

Efficient Functional Verification for Mixed Signal IP

Jonathan David

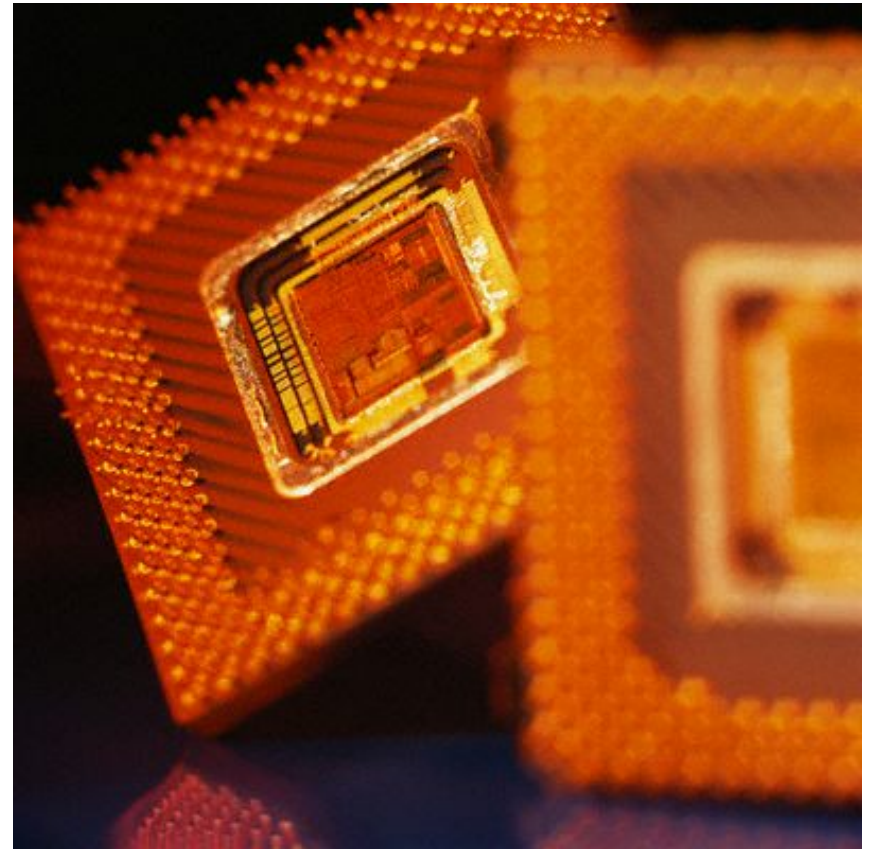
IEEE Behavioral Modeling and Simulation Workshop

October 2004

Agenda

Efficient Functional Verification for Mixed Signal IP

- Introduction
 - Requirements
 - Challenges
- Methodology
 - Flow Diagrams
 - Interface Modeling
 - Test Planning & Partitioning
- Key Models
- Case Study Results
- Conclusion



