The How To’s of Advanced Mixed-Signal Verification

John Brennan, Thomas Ziller, Kawe Fotouhi, Ahmed Osman

Cadence Design Systems
1. Metric-Driven Verification for MS
2. Verification Planning and Management in MS
3. Universal-Verification Methodology for MS
4. Real-number Modeling Capabilities
5. Analog and MS Assertions
6. Q&A
Metric-Driven Verification for Mixed-signal
John Brennan
The Winds of Change

Many forces at work to drive change

TIPPING POINT INDICATORS

- Digitally-calibrated, compensated
- Feedback between D and A
- Software controlled
- Power management
- 28nm and below
- Long Verification Cycles

Source: IBS
Mixed-signal Verification: Complexity Issues

- How do I build consistency between digital and analog teams?

- How do I verify the digital content in this SoC?

- How do I verify the mixed-signal interconnects?

- How do I abstract analog behavior?

- How do I verify the mixed-signal IP?
Advanced Verification Methodology

Functional Verification Approaches

Directed Tests Driven

Test targets

Test 1: ✓
Test 2: ✓
Test 3: x
Test 4: ✓

Least effective in finding the hidden bugs

Coverage Driven

Coverage targets

Coverage
CG1 ✓
CG2 ✓
CG3 x
CG4 ✓

Adds quality & productivity, but difficult to estimate completion

Metric Driven

Feature-based Verification targets
(Metrics for cov+checks)

Feature A
Subset 1 20%
Subset 2 80%
Feature B
Feature C
Feature D
50% 10%
100% 70% 20%

Feature-based plan with extended metrics enables efficient and accurate project closure

Metrics for DV

Verif. Plan
- Feature A
- Feature B
- Feature C
- Feature D

Chip Features

50% 20%
Metric Driven Verification (MDV): Overview

Planning with unified verification metrics

Coverage & Failure Analysis
Metric Visualization

Yes

No

Signoff?

Done

Plan

Construct

Execute

Measure / Analyze

Metric-based Executable Verification plan

Incisive® VIP Portfolio

Coverage Assertions Checks

Testbench Simulation, Formal, HW/SW Co-Sim, LPV, MSV, Sim-Acceleration, Emulation

IEM

JG

IES

ISX

SN

VE Start

Production

Prototype

Chip

Integration

Module

Set Two

Module

Set One

Actual Metrics Achieved

Target Metrics Milestones

Missed Milestone

Successful Milestone

PDF
Key Elements of MS Verification Solution

- **Integrated Environment**
  - Performance
  - Features

- **Digital Verification Concepts**
  - Simulation
  - Behavioral Modeling

- **Methodology**
  - Library
  - Tools abstracting analog and mixed-signal functionality

- **Planning**
- **Tracking to closure**
- **Execution and debugging**
Cadence mixed-signal verification solution

Bridging the GAP, addressing complexity

Focus for Today*
- Plan, Track, Analyse, Report
- Re-use and Automation
- Enabling technology
- Core simulation engine

Integration & Automation

Abstraction Level
- Analog: High accuracy
- Digital: High simulation throughput

Metric-Driven Verification Methodology
- TB Development
- Sim Management (Analog Design Environment)
- Analog Modeling (SMG)
- Multi-Mode Simulation (MMSim)
- Fast SPICE (XPS)
- Transistor Simulation (Spectre)
- Logic Simulation (Incisive)
- RNM Simulation
- Multi-Language Simulation (Incisive)
- UVM Mixed Signal
- PSL / SVA Assertion
- Functional Coverage

Plan, Track, Analyse, Report

Re-use and Automation

Enabling technology

Core simulation engine

Accellera
Systems Initiative

2015 DvCon Conference and Exhibition Europe
Agenda

1. Metric-Driven Verification for MS
2. Verification Planning and Management in MS
3. Universal-Verification Methodology for MS
4. Real-number Modeling Capabilities
5. Analog and MS Assertions
6. Q&A
Verification Planning in MS
Kawe Fotouhi
What is a Meaningful Verification plan?

- **Functional Verification** is the process of proving the convergence of the functional specification, the design intent, and the Test environment implementation.

