Extending Proven Digital Verification Techniques for Mixed-Signal SoCs with VCS AMS

Helene Thibieroz, Synopsys
Adiel Khan, Synopsys
Pierluigi Daglio, STMicroelectronics
Gernot Koch, Micronas

accellera SYSTEMS INITIATIVE
2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Agenda

This tutorial includes:

- Introduction to VCS AMS mixed-signal verification solution
- Technical presentation of AMS Testbench
- STMicroelectronics highlights their usage of VCS AMS to accelerate mixed-signal verification using Save and Restore
- Micronas describes their usage of VCS AMS behavioral modeling capabilities for their validation methodology of mixed-signal sensor-based applications.

VCS AMS

Mixed-signal Verification Solution

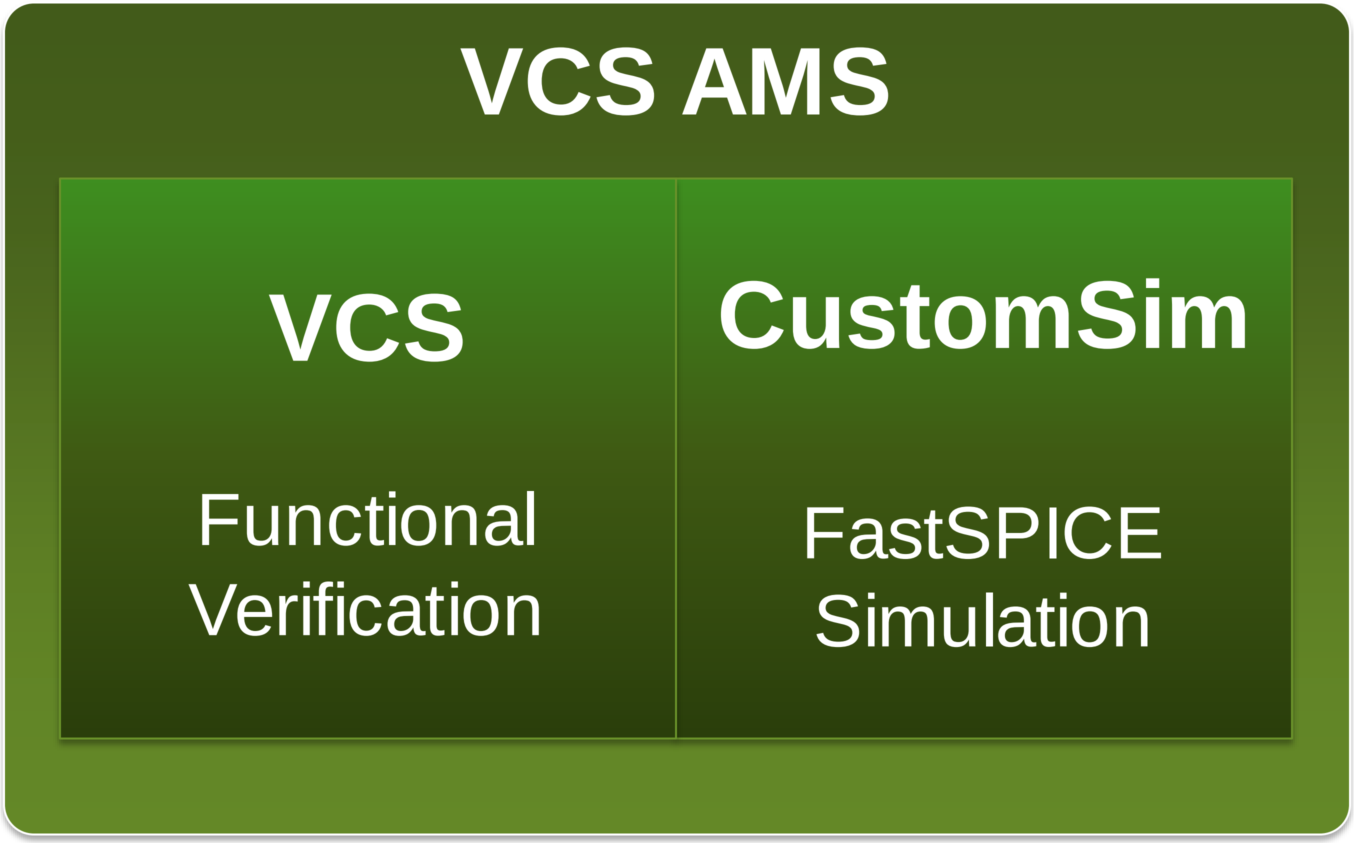
Helene Thibieroz

Product Marketing

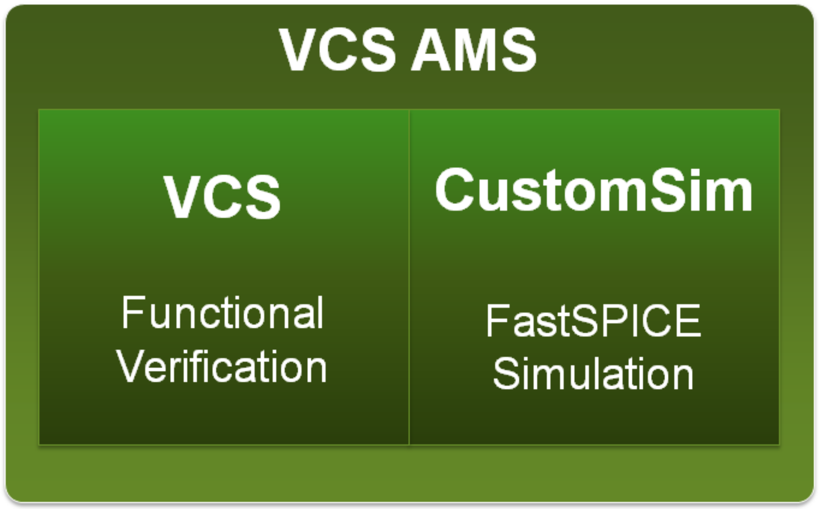
Synopsys

Introducing VCS AMS

Mixed-signal Verification Solution



VCS AMS – Technologies



- Performance
- Flexibility
- Broad Languages
- Debug

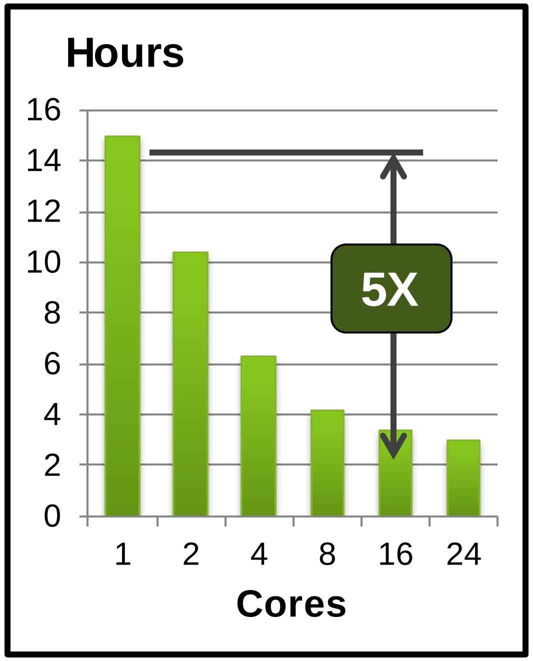
New! **AMS Testbench**

New! **Native Low Power**

VCS AMS – Performance

Best Performance with Transistor-level Accuracy

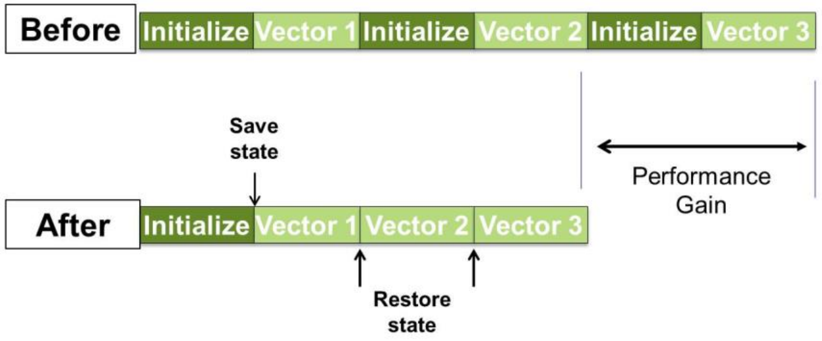
Multicore



15 hours to <u>3 hours</u>

RF RX, 300K tx

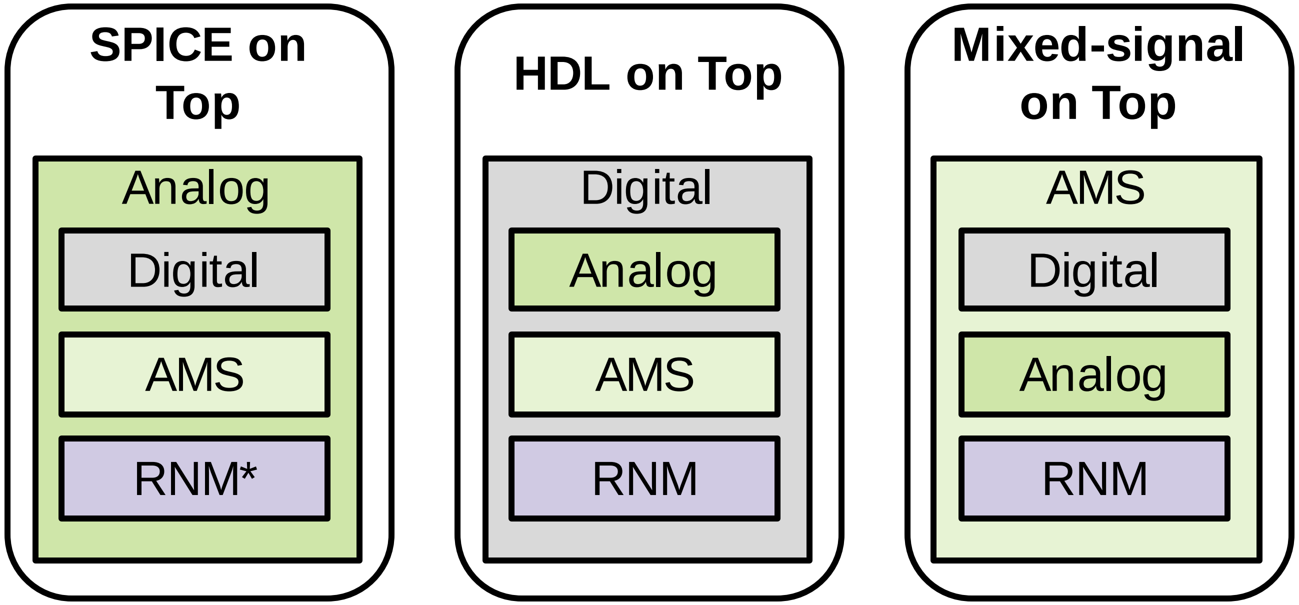
Save and Restore



Enables Mixed-signal Verification for Regression

VCS AMS – Flexibility

Multiple Topologies Offered for Complex SoCs



Multiple Topologies and Configurations

* Real Number Modeling

VCS AMS – Broad Language Support

Enables Complex Integration Schemes for Mixed-signal SoCs

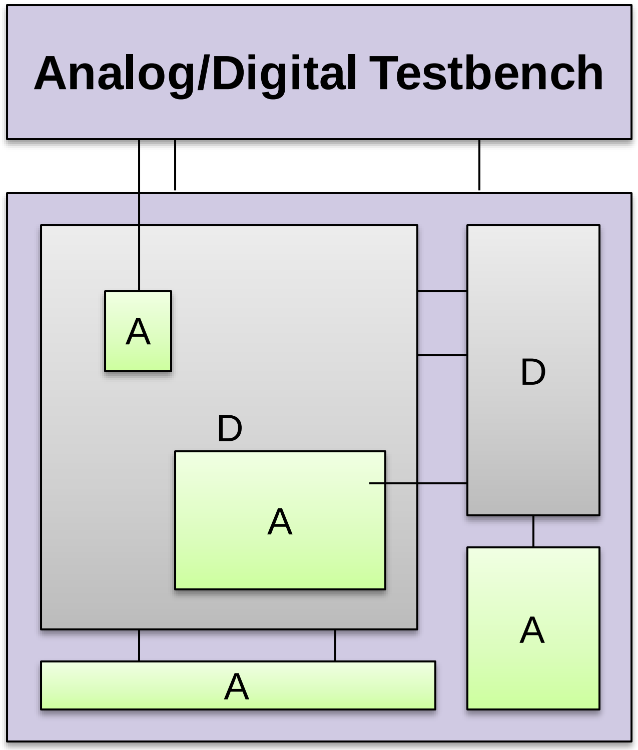
Analog	Digital	Mixed-signal
SPICE	Verilog	Verilog-AMS
Verilog-A	VHDL	Real Number Model
SPEF, DSPF, DPF	SystemVerilog	SystemVerilog nettype
	SystemC, Matlab	

VCS AMS – Debug

Assisted Setup and Debugging

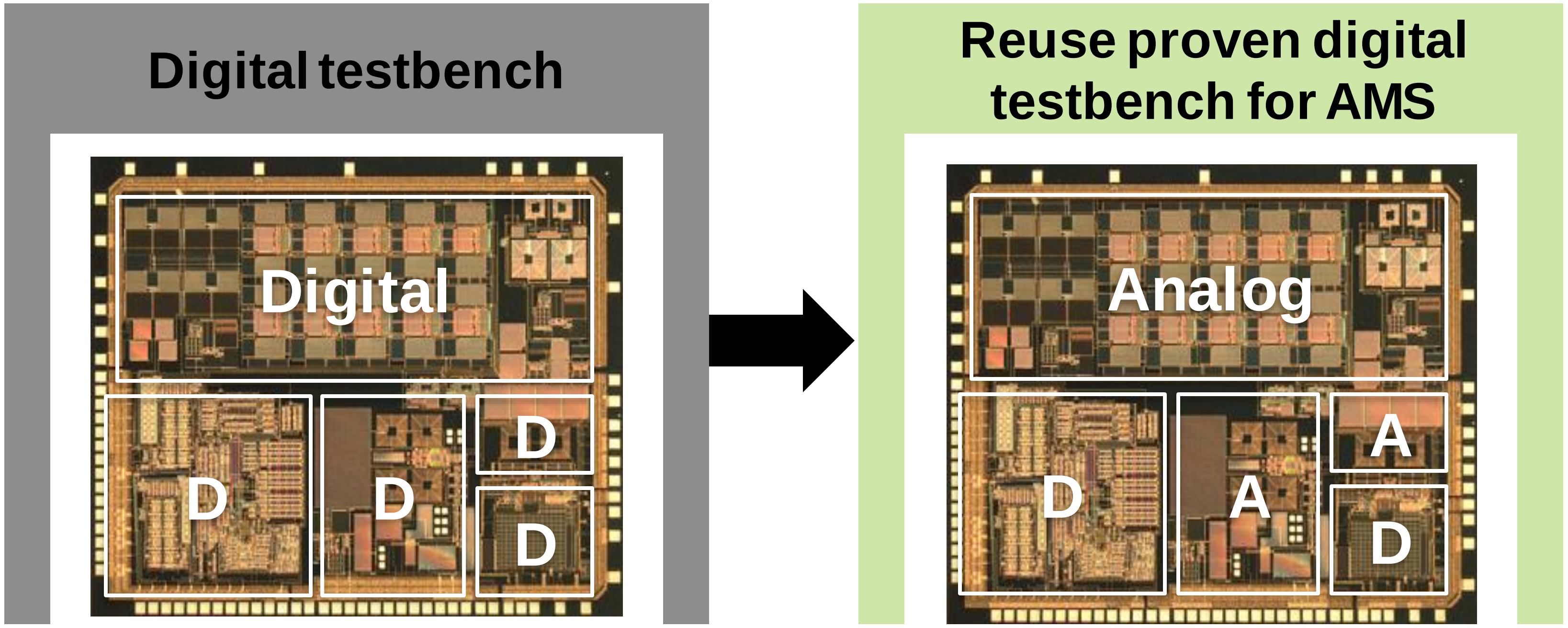
- Easy configuration
  - VCS use model
  - Netlist driven
- Automated insertion of A/D interface elements
  - Optimized for speed and accuracy
- Connectivity reports
  - Interface elements
  - Port mapping

Mixed-signal Testbench



VCS AMS – AMS Testbench

Expanding UVM Methodology for Analog



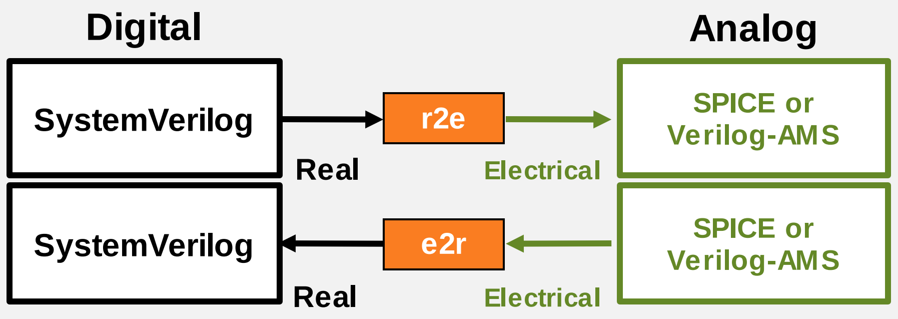
Rapid Development with Lower Risk

VCS AMS – AMS Testbench

Digital Verification Techniques for Mixed-signal SoCs

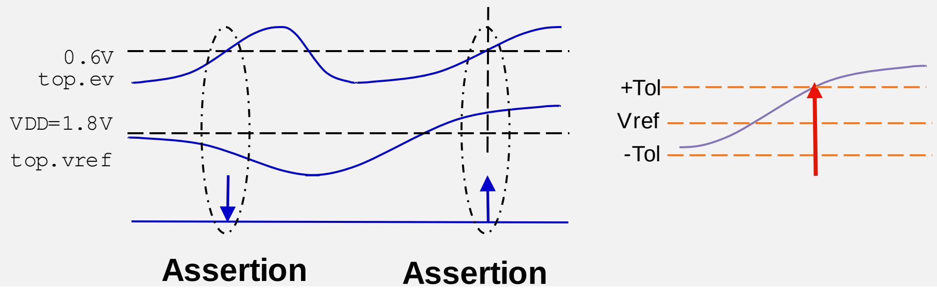
Connectivity

Electrical ⇔ Real conversion
Asynchronous analog events



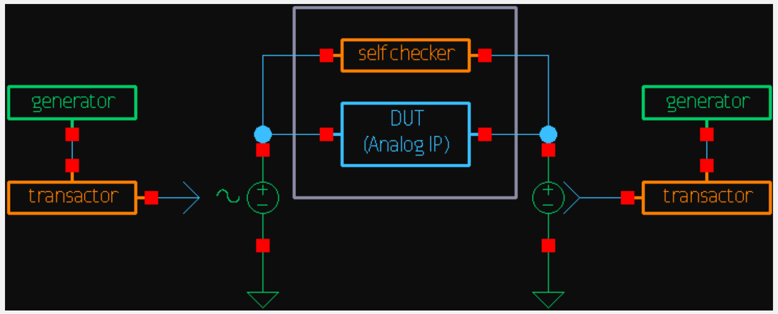
Metric-driven

AMS assertions
AMS constrained-random stimulus
AMS checkers
SystemVerilog real number modeling



