Chapter 5
Pipelined Wormhole Routers

The single-cycle wormhole router performs all the tasks involved per input and per output serially. Each packet should first complete routing computation (RC) (in the cases that lookahead routing computation is not involved in the design of the router), then fight for gaining access to the output via switch allocation/arbitration (SA) and move to the appropriate output via the multiplexers of the crossbar (Switch Traversal – ST). Eventually, the packet will reach the next router, after leaving the output buffer and crossing the link (Link Traversal – LT). We assume that the input/output links of the router are independently flow controlled, following the credit-based flow control described in the previous chapters.

A block diagram of the single-cycle router is shown in Fig. 5.1. The output buffers of the router can be either simple pipeline registers or normal flow-controlled buffers. In the first case, the credit counter refers to the available buffer slots of the buffer at the input of the next router, while in the second case, the credit counter mirrors the available buffers of the local output buffer. In the rest of this chapter, we adopt the first design option and assume that the output of the router consists of a simple pipeline register for the signals in the forward (valid, data) and in the backward direction (credit update).

Depending on the system's characteristics the network on chip should be able to operate at low and at high clock frequencies. In the case of single-cycle routers the clock frequency of the NoC router is limited by the cumulative delay of all operations depicted in Fig. 5.1 plus the clocking overhead, which for register-based implementations (edge-triggered flip-flops) is the sum of the clock to data out delay and the register's setup time (Weste and Harris 2010).

Achieving higher clock frequencies requires the separation of the timing paths of the single-cycle implementation to multiple shorter ones in terms of delay, called pipeline stages. In this way, the delay seen between any two registers is decreased, which allows increasing the operating clock frequency. The separation involves the addition of pipeline registers between selected tasks that retime the transfer of information across stages to different cycles of operation. This inevitable retiming

G. Dimitrakopoulos et al., *Microarchitecture of Network-on-Chip Routers: A Designer's Perspective*, DOI 10.1007/978-1-4614-4301-8_5

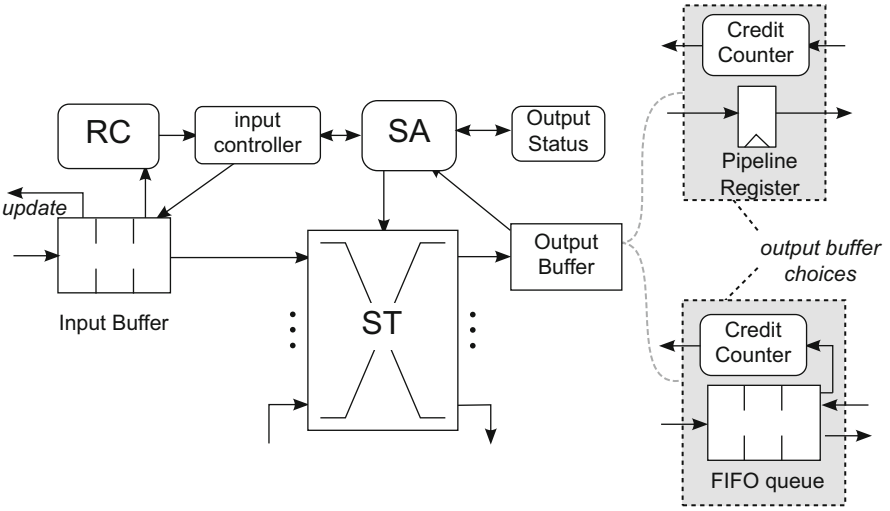

Fig. 5.1 An abstract organization of the single-cycle baseline wormhole router

complicates the operation of the router and introduces idle cycles, as seen by the flits of each packet, until other dependent operations of previous flits or packets are completed. The idle cycles imposed by such architectural dependencies are often called bubbles (empty pieces that flow through the pipeline without doing any actual work).

Pipelining is not only needed in high-speed configurations but it is also needed in energy constrained cases that the NoC should be able to sustain an acceptable operating frequency even under lowered voltage. Scaling the voltage of the circuit is a useful alternative for increasing energy efficiency and reducing power consumption especially in mobile devices that rely on a battery supply for their operation. However, lowering the voltage of the circuits increases significantly the delay of their constituent logic blocks that limits the maximum clock frequency of their operation. Pipelining retrieves back some of the lost MHz of clock frequency due to voltage scaling thus keeping a balance between energy efficiency and achievable performance.

In this chapter, we will describe in detail all the pipelined alternatives for wormhole routers, their implementation and their runtime characteristics. For the first time, the pipelined organization of routers is presented in a customizable manner where pipelining decisions are derived through two basic pipeline primitives: RC and SA pipeline. For each case, the cycle-by-cycle behavior will be analyzed and any microarchitectural implications that limit the router's throughput by necessitating the insertion of pipeline bubbles will be discussed and appropriate solutions will be derived.

5.1 Review of Single-Cycle Router Organization

Before diving into the design details of pipelined routers, we review in a higher level of abstraction the operations involved in a single-cycle router. The organization of a single-cycle router focusing on the request generation logic is shown in Fig. 5.2. Apart from the 3 main tasks or RC, SA, and ST, some secondary, low complexity, though critical tasks are involved. After the calculation of its destination output port through RC, a packet must generate a proper request (req) to SA, according to the current states of the input, as described by the *outLock* variable, and the destined output, as declared by the *outAvailable* flag.

At the output side, if a flit has won in SA and is about to be stored at the output buffer, it must consume a credit (Credit Consume – CC), that is, decrease the credit counter's value to reflect the current free slot availability of the output buffer (placed at the input of the next router). If the granted flit was a head or a tail flit, the output's *outAvailable* flag must be set accordingly through State Update (SU). Recall that when a head flit allocates output j, it sets $outAvailable[j] = 0$ in order to block requests from any other input to that output. Equivalently, the output is released ($outAvailable[j] = 1$) once the tail flit is granted, allowing head flits to fight again for that port in the next cycle. Once granted, the flit is dequeued from the input buffer (DQ). At the same time, the input buffer informs the previous router that a buffer slot is emptied, using the credit update mechanism.

