2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

Practical Asynchronous SystemVerilog Assertions

Doug Smith

accellera
SYSTEMS INITIATIVE

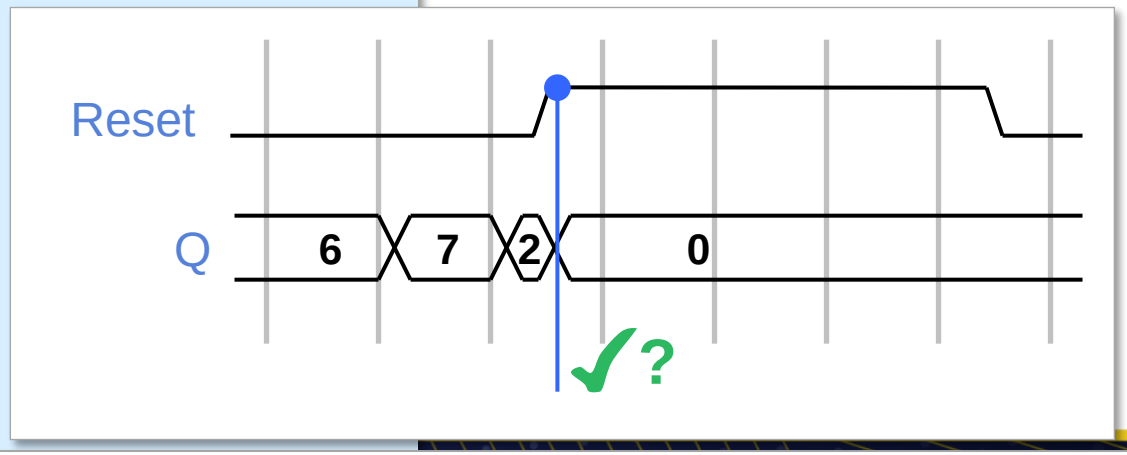
Asynchronous Assertions

```
module Counter ( input Clock, Reset, Enable,
Load, UpDn, input [7:0] Data, output [7:0] Q );

always @( posedge Reset or posedge Clock )
   if ( Reset )
      Q <= 0;
   else
      if ( Enable )
         if ( Load )
            Q <= Data;
         else
            if ( UpDn )
              Q <= Q + 1;
            else
              Q <= Q - 1;
endmodule
```

```
initial begin
   ... Reset = 1;
```

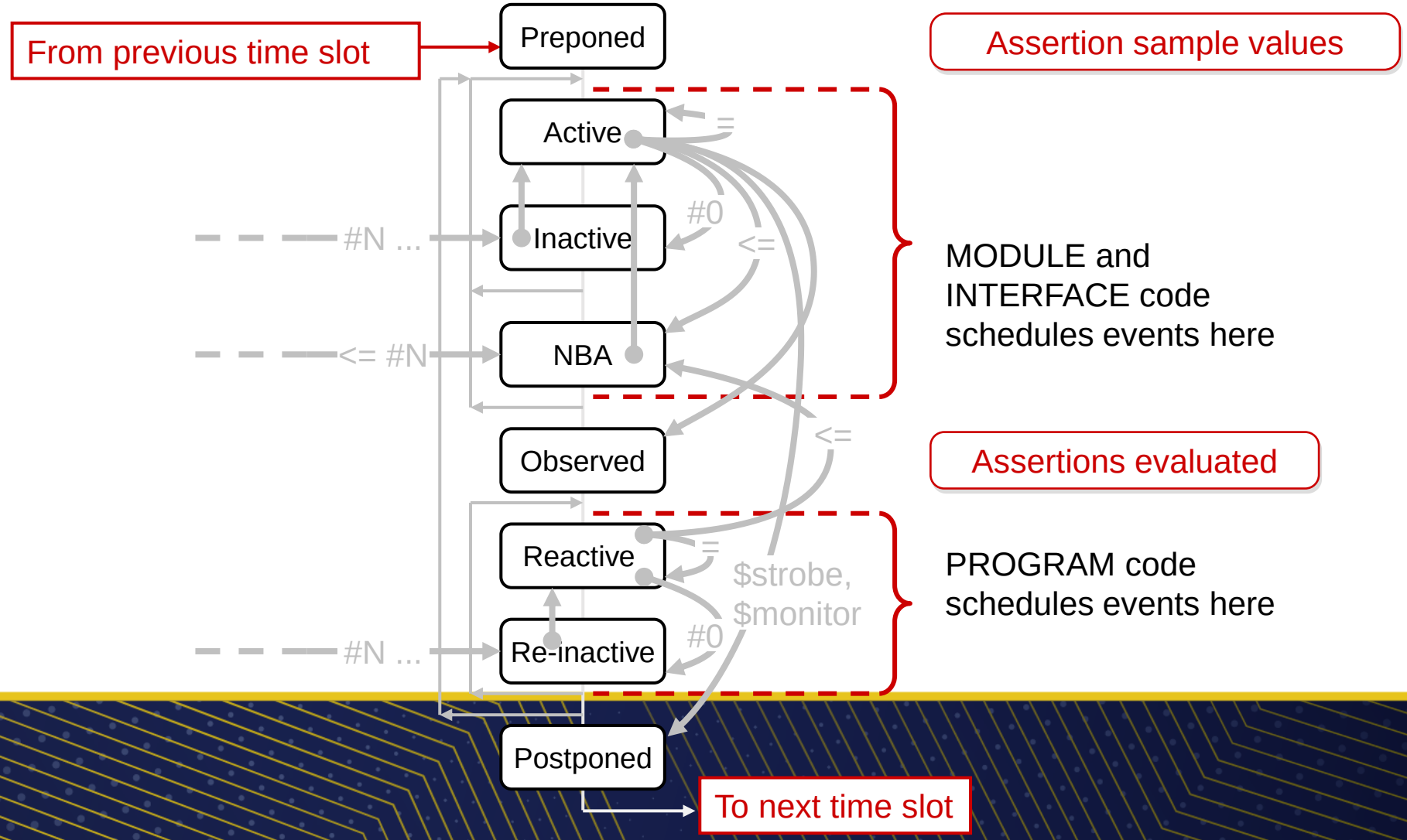
```
initial forever
   Clock = #5 ~Clock;
```