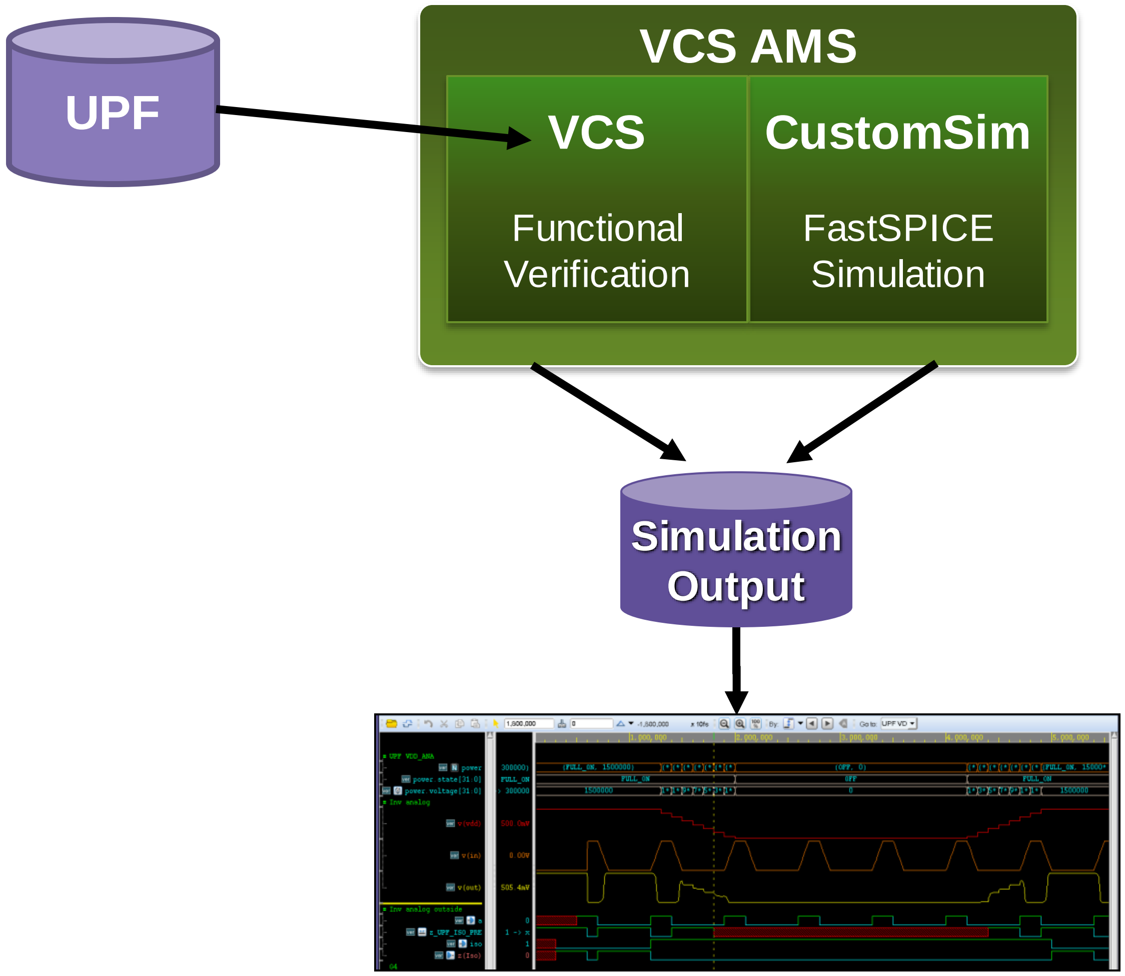
Self-checking

AMS testbench environment
AMS source generators
AMS functional coverage



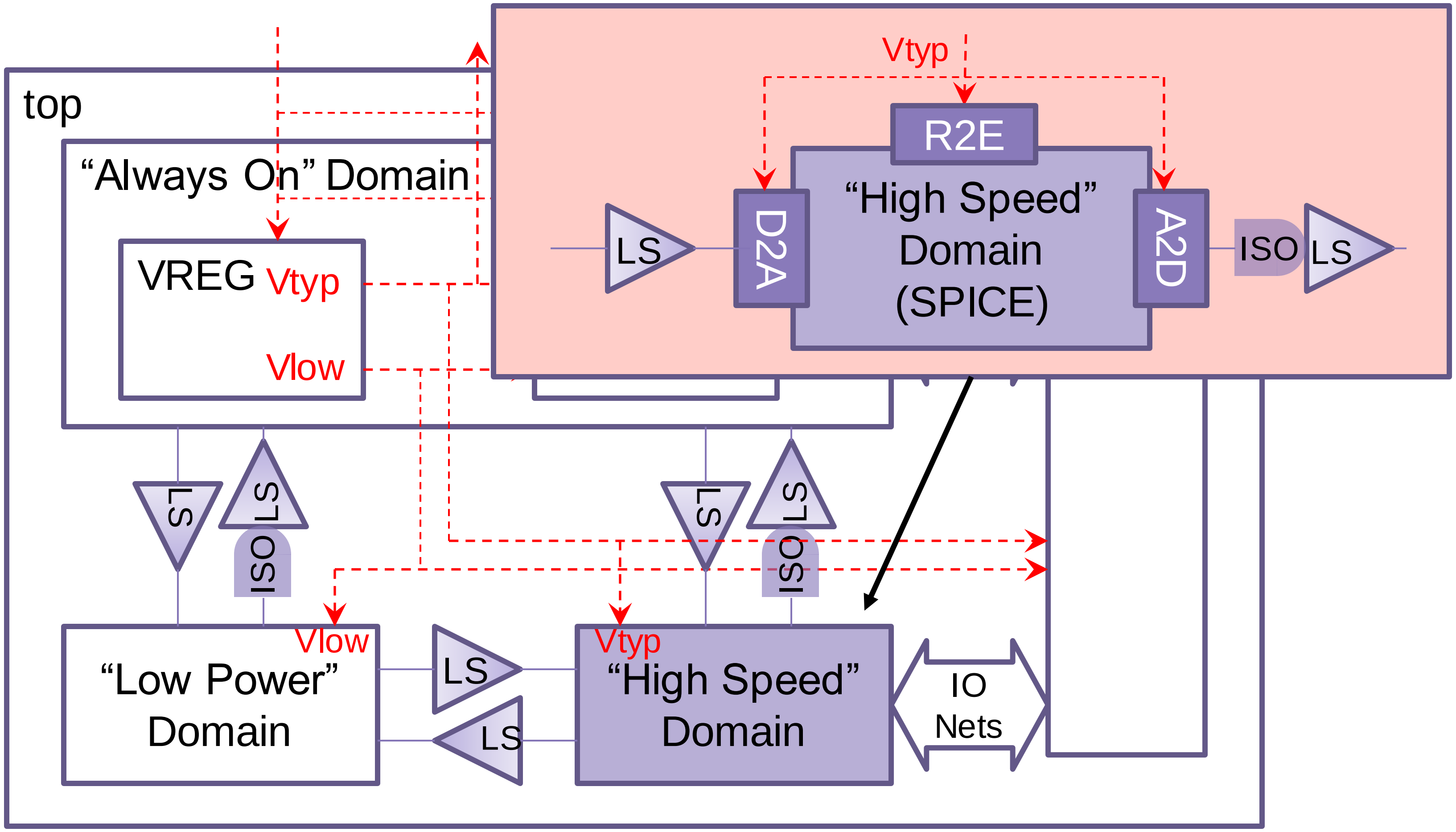
VCS AMS – Low Power Verification

VCS AMS with Native Low Power



VCS AMS – Low Power Verification

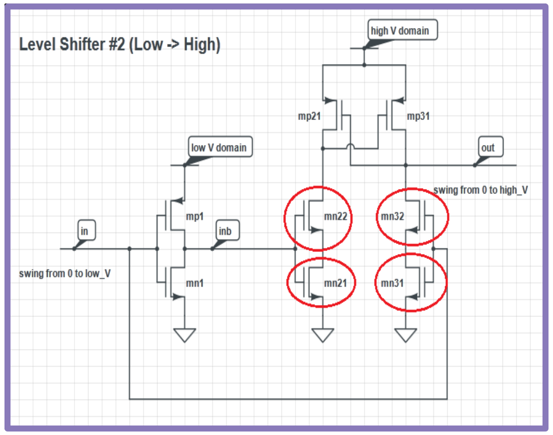
Introducing UPF-based Mixed-signal Verification



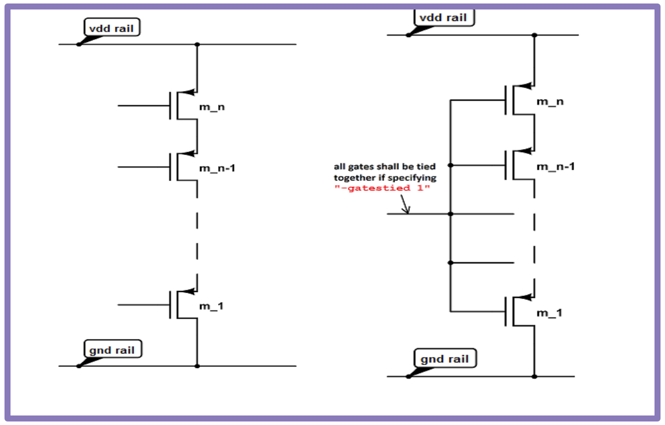
VCS AMS – Low Power Verification

Static Checking with Circuit Check (CCK)

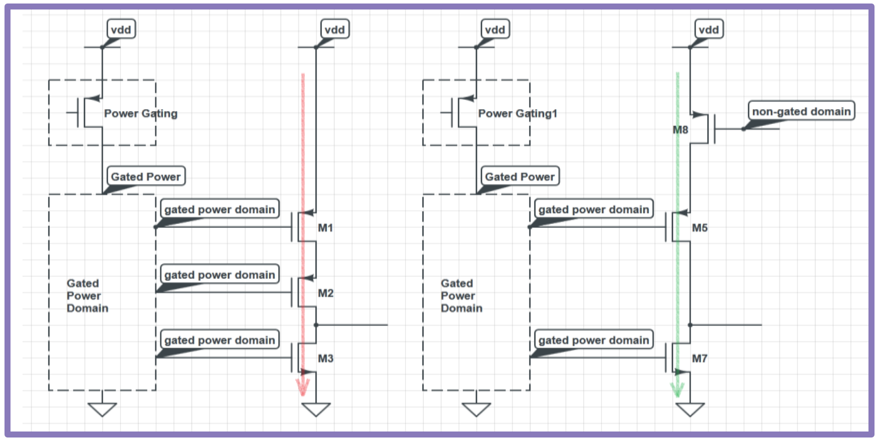
Missing Level Shifter Check



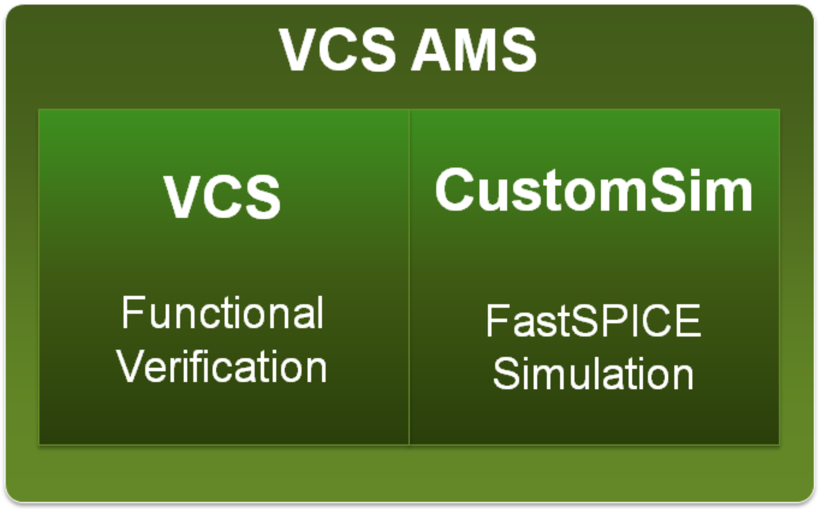
Stacking MOSFET between Rails


vdd rail
m_n
m_n-1
m_1
gnd rail
vdd rail
m_n
all gates shall be tied together if specifying "-gatestied 1"
m_n-1
m_1
gnd rail

Leakage Path Induced by Gated Power



VCS AMS – Summary



- Performance
- Flexibility
- Broad Languages
- Debug

New! **AMS Testbench**

New! **Native Low Power**

AMS Testbench

Extending Digital Verification to Analog

Adiel Khan

Helene Thibieroz

Agenda

Overview

Technical features

Demo

Q&A

AMS Testbench

Overview

What is AMS Testbench?

- AMS Testbench refers to Synopsys proprietary analog extensions to the UVM standard
- Extension of UVM-based techniques for mixed-signal verification

Why AMS Testbench?

- Top-level verification required for mixed-signal SoCs
  - Increasing IP Integration
  - More complex interaction between analog and digital

AMS Testbench is Synopsys solution for mixed-signal coverage-driven verification methodology

- include analog during top-level verification and assume analog block characterization is enough
  - High risk of design re-spins

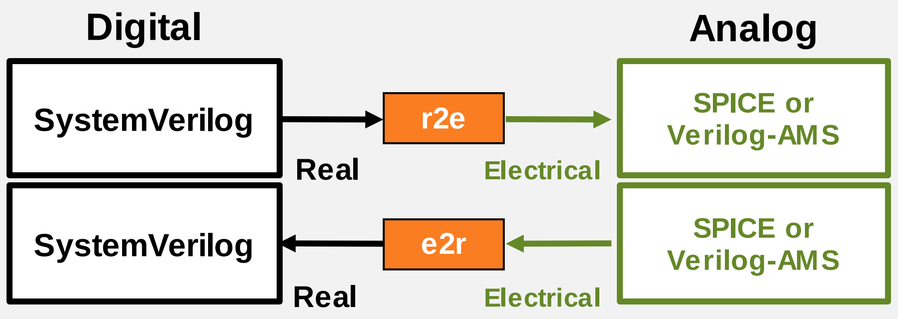
- Flow automation needed to include analog in full-chip verification planning strategy

VCS AMS for Regression

Digital Verification Techniques for Mixed-signal SoCs

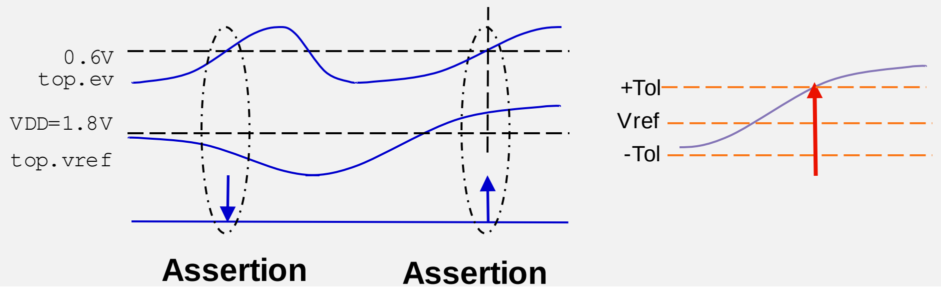
Connectivity

Electrical ⇔ Real conversion
Asynchronous analog events



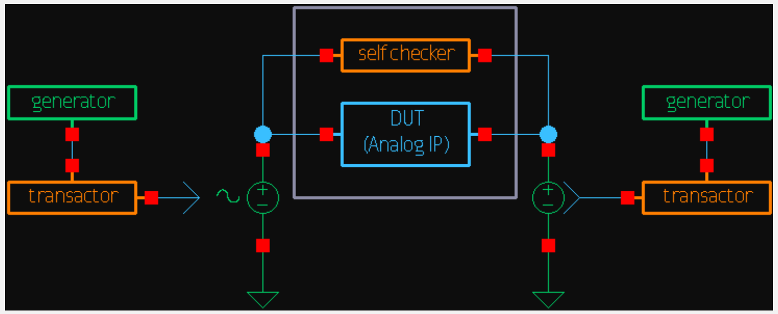
Metric-driven

AMS assertions
AMS constrained-random stimulus
AMS checkers
SystemVerilog real number modeling



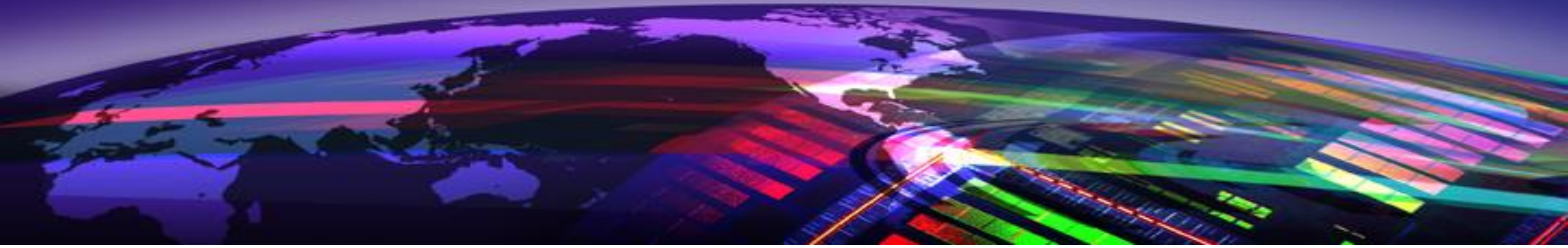
Self-checking

AMS Testbench environment
AMS source generators
AMS functional coverage

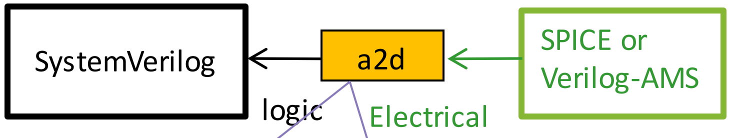


AMS Testbench

Digital Verification Techniques - Connectivity

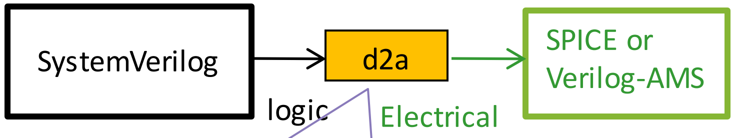


Logic⬄Analog Conversion



Automatic insertion of a2d connect models between SystemVerilog and SPICE

User can redefine the threshold



Automatic insertion of d2a connect models between SystemVerilog and SPICE

User can redefine the threshold

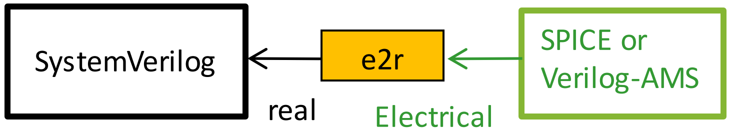
Example:

```
assign verilog_wire =
   top.i1.i2.x1.clk;
initial begin
  verilog_reg =
               top.i1.i2.x1.strb;
...
```

Example:

```
reg rst_reg;
assign top.i1.i2.x1.rst = rst_reg;
initial begin
...
rst_reg = 1'b0;
#5 rst_reg = 1'b1;
...
end
```

Real⇔Analog Conversion



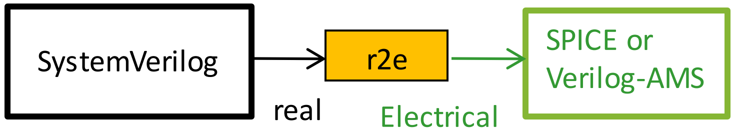
Easy XMR read access to internal analog signal voltage and current

```
$snps_get_volt(anode)
$snps_get_port_current(anode)
```

anode: full hierarchical analog node name

Example:

```
real r;
always @(posedge clk)
  r <=$snps_get_volt(top.i1.ctl);
```



Easy XMR write access to internal analog signal voltage.

```
$snps_force_volt(anode,val|real)
$snps_release_volt(anode)
```

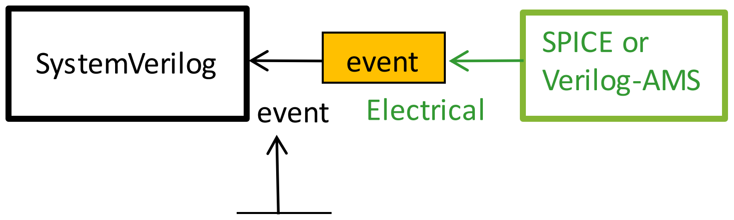
anode: full hierarchical analog node name

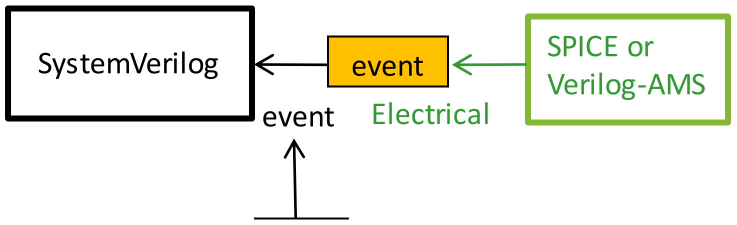
val|real: absolute value or real variable

Example:

```
real r;
initial
 $snps_force_volt(top.i1.ctl,0.0);

always @(posedge clk) begin
  r <= r+0.1;
  $snps_force_volt(top.i1.ctl,r);
```

Asynchronous Analog Events





```
$snps_cross(aexpr[,dir[,time
   _tol [,expr_tol]]]);
```
aexpr: analog expression based on system function

Example:

```
always
   @(snps_cross($snps_get_volt(t
   op.i1.ctl)-0.6,1))
begin
  $display("Signal ctl is raising
   above 0.6V");
```

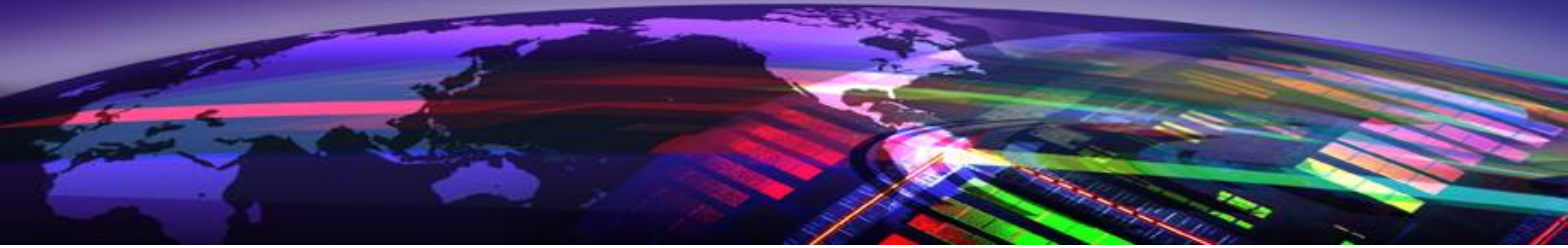
```
$snps_above(aexpr[,time_tol
   [,expr_tol]]]);
```
aexpr: analog expression based on system function

Example:

```
always
   @(snps_above($snps_get_volt(t
   op.i1.ctl)-0.6))
begin
  $display("Signal ctl is above
   0.6V");
```

AMS Testbench

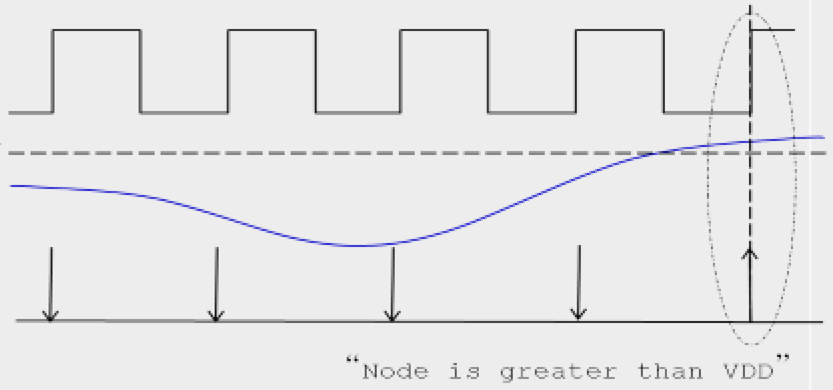
Digital Verification Techniques – Assertions and Checkers



