ARM TECHNOLOGY CONFERENCE 2010

*The Core Of Your Future*

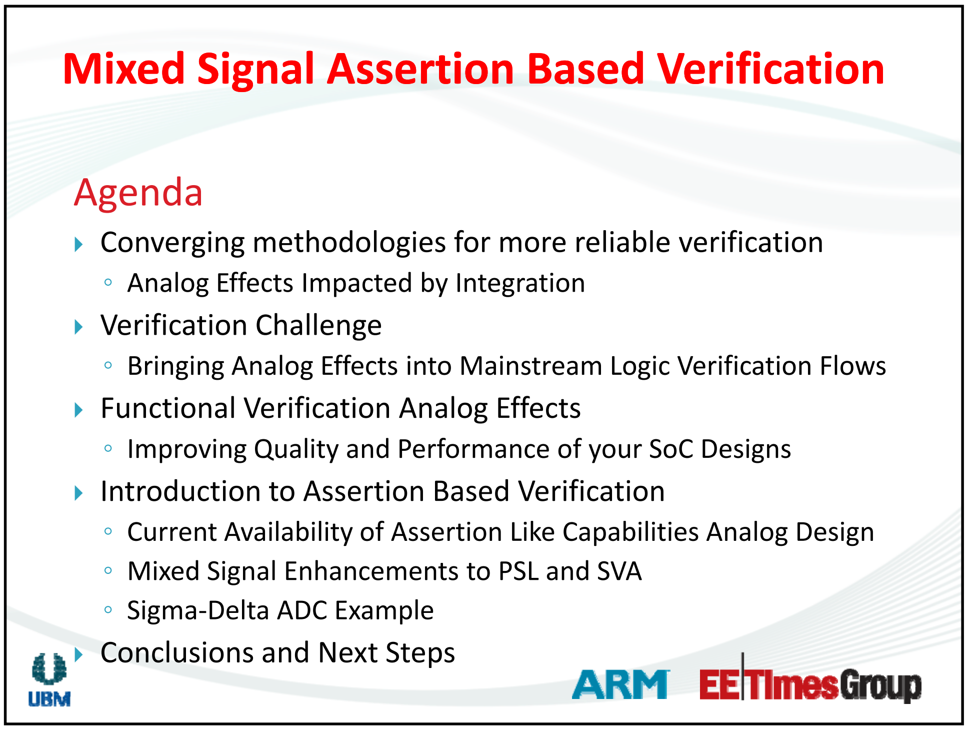
*November 9-11, 2010*

*The Santa Clara Convention Center*

*Mixed Signal Assertion Based Verification*

www.armtechcon.com

EE Times Group



## Mixed Signal Assertion Based Verification

### Agenda

- Converging methodologies for more reliable verification
  - Analog Effects Impacted by Integration
- Verification Challenge
  - Bringing Analog Effects into Mainstream Logic Verification Flows
- Functional Verification Analog Effects
  - Improving Quality and Performance of your SoC Designs
- Introduction to Assertion Based Verification
  - Current Availability of Assertion Like Capabilities Analog Design
  - Mixed Signal Enhancements to PSL and SVA
  - Sigma-Delta ADC Example
- Conclusions and Next Steps

UBM ARM EE Times Group

## Converging Methodologies for More Reliable Verification

### Bring Verification as a Methodology to Analog

- In this presentation we will talk about the potential impact and benefits of assertion based methodologies on analog mixed signal designs
- Is there a place for it and how can the industry move forward to leverage the benefits of portability, infrastructure and visibility into your design status that has been achieved with digital verification
- It will not cover every aspect of your analog verification needs and there will still be gaps closing on the specifications; however there will be immediate gains in SoC integration of analog designs

