Small Virtual Channel Routers on FPGAs Through Block RAM Sharing

Jimmy Kwa, Tor M. Aamodt
ECE Department, University of British Columbia
Vancouver, Canada
jkwa@ece.ubc.ca
aamodt@ece.ubc.ca

***Abstract*—As larger System-on-Chip (SoC) designs are attempted on Field Programmable Gate Arrays (FPGAs), the need for a low cost and high performance Network-on-Chip (NoC) grows. Virtual Channel (VC) routers provide desirable traits for an NoC such as higher throughput and deadlock prevention but at significant resource cost when implemented on an FPGA. This paper presents an FPGA specific optimization to reduce resource utilization. We propose sharing Block RAMs between multiple router ports to store the high logic resource consuming virtual channel buffers and present BRS (Block RAM Split), a router architecture that implements the proposed optimization. We evaluate the performance of the modifications using synthetic traffic patterns on mesh and torus networks and synthesize the NoCs to determine overall resource usage and maximum clock frequency. We find that the additional logic to support sharing Block RAMs has little impact on Adaptive Logic Module (ALM) usage in designs that currently use Block RAMs while at the same time decreasing Block RAM usage by as much as 40%. In comparison to designs that do not use Block RAMs, a 71% reduction in ALM usage is shown to be possible. This resource reduction comes at the cost of a 15% reduction in the saturation throughput for uniform random traffic and a 50% decrease in the worst case neighbour traffic pattern on a mesh network. The throughput penalty from the neighbour traffic pattern can be reduced to 3% if a torus network is used. In all cases, there is little change in network latency at low load. BRS is capable of running at 161.71 MHz which is a decrease of only 4% from the base virtual channel router design.**

I. Introduction

As the size of Field Programmable Gate Arrays (FPGAs) grow, so does the desire and ability to implement larger FPGA based System-on-Chip (SoC) designs. This in turn has resulted in an increasingly large number of cores that need a low cost and high performance method of communicating among each other. Previously proposed methods of connecting the various cores included using a shared bus or directly connecting each component to every component it communicates with. The desire for better scalability and performance has led researchers to propose using a Network-on-Chip (NoC) to provide the communication network [1]. An NoC provides a highly scalable structure that facilitates simultaneous communication among network nodes [1]. In this paper, we study NoCs composed of routers that are used to pass messages between two nodes. NoC designers want NoCs that use a minimal number of resources while providing as much performance as possible where performance refers to lower network latency and higher throughput. Previous studies have examined using FPGAs to simulate Application-Specific Integrated Circuit (ASIC) NoCs [2] or have attempted to directly use an NoC designed for ASICs on an FPGA [3]. However, designing an NoC for use on FPGAs presents its own unique issues and is a far less studied area compared to ASIC NoCs. An NoC optimized for an ASIC system will not necessarily be an efficient design on an FPGA and ASIC optimizations will not necessarily improve an FPGA NoC [3].

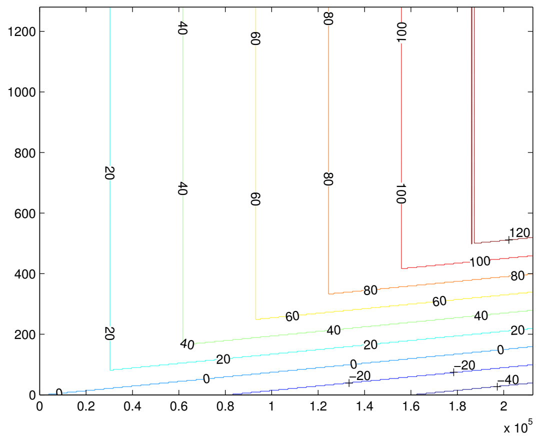


Fig. 1. Contour plot of the difference in maximum node count between BRS and CONNECT across different combinations of free M9Ks and ALMs. Positive values mean more BRS nodes.

One particular NoC component that is problematic to implement on an FPGA is a Virtual Channel (VC) router. A simple router without VCs can suffer from head-of-line blocking which harms the NoC's throughput. Head-of-line blocking is when the packet at the head of a queue stalls and the packets behind it are also blocked as a result. VCs allow for packets to go around other packets that have stalled [4]. Additionally, VCs are required to prevent deadlock in some routing algorithms [5]. The class of routing algorithms that require VCs to prevent deadlock include the well known and popular torus network based dimension-ordered routing [6]. While VCs are a common feature in routers for ASIC NoCs, efficient implementation in an FPGA NoC has been a challenge. A VC router can end up using up to five times as many resources as a router without VCs [7]. A significant part of the resource usage problem with VCs comes from the VC buffers [3]. VC buffers cannot use an FPGA's Distributed RAMs or Block RAMs in an efficient manner [3]. More details regarding this problem are provided in Section III.

The inefficiency in Block RAM usage for VC buffers is the key problem targeted by this work. We present the BRS (Block RAM Split) router architecture which addresses this problem by customizing the router design so the VC buffers can be implemented using a minimal number of Block RAMs while simultaneously freeing a large quantity of the Adaptive Logic Modules (ALMs) for use on user logic. Primarily this is achieved through a method of sharing a single Block RAM between the VC buffers of two router ports. VC buffer management is also modified to accommodate the additional limitations resulting from sharing a Block RAM.

