AI Driven Verification

Curtis Tsai
Cadence Design Systems
Introduction

AI-driven Verification

Debug
- SemanticDiff
- PinDown
- WaveMiner
- AutoTriage

Manager

Data & AI Platform

Design Repository

Formal
Simulation
Emulation
Prototyping
Virtual Platform

Wave data
Coverage data
Mined attributes
AI models

Mined attributes
AI models
Verification Platform for Fastest Debug TAT

AI-driven submission of tests to compute farm

Platforms Manager

AI Enabled Apps

1. AutoTriage
   - Automatically groups tests failing due to the same underlying bug

2. SemanticDiff
   - Automatically identifies code differences between design versions N and N+1

3. WaveMiner
   - Analyzes waveforms and automatically identifies root cause of bug (signals + time)

4. PinDown
   - Automatically predicts which check-ins are most likely to have introduced failures

Design Version N

GitHub

PERFORCE

Design Repository

Design Version N+1

Simulation

Platform

AI accelerated results for debug

AI-Enabled Debug for fastest TAT
What is Semantic Diff?

An advanced RTL design comparison tool that compares the two versions of RTL design and determines the Semantic Differences between them.

More sophisticated than text-based diff:
- It ignores comments/blank spaces/newlines
- It improves productivity as user can concentrate on files with maximum code changes

It reports the entities and other details of the RTL design with a specific rank where the Semantic Differences, user can concentrate on RTL where rank is higher which implies there are more Semantic Diff’s.
Semantic Diff Flow

Identify and rank semantic changes between two RTL versions

- Ignore harmless changes | Rank “complexity” of genuine logic changes

module cg (d, clk);
input d, clk;
reg orig;
reg clone;
reg g_latch;
wire w = orig ^ d;
wire gclk = clk & g_latch;
always @(clk or w)
if (~clk) g_latch <= w;
always @(posedge gclk) clone <= d;
always @(posedge clone) orig <= d;
fd : assert property (@posedge clk) orig == clone;
endmodule

module cg (d, clk);
input d, clk;
reg orig, clone, g_latch;
// Comments …
wire w = orig ^ d;
wire gclk = clk & g_latch;
always @(clk or w)
if (clk) g_latch <= w;
always @(posedge gclk) clone <= d;
always @(posedge clone) orig <= d;
fd : assert property (@posedge clk) orig == clone;
endmodule
WaveMiner flow

Waveform Analysis → Signals to Design Mapping → Failure Root Cause Ranking → Ranked Culprit Signals and Timepoints

User Defined Scope → Waveform Analysis

Signals to Design Mapping

Failure Root Cause Ranking

Ranked Culprit Signals and Timepoints

Debug Difference on Waveform

Passing View → Snapshots → Back annotation to source code

Failing View → Snapshots → Navigation on culprit signals

Ranked simulation time points

Options: Design releases

Snapshots
WaveMiner Results
Text Report

Signals ranked by relevance

Full hierarchical signal name

Top <N> ranked culprit signals and simulation timepoints to be investigated by the user

Ranked timepoints per signal

Failing waveform location

Passing waveform location

INFO (IWDG08): Returning top 5 signals.

1) SYSTEM.bridge32_top.bridge.pol_target unit.wishbone.master.r att
   700195000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   699950000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   50000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm

2) SYSTEM.bridge32_top.bridge_in_reg_frame_in
   697950000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   700195000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   697750000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   700165000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   689350000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm

3) SYSTEM.bridge32_top.bridge.parchk_pci_perr_in
   700045000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   679165000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   676950000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm

4) SYSTEM.bridge32_top.FRAME_in
   634855000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   360235000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   649145000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   633475000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   666175000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm

5) SYSTEM.bridge32_top.IDXY_in
   634885000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   360265000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   655255000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   666205000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
   632125000ps, ../multi_rev_dir/waves.shm, ../bad_wave_dir/waves.shm
WaveMiner Results Widget

New widget enables easy navigation on the results and provides high level information.

Signals are ranked in decreasing relevance order.

Double-click on signal, will bring it to waveform widget.