Mixed Signal Integration driven by

SOC approach, Serial I/O, cost,
smaller, faster, cheaper

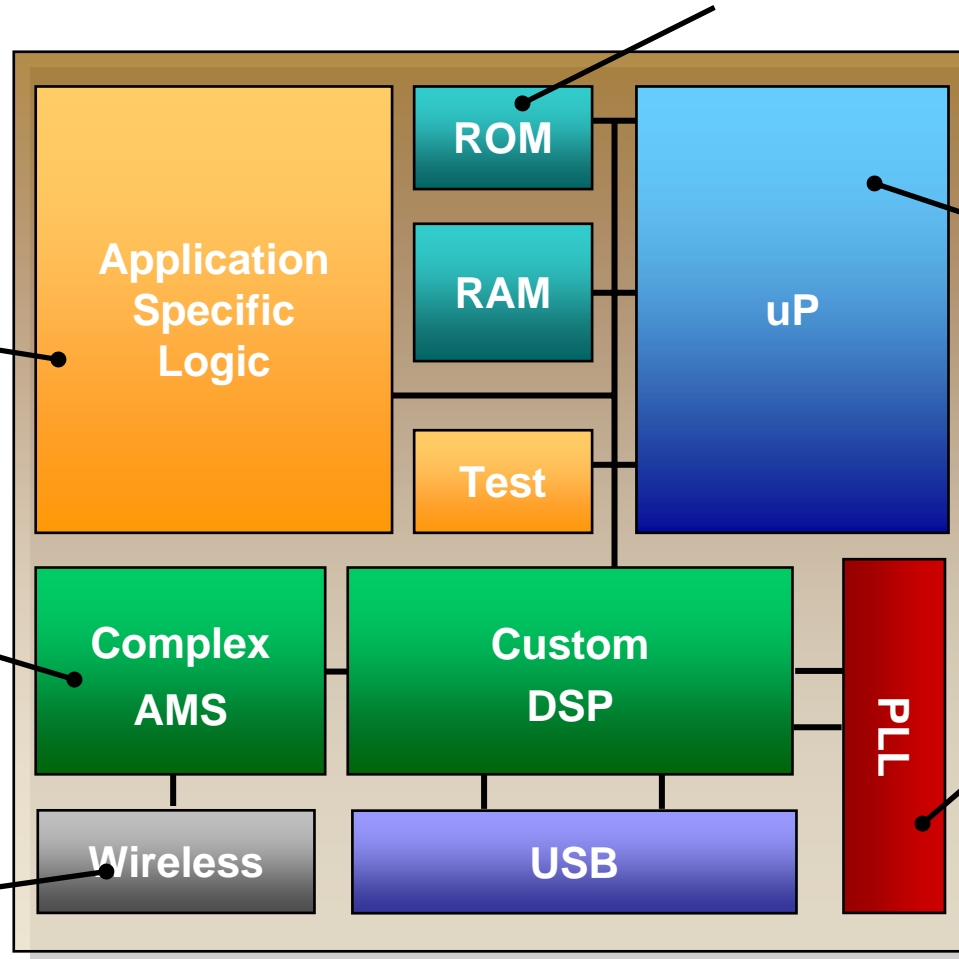


“... from our memory group”

“... high speed logic (analog techniques) from our application group”

“... Ser-Des IP for uP bus Interface”

“... from our RF specialists”



“Standard Core IP”

“... from PLL group”

The MS Design Verification Task

Does it work the way it should?

Does it meet the spec?

- Digital Specifications:
 - Function, Timing, Yield
 - Orthogonality allows Timing & Yield analyses after Synthesis, Place and Route.
 - Functional Verification is separate discipline
 - Formal methods becoming popular
- Mixed Signal Specifications
 - Function, Performance, Yield
 - “Function” complicated & linked to Performance
 - Simulation + post-processing are main tools
 - Visual review still common

Mixed Signal Verification methodology must be:



- Compatible with System (full-chip) Design Verification Methodology
 - Digital HDL compatibility – Transaction level stimulus/analysis reuse
- Compatible with Circuit Design Analysis (simulation) Methodology
 - Schematic db compatibility – Results comparable to circuit simulation – Minimal extra work by circuit designers
- Compatible with Project Schedule and Resource Availability
 - Minimal Run Times – Don't Tie up designer's compute resources
 - Or the designer.
 - Get Final answer "quickly" after design freeze.

Mixed-Signal Verification Challenges



- Time

- SPICE run time increases exponentially with circuit size $\sim N^2$
- FastSpice has lower exponent, reduced accuracy.
- Mixed Signal IP blocks can be 10-50K transistors
- Complicated “test patterns” can require long tests.

