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

Advanced UVM Based Chip Verification Methodologies with Full Analog Functionality

Simul Barua
Ulkasemi, Inc.

FNU Farshad
Ulkasemi, Inc.

Henry Chang
Designer's Guide Consulting, Inc.

Agenda

- Traditional full-chip verification flow
- Analog behavioral modeling
- AMS and DMS modeling approaches
- Case study: full-chip verification of a SoC with analog functionality
- Simulation results
- Trade-offs between the modeling approaches
- Summary & Conclusion

Traditional Full-Chip Verification Flow

- Traditional UVM-based chip-level verification environments
  - Focuses mainly on digital functionality
  - Analog design assumed correct, or signals tied to logic 1 or 0
- Integrating full analog functionality
  - Ensures complete functionality
  - Reduces the risk of failure
- Critical challenges of integrating analog functionality
  - UVM integration requires additional time and efforts
  - Slower simulation time of SPICE-based analog designs

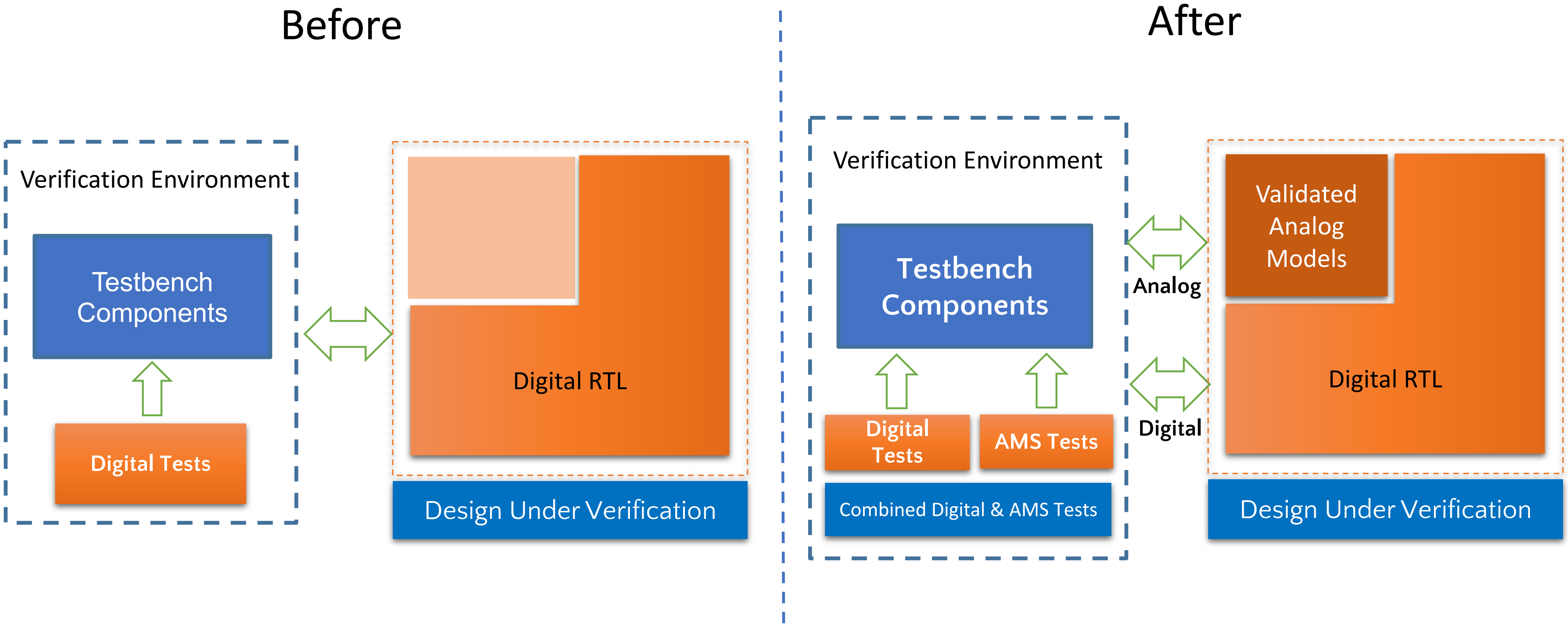
Example Bugs with Full Analog Functionality

