Agenda

Introduction: SystemVerilog Motivation
Vassilios Gerousis, Infineon Technologies
Accellera Technical Committee Chair

Session 1: SystemVerilog for Design

- **Language Tutorial** — Johny Srouji, Intel
- **User Experience** — Matt Maidment, Intel

Session 2: SystemVerilog for Verification

- **Language Tutorial** — Tom Fitzpatrick, Synopsys
- **User Experience** — Faisal Haque, Verification Central

Lunch: 12:15 – 1:00pm

Session 3: SystemVerilog Assertions

- **Language Tutorial** — Bassam Tabbara, Novas Software
- **Technology and User Experience** — Alon Flaisher, Intel
- **Using SystemVerilog Assertions and Testbench Together** — Jon Michelson, Verification Central

Session 4: SystemVerilog APIs
Doug Warmke, Model Technology

Session 5: SystemVerilog Momentum

- **Verilog2001 to SystemVerilog** — Stuart Sutherland, Sutherland HDL
- **SystemVerilog Industry Support** — Vassilios Gerousis, Infineon

End: 5:00pm

SVA Technology and User Experience

Alon Flaisher

Manager of Formal Property Verification

Mobile Platforms Group

Intel

Outline

- Assertion Use at Intel
  - RTL assertions in Intel® Centrino™ mobile technology
  - Sequential RTL assertions
- Assertion Usage Examples
  - Bus examples
  - Front-end examples
- Assertion Technology
  - Compilation to formal verification
  - Compilation to simulation

RTL Assertions in Intel® Centrino™ Mobile Technology

- RTL assertions have been used in Intel for over a decade
  - Utilized by a variety of tools
- Basic combinational assertions
  - Most are either *forbidden* or *mutex*
  - The RTL includes thousands of assertions

Impact of RTL Assertions

- RTL assertions caught >25% of all bugs!
  – Assertions were very effective in bug hunting (>25%) in the cluster test environment (CTE)
  – Second after designated cluster checkers (>50%)
  – Assertions were the most effective in bug hunting (>27%) in the full chip environment
- Assertions were the first to fail
  – They were more local than checkers
  – They were mostly combinatorial
- RTL assertions shortened the debug process
  – Assertions point directly at the bug

Sequential RTL Assertions

- SVA sequential assertions increase the ability to capture design intent
  - Identified as a missing capability in Banias design
- Assertions are used to capture:
  - Assumptions on the interface
  - Expected output
  - Local relations
- Identical assertions are used for Simulation and Formal Verification
  - In FV every property can also be used as an assumption
  - Supports full assume-guarantee paradigm

Template Library

- Consists of dozens of temporal and combinatorial properties
  - Both safety and liveness properties
  - Targeted mainly for RTL designers
- It is easier for designers to write assertions using the template library
  - No need to ramp-up on a formal specification language
  - Hides the subtle implementation details
- Validators use the template library as a “best known method”

Bus Examples

Forbid Consecutive Address Strobes

```
property no_two_ads;
  @(posedge busclk)
        disable iff (rst) not (ADSOut [*2]);
endproperty

assert property (no_two_ads);
```

Alternatively:

```
@(posedge busclk)
      disable iff (rst) (ADSOut |=> !ADSOut);
```

Hold Request Until Ownership

```
sequence breq_low_b4_ownership;
  !owner[*0:$] ##1 !breq
endsequence

property hold_breq_until_ownership;
  @(posedge busclk) disable iff (rst)
  $rose(breq) |=> not breq_low_b4_ownership;
endproperty

assert property (hold_breq_until_ownership);
```

Multi Cycle Path

```
property mcp (sample, sig, clk);
   @(posedge clk) disable iff (rst)
              sample |-> $stable(sig);
endproperty


assert property (mcp(seg, earlygnt, sclk));
```

