Verilog-AMS Modeling Best Practices: Missing Events & Transitions

Ron Vogelsong

February 3, 2015

Topics

- Analog Expressions & Analog/Discrete Conversion
- Transition Function Usage
- Using @cross and @above Events
- Using @absdelta Events
- Interaction of Analog & Discrete Solvers
- Discussion

Proper analog expression development

Analog expressions can be divided into three categories:

1. **Continuous**
   - No step changes in value
   - Dependent on only continuous variables
     voltage, current, time, transition()
2. **Discrete**
   - Value is constant most of the time
   - Steps to a new value occasionally
     @(cross()), @(timer()), if/case selections of constants, dependence on integer or any discrete variables
3. **Mixed** *(always a bad idea!)*
   - Value is sometimes continuously changing, other times steps

Most analog issues are caused by not comprehending the difference between continuous & discrete signals!

AMS Analog/Discrete Conversion Basics

– Discrete signals should be converted to a time-continuous format prior to driving analog nodes:

```
analog begin
   Aval = transition(Dval, 0, tr, tf);
   . . . .
```

– Analog continuous signals must be sampled to move to discrete domain:

```
always @(cross(Aval-0.5, 0, ttol, vtol))  Dreg=(Aval>0.5);
always  #Td  Dval=Aval;
always @(posedge CK)  Dval=Aval;
always @(absdelta(Aval, vdelta, ttol, vtol))  Dval=Aval;
```

– Discrete simulator assumes all analog real variables are continuous (it always linearly interpolates between analog timepoints)

Transition Function Usage

- Never attempt to use transition on a continuous time argument

```
V(out) <+ transition( V(vdd)*pstate, td,tr,tf );
```

- Separate continuous part from discrete portion for proper operation

```
V(out) <+ V(vdd)*transition( pstate, td, tr, tf );
```

- Don't force extra-small analog timesteps

```
V(out) <+ V(vdd)*transition( pstate, 1p, 1p, 1p );
```

- Use zero delay unless it's important, and reasonable rise/fall times

```
V(out) <+ V(vdd)*transition( pstate, 0, 100p, 100p, 10p);
```

- Optional 5th argument is time tolerance: limits minimum forced timestep when many transitions occur near same timepoint
- Default is transient parameter **transres** (defaults to 1e-9*stoptime)

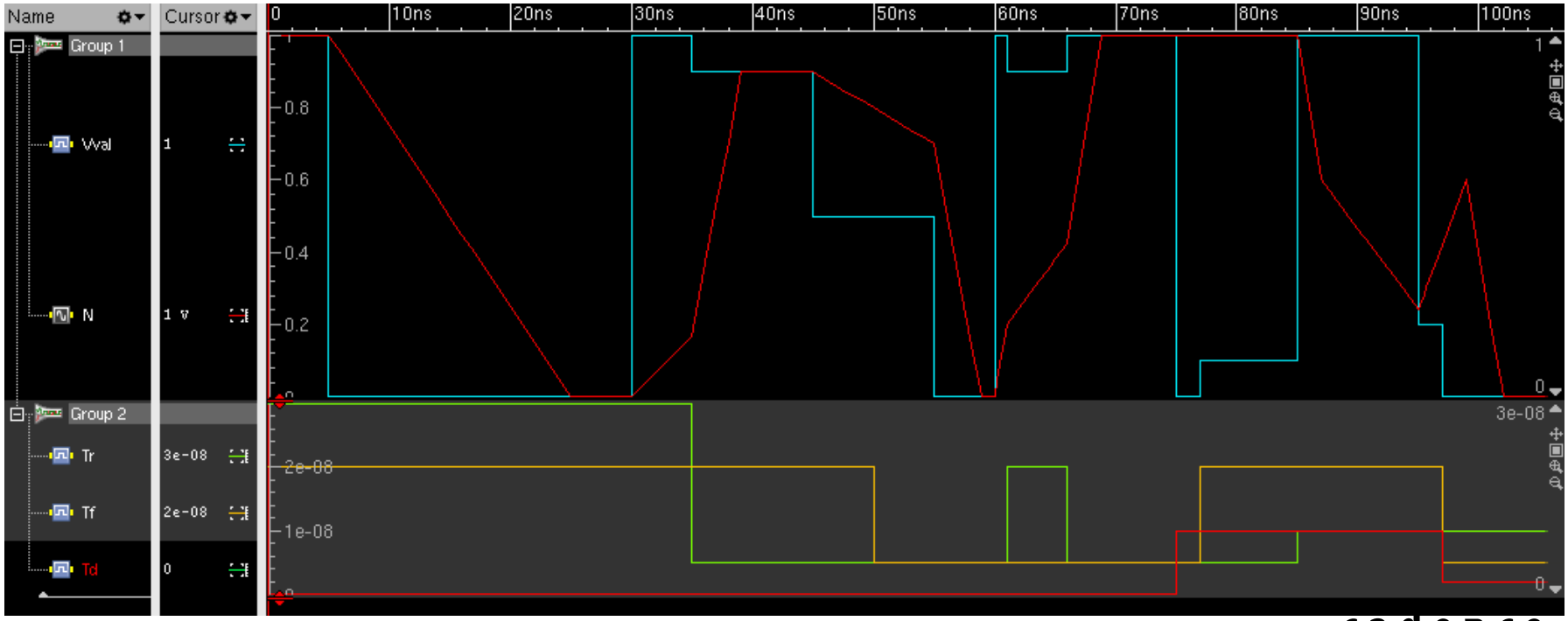
Note: transres must be smaller than any rise/fall times
– else it can lead to skipped event timepoints!