The router's organization of Fig. 5.2 reveals two distinct and converging paths. The *control path* starts with RC and the request generation module (req), continues with the SA stage, and ends up in the select signals of the crossbar (ST) as well as the grant signals delivered at each input buffer (dequeue). On the contrary, the

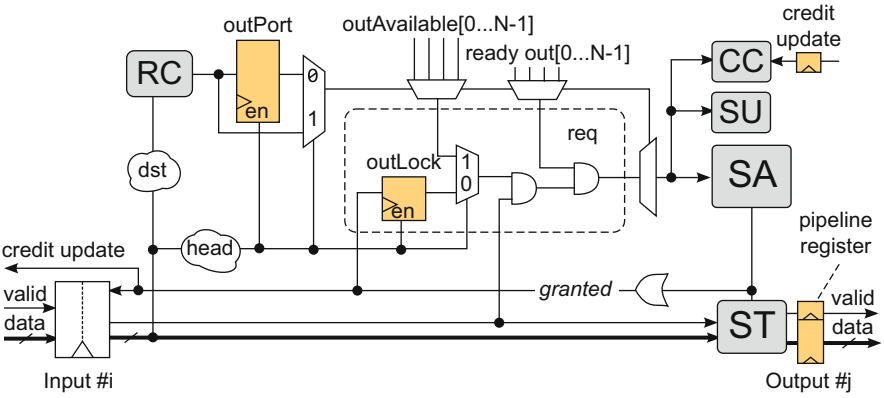

Fig. 5.2 The organization of a single-cycle router and the details of the input request generation logic. Output Credit consume (*CC*) and State Update (*SU*) operations can occur in parallel to SA (per-output arbitration) by checking the existence of at least one active request to the corresponding output

data path involves only multiplexing operations inside the input buffer, driven by the FIFO's pointers and the per-output multiplexers of the crossbar that end up at the output pipeline register.

5.1.1 Credit Consume and State Update

As explained in Chap. 3, updating the *outAvailable* flag and consuming the necessary credits should be triggered once a flit traverses the output multiplexer, and is about to be written to the output pipeline register. Although this might seem a safe and reasonable choice – and it is indeed for a single cycle router – it introduces some non negligible delay overhead. A closer elaboration reveals that in this organization, SU and CC must occur only after SA is completed and, most importantly, after a multiplexing of all inputs is performed (for example to check whether a head or tail flit exists at the granted input). This multiplexing is non-trivial in terms of delay and, in real-life applications, it may limit the benefits of pipelining.

The problem can be completely eliminated by making an important observation: both CC and SU can be executed without the need of knowing specifically *which* input allocates the output port or consumes an output credit. Simply knowing that *some input* wins in arbitration or sends a flit forward, suffices. Therefore, since the SA result is not required, those operations can occur in parallel to SA. For SU, this translates to checking whether any request from a head or a tail flit exists, to lower or raise the *outAvailable* flag, respectively. CC decrements the output credit counter if the corresponding output receives at least one request. Notice that once the output's *outAvailable* flag is lowered, request generation forbids any requests to that output, unless they originate from the winner input. The *outAvailable* flag is raised again once a tail flit makes a request (receiving a grant is guaranteed) and its new updated value will be visible to the rest inputs in the next clock cycle.

5.1.2 Example of Packet Flow in the Single-Cycle Router

The operations executed in the single-cycle wormhole router of Fig. 5.2 can be seen in Fig. 5.3. The execution diagram refers to the behavior of a single input that receives a consecutive traffic of incoming packets consisting of 3 flits (one head, one body and one tail flit). This kind of traffic is selected since it reveals easily any latency/throughput-related inefficiencies of pipelined organizations that will be presented in later sections. In parallel, the rest inputs follow a similar execution assuming that their requests and data move to a different output. A certain output can host the packet (on a flit-by-flit basis) of only one input at a time.

In cycle 0, a head flit is written at the input buffer (Buffer Write – BW), after crossing the link (Link Traversal – LT). The flit immediately appears at the frontmost position of the input buffer in cycle 1, and is able to execute all necessary operations

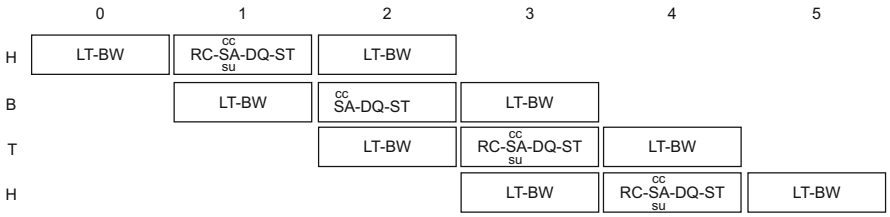

Fig. 5.3 The execution of necessary operations on a single-cycle wormhole router

within a single cycle: (a) the flit's destination field feeds the RC and the *outPort* bypass path is used to feed the request generation logic; (b) supposing that the output is available, the flit performs SA, while in parallel it consumes a credit (CC) and updates (SU) the *outAvailable* flag; finally, (c) the grant produced by SA is used to dequeue (DQ) the flit from the input buffer, in order to traverse the crossbar (ST).

As the head flit moves forward to the output pipeline register, a body flit is written at the input buffer. The output buffer has enough credits available, thus allowing the newly arrived body flit to use the stored *outPort* value and generate a request to SA. Being the only active request (the requests of all other inputs are nullified, since $outAvailable = 0$), the body flit is granted to move forward, after consuming a credit. At the same cycle, the head flit is moving to the next router. In cycle 3, the tail flit follows the same procedure, performing SU as well, in order to release the allocated port ($outAvailable = 1$), while the next packet's head flit arrives. In cycle 4, all previously allocated resources are already free and the following packet is able to generate a request and participate in arbitration, whatever its destined output port might be. Observing the rate of incoming and outgoing flits of this input, one would notice that a flit only requires a single cycle to exit the router, and no extra cycles are added in between packets. The only conditions under which a flit may be stalled is (a) if all the output buffer's slots are full, or (b) a head flit loses in arbitration (in this case the output port is still utilized, but by a different input).

In the rest of this chapter, we will modify the baseline single-cycle organization reviewed in this section in a step-by-step manner to derive pipelined implementations that isolate the RC and the SA stages from the rest with the goal to increase the router's clock frequency. Then the primitive RC and SA pipelined organizations will be combined in a plug-and-play manner to derive three-stage pipelined organizations that lead to even higher clock frequencies.

5.2 The Routing Computation Pipeline Stage

Pipelining RC from SA and ST is the simplest form of pipelining that can be performed to the router. RC is the first operation of the control path of the router. Thus, the RC pipelined organization will include only a pipeline register at the control path of the router, resulting to the organization shown in Fig. 5.4.