- Analog supply and bias connection bugs
- Interface bugs
- Basic functional bugs
- Bugs in the programming of the analog
- Analog signal flow bugs
- Bugs in digital/analog dataflow

Analog Behavioral Modeling Solutions

- To overcome the issues with SPICE schematics
- Written using EDA tool supported languages
- Can be validated against the schematic
- Easy to integrate into UVM based chip-level verification environments
- Faster simulation time
- Enables combined digital and analog testcases

Illuminating the Analog Section

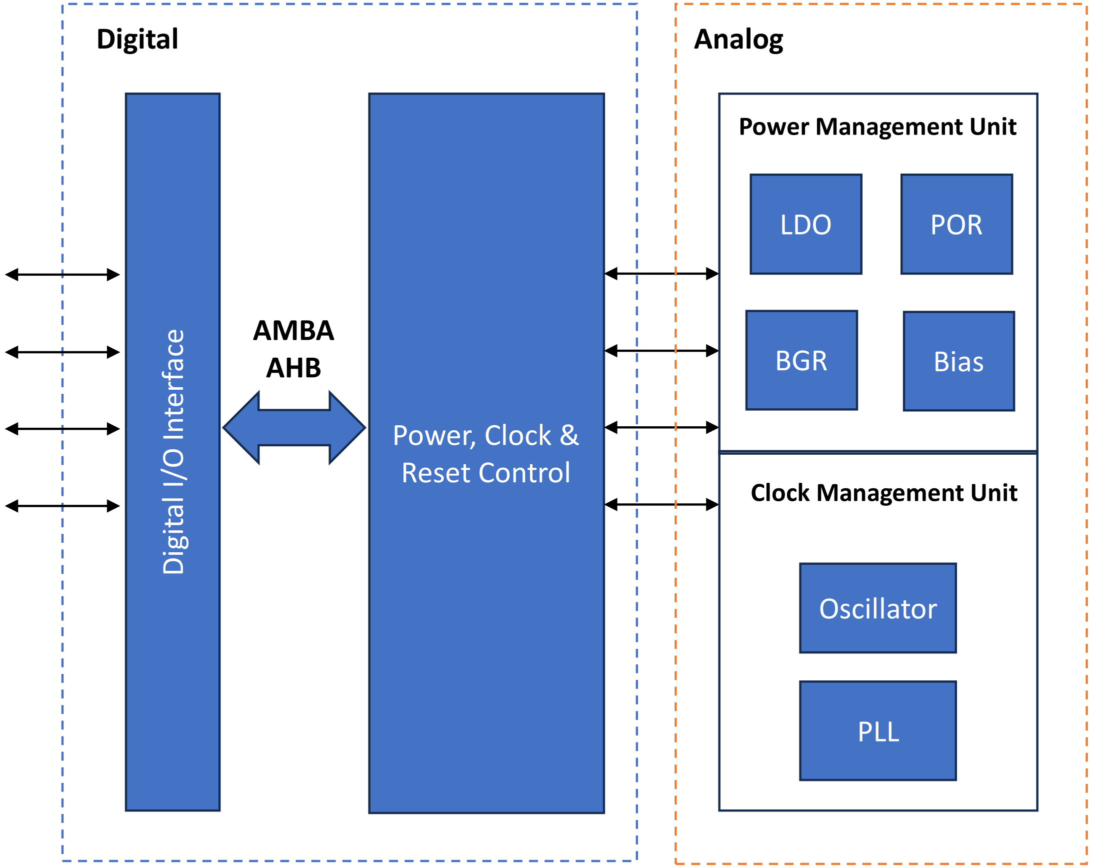


Different Analog Modeling Approaches

- Analog mixed-signal (AMS) approach using Verilog-AMS
  - Allows modeling of continuous and discrete signals
  - Requires both continuous and discrete solvers to run
  - Extra effort is needed to integrate into the UVM testbench environment
- Digital mixed-signal (DMS) approach using SystemVerilog
  - Continuous signals are modeled as real-valued numbers
  - Real number modeling (SV-RNM)
  - Discrete electrical modeling using user-defined nettype (SV-UDN)
  - Easy to integrate into the UVM testbench environment

