A Bandwidth-Optimized Routing Algorithm for Hybrid FPGA Networks-on-Chip

Shivukumar B. Patil, Tianqi Liu, and Russell Tessier
Department of Electrical and Computer Engineering, University of Massachusetts, Amherst, MA, USA 01003

***Abstract*—In this paper, a heuristic routing algorithm that is tuned for routing traffic in hybrid FPGA NoCs is presented. This multi-iteration routing algorithm requires a limited amount of hardware for prescheduled stream-based routing while allowing for bandwidth-optimized usage of NoC routing resources. By efficiently scheduling data streams, the remaining NoC bandwidth can be used for bursty, packet-switched traffic. We demonstrate our approach using a hybrid NoC and show an average 11% data stream bandwidth improvement for a collection of five benchmark traffic patterns.**

I. Introduction

The need for both hard and soft FPGA NoCs has been well-documented and a number of NoC architectures have been developed. Packet-switched (PS) NoCs (similar to those used on multi-core microprocessors) are well-suited to bursty traffic with unpredictable destinations. Time division multiplexed (TDM) NoCs route stream-based traffic using a schedule derived at compile time. In general, TDM NoCs exhibit reduced latency and energy consumption versus packet-switched NoCs since router buffering can be largely eliminated. This savings does come at the cost of some per-router storage for the schedule memory. Recently, hybrid FPGA NoCs [1][2] have been introduced that support both PS and TDM routing. Since TDM routing is performed at compile time and PS routing is performed at run time, care must be taken during TDM routing to provide low latency and high throughput for stream-based traffic without blocking additional PS traffic.

The selection of NoC channels and time slots for each TDM route requires an extensive spatial and temporal search even if routes are constrained to shortest paths. The lack of buffering for TDM routes requires time slots to be allocated sequentially. Additionally, the use of routing resources should be balanced across the network to allow for sufficient time slots for PS routing at run time. Previous work on TDM routing algorithms for FPGA NoCs [1][2] generally focuses on algorithm speed rather than optimizing the amount of TDM bandwidth or balancing routing resources by minimizing channel congestion. Given the amount of compile time needed for physical design of the FPGA logic, allocating a few seconds of additional compile time to obtain higher-bandwidth TDM routes can be an effective goal.

Our new TDM routing approach for hybrid NoCs maximizes and balances TDM bandwidth by using a multi-iteration one-step routing algorithm. For each routed TDM flit, the algorithm performs a breadth-first search that simultaneously considers both physical routing channels and time slots. A key aspect of the algorithm is its use of a Pathfinder [3] approach to ripup and reroute. Wire and time-slot overuse in a specific iteration is reflected in a non-decreasing cost value that can be used to gently guide routes away from a resource during later iterations. Our TDM routing algorithm is applied to a hybrid FPGA NoC for five widely-used NoC routing patterns.

II. Background

FPGA NoCs have several characteristics that differentiate them from multi-core microprocessor NoCs. Since FPGA cores are expected to be simple, transmitted data values are expected to arrive in order at the receiver. This issue generally results in shortest-path routing for TDM communication and communication protocols with predictable paths (e.g. dimension-ordered routing (DOR)) for PS routing [4].

Most previous compile-time TDM approaches first identify the routing channels used by paths and then select routing time slots. This two-step approach is easy to implement but can lead to suboptimal results. For example, Lu and Jantsch [5] use a depth-first search to select routing paths followed by scheduling. In Carle *et al.* [6], TDM routes are limited to DOR to reduce route search complexity. A previous hybrid FPGA router [1] can support simultaneous TDM and PS routing although all traffic is limited to DOR patterns. Multiple TDM routing approaches have combined channel with time slot selection. These algorithms greedily identify feasible solutions. Kapre *et al.* [7] attempt to schedule routes using the first available time slot. Multi-iteration ripup and reroute is mentioned but not implemented. Shpiner *et al.* [8] use a single-iteration random-greedy scheduling algorithm to determine route time slots. Evain and Diguet [9] use a space-time route allocation algorithm to minimize the TDM schedule.