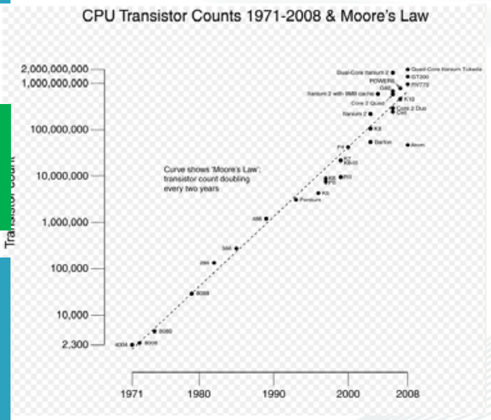
UBM ARM EETimesGroup

## Converging Methodologies for More Reliable Verification – Why Change?

Hardware keeps up with complexity
No productivity improvements gained

Memory Speeds
Enabled multi-core for SPICE, RF Simulation

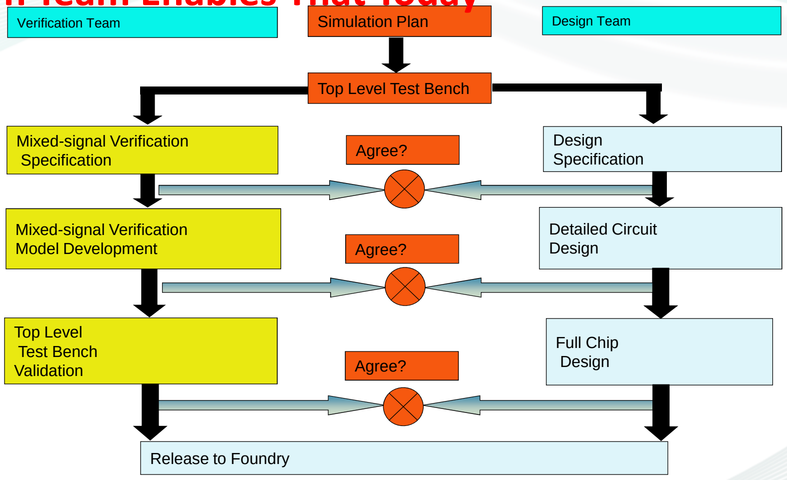
Analog effects impacting integration
•High speed I/O sensistivity
•Power startup/shutdown
•Audio-video Fidelity
Can breakdown on integration
•Require continuous verification to analog design intent



