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

Tecnhology 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

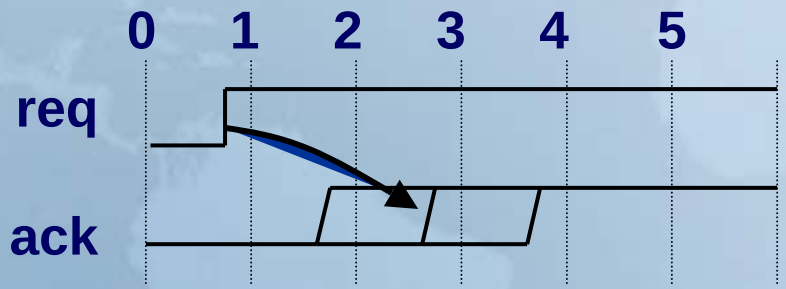
End: 5:00pm

SystemVerilog Assertions Language Overview

Dr. Bassam Tabbara
Technical Manager, R&D
Novas Software, Inc.

What is an Assertion?

A concise description of [un]desired behavior



Example intended behavior

"After the request signal is asserted, the acknowledge signal must come 1 to 3 cycles later"

SystemVerilog and Assertions

- Enhance SystemVerilog to support Assertion-Based Verification
  - White-box (inside block) assertions
  - Black-box (at interface boundaries) assertions
- Syntactic compatibility
  - Easy to code directly in-line with RTL
- Support for design and assertion reuse
  - Support for assertion “binding” to design from separate file
- Monitoring the design
  - Concurrent (“standalone”) assertions
  - Procedural (“embedded”) assertions

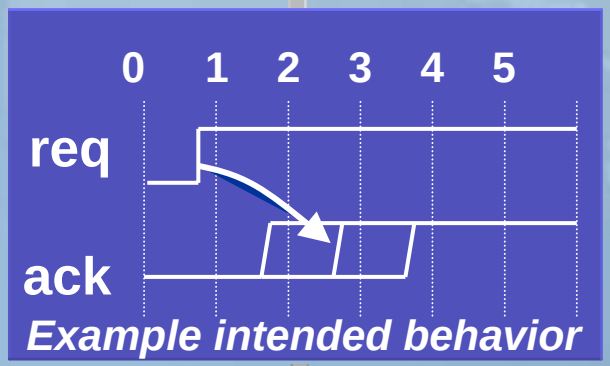
SystemVerilog Assertion Goals

- Expressiveness
  - Cover large set of design properties:
    - Block Implementation
    - Block Interaction
- Usability:
  - Easy to understand and use by:
    - Design Engineer
    - Verification Engineer
- Formalism:
  - Formal semantics to ensure correct analysis
  - Consistent semantics between simulation and formal design validation approaches

Concise and Expressive SVA

SVA Assertion

```
property req_ack;
 @(posedge clk) req ##[1:3] $rose(ack);
endproperty
as_req_ack: assert property (req_ack);
```



Example intended behavior

HDL Assertion

```
always @(posedge req)
  begin
     repeat (1) @(posedge clk);
     fork: pos_pos
        begin
           @(posedge ack)
           $display("Assertion Success",$time);
            disable pos_pos;
        end
       begin
          repeat (2) @(posedge clk);
          $display("Assertion Failure",$time);
          disable pos_pos;
       end
     join
  end // always
```

The Basics

4 Easy Lessons

Immediate assertions

```
assert ( expression ) action_block;
action_block ::= [statement] [else statement]
```

- Appears as a procedural statement
- Follows simulations semantics, like an “if”

```
if (expression) action_block;
```

- Action block
  - Executes *immediately*
  - Can contain system tasks to control severity, for example: **`$error`**, **`$warning`**, …