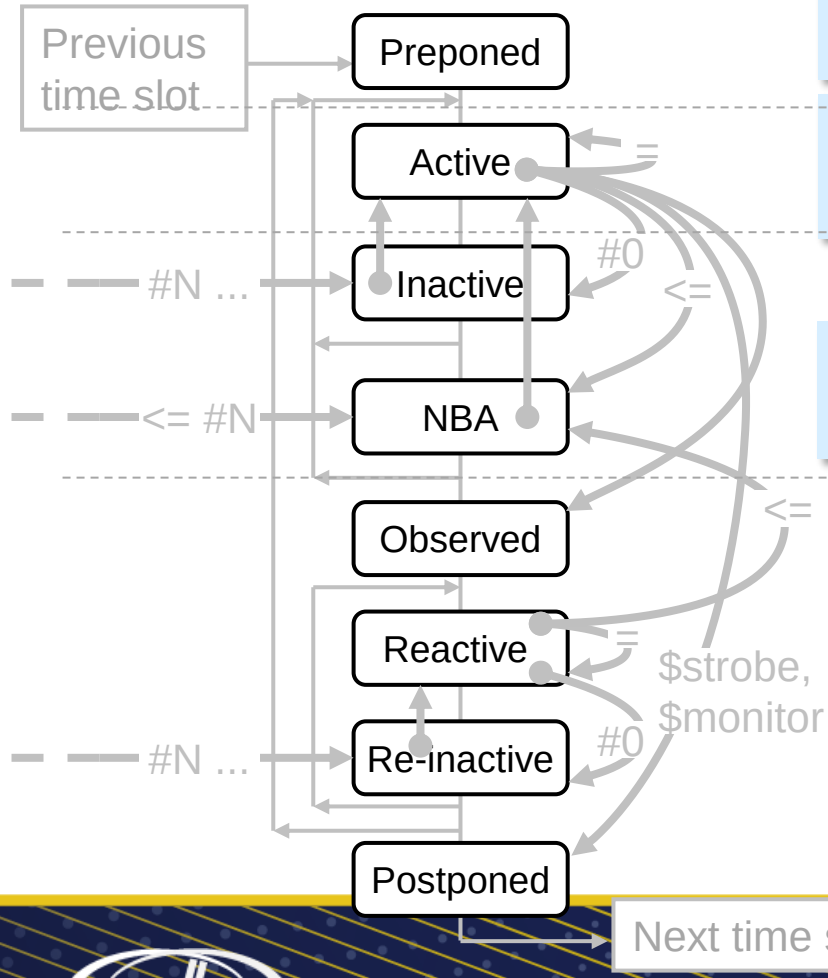

```
assert property ( @(posedge Reset) Q == 0 );
```

Will this work?

SystemVerilog Scheduler



Difficulty with Async Checking




```
assert property ( @(posedge Reset) Q == 0 );
```

```
initial begin
   ... Reset = 1;
```

```
initial forever
   Clock = #5 ~Clock;
```

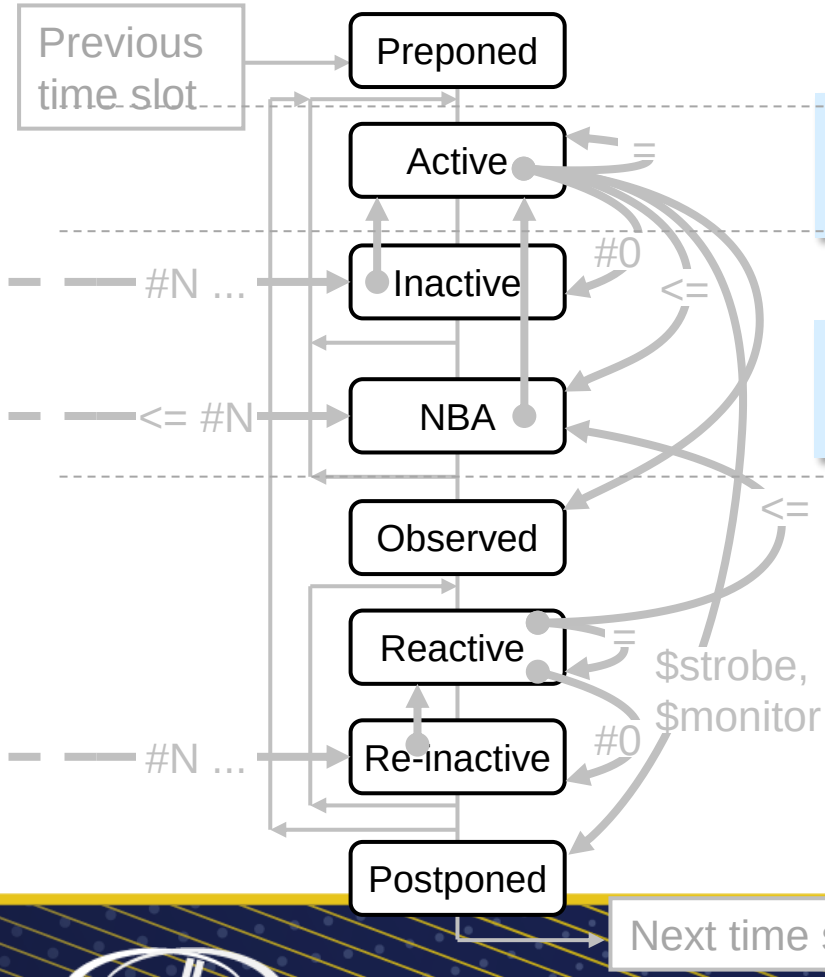
```
always @( posedge Reset ...)
   if ( Reset ) Q <= 0; ...
```

Reset	Q
0	Q_{prev}
1	Q_{prev}
1	0
1	0

RTL not updated yet!

accellera SYSTEMS INITIATIVE
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Difficulty with Async Checking




```
initial begin
   ... Reset = 1;
```

