2025
DESIGN AND VERIFICATION™
DVCON
CONFERENCE AND EXHIBITION
UNITED STATES
SAN JOSE, CA, USA
FEBRUARY 24-27, 2025

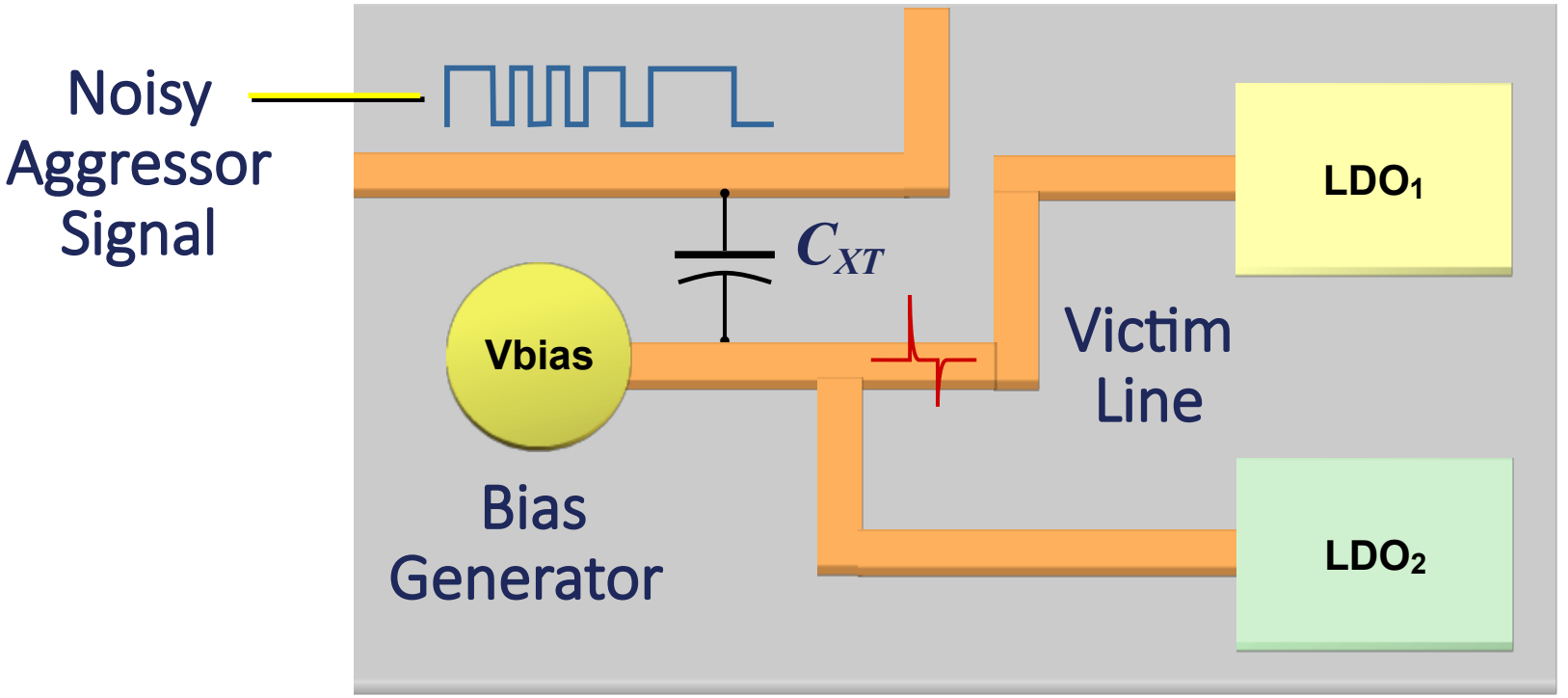
Catching the Elusive Voltage Spike with Analog/Mixed-Signal SVA/PSL Assertions