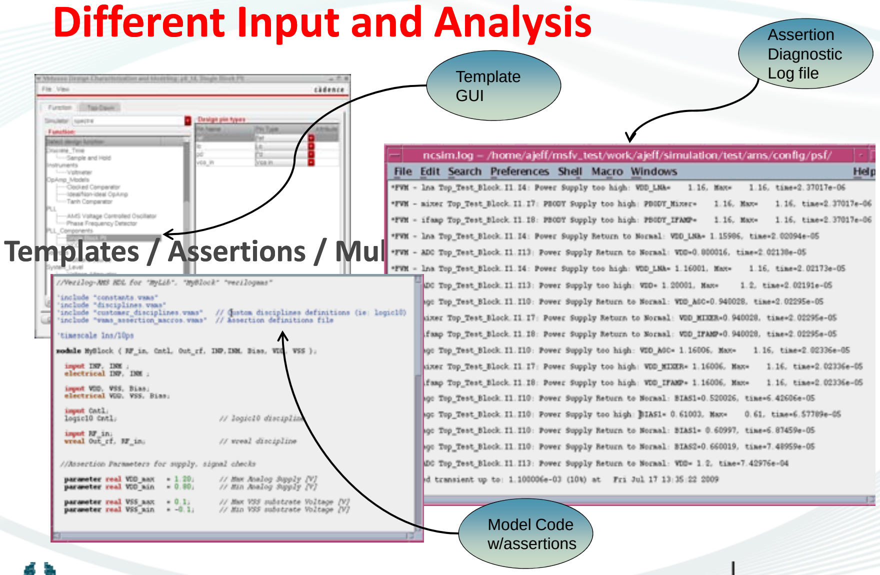
Wikipedia Moore's Law Image

UBM ARM EETimesGroup

## Analog Effects Into Logic Verification On Team Enables That Today



## Analog Effects Into Logic Verification Different Input and Analysis


Template GUI
Assertion Diagnostic Log file
Templates / Assertions / Mul
Model Code w/assertions
ncsim.log – /home/ajeff/msfv_test/work/ajeff/simulation/test/ams/config/psf/
File Edit Search Preferences Shell Macro Windows Help

```
//Verilog-AMS HDL for "MyLib", "MyBlock" "verilogams"

`include "constants.vams"
`include "disciplines.vams"
`include "customer_disciplines.vams"   // Custom disciplines definitions (ie: logic10)
`include "vams_assertion_macros.vams"  // Assertion definitions file

`timescale 1ns/10ps

module MyBlock ( RF_in, Cntl, Out_rf, INP,INM, Bias, VDD, VSS );

  input INP, INM ;
  electrical INP, INM ;

  input VDD, VSS, Bias;
  electrical VDD, VSS, Bias;

  input Cntl;
  logic10 Cntl;                          // logic10 discipline

  input RF_in;
  wreal Out_rf, RF_in;                   // wreal discipline


 //Assertion Parameters for supply, signal checks

  parameter real VDD_max    = 1.20;      // Max Analog Supply [V]
  parameter real VDD_min    = 0.90;      // Min Analog Supply [V]

  parameter real VSS_max    = 0.1;       // Max VSS substrate Voltage [V]
  parameter real VSS_min    = -0.1;      // Min VSS substrate Voltage [V]
```

```
*FVM - lna Top_Test_Block.I1.I4: Power Supply too high: VDD_LNA=    1.16, Max=    1.16, time=2.37017e-06
*FVM - mixer Top_Test_Block.I1.I7: PBODY Supply too high: PBODY_Mixer=    1.16, Max=    1.16, time=2.37017e-06
*FVM - ifamp Top_Test_Block.I1.I8: PBODY Supply too high: PBODY_IFAMP=    1.16, Max=    1.16, time=2.37017e-06
*FVM - lna Top_Test_Block.I1.I4: Power Supply Return to Normal: VDD_LNA= 1.15986, time=2.02094e-05
*FVM - ADC Top_Test_Block.I1.I13: Power Supply Return to Normal: VDD=0.800016, time=2.02130e-05
*FVM - lna Top_Test_Block.I1.I4: Power Supply too high: VDD_LNA= 1.16001, Max=    1.16, time=2.02173e-05
ADC Top_Test_Block.I1.I13: Power Supply too high: VDD= 1.20001, Max=     1.2, time=2.02191e-05
gc Top_Test_Block.I1.I10: Power Supply Return to Normal: VDD_AGC=0.940028, time=2.02295e-05
ixer Top_Test_Block.I1.I7: Power Supply Return to Normal: VDD_MIXER=0.940028, time=2.02295e-05
fmap Top_Test_Block.I1.I8: Power Supply Return to Normal: VDD_IFAMP=0.940028, time=2.02295e-05
gc Top_Test_Block.I1.I10: Power Supply too high: VDD_AGC= 1.16006, Max=    1.16, time=2.02336e-05
ixer Top_Test_Block.I1.I7: Power Supply too high: VDD_MIXER= 1.16006, Max=    1.16, time=2.02336e-05
fmap Top_Test_Block.I1.I8: Power Supply too high: VDD_IFAMP= 1.16006, Max=    1.16, time=2.02336e-05
gc Top_Test_Block.I1.I10: Power Supply Return to Normal: BIAS1=0.520026, time=6.42606e-05
gc Top_Test_Block.I1.I10: Power Supply too high: BIAS1= 0.61003, Max=    0.61, time=6.57789e-05
gc Top_Test_Block.I1.I10: Power Supply Return to Normal: BIAS1= 0.60997, time=6.87459e-05
gc Top_Test_Block.I1.I10: Power Supply Return to Normal: BIAS2=0.660019, time=7.48959e-05
DC Top_Test_Block.I1.I13: Power Supply Return to Normal: VDD= 1.2, time=7.42976e-04
d transient up to: 1.100006e-03 (10%) at   Fri Jul 17 13:35:22 2009
```

## Analog Effects Suitable for Methodology

Cadence RFKit
•Implemented Functional Verification Design Flow

Cadence RFKit
•Functional verification models
•Assertions applied to blocks

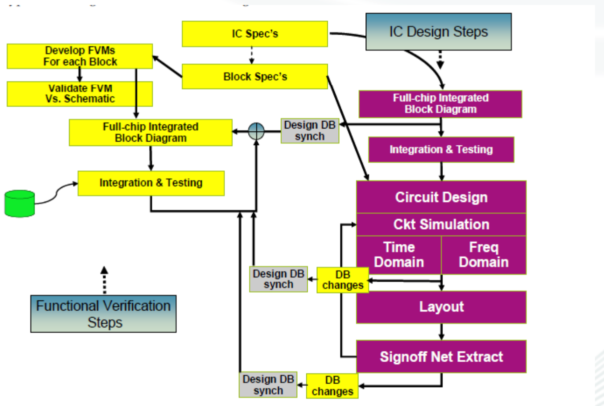


Figure 158:Functional verification & IC design flow

## Analog Effects Suitable for Assertions

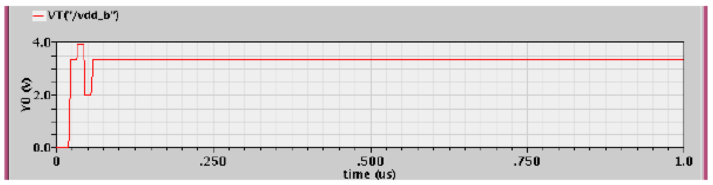
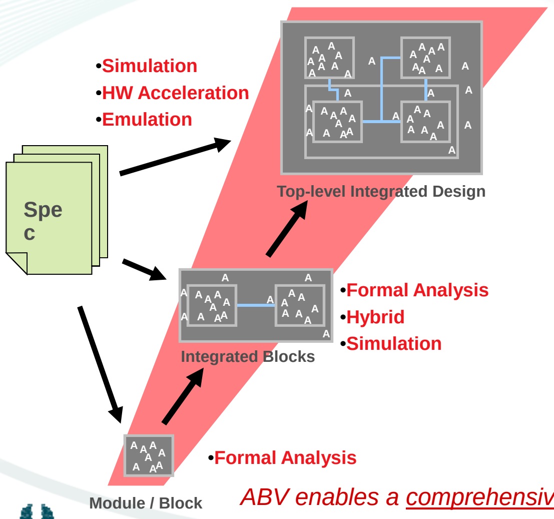


Figure 162:FVM Supply Variation

### RFKit Application of Assertions

- Assertions go with block for validation while blocks integrated
- Assertions leveraged in RFKit
  - Power supply and ground variation
  - Power down signal change
  - Supply resistance connected to supply changes
  - Gain variation at the output as the gain control voltage changes

