Fast Scalable FPGA-Based Network-on-Chip Simulation Models

Michael K. Papamichael
papamix@cs.cmu.edu

Cambridge, UK, July 2011
This Year’s MEMOCODE Contest

Objective
- Build the fastest simulator for a class of Networks-on-Chip
- Replicate cycle-by-cycle behavior of SW reference simulator

Simulator takes two inputs

1. Network configuration
   - number and input/output configuration of routers
   - network topology
   - number of virtual channels
   - credit delay cycles

2. Routing info and traffic pattern
   - routing information for each network router
   - number and type of packets to send

Many parameters → very large design space!
Dual-Engine NoC Simulator

- FPGA-based solution
  - Developed in Bluespec System Verilog
  - Targets the Xilinx ML605 board
  - Consists of two NoC simulation engines

- NoC simulation engines
  - High-performance direct-mapped engine
    - Supports up to moderately sized networks (~100 routers)
    - Provides 500x-1000x speedup
  - Highly scalable virtualized time-multiplexed engine
    - Supports all possible networks in the design space
    - Provides 5x-50x speedup

Dual-engine approach effectively covers entire design space
Outline

- Introduction
- FPGA-based NoC Simulator
  - FPGA Simulation Platform
  - Direct-Mapped NoC Simulation Engine
  - Virtualized NoC Simulation Engine
- Results
- Discussion
- CONECT + Demo
Outline

- Introduction
- FPGA-based NoC Simulator
  - FPGA Simulation Platform
  - Direct-Mapped NoC Simulation Engine
  - Virtualized NoC Simulation Engine
- Results
- Discussion
- CONNECT + Demo
FPGA-based Simulation Platform

Steps to run a simulation

1. Generate FPGA design for NoC config.
2. Configure FPGA, MicroBlaze & DRAM
3. Initialize NoC sim. w/ routing+traffic
4. Run sim. until termination condition
5. Extract simulation results

Host PC

MicroBlaze

DRAM

NoC Simulator
(direct-mapped or virtualized)

Xilinx ML605 Board

1. Generate FPGA design for NoC config.
2. Configure FPGA, MicroBlaze & DRAM
3. Initialize NoC sim. w/ routing+traffic
4. Run sim. until termination condition
5. Extract simulation results
Direct-Mapped NoC Simulator

- Why simulate... when you can prototype!

- Direct implementation of target NoC on the FPGA
  - Instantiates all routers, links, traffic sources, etc
  - Collection of routers connected according to NoC configuration
  - Additional logic required to detect termination conditions

- High performance at the cost of limited scalability
  - Achieves 500x-1000x speedup over software reference design
  - ML605 can fit up to ~100 routers of moderate complexity
    - 5-input/output, 4VC router occupies ~1% of LX240T FPGA
  - Need a more scalable solution for remaining design space
Direct-Mapped Router Architecture

- High-level block diagram of parameterized router
Virtualized NoC Simulator

- If resources start getting scarce, virtualize!

- Time-multiplexed implementation
  - Routers are simulated one at a time in successive clock cycles
  - Router, traffic source and link state stored in on-chip memory
  - Special care to retain proper ordering of events [Pellauer ’11]
  - Aggressive prefetching to maintain high simulation throughput

- Scales to very large networks with complex routers
  - Can cover entire design space on ML605
    - 256-router network occupies ~85% of LX240T
  - Only used when direct-mapped approach will not fit
Virtualized NoC Simulator Details

Router State
- Flit Buffers
- Credits
- Route Tables
- Other Scheduler State

Virtualized Router

Router Logic

Virtual Links
- Credit Links
- Delay
- Flit Links
- Flit/Credit Conn. Table

Traffic Sources
- Traffic Table

Traffic Table

Virtual Links

Traffic Sources

Traffic Table

Traffic Sources

Traffic Table

ML605 Usage (for largest network)
- 360 BRAMs (87%)
- 75% LUTRAM
- 60% Logic

≤ 102 BRAMs
≤ 40% Logic
≤ 256 BRAMs
≤ 1184 Kbits LUTRAM
Outline

- Introduction
- FPGA-based NoC Simulator
  - FPGA Simulation Platform
  - Direct-Mapped NoC Simulation Engine
  - Virtualized NoC Simulation Engine
- Results
- Discussion
- + Demo
Direct-Mapped Implementation Results

- LUT usage and frequency for single router

LUT usage and frequency for different numbers of input/output ports:

- 2 VCs:
  - @ 152 MHz
  - @ 101 MHz
  - @ 59 MHz

- 4 VCs:
  - @ 81 MHz
  - @ 54 MHz
  - @ 62 MHz
  - @ 20 MHz

- 8 VCs:
  - @ 30 MHz
  - @ 45 MHz
  - @ 14 MHz
Virtualized Implementation Results