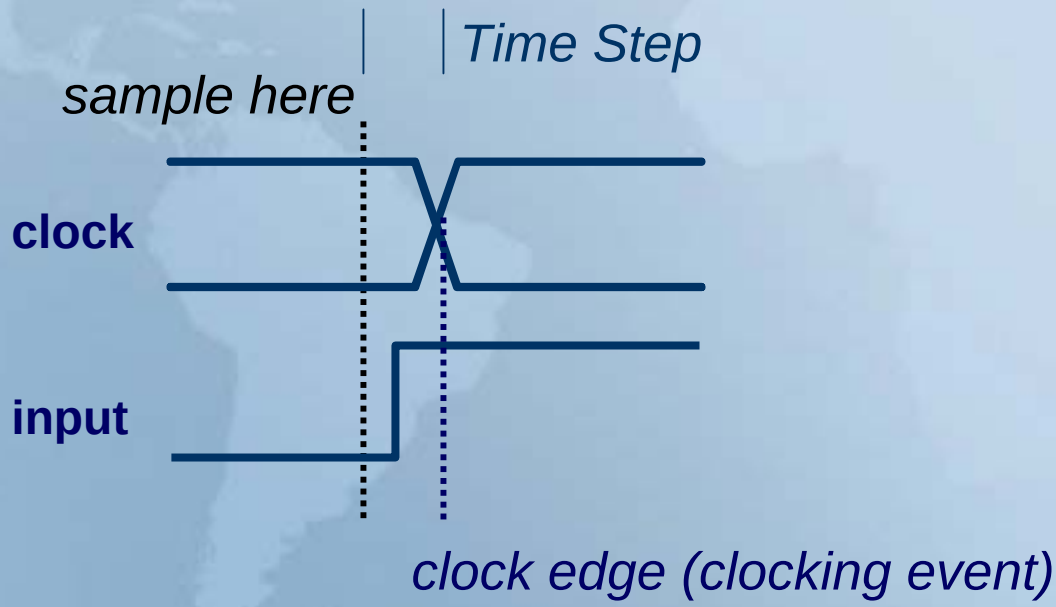
Concurrent assertions

```
assert property ( property_instance_or_spec )
  action_block;
action_block ::= [statement] [else statement]
```

- Appears outside/inside procedural context
- Follows cycle semantics using **sampled** values
- Action block
  - Executes in *reactive* region
  - Can contain system tasks to control severity, for example: **`$error`**, **`$warning`**, …

Assertion Sampling

- Values sampled at end of previous Time Step



Hierarchy of Assertion Constructs


```
assert, cover, bind,
declarative instantiation,
procedural instantiation
```

```
disable iff, not,
implication
```

```
repetition,(cycle)delay,
and, or, intersect,
first_match,
within, throughout
```

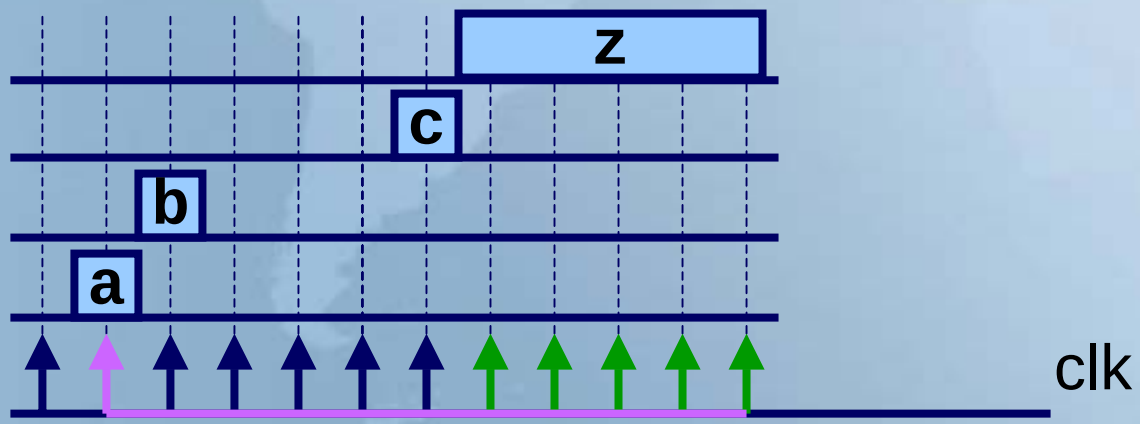
```
<expr>, <function>,
<temporal_edge_function>,
ended, matched
```

Congratulations !!!

