A Transport-Layer Network for Distributed FPGA Platforms

Sang-Woo Jun, Ming Liu, Shuotao Xu, Arvind
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
{wjun,ml,shuotao,arvind}@csail.mit.edu

***Abstract*—We present a transport-layer network that aids developers in building safe, high-performance distributed FPGA applications. Two essential features of such a network are virtual channels and end-to-end flow control. Our network implements these features, taking advantage of the low error characteristic of a rack level FPGA network to implement a low overhead credit based end-to-end flow control. Our design has many parameters in the source code which can be set at the time of FPGA synthesis, to provide flexibility in setting buffer size and flow control credits to make best use of scarce on-chip memory resources and match the traffic pattern of a virtual channel. Our prototype cluster, which is composed of 20 Xilinx VC707 boards, each with 4 20Gb/s serial links, achieves effective bandwidth of 85% of the maximum physical bandwidth, and a latency of 0.5us per hop. User feedback suggest that these features make distributed application development significantly easier.**

I. Introduction

In order to tackle large data intensive applications, many modern FPGA-based deployments are exploring the use of FPGA clusters, where a network of FPGAs are deployed and a large body of work is distributed across the FPGAs. A network protocol for an FPGA cluster largely has three important criteria: (1) it must be easily usable by an application developer, (2) It must have high performance with low latency, and (3) it must consume only a small amount of scarce on-chip FPGA memory.

Two essential features for a usable network implementation are virtual channels and end-to-end flow control, corresponding to the transport layer of the OSI network model. Without these features, the developer would have to manually manage channel multiplexing and deadlock management, making the development of high performance distributed FPGA applications difficult.

Due to the high engineering and performance overhead of existing network solutions, many inter-FPGA networks on a distributed FPGA deployment are implemented using low-overhead link-layer protocols such as Aurora using multi-gigabit serial transceivers included in the FPGA. Many existing network implementations using this network fabric often provide link and network level interfaces, but they rarely provide higher-level functionality such as end-to-end flow control.

For deadlock-free operations, all virtual channels need separate packet buffers which are large enough to mask network latency as well as bursts from multiple sources. Scarcity of on-chip memory resources prevent safe over-allocation of packet buffers, and using off-chip DRAM also consumes a lot of precious DRAM bandwidth. A solution may be clever allocation of buffer space by allowing different amount of buffers for different channels. Application developers can adjust the buffer space per channel to meet the performance criteria without increasing the total buffer requirement.

This paper presents the design and implementation results of a transport level network for a cluster of FPGAs. Our transport layer is parameterized such that flow control features for each virtual channel can be configured at FPGA synthesis time. Parameters include buffer size and flow control credits. We demonstrate that a parameterized transport-layer implementation can achieve high performance in a distributed FPGA environment while maintaining a small BRAM footprint, by adjusting a few parameters to best fit the usage characteristics of a virtual channel. In our router, we make use of the high reliability of the serial link and deterministic routing to ensure lossless in-order arriving of packets, greatly simplifying the transport layer protocol.

We have implemented a prototype of our network on a cluster of 20 Xilinx VC707 FPGA development boards, with 4 20Gb/s serial links each. Our prototype achieves an effective bandwidth of 17Gb/s per link, which is 85% of maximum physical link bandwidth, at a latency of 0.5us.

The rest of the paper is organized as follows: Section II covers the previous and related work. Section III describes our implementation of the network and transport layer, and Section IV describes the details of a prototype implementation of the network. Section V presents the performance evaluation of our implementation, and conclude in Section VI.

II. Related Work

FPGAs offer very desirable performance and power characteristics, but modern data-intensive applications often require more resources that are available on a single FPGA chip. As a result, exploration of distributed FPGA computing systems is gaining popularity. The scale of distributed FPGA deployments range from a cluster-in-a-box systems such as BlueHive [1], to rack-level deployments such as Maxwell [2], to datacenter scale deployments such as Catapult [3]. Some have also attempted heterogeneous deployments including GPUs and FPGAs [4], or to insert FPGA accelerators into the storage datapath [5], [6]. Such systems offer a much better power performance characteristics over their off-the-shelf server counterparts.