Fig. 1 shows how many more nodes can be synthesized when using a router architecture that uses Block RAMs compared to an architecture that does not. The graph was produced by extrapolating the resource data from Table I and Table III which will be discussed later in the paper. It shows how the BRS router compares against CONNECT [3], a design that does not use Block RAMs, under different combinations of resources. The bounds of the graph are set to match the number of M9Ks and ALMs on the FPGA board we were targeting which are 1280 and 212480 respectively. CONNECT does better when there are only a few Block RAMs available for use in the NoC. However, the minimum number of Block RAM required before a larger number of BRS routers can be synthesized is low compared to the total number of Block RAMs and beyond this point many more BRS routers can be synthesized relative to CONNECT.

Our results show that compared to a router that uses Block RAMs we achieve a 40% reduction in Block RAM usage for a torus NoC. At the same time, performance is unaffected at lower loads. When compared to a router that does not implement VC buffers in Block RAMs, a 71% reduction in ALM usage is shown. Also, our results show only a 4% decrease in clock frequency relative to the base virtual channel router design.

The remainder of the paper is structured as follows. Section II covers previous work related to NoCs on FPGAs and prior attempts at implementing a VC router. Section III reviews the design of a traditional VC router and details the problems with implementing the design on an FPGA. Section IV contains the proposed architectural changes to the VC router to optimize the design for FPGAs. Section V evaluates the effects on resource usage, network latency and clock speed of the proposed optimization before concluding in Section VI.

II. Related Work

Previous work has covered several topics related to our study. They primarily cover alternate FPGA router optimizations and methods of predicting NoC performance.

CONNECT [3] is one of the few previous works that attempts to present an FPGA optimized NoC that can use VC routers. CONNECT is an NoC generator that produces NoCs with highly parameterized single stage routers. CONNECT identifies several differences between ASIC NoC and FPGA NoC design such as the limitation in data storage. CONNECT highlights Block RAMs as a scarce resource that is hard to use efficiently and CONNECT avoids them completely in favour of using Distributed RAMs to implement the VC buffers. BRS takes a different path than CONNECT and explicitly targets using Block RAMs to significantly reduce usage of logic resources. RASoC [8] and LiPaR [9] also present routers optimized for FPGAs but they do not support virtual channels.

Lu et al. [10] try to optimize a VC router through changes to the VC buffer organization and allocator design. They use a memory block per router port to hold the VC buffers associated with the port. Normally the wide data lines to a memory block need to run through a multiplexer and multiplexers do not map to FPGA logic blocks very well. Instead they devise a system where the narrower memory block address lines are multiplexed. Our proposed optimizations differs in that we combine the VC buffers from two ports into a single memory block implemented using a single Block RAM. Lu et al.'s parallel VC/switch allocator optimization is orthogonal to our proposed optimizations and can be used to further reduce resource usage.

Targeting specific communication patterns is another approach to optimizing a router. By targeting a master-slave communication pattern, Leary et al. [11] are able to decrease resource usage and network latency while improving throughput. Singh et al. [12] generate multiple NoC configurations that are loaded at runtime to match the application which takes advantage of the reconfigurability of the FPGA.

There is also work looking at the performance of NoCs implemented on an FPGA. Shannon et al. [13] look at creating a model for predicting the performance of NoCs that have been implemented on an FPGA over a variety of topologies and FPGAs.

GCQ [14] proposes an FPGA based switch implementation that uses an alternate method of sharing Block RAMs among multiple data queues. Their work targets a Combined Input and Crosspoint Queued switch architecture and shares Block RAM among multiple crosspoint buffers in a large crossbar. They run the Block RAM at a higher clock frequency than the rest of the switch and time multiplex accesses to it. While their work targets the crosspoint buffers, our work focuses on reducing the size of the input buffers. Both optimizations can potentially be used in the same design.

Saldana et al. [15] evaluate several different FPGA NoC topologies in terms of resource usage and speed. They conclude that for networks with as few as 16 nodes, network topology usually has little effect on resource usage and for a resource and speed penalty a fully connected network is possible. However, their work shows that as the number of nodes increase and the NoC approaches the limits of the

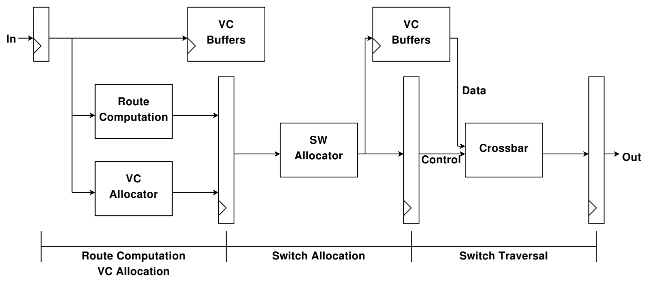


Fig. 2. Three stage VC router pipeline.

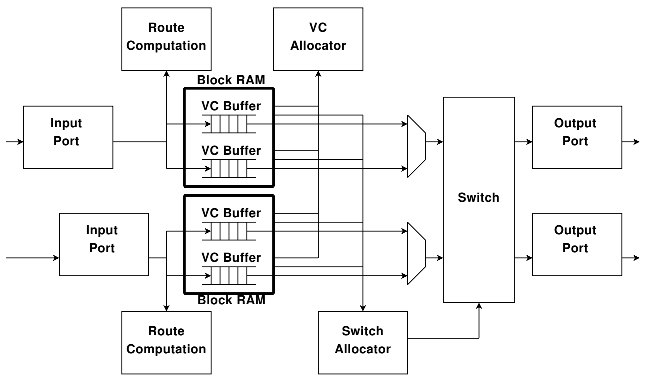


Fig. 3. Traditional VC router architecture. Figure drawn with only two input and output ports for clarity – we use 4 port routers in our evaluation in Section 5.