Sequential Regular Expressions

- Describing a sequence of events
- Sequences of Boolean expressions can be described with a specified time step in-between

```
@(posedge clk) a ##1 b ##4 c ##[1:5] z;
```

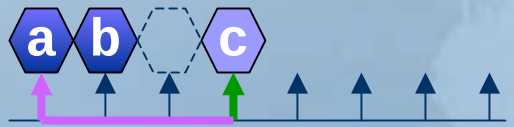


Regular Expression Examples

Basic sequencing

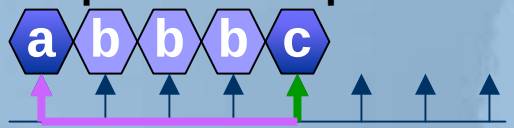

```
a ##1 b ##1 c …##1 z
```

Sequence with Delay



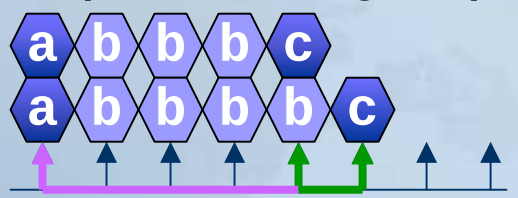

```
a ##1 b ##2 c
```

Expression Repetition



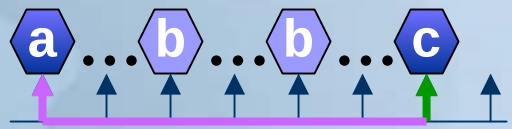

```
a ##1 b[*3] ##1 c
```

Expression Range Repetition



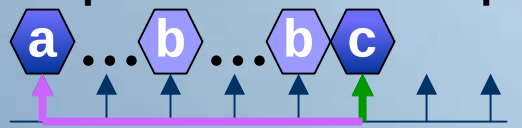

```
a ##1 b[*3:4] ##1 c
```

Expression Non-Consecutive "Counting" Repetition




```
a ##1 b[*=2] ##1 c
```

Expression "Goto" Repetition




```
a ##1 b[*->2] ##1 c
```

Sequences Encapsulate Behavior

- Can be Declared

```
sequence <name>[(<args>)];
  [@(<clocking>)] <sequence>;
endsequence
```

```
sequence s1(a,b);
  @(posedge clk) a[*2] ##3 b;
endsequence
```

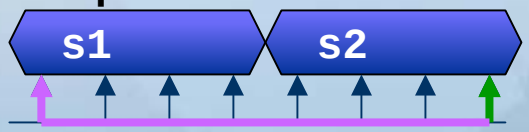
- Sequences Can Be Built From Other Sequences

```
sequence s2; @(posedge clk) c ##1 d;
endsequence
sequence s3; @(posedge clk) s1(e,f) ##1 s2;
endsequence
```

- Operations to compose sequences
  - and, or, intersect, within, throughout

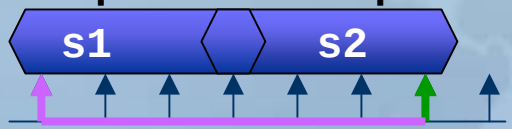
Sequence Operations

Sequence Concatenation



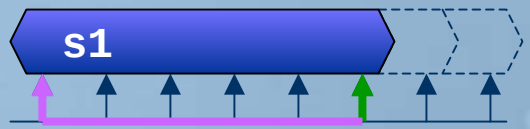

```
s1 ##1 s2
```

Sequence Overlap



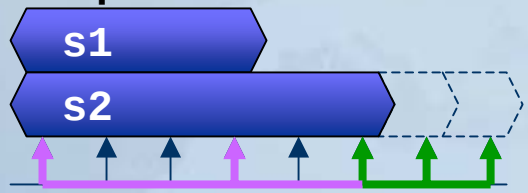

```
s1 ##0 s2
```

First Match




```
first_match(s1)
```

Sequence “and”