## Introduction to Assertion Based Verification

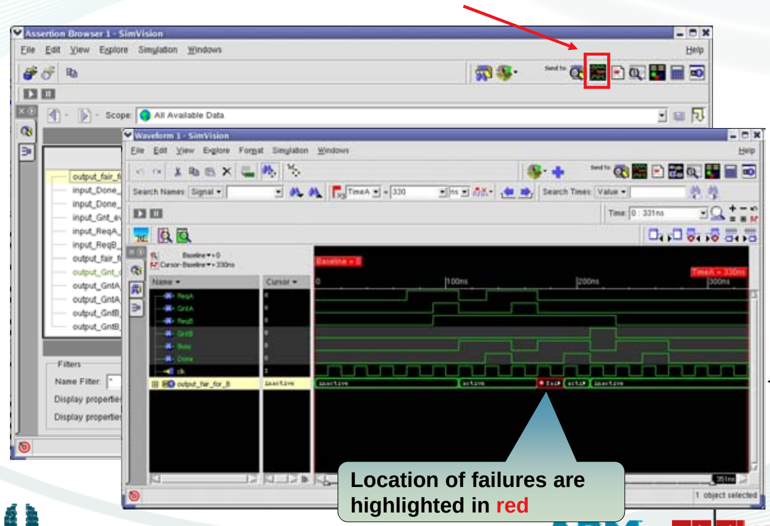


- Improve documentation
- Improved observability
  - Identifies errors where they take place instead of at the outputs
  - Monitors behavior for all vectors
- Improved controllability
- Improved coverage
- Verification starts sooner

*ABV enables a comprehensive verification approach*

## ABV Debugging

Select Assertion -> Select Waveform Icon

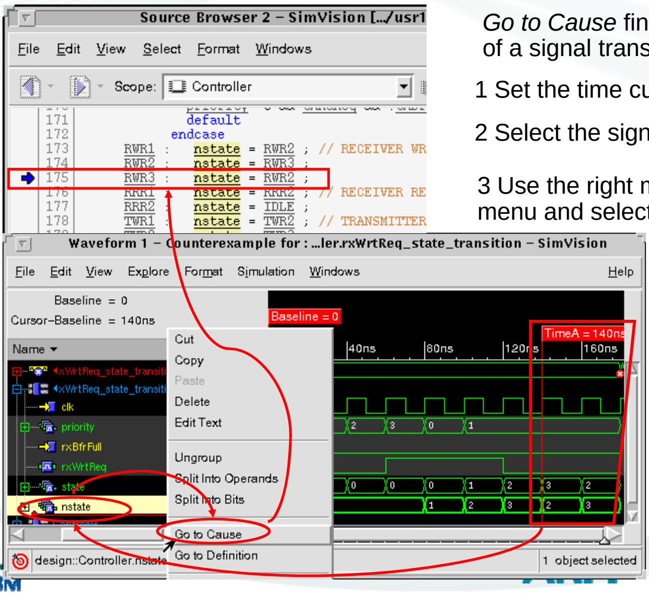


Assertion Browser 1 - SimVision
File Edit View Explore Simulation Windows Help
Scope: All Available Data
Waveform 1 - SimVision
Name
Cursor
output_fair_for_B
inactive
active
fail

Assertion state is added to waveform

Location of failures are highlighted in red

## ABV with Simvision: Go To Cause



```
        default
      endcase
  RWR1 :   nstate = RWR2 ; // RECEIVER WR
  RWR2 :   nstate = RWR3 ;
  RWR3 :   nstate = RWR2 ;
  RRR1 :   nstate = RRR2 ; // RECEIVER RE
  RRR2 :   nstate = IDLE ;
  TWR1 :   nstate = TWR2 ; // TRANSMITTER
```

