RTL implementation of HopliteRT

Marat Bekmyrza

I. Introduction

NoCs (network-on-chip) provide a low-cost, high-throughput communication interface for systems with multiple PEs (processing element). Hoplite [1] is an FPGA-based NoC that uses *deflection routing* to reduce the implementation cost of switches and remove the need for expensive buffers. The idea is to intentionally mis-route one of the packets conflicting over an output port of the switch. In this design there is a possibility that an unlucky packet might always get mis-routed and never reach its destination. Such behaviour is not acceptable for real-time applications that require guaranteed worst-case latency. Moreover, bufferless setup forces the switch to give the lowest priority to PE injection port so that valid packets from other ports are not lost. This might cause infinite source queueing delays for an unlucky PE which is always stalled due to continuous traffic.

HopliteRT [2] aims to remove such livelocks and introduce deterministic upper bounds on worst-case routing latency with the following modifications:

- Switch side: new routing policy with no extra LUTs required.
- PE side: two counters to enforce token bucket regulation policy at injection ports.

This report describes RTL implementation of HopliteRT modules: torus (top module), switch, and processing element wrapper (PE with two counters).

II. Implementation

A. Torus

HopliteRT topology is an unidirectional torus as shown in Fig. 1.

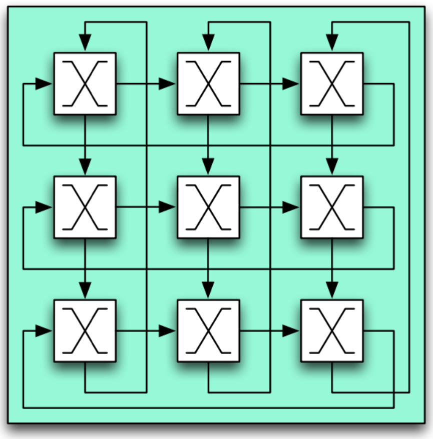

Fig. 1. NoC topology: 2D Unidirectional Torus [1].

Each node in Fig. 1 contains a switch and PE wrapper with data transfer lines shown in Fig. 2.

In torus topology, Hoplite uses Dimension Ordered Routing (DOR) policy: packets traverse first in the X-ring (horizontal) towards desired X-coordinate, and then in the the Y-ring (vertical) until the destination Y-coordinate is reached.

B. Switch

In Fig. 2, each data transfer line includes data itself, valid signal and destination address of the packet. Switch module has three inputs for data transfer W (West), N (North), and PE and two outputs E (East) and S (South). S is also an input to PE wrapper. Switch contains a dual 3-to-1 mux, shown in Fig. 3, that routes inputs to corresponding output ports based on arbiter logic.

Microarchitecture in Fig. 3 needs two 6-LUTs due to 4 select bits + 3 input data bits. However, [2] suggests to restrict some of the routing options to reduce select signal width down to 2 bits (reuse them for both muxes with different interpretations) and map the design to two 5-LUTs (which is one fracturing 6-LUT).

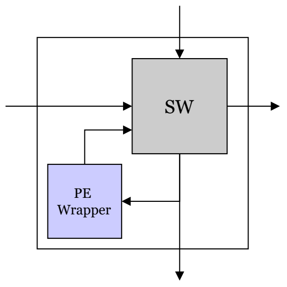

Fig. 2. Data transfer lines of Switch and PE Wrapper.

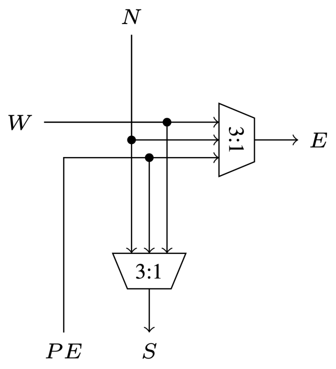

Fig. 3. Dual 3-to-1 mux microarchitecture.

Baseline Hoplite [1] had no path from N to E. Thus any W packet was always mis-routed when there was a conflict with N over S. HopliteRT [2] adopts new routing policy that allows N to E path, Fig. 3. The key idea now is to always prioritize W traffic over N traffic. This allows X-ring packets to be conflict free, while Y-ring traffic can get deflected. However, deflected Y-ring packet will have higher priority over any other Y-ring traffic once it deflects onto the X-ring. In this way, packet deflects at most once on each row. While it could get deflected infinitely many times in the baseline design.

Table I and Table II show the arbitration logic of HopliteRT routing policy. In Table I, 'v' means valid, 'to' means wants, and 'x' means don't care: e.g., W.v – W packet is valid, W to S – W packet wants to turn into S output. This design has no PE→E + W→S route to accommodate switch mapping on two 5-LUTs by sharing common mux select signals. PE packets have the lowest priority and stall on conflicts.

Explanation of routing policy shown in Table I:

i. W is valid and wants to turn S. Since W has the top priority, it gets the S port despite any conflicts. The only possible route is W→S + N→E.
ii. W is valid and wants to go E, i.e. W to S is 0. Again it has the top priority and gets the E port. However, now we have two possible routes with W→E: N→S or PE→S. Choice is N→S since N is valid and has higher priority.
iii. W is valid and wants to go E. It gets E port. Among two possible routes, choice is W→E + PE→S since N is not valid.
iv. W is not valid and N is valid. Thus, N is allowed to go S (no deflections): PE→E + N→S.
v. W and N are not valid, while PE is valid and wants to go S. It is allowed to go S, thus: PE→S + W→E.
vi. PE is valid, but wants to go E. It is allowed to go E, thus: PE→E + N→S.
vii. Don't care since all not valid.