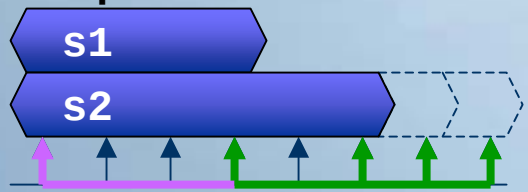



```
s1 and s2
```

Sequence “intersect”


```
s1 intersect s2
```

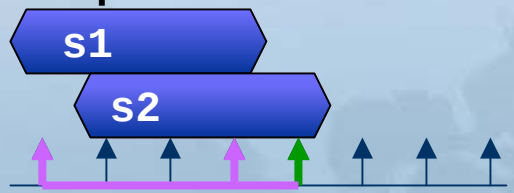
Sequence “or”




```
s1 or s2
```

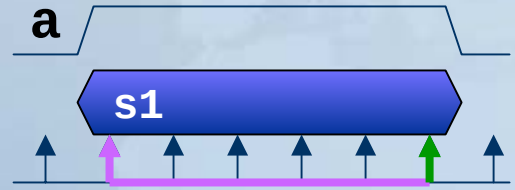
More Sequence Operations

Sequence Ended




```
s1 ##1 s2.ended
```

Expression “throughout” a Sequence



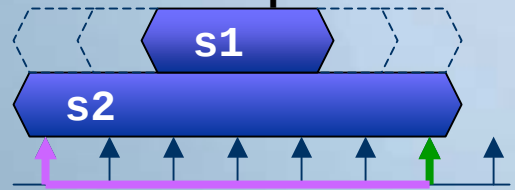

```
a throughout s1
```

Temporal Edge Functions


```
!a ##1 $rose(a);
```


```
a[*2] ##1 $fell(a)
```

Sequence “within” another Sequence




```
s1 within s2
```

Property Definition

- Property Declaration: **`property`**
  - Declares property by name
  - Formal parameters to enable property reuse
  - Top Level Operators
    - **`not`** *desired/undesired*
    - **`disable iff`** *reset*
    - **`|->, |=>`** *precondition*
- Assertion Directives
  - **`assert`** – checks that the property is never violated
  - **`cover`** – tracks all occurrences of property

```
property prop1(a,b,c,d);
  disable iff (reset)
            (a) |-> [not](b ##[2:3]c ##1 d);
endproperty
assert1: assert prop1 (g1, h2, hxl, in3);
```

Property implication

```
sequence_expr |-> [not] sequence_expr
sequence_expr |=> [not] sequence_expr
```

- |-> is overlapping implication
- |=> is non-overlapping implication
  same as:

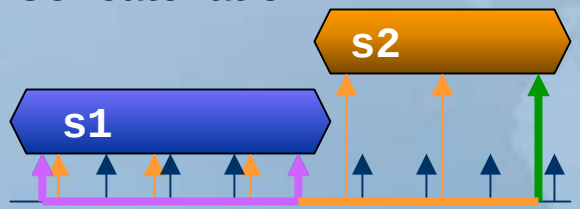
```
sequence_expr ##1 `true|-> [not] sequence_expr
```

- Most commonly used to attach a precondition to sequence evaluation

Multiple Clock Support

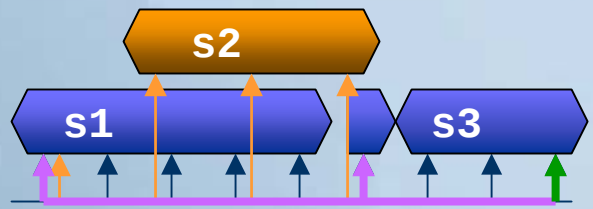
- Clocking event controls must be specified for each subsequence

