Hindawi Publishing Corporation
International Journal of Reconfigurable Computing
Volume 2015, Article ID 570836, 13 pages
http://dx.doi.org/10.1155/2015/570836

Research Article

Low Latency Network-on-Chip Router Microarchitecture Using Request Masking Technique

Alireza Monemi,[1] Chia Yee Ooi,[2] and Muhammad Nadzir Marsono[1]

[1]*Faculty of Electrical Engineering, Universiti Teknologi Malaysia, 81310 Johor Bahru, Malaysia*
[2]*Malaysia-Japan International Institute of Technology (MJIIT), Universiti Teknologi Malaysia, 54100 Kuala Lumpur, Malaysia*

Correspondence should be addressed to Muhammad Nadzir Marsono; nadzir@fke.utm.my

Received 30 June 2014; Revised 9 February 2015; Accepted 20 February 2015

Academic Editor: Miriam Leeser

Network-on-Chip (NoC) is fast emerging as an on-chip communication alternative for many-core System-on-Chips (SoCs). However, designing a high performance low latency NoC with low area overhead has remained a challenge. In this paper, we present a two-clock-cycle latency NoC microarchitecture. An efficient request masking technique is proposed to combine virtual channel (VC) allocation with switch allocation nonspeculatively. Our proposed NoC architecture is optimized in terms of area overhead, operating frequency, and quality-of-service (QoS). We evaluate our NoC against CONNECT, an open source low latency NoC design targeted for field-programmable gate array (FPGA). The experimental results on several FPGA devices show that our NoC router outperforms CONNECT with 50% reduction of logic cells (LCs) utilization, while it works with 100% and 35%~20% higher operating frequency compared to the one- and two-clock-cycle latency CONNECT NoC routers, respectively. Moreover, the proposed NoC router achieves 2.3 times better performance compared to CONNECT.

1. Introduction

The decreasing semiconductor transistor dimensions make it possible to integrate more Intellectual Property (IP) blocks in a single System-on-Chip (SoC). However, this poses new problems in inter-IP connectivity. Conventional shared bus interconnection lacks flexibility and scalability in dealing with a large number of IPs in a single chip. To alleviate these limitations, network-on-chip (NoC) has been introduced [1]. NoC separates the computation from the communication parts, while providing a scalable and flexible modular design.

In order to reduce on-chip memory usage, wormhole flow control algorithm is widely applied in NoC routers [2–7]. A wormhole router divides a packet into several smaller flow control digits (flits) and allows flits to be buffered in a flit-serial fashion order through several routers along the path. Moreover, applying several virtual-channels (VCs) on a single physical channel improves the overall NoC performance. VCs prepare multiple escape channels for active packets in the case of head-of-line (HoL) blocking. These escape channels result in higher throughput and also allows deadlock avoidance when the network traffic is high. However, adding these features leads to more complex router microarchitecture that requires more processing stages to deliver a packet from an input port to an output port. The increase in the number of router stages as well as the multiple number of hops that exist between a source and a destination IPs has resulted in higher communication latency compared to the conventional shared bus.

In order to reduce the communication latency while maintaining good throughput, a router needs to perform several stages such as route computation, VC allocation, and switch allocation in parallel. However, designing a low latency NoC router is still a challenge for on-chip systems [8]. In this work, a microarchitecture for a low latency NoC router is proposed (this work is open source and is publicly released at http://opencores.org/project,an-fpga-implementation-of-low-latency-noc-based-mpsoc). The advantages of existing related works such as look-ahead routing computation [2, 3], combination of VC/SW allocation [4, 5, 9], and replacing VC allocator with a queue of free VCs [3, 6, 9] are used. Meanwhile, several optimization techniques such as efficient masking of the allocation requests and broadcasting only two status bits instead of using proxy credit register have

been applied to provide high operating frequency, reasonable hardware cost, and high utilization of the available buffer space.

The remainder of this paper is organized as follows. In Section 2, the conventional NoC router pipeline stages and its hardware architecture are discussed. Section 3 reviews existing low latency NoC architectures. Section 4 proposes a low latency NoC router microarchitecture. In Section 5, the FPGA implementation results are analyzed and discussed. This section also discusses several optimization techniques which are applied to the proposed NoC router on an FPGA device.

2. Conventional Router Architecture

A conventional NoC router [10] has four consecutive pipeline stages.

(1) Route computation: this stage determines the output port that a packet must be sent to.

(2) VC allocation: this stage assigns an empty VC in the neighboring router connected to the output port. Since several header flits may send requests for the same VC, arbitration is required. The routing computation as well as the VC allocation only requires the header flit. The body and tail flits will follow their respective header flit.

(3) Switch allocation: if VC allocation is successful, the third stage sends request to the switch allocator to allocate the output port.

(4) Switch traversal: if the switch allocation is successful, the flit will be passed to the crossbar and be delivered to the output port.

A conventional virtual channel NoC router block diagram is illustrated in Figure 1. The NoC router consists of input ports, VC/SW allocators, routing computation module, and a crossbar. The input ports buffer input flits and send requests to the allocators. The routing computation module determines the output port based on the routing algorithm. After the route computation, a free output VC (OVC) in the next router is assigned to the input VC (IVC) by sending request to the VC allocator. If an OVC is successfully assigned, then another allocation request will be sent to the switch allocator. The crossbar is then configured to send the desired flit to the output port if the switch allocation request is granted. In order to send requests to the switch allocator, the available space in the next router buffer must be known. Hence, output port modules maintain a set of credit counters to keep track of available buffer space for each OVC.

The allocator is the most challenging module to design since the overall NoC router performance and area overhead will be dominated by this module. Moreover, allocators are located in the NoC critical path. An allocation is required when several agents (IVCs) require access to several resources (OVCs or output ports) simultaneously. Generally, three types of allocators are widely used in NoC router microarchitecture design, namely, wavefront [13], separable input-first, and separable output-first allocators.

Table 1: The number and size of arbiters used in conventional allocators [10]. p represents the number of input ports in an NoC router and v is the number of VCs per port.

Allocator type	First-stage arbiter		Second-stage arbiter	
	Number	Size	Number	Size
VC	pv	$v:1$	pv	$(p-1)v:1$
SW	p	$v:1$	p	$(p-1):1$

A comprehensive analysis on the following allocators [14] shows that separable input-first allocators have the advantage of lower communication delay, area overhead, and power consumption compared to other schemes. Hence, the separable input-first allocator has been chosen to be implemented in our low latency NoC router. A separable input-first allocator consists of two levels of arbitrations. In the first arbitration stage, for each input port, only one request of all IVC requests is granted. Since several input ports may request the same output resource, another arbitration stage is required to resolve this limitation. Table 1 shows the number of arbiters and the arbiters' size required for the VC and switch allocations. VC allocator consumes a large number of resources compared to SW allocator.

In this work, we assume that no input port sends a packet back to its own output port. As an example, when router a sends a packet to router b, it is not expected that router b sends that packet back to router a. This condition never happens in a network having minimal routing. Moreover, it must be even avoided in nonminimal routing as it results in a 180° turn in channel dependency graph which may lead to a deadlock condition [15]. This assumption reduces the hardware complexity and also the critical path delay of NoC router components such as the crossbar and allocators.

3. Related Works on Low Latency NoC Router

To eliminate the need for a separate pipeline stage for routing computation, *look-ahead* routing was proposed [16]. In the look-ahead routing, the output port is computed one router in advance and is attached to the header flit. Hence, at the header flit arrival time, the NoC router can initiate the sending of the allocation request to the precomputed output port while computing the next router output port in parallel.

