How to Use Formal Analysis to Prevent Deadlocks

Abdelouahab Ayari, Mentor, a Siemens Business
Mark Eslinger , Mentor, a Siemens Business
Joe Hupcey III, Mentor, A Siemens Business

accellera SYSTEMS INITIATIVE

Mentor® A Siemens Business

2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

- Deadlock cases to consider
- Traditional approach to deadlock verification
- Challenges with traditional approach to deadlock verification
- Mentor’s enhanced deadlock verification solution
- Application Cases For Deadlock Analysis
- Summary

Two Deadlock Cases to Consider

- The most difficult bugs to find in designs are deadlocks
  - But they are also the most critical!

- **Type A** : Can your design get into a state from which it can never escape?

Two Deadlock Cases to Consider

- The most difficult bugs to find in designs are deadlocks
  - But they are also the most critical!

- **Type A** : Can your design get into a state from which it can never escape?

- **Type B** : Can your design get into a state from which you can stay as long as you like (by avoiding opportunities to escape)?

Two Deadlock Cases to Consider

Type A deadlock
"Real Deadlock"

```
always_ff @(posedge clk or negedge rstn)
  if (~rstn) st <= IDLE;
  else
  begin
    case (st)
      IDLE:   st <= STATE1;
      STATE1: st <= STATE2;
      STATE2: st <= sel ? STATE3 : STATE2;
      STATE3: st <= sel ? STATE3 : STATE3;
    endcase
  end
```

Type B deadlock
"Maybe-escapable Deadlock"

Two Deadlock Cases to Consider

Type A deadlock
"Real Deadlock"

```
always_ff @(posedge clk or negedge rstn)
  if (~rstn) st <= IDLE;
  else
  begin
    case (st)
      IDLE:   st <= STATE1;
      STATE1: st <= STATE2;
      STATE2: st <= sel ? STATE3 : STATE2;
      STATE3: st <= sel ? STATE3 : STATE3;
    endcase
  end
```

Type B deadlock
"Maybe-escapable Deadlock"

How is deadlock analysis done so far ?

Using Simulation For Deadlock Analysis

- **Checker Implementation**
  - Often make use of watchdogs
    - FSM does not stay in state S for more than N cycles
    - Wait no more than M cycles for an acknowledge
- **Drawbacks**
  - How to recognize that a design is in deadlock state?
    - Can only observe that nothing has happened (no progress) for a long time
    - How long is too long?
  - How to differentiate between deadlock types A & B ?
    - True system lockup vs. potentially poor stimulus
  - How to generate “right” stimulus to check deadlock situation?
    - Simulation is incomplete anyway
    - Writing/Generating stimuli for deadlock requires a number of specific, synchronized interactions

Using Simulation For Deadlock Analysis

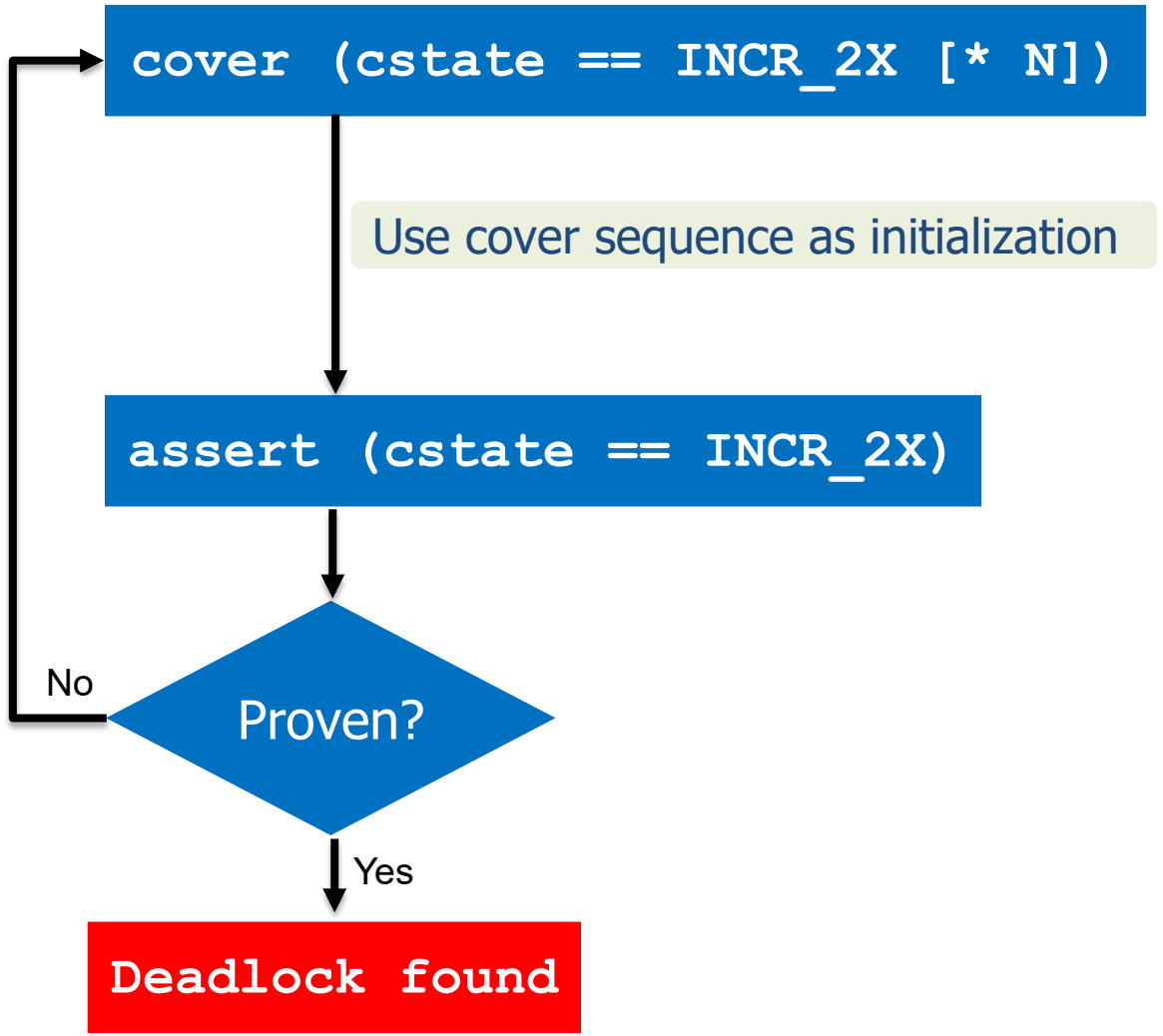
- **Checker Implementation**
  - Often make use of watchdogs
    - FSM does not stay in state S for more than N cycles
    - Wait no more than M cycles for an acknowledge
- **Drawbacks**
  - How to recognize that a design is in deadlock state?
    - Can only observe that nothing has happened (no progress) for a long time
    - How long is too long?
  - How to differentiate
    - T
  - How t
    - Si
    - Writing/Generating stimuli for deadlock requires a number of specific, synchronized interactions