Case Study: Full-Chip Verification of a SoC with Analog Functionality

- Behavioral model development of the analog subsystem
  - Using AMS, SV-RNM, and SV-UDN approaches
- Integration to UVM chip-level verification environment
- Assess the performance and challenges associated with each modeling approach



Modeling Process of Analog Subsystem

- Fully functional analog models
  - Supply and bias behavior
  - Analog assertions
- The Models-in-Minutes (MiM) tool
  - Speed up and automate the block-level model generation
  - Block-level self-checking testbench
  - Scripts to run model vs. schematic simulation
- Validation of the models against the schematics

MiM Specification of a VCO

```
PARAMETERS:
    cell: vco
PORTS:
    out:
        description: Output
        type: digital output
        behavior: osc
    in:
        description: Input
        type: voltage input
        nominal: 1.25V
    [7:0] vcoCF:
        description: Trimmer
        type: digital input
        behavior: vcoCF_val with protect
        nominal: 128
    enable:
        description: Enable
        type: digital input
        behavior:
            > alive = (enable === 1) && !Fault.
            > Enable = alive with smooth.
        nominal: 1
    bias:
        description: Bias current
        type: ibias input
        range: 5uA to 20uA
        nominal: 10uA
    vdda:
        description: Analog supply
        type: supply input
        range: 1.6V to 1.9V
        behavior: I = 10uA*Enable
        nominal: 1.8
    gnda:
        description: Analog ground
        type: ground input
        range: -10mV to 10mV
        nominal: 0.0
DISCRETE VARIABLES:
    Ctrl:
        description: clipped control voltage
        type: real
        value: clip(V(in) - 1.25, -500m, 500m)
        trigger: posedge osc
        units: V
    F0:
        description: computed center frequency
        type: wreal
        value: 2.374G + 500k*vcoCF_val
    f:
        description: computed output frequency
        type: wreal
        value: F0 + (25M*Ctrl)
        units: Hz
    osc:
        description: oscillator output
        type: reg
        value: #(0.5/f) ~osc
        enable: alive
        initial: 0
```

Generated Behavioral Models

AMS Model

```
`timescale 1s/1ps
`include "disciplines.vams"
`include "constants.vams"
module vco (
    out,
    in,
    vcoCF,
    enable,
    bias,
    vdda,
    gnda
);
output out;                       // Output
input in; electrical in;          // Input
input[7:0] vcoCF;                 // Trimmer
input enable;                     // Enable
input bias; electrical bias;      // Bias current
input vdda; electrical vdda;      // Analog supply
input gnda; electrical gnda;      // Analog ground
// DISCRETE BEHAVIOR {{{1
assign vcoCF_val = ^vcoCF !== 1'bX ? vcoCF : 0;
assign alive = (enable === 1) && !Fault;
assign Enable$safe = alive;
always @(posedge osc)
    Ctrl <= min(max(V(in) - 1.25+0, -500m+0), 500m+0);
assign F0 = 2.374G + 500k*vcoCF_val;
assign f = F0 + (25M*Ctrl);
always wait (alive)
    osc = #(`fromSeconds(max(500m/f, `TIME_PREC))) ~osc;
// Port Drivers {{{2
assign out = osc;
// Analog Behavior {{{2
analog begin
    ..................
    Enable = transition(Enable$safe, 0, 10n);
    I(vdda$br) <+ 10u*Enable;
end
endmodule
```

