

HSIM® Simulation Reference Manual

Version M-2017.03, March 2017

SYNOPSYS®

Copyright and Proprietary Information Notice

©2017 Synopsys, Inc. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at <https://www.synopsys.com/company/legal/trademarks-brands.html>.

All other product or company names may be trademarks of their respective owners.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043
www.synopsys.com

Contents

Related Products and Trademarks.....	xli
Related Publications	xli
Conventions.....	xlii
Customer Support	xliii

Part I: HSIM Core Basics

1. Introduction HSIM	3
HSIM Features.....	3
Interactive Circuit Analysis.....	4
HSIM Applications	5
Input/Output Data.....	6
Hierarchical Simulation Technology.....	6
Limitations and Recommendations	7
 2. Simulation Flow	 9
Simulation Flow Overview	9
Supported Output Formats	11
HSIM Quick Start.....	12
Power Supply Settings	15
Using HSIM in a Nanometer VLSI Design Flow.....	15
Pre-Layout Design Flow.....	15
Synthesizable Logic	17
Non-Synthesizable Logic	17
Analog/Memory	18
Post-Layout Design Flow	18
Circuit Extraction and Analysis.....	20
Interconnect Segmentation Resolution	21
Design Optimization.....	23

Contents

3. HSIM Command Groups	25
Global Macro Set-Up Commands	26
Netlist Commands	27
Digital Vector File Commands	27
Ungrounded/Grounded Capacitor Commands	27
Isomorphic Matching Commands	29
DC Initialization Commands.....	29
Performance/Accuracy Control Commands.....	30
Device Model Commands	33
Connectivity Checking Commands	35
Activity Checking Commands	35
High-Impedance Node Checking Commands	35
Signal Net Post-Layout Commands.....	37
DPF/SPEF Backannotation Commands	38
Selected Net Backannotation Command.....	40
Output Control Commands	40
AC and DC Analysis Commands.....	41
Monte Carlo Analysis Commands	41
Verilog-A Commands.....	42
Miscellaneous Commands.....	42
4. HSIM® Commands in Alphabetical Order	45
.BIASCHK	46
HSIMABMOS.....	52
HSIMABSVOLTOL	52
HSIMACCSUBTHTBL	53
HSIMACCURATERDS	53
HSIMACE	54
HSIMACFREQSCALE	54
HSIMACHECKFANOUT	54

HSIMACHECKFILENAME	55
HSIMACHECKOUTFMT	55
HSIMACHECKRFEDGE	56
HSIMACOUT	57
HSIMACOUTFMT	59
HSIMALLOWEDDV	59
HSIMAMOS	60
HSIMANALOG	61
HSIMANLASCIPRSN	62
HSIMASIM	63
HSIMAUTOADV	63
HSIMAUTOVDD	64
HSIMAUTOVDDSUP	65
HSIMB4REMOVE	65
HSIMBISECTION	66
HSIMBJTCC	70
HSIMBJTCV	70
HSIMBMOS	70
HSIMBSIM3ISUB	70
HSIMBUSDELIMITER	71
HSIMCAPCC	71
HSIMCAPFILE	72
HSIMCC	72
HSIMCHECKMOSBULK	74
HSIMCMIN	74
HSIMCOILIB	74
HSIMCONNCHECK	75
HSIMDCI	76
HSIMDCINIT	76
HSIMDCITER	77
HSIMDCOUTFMT	77

Contents

HSIMDCSOIACC	78
HSIMDCSTEP	78
HSIMDCSTOPAT	79
HSIMDCV	79
HSIMDELAYNLRMIN	79
HSIMDELVTO	80
HSIMDETAILBSIM4	80
HSIMDIOCC	81
HSIMDIOCV	81
HSIMDIODECURRENT	81
HSIMDIODEVT	82
HSIMDMR	82
HSIMDPF	82
HSIMDPFHIERID	83
HSIMDPFMULTISUB	83
HSIMDPFPFX	83
HSIMDPFREPORTNOBA	84
HSIMDPFSCALE	84
HSIMDPFSFX	85
HSIMDPFSPLITFD	85
HSIMDTNAME	86
HSIMDUMPOPI	86
HSIMEFFICIENTTBL	86
HSIMENHANCECAP	87
HSIMENHANCEDDC	87
HSIMENHANCEDIDDQ	88
HSIMENHANCEMOSIV	89
HSIMFLAT	89
HSIMFMODLIB	90
HSIMFORWARDBIASDIODE	90
HSIMFSDBDOUBLE	90

HSIMGATEMOD	91
HSIMGCSC	91
HSIMGMIN.....	92
HSIMGMINDC	92
HSIMGMIWELM	93
HSIMHASCOMPLEXBJT.....	93
HSIMHIERID	94
HSIMHIEROPT	94
HSIMHONORX	94
HSIMHSPICEVEC.....	95
HSIMHZ	95
HSIMHZALL.....	96
HSIMHZFANOUT.....	96
HSIMHZNDNAME	97
HSIMHZSTART	97
HSIMHZSTOP	98
HSIMHZTIME	98
HSIMHZXSUBCKT	98
HSIMIDDQ.....	98
HSIMIGISUB	99
HSIMINACTOPT	100
HSIMIMPORT.....	100
HSIMIPRECISION.....	101
HSIMITERMODE	102
HSIMKEEPALLGNDCAPS.....	102
HSIMKEEPNODESET.....	103
HSIMLMIN	103
HSIMLIS.....	103
HSIMLOGDCPROGRATE	104
HSIMLOGPROGRATE	104
HSIMLUMPCAP.....	104

Contents

HSIMMATCHOPT	105
HSIMMATRIXOPT	106
HSIMMEASOUT	106
HSIMMEASWINDOW	107
HSIMMONITORMRES	108
HSIMMODELSTAT	108
HSIMMONTECARLOINST	109
HSIMMONTECARLOMOD	109
HSIMMONTECARLOSAVEOUT	109
HSIMMOSPRECISION	112
HSIMMQS	113
HSIMMULTIDC	114
HSIMMSGFILTER	114
HSIMNODCNODES	115
HSIMNODECAP	115
HSIMNOMULTIEND	116
HSIMNOSIMTIME	116
HSIMNTLFMT	116
HSIMNTLRMIN	117
HSIMNVBS	117
HSIMNVDS	118
HSIMNVGS	118
HSIMOPCOMPRESS	118
HSIMOPTSEARCHEXT	119
HSIMOUTPUT	119
HSIMOUTPUTFLUSH	120
HSIMOUTPUTFSDBSIZE	121
HSIMOUTPUTIRES	121
HSIMOUTPUTMEAS	121
HSIMOUTPUTTBL	122
HSIMOUTPUTTRES	122

HSIMOUTPUTTSTEP	122
HSIMOUTPUTVRES	123
HSIMOUTPUTWDFSIZE	123
HSIMPARAMCACHING	124
HSIMPAPPRECISION	124
HSIMPORTCR	125
HSIMPORTI	125
HSIMPORTV	126
HSIMPOSTL	126
HSIMPOSTLMANY2M	128
HSIMPOSTLONE2M	128
HSIMPREFERVERILOGA	128
HSIMPREFLAT	129
HSIMPRINTSIMSTATUS	130
HSIMPROBEALL	130
HSIMPROBEAUTO	131
HSIMPROBEFILTER	131
HSIMPROBEREST	132
HSIMRASPFMODXY	132
HSIMRATSTART	132
HSIMRATSTOP	133
HSIMRAVTS	133
HSIMRCMATCHRELERR	134
HSIMRCNPO	134
HSIMRCPRECISION	135
HSIMRCRIO	136
HSIMRCRKEEPLEM	137
HSIMRCRKEEPMODEL	137
HSIMRCRKEEPNODE	138
HSIMRCRTAU	138
HSIMREDEFSUB	139

Contents

HSIMREDEFSUBNAME	139
HSIMREGNODE	142
HSIMRELCURTOL	144
HSIMREPORTAREA	144
HSIMRGATEMOD	145
HSIMRMAX	145
HSIMRMIN	145
HSIMRMVDIO	146
HSIMSAMPLERATE	146
HSIMSCALE	147
HSIMSELBA	147
HSIMSKIP	147
HSIMSNCS	148
HSIMSOIBULKMOD	149
HSIMSPEED	150
HSIMSPF, HSIMSPEF	154
HSIMSPFADDNETPINXY	154
HSIMSPF	155
HSIMSPFCC	155
HSIMSPFCCSCALE	155
HSIMSPFCHLEVEL	156
HSIMSPFCMIN	156
HSIMSPFCNET	156
HSIMSPFDSJNET	157
HSIMSPFFDELIM	157
HSIMSPFFEEDTHRU	157
HSIMSPFHLEVEL	159
HSIMSPFKEEPNODECAP	159
HSIMSPFKS	159
HSIMSPFMSGLEVEL	160
HSIMSPFMULTISUB	160

HSIMSPFNETPIN	161
HSIMSPFNETPINDEL	162
HSIMSPFNETWARNFILTER	163
HSIMSPFNOWARNONCAP	163
HSIMSPFPOS	164
HSIMSPFPOSSEARCH	164
HSIMSPFPOSTL	164
HSIMSPFPRINTPIN	165
HSIMSPFPRINTSUBNODE	166
HSIMSPFRAILRED	166
HSIMSPFRCNET	167
HSIMSPFREPORTNOBA	168
HSIMSPFSKIPNET	168
HSIMSPFSKIPPWNET	169
HSIMSPFSKIPSIGNET	169
HSIMSPFSPLITCC	169
HSIMSPFEFTRIPLET	169
HSIMSPFWARNFILE	170
HSIMSPICE	170
HSIMSPISCIUNITERR	172
HSIMSTEADYCURRENT	172
HSIMSTEP2TAUMAX	173
HSIMSTOPIFDCFAIL	173
HSIMSTOPIFNODC	173
HSIMSTNAME	174
HSIMSTSWAP	175
HSIMSUBBUSDELIM	176
HSIMTAUMAX	176
HSIMTIMEFORMAT	177
HSIMTIMESCALE	177
HSIMTLINEDV	178

Contents

HSIMTOP	179
HSIMTRAPEZOIDAL	179
HSIMUFILIB	180
HSIMUSEHM	180
HSIMUSEHMINST	182
HSIMUSEPREVIOUSDC	184
HSIMUSERLIB	184
HSIMUSEVA	184
HSIMUSEVATABLE	185
HSIMUTFCOMPRESSLEVEL	185
HSIMV2S	186
HSIMV2SD	187
HSIMVABRANCHPART	187
HSIMVACCBMATCH	188
HSIMVACROSSTTOL	188
HSIMVACROSSVTOL	188
HSIMVAPARTITION	189
HSIMVAPRINTVAR	189
HSIMVATBLERANGE	190
HSIMVATABLESIZE	190
HSIMVB\$3END	190
HSIMVB\$3START	190
HSIMVBSEND	190
HSIMVCD2VEC	191
HSIMVDD	191
HSIMVDSEND	192
HSIMVGSEND	192
HSIMVECTORFILE	192
HSIMVERILOGA	193
HSIMVHI, HSIMVHL	193
HSIMVHTH	194

HSIMVHTHRATIO	194
HSIMVLO	194
HSIMVLTH	195
HSIMVLTHRATIO	195
HSIMVPRECISION	196
HSIMVSRCDV	196
HSIMVSRCCLMIN	197
HSIMVSRCRMIN	197
HSIMVSRCSYNC	198
HSIMWARNFILTER	198
HSIMWARNIFNODC	199
HSIMWARNSTOP	199
HSIMWRAPPERSUB	200
HSIMX	200
HSIMXVERSION	201
HSIMZV	201
.PROTECT or .PROT	203
.UNPROTECT or .UNPROT	204
<hr/>	
5. Running HSIM	205
Specifying the Input Netlist	205
HSIM Initialization File	206
Parse Error Limit Setup	206
Invoking HSIM	206
<hr/>	
6. HSIM Library and Data Encryption	211
Encrypting Files	211
<hr/>	
7. HSIM Circuit Simulation Examples	213
Resistor Ladder Test Case	213
Resistor Ladder Simulation Example	215

Contents

SRAM Test Case	219
SRAM Example	220
Reviewing the SRAM Simulation Statistics	222
8. Input Netlist	225
Netlist Syntax Summary	226
Netlist Differences Between HSIM and SPICE	227
SPICE and HSIM Element and Capability Support	228
HSPICE String Parameter Support	230
Encrypted HSPICE® Netlists	231
Device Models	231
Gate Capacitance Model	232
Stress Model	232
Element Statements	233
Resistor	234
Capacitor	235
Self-Inductor	237
Mutual Inductor	238
Lossless Transmission Line	239
Lossy Transmission Line	240
Lossy Transmission Line Model	241
Lossy Transmission Line File	243
Field Solver Model	244
N-Port	247
MOSFET	249
MOSFET Model	251
Diode	254
Diode Model	256
Bipolar Junction Transistor (BJT)	256
Bipolar Junction Transistor (BJT) Model	257
Junction Field Effect Transistor (JFET) and Metal Semiconductor Field Effect Transistor (MESFET)	259
Junction Field Effect Transistor (JFET) Model	260
IF-ELSE Syntax in Netlists	261
IF-ELSE Block Rules	262
IF-ELSE Block Syntax Description	263
IF-ELSE Netlist Examples	265

Driving Sources and Input Stimuli	270
Independent Voltage Source	271
Independent Current Source	272
Pulse Source Function (PULSE)	273
Sinusoidal Source Function (SIN)	273
Single-Frequency Frequency Modulation (SFFM) Source Function	274
Amplitude Modulation (AM) Source Function.	275
Exponential Source Function (EXP).	276
Piecewise Linear Source Function (PWL)	276
Piecewise Linear Source Function with High Impedance State (PWLZ)	277
Voltage-Controlled Current Source (VCCS).	278
Voltage-Controlled Voltage Source (VCVS)	279
Ideal Transformer	281
Current-Controlled Current Source (CCCS)	282
Current-Controlled Voltage Source (CCVS)	284
Voltage-Controlled Resistor (VCR)	286
Voltage-Controlled Capacitor (VCC)	287
Laplace Element	288
Digital Vector File	291
Vector Statements	292
check_window	293
delay tdelay	294
enable.	295
io.	295
logichv or vih logiclv or vil logicxv.	296
mask.	297
period	298
radix	298
resistance out outz	299
signal vname	299
slope rise trise fall tfall	300
stop_at_error	301
tskip	301
triz.	302
tunit.	302
vchk_ignore	303
vhth voh vlth vol	303
vref.	304
vth.	305
Dynamic vhi and vlo for Logic HIGH and Logic LOW States	306
VCD Direct-Read Feature	306

Contents

Using the Signal Information File	306
Defining Bus Syntax with the #format Command	307
Defining Signal Directions	308
Defining Mapping Information with the #alias Command	311
Defining Attributes for Signals	312
VCD File Sample	317
Signal Information File Sample	319
Vector Data	320
Built-in Functions	323
Simulation and Control Statements	324
.alter	325
FF	326
.del param	327
.end	328
.endl	328
.ends	328
.force	329
.global	330
.ic	330
.include	331
.lib	331
.malias	332
.nodeset	333
.op	333
.option	334
.param	337
.release	337
.subckt	338
.temp	339
.tran	339
<hr/>	
9. Simulation Commands	341
Specifying HSIM Commands	341
Specifying HSIM Commands in the Top-Level Netlist	342
Specifying Commands in an External File	345
Aliasing Commands to User-Defined Names	346
Control Commands for Ungrounded Capacitors	346

Control Commands for Grounded Capacitors	347
Control Commands for Floating Capacitors	347
Control Commands for Isomorphic Matching	350
DC Initialization	351
Implicit Backward Euler Algorithm and Trapezoidal Integration Algorithm	353
Simulation Control Commands	353
Simulation Speed Control Summary	354
Piecewise Constant Setting	355
Examples of HSIM Simulation Control Commands	356
HSIMSPEED Example	356
HSIMPORTCR Example	358
First Demonstration	359
Second Demonstration	359
HSIMGCAP, HSIMGCAPR, HSIMFCAP, HSIMFCAPR Example	360
<hr/>	
10. Post-Layout Backannotation	363
Signal Net Post-Layout Control	363
Post-Layout Devices and RC Backannotation	363
Device Parameter Format (DPF)	364
RC Backannotation (DSPF/SPEF)	364
Node Capacitance Backannotation	364
HSIM-Supported DSPF Format for Hierarchically Extracted Feed-Through Nets	365
Top-Level DSPF File	367
Block-Level RC Extraction	368
Block-Level DSPF File	369
Enhanced HSIM Subcircuit Instance Parameters	369
<hr/>	
11. Postlayout Acceleration (PLX), Structured Backannotation (SBA), and SP2DSPF Utility	371
Backannotation Without Postlayout Acceleration	371
Backannotation with Postlayout Acceleration	373
HSIMSPFPLX	374
Structural Backannotation (SBA)	374
Running the Advanced SBA Flow	380
Running the SBA Flow	381

Contents

Specifying the Postlayout Netlist in the SBA Flow	382
Specifying Postlayout Device Models.	383
Defining the Backannotation Scope in the SBA Flow	384
SBA Commands	385
HSIMSPA	385
HSIMSBASF	385
HSIMSBACAP	385
HSIMSBAFRES	385
HSIMSBANTL	386
HSIMSBAPARAM	386
HSIMSBAMSGLEVEL	389
HSIMSBAMSGLIMIT	390
HSIMSBAHIERID	390
HSIMSBASFX	390
HSIMSBAPFX	391
HSIMSBAAUTOSUBMODELS.	391
SP2DSPF Utility.	392
Generating a DSPF File From the Flat Extracted Netlist	392
Running SP2DSPF	392
SP2DSPF Utility Parameters	393
-pre	393
-fpre	393
-pretop	393
-post	393
-fpost	394
-posttop	394
-an	394
-anan	394
-out	395
-dpf	395
-outdpf	395
-dspf	396
-outdspf	396
-pinports	396
-ms	396
-mm	397
-opt outnf	397
-opt outprefc	398
-opt outhierc	398
-opt outsubc	398
-opt serial	398

-opt capnet	398
-opt dupcc	399
-opt rpref	399
-opt ccpref, -opt gcpref	399
-opt vsr	400
References	401
<hr/>	
12. Simulation Output	403
HSIM Output Formats	403
FSDB Output Format	404
Nassda Output Format	405
WDF Output Format	407
Simultaneous Multiple Output Files	407
WSF Output Format	408
.out ASCII Output Format	408
PSF Output Format	408
PSF Float Output Format	409
UTF Output Format	409
rawfile Output Format	409
Output Control Statements	410
PRINT/.PROBE/.PLOT/.GRAPH Statements	410
Optional Settings	413
Print Average and RMS Currents	418
.lprint Statement	419
.STORE/.RESTORE Statement	420
.STORE	420
.RESTORE	421
.measure Statement	422
Rise, Fall, and Delay Time Measurements	423
Derivative Measurements	424
Average, RMS, MIN, MAX, Peak-To-Peak Measurements	425
Find and When Measurements	425
Equation Evaluation	426
Continuous Measurement	426
Jitter and Histogram Report	427
.FOUR Statement	428
.FFT Statement	428

Contents

Print Windows	430
Full-Chip Probing	430
Controlling the Full-Chip Probing Flow	433
13. Conversion Utilities	435
The wdf2tbl Utility	435
The wdf2pwl Utility	438
The hsencrypt Utility	440
The v2s Utility	442
14. AC Small-Signal Analysis	445
Overview of AC Analysis	445
AC Sources	449
AC Analysis Output	450
.alter	450
.print	451
AC Analysis Measurement	452
15. DC Analysis	453
DC Sweep of One or Two Source Value(s)	453
DC Sweep Parameter Value	455
DC Sweep Simulation Temperature	456
Multiple DC and Data Sweeps	456
DC Output and Control Commands	457
Measurement In DC Analysis	458
DC Interactive Mode Debugging	458
DC Interactive Mode Commands	459

Part II: HSIM Advanced Analysis

16. Interactive Mode Debugging	463
Overview of Interactive Mode Debugging	463
DC init. Interactive Mode Debugging	463
DC Interactive Mode Commands	464
Transient Interactive Mode Debugging.....	464
List of Interactive Mode Commands	466
Interactive Mode Commands.....	469
alias	471
ap.....	471
closelog	472
cont	473
dcon.....	473
dcpath	474
de	476
dn.....	477
dp.....	478
eid	479
ename	479
ev	480
exi.....	480
fcc	481
fmeta	482
fv	482
help	483
iap	483
idp	484
iev.....	484
inc	485
inv.....	487
lx.....	488
matche.....	488
matchn.....	489
nc	490
nctr.....	492
ni	494

Contents

nid	494
nm	495
nname	495
nv	495
op	496
openlog	496
pt	497
quit	497
rcf	497
restart	498
rv	498
savesim	499
stop <conditional>	499
stop -at	500
stop -list and stop -delete	500
vni	501
trace_thru_on	501
tree	502
17. Timing and Power Analysis	505
Overview of Timing and Power Analysis	505
Timing Checking	506
Setup Time Check	506
Hold Time Check	510
Pulse Width Check	514
Timing Edge Check	518
Timing Check Windows	522
Bisection Optimization	523
.model	523
.param	524
.tran	524
.alterparam	525
Power Checking	526
DC Path Check	527
Excessive Current Check	530
Excessive Rise/Fall Time Check	532
High Impedance Node Check	535
Power Check Windows	538

Activity Checking	538
Active Node Check	538
Active Files	540
Inactive Files	540
Selective Net Backannotation Flow	541
Example: Selective Net Backannotation in One Phase	542

18. Monte Carlo Analysis	545
Overview of Monte Carlo Analysis	545
Selecting the Monte Carlo Analysis Type	546
Selecting the Distribution Function Parameter Type	546
Gaussian Distribution	546
Uniform Distribution	547
Limit Distribution	547
Specifying Starting Seed	547
Selecting the HSIM Monte Carlo Analysis Mode	548

Part III: HSIM Advanced Modeling

19. Using Verilog-A with HSIM	553
Verilog-A Compiler	554
Verilog-A Options	555
PVA Versus Legacy Verilog-A Compiler	555
Verilog-A Architecture	555
Getting Started with the Verilog-A Compiler	556
Including Verilog-A Modules in HSIM Simulations	556
Including Verilog-A Modules in a SPICE Netlist	558
Including Verilog-A Modules in a SPECTRE Netlist	559
Including Verilog-A Modules in an ELDO Netlist	560
Verilog-A Model Explanation	560
HSIM .log file Information	564
Case Sensitivity Issue In SPICE Netlist	565
Verilog-A Module Name Clash with Subcircuit Names	565
Verilog-A Language and HSIM Implementation-Specific Features	566

Contents

HSIM Implementation-Specific Features	566
Integer and Real Number Variables	567
Natures and Disciplines	567
Kirchoff's Potential Law	567
Kirchoff's Flow Law	567
Nodes	568
Branches	568
Operators	569
Statements	570
Procedural Assignment Statements	571
Branch Contribution Statements	571
Block Statements	572
Case Statements	572
Repeat Loop Statements	573
While Loop Statements	573
For Loop Statements	573
Generate Statements	573
Analog Events	574
Cross Statements	574
Timer Statements	575
Analog Operators	576
ddt Operator	576
idt Operator	576
delay Operator	577
Transition Operator	577
slew Operator	577
bound_step Operator	578
discontinuity Operator	578
\$STOP Operator	578
\$finish Operator	578
Interpolation Function	578
Laplace Transform Filters	579
Analysis	579
I/O and Messages	579
\$strobe, \$monitor, \$display, \$fstroke	580
\$error	580
\$warn	580

Printing Variables in Verilog-A Instances	581
Multiple Inclusions of Module Files and Duplicate Declarations	581
Macro Definitions	581
Hierarchical Instantiation	581
Simulation Speed Limitations	583
Partitioning Verilog-A Modules	583
Isomorphic Matching Verilog-A Modules	583
Verilog-A Interactive Commands	583
ev, iev	583
nc, inc	584
Encrypted Verilog A Modules	584
Limitations and Unsupported Features	584
Legacy Verilog-A Compiler Support	585
Compiler Option	585
Supported Features	586
Supported HSIM-VA CommandS	586
Cross Statements	587
Using Tables for Verilog-A models	587
Printing Variables in Verilog-A Instances	587
Encrypted Verilog A Modules	587
<hr/>	
20. Ferroelectric Capacitor (FeCap) Model	589
Overview of the Ferroelectric Capacitor (FeCap) Model	589
FeCap Elements	590
FeCap Model	591
FeCap for Model Parameter Extraction	593
Measuring Ferroelectric Hysteresis Loops	594
FeCap Model Parameter Extraction	596
FeCap Model Temperature Coefficients	598
<hr/>	
21. User Model Interface (UMI)	599
UMI Models	599
Building a Dynamic Library	600
User Files	600

Contents

Header File UMI.h	601
Interface File UMI.c	606
Header File b3defs.h	608
Primary File for Model b3main.c	609
Model Parameter Processing File b3readmodel.c	614
Model Default Value Setting File b3setmodel.c	615
Geometry Assignment File b3assigngeometry.c	616
Temperature Updating File b3temp.c	617
Model Evaluation and Load File b3load.c	618
Charge-Based and Meyer Capacitance Models.	619
Charge-Based Capacitance Model Example	619
Meyer Capacitance Model Example.	622
<hr/>	
22. Flash Core Cell Model	625
Flash Cell Core Models	625
Floating Gate Modeling	626
Defining and Instantiating a Flash Model.	626
Optional Nwell Terminal	627
Conventions for HSIM Flash Core Cell Models	627
Initializing and Probing Flash Core Cell Models.	628
Flashlevel = 1 Parameters and Operation	628
Program Event.	628
Erase Event	629
Flashlevel = 1 Voltage Threshold Change	629
Flashlevel = 1 Parameter List and Default Values	630
Flashlevel = 1 Single Cell Simulation Example	631
Flashlevel = 2 Parameters and Operation	632
Flashlevel = 2 Program and Erase Events.	632
Program Event	632
Erase Event	633
Flashlevel=2 Threshold Voltage Change	634
Flashlevel=2 Parameter List and Default Values	636
MRAM Core Cell Models	637
Spin-Torque-Transfer (STT) MRAM Core Cell Model (MRES0).	638
MRES0 - STT MRAM Core Cell Definition.	639
MRES0 - Supported Parameter Description	639
MRES0 Instantiation Example	640
MRES0 - STT MRAM Core Cell Functionality	640
Limitations and Assumptions for the MRES0 Core Model	641

Dual-Active Layer (DAL) MRAM (MRES1)	641
MRES1 - DAL MRAM Core Cell Definition	642
MRES1 - Supported Parameter Description	643
MRES1 Instantiation Example	644
MRES1 - DAL MRAM Core Cell Functionality	644
Limitations and Assumptions for the MRES1 Core Model	645
Toggle MRAM Core Cell Model (MRES2)	645
MRES2 - Toggle MRAM Core Cell Definition	646
MRES2 - Supported Parameter Description	647
MRES2 Instantiation Example	647
MRES2 - Toggle MRAM Core Cell Functionality	647
Limitation of MRES Elements	647
Monitoring MRAM Array State Conditions	648
23. C Language Functional Model	649
Model Flow Overview	649
Windows NT/2000/XP Requirements	650
HP-UX Requirements	650
Model Definition	651
Model Creation APIs	652
hmCreateModel	652
hmSetModelAttr	652
Model Port APIs	653
hmDefAnalogPort	654
hmDefAnalogPortBus	654
hmDefDigitalPort	655
hmDefDigitalPortBus	655
hmDefVarAnalogPortBus	656
hmDefVarDigitalPortBus	656
hmDefCurrentPort	657
Internal State APIs	658
hmDefAnalogState	658
hmDefAnalogStateVec	658
hmDefDigitalState	658
hmDefDigitalStateVec	659
Model Interfaces	660
Simulation Interface APIs	660
hmSimStage	660
hmPresentTime	661
hmWakeUpModel	661

Contents

hmQuitSim	661
hmIntrSim	662
Name Interface APIs	662
hmPortName2PortId	662
hmPortId2PortName	662
hmStateName2StateId	663
hmStatId2StateName	663
hmPortName2CktNodeName	663
hmPortId2CktNodeName	663
hmModelInstName	664
hmModelFuncName	665
Analog Port Interface APIs	665
hmAnalogPortValue	665
hmAnalogPortValueById	666
hmAnalogPortBusValue	666
hmAnalogPortBusValueById	667
hmSetAnalogPortValue	667
hmSetAnalogPortValueById	667
hmSetAnalogPortBusValue	667
hmSetAnalogPortBusValueById	668
Digital Port Interface APIs	668
hmDigitalPortValue	669
hmDigitalPortValueById	669
hmSetDigitalPortBusDelay	669
hmSetDigitalPortBusDelayById	669
hmDigitalPortBusValue	670
hmDigitalPortBusValueById	670
hmSetDigitalPortDelay	670
hmSetDigitalPortDelayById	671
hmSetDigitalPortValue	671
hmSetDigitalPortValueById	671
hmSetDigitalPortBusValue	671
hmSetDigitalPortBusValueById	672
Current Port APIs	672
hmCurrentPortValue	672
hmCurrentPortValueById	673
hmSetCurrentPortValue	673
hmSetCurrentPortValueById	673
Port Capacitance APIs	674
hmPortCap	674
hmPortCapById	674
hmPortBusCap	674
hmPortBusCapById	675
hmSetPortCap	675

hmSetPortCapById	675
hmSetPortBusCap	675
hmSetPortBusCapById	676
Enable/Disable Port APIs	677
hmDisablePortDrive	677
hmDisablePortDriveById	677
hmEnablePortDrive	678
hmEnablePortDriveById	678
hmDisablePortBusDrive	678
hmDisablePortBusDriveById	678
hmEnablePortBusDrive	678
hmEnablePortBusDriveById	679
hmDisableAllPorts	679
hmEnableAllPorts	679
Set Port APIs	679
hmSetPortActive	680
hmSetPortActiveByID	680
hmSetPortBusActive	680
hmSetPortBusActiveByID	680
hmSetPortIdle	680
hmSetPortIdleByID	680
hmSetPortBusIdle	681
hmSetPortBusIdleByID	681
Port Delay/Strength APIs	681
hmSetPortDelayRes	682
hmSetPortDelayResByID	682
hmSetPortBusDelayRes	682
hmSetPortBusDelayResByID	683
Port Sensitivity APIs	683
hmSetPortEventDv	683
hmSetPortEventDvById	684
hmSetPortBusEventDv	684
hmSetPortBusEventDvById	685
Port Change APIs	685
hmPortChange, hmPortChangeById	686
hmPortBusChange, hmPortBusChangeById	686
hmAnyPortChange	686
Port Bus Size APIs	687
hmPortBusSize	687
hmPortBusSizeById	687
hmPortDir, hmPortDirById	688
Internal State Interface APIs: Analog State APIs	688
hmAnalogStateValue	689
hmAnalogStateValueById	689

Contents

hmAnalogStateVecValue	689
hmAnalogStateVecValueById	690
hmSetAnalogStateValue	690
hmSetAnalogStateValueById	690
hmSetAnalogStateVecValue	690
hmSetAnalogStateVecValueById	691
Internal State Interface APIs: Digital State APIs	691
hmDigitalStateValue	691
hmDigitalStateValueById, hmDigitalStateVecValue, hmDigitalStateVecValueById	691
hmSetDigitalStateValue	693
hmSetDigitalStateValueById	693
hmSetDigitalStateVecValue	693
hmSetDigitalStateVecValueById	693
Miscellaneous Interface APIs	694
hmModelInstParamValue	694
hmModelInstStrParamValue	695
hmSimOptValue	695
hmFree	695
hmMsg, hmWarn, hmError	696
Modeling Memory Core	696
hmDefMemCore	696
hmInitMemCore	697
hmReadMemCore	698
hmWriteMemCore	698
Examples	698
A/D and D/A Converter Examples	698
Port Delay Examples	702
Port Delay Example - C Functional Model File	703
Port Delay Example - HSIM Netlist File	704
RAM Example	704
Event Handling Example	708
hmEventHandle	708
<hr/> 24. HSIM Three-Dimensional Integrated Circuit (3D-IC) Modularization	711
Overview of 3D-IC Simulation Netlist	711
3D-IC Netlist Construct and Usage	711
Transient Analysis and Alters Simulation Features	712
Full Circuit Example	712

Module-Based HSIM Commands	712
3D-IC Post-Layout Back-Annotation	713
Flat IC Module DPFs with a Separate DPF for Silicon Interposer	714
Full 3D-IC Flat Extracted DPF	715
3D-IC Parasitic (SPF) Back-Annotation	716
Flat IC Module SPFs with a Separate SPF for Silicon Interposer	716
Full 3D-IC Flat Extracted SPF	717

Part IV: HSIM Appendices

A. Device Model Reference	721
Reference Sources for Device Models.....	721
B. Custom Output Interface.....	723
Overview of Output Formats	723
Call Custom Library Mechanism	724
Functional Specification	725
Common Include File	726
Create and Initialize the Waveform File Function.....	728
CreateWaveFile	728
Create a Header Function	730
CreateHeader.....	730
Create a Waveform Into a Waveform File	731
BeginCreateWave	731
CreateWave	732
EndCreateWave.....	734
Create a Duplicate Waveform Into a Waveform File	734
CreateDuplWave	735
Add Waveform Values Into a Waveform File	736
AddNextDigitalValueChange	736
AddNextAnalogValueChange	737
End the Waveform File Process.....	738
CloseWaveFile.....	738

Contents

Building a Waveform File Generator Library	739
<hr/>	
C. Waveform Viewer Customization	745
New	746
Modify, Delete	747
OK	748
.waveViewerTable	748
Rule 1.....	749
Rule 2.....	749
Rule 3.....	749
Rule 4.....	749
<hr/>	
D. HSIM/Eldo Compatibility	751
Starting HSIM in the Eldo Mode	751
Transistor Model Compatibility.....	752
HSIM-Supported Eldo-Specific Syntax	752
Eldo Syntax Not Supported by HSIM.....	753
Eldo Commands Partially or Fully Supported by HSIM	754
HSIM-Supported Eldo Options	756
Eldo Commands Not Supported in HSIM	757
HSIM Supported Eldo Macromodels	760
<hr/>	
E. HSIM-ADMS Integration.....	761
.....	761
<hr/>	
F. HSIM-Virtuoso Analog Design Environment Interface	763
HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support	763
HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support	763
All-In-One Package (AAIM)	764
AAIM Installation & Setup.....	764
AAIM Uninstallation	766
Native Netlist Integration (AANNI)	766

Native Netlist Integration Installation & Setup	766
Native Netlist Integration Features	766
Basic Native Netlist Integration Flow	767
Porting the Existing Design	768
Native Netlist Integration Window and Pull-Down Menus	768
Environment Setup	771
Setup Parameters	774
Netlisting	776
Regenerate the Netlist and Run HSIM	780
Toggle between Spectre and HSIM Simulation Database for Waveform Probing	780
CircuitCheck in the HSIM-Virtuoso Interface Environment	783
View Log File	783
View Output ASCII Files	784
Check in Synopsys License	786
Cadence Cross-probing	787
WaveView Analyzer Cross-probing	791
CoSim (AACoSim) Integration	793
UNIX Setup	793
CoSim Installation	793
Basic CoSim Flow	794
HSIM-Virtuoso CircuitCheck Integration	794
Native Netlist CircuitCheck	794
WaveView Analyzer Integration	805

G. HSIM-Virtuoso Interface Advanced Topics	807
Generating hsim/hsimD View and SimInfo	807
Modifying hsim/hsimD SimInfo	808
Removing hsim/hsimD SimInfo	813
Netlist Procedures for Component Primitives	813
HSIMD Netlist Procedures for Component Primitives	816
instParameters Field	816
componentName Field	817
termOrder Field	817
propMapping Field	817
namePrefix Field	817
namePrefix Field	818
Models, Macros, and Include Files	819
Models	819
hsimD (Direct)	820

Contents

Macros	820
Include File	821
Net Name Conversion Macro	821
Expansion of pPar, iPar	822
Assigning HSIM Parameters	822
Assigning HSIM Parameters for Subcircuit	823
Assigning an HSIM Instance Parameter	824
Assigning an HSIM Parameter to an Instance with a Subcircuit (Cell) Assigned the Same HSIM Parameter in Component Description Format	826
Naming Conventions	827
HSIM-Virtuoso Interface Ocean Script Command Usage	830
Public APIs for “HSIMOcean” package	831
Ocean Script Example	832
Socket (HSIM) and Direct (HSIMD) Integration	834
Installing Socket (HSIM) and Direct (HSIMD) Interfaces	834
Porting Existing Design	836
Netlist and Simulation	837
Starting the GUI and Selecting HSIM	837
Specifying a Host Machine	838
Setup Environment	838
Netlister Settings	840
Graphically Editing Stimulus Files	841
Analysis	843
Design Variables	844
Simulator Options	845
HSIM Parameters	847
Selecting Data to Save or Plot	849
Timing and Power Checks	849
Generating Netlists	852
Running Simulations	853
Viewing Results	854
Waveforms	854
Annotation	855
Load and Save Sessions	855
Monte Carlo Analysis	856
<hr/>	
H. HSIM-Virtuoso Interface Netlist Properties	861
HSIM Netlist Properties	861
HSIMD Netlist Properties	875

I. HSIM CCK	891
Overview of CCK Option	893
CCK Tutorial	893
Conventions	894
Using CCK Commands	894
Specifying Circuit Checks in Command Files	899
Running Circuit Check Operations	900
cckNoSimu	900
Passing Parameters Into CCK Commands	902
CircuitCheck.cck Command File	903
Include Statements	903
Running Parametric Checks	904
Checking Electrical Parameters	904
cckParam	904
Capacitor Values	906
MOSFET Width	906
MOSFET Length	907
MOSFET Drain/Source Area and Drain/Source Perimeter	907
MOSFET Gate Oxide Thickness	908
Diode Width, Length, and Area	908
Simulation Run Temperature	909
Model	909
Limiting the Number of Violations Reported	912
M-factor	913
Specifying Resistor Boundaries	913
Running Static Analysis for Devices	914
cckDevParam	914
Running Post-layout RC Checking	915
cckParasiticRC	916
Checking Design and Electrical Rules	917
Static Device Voltage Analysis	917
Running Device Voltage Analysis for Transistors	918
cckMosV	918
Running Device Voltage Analysis for BJT Devices	931
cckBjtV	931
Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices	932
cckCapV	933

Contents

cckDioV	936
cckResV	939
Running Subcircuit-Based Voltage Analysis Using the Static Approach	942
cckSubV	942
Running Diode Forward Bias Analysis	945
cckDiode	946
Running Element Current Analysis	948
cckElemI	948
Running a Reference Check for Instances and Subcircuits	950
cckMatchSub	950
Detecting Excessive Current Path	951
cckExiPath	951
Checking for Floating Gates and Analyzing Current Sources	952
cckFloatGateSrc	953
Checking NMOS Bulk Connections	956
cckNmosB_gt_DS	956
Finding Potentially Conducting NMOS Devices	960
cckNmosG_gt_DS	960
Checking NMOS Node to VDD Connection	965
cckNmosNodeToVDD	965
Checking Node Voltages	967
cckNodeVoltage	967
Checking Paths to Voltage Sources	969
cckPathToVsrc	969
Checking PMOS Bulk Connections	973
cckPmosB_lt_DS	973
Finding Potentially Conducting PMOS Devices	977
cckPmosG_lt_DS	977
Checking PMOS Node To GND Connection	984
cckPmosNodeToGnd	984
Running a Safe Operating Area Check	985
cckSOA	985
cckPost	992
Evaluating Signal Degradation or Correct Biasing	993
cckDynSubV	993

Checking Substrate Forward Bias	996
cckSubstrate	996
Checking for Unprotected Antenna Nodes	999
cckAntGate.	999
Checking Static Voltage Propagation Sharing	1000
Propagation Parameters	1001
Propagation Sharing	1001
Example	1001
Running Digital Logic and Memory Diagnostics	1003
cckFlashcore	1003
cckLatchUnInit	1004
cckLatchInElem	1005
cckLatchSkipElem	1006
Checking Stack-up Transistors.	1007
cckMaxStackUpNmos	1007
cckMaxStackUpPmos	1008
Checking and Classifying the Stuck Nodes	1009
cckMaxStuckAt.	1009
cckToggleCount	1010
cckConnReport	1011
Running Interactive Circuit Debugging Command for a Tracking Circuit	1017
Finding a Node First State Change After a Specified Time	1018
ntrig	1018
Running Timing Checks.	1023
cckMaxNmosToVdd	1023
cckMaxPmosToGnd	1024
cckMeasPathDelay.	1025
cckNodeMaxRF	1027
Running Static RC Delay Analysis – Estimate Slew Rate	1029
cckRCDDlyPath	1029
cckDlyAtNode.	1032
cckDlySkipElem	1033
cckDlySkipNode.	1033
cckLimitRisePmosFallNmos.	1034
cckRCFallDelay	1034
cckRCRiseDelay	1034

Contents

cckSetMosDir	1035
Running a Dynamic Device Voltage Check	1041
tcheck mosv	1041
tcheck diodev	1047
tcheck capv	1048
Running a Post-process Device Voltage Check	1050
Method 1	1050
Method 2	1051
Running Signal Integrity Checks	1051
cckDXtalk	1052
Signal Edge Characteristics	1054
Thresholds	1054
Rising Slope	1054
Falling Slope	1054
Validation	1054
Crosstalk Checking Methods	1055
Method 1	1055
Method 2	1056
Static Crosstalk Noise Analysis: Estimating Noise Glitches	1058
cckXtalk	1059
Detecting Leakage Current	1066
cckMaxStaticLeak	1066
cckOffLeakI	1067
Detecting Power-Down Floating Gates	1073
cckAnalogPDown	1073
cckAnalogPDownlth	1075
Running Static Analysis	1076
cckStaticHzNode	1078
Checking for a Static DC Path	1080
cckStaticDCPath	1080
CCK Utilities	1082
cckBasic	1082
cckCompareOp	1083
cckMatchSub	1085
cckPatternMatch	1086
cckPatternConstraint	1087

cckTgPair	1089
Running the CCK Tutorial	1091
Running CCK	1091
Test Case	1092
Test Case Example for tcheck mosv	1095
Index	1097

Feedback

Contents

About This Manual

This manual describes how to use the Synopsys HSIM® simulation tool. The following sections provide a guide to this manual, as well as to other documentation that accompanies this tool.

Related Products and Trademarks

This manual refers to the following products:

Synopsys HSIM®
Synopsys HSPICE®
Synopsys NanoSim®
Synopsys SolvNet® support site

Related Publications

For additional information about the HSIM® simulation tool, see:

- The HSIM® Release Notes, available on SolvNet (see [Accessing SolvNet](#)).
- Documentation on the Web, which provides HTML and PDF documents and is available on SolvNet (see [Accessing SolvNet](#)).

You might also want to refer to the documentation for the following related Synopsys products:

- HSPICE®
- NanoSim®

Conventions

The following conventions are used in Synopsys documentation.

Convention	Description
Courier	Indicates command syntax.
<i>Italic</i>	Indicates a user-defined value, such as <i>object_name</i> .
Purple	<ul style="list-style-type: none">■ Within an example, indicates information of special interest.■ Within a command-syntax section, indicates a default value, such as <code>:include_enclosing = true false</code>
Bold	<ul style="list-style-type: none">■ Within syntax and examples, indicates user input—text you type verbatim.■ Indicates a graphical user interface (GUI) element that has an action associated with it.
[]	Denotes optional parameters, such as: <code>write_file [-f filename]</code>
...	Indicates that parameters can be repeated as many times as necessary: <code>pin1 pin2 ... pinN</code>
	Indicates a choice among alternatives, such as <code>low medium high</code>
\	Indicates a continuation of a command line.
/	Indicates levels of directory structure.
Edit > Copy	Indicates a path to a menu command, such as opening the Edit menu and choosing Copy.
Ctrl+C	Indicates a keyboard combination, such as holding down the Ctrl key and pressing the C key.

Customer Support

Customer support is available through SolvNet online customer support and through contacting the Synopsys support center.

Accessing SolvNet

SolvNet includes an electronic knowledge base of technical articles and answers to frequently asked questions about Synopsys tools. SolvNet also gives you access to a wide range of Synopsys online services, which include downloading software, viewing documentation, and entering a call to the Synopsys support center.

To access SolvNet:

1. Go to the SolvNet Web page at <https://solvnet.synopsys.com>.
2. If prompted, enter your user name and password. (If you do not have a Synopsys user name and password, follow the instructions to register with SolvNet.)

If you need help using SolvNet, click Help on the SolvNet menu bar.

Contacting Synopsys Support

If you have problems, questions, or suggestions, you can contact Synopsys support in the following ways:

- Go to the Synopsys [Global Support Centers](#) site on www.synopsys.com. There you can find e-mail addresses and telephone numbers for Synopsys support centers throughout the world.
- Go to either the Synopsys SolvNet site or the Synopsys Global Support Centers site and [open a case online](#) (Synopsys user name and password required).

Feedback

About This Manual

Customer Support

Part 1: HSIM Core Basics

Introduction HSIM

Describes tool features, types of analysis, and targeted applications.

- HSIM Features
- Interactive Circuit Analysis
- HSIM Applications
- Input/Output Data

HSIM Features

The HSIM simulator is the core of the HSIM platform. HSIM performs transient analysis, DC analysis, AC analysis, and Monte Carlo analysis supporting the following circuit elements:

- MOSFET (Metal-Oxide Semiconductor Field-Effect Transistors)
- Bipolar transistorsHSIM
- Diodes
- Junction field-effect transistors
- Resistors
- Capacitors
- Self and mutual inductors
- Independent voltage and current sources
- Linear and nonlinear controlled voltage and current sources
- Lossless and lossy transmission lines

Chapter 1: Introduction HSIM

Interactive Circuit Analysis

Note: Some special elements are not currently supported in AC analysis. Refer to [Chapter 14, AC Small-Signal Analysis](#) for details.

HSIM expands and builds upon the production-proven HSIM simulator to address the most critical problems associated with the physical effects of interconnect wiring and short-channel effects in nanometer IC designs. HSIM is a complete transistor-level simulation and analysis platform for the design and verification of nanometer integrated circuits.

The HSIM structure is designed to maximize the designer's ability to simulate designs of various sizes and complexities with a cohesive set of HSIM tools. The HSIM core platform and suite consists of a collection of features.

Interactive Circuit Analysis

HSIM interactive circuit analysis provides a circuit debugging environment that interrupts simulation and performs interactive circuit diagnosis at selected points in time. HSIM analysis provides circuit information, such as:

- Node voltage
- Node capacitance
- Element current
- Element conductance and capacitance
- Fan-in and fan-out elements to a node
- Element terminal nodes
- Active element drivers to a node
- Active loading elements to a node
- Excessive current checks
- DC path between two nodes

HSIM Applications

HSIM provides analysis for many applications such as:

- Full-chip transistor-level functionality verification at pre-layout and post-layout stages
- High-speed circuit simulation for memory circuits, including:
 - DRAM
 - SRAM
 - ROM
 - EPROM
 - EEPROM
 - Flash memory
- Timing and power characterization for memory circuits with post-layout parasitics. Microprocessor, DSP, MPEG, and other large IP cores can also be characterized provided a reasonable number of user-selected input stimuli are made available.
- Cross-talk noise simulation
- High-speed analog and mixed-signal circuit simulation
- Functionality, timing, and power analysis report
- Full-chip post-layout simulation including all layout parasitics to determine the following:
 - Circuit performance under the influence of power net IR drop
 - Effect of coupling capacitance on delay, delay noise, functionality, or glitch power.

Input/Output Data

Simulation results stemming from the analysis allow probing designs to obtain the following:

- Nodal analog voltage waveforms
- Nodal digital logic-state waveforms
- Element branch current waveforms through a transistor, a resistor, a capacitor, an inductor or an independent voltage source

These output data can be displayed with either of the following waveform viewers:

- nWave
- SimWave
- WaveView

Check commands are used to provide detailed timing and power measurements. Refer to [Chapter 17, Timing and Power Analysis](#) for detailed information about check commands.

HSIM input data format is generally compatible with the input format of industrial standard circuit simulators such as HSPICE.

Hierarchical Simulation Technology

HSIM high capacity is provided by the innovative use of hierarchical technology—simulation that stores the circuit netlist hierarchically in memory instead of flattening the netlist.

This minimizes memory requirements for identical cells and subcircuits. HSIM isomorphic matching techniques eliminates redundant computation for identical subcircuits and provides greatly enhanced throughput.

To further improve throughput for post-layout simulation, HSIM options include parasitic reduction for both signal and power nets, as well as hierarchical back-annotation. These combined capabilities give HSIM the ability to accurately and efficiently simulate circuits of tens to hundreds of millions of transistors.

HSIM simulates and analyzes:

- Large circuit blocks
 - Groups of large interacting circuit blocks
 - Full-chip designs
-

Limitations and Recommendations

It is recommended that HSIM be employed to investigate and analyze nanometer effects prevalent among high-speed nanometer circuits, such as:

- The influence of power net IR drop on circuit performance
- Cross-coupling
- Full-chip post-layout simulation with layout parasitics

Many of these problems may not be observable while running independent block-level simulations and will require full-chip circuit simulation for detection and correction.

Caution: HSIM is not recommended for circuits containing fewer than 100 transistors. HSIM is more effective for simulating VLSI circuits.

In addition, HSIM should not be used as an exhaustive simulation tool to verify large designs with a huge number of test vectors. HSIM is not designed to handle such applications. It is suggested that test vectors be intelligently selected for key functionality and critical circuit behavior to be tested without using exhaustive vector sets. If it is necessary that large sets of test vectors are to be simulated, dedicate a powerful computer to execute HSIM for the required time.

Feedback

Chapter 1: Introduction HSIM

Input/Output Data

Simulation Flow

Describes the HSIM simulation flow including parsing a SPICE-compatible ASCII input netlist, optional parasitic reduction, building and optimizing the hierarchical simulation database, DC initialization, transient simulation, and outputting the results in ASCII or a binary format. Additionally, a Quick Start section provides the tools to aid in determining which commands to use for optimum performance and precision trade-offs by controlling some of the HSIM algorithms.

- [Simulation Flow Overview](#)
- [Supported Output Formats](#)
- [HSIM Quick Start](#)
- [Using HSIM in a Nanometer VLSI Design Flow](#)

Simulation Flow Overview

The HSIM simulation flow begins by parsing a SPICE-compatible ASCII input netlist as described in [Chapter 8, Input Netlist](#). Optional parasitic RC reduction can be executed on a netlist that contains significant parasitic data. The RC reduction operation reduces each interconnect network into a significantly smaller network with equivalent timing behavior, which increases the efficiency of the HSIM simulation.

Layout parasitics can be back-annotated through a Detailed Standard Parasitic Format (DSPF) or Standard Parasitic Exchange Format (SPEF) route in which one or more files containing the extracted interconnect parasitic RC data are parsed, reduced under your control, and applied to the circuit being simulated. Both signal and power net DSPF/SPEF files pass through similar, but unique, backannotation sub-flows.

Chapter 2: Simulation Flow

Simulation Flow Overview

The device parameter format backannotation capability supports this flow. In this sub-flow, extracted device geometries (W, L, AS, AD, PS, PD, etc.) are taken from the LPE or ideal netlist, and back-annotated to the active devices contained in the original pre-layout schematic netlist. Refer to [Chapter 10, Post-Layout Backannotation](#).

After the netlist is parsed in, HSIM builds and optimizes the hierarchical simulation database. In most cases, memory usage peaks at this point. After the simulation database has been built, storage requirements typically decrease prior to transient simulation.

DC initialization begins after the hierarchical simulation database is constructed. This initialization process is completed when steady state convergence is achieved or when the iteration count reaches the user-defined limit.

The transient simulation follows DC initialization, and it proceeds until the final simulation time is reached. The default settings for simulation control are provided by HSIM. Simulation results are available in either an ASCII or a binary format. The default binary output format is FSDB, which is readily viewable by the nWave waveform viewer.

An overview of the HSIM Simulation Flow is shown in [Figure 1 on page 11](#).

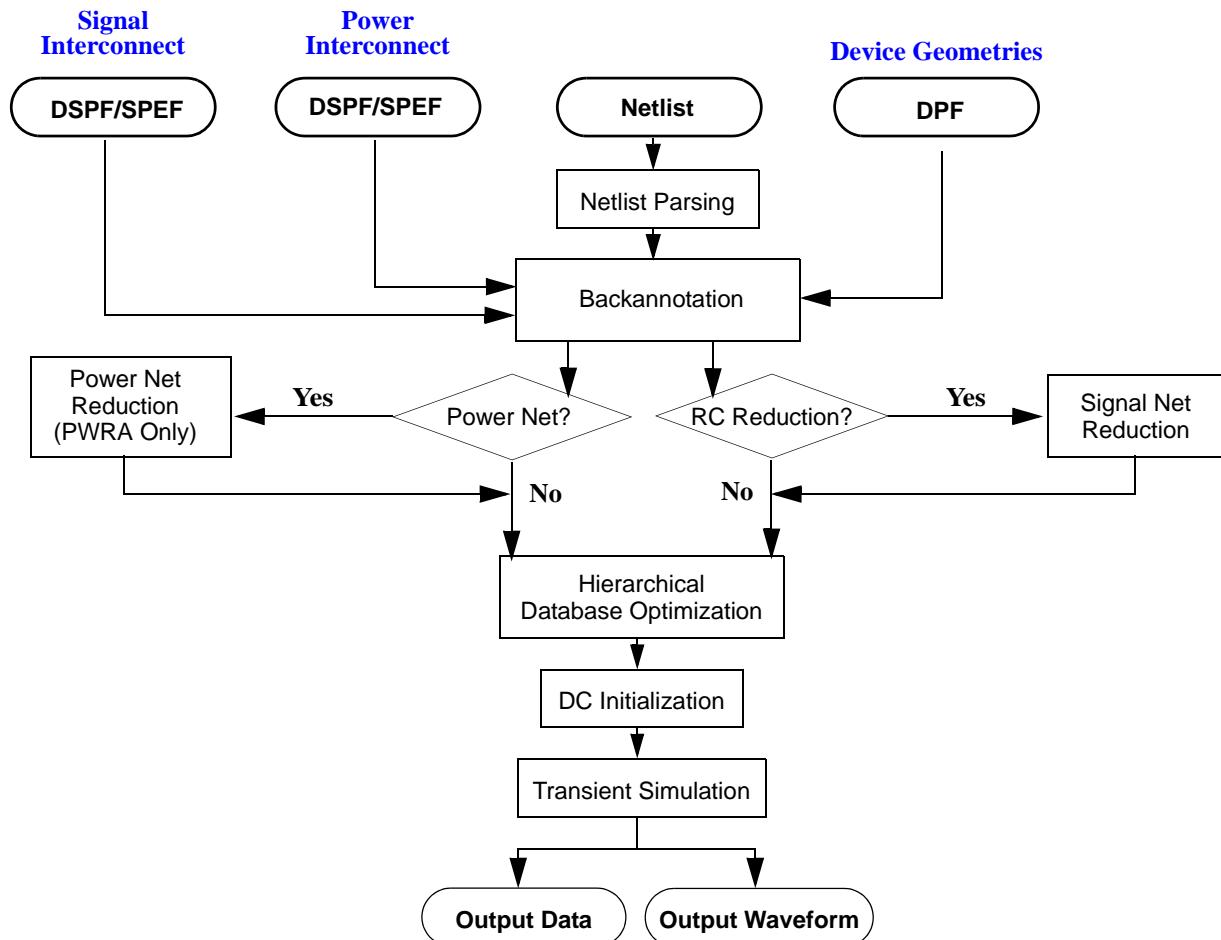


Figure 1 HSIM Simulation Flow

Supported Output Formats

HSIM also supports two ASCII formats:

- Plain ASCII
- HSIM Output Format

Note: For more information, see [HSIM Output Formats on page 403](#) or contact Synopsys for information on other available output formats.

Chapter 2: Simulation Flow

HSIM Quick Start

HSIM format provides a single output format that can be translated into other formats as required by various waveform display tools. The hs2tbl utility included with the software can convert this output format into tabular data that can be displayed by either of the following:

- Xgraph in an X windows environment
 - Dplot in a Windows NT/Windows 2000 environment
-

HSIM Quick Start

The HSIM Quick Start provides a means to quickly and easily get a design running in HSIM by applying a set of appropriate values for the simulation. [Figure 2 on page 14](#) describes how to apply the correct commands and values to achieve the performance/precision trade-offs for analog, mixed-signal, memory, digital, and post-layout designs.

The HSIM simulation setup goals are:

1. Obtain functionality
2. Coarse optimization of timing, voltage, and current
3. Fine optimization of timing, voltage, and current

To determine successful simulation criteria, perform the following steps:

1. From the following, find the type of design that best describes the project:
 - Analog
 - Mixed Signal
 - Memory
 - Digital

If the design also includes post-layout data, the options identified by the Post Layout indicator should be appended. The HSIM commands shown in the flow chart are based on engineering results, where more than one value is specified for a command. You should start with the *first* value indicated for performance or the *last* value indicated for precision. If more than two values are provided in-between, it is a trade off between performance and precision.

Note: These commands are available during setup to both global and subcircuit-based implementations when the algorithm has localization applicability. If a design includes more than one type of design style, use .subckt to apply the appropriate commands to the respective design styled subcircuits. Refer to [.subckt on page 338](#).

2. If your simulation is not functional or optimized using the initial settings, see the settings in [Figure 2 on page 14](#) with dotted lines pointing to them. Adjust the values for the commands to achieve the desired performance/precision required for your design.

Note: The command boxes in the flow chart in [Figure 2 on page 14](#) shows the options available to the commands. Command values indicate the default in *bold italic* and read from *left to right* increases precision. For ease of reading, the HSIM prefix has been omitted, thus the HSIM prefix must be applied to all commands identified in the flow chart. For example, PARPRECISION must be inserted into the simulation deck as HSIMPARPRECISION.

In [Figure 2 on page 14](#), the MOSPRECISION default is 0; 0 should provide the highest performance and the available values to set are 0,1, 2, 3, 4, and 5, with 5 providing the highest precision.

In a mixed signal design with no post-layout data, for example, insert the following parameters into the simulation deck to obtain functionally correct results.

HSIMANALOG=2

HSIMANALOG=2 provides Mixed Signal recommendations modified by the Analog functional decision box to obtain a desired functionality.

HSIMENHANCEDC=1

HSIMENHANCEDC=1 provides Mixed Signal recommendations modified by the DC Flow functional decision box to obtain a desired functionality.

HSIMSPEED=3

HSIMSPEED=3 provides Mixed Signal recommendations.

Feedback

Chapter 2: Simulation Flow

HSIM Quick Start

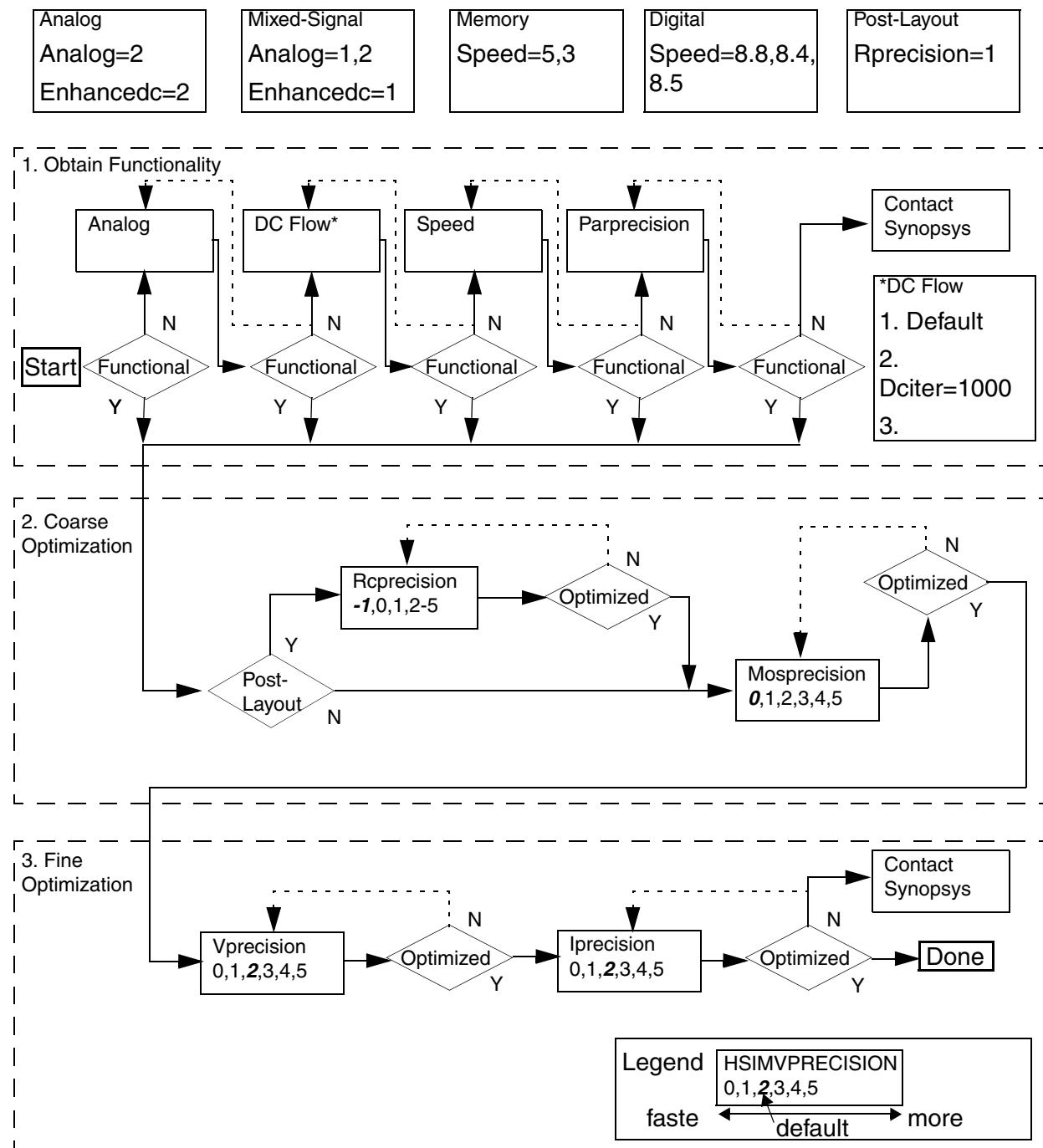


Figure 2 HSIM Performance/Precision Settings

Power Supply Settings

Voltages are influenced by the supply voltage under the control of [HSIMVDD on page 191](#). In cases when HSIM cannot easily find DC convergence, set [HSIMAUTOVDD on page 64](#) = 1. These cases include situations in which there are power nets in the netlist, and when there are multiple power supplies used for different parts of the transistors.

Using HSIM in a Nanometer VLSI Design Flow

A typical nanometer very large system integration (VLSI) design effort can be divided into pre-layout and post-layout design flows. These design flows are discussed in the following sections:

- [Pre-Layout Design Flow](#)
 - [Synthesizable Logic](#)
 - [Non-Synthesizable Logic](#)
 - [Analog/Memory](#)
 - [Post-Layout Design Flow](#)
-

Pre-Layout Design Flow

Pre-layout design flow is designed to create functionally correct designs for layout implementation. Estimated parasitics for the inter-block interconnects are often used for exploring the timing behavior at this early design stage.

At the block level illustrated in [Figure 3 on page 16](#), HSIM shows the use of three design types in a pre-layout flow. A full-chip functionality and timing simulation flow is an extension from the block-level flow. By assembling the netlists of all the individually verified blocks and providing the top-level input stimuli, the full-chip can be simulated without difficulties or complications.

Chapter 2: Simulation Flow

Using HSIM in a Nanometer VLSI Design Flow

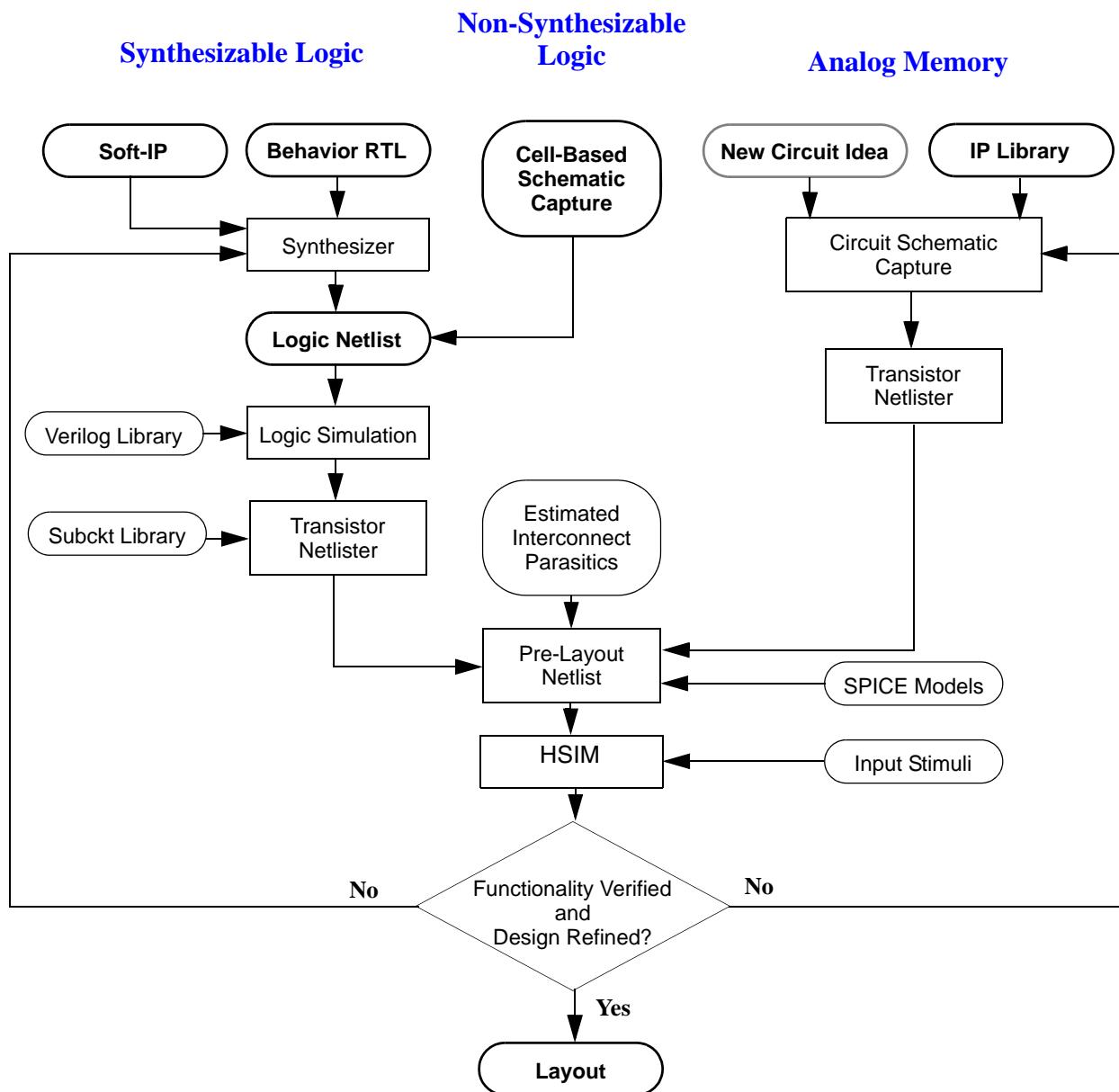


Figure 3 Pre-Layout Design Flow

HSIM provides design flows for the following three design styles:

- [Synthesizable Logic](#)
 - [Non-Synthesizable Logic](#)
 - [Analog/Memory](#)
-

Synthesizable Logic

Synthesizable logic uses high-speed standard cells for digital design. Simulations can be run to explore cross-talk issues along certain critical paths. Then, the information from this step is used to refine layout strategies that prevent cross-talk problems. Block current obtained through the simulation can be used for power bus sizing.

To maximize HSIM capabilities in the synthesizable logic flow, the subcircuit library and the estimated interconnect parasitics must be provided. The estimated parasitics are placed inside the circuit netlist with the other functional elements of the design, and then simulated by HSIM.

The subcircuit library transforms the design from a gate-level representation to a transistor-level representation. The estimated parasitics inject an early physical dimension into a pure logical design in order to study the potential physical effects within the design. Estimated parasitics may include any of the following:

- Long interconnects for a timing behavior assessment, particularly for clock net analysis (jitter and skew).
- Estimated cross-coupling capacitors for a cross-talk assessment
- All possible wire capacitors for a block current assessment

Note: Estimated parasitics are not required for the functional verification of a given circuit.

Non-Synthesizable Logic

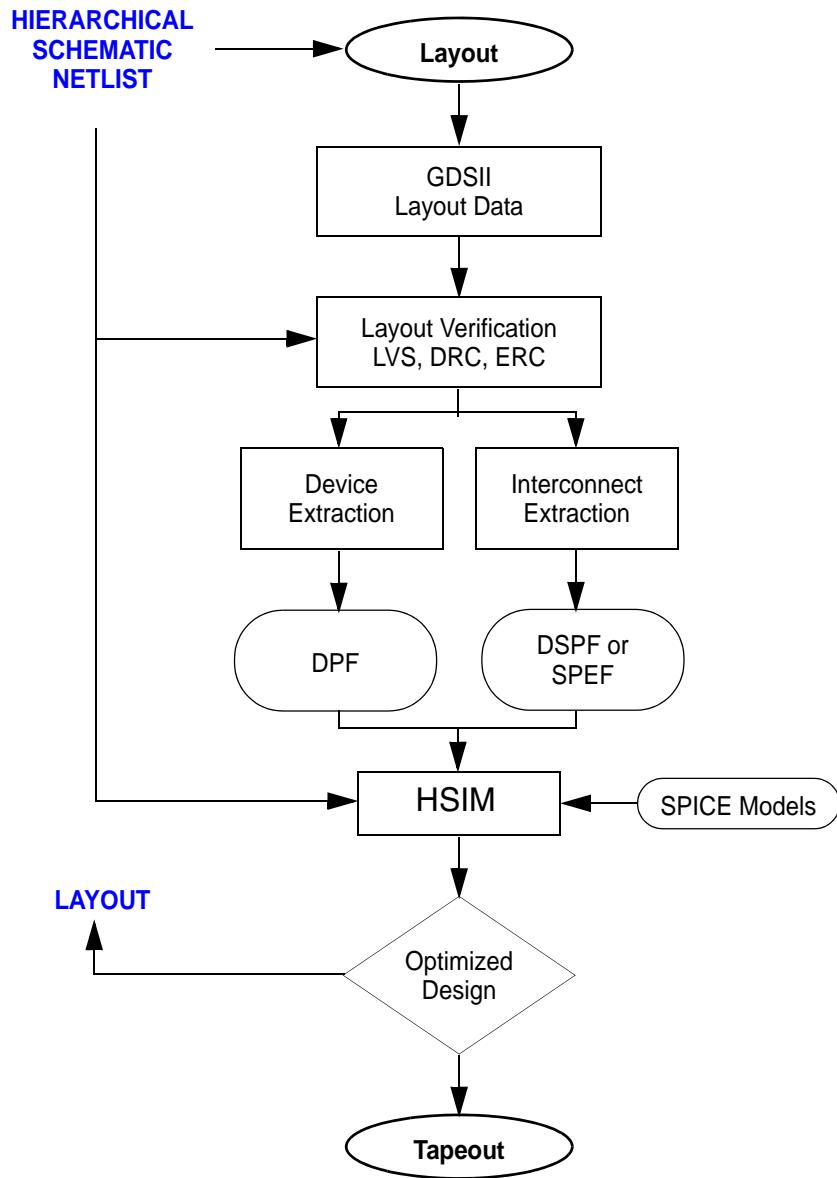
Non-synthesizable logic is often used for high performance Application-Specific Integrated Circuit (ASIC) and semi-custom designs. These types of designs typically utilize complex design techniques such as dynamic logic, and are subject to the circuit effects of noise and ground bounce. As logic simulators do not provide an accurate description of the behavior of these circuits, HSIM simulation is strongly recommended.

Analog/Memory

For analog/memory design styles, HSIM provides speed and accuracy to successfully simulate these circuit types.

Post-Layout Design Flow

The main purpose of the post-layout design flow is to optimize the design by layout refinements and to verify circuit performance in the presence of layout parasitics. In particular, to determine the influence of IR drop and coupling capacitance on design characteristics. An overview of the post-layout design flow is shown in [Figure 4 on page 19](#).



This flow is the recommend post-layout flow. HSIM also supports simulating an ideal netlist and other post-layout flows.

Figure 4 Post-Layout Design Flow

Chapter 2: Simulation Flow

Using HSIM in a Nanometer VLSI Design Flow

The post-layout flow begins with a GDSII layout database. The next stage is physical layout verification to ensure there are no major flaws in the layout, such as:

- DRC
- ERC
- LVS

After physical layout verification, the basic functionality of the layout is verified. This is best accomplished by doing the following:

- Extracting the active components of the design
- Back-annotating DPF to the pre-layout hierarchical netlist

Once this is completed, a top-level full-chip simulation can be performed. Top-level stimuli from the pre-layout flow can be used to ensure that the basic functionality of the extracted netlist has not changed.

Once design functionality has been confirmed, layout parasitics can be incorporated into the simulator and the results of the simulation can be used to optimize the design. Optimization goals include:

- Enhancing circuit speed
- Maintaining power consumption within specification
- Increasing the design margin
- Improving circuit reliability and robustness
- Decreasing sensitivity to manufacturing variability

The circuit characteristics to address in this phase include:

- Speed
- Power
- Reliability
- Manufacturability

Circuit Extraction and Analysis

Enhanced design margins provides greater tolerance against manufacturing fluctuations. Greater circuit reliability ensures a long product life in the field. To refine a layout, a detailed circuit must be extracted and analyzed to determine if and where the circuit needs improvement.

A detailed extracted circuit netlist consists of the following:

- Active circuit devices
- The device's topological connectivity
- Interconnect parasitics for both signal and power nets

Many extraction tools are configured to extract both active devices and interconnect parasitics into a single, flat netlist file. This straightforward approach eliminates any potential problems with issues such as name matching between netlist and DSPF/SPEF files. The one provision is that HSIM must have sufficient memory available to read-in the entire flat netlist.

After the design netlist is parsed, HSIM partitions the design, looking for hierarchy with the available partitioned subcircuits. This ensures that HSIM utilizes the hierarchy in the design for memory efficiency even if the input netlist is flat. Since HSIM peak memory usage occurs while parsing the netlist and building the hierarchical simulation structures, if the circuit netlist can be parsed, simulation can usually proceed. This process works well for circuits of small to medium size but full advantage of the hierarchical simulation engine cannot be realized in the same way that it can with a hierarchical circuit of many levels, containing a large number of related small isomorphic instances.

Most extraction tools also offer separate netlisting for active devices and interconnect parasitics after extraction is complete. HSIM is designed and implemented to take maximum advantage of this flow and data. Extracting the active devices to the Device Parameter Format (DPF) and interconnect parasitic to DSPF/SPEF provides for efficient backannotation into the hierarchical simulation database created after parsing the hierarchical pre-layout netlist.

Interconnect Segmentation Resolution

Another issue is the resolution of the interconnect segmentation. When a fine interconnect segmentation resolution is requested, a flat extractor can create a very large number of extracted parasitic elements. Although HSIM uses efficient hierarchical data structures, it is not completely immune from the large data volume problems associated with parsing in these large flat netlists. The amount of extracted data must be controlled by selecting the appropriate segmentation resolution in the parasitic extraction process.

Note: For a large parasitic database, it is recommended to enable RC reduction features in HSIM.

Chapter 2: Simulation Flow

Using HSIM in a Nanometer VLSI Design Flow

This substantially reduces the memory required to store the interconnect parasitics and ensures that the transient simulation proceeds with maximum efficiency.

A memory circuit contains the same core cell that composes the majority of the layout. If extracted hierarchically, the netlist size can be reduced to one-fifth the size of the same netlist from a flat extraction. System-on-Chips (SoC) also have memory and other regular structures, although the storage saving might not be as great as that of memory circuits.

If the extracted interconnect parasitics can be generated in DSPF/SPEF, then the DSPF backannotation capability allows a net-by-net RC reduction and instantiation. This feature removes the need to store all layout parasitics because each net is reduced and back-annotated one net at a time. This approach effectively eliminates the typical bottleneck associated with handling large flat parasitic netlists and provides greater circuit simulation capacity.

HSIM share signal net reduction technologies and the parameter settings operate identically in each simulator. For signal nets, HSIM offers hierarchical backannotation capability where the reduced signal net can be partitioned and distributed into the pre-layout netlist; preserving the benefits of hierarchical simulation as much as possible. Compared to HSIM flattening backannotation, hierarchical backannotation offers the following benefits for signal nets:

- Small, but not insignificant, throughput improvement.
- Large increase in memory efficiency.

HSIM power net capabilities offer a significant enhancement over HSIM. HSIM offers small power net reduction technologies such as RMIN elimination. However HSIM flattening backannotation of the globally coupled power nets defeats the hierarchical simulation engine and forces flat simulation.

HSIM adds two key pieces of innovative technology that enable efficient full-chip post-layout simulation including all power nets, they are:

- Power Net reduction
- Hierarchical power net backannotation

Power Net reduction has two sub-flows that are applied in the following sequence:

1. Straight-forward timing-based sub-flow largely preserves the power net topology and applies resistor reduction based on a specified threshold, series merging, and parallel elimination to reduce the number of power net resistors by an order of magnitude.

2. The second sub-flow alters the power net topology and applies user controlled heuristics of varying aggression levels that transform the power net structures into significantly smaller networks.

These reduced power nets can be saved at intermediate steps in the reduction process facilitating quick turnaround times for repeated simulations. The final reduced power networks are then back-annotated to the hierarchical pre-layout database using HSIM hierarchical backannotation capability. Circuits that most lend themselves to a hierarchical simulator solution typically have, but are not exclusively limited to, small isomorphic instances that are greatly replicated, post-layout, with all signal and power net interconnect parasitics incorporated into the simulator.

HSIM can precisely verify circuits of tens or even hundreds of millions of transistors at tape out providing the following benefits:

- Increased confidence in achieving first time simulation success.
- Increased probability of ramping the first design to high volume.
- High and consistent yields.

Design Optimization

Design optimization is accomplished through an iterative layout improvement process based on HSIM simulation and analysis results generated in each iteration. Obtaining a highly optimized design may require several iterations; longer backend verification and optimization schedules may need to be allocated. However, additional analysis time can result in fewer silicon re-spins, more competitive products, and higher product quality.

Feedback

Chapter 2: Simulation Flow

Using HSIM in a Nanometer VLSI Design Flow

HSIM Command Groups

Lists the HSIM commands under their functional categories.

- Global Macro Set-Up Commands
- Netlist Commands
- Digital Vector File Commands
- Ungrounded/Grounded Capacitor Commands
- Isomorphic Matching Commands
- DC Initialization Commands
- Performance/Accuracy Control Commands
- Device Model Commands
- Connectivity Checking Commands
- Activity Checking Commands
- High-Impedance Node Checking Commands
- Signal Net Post-Layout Commands
- DPF/SPEF Backannotation Commands
- Selected Net Backannotation Command
- Output Control Commands
- AC and DC Analysis Commands
- Monte Carlo Analysis Commands
- Verilog-A Commands
- Miscellaneous Commands

Chapter 3: HSIM Command Groups

Global Macro Set-Up Commands

Global Macro Set-Up Commands

For details about these commands, see [HSIM Quick Start on page 12](#).

Global Macro Set-up Command	Default Setting
HSIMPRECISION on page 101	2
HSIMMOSPRECISION on page 112	0
HSIMPARPrecision on page 124	1
HSIMRCPRECISION on page 135	-1
HSIMVPRECISION on page 196	2

Netlist Commands

Netlist Command	Default Setting
HSIMNTLFMT on page 116	hspice
HSIMOPTSEARCHEXT on page 119	
HSIMSUBBUSDELIM on page 176	
HSIMTOP on page 179	null
HSIMWRAPPERSUB on page 200	

Digital Vector File Commands

Vector File Command	Default Setting
HSIMVCD2VEC on page 191	
HSIMVECTORFILE on page 192	
HSIMVHI , HSIMVHL on page 193	
HSIMVLO on page 194	

Ungrounded/Grounded Capacitor Commands

For information about how the HSIMFCAP, HSIMFCAPR, HSIMFCM, HSIMGCAP, and HSIMGCAPR commands work together, see [Control Commands for Floating Capacitors on page 347](#).

Ungrounded/Grounded Capacitor Commands	Default Setting
HSIMFCAP	1E-13

Feedback

Chapter 3: HSIM Command Groups

Ungrounded/Grounded Capacitor Commands

Ungrounded/Grounded Capacitor Commands	Default Setting
HSIMFCAPR	0.5
HSIMFCM	3
HSIMGCAP	1E-15
HSIMGCAPR	0.02
HSIMKEEPALLGNDCAPS on page 102	
HSIMLUMPCAP on page 104	1

Isomorphic Matching Commands

Isomorphic Matching Command	Default Setting
HSIMRCMATCHRELERR on page 134	0 . 05 or 5%
HSIMPORTCR on page 125	0 . 01 or 1%
HSIMPORTI on page 125	0 . 001 pV
HSIMPORTV on page 126	1 mV

DC Initialization Commands

DC Command	Default Setting
HSIMANALOG on page 61	1
HSIMANLASCIIPRSN on page 62	
HSIMDCI on page 76	5E- 7 Amperes
HSIMDCINIT on page 76	1
HSIMDCITER on page 77	250
HSIMDCSTEP on page 78	1000 picoseconds
HSIMDCSTOPAT on page 79	
HSIMDCV on page 79	0 . 001 Volts
HSIMENHANCEDC on page 87	0
HSIMIDDQ on page 98	
HSIMKEEPNODESET on page 103	0

Chapter 3: HSIM Command Groups

Performance/Accuracy Control Commands

DC Command	Default Setting
HSIMMULTIDC on page 114	1
HSIMNODCNODES on page 115	0
HSIMSTOPIFNODC on page 173	0
HSIMWARNIFNODC on page 199	0

Performance/Accuracy Control Commands

Note: For information about how the HSIMFCAP, HSIMFCAPR, HSIMFCM, HSIMGCAP, and HSIMGCAPR commands work together, see [Control Commands for Floating Capacitors on page 347](#).

Performance/Accuracy Control Command	Default Setting
HSIMACE on page 54	0
HSIMALLOWEDDV on page 59	0 . 3 Volts
HSIMANALOG on page 61	1
HSIMASIM on page 63	0
HSIMAUTOVDD on page 64	0
HSIMAUTOVDDSUP on page 65	0
HSIMENHANCEDIDDQ on page 88	0
HSIMFCAP	1E-13
HSIMFCAPR	0 . 5
HSIMFCM	3
HSIMFLAT on page 89	0

Performance/Accuracy Control Command	Default Setting
HSIMGCAP	1E-15
HSIMGCAPR	0 . 02
HSIMHIEROPT on page 94	0
HSIMGCSC on page 91	1
HSIMIDDQ on page 98	
HSIMIPRECISION on page 101	2
HSIMKEEPALLGNDCAPS on page 102	
HSIMLUMPCAP on page 104	1
HSIMMATCHOPT on page 105	0
HSIMMQS on page 113	1
HSIMPORTCR on page 125	0 . 01 or 1%
HSIMPORTI on page 125	0 . 001 pA
HSIMPORTV on page 126	0 . 001
HSIMPREFERVERILOGA on page 128	0
HSIMSAMPLERATE on page 146	160
HSIMSNCS on page 148	1
HSIMSPEED on page 150	3
HSIMSPICE on page 170	0
HSIMSTEADYCURRENT on page 172	10 nA
HSIMSTEP2TAUMAX on page 173	0
HSIMTAUMAX on page 176	2 ns

Feedback

Chapter 3: HSIM Command Groups

Performance/Accuracy Control Commands

Performance/Accuracy Control Command	Default Setting
HSIMTIMESCALE on page 177	1 picosecond
HSIMTLINEDV on page 178	1
HSIMTRAPEZOIDAL on page 179	0
HSIMVDD on page 191	3.0 Volts
HSIMVSRCDV on page 196	
HSIMVSRCSYNC on page 198	0

Device Model Commands

Device Model Command	Default Setting
HSIMABMOS on page 52	0
HSIMACCURATERDS on page 53	0
HSIMAUTOVDD on page 64	
HSIMB4REMOVE on page 65	0
HSIMBSIM3ISUB on page 70	
HSIMBJTCC on page 70	0
HSIMBJTCV on page 70	0
HSIMBMOS on page 70	0
HSIMCAPCC on page 71	0
HSIMCC on page 72	0
HSIMCHECKMOSBULK on page 74	0
HSIMDCSOIACC on page 78	0
HSIMDETAILBSIM4 on page 80	0
HSIMDIOCC on page 81	0
HSIMDIOCV on page 81	0
HSIMDIODECURRENT on page 81	0
HSIMDIODEVT on page 82	0.5
HSIMEFFICIENTTBL on page 86	0
HSIMENHANCECAP on page 87	0
HSIMENHANCEDIDDQ on page 88	0

Feedback

Chapter 3: HSIM Command Groups

Device Model Commands

Device Model Command	Default Setting
HSIMENHANCEMOSIV on page 89	0
HSIMFORWARDBIASDIODE on page 90	0
HSIMGATEMOD on page 91	0
HSIMIDDDQ on page 98	
HSIMIGISUB on page 99	0
HSIMMODELSTAT on page 108	0
HSIMUSEHM on page 180	
HSIMUSEHMINST on page 182	
HSIMUSERLIB on page 184	
HSIMNVBS on page 117	200
HSIMNVDS on page 118	60
HSIMNVGS on page 118	60
HSIMVBS3END on page 190	
HSIMVBS3START on page 190	
HSIMVDD on page 191	3 . 0 Volts
HSIMVDSEND on page 192	HSIMVDD+1

Connectivity Checking Commands

Connectivity Checking Command	Default Setting
HSIMBUSDELIMITER on page 71	
HSIMCONNCHECK on page 75	1
HSIMSUBBUSDELIM on page 176	

Activity Checking Commands

Activity Checking Command	Default Setting
HSIMACHECKFANOUT on page 54	0
HSIMACHECKRFEDGE on page 56	0
HSIMINACTOPT on page 100	2

High-Impedance Node Checking Commands

High-Impedance Node Checking Command	Default Setting
HSIMHZ on page 95	0
HSIMHZALL on page 96	0
HSIMHZFANOUT on page 96	0
HSIMHZNDNAME on page 97	
HSIMHZSTART on page 97	
HSIMHZSTOP on page 98	

Feedback

Chapter 3: HSIM Command Groups

High-Impedance Node Checking Commands

High-Impedance Node Checking Command	Default Setting
HSIMHZTIME on page 98	5 . 0
HSIMHZXSUBCKT on page 98	

Signal Net Post-Layout Commands

Signal Net Post-Layout Command	Default Setting
HSIMCMIN on page 74	
HSIMDELAYNTLRMIN on page 79	0
HSIMLMIN on page 103	1E-11
HSIMNTLRMIN on page 117	Current value of HSIMRMIN .
HSIMPOSTL on page 126	0
HSIMPOSTLMANY2M on page 128	0
HSIMPOSTLONE2M on page 128	0
HSIMRCNPO on page 134	0
HSIMRCRIO on page 136	
HSIMRCRKEEPELEM on page 137	
HSIMRCRKEEPMODEL on page 137	
HSIMRCRKEEPNODE on page 138	
HSIMRCRTAU on page 138	1.0e-10 seconds
HSIMRMIN on page 145	0.1
HSIMVSRCLMIN on page 197	1.0e-10
HSIMVSRCRMIN on page 197	0.1

DPF/SPEF Backannotation Commands

DPF/SPEF Backannotation Commands	Default Setting
HSIMCAPFILE on page 72	
HSIMDPF on page 82	
HSIMDPFHIERID on page 83	Period character (".")
HSIMDPFMULTISUB on page 83	0
HSIMDPFPFX on page 83	m_
HSIMDPFREPORTNOBA on page 84	0
HSIMDPFSCALE on page 84	1
HSIMDPFSFX on page 85	Asterisk character ("@")
HSIMDPFSPLITFD on page 85	0
HSIMDTNAME on page 86	
HSIMSPFEFTRIPLET on page 169	2
HSIMSPF , HSIMSPEF on page 154	
HSIMSPFADDNETPINXY on page 154	
HSIMSPFCC on page 155	0
HSIMSPFCCSCALE on page 155	1
HSIMSPFCHLEVEL on page 156	
HSIMSPFCMIN on page 156	
HSIMSPFCNET on page 156	
HSIMSPFDSJNET on page 157	0
HSIMSPFFDELIM on page 157	0

DPF/SPEF Backannotation Commands	Default Setting
HSIMSPFFEEDTHRU on page 157	0
HSIMSPFHLEVEL on page 159	
HSIMSPFKEEPNODECAP on page 159	
HSIMSPFKS on page 159	0
HSIMSPFMSGLEVEL on page 160	1
HSIMSPFMULTISUB on page 160	0
HSIMSPFNETPINDEL on page 162	
HSIMSPFNETWARNFILTER on page 163	
HSIMSPFNOWARNONCAP on page 163	0
HSIMSPFPOS on page 164	
HSIMSPFPRINTPIN on page 165	0
HSIMSPFRCNET on page 167	
HSIMSPFREPORTNOBA on page 168	0
HSIMSPFSKIPNET on page 168	
HSIMSPFSKIPPWNET on page 169	0
HSIMSPFSKIPSIGNET on page 169	0
HSIMSPFSPLITCC on page 169	0
HSIMSPFWARNFILE on page 170	0
HSIMSTNAME on page 174	
HSIMSTSswap on page 175	

Selected Net Backannotation Command

Selected Net Backannotation Command	Default Setting
HSIMACHECKFILENAME on page 55	
HSIMACHECKOUTFMT on page 55	

Output Control Commands

Output Control Command	Default Setting
HSIMCOILIB on page 74	
HSIMMEASOUT on page 106	0
HSIMOUTPUT on page 119	fsdb format.
HSIMOUTPUTFLUSH on page 120	-1
HSIMOUTPUTFSDBSIZE on page 121	-1
HSIMOUTPUTIRES on page 121	1E-9, or 1 nA
HSIMOUTPUTMEAS on page 121	1
HSIMOUTPUTTBL on page 122	
HSIMOUTPUTTRES on page 122	1E-12 or 1 ps
HSIMOUTPUTTSTEP on page 122	Not set.
HSIMOUTPUTVRES on page 123	1E-3, or 1 mV
HSIMOUTPUTWDFSIZE on page 123	-1
HSIMTIMEFORMAT on page 177	1
HSIMUTFCOMPRESSLEVEL on page 185	3

AC and DC Analysis Commands

AC Analysis Command	Default Setting
HSIMACOUT on page 57	0
HSIMACOUTFMT on page 59	0
HSIMACFREQSCALE on page 54	
HSIMANLASCIPRSN on page 62	
HSIMDCI on page 76	5E-7 A
HSIMDCINIT on page 76	1
HSIMDCITER on page 77	250
HSIMDCOUTFMT on page 77	0
HSIMDCSOIACC on page 78	0
HSIMDCSTEP on page 78	1000 picoseconds
HSIMDCSTOPAT on page 79	
HSIMDCV on page 79	0.001 Volts
HSIMUSEPREVIOUSDC on page 184	0

Monte Carlo Analysis Commands

Monte Carlo Analysis Command	Default Setting
HSIMMONTECARLOMOD on page 109	1
HSIMMONTECARLOINST on page 109	1
HSIMMONTECARLOSAVEOUT on page 109	1

Verilog-A Commands

Verilog-A Command	Default Setting
HSIMPREFERVERILOGA on page 128	
HSIMUSEVA on page 184	
HSIMUSEVATABLE on page 185	0
HSIMVABRANCHPART on page 187	
HSIMVACROSSTTOL on page 188	100 picoseconds
HSIMVACROSSVTOL on page 188	0.1 Volt
HSIMVAPARTITION on page 189	0
HSIMVAPRINTVAR on page 189	0
HSIMVATBLERANGE on page 190	2*Vdd
HSIMVTABLESIZE on page 190	100
HSIMVERILOGA on page 193	

Miscellaneous Commands

Miscellaneous Command	Default Setting
HSIMACHECKFANOUT on page 54	0
HSIMACHECKRFEDGE on page 56	
HSIMALLOWEDDV on page 59	0.3 Volts
HSIMDELVTO on page 80	
HSIMHIERID on page 94	Period character (“.”)

