Comparative Evaluation of FPGA and ASIC Implementations of Bufferless and Buffered Routing Algorithms for On-Chip Networks

Yu Cai, Ken Mai and Onur Mutlu
Dept. of Electrical and Computer Engineering, Carnegie Mellon University
yucaicai@gmail.com, kenmai@ece.cmu.edu, onur@cmu.edu

Abstract— **Most existing packet-based on-chip networks assume routers have buffers to buffer packets at times of contention. Recently, deflection-based bufferless routing algorithms have been proposed as an alternative design to reduce the area, power, and complexity disadvantages associated with buffering in routers. While bufferless routing shows significant promise at an algorithmic level, these algorithms have not been shown to be efficiently implementable in practice. Neither were they extensively compared to existing buffered routing algorithms in realistic designs. This paper presents our comparative evaluation of and experiences with realistic FPGA and ASIC designs of state-of-the-art (1) virtual-channel buffered, (2) deflection-based bufferless, and (3) deflection-based buffered routing algorithms using two different network topologies and network sizes. We show that bufferless routing algorithms are implementable without significant complexity, and compare their performance, area, frequency, and power consumption to their buffered counterparts. Our results indicate that bufferless routing can lead to significant area (38%), power consumption (30%), and router cycle time (8%) reductions over the best buffered router implementation on 65nm ASIC design, while operating at higher frequency.**

I. INTRODUCTION

On-chip interconnection networks are envisioned to be the communication backbone between cores, caches, and memory controllers in a multi-core chip [9]. These networks consist of packet-switched routers that connect each node within the network where a node can consist of a core, a cache slice, a memory controller or a combination of them. As the number of cores on chip increases, the number of routers connecting the nodes increases proportionally. As such, designing efficient routers in terms of power, area, and performance becomes increasingly more desirable to keep the communication backbone efficient. Existing on-chip network designs and implementations [39, 23, 20, 42] were based on the assumption that routers need to buffer incoming packets before making a packet switching (also called routing or scheduling) decision. The purpose of buffering is to improve bandwidth-efficiency in the network at times of congestion: if two incoming packets need to go out of the same output port of the router, only one of them can proceed and the other is buffered to be transmitted at a future time. However, buffers also have several disadvantages: 1) they consume additional energy/power, 2) occupy significant on-chip network area, which was shown to be 75% in the TRIPS prototype chip [20], and 3) increase design complexity due to the management needed for their allocation and deallocation.

Recently, bufferless routing algorithms have been proposed to overcome these disadvantages [30, 27, 19, 18, 11, 12, 13, 14, 15, 2, 5, 28, 32, 33] of buffered routing. The basic idea of bufferless routing algorithms is to eliminate the input and output buffers in routers. When two incoming packets need to go out of the same output port, one of them is intentionally either (1) "misrouted" or "deflected" to an "undesirable" output port or (2) dropped and retransmitted, instead of being buffered. If contention is *not* common in the network, which was observed for most real applications run on on-chip networks [6, 24, 25, 30], then the performance of bufferless routing can be close to that of buffered routing [30, 11, 12, 13, 5]. Recent simulation-based research [30] has shown that bufferless routing provides significant energy consumption reduction over buffered routing while having similar performance using real applications. Hence, bufferless routing holds significant promise to simplify on-chip network design.

Unfortunately, previous research did not investigate *in detail* the efficient implementation (challenges and benefits) of bufferless routing in a real design. While bufferless routing eliminates buffers, it requires extra provisions to ensure that the network does not live-lock. In particular, the router needs to prioritize the oldest packet to the desired output port in order to guarantee forward progress [30, 12]. The complexity of oldest-first arbitration in routing can hinder the adoption of bufferless routing in practice [12]. In addition, bufferless routing causes 1) header information to be transmitted with each flit, and 2) increased occurrence of deflections and hence increased number of link traversals, both of which can lead to increased energy consumption. Finally, some previous research evaluated the area benefits of bufferless routing using back-of-the-envelope calculations. These three aspects of bufferless routing are very difficult to evaluate without a real implementation.

Our goal in this paper is to realistically and comprehensively evaluate the implementability, benefits, and disadvantages of bufferless routing in comparison to state-of-the-art buffered routing algorithms. In particular, we would like to perform area, power, frequency, performance, and complexity comparisons between buffered and bufferless routing algorithms using real, comparable implementations of each. We would also like to investigate possible logic-level optimizations enabled by the elimination of buffers and provide an experience report of the challenges encountered in the design of both bufferless and buffered routing algorithms.

To this end, we have implemented several highly-optimized versions of a deflection-based bufferless routing algorithm, BLESS [30], and state-of-the-art virtual-channel-based buffered routing algorithms on FPGA chips using the BEE2 board. This paper provides extensive comparisons of these implementations, pointing out the design challenges and power/performance, area/complexity advantages/disadvantages/limitations of each.

This work makes the following major contributions:

1. To our knowledge, we provide the first realistic and detailed implementation of deflection-based bufferless routing in on-chip networks and its extensive comparative evaluation with state-of-the-art buffered routing on the same FPGA platform and the same 65nm ASIC CMOS technology.

2. We show that bufferless routing is efficiently implementable in mesh and torus based on-chip networks, leading to significant area (38%), power consumption (30%) reductions over the *best buffered router implementation* on a 65nm ASIC design. Even on an FPGA design, BLESS leads to 24% area reduction and 18% power reduction. Bufferless routing has similar packet latency as buffered routing at low packet injection rates. As the path diversity in the network increases, the throughput of bufferless routing becomes closer to that of buffered routing.

3. We identify that oldest-first arbitration, which is used to guarantee livelock freedom of bufferless routing algorithms, as a key design challenge of bufferless router design. However, with careful design, oldest-first arbitration can be pipelined and can outperform virtual channel arbitration, enabling higher frequencies with bufferless routing than with buffered routing.

II. BACKGROUND

On-chip networks (NoCs) use packet-switched routers, which communicate information from one node to another (see Fig. 1). A packet is divided into flits, each of which acts as a flow-control unit. Each router has multiple input ports and multiple output ports. For example, a router in a 2-dimensional mesh network usually has 5 input ports (one for each of the incoming 4 directions and one from the resource it is connected to) and 5 output ports (one to each of the outgoing directions and one to the resource it is connected to). The function of a router is to determine the output port each flit needs to take and transmit each flit in a packet from the incoming input port to an outgoing input port. To accomplish this, the router can use buffering to buffer flits when multiple incoming flits need the same output port.

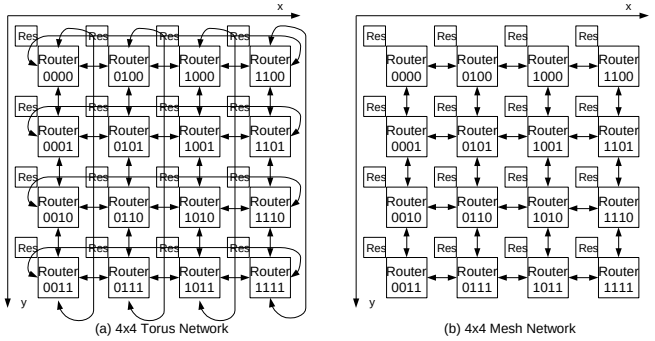

Fig. 1 Network topology examples: torus and mesh

A. Buffered Routing

Traditional routers in on-chip networks and off-chip networks have used buffers in input ports, output ports, or both (e.g., see Fig. 2). When a flit arrives in the router, it is buffered in a FIFO buffer associated with the corresponding input port. All flits at the head of the different buffers that need to access the same output port arbitrate for that output port. This arbitration usually prioritizes one input port over other in round robin order across different cycles and is commonly referred to as round-robin arbitration.