To remove the dependency between VC allocation and switch allocation, *speculative switch allocator* was proposed [17]. In speculative allocation, a header flit is allowed to send requests to both switch and VC allocators in parallel by speculating that an OVC will be assigned successfully. However, in the case when the VC allocation is unsuccessful, the granted switch allocation is ignored. Since ignoring a granted request results in an unused time slot, the NoC sets a higher priority to the switch allocation requests that have already been assigned OVCs (nonspeculative requests) to the ones that have not been assigned any OVC. The speculative allocation performs well when the router carries light traffic. However, in dealing with heavy traffic, it becomes inefficient due to the increase in unsuccessful speculations.

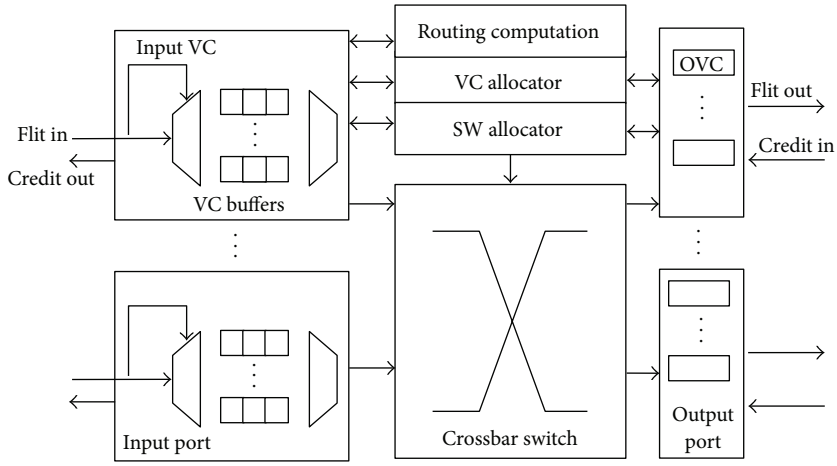

Figure 1: Conventional VC router architecture.

Mullins et al. [6] proposed a *precomputing arbitration* technique to reduce critical path delay of the separable input-first VC allocator. In this work the switch allocation stage has been removed from critical path. However, it is not efficient when dealing with noncongested traffic, as conflict between the newly arrived flits may result in unused crossbar time slots.

Kumar et al. [18] proposed *express virtual channels* (EVCs). The EVCs have been added to the normal VC which allows the packets travelling to long distance core, bypassing multiple routers along the path. EVCs result in reduction of the overall communication latency. However, they require additional area overhead and have no effect on flits that are sent between neighboring cores.

Dynamic Priority-Based Fast Path NoC router [19] prioritizes flits traveling in frequently used paths to bypass the switch allocation stage by sending arbitration request one clock cycle in advance to the next router. The proposed architecture requires additional hardware modules such as path frequency analyzer as well as priority-based arbiter.

Lu et al. [5] proposed a low latency NoC router microarchitecture for FPGA-based implementation. In order to design a two-clock-cycle latency router, a parallel switch/VC allocator is proposed. The main drawback of the design is that an IVC can only store the flits that belong to the same packet. Moreover, even when a tail flit of a packet has been sent out but as long as it remains inside the buffer of next router, the IVC cannot accept a new packet. As a result, although the whole packet has been sent out from the router, as long as the tail flit has not been sent out from a neighboring router, the assigned OVC cannot be reallocated. This problem happens because the counters which are supposed to keep track of credit signals are implemented inside the input port modules while no mechanism is implemented to pass the remaining credits to the new input port that wants to use the OVC. Therefore, the only condition on which an OVC can be reallocated is when it becomes empty. This will cause inefficient buffer usage and results in high latency during heavy traffic (refer to Section 5.4 for experimental configuration and description). Moreover, the maximum operating frequency was not disclosed [5] to estimate the effect of such approach.

Table 2: Synthesis results for CONNECT and SOTA mesh network [7].

4 × 4 mesh w/4 VCs	Xilinx LX240T		Xilinx LX760	
	LCs	MHz	LCs	MHz
SOTA [11] (32-bit)	36%	158	12%	181
CONNECT [12] (32-bit)	15%	101	5%	113

In [3], a *combined VC and switch allocation* method was proposed. The VC allocation is replaced by a queue of free VCs for each destination port. Similar to the speculative method, the proposed method needs to set a higher priority for requests from nonheader flits. It is possible that a header flit which has been granted by the switch allocation may not be able to be assigned any OVC. It suffers the same problem as the speculation method during heavy traffic. The proposed NoC is open source [11] and is targeted for ASIC.

CONNECT [7] is an open source reconfigurable soft core NoC targeted for FPGA devices. CONNECT supports different numbers of VCs, buffer size, allocation type, and pipeline stages. The router pipeline stages can be configured to a minimum of one-clock-cycle latency. The authors compared their router architecture with [3]. The comparison results (Table 2) show significant reduction in hardware cost. However, the reported maximum operating frequency is low since all router stages are cascaded in series. Another drawback of the design is the implementation of all the input memory buffers as LCs. Although FPGAs have sufficient amount of embedded memory blocks, this trade-off is critical to achieve one-clock-cycle latency. Replacing memory blocks by LCs causes a high hardware resource usage specifically for wide buffer sizes.

Our proposed NoC router adapts some of the distinct features of previous works such as nonspeculation switch/VC allocation, merging input buffers for FPGA optimization [5], queuing of free VCs, high operating frequency [3], and low hardware cost [7]. At the same time, our proposed router

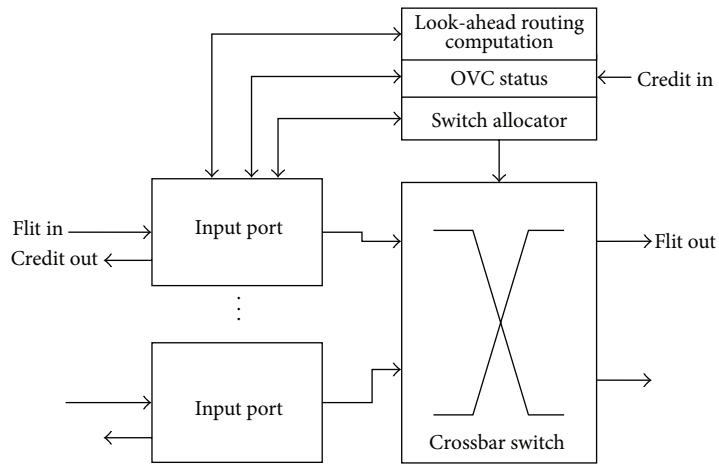

Figure 2: The functional block diagram of the proposed NoC router.

microarchitecture overcomes the limitations of previous works such as memory buffer usage inefficiency [5], high hardware cost [3], low operating frequency, and imbalanced usage of memory blocks and LCs [7]. In order to evaluate our approach, we benchmark our work with CONNECT NoC router [7].

4. Proposed NoC Router Microarchitecture

In this work, we present a two-clock-cycle latency router microarchitecture with parallel VC and switch allocator. Our proposed architecture eliminates the need of setting higher priority to any IVC requests. In the proposed NoC router architecture, any request which has been granted service by the switch allocator is able to pass a flit to the output port successfully. An efficient masking technique is proposed to filter all switch allocation requests that are not able to pass flits to the output port, either due to the lack of free space in assigned VC or due to the lack of free VC in the output port for nonassigned VC requests. The masking technique also provides an efficient usage of VC memory buffers. Our proposed technique has minimal impact in timing and area overhead of an NoC router. It is also fully parameterizable in terms of number of VCs, buffer width, and flit width.

Figure 2 illustrates the functional block diagram of the proposed router architecture. The main components include the input switches, look-ahead routing computation module, OVC status module, SW allocator module, and a crossbar. Input port module is the first module that receives the packet flits. Upon receiving a flit, the VC-ID of each flit is examined in order to allocate it into the respective VC FIFO buffer. Look-ahead routing is applied in order to relax the dependencies of the first stage of conventional router, which is the route computation.