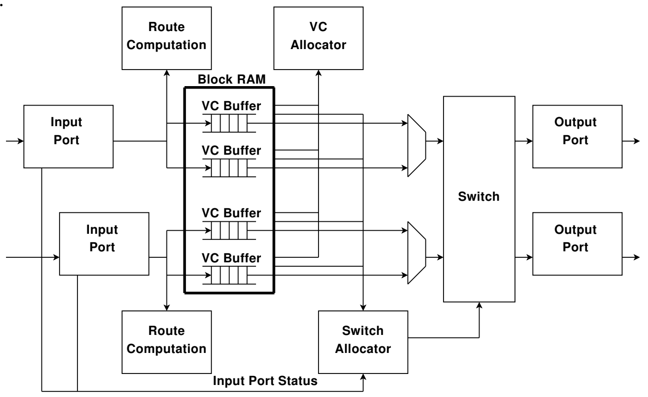


Fig. 4. BRS architecture. Figure drawn with only two input and output ports for clarity – we use 4 port routers in our evaluation in Section 5.

routing capacity of the FPGA, mesh and hypercube topologies perform best. Our proposed modifications do not affect the wiring between routers and can be used to reduce Block RAM usage for larger mesh networks.

Francis et al. [16] propose using an FPGA architecture with Time Division Multiplexed wiring and hard routers to reduce the size and improve performance of the NoC. BRS does not require either of these resources to function. Kapre et al. [17] study a time multiplexed FPGA NoC while our work targets improving a packet-switched NoC.

III. Traditional VC Router Architecture

VC routers were introduced to improve the throughput of an NoC [4] and allow a greater variety of routing algorithms to run deadlock-free [5]. However, the large amount of FPGA resources consumed by the traditional VC router architecture [7] is a barrier to more widespread adoption of VC routers on FPGA.

The traditional VC routing process is broken into four pipelined stages [6]. These stages include Routing Computation, Virtual Channel Allocation, Switch Allocation and Switch Traversal. Each input port of the router has an associated input unit that contains the input port's VC buffers and VC state information. The router also contains a VC allocator and switch allocator to manage access to the output VCs and reserve cycles on the switch. To reduce the number of clock cycles to traverse the router, the number of pipeline stages is reduced to three through lookahead routing computation [18]. Lookahead routing computation means performing the destination port calculation one router hop in advance. Destination port calculation for the next router and VC Allocation for the current router are independent from each other and can be performed in parallel. An example of the resulting three stage pipeline can be seen in Fig. 2. While other well known speculation based optimizations exist that allow for other stages to be performed in parallel [19], our router architecture only uses lookahead routing computation.

The router architecture is similar to the one shown in Fig. 3. A flit is sent by an upstream router or a node attached to the local input port. At the input port, the flit is stored for one cycle while the router determines which VC buffer it needs to be steered towards. Once this is determined, the flit is passed to the correct VC buffer. At the same time, the incoming flit is checked to see if it is the first flit in a packet which is known as the head flit. The head flit contains additional information such as the address of the final destination for the flit. This information is passed on to the Route Computation block. The Route Computation block performs the lookahead routing computation [18] to calculate the flit's desired output port at the next downstream router. This is done so the route computation can be done in parallel with VC allocation which depends on the information from route computation. The VC allocation stage attempts to allocate an output VC to each head flit received by the downstream router. Upon success, the allocation result is recorded and used to direct the remaining flits in the packet to the same output VC. When the Route Computation block processes a head flit, the flit is tagged with the output port the next router should use to send the flit. At the next router, this information tag is read and a request for a VC associated with that output port is sent to the VC allocator. When possible, the VC allocator returns a grant signal to the head flit's VC which records the successful allocation. Each VC buffer on each input port that is holding a head flit that won VC allocation or a non-head flit that follows a head flit that won VC allocation, sends a request to reserve cycles on the switch to the switch allocator. Each output port can only accept a flit from one VC across all input VCs per cycle. Also only one VC from each input port can send a flit to an output port per cycle. The switch allocator enforces this constraint by only giving a grant to one VC per input port and granting access to at most one input port to an output port. There are multiple ways to implement the VC and switch allocators. The different methods vary in size, performance and maximum clock speed. We choose to use separable input first allocators that are comprised of multiple round robin arbiters. Their benefit is small size and high clock speed. When a VC buffer is given a grant by the Switch Allocator, during the next cycle the flit at the front of the buffer is removed and sent into the switch to be moved to the output port. Once received by the output port, during the next cycle the flit is sent across the link towards the next downstream router. To track the amount of free buffer space at a downstream router, a credit system is used. An output port starts with a full set of credits which is decremented as it sends out flits. When the output port runs out of credits, it knows the downstream router can no longer accept new flits. As buffer space opens up on the downstream router, credits are sent back upstream to inform it of the new free space. This tracking is done on a per input VC basis so each buffer has its own credit count.

The VC buffers are problematic to implement on an FPGA due to inefficient mapping to the storage resources [3]. On an FPGA, storage comes from using logic resources or Block RAMs. Xilinx's Distributed RAM uses Lookup Tables (LUTs) as storage instead of for logic while Altera uses register bits inside of ALMs. Block RAMs are larger storage units that cannot be subdivided and are far less abundant than LUTs or ALMs on an FPGA. Implementing the VC buffers by using logic resources will use up a significant amount of LUTs or ALMs with the problem getting worse at larger VC counts or buffer depth. On the other hand, VC buffers are small relative to the storage capacity of a Block RAM. Using Block RAMs to store the VC buffers will use up a highly limited resource and a large part of the reserved Block RAM will remain unused due to the inability to subdivide them.

