DART: Fast and Flexible NoC Simulation using FPGAs

Danyao Wang, Natalie Enright Jerger, J. Gregory Steffan

DART: Fast and Flexible NoC Simulation using FPGAs

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

1. Introduction

A packet-switched network-on-chip (NoC) is becoming a more compelling choice for the communication backbone in next-generation multicores and systems-on-chip [2]. NoC designs are sensitive to many parameters such as topology, buffer sizes, routing algorithms, and flow control mechanisms. Hence, system architects and researchers must include detailed NoC simulation as part of any complete system simulation. However, detailed NoC simulation adds to the already burdensome computation required to accurately perform full-system evaluation.

NoCs are normally simulated in software, as standalone NoC simulators [3], [7] and also as the interconnect component of large full-system simulators [1], [5]. Software NoC simulators have the advantages of being very flexible, easy to program, fast to compile, and deterministic (making them suitable for debugging). However, software simulators are very slow for large NoCs, and can require the user to reduce simulation detail to maintain reasonable simulation times.

Several FPGA-based NoC emulators have been proposed [4], [6], [8], [9] that reduce simulation time by several orders of magnitude. These dramatic speedups are possible because the emulator is typically implemented by laying out the entire NoC on the FPGA, allowing the FPGA to exploit all available fine and coarse grain parallelism between events in the NoC. However, mapping the simulated NoC directly onto the FPGA has three key drawbacks: (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 available FPGA capacity.

A Flexible NoC Simulation Engine: To address these challenges we propose *DART*, an FPGA-based NoC simulation engine where the simulated NoC and the architecture of DART are decoupled and independent. In this paper, we present an implementation of DART on a Virtex 2 Pro FPGA, demonstrating: (i) an overlay engine that provides software-like accessibility to the FPGA, allowing different NoC architectures to be simulated without recompiling/resynthesizing the DART engine; and (ii) over $300\times$ speedup over the cycle-based software simulator Booksim [3], while maintaining the same level of simulation accuracy.

2. Related Work

Genko et al. [4] describe an emulation platform that consists of programmable traffic generators and receptors that drive a 6-switch NoC and is $2600\times$ faster than a SystemC simulation of the same network. While this platform supports programmable traffic patterns and statistics counters, changing the configuration of the network requires re-generating the emulator. DRNoC [6] 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 new component to model different networks. However, partial reconfiguration requires a special design flow and incurs area overheads; it's also only available for select devices. In contrast, DART's configuration interface is based on a generic shift register and can be implemented on any FPGA.

NoCem [8] improves emulation density over Genko et al.'s design [4] 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. [9] virtualize a single router on an FPGA, allowing the simulation of a 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 is implemented entirely on-chip and does not suffer from this bottleneck.

3. DART Architecture

A basic NoC simulation requires that flits be forwarded around the network while modeling the timing of flit transfers. In DART we abstract the components of a NoC into three primitive types: *Flit Queues* (FQs), *Traffic Generators* (TGs), and *Routers*. In the DART architecture (Figure 1), each node contains FQs, a

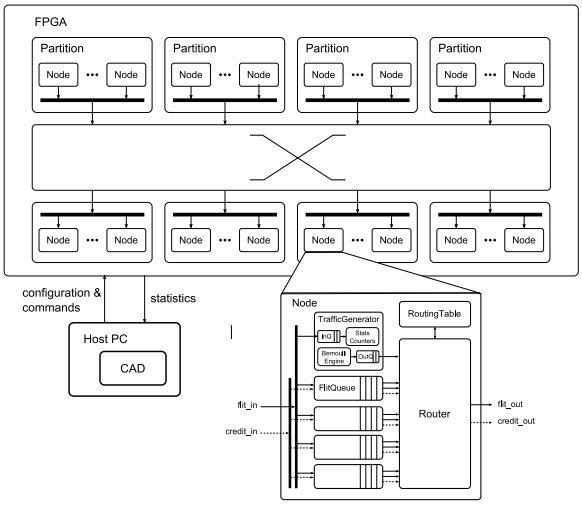

Figure 1. DART architecture. Each node models a 5-ported router, and nodes in the same partition share a crossbar port. The host PC communicates with the FPGA over a serial link.