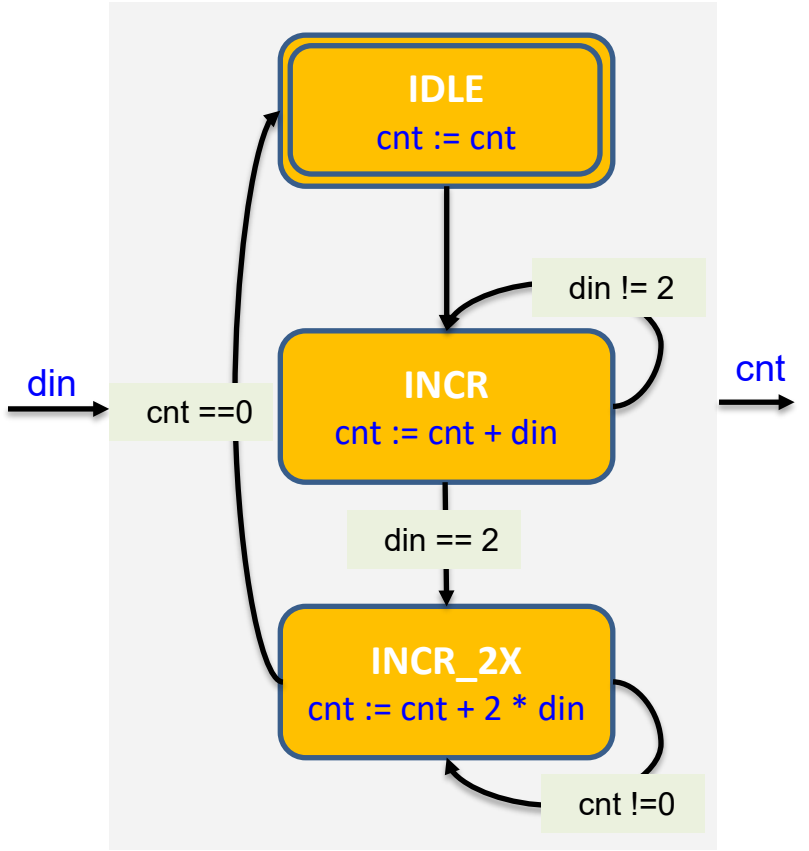
Simulation is incomplete ☹

Deadlock verification is a time consuming task ☹

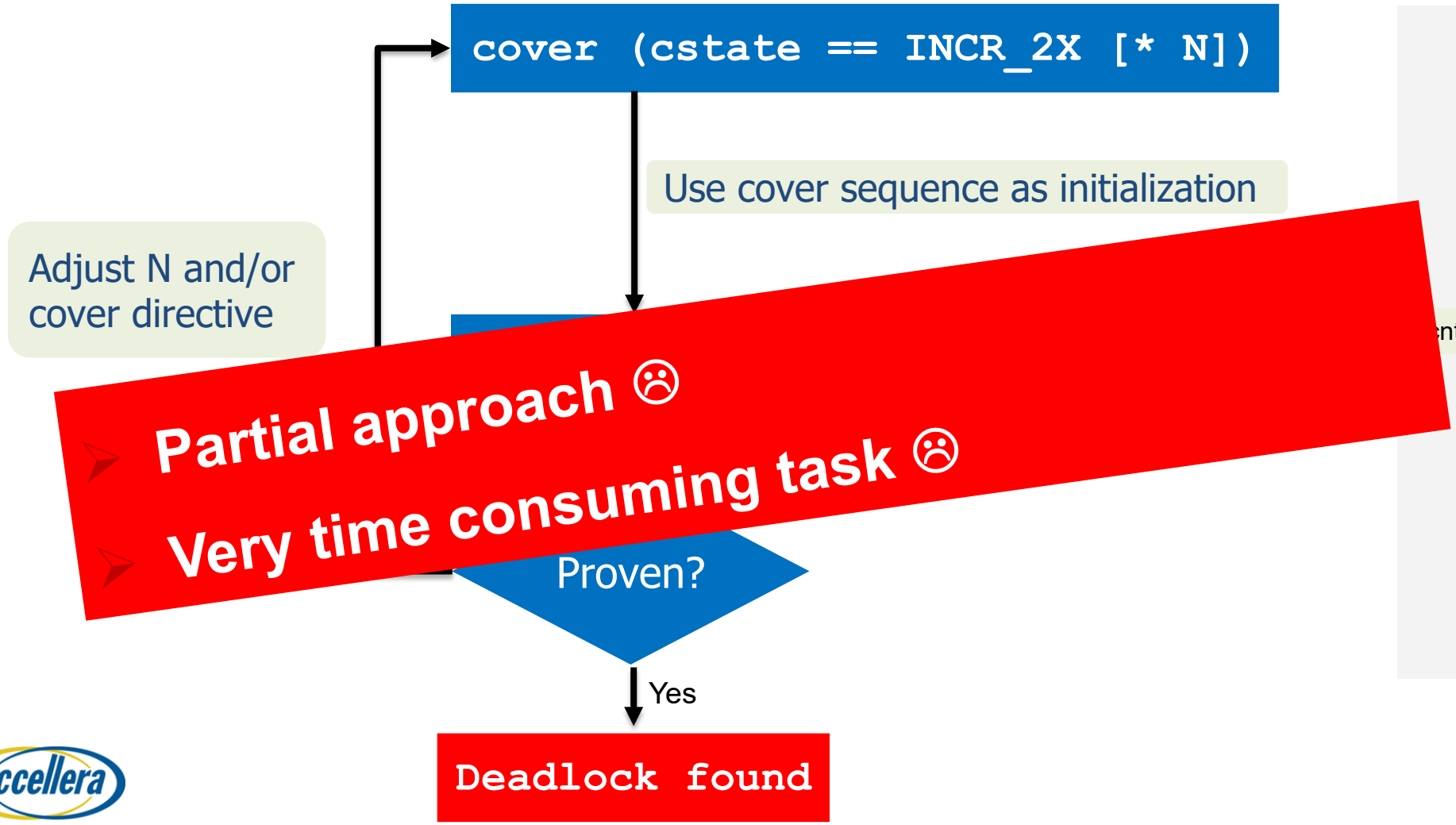
Use SVA Safety Properties For Deadlock Analysis

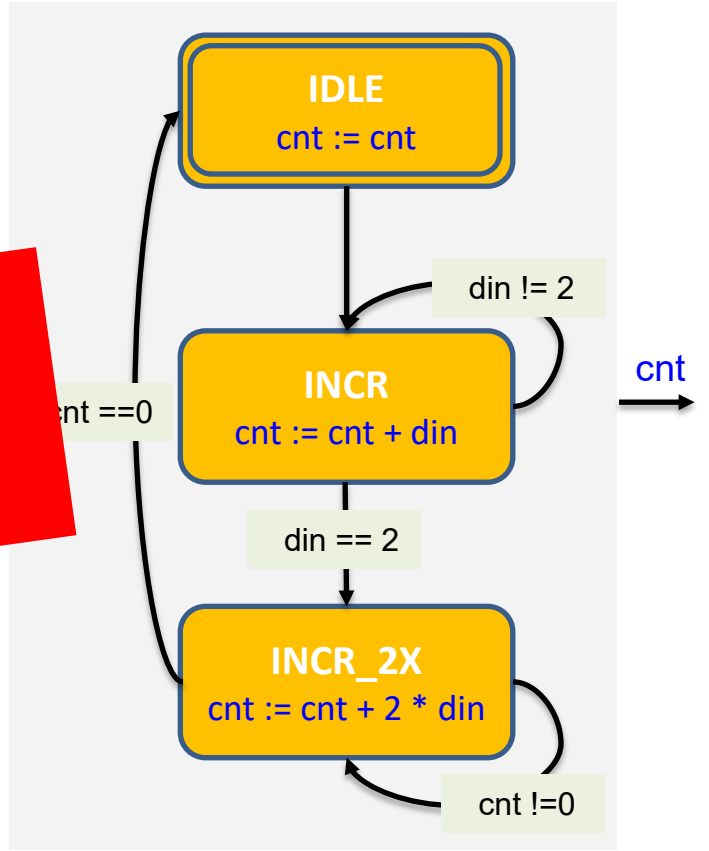
Adjust N and/or cover directive





Use SVA Safety Properties For Deadlock Analysis

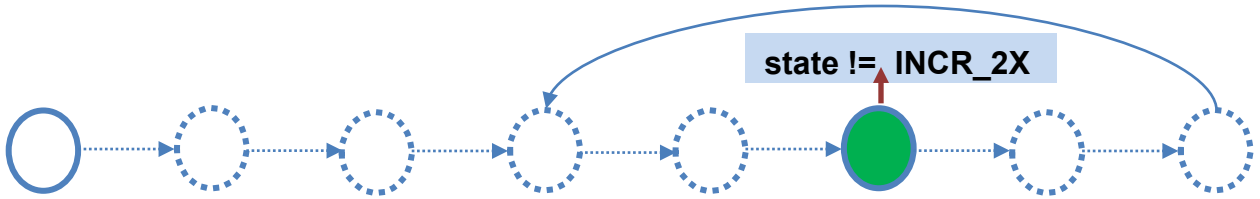




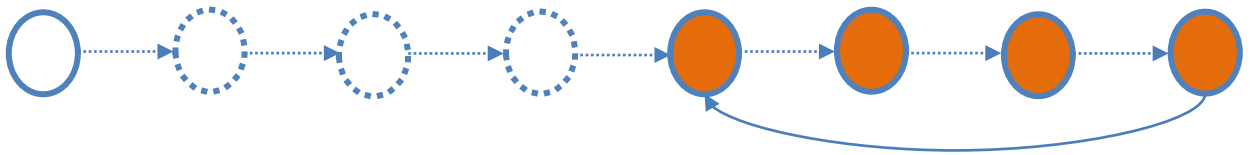
Use SVA Liveness Properties For Deadlock Analysis

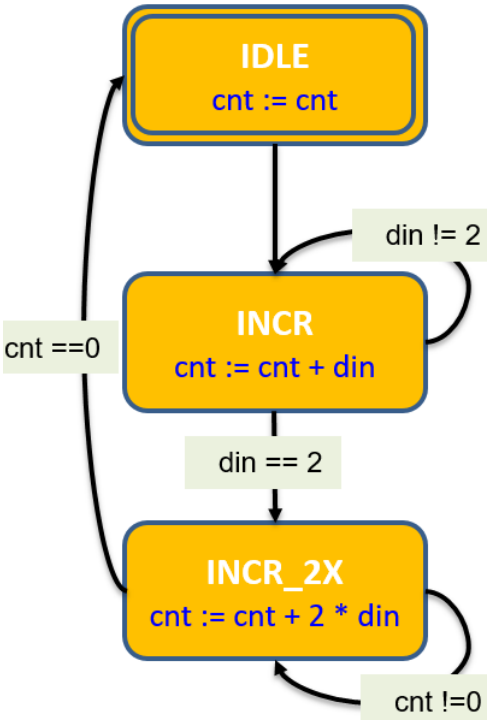
ϕ ≡ always property (s_eventually (state != INCR_2X))

- **ϕ is Valid :** in all traces FSM can exit state INCR_2X



- **ϕ is violated:** There exists a trace where FSM is in state INCR_2X for almost all the time