Miscellaneous Command	Default Setting
HSIMHSPICEVEC on page 95	0
HSIMLIS on page 103	0
HSIMLOGDCPROGRATE on page 104	0 .1
HSIMLOGPROGRATE on page 104	0 .1
HSIMINACTOPT on page 100	2
HSIMNODECAP on page 115	
HSIMNOSIMTIME on page 116	0
HSIMOPCOMPRESS on page 118	0
HSIMPRINTSIMSTATUS on page 130	1
HSIMREDEFSUB on page 139	0
HSIMSCALE on page 147	1E-6
HSIMSPISCIUNITERR on page 172	
HSIMTIMESCALE on page 177	1 picosecond
HSIMTRAPEZOIDAL on page 179	0
HSIMVCD2VEC on page 191	
HSIMVECTORFILE on page 192	
HSIMVHI, HSIMVHL on page 193	
HSIMVLO on page 194	
HSIMWARNSTOP on page 199	0

Feedback

Chapter 3: HSIM Command Groups

Miscellaneous Commands

HSIM® Commands in Alphabetical Order

Lists the HSIM® commands in alphabetical order.

.BIASCHK

Monitors device voltage bias, current, size, expression, region, or temperature.

Syntax

As an expression monitor

```
.BIASCHK 'expression' [limit=lim] [noise=ns]
+ [max=max] [min=min]
+ [simulation=op|dc|tr|all] [monitor=v|i|w|l]
+ [tstart=time1] [tstop=time2] [autostop]
+ [interval=time] [BIASNAME=val]
```

As an element and model monitor

```
.BIASCHK type terminal1=t1 [terminal2=t2]
+ [limit=lim] [noise=ns] [max=max] [min=min]
+ [simulation=op|dc|tr|all] [monitor=v|i]
+ [name=name1,name2,...]
+ [mname=modname_1,modname_2,...]
+ [tstart=time1] [tstop=time2] [autostop]
+ [except=name_1,name_2,...]
+ [interval=time] [sname=subckt_name1,subckt_name2,...]
+ [BIASNAME=val]
```

As a region monitor

```
.BIASCHK MOS [region=cutoff|linear|saturation]
+ [simulation=op|dc|tr|all]
+ [name=name1,name2,...]
+ [mname=modname_1,modname_2,...]
+ [tstart=time1] [tstop=time2] [autostop]
+ [except=name1,name2,...]
+ [interval=time] [sname=subckt_name1,subckt_name2,...]
+ [BIASNAME=val]
```

As a length and width monitor

```
.BIASCHK type monitor=w|l
+ [limit=lim] [noise=ns] [max=max] [min=min]
+ [simulation=op|dc|tr|all]
+ [name=devname_1,devname_2,...]
+ [name=devname_n,devname_n+1,...]
+ [mname=modelname_1,modelname_2,...]
+ [tstart=time1] [tstop=time2] [autostop]
+ [interval=time] [sname=subckt_name1,subckt_name2,...]
```

+ [BIASNAME=val]

As a temperature monitor

```
.BIASCHK type monitor=temp
+ [limit=lim] [max=max] [min=min]
+ [simulation=op|dc|tr|all]
+ [name=devname_1, devname_2,...]
+ [mname=modelname_1, modelname_2,...]
+ [sname=subckt_name1, subckt_name2, ...]
+ [tstart=time1] [tstop=time2] [autostop]
+ [message="string"]
```

Argument	Description
type	Element type to check. MOS (C, BJT, ...) For a monitor, <i>type</i> can be DIODE, BIPOLAR, BJT, JFET, MOS, NMOS, PMOS, R, or C. When used with REGION, <i>type</i> can be MOS only.
terminal 1, 2	Terminals between which HSPICE checks (that is, checks between <i>terminal1</i> and <i>terminal2</i>): <ul style="list-style-type: none"> ▪ For MOS level 57: nd, ng, ns, ne, np, n6 ▪ For MOS level 58: nd, ngf, ns, ngb ▪ For MOS level 59: nd, ng, ns, ne, np ▪ For other MOS level: nd, ng, ns, nb ▪ For resistor: n1, n2 ▪ For capacitor: n1, n2 ▪ For diode: np, nn ▪ For bipolar: nc, nb, ne, ns ▪ For JFET: nd, ng, ns, nb For type=subckt, the terminal names are those pins defined by the subcircuit definition of mname.
limit	Bias check limit that you define. Reports an error if the bias voltage (between appointed terminals of appointed elements and models) is larger than the limit.
noise	Bias check noise that you define. The default is 0.1v. Noise-filter some of the results (the local maximum bias voltage that is larger than the limit). The next local max replaces the local max if all of the following conditions are satisfied: <i>local_max-local_min noise</i> . <i>next local_max-local_min noise</i> . This local max is smaller than the next local max. For a parasitic diode, HSPICE ignores the smaller local max biased voltage and does not output this voltage. To disable this feature, set the noise detection level to 0.
max	Maximum value.

Chapter 4: HSIM® Commands in Alphabetical Order

.BIASCHK

Argument	Description
min	Minimum value.
name	Element name to check. If name and mname are not both set for the element type, the elements of this type are all checked. You can define more than one element name in keyword name with a comma (,) delimiter. If doing bias checking for subcircuits: <ul style="list-style-type: none">▪ When both mname and name are defined while multiple name definitions are allowed if a name is also an instance of mname, then only those names are checked, others will be ignored.▪ This command is ignored if no name is an instance of mname.▪ For name definitions which are not of the type defined in mname will be ignored.▪ If a mname is not defined, the subcircuit type is determined by the first name definition.
mname	Model name. If you are doing bias checking for a subcircuit, it is the subcircuit definition name. HSPICE checks elements of the model for bias. If you define mname, then HSPICE checks all devices of this model. You can define more than one model name in the keyword mname with the comma (,) delimiter. If <i>mname</i> and <i>name</i> are not both set for the element type, the elements of this type are all checked. If doing bias checking for subcircuits: <ul style="list-style-type: none">▪ Once there is one and only one mname defined, the terminal names for this command are those pins defined by the subckt definition of mname.▪ Multiple mname definitions are not allowed.▪ Wildcards are supported for mname.▪ If only mname is specified in a subckt bias check, then all subcircuits will be checked. See also sname below.
region	Values can be cutoff, linear, or saturation. HSPICE monitors when the MOS device, defined in the .BIASCHK command, transitions to and from the specified region (such as cutoff).
simulation	Simulation type you want to monitor. You can specify op, dc, tr (transient), and all (op, dc, and tr). The tr option is the default simulation type.
monitor	Type of value you want to monitor. You can specify v (voltage), i (current), w, and l (device size) for the element type, or temperature. This parameter is not used for an expression-type monitor.
tstart	Bias check start time during transient analysis. The default is 0.

Argument	Description
tstop	Bias check end time during transient analysis. The analysis ends on its own by default if you do not set this parameter.
autostop	When set, HSPICE supports an autostop for a biaschk card so that it can report error messages and stop the simulation immediately.
except	Specify the element or instance that you do not want to bias check.
interval	Active when .OPTION BIASINTERVAL is set to a nonzero value. This argument prevents reporting intervals that are less than or equal to the time specified.
sname	Name of the subcircuit definition that the <i>type</i> of element of lies in. HSPICE checks all elements in this subcircuit for bias. You can define more than one subcircuit name in the keyword <i>sname</i> with a comma (,) delimiter. If you are doing bias checking for a subcircuit, <i>sname</i> = the X-element name.
biasname	Keyword to organize multiple .biaschk commands and their outputs in final bias check results file for viewing violation details in GUI applications such as SAE and HAI.
message	"string" is a user-defined warning message for any issue that .BIASCHK monitors. The issue is reported in the *.lis file, including the string you specify. See Example 9.

Description

Use this command to monitor the voltage bias, current, device size, expression, region, or temperature during analysis. The output reports:

- Element (instance) name
- Time
- Terminals
- Bias that exceeds the limit
- Number of times the bias exceeds the limit for an element
- User-defined warning message for monitored temperature exceeding limits

HSPICE saves the information as both a warning and a bias check summary in the *.lis file or a file you define in the BIASFILE option. You can use this command only for active elements, resistors, capacitors, and subcircuits.

Chapter 4: HSIM® Commands in Alphabetical Order

.BIASCHK

More than one simulation type or all simulation types can be set in a single .BIASCHK command. Also, more than one region can be set in a single .BIASCHK command.

After a simulation that uses the .BIASCHK command runs, HSPICE outputs a results summary including the element name, time, terminals, model name, and the number of times the bias exceeded the limit for a specified element.

The keywords *name*, *mname*, and *sname* act as OR'd filters for element selection. Also, if *type* is *subckt* in a .BIASCHK command that tries to check the ports of a subcircuit, the keyword *sname* then behaves identically to the *name* keyword.

Element and model names can contain wildcards, either "?" (stands for one character) or "*" (stands for 0 or more characters).

If a model name that is referenced in an active element command contains a period (.), then .BIASCHK reports an error. This occurs because it is unclear whether a reference such as x.123 is a model name or a subcircuit name (123 model in "x" subcircuit). With version F-2011.09-SP1, you can conduct node voltage error checks within model subcircuits instead of defining these in a netlist (top-level).

If you do not specify an element and model name, HSPICE checks all elements of this type for bias voltage (you must include *type* in the BIASCHK card). However, if *type* is *subckt* at least one element or model name must be specified in the .BIASCHK command; otherwise, a warning message is issued and this command is ignored.

Note: To perform a complete bias check and print all results in the Outputs Biaschk Report, do not use .protect/.unprotect in the netlist for the part that is used in .biaschk. For example: If a model definition such as model nch is contained within .prot/.unprot commands, in the * .lis you'll see a warning message as follows: **warning** : model nch defined in .biaschk cannot be found in netlist--ignored

Examples

Example 1 Monitoring an expression:

```
.biaschk 'v(1)' min='v(2)*2' simulation= op
```

Example 2 Monitoring element m1 and model types between two specified terminals.

```
.biaschk nmos terminal1=ng terminal2=ns simulation=tr name=m1
```

Example 3 Monitoring MOSFET model m1 whose bias voltage exceeds 2.5 V and interval exceeds 5 ns.

```
.biaschk nmos terminal1=nb terminal2=ng limit=2.5  
+ mname=m1 interval=5n
```

Example 4 The following two examples use .BIASCHK commands that do not require terminal specifications. Example 4 monitors the MOS transistor region of operation

```
.biaschk mos region=saturation name=x1.m1 mname=nch name=m2
```

Example 5 Monitors MOS transistor length and width.

```
.biaschk mos monitor=l mname=m* p* min=1u minu=op
```

Example 6 Defines the differences between using .BIASCHK with a MOSFET instance and macro models. If the MOSFET in your netlist is written as follows:

```
mp0 gd s sub dgxnft w=1.0u l=0.18u
```

...then the .BIASCHK statement can be written as:

```
.biaschk pmos terminal1=ns terminal2=nd mname=m*.dgnfet*  
+ limit=0.9v
```

Example 7 If a macro model is used, then the MOSFET is defined inside a subcircuit:

```
.subckt test g d  
mp0 g d s sub dgxnft w=1.0u l=0.18u  
.model dgxnft nmos level=54  
.ends  
X1 g d test
```

For this case

```
.biaschk pmos terminal1=ns terminal2=nd mname=x*.dgnfet*  
+ limit=0.9v
```

Example 8 Temperature monitoring

```
*Monitor temperature (main netlist)  
.temp 180  
x1 c b e vpb4u area=4
```

```
*Model file  
.subckt vpb4u 1 2 3 ...  
q0 c b e n_bjt DTEMP=30  
.model n_bjt npn  
.ends vpb4u  
.biaschk subckt monitor=temp max=200 min=-40 mname=vpb4u
```

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMABMOS**Output in *.lis file**

```
**warning** (test.sp: 4) Element temperature of vpb4u.q0, 210,  
has exceeded max or min limit.
```

Example 9 User defined message

```
.biaschk nmos terminal1=nd monitor=i limit=-1u  
+ message=' mosfet terminal current exceeds max value'
```

The *.lis file reports the following warning:

```
**warning** (t1.sp:33) mosfet terminal current exceeds max value  
type terminals time Vbias method model-name element-name  
subckt-name nmos i(nd) 0. 905.7552n limit nmos x1.mn inv
```

For a full example netlist go to:

```
$installdir/demo/hspice/apps/biaschk.sp
```

HSIMABMOS

Sets the Precise Model on the coupling effect between MOSFET drain, source, gate, and bulk terminals. Simulation slows down significantly when HSIMABMOS is set to 1. You should only use it locally.

Syntax

```
.param HSIMABMOS = <0 | 1>
```

Argument	Description
0	Turns HSIMABMOS off (the default).
1	Turns HSIMABMOS on and is recommended for flash memory circuits.

HSIMABSVOLTOL

Sets the absolute voltage tolerance.

Syntax

```
.param HSIMABSVOLTOL = value
```

HSIMACCSUBHTBL

When HSIMACCSUBHTBL is set to 1, subthreshold region of BSIM3 model gets double size mesh in model table. Default value is 0.

Syntax

```
.param HSIMACCSUBHTBL = < 0 | 1 >
```

HSIMACCURATERDS

Controls accuracy levels for the NRS or NRD instance parameters used in all MOSFET models.

Syntax

```
.param HSIMACCURATERDS = <0 | 1 | 2>
```

Argument	Description
0	HSIM applies a simple scaling method to calculate drain current to achieve good accuracy and a reasonable run time. This is the default setting.
1	Applies a more accurate scaling method to calculate drain current.
2	Applies a SPICE-like method to calculate current and capacitance.

Description

When you set HSIMACCURATERDS =1, there is a penalty in simulation time compared to the default setting of 0. When you set HSIMACCURATERDS =2, HSIM applies a SPICE-like method to calculate current and capacitance. Because this value has a long run time penalty, it is recommended only when you need the highest possible accuracy.

HSIMACE

Improves simulation runtimes specifically for memory design characterizations that require very high accuracy and contain large partitions.

Syntax

```
.param HSIMACE = <0|1|2>
```

Argument	Description
0	Turns HSIMACE off.
1	Enables the advanced time-step and model evaluation algorithm for large partitions. This is the recommended setting for most memory design characterizations.
2	Same as the 1 argument, but with a more conservative internal threshold criteria. Use this setting for sensitive circuits, such as DLLs, that require additional accuracy.

HSIMACFREQSCALE

Sets the frequency units when HSIM outputs the results of AC analysis.

Syntax

```
.param HSIMACFREQSCALE=freq_units
```

Examples

In the following example, frequency units are set to MHz.

```
.param HSIMACFREQSCALE=1e6
```

HSIMACHECKFANOUT

Checks all nodes or only the nodes that connect to MOSFET gates. This command is useful when you only want to check driver nodes.

Syntax

```
.param HSIMACHECKFANOUT=<0 | 1>
```

Argument	Description
0	Checks all of the nodes. This is the default setting.
1	Checks only nodes that are connected to MOSFET gates.

Examples

```
.param HSIMACHECKFANOUT=1  
.acheck '*' dv=0.5 exclude='in,1'
```

Check all driver nodes except the in and 1 that have voltage changes greater than 0.5 volts during the simulation.

HSIMACHECKFILENAME

Writes active nets to the specified file.

Syntax

```
.param HSIMACHECKFILENAME=<filename>
```

Description

If HSIMACHECKFILENAME is not specified, and HSIMACHECKOUTFMT is set to 1, HSIM writes the active nets to the <prefix>.rcxt file. If HSIMACHECKFILENAME is not specified, and HSIMACHECKOUTFMT is set to 2, HSIM writes the active nets to the <prefix>.hsimba file. In both cases, <prefix> is the -o parameter specified in the HSIM command line.

HSIMACHECKOUTFMT

Specifies the format to report the active nets. It is mandatory during the Initial Activity Run. It is not included in the post-layout simulation run.

Syntax

```
.param HSIMACHECKOUTFMT=<1 | 2>
```

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMACHECKRFEDGE**

Argument	Description
1	Reports active nets in StarRC (*.rcxt) format for the selective net extraction Flow.
2	Reports active nets in HSIM backannotation (*.hsimba) format for the selective net backannotation flow.

HSIMACHECKRFEDGE

Checks if signals have rising or falling edge activity for the nodes that belong to the node pattern.

Syntax

```
.param HSIMACHECKRFEDGE=<0 | 1>
```

Argument	Description
0	No edge checking (default).
1	Checks all nodes defined by node pattern.

Description

The rising threshold is set as 0.75*(global voltage), and the falling threshold is set as 0.25*(global voltage). If a signal starts off low and rises above the rising threshold, a rising-edge activity is reported in the file_name.rfedge file. Conversely, if a signal starts off high and falls below the falling threshold, a falling-edge activity is reported in the same file.

Examples

```
.param HSIMACHECKRFEDGE=1  
.acheck V(x3m.x0.x1.xsap.x1.*)
```

The output file format of the `HSIMACHECKRFEDGE` parameter is:

hierarchy_node node_voltage activity_time(ns) activity_type

The following is an output file example. [Table 1](#) shows the output file format.

```

Output file: hsim.rfedge
*HSIM Win32 Debug Version 6.0 - 155206302004
*Tracking No - HSIM 2005.26.7
*Copyright (C) 1998 - 2005. All rights reserved.
*
*
*Rising (R) and Falling (F) edge detection

```

Table 1 HSIMACHECKRFEDGE Output File Format

hierarchy_node	node_voltage	activity_time (ns)	activity_type
x3m.x0.x1.xsap.x1.pc	0.77	4.90	F
x3m.x0.x1.xsap.x1.x2.gdx	2.49	30.40	R
x3m.x0.x1.xsap.x1.da1	2.55	30.74	R
x3m.x0.x1.xsap.x1.dan	0.71	31.09	F
x3m.x0.x1.xsap.x1.x2.ctrb	0.63	120.86	F
x3m.x0.x1.xsap.x1.s1	2.53	126.20	R
x3m.x0.x1.xsap.x1.s2	2.53	147.12	R
x3m.x0.x1.xsap.x1.out	0.64	147.12	F

HSIMACOUT

Controls the definitions used for magnitude (m), phase (p) and decibel (db) modifiers for AC analysis.

Syntax

```
.param HSIMACOUT=<0 | 1>
```

Argument	Description
0	Applies the SPICE definition.
1	Applies the default definition.

Description

The following formulae apply to the SPICE definition:

0

```
vm (nd1,nd2) = abs(v(nd1)-v(nd2))  
vp (nd1,nd2) = phase(v(nd1)-v(nd2))  
vdb (nd1,nd2) = 20 log10(abs(v(nd1)-v(nd2)))  
1 (default)  
vm (nd1,nd2) = vm(nd1) - vm(nd2)  
vp (nd1,nd2) = vp(nd1) - vp(nd2)  
vdb (nd1,nd2) = 20 log10(vm(nd1)/vm (nd2))
```

Note: abs and phase are the magnitude and the phase of the complex phasor.

The following formulae apply to the default definition:

```
vm (nd1,nd2)=vm(nd1) - vm(nd2)  
vp (nd1,nd2)=vp(nd1) - vp(nd2)  
vdb (nd1,nd2)=20 log10(vm(nd1)/vm (nd2))
```

Examples

```
.print ac vm(n1,n2) vp(n2,n3) vdb(n3,n4) vr(n4,n5) vi(n5,n1)
```

The following formulae are applied to this definition:

- $vm(nd1,nd2) = (abs(v(nd1)-v(nd2)))$
- $vp(nd1,nd2) = phase(v(nd1)-v(nd2))$
- $vdb(nd1,nd2) = 20 \log10(abs(v(nd1)-v(nd2)))$
- $vrvr(n4,n5) = real(v(n4)-v(n5))$
- $vivi(n5,n1) = imag(v(n5)-v(n1))$

HSIMACOUTFMT

Sets the AC analysis output format.

Syntax

```
.param HSIMACOUTFMT = <0 | 2 | 4 | 7>
```

Argument	Description
0	Default ASCII table format.
2	ASCII raw file format.
4	Similar to HSIMACOUTFMT=2, but if there is an external sweep in AC analysis, the results for each value of the external sweep go to a separate external file.
7	This option is for the WDF output format. Use this option with HSIMOUTPUT =wdf to allow the AC results to be printed in WDF format.

Description

HSIMACOUTFMT can set the following file output file formats for AC analysis:

- 0 specifies that results are written to the <project>.ac file.
- 2 specifies that results are written to the <project>.ac.raw file.
- 4 specifies that if there is an external sweep in AC analysis, the results for each value of the external sweep go to a separate external file such as <project>.ac.raw.s1, <project>.ac.raw.s2, and so on.
- 7 is used for output formats of third party vendors. Use this option with [HSIMOUTPUT](#). Combining HSIMACOUTFMT and [HSIMOUTPUT](#) on [page 119](#) allows the AC result to be printed in third party vendor output formats.

HSIMALLOWEDDV

Dynamically adjusts the time step size so that each node voltage change over the time step is limited by the specified value.

Syntax

```
.param HSIMALLOWEDDV =voltage_value
```

Description

The default value for HSIMALLOWEDDV is 0.3V. The larger the value is, the faster the simulation speed. Better precision is achieved by using a smaller HSIMALLOWEDDV value. The default of 0.3V might give the optimal trade-off between precision and speed in most applications.

A reduced HSIMALLOWEDDV setting is recommended for small analog circuits and circuits with low power-supply voltage. HSIMALLOWEDDV is recommended to be set to 10% of the power-supply voltage when the power-supply voltage is below 2V. By specifying [HSIMVDD on page 191](#), HSIMALLOWEDDV is reset to 10% of the [HSIMVDD on page 191](#) value.

HSIMALLOWEDDV can be set in any local subcircuit, either subcircuit definition or subcircuit instance, for a local effect on the associated subcircuit. If a design has multiple power supplies, [HSIMAUTOVDD on page 64](#) can be used to perform a path search and set HSIMALLOWEDDV as required.

HSIMAMOS

When set to 1, HSIMAMOS turns on the Precise Model for the coupling effect of MOSFETs in a subcircuit definition or a subcircuit instance.

Syntax

```
.param HSIMAMOS = <0 | 1>
```

Description

The \$AMOS setting is limited to an individual MOSFET, either at the top level or within a subcircuit definition. If \$AMOS is set to 1 for a MOSFET in the subcircuit definition, then each corresponding MOSFET in the subcircuit instances have the same setting. The default is 0.

HSIMANALOG

Controls the complexity of the analog simulation algorithm.

Syntax

```
.param HSIMANALOG = <-1 | 0 | 1 | 2 | 3>
```

Argument	Description
-1	In simulating digital circuits where no analog circuit behavior is expected, set HSIMANALOG=-1 . This setting assumes no sensitive coupling exists between neighboring subcircuits.
0	Might cause some feedback couplings across the hierarchy boundary not to be simulated accurately.
1	Extends precision to cover feedback couplings across the hierarchy boundary. Unless the memory usage is crucial, HSIMANALOG=1 is recommended because it increases memory usage in order to flatten the subcircuits containing feedback couplings.
2	Further increases the complexity of the analog simulation algorithm by extending the feedback couplings to a wider scope, such as including voltage control oscillators (VCOs) found in most PLL circuits. HSIMANALOG=2 is recommended for use with analog circuits containing highly sensitive topology such as: <ul style="list-style-type: none"> ▪ Bandgap reference voltage generator ▪ High-gain amplifier ▪ VCO ▪ Switched-capacitor filter ▪ PLL ▪ A/D and D/A converters
3	Automatically applies HSIMSPICE on page 170 =3 to those MOSFETs involved in feedback coupling. This setting is recommended for simulating A/D and D/A converters.

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMANLASCIPRSN**Description**

To achieve optimal HSIM performance in analog and mixed-signal circuit simulation, HSIMANALOG controls the complexity of analog simulation algorithm. The higher the value you specify, the more time-consuming and precise the analog simulation algorithm is.

The HSIMANALOG=0 and HSIMANALOG=1 settings are used to simulate full-custom circuits or memory circuits. Compared with HSIMANALOG=-1, these commands provide for increased complexity from more precise simulation in local feedback coupling between neighboring subcircuits. The difference between HSIMANALOG=0 and HSIMANALOG=1 is related to the circuit hierarchy.

The cross-coupled inverter pair in an SRAM cell is simulated more accurately compared with the HSIMANALOG=-1 setting.

Setting HSIMANALOG=2 or HSIMANALOG=3 causes simulation speed to slowdown due to the more conservative analog simulation algorithms. There are two methods to speed up analog simulation.

1. Set HSIMANALOG=2 or HSIMANALOG=3 in selected subcircuits that contain sensitive analog subcircuits.
2. Set HSIMANALOG=1.5, which invokes the fast mixed-signal simulation mode. HSIMANALOG=1.5 applies the same analog simulation algorithm as the setting HSIMANALOG=2.

Note: The difference between HSIMANALOG=1.5 and HSIMANALOG=2 is that HSIMGCSC is automatically set to 0 for those MOSFETs not involved in feedback coupling. The speedup in setting HSIMANALOG=1.5 is achieved by sacrificing the timing delay precision in the digital portion of the circuit.

HSIMANLASCIPRSN

Sets the precision as the number of printed digits in output ASCII files. The minimum value is 2 and maximum value is 15. This command sets HSIM output of DC analysis to 12-digit representation. Any number larger than 15 is reset to 15, and any number smaller than 2 is reset to 2.

Syntax

```
.param HSIMANLASCIPRSN =value
```

HSIMASIM

Enables a more conservative analog simulation algorithm for smaller analog circuits containing less than 50K elements. It is not recommended to apply this setting for large full-chip simulation, especially for large memory circuits. The default is 0.

HSIMASIM is a local variable. If it is a macro-model, it lists the other variables that are set through it.

Syntax

```
.param HSIMASIM = <0 | 1>
```

HSIMAUTOADV

Dynamically adjusts the time step size so that each node voltage change over the time step is limited by the specified fraction of the detected supply voltage.

Syntax

```
.param HSIMAUTOADV = value
```

Argument	Description
value	Specifies a fraction of the detected supply voltage value between 0 and 1.

Description

You use HSIMAUTOADV with HSIMAUTOVDD to set the allowed voltage change for a time step based on the detected supply voltage for the related circuit block. The allowed voltage change is determined as:

$$\text{allowed_dv} = \text{HSIMAUTOADV} * \text{detected_supply_voltage}$$

For example, if HSIMAUTOADV=0.2, and the detected supply voltage is 3V, then allowed_dv = 0.2 * 3V = 0.6V.

You can set HSIMAUTOADV at the top or subcircuit level. Its default value is 0.1.

A typical usage of HSIMAUTOADV is setting it to different values based on accuracy requirements of different blocks.

HSIMAUTOVDD

Traces a design and automatically identifies multiple power supplies.

Syntax

```
.param HSIMAUTOVDD = <0 | 1 | 2 | 3>
```

Argument	Description
0	Turns HSIMAUTOVDD off (default).
1	Use a setting of 1 for designs containing multiple power supplies. This setting performs a sophisticated path search to all feasible power supplies associated with every element and node. Note that you can use HSIMVDD to manually set the voltage supply value if the design contains only a single power supply. Do not use HSIMAUTOVDD and HSIMVDD together.
2	A setting of 2 has the same functionality as HSIMAUTOVDD=1, but also detects PWL/PULSE power supplies and sets voltage supplies to the respective maximum values in the detected PWL/PULSE functions.
3	A setting of 3 is the latest and most advanced power supply detection algorithm. It has same functionality as HSIMAUTOVDD=2 but can also detect internal power supply nodes in designs with header/footer switches, and sets correct voltage supply values to subcircuits with multiple partitions and power supplies. Use this value instead 2 and 3 as these two values will become obsolete in future releases.

Description

When simulating designs containing multiple power supply voltages, HSIM can trace the design and automatically identify multiple power supplies. HSIMAUTOVDD controls time-steps, logical print voltage thresholds, and MOS model table look-up ranges. HSIMAUTOVDD is a global parameter with a default value of 0.

HSIMAUTOVDDSUP

Determines the logical print voltage thresholds.

Syntax

```
.param HSIMAUTOVDDSUP = <0 | 1>
```

Description

In cases when a single node can be driven by multiple power supplies, use the HSIMAUTOVDDSUP=1 command to determine the logical print voltage thresholds. Specify HSIMAUTOVDDSUP = 0 to use the lowest supply voltage for the .lprint threshold calculation. This is the default.

In cases when logical print statements (.lprint) are used, and vlth/vhth or HSIMVLTH/HSIMVHTH are not specified, HSIM issues a warning message and uses the 30% and 70% of HSIMVDD as the lower and upper thresholds, respectively. If a design contains multiple power supplies, the setting HSIMAUTOVDD=1 automatically sets HSIMVLTH/HSIMVHTH to 30%/70% of the multiple voltage supplies for associated nodes.

Note: HSIMAUTOVDDSUP is only triggered if HSIMAUTOVDD = 1.

HSIMB4REMOVE

Use this command with BSIM4 models to remove the unused tables in the simulation for better memory utilization.

Syntax

```
.param HSIMB4REMOVE = <0 | 1>
```

Argument	Description
0	Unused tables are left in the simulation.
1	Removes unused tables from the simulation.

Description

When HSIMB4REMOVE=1 is set, the following usages stop simulations:

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMBISECTION**

- HSIMSPICE=2 is specified at the top and the lower levels of the subcircuits use HSIMSPICE=0/1.
- HSIMSPICE=2 is specified at the top and the lower levels of the subcircuits use HSIMSPICE=0/1.

HSIMBISECTION

Supports concurrent bisection optimization on multiple pins without re-reading the netlist, greatly reducing the data I/O overhead.

Syntax

```
.param HSIMBISECTION = <0 | 1>
```

Description

Concurrent bisection optimization only applies to PWL voltage source elements. Additional parameters must be specified in the following statements:

- PWL voltage source statement (bisectparam). Add the following syntax to the PWL statement:
`bisectparam=1`
- .param statement. Add the following syntax to the .param statement:
`.param HSIMBISECTION=1`
- .tran statement

Examples

```
.param HSIMBISECTION=1

Vin ck 0 pwl
+0 0
+'19n' 0
+'t_time+19n' 5
+bisectparam=1
Ven d 0 pwl
+'0' 0
+'delaytime' 0
+'t_time2+delaytime' 5
+ bisectparam=1
Vin2 ck2 0 pwl
+'0' 0
+'19n' 0
+'t_time+19n' 5
+bisectparam=1
Ven2 d2 0 pwl
+'0' 0
+'delaytime2' 0
+'t_time2+delaytime2' 5
+ bisectparam=1
.param delaytime=opt(0.0n, 0.0n, 20.0n)
.param delaytime2=opt2(0.0n, 0.0n, 20.0n)
.param t_time=alter1(0.1n, 0.1n, 0.1n, 0.5n, 0.5n, 0.5n, 1n, 1n, 1n)
.param t_time2=alter2(0.1n, 0.5n, 1n, 0.1n, 0.5n, 1n, 0.1n, 0.5n, 1n)
.tran 0.1n 40.0n
+ sweep
+ optimize=opt
+ result=maxvout
+ model=optmod
+ alterparam=alter1
+ alterparam=alter2
.tran 0.1n 40.0n
+ sweep
+ optimize=opt2
+ result=maxvout2
+ model=optmod
+ alterparam=alter1
+ alterparam=alter2
.model optmod opt
+ method=bisection
+ relin=1.0e-3
.measure tran ck_slope param=t_time
.measure tran d_slope param=t_time2
```

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMBISECTION**

```
.measure tran mt_delaytime param=delaytime
.measure tran maxvout max v(q) goal=5
.measure tran setuptrig trig v(d) val=2.5 rise=1
+ targ v(ck) val=2.5 rise=1
.measure tran ck_slope2 param=t_time
.measure tran d_slope2 param=t_time2
.measure tran mt_delaytime2 param=delaytime2
.measure tran maxvout2 max v(q2) goal=5
.measure tran setuptrig2 trig v(d2) val=2.5 rise=1
+ targ v(ck2) val=2.5 rise=1
```

In this example, HSIM extracts the setup time for two different flip-flops at the same time. The benefit of concurrent bisection optimization that is the total iteration time is similar to the iteration time required to extract one dff setup time, so this greatly reduces the total run time. Setup time can be extracted for a memory from data bus to clock without concurrent bisection by separately extracting the setup time for each data bus bit. The entire data bus-to-clock setup time can be extracted at the same time. The result of this example contains nine .mt files including the following:

```
2dff.mt.a0 :  
Measurement results:  
ck_slope=1.00000000000e-010  
d_slope=1.00000000000e-010  
mt_delaytime=1.87695300000e-008  
maxvout=5.00000000000e+000 at=4.00000000000e-008  
setuptime=2.30000000000e-010 targ=1.90500000000e-008  
trig=1.88200000000e-008  
ck_slope2=1.00000000000e-010  
d_slope2=1.00000000000e-010  
mt_delaytime2=1.87304700000e-008  
maxvout2=5.00000000000e+000 at=4.00000000000e-008  
setuptime2=2.70000000000e-010 targ=1.90500000000e-008  
trig=1.87800000000e-008  
2dff.mt.a1 :  
Measurement results:  
ck_slope=1.00000000000e-010  
d_slope=5.00000000000e-010  
mt_delaytime=1.84960900000e-008  
maxvout=5.00000000000e+000 at=4.00000000000e-008  
setuptime=3.04000000000e-010 targ=1.90500000000e-008  
trig=1.87460000000e-008  
ck_slope2=1.00000000000e-010  
d_slope2=5.00000000000e-010  
mt_delaytime2=1.84863200000e-008  
maxvout2=5.00000000000e+000 at=4.00000000000e-008  
setuptime2=3.14000000000e-010 targ=1.90500000000e-008  
trig=1.87360000000e-008  
...  
2dff.mt.a8 :  
Measurement results:  
ck_slope=1.00000000000e-009  
d_slope=1.00000000000e-009  
mt_delaytime=1.87304700000e-008  
maxvout=5.00000000000e+000 at=4.00000000000e-008  
setuptime=2.70000000000e-010 targ=1.95000000000e-008  
trig=1.92300000000e-008  
ck_slope2=1.00000000000e-009  
d_slope2=1.00000000000e-009  
mt_delaytime2=1.86523500000e-008  
maxvout2=5.00000000000e+000 at=4.00000000000e-008  
setuptime2=3.48000000000e-010 targ=1.95000000000e-008  
trig=1.91520000000e-008
```

HSIMBJTCC

Invokes the exact charge conservation model for the charge stored by the BJT capacitors during the simulation. The default is 0.

Syntax

```
.param HSIMBJTCC = <0 | 1>
```

HSIMBJTCV

Invokes the approximate charge conservation model for the charge stored by the BJT capacitors during the simulation. The default is 0.

Syntax

```
.param HSIMBJTCV = <0 | 1>
```

HSIMBMOS

Enables the Precise Model for MOSFET terminals.

Syntax

```
.param HSIMBMOS = <0 | 1>
```

Description

When set to 1, HSIMBMOS enable the Precise Model on coupling effect between MOSFET drain, source, and bulk terminals. The default value is 0. HSIMBMOS=1 is recommended for flash memory circuits.

HSIMBSIM3ISUB

HSIMBSIM3ISUB is used to simulate accurate substrate current with the BSIM3 model. HSIM does not account for this current component without setting the parameter.

Syntax

```
.param HSIMBSIM3ISUB =model_name
```

Description

The syntax accepts the model name or a wildcard. For multiple models, the parameter can be used multiple times. For example:

```
HSIMDBSIM3ISUB = nch*
HSIMDBSIM3ISUB = pch*
```

HSIMBUSDELIMITER

Specifies the bus notation used in a SPICE netlist.

Syntax

```
.param HSIMBUSDELIMITER =bus_notation
```

Description

HSIMBUSDELIMITER is a global parameter with a default of null.

HSIMBUSDELIMITER specifies the bus notation used in SPICE netlist and is referenced in 2 locations:

- hdl testbench interface: Bus signal name in a testbench module is broken down into bit signal names with this bus notation so that HSIM can find the signal connection in the SPICE netlist.
- SPEF: If SPEF is specified, the parser converts all bus names in SPEF input so that they can be matched in HSIM database for back annotation.

Examples

If a bus is named A<0:3>, then applying .param HSIMBUSDELIMITER=<> becomes:

```
A<0>, A<1>, A<2> & A<3>
```

HSIMCAPCC

Invokes the exact charge conservation model for the charge stored by a capacitor during the simulation. The default is 0.

Syntax

```
.param HSIMCAPCC = <0 | 1>
```

HSIMCAPFILE

Controls adding capacitance to previously back-annotated nodes.

Syntax

```
.param HSIMCAPFILE = node_name cap_value <otc|occ>
```

Argument	Description
<i>node_name</i>	Either a node name or a node pattern containing the asterisk (*) wildcard character.
<i>cap_value</i>	Capacitance value.
otc	Keyword to overwrite total capacitance.
occ	Keyword to overwrite constant capacitance.

Description

If otc or occ is not specified, the capacitance value *cap_value* is added to any capacitance previously back-annotated to the *node_name*.

Examples

```
A1 10fF  
B1 1.5f occ  
x1.n* 0.6f otc
```

In this example:

- The first syntax statement above adds 10fF to node A1.
- The second syntax statement above overwrites the constant node capacitance of node B1 to 1.5fF.
- The third syntax statement above overwrites the total node capacitance of all nodes that match with x1.n* to 0.6fF.

HSIMCC

Enables the charge-conservation model for MOSFETs.

Syntax

```
.param HSIMCC = <0 | 1>
```

Description

When set to 1, HSIMCC, enables the charge-conservation model for MOSFETs in the circuit. Similar to \$AMOS, \$CC is set to an individual MOSFET and enables the charge-conservation model for the MOSFET if \$CC=1 is appended at the end of a MOSFET statement. The charge-conservation model is recommended for a charge-pump circuit or A/D and D/A converters.

Other than [HSIMAMOS on page 60](#) and [HSIMCC on page 72](#), control parameters [HSIMSPICE on page 170](#) and [HSIMGCSC on page 91](#) also determine the MOSFET model complexity. All these model control parameters can be locally set such that the setting only affects those MOSFET elements within the subcircuit blocks specified with the setting including:

- [HSIMABMOS on page 52](#)
- [HSIMAMOS on page 60](#)
- [HSIMASIM on page 63](#)
- [HSIMBMOS on page 70](#)
- [HSIMCC on page 72](#)
- [HSIMGCSC on page 91](#)
- [HSIMSPICE on page 170](#)

Device model control parameters are shown in [Table 2](#)

Table 2 Device Model Control Parameters

Parameter	Type	Default	Location(s)
\$AMOS	Boolean	0	MOSFET Element
\$CC	Boolean	0	MOSFET Element
\$SPICE	Int	0	MOSFET Element

Examples

```
M1 drain gate source bulk mos_model w=0.3u l=0.13u $AMOS=1
M2 dr     ga    sor      sub model2      w=0.6u l=0.18u $CC=1
```

HSIMCHECKMOSBULK

Performs a sanity check to identify potential forward bias conditions in MOSFET parasitic diodes.

Syntax

```
.param HSIMCHECKMOSBULK = <0 | 1>
```

Description

HSIM reports warnings when the bulk of a NMOSFET transistor can potentially be greater than 0.5v or the bulk of a PMOSFET transistor can potentially be smaller than 0.5v. The bulk node of the MOSFET transistor has to be connected to a node to ground voltage source element. HSIM uses the DC voltage value of a DC voltage source or the maximum and minimum voltage of a variable voltage source to determine whether forward bias condition can occur.

HSIMCMIN

Sets the minimum capacitance value.

Syntax

```
.param HSIMCMIN =cap_value
```

Description

HSIMCMIN sets the minimum capacitor value allowed in simulation. All capacitors with values smaller than HSIMCMIN are ignored. This command is used to eliminate small capacitors. If the netlist file or DSPEF/SPEF file contains negative capacitors, HSIMCMIN needs to be set to a negative value.

HSIMCOILIB

Specifies the full path of the waveform file generator library.

Syntax

```
.param HSIMCOILIB=path
```

Description

The full path should include the library name. `HSIMCOILIB` is optional. If there is no `HSIMCOILIB` specified, HSIM searches for the `lib<out_format>.so` in the following directories, in the order shown:

1. Run directory
2. `$HOME` directory
3. `$HSIM_HOME/platform/<port>/bin`
4. `LD_LIBRARY_PATH` on Solaris and on Linux, and `SHLIB_PATH` on HP where `<output_format>` is the value of [HSIMOUTPUT on page 119](#). All the letters in `<output_format>` have to be in upper case.

Examples

To generate a `MYFORMAT` format, use the following two HSIM commands:

```
.param HSIMOUTPUT=MYFORMAT  
.param HSIMCOILIB=/usr/local/lib/libMYFORMAT.so
```

HSIMCONNCHECK

Controls connectivity checking.

Syntax

```
.param HSIMCONNCHECK = <0 | 1 | 1.1 | 2>
```

Argument	Description
0	When <code>HSIMCONNCHECK=0</code> , no connectivity check is performed.
1	When <code>HSIMCONNCHECK=1</code> , the default value, a connectivity check prints a warning message if connectivity errors are detected. In this setting, the simulation continues if there are errors.
1.1	When <code>HSIMCONNCHECK=1.1</code> , a more thorough check is performed and the simulation takes longer to finish.
2	When <code>HSIMCONNCHECK=2</code> , any error messages are printed and simulation stops if connectivity errors are detected.

HSIMDCI

Controls DC convergence checking.

Syntax

```
.param HSIMDCI = value
```

Description

During DC convergence, a convergent state is achieved if, for each DC iteration, the current flowing into any net changes by less than HSIMDCI. By default, HSIM only checks against HSIMDCV. HSIMDCI is only checked if HSIMIDDO is enabled. The default value is 5E-7. Units are in Amperes.

HSIMDCINIT

Controls the initialization of DC analysis.

Syntax

```
.param HSIMDCINIT = <0 | 1>
```

Argument	Description
0	<p>Use this argument for situations when there is no need for DC initialization. When HSIMDCINIT is set to 0, DC initialization is suppressed and the transient simulation starts immediately following the preprocessing. Under this condition, the DC initialization proceeds as if running the transient simulation using a zero initial state for each node voltage, unless the node has an initial condition setting through the use of the .ic command.</p> <p>Note that you should not set HSIMDCINIT=0 when simulating a design with instantiated DC voltage sources.</p>
1	<p>When HSIMDCINIT is activated by default or is set to 1, the DC initialization is terminated when a convergent state is reached, or when the number of iteration steps exceeds the limit specified by HSIMDCITER on page 77.</p>

Description

The [HSIMDCINIT](#) command is used to activate DC initialization. DC initialization is activated after the netlist processing and hierarchical database building phase is completed and before the beginning of a transient simulation.

A convergent state is achieved if each node voltage change in an iteration step is less than the voltage specified by [HSIMDCV](#) on page 79, and each net current is less than the current specified by [HSIMDCI](#) on page 76. These commands can be configured to create levels of precision for DC operations needed in their specific circuits.

HSIMDCITER

Limits the number of DC iteration steps.

Syntax

```
.param HSIMDCITER = value
```

Description

When [HSIMDCINIT](#) on page 76 is activated by default or is set to 1, the DC initialization is terminated when a convergent state is reached, or when the number of iteration steps exceeds the limit specified by HSIMDCITER. A convergent state is achieved if each node voltage change in an iteration step is less than the voltage specified by [HSIMDCV](#) on page 79, and each net current is less than the current specified by [HSIMDCI](#) on page 76.

HSIMDCOUTFMT

Sets the DC output format.

Syntax

```
.param HSIMDCOUTFMT = <0 | 2 | 4 | 7>
```

Argument	Description
0	The default output format is ASCII table. Results are written to the <project>.dc file.

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMDCSOIACC

Argument	Description
2	This output format is ASCII raw file. Results are written to the <project>.dc.raw file.
4	This format is similar to HSIMDCOUTFMT=2, however if there is an external sweep in DC analysis, the results for each value of the external sweep go to a separate external file such as the following: <project>.dc.raw.s1, <project>.dc.raw.s2, and so on.
7	This option is used for the WDF output format. Use this option with HSIMOUTPUT =wdf to allow the DC results to be printed in WDF format.

Description

Currently, HSIM only supports the WDF format as shown in the following example.

Examples

```
.param HSIMDCOUTFMT=7  
.param HSIMOUTPUT=wdf
```

HSIMDCSOIACC

Controls whether the BSIM3SOI model is used with the self-heating turned on or off during DC initialization. To turn self-heating off, set HSIMDCSOIACC to 0. To turn it on, set the value to 1.

Syntax

```
.param HSIMDCSOIACC = <0 | 1>
```

HSIMDCSTEP

Sets the DC initialization time step. Determines the smallest time step used in DC initialization. The default is 1000. Units are in picoseconds.

Syntax

```
.param HSIMDCSTEP = value
```

HSIMDCSTOPAT

Specifies the iteration number at which to stop DC analysis.

Syntax

```
.param HSIMDCSTOPAT = value
```

HSIMDCV

Controls the DC convergent state threshold for each DC iteration. During DC convergence, a convergent state is achieved if for each DC iteration a node voltage change is less than HSIMDCV. The default value is 0.001. Units are in Volts.

Syntax

```
.param HSIMDCV = value
```

HSIMDELAYNTLRMIN

Removes resistors with values smaller than the HSIMNTLRMIN value.

Syntax

```
.param HSIMDELAYNTLRMIN = <0 | 1>
```

Description

HSIMDELAYNTLRMIN removes resistors with values smaller than [HSIMNTLRMIN on page 117](#). If a DSPF/SPEF file is specified, backannotation is performed.

When HSIMDELAYNTLRMIN=1, shorting the resistors according to [HSIMNTLRMIN on page 117](#) is delayed until after backannotation.

Note: If you use the backannotation flow and set a larger HSIMRMIN/HSIMVSRCRMIN/HSIMNTLRMIN value than some schematic resistances, and do not set HSIMRCRKEPELEM=1 or HSIMDELAYNTLRMIN=1, HSIM might generate the wrong

connectivity during the backannotation process and cause a critical circuit function error. To prevent this problem, set HSIMDELAYNLRMIN=1.

HSIMDELVTO

Specifies flash core transistor usage.

Syntax

```
.param HSIMDELVTO = devto_valuesubckt = subcircuit_name
```

Description

HSIMDELVTO sets DELVTO for individual transistors or those in subcircuits so the transistor can be used on flash core cells as shown in the following example.

Note: Previously, HSIMDELVTO was allowed on instances only but is now extended to subcircuits.

Examples

```
.param HSIMDELVTO=0.1 subckt=core_sub
```

This example sets DELVTO to 0.1 for all transistors in the core_sub subcircuit.

HSIMDETAILBSIM4

Provides the RGATEMOD model parameter with good accuracy. Specify HSIMDETAILBSIM4 = 1 only for the BSIM4 model to provide the RGATEMOD model parameter with good accuracy. The default is HSIMDETAILBSIM4=0.

Syntax

```
.param HSIMDETAILBSIM4 = <0 | 1>
```

HSIMDIOCC

Invokes the exact charge conservation model for the charge stored by the diode capacitors during the simulation. The default is 0.

Syntax

```
.param HSIMDIOCC = <0 | 1>
```

HSIMDIOCV

Invokes the approximate charge conservation model for the charge stored by the diode capacitors during the simulation. The default is 0.

Syntax

```
.param HSIMDIOCV = <0 | 1>
```

HSIMDIODECURRENT

Calculates the dc current of MOSFET diodes.

Syntax

```
.param HSIMDDIODECURRENT = <0 | 1 | 2>
```

Description

The default of 0 does not calculate the DC current of MOSFET parasitic diodes.

When set to 1, this command calculates the DC current of MOSFET parasitic diodes. In addition, set the HSIMBMOS parameter to ensure a correct partition to calculate current calculation effect.

When set to 2, this command also calculates the DC current of MOSFET parasitic diodes. If the HSIMABMOS parameter is also set, there is much greater performance penalty when HSIMDIODECURRENT is set to 2.

HSIMDIODEVT

Calculates the junction diode voltage.

Syntax

```
.param HSIMDIODEVT = voltage
```

Description

In CMOS circuitry, MOSFET junction diodes are usually in the reverse bias region and diode voltage can be ignored. If a MOSFET junction diode becomes forward biased during simulation, and its voltage drop is bigger than `HSIMDIODEVT`, and MOSFET drain, SOURCE, and BULK are in the same partition, HSIM calculates the junction diode voltage. The default voltage is 0.5

HSIMDMR

Enables multiple time-step selection (multi-rate) within individual large, hierarchical partitions. Target applications for `HSIMDMR` include postlayout memory circuits such as SRAM, CAM, and ROM.

Syntax

```
.param HSIMDMR = <0 | 1>
```

Argument	Description
0	All internal portions of hierarchical partitions take uniform time-steps (default).
1	Enable multi-rate within hierarchical partitions.

HSIMDPF

Defines the DPF file for device parameter backannotation.

Syntax

```
.param HSIMDPF = DPF_file
```

Description

The advantage of DPF backannotation is that the prelayout hierarchy is maintained for simulation.

For MOSFET devices, the supported DPF parameters are: L, W, AD, AS, PD, PS, NRD, NRS, SA, SB, SD, NF, DELVTO, MULU0, RGEOMOD, RDC, RSC, SCA, SCB, SCC, SA1, SA2, SA3, SA4, SA5, SA6, SA7, SA8, SA9, SA10, SB1, SB2, SB3, SB4, SB5, SB6, SB7, SB8, SB9, SB10, SW1, SW2, SW3, SW4, SW5, SW6, SW7, SW8, SW9, SW10.

HSIMDPFHIERID

Specifies the hierarchical separator used in DPF file. If the hierarchical separator used in DPF file is different from HSIMDPFHIERID, the hierarchical separator must be specified with HSIMDPFHIERID. The default character is a period (".").

Syntax

```
.param HSIMDPFHIERID = separator_character
```

HSIMDPFMULTISUB

Controls how to read multiple subcircuits in a DPF file.

Syntax

```
.param HSIMDPFMULTISUB = <0 | 1>
```

Argument	Description
0	Does not read multiple subcircuits in a DPF file (default).
1	Allows reading multiple subcircuits in a single DPF file.

HSIMDPFPFX

Specifies the DPF prefix string.

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMDPFREPORTNOBA

Syntax

```
.param HSIMDPFPFX = prefix_character
```

Description

The device name in the DPF file usually comes in two types. The extractor may reverse the hierarchy or prepend a string with leading M to the device name. For example, in pre-layout netlist, A device named x1.x2.mp1 can become M_x1.x2.mp1 or mp1.x2.x1 in the DPF file.

HSIM automatically checks for reverse hierarchy in device name. But if the name is prepended with a prefix string, you need to specify that prefix string using `HSIMDPFPFX` so HSIM can correctly extract the hierarchical name. The default prefix string is `m_`.

HSIMDPFREPORTNOBA

Controls whether to generate a report for input netlist devices that are not back-annotated. Checks and reports any input netlist devices that are not DPF back-annotated. The report is stored in an `output_filename.dpfnoba` file.

Syntax

```
.param HSIMDPFREPORTNOBA = <0 | 1>
```

Argument	Description
0	Does not generate a report for input netlist devices not back-annotated (default).
1	Generates a report for input netlist devices not back-annotated.

HSIMDPFSCALE

Specifies the scale factor for device parameters in the DPF file. For example, if the unit used in the DPF file is micron, then you need to specify `HSIMDPFSCALE=1u`.

Syntax

```
.param HSIMSCALE = unit
```

HSIMDPFSFX

Specifies delimiter the delimiter character used on fingered devices in the DPF file. The default character is an asterisk ("@").

Syntax

```
.param HSIMDPFSFX = suffix_character
```

HSIMDPFSPLITFD

Back-annotates the parameter values for finger devices.

Syntax

```
.param HSIMDPFSPLITDF = <0 | 1>
```

Argument	Description
0	Does not back-annotate the parameter values for each individual finger device (default).
1	Back-annotate the parameter values for each individual finger device.

Description

When HSIM encounters finger devices during DPF annotation, the device parameter values for each finger device group are averaged during the backannotation process. In order to back-annotate the parameter values for each individual finger device, `HSIMDPFSPLITFD` must be set to 1 (default is 0).

HSIMDTNAME

Forces HSIM to recognize the specified terminal names for a particular device type.

Syntax

```
.param HSIMDTNAME = "device_type:term1, term2, term3, . . . "
```

Description

This parameter is used when uncommon device terminal names are specified in the post-layout netlists. It provides a mechanism to force HSIM to recognize the specified terminal names for a particular device type.

Examples

In this example, device_type can be i,v,m,q,r,c,d, and l. The number of terminals specified must correspond to the number of terminals associated with the particular device type.

To use tm1, tm2, tm3, and tm4 to denote terminals of a MOS device in the post-layout file (for example, m1:tm2), specify:

```
.param HSIMDTNAME="m:tm1,tm2,tm3,tm4"
```

HSIMDUMPOPI

Dumps element currents to an operating point. The default is 0.

Syntax

```
.param HSIMDUMPOPI = <0 | 1>
```

HSIMEFFICIENTTBL

Accounts for STILOD/WPE effects of pre-layout or post-layout simulation.

Syntax

```
.param HSIMEFFICIENTTBL = <0 | 1>
```

Description

Set `HSIMEFFICIENTTBL = 1` with the instance parameters SA, SB, SD, SCA, SCB, SCC, SC in the netlist to account for STILOD/WPE effects of pre-layout or post-layout simulation to minimize simulation memory with BSIM4 table models. This command dynamically updates the table values by considering these parameters during simulation while maintaining accuracy.

This command applies only to table models. However, if you set `HSIMSPICE=1` or higher, that the table models are turned off and direct modeling is used.

HSIMENHANCECAP

Applies an internal table to capacitance values from BSIM equations. For the BSIM4 model, the default setting is equivalent to `HSIMENHANCECAP=1`.

Syntax

```
.param HSIMENHANCEDCAP = <0 | 1 | 2>
```

Argument	Description
0	Calculates the capacitance values of a MOSFET transistor with an efficient formula (default).
1	Applies a compact internal table to calculate capacitance values from the associated BSIM equations.
2	Uses a more extensive internal table to calculate capacitance values from detailed BSIM equations. For the BSIM4 model, the default setting is equivalent to <code>HSIMENHANCECAP=1</code> .

HSIMENHANCEDC

Controls DC initialization.

Syntax

```
.param HSIMENHANCEDC = <0 | 1 | 2 | 3>
```

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMENHANCEDIDQ

Argument	Description
0	Does not manually specify a DC initialization algorithm (the default).
1	Uses a more conservative DC initialization algorithm and settings and increases the DC iteration limit to 1000.
2	Utilizes the voltage-dependent MOSFET capacitance model in DC initialization.
3	Opens the potential for performance tradeoffs to provide a more conservative approach targeted at achieving better DC convergence results.

HSIMENHANCEDIDQ

Speeds up the simulation by setting a pre-defined time slot using a PWC function.

Syntax

```
.param HSIMENHANCEDIDQ =pwc_function_syntax
```

Description

HSIMENHANCEDIDQ causes HSIM to run very slowly during transient simulation. To speed up the simulation, set a pre-defined time slot using a PWC function in IDDQ current evaluation. The following example turns on the IDDQ current evaluation during a window defined by the 20n, 30n parameters.

Examples

```
.param HSIMENHANCEDIDQ=pwc(On, 0, 20n, 2, 30n, 0)
```

In addition to setting HSIMENHANCEDIDQ on user-selected subcircuits, this command can also be activated during some simulation periods using PWC syntax. This flexibility of spatial and temporal specifications can be used to analyze the subcircuit and the time period where accurate device model and leakage current computations are needed and help reduce CPU time for quiescent or leaking current measurements.

```
.hsimparam subckt=acu_blk HSIMENHANCEDIDQ=pwc(0.0n,0,100n,1)
```

The syntax in this example sets enhanced IDDQ on the acu_blk block to 1 after time=100n, causing the simulation to proceed as usual before 100n. After 100n, the most accurate model and evaluation is employed for leakage current, device model computation, and so on.

HSIMENHANCEMOSIV

Calculates static drain current.

Syntax

```
.param HSIMENHANCEMOSIV = <0 | 1>
```

Description

When HSIMENHANCEMOSIV = 0, HSIM applies a compact internal table to calculate the static drain current value of a MOSFET transistor. When HSIMENHANCEMOSIV = 1, it uses a more extensive internal table to calculate the drain current value in order to capture the fine detail of the device characteristics.

Note: The default grid number in the Vbs dimension is 6.

HSIMFLAT

Selects a flat or a hierarchical simulation.

Syntax

```
.param HSIMFLAT = <0 | 1>
```

Description

In simulating subcircuit cells, the efficiency gained by saving identical subcircuit computations is significant enough to compensate for the overhead. If each subcircuit cell has a different behavior during the simulation, then the flat simulation might have an efficiency advantage over hierarchical simulation. Hierarchical simulation still has an advantage in saving the storage space. To allow a trade-off choice in memory usage and simulation speed, the HSIMFLAT command enables either a flat or a hierarchical simulation.

When HSIMFLAT=1, HSIM does not flatten the entire circuit; it only selectively flattens those subcircuits with no sufficient subcircuit instances, or the

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMFMODLIB

subcircuit that has a different behavior from other subcircuit instances. The default is 0.

HSIMFMODLIB

Specifies the library path to the netlist.

Syntax

```
.param HSIMFMODLIB = lib_file_name
```

HSIMFORWARDBIASDIODE

Applies internal tables to calculate static drain.

Syntax

```
.param HSIMFORWARDBIASDIODE = <0 | 1>
```

Argument	Description
0	Applies a compact internal table to calculate the static drain current value of a MOSFET transistor (the default).
1	Uses a more extensive internal table to calculate the drain current value in order to capture the fine detail of the device characteristics. The default grid number in the Vbs dimension is 6.

HSIMFSDBDOUBLE

Set `HSIMFSDBDOUBLE = 1` to store signal values in double precision for the FSDB output format. The default is 0, which stores signal values as floating point.

Syntax

```
.param HSIMFSDBDOUBLE = <0 | 1>
```

HSIMGATEMOD

Set `HSIMGATEMOD=1` to model the constant gate electrode resistance. The default is 0.

Syntax

```
.param HSIMGATEMOD = <0 | 1>
```

HSIMGCSC

Controls the calculation of MOSFET gate and junction diode capacitances.

Syntax

```
.param HSIMGCSC = <0 | 1>
```

Argument	Description
0	<p>Simplifies the voltage-dependent MOSFET capacitances to average capacitances and ignores the Miller Effect between gate-to-drain and gate-to-source terminals. This simplification typically achieves a 2X speedup but may lose some precision, especially for circuits operating at high frequencies in which the timing delay is sensitive to how the MOSFET capacitances are handled.</p> <p>This value is recommended for low-frequency applications or high-speed functionality simulation where 5-10% precision loss can be tolerated.</p>
1	<p>This default value specifies the most precise mode, which you can set at various levels of the circuit hierarchy. When <code>HSIMSPEED = 8</code> and <code>HSIMGCSC</code> are set, the simulation time steps are controlled conservatively. If <code>HSIMGCSC</code> is set at local circuits, only these circuit parts are affected by the conservative time step selection.</p>

Description

The MOSFET gate and junction diode capacitances are voltage-dependent. This requires calculation for each capacitance value within the MOSFET at every time step. In addition, the Miller Effect due to the coupling capacitors

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMGMIN**

between gate-to-drain and gate-to-source terminals requires more computations and slows down the simulation speed.

HSIMGMIN

Specifies the minimum conductance added to all PN junctions for a time sweep in transient analysis.

Syntax

```
.param HSIMGMIN = value
```

Argument	Description
value	Specifies the conductance value. The default is 1e-12.

Description

Note that if you set `.option gmin` in the netlist, the `HSIMGMIN` command overrides that value.

HSIMGMINDC

Specifies conductance in parallel for PN junctions and MOSFET nodes in DC analysis.

Syntax

```
.param HSIMGMINDC = value
```

Argument	Description
value	Specifies the conductance value.

Description

Use this option to specify conductance in parallel for all PN junctions and MOSFET nodes except gates in DC analysis. `HSIMGMINDC` helps overcome DC convergence problems caused by low values of off-conductance for pn junctions and MOSFETs.

Note that if you set `.option gmindc` in the netlist, the `HSIMGMINDC` command overrides that value.

HSIMGMIWELM

Supports the HSPICE®-compatible W-element of the RLGC format.

Syntax

```
.param HSIMGMIWELM = <0 | 1>
```

Description

HSIM supports the HSPICE compatible W-element of the RLGC format in model or file formats with the same syntax. To invoke this option, use HSIMGMIWELM=1. The default value (0) uses the HSIM implementation.

HSIMHASCOMPLEXBJT

Controls simulation when using a complex BJT model, such as VBIC or Mextram.

Syntax

```
.param HSIMHASCOMPLEXBJT = <0 | 1 | 2>
```

Argument	Description
0	Sets HSIMTIMESCALE and HSIMALLOWEDDV to their default values.
1	Sets HSIMTIMESCALE to 0.001 and HSIMALLOWEDDV to 0.1, unless smaller values are explicitly specified for subcircuits containing VBIC or Mextram devices.
2	Tightens simulation to be more synchronous, in addition to tightening up HSIMTIMESCALE on page 177 and HSIMALLOWEDDV on page 59 .

Description

When a complex BJT model such as VBIC or Mextram is used in a design, it is recommended to set HSIMHASCOMPLEXBJT = 1 or =2. When a complex BJT model such as VBIC or Mextram is used in a design, it is recommended to set HSIMHASCOMPLEXBJT=1 or =2.

HSIMHIERID

Specifies the hierarchical separator character.

Syntax

```
.param HSIMHIERID = separation_character
```

Description

HSIMHIERID specifies the hierarchical separator character. The following example illustrates the use of colon (:) as the hierarchical separator.

```
.param HSIMHIERID=':'
```

The default is the period character (.).

HSIMHIEROPT

Optimizes performance and accuracy locally.

Syntax

```
.param HSIMHIEROPT = <0 | 1>
```

Description

Similar to [HSIMASIM on page 63](#), HSIMHIEROPT optimizes the performance and accuracy by using a conservative analog simulation algorithm. The main difference is that this is a local command, while HSIMASIM is a global command.

Specify a value of 1 to use the more conservative algorithm. The default is 0.

HSIMHONORX

Determines command precedence for HSIM commands specified with [HSIMX on page 200](#) or the hsim -x command line option and in the netlist.

Syntax

```
.param HSIMHONORX = <0 | 1>
```

Argument	Description
0	Specifies that the settings in the netlist take precedence over the settings in HSIMX on page 200 or the <code>hsim -x</code> command line option.
1	Specifies that the settings in HSIMX on page 200 or the <code>hsim -x</code> command line option take precedence over the settings in the netlist. This is the default value.

HSIMHSPICEVEC

Creates a compatible format between HSPICE® and HSIM to handle the slight difference in formats for the bus notation of vector files.

Syntax

```
.param HSIMHSPICEVEC = <0|1>
```

Description

Suppose the bus notation in a vector file is:

```
vname ck[1:3]
```

When `HSIMHSPICEVEC = 0` (the default), this bus is recognized as `ck[1]` `ck[2]` `ck[3]`. When `HSIMHSPICEVEC = 1`, this bus is recognized as `ck1` `ck2` `ck3`. It is compatible with HSPICE.

HSIMHZ

Performs Hi-Z node checking.

Syntax

```
.param HSIMHZ = <0|1>
```

Description

When `HSIMHZ = 1`, HSIM performs Hi-Z node checking. If the Hi-Z check detects a Hi-Z node on one subcircuit instance, it is possible that there are more Hi-Z nodes on other similar subcircuit instances.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMHZALL**

If the expected state (from vector file) is "x", there is no "DOUT error" triggered regardless of the measured state (from simulation).

If the expected state is not "x", only the following three conditions trigger a "DOUT error" :

- expect = 1 state = 0
- expect = 0 state = 1
- expect= (any thing other than "x") state= x

Note: HSIMHZ = 1 reports only one Hi-Z node from all possible Hi-Z nodes on other similar subcircuit instances in order to streamline the Hi-Z node report and improve performance. The default is HSIMHZ = 0.

The HSIMHZ command can generate output in plain format by setting "cckOutput format=plain" in the CCK command file.

HSIMHZALL

Overwrites the HSIMHZ default behavior.

Syntax

```
.param HSIMHZALL = <0 | 1>
```

Description

When HSIMHZALL = 1 and HSIMHZ = 1 are set, all existing Hi-Z nodes found on similar subcircuit instances are reported. All Hi-Z results reported using HSIMHZALL = 1 should match the results achieved when pcheck zstate is run. The default is HSIMHZALL = 0.

The HSIMHZALL command can generate output in plain format by setting "cckOutput format=plain" in the CCK command file.

HSIMHZFANOUT

To avoid unnecessary internal node checks, HSIMHZFANOUT limits high impedance state checks to those driver nodes with fan-outs.

Syntax

```
.param HSIMHZFANOUT = <0 | 1 | 2 | 3>
```

Argument	Description
0	Checks all nodes.
1	Checks the nodes that have a direct connection to a gate of transistors.
2	Checks nodes having a direct connection to a transistor bulk.
3	Checks nodes having a direct connection to a transistor gate, or to a BJT base (the default).

HSIMHZNDNAME

Defines the node name to be checked. Each node name can be the name of a single node or a node name containing an wildcard asterisk (*) character which represents a group of node names.

Syntax

```
.param HSIMHZNDNAME = 'node_name'
```

HSIMHZSTART

Starts the check at the specified start time value. Checks are performed within the time window defined by HSIMHZSTART and HSIMHZSTOP. You can use multiple time windows with multiple HSIMHZSTART and HSIMHZSTOP values.

Syntax

```
.param HSIMHZSTART = start_time
```

HSIMHZSTOP

Stops the check at the specified stop time value. Checks are performed within the time window defined by HSIMHZSTART and HSIMHZSTOP. You can use multiple time windows with multiple HSIMHZSTART and HSIMHZSTOP values.

Syntax

```
.param HSIMHZSTOP = stop_time
```

HSIMHZTIME

Threshold time for a specified node staying in the high-impedance state to be reported.

Syntax

```
.param HSIMHZTIME = time
```

Description

Any specified node staying in the high-impedance state longer than HSIMHZTIME is reported in the *title_name.hz* file.

HSIMHZTIME is specified in nanoseconds. The default is 5.

HSIMHZSUBCKT

Excludes the specified subcircuit from checking. HSIMHZSUBCKT excludes checking the specified subcircuit. To exclude multiple subcircuits, use one HSIMHZSUBCKT for each subcircuit.

Syntax

```
.param HSIMHZSUBCKT = subckt_name
```

HSIMIDDQ

Tightens the simulation accuracy criteria.

Syntax

```
.param HSIMIDDQ = <1 | 2>
```

Argument	Description
1	Increases the DC iteration number for higher quality DC convergence. It also tightens up the precision criterion during transient simulation. Leakage current between is measured in the V_{DD} and G_{ND} nodes. The elements that conduct the current can be either in the strong-inversion region or in the subthreshold region.
2	Further tightens the precision criterion for more enhanced accuracy. Leakage is measured current through the regular or parasitic diode junction.

Description

The steady current state is commonly referred to as the IDDQ current. IDDQ consists of two components: HSIMIDDQ=1 and HSIMIDDQ=2. THSIMIDDQ=1 and HSIMIDDQ=2 provide control commands to tighten up the precision criterion for the IDDQ measurement. To obtain accurate IDDQ measurement results, you must tighten up the simulation accuracy criterion.

HSIMIGISUB

Calculates the static gate leakage currents and the substrate current.

Syntax

```
.param HSIMIGISUB = <0 | 1 | 2 | 3>
```

Argument	Description
0	Default.
1	Includes both the static gate leakage currents and the substrate current in the calculation.
2	Includes the substrate current in the calculation but not the gate leakage currents.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMINACTOPT**

Argument	Description
3	Includes the gate leakage currents in the calculation but not the substrate current.

Description

In order to minimize the memory usage and the computation time, by default, the static gate leakage currents and the substrate current are not included in the calculation of I-V result for a MOSFET transistor with the BSIM4 model. You can use the `HSIMIGISUB` command to specify the calculation.

HSIMINACTOPT

Controls how to generate inactive and active list files. The active .ach list file is always created with .acheck command. The inactive list file .ina or .ina_all can be selectively created. Use the `HSIMINACTOPT` command to control how to generate the files.

Syntax

```
.param HSIMINACTOPT = <1 | 2 | 3>
```

Argument	Description
1	Generates the entire inactive list file .ina_all.
2	Generates the surrounding inactive list file .ina (the default).
3	Generates both the .ina_all and .ina files.

HSIMIMPORT

Improves performance when probing devices and currents from voltage sources.

Syntax

```
.param HSIMIMPORT = <0 | 1>
```

Argument	Description
0	Turns HSIMIPORT off (the default).
1	Enables the HSIMIPORT current collection method so that HSIM does not perform additional device evaluations for devices connected to voltage sources.

Description

The HSIMIPORT command eliminates the need for additional device evaluations required to obtain the correct current values drawn by the elements whose currents are being probed. It provides better probing performance with a new method of collecting device currents without additional device (MOS as well as non-MOS) evaluations.

HSIMIPRECISION

Controls current precision. For details about the HSIMIPRECISION settings, see [HSIM Quick Start on page 12](#).

Syntax

```
.param HSIMIPRECISION = <0 | 1 | 2 | 3 | 4>
```

Description

HSIMIPRECISION controls current precision. Similar to VPRECISION for voltage, HSIMIPRECISION defines current thresholds considered to be negligible during simulation.

A value of 0 specifies the fastest performance, while a value of 5 specifies the most precision. The default is 2.

The HSIMIPRECISION command controls:

- Isomorphic matching based on port current matching (see [HSIMPORTI on page 125](#)).
- Use of latency (see [HSIMSNCS on page 148](#)).
- Sub-threshold and low Vds currents computation (see [HSIMIDDQ on page 98](#)).
- Current printing precision (see [HSIMOUTPUTIRES on page 121](#)).

HSIMITERMODE

Enables the Newton Raphson iteration scheme. The Newton Raphson iteration scheme is available in HSIM to provide a high level of accuracy using the iteration algorithm. The trade off is that using this algorithm might cause impact performance.

Syntax

```
.param HSIMITERMODE = <0 | 1 | 2>
```

Argument	Description
0	Does not enable this mode (the default).
1	Uses the HSIM proprietary iteration scheme, which balances accuracy and performance.
2	Uses SPICE-like Newton Raphson iteration.

HSIMKEEPALLGNDCAPS

Specifies whether capacitors connected to a zero constant voltage source are lumped to the signal node.

Syntax

```
.param HSIMKEEPALLGNDCAPS = <0 | 1>
```

Description

Specifies whether or not to keep capacitors connected to a zero constant voltage source. Set `HSIMKEEPALLGNDCAPS=1` to keep all ground capacitors independent of the [HSIMLUMPCAP on page 104](#) setting. You can also specify `HSIMLUMPCAP=0` with `HSIMKEEPALLGNDCAPS` to specify whether (`HSIMKEEPALLGNDCAPS=0`) or not (`HSIMKEEPALLGNDCAPS=1`) capacitors connected to a zero constant voltage source are lumped to the signal node. See [HSIMLUMPCAP on page 104](#) for more details.

HSIMKEEPNODESET

Keeps nodeset values.

Syntax

```
.param HSIMKEEPNODESET = <0 | 1>
```

Description

Ensures that all nodeset values are kept during the first 1/10 of the total iterations. HSIM automatically identifies cross-coupled nodes in circuits and applies a logic 1 nodeset value to one of the nodes. This avoids a meta stable condition after DC initialization. Since the nodeset value is kept only at the first iteration, it can be overridden. The default is 0.

HSIMLMIN

Sets the minimum inductor value.

Syntax

```
.param HSIMLMIN = value
```

Description

Sets the minimum inductor value allowed in simulation. All inductors with a value smaller than HSIMLMIN are short-circuited. This command is used at the netlist parser of HSIM to eliminate small inductors. It can only be set globally. The default is 1E-11.

HSIMLIS

Concatenates input files in a single file. When HSIMLIS = 1, HSIM concatenates all input files into a single file.

Syntax

```
.param HSIMLIS = <0 | 1>
```

HSIMLOGDCPROGRATE

Controls the DC simulation progress status reporting. HSIMLOGDCPROGRATE controls the DC simulation status report. DC simulation status is reported when 10% of the total iteration is completed. The default is 0.1.

Syntax

```
.param HSIMLOGDCPROGRATE = progress_rate_value
```

HSIMLOGPROGRATE

Controls the DC simulation progress status reporting. HSIMLOGPROGRATE controls the transient simulation status report. Transient simulation status is reported when 10% of the total iteration is completed. The default is 0.1.

Syntax

```
.param HSIMLOGPROGRATE = progress_rate_value
```

HSIMLUMPCAP

Keeps all the capacitors as regular elements.

Syntax

```
.param HSIMLUMPCAP = <0 | 1 | 2>
```

Argument	Description
0	If HSIMLUMPCAP=0 and HSIMKEEPALLGNDCAPS=0, only the capacitors connected to the zero constant voltage source are lumped to the signal node. The capacitors connected to the non-zero constant voltage source are not lumped to the signal node. If HSIMLUMPCAP=0 and HSIMKEEPALLGNDCAPS=1, capacitors connected to either zero or non-zero voltage sources are not lumped to the signal nodes. The default value of HSIMKEEPALLGNDCAPS is 0. Use HSIMLUMPCAP=0 only if the simulation requires very high precision on peak current value.
1	HSIMLUMPCAP=1 is the default. All capacitors from signal nodes to constant voltage source nodes are lumped to the signal nodes. The default value is suitable for most applications.
2	HSIMLUMPCAP=2 keeps the gnd capacitors for backannotation and removes them after backannotation.

Description

Use the HSIMLUMPCAP command to keep all the capacitors as regular elements. This control command can be a global setting or local setting.

Note: This command can be expensive in terms of CPU time and memory usage.

HSIMMATCHOPT

Optimizes matching in transient analysis.

Syntax

.param HSIMMATCHOPT = <0 | 1 | 2>

Argument	Description
0	Turns optimization off (the default).
1	Enables optimization for prelayout simulations.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMMATRIXOPT**

Argument	Description
2	Enables optimization for postlayout simulations.

Description

HSIM uses hierarchical storage and isomorphic matching to improve performance and capacity. During transient analysis, if you set HSIMMATCHOPT to 1 or 2, it eliminates unnecessary matching and redundant evaluations to improve performance. The amount of performance gain varies depending on the circuit type, design style, and parasitic content. Input vectors also affect optimization efficiency.

HSIMMATRIXOPT

Optimizes matrices for better performance in transient analysis.

Syntax

```
.param HSIMMATRIXOPT = <0 | 1 | 2>
```

Argument	Description
0	Turns off optimization (default).
1	Detects and skips zero coupling terms in a matrix.
2	Approximates small coupling terms with zeros.

Description

During simulation, many coupling terms in a matrix can become zero or negligible. HSIM detects this situation and optimizes the matrix for better performance. If you set HSIMMATRIXOPT=1, HSIM detects the zero coupling terms in the matrix and does not evaluate them, with no loss of accuracy. If you set HSIMMATRIXOPT=2, HSIM also approximates small coupling terms with zeros at some loss of accuracy.

HSIMMEASOUT

Generates an output file from a .measure command.

Syntax

```
.param HSIMMEASOUT = <0 | 1 | 2>
```

Argument	Description
0	Generates <prefix>.mt<.a#> as the output file, where <prefix> is the output prefix, <.a#> is the alter number if alter is used (default).
1	Generates <prefix>.mt<.#> as the output file, where <prefix> is the output prefix, <.#> is the alter number if alter is used.
2	Generates <prefix>.mt.all as the output file, where <prefix> is the output prefix.

Description

If you specify 0, the format in the file (for example, hsim.mt.a0) is a user-readable format. If you specify 1, the format in the file (for example, hsim.mt.0) is in HSPICE-like (tabular) format. If you specify 2, if alter is used, all data is outputted into the file (for example, hsim.mt.all) in HSPICE tabular format similar to HSIMMEASOUT=1.

HSIMMEASWINDOW

Limits waveform printing of .meas signals to the time window specified in the .print statement.

Syntax

```
.param HSIMMEASWINDOW = <0 | 1>
```

Examples

```
.param HSIMMEASWINDOW=1  
.print window 100u 400u  
.meas tran vpeak peak v(2)
```

When HSIMMEASWINDOW=1, it limits the peak voltage measurement for v(2) by the .print window range.

HSIMMONITORMRES

Monitors and reports the initial state conditions and/or state transition information for MRAM bit cells. HSIM has the ability to monitor and report the initial state conditions and/or state transition information of any MRAM bit cells when using the MRES built-in proprietary MRAM models.

Syntax

```
.param HSIMMONITORMRES = <0 | 1 | 2 | 3>
```

Argument	Description
0	Turns off HSIMMONITORMRES (the default).
1	Reports MRES state transitions to the outputprefix.monitormres file.
2	Same as 1, plus prints a state change message to screen and .log file.
3	Same as 2, plus prints ALL MRES initial states to the outputprefix.icmres file.

HSIMMODELSTAT

Lists model statistics in the log file.

Syntax

```
.param HSIMMODELSTAT = <0 | 1>
```

Description

You can view the number of instances created for each device model by setting HSIMMODELSTAT = 1 If set, a table listing the model statistics in the log file is displayed. HSIMMODELSTAT is turned off by default. A sample statistics table is shown in the following example.

Examples

```
Model Statistics
MOS - n: 10
MOS - p: 25
JFET - njf1: 5
BJT - bjt1: 6
```

HSIMMONTECARLOINST

Prints the device instance parameter variations. Set `HSIMMONTECARLOINST = 1` to print the device instance parameter variations for each iteration. Instance parameter variations are saved in files ending with the `.mip` suffix.

Note: The default behavior is `HSIMMONTECARLOINST = 1`

Syntax

```
.param HSIMMONTECARLOINST = <0 | 1>
```

HSIMMONTECARLOMOD**Syntax**

```
.param HSIMMONTECARLOMOD = <0 | 1>
```

Description

Prints the model parameter variations.

Set `HSIMMONTECARLOMOD = 1` to print the model parameter variation for each iteration.

Model parameter variations are saved in files with the `.mmp` suffix.

The default behavior is `HSIMMONTECARLOMOD = 1`.

HSIMMONTECARLOSAVEOUT

Keeps all Monte Carlo iteration files.

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMMONTECARLOSAVEOUT

Syntax

```
.param HSIMMONTECARLOSAVEOUT = <0 | 1>
```

Description

By default, Monte Carlo analysis only keeps the of the last iteration of following files:

- Waveform
- Measure
- Model Parameter Variation
- Device Instance Parameter Variation

Set HSIMMONTECARLOSAVEOUT = 1 to keep all of the files listed above for each iteration with the .m# suffix.

The final statistical information results of all .measure commands are saved into the .mc file for transient analysis or .dc_mc file for DC analysis. The output contains the following information:

Nominal Result

The measured result with the original value defined in parameter distribution function.

Mean Result

The average of all the measured results of a measure statement after iteration is completed.

Variance

The following formula:

$$\frac{\sum_{i=1}^n (x_i - \bar{x}_{ave})^2}{n}$$

where the following apply:

where the following apply:

- x_i is the measured result of a measure statement at the i th HSIM iteration.
- \bar{x}_{ave} is the Mean Result.
- $i=1,\dots,n$, and n number of HSIM iterations.

Standard Deviation

The square root of the variance.

Minimum Result

The smallest measured result of a measure statement.

Maximum Result

The largest measured result of a measure statement.

Average Deviation

The average deviation is calculated as follows:

$$\frac{\sum_{i=1}^n |x_i - x_{ave}|}{n}$$

nb

The MCA histogram is depicted in 10 bins, evenly divided between the minimum and maximum result. The number per bin is the number of measured results that fall into that value range.

run_max and run_min

The index for the measured files where the maximum and minimum results are stored.

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMMOSPRECISION**Examples***Example 10 HSIM Monte Carlo Example*

```
input:  
.tran 1e-011 10u sweep monte=45  
.param HSIMMONTECARLO=1  
.param mlen1=gauss(2e-005, .5, 3)  
.param mlen2=agauss(1.47e-006, 1e-012, 3)  
.param mwidth1=unif(2e-005, .5)  
.param mwidth2=aunif(1.47e-006, 1e-012)  
.meas vx1vcococontrol find v(x1.vcoctrl) at=9.5u  
.output:  
meas_variable=vx1vcococontrol  
nominal=2.480476e-001  
mean=2.481729e-001 varian=4.461293e-005  
stddev=6.679291e-003 avgdev=5.256493e-003  
min=2.336792e-001 max=2.669093e-001  
run_min=1 run_max=14  
2.370022e-001, nb=1, freq=2.222222e-002 | *  
2.403252e-001, nb=5, freq=1.111111e-001 | *****  
2.436482e-001, nb=5, freq=1.111111e-001 | *****  
2.469712e-001, nb=8, freq=1.777778e-001 | *****  
2.502943e-001, nb=12, freq=2.666667e-001 | *****  
2.536173e-001, nb=6, freq=1.333333e-001 | *****  
2.569403e-001, nb=5, freq=1.111111e-001 | *****  
2.602633e-001, nb=1, freq=2.222222e-002 | *  
2.635863e-001, nb=1, freq=2.222222e-002 | *  
2.669093e-001, nb=1, freq=2.222222e-002 | *
```

HSIMMOSPRECISION

Controls MOSFET small signal approximations. For details about the **HSIMMOSPRECISION** settings, see [HSIM Quick Start on page 12](#).

Syntax

```
.param HSIMMOSPRECISION = <0|1|2|3|4|5>
```

Description

HSIMMOSPRECISION is a macro command. It controls MOSFET small signal approximations that are available to HSIM. Simulators apply differing degrees of approximations to increase performance. For example, some digital designs can be simulated with enough precision using average capacitance models,

whereas sensitive analog designs might require voltage dependent capacitance modeling and charge conservation.

A value of 0 specifies to use table look-up models for MOSFET devices for faster simulations, while a value of 5 specifies to use precise analytical equations for a more accurate simulation. The default is 0. The following commands also affect the HSIMMOSPRECISION command:

- [HSIMDETAILBSIM4 on page 80](#)
- [HSIMDIODECURRENT on page 81](#)
- [HSIMSPICE on page 170](#)

HSIMMQS

Controls dynamic time step adjustment during transient simulation.

Syntax

```
.param HSIMMQS = <0 | 1 | 2>
```

Argument	Description
0	Specifies the most conservative selection. This option specifies a uniform time step for every node at any time.
1	Specifies the step size at each node, depending on its time constant, and can be multiple times of the minimal step size at that time (default).
2	Results in larger time steps for slow subcircuits, which may improve simulation speed. Specify this setting when HSIMSPEED on page 150 is less than or equal to 5 for ASIC or standard-cell designs, digital designs, and other designs that have simple subcircuits with channel-connected blocks (CCBs).

Description

During the transient simulation, the time step is adjusted dynamically such that the voltage change over the time step is limited to a predefined voltage value, defined by [HSIMALLOWEDDV on page 59](#). The time constant of the transient behavior at each node of the circuit could be very different at the same time;

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMMULTIDC**

namely, a sharp transition could occur to a node at the same time when other nodes have very slow transition.

If a uniform time step is chosen for every node in the circuit, then the smallest time step among all the nodes is needed in order to meet the requirement that the voltage increment over each time step is no more than the value defined by [HSIMALLOWEDDV on page 59](#). Such uniform step size selection can be too conservative because it is unnecessarily small for those nodes with a slow transition.

HSIMMULTIDC

Enables the multi-rate time-step algorithm during DC convergence to improve performance. To disable multi-rate time-step selection during DC convergence, set `HSIMMULTIDC=1` (the default). To enable the multi-rate time-step selection during DC convergence, set `HSIMMULTIDC=0`.

Syntax

```
.param HSIMMULTIDC = <0 | 1>
```

HSIMMSGFILTER

Filters messages. Add `HSIMMSGFILTER` at the beginning of the top-level netlist in order to have an effect on the messages displayed.

Syntax

```
.param HSIMMSGFILTER = [max_messages:] "message_text"
```

Argument	Description
<code>max_messages:</code>	Limits the number of messages to suppress.
<code>message_text</code>	Specifies the message text to suppress from printing.

HSIMNODCNODES

Lists the nodes that do not converge at the completion of DC initialization. Specify HSIMNODCNODES = 1 to generate a *prefix*.nodcnodes file that lists the nodes that do no converge at the completion of DC initialization. Specify to HSIMNODCNODES = 2 to output Max Dv in the *prefix*.nodcnodes file.

The default is 0.

Syntax

```
.param HSIMNODCNODES = <0|1|2>
```

HSIMNODECAP

Controls how to report average capacitance.

Syntax

```
.param HSIMNODECAP="<subckt_name> node_pattern <level>"
```

Description

When HSIM is invoked, if -o file_name is specified, the average capacitance of the specified node_pattern is reported and stored in the capacitance report file hsim.cap or file_name.cap. If the string node_pattern contains asterisk (*) wildcard character(s), or if a subcircuit name or level is specified, that string must be enclosed with double ("") or single ('') quotation characters as shown in the following example.

Examples

```
.param HSIMNODECAP="x1.x2.*"  
.param HSIMNODECAP='x1.*.x2.*'
```

- subckt_name

If subckt_name is specified, only nodes in instances of the given subcircuit are printed.

- level

If level is specified and the node pattern contains wild card characters, matching is only applied up to the number of hierarchical levels specified.

HSIMNOMULTIEND

Errors out during parsing when multiple .end statements exist. HSIM accepts multiple .end statements. If you want HSIM to error out during parsing when multiple .end statements exist, set HSIMNOMULTIEND=1.

Syntax

```
.param HSIMNOMULTIEND = <0 | 1>
```

HSIMNOSIMTIME

Controls simulation time reporting. Set HSIMNOSIMTIME to control the simulation time reported on the screen.

Syntax

```
.param HSIMNOSIMTIME = <0 | 1>
```

Argument	Description
0	Displays a stop watch on screen to report the current simulation time. The time display updates at every 0.01% of total transient simulation time is completed. This is the default value.
1	Turns the stop watch feature off.

HSIMNTLFMT

Specifies the netlist format of the input file.

Syntax

```
.param HSIMNTLFMT = netlist_format
```

Description

HSIMNTLFMT specifies the netlist format of the input file. It is an alternative to the -<netlist_format> command line option. It is a global parameter and its

default is hspice. When using this command, the following restriction applies:
`HSIMTOP` and `HSIMNTLFMT` must be the first commands in the input.

Examples

The following two methods have the equivalent effect.

Method 1

Place `HSIMNTLFMT=spectre` in the input file, then enter the following at the command line:

```
A> HSIM input_spectre_file
```

Method 2

Run HSIM at the command line as shown below:

```
A> HSIM -spectre input_spectre_file
```

HSIMNTLRMIN

Shorts resistors with values smaller than a specified threshold value.

Syntax

```
.param HSIMNTLRMIN = threshold_value
```

Description

`HSIMNTLRMIN` is a subcircuit-level parameter that shorts resistors with values smaller than a specified threshold immediately after parsing a netlist. If `HSIMNTLRMIN` is not set, the default value is the current value of `HSIMRMIN`.

HSIMNVBS

Specifies the grid resolution.

Syntax

```
.param HSIMNVBS = value
```

Description

The grid resolution for V_{gs} voltage is equal to the V_{gs} voltage range divided by the value specified by [HSIMNVGS on page 118](#), which is the number of grids in V_{gs} dimension. Similarly, [HSIMNVDS on page 118](#) and [HSIMNVBS on](#)

[page 117](#) specify the grid numbers in V_{ds} and V_{bs} dimension, respectively. The default is 200.

Note: [HSIMVGSEND on page 192](#), [HSIMVDSEND on page 192](#), and [HSIMVBSEND on page 190](#) can also be included in each individual MOSFET model to achieve flexible control.

HSIMNVDS

Specifies the grid numbers in the VDS dimension. You can use HSIMNVDS to specify the grid numbers in VDS dimension. The default is 60.

Syntax

```
.param HSIMNVDS = value
```

HSIMNVGS

Specifies grid numbers in the VGS dimension. The default is 60.

Syntax

```
.param HSIMNVGS = value
```

HSIMOPCOMPRESS

Specifies the operating point output file format.

Syntax

```
.param HSIMOPCOMPRESS = <0 | 1 | 2>
```

Argument	Description
0	Specifies regular text format (.ic extension, which is the default setting).
1	Specifies .gz format.

Argument	Description
2	Specifies .Z format.

Description

When .op statement is specified in the netlist, HSIM prints each node voltage at the specified time into a file. This command specifies the format of the output file.

The prefix of the output file is `hsim` by default, but can also be specified on the command line by entering `-o file_name`. The default value of time is 0 ns.

Note: To dump the element currents at an operating point, set
`HSIMDUMPOPI = 1`. The default is 0.

HSIMOPTSEARCHEXT

Specifies the file extension used in the .OPTIONS SEARCH (5-47) statement.

Syntax

```
.param HSIMOPTSEARCHEXT = extension_string
```

Description

`HSIMOPTSEARCHEXT` specifies the file extension used in the .OPTIONS SEARCH (5-47) statement. During the subckt/macro file search, it looks for the `<cell>.<ext>` file in the specified search paths. The default extensions are `inc` and `sp`. For example, `param HSIMOPTSEARCHEXT= cir`. If the cell name it looks for is `AND2`, it looks for file `AND2.cir` in the search paths.

HSIMOUTPUT

Specifies the waveform output format.

Syntax

```
.param HSIMOUTPUT = format
.param HSIMOUTPUT fsdb[_version]
```

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMOUTPUTFLUSH**

Argument	Description
fsdb_5.1	Selects the default and the latest version of FSDB. The output file format will be in the FSDB version 5.1.

Description

Valid values are: fsdb (the default), wdf, vpd, nassda, and out.

The HSIM tool supports FSDB version 5.1 format.

Example 11 Output waveform format will be fsdb v5.1

```
.param HSIMOUTPUT fsdb_5.1
```

HSIM supports the VPD (VCD plus dump) output format for all supported platforms.

VPD is a binary format that supports both analog and digital waveforms. You can save voltage, current, and logic waveforms in the VPD output file. The VPD output generated by HSIM can be displayed using either CosmosScope or DVE.

To invoke the feature, use:

```
.param HSIMOUTPUT=vpd
```

HSIMOUTPUTFLUSH

Forces simulation data from memory to the disk file according to the specified time period.

Syntax

```
.param HSIMOUTPUTFLUSH = time_period_value
```

Description

While running the simulation, HSIM keeps the simulation result in a memory buffer. When the memory buffer is full, HSIM transfers the simulation result data from the memory buffer to the disk file and frees up the memory buffer for the up-coming simulation result data. HSIMOUTPUTFLUSH changes this behavior by forcing HSIM to transfer the simulation result data to the disk file time for every time period specified by HSIMOUTPUTFLUSH, even before the memory is filled. It ensures that HSIM flushes the result based on the memory buffer

capacity. The default is -1, which ensures that HSIM flushes the result based on the memory buffer capacity. For example: HSIMOUTPUTFLUSH=200n

HSIMOUTPUTFSDBSIZE

Controls the output size of the FSDB output file.

Syntax

```
.param HSIMOUTPUTFSDBSIZE = size_value
```

Description

HSIMOUTPUTFSDBSIZE controls the output size of the FSDB output file. The default value provides no splitting. HSIMOUTPUTFSDBSIZE = 1 means split the FSDB file every 1 MB. HSIMOUTPUTFSDBSIZE = 200 means split the FSDB file every 200 MB.

HSIMOUTPUTIRES

Controls the waveform output resolution.

Syntax

```
.param HSIMOUTPUTIRES = output_resolution_value
```

Description

To control the waveform output file size, HSIM does not write a new waveform data point for a given signal unless the value of the signal has changed from the last written data point by more than the output resolution. The output resolution is controlled by HSIMOUTPUVRES for voltage signals and HSIMOUTPUTIRES for current signals. The default is 1E-9 (1 nA).

HSIMOUTPUTMEAS

Writes the signals referenced in .meas to the output file.

Syntax

```
.param HSIMOUTPUTMEAS = <0 | 1>
```

Chapter 4: HSIM® Commands in Alphabetical OrderHSIMOUTPUTTBL

Description

Specifies whether (1, the default) or not (0) the signals referenced in .meas are output to the fsdb output file.

HSIMOUTPUTTBL

Creates waveform output files in csdf or raw formats.

Syntax

```
.param HSIMOUTPUTTBL = format
```

Description

HSIMOUTPUTTBL allows the creation of output waveform file in the SPICE raw and csdf formats. These files are created in post-processing after the simulation completes. Set *format* to csdf for csdf format or rawfile for raw format.

HSIMOUTPUTTRES

Controls the output time resolution.

Syntax

```
.param HSIMOUTPUTRES =value
```

Description

The HSIMOUTPUTRES value cannot exceed 1000 times the HSIMTIMESCALE value. Note that when you use HSIMOUTPUTRES with a value that is aggressively greater than the simulation time step, which is confined by HSIMOUTAUMAX, the resulting waveforms might be distorted. The default value is 1E-12 (1ps).

HSIMOUTPUTTSTEP

Sets the output result format to be a fixed time step.

Syntax

```
.param HSIMOUTPUTTSTEP =value
```

Description

The default for `HSIMOUTPUTTSTEP` is not set. The result format is in value changed form. When `HSIMOUTPUTTSTEP` is set to a certain time value, such as 1ns or 100ps, the result format is in fixed time steps by every 1ns or 100ps repetitively.