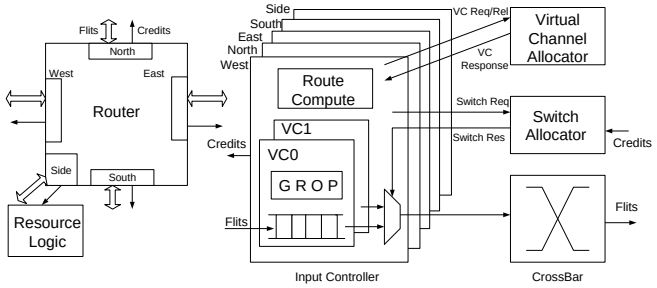

Fig. 2 Microarchitecture of the evaluated VC-buffered router

There have been several optimizations proposed for buffered routers that are implemented in existing systems. Wormhole routing [8] simplifies the routing of a whole packet by performing route computation only for the head flit. Only head flit contains packet header information. All remaining flits follow the route computed for the head flit (i.e. the output port assigned to the head flit) as if the later parts of the worm follow the head of the worm, instead of independently being assigned an output port. Virtual channels (VCs) [7] multiplex a single physical port between multiple virtual ports. The idea is to associate multiple FIFO buffers, called virtual channels, with a single input/output port to reduce head-of-line blocking and enable prioritization. The virtual channels of an input port compete with each other for the physical channel, so adding virtual channels complicates the arbitration process. Round-robin arbiters in a router work by prioritizing among the input virtual channels in a round robin manner: the arbiter chooses flits in different VCs in a round-robin order such that a flit from the next non-empty VC is selected every cycle. The downside of VCs is increased arbitration complexity, increased number of buffers, and increased hardware cost. Buffering in the network requires one additional complexity. To ensure there is no deadlock in the network, buffer allocation and deallocation across different routers need to be flow-controlled. A router should not be able to allocate space in a downstream router if the downstream router does not have enough available buffers. A commonly used flow control mechanism is credit-based flow control [10]. The downstream router sends credits indicating the availability (i.e, freeing) of a buffer in its input ports to its neighboring routers. Before allocating a downstream buffer to a flit, each router checks whether or not there are enough credits are available downstream. Note that the implementation of credit-based flow control requires a back-network that communicates credit information.

Several optimizations have also been proposed to reduce the latency of virtual channel routers in NoCs. The two major optimizations are 1) lookahead routing [8, 17], where the computation of the route of a packet is performed in the previous router, thereby eliminating route computation from the critical path of packet processing in a router, and 2) speculation [31, 35], which speculatively allocates an output VC for a packet, thereby reducing router latency when this speculation succeeds.

B. Bufferless Routing

We will use the BLESS routing mechanisms [30], and in particular FLIT-BLESS [30], as the bufferless router baseline in this paper. BLESS eliminates all input and output buffers in a router (the pipeline latches in the router pipeline are not eliminated) with the goal of reducing the area, power consumption, and complexity of the on-chip network. When a flit arrives at the router, instead of being buffered, it is routed immediately to an output port. Since there are no buffers, all packets that arrive at the router must be immediately forwarded to an adjacent router. If multiple flits contend for the same output port, one of them has to be routed to another output port that is not desirable. In other words, a flit is deflected or misrouted to an undesirable output port (which does not reduce the flit's distance to the destination) if no desirable output ports are available due to other flits being routed there. The basic idea in BLESS is that the deflected or misrouted packets will eventually reach destinations by going through a different path through the network than the optimal one.

FLIT-BLESS is a simple form of bufferless routing that performs routing for each flit independently. Each flit has header information associated with it, and flits from the same packet can be routed to different output ports. To ensure livelock freedom (i.e, a flit does not get deflected indefinitely), FLIT-BLESS forms a consistent total order among all flits and performs oldest-first arbitration among the incoming flits in each router. When multiple flits contend for the same output port, the oldest one among them wins the arbitration and is assigned the output port whereas others are potentially deflected to an undesirable output port. A total packet order using oldest-first arbitration guarantees that the oldest flit in the network will always make forward progress to its destination, guaranteeing the network to be livelock free, as shown in [30].

To ensure that the network is not overloaded with continuously deflected packets and to avoid packet dropping, BLESS performs injection control. A resource (or node) can inject a flit into the router only when at least one incoming link in the router is free. This ensures that no packets need to be dropped in BLESS unlike some other proposals for bufferless networks [18, 19]. We refer the reader to [30, 2, 5, 11, 12, 13] for details on bufferless routing.

Previous work [30] extensively evaluated the performance of BLESS compared to buffered routing using simulation and showed that for real applications the performance of BLESS is comparable to buffered routing. The work also provided simulation-based energy comparisons and back-of-the-envelope area comparisons between BLESS and buffered routing, showing promising reductions in network energy (40%) and network area (60%). Unfortunately, the evaluation did not take into account implementation challenges in

both BLESS and buffered routing, and did not provide detailed area and power estimates on a real implementation. **Our goal is to provide a more realistic comparison of BLESS and buffered routing with a real implementation of each and to realistically evaluate the implementability, benefits, and disadvantages of bufferless routing in comparison to state-of-the-art buffered routing algorithms.**

Note that the deflection-based routing algorithm of BLESS can be used with buffers, in order to reduce the probability of congestion in the network. Moscibroda and Mutlu [30] call this mechanism BLESS with buffers, and we evaluate it in this paper.

C. Topology

While buffered routing is implementable in any topology, bufferless routing is restricted to certain topologies because *every* incoming flit must be able to be routed right away to an outgoing link immediately. This requires the number of outgoing links in a router to be greater than or equal to the number of ingoing links. A mesh topology satisfies this requirement, and we use a 2D mesh for our basic implementation. A torus topology also satisfies the requirements for BLESS. We also use a torus topology in our alternative implementations. BLESS was not previously evaluated for a torus topology.

III. Evaluated Router Microarchitectures

We first describe in detail the microarchitecture and logic design of the buffered, bufferless and deflection-based buffered routers we prototyped. We use round-robin VC arbitration for the buffered router and oldest-flit-first arbitration for the bufferless router. The buffered router uses the commonly implemented dimension order (XY) routing, whereas the bufferless router uses the FLIT-BLESS routing algorithm [30], which is a modified form of dimension-ordered routing with deflections enforced if a flit cannot be assigned a productive output port.

A. Baseline Buffered Router Microarchitecture

Fig. 2 shows the microarchitecture of our state-of-the-art baseline buffered router. We have parameterized the router to make the design easily configurable to support various features in addition to credit-based flow control, wormhole routing, and virtual channels. These parameters are as follows:

p: the number of physical input and output ports; we set p=5 in our evaluation of mesh and torus networks;

v: the number of virtual channels supported by each physical port; *w*: width of each flit;

l: the number of flits in each packet;

s: width of the control signals between routers for credit-based flow control for buffer management.

For the look-ahead router, *s* also contains the routing address for the next hop. We evaluate different number of virtual channels by varying the parameter *v*. If v is set to 1, the buffered router is essentially a canonical wormhole router with a single FIFO buffer for each physical port; we call this configuration the 1-VC configuration. The size of each FIFO buffer is set to buffer enough flits such that a whole packet can be buffered in one virtual channel at any given time. We use non-atomic allocation of virtual channels, i.e., a virtual channel at any given point can contain flits from multiple packets. We briefly describe the major modules implemented in our prototype baseline buffered router.

Input Controller Module: The input controller is responsible for controlling and managing all actions (allocation, buffer allocation) related to the virtual channels in an input port. It contains one or more virtual channels and one route computation module. For each VC, the input controller performs both packet-level and flit-level processing.

Virtual Channel (VC) Module: Virtual channel logic contains a FIFO buffer and a set of state registers. The Global State Register is used to record different processing stages a packet is in, such as *idle*, *routing*, *waiting output VC*, and *active*. The Routing Register holds the output port number for the incoming packet. The Output VC Register holds the allocated output virtual channel number. Finally, the FIFO Head and Tail Pointer Registers manage flit reads and writes. The VC buffer is a simple FIFO buffer that stores incoming flits. It includes control logic to forward the head flit in a packet to the routing logic for route computation and body/tail flits to the switch allocator for arbitration. Only the flit in the head of the FIFO queue participates in route computation or output-VC/switch arbitration. Note that there are as many VC modules as the total number of virtual channels, i.e. (number of VCs per input port) x (number of input ports).

Route Computation (RC) Module: The purpose of this module is to compute the route of a packet. Since we use wormhole routing, route computation is performed only for the head flit. The routing logic uses XY (dimension-order) routing for direct k-ary n-cube networks, such as torus and mesh networks. XY routing is commonly implemented in NoCs and details of its operation and properties can be found in [10].

Virtual-Channel Allocator (VA) Module: The purpose of this module is to allocate an output virtual channel for the packet based on the result of the route computation. Only the head flit in a packet is processed by this module because other flits follow the head flit due to wormhole routing. The VA module maintains a table containing information for each output VC. The table indicates whether or not each output VC is already assigned an input VC whose contents will be forwarded to the output VC. Allocation of an output VC to an input VC is done using a round-robin arbiter that selects an input VC. For each output VC, the arbiter chooses a different input VC in a round-robin order such that a non-empty VC is selected every cycle.