IV. BRS Architecture

Our work presents a new and simple method of implementing and managing the VC buffers to reduce the amount of resources consumed while limiting the impact on performance. The key to our work is a method of sharing the space inside a Block RAM between the flit data portion of the VC buffers associated with two of the router input ports. This change can be seen in Fig. 4 where the VC buffers for two input ports are now held within a single Block RAM as opposed to Fig. 3 where they are separate memory blocks that could be implemented in either Block RAMs or logic resources. Flit header information is small and is required to be accessed more often than the flit data so it is still stored in logic resources.

We designed the BRS architecture to target an FPGA that uses Altera M9K Block RAMs but the idea can be extended to other Block RAMs that support true dual port access. The M9K memory block comes with several limitations that must be addressed before they can be used to store the desired VC buffers. The primary issue is the number of ports on the Block RAMs. The M9K Block RAMs have two read/write ports. On any given cycle, a port can be used to either read or write to a memory location in the Block RAM. A single port can also be used to read and write to the same memory location during the same cycle. In our work we configure the Block RAM such that a read and write during the same clock cycle will return the new data if the same Block RAM port is used and the old data if different Block RAM ports are used. For Block RAMs that do not support this configuration, additional logic will be required to read the new data during a simultaneous read and write. In the baseline design, a VC buffer can both be written to and read from during the same cycle if a flit arrives at the input port at the same time the VC is sending a flit through the switch to an output port. Since two ports will be sharing a single Block RAM, this can result in accesses to as many as four different memory locations during the same cycle. This behaviour is not compatible with Block RAMs which can access at most two different locations at the same time. The second issue with the Block RAMs is the limitation on the data width and VC count and depth. While operating as a true dual port memory, the M9K Block RAMs have a maximum data width of 18 bits and can hold 512 entries of data of this size.

To address the issue of a limited number of Block RAM ports, modifications to the VC buffer management were required. Situations where more than two locations in a Block RAM were accessed needed to be avoided. This required implementing a form of stalling when a write or read to the Block RAM could not be performed during the current cycle and ensuring that the write or read is performed in the future. The router only writes to a Block RAM when a new valid flit arrives at an input port. Reads from a Block RAM only occur when a VC on one of the router ports sharing the Block RAM is able to win switch allocation. The lookahead route information is read and recorded when a head flit initially enters the VC buffer and does not need to be read out of a Block RAM. This reduces resource usage but comes at the expense of preventing VC reassignment until an entire packet has been sent.

To stall writes to a Block RAM, information would need to be passed to the upstream router to indicate a flit was received but dropped and needs to be resent or to block the flit from being sent in the first place. In the interest of keeping the credit logic simple, we opted to give priority to Block RAM write attempts so they are never stalled. The potential downside of this is an increase in network latency from congestion. A router with several flits backed up in its VC buffers may have to hold a normally sendable flit for an extra cycle to receive a flit from an upstream router that currently has nearly empty VC buffers. At most two new flits can arrive on the two input ports sharing a Block RAM during a single cycle. The router ports can use the two ports on the Block RAM to write both flits into their respective VC buffer. This way if an upstream router has credits available, it is able to send a flit with a guarantee that the downstream router has buffer space and will accept the flit.

Since Block RAM writes are given priority, the VC buffer management logic needs to stall VC buffer reads when both Block RAM ports are used for writes. This stalling is performed by canceling requests to the Switch Allocator. This is done to prevent a VC from reserving time on the switch that it cannot use and consequently resulting in unnecessarily preventing another VC from using the switch during the next cycle. There are several different combinations of potential read and write requests to a Block RAM that need to be handled in different ways. The straightforward cases are when either both or neither of the ports sharing a Block RAM receive a new flit. When both router input ports receive a new flit, two memory locations need to be accessed inside the Block RAM. This automatically ties up both ports on the Block RAM so all switch requests from any of the VCs from either router port need to be squashed. The case where neither port receives a new flit leaves both Block RAM ports free. This allows the VCs for both router ports to send their requests to the switch allocator. The case where only one of the router input ports receives a new flit is a bit more complex. The incoming flit will tie up one of the Block RAM's ports which means only one of the two router ports can read their flit data from the Block RAM. Priority for sending their requests to the switch allocator is given to the port that is not receiving a new flit. The switch requests of the router port receiving a new flit are masked off unless the other port sharing the Block RAM does not have any switch requests to make. This port selection can sometimes lead to unnecessary stalling when the port selected loses switch allocation and the port that has its requests canceled would have won. However, this avoids the need for additional Switch Allocators that would be required to redo switch allocation for the other port during the same cycle.

The limited data width on the Block RAMs creates a limitation on channel width of the VC router. The data width of flits passing through the network is limited to the maximum data width of a Block RAM which is 18 bits for the M9K Block RAM. Increasing the data width beyond this would force the need for multiple Block RAMs to hold the VC buffers of two router ports. The limited capacity of a single Block RAM also creates a limitation on the number of VCs and the depth of their buffers. If the number of VCs multiplied by the depth of the buffer and doubled to account for sharing between two ports exceeds 512, additional Block RAMs are once again required to hold the VC buffers.

The ports are chosen to share a Block RAM for their VC buffers through a configurable parameter. Different pairings can lead to different effects on the performance of the router. Ports that frequently receive or send flits together should be separated into different pairs.