SV-RNM Model

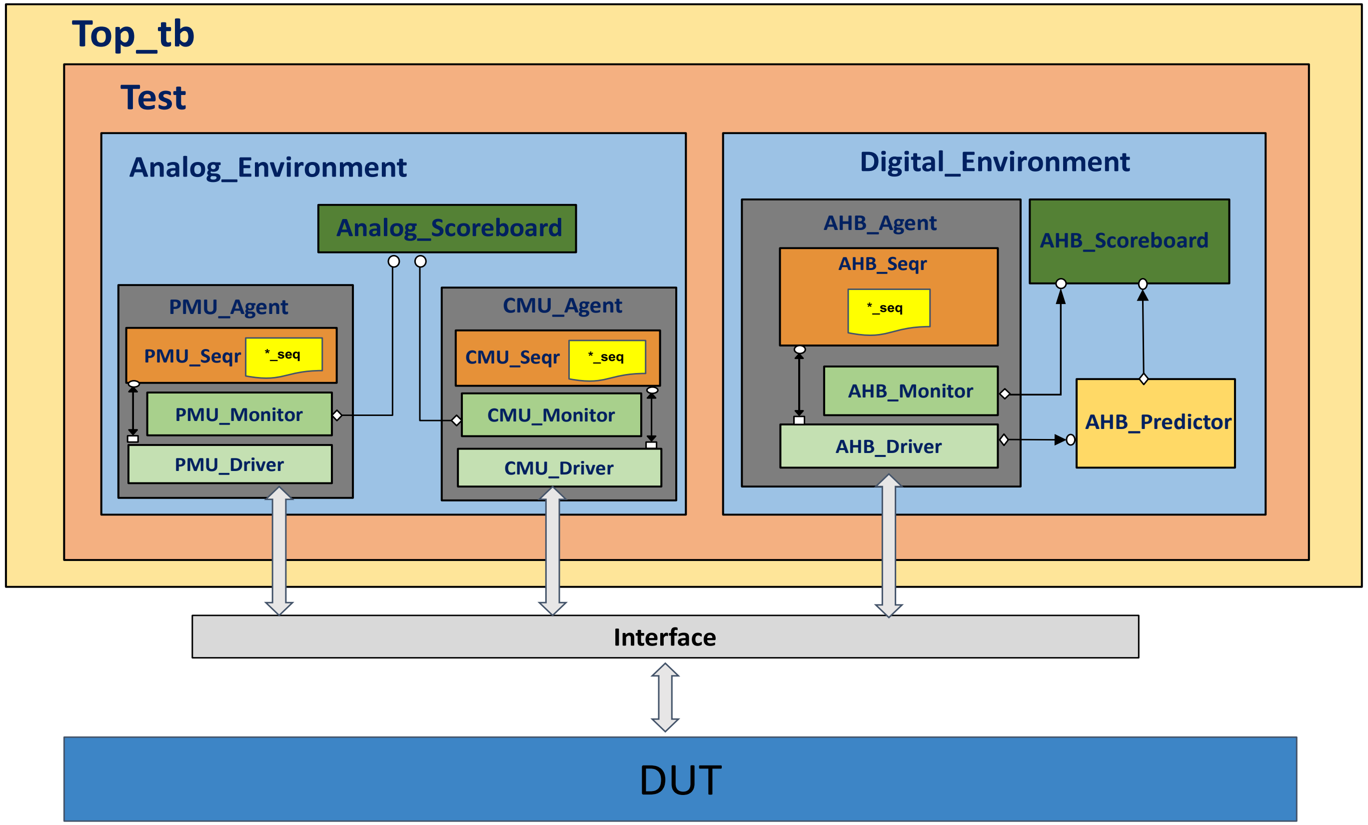
```
`timescale 1s/1ps
// MODULE HEADER {{{1
module vco (
    out,
    in,
    vcoCF,
    enable,
    bias,
    vdda,
    gnda
);
output out;           // Output
input real in;        // Input
input[7:0] vcoCF;     // Trimmer
input enable;         // Enable
input real bias;      // Bias
input real vdda;      // Analog Supply
input real gnda;      // Analog Ground

// Discrete variables {{{2
..................

// DISCRETE BEHAVIOR {{{1
// Discrete variable assignments {{{2
assign vcoCF_val = (^vcoCF !== 1'bX ? vcoCF : 0);
assign Enable = (enable === 1) && !Fault;
assign Ctrl = clip(V(in) - 1.25, -500e-3, 500e-3);
assign F0 = 2.374e9 + (500e3*vcoCF_val);
assign f = F0 + (25e6*Ctrl);
always wait(Enable)osc <= #(0.5/f) ~osc;
// Drive ports {{{2
assign out = osc;
..................
endmodule
```

