DART: A Programmable Architecture for NoC Simulation on FPGAs

Danyao Wang, Natalie Enright Jerger, J. Gregory Steffan
Department of Electrical and Computer Engineering
University of Toronto
Toronto, ON, Canada
{wangda,enright,steffan}@eecg.toronto.edu

ABSTRACT

The increased demand for on-chip communication bandwidth as a result of the multi-core trend has made *networks on-chip* (NoCs) a compelling choice for the communication backbone in next-generation systems [3]. However, NoC designs have many power, area, and performance trade-offs in topology, buffer sizes, routing algorithms and flow control mechanisms—hence the study of new NoC designs can be very time-intensive. To address this challenge we propose DART, a fast and flexible FPGA-based NoC simulation architecture. Rather than laying the NoC out in hardware on the FPGA like previous approaches [8, 6], our design virtualizes the NoC by mapping its components to a generic NoC simulation engine, composed of a fully-connected collection of fundamental components (e.g., routers and flit queues). This approach has two main advantages: (i) since FPGA implementation is decoupled it can simulate any NoC; and (ii) any NoC can be mapped to the engine without resynthesizing it, which can take time for a large FPGA design. We demonstrate that an implementation of DART can achieve over 100× speedup relative to a cycle-based software simulator, while maintaining the same level of simulation accuracy.

Categories and Subject Descriptors

C.1.2 [**Computer Systems Organization**]: Multiprocessors; Interconnection Architectures

General Terms

Design, Performance, Measurement

Keywords

Network-on-chip, simulation, FPGA

1. INTRODUCTION

Modern multi-cores and systems-on-chip increasingly use packet-switched networks-on-chip (NoCs) to meet the growing demand for on-chip communication bandwidth, as more cores are incorporated into each chip. NoC designs are sensitive to many parameters such as topology, buffer sizes, routing algorithms, and flow control mechanisms. Detailed NoC simulation is essential to accurate full-system evaluation.

Software simulation is used widely, both as stand-alone NoC simulators [15, 4] and as the interconnect component of large full-system simulators [7, 1]. These tools have the advantages of being very flexible, easy to program, fast to compile, and deterministic (making them amenable to debugging). However, simulation of large NoCs in software is slow, which adds to the already burdensome computation required to perform detailed full-system simulation. To maintain reasonable simulation times, the user is often forced to reduce simulation detail.

The increased on-chip logic and memory capacities of recent FPGAs allow an entire on-chip system to be implemented on a single device. FPGA-based NoC emulators [6, 16, 20, 8] can reduce simulation time by several orders of magnitude compared to software. These dramatic speedups are possible because the emulator is constructed by laying out the entire NoC on the FPGA, allowing the hardware to exploit all available fine and coarse grain parallelism between the emulated events in the NoC. However, this direct-mapped approach has three key drawbacks relative to software simulation: (i) any change in the simulated NoC requires manual redesign of the emulator HDL, (ii) redesign in turn requires complete compilation/synthesis of the FPGA design, which can take hours, or up to a day for a large design, and (iii) the maximum simulatable NoC size is determined by the FPGA capacity.

1.1 A Flexible NoC Simulation Acceleration Engine

We first proposed the DART architecture for FPGA-based NoC simulation to bridge the gap between pure software and hardware approaches [19]. The DART architecture is parametrized, and the parameters can be set by software at run-time to simulate different NoCs without modifying the hardware simulator on the FPGA. In this paper, we describe in more detail the design decisions of DART and investigate the performance and scalability of the DART architecture.

Fig. 1 shows the organization of a DART simulator, which consists of a fully-connected collection of fixed-function components that model the building blocks of an NoC: traffic generators, routers and queues. Configurable parameters within each node allows behaviors of individual nodes to be altered to match nodes in the simulated NoC. The global interconnect provides all-to-all communication between DART nodes, thus allowing simulation of different topologies without resynthesizing the design. Furthermore, simulated cycles are decoupled from FPGA cycles through the use of a global time counter: this counter is incremented once ev-

NOCS '11, May 1-4, 2011 Pittsburgh, PA, USA

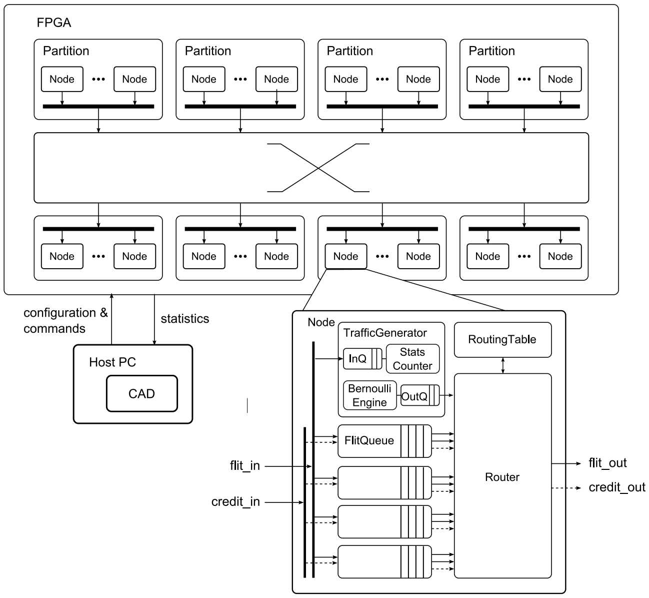


Figure 1: DART Simulator architecture on the FPGA

ery simulated-cycle after all network transfers for that cycle are simulated (which may take a variable number of FPGA cycles). Virtualizing simulation time allows us to optimize the DART components for area efficiency. This partially addresses the limitation on simulatable NoC size due to FPGA capacity constraints, allowing us to easily add support for more simulated nodes than physical DART nodes to future versions of DART.