In contrast, our approach combines path and time slot selection in a single pass. TDM routes are not restricted to DOR; they can take any available shortest path from source to destination. The approach uses multiple rip-up and reroute passes to eliminate the effects of net ordering. Although our TDM routing algorithm could be applied to any FPGA NoC that supports TDM, it is optimized for hybrid NoCs with both TDM and packet-switching. A router from a typical FPGA hybrid NoC [2] is shown in Fig. 1. The router contains a context memory for TDM schedule information that can configure the crossbar to connect a specific source port to each output port (e.g. S, N, ..). Output ports with a $-$ in the context memory on a specific cycle instead accept PS data. If a TDM connection is allocated and no input data is available, PS data

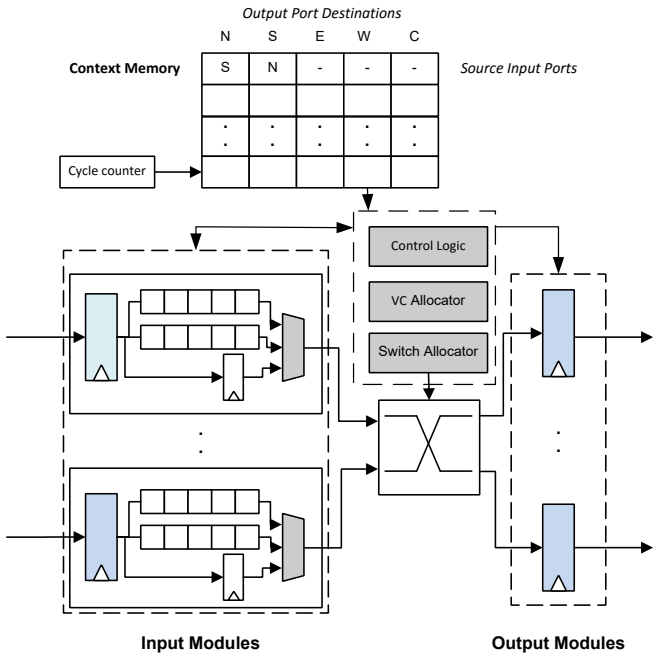

Fig. 1: Router for Hybrid FPGA NOC [2]

may be forwarded instead. Each *input module* contains two sets of FIFOs that serve as PS buffers and a bypass register for TDM data. Packet-switched routing is performed using DOR. Since the TDM schedule length is short (eight time slots), effective routing heuristics are needed to efficiently use available bandwidth.

III. Routing Algorithm Implementation

Our multi-iteration routing algorithm attempts to maximize TDM routing up to the allocation requested by the user at compile time. Unused routing time slots can be used for packet-switched routing. Our algorithm consists of three parts illustrated in Algorithms 1, 2, and 3. Each FPGA source component (e.g. soft processor, IP core) communicates with one or more destination components forming a *connection*. Each connection supports one or more periodic data transmissions (e.g. flits). To provide comparisons to PS routing, four flits are considered a packet.

Algorithm 1 shows the top-level loop of data communication scheduling. An attempt is made to schedule a route for each connection beginning at a *starting time slot* at the source router. After paths consisting of (inter-router channel, time slot) assignments are made for each connection, a check for route success is performed. A successful schedule includes no inter-router channel, time slot pairs that are shared by multiple connections. If unsuccessful, paths for connections without overlap are locked and another set of routing attempts for the remaining unrouted connections are performed beginning at the next starting time slot (e.g. starting time slot+1). The process terminates when all connections have been successfully scheduled with no overlap or no schedule can be found for them. In the latter case, unrouted connections are routed at run-time using packet switching.

Data: List of multi-fanout connections (input to output)
Result: List of routing paths (S) for every connection
/* Try all starting time slots */
for i=0 to num_slots **do**
  unrouted_set = all connections; start slot = i
  for j = i, j < num_slots, j++ **do**
    unrouted_set = *perform_set_route*(unrouted_set, j)
    if no shared (channel, time slot) paths **then**
      break
    end
  end
end

Algorithm 1: Traffic path algorithm

Data: Set of unlocked connections, starting slot j
Result: List of routing paths (S) with no (channel, slot) pair overlaps
Result: Set of connections with (channel, slot) pair overlap
Pathfinder_ts(j, set)
Lock all connections with paths with no (channel, slot) pairs that overlap
set = set - locked connections

Algorithm 2: Route a set of unlocked connections *perform_set_route*

Algorithm 2 illustrates the use of our Pathfinder_ts algorithm that considers both spatial (channels) and temporal (time slots) information. One application of Pathfinder_ts is performed per starting time slot. Connections that are successfully routed are locked.