Overlapping Transition Operations

- Transition function declares how to ramp to new value

```
V(N) <+ transition( Vval, Td, Tr, Tf );
```

- If Vval changes mid-transition, it updates future waveform
  - Selects new ramp rate toward Vval, arrives in <= Tr or Tf seconds
  - If Td>0 at change, it will wait Td before modifying ramp rate

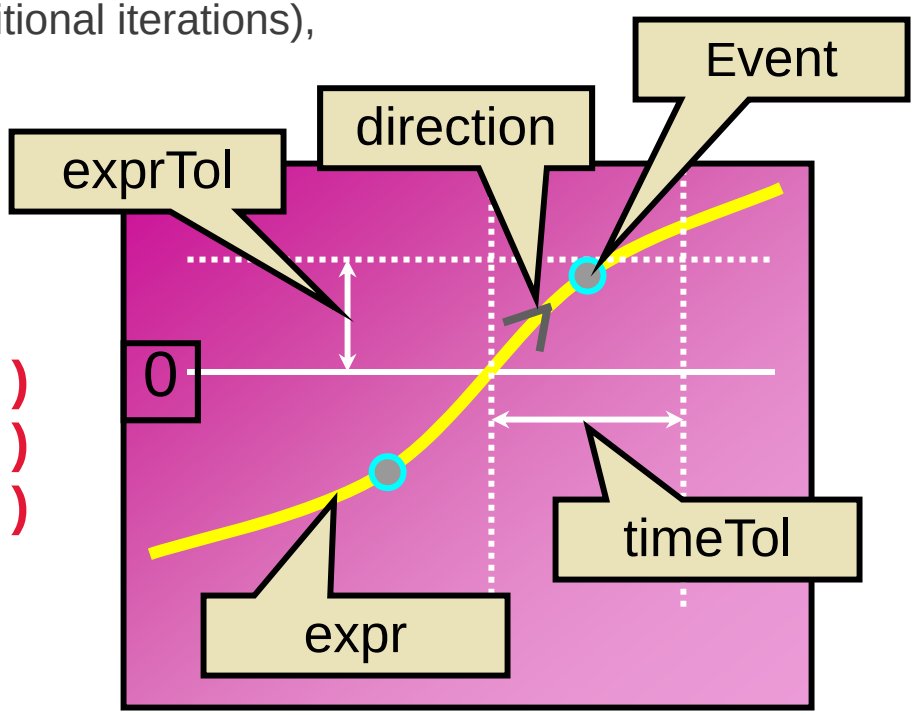


cadence

cross Event Operator

`@cross( expr, direction, timeTol, exprTol )`

- Generates event when *expr* crosses 0 in specified direction
- Timepoint is placed after the crossing within tolerances
- If timepoint selected based on simulation criteria exceeds tolerances, system will iterate until it converges to a timepoint within the box
- To limit to a single backstep (no additional iterations), set the transient analysis option:
  `fastcross = yes`
- To get the exact time of crossing, use **`last_crossing( expr )`**