V. Experimental Results

Our work was builds on top of the Booksim Open Source Network-on-Chip Router RTL [20]. We generate mesh and torus networks with BRS routers to demonstrate the effect on performance and synthesize the designs to determine the effect on resource usage. Synthesis is done using the commercial tool Quartus II and targets an Altera DE4 FPGA board (EP4SGX530KH40C2). Simulation of the design is performed using ModelSim-Altera. Synthetic traffic patterns that inject 4 flit long packets is used. The routers are set to have 2 VCs per port with VC buffer depth of 16. For the routing algorithm, XY dimension ordered routing is used. In our router design, the East and West ports on the router share a single Block RAM and the North and South ports share a Block RAM. There is only one local port per router and it is attached to a packet generator. The local port has its own non-shared Block RAM for its VC buffers. In the interest of reducing synthesis and simulation runtime length, our experiments are performed on 4x4 networks. However, our modifications can also be applied to routers in larger networks and we expect our results to hold for larger networks as well unless indicated otherwise.

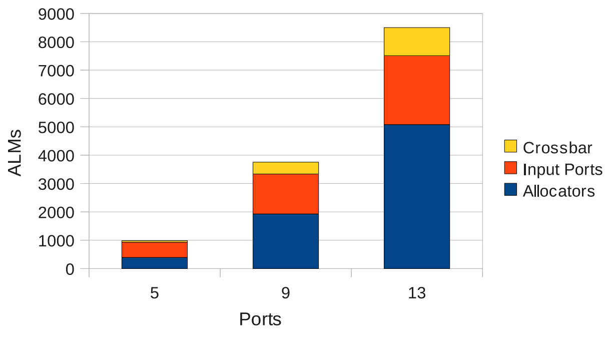


Fig. 5. ALM Usage as router port count increases.

TABLE I
4x4 Mesh Resource Utilization.

	Booksim	BRS	% Diff
Centre Rtr (ALM)	1042	1121	7.59
Total NoC (ALM)	12877	13237	2.80
Centre Rtr (M9K)	5	3	-40.00
Total NoC (M9K)	64	48	-25.00

A. *Resource Utilization*

Higher degree topologies are possible on FPGA and can result in lower NoC diameter and less routing delay [15]. However, this benefit needs to be weighed against the cost of the higher radix routers it requires. Fig. 5 shows how the components of the Booksim router we base our work off grow in size as the number of ports increase. The VC and switch allocator components of the router grow quadratically. In the rest of this study we focus on mesh and torus topologies which have lower area cost at the potential expense of higher network latency.

The synthesis results from Quartus II in Table I show the average resource usage of the four routers in the center of the 4x4 mesh NoC and the total resources used by the entire NoC. This data indicates that the additional logic to support sharing a Block RAM between two ports has low impact on the number of ALMs used to generate the NoC. The NoC with our

TABLE II
4x4 Torus Resource Utilization.

4x4 Torus	Booksim	BRS	% Diff
ALM Usage (Avg)	1050	1105	5.29
ALM Usage (Total)	16504	17383	5.33
M9K Usage (Avg)	5	3	-40
M9K Usage (Total)	80	48	-40

modified routers only uses 2.80% more ALMs than the NoC without modifications. The difference in M9K Block RAM usage is much more significant with usage decreases of up to 40%. In the original Booksim router, edge and corner routers use fewer Block RAMs due to unused ports being optimized out. As a result, these routers do not benefit as much from our router modification. However, edge and corner routers become a smaller factor relative to the centre routers as the mesh NoC is scaled up to higher core counts. The net result on the 4x4 mesh NoC is a 25% reduction in Block RAM usage with low change in ALM usage. When these results are extrapolated they show an estimated 30% reduction in Block RAM usage in mesh NoC as small as 6x6.

We synthesize a 4x4 torus NoC with the same parameters as the 4x4 mesh NoC. The synthesis results from the torus NoC are shown in Table II. The results show strong similarities to the synthesis results of the centre routers in the 4x4 mesh. As a torus network has no edges or corners, every router hits the 40% decrease in M9K usage that only the centre routers in the mesh NoC achieved. Although this comes along with the same ALM increase the centre router in the mesh NoC had, it is ultimately only a slightly larger than 5% increase in ALM usage.

Table III displays a comparison between BRS and an FPGA optimized NoC generated by CONNECT [3]. Unlike BRS which uses Block RAMs, CONNECT generates an NoC that avoids using Block RAMs in the interest of leaving them for the user logic. To follow our implemented 4x4 mesh design as closely as possible, the CONNECT NoC generated is also a 4x4 mesh. The CONNECT NoC is set to have 2 VCs per input port and each VC has a VC buffer with a depth of 16. For the allocator, a separable input-first round-robin allocator is used and the flit data width is set to 18.