HSIMOUTPUTVRES

Controls output voltage resolution.

Syntax

```
.param HSIMOUTPUTVRES =output_resolution_value
```

Description

To control the waveform output file size, HSIM does not write a new waveform data point for a given signal unless the value of the signal has changed from the last written data point by more than the output resolution. The output resolution is controlled by `HSIMOUTPUTVRES` for voltage signals and `HSIMOUTPUTIRES` for current signals. The default is 1E-3 (1 mV).

HSIMOUTPUTWDFSIZE

Controls the output size of the WDF output file.

Syntax

```
.param HSIMOUTPUTWDFSIZE =value
```

Description

The default value (-1) for `HSIMOUTPUTWDFSIZE` specifies no splitting. `HSIMOUTPUTWDFSIZE = 1` means splits the WDF file every 1 MB. `HSIMOUTPUTWDFSIZE = 200` means splits the WDF file every 200 MB and so on.

HSIMPARAMCACHING

Improves performance when parsing a netlist that has subcircuit or device instances with a large number of parameters.

Syntax

```
.param HSIMPARAMCACHING = <0 | 1>
```

Argument	Description
0	Turns off calculating and storing parameter values in a cache (the default).
1	Calculates and stores parameter values in a cache, which can improve netlist parsing performance.

Description

When parsing a netlist with subcircuit or device instances that have a large number of parameters, HSIM has to search the netlist database for each parameter, compute its value, and pass it to corresponding subcircuit or model definition. This stage can be very time consuming.

To improve performance parsing a netlist that has instances with a large number of parameters, set `HSIMPARAMCACHING=1` to pre-calculate and store the parameter values in a cache.

HSIMPAPRECISION

Controls the coarseness or aggressiveness of partitioning. For further details about the `HSIMPAPRECISION` settings, see [HSIM Quick Start on page 12](#).

Syntax

```
.param HSIMPAPRECISION = <0 | 1 | 2 | 3 | 4>
```

Description

`HSIMPAPRECISION` is a macro command. It controls the coarseness or aggressiveness of partitioning. Simulators use partitioning to handle large designs and by breaking a large mathematical problem into smaller, more manageable matrices. Without this approach, all simulators would have the

same size limitations as traditional SPICE. If this algorithmic approach is done appropriately, performance will increase with minimal impact on precision.

A value of 0 specifies smaller partition sizes, which results in faster simulations. A value of 4 specifies larger partition sizes, which results in slower, more accurate simulations. The default is 1.

In addition to the HSIMPAPRECISION command, you can also specify the following commands for greater flexibility in partitioning:

- [HSIMABMOS on page 52](#)
- [HSIMAMOS on page 60](#)
- [HSIMBMOS on page 70](#)

HSIMPORTCR

Controls the subcircuit capacitance ratio for isomorphic matching.

Syntax

```
.param HSIMPORTCR = tolerance_value
```

Description

HSIM matches any two identical subcircuits (or partitions) that have similar port voltages and capacitive loading (within specified tolerances). If the port loading difference among partitions are within the tolerance value specified by HSIMPORTCR (default 0.01 or 1%), all such partitions are matched and share simulation results.

Examples

```
.param HSIMPORTCR=0.05
```

To determine isomorphic matching, use a 5% tolerance for port capacitance ratio.

HSIMPORTI

Defines the port current tolerance.

Syntax

```
.param HSIMPORTI =value
```

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMPORTV

Description

The port current tolerance is defined such that two corresponding port currents are considered matched if their difference is less than the value specified by HSIMPORTI. By default, HSIM only considers port voltage tolerance. However, to obtain accurate currents through small resistors and zero volt floating voltage sources, you should specify a HSIMPORTI value. The unit is in Ampere.

HSIMPORTV

Specifies the port voltage tolerance value.

Syntax

```
.param HSIMPORTV =value
```

Description

Voltage drops across parasitic resistors generally result in different port voltages at identical subcircuits connected in parallel. For example, the voltage at the memory core cell port connecting to the bit-line is different for each memory cell connecting to the same bit-line when a parasitic resistor exists between any two neighboring memory core cells. The port voltages are otherwise identical in the absence of resistors.

If the port voltage difference exceeds the tolerance value specified by HSIMPORTV, each subcircuit requires separate calculations. If it does not exceed the tolerance voltage, calculated results can be shared among subcircuits with port voltage difference being less than the tolerance value specified by HSIMPORTV. The default is 1 mV.

To provide additional speed-up for post-layout simulation, HSIMPOSTL = 3 relaxes the port voltage tolerance from 1 mV to 10 mV.

HSIMPOSTL

Controls the RC reduction method.

Syntax

```
.param HSIMPOSTL = <0 | 1 | 2 | 3>
```

Argument	Description
0	Turns off RC reductions (the default).
1	Activates the RC reduction method.
2	This setting is recommended when: <ul style="list-style-type: none"> ▪ The circuit contains nets that connect to a large number of MOSFET drain or source terminals. ▪ Current flow direction in these transistors is primarily bidirectional.
3	To speed up post-layout simulation, this setting relaxes the port voltage tolerance from 1 mV to 10 mV.

Description

There is no RC reduction when the default setting `HSIMPOSTL=0` is used. If you specify this value, HSIM checks every resistor in RC network, including the SPF RC network against the `rmin` value. If all the resistors are greater than `rmin`, no processing is performed. If at least one resistor is smaller than `rmin`, the whole RC network is processed for parallel reduction and `rmin`:
 $rmin=\min(HSIMRMIN, HSIMVSRCRMIN)$. Note that this calculation is processed no matter which RC network is connected to VSRC nodes.

When `HSIMPOSTL = 1` is set, the RC reduction method is activated. HSIM automatically reduces the RC network in each signal net to a smaller equivalent RC network. The reduced network contains the following elements:

- Resistors
- Capacitors
- Multiterminal control sources

The reduced network has nearly identical time-domain circuit behavior as the unreduced network. You can also set `HSIMPOSTL` on a subcircuit level.

The RC reduction algorithm set by `HSIMPOSTL = 1` handles parasitic RCs in the signal nets where the loading gates have coupled capacitance to the net through the transistor gate terminals such as a clock tree. In cases where most transistors are connected to the net through their drain or source terminals, the traditional RC reduction algorithm is not effective because the transistor current direction could flow either way. Such is the case of bit-line in a memory core. The pass transistor connected between a bit-line and a core cell capacitor has a different current direction in the read or write cycle. A proprietary algorithm is

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMPOSTLMANY2M

implemented to reduce parasitic RCs in nets connecting to MOSFET drain or source terminals.

HSIMPOSTLMANY2M

Enables a more conservative reduction algorithm.

Syntax

```
.param HSIMPOSTLMANY2M = <0 | 1>
```

Description

Works in conjunction with `HSIMPOSTL` on nets with many-to-many connections and enables fine tuning for application specific reduction. Specify `HSIMPOSTLMANY2M = 1` to enable a more conservative reduction algorithm, applied for RC networks with many-to-many connections. Examples of nets with many-to-many connections are memory bitline and clock tree mesh nets (with many drivers to many receivers). The default is 0.

HSIMPOSTLONE2M

Enables a more conservative RC reduction algorithm.

Syntax

```
.param HSIMPOSTLONE2M = <0 | 1>
```

Description

Works in conjunction with `HSIMPOSTL` on nets with one-to-many connections and enables fine tuning for application specific reduction. Set `HSIMPOSTONE2M = 1` to enable a more conservative reduction algorithm, applied for RC networks with one-to-many connections. Examples of nets with one-to-many connections are memory wordline and clock tree nets (with one driver to many receivers). The default is 0.

HSIMPREFERVERILOGA

Specifies the priority order to search names

Syntax

```
.param HSIMPREFERVERILOGA = <0 | 1>
```

Description

Whenever there is a clash between the names of subcircuits, C-modes, and Verilog-A model names in an HSIM netlist, HSIM resolves the names in the following order:

1. Subcircuit name.
2. C-model name.
3. Verilog-A model name.

For each X instance in the HSIM netlist, HSIM first attempts to match the name with the subcircuit name. If the subcircuit name is not matched, HSIM tries to find a C-model with the same name. Verilog-A models with that name are searched last.

To increase the Verilog-A priority with respect to other model types in HSIM, use the following syntax for HSIMPREFERVERILOGA:

```
.param HSIMPREFERVERILOGA=1
```

When HSIMPREFERVERILOGA is specified, the Verilog-A model name option supersedes all other options and the order changes as follows:

1. Verilog-A model name.
2. Subcircuit name.
3. C-model name.

HSIMPREFLAT

Flattens the netlist before partitioning.

Syntax

```
.param HSIMPPREFLAT = <0 | 1>
```

Description

HSIMPPREFLAT=1 is equivalent to reading in a flat netlist. This command is valuable in some cases of post-layout simulation where the parasitic resistors and capacitors (RC)s are defined within the subcircuits.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMPRINTSIMSTATUS**

Note: This command is not recommended for large circuit simulation due to high memory usage considerations.

HSIMPRINTSIMSTATUS

Controls the simulation time refresh rate.

Syntax

```
.param HSIMPRINTSIMSTATUS = <0 | 1>
```

Description

HSIMPRINTSIMSTATUS controls the rate at which simulation time is refreshed. The default (1) is to refresh simulation status on-screen every nanosecond.

With long simulation times, especially when a simulation is running in the background, this refresh rate is typically too short. To refresh the screen every 10% of the total simulation time, specify a value of 0.

HSIMPROBEALL

Enables the full-chip probing flow for voltages. For more information about the full-chip probing flow, See the [Full-Chip Probing on page 430](#) section.

Syntax

```
.param HSIMPROBEALL = <0 | 1>
```

Argument	Description
0	Disables the full-chip probing flow (default).
1	Enables the full-chip probing flow for voltages.

HSIMPROBEAUTO

Determines whether to run phase 2 of the full-chip probing flow automatically. For more information about the full-chip probing flow, See the [Full-Chip Probing](#) section.

Syntax

```
.param HSIMPROBEAUTO = <0 | 1 | 2>
```

Argument	Description
0	Disables phase 2 of the full-chip probing flow from running automatically.
1	Enables phase 2 of the full-chip probing flow to run automatically.
2	Enables phase 2 of the full-chip probing flow to run automatically and deletes the phase 1 file after the simulation completes.

HSIMPROBEFILTER

Lets you control printing and provides a way to filter unwanted problems without modifying the input netlist.

Syntax

```
.param HSIMPROBEFILTER = <"all" "v" "i" "x" "l" "meas">
```

Argument	Description
"all"	Prints all variables (current, voltage, xprint, and so on). This is the default setting.
"v"	Prints only voltage variables.
"i"	Prints only current variables.
"x"	Prints only subcircuit port current variables.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMPROBEREST**

Argument	Description
"1"	Print only logic variables.
"meas"	Prints only .meas variables.

HSIMPROBEREST

Specifies whether to merge waveform files in the full-chip probing flow. The default is 0 (do not merge waveform files).

Syntax

```
.param HSIMPROBEREST = <0 | 1>
```

Description

For more information about the full-chip probing flow, See the [Full-Chip Probing on page 430](#) section.

HSIMRASPFMODXY

Calculates bounding box data.

Syntax

```
.param HSIMRASPFMODXY = <0 | 1>
```

Description

HSIMRASPFMODXY is used when the extractor generates bounding box data instead of true node coordinates. If it is set to 1, the coordinates of the bounding box centers are calculated and used for visualization of resistor connectivity.

HSIMRATSTART

Controls the starting time for the second phase RA simulation.

Syntax

```
.param HSIMRATSTART = time
```

Description

This command is equivalent to the TSTART RA command, where TSTART is specified in the RA Tcl command file. The difference is that HSIMRATSTART is parsed and processed in the first phase simulation.

By default, when HSIMPWRA or HSIMSIGRA is enabled HSIM collects dynamic information throughout transient analysis. You can narrow this time window by setting the HSIMRATSTART and HSIMRATSTOP commands. When these two commands are set, HSIM collects dynamic information only during the specified time interval.

HSIMRATSTOP

Controls the stop time for the second phase RA simulation.

Syntax

```
.param HSIMRATSTOP = time
```

Description

This command is equivalent to the TSTOP RA command, where TSTOP is specified in the RA Tcl command file. The difference is that HSIMRATSTOP is parsed and processed in the first phase simulation.

By default, when HSIMPWRA or HSIMSIGRA is enabled HSIM collects dynamic information throughout transient analysis. You can narrow this time window by setting the HSIMRATSTART and HSIMRATSTOP commands. When these two commands are set, HSIM collects dynamic information only during the specified time interval.

HSIMRAVTS

Determines the way device currents are processed and recorded for RA analysis. This command lets you control the tradeoff between disk usage and first phase simulation overhead.

Syntax

```
.param HSIMRAVTS=<0 | 1 | 2>
```

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMRCMATCHRELERR**

Argument	Description
0	In the first phase of RA simulation, device currents are averaged and written to the disk at every HSIMRATAU interval.
1	In the first phase of RA simulation, device currents are written to the disk whenever the simulation computes new values. Computed current values are saved straight to the disk and averaging is performed during second phase of RA simulation. This setting is recommended when HSIMRATAU is significantly smaller than first phase simulation time step.
2	If the HSIMRATAU value is greater than the HSIMTAUMAX value, HSIM uses the HSIMRAVTS=0 approach and saves the averaged currents. Otherwise HSIM uses the HSIMRAVTS=1 approach and saves non-averaged (instant) current values (the averaging is done at the second phase of RA). This is the default setting.

HSIMRCMATCHRELERR

Controls the tolerance on RC isomorphic matching.

Syntax

HSIMRCMATCHRELERR = *tolerance_value*

Description

HSIM matches any two identical subcircuits (or partitions) that have similar resistance and capacitance values (within specified tolerance). If the RC value difference among partitions are within the tolerance value specified by HSIMRCMATCHRELERR (default 0.05 or 5%), all such partitions are matched and share simulation results.

HSIMRCNPO

Controls the hierarchical RC reduction process.

Syntax

```
.param HSIMRCNPO = <0 | 1 | 2>
```

Argument	Description
0	Performs RC reduction independently in each subcircuit (the default). Even if there is a connection between two RC networks located in different subcircuit instances, they are still treated as two separate RC networks.
1	Performs RC reduction hierarchically, causing all of the connections between RC elements to be taken into account, including the connections between elements located in different subcircuit instances. This method is similar to selective flattening of RC networks without flattening the containing subcircuits. When necessary, RC networks located at lower levels of hierarchy are pulled out of their subcircuits and moved to higher levels of hierarchy.
2	Only the resistor elements (R-networks) are pulled out before reduction. After the R-networks are processed, capacitor reduction is performed as it is when HSIMRCNPO = 0.

Description

For RC reduction, HSIMRCNPO has the same effect as HSIMPREFLAT however, HSIMPREFLAT flattens entire subcircuits, which can be extremely inefficient in terms of speed and memory consumption compared to the selective RC network pull-out. HSIMRCNPO is a global parameter.

HSIMRCPRECISION

Controls the degree of reduction applied to designs. For details about the HSIMRCPRECISION settings, see [HSIM Quick Start on page 12](#).

Syntax

```
.param HSIMRCPRECISION = <-1 | 0 | 1 | 2 | 3 | 4 | 5>
```

Description

HSIMRCPRECISION is a macro that controls the degree of RC and coupling capacitor reduction tolerances applied. Reduction is required for large designs with back-annotated layout parasitics. HSIMRCPRECISION determines the precision and performance trade-off associated with the different methods of reduction available to HSIM. Some of these methods include serial, parallel, and moment based reduction.

A value of 0 specifies aggressive RC reduction, while a value of 5 specifies no RC reduction. The default is -1 (off). The following commands also affect RC reduction and tolerances:

- [HSIMPOSTL on page 126](#)
- [HSIMRCRTAU on page 138](#)
- [HSIMRMIN on page 145](#)

For more information about coupling capacitor reduction, see [Control Commands for Floating Capacitors on page 347](#).

HSIMRCRIO

Controls how to preserve nodes specified in .ic/.nodeset/.print statements.

Syntax

```
.param HSIMRCRIO = <0 | 1 | 2>
```

Argument	Description
0	Keeps track of where a node is merged to when it is reduced during RC reduction. This ensures that input and output (I/O) statements such as .ic/.nodeset/.print can still apply.
1	If an improvement of precision for internal RC nodes is desired, specify this value to preserve all nodes specified in .ic/.nodeset/.print statements.
2	Set this values to preserve the nodes with .ic/.nodeset/.print and keep the adjacent RLC elements.

Description

The commands and parameters that preserve subcircuit nodes and elements during RC reduction are shown in [Table 3](#).

Table 3 Commands that Preserve Subcircuit Node/Elements During RC Reduction

Parameter	Description
.measure	All nodes and RLC elements specified in .measure statements.
VEC CHECK	All nodes specified in VEC CHECK.
\$MODEL=model_name	All RLC elements with \$MODEL=model_name specified and model_name is specified in parameter HSIMRCRKEEPMODEL.

HSIMRCRKEEPELEM

Preserves elements in RC reduction if they match the specified pattern. HSIMRCRKEEPELEM can be used to specify an element pattern. All the elements that match the pattern are preserved in RC reduction.

Syntax

```
.param HSIMRCRKEEPELEM = element_pattern
```

HSIMRCRKEEPMODEL

Preserves the specified model in RC reduction. HSIMRCRKEEPMODEL preserves the specified model name in the \$MODEL statement. See the following example.

Syntax

```
.param HSIMRCRKEEPMODEL = model_name
```

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMRCRKEEPNODE

Examples

In the following example, R1 is preserved during the RC reduction because the R1 description includes rrb in the \$MODEL assignment and rrb is specified in an HSIMRCRKEEPMODEL statement. A similar situation applies to C2.

```
.param hsimrcrkeepmodel=rrb
.param hsimrcrkeepmodel=rrc

R1    1    2    100    $MODEL=rrb
C2    2    3    15f    $MODEL=rrc
...
```

HSIMRCRKEEPNODE

Preserves nodes in RC reduction if they match the specified pattern.

HSIMRCRKEEPNODE can specify a node pattern such that all the nodes that match the pattern and the RLC elements directly connected to the node are preserved in RC reduction.

Syntax

```
HSIMRCRKEEPMODE = node_pattern
```

HSIMRCRTAU

Controls the accuracy of the RC reduction solver.

Syntax

```
HSIMRCRTAU = value
```

Description

Controls the accuracy of the RC reduction solver and defines a cutoff frequency for RC networks during reduction. The default value is 1.0e-10s. It is not recommended to set it higher than default value.

The only time to use HSIMRCRTAU for a more accurate RC reduction is when HSIMPOSTL = 1. If you need better accuracy, set HSIMRCRTAU to a smaller value. While you might not see a big difference in every design, this is the correct way to use HSIMRCRTAU and HSIMPOSTL. Use either the default HSIMRCRTAU value or set it to a smaller value such as 5ps.

When **HSIMPOSTL** = 0(the default), there is no RC reduction, so **HSIMRCRTAU** **HSIMRCRTAU** is irrelevant. When **HSIMPOSTL** = 2, RC reduction is relatively aggressive, so it does not make sense to use **HSIMPOSTL** = 2 with **HSIMRCRTAU**.

HSIMREDEFSUB

Allows a simulation to continue when there are duplicate subcircuit names. When you specify a value of 1 for this command in a simulation with duplicate subcircuit names, the last subcircuit definition is selected and the previous definition is ignored. When you specify a value of 2 for this command in a simulation with duplicate subcircuit names, the first subcircuit definition is selected and the previous definition is ignored.

The default is 0.

Syntax

```
.param HSIMREDEFSUB = <0|1|2>
```

Examples

The following netlist has two subcircuit definitions for **inv**:

```
.subckt inv a b  
...  
.ends  
.subckt inv p q r  
...  
.ends
```

HSIM selects the second subcircuit definition for subcircuit **inv** if:

- The syntax statement shown below is present in the netlist.
- It is located at the top-level and outside of any subcircuit definition.

```
.param HSIMREDEFSUB=1
```

HSIMREDEFSUBNAME

Lets you specify a list of subcircuits to be considered in the context of the duplicated circuits selected by the **HSIMREDEFSUB** command. You can use

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMREDEFSUBNAME**

wildcard characters in the subcircuit names. Note that you can only use this command with HSIMREDEFSUB.

Syntax

```
.param HSIMREDEFSUBNAME = "subcircuit_name1"  
      "subcircuit_name2" ...
```

Examples

The following example shows how HSIMREDEFSUB works without the HSIMREDEFSUBNAME command.

```
.param HSIMREDEFSUB=2  
.global vdd1 vdd2 vdd3  
V1 vdd1 0 3  
  
.subckt my_inv1 a z  
M1first z a vdd1 vdd1 P250 w=1u l=1u  
M2first z a 0 0 N250 w=1u l=1u  
.ends  
  
.subckt my_inv1 a z  
M1last z a vdd2 vdd2 P250 w=1u l=1u  
M2last z a 0 0 N250 w=1u l=1u  
.ends  
  
.subckt my_inv2 a z  
M1first z a vdd3 vdd3 P250 w=1u l=1u  
M2first z a 0 0 N250 w=1u l=1u  
.ends  
  
.subckt my_inv2 a z  
M1first_ z a vdd4 vdd4 P250 w=1u l=1u  
M2first_ z a 0 0 N250 w=1u l=1u  
.ends  
...
```

After it processes this netlist, HSIM issues the following warning and error messages:

```

Warning: file 'test2.cir' line 13: attempt to redefine subckt
'my_inv1', will use the first definition, ignoring later
definition(s)
--> .subckt my_inv1 a z

Error: file 'test2.cir' line 18: attempt to redefine subckt 'my_',
will use the first definition, ignoring later definition(s)
--> .subckt my_inv1 a z

Warning: file 'test2.cir' line 23: attempt to redefine subckt
'my_inv2', will use the first definition, ignoring later
definition(s)
--> .subckt my_inv2 a z

Warning: file 'test2.cir' line 28: attempt to redefine subckt
'my_inv2', will use the first definition, ignoring later
definition(s)
--> .subckt my_inv2 a z

```

The following example shows how HSIMREDEFSUB works with the HSIMREDEFSUBNAME command.

```

.param HSIMREDEFSUB=1
.param HSIMREDEFSUBNAME="my_inv2"
.global vdd1 vdd2 vdd3
...
.subckt my_inv1 a z
M1first z a vdd1 vdd1 P250 w=1u l=1u
M2first z a 0 0 N250 w=1u l=1u
.ends

.subckt my_inv1 a z
M1last z a vdd2 vdd2 P250 w=1u l=1u
M2ilast z a 0 0 N250 w=1u l=1u
.ends

.subckt my_inv2 a z
M1first z a vdd3 vdd3 P250 w=1u l=1u
M2first z a 0 0 N250 w=1u l=1u
.ends

.subckt my_inv2 a z
M1first_ z a vdd4 vdd4 P250 w=1u l=1u
M2first_ z a 0 0 N250 w=1u l=1u
.ends
...

```

After it processes this netlist, HSIM issues the following warning and error messages:

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMREGNODE**

```
Error: file 'test3.cir' line 15: attempt to redefine subckt  
'my_inv1' is not allowed unless 'my_inv1' is set within parameter  
HSIMREDEFSUBNAME explicitly (wildcard is allowed)  
--> .subckt my_inv1 a z

Error: file 'test3.cir' line 20: attempt to redefine subckt  
'my_inv1' is not allowed unless 'my_inv1' is set within parameter  
HSIMREDEFSUBNAME explicitly (wildcard is allowed)  
--> .subckt my_inv1 a z

Warning: file 'test3.cir' line 25: attempt to redefine subckt  
'my_inv2', will use the last definition, overwriting previous  
definition(s)  
--> .subckt my_inv2 a z

Warning: file 'test3.cir' line 30: attempt to redefine subckt  
'my_inv2', will use the last definition, overwriting previous  
definition(s)  
--> .subckt my_inv2 a z
```

HSIMREGNODE

Improves performance and reduces memory usage when simulating designs with voltage-regulated power nodes.

Syntax

```
.param HSIMREGNODE = "node=node_type...  
    input=regnode_drivers...  
    <subckt=subcircuit_name...>|<inst=instance_name...>  
    <level=value>"
```

Argument	Description
node=node_type	Specify one of the following node types: ■ Full hierarchical node name. ■ Subcircuit port name, which uses the subckt=subcircuit_name argument.

Argument	Description
<code>input=regnodel_drivers</code>	Specifies one of the following instance types. <ul style="list-style-type: none"> ▪ Full hierarchical instances (transistors or instances) that drive the regulated node when you specify a full hierarchical node name. ▪ Subcircuit instances (transistors or instances) that drive the regulated node when you use the <code>subckt=subcircuit_name</code> argument. Separate the list of regulated nodes with a space.
<code>subckt=subcircuit_name</code>	Specifies the subcircuit name when <code>HSIMREGNODE</code> is applied to a subcircuit.
<code>inst=instance_name</code>	Specifies the instance name when <code>HSIMREGNODE</code> is applied to an instance.
<code>level=value</code>	Controls the coupling effects and current probing of regulated node partitions. Specify one of the following values: <ul style="list-style-type: none"> ▪ 1 to enable the fastest regulated node partitioning algorithm. This option disregards the coupling between the loading and driving circuitry of the regulated node partition. Current probing through a regulated node is not supported. ▪ 2 specifies, in addition to 1, to allow current probing through a regulated node partition. ▪ 3 (default) enables the most accurate regulated node partitioning algorithm. In addition to 2, this option also considers the loading effects on the driving circuitry (feedback). Specify this value when the dynamic load/fanout can influence the regulated circuitry, or if it is necessary to determine the current and voltage specifications.

Description

In pre-layout simulations, `HSIMREGNODE` applies the decoupled nature of designs with constant voltage sources to designs with voltage-regulated power nodes. The result is better performance and improved memory efficiency simulating this type of design.

Note: You cannot use both the `inst` and `subckt` arguments in the same `HSIMREGNODE` command.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMRELCURTOL****Examples**

```
.param HSIMREGNODE="node=xtop.vddinternal  
input=xtop.xio.x1.xhd_switch"
```

Applies HSIMREGNODE to the xtop.xio.x1.xhd_switch driver and its vddinternal port.

```
.param HSIMREGNODE=" node=vddinternal input=xhd_switch  
inst=xtop.xi0.x1"
```

HSIM searches the xtop.xio.x1 instance and looks for its xhd_switch internal instance and the related vddinternal port to apply HSIMREGNODE.

```
.param HSIMREGNODE=" node=vddinternal input=xi0 xi1  
subckt=sram_head_switch_10"
```

HSIM searches the sram_head_switch_10 subcircuit and looks for its xi0 and xi1 internal instances and the related vddinternal port to apply HSIMREGNODE.

HSIMRELCURTOL

Sets the relative current tolerance.

Syntax

```
.param HSIMRELCURTOL = value
```

HSIMREPORTAREA

Reports the total transistor gate area. When HSIMREPORTAREA=1 a reported is printed that shows the length, width, and gate area. The default is 0.

Syntax

```
.param HSIMREPORTAREA = <0|1>
```

Examples

```
Total MOS length : 3.6e-06  
Total MOS width : 0.000704  
Total MOS gate area : 1.2672e-10  
*****
```

HSIMRGATEMOD

Models gate load resistance. Set `HSIMRGATEMOD = 1` to model constant gate load resistance. The default is 0.

Syntax

```
.param HSIMRGATEMOD = <0 | 1>
```

HSIMRMAX

Sets the maximum resistor value.

Syntax

```
.param HSIMRMAX = value
```

Description

To eliminate large resistors, this command sets the maximum resistor value allowed in a simulation. All resistors with a value larger than `HSIMRMAX` are open-circuited. Because it is performed before RC reduction, eliminating large resistors also helps give a more effective RC reduction with little loss of precision. The default value is `1e15`.

HSIMRMIN

Sets the minimum resistor value.

Syntax

```
.param HSIMRMIN = value
```

Description

To eliminate small resistors, this command sets the minimum resistor value allowed in simulation. All resistors with a value smaller than `HSIMRMIN` are short-circuited. Since it is performed before RC reduction, eliminating small resistors also helps give a more effective RC reduction with a very minor loss of precision. `HSIMRMIN` can be set locally and globally. A local setting of `HSIMRMIN` has a higher precedence over the global setting. The default value is 0.1.

HSIMRMVDIO

Removes voltage clipping diodes.

Syntax

```
.param HSIMRVDIO = <0 | 1 | 2>
```

Argument	Description
0	Keeps the diodes connected between two voltage sources.
1	Removes the diodes connected between two voltage sources.
2	Removes the diodes connected between two voltage sources or detected power nodes. This is the default value.

Description

HSIMRMVDIO removes voltage clipping diodes connected to voltage sources. Set this command value to 1 or 2 if another command is required to monitor these types of diodes.

HSIMSAMPLERATE

Determines how long the sine wave should trigger the input node.

Syntax

```
.param HSIMRSAMPLERATE = value
```

Description

For sine voltage sources, HSIM needs to determine how long the sine wave should trigger the input node. The default value of HSIMSAMPLE = 160 means that for each cycle there are 160 triggering points.

Examples

In the following example every 3ps the input sine wave triggers the input node. For a sine wave input with a 2 GHz frequency:

```
one cycle=1/2GHz=500ps  
500ps/160 points=~3ps/point
```

HSIMSCALE

Sets the scale factor for the specified subcircuit.

Syntax

```
.param HSIMRSSCALE = value
```

Examples

```
.hsimparam subckt=need_to_scale HSIMSCALE=1.e-6
```

HSIMSELBA

Determines whether to run the one or two phase selected net simulation flow.

Syntax

```
.param HSIMSELBA = <0 | 1>
```

Argument	Description
0	Specifies to run the two-phase selected net simulation flow.
1	Specifies to run the one phase selected net simulation flow.

Description

For more information about the selected net simulation flow, see the [Selective Net Backannotation Flow on page 541](#) section.

HSIMSKIP

Lets HSIM ignore subcircuits or instances in a simulation without modifying the netlist.

Syntax

```
.param HSIMSSKIP = <0 | 1>
```

Description

This command lets you specify subcircuits or instances to ignore in a simulation using the following syntax:

```
.hsimparam <subckt [inst] | inst> = <name> HSIMSKIP=1
```

You can specify wildcards in the instance-based based syntax.

Examples

```
.hsimparam subckt=sub1 HSIMSKIP=1
```

Specifies that instances referering to `sub1` are removed from the database prior to the simulation.

```
.hsimparam inst=x1.x*1 HSIMSKIP=1
```

Specifies that instances matching the `x1.x*1` pattern are removed from the database prior to the simulation.

```
.hsimparam subckt=sub1 inst=x*1 HSIMSKIP=1
```

Specifies that within every instance refering to `sub1` and instances matching the `x*1` pattern are removed from the database prior to the simulation.

HSIMSNCS

Bypasses idle subcircuit elements to speed up simulation.

Syntax

```
.param HSIMRSNCS = <0 | 1>
```

Description

Using circuit latency may cause the simulation to run faster. Instead of simulating every circuit element at every time step, setting `HSIMSNCS = 1` (the default) bypasses those subcircuit elements at the time when those elements are idle. A subcircuit is treated as being idle if every node in the subcircuit is idle.

Table 4 contains the HSIMSNCS parameter descriptions.

Table 4 HSIMSNCS Node Voltage Parameters

Parameter	Description
vn+1, vn	Node voltages at two consecutive time points
M	Tolerance defined by $M=h*Itolerance/C$
h	Time stepsize
C	Node capacitance
Itolerance	Steady current tolerance defined by HSIMSTEADYCURRENT.

The latency setting using HSIMSNCS = 1 should not cause a precision problem if the circuit functionality is not affected by neglecting the small current defined by HSIMSTEADYCURRENT.

The default value of HSIMSNCS = 1 can only be set at the top level, so it has a global effect on the entire circuit.

Examples

The criterion for determining if a node is idle is by checking the node voltage change:

$$|V_{n+1} - V_n| < M$$

HSIMSOIBULKMOD

Fixes the SOI floating bulk node voltage.

Syntax

```
.param HSIMSOIBULKMOD = <0 | 1 | 2>
```

Argument	Description
0	Keeps the floating bulk node voltage (default).

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMSPEED**

Argument	Description
1	Fixes the Vbs value for NMOS and PMOS SOI. In addition, you need to set the following commands to specify the Vbs values: <ul style="list-style-type: none">▪ HSIMSOIVBSP=<i>value</i> where <i>value</i> is a real number for the PMOS SOI Vbs value.▪ HSIMSOIVBSN=<i>value</i> where <i>value</i> is a real number for the NMOS SOI Vbs value.
2	Fixes the Vb value for NMOS and PMOS SOI. In addition, you need to set the following commands to specify the Vb values: <ul style="list-style-type: none">▪ HSIMSOIVBP=<i>value</i> where <i>value</i> is a real number for the PMOS SOI Vb value.▪ HSIMSOIVBN=<i>value</i> where <i>value</i> is a real number for the NMOS SOI Vb value.

HSIMSPEED

Sets the simulation speed in relation to precision.

Syntax

```
.param HSIMSPEED = <0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8>
```

Argument	Description
0	Set this value when the simulation result is expected to closely match with the SPICE result. This value is recommended only for simulating small analog circuits with a circuit size less than 1,000 elements.

Argument	Description
1	<p>Significantly speeds up simulation compared to the value of 0 is achieved while the precision loss is minimal, especially for large circuits. Significant precision loss may occur when setting HSIMSPEED on page 150 = 1 in two applications:</p> <ul style="list-style-type: none"> ▪ Simulating leakage currents ▪ Simulating circuits with very low power supply voltage <p>Accurate simulation result can be achieved by using HSIMSPEED on page 150 in the case of leakage current simulation, setting HSIMSTEADYCURRENT on page 172 to about 1% of the expected leakage current value is recommended when the HSIMSPEED on page 150 value is greater than 0.</p> <p>Note that when simulating circuits with low power supply voltage, setting HSIMALLOWEDDV on page 59 to about 10% of the power supply voltage is recommended.</p>
2	<p>With this value a further speedup is achieved by flattening selected circuit hierarchies, which reduces some hierarchical simulation overhead. The selection of a subcircuit for flattening is determined by evaluating the probability of its subcircuit instances to share the same simulation result. There should be little precision loss in setting a value of 2 versus the setting of 1, but memory usage could increase.</p>

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMSPEED

Argument	Description
3	<p>The default value of 3 provides additional speed-up through the multirate time stepsize selection.</p> <p>When HSIMSPEED on page 150 is set to less than 3, the time stepsize selection is uniform such that a common stepsize is selected for every subcircuit.</p> <p>When the HSIMSPEED on page 150 value is greater than or equal to 3, different time stepsizes are selected in different subcircuits, such that a smaller time step is selected for subcircuits with faster transient activities, while a larger time step is selected for subcircuits with slow transient activities. This speed-up in multirate step selection is achieved by reducing the total number of time steps. The precision loss in setting in HSIMSPEED on page 150 compared with lower values is significant only in very few applications such as simulating high-sensitivity analog circuits.</p> <p>However, it is generally not necessary to lower the HSIMSPEED on page 150 value: the HSIMANALOG on page 61 control parameter achieves analog simulation speed without sacrificing precision.</p>
4	<p>This value achieves an increased speed-up compared to HSIMSPEED on page 150 by relaxing the precision resolution. A value of 4 is recommended for simulating digital circuits, memory circuits, or analog circuits with lower sensitivity. In such applications, the precision difference between the values of 3 and 4 is minimal.</p>
5	<p>This value is recommended for functional simulation, including memory and mixed-signal circuits. In this case, simulation speed-up is achieved by sacrificing timing delay precision. In general, 5-10% of timing delay error is expected when using this high-speed simulation mode.</p>
6	<p>This value was formerly used for functional simulation of digital circuits and low sensitivity analog circuits. However, it is now recommended that HSIMSPEED on page 150.</p>
Note: HSIMSPEED on page 150=6 is now obsolete.	

Argument	Description
7	This value is recommended for functional simulation of digital circuits or low sensitivity analog circuits. It is optimized for circuits such as ROM and CAM.
8	This value is recommended for functional simulation optimized for Flash memory and mixed-signal circuits. There are three submodes: HSIMSPEED on page 150=8 . 4 , HSIMSPEED on page 150=8 . 7 and HSIMSPEED on page 150=8 . 8 to take advantage of circuit latency in large circuit blocks to speed up simulation. These submodes are especially useful in circuits that contain voltage regulators or charge pumps. Note: The interactive tree command only works in HSIMSPEED on page 150=8 . 7 .

Description

Higher [HSIMSPEED on page 150](#) values cause faster simulation speed at reduced simulation precision. [HSIMSPEED on page 150](#) can be selectively set in local subcircuits such that:

- Lower [HSIMSPEED on page 150](#) values are set for either of the following:
 - Analog subcircuits
 - Timing critical subcircuits
- Higher [HSIMSPEED on page 150](#) values can be set for digital subcircuits.

Caution: The performance of local [HSIMSPEED on page 150](#) setting may still rely on the global [HSIMSPEED on page 150](#) value set at the top level.

Some parameters that are automatically controlled by [HSIMSPEED on page 150](#) value cannot be locally set in subcircuits. They follow the global [HSIMSPEED on page 150](#) value.

Examples

When setting [HSIMSPEED on page 150](#) in a local subcircuit while setting [HSIMSPEED on page 150=5](#) in the top-level, the value of [HSIMIDDQ on page 98](#) remains at the value of 1 in this subcircuit: [HSIMSPEED on page 150](#) [HSIMIDDQ on page 98](#) can only be globally set. This global setting is determined by the top-level setting of [HSIMSPEED on page 150 =5](#).

HSIMSPF, HSIMSPEF

Specifies the DSPF and SPEF file names.

Syntax

```
.param HSIMRSPF = file_name and .param HSIMRSPF = file_name
```

Description

HSIMSPF defines the DSPF file for backannotation. HSIMSPEF defines the SPEF file for backannotation. In the following example, file1 contains parasitic RC in DSPF format, and file2 contains parasitic RC in SPEF format, and they are both back-annotated onto the transistor netlist.

HSIMSPFADDNETPINXY

Lets you insert a "*|P" net pin by specifying XY coordinates.

Syntax

```
.param HSIMSPFADDNETPINXY = "net X=xcoord Y=ycoord  
layer=layer pinxydist=xydist"
```

Argument	Description
<i>net</i>	Specifies the net name.
<i>X=xcoord</i>	X coordinate (in microns).
<i>Y=ycoord</i>	Y coordinate (in microns).

Description

This command finds the closest location to the specified XY coordinates on defined metal layer. It uses the following formula to calculate the distance:

$$d = \sqrt{dX^2 + dY^2}$$

where $dX = X_n - X_u$; $dY = Y_n - Y_u$

X_n and Y_n - node coordinates

X_u and Y_u - user-specified coordinates

HSIMSPF

Specifies the backannotation file formats.

Syntax

```
.param HSIMSPF=file1 HSIMSPEF=file2
```

Description

HSIMSPF defines the DSPF file for backannotation. HSIMSPEF defines the SPEF file for backannotation. In the following example, file1 contains parasitic RC in DSPF format, file2 contains parasitic RC in SPEF format, and they are both back-annotated onto the transistor netlist.

HSIMSPFCC

Turns on the processing of coupling capacitors in DSPF/SPEF files.

Syntax

```
.param HSIMSPFCC = <0 | 1>
```

Description

This command turns on the processing of coupling capacitors from DSPF/SPEF file. The default is 1 unless HSIMPWRA is set to 1. Then HSIMSPFCC defaults to 0.

HSIMSPFCCSCALE

Specifies the scale factor for coupling capacitors.

Syntax

```
.param HSIMSPFCCSCALE = scale_value
```

Description

Specifies the scale factor value by which to multiply the value of each coupling capacitor. When coupling capacitors are processed from DSPF/SPEF, the value of each coupling capacitor is multiplied by the HSIMSPFCCSCALE value. The default is 1.

HSIMSPFCHLEVEL

Specifies the hierarchical level to back annotate capacitance. For all nets at a level in the hierarchy below the `HSIMSPFCHLEVEL` value only capacitance is back-annotated. For nets that are at a level above or equal to the specified level, RC backannotation is performed.

Syntax

```
.param HSIMSPFCHLEVEL = level
```

HSIMSPFCMIN

Controls whether to perform RC or capacitance only back annotation.

Syntax

```
.param HSIMSPFCMIN = value
```

Description

For all nets with capacitance larger than or equal to the `HSIMSPFCMIN` value, RC backannotation is performed. For all nets with capacitance smaller than the specified value, only the capacitance is back-annotated.

HSIMSPFCNET

Back-annotes capacitance only by net name.

Syntax

```
.param HSIMSPFCNET = net_name
```

Description

For all net names matching the net name specified in `HSIMSPFCNET`, only capacitance is back-annotated. You can specify a wildcard character in the net name. The name must be quoted if wildcard characters are used. This command can be specified multiple times.

Examples

```
.param HSIMSPFCNET="x1/*"
```

All the nets matching “x1/*” have capacitance-only backannotation and all the other nets have full RC backannotation.

```
.param HSIMSPFCNET="x1/*" HSIMSPFRCNET=" "
```

All the nets matching “x1/*” have capacitance-only backannotation. All other nets have no annotation.

HSIMSPFDSJNET

Controls the reporting detail for disjointed components. This command controls the reporting of elements and pins for disjointed components according to the value you specify.

Syntax

```
.param HSIMSPFDSJNET = <0 | 1 | 2>
```

Argument	Description
0	Generates no report (the default).
1	Reports pins only.
2	Compiles a complete report including elements and pins on each disjointed component.

HSIMSPFFDELIM

Specifies SPF delimiter characters. In SPEF, HSIMSPFFDELIM specifies delimiter character used on fingered devices.

Syntax

```
.param HSIMSPFFDELIM = delimiter_character
```

HSIMSPFFEEDTHRU

Suppresses the merging of net ports.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMSPFFEEDTHRU****Syntax**

```
.param HSIMSPFFEEDTHRU = <0 | 1>
```

Argument	Description
0	Merges net ports (the default).
1	Does not merge net ports, and the DSPF file for a low-level subcircuit remains the same.

Description

Merging the net ports into one node may introduce some small inaccuracies. To separate the net ports set `HSIMSPFFEEDTHRU` to 1. The top-level subcircuits have the syntax shown in the following example.

Examples

```
* | NET VDD 10ff
* | P (VDD X 0.0)
* | I (X1:VDD X1 VDD X 0ff)
* | I (X1:VDD_1 X1 VDD_1 X 0ff)
* | I (X1:VDD_2 X1 VDD_2 X 0ff)
* | I (X2:VDD X2 VDD X 0ff)
* | I (X2:VDD_1 X2 VDD_1 X 0ff)
* | I (X2:VDD_2 X2 VDD_2 X 0ff)
R1 VDD X1:VDD 10.0
R2 VDD X2:VDD 10.0
R3 VDD X1:VDD_1 10.0
R4 VDD X2:VDD_1 10.0
R5 VDD X1:VDD_2 10.0
R6 VDD X2:VDD_2 10.0
```

After back-annotating the low-level subcircuit cell, the following two additional ports are added to the subcircuit.

- VDD_1
- VDD_2

When HSIM back-annotates the top-level subcircuit without port merging, all the instance pins are correctly matched with those of the top-level DSPF file as follows:

- X1:VDD, X1:VDD_1, X1:VDD_2
- X2:VDD, X2:VDD_1, X2:VDD_2

It is important that pin names, such as `VDD_2` in the previous example, are defined to match the instance pin names one level up in the hierarchy.

Otherwise, connections are not be made by the simulator. HSIM uses the following parameters to control backannotation for hierarchical SPF for feed-through nets:

```
.param HSIMSPFFEEDTHRU=1
```

HSIMSPFHLEVEL

Controls RC backannotation according to the hierarchy level.

Syntax

```
.param HSIMSPFHLEVEL = hierarchy_level_value
```

Description

For all nets at a level in the hierarchy above HSIMSPFHLEVEL, RC backannotation is performed. For nets lower than HSIMSPFHLEVEL, only the capacitance is back-annotated.

HSIMSPFKEEPNODECAP

Merges pre-layout node capacitance with the value from the SPF file.

Syntax

```
.param HSIMSPFKEEPNODECAP = <0 | 1>
```

Description

Specify HSIMSPFKEEPNODECAP = 1 to merge pre-layout node capacitance with the value from the SPF file. By default, HSIM overwrites the pre-layout net capacitance with what is contained in the SPF netlist.

HSIMSPFKS

Keeps backslash characters (\) in names in the SPF/SPEF file.

Syntax

```
.param HSIMSPFKS = <0 | 1>
```

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMSPFMSGLEVEL

Description

This command controls whether to keep backslashes (\) in names in SPF/SPEF file. The default is 0, which removes all of these characters.

HSIMSPFMSGLEVEL

Controls the printing of error and warning messages.

Syntax

```
.param HSIMSPFMSGLEVEL = <0 | 1 | 2 | 3>
```

Argument	Description
0	No net error or warning messages are printed.
1	Error messages are printed for discarded nets.
2	Error and warning messages are printed for discarded nets.
3	Error and warning messages are printed for all nets.

Description

An error is defined as a connectivity problem that causes the net to be discarded. A warning is a connectivity problem that can be corrected. A net can have both types of problems.

HSIMSPFMULTISUB

Allows reading multiple subcircuits in a single SPF file. Specify HSIMSPFMULTISUB = 1 to permit multiple .subckt statements in the SPF file. The default is 0.

Syntax

```
.param HSIMSPFMULTISUB = <0 | 1>
```

HSIMSPFNETPIN

Adds new instance pin connections to an SPF net.

Syntax

```
.param HSIMSPFNETPIN = "<net_name> <connect_point>
<new_instance_pin>"
```

Argument	Description
<i>net_name</i>	SPF net name.
<i>connect_point</i>	Can be a sub-node, instance pin, or pin of the SPF net.
<i>new_instance_pin</i>	Instance pin name that needs to be connected to the <i>connect_point</i> . For example: .param HSIMSPFNETIPIN="vdd vdd:3 iv1:p" that connects the positive terminal of iv1 current source to the vdd:3 sub-node after SPF backannotation. You must specify the iv1 current source in the netlist file.

Examples

To illustrate this methodology, [Example 12](#) provides a simple example of a prelayout statement.

Example 12

```
* prelayout.sp
.param HSIMSPF=vdd.dsfpf
VVDD VDD 0 1.8
MP A B VDD VDD p
...
```

and in the vdd.dsfpf file, the net VDD is specified as:

```
* |NET VDD 10ff
* |P (VDD_1 X 0.0)
* |P (VDD_2 X 0.0)
* |I (MP:S MP S X Off)
R1 VDD MP:S 10.0
R2 VDD_2 MP:S 10.0
```

[Figure 5 on page 162](#) shows the resulting postlayout netlist in which the VDD_1 and VDD_2 pins are merged to the VDD node.

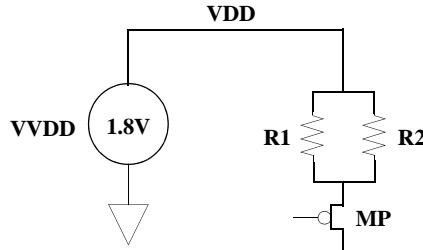
Chapter 4: HSIM® Commands in Alphabetical Order**HSIMSPFNETPINDEL**

Figure 5 Postlayout Netlist Circuit

To model the power net with packaging effects, the pads are connected to separate voltage sources with 1.79V and 1.81V values. This is accomplished via package instances defined by the **rlc** subcircuit, instead of connecting the **VDD_1** and **VDD_2** pads directly to the 1.8V voltage source. To achieve this, the following netlist file is used:

```
* prelayout_and_packaging.sp
.param HSIMSPF=vdd.dsfp
VVDD VDD 0 1.8
MP A B VDD VDD p
.subckt rlc in out r=r l=l c=c
R in i r
C i 0 c
L i out l
.ends
vvdd_1 vpad_1 0 1.79
vvdd_2 vpad_2 0 1.81
xvdd_1 vpad_1 vdd rlc r=1 l=1n c=10f
xvdd_2 vpad_2 vdd rlc r=1 l=1n c=10f
.param HSIMSPFMERGEPIN=0
.param HSIMSPFNETIPIN='vdd vdd_1 xvdd_1:out'
.param HSIMSPFNETIPIN='vdd vdd_2 xvdd_2:out'
...
```

HSIMSPFNETPINDEL

Converts one or more selected DSPF net pins into the internal node(s) of the net.

Syntax

```
.param HSIMSPFNETPINDEL = net_name net_pin1 net_pin2 ...
```

HSIMSPFNETWARNFILTER

Suppresses printing floating nets in the log file.

Syntax

```
.param HSIMSPFNETWARNFILTER = net_name_pattern
```

Description

In the DSPF and SPEF files, there can be many floating nets which only exist in the layout. These nets are typically of the name In_# where # is a number. It is desirable for these nets not to be listed in the .log file in order to reduce the number of warnings.

This command allows you to specify the name pattern of the floating nets, with wildcard character support, such that the nets are not listed. There are no warning message for the floating nets which match the name pattern(s). There are warning messages for all DSPF/SPEF nets which match the floating net pattern, but do not exist in the schematic netlist and are attached to the real instances.

HSIMSPFNOWARNONCAP

Controls whether or not to print a warning message when a SPF net cannot be found in the netlist.

Syntax

```
.param HSIMSPFNOWARNONCAP = value
```

Description

If a capacitance value is specified for HSIMSPFWARNOCAP, only those SPF nets with net capacitance greater than the specified value report a warning message when the nets do not exist. If you specify HSIMSPFWARNOCAP = 0, HSIM gives a warning for every net when the net does not exist. If you specify HSIMSPFWARNOCAP = 1.e-13, for example, HSIM only gives warning for the net with a capacitance value greater than 100 fF.

HSIMSPFPOS

Reads placement information from the specified file.

Syntax

```
.param HSIMSPFPOS = file_name
```

Description

This command reads in the position information from StarRC- generated placement file. This information is required for the hierarchical instances in the hierarchically extracted netlists.

HSIMSPFPOSSEARCH

Automatically detects instance placement information in a hierarchical DSPF.

Syntax

```
.param HSIMSPFPOSSEARCH = <0 | 1>
```

Description

HSIM requires instance subcircuit placement and orientation information when a DSPF is generated from hierarchical extraction. The placement and orientation data is needed to generate accurate RAGDS data for power reliability analysis (PWRA) or signal reliability analysis (SIGRA). If a DSPF is produced by a flat RC extraction, the HSIMSPFPOSSEARCH default is 0. This means no automatic detection of instance placement and orientation. When a DSPF is hierarchical, this command is automatically set to 1 to a output position file.

HSIMSPFPOSTL

Controls parasitic reduction.

Syntax

```
.param HSIMSPFPOSTL = <0 | 1 | 2 | 3>
```

Description

HSIMSPFPOSTL controls parasitic reduction just like [HSIMPOSTL](#) on [page 126](#). The only difference is that HSIMPOSTL applies reduction only to back-annotated RCs.

HSIMSPFPRINTPIN

Allows printing both pre- and post-layout results.

Syntax

```
.param HSIMSPFPRINTPIN = "<0|1|2|3>
<term=<1|2|3|4|...>,<1|2|3|4|...> ...> <node=node_name1
node_name2 ...>"
```

Argument	Description
0	Prints only the net node (the default).
1	Prints the top-level instance pin.
2	Prints hierarchical names.
3	Prints both the net node and top-level instance pins.
term=<1 2 3 4 ...>, <1 2 3 4 ...> ...	term specifies the device terminal for printing: For MOS devices: <ul style="list-style-type: none">■ 1 for drain■ 2 for gate■ 3 for source■ 4 for bulk For resistor devices: <ul style="list-style-type: none">■ 1 for drain■ 2 for source
node=node_name1 node_name2 ...	Prints only instance pins connected to the specified nodes.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMSPFPRINTSUBNODE****Description**

HSIM permits printing of both pre- and post-layout results. In pre-layout, the node name is aliased to the net name. This is accomplished in the following syntax:

```
.param HSIMSPFPRINTPIN = 1  
.print v(a)
```

In nWave, the format is:

```
v(a::xi7:i)
```

as shown in the following syntax:

```
<netname>::<inst>:<pin>
```

Examples

In the following example, all instance pins are printed.

```
HXI7:I  
.XI6-MM2:D
```

HSIMSPFPRINTSUBNODE

Enables sub-node printing of back-annotated SPF nets.

Syntax

```
.param HSIMSPFPRINTSUBNODE = <0 | 1>
```

Argument	Description
0	Disables sub-node printing (default).
1	Enables subnode printing.

HSIMSPFRAILRED

Lets you apply power net reduction in transient analysis without a power net reliability analysis (PWRA) license. Set the `HSIMSPFRAILRED` value to a positive integer to enable power net reduction.

Syntax

```
.param HSIMSPFRAILRED = <-1 | 0 | 1 | 2 | 3 | 4 | 5>
```

Argument	Description
-1	Specifies to apply the generic RC reduction algorithms controlled by HSIMPOSTL to both signal and power nets. This is the default.
0	Specifies to only run parallel-series power reduction.
1, 2, 3	Provides varying speed and accuracy tradeoffs between memory usage and simulator precision. The lower the integer value, the less aggressive the reduction.
4	Specifies the most aggressive power net reduction mode, applying the greatest effort and strongest reduction techniques. It has the following characteristics: <ul style="list-style-type: none"> ▪ Most significantly reduces the number of power net parasitic components. ▪ Does not preserve the underlying topology of the power network. ▪ Maintains a fairly close correlation for all impedances between the power pads and the device source or drain connections.
5	Specifies to consider the power network decoupled.

HSIMSPFRCNET

Runs backannotation for the specified nets.

Syntax

```
.param HSIMSPFPFRCNET = net_name
```

Description

Runs backannotation for all nets names that match the specific net name. You can use a wildcard character in the net name. The name must be quoted if wildcard characters are used. You can specify this command multiple times.

Examples

```
.param HSIMSPFRCNET="x1/x2/*"
```

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMSPFREPORTNOBA**

All the nets matching “x1/x2/*” run full RC backannotation. All the remaining nets have capacitance-only backannotation.

```
.param HSIMSPFRCNET="x1/*" HSIMSPFCNET=" "
```

All nets matching “x1/*” run full RC backannotation. All other nets have no annotation.

```
.param HSIMSPFRCNET="x1/*" HSIMSPFCNET="x2/*"
```

All nets matching “x1/*” run full RC backannotation. All nets matching “x2/*” will do capacitance-only backannotation. All other nets have no annotation.

HSIMSPFREPORTNOBA

Controls instance and node backannotation data reporting.

Syntax

```
.param HSIMSPFREPORTNOBA = <0 | 1 | 2 | 3>
```

Argument	Description
0	No data reporting (the default).
1	Reports instances containing no backannotation data. No nets inside the instances have SPF data.
2	Reports all nodes containing no SPF data.
3	Reports all nodes and ports containing no SPF data.

HSIMSPFSKIPNET

Does not run backannotation for specified nets.

Syntax

```
.param HSIMSPFSKIPNET = net_name
```

Description

Does not run backannotation for the net names that match the specified net name. You can use a wildcard character in the net name. The name must be quoted if wildcard characters are used. You can use this command multiple times.

HSIMSPFSKIPPNET

Controls backannotation for power nets that are connected to constant voltage source not to be back-annotated. The default is 0.

Syntax

```
.param HSIMSPFSKIPPNET = <0 | 1>
```

HSIMSPFSKIPSIGNET

Controls backannotation of signal nets that are connected to constant voltage source not to be back-annotated. The default is 0.

Syntax

```
.param HSIMSPFSKIPSIGNET = <0 | 1>
```

HSIMSPFSPLITCC

Splits coupling capacitors into two grounded capacitors with the same capacitance. The default is 0.

Syntax

```
.param HSIMSPFSPLITCC = <0 | 1>net_name
```

HSIMSPFEFTRIPLET

Sets the value for triplet analysis.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMSPFWARNFILE****Syntax**

```
.param HSIMSPEFTRIPLET = <1 | 2 | 3>
```

Argument	Description
1	Best case.
2	Typical case.
3	Worst case.

Description

In SPEF, parasitic RC values can be specified as a triplet for best (1), typical (2), and worst case (3). HSIM uses one value in a triplet for the analysis. This option specifies which value in a triplet to use. The default is 2, which uses the 2nd value (typical) in the triplet.

HSIMSPFWARNFILE

Writes SPF-related warnings to a file to redirect all SPF-related warnings to an <output_prefix>.spfwarn file. The default is 0.

Syntax

```
.param HSIMSPFWARNFILE = <0 | 1>
```

HSIMSPICE

Controls simulation precision for MOSFETs.

Syntax

```
.param HSIMSPICE = <0 | 1 | 2 | 3>
```

Argument	Description
0	This default value generates the Table Model during the HSIM preprocessing such that the efficient Table Model reduces the MOSFET model evaluation time during circuit simulation.

Argument	Description
1	This value suppresses the usage of Table Model and directly evaluates the exact MOSFET DC Model equations to calculate the I_{DS} current from the terminal node voltages.
2	This value further improves the simulation precision by employing the exact MOSFET Gate Capacitance Model equations, instead of the efficient model equations used when HSIMSPICE is set to either 0 or 1.
3	This setting precisely models and simulates the coupling effects between each pair of gate/drain/source/bulk terminals for each MOSFET affected by the HSIMSPICE = 3 setting.

Description

When HSIMSPICE = 0, (the default) the Table Model in general causes little precision loss unless the circuit functionality is sensitive to the small voltage variation that has similar magnitude of the voltage grid used in the Table Model.

However, the Table Model only contains the DC data of MOSFET I_{DS} current with respect to the terminal node voltages. The MOSFET gate capacitances are separately calculated by a set of efficient equations.

When HSIMSPICE = 1, the MOSFET gate capacitances are calculated in the same way as the setting of 0. In case of simulating high-sensitivity analog circuits, where the simulation precision is sensitive to the small difference in I_{DS} current caused by linear interpolation between grid points of table data, it may require choosing a very small voltage grid resolution to improve the precision.

This value may be impractical due to a very large number of table data, which also requires a long time to generate in the preprocessing. Although the setting of 1 improves the precision, the simulation speed will inevitably slow down due to the more time-consuming analytical MOSFET Model evaluation. So this setting is suggested in simulating small and highly sensitive analog circuits, or in local subcircuit blocks where the simulation precision is sensitive to the Table Model grid resolution. When set locally, this value only covers the MOSFETs affected by the setting.

When HSIMSPICE = 2, the model equations for MOSFET I_{DS} current are the same when HSIMSPICE is set to 1 or 2. For simulating circuits such as a charge-pump voltage generator, analog-to-digital (A/D), or digital-to-analog (D/A) converters, it is recommended that HSIMSPICE = 2 be used.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMSPISCIUNITERR**

When HSIMSPICE = 2, it includes the most precise setting includes all the models specified by HSIMSPICE = 2.

HSIMSPISCIUNITERR

Allows mixed syntax values in a simulation.

Syntax

```
.param HSIMSPISCIUNITERR = <0 | 1>
```

Description

This command handles values that are mixed scientific notation and units. Mixed notation such as 1.0E-6F is recognized as 1.0E-21 however, this is an illegal SPICE syntax format. Set HSIMSPISCIUNITERR =1 to cause HSIM to error out when parsing mixed syntax.

If mixed syntax is read by a DSPF file, it is converted as is; even if the HSIMSPISCIUNITERR =1 is set.

HSIMSTEADYCURRENT

Skips the simulation of idle subcircuits.

Syntax

```
.param HSIMSTEADYCURRENT = value
```

Description

If HSIMSNCS is set to 1, the simulation of idle subcircuits is skipped to boost the simulation speed. A subcircuit is treated as idle if every node in the subcircuit is idle. The criterion to determine whether a node is idle is by checking the node voltage change shown in [HSIMSNCS on page 148](#).

The higher the value of HSIMSTEADYCRRENT, the more likely a subcircuit becomes idle, which enables the simulation to run faster.

In HSIM, HSIMSTEADYCRRENT can be set in any local subcircuit, either subcircuit definition or subcircuit instance, such that it has a local effect on the associated subcircuit. The default value is 10nA.

HSIMSTEP2TAUMAX

Sets the tstep value in the .tran command as specified by the HSIMSTEPTAUMAX value.

Syntax

```
.param HSIMSTEP2TAUMAX = <0 | 1>
```

Description

If HSIMSTEP2TAUMAX=1, the tstep value in the .tran command is used as the HSIMTAUMAX value. If multiple tstep/tstop pairs are specified in the .tran command, HSIM automatically sets the HSIMTAUMAX command as a PWC function of the step/stop time pairs specified in .tran. The default is 0.

Examples

```
.tran 0.01ns 10ns 0.05ns 100ns
.param HSIMSTEP2TAUMAX=1
=> .param HSIMTAUMAX=pwc(0ns, 0.01ns, 10ns, 0.05ns)
```

HSIMSTOPIFDCFAIL

Set HSIMSTOPIFDCFAIL = 1 if you want to stop a simulation when DC does not converge. The default is 0.

Syntax

```
.param HSIMSTOPIFDCFAIL = < 0 | 1 | 2 >
```

Description

When DC does not converge, if HSIMSTOPIFDCFAIL is set to 1, simulation will directly stop and quit. If HSIMSTOPIFDCFAIL is set to 2, MaxDV is reported when its value is greater than HSIMVDD (if set) or power supply (if HSIMVDD is not set), then simulation will stop and quit. Default is 0.

HSIMSTOPIFNODC

Stops a simulation at the end of DC initialization.

Syntax

```
.param HSIMSTOPIFNODC = <0 | 1>
```

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMSTNAME

Description

Specify HSIMSTOPIFNODC = 1 to stop the simulation at the end of DC initialization if no DC solution is found. If DC does not resolve all nodes, an error is reported. The default is 0.

HSIMSTNAME

Provides a mechanism to match terminal names in prelayout and postlayout netlists.

Syntax

```
.param HSIMSTNAME = "sub=sub_name pre_term1=post_term1  
pre_term2=post_term2 ..."
```

Description

This command is used when terminal names of a prelayout subcircuit do not match the terminal names in the postlayout instance pin definition. It provides a mechanism for one-to-one mapping of terminal names for a particular subcircuit.

Examples

Pre-layout netlist :

```
.subckt nmos drn gate src body  
...  
.ends nmos  
  
xi1 n1 n2 n3 n4 nmos
```

DSPF file :

```
* |NET n1  
* |I xi1:a  
  
* |NET n2  
* |I xi1:b  
  
* |NET n3  
* |I xi1:c  
  
* |NET n4  
* |I xi1:d
```

In the previous example, subcircuit terminal names drn, gate, src, and body correspond to a, b, c, and d in the DSPF file, respectively. The following HSIMSTNAME command allows for terminal name matching.

```
.param HSIMSTNAME = "sub=nmos drn=a gate=b src=c body=d"
```

HSIMSTSWAP

Provides a mechanism to match terminals in pre- and post-layout netlists.

Syntax

```
.param HSIMSTSWAP="sub=sub_name port# port#"      (where the  
port# starts at 1)
```

Description

This command is used when terminals in a pre-layout subcircuit definition are swapped in the post-layout instance pin definition. It provides a mechanism to indicate that two particular terminals of a given subcircuit are swapped. An example is given below:

Examples

Pre-layout netlist:

```
.subckt nmos drn gate src body  
...  
.ends nmos  
  
xi1 n1 n2 n3 n4 nmos
```

DSPF file:

```
* |NET n1  
* |I xi1:src  
  
* |NET n2  
* |I xi1:gate  
  
* |NET n3  
* |I xi1:drn  
  
* |NET n4  
* |I xi1:body
```

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMSUBBUSDELIM**

In the example above, the drn and src terminals are swapped in the DSPEF file, causing a mismatch during backannotation. In order to avoid this problem, use the following HSIMSTSWAP command:

```
.param HSIMSTSWAP="sub=nmos 1 3"
```

HSIMSUBBUSDELIM

Controls how HSIM expands buses in a .subckt line in a netlist.

Syntax

```
.param HSIMSUBBUSDELIM = delim_character
```

Description

This command specifies the bus notation used in a SPICE netlist and is referenced in two locations:

- hdl test interface: A bus signal name in a testbench module is broken down into bit signal names with this bus notation so that HSIM can find the signal connection in the SPICE netlist.
- SPEF: If SPEF is specified, the parser converts all bus names in SPEF input so that they can be matched in the HSIM database for back annotation.

Examples

If a bus is named A<0:3>, applying .param HSIMSUBBUSDELIM=<> becomes:

A<0>, A<1>, A<3>

HSIMTAUMAX

Defines an upper limit for the simulation time step size.

Syntax

```
.param HSIMTAUMAX = value
```

Description

This command defines an upper limit for the time step size that safeguards the unusual situation when a sudden fast transition is self-triggered within an almost idle subcircuit. The HSIMTAUMAX setting in general has little effect on simulation speed, except in the case of low-frequency circuit simulation. Due to

the large time constant of low-frequency circuits, which can allow larger time step size, the 2 ns default value for HSIMTAUMAX could be too small. It is recommended to increase the value to a similar order of the time constant of the circuit.

In HSIM, you can set HSIMTAUMAX at the top-level such that it has a global effect on the entire circuit. You can also specify it locally; however, the locally applied value must be more conservative. If HSIMTAUMAX is more aggressively applied, no local effects are derived.

HSIMTIMEFORMAT

Controls the print out format of the CPU time in seconds, minutes, hours, or days.

Syntax

```
.param HSIMTIMEFORMAT = <1|2|3|4|7|8|15>
```

Argument	Description
1	Seconds only (the default)
2	Minutes only
3	Minutes/seconds
4	Hours only
7	Hours/minutes/seconds
8	Days only
15	Days/hours/minutes/seconds

HSIMTIMESCALE

Changes the default time unit.

Syntax

```
.param HSIMTIMESCALE = value
```

Description

The time unit in HSIM is 1 picosecond (1E-12 second) by default. You can change this default time unit by setting the `HSIMTIMESCALE` global parameter `HSIMTIMESCALE`. Refer to [Table 5](#).

Table 5 Simulation Time-Scale Control Parameters

Command	Description
<code>HSIMTIMESCALE=10</code>	Sets the time unit to 10 picoseconds.
<code>HSIMTIMESCALE=0.1</code>	Sets the time unit to 0.1 picosecond.

The default value of 1 should have adequate time precision resolution for practical applications in which the operating frequency is likely to be less than 100 GHz. The time unit value limits the lower bound of time stepsize. Each time stepsize must be an integer multiple of the time unit.

Examples

Example 13 Setting `HSIMTIMESCALE=100` limits each time stepsize to be:

`100 ps, 200 ps, ... 1 ns, 1.1 ns, ...`

It is recommended to adjust the value of `HSIMTIMESCALE` to the same order of the required time resolution.

HSIMTLINEDV

Controls the simulation time step for transmission lines.

Syntax

```
.param HSIMTLINEDV = value
```

Description

This command affects selecting timestep when simulating transmission lines. It provides direct influence on precision control of the signal propagation through transmission lines.

HSIMTLINEDV is similar to HSIMTALLOWEDDV, but HSIMTLINEDV only affects the lossy transmission line. The separate command is necessary because the transmission line behavior is determined both by voltage and current wave propagation in the transmission line. The current quantities have different dimensions than voltage quantities, so HSIMTALLOWEDDV is not a proper parameter to set current waves tolerances. HSIMTLINEDV sets relative tolerance for current waves.

A smaller HSIMTLINEDV value leads to a smaller time step, which causes higher precision and slower simulation speed and vice versa. The allowed parameter range is from 1.0e-3 to 1.0e+3. The default value is 1.0.

HSIMTOP

Specifies the top-level subcircuit name.

Syntax

```
.param HSIMTOP = top_subcircuit_name
```

Description

This command specifies the top-level subcircuit name. It is an alternation to the -top command line option and applies globally. The default is null.

When using this command, the following restriction applies: HSIMTOP and HSIMNTLFMT must be the first commands in the input.

HSIMTRAPEZOIDAL

Enables the algorithm to simulate oscillators with inductors.

Syntax

```
.param HSIMTRAPEZOIDAL = <0 | 1>
```

Description

To simulate oscillators containing inductors, you must set HSIMTRAPEZOIDAL = 1. This command invokes the second-order numerical integration algorithm: the Trapezoidal Algorithm. Though more suitable for simulating the RLC oscillator circuit, the Trapezoidal Algorithm may cause artificial oscillation in high-impedance nodes. This command can be set globally or locally in selected subcircuits. The default is 0.

HSIMUFILIB

Specifies the complied library location.

Syntax

```
.param HSIMUFILIB=compiled_library_path
```

Description

The UFI code typically consists in the following files:

- **UFI.c**: support code, do not edit
- **b3ufimain.c**: support code, do not edit
- **UFI.h**: header file, do not edit
- **b3ufiread.c**: edit to define custom parameters
- **b3ufiset.c**: edit to define custom parameters
- **b3ufi.h**: edit to define custom parameters
- **b3ufild.c**: edit to define custom program/erase equations

Examples

```
.param HSIMUFILIB=./ufi_code/ufi.so
```

HSIMUSEHM

Specifies a list of C model names.

Syntax

```
.param HSIMUSEHM = model_names
```

Description

Instantiation of a C functional model is defined as a regular subckt instantiation. If both a subckt definition and a C model exist, HSIM uses the subckt definition by default. You can use the **HSIMUSEHM** command to specify a list of C model names so that HSIM uses the C model instead of subckt definition if both exist.

Examples

The following example is a SPICE netlist file for an entire circuit.

Example 14

```
** SPICE netlist file of the entire circuit
* C-functional model inv_D, instead of electrical subcircuit *
inv_D,
* is used for instance X3 if the following line is *
** uncommented.
*.param HSIMUSEHM=inv_D
.global vdd
vdd 0 3.0
Vin in 0 pulse(0 3.0 1n 0.1n 0.1n 10n 20n)
X1 in n1 inv_E
* C functional model inv_C is used for X2
X2 n1 n2 inv_C
* C-functional model inv_D is used for X3 if HSIMUSEHM=inv_D is
defined
X3 n2 n3 inv_D
.tran 1n 100n

.subckt inv_E in out
MP out in vdd vdd np w=2u l=0.18u
MN out in 0 0 nm w=1u l=0.18u
.ends
.subckt inv_D in out
MP out in vdd vdd np w=2u l=0.18u
MN out in 0 0 nm w=1u l=0.18u
.ends
.end
** end of SPICE netlist file

/* .c file with C functional model */
hsimC_model ()
{
    pHMMODEL pInv1, pInv2;

    pInv1=hmCreateModel ("inv_C", inv1);
    pInv2=hmCreateModel ("inv_D", inv2);
    .....
}

void inv1 ()
{
    ...
}

void inv2 ()
```

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMUSEHMINST**

```
{  
...  
}  
/* end of .c file */
```

HSIMUSEHMINST

Specifies a C model instantiation.

Syntax

```
.param HSIMUSEHMINST = C_model_instantiation
```

Description

When both a subckt definition and a C model exist, device patterns can be specified to use C model instantiation.

When subckt and C model definitions exist for inv_D, the commands in the example can be used to force C model instantiation on matching devices.

Examples

The following example is a SPICE netlist file for an entire circuit.

```

** SPICE netlist file of the entire circuit
* C-functional model inv_D, instead of electrical subcircuit *
inv_D,
* is used for instance X3 if the following line is *
** uncommented.
*.param HSIMUSEHM=inv_D
.global vdd
vdd 0 3.0
Vin in 0 pulse(0 3.0 1n 0.1n 0.1n 10n 20n)
X1 in n1 inv_E
* C functional model inv_C is used for X2
X2 n1 n2 inv_C
* C-functional model inv_D is used for X3 if HSIMUSEHM=inv_D is
defined
X3 n2 n3 inv_D
.tran 1n 100n

.subckt inv_E in out
MP out in vdd vdd np w=2u l=0.18u
MN out in 0 0 nm w=1u l=0.18u
.ends
.subckt inv_D in out
MP out in vdd vdd np w=2u l=0.18u
MN out in 0 0 nm w=1u l=0.18u
.ends
.end
** end of SPICE netlist file

/* .c file with C functional model */
hsimC_model ()
{
    pHMMODEL pInv1, pInv2;

    pInv1=hmCreateModel ("inv_C", inv1);
    pInv2=hmCreateModel ("inv_D", inv2);
    .....
}

void inv1 ()
{
    .....
}

void inv2 ()
{

```

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMUSEPREVIOUSDC

```
....  
}  
/* end of .c file */  
.param HSIMUSEHMINST=x1.xa  
.param HSIMUSEHMINST=x1.xb.*
```

HSIMUSEPREVIOUSDC

Reuses the previous DC solution.

Syntax

```
.param HSIMUSEHPREVIOUSDC = <0 | 1>
```

Description

When performing a large number of DC steps over a voltage source range it is often practical to reuse the previous solution to seed the subsequent solution. To have HSIM reuse the previous solution as a seed for the next incremental DC solution, set `HSIMUSEPREVIOUS = 1`. The default value is 0, which results in a full independent DC analysis being performed for every increment.

HSIMUSERLIB

Specifies a UMI model library path.

Syntax

```
.param HSIMUSEHMINST = library_name
```

Examples

In the following example, `user.so` is the name holder for the dynamic library.

```
param HSIMUSERLIB=". /user.so"
```

HSIMUSEVA

Specifies the Verilog-A module name.

Syntax

```
param HSIMUSEVA = module_name
```

Description

In a netlist, a Verilog-A module can have the same name as a SPICE subcircuit. By default, HSIM uses the SPICE subcircuit name in the simulation. If you specify `HSIMUSEVA=module_name`, HSIM uses the Verilog-A module instead of its SPICE counterpart.

HSIMUSEVATABLE

Activates the table model for Verilog-A models. Set `HSIMUSEVATABLE = 1` to activate the table model for Verilog-A models. The default is 0.

Syntax

```
.param HSIMUSEVATABLE= <0 | 1>
```

HSIMUTFCOMPRESSLEVEL

Controls the amount of compression in `utf` output.

Syntax

```
.param HSIMUTFCOMPRESSLEVEL=compression_level
```

Description

If simulations runs take longer in the `utf` format than they do in other output formats, you can use the `HSIMUTFCOMPRESSLEVEL` command to control the amount of compression in the output. For example:

```
.param HSIMUTFCOMPRESSLEVEL=compression_level_number
```

where `compression_level_number` can be an integer 0 through 9, with 9 specifying the greatest amount of compression. The default is 0, which means no compression.

You can also increase the run time for `utf` output by directing the output to a local disk, such as `/tmp`.

Note: Contact Synopsys to obtain the latest `libUTF.so` library.

HSIMV2S

Invokes the v2s utility during netlist parsing.

Syntax

```
.param HSIMV2S="<options> filename"
```

Description

This command invokes the v2s utility during netlist parsing. It is an alternative to running v2s manually in order to obtain the converted output and including the file in the SPICE input. Instead, the entire procedure is done inside HSIM.

HSIMV2S specifies the v2s arguments and tells HSIM that certain netlist information is read from Verilog inputs. It inserts default arguments shown in the following table if they are not explicitly specified.

Table 6 HSIMV2S Option Defaults

Option	Default
-s	Provided for compatibility with v2s. Will be ignored.
-const0	HSIMLOGICHV
-const1	HSIMLOGICLV
-bn	HSIMBUSDELIMITER
-top	HSIMTOP
-o	Provided for compatibility with v2s. (v2s.spi will be ignored)

Examples

```
.param HSIMV2S="top.v -o top.spi"
```

This example converts modules defined in top.v to subckt definitions and writes the output to top.spi.

Note: Refer to the demo at hsim/tutorial/v2s.

HSIMV2SD

Reads the Verilog netlist directly instead of invoking v2s.

Syntax

```
.param HSIMV2S=<options> filename"
```

Description

This utility works in a similar manner to HSIMV2S however, HSIMV2SD reads the Verilog netlist directly instead of invoking v2s. Additionally, the entire process is done within HSIM. All HSIMV2S options can be used with HSIMV2SD except the -o option as described in the following table.

Table 7 HSIMV2S Option Defaults

option	Default
-s	Provided for compatibility with v2s. Will be ignored.
-const0	HSIMLOGICHV
-const1	HSIMLOGICLV
-bn	HSIMBUSDELIMITER
-top	HSIMTOP
-o	Provided for compatibility with v2s. (v2s.spi will be ignored)

Examples

```
.param HSIMV2SD='verilog_netlist' -top top_sub_circuit_name.
```

HSIMVABRANCHPART

Partitions current nodes.

Syntax

```
.param HSIMVAPARTITION="module_name"
```

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMVACCBMATCH**Description**

This command partitions current nodes in addition to the HSIM default nodes. By default, HSIM considers current branch nodes as inout regardless of their declaration in a module. The `HSIMVABRANCHPART` command forces HSIM to consider all nodes.

HSIMVACCBMATCH

Partitions current nodes.

Syntax

```
.param HSIMVACCBMATCH=<0 | 1>
```

Description

Set `HSIMVACCBMATCH=1` to enable isomorphic matching for Verilog-A modules. The default is 0. If a design has many leaf level Verilog-A modules with no internal state, set this parameter to 1 to enable matching for better performance and memory usage. In the legacy Verilog-A solution, the default is 1.

HSIMVACROSSTOL

Adds the proper time point for the Verilog-A `cross` function.

Syntax

```
.param HSIMVACROSSTOL= value
```

Description

`HSIMVACROSSTOL` is for adding the proper time point in HSIM for the Verilog-A `cross` function by defining the tolerance of the time specification. The default value is 100p. For a tight simulation with a high accuracy demand, the default value of 100p might be too large and should be set to a smaller value.

HSIMVACROSSVTOL

Adds the voltage tolerance for the Verilog-A `cross` function.

Syntax

```
.param HSIMVACROSSVTOL = value
```

Description

Use `HSIMVACROSSVTOL` to add voltage tolerance in HSIM for Verilog-A the cross function. The default is +0.1V. For a tight simulation with a high accuracy demand, the default value of 0.1V may be too large and should be set to a smaller value.

HSIMVAPARTITION

Partitions a Verilog-A module.

Syntax

```
.param HSIMVAPARTITION = <0 | 1>
```

Description

When you set `HSIMVAPARTITION=1`, HSIM automatically finds Hi-Z nodes in the module for partitioning. In the legacy Verilog-A solution, HSIM re-orders the ports and internal nodes of the module so that inout and output nodes are channel-connected and separated from input nodes. In order for partitioning to work efficiently, the input andinout,output ports must be defined correctly. There should not be any current branches from input ports. The default is 1.

HSIMVAPRINTVAR

Prints variables in Verilog-A instances.

Syntax

```
.param HSIMVAPRINTVAR= <0 | 1>
```

Description

When `HSIMVAPRINTVAR=0` (the default), no Verilog-A instance variables are printed. Set `HSIMVAPRINTVAR=1` to print all variables in Verilog-A instances.

HSIMVATBLERANGE

Defines the range of the Verilog-A table. This command defines the range of the Verilog-A table. The default value is 2*Vdd.

Syntax

```
.param HSIMVATBLERANGE = value
```

HSIMVATABLESIZE

Defines the number of elements in the Verilog-A table. This command defines the number of elements in the Verilog-A table. The default is 100.

Syntax

```
.param HSIMVATBLERANGE = value
```

HSIMVBS3END

Specifies the ending Vbs voltage. with a positive value.

Syntax

```
.param HSIMVBS3END = value
```

HSIMVBS3START

Specifies the starting Vbs voltage. The specified value must be non-positive.

Syntax

```
.param HSIMVBS3START = value
```

HSIMVBSEND

Specifies the HSIMVDD value for the BSIM3 model.

Syntax

```
.param HSIMVBSEND = value
```

Description

The default value is HSIMVDD+1. For the BSIM3 model, the default value for HSIMVBSEND is 20.

HSIMVCD2VEC

Specifies the VCD and signal information file names.

Syntax

```
.param HSIMVCD2VEC = "-nvcd vcd_file1-nsig sig_file1"
.param HSIMVCD2VEC = "-nvcd vcd_file2-nsig sig_file2"
...
.param HSIMVCD2VEC = "-nvcd vcd_file_n-nsig sig_file_n"
```

Description

vcd_file is the VCD file name. *sig_file* is the signal information file name. This file maps the digital signals in the VCD file to the analog signals in the design netlist. You specify the VCD file and signal information files in the HSIM netlist. You can specify multiple files.

For details on signal information file, see [Using the Signal Information File on page 306](#).

See the HSIM log file if there are errors processing the VCD or signal information files. Simulation stops if there is an error processing these files.

HSIMVDD

Influences supply voltage control.

Syntax

```
.param HSIMVDD = value
```

Description

Voltages are influenced by the supply voltage under the control of HSIMVDD. The default for HSIMVDD is 3V.

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMVDSEND**

In cases where HSIM cannot easily find DC convergence, set [HSIMAUTOVDD on page 64](#) = 1 to aid HSIM. These cases include situations where there are POWER nets present in the netlist, and where there are multiple power supplies used for different parts of transistors.

HSIMVDSEND

Specifies the VDS ending value.

Syntax

```
.param HSIMVDSEND = value
```

Description

HSIMVDSEND IS A MOSFET table control command that specifies the VDS voltage ending value. The default value is HSIMVDD+1.

HSIMVGSEND

Specifies the VGS ending value.

Syntax

```
.param HSIMVGSEND = value
```

Description

HSIMVGSEND IS A MOSFET table control command that specifies the VGS voltage ending value. The default value is HSIMVDD+1.

HSIMVECTORFILE

Specifies the digital vector input and output files.

Syntax

```
HSIMVECTORFILE = file_name
```

Description

Specifies the digital vector input and output files for HSIM. The default unit of time for handling vector files is nanoseconds. Any number of vector files can be

specified by repeatedly specifying HSIMVECTORFILE. To change the unit of time, such as from nanosecond to picoseconds, use the tunit statement. Refer to [tunit on page 302](#). The maximum number of vector file characters per line is 2048.

HSIMVERILOGA

Specifies a Verilog-A modules file.

Syntax

```
.param HSIMVERILOGA = v-a_module_file_path
```

Description

To include Verilog-A modules in a SPICE netlist, use the HSIMVERILOGA command to specify the source file name. The path to the file name can be relative or absolute.

The Verilog-A source file is compiled and the model is instantiated in the circuit in the same format as a SPICE subcircuit.

HSIMVHI, HSIMVHL

Specifies the dynamic threshold voltage values.

Syntax

```
.print HSIMVHI =
  "vhi_1*(logic_exp1)+vhi_2*(logic_exp2)<+vhi_3*
  (logic_exp3) ...>"
```

Argument	Description
vhi_1, vhi_2, vhi_3	Voltage values.
logic_exp1, logic_exp2, logic_exp3	Can be expressions such as b0*b1==1 when both b0 and b1 are 1s, or b0*b1==0, when both b0 and b1 are 0s. b0 and b1 are the netlist nodes.

Description

The vector check process uses the fixed threshold voltage values throughout the entire simulation. However, these fixed threshold voltage values cannot capture the circuit dynamic behavior in all cases. Use .print HSIMVHI and .print HSIMVHL to specify dynamic threshold voltage values for logic HIGH and logic LOW states to resolve this problem.

HSIMVHTH

Specifies the voltage threshold for the HIGH logic state. The default value is 9.7*VDD.

Syntax

```
.param HSIMVHTH = value
```

HSIMVHTHRATIO

Specifies the ratio for the voltage threshold for the HIGH logic state.

Syntax

```
.param HSIMVHTHRATIO = value
```

Description

Specifies the ratio for the voltage threshold for the HIGH logic state at a ratio other than 0.7*V_{DD}. The default is 0.7. This parameter is specified globally. For example:

```
.param hsimautovdd=1 hsimvlthratio=0.4 hsimvhthratio=0.6
```

The 3.0 supply domain is 1.2 vlhs / 1.8 vhth and the 5.0 supply domain is 2.0 vlhs / 3.0 vhth threshold levels.

HSIMVLO

Specifies the dynamic threshold voltage for the LOW logic state. The dynamic threshold voltage values for the logic HIGH and logic LOW states can be limited to the selected signal by using vector file masks.

Syntax

```
.print HSIMVLO =
  "vhi_1*(logic_exp1)+vhi_2*(logic_exp2)<+vhi_3*
  (logic_exp3) ...>"
```

Argument	Description
vhi_1, vhi_2, vhi_3	Voltage values.
logic_exp1, logic_exp2, logic_exp3	Can be expressions such as b0*b1==1 when both b0 and b1 are 1s, or b0*b1==0, when both b0 and b1 are 0s. b0 and b1 are the netlist nodes.

Examples

```
signal in1_1 in1_2 in2_1 in2_2 out1 out2 outx invitout
radix 11111111
io iiiioooo
mask m3 invitout
vhth HSIMVHI m3
vlth HSIMVLO m3
```

In this example, the .print HSIMVHI=... and .print HSIMVLO commands are applied to signal invitout only. outx, out1, and out2 continue using the default threshold voltage values.

HSIMVLTH

Specifies the ratio for the voltage threshold for the LOW logic state. The default is 0.7*VDD.

Syntax

```
.param HSIMVLTH = value
```

HSIMVLTHRATIO

Specifies the ratio for the voltage threshold for the LOW logic state at a ratio other than 0.3*V_{DD}.

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMVPRECISION**Syntax**

```
.param HSIMVLTHRATIO = value
```

Description

Specifies the ratio for the voltage threshold for the LOW logic state at a ratio other than $0.3 \times V_{DD}$. This parameter is specified globally. The default is 0.3.

HSIMVPRECISION

Controls voltage precision. For details about the HSIMVPRECISION settings, see [HSIM Quick Start on page 12](#).

Syntax

```
.param HSIMVPRECISION = <0 | 1 | 2 | 3 | 4 | 5>
```

Description

HSIMVPRECISION is a macro command that controls voltage precision. The concept of voltage precision is based on identifying the voltage level/contribution that a simulator considers negligible. Simulators, including traditional SPICE, need to set these type of thresholds to ensure adequate throughput is maintained during the millions of mathematical calculations resolving Kirhoff's voltage laws.

A value of 0 specifies the fastest performance, while a value of 5 specifies the most accurate precision. The default is 2.

The HSIMVPRECISION command controls:

- Maximum voltage between time steps (relative to [HSIMVDD on page 191](#)).
- Isometric matching based on port voltage matching (see [HSIMPORTV on page 126](#)).
- Independent voltage sources sampling.
- Absolute maximum time step (see [HSIMTAUMAX on page 176](#)).
- Voltage printing precision (see [HSIMOUTPUTVRES on page 123](#)).

HSIMVSRCDV

Adjusts the simulation time step on the primary input voltage source nodes.

Syntax

```
.param HSIMVSRCDV = value
```

Description

Similar to HSIMALLOWEDDV, you can use the HSIMVSRCDV command to adjust the simulation time step on the primary input voltage source nodes by setting the voltage change limit. The internally-calculated HSIMVSRCDV value for a particular input voltage source node is one tenth of abs(Vmax-Vmin). If a circuit has multiple voltage sources, there could be different internally-determined HSIMVSRCDV values.

A global voltage change limit for all voltage source nodes can be set by setting the HSIMVSRCDV value. The smaller the value, the slower the simulation speed.

HSIMVSRCCLMIN

Shorts any inductor that connects to a voltage source and is smaller than the specified value.

Syntax

```
.param HSIMVSRCCLMIN = value
```

Description

The HSIMVSRCCLMIN global command is similar to HSIMLMIN, except it applies only to inductor networks that connect to voltage sources. HSIMVSRCCLMIN can only be set globally. Any inductor that connects to a voltage source and is smaller than the HSIMVSRCCLMIN value is shorted. The default value is 1e-10.

HSIMVSRCRMIN

Shorts any resistor that connects to a voltage source and is smaller than the specified value.

Syntax

```
.param HSIMVSRCRMIN = value
```

Chapter 4: HSIM® Commands in Alphabetical Order
HSIMVSRCSYNC**Description**

The HSIMVSRCRCLMIN global command is similar to HSIMRMIN, except it applies only to resistor networks that connect to voltage sources. HSIMVSRCRMIN can only be set globally. Any resistor that connects to a voltage source and is smaller than the HSIMVSRCRMIN value is shorted. The default value is 0.1.

HSIMVSRCSYNC

Synchronizes sampling points for different sinusoidal voltage sources.

Syntax

```
param HSIMVSRSYNC = <0 | 1>
```

Argument	Description
0	Specifies that the sampling points are generated independently for different voltage sources (the default).
1	Specifies that the sampling points are generated only for voltage sources with highest frequency and used for voltage sources at other frequencies.

Description

When HSIMVSRCSYNC=0 (the default), HSIM generates the sampling points for different sinusoidal voltage sources independently, which might result in oversampling and reduce performance. If you set HSIMVSRCSYNC=1, HSIM generates sampling points for the voltage source with highest frequency only, and applies them to voltage sources at other frequencies.

HSIMWARNFILTER

Filters warnings. Add HSIMWARNFILTER at the beginning of the top-level netlist in order to have an effect on the warnings displayed.

Syntax

```
.param HSIMWARNFILTER = "[max_warnings:] warning_text"
```

Argument	Description
<i>max_warnings</i> :	Limits the number of warnings to suppress.
<i>warning_text</i>	Specifies the warning text to suppress from printing.

Examples

```
.param HSIMWARNFILTER="remove unused subckt"
```

Removes all warning messages that contain the remove unused subckt text.

```
.param HSIMWARNFILTER="10:matched devices"
```

Prints only the first 10 warnings that contain the matched devices text and suppresses any other warnings that contain that text.

HSIMWARNIFNODC

Warns if DC initialization does not resolve all nodes.

Syntax

```
.param HSIMWARNIFNODC = <0 | 1>
```

Description

Set HSIMWARNIFNODC = 1 warns if DC initialization does not resolve all nodes. The process reports the warning in the log file. The default is 0.

HSIMWARNSTOP

Terminates simulation upon reaching a user-specified number of warnings.

Syntax

```
.param HSIMWARNSTOP = <0 | 1>
```

Description

Set HSIMWARNSTOP = 1 to terminate simulation upon reaching a user-specified number of warnings. The maximum number of warnings can be set by setting:

Chapter 4: HSIM® Commands in Alphabetical Order**HSIMWRAPPERSUB**

```
.option warnlimit=#
```

You must set `HSIMWARNSTOP = 1` in order for `.option warnlimit` to take any effect. The default is 0.

HSIMWRAPPERSUB

Specifies a subcircuit name/pattern as a device model name.

`HSIMWRAPPERSUB` specifies a subcircuit name/pattern that can be used as device model name.

Syntax

```
.param HSIMWRAPPERSUB = subckt_name
```

Examples

```
.param HSIMWRAPPERSUB=nfet
.subckt nfet d g s b
...
.ends
M1 d g s b nfet w=.. l=..
```

With `HSIMWRAPPERSUB` specified, HSIM treats M1 as a subckt instance of `subckt nfet` instead of warning about model “nfet not found ...”

HSIMX

`HSIMX` enables a macro command that provides a simple way to turn on command settings appropriate for recent technologies, as well as recently implemented HSIM technologies. If you use `HSIMSPEED=3` to `HSIMSPEED=5` in existing designs, `HSIMX` provides equal or better performance, with similar accuracy, in new designs.

Note: The `HSIMX` command is equivalent to the `hsim -x` command line option.

Syntax

```
.param HSIMX = <0|1>
```

Argument	Description
0	Turns HSIMX off (the default).
1	Turns HSIMX on.

HSIMXVERSION

Determines which HSIMX command set HSIM uses. The default is the current major version of the HSIM binary.

Syntax

```
.param HSIMXVERSION = version
```

Argument	Description
<i>version</i>	Specify the version in the following format: <i>yyyy.mm</i> ; for example, 2011.09. Note that you cannot specify a major version prior to the 2011.09 release.

HSIMZV

Controls how to merge the terminals of zero-volt voltage sources in a prelayout netlist.

Syntax

```
.param HSIMZV = <0 | 1 | 2>
```

Argument	Description
0	Retains the zero-volt voltage source and the two terminal nodes. There is no merging of terminals.

Feedback

Chapter 4: HSIM® Commands in Alphabetical Order

HSIMZV

Argument	Description
1	Retains the zero-volt voltage source and terminal nodes if current probing is applied to the voltage device. If there is no probing, HSIM removes the voltage source and merges the terminal nodes. This is the default setting.
2	HSIM always merges the terminal nodes and removes the zero-volt voltage source.

.PROTECT or .PROT

Keeps models and cell libraries private as part of the encryption process in HSPICE.

Syntax

.PROTECT

Description

Use this command to designate the start of the file section to be encrypted when using Metaencrypt.

- Use .UNPROTECT to end the file section that will be encrypted.
- Any elements and models located between a .PROTECT and an .UNPROTECT command inhibit the element and model listing from the LIST option.
- The .OPTION NODE nodal cross-reference and the .OP operating point printout do not list any nodes that are contained between the .PROTECT and .UNPROTECT commands.

Note: If you use .prot/.unprot in a library or file that is not encrypted you will get warnings that the file is encrypted and the file or library is treated as a “black box.”