Examples

```
@(cross(V(in),+1,tt,vt))
@(cross(V(in),-1,tt,vt))
@(cross(V(in), 0,tt,vt))
```

Threshold crossing sensing

@(cross(expr, direction, ttol, vtol)) statement;

- Only executes the statement on a timepoint where the expression crosses zero in the specified direction (1=up, -1=down, 0=either)
- It will search for acceptable timepoint on crossing to ensure it's within time and voltage tolerance of interpolated crossing point
- Define tolerances large as reasonable to limit time spent searching!
- If unspecified, default tolerances are chosen based on system vtol and reltol *and* the history of all signals! (see simulator "relref" option)

```
@(cross(V(in)-Vth, 1))  begin …   // unknown/variable tolerances
```

```
@(cross(V(in)-Vth, 1, 100p, 50m))  begin …
```

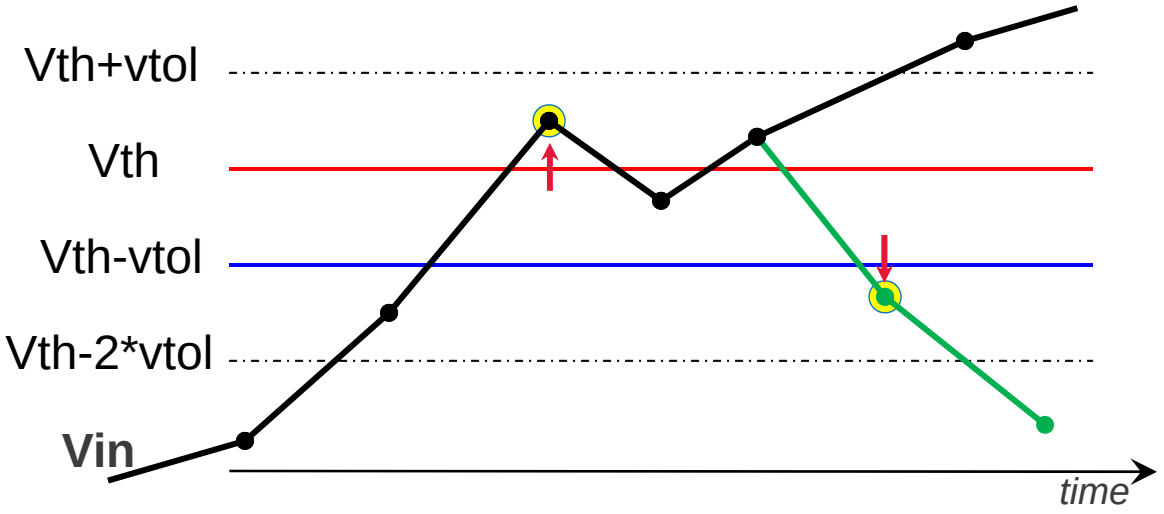
- Use @(above()) if you want the test to also occur at DC op point:

```
@(above(V(in)-Vth, 100p, 50m))  begin …
```

Hysteresis & complementary @(above())'s

- Note that "vtol" is used both for tolerance *and* hysteresis effect to suppress multiple triggering on small wiggles
- This effectively limits how large you can set the tolerance
- To define complementary @(above)'s, offset one by the hysteresis value so thresholds are properly separated for consistent results!

```
always @(above(V(in)-Vth, ttol, vtol)) State=1;
always @(above(Vth-V(in)-vtol, ttol, vtol)) State=0;
```



When to use @cross or @above

- For unidirectional testing, only difference is at DC:

```
always @(cross(V(in)-Vth, 1, 10p, 10m)) begin
    // this is executed when Vin crosses Vth ascending
```

```
always @(above(V(in)-Vth, 10p, 10m)) begin
    // this is executed when Vin crosses Vth ascending
    // and at time zero when Vin is greater than Vth
```

- For bidirectional check, use **cross** for single threshold:

```
always @(cross(V(in)-Vth, 0, 10p, 10m)) K=(V(in)>Vth);
```

- Use **above** for dual-threshold hysteresis:

```
always @(above(V(in)-Vth-10m, 10p, 10m)) K=1;
always @(above(Vth-10m-V(in), 10p, 10m)) K=0;
```

Split positive/negative sensing without hyseresis can misfire due to differing interpretations of hysteresis!