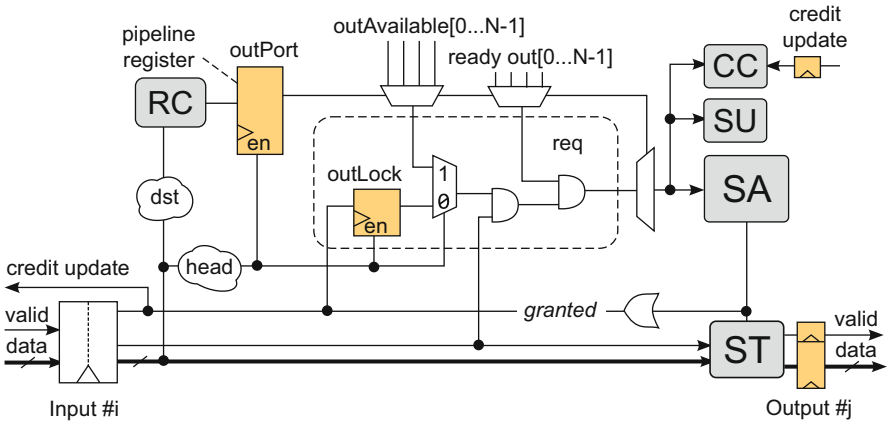

Fig. 5.4 The organization of the router that pipelines the RC stage of the control path from SA and ST. The *outPort* state variable that holds the output port requests of each packet acts as the pipeline register

The only difference compared to the un-pipelined version lies around the *outPort* register of each input. In the single-cycle organization this register is bypassed via a multiplexer when a head flit appears at the frontmost position of the input buffer. This bypass is necessary for allowing the head flit to generate the requests to the SA in the same cycle. In the RC pipelined organization this multiplexer is removed allowing the *outPort* register to play the role of the pipeline register in the control path that separates RC from SA and ST. In both cases, the *outPort* register is set (storing the output port request of the corresponding packet), when the head flit of the packet appears at the frontmost position of the input buffer ($isHead(Q) = true$), and it resets when the tail flit of the packet is dequeued from the input buffer ($isTail(Q) = true$ and *granted*).

Using this organization the critical path of the router is reduced by the delay of the RC unit and in most tested configurations starts from the *outPort* register, passes through the request generation logic and arbitration and ends up at the output pipeline register. Please note that, since now the delay of the control path is shortened, depending on the exact delay profile of the pipelined control path, the critical path of the router may migrate from the control path and move to the data path of the design.

The cycle-by-cycle execution of the RC control pipelined version of the router is shown in Fig. 5.5. In cycle 0 the head flit of a packet arrives at an input and is stored in the input buffer (BW). Then in cycle 1 the head flit performs RC and stores the output port requests of its packet to the *outPort* pipeline register. In parallel a body flit arrives at the same input. During cycle 2 the head flit performs SA and, assuming that it is successful, it dequeues (DQ) itself from the input buffer and moves to the crossbar that implements ST. In parallel to SA, CC and SU operations take place, consuming a credit and lowering the *outAvailable* flag. The body flit that arrived at

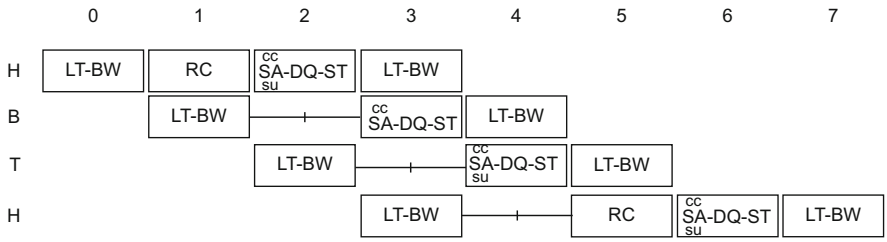

Fig. 5.5 The operation of a router that pipelines RC from SA and ST

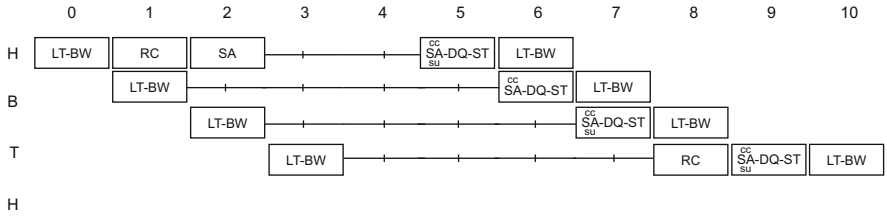

Fig. 5.6 The operation of a pipelined router that executes RC in the first pipeline stage and SA-ST in the second and exhibits idle cycles due to the unsuccessful switch allocation (*SA*) of the first packet

cycle 1 waits in the buffer, while the tail flit of the same packet arrives in the same cycle and occupies the next buffer position behind the body flit.

If the SA operation was not successful, either because another input was granted access to the same output port, or because the output port didn't have enough credits, the head flit would continue trying. The only effect of such unsuccessful trials would be to shift the execution example of this input, depicted in Fig. 5.5 some cycles to the right, as shown in Fig. 5.6. The SU and CC operations would still take place, but only for the winner input port that does not experience the idle cycles seen by the input that lost SA and depicted in Fig. 5.6.

In cycle 3, of Fig. 5.5, the head flit moves to the link (LT) and approaches next router. Now, the body flit is at the frontmost position of the input buffer and performs SA and CC. The output that was given to the head flit in the previous cycle is now unavailable for all inputs except this one. Therefore, the request generated by the body flit would be satisfied for sure since it will be the only active one. Consequently, in cycle 3 the body flit would be dequeued and switched to the selected output. The tail flit remains idle waiting its turn to arrive to the frontmost position of the input buffer.

Assuming that the input buffer is allocated non-atomically, meaning that flits from different packets can be present at the same time on the same input buffer, the head flit of a second packet arrives in cycle 3 too. When atomic buffer allocation is employed, no new flit would arrive at this input, until the buffer is completely empty from the flits of the previous packet. Implementing atomic buffer allocation in terms of flow control policy is discussed in Sect. 3.1.2.

In cycle 4, the body flit of the first packet is on the link, and the tail flit of the same packet completes SA, CC and ST, releasing in parallel the output port (SU). Ideally, the head flit of the second packet could have completed RC. However, in the examined configuration of the RC pipeline, overlapping of RC with the tail's SA operation is not allowed. The reason for this limitation is that the RC unit is fed with the destination field of the head flit only when the head flit is at the frontmost position of the input buffer. In cycle 4 the frontmost position is occupied by the tail flit that will be dequeued at the end of the cycle and move to the output of the router. Therefore, the head flit of the second packet can feed the RC unit with the necessary info not earlier than cycle 5, e.g., when the tail flit is already on the link.