Multi-Clock Sequence Concatenation




```
@(clk1) s1 ## @(clk2) s2
```

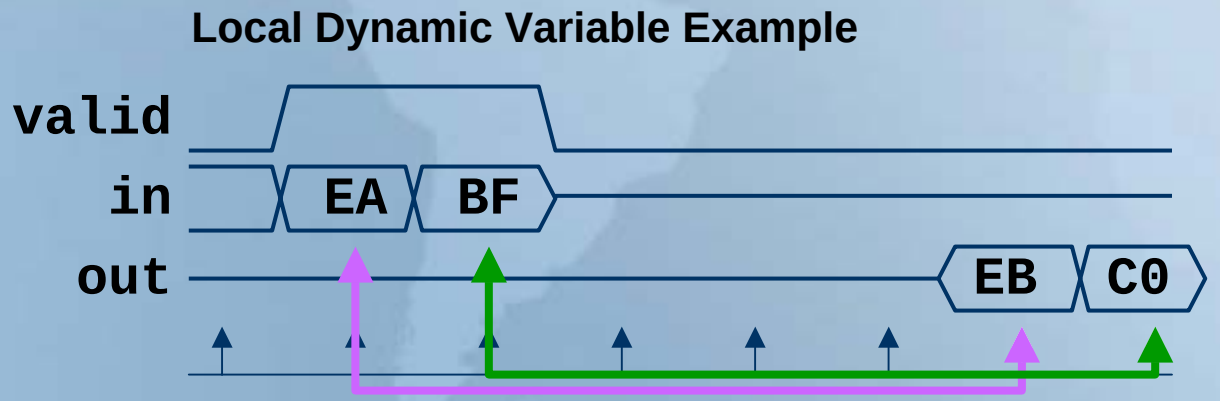
Multi-Clock Sequence Matching




```
s1 ##1 s2.matched ##1 s3
```

Manipulating Data: Local Dynamic Variables

- Declared Locally within Sequence/Property
  - New copy of variable for each sequence invocation
- Assigned anywhere in the sequence
- Value of assigned variable remains stable until reassigned in a sequence




```
property e;
int x;
(valid,(x=in))|=> ##5(out==(x+1));
endproperty
```

Embedding Concurrent Assertions

```
sequence s1;
   (req && !gnt)[*0:5] ##1 gnt && req ##1 !req ;
endsequence

always @(posedge clk or negedge reset)
  if(reset == 0) do_reset;
  else if (mode == 1)
    case(st)
    REQ: if (!arb)
              if (foo)
                st <= REQ2;
                 PA: assert property (s1);
```

- Automatically Updates Enabling Condition as Design Changes
- Infers clock from instantiation

- Requires User to Update Manually as Design Changes

```
property p1;
  @(posedge clk) ((reset == 1) && (mode == 1)
                  && (st == REQ) && (!arb) && (foo)) => s1;
endproperty

DA: assert property (p1);
```