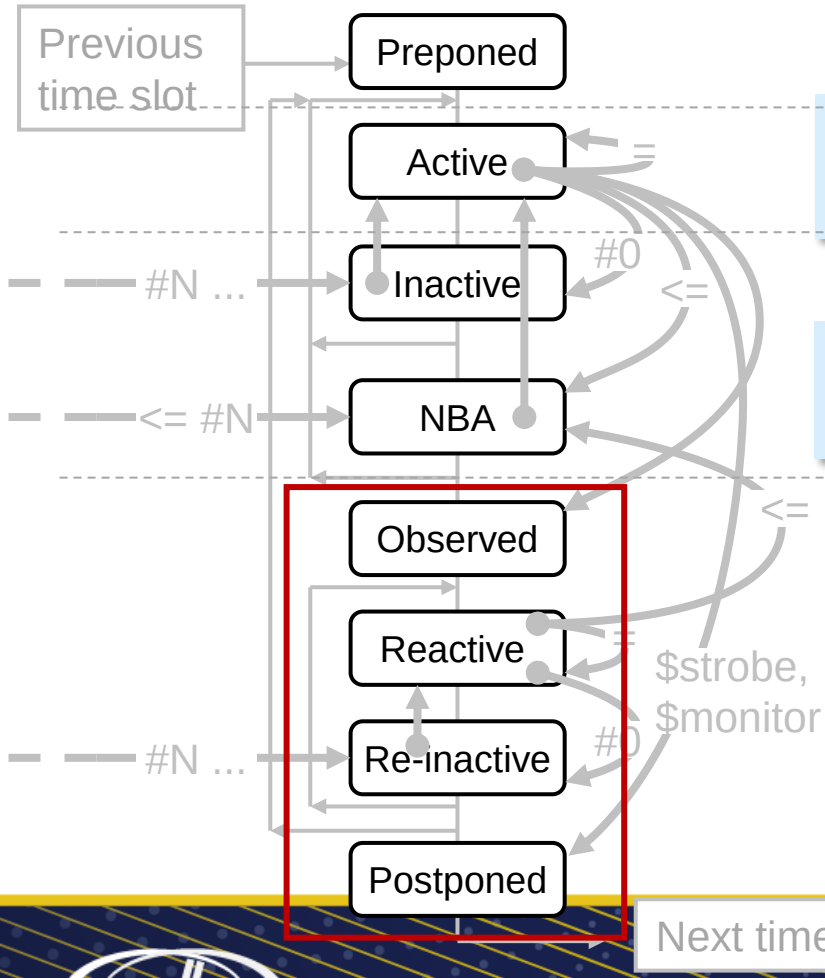
```
always @(posedge Reset)
   assert ( Q == 0 );
```

```
always @( posedge Reset ...)
   if ( Reset ) Q <= 0; ...
```

Reset	Q
0	Q_{prev}
1	Q_{prev}
1	0
1	0

RTL not updated yet!

Difficulty with Async Checking




```
initial begin
    ... Reset = 1;
```

```
always @( posedge Reset ...)
    if ( Reset ) Q <= 0; ...
```

Reset	Q
0	Q_{prev}
1	Q_{prev}
1	0
1	0

Checking needs delayed

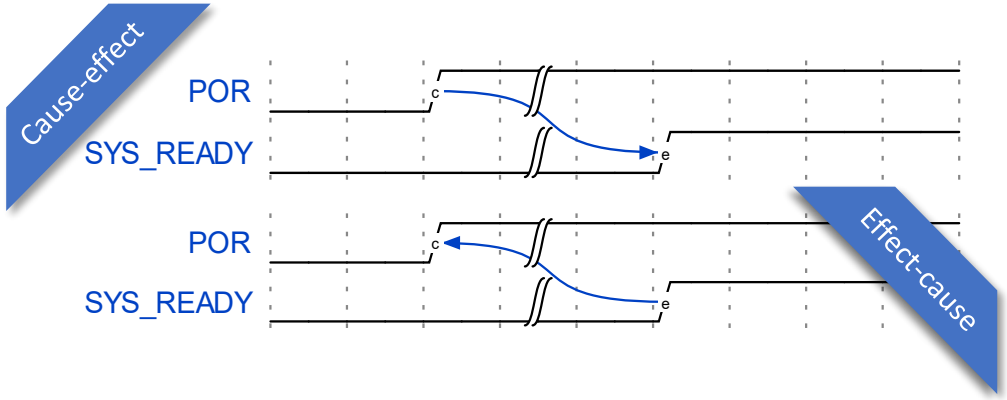
accellera SYSTEMS INITIATIVE
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Requirements for Async Assertions

Portable across simulators

Deterministic

Thorough – checks in both directions



In other words, ***practical asynchronous assertions***

See DVCon 2010 paper, "Asynchronous Behaviors Meet Their Match with SystemVerilog Assertions," for other solutions and handling async protocols

https://www.doulos.com/knowhow/systemverilog/asynchronous-behaviors-meet-their-match-with-systemverilog-assertions/

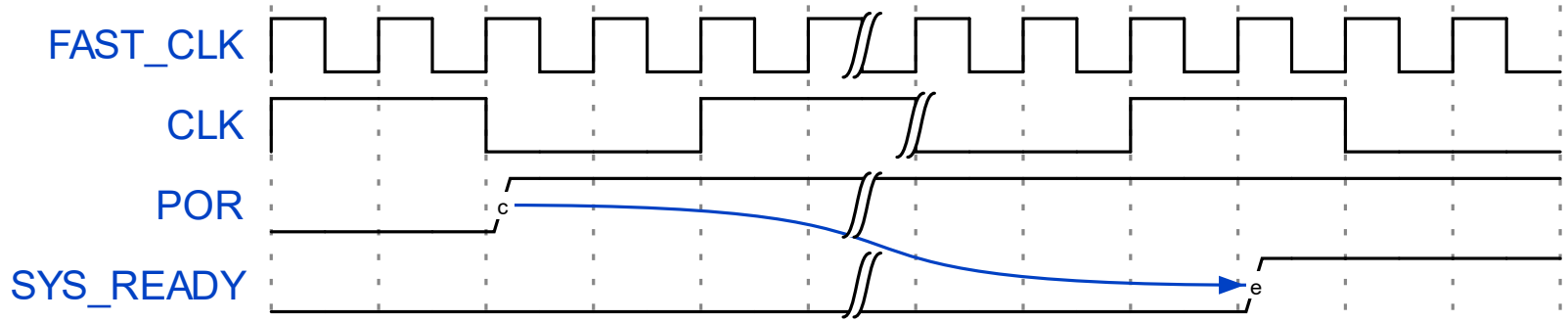
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
MARCH 4-7, 2024

Common Methods for Async Checking

accellera SYSTEMS INITIATIVE

Synchronous, oversampling, or fast clock




```
default clocking cb @(posedge CLOCK); endclocking
```

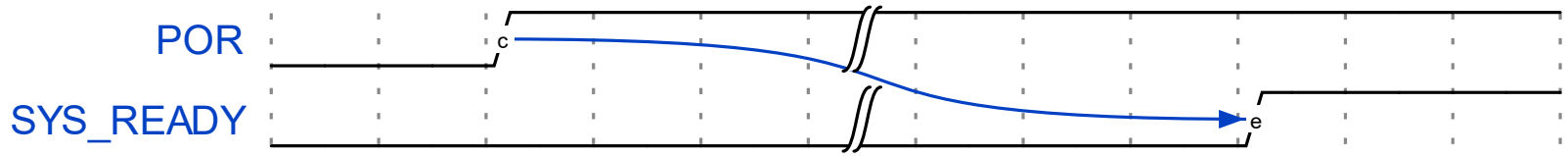
OR

```
default clocking cb @(posedge FAST_CLK); endclocking
```

```
assert property ( $rose(POR) |-> ##[0:$] SYS_READY );
```

What about glitches?

