# Just do it!

## Who cares if a Structural Analysis tool is using Formal Verification

Scott Aron Bloom  Chief Technical Officer of Blue Pearl Software

## Introduction

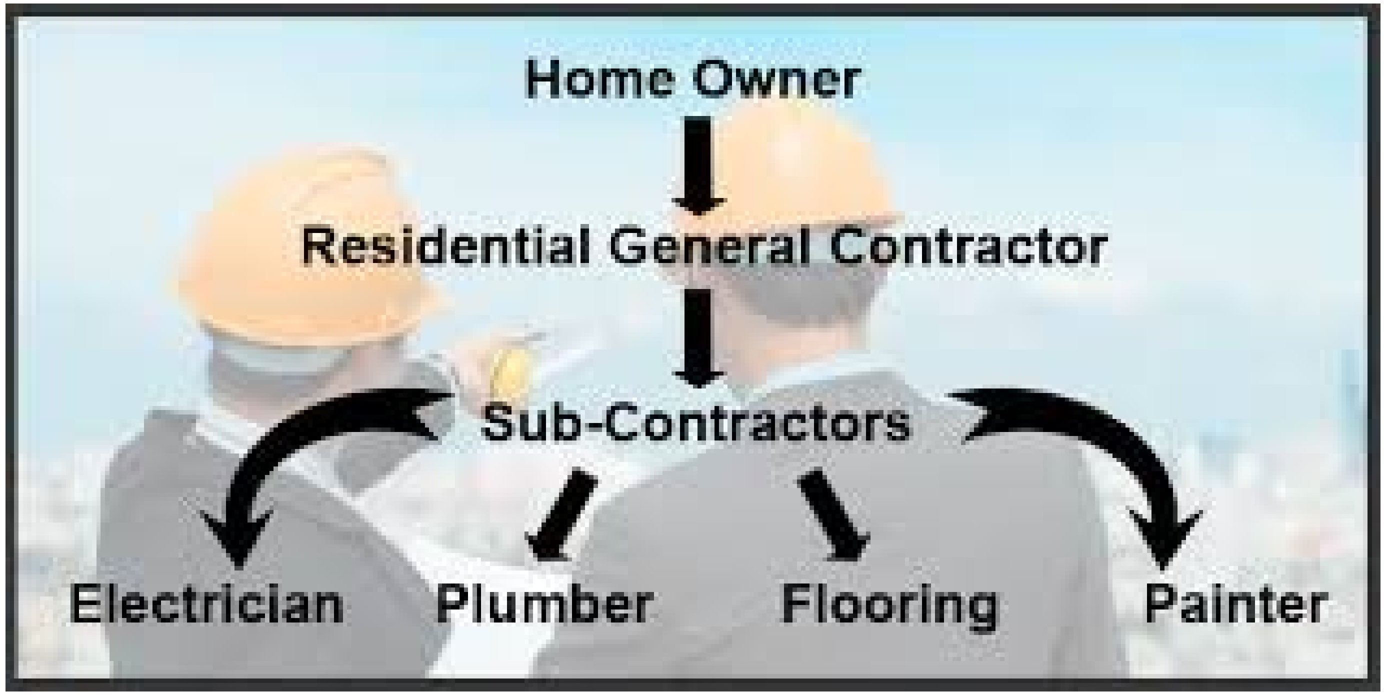
- **Scott Aron Bloom  Chief Technical Officer of Blue Pearl Software**
  - 20+ years of experience in software development
  - Founder and principal developer for OnShore Consulting Services specializing in EDA and Qt development.
  - Founder and VP of Engineering, at Stelar Tools, an EDA startup
  - Product development positions at AccelChip, Mentor Graphics, Exemplar Logic and Interconnectix
  - Email: scott.aron.bloom@bluepearlsoftware.com

## Agenda

- Premise for the talk
- What is Linting
- What is Formal Verification
- Formal Verification Tool vs Formal Verification Analysis in Linting
- Why does a linting tool use formal
- RTL Designers should be experts in their designs, not the tools
- Q/A & Conclusion