Charles Dančak, Betasoft Consulting, Santa Clara

Betasoft
Consulting

accellera
SYSTEMS INITIATIVE

On-Chip Bias Line Subject to Spikes

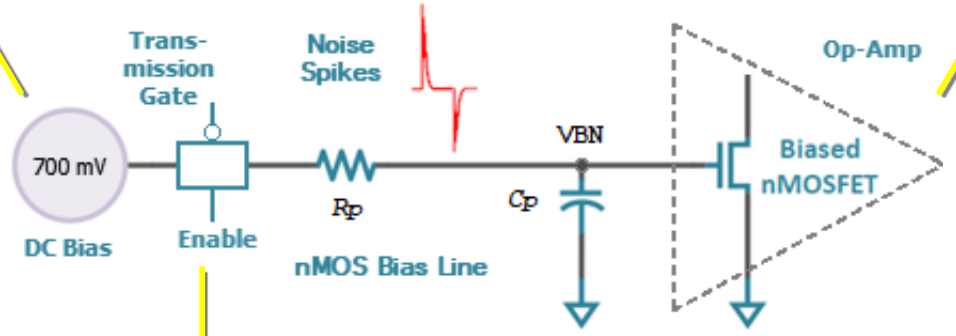


Long Wires
Prone to
Crosstalk

- Spikes affecting a bias voltage can arise from excess crosstalk or noise bursts.
- Even with careful routing and shielding, such glitches can lead to malfunction.

Simplified Model of Bias Line

*n*MOS Bias Generator



LDO Regulator

Disconnect Bias in Power-Down Mode

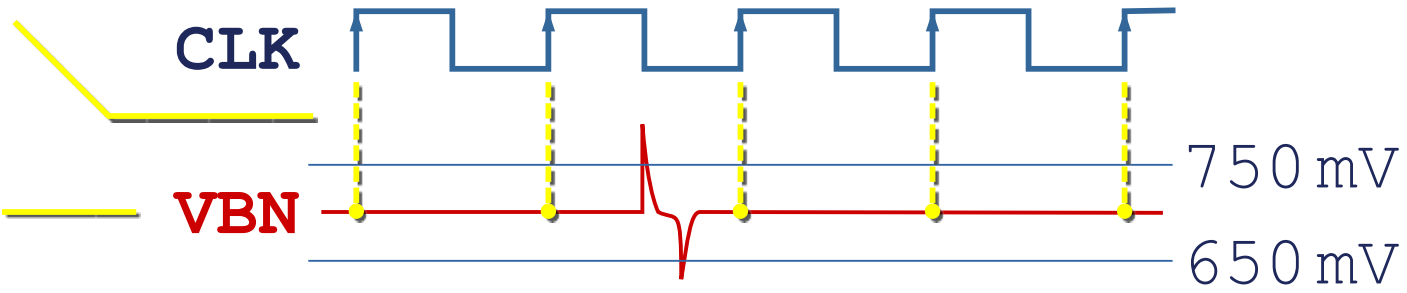
△ The Problem:
Anomalous signal behavior between clock edges won't be detected by an unaided SystemVerilog assertion.

- Bias line **VBN** drives the *n*MOS bias pin on a typical two-stage CMOS op-amp.
- Transmission gate was open during **PWR_DN**, but is re-enabled in **RESUME**.
- Bias level **VBN** will then rise exponentially to its valid range: 700 mV ± 50.