*Go to Cause* finds the assignment of a signal transition in the waveform

1 Set the time cursor

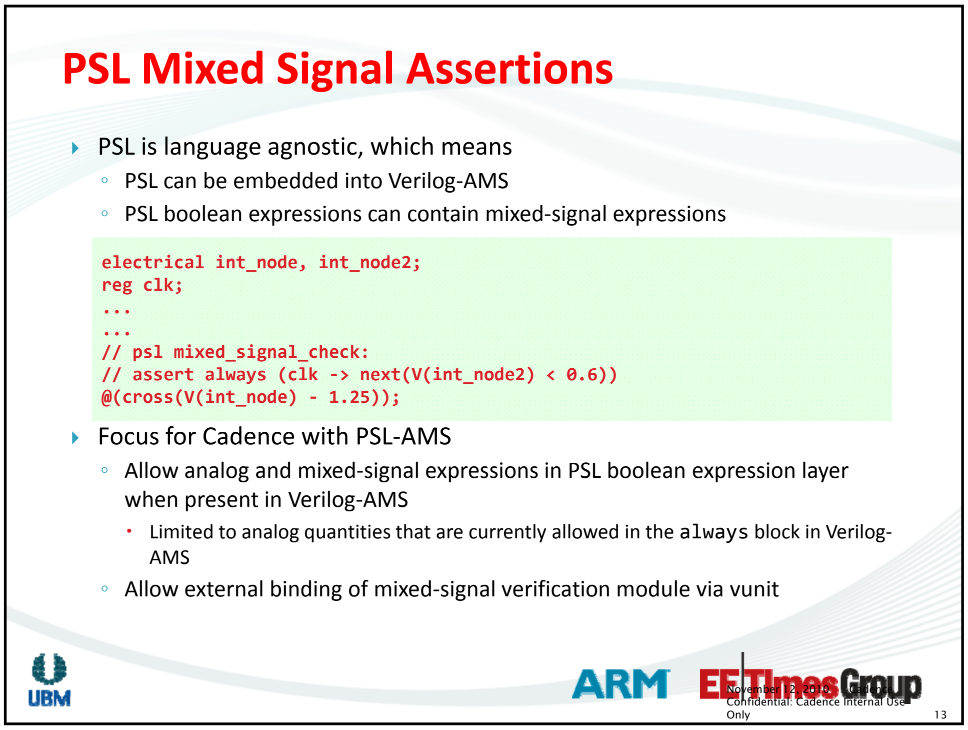
2 Select the signal of choice

3 Use the right mouse button pop-up menu and select *Go to Cause*

## Assertions Using Behavioral Language

```
// GENERIC VERILOGAMS ASSERTION CHECK MACROS:
// This macro prints a message and sets an error flag when the Val is
// too small or too large, and prints an OK message and unsets the error
// flag when it returns to normal operating range.  Note that it executes
// Td (seconds) after a crossing event occurs, so that shorter glitches
// will be ignored.  First test is performed at time Td. Only difference
// between analog & discrete version is the @() line which defines when
// to repeat the test.  Note that the En indicates checking status, not
// operational status, so disabled just means do no checking (error flags
// will stay in "OK" state).

// ARGUMENT LIST:
//  Val    = input value to check
//  Max,Min= validity range of Val.  Prints message when out-of-range.
//  Desc   = quoted text for the message - prints as:  "{Desc} is too high"
//  Name   = quoted text for value info - prints as:  "{Name} = {Val}"
//  Flg    = register or integer to hold state (1=error, 0=OK or not enabled)
//  En     = input flag to enable assertion checking: 0 means stop checking.
//  Td     = time delay between event and check; also analog cross time tol.
//  Vtol   = crossing voltage tolerance (analog form only)

`define ACHECK(Val,Max,Min,Desc,Name,Flg,En,Td,Vtol) \
  reg Flg=1'b0;   \
  always begin    \
   if (En===1'b0) begin \
    Flg<=1'b0;    \
    wait (En===1'b1); \
   end            \
   #(Td/1n) if (Val<(Min)) begin \
    if (Flg==1'b0) $display( \
          "*FVM - %s %m: %s too low: %s=%8g, Min=%8g, time=%.5e\n", \
                `MNAME, Desc,      Name, Val,     Min,        $abstime); \
    Flg<=1'b1; \
   end          \
   else if (Val>(Max)) begin \
    if (Flg==1'b0) $display( \
          "*FVM - %s %m: %s too high: %s=%8g, Max=%8g, time=%.5e\n",\
                `MNAME, Desc,      Name, Val,     Max,        $abstime); \
    Flg<=1'b1; \
   end          \
   else if (Flg==1'b1) begin \
    $display("*FVM - %s %m: %s Return to Normal: %s=%8g, time=%.5e\n",  \
                 `MNAME, Desc,                Name, Val,      $abstime);  \
    Flg<=1'b0; \
   end          \
   @(En or cross(Val-(Max),0,Td,Vtol) or cross(Val-(Min),0,Td,Vtol)); \
  end
```

- Mixed-signal languages such as Verilog-A/MS can be used to create monitoring models
  - Requires some level of modeling expertise and understanding of mixed-signal simulation semantics ☹
  - Generally relies on 'if (expr) $display "msg"' approach
    - Analogy: "print" statements in software debug. Where's the IDE?
  - Must edit model, or testbench, or design itself
    - No vunit/"bind" capability
  - No assertion browser/waveforms/debug capability
- Does not leverage any of the system (IDE) already built around PSL or SVA
  - E.g. Functional coverage on assertions