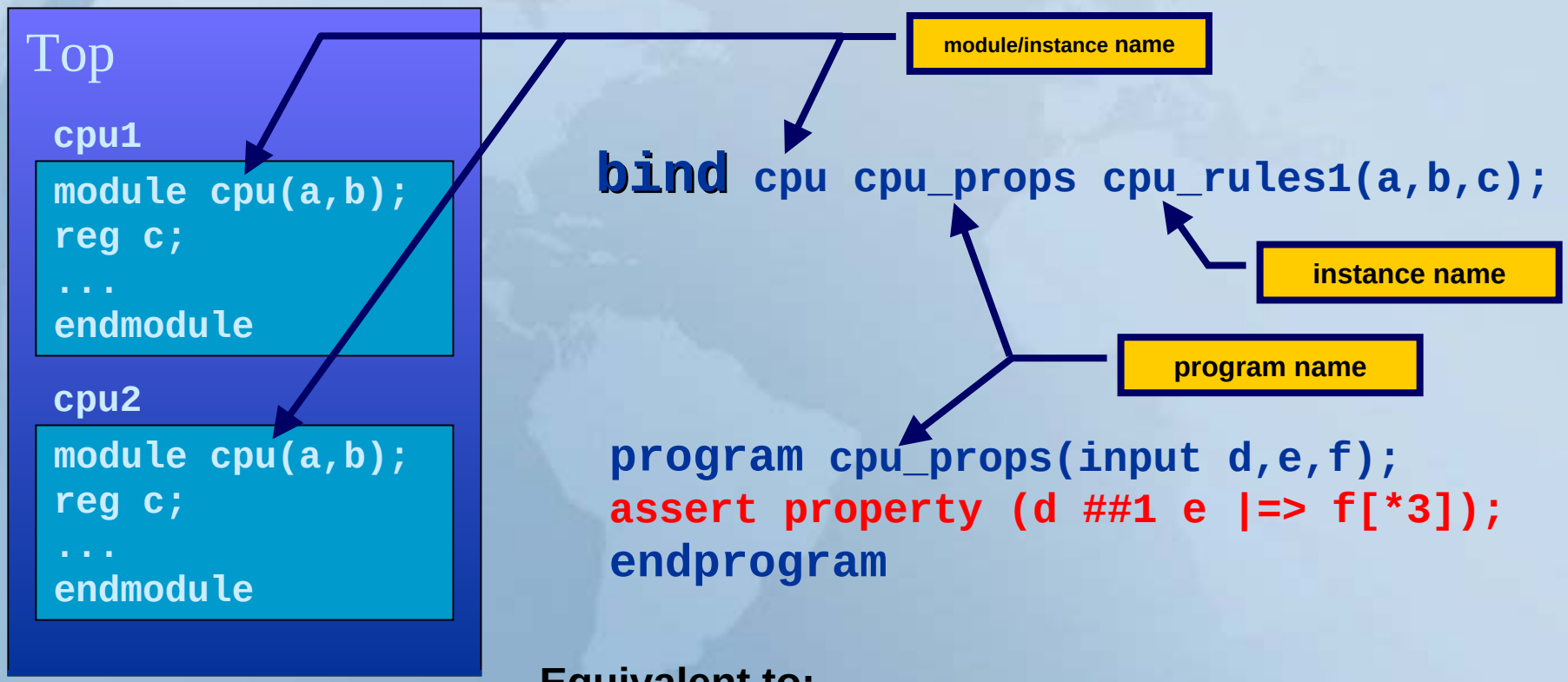
Bind statement

```
bind module_or_instance_name instantiation;
```

- No semantic changes to assertion
- Minimal change to design code
- Assertions included in the instantiation
- Allows binding a module, program and interface instance
- Mechanism to attach verification IP to module or module instance

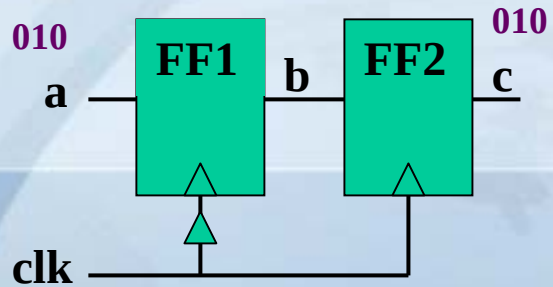
Bind statement

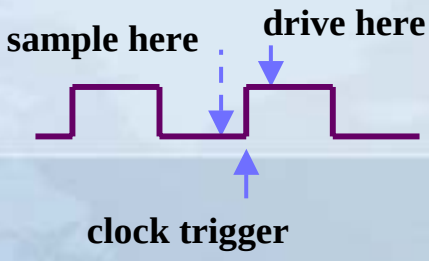
bind module_or_instance_name instantiation;