TG, and a Router, although the TG may or may not be used. Nodes are grouped into partitions that are interconnected by a crossbar, allowing all-to-all communication between any node pairs. DART simulates a NoC by mapping the simulated NoC to DART nodes, and restricts the communication pattern through the interconnect to model the connectivity of the simulated topology. Each DART node has parameters that can be configured to match the properties of the component they simulate, without modifying the DART HDL.

To capture the timing of flit transfers, we define a *time step* as a flit cycle in the simulated NoC. A global counter keeps track of current simulation time, and is only incremented when all flit transfers for this time step are performed. A time step may take multiple DART cycles to simulate. This virtualization of time allows DART to trade speed for area efficiency.

We use *flit* and *credit descriptors* to model the traffic within the simulator. The 32-bit flit descriptor contains (i) the metadata for routing and flow control, (ii) a timestamp that indicates when the flit should be forwarded to the next node, and (iii) the injection time to compute latency at the destination. The 11-bit credit descriptor only contains a timestamp and a VC identifier. A flit's lifetime starts when it is injected at a TG. It is then alternately forwarded between FQs and Routers until it is received at the destination.

3.1. Flit Queue

The Flit Queue (FQ) component models the bandwidth and latency constraints of a wire link. Each FQ encapsulates multiple virtual channels (VCs). The VC buffers are implemented using a block-RAM that is statically partitioned among the VCs. A Verilog parameter controls the number of VCs to incorporate (set to two for this paper). Each incoming flit is queued according to its VC and the new dequeue timestamp is computed using the following algorithm:

```
N_through ++
if( T_enqueue>T_last_flit || N_through>=bandwidth )
     T_dequeue = max( T_enqueue, T_last_flit+1 )
     N_through = 1
else
     T_dequeue = T_enqueue
T_dequeue += latency
```

Here $N_{through}$ counts the number of flits through the FQ during a time step. $T_{lastflit}$ is the dequeue timestamp of the previous flit less the link latency. The $latency$ and $bandwidth$ parameters are configurable on a per-FQ basis.

3.2. Traffic Generator

When enabled, the Traffic Generator (TG) component injects packets using a Bernoulli process. Packet size (minimum 2 flits), destination node address, and the average injection interval are configurable per-TG. TGs also serve as traffic sinks. They record the number of packets received and the cumulative packet latency. Each TG 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 et al. [3] and allow a TG to lag behind the current simulation time when its output buffer is full, to model an infinite source queue.

3.3. Router

The Router component models a canonical five-stage wormhole VC router with credit-based flow control [3]. The number of ports is controlled by a Verilog parameter, set to five for this paper. Each Router connects to four FQs that model per-port input buffers and one TG. Table-based routing is used, and the table content is configurable for each Router without reprogramming the FPGA. In contrast to a real 5-ported wormhole router, the Router component forwards one flit per DART cycle. A time step is simulated by iterating over all ready input FQs and processing them sequentially over multiple DART cycles. Meanwhile, the global time counter is stalled so all flits appear to be routed in the same time step. By doing so, we can use simple arbiters to simulate the switch allocator and the crossbar, which constitute a significant portion of the area in a real router. Head-of-line blocking can arise if the selected input VC cannot be routed due to failed VC allocation or lack of credits. We solve this by setting an *inspected* flag for the offending VC so it is not selected again until the next time step. This creates a bubble in the Router pipeline and wastes a DART cycle.

Pipeline latency of the simulated router is modeled by incrementing the flit timestamp when it leaves the Router; this value is configurable per Router. Contention in VC and switch allocation are also modeled by adjusting the timestamp appropriately.

Credit-based flow control is implemented using a credit counter for each output VC, for which initial

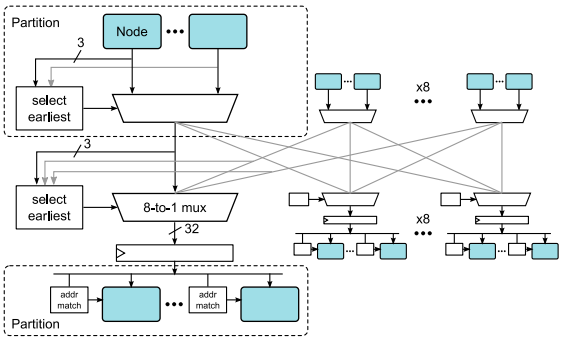

Figure 2. DART's 8×8 concentrated crossbar interconnect

credit values are configurable. When a flit is routed, the output VC's counter is decremented, and a credit is sent to the corresponding input FQ of the upstream Router through the global interconnect. The corresponding counter is updated when the credit is received.