Event based methods




```
bit cover_por = 0;
cover property ( @(posedge POR) 1 ) cover_por = 1;
```

Then …

```
assert property ( @(posedge SYS_READY) cover_por );
```

What if SYS_READY never occurs?

Pros and Cons

Oversampling

- Pro – works with any verification flow (sim, emulation, formal, prototyping)
- Con – glitchy RTL behavior may go undetected

Coverage

- Pro – easy to write sophisticated scenarios
- Con – overlapping events undetected or event never occurs

Solution? Delay assertion checking …

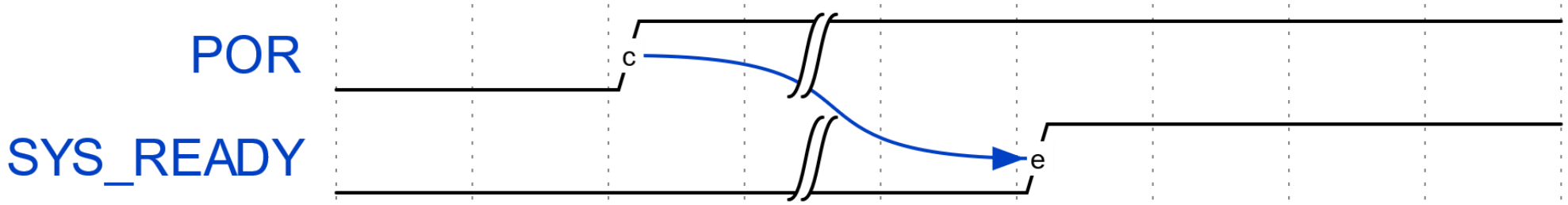
2024
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

Cause and Effect

accellera
SYSTEMS INITIATIVE

Scenario 1

Async signal causes another async event

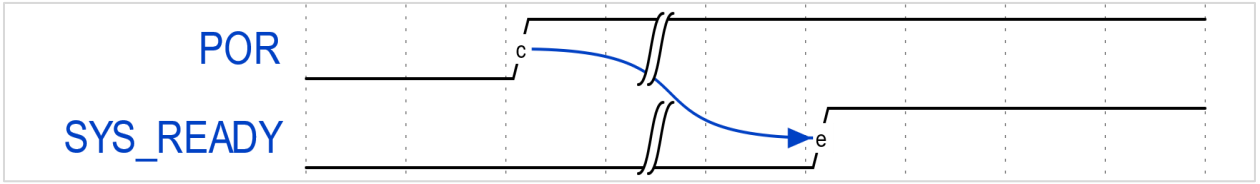



```
assert property ( @(posedge POR) 1 |-> @(posedge SYS_READY) 1 );
```

- This is a *weak* property
- Make it *strong*

```
assert property ( @(posedge POR) 1 |-> @(posedge SYS_READY) s_eventually(1) );
```

Coverage Alternative




```
bit coverage[string];
cover property ( @(posedge POR       ) 1) coverage["POR"       ]++;
cover property ( @(posedge SYS_READY) 1) coverage["SYS_READY"]++;
```

```
final begin
  if ( coverage["POR"] ) begin
     assert ( coverage["SYS_READY"] == coverage["POR"] ) else
        $error( ... );
  end
end
```

coverage[*]

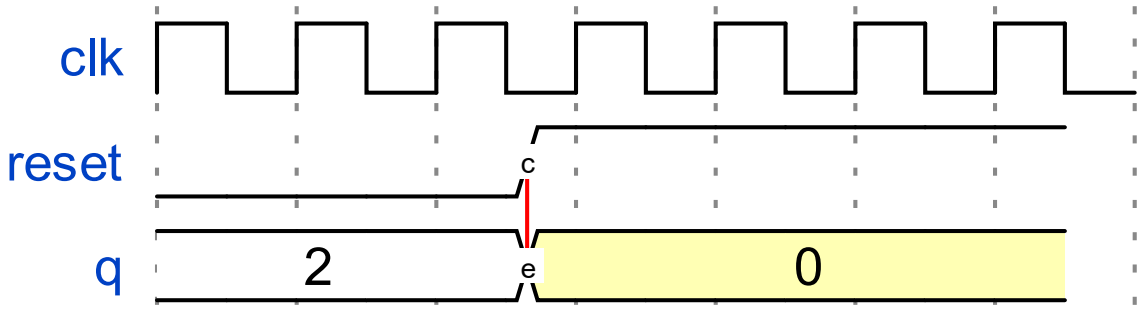
POR	SYS_READY
1	1

Expect a cause for each effect

accellera SYSTEMS INITIATIVE
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Scenario 2

Async signal causes RTL updates

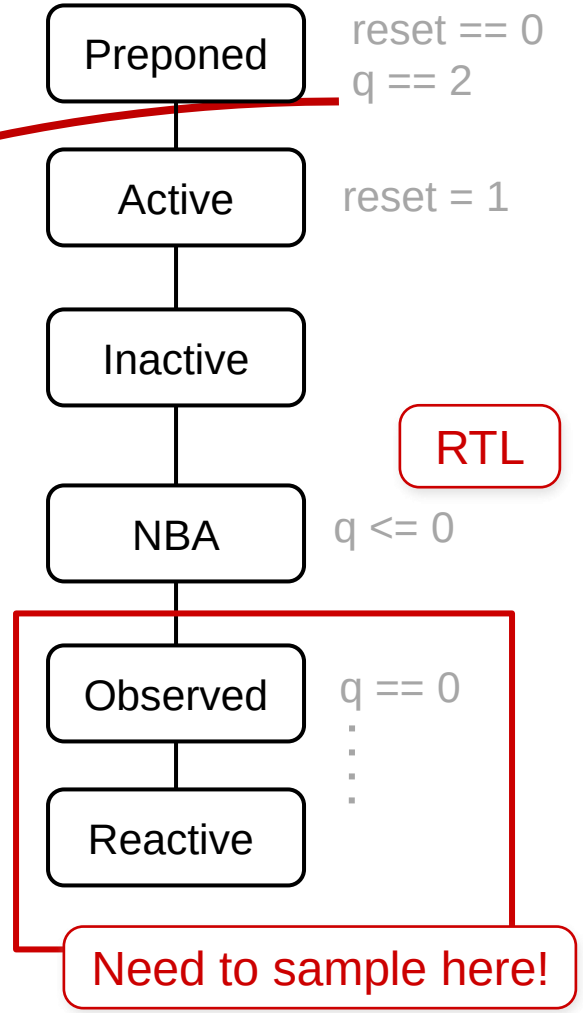



```
assert property ( @(posedge reset) q == 0 );
```

