

Facilitating Unreachable Code Diagnosis and Debugging

Hong-zu Chou and Sy-yen Kuo

National Taiwan University

Kai-hui Chang

Avery Design Systems



ASPDAC
Jan. 27, 2011



[Motivation]

- Code coverage is an important metric in design verification
 - Typically, it is performed using logic simulation with constrained-random testbench
 - May miss corner cases and is not accurate
 - Formal code statement reachability analysis provides proving capabilities [Chou *et al.*, ASPDAC10]
- Code that is proven to be unreachable is called dead code
 - Dead code is usually associated with bugs
 - Diagnosing the cause of problem can be challenging

Challenges in Unreachability Diagnosis

- Engineers perform debugging using waveforms
 - Unreachability means no waveform is available
 - Engineers have to analyze nonexistent paths to identify the cause of the problem
- Automatic error diagnosis and repair methods cannot be used
 - These algorithms require the “correct values” of a signal to be known
 - Unreachability means there are no correct values to be solved for

[Our Contributions]

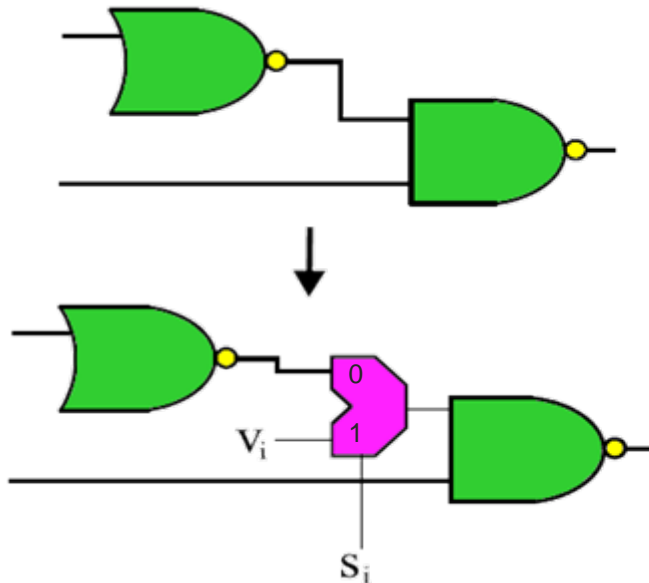
- A new symbolic simulation algorithm that can explore nonexistent paths
 - “Liberated variables” give the algorithm the freedom to explore new paths
 - Cannot be achieved using traditional synthesis-based formal methods
- Error diagnosis is then applied to analyzing the symbolic condition to execute the target code
 - Key variables that contribute to the unreachability will be identified
 - Suggested values to solve the problem is provided

[Outline]

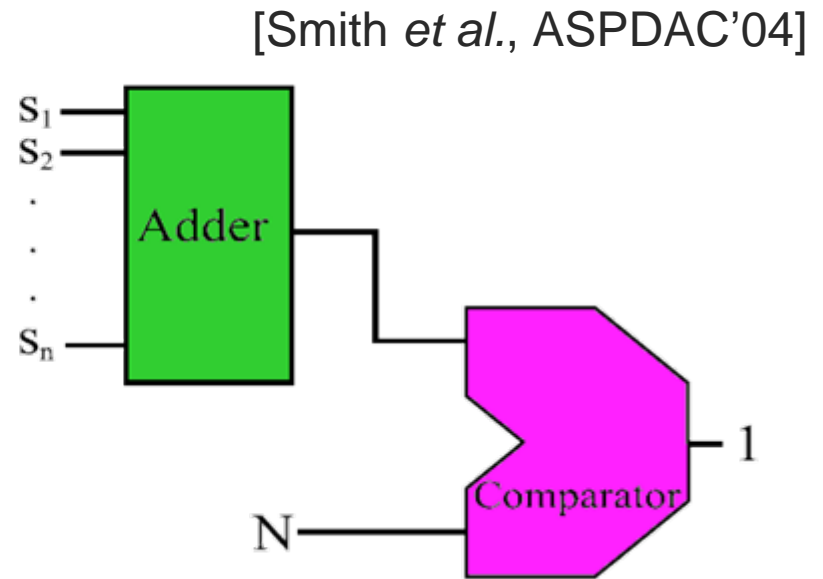
- Background – error diagnosis
- Problem formulation
- Common causes of unreachability
- Unreachability diagnosis technique
- Experimental results
- Conclusions