Algorithm 3 forms the heart of our route scheduling approach. Each source-destination connection i is built from a series of channel, time slot pairs along a shortest path. Multi-fanout connections are supported and multiple iterations of rip-up and reroute are performed. As a route proceeds from source to destination, the cost of a new channel, time slot pair is considered. The lowest cost pair is selected at each step to create the lowest-cost scheduled path with cost C_i.

$$c_n(ts) = (1+congestion_{ts} \times p_{fac})(1+history_{ts} \times h_{fac}) \quad (1)$$

The Pathfinder cost function [3] in (1) determines the cost of using an adjacent node (channel, time slot pair) in Algorithm 3. Congestion cost ($congestion_{ts}$) of a node is the number of routing paths that are currently sharing it. History cost ($history_{ts}$) is a non-decreasing value which is increased by the amount of congestion cost at the end of each iteration. Values p_{fac} and h_{fac} are congestion cost and history cost multiplication factors, respectively. They are set to constant values of 1.2 and 0.2, respectively, as determined via experimentation.

Data: List of connections (source, destination, and starting time slot)
Result: List of routing paths (S) for each connection of channel, time slot pairs

while shared (channel, slot) pairs exist and (iteration < max_iter) or iteration = 0 **do**
  for all connections i starting at time slot $i.slot$ **do**
    Rip up existing connection path P_i
    $P_i \leftarrow$ (source, i.slot)
    Initialize expansion_list = **new** MinHeap()
    while all destinations d_{ij} not found **do**
      Remove lowest cost node m with time slot j from expansion_list
      while fanouts n of node m at time slot (j+1) mod max_slot on shortest path exist **do**
        Add n to expansion_list at cost c_n + C_i
      end
      for all nodes n in path from d_{ij} to source **do**
        Update c_n
        Add n to P_i
      end
    end
  end
end

Algorithm 3: Pathfinder_ts algorithm including time slots

Topology	64-node, 8 × 8 2D mesh
Technology	45 nm at 1.1V, 1.0 GHz
Channel width	128 bits (16 bytes)
Packet size	4 flits
Virtual channels	2/port
Buffer size per VC	10 flits in depth

TABLE I: Routing Parameters

IV. Experimental Approach

To evaluate the performance of our scheduling algorithm on a hybrid NoC, Booksim 2.0 [10] was used to assess data throughput, packet latency and NoC dynamic power. The simulator was previously modified to allow for hybrid TDM and packet-switched routing and power measurements [2] and a physical router layout was used to obtain NoC physical parameters [2]. Each router component and the inter-router wires were assigned dynamic energy consumption values determined via the post-synthesis simulation described in [2]. These values were scaled by flit-level toggle rates determined during the Booksim simulations. The network was warmed up with enough packets to reach a stable state prior to results recording. Unless otherwise stated, the NoC parameters shown in Table I were used for experimentation. All experiments use a routing schedule of eight time slots and 500 Pathfinder_ts iterations.

Using our modified version of Booksim, three communication patterns for hybrid routing were considered [10]. The patterns include 1) transpose (messages from (x, y) are sent to (y, x)), 2) bit-reverse (messages from the node labeled x are sent to the node whose label is *bit-reversed*(x), and 3) tornado (messages from (x, y) are sent to $(x + \frac{k}{2} - 1 \bmod k,\ y + \frac{k}{2} - 1 \bmod k)$ where k is the dimension of x and y. Results are also reported for 2-side (sources and destinations on parallel edges of the NoC) and 4-side (sources and destinations along the NoC perimeter) traffic patterns [2] at throughput rates of 10 Gbps for each source-destination pair. Combined execution of Algorithms 1, 2, and 3 requires on the order of seconds.

Pattern	Single Iteration				Pathfinder_ts			
	BW %	Min/ Max	Ave	σ	BW %	Min/ Max	Ave	σ
BitRev.	53	1/8	4.3	1.8	55	3/8	4.4	1.6
Tornado	28	1/5	2.2	0.8	31	2/3	2.5	0.5
Transp.	56	2/8	4.5	1.9	56	3/8	4.5	1.7
2-Side	71	5/7	5.7	0.7	99	7/8	7.9	0.2
4-Side	79	5/8	6.5	0.9	100	8/8	8.0	0.0

TABLE II: Time slot routing results of single iteration shortest path (SP) and Pathfinder_ts (PF) algorithms for single-source, single-destination traffic patterns.