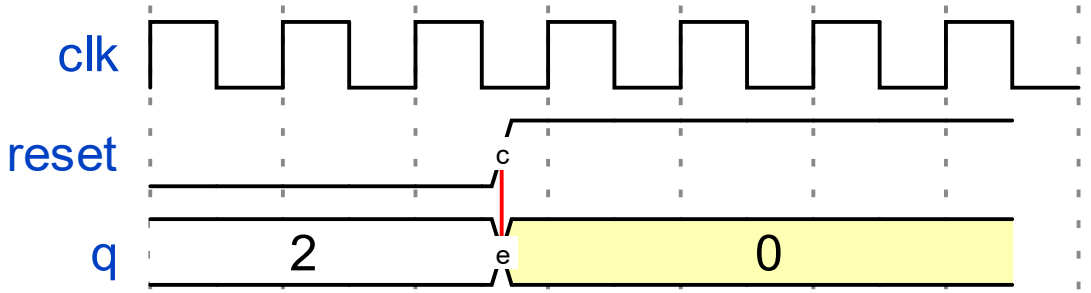
✗

Inputs from *Preponed*

q == 2



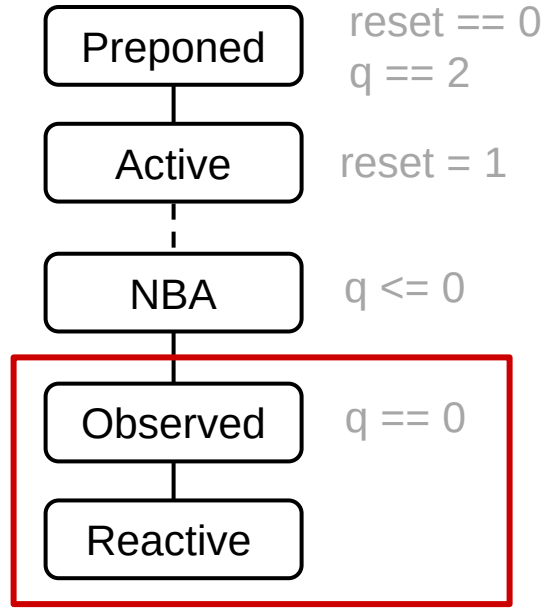
Program blocks



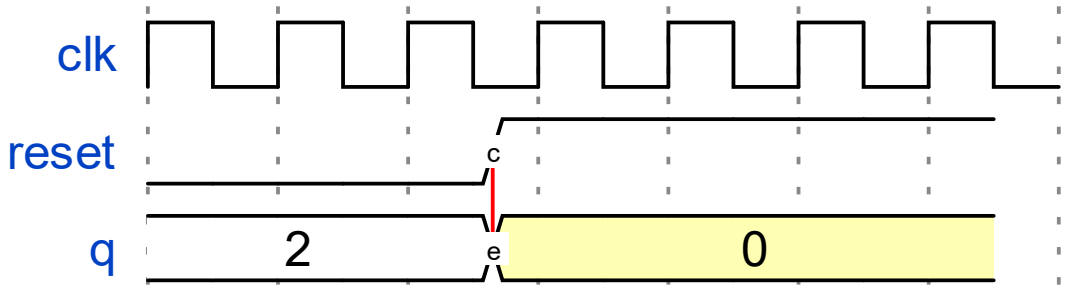
Inputs from *Observed*

```
program async_asserts;
   initial
      forever
        @(posedge reset)
            assert ( q == 0 );
endprogram
```

Scheduled in *Reactive*



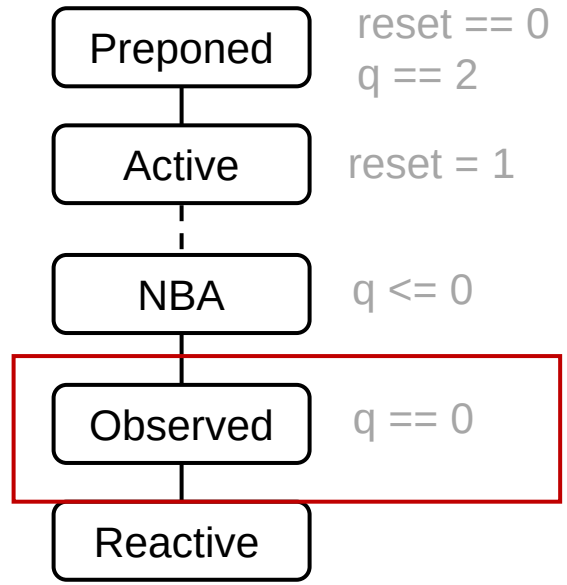
Sequence event



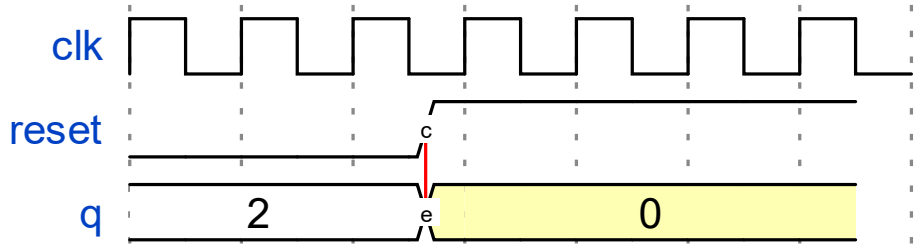

```
sequence seq_reset_event;
  @(posedge reset) 1;
endsequence

always
   @(seq_reset_event) assert ( q == 0 );
```