Contributions We make two main contributions: First, we design and evaluate a novel overlay architecture that enables software run-time configurable NoC simulation on FPGAs. The novelty of DART lies not in the NoC features implemented but in its approach to simulation: *fast and flexible* FPGA simulation *without resynthesis*. Second, we demonstrate a complete implementation of the DART simulation toolset that achieves over 100× speedup relative to the cycle-based software simulator Booksim[4], while maintaining the same level of simulation accuracy.

2. RELATED WORK

2.1 Software Simulation

Cycle-accurate software NoC simulators incorporate detailed models of network components, and evaluate network performance by capturing the timing of messages as they traverse the simulated network. Software simulators exist both as stand-alone interconnect simulators such as Booksim [4] and SICOSYS [15], and as the network component in full-system simulators such as Garnet [1] of GEMS [10] and SimFlex [7]. However, detailed full-system simulation is slow. Recent work including ProtoFLEX [2] and RAMP Gold [17] speed up processor simulation using FPGAs. The NoC component can become the bottleneck in the simulation of large systems.

As multi-core processors become widely available, one way to improve software simulation speed is to leverage multithreading. The main challenge in parallelizing NoC simulators using threads is that NoC simulation is communication intensive. The fine-grained synchronizations can incur high communication overhead. DARSIM [9] is a parallel NoC simulator that achieves good scalability for up to 4 threads in cycle-accurate mode, which requires two global synchronizations per simulated cycle. Relaxing this constraint allows good scaling to 8 threads at the cost of lower accuracy.

2.2 FPGA-based Emulation

Genko et al. [6] describe an emulation platform that consists of programmable traffic generators and receptors that drive a 6-switch NoC and is 2600× faster than a SystemC simulation of the same network. While this platform supports programmable traffic patterns and statistics counters, changing the router configuration requires resynthesis of the emulator. DRNoC [8] circumvents this requirement by leveraging the partial reconfigurability of Xilinx FPGAs. The DRNoC host FPGA is divided into grids; each grid slot can be dynamically reconfigured to implement a different router model. However, partial reconfiguration requires a special design flow and incurs area overheads; it is also only available for select devices. In contrast, DART's configuration interface is based on a generic shift register and is portable to any FPGA. NoCem [16] improves emulation density over Genko et al.'s design [6] and implements a 9-node mesh network on a single FPGA by eliding the router pipeline details and virtual channels. Instead of sacrificing these important details, we employ a simple design for each DART Router: each has multiple input ports but only one output port, and models the all-to-all switching in a simulated router by routing one input port per DART cycle.

Wolkotte et al. [20] virtualize a single router on an FPGA, allowing the simulation of an NoC with multiple routers. An off-chip ARM processor stores N contexts for the router model and orchestrates the emulation of the N-node network. This approach allows the router model to be much more detailed. However, the off-chip ARM/FPGA communication link is a performance bottleneck. DART's simulation components are implemented completely on-chip and DART does not suffer from this bottleneck.

AcENoCs [11] provides a novel HW/SW design for NoC emulation. It leverages a soft-processor for traffic generation and source queue functionality, freeing FPGA resources to allow for more detailed routers, and enabling a network size comparable to DART.

DART is similar in spirit to the RAMP Gold [18] and ProtoFLEX [2] processor simulators that decouple the simulator architecture on the FPGA from that of the simulated system. Both RAMP Gold and ProtoFlex use host multi-threading to simulate large-scale multi-cores on top of a single processor pipeline. Compared to a processor, nodes in an NoC have simpler pipelines but carry out more fine-grained communication. To efficiently model these characteristics, DART uses multiple functional components that are synchronized to exploit the fine-grained parallelism in NoC simulation.

A-Ports [13] separates the timing and functional models of a processor simulator using components that communicate asynchronously. DART uses a globally synchronous approach to avoid the need of large buffers in intermediate nodes. To overcome FPGA size constraints, HAsim [14] uses a novel time-division multiplexing scheme to simulate a larger number of cores and large NoC design.

3. DART ARCHITECTURE

The basis of the DART architecture is to provide programmability by decoupling (i) the simulator architecture from the architecture of the simulated NoC, and (ii) DART cycles from simulated cycles. To provide a configurable functional model for NoC simulation, we abstract common NoC functionalities into three basic components: *Traffic Generators* (TGs), *Flit Queues* (FQs) and *Routers*. Components can be mixed and matched to model more complex NoC nodes. A given topology is then simulated by configuring the Routers to only forward to their simulated neighbors via the global interconnect; this configuration does not require resynthesis.

Traffic Model A typical NoC carries two types of traffic:*flits (flow control units)* that carry data messages and *credits* that are exchanged between neighboring routers to enforce flow control [4]. DART models flits and credits using *descriptors* that contain only the information necessary to forward them from source to destination. A 36-bit flit descriptor encodes the injection and next-transfer timestamps, source and destination addresses, and boolean flags for the flit type (head, tail, and warmup). Warmup flits are used to bring the network to steady state and hence do not have their latencies recorded. Not encoding the data payload saves area as fewer bits are stored and passed between DART nodes. We choose 36 bits to match the port width of embedded RAM blocks on the FPGA, which are used to implement the flit buffers. Anything wider doubles the RAM usage as two RAM blocks must be used in parallel to support the data width. A credit descriptor encodes only a timestamp and a virtual channel ID.