Conventional VC allocator has two main drawbacks in terms of circuit complexity and high critical path delay. Using a conventional VC allocator, it is possible to allocate multiple OVCs inside the same output port in one clock cycle. However, in any clock cycle, only one flit can be passed to an output. Hence, multiple simultaneous OVCs allocations will give no significant improvement in performance. A speculative NoC with canonical VC allocator may outperform a speculative NoC using queues of VCs in terms of performance, as simultaneous multiple VC allocations on one port reduce the number of speculative requests and, thereby, speculation failure. However, it has no advantage when compared to nonspeculative NoC using queue of VCs. For instance, consider a situation when two packets of p_a and p_b are located in different router ports. Both packets are not yet assigned an output VC and are targeting the same destination port which has only one available OVC. In this situation, the use of canonical VC allocator would result in 50% speculation failure because the grant from switch and VC allocation may not be given to the same packet. On the contrary, the use of queue of VCs results in the grant of available OVC to the packet which receives the switch allocation grant.

Using queue of available VCs has the advantage of less area overhead and faster execution time as it only requires one arbitration stage. The second arbitration stage has been added to the canonical VC allocator to remove conflicts of assigning one OVC to several IVCs. By limiting VC allocation upon successful switch allocation, this conflict is removed by the switch allocator.

In our design, we replace the VC allocator with the queue of free OVCs. The OVC will be assigned whenever a nonassigned VC request (header flit) is granted by the switch allocator. Since all nonassigned VC requests which were not able to receive any OVC were masked before sending to the switch allocator, VC is assigned in a nonspeculative fashion. The VC status module keeps track of all OVCs, providing information on available space (credits) in each OVC and the list of free OVCs for each port. A free OVC is defined as an OVC which has not been assigned and has at least one available credit.

The switch allocator grants allocation requests sent from input ports. As previously mentioned, allocation is located in the critical path. The NoC critical path delay can be divided into three phases: *masking*, *allocation,* and *updating*. Designing an efficient low latency router requires reduction of these delay components as much as possible.

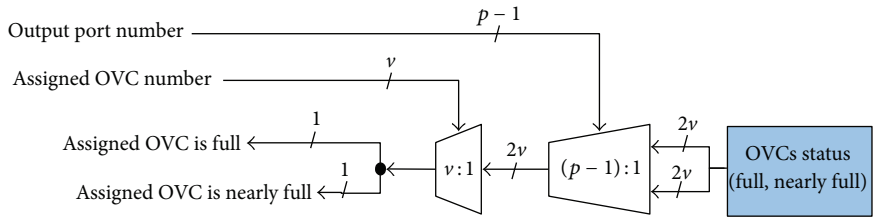

Figure 3: The status signals transferring from OVCs to an IVC.

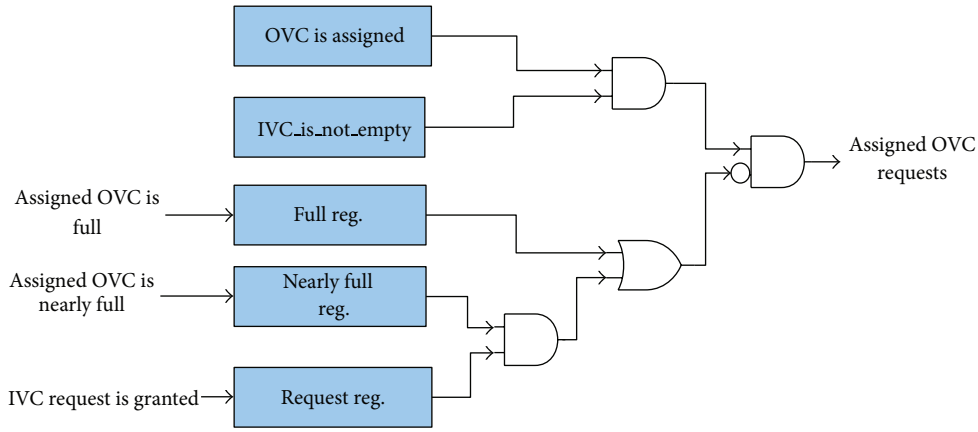

Figure 4: The assigned OVC requests masking.

4.1. Masking. One of the challenges to design a low latency NoC router is to mask all switch allocation requests when the assigned OVC buffer space is full. To this end, the conventional router implements $(p-1)v$ credit counters inside output ports to keep track of available space in each OVC. One way to mask the input requests is to transfer the status signals of assigned OVCs to the IVCs using multiplexer. However, it increases the critical path due to the multiplexing delay.

Another way for masking is to locate the credit counters locally inside input port modules and multiplex the credit-in signals from output ports to the input ports. This approach removes the multiplexing delay from critical path. However, it needs a mechanism to initiate the counters with the number of available credits inside the newly assigned OVC. The related work in [3] proposed the use of proxy counters inside the input ports. These proxy counters are initialized in the first cycle of switch allocation with the credits values inside the output ports. This approach suffers from additional area overhead due to the redundant counters implementation and also large multiplexer width. Note that transferring one single bit to all IVC requires pv multiplexers with the size of $(p-1)v:1$. Hence, the resource consumption becomes more critical when the buffer size increases.

In our approach, we propose a feedback of only two fixed status bits for each IVC. These two bits determine the condition of assigned OVCs which are full and nearly full (i.e., has only one empty place). Figure 3 illustrates the architecture for transferring the assigned OVC status to an IVC. It consists of two levels of multiplexing, where the first level selects all OVC status inside the output port and the second level returns the assigned OVC status bits. This structure is repeated for each respective IVC. The status bits are registered at the input ports in order not to add the multiplexing time to the router critical path. The one-clock-cycle delay in receiving status bit will be handled by the masking filter inside each input port. To this end, the granted IVC request that is delivered to each input port at the end of allocation stage is stored in a register. Hence, this register indicates the condition of IVC request in the last clock cycle. Now, each IVC masks the requests if (i) the status registers show the full status for the assigned OVC and (ii) an assigned OVC status register is nearly full and the request status register indicates that the previous IVC request has been granted by the allocator. In the second condition, when the last request is granted, the real OVC status is full but will be updated in the next clock cycle. The masking filter is illustrated in Figure 4.

Another challenge in sending requests to the switch allocator is the time when the IVC has not been assigned any OVC. In speculative methods [3, 17, 20], the nonassigned VC requests are not masked and the validity of granted signals is checked during the updating time. In our proposed design, we keep track of all OVCs inside a status module (i.e., available space (credits) for each OVC), the list of free OVCs for each output port, and one registered flag indicating if the output port has any available OVCs. We use these flags for masking the nonassigned OVC request in the case when there is no available VC in the output port. The nonassigned OVC request masking is illustrated in Figure 5. Allocation requests will be sent to the switch allocator if any of the nonassigned or assigned OVC requests are asserted.

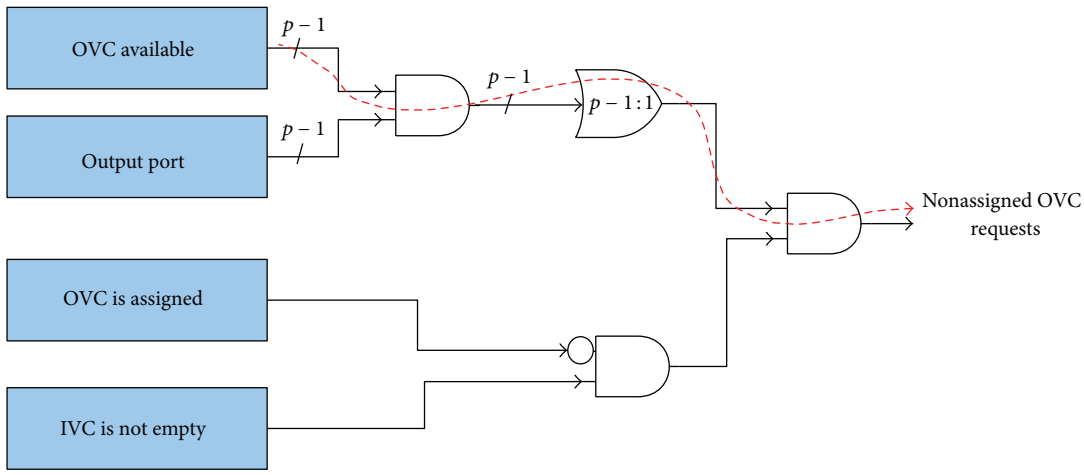

Figure 5: The nonassigned OVC request masking.