Immediate Assertions

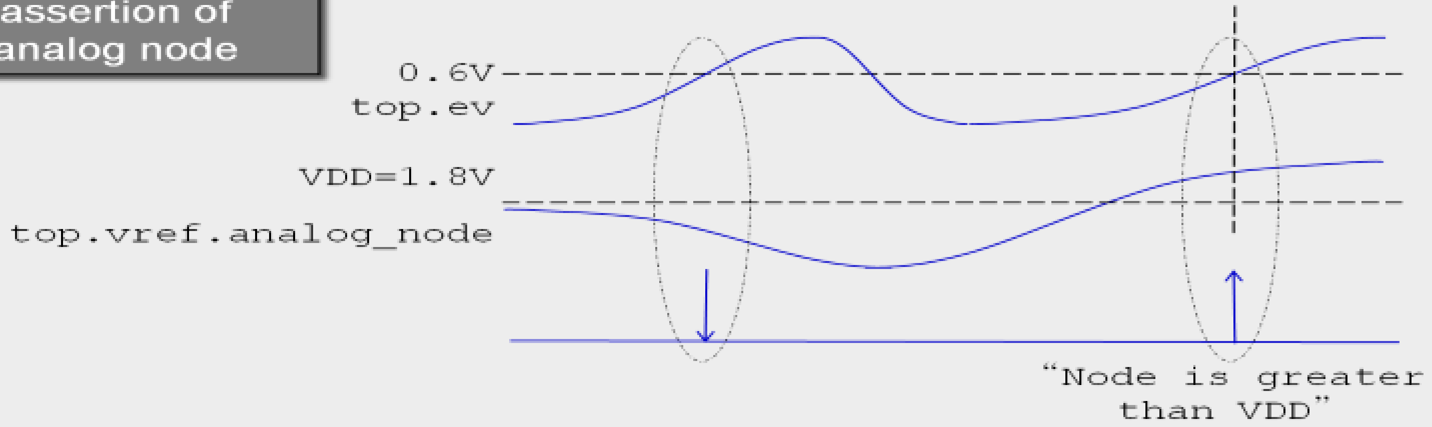
Synchronous immediate assertion of analog node

```
always @(posedege clk)
   assert(top.vref.analog_node <= 1.8)
   else $error("Node is greater than VDD");
```

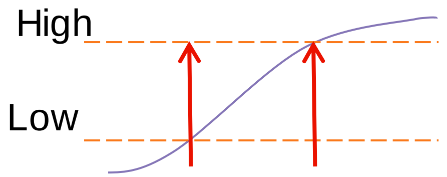
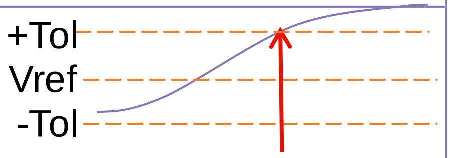
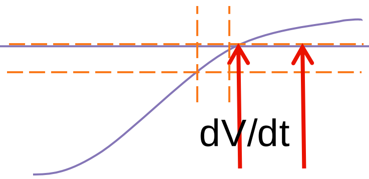
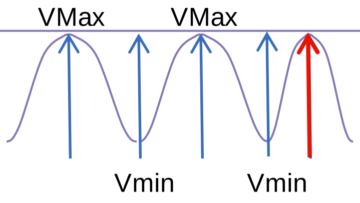


Asynchronous immediate assertion of analog node

```
always @(snps_cross($snps_get_volt(top.ev)-0.6,1))
   assert(top.vref.analog_node <= 1.8)
   else $error("Node is greater than VDD");
```

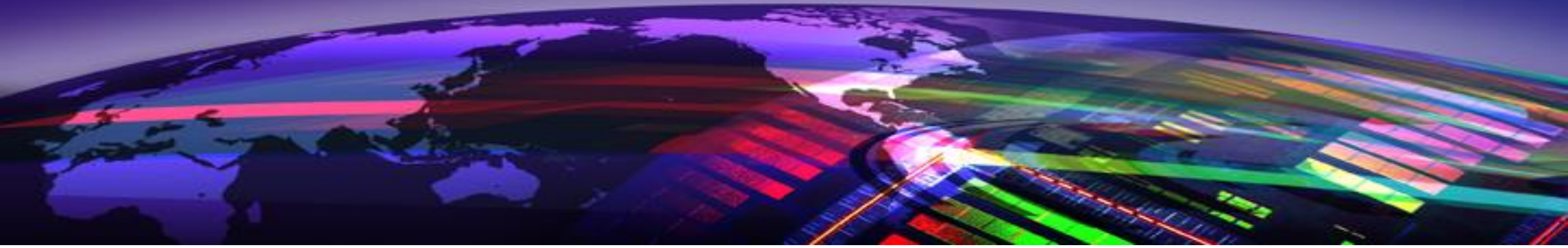


AMS Testbench Checkers

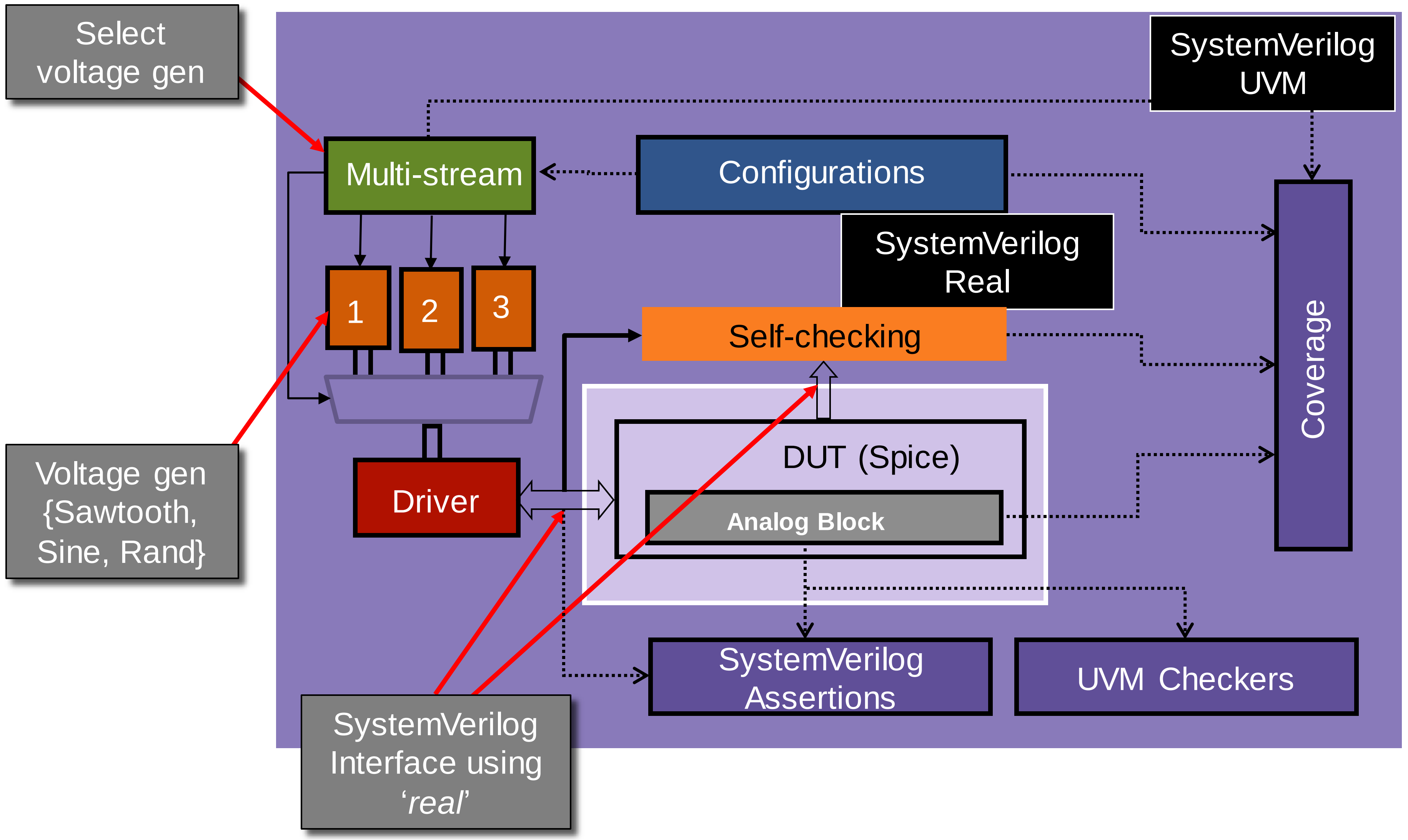
Checkers		
sv_ams_threshold_checker	Checks that analog signal remains within a given high and low threshold. Can perform this check synchronously or asynchronously	
sv_ams_stability_checker	Checks that analog signal remains below or above a given threshold. Can perform this check synchronously or asynchronously	
sv_ams_slew_checker	Checks that analog signal rises/falls with a given slew rate(+/- tolerance). Can perform this check synchronously or asynchronously	
sv_ams_frequency_checker	Checks that analog signal frequency is within a given tolerance	

AMS Testbench

Digital Verification Techniques – Self-checking

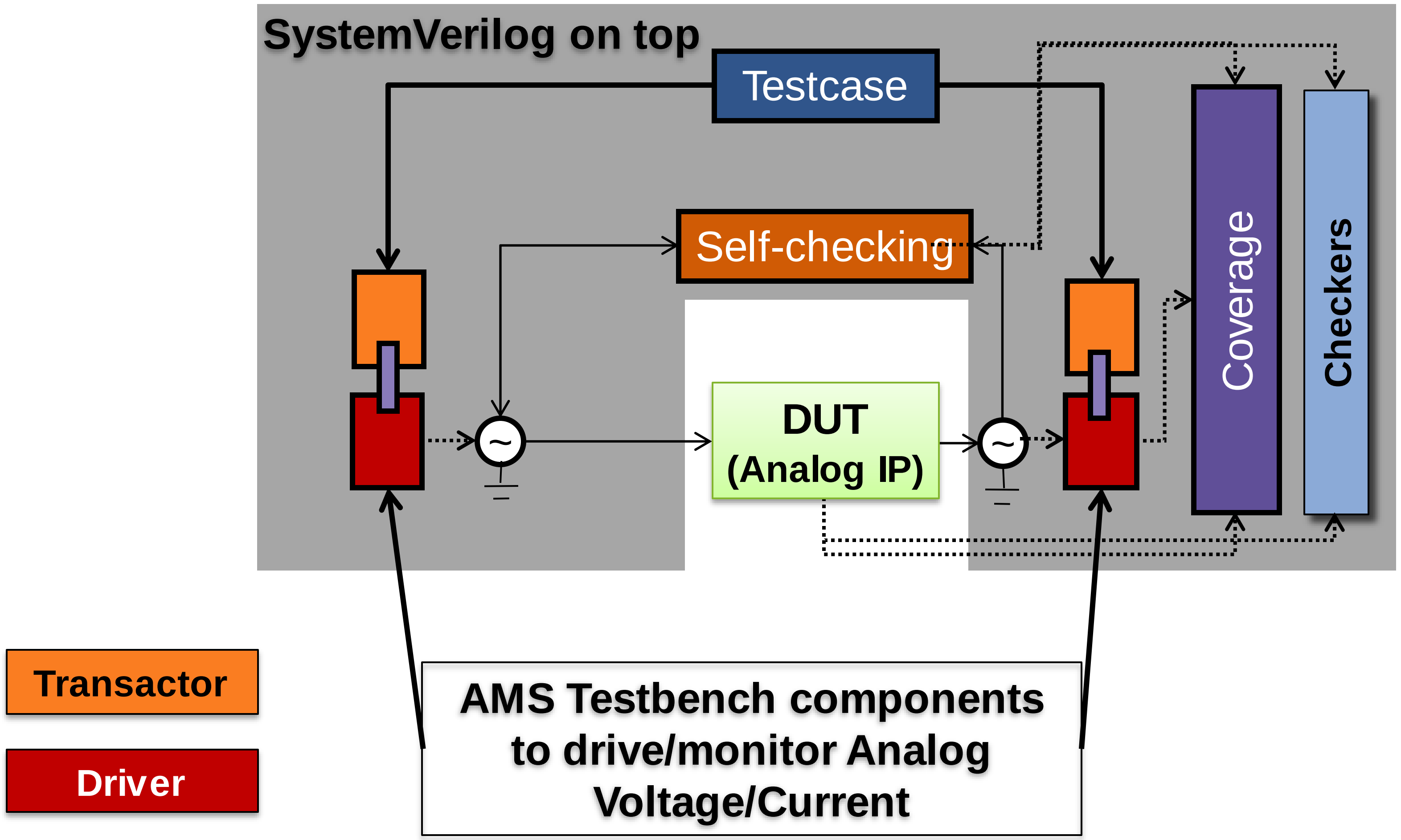


AMS Testbench Architecture



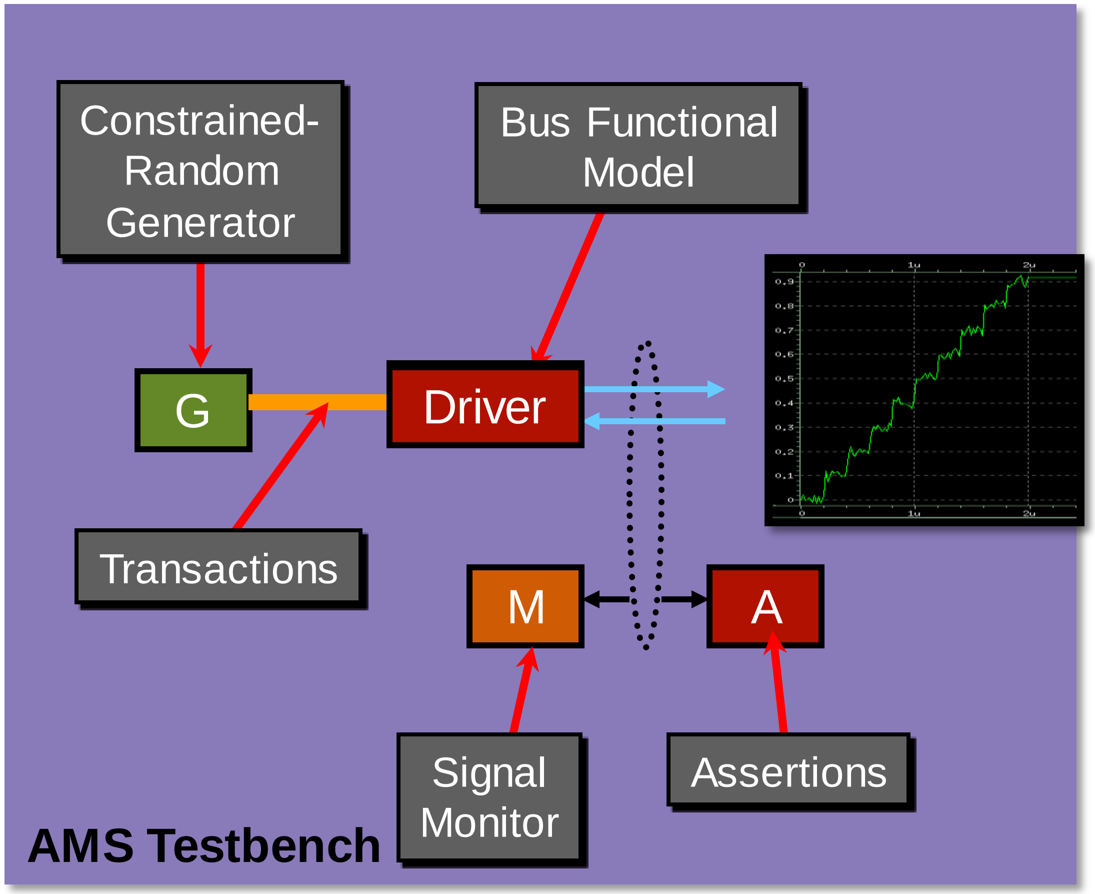
Analog IP Early Verification

Possible Usage Before SoC Integration



AMS Testbench components to drive/monitor Analog Voltage/Current

AMS Testbench Components



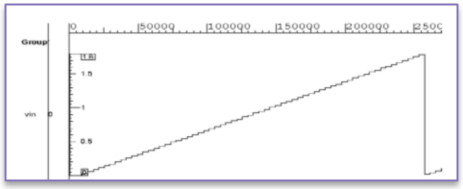
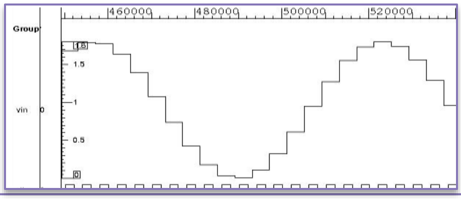
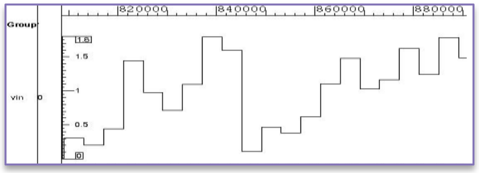
AMS Testbench Generators

- Sine voltage generator
- Sawtooth voltage generator
- Square voltage generator
- Fully customizable

AMS Testbench Checkers

- Threshold
- Stability
- Window
- Slew Rate
- Frequency

AMS Testbench Generators

Checkers		
class sv_ams_generic_src	Provides base class infrastructure for building voltage generators	
sv_ams_sawtooth_voltage_gen	Base class aimed at generating sawtooth waveforms with minimum/maximum voltage and frequency	
sv_ams_sine_voltage_gen	Base class aimed at generating sine waveforms with minimum/maximum voltage and frequency	
sv_ams_rand_voltage_gen	Base class aimed at generating random waveforms with minimum/maximum random voltage sweep	

AMS Testbench Generators

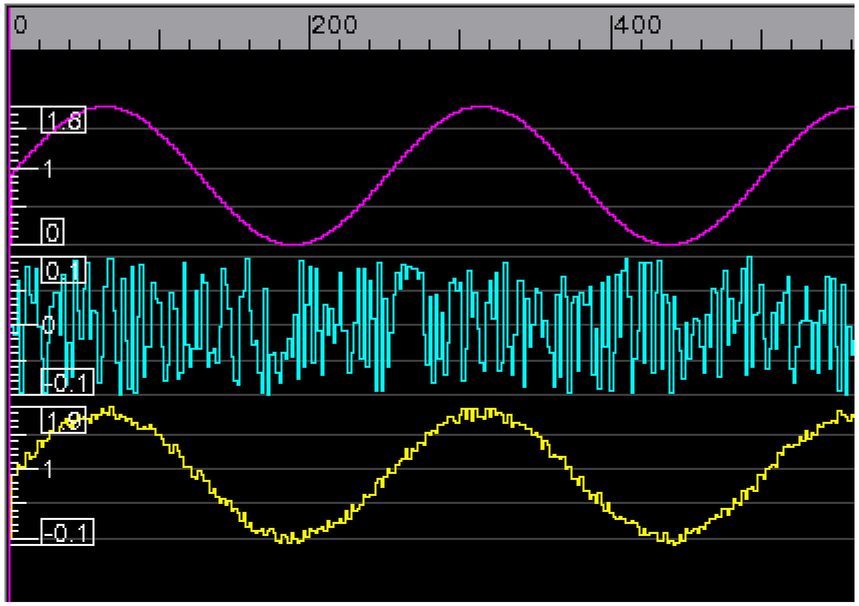
Sine voltage generator
- Vmax=1.0V,
- Vmin=-1.0V
- F=1.0MHz

Construct sine wave generator
Default is auto-run throughout *run_phase()*

```
...

class my_env extends uvm_component;
   ...
  sv_ams_sine_voltage_gen#(-1.0, +1.0, 1.0E6) sGen_IN;

 function void build_phase(uvm_phase phase);
   super.build_phase(phase);
   uvm_resource_db#(virtual ams_src_if)::
               set("*", "uvm_ams_src_if", aif, this);
   sGen_IN  = sv_ams_sine_voltage_gen#
   (-1.0, +1.0, 1.0E6)::type_id::create("sine", this);

 endfunction
```

Example 1 - Noise Simulation

Noise Injection into Mixed-signal Simulation



Sine Wave
(0-1.8V)

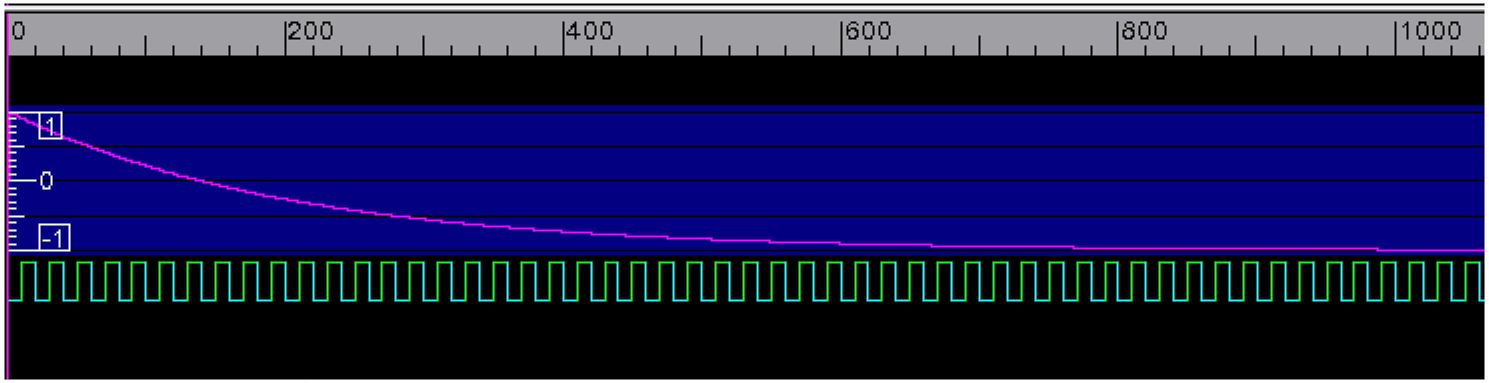
Noise
(-100 to 100mV)

