Deadlock Verification For Dummies – The Easy Way of Using SVA and Formal

Mark Eslinger
Jeremy Levitt
Joe Hupcey III
Mentor, A Siemens Business, Fremont, CA

Abstract

RTL simulation cannot directly tell if a digital system is deadlocked — you can only observe that nothing has happened for a long time, and this is highly dependent on the “right” stimulus being applied.

In contrast, formal verification has the ability to find deadlock conditions in your design. However, the traditional iterative approach using written liveness and safety properties in combination with manually written constraints can be time consuming and error prone even in expert hands. While there are nonstandard assertion languages that can be used, these are reserved for academic practitioners and not useful for the typical RTL-aware design and verification engineers who use industry standard System Verilog Assertions (SVA).

In this paper we will show how combining the above concepts using normal SVA liveness properties allows for RTL engineers to achieve the benefit of formal deadlock analysis without the iterative component or learning a non-standard assertion language. Deadlock verification for dummies!

New Approach: Combine CTL/LTL Results

- **CTL Analysis**
  - Infer CTL property from SVA description
  - No need for engineers to write CTL properties
  - Directly target “real” deadlock situations
  - Enabled by new & improved formal engines

- **LTL Analysis**
  - Continuity with existing tool behavior
  - Leverage CTL analysis to expose escape routes

Results shown in easy to understand format

LTL deadlock counterexample

**Solution: Simultaneously Leverage Case (B)**

- If Case (A) is proven, case (B) counterexample (CEX) will be shown with escape routes
- There must be an escape route; otherwise CEX would be a case (A) CEX
- Add constraints on the escape routes
  - Example: if reset used to escape, constrain reset to not assert after design initialization (normally done automatically)
  - Poster FSM Example: Constraining “idle” to eventually equal the value of 2
  - For a complete proof, iterate thru escape routes and add new constraints until:
    - There are no more type-B CEX – congrats, your system is deadlock-free!
    - OR
    - There IS a type-A CEX – meaning there is deadlock situation, and you have a CEX from formal analysis to debug/correct it

Limitations of Simulation

System deadlock is virtually impossible to detect with RTL VHDL or Verilog simulations!

- Simulation cannot directly detect if the design is deadlocked
- Can only observe that nothing’s happened for a long time
- How long is too long?
- Simulation cannot differentiate between cases A and B
  - True system lookup vs. potentially poor stimulus
- Simulation is dependent on users generating the “right” stimulus
  - This is of course how all bugs are missed with simulation, but particularly so for bugs that require a number of specific, synchronized interactions

Summary

- The risk of a design going into deadlock is nearly impossible to detect with RTL simulation; hard to do with traditional formal
- Combining “LTL” and “CTL” analysis results, leveraging standard SVA syntax, and using new & more powerful RTL engines, enables regular engineers to effectively utilize CTL analysis
- Detecting RTL deadlock is now easier with Mentor’s PropCheck using these advanced algorithms under-the-hood