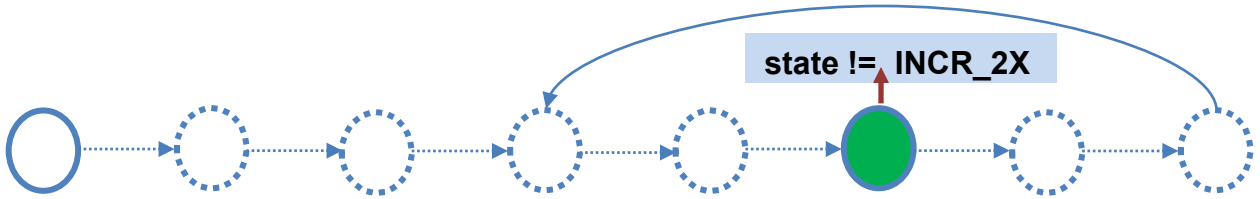




Use SVA Liveness Properties For Deadlock Analysis

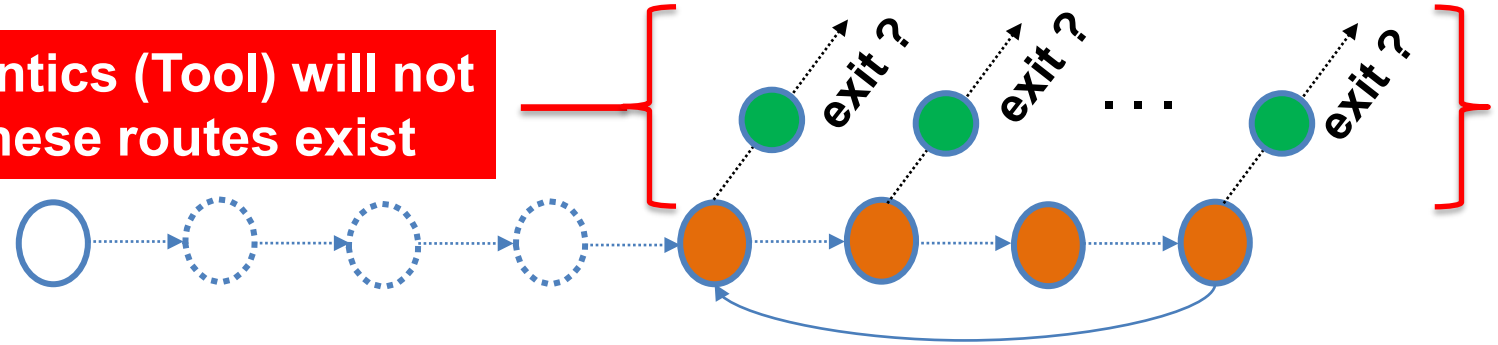
ϕ ≡ always property (s_eventually (state != INCR_2X))

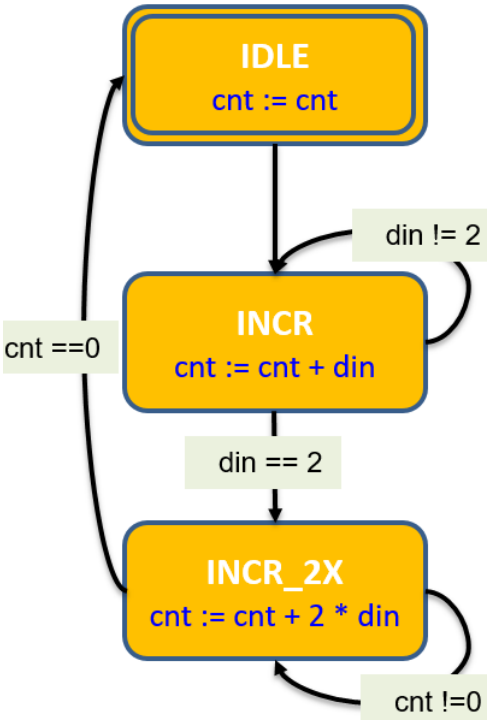
- **ϕ is Valid :** in all traces FSM can exit state INCR_2X



- **ϕ is violated:** There exists a trace where FSM is in state INCR_2X for almost all the time

LTL semantics (Tool) will not check if these routes exist

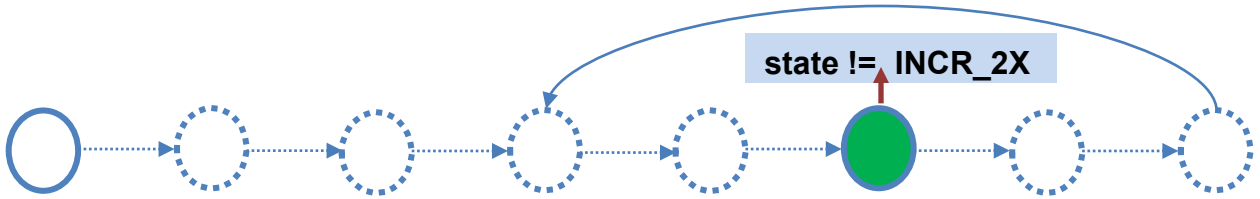




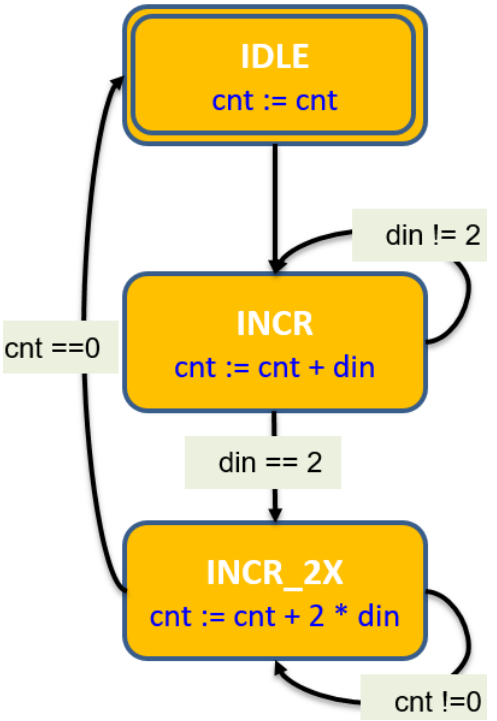
Use SVA Liveness Properties For Deadlock Analysis

ϕ ≡ always property (s_eventually (state != INCR_2X))

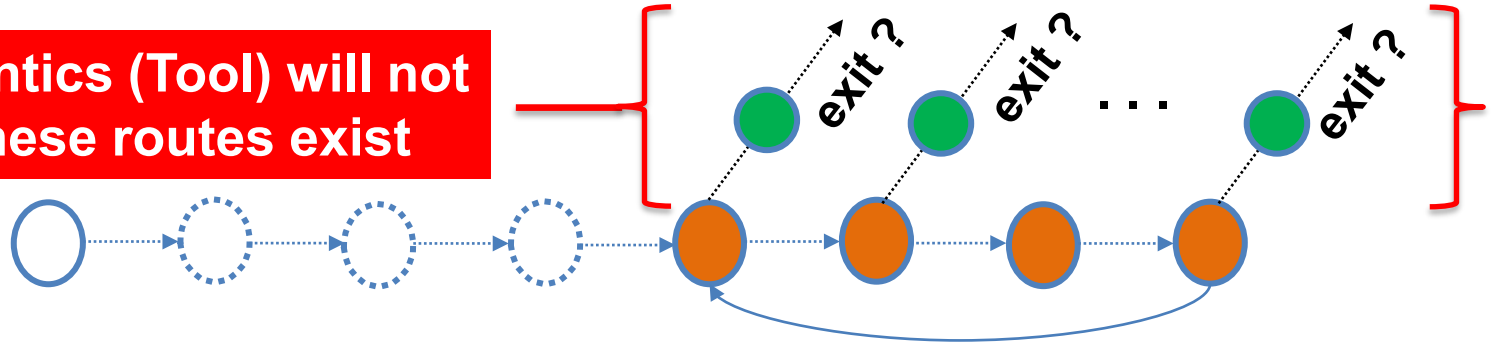
- **ϕ is Valid :** in all traces FSM can exit state INCR_2X



- **ϕ is violated:** There exists a trace where FSM is in state INCR_2X for almost all the time