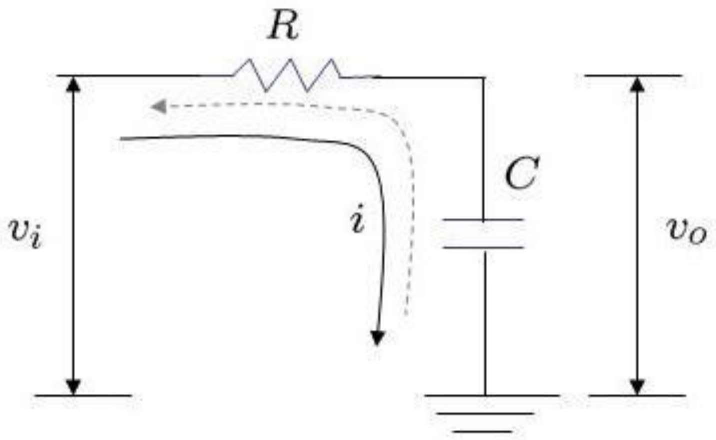
Sine plus noise

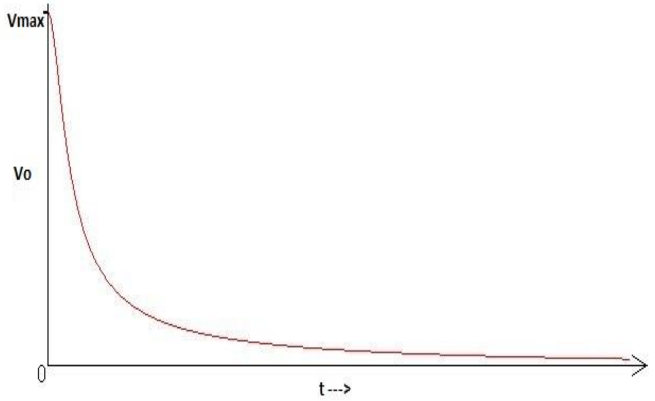
Random Noise Injected Using Sine Source

Example 2 - RC Voltage Generator

Can Be Used Within Testbench to Drive Analog Nodes





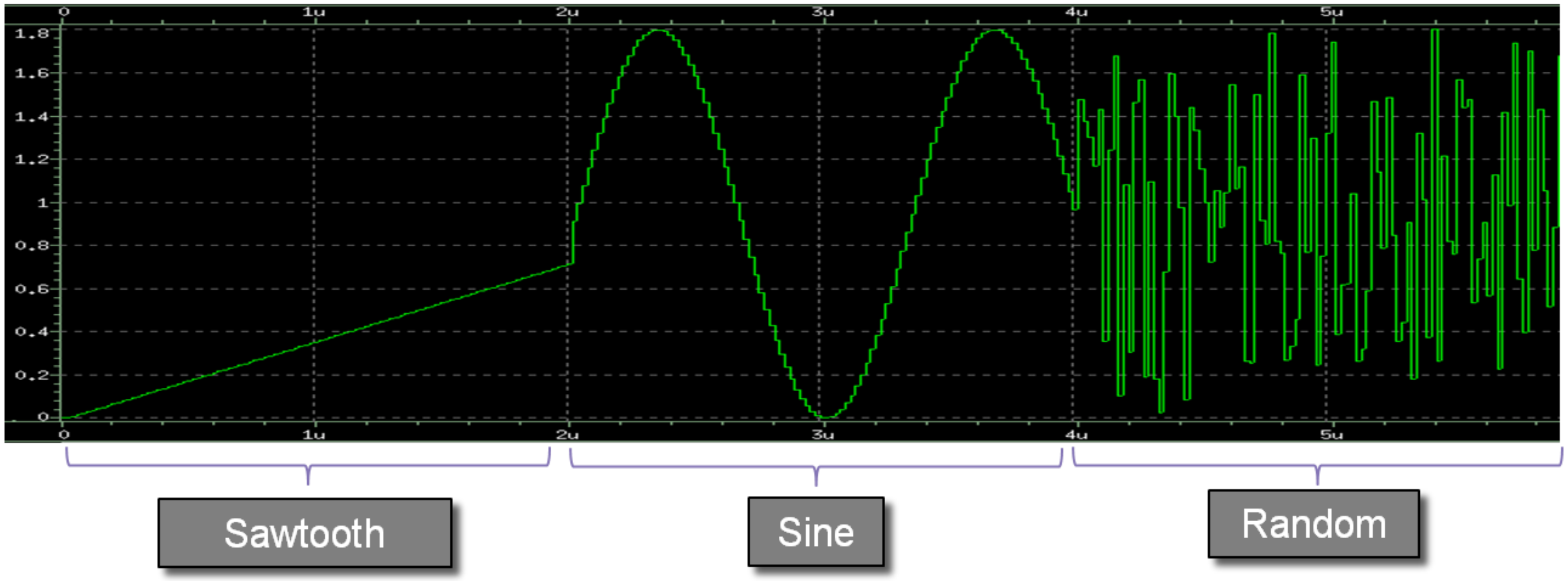


This custom source generator:

- ***Vmin=-1V, VMax=1V, Freq=1MHz, RC=200ns***

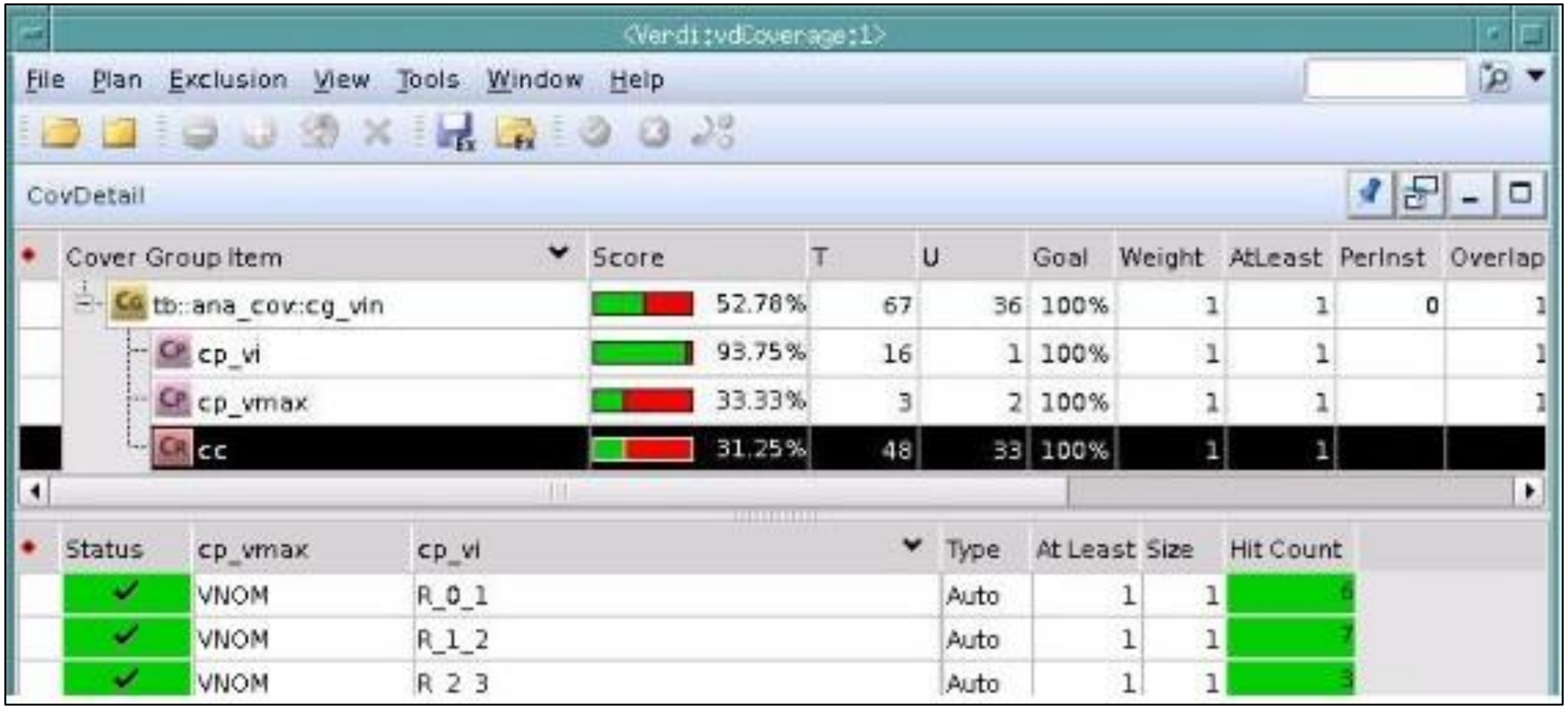
Example 3: Custom Voltage Generator

Different Source Types Can Be Combined



AMS Functional Coverage

Analog and Digital Coverage in Same Verification Plan



Cover Group Item	Score	T	U	Goal	Weight	AtLeast	PerInst	Overlap
tb::ana_cov::cg_vin	52.78%	67	36	100%	1	1	0	1
cp_vi	93.75%	16	1	100%	1	1		1
cp_vmax	33.33%	3	2	100%	1	1		1
cc	31.25%	48	33	100%	1	1		

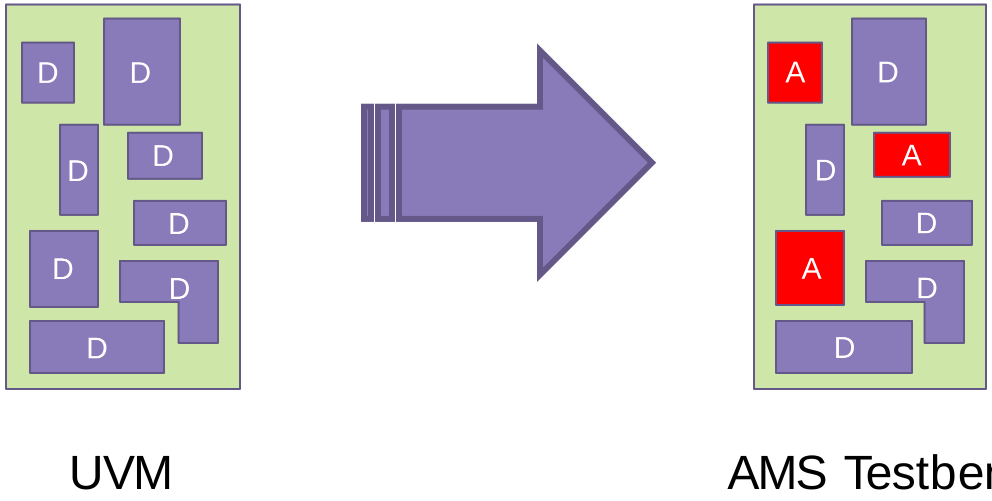
Status	cp_vmax	cp_vi	Type	At Least	Size	Hit Count
✓	VNOM	R_0_1	Auto	1	1	6
✓	VNOM	R_1_2	Auto	1	1	7
✓	VNOM	R_2_3	Auto	1	1	3

By clearly referencing both analog and digital coverage groups, both domains can be verified together

AMS Testbench - Summary

Functional verification methodology for mixed-signal SoC

- Extends SystemVerilog Testbench Environment to mixed-signal domain
- Provide mixed-signal coverage in SystemVerilog testbench



VCS AMS in STMicroelectronics

Pierluigi Daglio

AMS Design Verification Flows Manager

SPA - TR&D Smart Power
Design Enablement

Mauro Scandiuzzo

Field Application Engineer

AMS & IPG Marketing & Applications

life.augmented

DVCon Europe - Munich
14th October 2014

Outline

- STMicroelectronics Product Segments
- STMicroelectronics Analog/Mixed-Signal Scenario
  - Verification flow and purpose
  - The application
  - Design and process
- Usage of Unique VCS AMS Features for Superior Productivity
  - Assertions
  - Save and Restore
- Conclusions

Product Segments

life.augmented

- Sense & Power and Automotive Products (SP&A)
  - Analog, MEMS & Sensors (AMS)
  - Automotive Product Group (APG)
  - Industrial & Power Discrete Group (IPD)
- Embedded Processing Solutions (EPS)
  - Digital Convergence Group (DCG)
  - Imaging, BiCMOS, ASIC & Silicon Photonics (IBP)
  - Microcontroller, Memory & Secure MCU (MMS)

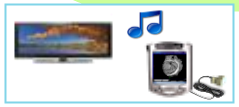
Smart Power Product Segments

Sense & Power and Automotive Products (SP&A)

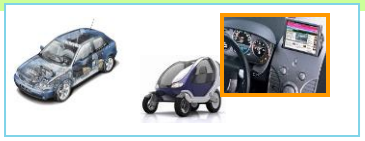
- Analog, MEMS & Sensors (AMS)
- Automotive Product Group (APG)
- Industrial & Power Discrete Group (IPD)

Embedded Processing Solutions (EPS)

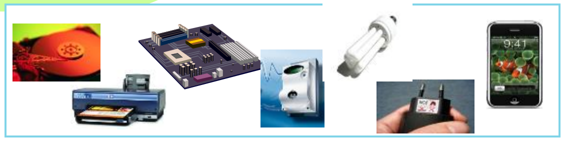
- Digital Convergence Group (DCG)
- Imaging, BiCMOS, ASIC & Silicon Photonics (IBP)
- Microcontroller, Memory & Secure MCU (MMS)



- Audio amplifier
- Ultrasound echography
- High performance analog



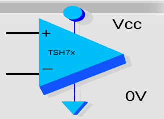
- Airbag
- Gasoline Direct Injection
- ESP/ABS
- Car Radio
- Precision battery monitoring



- HDD Pow er Combo
- Printer head & motor drivers
- Motherboard DCDC converter
- LED bulb driver
- Pow er Line driver
- Mobile pow er management
- Mobile display drivers

Smart Power Application Fields

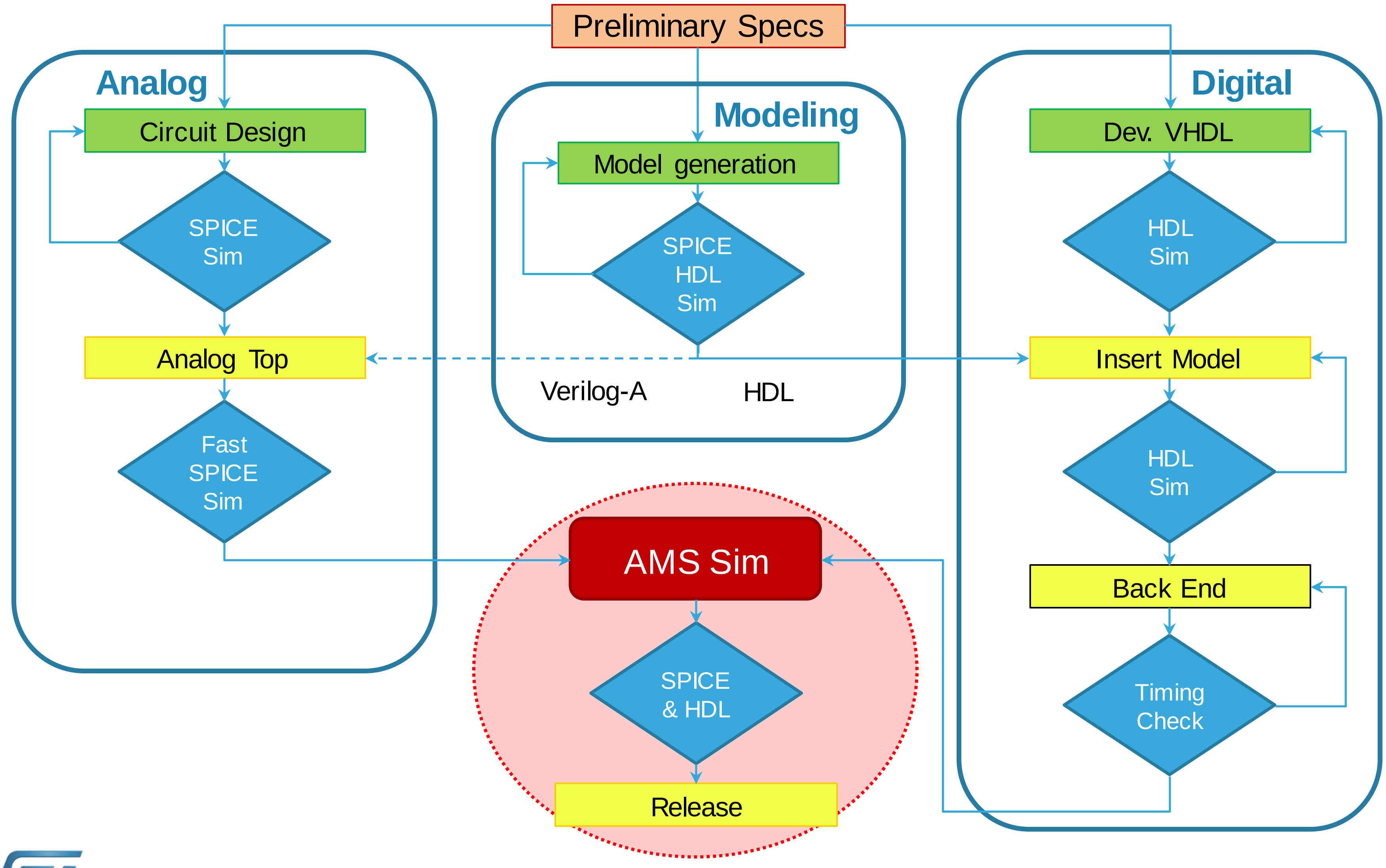
by Technology Platform Segment

BCD Segment	Technology Platforms	Application Field
Off Line BCD *600V – 5kV*	• BCD6s OFFLINE • BCD6s HV Transformer 0.32µm	Lighting Motors Electrical Car
SOI BCD *190V – 300V*	• SOI-BCD6s 0.32µm • SOI-BCD8s 0.16µm	Full digital amplifier Ultrasound Ecography AMOLED Power Supply
Advanced BCD *7V – 100V*	• BCD8A/As 0.18µm • BCD8sP 0.16µm • BCD8sAUTO 0.16µm • BCD9s 0.11µm • BCD9sL 0.11µm • BCD10 90nm • BCD11 65nm	HDD Audio amplifier Power Line modem Printers Power Supply Power Management
High Voltage CMOS *16V – 40V*	• HVG8A 0.18µm • HVCMOS8A 0.18µm	Electronic labels E-book Automotive Sensor IC Bio Medical  Advanced Analog

ST Analog/Mixed-Signal Scenario

- Simulation and verification of large IPs and macro-cells
  - Transistor-level simulation
  - Static and dynamic electrical rule checks (ERC)
  - Safe operating area (SOA)
  - Analog behavioral languages (Verilog-A)
- Simulation and verification of A/M-S systems
  - System-level analog/digital co-simulation
  - Digital test benches
  - Fast and reliable simulation (speed & accuracy)
  - HDL languages and analog behavioral languages

AMS Verification Flow

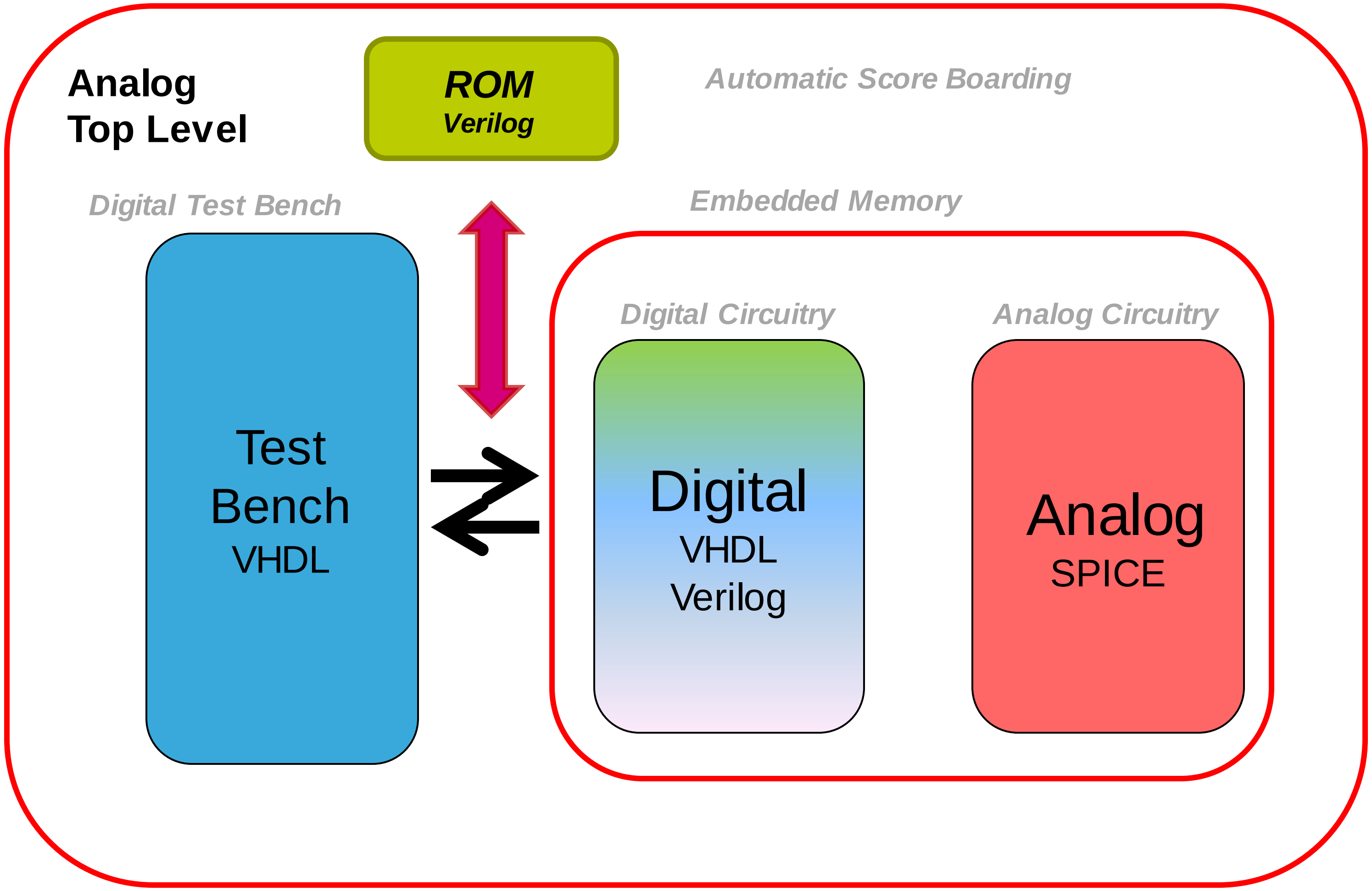


AMS Verification Purpose

Purpose: Verify complex designs at top-level both with digital and analog parts

- It is possible to verify complex designs mixing among …
  - **netlist configurations**
    - *digital*: VHDL or Verilog post synthesis
    - *analog*: pre-layout or post-layout (trade-off simulation time)
  - **operation conditions**
    - *digital*: for Verilog (min & max delay)
    - *analog*: TYP, SSA, FFA, ….
  - **modes**
    - *Usermode*: boot, read, erase, prog with algorithm, …
      (user set of operations)
    - *Testmode*: DMA MTX_CELL, DMA REF_CELL, Erase Reference matrix
      (IP validation)