Internal operation of @cross & @above

Maintains current state (high or low)
and previous point (time and value)

On each analog timepoint solution:

- If state hasn't changed, just update previous point info
- If state changes in insensitive direction, also update state
- If state changes in sensitive direction:
  - Interpolate from previous point to estimate where crossing occurred
  - Measure distance (time & voltage) from estimate to current point
  - If within ttol & vtol, accept it: update state & generate event
  - Else tell analog solver to backstep to a better time estimate

@cross/above are only checking analog when active:
always block with **#delay** in loop would disable them!

Analog block does not allow "if" around "@cross"!

Choice of tolerances for @cross & @above

- Voltage tolerance should be:
  - smaller than minimum possible signal level (hysteresis limitation)
  - As large as possible to minimize searching iterations
- Time tolerance should be:
  - As narrow as desired for proper timing accuracy
- Most models use these events for time accuracy
  - Rapidly changing signals don't need high voltage accuracy
- If time accuracy isn't an issue, consider not using events!
  - "if" statements, "last_crossing" checks can identify crossings

Specifying vtol too large can result in apparently "missed" crossings when signal level gets too small!

Non-cross-based analog threshold sensing

– If per-timestep selection is sufficient, just use an **if** statement to detect when the state changes

```
if ( (V(in)>Vth) != pstate) begin
    pstate = (V(in)>Vth);
    {do whatever needs to happen when state changes}
```

– To measure crossing time, without actually stepping to exact point, use last_crossing to measure interpolated crossing time

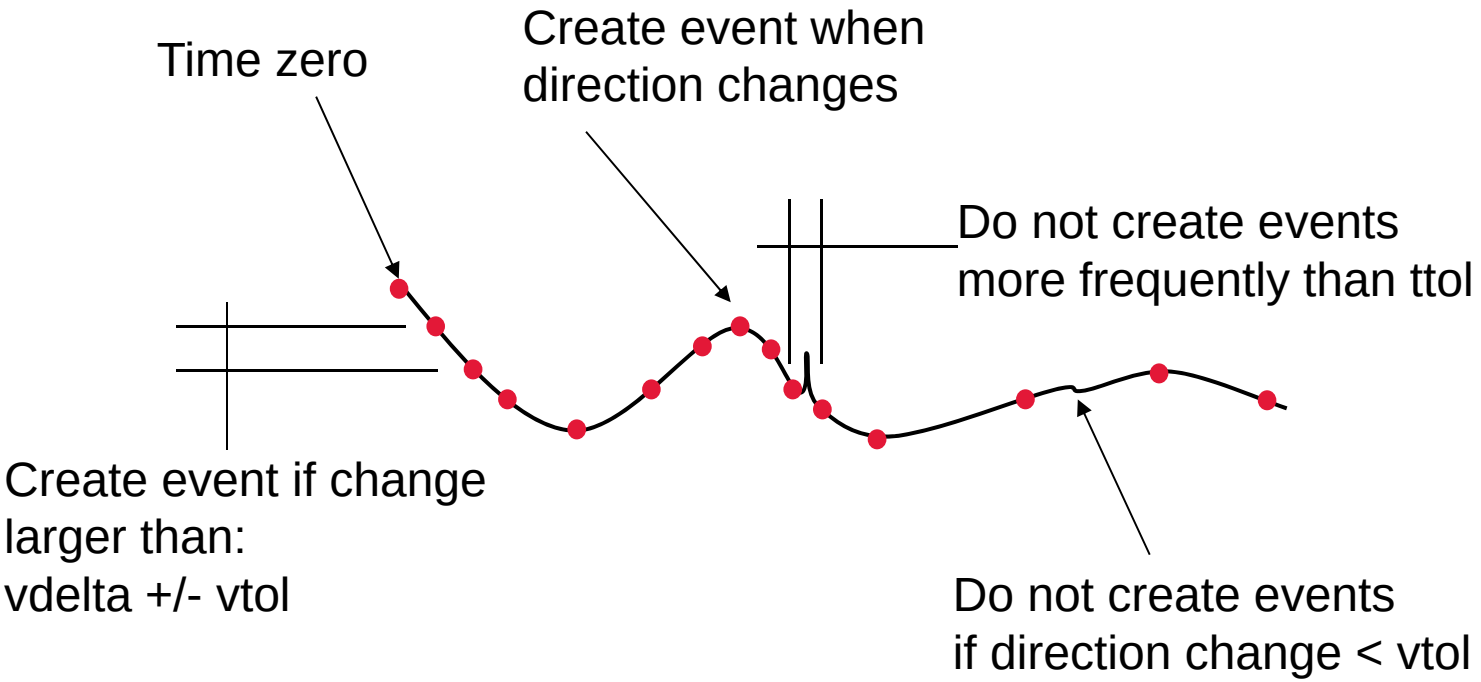
```
newpt = last_crossing(V(in)-Vth,1);
if (newpt>tcross) begin
    tcross = newpt;
    {do whatever needs to happen when state changes}
```

