Outline

• Introduction
• Motivation
• Automated Failure Triage
  – Overall Flow
  – Signature Generation
  – Failure Binning
• Case Study
• Conclusion
Introduction

- Verification is the most time consuming component of the design cycle
  - Ranges from 50% - 70% of entire effort

- Debug consumes approximately 50%
  - Tracing through waveform and source code (40%)
  - Performing Triage (15%)
  - Re-running simulations (15%)
  - Discussions with colleagues (10%)
  - Others (20%)
Introduction

• Efforts to ease debug pain
  – Assertion Based Verification (ABV)
    • The more assertions are written, the earlier bugs can be found
  – Debuggers and waveform viewers
    • Improved efficiency with tools like Verdi, DVE, SimVision, Questa
  – Automated root cause analysis tools
    • Vennsa OnPoint: Ability to understand origin of failures and how to make fix
  – Academic side:
    • Improving root cause analysis engines and post-silicon debug

• What about Failure Triage?
Outline

• Introduction
• Motivation
• Automated Failure Triage
  – Overall Flow
  – Signature Generation
  – Failure Binning
• Case Study
• Conclusion
Motivation

High level debug: DV engineers
- find general bug area
- identify best engineer to look at it

Mid level debug: DV & Design engineers
- understand cause of bug
- find proximity of source

Low level debug: Design engineers
- understand exact source of bug
- determine how to make the fix
Motivation

• Failure Triage Example:

- Which ones are related, which are not?
- Same failures/same reasons? How many?
- Which ones to “file” as a bug? To who?
- What’s the source: design, testbench, env?
- Who is the best engineer to look at this?

Nightly Regression tests

Probably yours

Not mine

Yours

Not my problem

Takes time & Wastes time
Motivation

- Why is Triage difficult?
- Example: Two different bug sources, same checker
Motivation

- Why is Triage difficult?
- Example 2: One bug, caught by different checkers
Outline

• Introduction
• Motivation
• Automated Failure Triage
  – Overall Flow
  – Signature Generation
  – Failure Binning
• Case Study
• Conclusion
Automated Failure Triage

- Improved binning is based on effective characterization of failures

- Signatures can characterize failures
  - Log and Error messages
    - Provide information on components and functions targeted by tests and conditions of failure
  - Excitation and Propagation path of bug

- Comparison of signatures differentiate failures
Automated Failure Triage

• Flow Overview

1. Simulation → Re-simulation w/ dump on → Waveform 1 → Root Cause Analysis Engine → Signature Generation

2. Simulation → Re-simulation w/ dump on → Waveform 2 → Root Cause Analysis Engine → Signature Generation

3. Simulation → Re-simulation w/ dump on → Waveform 3 → Root Cause Analysis Engine → Signature Generation

ERROR 1

ERROR 2

ERROR 3
Automated Failure Triage

Signature Generation

Signature 1

Signature 2

Signature 3

Failure Binning Engine

Bin 1

Bin 2

Bin X

Designer: Joe

Designer: Tim

DV: Bob
Automated Failure Triage

• Signature generation
  – Goal: identify excitation and propagation paths of bugs

• Accomplished by Root Cause Analysis tools
  – Vennsa OnPoint is one such industrial tool
  – Path Tracing, SAT, QBF, SMT, BDD in academia
Automated Failure Triage

- Signatures can contain:
  - Signals where fixes are propagating
  - Values that are propagating (buggy value or fix value)
  - Time of propagation for that signal
  - The more such information the better

<table>
<thead>
<tr>
<th>Signal</th>
<th>Value</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>acc</td>
<td>1'b1</td>
<td>33400ns</td>
</tr>
<tr>
<td>wacc</td>
<td>1'b1</td>
<td>33400ns</td>
</tr>
<tr>
<td>ctrl</td>
<td>1'b1</td>
<td>33410ns</td>
</tr>
<tr>
<td>ctrl_blank</td>
<td>1'b1</td>
<td>33410ns</td>
</tr>
<tr>
<td>blank</td>
<td>1'b0</td>
<td>33420ns</td>
</tr>
<tr>
<td>pad_o</td>
<td>1'b0</td>
<td>33430ns</td>
</tr>
</tbody>
</table>
Automated Failure Triage