Equivalent to:

```
assert property (top.cpu1.a ##1 top.cpu1.b |=> top.cpu1.c[*3]);
assert property (top.cpu2.a ##1 top.cpu2.b |=> top.cpu2.c[*3]);
or
cpu_props cpu_rules1(a,b,c); // in module cpu
```



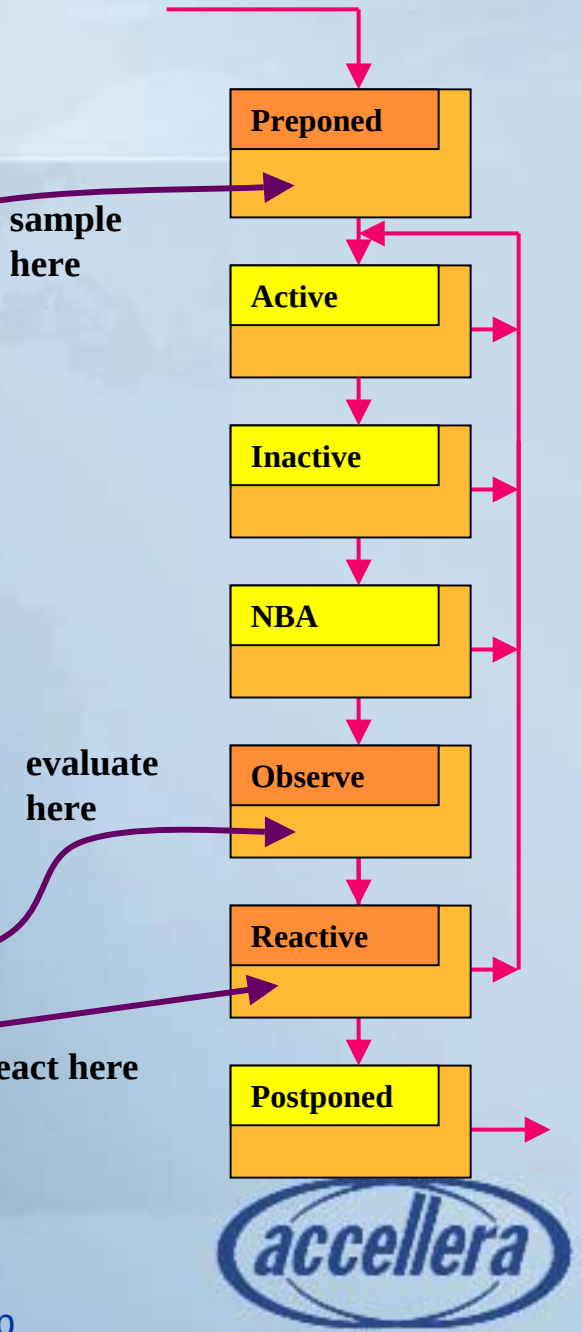



```
assign #0 gclk = clk;
always @(posedge gclk) b = a; // FF1
always @(posedge clk) c = b;   // FF2
```

```
clocking @(posedge clk)
sequence sa; !a ##1 a ##1 !a; endsequence
sequence sc; !c ##1 c ##1 !c; endsequence
```

```
property p  sa => [2] sc; endproperty
```

```
assert property(p)  pass_statement;
                    else  fail_statement;
```



SystemVerilog Assertions Summary

- Use clocked/sampled semantics for signals
  - Ensure compatibility with formal & synthesis tools
  - Avoid race conditions
- Declarative assertions add flexibility
  - Monitoring and evaluation
  - Reuse
- **Support design/assertion IP creation and reuse**
- **Enhanced scheduling allows building reactive testbenches**
- **Enhanced coverage of functional spec**