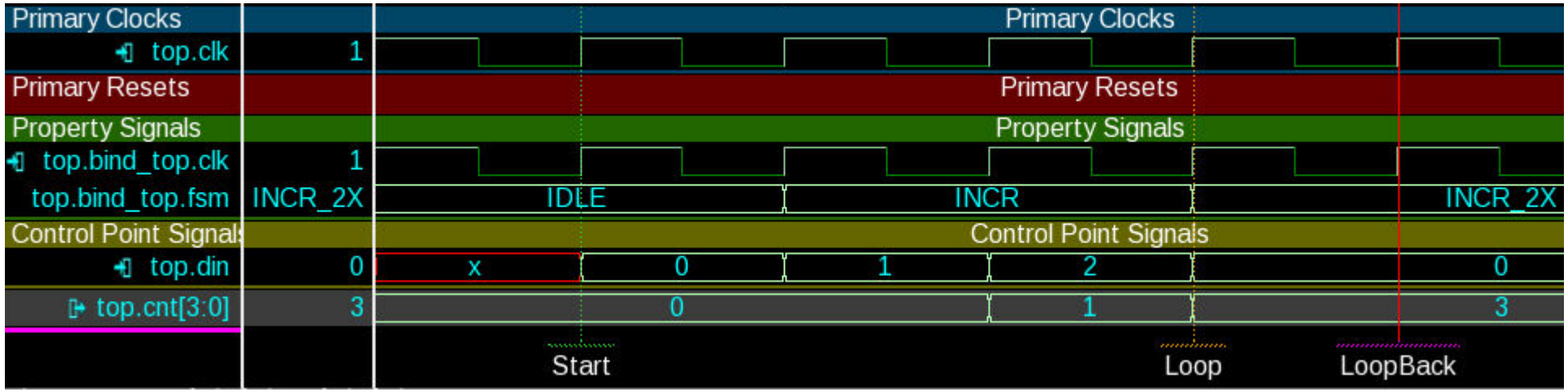
LTL semantics (Tool) will not check if these routes exist

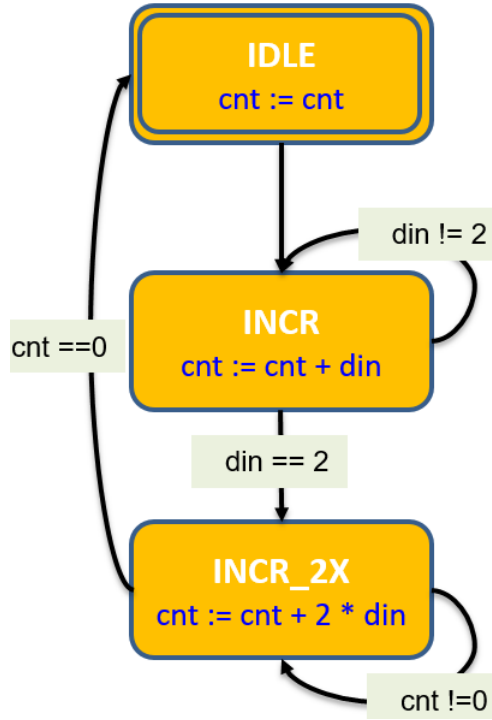


→ **Maybe-escapable Deadlocks**

Example

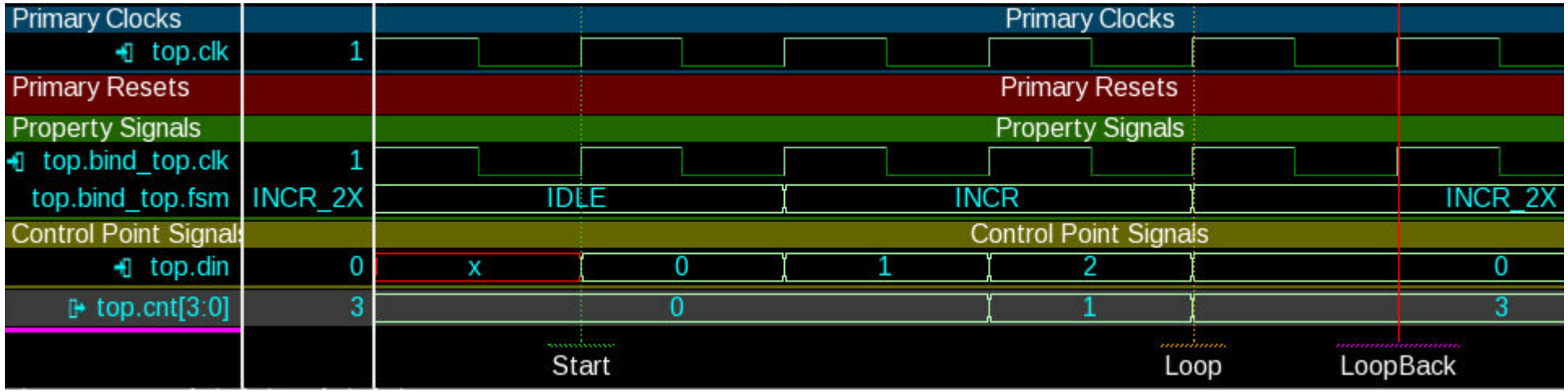
```
//Check
assert property (s_eventually (state != INCR_2X))
```

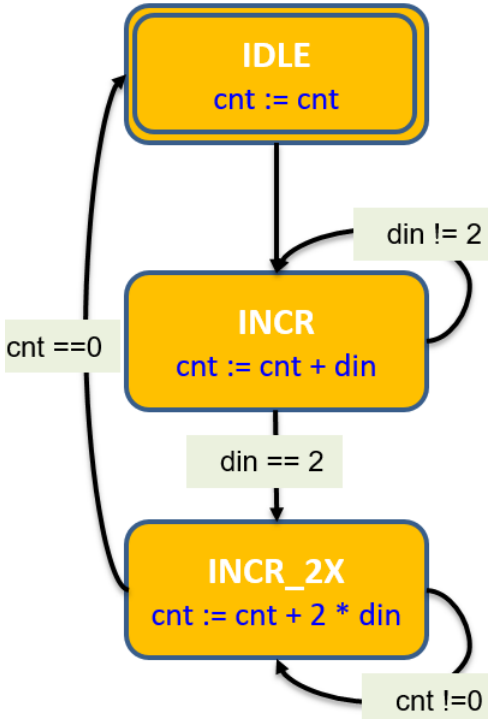




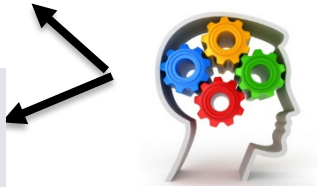
Example

```
//Check
assert property (s_eventually (state != INCR_2X))
```



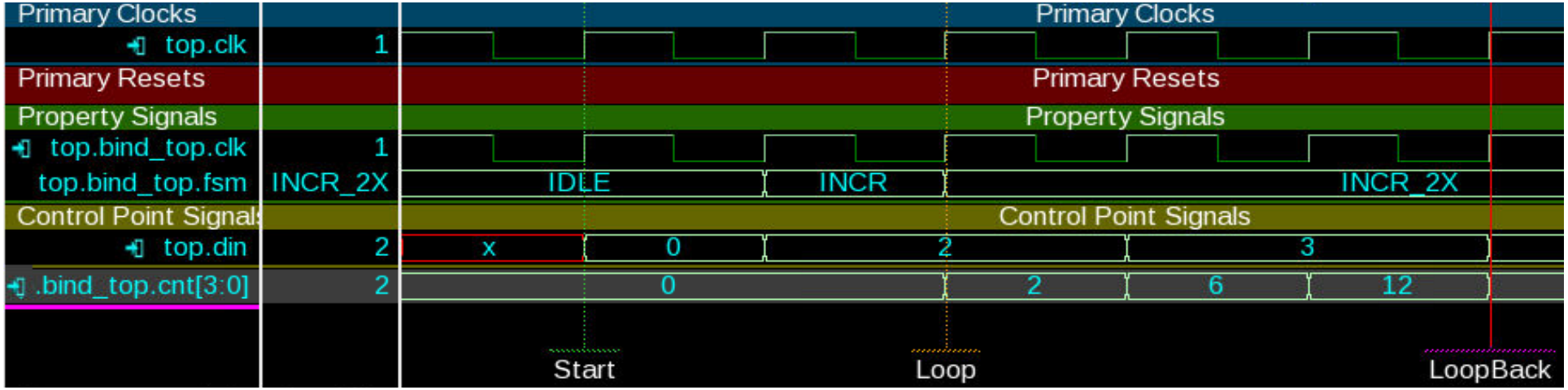


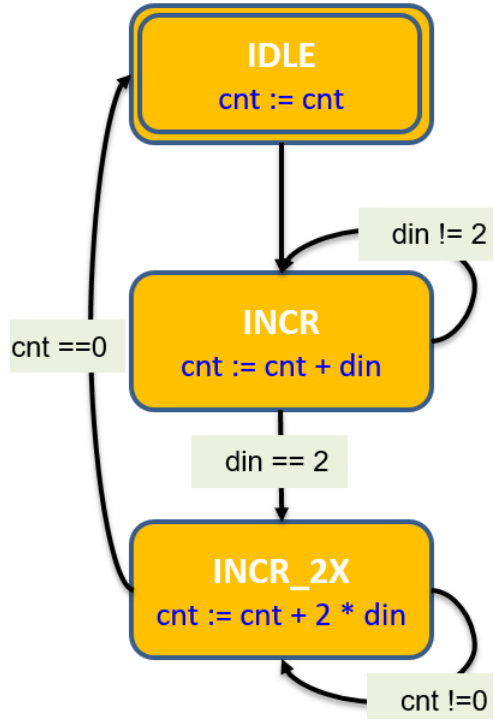

```
//Fairness conditions:
assume property (s_eventually (din != 'b0))
```