Timing Model To capture the timing of flit transfers, we use a global time counter to synchronize all network events. Each flit contains a *timestamp* that indicates when the next transfer of this flit should happen. As a flit traverses the network, its timestamp is updated by intermediate DART nodes to reflect the delay due to pipeline latency and simulated contention. Credit transfers are timed similarly. Upon arrival at the destination TG, a flit's latency is computed by subtracting the injection timestamp from the arrival timestamp. The 10-bit timestamps allow DART to correctly compute latency provided a flit's latency does not exceed 1024 simulated cycles. We believe this is a reasonable compromise to keep the flit descriptors within 36 bits as most on-chip communication takes no more than a few hundred cycles. However, the maximum simulation length DART supports is not limited to 1024 cycles. By using signed subtractions to compare timestamps, we can correctly determine the chronological order of timestamps within 512 simulated cycles even when the global time counter wraps around. Since the DART design guarantees that timestamps of all flits traversing the global interconnect fall within a N-cycle window, where N is the simulated latency of the router pipeline and is smaller than 512, flits will always be delivered in correct simulation order.

Design Space Coverage The bit widths of the other descriptor fields are also chosen to be minimum size while still providing sufficient functionality coverage. The 8-bit node addresses, 3-bit port ID and 2-bit virtual channel (VC) ID allow DART to scale to 256 nodes, 8 ports per node and 4 virtual channels per port. Configurations that fit within these flit widths can be setup in software at run-time. These widths do not fundamentally limit the size a NoC that DART can simulate; larger sizes can be accommodated through re-synthesis with only minor HDL changes.

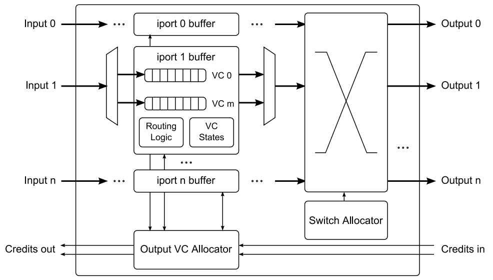


Figure 2: Data path of canonical wormhole VC router

3.1 Flit Queue (FQ)

The Flit Queue component models the VC buffers at a router's input ports and the bandwidth/latency constraints of the link feeding the port. The buffers are independent FIFO (first-in-first-out) queues that are implemented by statically partitioning a single block-RAM among the VCs. A Verilog parameter controls the number of VCs to incorporate. Each incoming flit is queued according to its VC after its timestamp is updated to reflect the delay it experiences traversing the link due to latency and bandwidth constraints – both parameters are configurable per FQ. A flit is forwarded to the next-hop Router when it gets to the front of the FIFO and the global simulation time is equal to its timestamp. This ensures all flits arrive at a Router in chronological order, which is required for correct simulation of resource contention. A separate FIFO is used for the credit channel. Similar to the flits, a credit can leave an FQ only during its scheduled dequeue time.

3.2 Traffic Generator (TG)

When enabled, the Traffic Generator component injects traffic in one of two modes: synthetic or dynamic. The former is useful for stress testing the simulated network. The latter provides an interface to incorporate DART into a full-system simulator. The mode is configurable per TG. In synthetic mode, a TG injects flits in bursts of fixed-sized packets using a Bernoulli process. Packet size (minimum 2 flits), destination node address, and injection interval are configurable per TG. In dynamic mode, a TG receives packet descriptors from the host PC and injects packets according to the descriptors. Packet size can be varied from 2 to 256 flits in powers of 2. Packet descriptors can be generated from either a memory access trace or a processor simulator running concurrently with DART.

In addition to the injection state machines, each TG also contains two FQs: the *input buffer* models the last-hop delay to the TG, and the *output buffer* models the source queue. We use the same technique from Dally and Towles [4] and allow the injection state machine to lag behind the current simulation time when the output buffer is full, to model an infinite source queue. TGs also serve as traffic sinks and record the number of packets received and the cumulative packet latency. More statistics counters can be easily added.

3.3 Router

State-of-the-art NoCs use the classic wormhole VC router (Fig. 2), which is composed of per-VC flit buffers, routing logic, VC and switch allocators and a crossbar. Since the FQs model the flit buffers, the Router component only encapsulates the routing and allocation logic. Fig. 3 shows the Router datapath. A Verilog parameter controls the number

