Wormhole NoC Router

Sumit Mondal

***Abstract*— Network on Chip is the modern solution to existing and future interconnection problems in complex multi-processor chip designs. One of the key factors of an NoCs performance in terms of throughput and area is the flow control schema. This paper presents the register-transistor-level (RTL) design of a 2D Mesh NoC with wormhole flow control. Wormhole switching is a flit-based flow control scheme that increases buffer utilization and throughput performance. The functionality is evaluated using a modular testbench framework. Performance metrics, specifically reception rate, is evaluated for the design with various buffer depths.**

I. INTRODUCTION

Multi-Processors are becoming more ubiquitous, thus there is a need for scalable, adaptive and efficient flit-switched Network-on-Chip (NoC) designs. NoCs form the interconnection fabric connecting various components of a multicore system including compute units, register files, cache blocks and other IP cores. Router modules within NoCs implement various routing and flow control algorithms which allocate and move data at flit granularity. For performance reasons, designs of NoCs require small latency, high bandwidths, low power-consumption, and sustainable throughput. One of the key factors of an NoCs performance is the flow control schema, or how the NoC governs when and how messages stop and proceed.

This paper presents the design, RTL implementation and characterization of wormhole flow control on 2D NoCs. Wormhole switching is a flit level flow control algorithm that efficiently utilizes buffer space. Wormhole switching increases buffer utilization by allowing flits of a packet to continue to the next router as long as the downstream router has enough buffer space for that flit.

For the router we are designing, packets are routed using Dimension Ordered Routing (DoR). Routers use an on-off signaling mechanism to implement buffer backpressure between adjacent routers. The router we propose is a 3-stage router: Buffer Write, Route Compute/Switch Allocation, Switch Traversal/Buffer Read. The NoC in design will be a 4x4 mesh topology. Multiple lengths of wormhole buffers are tested for maximum performance. Single flit packets are used to verify the NoC design. A modular testbench framework is used to ensure functional correctness of each device under test (DUT) by monitoring assertion signals and using a status monitor module. Power, throughput and latency metrics observed for various traffic patterns will be used to determine the efficiency and scalability of the wormhole NoC design.

The rest of the paper is organized as follows. Section 2 provides an introduction to the design and design methodology, as well as an in-depth review of each of the design modules. Section 2 also includes some proposed designs for a wormhole router with Virtual Channels. Section 3 explains our verification methodology at the block level and router level. Section 4 describes the simulation environment for the 4x4 Mesh NoC and an analysis of the design's throughput. Section 5 discusses the future work. Section 6 describes the effort in the proposed designs. Section 7 concludes the paper. Section 8 outlines the individual contributions. Section 9 lists the references for the design, paper, and analysis.

II. DESIGN

Cores in a direct-network topology, such as the 4x4 mesh that we propose, communicate with each other by passing messages to each other or through intermediate nodes. Messages may be divided into one more equal sized flits, but in our design we focus on single-flit packets The unique aspect of our design is that the intermediate nodes switching messages is implemented using wormhole switching.

Wormhole switching is best explained by first describing simpler switching techniques. Circuit Switching describes a NoC that decides a dedicated path between source and destination of a packet before the packet is ever sent. Packet switching methods use information encoded in the packet to be independently routed to the proper destination. In packet-switching the buffers at each router and the links between them are allocated to the entire packet. Packet switching methods are suboptimal because buffer utilization will be low, buffer size will need to be large, and the latency between source and destination is forced to be proportional to the number of hops [2].

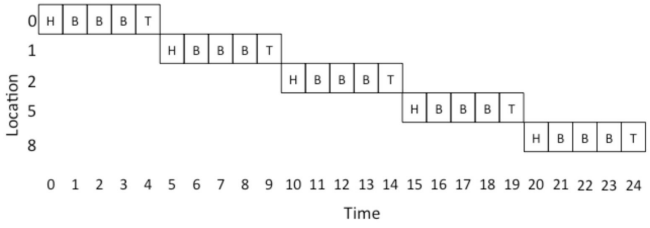

Figure 1. *Time-space diagram of a packet-switched routing algorithm [1].*

Wormhole switching allows for the buffers to be allocated at the flit level, meaning buffers can hold more than one message, or messages that are being routed to different destinations. The flits of a packet are relayed through the network in a pipelined fashion [2]. As long as there is space in the downstream routers buffer for the flit, it will be sent to the downstream router. Flits and messages will always be in order, i.e. flits of two different messages may not be interleaved. One disadvantage of wormhole switching is that if the header

packet of the leading message is blocked from entering a buffer, all trailing messages will be blocked as well. This can be resolved by smart use of virtual channels in the input port.