Sequence end point in *Observed*



Procedural concurrent assertions



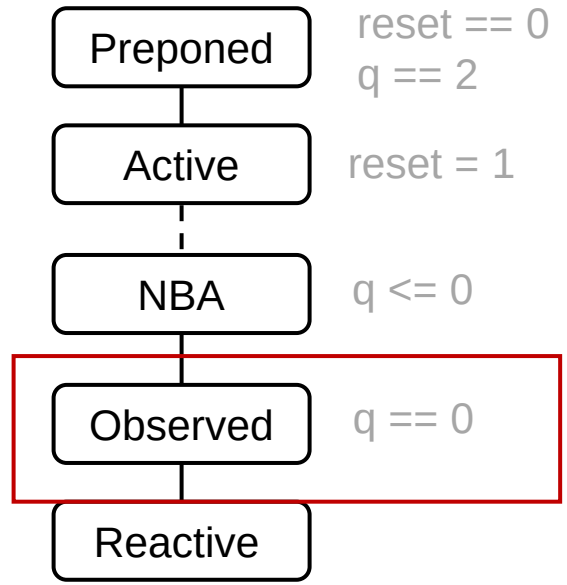
Procedural concurrent assertions mature in *Observed*

```
always @(posedge reset)
   assert property ( 1 )
      assert ( q == 0 );
```

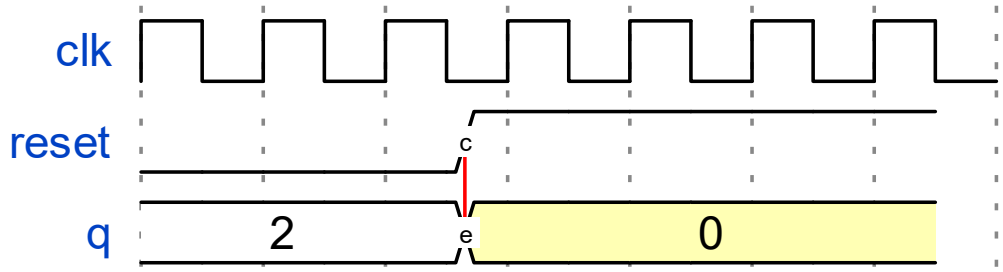
assert() executes in *Observed*

OR

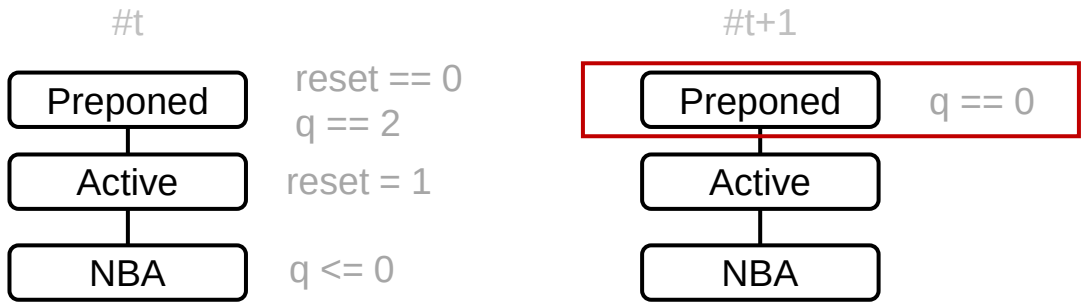
```
always_comb
   assert property ( @(posedge reset) 1 )
      assert ( q == 0 );
```



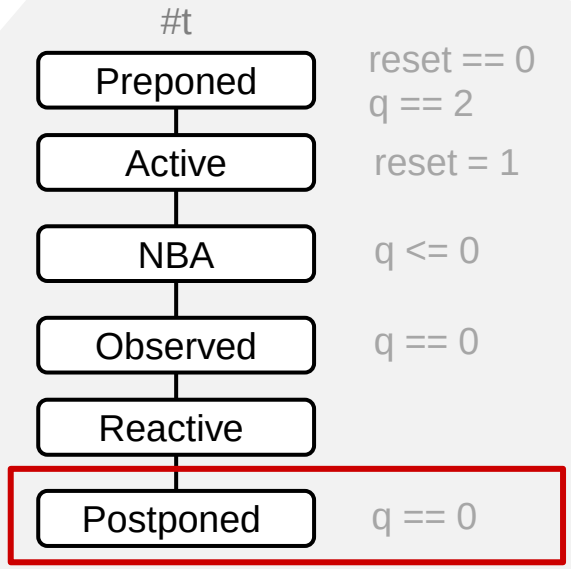
A timing delay




```
assign #1 delayed_reset = reset;
  assert property ( @(posedge delayed_reset) q == 0 );
```

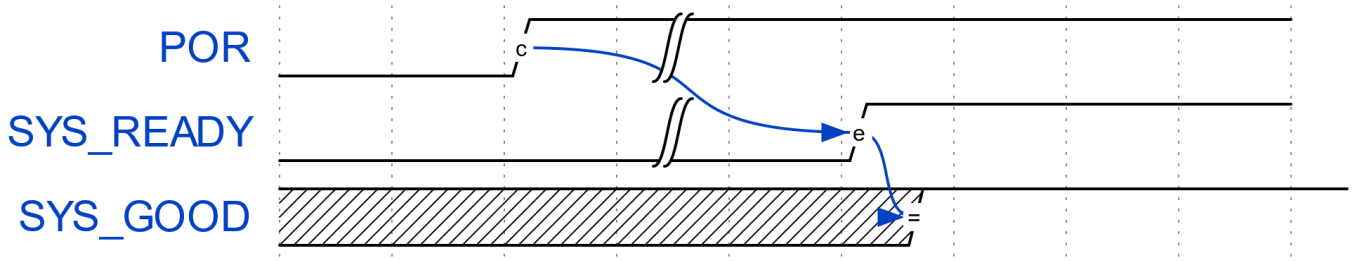



```
assign #1step delayed_reset = reset;
  assert property ( @(posedge delayed_reset) q == 0 );
```