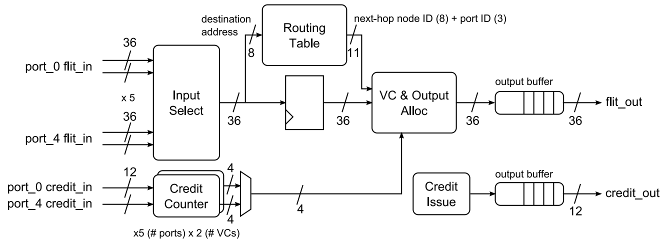


Figure 3: DART Router datapath

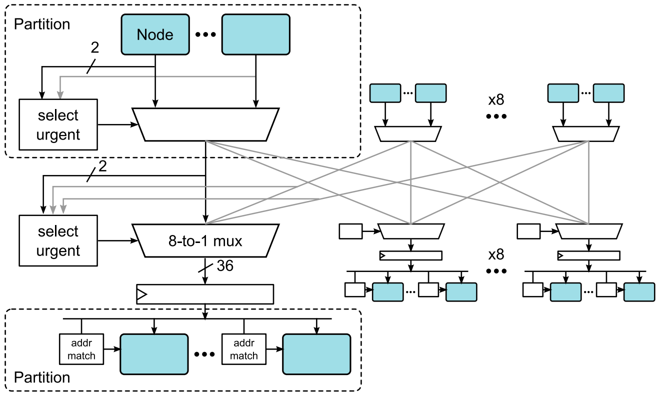


Figure 4: DART's global interconnect. Nodes are grouped into partitions so the crossbar needed is small

of ports. Table-based routing is used to allow different routing algorithms and facilitate the simulation of a wide range of topologies. The table contents are configurable after synthesis by a host PC.

A 4-bit counter for each output VC is used to implement credit-based flow control. Initial credit values represent the number of entries in the input buffer at the downstream router. The counter is decremented when a flit is routed, and incremented when a credit is received. The values are configurable for each VC and Router.

Area-Speed Trade-off The allocators and the crossbar in the classic router are complex structures [12]. The virtualized simulation time in DART allows us to implement the same functionality in the DART Router using simple arbiters and a multiplexer by trading off simulation speed. The Router component routes one flit per DART cycle. By routing the input VCs one at a time while holding the global time counter, the DART router can model any multi-ported classic router. A round-robin scheme selects an input VC to route in each DART cycle. Delay due to failed VC allocation or credits is modeled by incrementing the timestamp of the flit while it remains in the FQ. When a flit is finally routed, its timestamp is incremented by a fixed pipeline latency. This pipeline latency is configurable per Router.

3.4 Global Interconnect

The global interconnect provides uniform-latency communication between all DART nodes. By configuring the routing tables appropriately, DART can simulate any topology. The maximum node radix is limited by the number of ports configured in the Router components. Fig. 4 shows the interconnect organization, where nodes are grouped into partitions and the partitions are connected by a small crossbar. We choose this organization over a full crossbar to conserve area. A separate, narrower, but otherwise identical interconnect carries the credit traffic. Both intra- and inter-partition arbitration use round-robin arbiters, with priority given to flits with timestamps equal to the current simulation time. These flits must be forwarded first before ticking the global time counter to prevent late flits, which may cause out-of-order flits at the next Router. Flits with timestamp ahead of the current simulation time can be forwarded out-of-order across the global interconnect because flits destined for each FQ remain in order. The priority is implemented by having two separate sets of arbiters. Because it takes a cycle to detect that all flits with the current timestamp have crossed the interconnect, each simulated cycle takes at least 2 DART cycles.

The partitions are the throughput bottleneck because only one flit can be sent and received by a partition per DART cycle. For a fixed number of DART nodes, varying the size of the partition trades off the global interconnect throughput for implementation area. For our current 9-node implementation, we use 8 partitions connected by an 8 × 8 crossbar. In general, the largest crossbar that fits in the device once the nodes are implemented should be chosen.

3.5 Configuration and Data Collection

DART nodes are configured by connecting the configurable fields in a 16-bit shift register chain. A small finite state machine for each routing table is also connected to the chain. It captures the data that is shifted in and populates the routing table. The configuration byte-stream is generated on a host PC and sent to the FPGA via the serial port. We use a custom 16-bit command protocol, which can be implemented on any physical interface. Similarly, performance counters are read back by shifting them through a 16-bit-wide chain. Currently three counters are incorporated per TG to record the number of injected and received packets (32 bits) and the cumulative packet latency (64 bits). More counters can be easily added to this shift register chain.

3.6 Software Tools

The DART software tools run on a host PC connected to the FPGA where the hardware simulator resides. They allow the dynamic reprogramming of the hardware simulator after it is implemented on the FPGA. The DARTgen tool creates the configuration byte-stream from two input files that specify the on-chip DART architecture and the user network to be simulated respectively. Nodes and links in the user network are mapped to Routers and Flit Queues. We use a round-robin scheme to balance the number of used DART nodes across different partitions. This provides sufficient load balancing because the on-chip communication bottleneck is within each partition, where the nodes contend for the shared access to the inter-partition crossbar. Each DART node is annotated with the properties of the corresponding user network node. The configuration byte-stream is generated by writing the contents of the configuration registers to a file. It is used by the DARTportal tool, which provides a command-based interactive interface to configure, run and collect data from the simulator.

4. IMPLEMENTATION

We design the DART components in Verilog HDL. Device specific constructs are avoided whenever possible so the simulator core can be implemented on different FPGA systems with minimal changes. To demonstrate the functionality of

Table 1: Resource utilization breakdown of a 9-node DART on a XC2VP30 FPGA

Module	Per-Module Resource Util.			% of Total
	4-LUTs	FFs	BRAMs	4-LUTs
Traffic Gen.	691	500	2	24.7%
Flit Queue	305	145	1	43.5%
Router	612	201	0.5	21.8%
Global Inter.	2144	104	0	8.5%
Control Unit	152	70	0	0.6%
UART interface	208	171	1	0.8%
Total	26,38	13,192	99	100%
% of Available	96%	48%	72%	

Table 2: 3×3 mesh configuration parameters

Topology	3 × 3 mesh
Link latency	1 flit cycle
Router architecture	Input queue
Routing algorithm	Dimension-order (XY)
# of VCs per port	2
VC Allocation	Round-robin
Input VC buffer size	5
Router pipeline latency	5 flit cycle
Traffic pattern	Permutation traffic
Packet size	2 flits

DART and to obtain real measurement of simulation speed, we implement a 9-node DART on a Xilinx University Program Virtex-II Pro Development System (XUPV2P) [21]. We use the Xilinx ISE 10.1 software suite for synthesis and implementation. Note that the global interconnect size and flit descriptor size can be trivially extended to implement a larger DART system.