AMS Verification - The Application



AMS Verification Design and Process

Design Complexity

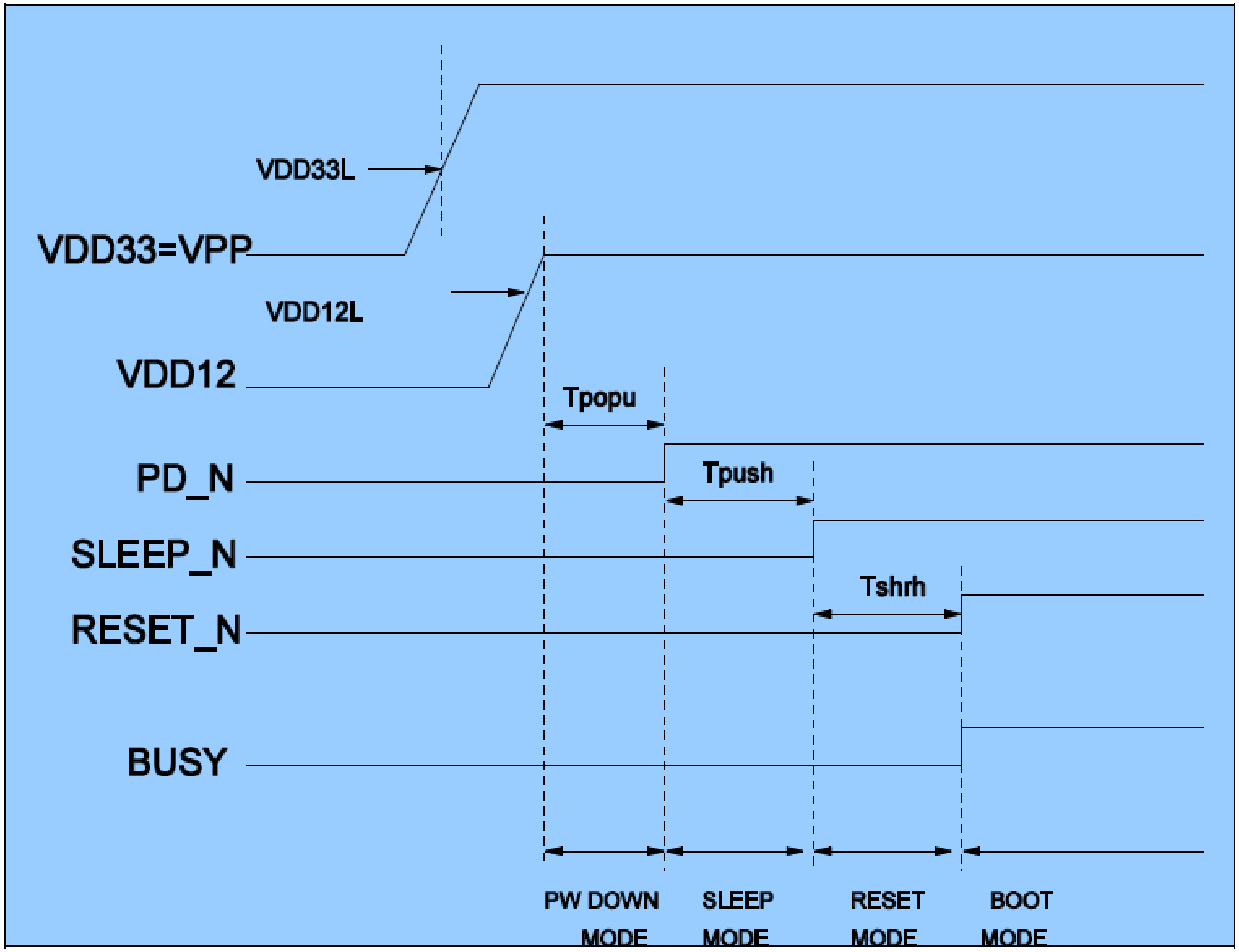
- Memory Size 136KB Single Module
  - Organized into 3*32KB-sector, 5*8KB and 1*TF sector (8KB)
- Double Voltage Supply
  - Analog supply: 1.62V-3.6V
  - Digital supply: 1.08V-1.32V
- x32-bit Reading (internally, 32b + 1b Redundancy) and Writing (internally, 2 bits by 2)
- Embedded Program/Erase Code

Technology

- Low Power: Double Poly, Triple-Well
- Differential Oxide: GO1-LV, GO2-MV, HV
- Low Power Consumption
- MIM Capacitors
- High Resistivity Poly1 Resistors Available

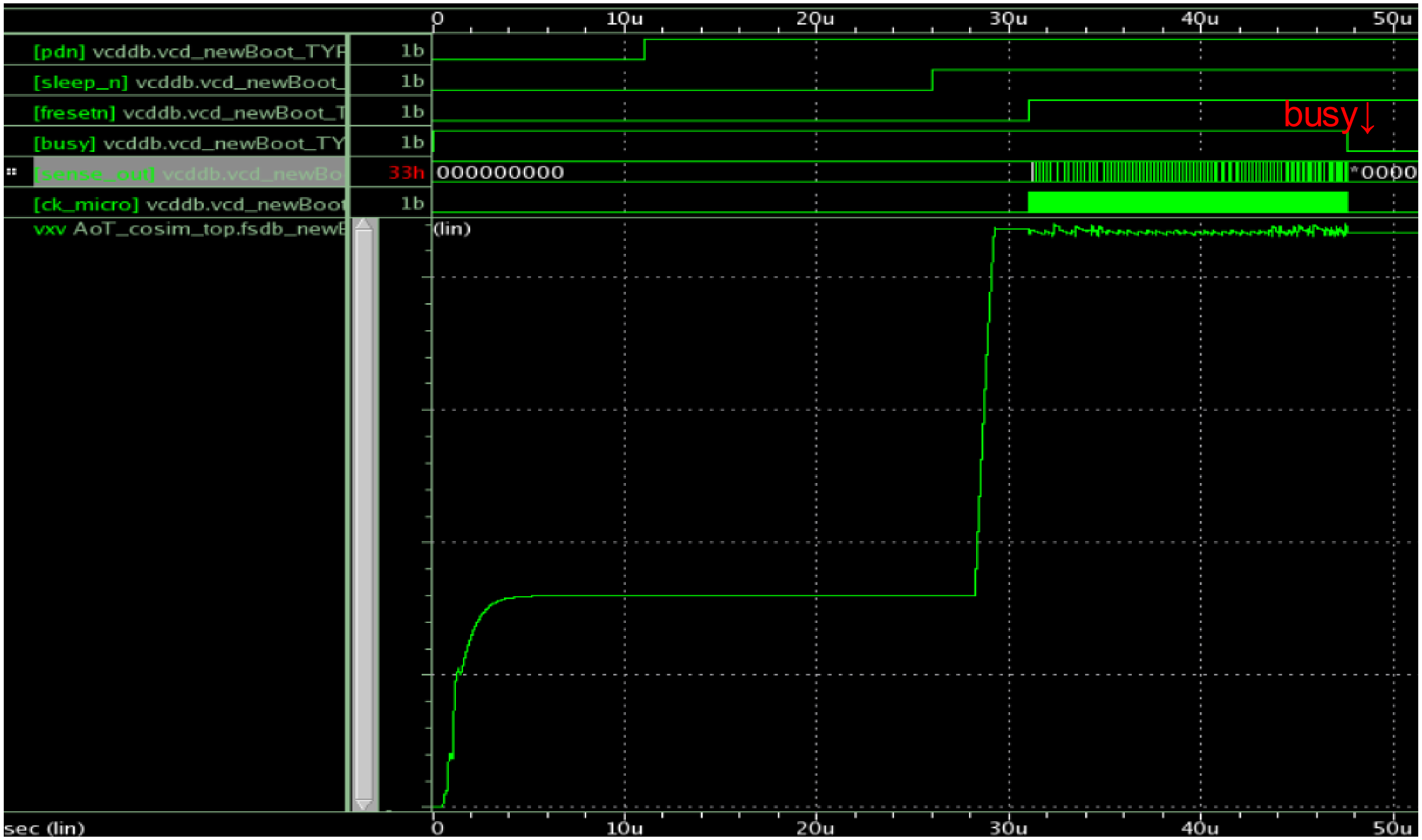
life.augmented

Power-up Sequence



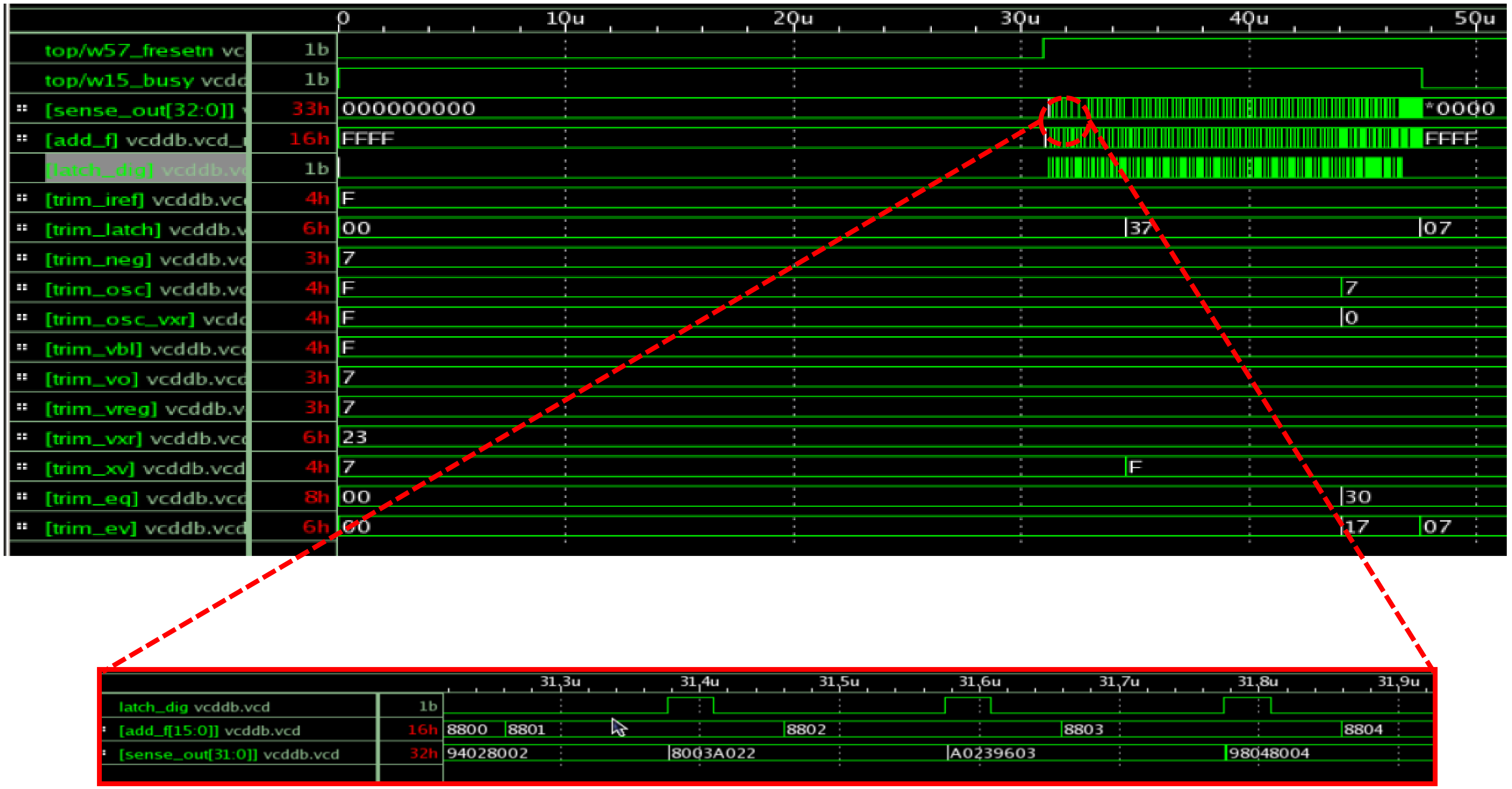
life.augmented

Waveforms - Boot Operations



0
10u
20u
30u
40u
50u
[pdn] vcddb.vcd_newBoot_TYP
1b
[sleep_n] vcddb.vcd_newBoot_
1b
[fresetn] vcddb.vcd_newBoot_T
1b
[busy] vcddb.vcd_newBoot_TY
1b
[sense_out] vcddb.vcd_newBo
33h
000000000
*0000
[ck_micro] vcddb.vcd_newBoot
1b
vxv AoT_cosim_top.fsdb_newB
(lin)
busy↓
sec (lin)

Boot Trimming - Sense Out Details



```
                                  0          10u         20u         30u         40u         50u
top/w57_fresetn vc          1b
top/w15_busy vcdd           1b
[sense_out[32:0]]          33h  000000000                                                   *0000
[add_f] vcddb.vcd_         16h  FFFF                                                        FFFF
[latch_dig] vcddb.vc        1b
[trim_iref] vcddb.vc        4h  F
[trim_latch] vcddb.v        6h  00                                  37                      07
[trim_neg] vcddb.vc         3h  7
[trim_osc] vcddb.vc         4h  F                                               7
[trim_osc_vxr] vcdd         4h  F                                               0
[trim_vbl] vcddb.vcc        4h  F
[trim_vo] vcddb.vcc         3h  7
[trim_vreg] vcddb.v         3h  7
[trim_vxr] vcddb.vcc        6h  23
[trim_xv] vcddb.vcd         4h  7                                   F
[trim_eq] vcddb.vcc         8h  00                                              30
[trim_ev] vcddb.vcd         6h  00                                              17          07
```

```
                                 31,3u      31,4u      31,5u      31,6u      31,7u      31,8u      31,9u
latch_dig vcddb.vcd           1b
[add_f[15:0]] vcddb.vcd      16h  8800  8801              8802              8803              8804
[sense_out[31:0]] vcddb.vcd  32h  94028002        8003A022          A0239603          98048004
```

VCS AMS Automatic Checks w/Assertions

Digital input

Analog output

Trigger

```
checker UCK (.addr(add_f[15:0]), .data(sense_out[31:0]), .clk(latch_dig));

module checker (
addr ,
data ,
clk
);
```

Checker

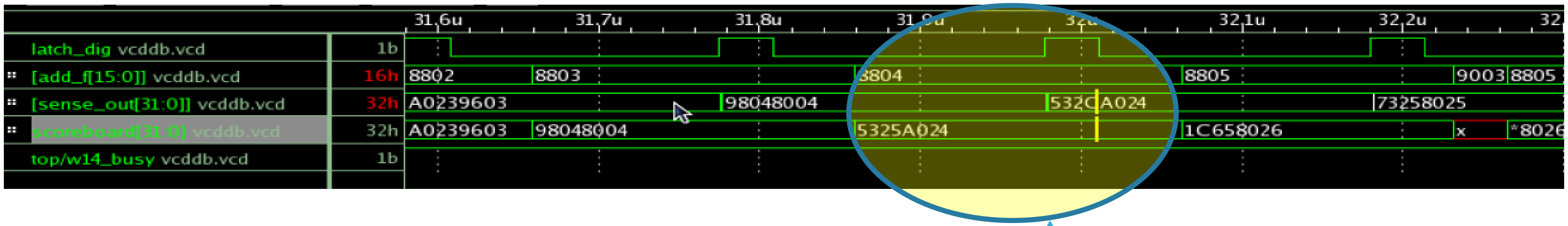
```
...
assign scoreboard = ce ? mem[addr] : 32'b0;

initial begin
   ...
   file= $fopen(report.dat);
   $readmemh("mem.dat", mem);
   ...
end
```

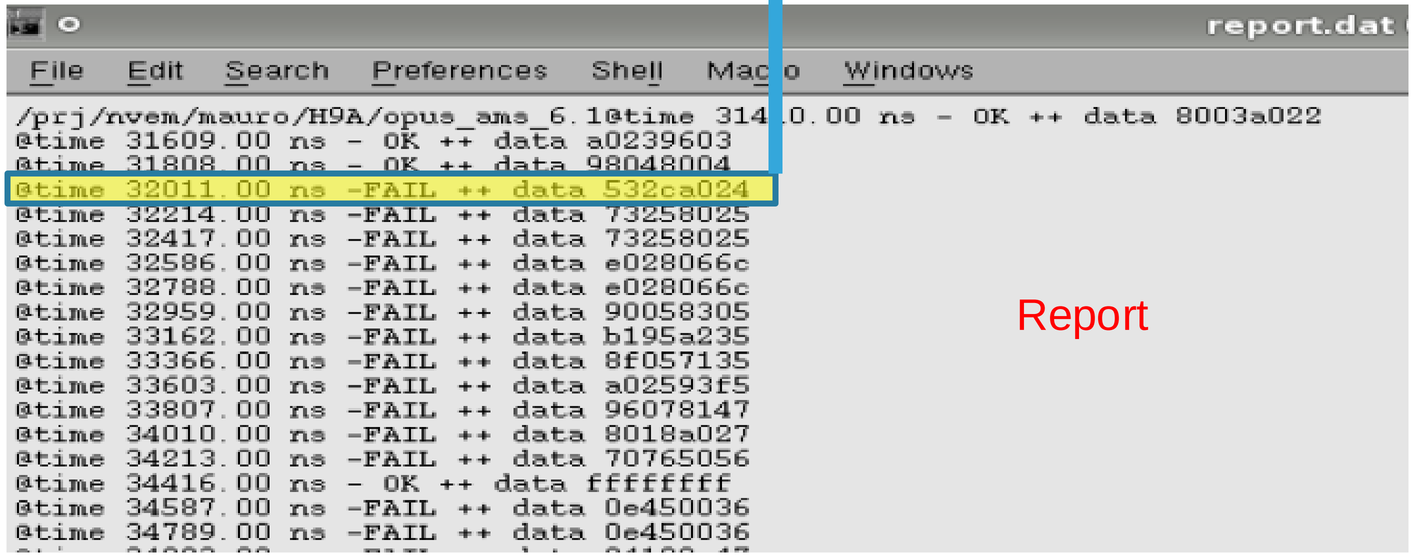
Scoreboard

```
property BUS_MATCH (clk, busA, busB , enable);
     @(negedge clk)
                  enable |=> (busA == busB );
endproperty

myCheck : assert property (BUS_MATCH (clk, data, scoreboard, ce))
              begin
                     $fdisplay(file,"@time %t - OK ++ data %h ", $time, data);
              end
              else    begin
                     $fdisplay(file,"@time %t -FAIL ++ data %h", $time, data);
                     $warning("@time %t -FAIL ++ data %h", $time, data);
              end
```

Assertion

Assertions Simulation Results


31,6u
31,7u
31,8u
31,9u
32u
32,1u
32,2u
latch_dig vcddb.vcd 1b
[add_f[15:0]] vcddb.vcd 16h 8802 8803 8804 8805 9003 8805
[sense_out[31:0]] vcddb.vcd 32h A0239603 98048004 532CA024 73258025
scoreboard[31:0] vcddb.vcd 32h A0239603 98048004 5325A024 1C658026 x *8026
top/w14_busy vcddb.vcd 1b