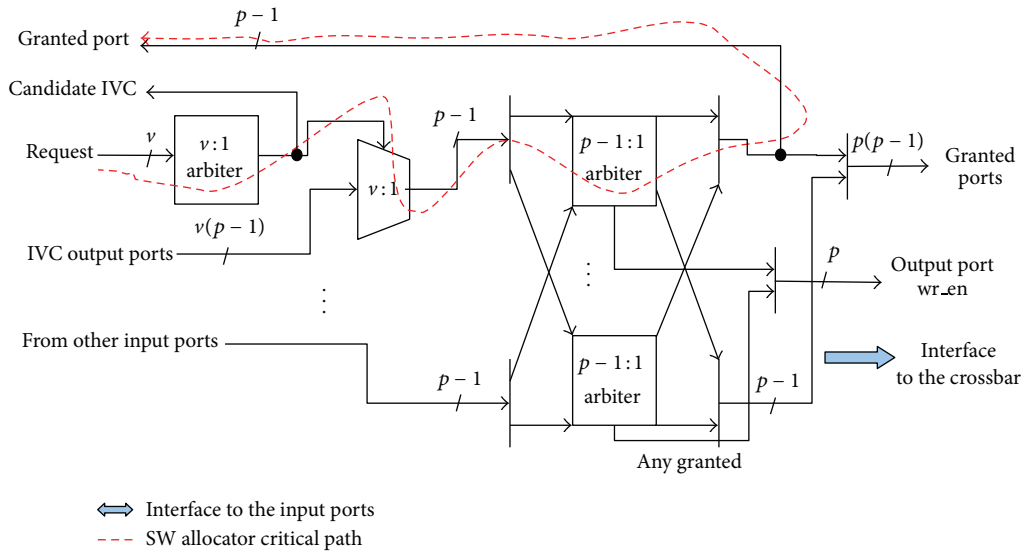

Figure 6: The switch allocator functional block diagram.

The additional masking gates for filtering invalid nonassigned OVC requests result in having higher masking delay when compared to speculative NoC router. However, as will be discussed in next subsection, speculative NoC router requires longer allocation time due to priority based switch allocation.

4.2. Allocation. After masking the IVC requests, these requests are sent to the switch allocator. The functional block diagram of the switch allocator is illustrated in Figure 6. The critical path of the allocation stage is highlighted using the dotted line. The path starts by passing masked IVCs requests to a $v:1$ arbiter to grant one request for each input port. Then, a $v:1$ multiplexer selects the winner IVC output port among the rest. Next, the winner output ports will be sent to the second $p-1:1$ arbiter in order to arbitrate between input ports that request the same output port. Finally, the results of the second arbitration stage are sent back to each input port. The result of the first stage arbitration is also returned to the input port as a candidate IVC. The candidate outputs will be used to prepare the updating signal before the granted result becomes available.

Due to having two levels of arbitrations in the switch allocator, arbiter delay is an important parameter in defining the NoC critical path. Hence, to minimize the arbitration delay, *fast arbiter* [21] proposed by Dimitrakopoulos et al. is used.

Speculative switch allocator prioritizes the nonspeculative requests over the speculative one and, hence, it has longer allocation delay compared to our design. The speculative switch allocator functional block diagram is shown in Figure 7. Prioritizing nonspeculative requests over the speculative ones is done by using two parallel switch allocators: one for handling speculative requests and another one for nonspeculative ones. Switch allocator architecture is shown in Figure 6. The switch allocation grants are obtained by ORing both speculative and nonspeculative grants for the same input port. However to remove any conflict between

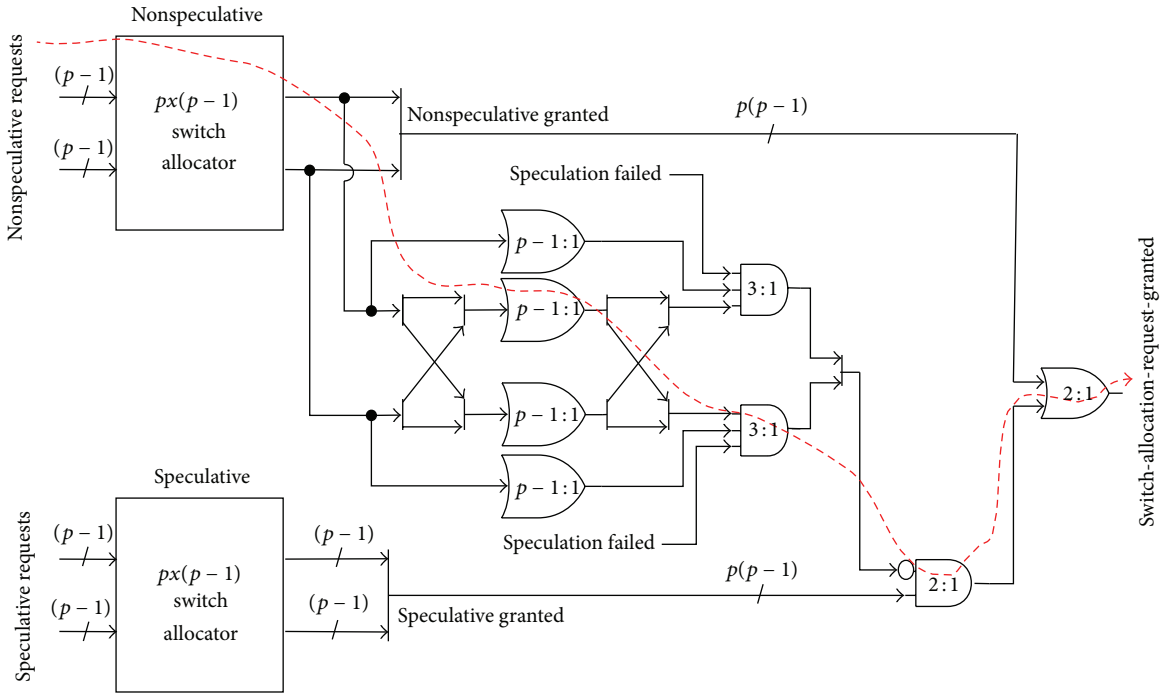

Figure 7: The speculative switch allocator block diagram.

both allocators, the speculative grants are masked in three conditions. First is when one input port receives both speculative and nonspeculative grants. Second is when same output port is granted for both speculative and nonspeculative requests. Third condition is when the speculation fails due to unavailability of free OVC in the destination output port. The first condition is checked by adding p number of $p - 1:1$ reduction OR gates at the granted nonspeculative output ports number of each input port. The second condition is checked by adding p number of $p-1:1$ reduction ORs to sum all granted nonspeculative signals of all input ports which point to the same output port.

Since the invalid nonassigned OVC requests are masked at the beginning of switch allocation stage, our design has longer masking delay when compared to speculative NoC router. However, it needs only one switch allocator module where the granted destination port can be used directly without any speculation mask. As the added masking components in our design are similar to the masking components that are needed in speculative NoC router, it is expected that both designs have similar critical path delay.