Table 1 shows the resource breakdown of DART components as implemented on the XUPV2P platform. Because every two Routers share a dual-ported routing table to make efficient use of the dual-ported block-RAM, each Router uses 0.5 block-RAMs on average. The maximum number of DART nodes that fit on this FPGA is nine. Each node consists of one TG, one 5-ported Router, and four FQs with 2 VCs each. We use 8 partitions in the global interconnect.

5. ANALYSIS

In this section we validate DART's simulation results using Booksim as a reference. Because Booksim is widely used among NoC researchers, we hope this choice of baseline provides more confidence in DART's correctness and performance potential. We measure DART's speedup over Booksim using our Virtex-II Pro implementation. We also investigate the performance cost of a programmable simulator architecture and DART's scalability.

5.1 Correctness

We developed a cycle-accurate DART architecture simulator in C++ prior to building the FPGA architecture to explore different design options and to verify the correctness against Booksim. The architecture simulator also serves as the design specification to verify the hardware design. Results shown in this section are obtained from this architecture simulator.

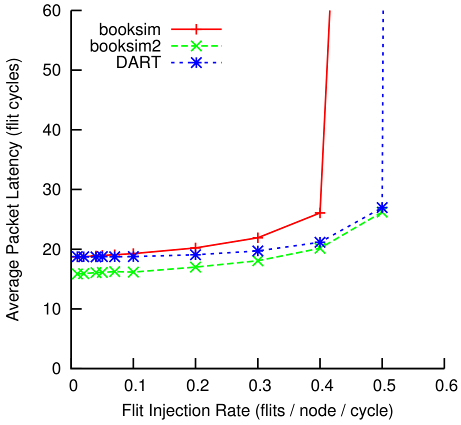
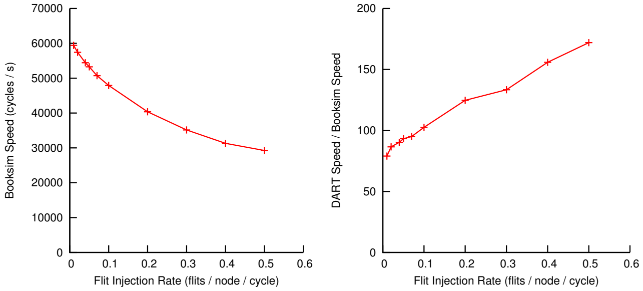


Figure 5: Average packet latency for Booksim and DART for a 3×3 mesh



(a) (b)

Figure 7: DART performance: (a) Booksim simulation speed, (b) Speedup achieved by DART vs. Booksim

We simulate a 9-node network and compare the average packet latency of Booksim and DART (Fig. 5). Table 2 shows the parameters of the simulated network. To investigate the accuracy loss DART incurs by not modeling the delay through each stage separately, we simulate two router configurations in Booksim: *booksim* has a 5-cycle routing delay and zero switch and VC allocation delay, while *booksim2* has a 4-cycle routing delay, 1-cycle switch allocation delay and zero VC allocation delay. We simulate 15,000 warm-up cycles, 30,000 measurement cycles and a draining phase. The flit injection rate is varied from 0.01 until saturation. DART tracks Booksim closely at low injection rates. At higher injection rates, the one-stage pipeline in the Router results in a less accurate latency measurement. This is evident in that DART latency is enveloped by the two Booksim configurations that have the same overall router latency but different latencies at each stage. To further investigate the mismatch, Fig. 6 shows the distribution of packet latencies at 0.4 flits per cycle. The peaks at 8, 14, 20, 26, and 32 correspond to the zero-load latencies for 0, 1, 2, 3, and 4-hop paths. The lower peaks reflect the queuing delay and resource contention packets experience at the routers. Booksim has a much longer tail than DART. Because all contentions (buffer, VC, and switch) are modeled in one stage in the DART Router, DART may under predict the latency for a flit to acquire all resources. However, the similar overall shapes of the two distributions increases our confidence that DART produces useful predictions of network performance trends.