Template example

Front End Examples

FSM Related Assertions

- An FSM must only transition as follows:
  - (*Detect; Loading; Detect*), or
  - (*Detect ; Loading; Delivery; Detect*)
- All signals are stable between clock ticks
- An FSM is never stuck in *loading*

```
property not_stuck_in_loading;
  @(posedge clk) disable iff (rst)
        true[*0:$] ##1 state!=LOADING;
endproperty
assert property (not_stuck_in_loading);
```

FV Flow Compilation

Useful Library Element

- Following an enabler, signal s1 must be asserted strictly before signal s2:

```
property before (enabler, s1, s2, clock, reset);
  @(clock) disable iff (reset)
      enabler |=> not (!s1[*0:$] ##1 s2)
endproperty
```

Usage Example

- When a transaction starts, ads1 should appear strictly before ads2

```
assert property (
  before ($rose(SOT), ads1, ads2, posedge clk, rst));
```

How to Build an FV Checker

- We build a checker that looks for the counter-example of the property
- We therefore need to look at the negation of the property
- Ignoring clocks and resets, the counter-example of ads1 then ads2 is:

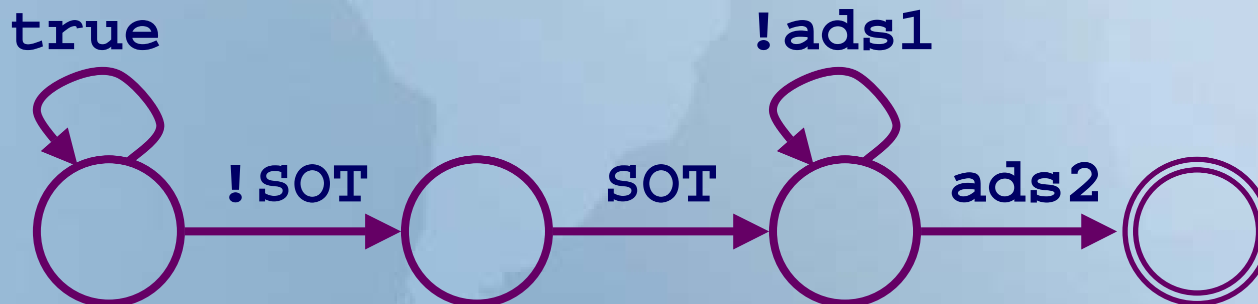
```
true[*0:$] ##1 !SOT ##1 SOT ##1 !ads1[*0:$] ##1 ads2
```