## Premise of the talk

- EDA – Electronic Design **Automation**



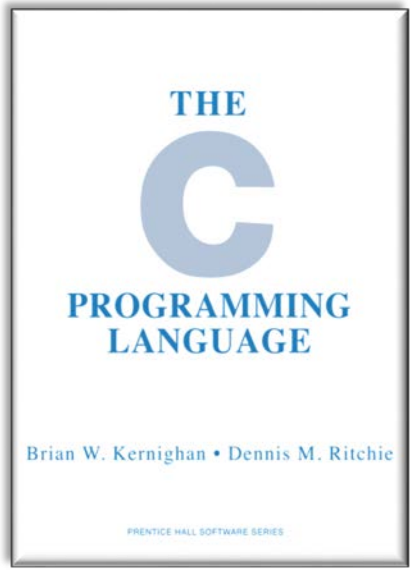
- Just do it! You need to be an expert in what you do not what EDA vendors provide

## Agenda

- Premise for the talk
- What is Linting
- What is Formal Verification
- Formal Verification Tool vs Formal Verification Analysis in Linting
- Why does a linting tool use formal
- RTL Designers should be experts in their designs, not the tools
- Q/A & Conclusion

## Linting History

- From “C” to Shining RTL….
  - “Lint”: Original name given to a program that flagged some suspicious and non-portable constructs (likely to be bugs) in C code
  - The term is now applied generically to tools that flag suspicious usage in software written in any computer language (C, VHDL, Verilog, Java, etc.)


THE
C
PROGRAMMING LANGUAGE
Brian W. Kernighan • Dennis M. Ritchie
PRENTICE HALL SOFTWARE SERIES

# Types of RTL Lint Checks

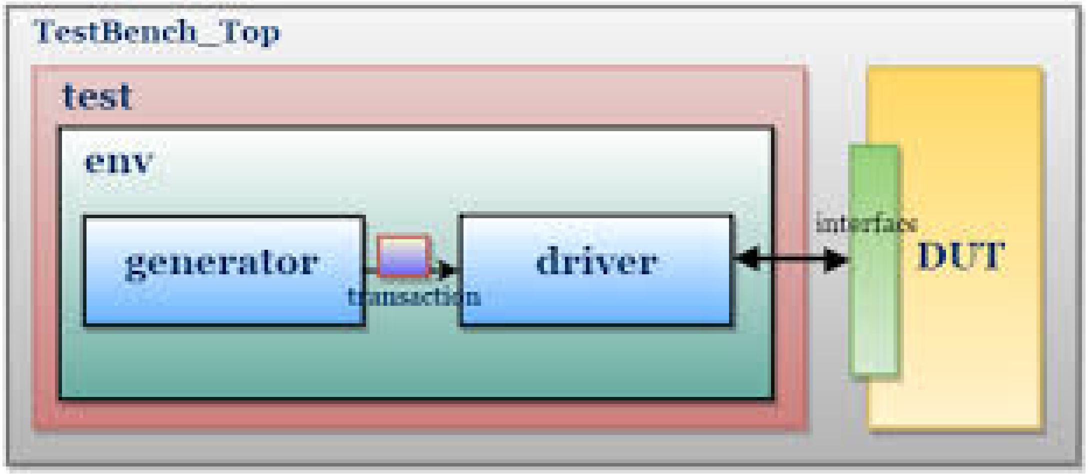
- Syntax checking
- Style checking
- Semantic checking

# What About Simulation / Synthesis

– Simulation is only as good as your testbench

– Synthesis assumes the code is good, and synthesizes it as such

– Lint doesn't find every problem with every design



# Simulation vs. Formal Verification

| | Simulation | Formal Verification |
|---|---|---|
| **Exhaustivity**<br>(measurement of the ability to thoroughly observe all possible input scenarios) | * Not possible to simulate all possible states in a design even with 100's of CPUs and months of simulation<br>* Focus on scenarios and assertions to break the design | * Explores all possible states<br>* Results in high reliability RTL<br>* Shifts focus on the correct functional behavior |
| **Controllability**<br>(measurement of the ability to activate, stimulate, or sensitize a specific point with the design) | * Must conceive vectors, scenarios to "adequately" simulate the design<br>* Likely to miss corner case scenarios | * No stimulus required<br>* Start early in the design cycle<br>* Exhaustive all corner cases issues found |
| **Observability**<br>(measurement of the ability to observe the effects of a specific, internal, stimulated point within the design) | * Must propagate bugs to output pins or insert logic assertions to expose and debug bugs | * Automatically isolates root cause of bugs<br>* Visualize incorrect behavior and fix it faster |