SV-UDN Model

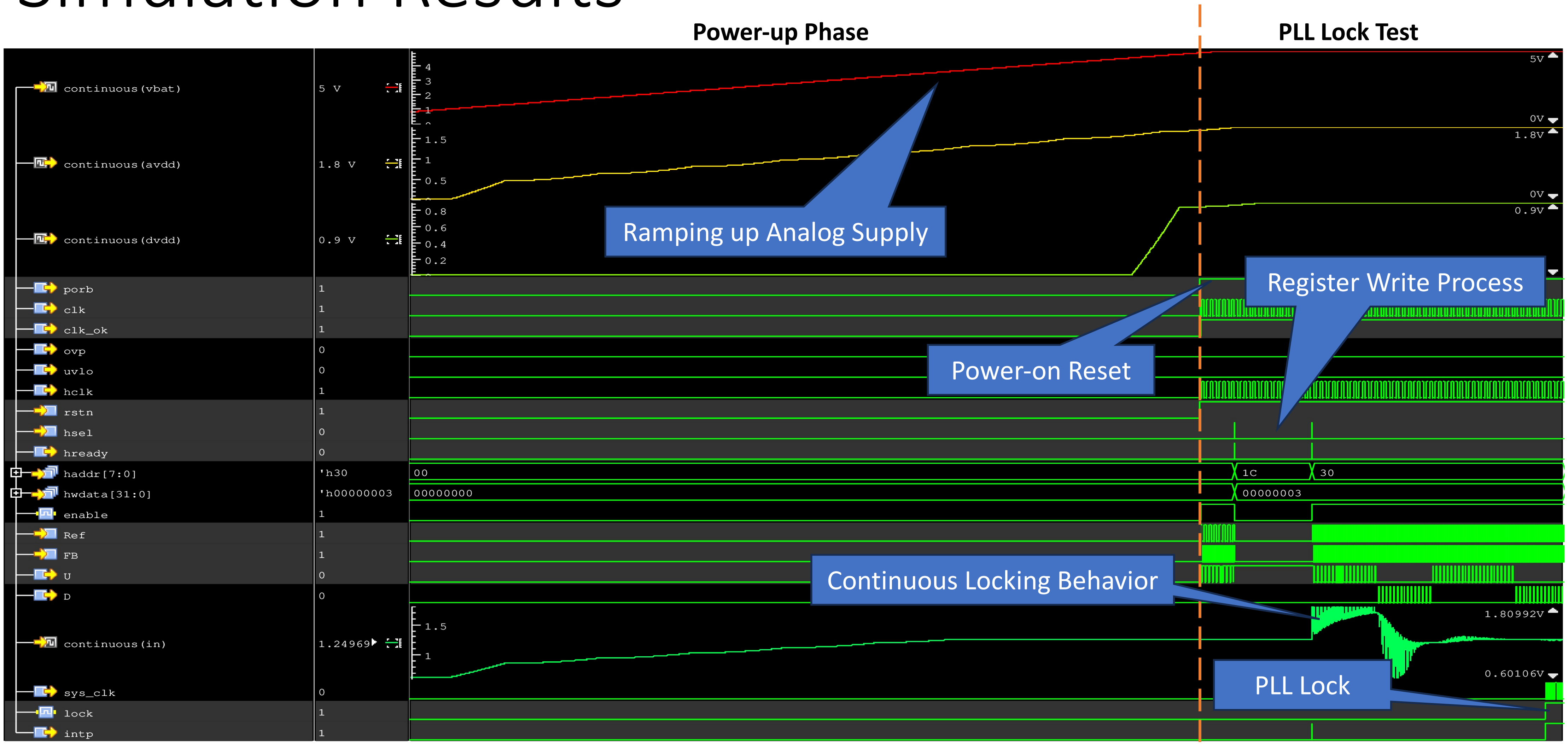
```
`timescale 1s/1ps
// MODULE HEADER {{{1
module vco (
    output out,            // Output
    input in,              // Input
    input[7:0] vcoCF,      // Trimmer
    input enable,          // Enable
    input bias,            // Bias current
    input vdda,            // Analog supply
    input gnda             // Analog ground
);
// Discrete-Electrical (DE) Transceivers {{{2
real in$Vobs, in$Iobs, in$Idrv, in$Gdrv;
DE_norton Xtcvr_in (in,
    in$Vobs, in$Iobs, in$Idrv, in$Gdrv);
real bias$Vobs, bias$Iobs, bias$Vdrv, bias$Rdrv;
DE_thevenin Xtcvr_bias (bias,
    bias$Vobs, bias$Iobs, bias$Vdrv, bias$Rdrv);
real vdda$Vobs, vdda$Iobs, vdda$Idrv, vdda$Gdrv;
DE_norton Xtcvr_vdda (vdda,
    vdda$Vobs, vdda$Iobs, vdda$Idrv, vdda$Gdrv);
real gnda$Vobs, gnda$Iobs, gnda$Idrv, gnda$Gdrv;
DE_norton Xtcvr_gnda (gnda,
    gnda$Vobs, gnda$Iobs, gnda$Idrv, gnda$Gdrv);
// DISCRETE BEHAVIOR {{{1
assign vcoCF_val = ^vcoCF !== 1'bX ? vcoCF : 0;
assign alive = (enable === 1) && !Fault;
assign Enable$safe = alive;
always @(posedge osc)
    Ctrl <= min(max(in$Vobs - 1.25+0, -500e-3+0), 500e-3+0);
assign F0 = 2.374e9 + 500e3*vcoCF_val;
assign f = F0 + (25e6*Ctrl);
always wait (alive)
    osc = #(`fromSeconds(max(500e-3/f, `TIME_PREC))) ~osc;
assign Enable = Enable$safe;
assign vdda$Idrv = 10e-6*Enable;
assign out = osc;
endmodule
```