Double-click on time point, will put the debug cursor at that time in debug's waveform widget.
WaveMiner - Visualization

Passing Session

Failing Session

Back annotation to source code

Ranked simulation time point to debug

Top ranked signals

Now you have the two Debug sessions and signal is there, with markers for its time points. The recommendation is to do driver tracing at the time points suggested by WaveMiner.
AutoFocus flow

Regression Data

Test_case1
Test_case2
....
Test_caseN

Output (JSON format)
Test_case1, seed=32121134
Test_case5, seed=23422324
....

The output format includes
• Test case name
• The corresponding random seed

Target
• Hit 90% of original regression coverage for modified modules
AI for Regression Productivity

Machine Learning for coverage closure
Trends in Hardware / Software Development

Exponential Challenge

ROI mindset: Bugs found per $ per day

Verification Throughput
Verification Solution

Find and fix the most bugs per $ compute per day

Total Verification Management
Regression Manager – Debug Platform – VIP – System VIP – C code generator

Smartest Apps

Fastest Engines

Most Choice of Compute

Formal
X86 or Arm® CPU

Simulation
X86 or Arm CPU

Virtual and Hybrid
X86 CPU

Emulation
Custom Processor

Prototyping
FPGA

Verification Cloud
Where Does Machine Learning Fit in a Typical MDV Timeline

**Environment Development**
- Develop / Reuse components
- Create tests
- Add functional coverage model

**RTL Verification / bugfix cycle**
- Developer check in tests
- Nightly runs on code changes
- Weekly complete regressions
- Ends when bug rate hits a suitably low threshold

**Bug hunting**
- Add additional corner case scenarios
- Fill up available resources with randomized runs
- Typically budgeted for a specific amount of time and given specific resources

**Coverage closure**
- Regress continuously until convergence
- Analyze gap to 100%
- Refine and create directed testcases
- Ends with 100% coverage achieved or "close enough", i.e. at schedule deadline
The Machine Learning Flow

Machine Learning analyzes patterns hidden in verification regression results.
Original Regression
- 50 tests, 100 seeds per test (5,000 runs)

Random control

```
class cfg_c extends uvm_sequence_item;
    rand focus_e focus;
    rand [2:0] rank;
    ...
endclass
```

```function void test::setup();
    cfg_c cfg = get_config();
    cfg.randomize();
    set_config_info(cfg);
    endfunction```

Generated Regression
30 tests, 1,500 runs

ML Regression Coverage model

Some bins not regained

Original Regression Coverage model

Generated new regression runs

Some bins newly hit
Xcelium Machine Learning App

Use Cases

Can be used early in design process to find bugs:
- Before even user has started to put coverage
- Target failure signatures rather than coverage
- Recent example: 17 rare signatures. Found 12 more.

Regression Compression - Same coverage in less time
- Replace Original regression with ML generated regression

Requires a modified form of Reinforcement Learning:
- Target unhit cross bins: cover (A x B)
- Target unhit bins using structural and statistical correlation
- Create new streams by stitching sequences

Bug Hunting – Find bugs early
- Augment Original regression with ML generated regression

Extend to Cousin Bug Hunting
- Extend to Coverage Maximization
- Target rare bins
Results – Faster Regression and Matching Coverage

**Regression CPU Cycles**

- **Example 1**: 3X
- **Example 2**: 4X
- **Example 3**: 4.5X

**Coverage**

- **Example 1**: 99%
- **Example 2**: 99%
- **Example 3**: 99%
Using ML for Bug Hunting

- Augment full regression with ML-generated runs
  - The ML-generated regression will create a higher percentage of more rare scenarios
  - The bug rate of the ML runs (unique signature / cpuH) will typically be higher than the full regression
  - Use in conjunction with the full regression until the full regression no longer finds new bug signatures

![Venn Diagram](image)

Unique bug rate vs. Time

ML Bugs Found

Full Regression Bugs Found

- Baseline
- ML
Coverage Closure With Iterative Learning

- Orange is the baseline is regression runs without ML
- Green trains a model after 4 iterations of orange and then continues
- Red does iterative learning after 4 more iterations