4.3. Register Updating. The switch allocation results are broadcasted to the other modules (input ports, VC status module, and the crossbar) to update the internal registers. The result of switch allocation is used to update the input ports' internal registers such as input buffers status registers. Other registers that must be updated at the end of the allocation stage are the registers located in OVC status module. Two registers are allocated for each pv OVC. A one-bit status register indicates if the respective OVC has already been assigned. Meanwhile, another credit counter keeps track of credits in the output port.

In our proposed design, the VC allocation and switch allocation are done in parallel when a header flit receives a grant signal from the switch allocator. At this time, the assigned OVC status register must be asserted to indicate that the respective OVC has been assigned. On the other hand, when a tail flit is granted, the assigned OVC must be deasserted. Moreover, the credit counters must be decremented by one for each granted IVC request. Hence, the challenge is to generate the update signals for the status registers when the OVC has not yet been assigned.

The OVC updating functional block diagram is illustrated in Figure 8. The OVC registers updating process is divided into four stages. In the first stage, one candidate OVC is selected for each output port. First, the list of free OVCs is generated by masking the unallocated OVC which has no available credits. Then, the candidate OVC is obtained by passing the list of available OVCs to a $v:1$ round-robin arbiter for each output port and then is broadcasted to all input ports. In the second stage, each input port selects the OVC which will be accessed by the candidate IVC. The assigned OVC for each IVC is stored in an internal register. For header flits without an assigned OVC, a candidate OVC will be selected. In the third stage, the granted port signal is received from allocator. Passing the granted port and the OVCs number to a decoder will generate a one-hot code depicting OVC credit counter which must be updated. Masking this signal with the tail and header flags of the input flit will generate another two one-hot codes. The first code indicates if the respective OVC must be allocated when sending the header flit. The second code specifies if the respective OVC must be released when the tail flit is passing through. In the forth stage, all signals from all output ports are ORed to generate the final updating signals. The OVC update delay path is illustrated

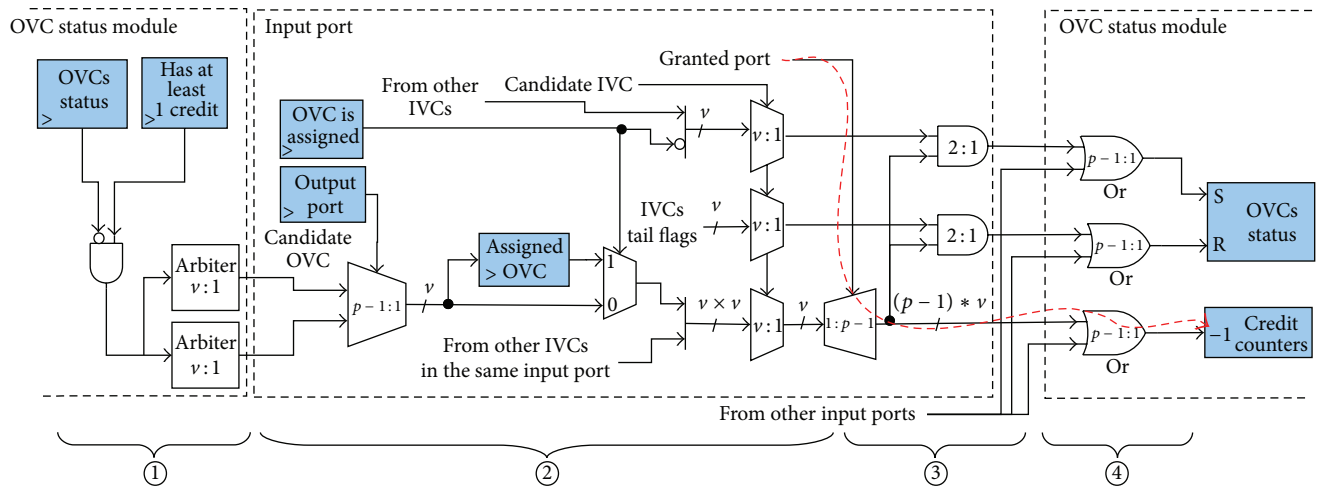

Figure 8: The OVC updating schematic diagram.

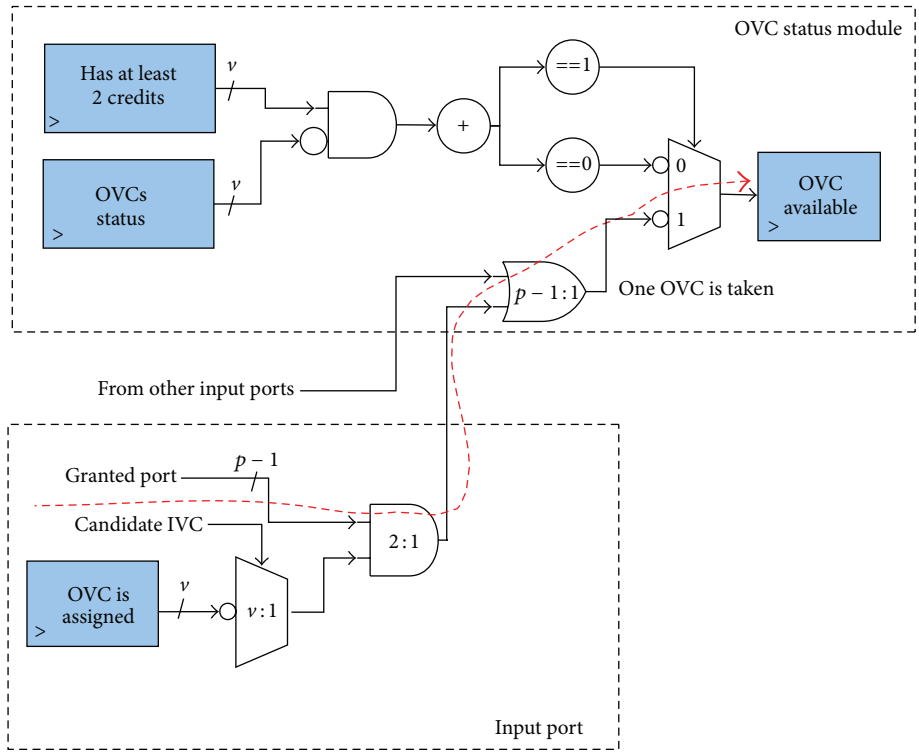

Figure 9: Registering the OVC available signal.

using a dotted line in Figure 8. It consists of one AND gate (inside the demultiplexer), a wide $p - 1:1$ OR gate plus a subtracter which is equal to the IVC update time.

The last OVC register that must be updated at the end of the switch allocation is the one-bit registers that show the availability of OVC on the output ports. These signals can be directly extracted by using a $v:1$ OR gate after the list of available OVCs for each port. However, the extraction delay would be added to the masking time and this results in a longer critical path delay. Hence, a circuit shown in Figure 9 registers this signal. In order to send a nonassigned OVC request to the switch allocator, the destination output port must have at least one free OVC with two available credits. To this end, the number of free OVCs with more than one credit is counted. The output port is marked as nonavailable OVC port in two conditions: first if there is no available OVC in desired port and second when the port has only one available OVC, but it is to be taken by an IVC. The critical path to update the available OVCs register is shown using a dotted line in Figure 9.