The data in Table III is taken from synthesizing the CONNECT NoC in Quartus II. The CONNECT VC Queue column displays the number of ALMs allocated for use in implementing the VC buffer storage for routers at the centre of the mesh NoC. By comparing this against the total number of ALMs used to implement the complete router, it shows that a huge percentage of the ALMs used by CONNECT are being allocated for use as VC buffer storage. At the cost of the additional Block RAMs to implement the design and any effects on performance, our router has ALM usage that is lower by 71%. The ALM usage decrease in the centre routers is almost exactly in line with the overall decrease in ALM usage which shows that larger networks, that have a larger percentage of centre routers, will maintain the 71% decrease

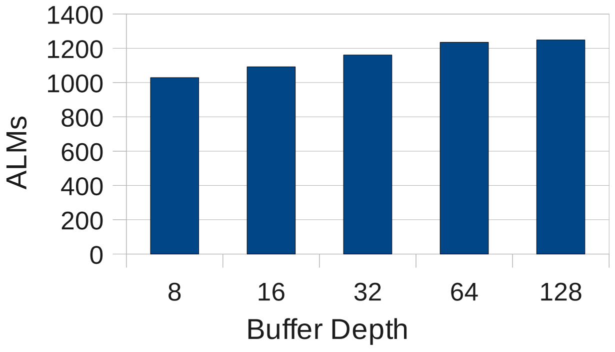


Fig. 6. ALM usage of BRS with varying VC buffer sizes. All shown configurations also use 3 M9K Block RAM.

TABLE IV
Maximum frequency for 4x4 networks.

Max freq (MHz)	4x4 Mesh	4x4 Torus
CONNECT	72.37	65.58
BRS	161.71	149.43
Booksim	167.31	154.20

in ALM usage.

The number of bits used in a Block RAM can be calculated by multiplying together the number of ports sharing the Block RAM, VCs per port, depth of the VC buffers and width of the VC buffers. Using our presented configuration, at most 1152 out of 9216 bits in a M9K Block RAM are occupied at the same time and the rest of the space is unused. The percentage of the Block RAM that is used can be increased if the desired NoC can take advantage of deeper VC buffers or a larger number of VCs. Fig. 6 shows how VC buffer sizes affect ALM usage in BRS. Due to using Block RAMs to store the VC Queue, significantly larger VC queues can be used in exchange for a small increase in ALM usage and no increase in Block RAM usage. However, in the cases where larger VC buffers are not useful it would be beneficial for an NoC to have access to smaller Block RAMs on the FPGA. This would increase the ratio of I/O bandwidth to memory capacity to better match the demands of the VC buffers.

B. *Timing*

We used Altera's TimeQuest Timing Analyzer to find the maximum clock frequency for BRS and compared it against Booksim and CONNECT. The results are shown in Table IV. CONNECT trades off a higher clock speed for fewer router pipeline stages so it has the lowest clock frequency. Booksim can run at 2.3 times the clock frequency of the CONNECT NoC. BRS has a maximum clock frequency of only 4% less than Booksim and remains 2.2 times higher than CONNECT.

C. *Performance Analysis*

The graphs in Fig. 7 show how our router modifications affect network latency under different synthetic traffic patterns. Synthetic traffic patterns are used to show how a network

TABLE III
BRS VS CONNECT. A 4X4 MESH ALM UTILIZATION COMPARISON.

ALM Usage	CONNECT [3]	CONNECT VC Queues [3]	% VC Queue [3]	**BRS**	**% Difference**
Centre Router	3965	2665	67.22	**1121**	**-71.74**
Total NoC	46091	33761	73.25	**13237**	**-71.28**

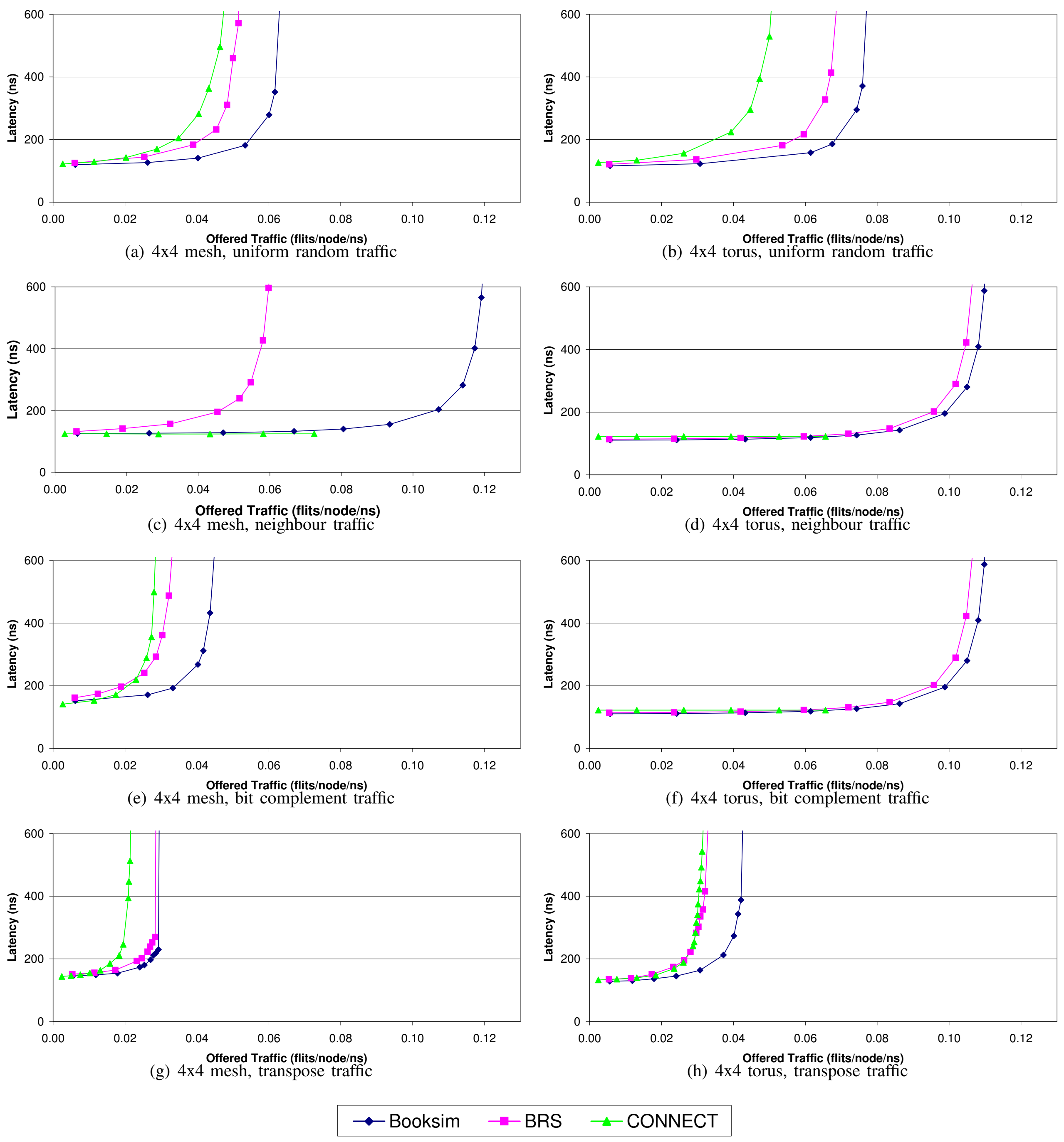
responds to different packet injection patterns. Many of these synthetic patterns represent communication patterns found in real applications. After a 10000 cycle warmup period, packets entering the network for the next 10000 cycles were tagged as a measurement packet and marked with the cycle number the head flit entered the source queue. For these packets, network latency is measured as the difference between the cycle count tagged on the head flit and the cycle count the tail flit leaves the network. The simulation is run until all marked packets have left the network and the average latency of all marked packages is recorded. To account for the different cycle length of the different designs, the data is normalized to a nanosecond scale instead of cycles. The clock frequency data can be seen in Table IV. We sweep across a range of offered traffic to compare the latency of each network and their saturation throughput.