The TCP/IP network protocol stack is by far the most popular protocol for internetworking computer systems, but it may not be a good fit for inter-FPGA communication as

it is a complex and resource-heavy protocol designed for an unpredictable network such as the internet. Some FPGA cluster projects have used Ethernet's physical and data link layers for its network, but full implementation of the TCP/IP stack is rare unless it has to interact with a legacy interface [7]. Datacenter scale protocols such as Infiniband [8] implement a more efficient transport layer protocol on top of a more reliable network layer implementation. It also offloads major parts of the protocol to the NIC to achieve higher performance. Some have modified the TCP protocol [9] to significantly reduce the packet buffer size using intelligent congestion control.

BlueLink [10] demonstrated that a new protocol using high-speed serial links has a better area-performance characteristics than implementing existing network protocols. Many distributed FPGA computing systems have demonstrated high performance with FPGA nodes networked over such high-speed serial links [11], [2]. Some have developed meta language compilers that generate application-specific network logic with features such as flow control from separate network specifications [12].

III. Network Architecture

The overall architecture of the network components can be seen in Figure 1. The network architecture can be divided largely into two parts, the network layer and the transport layer. The network layer is implemented in the form of the router, and the transport layer is implemented in the endpoints that are chained to the router interface. Flow control is implemented in both layers.

The distributed application components communicate with remote nodes using the network *endpoints*. Endpoints expose send and receive interfaces, and behaves like a FIFO, in that it blocks when it cannot safely send any more packets. Many endpoints can be instantiated, resources permitting, and each endpoint can have a different *type*, meaning it can expose send and receive interfaces of different bit widths.

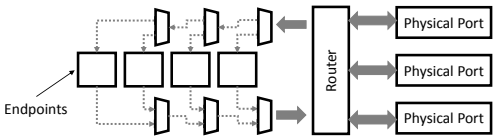


Fig. 1: Network Architecture

A. Network Layer

The network layer implements lossless, in-order packet routing, which assures that packets always arrive in the order they were sent. This removes the need for try-resend or reordering functionalities at the transport layer, allowing a much simpler design.

The router implements in-order routing by being deterministic, in that a packet from a certain endpoint of a certain source node being delivered to a certain destination node will always travel through the same path Parallelism is achieved by distributing packets from different endpoints to different paths. The router is oblivious to the existence of multiple network endpoints or virtual channels they represent. The router only deals with routing individual packets, and higher level functions such as virtual channels and end-to-end flow control is implemented by the endpoints, which are organized into a chain to reduce fan-in on the FPGA.

A packet consists of four fields: `source node ID`, `destination node ID`, `endpoint ID` and `payload data`. Destination node ID and payload data is supplied by the user. Source node ID is supplied by the router, and the endpoint ID is determined by the position of an endpoint in the endpoint chain. `Endpoint ID` is filled out by the endpoint chain when a packet is injected into it, and it is used to direct a packet to the correct endpoint at the receiving side.

B. Transport Layer

Virtual channels multiplex a single physical network link to provide the logical interface of multiple links. Figure 2 describes the flow of packets in such an environment. Our network implements a per-channel end-to-end flow control, so that a sender can only send data onto the network when it is guaranteed that the receiving endpoint has enough buffer space to accommodate it. The transport layer is implemented in individual endpoints, and its design aims to provide a very low latency and efficient memory space usage. Each design can have multiple instantiations of endpoints, parameterized differently.