Spikes Can Elude an Unaided Assertion

Evaluate at Clock Ticks

Bias Should Be Stable



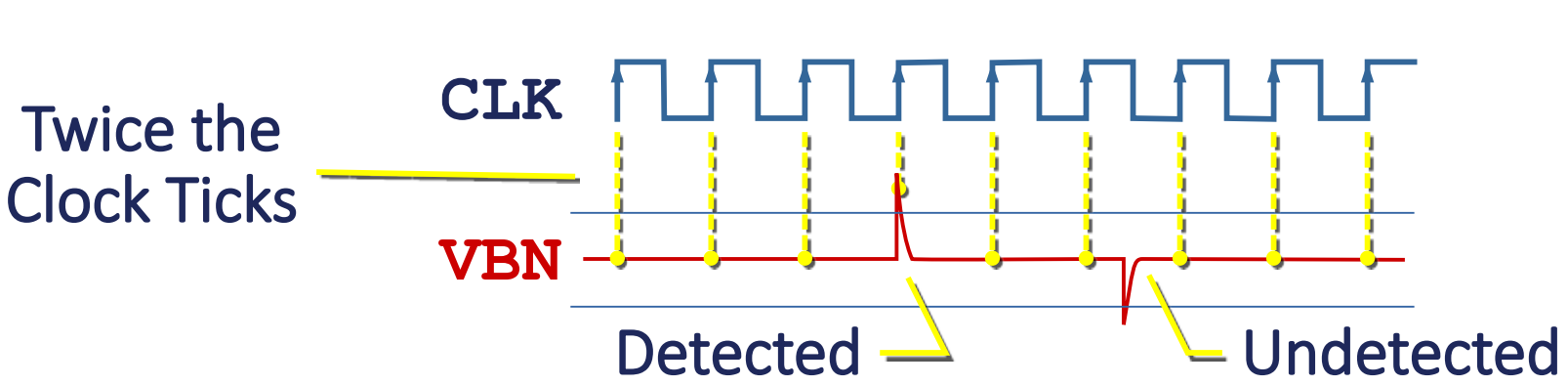
Default Clocking: `@(posedge CLK)`

SystemVerilog Assertion

```
//Check whether bias value VBN stays in range:
  VBN_VALUE_chk:
  assert property((VBN >= 0.650) && (VBN <= 0.750));
```

- These spikes on **VBN** are so narrow they fall in between assertion clock edges.
- A concurrent assertion is evaluated only at clock ticks [SVA Handbook, §2.3].
- Assertion **VBN_VALUE_chk** thus passes blindly—missing the spikes entirely.

Throwing More Points at the Problem



△ A Trade-Off:
As clock period goes from ns to ps to fs, simulation is slower.

```
//Check whether bias value VBN stays in range:
  VBN_VALUE_chk:
  assert property((VBN >= 0.650) && (VBN <= 0.750));
```

Discrete-Time Values of **VBN**

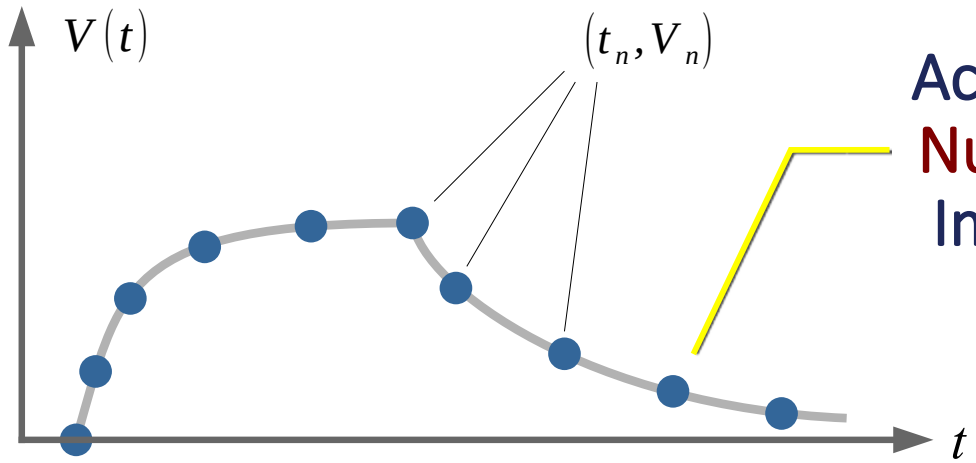
- By increasing the assertion clock rate, we evaluate bias **VBN** more frequently.
- First spike is successfully detected, and assertion **VBN_VALUE_chk** will fail.
- But there is no guarantee of catching a second spike that is yet more narrow.

Paradigm 1: Discrete Time-Value Pairs

A Real-Number Waveform

$$V(t) = \{ (t_1, V_1), (t_2, V_2), \ldots \}$$

Mathematical Time Sequence of Points



Accuracy Requires Numerous Points, Impacting Speed

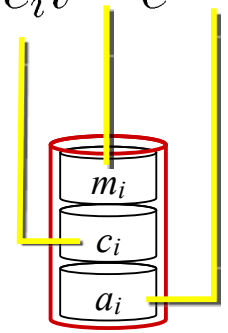
- Modeling analog waveforms using time-value pairs is inherently point-based.
- Finite time interval between two closely-spaced points can still conceal a glitch.
- Unaided assertion like **`VBN_VALUE_chk`** may thus blindly pass a narrow spike.