From Fig. 7(a) it can be seen that the proposed modifications result in a 15% decrease in the saturation throughput for a 4x4 mesh NoC running uniform random traffic. The difference in performance comes from the new stalling conditions introduced by the limited port count on the shared Block RAMs. At lower injection rates, the effect on packet latency is nearly non existent. Lower injection rates lead to lower traffic on the network which in turn reduces the likelihood of two ports sharing a Block RAM both receiving a flit during the same cycle which would stall reading flit data to be sent out of the VC buffers. Despite the decrease in saturation throughput, the network with our routers still beats the network generated by CONNECT by 10%. On a 4x4 torus network, BRS performs even better against the Booksim router and CONNECT. As shown in Fig. 7(b) the modified routers now only reduce saturation throughput by 10% relative to Booksim and does 40% better than CONNECT. Using a torus network instead of a mesh network reduces the average number of hops from source to destination and increases the number of links in the network the traffic is spread over. These factors reduce link utilization for equal levels of offered traffic which in turns reduces the likelihood of the new stalling conditions being triggered.

Traffic patterns other than uniform random can result in different effects on performance. In particular the neighbour traffic pattern can lead to two different extremes depending on whether the network is a mesh network or a torus network. This effect can be seen in Fig. 7(c) which shows the mesh network case and Fig. 7(d) which shows the torus network case. Neighbour traffic on a mesh network is a worst case scenario for our routers while neighbour traffic on a torus network is one of the best cases. Under the neighbour traffic pattern each node sends a message to the node one hop to the right and one hop up. On both mesh and torus topologies, the input ports on a router never compete for the same output port. This is a highly favourable traffic pattern for the Booksim router and the routers generated by CONNECT and allows these routers to consistently receive and send flits on every port each cycle. Even at a 100% injection rate, represented by the far right point on CONNECT's lines in the graphs, the CONNECT NoC's latency remains constant. In the case of the Booksim router, constant latency all the way to 100% is not quite achievable due to having to wait for a VC buffer to be empty before reusing it. While our routers also benefit from the neighbour traffic pattern relative to uniform random traffic, sharing a Block RAM between the VCs of two ports means our router cannot consistently send and receive flits on all ports. This prevents our routers from improving as much as the other two designs. However, at lower levels of offered traffic, BRS has similar latency to the other two designs and near 100% injection rate is an extreme situation that is unlikely to occur.

In contrast to when it is run on a mesh, when neighbour traffic in run on a torus network, it turns out to be highly beneficial to BRS. When neighbour traffic is run on a torus, only West, South and Local input ports, which use three different Block RAMs, receive flits each cycle. The result is the new stalling conditions are never triggered. Cycle for cycle, our modified router matches the Booksim design but when normalized to nanoseconds the performance is around 3% worse due to a lower clock frequency. For the neighbour traffic pattern on a torus, the CONNECT NoC is able to achieve a 100% offered traffic rate. However, due to the difference in clock frequency it has a saturation throughput around 40% worse than our modified routers.

Fig. 7(e) and Fig. 7(f) show how the different networks respond to a bit complement traffic pattern while Fig. 7(g) and Fig. 7(h) show the response to a transpose traffic pattern. On a mesh running bit complement traffic, our routers have a 30% lower saturation throughput compared to Booksim but is nearly 15% better than CONNECT. On a torus running bit complement traffic, the different networks perform similarly to neighbour traffic. In the case of transpose traffic, our router nearly matches Booksim in terms of performance. In the transpose traffic pattern, no router receives data from more than 2 input ports. In only 2 of the 16 routers in the 4x4 mesh do the 2 input ports share a single Block RAM. The result of this is the new stalling conditions are rarely triggered. The saturation throughput of our router network in this traffic pattern is only 3% worse than the Booksim router network and 30% higher than the CONNECT network.