## Why Designers Use RTL Linting

- Its all about productivity!
- Faster and sooner than Simulation and/or Synthesis
  - “Verify as you Code”
  - Preforms DRC checks
  - Automates checking vs “Hand creating test benches to find issues”

## Agenda

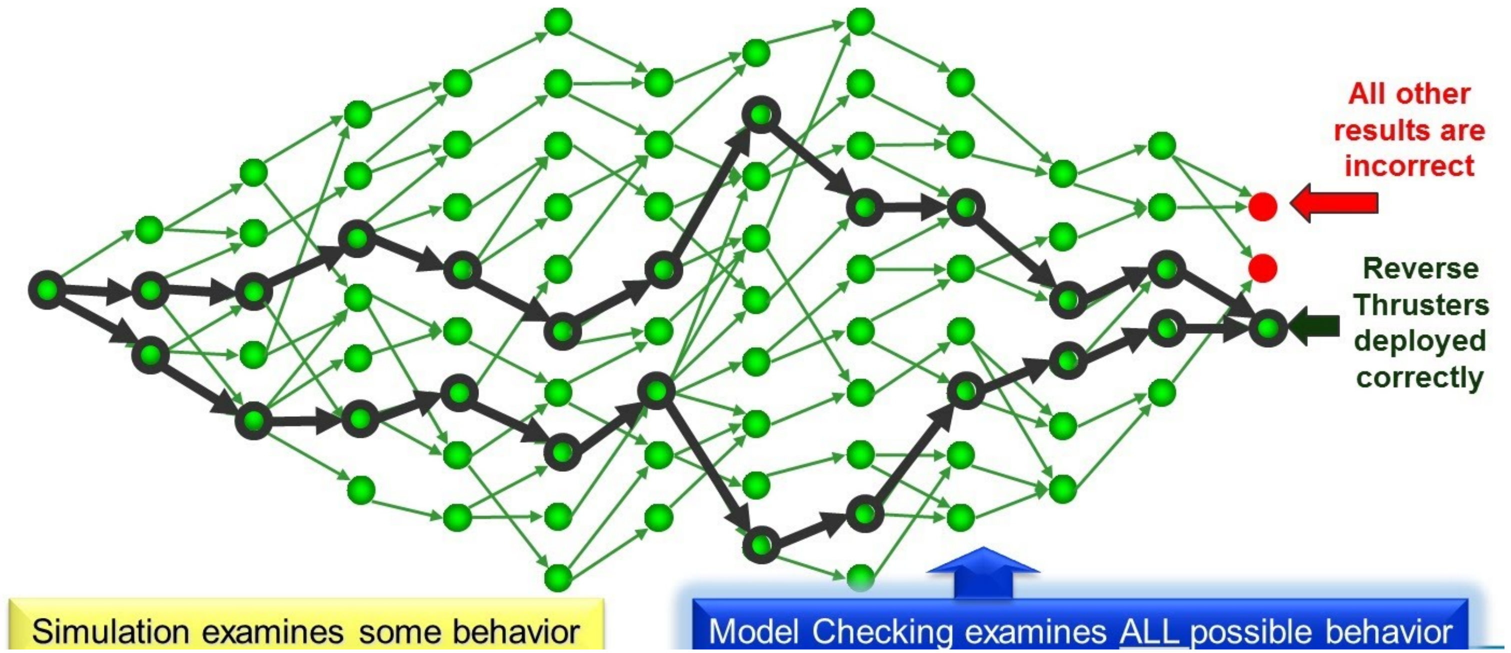
- Premise for the talk
- What is Linting?
- What is Formal Verification?
- Formal Verification Tool vs Formal Verification Analysis in Linting
- Why does a Linting tool use Formal Verification?
- RTL Designers should be experts in their designs, not the tools
- Q/A & Conclusion

## What is Formal Verification

- Formal Verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics.