3.4. Interconnect

The global interconnect provides all-to-all communication between DART nodes. By restricting the actual communicating pairs in the routing tables, DART can simulate any regular or irregular topology, provided the maximum node radix is less than the number of input ports in the Router component. Although a crossbar is a first intuition, its area increases quadratically with network size which can be expensive as DART's size increases. Instead, DART nodes are grouped into partitions and interconnected by a concentrated crossbar (Figure 2). Both intra- and inter-partition arbitrations use the three LSBs of the flit timestamp as priority to guarantee chronological simulation of flit transfers. The partitions are the main throughput bottleneck of the interconnect since each partition can only send and receive one flit per DART cycle. Varying the degree of concentration trades crossbar area for performance. We currently use an 8×8 crossbar as it strikes a good area/performance balance. Section 4 discusses these trade-offs in more detail. Credit traffic uses a separate interconnect that is similar but narrower.

3.5. Configuration and Data Collection

As mentioned earlier, each DART node has configurable parameters (packet size, latency, bandwidth, etc.). With the exception of the routing tables, these parameters are chained in a 16-bit shift register. A software tool on the host PC sends the configuration bits over an RS232 serial interface. The block-RAM-based routing tables are connected to the input end of the shift register. An finite state machine captures a chunk of the configuration bits to populate the block-RAM. When configuration completes, an enable signal is asserted to start the simulation.

Similar to the configuration registers, a 16-bit shift register is used to read simulator output performance counters. We currently have three counters per TG to measure the number of injected and received packets and the cumulative packet latency. More counters can be easily added to this shift register chain. Since configuration and stats collection are only performed once before and after the simulation, the host PC–FPGA communication latency is not performance critical.

3.6. Using DART

We have automated the process of mapping a user's simulated NoC to DART. In the current implementation, the configuration management tool runs on a host PC and communicates with DART over an RS232 serial link. This interface is not performance critical, since it is only used for configuration, commands, and gathering results. We use a round-robin scheme to evenly distribute the simulated NoC nodes across the DART partitions—because the global interconnect is symmetric, balancing the load this way is sufficient to achieve good performance.

4. Evaluation

In this section, we quantify the trade-offs made in the design of DART and evaluate its performance by comparing with Booksim 2.0 [3], which is widely used in NoC studies. The results presented are obtained using a cycle-accurate simulator of the DART architecture, which has been verified to be identical to our actual HDL implementation.

DART's simplified Router component allows a 9-node DART engine to fit on a Virtex II Pro (XC2VP30) FPGA while modeling detailed VCs, wormhole routing, credit-based flow-control, and providing run-time configurability. However, there is a performance penalty because DART serializes the switching that is normally performed concurrently in a real router. Using the number of time steps simulated per DART cycle (*SPC*) as a performance metric, Figure 3 shows the Router component's impact on performance for two simple benchmarks. Both cases use 2 VCs, 5-flit VC buffers, and permutation traffic with 2-flit packets and flit injection rates between 0.1 and 0.8. While the ideal SPC is 1.0, we expect DART's achievable SPC to be inversely proportional to the average number of busy input ports per time step. This agrees with the figure, since DART's performance degrades gracefully with increased network load.

A second trade-off in DART is the use of the concentrated crossbar interconnect to allow the simulation of arbitrary topologies without modification to the DART HDL. The contention at the crossbar input and output ports is the main performance bottleneck. For a DART simulator with a fixed number of nodes, increasing the crossbar size alleviates this contention by reducing the size of the partitions. However, crossbar area grows quadratically with the number of ports. Figure 4 shows the scalability of various crossbar configurations for

