PROTOFLEX: Co-Simulation for Component-wise FPGA Emulator Development

Eric S. Chung, James C. Hoe, Babak Falsafi
{echung, jhoe, babak}@ece.cmu.edu

SimFlex
Computer Architecture Lab (CALCM)
Carnegie Mellon University
http://www.ece.cmu.edu/~simflex
Motivation

• **Community goal**
  – FPGA shared-memory multiprocessor research infrastructure

• **Development obstacles**
  – Functional verification of multiprocessor RTL not easy!
  – Distributed collaborators with independent research goals
  – Don’t have teams of engineers but few (smart) researchers

• **RTL development method for researchers needed**
  – HW functional validation for target model
  – Concurrent development
  – Gradual transition to full emulation
PROTOFLEX

- Systematic methodology for FPGA emulator development
  - Rely on validated component-based simulators for reference
  - Create equivalent RTL piece-wise—then co-simulate for validation

**Software-only simulation reference system**

- Core (C++)
- Caches (C++)
- Coherence Engine (C++)
- NIC (C++)

**Verification of RTL with co-simulation**

- Core (C++)
- Caches (C++)
- Coherence Engine (Verilog)
- NIC (C++)

- **Advantages**
  - Gradual SW to HW transition
  - Concurrent RTL development of agreed reference model
  - Subsystem characterization
Necessary Ingredients

• Agreed-upon simulator w/ component-based interfaces
  – Examples: FLEXUS, ASIM, Liberty, etc.
  – Our simulator choice: FLEXUS
  – Cycle- and execution-driven, component-based simulator
  – Full-system support executes unmodified workloads + OS
  – 20 components to support DSM, CMP configurations

• Generate interfaceable software objects from RTL
  – HDL to C++ generator: Verilator
  – Compiles synthesizable Verilog into equivalent C++ object
  – Object can be instantiated in software and “clocked”
  – Component wrapper to map between RTL signals & data structures

**FLEXUS + Verilator enable PROTOFLEX methodology**
Outline

I Motivation
II PROTOFLEX
III Necessary Ingredients
IV Case Study: Cache Coherence
V Testing Strategies
VI Limitations
VII Conclusion
Case Study: Cache Coherence

- **Protocol Engine in FLEXUS**
  - Distributed, MSI directory-based protocol based on Piranha
  - Protocol verified in Murphi
  - Performance-optimized (e.g., NAK-free)
  - Protocol microcoded in symbolic C-like language
  - Parameterizable (1 to 32 transactions)

- **Porting to RTL**
  - C++ model → Bluespec → Verilog (→ Verilated C++ object)
  - Interfaced to FLEXUS’s distributed shared memory timing model
  - Same microcode from C++

**PROTOFLEX enabled design + validation in 6 weeks**
Essential for FPGA emulator component development
Case Study: Cache Coherence

Home Engine
- Handle request to home memory
- 7000L C++ / 4000L Bluespec
- 8000 slices, 46 MHz*

Remote Engine
- Handle request to remote memory
- Same code/timing/slices as Home

Local Engine
- Optional hardwired fast path for local accesses
- 1000L C++ / 2000L Bluespec
- 14000 slices, 84MHz*

* Synthesis Target: Xilinx Virtex-II Pro 70

Home & Remote engines occupy 50% Virtex-II Pro 70 w/o tweaks
Testing Strategies

• **Isolated and in-system component testing**
  – Trace errors (e.g., deadlocks, bad responses) to culprit RTL
  – Run in realistic operating conditions (boot unmodified Solaris)
  – E.g., races in OLTP on DB2/Oracle, memory-bound cases in Ocean

• **Advanced simulator test and debug features**
  – Adjustable debugging levels, assertions identified propagating errors
  – Surrounding components (e.g., Cache, NIC) detected bad messages
  – Adjustable simulator system configuration to force rare corner cases, e.g., writeback races
  – Collect simulation checkpoints as regression suite
    – e.g., sample multi program phases for coverage rather rerun whole program

• **With PROTOFLEX, generate representative test cases quickly**
  – Testing infrastructure + benchmarks already present
  – Handwritten testbenches for multi-node interactions would be HARD
Limitations

• Validated RTL only good as reference simulator
  – Microarchitectural details may be absent
  – E.g., simulator cache, protocol engines only keep addresses not values

• Co-simulation performance
  – RTL-level simulation of component dominates co-simulation time
  – But, alternative is to simulate entire system in RTL

• Simulator metadata
  – Some communication contain metadata for statistics tracking
  – Must facilitate by component wrapper or implement in RTL
Conclusion

• **FPGA emulator development is hard**
  – Numerous, complex components to develop in RTL
  – Distributed collaborators

• **PROTOFLEX enables:**
  – Refinement path towards implementing full-system MP emulator
  – Concurrent development of infrastructure
  – Accelerate robust bring-up of final design in FPGA
  – Subsystem validation and characterization
Additional Information

• **FLEXUS** available at [www.ece.cmu.edu/~simflex/](http://www.ece.cmu.edu/~simflex/)
  – Tutorial at ISCA 2006 (July)

• Verilator and tutorials available at [www.veripool.com](http://www.veripool.com)

• **PROTOFLEX** being developed for TRUSS project
  – Total Reliability Using Scalable Servers
  – [www.ece.cmu.edu/~truss](http://www.ece.cmu.edu/~truss)

• Thanks! Questions? echung@ece.cmu.edu
Backup Slides
Co-simulation Methods

- Functional verification
  - Detect functional divergences
- Timing verification
  - Detect any timing divergences
FLEXUS Component Abstraction

- Component-based simulators
  - Built from timing-independent (like RDL), software components
  - 20 components to support baseline DSM and CMP configurations

- Component Interfaces
  - Ports, FIFO Channels, payload is arbitrary C++ data type

- Why care about component abstraction?
  - RTL/SW co-simulation (map payloads to/from RTL signals)
  - Concurrent porting into RTL with agreed reference model

```c
doCycle {
    MemoryMessage aMessage;
    aMessage.theAddress = ...;
    send(aMessage, ...);
}
```

```verilog
always @ (posedge CLK)
    begin
        address[31:0] <= ...;
        addr0 ...;
        addr1 ...;
    end

handleMsg (uint address,..)
{
    ...;
}
```