4.4. Routing Algorithm. Routing algorithm determines the output port which a packet must be sent to to reach its destination. In deterministic routing, for each distinguishable

source and destination pair, there is only one path which the packets can pass through. Deterministic routings act well when dealing with uniform traffic where congestion has been distributed equally across all links in an NoC. However, the nature of NoC traffic is bursty [22] which results in imbalanced distribution of traffic across all links. Hence, deterministic routing results in poor performance for such traffic. In adaptive routing, multiple paths between a source and destination nodes can be selected according to some congestion metrics. Hence, the routing algorithm will select from possible output port(s) which a packet can be sent to. As packets can be sent to multiple ports, a port selection module is required to select the desired output port among them. In the case of look-ahead deterministic routing algorithm, only single output port is selected and it can be directly used in our proposed design. However, for supporting adaptive routing, our proposed router requires the implementation of additional port selection module. The architecture of port selection module varies depending on the network topology and routing. An example of port selection module for a 2D mesh topology using minimal routing can be found in related works [23, 24]. In a 2D mesh having minimal routing, each packet can be sent up to two destination ports. The quarter in which the destination port has been located is added to the look-ahead routing results. In the case when the packet can be sent to only one destination port (i.e., the destination node is located in the same row or column with current router), another one-bit flag is added to show that the result of look-ahead routing must not be overwritten. The port selection module is shown in Figure 10. This module consists of two-level multiplexers. The first level multiplexer selects the preferred output port based on the quarter in which the destination port is located. Another multiplexer is required to select between the preferred output port and originally look-ahead routing result based on the *overwrite* flag.

As the destination port is required at the beginning of switch and VC allocation stages, the delay of port selection modules will be added to the router critical path. Moreover, it depends on the NoC topology and routing algorithm, where the delay increases with the maximum number of permitted output ports for one packet. While the following technique can be adapted to our design, we proposed another alternative [25] to send the precomputed preferred output ports to the neighbouring routers and make the port selection inside the look-ahead routing module. The following technique has no influence in the maximum operating frequency and has minimal area overhead (<2%). The proposed technique can improve the performance of the router in dealing with imbalanced traffic, for example, 10% higher injection ratio in a 5×5 mesh based NoC using partial adaptive routing in dealing with matrix-transpose traffic.

5. FPGA Implementation

In this section, the optimization techniques for NoC based FPGA implementation are discussed. The analyses on the implementation results using both DE2-115 (Cyclone IV) and DE4 (Stratix IV) Altera FPGA development boards are discussed.

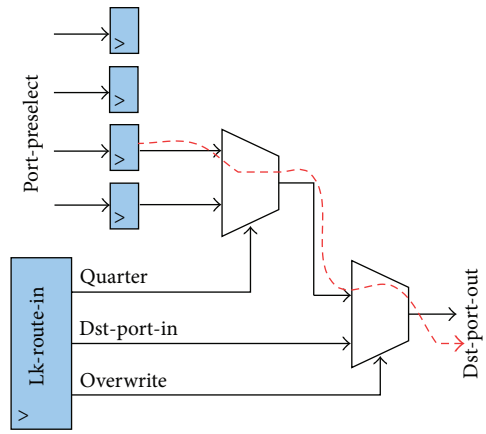

Figure 10: Port selection for 2D mesh.

5.1. Replace One-Hot Multiplexers with Conventional Multiplexers. One-hot multiplexers have an advantage of low area overhead and low latency in ASIC [26]. On the other hand, the output of arbiters is presented as one-hot code to control one-hot multiplexers. Hence, in an ASIC NoC, it is preferred to use one-hot multiplexing. However, NoC realization in FPGA using one-hot multiplexing requires more logic elements. Since a large portion of the NoC router consists of multiplexers (e.g., the crossbar switch or for broadcasting the status signals between different modules), replacing one-hot multiplexer with conventional multiplexer can result in a lower hardware resources usage.

5.2. Merging All IVCs Buffers of the Same Input Port in One Block Memory. The conventional implementation of input port VC buffers in an ASIC NoC router is done by demuxing the received flits into designated VCs which are located in separated memory (either Flip-Flop or SRAM) and then multiplexing the VCs to the crossbar (Figure 1). However, implementation of multiplexers is more costly on FPGAs, in comparison to ASIC. Besides that large multiplexing width increases the area overhead and power consumption of a VC router in comparison to other alternatives such as Multiple Physical (MP) Networks. By partitioning physical channels across multiple independent networks, the MP NoC is able to remove the need for having such very wide multiplexers [27]. In our proposed design, to remove these large-width multiplexers, as suggested in [5], we implemented all IVCs buffers of an input port inside a single dual port memory, where one port of this memory is dedicated for writing the incoming flits and the other port is used for reading the desired flit which is going to be passed to the crossbar. The memory read and write addresses are selected from IVCs read and write pointers (Figure 12). This approach removes costly multiplexers/demultiplexers from the input buffers. Moreover, this approach reduces the overall number of used block memories in an FPGA device due to the mapping of each individual dual port memory into one block memory. However, the input switch needs to keep track of the flit header information separately. The related work in [5] uses registers to store these data, which allows only a single

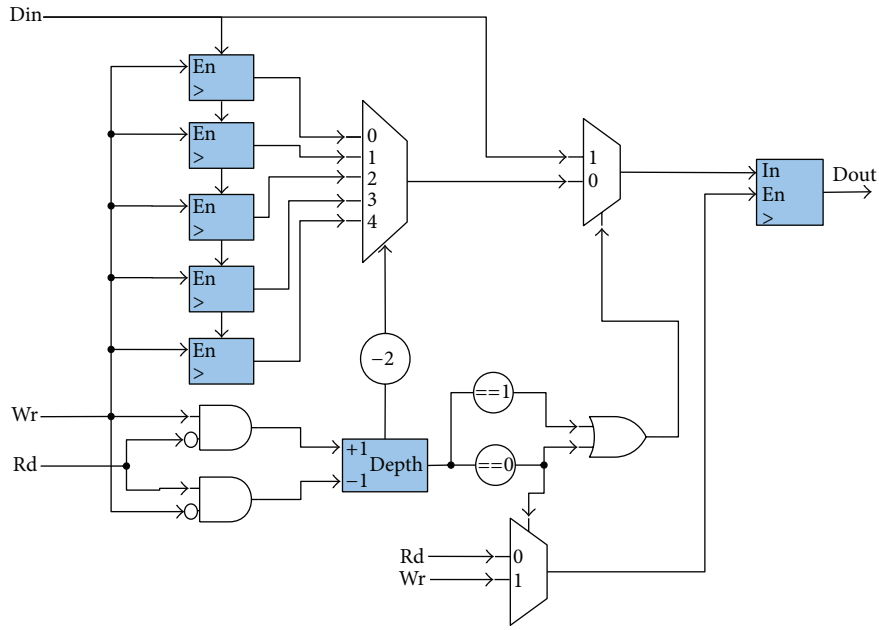

Figure 11: The block diagram of FWFT FIFO with depth of 6.

packet to be stored on a single VC buffer, which results in inefficient buffer usage. In our proposed microarchitecture, we store the flit header information (such as output port address, look-ahead port address, and tail flag) inside a small first-word fall-through (FWFT) FIFOs. This architecture allows multiple packets to be stored depending on the buffer availability. Since these FWFT FIFOs are used frequently inside each input port, we design a resource-lean FWFT FIFO as illustrated in Figure 11.

5.3. Synthesis Results. The NoC code was written in Verilog HDL and compiled using Quartus II version 13 software. The code was verified using Modelsim Altera version. Figure 13 shows the comparison of LCs utilization of the proposed NoC router when it is configured with two and four VCs. In total, an NoC router with four VCs consumes twice LCs compared to the one with two VCs. Cyclone IV EP4CE115 FPGA device has been used in this analysis.