V. Results

In this section, the routing results from using our Pathfinder_ts routing algorithm are directly compared against results using a single iteration shortest-path (SP) TDM approach [2]. The latter algorithm is similar to the routing algorithm described in [1]. All routes for TDM follow shortest paths.

The routing results shown in Table II illustrate the benefits of multi-iteration path selection and scheduling for TDM routing versus the previous routing approach. Pathfinder_ts (PF) achieves improved bandwidth (BW%) for all five traffic patterns. Up to 100% of required source-destination communication (BW%) in the routing network can be accommodated using PF with an average 11% bandwidth improvement for PF versus SP. The remaining available bandwidth can be allocated and used at run-time using longer-latency packet switched routing. The table also shows that the time slot usage for TDM in the routers is more balanced if PF is used. The minimum, maximum, and average time slot usage (out of 8 available slots) across the routers is noted. By balancing slot usage for TDM at compile-time, unused slots can be more easily used for packet-switched routing at run-time.

Figure 2 shows average packet latency for transpose, tornado, and bit-reverse traffic patterns under increasing traffic injection rates. TDM-only routing using PF and SP are used to route connections at compile-time. The packet latency of routing using these approaches is compared against routing all traffic using packet switching (PS) with DOR. For PS-only traffic, latency increases sharply at the injection rates of 0.04 to 0.06, due to limitations in dimension order routing. In TDM-only routing, the PF latency is better than the SP latency for all traffic patterns. Since more TDM routing slots are available with PF, data is assigned to a free slot more easily at the source, reducing source-to-destination latency. For tornado and low-injection-rate transpose and bit-reverse, PS-only achieves lower latency in some cases since PS traffic can

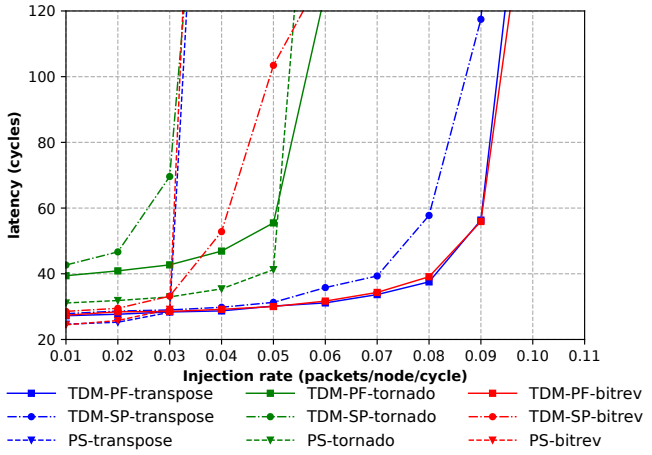

Fig. 2: Average packet latency for transpose, tornado and bit-reverse traffic patterns using only compile-time TDM or run-time PS routing using DOR.

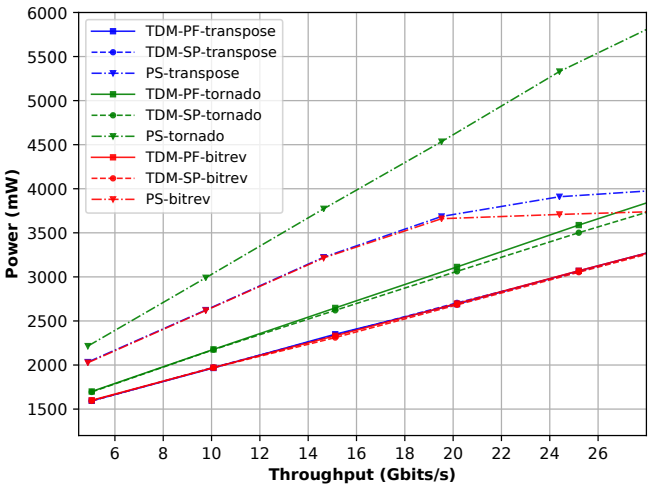

Fig. 4: Total power consumption for transpose, tornado and bit-reverse traffic using PS-only or TDM-only routing.