These run with maximum efficiency:
No additional forced timepoints needed!

Examining *absdelta* Event in Verilog-AMS

`@(absdelta(expr, delta, ttol, vtol))`

- Generates digital event when analog signal changes more than a specified amount.
- Generates events at analog timepoints, interpolates between them when needed, or skips them when not needed (based on **delta**).
- This is commonly used in E2R connect modules.



cadence

Moving Analog Timepoints Into Discrete Domain

- Using absdelta function for signal conversion to discrete:

```
always @(absdelta(V(in),10m,100p,2m)  Vsamp=V(in);
```

```
always @(absdelta(V(aclk),0.2,5p,0.1)  CK=(V(aclk)>Vth);
```

- Using absdelta to force per-timepoint events:

```
real TCK, Dval, Vdelta=100m;
analog TCK=1-TCK;
// update at every analog timepoint, or every quarter-point
// when the voltage changes by greater than Vdelta:
always @(absdelta(TCK,0.25,5p,0.1)  or above(V(in)+1k))
    if (TCK==0 || TCK==1 || Dval-V(in)>Vdelta) Dval=V(in);
```

  - Can then do signal processing, interpolation, etc. in discrete context!
  - No impact to analog solver!

Electrical-to-Logic conversion

- @cross used for E2L conversion
- @cross also used in L2E conversion
  - Same operation as E2L conversion to resolve value of logic net
- Continuity on analog side important for clean operation
  - Ramping analog signals easily convert to logic
  - Discontinuous analog changes problematical in @cross
  - Direct analog change due to logic event (discrete value used in analog expression, driving E2L) can lead to conversion failure

Attempts to convert discontinuous analog signal to discrete context is common cause of missing events!

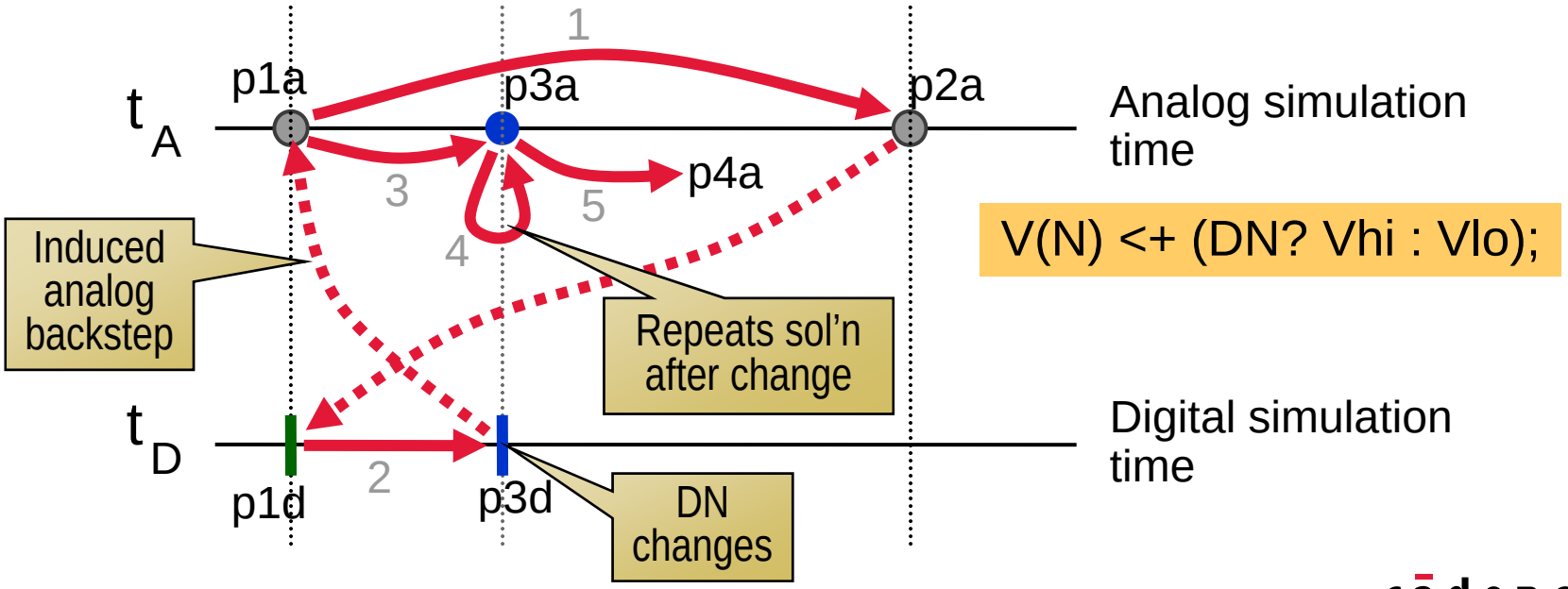
Interaction of Analog & Discrete solvers