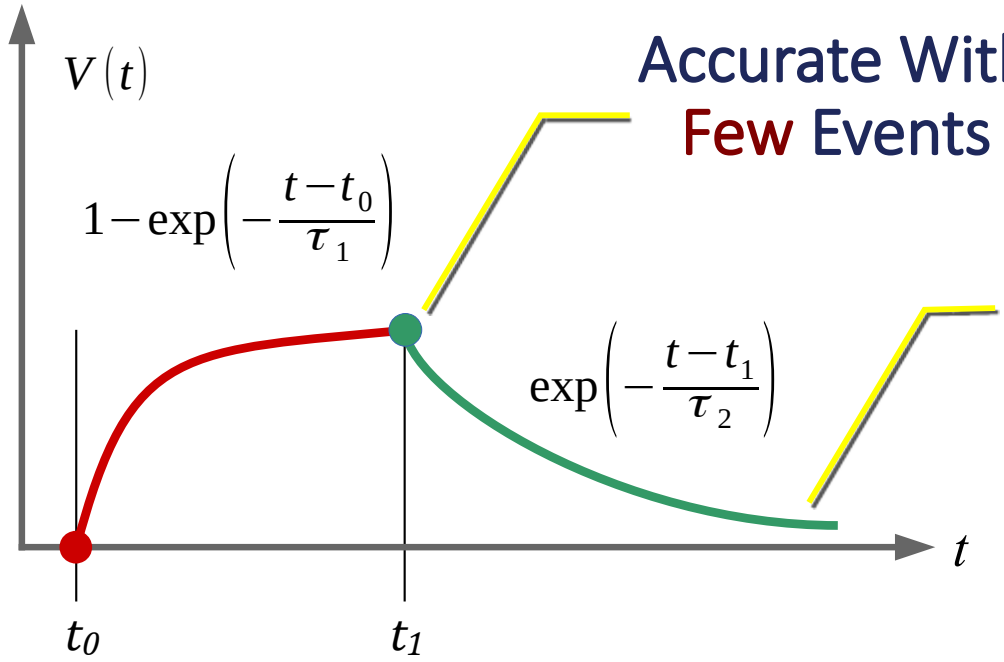
Paradigm 2: Continuous-Time Monitoring

XMODEL Waveform
(Type **`xreal`**)

$$V(t) = \sum_i c_i t^{m_i} e^{-a_i t}$$

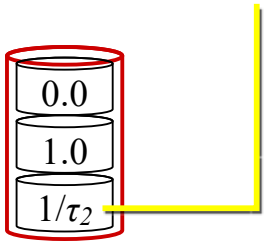
Simplified
`struct`





Accurate With
Few Events

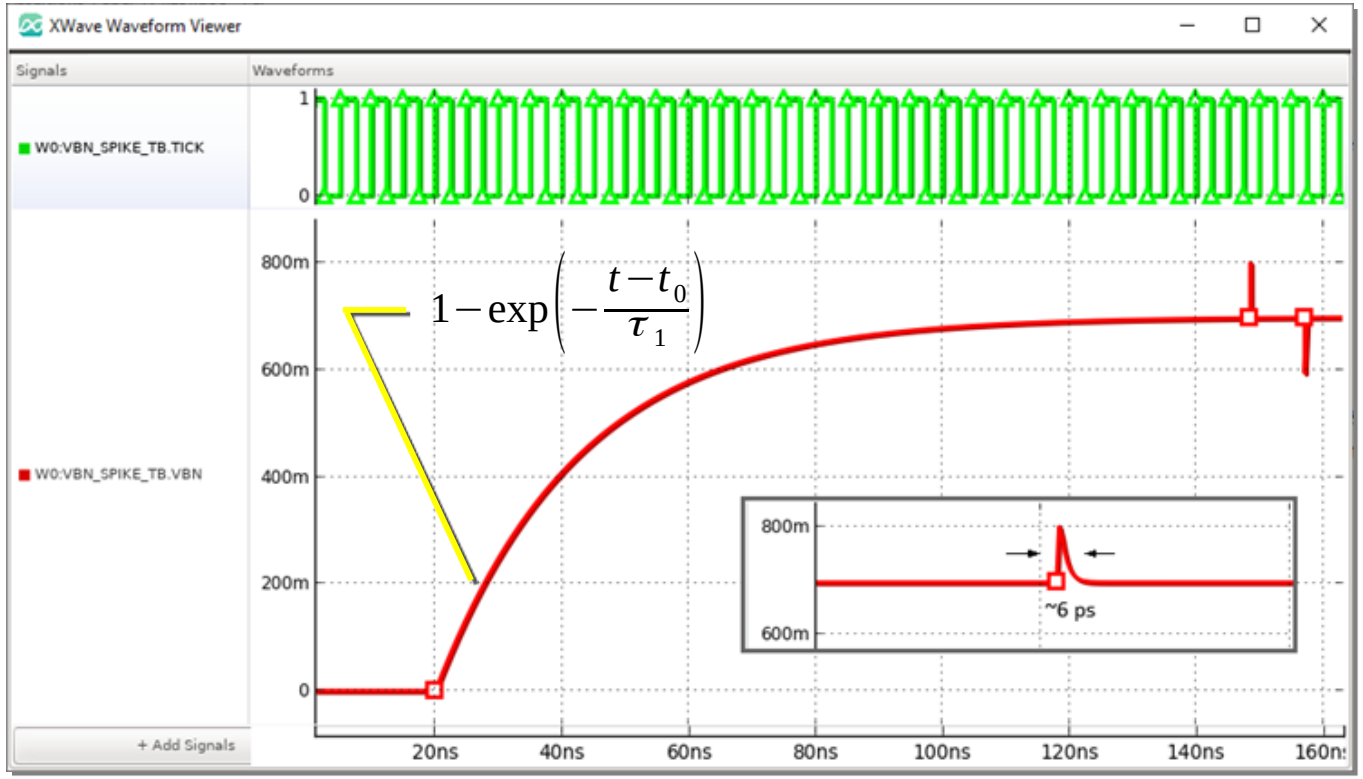
Piecewise-Functional
Decaying Exponential
(Fall Time τ_2)