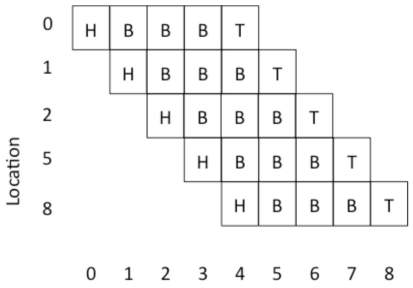

Figure 2. *Time-space diagram of a wormhole switching algorithm [1].*

To accompany worm-hole switching, the designers have decided to implement an on/off flow control protocol. An issue that arises is when a flit is unable to proceed to the following router due to its buffer being full, how should it be notified and what should it do. On/off flow control defines that the flit should wait at its current buffer, according to an ON/OFF signal from the downstream router, or more specifically a downstream buffer [1]. When the downstream buffer has space for a flit, the ON/OFF signal is high, and if it is full the ON/OFF signal is low. This works well when the round-trip latency is low, such as our design where it is defined to be two cycles.

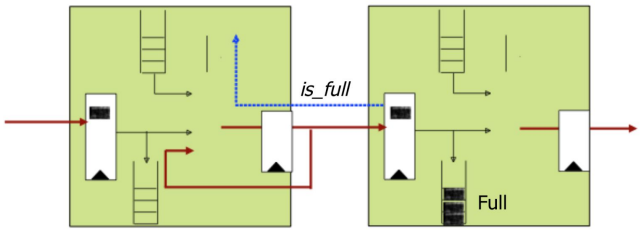

Figure 3. *Digital logic level description of on/off flow control with an is_full signal [1].*

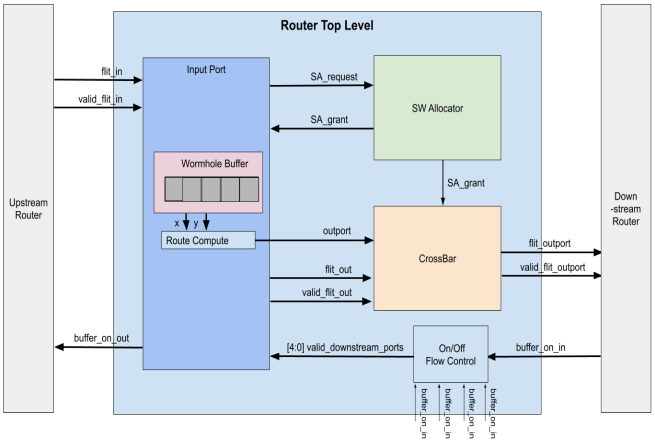

Figure 4. *Router Microarchitecture*

The rest of this section outlines block level explanations of the proposed Wormhole NoC router according to this diagram.

A. FIFO/Wormhole Buffer

The data structure we designed to hold flits in the router design is a circular buffer. This FIFO circular buffer allows single-cycle read, write, and read&write operations. The initial design has a FIFO depth of 16, and a FIFO width of 64, or corresponding to the flit size. The depth will vary in the simulation results.

The FIFO buffer module has an on/off credit system to signify to upstream routers if the buffer is able to accept more flits. This on/off credit system is designed with a 2-cycle credit latency in mind. Essentially, this means that if the buffer is 2 flits away from being full, the on/off signal tells the upstream router that this buffer is not able to accept any more flits.