Switch Allocator (SA) Module: The purpose of this module is to allocate each crossbar port to one of the VCs that are contending for the port. The switch allocator maintains an availability table for each of the output VCs based on how many buffers are available in the downstream router. Contrary to the traditional design where the VC allocator processes the input credits, we use the SA to record available downstream buffer space. This reduces unnecessary communication between the VC and the SA as the SA always needs to know the availability of downstream buffers. The priority among competing input VCs on the same input port and priority among competing input port is computed using round robin order to ensure fairness.

Crossbar: The crossbar (or switch) contains logic to connect physical input ports to physical output ports. Each input port shares a single crossbar port between different virtual-channel buffers. Every cycle, a flit of the selected VC is granted passage on the crossbar to the appropriate VC in the appropriate output port.

Operation of the Buffered Router: The basic steps undertaken by our 4-stage VC router for a given packet/flit are as follows. We later describe the techniques to reduce the number of stages on the critical path of the router to respectively 3 and 2, which we also evaluate as part of our prototype.

0. *Idle State*: The virtual channel is in *Idle* state at cycle 0. When a flit of a new packet arrives at the input controller, the packet destination address and virtual channel field are extracted and decoded. At the start of cycle 1, the destination address is sent to the routing logic as shown in Fig 2. At the same time, this flit is stored in the FIFO buffer based on its virtual channel field value, which was assigned by the previous router that allocated the output virtual channel for the packet. The input controller then changes the global register state for the virtual channel from *Idle* to *Routing*.

1. *Route Computation Stage*: During cycle 1, the RC module calculates which output port the head flit should be forwarded to using the XY routing algorithm. The destination output port is ready at the end of cycle 1.

2. *Virtual-Channel Allocation Stage*: At the start of cycle 2, the input controller changes the state register from *Routing* to *VC allocation* and stores the returned route from RC into the route register. Simultaneously, the input controller sends a virtual channel allocation request to the VC allocator given the packet's physical output port number assigned by RC. The VC allocator then arbitrates VC requests from all the VC input controllers. At the end of cycle 2, the VC allocator sends the result of VC allocation, indicating which output VC is allocated for the head flit in which input VC, to input controller.

3. *Switch Allocation Stage*: At the start of cycle 3, the input controller checks the results from the VC allocator. If the VC allocator failed to allocate any output VCs for the input VC, the input controller stalls and keeps that VC in VC allocation stage to repeat the work in Step 3. If the input VC is successfully assigned an output VC, then its state is changed to the *Active* state. The input controller sends the switch allocation request to the switch allocator with its input VC's physical port destination and output virtual channel number. At the end of this cycle, the switch allocation logic matches all the switch allocation requests from different input ports to available output ports and sends the switch allocation results to each allocated input VC. It also sends the path configuration signal to the crossbar (this configures the connections of the crossbar such that each input port is connected to the allocated output port).

4. *Switch Transfer Stage*: At the start of the cycle 4, the input controller sends out the head flit of its selected VC (if any) to the crossbar. The unselected, active input VCs (if any) will remain in *Active* state and repeat Step 4 in the next cycle. If the flit in the selected VC is not a tail flit, the VC stays in the *Active* state and competes for switch allocation for the next flit in Step 4 (in the next cycle), while the current flit is being transferred via the crossbar. If the flit in the selected VC is a tail flit, then the selected VC changes into the *Idle* state and the input controller sends a VC release signal to VC allocator to recycle the assigned output VC number (because the entire packet is now transferred to the output VC).

Latency Reduction Techniques (Lookahead Route Computation): It is simple to reduce the number of pipeline stages from four to three by performing route computation ahead of time in a previous router to remove it from the critical path [8, 17]. A new field called *next hop routing* is added into the head flit to record output routing of the downstream router for this packet. When a new head flit arrives, the VC goes directly to the *VC Allocation* state, skipping route computation, which was already performed in the previous router for this head flit. The input controller extracts the *next route* field as its output port and sends a request to the VC allocator for an available output VC on that output port. In the meantime, RC calculates the route of this head flit in the downstream router.

B. Bufferless (BLESS) Router Microarchitecture

Our BLESS router implements the FLIT-BLESS routing algorithm, which is described in detail in [30, 11]. The algorithm first prioritizes the incoming flits based on their ages, ranking older flits over younger ones. Then, it assigns output ports to each flit based on that prioritization order. The oldest flit is allocated an output port first, and the youngest flit is allocated an output port last. Each incoming flit has productive port(s) that it prefers to go to. A productive port is a port that would reduce the distance of the flit to its destination. For each flit, the BLESS routing algorithm first tries to assign a productive port. If no productive port is available at the time of port assignment, the flit is assigned a non-productive port, i.e. it is deflected. If there are two possible ports for a flit, then the arbiter picks the port in the X direction over that in the Y direction.

Based on this algorithm, the microarchitecture of the BLESS router we designed is divided into three stages (Fig. 3). Note that the productive ports of a flit is determined by the *Route Computation* logic in stage 1 and fed into the arbiter using the rmatrix (route matrix). At the same time, *Rank Priorities* logic ranks the priorities of incoming flits based on their ages in parallel with RC. The port assignment is done by the arbiter in stage 2. Once a port is assigned, each fit proceeds through the *Crossbar* stage to its output port. We describe each of the three stages.

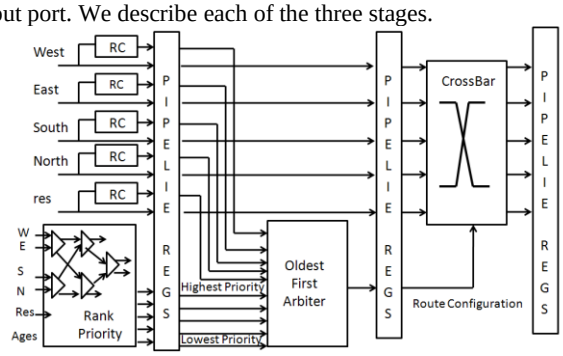

Fig. 3 BLESS router microarchitecture

Route Computation (RC): In this stage, each incoming flit is processed in parallel by RC modules to determine the shortest direction it can take in X or Y dimensions to its destination. This stage also generates the ready signal to notify the resource whether or not its current flit can be injected into the network. To generate this signal, the router samples each RC module to determine if either less than four input ports have valid incoming flits coming in or at least one of the incoming flits has reached its destination. If either case is true, the ready signal is asserted indicating that the resource can inject into the router.

Rank Priorities: Priority comparison sub-module is the first step of an arbiter, ranking the incoming flits based on their ages through a series of comparators in rank priority (Fig. 3). The arbiter ranks the flits in age priority order with the oldest flit assigned the highest priority while the youngest flit assigned the lowest. If an input port does not have a valid flit during that particular cycle, it will bubble down in the priority list. The priority of the resource's port (i.e., the injection port) is not calculated as it is always the lowest. Note that the size of the age priority field is determined based on the maximum time a packet can spend in the network.

Switch Allocation (SA): The microarchitecture of oldest first arbiter for BLESS is shown in Fig. 4. After the first stage produces a priority and effective output port lists, the second stage, through a series of arbiter cells as shown in Figure 4, assigns an output port to each flit one by one in priority order, going from highest to lowest priority. Each arbiter cell takes as input the current output port availability matrix (*amatrix* in Fig. 4), the input port number and the desired effective output port matrix, *rmatrix* (Fig. 4). The input port number is used to select its corresponding productive output port list from *rmatrix*. Based on output availability bitmap, the arbiter first tries to allocate an available productive output port in the X dimension. If it fails, it will try to allocate a productive port in the Y dimension. If it fails again, the flit will be deflected, and the arbiter chooses an available output port based on a *fixed order* from north, south, east and then west. When the arbiter cell finishes processing a flit, it generates a new *amatrix* to the next arbiter cell, marking the just-assigned output port unavailable and updating the route configuration. *Route configuration* is a list of the five ports showing which input port is assigned to which output in that cycle. Going through each arbiter cell, the number of available output ports decreases. The resource, if injecting, is allocated an output port after the lowest priority input port is assigned its output port. After all the five arbiter cells finish port allocation, route configuration table is generated to configure the behavior of the crossbar for the next cycle. As we can see in Fig. 4, P0, which corresponds to the oldest incoming flit, is always the first to be allocated an output port (P4 is for the flit to be injected).