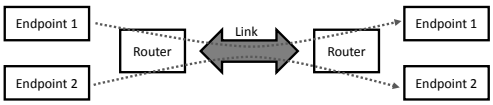


Fig. 2: Packet Flow in Virtual Channels

The structure of an endpoint is described in Figure 3. Whenever a packet is received by an endpoint, it checks a table of packets received per source node to determine if it is time to send a flow control credit to the source node. If the send budget of the source node is predicted to have become small enough and there is enough space on the local receive buffer, it enters a packet into the ack queue and marks the amount of space as allocated.

In order to maintain maximum bandwidth, flow control packets must be received before the send budget of the source node runs out. However, it is often not possible to provide a large enough buffer to conservatively accommodate the flow control packet's round trip latency for all nodes in the system.

Under such constraints, each endpoint can have a different flow control configuration that attempts to best suit its usage. For example, for some endpoints the expected traffic pattern may be that most of the data transfer happens between a pair of two nodes. In such a case, it might be effective to have a very low granularity flow control, so that a large buffer is allocated on request, but the total receive buffer may be small. On the other hand, if many nodes are expected to send data to one node, a fine granularity flow control and a large buffer may be required for performance. If the endpoint is used for low-bandwidth traffic such as commands, the buffer size and granularity can be set to a small value. To enable such control, endpoints are initialized with the parameters described in Table I.

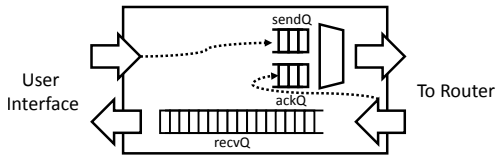


Fig. 3: Endpoint Architecture

Component	LUTS	RAMB36
Aurora Link	4843	36
Router	3743	0
Endpoint ($\times 2$)	753	3
Total	10092	42
Virtex 7 Percentage	(3%)	(4%)

TABLE II: FPGA Resource Utilization

Initially, all nodes start with a small send budget (`initBudget`) to all remote nodes, and therefore the actual size of the receive packet buffer is $initBudget \times nodeCount$ slots larger than the `BufferSize` parameter. When the first packet arrives, space in the receive buffer is allocated to the source node and a flow control packet is sent. The endpoint can choose to periodically allocate only `initBudget` size buffers, instead of `FlowCredit` in an attempt to more fairly allocate buffer space across source nodes. Yielding buffers like this achieves better buffer usage when many nodes are going to send data to one node.

The network infrastructure also provides an unmanaged endpoint which does not implement a transport layer protocol. The unmanaged endpoint can sustain the highest bandwidth and lowest latency as it does not check if the receiving endpoint has available buffers before sending packets. It should be used very carefully, since if the arriving data is not always immediately consumed and dequeued from the receiving buffer, it may cause the entire network to block.

IV. Implementation Details

We have implemented a prototype of the network described using a cluster of machines. Each node in the cluster consists of one Intel Xeon-based server, Xilinx VC707 FPGA development board, and a network expansion card which pinned out eight GTX multi-gigabit serial transceivers. Two lanes were grouped together to form a channel, resulting in a fan-out of up to 4 channels per node. A node can implement any direct network with a fan-out per node of 4 or less, such as a 2D mesh or torus.

The link latency of an aurora link based on the GTX multi-gigabit transceiver was measured to be around 0.48us, which translates to about 75 cycles on the 6.4ns user clock.

A. FPGA Resource Utilization

We have measured the FPGA resource utilization of our network using a simple setup with two endpoints: one high speed endpoint with larger flow control credit and buffer size (Credit size of 200 and buffer size of 1024 packets), and one small endpoint with smaller buffers. The endpoint row in the table below described the larger endpoint. The router component includes the chaining logic used to link the endpoints to it. The user logic was clocked at 125MHz.

Parameter	Description
BufferSize	Size of the total allocated buffer space
FlowOffset	Offset of flow control packet transmission
FlowCredit	Number of packets each flow control credit represents

TABLE I: Endpoint Parameters

V. Performance Evaluation

We demonstrate that the performance of the network does not suffer from the addition of transport-layer network functions. We measured the bandwidth and latency of the network under various configurations, and show that our network can usually achieve a bandwidth of 17Gb/s, which is 85% of the maximum physical link bandwidth. This performance is reasonable considering the packet header and flow control overhead.