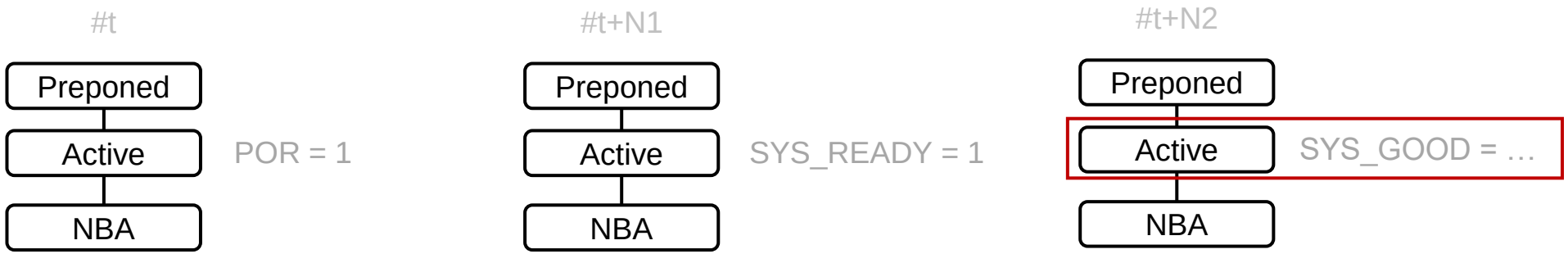
Scenario 3

Async event causes async event with updates

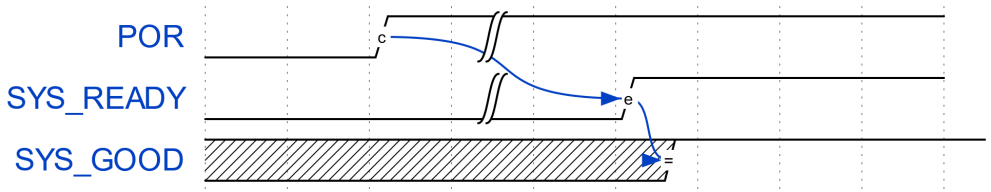


Multiclocked property

```
assert property (@(posedge POR) 1 |-> @(posedge SYS_READY) s_eventually(1)
                 |-> @(posedge SYS_GOOD) s_eventually(1));
```

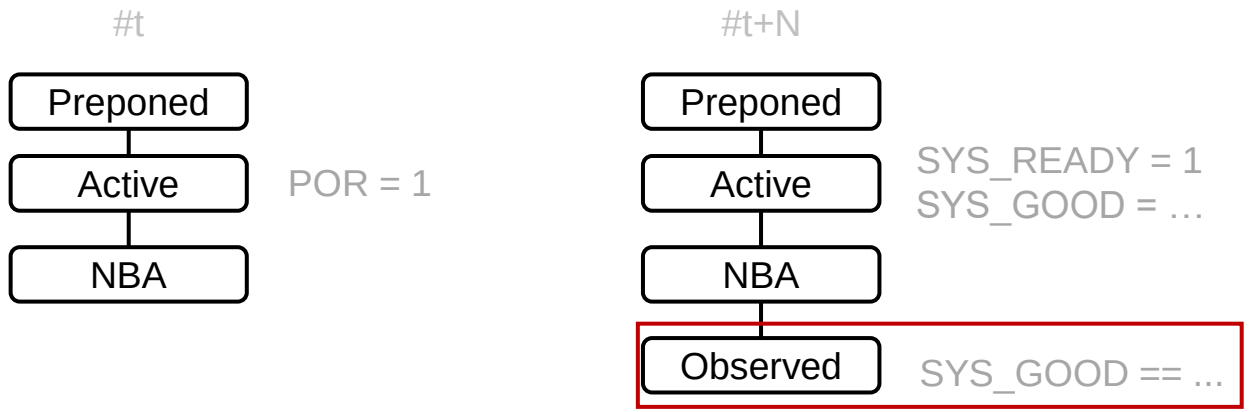


Overlapping behavior



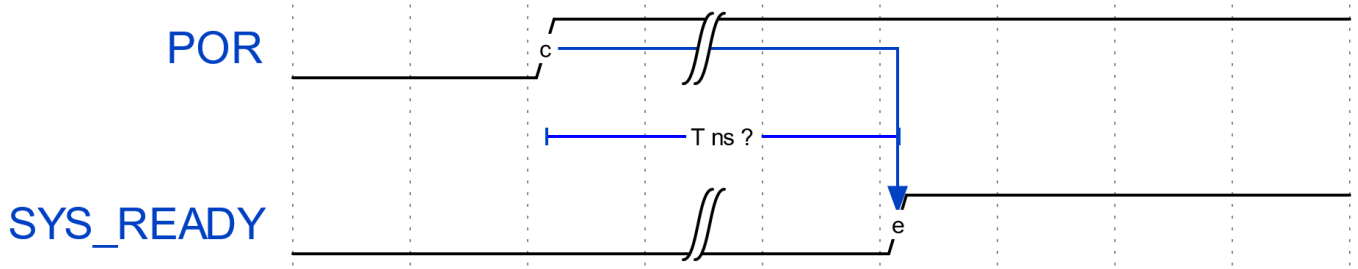

```
always_comb
  assert property ( @(posedge POR) 1 |-> @(posedge SYS_READY) s_eventually(1))
    assert(SYS_GOOD == ...);
```

assert() executes in *Observed*



Scenario 4

Async timing window




```
property prop_check_timing;
    realtime start;
    realtime finish;

    @(posedge POR)         (1, start  = $realtime) |->
    @(posedge SYS_READY)   (1, finish = $realtime) ##0
    (finish - start) == timing_window;
endproperty

assert property ( prop_check_timing );
```

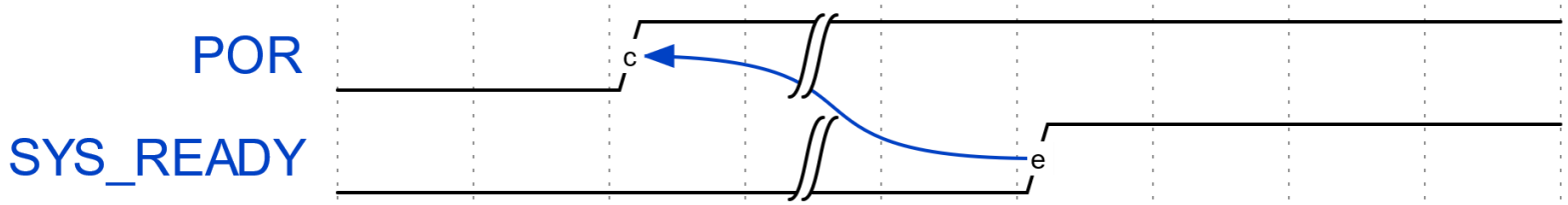
2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

The Effect and its Cause

accellera SYSTEMS INITIATIVE

Scenario 5

Async event caused by another event




```
bit cov_por;
cover  property ( @(posedge POR ) 1   ) cov_por = 1;
assert property ( @(posedge SYS_READY) cov_por );
```