This bubble in the RC control pipeline will appear in any case that two different packets arrive at the same input back-to-back in consecutive cycles and it occurs only after the end of the first packet. Packets from different inputs are not affected. For example, when the tail flit of a packet from input i is leaving from output k, it does not impose any idle cycle to a packet from input j that allocates output k in the next cycle.

Therefore, RC for the head flit of the second packet is completed in cycle 5 and the flow of flits in the pipeline continue the same way as before in the following cycles.

5.2.1 Idle-Cycle Free Operation of the RC Pipeline Stage

The bubble appearing in the RC control pipeline is an inherent problem of the organization of the router that does not allow the control information carried over by flits, not in the frontmost position of the buffer, to initiate the execution of a task, such as RC, in parallel to the tasks executed for the flit that occupies the frontmost position of the input buffer. In the RC control pipeline the information of both the frontmost and the second frontmost position would have been required to eliminate idle cycles across consecutive packets.

This requirement can be satisfied by adding in parallel to the control pipeline register (*outPort*) a data pipeline register that acts as a 1-slot pipelined elastic buffer (EB) (see Sect. 2.1.3 for details). RC would be initiated by the frontmost position of the normal input buffer, while all the rest tasks such as request generation, SA and ST would start from the intermediate pipelined EB, thus allowing the parallel execution of RC for the new packet and SA-ST for the tail of the old packet. This organization is shown in Fig. 5.7.

In this configuration, when a head flit appears in the frontmost position of the input buffer, it executes RC and updates the *outPort* register, while moving in parallel to the intermediate EB. The EB will only write incoming data when empty, or when it is about to become empty in the same cycle (dequeued). On the contrary, when a tail flit moves to the intermediate EB it will reset the *outPort* register when it is ready to leave the EB (it received a grant). If in the same cycle, the head flit of a new packet tries to set the *outPort* register and move to the EB and the tail flit of

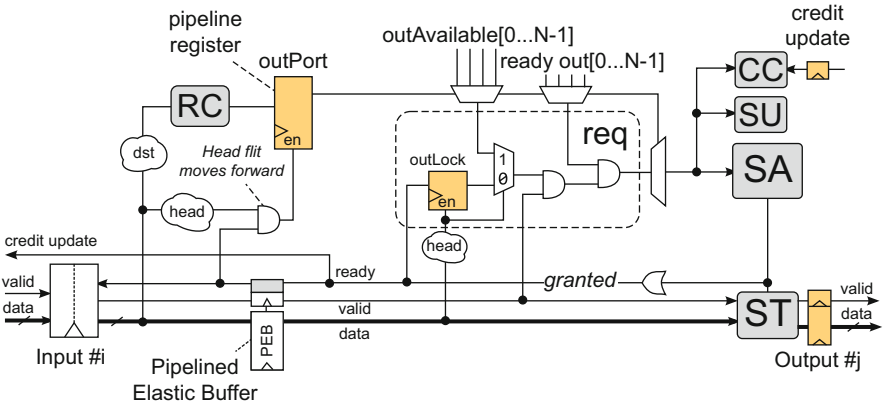

Fig. 5.7 The organization of a router that pipelines RC from SA and ST using a pipeline register both it the control and in the data path of the router. The pipeline register of the datapath acts as an 1-slot EB that holds the flits ready for request generation and SA

the old packet that lies in the EB tries to reset the *outPort* register and leave the EB, priority is given to the head flit of the new packet. Notice that the *outPort* register is protected, so that a previous packet's saved request is not overwritten by the head flit of the next packet. If a tail is stalled at the intermediate EB (e.g. if all of the output buffer's slots where occupied), setting a new value to the *outPort* register by a head flit at the input buffer would not be allowed.

The intermediate EB acts as an extension of the input FIFO buffer. It receives the grants produced by SA instead of the main input buffer and guides the generation of the credit update signals send to the upstream connections. A credit update is sent backwards, not when a flit leaves the input buffer, but instead, when it leaves the intermediate 1-slot EB. Keep in mind that now, the input buffer can actually hold $b + 1$ flits (b in the main input buffer plus 1 in the intermediate EB), implying that the credit counter responsible for counting the free slots of this input (e.g. at the output of the adjacent router), must have a maximum value of $b + 1$, instead of b.

An example of the operation of a router that includes both control and data pipeline segments is depicted in Fig. 5.8 using the same flow of flits as in the previous example, where pipelining of the RC stage was done only in the control path. The first true difference between the two cases appears in cycle 1. The head flit once it completes RC it is dequeued from the input buffer and moves to the pipelined EB. The head flit will wait there until it wins in SA and moves to its selected output. As long as the head flit is stalled in the pipelined EB stage all the rest flits are stalled inside the input buffer.

In cycle 2 two operations occur in parallel. The first one involves the operations of the head flit that participates in SA, wins a grant, consumes a credit, updates the *outAvailable* flag, and gets dequeued from the intermediate EB moving towards its destined output port via ST. The second one involves the operation of the body flit that moves from the input buffer to the intermediate 1-slot EB. The intermediate EB

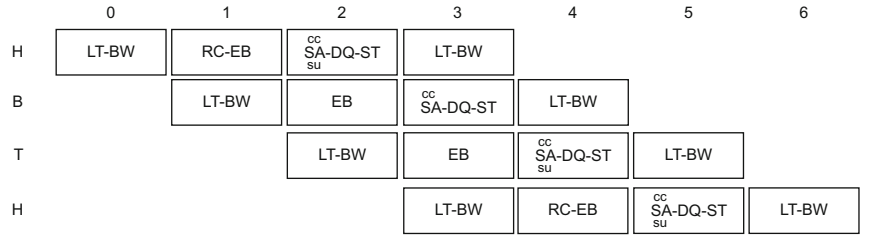

Fig. 5.8 The operation of the pipelined router that includes an RC pipeline register both in the control and the datapath. The pipeline register of the datapaths acts as a 1-slot elastic buffer. The EB task of the diagram implies writing to this intermediate EB

in this cycle enqueues the body flit while it dequeues the head flit that was stored in the previous cycle. The same happens also in cycle 3 for the body and the tail flit. At the end of cycle 3, the tail flit of the first packet has left the input buffer and moved to the intermediate EB. Thus, at the beginning of cycle 4, the frontmost flit of the input buffer is the head flit of the new packet. Since the tail flit that occupies the intermediate EB is leaving, the head flit will take its position at the end of cycle 4, completing RC in parallel. Since the tail flit updated the *outAvailable* flag in cycle 4, the head flit currently in the EB can make a request in cycle 5 even for the same output port. In this way, the flits of the two consecutive packets arriving at the same input pass through the router un-interrupted without experiencing any idle cycles, even if they are heading to the same output port.