Non-Deterministic Automaton

- Ignoring clocks and resets, the non-deterministic automaton that implements

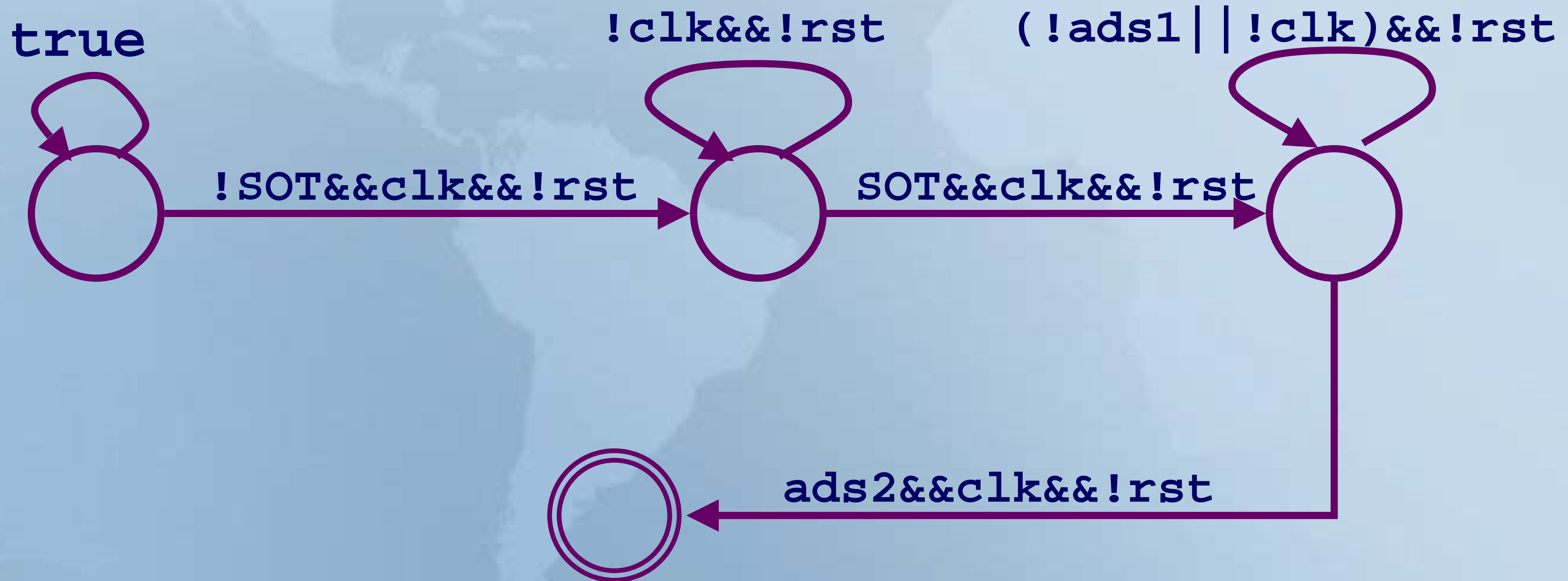
  `true[*0:$] ##1 !SOT ##1 SOT ##1 !ads1[*0:$] ##1 ads2`

  is the following:

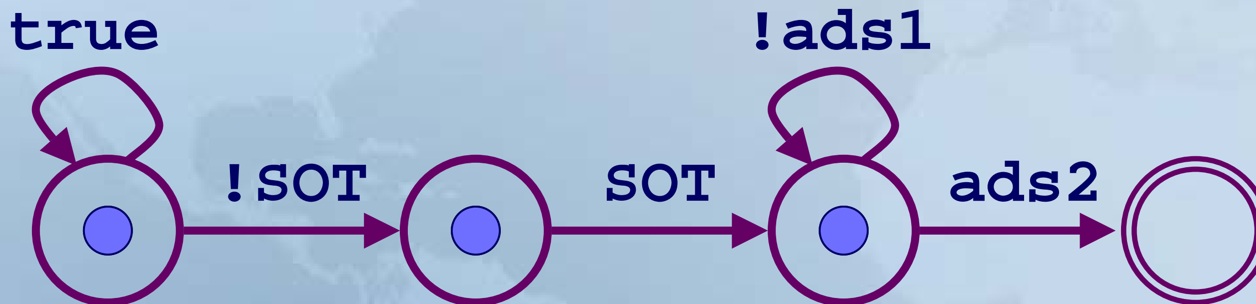


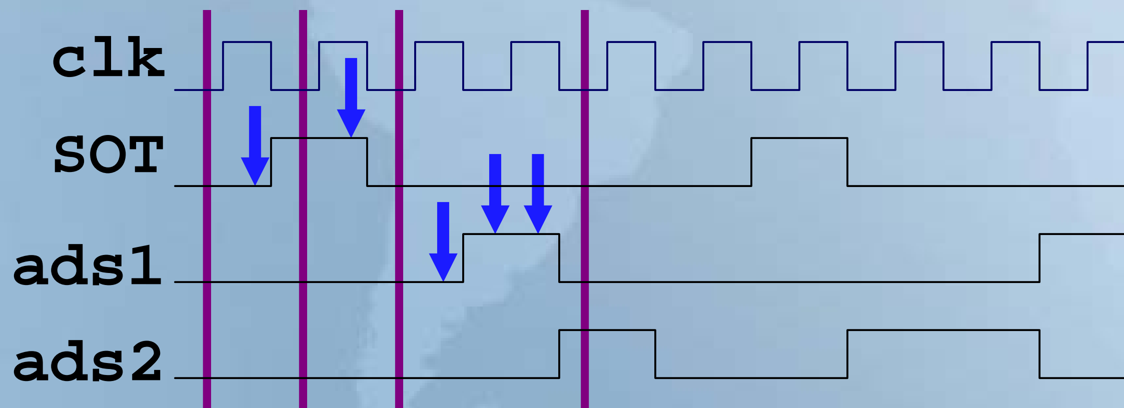
Adding Clocks and Resets

- The complete non-deterministic automaton that implements the checker is :

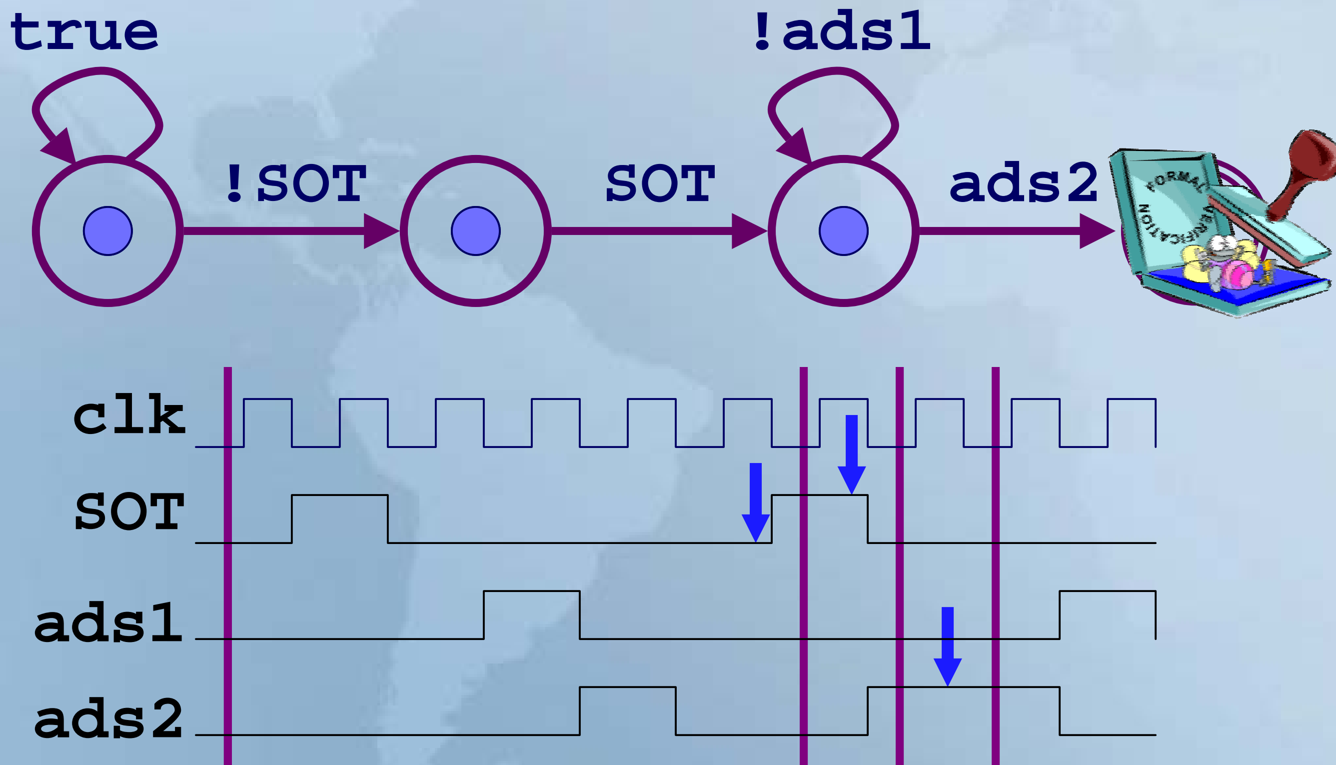


Non-Deterministic Run (1st choice)