Chip-Level UVM Verification Environment

- Analog environment
  - PMU_Agent
  - CMU_Agent
- Digital environment
  - AHB_Agent
- Chip-level testcases
  - Initiate power-up
  - Configure the PLL
  - Detect PLL lock interrupt



Simulation Results



Ramping up Analog Supply

Register Write Process

Power-on Reset

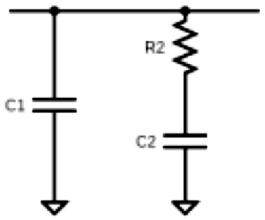
Continuous Locking Behavior

PLL Lock

Comparison Between the Modeling Approaches

- AMS approach supports accurate and straightforward analog circuitry modeling
  - Nodes and branches
  - Available system functions for complex mathematical operations
  - Accurate power consumption modeling
- Both the DMS approaches add extra complexity to modeling continuous behavior
  - Require discrete-time transfer functions for complex behavior
  - SV-RNM supports the modeling of either voltage or current

AMS vs DMS Model of the PLL Loop Filter



```
`include "disciplines.vams"
`include "constants.vams"
`timescale 1s/1ps
// MODULE HEADER {{{1
module LF (
    out,
    in,
    gnda
);
output out; electrical out;   // Output
input in; electrical in;      // Input
input gnda; electrical gnda;  // Ground

// Internal Node declarations {{{2
electrical n;                 // Node between R2 & C2
// Branch declarations {{{2
branch (in, out) short;       // Short between in & out
branch (out, gnda) C1;        // Capacitor C1
branch (out, n) R2;           // Resistor R2
branch (n, gnda) C2;          // Capacitor C1

// ANALOG BEHAVIOR {{{1
analog begin
    // Branch short: Short between in & out {{{3
    V(short) <+ 0.0;
    // Branch C1: Capacitor C1 {{{3
    I(C1) <+ c1 * ddt(V(C1));
    // Branch R2: Resistor R2 {{{3
    V(R2) <+ (r2)*I(R2);
    // Branch C2: Capacitor C1 {{{3
    I(C2) <+ c2*ddt(V(C2));
end
endmodule
```

```
`timescale 1s/1ps
// MODULE HEADER {{{1
module LF (
    out,
    in,
    gnda
);
output out; real out;        // Output
input in; real in;           // Input
input gnda; real gnda;       // Ground

// Calculating the coefficients found from
// z-transform equation
assign a0 = 2*C1*T+2*C2*T+4*C1*C2*R2;
assign a1 = (-8*C1*C2*R2)/a0;
assign a2 = (-2*C1*T-2*C2*T+4*C1*C2*R2)/a0;
assign b0 = (T**2+2*C2*R2*T)/a0;
assign b1 = (2*T**2)/a0;
assign b2 = (T**2-2*C2*R2*T)/a0;
assign x0 = in;
assign y0 = (x0*b0 + x1*b1 + x2*b2 -y1*a1 - y2*a2);

always begin
    x1 <= x0;
    x2 <= x1;
    y1 <= y0;
    y2 <= y1;
    #(1ns);                   // Sampling delay
end

assign out = (y0>2.5?2.5:(y0<0.0?0.0:y0));
endmodule
```

Integration to UVM Testbench

- SV-RNM and SV-UDN models
  - Easy to integrate
- AMS models
  - Require connectmodules
  - UVM-AMS conversion wrapper to communicate with UVM

```
// Wrapper for PMU
`timescale 1s/1ps
`include "disciplines.vams"
`include "constants.vams"
module PMU_wrapper(
    V_vdd,
    I_vdd,
    .......
)
input V_vdd; wreal V_vdd;
output I_vdd; wreal I_vdd;
................
// Internal Signals
electrical vdd;
................
real I_vdd_real;
// PMU Instantiation
PMU pmu(
    .vdd(vdd),
    ................
)
// Saving current values coming from analog system
always @(absdelta(I(vdd),1n,1n,1u)) I_vdd_real = I(vdd);
assign I_vdd = I_vdd_real;
................
analog begin
    // Driving analog signals
    V(vdd) <+ transition(V_vdd, 0, 10n);
    ................
end
endmodule
```

Electrical to Real Conversion

Real to Electrical Conversion

Performance at the UVM Chip-Level Verification Environment

Modeling Approach	Total Simulation Time
Real Number Models (SV-RNM)	~2 minutes
Discrete Electrical Models (SV-UDN)	~2 minutes
AMS Models	~20 minutes

Summary and Conclusion

- Analog modeling is crucial for full-chip functional verification
- AMS approach
    - Accurate analog functional modeling
    - Poor performance at chip-level verification
- DMS approaches
    - High performance and easy integration to UVM
    - SV-RNM: limitation in detailed functional modeling
    - SV-UDN: steeper learning curve
- A hybrid approach solution employing both the AMS and DMS approach
    - AMS models to find bugs available in continuous-time simulation
    - DMS models to find bugs in RTL and analog communication

Questions

- Any questions?