5.3 The Switch Allocation Pipeline Stage

The second interesting form of pipelining for the router is the separation of SA from ST, which can be combined with unpipelined or pipelined RC organizations (either in the control or both the control and datapath) and give efficient pipelined architectures. The per-output arbiters that implement the SA stage receive the requests from all inputs and produce a valid input-output match. In contrast to the static and local nature of the RC operation, SA is a function of several dynamic parameters that create dependencies across inputs and make the design of SA pipeline stage challenging. In the following sections, three different approaches are presented, that reveal those dependencies and offer realistic solutions.

5.3.1 Elementary Organization

Pipelining the router at the end of the SA stage means that the grant signals produced by the arbiters are first registered and then, in the next cycle, distributed

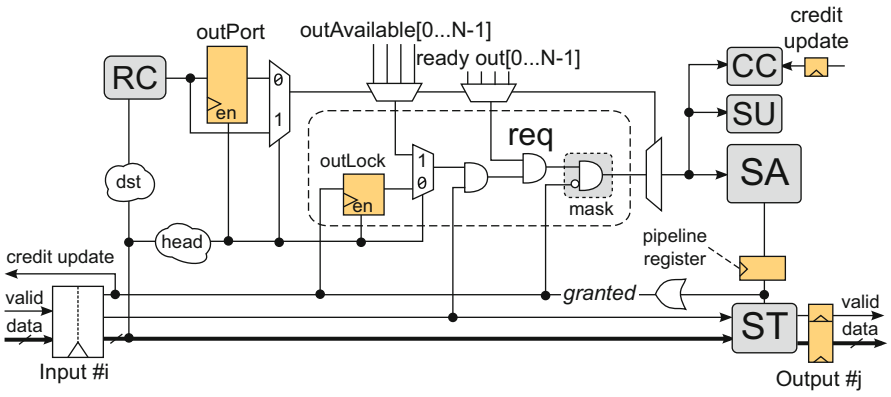

Fig. 5.9 The organization of the router that separates in different pipeline stages SA from ST using a pipeline register at the output of the SA unit. New requests are generated only when the grants of the previous requests have been first delivered thus allowing the generation of a new request per input once every two cycles

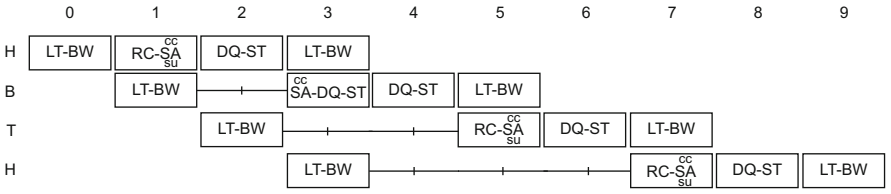

Fig. 5.10 The operation of the pipelined router that delays the delivery of the grants to the input buffer and the crossbar

to the inputs and the output multiplexers of the router, as shown in Fig. 5.9. This organization corresponds to a purely control pipelined organization and faces an inherent problem. Every flit that requests a certain output at cycle t_0, it will receive a grant at cycle $t_0 + 1$, and should decide how to react in the next cycle. The first obvious choice is to wait, meaning that a new flit will depart from its input every two cycles; one cycle spent for the request and one for accepting the grant that appears one cycle later. While waiting for the grants to come, an input should not send a new request. To achieve this bevahior the requests produced in the current cycle should be masked using the grants of the previous cycle (denoted as 'mask' in Fig. 5.9). If the input was previously granted, then the current requests are nullified, thus causing the arbiter to produce an empty grant vector for this input in the next cycle. Once the masking logic understands the existence of an empty grant vector it allows the requests to pass to the arbiter. In this way, a new grant vector is produced every two cycles thus adding a bubble between any two flits that reach the SA stage.

The behavior of the pipelined router that follows the organization of Fig. 5.9, is depicted in Fig. 5.10. In this example, the RC stage is considered to be unpipelined. The head flit that arrives in cycle 0, performs RC and SA during cycle 1 and in

parallel consumes the necessary credit and updates the state of its destined output. The grants return in cycle 2 that causes the head flit to get dequeued from the input buffer (DQ) and move to the crossbar. Since the head flit will leave at the end of cycle 2, it should not produce a new request in this cycle. This is handled by the request masking logic of Fig. 5.9 that cuts any requests generated in cycle 2 and in effect cuts any grants delivered in cycle 3. The rest inputs, although they have not received their grants, they don't produce also any requests in cycle 2 for the same output; their requests are blocked since the *outAvailable* flag of their destined output has been lowered in cycle 1 during SU.

The body flit arrives at the frontmost position of the input buffer at the end of cycle 2. In the beginning of cycle 3, it generates its own set of requests that will be delivered in cycle 4. The requests of the body flit survive the request masking logic since in cycle 3 the input does not receive any grants. Since the output is locked by the grant given to the head flit of the packet, the body flit will receive a grant for sure. The same operation continues in the next cycles where an empty cycle is added for each flit after SA.

5.3.2 Alternative Organization of the SA Pipeline Stage

The delay between request generation and grant delivery leads to an idle cycle between every pair of flits of the packet. By observing that the body and tail flits do not need to generate any request and they can move directly to ST by inheriting the grants produced by the head flit of the same packet, we can remove all the idle cycles experienced by the non-head flits. The body and tail flits before moving to ST should just check the availability of buffer slots at the output buffer.

In this configuration the grants produced by the SA should be kept constant for all packet's duration. According to the organization depicted in Fig. 5.11, the pipeline register that was used to register the SA grants per output, is now replaced by a register that is updated under the same conditions used to update the *outAvailable* flag: grants are stored or erased when at least a request by a head or a tail flit is made, respectively. Now, once a head flit wins arbitration, grants persist until the tail flit resets them. Although the head flit should wait for the grants to return, the body and tail flits are dequeued once they have an active request (a request is always qualified by the status of the credit counters). This condition is implemented by the multiplexer in the backward direction. Please notice that the request mask used in Fig. 5.9 is removed and the initial request generation logic is restored at the input side.