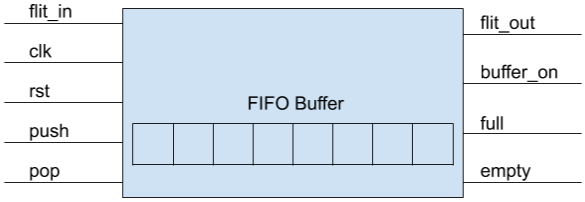

Figure 5. *FIFO Buffer Module*

B. Switch Allocator

The Switch Allocator module handles movement of flits from input buffers to the shared crossbar switch. Without VC's, the contention between the input ports and the crossbar switch is fairly simple and can be implemented solely by using a round-robin arbiter. This arbiter module implements a round-robin scheme to resolve contention among virtual channel buffers requesting access to output ports. After allocating a resource to one of its requestors, it upgrades the priority of all requesters in a circular fashion. This scheme was chosen for its simplicity, fairness and to avoid traffic pattern dependent starvation. The round-robin arbiter was largely taken from FreeCores, http://freecores.github.io/, an open source hardware design library, with minor adaptations to fit our 5 port design.

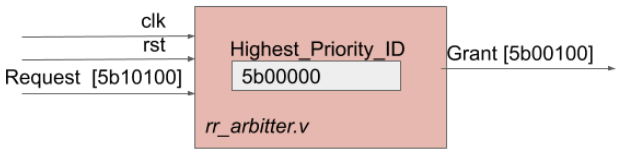

Figure 6. *Round Robiter Arbiter Module*

C. *Crossbar Switch*

The Crossbar switch is a combinational logic block which connects and moves flits, and their corresponding valid signals, from the input block to the specified output port. Therefore, each output port connects and propagates at most one input flit, from all of the 5 input ports.

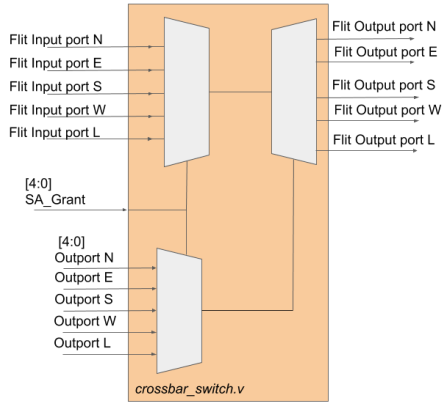

Figure 7. *Module Level diagram of Crossbar Switch*

D. *Input Port*

The Input Port module holds the FIFO buffers, which may act as Virtual Channels, and surround the fifo buffers with routing information. Each input port (i.e north, south, east, west, and local) is connected to the Switch Allocator, the crossbar, the internal router, the neighboring input port, and the neighboring router.

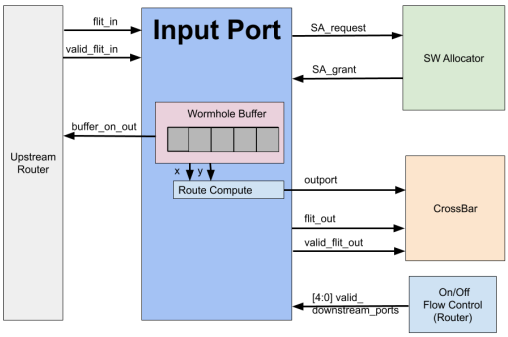

Figure 8. *Input Port Module connection map*

Only one flit can be pushed to the input port module per clock cycle. Only one flit can be read from the FIFO buffer module per clock cycle. The Input Port is capable of outputting a valid flit once per cycle, as part of the 3 stage pipeline/FSM design.

Section	Bits	Description
Route	4	4x4 Mesh requires 2 bits each for x and y coordinates
Type	2	Head, Body, or Tail flit designation
VCid	2	2 bits for up to 4 VCs
Payload	56	Payload data. Made so that flit size is 64-bit

Table 1. Flit Encoding Schema

The router in-port is a sort of mixed design between a pipeline and an internal finite state machine. Constantly, the input port is pushing valid flits into the FIFO buffer, while the state machine cycles through three states: BW(Buffer Write), RC_SA(Route Compute and Switch Allocate), and ST_BR(Switch Traversal and Buffer Read).