Example

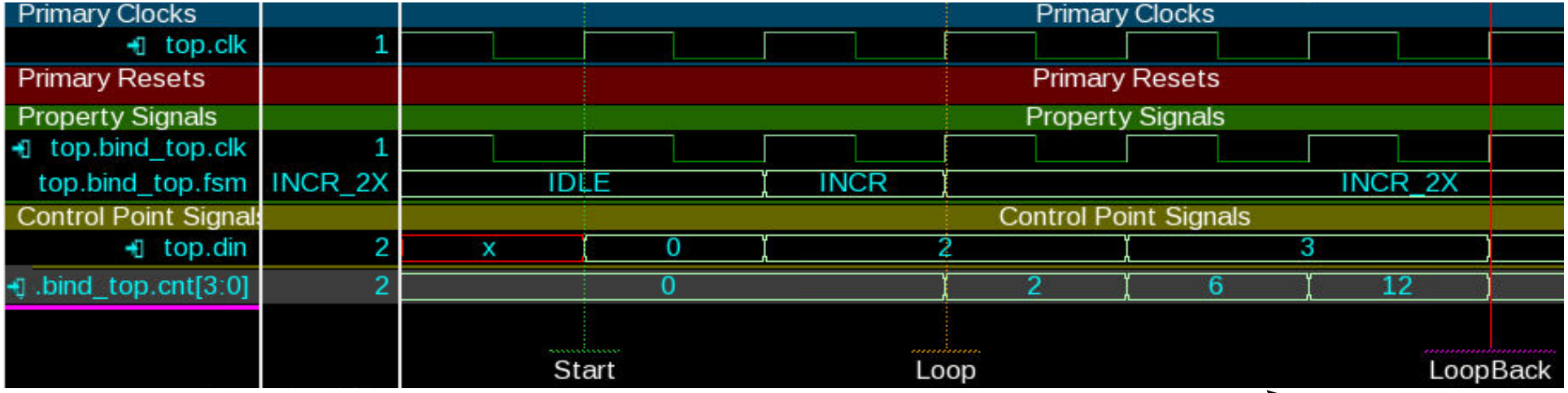
```
//Check
assert property (s_eventually (state != INCR_2X))
```

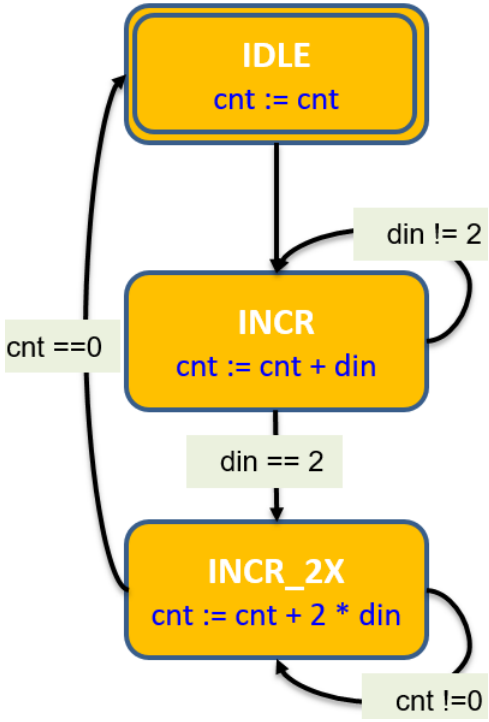


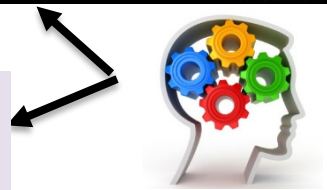


Example

```
//Check
assert property (s_eventually (state != INCR_2X))
```



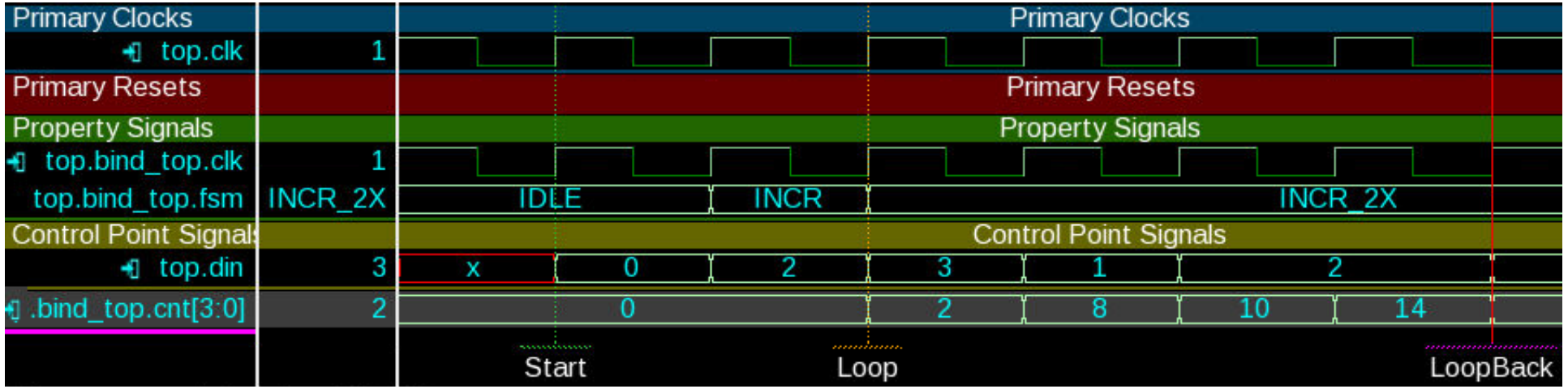


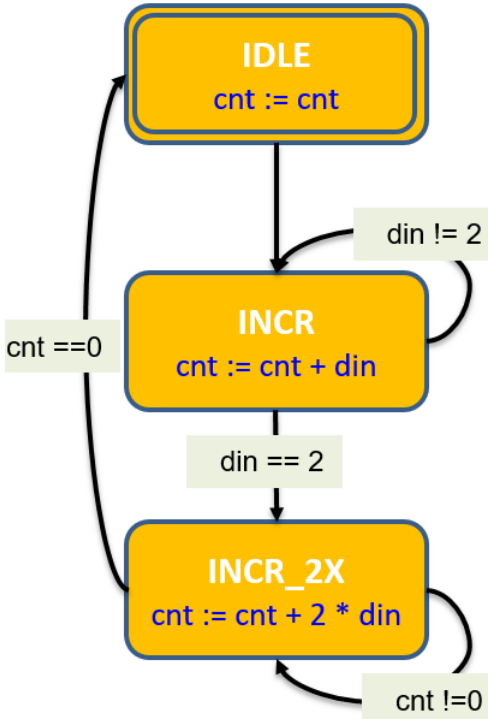


```
//Fairness conditions:
assume property (s_eventually (din != 'b0))
assume property (s_eventually (din == 2'b01))
```

Example

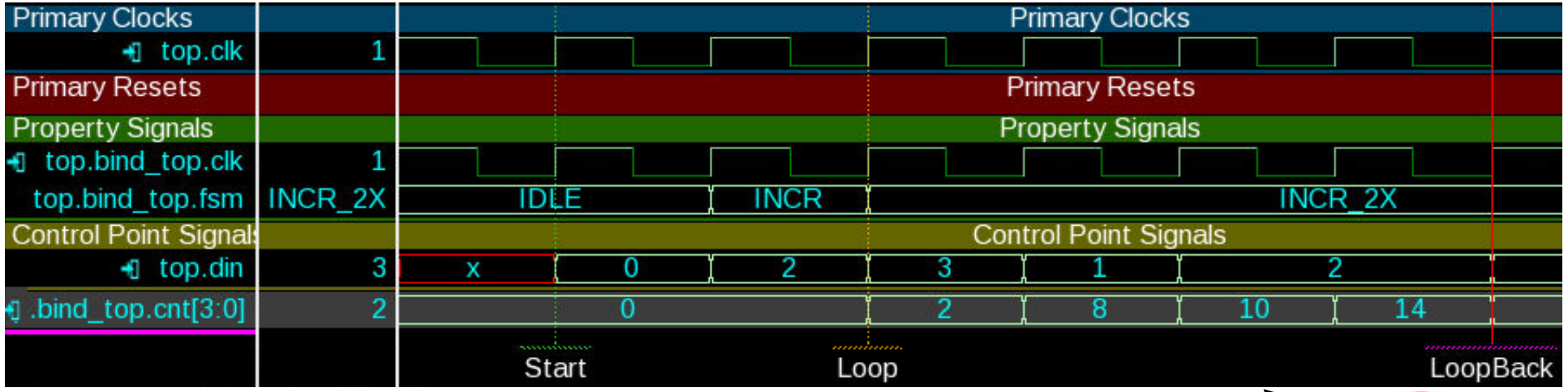
```
//Check
assert property (s_eventually (state != INCR_2X))
```