- Pairwise comparisons of signatures determine similarities of propagation paths
- Weights can be assigned to discount common subsets
- What score would you give these two paths?
Automated Failure Triage

- Failure Binning
  - Score of pairwise comparison determines relative similarities
    - algorithm is complex, paths are typically trees
  - Clustering algorithm creates bins from the scores
    - existing graph clustering techniques + heuristics

PATHS

<table>
<thead>
<tr>
<th>Path</th>
<th>Bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
</tbody>
</table>
Outline

• Introduction
• Motivation
• Automated Failure Triage
  – Overall Flow
  – Signature Generation
  – Failure Binning
• Case Study
• Conclusion
Case Study 1

- FPU design
- Correctness verified by:
  - One end-to-end functional checker
  - Exception checker
  - Dozens of assertions
- Sanity simulation ran and “silly” bugs fixed
- Simulation results in 13 errors
  - 8 checkers failed
  - 2 exceptions caught
  - 3 assertions fired
- 5 bugs, 5 bins
Case Study 1

- Bug: Switched case items
- Failures: 2 assertions fire, 1 checker failure and 1 exception failure
- All in one bin

```vhdl
-> always @(signa or signb or add ... 
->   ...
->   // Bug: switched assignments
->   3'b0_0_0: sign_d = 1;
->   3'b0_1_0: sign_d = !fractb_lt_fracta;
->   3'b1_0_0: sign_d = fractb_lt_fracta;
->   3'b1_1_0: sign_d = 0;
->   // Fix:
->   //3'b0_0_0: sign_d = fractb_lt_fracta;
->   //3'b0_1_0: sign_d = 0;
->   //3'b1_0_0: sign_d = 1;
->   //3'b1_1_0: sign_d = !fractb_lt_fracta;
```
Case Study 1

- Bug: signals arrive one cycle early
- Failures: 4 checker failures and 1 assertion fires
- All in one bin

```plaintext
// Bug: missing one clock delay
always @(posedge clk)
    quo <= #1 opa / opb;
always @(posedge clk)
    rem <= #1 opa % opb;

// Fix:
//always @(posedge clk) begin
  // quo1 <= #1 opa / opb;
  // quo  <= #1 quo1;
//end
//always @(posedge clk) begin
  // rem1 <= #1 opa % opb;
  // rem  <= #1 rem1;
//end
```
Case Study 2

- VGA controller
- Correctness verified by:
  - Golden model in C
  - 50 assertions
- Sanity simulation ran and “silly” bugs fixed
- Simulation results in 10 errors
  - 2 checker failures
  - 8 assertions fired (6 different type of assertions)
- 4 bugs, 4 bins
Case Study 2

- Bug: Wrong default value
- Failures: 4 assertions fire (3 different)
- All in one bin

```verilog
case study or vdat_buffer_empty or colcnt or DataBuff
or rgb_fifo_full or clut_ack or clut_q or Ba or Ga of Ra)
  always @(c_state or vdat_buffer_empty or colcnt or DataBuffer
or rgb_fifo_full or clut_ack or clut_q or Ba or Ga or Ra)
  begin : output_decoder

  // initial values
  // Bug incorrect initial value
  ivdat_buf_rreq = 1'b1;

  // Fix:
  ivdat_bug_rreq = 1'b0;
```
Analysis

• Quality:
  – Vast majority of unique bugs can be distinguished
  – Very few are “wrongly” binned together, but are very close
  – Results in hours of savings versus manual comparison

• Performance:
  – Triage algorithm takes minutes for hundreds of failures
  – Root cause analysis takes more time
    • Each failure must be analyzed for signature (minutes to hours)
  – Root cause analysis is better suited for blocks/IP level

• Scaling:
  – Triage did not exhibit scaling issues for 100s of failures
Outline

- Introduction
- Motivation
- Automated Failure Triage
  - Overall Flow
  - Signature Generation
  - Failure Binning
- Case Study
- Conclusion
Conclusion

• Failure Triage is a difficult debugging task
• Hard to distinguish between single/multiple bugs source leading to same/different failures
• Introduced a novel failure triage approach
  – Based on automated root cause analysis
  – Groups similar failures in the same bin
  – Showed effectiveness using small case studies
• Can save lots of time and catch bugs earlier
• More work to scale to system/chip level
  – leverage other signatures when automated root cause analysis not available
Questions?

Thank you

Sean@vennsa.com