Processor Verification with hwBugHunt

Jose Renau

http://masc.cse.ucsc.edu
Processor verification is difficult

More than half of the time spent in verification

Solving a bug requires
  Detection by building testbenches
  Location by ???

This work focusses on the bug location
Processor Flow

- Run test suite
  - Bugs Detected?
    - No
      - Expand test suite
    - Yes
      - Instrument HDL
        - Run test suite
        - Bug solve
  - Enough?
    - No
      - Expand test suite
    - Yes
      - Finish
Gathering Statistics

- Create checkpoints as the testbench executes
- Gather coverage metrics like toggle or line coverage
- Run as many testbenches as possible
- We create a checkpoint whenever an instruction retires

Time

<table>
<thead>
<tr>
<th>Simulator (testbench)</th>
<th>HDL Processor (DUT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OK</td>
<td>OK</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

Coverage Checkpoints

OK

Not-OK (bug detected)

=?
Whenever the code is executed
  - Include it on the coverage checkpoint
  - If a bug is detected
    - Look for lines/values used for the first time recently

**CORRECT:**
0: if (Rav_in == Rbv_in)
1:   temp = `ZERO_64;
2: else
3:   temp = `ONE_64;

**INCORRECT:**
0: if (Rav_in == Rbv_in)
   temp = `ONE_64;
else
   temp = `ZERO_64;
Line of Code Ranking Algorithm

confidence = 0
for_each_checkpoint in Failed run { |nok|
  for_each_checkpoint in Pass run { |ok|
    confidence++
    # Join all ok ckps with oldest nok ckp
    past = Join ok[0:100] nok[0]
    # Join recent nok ckps
    recent = Join nok[1:100]
    # What did happen on the last failed run?
    res = Diff recent past

    for_each_mark in res { |mark|
      counter[mark]++
    }
  }
}

for_all_marks { |mark|
  puts mark, 100*counter[mark]/confidence
}
Setup

- We use an EV6-like processors
  - Illinois Verilog Model (IVM)
  - Over 30K Verilog lines of code

- Introduce bugs on the design

- As testbench we run SPECint applications
  - Compare retiring instructions against architectural simulator
Results

- Correctly locates 62% of the bugs
- 0.6 false positives on average
- 3.1 false positives for incorrectly located bugs
Conclusions

- Simple mechanism to locate bugs on processors
- Highly effective with low number of false positives
- Soon more details on ISQED 2008
Contact Information

Web: http://masc.soe.ucsc.edu/

Jose Renau
renau@soe.ucsc.edu