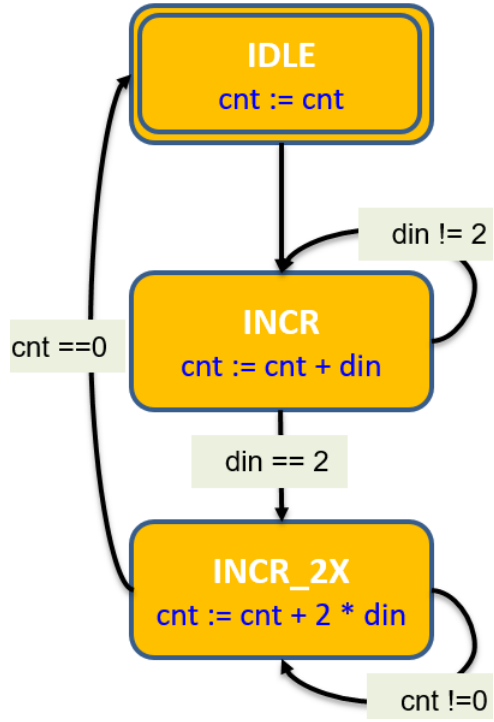


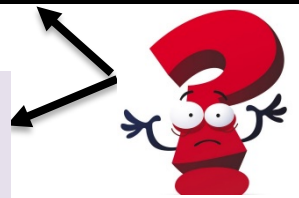


Example

```
//Check
assert property (s_eventually (state != INCR_2X))
```







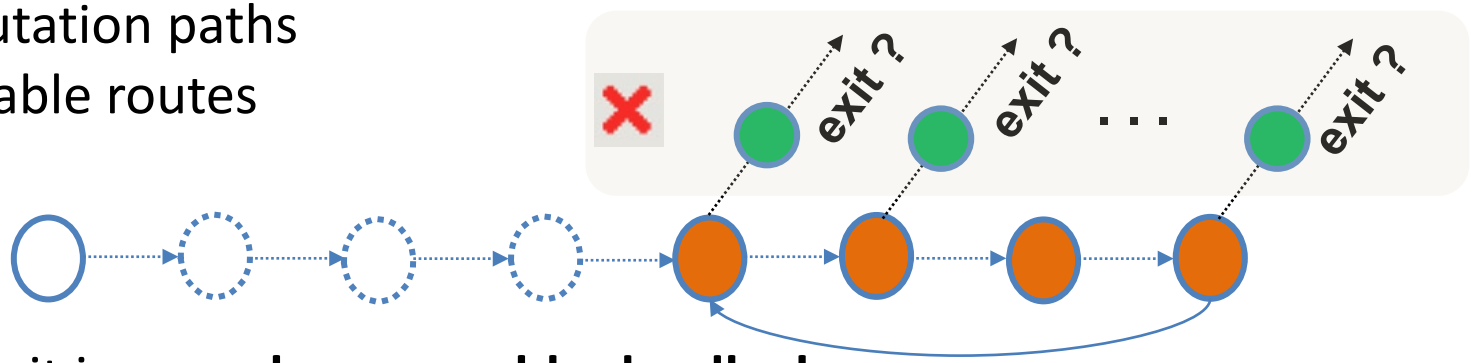
```
//Fairness conditions:
assume property (s_eventually (din != 'b0))
assume property (s_eventually (din == 2'b01))
???
```

Use SVA for Deadlock Analysis

- LTL semantics (Tool) will not check if escape routes exist
- User has to add fairness conditions to guide the tool finding escape routes
- User is often facing a painful debug activity
  - very difficult
  - time consuming
  - inefficient
  - If user don’t succeed to find escape routes, tool will not check other paths and eventually miss real deadlocks

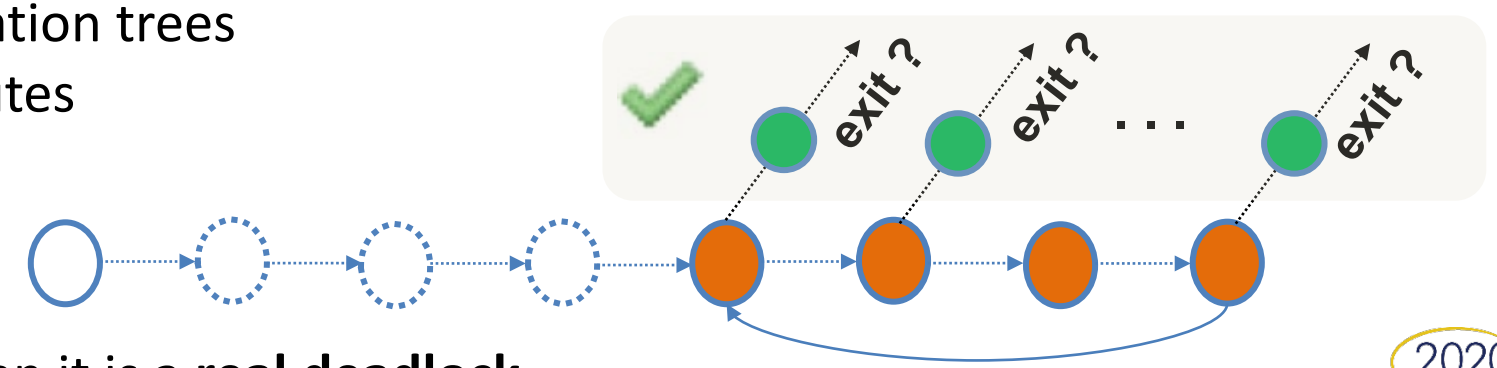
LTL vs CTL

- **LTL**: semantics based-on computation paths
  - It does not check for escapable routes



  - If deadlock is reported, then it is a **maybe-escapable deadlock**

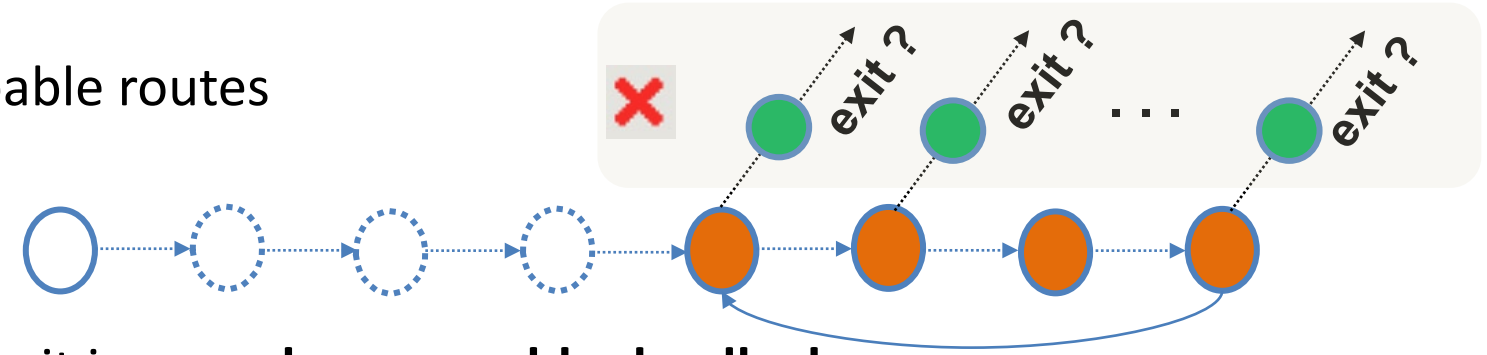
- **CTL:** semantics based computation trees
  - It checks for escapable routes