- Consistent spike detection demands a departure from a point-based paradigm.
- But how do we monitor **`VBN`** continuously in time—not just at discrete points?
- Paradigm 2: Represent **`VBN`** as an analytic function, everywhere differentiable.

One Event per Analytic Expression

`xreal`
Waveform
Rendering



Simulation
Event
Markers

- The XWAVE viewer shows **`xreal VBN`** signal, with event markers enabled.
- With far fewer events, Xcelium will run at normal logic-simulation speeds.

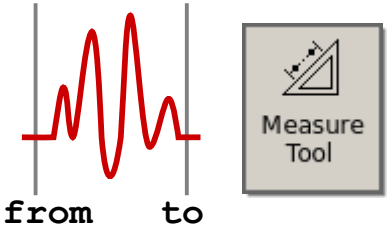
Solution: Measure the Analytic Waveform

XMODEL Measuring Elements

~/LDO/150

```
//Measure max, min between successive TICK edges:
  meas_max XP_MAX(.in(VBN), .out(VBN_max),
                  .from(TICK_xb), .to(TICK_xb)
  );
  meas_min XP_MIN(.in(VBN), .out(VBN_min),
                  .from(TICK_xb), .to(TICK_xb)
  );
//Boolean condition for bias in range:
  let VBN_VALID = (
    (VBN_min >= 0.650) && (VBN_max <= 0.750)
  );
```