(a) 4x4 mesh, uniform random traffic
(b) 4x4 torus, uniform random traffic
(c) 4x4 mesh, neighbour traffic
(d) 4x4 torus, neighbour traffic
(e) 4x4 mesh, bit complement traffic
(f) 4x4 torus, bit complement traffic
(g) 4x4 mesh, transpose traffic
(h) 4x4 torus, transpose traffic

Fig. 7. Network Latency vs Offered Traffic for multiple topologies and traffic patterns. Lower latency means better performance.

VI. Conclusion

This paper presented a novel approach to improving the efficiency of using Block RAMs to reduce the size of a VC Router implementation on FPGAs and the BRS router architecture that uses the proposed optimization. We discussed the problem of VC buffer implementation and how they have a large impact on the ALM usage of a router and how a Block RAM implementation can be used to reduce this impact. We present a method of modifying the router to share Block RAMs between the VC buffers of two ports to minimize the Block RAM usage of BRS. The BRS routers were both synthesized and simulated using 4x4 mesh and 4x4 torus NoCs. The resource usage results displayed a 25% decrease in Block RAM usage and comparable ALM usage to a base router design that implements their VC buffers in Block RAMs. In comparison to a router that did not use Block RAMs, a 71% reduction in ALM usage was shown. In terms of performance, there was only a 15% decrease in saturation throughput when running uniform random traffic and 50% for neighbour traffic on a mesh which can be reduced to 3% if run on a torus. The effect on maximum clock frequency was a minor 4% decrease relative to the original design.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceedings of the 38th Annual Design Automation Conference*, Jun. 2001, pp. 684–689.

[2] B. Neji, Y. Aydi, R. Ben-atitallah, S. Meftaly, M. Abid, and J.-L. Dykeyser, "Multistage interconnection network for MPSoC: Performances study and prototyping on FPGA," in *3rd Int'l Design and Test Workshop*, Dec. 2008, pp. 11–16.

[3] M. K. Papamichael and J. C. Hoe, "CONNECT: Re-examining conventional wisdom for designing NoCs in the context of FPGAs," in *Int'l Symposium on Field Programmable Gate Arrays*, Feb. 2012, pp. 37–46.

[4] W. J. Dally, "Virtual-channel flow control," in *17th Int'l Symposium on Computer Architecture*, Jun. 1990, pp. 60–68.

[5] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," in *IEEE Transactions on Computers*, May 1987, pp. 547–553.

[6] W. J. Dally and B. Towles, in *Principles and Practices of Interconnection Networks*, 2003.

[7] G. Schelle and D. Grunwald, "Exploring FPGA network on chip implementations across various application and network loads," in *Int'l Conference on Field Programmable Logic and Applications.*, Sep. 2008, pp. 41–46.

[8] C. A. Zeferino, M. E. Kreutz, and A. A. Susin, "RASoC: A router softcore for Networks-on-Chip," in *Design, Automation and Test in Europe Conference and Exhibition*, Feb. 2004, pp. 198–203.

[9] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri, "LiPaR: A Light-Weight parallel router for FPGA-based Networks-on-Chip," in *Proceedings of the 15th ACM Great Lakes symposium on VLSI*, Apr. 2005, pp. 452–457.

[10] Y. Lu, J. McCanny, and S. Sezer, "Exploring virtual-channel architecture in FPGA based networks-on-chip," in *Int'l SOC Conference (SOCC)*, Sep. 2011, pp. 302–307.

[11] G. Leary, K. Mehta, and K. S. Chatha, "Performance and resource optimization of NoC router architecture for master and slave IP cores," in *5th IEEE/ACM/IFIP Int'l Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, 2007, pp. 155–160.

[12] A. Singh, A. Kumar, T. Srikanthan, and Y. Ha, "Mapping real-life applications on run-time reconfigurable NoC-based MPSoC on FPGA," in *Int'l Conference on Field-Programmable Technology (FPT)*, Dec. 2010, pp. 365–368.

[13] J. Lee and L. Shannon, "Predicting the performance of application-specific NoCs implemented on FPGAs," in *Int'l Symposium on Field Programmable Gate Arrays*, Feb. 2009, pp. 23–32.

[14] Z. Dai and J. Zhu, "Saturating the transceiver bandwidth: switch fabric design on FPGAs," in *Int'l Symposium on Field Programmable Gate Arrays*, Feb. 2012, pp. 67–76.

[15] M. Saldana, L. Shannon, and P. Chow, "The routability of multiprocessor network topologies in FPGAs," in *Int'l Workshop on System-Level Interconnect Prediction*, Mar. 2006, pp. 49–56.

[16] R. Francis and S. Moore, "Exploring hard and soft networks-on-chip for FPGAs," in *Int'l Conference on Field-Programmable Technology (FPT)*, Dec. 2008, pp. 261–264.

[17] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon, "Packet switched vs. time multiplexed FPGA overlay networks," in *IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Apr. 2006, pp. 205–216.

[18] M. Galles, "Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip," in *Symposium on Hot Interconnects*, Aug. 1996, pp. 141–146.

[19] L. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Int'l Symposium on High-Performance Computer Architecture*, Jan. 2001, pp. 255–266.

[20] D. U. Becker, "Open source Network-on-Chip router RTL," in *http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router*.