## Uses of Formal Verification In EDA

- Equivalence checkers
- Property checkers
- Model checkers


All other results are incorrect
Reverse Thrusters deployed correctly
Simulation examines some behavior
Model Checking examines ALL possible behavior

## Agenda

- Premise for the talk
- What is Linting?
- What is Formal Verification?
- Formal Verification Tool vs Formal Verification Analysis in Linting
- Why does a Linting tool use Formal Verification?
- RTL Designers should be experts in their designs, not the tools
- Q/A & Conclusion

## FV Tool vs Algorithm in Lint

- Stuck at Zero
  - Analysis to pinpoint signals that never change values
  - Could be stuck at One as well

# FV Tool vs Algorithm in Lint

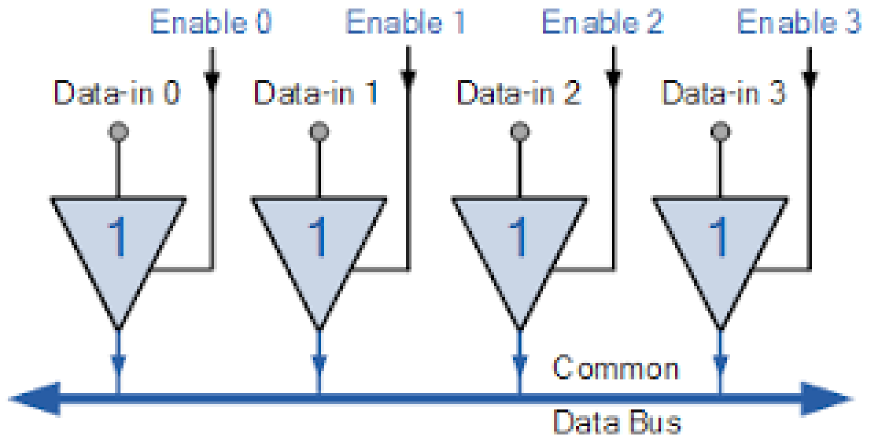
- Stuck at Zero
  - FV Tool requires the user to create an assertion on every register
  - Algorithm in Lint, the tool creates the assertion for every register

```
input [3:0] cmpValue; // range 0->15

if ( RST == 0 )
    output = 4’b0000;
else
begin
   if ( cmpValue == 16 ) //out of range never true
       output = inpValue;
   else
       ……
       tmp = ~inpValue
       ……
       output <= inpValue && tmp; // always zero
       ……
    end
end
```

# FV Tool vs Algorithm in Lint

- Potential multi-drive state on tristate signal
  - Analysis to pinpoint conditions where tri-state drivers are enabled at the same time
  - Also finds when a tri-state signal is never driven



# FV Tool vs Algorithm in Lint

- Multi-driven Tri-state
  - For every Tri-state signal, a FV Tool user would have to verify every driver is only active at one time
  - Algorithm in Lint, the tool creates assertion for every Tri-state signal and every driver automatically

```
begin // correct behavior, trivial
   if ( value )
       dout <= en ? i1 : 1’bz;
   else
       dout <= en ? i2 : 1’bz;
end
```

vs

```
begin // assignments spread throughout module
   ……
   cond1 <= inpValid;
   if ( cond1 )
       dout <= en ? i1 : 1’bz;
   ……
   cond2 <= inpValid;
   if ( cond2 )
       dout <= en ? i2 : 1’bz;
end
```

# Challenges of Formal Verification

- FV is based on creating a valid model to represent the question at hand

- Three answers are possible
  - Assertion Proven/Validated
  - Assertion Disproven
  - I Don't Know?
    What does this mean?

## Agenda

- Premise for the talk
- What is Linting?
- What is Formal Verification?
- Formal Verification Tool vs Formal Verification Analysis in Linting
- Why does a Linting tool use Formal Verification?
- RTL Designers should be experts in their designs, not the tools
- Q/A & Conclusion

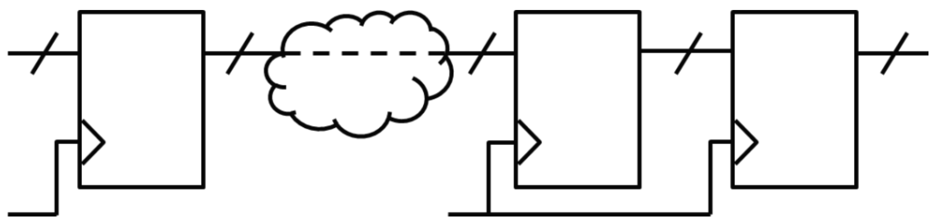
## Why Use Formal in Lint?