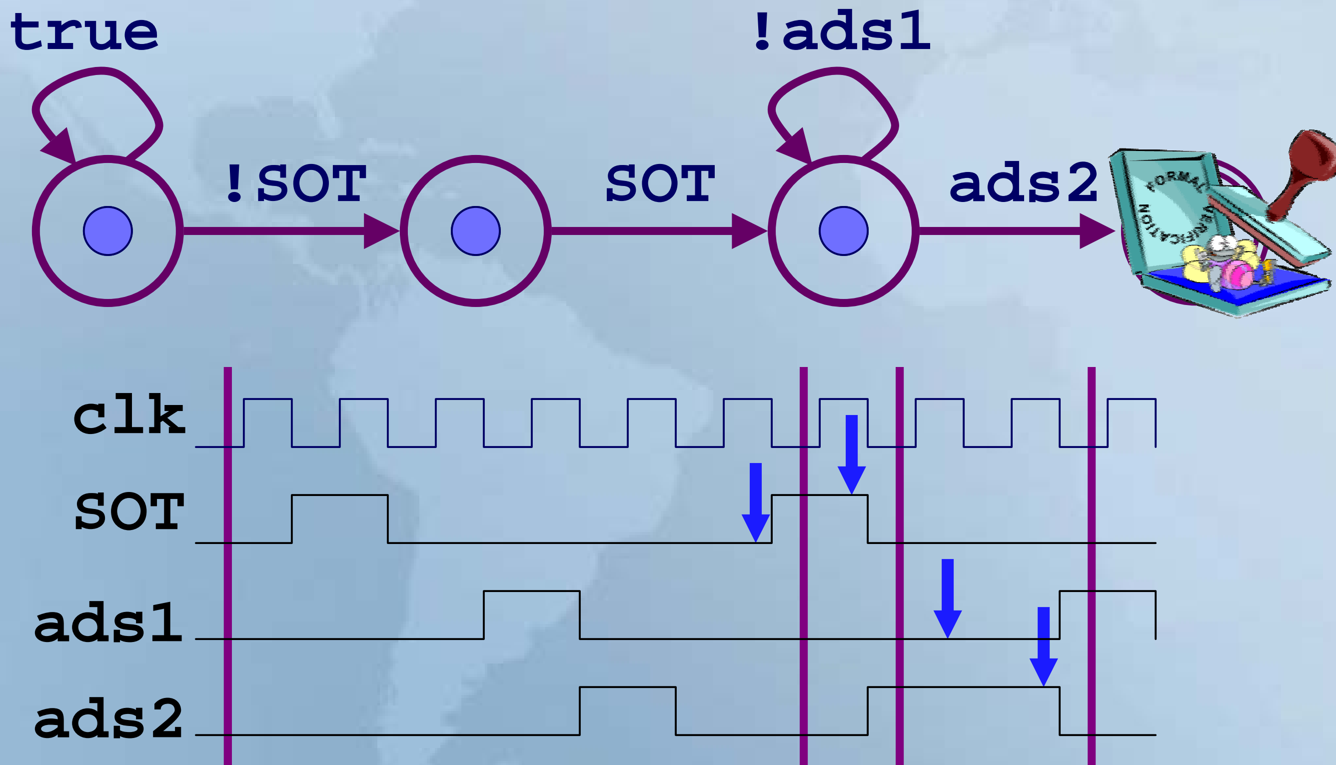


Non-Deterministic Run (2nd choice)



accellera

Non-Deterministic Run (3rd choice)