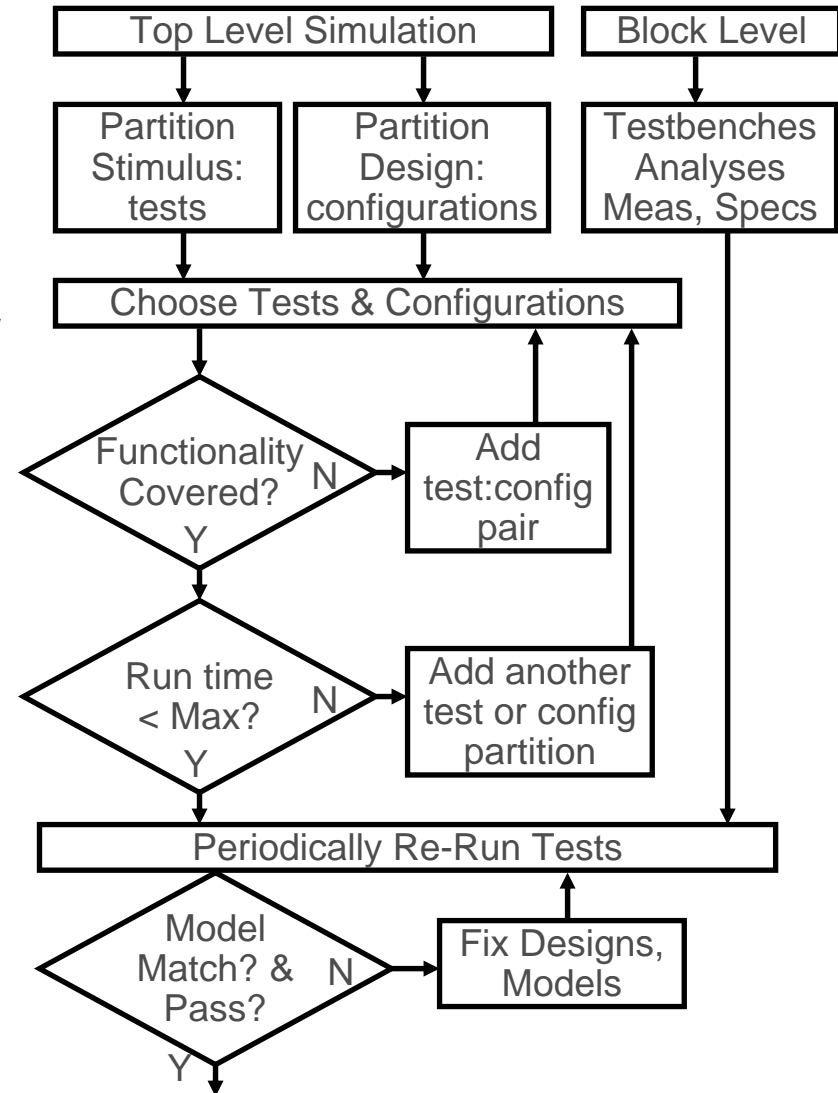
- Environment

- Minimize “programming” by circuit designers.
- while enabling maximum automation of simulation.

Overview – 2 simulation scopes



- Top Level – Full circuit
 - Use [logic] behavioral model & tests for entire Circuit
 - Blockwise replacement with transistor circuit – compare results.
 - Validate each block and all block interfaces at the device level.
- Block Level
 - Validate additional specifications
 - Calibrate behavioral model using analyses unavailable at top level
- Cell Level is assumed complete
 - Validates Cell design
 - Generates data needed for following design activities (ie synthesis)