Note: To perform a complete bias check and print all results in the Outputs Biaschk Report, do not use .protect/.unprotect in the netlist for the part that is used in .biaschk. For example: If a model definition such as model nch is contained within .prot/.unprot commands, in the *.lis you'll see a warning message as follows: **warning** : model nch defined in .biaschk cannot be found in netlist--ignored

Usage Note: The .prot/.unprot feature is meant for the encryption process and *not* netlist echo suppression. Netlist and model echo suppression is on by default since HSPICE C-2009.03. For a compact and better formatted output (*.lis) file, use .OPTION LIS_NEW

See Also

[.UNPROTECT or .UNPROT](#)

Chapter 4: HSIM® Commands in Alphabetical Order

.UNPROTECT or .UNPROT

.UNPROTECT or .UNPROT

Restores normal output functions previously restricted by a .PROTECT command as part of the encryption process in HSPICE.

Syntax

.UNPROTECT

Description

Use this command to restore normal output functions previously restricted by a .PROTECT command.

- Any elements and models located between .PROTECT and .UNPROTECT commands, inhibit the element and model listing from the LIST option.
- Neither the .OPTION NODE cross-reference, nor the .OP operating point printout list any nodes within the .PROTECT and .UNPROTECT commands.
- The .UNPROTECT command is encrypted during the encryption process.

Note: The following are usage notes:

- If you use .prot/.unprot in a library or file that is not encrypted warnings are issued that the file is encrypted and the file or library is treated as a “black box.”
- To perform a complete bias check and print all results in the Outputs Biaschk Report, do not use .protect/.unprotect in the netlist for the part that is used in .biaschk. For example: If a model definition such as model nch is contained within .prot/.unprot commands, in the * .lis you'll see a warning message as follows: **warning** : model nch defined in .biaschk cannot be found in netlist-- ignored
- The .prot/.unprot feature is meant for the encryption process and *not* netlist echo suppression. Netlist and model echo suppression is on by default since HSPICE C-2009.03. For a compact and better formatted output (* .lis) file, use .OPTION LIS_NEW

See Also

[.PROTECT or .PROT](#)

Running HSIM

Provides instructions for setting up a netlist and running HSIM.

- Specifying the Input Netlist
- HSIM Initialization File
- Parse Error Limit Setup
- Invoking HSIM

Specifying the Input Netlist

HSIM accepts most SPICE-compatible netlists with following components.

- Circuit description
- Device models for the following:
 - Diode
 - BJT
 - JFET
 - MOSFET
- Input stimuli
- Output specification

Only one netlist file can be specified as the HSIM input file. If a netlist contains multiple files, the additional files can be included in the primary netlist file. To include additional netlist files, use one of the following:

- .include
- .lib: Used for device models

HSIM Initialization File

Before reading the input netlist file, HSIM searches for the hsim.ini initialization file in the following directories, in the order shown:

- local dir
- \$HSIM_PROJECT
- \$HOME
- \$HSIM_HOME/etc

Only the first initialization file that is found is read. An initialization file in the \$HSIM_HOME/etc directory provides the default settings for all users. Each user can have a default setup by creating an hsim.ini file in each user's home directory.

To keep the original netlist file intact, add HSIM commands to hsim.ini.

```
.param HSIMOUTPUT=wdf /* generates WaveView wdf file */  
.print v(*) level=1 /* prints all top level nodes voltage waveforms */
```

Parse Error Limit Setup

The environment variable HSIM_PARSE_ERROR_LIMIT controls the maximum number of parse errors before HSIM aborts netlist compilation. The default value is 20. The number of parse errors can be changed, as shown in the following example:

```
% setenv HSIM_PARSE_ERROR_LIMIT 30 (for csh/tcsh)  
$ export HSIM_PARSE_ERROR_LIMIT=30 (for bash/ksh)
```

Invoking HSIM

To invoke HSIM, use the following command syntax.

```
hsim <[-i] |netlist_format> netlist_name <[-o] out_file>  
<-fsdb out_file.fsdb> <-wdf out_file.wdf> <-time  
final_time> <-top subckt_name> <-critic arguments>  
<-memlimit memory> <-wait_lic 1> <-cktsizeonly> <-case
```

```
0|1> <-post_devv> <-dp> <-h> <-webhelp> <-x>
<-parseonly> <-alter value> <-sweep value> <-monte value>
<-exec_num value>
```

Table 8 HSIM Command Line Arguments

Argument	Description
<code>-i</code>	Specifies the input netlist and provides compatibility with SPICE.
<code>netlist_format</code>	Specifies one of the following input netlist formats: <ul style="list-style-type: none"> ▪ <code>-spectre</code> for a Spectre® input file ▪ <code>-eldo</code> for an Eldo input file ▪ <code>-mcspice</code> for an MCSPICE input file ▪ <code>-tispice</code> for a TISPICE input file ▪ <code>-stver</code> for an ST-Eldo input file
<code>netlist_name</code>	Specifies the input netlist file name.
<code>-o out_file</code>	Specifies the prefix for output and log files. The default prefix name is <code>hsim</code> . HSIM generates log files that contain simulation statistics and the output files that contain simulation results.
<code>-fsdb out_file.fsdb</code>	Specify <code>-fsdb</code> to perform <code>.fft</code> and <code>.measure</code> analyses in the post-processing step. After regular simulation is completed, the simulation output is stored in a <code>.fsdb</code> file, such as <code>out_file.fsdb</code> . You can add <code>.measure</code> and/or <code>.fft</code> statements to the netlist file and invoke HSIM to perform <code>.fft</code> and <code>measure</code> analyses without repeating the simulation.
<code>-wdf out_file.wdf</code>	Specify <code>-wdf</code> to perform <code>.fft</code> and <code>.measure</code> analyses in post-processing step. After regular simulation is completed, the simulation output is stored in a <code>.wdf</code> file, such as <code>out_file.wdf</code> . You can add <code>.measure</code> and/or <code>.fft</code> statements to the netlist file and invoke HSIM to perform <code>.fft</code> and <code>measure</code> analyses without repeating the simulation.
<code>-time final_time</code>	Specifies the stop time in transient analysis.
<code>-top subckt_name</code>	Specifies the top level subcircuit name.

Chapter 5: Running HSIM

Invoking HSIM

Table 8 HSIM Command Line Arguments (Continued)

Argument	Description
<code>-critic arguments</code>	Specifies the Critic command arguments that perform analysis on post-layout synthesis-driven designs. Find Critic documentation on SolvNet.
<code>-memlimit memory</code>	Checks the amount of memory available.
<code>-wait_lic</code>	<code>-wait_lic</code> queues the job if the requested license keys are not available, just as defined by setting the <code>HSIM_WAIT_LICENSE</code> environment variable to 1: <code>% setenv HSIM_WAIT_LICENSE 1 (for csh/tcsh)</code> <code>\$ export HSIM_WAIT_LICENSE=1 (for bash/ksh)</code> To invoke the HSIM 64 Bit executable, set the following environment variables: <code>% setenv HSIM_64 1 (for csh/tcsh)</code> <code>\$ export HSIM_64=1 (for bash/ksh)</code>
<code>-cktsizeonly</code>	Reports which size-based license is necessary.
<code>-case 0 1</code>	Enables case-sensitive simulation.
<code>-dp</code>	Set the CPU to double precision mode.
<code>-h</code>	Lists all of the command line options.
<code>-webhelp</code>	Opens an online help collection in a web browser. Click “How to Use Help” in the lower left corner of the Help window for details about how to use Help features such as the Help icons, Search, and Bookmarks.
<code><-x></code>	Enables a macro command that provides a simple way to turn on command settings appropriate for recent technologies, as well as recently implemented HSIM technologies. If you use <code>HSIMSPEED=3</code> to <code>HSIMSPEED=5</code> in existing designs, this command line option provides equal or better performance, with similar accuracy, in new designs.
<code>-sdb</code>	See the Full-Chip Probing section.
<code>-acheckdv</code>	See the Full-Chip Probing section.

Table 8 HSIM Command Line Arguments (Continued)

Argument	Description
-probelist	See the Full-Chip Probing section.

HSIM provides the following command line options that let you execute individual simulations when the input deck requires many parallel, independent runs such as `.alter` statements, data sweeps, and Monte Carlo runs.

Table 9

Argument	Description
-parseonly	<p>Instructs HSIM to only parse the input netlist and provide a summary of the type of parallel simulation, as well as the number of iterations written to the <code>*rpt</code> output file.</p> <p>For example, a simulation netlist containing an <code>.alter</code> statement, or two data sweeps, or two Monte Carlo iterations <code>-parseonly</code> reports the following information, respectively:</p> <pre> ... Analysis search results: ALTER: 2 iterations ... Analysis search results: TR SWEEP: 2 iterations ... Analysis search results: MC: 2 iterations ... </pre>
-alter value	<p>Instructs HSIM start to simulation beginning with the specified value for the <code>.alter</code> iteration. For example, if the netlist contains 9 <code>.alter</code> statements, the indeces are from 0-8. If you specify <code>-alter 4</code>, HSIM simulates the 4th index (or 5th <code>.alter</code> iteration) through the final <code>.alter</code> to completion.</p>
-sweep value	<p>Specifies the same rules as the <code>-alter</code> command line option, but corresponds to input netlists containing <code>.data</code> transient sweeps.</p>

Chapter 5: Running HSIM

Invoking HSIM

Table 9

Argument	Description
<code>-monte value</code>	Specifies the same rules as the <code>-alter</code> command line option, but corresponds to input netlists containing Monte Carlo analysis (<code>-monte</code> on a <code>.tran</code> line). Note: Monte Carlo indeces range from 1 --> N notation, as opposed to 0 --> N-1 notation for alter/sweep analyses.
<code>-exec_num value</code>	Instructs HSIM to only simulate the specified number of iterations. According to the <code>-alter</code> value, HSIM simulates all iterations from starting point (the user-specified index value) to the last iteration. If you want to simulate only a single iteration, for example, use <code>-exec_num 1</code> .

In addition, HSIM can use the following command line arguments.

Table 10 HSIM Command Line Arguments

Argument	Description
<code>-ra</code>	Invokes Phase II of Reliability Analysis.
<code>-ralayout</code>	Generates the violation map for reliability analysis.
<code>-sp2dspf</code>	Invokes the SP2DSPF utility.
<code>-r</code>	Invokes static power net resistance analysis.
<code>-rout</code>	Specifies report generation for static power net resistance analysis.
<code>-primerail</code>	Generates the IVEC file for PrimeRail from reliability analysis data.
<code>-post_devv</code>	Enables post-processing for the <code>hsimDeviceV</code> circuit check command.

HSIM Library and Data Encryption

HSIM® can use the HSPICE© method for encrypting files, metaencrypt.

Encrypting Files

The encryption feature is useful for IP providers who want to ship their designs in the encrypted form and also want to be able to simulate their designs. Previous versions of HSIM can read the encrypted files, but the encryption needs to be done by a binary that is only available under the HSPICE installation. The G-2012.06 version of HSIM lets you use the metaencrypt utility without an HSPICE installation.

The functionality and usage and functions of the metaencrypt utility is the same as described in the *HSPICE User Guide: Basic Simulation and Analysis, Library and Data Encryption* chapter.

Feedback

Chapter 6: HSIM Library and Data Encryption

Encrypting Files

HSIM Circuit Simulation Examples

Provides circuit simulation examples such as: an example of a circuit with a resistor ladder containing 2 billion resistors and an SRAM circuit containing up to 51 million transistors.

This chapter provides two examples of circuit simulation:

- Resistor ladder circuits containing 2 billion resistors
- SRAM circuits containing up to 51 million transistors

The example directory is located at \$HSIM_HOME/tutorial.

Resistor Ladder Test Case

[Figure 6 on page 214](#) illustrates a resistor ladder that is constructed hierarchically with the following features:

- The lowest level subcircuit is called r10 and contains 20 resistors.
- Each resistor has a resistance of $1.0e^{12}$ ohms.
- The equivalent resistance between subcircuit port nodes A and B is $2.0e^{11}$ ohm.
- The highest level subcircuit is called r1b. It contains 2 billion resistors with equivalent resistance of 2000 ohms.

Chapter 7: HSIM Circuit Simulation Examples

Resistor Ladder Test Case

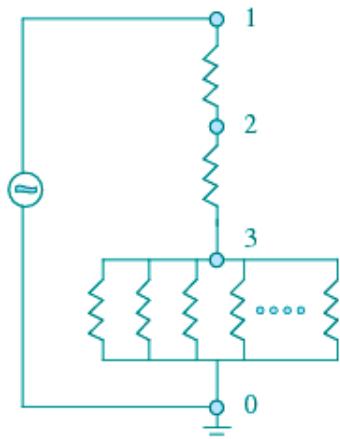


Figure 6 The Resistor Ladder Test Case Circuit

The circuit in [Figure 6 on page 214](#) shows the top-level containing:

- Two 1000 ohm resistors
- One instance of subcircuit r1b forming a resistor ladder
- 2,000,000,002 total resistors

The input is a PWL voltage source that ramps up then down from 0V to 3V to 0V as illustrated in [Figure 7 on page 215](#).

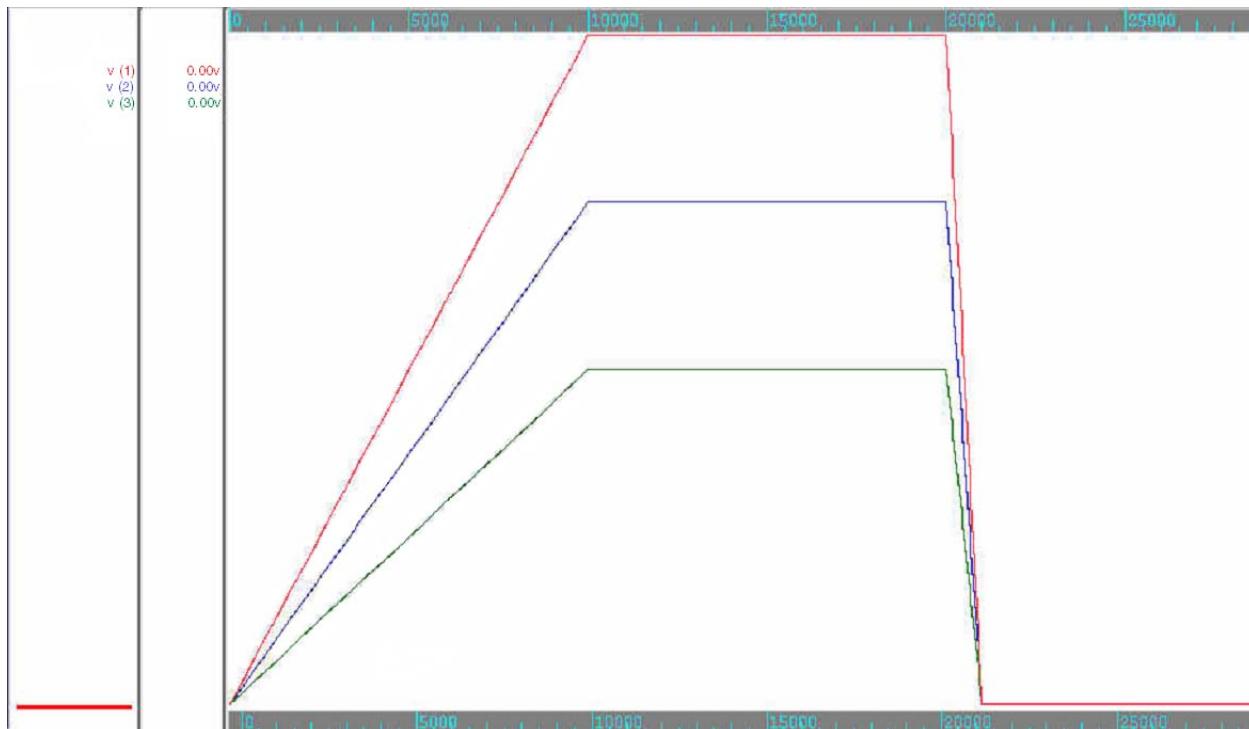


Figure 7 Piecewise Linear Driving Voltage and Simulation Results

Note: The example given above has an artificially high number of resistor branches in the voltage divider to demonstrate HSIM capacity and speed in handling a large hierarchical resistive ladder.

Resistor Ladder Simulation Example

The following instructions provide step-by-step guidance for the resistor ladder test case.

1. Run the Resistor Ladder Simulation by changing the directory to tutorial/2br. Use run script to run the tutorial.
2. After the resistor ladder simulation is complete, use a waveform display tool to inspect the simulation results.

The results are shown in [Figure 7 on page 215](#). This test case circuit shown [Figure 6 on page 214](#) in functions as a voltage divider where the voltages of node 2 and node 3 follow the voltage of input node 1.

Chapter 7: HSIM Circuit Simulation Examples

Resistor Ladder Test Case

3. Review the resistor ladder simulation statistics by inspecting the simulation statistics collected in the hsim.log file. The simulation takes only a few seconds. Most of the simulation time is used for netlist processing and hierarchical storage of circuit elements.

Transient simulation has the following requirements:

- Time < 0.1 second
- Peak memory usage is 29M
- Average memory usage is 0.0145 byte per element

[Figure 7 on page 215](#) shows the results of the Resistor Ladder test.

Here is the input netlist for the Resistor Ladder test case:

```
* 2 billion resistor circuits
.subckt r10 a b
ra1 a i1 1.0E12
rb1 i1 b 1.0E12
ra2 a i2 1.0E12
rb2 i2 b 1.0E12
ra3 a i3 1.0E12
rb3 i3 b 1.0E12
ra4 a i4 1.0E12
rb4 i4 b 1.0E12
ra5 a i5 1.0E12
rb5 i5 b 1.0E12
ra6 a i6 1.0E12
rb6 i6 b 1.0E12
ra7 a i7 1.0E12
rb7 i7 b 1.0E12
ra8 a i8 1.0E12
rb8 i8 b 1.0E12
ra9 a i9 1.0E12
rb9 i9 b 1.0E12
ra0 a i0 1.0E12
rb0 i0 b 1.0E12
.ends
.subckt r100 a b
xa1 a b r10
xa2 a b r10
xa3 a b r10
xa4 a b r10
xa5 a b r10
xa6 a b r10
xa7 a b r10
xa8 a b r10
xa9 a b r10
xa0 a b r10
.ends
.subckt r1000 a b
xa1 a b r100
xa2 a b r100
xa3 a b r100
xa4 a b r100
xa5 a b r100
xa6 a b r100
xa7 a b r100
xa8 a b r100
xa9 a b r100
xa0 a b r100
.ends
.subckt r10K a b
```

Chapter 7: HSIM Circuit Simulation Examples

Resistor Ladder Test Case

```
xa1 a b r1000
xa2 a b r1000
xa3 a b r1000
xa4 a b r1000
xa5 a b r1000
xa6 a b r1000
xa7 a b r1000
xa8 a b r1000
xa9 a b r1000
xa0 a b r1000
.ends
.subckt r100K a b
xa1 a b r10K
xa2 a b r10K
xa3 a b r10K
xa4 a b r10K
xa5 a b r10K
xa6 a b r10K
xa7 a b r10K
xa8 a b r10K
xa9 a b r10K
xa0 a b r10K
.ends
.subckt r1M a b
xa1 a b r100K
xa2 a b r100K
xa3 a b r100K
xa4 a b r100K
xa5 a b r100K
xa6 a b r100K
xa7 a b r100K
xa8 a b r100K
xa9 a b r100K
xa0 a b r100K
.ends
.subckt r10M a b
xa1 a b r1M
xa2 a b r1M
xa3 a b r1M
xa4 a b r1M
xa5 a b r1M
xa6 a b r1M
xa7 a b r1M
xa8 a b r1M
xa9 a b r1M
xa0 a b r1M
.ends
.subckt r100M a b
```

```
xa1 a b r10M
xa2 a b r10M
xa3 a b r10M
xa4 a b r10M
xa5 a b r10M
xa6 a b r10M
xa7 a b r10M
xa8 a b r10M
xa9 a b r10M
xa0 a b r10M
.ends
.subckt r1B a b
xa1 a b r100M
xa2 a b r100M
xa3 a b r100M
xa4 a b r100M
xa5 a b r100M
xa6 a b r100M
xa7 a b r100M
xa8 a b r100M
xa9 a b r100M
xa0 a b r100M
.ends
in 1 0 pwl 0 0 10n 3 20n 3 21n 0
r1 1 2 1000
r2 2 3 1000
x1 3 0 r1B

.tran 1n 30n
.print tran v(1) v(2) v(3)
.end
```

SRAM Test Case

The SRAM test case computation is focused on memory cell and sense amplifier operations.

The SRAM circuit is constructed hierarchically and contains the following elements:

- Memory cells
- Sense amplifiers
- Read/Write (R/W) control logic

Note: The address decoder is not included in the SRAM test case.

Chapter 7: HSIM Circuit Simulation Examples**SRAM Test Case**

This test case has the following characteristics:

- Memory cell: Typical 6-transistor SRAM cell.
- Sense amplifier: Differential amplifier commonly used in SRAM circuits.
- Memory bank: 1024 rows, 32 columns

The word lines used in this case have the following properties:

- Only four rows are used in this simulation: w10, w11, w12, and w13.
- Remaining word lines are connected to w14 and then to GND.

Each pair of bit-lines is connected to the sense amplifier. Subcircuit 32K contains 32K memory cells and 32 sense amplifiers. This memory bank is used to construct the following:

- 128K memory bank (subcircuit 128K).
- 512K memory bank (subcircuit 512K).

The circuits constructed have the following characteristics:

- Circuit 1 sram1.spi contains the following:
 - 1 active 512 K memory bank
 - 3.1 million transistors
- Circuit 2 sram2.spi contains the following:
 - 1 active 512 K memory bank
 - 3 dummy 512K memory banks
 - 12.6 million transistors
- Circuit 3 sram3.spi contains the following:
 - 1 active 512K memory bank
 - 15 dummy 512K memory banks
 - 51 million transistors

SRAM Example

The following instructions provide step-by-step guidance for the SRAM test case.

1. Run the SRAM Simulation by changing the directory to tutorial/sram and running the following test scripts, located in the tutorial directory:

- run1: Performs 512K memory circuit simulations.
 - run2: Performs 2Mg memory circuit simulations.
 - run3: Performs 8Mg memory circuit simulations.
2. After simulation is complete, inspect the log files and check for the following:
- CPU time
 - Memory usage

The three simulation results are shown in [Figure 8 on page 222](#).

- da0: A bidirectional input/output (I/O) port driven by a PWLZ voltage source. This is an HSIM circuit simulation-only feature that supports bidirectional operation.
- bl0: Column 0 bit line
- bl0n: Column 0 bit line bar

The CPU time to simulate the 51 million transistor SRAM is only 5 times longer than the time to simulate the 3.1 million transistor SRAM.

Note: The 5-times increase in CPU time and memory utilization is significantly less than the 16-times increase in circuit size.

During the write cycle, the input vector writes the following data into the rows and columns shown in [Table 11 on page 221](#).

Table 11 Input Vector Data Write Distribution

DATA	COLUMN	ROW
1 (one)	0	0
0 (zero)	0	1
1 (one)	0	2
0 (zero)	0	3

HSIM reads back the data bits from the core cells during the read cycle.

The results of the three simulations are very close. The waveform comparison for the three simulation results is shown in [Figure 8 on page 222](#). These results provide the following data:

Chapter 7: HSIM Circuit Simulation Examples

SRAM Test Case

- Confirm a correct data R/W operation
- Verifies there is no precision degradation in the HSIM simulation as the circuit size increases substantially.

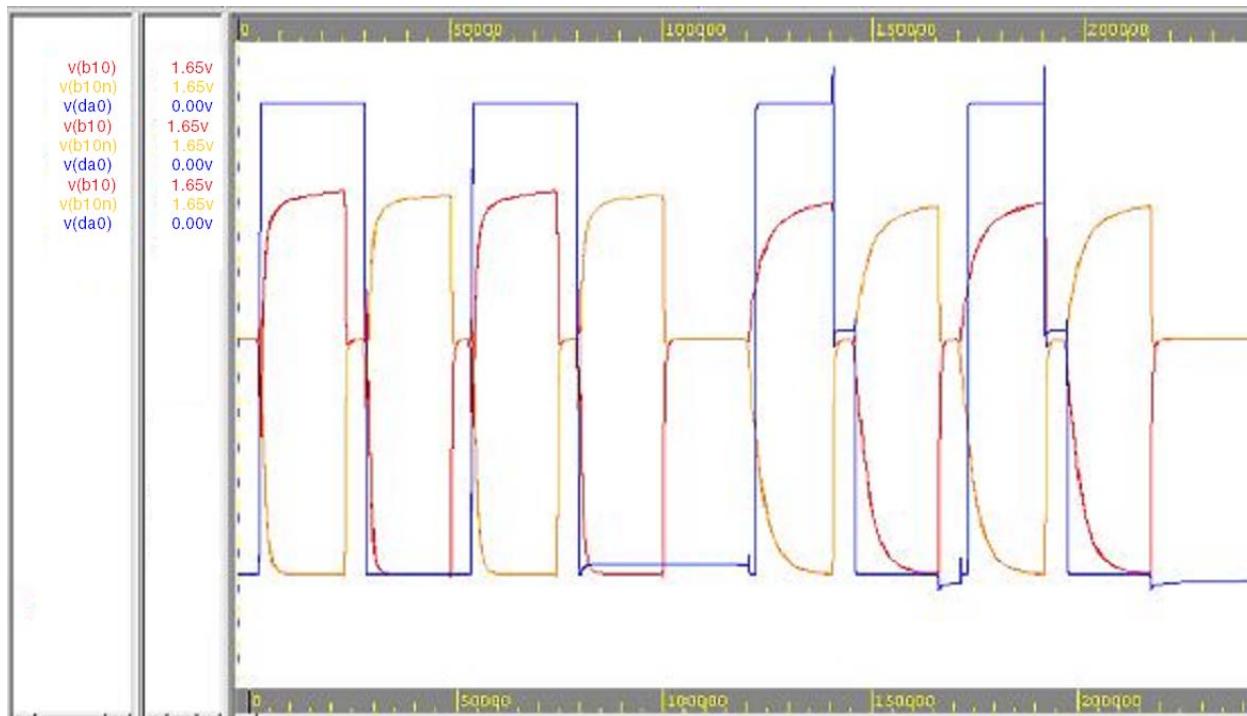


Figure 8 Three Overlapping SRAM Test Case Simulation Results

Reviewing the SRAM Simulation Statistics

This example demonstrates HSIM efficiency. In a desktop PC with Pentium II 550 MHz processor and Microsoft Windows 2000, the simulation provides the following results shown in [Table 12 on page 222](#).

Table 12 Desktop PC Run Results

RUN	RESULTS
run1	<ul style="list-style-type: none">■ 3.1 million SRAM transistors■ 8.81 seconds■ 11 Mbytes (of memory utilization)

Table 12 Desktop PC Run Results

RUN	RESULTS
run2	<ul style="list-style-type: none">▪ 12.6 million SRAM transistors▪ 16.6 seconds▪ 14.6 M bytes
run3	<ul style="list-style-type: none">▪ 51 million SRAM transistors▪ 42.5 seconds▪ 25.6 Mbytes

Note: Simulation results vary, depending on the hardware and software versions used.

Feedback

Chapter 7: HSIM Circuit Simulation Examples

SRAM Test Case

Input Netlist

Describes the procedure for developing an HSIM netlist using one or multiple files. The netlist file includes information such as: circuit topology description consisting of circuit elements and their connectivity, element models and their parameter values, simulation control parameters, stimulus input sources, and output specifications.

This chapter contains the following sections:

- [Netlist Syntax Summary](#)
- [Netlist Differences Between HSIM and SPICE](#)
- [Device Models](#)
- [IF-ELSE Syntax in Netlists](#)
- [Driving Sources and Input Stimuli](#)
- [Digital Vector File](#)
- [VCD Direct-Read Feature](#)
- [Vector Data](#)
- [Built-in Functions](#)
- [Simulation and Control Statements](#)

Netlist Syntax Summary

HSIM input netlists contain some or all of the following information:

- Circuit topology description consisting of circuit elements and their connectivity.
- Element models.
- Simulation control parameters.
- Stimulus input sources and output specifications.

These data can be described in either single file or multiple files. When multiple files are involved, an .include statement must be used to include main netlist files. Refer to [.include on page 331](#).

HSIM netlist syntax can be summarized in the table below.

Table 13 Netlist Syntax Summary

Syntax	Description
Title line	The first line in the main netlist file is treated only as the title line. It has no effect on the circuit description.
Characters	Except for filename and the directory pathname used in the .include and .lib statements, each character in the netlist is case-insensitive.
Buffer size	The buffer size to store the token or identifier can be set up to 1,024 characters.
Asterisk symbol	A line that starts with an asterisk (*) is treated as a comment line. It is ignored.
Dollar symbol	To add comments after the input text on the same line, precede that comment with the dollar (\$) character.
Line continuation symbols	A line may continue with a backslash \' or double back slash \\ character placed at the end of the line. A plus (+) sign can also be placed at the beginning of the next line.
Key identification character	Each element description starts with a specific key identification character, such as M for MOSFET and R for resistor.

Table 13 Netlist Syntax Summary (Continued)

Syntax	Description
Period symbol	Other than the element description, there are lines that start with the period character ‘.’. Examples include:.param - defines the parameter value.model - defines the device model
Element and control lines	The element and control lines placed inside the subcircuit scope are mostly effective within the subcircuit.
Line sequence	The line sequence has no effect on the circuit description.
Environment variable	HSIM accepts an environment variable in the pathname of a file that appears in a netlist. For example:
	.inc "\$sim_dir/testbench"
Line placement	Except for elements defined within the subcircuit definition which cannot be placed outside the scope between .subckt and .ends, any element or control option line can be placed anywhere between the first or title line and the last line. The last line is .end.
HSIMNOMULTIEND on page 116	HSIM accepts multiple .end statements. If you want HSIM to error out during parsing when multiple .end statements exist, issue the following command: .param HSIMNOMULTIEND=1

Netlist Differences Between HSIM and SPICE

The input netlist format for HSIM is almost completely compatible with that of SPICE or HSPICE® simulators. There are some differences which are described as follows:

- HSIM reads input netlist and performs the circuit simulation, even when the netlist contains elements that are supported by SPICE but are not supported by HSIM. Elements that are not supported are ignored and warning messages are displayed. To view which elements HSIM does not support,

Chapter 8: Input Netlist

Netlist Differences Between HSIM and SPICE

refer to [Table 14 on page 228](#).

- HSIM also supports elements that are not supported in SPICE. For detailed information refer to [Table 14 on page 228](#).
- HSIM also supports the string parameters used in HSPICE. However, there are some limitations. See the [HSPICE String Parameter Support](#) section.

SPICE and HSIM Element and Capability Support

The table below shows the relationship between SPICE and HSIM elements that are supported or not supported by each other.

Table 14 Element Support Matrix

Element List	SPICE Elements/ HSIM Supported	HSIM Element/ Not SPICE Supported
PASSIVE		
Resistor	Yes	
Capacitor	Yes	
Self and Mutual Inductors	Yes	
Lossless Transmission Line (LLTL)	Yes	
Lossy Transmission Line (LYTL)	Yes	
ACTIVE		
Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET)	Yes	
Diode	Yes	
Bipolar Junction Transistor (BJT)	Yes	
Junction Field-Effect Transistor (JFET)	Yes	
Metal-Semiconductor Field-Effect Transistor (MESFET)	Yes	

Table 14 Element Support Matrix (Continued)

Element List	SPICE Elements/ HSIM Supported	HSIM Element/ Not SPICE Supported
SOURCE		
Current-Controlled Current Source (CCCS)	Yes	
Current-Controlled Voltage Source (CCVS)	Yes	
Voltage-Controlled Current Source (VCCS)	Yes	
Voltage-Controlled Voltage Source (VCVS)	Yes	
Independent Voltage Sources:	Yes	
■ AM		
■ DC		
■ EXP		
■ PULSE		
■ PWL		
■ SFFM		
■ SIN		
OTHER		
Voltage-Controlled Resistor (VCR)	Yes	
Voltage-Controlled Capacitor (VCC)	Yes	
PWLZ Element		Yes
VECTOR Input Element		Yes
VECTOR Output Automatic Comparison		Yes
Functional element modeled by C behavioral description. Refer to Chapter 23, C Language Functional Model .		Yes

Chapter 8: Input Netlist

Netlist Differences Between HSIM and SPICE

The table below shows the relationship between SPICE and HSIM capabilities that are supported or not supported by each other.

Table 15 Capabilities Support Matrix

Capabilities List	SPICE Capabilities/ HSIM Supported	SPICE Capabilities/ Not HSIM Supported	HSIM Capabilities/ Not SPICE Supported
Transient Analysis	Yes		
DC Operating Point	Yes		
Measure	Yes		
Fourier Transform, and Fast Fourier Transform	Yes		
Bisection Optimization	Yes		
AC Analysis	Yes		
DC Analysis	Yes		
Sensitivity Analysis		Yes	
Noise Analysis		Yes	
Pole-Zero Analysis		Yes	
Timing Analysis			Yes
Power Analysis			Yes
Interactive Mode			Yes

HSPICE String Parameter Support

HSIM supports the string parameters used in HSPICE. In HSPICE, the convention is to pass the string using the function "str()". To use a string parameter, you can specify:

```
.param myVar = str('myString')
```

Then you can refer back to this string by using another expression as shown below :

```
.param abc= str (myVar)
```

String parameters have the following limitations:

- They do not apply to the S-parameter, B-element, or W-element.
- They do not apply to file name.
- Because HSIM does not support the L model in SPICE format, string parameter does not support a model name of L.
- They do not apply to those resistors whose models include either the “vc1r” or “vc2r” parameters.
- They do not apply to those capacitors whose model levels are 6 (fero-cap), 7 (mosvar) and 22 (mos cap).
- They do not apply to a MOS whose model level is 69 (psp model).
- They do not apply to SOI MOS.

Encrypted HSPICE® Netlists

HSIM is able to parse encrypted HSPICE transistor level netlists, including Triple DES (3DES) and Verilog-A encryption, as long as the netlist was encrypted using the `metaencrypt` encryption utility.

For details on `metaencrypt`, refer to the “Library and Data Encryption” chapter in the *HSPICE® User Guide: Basic Simulation and Analysis*.

Device Models

HSIM supported device models include the following:

- BSIM1 MOSFET Model
- BSIM3 MOSFET Model v3.0, v3.1, v3.2, v3.2.3, v3.2.4, and v3.3.0
 - Note:** The stress model formula originally included in BSIM4.3 is also available in the BSIM3 Model by setting `stimod=1`.
- BSIM3SOI Model v2.0, v2.2.1, v2.2.2, v2.2.3, v3.0, v3.1, and v3.2

Chapter 8: Input Netlist

Device Models

- BSIM4SOI Model v4.0. HSIM supports the following instance parameters supported in HSPICE: MULID0, MULVSAT, DELK1, DELNFCT and DTEMP for BSIM SOI4.
- BSIM4 MOSFET Model, v4.1, v4.2, v4.2.1, v4.3.0, v4.4.0, v4.5.0, v4.6.0, and v4.6.3
- HiSIM MOSFET Model v2.3.0
- Philips MOSFET Models: MOS9, MOS11, MOS30, MOS31, and MOS40
- EKV MOSFET Model
- SPICE Level-1/Level-2/Level-3 MOSFET Models
- BJT Models: G.-P. Model, VBIC Model, Mextram Model, HICUM Model
- Diode Model: JUNCAP2, v200.1
- PSP Model v102.0, v103, V103.1
- MOS varactor model in Spectre® format (model mname mosvar)
- JFET Model
- Ferroelectric Capacitor (FeCAP) Model. Refer to [Chapter 20, Ferroelectric Capacitor \(FeCap\) Model.](#) for detailed information.
- MOS Varactor, version 1.0

Gate Capacitance Model

HSIM uses its own gate capacitance model to closely match corresponding SPICE results. The gate capacitance model automatically adjusts its parameter values according to the given SPICE model. The optional charge conservation feature can be enabled to rigorously conserve charge at a slight expense of the simulation speed.

Stress Model

The stress model formula originally included in BSIM4.3 Model is also available in BSIM3 Model by setting stimod=1.

Element Statements

Netlist elements are described as follows:

```
NAME node1 node2 <... nodeN> <model_name> evaluate <optional
parameters>
```

Element statements are described in the table below.

Table 16 Element Syntax Conventions

Parameter	Description
NAME	Specifies the type and the name of an element. The first letter in the name field identifies the element type. For example: <ul style="list-style-type: none"> ▪ M represents a MOSFET ▪ C represents a capacitor ▪ R represents a resistor The remaining alphanumeric characters in the name field define the unique element name.
node1 node2 <... nodeN>	Specifies the names of the circuit nodes to which the element is connected.
model_name	Refers to a model. Detailed information of that model is provided in a separate model definition.
evaluate	Specifies the value of the element. For example, the statement R100 a12 b55 1000 indicates that a resistor named R100 is connected to nodes a12 and b55, and the resistance value is 1000 ohms.
Line continuation	Statement continues to the next line. A plus sign (+) sign must be used at the beginning of the continuation line.
Commas	Provided only for ease of reading. Values are recognized as separate with blank spaces. For example, in the following line of code, x1, y1, x2, and y2, with or without commas, are recognized as four separate variables: Gaa pos_n neg_n <vccs> pwl(1) nc1+ nc1- <delta=val2> x1, y1, x2, y2

Chapter 8: Input Netlist

Device Models

Note: In this document, the syntax for some elements and models can appear to be multiple lines. However, the plus sign (+) is not included as those lines are intended to be used as single lines.

Resistor

Syntax

```
Raa n1 n2 <model_name> <r=>rval    <tc1=val2 <tc2=val3>
    <scale=val4> <m=val5> <dtemp=val6> <l=val7> <w=val8>
    <c=val9>
Raa n1 n2 <r=>"expr"
```

Parameters

Resistor parameters are described in the table below.

Table 17 Resistor Parameters

Parameter	Default	Description
Raa		Resistor element name which must begin with the character R.
n1		First node name.
n2		Second node name.
model_name		Resistor model name. It is used to refer to a resistor model.
r		Keyword to identify resistance value, rval, in ohms at room temperature.
tc1		First-order coefficient for temperature effect.
tc2		Second-order coefficient for temperature effect.
scale	1	Element scale parameter which scales the resistance by the specified value.
expr		A mathematical expression defining the resistance. The expression may contain controlling variables of node voltages and/or branch currents in the circuit.
m	1	Multiplier for parallel instances of a resistor.

Table 17 Resistor Parameters (Continued)

Parameter	Default	Description
dtemp	0	Difference between the resistor temperature and the global circuit temperature in degrees Celsius.
l	0	Length of resistor. When unspecified, l is assigned the default value.
w	0	Width of resistor. When unspecified, w is assigned the default value.
c	0	Capacitance connected from n2 to ground. When unspecified, c is assigned the default value.

Examples

```
RK15    node1 node5    100
R1990  X12      X90     1.5K
rbx     n1       n2      "200+0.02*(v(n3) - v(n4))"
```

Capacitor

Syntax

Caa n1 n2 <model_name> <c=>cval <tc1=val2 <tc2=val3>> <scale=val4>
<m=val5> <dtemp=val6> <ic=val7> <l=val7> <w=val8>

Caa n1 n2 <c=>"expr"

Caa n1 n2 Q="expr"

Caa n1 n2 POLY C0 C1 ...

Parameters

Capacitor parameters are described in the table below.

Table 18 Capacitor Parameters

Parameter	Default	Description
Caa		Capacitor element name, which must begin with the character C.

Table 18 Capacitor Parameters (Continued)

Parameter	Default	Description
n1		Positive node name.
n2		Negative node name.
model_name		Capacitor model name. It is used to refer to a capacitor model.
c		Keyword to identify capacitance value, cval, in farads at room temperature.
Q		Q identifies the capacitor charge in Coulombs given by expression expr, which can depend on node voltages. In this case value of the capacitance is determined by differentiation of expression over node voltages. This element can be charge conserving if parameter hsimcapcc=1 is set (see section Simulation parameters, subsection Device model parameters).
tc1		First-order coefficient for temperature effect.
tc2		Second-order coefficient for temperature effect.
scale	1	Element scale parameter.
expr		A mathematical expression defining the capacitance. The expression may contain controlling variables of node voltages and/or branch currents in the circuit.
m	1	Multiplier for parallel instances of a capacitor.
dtemp	0	Difference between the capacitor temperature and the global circuit temperature in degrees Celsius.
ic		Initial voltage across the capacitor. The .ic on page 330 overrides this value.
l	0	Length of capacitor. When unspecified l is assigned the default value.
w	0	Width of capacitor. When unspecified, w is assigned the default value.

Table 18 Capacitor Parameters (Continued)

Parameter	Default	Description
POLY		Specifies capacitance using a polynomial form.
C0 C1 ...		Coefficient of a polynomial in voltage describing the capacitor value. An example is: $C=C0 + C1 * V + C2 * V * V + \dots$

Note: The Ferroelectric Capacitor Model from Ramtron is supported as a Level-6 Model. The MOSCAP Model from Motorola is supported.

Examples

```
C12A5    HB12    0          1.2pf
cpump  aref  ngnd  "1e-13 * v(bias1) * v(state3)"
```

Self-Inductor

Syntax

```
Laa n1 n2 <l=>lval <tc1=val2 <tc2=val3>> <scale=val4>
<m=val5> <dtemp=val6> <ic=val7>
```

Parameters

Self-inductor parameters are described in the table below.

Table 19 Self-Inductor Parameters

Parameter	Default	Description
Laa		Self-inductor element name which begin with the character L.
n1		Positive node name.
n2		Negative node name.
l		Key word to identify self inductance value, lval.
tc1		First-order coefficient for temperature effect.

Feedback

Chapter 8: Input Netlist

Device Models

Table 19 Self-Inductor Parameters (Continued)

Parameter	Default	Description
tc2		Second-order coefficient for temperature effect.
scale	1	Element scale parameter.
m	1	Multiplier for parallel instances of self-inductor.
dtemp	0	Difference between the self-inductor temperature and the global circuit temperature in degrees Celsius.
ic		Initial current flowing through the inductor.

Example

LT170 42 69 1uh

Mutual Inductor

Syntax

K11 L22 L33 <k=>kvalue

Parameters

Mutual inductor parameters are described in the table below.

Table 20 Mutual Inductor Parameters

Parameter	Description
K11	Mutual inductor element name, which must begin with the character K.
L22	First coupled inductor name.
L33	Second coupled inductor name.
k	Keyword to identify mutual coupling coefficient value, kvalue. The value is greater than 0, and less or equal to 1.

Example

K200 L100 L32 0.6

Lossless Transmission Line

Syntax

```
Taa in_n  refin_n  out_n  refout_n z0=val2 td=val3 <l=val4>
Taa in_n  refin_n  out_n  refout_n z0=val2 f=val3 <nl=val4>
```

Parameters

Lossless transmission line parameters are described in the table below.

Table 21 Lossless Transmission Line Parameters

Parameter	Default	Description
Taa		Lossless transmission line element name, which must begin with the character T.
in_n		Input node name.
out_n		Output node name.
refin_n		Reference node name for input signal.
out_n		Reference node name for output signal.
z0		Characteristic impedance.
td		Transmission delay time in the unit of second per meter.
l	1	Transmission line length in the unit of meter.
f		Characteristic frequency to determine the normalized electrical length which can be specified by the nl parameter.
nl	0.25	Normalized electrical length of the transmission line with respect to the wavelength in the line at the frequency specified by f parameter.

Example

```
T100 in1  gnd out1 gnd  z0=50  td=1n l=2
```

In this example, a 2-meter transmission line T100 is connected from node `in1` to node `out1` with the reference nodes for input signal and output signal

grounded. The characteristic impedance is 50 ohms and the transmission delay is 1 ns per meter.

Lossy Transmission Line

HSIM supports lossy multiconductor transmission lines. The N-Wire line ($N=1,2,3,..$) has $(2*n+2)$ external terminals which must be specified in the element description. The line is assumed to be uniform in the direction of its length.

Syntax

```
Waa n1 <n2 ... nN > nR m1 <m2 ... mN > mR n=N l=val3  
rlgmodel=model_name|rlgcfile=file_name|  
fsmodel=fs_model_name
```

This syntax indicates the value of N is at least 1 (one), and that at least one terminal must be specified.

Parameters

Lossy transmission line parameters are described in the table below.

Table 22 Lossy Transmission Line Parameters

Parameter	Description
Waa	Lossy transmission line element name, which must begin with the character W.
nk	Input node name for the kth wire of the transmission line.
nR	Reference node name for input signal.
mk	Output node name for the kth wire of the transmission line.
mR	Reference node name for output signal.
n	Number of wires (signal conductors).
l	Transmission line length in the unit of meter.
rlgmodel	Model name for the transmission line.
rlgcfile	Specifies a file name that contains RLGC information.

Table 22 Lossy Transmission Line Parameters

Parameter	Description
fsmodel	Specifies a model name for the field solver which calculates RLGC information.

Example

```
W100    n1 n2 n3 0 m1 m2 m3 0 n=3 l=0.3 rlgcmodel=tline1
```

Lossy Transmission Line Model

The n-wire transmission line behavior can be described by the following equations:

$$\begin{aligned} -\frac{\partial V}{\partial x} &= R I + L \frac{\partial I}{\partial t} \\ -\frac{\partial I}{\partial x} &= G V + C \frac{\partial V}{\partial t} \end{aligned}$$

$$V=V(x,t) \text{ and } I=I(x,t)$$

are n-dimensional voltage and current vectors. L, C, R, and g are inductance, capacitance, resistance, and conductance matrices per unit length. In the case of temporal dispersion, these matrices can be frequency dependent.

The current implementation of the multiwire lossy transmission line model assumes the following functional form of the matrices in the frequency domain: $L=L_0$ (frequency independent), $C=C_0$ (frequency independent), $R=R_0 + (1+j)f\frac{1}{2}R_s$, $G=G_0 + fG_d$.

Here f is the frequency, j is imaginary unit, matrices R_0 , G_0 , R_s , and G_d are frequency independent. Matrices R_s and G_d are known as the skin effect matrix and the dielectric loss matrix.

Matrices L_0 , C_0 , R_0 , G_0 , R_s , and G_d are symmetrical and positive definite. All entries of the matrices L_0 , R_0 , R_s should be nonnegative. All diagonal terms of the matrices C_0 , G_0 , and G_d should be non-negative, while all off-diagonal entries should be non-positive. Diagonal terms of matrices L_0 and C_0 should be positive. If some of the matrices are omitted, their default values are assumed to be zeros. Matrices L_0 and C_0 should always be defined. If any of the other four matrices is defined, R_0 should also be defined.

Chapter 8: Input Netlist

Device Models

Syntax

```
.model model_name w modeltype=rlgc n=val_n lo=L_matrix  
co=C_matrix <ro=Ro_matrix <go=Go_matrix> < rs=Rs_matrix>  
<gd=Gd_matrix>>
```

Parameters

Lossy transmission line model parameters are described in the table below.

Table 23 Lossy Transmission Line Model

Parameter	Description
model_name	The model name for the transmission line.
w	Keyword to identify the lossy transmission line model.
modeltype	Model format selector. At present, only rlgc format is available. This format is used to indicate that L, C, R, and G matrices are used. This is the only format that is supported by the HSIM at present.
n	Number of signal conductors. This number should be the same as that used on the element line.
lo	Inductance matrix in henries per meter .
co	Capacitance matrix in farads per meter.
ro	Resistance matrix in ohms per meter.
go	Shunt conductance matrix in ohms per meter.
rs	Skin-effect resistance matrix in ohms per meter.
gd	Dielectric-loss conductance matrix in ohms per meter.

Because of the matrix symmetry, only part of the matrices must be specified. HSIM requires specification of the lower diagonal part of matrices in the row major order. Each matrix keyword is followed by $N^*(N+1)/2$ matrix coefficients. For instance, in the case of $n=3$, the matrix lo should be specified in the following format: lo=Lo11 Lo21 Lo22 Lo31 Lo32 Lo33.

Example

```
.model tline1 w modeltype=rlgc n=3
+ lo=
+ 1.9e-07
+ 2.0e-08 2.0e-07
+ 1.0e-08 2.0e-08 2.1e-07
+ co=
+ 2.4e-10
+ 2.0e-11 2.3e-10
+ 1.0e-11 2.0e-11 2.2e-10
+ ro=
+ 3.0
+ 0.0 3.0
+ 0.0 0.0 3.0
+ rs=
+ 4.2e-4
+ 0 4.2e-4
+ 0 0 4.2e-4
+ gd=
+ 3.1e-11
+ 3.0e-12 3.1e-11
+ 3.0e-12 3.0e-12 3.1e-11
```

Lossy Transmission Line File

Lossy transmission line data can be specified in a separate RLGC file. This method does not offer any advantage over RLGC model which can also be placed in a separate file and included through .include statement, however it is supported for compatibility.

To use RLGC file, the element statement should use keyword **RLGCFILE=<file.name>**. The file <file.name> specifies transmission line data in the following format:

```
<N>
<L-matrix>
<C-matrix>
<R-matrix>
<G-matrix>
<Rs-matrix>
<Gd-matrix>
```

where <N> is the number of lines. The following the L, C, R, G, Rs, and Gd matrices should be given in lower triangular format similar to how these matrices are specified in RLGC model. This is a positional format and hence has no keywords. Star in the first position can be used to include comments.

Chapter 8: Input Netlist

Device Models

Example

```
*  
* N= 3  
*  
* LO= 2.2e-6 3.0e-6 2.8e-6  
* 4.0e-7 5.0e-7  
* 8.0e-8  
* CO= 2.4e-11 2.1e-11 2.5e-11  
* -5.0e-12 -6.0e-12  
* -1.0e-12  
* RO= 20.0 25.0 30.0  
* 0 0  
* 0  
* GO= 6.0e-4 5.0e-4 5.5e-4  
* -1.0e-4 -9.e-5  
* -2.0e-5  
* RS= 1.0e-3 1.2e-3 1.4e-3  
* 0 0  
* 0  
* GD= 5.0e-13 5.2e-13 4.8e-13  
* -1.0e-13 -8.0e-14  
* -1.5e-14
```

Starting from the 2007.03 HSIM release, HSIM also supports the HSPICE-compatible W-element of the RLGC format in model or file formats with the same syntax. To invoke this option, use .param HSIMGMIWELM=1. The default value (0) uses the HSIM implementation.

Field Solver Model

HSIM supports field solver syntax for finding the L, C, Ro, Go, Rs, and Gd matrices required for simulation of a lossy transmission line. Field solver syntax

provides the specification for a two-dimensional geometry consisting of conductors, ground planes, and material parameters such as dielectric permittivity, conductivity, loss tangent, etc.

The syntax assumes that a transmission line flows in z-direction and is situated in layered medium referred to as stack. The y-axis is assumed to be orthogonal to layers. The stack has at least one ground plane with xz-coordinates at the bottom with an optional top ground plane in some cases. The medium above the bottom plane consists of one or more layers of dielectric containing one or more conductors. Refer to [Figure 9 on page 245](#).

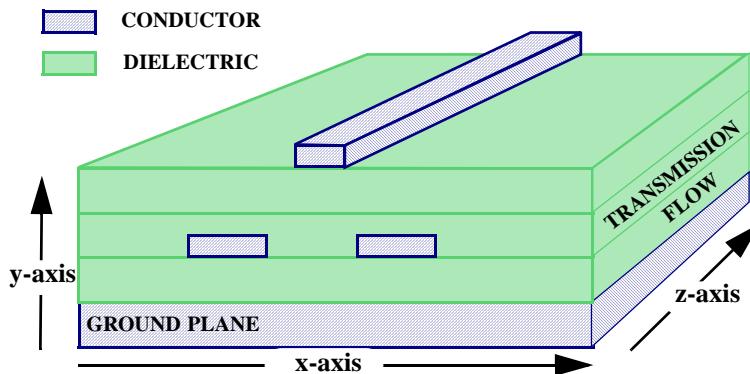


Figure 9 Typical Field Solver IC Model

The element referring to the field-solver model must use the following parameter to indicate that field-solver will be used:

```
fsmodel=fs_model_name
```

The field solver model card uses the following syntax:

```
.model fs_model_name W ModelType=FieldSolver +
    layerstack=stack_name <FSOptions=FS_Option> +
    conductor=(material=material1, shape=shape1,
    origin=(x1,y1)) /
...
+ conductor=(material=materialN, shape=shapeN,
    origin=(xN,yN)) /
```

The `layerstack` keyword gives a name of the stack which should be specified somewhere else. Keywords `conductor` in number N (which must be the same as N in the transmission line element specification) assigns material,

Chapter 8: Input Netlist

Device Models

geometrical shape, and reference origin to each conductor. Details of material and shape are given with separate cards.

Material specification uses the following syntax:

```
.material material_name DIELECTRIC|METAL <er=val1 mu=val2  
losstangent=val3 conductivity=val4 >
```

There are two predefined material names: PEC for perfect conductor and AIR for free space. Required parameters have self-explanatory names, relative permittivity, er, and relative permeability, mu, must be positive and not less than one.

There are two shapes supported, rectangular box and a strip. The following syntax is used:

```
.shape name RECTANGLE width=val1 height=val2  
.shape name STRIP width=val1
```

Width referred to x-direction, height referred to y-direction. The stack has the following syntax:

```
.layerstack stack_name  
+ layer=(material1, thickness1)  
...  
+ layer=(materialK, thickness)  
< + background=materialB>
```

The first layer must be a conductor and it is bottom ground plane. The following layers 2, 3, etc. are listed in the order of increasing coordinate y. Thickness of each layer should also be given. Each conductor except ground plane(s) must be completely situated in one dielectric layer. Touching interface of layers is allowed. Optional background is a layer of infinite thickness above layer number K. It is used only if there is no top ground plane. default background is AIR.

Additional options to field solver can be passed through the following command:

```
.fsoptions FS_Option < printdata=YES|NO computero=YES|NO  
computego=YES|NO computers=YES|NO computegd=YES|NO  
accuracy=HIGH|MEDIUM|LOW >
```

Matrices L and C are always computed. Computing matrices Ro, Go, Rs, and Gd should be requested with the above commands. The computed matrices can be written down to external file fs_model_name.rlgc if printdata=YES. Command accuracy affects the two-dimensional meshes which are built to solve Laplace equation in 2-dimensional domain. Higher accuracy results in

finer mesh and larger matrices for electromagnetic fields. Refer to the following example.

Example

```

* Half Space, er=1
* XXXXXX XXXXXX
* ----- Z=0.000302
* er=3.2 thickness=0.0001
*
* XXXXX
* ----- Z=0.000202
* er=4.3 thickness=0.0002
* ----- Z=2e-06
* //// Bottom Ground Plane ///////////
* ----- Z=0
* Materials
.material copper METAL conductivity=5e+07
.material diel_1 DIELECTRIC er=4.3 losstangent=1e-3
conductivity=8e-4
.material diel_2 DIELECTRIC er=3.2 losstangent=1e-3
conductivity=8e-4
* conductor crosssection shapes
.shape rect RECTANGLE width=350e-6 height=70e-6
* Dielectric stack
.layerstack Stack
+ layer=(copper, 1u)
+ layer=(diel_1, 200u)
+ layer=(diel_2, 100u)
+ background=air
* Field-solver options
.fsoptions myOption accuracy=HIGH
+ ComputeRo=yes ComputeRs=yes ComputeGo=Yes ComputeGd=yes
+ printdata=YES
.model demo W ModelType=FieldSolver
+ layerstack=Stack FSOPTIONS=myOption
+ conductor=(material=copper, shape=rect, origin=(0u, 202u))
+ conductor=(material=copper, shape=rect, origin=(500u, 302u))
+ conductor=(material=copper, shape=rect, origin=(1000u, 302u))

```

N-Port

N-port is an element with 2^*N terminals (N ports) for which the behavior is specified in the table form in the frequency domain. For HSIM, these tables should contain S-parameters of n-port. The following syntax is used for an N-port element:

Chapter 8: Input Netlist

Device Models

Syntax

```
Sname port1 ref1 port2 ref2 ... portN refN file=filename  
<format=touchstone> <scale=scalevalue>
```

Parameters

In this syntax example, the following rules apply:

Sname

The element name which should start with S.

port1 ref1 ... portN refN

The 2*N terminals that define N ports.

port1, ref1

The first port; followed by additional ports as required.

file

Contains a table of S-parameters in one of acceptable formats.

format

Specifies the format of the table. The allowed format is touchstone.

scale

A multiplier for frequency in the table. For example: if scale=1e9, then the frequency is given in Gigahertz.

Alternate Syntax

The following alternate syntax can be used:

```
Sname port1 port2 ... portN ref mname=<s_model_name>
```

Where mname=<s_model_name> is reflected in:

```
.MODEL <s_model_name> s tstonefile=filename
```

Alternate Parameters

The Sname, port1... portN ref parameters are the same as the ones defined above. The additional parameters are:

mname

The name of the S model.

tstonefile

The name of a Touchstone file. The data in this file contains frequency-dependent array of matrixes. Touchstone files must follow the .s#p file extension rule, where # represents the dimension of the network.

For details, see *Touchstone® File Format Specification* by the EIA/IBIS Open Forum (<http://www.eda.org>).

Refer to the *HSPICE Simulation and Analysis User Guide* (Y-2006.03 or later) for more information on S model syntax.

MOSFET**Syntax**

```
maa ndr nga nso <nbu> model_name <l=val2> <w=val3> <ad=val4>
    <as=val5> <pd=val6> <ps=val7> <nrd=val8> <nrs=val9>
    <m=val10> <geo=val11> <rdc=val12> <rsc=val13>
    <delvto=val14> <off> <dtemp=val15>
```

Parameters

MOSFET parameters are described in the table below.

Table 24 MOSFET Parameters

Parameter	Default	Description
maa		MOSFET element name which must begin with the character M.
ndr		Drain node name.
nga		Gate node name.
nso		Source node name.
nbu		Bulk node name.
model_name		MOSFET model name. It is used to refer to a MOSFET model.
		Gate length of MOSFET.
w		Gate width of MOSFET.

Chapter 8: Input Netlist

Device Models

Table 24 MOSFET Parameters (Continued)

Parameter	Default	Description
ad		Drain diffusion area.
as		Source diffusion area.
pd		Perimeter of drain-bulk junction.
ps		Perimeter of source-bulk junction.
nrd		Number of squares of drain diffusion in drain resistance calculation.
nrs		Number of squares of source diffusion in drain resistance calculation.
m	1	Multiplier for parallel instances of MOSFET.
geo	0	Drain/source sharing selector for MOSFET with model parameter ACM=3.
rdc	0	Additional drain resistance.
rsc	0	Additional source resistance.
delvto	0	Zero-bias threshold voltage shift.
off	ON	Set initial condition to OFF for this element in DC operating point calculation.
dtemp	0	Difference between the MOSFET temperature and the global circuit temperature in degrees Celsius.

Example

MP1X 29 27 0 20 NMOS1 L=0.13U W=0.2U

M355 20 17 76 10 PMOS2 L=0.18U W=0.5U

MOSFET Model

Syntax

```
.model model_name nmos|pmos <encmode=0|1> <level=val2>
    <parameter1=val3> <parameter2=val4 ...>
```

Parameters

MOSFET model parameters are described in the table below:

Table 25 MOSFET Model Parameters

Parameter	Description
model_name	MOSFET model name. MOSFET element refers to the model through this name.
nmos pmos	Keyword to identify n-MOSFET or p-MOSFET model: select one name. (See the examples below)
encmode	Applicable to BSIM4 model libraries. Use 1 (on) to suppress the printing of warning/error messages originating in BSIM4.* model cards. Default is 0 (off).
level	Selector for different levels of MOSFET model.
parameter	Model parameter.
ako	pSPICE AKO model support. (See the example below)

Examples

```
.model p18_L18_W120_ aka: p18 PMOS
```

This means that a new model with the name "p18_L18_W120_" will be created based on the reference model "p18".

```
.model mos2 nmos    level=49 tox=8e-09 tnom=25
```

```
.model mos1 pmos    level=54 vto=-0.55
```

Chapter 8: Input Netlist

Device Models

MOSFET models are described in the table below.

Table 26 MOSFET Models

Model Level	Version	MOSFET Model
49/53	3.0	BSIM3v3.0
49/53	3.1	BSIM3v3.1
49/53	3.24 [default]	BSIM3v3.24
49/53	3.3	BSIM3v3.3
47	N/A	BSIM3v2
54	4.0/4.1	BSIM4.0.0/4.1.0
54	4.21	BSIM4.2.0 and BSIM4.2.1
54	4.3	BSIM4.3
54	4.4	BSIM4.4
54	4.5	BSIM4.5, TMI2.01
54	4.6.0 [default]	BSIM 4.6
54	4.6.1	BSIM4.6.1
54	4.7	BSIM4.7
54	4.8	BSIM4.8
57	2.0	B3SOIPD2.0
57	2.1	B3SOIPD2.2.1
57	2.2	B3SOIPD2.2.2
57	3.0	B3SOIPD3.0
57	3.1	B3SOIPD3.1
57	3.2 [default]	B3SOIPD3.2

Table 26 MOSFET Models (Continued)

Model Level	Version	MOSFET Model
64/111	1.1.2 [default]	HiSIM1.1.1 and 1.2
70	4.4	BSIM4SOI
73	1.1.2	HiSIM_HV1.1.2
73	1.2.1	HiSIM_HV1.2.1
68	2.3.0 [default]	HiSIM2.3.0
55	2.6	EKV Model
50	N/A	Philips MOS9 Model
62	1, 2	RPI TFT Model
63	1100	Philips MOS11 Model Level 1100
63	1101	Philips MOS11 Model Level 1101
63	1102.3 [default]	Philips MOS11 Model Level 1102.3
66	2.1	HSPICE HVMOS
69	103.1	PSP Model
70	4.2	BSIM4SOI
72	105	BSIM-MG105, TMI2.01
72	105.031	BSIM-MG105.031
72	105.04	BSIM-MG105.04
72	106	BSIM-MG106
72	106.10	BSIM-MG106.10
72	107.0	BSIM-MG 107.0.0 Beta
6	N/A	Motorola SSIM Model

Chapter 8: Input Netlist

Device Models

Table 26 MOSFET Models (Continued)

Model Level	Version	MOSFET Model
10	N/A	Motorola SSIM SOI Model
13	N/A	BSIM1
3	N/A	SPICE Level-3 Model
2	N/A	SPICE Level-2 Model
1	N/A	SPICE Level-1 Model

Note:

1. The RPI TFT Model was originally developed at Rensselaer Polytechnic Institute.
2. HSIM supports the non-quasi-static (NQS) feature for the BSIM3 and BSIM4 models. The NQS feature is turned on if NQSMOD=1 appears at BSIM3 model file or TRNQSMOD=1 appears at BSIM4 model file.
3. Juncap is the capacitance supported in the MOS9 and MOS11 models.
4. Both the built-in model and UMI approach are supported for the HSIM model by STARC/Hiroshima University in Japan.
5. Consent from Motorola is required to activate the Level-6 SSIM model.
6. HSIM supports SOIMOD=0 (PD mode) SOIMOD=1 (DD mode) and SOIMOD=2 (FD mode) for BSIM3SOlv3.2.

Diode**Syntax**

```
Daa pos_n neg_n model_name <area=val2> <pj=val3> <m=val4>
    <off> <ic=val5> <pgate=val6>
Daa pos_n neg_n model_name <w=val2> <l=val3> <pj=val3>
    <m=val4> <off> <ic=val4>
```

Parameters

Diode parameters are described in the table below.

Table 27 Diode Parameters

Parameter	Default	Description
Daa		Diode element name, which must begin with the character D.
pos_n		Positive node name.
neg_n		Negative node name.
model_name		Diode model name. It is used to refer to a diode model.
area	1	Area of diode. If area is un-specified, it is calculated from w and l.
w		Width of diode.
l		Length of diode.
pj		Periphery of diode junction. If pj is unspecified, the default value is used and it is calculated from w and l.
m	1	Multiplier for parallel instances of diode.
OFF	ON	Set initial condition to OFF for this element in DC operating point calculation.
ic		Initial voltage across the diode, when uic is specified in .tran statement. The .ic statement overrides this value. Refer to .ic on page 330 .
pgate		Length of the side-wall in the AB diffusion area, which is under the gate. In the model card, LG uses this value.

Example

```
DF127 n12 mm99 diode1
```

Diode Model

Syntax

```
.model model_name d <level=val2> <parameter1=val3>  
    <parameter2=val4 ...>
```

Parameters

Diode model parameters are described in the table below.

Table 28 Diode Model Parameters

Parameter	Default	Description
model_name		Diode model name. Diode element refers to the model through this name.
d		Keyword to identify diode model.
level	1	Selector for different levels of diode model: <ul style="list-style-type: none">■ level=1 for junction diode■ level=2 for Fowler-Nordheim diode■ level=3 for junction diode with geometric processing.■ level=6 for JUNCAP2 version 200, 200.1
parameter		Model parameter.

Example

```
.model diode1 d level=1 bv=4.0 cj=2e-9 pb=0.6
```

Bipolar Junction Transistor (BJT)

Syntax

```
Qaa ncol nbase nemit <nsub> model_name <area=><val2>  
    <m=val3> <areab=val4> <areac=val5>
```

Parameters

BJT parameters are described in the table below.

Table 29 Bipolar Junction Transistor (BJT) Parameters

Parameter	Default	Description
Qaa		BJT element name, which must begin with the character Q.
ncol		Collector node name.
nbase		Base node name.
nemit		Emitter node name.
nsub		Substrate node name.
model_name		BJT model name. It is used to refer to a BJT model.
area	1	Area multiplying factor.
m	1	Multiplier for parallel instances of BJT.
areab	Area	Base area multiplying factor.
areac	Area	Collector area multiplying factor.

Example

```
Q100 CX BX EX      bjt1      area=1.5
Q50A 11 265 4    20 bipolar1 1.5
```

Bipolar Junction Transistor (BJT) Model

Syntax

```
.model model_name npn|pnp <level=val1> <parameter1=val2>
<parameter2=val3 ...>
```

Chapter 8: Input Netlist

Device Models

Parameters

BJT Model parameters are described in the table below.

Table 30 Bipolar Junction Transistor (BJT) Model Parameters

Parameter	Default	Description
model_name		BJT model name. BJT element refers to the model through this name.
npn pnp		Keyword to identify either npn or pnp transistor model: select one.
level	1	Selector for different levels of BJT model. <ul style="list-style-type: none">■ level=1 for Gummel-Poon model■ level=4 for VBIC model■ level=6 for Mextram Model■ level=8 for HICUM Model■ level=13 for HICUM0 Model
Parameter		Model parameter.

Example

```
.model bjt1 npn    bf=120 br=30    is=1e-15
```

Table 31 BJT Models

Model Level	Version	BJT Model
1	N/A	Gummel-Poon BJT Model
4	1.15	VBIC-1.15
4	1.2 [default]	VBIC-1.2
6	503	Mextram-503
6	504 [default]	Mextram-504
8	2.1	HICUM version 2.1
8	2.2	HICUM version 2.2
8	2.2.1	HICUM version 2.2.1

Table 31 BJT Models (Continued)

Model Level	Version	BJT Model
8	2.3.1	HICUM version 2.3.1
13	1.1	HICUM0

Junction Field Effect Transistor (JFET) and Metal Semiconductor Field Effect Transistor (MESFET)

Syntax

```
Jaa    ndr nga nso model_name <<area=>area1 |<l=val2>
      <w=val3>> <m=val4> <dtemp=val5>
```

Parameters

JFET and MESFET parameters are described in the table below.

Table 32 Junction Field Effect Transistor and Metal Semiconductor Field Effect Transistor (JFET and MESFET)

Parameter	Default	Description
Jaa		JFET or MESFET element name, which must begin with the character J.
ndr		Drain node name.
nga		Gate node name.
nso		Source node name.
model_name		JFET or MESFET model name. It is used to refer to a model.
area	1	Area multiplying factor.
		Gate length of JFET or MESFET.
w		Gate width of JFET or MESFET.
m	1	Multiplier for parallel instances of JFET or MESFET.

Table 32 Junction Field Effect Transistor and Metal Semiconductor Field Effect Transistor (JFET and MESFET) (Continued)

Parameter	Default	Description
dtemp	0	Difference between the transistor temperature and the global circuit temperature in degrees Celsius.

Example

```
J2    4    3    1    jfet1 area=2
```

Junction Field Effect Transistor (JFET) Model

Syntax

```
.model model_name njf|pjf <level=val2> <capop=val3>
<parameter1=val4> <parameter2=val5 ...>
```

Parameters

JFET Model parameters are described in the table below.

Table 33 Junction Field Effect Transistor (JFET) Model

Parameter	Default	Description
model_name		JFET model name. JFET element refers to the model through this name.
njf pjf		Keyword to identify either n-JFET or p-JFET model: select one.
level	1	Selector for different levels of JFET model. Level value can be 1, 2, or 3. level=3 model is for MESFETs where the following parameters apply: <ul style="list-style-type: none">▪ SAT=0 selects standard Curtice model.▪ SAT=1 selects the model that includes vgst in the argument of hyperbolic tangent function.▪ SAT=2 selects Statz model with constant exponent and denominator.▪ SAT=3 selects Statz model with parameter values for exponent and denominator.

Table 33 Junction Field Effect Transistor (JFET) Model (Continued)

Parameter	Default	Description
capop		Capacitor model selector. capop=0 selects default capacitance equation. capop=1 selects Statz capacitance equation for MESFETs.
parameter		Model parameter.

Example

```
.model jfet1 njf      is=2e-14 acm=0      capop=0
```

IF-ELSE Syntax in Netlists

IF-ELSE syntax can be added to condition-controlled netlists to change the circuit topology, expand the circuit, set parameter values for each device instance, or select different model cards in each IF-ELSE block.

The following is a typical example of an IF-ELSE block.

```
.if (condition1)
<statement_block1>
# The following statement block in {braces} is
# optional, and you can repeat it multiple times:
{ .elseif (condition2)
<statement_block2>
}
# The following statement block in [brackets]
# is optional, and you cannot repeat it:
[ .else
<statement_block3>
]
.endif
```

IF-ELSE Block Rules

There are rules that must be observed when using IF-ELSE blocks for condition-controlled netlists.

1. In an .IF, .ELSEIF, or .ELSE condition statement, complex Boolean expressions must not be ambiguous. For example, change (a==b && c>=d) to ((a==b) && (c>=d)).
2. In an .IF, .ELSEIF, or .ELSE statement block you can include most valid HSIM analysis and output statements. The exceptions are:
.END, .ALTER, .MACRO, .EOM, .GLOBAL, .DEL, .LIB,
.MALIAS, .ALIAS, .LIST, .NOLIST, and .CONNECT statements.
3. You can use IF-ELSEIF-ELSE blocks to select different sub-modules or to structure the netlist by using .INC, .LIB, and .VEC statements.
4. If two or more models in an IF-ELSE block have the same model name and model type, they must also be the same revision level.
5. Parameters in an IF-ELSE block do not affect the parameter value within the condition expression. HSIM updates the parameter value only after it selects the IF-ELSE block.
6. You can nest IF-ELSE blocks.
7. You can include an unlimited number of .ELSEIF statements within an IF-ELSE block.
8. You cannot include sweep parameters or simulation results within an IF-ELSE block.
9. You cannot use an IF-ELSE block within another statement. In the following example, HSIM does not recognize the IF-ELSE block as part of the resistor definition:

```
r 1 0
.if (r_val>10k)
+ 10k
.else
+ r_val
.endif
```

10. You can NOT include IF-ELSE block within a .model statement.
11. You can use IF-ELSE blocks inside .subckt statement; however, you can use only element statements and .MODEL inside these IF-ELSE blocks.
.LIB, .MODEL, or similar commands cannot be placed within these IF-

ELSE blocks. Please refer to [Example 19 on page 270](#) for the proper usage of IF-ELSE blocks for model binning.

12. No sweep variables or .ALTER variables are allowed inside an IF-ELSE evaluation expression. The example below is not allowed.

```
*This is NOT allowed
.dc vg 0 3 0.1 sweep swp_var poi 7 -40 -25 -10 5 20 35 50
.if (swp_var > 0)
...
.else
...
.endif
```

Likewise, the variable name inside the .ALTER variable can not be `swp_var`.

13. When `.subckt` is used inside an IF-ELSE block, the same `.subckt` name can have multiple definitions as shown in the example below.

```
.IF (select==1)
.subckt sub1 n1 n2
...
.ends
.ELSE
.subckt sub1 n1 n2 n3
...
.ends
.ENDIF
```

However, the example below can not be supported because the `.subckt` is re-defined.

```
.IF (select==1)
.subckt sub1 n1 n2
...
.ends
.subckt sub1 n1 n2 n3
...
.ends
.ENDIF
```

IF-ELSE Block Syntax Description

Syntax

```
.IF (condition1)
...

```

Chapter 8: Input Netlist
IF-ELSE Syntax in Netlists

```
<.ELSEIF (condition2)
...
<.ELSE
...
.ENDIF
```

Arguments

condition1

The condition must be true before HSIM executes the commands that follow the .IF statement.

condition2

The condition must be true before HSIM executes the commands that follow the .ELSEIF statement. HSIM executes the commands that follow *condition2*, only if *condition1* is false and *condition2* is true.

Description

HSIM executes the commands that follow the first .ELSEIF statement, only if *condition1* in the preceding .IF statement is false, and *condition2* in the first .ELSEIF statement is true.

If *condition1* in the .IF statement and *condition2* in the first .ELSEIF statements are both false, then HSIM moves on to the next .ELSEIF statement, if there is one. If this second .ELSEIF condition is true, HSIM executes the commands that follow the second .ELSEIF statement, instead of the commands after the first .ELSEIF statement.

HSIM ignores the commands in all false .IF and .ELSEIF statements, until it reaches the first .ELSEIF condition that is true. If no .IF or .ELSEIF condition is true, HSIM continues to the .ELSE statement.

The .ELSE statement precedes one or more commands in a conditional block, after the last .ELSEIF statement, but before the .ENDIF statement. HSIM executes these commands by default, if the conditions in the preceding .IF statement, and in all of the preceding .ELSEIF statements in the same conditional block, are all false.

The .ENDIF statement ends a conditional block of commands that begin with an .IF statement.

Example

```
.IF (a==b)
.INCLUDE /myhome/subcircuits/diode_circuit1
...
ELSEIF (a==c)
.INCLUDE /myhome/subcircuits/diode_circuit2
...
ELSE
.INCLUDE /myhome/subcircuits/diode_circuit3
...
ENDIF
```

IF-ELSE Netlist Examples

The following examples demonstrate some of the ways in which IF-ELSE blocks can be used in condition-controlled netlists.

Chapter 8: Input Netlist

IF-ELSE Syntax in Netlists

Example 15 IF-ELSE Block in Subcircuit Definition

```
*****
* This Case is for If...else
* The subckt call in if...else block
.GLOBAL VNODE
** Options
.OPTION POST=2 $PROBE NODE LIST
** Parameter
.PARAM VDD=3v
.PARAM VHI=1.0V VLO='-VHI'
.PARAM TDELAY=0 TRISE=0.1n TFALL=0.1n TPLAT=5n TPERIOD=10n
** Analysis
.TRAN 10p 20n
** Probe
.PROBE TRAN
+    VIN = V(IN)
+    VOUT = V(OUT)
** Circuit (Source and Load)
VCC   VNODE   0   VDD
VIN   IN      0   PULSE ( VHI VLO TDELAY TRISE TFALL TPLAT TPERIOD )
RIN   IN      0   50
** Subckt Definition
* vth=0.7v
.SUBCKT INV1 IN OUT L=0.1u W=1u PD=4u PS=4u AD=10p AS=10p
MN1   OUT      IN   GND   GND   NCH1 L=L W=W PD=PD PS=PS AD=AD AS=AS
MP1   OUT      IN   VNODE VNODE PCH1 L=L W=W PD=PD PS=PS AD=AD AS=AS
.ENDS
.MODEL NCH1 NMOS LEVEL=49 VERSION=3.2 VTH0=0.8
.MODEL PCH1 PMOS LEVEL=49 VERSION=3.2 VTH0=-0.8
*vth=1.48v
.SUBCKT INV2 IN OUT L=1u W=20u PD=40u PS=40u AD=20p AS=20p
MN2   OUT      IN   0      0      NCH2 L=L W=W PD=PD PS=PS AD=AD AS=AS
MP2   OUT      IN   VNODE VNODE PCH2 L=L W=W PD=PD PS=PS AD=AD AS=AS
.ENDS
.MODEL NCH2 NMOS LEVEL= 2
+    vto = 1.48    kp = 8.0e-05    gamma = 0.6
.MODEL PCH2 PMOS LEVEL=2
+    vto = -1.48   kp = 7.5e-05   gamma = 0.8
** If...Else... Block
.IF ( VHI > 1.5 )
    X1 IN OUT INV2 L=2u w=20u
.ELSEIF ( 0.6 < VHI < 1.5 )
    X1 IN OUT INV1
    XAMP OUT 0 AMPOUT AMP
    RL      AMPOUT 0  2K
    .PROBE V(AMPOUT)
.ELSE
    R1 IN OUT 1Meg
```

```

.ENDIF
** Load
RLOAD OUT 0 50

** AMP
.SUBCKT AMP V+ V- VO
RI V+ V- 2MEG
RO VD VO 75
E1 VD 0 V+ V- 20
.ENDS AMP

.END
*****

```

Example 16 IF-ELSE Block to Define Parameter Values

```

***** OPTION LIST POST OPTS *****
.OPTION LIST POST OPTS
.TEMP -40 -25 -10 5 20 35 50
.DC VG 0 3 0.1
.IF (TEMPER>25)
    .PARAM PVTO = 100m
.ELSE
    .PARAM PVTO = 200m
.ENDIF
**** MOSFET *****
M1 D G S B NMOS L=1u W=1u
**** Bias Condition ***
VD D 0 3
VG G 0 1
VS S 0 0
VB B 0 0
.MODEL NMOS NMOS LEVEL = 3
+ VTO = PVTO
.END
*****

```

Chapter 8: Input Netlist
IF-ELSE Syntax in Netlists*Example 17 IF-ELSE Block to Define Model Cards*

```
*****
* if-else with model cards selected
.option post
vd d 0 3v
vg g 0 1v
vs s 0 0v
vb b 0 0v
.param length=0.25e-6
m1 d g s b nch l='length' w=0.5e-6
*** select model based on channel length
.if ((length>=2.4e-007) && (length<3.5e-007))
    .model nch nmos level=54 version=4.2
    + vth0 = 0.4
.elseif ((length>=3.5e-007) && (length<5e-007))
    .model nch nmos level=54 version=4.2
    + vth0 = 0.42
.endif
.op
.dc vg 0 3 0.1
.print i(vd)
.end
*****
```

Example 18 Nested IF-ELSE Blocks

```
*****
.option post list node probe
.probe v(*)  
  
.param v_sw1=1
+      v_sw2=1
.param test1=0 test2=0 test3=0
+      test4=0  
  
.if (v_sw1 == 1)
.param test4=1
    .if (v_sw2 == 1)
        .tran 0.05n 50n
        Vin1 in_in1 0 pulse(0 1.0 0 500p 500p 4.5n 10n)
        .param test1=1
    .elseif (v_sw1 == 2)
    .else
        .tran 0.05n 100n
        Vin1 in_in1 0 pulse(0 1.0 0 500p 500p 9.5n 20n)
        .param test2=1
    .endif
.else
    .tran 0.05n 200n
    Vin1 in_in1 0 pulse(0 1.0 0 500p 500p 19.5n 40n)
    .param test3=1
.endif  
  
Xbuf1 in_in1 out_out1drv1  
  
Rload1 out_out1 gnd R=10k  
  
.subckt drv1 in out
Rout1 in out0 R=5
Cout1 in gnd C=10p
Rdamp1 out0 out R=1
.ends  
  
.print tran par('test1') par('test2') par('test3')
+      par('test4')
+      par('v_sw1') par('v_sw2')
.end
*****
```

Chapter 8: Input Netlist
Driving Sources and Input Stimuli**Example 19 IF-ELSE Blocks for Model Binning**

```
*****
* Test circuit with model binning
.subckt nchi_ll 1 2 3 4
.param length=0.25e-6
*** select model based on channel length
.if ((length>=2.4e-007) && (length<3.5e-007))
m1 1 2 3 4 nch1 l='length' w=0.5e-6
.elseif ((length>=3.5e-007) && (length<5e-007))
m1 1 2 3 4 nch2 l='length' w=0.5e-6
.endif
.ends nchi_ll
.model nch1 nmos level=54 version=4.2
+ vth0 = 0.4
.model nch2 nmos level=54 version=4.2
+ vth0 = 0.42
vdd vdd gnd 1.0v
vsub vsub gnd 0.0v
vin 1 gnd dc 1.0v
xt1 2 1 gnd gnd nchi_ll length=2.5e-7
xt2 2 1 gnd gnd nchi_ll length=4.5e-7
.option post
.op
.end
*****
```

Driving Sources and Input Stimuli

Element statements for independent sources and dependent sources that drive an electronic circuit are summarized in the following sections:

- [Independent Voltage Source](#)
- [Independent Current Source](#)
- [Pulse Source Function \(PULSE\)](#)
- [Sinusoidal Source Function \(SIN\)](#)
- [Single-Frequency Frequency Modulation \(SFFM\) Source Function](#)
- [Amplitude Modulation \(AM\) Source Function](#)
- [Exponential Source Function \(EXP\)](#)
- [Piecewise Linear Source Function \(PWL\)](#)

- Piecewise Linear Source Function with High Impedance State (PWLZ)
 - Voltage-Controlled Current Source (VCCS)
 - Voltage-Controlled Voltage Source (VCVS)
 - Ideal Transformer
 - Current-Controlled Current Source (CCCS)
 - Current-Controlled Voltage Source (CCVS)
 - Voltage-Controlled Resistor (VCR)
 - Voltage-Controlled Capacitor (VCC)
 - Laplace Element
-

Independent Voltage Source

Syntax

Vaa pos_n neg_n <>dc=>dcval <tranfun>

Parameters

Independent voltage source parameters are described in the table below.

Table 34 Independent Voltage Source Parameters

Parameter	Default	Description
vaa		Independent voltage source name, which must begin with the character V.
pos_n		Positive node name.
neg_n		Negative node name.
dc		Keyword for DC source value in volt.
dcval	0	The DC voltage source value. If not specified, the default value is used.
tranfun		Transient source function.

Chapter 8: Input Netlist
Driving Sources and Input Stimuli**Example**

VDD	1	0	3.3V
VBB	A10	Gnd	1.5

Independent Current Source**Syntax**

```
Iaa pos_n    neg_n <>dc=>dcval <tranfun> <m=val2>
```

Parameters

Independent current source parameters are described in the table below.

Table 35 Independent Current Source Parameters

Parameter	Default	Description
Iaa		Independent current source name. It must begin with the character I.
pos_n		Positive node name.
neg_n		Negative node name.
dc		Keyword for DC source value in ampere.
dcval	0	The DC current source value. If not specified, the default value used.
transfun		Transient source function.
m	1	Multiplier for parallel instances of independent current source.

Example

```
IPX N1 N2 dc 0.005
```

Pulse Source Function (PULSE)

Syntax

```
pulse <(> val1 val2 <t_delay <t_rise <t_fall <pulse_width  
<pulse_period >>>> <) >
```

Parameters

Pulse source function parameters are described in the table below.

Table 36 Pulse Source Function Parameters

Parameter	Default	Description
pulse		Keyword to identify pulse source function.
val1		Initial value of the pulse source.
val2		Pulse peak value.
t_delay	0.0	Delay time before the first onset ramp. The unit of measure is seconds.
t_rise		Rise time of the pulse. The unit of measure is second.
t_fall		Fall time of the pulse. The unit of measure is second.
pulse_width		Pulse width. The unit of measure is second.
pulse_period		The pulse period. The unit of measure is second.

Example

```
VIN 1 0 pulse (0 2.5      2N    0.5N 0.5N 5N 10N)
```

```
I2   2 0 pulse (0 1e-3     1N    0.5N 0.5N 3N 6N)
```

Sinusoidal Source Function (SIN)

Syntax

```
sin <(> v_dc v_amplitude <freq <t_delay <ATHETA <APHI >>>  
<) >
```

Chapter 8: Input Netlist
Driving Sources and Input Stimuli

Parameters

Sinusoidal source function parameters are described in the table below.

Table 37 Sinusoidal Source Function Parameters

Parameter	Default	Description
sin		Keyword to identify sinusoidal source function.
v_dc		Offset value of the source.
V_amplitude		Amplitude value of the source.
freq		Frequency of the source.
t_delay		Delay time before the onset of sinusoidal value. The unit of measure is second.
A _{THETA}	0	Damping factor in unit of 1/second.
A _{PHI}	0	Phase delay in unit of degree.

Example

```
VDA      ax 0 sin (0 3.3   100meg 1N   2e8)
Isource n1 0 sin (0 1e-3 5e6      ON     0     30)
```

Single-Frequency Frequency Modulation (SFFM) Source Function

Syntax

```
sffm <(> v_offset v_amplitude <freq_ca <md_ind <freq_sig>>>
<)>
```

Parameters

SFFM source function parameters are described in the table below.

Table 38 SFFM Source Function Parameters

Parameter	Default	Description
sffm		Identifies single-frequency FM source function.

Table 38 SFFM Source Function Parameters (Continued)

Parameter	Default	Description
v_offset		Offset value of the source.
v_amplitude		Amplitude value of the source.
freq_ca		Carrier frequency in Hz.
md_ind	0	Modulation index.
freq_sig		Signal frequency in Hz.

Example

```
V2      bx 0 sffm (0 1.8    1.0e6    4    10K)
```

Amplitude Modulation (AM) Source Function**Syntax**

```
am <(> amplitude offset freq_mod freq_ca <td> <) >
```

Parameters

AM source function parameters are described in the table below.

Table 39 Amplitude Modulation (AM) Source Function Parameters

Parameter	Description
amplitude	Signal amplitude.
offset	Offset value.
freq_mod	Modulation frequency in Hz.
freq_ca	Carrier frequency in Hz.
td	Delay time.

Example

```
VDA      ax 0 am (1.8 0    1k 100k    0.5m)
```

Exponential Source Function (EXP)

Syntax

```
exp <(> val1 val2 <td_rise <tr_const <td_fall <tf_const >>>
<)>
```

Exponential source function parameters are described in the table below.

Table 40 Exponential Source Function Parameters

Parameter	Default	Description
exp		Keyword to identify exponential source function.
val1		Initial value of the exponential source. The unit of measure is volt.
val2		Pulse peak value. The unit of measure is volt.
td_rise	0	Delay time for rising edge. The unit of measure is second.
td_fall		Delay time for falling edge. The unit of measure is second.
t_fall		Fall time of the pulse. The unit of measure is second.
tr_const		Time constant for rising edge. The unit of measure is second.
tf_const		Time constant for falling edge. The unit of measure is second.

Example

```
V77      77    0 exp (0    2.5      2N    30N    2N    40N)
Isource   n1    0 exp (0 1e-3      2N    15N    2N    30N)
```

Piecewise Linear Source Function (PWL)

Syntax

```
pwl <(> t1 val1 <t2 val2 ...tN valN> <r <=t_REPEAT>> <td=valm
> <)>
```

Parameters

Piecewise linear source function parameters are described in the table below.

Table 41 Piecewise Linear Source Function Parameters

Parameter	Default	Description
pwl		Keyword to identify piecewise linear source function.
tk		kth time point.
valk		Source value at kth time point.
r		Keyword to identify repeat function.
t_REPEAT	0	Start time for repeat function which must be less than the greatest time point tN. The unit of measure is second.
td		Delay time.

Example

```
VXD 5 0 pwl (0 0 2N 3.3 10N 3.3 12N 0 20N 0 22N 3.3)
```

Piecewise Linear Source Function with High Impedance State (PWLZ)

Syntax

```
pwlz <(> t1 val1 <t2 val2 t3 z t4 val4 ...> <r <=t_REPEAT
>> <td=val5 > <> >
```

z can be used in place of source value. The voltage source will be disconnected for time periods marked with the keyword z.

Example

```
VXD 5 0 pwlz (0 0 2N 0.75 10N 1.5 50N z 60N 0.75)
```

In this example, node 5 is connected to a 0V source at time 0, and rises from 0V to 0.75V in 2 ns. Between 2 ns and 10 ns, the voltage source value rises from 0.75V to 1.5V. The voltage source value stays at 1.5V between 10 ns and 50 ns. Starting from 50 ns, node 5 is disconnected from the voltage source until 60 ns. It is connected to a 0.75V voltage source after 60 ns.

Voltage-Controlled Current Source (VCCS)

VCCS syntax statements are described below:

Linear Syntax

```
Gaa pos_n neg_n <vccs> nc+ nc- transconductance  
<max=val2> <min=val3> <m=val4>
```

Polynomial Syntax

```
Gaa pos_n neg_n <vccs> poly(N) nc1+ nc1- ....  
<ncN + ncN-> <min=val2> <max=val3> <m=val4> p0 <p1 ....>
```

Piecewise Linear Syntax

```
Gaa pos_n neg_n <vccs> pwl(1) nc1+ nc1- <delta=val2>  
<m=val3> x1, y1, x2, y2 ....
```

Multi-Input Gates Syntax

```
Gaa pos_n neg_n <vccs> logic_gate(m) nc1+ nc1- .... ncm+  
ncm- <scale=val5> <m=val6> <delta=val2>
```

Behavioral Current Source Syntax

```
Gaa pos_n neg_n <vccs> cur="expr"
```

Parameters

VCCS parameters are described in the table below.

Table 42 Voltage-Controlled Current Source (VCCS) Parameters

Parameter	Default	Description
Gaa		Voltage-controlled current source name which must begin with the character G.
pos_n		Positive node name for controlled source.
neg_n		Negative node name for controlled source.
vccs		Identify voltage-controlled current source.
nc+/-		Positive or negative controlling node. Use one pair for each dimension.
transconductance		Transfer factor from voltage-to-current.
max		Maximum current value.

Table 42 Voltage-Controlled Current Source (VCCS) Parameters (Continued)

Parameter	Default	Description
min		Minimum current value.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
poly		Polynomial controlling function.
p0, p1, ...		Polynomial coefficients.
pwl		Piecewise linear controlling function.
delta	0.25 of smallest breakpoint distance	Parameter to control piecewise linear corners.
m		Multiplier for parallel instances.
logic_gate		Choose one from reserved words: <ul style="list-style-type: none"> ▪ AND ▪ NAND ▪ OR ▪ NOR
cur		Keyword to identify current output.
expr		A mathematical expression defining the current from node pos_n to node neg_n. The expression may contain controlling variables of node voltages and/or branch currents in the circuit.

Example

GXY 5 0 1 0 0.002

Voltage-Controlled Voltage Source (VCVS)

VCVS syntax statements are described below:

Linear Syntax

```
Eaa pos_n neg_n <vcvs> nc+ nc- vgain <min=val2> <max=val3>
```

Polynomial Syntax

```
Eaa pos_n neg_n <vcvs> poly(N) nc1+ nc1- ... <ncN+ ncN->  
<min=val2> <max=val3> p0 <p1 ...>
```

Piecewise Linear Syntax

```
Eaa pos_n neg_n <vcvs> pwl(1) nc1+ nc1- <delta=val2> x1,  
y1, x2, y2 ...
```

Multi-Input Gates Syntax

```
Eaa pos_n neg_n <vcvs> logic_gate(m) nc1+ nc1- ...  
ncm+ ncm- <delta=val2>
```

Delay Element Syntax

```
Eaa pos_n neg_n <vcvs> delay nc+ nc- td=td1
```

Behavior Voltage Source Syntax

```
Eaa pos_n neg_n <vcvs> <scale=val5> <m=val6> vol="expr"
```

VCVS Parameters

VCVS parameters are described in the table below.

Table 43 Voltage-Controlled Voltage Source (VCVS) Parameters

Parameter	Default	Description
Eaa		Voltage-controlled voltage source name, which must begin with the character E.
pos_n		Positive node name for controlled source.
neg_n		Negative node name for controlled source.
vcvs		Keyword to identify voltage-controlled voltage source.
nc+/-		Positive or negative controlling node. Use one pair for each dimension.
vgain		Voltage gain factor.
max		Maximum voltage value.
min		Minimum voltage value.

Table 43 Voltage-Controlled Voltage Source (VCVS) Parameters (Continued)

Parameter	Default	Description
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
poly		Polynomial controlling function.
p0, p1, ...		Polynomial coefficients.
pwl		Piecewise linear controlling function.
delta	0.25 of smallest breakpoint distance	Parameter to control piecewise linear corners.
logic_gate		Choose one from reserved words <ul style="list-style-type: none"> ▪ AND ▪ NAND ▪ OR ▪ NOR
delay		Delay element.
td		Delay time.
vol		Keyword to identify voltage output.
expr		A mathematical expression defining the current from node pos_n to node neg_n. The expression may contain controlling variables of node voltages and/or branch currents in the circuit.