OR

```
sequence seq_past_por;
    @(posedge POR) 1 ##1 @(posedge SYS_READY) 1;
endsequence

assert property ( @(posedge SYS_READY) seq_past_por.triggered );
```

Not as portable

Overlapping effect-cause

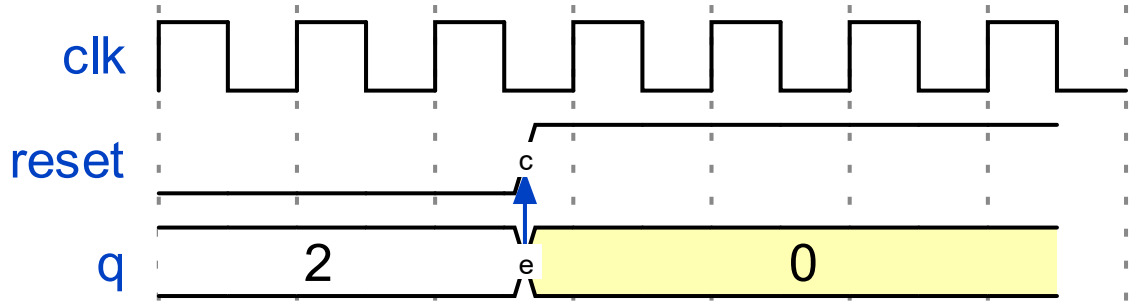

```
sequence seq_past_por;
    @(posedge POR) 1 ##0 @(posedge SYS_READY) 1;
endsequence

assert property ( @(posedge SYS_READY) seq_past_por.triggered );
```

Avoid - inconsistent support across tools

Scenario 6

RTL updated by async event

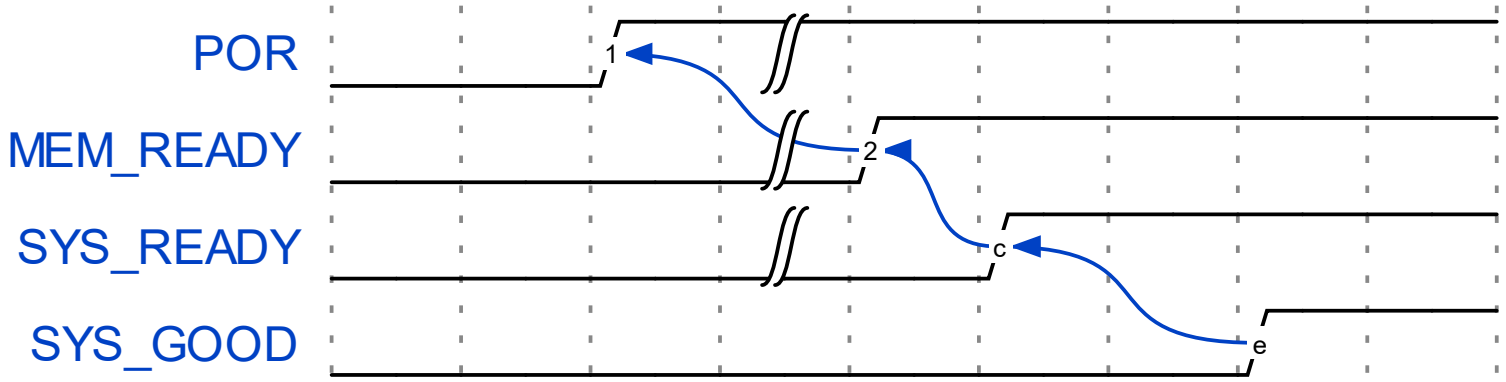



```
bit cov_reset;
always @(posedge reset) cov_reset = 1;

always_comb
    assert property ( @(q) 1 )
       assert ( cov_reset )
          cov_reset = 0;
       else $error ( … );
```

Scenario 7

Multiple causes for a sequence of events

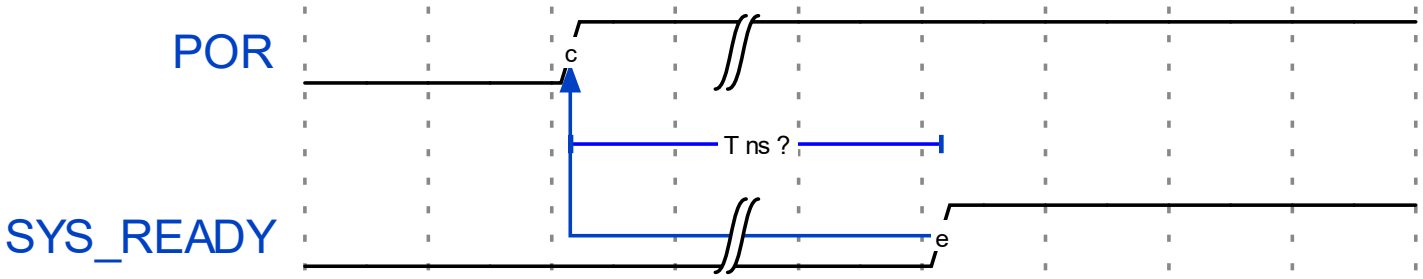



```
bit cov_por, cov_mem_ready, cov_sys_ready;
cover  property ( @(posedge POR)       1) cov_por = 1;
cover  property ( @(posedge MEM_READY) 1) cov_mem_ready = 1;
cover  property ( @(posedge SYS_READY) 1) cov_sys_ready = 1;

assert property ( @(posedge SYS_GOOD) cov_por &&
                                      cov_mem_ready &&
                                      cov_sys_ready );
```

Scenario 8

Async timing window effect-and-cause




```
bit cov_por;
realtime start;
cover property ( @(posedge POR) 1 ) begin
    cov_por = 1;
    start = $realtime;
end

assert property ( @(posedge SYS_READY) cov_por &&
                  (($realtime - start) <= timing_window ));
```

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

SAN JOSE, CA, USA
MARCH 4-7, 2024

Summary

accellera SYSTEMS INITIATIVE

Recommendations

Asynchronous bus protocols

- Use multi-clocked properties (usually straightforward)

Asynchronous controls

- Oversampling generally good enough
- Coverage approach works in most cases (plus bonus of functional coverage)
- Other scenarios, find a way to delay the checker’s sampling

2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES
SAN JOSE, CA, USA
MARCH 4-7, 2024

Questions?

Examples available at: https://edaplayground.com/x/qB72

accellera SYSTEMS INITIATIVE