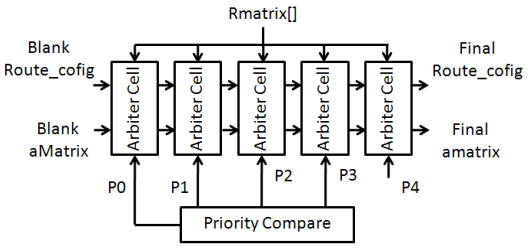

Fig. 4 Oldest-first arbiter microarchitecture

Generally, the oldest-first arbiter is somewhat complex as it needs to calculate priorities for each port and then assign the output ports one by one from the highest to the lowest priority. The total delay of the arbiter is the priority calculation logic and the serial arbiter cells: *DelayArbiter = DelayRank + DelayArbiterCellStage*. Priority rank calculation delay is proportional to *log2(port number)* delay comparators. The delay of the port assignment is proportional to the number of ports multiplied by the delay of each arbiter cell. The delay of each arbiter cell is also linearly proportional to the number of ports, which indicates that the delay of output port assignment is proportional to the square of the number of incoming ports. If we treat arbitration as an atomic operation, it will take a long time and it will likely cause BLESS to have low clock frequency than a 1-VC router. However, with the design in Fig. 2, we pipeline the arbiter into two stages and hide *priority ranking* latency with *route computation* latency. This can greatly decrease the critical path delay of oldest-first arbitration. Therefore, the delay of the oldest-first arbiter for the BLESS router can be less than that of 1-VC buffered (Sec. 4).

C. *Deflection-based Lightly-Buffered Router Microarchitecture (BLESS with Buffers)*

In order to increase throughput and decrease latency for medium and high injection rates in BLESS, we add a single-flit size buffer for each input port plus some control logic to buffer an incoming flit in case it does not find an effective output, which can reduce its distance to its destination. We call this design *BLESS with buffers* [30] (see Fig. 5). The key difference from the buffered router is that the basic principle of deflection routing remains the same as in BLESS, and flow-control is purely local as in BLESS. The differences between BLESS and "BLESS with buffers" are twofold. First, *BLESS with buffers* has a controller in each input port to manage the buffer. This controller also generates control signals to control the behavior of the two pipeline stages. Second, *BLESS with buffers* has a *mustSchedule* bit [30] for each flit of each pipeline stage, which indicates that the flit must leave the router because some other flit will need to be buffered.

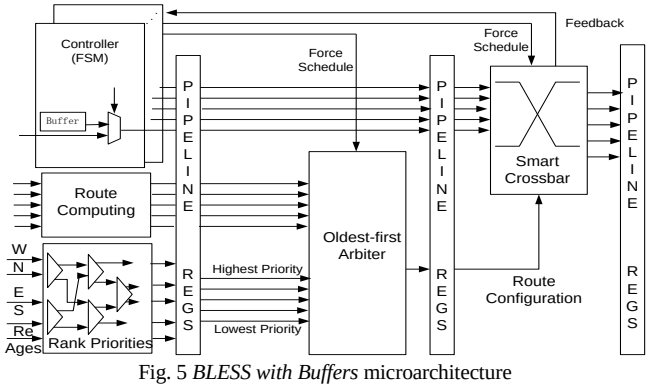

Fig. 5 *BLESS with Buffers* microarchitecture

An incoming flit is buffered in the single-flit buffer if it cannot be forwarded to a productive output port (instead of being deflected as in BLESS). A flit in the buffer is marked *mustSchedule* if the same input port receives another flit. Note that, whether or not a flit receives a productive output port is determined at the end of the router pipeline. Therefore, when a flit is marked *mustSchedule*, there might already be flits in the router pipeline that might not receive productive output ports. Since there is a single buffer entry and a new flit is present in the input port, all such flits *must* be scheduled regardless of whether or not they will receive productive output ports. To achieve this, the *mustSchedule* bits of the flits associated with an input port in the router pipeline are all set, when the *mustSchedule* bit of the buffered flit is set. All *mustSchedule* flits have to be sent out of the router; they cannot be buffered.

When a flit stays in the input buffer, its *age* increases every clock cycle. The flit can eventually become old enough to gain high priority to leave that router, guaranteeing deadlock freedom.

The stages of *BLESS with buffers* are briefly described below.

Stage 1: If both the incoming flit and the flit in the buffer are valid, the controller selects the flit in the buffer for routing and stores the incoming flit in the buffer. In addition, the *mustSchedule* bit in stage 1 of that port is set. In the "rank priorities" logic, the flit with the mustSchedule bit set gets higher priority than flits from other ports with *mustSchedule* set to zero. At the same time, the controller forces the *mustSchedule* bit in stages 2 and 3. Thus, previous valid flits have to leave the router to make way to the *mustSchedule* flit that cannot be buffered.

Stage 2: This stage receives flits to be switched, desired output port list, assigned priorities, and the *mustSchedule* bit from stage 1. It works the same as the BLESS router except for how it deals with the *mustSchedule* bit. If the *mustSchedule* bit from stage 1 is set, the controller forces the *mustSchedulebit* in stage 2 to be set.

Stage 3: The crossbar is slightly different from the crossbar of BLESS. If the inherited *mustSchedule* bit from previous stage is set or the controller forces it to be set, the *mustSchedule* bit in stage 3 will be set. Just like in BLESS, all incoming valid flits have now been assigned a unique output port, either a desirable one or an undesirable one (deflected port). Only those flits that 1) have their *mustSchedule* bits set or 2) are assigned productive output ports can leave the router. All other flits are blocked. The latter type of flits notify the controller that the contents in the input buffer are still valid, and the flit in the input buffer should compete for arbitration in the next cycle.

IV. EXPERIMENTAL METHODOLOGY

We evaluate the performance of routers via FPGA and ASIC-process implementations of each. The different versions of VC buffered routers are compared to the BLESS router in terms of chip area, clock frequency/critical path delay, power consumption, packet delivery latency, and network saturation throughput.

Interconnection Network Model: We implemented two types of on-chip network topologies, 2D mesh and 2D torus, for various performance comparisons. We evaluate network dimensions of 3x3 and 4x4. Each router has its own address as {x, y} to locate its position in the network for dimension-order routing. Each router has 5 input ports and 5 output ports, including the injection port. Each data packet consists of 4 flits and each flit is 32-bit wide to cross the 32-bit data link in one clock cycle. For the buffered router, additional signaling links are used to convey credits between adjacent routers for flow control.

Request Generation Patterns: Each of the routers is associated with an injection logic. The injection logic mimics the behavior of processor cores and cache by injecting various synthetic traffic patterns into the network. We investigated six different traffic patterns, which are listed and described in Table I. For uniform random (UR) and nearest neighbor (NN) traffic, the destination address of each packet is determined by a defined statistical process. We use a linear feedback shift register (LFSR) to get the destination address in a pseudo-random fashion. For transpose (TR), tornado (TOR), bit complement (BC), and hot spot (HS), the destination address is determined according to the criteria listed in Table I. All these traffic patterns are experimented with in the real FPGA implementations of the NoC with 100-million packet injections.

TABLE I. Evaluated traffic patterns

UR	Router chooses a random destination among other routers with equal probability and sends a packet to that destination. The probability is equal among the other routers
NN	Each node sends a packet to one of its immediate neighbors with equal probability
TOR	Node {X, Y} sends packets to node {X+k/2−1, y} mod k for the k-ary network (k=4)
TR	Node{X, Y} sends packets to node {Y, X}
BC	Node with address {b0, b1, b2, b3} in bits sends packets to the node with address NOT{b0, b1, b2, b3} in bits
HS	All the nodes send the packet to a certain single node. The hot spot can act as receiver only or can be transmitter and receiver.

FPGA Evaluation Hardware Platform: We use the Berkeley Emulation Engine 2 (BEE2) board to emulate both the BLESS and buffered on-chip networks. The BEE2 board is a multi-FPGA board containing five Xilinx Virtex-II Pro FPGAs. Each Virtex-II Pro FPGA contains two on-chip PowerPC 405 cores. Additionally, firmware can run on these PowerPC cores to collect emulation results. We currently use the control FPGA to emulate the 4x4 and 3x3 networks. As shown in Figure 1, each router has its own local injection logic. Together with auxiliary logic, it can be integrated with other Xilinx IPs to form a working evaluation system. When simulation starts, the PowerPC initiates a start command and configures the emulation core with a set of parameters. These include flit injection rate and traffic patterns. Every time interval, the emulation core returns back the simulation results such as the number of flits sent and received, the maximum and average age of the packets, and the total clock cycles elapsed. The PowerPC collects all these results and stores them for further analysis. During network performance analysis, we run the PowerPC core at 400MHz and the on-chip bus at 100MHz to collect latency data. The routers and their injection nodes run at their maximum possible clock frequency.