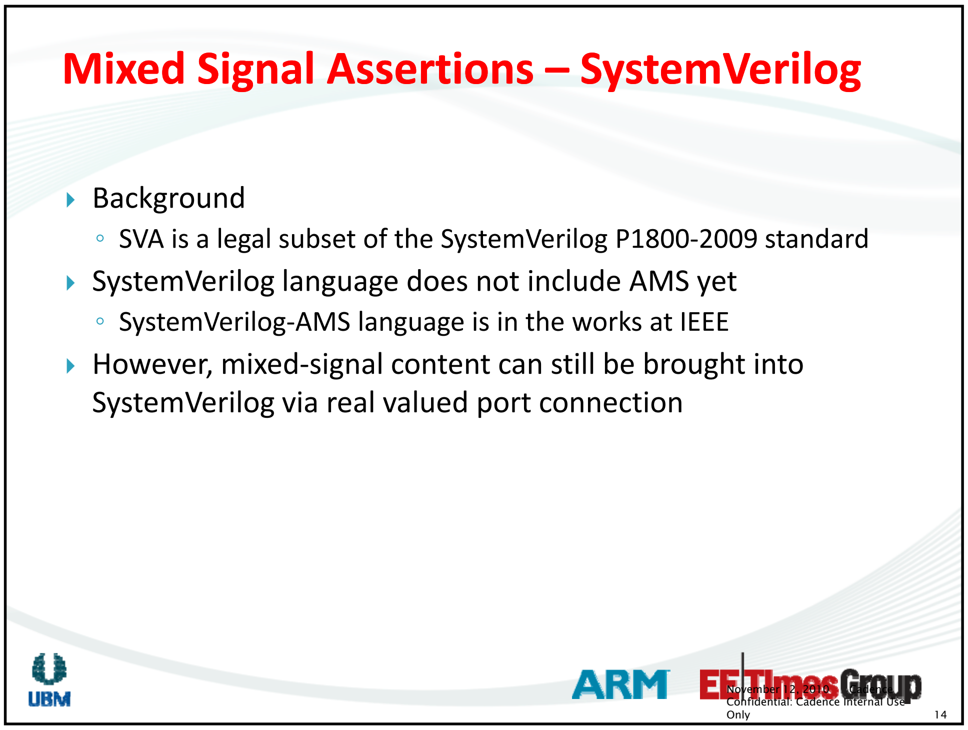


## PSL Mixed Signal Assertions

- PSL is language agnostic, which means
  - PSL can be embedded into Verilog-AMS
  - PSL boolean expressions can contain mixed-signal expressions

```
electrical int_node, int_node2;
reg clk;
...
...
// psl mixed_signal_check:
// assert always (clk -> next(V(int_node2) < 0.6))
@(cross(V(int_node) - 1.25));
```

- Focus for Cadence with PSL-AMS
  - Allow analog and mixed-signal expressions in PSL boolean expression layer when present in Verilog-AMS
    - Limited to analog quantities that are currently allowed in the `always` block in Verilog-AMS
  - Allow external binding of mixed-signal verification module via vunit

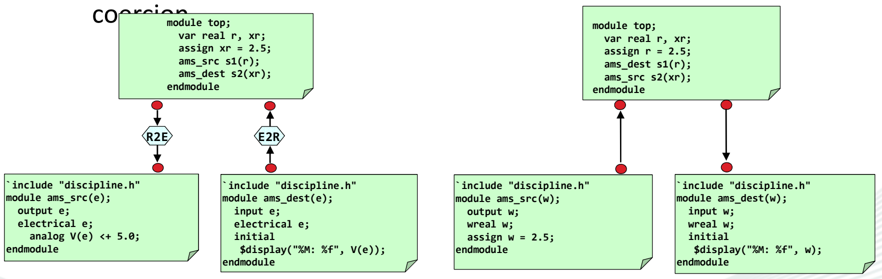


## Mixed Signal Assertions – SystemVerilog

- Background
  - SVA is a legal subset of the SystemVerilog P1800-2009 standard
- SystemVerilog language does not include AMS yet
  - SystemVerilog-AMS language is in the works at IEEE
- However, mixed-signal content can still be brought into SystemVerilog via real valued port connection

## SystemVerilog Cross Domain Connectivity

- SystemVerilog testbench driven methodology
  - 3 classes of connectivity supported:
    - Data and net types between Verilog-2001 and Verilog-AMS objects
    - SV Real Variable to Verilog-AMS Electrical – insertion of E2R connect module
    - SV Real Variable to Verilog-AMS WReal – direct connection by coercion



```
module top;
  var real r, xr;
  assign xr = 2.5;
  ams_src s1(r);
  ams_dest s2(xr);
endmodule
```

```
`include "discipline.h"
module ams_src(e);
  output e;
  electrical e;
    analog V(e) <+ 5.0;
endmodule
```

```
`include "discipline.h"
module ams_dest(e);
  input e;
  electrical e;
  initial
   $display("%M: %f", V(e));
endmodule
```

```
module top;
  var real r, xr;
  assign r = 2.5;
  ams_dest s1(r);
  ams_src s2(xr);
endmodule
```

```
`include "discipline.h"
module ams_src(w);
  output w;
  wreal w;
  assign w = 2.5;
endmodule
```