Background - Error Diagnosis

1. To model errors: insert MUXes into the circuit
2. To limit the number of allowed errors: use an adder and a comparator
3. Convert the circuit to CNF
4. Constrain inputs/outputs using input vectors/correct output responses



Error modeling



Error-cardinality constraint

[Problem Formulation]

- Given a testbench, a design, a list of liberated variables and the target unreachable code
- Find a set of variables that contribute to the unreachability
- Also provide suggested values to solve the problem

[Common Causes of Unreachability]

- Hardware bugs
 - Conflicting conditions
 - Obsolete code
- Design modalities
- Testbench errors
 - Over-constrained rules
- Reachability analysis limitation
 - Insufficient sequential depth

Modified Symbolic Simulation Algorithm for Unreachability Diagnosis

```
Procedure unreachability_diagnosis()
01. event = ...
02. curr = ...
03. while event != null
04.     whenever a liberated variable,  $v_i$ , is accessed,
           replace returned symbolic trace  $V_i$  with  $V_{i_S} ? V_{i\_free} : V_i$ ;
05.     if statement is a conditional block with condition cond
06.         curr_sym_cond  $\&=$  cond;
07.         do not execute statement if curr_sym_cond is proven to be 0;
08.         if code_to_be_diagnosed is reached
09.             unreachability_diagnosis(curr_sym_cond);
10.         else if statement is a non-conditional block with condition cond
11.             restore curr_sym_cond to its constraint;
12.         else if statement is a non-conditional block with condition cond
13.             never execute statement;
14.         even if statement is a non-conditional block with condition cond
15.     statement = statement.next;
```

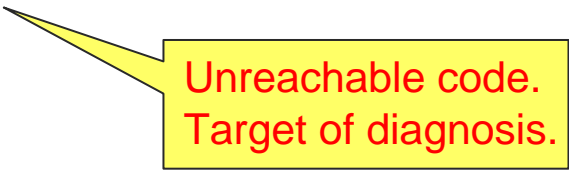
When liberated variables are accessed, a MUX is introduced so that the variable has the freedom free to take a new value.

This gives the symbolic simulator the freedom to explore previously-impossible paths.

Unreachability Diagnosis – Example

```
module example;
reg mode, clk;
reg [7:0] result, a, b;
always @(posedge clk) begin
    if (mode == 0)
        result= a + b;
    else
        result= a - b;
end
initial begin
    mode= 0;
    a= $random;
    b= $random;
end
```

1. Select liberated variables:
mode, a



Unreachable code.
Target of diagnosis.

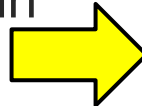
Unreachability Diagnosis – Example

```

module example;
reg mode, clk;
reg [7:0] result, a, b;
always @(posedge clk) begin
  if (mode == 0)
    result = a + b;
  else
    result = a - b;
end
initial begin
  mode = 0;
  a = $random;
  b = $random;
end

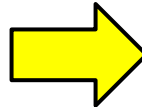
```

mode=0



mode = $mode_sel ?$
 $mode_free : 0$

result=a+b



result = (($mode_sel ? mode_free : 0$)
 $== 0$) ? ($a_sel ? a_free : 0$) + b :
 $(a_sel ? a_free : 0) - b$

1. Select liberated variables:
mode, a
2. Modify symbolic simulation
algorithm for MUX insertion
variables are

Unreachability Diagnosis – Example

```
module example;
reg mode, clk;
reg [7:0] result, a, b;
always @(posedge clk) begin
    if (mode == 0) ←
        result= a + b;
    else
        result= a - b;
end
initial begin
    mode= 0;
    a= $random;
    b= $random;
end
```

$mode = mode_sel ?$
 $mode_free : 0$

1. Select liberated variables:
mode, a
2. Modify symbolic simulation algorithm for MUX insertion when liberated variables are accessed
3. Since symbolic condition for entering the else branch is no longer false, symbolic simulation will execute the target code under condition:
 $((mode_sel ? mode_free : 0) \neq 0)$
4. Unreachability diagnosis is performed on the symbolic condition to identify the select signals to be asserted