The stored grants always drive the select lines of the output multiplexer transferring to the output register data and their valid signals. However, we should guarantee that the valid signal seen at the output buffer corresponds always to a legal flit; a flit is legal if it is both valid and the output buffer has enough credits to accept it. Delivering to the output multiplexer the valid signal of the input buffer as done in previous cases of Figs. 5.2, 5.4, 5.7 and 5.9 is not enough in this configuration, since

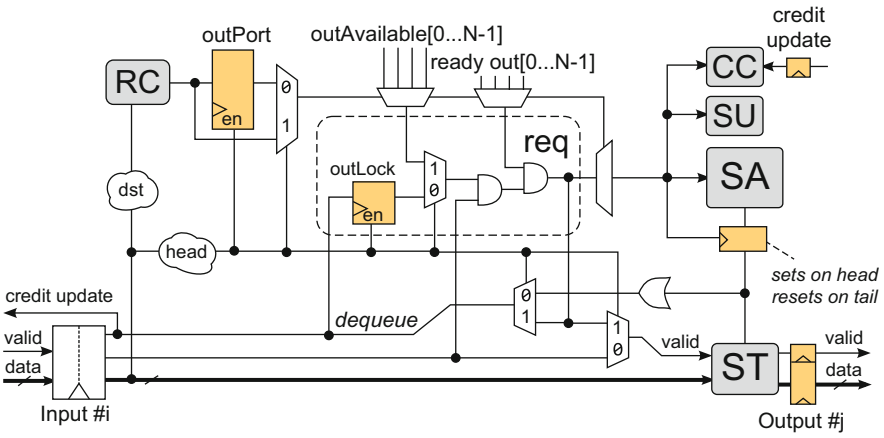

Fig. 5.11 The organization of the pipelined router that pipelines the grants of the SA unit and keeps them for direct use by the body and tail flits of the same packet, thus breaking the dependency across the request and the delayed arrival of the corresponding grants

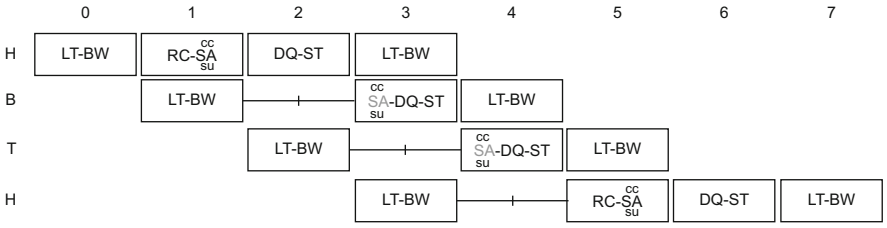

Fig. 5.12 The operation of the alternative SA pipelined router, where body and tail flits move directly to ST once they have the necessary credits by inheriting the grants produced by the head flit of the same packet and stored in the corresponding pipeline register

the output buffer will receive via the output multiplexer body/tail flits that have not been qualified for the necessary credits. To resolve this issue a mutiplexer is added in the forward direction, as shown in Fig. 5.11, that selects the qualified valid signal produced by the request generation logic in the case of body/tail flits instead of the normal valid signal.

The cycle-by-cycle operation of this alternative SA pipelined organization is presented in Fig. 5.12. A head flit is written to the input buffer in cycle 0 and issues a request to SA in cycle 1 after having completed RC in the same cycle. The grants from SA return in cycle 2 and saved for later use by the body and tail flits of the same packet. In cycle 2, the head flit accepts the grant and gets dequeued moving to its destined output via the output multiplexer (ST). In cycle 3, the body flit of the same packet arrives at the frontmost position of the input buffer. Without sending any request to the SA (requests are actually generated only for the purpose of ready qualification) once the body flit has at least one credit for its selected output it gets dequeued and moves to ST after consuming one credit. The ST stage will switch

correctly this body flit driven by the stored grant signals at the corresponding output. The tail flit in cycle 4 repeats the same procedure and moves to ST without waiting for any SA grants. The head flit of the new incoming packet reaches the frontmost position of the input buffer in cycle 5 initiating a new round of RC, SA and ST operations, similar to the head flit of the previous packet.

This stored grants approach turns the previously presented "elementary" SA pipeline an obsolete choice. It reduces bubbles significantly with only minimal delay overhead to the router's control path. It also looks as if it simplifies the allocation procedure. However, in essence, it simply adds extra state registers to the arbiter's path and a multiplexer at both control and data paths. Therefore, this approach is avoided in single cycle version, and the original request generation logic is preferred. For the same reason, the stored grants, will not be used at the next pipeline SA configuration that uses a pipeline register both in the control and in the datapath although it could have been a possible choice. The stored-grants approach for the body and tail flits is a useful pipeline alternative when SA is separated from ST solely in the control path.

5.3.3 Idle-Cycle Free Operation of the SA Pipeline Stage

The dependencies arising from delaying the delivery of the grants of SA to the crossbar and to the inputs of the routers can be alternatively resolved by adding an extra input pipeline register to the data path. The added data pipeline register, shown in Fig. 5.13, does not have to be flow-controlled since no flit will ever stall in this position. This data pipeline register is just used to align the arrival of the registered grant signals with the arrival of the corresponding flit to the input of the crossbar.

Since the grant signals should be always aligned to the corresponding data, the delivery of the grant signals to the inputs should move before the grant pipeline register (as done in Fig. 5.13). This is needed since the dequeued data will reach the input of the crossbar one cycle later; they will spend one cycle passing the data pipeline register. This extra cycle also requires an extra buffer slot at the output buffer (at the input of the next router) for full throughput operation, since forward latency L_f is increased by 1.

The pipeline flow diagram that corresponds to the pipelined organization of Fig. 5.13 is shown in Fig. 5.14. In this case, the head flit in cycle 1 performs RC and SA and after accepting the grants in the same cycle it is dequeued and moves to the data pipeline register after having consumed the necessary credit. In cycle 2, the head flits leaves the data pipeline register and moves to the selected output using the grants produced by the corresponding output arbiter in the previous cycle. In the same cycle, the following body flit that arrived in cycle 1 and is placed now in the frontmost position of the input buffer, can perform SA and once granted it can move also to the data pipeline register consuming in parallel the necessary downstream credit. The same holds for the following tail flit that can perform all needed operations without experiencing any idle cycles. In cycle 4, the full overlap