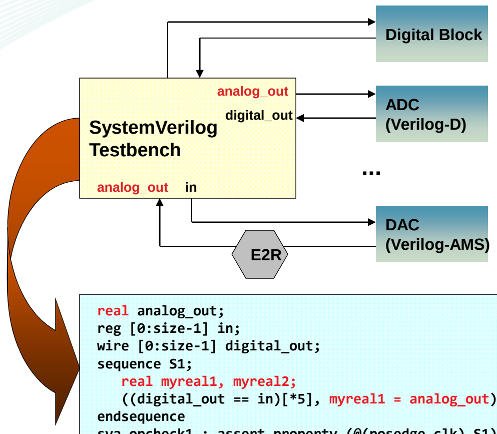
```
`include "discipline.h"
module ams_dest(w);
  input w;
  wreal w;
  initial
   $display("%M: %f", w);
endmodule
```

**SystemVerilog Real Variable Connecting to Verilog-AMS Electrical**

**SystemVerilog Real Variable Connecting to Verilog-AMS Wreal**

## SVA Assertions for Mixed-Signal

- Relies on SV real variables



```
real analog_out;
reg [0:size-1] in;
wire [0:size-1] digital_out;
sequence S1;
   real myreal1, myreal2;
   ((digital_out == in)[*5], myreal1 = analog_out) ##1 myreal1 > 0.5;
endsequence
sva_opcheck1 : assert property (@(posedge clk) S1);
```

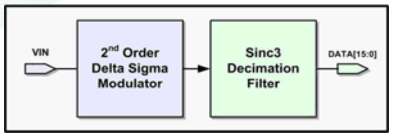
## Leveraging Assertions In an ADC

### Capturing Specification of Sequential Behavior

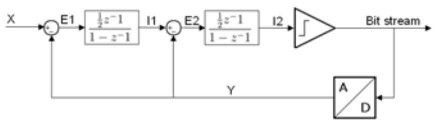
- ADC's, DAC's, Switch Cap Filters, Serdes good candidates
  - Specification of circuits contains sequential behavior
- ADC example the following behavior is verified
  - Integrator behavior
  - Comparator Operations
  - Loop Stability Fundamentals

UBM ARM EETimesGroup

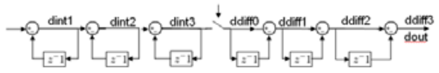
## ADC Architecture Representation



- Sigma Delta Architecture
  - Modulator and Filter


X
E1
$\frac{\frac{1}{2}z^{-1}}{1-z^{-1}}$
I1
E2
$\frac{\frac{1}{2}z^{-1}}{1-z^{-1}}$
I2
Bit stream
Y
A
D

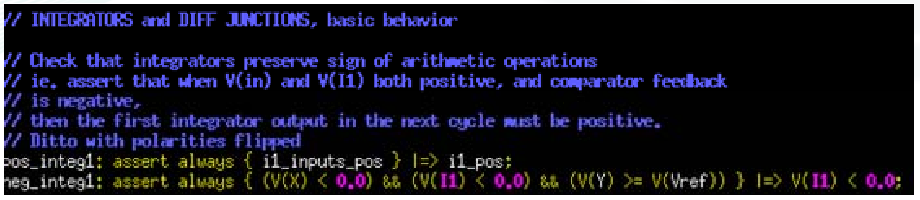
- Modulator with Integrator Feedback



- Filter

UBM ARM EETimesGroup

## ADC PSL Checking Basic Functionality



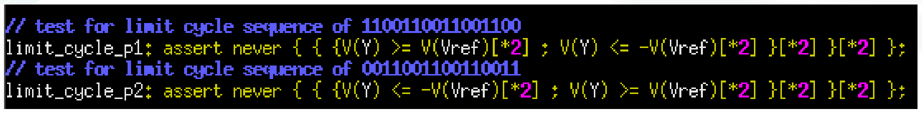
```
// INTEGRATORS and DIFF JUNCTIONS, basic behavior

// Check that integrators preserve sign of arithmetic operations
// ie. assert that when V(in) and V(I1) both positive, and comparator feedback
// is negative,
// then the first integrator output in the next cycle must be positive.
// Ditto with polarities flipped
pos_integ1: assert always { i1_inputs_pos } |=> i1_pos;
neg_integ1: assert always { (V(X) < 0.0) && (V(I1) < 0.0) && (V(Y) >= V(Vref)) } |=> V(I1) < 0.0;
```

- Integrator Functionality
  - The sign of the integrator preserved positive cycle and negative cycle
  - First assertions leverages internally generated VerilogAMS values