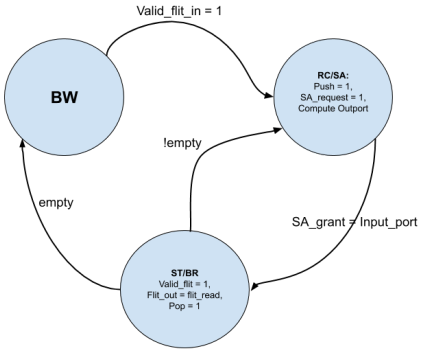

Figure 9. *State Diagram for Router In-port*

The Buffer Write stage is only entered when the FIFO buffer was previously empty, otherwise the state machine mainly cycles between SA and ST. In the SA stage, the input port design reads the output of the FIFO buffer (where the FIFO read pointer is pointing), computes the route and output port for the flit based on Dimension Ordered Routing, and sends a switch request and the output port to the Switch Allocator. Once the switch allocator gives a grant to the specified input port, and the corresponding downstream buffer is on (in reference to the computed outport), then the FSM changes state to ST/BR. In ST/BR, the flit is sent out with a valid signal to the Crossbar, and the flit is popped from the FIFO buffer. If the buffer is empty afterwards, it enters BW, otherwise it transitions back to RC/SA state.

Route Computation

The route computation logic is internal to the Input port. Specifically, it lies inside the RC_SA stage of the FSM. The logic computes the next hop, or the output port/direction for the head flit as it arrives into the Input Port. This logic computes the Direction Ordered route for each packet, and

gives this information to the Crossbar, and compares the output port to the switch allocator's grant.

E. Router Top level

The router module implements 5 input-blocks (North, South, East, West, and Local), a Switch allocator and a Crossbar switch module. This module also takes in all the downstream buffer_on signals, and concatenates them to create a full list of valid downstream buffers to supply to all of the 5 input ports. The router module outputs 3 signals and inputs 3 signals in each of its directions (N,E,S,W,L). These signals are the flit_inport,flit_outport,valid_inport,valid_outport, and the buffer_in, and buffer_out signals. As long as there is no back pressure being applied to the router, meaning all the downstream buffers are on, the router is capable of producing a valid output flit once every cycle. The initial latency of producing a valid flit is 3 cycles: 1. Buffer Write, 2. Route-Compute/Switch Allocate, 3. Switch Traversal/Buffer Read.

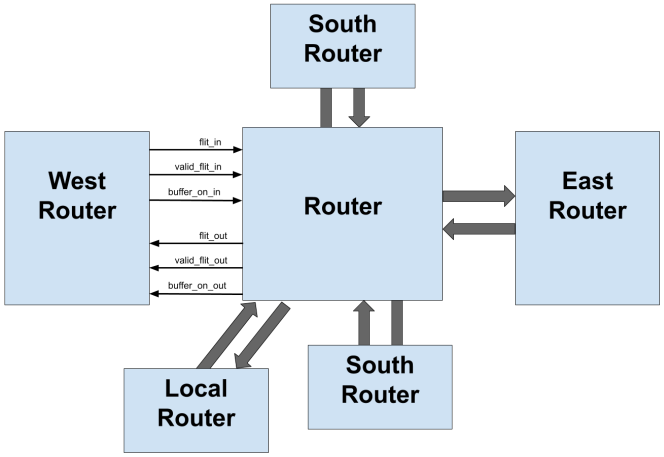

Figure 10. *Router Top Level Module*

A 2D mesh topology can be implemented by connecting router ports in a 2D grid fashion.

III. VERIFICATION

Functional verification is performed on all of the design modules listed in Section II. Testbenches have been created to test specific modules and their corresponding functional correctness with respect to the design specifications. So far, the team has only completed the following designs and testbenches.

A. FIFO Buffer

The FIFO buffer testbench ensures that the circular buffer is able to perform 3 operations in a single-cycle manner: read, write, and read&write. The testbench performs one of each, then continues on to test 20 random variations of the 3 operations. The testbench ensures that the FIFO operation is correct, such that the flits being popped are in the same order that they are pushed.

Figure 11. *Simulation output of the Circular Buffer module.*

The figure above represents a shortened testbench with only 5 total operations. It is verified that the buffer can read and write data in order according to the input push and pop signals.

B. Round-Robin arbiter

The round robin arbiter testbench initializes a on-off encoded requestor priority queue. The arbiter module should loop through the queue and generate a grant priority queue with the next highest priority bit set to 1. The output of the grant priority queue is monitored over 50 clock cycles. At most one resource is allocated every clock cycle.

