Using the Liberty Simulation Environment

Manish Vachharajani, David Penry, Neil Vachharajani
Spyridon Triantafyllis, Ram Rangan, Jon Wu,
Jason Blome
Prof. David August

The Liberty Computer Architecture Research Group
http://liberty.princeton.edu/
Princeton University
The Liberty Tools Vision

Application(s) → Liberty Compiler → Optimized Executables

Machine Description → Liberty Simulator Builder → Machine Simulator Instance
Tools and Methodologies
Simulation

- Architects specify architecture informally
- Simulator writers construct simulator from specification
  - Usually written in a sequential language such as C or C++
  - Major architectural changes are expensive to implement
- Why so expensive?
  - Lack of reuse
  - No separation of concerns
    - Control and data
    - Timing and functionality
  - Cognitive mismatch between the model description and the machine

Machine Simulator
The Mapping Problem

- Mapping can be laborious and potentially error prone
- Temptation to approximate difficult portions of the architecture
- Sequential code offers little to assure architects of mapping correctness
- Seemingly innocuous changes in the simulator --> unanticipated structural change in the microarchitecture
The Mapping Problem in Practice

- Must constantly remap from hardware to simulator to avoid pitfalls
  - Mapping process slow and laborious
  - Locked into big architecture decisions without experimentation
  - Can’t keep simulator up to date
    - Simulator doesn’t help for VLSI feedback since it is no longer accurate
- Reverse mapping is difficult – Not Transparent
  - Simulators are hard to validate -- can’t trust simulator results
  - Simulators don’t always model the hardware well [Hennessy/Flash vs. Flash]
- Mapping accuracy important
  - Design details must accurately be modeled to show true performance of architecture [Berger/sim-alpha]
- Temptation to approximate is strong
  - Timing is often approximated, without validation of approximation [Emer/Asim]
Addressing the Mapping Problem

- Build a simulator from a transparent specification – there’s no need to map
- Keep simulator artifacts hidden from users

- Hardware is concurrent
  - Description **must be** concurrent

- Hardware is structural
  - Description **must be** structural

**The Liberty Simulation Environment**
The Liberty Simulation Environment

- LSE specification
  - Modules
  - Architectural blocks
  - Communication elements
  - Hierarchical
- Connections
  - Modules have ports
  - Ports connected to form machine description
- LSE simulator constructor
  - Builds a simulator from the **concurrent structural** specification
Not all concurrency is created equal!

- Not all models of concurrency solve the mapping problem
  - Timing restrictions (SystemC)
  - All inputs needed to produce any output (EXPRESSION, Asim)
    - No zero-latency loops
- Require repartitioning - merging/splitting blocks
  - Partitioning depends on configuration
  - Repartitioning => Remapping!
LSE Concurrent Semantics

- Zero-latency connections

- Zero-latency loops allowed
  - Modules perform partial evaluations

- Iterate until convergence
  - Guaranteed: output signals change only once
  - Efficient: a schedule can be generated

- There is no need to repartition in LSE
Concurrency and Reuse

• The right kind of concurrency solves the mapping problem and allows modularity
  • No repartitioning needed

• Modularity can lead to reuse of modules....
  • Faster modeling
  • Better verification
  • Iterative design

• To achieve high reuse, need to separate concerns:
  • Functionality from timing
  • Datapath from control
  • Model from data collection
Separation of Functionality and Timing

- Modules provide different behaviors:
  - Functionality (bpred, nextpc)
  - Timing (flop)

- Functionality and timing independent
  - Can reuse functional blocks with new timing
  - Can reuse timing blocks with new functionality

- Zero-latency connections and loops essential
Module of Computation
Details for the Configurer

1. Any time a module’s inputs change it will be scheduled for activation this cycle. (Remember modules can be scheduled more than once per time cycle).