FPGA Area/Frequency/Power Estimation: Router architectures are implemented in Verilog HDL. They are synthesized to 2vp70-ff1704Xilinx VirtexII-Pro FPGA with speed grade 7. The resulting netlist is placed and routed to the same FPGA using Xilinx ISE 9.2i. Xilinx synthesis tools are used to obtain the maximum frequency and the delay of the critical path. The area of BLESS and buffered routers is compared by their resource usage in the FPGA. The power consumption of both routers is estimated by XPower [34], a commercial tool provided by Xilinx. XPower reports the power of individual design parts, including signals, clocks, logic, input/outputs, and static leakage power. *Signal power* is the power dissipated by the wiring between logic blocks. *Clock power* is consumed by the clock tree on the FPGA. *Logic power* is the power consumed by combinational and sequential logic. *Input/output power* is the power consumed by the FPGA's input/output buffers. Since both BLESS and buffered routers are on-chip and do not affect input/output power, we ignore input/output power. *Static power* is always overestimated by XPower as a large constant value regardless of working frequency and logic design so we cannot compare the static power of different designs due to the XPower's limitations. As such, we will only compare the logic, signal, and clock power between BLESS and buffered router architectures. Note that it is expected that eliminating buffers reduces static power, which makes our power estimations biased in favor of the buffered router.

ASIC Evaluation: We also use the Synopsys Design Compiler tool to synthesize each router's Verilog design using an industrial 65nm bulk CMOS process to further investigate its area, power, and frequency in a real ASIC design flow.

V. Experimental Results and Analysys

We first show the results of area, power, and frequency obtained by analyzing each router implementation with synthesis tools. Second, we present network-level comparisons. Third, we compare the networks in terms of delay and throughput using synthetic traffic patterns injected into the FPGA implementation. Finally, we provide analysis results from an ASIC implementation.

A. FPGA Area Consumption

The elementary programmable logic block in an FPGA is called a slice. Each slice consists of two 4-input LUTs (Look-Up Tables), which can implement any 4-input Boolean function; two 1-bit registers that can be configured either as flip-flops or latches, and two dedicated user-controlled multiplexers and other arithmetic logic. These basic FPGA hardware resources are allocated to the designed logic. The area of each router design therefore can be determined by how many slices have been allocated on the FPGA. Table II shows the total area of each router in terms of the number of slices, flip-flops, and LUTs used and the percentage of FPGA resources consumed for the BLESS router, BLESS with buffers and buffered routers with 1, 2, 4 virtual channels. Fig. 6 breaks down the area usage of each router to individual router components (input controller, VA, SA, and crossbar).

TABLE II. FPGA area usage of various routers

	BLESS	BLESS(Buf)	1-VC	2-VC	4-VC
Slices	634	930 (2%)	989 (2%)	2098(6%)	4638(14%)
FF	335	695 (1%)	947 (1%)	1764(2%)	3480(5%)
LUTs	1090	1671 (2%)	1392(2%)	3133(4%)	7362(11%)

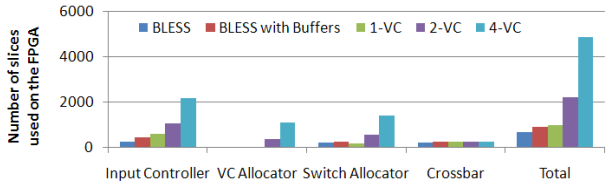

Fig. 6 FPGA resource consumption of various routers

Several observations are in order from these figures. First, BLESS consumes significantly smaller area than all the other routers. A single-VC wormhole router that has a single 4-flit FIFO in each of its input ports consumes 55.9% more area than BLESS that has no input buffers. As the number of VCs increases in the buffered router, BLESS's area advantage becomes significantly more pronounced. The 2-VC and 4-VC buffered routers respectively consume 3.31 and 7.32 times more area than BLESS. Hence, we conclude that BLESS is a significantly more area-efficient router design than even the simplest buffered designs with very few buffers.

Second, as expected, BLESS's area advantage mainly comes from a significantly smaller area in the *input controllers*, which contain the input FIFO buffers in buffered designs that BLESS eliminates. Note that in our design we did not make the buffer entries large as we emulate only packets with short flits (4 bytes per flit) and each buffer takes up four entries to store the incoming flits. BLESS and BLESS with buffers also do not need VC allocation logic as they eliminate VCs and are therefore more area-efficient than multiple-VC designs which consume significant area to perform allocation of multiple VCs. Since the crossbar remains very similar in BLESS and buffered designs, its area consumption is relatively constant across designs. The only exception is BLESS with buffers, as its crossbar has to prevent flits with *mustSchedule* bit set to zero from leaving the router.

Third, perhaps surprisingly, BLESS requires 18% more area in the switch allocator logic than the 1-VC buffered router. The overhead of BLESS comes from the large number of gates required to implement the oldest-first arbitration logic between the 4 input ports (as described earlier, this is needed to prevent livelock). BLESS with buffers takes 54% more area than 1-VC due to the complexity of oldest-first arbitration plus the additional logic to deal with the *mustSchedule* bits. This complexity is not present in the buffered routers that perform round-robin arbitration across all VCs. However, as the number of VCs increases in the buffered router, BLESS's area disadvantage in the switch allocator turns into an area advantage. Arbitrating between 8 input VCs in the 2-VC buffered design (2 VCs per input port) using round robin consumes more area than arbitrating between 4 input ports using oldest-first. **We conclude that the area-efficiency of the oldest-first arbiter used in our implementation becomes relatively higher than the area**

efficiency of the round-robin arbiter used in our buffered router as the number of VCs increases.

B. Frequency Analysis

Table III shows the critical path delay and maximum clock frequency of various routers. BLESS router's clock frequency is 20% higher than the buffered 1-VC router. With careful design of the oldest-first arbiter, we were able to divide it into two atomic operations (rank priority and oldest-first arbitration) and hide the latency of rank prioritization under the latency of route computation. Due to additional processing on *mustSchedule* bits, BLESS with buffers has an 8% lower clock frequency than a single virtual channel router. As the number of VCs in the buffered router increases, the critical path of the buffered router increases due to increased complexity of switch allocation between multiple virtual channels. In fact, BLESS becomes even more advantageous in terms of frequency over a 2-VC buffered router, providing 1.56 times higher frequency. The arbitration complexity of multiple VCs is higher than that of BLESS because in BLESS only 5 input ports arbitrate for 5 output ports whereas as the number of ports that participate in arbitration increases with the number of VCs. For example, with a 2-VC router, 10 virtual input ports arbitrate for 10 virtual output ports, the number of virtual ports that participate in arbitration becomes 20 with 4 VCs. Hence, *we conclude that BLESS provides a higher frequency router design than buffered routing.*

TABLE III. Critical path length of various routers

	BLESS	BLESS(Buf)	1-VC	2-VC	4-VC
Delay (ns)	13.285	17.335	15.948	34.081	71.788
Freq. (MHz)	75.27	57.69	62.704	29.342	13.93

Figure 7 shows the delay of each pipeline stage in all the routers. We make several observations. First, switch allocation is the critical path in buffered routers, while priority ranking is the critical path for the BLESS router. As the switch allocation of the buffered router is atomic, it is not easy to divide it into multiple stages to reduce the critical path. As the number of VCs increases, more allocation requests need to be processed by the switch allocator and the increase in delay is nearly linear with the number of VCs.

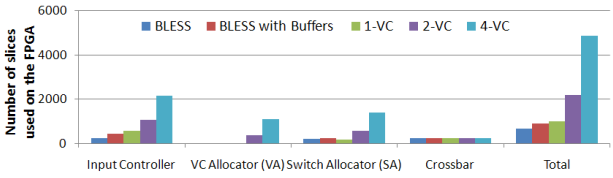

Fig. 7 Combinational delay of each pipeline stage of various routers

Second, all buffered routers have very similar logic delay for the input controller and the crossbar. Despite adding buffers to the input controller for the buffered router, we optimized the router such that buffer writing/reading is performed in parallel with route computation. By moving the input buffer out of the critical path, our buffered router input controller implementation hides the buffering delay. Therefore, all routers have the same delay for their input controllers. As the crossbars we used for each router, except for BLESS with buffers, are almost exactly the same, it is expected that they have the same delay.