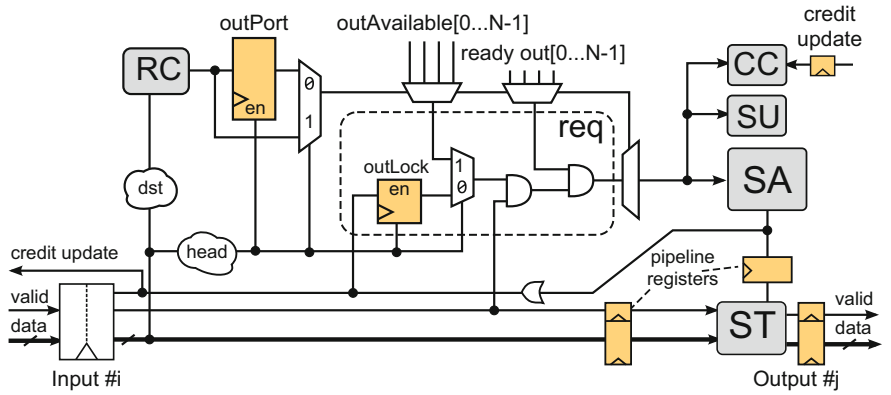

Fig. 5.13 The organization of the router that pipelines RC-SA from ST using a pipeline register both in the control path, that registers the grants of SA, and in the data path, that registers the data arriving at the input of the crossbar

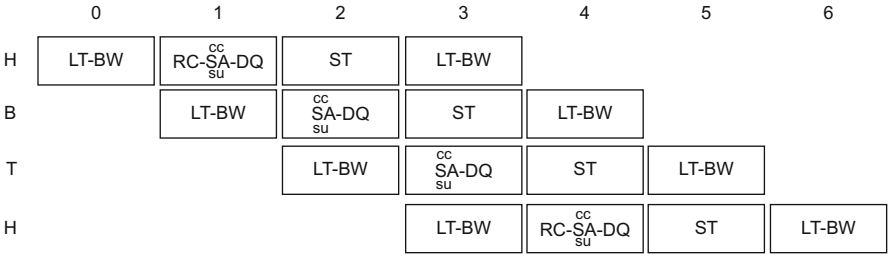

Fig. 5.14 The operation of a pipelined router that executes RC-SA in the first pipeline stage and ST in the second, and uses pipeline registers both in the control path and in the datapath of the router

allowed between the operations per pipeline stage is fully revealed. The body flit is on the link, the tail flit performs ST, while the head flit of the next incoming packet from the same input (it could be also from a different input), performs RC and SA and at the end of the cycle moves to the data pipeline register after having consumed an available credit.

The main contribution of the SA pipeline stage is the isolation of the crossbar from the rest of the control and data path logic (the remaining data path logic consists mostly of the data multiplexing inside the input buffers). Depending on the radix of the router, its data width, and other placement options the crossbar may have a significant contribution to the final delay. In low-radix cases, though, the delay of the crossbar is not the critical factor that determines the speed of the router.

The idle cycles that appear in the straightforward SA pipelined organization can be removed by either relying to a re-organization of the input request unit that forwards the body and tail flits directly to ST, provided that they have credits available, or by adding another data pipeline register placed at the inputs of the crossbar that would align the arrival of grants and the corresponding data.

5.4 Pipelined Routers with RC and SA Pipeline Stages

The RC and SA pipeline stages can be combined in pairs and derive 3-stage pipelined implementations that schedule RC, SA, and ST in different clock cycles. Some of the possible alternatives do not offer any real benefit to the design of the router since they preserve certain architectural dependencies that introduce significant number of bubbles in the pipeline. Instead, in this section, we will describe two of the most useful alternatives that isolate the internal timing paths of the router and minimize at the same time the idle cycles in the operation of the router.

5.4.1 Pipelining the Router Only in the Control Path

One of the two organizations discussed in this chapter includes an RC pipeline and a SA pipeline stage, where pipelining occurs, in both cases, only in the router's control path. For the SA-control pipeline we assume the organization presented in Sect. 5.3.2. In this case, the output of the SA is driven to a pipeline register that returns the grants to the inputs, but also stores them for direct use by the body and tail flits of the same packet that move directly to SA once they have available credits. The complete organization of the 3-stage control-path pipelined router is shown in Fig. 5.15, while an example of its operation is shown in Fig. 5.16.

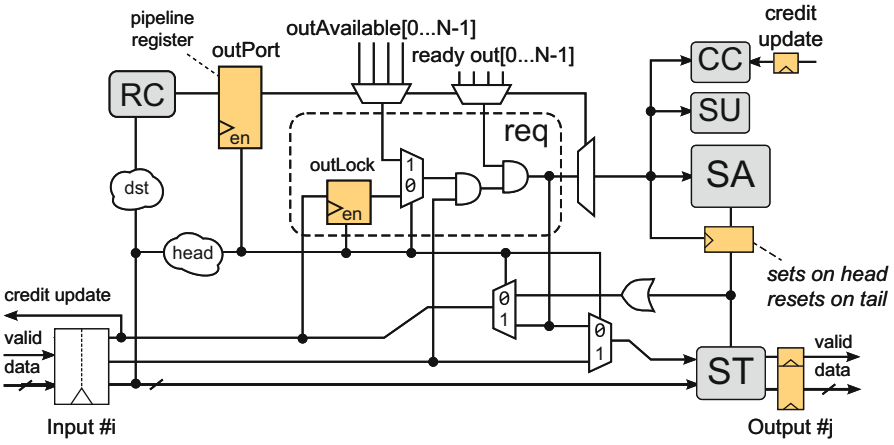

Fig. 5.15 The organization of a 3-stage pipelined router that executes RC in the first pipeline stage, SA in the second and ST in the last stage. Pipeline registers have been added only in the control path of the router

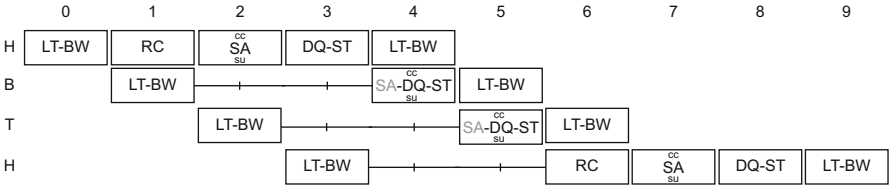

Fig. 5.16 The flow of flits in a 3-stage pipelined router where pipelining is employed only in the control path of the RC and SA stages

The organization of the router is just a composition of the RC control-only pipeline stage, shown in Fig. 5.4, where the *outPort* register acts as the pipeline register and SA control pipeline stage of Fig. 5.11, where the grants are stored at the output of the SA and re-used by the body and tail flits of the packet.