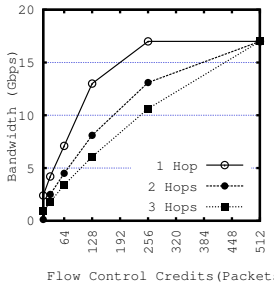
Single Endpoint Over Multiple Hops : We measured the bandwidth of the network implementation by measuring the time it takes for a single endpoint to send a large amount of data to remote nodes variable hops away, under various flow control settings. Larger credit settings mean a larger buffer size is required. Flow control offset was set to be half of the credit size. The results can be seen in Figure 4a.

When the flow control credit was small, performance of the network was lower when going over a longer network distance. This is because the round trip latency over multiple hops is longer than the time it takes to deplete the send buffer, resulting in idle cycles when no data can safely be sent over the network. With the low network latency of the serial links, maximum bandwidth over 3 network hops could be achieved using a single endpoint when the flow control credit is over 512 packets large.

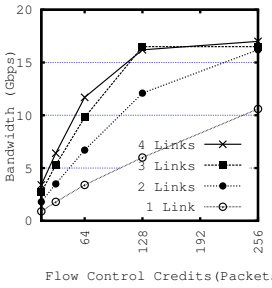
Multiple Endpoints Over Multiple Hops : Since most interesting distributed FPGA applications will have more than one network endpoint, maximum network performance can be achieved even when a single endpoint's flow control credit setting is large enough. We measured the aggregate network bandwidth of a varying number of endpoints sending data to a node three network hops away. We also measured the performance with varying flow control credit sizes. Flow control offset was set to be half of the credit size. The results can be seen in Figure 4b. It shows that a collection of smaller sized endpoints can saturate the network by filling in each others' idle cycles.

Buffer Size and Flow Control Offset : Endpoints can be characterized not only by its flow control credit size, but also by the flow control offset and buffer size parameters. The same amount of buffer space can also be allocated to a different number of nodes under different flow control credit settings. Setting a smaller offset has the risk of incurring idle time by delivering a flow control packet too late, but a large offset requires a larger buffer to accommodate earlier buffer allocation.

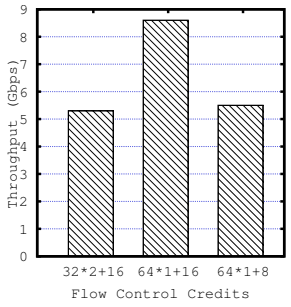
We measured the effect of such parameters by having three nodes send a stream of packets to the same remote node. We tested three scenarios, described in Table III. Two had the same



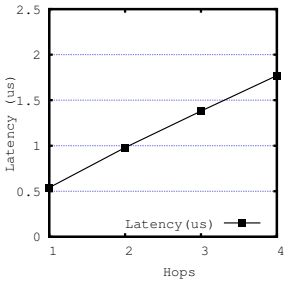
(a) Network Bandwidth With Variable Network Distance



(b) Network Bandwidth With Variable Number of Channels



(c) Network Bandwidth With Different Flow Control Settings



(d) Network Latency Per Hop

total buffer size organized into different organizations, and one had a smaller buffer. In the first scenario, the three source nodes will be contending to be scheduled into the two possible slots, where in the latter two scenarios they will be contending for one 64 packet slot.

Setting	Description
32*2+16	Buffer has space for two 32 packet blocks, with offset of 16
64*1+16	Buffer has space for one 64 packet block, with offset of 16
64*1+8	Buffer has space for one 64 packet block, with offset of 8

TABLE III: Flow Control Parameters

The results can be seen in Figure 4c. It shows that even with the same buffer size, having a larger credit setting is beneficial to a small buffer configuration. The difference is pronounced enough that even reducing buffer usage further by making the offset smaller results in a better performance compared to the configuration with smaller credit sizes.