- A good and meaningful verification plan will prove that convergence.
Fundamentals of a Good Verification Plan

- Design Intent
  - Directly links and maps all specified features and key details
  - Correctly captures important implementation specific concerns
  - Be able to correlate features with corresponding measured metrics

- Functional Specification
  - Functional & Design Specs

- Implementation of verification env
  - Metrics

- Verification Plan
  - design and Verification team
Creating a Feature based Verification Plan I

Feature Identification

• Get all project related people together
  – Analog designer, analog and digital verification engineer, Marketing, Concept, Software, ...
• „Brainstorm“ plan hierarchy and features based on
  – Specification
  – KnowHow, experience & gut feeling
• Feature analysis focuses on :
  – "What" to verify
  – Which domain (analog/digital) to verify
  – "How" to verify
• Feature Examples
  – Device mode and configuration options
  – Traffic or protocol handling
  – Protocol or device exception handling
  – Performance specification
  – Operation conditions (PVT)
    • Process variations
    • Voltage supply
    • Temperature
  – Application modes
  – External connections
  – Typical and critical use and corner cases (duty cycle, phase noise ratio etc.)
Creating Feature based Plan II

Attribute Elaboration

- Translating Feature requirements into concrete metric goals
- Ask HOW features will be measured
- Identify required testcases, coverage and checks metrics
- Which attributes and values are important?
  - Driven by the spec
- Where should each value be observed?
  - At boundary or involving internal signals
- When are the values valid to be sampled?
  - reaching a certain voltage in a given time window after power-up
PLL output (txi_clk) analog performance and functional feature

- Test the SNR in combination with ref_clk offset
- Cover corner case frequency
- Check PLL locks
PLL (txi_clk) output Digital

Cover all possible output frequency (randomize fsynth) FRACTIONAL and INTEGRAL MODE
PLL Core feature – modelling requirements

PLL locks even if it shouldn’t if the dutycycle of the ref clock is too large
Agenda

1. Metric-Driven Verification for MS
2. Verification Planning and Management in MS
3. Universal-Verification Methodology for MS
4. Real-number Modeling Capabilities
5. Analog and MS Assertions
6. Q&A
UVM for Mixed-signal
Thomas Ziller
Using UVM to Apply MDV

- Components of a MDV environment
  - Automated Stimulus Generation
  - Independent Checking
  - Coverage Collection

![Diagram of UVM components](image-url)
VAMS Netlist

DUT
VAMS/Transistor

Gasket
SigGen
Sampler

Wreal/Electrical

Real Numbers

Customizable dms_wire gasket

SV TB

UVM Classes

Real Numbers

master_agent

sequencer

monitor

driver

VIF

Real Numbers

IF

CTRL

CTRL

CTRL

UVM

Classes

`uvm_do_with ana1_wire_seq {
    clk_period == 0.5; // sample clk
    ampl == 0.001;     // 1 mV
    bias == 1.1;
    freq == 100e6;     // 100 MHz
    phase == 0.0;
}

Bias

Amplitude

Phase

Frequency

0

0
SV RNM: Coverage/Randomization

- Coverage/Randomization of reals
- Cadence provides full coverage/randomization support
  - Full compliment of real variable usage in randomization

```verilog
// Vector bins with precision
class my_tb_cls;
    rand real voltage;
    constraint my_constr {voltage dist
        { [1.0 :1.25] := 1,
            [1.25:1.5 ] := 10,
            [1.5 :2.0 ] := 1
        };
    }
covergroup cg {
    my_voltage : coverpoint voltage {
        type_option.real_interval = 0.1;
        bins b1[] = {[1.0:2.0]};
    }
endgroup : cg
endclass
```

Randomization of the voltage
Coverage of what voltage values were generated
N-Fractional PLL Mixed-Signal
Constrained Random Simulation Results

Constrained random variations

Settling
PLL lock
Cal. done
Calibration
Settling
Cal. done
Lock
Turn on avdd supply
N-Fractional PLL Mixed-Signal
"avdd" Supply Range Checking

UVM-MS env.
Constrained Random Number Gen. [2.325 ... 2.675]
dms_wire Analog UVC

PLL DUT
Vreg PLL Volt. Regulator
Vco and Div/2
1.2V
div2clk

avdd electrical

+7%
+5%
+2.5
-5%
-7%

over_volt
supply_ok
under_volt

0V

over_volt
supply_ok
under_volt

accellera
SYSTEMS INITIATIVE

2015
DESIGN AND VERIFICATION
CONFERENCE AND EXHIBITION
EUROPE
N-Fractional PLL Mixed-Signal

"avdd" Supply Range Checking

- **vco starts** when the `avdd` supply is within the 2.5V±5% range.
- **vco remains turned-off** when `avdd` is less than 2.375V.
- **under_voltage condition** when `avdd` is less than 2.375V.