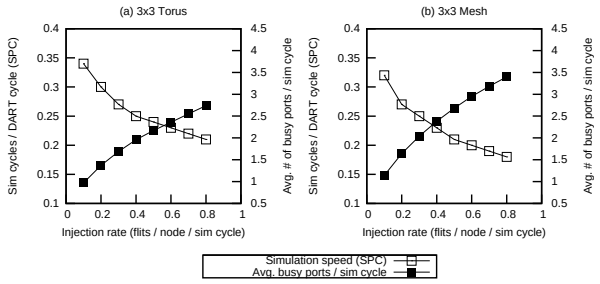

Figure 3. Performance impact due to the serialization in the Router component

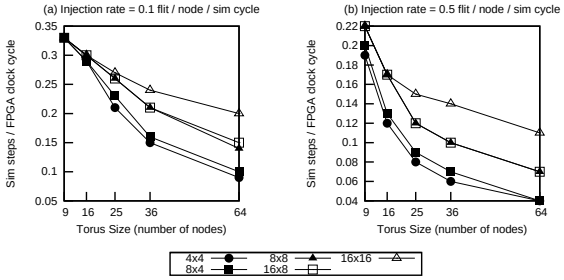

Figure 4. Performance impact of crossbar sizes

different DART sizes. Each data point is evaluated using a torus with permutation traffic. The results for the 4×8 and 8×16 configurations are not shown here because their performance is similar to that of the 8×4 and 16×8 configurations respectively. Relative to a square crossbar, doubling the number of either the input ports or output ports only improves performance slightly as the asymmetric configurations do not fully remove the contention due to concentration. Hence, the largest square crossbar that meets the area constraint should always be used, which is 8×8 for the FPGA hardware in use.

Our current implementation of the 9-node DART uses 13,050 slices (95% available) on the XC2VP30 FPGA and runs at 50 MHz. The critical path includes the logic that selects the earliest flits to cross the global interconnect. Using the same benchmarks as in Figure 3, we compare the number of milliseconds it takes DART and Booksim to simulate a time step. We measured Booksim's runtime on a 2.66 GHz Intel Core 2 Quad linux workstation and averaged the measurements over 20 runs. Both simulators simulated approximately 30K cycles and the reported average packet latencies agree within 5%. Figure 5a shows DART's speedup. The upward trend is because the 9-node benchmarks do not saturate the global interconnect even at high injection rates. As a result, DART's performance degrades more slowly than Booksim. Under similar network load, Booksim exhibits worse scaling than DART when network size increases (Figure 5b); however, the interconnect can become the bottleneck in larger DART systems, in which case additional network load results in declining speedup.

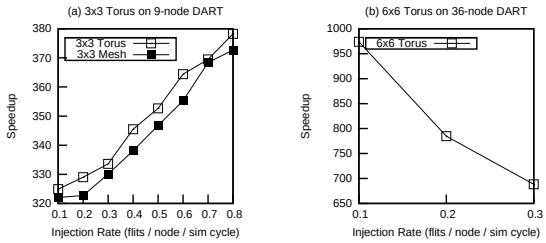

Figure 5. Speedup of DART vs. Booksim

5. Conclusions and Future Work

In this paper, we introduced a flexible NoC simulation engine that provides software-like accessibility to the FPGA and achieves significant speedup over software simulators while maintaining the same level of accuracy. The DART simulator architecture is decoupled and independent from the simulated NoC and allows us to trade speed for programmability and area efficiency. Moving forward, we plan to add multiple contexts to each DART node to allow the user to further trade speed for simulated network capacity if necessary. It is also possible to integrate DART with a full system simulator by replacing the synthetic Traffic Generators with application or trace driven ones to enable comprehensive evaluation of the entire system.

References

[1] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," in *Performance Analysis of Systems and Software, ISPASS*, Apr 2009.

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

[3] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[4] N. Genko, D. Atienza, G. De Micheli, J. Mendias, R. Hermida, and F. Catthoor, "A complete network-on-chip emulation framework," in *Design, Automation and Test in Europe, DATE*, Mar 2005.

[5] N. Hardavellas, S. Somogyi, T. F. Wenisch, R. E. Wunderlich, S. Chen, J. Kim, B. Falsafi, J. C. Hoe, and A. G. Nowatzyk, "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.

[6] Y. Krasteva, F. Criado, E. de la Torre, and T. Riesgo, "A Fast Emulation-Based NoC Prototyping Framework," in *Reconfigurable Computing and FPGAs*, Dec 2008.

[7] V. Puente, J. Gregorio, and R. Beivide, "SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems," in *Euromicro Workshop on Parallel, Distributed and Network-based Processing*, 2002.

[8] G. Schelle and D. Grunwald, "Onchip interconnect exploration for multicore processors utilizing FPGAs," in *2nd Workshop on Architecture Research using FPGA Platforms*, 2006.

[9] P. Wolkotte, P. Holzenspies, and G. Smit, "Fast, Accurate and Detailed NoC Simulations," in *Networks-on-Chip, NOCS*, May 2007.