Multi-Hop Latency : Network latency was measured by measuring the round-trip latency by sending a packet to nodes of varying distances, where the user logic immediately sends the packet back to the original sender. The results can be seen in Figure 4d. We show a consistent latency of less than 0.5us per hop.

VI. Conclusion

In this paper, we have presented our design of a parameterized, low overhead transport-layer network that provides useful features such as virtual channels and end-to-end flow control. Our network takes advantage of the high reliability of the high-speed serial links, which are integrated in the FPGA fabric, to implement a lossless in-order network layer, which allowed us to simplify the transport layer and use less FPGA resources. The design of the transport layer is parameterized, so that the developer can choose to use less resources while meeting the performance requirements of the individual endpoint. Our prototype implementation demonstrated a high performance in an FPGA cluster setting. We predict that our network will accelerate future research of distributed FPGA applications.

Acknowledgement

This work was partially funded by Quanta (Agmt. Dtd. 04/01/05) and Lincoln Laboratory(PO7000261350). We also thank Xilinx for their generous donation of VC707 FPGA boards and FPGA design expertise.

References

[1] S. Moore, P. Fox, S. Marsh, A. Markettos, and A. Mujumdar, "Bluehive - a field-programable custom computing machine for extreme-scale real-time neural network simulation," in *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on*, April 2012, pp. 133–140.

[2] R. Baxter, S. Booth, M. Bull, G. Cawood, J. Perry, M. Parsons, A. Simpson, A. Trew, A. McCormick, G. Smart, R. Smart, A. Cantle, R. Chamberlain, and G. Genest, "Maxwell - a 64 fpga supercomputer," in *Adaptive Hardware and Systems, 2007. AHS 2007. Second NASA/ESA Conference on*, Aug 2007, pp. 287–294.

[3] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Prashanth, G. Jan, G. Michael, H. S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Yi, and X. D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," *SIGARCH Comput. Archit. News*, vol. 42, no. 3, pp. 13–24, Jun. 2014.

[4] K. H. Tsoi and W. Luk, "Axel: A heterogeneous cluster with fpgas and gpus," in *Proceedings of the 18th Annual ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '10. New York, NY, USA: ACM, 2010, pp. 115–124.

[5] S.-W. Jun, M. Liu, K. E. Fleming, and Arvind, "Scalable multi-access flash store for big data analytics," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014, pp. 55–64.

[6] S.-W. Jun, M. Liu, S. Lee, J. Hicks, J. Ankcorn, M. King, S. Xu, and Arvind, "Bluedbm: An appliance for big data analytics," in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA '15. New York, NY, USA: ACM, 2015, pp. 1–13.

[7] M. Blott, K. Karras, L. Liu, K. Vissers, J. Bär, and Z. István, "Achieving 10gbps line-rate key-value stores with fpgas," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*. Berkeley, CA: USENIX, 2013.

[8] I. T. Association, *Infiniband*, 2014 (Accessed November 18, 2014). [Online]. Available: http://www.infinibandta.org

[9] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proceedings of the ACM SIGCOMM 2010 Conference*, ser. SIGCOMM '10. New York, NY, USA: ACM, 2010, pp. 63–74.

[10] A. Theodore Markettos, P. Fox, S. Moore, and A. Moore, "Interconnect for commodity fpga clusters: Standardized or customized?" in *Field Programmable Logic and Applications (FPL), 2014 24th International Conference on*, Sept 2014, pp. 1–8.

[11] T. Bunker and S. Swanson, "Latency-optimized networks for clustering fpgas," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, April 2013, pp. 129–136.

[12] K. E. Fleming, M. Adler, M. Pellauer, A. Parashar, Arvind, and J. Emer, "Leveraging latency-insensitivity to ease multiple fpga design," in *Proceedings of the ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '12. New York, NY, USA: ACM, 2012, pp. 175–184.