Example

```
EXY 5    0    1    0    0.5
```

Ideal Transformer

Syntax

```
Exxx n+ n- <transformer> in+ in- k
```

Chapter 8: Input Netlist

Driving Sources and Input Stimuli

Parameters

Ideal transformer parameters are described in the table below.

Table 44 Ideal Transformer Parameters

Parameter	Description
Exxx	Ideal transformer name. The name must begin with E.
n +/-	Positive or negative control element nodes.
in +/-	Positive or negative controlling nodes.
k	Ideal transformer turn ratio: $V(in+, in-) = k * V(n+, n-)$

Example

```
E1 t1      t2      transformer    b1      b2      5
```

Current-Controlled Current Source (CCCS)

CCCS syntax statements are described below:

Linear Syntax

```
Faa pos_n neg_n <cccs> vc igain <min=val2> <max=val3>
<m=val4>)
```

Polynomial Syntax

```
Faa pos_n neg_n <cccs> poly(N) vc1 ... vcN <min=val2>
<max=val3> <m=val4> p0 <p1 ...>
```

Piecewise Linear Syntax

```
Faa pos_n neg_n <cccs> pwl(1) vc <delta=val2> <m=val3> x1,
y1, x2, y2 ...
```

Multi-Input Gates Syntax

```
Faa pos_n neg_n <cccs> logic_gate(m) vc1 ... vcm
<delta=val2> <m=val3>
```

Parameters

CCCS parameters are described in the table below.

Table 45 Current-Controlled Current Source (CCCS) Parameters

Parameter	Default	Description
Faa		Current-controlled current source name which must begin with the character F.
pos_n		Positive node name for controlled source.
neg_n		Negative node name for controlled source.
cccs		Keyword to identify current-controlled current source.
vc		Voltage source name for the controlling current to flow. Use one for each dimension.
igain		Current gain factor.
max		Maximum current value.
min		Minimum current value.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
poly		Polynomial controlling function.
p0, p1		Polynomial coefficients.
pwl		Piecewise linear controlling function.
delta	0.25 of smallest breakpoint distance	Parameter to control piecewise linear corners.
m		Multiplier for parallel instances.

Chapter 8: Input Netlist
Driving Sources and Input Stimuli

Table 45 Current-Controlled Current Source (CCCS) Parameters (Continued)

Parameter	Default	Description
logic_gate		Choose one from reserved words: <ul style="list-style-type: none"> ▪ AND ▪ NAND ▪ OR ▪ NOR
xk		Controlling current through vc source.
yk		Corresponding output current of xk.

Example

```
F1 5 0 VIN    0.2
```

Current-Controlled Voltage Source (CCVS)

CCVS syntax statements are described below:

Linear Syntax

```
Haa pos_n neg_n <ccvs > vc transresistance <min=val2>
<max=val3>
```

Polynomial Syntax

```
Haa pos_n neg_n <ccvs > poly(N) vc1 ..... vcN <min=val2>
<max=val3> p0 <p1 ...>
```

Piecewise Linear Syntax

```
Haa pos_n neg_n <ccvs > pwl(1) vc <delta=val2> x1, y1, x2,
y2 ...
```

Multi-Input Gates Syntax

```
Haa pos_n neg_n vcr logic_gate(m) vc1+ vc1- ... vcm+ vcm-
<delta=val2> x1, y1, x2, y2 ...
```

Parameters

CCVS parameters are described in the table below.

Table 46 Current-Controlled Voltage Source (CCVS) Parameters

Parameter	Default	Description
Haa		Voltage-controlled resistor name which must begin with the character H.
pos_n		Positive node name.
neg_n		Negative node name.
ccvs		Keyword to identify voltage-controlled resistor.
nc+/-		Positive or negative controlling node. Use one pair for each dimension.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
p0, p1, ...		Polynomial coefficients.
poly		Polynomial controlling function.
pwl		Piecewise linear controlling function.
delta	0.25 of the smallest breakpoint distance	Parameter to control piecewise linear corners.
logic_gate		Choose one from reserved words: ■ AND ■ NAND ■ OR ■ NOR
xk		Controlling voltage through vc source.
yk		Corresponding output element value.

Example

H23 5 0 VIKIN 121.0

Voltage-Controlled Resistor (VCR)

VCR syntax statements are described below:

Linear Syntax

```
Gaa pos_n neg_n vcr vc+ vc- transfactor
```

Polynomial Syntax

```
Gaa pos_n neg_n vcr poly(N) vc1+ vc1- ... <vcN+ vcN-> P0  
<P1 ...>
```

Piecewise Linear Syntax

```
Gaa pos_n neg_n vcr pwl(1) vc+ vc- <delta=val2> x1, y1, x2,  
y2 ...
```

Multi-Input Gates Syntax

```
Gaa pos_n neg_n vcr logic_gate(m) vc1+ vc1- ... vcm+ vcm-  
<delta=val2>
```

Parameters

VCR parameters are described in the table below.

Table 47 Voltage-Controlled Resistor (VCR) Parameters

Parameter	Default	Description
Gaa		Voltage-controlled resistor name which must begin with the character G.
pos_n		Positive node name.
neg_n		Negative node name.
vcr		Keyword to identify voltage-controlled resistor.
nc+/-		Positive or negative controlling node. Use one pair for each dimension.
N	1	Polynomial dimension for controlling sources. N is 1, 2, or 3.
p0, p1, ...		Polynomial coefficients.
poly		Polynomial controlling function.

Table 47 Voltage-Controlled Resistor (VCR) Parameters (Continued)

Parameter	Default	Description
pw1		Piecewise linear controlling function.
delta	0.25 of the smallest breakpoint distance	Parameter to control piecewise linear corners.
logic_gate		Choose one from reserved words: ■ AND ■ NAND ■ OR ■ NOR
xk		Controlling voltage through vc source.
yk		Corresponding output element value.

Example

```
Gsw3    3      0 vcr      PWL(1) 2      0      0V,      1K,      1V, 10K
```

Voltage-Controlled Capacitor (VCC)

Syntax

```
Gaa pos_n neg_n vccap pw1(1) nc+ nc- <delta=val2> x1, y1,  
x2, y2 ... <tcl=val3 <tc2=val4>> <scale=val5> <m=val6>  
<ic=val7>
```

Parameters

VCC parameters are described in the table below.

Table 48 Voltage-Controlled Capacitor (VCC) Parameters

Parameter	Default	Description
Gaa		Voltage-controlled capacitor name, which must begin with the character G.
pos_n		Positive node name.

Table 48 Voltage-Controlled Capacitor (VCC) Parameters (Continued)

Parameter	Default	Description
neg_n		Negative node name.
vccap		Keyword to identify voltage-controlled capacitor.
nc+/-		Positive or negative controlling node.
pwl		Piecewise linear controlling function.
delta	0.25 of the smallest breakpoint distance	Parameter to control piecewise linear corners.
xk		Controlling current through vc source.
yk		Corresponding output current of xk.
tc1		First-order coefficient for temperature effect.
tc2		Second-order coefficient for temperature effect.
scale	1.0	Element scale parameter.
m	1	Multiplier for parallel instances of capacitor.
ic		Initial voltage across the capacitor. The .ic Statement will override this value.

Laplace Element

A laplace element performs transformation of an input signal to output signal given by an s-domain rational function R(s) as shown in this [Equation](#) :

$$R(s) = \frac{Pn(s)}{Qm(s)}$$

where numerator Pn(s) and denominator Qm(s) are polynomials in complex frequency s with real coefficients of power n and m respectively as shown in [Equation 20](#):

Example 20

$$P_n(s) = a_0 + a_1*s + a_2*s^2 + \dots + a_n*s^n,$$

$$Q_m(s) = b_0 + b_1*s + b_2*s^2 + \dots + b_m*s^m.$$

These polynomials can be specified either through their coefficients or through their roots. The power of the denominator should be larger than the power of the numerator.

The input and output signals can have the following meanings:

Input signal

Node voltage

Output signal

Node voltage or branch current. Depending on which type of output signal is used (voltage or current) either of the following laplace elements are possible:

- VCVS
- VCCS

If rational function R(s) is specified through polynomial coefficients, then the following syntax is used for Laplace element as shown in the following syntax:

Syntax

```
Gname out1 out2 LAPLACE in1 in2 a0 a1 a2...an /
  b0 b1...bm <scale=value>
Ename out1 out2 LAPLACE in1 in2 a0 a1 a2...an /
  b0 b1...bm <scale=value>
```

Parameters

In this syntax, the following definitions apply:

Line 1

Specifies VCCS

Line 2

Specifies VCVS

out1 and out2

Output nodes

in1 and in2

Input nodes

Chapter 8: Input Netlist
Driving Sources and Input Stimuli**LAPLACE**

A keyword to distinguish the element

a0 ... an

Coefficients of polynomial of power n in numerator

b0 ... bm

Coefficients of polynomial of power m in denominator.

scale

Optional parameter used as a multiplier for output.

/

Separator between numerator and denominator coefficients.

Example

```
g1 0 m1 LAPLACE n1 0 1 / 1 0.2e-9 scale=2  
e2 m2 0 LAPLACE n1 0 1 / 1 0.2e-9 scale=2
```

Rational function R(s) can be specified through the polynomial's root. Because polynomials have real coefficients, their roots should be either real or complex conjugate. In case of multiple roots they must be specified as many times as the multiplicity of the roots.

Syntax

```
Gname out1 out2 POLE in1 in2 a az1,fz1 ... azn,fzn /  
    b bp1,fp1 ... bpm,fpm <scale=value>  
Ename out1 out2 POLE in1 in2 a az1,fz1 ... azn,fzn /  
    b bp1,fp1 ... bpm,fpm <scale=value>
```

Parameters

In this syntax, the following definitions apply:

$$P_n(s) = a * (s + a_1 - 2 * \pi * j * f_z1) * \dots * (s + a_n - 2 * \pi * j * f_{zn}),$$

Numerator polynomial

$$Q_m(s) = b * (s + b_1 - 2 * \pi * j * f_p1) * \dots * (s + b_n - 2 * \pi * j * f_{pn}),$$

Denominator polynomial

$$\pi = 3.1415926535$$

j:

An imaginary unit

Comma:

Separates real and imaginary parts of roots. Commas are optional.

Scale:

A multiplier for output variable. If polynomial is constant has only zero root, it can be omitted.

/:

Separator between numerator and denominator coefficients.

Example

```
e3 m3 0 POLE n3 0 5e9 / 1 +5e9, 0
g4 0 m4 POLE n4 0 25e18 / 1 +5e9, -1e8 +5e9, +1e8 scale=2
```

Digital Vector File

Digital vector input and output files for HSIM are specified with the [HSIMVECTORFILE on page 192](#) command. A digital vector file consists of a vector definition section and a vector data section. The vector definition section describes the signals for each of the following:

- Vector stimulus
- Vector type (input stimulus / expected digital output)
- Rise/fall time
- Driving strength
- Cycle period (optional)

The vector data section describes the signal states at specified times in tabular format. The vector definition section must be placed before the vector data section. Long lines in vector file can be split by putting a back-slash (\) at the end of a line or a plus sign (+) at the beginning of next line. A line beginning with a semi-colon character ; is treated as a comment line.

The global parameters defined in the simulation file can be referenced inside the vector file. This feature is available to the value field of the following commands:

- delay
- rise
- fall

Chapter 8: Input Netlist

Digital Vector File

- slope
- period
- check_window (start/stop)
- tunit
- logichv
- logiclv
- vhth
- vlth
- resistance

Example

```
/* in simulation file */
.param LOGICHV_PARAM=2.7
.param LOGICLV_PARAM=0.3
.....
/* in vector file */
logichv LOGICHV_PARAM
; define logichv to be 2.7 V
logiclv LOGICLV_PARAM
; define logiclv to be 0.3 V
.....
```

Vector Statements

Following are descriptions of the vector statements:

- [check_window](#)
- [delay|tdelay](#)
- [enable](#)
- [io](#)
- [logichv or vih|logiclv or vil|logicxv](#)
- [mask](#)
- [period](#)
- [radix](#)
- [resistance|out|outz](#)
- [signal|vname](#)

- slope|rise|trise|fall|tfall
- stop_at_error
- tskip
- triz
- tunit
- vchk_ignore
- vhth|voh|vlth|vol
- vref
- vth

check_window

The check_window statement is to define a time window around the vector strobe time or user-defined first_time such that the output comparison, for signals specified as output in .io statement, is checked over this time window.

Syntax

```
check_window start_offset stop_offset steady <mask_name>
or
check_window start_offset stop_offset steady period_time
    first_time <mask_name>
```

Description

In the first syntax statement above, the values specified by start_offset and stop_offset define the time window as [t-start_offset, t+stop_offset] in which t is the vector stop time. The unit of time for start_offset and stop offset is nanosecond. When steady is specified as 1, the comparison check passes if the output state matches with the expected state throughout the time window period. When steady is specified as 0, the output comparison passes as long as the output state ever reaches the expected state at any time within the window. mask_name is optional. When mask_name is specified, check_window applies to the signals defined under mask_name only.

In the second syntax statement above, the values specified by start_offset and stop_offset define the time window as [t-start_offset, t+stop_offset] in which t is the first time. The checking will be repeated every period_time. The unit of time for start_offset, stop_offset, period_time, and first_time is nanosecond. When steady is specified as 3, the comparison check passes if the output state matches with the expected state throughout the time window period. When

Chapter 8: Input Netlist

Digital Vector File

steady is specified as 2, the output comparison passes as long as the output state ever reaches the expected state at any time within the time window. mask_name is optional. When mask_name is specified, check_window applies to the signals defined under mask_name only.

Examples

	a	b	c	d
signal				
radix	1	1	1	1
io	i	o	o	i
check_window	1.5	2.0	1	
	1	1	0	1
.....				
signal	clk	D2	D3	
radix	1	1	1	
io	i	o	o	
mask	m1	D2		
check_window	0.2	0.2	3 10 4	m1
0 0 X X				
5 1 0 1				
6.5 1 1 1				
8 1 0 0				
10 0 0 1				
10.2 0 1 0				
15 1 1 0				
15.2 1 0 1				
17.3 1 1 1				
18.6 1 0 0				
20 0 0 0				
.....				

In this example, the output comparison on signal D2 passes if the logic state of D2 is 0 between 3.8ns to 4.2ns, and the logic state of D2 is 1 between 13.8ns to 14.2ns, and so on.

delay|tdelay

Description

The delay statement defines the timing shift between the actual vector time and the vector time defined in the vector file. The statement begins with a keyword delay or tdelay followed by the delay time. The unit of time is nanosecond. The delay value can be either positive or negative. If the delay value is negative, then the actual vector time is sooner than the specified vector time.

Syntax

```
delay|tdelay    time1
```

Example

```
delay      2
```

This shifts each vector time by 2 ns later.

enable**Description**

The enable statement specifies the controlling signal of a bidirectional signal as shown in the example below.

Example

```
radix 1 1 1 1 1 1  
vname SCL SDA TEST1 TEST2 VEP_PAD A  
io b b i i b i  
enable A 0 1 0 0 0 0
```

This enable statement indicates that whenever signal A becomes 1, the SDA will be in the output state. To reverse the control logic, add a tilde character (~) in front of control signal: ~A.

io**Description**

The I/O statement defines the type of each column of signal node(s). The statement starts with the keyword io followed by the type for each column. A signal type can either be one of the following:

- i: As input stimulus to the circuit.
- o: As expected output to compare with simulated output.
- u: Unused.
- b: Bidirectional nodes

Spaces can be inserted between columns to improve readability. If the io statement is not specified, all the signals are assumed to be input signals.

Syntax

```
io dddd dddd ...
```

Chapter 8: Input Netlist
Digital Vector File**Example**

```
io iiii oooo uuuu bb
```

The io statement indicates the signal nodes in the first 4 columns are input signals and the nodes in the next columns are outputs that require output comparison. The last two nodes are bi-directional. The data assigned with a z-state will mark a high impedance bidirectional state when the output signal emerges from the bidirectional terminal.

logichv or vih|logiclv or vil|logicxv**Description**

Note: For compatibility with vector file statements of other simulators, the following are interchangeable:

- logichv and vih
- logiclv and vil
- logicxv

The logichv (or vih), logiclv (or vil), and logicxv statements have the following characteristics:

- Specify the voltage values of logic 1, logic 0, and logic x for all input stimuli.
- Begin with the keyword logichv (or vih) or logiclv (or vil) followed by a voltage value.
- If the unit is not specified, volt is used as the default unit.
- logichv (or vih), logiclv (or vil) can also be specified from parameters HSIMLOGICHV and HSIMLOGICLV.
- If the logichv and logiclv values are not specified in the vector file or by using HSIM parameters, the default values will be used:
 - 3V for logichv
 - 0V for logiclv
 - logiclv for logicxv

A specific voltage can be specified when x is encountered in the vector file.

Syntax

```
logichv|vih|logiclv|vil val2
```

Examples

```
logichv 2.1V
logiclv 0.2V
```

mask

The values specified by slope/rise/fall/logichv/logiclv/vhth/vlth/delay define the analog waveform shape of each signal in the vector file. This is used when signals require values other than the globally defined ones. The mask pattern can be specified to define those selected signals.

The mask statement defines the mask name to represent the mask pattern. The statement starts with a keyword mask, followed by a mask name and then the mask pattern. The mask pattern can be specified by either the values or the signal nodes. If the signal nodes are used to describe the mask pattern, a logic 1 is set to each signal node at the corresponding column location. Any unspecified column is default to logic 0.

Syntax

```
mask  mask_name  dddd dddd ...
mask  mask_name  node1 <node2 ... >
```

Examples

```
signal    a      b      c      d
mask      m1    01      0      1
mask      m2    b      d
```

The above statements specify that both m1 and m2 have a mask pattern of 0101.

For those statements which take an optional mask pattern, the pattern can be specified by either the values or the predefined mask name.

```
signal    a b[0-3]  c  d  e[0-15]
radix    14114444
logichv  3.0
logichv  2.5 0f000000
logichv  2.8 m3
```

This example defines the logic 1 voltage for signals a, d, e[0-7] and e[12-15] as 3.0V. The logic 1 voltage for b[0-3] is 2.5V. The logic 1 voltage for signals c and e[8-11] is 2.8V. The mask pattern for m3 is 001000f0.

Chapter 8: Input Netlist

Digital Vector File

period**Description**

The period statement is used to define the cycle time between two consecutive vector patterns in the vector data section. When the period time is defined, the first column data of each vector pattern should start with digital logic pattern such as 1 and 0. In such case, the i-th vector pattern is implicitly defined to be at time $i-1$ times the cycle time defined in the period statement. If an extra vector needs to be inserted in a cycle, use the pound # character to define a delay time from its predecessor.

Syntax

```
period time1
```

Example

```
period      25
period      25ns
period      0.025us
tskip
```

Each period statement above defines the cycle time to be 25 ns. tskip is described below.

```
=====
period 10
0101 <-- 0ns
#5 0101 <-- vector changed at 5n, which is relative to previous line
1110 <-- 10ns
0101 <-- 20ns
=====
```

radix**Description**

The radix statement specifies the number of signal nodes associated with the column location in the vector file. Each column can specify up to 4 signal nodes, so the valid radix values are from 1 to 4. The statement starts with a keyword radix, followed by a list of numbers from 1 to 4, which represents the radix of the corresponding column. Spaces can be inserted between the numbers. This does not affect functionality, however, it improves readability. This statement must be specified in the vector file.

Syntax

```
radix      dddd dddd ...
```

Example

```
radix    1111 4444 1234
```

The above radix statement indicates there are 12 columns with a total of 30 signal nodes in the vector file. Each digit represents one column and the number of nodes in that column.

resistance|out|outz**Description**

Output resistance can be specified for all the input signals in a vector file by the resistance statement. The statement starts with the keyword resistance (out/ outz) followed by a value for output resistance. If no unit is specified in the resistance value, the default unit is ohms. The default output resistance is 0 ohm.

Syntax

```
resistance | out | outz rval12
```

Example

```
resistance 10
```

signal|vname**Description**

Note: For compatibility with vector files of other simulators, signal and vname are interchangeable.

The signal|vname statement describes all the nodes that this vector file is driving or checking. The statement starts with the keyword signal|vname followed by a list of node names. The number of node names specified in the signal|vname statement must be the same as the total number of signal nodes defined in the radix statement. Bus notation is accepted in the node name specification. The followings are the bus notation accepted by HSIM.

[N-M] , [N:M] , <N-M>, <N:M>

The signal|vname statement must be specified in the vector file.

Syntax

```
signal | vname    node1 <node2 ...>
```

Chapter 8: Input Netlist

Digital Vector File

Examples

```
signal n1 n2 n3 data[0-7]  
vname n1 n2 n3 data[0-7]
```

The above signal|vname statement lists 11 nodes in the vector file as shown below.

```
n1 n2 n3 data[0] data[1] data[2] data[3] data[4] data[5] data[6]  
data[7]
```

To interpret the bus signal without brackets '['and ']', the bus is represented using the notation [N~M].

```
signal n1 n2 n3 data[0~7]  
vname n1 n2 n3 data[0~7]
```

The above signal statement lists 11 nodes in the vector file in the following order:

```
n1 n2 n3 data0 data1 data2 data3 data4 data5 data6 data7
```

The bus notation of vector file format is slightly different between HSPICE and HSIM. In order to make it compatible the [HSIMHSPICEVEC on page 95](#) command is introduced. The default value is 0.

```
vname ck[1:3]  
.param HSIMHSPICEVEC=0
```

This bus is recognized as "ck[1]" "ck[2]" "ck[3]".

```
vname ck[1:3]  
.param HSIMHSPICEVEC=1
```

This bus is recognized as "ck1" "ck2" "ck3" to be compatible to HSPICE notation.

slope|rise|trise|fall|tfall**Description**

Rise and fall times can be specified for input signals by the following statements:

- slope: The slope statement starts with a keyword slope followed by a value for the rise and fall time.
- rise: The rise statement starts with the keyword rise (or trise) followed by a value for the rise time.
- fall: The fall statement starts with the keyword fall (or tfall) followed by a value for the fall time.

The slope, rise, and fall times have the following characteristics:

- If no unit is specified in the time, the default tunit is 1 nanosecond.
- The rise/fall statements overwrite the slope statement.
- If no rise/fall time is specified, the default is 1 ps.

Syntax

```
slope|rise|trise|fall|tfall time1
```

Examples

```
slope 10ps
```

The rise and fall times for the input signals are both 10 ps.

Note: A space between 10 and ps is not permissible.

```
rise 0.1
```

The rise time for the input signals is 0.1 ns, and the default fall time is 1 ps.

stop_at_error

stop_at_error is used to stop circuit simulation whenever output comparisons are performed and mismatched outputs occur.

tskip

tskip is only used when the period statement is defined in the vector file. tskip skips the vector time for each vector pattern defined in the first vertical column. The vector time for the i-th vector row is i-1 times the cycle time defined in the period statement.

Syntax

```
tskip
```

Examples

Refer to the example for period on page 298.

Chapter 8: Input Netlist

Digital Vector File

triz

triz specifies the output impedance, when the signal (for which the mask applies) is in tristate; it applies only to the input signals. triz has no effect on the output signals

Note: If you do not specify the tristate impedance of a signal in a triz statement, 1000M is assumed.

If you apply triz more than once to a signal the last statement overrules the previous statements and a warning is issued.

Syntax

```
triz <output_impedance>
```

Examples

```
triz 15.1Meg  
triz 150Meg 1 1 1 0000 00000000  
triz 50.5Meg 0 0 0 137F 00000000
```

The first triz statement sets the high impedance resistance globally, at 15.1 Mohms. The second triz statement increases the value to 150 Mohms, for vectors 1 to 3. The last triz statement increases the value to 50.5 Mohms, for vectors 4 through 7.

tunit

The tunit statement defines the unit of time in the vector file. The default value is nanosecond; written as 1.0e-9 second.

Syntax

```
tunit      time_unit1
```

Examples

```
tunit      1.0e-8  
tunit      10ns  
tunit      0.01u
```

Each tunit statement above defines the unit of time in the vector file as 10 ns.

Note: The tunit statement will affect the default time units of all parameters within the same vector file.

```
tunit 1ps  
delay 1.0
```

The amount of delay time is 1 ps because the default time unit has been changed from 1 ns to 1 ps by the tunit statement.

vchk_ignore

The vchk_ignore command causes checking to be ignored for all nodes specified when using the optional mask between times specified by t1 and t2.

Syntax

```
vchk_ignore t1 t2 <mask>
```

Parameters

t1

t1 is the start time.

t2

t2 is the end time.

mask

<mask> is an option.

If <mask> is not specified, all signals between t1 and t2 will be ignored. To ignore selected signals over the entire time period, specify the t1 start and t2 end times. This command can be repeated for cumulative effect.

Example

```
vchk_ignore 0 2 0101
```

In this example, the apply t2 to the specified mask from 0 to 2 ns.

vhth|voh|vlth|vol

Description

Note: For compatibility with vector file statements of other simulators, the following are interchangeable:

- vhth and voh
- vlth and vol

The vhth (or voh) and vlth (or vol) statements have the following characteristics:

Chapter 8: Input Netlist

Digital Vector File

- Specify the threshold voltages of logic HIGH and logic LOW states for the expected output check signals.
- The statement begins with the keyword vhth or vlth, followed by a voltage value.
- If the unit is not specified, volt is used as the default unit.
- The vhth (or voh) and vlth (or vol) can also be specified from HSIM parameters [HSIMVHTH on page 194](#) and [HSIMVLTH on page 195](#).
- If vhth and vlth are not specified in the vector file or set by the HSIMVHTH and HSIMVLTH parameters in the netlist, the following will be set:
 - vhth will be set to 70% of voltage between logichv and logiclv
 - vlth will be set to 30% of voltage between logichv and logiclv

Syntax`vhth|voh|vlth|vol val2`**Example**

```
vhth 1.8  
vlth 0.3
```

vref

Similar to delay, vref specifies the name of the reference voltage for each input vector to which the mask applies. It applies only to input signals. vref has no effect on output signals.

Note: If you do not specify the reference voltage name of the signals in a vref statement, HSIM assumes 0.

If more than one vref statement is applied, the last statement overrules the previous statements, and a warning is issued.

Syntax`vref <reference_voltage>`**Examples**

```
VNAME v1 v2 v3 v4 v5[[1:0]] v6[[2:0]] v7[[0:3]] v8 v9 v10  
VREF 0  
VREF 0 111 137F 000  
VREF vss 0 0 0 0000 111
```

When HSPICE or HSPICE RF implements these statements into the netlist, the voltage source realizes v1:

```
v1 V1 0 pwl(.....)  
as well as v2, v3, v4, v5, v6, and v7.  
However, v8 is realized by  
v8 V8 vss pwl(.....)  
v9 and v10 use a syntax similar to v8.
```

vth

Similar to tdelay, vth specifies the logic threshold voltage for each output signal to which the mask applies. The threshold voltage determines the logic state of output signals for comparison with the expected output signals. vth has no effect on the input signals.

Note: If you do not specify the threshold voltage of the signals in a vth statement, HSIM assumes 1.65.

If you apply more than one VTH statement to a signal, the last statement overrules the previous statements, and a warning is issued.

Syntax

```
VTH <logic-threshold_voltage>
```

Example

```
VTH 1.75  
VTH 2.5 1 1 1 137F 00000000  
VTH 1.75 0 0 0 0000 11111111
```

In the above example, the first VTH statement sets the logic threshold voltage at 1.75V. The next line changes that threshold to 2.5V, for the first 7 vectors. The last line changes that threshold to 1.75V, for the last 8 vectors.

All of these examples apply the same vector pattern and both output and input control statements. Therefore, the vectors are all bidirectional.

Dynamic vhi and vlo for Logic HIGH and Logic LOW States

The vhi and vlo vector file statements define threshold voltages for the logic HIGH and logic LOW states of the expected output check signals. This HIGH and LOW logic values are static or fixed state voltages.

VCD Direct-Read Feature

A VCD file is an open industry standard ASCII format file, generated by a test pattern generator, RTL simulator, or other high-level simulator. A VCD file does not contain signal direction or waveform characteristics. Therefore, you are required to create a signal information file to map the signal names in the VCD file to the node names in the design netlist, and also to provide this signal information for each signal to HSIM.

A VCD file contains three sections:

- The *header information* section describes the date, the version number of the simulator, and the time-scale used.
- The *variable definitions* section contains the scope of the hierarchy, and type of variables. A variable can be a scalar or a bus. Each variable is represented by a unique identifier character.
- The *value changes* section contains the actual value changes for all variables specified at each simulation time increment. Only the variables, which change during a time increment, are listed. Each variable with a set of values forms a vector over time.

To read-in a VCD file to HSIM, use the [HSIMVCD2VEC on page 191](#) command.

Using the Signal Information File

The signal information file provides information to map the variable names in the VCD file to the node names in the design netlist. All signal names in the signal information file are case-sensitive because Verilog variable names are case-sensitive. If multiple expressions match the same signal, the last match takes priority. To avoid mapping problems, always use unique matches.

The following procedures enable you to create the signal information file:

1. Define bus syntax in the design netlist.
See the [Defining Bus Syntax with the #format Command](#) section.
2. Define signal directions, such as input, output, or bi-directional.
See the [Defining Signal Directions](#) section.
3. Map the variable names in the VCD file to the node names in the design netlist.
See the [Defining Mapping Information with the #alias Command](#) section.
4. Define analog waveform characteristics, such as *rise time* and *fall time*.
See the [Defining Attributes for Signals](#) section.

Defining Bus Syntax with the #format Command

The `#format` command specifies the bus naming format. The default name format, if the `#format` command is not specified, is `% [#]`. For a character descriptions, see [Table 49 on page 307](#).

Table 49 % and # character descriptions

Character	Description
<code>%</code>	Represents the variable names
<code>#</code>	Represents the bus indexes

See the following syntax:

```
#format | #fo  bus_naming_format
```

Only one `#format` command is allowed in a signal information file. Create another signal information file, if multiple formats are needed.

[Example 21](#) displays the bus line format in a VCD file.

Example 21 Bus line format command example

```
$var reg 3 k tA [2:0] $end
```

The `#format` command causes `tA` to expand to the following names in the VCD file.

For a description of the commands and corresponding mapping names, see [Table 50](#).

Table 50 #format command mapping names

Command	Mapping name
#format %<#>	tA<2> tA<1> tA<0>
#format %.#	tA.2 tA.1 tA.0
#format %[#]	TA[2] tA[1] tA[0]

Defining Signal Directions

You must define signal directions to the variable names in the VCD and EVCD files. Signal directions can be input signals, output signals, or bi-directional signals. HSIM uses all input, output, and bi-directional signals specified in the VCD file for the simulation. HSIM uses input signals as stimulus for the simulation. Simulation output signals are checked against simulation output signals for any mismatch. Bidirectional signals require enable signal(s) to control the direction, either as input signals or output signals. All other signals in the VCD file, which are not specified in the signal information file, are ignored.

NOTE: Bi-directional signals cannot be included within an `enable_signal` logic expression.

The `#bi-directional enable_signal` option (see [Example 22](#)) can be specified as a logic expression, as shown in [Example 23](#).

Example 22 Bi-directional enable_signal

```
#input | #in signal_name(s)
#output | #out signal_name(s)
#bidirectional | #bi signal_name(s) [in | out enable_signal(s)]
```

Example 23 Logic expression of bi-directional enable_signal

```
#format %[#]
#in cen oen
#in a[[3:0]]
#out c b*
#bi io[[7:0]] (out cen & oen)
```

[Example 23](#) declares the following (description of each line):

- bus format
- input signals `cen` and `oen`
- input bus signal `a [[3 : 0]]`
The outer bracket signifies bus notification, while the inner bracket signifies a wild card range.
- output signal `c` and bus `b`
The asterisk character (*) is a wild card for any group of characters.
- bi-directional signal bus `io [[7 : 0]]` is controlled by signals `cen` and `oen`
- bus `io [[7 : 0]]` is output when both `oen` and `cen` are logic-high; otherwise bus `io` is input

Logic expression operators are supported to define enable signals. See [Table 51](#) for definitions of the logic expression operators.

Table 51 Logic expression operators

Operator	Definition
<code>~</code>	NOT
<code>!</code>	NOT
<code>^</code>	exclusive OR
<code>&</code>	AND
<code>&&</code>	AND
<code> </code>	inclusive OR
<code> </code>	inclusive OR

Important: You must insert a space between a logical operator and the signal name. If there is no space, the signal name will not be recognized. For example:

```
#bi bi_driver (in ~ ctrl)
```

Note the space between the NOT operator (~) and the signal name (ctrl).

Chapter 8: Input Netlist
VCD Direct-Read Feature

You can use wild cards to match node names, as described in [Table 52](#).

Table 52 Wild card support

Wildcard	Description
*	Matches any group of characters
?	Matches any single character
[:]	Matches any range expression

You need to provide the full hierarchical name for the specified signal in the signal information file, based on the VCD file hierarchy. The period (.) character is used as the hierarchical delimiter in the signal information file. If you choose not to specify the full hierarchical name, see [The #scope Command](#) section. See the [Signal Information File Sample](#) section for a detailed sample.

HSIM uses input signals as stimulus for the simulation. Each input signal is converted to a PWL voltage source, in series with a resistor. The rise and fall times are used to smoothly transition from one state to another.

Each output signal is converted to an expected output checking statement. During simulation, the actual voltages are converted to a digital value and compared to the expected output checking statement at the specified times. A warning is generated, if the states differ. HSIM checks the output signals to the expected outputs at every time point specified in the VCD file.

See [Example 24](#) for further details.

Example 24 Signal direction example

```
...
$var reg 4 " A [3:0] $end
$var reg 4 % B [3:0] $end
...
$dumpvars
b0000 "
b0000 %
$end
#10
b0010 "
#12
b0001 %
```

Bus A is defined as input and bus B is defined as output in the signal information file. At time 0, bus A is initialized to 0000; at time 10, it changes state from 0000 to 0010; at time 12, there is no change in the state. Since bus B is specified as output, the simulation output of bus B is checked against the expected result specified in the VCD file. HSIM performs vector-checking at every time point (0, 10, 12), even if bus B does not change state at time 10.

A delay usually occurs between the logic simulation and the transistor-level simulation. Therefore, HSIM issues warning messages if the state in the actual simulation does not match the VCD expected output. You can use the #odelay command to delay the expected outputs in the VCD file, so accurate vector-checking is implemented during simulation without generating warning messages. Refer to the #odelay command in the section [The #idelay and #odelay Commands](#) for further details.

Defining Mapping Information with the #alias Command

The #alias command maps the signal names in the VCD file into different names to match with the design netlist. You can use multiple aliases and regular expressions, but use only one specific alias for each signal name. If multiple patterns match the same name, the last pattern takes priority.

The following are situations in which you must use the #alias command:

- Different names are detected in the VCD file and the design netlist.
For example, you want to map signal name A1 in the VCD file to node name B1 in the design netlist.
- Different hierarchy in the VCD file and the design netlist.
For example, you want to map the signal from top module *top.A1* to the node name of the top hierarchy of the design netlist A1.

See the following syntax:

```
#alias | #al vcd_signal_name netlist_node_name
```

In [Example 25](#), the #alias commands convert the following signal names in the VCD file to map with the signal names in the design netlist. The percent sign (%) represents the signal name, and the pound sign (#) represents the bus index. See [Table 49](#).

Example 25 #alias command example

```
#alias tX64 x_64      ! alias #1
#alias t% %           ! alias #2
#alias tAP% Ap_%     ! alias #3
#alias tAPE% APE%    ! alias #4
```

Chapter 8: Input Netlist
VCD Direct-Read Feature

See [Table 53](#) for the mapping description.

Table 53 #alias command converting signal names

From (VCD file)	To (design netlist)	#alias matched
TAPEND	APEND	4
tL2CIB	L2CIB	2
tX64	x_64	1
TAPEB	APEB	4
TGBLB	GBLB	2
TAP0	Ap_0	3
TADD	ADD	2
TAPDD1	Ap_DD1	3

Defining Attributes for Signals

The signal information file supports commands that define various attributes for signals, such as *rise time* or *fall time*. Multiple specifications of each individual command is allowed in the signal information file. If more than one individual command is specified to a signal, the last command overrides the previous one. If the signal name is not specified, the value applies to all input or output signals.

See the following supported commands:

- [The #edge_shift Command](#)
- [The #idelay and #odelay Commands](#)
- [The #outz Command](#)
- [The #scale Command](#)
- [The #trise, #tfall, and #slope Commands](#)
- [The #triz Command](#)
- [The #vih and #vil Commands](#)

- The #voh and #vol Commands
- The #scope Command

The #edge_shift Command

The #edge_shift command specifies the timing shift on the transitions for specified signals. The default values for all #edge_shift arguments are 0.0. See the following syntax, and descriptions of the command arguments:

```
#edge_shift time1 time2 time3 signal_name(s)
```

Table 54 #edge_shift command argument descriptions

Argument	Description
<i>time1</i>	Specifies the time shift to be applied to positive transitions (to 1 or H)
<i>time2</i>	Specifies the time shift to be applied to negative transitions (to 0 or L)
<i>time3</i>	Specifies the time shift of all other transitions

The #idelay and #odelay Commands

The #idelay and #odelay commands specify the delay time of the specified input and output signals. The default values for #idelay and #odelay are 0.

See the following syntax:

```
#idelay value signal_name(s)
#odelay value signal_name(s)
```

The #outz Command

The #outz command specifies the drive resistance for specified input signals—for instance, the “H” and “L” state characters. State characters 1 and 0 are considered to be strong logic and have infinite strength. The default value for #outz is 0.0.

See the following syntax:

```
#outz value signal_name(s)
```

The #scale Command

The #scale command specifies the time multiplier—a global setting that applies to all signals. This command does not specify individual signals. The

Chapter 8: Input Netlist
VCD Direct-Read Feature

default value for #scale is 1.0. The #scale command applies to all time variables, such as #rise and #fall.

See the following syntax:

```
#scale option
```

The #trise, #tfall, and #slope Commands

The #trise and #tfall commands specify the *rise time* and *fall time* for specified signals, respectively. The #slope command applies to both *rise time* and *fall time*. The default values for #trise, #tfall, and #slope are 0.0.

See the following syntax:

```
#trise value signal_name(s)  
#tfall value signal_name(s)  
#slope value signal_name(s)
```

The first example in [Example 26](#) assigns 10ps as the fall time for all the signals.

If multiple patterns match the same signal name, the last pattern takes priority. The second example in [Example 26](#) assigns 10ps as the fall time for all signals, except c and bus io.

Example 26 #tfall command example

```
#tfall 10ps  
#tfall 10ps *  
#tfall 1ps c io*
```

The #triz Command

The #triz command specifies the output impedance for specified tri-state input signals. The default value for #triz is 1000Meg.

See the following syntax:

```
#triz value signal_name(s)
```

The #vih and #vil Commands

The #vih and #vil commands specify the logic-high and logic-low input voltages of specified signals. By default, HSIM detects the rail-to-rail voltage supply of the design and sets #vih to the maximum rail voltage, and #vil to the minimum rail voltage.

See the following syntax:

```
#vih value signal_name(s)  
#vil value signal_name(s)
```

Example 27 assigns 1.1V as logic-high, and 0.2V as logic-low for all input signals.

Example 27 #vih and #vil command examples

```
#vih 1.1
#vil 0.2
```

Example 28 overrides the logic-high of bus add as 3.3V, and logic-low of reset as 0.0V. You could override the global values of logic-high and logic-low for specific input signals.

Example 28 #vih and #vil command examples

```
#vih 3.3 add*
#vil 0.0 reset
```

The #voh and #vol Commands

The #voh and #vol commands specify the output voltage logic-high and logic-low of specified output signals. The default value of #voh is set to vih; #vol is set to vil.

See the following syntax:

```
#voh  value  signal_name(s)
#vol  value  signal_name(s)
```

Example 29 assigns 3.3V and 0.0V as the global logic-high and logic-low, respectively, for all output signals except bus out. The logic-high of bus out is overridden as 1.1V.

Example 29 #voh and #vol command examples

```
#voh 3.3
#vol 0.0
#voh 1.1 out*
```

The #scope Command

The #scope command specifies the name of the scope from which signals are used for matching. By default, if #scope is not specified, HSIM processes the signals starting from the top level in the VCD file. Note that the signals must be selected using #in, #bi or #out to be used as stimuli. You must specify the corresponding scope, so that the correct signals are used for simulation. If #scope is not specified, HSIM searches the signals starting from the top-level of the VCD file. This eliminates the use of the #alias command to map the signals in the specified scope to the top-level hierarchy in the design netlist.

Only one #scope command is allowed in a signal information file. Create another signal information file if multiple scopes are needed.

Chapter 8: Input Netlist
VCD Direct-Read Feature

Use the following syntax:

```
#scope scope_name
```

The scope definition in the VCD file is shown in [Example 30](#).

Example 30 Scope definition in a VCD file

```
$var wire 1 A data_in $end
$var wire 32 B data_out [31:0] $end

$scope module st $end
$var wire 1 `data_in $end
$var wire 32 ^data_out [31:0] $end
$upscope $end

$scope module tpu $end
$var wire 1 _data_in $end
$var wire 32 Y data_out [31:0] $end
$upscope $end
```

[Example 31](#) uses st as the top-level hierarchy in the VCD file, and searches the signals only in scope st.

Example 31 #scope st command example

```
#scope st
```

The #vchk_ignore Command

The #vchk_ignore command disables output checking during the specified time interval for the specified signals.

Use the following syntax:

```
#vck tstart tstop [output_signals]
```

tstart

Start of the time interval to be ignored.

tstop

End of the time interval to be ignored.

output_signals

Specifies the output signals to ignore. By default, all output signals are ignored during the specified time interval.

VCD File Sample

See [Example 32](#) for a sample VCD file:

Example 32 VCD file sample

```
$date
    Sun Oct 10 18:20:30
$end
$version
    Synopsys VCS Version 7.0
$end
$timescale
    1ns
$end

$scope module adder $end
$var reg      1 ! CIN $end
$var reg      4 " A [3:0] $end
$var reg      4 # B [3:0] $end
$var wire     1 $ COUT $end
$var wire     4 % S [3:0] $end
$upscope $end

$enddefinitions $end
$dumpvars
0!
b0000 "
b0000 #
0$
b0000 %
$end
#10
b0000 #
b0000 "
0!
#12
b0000 %
0$
#20
b0001 #
#22
b0001 %
#30
b0010 #
#32
b0010 %
#40
```

Chapter 8: Input Netlist
VCD Direct-Read Feature

```
b0011 #
#42
b0011 %
#50
b0100 #
#52
b0100 %
#60
b0101 #
#62
b0101 %
#70
b0110 #
#72
b0110 %
#80
b0111 #
#82
b0111 %
#90
b1000 #
#92
b1000 %
#100
b1001 #
```

Signal Information File Sample

See [Example 33](#) for a sample signal information file:

Example 33 Signal information file sample

```
; Define bus syntax
#format      %[#]

; Define signal directions
#in          adder.CIN
#in          adder.A[[3:0]]
#in          adder.B[[3:0]]
#out         adder.COUT
#out         adder.S[[3:0]]

; Define mapping information
#alias       adder.CIN      CIN
#alias       adder.A[3]      A[3]
#alias       adder.A[2]      A[2]
#alias       adder.A[1]      A[1]
#alias       adder.A[0]      A[0]
#alias       adder.B[3]      B[3]
#alias       adder.B[2]      B[2]
#alias       adder.B[1]      B[1]
#alias       adder.B[0]      B[0]
#alias       adder.COUT     COUT
#alias       adder.S[3]      S[3]
#alias       adder.S[2]      S[2]
#alias       adder.S[1]      S[1]
#alias       adder.S[0]      S[0]

; Define analog waveform characteristics
#vih        3.3
#vil        0.0
#voh        3.0
#vol        0.3
#slope     10ps
```

Chapter 8: Input Netlist

Vector Data

See [Example 34](#) for a sample signal information file generated with the #scope command:

Example 34 Signal information file sample generated with the #scope command

```
; Define bus syntax
#format      %[#]

; Define scope
#scope       adder

; Define signal directions
#input       CIN
#input       A*
#input       B*
#output      COUT
#output      S*

; Define analog waveform characteristics
#vih        3.3
#vil        0.0
#voh        3.0
#vol        0.3
#slope      10ps
```

Vector Data

Vector data are defined in tabular format. The first vertical column is time. The values in the time column must be in increasing order. If no unit is specified in the value, the default unit of nanosecond will be used. If the period is specified in the vector definition section, it is not necessary to specify the time at each row of vector data section.

The time at the i-th row is implicitly defined to be i-1 times the period. Each subsequent vertical column defines the logic pattern associated with the corresponding signal node(s) defined in signal statement. The pound sign (#) at the first vertical column indicates that relative time is used.

Each input data bit can be one of the following valid states.

- 0: Drive to ZERO
- 1: Drive to ONE

- z: High Impedance
- x: Drive to ZERO

The expected output bit can be one of the following states:

- 0: Expect ZERO
- 1: Expect ONE
- x: Don't Care
- u: Don't Care
- h: Expect HiZ-H state
- l: Expect HiZ-L state
- z : Expect either HiZ-H or HiZ-L state

The [HSIMHZ on page 95](#) command controls HSIM treatment of output bits. When HSIMHZ=0 (default), the following bits are treated identically: h, l, x, and u.

When HSIMHZ=1, each output bit has a unique definition. Do not use h with 1, or l with 0. Do not use x or u with z.

When HSIMHZ= -1, HSIM treats h and 1 identically, and l and 0 identically.

Chapter 8: Input Netlist

Vector Data

Example

```
signal CK1M IOWORDN ROMN RDN \
PA<15-12> PA<7-4> \
PA<11-8> PA<3-0>

radix 1111 4444
io iiii iiii
0 1001 fffe
202 1001 0000
246 0001 0000
250 0000 00a0
300 1000 0000
346 0000 z300
396 0001 z000
400 1001 z000
402 1001 z002
446 0001 z002
signal CK1M IOWORDN ROMN RDN \
PA<15-12> PA<7-4> \
PA<11-8> PA<3-0>
period 50
radix 1111 4444
io iiii iiii
1001 fffe
1001 0000
0001 0000
0000 00a0
1000 0000
0000 z300
0001 z000
1001 z100
signal CK1M IOWORDN ROMN RDN \
PA<15-12> PA<7-4> \
PA<11-8> PA<3-0>

radix 1111 4444
io iiii iiii
0 1001 fffe
#10 1001 0000
#4 0001 0000
#20 0000 00a0
50 1000 0000
#5 0000 z300
```

The patterns are applied at the times indicated below:

- The first pattern is applied during 0 – 10 ns
- The second pattern during 10 ns – 14 ns

- Third pattern during 14 ns – 34 ns
- Fourth pattern during 34 ns – 50 ns
- Fifth pattern during 50 ns – 55 ns
- Sixth pattern starts at 55 ns

Built-in Functions

The HSIM built-in functions and their descriptions are described in the table below.

Table 55 Built-in Functions

Function	Description
$\sin(x)$	Returns sine of x.
$\cos(x)$	Returns cosine of x.
$\tan(x)$	Returns tangent of x.
$\text{asin}(x)$	Returns arc sine of x.
$\text{acos}(x)$	Returns arc cosine of x.
$\text{atan}(x)$	Returns arc tangent of x.
$\text{atan2}(x,y)$	Returns the arc tangent of x/y using the signs of x and y to determine the quadrant of the return value.
$\text{sinh}(x)$	Returns hyperbolic sine of x.
$\cosh(x)$	Returns hyperbolic cosine of x.
$\tanh(x)$	Returns hyperbolic tangent of x.
$\text{abs}(x)$	Returns the absolute value of x.
$\text{sqrt}(x)$	Returns square root of x.
$\text{pow}(x,y)$	Returns the value of x raised to the integer part of y: $x(\text{integer part of } y)$.

Table 55 Built-in Functions (Continued)

Function	Description
pwr(x,y)	Returns pow(abs(x),y) with the sign of x.
log(x)	Returns nature log of abs(x) with the sign of x.
ln(x)	Same as log(x).
log10(x)	Returns base 10 log of abs(x) with the sign of x.
exp(x)	Returns the e to power of x.
db(x)	Returns $20.0 * \log_{10}(\text{abs}(x))$ with the sign of x.
floor(x)	Returns the largest integral value which is not grater than x.
ceil(x)	Returns the smallest integral value which is not smaller than x.
int(x)	Returns ceil(x) if x is less than 0, otherwise returns floor(x).
trunc(x)	Same as int(x).
sgn(x)	Returns 0 if x is 0, otherwise returns the sign of x.
sign(x)	Same as sgn(x).
sign(x,y)	Returns sign(y) * abs(x).
min(x,y)	Returns the minimum of x and y.
dmin(x,y)	Same as min(x,y).
max(x,y)	Returns the maximum of x and y.
dmax(x,y)	Same as max(x,y).

Simulation and Control Statements

This sections lists the simulation and control statements:

- .alter
- FF
- .del param
- .end
- .endl
- .ends
- .force
- .global
- .ic
- .include
- .lib
- .malias
- .nodeset
- .op
- .option
- .param
- .release
- .subckt
- .temp
- .tran

.alter

Description

The .alter statement repeats a simulation using alternative parameter values and modified netlist. In each simulation run, the output files, such as hsim.ic, hsim.fsdb, and so on, will have the suffix .a#, where # is the run number.

Syntax

```
.alter
```

Example

```
* first simulation run
.temp 25
.....
.alter
.temp 100
.end
```

The circuit temperatures for the two runs are: Run 1; : 25° C and Run 2; 100° C.

FF**Description**

The .data statement allows modifying parameter values in transient simulation. This command is for cell or block characterization and optimization. Refer to [Chapter 17, Timing and Power Analysis, Bisection Optimization on page 523](#). The group of parameter values is included in the input file.

Syntax

```
.data dlink + param1 <param2 param3 ... paramN> + val1a
    <val2a val3a ... valNa> + val1b <val2b val3b ... valNb>
    +
.enddata
```

Parameters

dlink

dlink is also used in the .tran command

param

Parameter names

Note: Parameter names must begin with an alphabetic character, but thereafter can contain numbers and some special characters.

val

Parameter values.

Examples

```
.tran 0.1n 100n sweep data=dlink1
.data dlink1
+ L      W      CLOAD
+ 0.18u   0.36u    10f
+ 0.13u   0.26u    5f
.enddata
```

HSIM accepts the following parameter values and performs the first transient simulation:

L=0.18u, W=0.36u, CLOAD=10f

Afterwards, the simulation is repeated for the following:

L=0.13u, W=0.26u, CLOAD=5f

.del param

Description

The .del param statement removes selective parameter setting which is especially useful in the .alter section.

Syntax

```
.del param <param1> <param2> ... <subckt=<subckt_name>>
```

For instance, a vector file can be removed from the simulation in the [.alter on page 325](#).

Example

```
. param hsimvectorfile=vec1
.....
.....
.alter
.del param hsimvectorfile
.param hsimvectorfile=vec2
.....
```

In this case, the vector file vec1 is removed in the .alter section and another file, vec2, is added to the simulation. If the .del param statement is not used, then both vec1 and vec2 files are included in the simulation in the [.alter on page 325](#).

.end**Description**

The .end statement defines the end of an HSIM run.

Syntax

```
.end <comments>
```

If .end is not specified in an input netlist, the default is end of file. An input netlist can contain multiple HSIM runs by having an .end at the end of each run. In each HSIM run, the output files, such as hsim.ic, hsim.fsdb, and so on, will have the suffix .e#, where # is the run number.

Example

See the example for [.alter on page 325](#).

.endl

The .endl statement ends the library macro definition.

Syntax

```
.endl <lib_entry_name>
```

Example

See the example for [.lib on page 331](#).

.ends

The .ends statement specifies the end of a subckt definition.

Syntax

```
.ends <subckt_name>
```

Example

See the example for [.subckt on page 338](#).

.force**Syntax**

```
.force node_name voltage_value <subckt=subckt_name>  
      <time=force_time>
```

Parameters

node_name

node_name can be a specific node name or a pattern.

subckt_name

subckt_name is the subcircuit name. When the node_name subcircuit parameter is used, the node is inside that subcircuit. Otherwise, node_name is assumed to be a hierarchical name.

force_time

force_time is the starting time that forces the specified node to stay at a specified constant voltage_value. Time unit is seconds.

Description

.force forces node_name to stay at the same voltage_value as force_time until one of the following occurs:

- Simulation ends
- When the constant node voltage status is released by either of the following:
 - .release
 - rv.

.force does not work on voltage source or vector file input nodes. Refer to the following sections for additional information:

- [fv on page 482](#)
- [rv on page 498](#)
- [.force on page 329](#)

Example

```
.force pump 6.5 time=100u
```

.global

The .global statement defines global nodes.

Syntax

```
.global node1 <node2 ...>
```

A global node can be directly referenced from any level of hierarchy. Any node name appearing at the top-level or any low level hierarchy is connected to the same node. Declaring a global node allows a direct reference to an internal subcircuit node without defining the node in the subcircuit port list. HSIM recognizes any of the following as the ground node:

- 0 (zero)
 - gnd
 - gnd!
 - ground
-

.ic

The .ic statement defines the initial condition.

Syntax

```
.ic v(node1)=val2 <v(node2)=val3 ...> <subckt=sub_name>  
      <level=val4>
```

The specified node can be the node name of a single node or a pattern containing asterisk (*) wildcard character that represents a group of nodes matching the pattern. The optional setting of level is specified to control the scope of wildcard match.

Parameters

The .IC statement parameters are described in the table below.

Table 56 .IC Statement Parameters

Parameter	Default	Description
subckt=sub_name		Initial condition is set to the specified node name(s) within all instances of the specified subcircuit name. This subckt setting is equivalent to placing the .ic statement within the subcircuit definition.

Table 56 .IC Statement Parameters

Parameter	Default	Description
level=val4	-1	<p>This setting is effective only when the asterisk (*) wildcard character is specified in the output variable. The level value val4 specifies the number of hierarchical depth levels when matching the wildcard node/element name.</p> <ul style="list-style-type: none"> ▪ When val4 is set to 1, the wildcard match applies to the same depth level where the .ic statement is located. ▪ When val4 is set to 2, it applies to same level and one level below the current level where .ic is located. ▪ When val4 is set to 0, no wildcard name is matched so that any output variable containing wildcard character is ignored. ▪ When val4 is set to -1, the wildcard match applies to all the depth levels below and including the current level of .ic statement.

Example

```
.ic v(1)=1.8 v(4)=3.3 v(x1.x2.*)=2.2
```

.include

The .include statement includes another data file.

Syntax

```
.include file_name
```

or

```
.inc file_name
```

Example

```
.inc netlist.net
```

.lib

The .lib statement includes the model library.

Syntax

```
.lib file_name library_name
```

This is a library call statement which indicates the specified library entry in the library file should be read in.

You can also use:

```
.lib library_name
```

This is a library definition statement that begins the library entry in the library file. The .endl indicates the end of the library entry. Refer to [.endl on page 328](#).

Parameters

The .lib statement parameters are described in the table below.

Table 57 .LIB Statement Parameters

Parameter	Description
file_name	The filename of the library.
library_name	The library entry name.

Examples

```
.lib models.lib TT
```

The TT portion of models.lib is read in.

```
.lib TT
.model ck1 nmos level=49
...
.endl TT
```

A library is defined.

.malias

The .malias statement provides an alias for a model name.

Syntax

```
.malias model_name alias_name1 <alias_name2 ...>
```

The word alias_name1 is aliased to model_name.

Example

```
.malias tn013 ma mb
```

Both model names ma and mb are aliased to tn013.

.nodeset

This statement sets the starting voltage at DC initialization.

Syntax

```
.nodeset v(node1)=val1 <v(node2)=val3 ...>  
<subckt=sub_name> <level=val4>
```

Description

The specified node can be the node name of a single node or a pattern containing a asterisk (*) wildcard character that represents a group of nodes matching the pattern. The optional setting of level controls the scope of wildcard match. Refer to [.ic on page 330](#) for details.

The optional setting of subckt is for the nodes within all instances of the specified subcircuit name; this is equivalent to placing the .nodeset statement inside the subcircuit definition.

Example

```
.nodeset v(1)=1.8 v(4)=3.3 v(x1.x2.*)=2.2
```

.op

The .op statement dumps the operating condition at the specified time(s).

Syntax

```
.op <time1> <time2 ...>
```

Description

The DC operating point output is partially supported in HSIM. HSIM only creates the node voltages. The DC operating analysis should generate node voltages and element currents at the resulting DC operating point.

When .op statement is specified in the netlist, HSIM prints each node voltage at the specified time into a file. The format of the output file can be changed by setting [HSIMOPCOMPRESS on page 118](#).

.option

Defines optional values.

Syntax

```
.option name1=val1 <name2=val2> . . . <nameN=valN>
```

Description

HSIM recognizes popular simulation options described in the table below..

Table 58 SPICE Simulation Options Recognized by HSIM

aspec	defps	wl
badchr	genk	scale
defad	gmin	scalm
defas	gmindc	search
defl	klim	tnom
defw	nowarn	
defnrd	parhier	
defnrs	spice	
defpd	warnlimit	

All other SPICE options are ignored.

Note: [Chapter 12, Simulation Output](#) describes output statements that are supported in HSIM.

Parameters

The .option parameters are described in the table below.

Table 59 .option Parameters

Parameter	Default	Description
aspec		Makes the simulation compatible with aspec setting, such as when the wl option (see below) is activated. scale and scalm are set to scale geometry dimension to microns.
badchr		Produces a warning if n unprintable characters are detected in the input file.
defad	0	The default value for drain-bulk junction diode area in a MOSFET.
defas	0	The default value for source-bulk junction diode area in a MOSFET.
defl	0	The default value for channel length in a MOSFET.
defnrd	0	The default value for the number of squares for MOSFET drain resistor.
defnrs	0	The default value for the number of squares for MOSFET source resistor.
defpd	0	The default value for drain bulk junction diode perimeter in a MOSFET.
defps	0	The default value for source bulk junction diode perimeter in a MOSFET.
defw	0	default value for channel width of a MOSFET.
genk	1	When assigned the value of 1 (one), genk is the control parameter for the automatic calculation of second-order mutual inductance for multiple coupled inductors.
gmin	1e-12	Specifies the minimum conductance added to all PN junctions for a time sweep in transient analysis.

Table 59 .option Parameters (Continued)

Parameter	Default	Description
gmindc		Specifies conductance in parallel for PN junctions and MOSFET nodes in DC analysis.
klim	0.01	Minimum mutual inductance threshold for calculation of second-order mutual inductance.
nowarn		Suppress warning messages.
Parhier=global		A parameter name specified at a higher hierarchical level which prevails over the same parameter name specified at a lower level.
Parhier=local	global	A parameter name specified inside a subcircuit which prevails over the same parameter name specified at a higher hierarchical level.
scale	1.0	Element scale parameter. It scales the geometry of the transistor instance.
scalm	1.0	Model scaling factor. Relevant model parameters are scaled by this value.
search=dir_path		Sets the search path for model libraries and included files. The HSIM searches the specified directory for libraries used in the simulation.
spice		Makes the simulation compatible with SPICE setting, such as:
		tnom=27
		defnrd=1
		defnrs=1
tnom	25 or 27	Simulation reference temperature. The default value is 25 degrees C unless .option SPICE is specified; then the default is 27° C.

Table 59 .option Parameters (Continued)

Parameter	Default	Description
warnlimit=y		Limits the number of warnings to appear in the .log file. Value y is the total number of warnings allowed for each warning type.
wl		MOSFET element geometry order is changed from (defaulted) length-width to width-length.

Example

```
.option scale=1e-6
```

.param

The .param statement defines parameters.

Note: Parameter names must begin with an alphabetic character, but thereafter can contain numbers and some special characters.

Syntax

```
.param name1=val1 <name2=val2 ...>
```

Example

```
.param a=2 b=4
.param c='b+3*a'
```

.release

Syntax

```
.release node_name <subckt=subckt_name> <time=release_time>
```

Parameters

node_name

node_name can be a specific node name or a pattern.

subckt_name

subckt_name is the subcircuit name. When the node_name subcircuit parameter is used, it is the node inside that subcircuit. Otherwise, node_name is assumed to be a hierarchical name.

release_time

release_time is the starting time that releases the specified node. Time unit is seconds.

Description

.release releases the node voltage from the value fixed by the .force command or by the interactive rv command. The node voltage are then determined by the regular simulation result. Refer to the following sections for additional information:

- [fv on page 482](#)
 - [rv on page 498](#)
 - [.force on page 329](#)
-

.subckt

The .subckt statement defines a subcircuit.

Syntax

```
.subckt sub_name term1 <term2 ...> <parameter1=val1>
      <parameter2=val2 ... >
```

Example

```
.subckt inverter 1 2
M1 2 1    0    0    nmos1 w=0.36u l=0.18u
M2 2 1    vdd vdd pmos1 w=0.36u l=0.18u
.ends
```

The subcircuit instance is defined below:

Syntax

```
Xaa term1 <term2 ...> sub_name <parameter1=val2>
      <parameter2=val3>
```

Example

```
Xinv1 1    2    inverter
```

.temp

The .temp statement defines the temperature.

Syntax

```
.temp temp1 <temp2 ...>
```

Description

The temperature value is in degrees Celsius. If multiple temperatures are specified, HSIM will perform one simulation for each temperature with the output files (i.e. hsim.ic, hsim.fsdb, etc.) having a .t# suffix. The default temperature is 25 degrees Celsius.

Example

```
.temp 85
```

.tran

The .tran statement defines transient analysis.

Syntax

Three versions of sweep syntax are supported.

```
.tran steptime stoptime <uic> <sweep data=data_name>
.tran steptime stoptime <uic> sweep var pstart pstop incr
.tran steptime stoptime <uic> sweep var poi np p1 p2 ...
```

Description

Except for uic and sweep data, all optional parameters specified after stoptime are ignored.

For more information about sweep data, refer to the example in [FF on page 326](#).

Parameters

The .tran statement parameters are described in the table below.

Table 60 .TRAN Parameters

Parameter	Description
steptime	A placeholder for compatibility with the SPICE syntax—this parameter does not effect HSIM simulation.

Chapter 8: Input Netlist
Simulation and Control Statements*Table 60 .TRAN Parameters*

Parameter	Description
uic	Disables DC initialization and uses the IC conditions as DC solution. If a node does not have an I condition, its DC solution will be 0.
var	Parameter name or keyword temp (for temperature sweep)
pstart	Starting value.
pstop	Final value.
incr	Increment value.
np	Number of points.
p1, p2	Values for the parameter sweep.
poi	Indicates that type of sweep is a list of points.
sweep	Indicates sweep operation according to the parameter values specified in data_name. Refer to the example in FF on page 326 .

Example

```
.tran 1N 100N
```

Simulation Commands

Describes how to specify the HSIM simulation commands and how they affect performance and accuracy.

- Specifying HSIM Commands
- Control Commands for Ungrounded Capacitors
- Control Commands for Grounded Capacitors
- Control Commands for Floating Capacitors
- Control Commands for Isomorphic Matching
- DC Initialization
- Implicit Backward Euler Algorithm and Trapezoidal Integration Algorithm
- Simulation Control Commands
- Piecewise Constant Setting
- Examples of HSIM Simulation Control Commands

Specifying HSIM Commands

There are two methods for specifying HSIM control commands:

- In the top-level netlist file.
- In an external file included in the main netlist.

When commands are specified inside the netlist file, some HSIM commands can be flexibly placed within either of the following:

- Subcircuit
- Top-level netlist

Chapter 9: Simulation Commands

Specifying HSIM Commands

This section covers the following topics:

- [Specifying HSIM Commands in the Top-Level Netlist](#)
 - [Specifying Commands in an External File](#)
 - [Aliasing Commands to User-Defined Names](#)
-

Specifying HSIM Commands in the Top-Level Netlist

Specifying HSIM commands within the netlist file has the advantage that their influence can be localized to the module in which the command is defined. Also, the netlist can still be simulated by other SPICE simulators. The HSIM commands are ignored.

An HSIM command specified at the top-level netlist is a global command which affects the entire circuit. When specified with a different value within a subcircuit, the HSIM command is a local command and the value only applies at the local level within the subcircuit. Commands defined in a subcircuit alone only affect values within the range of the subcircuit.

Start each HSIM control command with the keyword HSIM, shown in the syntax sample below, to identify the commands and avoid interference with other commands supported in SPICE.

```
hsimparameter_name=value1
```

The modified netlist containing the HSIM command setting with a real value operates correctly when performing a standard SPICE simulation using the same netlist. However, using the HSIM command setting with string value or PWC function can not be simulated in a standard SPICE simulation. These commands can be entered in `hsim.ini` to avoid this problem. There are the following different syntax types used to set the HSIM command:

1. A single value HSIM command is input as follows:

```
.param hsimparameter_name=value1  
<hsimparameter_name2=value2>
```

2. A PWC function controlled by simulation time command is input as follows:

```
.param hsimparameter_name3=pwc (t1, val1, t2, val2, ..., tN,  
valN)
```

Syntax Definitions

t1, t2, ... tN

Simulation time.

val1, ... valN

Setting different parameter values corresponding to t1, t2, ... tN represent different simulation times.

A PWC function controlled by a scheduled event is input as follows:

```
.param hsimparameter_name4=pwc (label1, val1, label2, val2,
..., labelN, valN)
.schev labelN var=variable val=threshold_val edge=<r|f|c>
<from=time1 to=time2>
```

labelN

Label name of scheduled event command.

variable

Node name (v(node_name))or branch current (i(elem_name)).

r

Rise

f

Fall

c

Cross

Examples

```
.param hsimalloweddv=pwc(schev1, 0.3, schev2, 0.1)
.schev schev1 var=v(node1) val=1.5 edge=r
.schev schev2 var=v(node2) val=1.5 edge=f
```

The setting can be placed anywhere at the top-level netlist outside each subcircuit definition. Some commands can also be specified as local commands, which are either defined within the subcircuit definition or attached to a subcircuit instance.

To attach the command setting to the end of port list in the subcircuit definition, add the command within the subcircuit definition that affects all instances of the subcircuit, except for those instances with the same setting but a different command value. Use the following syntax.

```
.subckt subcircuit_name port_1 <port_2 ...>
hsimparameter_name1=value1 <hsimparameter_name2=value2
... >
```

Chapter 9: Simulation Commands

Specifying HSIM Commands

To set the command in the subcircuit instance to affect only the instance, append the setting to the end of the subcircuit instance call. Use the following syntax.

```
Xname port_1 <port_2 ...> subcircuit_name  
    hsimparameter_name1=value1 <hsimparameter_name2=value2  
    ...>
```

Specify exactly where the global commands are to be placed using one of the following syntax statements:

```
.param hsimparameter_name1=val1 <hsimparameter_name2=val2>  
.param hsimparameter_name3=pwc(t1, val1, t2, val2, ..., tN,  
    valN)
```

The following syntax applies to the subcircuit definition associated with the subcircuit instance.

```
.hsimparam <subckt=sub_name>  
<inst=subcircuit_instance_name>  
hsimparameter_name1=val1 <hsimparameter_name2=val2 ...>
```

- If the subcircuit definition is specified, the particular HSIM command is effective for each instance of the specified subcircuit definition `sub_name`.
- If the subcircuit instance is specified, the HSIM command is effective for the specified subcircuit instance `subcircuit_instance_name` at the top-level when `subckt` is not specified or at the specified `subckt` level.
- If both the subcircuit definition and the subcircuit instance are specified, then the subcircuit instance always overrides the subcircuit definition even if different values are applied.

Equivalent to adding the `.param HSIMANALOG=2` statement into the subcircuit definition of `samp`:

```
.hsimparam subckt=samp HSIMANALOG=2
```

Equivalent to appending `HSIMANALOG=2` to the end of instance `X1` within the subcircuit definition `samp`:

```
.hsimparam subckt=samp inst=X1 HSIMANALOG=2
```

Equivalent to appending `HSIMANALOG=2` to the end of top-level instance `X1`:

```
.hsimparam inst=X1 HSIMANALOG=2
```

In addition to setting the command at the global and subcircuit levels, some transistor-specific commands can be specified at the individual transistor level.

Unlike other commands, transistor-specific commands do not start with the HSIM keyword HSIM.

Transistor-specific commands are distinguished from the commands supported in SPICE by starting with the dollar sign character (\$). These commands must start with the dollar sign character (\$) so the commands are treated in the same way as a comment in a standard SPICE parser. Use the following syntax:

```
Mname drain gate source bulk model_name w=width l=length  
<ad as pd ps nrd nrs ...> $parameter_name1=value1
```

Specifying Commands in an External File

HSIM commands can be specified in a separate file outside the netlist. When specified at the top-level netlist, they can be directly relocated to another file. This is done by including the file in the main netlist using the .include statement. Refer to [Chapter 8, Input Netlist; .include on page 331](#).

The advantage of specifying control commands in an external file is that the fewest modifications are made to the main netlist. Specifying the commands in a separate file, then including the file in HSIM using the .include option, does not change the original circuit netlist.

Note: HSIM supports the TISPICE .SECTION statement conventions for including netlist commands.

Some HSIM commands can possess values as a piecewise constant (PWC) function of time in order to have different values for different time windows. Refer to the examples in [Specifying HSIM Commands in the Top-Level Netlist on page 342](#).

This feature provides higher flexibility in controlling the simulation speed and precision trade off. More conservative precision settings can be used in time windows when simulation precision is most critical, while aggressive speed settings can be used in modules or time windows when precision is not so critical. For example, in phase-locked loop (PLL) circuit simulation, higher precision is necessary when the PLL circuit is approaching the lock-in stage.

Note: This new pwc function is not supported by other SPICE simulators. It must be commented out if the netlist is simulated by other SPICE simulators.

Aliasing Commands to User-Defined Names

The .palias command is used to alias HSIM commands to a user-defined name:

```
.palias hsim_parameter_name alias1 alias2 ...
```

.palias can alias multiple alias names to the same HSIM command. The alias name must begin with hsim. The .palias command must be defined before using the alias name in the netlist.

Example

```
.palias HSIMMQS hsim_multi_rate
```

Control Commands for Ungrounded Capacitors

Capacitors have a complicated impact on simulation efficiency. If a capacitor has one of its terminals connected to a constant voltage source, such as V_{DD} or GND, this grounded capacitor does not require extra computations because the capacitance value only affects the diagonal entry value in the circuit admittance matrix.

However, if both terminals of a capacitor are connected to signal nodes, such an ungrounded capacitor occupies not only the diagonal entries associated with its two terminals but also the off-diagonal entries associated with the coupling between the two terminals. This causes a major impact on the computational complexity in solving the matrix equations. The existence of off-diagonal entries affect the matrix solution and might also create additional non-zero couplings in the off-diagonal entries, called fill-ins in the matrix solution.

The coupling effect between both terminals of an ungrounded capacitor, is proportional to the ungrounded capacitance value as well as the ratio of the capacitance to the total node capacitance of each terminal. The total node capacitance is defined to be the sum of all capacitances, both grounded and ungrounded capacitances, connected to the node. The default setting of modeling an ungrounded capacitor depends on the absolute capacitance value and its ratio to the total node capacitance. The ratio is defined as the maximum of the ratios to either terminal node capacitance.

Control Commands for Grounded Capacitors

Each linear capacitor from a signal node to a constant voltage source node is considered a grounded capacitor in order to:

- Improve simulation speed
- Reduce memory usage

HSIM does not keep these types of capacitor as a regular element. They become the property of the node and each node only keeps track of the sum of all such capacitors to the constant voltage source nodes. This does not affect simulation precision because constant voltage source nodes are AC grounded during transient simulation.

The limitation to this methodology is that when voltage source node currents are reported, HSIM can no longer keep track of where the lumped capacitors are connected. Therefore, the simulator cannot account for the corresponding capacitive current.

Missing these capacitive components mostly affects the peak current values of voltage source nodes and has less effect on the average current values. If the circuit does not have many grounded capacitors, then the effect on current values is minimal.

Control Commands for Floating Capacitors

HSIM can process floating capacitors with three different models, as shown in Figure 10.

Feedback

Chapter 9: Simulation Commands

Control Commands for Floating Capacitors

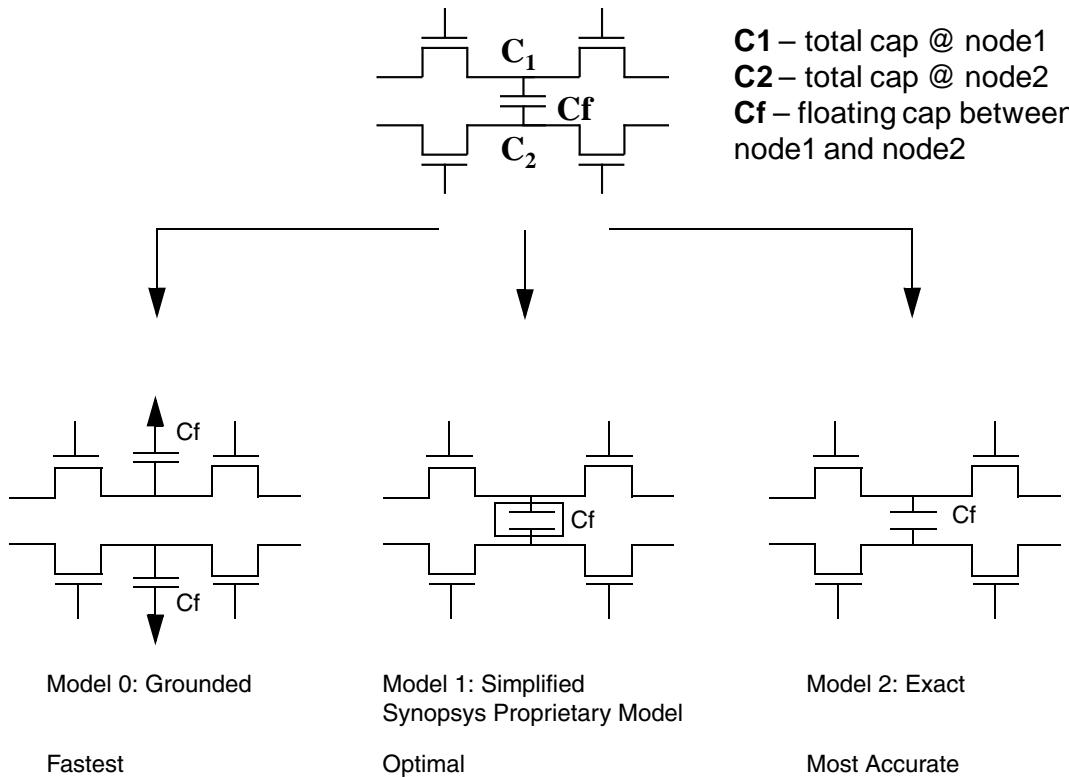


Figure 10 Floating Capacitor Models

You use the HSIMFCM command to specify the floating capacitor model:

HSIMFCM Argument	Description
0	The grounded model ignores the coupling effect between capacitor terminals. In this case, the capacitance coupling between terminals is modeled as two grounded capacitors: one connected between node A and the ground, and the other connected between node B and the ground.

Moreover, each capacitance value is the same as the original ungrounded capacitance value.

This model provides the best speed performance but may lose some precision if the coupling effect is important.

HSIMFCM Argument	Description
1	The simplified mode handles the coupling effect approximately. This model provides speed and precision performance in between those of grounded and exact models.
2	The exact model handles the coupling effect precisely by loading the capacitance value into the diagonal and off-diagonal entries of the admittance matrix. This model is precise, but if the circuit contains many ungrounded capacitors it might slow down the simulation considerably.
3	This is the default value for HSIMFCM. You can further control the floating capacitor model selection with the following commands: HSIMANALOG, HSIMFCAP, HSIMFCAPR, HSIMGCAP, and HSIMGCAPR. See the following figure. The other command defaults are: HSIMANALOG=1 HSIMFCAP=1E-13 HSIMFCAPR=0.5 HSIMGCAP=1E-15 HSIMGCAPR=0.02

Chapter 9: Simulation Commands

Control Commands for Isomorphic Matching

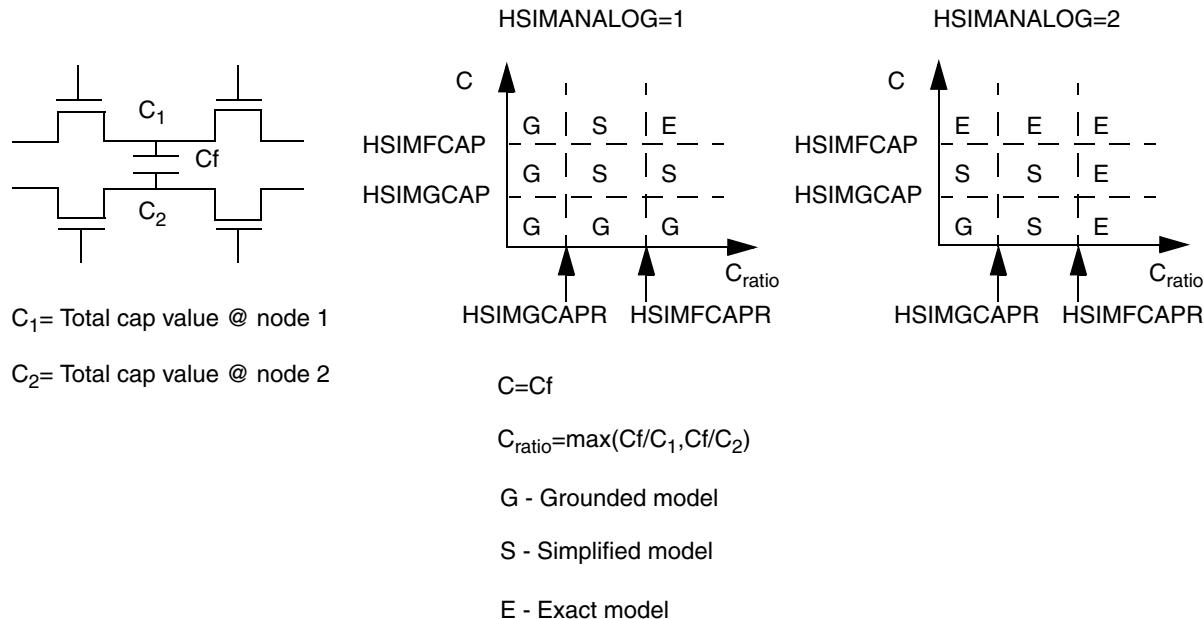


Figure 11 Controlling the Floating Capacitor Model Selection with HSIMANALOG, HSIMFCAP, HSIMFCAPR, HSIMGCAP, and HSIMGCAPR

In Figure 11, if the value on the C axis is a large portion of the overall node capacitance (the Cratio value), a more accurate model is used to preserve the coupling effect. If the C and Cratio values are smaller, the coupling effect has less impact, so a faster, less accurate model can be used. The HSIMFCAP, HSIMFCAPR, HSIMGCAP, and HSIMGCAPR commands control these values.

Note: When HSIMANALOG=2, the matrix values are more conservative.

Control Commands for Isomorphic Matching

If identical subcircuit instances have the same behavior, they can share:

- Storage
- Simulation results

These instances of the same subcircuit definition may behave differently if the driving port voltages or fanout port capacitances are different. Control commands are provided to define the tolerances in determining whether subcircuit instances can share the same computation result. The control commands are shown in [Table 61](#).

Table 61 Port Control Command Descriptions

Command	Description
Port Voltage Tolerance	Port voltage tolerance is defined by HSIMPORTV on page 126 such that two corresponding port voltages are considered matched if their difference is less than the value specified by HSIMPORTV on page 126 .
Port Capacitance	Port capacitance is defined as the total lumped capacitance contributed by elements. This can be either inside or outside the subcircuit connected to the port. The port capacitance tolerance is defined by HSIMPORTCR on page 125 .
Port Current Tolerance	Port current tolerance is defined by HSIMPORTI on page 125 such that two corresponding port currents are considered matched if their difference is less than the value specified by HSIMPORTI on page 125 . By default, only port voltage tolerance is considered. For small resistors and zero volt floating voltage sources, in order to obtain accurate currents through those elements, HSIMPORTI on page 125 may need to be specified. Units are in Amperes.

Two port capacitances are considered matched if the ratio of the capacitance difference to the average of two corresponding port capacitances is less than the value specified by [HSIMPORTCR on page 125](#).

DC Initialization

DC initialization is activated after the netlist processing and hierarchical database building phase is completed and before the beginning of a transient simulation. However, under certain special situations, there is no need to

Chapter 9: Simulation Commands

DC Initialization

perform DC initialization to achieve the desired results. The control command [HSIMDCINIT on page 76](#) is used to activate the DC initialization.

Note: When [HSIMDCINIT on page 76](#) is set to zero, DC initialization is suppressed and the transient simulation starts immediately following the preprocessing. Under this condition, the DC initialization proceeds as if running the transient simulation using zero initial state for each node voltage, unless the node has an initial condition setting through the use of the .ic command.

When [HSIMDCINIT on page 76](#) is activated by default or is set to 1, the DC initialization is terminated when a convergent state is reached, or when the number of iteration steps exceeds the limit specified by [HSIMDCITER on page 77](#). A convergent state is achieved if each node voltage change in an iteration step is less than the voltage specified by [HSIMDCV on page 79](#), and each net current is less than the current specified by [HSIMDCI on page 76](#). These commands can be configured to create levels of precision for DC operation needed in their specific circuits. DC initialization commands are described in [Table 62](#).

Table 62 DC Initialization Commands

Command	Type	Default Value	Location(s)
HSIMDCINIT on page 76	Boolean	1	Top-level
HSIMDCITER on page 77	Int	250	Top-level
HSIMDCV on page 79	Double	0.001	Top-levelsubckt Def.subckt Inst
HSIMDCI on page 76	Double	5E-7	Top-levelsubckt Def.subckt Inst
HSIMDCSTEP on page 78	Int	1000	Top-level
HSIMMULTIDC on page 114	Boolean	1	Top-level
HSIMENHANCEDC on page 87	Boolean	0	Top-level
HSIMKEEPNODESET on page 103	Boolean	0	Top-level
HSIMNODCNODES on page 115	Boolean	0	Top level

Table 62 DC Initialization Commands

Command	Type	Default Value	Location(s)
HSIMSTOPIFNODC on page 173	Boolean	0	Top level
HSIMWARNIFNODC on page 199	Boolean	0	Top level

Implicit Backward Euler Algorithm and Trapezoidal Integration Algorithm

Interrogation algorithms used by HSIM to solve equations are described in [Table 63](#).

Table 63 Integration Algorithms

Algorithm	Description
Implicit Backward Euler Algorithm	HSIM applies the Implicit Backward Euler Algorithm by default. It is a first-order numerical integration algorithm that solves differential equations. It has been well researched in the literature that the Implicit Backward Euler Algorithm suppresses the oscillation in the RLC circuit.
Trapezoidal Algorithm	To simulate oscillators containing inductors, set HSIMTRAPEZOIDAL on page 179 to 1. This command invokes the second-order numerical integration algorithm: the Trapezoidal Algorithm. Though more suitable for simulating the RLC oscillator circuit, the Trapezoidal Algorithm may cause artificial oscillation in high-impedance nodes.

Simulation Control Commands

Circuit simulations always contain a trade-off between:

- Speed
- Precision

Accurate simulation requires sophisticated models and conservative algorithms that demand more computations and result in longer simulation time. However,

Chapter 9: Simulation Commands

Simulation Control Commands

simulation speed can be pushed to go faster through the use of simplified models and more aggressive algorithms. This produces less accurate results. Selecting the optimal trade-off between speed and accuracy performance is a complicated task because the same model and simulation algorithm may have dramatically different speed and accuracy performance in various circuit types. For example, the simplified MOSFET table model may provide accurate performance for digital circuits, but provide insufficient accuracy for high-sensitivity analog circuits.

Simulation Speed Control Summary

HSIM precision and speed performance are affected by many control commands which control the selection of various models and computational algorithms. [HSIMSPEED on page 150](#) is a macro command containing a few HSIM commands controlling HSIM's simulation speed and precision. It is also possible to directly set these individual commands and overwrite the default command value defined by HSIMSPEED.

[HSIMSPEED on page 150](#) control commands are summarized in the [Table 64 on page 354](#) and [Table 65 on page 355](#).

Table 64 Simulation Control Commands

Command	Type	Default Value	Location(s)
HSIMSNCS on page 148	Boolean	1	Top-level
HSIMMQS on page 113	Int	1	Top-level
HSIMSTEADYCURRENT on page 172	Double	1E-8	Top-levelsubckt Def.subckt Inst
HSIMALLOWEDDV on page 59	Double	0.3	Top-levelsubckt Def.subckt Inst
HSIMTAUMAX on page 176	Double	2E-9	Top-level Def.subckt Inst
HSIMSPICE on page 170	Int	0	Top-levelsubckt Def.subckt Inst
HSIMGCSC on page 91	Boolean	1	Top-levelsubckt Def.subckt Inst

Table 64 Simulation Control Commands (Continued)

Command	Type	Default Value	Location(s)
HSIMFLAT on page 89	Boolean	1	Top-levelsubckt Def.subckt Inst

[Table 65 on page 355](#) shows the relationship between HSIMSPEED and other control commands.

Table 65 Speed Command Relationships

HSIMSPEED on page 150	0	1	2	3	4	5	7	8
HSIMSNCS on page 148	0	1	1	1	1	1	1	1
HSIMFLAT on page 89	1	1	1	1	1	1	1	1
HSIMGCSC on page 91	1	1	1	1	1	0	0	0
HSIMALLOWEDDV on page 59	0.1	0.3	0.3	0.3	0.3	0.5	0.5	0.5
HSIMSTEADYCURRENT on page 172	1E-8	1E-8	1E-8	1E-8	1E-7	1E-7	1E-7	1E-7
HSIMMQS on page 113	0	0	0	1	1	1	1	1

Note: HSIMSPEED=6 is now obsolete.

Piecewise Constant Setting

The control commands for adjusting HSIM simulation precision and speed, have a continuous effect from the beginning to the end of simulation. In practice, there may be different precision requirements in different simulation cycles such that the higher precision performance is crucial only in certain time windows. For example: high precision is necessary in PLL circuit simulation when it approaches the lock-in state.

To optimize the trade-off in simulation speed and precision, it is preferred to have control commands adjustable in different time windows. A more

Chapter 9: Simulation Commands

Examples of HSIM Simulation Control Commands

conservative setting is applied to time windows when the precision is crucial and a more relaxed setting is applied otherwise in order to speed up simulation.

In HSIM, the following commands can have different values in different time windows in order to achieve better precision-speed trade-off:

- [HSIMALLOWEDDV on page 59](#)
- [HSIMENHANCEDIDDQ on page 88](#)
- [HSIMSPICE on page 170](#)
- [HSIMSTEADYCURRENT on page 172](#)
- [HSIMTAUMAX on page 176](#)

Except for [HSIMTAUMAX on page 176](#), each command can be set within a subcircuit such that the effect covers selected time windows in specified subcircuit.

Example

```
.param HSIMALLOWEDDV=pwc(0, 0.5, 100n, 0.1, 200n, 0.01, 350n, 0.3)
```

This setting assigns piecewise constant values to command [HSIMALLOWEDDV on page 59](#), which is 0.5 from time 0 to 100 ns, then 0.1 from 100 ns to 200 ns, 0.01 from 200 ns till 350 ns, and finally 0.3 after 350 ns.

Note: Unlike pwl, pwc requires the use of commas. If commas are not used to separate the variables, a syntax error will occur.

Examples of HSIM Simulation Control Commands

The following examples illustrate how and when to set HSIM simulation control commands, and the results of setting the following commands:

- [HSIMSPEED on page 150](#)
- [HSIMPORTCR on page 125](#)
- [HSIMGCAP, HSIMGCAPR, HSIMFCAP, and HSIMFCAPR](#)

HSIMSPEED Example

This example is stored in the directory \$HSIM_HOME/tutorial/fadd8. It demonstrates how to control the speed of a simulation.

The steps for running this demonstration follow.

1. Run all the scripts. There are six run scripts for running six simulations with different settings on command [HSIMSPEED](#) on page 150.

Table 66 Run Scripts

Script	Runs
run0	Sets HSIMSPEED on page 150 to 0, which is the most precise mode
run1	Sets HSIMSPEED on page 150 to 1
run2, run3, run4, run5	Sets HSIMSPEED on page 150 to 2, 3, 4 and 5, respectively.

2. Observe the CPU time used for each simulation. The simulation speed accelerates with the larger HSIMSPEED value setting, although the speedup is quite insignificant due to the small circuit size in this example.
3. Compare the simulation results of each simulation, using the results of run1 as reference. The simulation results shown in [Figure 12 on page 358](#) are the overlay of [HSIMSPEED](#) on page 150=1, 3 and 5. In this example, the waveforms of [HSIMSPEED](#) on page 150=1 and [HSIMSPEED](#) on page 150=3 are almost identical, but [HSIMSPEED](#) on page 150=5 has significant delay difference compared with the other two simulation results.

Chapter 9: Simulation Commands

Examples of HSIM Simulation Control Commands

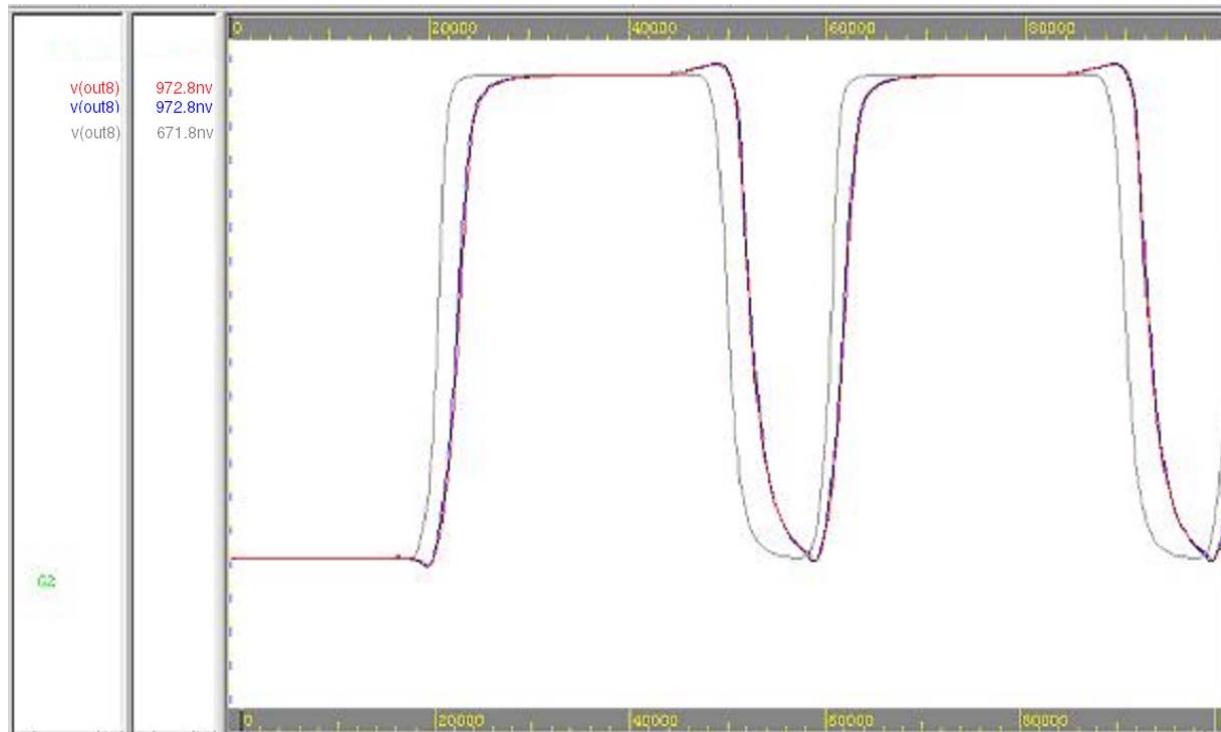


Figure 12 Results of Three Different HSIMSPEED Settings

HSIMPORTCR Example

The [HSIMPORTCR](#) on page 125 example is stored in directory \$HSIM_HOME/tutorial/isocap. It demonstrates the role of fanout load capacitance in isomorphic matching.

There are two identical circuits that have different control command settings. The circuit contains two similar inverter chains with slightly different port capacitance values.

In the first circuit, each inverter chain has slightly different capacitance values on some nodes.

Node 2: 10 fF
Node12: 10.05 fF

HSIM will treat the two nodes in the same way because the difference of capacitance is less than the default setting of 1%. Node 3 has 100 fF and node 13 has 101 fF. HSIM will also treat the two nodes in the same way because the

percentage difference is the 1 fF divided by the average node capacitance at nodes 3 and 13. The average node capacitance is higher than 100.5 fF due to an inclusion of the average transistor gate and drain/source capacitances connected to node 3 or 13. Node 4 has 100 fF and node 14 has 105 fF, HSIM will treat them differently because the difference exceeds 1%. Node 5 has 10 fF and node 15 has 10 fF, HSIM will also treat them the same.

First Demonstration

Following are the instructions for this demonstration.

