Depth-Driven Verification of Simultaneous Interfaces

Ilya Wagner, Valeria Bertacco, Todd Austin
Advanced Computer Architecture Lab
University of Michigan
Motivation

- SoCs are becoming the most often used design
SoC – Verification Challenge

- Multiple sub-blocks designed by different people
- All rely on adherence to communication protocols
- Need scalable and flexible framework to verify communication between components
Example Design

- A bus arbitrator

![Diagram showing an arbitrator with inputs from CPU, USB, Memory, and GPU]

- Challenge: Verify corner cases of input interaction
  - High-priority input during a low priority transaction
  - Two simultaneous high-priority inputs
Traditional Verification Methodology

- Limitations
  - Open-loop test generators suffer from low design coverage
  - Hard to specify tests for complex corner cases with concurrent transactions
Contributions

- Enable verification of SoC designs
  - Fits in traditional verification methodology
  - Feed-back based verification system
  - Template language
    - Specify complex corner cases with concurrent transactions on parallel interfaces
    - Hierarchical modeling environment
  - Goal-sensitive feedback metrics
    - Depth analysis of circuit structure
Our Verification Methodology
Markov Model

Markov Model:
- Directed Graph
- Edge labels are probabilities of transition

Vertices:
- Low-level interface commands
- High-level scenario steps
Hierarchical Markov Model

High-level Markov Model

Low-level Markov Models

CPU_port

GPU_port

Design
IQTest: Template Files
Template Files

- Flexible and simple language for specifying the interface
- Use simple constructs to represent a sequence of inputs or a sequence of scenario steps
- Variables pass information deterministically or probabilistically
Example Template

/ Global Variables /
  global { shared_var; }  
/ Global Markov Model /
  TopModel ( global ) {
    cmd [1:0] = { in0, in1 }; 
    vertex (send_one) { ... } 
    vertex(burst) { ... } 
  }
/ Local Markov model /
  local1 {
    using global::shared_var; 
    vertex (send) { ... } 
    vertex (request) { ... } 
  }
/ Binding /
  dut.CPU_port : local1(in0); 
  dut.GPU_port : local1(in1); 

- `shared_var` used for communication between local models
- `cmd` used for signaling to local models
- Only one description of local model needed
IQTest: Activity Analysis
Activity Signals and Analyzer

- Used to steer the test generation towards corner-case scenarios
  - Edges in Markov model reinforced from feedback values

- Internal nodes in the design representing critical activities
  - Port collision signal in network switch
  - Interrupt / branch mispredict in a microprocessor
Activity Signals and Analyzer

- Previous research used a handful of key activity signals
- The feedback was too coarse
  - Prevented exploration of different scenarios
- All signals were user-selected
  - No information was drawn from design itself
Depth-Driven Activity Monitoring

- User-specified signals are used to extract more feedback signals
Depth-Driven Activity Monitoring

- User-specified signals are used to extract more feedback signals
Depth-Driven Activity Monitoring

- User-specified signals are used to extract more feedback signals
Depth-Driven Activity Monitoring

- User-specified signals are used to extract more feedback signals
  - Backward traversal of RTL logic
  - Assign “depth” values to logic driving signals
  - Assign weight proportional to the depth

![Diagram showing logic gates and signal flow with depth levels labeled Depth 2, Depth 1, and Depth 0.](image-url)
Depth-Driven Activity Monitoring

- Property output is used only for correctness check
IQTest
Related Work

Bayesian Networks:

Markov Models:

Weighted BDDs:
K. Shimizu and D. Dill. “Deriving a simulation input generator and a coverage metric from a formal specification.” DAC 2002

Our previous work:
- No hierarchical specification, no depth-driven feedback
Experimental Setup

- Compared IQTest to StressTest and Random
- Implemented IQTest feedback of depths 1,2,3
- Run 25 times each design with different random seeds
- Maximum search effort for each bug is at most 75000 cycles
Designs Under Test

- **DLX Core**
  - MIPS-lite ISA 5 stage pipeline
  - 30 buggy cores (easy-moderately hard)

- **Alpha Core**
  - Alpha 5 stage pipeline
  - 10 buggy cores (hard)

- **Switch**
  - 5x5 crossbar logic /w input buffers
  - 3 virtual channels per input
  - Adaptive cut-through routing version
  - 10 bugs (easy-hard)
Results: DLX pipeline

- Depth-2 performs best for harder bugs
- Similar in Alpha
Results: Alpha pipeline

- Depth-2 performs best for harder bugs
Results: Switch Design

- StressTest cannot be used on this design!
- Found 3 actual bugs in the design
  - Hard corner cases that slipped through Random testing
IQTest finds more bugs faster

- Key contributions
  - Hierarchical modeling environment
    - Allows for verification of multiple simultaneous interfaces
    - High-level models to specify scenarios
    - Low-level models to specify interfaces and stimuli
  - Depth-driven quality evaluation
    - User points to key signals in the design
    - Logic influencing the key signals is used for feedback
    - Increased quality of feedback with depth
Current work

- Adding numeric coverage metric to IQTest
- Enhancing feedback mechanism
- Generating more complex scenarios