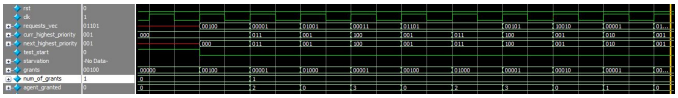

Figure 12. *Simulation output for Round-robin arbiter*

As can be seen in Figure , the "highest_priority" counter keeps track of the last granted agent. Based on the current incoming request bit stream, the grant is given to the requestor with an ID that is higher than the value stored in "highest_priority".

C. Router In-Port

The in-port testbench is very similar to the circular buffer testbench in that it ensures that flits enter and exit the port in order, but the extra is that it ensures that the state-machine never enters an unknown state and that the buffer_on and vc_valid signals are sent and processed properly. The buffer_on is set high when the circular buffer is full, sending it to upstream routers, and the vc_valid signal must be obeyed from the downstream router to ensure that the packets can be sent out from the in-port module

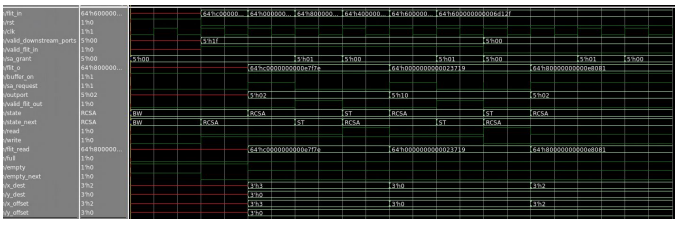

Figure 13. *Simulation output for router in-port module*

The figure above shows the output for the testbench, and verifies that the flits are being sent in and out in order. It also verifies that if the valid_downstream_buffers does not agree

with the selected output port, a flit_valid signal is never sent out, and rather the FSM re-enters the SA state to re-contest.

D. Router Level Testbench

The top-level testbench verifies that flits can enter the router from multiple directions, arbitrate properly and are outputted to the correct output port in a 3-cycle pipelined fashion.

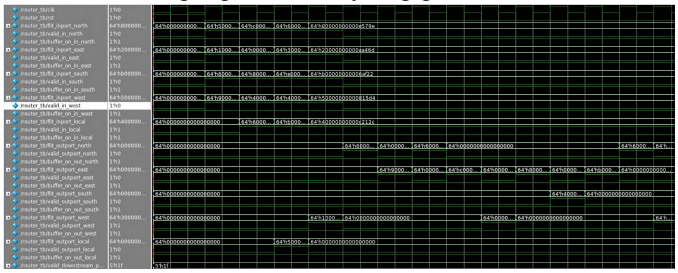

Figure 14. *Simulation of Router Level testbench.*

This testbench verifies that these 20 packets that enter (5 packets per port) arbitrate and exit the router properly to their designated output port. It also verifies that the router is capable of outputting one valid flit per cycle once the pipeline begins.

IV. Mesh Simulation

A. Throughput

The following simulation results are for a 4x4 Mesh topology created by connected 16 of the router modules described in Section 2.E.

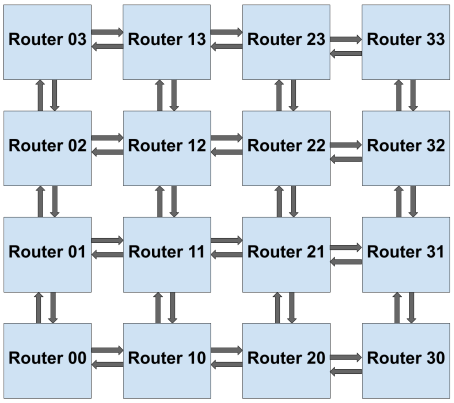

Figure 15. *Mesh 4x4 arrangement of the wormhole routers*

The reception rate is calculated by the following equation:

$$reception\ rate = \frac{total\ flits\ received\ by\ the\ network}{number\ of\ cores * number\ of\ cycles}$$

The total number of cycles for the following simulations is 5000. Traffic Injection rate is defined as packets/cycle/node. In the following plot we simulated uniform random traffic. The total received flits by the network was calculated using a scoreboard in a SystemVerilog testbench.