Switch generates output data valid signals based on who gets the port depending on routing policy and whether the packet has arrived at its destination. When packet arrives and gets the S port (which is the PE exit), valid for PE input is set high and S port valid is set low. Switch also provides ready signal for PE indicating that it is ready to receive PE data when PE data is valid and it is allowed to go towards its desired port.

TABLE I
ROUTING POLICY

	W.v	N.v	PE.v	W to S	PE to S	Route
i	1	x	x	1	x	W→S + N→E
ii	1	1	x	0	x	W→E + N→S
iii	1	0	x	0	x	PE→S + W→E
iv	0	1	x	x	x	PE→E + N→S
v	0	0	1	x	1	PE→S + W→E
vi	0	0	1	x	0	PE→E + N→S
vii	0	0	0	x	x	x

TABLE II
MUX SELECT

sel1	sel0	Route
0	0	W→E + N→S
0	1	W→S + N→E
1	0	PE→E + N→S
1	1	PE→S + W→E

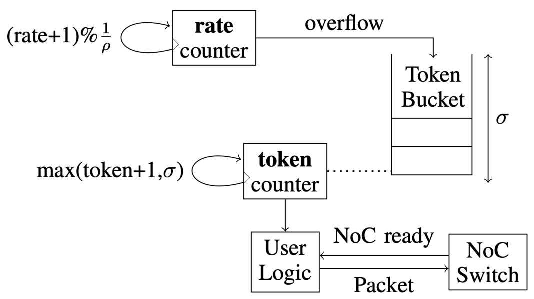

Fig. 4. Cascaded counters that realize token bucket regulation policy [2].

C. Processing Element Wrapper

PE wrapper contains PE and two counters. PE communicates with its switch, in Fig. 2, using AXI interface protocol. Switch waits for valid signal from PE before asserting its ready signal. Therefore, PE should keep valid stable until the transfer occurs. Transfer occurs when both valid and ready are asserted. PE is permitted to change the information at the next clock cycle after data transfer.

Two cascaded counters realize tocken bucket regulator at the PE injection port. Parameters of the regulator: ρ - rate of packet injection $<$ 1 packet/cycle) and σ - maximum burst of consecutive packets $\geq$ 1. First free-running rate counter overflows after reaching $1/\rho$ and at each overflow one token is added to token counter. This second token counter saturatues at σ. PE is allowed to send a packet only when NoC is ready and it has an available token. After transfer occurs, token counter is decremented.

III. EXPERIMENTS

A. Testing Switch

Switch module behaviour was tested for individual functional correctness using Verilog testbench. Memory file with different inputs for each port (W, N, PE) was created using Python:

- 8 different valids for {w.v, n.v, pe.v}.
- 4 different destination addresses: to local PE (same X, same Y), to S (same X, different Y), to same row E (different X, same Y), to different row E (different X, different Y).

Outputs from each port were recorded in memory file and tested using Python:

- W packet was transferred to correct port {W.v, W.addr}:
  - W wants to S. If arrived, should be sent to local PE and S.v = 0. Else, should propagate to S.

    - W wants to E. Should propagate to E.
- N packet was transferred to correct port {N.v, N.addr}:
    - W allowed to S. If arrived, should be sent to local PE and S.v = 0. Else, should propagate to S.
    - W not allowed to S. Should propagate to E.
- PE packet was transferred to correct port {PE.v, PE.addr}:
    - PE wants to S and allowed to S. Should propagate to S.
    - PE wants to E and allowed to E. Should propagate to E.

All the tests were passed successfully.

B. Testing Torus

Torus behaviour was also tested for functional correctness by generating random traffic from each PE. Dimensions of torus was 4×4. Input memory file with random destination addresses and unique data was created for each PE: to S direction short and long distance, to E same row and different row, and invalid data. Information about received packets from each PE was collected and tested for the following points:

- Number of packets send by a PE should be the same as indicated in input memory file.
- PE should not receive a packet that was not sent by anyone.
- Packets sent by PE should be received by the destination.
- PE should not receive a packet multiple times.

All the tests with variable ρ and σ parameters were passed successfully.

IV. Conclusion

This project develops RTL implementation of HopliteRT NoC designed for real-time applications. HopliteRT provides upper bounds on worst-case routing latency by adopting new routing policy and two additional counters at PE side. In this work, restricted version of router mux was used. For future work, the larger mux could be tested. More importantly, a way of recording packet latencies should be integrated and tested under various network traffic strategies.

References

[1] N. Kapre and J. Gray, "Hoplite: Building austere overlay NoCs for FPGAs," in International Conference on Field-Programmable Logic and Applications (FPL'15), 2015, FPL, 1–8.

[2] S. Wasly, R. Pellizzoni, and N. Kapre, "HopliteRT: An efficient FPGA NoC for real-time applications," in International Conference on Field Programmable Technology (ICFPT'17), 2017, IEEE, 64–71.