- DC operation – Discrete segment initialized first:
  - Discrete variable initializations
  - All **`initial`** blocks executed at time zero
  - All **`always`** blocks executed at time zero
  - Analog solver iterates to operating point

- Transient operation – Analog leads Discrete evaluations:
  - Analog solver *chooses* timestep size, *iterates* to new timepoint
  - Discrete solver runs sequentially from previous to new timepoint
- Analog access from digital simulator very efficient
  - It's just an interpolation between existing analog timepoints
- Analog sensitivity to discrete event requires reevaluation
  - When discrete event effects analog result, timepoint must *roll back*
  - New analog solution at point of discrete event, then event evaluated

Logic-to-Electrical conversion
Using a digital value in an analog context

- Analog steps from p1a to p2a; digital then processes from p1d to ...
- The D2A event (p3d) forces analog simulator to backup & reevaluate
- For consistent operation, expression that changes with digital value should be wrapped with **transition** function so no discontinuity!
- Transient parameter **d2aminstep** defines minimum size of step after the forced backstep caused by the D2A event.
- Default **d2aminstep** (when set to zero) is system minstep; Nominal suggested value is 10% of minimum risetime for digital signals.



cadence

Efficiencies of common analog operations

- @(**cross**(expr, dir, ttol, vtol)), @(**above**(expr, ttol, vtol))
  - forces search for analog crossing point
  - may require multiple speculative analog timepoint evaluations
  - can force smaller timesteps (degree dependent on tolerances)
- analog access to discrete variables
  - forces backstep to resync analog solution at point of discrete change
  - can force smaller timesteps (due to backstepping)
- **transition**(dval,td,tr,tf), @(**timer**(tval,dt)), pwl/pulse source
  - only forces stepping to specific future timepoints (slows down simulation to the extent that you force excessively small steps)
- **bound_step**(dt)
  - limits maximum size of next analog timestep
- discrete or analog access to analog values, including usage of **absdelta**(expr, vdelta, ttol, vtol)
  - no significant analog overhead

$cds_get_analog_value operation

- Discrete function to sample analog node/instance value
- Can measure potential, flow, power, parameter

```
reg isok;
real rval;
initial isok = $cds_analog_is_valid(“top.i1.vdd”,”potential”);
always @(posedge ck) if (isok) rval = $cgav (“top.i1.vdd”,”potential”);
```

- Directly samples analog value from discrete context
- No sensitivity – for measurement, not dependent source!
- Issues with current measurement? TBD.

Not-yet-supported VerilogAMS features

- Initialization of real arrays?
- Access of analog busses from discrete context?
- Sensitivity to analog variables from discrete context?

→ Significant improvements to existing limitations of VerilogAMS language implementation are on the schedule for later this year, but specific features & timing still TBD.

DISCUSSION


cadence®