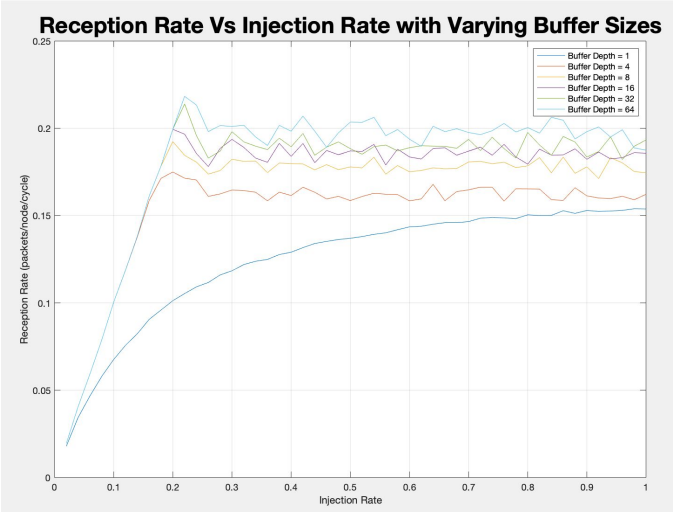

Figure 16. *Reception Rate vs Injection Rate with varying Buffer sizes for an RTL Wormhole 4x4 Mesh Network*

There are slight benefits to increasing the buffer depth compared to the reception rate of the network. At a buffer depth of 8, the peak reception rate is approximately 0.175. The peak reception rate of a buffer depth of 64 is approximately 0.22. The smallest reception rate is when the Mesh network is simulated with a buffer depth of 1, with a max reception rate of approximately 0.15. Peak theoretical throughput for a 4x4 Mesh is 1 flit/cycle/node.

These results are fairly consistent with those gathered by Garnet 2.0 with the same simulation environment: 16 CPUs, wormhole switching, and uniform random traffic. For example, peak reception rate with wormhole and buffer depth equal to 1 in garnet has a peak reception rate of approximately 0.13. For buffer sizes of 8 and 64 with wormhole routing, garnet produces a peak reception rate of approximately 0.19.

Further analysis of the RTL simulation shows that more often than not the ports that are affected by head-of-line blocking are the east ports of all routers_1x and the west ports of all routers_2x. For most simulations the testbench declared which buffers were turned off and most often the buffers that were turned off by the simulation were on the middle link from east to west or vice versa. This makes sense as one would expect the middle crossing to have the maximum traffic, especially east/west traffic due to dimension ordered routing always routing packets east and west first.

V. Future Work

A simple adaptation to our design would be a less restrictive routing algorithm, such as west-first or north-last. The issue with these routing algorithms is that you cannot easily synthesize these algorithms because they require a random choice. A random choice in synthesizable hardware must be created from a linear-shift feed-back register (LSFR); one of these modules may be available from FreeCores.

A more thorough adaptation to the design would be the introduction of virtual channels. One of the key issues with wormhole routing is head-of-line blocking [2]. If the packet at the front of the buffer is blocked from exiting at its destined output port, all trailing packets in the buffer must wait as well. This scenario can lead to worsened network throughput and even deadlock. An implementation of our design that instantiates multiple FIFO buffers within an input port could resolve these issues. Messages that would have been stopped from head-of-line blocking can now take another path through an input port to find its way to the destination. This would require much more complex circuitry to arbitrate and allocate all of the virtual channels within the router.

VI. Conclusion

Network-on-Chip is becoming the norm for interconnection fabrics for multi-core processing systems. The router is a key component for direct network topologies that switch packets from node to node. The design proposed by the team of engineers is a wormhole switched router. Wormhole switching increases throughput of packets in an interconnection network on chip by pipelining flits through the use of input FIFO buffers on the router. The wormhole router designed in this paper is a 3 stage router: Buffer Write, Route Compute/Switch Allocate, and Switch Traversal/Buffer read. The router is capable of outputting valid flits in one of 5 directions per cycle. The team of engineers simulated the wormhole router in a 4x4 mesh network, with uniform random traffic, and was able to gather reception rates between 0.15 flits/cycle/node to 0.22 flits/cycle/node depending on the buffer depth of the wormhole router. One of the major causes for not being able to increase the throughput to the theoretical peak of 1 flit/node/cycle is the head-of-line blocking caused by the wormhole routing. The design can be improved by the introduction of multiple Virtual Channels per input port on the Router. In conclusion, the design team was able to create an RTL description of a VC-less wormhole router with an accompanying 4x4 mesh simulation environment.