real Outputs



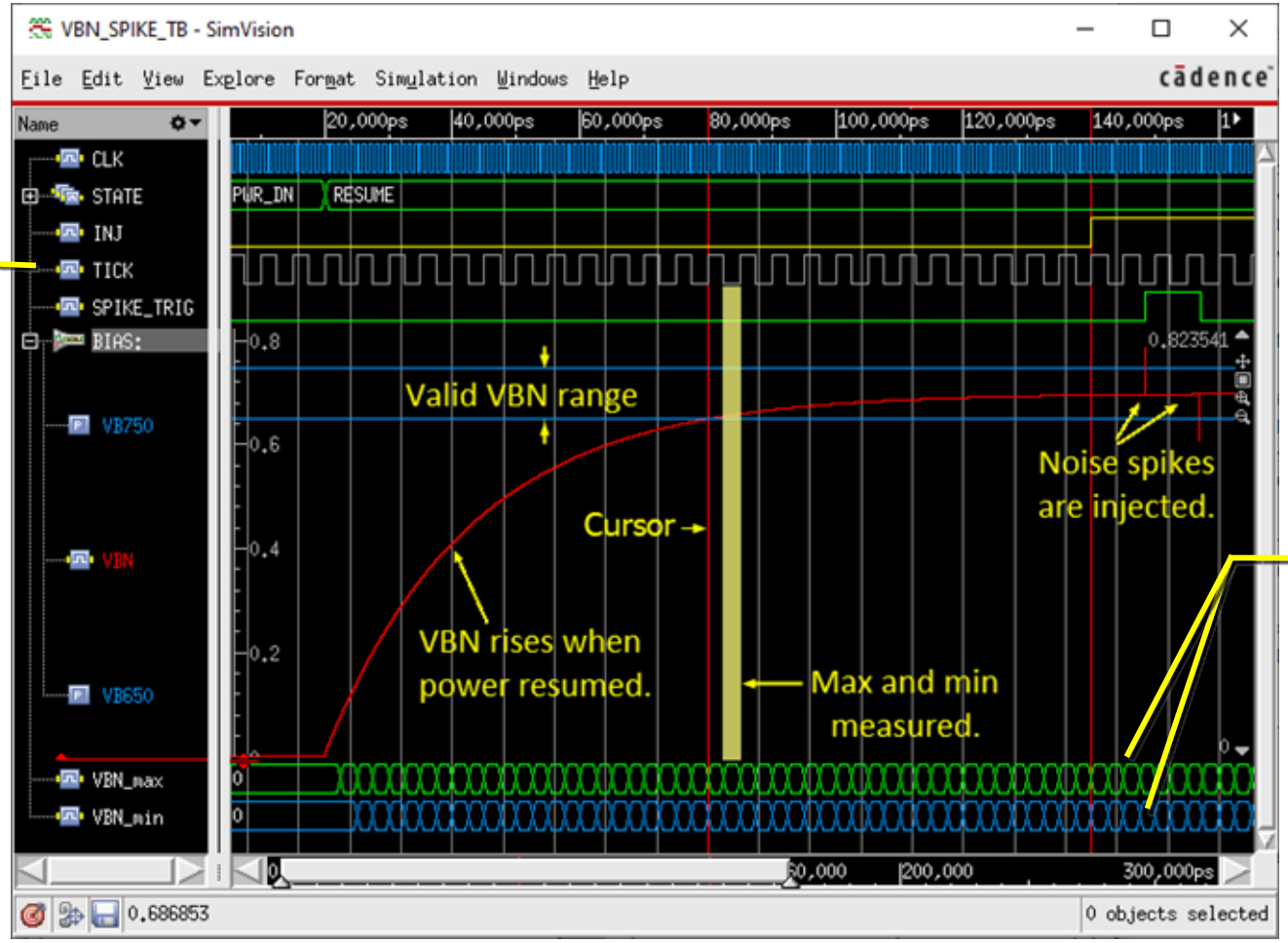
Referenced in SVA Code

- Analytic functions are differentiable, allowing many waveform measurements.
- Library element **meas_max** finds input signal's peak over some time window.
- From the analytic time-domain expression, it can compute the first derivative.
- Extracts, from list of falling zero-crossings, the highest peak inside the window.

Make Continuous-Time Measurements

Successive Triggering **TICK** Edges

Triggered Elements



VBN Waveform (**xreal**/**real**)

Measured Max, Min Each **TICK**

True Measured **VBN** Extrema

Assert that VBN Stay in Range

SVA Sequence

Concurrent Property

XMODEL -Aided Assertion

~/LDO/150

```
//Time window to check VBN_VALID:
  sequence WINDOW_seq;
    $rose(VBN_VALID) ##1 VBN_VALID[+] ##1 $fell(STATE == RESUME);
  endsequence: WINDOW_seq

//VBN shall remain valid during window:
  property VBN_STABLE_pro;
  //Activate when bias enters its range:
   (STATE == RESUME) && $rose(VBN_VALID) |-> //Antecedent clause.
     (VBN_VALID throughout WINDOW_seq);      //Consequent clause.
  endproperty: VBN_STABLE_pro

//Assert property VBN_STABLE_pro:
  VBN_STABLE_chk:
  assert property(VBN_STABLE_pro)
    $info("VBN_STABLE passing ... );
  else begin
    ++FAILURES; //Failure count.
    $error("VBN_STABLE failing ...);
  end
```