We also generated two CONNECT [7] NoC routers, one with one- and another with two-clock-cycle latency using the online CONNECT NoC router generator tools [12]. The proposed NoC router and the two CONNECT NoC routers are configured with the same parameter of 4 VCs with capacity of 4 flits per each VC and flit payload width of 32 bits. To obtain the maximum operating frequency, we connect 16 NoC routers to form a 4 × 4 mesh based topology. The hardware utilization summary and the maximum operating frequency obtained for Altera Cyclone IV EP4CE115 and Stratix IV EP4SGX230 FPGA are shown in Table 3. With the same router parameters, CONNECT is twice costly in term of LCs utilization due to implementation of input memory buffers as LCs. Our proposed two-clock-cycle NoC architecture is able to work with two times faster operating frequency compared to CONNECT NoC when configured as one-clock-cycle latency router. When compared to the two-clock latency CONNECT router, our proposed NoC router is 35% and 20% faster when implemented in Cyclone IV and Stratix IV FPGA, respectively.

5.4. Performance Results. In order to compare network performance results, we generate cycle-accurate behavioral model of the one-clock-cycle latency CONNECT [12] and our proposed architecture using Verilator [28]. Both NoC are configured in a 5 × 5 mesh topology having 4 VCs on each port with the size of 4 flits per each VC and the flit payload width of 32 bits. As CONNECT [12] only supports dimension order routing (DoR), we use DoR for both routers. All NoC endpoints are connected to the custom traffic generator modules. The traffic generators are responsible for injecting network packets into the NoC routers and collecting performance statistics as the packets are received by destination cores.

Figures 14 and 15 illustrate the performance analyses of our proposed NoC router against one-clock-cycle latency CONNECT router under uniform random traffic. For a fixed injection ratio, each endpoint injects packets of 5 flits into randomly selected destination router. In Figure 14, we represent the delay and load in clock cycles and flits per clock cycle, respectively. Adopting the same operating frequency, as expected CONNECT-one-clk has lower average packet latency, since our design is two clk cycles' latency router. However our experimental results showed when the load injection ratio is above 40% our proposed architecture outperforms CONNECT-one-clk.

Figure 15 illustrates an example when each router is running at its own maximum frequency (CONNECT with 40 MHz and proposed design with 80 MHz). In this figure, the load and the delay are scaled based on Gbit/second and nanosecond, respectively. Our proposed architecture offers significantly lower delay as injection ratio increases.

Table 3: Hardware utilization summary of one NoC router on Cyclone IV & Stratix IV Altrea FPGA.

4 VCs/4 flits per VC	EP4CE115			EP4SGX230		
	Proposed NoC	CONNECT two-clock	CONNECT one-clock	Proposed NoC	CONNECT two-clock	CONNECT one-clock
Logic cells (LCs)	2,890 (2.5%)	5,934 (5.2%)	5,690 (5.0%)	2,722 (1.5%)	5,473 (3%)	5,570 (3%)
Memory blocks (M9K)	5 (1.2%)	—	—	5 (0.6%)	—	—
Maximum frequency	88 MHz	65 MHz	41 MHz	177 MHz	148 MHz	91 MHz

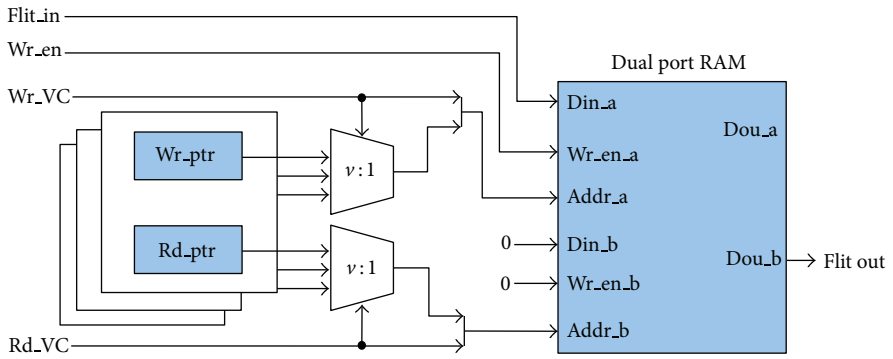

Figure 12: The input port buffer block diagram.

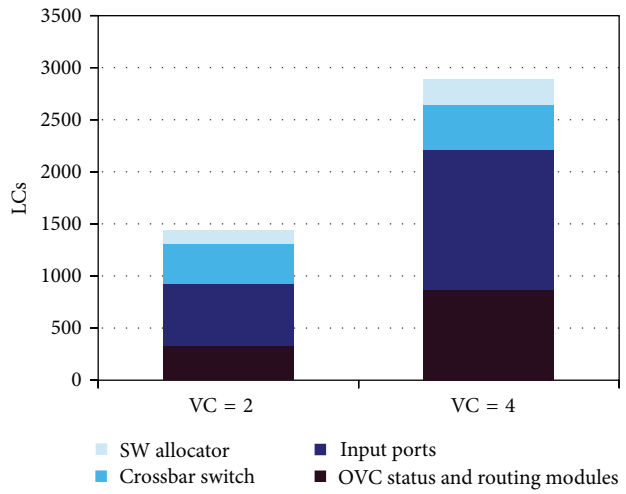

Figure 13: Logic cost comparison for VCs number of 2 and 4 (ports number = 5, payload width = 32, and buffer width = 4).

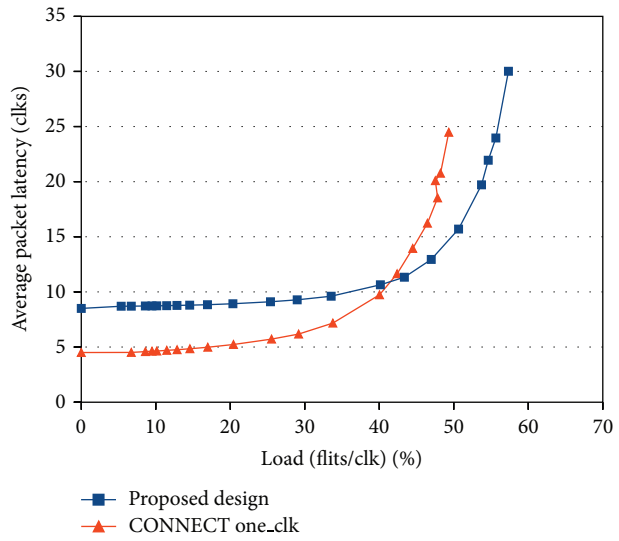

Figure 14: Load-delay performance, for CONNECT-one-clock and proposed NoC under uniform random traffic in terms of clk cycles.

Moreover, the maximum injection ratio which can be fed to our proposed NoC is 2.3 times higher compared to CONNECT-one-clk.

In order to observe the influence of VC reallocation algorithm, we generate two NoC router architectures, one with nonempty VC reallocation ability and the second with VC reallocation that requires empty VC. Our experiment showed when the number of VCs is 4, there is no significant difference in performance metric. However as illustrated in Figure 16 for the router with 2 VCs, nonempty VC reallocation provides approximately 40% higher maximum injection ratio compared to VC reallocation that requires empty VC.

6. Conclusion

In this work, a two-clock-cycle pipeline wormhole virtual channel NoC router microarchitecture was proposed. The router first pipeline stage is the parallel look-ahead route computation with a nonspeculative VC/switch allocation. The second stage is the crossbar switch traversal. The proposed router microarchitecture is optimized in three main criteria, which are hardware cost, maximum operating frequency, and QoS compared to existing works.