Interfacing Digital and Analog Circuit interface types

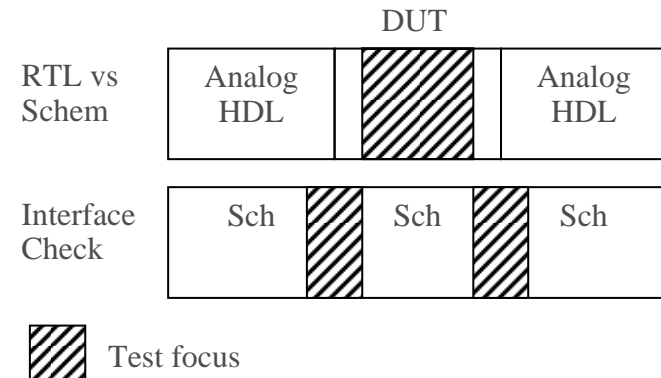
- Voltage interface – ie CMOS logic
 - Low source impedance, High load impedance
 - conversion provided in Verilog-AMS language
- Current interface – ie Bias Current Mirror
 - High source impedance, Low load impedance
 - Conversion to analog would require a “real” value for current
 - Conversion to logic with Voltage interface possible
- Balanced, Differential, ie LVDS, CML or Ethernet.
 - Matched Source and load impedance for max power transfer
 - requires custom interface models – manually inserted
- Data Converter ie ADC or DAC
 - conversion can be done directly in an AMS behavioral model

Test Planning and Partitioning



- Separate Tests
 - more tests can be run in Parallel
- Multiple Configurations
 - Fewer Transistors – More Speed
 - Model vs. Circuit each - Compare
 - Eliminate Inapplicable tests
- Interface Check Configurations
 - Longer runs so Fewer Tests
 - Just confirm Interface working correctly.

Configuration	Test 1	Test 2	Test 3	Test 4	Test 5	Test 6	Test 7	Test 8	Test 9	Test 10
Behavioral Only	X	X	X	X	X	X	X	X	X	X
Block 1	X	X	X			X	X			X
Block 2	X	X	X	X	X		X	X	X	
Block 3	X	X	X				X			
Block 4	X	X	X				X			
Block 5	X	X	X				X			
IF Chk Block 1	X		X							X
IF Chk Block 2	X	X					X			
IF Chk Block 3	X		X				X			
IF Chk Block 4	X			X		X				
IF Chk Block 5	X				X					X