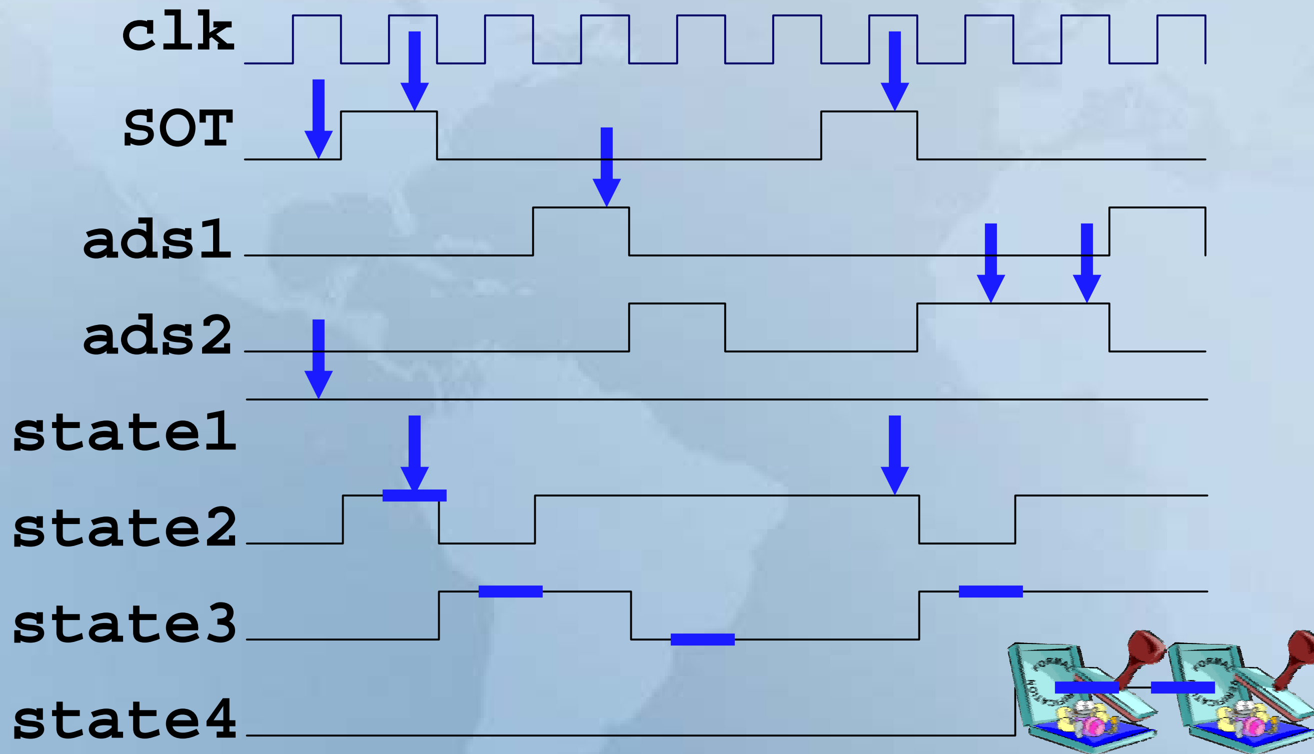
accellera

Simulation Flow Compilation

Determinization of the Checker

```
always @(posedge clk or rst)
   if (rst) then begin
       state1 <= 1;  state2 <= 0;
       state3 <= 0;  state4 <= 0;
   end
   else begin
       state1 <= 1;
       state2 <= state1 && !SOT;
       state3 <= state2 && SOT  ||  state3 && !ads1;
       state4 <= state3 && ads2;
       if (state4)  then $display("ads1_then_ads2");
     end
```

Deterministic Simulation




```
state2 <= state1 && !SOT;
state3 <= state2 && SOT   ||  state3 && !ads1;
state4 <= state3 && ads2;
```


accellera

Summary

- RTL assertions are a powerful validation tool
  - Have been used in Intel for over a decade
  - Sequential assertions improve the ability to capture design intent
- Assertions allow RTL designers to get involved in verification
  - Template Library simplifies use-model
- Same assertions used for FV and Simulation
  - Non-deterministic automatons are used in FV
  - Deterministic checkers are used for DV
- SVA capability is proven to work at Intel