True Measured Extrema

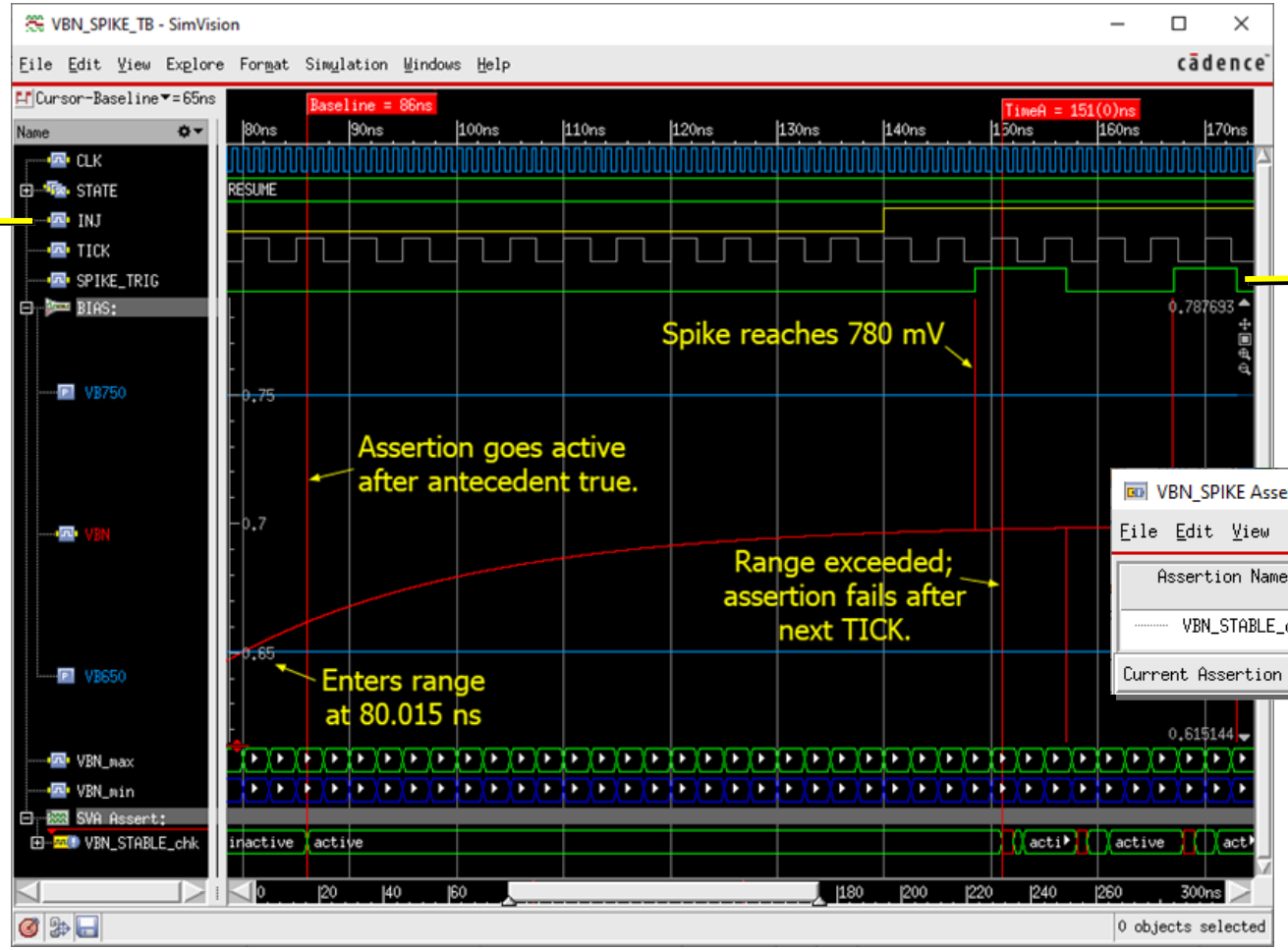
```
//Condition with meas_max/min:
  let VBN_VALID =
    ((VBN_min >= 0.650) && (VBN_max <= 0.750));
```