1. Invoke run script run1.
2. View the simulation results.

Observe that the results of node 2 and node12 are the same, and those of node 3 and node 13 are the same. The results of node 4 and node 14 are different and those of node 5 and node15 are different because their inputs are different. Since the differences between the two inverter-chains are relatively small, HSIM can be forced to treat the two inverter-chains in the same way.

Second Demonstration

For the second circuit, the command [HSIMPORTCR on page 125](#)=0.05 is used which forces HSIM to match two similar nodes with a capacitance tolerance of 5%. This effectively matches the two inverter chains.

The instructions for this demonstration follow.

1. Execute run script run2.
2. Observe the corresponding nodes. Each inverter chain has exactly the same waveform. When compared to the results of the first simulation, the maximum error at node 15 is approximately 2.5%.

Chapter 9: Simulation Commands

Examples of HSIM Simulation Control Commands

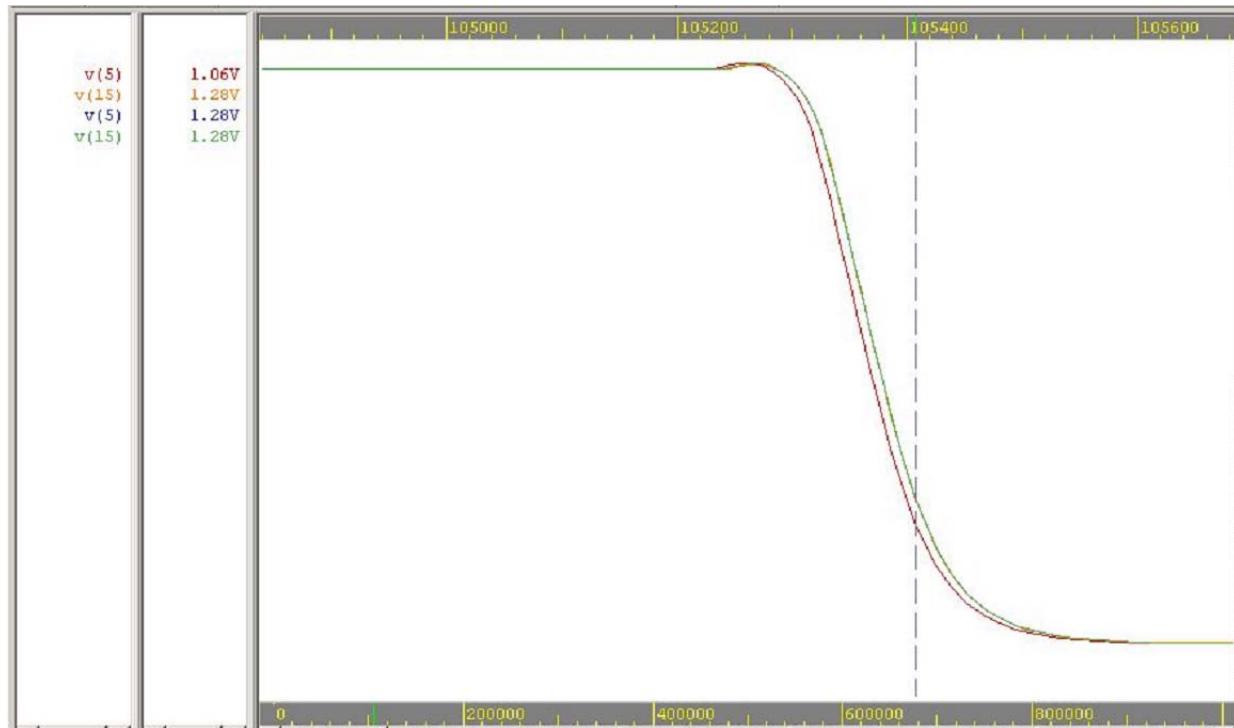


Figure 13 Results of Different HSIMPORTCR Settings

HSIMGCAP, HSIMGCAPR, HSIMFCAP, HSIMFCAPR Example

This example is stored in directory \$HSIM_HOME/tutorial/gscap. The example demonstrates how to control the splitting of ungrounded capacitors in a netlist.

This circuit has two identical inverter chains with 0.1 fF, 1 fF, 10 fF, 100 fF or 1 pF cross-coupling capacitors added between them at certain regularly spaced nodes. Depending on the settings of HSIMGCAP, HSIMGCAPR, HSIMFCAP, and HSIMFCAPR, some small ungrounded capacitors might be split to the ground; some large ungrounded capacitors might be kept as the regular capacitors; and the medium ungrounded capacitors will be approximated.

The instructions for this demonstration follow.

1. Execute script run1. This script uses default settings (HSIMGCAP=1fF, HSIMGCAPR=0.02, HSIMFCAP=100fF, HSIMFCAPR=0.5) and results in one split capacitor, three approximated capacitors, and one regular capacitor.
2. Execute script run2. This script sets HSIMGCAP to 1.0E-12 and HSIMFCAP to 1.0E-11, which split all the ungrounded capacitors.
3. Execute script run3. This script sets HSIMGCAP=0.01fF, HSIMGCAPR=0.00001, HSIMFCAP=0.02fF, and HSIMFCAPR=0.00002 and keeps all of the ungrounded capacitors as regular capacitors.

The simulation results are shown in [Figure 14 on page 362](#). The simulation results of run1 and run3 are fairly close, whereas results of run2 are poor. This is to be expected due to a total missing of the coupling effect between the two inverter-chains in run2. The agreement in results between run1 and run3 is excellent. This indicates that the default setting of run1 can result in a very accurate result. The default settings intelligently approximated the coupling effect: splitting the small ungrounded capacitance, modeling the intermediate ungrounded capacitance with approximation, and precisely handling only the large ungrounded capacitance.

These results demonstrate that computing time can be greatly reduced, without losing much precision, when very large circuits are simulated by using this intelligent cross-coupling capacitance handling method.

Feedback

Chapter 9: Simulation Commands

Examples of HSIM Simulation Control Commands

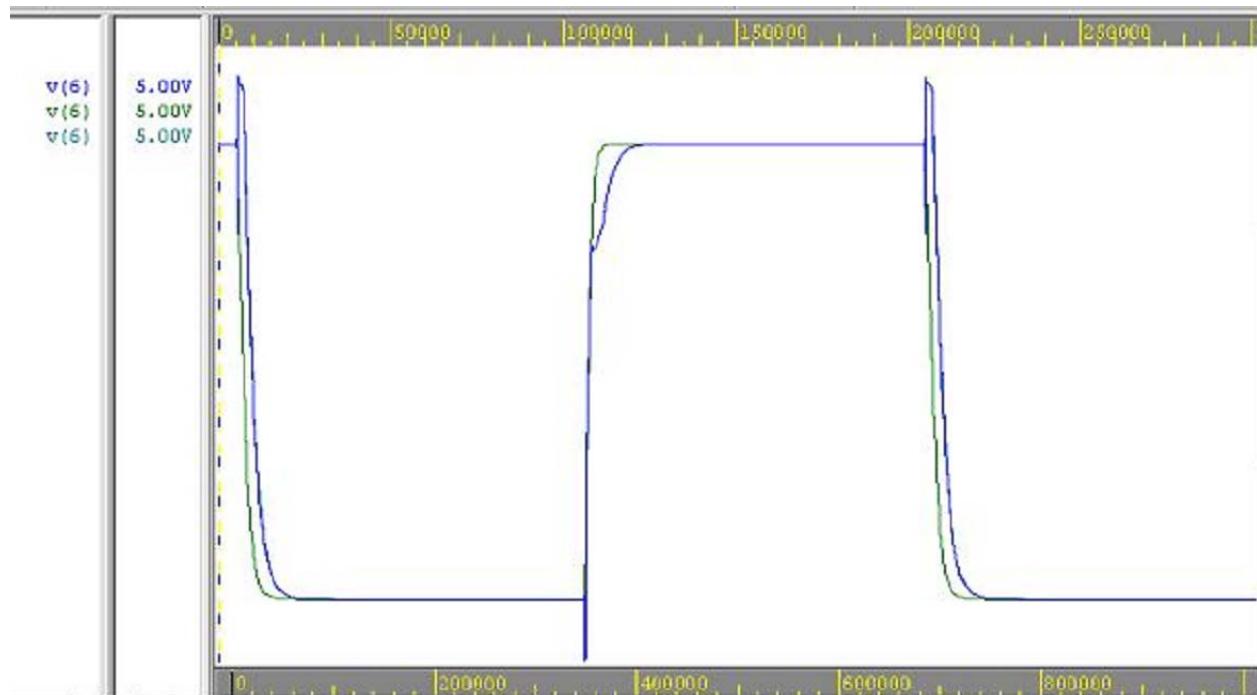


Figure 14 Results of Different HSIMFCAP and HSIMGCAP Settings

Post-Layout Backannotation

Describes the post-layout backannotation features of HSIM.

- [Signal Net Post-Layout Control](#)
- [HSIM-Supported DSPF Format for Hierarchically Extracted Feed-Through Nets](#)

Signal Net Post-Layout Control

The presence of parasitic RCs increases memory usage and slows down the simulation performance. To improve the performance in post-layout simulation containing a large number of parasitic RCs, [HSIMPOSTL on page 126](#) controls the reduction of signal net parasitic RCs to trade off minor precision loss for significant speedup.

This section covers the following topics:

- [Post-Layout Devices and RC Backannotation](#)
- [Device Parameter Format \(DPF\)](#)
- [RC Backannotation \(DSPF/SPEF\)](#)
- [Node Capacitance Backannotation](#)

Post-Layout Devices and RC Backannotation

A post-layout netlist typically contains:

Chapter 10: Post-Layout Backannotation

Signal Net Post-Layout Control

- Transistors
- Parasitic capacitors
- Parasitic resistors

There are two types of post-layout netlists:

- A self-contained netlist includes all the transistor, resistor, and capacitor elements in the same netlist. The netlist can be flat or hierarchical and can be readily simulated by HSIM.
- A back-annotated netlist has two separate netlists:
 - A netlist of parasitic RCs only, represented in DSPF/SPEF.
 - A transistor netlist without parasitic RCs. A transistor netlist can be either a pre-layout hierarchical netlist or extracted netlist from the layout.

Device Parameter Format (DPF)

If the transistor netlist is a pre-layout hierarchical netlist, the geometric device data can be back-annotated to the hierarchical netlist from a device parameter format (DPF) netlist file. The DPF is a flat SPICE netlist containing only MOSFETs with extracted geometric data. Alternatively, the DSPF file instance section, if present, can also be used to back-annotate device geometry data. For a list of the PF annotation-related commands, see [DPF/SPEF Backannotation Commands on page 38](#).

RC Backannotation (DSPF/SPEF)

The transistor netlist is ready for simulation but no parasitic RC element is simulated unless the parasitic resistors and capacitors in the DSPF/SPEF netlist are annotated into the transistor terminals through the setting of parameter [HSIMSPF](#), [HSIMSPEF](#) on page 154 .

Node Capacitance Backannotation

Other than the SPF file, HSIM can also annotate capacitances represented in the following format by the pair of node name and the corresponding node capacitance. The filename of the capacitor file is specified by [HSIMCAPFILE on page 72](#).

HSIM-Supported DSPF Format for Hierarchically Extracted Feed-Through Nets

There are two issues related to RC backannotation for feed-through nets using HSIM:

- Difference between logical and physical pins
- Capacity limits in the RC extractor

In a physical design, multiple physical pins can map to one logical pin which is called the feed-through pin. HSIM reads an original logical netlist. When back-annotating parasitic RC through DSPF files, multiple physical pins merge into one, which creates an inaccurate RC backannotation. Examples of feed-through nets include:

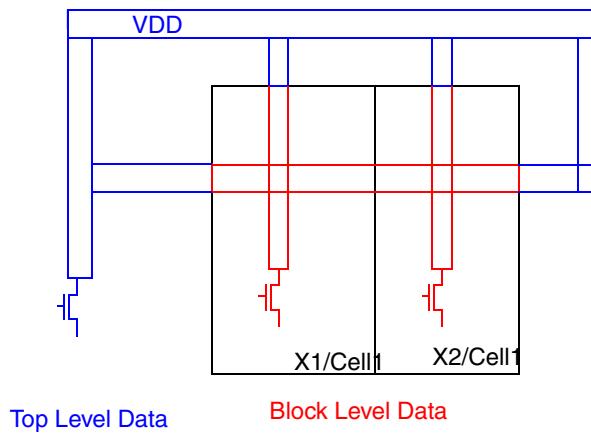
- Physical Designs: Power Nets
- Memory Designs: Word lines, bit lines, and power nets

Due to the capacity limitations of RC extractors, power net extraction sometimes becomes impossible. One solution aimed at conquering these capacity limitations is to divide the workload by extracting the blocks and top-level separately.

To solve feed-through net and extractor capacity problems, HSIM takes advantage of extended DSPF files to model physical connections. HSIM reads the logical netlist, RC back-annotated by the extended DSPF's, and correctly connects the additional physical pins. [Figure 15 on page 366](#) provides an example of this process.

Chapter 10: Post-Layout Backannotation

HSIM-Supported DSPF Format for Hierarchically Extracted Feed-Through Nets

*Figure 15 Extended DSPF Example*

[Figure 15 on page 366](#) shows the layout of instances X1 and X2. These instances are referenced by the same Cell1 master cell. Geometries shown in blue are top-level data and geometries in red are block-level data, also referred to as cell level data. Two DSPF's are generated by the following extraction flow:

- Extract Cell1 once
- Extract the top-level

Cell1 contains 3 externally connected VDD physical connection points. Logically, one VDD pin is represented in the logical netlist in this example. X1 and X2 are abutted cells. One X1 VDD pin is connected to a X2 VDD pin.

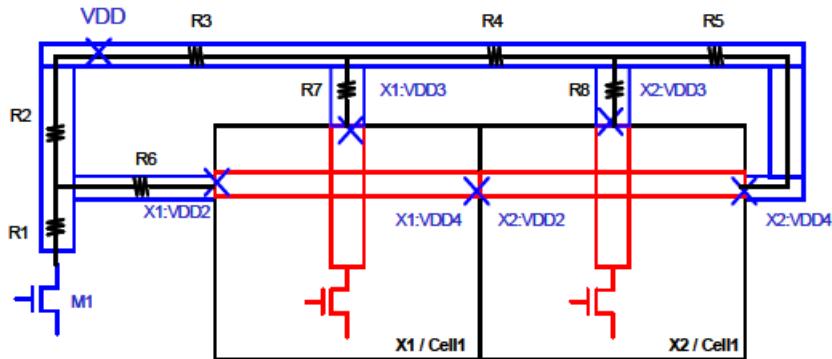


Figure 16 Top-Level Geometry Extraction

In [Figure 16 on page 367](#), the geometry extractions are shown as follows:

- blue-to-blue connections: Top-level geometry extraction to top-level device connections.
- blue-to-red connections: The top-level geometry extraction to hierarchical instance(X1 and X2) pin connections.
- red-to-red connections: The hierarchical blocks recognize X1 and X2 pin connections.

Top-Level DSPF File

An instance name and pin name must be defined for all instance pins connected to the top-level VDD net as shown in the following top-level DSPF file syntax example:

Note: In the following example, comments following a \$ sign are not present in the actual file. These comments are for documentation purposes only.

Chapter 10: Post-Layout Backannotation

HSIM-Supported DSPF Format for Hierarchically Extracted Feed-Through Nets

```

* | NET VDD xxPF          $Net to be back-annotated
* | P  (VDD I xxPF X Y)   $beginning of the net
* | I  (X1:VDD2 X1 VDD2 B xxPF X Y) $description
* | I  (X1:VDD3 X1 VDD3 B xxPF X Y) $Defining pin for the net

* | I  (X1:VDD4 X1 VDD4 B xxPF X Y) $Defining Instance pin
* | I  (X2:VDD2 X2 VDD2 B xxPF X Y) $connection to the net at
* | I  (X2:VDD3 X2 VDD3 B xxPF X Y) $the level top
* | I  (X2:VDD4 X2 VDD4 B xxPF X Y)
* | I  (M1:SRC M1 SRC B xxPF X Y)

R1 M1:SRC VDD:2 0.1          $Resistor connects to a top
                             $level transistor and a top
                             $level vdd subnet

R2 VDD:2 VDD 0.1            $Resistor connects to top
                             $level vdd subnets

R3 VDD VDD:3 0.1
R4 VDD:3 VDD:4 0.1
R5 VDD:4 X2:VDD4 0.1       $Resistor connects to a
                             $top level
                             $subnet and an inst. pin

R6 VDD:2 X1:VDD2 0.1
R7 VDD:3 X1:VDD3 0.1
R8 VDD:4 X2:VDD3 0.1
R9 X1:VDD4 X2:VDD2 0.1     $Resistor connects to two
                             $inst. pins

C1 VDD:2 0 ...              $Cap definition

```

Block-Level RC Extraction

RC extraction of block-level data with pin definition which is based on pin assignment consistent to the top-level as shown in [Figure 17 on page 369](#).

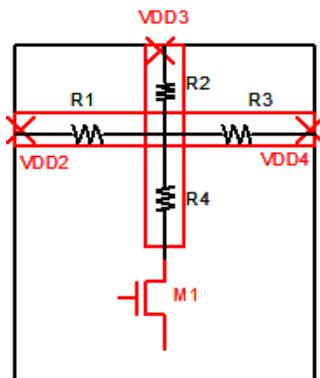


Figure 17 Block-Level Data Pin Definition

Block-Level DSPF File

The following is an example of the block-level DSPF file:

```
.subckt Cell1
* | NET VDD xxPF
* | P (VDD2 I xxPF X Y)
* | P (VDD3 I xxPF X Y)
* | P (VDD4 I xxPF X Y)
* | I M1:SRC M1 SRC
R1 VDD2 VDD:2 0.1
R2 VDD3 VDD:2 0.1
R3 VDD4 VDD:2 0.1
R4 VDD:2 M1:SRC 0.1
...
...
```

\$Needs .subckt definition if
\$not top level
\$Net definition

\$Pin definition which makes
\$external
\$connection at one level up
\$in the hierarchy

Enhanced HSIM Subcircuit Instance Parameters

This subcircuit instance description is used to support hierarchical DSPF flows. The location and orientation of hierarchically extracted subcircuits are with respect to the instance's local axes, which need to be translated into the global axes.

Chapter 10: Post-Layout Backannotation

HSIM-Supported DSPIF Format for Hierarchically Extracted Feed-Through Nets

StarRC provides HSIM with a Hierarchical Instance position file containing the location information for each hierarchical instance.

To generate a placement file generated by StarRC, add the following commands in the StarRC cmd_file when running extraction:

```
PLACEMENT_INFO_FILE: YES  
PLACEMENT_INFO_FILE: <filename> .
```

If no filename is specified, StarRC will dump out a file named blockname.placement_info. The placement file will contain the following information: Angle, Reflection, Location of Cell, Cell Name, and Cross-referenced Instance Name. This placement file is then read into HSIM using the HSIMSPFPOS parameter.

The following example shows sample output syntax.

Example 35 Placement File Generated by StarRC

```
* SKIP CELL PLACEMENT FILE  
* VENDOR "Synopsys, Inc."  
* PROGRAM "Star-RCXT X-2005.06" - Program name and version  
* DATE "Mon Dec 5 11:32:56 2005" - Date when file was generated  
* UNIT "MICRONS"  
TOP_CELL=<cell_name> <instance_name> <cell_name> <X-coord> <Y-coord> <Angle> <reflection>  
XSI_0 INV1 49 132 0 NO - a cell not rotated or flipped  
XSI_50 XOR2 484 132 180 NO - a cell rotated 180 counter-clockwise (ccw)  
XSI_61 XOR2 124 312 180 YES - a cell reflected about the X-axis and then rotated 180 degrees ccw
```

Note: If both rotation and reflection are necessary, the instance will first be reflected about the x-axis and then rotated counter-clockwise by the angle provided. These operations follow the GDSII standard syntax.

Postlayout Acceleration (PLX), Structured Backannotation (SBA), and SP2DSPF Utility

This chapter describes how to use the HSIM PLX postlayout acceleration option for hierarchical backannotation (HBA) technology, structured backannotation (SBA) technology, and the SP2DSPF utility.

Postlayout analysis for nanometer effects is now a necessity. In dynamic circuit simulation, postlayout analysis can be performed by simulating a circuit with all the parasitic RCs and coupling capacitors extracted from the layout. A brute force approach to this problem—simulation of the flat extracted netlist with all the parasitic RCs and coupling capacitors—is not practical because of the huge size of the extracted netlist. Direct simulation of the extracted netlist consumes an enormous amount of CPU time and memory. To solve this problem, HSIM provides the Postlayout Acceleration option (PLX), including an optimized hierarchical backannotation technology (HBA).

Backannotation Without Postlayout Acceleration

Backannotation in HSIM is done from the net definitions of the DSPF/SPEF files. Refer to the *HSIM® Simulation Reference Manual: Chapter 9, Post-Layout Backannotation* for a detailed description of the HSIM commands required to set up backannotation from DSPF/SPEF files. An example to illustrate backannotation without the PLX option is shown in [Figure 18 on page 372](#).

Chapter 11: Postlayout Acceleration (PLX), Structured Backannotation (SBA), and SP2DSPF Utility

Backannotation Without Postlayout Acceleration

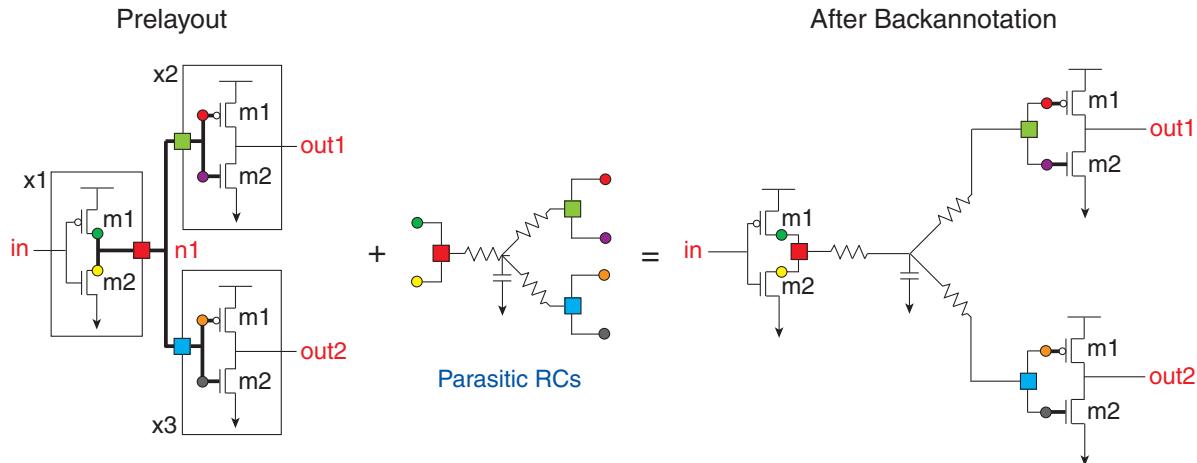


Figure 18 Backannotation Without the PLX Option

The node being back-annotated is n1 in the DSPF file, which contains all the parasitic RCs for that node. In order to back-annotate the node n1, HSIM disconnects all the devices from the node n1 and reconnects them to the corresponding ports of the net. In doing this, HSIM has to flatten the circuit to bring all the net connections on the same hierarchy level. This may not be a problem when only a small number of nets are back-annotated, or there are no coupling capacitors in the backannotation. In that case, HSIM may partially flatten the circuit, but the overall hierarchy is preserved. By maintaining the hierarchy in the circuit, HSIM can apply its hierarchical simulation methodology to achieve high simulation speed.

However, when the backannotation involves many nets (full chip extraction), or there are a great number of coupling capacitors, the hierarchy is destroyed with many random connections between the nets. HSIM must then flatten virtually the entire circuit, such that the advantages of hierarchical simulation cannot be used and the simulation speed degrades significantly. When the power and ground nets are back-annotated, which almost every device in the circuit has a connection to, HSIM also must flatten the circuit to bring all the connections onto the same level of the hierarchy. This again results in a huge flat circuit to be simulated, thus making the simulation run times unacceptable in many cases.

As a result, the standard flat backannotation approach can be efficiently applied only to either of the following:

- Block level postlayout simulations.
- Designs with low signal net coupling or low coupling through power nets.

When there is a need for full chip postlayout analysis, or when the power and ground nets need to be back-annotated, the PLX option provides for hierarchical backannotation to maintain the prelayout circuit hierarchy in the presence of coupling, retaining memory efficiency, and simulation performance.

Backannotation with Postlayout Acceleration

With postlayout acceleration, HSIM does not flatten the circuit in order to add the parasitic RCs. Applying a sophisticated optimization algorithm, HSIM distributes the parasitic RCs among different levels of hierarchy and subcircuits. The hierarchy of the prelayout netlist is completely preserved. The effect of hierarchical backannotation in the PLX option is shown in [Figure 19](#). In the schematic example, the backannotation of a signal net with hierarchical backannotation is shown.

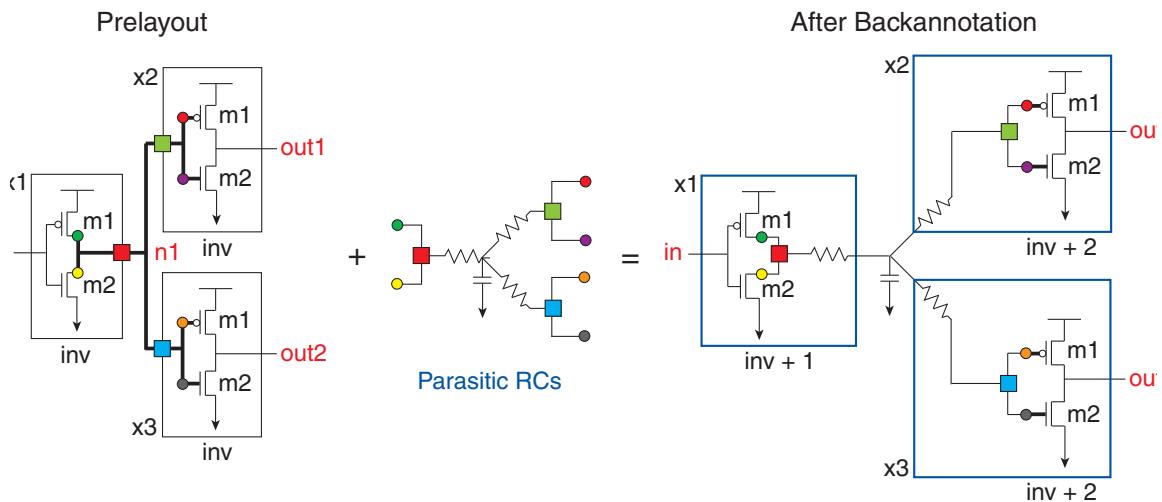


Figure 19 Effects of PLX on Backannotation

In addition to saving the hierarchy of the circuit, the PLX option processes all coupling capacitors using file operations. During these operations, the coupling capacitors are handled as ungrounded capacitors. Refer to the *HSIM Simulation Reference Manual: Chapter 8, Simulation Commands*, for additional information on coupling capacitors. Many of the coupling capacitors can be split with no effect on the accuracy, so the actual number of the capacitors that are added to the circuit is small. By filtering the coupling capacitors on a file basis,

HSIM HBA can accept virtually any number of coupling capacitors without a large increase in memory usage. The coupling capacitors added to the circuit are also distributed over the hierarchy, so all the benefits of the HSIM hierarchical simulation are fully utilized.

HSIMSPFPLX

HSIMSPFPLX is a flag that enables or disables postlayout acceleration and the HBA flow.

Syntax

In order to activate the PLX option, use the syntax:

```
.param HSIMSPFPLX=1
```

where:

- HSIMSPFPLX=0 (the default) turns PLX off.
- HSIMSPFPLX=1 sets HBA processes incoming DSPF/SPEF file parasitic RCs.

Structural Backannotation (SBA)

Structural backannotation is designed to handle structural mismatches allowed by LVS between schematic and layout.

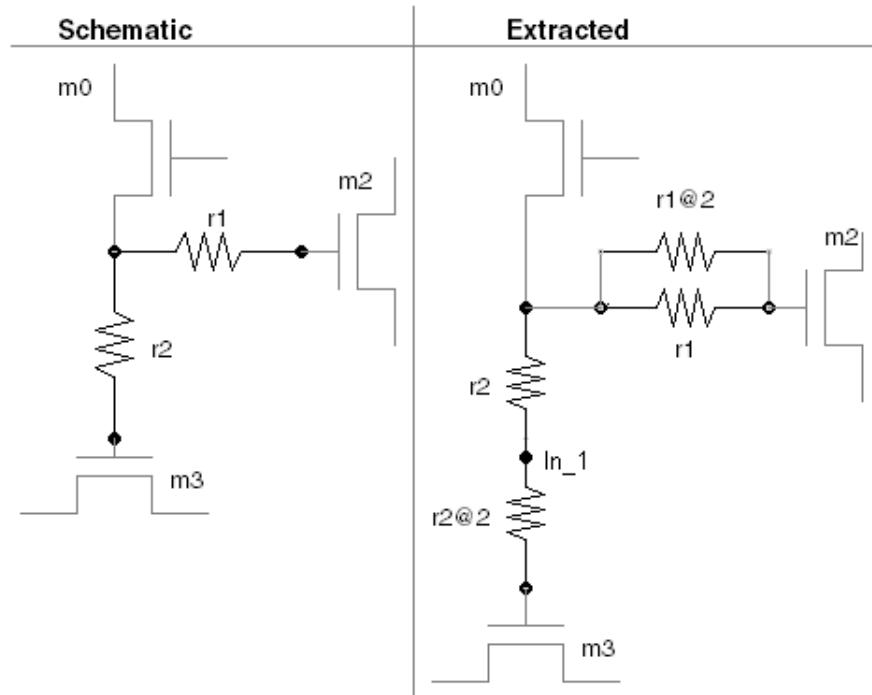
The SBA flow offers the following capabilities:

- SBA matches nets and devices by name and connectivity. It allows HSIM to identify structural differences between prelayout and postlayout netlists.
- SBA is performed before the simulation database is built, so it can back-annotate structural differences, as well as devices with non-generic physical parameters.
- When the SBA flow is used, DPF backannotation is not required and should not be used.
- SBA back-annotates structural differences and results in more accurate DSPF backannotation with fewer warnings.

The following examples demonstrate the features of SBA.

1. SBA back-annotates functional resistors with extra postlayout nodes.

In the figure below, the schematic netlist contains resistors, r_1 and r_2 . In contrast, the extracted netlist contains two additional resistors, $r1@2$ and $r2@2$, and an extra node ln_1 .



The SBA algorithm recognizes that schematic netlist is electrically equivalent to extracted one (they are LVS clean) and subsequently back-announces the extra ln_1 node and the additional $r1@2$ and $r2@2$ devices into original schematic netlist.

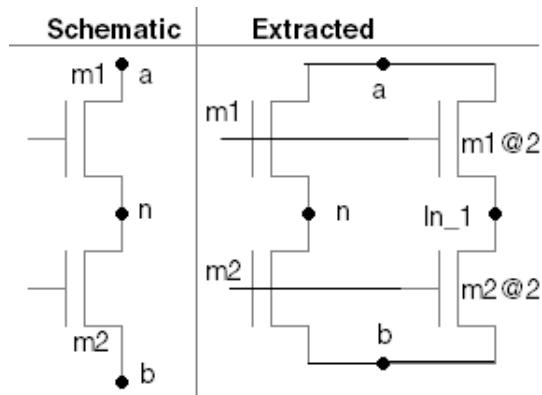
2. SBA back-announces complex finger devices.

In the figure below, the schematic netlist contains two MOSFETs, $m1$ and $m2$ in series. During layout, the $m1$ and $m2$ transistors were fingered, which resulted in an extracted netlist that contains two additional transistors, $m1@2$ and $m2@2$, as well as an extra node ln_1 .

Feedback

Chapter 11: Postlayout Acceleration (PLX), Structured Backannotation (SBA), and SP2DSPF Utility

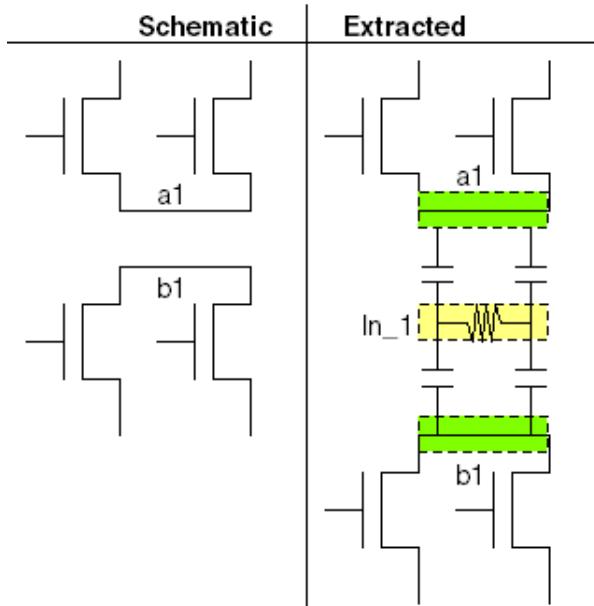
Structural Backannotation (SBA)



The SBA algorithm matches these two structures and back-annotates the missing ln_1 net and the $\text{m1}@2$ and $\text{m2}@2$ devices from the extracted netlist.

3. SBA back-annotates postlayout fill-in nets.

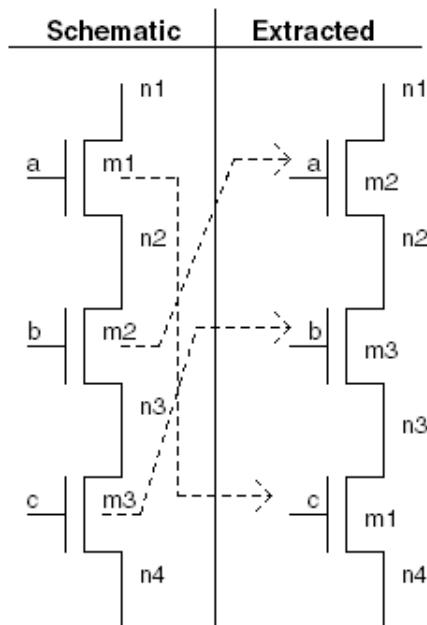
Fill-in nets are often inserted in the layout to planarize metal layers and reduce the risk of manufacturing failures. As a result, fill-in nets appear only in the extracted netlist. In the figure below, net ln_1 is a fill-in net that only appears in the extracted netlist, however, because ln_1 was placed in close proximity to nets a1 and b1 , it is coupled to these nets through parasitic capacitance.



SBA identifies postlayout fill-in nets and back-annotates them to the original schematic netlist.

4. SBA back-annotes swapped devices.

Schematic devices can often be swapped or rotated during layout. The resulting extracted netlist is LVS clean; however, the device connections might differ. The figure below demonstrates the swapping of the devices m1, m2 and m3 between the schematic and extracted netlists.



SBA detects swapped devices and back-annotes them into the original schematic netlist.

5. SBA back-annotes subcircuit or Verilog-A modeled devices.

To model the nanometer effects, extracted netlists can contain devices that are modeled in Verilog-A or are wrapped inside of subcircuits. In the figure below, the schematic netlist contains ideal resistor R1. However, during layout, the R1 resistor is implemented using NWELL. Furthermore, to accurately model the PN junction behavior of the NWELL resistor, it is extracted as subcircuit instance that contains a resistor and two diodes.

Feedback

Chapter 11: Postlayout Acceleration (PLX), Structured Backannotation (SBA), and SP2DSPF Utility

Structural Backannotation (SBA)

Schematic	Extracted
R1 a b 25	<pre>XR1 a b vss RN w=4u l=10usubckt RN p n sub w=1u l=1u R1 p n r='10*l/w' D1 sub p DN area='l*w/2' pj='(l+w)' D2 sub n DN area='l*w/2' pj='(l+w)' .ends</pre>

SBA can be configured using the [HSIMSBAPARAM](#) parameter to recognize the XR1 extracted instance as a resistor and back-annotate it to the schematic netlist.

Below is a diagram that summarizes the SBA flow.

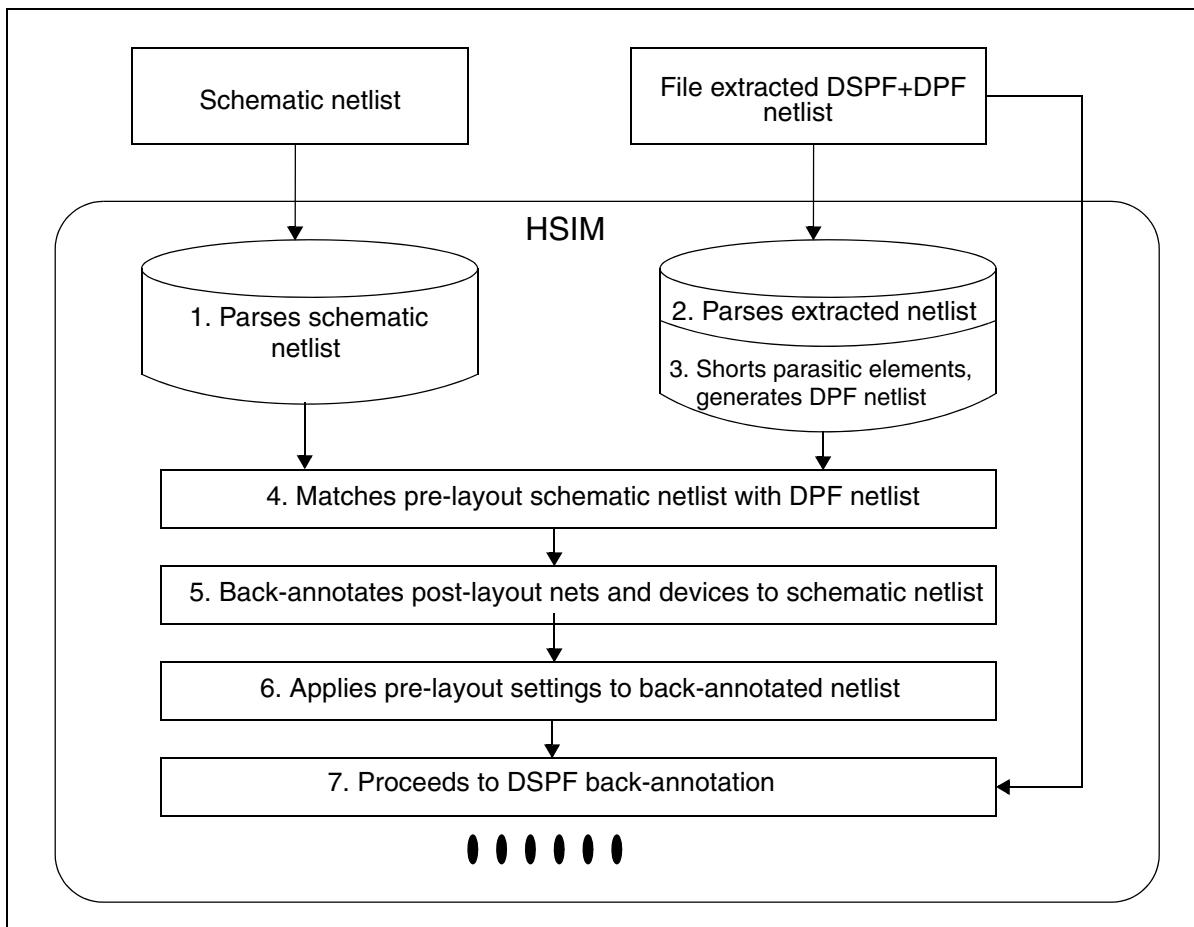


Figure 20 Structural Backannotation Flow

As shown in [Figure 20](#), the SBA flow works as follows:

1. HSIM parses the prelayout schematic netlist.
2. Parses the extracted DSPF+DPF netlists.
3. Shorts all parasitic RC's from the SPF section of the extracted netlist and internally generates flat extracted DPF netlist that contains only functional devices.
4. Matches prelayout schematic netlist with generated flat DPF netlist. The netlist-to-netlist comparison starts with top-level anchor nodes and propagates through entire design. As a result, nets and devices are matched by name and connectivity.

5. After the netlist-to-netlist comparison is complete, HSIM back-annotates the mapped devices from the extracted DPF netlist into the original prelayout schematic netlist.
6. Once extracted devices and nets are back-annotated into the original prelayout schematic netlist, the prelayout accuracy settings (.print, .measure and .IC) are re-applied to the structurally back-annotated netlist.
7. If the HSIMSPF parameter is set, HSIM performs DSPF backannotation.
8. Finally, HSIM builds the netlist database, performs DC initialization and proceeds to transient simulation.

Note: SBA requires a clean LVS database. If there are LVS mismatches, you see unmatched nodes or devices in the SBA report.

Running the Advanced SBA Flow

There are two ways to run SBA:

- The advanced SBA flow, described in this section, which is more flexible handling different design methodologies. However, it requires more user intervention to set up.
- The SBA flow, which is easier to set up but has specific requirements. For more information about this flow, see the [Running the SBA Flow](#) section.

Note: If you are a new SBA user, see the [Running the SBA Flow](#) section for more information about how to run SBA. Use the advanced SBA flow described in this section only when the requirements to run the SBA flow are not met.

To run the advanced SBA flow, set `HSIMSBA=1` in the top-level netlist. In addition, you need to specify the `HSIMSBANTL` command. The `HSIMSBANTL` command must point to a separate postlayout netlist file that has to be generated manually. The postlayout netlist file should be a self-contained SPICE netlist with the instance of the extracted subcircuit and device libraries as shown in [Figure 21](#).

For more information about the contents of the `HSIMSBANTL` file, see the [SBA Commands](#) section.

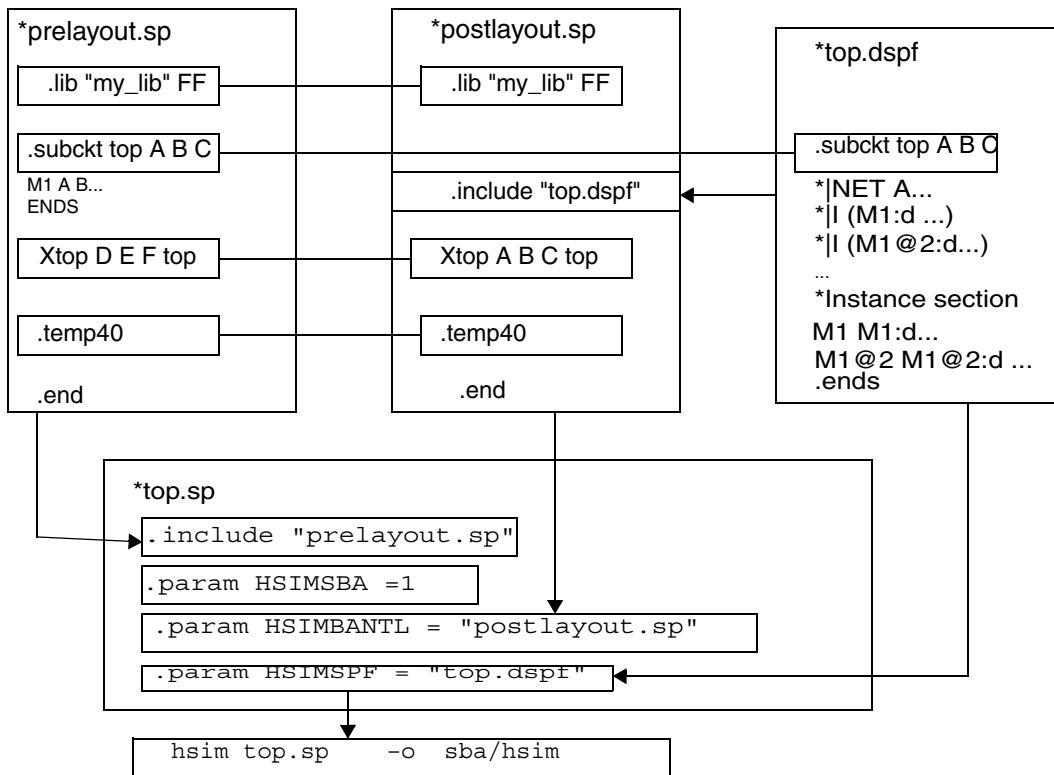


Figure 21 Running the Advanced SBA Flow

Running the SBA Flow

The SBA flow shown in Figure 22 lets you run backannotation using the information defined in the prelayout netlist and eliminates the need to create a postlayout netlist with the `HSIMSBANTL` command.

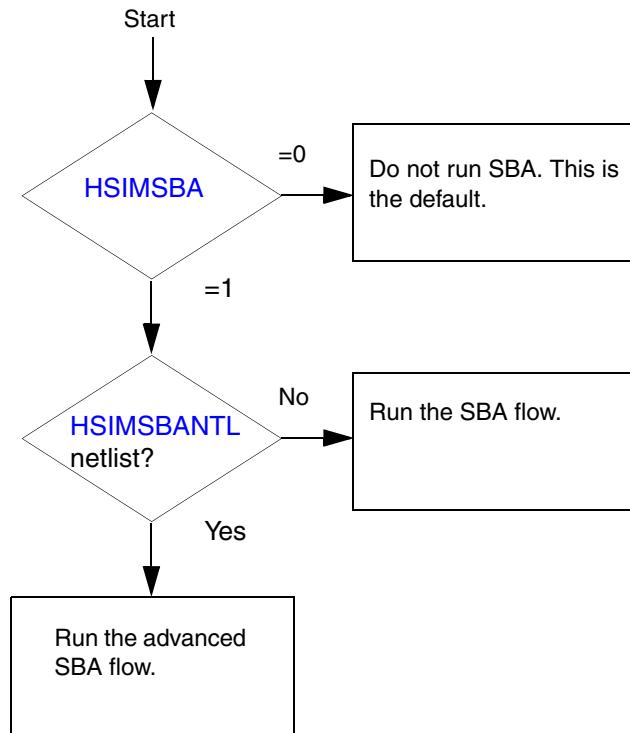


Figure 22 Selecting the SBA Flow To Run

You still need to set `HSIMSBA=1` to run SBA. When `HSIMSBA=1`, SBA looks for the `HSIMSBANTL` command. If found, the advanced SBA flow is run. If `HSIMSBANTL` is not found, the SBA flow is used.

To run the SBA flow, you need to specify:

- A postlayout netlist location in the prelayout netlist.
- The postlayout device models in the prelayout netlist.

Note: The advanced SBA and SBA flow, both support backannotation of multiple subckts.

Specifying the Postlayout Netlist in the SBA Flow

You use the `HSIMSBASF` command in the prelayout netlist to specify the postlayout netlist file name.

If **HSIMSBASPF** is not specified, SBA looks for the **HSIMSPF** command and uses its defined files for the postlayout netlist. You can omit the **HSIMSBASPF** command if the postlayout netlist is defined by the **HSIMSPF** command. An error message is displayed if **HSIMSBA=1**, but both **HSIMSBASPF** and **HSIMSPF** commands are not specified. [Figure 23](#) shows how SBA chooses the postlayout netlist to use.

For information about the **HSIMSPF** command, see the *HSIM® Simulation Reference Manual*.

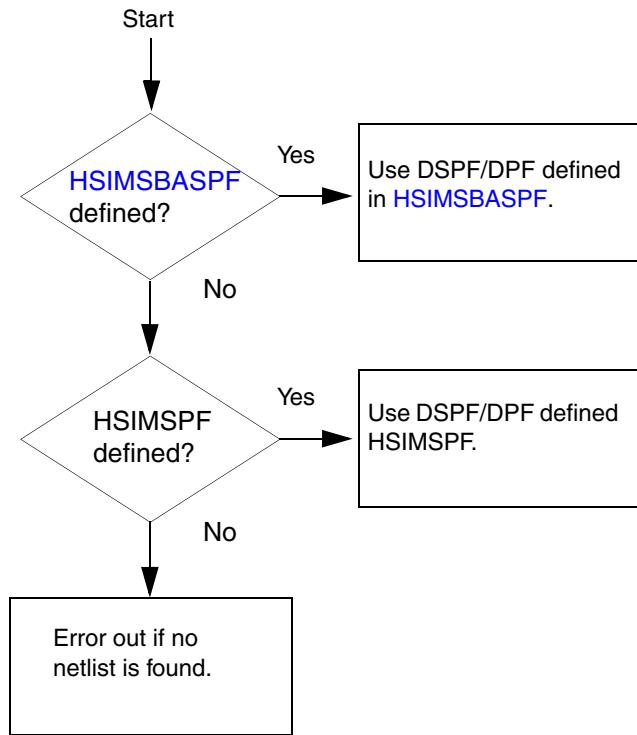


Figure 23 Selecting the Postlayout Netlist in the SBA Flow

Specifying Postlayout Device Models

The SBA flow uses prelayout netlist device models for parsing the postlayout netlist devices. All postlayout device models need to be defined in the prelayout netlist.

Defining the Backannotation Scope in the SBA Flow

When a subcircuit is defined in the postlayout netlist file:

- If the same subcircuit name is also found in the prelayout netlist, that subcircuit becomes the scope for SBA.
- When the subcircuit name is not found in the prelayout netlist, devices in the postlayout netlist are back-annotated to the top-level circuit.

Note: You can specify multiple subcircuits in the HSIMSBASPF netlist file.

When the subcircuit definition does not exist in the postlayout netlist file:

- SBA back-annotates to the top-level circuit.
- If multiple DSPF/DPF files are specified in HSIMSBASPF without a subcircuit definition, SBA back-annotates all devices to the top-level circuit.

Note: For both [Running the Advanced SBA Flow](#) and [Running the SBA Flow](#), HSIM outputs a *.stname file that contains HSIMSTNAME statements for all macro model devices. When you specify SBA (HIMSBA=1) and RC backannotation (HSIMSPF=*filename*), SBA, by default, identifies each macro model device and automatically outputs its terminal mapping data in an HSIMSTNAME command. At the end of SBA processing, the *.stname output file contains HSIMSTNAME commands for all macro model devices that SBA detects.

In the case when this automatic function is enabled, and you have manually specified HSIMSTNAME statements, SBA honors your specification. In the *.stname file, SBA places a comment character "*" in front of the automatically generated statements.

To disable this automatic function, which is on by default, specify
.param HSIMSBAAUTOSTNAME = 0.

SBA Commands

HSIMSBA

HSIMSBA is a flag that enables or disables the SBA flow.

Syntax

```
.param HSIMSBA=0 | 1
```

If HSIMSBA is set to 0, all SBA parameters are ignored and the SBA is disabled. If it is set to 1, the SBA flow is enabled. The default is 0.

Note: SBA requires the HSIM-PLX license feature.

HSIMSBASF

You use the HSIMSBASF command to specify the postlayout netlist file name. You use this command in the SBA flow.

Syntax

```
.param HSIMSBASF post_layout_file_name
```

post_layout_file_name can be a DSFS or DPF (StarRC IDEAL netlist) that contains the postlayout devices and device parameters.

HSIMSBACAP

Syntax

```
.param HSIMSBACAP=0 | 1
```

Specify HSIMSBACAP=0 to disable backannotation of functional capacitors. Specify HSIMSBACAP=1 to enable backannotation of functional capacitors (the default).

HSIMSBAFRES

Syntax

```
.param HSIMSBAFRES=0 | 1
```

Specify HSIMSBACAP=0 to disable backannotation of functional resistors. Specify HSIMSBACAP=1 to enable backannotation of functional resistors (the default).

HSIMSBANTL

HSIMSBANTL is used to specify the postlayout netlist required for the advanced SBA flow as shown in [Figure 21](#).

Syntax

```
.param HSIMSBANTL=<filename>
```

Where <filename> is the name of the postlayout netlist file to use. The postlayout netlist should contain the following:

- The top-level instance of the extracted subcircuit.
- An .INCLUDE statement of the extracted subcircuit in DSPF or DPF format.
- A device library or models that are used in the extracted subcircuit.
- An .END statement.

HSIMSBAPARAM

The HSIMSBAPARAM command is used to help match subcircuit instances for the SBA flow. You need to specify the HSIMSBAPARAM command if your DSPF/DPF primitive design devices are in the form of subcircuit models.

Syntax

```
.param HSIMSBAPARAM="sub=<subckt> parse {<device>}"
```

Syntax Description

<subckt>

The name of the wrapper subcircuit or Verilog-A module that requires special treatment during SBA.

<device>

The discrete device that is substituted by SBA during matching in place of wrapper subcircuit or Verilog-A module.

Example One

In the example below, the resistor R1 in the schematic netlist was extracted as a subcircuit instance XR1.

Schematic	Extracted
R1 a b 25	<pre> XR1 a b vss RN w=4u l=10usubckt RN p n sub w=1u l=1u R1 p n r='10*l/w' D1 sub p DN area='l*w/2' pj='(l+w)' D2 sub n DN area='l*w/2' pj='(l+w)' .ends </pre>

The R1 and XR1 are entirely different elements. R1 is a discrete resistor with two terminals. XR1 is an instance of a subcircuit "RN" with three connections. Subcircuit RN has one resistor and two diodes. For SBA to map the schematic "R1" resistor with the extracted "XR1" instance HSIMSBAPARAM must be set in the postlayout netlist as follows:

```
.param HSIMSBAPARAM="sub=RN parse {Rext p n 10}"
```

This parameter instructs SBA to map any instance of subcircuit "RN" in the extracted netlist to a discrete resistor "Rext p n 10". Observe that the name and the value of Rext resistor are irrelevant since they are used only to match the type of the device. However, to establish proper connectivity, the node names "p" and "n" of "Rext" resistor must match the port names in the ".SUBCKT RN p n sub" definition.

After "R1" and "XR1" devices are matched by SBA, the "XR1" instance from the extracted netlist is back-annotated in place of the ideal "R1" schematic resistor.

Example Two

In this example, the subcircuit instance XR1 in the schematic netlist was extracted as an ideal R1 resistor.

Schematic	Extracted
<pre> XR1 a b vss RN w=4u l=10usubckt RN p n sub w=1u l=1u R1 p n r='10*I/w' D1 sub p DN area='l*w/2' pj='(l+w)' D2 sub n DN area='l*w/2' pj='(l+w)' .ends </pre>	<pre> R1 a b 25 </pre>

For SBA to map schematic the "XR1" instance with the ideal extracted "R1" resistor, HSIMSBAPARAM must be set in the prelayout netlist as follows:

```
.param HSIMSBAPARAM="sub=RN parse {Rsch p n 10}"
```

After SBA matches the "XR1" instance in the schematic netlist with the extracted "R1" resistor, it back-annotates the extracted "R1" resistor. The final simulation netlist contains resistor "R1 a b 25".

Example Three

In this example, the XM1 subcircuit instance in the schematic netlist was extracted as the NHC subcircuit instance.

Schematic	Extracted
<pre> XM1 a b c d NCH w=0.45u l=0.18usubckt NCH d g s b w=0 l=0 .MA d g s b w=w l=l .ends </pre>	<pre> XM1 a b c d sub NCHEX w=0.5u l=0.2usubckt NCHEX d g s b sub w=0 l=0 .MB d g s b NMOS w=w l=l .D1 sub b DN area='l*w/2' pj='l+W' .ends </pre>

XM1 in the schematic netlist is an instance of the NCH subcircuit with 4 terminals, whereas in the extracted netlist it is an instance of the subcircuit NCHEX with 5 terminals. For SBA to map the XM1 schematic instance to the extracted one, HSIMSBAPARAM must be set in both netlists as follows:

Schematic:

```
.param HSIMSBAPARAM="sub=NCH parse {Mx d g s b N w=0.3u l=0.18u}"
```

This parameter instructs SBA to map any instance of NCH subcircuit to a discrete MOSFET device, Mx d g s b N w=0.3u l=0.18u. Observe that the device name Mx and device model N, as well as values of w and l, are arbitrary since they are used only to match the type of devices. However, to establish proper connectivity, the node names d, g, s and b of Mx must match the port names in the .subckt NCH d g s b definition.

Extracted:

```
.param HSIMSBAPARAM="sub=NCHEX parse {Mx d g s b N w=0.3u l=0.18u}"
```

This parameter instructs SBA to map any instance of the NCHEX subcircuit to a discrete MOSFET device, Mx d g s b N w=0.3u l=0.18u. Observe that the device name Mx and device model N, as well as values of w and l, are arbitrary since they are used only to match the type of devices. However, to establish proper connectivity, the node names d, g, s and b of Mx must match the port names in the .subckt NCHEX d g s b sub definition. The extra sub port in the subckt definition is ignored.

After these mapping steps, a correspondence between the XM1 instances in schematic and extracted netlists is established, and the extracted XM1 instance is back-annotated in place of the schematic XM1 instance.

HSIMSBAMSGLEVEL

HSIMSBAMSGLEVEL controls the number of warnings printed to the log file during SBA backannotation.

Syntax

```
.param HSIMSBAMSGLEVEL=0 | 1 | 2 | 3 | 4 | 5 | 100
```

Syntax Description

HSIMSBAMSGLEVEL can be set to any integer between 0-5. If it is set to 0, no SBA warnings are reported. When set to 5, detailed SBA warnings are reported to the HSIM log. The default value is 1.

When HSIMSBAMSGLEVEL=100 SBA outputs detailed warnings and error messages. In addition, it creates a *.match file that contains a summary of matched and unmatched nodes and devices, similar to the LVS discrepancy report. The *.match file helps you debug SBA node and device mismatches.

HSIMSBAMSGLIMIT

HSIMSBAMSGLIMIT sets the limit on the total number of warning messages issued during the matching stage of SBA.

Syntax

```
.param HSIMSBAMSGLEVEL=<value>
```

Syntax Description

<value> is an integral value that specifies the absolute limit on the number of warnings generated by SBA. The default is 500. This setting does not override the global warning limit settings (like '.opt warnlimit'). SBA stops generating warning messages as soon as either of these limits is reached.

HSIMSBAHIERID

HSIMSBAHIERID is used in the postlayout file to set the proper hierarchical separator in the device name.

If we have the following device In the extracted netlist:

```
*Instance Section  
M_X1/M3#1 A B C D NMOS ....
```

The following setting should be used in the postlayout netlist:

```
.param HSIMSBAHIERID="/"
```

HSIMSBASFX

HSIMSBASFX is used in the postlayout file to set the proper finger delimiter in the device name.

If we have the following device In the extracted netlist:

```
*Instance Section  
M_X1/M3#1 A B C D NMOS ....
```

The following setting should be used in the postlayout netlist:

```
.param HSIMSBASFX="#"
```

HSIMSBAPFX

HSIMSBAPFX is used in the postlayout file to set the proper prefix in the name of extracted devices.

If we have the following devices in the extracted netlist:

```
*Instance Section
M_X1/M3#1 A B C D NMOS ...
R_X1/R1 F N 100
X_X1/XR2 K L RN W=1 L=10 ...
```

The following setting should be used in the postlayout netlist:

```
.param HSIMSBAPFX="M_"
```

HSIMSBAAUTOSUBMODELS

When subcircuit models exist in the postlayout netlist, you can use HSIMSBAAUTOSUBMODELS instead of [HSIMSBAPARAM](#) to identify the subcircuit models. When you specify HSIMSBAAUTOSUBMODELS, HSIM automatically identifies subcircuit models and detects replacement elements. This command improves SBA usability when there are subcircuit models in the postlayout netlist.

Syntax

```
.param HSIMSBAAUTOSUBMODELS 0 | 1 | 2 | 3
```

0

Disables the detection of subcircuit models.

1

Instructs SBA to scan the top level of the postlayout netlist and search for subcircuit instances. If found, SBA applies HSIMSBAPARAM="subckt_name" to both the prelayout and postlayout netlists (unless there is an explicit HSIMSBAPARAM definition for the subcircuit).

2

Instructs SBA to scan all the subcircuits in the prelayout netlist and apply an HSIMSBAPARAM command for every subcircuit included in the netlist with a .lib directive (unless there is an explicit HSIMSBAPARAM definition for the subcircuit). This option only applies HSIMSBAPARAM to the prelayout netlist.

3

Applies option 1 first, then option 2 (the default).

SP2DSPF Utility

Generating a DSPF File From the Flat Extracted Netlist

Backannotation of parasitic RCs from detailed standard-parasitic format[\[1\]](#) (DSPF) or standard-parasitic extended format[\[2\]](#) (SPEF) files form the core technology for HSIM's signal integrity and power/signal net reliability analysis.

Some extraction tools have problems outputting their results as DSPF files. These tools extract flat SPICE netlists with parasitic RCs so that making flow changes that permit the use of DSPF files may require additional effort from HSIM users. In HSIM, simulating large flat extracted netlists is much less efficient when compared to simulating the hierarchical netlist back-annotated from a DSPF file. Moreover, the HSIM reliability features can only be used if backannotation is applied.

The SP2DSPF utility generates the following files from a prelayout netlist and flat or hierarchical extracted netlist with parasitic RCs:

- DSPF file
- DPF (Device Parameter Format) file (optional)

Non-parasitic devices from the extracted netlist, such as MOSFETs, are matched with their prelayout counterparts and can be output in the DPF file. The resulting DSPF and DPF files can then be used for back-annotating the prelayout netlist.

Running SP2DSPF

To run SP2DSPF, use the following syntax:

```
hsim -sp2dspf [<parameter file>] [<parameter setting> ...]
```

A list of parameters for the current run are contained in either of the following:

- Parameter file: For parameters used in numerous runs.
- Specified on the command line: For parameters with one-time or limited usage.

Note: Parameters read from the file or from the command line are identical.

The following example shows the hsim -sp2dspf syntax that refers to a parameter file.

```
hsim -sp2dspf p1_set.txt -anan gnd
```

Each parameter setting is a keyword prefixed with a dash (-) character and may be followed by parameter value(s). A single or multiple parameters can be inserted in a single syntax line when creating a parameter file.

SP2DSPF Utility Parameters

-pre

```
-pre <file name>
```

-pre is a required parameter that specifies a prelayout netlist.

-fpre

```
-fpre <format> (default: hspice)
```

-fpre specifies the prelayout netlist format where **format** is the netlist format name (for example, spectre). The list of available formats is the same as for HSIM. **hspice** is the default value.

-pretop

```
-pretop <subcircuit name>
```

-pretop specifies which prelayout netlist subcircuit is used as the top-level subcircuit. The top-level netlist instance is the default used as the top-level subcircuit.

-post

```
-post <file name>
```

-post is a required parameter that specifies the name of the extracted netlist.

-fpost

-fpost <format> (default: hspice)

-fpost specifies the extracted netlist format where *format* is the netlist format name (for example, spectre). The list of available formats is the same as for HSIM. hspice is the default value.

-posttop

-posttop <cell name>

-posttop specifies which extracted netlist subcircuit should be used as the top-level subcircuit. The default is the top-level netlist instances which are used as the top-level subcircuit.

-an

-an <node name> <node name> [<node name> <node name> ...]

-an specifies anchor node pairs used as starting points for the matching process. The pairs have the following form:

- First Node: The first node of each pair is a prelayout netlist node.
- Second Node: The second node is an extracted netlist node.

Only top-level nodes are allowed and at least one pair of anchor nodes is required. -an can be specified multiple times, as shown in the following example:

```
-an          0          gnd! pwr vdd
-an          vcc        vss
-anan <node name> [<node name> ...] (default: *)
```

-anan

-anan specifies anchor nodes with identical names in prelayout and extracted netlists. Specifying -anan *node_name* is equivalent to specifying -an *node_name* *node_name*, for example, -anan provides a shorter and more convenient way to specify anchor nodes when they are named identically in both input netlists.

Additionally, -anan supports wildcard characters such as (*) and (?). When wildcards are used as a node name in -anan, the anchor node selection is performed in accordance with the following:

1. All top-level nodes whose names match the wildcard are found in both prelayout and extracted netlists.
2. The names of prelayout and extracted nodes are compared and every pair consisting of a prelayout and an extracted node with identical names is used as a pair of anchor nodes.

The following example shows a wildcard used with the -anan parameter.

```
-anan *
```

-anan * is the default for -anan. In this example, the program searches all top-level nodes in both the prelayout and extracted netlists for nodes with identical names. These nodes are then used as anchor nodes for matching process.

-out

```
-out <file name prefix> (default: hsim -sp2dspf)
```

-out specifies the default output file name prefix. If the name is concatenated, it will have the following properties:

- log extension: it is used as the log file name
- .dpf: DPF output
- .dspf: DSPF

Note: If different names are explicitly assigned using the -outdpf and -outdspf parameters, -out is overridden for these parameters.

-dpf

```
-dpf on/off (default: off)
```

Enables/disables DPF file output.

-outdpf

```
-outdpf <file name>
```

-outdpf specifies output DPF file name and enables DPF file output, provided it is not disabled explicitly by **-dpf** parameter. **-outdpf** has no effect if DPF output is disabled by explicit **-dpf** parameter. If **-outdpf** is not specified, DPF file name is built in accordance with output file name prefix setting **-out**. If output DPF file name is exactly the same as output DSPF file name, the output is directed to the DSPF file to form its instance section. Otherwise, a standalone DPF file is generated.

-dspf

-dspf on/off (default: on)
-dspf enables or disables DSPF file output.

-outdspf

-outdspf <file name>
-outdspf specifies the output DSPF file name and enables DSPF file output, provided it is not disabled explicitly by **-dpf** parameter. If **-outdspf** is not specified, the DSPF file name is built in accordance with output file name prefix setting in **-out**.

-pinports

-pinports on/off (default: on)
-pinports specifies whether SP2DSPF should report all top postlayout top-level ports as pins in the resultant DSPF file. This option is only effective when postlayout top-level subcircuit is explicitly specified by **-posttop** parameter.

-ms

-ms <subcircuit name> <subcircuit name> [<subcircuit name> <subcircuit name> ...]
-ms specifies isomorphic subcircuit pairs in both prelayout and extracted netlists.

- The first subcircuit in the pair is a subcircuit from the prelayout netlist.
- The second subcircuit in the pair is a subcircuit from the extracted netlist.

This initial matching of subcircuits is used by **hsim -sp2dspf** as a hint.

Caution: Use `-ms` with CAUTION. Providing initial matching information to the program can greatly increase its performance and reduce its peak memory requirements, however if the initial matching is wrong, caused by different subcircuits being specified as matched, it may take MORE time and memory for the program to determine the correct answer.

There is usually no need to use `-ms` because the hierarchies of the prelayout and extracted netlists are substantially different; the prelayout netlist is mostly hierarchical while the extracted netlist is flat. However, for specific gate level circuits in ASIC designs, `-ms` may help, such as when an ASIC has a single (top) level with many instances of the cells. Netlists for cells with parasitic RCs are usually available, so the extracted netlist has additional parasitics only on the top-level. In this case, `-ms` can be used to set up the initial correspondence between the cells as shown in the following example:

```
-ms inv inv  
-ms nand2 nand2  
-ms nor2 nor2
```

-mm

```
-mm <model name> <model name> [<model name> <model name> ...]
```

`-mm` specifies pairs of device models that should be considered to be the same in both prelayout and extracted netlists.

- The first name in the pair is a device model from the prelayout netlist.
- The second name in the pair is a device model from the extracted netlist.

-opt outnf

```
-opt outnf <reverse|prefix> (default: prefix)
```

`-opt outnf` specifies output DPF/DSPF device naming convention as follows:

`reverse`

In reverse mode, device names are built in accordance with reverse hierarchy (bottom-to-top) convention; e.g., `-m12.x32.x4.xcell`.

prefix

In prefix mode device names are built in accordance with straight hierarchy (top-to-bottom) convention and prepended with corresponding device type prefix; e.g., m_xcell.x4.x32.m12. The separator character used between the device type prefix and the actual hierarchical name can be changed using the -opt outprefc parameter.

-opt outprefc

-opt outprefc <prefix character>|none> (default: '_')
-opt outprefc specifies a separator character, which is used in extracted device names between device type prefix and the hierarchical name of the device. Refer to the -opt outnf parameter above for additional information. -opt prefcc has no effect in modes other than -opt outnf prefix.

-opt outhierc

-opt outhierc <separator character> (default: '.')
-opt outhierc specifies instance name separator character used in hierarchical names in the output files.

-opt outsubc

-opt outsubc <separator character> (default: '@')
-opt outsubc specifies the subscript separator character used in fingered device names in output files.

-opt serial

-opt serial <any|flip|none> (default: any)
-opt serial controls processing of parallel-serial transistor groups. any allows transistors in serial chains to be permuted in an arbitrary manner. flip requires that transistors in all serial chains are arranged in the same order, but some chains can be reversed. none disables processing of serial transistor chains completely.

-opt capnet

-opt capnet <empty|node|pins> (default: node)

-opt capnet controls the format in which single-ground-capacitor nets are written to the DSPF file.

empty mode

In empty mode only *|NET statements with corresponding capacitance values are written. Neither net pin, sub-nodes, nor any other devices are reported for the net.

node mode

The node mode writes a single grounded capacitor element to DSPF, with no net pin/sub-node list, in addition to the information written in the empty mode.

pins mode

The pins mode creates a complete net pin/sub-node list for each single-ground-capacitor net in addition to the information written in the node mode.

-opt dupcc

-opt dupcc on/off (default: off)

-opt dupcc specifies whether coupling capacitors connected between two parasitic nets should be reported in the resultant DSPF file as follows:

on

Both nets.

off

Default; only one net.

-opt rpref

-opt rpref <prefix string> (default: "_")

-opt rpref specifies the prefix used in the resultant DSPF file in parasitic resistor names. Resistor names are built as follows: r<prefix><numerical index>.

-opt ccpref, -opt gcpref

-opt ccpref <prefix string> (default: "_c")

-opt gcpref <prefix string> (default: "_g")

-opt ccpref and -opt gcpref specify prefaces used in the resultant DSPF file in coupling an ground capacitor names respectively. Capacitor names are built as follows: c<prefix><numerical index>.

-opt vsr

-opt vsr <resistance value> (default: 1e-3)

-opt vsr specifies the very small resistance (VSR) value used to implement connectivity whenever there is a need to introduce an additional net or instance pin in a DSPF file. For example, if two nodes that would be essentially shorted are required, then the resistor of this small value between the two nodes will be generated.

References

- [1] A Cadence® developed format based on SPF. DSPF is similar to Spice but includes comments and a structure making it easier to organize netlist information into the original circuit with additional information used by RC trees.
- [2] SPEF has extended capability and a small format compared to SPF and is defined in the Delay-Calculation-System (DCS) standard of the Open Verilog International.

Feedback

Chapter 11: Postlayout Acceleration (PLX), Structured Backannotation (SBA), and SP2DSPF Utility References

Simulation Output

Describes HSIM output formats and output control statements.

- [HSIM Output Formats](#)
- [Output Control Statements](#)
- [Equation Evaluation](#)
- [Continuous Measurement](#)
- [Jitter and Histogram Report](#)
- [.FOUR Statement](#)
- [.FFT Statement](#)
- [Print Windows](#)
- [Full-Chip Probing](#)

HSIM Output Formats

This chapter describes HSIM output formats. The [HSIMOUTPUT](#) on page 119 command sets the desired output format. For a list of the HSIM output commands, see [Output Control Commands](#) on page 40.

Chapter 12: Simulation Output

HSIM Output Formats

The available formats are listed in [Table 67](#). The full descriptions and limitations of the formats are defined in the following sections. Click on the Format Type to view the descriptions.

Table 67 HSIM Output Formats

Format Type	Supported Viewer
FSDB Output Format	Binary format supported by the nWave waveform display tool. This is the HSIMOUTPUT default.
Nassda Output Format	NOF is a unique text output format in HSIM.
WDF Output Format	Binary format supported by the Sandwork SPICE Explorer waveform display tool.
WSF Output Format	A Cadence® format that stores output waveform data.
.out ASCII Output Format	ASCII format similar to a combined control and data file.
PSF Output Format	psfbin is a binary file for Cadence waveform viewer.
PSF Float Output Format	psfbinf is a binary-float file for Cadence waveform viewer.
UTF Output Format	Binary file for Veritools waveform viewer.
rawfile Output Format	ASCII file for the Berkeley SPICE output format.
Simultaneous Multiple Output Files	Multiple output files can be produced with the use of ampersand (&) symbol between the format specifiers.

FSDB Output Format

FSDB is a binary format supported by the nWave™ waveform display tool. Since the fsdb is a binary file, the file size is typically half the size of an ASCII output file. Additionally, a binary file is processed more efficiently by the nWave waveform viewer. Signal values are stored as floats by default. In order to store the values as double, set [HSIMFSDBDOUBLE](#) on page 90.

Nassda Output Format

The Nassda Output Format (NOF) is a unique text output format in HSIM. This output format is set when the control parameter HSIMOUTPUT is set to nassda.

Example 36

```
.param HSIMOUTPUT=nassda
```

This output format consists of two files: a control file (.ctr) and a data file (.out). Following is the Backus Naur Form (BNF) of the two files.

<pre><nassda_ctrl_file> <ctrl_settings> <signal_list> <comments> <setting> <keyword> <signal> <nassda_out_file> <time_slot> <time_line> <data_line> <time> <id> <value> <name> <EOL></pre>	<pre>::=<ctrl_settings><signal_list> ::=(<comments> <setting>)* ::=(<comments> <signal>)* :=# <anything> <EOL> ::=<keyword> <value> <EOL> :=timescale vscale iscale ::=<id> <name> <EOL> ::=(<time_slot>)* ::=<time_line> (<data_line>)+ ::=<time> <EOL> ::=<id> <value> <EOL> :=integer :=integer :=double :=string :=end-of-line</pre>
--	--

A utility program hs2tbl is provided with HSIM software. The program converts NOF to tabular data output that can be displayed with either of the following:

- Xgraph In the Xwindows environment
- Dplot In the Windows NT environment.

Chapter 12: Simulation Output

HSIM Output Formats

Example 37 Nassda Control File

```
#hsim plot control file
#Version 6.0
timescale 1
vscale 0.1
iscale 0.001
0 v(bl0n)
1 v(en)
2 v(bl0)
3 v(vddx)
4 v(bl5n)
5 v(enb)
6 v(gndx)
7 v(io0)
8 v(rd)
9 v(wr)
10 v(pch)
11 v(inio0)
12 i(vsu)
```

Example 38 Nassda Data File

```
#hsim plot file
#Version 6.0
0 15
1 0
2 15
3 17.6646
4 15
5 30
6 13.2823
7 2.81737e-007
8 0
9 0
10 0
11 0
12 -0.121817
2000
12 -0.124846
4000
12 -0.00316119
4800
10 0
11 0
12 -0.00106606
4806
0 14.9968
2 14.9968
4 14.9968
7 2.81737e-007
10 0.99
11 0.15
```

WDF Output Format

WDF is a binary format supported by the Sandwork SPICE Explorer waveform display tool. HSIM generates an .wdf output file in WDF format if **HSIMOUTPUT** is set to wdf.

```
.param HSIMOUTPUT=wdf
```

Simultaneous Multiple Output Files

Multiple output files can be produced with the use of ampersand (&) symbol between the format specifiers.

Chapter 12: Simulation Output

HSIM Output Formats

To produce FSDB and NOF formatted output files, set the following parameter:

```
.param HSIMOUTPUT="fsdb&nassda"
```

The FSDB format can be combined with any other format or formats. Certain combinations of the output formats are not allowed such as:

- out&nassda: Because both formats produce .out file.
- nassda&wsf: Because both formats use the COI interface.

WSF Output Format

WSF is a Cadence format that stores output waveform data. HSIM generates a .wsf output file when HSIMOUTPUT is set to wsf.

```
.param HSIMOUTPUT=wsf
```

.out ASCII Output Format

The .out format is an ASCII format similar to a combined nassda control and data file. HSIM generates a .out file if HSIMOUTPUT is set to out.

```
.param HSIMOUTPUT=out
```

PSF Output Format

psfbin is a binary file for the Cadence Waveform viewer.

```
.param HSIMOUTPUT=psfbin
```

Note: If the PSF binary output file size exceeds 2GB, the output file size increases even when you also use the HSIMOUTPUTTRES command. To maintain the effect of HSIMOUTPUTTRES with psf binary output format, specify HSIMOUTPUT=psfbin2 (instead of psfbin). However, doing so imposes a 2GB file size limit.

You can generate multiple output formats. For example:

```
.param HSIMOUTPUT=psfbin51&&psfbin
```

Generates both PSF V5.1.4 and V6.1 output formats.

```
.param HSIMOUTPUT=psfbin51&wdf
```

Generates both PSF V5.1.4 and wdf output formats.

PSF Float Output Format

psfbinf is a binary file for Cadence Waveform viewer. HSIM generates PSF binary format when HSIMOUTPUT set to psfbinf.

```
.param HSIMOUTPUT=psfbinf
```

UTF Output Format

utf is a binary file for Veritools' Waveform Viewer. HSIM generates utf binary format when HSIMOUTPUT set to utf. It uses version 2007.1.2 of the Veritools UTF API.

```
.param HSIMOUTPUT=UTF
```

HSIM searches for the lib<out_format>.so file in the following directories, in the order shown:

- Run directory
 - \$HOME directory
 - \$HSIM_HOME/platform/<port>/bin
 - LD_LIBRARY_PATH on Solaris and on Linux, and SHLIB_PATH on HP
- <output_format> is the value of HSIMOUTPUT and all the letters in <output_format> are in upper case.

Note: Please contact Synopsys to obtain the latest libUTF.so library.

rawfile Output Format

rawfile is an ASCII file for the Berkeley SPICE output format. HSIM generates raw format when both HSIMOUTPUT is set to fsdb and HSIMOUTPUTTBL is set to rawfile.

Example

```
.param HSIMOUTPUT=fsdb
.param HSIMOUTPUTTBL=rawfile
```

Output Control Statements

The output control statements include the following:

.print/.probe/.plot/.graph

Prints the output. Each of the statements is indistinguishable from each other in HSIM.

.lprint

Prints the logic waveforms of output data.

.print window

Prints output data only within the specified time window.

.store/.restore

Saves the intermediate simulation data and restores the saved simulation data.

.measure

Prints the results of user-specified defined analysis.

.four

Prints the results of Fourier analysis.

.fft

Prints the results of Fast Fourier analysis.

Syntax and descriptions are shown in the following sections.

PRINT/.PROBE/.PLOT/.GRAPH Statements

HSIM produces output waveforms from the following statements:

- .print
- .probe
- .plot
- .graph

All these statements are treated in the same way in HSIM.

The .print statement can be placed either at the top-level netlist or within any subcircuit definition. If it is located within the subcircuit definition, the scope of printout only covers node voltages or element currents within that particular subcircuit.

HSIM does not support the HSPICE command .option post. This command dumps all node voltage waveforms, all current waveforms of independent voltage sources and waveforms from .print and .probe statements. HSIM does however provide similar output commands as described below.

Note: Unless an ASCII output mode is specified, HSIM only produces a waveform output. HSIM does not produce SPICE .print type tabular format.

Note: The syntax for .print is also valid for .probe, .plot, and .graph.

Note: The syntax for .print <file=*name*> and <format=*name*> must be specified together. Using one or the other alone will be ignored. However, multiple formats can be specified by repeatedly specifying format=*name*.

Syntax

```
.print <tran> <name1=>ov1 <<name2=>ov2 ... >
      <subckt=sub_name> <level=val2> <matchport=val3>
      <outputres=val4> <adonly=1> <branch=name> <file=name>
      <format=name>
```

The output variables ov1 and ov2 specify the type of outputs. Supported output types are shown in [Table 68 on page 411](#).

Table 68 Output Variable Types

Types	Description
v(node_name)	Prints the node voltage waveform of the specified node_name. In case the node_name contains asterisk (*) wildcard character, HSIM will print all the voltages of matched nodes. In addition to the asterisk (*) wildcard character, the simulator also supports the single question mark (?) wildcard character so that the waveforms of all the matched nodes are printed.

Chapter 12: Simulation Output

Output Control Statements

Table 68 Output Variable Types (Continued)

Types	Description
i(element_name)	Prints the branch current waveform through the element element_name. The element name may contain the asterisk (*) wildcard character and/or question mark (?) wildcard character.
v(node1, node2)	Prints the difference v(node1)-v(node2). No wildcard character is allowed in either node1 or node2.
par('expr')	Prints the functional value of the expression expr. No wildcard character is allowed in the expression expr.
X(subckt_node_path)	Prints the current flowing into the port of the specified subcircuit. The name specified by subckt_node_path contains combined information of hierarchical_path_name.vname; the full hierarchical path name of the subcircuit is specified by hierarchical_path_name; vname denotes the node name of a port of the subcircuit or the name of a global node which connects to elements within the subcircuit. A wildcard can not be used in the vname but can be used in a hierarchical path name. This feature allows printing the subcircuit block current of the specified subcircuit instance. When HSIMXSW is set, the total switching current of the subcircuit that is connected to the port is reported.
in(node_name)	Prints the node current waveform flowing into the specified node_name. The node current is the sum of currents leaving from a node to a device and the currents entering to a node from a device. The node_name can contain the wildcard characters * or ? to match nodes by pattern. Note: This node current printout may be very CPU-intensive, especially for high connectivity nodes like VDD. Therefore you should use it discreetly. For high connectivity nodes, it is better to use an element current printout such as i(VVDD). Wildcard characters such as an asterisk (*) should not be used for all nodes in a netlist. Refer to the interactive command ni on page 494 for a node current description.

Table 68 Output Variable Types (Continued)

Types	Description
isw(element_name)	Prints the total switching current connected to the element. The element must be a voltage source connected to ground. Switching current is defined as the positive current that is charging either a node to grounded capacitor or a coupling capacitance, the discharging current is not considered.

Optional Settings

The optional settings of the matchport and level parameters are specified to control the scope of wildcard match. The subckt setting allows printouts for all instances of the specified subcircuit name. Optional settings are shown in [Table 69 on page 413](#)

Table 69 Optional Settings for .Print/.Probe/.Plot/.Graph Statements

Setting	Description
subckt=sub_name	Prints the node voltage(s) and/or element current(s) of the output variable(s) specified in the same .print statement. The printout applies to the specified node name(s) and/or element name(s) within all instances of the specified subcircuit name. This subckt setting is equivalent to placing the .print statement within the subcircuit definition.
level=val2	This setting is effective only when the wildcard character is specified in the output variable. The level value val2 specifies the number of hierarchical depth levels when the wildcard node/element name matches. <ul style="list-style-type: none"> ▪ When val2 is set to 1, the wildcard match applies to the same depth level where the .print statement is located. ▪ When val2 is set to 2, it applies to the same level and to one level below the current level where .print is located. ▪ When val2 is set to -1, the wildcard match applies to all the depth levels below and including the current level of .print statement. ▪ The default value of val2 is -1.

Chapter 12: Simulation Output

Output Control Statements

Table 69 Optional Settings for .Print/.Probe/.Plot/.Graph Statements (Continued)

Setting	Description
matchport=val3	This setting is effective only when the wildcard character is specified in the output variable and the output variable type is voltage, i.e. v(node_name). <ul style="list-style-type: none"> ▪ When val3 is set to 1, the wildcard matching extends to match the port nodes of the subcircuit instance name. ▪ When val3 is set to 0, the port nodes are excluded from the wildcard match. ▪ The default value of val3 is 0 (zero).
outputres=val4	outputs=val4 allows specifying the output resolution for all the print items listed in the .print statement. If outputres is not specified, values from the global parameters HSIMOUTPUTVRES on page 123 and HSIMOUTPUTIRES on page 121 will be used.
name1, name2	The optional names name1 and name2 define the output variables ov1 and ov2 as name1 and name2, respectively. A wildcard character is not allowed in the output variable expression when it is redefined for another name.
adonly=1	If adonly is specified as 1, then only the nodes that connect to at least one active device will be printed. The setting is effective only when the wildcard character is specified in the output variable and the output variable type is voltage, i.e. v(node_name).
filter=pattern	When printing node voltage(s) and/or element current(s) that are specified by wildcard patterns such as: .print v(x1.x2.*), nodes/elements that match the pattern specified in the filter clause will not be printed. Each filter applies to all wildcard voltages/currents being printed on the .print statement. For example: .print v(x1.x2.*) i(x1.x2.*) filter='x1.x2.n*' filter='x1.x2.a*' This syntax example will print the voltages of all nodes in subckt x1.x2 that do not start with n or a, and the current of all elements in subckt x1.x2 that do not start with either n or a.

Table 69 Optional Settings for .Print/.Probe/.Plot/.Graph Statements (Continued)

Setting	Description
branch=name	<p>To print an entire branch in the circuit, simply write .print branch='branch_name'. No output variable is required when using the branch option. Wildcard characters are not supported in the branchname:</p> <p>.print branch='x1.x2.x3' is equivalent to the following:</p> <pre>.print v(x1.x2.x3.*) .print v(x1.x2.*) .print v(x1.*)</pre>
file=name format=name	<p>The file=and format=options allows users to direct results of the current print statement to a specified file. In order for this feature to be activated, you must specify both the filename with file=and the format name with format=. Also, note that the formats specified with format= does not necessarily have to be specified with the HSIMOUTPUT parameter. For example:</p> <pre>.param HSIMOUTPUT=fsdb .print v(n1) file=foo format=out .print v(n2) file=bar format=wdf .print v(n3)</pre> <p>In this example, v(n1) will be written to foo.out, v(n2) to bar.wdf, and v(n3) to hsim.fsdb (since that is the default output file according to the HSIMOUTPUT parameter). Note that now for this feature to be activated, you must specify BOTH file=and format=. If either one is missing, the results will be printed to the default file.</p>

Examples

The following are optional setting examples.

Feedback

Chapter 12: Simulation Output

Output Control Statements

```
.subckt samp .....
.print v(x1.*) v(q*)
.....
.ends
```

Prints node voltages for all nodes which match x1.* and q* in all instances of subcircuit samp.

```
.print v(x1.*) v(q*) subckt=samp
```

Same as the first example which places the .print statement within the subcircuit definition of samp.

```
.print v(*)
```

Prints all node voltages of the circuit.

```
.print v(*) level=1
```

Prints all top-level node voltages.

```
.print v(x1.*) level=3
```

Prints node voltages within subcircuit instance x1. The printout includes nodes within x1, and the nodes in two levels below x1. The nodes located more than two levels deeper than the level of instance x1 are excluded.

```
.print v(x1.*) level=3 matchport=1
```

Prints all the node voltages as in the previous example and each port node voltage of instance x1.

```
.print diff=v(x1.a, x2.b)
```

Prints the node voltage difference v(x1.a)-v(x2.b). The variable diff is defined as the node voltage difference.

```
.print i(mp1) subckt=latch
```

Prints the branch currents for element name mp1 in all instances of subcircuit latch.

```
.print tran I_cur=par('I(x1.mp1) + I(x2.mp2) - I(x1.x2.mp1)')
```

Prints I_cur which is defined as the functional value of the following:

```
I(x1.mp1) + I(x2.mp2) - I(x1.x2.mp1).
.print wrong=v(x1.*)
```

This print statement will be ignored. The wildcard is not allowed in case the output variable is redefined for another name.

```
.print x(x1.x2.vcc)
```

Prints the subcircuit block current entering the vcc port of the subcircuit instance x1.x2.

```
.print V(x1.* adonly=1)
```

Prints all the nodes, inside the instance x1, that are connected to at least one active device.

```
.print in(n*)
```

Prints current waveform of all nodes that match n*.

```
i1(m1)=50u:node->device  

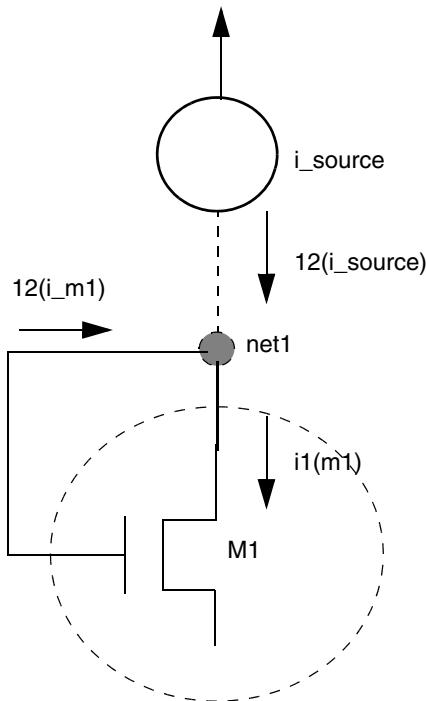
i2(m1)+-10u:device->node  

i2(i_source)=-40u:device->node  

.print in(net1)
```

In this example, for the M1 device, the total current is equal to the current leaving from net1 to device M1 plus the current entering net1 from device: $M1 = i1(m1) + i2(m1) = 50u + (-10u) = 40u$.

So the `print in(net1)` statement prints the 40u as the current value for the node, net1:



Print Average and RMS Currents

For the printout of currents or block-level currents, HSIM also supports the printout of their average and RMS values. The average results will be written to a file that uses the output prefix plus the avg suffix. For instance, if the output prefix is output, the average current values will be written to the file named output_avg. RMS results will be written to a file using the output prefix plus the rms suffix. For example, if the output file name is output, the RMS current values will be written to the file named output_rms.

```
.print <tran> <name=>ov window=time_win start=time1  
stop=time2 step=time_step
```

ov

Specifies the output variable. The eligible types are i(element_name) and x(subckt_node_path) only. See the descriptions stated in [Table 68 on page 411](#) for the usage on i(element_name) and x(subckt_node_path).

Note: Only one output variable is allowed in each statement.

name

Defines the output variable ov as name. If name is used, wildcard characters are not allowed for the output variable, ov.

window=time_win start=time1 stop=time2 step=time_step

The current waveform is averaged and RMSed over the time interval specified by the parameter window. The first calculation begins at the time specified by start and covers the time interval between start-window/2 and start+window/2. The calculations repeat at the increment specified by step and ends at the time specified by stop.

Example

```
.print i(m1) window=6n start=30n stop=100n step=10n
```

Prints the average and RMS branch current for element name m1. The time window for first calculation is from 27 ns to 33 ns (the centered start time is 30 ns) and repeats at 10 ns. The last calculation centered at 100 ns.

```
.print x(xadd.vdd) window=10n start=10n stop=70n step=20n
```

Prints the average and RMS subcircuit block current entering the V_{DD} port for the instances xadd. The time window for first calculation is from 5 ns to 15 ns (the centered start time is 30 ns) and repeats at 20 ns. The last calculation centered at 70ns.

```
.print in(m2) window=10n start=5ns stop=100n step=20n
```

Prints the average and RMS current for node name m2. The time window for first calculation is from 20ns to 30ns (the centered time is 25ns) and repeats every 20ns. The last calculation time centered at 85ns.

.lprint Statement

HSIM generates logic waveforms from .lprint statements. The signal can be 1/0/U. If the signal is used to cover Hi-Z(H) and Lo-Z(L), HSIMHZ=1 must be set. The default value is 0. The .lprint statement syntax is similar to .print.

```
.lprint <tran> <vlth> <vhth> <name1=>ov1 <<name2=>ov2 ....>
<subckt=sub_name> <level=val2> <matchport=val3>
<adonly=1>
```

vlth

Threshold voltage for LOW logic state. The 0 (zero) state is printed if the node voltage is less than or equal to vlth. If the node voltage is between vlth and vhth, the X state is printed. Its default value is the value given by the HSIM global parameter [HSIMVLTH on page 195](#).

vhth

Threshold voltage for HIGH logic state. The 1 (one) state is printed if the node voltage is higher than or equal to vhth. If the node voltage is between vlth and vhth, the X state is printed. Its default value is the value given by the HSIM global parameter [HSIMVHTH on page 194](#).

ov1, ov2

Specifies the output variables in the format of v(node_name). In case the wildcard contains wildcard characters, '*' or '?', HSIM will print the logic states of all of the matched nodes.

<name1>, <subckt>, <level>, <matchport>, <adonly>

Optional settings; see .PRINT/.PROBE/.PLOT/.GRAPH statements.

Note: If vlth and vhth are not specified in the .lprint statement and [HSIMVLTH on page 195](#) and [HSIMVHTH on page 194](#) are not explicitly specified in the netlist, HSIM will issue a warning message and the 30% and 70% of supply voltage thresholds, defined by [HSIMVDD on page 191](#), will be used. If a design

Chapter 12: Simulation Output

Output Control Statements

involves multiple power supplies, use [HSIMAUTOVDD](#) on page 64 to automatically set HSIMVLTH and HSIMVHTH in accordance with the circuit's supply voltage.

Example

```
.subckt samp .....  
.lprint 1 3 v(*)  
.ends  
.lprint v(x1.n1) v(x1.x2.*)
```

This example prints all the node logic waveforms for all nodes in each instance of subcircuit samp with LOW threshold voltage of 1V and HIGH threshold voltage of 3V, the logic waveform at node x1.n1, all node logic waveforms within the subcircuit instance x1.x2.

.STORE/.RESTORE Statement

The .store statement saves the intermediate simulation data. The .restore statement restores the saved data file. See the following sections:

- [.STORE](#)
- [.RESTORE](#)

.STORE

```
.store <file=simul_file> <time=time1 <time=time2 ...> >  
      <repeat=time_interval> <split=1>
```

If no simul_file name is specified, then the default file name is <output_prefix>.iic. If the repeat parameter value is specified, then only the first time value, time1, is effective, and the store operation is performed in every time_interval. The default time is 0. split=1 will split the intermediate simulation data into multiple files with time1 and time2 suffixes. time1 and time2 will be converted into pico seconds to become the suffix of the file name.