  - If deadlock is reported, then it is a **real deadlock**

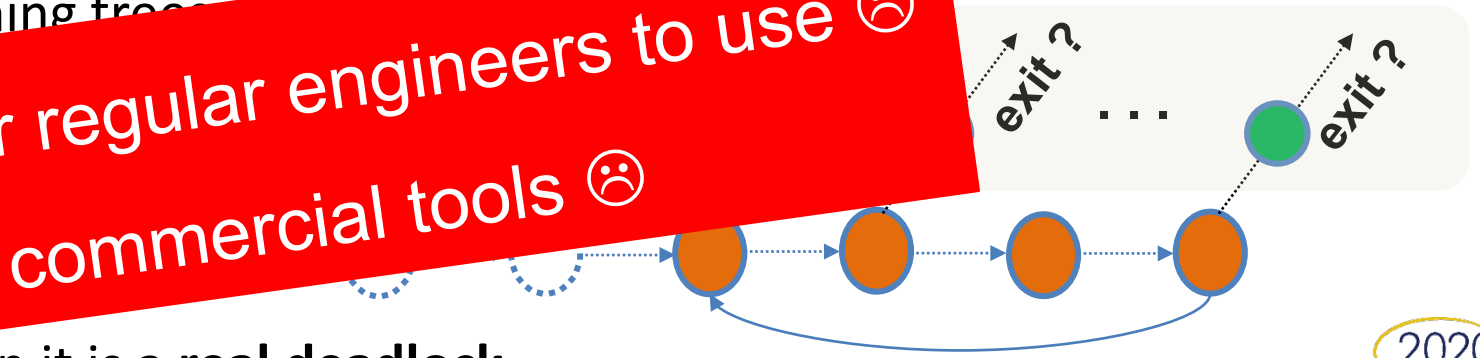
LTL vs CTL

- **LTL**: semantics consider paths
  - It does not check for escapable routes



  - If deadlock is reported, then it is a **maybe-escapable deadlock**

- **CTL:** semantics consider branching trees
  - It checks for escapable routes



  - If deadlock is reported, then it is a **real deadlock**

× CTL is too "academic" for regular engineers to use ☹

× CTL is not supported by commercial tools ☹

Summary So Far For Deadlock Analysis

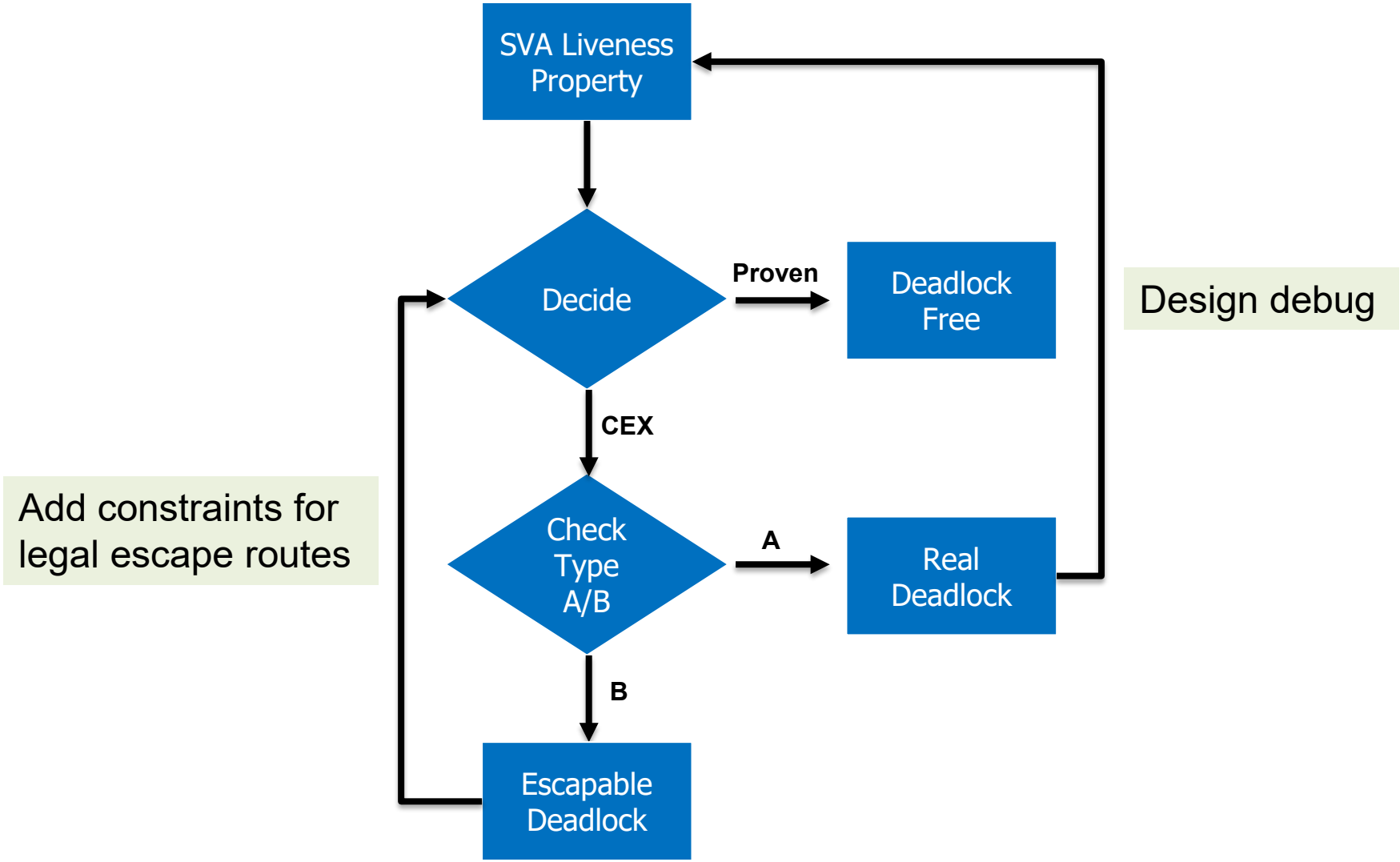
Approach	Rating	Comment
Simulation	☹ ☹	Incomplete and very time consuming
LTL/SVA Safety Properties	☹ ☹	Incomplete and very time consuming
LTL/SVA Liveness Properties	☹	Incomplete and time consuming
CTL Properties	☹	Academic and no commercial tool support

We need a new approach

Mentor's Approach Finds Deadlock Issues Faster

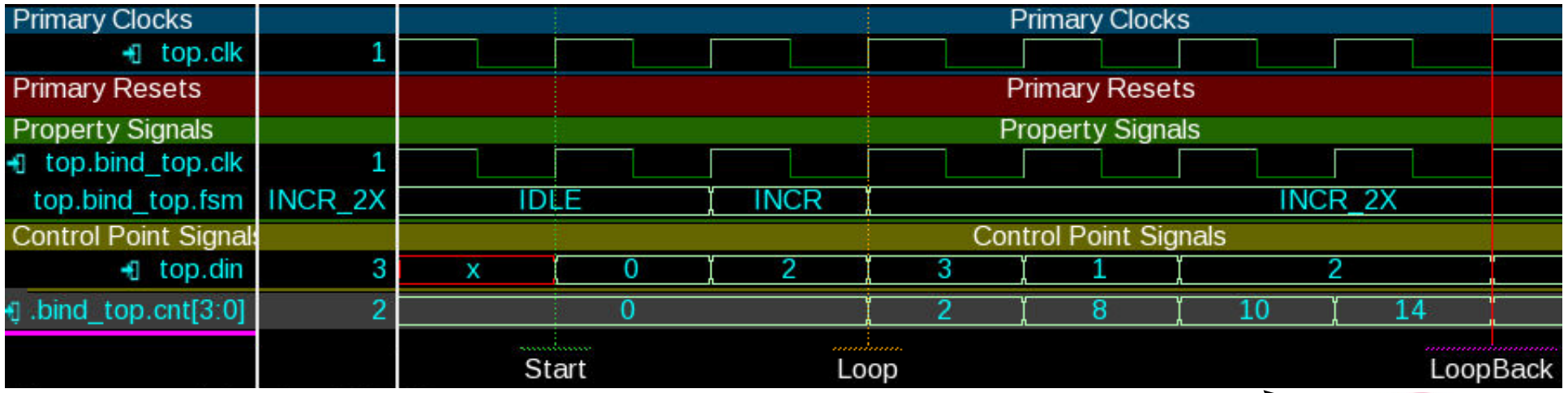
- **Combining LTL and CTL semantics for Liveness properties**
  - LTL/SVA used explicitly to express deadlock properties
  - CTL used implicitly to find real deadlock cases or escapable routes
- **Approach implemented in Questa PropCheck**
  - No change in current flow
  - Assertion writing and debugging are same as before (as for SVA)
  - User does not have to infer/write CTL properties
  - Automated engine orchestration to deal with inferred CTL properties
- **Approach is complete**
  - Reporting of proofs for deadlock free cases
  - Reporting of real deadlock cases
  - Reporting of escapable deadlock cases