Ethernet PHY Test & Configuration Plan

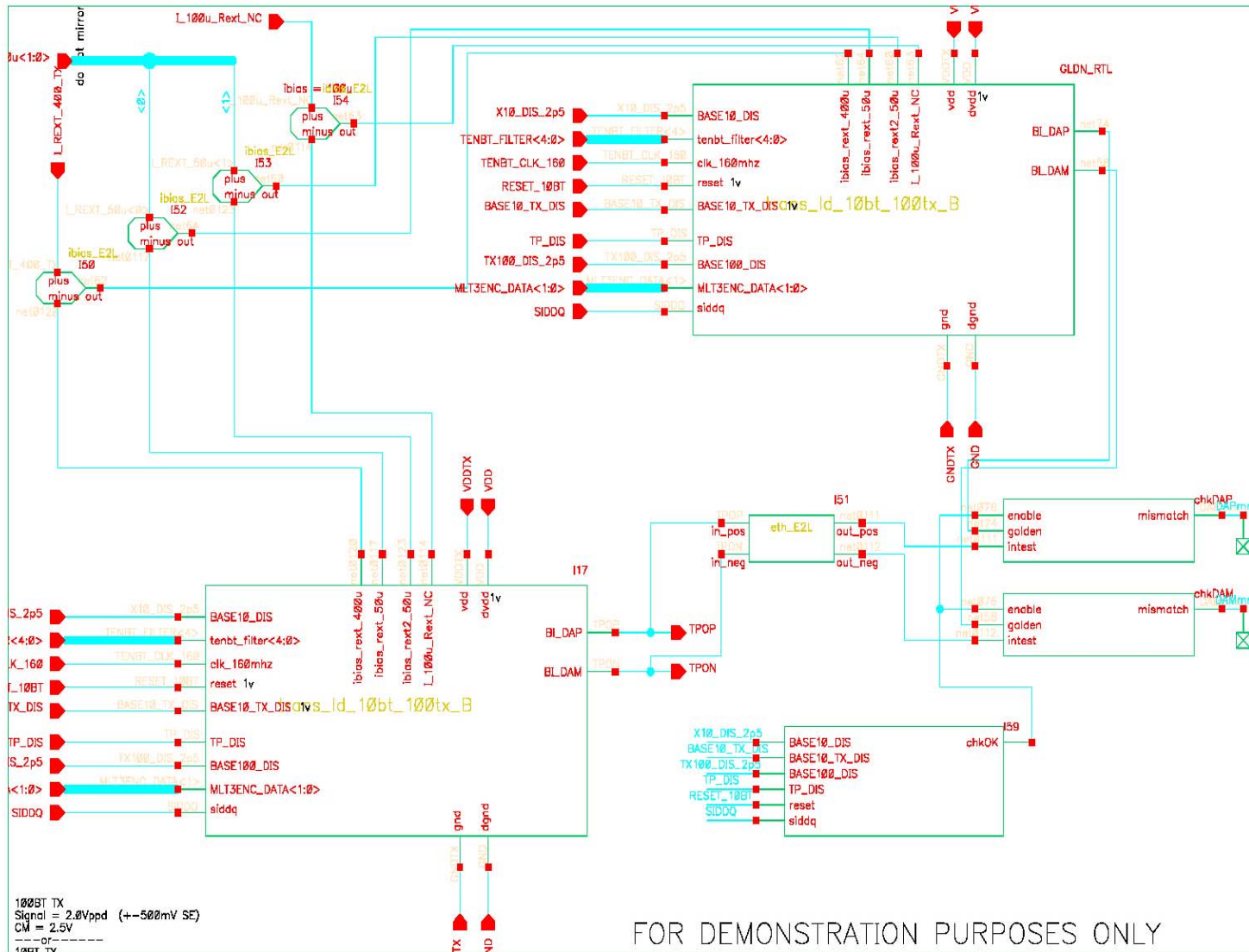


Model Type Legend

- R RTL (digital model)
- S Schematic (transistors)
- G Gate (gate level netlist)
- A Analog Behavioral
- M Mixed-Signal Behavioral
- X Compare - Schematic + RTL ref model
- E Extracted - from layout with parasitics

Configuration	Common_BG	Common_PLL	Bias_Port	Trans	AnCtr	DigCtr	RCVR	Channel Out	Channel In	2nd Phy	CONF		X_FD100				X_FD10			
											Cnf FD100	Cnf FD10	X_sml_ones	X_sml_rndm	X_lrg_ones	X_lrg_rndm	X_sml_ones	X_sml_rndm	X_lrg_ones	X_lrg_rndm
Functional Verif	R	R	R	R	R	R	R	R	R	R	X	X	X	X	X	X	X	X	X	X
Functional Verif ExtLpBk	R	R	R	R	R	R	R	R			X	X	X			X	X			X
TRANS model ck	A	R	A	X	R	R	R	M			X	X	X	X	X		X	X	X	
TRANS IF ck Int	S	R	S	S	S	R	R	M			X	X	X				X			
TRANS IF ck - pads	A	R	A	S	R	R	M	S			X	X	X				X			
BIAS model ck	A	R	X	A	R	R	M	M			X	X	X				X			
BIAS IF ck TRANS	S	R	S	S	S	R	M	M			X	X	X				X			
BG model ck	X	M	A	M	R	R	*	M			X	X	X				X			
BG IF chk	S	S	S	S	R	R	*	M			X	X	X				X			
RCVR model chk	-	-	-	-	-	R	X*	A			-	-	-	-	-	-	-	-	-	-
RCVR IF chk	-	-	-	-	-	R	S*	S			-	-	-	-	-	-	-	-	-	-