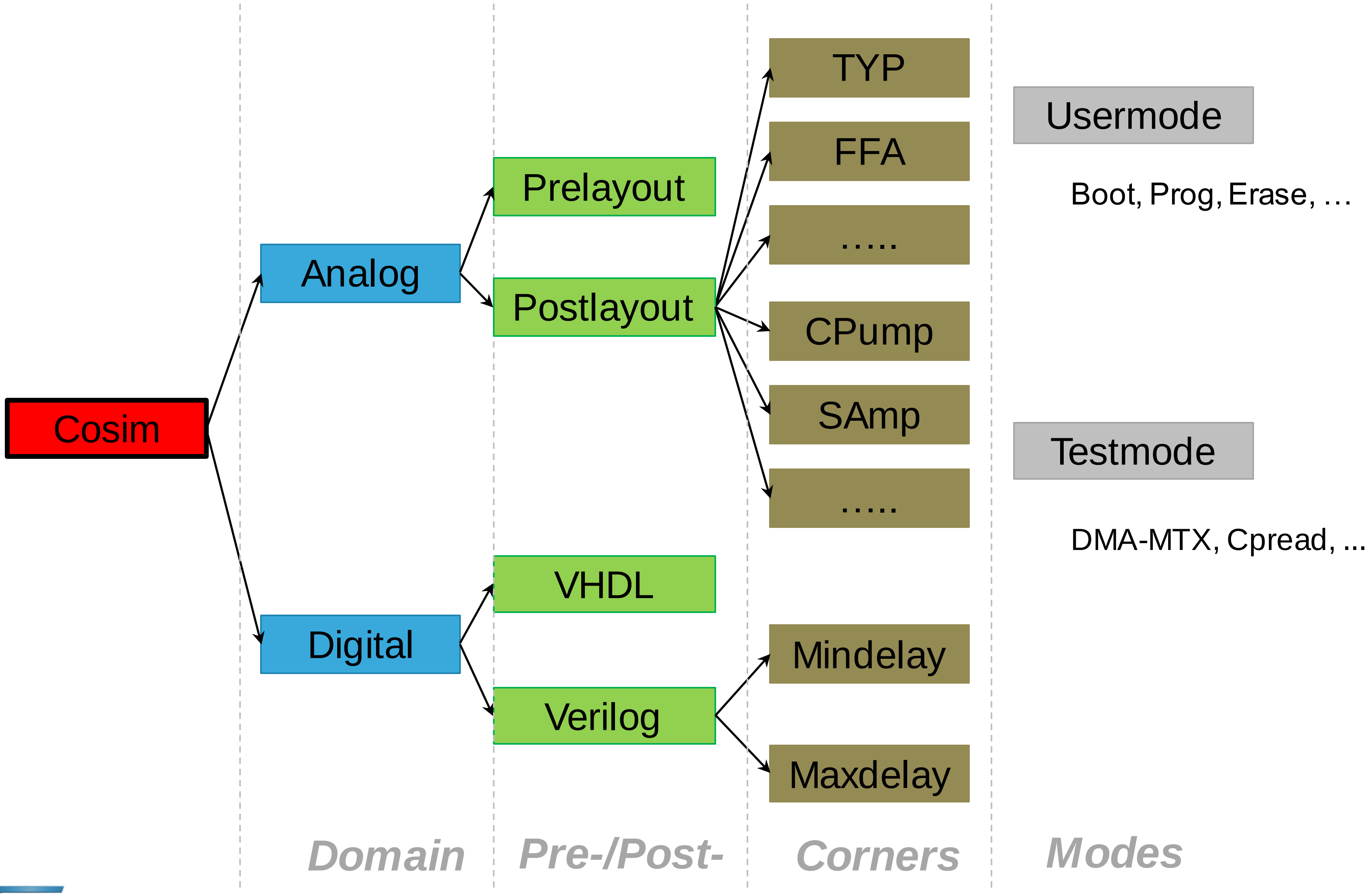


```
/prj/nvem/mauro/H9A/opus_ams_6.1@time 31410.00 ns - OK ++ data 8003a022
@time 31609.00 ns - OK ++ data a0239603
@time 31808.00 ns - OK ++ data 98048004
@time 32011.00 ns -FAIL ++ data 532ca024
@time 32214.00 ns -FAIL ++ data 73258025
@time 32417.00 ns -FAIL ++ data 73258025
@time 32586.00 ns -FAIL ++ data e028066c
@time 32788.00 ns -FAIL ++ data e028066c
@time 32959.00 ns -FAIL ++ data 90058305
@time 33162.00 ns -FAIL ++ data b195a235
@time 33366.00 ns -FAIL ++ data 8f057135
@time 33603.00 ns -FAIL ++ data a02593f5
@time 33807.00 ns -FAIL ++ data 96078147
@time 34010.00 ns -FAIL ++ data 8018a027
@time 34213.00 ns -FAIL ++ data 70765056
@time 34416.00 ns - OK ++ data ffffffff
@time 34587.00 ns -FAIL ++ data 0e450036
@time 34789.00 ns -FAIL ++ data 0e450036
```

Report

life.augmented

VCS AMS Save and Restore - Setup



VCS AMS Save and Restore Capability

- A "memory image" of the simulation can be saved and restored at a later time
- At restore time, the netlist and the simulation set-up cannot be changed
- Typically used for running many functional simulations on the same design after the power-up

Save

```
% simv -ucli
ucli% run 100
ucli% save sim_state
ucli% quit
```

Restore

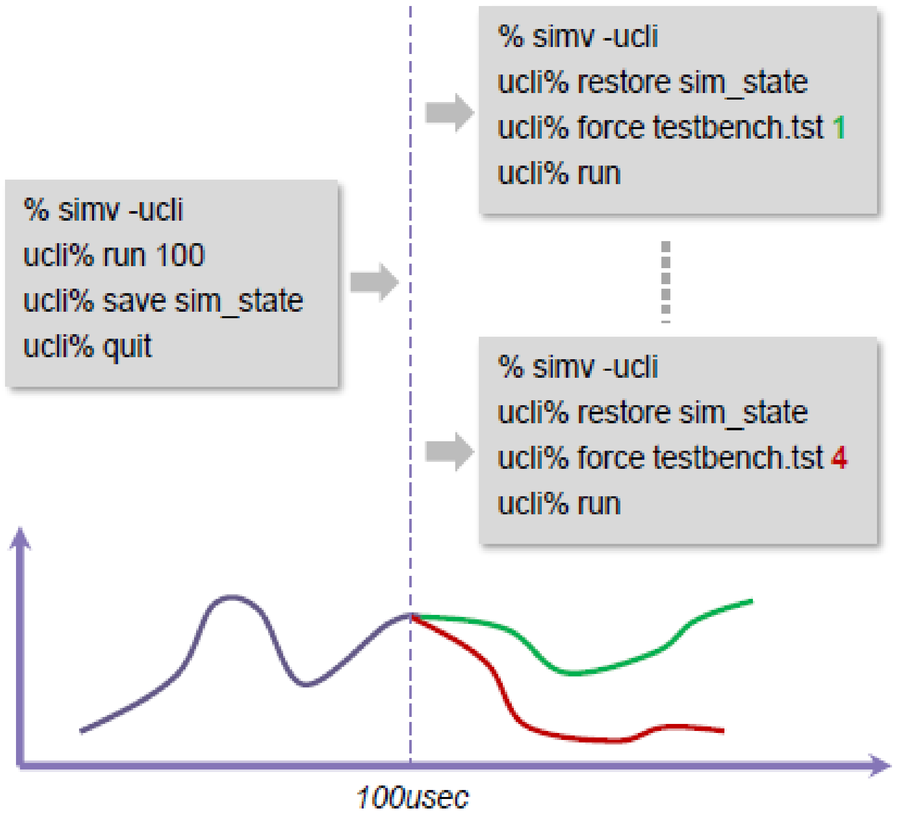
```
% simv -ucli
ucli% restore sim_state
ucli% run
```

life.augmented

VCS AMS Save and Restore Performance

- Multiple runs can be executed by forcing a test selector variable
  - A run is chosen by forcing the test selector via UCLI

```
module testbench;
int tst;
...
case (tst)
        1: test1( );   // Run Test No 1
        2: test2( );   // Run Test No 2
        3: test3( );   // Run Test No 3
        4: test4( );   // Run Test No 4
        5: test5( );   // Run Test No 5
default: test0 ( );
endcase
...
end
```

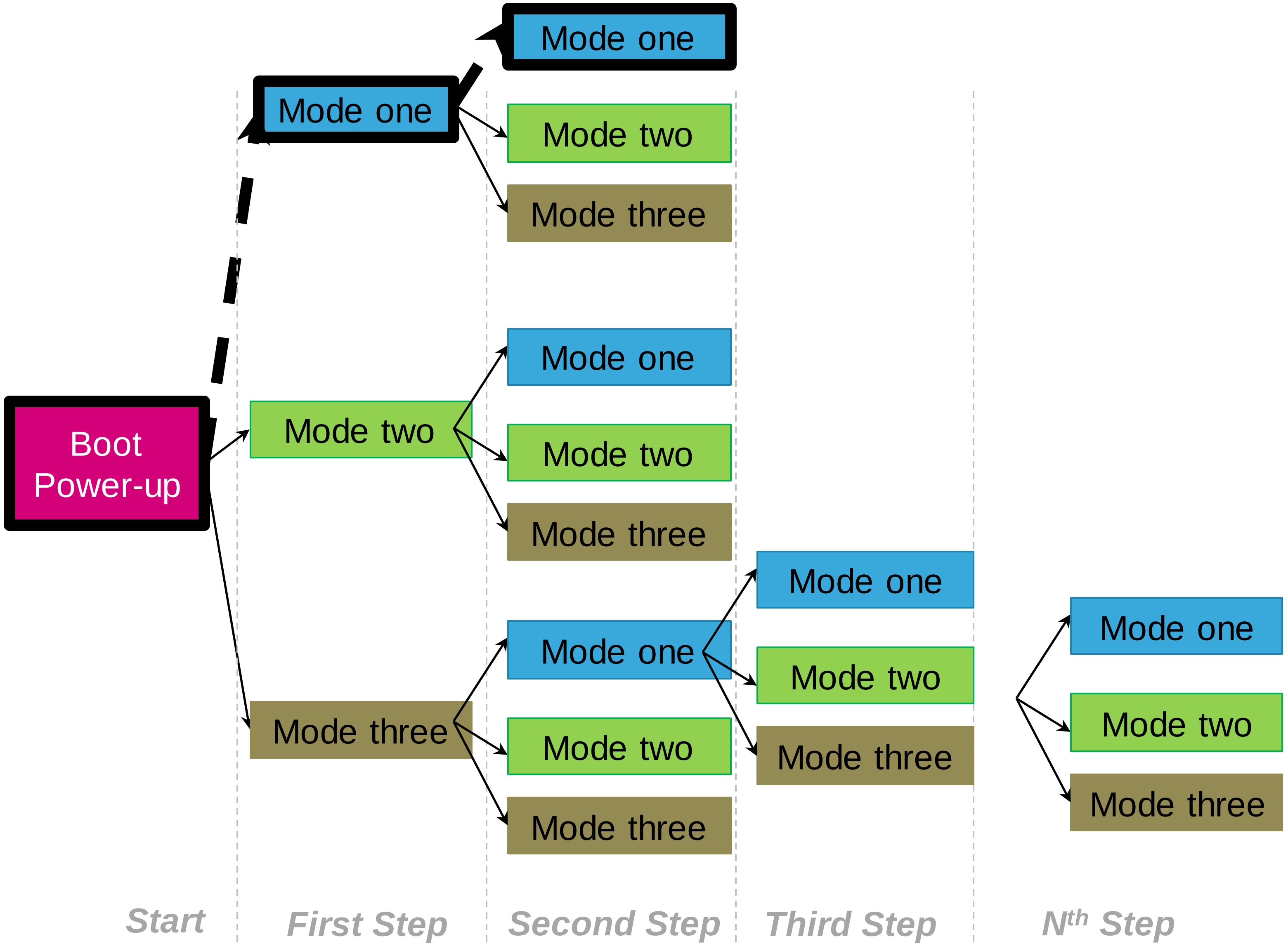


```
% simv -ucli
ucli% run 100
ucli% save sim_state
ucli% quit
```

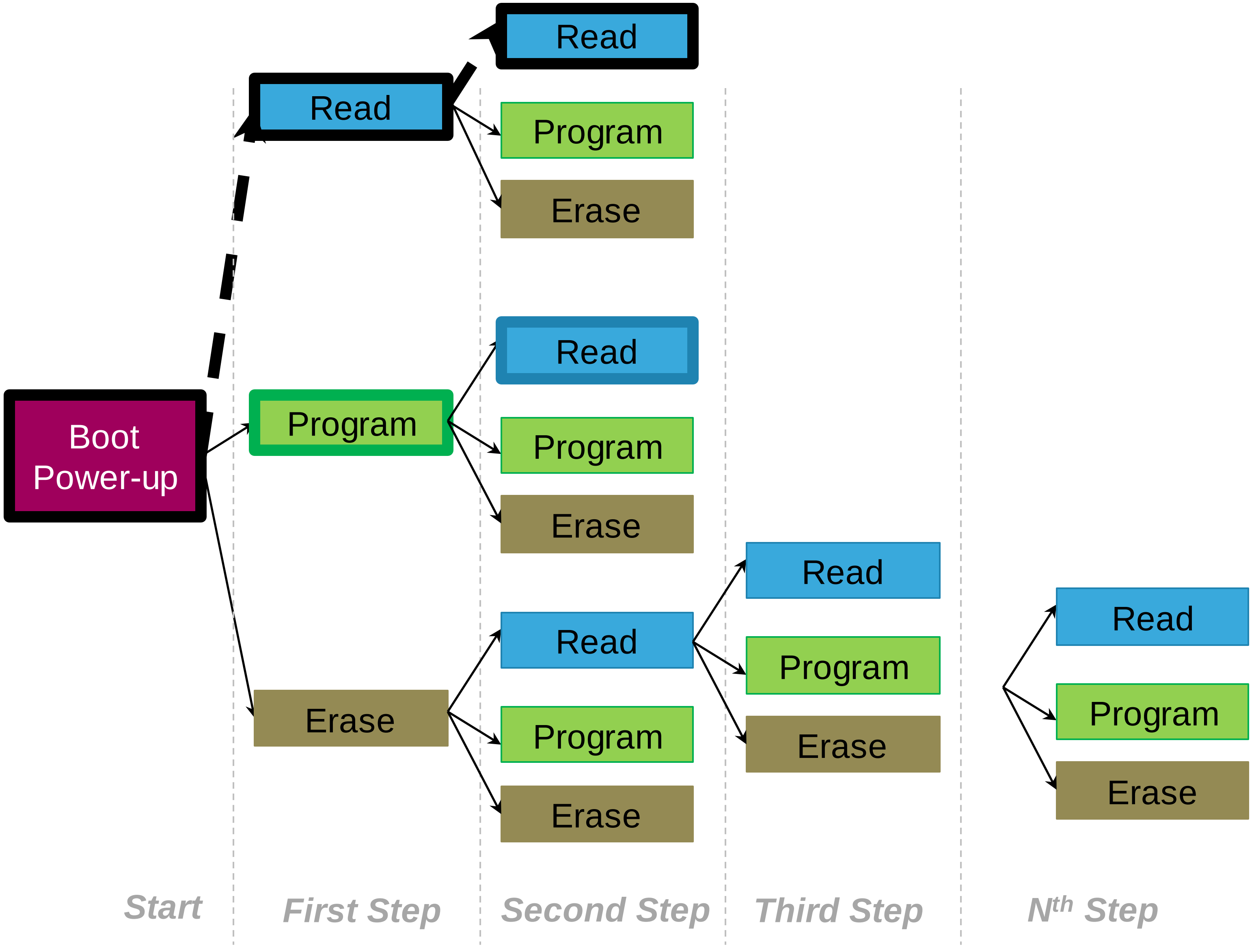
```
% simv -ucli
ucli% restore sim_state
ucli% force testbench.tst 1
ucli% run
```

```
% simv -ucli
ucli% restore sim_state
ucli% force testbench.tst 4
ucli% run
```

Multi-Mode Approach w/Save and Restore

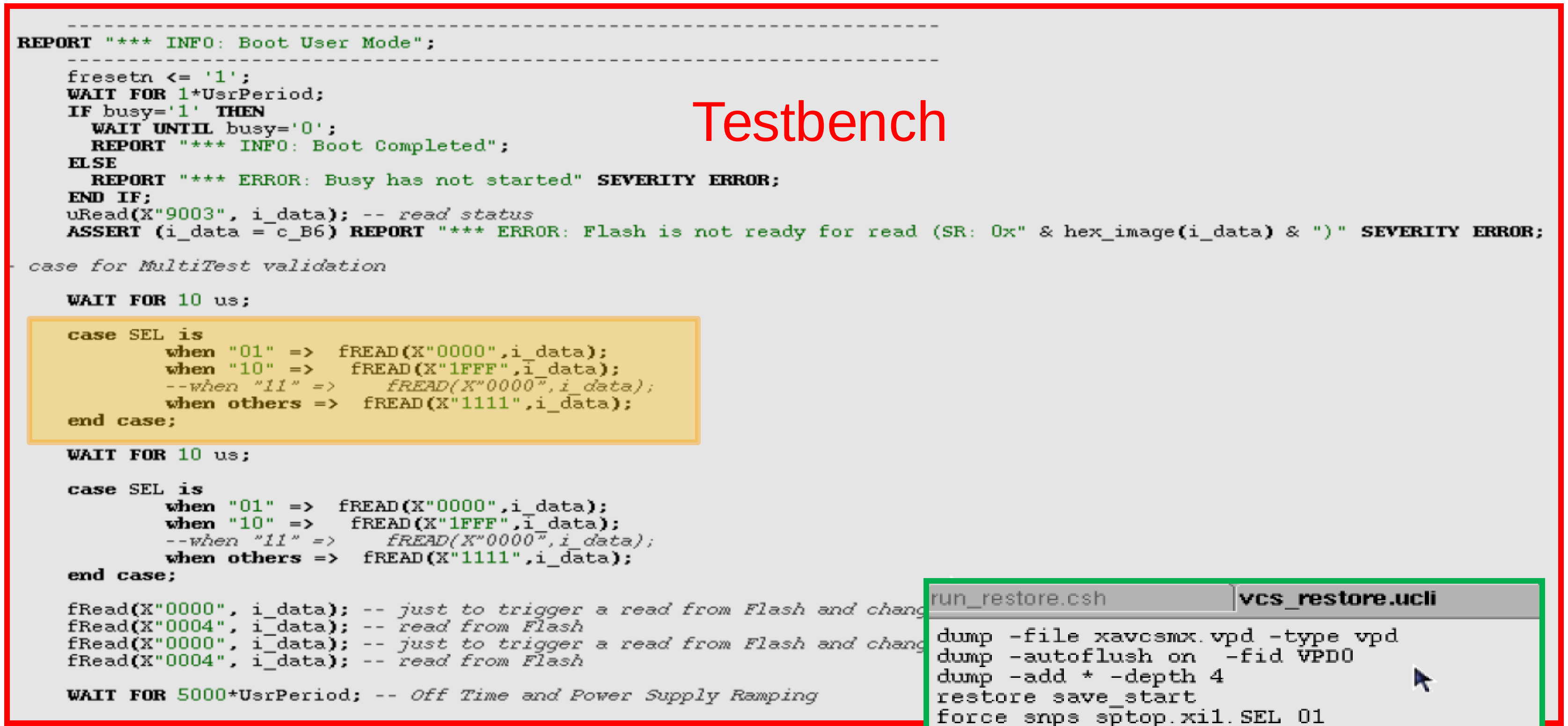


Real Application Save & Restore Capability



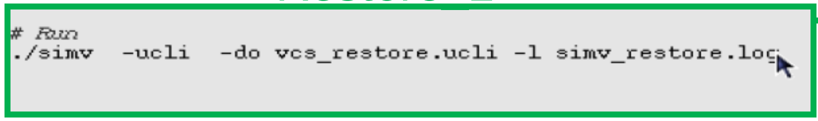
Save and Restore - Boot & Multiple Read

Testbench

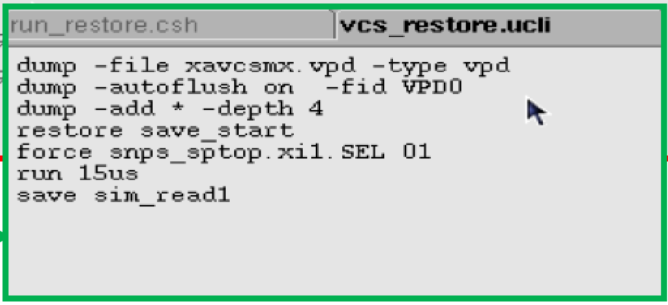


```
    ----------------------------------------------------------------------
REPORT "*** INFO: Boot User Mode";
    ----------------------------------------------------------------------
    fresetn <= '1';
    WAIT FOR 1*UsrPeriod;
    IF busy='1' THEN
      WAIT UNTIL busy='0';
      REPORT "*** INFO: Boot Completed";
    ELSE
      REPORT "*** ERROR: Busy has not started" SEVERITY ERROR;
    END IF;
    uRead(X"9003", i_data); -- read status
    ASSERT (i_data = c_B6) REPORT "*** ERROR: Flash is not ready for read (SR: 0x" & hex_image(i_data) & ")" SEVERITY ERROR;

- case for MultiTest validation

    WAIT FOR 10 us;

    case SEL is
            when "01" =>  fREAD(X"0000",i_data);
            when "10" =>   fREAD(X"1FFF",i_data);
            --when "11" =>     fREAD(X"0000",i_data);
            when others =>  fREAD(X"1111",i_data);
    end case;
    WAIT FOR 10 us;

    case SEL is
            when "01" =>  fREAD(X"0000",i_data);
            when "10" =>   fREAD(X"1FFF",i_data);
            --when "11" =>     fREAD(X"0000",i_data);
            when others =>  fREAD(X"1111",i_data);
    end case;

    fRead(X"0000", i_data); -- just to trigger a read from Flash and chang
    fRead(X"0004", i_data); -- read from Flash
    fRead(X"0000", i_data); -- just to trigger a read from Flash and chang
    fRead(X"0004", i_data); -- read from Flash

    WAIT FOR 5000*UsrPeriod; -- Off Time and Power Supply Ramping
```

Restore_1

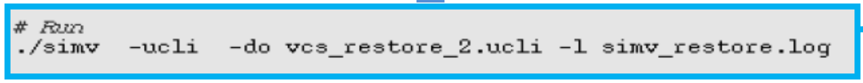


```
# Run
./simv  -ucli  -do vcs_restore.ucli -l simv_restore.log
```

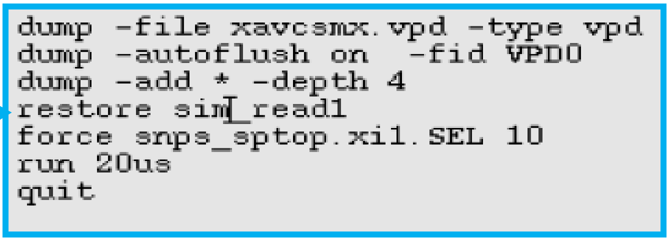


```
dump -file xavcsmx.vpd -type vpd
dump -autoflush on  -fid VPD0
dump -add * -depth 4
restore save_start
force snps_sptop.xi1.SEL 01
run 15us
save sim_read1
```