Compared to [3], the proposed nonspeculative allocation removes the need for prioritizing any IVC request and

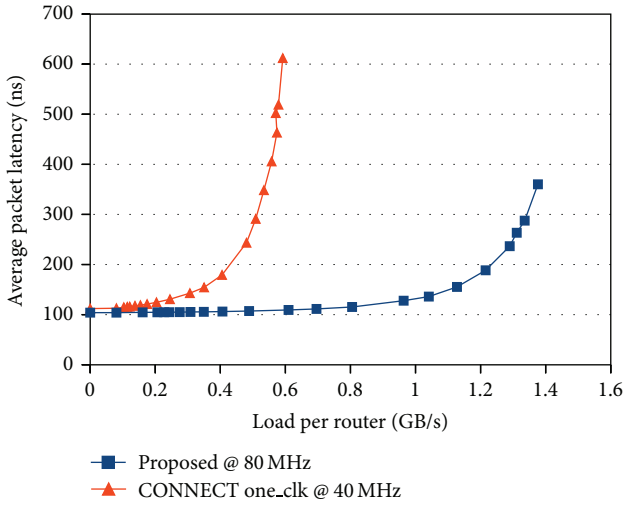

FIGURE 15: Load-delay performance for CONNECT-one-clock at 40 Mhz and proposed NoC router at 80 Mhz under uniform random traffic.

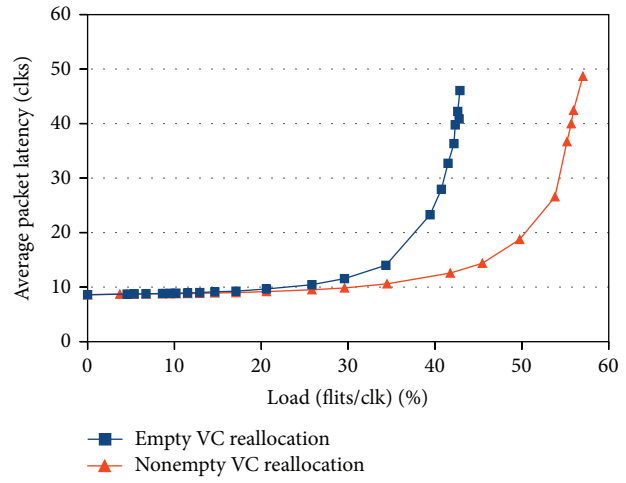

FIGURE 16: Nonempty VC reallocation performance versus empty VC reallocation for NoC router with 2 VC, under uniform random traffic.

reduces unused time slots, which result in better QoS. Broadcasting only two bits of OVC status to the input ports instead of applying proxy credit counters results in lesser hardware cost. Our analysis shows that the proposed masking technique has negligible influence on NoC critical path delay. Compared to CONNECT [7], our NoC router has 50% lesser LCs utilization. The proposed router works with two times higher operating frequency when CONNECT is configured as one-clock-cycle latency and is 35%~20% faster when CONNECT is configured as two-clock-cycle latency router. Our experimental results showed that our proposed two-clock-cycle NoC architecture outperforms CONNECT-one-clk by 2.3 times in terms of performance.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgment

This work is supported by the Ministry of Education of Malaysia, Fundamental Research Grant scheme (UTM Vote no. 4F327).

References

[1] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proceedings of the Design Automation Conference*, pp. 684–689, 2001.

[2] R. Mullins, A. West, and S. Moore, "The design and implementation of a low-latency on-chip network," in *Proceedings of the Asia and South Pacific Design Automation Conference*, pp. 164–169, January 2006.

[3] D. U. Becker, *Efficient microarchitecture for network-on-chip routers [Ph.D. thesis]*, Stanford University, Stanford, Calif, USA, 2012.

[4] S. T. Nguyen and S. Oyanagi, "The design of on-the-fly virtual channel allocation for low cost high performance on-chip routers," in *Proceedings of the 1st International Conference on Networking and Computing (ICNC '10)*, pp. 88–94, IEEE, November 2010.

[5] Y. Lu, J. McCanny, and S. Sezer, "Exploring virtual-channel architecture in FPGA based networks-on-chip," in *Proceedings of the 24th IEEE International System on Chip Conference (SOCC '11)*, pp. 302–307, September 2011.

[6] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in *Proceedings of the 31st Annual International Symposium on Computer Architecture*, vol. 32, pp. 188–197, ACM, June 2004.

[7] M. K. Papamichael and J. C. Hoe, "CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs," in *Proceedings of the 2012 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA '12)*, pp. 37–46, ACM, February 2012.

[8] J. D. Owens, W. J. Dally, R. Ho, D. N. Jayashima, S. W. Keckler, and L.-S. Peh, "Research challenges for on-chip interconnection networks," *IEEE Micro*, vol. 27, no. 5, pp. 96–108, 2007.

[9] A. Kumar, P. Kundu, A. P. Singh, L.-S. Peh, and N. K. Jha, "A 4.6 Tbits/s 3.6 GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS," in *Proceedings of the 25th IEEE International Conference on Computer Design (ICCD '07)*, pp. 63–70, October 2007.

[10] S. Kumar, A. Jantsch, J.-P. Soininen et al., "A network on chip architecture and design methodology," in *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pp. 105–112, Pittsburgh, Pa, USA, 2002.

[11] Open source network-on-chip router RTL, 2013, https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router.

[12] CONNECT: Configurable network creation tool, 2014, http://users.ece.cmu.edu/~mpapamic/connect/.

[13] Y. Tamir and H.-C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 1, pp. 13–27, 1993.

[14] D. U. Becker and W. J. Dally, "Allocator implementations for network-on-chip routers," in *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*, pp. 52:1–52:12, ACM, November 2009.

[15] G.-M. Chiu, "The odd-even turn model for adaptive routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729–738, 2000.

[16] M. Galles, "Spider: a high-speed network interconnect," *IEEE Micro*, vol. 17, no. 1, pp. 34–39, 1997.

[17] L. S. Peh and W. J. Dally, "Delay model and speculative architecture for pipelined routers," in *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, pp. 255–266, October 2000.

[18] A. Kumar, L. S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: towards the ideal interconnection fabric," in *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*, pp. 150–161, ACM, June 2007.

[19] D. Park, R. Das, C. Nicopoulos et al., "Design of a dynamic priority-based fast path architecture for on-chip interconnects," in *Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects*, pp. 15–20, August 2007.

[20] J. Kim, C. Nicopoulos, D. Park, V. Narayanan, M. S. Yousif, and C. R. Das, "A gracefully degrading and energy-efficient modular router architecture for on-chip networks," in *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA '06)*, pp. 4–15, IEEE, Boston, Mass, USA, June 2006.

[21] G. Dimitrakopoulos, N. Chrysos, and K. Galanopoulos, "Fast arbiters for onchip network switches," in *Proceedings of the International Conference on Computer Design (ICCD' 08)*, pp. 664–670, IEEE, 2008.

[22] V. Soteriou, H. Wang, and L.-S. Peh, "A statistical traffic model for on-chip interconnection networks," in *Proceedings of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS '06)*, pp. 104–116, IEEE, September 2006.

[23] J. Kim, D. Park, T. Theocharides, N. Vijaykrishnan, and C. R. Das, "A low latency router supporting adaptivity for on-chip interconnects," in *Proceedings of the 42nd Design Automation Conference (DAC '05)*, pp. 559–564, ACM, June 2005.

[24] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in *Proceedings of the IEEE 14th International Symposium on High Performance Computer Architecture (HPCA '08)*, pp. 203–214, IEEE, Salt Lake City, Utah, USA, February 2008.

[25] N. Najib, A. Monemi, and M. N. Marsono, "Partially adaptive look-ahead routing for low latency network-on-chip," in *IEEE Student Conference on Research and Development (SCOReD '14)*, 2014.

[26] T. Ahonen and J. Nurmi, "Synthesizable switching logic for network-on-chip designs on 90 nm technologies," in *Proceedings of the International Conference on IP Based SoC Design (IP-SOC '06)*, pp. 6–7, 2006.

[27] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni, "Virtual channels and multiple physical networks: two alternatives to improve NoC performance," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 12, pp. 1906–1919, 2013.

[28] W. Snyder, P. Wasson, and D. Galbi, *Verilator—Convert Verilog code to C++/SystemC*, 2014, http://www.veripool.org/wiki/verilator.