- Experts in FV (EDA tool developers) create focused models based on design + check being analyzed

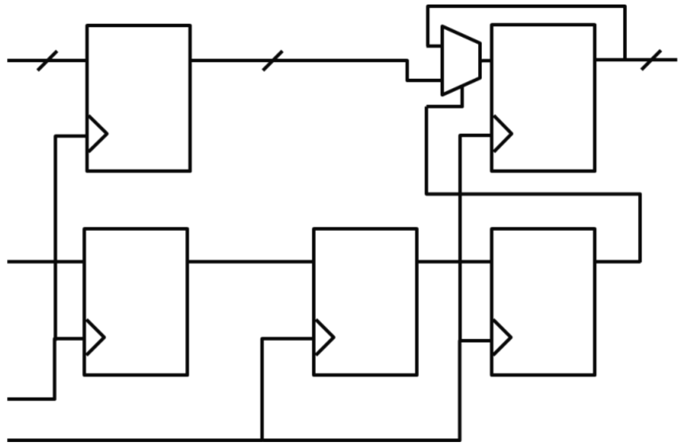
- Not perfect, however more productive than hand creating all the asserts in a separate FV tool
  - Don’t code in gates, code in RTL? Why, productivity
  - How many asserts are required to check 100 RTL checks on 1000 lines of RTL code?

## Why use Formal in Lint?

– Multi-bit CDC crossing, showing the potential vs reporting actual issues



Nonpreferred bus synchronization



Mux based bus synchronization

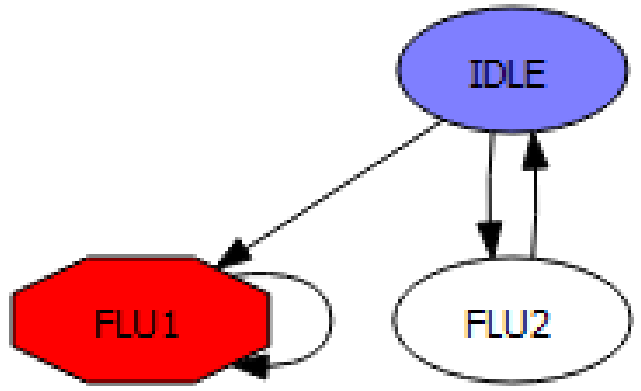
## To FV Or Not To FV

- Some checks are clearly structural only
  - No formal necessary, zero added value of formal

- Many checks can be implemented without formal
  - Less likely to have the “could not prove” outcome
  - May be sub-optimal on average

## FSM Example – Code not reachable

- Code not reachable
  - FSM case item + if statement conflict
  - Time consuming to implement assertions by hand
  - Automated FV assertion driven can be slow and inconclusive
  - Structural analysis can provide the same quality of results



# FSM Example – Code not reachable

```
reg [1:0] state; // 4 valid states
always @(posedge clock)
begin
    if ( reset )
        state <= 0;
    else
        case( state )  // synthesis fullcase
        0: if ( state == 0 ) // always true
               state <= 1;
           else
               state <= 2;  // never true
        1: if ( a && !a ) // never true
               state <= 2;
           else
               state <= 1; // code not reachable
        2: state <= 0;
        // no default or state 3 definition
        endcase
end
```

- Code not reachable
  - 4 states possible
  - All states have in and out transitions
  - Both Structural and FV can be used to determine Case 1 being a terminal state
  - Creating constraints/assertions would be time consuming for 4 states, let alone for a real FSM
  - Constraints are based on system knowledge that can change

## Is Formal Needed in Lint?

- Some checks MUST be done in formal
  - Forcing the user to become an expert in assertions, essentially forces the user to spend money on AEs

- However, for many checks, the best solution is Formal.
  - Using internally generated (not exposed) assertion models for many checks is the best use of an engineer’s time

## ROI of Verification Tools

- Primary expense for running lint should be the tool
  - NOT the maintaining of the test harness/inputs
  - NOT the maintaining of the “golden” vs “current” results on every design change
  - NOT the hiring of tool experts to run the tool
  - NOT the EDA AEs to help when stuck
- When you use FV Tools all these points fail

## Agenda

- Premise for the talk
- What is Linting?
- What is Formal Verification?
- Formal Verification Tool vs Formal Verification Analysis in Linting
- Why does a Linting tool use Formal Verification?
- RTL Designers should be experts in their designs, not the tools
- Q/A & Conclusion