Restore_2



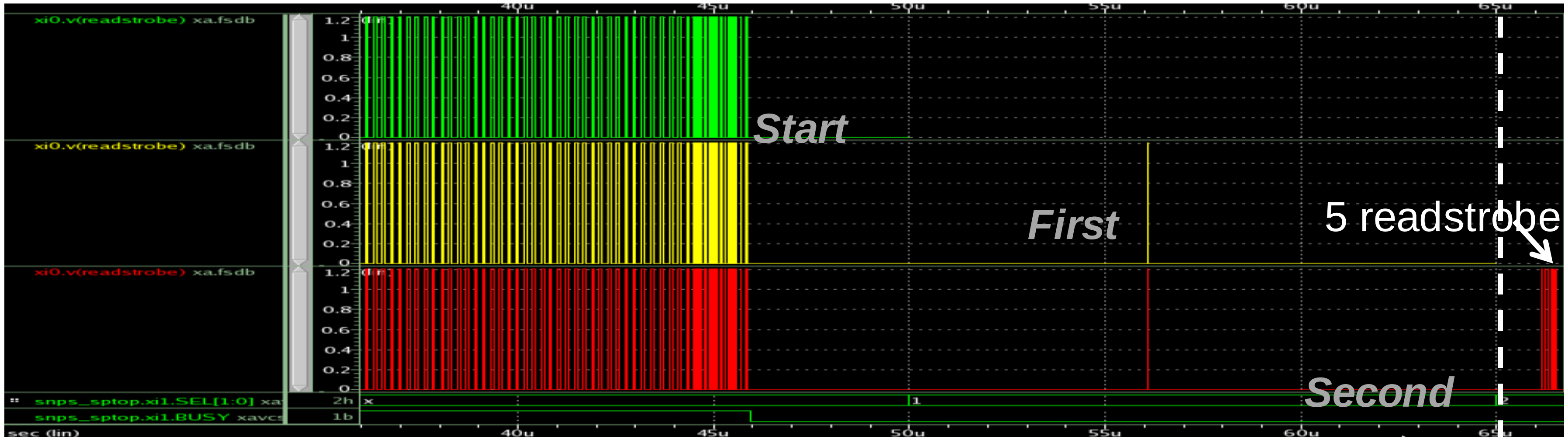
```
# Run
./simv  -ucli  -do vcs_restore_2.ucli -l simv_restore.log
```



```
dump -file xavcsmx.vpd -type vpd
dump -autoflush on  -fid VPD0
dump -add * -depth 4
restore sim_read1
force snps_sptop.xi1.SEL 10
run 20us
quit
```

life.augmented

Save and Restore - Simulation Results


xi0.v(readstrobe) xa.fsdb
xi0.v(readstrobe) xa.fsdb
xi0.v(readstrobe) xa.fsdb
snps_sptop.xi1.SEL[1:0] xa
snps_sptop.xi1.BUSY xavcs
sec (lin)
40u
45u
50u
55u
60u
65u
Start
First
Second
5 readstrobe

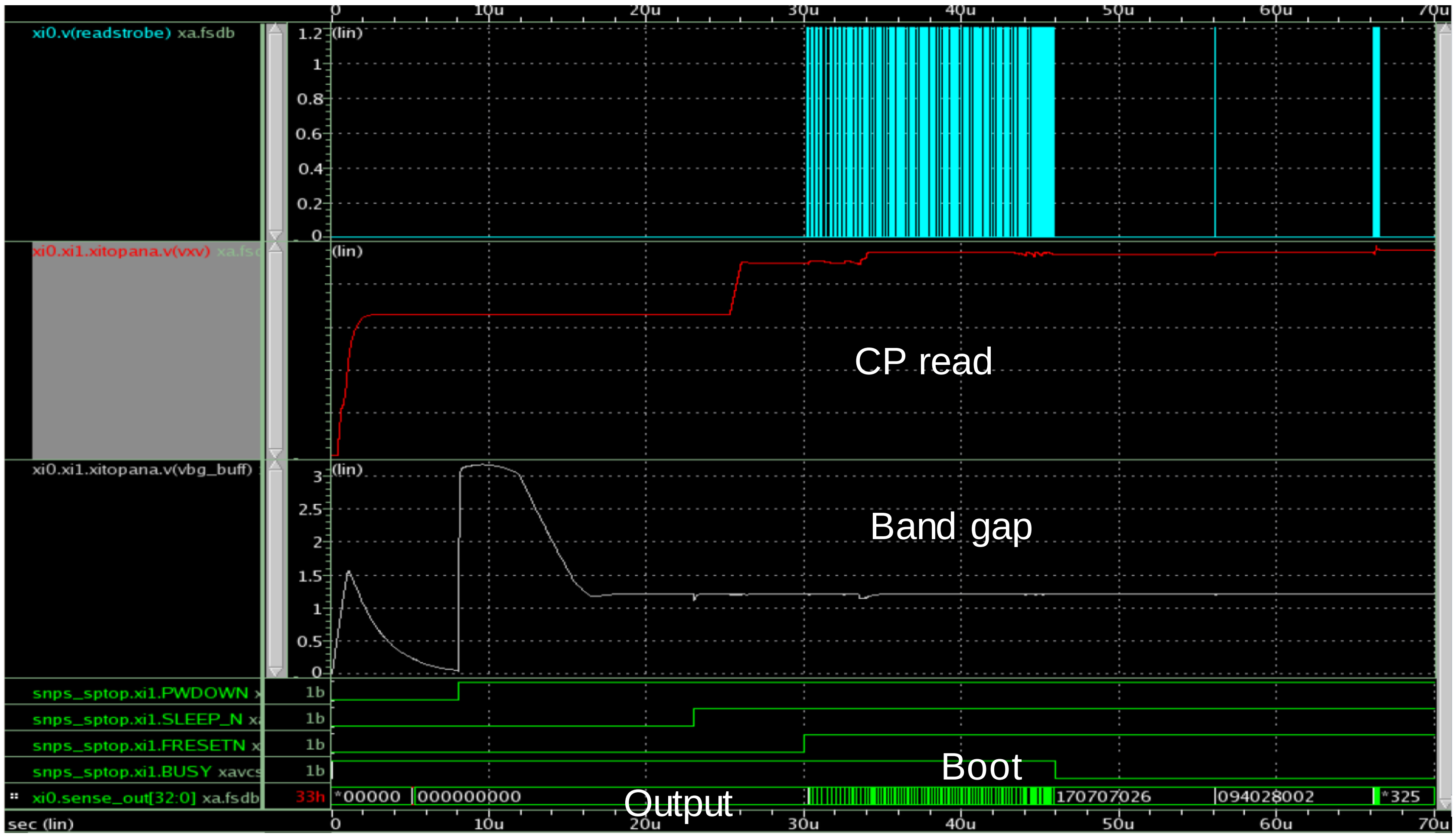
- Power up simulation takes 1h 30m
- Read operation takes 10m

- Power up + first read operation = 1h 40m

✓ With the Save and Restore capability in this case we save 1h 30m at the first operation then we save 1h 40m at the second operation, and so on…

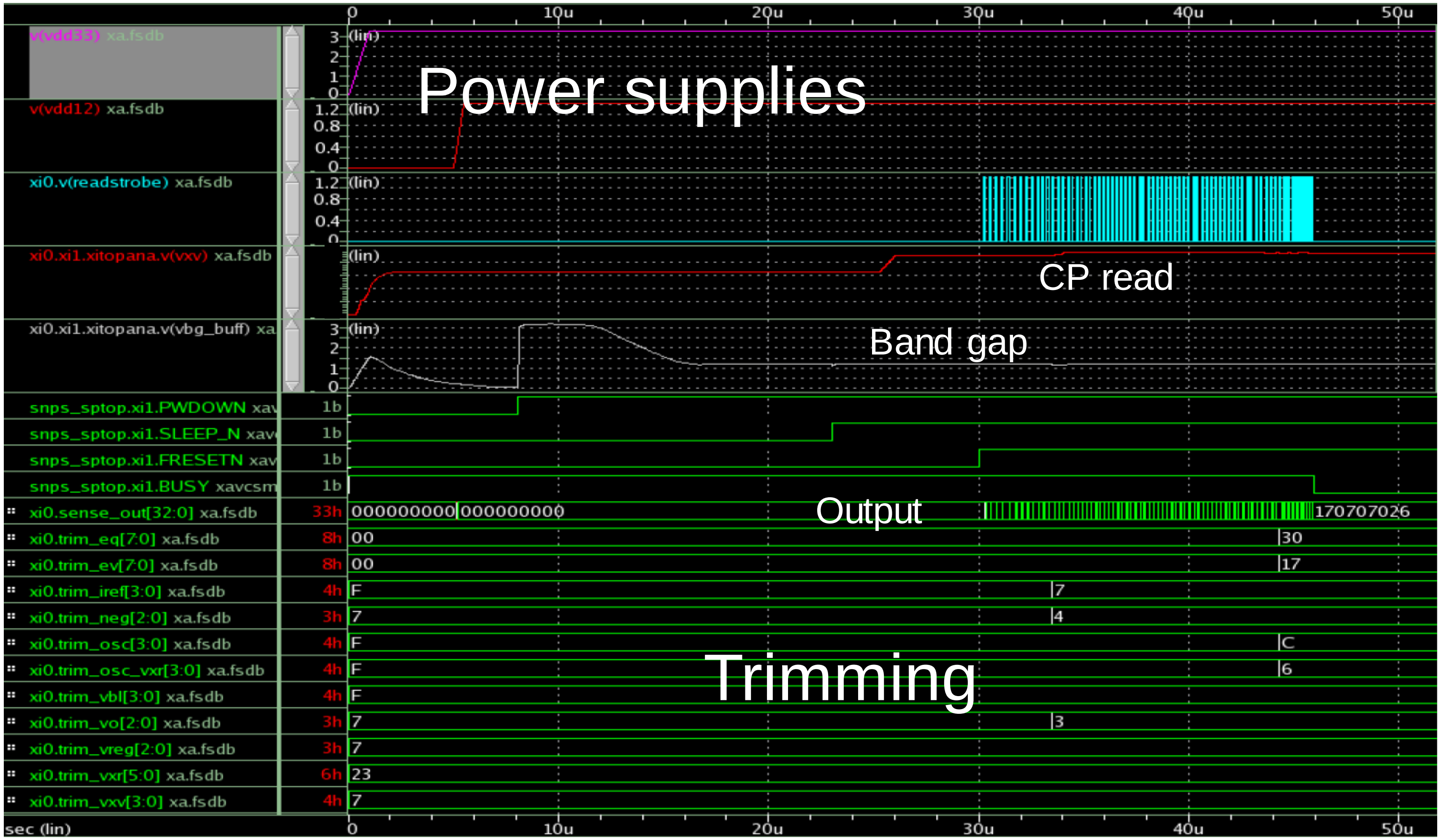
✓ With more complex operations like program and erase we can save much more time

Power-up & Multiple Read Operations

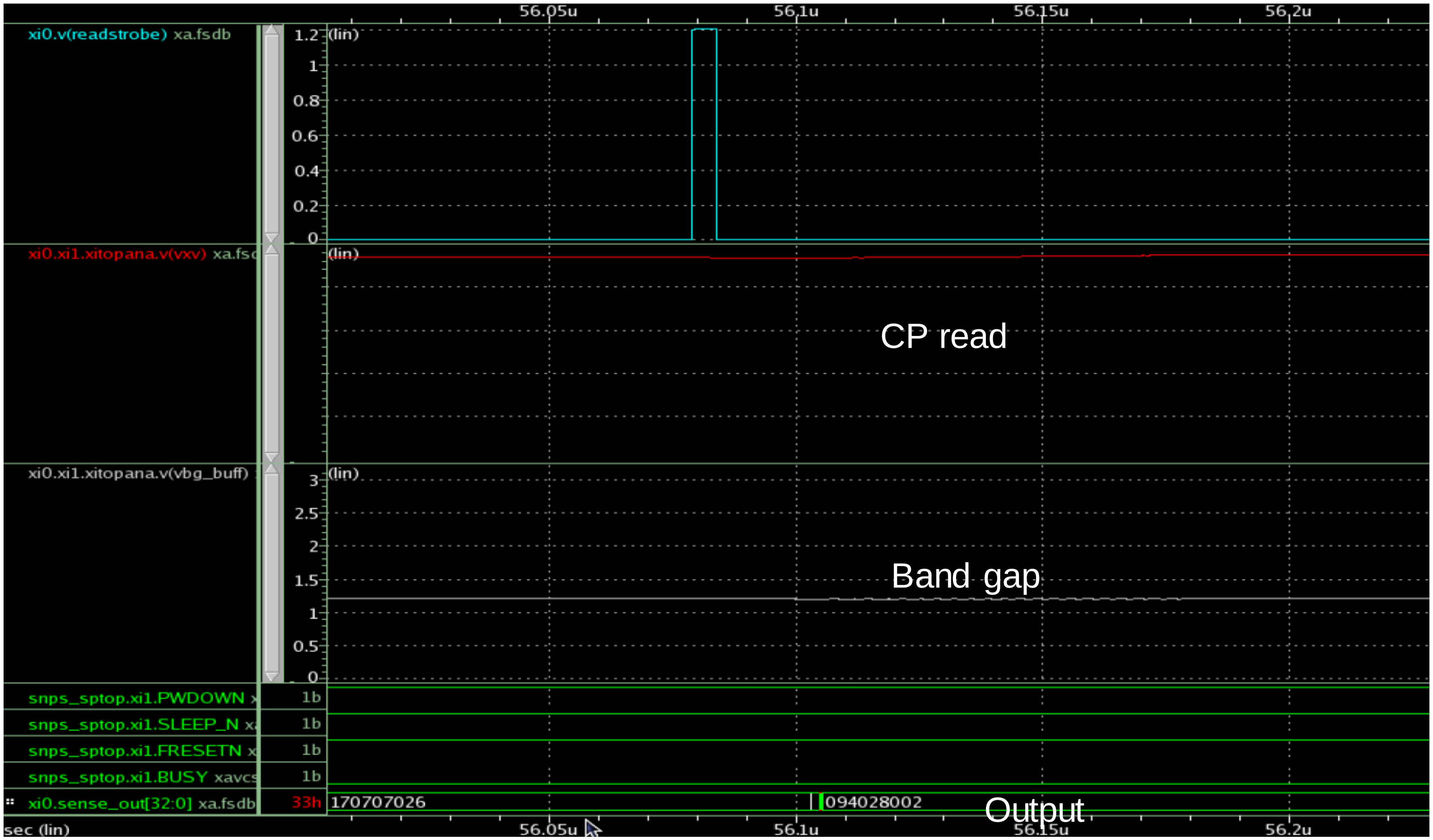


life.augmented

Power-up Details

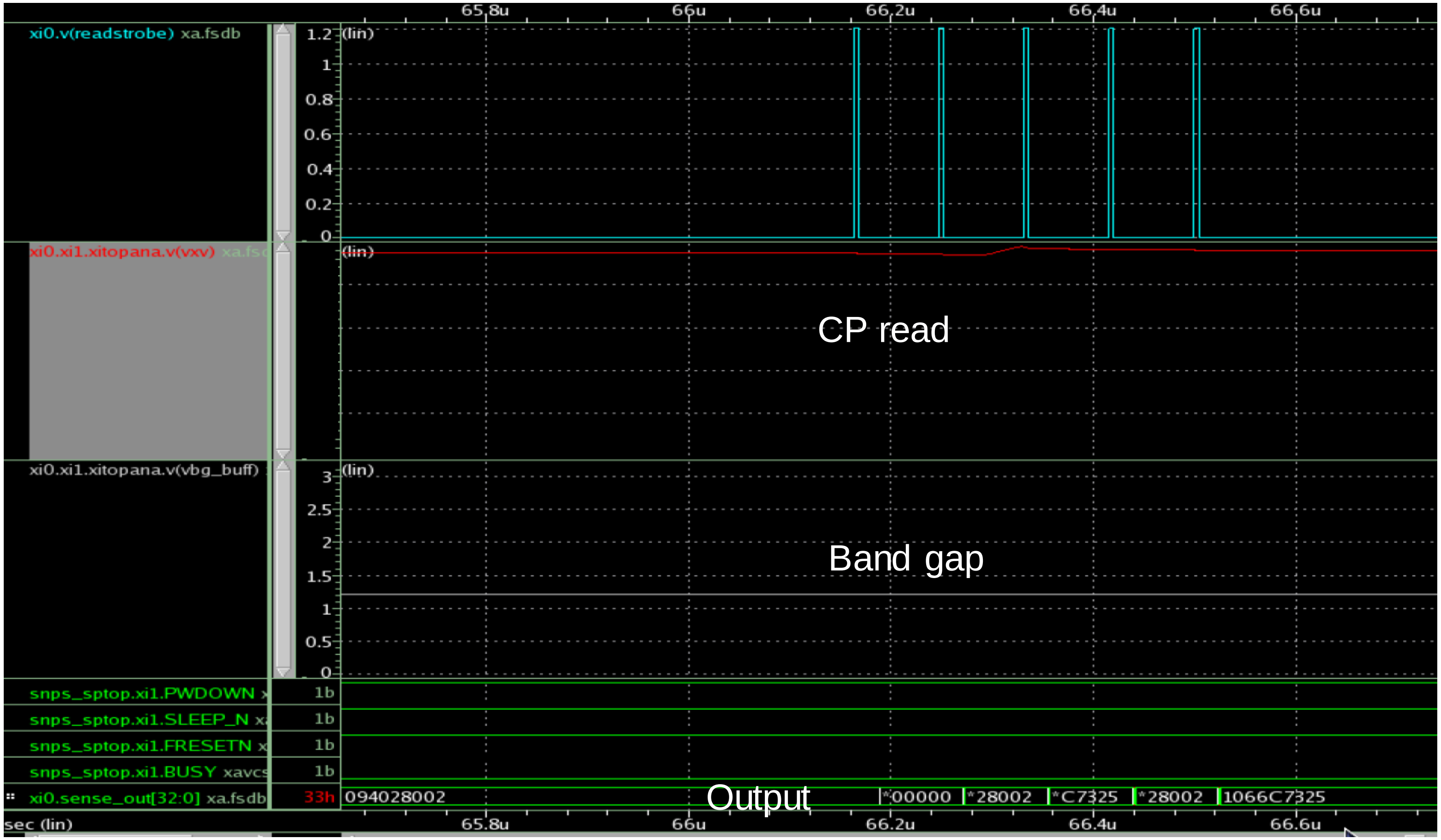


First Read Operation After Boot



life.augmented

Five Further Reads After Save & Restore



life.augmented

Conclusions

- **Synopsys VCS AMS meets ST needs for AMS design verification**
  - Fastest solution, accurate and easy to set up
  - Reliable
  - Valuable support from applications engineers
  - Collaboration with Synopsys expected soon to address GUI improvements

- **Unique features to speed up the whole design cycle**
  - Assertions to monitor analog to digital communication and prevent design failures
  - Save and restore (multi-scenario verification increases coverage)

Thank You

Behavioral modelling of Analog-on-Top Mixed-Signal ICs

Gernot Koch

DVCon, Munich, October 14th, 2014

Micronas at a glance

Known and recognized in the **automotive** and **industrial** business as a reliable global partner for **intelligent, sensor-based system solutions**

About **900** employees worldwide

Leading **supplier** of **hall sensors** for the **automotive** industry

- Core business **Automotive**
- Focus on **sensors** and **sensor-based solutions**

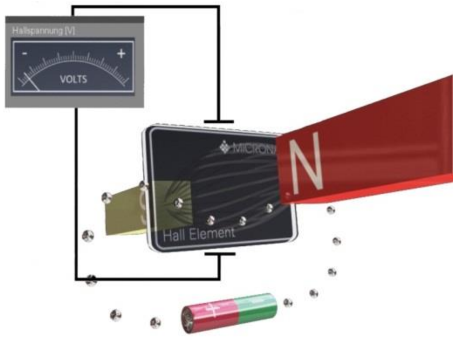
Full in-house production with own **waferfab** and **backend operations** including testing and packaging

zero ppm quality to ensure **customer** satisfaction

Commitment to **environmental** protection

AMS Applications

Hall sensors

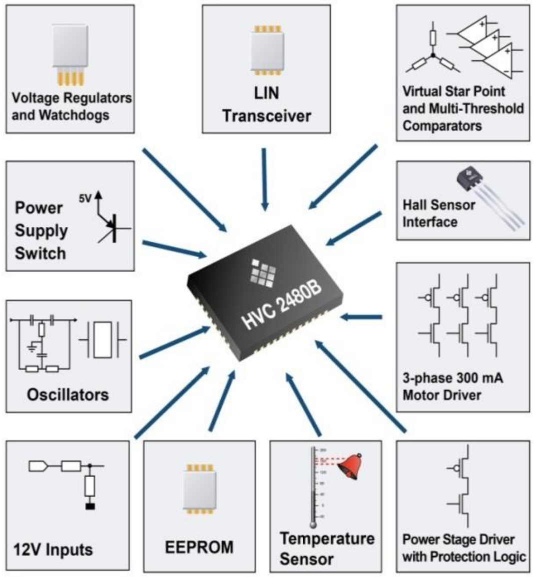


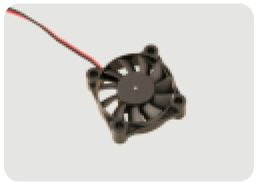
Switches
(buckle, etc.)

