in systemverilkog formal verification
Here almost all the properties am get
may be the real cause of bounded pa

Sources

SystemVerilog Assertions and Formal Verification - Course
ucsc-extension • 1

Formal Verification | Sie
verificationacademy

ervic

Answer

In SystemVerilog formal verification of a chip design, getting "bounded pass" or inconclusive results for most properties can occur due to several reasons:

1. **State Space Explosion**: As the design complexity increases, the state space that needs to be explored by the formal tool grows exponentially. This can lead to the formal tool timing out or running out of memory before fully verifying the properties, resulting in bounded passes. [1]
2. **Incomplete Constraint Specification**: If the constraints or assumptions provided to the formal tool are incomplete or inaccurate, it may not have enough information to fully verify the properties, leading to bounded passes. [1]
3. **Complex Properties**: Overly complex properties or those involving deep sequential behavior can be challenging for formal tools to verify exhaustively, resulting in bounded passes. [1]
4. **Abstraction Modeling**: Inadequate or incorrect abstraction modeling of the design or environment can limit the formal tool's ability to reason about the properties, causing bounded passes. [1] [2]
5. **Tool Limitations**: Formal verification tools may have inherent limitations in handling certain types of designs or properties, leading to bounded passes in some cases. [2]

To address bounded passes and improve formal verification results, you can try the following:

1. **Simplify Properties**: Break down complex properties into simpler ones or use assume-guarantee reasoning to make them more tractable for formal tools. [1]
2. **Refine Constraints**: Review and refine the constraints and assumptions provided to the formal tool to ensure they accurately capture the design's behavior and environment. [1] [2]
3. **Improve Abstraction Models**: Develop more accurate and comprehensive abstraction models for the design and its environment to aid the formal tool's reasoning. [2]
4. **Increase Resources**: Allocate more computational resources (CPU, memory) to the formal tool, allowing it to explore a larger state space before timing out. [1]
5. **Use Complementary Techniques**: Combine formal verification with other verification techniques like simulation or emulation to achieve better overall coverage. [2]
6. **Consult Tool Experts**: Seek guidance from formal verification tool experts or vendors to understand tool-specific limitations and best practices for your design. [2]

It's important to note that bounded passes do not necessarily mean the design is incorrect; they indicate that the formal tool could not fully verify the properties within the given constraints and resources. Addressing the underlying causes can help improve formal verification results and increase confidence in the design's correctness. [1] [2]

Sources

Latest SystemVerilog topics - Verification Academy
verificationacademy · 1

A Gentle Introduction to Formal Verification - systemverilog.io
systemverilog · 2

A Blueprint for Formal Ver
systemverilog · 3

Answer

Here is an example of a bounded pass in SystemVerilog formal verification:

```verilog
property req_gnt_prop;
    @(posedge clk) req |-> ##[1:5] gnt;
endproperty

req_gnt_assert: assert property (req_gnt_prop);
```

This property states that whenever the `req` signal is asserted, the `gnt` signal must be asserted within 1 to 5 clock cycles after that.

If the formal tool reports a "bounded pass" for this property, it means the tool was able to prove the property holds true up to a certain depth (number of clock cycles), but could not conclusively prove it for all possible cycles. [2]

For example, the tool may report:

```
%%% Proof Status for property req_gnt_prop: Bounded Proof @ 12
```

This indicates the property was proven to hold for up to 12 clock cycles, but the tool could not complete the proof for cycles beyond that due to state space explosion or other resource constraints. [1] [2]

A bounded pass is an inconclusive result - it does not guarantee the property will never be violated, only that no violation was found up to the reported bound. To improve the verification confidence, you may need to adjust constraints, use abstraction techniques, or allocate more resources to allow deeper proofs. [1] [2]