C. Power Analysis

We use XPower to estimate the total power consumption of each router. XPower is a widely used commercial tool that performs power estimation for Xilinx FPGAs. The tool is able to break down the sources of power consumption for a certain design given a specific working environment. In our analysis, the design works under the environment of voltage source V*ccint* = 1.50v, V*ccaux* = 2.50v, and V*cc025* = 2.50v. The junction temperature is set to 29℃, ambient temperature to 25℃, case temperature to 29℃, and Theta J-A (i.e., thermal resistance) to 9℃/W. We evaluate the router at the various clock rates, including 10MHz, 25MHz and 50MHz. For each clock speed, the power consumption of the individual routers is investigated under various injection rates (5% to 50%, in fraction of cycles during which a flit is injected to each node). Figures 8-10 respectively show the power consumption results at 10, 25, and 50 MHz. There are several key observations from these figures. First, the BLESS router consumes significantly less power than all types of buffered routers for all clock frequencies and for all injection rates. For example, at 10MHz and 5% injection rate, the 1-VC router consumes 1.6X more *total power* than the BLESS router. This power savings of BLESS is a direct result of eliminating the management of buffers.

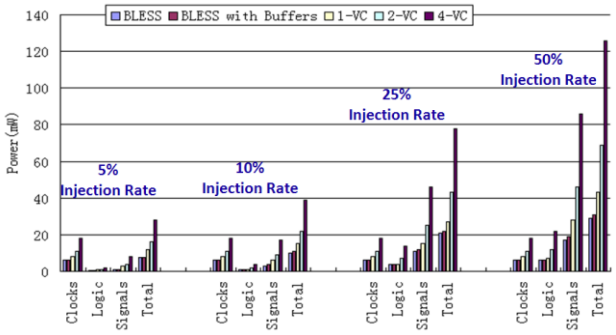

Fig. 8 Power consumption of various routers running at 10 MHz

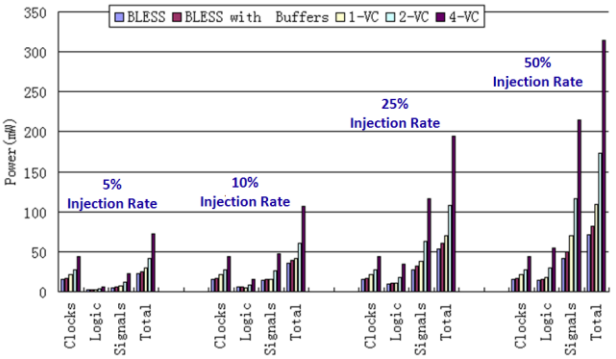

Fig. 9 Power consumption of various routers running at 25 MHz

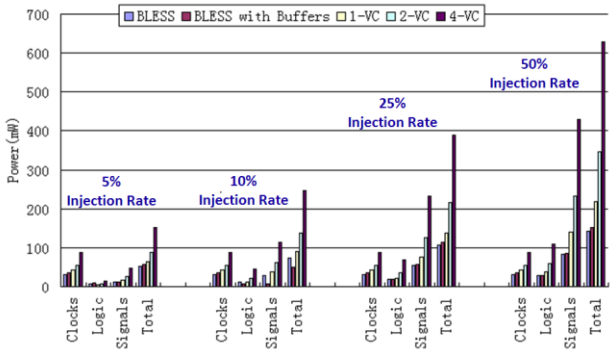

Fig. 10 Power consumption of various routers running at 50 MHz

Second, as the number of VCs increases, the power consumption of the buffered router also increases proportionally. At 10MHz and 5% injection rate, the 2-VC router consumes 2.13 times and the 4-VC router consumes 3.73 times more power than the BLESS router. Virtual channel allocation, switch allocation, as well as buffer management becomes more complicated, requiring more signaling to be active as number of VCs increases, thereby leading to significantly larger power consumption over BLESS.

Third, as injection rate into the router increases, the power consumption benefit of BLESS over the buffered routers also

increases. The buffered routers have significantly more internal communication signals and wires between different blocks (described in Section 3.1) to enable buffer management and virtual channel allocation, a complexity that does not exist in BLESS. As injection rate increases, more of these signals become active, leading to more communication and hence higher power consumption. Note that this conclusion is supported by the observation that the *signaling* power (denoted as *Signals* in Figs. 8, 9, 10) shows the greatest disparity between bufferless and buffered routers at high injection rates (e.g., 50%).

Fourth, as injection rate increases, the signaling power, the power consumed by interconnect within the router, consumes a greater fraction of the total router power. This is expected as more of interconnect becomes active more frequently with more frequent injections into the router. Buffered routers design can be made more efficient at higher frequencies by reducing the amount of interconnect.

Fifth, at low injection rates, which are more common in general-purpose applications run on chip multiprocessor on-chip networks (as shown in [30]), the power consumed by the clock tree dominates the total power consumption of the bufferless router. Therefore, to increase the efficiency of bufferless routers even further in such designs, designers should focus on optimizing clock tree implementations and investigate clock power reduction techniques.

D. Network Analysis

We show the FPGA area usage for the 3x3 and 4x4 torus on-chip networks in Figure. 11. The 3x3 BLESS network uses 22% fewer slices, 55% fewer flip-flops and 6% fewer 4-input LUTs compared to the 3x3 network that is built with the most area-efficient buffered router with only one VC. For the 4x4 network, BLESS network uses 24% fewer slices, 55% fewer flip-flops, and 7% fewer 4-input LUTs than the 1-VC buffered network.

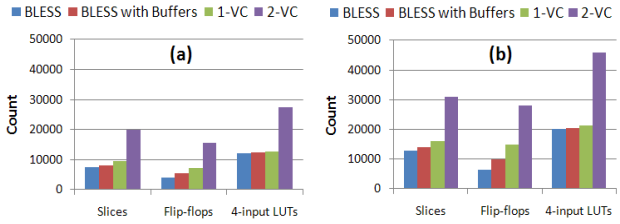

Fig. 11 FPGA area usage of (a) 3x3 and (b) 4x4 torus designs

BLESS network uses much fewer flip flops than buffered networks because 1) it eliminates all input buffers, which are implemented with flip-flops, 2) it does not need to maintain buffer management related state (such as *Idle*, *Routing*, *VC allocation*, *Active* and *Transfer* states) as done in the buffered network. However, the combinational logic area complexity of the BLESS network is not greatly better than that of the 1-VC buffered network: BLESS network uses only 7% fewer 4-input LUTs than the 1-VC buffered network. This is because BLESS routers need to prioritize flits using oldest-first arbitration, which requires significant combinational logic complexity. However, when the buffered network uses 2-VCs, combinational logic complexity of oldest-first arbitration is dwarfed by the combinational logic complexity of managing multiple virtual channels: the 3x3 BLESS network consumes approximately 30% of the area of a 2-VC buffered network, showing that BLESS has significant area-efficiency advantages for the entire network.

Figure 12 shows the entire network power consumption for respectively 3x3 and 4x4 torus networks at 25MHz with 5% and 10% injection rates. In the 3x3 configuration, BLESS saves 25% total power under 5% injection rate and 23% under 10% injection rate compared to the 1-VC network. For the 4x4 network, BLESS saves 18% and 19% total power respectively under 5% and 10% injection rates, compared to the 1-VC network. We can see that BLESS consumes slightly more logic power than the 1-VC router due to the complexity of oldest-first arbitration used to prioritize incoming flits, but it consumes significantly less signaling power because it does not perform buffer state management or virtual channel allocation. The BLESS network's power savings is higher when compared to the 2-VC network, ranging from 38% to 72%. We conclude that BLESS leads to substantial overall network power savings, ranging from 18% to 72%, over buffered networks even though it increases arbitration complexity in each router.

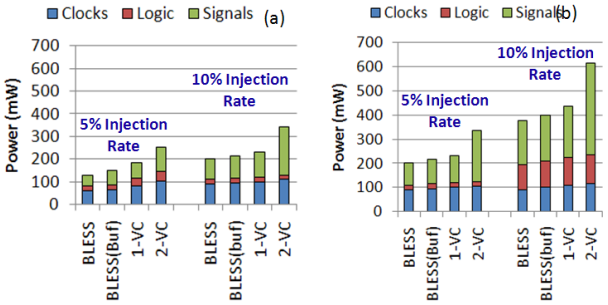

Fig. 12 FPGA power consumption of (a) 3x3 and (b) 4x4 torus designs

E. Network Packet Delay and Throughput

Figs. 13 and 14 show packet latency as injection rate (in flits/node/cycle) is varied with 6 different traffic patterns for 4x4 torus and mesh networks respectively. All traffic patterns are run on our FPGA implementation of BLESS and 2-VC buffered networks. We use a 3-stage BLESS router and a 3-stage 2-VC channel router, which incorporates look-ahead routing. We set the latency of each router to 3 cycles to provide a fair comparison (Note that the frequency of *this* BLESS router is actually 2.56 times higher but we do not factor this into our evaluation to favor the buffered router). *Hot spot* pattern is emulated by having each router send packets to the router at address 0101. Several observations are in order from these load-latency curves. First, as expected, BLESS network's latency is similar to that of the buffered network at low flit injection rates for all traffic patterns. However, as injection rate increases buffered network becomes lower latency because BLESS starts deflecting packets due to congestion. This happens only in those patterns where congestion happens. Since congestion dos not arise in *bit complement*, *tornado*, and *transpose* patterns on the torus, and the *tornado* pattern on the mesh, packet latency stays constant with injection rate. Buffered network is better able to tolerate congestion and leads to reduced latency at high injection rates because buffering packets leads to better bandwidth utilization than deflecting them.