2. When a module is activated it must produce any outputs it can compute given its current state and its inputs.
Separation of Datapath from Control

- Control explicit
  - Connection creates forward and backwards control signals
  - Module manipulates control separately from data

- Control signal manipulation can be overridden in machine description
  - Separates datapath and control
  - Reuse of module functionality with new control

- Again, only possible with zero-latency loops
Separation of Datapath from Control
Default Flow Control Style

- **DATA** – information flowing over a connection
- **ENABLE** – forward control signal instructing use of data
- **ACK** – reverse control signal indicating availability of resources

Data **independent** resource availability
1. Receiver sends ACK
2. Sender sends DATA and ENABLE, where ENABLE=ACK.

Data **dependent** resource availability
1. Sender sends DATA
2. Receiver looks at data and sends appropriate ACK
3. Sender looks at ACK and sends ENABLE=ACK
Separation of Datapath from Control Functions
Separation of Model from Data Collection

• Each module emits events during execution
  • Events have attached per-event data
• We define statistics to catch these events
  • Compute summary statistics
  • Display messages for debugging/logging
• Can collect new data, reusing timing, functionality, and control
Additional Features for Reusability

• Parameters of modules can include algorithms (user points and user functions)

• Unconnected ports can have semantics

• Unanticipated connections can be made
  • Algorithmic parameters and control logic can look at state exported from other modules or at port state
  • Queries export state
  • Control and user functions can call queries
Hands On Exploration

Let’s use the system a bit to get comfortable with it. Then we’ll explain module writing....
Installing Liberty

- SSH to tolerance.cs.princeton.edu
- Set your CVSROOT environment variable
  
  ```
  $ export CVSROOT=:ext:yourname@forest.cs.princeton.edu:/liberty/repository
  $ export CVS_RSH=ssh
  ```

- Check out the source code
  
  ```
  $ mkdir liberty
  $ cd liberty
  $ cvs checkout src
  ```

- Set up your environment
  
  ```
  $ src/scripts/scripts/l-env > ~/.liberty
  $ source ~/.liberty
  ```

- Build the code
  
  ```
  $ cd src
  $ ./goLiberty
  ```
Installing Liberty
Setup IMPACT and Testing

- Setup IMPACT environment variables
  
  $ cp ~nvachhar/.impact ~
  $ source ~/.impact
  $ export USER_BENCH_PATH1=/liberty/benchmarks

- Test the installation
  
  $ cd
  $ mkdir liberty-test
  $ cd liberty-test
  $ cp ~/liberty/src/simulator/test/features/demo.xml .
  $ echo wc | ls-prep-bench O -spath /liberty/runs_noi_nos
  $ ls-build demo
  $ echo wc | ls-run-bench demo
Writing Modules

- Modules are written in stylized C
  - Module writers produce a .xml file describing the module
  - And a .clm file which has the code
  - The .clm file gets preprocessed to form a .c file
- Keeping state and specifying behavior
  - Global variables in the .clm file defines per instance state
  - Function specify behavior
  - There is a API between the module code and the framework
  - You define callbacks that the framework calls when it needs the module to do some work
  - The framework provides functions for modules to inquire about system state
Stylized C

- To define a global variable
  - GLOBDEF(type, variable name) – to define a global variable
  - GLOB(variable name) – to use a variable
  - return-type FUNC(function name, argument list) – to define a call back function
  - void HANDLER(port name) – to define a port handler

- Callback functions
  - init – called at the beginning of simulation
  - finish – called at the end of simulation
  - phase_start – called at the beginning of each cycle
  - phase_end – called at the end of each cycle
  - phase – called whenever a module’s inputs change
Phase vs. Handlers

- LSE supports two methods of activating modules when inputs arrive
- Phase function
  - This gets called when any input (not associated with a handler) changes state
  - Use when aggregating a lot of inputs to produce few outputs
- Handler (per port)
  - This gets called when the given port’s status change
  - Use when input individually produces output

It’s probably **not** a good idea to mix handlers and phase functions
Configurations

- ~/nvachhar/liberty-test/full.xml