Linear, angular
(steering wheel)

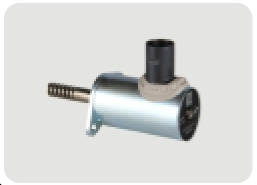
Current sensor
(power module)

HV controllers


Voltage Regulators and Watchdogs
LIN Transceiver
Virtual Star Point and Multi-Threshold Comparators
Power Supply Switch
5V
Hall Sensor Interface
HVC 2480B
Oscillators
3-phase 300 mA Motor Driver
12V Inputs
EEPROM
Temperature Sensor
Power Stage Driver with Protection Logic

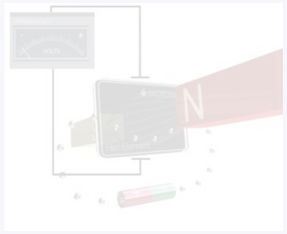
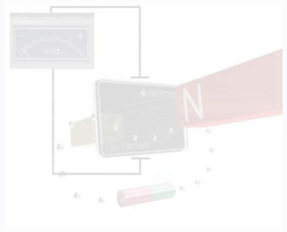
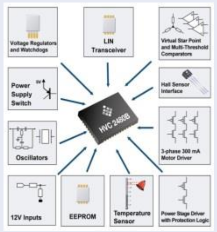


Fans



Actuators

Toplevel Simulation Strategy

Simulation style	Speed	Accuracy	Applied to
Full Spice	--	++	
Verilog / Spice	-	++	
Verilog / VerilogA(MS)	0	0	
Discrete Real-type	++	-	

Discrete Real-Type

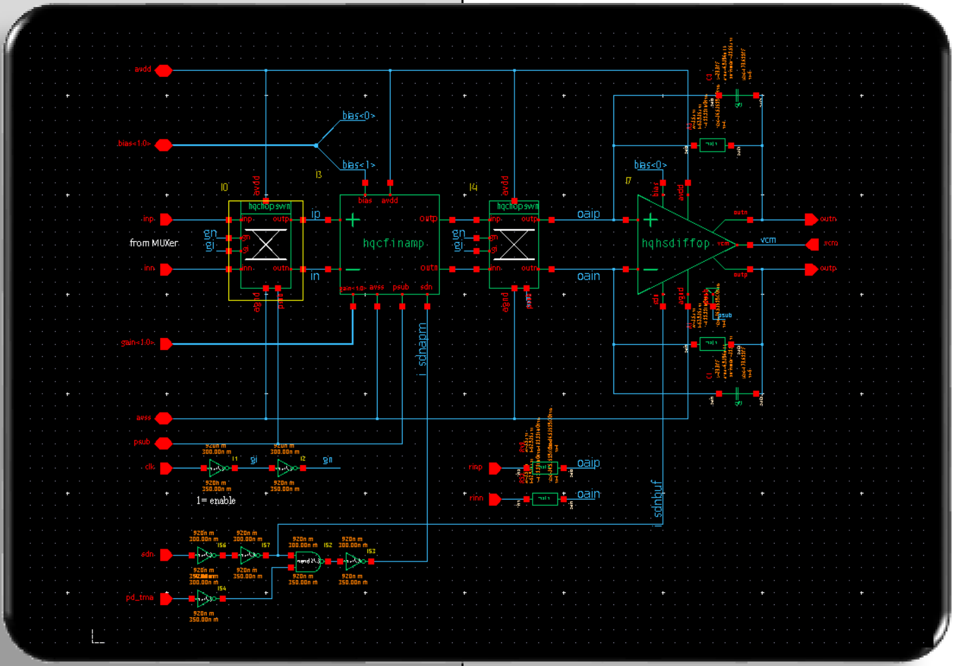
MICRONAS

- Pure digital simulation (Verilog)
- Analog behavior modeled with real values (analogy: wreal)

Time discrete
Value continuous

Example: Difference amplifier w/ programmable gain

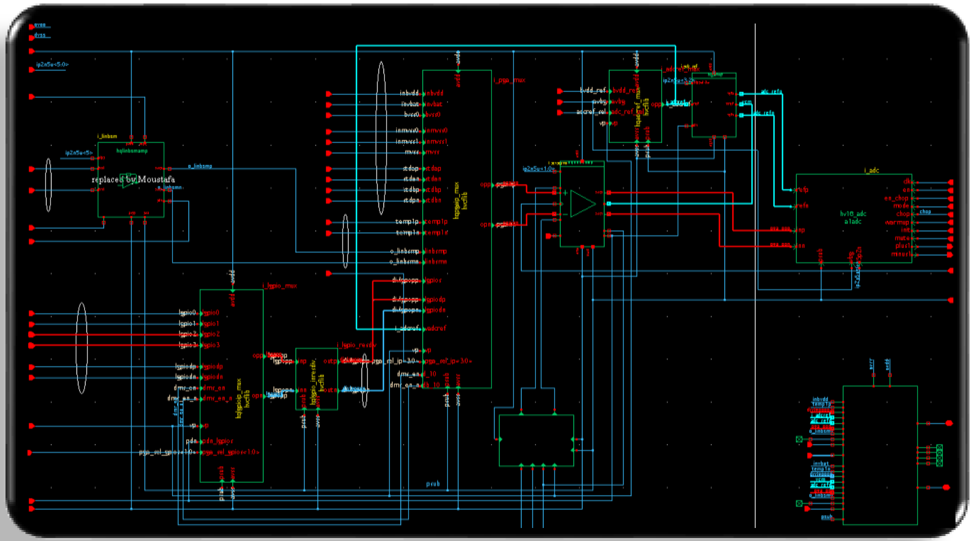
```
always @(r_inp or r_inn or r_vcm or r_rinp or r_rinn or gain) begin
  if (gain == 3)
     abs_gain = 4.0;
  else if (gain == 2)
     abs_gain = 10.0
  else if (gain == 1)
     abs_gain = 20.0;
  else
     abs_gain = 40.0;
  oaip = r_inp * abs_gain + r_rinp;
  oain = r_inn * abs_gain + r_rinn;
  if (r_vcm + (oaip - oain)/2.0 < r_avdd)
     r_outp = r_vcm + (oaip - oain)/2.0;
  else
     r_outp = r_avdd;
  if (r_vcm - (oaip - oain)/2.0 > 0.0)
     r_outn = r_vcm - (oaip - oain)/2.0;
  else
     r_outn = 0.0;
end
```



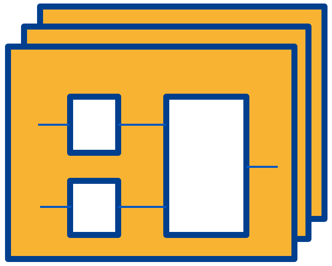
Netlisted topology

Design database

Schematic hierarchy (OA):



Representations (Views):

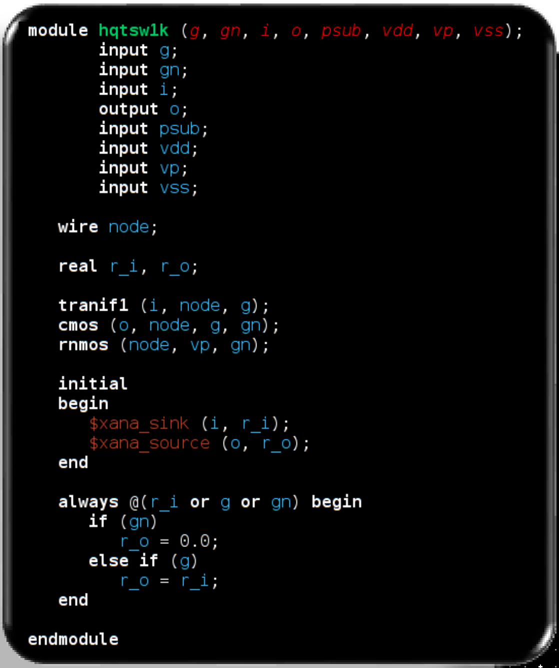


Schematic
Verilog
…

Requirements

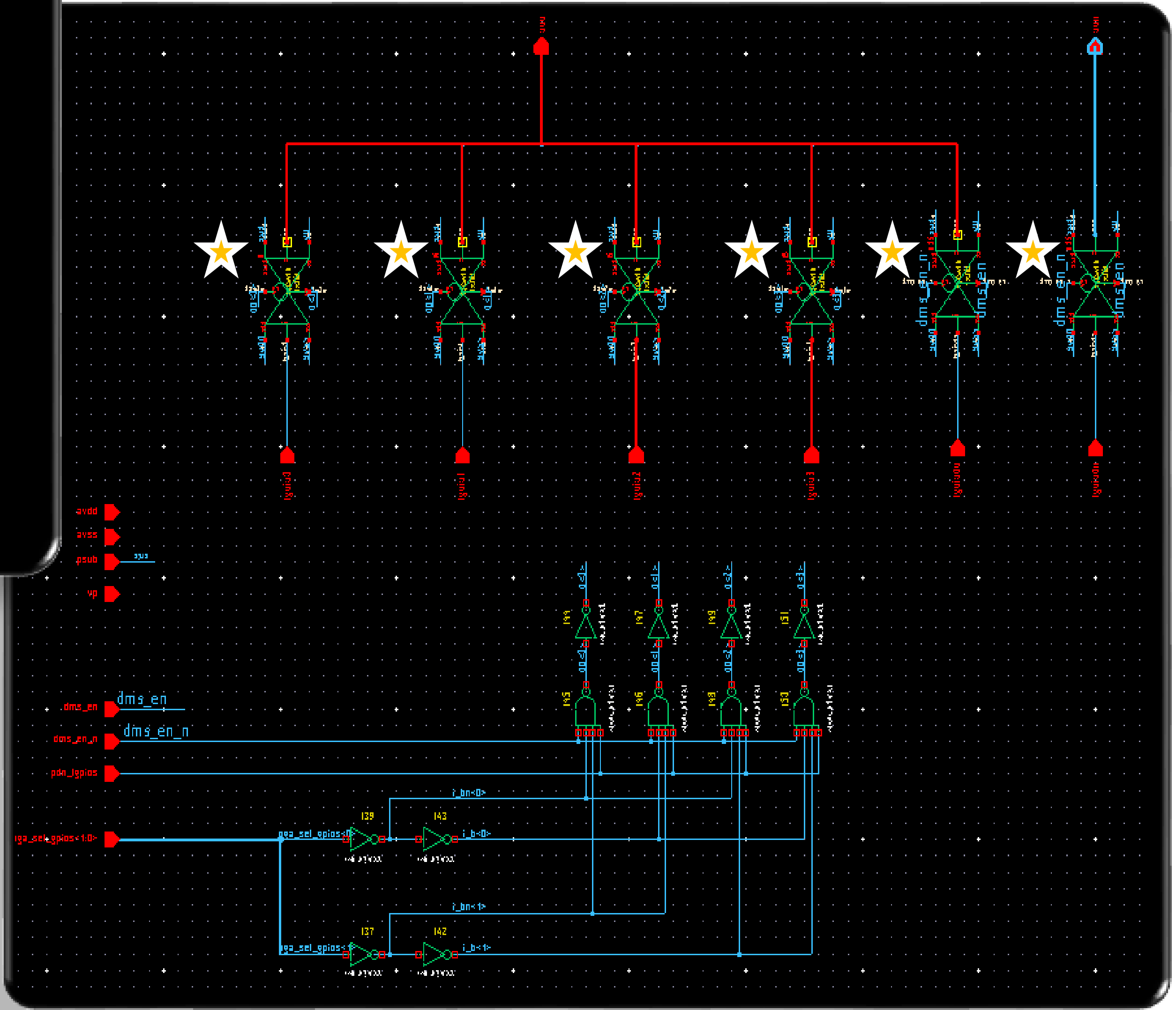
- Netlisted hierarchy
  - No out-of-sync problems
  - Only one master
  - Minimized modeling effort
- Netlisted languages
  - SPICE
  - Verilog

Netlisted topology



```
module hqtswlk (g, gn, i, o, psub, vdd, vp, vss);
       input g;
       input gn;
       input i;
       output o;
       input psub;
       input vdd;
       input vp;
       input vss;

   wire node;

   real r_i, r_o;

   tranif1 (i, node, g);
   cmos (o, node, g, gn);
   rnmos (node, vp, gn);

   initial
   begin
      $xana_sink (i, r_i);
      $xana_source (o, r_o);
   end

   always @(r_i or g or gn) begin
      if (gn)
         r_o = 0.0;
      else if (g)
         r_o = r_i;
   end

endmodule
```

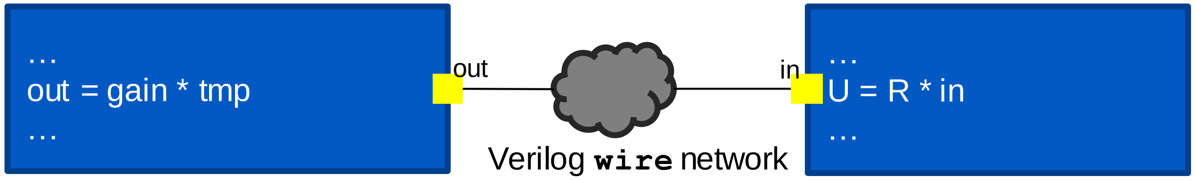
★ Model ▲ Model contained



- reuse of topology
  - 1 cell altered
- schematic logic retained & verified

Discrete Real-type (DRT) modelling

- **Signal flow**: Verilog wire to transports non-logic values



Netlisting ✓

- **Reciprocity**: Bi-directional transport
  - Verilog wire transports composite values
  - Direction configured separately for each component
  - Needed to model e.g. a LIN phy

Distributed Ohm's law ✓

- **Duality**: Verilog wire still transports digital values

Mixed digital and analog ✓

Micronas custom PLI: **$XANA**

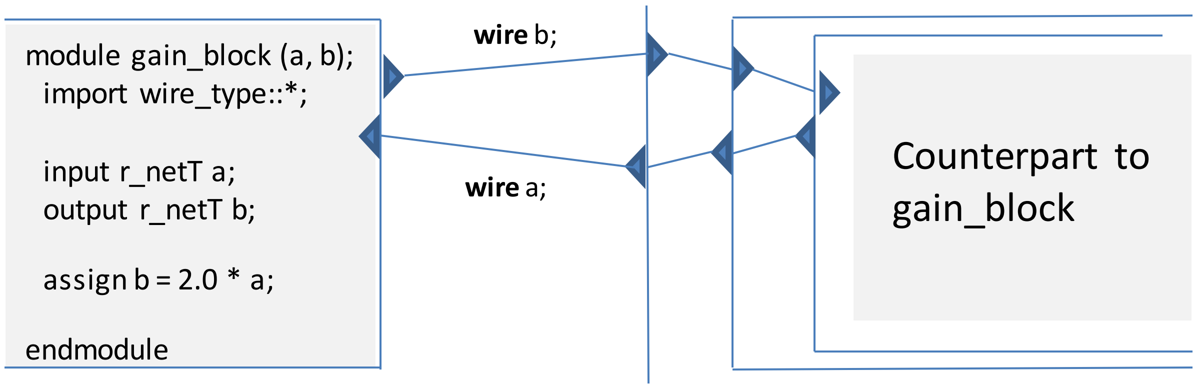
SystemVerilog extension in VCS: **-xlrm coerce_nettype**

Signal flow

Wire type definition:

```
package wire_type;

  nettype real r_netT with r_res;

  function automatic real r_res (input real drivers []);
    r_res = 0.0;
    foreach (drivers[k]) r_res += drivers[k];
  endfunction

endpackage
```

Models & hierarchy



```
module gain_block (a, b);
  import wire_type::*;

  input r_netT a;
  output r_netT b;

  assign b = 2.0 * a;

endmodule
```

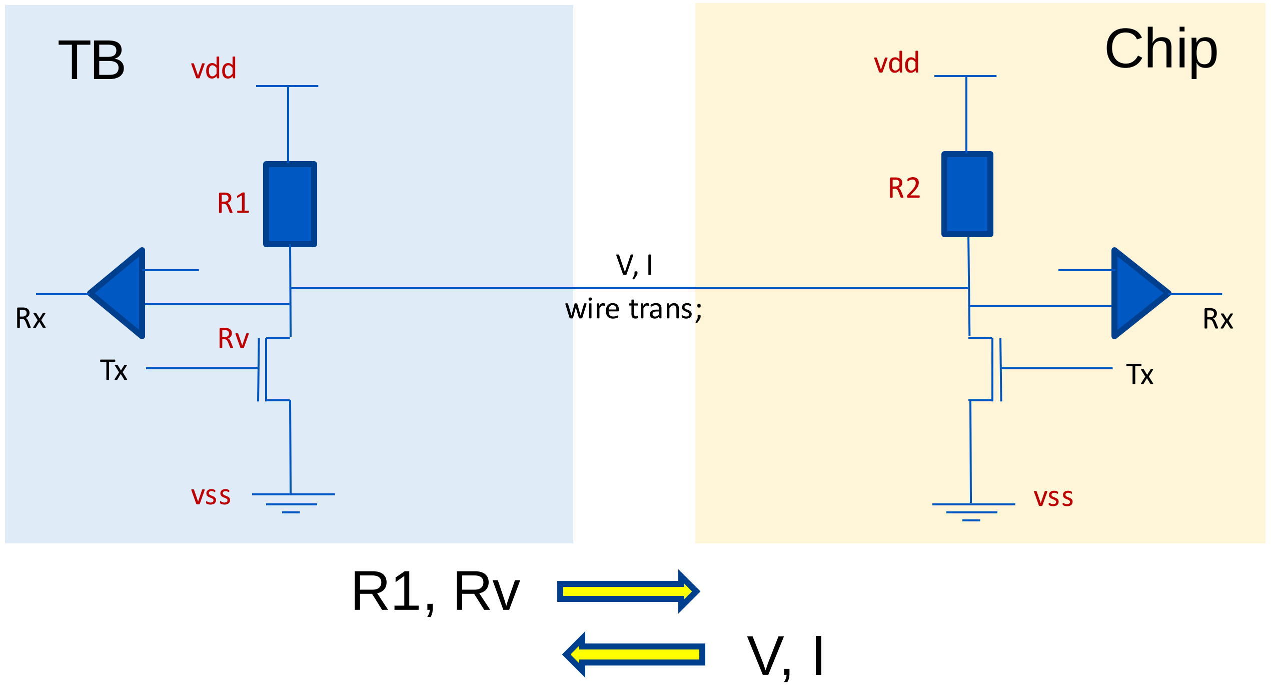
Enabled by vendor specific extension: VCS: **-xlrm coerce_nettype**

SV type 'interconnect' is similar, but requires netlister / code change

Reciprocity

E.g. LIN interface
- overcurrent
- standby
- …

```
package wire_type;

  typedef struct {
    real R1,Rv, V, I;
  } viT;

  nettype viT vi_netT with vi_res;

  function automatic viT vi_res (
   input viT drivers []);
    vi_res = '{0.0, 0.0, 0.0, 0.0};
    foreach (drivers[k]) begin
         vi_res.R1 += drivers[k].R1;
         vi_res.Rv += drivers[k].Rv;
         vi_res.V += drivers[k].V;
         vi_res.I += drivers[k].I;
    end
  endfunction

endpackage
```

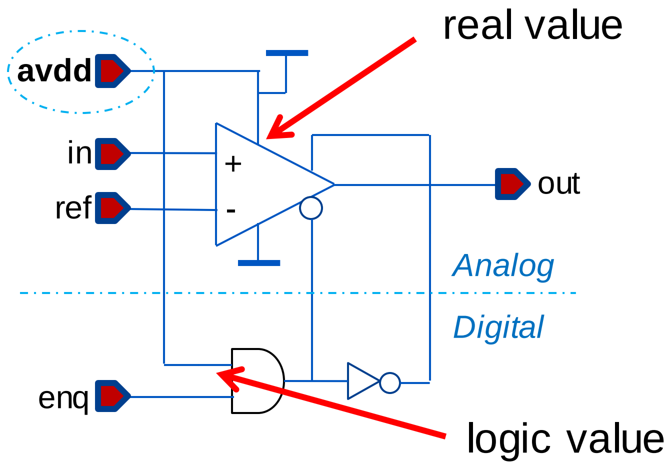


```
module TB (trans);
   import wire_type::*;
   inout vi_netT trans;
   …
   parameter r1 = 10.0;
   // check V, I, calc Rv based on Tx&R1
   …
   assign trans =  '{R1, Rv, 0.0, 0.0};
endmodule
```

```
module Chip (trans);
   import wire_type::*;
   inout vi_netT trans;
   …
   parameter r2 = 10.0
   // calc V, I, based on R2,R1,Rv,Tx
   …
   assign trans =  '{0.0, 0.0, V, I};
endmodule
```

Duality

- nets transporting real values may also need to transport verilog logic values
- e.g **avdd**
  - stimuli for Analog & Digital domains



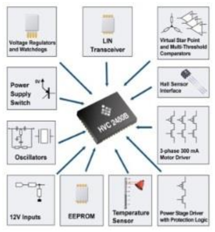
  - amplifier output out limited by avdd value & avdd is logic 1 for enable logic in schematic
- very useful for power-up mode analysis

Not yet possible with off-the-shelf simulators
Requires custom PLI

Summary, Results

MICRONAS

Run times:


Voltage Regulators and Watchdogs
LIN Transceiver
Virtual Star Point and Multi-Threshold Comparators
Power Supply Switch
Hall Sensor Interface
HVC 2480B
Oscillators
3-phase 300 mA Motor Driver
12V Inputs
EEPROM
Temperature Sensor
Power Stage Driver with Protection Logic

Realtime µC-S/W@20MHz	570µs
Discrete Real-Type models	2.8s
VerilogAMS behavioral models	354s
Verilog SPICE (single threaded)	25.5h

VCS

VCS-AMS

Modeling effort:

ADC with netlisted topology	2 days
ADC from scratch (no netlisting)	14 days

Common testbench:

- Netlisted topology allows testbench re-use for all configurations
- A few tricks allow the same simulation scenario to be applied to all configurations

Scenarios can be developed with fast turnaround using DRT models and then applied to all configurations

Questions

Thank you !