Second, BLESS has lower saturation throughput than the buffered network because deflections increase the load on the network links, leading to increased network utilization and saturation as there is no other place than the links to keep the packets at times of congestion. With *hot spot* traffic on the mesh, BLESS saturates at an injection rate of 0.033 flits/node/cycle whereas buffered network saturates at a rate of 0.058. *Hot spot* traffic pattern presents the worst case for BLESS among the evaluated patterns because it results in the highest congestion, leading to a high deflection rate. Third, BLESS's performance (saturation throughput and latency at high injection rates) becomes much more comparable to the buffered network for all traffic patterns when the network topology is a torus instead of a mesh. This is because the additional links in the torus adds 1) *path diversity*, which increases the number of paths that can be taken by deflected packets, 2) ensure the torus does not suffer from congestion in edge routers. Since BLESS essentially uses links as "temporary buffers," additional links enable the BLESS network to sustain a higher throughput. As a result, with *hot spot* traffic, the saturation throughput of BLESS on the torus is 0.055, which is close to the buffered network's 0.066. We conclude that a torus topology is a good substrate for bufferless routing. Overall, our results show that BLESS has competitive latency with 2-VC buffered networks at

low injection rates in both torus and mesh topologies, and almost-competitive saturation throughput in the torus topology. Given that BLESS actually has higher frequency (2.56 times), lower network power consumption (by at least 38%), and lower area consumption (by approximately 62%) compared to the 2-VC router (as we showed in earlier sections), we conclude that BLESS can be an effective alternative to buffered networks.

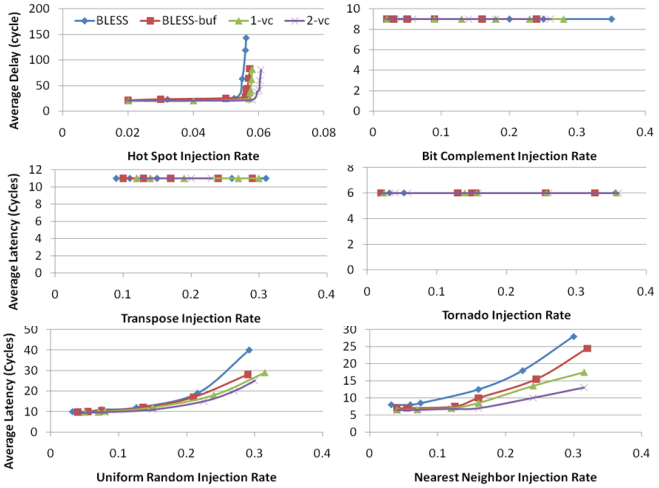

Fig. 13 Packet latency with various routers on the 4x4 torus

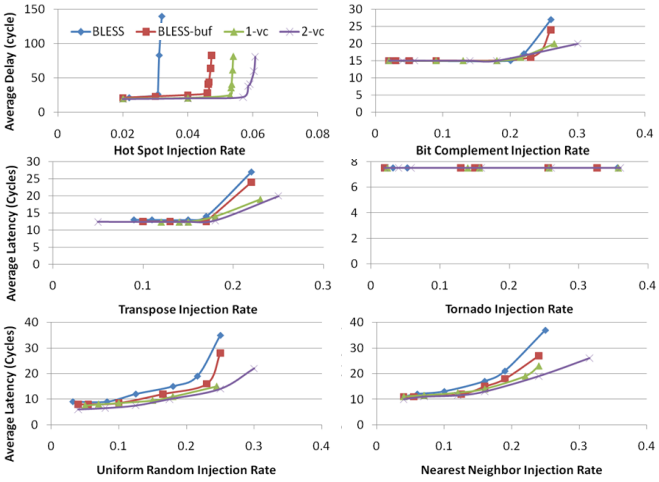

Fig. 14 Packet latency with various routers on the 4x4 mesh

F. ASIC Implementation Results

To explore the tradeoffs of different router microarchitectures in an ASIC implementation, we synthesized the same RTL Verilog design of each router we discussed above. Results for area, power, and frequency are derived from synthesizing the Verilog design to the TSMC 65nm standard cell library using the Synopsys Design Compiler with the highest level of optimization enabled. Table IV shows the results. Power numbers inside the parentheses indicate the power consumption of each router for the maximum frequency it can run at. For fair comparison with BLESS, power numbers *outside* the parentheses indicate the power consumption of each router, assuming all routers run at the same clock frequency as the BLESS router. Although the exact numbers of FPGA results are different from ASIC results, the relative relationships between different routers are the same. BLESS is still the most efficient design with 38% area and 30% power saving compared to the 1-VC router, the simplest buffered router. At the same time, with our balanced oldest-first arbiter, BLESS is still 8% faster than the 1-VC router. Similarly to the FPGA results, when the number of virtual channels increases, BLESS's area, power, and frequency advantages increase over the buffered routers.

Table IV: 65nm ASIC results for all the routers

Router	Area (μm^2)	Power (mW)	Max Frequency
BLESS	12353	2.14	214.6MHz (4.66ns)
BLESS (Buf)	16854	2.74 (2.61)	204.5MHz (4.89ns)
1-VC	20139	3.03 (2.79)	197.2MHz (5.07ns)
2-VC	36152	5.12 (2.40)	100.4MHz (9.93ns)
4-VC	79900	10.11 (2.37)	50.3MHz (19.9ns)

VI. Related Work

To our knowledge, this is the first paper that provides realistic and detailed FPGA and ASIC-process prototype network-on-chip implementations of bufferless routing and compares them extensively with that of buffered routing. We briefly describe related work in both buffered and bufferless routing domains.

On-chip network prototypes: Several prototype chip implementations incorporated on-chip networks. These include the MIT RAW chip [39], the UT TRIPS chip [20], the Intel 80-core TeraFlops Polaris chip [23], and the Tilera TILE64 processor [42]. All of these NoC implementations used buffered routers on a 2D mesh topology. Buffered on-chip networks were shown to consume 30% of the system power in the Intel 80-core TeraFlops Polaris chip [23] and 40% of the system power in the MIT RAW chip [39]. Gratz et al. [20] showed that router input buffers occupy 75% of the total on-chip network area in the TRIPS chip. While these works provided a good characterization of buffered routers, they have not implemented bufferless routers and their characteristics. Our work leverages the insights developed in these works to build a baseline state-of-the-art router, which we comparatively evaluate with bufferless routing in our FPGA-based network-on-chip prototype.

Recent bufferless routers: Recent works provided various algorithms for and forms of bufferless routing [30, 27, 18, 19, 11, 12, 13, 5, 2, 28, 14, 15, 32, 33], but most of these works did not study the feasibility and advantages/disadvantages in detail in an implemented bufferless router prototype as our work does.

Moscibroda and Mutlu [30] described livelock-free algorithms for bufferless routing and evaluated these algorithms in simulation of 2D mesh networks. They found that for low injection rates, the common case in most realistic applications, bufferless routing provides similar performance as buffered routing while reducing the on-chip network energy consumption by 40%. They performed back-of-the-envelope calculations to evaluate the area benefits of bufferless routing. However, they did not study the implementation complexity of bufferless routers. Our work builds upon [30] by implementing the algorithms developed in [30] for both 2D mesh and torus topologies, evaluating their complexity, area, power, and latency in a realistic implementation, and providing extensive comparisons to buffered routing. By using a real FPGA/ASIC implementation, we find that 1) the frequency of a bufferless router exceeds that of a buffered router, via careful design of the oldest-first arbiter, 2) bufferless routing provides significant area and power benefits over buffered routing on an FPGA/ASIC.

The closest work to ours is that of Michelogiannakis et al. [28], which evaluated bufferless and buffered routers using RTL implementations. They found that the bufferless routers are only marginally more energy efficient than a carefully designed buffered router. We believe our work provides another data point in open literature to complement their implementation and results by optimizing both the bufferless and the buffered router designs on the same FPGA or ASIC process.