5.2 Speedup vs. Software Simulation

In Fig. 7 we evaluate the 3×3 mesh benchmark described in Table 2 on the XUPV2P DART implementation and com-

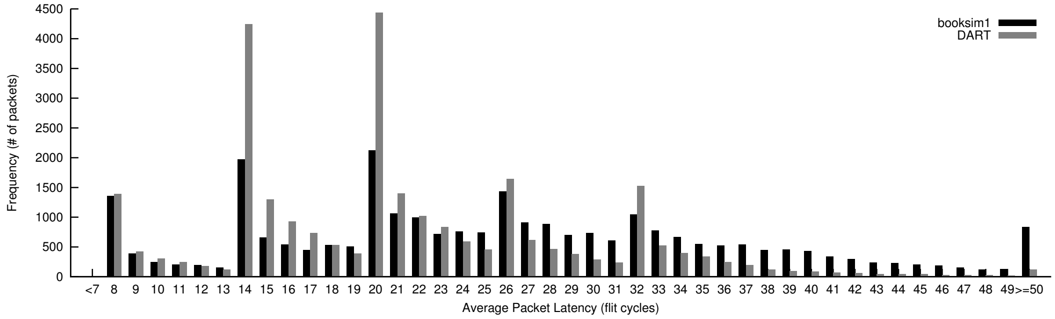
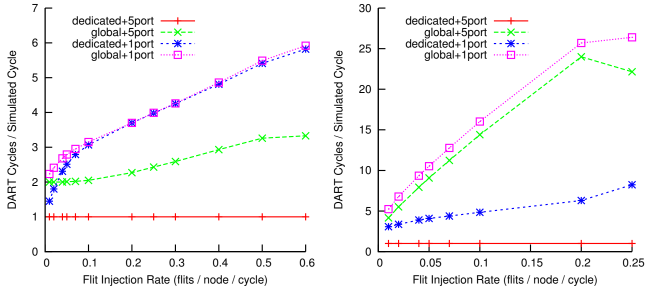


Figure 6: 3 × 3 Mesh. Packet latency distribution measured by Booksim and DART (flit injection rate = 0.4)



(a) (b)

Figure 8: Overhead of the DART interconnect and simplified Router model for a (a) 9-node and (b) 64-node DART. 3×3 and 8×8 mesh with random permutation traffic simulated

pare the simulation speed to Booksim. For the Booksim baseline, we measure the execution time of the main loop, excluding network setup, on a 2.66 GHz Core 2 Quad Linux workstation. Each data point is an average over 20 runs. We measure DART's execution time in DARTportal from the sending of the "Run" command and until the end-of-simulation signal is received back from the simulator. Configuration time is excluded. The speedup is the ratio of the number of cycles simulated per second in DART to that in Booksim. We observe that Booksim's simulation speed decreases with increasing injection rate. DART's speed is roughly constant, with all measured runtimes falling within 3% (0.528 ms) of the average (20.5 ms for 50,000 cycles)—this is because DART's execution time increases slowly with traffic, and is largely masked by the high IO overhead to send and read-back commands to/from the simulator, which accounts for over 50% of the measured time (about 11.7 ms). As a result, DART achieves greater speedup at higher packet injection rates. The IO overhead can also be amortized in longer-running simulations of networks larger than the 3×3 mesh used here.

5.3 Cost of Programmability

The main alternative to DART's programmable architecture is one that is directly laid-out in the FPGA fabric. In this section we measure the performance cost of DART's programmability by measuring the overhead (extra cycles required) of DART's global interconnect and the simplified Router model, relative to a model with a dedicated interconnect and full routers. As shown in Fig. 8, we measure for a 9-node and a 64-node DART all combinations of the two types of interconnect and two types of router:

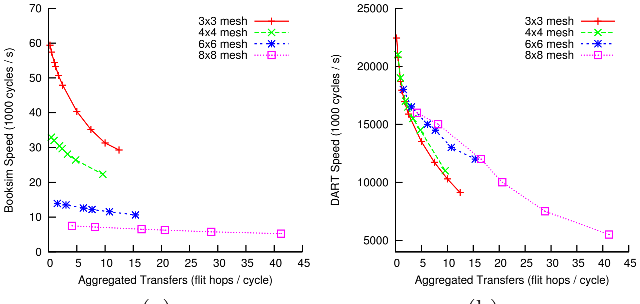
- **dedicated**: Baseline interconnect with dedicated links between connected ports on neighboring nodes
- **global**: DART interconnect with 8 partitions
- **5port**: True 5-ported router
- **1port**: DART Router that routes 1 flit per DART cycle

Fig. 8a shows the number of DART cycles required per simulated cycle for the 3×3 mesh benchmark from Table 2. The baseline (dedicated+5port) has a constant cycles per second (CPS) of 1 as it corresponds to a direct mapping of the 9-node mesh NoC. Global+5port shows the performance loss due to the DART interconnect. The timer increment bubble, described in Section 3.4, limits the minimum CPS to 2. Increased traffic causes more contention over the interconnect and lower CPS. Dedicated+1port shows the performance loss due to the serial processing of input VCs in the Router. CPS increase with network traffic, as each Router has more input VCs in use. Global+1port shows that for 9 nodes, because of the small number of nodes and low throughput of the Router, the global interconnect is not the performance bottleneck. However, Fig. 8b shows that with more nodes contention increase for the global interconnect and it can become the bottleneck. An appropriate interconnect size should be chosen for each DART implementation. DART's interconnect uses more area than dedicated links, but the overhead is compensated for by the simplified Router. Thus, the overall area cost is comparable to published results from existing direct mapped emulators [6, 16]. We believe the performance penalty is a worthwhile tradeoff for the ability to reconfigure the simulator at run-time without any hardware modification.

5.4 Scalability

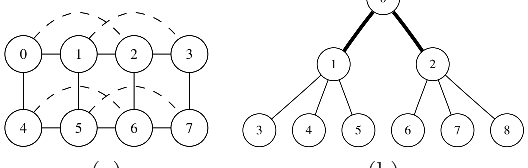
We explore the scalability of DART beyond 9 nodes on a larger FPGA. In comparison to the 2VP30 device, the Virtex 5 LX330T provides 10× more LUTs and FFs but only 4× more 18Kb block-RAMs. Each 18Kb block-RAM can accommodate 4 FQs with 2 VCs allowing DART to scale to 64 nodes, limited by the amount of logic resources on the FPGA.