RTL vs. Schematic Comparison



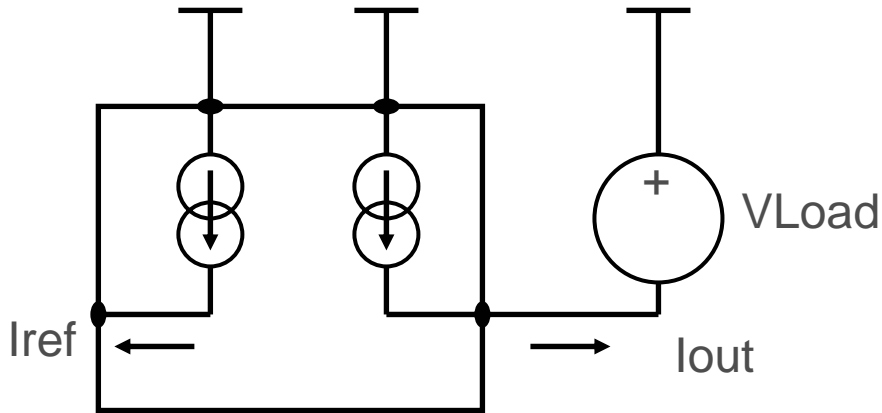
eth_E2L.vams

differential signal to differential logic



```
// VFS_AMS_PHY180.eth_E2L:verilogams
// last revised: 07/20/04 jbdavid
`include "constants.vams"
`include "disciplines.vams"
// DEFINE & TIMESCALE :
`timescale 1ns/10ps
module eth_E2L (
    output out_neg, out_pos,
    input in_neg, in_pos );
    logic out_pos, out_neg;
    electrical in_pos, in_neg;
    reg inneg, inpos;
// INTERNAL NODES :
    assign out_pos = inpos;
    assign out_neg = inneg;
    parameter real vth = 0.5;
// diff input threshold for detection
    always @(cross(V(in_pos,in_neg)
        - vth,1)) begin
        inpos = 1; inneg = 0;
    end
    always @(cross(V(in_pos,in_neg)
        - vth,-1)) begin
        inpos = 0; inneg = 0;
    end
    always @(cross(V(in_pos,in_neg)
        + vth,1)) begin
        inpos = 0; inneg = 0;
    end
    always @(cross(V(in_pos,in_neg)
        + vth,-1)) begin
        inpos = 0; inneg = 1;
    end
endmodule
```

Current Mirror Model Calibration



DC sweep: VLoad 0 to Vcc
record $H(\beta) = I_{out}/I_{ref}$

```
analog begin ...  
    Iref_rext = I( I_REXT_REF100 ,VSS);  
    Hext100_1 = $table_model( V(VDD,I_REXT_100[1]),  
        "Hext100_1.tbl", "1CL");  
    I(VDD, I_REXT_100[1]) <+ Iref_rext*Hext100_1;  
... end  
endmodule
```

```
# Table File :  
'/cds/vsde41/tools/acv/etc/components/s  
pectre_prim/Bias_port/modelCal/Bias_por  
t_Hext100_0.tbl'  
# Result File :  
'results/VFS_AMS_PHY180_sims/bias_port_  
test/schematic/DefaultSet/PvtCharvout-  
BiasPortGain.res'  
# Created on : 'Sat Aug 7 2004 at  
20:48:06'  
# Created by : 'jbdavid'
```

```
# Vload  
#           Hext0  
0           0.0      # keys='0 '  
0.05       0.284484 # keys='0.05 '  
0.1        0.515482 # keys='0.1 '  
0.15       0.69403  # keys='0.15 '  
0.2        0.822203 # keys='0.2 '  
0.3        0.949932 # keys='0.3 '  
0.4        0.981635 # keys='0.4 '  
0.5        0.98805  # keys='0.5 '  
1          0.989314 # keys='1 '  
1.5       0.989427 # keys='1.5 '  
2         0.989507 # keys='2 '  
2.5       0.989598 # keys='2.5 '
```