# Forget About The Man in the Corner

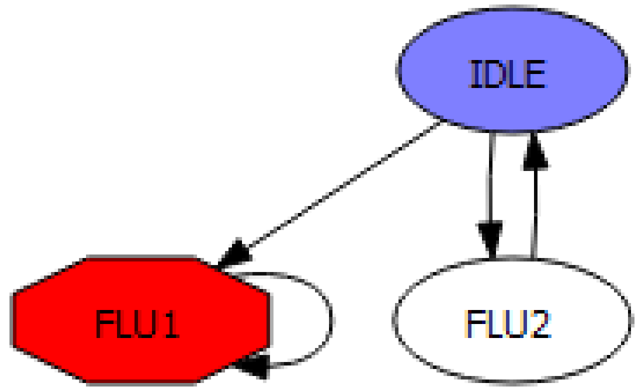
- Care about the checks not the implementation of the checks
- Many checks can be implemented using a variety of algorithms
- Reusing an algorithm previously targeting a stand alone tool, to drive an automatic verification process, is a big part of the A in EDA

# Algorithms Have Different Downsides

- We’ve discussed formal
- Pattern based analysis can fail to match the pattern
  - So “no” is sometimes a “could not check”
- Patterns often change with different design structures
  - One man’s synchronizer, is another man’s glue logic
- An EDA engineer’s job is to write algorithms that look at the check requested, analyze the design enough to choose the best tool to run that check on that design.
  - Not to always use a hammer to nail in a screw

# Example FSM Dead State Analysis

- A state is “dead” when no set of inputs can put the FSM into that state
- Often caused by structural issues.  State variable **never** gets assigned to the required value
- Sometimes caused by impossible conditions that if satisfied would lead to the state variable getting assigned properly



## Example FSM Dead State Analysis

```
reg [2:0] state; // 7 valid states
always @(posedge clock)
begin
    if ( reset )
        state <= 0;
    else
        case( state )  // synthesis fullcase
        0: state <= 1;
        1: state <= 2;
        2: state <= 4;
        3: state <= 4;
        4: state <= 0;
        default: state <= 0;
    endcase
end
```

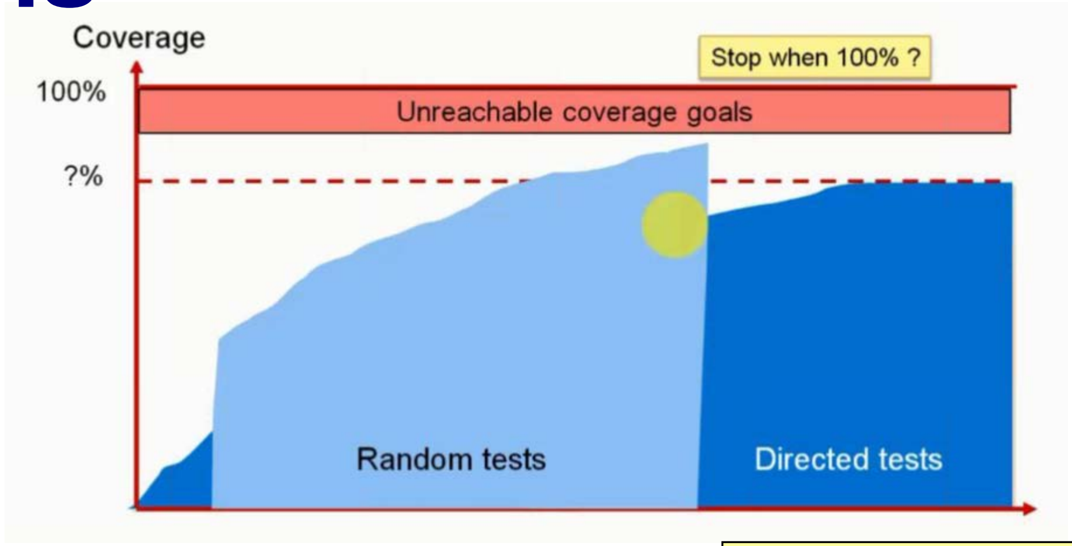
- Dead State
  - 7 states possible
  - State 2 “should” transition to 3
  - FV not necessary at all to find most common FSM issues