`supply_ok`
Covergroup definitions:

```vhdl
covergroup bias_cg;
  bias_cp : coverpoint bias {
    bins over_volt = {[2.625:10]};
    bins supply_ok = {[2.375:2.625]};
    bins under_volt = {[0:2.375]};
  }
endgroup // bias_cg

covergroup cg_fsynth;
  cp_fsynth : coverpoint fsynth{
    illegal_bins a = {[14'h2201:14'h3fff]};
    option.auto_bin_max = 25;
  }
endgroup : cg_fsynth
```

MS Regression Control & Analysis
Functional Coverage Results Example (20 runs)
1. Metric-Driven Verification for MS
2. Verification Planning and Management in MS
3. Universal-Verification Methodology for MS
4. Real-number Modeling Capabilities
5. Analog and MS Assertions
6. Q&A
Real-number Modeling Capabilities
Ahmed Osman
Performance: Simulation throughput

Behavioral Modeling DMS vs AMS

- Model analog block operation as discrete real data
  - Signal flow-based modeling approach
- Key advantages of RNM
  - Discrete solver only
  - Very high simulation performance
  - Event driven or sampled data modeling of analog operation
  - No analog solver, no convergence problems!
  - Can be written by analog designers and/or digital verification engineers
- RNM languages include
  - Verilog-AMS (wreal)
  - VHDL
  - SystemVerilog
  - e

Accellera Systems Initiative
Analog or Real Modeling: What is the Difference?

Analog Modeling

• Describes current vs. voltage relationship between nodes in model
• Newton-Raphson iteration process performs matrix inversion to solve all voltage and currents
• Timestep until next solution is selected based on accuracy criteria

Real Modeling

• No matrix solution – output computed directly from input & internal state. Model defines when to perform each internal computational segment
• No continuous time operation – only sampled, clocked, and/or event-driven operations. Updates can be performed when inputs change and/or at specific time increments
• Same format for digital and real modelling – difference is data type
SystemVerilog IEEE 1800-2012 LRM

– **User-Defined Types (UDTs)**
  - Allows for single-value real nettypes
  - Keyword used: `nettype`
  - Allows for multi-value nets (multi-field record style)
  - It can hold one or more values (such as voltage, current, impedance)
    in a single complex data type that can be sent over a wire

– **User-Defined Resolution (UDRs)**
  - Functions to resolve user-defined types using keyword: `with`
  - Specifies how to combine user defined types

– **Interconnect Nets**
  - Types
    - Explicit: Type-less/Generic nets with keyword: `interconnect`
    - Implied: A Verilog(-AMS) net with keywords: `wire`, `tri`, `wand`, `triand`, `wor`, or `trior`
  - Used only for a net or port declarations
User-Defined Nets can carry one or more values over a single net.

Real values can be used to communicate voltage, current and other values between design blocks.

User-Defined Resolutions (UDR) functions are used to combine multiple outputs together.

UDR
Programmable means to define how multiple fields in a UDT are resolved.
Declaring User-Defined Nettype

- A SystemVerilog user-defined nettype without any resolution function can be declared as:

```systemverilog
nettype T myNet;

module top;
    nettype T myNet;
    myNet w;
    assign w = T'{0.1, 0.2, 1'b1, 10};
    initial begin
        $display("Value of w -> %f => %p", $realtime, w);
        #1 $display("Value of w -> %f => %p", $realtime, w);
        #5 $display("Value of w -> %f => %p", $realtime, w);
    end
endmodule
```

Value of w -> 0.000000 => '{voltage:0, current:0, field3:'h0, field4:x}
Value of w -> 1.000000 => '{voltage:0.1, current:0.2, field3:'h1, field4:10}
Value of w -> 6.000000 => '{voltage:0.1, current:0.2, field3:'h1, field4:10}
Declaring User-Defined Net with Resolution Function