Several other previous works [29, 27, 18, 19, 21, 40] also studied the use of bufferless routing in on-chip networks. [18, 19, 21] require packet dropping when congestion arises, a complexity that is not present in the bufferless routing techniques we implement and evaluate. [40] does not seem to provide livelock freedom. These previous studies mainly consist of simulation-based evaluation of deflection routing and packet dropping algorithms on performance. As such, they do not evaluate 1) the energy consumption, routing

latency, and area of deflection-based bufferless routers, 2) describe the implementation advantages/disadvantages encountered in designing BLESS and BLESS with buffers.

Improvements in bufferless routers: Recent works proposed mechanisms to improve the efficiency of bufferless routers by providing simpler livelock and deadlock freedom mechanisms as well as deflection routing implementations [12], source throttling based congestion control techniques [5, 32, 33], and the use of minimal buffering mechanisms [13]. Using these techniques can improve results we show for bufferless routers. Recent work also devised deflection based bufferless router designs for hierarchical ring networks [2, 14, 15], showing significant energy efficiency benefits, but did not evaluate such designs using real FPGA or ASIC-synthesis implementations. A summary of recent progress in bufferless deflection routing can be found in [11].

Deflection routing: Buffered or bufferless versions of deflection routing had been proposed for distributed systems [3] and used in massively parallel machines such as the HEP [36, 37], the Tera [1], and the Connection Machine [22] in their large-scale interconnection networks that connect different chips. These techniques are not disclosed in detail and, to our knowledge, have not been publicly evaluated in terms of energy consumption or performance. Some of these deflection routing algorithms do not eliminate buffers [38]. The Chaos router [26], which was proposed to connect multiple chips, uses a form of deflection routing when a node is congested, but it still buffers packets in the router. An implementation of the Chaos router was realized as an ASIC. However, its evaluation consisted of performance studies and did not quantify area, power consumption, latency optimizations, and complexity of buffered deflection routing.

Other buffered router prototypes: Many prototype routing chips with buffered routers were designed and evaluated for off-chip interconnection networks. Examples include the torus routing chip [8], the SGI routing chip [17], and TRIPS [20].

Buffered router optimizations: Virtual channels [7], wormhole routing [8], speculation [35], and other latency optimizations for buffered routers [31] have been studied in the past. Our buffered router design incorporates these optimizations, thereby providing a state-of-the-art baseline to which we compared bufferless routing.

Bufferless routing in theory and optical networks: Theoretical studies [16, 4] have evaluated static algorithms for deflection routing and did not provide implementations. Deflection routing has been implemented in optical transmission networks [41], which have very different energy and performance characteristics than the on-chip electrical networks we examine.

VII. Conclusion

Bufferless routing promises large reductions in network area, power, and complexity compared to buffered routing commonly employed in existing on-chip networks. This paper investigates if this promise translates to real benefits in on-chip network designs by comparatively evaluating bufferless and buffered network prototypes using FPGA and ASIC-synthesis implementations.

Our extensive experiments and analyses on different types of network topologies, different network sizes, and a large number of traffic patterns suggest that while bufferless routing can increase implementation complexity of routers by requiring more complex oldest-first arbitration, this complexity can be overcome with careful design of the arbiter. Overall, bufferless routing leads to significant power, area, and router cycle time savings in both mesh and torus topologies against buffered routing in real implementations. Two trends likely increase the advantages of bufferless routing. First, as path diversity in the network increases with richer topologies, bufferless routing starts performing similarly to its buffered counterpart. Second, the trend in existing network-on-chip design is to have multiple virtual channels. As the number of virtual channels increases in the buffered design, bufferless routing provides even higher frequency, higher area savings, and lower power consumption in real implementations. We conclude that bufferless routing can provide an efficient way of designing future interconnects.

References

[1] R. Alverson et al. The Tera computer system. ICS, 1990.
[2] R. Ausavarungnirun et al., Design and Evaluation of Hierarchical Rings with Deflection Routing. SBAC-PAD, 2014.
[3] P. Baran. On distributed communications networks. IEEE Transactions on Communications, Mar. 1964.
[4] C. Busch et al. Routing without flow control. SPAA, 2001.
[5] K. Chang et al. HAT: Heterogeneous Adaptive Throttling for On-Chip Networks. SBAC-PAD, 2012.
[6] S. Cho and L. Jin. Managing distributed, shared L2 caches through OS-level page allocation. MICRO, 2006.
[7] W. J. Dally. Virtual-channel flow control. ISCA, 1990.
[8] W. J. Dally and C. L. Seitz. The torus routing chip. Distributed Computing, 1986.
[9] W. J. Dally and B. Towles. Route packets, not wires: On-chip interconnection networks. DAC, 2001.
[10] W. J. Dally and B. Towles. Principles and Practices of Interconnection Networks. Morgan Kaufmann, 2004.
[11] C. Fallin et al., Bufferless and Minimally-Buffered Deflection Routing. Chapter in *Routing Algorithms in Networks-on-chip*, Springer, 2014.
[12] C. Fallin et al., CHIPPER: A Low-Complexity Bufferless Deflection Router. HPCA, 2011.
[13] C. Fallin et al., MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect. NOCS, 2012.
[14] C. Fallin et al., HiRD: A Low-Complexity, Energy-Efficient Hierarchical Ring Interconnect. CMU SAFARI Tech. Report, 2012.
[15] C. Fallin et al., A High-Performance Hierarchical Ring On-Chip Interconnect with Low-Cost Routers. CMU SAFARI Tech. Report, 2011.
[16] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. STOC, 1992.
[17] M. Galles. Spider: A High-Speed Network Interconnect. IEEE Micro, 17(1), 1997.
[18] C. Gomez, M. E. Gomez, P. Lopez, and J. Duato. A bufferless switching technique for NoCs. Wina, 2008.
[19] C. Gomez, M. E. Gomez, P. Lopez, and J. Duato. Reducing packet dropping in a bufferless NoC. Euro-Par, 2008.
[20] P. Gratz, C. Kim, R. McDonald, S. W. Keckler, and D. Burger. Implementation and evaluation of on-chip network architectures. ICCD, 2006.
[21] M. Hayenga et al., SCARAB: A Single Cycle Adaptive Routing and Bufferless Network. MICRO, 2009.
[22] W. D. Hillis. The Connection Machine. MIT Press, 1989.
[23] Y. Hoskote et al. A 5-ghz mesh interconnect for a teraflops processor. IEEE Micro, 2007.
[24] N. D. E. Jerger et al. Circuit-switched coherence. NOCS, 2008.
[25] J. Kim, J. D. Balfour, and W. J. Dally. Flattened butterfly topology for on-chip networks. MICRO, 2007.
[26] S. Konstantinidou and L. Snyder. Chaos Router: Architecture and Performance. ISCA 1991.
[27] Z. Lu, M. Zhong, and A. Jantsch. Evaluation of on-chip networks using deflection routing. GLSVLSI, 2006.
[28] G. Michelogiannakis et al., Evaluating Bufferless Flow Control for On-chip Networks. NOCS, 2010.
[29] M. Millberg et al. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. DATE, 2004.
[30] T. Moscibroda and O. Mutlu. A case for bufferless routing in on-chip networks. ISCA, 2009.
[31] R. Mullins, A. West, and S. Moore. Low-latency virtual-channel routers for on-chip networks. ISCA, 2004.
[32] G. Nychis et al., Next Generation On-Chip Networks: What Kind of Congestion Control Do We Need? HotNets, 2010.
[33] G. Nychis et al., On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects. SIGCOMM, 2012.
[34] M. Parlak et al. A novel computational complexity and power reduction technique for H.264 intra prediction. IEEE Transactions on Consumer Electronics, 2008.
[35] L.-S. Peh and W. J. Dally. A delay model and speculative architecture for pipelined routers. HPCA, 2001.
[36] B. J. Smith. A pipelined shared resource MIMD computer. ICPP, 1978.
[37] B. J. Smith. Architecture and applications of the HEP multiprocessor computer system. SPIE, 1981.
[38] B. J. Smith, Apr. 2008. Personal communication.
[39] M. B. Taylor et al. Evaluation of the Raw microprocessor: An exposed-wire-delay architecture for ILP and streams. ISCA, 2004.
[40] S. Tota, M. R. Casu, and L. Macchiarulo. Implementation analysis of NoC: a MPSoC trace-driven approach. GLSVLSI, 2006.
[41] X. Wang et al. Burst optical deflection routing protocol for wavelength routing WDM networks. SPIE/IEEE Opticom, 2004.
[42] D. Wentzlaff et al. On-chip interconnection architecture of the Tile processor. IEEE Micro, 2007.