Bias_port.va

Current Mirror Model



```
// VFS_AMS_PHY180, Bias_port, vloga
// last revised: 8/07/04 jbdavid
// DESCRIPTION : this is a
// calibrated behavioral model
// LIMITATIONS : no output clamp on SIDDQ
`include "constants.vams"
`include "disciplines.vams"
//=====
module Bias_port( I_REXT_REF100,
  SIDDQ, VDD, VSS,
  I_REXT_100 ); // PINS :
output [1:0] I_REXT_100;
input I_REXT_REF100;
input SIDDQ, VDD, VSS;
electrical [1:0] I_REXT_100;
electrical I_REXT_REF100;
electrical SIDDQ, VDD, VSS;
// PARAMETERS: (Comment each one)
// Model calibrated at '20:48:06' on 'Sat Aug 7 2004' by 'jbdavid'
parameter real VdsatIN = 1.013308; //VdsatIN=0.5
parameter real VthSIDDQ = 1.2;
parameter real VminOut = 0.7;
parameter real Roffext = 1.000571e+12; // Roffext = 1G
parameter real Cdsrext100_1 = 2.698685e-13 ;
// Cdsrext100_1 = 200f
parameter real Cdsrext100_0 = 2.698685e-13 ;
// Cdsrext100_0 = 200f
```

```
// LOCAL VARIABLES: (Comment each one)
integer siddq; // = 1(true) if siddq > vth other wise 0
real Iref_rext, Vdd;
real Hext100_1, Hext100_0;
real vdsatin;
//-----
resistor #(.r(Roffext)) RinIext (I_REXT_REF100, VSS);
capacitor #(.c(Cdsrext100_1)) C8 (I_REXT_100[1], VDD);
capacitor #(.c(Cdsrext100_0)) C9 (I_REXT_100[0], VDD);
//-----
analog begin
  siddq = V(SIDDQ,VSS) > VthSIDDQ?1:0; // "bias" signal,
  vdsatin = VdsatIN;
  if (!siddq) begin
    V( I_REXT_REF100 ,VSS) <+ vdsatin;
  end
  Iref_rext = I( I_REXT_REF100 ,VSS);
  Hext100_1 = $table_model( V(VDD,I_REXT_100[1]),
    "Hext100_1.tbl", "1CL");
  I(VDD, I_REXT_100[1]) <+ Iref_rext*Hext100_1;
  Hext100_0 = $table_model( V(VDD,I_REXT_100[0]),
    "Hext100_0.tbl", "1CL" );
  I(VDD, I_REXT_100[0]) <+ Iref_rext*Hext100_0;
  I(VDD , VSS) <+ 3* Iref_rext;
end
endmodule
```

Top-Level Run Results



- Sample runs consisted of 3 tests in 1 run,
 - 100Mb FD Ext. Loopback configuration, Transfer short block, long block
- Configurations focused on
 - Top level functional model
 - Transmit block model checks
 - Transmit block Interface checks

Configuration	Trans Count	Ams (spectre)	Ams (Accel.)	Ams (Ultrsim)
Functional Verif	0	2.75 min	2.75 min	--
Functional Verif (schematics)		--	--	31 hr
TRANS model ck	547	64.8 hr	19.7 hr	3.5 hr
TRANS IF ck Int	1126	114.7 hr	37.8 hr	4.5 hr

Top-Level Regression Test Estimates



Sim Engine	CPU Hours			Percent savings
	Full Transistor Level Design	Partitioned - Transistor Level I/F	Partitioned - Characterized Behavioral I/F	
Spectre standard	37008	6935	5142	86%
Spectre accelerated	12336	2312	1714	86%
Ultrasim	780	224	194	75%
Spectre standard	9 wk 24 cpus	12 d 24 cpus	9 d 24 cpus	
Spectre accelerated	3 wk 24 cpus	4 d 24 cpus	3 d 24 cpus	
Ultrasim	1 wk 5 cpus	5 d 2 cpus	4 d 2 cpus	

65 tests, 29 configurations in top-level regression suite

Requires: ~3 man-days per model for development, test and calibration setup.

Conclusions



demonstrated partitioning approach

- Provides a verified HDL model for system design verification
- Increases the ability to trade additional CPUs for Schedule.
- Reduces CPU requirements or Verification Cycle time by ~80%
- Requires minimal designer interaction during verification cycle
- Uses a minimum number of custom interface behavioral models
 - Development, validation and calibration simplified through use of commercially available Aptivia environment

cadence

**BREAK ON
THROUGH**