- A user-defined SystemVerilog nettype with its resolution functions can be declared as:

```systemverilog
nettype data_type nettype_identifier with
    [package_scope|class_scope] tf_identifier ;
```

- `nettype_identifier` is the identifier you specify for the nettype.
- `[package_scope|class_scope] tf_identifier` can be a Cadence built-in resolution function or any `typedef` to the built-in real type `nettype data_type nettype_identifier with [package_scope|class_scope] tf_identifier ;`.

//Declaring a UDT nettype with UDR
```systemverilog
nettype T wTsum with Tsum;
```

// user-defined data type T
typedef struct {
    real field1;
    real field2;
} T;

// user-defined resolution function Tsum
```systemverilog
function automatic T Tsum (input T driver[]);
    Tsum.field1 = 0.0;
    Tsum.field2 = 0.0;
    foreach (driver[i]) begin
        Tsum.field1 += driver[i].field1;
        Tsum.field2 += driver[i].field2;
    end
endfunction
```
## User-Defined Nettype Example

### Data Type and Resolution Function (As a Package)

```plaintext
package temp_pkg;

// user-defined data type T
typedef struct {
    real field1;
    real field2;
} T;

// user-defined resolution function Tsum
function automatic T Tsum (input T driver[]);
    Tsum.field1 = 0.0;
    Tsum.field2 = 0.0;
    foreach (driver[i]) begin
        if (driver[i].field1 !== `wrealZState)
            Tsum.field1 += driver[i].field1;
        if (driver[i].field2 !== `wrealZState)
            Tsum.field2 += driver[i].field2;
    end
endfunction

// A nettype declaration with datatype and resolution function
nettype T wTsum with Tsum;
endpackage
```

### Model

```plaintext
import temp_pkg::*;

module top;
    wTsum w;
    T myvar;
    assign myvar = w;
    driver1 d1(w);
    driver2 d2(w);
    receiver1 r1(w);
endmodule

module receiver1 (input wTsum rec_1);
    always @(rec_1.field1, rec_1.field2)
        $display($time, ," sum = %f  flag = %f \n", rec_1.field1, rec_1.field2);
endmodule

module driver1 (output wTsum dr_1);
    assign dr_1 = T'{1.0, 2.0};
endmodule

module driver2 (output wTsum dr_2);
    assign dr_2 = T'{3.0, 4.0};
endmodule
```

**Resolved** 

{4.0, 6.0}
Electrical Package in SystemVerilog

- An Electrical Package for Systemverilog *(EE_pkg.sv)* defines an electrical equivalent net (V-I-R) for use in discrete analog behavioral models.
- You can use the new EE_pkg package to port existing wreal models to SV.

- Describes the structure **EEstruct** (UDT) which consists of three reals namely V, I and R.

```cpp
package EE_pkg;

// Struct to define Voltage, current, and resistance
typedef struct {
    real V;
    real I;
    real R;
} EEstruct;
```

- Has a UDR function that describes how the resolution of V, I and R are resolved, *res_EE*.
- This package ends with the nettype declaration statement:
- The **EEnet** will conform to Kirchoff's laws.

```
nettype EEstruct EEnet with res_EE;
```
Case Study 1: N-Fractional PLL Mixed Signal

Diagram of a fractional PLL mixed signal voltage regulator:
- Voltage Regulator
- Charge Pump
- Loop Filter
- VCO
- Divider
- Level Shifter
- ESD
- PFD
- 2MHz refclk Level Shifter
- \( \Sigma \Delta \) Modulator + Digital Control
Case Study 1: N-Fractional PLL Mixed Signal
Case Study 1: N-Fractional PLL Mixed Signal

- Loop Filter Voltage output (Verilog-AMS vs. SV EE_pkg)

<table>
<thead>
<tr>
<th></th>
<th>SV-RNM</th>
<th>VAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Time</td>
<td>47 seconds</td>
<td>1 hr 8 min. 32 sec</td>
</tr>
</tbody>
</table>

A speed gain of 90x over mixed-signal Verilog-AMS
Case Study 2: 3rd – order Feed-forward Gm-C ∆Σ ADC

High-level Sizing and frequency scaling

Schematic of 3rd – order Gm-C ∆Σ ADC
Case Study 2: 3rd-order Feed-forward Gm-C $\Delta \Sigma$ ADC

Simulation results for input signal $= 80\text{mV}$
Case Study 2: 3rd-order CIFF Gm-C \( \Delta \Sigma \) ADC

Simulation results for \( a_{in} = 80mV \)

- **Spectrum Assistant** has been used in ViVA to evaluate various spectrum properties, e.g. SINAD, ENOB, THD, etc.