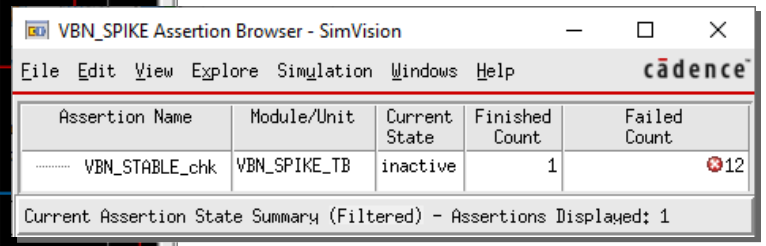
Aided Assertion Detects Every Spike

Enable Spike Injection

Up or Down Spike Trigger

Testbench Instrumentation

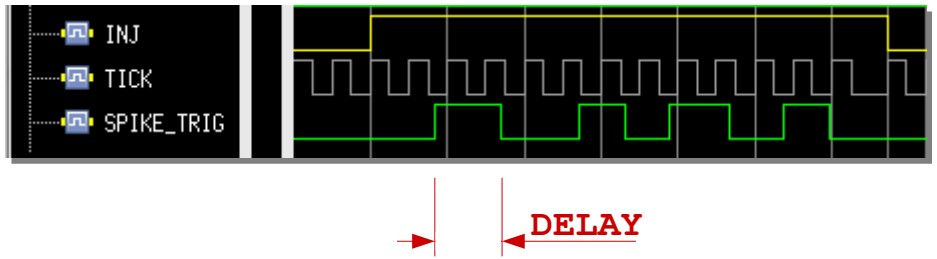


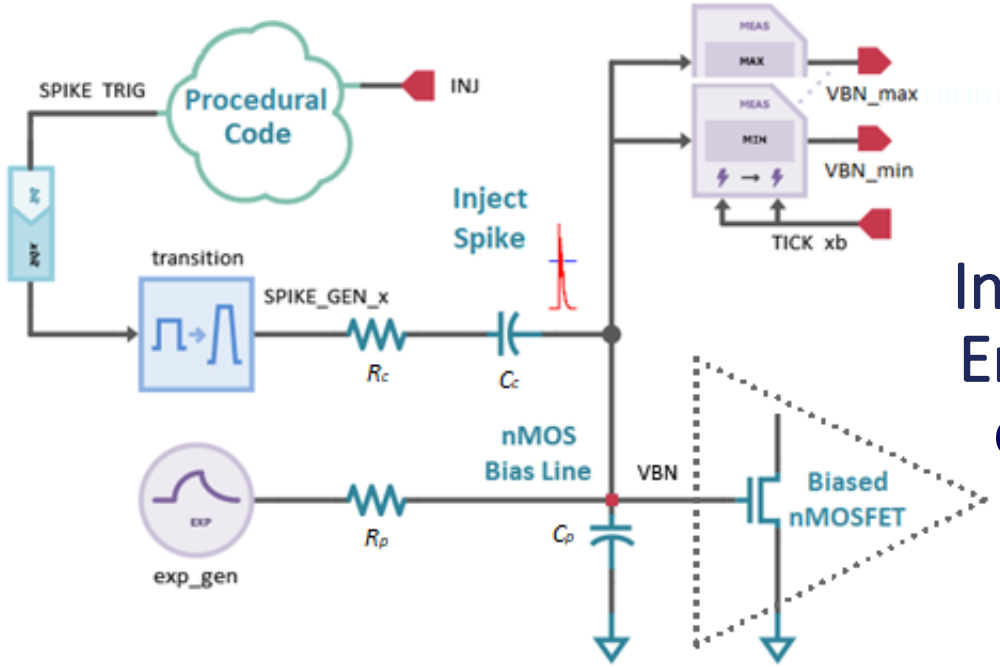


VBN_STABLE_chk Failures (Total: 12)

Injecting Spikes onto VBN


```
while (INJ) begin //Procedural loop.
//Control density of random spikes:
  DELAY = $urandom_range(10000, 5000);
//Delay next edge, in ps units:
  #(DELAY) SPIKE_TRIG <= ~SPIKE_TRIG;
end
```





Injected Spikes Emulate Noise on Bias Line

- Need some means of injecting spikes, analogous to a lab waveform generator.
- Injection subcircuit built from procedural code and XMODEL library elements.
- Resistors, capacitors chosen to yield narrow spikes, several picoseconds wide.

Conclusions & Questions

- Can we detect noise spikes, picoseconds wide, between clock edges?
- Yes. Assertions aided by XMODEL can catch voltage spikes and glitches.
- Elements like **`meas_max`** continuously monitor signal maxima over a time interval—without the need to sample values more frequently.
- Testbench caught every spike on bias line **`VBN`**, with no speed penalty.