Unreachability Diagnosis - Example

- Unreachability diagnosis is performed on $((mode_sel ? mode_free : 0) \neq 0)$
- A SAT solver is used to find a solution that can satisfy the condition
 - In this example, $mode_sel = 1$ and $mode_free = 1$
 - Signal mode contributes to the unreachability
 - If its value is 1, then the code can be reached
- Similar to traditional error diagnosis, cardinality constraints are necessary
 - To narrow down the problem

[Implementation Insights]

- Selection of liberated variables
 - Typically involves all design variables except clocks and resets
 - Hierarchical approach can be applied
 - Narrow down the problem to a few blocks first
 - Then look into the blocks
- Values returned for V_{free} are suggestions on how to fix the problem

Implementation Insights

- Shared select line/free variables can improve unreachability diagnosis performance
- For code that models hardware
 - Variables updated at the same cycle can share the same select and free symbols
 - At the RTL, each variable is typically updated only once at a clock
- For testbench code
 - Select lines typically cannot be shared because a variable may be updated many times at a time step

[Experimental Results]

- DLX design from Bug UnderGround project in University of Michigan
 - One of the few publicly available designs that contain non-trivial dead code
 - Design contains 40 bugs, but 6 of them can never be triggered
 - Goal: diagnose the causes of the 6 dead bugs
- Two industrial designs were also used
 - A block in a multimedia chip
 - A block in a high-speed I/O interface design

Diagnosis Example

- Bug description: if write to r7 is followed by ADD with rt=r7 write to r14 occurs

- Bug triggering code:

```
RDwire = ((IR4[`op]==`SW) && (IR4[rt]==5'd7) &&
(RDaddr5==5'd7) && (IR5[`op]==`ADD)) ?
5'd14 : RDaddr5;
```

- Diagnosis result:

```
Diagnosis: DUV.IR2, suggested
Variable DUV.IR2, at time 250
value= 32'b0000000000000000
```

```
Variable DUV.IR2, at time 305 (negedge clk, #5),
value= 32'b10000000000000000000000000000000;
```

ADD

```
Variable DUV.IR2, at time 405 (negedge clk, #5),
value= 32'b10101100000001110000000000000000;
```

SW

rt=7

ADD is an illegal OP code. Correct instruction: OP=`SPECIAL_OP with subtype = ADD

DLX Result – Full Chip

Case	Runtime	Memory (MB)	#Liberated variables	#Diagnosis
Bug20	3m20s	637.36	217	1
Bug22	1m22s	504.95	216	7
Bug29	8m49s	601.92	217	24
Bug31	14m45s	815.19	216	5
Bug33	28m39s	1146.15	218	1
Bug34	28m48s	1146.35	218	1
CaseA	29m17s	897.93	215	8
CaseB	29m13s	887.22	215	8

CaseA and CaseB are unreachable due to over-constrained testbenches. All other cases use properly-constrained testbenches and unreachability is due to design bugs.

DLX Result – Buggy Module

Case	Runtime	Memory (MB)	#Liberated variables	#Diagnosis
Bug20	45s	357.73	10	1
Bug22	43s	417.61	67	3
Bug29	4m52s	549.41	68	9
Bug31	8m1s	655	67	2
Bug33	13m26s	584.22	16	1
Bug34	13m24s	584.33	16	1
CaseA	7m14s	394.69	7	1
CaseB	7m31s	394.29	7	1

- Runtime is shorter and diagnosis is more accurate
- Hierarchical approach can be useful

Industrial Case Result

Case	Lines of RTL	Runtime	#Liberated variables	#Diagnosis
DesignA	5074	1m34s	236	36
DesignB	8068	32m5s	1520	15

- Diagnosis involving 2 variables were reported for DesignA
 - Without our diagnosis, 27730 combinations of variables need to be checked
 - With our diagnosis, 99.8% possible combinations can be eliminated
- Our diagnosis can narrow down the problem

Conclusions

- Unreachability diagnosis is challenging
 - No counterexamples exist for debugging
- A new symbolic simulation algorithm that can explore nonexistent execution paths
- Error diagnosis based on symbolic conditions can identify the cause of unreachability
- Experimental results show that our techniques can successfully narrow down the problem