The time point to be stored is affected by TPERIOD, where TPERIOD = ([HSIMTAUMAX](#) / [HSIMTIMESCALE](#)). For example, if you specify the following syntax:

```
.store time=t1 time=t2...
```

If t2 is greater than or equal to TPERIOD, the next stored time point, t2, is used to store simulation data. If t2 is less than TPERIOD, a warning message is printed in the log file.

Example

```
.param HSIMTAUMAX=1n
.store file=store_db time=11n time=12n
```

Because TPERIOD=1n (TPERIOD=1n/1=1n) in the previous example, the next stored time, t2=12n is valid and is used to store simulation data.

```
.param HSIMTAUMAX=2n
.store file=store_db time=11n time=12n
```

Because TPERIOD=2n (TPERIOD=2n/1=2n) in the previous example, the next stored time should be t2=11n+2n=13n. So the specified stored time, 12n, becomes invalid. In this situation, a warning message is written in the log file:

Warning: cannot schedule to save simulation at 12ns (too close to last save...)

```
.store file=simul_file1 time=123n repeat=1000n
```

The simulation data are stored at time being 123 ns, 1123 ns, 2123 ns, and so on.

.RESTORE

```
.restore <outformat=file_format> <file=simulation_file> <time=time1>
<outfile=wave_file>
```

The filename information is required for the .restore statement. The keyword file= is optional. If the time parameter information is not specified, then the most recently saved simulation data in the saved file will be restored. If the outfile parameter is specified, the waveform for each output variable from the wave_file will be added to the output. The fsdb and wdf output formats are currently supported. The outformat parameter specifies the waveform format to be added. The default for outformat is fsdb. The simulation file can be a full path file name or a file prefix.

Caution: When using output to restore, do not use the same -o <prefix> used in the previous simulation. Using the same prefix WILL corrupt the save file as well as the previous .fsdb file.

Examples

```
.restore file=simul_file1 time=1000n
```

The simulation data are restored at time 1000n from the previously stored simulation data file, simul_file1.

Chapter 12: Simulation Output

Output Control Statements

```
.restore outformat=wdx file=simul_file1 time=1000n  
outfile=test.wdf
```

The simulation data are restored at time 1000n from the previously stored simulation data file, simul_file1. Also, the waveform for each output variable from test.wdf will be added to the output waveform file in .WDF format.

.measure Statement

Measurement output format can be controlled by the following option parameters:

```
.option ingold=x  
.option measdgt=x
```

Table 70 shows the command syntax and output descriptions for ingold and measdgt.

Table 70 Command Descriptions

Syntax	Output Description
.OPTION AUTOSTOP	Terminates the simulation after completing all .measure statements. In HSIM, the support of this feature is limited. Measure types supported: Find and When Measurements; Trig and Targ Measurements; Equation Evaluation for temporary output variables only. Limitation: there should not be any dependencies among measurement statements. Also, waveform comparison is not supported.
.OPTION INGOLD=0	HSIM print values in engineering format as follows: 1g=1e9, 1x=1e6, 1k=1e3, 1m=1e-3, 1u=1e-6, 1n=1e-9, 1p=1e-12, 1f=1e-15
.OPTION INGOLD=1	HSIM print values in engineering format for values between 0.1 and 999. Any other values will be printed in exponential format.
.OPTION INGOLD=2	[default] HSIM print values in exponential format
.OPTION MEASDGT=6	HSIM will print values with 6 decimal digits.
.OPTION MEASDGT=12	[default]

HSIM supports the .measure statement. The output variables which can be measured by the .measure statement are node voltage v(n1), voltage difference v(n1,n2), branch current i(m1), and any .measure result variable. The following measurement types are supported. All .measure results are saved in the file out_file.mt.

For information about out_file, refer to [Chapter 5, Running HSIM](#).

Rise, Fall, and Delay Time Measurements

.measure <tran> result trig ... targ ...

trig

Trigger. The syntax for trig is:

```
trig trig_var val=val2 <td=td1> <cross=val3> <rise=val4>
<fall=val5>
trig at=val6
```

targ

Target. The syntax for targ is:

```
targ targ_var val=val2 <td=td1> <cross=val3|last>
<rise=val4|last> <fall=val5|last>
```

Arguments for trig and targ

cross

The syntax for cross is:

```
cross=val3, rise=val4, and fall=val5
```

In the syntax for cross=val3, rise=val4, and fall=val5, the val3, val4, and val5 values can be either numbers or the keyword last. A number indicates which occurrence of a cross, rise, or fall event is used. If the keyword last is used, the last occurrence of a cross, rise or fall event is used.

```
fall/rise/cross=-1, -2 ... (in addition to 'last')
```

td

The syntax for td is:

```
td=td1
```

td=td1 is the amount of simulation time that must elapse before a measurement is performed. The number of cross, rise, or fall occurrences is counted after the required td1 simulation time.

Chapter 12: Simulation Output

Output Control Statements

val

The syntax for val is:

`val=val2`

`val=val2` is the value of a target or trigger variable at which the cross, rise, or fall counter is incremented by one.

`trig_var and tart_var`

The syntax for trig_var and tart_var is:

`trig_var` and `tart_var` are the output variable names for trig and targ.

Example

```
.measure tran tw0 trig v(dep) val=1.5 fall=4 targ v(ffarb) val=0.5
td=10n rise=1
.measure <name> when v(<name2>)=<val> rise=<-1/-2/last>
from=<val> to=<val>
```

Derivative Measurements

Derivative measurements use one of the following three syntax statements:

```
.measure <tran> result derivative var at=val
.measure <tran> result derivative var1 when var2=val1
  <td=td1> <rise=val2|last> <fall=val3|last>
  <cross=val4|last>
.measure <tran> result derivative var1 when var2=var3
  <td=td1> <rise=val1|last> <fall=val2|last>
  <cross=val3|last>
```

Example

```
.measure tran slope derivative v(n) at=10n
```

This calculates the derivative of v(n) when transient analysis is at 10ns.

```
.measure tran rise_tr derivative v(n) when v(n)='VDD/2' rise=2
```

This calculates the derivative of v(n) when v(n) reaches half of the VDD voltage level at 2nd rising edge.

```
.measure tran cross_tr derivative v(n) when v(n1)=v(n2)
```

This calculates the derivative of v(n) when both node voltages of n1 and n2 are equal.

Average, RMS, MIN, MAX, Peak-To-Peak Measurements

.measure <dc> result func out_var1 from=time1 to=time2

func can be one of the following reserved words.

avg

Average

max

Maximum

min

Minimum

pp

Peak-To-Peak

rms

Root Mean Square

integ

Integral

Example

```
.measure avg_ivdd avg i(vdd) from=10n to=250n
```

Find and When Measurements

```
.measure <tran> result when out_var1=val2 <td=td1>
    <rise=val3|last> <fall=val4|last> <cross=val5|last>
.measure <tran> result when out_var1=out_var2 <td=td1>
    <rise=val3|last> <fall=val4|last> <cross=val5|last>
.measure <tran> result find out_var1 when out_var2=val2
    <td=td1> <rise=val3|last> <fall=val4|last>
    <cross=val5|last>
.measure <tran> result find out_var1 when out_var2=out_var3
    <td=td1> <rise=val3|last> <fall=val4|last>
    <cross=val5|last>
.measure <tran> result find out_var1 at=val2
```

Refer to the examples in [Equation Evaluation on page 426](#).

Equation Evaluation

```
.measure <tran> result param='expr'
```

Example

```
.measure teq when v(rb)=v(rt)
```

```
.measure veq find v(eq) when v(rb)=0.5 fall=3
```

```
.measure eq_sum param='teq + veq'
```

.measure statement 3 above evaluates the expression that is a function of results from .measure statement(s) 1 and 2. The expression for param can not be a function of branch current(s) or node voltage(s).

Continuous Measurement

The continuous measurement feature of HSIM simulator is used by specifying the tran_cont option in the .measure statement. This type of measure will perform the specified measurement continuously until the end of simulation or whenever the measurement condition is fulfilled.

```
.measure tran_cont result trig ... targ ...
.measure tran_cont result when out_var1=val2 <td=td1>
    <rise=val3|last> <fall=val4|last> <cross=val5|last>
.measure tran_cont result when out_var1=out_var2 <td=td1>
    <rise=val3|last> <fall=val4|last> <cross=val5|last>
.measure tran_cont result find out_var1 when out_var2=val2
    <td=td1> <rise=val3|last> <fall=val4|last>
    <cross=val5|last>
.measure tran_cont result find out_var1 when
    out_var2=out_var3 <td=td1> <rise=val3|last>
    <fall=val4|last> <cross=val5|last>
```

A specific measure output file will be created, <output_prefix>_<result>.mt, to store the continuous measurement results.

Example

```
.measure tran_cont cont_vout1 find v(out1) when v(a1)=2.5 fall=1
```

.measure statements will continuously find the voltage out1 when the voltage value of node a1 reaches to 2.5 starting from the first falling edge. In this

example, the additional output filename is hsim_cont_vout1.mt if the output file specifier is not used. hsim_cont is the output_prefix.

```
.measure tran_cont cont_vout1 when v(a1)=2.5 fall=2
```

The .meas statement will continuously report the time when the voltage value of node a1 reaches 2.5V, starting from the second falling edge.

```
.measure tran_cont cont_va1_pulse trig v(a1) val=2.5 rise=1 targ v(a1) val=2.5 fall=1
```

The .measure statement will continuously measure the pulse width of node a1. The output filename will be hsim_cont_va1_pulse.mt. hsim_cont is the output prefix.

Jitter and Histogram Report

Jitter specifications are of interest in PLL design. By using continuous measurement, each cycle period after lock-in can easily be measured. With the jitter parameter in the continuous measurement command, HSIM reports the jitter value and a histogram to show whether the design is within the specifications. In jitter=nb, nb is the number of bins.

A jitter histogram is evenly divided into nb between the minimum and maximum results. Both the histogram and jitter will be stored in the <output_prefix>_<result.mt.hist> file. [Example 39](#) illustrates the example syntax used to produce the associated histogram.

Example 39

```
.measure tran_cont td trig v(pllclk) val=0 td=4500ns rise=1 targ v(pllclk) val=0 td=4500ns rise=2 jitter=10
/* Add this parameter into continuous measurement command */
```

In the hsim_td.mt.hist file:

Chapter 12: Simulation Output**.FOUR Statement**

```
meas_variable=td
jitter=4.180651e-012
min=9.996815e-009 max=1.000100e-008
run_min=1 run_max=81
9.997233e-009, nb=1, freq=1.010101e-002 |
9.997651e-009, nb=2, freq=2.020202e-002 | *
9.998069e-009, nb=4, freq=4.040404e-002 | ***
9.998487e-009, nb=11, freq=1.111111e-001 | *****
9.998906e-009, nb=11, freq=1.111111e-001 | *****
9.999324e-009, nb=26, freq=2.626263e-001 | *****
9.999742e-009, nb=20, freq=2.020202e-001 | *****
1.000016e-008, nb=18, freq=1.818181e-001 | *****
1.000058e-008, nb=4, freq=4.040404e-002 | ***
1.000100e-008, nb=2, freq=2.020202e-002 | *
```

.FOUR Statement

HSIM supports Fourier transformation. Any valid voltage or current output variable can be used in the .four analysis. The results are stored in a tabular format in the .log file. In addition, one .ft file is generated for each .four output variable.

```
.four freq1 out_var1 <out_var2 ...>
```

The variable value freq1 specifies the fundamental frequency for the analysis, while the output variable(s) are indicated as out_var1, out_var2, and so on.

Example

```
.four 20Meg v(2) v(3)
```

.FFT Statement

HSIM supports Fast Fourier Transform (FFT) analysis. Any valid voltage or current output variable can be used in the FFT analysis. The results of FFT analysis are stored in a tabular format in the .log file. In addition, one .ft file is generated for each FFT output variable.

```
.fft out_var1 <start=time1> <stop=time2> <np=fft_np>
      <format=norm|unorm> <window=win1> <alfa=a1>
      <freq=freq1> <fmin=freq2> <fmax=freq3>
```

out_var1 can be any valid voltage or current output variable. Parameters start and stop specify the start time and end time of the analysis, respectively. The default value for start is 0, and default value for stop is stoptime in the .tran statement. Refer to [Chapter 8, Input Netlist, .tran on page 339](#) for details. FFT statements are described in [Table 71 on page 429](#).

Table 71 FFT Statements

Parameter	Description
alfa	Used with Gauss or Kaiser window for controlling the highest sidelobe level, bandwidth, and so on. A valid value is $1 \leq alfa \leq 20$. The default value is 3.
freq	Specifies the frequency of interest. If freq value is nonzero, the output listing is limited to the harmonics of this frequency. The default value is 0 Hz. The Total Harmonic Distortion (THD) for the harmonics is also printed.
fmin	Specifies the lower frequency boundary for the FFT output to be printed to the logfile. The default value is $1/T$ (Hz) with $T=(stop - start)$.
fmax	Specifies the upper frequency boundary for the FFT output to be printed to the logfile. The default value is $0.5 * np * fmin$ (Hz).
format	Indicates the output format. A valid value is either norm for normalized magnitude or unorm for un-normalized magnitude. The default value is norm.
np	Specifies the numbers of points used in the FFT analysis, and must be a power of 2. If the specified value is not a power of 2, HSIM will adjust it to the closest higher number of the power of 2. The default value is 1024.
window	Indicates the window type. The values of this parameter are as follows: <ul style="list-style-type: none"> ▪ rec - Simple rectangular truncation window, default value ▪ bart - Bartlett triangular window ▪ hann - Hanning window ▪ hamm - Hamming window ▪ black - Blackman window ▪ harris - Blackman-Harris window ▪ gauss - Gaussian window ▪ kaiser - Kaiser-Bessel window

Chapter 12: Simulation Output

Print Windows

Example

```
.fft v(3) fmin=0.5g fmax=2.2g window=gauss
```

Print Windows

Typically, the HSIM simulation output file size is smaller than the output file size of SPICE. Since the circuit size that can be handled by HSIM is much larger than that for SPICE simulators, it is expected that the HSIM output file size may be large for large circuits.

In addition to limiting the number of printout nodes or elements, specifying the print window(s) can also reduce the output file size. The output data is printed only when the print time is within the specified print time window(s).

You can split the output data in different time windows into different files with the `split` option.

Syntax

```
.print window start_time1 <stop_time1> <start_time2>
    <stop_time2> ... <split>>>
```

Example

```
.print window 100n 1u 4u 5u
```

The printout is limited to the time windows between 100 ns and 1 us, and between 4 us and 5 us.

```
.print window 100n 1u 4u 5u split
```

The printout for the time windows between 100 ns and 1 us, and between 4 us and 5 us, are split into two files, `*.1` and `*.2`.

Full-Chip Probing

The full-chip probing flow provides better performance in large designs running transient analysis and initializing output variables when voltage probing is enabled (current probing is not currently supported). This flow consists of two phases:

- In phase 1, all node voltages waveform data is dumped to a compact database (SDB) file during transient simulation.
- In phase 2, which takes place after simulation completes, HSIM generates the waveform files containing all or a selection of the requested signals. The last step of phase 2 is to merge the regular waveform output file from phase 1 with the phase 2 data so that the final result is a single waveform file. For performance reasons, waveform file merging is disabled by default. It can be enabled by setting [HSIMPROBEREST](#) on page 132 to 1 to enable waveform file merging.

[Figure 24 on page 432](#) shows an overview of phase 1 and phase 2.

Chapter 12: Simulation Output

Full-Chip Probing

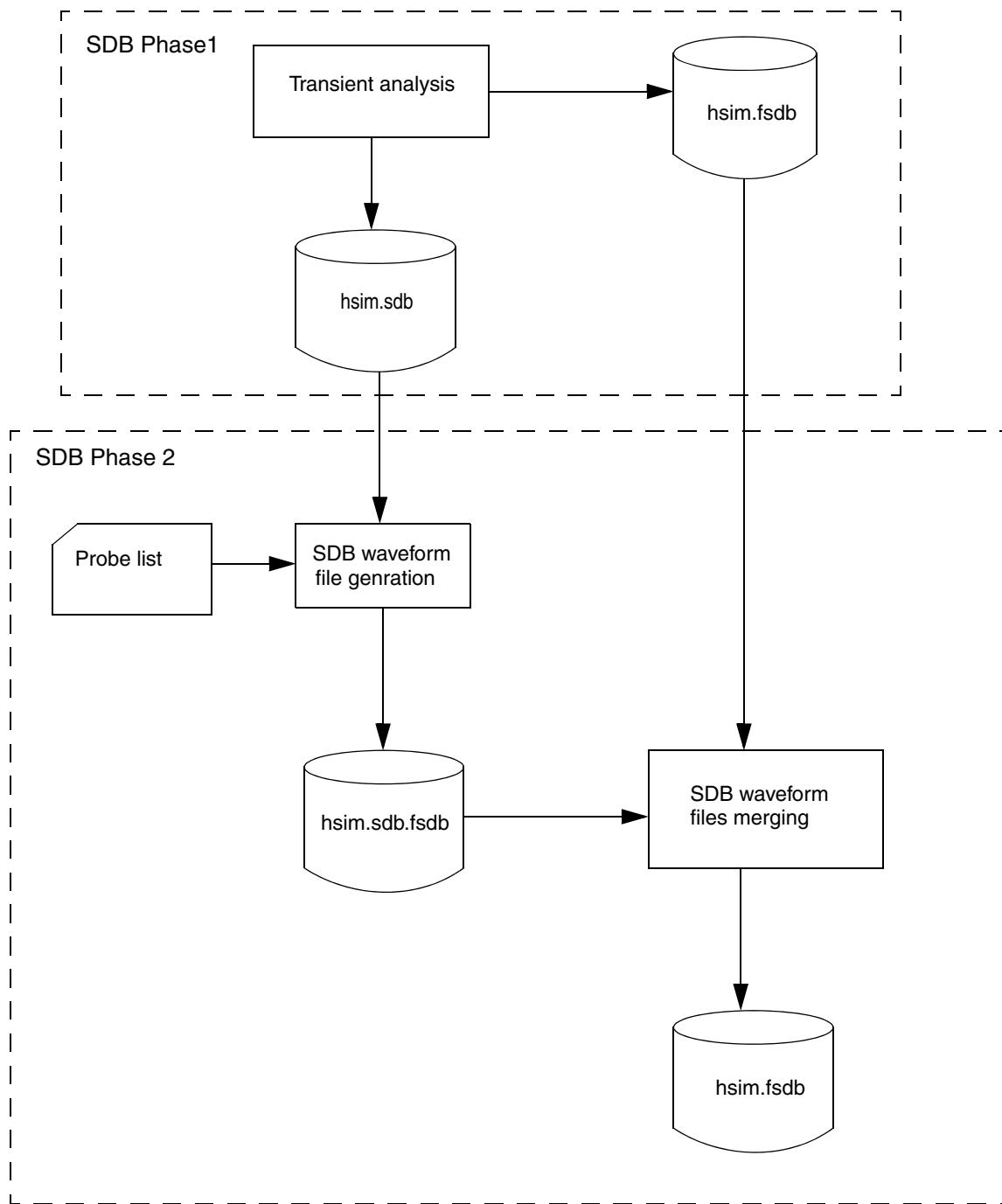


Figure 24 Overview of Phase 1 and Phase 2

By default, the waveform file generated contains signals specified in the netlist using `.print` or `.probe` statements. You can use `-probelist` command line option to specify a list of signals to be generated in phase 2.

The output file names in phases 1 and 2 are controlled by the `-o` command line option.

Printing of logical waveforms requires that thresholds be defined to convert analog signals to digital values. The full-chip probing flow supports single supply thresholds set by default based on [HSIMVDD on page 191](#), or manually using the [HSIMVHTH on page 194](#)/[HSIMVHTRATIO on page 194](#)/[HSIMVLTH on page 195](#)/[HSIMVLTH on page 195](#) commands. It does not support thresholds automatically set based on detected power supplies when you use [HSIMAUTOVDD on page 64](#).

The full-chip probing flow only supports:

- Transient analysis. This flow is automatically disabled if any other analyses (AC, DC) are specified. It is also disabled for multi-run analyses (`.alter`, `.data`, Monte Carlo).
- Regular voltages and logical waveforms. It does not support expressions or currents. All signals that are not supported are dumped to the regular waveform output file.
- The WDF and FSDB output file formats.

Controlling the Full-Chip Probing Flow

By default, phase 2 runs automatically after phase 1. To save the data from phase 1 to run phase 2 later, set the [HSIMPROBEAUTO on page 131](#) command to 0. You can run phase 2 as a standalone operation on an existing phase 1 file with the `-sdb` command line option.

When you run phase 2 as a standalone operation, you can enable the activity check feature with the `-acheckdv` command line option. For example:

```
hsim -sdb sdb_file -acheckdv deltab
```

In this example, all nodes with voltage changing by more than `deltab` from their initial voltage are reported in a file with a suffix of `.sdbacheck`.

When you run phase 2 standalone operation using the `-sdb` command line option, the default behavior is to write out all the signals that were requested by `.print` or `.probe` statements in the netlist used to generate the phase 1

Feedback

Chapter 12: Simulation Output

Full-Chip Probing

output. You can use the `-probelist` option to request any signal in phase 2. For example:

```
hsim -sdb sdb_file -probelist probelist_file
```

In this example the `.file` contains the signals to be included in the waveform file output. Voltages, logical and currents are supported. Wildcards and level scoping are supported.

Conversion Utilities

Provides details about HSIM utility commands that convert FSDB files.

In HSIM, four utility commands are available to convert the FSDB and WDF files.

- `wdf2tbl`: Converts HSIM generated, wdf simulation output format to tabular format.
- `wdfb2pwl`: Extracts given signals in the signal_file from WDF output to the piecewise linear (PWL) voltage source elements.

The `wdf2tbl` Utility

```
wdf2tbl -i wdf_file <-hn> <-stdout> <-s sig_file>
<-o out_file> <-csdf> <-start time1> <-stop time2> <-step
steptime>
```

or

```
wdf2tbl -i wdf_file <-hn> <-stdout> <-s sig_file>
<-o out_file> <-csdf> <-at time1 <time2 ...>>
```

The `wdf2tbl` parameters are described in [Table 73 on page 437](#).

Table 72 WDF2TBL Parameters

Parameter	Description
<code>-i</code>	Indicates the WDF file generated by HSIM.
<code>-s</code>	Specifies a text file to contain the signal names whose values will be converted to the tabular data format.

Chapter 13: Conversion Utilities

The wdf2tbl Utility

Table 72 WDF2TBL Parameters (Continued)

Parameter	Description
-o	Specifies a text file to contain the converted tabular data result. The default out_file name is composed of the wdf filename with the suffix .tbl.
-hn	Prints signal horizontally in the output table file.
-stdout	Prints output table on the screen.
-csdf	Indicates Common Simulation Data Format (CSDF) is the selected tabular output format.
-start	The first conversion begins at the time specified by -start. The unit of time is nanosecond.
-step	After the conversion begins, it is repeated at the increments specified by -step. User may specify multiple steps after the -step flag. The unit of time is nanosecond.
-stop	The conversion ends at the time specified by -stop. The unit of time is nanosecond.
-at	The conversion is performed for the specific time point. The unit of time is nanosecond.

Example 40 signal_file example with variables.

```

analog_unit n
analog_to_digital 0.7 2.5
time_step 20 30
precision_digit 6
hex a_8_1 a8 a7 a6 a5 a4 a3 a2 a1
spa
oct a_8_1 a8 a7 a6 a5 a4 a3 a2 a1
spa a8 a7 a6 a5 a4 a3 a2 a1
spa
v(a1)
v(xfull.o1)

```

The parameters used in the above example are shown in [Table 73 on page 437](#).

Note: These parameters only apply if –csdf is not specified.

Table 73 .WDF2TBL Example Parameter Descriptions

Parameter	Description
analog_unit	Specifies the unit. The available units are as follows:
m	milli
u	micro
n	nano
p	pico
f	femto
analog_to_digital	Specifies the low and high thresholds to convert analog values to digital representation. In the above example, 0.7 specifies the low threshold, and 2.5 specifies the high threshold.
time_step	Specifies the time steps. The conversion takes place at the specified time step values and their multiples. The selected unit of measure is nanosecond.
precision_digit	Specifies the number of precision digits of the printed values.
hex	Groups the digital signals into hex format. The syntax follows: hex group_name signal1_in_wdf signal2_in_wdf signal3_in_wdf ...
oct	Groups the digital signals into octal format. The syntax follows: oct group_name signal1_in_wdf signal2_in_wdf singal3_in_wdf ...

Chapter 13: Conversion Utilities

The wdf2pwl Utility

Table 73 .WDF2TBL Example Parameter Descriptions

Parameter	Description
spa	The keyword to insert space between signals in the tabular format. In default, there is no space between the digital signals in tabular format.

The wdf2pwl Utility

```
wdf2pwl -i wdf_file <-s sig_file> <-o out_file> <-start time1> <-stop time2>
```

The wdf2pwl parameters are described in [Table 74 on page 438](#).

Table 74 WDF2PWL Parameters

Parameter	Description
-i	Indicates the WDF file generated by HSIM.
-s	Specifies that a text file must contain the signal names whose values will be converted to PWL voltage and current source elements. Additionally, specify the node name and signal type along with the signal name if the signal name is not the preferred node name used in voltage and current source elements.
-start	Specifies the time for the first conversion to start. The unit of time is nanosecond.
-stop	Specifies the time for the conversion to stop. The unit of time is nanosecond.
-o	Specifies a text file to contain the converted piecewise linear (PWL) voltage source elements. The default out_file name is composed of the wdf filename with the suffix .pwl.

Example 41 signal_file syntax example.

```
v(a1)
i(b1)
current_i1 vdd1 i
voltage_v1 vdd2 v
```

v(a1), i(b1), current_i1, and voltage_v1 are the existing signal names in the WDF file. No additional node name and signal type are associated with v(a1) and i(b1). a1 and b1 from the signal name will be used as the node name in both voltage and current source elements. The source element for a1 will be the voltage source because of v(a1) and the source element for b1 will be the current source because of i(b1).

current_i1 is tagged with the following:

Node name

vdd1

Type

The source element generated for the current_i1 signal is the current source with the vdd1 node name.

voltage_v1 is tagged with the following:

Node name

vdd2

Type

v

The source element generated for voltage_v1 signal is voltage source with vdd2 node name in it.

Chapter 13: Conversion Utilities

The hsencrypt Utility

Example 42 out_file syntax example.

```
Va1 a1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
Ib1 b1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
Icurrent_i1 vdd1 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
IVoltage_v1 vdd2 0 pwl (
+ 0.0000e+000 0.0000e+000
+ 2.0000e-008 0.0000e+000
+ 2.0020e-008 1.0000e-001
+ 2.0040e-008 2.0000e-001
+ 2.0060e-008 3.0000e-001
+ 2.0080e-008 4.0000e-001
)
```

The hsencrypt Utility

hsencrypt is an encryption program to encrypt netlist files. Netlist encryption allows proprietary models, parameters, and circuits to be distributed without revealing any intellectual property. Recipients of an encrypted netlist can run simulations using HSIM but cannot print or query the encrypted circuit elements or nodes. HSIM can only read encrypted netlist files that are encrypted using hsencrypt.

To encrypted portion of the netlist file, insert .PROTECT and .UNPROTECT statements around text to be encrypted and then run `hs encrypt`. `hs encrypt` produces an ASCII text file in which all text that follows .PROTECT and precedes .UNPROTECT is encrypted. The syntax follows:

```
% hs encrypt <key> <netlist_file> [<output_file>]
```

`hs encrypt` requires command arguments:

Netlist file name

The netlist file is a SPICE netlist that contains .PROTECT/.UNPROTECT statements and the key is used for encryption or decryption.

Encryption key

`xxxxyyzzz` is the key used for encryption and decryption as shown in the [Example](#).

Output file name

Optional. The encrypted text will be printed to [<output_file>]. If you do not specify a file name, it will be printed to standard output.

Note: The `hs encrypt` utility will not undergo any further enhancement. Instead, support will be continued for the decryption of data that is encrypted using `meta encrypt` utility, to ensure compatibility with HSPICE.

For details on `meta encrypt`, refer to the “Library and Data Encryption” chapter in the HSPICE® User Guide: Basic Simulation and Analysis.

Example

```
% hs encrypt xxxxyyzzz models models.enc
```

Using the `xxxxyyzzz` key encrypts the portion of the `models` file specified with .PROTECT/.UNPROTECT and writes the file to `models.enc`.

Included files and library files specified using .INCLUDE and .LIB statements will not be processed by `hs encrypt` directly as shown in the following example:

```
.PROTECT  
.lib prop.mod TT  
.UNPROTECT
```

`hs encrypt` will encrypt the `.lib prop.mod TT` text but not the context of the library file `prop.mod`. If the `prop.mod` file content needs to be encrypted, insert

Chapter 13: Conversion Utilities

The v2s Utility

.PROTECT/.UNPROTECT statements inside prop.mod and run `hsencrypt` on that file.

The encrypted file can be used in .INCLUDE and .LIB statements just like any other file.

See Also

[.PROTECT or .PROT](#)
[.UNPROTECT or .UNPROT](#)

The v2s Utility

v2s is a program to convert the netlist from Verilog to SPICE.

`v2s <options> filename`

where filename is one or more Verilog input files that describe the design netlist. The options are shown in [Table 75 on page 442](#).

Table 75 V2S Parameter Descriptions

Option Parameter	Description
<code><-h[elp]></code>	Prints a short-form command usage description.
<code><-bn bus_id></code>	Specifies the bus notation. Output names are converted with the following bus symbols according to the bus_id value: 0 - [default] Expands buses like: A[0] A[1] A[2] ... 1 - Expands buses like: A<0> A<1> A<3> ... 2 - Expands buses like: A0 A1 A3 ...
<code><-const0 val></code>	Specifies voltage level for CONST0.
<code><-const1 val></code>	Specifies voltage level for CONST1.
<code><-pe></code>	Prints empty subckt definition. By default, an empty module was detected and not outputted.
<code><-top name></code>	Specifies the top-level subckt name.

Table 75 V2S Parameter Descriptions (Continued)

Option Parameter	Description
<-map file>	Specifies a name mapping table used when the subckt port mismatches.
<-mapfile file>	Automatically renames the module, node, port, and instance names to ensure syntax agreement as v2s translates a Verilog netlist to SPICE. By default, v2s outputs a mapfile to track the name changes however. If desired, the file name can be changed by specifying -mapfile filename.
<-ks>	Keeps the backslash character in names.
<-v>	Prints a program version.
<-o file>	Specifies the output file name. The default is v2s.spi.
<-s file>	Specifies the SPICE subckt definition files. It is required to output the correct port ordering if instance's port connections are referenced by name.
<-eldo>	The subckt definition file in Eldo format.
<-case 0 1>	Specifies case sensitivity. The default is 0.

Chapter 13: Conversion Utilities

The v2s Utility

Table 75 V2S Parameter Descriptions (Continued)

Option Parameter	Description
<-mapstr>	<p>Refers to characters used as instance or net names in a module that are used as comments in SPICE; i.e., the asterisk, dollar, and ampersand characters (*, \$, &), will be replaced using V2S during Verilog netlist translation with an underscore (_) character.</p> <p>If V2S discovers a pre-existing identifier with a similar name, it will modify the new string to create a unique identifier. These changes will be logged in the map file.</p> <p>V2S -mapstr option provides a user option to specify a different map string as shown in the following example:</p> <p>In the Verilog file.</p> <pre>wire *logic0*; nand U1(.a(in), .b(*logic0*), .out(out));</pre> <p>Then V2S by default will map *logic0* to _logic0_ in the xu1 in _logic0_ out nand SPICE netlist.</p> <p>Users may also select a different replacement string using the -mapstr option as follows:</p> <pre>V2S ... -mapstr _abc_</pre> <p>Then the SPICE file will appear as follows:</p> <pre>Xu1 in _abc_logic0_abc_ out nand.</pre>

Example

```
v2s top.v -s model -o top.spi -top top
```

This example will convert modules defined in top.v to subckt definitions and write output to top.spi with top-level subckt name top.

Note: Refer to the demo tutorial/v2s.

AC Small-Signal Analysis

Provides details of AC Small-signal Analysis (AC), a frequency response analysis that calculates the small-signal response of a circuit to a combination of inputs. This chapter provides details on AC analysis support for: MOSFET, BJT, JFET, Diode, Resistor, Capacitor, Inductor, Voltage and Current sources (VCVS, VCCS, CCCS, and CCVS), and Mutual Inductor.

- [Overview of AC Analysis](#)
- [AC Sources](#)
- [AC Analysis Output](#)
- [AC Analysis Measurement](#)

Overview of AC Analysis

AC Small-Signal Analysis (AC) is a frequency domain analysis that calculates the small-signal response of a circuit to a combination of inputs. This is accomplished by creating a linear solution for the circuit at its DC operating point. AC analysis has the following features:

- Nonlinear devices are transformed to linear devices around their bias point value before running an AC analysis. Examples include:
 - Voltage-controlled sources
 - Current-controlled sources
- AC analysis only considers gain and phase responses of a circuit because it is a linear analysis
- AC analysis does not limit voltages or currents.

Chapter 14: AC Small-Signal Analysis

Overview of AC Analysis

During AC analysis the simulator seeks a linear solution for the circuit in the frequency domain at the appropriate operating point. Hence, as a first step in conducting AC analysis, the operating point information is determined.

Each voltage or current source can have any or all of the following components:

- AC component
- DC component
- Transient component

For AC analysis, the transient component of the source is ignored and the DC component is used to determine the operating point. If the DC component is unspecified, the default value zero is used.

It is recommended that source magnitude be set to 1. This will ensure that the measured output equals the gain, relative to the input source, at that output.

The following elements are supported in AC analysis:

- MOSFET
- BJT
- JFET
- Diode
- Resistor
- Capacitor
- Inductor
- Voltage and Current sources
 - Voltage-Controlled Voltage Source (VCVS)
 - Voltage-Controlled Current Source (VCCS)
 - Current-Controlled Current Source (CCCS)
 - Current-Controlled Voltage Source (CCVS)
- Mutual inductor

After the operating point is determined, frequency domain analysis can begin. The following statement invokes AC analysis and specifies the required frequency range.

```
.ac sweep_type nf start stop
```

AC analysis is conducted for the frequency range between start and stop.

sweep_type

One of the following keywords:

dec: for decade increment

oct: for octave increment

lin: for linear increment

poi: for list of points

nf: number of frequencies for lin and poi types; or number of frequencies per decade for dec type, per octave for oct type.

start

Starting frequency for dec, oct, lin.

stop

Final frequency for dec, oct, lin.

Example

```
.ac poi 4 1e8 5e8 8e8 1e9
```

Note: POI (Points of Interest)

AC analysis is conducted at four different frequencies:

- 100 MHz
- 500 MHz
- 800 MHz
- 1 GHz

AC frequency analysis can be performed when sweeping some external parameter(s) or value(s) of an independent source. An external sweep is specified by augmenting the .ac statement described above with the keyword *sweep* followed by one of the following specifications.

```
sweep variable_name start stop incr
sweep variable_name sweep_type np start stop
sweep variable_name poi np p_1 p_2 ... p_n
sweep data=data1
```

Here *variable_name* is one of the following:

- Independent voltage source name
- Independent current source name

Chapter 14: AC Small-Signal Analysis

Overview of AC Analysis

- Parameter name

Note: Parameter names must begin with an alphabetic character, but thereafter can contain numbers and some special characters.

- Keyword: temp for temperature

The first format specifies variation of variable_name in the interval bounded by start and stop on the linear scale, in the increment of incr. The second and the third formats are similar to the format of the internal frequency sweep. In the second format the keyword sweep_type can be one of the following:

- lin
- dec
- oct

Note: np is number of points for lin and poi types. It can also be the number of points per decade for dec type or per octave for oct type.

When a list of points is specified, poi applies. If several variables are to be changed in AC analysis, the data syntax can be used.

Examples

```
.ac dec 10 1.e6 1.e9 sweep vcc 2 4 0.25
```

The above code example sweeps frequencies between 1 MHz and 1 GHz by placing 10 frequencies per decade for each value of voltage source vcc changing between 2V and 4V in increment of 0.25V.

```
.ac oct 4 1.e6 1.e9 sweep r2val dec 10 1e4 1e5
```

The above code sample sweeps frequencies between 1 MHz and 1 GHz by placing 4 frequencies per octave for each value of parameter r2val changing between 10 K and 100 K uniformly on logarithmic scale with 10 points per decade.

```
.ac lin 10 0.1e9 1.e9 sweep width 1e-7 2e-7 1e-8
```

The above code example sweeps frequencies between 100 MHz and 1 GHz by uniformly placing 10 frequencies between them for each value of parameter width changing between 0.1 μm and 0.2 μm in increment of 10 nm.

```
.ac poi 6 10x 20x 100x 200x 1g 2g sweep temp 0 100 10
```

The above code example sweeps frequencies through a given set of six values for each value of temperature changing between 0° C and 100° C in increment of 10° C.

```
.ac lin 11 1e8 1e9 sweep data=data4ac
.data data4ac
p1    p2    p3
1.0   2.0   3.0
1.1   2.1   3.1
1.2   2.2   3.2
1.3   2.3   3.3
.enddata
```

In AC analysis, four outer sweeps are performed. At the first outer sweep, parameters p1, p2, p3 take the values of 1, 2, and 3, respectively. At the second outer sweep, they take the values of 1.1, 2.1, and 3.1, respectively, and so on.

AC Sources

To perform AC analysis, AC source values must be defined. By default, DC voltage and current sources have AC values equal to zero. Non-zero AC values can be specified on the independent voltage/current sources statement after DC value and before transient function. The syntax is:

```
ac mag <ph>
```

where

- ac is a keyword
- mag is the magnitude of the source
- ph (an optional parameter) is the phase of the source in unit of degrees

When specifying the phase, the following convention is used, all AC signals are proportional to:

```
exp(j (W*t + q) )
```

where

- j is imaginary unit
- t is time
- Ω is cyclic frequency ($2 * \pi * \text{frequency}$)
- Phase θ is measured in radians

Chapter 14: AC Small-Signal Analysis

AC Analysis Output

Example

```
v1 node1 0 3.3 ac 1 45 pulse(3.3 0 1n 0.1n 0.1n 0.4n 2n)
```

Voltage source v1 has the following properties:

- dc value of 3.3V
- ac magnitude of 1V
- Phase of 45 degrees
- Followed by the pulse function.

Example

```
i2 node2 node3 1m ac 1
```

Current source i2 is connected between node2 and node3, with DC value of 1mA, ac magnitude of 1A, and zero phase.

```
v4 node4 node5 ac 1 30 pwl (0n 0.0 1n 1.0)
```

Voltage source v4 is connected between node4 and node5, with default DC value of zero, ac magnitude of 1V and phase of 30 degrees, followed by a piecewise linear function (pwl).

AC Analysis Output

The output of the .print ac statement goes to the outfile.ac if –o outfile. This file is specified when invoking HSIM simulation.

Note: The default file name is hsim.ac.

.alter

If the .alter statement is included in the netlist, the output goes to the output file:

```
.ac.a<n>
```

where

<n> is the alter statement number.

The output file contains names of output variables and their values for each frequency of the sweep in the tabular form. The number of output files corresponds to the number of outer sweeps.

Output variables in AC analysis are complex numbers. Thus, the output syntax in AC analysis provides additional support than that for transient and DC analyses.

.print

```
.print ac var1 <var2 ...>
```

Each output variable such as var1 could be:

- Node voltage
- Voltage between two nodes
- Branch current
- A function of the above

The output variable should contain a modifier such as suffix, to specify the part of the complex number to be printed. Allowable modifiers include:

- r: Real part
- i: Imaginary part
- m: Magnitude
- p: Phase

Voltages between two nodes in addition to modifiers described above can also have the following modifier:

- db: Decibel

Examples

```
.print ac vr(node1) vi(node1) vm(node1) vp(node1)
```

Prints real part of node voltage as well as its imaginary part, magnitude, and phase.

```
.print ac im(v1) ir(r2) ii(r2) im3(m3) ip3(m3)
```

Prints magnitude of current through voltage source v1, real and imaginary part of current through resistor r2, and magnitude and phase of the current through source terminal of MOSFET m3.

In the case of the voltages between two nodes, say nd1 and nd2, the exact meaning of the modifier is defined through the complex phasors of voltages, $v(nd1)$ and $v(nd2)$. The real and imaginary parts are defined as follows:

Chapter 14: AC Small-Signal Analysis

AC Analysis Measurement

```
vr(nd1,nd2)=real(v(nd1)-v(nd2)) ,  
vi(nd1,nd2)=imag(v(nd1)-v(nd2)) .
```

There are two different definitions for the following modifiers:

- Magnitude
- Phase
- Decibel

AC Analysis Measurement

Measurement in AC analysis is similar to measurement in transient analysis.

Example

```
.measure ac max vr(node1)
```

The maximum value of `vr(node1)` is determined and the result is stored in the file `outfile.ac_mt` if `-o outfile` is specified when invoking HSIM simulation. The default file name is `hsim.ac_mt`.

DC Analysis

Provides details for using DC Analysis to determine the quiescent state or operating point information of the circuit, or a DC sweep of a source value. DC analysis features described in this chapter include: sweeping a parameter value, sweeping a temperature value, and sweeping any combination of parameter and temperature values.

- [DC Sweep of One or Two Source Value\(s\)](#)
- [DC Sweep Parameter Value](#)
- [DC Sweep Simulation Temperature](#)
- [Multiple DC and Data Sweeps](#)
- [DC Output and Control Commands](#)
- [Measurement In DC Analysis](#)
- [DC Interactive Mode Debugging](#)
- [DC Interactive Mode Commands](#)

DC Sweep of One or Two Source Value(s)

DC analysis is used to determine the quiescent state or operating point information of the circuit. Different types of DC analysis are available in the HSIM simulator as described below:

- Sweep a source value.
- Sweep the values of two voltage or current sources.
- Sweep a parameter value.

Chapter 15: DC Analysis

DC Sweep of One or Two Source Value(s)

- Sweep temperature value.
- Any combination of the above.

Voltage or current sources must be specified as a netlist item if they are included in the .dc statement.

Either of the following syntax statements can be used.

```
.dc var start stop incr <var2 start2 stop2 incr2>
.dc var start stop incr <sweep var2 type np start2 stop2>
```

Sweeps the voltage or current source var. If the second source var2 is specified, then the first source is swept over its range for each value of the second source.

Table 76 Sweep One or Two Source Value Keyword Descriptions

Keyword	Description
sweep	Indicates that the second sweep has a different type of variation, such as the following examples: <ul style="list-style-type: none">■ dec■ oct■ poi
start	Starting value.
stop	Final value.
incr	Linearly increment value.
start2	Starting value of the second source.
stop2	Final value of the second source.
incr2	Linearly increment value of the second source.

Table 76 Sweep One or Two Source Value Keyword Descriptions

Keyword	Description
sweep-type	<p>One of the following keywords:</p> <ul style="list-style-type: none"> dec - for decade increment oct - for octave increment lin - for linear increment poi - for list of points <p>np - number of points for lin and poi types; or number of points per decade for dec type, per octave for oct type.</p>

Example

```
.dc i2 0 10m 0.5m
```

The current source **i2** is swept from 0 A to 10 mA in increment of 0.5 mA.

```
.dc vds 0 2 0.1 vgs 0 2 1
```

Voltage source **vds** is swept from 0V to 2V in increment of 0.1V at **vds** values of: 0.0V, 1.0V, 2.0V.

```
.dc v1 poi 4 0 0.3 0.5 1
```

The voltage source **v1** is swept at values of: 0.0V, 0.3V, 0.5V, 1.0V.

```
.dc vds 1 3 0.2 sweep r1 dec 3 5k 500k
```

Voltage source **vds** is swept from 1V to 3V in increment of 0.2V with the resistor **r1** value being swept from 5 Kohm to 500 Kohm by 3 values per decade (dec.).

DC Sweep Parameter Value

```
.dc param_name start stop incr
```

Sweeps **param_name**. **.dc** has the following parameters:

Chapter 15: DC Analysis
DC Sweep Simulation Temperature

start
 Starting value
stop
 Final value
incr
 Increment value

Example

```
.dc param3 1 5 1
```

Parameter param3 is swept from 1 to 5 in increments of 1.

DC Sweep Simulation Temperature

```
.dc temp start stop incr
```

Sweeps the simulation temperature. .dc temp has the following parameters:

start
 Starting temperature
stop
 Final temperature
incr
 Increment value

Example

```
.dc temp poi 4 0 25 50 75
```

HSIM is conducted at the following temperatures: 0 °C, 25 °C, 50 °C, 75 °C.

Multiple DC and Data Sweeps

Double sweeps on parameters or a combination of a source value and a parameter can be performed on HSIM.

```
.dc param1 start1 stop1 incr1 sweep param2 start2 stop2 incr2
```

Sweeps parameter param1 from start1 to stop1 in increments of incr1 for each value of parameter param2 that takes a value from start2 to stop2 in increments of incr2.

Examples

The following example sweeps voltage source v1 from 0V to 5V in increment of 0.5V for each value of parameter par2 being 1, 2, 3, ..., 10.

```
.dc v1 0 5 0.5 sweep par2 lin 10 1 10
```

The following example sweeps parameter par1 from 0 to 5 in increment of 1 for each value of parameter par2 : 1, 2, 3, ..., 10.

```
.dc par1 0 5 1 sweep par2 lin 10 1 10
```

Note: If one `sweep` variable is a source value and the other is a parameter or temperature, then the source should always be specified as the first `sweep` variable. If this rule is not followed, a Warning message is generated and the two `sweep` variables are interchanged.

If several variables are to be changed for each point of HSIM, the `data` syntax can be used.

In the following example, the `sweep` information is taken from the `data1` block.

```
.dc data=data1
```

In the following example, DC analysis is performed for four points. At the first point, parameters p1, p2, p3 take value of 1, 2, and 3, respectively. At the second point, they will take value of 1.1, 2.1, and 3.1, respectively, and so on.

```
.dc data=data1
.data data1
p1  p2  p3
1.0  2.0  3.0
1.1  2.1  3.1
1.2  2.2  3.2
1.3  2.3  3.3
.enddata
```

DC Output and Control Commands

To print output variables/parameters in HSIM use the following statement:

```
.print dc var1 <var2 ...>
```

Chapter 15: DC Analysis
Measurement In DC Analysis**Example**

```
.print dc v(node1) v(node2,node3) i(R4)
+ fun5=par('v(node4)*2.345')
```

Unlike output in transient analysis, the output of .print dc statement goes to the file outfile.dc if –o outfile is specified when invoking HSIM simulation. The default file name is hsim.dc. If the .alter statement is included in the netlist, the output goes to files outfile.dc.a<n>, where <n> is the alter number.

The HSIM output contains names of the variables and the associated values in ASCII format. In the case of two multiple sweeps or double sweeps, a separate table is printed for each value of the external loop variable.

Measurement In DC Analysis

Measurement in HSIM is similar to measurement in transient analysis. The syntax follows:

```
.measure <dc> result func out_var1
```

Example

```
.measure dc vn0 max v(node1)
```

The maximum value of v(node1) is determined and the result is stored in the file outfile.dc_mt if –o outfile is specified when invoking HSIM simulation. The default file name is hsim.dc_mt.

DC Interactive Mode Debugging

There are two methods to invoke the interactive mode.

1. Specify the desired DC iteration stop point by including the following [HSIMDCSTOPAT](#) on page 79 command in the top-level netlist, as shown:

```
.param HSIMDCSTOPAT=n_iter
```

where the value of *n_iter* is the interruption number.

2. Press [Ctrl] [C] simultaneously at any time during DC iteration.

DC Interactive Mode Commands

The interactive mode commands that are only used for DC include the following:

- dcount
- quietdc
- quite
- pt

`dccont <diteration>`

Continues DC iteration. If the optional iteration increment diteration, is followed by dccont, DC will continue for another diteration iterations, then enter the DC interactive mode. If the optional iteration increment is not specified, DC will continue without entering the interactive mode. It will stop interactively if [Ctrl] [C] are pressed simultaneously.

`quitdc`

Terminates DC iteration.

`quit`

Terminates both DC iteration and transient simulation

`pt`

Obtains the current DC iteration number.

The other transient interactive mode commands are supported in DC mode.

Feedback

Chapter 15: DC Analysis

DC Interactive Mode Commands

Part 2: HSIM Advanced Analysis

Interactive Mode Debugging

Describes the HSIM interactive circuit debugging environment that probes dynamic circuit data (circuit elements and their parameters, circuit connectivity, and instantaneous signal values) and circuit behavior (voltage and current waveforms simulated up to the present time instance) using interactive mode commands. It also describes the Tool Command Language's (TCL) interactive control features.

- [Overview of Interactive Mode Debugging](#)
- [List of Interactive Mode Commands](#)
- [Interactive Mode Commands](#)

Overview of Interactive Mode Debugging

The interactive circuit debugging environment probes the dynamic circuit data and circuit behavior with interactive mode commands. The circuit data includes circuit elements and the parameters, circuit connectivity, and instantaneous signal values. The dynamic circuit behavior includes voltage and current waveforms simulated up to the present time step.

The interactive debugging environment can be used in DC simulation and transient analysis simulation.

DC init. Interactive Mode Debugging

There are two methods to invoke the interactive mode.

Chapter 16: Interactive Mode Debugging

Overview of Interactive Mode Debugging

Method 1

Specify the desired DC iteration stop point by including the following syntax line at the top-level netlist.

```
.param HSIMDCSTOPAT=n_iter
```

where *n_iter* is the number of the iteration.

Method 2

Press Ctrl-C at any time during DC iteration.

DC Interactive Mode Commands

The interactive mode commands for DC analysis are as follows:

dccont <diteration>

Continues DC iteration as follows:

- If the optional iteration increment *diteration* is followed by dccont, DC will continue for another *diteration* iterations, then enter the DC interactive mode.
- If the optional iteration increment is not specified, DC will continue without entering the interactive mode. It will stop interactively by pressing [Ctrl] [C].

quitdc

Terminates DC iteration.

quit

Terminates both DC iteration and transient simulation

pt

Obtains the current DC iteration number of DC iterations.

Other transient interactive mode commands are supported in DC mode.

Transient Interactive Mode Debugging

The transient interactive mode permits you to perform either of the following operations at any time.

- Change simulation control parameters
- Store simulation database

Refer to [inc on page 485](#) and [savesim on page 499](#).

When activated, the interactive mode prompt HSIM> appears on the screen at the interruption time. It allows the interactive mode commands to be invoked. The commands are described in this chapter.

During each interactive session, Tool Command Language (TCL) uses a built-in command history that lists all the previous commands. The commands described in [Table 77 on page 465](#), can be used to retrieve previous commands.

Table 77 TCL Command Descriptions

Command	Description
!!	Most recently used command
!#	The #th command on the command history list

Time is measured in nanoseconds in the interactive mode.

There are three methods to invoke the interactive mode.

1. Specify the desired interruption time by including the following syntax in the top-level netlist:

```
.param HSIMSTOPAT=time1
```

The value of `time1` is the interruption time measured in seconds.

An equivalent method is to include the following syntax in the input netlist:

```
.stop at=time1 <cmf=interactive_command_file>
```

When `interactive_command_file` is specified, the commands included in the `interactive_command_file` are executed when the simulation is switched into the interactive mode.

2. Press [Ctrl] and [C] simultaneously at any time after DC initialization.
3. Specify the conditional interruption in the netlist.

```
.stop v(node_name) val=voltage_value <edge=r|f|c>
<from=time1> <to=time2> <cmf=interactive_command_file>
```

Chapter 16: Interactive Mode Debugging

List of Interactive Mode Commands

The syntax is described in [Table 78 on page 466](#).

Table 78 Conditional Interruption Syntax Descriptions

Parameter	Description
node_name	Specifies the voltage node, such as n1 or out2.
voltage_value	The target voltage value of the node at which the simulation stops. The simulation will enter the interactive mode if this is set in the previous batch mode.
edge	<p>The format of edge is similar to that of .measure.</p> <ul style="list-style-type: none"> ▪ r indicates action at the rising edge when the node voltage reaches the target value. ▪ f indicates action at the falling edge. ▪ c indicates action at either the rising or the falling edge. ▪ If no edge parameter is specified, c is the default.
from, to	This parameter specifies the time window in which to check the condition.

Following is an example of a command that is placed in the input netlist.

```
.stop v(out2) val=1.5 edge=r from=0 to=100n
```

If the `out2` node reaches 1.5V at the rising edge of a time window between 0 ns and 100 ns, HSIM enters the interactive simulation mode.

List of Interactive Mode Commands

[Table 79 on page 466](#) lists the commands used by HSIM.

Note: [Table 79 on page 466](#) lists the commands in Key Word alphabetical order. Command descriptions are documented in alphabetical order by command name in the sections that follow.

Table 79 Interactive Mode Debugging Commands

Command	Function
alias	Create Alias

Table 79 Interactive Mode Debugging Commands (Continued)

Command	Function
ap	Add Printout Nodes
closelog	Close Interactive Mode Logfile
cont	Continue Simulation
dcon	Print Max. Connectivity Node Name
dcpath	Find DC Current Path
dn	Dump User-Specified Nodes
dp	Delete Printout Nodes
eid	Get Element Index From Element Name
ename	Get Element Name From Element Index
ev	Get Element Information
exi	Report Elements with Excessive Current
fcc	Report Flash Core Cell Elements
fmeta	Find Meta Condition
fv	Force Constant Voltage to Nodes
help	Get Interactive Mode Command Syntax and Description
HSIMDELVDTO	Flash Core Transistor Usage Syntax
HSIMALLOWEDDV, HSIMSTEADYCURRENT, HSIMSPICE,HSIMTAUMAX	Change Simulation Control Parameters
iap	Add Printout Node by Node Index
idp	Delete Printout Node by Node Index
iev	Get Element Information by Element Index

Chapter 16: Interactive Mode Debugging

List of Interactive Mode Commands

Table 79 Interactive Mode Debugging Commands (Continued)

Command	Function
inc	Get Node Connectivity Information by Node Index
inv	Get Node Voltage and Capacitance by Node Index
lx	List subckt_name Instances
matche	Find Element Names of a Specified Pattern
matchn	Find Nodes with Logic Value Changes
nc	Get Node Connectivity Information by Node Name
nctr	Print connected Transistors Through R/L/C Network
ni	Print Node Current
nid	Get Node Index from Node Name
nm	Print Nodes Merged to a Specified Node Name
nname	Get Node Name from Node Index
nv	Get Node Voltage and Capacitance by Node Name
op	Dump Operating Point
openlog	Open Interactive Mode Logfile
pt	Get Current Time
quit	Terminate Simulation
rcf	Read Commands from the File
restart	Restart Simulation from Saved Database
rv	Release Constant Voltage From Nodes
savesim	Save Simulation Database
stop <condition>	Conditional Interruption

Table 79 Interactive Mode Debugging Commands (Continued)

Command	Function
stop -at	Interrupt Simulation
stop -delete	List and Delete Interruption Points
stop -list	
vni	Print Voltage Source Node
trace_thru_on nodeName	Traces the Circuit through Conducting Elements and Identifies High Impedance
tree	Tree Command

Interactive Mode Commands

This section lists the interactive mode commands:

- [alias](#)
- [ap](#)
- [closelog](#)
- [cont](#)
- [dcon](#)
- [dcpath](#)
- [de](#)
- [dn](#)
- [dp](#)
- [eid](#)
- [ename](#)
- [ev](#)
- [exi](#)
- [fcc](#)

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

- [fmeta](#)
- [fv](#)
- [help](#)
- [iap](#)
- [idp](#)
- [iev](#)
- [inc](#)
- [inv](#)
- [lx](#)
- [matche](#)
- [matchn](#)
- [nc](#)
- [nctr](#)
- [ni](#)
- [nid](#)
- [nm](#)
- [nname](#)
- [nv](#)
- [op](#)
- [openlog](#)
- [pt](#)
- [quit](#)
- [rcf](#)
- [restart](#)
- [rv](#)
- [savesim](#)
- [stop <conditional>](#)
- [stop -at](#)
- [stop -list and stop -delete](#)

- [vni](#)
 - [trace_thru_on](#)
 - [tree](#)
-

alias

Creates alias command.

Syntax

```
alias alias_n cmd_name
```

Example

```
HSIM> alias continue cont
```

Description

This command creates the alias name `alias_n` for the interactive command `cmd_name`.

Note: `continue` has the same effect as the command `cont`.

ap

Adds printout for node and device patterns.

Syntax

```
ap <-logic> <-vlth value> <-vhth value> node_name  
<v(node_name)> <i(device_name)>
```

Description

If `-logic` is specified, this command prints the logic waveform. If `-vlth/-vhth` are specified, that value will be the threshold.

`ap` adds printout nodes in the waveform node list and starts printing the waveform data for the specified nodes from the current time. The `node_name` may contain the asterisk (*) wildcard character to specify all the nodes matching the specified pattern. The `ap` command can be used with `dp` to control the time windows for the waveform data of selected nodes. Refer to [dp on page 478](#) for additional details.

To debug circuit behavior, it may be necessary to print the waveform data of every node in the circuit. This data can produce a large output file which slows

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

down the simulation if the circuit size is large. In most cases it is only necessary to debug the waveform of selected time windows.

The interactive mode commands ap and dp permit restricting the printout data to the specified time window(s). This reduces the waveform file size. This command only works when the FSDB and WDF output formats are used.

Examples

In the following example, the pt command obtains the present time information. The example uses ap to add nodes xalu3.xlatch2.mi5 and all nodes matching the pattern xpll.* into the waveform list. These nodes will be printed starting from the current time at 125.3 ns. Refer to [pt on page 497](#) for details about pt.

If the dp command is not entered, then the printout of these nodes continues until the end of the simulation. If dp is entered, the printout of those specified nodes is terminated at the time dp is entered.

```
HSIM> pt  
tnow=125.3ns      tmax=1000ns  
HSIM> ap    xalu3.xlatch2.mi5 xpll.*
```

In the following example, from 251.4 ns the nodes matching the pattern xpll.x2.* will not be printed.

```
HSIM> pt  
tnow=251.4ns      tmax=1000ns  
HSIM> dp xpll.x2.*
```

In the following example, the pt command obtains the present time information. The example uses ap to print out the waveforms for the current of the x1.x2.m1 device and *.m1 devices that match the pattern. These nodes are printed starting from the current time at 100ns. Refer to [pt on page 497](#) for details about pt.

```
HSIM> pt  
tnow=100nsns tmax=1000ns  
HSIM> ap i(x1.x2.m1) i(*.m1)
```

closelog

Closes interactive mode logfile.

Syntax

```
closelog
```

Description

The closelog command closes the interactive mode log file that was opened by the openlog command. The log file contains all records of interactive mode commands and the results reported by the commands entered between openlog and closelog. For details about openlog, refer to [openlog on page 496](#).

cont

Continues simulation.

Syntax

```
cont <dtime>
```

Description

This command continues the transient simulation. If the optional time increment *dtime*, measured in nanoseconds, is followed by the cont command, the simulation stops and again enters the interactive mode at time $t+dtime$. *t* is the current time. If $t+dtime$ is less than the simulation stop time, the simulation will continue until $t+dtime$, then enter the interactive mode at $t+dtime$. If the optional time increment is not specified, the simulation continues until it is completed, or until the interrupt [Ctrl] and [C] is entered.

Examples

```
HSIM> cont 25.3
```

Continues the simulation and enters the interactive mode 25.3 ns later.

```
HSIM> cont
```

Leaves the interactive mode and continues the simulation.

dcon

Prints max. connectivity node name.

Syntax

```
dcon [val]
```

Description

The dcon command finds the circuit node that has the most connections based on the netlist. When *val* is specified, nodes with more connections than *val*

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

are reported. Without `val`, the command reports the most connected node in the netlist. Each node will contain subckt name, node name, number of connection, and a flag `Vsrc` showing if it is voltage source.

Example

```
HSIM> dcon 20
```

Reports the nodes that have more than 20 connections in the netlist.

dcpath

Finds the DC current path.

Syntax

```
dcpath <-ith current1> <ith2 current2> <-time time1> <-file  
file1> <node1 < node2 ...>>
```

Description

`dcpath` finds the DC current path between specified nodes at present time or a given time later than present time. The options are as described in [Table 80 on page 474](#).

Table 80 Find DC Current Path: `dcpath` Syntax Descriptions

Parameter	Description
-ith current1	Specifies the threshold current of DC path. The default value is 5e-5.
-ith2 current2	Specifies the threshold current for additional remark on the DC path generated by "-ith". When this optional parameter is set, MOSFET devices on the DC path with an <code>Ids</code> current that exceeds the <code>ith2</code> value are marked with wildcard ("*"). The default <code>ith2</code> value is 1e-3A.
-time time1	Specifies the time, which is measured in the unit of nanosecond 1.0e-9 second, for HSIM to enter interactive mode and perform DC path search at that time. If -time parameter is not specified, then the DC current path is searched at the current time.
-file file1	Redirects the DC path search result to the specified file. If the file is not specified, then the result is printed on the screen.

Table 80 Find DC Current Path: *dcpath* Syntax Descriptions (Continued)

Parameter	Description
node1 node2 ...	Specifies the terminal node names of DC current path. The DC path search starts from any node specified in this node list and ends when it reaches either another node in the list or a DC voltage source node. If no node is specified, then HSIM reports DC current path(s) between any pair of voltage source nodes.

Example

```

* -----
* DC Path Check
* title      = t1
* ith        = 1e-09
* ith2       = 8e-06
* period     =
* delay      =
* at         = 3e-08
* sort       = 0
* separate_file = 1
*
path 1 from vdd! to 0 @ 3e-08s
    i5.m1.m0.lvpd (PMOS: Ids=3.12947e-06)
        source vdd! 3.6
        drain  o 3.5618
        gate   cm2 0.597389
    i5.m0.m0.lvn (NMOS: Ids=3.02668e-06)
        source 0 0
        drain  o 3.5618
        gate   cm2 0.597389
path 2 from vdd! to 0 @ 3e-08s
    * m1.m0.lvpd (PMOS: Ids=8.05206e-06)
        source vdd! 3.6
        drain  cm2 0.597389
        gate   cm1 3.08191
    * m3.m0.lvn (NMOS: Ids=8.63069e-06)
        source 0 0
        drain  cm2 0.597389
        gate   cm2 0.597389
* t1: Total number of paths = 2

```

The report is generated based on the threshold current "ith" with a value of 1e-9Amp. Based on this report an "ith2" threshold current can be specified with a value of 8e-6Amp. As a result a wildcard ("*") placed in front of the MOSFET

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

devices that have an I_{ds} current exceeding this i_{th2} value. In this example the candidates are "m1.m0.lvpd" and "m3.m0.lvn".

Note: Do not specify a Z-state within PWLZ when you perform a dcpath check. If a PWLZ source and Z-state are specified within PWLZ, HSIM does not consider PWLZ as a voltage source because there might be a period of time when PWLZ is not driven by voltage source elements. The dcpath check has no indicators of the starting VSRC node locations, and because of this situation, the reported path might not be a legitimate path.

Using dcpath

Keep in mind the following guidelines when using the dcpath interactive command.

1. Two or more node names (pattern) need to be specified to form the pair of nodes for the DC path search.
2. If no node names (pattern) are specified, HSIM automatically forms the vsrc pairs among all of the recognized vsrc nodes for the DC path search.
3. Among the specified node names (pattern), if one (or more than one) is marked with a wildcard ("*") then HSIM treats it as (2) above.
4. The node name pattern can support partial wild cards, such as $*abc^*$, $*abc$, or abc^* .
5. If the interactive dcpath command is used to form "selective" vsrc pairs, we recommended you do the following:
 - a. Apply the interactive command "vni" to list all of the vsrc nodes in the design.
 - b. Reference this vsrc node list and apply the dcpath command with the above guidelines from items (1) to (4).

For details about dcpath power check, refer to [Chapter 17, Timing and Power Analysis, DC Path Check on page 527](#).

de

Finds larger circuit capacitor element.

Syntax

```
de [ [-ncap [val]] ] | [ [-pcap [val]] ]
```

Description

This command finds the circuit capacitor element with the large capacitance as described in the following bullets.

-ncap [val]

If val is specified, -ncap [val] finds capacitors with capacitance smaller than val. If val is not specified, it finds the minimum negative capacitance. val is in farads.

-pcap [val]

If val is specified, -pcap [val] finds capacitors with capacitance greater than val. If val is not specified, it finds the maximum positive capacitance. val is in farads.

Example

```
HSIM > de -pcap
Maximum CAP=1pF at element c5, time_step=1e+006ps
HSIM > de -pcap 1e-14
    CAP elem c4 cap=1e-013
    CAP elem c5 cap=1e-012

HSIM > de -pcap
Maximum CAP=1pF at element c5, time_step=1e+006ps
```

dn

Dumps user-specified nodes.

Syntax

```
dn [[-dv [val]] | [-v [val]] | [-ncap [val]]] [-print [file]]
```

Description

This command finds the circuit node that has the large voltage change, voltage, or node capacitance. The options are as described in [Table 80 on page 474](#)

-dv [val]

Finds the node with change voltage greater than val (if val is specified) or the maximum delta voltage (if no val is specified). The unit of val is volt.

-v [val]

Finds the node with voltage greater than val (if val is specified) or the maximum voltage (if no val is specified). The unit of val is volt.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

-ncap [val]

Finds the node with capacitance greater than val (if val is specified) or the maximum capacitance (if no val is specified). The unit of val is farad.

-print [file]

Adds the matching nodes to the printout node list and create an nWave .rc file for signal restore. If no file is specified, the .rc file is named hsim.rc (if no output prefix is specified when running HSIM) or {prefix}.rc (if -o prefix is specified when running HSIM).

Examples

HSIM> dn -ncap 1e-15

Reports nodes that have capacitance value greater than 1e-15f.

HSIM> dn -dv

Reports the node that has maximum voltage change.

HSIM> dn -v 0.5 -print

Reports nodes with voltage greater than 0.5v. Also add those nodes into a printout list and generate a .rc file that can be debugged using nWave.

An example of the generate .rc is shown below:

```
; rc file created by command dn
;
addGroup "Gxxx.rc"
addSignal -w analog -c ID_ORANGE5 /xbuff/v(n4)
addSignal -w analog -c ID_ORANGE5 /v(n2)
addSignal -w analog -c ID_ORANGE5 /v(n4)
addSignal -w analog -c ID_ORANGE5 /v(n5)
addSignal -w analog -c ID_ORANGE5 /xbuff/v(n3)
```

dp

Deletes printout nodes.

Syntax

dp node_name1 <node_name2 ...>

Description

dp deletes printout node(s) from the waveform node list and stops printing the waveform data for the specified node(s) from the current time. The node_name may contain the asterisk (*) wildcard character to specify all the node(s) that

match the entered pattern. Together with the ap command, the time window(s) for the waveform data of selected nodes can be controlled. Refer to [ap on page 471](#) for details about ap.

To debug the circuit behavior, it may be necessary to print the waveform data for every node in the circuit. However, this generates a large output file which can slow down HSIM simulation when the circuit size is very large. In most cases, only the selected time window(s) requires debugging. The interactive mode commands ap and dp restrict the printout data to the specified time window(s) which reduces the waveform file size. Refer to the example in [ap on page 471](#).

eid

Gets element index from element name.

Syntax

```
eid elem_name
```

Description

eid obtains an index number to a specific element within the HSIM simulation database. To access the element information through the query commands such as ev and iev, either the element name or the corresponding element index number is required. If the element name is very long, it may be convenient to use the index number instead. The eid command accesses the element index number. In the following example, the index number is 168.

Refer to the following for additional information on ev and iev: [ev on page 480](#) and [iev on page 484](#).

Example

```
HSIM> eid xalu3.xlatch2.mi5
```

ename

Gets element name from element index.

Syntax

```
ename elem_index
```

Description

ename reports the element name corresponding to the specified index number.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

Example

```
HSIM> ename 168  
xalu3.xlatch2.mi5
```

ev

Gets element information.

For models that include drain/source resistors (RDSMOD=1 or NRD/NRS instance parameters) the reported source/drain voltage is representative of the internal drain/source voltage, which may differ from the external source/drain node voltage.

Syntax

```
ev element_name
```

Description

ev prints the detailed element information for the given `element_name` at the specific time it is activated including:

- Element terminals
- Element type
- Terminal voltages
- Element currents
- Element conductance
- Element capacitances

The `element_name` can contain the asterisk (*) wildcard character.

Example

```
HSIM> ev xalu3.xlatch2.mi5  
xalu3.xlatch2.mi5 (168) - DRN: xalu3.xlatch2.n24 (755), GATE:  
xalu3.xlatch2.q (888), SRC: Gnd (0), BULK: Gnd (0)  
NMOS VG=2.23, VD=0.87, VS=0, VB=0  
Ids=2.352E-5, gds=1.13E-4, gm=2.115E-4  
Voltage-dependent diode cap: cbd=2.17E-15, cbs=1.23E-15  
Voltage-dependent gate cap: cgs=3.17E-15, cgd=1.21E-16,  
cgb=2.22E-15
```

exi

Reports elements with excessive current.

Syntax

```
exi <-ith current_val> elem_name1 <elem_name2 ... >
```

Description

exi identifies and reports the element(s) from the specified element name(s) that has excessive current greater than the specified threshold current value. The unit of the threshold current value is ampere. The default value is 5e-5. The element name may contain asterisk (*) wildcard character to represent all elements matching the pattern.

Example

```
HSIM> exi -ith 1e-5 x1.*
```

Each element in instance x1 is reported if the element current value exceeds 10 uA at the present time.

fcc

Reports flash core cell elements.

Syntax

```
fcc <-dvth value> <-file filename> <-save filename>  
      elem_name1 <elem_name2... >
```

Description

fcc identifies and reports the flash core cell elements from specified element names having a threshold voltage shift with either of the following parameters:

- Greater than or equal to the specified value if it is positive or zero
- Smaller than or equal to the specified value if it is negative or zero

fcc The element name may contain an asterisk (*) wildcard character to represent all elements matching the pattern.

The fcc command supports the following options: BSIM3, BSIM4, EKV, MOS9, MOS11, and MOS Level-1.

Examples

In the example below, each element in instance x1 is reported if the element threshold voltage shift is greater than or equal to 0.2 V at the present time:

```
HSIM> fcc -dvth 0.2 x1.*
```

The example below shows a printout:

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

```
m1, Vth=-0.870335, dVth=-3.37534, delvto=0  
m3, Vth=-0.500000, dVth=-2.12673, delvto=0.2
```

The -file option saves the printout to a file instead of showing it on a monitor. This is convenient for test cases where there are many flash core cells and dumping to a file would be much faster.

The -save option saves a printout in a file that can be used to restore a simulation with flash core cells. The syntax is as follows:

```
.hsimparam HSIMDDVTO=dvth_value inst= elem_name
```

Using the syntax in the previous syntax, the following is saved.

```
.hsimparam HSIMDDVTO=-3.37534 inst=m1  
.hsimparam HSIMDDVTO=-2.12673 inst=m3
```

fmeta

Finds meta condition.

Syntax

```
fmeta
```

Description

fmeta finds the meta-stable conditions in the circuit. Meta-stable conditions can induce large currents through power supply, it can also cause oscillation that slows down the simulation. If meta-stable conditions are found after DC initialization, either node logic 0 or 1 should be forced and then released using the fv and rv command. A logic 1 or 0 ic condition can also be applied on one of the nodes and then, rerun the simulation.

Example

```
HSIM > fmeta  
Find meta stable nodes  
potential meta stable node nod1(1192) nod2 (1193)  
element mn2(6698)
```

fv

Forces constant voltage to nodes.

Syntax

```
fv node1 <node2...> val1
```

Description

`fv` forces the specified nodes to stay at the specified constant voltage. The node voltage stays at the same value from the current time until either the end of simulation or when the constant node voltage status is released by the command `rv`. In the interactive mode, the `fv` command does not work on voltage source nodes or vector files. Refer to [rv on page 498](#).

Example

```
HSIM> fv    pump 6.5
```

The voltage at node `pump` stays at 6.5V from the current time on.

help

Gets interactive mode command syntax and description.

Syntax

```
help <command_name>
```

Description

`help` displays a brief description of the specified command. If a command is not specified, all interactive mode commands are displayed. To get the command syntax for a specific command, enter the following:

```
<cmd_name> -help
```

Example

```
HSIM> help ename  
ename - get element name by element id.
```

iap

Adds printout node by node index.

Syntax

```
iap <-logic> <-vlth value> <-vhth value> node_index
```

Description

If `-logic` is specified, this command prints the logic waveform. If `-vlth/-vhth` are specified, that value will be the threshold.

`iap` adds printout nodes that are specified by the node index to the waveform data list. This command works the same as `ap` except the node index number

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

follows the iap command while the node name follows the ap command. This command works only in FSDB and WDF output formats. For details about ap, refer to [ap on page 471](#).

Example

```
HSIM> iap 168
```

idp

Deletes printout node by node index.

Syntax

```
idp node_index1 <node_index2 ...>
```

Description

idp deletes printout node, specified by the node index, from the waveform data list. This command functions as does dp, except the node index number follows the idp command where the node name follows the dp command. This command operates only in FSDB and WDF output formats. For details about dp, refer to [dp on page 478](#).

Example

```
HSIM> idp 168
```

iev

Gets element information by element index.

For models that include drain/source resistors (RDSMOD=1 or NRD/NRS instance parameters) the reported source/drain voltage is representative of the internal drain/source voltage, which may differ from the external source/drain node voltage.

Syntax

```
iev elem_index
```

Description

iev prints the detailed element information for the specified element index, including:

- Element terminals
- Element type

- Terminal voltage
- Element current
- Element conductance
- Element capacitance

Example

```
HSIM> iev 168
xalu3.xlatch2.mi5 (168) - DRN: xalu3.xlatch2.n24 (755), GATE:
xalu3.xlatch2.q (888), SRC: Gnd (0), BULK: Gnd (0)
NMOS VG=2.23, VD=0.87, VS=0, VB=0
Ids=2.352E-5, gds=1.13E-4, gm=2.115E-4
V-dependent diode cap: cbd=2.17E-15, cbs=1.23E-15
V-dependent gate cap: cgs=3.17E-15, cgd=1.21E-16, cgb=2.22E-15
```

inc

Gets node connectivity information by node index.

Syntax

```
inc node index <-level num> <-inst name> <-etype
  (c|d|f|h|i|l|m|q|r|v)> <-iin val2> <-iout val3> <-term
  i<,j ...> > <-on val4> <-total>
```

Description

inc prints the detailed node connectivity information for the given node index. If no option is specified, then all the elements connected to the node are printed. Each option selectively prints a subset of elements described in [Table 81 on page 485](#).

Table 81 inc Command Element Descriptions

Parameter	Description
-level num	Only prints the elements that are located in the hierarchies between the current level where the specified node index is located, to num-1 levels below the current level.
-inst name	Only prints the elements that are located within the given instance name.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

Table 81 inc Command Element Descriptions (Continued)

Parameter	Description
-etype (...)	<p>Only prints the elements that match the specified type(s). More than one character can be specified. Each character defines the element type. For example:</p> <ul style="list-style-type: none"> ▪ c specifies the capacitor ▪ r specifies the resistor ▪ m specifies the MOSFET <p>Space is not allowed between the element characters when more than one element is specified.</p>
-iin val2	Only prints the element which, at the current time, has conducting current higher than the specified value val2 flowing into the specified node index.
-iout val3	Only prints the element which, at the current time, has conducting current higher than the specified value val3 flowing out of the specified node index.
-term i,j ...	<p>Only prints the elements connected to the specified terminal number(s) of the specified node index. Multiple terminal numbers, separated by a comma (,), can be specified. Space is not</p> <p>allowed between the comma (,) and the terminal number.</p>
-on val4	Only prints the conducting element which, at the current time, has conducting current higher than the specified current value val4, and is connected to the specified node index.
-total	The conducting current also includes the capacitor current.

Examples

A description follows each example.

HSIM> inc 888

Prints all the elements connected to the 888th node.

HSIM> inc 888 -level 2

Prints elements connected to the 888th node and are also located in either the same hierarchical level as the 888th node or one level below.

```
HSIM> inc 888 -inst xalu3.xlatch2
```

Prints the elements connected to the 888th node and are also located within the subcircuit instance xalu3.xlatch2.

```
HSIM> inc 888 -etype rm
```

Prints resistors and MOSFET transistors connected to the 888th node.

```
HSIM> inc 888 -iin 0.001
```

Prints elements which conduct more than 1 mA of current flowing into the 888th node.

```
HSIM> inc 888 -iout 0.00012
```

Prints elements which conduct more than 120 uA of current flowing out of the 888th node.

```
HSIM> inc 888 -term 1,3 -etype m
```

Prints the MOSFET transistors which have either drain or source terminal connecting to the 888th node.

```
HSIM> inc 888 -on 1E-4
```

Prints elements which conduct more than 100 uA of current and are also connected to the 888th node.

inv

Gets node voltage and capacitance by node index.

Syntax

```
inv node_index
```

Description

inv prints node voltage, slope of node voltage waveform, and node capacitance for the given node index. Each value is evaluated at the most recent time, which is when the node voltage is last updated. The data are then printed.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

Example

If the current time is 100 ns and the printed voltage is 2.5V with the printed time 95 ns, it implies the node voltage remains unchanged at 2.5V between 95 ns and 100 ns. The capacitance report contains both constant and voltage-dependent capacitances. The constant capacitance is the sum of constant capacitances connected to the node and the voltage-dependent capacitance is the sum of all MOSFET capacitances connected to the node.

```
HSIM> inv 888
V=5, dV/dt=3528.7, time=5.278E-8
Constant capacitance=2.38fF, V-dependent capacitance=5.73fF
Total capacitance=8.11fF
```

lx

Lists subckt_name instance.

Syntax

```
lx subckt_name
```

Description

lx lists all instances of specified subckt_name.

Example

```
HSIM> lx add8
x1.x1
x1.x3
x1.x2
x1.x2
x1.x10
x1.x4
x1.x5
x1.x6
x1.x7
x1.x8
x1.x9
```

matche

Finds element names of a specified pattern.

Syntax

```
matche pattern
```

Description

`matche` finds all element names that match the specified pattern.

matchn

Finds nodes with logic value changes.

Syntax

`matchn [val]`

Description

In order to trace the logic signal value changes, `matchn` has been enhanced with the following new options. The logic values for a signal are one of the following: 0, 1, X, and Z.

Those options require that the matched nodes have `.lprint` specified for the netlist so the waveform histories of those nodes are kept. The matched nodes without waveform history will be ignored. Currently the only way to specify the matching condition for those options is using `-ntl`.

- `-change val`: Define the change of a node. `val` can be one of rise, fall, or all.
- `rise`: A node changes from 0 to 1.
- `fall`: A node changes from 1 to 0.
- `all`: A node has any changes in values (0,1,Z,X)
- `-print [file]`: Add the matching nodes to the printout node list and create an nWave .rc file for signal restore. If no file is specified, the .rc file is named `hsim.rc` (if no output prefix is specified when running HSIM) or `{prefix}.rc` (if `-o prefix` is specified when running HSIM).

By default, waveform histories for those nodes will be analyzed from time 0 to the current simulation time. The waveform range can also be controlled by the following two options:

- `-start time`: Specify the start time of the waveform range. The default is 0.
- `-stop time`: Specify the stop time of the waveform range. The default is the current time.

Example

```
HSIM> matchn -ntl xp11.x2.* -change rise -print xxx.rc
```

Reports the nodes that have logic value changes from 0 to 1 in the current simulation run. Create an nWave .rc file `xxx.rc`.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

Note: This is assuming .lprint xp11.x2.* is required to be in the netlist.

nc

Gets node connectivity information by node name.

Syntax

```
nc node_name <-level num> <-inst name> <-etype  
(c|d|f|h|i|l|m|q|r|v)> <-iin val2> <-iout val3> <-term  
i<,j ...> <-on val4> <-total>
```

Description

nc prints the detailed node connectivity information for the given node name. If there is no option specified, then all the elements connected to the node are printed. Each option selectively prints a subset of elements.

Arguments

[Table 82 on page 490](#) shows the parameters.

Table 82 Node Name Connectivity Information Command Parameters

Parameter	Description
-level num	Only prints elements that are located in the hierarchies between the current level where the specified node is located, to num-1 levels below the current level.
-inst name	Only prints elements that are located within the given instance name.
-etype ...	Only prints elements that match the specified type(s). More than one character can be specified such that each character defines the element type. For example: <ul style="list-style-type: none">▪ c specifies the capacitor▪ m specifies the MOSFET▪ Space is not allowed between element characters when more than one element is specified.
-iin val2	Only prints the element which, at the current time, is conducting current higher than the specified value val2. The current is flowing into the specified node name.

Table 82 Node Name Connectivity Information Command Parameters (Continued)

Parameter	Description
-iout val3	Only prints the element which, at the current time, is conducting current higher than the specified value val3. The current is flowing out of the specified node name.
-term i,j ...	Only prints the elements that connect to the specified terminal number(s) of the specified node name. Multiple terminal numbers, separated by a comma (,), can be specified. No space is allowed between the comma (,) and the terminal number.
-on val4	Only prints the element that is connected to the specified node name and, at the current time, is conducting current higher than the specified current value val4.
-total	The conducting current also includes the capacitor current.

Examples

A description follows each example of code.

```
HSIM> nc xalu3.xlatch2.q
```

Prints all the elements connected to node xalu3.xlatch2.q.

```
HSIM> nc xalu3.xlatch2.q -level 2
```

Prints elements connected to the node xalu3.xlatch2.q and are also located in either the same hierarchical level as xalu3.xlatch2.q or one level below.

```
HSIM> nc xalu3.xlatch2.q -inst xalu3.xlatch2
```

Prints the elements connected to node xalu3.xlatch2.q and are also located within the subcircuit instance xalu3.xlatch2.

```
HSIM> nc xalu3.xlatch2.q -etype rm
```

Prints resistors and MOSFET transistors connected to the node xalu3.xlatch2.q.

```
HSIM> nc xalu3.xlatch2.q -iin 0.001
```

Prints elements which conduct more than 1mA of current flowing into the node xalu3.xlatch2.q.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

```
HSIM> nc xalu3.xlatch2.q -iout 0.00012
```

Prints elements which conduct more than 120uA of current flowing out of the node xalu3.xlatch2.q.

```
HSIM> nc xalu3.xlatch2.q -term 1,3 -etype m
```

Prints the MOSFET transistors which have either drain or source terminal connecting to the node xalu3.xlatch2.q.

```
HSIM> nc xalu3.xlatch2.q -on 1E-4
```

Prints elements which conduct more than 100 uA of current and are also connected to the node xalu3.xlatch2.q.

nctr

Prints connected transistors through R/L/C network.

For a given node name, nctr uses it as starting point to traverse R/L/C network to find all the connected transistor device(s).

Syntax

```
nctr <node_name> <-term (terminal index)>
      <-on (conducting threshold current)>
      <-iin (incoming threshold current)>
      <-iout (outgoing threshold current)>
      <-etype (device type)> <-total>
```

Description

This command prints connected transistors through R/L/C network. For a given node name, nctr uses it as starting point to traverse R/L/C network to find all the connected transistor device(s).

Note: nctr usage is the same as for the nc command.

Examples

Example 43

```
HSIM > nctr out1
```

Prints transistor connection through R/L/C network:

```
Node: out1
r1 (1) - 100 ohms POS:out1 (5), NEG:in2 (4)
xmos1.mn1 (4) NMOS - DRN:out1 (5), SRC:out2 (6), GATE:in1 (3),
BULK:gnd (2)
xmos1.mp1 (6) PMOS - DRN:out1 (5), SRC:vdd (9), GATE:in1 (3),
BULK:vdd (9)
```

```
Node: in2
mn3 (0) NMOS - DRN:in2 (4), SRC:dummys (1), GATE:dummyin (0),
BULK:gnd (2)
r1 (1) - 100 ohms POS:out1 (5), NEG:in2 (4)
xmos2.mn1 (7) NMOS - DRN:out3 (7), SRC:out4 (8), GATE:in2 (4),
BULK:gnd (2)
xmos2.mn2 (8) NMOS - DRN:out4 (8), SRC:gnd (2), GATE:in2 (4),
BULK:gnd (2)
xmos2.mp1 (9) PMOS - DRN:out3 (7), SRC:vdd (9), GATE:in2 (4),
BULK:vdd (9)
```

Example 44

```
HSIM > nctr out1 -term 1
```

Prints transistor connection through R/L/C network:

```
Node: out1
r1 (1) - 100 ohms POS:out1 (5), NEG:in2 (4)
xmos1.mn1 (4) NMOS - DRN:out1 (5), SRC:out2 (6), GATE:in1 (3),
BULK:gnd (2)
xmos1.mp1 (6) PMOS - DRN:out1 (5), SRC:vdd (9), GATE:in1 (3),
BULK:vdd (9)
```

```
Node: in2
mn3 (0) NMOS - DRN:in2 (4), SRC:dummys (1), GATE:dummyin (0),
BULK:gnd (2)
```

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

Example 45

```
HSIM > nctr out1 -term 1 -on 1e-10
Prints transistor connection through R/L/C network:
```

```
Node: out1
r1 (1) 8.50951e-009 A - 100 ohms POS:out1 (5), NEG:in2 (4)
xmos1.mn1 (4) NMOS -8.79645e-009 A - DRN:out1 (5), SRC:out2 (6),
GATE:in1 (3), BULK:gnd (2)
xmos1.mp1 (6) PMOS 2.81614e-010 A - DRN:out1 (5), SRC:vdd (9),
GATE:in1 (3), BULK:vdd (9)
```

```
Node: in2
```

ni

Prints node current.

Syntax

```
ni node_name
```

Description

`ni` prints the node current for the given node name. For nodes not connected to voltage sources, the node current is the sum of all the branch currents flowing away from the node. In this definition the branch currents entering the node are neglected. In the case of nodes connected to voltage sources, the node current is defined as the sum of all the branch currents.

Example

```
HSIM > ni xiwire11.n0_4
Node xiwire11.n0_4 (18):
Node current=6.95149e-006.
```

nid

Gets node index from node name.

Syntax

```
nid node_name
```

Description

`nid` prints the node index corresponding to the given node name.

Example

```
HSIM> nid xalu3.xlatch2.q  
888
```

nm

Prints nodes merged to a specified node.

Syntax

```
nm node
```

Description

`nm` prints nodes merged to the specified node.

nname

Gets node name from node index.

Syntax

```
nname node_index
```

Description

`nname` prints the node name corresponding to the given node index.

Example

```
HSIM> nname 888  
xalu3.xlatch2.q
```

nv

Gets node voltage and capacitance by node name.

Syntax

```
nv node_name
```

Description

`nv` prints the node voltage, slope of node voltage waveform, and node capacitance for the given node name. Each value is evaluated at the most recent time when the node voltage is last updated. The data are printed.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

Example

If the current time is 100 ns and the printed voltage is 2.5V with printed time 95 ns, this implies that the node voltage stays unchanged at 2.5V between 95 ns and 100 ns. The capacitance report contains both constant and voltage dependent capacitances. The constant capacitance is the sum of constant capacitances connected to the node; and the voltage dependent capacitance is the sum of all MOSFET capacitances connected to the node. The `node_name` can contain the asterisk (*) wildcard character.

```
HSIM> nv xalu3.xlatch2.q
V=5, dV/dt=3528.7, time=5.278E-8
Constant capacitance=2.38ff, V-dependent capacitance=5.73ff
Total capacitance=8.11ff
```

op

Dumps operating point.

Syntax

```
op file_name <node_pattern>
```

Description

`op` prints the operating point state, such as the node voltage value, at current time into the specified file `file_name`. The optional `node_pattern` limits the printout nodes to those nodes matching the pattern, which may contain asterisk (*) wildcard character. The node voltage in the printout file has the following format:

```
.ic v(node_name)=node_voltage_value
```

The file can be included as an initial condition file in a later simulation which starts from the same operating condition listed in this file.

Example

```
HSIM> op file1
```

openlog

Opens interactive mode logfile.

Syntax

```
openlog logfile_name <-a>
```

Description

`openlog` opens the interactive mode `logfile_name` which contains the record of interactive mode commands and the results reported by these commands until the logfile is closed by the `closelog` command. If `-a` is specified, the contents will be appended to any preexisting file of the same name. For detailed information about `closelog`, refer to [closelog on page 472](#).

Example

```
openlog file2
```

pt

Gets current time.

Syntax

```
pt
```

Description

`pt` prints the current time and the final simulation time.

Example

```
HSIM> pt
tnow=25.6ns, tmax=100ns
```

quit

Terminates simulation.

Syntax

```
quit
```

Description

`quit` terminates the simulation.

Example

```
HSIM> quit
```

rcf

Reads commands from the file.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

Syntax

```
rcf cmd_file
```

Description

`rcf` reads and executes each interactive mode command listed in the specified `cmd_file` file.

Example

```
HSIM> rcf command_file2
```

restart

Restarts simulation from saved database.

Syntax

```
restart time1 <file_name>
```

Description

`restart` restarts simulation at the earlier time specified by `time1`. The time is measured in nanoseconds. If the `file_name` is specified, it indicates the saved file from either the interactive `savesim` command or from the `.store` command in the netlist.

When restarting simulation inside the interactive mode at an earlier time, the output file will be created with `.rs#` suffix with # representing the number of restarts. For detailed information about `savesim`, refer to [savesim on page 499](#). For `.store`, refer to [Chapter 12, Simulation Output, .STORE/.RESTORE Statement on page 420](#).

Example

```
HSIM> restart 35 save_file
```

rv

Releases constant voltage from nodes.

Syntax

```
rv node1 <node2 ...>
```

Description

`rv` releases the node voltages from the value(s) fixed by the `fv` command. The node voltages are then determined by the regular simulation result. Refer to [fv on page 482](#) for details about `fv`.

Example

```
HSIM> rv    n2    n5
```

savesim

Saves simulation database.

Syntax

```
savesim file_name
```

Description

`savesim` saves the simulation database at the current time to the specified file. The `file_name` must be specified when this command is first used. If `file_name` is not specified, the previously entered filename will be used.

Example

```
HSIM> savesim save_file
```

For related information, refer to [Chapter 12, Simulation Output, .STORE/.RESTORE Statement on page 420](#).

stop <conditional>

Interrupts simulation under specified condition.

Syntax

```
stop node_name -val voltage_value <-edge r|f|c> <-from time1>
  <-to time2> <-cmf interactive_command_file>
stop node_name val=voltage_value <-edge=[r|f|c]>
  <from=time1> <to=time2> <cmf=interactive_command_file>
```

Description

This command pauses the simulation at the specified condition. `node_name` is the voltage node such as `n1` or `out2`, and `voltage_value` is the target voltage value of the node at which stop action occurs. `-edge` is in a format similar to the one in `.measure`.

Arguments

r

Indicates stop at the rising edge when the node voltage reaches the target value.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

f

Indicates stop at the falling edge.

c

Indicates stop at either the rising edge or the falling edge. If the edge parameter is not specified, the default is c.

-from

Defines the start of the time window in which to check the condition.

-to

Defines the end of the time window in which to check the condition.

Example

```
HSIM> stop n1 -val 1.5 -edge r -from 0 -to 100
```

The simulation will be paused if node n1 reaches 1.5V at a rising edge within the time window of 0 ns to 100 ns.

stop -at

Interrupts simulation.

Syntax

```
stop -at time1 <-cmf interactive_command_file>
stop at=time1 <cmf=interactive_command_file>
```

Description

If the `-cmf` option is specified, this command pauses the simulation at the specified time `time1` and executes the commands included in the `interactive_command_file`. Time is measured in nanoseconds.

Example

```
HSIM> stop -at 1050
```

stop -list and stop -delete

Lists and deletes interruption points.

Syntax

```
stop -list
stop -delete st_point1
```

Description

The `stop -list` statement lists all the existing stop points and the associated numbers. The `stop -delete` will remove the stop point.

Example

```
HSIM> stop -list  
HSIM> stop -delete
```

vni

Prints voltage source node.

Syntax

```
vni [-threshold value]
```

Description

`vni` prints the voltage source node with its current greater than or equal to the threshold value only.

trace_thru_on

Traces the circuit through conducting elements and identifies high impedance.

Syntax

```
trace_thru_on nodeName
```

Description

`trace_thru_on nodeName` determines whether a node is in the high-z mode then debugs leakage currents through OFF-elements in the node. It traces from the specified node through conducting MOSFETs and resistors until reaching either vsrc or OFF-elements.

Example

For the `xib.i609_d` node, `trace_thru_on xib.i609_d` produces the following printout.