- LUT usage and frequency for 256-router network

<table>
<thead>
<tr>
<th>Number of Input/Output Ports</th>
<th>2 VCs</th>
<th>4 VCs</th>
<th>8 VCs</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>@ 66 MHz</td>
<td>@ 56 MHz</td>
<td>@ 34 MHz</td>
</tr>
<tr>
<td>8</td>
<td>@ 35 MHz</td>
<td>@ 28 MHz</td>
<td>@ 17 MHz</td>
</tr>
<tr>
<td>12</td>
<td>@ 30 MHz</td>
<td>@ 19 MHz</td>
<td>@ 7 MHz</td>
</tr>
<tr>
<td>16</td>
<td>@ 17 MHz</td>
<td>@ 12 MHz</td>
<td>@ 12 MHz</td>
</tr>
</tbody>
</table>
Results for Contest Networks

- Five network and router configurations

<table>
<thead>
<tr>
<th>Network Name</th>
<th>Routers</th>
<th>Ports/router</th>
<th>VCs</th>
<th>Credit Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>butterfly</td>
<td>112</td>
<td>3</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>highradix</td>
<td>16</td>
<td>16</td>
<td>8</td>
<td>15</td>
</tr>
<tr>
<td>mesh</td>
<td>253</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>torus</td>
<td>252</td>
<td>7</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>hypercube</td>
<td>256</td>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Implementation results for contest networks

<table>
<thead>
<tr>
<th>Network</th>
<th>Xilinx Virtex-6 LX240T</th>
<th>Xilinx Virtex-6 LX760T</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sim. Type</td>
<td>% LUTs</td>
</tr>
<tr>
<td>butterfly</td>
<td>Direct Map</td>
<td>86%</td>
</tr>
<tr>
<td>highradix</td>
<td>Virtualized</td>
<td>63%</td>
</tr>
<tr>
<td>mesh</td>
<td>Virtualized</td>
<td>3%</td>
</tr>
<tr>
<td>torus</td>
<td>Virtualized</td>
<td>8%</td>
</tr>
<tr>
<td>hypercube</td>
<td>Virtualized</td>
<td>8%</td>
</tr>
</tbody>
</table>

All configurations lie at “edge” of design space, i.e. max-out at least one parameter.
Outline

- Introduction
- FPGA-based NoC Simulator
  - FPGA Simulation Platform
  - Direct-Mapped NoC Simulation Engine
  - Virtualized NoC Simulation Engine
- Results
- Discussion
- CONNECT + Demo
Design Principles

- **Correctness First**
  - First only focused on correctness and verify design
  - Then optimize entire system and individual components

- **Parameterization and Modularity**
  - Built parameterized versions of all submodules
  - Utilize Bluespec’s powerful parameterization mechanisms

- **Harnessing the power of Bluespec**
  - Static elaboration for easy parameterization
  - Define clean interfaces for modularity
  - Rely on type checking for early bug detection
  - Maintain high-level of abstraction to harness complexity
// Main simulation rule for virtualized design. Simulates network one router at a time.
rule simulate_vcycle (init_done);

// Gather incoming flits/credits and router state
let source_flit = virtualSources.getFlit(cur_router); // Gather flits from source
in_links.in_flits = gatherIncomingFlits(source_flit, flitConnections, flitLinks); // Gather flits from routers
in_links.in_credits = gatherIncomingCredits(creditConnections, creditLinks); // Gather credits
RouterState_t router_state_before = routerState.value(); // Get router state

// Simulate cycle for this router and write new router state
let router_simulation_result = virtualRouter.simulateCycle(router_state_before, in_links, ...); // extract new router state
RouterState_t router_state_after = tpl_1(router_simulation_result); // Write new router state
RouterOutLinks_t out_links = tpl_2(router_simulation_result); // extract flits/credits to send
routerState.write(cur_router, router_state_after); // Write new router state

// Send outgoing flits/credits
flitLinks.putFlits(cur_router, out_links.out_flits, tickVirtualClock); // Send flits to other routers
virtualSources.putCredits(cur_router, out_links.out_credits[0]); // Send credits to traffic sources
creditLinks.putCredits(cur_router, out_credits_to_routers, tickVirtualClock); // Send credits

// Advance to next router
cur_router <= next_router;
endrule
Outline

- Introduction
- FPGA-based NoC Simulator
  - FPGA Simulation Platform
  - Direct-Mapped NoC Simulation Engine
  - Virtualized NoC Simulation Engine
- Results
- Discussion
- CONECT + Demo
Taking it to the next level - CONECT

- Direct-mapped approach implements an actual NoC
  - Parameterized
  - FPGA-friendly
  - Supports arbitrary network topologies

Build on top of this to create a useful tool!

CONECT: Configurable Network Creation Tool

- Highly parameterized Network-on-Chip generation tool
  - # routers, topology, routing, allocation, # VCs, buffer width/depth, etc
- Back end developed in Bluespec System Verilog
- Python command-line interface and web interface (demo)
CONECT Web Interface Demo
Thanks!

Questions?