<table>
<thead>
<tr>
<th></th>
<th>SV-RNM</th>
<th>VAMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQNR</td>
<td>72.92 dB</td>
<td>72.33 dB</td>
</tr>
<tr>
<td>SINAD</td>
<td>71.06 dB</td>
<td>72.33 dB</td>
</tr>
<tr>
<td>ENOB</td>
<td>11.515</td>
<td>11.72</td>
</tr>
<tr>
<td>THD %</td>
<td>18.19m %</td>
<td>8.1m %</td>
</tr>
<tr>
<td>Noise Floor (per sqrt Hz)</td>
<td>-126 dB/sqrt Hz</td>
<td>-125.3 dB/sqrt Hz</td>
</tr>
<tr>
<td>CPU Time</td>
<td>0.4 seconds</td>
<td>92.5 seconds</td>
</tr>
</tbody>
</table>

A speed gain of 230x over mixed-signal Verilog-AMS
# Agenda

1. Metric-Driven Verification for MS
2. Verification Planning and Management in MS
3. Universal-Verification Methodology for MS
4. Real-number Modeling Capabilities
5. Analog and MS Assertions
6. Q&A
Analog and MS Assertions
Ahmed Osman
## Automation & re-use thru Assertions in Digital, Analog, and Mixed Signal

<table>
<thead>
<tr>
<th>Why Assertions?</th>
<th>Language Support</th>
<th>Not New for Analog</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assume</td>
<td>SVA</td>
<td>Device checks</td>
</tr>
<tr>
<td>Assert</td>
<td>PSL</td>
<td>Spectre MDL</td>
</tr>
<tr>
<td>Cover</td>
<td></td>
<td>$cds_get_analog_value</td>
</tr>
</tbody>
</table>

### Data Converters
- e.g. Monotonicity, DNL, comparator meta-stability

### Digitally-assisted Analog
- e.g. Calibration / process variability compensation

### Systems with Feedback
- PLL: e.g. PLL lock-in time, Output frequency tuning
- Sigma-Delta: e.g. Integrator stability, presence of tones

### Multiple Modes
- Power modes, programmable gain, adaptive filters
Analog / Mixed-signal PSL Assertions

- Real Assertion (using RNM data type)
  - PSL with explicitly declared *wreals*
  - SVA using real variable

```plaintext
real vin;
// psl vin_check : assert always ( 1.2 < vin && vin < 1.3 )
// @(posedge clk);
```

- Analog Assertion (electrical domain behavior)
  - PSL or `e` containing analog objects or access functions or operators
  - (This is not possible in SVA since there is no analog object allowed in SV)

```plaintext
electrical vin;
// psl vin_check : assert always ( 1.2 < V(vin) && V(vin) < 1.3 )
// @( cross(V(clk)-1.25));
```
Analog PSL assertions:
Verification Unit

• Verification units in PSL can contain analog objects
• Write your PSL statements/vunit into a file, e.g. inv_vams.pslvlog
• Example:

```verilog
testbench bench
  input in1;
  output out1;
  electrical in1, out1;

  assign in1 = 1.2;
  assign out1 = 0;

  vunit inv_vams_inst_vunit(INV_vams)
  {
    // psl assert
    // always ( V(out1) < 1.25 )
    // @( cross(V(in1)-1.25));
  }
endmodule

module INV_vams ( out1, in1 );
  output out1;
  input in1;
  electrical in1, out1;

  analog begin
    if (V(in1) >= 1.25)
      V(out1) <+ 0.0;
    else
      V(out1) <+ 2.5;
  end
endmodule
```
Demo
Questions