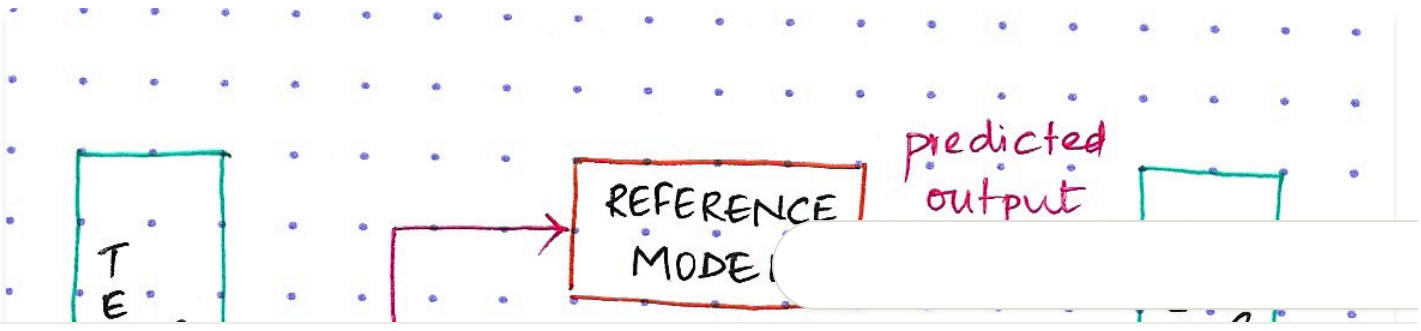
Share Rewrite

Keep exploring

how does bounded proof analysis work in formal verification +

what are the benefits of using bounded pass in formal verification +

can you provide a real-world example of bounded pass in a verification project +



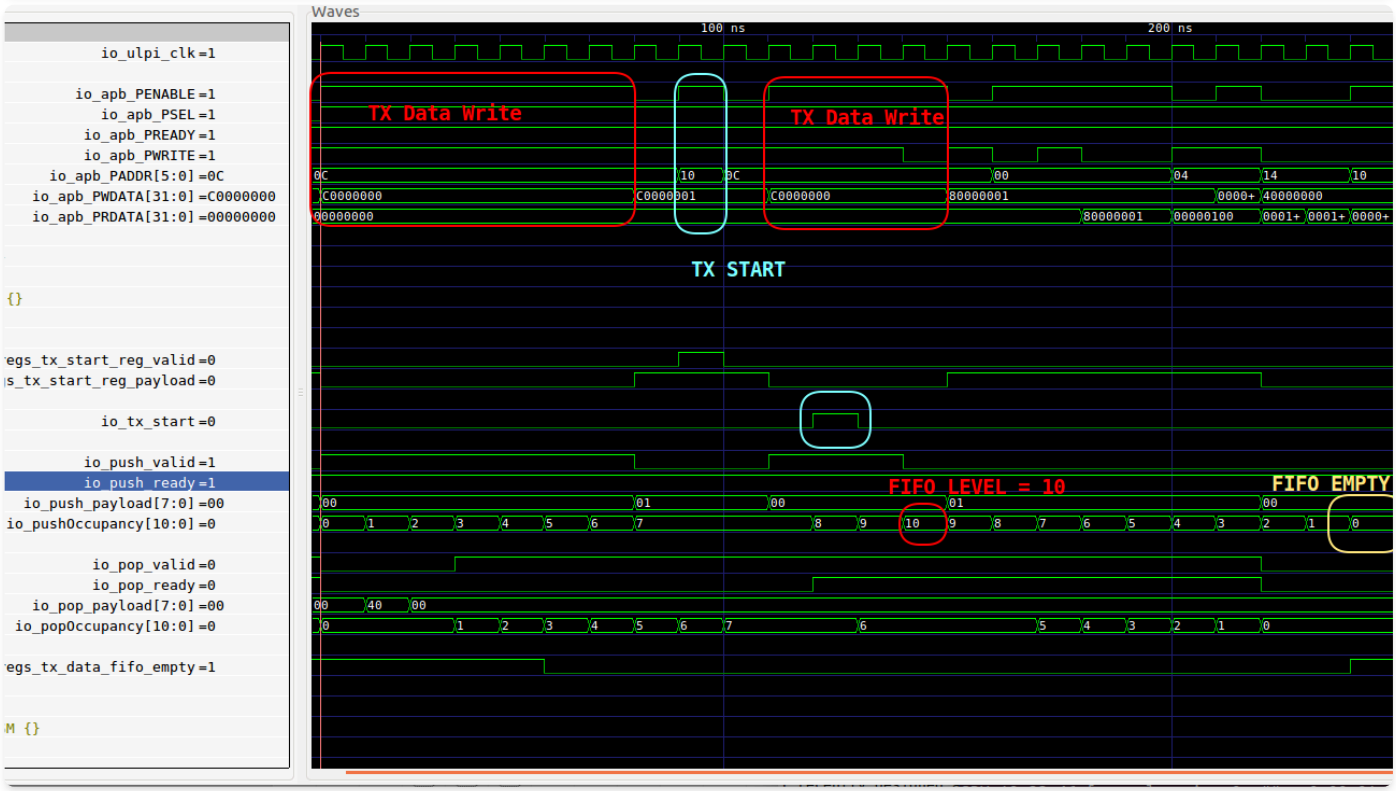
Home Discover Library

```
module counter (output logic[2:0] count_out,
          input  logic[2:0] data_in, // data to load
          input  logic ld_enb, count_enb, rst_n, clk);
  logic [2:0]  count;   logic  tc;
... // DUT design
endmodule // counter
```

```
module counter_props (
  input logic[2:0] count_out,
  input  logic ld_enb, rst_n,
  input  logic [2:0] count,
  input  logic tc,
  input  logic clk );
... // assertions
endmodule : counter_props
```

```
module counter_tb;
  logic  ld_enb=0;   logic [2:0]  data_in;
  wire  [2:0]  count_out;   logic  count_enb=0;
  logic  clk=0;   logic  rst_n=1;
 initial forever clk = ~clk;
counter
 DUT  (
    .ld_enb (ld_enb ) ,
   .data_in (data_in ) ,
   .count_out (count_out ) ,
   .count_enb (count_enb ) ,
   .clk (clk ) ,
   .rst_n (rst_n ) );

 bind counter counter_props counter_props_1(
       .count_out(count),
       .ld_enb(ld_enb),
       .rst_n(rst_n),
       .count(count),
       .clk(clk),
       .tc(tc));
  ... // testbench
endmodule : counter_tb
```




```
typedef enum {IDLE, RUN, ...} cmd_t;
class Packet;
```