We explore the performance of larger DART implementations using the architecture simulator. The predicted runtime does not include communication overhead to and from the host PC. Fig. 9 highlights the different scaling trends of Booksim and DART for four different size mesh networks. The aggregated flit transfers per simulated cycle is the product of the flit rate, average number of hops between source



(a) (b)

Figure 9: Booksim (a) and DART (b) simulation speed for various networks



(a) (b)

Figure 10: Two microbenchmarks: (a) 4×2 mesh with express links, and (b) 2-level tree

and destination pairs and number of nodes. It measures the overall amount of in-flight traffic that traverses the network every cycle. Booksim's simulation speed depends on both the size of the simulated network and the amount of network activity because it must simulate every cycle including those when the simulated network is idle. This overhead dominates simulation time for large networks and the network activity becomes an insignificant factor for performance. DART's simulated time advances faster when the simulated network is idle. Its simulation speed thus depends only on the amount of network activity. As a result, DART's speedup over Booksim varies from 300× for the 3×3 mesh to 2000× for the 8×8 mesh. These estimates are higher than the measured speedup from Section 5.2 due to the overhead of sending commands to the FPGA. In long-running simulations, this overhead can be amortized. The design focus for DART is on improving area efficiency so more simulator nodes can be implemented on a given FPGA.

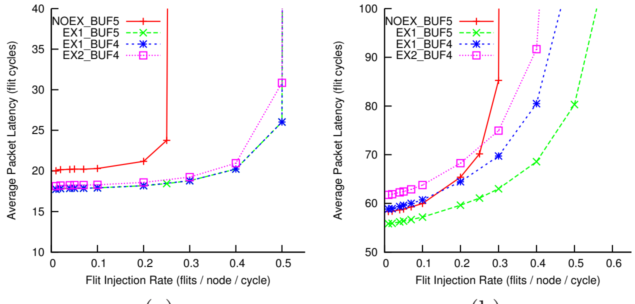
6. CASE STUDIES

We choose two examples (Fig. 10) to demonstrate DART's ability to simulate different network configurations without resynthesis. The results presented in this section are simulated using a randomly generated permutation traffic pattern. All configurations are implemented on the same 9-node DART described in Section 4.

6.1 Mesh with Express Links

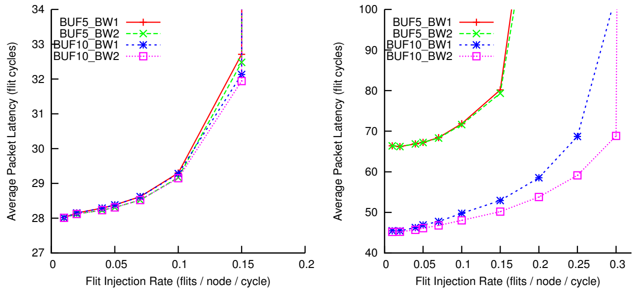
Fig. 10a illustrates a simplified version of express cube [5]. The solid lines represent local links and the dashed lines represent express links that allow non-local traffic to bypass intermediate nodes. Fig. 11 shows the average packet latency for the following configurations:

- **NOEX_BUF5**: No express link, 5 flits/VC input buffer
- **EX1_BUF5**: With express links, 5 flits/VC input buffer



(a) (b)

Figure 11: Express links performance: (a) 2-flit packets, and (b) 16-flit packets



(a) (b)

Figure 12: Tree performance: (a) 2-flit packets, and (b) 16-flit packets

- **EX1_BUF4**: With express links, 4 flits/VC input buffer
- **EX2_BUF4**: EX1_BUF4 with 2-cycle express links

For both small (2-flit) and large (16-flit) packets, the express links reduce packet latency because flits traverse fewer hops. The increased bisection bandwidth afforded by the added links also allows the network to accept higher load before saturating. To compensate for the additional area the added express link port incurs in each router, we reduce the input buffer size from 5 flits to 4 flits (EX1_BUF4 vs. EX1_BUF5). The resulting performance degradation is more pronounced for large packets because they are more bursty, hence more sensitive to buffer space. Because the express links span two hops, we increase their latency to 2 cycles while keeping the latency of other links at 1 cycle (EX2_BUF4). This configuration causes higher latency for large packets because credits take longer to replenish on the slower express links. However, the network still saturates later than the NOEX_BUF5 baseline. We also experimented with 4-flit and 8-flit packets, and the results are in line with the trend shown here.

6.2 Tree

Fig. 10b illustrates a tree where two sub-trees are linked by a root router, where the bold lines represent global links. This organization captures the essence of building blocks in a hierarchical on-chip network. Only the leaf nodes generate and receive traffic, and 50% of the generated traffic cross the root router. Fig. 12 shows the average packet latency for the following configurations:

- **BUF5_BW1**: Unit latency and bandwidth for all links, 5 flits/VC input buffer
- **BUF5_BW2**: BUF5_BW1, bandwidth = 2 flits/cycle on global links
- **BUF10_BW1**: BUF5_BW1 with 10 flits/VC input buffer
- **BUF10_BW2**: BUF5_BW2 with 10 flits/VC input buffer

For both small and large packets, increasing the global link bandwidth (BUF5_BW2 vs. BUF5_BW1, BUF10_BW2 vs. BUF10_BW1) does not significantly improve packet latency because all flits crossing the global links must be first stored in the buffers of the gateway routers (nodes 1 and 2), which form the performance bottleneck. Increasing the buffer space to 10 flits significantly reduces latency. Again the reduction is greater for largest packets because bursty traffic is more sensitive to buffer sizes. Moreover, Fig. 12b shows that once the buffer space bottleneck is removed, increasing global link bandwidth can provide additional performance improvement.