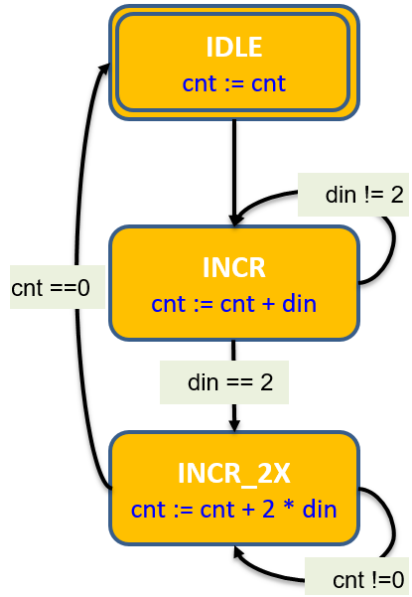
Questa Formal Deadlock Approach

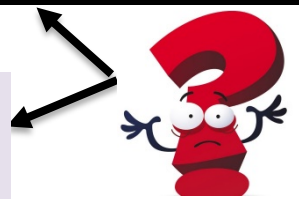


Example

```
//Check
assert property (s_eventually (st != INCR_2X))
```

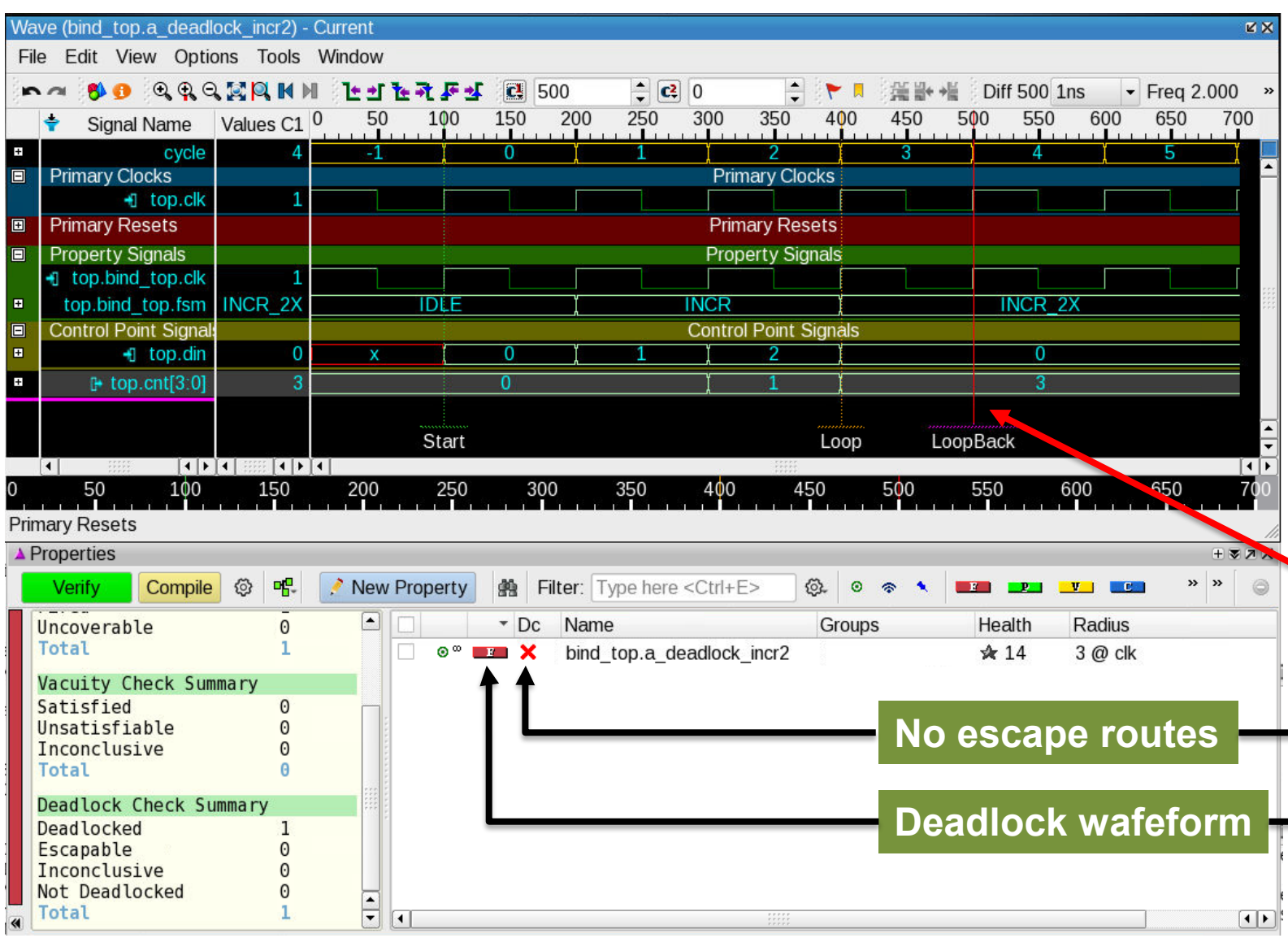


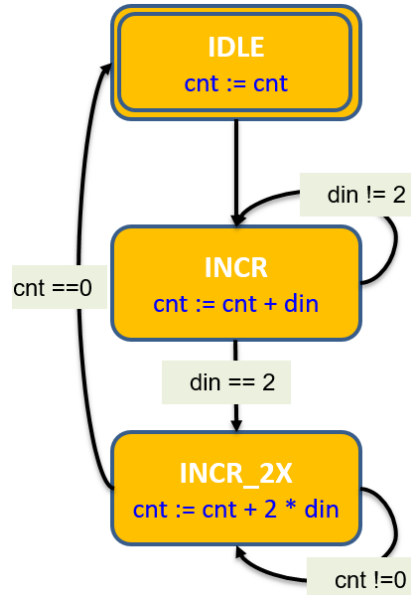




```
//Fairness conditions:
assume property (s_eventually (din != 'b0))
assume property (s_eventually (din == 2'b01))
???
```

Questa Formal Deadlock Analysis: Example





Real Deadlock:
cnt will never be 0 to exit state INCR_2X

No escape routes

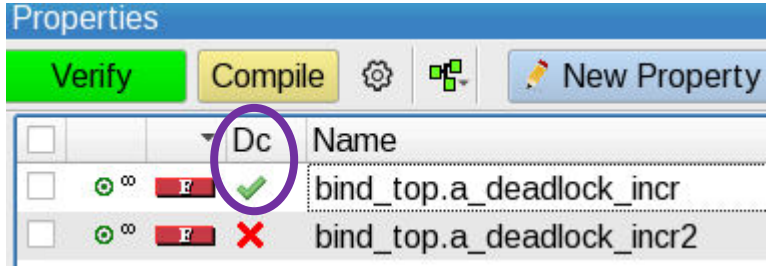
Deadlock wafeform

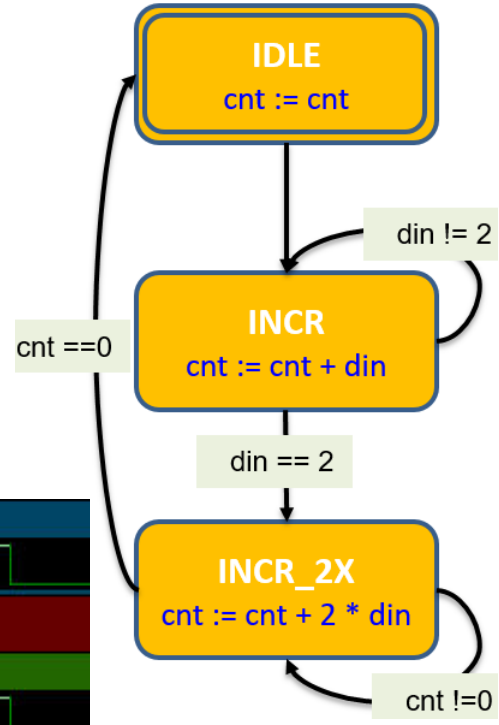
accellera SYSTEMS INITIATIVE

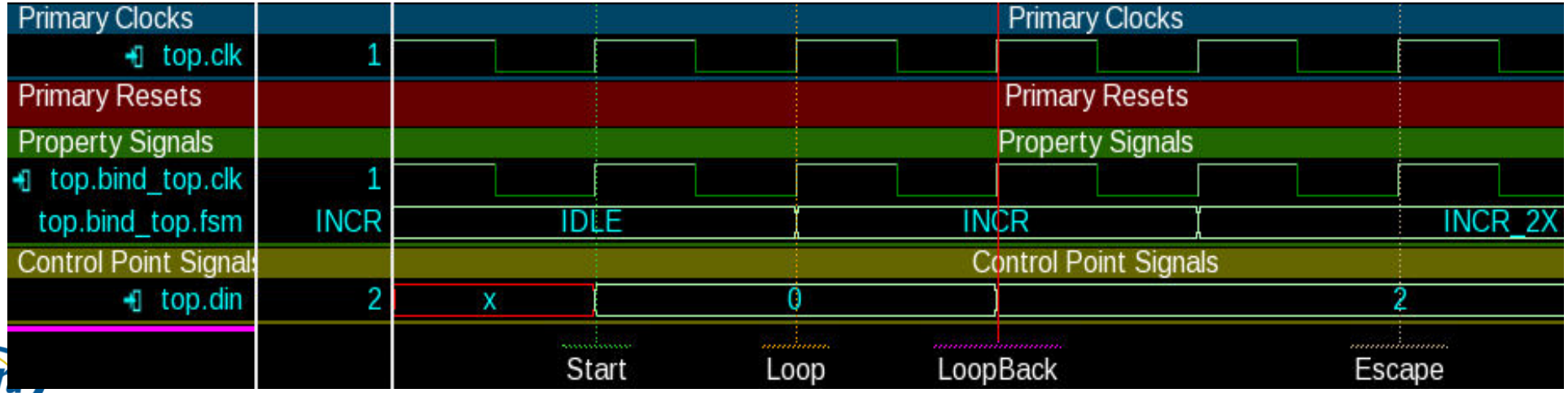
2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

Escapable Deadlock

```
//Check
assert property (s_eventually (state != INCR_2X))
assert property (s_eventually (state != INCR))
```







“ This debug work is much simpler than the one with the traditional method looking only at maybe-escapable deadlocks. Having the extra information that it is not an escapable deadlock, allows to reduce debug time a lot “

Laurent Arditi, PhD
Arm Ltd

Real World Case Study: Arm's Usage of Questa PropCheck Deadlock Analysis

- Instruction Fetch unit FSMs
  - Local FSMs are resilient to incorrect or unexpected environment behaviours
  - Maybe-escapable deadlocks are frequent and safe
  - A few results showed unescapable deadlocks
  - Proof time is a few minutes, with no overhead for also running the unescapable deadlock checks
- L1 data cache arbiter
  - All assertions are proven
- Credit-based protocol
  - Can prove that no credit is lost
  - A few critical bugs found

Some Cases for Deadlock Analysis

- **FSM Deadlocks**
  - No deadlock on some states
- **Arbitration**
  - Every process often get grant (no starvation)
- **Interfaces**
  - Server/master often get bus access
  - Server/master is often ready to accept/send data
- **Handshaking**
  - Often request acknowledged
- …

- **Applicable for any SVA livness property**

Summary

- The risk of a design going into deadlock is nearly impossible to detect with RTL simulation; hard to do with traditional formal

- Detecting RTL deadlock is now easier with Mentor's PropCheck using these advanced algorithms under-the-hood