First of all, since only one flit can deliver its control information per input (only one in the frontmost position of the input buffer), there should be at least one idle cycle between consecutive packets, as depicted in Fig. 5.16. This is revealed in cycle 5 where the head flit of the second packet waits unnecessarily for the tail flit to leave the input buffer and complete RC in cycle 6, although it actually arrived at the input of the router in cycle 3. Besides that, the rest flits experience an un-interrupted flow. For example the body and tail flits re-use in cycles 4 and 5 the grants returned to their input in cycle 3 after the request generated in cycle 2 by the head flit of the same packet.

5.4.2 *Pipelining the Router in the Control and the Datapath*

The idle cycles can be removed by employing combined control and data pipelines for both the RC and the SA stage. The organization of the router that employs this pipelined configuration is shown in Fig. 5.17. By observing closely the block diagram of Fig. 5.17, we can see that the organization presented is derived by stitching together the RC and SA combined pipelines presented in Figs. 5.7 and 5.13, respectively.

The pipelined operation experienced by the flits of a certain input that belong to two consecutive packets is shown in Fig. 5.18. The first flit (head flit) that arrives in cycle 0 will leave the router four cycles later. In cycle 1 it completes routing computation and at the end of the cycle it moves to the pipelined EB of the RC stage. This movement required the dequeue of the head flit from the input buffer. In cycle 2, the requests stored in the *outPort* pipeline register are sent to SA that returns the corresponding grants in the same cycle. These grants are used to dequeue the head flit from the intermediate EB and place it to the pipeline register at the input of the crossbar. In the meantime, the body flit of the same packet has arrived and moved to the intermediate EB. During cycle 3 the head flit just moves through the crossbar

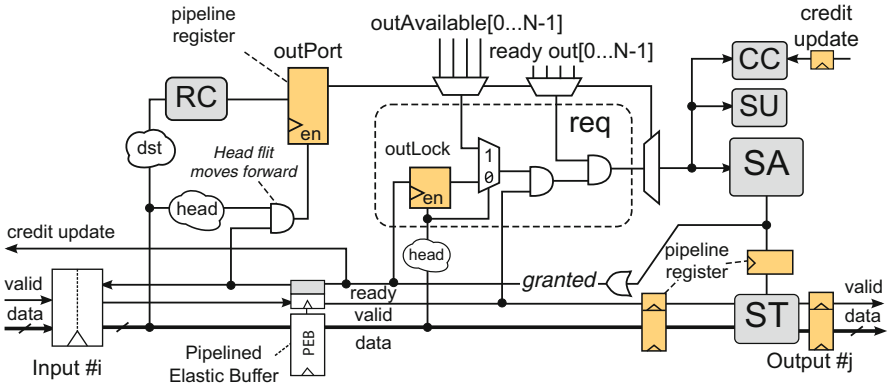

Fig. 5.17 The organization of a 3-stage pipelined router derived by the composition of RC and SA pipeline stages that includes pipeline registers both in the control and the datapath of the router

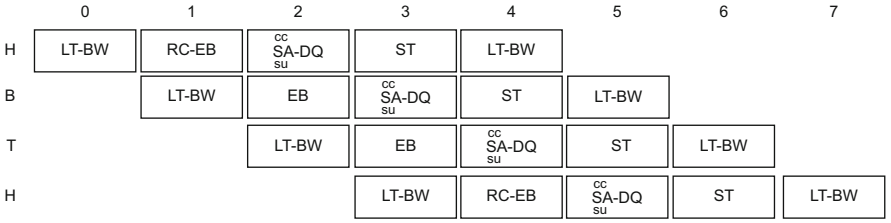

Fig. 5.18 The operation of a 3-stage pipelined router that exhibits no idle cycles by employing pipeline registers at the end of RC and SA both in the control and data path of the router

and reaches its selected output, while the body flit that follows takes its place in the data register of the SA stage after receiving a grant and consuming its credit. The position previously held by the body flit is now occupied by the incoming tail flit that has been dequeued from the input buffer and moved to the EB of the RC stage. Cycle 4 evolves in a similar manner. Due to the two data pipeline registers, the frontmost position of the input buffer is free for the head flit of the next packet that arrived at the same input. This characteristic allows the head flit to complete RC and move to the intermediate EB accordingly.

In this configuration, idle cycles are neither experienced by the flits of the same packet nor by the flits across different packets, and all flits continue moving to their selected output at full throughput. Any idle cycles experienced would be a result of output contention due to the characteristics of the traffic pattern and the routing algorithm, and not a result of the internal microarchitecture of the router.

Between the two extremes 3-stage pipelined solutions of using pipelining only in the control path or both in the control and the data path, there are other two intermediate configurations. The first one employs control pipeline at the RC stage and combined pipelined at the SA stage, while the second does the opposite; it employs combined pipelining at the RC stage and control-path-only pipeline in the

SA stage. The behavior of each solution in terms of throughput can be easily derived by the behavior experienced by each sub-component analysis in Sects. 5.2 and 5.3. Whenever the RC stage is pipelined only in the control path, one idle cycle should be added between the end of a packet (tail flit) and the start of the next one (head flit) that arrives at the same input in consecutive cycles, assuming that the input buffer is allocated not atomically. On the contrary, when the SA stage is pipelined only in the control path then an idle cycle is inserted for the head flit; the head flit is obliged to wait in the input buffer for one cycle, until the grant from the SA arrives.

Depending on the exact delay profile of the modules that participate in the design of a router, such as routing computation, request masking and arbitration, grant handling and dequeue operations, as well as credit consume and crossbar traversal, the presented pipelined solutions may lead to different designs in the energy-delay space. In any case, the selection of the appropriate pipeline organization is purely application-specific and needs scenario-specific design space exploration. In this chapter, our goal was to present the major design alternatives in a customizable manner, e.g., every design can be derived by combining the two primitive pipelined organizations for the RC and SA stage that lead to reasonable configurations. Other ad-hoc solutions that eliminate the idle cycles of the control pipeline without the need for data pipeline stages may be possible after certain "architectural" tricks, but their design remains out of the scope of this book.

5.5 Take-Away Points

The main tasks of a wormhole router includes RC, SA and ST. Executing the tasks of the router, in an overlapped manner, in different pipeline stages can be derived by following a compositional approach, where the primitive pipeline stages, are stitched together to form many meaningful pipelined configurations. Pipeline registers can be added either in the control or in the datapath of router, leading to different tradeoffs in terms of the achieved clock frequency and the idle cycles that appear in the flow of flits inside the router's pipeline.