# Run the Check….

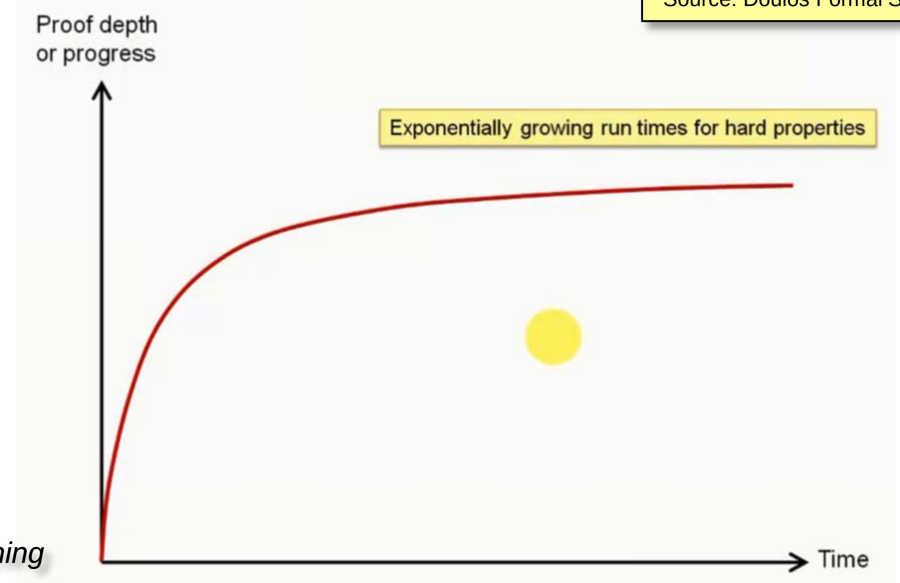
- For any type of bug in RTL, often multiple algorithms exist for finding the problem
- Why should the USER care which algorithm is applied to find the problem?
- Question should be: How long do you want to spend analyzing potential dead code?
- How long do you want to spend setting up the tool to run a given check?
- Small testcases, presented to prove the value of a check, are often contrived to show the issue, but in real life the check can take a very long time when run on a real-word design

# Diminishing Returns

- Longer runtimes don't necessarily lead to any significant amount new results
- Especially with FV
  - Tweaking conditions on a previous timeout may just delay the "could not prove", rather than bringing the proof to conclusion



Source: Doulos Formal Short Training



*Source Doulos Training*

# Focus On What is Important

- Style Analysis vs longer Verification
  - Many styles have been created to prevent bugs
  - From C/C++

```
if ( 1 == a ) vs if ( a == 1 )
```

- Many bugs are “typos”
- Can be found with simple linting style checks

## Example

- False Path caused by improper if conditions

```
if ( a == ’b010 )
   tmp1 <= in;
if ( a == ’b01 )
   tmp2 <= tmp1;
Out <= tmp2;
```

- “issue” can be reported/found in multiple ways
- REAL issue, use defined constants for all comparison values
  - Simple, fast, trivial automated analysis check.

## If The Code Has a Bug

- When a FP is not a bug, an advanced check is required to find the issue

```
if ( a == `STATE2 )
   tmp1 <= in;
if ( a == `STATE1 )
   tmp2 <= tmp1;
Out <= tmp2;
```

- Now, tool requires a “Find the False Path” check

## Let The Tool Chose …

- You determine the check you want
  - Tool should determine based on design criteria best approach to solve the check

- You don’t tell your General Contractor to hire (or when to hire) a plumber or carpenter
  - You tell him to redo your kitchen

## Let The Tool Chose …

- The “run a little longer” vs diminishing results loop
  - Is a total waste of time
  - After a while, just spinning your wheels, not improving your verification results
- Don’t insist on being a backseat verifier
- Don’t let perfection prevent good enough
- You don’t tell your contractor how long to work on the sink vs tile

## Agenda

- Premise for the talk
- What is Linting?
- What is Formal Verification?
- Formal Verification Tool vs Formal Verification Analysis in Linting
- Why does a Linting tool use Formal Verification?
- RTL Designers should be experts in their designs, not the tools
- Q/A & Conclusion

# Conclusions and Summary

- Infinite resources, time and money, run formal on every possible condition on every possible line of code, with infinite runtime to solve each assertion
- Real world requires compromise
- The mythical verification model, of one line of Verification Code for every line of design code, is rarely if every achieved by any design team.
  - Budgets, time, hardware resources to implement verification
- For more information, please visit http://www.bluepearlsoftware.com or email me at scott.aron.bloom@bluepearlsoftware.com