UBM

ARM EETimesGroup

## ADC PSL Checking Sequential Bit Patterns



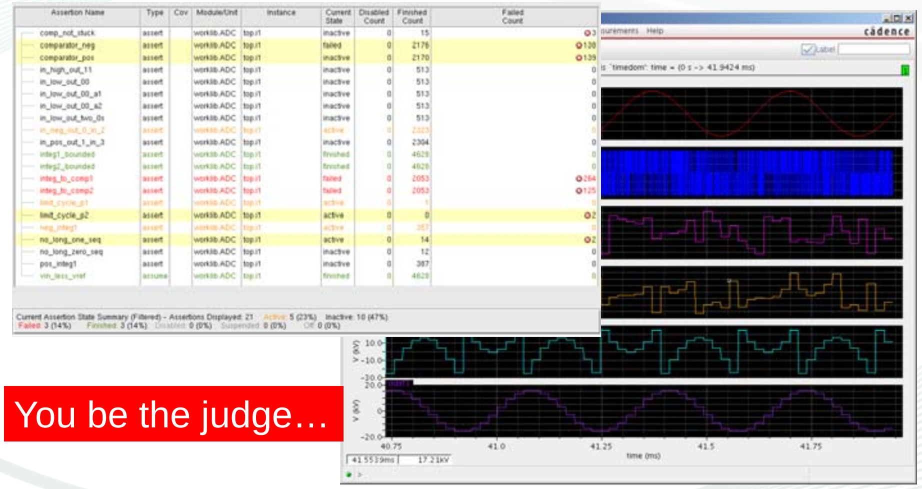
```
// test for limit cycle sequence of 1100110011001100
limit_cycle_p1: assert never { { {V(Y) >= V(Vref)[*2] ; V(Y) <= -V(Vref)[*2] }[*2] }[*2] };
// test for limit cycle sequence of 0011001100110011
limit_cycle_p2: assert never { { {V(Y) <= -V(Vref)[*2] ; V(Y) >= V(Vref)[*2] }[*2] }[*2] };
```

- Modulator Stability
  - Checking for the presence of undesirable bit patterns
  - Repeating bit patterns leading to audible tones/clicks

UBM

ARM EETimesGroup

## ADC PSL Viewing the Results
## Assertion Browser vs. Waveforms



| Assertion Name | Type | Cov | Module/Unit | Instance | Current State | Disabled Count | Finished Count | Failed Count |
|---|---|---|---|---|---|---|---|---|
| comp_not_stuck | assert | | worklib.ADC | top.i1 | inactive | 0 | 15 | 3 |
| comparator_neg | assert | | worklib.ADC | top.i1 | failed | 0 | 2176 | 138 |
| comparator_pos | assert | | worklib.ADC | top.i1 | inactive | 0 | 2170 | 139 |
| in_high_out_11 | assert | | worklib.ADC | top.i1 | inactive | 0 | 513 | 0 |
| in_low_out_00 | assert | | worklib.ADC | top.i1 | inactive | 0 | 513 | 0 |
| in_low_out_00_a1 | assert | | worklib.ADC | top.i1 | inactive | 0 | 513 | 0 |
| in_low_out_00_a2 | assert | | worklib.ADC | top.i1 | inactive | 0 | 513 | 0 |
| in_low_out_two_0s | assert | | worklib.ADC | top.i1 | inactive | 0 | 513 | 0 |
| in_neg_out_0_in_2 | assert | | worklib.ADC | top.i1 | active | 0 | 2323 | 0 |
| in_pos_out_1_in_3 | assert | | worklib.ADC | top.i1 | inactive | 0 | 2304 | 0 |
| integ1_bounded | assert | | worklib.ADC | top.i1 | finished | 0 | 4628 | 0 |
| integ2_bounded | assert | | worklib.ADC | top.i1 | finished | 0 | 4628 | 0 |
| integ_to_comp1 | assert | | worklib.ADC | top.i1 | failed | 0 | 2053 | 264 |
| integ_to_comp2 | assert | | worklib.ADC | top.i1 | failed | 0 | 2053 | 125 |
| limit_cycle_p1 | assert | | worklib.ADC | top.i1 | active | 0 | 1 | 0 |
| limit_cycle_p2 | assert | | worklib.ADC | top.i1 | active | 0 | 0 | 2 |
| neg_integ1 | assert | | worklib.ADC | top.i1 | active | 0 | 357 | 0 |
| no_long_one_seq | assert | | worklib.ADC | top.i1 | active | 0 | 14 | 2 |
| no_long_zero_seq | assert | | worklib.ADC | top.i1 | inactive | 0 | 12 | 0 |
| pos_integ1 | assert | | worklib.ADC | top.i1 | inactive | 0 | 387 | 0 |
| vin_less_vref | assume | | worklib.ADC | top.i1 | finished | 0 | 4628 | 0 |

Current Assertion State Summary (Filtered) - Assertions Displayed: 21 Active: 5 (23%) Inactive: 10 (47%)
Failed: 3 (14%) Finished: 3 (14%) Disabled: 0 (0%) Suspended: 0 (0%) Off: 0 (0%)

You be the judge…

UBM

ARM EETimesGroup

## Mixed Signal Assertion Based Verification

Is Assertion Based Verification Applicable to Mixed Signal Designs?

- ADC's, DAC's, Switch Cap Filters, Serdes good candidates
- ADC Example Demonstrated Verification of Key Behavior
  - Integrator behavior
  - Comparator Operations
  - Loop Stability Fundamentals
- Technology Exists Today to Augment Existing Methodologies with Assertion Based Verification

UBM

ARM EETimesGroup