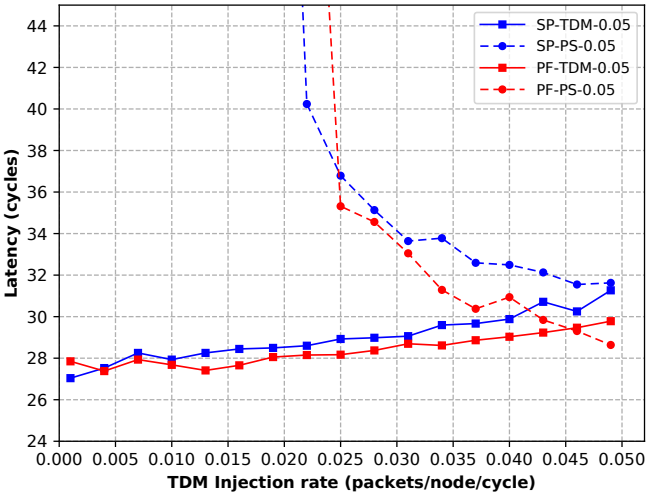

Fig. 3: Average NoC latency for varying amounts of TDM versus PS traffic for transpose under a moderate, fixed total packet injection rate of 0.05 packets/node/cycle. Note that TDM latency increases as TDM injection rate increases due to more contention for time slots.

be immediately transmitted by a source rather than having to wait for a pre-scheduled TDM time slot.

Figure 3 shows results for experiments when the total packet injection rate at each source is constant at 0.05 packets/node/cycle. The fraction of transmitted data that is scheduled via TDM versus the amount of PS traffic is varied from all PS (left) to all TDM (right). Initially, as the amount of PS traffic is high, all packets have high latency. As the TDM injection rate is increased, the PS traffic and associated latency are reduced. Latencies for TDM traffic routed with the PF algorithm are consistently less than corresponding SP results.

Figure 4 shows the total power consumption of the NoC for SP-only, PF-only, and PS-only routing for three traffic patterns. For SP- and PF-only routing, the routes have similar power numbers at low injection rate. At higher injection rates, the power consumption of PF routing is slightly greater than that of SP due to a larger number of time slots available for routing.

VI. Conclusion

In this paper, a heuristic routing algorithm that is tuned for routing traffic in hybrid FPGA NoCs is presented. This multi-iteration router balances routed traffic across routing resources to reduce congestion. Multiple rip-up and reroute iterations are used to enhance low-latency, time-scheduled communication. Substantial data stream bandwidth and latency improvement for a collection of five benchmarks is shown using our new routing approach.[1]

References

[1] N. Kapre, "Marathon: Statically-Scheduled Conflict-Free Routing on FPGA Overlay NoCs," in *FCCM*, May 2016, pp. 156–163.
[2] T. Liu, N. K. Dumpala, and R. Tessier, "Hybrid hard NoCs for efficient FPGA communication," in *FPT*, Dec. 2016, pp. 157–164.
[3] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *FPGA*, Feb. 1995, pp. 111–117.
[4] M. S. Abdelfattah, A. Bitar, and V. Betz, "Take the highway: Design for embedded NoCs on FPGAs," in *FPGA*, Feb. 2015, pp. 98–107.
[5] Z. Lu and A. Jantsch, "TDM virtual-circuit configuration for network-on-chip," *IEEE TVLSI*, vol. 16, no. 8, pp. 1021–1034, Aug. 2008.
[6] T. Carle et al., "Static mapping of real-time applications onto massively parallel processor arrays," in *Int'l Conf. on Application of Concurrency to Sys. Design*, Jun. 2014, pp. 112–121.
[7] N. Kapre et al., "Packet switched vs. time multiplexed FPGA overlay networks," in *FCCM*, May 2006, pp. 1–10.
[8] A. Shpiner et al., "On the capacity of bufferless networks-on-chip," *IEEE TPDS*, vol. 26, no. 2, pp. 492–506, Mar. 2014.
[9] S. Evian and J.-P. Diguet, "Efficient space-time NoC path allocation based on mutual exclusion and pre-reservation," in *ACM GLSVLSI*, Mar. 2007, pp. 457–460.
[10] N. Jiang et al., "A detailed and flexible cycle-accurate network-on-chip simulator," in *ISPASS*, Apr. 2013, pp. 86–96.

[1]This research was funded by a grant from Xilinx Corporation.