7. EXTENDING DART

DART can and will be extended in the following three ways. First, the maximum size of NoC that can be simulated by the current realization of DART is limited by the amount of on-chip resources. However, it is relatively straightforward to extend DART by virtualizing each DART node to support the contexts of multiple simulated nodes, freeing DART to simulate a larger number of nodes than there are physical nodes on the FPGA. A similar approach has been used to study large scale multi-processor systems using FPGAs [2, 17, 14]. Second, the router model can be extended to support adaptive routing algorithms and more sophisticated router architectures such as speculative routing by using soft processors as Router nodes. While an off-the-shelf soft processor may not meet the area and timing constraint of the DART Router node, there is potential for a reduced-feature soft processor that is optimized for routing operations. Finally, DART can be integrated into an existing full-system simulator to enable more comprehensive studies of the NoC and other system components.

8. CONCLUSIONS

We introduced a software-programmable overlay architecture for NoC simulation on FPGAs. By decoupling the simulator architecture from the architecture of the simulated NoC and virtualizing simulation time, DART improves upon existing FPGA-based emulators by eliminating the high cost of modifying and resynthesizing the hardware emulator when simulating different NoCs. At the same time, DART is significantly faster than software NoC simulators. Using an implementation of a 9-node DART simulator on a Virtex II Pro FPGA, we demonstrate over 100-fold speedup over Booksim while maintaining similar level of accuracy. Through two examples, we also show that irregular NoCs can be easily set up and simulated on DART.

Acknowledgements

This research was supported by the Natural Science and Engineering Research Council (NSERC). We thank Jason Anderson and Andreas Moshovos for their feedback on this work. Additionally, we would like to thank the anonymous reviewers for their constructive suggestions.

9. REFERENCES

[1] N. Agarwal *et al.*, "GARNET: A detailed on-chip network model inside a full-system simulator," in *ISPASS*, April 2009.

[2] E. Chung *et al.*, "PROToFLEX: FPGA-accelerated Hybrid Functional Simulator," in *IPDPS*, March 2007.

[3] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *Proc. Design Automation Conference*, 2001.

[4] ——, *Principles and Practices of Interconnection Networks.* Morgan Kaufmann Publishers Inc., 2003.

[5] W. Dally, "Express cubes: Improving the performance of k-ary n-cube interconnection networks," *IEEE TOCs*, pp. 1016–1023, 1991.

[6] N. Genko *et al.*, "A complete network-on-chip emulation framework," in *DATE*, March 2005.

[7] N. Hardavellas *et al.*, "SimFlex: a fast, accurate, flexible full-system simulation framework for performance evaluation of server architecture," *SIGMETRICS Perform. Eval. Rev.*, vol. 31, no. 4, pp. 31–34, 2004.

[8] Y. Krasteva *et al.*, "A Fast Emulation-Based NoC Prototyping Framework," in *RECONFIG*, Dec. 2008.

[9] M. Lis *et al.*, "Scalable, accurate NoC simulation for the 1000-core era," in *ISPASS*, April 2011.

[10] M. M. K. Martin *et al.*, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[11] V. Pai, S. Lotlikar, and P. Gratz, "AcENoCs: A configurable HW/SW platform for FPGA accelerated NoC emulation," in *IEEE International Conference on VLSI Design*, Jan 2011.

[12] L. Peh and W. Dally, "A delay model and speculative architecture for pipelined routers," in *HPCA*, 2001.

[13] M. Pellauer *et al.*, "A-Ports: An Efficient Abstraction for Cycle-Accurate Performance Models on FPGAs," in *FPGA*, Feb. 2008.

[14] M. Pellauer et al., "HAsim: FPGA-based high-detail multicore simulation using time-division multiplexing," in *HPCA*, February 2011.

[15] V. Puente et al., "SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems," in *Euromicro Workshop on Paral. and Distr. Processing*, 2002.

[16] G. Schelle and D. Grunwald, "Onchip Interconnect Exploration for Multicore Processors Utilizing FPGAs," in *WARFP*, 2006.

[17] Z. Tan *et al.*, "RAMP Gold: An FPGA-based architecture simulator for multiprocessors," in *Proc. Design Automation Conference*, 2010, pp. 463–468.

[18] Z. Tan et al., "A Case for FAME: FPGA Architecture Model Execution," in *ISCA*, May 2010.

[19] D. Wang, N. Enright Jerger, and J. Steffan, "DART: Fast and Flexible NoC Simulation using FPGAs," in *WARP*, 2010.

[20] P. Wolkotte, P. Holzenspies, and G. Smit, "Fast, Accurate and Detailed NoC Simulations," in *First Int'l Symp. on Networks-on-Chip*, May 2007.

[21] Xilinx, Inc., "Xilinx University Program Virtex-II Pro Development System Hardware Reference Manual," 2008, http://www.xilinx.com/univ/XUPV2P/Documentation/ug069.pdf.