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

```
at time (150000.0), from node xib.i609_d (id=173564) trace thru
ON elems
    thru xia.xi5.rpi608 to node xia.sa_la<33>
    thru xib.rpi619 to node xib.i40_d
    thru xib.rpi603 to node xib.i605_d
    thru xib.rpi608 to node xib.i609_d
    thru xib.rpi613 to node xib.i613_plus
    thru xib.rpi618 to node xib.i617_d
    thru xib.rpi606 to node xib.i606_plus
    thru xib.rpi624 to node xib.i624_minus
    thru xib.mi627 to node xib.i627_d
    thru xib.rpi630 to node xib.i705_d
    thru xib.mi719 to node xib.i17_minus
    thru xib.rpi17 to node xib.i633_d
    thru xib.mi633 to node vbl
```

tree

Prints hierarchical tree structure for specified subckt instances.

Syntax

```
tree [<subinst_pattern> ...]<-level num> <-a> <-def>
      <-count> <file filename> <subinst_pattern> -subckt
      instances
```

Arguments

-level num

Prints the subckt instances located in the hierarchies between the current level where the specified instance is located, to num-1 levels below the current level.

-a

Prints the full hierarchical instance name.

-def

Prints the subckt name of the instance.

-count

Prints the number of elements inside the specified instance.

-file *filename*

Prints the results in a file named *filename*. Without this option, the results are displayed on the screen.

Examples

The following example prints the complete hierarchical tree.

```
> xtop
> +---x1
> +---xa
> +---xb
> +---x2
> +---x1
> +---xa
> +---xb
> +---xc
```

The following example prints hierarchical trees of xtop.x1 & xtop.x2 down to next hierarchy tree xtop.x* -level 2.

```
> x1
> +---xa
> +---xb
> x2
> +---x1
```

In the following example, tree -a -def prints the complete tree with full hierarchical instance name and subckt definition name.

```
> xtop (cir)
> xtop.x1 (cir1)
> xtop.x1.xa (and2)
> xtop.x1.xb (or2)
> xtop.x2 (cir2)
> xtop.x2.x1 (cir3)
> xtop.x2.x1.xa (nand2)
> xtop.x2.x1.xb (nand2)
> xtop.x2.x1.xc (nand2)
```

Feedback

Chapter 16: Interactive Mode Debugging

Interactive Mode Commands

Timing and Power Analysis

Describes HSIM timing checking (setup, hold, pulse width, edge, checking windows, bisection optimization) and power analyses (DC path, excessive current, excessive rise/fall, high impedance node) tools that work within the netlist file or in a separate file included in the netlist file.

- [Overview of Timing and Power Analysis](#)
- [Timing Checking](#)
- [Power Checking](#)
- [Activity Checking](#)

Overview of Timing and Power Analysis

HSIM provides a set of commands that perform timing and power analyses. Each command must be specified either within the netlist file or in a separate file included in the netlist file using the .include option. (See [Chapter 8, Input Netlist, .include on page 331](#) for information about the .include statement.)

Checks use the statements shown below:

- .tcheck: Timing check command
- .pcheck: Power check command

If there is either a timing or power check error, it is reported in one of the following files:

- hsim.chk
- out_file.chk: If -o out_file is specified when invoking HSIM.

Chapter 17: Timing and Power Analysis

Timing Checking

If separate_file is set in the command, the report file name will include the check name and the title name. For example:

hsim.edge_t1.chk

Timing Checking

This section provides descriptions of the HSIM timing checks:

- [Setup Time Check](#)
 - [Hold Time Check](#)
 - [Pulse Width Check](#)
 - [Timing Edge Check](#)
 - [Timing Check Windows](#)
 - [Bisection Optimization](#)
-

Setup Time Check

Checks whether the reference node transition time minus signal node transition time is less than the setup time.

Syntax

```
.tcheck title_name setup sig_name edge_type ref_name  
ref_edge_type setup_time <inst=sub_inst_name>  
<subckt=sub_name> <window>window_limit>  
<vlth=logic_low_voltage> <vhth=logic_high_voltage>  
<refvlth=ref_logic_low_voltage>  
<refvhth=ref_logic_high_voltage> <separate_file=0|1>
```

Parameters

Table 83 Setup Time Check Parameters

Parameter	Description
title_name	Defines the title name of this setup time check. Setup time errors during this timing check are listed in the timing/power check error file. The name of the setup time error starts with this title name.
sig_name	Defines signal node name(s) that can be the node name of a single node or a node name containing an asterisk (*) wildcard character that represents a group of node names. The node name with a wildcard character must be specified as v(node_name) or 'node_name', because in the HSIM netlist parser and in SPICE syntax all characters after wildcards are treated as comments and are ignored. When using the subckt parameter, sig_name is the node name inside the subcircuit—it must not be the full path name. If the subckt parameter is not used, then sig_name should be the full path name.
ref_name	Defines the reference node name. ref_name is the node name of a single node; it must not contain an asterisk (*) wildcard character. When using the subckt parameter, the ref_name is the node name inside the subcircuit which must not be the full path name. If the subckt parameter is not used, then the ref_name must represent the full path name.
edge_type	Defines the permissible state transition type for signal node with r, f, and x. Notice that: <ul style="list-style-type: none"> ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition—from 0 to 1 or from 1 to 0.

Chapter 17: Timing and Power Analysis

Timing Checking

Table 83 Setup Time Check Parameters (Continued)

Parameter	Description
ref_edge_type	Defines the permissible state transition type for reference node with r, f, and x. Notice that: <ul style="list-style-type: none"> ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition—from 0 to 1 or from 1 to 0.
setup_time	Defines the setup time. A setup timing error occurs when the time difference between the permissible state transition time of the reference node and the permissible state transition time of the signal node is less than setup_time. If setup_time is negative, window_limit must be specified.
inst=sub_inst_name	Specifies sub-circuits by the instance's full hierarchical name, unlike the subckt option which specifies sub-circuits by the definition name. Wildcard characters are allowed in this option. See Example to know more. Note: The two sub-circuit options, subckt and inst cannot be used simultaneously.
sub_name	When this parameter is specified, the operation is performed on all instances of the particular subcircuit. When using the subckt parameter, sig_name and ref_name are the node names inside the subckt sub_name in which both sig_name and ref_name must not contain a wildcard character.
window_limit	When window_limit is specified, the setup timing check error occurs when the signal node has a permissible transition at the time interval ($t_{ref} - setup_time, t_{ref} + window_limit$). If setup_time is negative, window_limit must be specified. If setup_time is positive and window_limit is not specified, the setup timing check error occurs when the signal node has a permissible transition at the time interval ($t_{ref} - setup_time, t_{ref}$).

Table 83 Setup Time Check Parameters (Continued)

Parameter	Description
logic_low_voltage	Represents the threshold voltage of logic state 0 (zero). The signal node is in logic 0 state if its node voltage is lower than logic_low_voltage.
logic_high_voltage	Represents the threshold voltage of logic state 1 (one). The signal node is in logic 1 state if the signal node voltage is higher than logic_high_voltage.
ref_logic_low_voltage	Represents the low logic threshold voltages for the reference node. When logic_low_voltage is specified, ref_logic_low_voltage is defined to the same value if it is not separately defined by refvlth.
ref_logic_high_voltage	Represents the high logic threshold voltages for the reference node. When logic_high_voltage is specified, ref_logic_high_voltage follows the same value if it is not separately defined by refvhth.
separate_file=0 1	Specifies whether a separate output file is needed: <ul style="list-style-type: none"> ▪ 0: [default] No separate output file ▪ 1: This separate file is used to keep the output: hsim.setup_<title_name>.chk

Note: If vhth=logic_high_voltage and vlth=logic_low_voltage are not specified, HSIM uses 0.7*HSIMVDD for vhth and 0.3*HSIMVDD for vlth.

Also, the setup timing check only checks the transition near the reference node edge.

Examples

```
.tcheck check1 setup data1 x clk f 2n
.tcheck check2 setup v(x1.*.data1) x clk f 2n
.tcheck check2 setup 'x1.*.data1' x clk f 2n
```

A setup time error occurs when any state transition at node data1 occurs less than 2 ns before the fall transition of the node clk. The error is reported following the title name of check1. Similar checks are performed for all nodes that match the pattern x1.*.data1. The setup timing error, if any, is reported in

Chapter 17: Timing and Power Analysis

Timing Checking

the timing/power check file following the title name of check2. The last two examples clearly illustrate different ways to specify the sig_name information.

```
.tcheck      check33 setup d x clk r 3n subckt=dff
```

subckt is specified in which d and clk are the node names inside the subckt. The setup time check reports error for every dff subcircuit instance if the state transition at data pin d occurs less than 3 ns before the rise transition at the clk pin within the same dff subcircuit instance.

```
.tcheck check3 setup qn f clk r -0.5n window=10n
```

A setup time error, following the title name of check3, is reported when a fall state transition at node qn occurs 0.5 ns after the node clk has a rise transition, and this fall state transition at qn occurs no later than 10 ns after the rise transition at node clk.

The inst option can be illustrated with the help of a test case, where a subcircuit “inv” is instantiated by x1, x2, and x3. With the subckt option, all three sub-circuit instances x1, x2, and x3 will be included, but with the help of the inst option, you can choose any of the three sub-circuit instances.

The limitation which is inherited from the subckt option and is applicable for the inst option is, wildcard characters cannot be used in the node name and the reference node name.

Example 46 inst option

```
.tcheck check33 setup d x clk r 3n inst=x1.x2
```

Hold Time Check

Checks whether signal node transition time minus reference node transition time is less than hold_time.

Syntax

```
.tcheck title_name hold sig_name edge_type ref_name  
    ref_edge_type hold_time <inst=sub_inst_name>  
    <subckt=sub_name> <window>window_limit>  
    <vlth=logic_low_voltage> <vhth=logic_high_voltage>  
    <refvlth=ref_logic_low_voltage>  
    <refvhth=ref_logic_high_voltage> <separate_file=0|1>
```

Parameters

Table 84 Hold Time Check Parameters

Parameter	Description
title_name	Defines the title name of this hold time check. Hold time errors during this timing check are listed in the timing/power check error file and starts with this title name.
sig_name	Defines signal node name(s) that can be the node name of a single node, or a node name containing the asterisk (*) wildcard character that represents a group of node names. The node name with wildcard character must be specified as v(node_name) or 'node_name', because in the HSIM netlist parser and in SPICE syntax all characters after '*' are treated as comments and are ignored. When using the subckt parameter, sig_name is the node name inside the subcircuit—it must not be the full path name. If the subckt parameter is not used, then sig_name should be the full path name.
ref_name	Defines the reference node name. ref_name is the node name of a single node, and it can not contain an asterisk (*) wildcard character. When using the subckt parameter, the ref_name is the node name inside the subcircuit—it must not be the full path name. If the subckt parameter is not used, then the ref_name must be the full path name.
edge_type	Defines the permissible state transition type for the signal node with r, f, and x. Notice that: <ul style="list-style-type: none"> ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition—from 0 to 1 or from 1 to 0.
ref_edge_type	Defines the permissible state transition type for the reference node with r, f, and x. Notice that: <ul style="list-style-type: none"> ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition—from 0 to 1 or from 1 to 0.

Chapter 17: Timing and Power Analysis

Timing Checking

Table 84 Hold Time Check Parameters (Continued)

Parameter	Description
hold_time	Defines the hold time. A hold timing error occurs when the time difference between the permissible state transition time of signal node and the permissible state transition time of reference node is less than hold_time. If hold_time is negative, window_limit must be specified.
inst=sub_inst_name	Specifies sub-circuits by the instance's full hierarchical name, unlike the subckt option which specifies sub-circuits by the definition name. Wildcard characters are allowed in this option. See Example to know more. Note: The two sub-circuit options, subckt and inst cannot be used simultaneously.
sub_name	When this parameter is specified, the operation is performed on all instances of the particular subcircuit. When using the subckt parameter, sig_name and ref_name are the node names inside the subckt sub_name in which both sig_name and ref_name must not contain a wildcard character.
window_limit	When window_limit is specified, the hold timing check error occurs when the signal node has a permissible transition at the time interval ($t_{ref} - window_limit, t_{ref} + hold_time$). If hold_time is negative, window_limit must be specified. If hold_time is positive and window_limit is not specified, the hold timing check error occurs when the signal node has a permissible transition at the time interval ($t_{ref}, t_{ref}+hold_time$).
logic_low_voltage	Defines the threshold voltage of logic 0 (zero) state of signal node. The signal node is in logic 0 state if its node voltage is less than logic_low_voltage.
logic_high_voltage	Defines the threshold voltage of logic 1 state of the signal node. The signal node is in logic 1 state if the signal node voltage is greater than logic_high_voltage.

Table 84 Hold Time Check Parameters (Continued)

Parameter	Description
ref_logic_low_voltage	Defines the low logic threshold voltage for the reference node. When logic_low_voltage is specified, ref_logic_low_voltage will be defined to the same value if it is not separately defined by refvhth.
ref_logic_high_voltage	Defines the high logic threshold voltage for the reference node. When logic_high_voltage is specified, ref_logic_high_voltage will be defined to the same value if it is not separately defined by refvhth.
separate_file=0 1	Specifies whether a separate output file is needed: <ul style="list-style-type: none"> ▪ 0: [default] No separate output file ▪ 1: The following separate file is used to keep the output: hsim.hold_<title_name>.chk

Note: The hold timing check only checks the transition near the reference node edge.

Examples

```
.tcheck check3 hold data x clk f 2.1n
```

A hold time error occurs when any state transition at node data occurs at the time interval ($t, t+2.1$ ns), where t is the time when node clk has a fall state transition. The error is reported following the title name of check3.

```
.tcheck check4 hold qn f clk r -0.5n window=10n
```

A hold time error, following the title name of check4, is reported when a fall state transition at node qn occurs at the time interval ($t -10$ ns, $t -0.5$ ns) in which where t is the time when the node clk has a rise transition.

The `inst` option can be illustrated with the help of a test case, where a sub-circuit “inv” is instantiated by x1, x2, and x3. With the `subckt` option, all three

Chapter 17: Timing and Power Analysis

Timing Checking

sub-circuit instances, x1, x2, and x3, will be included, but with the help of the `inst` option, you can choose any of the three sub-circuit instances.

The limitation which is inherited from the `subckt` option and is applicable for the `inst` option is, wildcard characters cannot be used in the node name and the reference node name.

Example 47 inst option

```
.tcheck check33 hold x clk r 3n inst=x1.x2
```

Pulse Width Check

Checks whether the pulse width (time difference between rise and fall state transitions) of the specified node meets the required range.

Syntax

```
.tcheck title_name pulsew sig_name low_min_time low_max_time  
high_min_time high_max_time <inst=sub_inst_name>  
<subckt=sub_name> <vlth=logic_low_voltage>  
<vhth=logic_high_voltage> <vlow=low_min_voltage>  
<vhigh=high_max_voltage> <separate_file=0|1>
```

Parameters

Table 85 Pulse Width Check Parameters

Parameter	Description
title_name	Defines the title name of this pulse width check. Any violation against the required pulse width range resulted from this check will be listed in the timing/power check error file, and starts with this title name.
sig_name	Defines signal node name(s) which can be the node name of a single node or a node name containing an asterisk (*) wildcard character representing a group of node names. The node name with a wildcard character must be specified as v(node_name) or node_name, because in the HSIM netlist parser and in SPICE syntax all characters after '*' are treated as comments and are ignored. When using the subckt parameter, sig_name is the node name inside the subcircuit -- it must not be the full path name. If the subckt parameter is not used, then sig_name must be the full path name.
logic_low_voltage	Defines the threshold voltage of logic 0 stat. The node has a logic 0 state if its node voltage is lower than logic_low_voltage.
logic_high_voltage	Defines the threshold voltage of logic 1 state of the node. The node has a logic 1 state if the node voltage is higher than logic_high_voltage.
low_min_time	Defines the minimum value of the low-state pulse. Each low-state pulse (the time period the node stays at the logic 0 state) is required to be greater than low_min_time and less than low_max_time.
low_max_time	Defines the maximum value of the low-state pulse. Each low-state pulse (the time period the node stays at the logic 0 state) is required to be greater than low_min_time and less than low_max_time.

Chapter 17: Timing and Power Analysis

Timing Checking

Table 85 Pulse Width Check Parameters (Continued)

Parameter	Description
inst=sub_inst_name	Specifies sub-circuits by the instance's full hierarchical name, unlike the <code>subckt</code> option which specifies sub-circuits by the definition name. Wildcard characters are allowed in this option. See Example to know more. Note: The two sub-circuit options, <code>subckt</code> and <code>inst</code> cannot be used simultaneously.
sub_name	When this parameter is specified, the operation is performed to all instances of the particular subcircuit. When using the <code>subckt</code> <code>sub_name</code> parameter, <code>sig_name</code> and <code>ref_name</code> are the node names inside the <code>subckt</code> <code>sub_name</code> in which both <code>sig_name</code> and <code>ref_name</code> must not contain a wildcard character.
high_min_time	Defines the minimum value of the high-state pulse. Each high-state pulse (the time period the node stays at the logic 1 state) is required to be greater than <code>high_min_time</code> and less than <code>high_max_time</code> .
high_max_time	Defines the maximum value of the high-state pulse. Each high-state pulse (the time period the node stays at the logic 1 state) is required to be greater than <code>high_min_time</code> and less than <code>high_max_time</code> .
low_min_voltage	Defines the threshold voltage of logic 0 of the node. If defined, the node has a logic 0 state only when its minimum voltage during the pulse is lower than <code>low_min_voltage</code> . The default is the value of <code>logic_low_voltage</code> .
high_max_voltage	Defines the threshold voltage of logic 1 of the node. If defined, the node has a logic 1 state only when its maximum voltage during the pulse is higher than <code>high_max_voltage</code> . The default is the value of <code>logic_high_voltage</code> .
separate_file=0 1	Specifies whether a separate output file is needed: <ul style="list-style-type: none">▪ 0: [default] No separate output file▪ 1: The following separate file is used to keep the output: hsim.pulsew_<title_name>.chk

Examples

```
.tcheck check5 pulsew data[3] 8n 11n 7n 9n
```

A pulse width violation error, following the title name of check5, is reported when the low pulse width at node data[3] is less than 8 ns or greater than 11 ns; or when the high pulse width is less than 7 ns or greater than 9 ns.

The default value is used in this example because vlth is not defined. The default values are:

- vhth: $3V \times 0.7 = 2.1$ volts
- vlth: $3V \times 0.3 = 0.9$ volts

```
.tcheck chk_pwidth pulsew v(*) 0.3n 1000n 0.3n 1000n vlth=0.4  
vhth=0.6 vlow=0.15 vhigh=1.0
```

This command checks the pulse width of all the nodes in the circuit. A violation will be reported for a node if the low pulse width is less than 0.3 ns or greater than 1000 ns; or when the high pulse width is less than 0.3 ns or greater than 1000 ns. A pulse is low when the node voltage is less than 0.4V, and the lowest voltage in a pulse must be lower than 0.15V. A pulse is high when the node voltage is higher than 0.6V, and the highest voltage in a pulse must be higher than 1.0V.

Note: Using tcheck chk_pwidth pulsew v(*) 0.3n 1000n 0.3n 1000n vlth=0.4 vhth=0.6 results in small voltage dips to be reported. For example, a node moving from 0.5 volts dips to 0.35 volts and comes up still reports a pulsew error.

The `inst` option can be illustrated with the help of a test case, where a sub-circuit “inv” is instantiated by x1, x2, and x3. With the `subckt` option, all three sub-circuit instances, x1, x2, and x3, will be included, but with the help of the `inst` option, you can choose any of the three sub-circuit instances.

The limitation which is inherited from the `subckt` option and is applicable for the `inst` option is, wildcard characters cannot be used in the node name and the reference node name.

Example 48 inst option

```
.tcheck check33 pulsew d x clk r 3n inst=x1.x2
```

Timing Edge Check

Checks the time delay between two specified nodes and reports error when the delay doesn't meet the specified range.

Syntax

```
.tcheck title_name edge sig_name edge_type ref_name  
    ref_edge_type min_time max_time <inst=sub_inst_name>  
    <subckt=sub_name> <window>window_limit>  
    <vlth=logic_low_voltage>  
<vhth=logic_high_voltage> <refvlth=ref_logic_low_voltage>  
<refvhth=ref_logic_high_voltage> <trigger=0|1|2>  
    <separate_file=0|1>
```

Parameters

Table 86 Timing Edge Check Parameters

Parameter	Description
title_name	Defines the title name of this timing edge check. Every violation resulting from this timing check is listed in the timing/power check error file and starts with this title name.
sig_name	Defines signal node name(s) which can be the node name of a single node or a node name containing an asterisk (*) wildcard character representing a group of node names. The node name with wildcard character must be specified as v(node_name) or 'node_name', because in the HSIM netlist parser and in SPICE syntax, all characters after '*' are treated as comments and are ignored.
ref_name	When using the subckt parameter, sig_name is the node name inside the subcircuit, and must not be the full path name. If the subckt parameter is not used, then the sig_name should be the full path name When using the subckt parameter, the ref_name is the node name inside the subcircuit—it must not be the full path name. If the subckt parameter is not used, then the ref_name should be the full path name.
edge_type	Defines the permissible state transition type for the signal node with r, f, or x. Notice that: <ul style="list-style-type: none"> ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition—from 0 to 1 or from 1 to 0.

Chapter 17: Timing and Power Analysis

Timing Checking

Table 86 Timing Edge Check Parameters (Continued)

Parameter	Description
ref_edge_type	Defines the permissible state transition type for the reference node with r, f, or x. Notice that: <ul style="list-style-type: none"> ▪ r indicates timing check is only performed when the state transition is from 0 (zero) to 1 (one). ▪ f indicates the 1 to 0 transition. ▪ x indicates any state transition—from 0 to 1 or from 1 to 0.
min_time	Defines the lower boundary of the timing edge difference between the signal and reference nodes. A timing edge error is reported when the timing edge difference is less than min_time. The timing edge difference is calculated only for the pair of permissible state transitions at the signal node and the reference node.
max_time	Define the upper boundary of the timing edge difference between the signal and reference nodes. A timing edge error is reported when the timing edge difference is greater than max_time. The timing edge difference is calculated only for the pair of permissible state transitions at the signal node and the reference node.
window_limit	Eliminates a timing edge error report if the timing edge difference exceeds window_limit. window_limit is optional.
logic_low_voltage	Defines the threshold voltage of the logic 0 (zero) state of the signal node. The signal node has a logic 0 state if its node voltage is lower than logic_low_voltage.
logic_high_voltage	Defines the threshold voltage of the logic 1 (one) state of the signal node. The signal node has a logic 1 state if the signal node voltage is higher than logic_high_voltage.
ref_logic_low_voltage	Defines the low logic threshold voltage for the reference node. When logic_low_voltage is specified, ref_logic_low_voltage will be defined to the same value if not separately defined by refvhth.
ref_logic_high_voltage	Defines the high logic threshold voltage for the reference node. When logic_high_voltage is specified, ref_logic_high_voltage will be defined to the same value if not separately defined by refvhth.

Table 86 Timing Edge Check Parameters (Continued)

Parameter	Description
inst=sub_inst_name	<p>Specifies sub-circuits by the instance's full hierarchical name, unlike the subckt option which specifies sub-circuits by the definition name. Wildcard characters are allowed in this option. See Example to know more.</p> <p>Note: The two sub-circuit options, subckt and inst cannot be used simultaneously.</p>
sub_name	<p>When this parameter is specified, the operation is performed on all instances of the particular subcircuit. When using the subckt parameter, sig_name and ref_name are the node names inside the subckt sub_name in which both sig_name and ref_name must not contain a wildcard character.</p>
trigger value=0 1 2	<p>Defines the condition to trigger the timing edge check.</p> <ul style="list-style-type: none"> ▪ 0 [default] If the value is 0, any permissible state transition at either signal node or reference node triggers the timing edge check. ▪ 1 Only the permissible state transition at the signal node triggers the check. ▪ 2 Only the permissible state transition at the reference node triggers the check. Trigger value is optional.
separate_file=0 1	<p>Specifies whether a separate output file is needed:</p> <ul style="list-style-type: none"> ▪ 0 [default] No separate output file. ▪ 1 The following separate file is used to keep the output the following: hsim.edge_<title_name>.chk

Description

Note that this command compares the edge time difference between the most recent target signal transition and reference signal transition only. The qualified target-reference signal pair can vary depending on their transition order. For example, if there is a target signal transition this check looks for the reference signal transition, which happens prior and also most recently to this target signal transition. Similarly, if there is a reference signal transition this check looks for the target signal transition, which happens prior and also most recently to this reference signal transition. See the following example.

Chapter 17: Timing and Power Analysis

Timing Checking

Examples

```
.tcheck check6 edge data x ctrl r 2n 5n
```

When node data has a state transition, such as at time t2, the time edge difference t2-t1 must be within the range of 2 ns and 5 ns. Otherwise, a timing edge error is reported following the title name of check6. Time t1 is the most recent rise state transition time at node ctrl before time t2. When node ctrl has a rise state transition at time t4, the time edge difference t4-t3 must be within the range of 2 ns and 5 ns. Otherwise a timing edge error is also reported. The time t3 is the most recent state transition time at node data before time t4.

```
.tcheck check7 edge data x ctrl r 2n 5n trigger=2
```

It is similar to the example above, except that the edge error check is triggered only by the rise state transition at node ctrl. Any edge error is reported following the title name of check7.

```
.tcheck check8 edge data x ctrl r 2n 5n trigger=2 window=10n
```

When node ctrl has a rise state transition at time t1, an edge error is reported (following the title name of check8) only when t1-t2 is less than 2 ns or is less than 10 ns but greater than 5 ns. The time t2 is the most recent state transition time at node data before time t1.

The `inst` option can be illustrated with the help of a test case, where a sub-circuit “inv” is instantiated by x1, x2, and x3. With the `subckt` option, all three sub-circuit instances, x1, x2, and x3, will be included, but with the help of the `inst` option, you can choose any of the three sub-circuit instances.

The limitation which is inherited from the `subckt` option and is applicable for the `inst` option is, wildcard characters cannot be used in the node name and the reference node name.

Example 49 inst option

```
.tcheck check33 edge d x clk r 3n inst=x1.x2
```

Timing Check Windows

Defines the timing windows for the timing check commands.

Syntax

```
.tcheck window start_time1 <stop_time1 <start_time2  
<stop_time2 ...>>
```

Description

.tcheck window defines the timing windows for the timing check commands. All timing check commands use the same set of windows specified by this command. If this command is not specified, the default window for all timing check commands will be from the beginning of simulation to the end of simulation.

Multiple timing check windows can be specified by providing multiple pairs of start and stop time values. If the stop time is not provided in the last window, it will be extended to the end of simulation.

Example

```
.tcheck window 10n 20n 110n 120n 210n
```

In this example, the command specifies three timing check windows. The first window is from 10 ns to 20 ns, the second window is from 110 ns to 120 ns, and the third window is from 210 ns until the end of simulation.

Bisection Optimization

HSIM supports bisection optimization (see [HSIMBISECTION on page 66](#)). The bisection results are printed in the .optz file. The results of the .measure and .print statements of the last bisection iteration are stored in the .mt and .fsdb/.out files, respectively. Several statements are required to perform bisection optimization.

- .model statement
- .param statement
- .tran statement

.model

Syntax

```
.model opt_model_name opt method=bisection|passfail  
      relin=tolerance
```

The optimization model reference name is opt_model_name. The keyword opt indicates that this particular .model statement is for bisection optimization usage. The same name is used in the corresponding .tran statement. method specifies the method to use in the bisection optimization—a valid value is either bisection or passfail with bisection being the default.

Chapter 17: Timing and Power Analysis

Timing Checking

relin specifies the error tolerance. Its default value is 10e-3. Bisection optimization continues halving the target region until the interval between successive test values meets the criteria shown in the following equation:

$$\frac{|X_n - X_{n-1}|}{X_{upper} - X_{lower}} \leq relin$$

Then, bisection optimization reports the value of X_n associated with the measured value that passed when method=passfail or the measure value minus the goal that is less than zero when method=bisection.

.param**Syntax**

```
.param param_name=optxxx(init_value, low_value,  
upper_value)
```

Description

The name of the parameter used in bisection optimization is param_name. optxxx is the selected optimization parameter reference name—xxx can be replaced with a suitable choice. The same optxxx name is referenced in the corresponding .tran statement. The initial value, the lower boundary, and the upper boundary of the parameter are specified as init_value, low_value, and upper_value, respectively.

Example

```
.param param_name=alterxxx(value1, value2, ., valuen)
```

In the previous syntax, alterxxx is used with alterparam in .tran command. Please see the following .tran syntax.

.tran**Syntax**

```
.tran steptime stoptime sweep optimize=optxxx  
result=measure_var model=opt_model_name  
<alterparam=alterxxx>
```

Description

The steptime and stoptime are specified as *steptime* and *stoptime*, respectively. optxxx is the same optimization parameter reference name in the corresponding .param statement. The result parameter specifies the measure variable defined in .measure statement. The model parameter is specified with

the same model optimization reference name in the corresponding .model statement.

.alterparam

Description

The alterparam will accomplish the following:

- Alter the parameter value which is defined in alterxxx.
- Prevent HSIM from re-reading the netlist making the simulation faster.

Example

```
Vin ck 0 pwl
+0 0
+'19n' 0
+'t_time+19n' 5
+bisectparam=1
Ven d 0 pwl
+'0' 0
+'delaytime' 0
+'t_time2+delaytime' 5
+ bisectparam=1
.param delaytime=opt(0.0n, 0.0n, 20.0n)
.param t_time=alter1(0.1n, 0.1n, 0.1n, 0.5n, 0.5n, 0.5n, 1n, 1n)
.param t_time2=alter2(0.1n, 0.5n, 1n, 0.1n, 0.5n, 1n, 0.1n, 0.5n, 1n)
.tran 0.1n 40.0n
+ sweep
+ optimize=opt
+ result=maxvout
+ model=optmod
+ alterparam=alter1
+ alterparam=alter2
.model optmod opt
+ method=bisection
+ relin=1.0e-3
.measure tran ck_slope param=t_time
.measure tran d_slope param=t_time2
.measure tran mt_delaytime param=delaytime
.measure tran maxvout max v(q) goal=5
.measure tran setuptrig trig v(d) val=2.5 rise=1
+ targ v(ck) val=2.5 rise=1
```

This test case extracts the setup time between data pin d and clock pin ck. In the .tran command there are two alterparams, alter1 and alter2. Both

Chapter 17: Timing and Power Analysis

Power Checking

alterparams have nine values allowing HSIM to extract the setup time under nine different conditions. HSIM generates nine .mt files.

```
dff.mt.a0:  
Measurement results:  
ck_slope=1.00000000000e-010  
d_slope=1.00000000000e-010  
mt_delaytime=1.87695300000e-008  
maxvout=5.00000000000e+000 at=4.00000000000e-008  
setuptime=2.30000000000e-010 targ=1.90500000000e-008  
trig=1.88200000000e-008  
dff.mt.a1:  
Measurement results:  
ck_slope=1.00000000000e-010  
d_slope=5.00000000000e-010  
mt_delaytime=1.84960900000e-008  
maxvout=5.00000000000e+000 at=4.00000000000e-008  
setuptime=3.04000000000e-010 targ=1.90500000000e-008  
trig=1.87460000000e-008  
...  
dff.mt.a8:  
Measurement results:  
ck_slope=1.00000000000e-009  
d_slope=1.00000000000e-009  
mt_delaytime=1.87304700000e-008  
maxvout=5.00000000000e+000 at=4.00000000000e-008  
setuptime=2.70000000000e-010 targ=1.95000000000e-008  
trig=1.92300000000e-008
```

bisectparam=1 must also be added to the PWL voltage source to prevent HSIM from re-reading the netlist. HSIM only supports this feature in PWL voltage source as described in the following section.

Power Checking

This section describes the power checks:

- [DC Path Check](#)
- [Excessive Current Check](#)
- [Excessive Rise/Fall Time Check](#)
- [High Impedance Node Check](#)
- [Power Check Windows](#)

DC Path Check

This command checks the DC current path(s) among voltage sources in the circuit. All reported DC current paths are written into a separate file. This prevents mixing data with other violations reported by other timing and power checks.

Syntax

```
.pcheck title_name dcpath <ith=threshold_current> <node1
    <node2 ...>> <period=period_time | delay=delay_time>
    <start=start_time1 <stop=stop_time1 <start=start_time2
    <stop=stop_time2 ...>>> <num=warn>>
.pcheck title_name dcpath <ith=threshold_current> <node1
    <node2 ...>> at=t1 <at=t2 ...> <num=warn>
```

Note: Do not specify a Z-state within PWLZ when you perform a depth check. If a PWLZ source and Z-state are specified within PWLZ, HSIM does not consider PWLZ as a voltage source because there might be a period of time when PWLZ is not driven by voltage source elements. The dcpath check has no indicators of the starting VSRC node locations, and because of this situation, the reported path might not be a legitimate path.

The .pcheck command output supports the plain format by setting “cckOutput format=plain” in the CCK command file.

Chapter 17: Timing and Power Analysis

Power Checking

Parameters*Table 87 DC Path Check Parameters*

Parameter	Description
title_name	Each DC path is reported in the file title_name.
ith	Defines threshold_current, the value of the threshold current, which has a default value of 50 uA.
node1, node2 ... nodeN	The specified names define the nodes that DC current path checking will be performed between these nodes. The DC path search starts from any node specified in this node list and ends when it reaches another node in the list. If no node is specified, then HSIM reports dc current path(s) between any pair of voltage source nodes.
period	When specified, the DC path is checked for every period_time starting from the start time of each specified window defined by start_time and stop_time. If the period is not specified, the checking point will depend on the change points of PWL input sources. For better control of pcheck dcpath, always specify period= when start= is issued.
delay	When specified, the DC current path is checked at each time defined as t+delay_time. t is the time an input voltage source changes to a new voltage level. period and delay cannot be used simultaneously.
at	When specified, the DC current path is checked at the time defined by at=t1.
start	Specifies the start_time of the window. When specified, the checking point begins at this time. For better control of pcheck dcpath, always specify period= when start= is issued.
stop	Specifies the stop_time of the window.
num	Limits the number of warnings generated by a particular analysis command defining it. Only the specified number of warnings is generated for a particular analysis command as defined by that command. The default value is 300. If specified number is set to less than or equal to 0, the warnings are unlimited.

Output File Name

If you specify -o out_file when invoking HSIM, the output file name is out_file.dcpath_title_name.chk. Otherwise, the default name is hsim_dcpath_title_name.chk.

Examples

```
.pcheck result1 dcpath ith=1u start=5n stop=110n
```

If the input source is defined as follows:

```
vin in 0 pwl (0 0 10n 0 11n 6.5 12n 6.5 12.5n 0)
```

and the pcheck command is issued without using -period then the following occurs:

The first checking point will be at time=5ns specified in start=

It will also check 11n and 12.5n point, not 12n point for it has the same value of point 11n.

Based on the analyses of the previous bullets, the checking points in this example are at time 5n, 11n, and 12.5n respectively.

```
.pcheck dc1 dcpath ith=1e-6 vdd gnd delay=5n start=10n stop=210n
```

.pcheck checks each DC current path between vdd and gnd every 5 ns after a voltage level change at any input voltage source. The dc current path is checked only during the time window between 10 ns and 210 ns. The dc current path is reported in the dc1 file hsim.dcpath_dc1.chk if the dc path current exceeds 1 uA when checked.

```
.pcheck dc2 dcpath ith=1e-6 vcc vss period=10n start=10n stop=210n
```

.pcheck checks each DC current path between vcc and vss in every 10 ns starting from simulation time at 10 ns and simulation time at 210 ns (included). The dc current path is reported in the dc2 file hsim.dcpath_dc2.chk if the dc path current exceeds 1 uA when checked.

```
.pcheck dc3 dcpath vcc vss at=130n at=150n
```

.pcheck checks each DC current path between vcc and vss at 130 ns time and 150 ns time. The DC current path is reported in the file dc3 if the dc path current exceeds the default value of 50 uA at 130 ns or at 150 ns time.

Excessive Current Check

The excessive element-current check exi checks when the current of specified element(s) exceeds a threshold value.

Syntax

```
.pcheck title_name exi elem1 <elem2...>
    <ith=threshold_current> <tth=exi_time>
    <start=start_time1 <stop=stop_time1 <start=start_time2
    <stop=stop_time2 ...>>> <level=val1> <factor=val2>
    <separate_file=0|1>
```

Parameters

Table 88 Excessive Current Check Parameters

Parameter	Description
title_name	Defines the title name of this excessive current check. Each excessive current report result from this check is listed in the timing/power check error file, and starts with this title name.
elem1, elem2 . . . elemN	The excessive current check is applied to each specified element elem1, elem2 . . . elemN. The element name can include the asterisk (*) wildcard character – the excessive current check applies to each element with matching pattern. The element name with a wildcard character must be specified as i(elem_name) or 'elem_name' because in the HSIM netlist parser and in SPICE syntax, all characters after (*) are treated as comments and are ignored.
ith	Defines threshold_current, the value of the threshold current. An element is reported to have excessive current if its element current exceeds threshold_current for a time duration longer than exi_time.
tth	Defines exi_time, the time duration. An element is reported to have excessive current if its element current exceeds threshold_current for a time duration longer than exi_time
start, stop	Specifies time window(s). The excessive element current is checked at the time within the specified time window(s). If no time window is specified, the check is performed from the time 0 ns to the end of simulation.
level	level is effective when the wildcard character is specified in elem1 <elem2..>. For additional information, refer to the .print command in Chapter 12, Simulation Output, Output Control Statements on page 410 . The default value of val1 is -1.
factor	If factor is specified, pcheck will choose the larger one of threshold_current and val2*(w/l) as the new threshold_current. w is the transistor width and l is the transistor length.

Chapter 17: Timing and Power Analysis

Power Checking

Table 88 Excessive Current Check Parameters (Continued)

Parameter	Description
separate_file=0 1	Specifies whether a separate output file is needed: <ul style="list-style-type: none">▪ 0: [default] No separate output file▪ 1: This separate file is used to keep the output: hsim.exi_<title_name>.chk

Example

```
.pcheck largei exi m1 m2 i(x1.*) ith=1e-3 tth=3n start=100n
```

.pcheck checks if the elements m1, m2, and all elements within instance x1 have current greater than 1 mA for longer than 3 ns. Any element detected with excessive current after 100 ns is reported in the timing/power check file out_file.chk following the title name of largei. The name out_file is the output file prefix with the default name being HSIM.

Excessive Rise/Fall Time Check

The excessive rise/fall time check exrf checks if the specified node(s) have excessive rise and/or fall times.

Syntax

```
.pcheck title_name exrf node1 <node2 ...> <fanout=val2>
    <rise=rtime> <fall=ftime> <utime=utime1> <xsubckt=subx>
    <subinfo=0|1> <vlth=logic_low_voltage>
    <vhth=logic_high_voltage> <start=start_time1
    <stop=stop_time1 <start=start_time2 <stop=stop_time2
...>>> <separate_file=0|1>
```

Parameters

Table 89 Excessive Rise/Fall Time Check Parameters

Parameter	Description
title_name	Any excessive rise/fall time violation is reported in the timing/power check file xxxx.chk following the title name of title_name.
node1, node2 ...nodeN	Defines signal node name which can be the node name of a single node or a node name with the asterisk (*) wildcard character that represents a group of node names. The node name with wildcard character must be specified as v(node_name) or node_name because in the HSIM netlist parser and in SPICE syntax, all characters after '*' are treated as comments and are ignored.
fanout	Defines the fanout of driver nodes. If fanout is set to 1 (one), only the driver nodes with fanouts are checked to avoid unnecessary check on internal nodes within logic gates. If fanout is set to 0 (zero), both the internal node and the driver node are checked. The default value is 0. This parameter is optional. <ul style="list-style-type: none"> ▪ fanout=0: all nodes. ▪ fanout=1: nodes that have direct connection to transistor's gate. ▪ fanout=2: nodes that have direct connection to transistor's bulk.
rise	Defines the rise time of the signal as time duration t2-t1. t1 is the time when the rising signal voltage crosses the voltage level logic_low_voltage. t2 is the time when the same continuously rising signal voltage crosses the voltage level logic_high_voltage. The default value of rtime is 5 ns.
fall	The fall time (ftime) of the signal is defined as t4-t3. t3 is the time when the falling signal voltage crosses the voltage logic_high_voltage. t4 is the time when the same continuously falling signal voltage crosses the voltage logic_low_voltage. The default value is 5 ns.

Chapter 17: Timing and Power Analysis

Power Checking

Table 89 Excessive Rise/Fall Time Check Parameters (Continued)

Parameter	Description
utime	The U-state time (utime1) of the signal is defined as t6-t5. t5 is the time signal voltage enters the U state which is when the signal voltage is between logic_low_voltage and logic_high_voltage. t6 is the time the signal voltage leaves the U-state and the signal voltage is the same as the earlier voltage at t5 -- U-state represents an incomplete rise or fall transition. The default value is 5 ns.
xsubckt	Excludes the specified subcircuit from the check. Specify this option only when node1 is v(*). To exclude multiple subcircuits, use multiple xsuckt arguments: xsubckt=sub1 xsubckt=sub2 ... The maximum number of subcircuits to exclude is 1024.
subinfo	Specifies whether to print the subcircuit information in the report. The default is not to print the subcircuit information (0).
vlth	Defines the logic low state (logic_low_voltage).
vhth	Defines the logic high state (logic_high_voltage).
start, stop	The exrf check is performed within the time window defined by start_time1, stop_time1 . . . start_timeN, stop_timeN. An excessive rise/fall time violation is reported in the following conditions. <ul style="list-style-type: none"> ▪ The signal rise time exceeds rtime if it is specified following rtime, or utime1. ▪ The signal fall time exceeds ftime if it is specified following ftime, or utime1. ▪ The U-state duration exceeds utime1.
separate_file=0 1	Specifies whether a separate output file is needed: <ul style="list-style-type: none"> ▪ 0: [default] No separate output file ▪ 1: The following separate file is used to keep the output: hsim.exrf_<title_name>.chk

Example

```
.pcheck longrf exrf a1 a2 v(x1.x2.*) fanout=1
+ rise=3n fall=4n vlth=0.3 vhth=2.7 start=100n stop=1000n
```

This command checks if the signal voltages at nodes a1, a2, and the driver nodes that match the pattern of x1.x2.* have excessive rise and fall times. An exrf is checked between 100 ns and 1,000 ns. A violation is reported in the timing/power check file following the title name of longrf if the signal rise time exceeds 3 ns, or the signal fall time exceeds 4 ns, or the U-state time exceeds the default of 5 ns. The U-state is defined as the voltage between 0.3V and 2.7V.

```
.pcheck check1 exrf v(*) rise=0.05n fall=0.05n vlth=0.5 vhth=4.5
+ start=0 stop=100n subinfo=1
```

Because subinfo is set to 1, the subcircuit information is printed as shown in the following report file:

```
...
check1: node v(xnand.in1a) in 'nand' excessive fall time
from time 1.0095e-08 to 1.016e-08 (6.5e-11)
check1: node v(xnand.x2.n1) in 'nand.nor' excessive rise
time from time 0 to 1.017e-08 (1.017e-08)
...
```

High Impedance Node Check

This command checks for the high impedance state in the circuit.

Syntax

```
.pcheck title_name zstate node1 <node2 ...> <fanout=val2>
<xsubckt=subx> <psubckt=subp> <ztime=ztime>
<start=start_time1 <stop=stop_time1 <start=start_time2
<stop=stop_time2 ...>>> <separate_file=0|1> <num=warn>
```

Chapter 17: Timing and Power Analysis

Power Checking

Parameters*Table 90 High Impedance Node Check*

Parameter	Description
title_name	Every high-impedance is reported in the high-impedance check file .chk following the title name of title_name.
node1 .. nodeN	Each name, specified by node1 ... nodeN defines signal node name. Each name can be the node name of a single node or a node name containing an asterisk (*) wildcard character which represents a group of node names. The node name with wildcard character must be specified as v(node_name) or 'node_name', because in the HSIM netlist parser and in SPICE syntax, all characters after (*) are treated as comments and are ignored.
fanout	The optional setting fanout=val2 limits the high impedance state checking to those driver nodes with fanouts. This setting avoids unnecessary checks on internal nodes in logic gates. fanout=0 checks both the internal node and the driver node. <ul style="list-style-type: none"> ▪ fanout=0: all nodes. ▪ fanout=1: Nodes that have direct connection to transistor gate. ▪ fanout=2: Nodes that have direct connection to transistor bulk. ▪ fanout=3: Nodes that have direct connection to transistor gate or bjt base. This is the default value.
psubckt	Only checks the ports of the subcircuit subp. This option should be used only when node1 is v(*). To exclude multiple subcircuits, use one psubckt for each subp. For example: psubckt=sub1 psubckt=sub2 ... The maximum total number of psubckt is 1024.

Table 90 High Impedance Node Check (Continued)

xsubckt	Exclude the specified subcircuit subx for the check. This option should be used only when node1 is v(*). To exclude multiple subcircuits, use one xsubckt for each subx. For example: xsubckt=sub1 xsubckt=sub2 ... The maximum total number of xsubckt is 1024.
ztime	Any specified node staying in the high-impedance state longer than ztime will be reported in the timing/power check file following the title name of title_name. The default value of ztime is 5 ns.
start, stop	This check is performed within the time window defined by start_time1, stop_time1 .. start_timeN, stop_timeN.
separate_file=0 1	Specifies whether a separate output file is needed: <ul style="list-style-type: none"> ▪ 0: [default] No separate output file ▪ 1: The following separate file is used to keep the output: hsim.zstate_<title_name>.chk
num	Limits the number of warnings generated by a particular analysis command defining it. Only the specified number of warnings is generated for a particular analysis command as defined by that command. The default value is 300. If specified number is set to less than or equal to 0, the warnings are unlimited.

Examples

```
.pcheck highz zstate a1 a2 v(x1.x2.*) ztime=50n start=100n
stop=1000n
```

This command checks the high-impedance state of the nodes a1, a2 and those nodes matching the pattern of x1.x2.*. The high-impedance state is checked between 100 ns and 1,000 ns. If any specified node remains in the high impedance state longer than 50 ns, it will be reported in the timing/power check file following the title name of highz.

```
.pcheck highz zstate v(*) xsubckt=wctrl xsubckt=rctrl
```

This command checks the high-impedance state of all nodes except those in subcircuits wctrl and rctrl.

Power Check Windows

Syntax

```
.pcheck window start_time1 <stop_time1 <start_time2  
      <stop_time2 ...>>>
```

Description

.pcheck window defines the time window(s) for each power check command that doesn't have its own time window specification. Each power check with its own time window specification will not be affected by this window setting.

Examples

```
.pcheck window 10n 20n 110n 120n 210n
```

This sets three time windows for those power checks without time window specification. The time windows are from 10 ns to 20 ns, from 110 ns to 120 ns, and from 210 ns to the end of simulation.

Activity Checking

This section describes the commands that perform activity checks:

- [Active Node Check](#)
 - [Active Files](#)
 - [Inactive Files](#)
-

Active Node Check

.acheck finds the active nodes in a design. A node is active if it has a voltage change during the simulation runtime.

Syntax

```
.acheck node_pattern <level=val2> <dv=val> \  
      <start=start_time>  
<stop=stop_time> <exclude=exc_pattern>
```

Parameters

Table 91 Activity Checking Parameters

Parameter	Description
node_pattern	Defines the signal node name(s) which can be the node name of a single node or a node name containing wildcard character '*' representing a group of node names. The node name with wildcard character must be specified as v(node_name) or node_name, because in the HSIM netlist parser and in SPICE syntax, all characters after '*' are treated as comments and are ignored.
dv=val	Defines the threshold of voltage difference. A node is active when the voltage change, compared to the initial value of the node, is larger than val. DEFAULT of val is 0.1 volt.
exclude=exc_pattern	Defines the signal node name(s) which are excluded from the list of nodes that need to be checked. Wildcard characters can be used and need to be quoted such as: 'a*'
level=val2	<p>This setting is effective only when the wildcard character is specified in the output variable. The level value val2 specifies the number of hierarchical depth levels when the wildcard node/element name matches. The level option acts on both active and inactive files.</p> <ul style="list-style-type: none"> ▪ When val2 is set to 1, the wildcard match applies to the same depth level where the <code>.acheck</code> statement is located. ▪ When val2 is set to 2, it applies to the same level and to one level below the current level where <code>.acheck</code> is located. ▪ When val2 is set to 0, no wildcard name is matched so that any output variable containing wildcard character is ignored. ▪ When val2 is set to -1, the wildcard match applies to all the depth levels below and including the current level of <code>.acheck</code> statement. ▪ The default value of val2 is -1.
start=start_time, stop=stop_time	Specifies the start_time and stop_time in the time window. The activity is checked at the time within the specified time. If no time window is specified, the check is performed from the time 0 ns to the end of simulation.

Chapter 17: Timing and Power Analysis

Activity Checking

After simulation, the active node list is saved in the .ach file, The inactive node list is saved in .ina and .ina_all files. The .ina file keeps all the inactive nodes not affected by their surrounding active nodes. The .ina_all file keeps all the inactive nodes in the entire circuit. .ina_all file can be huge. The formats of the activity files are in ASCII and line-by-line. [Example](#) shows the .ach file.

Note: In the case of a 2-input NAND gate with in1 and in2 inputs and an out output, if in1 is connected to GND and in2 is given an input pulse, the out node will be reported in the .ina file because the out node is an inactive node that is not affected by the in2 active node.

Active Files

Example

```
*HSIM Win32
*Copyright (C) 1998 - 2005. All rights reserved.
*
*
*Active driver node list
ina
```

Inactive Files

The format of the inactive files is:

Syntax

```
<index> <node_name> 0 <voltage>
```

Parameters

Table 92 Inactive File Parameters

Parameter	Description
index	A generated index name
node_name	The inactive node name
voltage	The constant voltage kept by the node

Example

```
*HSIM Win32
*Copyright (C) 1998 - 2005. All rights reserved.
*
*
*Inactive driver node list (complete)
V_inact_0 0 0 0
V_inact_1 xinvseta.inv 0 1.52086
V_inact_2 xinvsetb.inv 0 3
```

Selective Net Backannotation Flow

The traditional selective net backannotation flow has two phases, one for the prelayout simulation and one for postlayout simulation:

- In the first phase, HSIM simulates the prelayout netlist and automatically determines which nets are active and writes a file (*file.hsimba*) that contains the active nets. You use the .acheck command to define the active nets and the threshold value. For example:

```
.acheck '*' dv=0.25
```

Checks all nodes and reports only those which have a voltage change greater than 0.25v into a *<prefix>.hsimba* file

- To run the second phase, you need to include the *<prefix>.hsimba* file manually as input to the second phase, post-layout simulation. You also need to comment out the .acheck command you used in the prelayout simulation. HSIM back-annotates resistance and capacitance for all the nets in the *<prefix>.hsimba* file. All other nets should have no backannotation.

Chapter 17: Timing and Power Analysis

Activity Checking

HSIM also has a new flow that lets you run the prelayout and postlayout simulations in one single phase. To run this flow, set [HSIMSELBA on page 147](#)=1, which performs the following operations:

1. Disables the [HSIMSPF on page 155](#)/[HSIMSPF, HSIMSPEF on page 154](#)/[HSIMDPF on page 82](#)/[HSIMSBA](#) commands.
2. Runs a prelayout simulation with a default dv value of .acheck.

You can use the optional .acheck command if you want to filter the active nodes or to change the threshold value of the dv definition.

3. After the prelayout simulation finishes, HSIM writes out a <prefix>.hsimba file that contains all active nets in the following format:

```
.param HSIMSPFRCNET = vcon  
.param HSIMSPFRCNET = inb2  
.param HSIMSPFRCNET = inb3  
.param HSIMSPFRCNET = x1/x2/nn1
```

4. Next, HSIM launches a postlayout simulation with the following modifications:
 - Creates a new input file, <prefix>_selba.sp.
 - Includes <prefix>.hsimba in the <prefix>_selba.sp file.
 - Includes the original, prelayout input file in <prefix>_selba.sp.
 - Enables [HSIMSPF](#), [HSIMSPEF on page 154](#).

Example: Selective Net Backannotation in One Phase

This example uses the following input file to demonstrate a selective net simulation in one phase:

```
.param HSIMVDD=1.5v  
.param HSIMSPF=rc.spf  
.param HSIMDPF=rc.spf  
.param HSIMSELBA=1  
.inc models.inc  
vvdd vdd 0 1.5  
vvss vss 0 0  
X1 A B vdd vss test  
V1 A vss pwl (0 0 100n 1.5 200n 1.5 300n 0)  
.tran 1n 1000n  
.inc netlist.sp  
.acheck '*' dv=0.25  
.end
```

To run a selective net simulation with this netlist in one phase, do the following steps:

- To run the prelayout simulation, specify: `hsim input.sp -o test/output`.
`HSIMSELBA=1` starts the one phase selective net backannotation. HSIM disables the `HSIMSPF` and `HSIMDPF` commands and runs the prelayout simulation with the user-specified `.acheck` command.

HSIM creates `test/output.hsimba` which contains all the active net information.

HSIM generates the following `input_selba.sp` input file for the postlayout simulation:

```
.param HSIMSELBARUN =2
.inc ./test/output.hsimba
.inc ./input.sp
.end
```

- The `input_selba.sp` file includes the original `input.sp` file, `output.hsimba`, and `.param HSIMSELBARUN=2`.
- `HSIMSELBARUN=2` instructs HSIM to launch the following post-layout simulation automatically:

```
hsim input_selba.sp -o test/output_selba
```

The following command lets you rename the output file suffix:

```
.param HSIMSELBASFX="_selba"
```

This command controls the output suffix, which is appended to the user-defined `-o` specification of second phase output files when running one-phase selective net backannotation flow. The default naming of the output suffix in the second phase is `_selba`. For example, if you specify:

```
hsim input.sp -o output/1
```

And the HSIM settings contain:

```
.param HSIMSELBASFX=test
```

The phase 1 output file is `1.log` and the phase 2 output file is `1test.log`.

Note that if you set this parameter to "" (empty set), second-phase output files overwrite the first-phase output files containing the same name.

Feedback

Chapter 17: Timing and Power Analysis

Activity Checking

Monte Carlo Analysis

Provides information on how Monte Carlo Analysis (MCA) calculates the circuit response to the statistical variation of design and model parameters by applying randomly selected values from user-specified distributions.

- Overview of Monte Carlo Analysis
- Selecting the Monte Carlo Analysis Type
- Selecting the Distribution Function Parameter Type
- Specifying Starting Seed
- Selecting the HSIM Monte Carlo Analysis Mode

Overview of Monte Carlo Analysis

Monte Carlo Analysis (MCA) calculates the circuit response to changes component values by randomly varying all of the model parameters for which a tolerance is specified. This provides statistical data on the impact of a device parameter's variation on circuit operation. Model parameters are given tolerances. Multiple simulations are run using these tolerances for:

- DC analysis
- Transient analysis

One frequent use of MCA is to predict the yields of circuit production runs. MCA obtains statistical information by applying random variation to parameters in the following:

- Device models
- Circuit element instances

Chapter 18: Monte Carlo Analysis

Selecting the Monte Carlo Analysis Type

Parameters are specified with nominal values and tolerances along with the selected random number distribution function. The analysis involves:

- Several simulation runs
- Collecting all simulation results
- Producing a statistic summary report

The MCA commands are described in the following sections.

Selecting the Monte Carlo Analysis Type

For DC analysis:

```
.dc var start stop incr sweep monte=iter_num
```

For transient analysis:

```
.tran steptime stoptime sweep monte=iter_num
```

The *iter_num* represents the number of HSIM iterations (for example: simulation runs) to be performed.

Selecting the Distribution Function Parameter Type

Three distribution functions can be used in the analysis:

- Gaussian Distribution
 - Uniform Distribution
 - Limited Distribution
-

Gaussian Distribution

Either of the following syntax statements can be used.

```
.param param_name=gauss(nominal_val, rel_variation,  
+ sigma)  
.param param_name=agauss(nominal_val, abs_variation,  
+ sigma)
```

Uniform Distribution

Either of the following syntax statements can be used.

```
.param param_name=unif(nominal_val, rel_variation)
.param param_name=aunif(nominal_val, abs_variation)
```

Limit Distribution

```
.param param_name=limit(nominal_val, abs_variation)
param_name
```

The parameter whose value is calculated by applying the distribution function.

gauss

Gaussian distribution using relative variation.

agauss

Gaussian distribution using absolute variation.

unif

Uniform distribution using relative variation.

aunif

Uniform distribution using absolute variation.

limit

Limit distribution using absolute variation.

Note: abs_variation=rel_variation * nominal_value.

To take the effects of both element and model parameter variation into account, the distribution function is recalculated for each circuit element and each device model.

Specifying Starting Seed

```
.OPTIONS SEED=<seed_number>
```

Chapter 18: Monte Carlo Analysis

Selecting the HSIM Monte Carlo Analysis Mode

Use .OPTIONS SEED to specify the starting seed for the random number generator.

Selecting the HSIM Monte Carlo Analysis Mode

Use the following commands for Monte Carlo analysis:

- [HSIMMONTECARLOINST on page 109](#)
- [HSIMMONTECARLOSAVEOUT on page 109](#)

HSIM supports Monte Carlo analysis only in the HSPICE® netlist format, and only in transient analysis. The only supported flow is as described in the *HSPICE® User Guide: Simulation and Analysis, Chapter 20, Monte Carlo - Traditional Flow and Statistical Analysis*.

Monte Carlo analysis is compatible with structural backannotation (SBA).

The following HSPICE netlist commands and options are supported:

```
.tran ... sweep Monte=val [firstrun=firstrnum]
.tran ... sweep Monte=list(list_specification )
val ::= integer | parameter
firstrnum ::= integer | parameter
list_specification ::= (sample | range) {, sample} {, range}
sample ::= val
range ::= start_val:end_val
```

Traditional Monte Carlo defines a random variable with the use of a distribution function. HSIM supports the same distribution functions as HSPICE:

```
UNIF(nominal_val, rel_variation [, multiplier])
AUNIF(nominal_val, abs_variation [, multiplier])
GAUSS(nominal_val, rel_variation, num_sigmas [, multiplier])
AGAUSS(nominal_val, abs_variation, num_sigmas [, multiplier])
LIMIT(nominal_val, abs_variation)
```

The HSIM Monte Carlo outputs files are shown in [Table 93](#).

Table 93 Monte Carlo output files

File Extension	Description
.mip	Lists all instance parameter values affected by Monte Carlo random variations.

Table 93 Monte Carlo output files

File Extension	Description
.mmp	Lists all model parameter values affected by Monte Carlo random variations.
.mc .mt	Concatenates measure results from all Monte Carlo iterations.
.mc	Shows a statistical summary of .measure results for all Monte Carlo iterations, including a histogram.

Feedback

Chapter 18: Monte Carlo Analysis

Selecting the HSIM Monte Carlo Analysis Mode

Part 3: HSIM Advanced Modeling

Using Verilog-A with HSIM

Provides information on how the Verilog-A compiler adds behavioral modelling capability to HSIM. This implementation of Verilog-A is based on the analog subset of Accellera organization's Verilog-AMS Language where, the Verilog-A parser reads source code written in the Verilog-A language and provides HSIM with the necessary simulation information. With the Verilog-A feature, designs including Verilog-A modules can be included in the HSIM netlist.

As of the F-2011.09 release, a new Verilog-A front-end and evaluation engine called PVA has been integrated into HSIM. PVA replaces the legacy Verilog-A engine as the default Verilog-A solution in HSIM simulation. PVA is currently also used in XA and HSPICE® simulation.

- [Verilog-A Compiler](#)
- [Verilog-A Options](#)
- [Including Verilog-A Modules in HSIM Simulations](#)
- [Verilog-A Model Explanation](#)
- [HSIM .log file Information](#)
- [Case Sensitivity Issue In SPICE Netlist](#)
- [Verilog-A Module Name Clash with Subcircuit Names](#)
- [Verilog-A Language and HSIM Implementation-Specific Features](#)
- [Operators](#)
- [Statements](#)
- [Analog Events](#)
- [Analog Operators](#)
- [Interpolation Function](#)
- [Laplace Transform Filters](#)

Chapter 19: Using Verilog-A with HSIM

Verilog-A Compiler

- [Analysis](#)
- [I/O and Messages](#)
- [Printing Variables in Verilog-A Instances](#)
- [Multiple Inclusions of Module Files and Duplicate Declarations](#)
- [Macro Definitions](#)
- [Hierarchical Instantiation](#)
- [Simulation Speed Limitations](#)
- [Partitioning Verilog-A Modules](#)
- [Isomorphic Matching Verilog-A Modules](#)
- [Verilog-A Interactive Commands](#)
- [Encrypted Verilog A Modules](#)
- [Limitations and Unsupported Features](#)
- [Legacy Verilog-A Compiler Support](#)

Verilog-A Compiler

The Verilog-A compiler adds analog behavioral modeling capability to HSIM. The Verilog-A parser reads source code written in the Verilog-A language and provides HSIM with the necessary simulation information. With the Verilog-A feature, designs including Verilog-A modules can be included in the HSIM netlist.

The HSIM implementation of Verilog-A is based on the analog subset of *Verilog-AMS Language Reference Manual, v2.2*. Limitations and extensions are listed in [Verilog-A Language and HSIM Implementation-Specific Features on page 566](#).

Note: Additional information is contained in the following documentation: *Accellera, Verilog-AMS Language Reference Manual: Analog & Mixed-Signal Extensions to Verilog HDL, Version 2.2*.

Verilog-A Options

This section covers the following topics:

- [PVA Versus Legacy Verilog-A Compiler](#)
 - [Verilog-A Architecture](#)
 - [Getting Started with the Verilog-A Compiler](#)
-

PVA Versus Legacy Verilog-A Compiler

PVA is the default Verilog-A engine, which offers better performance, better compliance to LRM-2.2, and better compatibility to XA and HSPICE simulation than the legacy Verilog-A engine. However, you can still use the legacy Verilog-A compiler. For information about how to use the legacy Verilog-A compiler, see the [Legacy Verilog-A Compiler Support](#) section.

PVA always uses compiled Verilog-A modules before simulation in HSIM, rather than using interpreted Verilog-A code. In the legacy Verilog-A engine, you have to set compiler option to activate compilation mode.

Verilog-A Architecture

The HSIM Verilog-A compiler interface is shown in [Figure 25 on page 555](#).

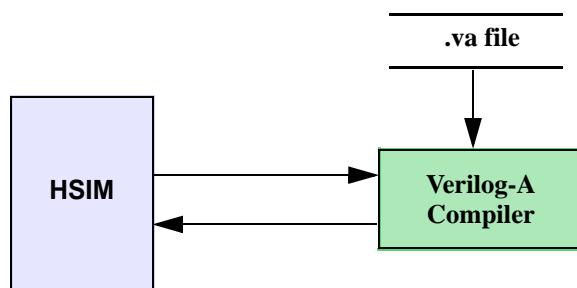


Figure 25 Verilog-A Compiler Architecture

Getting Started with the Verilog-A Compiler

The Verilog-A compiler is invoked within HSIM during the HSIM simulation run.

The following code sample is provided to familiarize users with Verilog-A.

1. Create an example Verilog-A source file, res.va.

```
resistor
`include "discipline.h"
module resistor(a,b);
inout a,b;
electrical a,b;
branch(a,b) res;
parameter real R=1;
analog begin
I(res) <+ V(res)/R;
end
endmodule
```

2. Create the example HSIM netlist file, res.sp.

```
* res.sp
.param HSIMVERILOGA="res.va"
vsource 1 0 DC 5
r1 1 2 10K
x1 2 0 resistor R=10K
c1 2 0 1uF
.tran 0.1n 100n
.end
```

3. Start HSIM with the command.

```
HSIM.exe res.sp
```

This process will successfully run a Verilog-A simulation using HSIM.

Including Verilog-A Modules in HSIM Simulations

The HSIM parser invokes the Verilog-A parser when it executes on a netlist containing specific Verilog-A include statements. After parsing the module, the Verilog-A parser registers the module with the HSIM simulator. All the information, warning and error messages go into the simulation log file.

Verilog-A files are written in text file(s) that are usually named as <filename>.va. Although, it is a good practice to name your Verilog-A files with

.va extension, it is not required by HSIM to do so. A sample Verilog-A file is shown below.

File "res.va":

```
~include "discipline.h"
~include "constants.h"
// res
//
// - resistor
//
// vp,vn:      terminals (V,A)
//
// INSTANCE parameters
//      r=resistance (Ohms)
//
// MODEL parameters
//      {none}
//

module res(vp, vn);
inout vp, vn;
electrical vp, vn;
parameter real r=1K;
analog
    I(vp, vn) <+ V(vp, vn)/r;

endmodule
```

Important: "discipline.h" and "constants.h" are header files that define the disciplines/natures and constants respectively. These files are searched according to the path given in the Verilog-A file. If they are not found, they will be read from \$HSIM_HOME/include. The files included in this directory include:

Chapter 19: Using Verilog-A with HSIM

Including Verilog-A Modules in HSIM Simulations

```
"discipline.h"
"disciplines.h"
"constant.h"
"constants.h"
"epva.h"
"discipline.vams"
"disciplines.vams"
"driver_access.vams"
"compact.vams"
"const.vams"
"constant.vams"
"constants.vams"
"standard.vams"
"constants.va"
"constant.va"
"const.va"
"disciplines.va"
"discipline.va"
"logic.va"
"standard.va"
"std.va"
```

Including Verilog-A Modules in a SPICE Netlist

Verilog-A modules are included in a SPICE netlist by specifying the source file name in [HSIMVERILOGA on page 193](#) or an HSPICE .hdl statement.

Syntax

```
.param HSIMVERILOGA=<"file_name">
.hdl "file_name"
```

Arguments

file_name <file_name>

file_name may include a relative or absolute path name and is the name of the Verilog-A source file to be compiled. The Verilog-A model is then instantiated in the netlist by “X<text>” like a SPICE subcircuit.

Example

A sample SPICE file with Verilog-A statement is below.

```
* sample RC circuit
.param HSIMVERILOGA=".res1.va"

vpulse n0 gnd pulse 0 3000.0m 0 30n 30n 300n 600n
x1 n0 n1 res1
c1 n1 gnd 100p
rr1 n0 nn1 1K
cc1 nn1 gnd 100p

.meas n0 rms v(n0)
.meas n1 rms v(n1)
.meas n2 rms v(nn1)

.tran 1n 2000n
.print v(*) i(cc1) i(rr1) level=1
.param HSIMOUTPUT=wdx
.end
```

Including Verilog-A Modules in a SPECTRE Netlist

To incorporate Verilog-A modules (contained in a Verilog-A source file) in a SPECTRE® netlist, specify the source file name in the `ahdl_include` statement in the netlist.

Syntax

```
ahdl_include <file_name>
```

Arguments

`file_name`

`file_name` may include a relative or absolute pathname as the name of the Verilog-A source file to be compiled. Since, HSIM can run SPECTRE netlists directly. The Verilog-A model is then instantiated in the netlist by any SPECTRE element.

Example

A sample SPECTRE netlist file is given below.

Chapter 19: Using Verilog-A with HSIM

Verilog-A Model Explanation

```
simulator lang=spectre  
  
ahdl_include "res.va"  
  
v1 n0 gnd vsource type=dc dc=1  
a95 n0 n1 RES r=1k  
r0 n1 gnd resistor r=1k  
  
.end
```

Including Verilog-A Modules in an ELDO Netlist

To incorporate Verilog-A modules (contained in a Verilog-A source file) in an ELDO netlist, specify the source file name in a .verilog statement in the netlist.

Syntax

```
.verilog <file_name>
```

Arguments

file_name

file_name may include a relative or absolute pathname as the name of the Verilog-A source file to be compiled. The instantiation is done similar to SPICE/SPECTRE netlists.

Verilog-A Model Explanation

The following examples are included to provide a better understanding of Verilog-A models.

The following code resistor.va shows how a resistor can be implemented and coded in Verilog-A. The numbers shown on the margins of the Verilog-A file are only for illustration purposes, please do not include them into the actual Verilog-A source file. Refer to examples for the disciplines.h file.

Example 50

```

0 // resistor.va
1 `include "discipline.h"
2 module resistor(a,b);
3 inout a,b;
4 electrical a,b;
5 branch(a,b) res;
6 parameter real R=1;
7 analog begin
8 I(res) <+ V(res)/R;
9 end
10 endmodule

```

The following is a line-by-line explanation of the code:

Line no.	Code	Description
0		This line is a comment. All comments start with //.
1	include discipline.h	Directs the compiler to insert the contents of discipline.h before compiling.
2	module resistor(a,b)	Defines the interface between the resistor module and HSIM. It tells HSIM that there is a module called resistor that has two ports, a and b.
3	inout a,b	Specifies the direction of the ports. The Verilog-A language standards define three port directions, input, output and inout. Each port must have a direction associated with it.
4	electrical a,b	Defines the disciplines for each of the ports. Users can define and name a discipline. Typical disciplines may be electrical, thermal, kinematic, etc. Refer to disciplines.h for defining disciplines.
5	branch(a,b) res	Defines a branch res. A branch is simply defined as a path between two nodes.

Chapter 19: Using Verilog-A with HSIM

Verilog-A Model Explanation

Line no.	Code	Description
6	parameter real R=1	Defines the R parameter that has a default value of 1 (one). Parameters provide another interface between the HSIM netlist and Verilog-A module. If you decide to specify another value for R, this may be done at the HSIM netlist level (without the need to change and recompile the Verilog-A module in the source file).
7	analog begin	Specifies the beginning of an analog block. An analog block is where the behavioral description of the component being modeled is specified. There can only be one analog block for each module.
8	I(res) <+ V(res)/R	Specifies the behavioral description of the resistor. The <+ in this line is called a contribution statement. The line is read as: The current flowing through branch res is contributed to by the voltage across branch res divided by the parameter P.
9	end	Specifies the end of the analog block.
10	endmodule	Specifies the end of the module.

An example HSIM netlist, res.sp, which contains a reference to the Verilog-A module, is shown in [Example 51](#).

Example 51

```
0 ***** res.sp
1 .param HSIMVERILOGA "resistor.va"
2 vsource 1 0 dc 5
3 r1 1 2 10K
4 xr2 2 0 resistor R=10K
5 c1 2 0 1uF
6 .tran 1n 1u
7 .end
```

The following is a line-by-line explanation of the code

Line no.	Code	Description
0		HSIM comment line.
1	verilog resistor.va	A reference to the Verilog-A source file, resistor.va. It tells HSIM that there is a Verilog-A file to be used within the netlist.
2	vsource 1 0 dc 5	Creates a 5V DC voltage source.
3	r1 1 2 10K	Creates a 10k resistor.
4	xr2 2 0 resistor R=10K	Defines an instance, xr2, of the module, resistor. Note that x defines a Verilog-A component (much the same as an r defines a SPICE resistor). Once the instance has been named (in this case xr2), the nodes that it connects to are specified. These nodes will correspond to the terminals specified in the Verilog-A module contained within the res.va file. In this example, node 2 corresponds to terminal a and node 0 corresponds to terminal b in the Verilog-A module resistor. Once the nodes have been specified, assignment of the parameters is done. In this case, the parameter R is assigned a value of 10k, overriding the default value of 1.
5	c1 2 0 1uF	Creates a 1uF capacitor.
6	.tran	Specifies transient analysis.
7	.end	Marks the end of the SPICE netlist.

Executing the HSIM command on this netlist causes HSIM to invoke the Verilog-A compiler. The Verilog-A compiler would do the following:

- Compile the res.va input file.
- Register the module resistor with HSIM, given that res.va is syntactically and semantically correct.
- The Verilog-A compiler will send all the Info, Warning, and Error messages to the simulation log file.

Chapter 19: Using Verilog-A with HSIM**HSIM .log file Information**

- If res.va contains syntactic or semantic errors, all error messages would be directed to the simulation log file.
- The xr2 instance can then be used in an HSIM simulation, just like any other instances, within the supported analysis types.

HSIM .log file Information

During simulation, HSIM prints several messages related to the Verilog-A models. When HSIM invokes the Verilog-A parser, parser issues and information messages are seen on line 13. For each model, read into this module file, the parser issues a message when registering these models on line 14.

When the circuit statistics are generated at a later time, the number of times Verilog-A models are instantiated are shown. The Verilog-A models are listed as P VAMOD elements on line 66. A model-by-model list of instantiations are also given in the .log file on line 71. The following example illustrates the .log file output.

```
21 Reading (pass #1) 'res.sp'22 Reading Verilog-A model file by
pVA: resistor.va
23
24
25 Begin of pVA compiling ...
26
27 Parsing 'resistor.va'
...
31 End of pVA compiling
...
48 Register Verilog-A Module:resistor
...
61 Circuit Statistics
62
63 RES Elements : 1
64 GCAP Elements : 1
65 VSRC_DC Elements : 1
66 PVAMOD Elements : 1
67
68 Total # of Elements : 3
69 Total # of Nodes : 4
70
71 resistor : 1
72
```

Case Sensitivity Issue In SPICE Netlist

Since Verilog-A Language is case sensitive and SPICE is case insensitive, there might be problems with the module names. In order to overcome this issue, HSIM looks for the lowercase model name in default mode. If the -case 1 option is used during HSIM invocation, then HSIM honors the case of the Verilog-A module name.

Verilog-A Module Name Clash with Subcircuit Names

Whenever there is a clash between the names of subcircuits in an HSIM netlist, C-model names and Verilog-A model names, HSIM resolves the name in the following order.

1. Sub-circuit Name
2. C-model Name
3. Verilog-A Model Name

For each X instance in the HSIM netlist, HSIM first attempts to match the name with the subcircuit name. If the subcircuit name is not matched, then HSIM tries to find a C-model with the same name. Verilog-A models with that name are searched last.

To increase the Verilog-A priority vs. other model types in HSIM, use [HSIMVERILOGA](#) on page 193.

When [HSIMPREFERVERILOGA](#) on page 128 is specified, the Verilog-A Model Name option supersedes all other options and the order changes as follows:

1. Verilog-A Model Name
2. Subcircuit Name
3. C-model Name

To select a particular Verilog-A module instead of a sub-circuit having same name for simulation, use [HSIMUSEVA](#) on page 184 . Use this command instead of [HSIMPREFERVERILOGA](#) when you want only the specified Verilog-A module.

Verilog-A Language and HSIM Implementation-Specific Features

This section provides an overview of the Verilog-A language and specific HSIM implementations. Complete Verilog-A details are provided in the OVI, Verilog-AMS Language Reference Manual: Analog & Mixed-Signal Extensions to Verilog HDL, Version 2.0. Open Verilog International and other related textbooks.

HSIM Implementation-Specific Features

HSIM implementation-specific features documented in this manual include the following:

- Data types
- Parameters

Integer or real type parameter declarations are supported by HSIM Verilog-A. Since they are converted to real double precision numbers during simulation, the default type is real if not otherwise specified. Only scalar parameters are supported.

Parameters can also be declared with an optional permissible range. The parameter value is verified with the specified range during instantiation stage. Simulation aborts if verification fails.

Parameters represent constants. Their values are resolved at instantiation stage and cannot be modified during simulation. Following are examples:

Example 52

```
parameter integer rate1=12 from [0:inf] exclude (30:50) exclude  
100;  
parameter rate2=12; //real is the default type.  
parameter real rate3=12;  
parameter real poles[0:2]={1.0, 2.0, 3.0}; // Not allowed.
```

In these cases, rate1 is declared as an integer with default value of 12 and, rate2 and rate3 are declared as real parameters.

Integer and Real Number Variables

Variables can be declared as integer or real. Arrays are supported. Values of variables are preset to zero during instantiation and the values can be updated during simulation. Examples are as following:

```
parameter integer ran=5
integer a1[1:ran];
real a2[1:-5];
```

Note: HSIMs .store/.restore feature does not support Verilog-A variables.

Natures and Disciplines

Natures define a set of attributes for physical and simulation quantities, such as units, access function names or tolerances. Sample excerpts from standard definitions are shown below:

```
nature Current
    units="A";
    access=I;
    abstol=1e-12
endnature
nature Voltage
    units="V";
    access=V;
    abstol=1e-6;
endnature
```

The three attributes, units, access and abstol, are mandatory for declaration of natures. HSIM aborts when access is not defined but defaults units and abstol to NA and 1e-6 if they are not defined.

Disciplines bind natures to potential and flow as defined in Kirchoff's laws.

Kirchoff's Potential Law

Potential binding defines the access function of nature.

Kirchoff's Flow Law

Flow binding defines that of the nature which obeys Kirchoff's Flow Law.

An example defining analog circuit signal electrical is as follows:

Chapter 19: Using Verilog-A with HSIM

Verilog-A Language and HSIM Implementation-Specific Features

```
discipline electrical
    potential      Voltage;
    flow          Current;
enddiscipline
```

Multiple declarations do not stop HSIM parsing. However, the new definition is discarded with a Warning message.

Nodes

Nodes in Verilog-A modules are associated with disciplines. A node declared without a range is a scalar node. A node declared with a range is a vector node or analog bus. HSIM only allows constant node widths overriding the width during instantiation is not permitted.

```
electrical out, in; // scalar nodes
electrical [5:10] node1; // vector nodes
```

Branches

Branches can be explicitly declared by the branch keyword and given terminal pairs. HSIM only supports declaration of scalar branches. A branch defines a path between two terminals. Potential difference, i.e. voltage, when the terminals are declared as electrical, can be accessed using the access function V(<branch name>) defined by voltage nature. Flow, i.e. current for electrical terminals, is accessed by I(<branch name>) defined by current nature.

Examples of declarations are as shown below:

```
electrical a, b
electrical [-3:0] a1;
electrical [17:20] b1;
branch (a, b) br1;
branch (a1, b1) x; // not supported, bus to bus
branch(a1[-2], b1[17]) y;
```

To access potential difference and flow of branch br1, we use V(br1) and I(br1).

Implicit branch declaration is also supported. When an access function, for example V(b), is referred to in the behavioral description and b is a node, an implicit branch is constructed between node b and ground. Similarly V(a,b) where a and b are nodes, constructs a branch from a to b.

There can be only one implicit branch between the same two terminals. That is, multiple references of V(a) are all referring to the same branch. This also

applies to $V(a,b)$ and $V(b,a)$, for which the access functions return the voltage of $V(a)-V(b)$ and $V(b)-V(a)$ respectively.

Operators

The supported operators for expressions are tabulated in this section:

- Unary operators.
- Binary operators.
- Ternary operators.

[Table 94](#) lists the Verilog-A unary operators.

Unary Operators

Table 94

Operator	Description	Allowable Operands	Example
+	positive	Integer, real	$I=+10; // I=10$
-	negative	Integer, real	$R=-14.1; // R=-14.1$

[Table 95](#) lists the Verilog-A binary operators.

Binary Operators

Table 95

Operator	Description	Allowable Operands	Example
+	add	Integer, real	$I=10+2.1; // I=12.1$
-	subtract	Integer, real	$I=10-3.7; // I=6.3$
*	multiply	Integer, real	$I=2*4.2; // I=8.4$
/	divide	Integer, real	$I=4/2.0; // I=2.0$
<	less than	Integer, real	$I=5<7; // I=1$
>	greater than	Integer, real	$I=5>7; // I=0$

Chapter 19: Using Verilog-A with HSIM
 Statements
Table 95

Operator	Description	Allowable Operands	Example
<code><=</code>	less than or equal to	Integer, real	<code>I=5<=7; // I=1</code>
<code>>=</code>	greater than or equal to	integer, real	<code>I=5>=7; // I=0</code>
<code>==</code>	equal to	Integer, real	<code>I=5==7; // I=0</code>
<code>!=</code>	not equal to	Integer, real	<code>I=5!=7; // I=0</code>
<code>&&</code>	logical and	Integer, real	<code>I=(1==2) &&(3==3); // I=0</code>
<code> </code>	logical or	Integer, real	<code>I=(1==2) (3==3); // I=1</code>
or	event or	Event expression	<code>@(initial_step or cross(V(in), +1))</code>

[Table 96](#) lists the Verilog-A ternary operators.

Ternary Operators

Table 96

Operator	Allowable Operands	Example
<code>Cond? expr1:expr2</code>	Integer, real	<code>I=(3>2)? 1: 0;</code>

Cond, expr1 and expr2 are all expressions. The operator evaluates expr1 if the condition is true, and otherwise expr2.

Statements

Supported statement types include the following:

- Procedural Assignment
- Branch Contribution
- Block

- If-Else
- Case
- Repeat Loop
- While Loop
- For Loop
- Generate

Procedural Assignment Statements

Procedural assignment statements evaluates the expression on the RIGHT hand side and assigns the value to a variable on the LEFT hand side. The variable has to be modifiable during simulation, i.e. it is declared as integer, real or an integer element or real array. Procedural assignment takes effect immediately when the statement is executed. The RIGHT-hand operand can be any arbitrary scalar expression.

```
real a[0:2], sum;
.
.
.
r[0]=5.0;
r[1]=3.0;
r[2]=2.0;
sum=r[0]+r[1]+r[2];
```

Branch Contribution Statements

The LEFT-hand operand of the branch contribution specifies a source branch signal. It is in the form of access function of the branch, to which the RIGHT hand side expression contributes. During simulation, HSIM evaluates the value of the RIGHT-hand side expression and accumulates it to the source branch. After all the statements in the module having been executed, thus the algebraic expressions for the model have been evaluated, the system is solved by the HSIM kernel for the time point. Therefore, the contribution statement is also referred to as a simultaneous assignment.

It is important to note that this operation is cumulative. Voltage contribution branches are summed up as if they are in series, while current contribution branches are summed up in parallel. In the following two contribution statements:

Chapter 19: Using Verilog-A with HSIM
Statements

```
V(br) <+ 3;  
V(br) <+ 4;
```

the module is equivalent to the following single statement:

```
V(br) <+7;
```

Block Statements

A block statement consists of a list of statements. The statements are executed sequentially as in the following syntax example, which shows a begin and end statements:

```
begin  
    integer k;  
    k=k+1;  
end
```

An if-else statement describes conditional execution behavior as shown in the following example:

```
if(value<0) $strobe("value < 0");  
else if((value>0) && (value<=1)) $strobe("value in (0,1]");  
else if((value >1) && (value<=2)) $strobe("value in (1,2]");  
else $strobe("value > 2");
```

Case Statements

A case statement defines a multiway decision behavior structure. The testing condition, which is the argument expression to the case, is evaluated and compared against the condition labels in the case items. When found, the execution branches to the corresponding statements. When no matching condition is found and default statements are defined after default label, the statements are executed. Unlike C language, the condition label can also be an expression, which is evaluated during condition matching. The following syntax example illustrates a case statement:

```
case(1)  
    (value<0): $strobe("value < 0");  
    ((value>0) && (value<=1)): $strobe("value in (0,1]");  
    ((value>1) && (value<=2)): $strobe("value in 1,2]");  
    default $strobe("value > 2");  
endcase
```

Repeat Loop Statements

A repeat loop statement is used to run the enclosed statements repeatedly for a fixed number of times. A repeat statement repeats the associated statements a fixed number of times given by the argument to the repeat keyword as shown in the following syntax example:

```
repeat (5) begin
    i=i+1;
    total=total + i;
end
```

While Loop Statements

The while statement executes the associated statements repetitively when the condition expression to the while keyword is evaluated as true as shown in the following syntax example:

```
while(a<10) begin
    i=i+1;
    sum=sum+i
    a=a+1;
end
```

For Loop Statements

The for loop statement executes the associated statements repetitively when the condition expression evaluates true. The statement extends the previous while statement with additional index manipulating facilities as shown in the following syntax example:

```
for(i=1; i<10; i=i+1) begin
    i=i+1;
    sum=sum+i
end
```

Generate Statements

The generate statement unrolls its statement during compilation as shown in the following syntax:

Chapter 19: Using Verilog-A with HSIM

Analog Events

```
generate i (3,1) begin
    V(out[i]) <+ 3;
end
```

The generate statement is compiled as follows:

```
V(out[3]) <+ 3
V(out[2]) <+ 3
V(out[1]) <+ 3
```

Analog Events

`initial_step` returns true when the simulation is at the first step in simulation. It has the following syntax:

```
@(initial_step) begin
    v1=0;
    v2=1;
end
```

Limitation: HSIM does not support `initial_step` with any argument.

Cross Statements

`cross` returns true when the signal in its argument crosses the given threshold along a specified direction. `cross` also adjusts the simulation timestep in order to precisely catch the processing point.

```
@(cross(V(in)-2.5, +1)) begin
    V(out) <+ transition(x, 1n, 3n);
end
```

In this example, when "V(in)-2.5" becomes 0 "and" is going in the positive direction, the action below the `cross` statement will be evaluated and executed.

HSIM places the simulation time step close to the crossing point during the VA crossing event. By default, this should be determined by the simulator engine dynamically, however, you can apply the third optional parameter `t_tol` and fourth optional parameter `s_tol` to the `cross` function as guidance for HSIM to place the time step properly.

```
@(cross(V(in)-2.5, +1, 10p, 0.01v) begin
    V(out) <+ transition(x, 1n, 3n);
end
```

This example indicates a similar execution procedure, except that the simulation time step (the event) will most likely fall into the box formed by `t_tol(10ps)` and `s_tol(0.01v)`. The `t_tol` and `s_tol` parameters have a default value of 0. See [Figure 26 on page 575](#) for an illustration of the example.

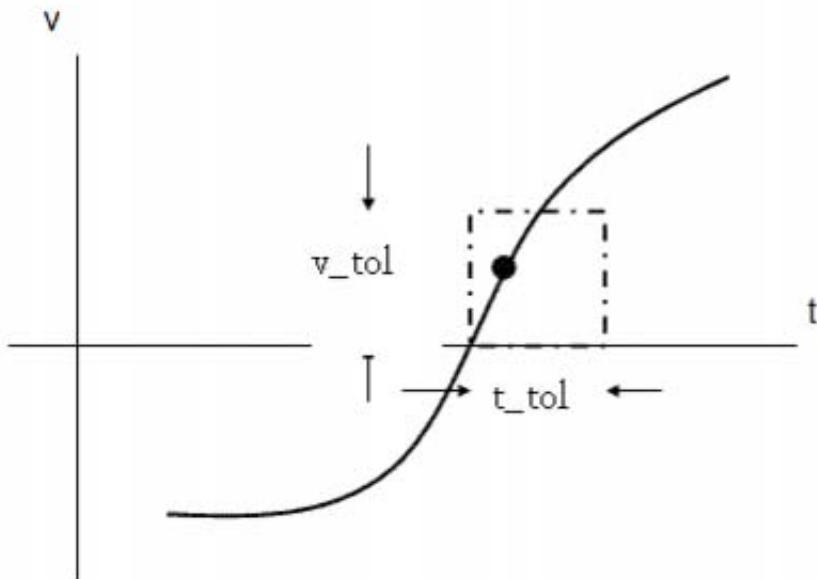


Figure 26 Simulation Time Step Placement

HSIM has its own time step selection scheme during the simulation, but with PVA integration, it honors the setting of `t_tol` and `s_tol` in the cross function statement to accurately locate the time step. For the legacy Verilog-A solution to set the tolerance, see the [Legacy Verilog-A Compiler Support](#) section.

Timer Statements

`timer` returns true when the simulation time reaches the time point or periodic time points specified in its argument as shown in the following example:

```
@(timer(0, period)) x=-x;
V(out) <+ transition(x, 1n, 10n);
```

Chapter 19: Using Verilog-A with HSIM

Analog Operators

In the syntax example above, `x` changes sign at $t=0$ and $0+n*\text{period}$, where n is a positive integer. The `period` cannot be negative and it must be greater than zero. An error message is displayed when an illegal period is assigned to the `timer` function.

Analog Operators

This section lists the analog operators:

- [ddt Operator](#)
- [idt Operator](#)
- [delay Operator](#)
- [Transition Operator](#)
- [slew Operator](#)
- [bound_step Operator](#)
- [discontinuity Operator](#)
- [\\$STOP Operator](#)
- [\\$finish Operator](#)

ddt Operator

The `ddt` operator computes the time derivative of its argument. In current implementation, tolerance specification is ignored and the value of [HSIMALLOWEDDV on page 59](#) parameter is used. Following is the syntax for `ddt`:

```
V(out) <+ ddt(V(in));
```

idt Operator

The `idt` operator computes the time integral of its argument. In the current implementation, the tolerance specification is ignored and the value of [HSIMALLOWEDDV on page 59](#) is used. Following is the syntax for `idt`:

```
V(out) <+ idt(V(in),1);
```

delay Operator

The `delay` operator delays a continuous signal by a given delayed time. The delay time and maximum delay have to be positive. The following error message is given when either of them evaluates negative:

```
DelayDelay/maximum delay time has to be positive.
```

The delay function syntax follows:

```
v(out) <+ delay(v(in), 5n);
```

Transition Operator

The transition operator smooths out and delays a piece-wise signal. The delay time has to be nonnegative. The rise/fall time has to be positive. The current implementation uses a default rise/fall time of 1ps if they are not user- or model-specified. The following error messages are displayed for delay, rise, and fall times that are evaluated and found to be negative:

```
DelayTransition delay cannot be smaller than zero.  
Rise Transition rise time cannot be smaller than zero  
Fall Transition fall time cannot be smaller than zero
```

Following is the syntax for transition:

```
v(out) <+ transition(v(in), 1n, 5n, 10n);
```

slew Operator

The `slew` contribution filter bounds the rate of change of the signal in its argument. The maximum positive and negative slew rate has to be positive and negative constant number during evaluation.

The following error messages are given respectively when either maximum positive or negative slew rate is evaluated negative.

- Maximum positive slew rate must be greater than zero
- Maximum negative slew rate must be smaller than zero

The slew function syntax follows:

```
v(out) <+ slew(v(in), slewrate);
```

bound_step Operator

The `bound_step` statement limits the maximum time-step of transient analysis. The step size has to be positive. The following error message is issued when a negative step is given:

```
Time step of bound_step has to be positive
```

The bound function syntax follows:

```
bound_step(1n) ;
```

discontinuity Operator

HSIM will ignore large voltage deviations for Verilog-A modules when discontinuity is used as shown in the following syntax:

```
discontinuity(0) ;
```

\$STOP Operator

The `$stop` operator stops simulation from a Verilog-A module. HSIM enters into the interactive mode when `$stop` is activated. Following is the syntax for `$stop`:

```
$stop
```

\$finish Operator

The `$finish` operator ends simulation from within Verilog-A modules. Following is the syntax for `$finish`:

```
$finish
```

Interpolation Function

The `$table_model` function models the behavior of a system by interpolating between data points that are samples of that system's behavior. The `$table_model` function supports 1D, 2D and 3D tables. The syntax for the `$table_model` function is as follows.

```
$stable_model(table_inputs, table_data_source,  
    table_control_string)
```

An example of a 1D table follows:

```
$stable_model(V(node1), "./voltage_samples", "L")
```

Laplace Transform Filters

Use the laplace transform function to filter the input waveform as shown in the following example:

```
V(out) <+ laplace_zp(V(in), [0,1], [1, -0.4, 0.2]);
```

There are four types of laplace transform functions:

- laplace_zp(expr,z,p): zero-pole
- laplace_zd(expr,z,d): zero-denominator
- laplace_np(expr,n,p): numerator-pole
- laplace_nd(expr,n,d): numerator-denominator

The following error message is issued when non-constant expressions are specified for zeros and poles:

```
Illegal zeros/poles assigned to laplace functions
```

Analysis

Currently, only the following three analysis functions are supported:

- analysis("static"): Applies to DC initialization
- analysis("ic"): Applies to DC initialization
- analysis("tran"): Applies to Transient Analysis

I/O and Messages

The section lists the I/O and messaging commands:

- \$strobe, \$monitor, \$display, \$fstrobe
 - \$error
 - \$warn
-

\$strobe, \$monitor, \$display, \$fstrobe

The \$strobe, \$monitor, \$display, and \$fstrobe statements are used in debugging module variables during simulation. Their usage is similar to (f)printf in C language. The format string supports C language specifiers as well as the %m Verilog-A specific format converter specifier. \$strobe, \$monitor, and \$display are treated the same. \$fstrobe dumps messages to a designated file channel.

If the format string is not given, it is assumed to be “”. If there are fewer items in the output than are specified, HSIM produces an error and stops. It issues the following error message:

Too few items to print according to the output format

If there are more items than are listed in the format specifiers, a warning is issued as shown in the following example:

Too many items to print, omitting the extras

\$error

\$error prints out a message in its argument, triggers the HSIM error routine, and aborts simulation with the following error message:

User requested \$warn() task

\$warn

\$warn prints out a message in its argument, triggers the HSIM Warning routine, and logs the Warning message in both the simulation log file and the console. The Warning message is prefixed with "#WARNING#".

Printing Variables in Verilog-A Instances

To print a variable in a Verilog-A instance, use the following syntax:

```
.print <instance_name>:<variable_name>
```

For example:

```
.print xmt:torque
```

To print variables in the legacy Verilog-A solution, see the [Legacy Verilog-A Compiler Support](#) section.

Multiple Inclusions of Module Files and Duplicate Declarations

A Verilog-A file is parsed only once in a given netlist. HSIM ignores multiple references to the same module.

If different modules are declared using the same name, the new definition is discarded by the parser. HSIM issues a Warning in the simulation log. The same scheme is applied to both disciplines and natures.

Macro Definitions

Macros are defined using ‘define’ compiler directives. The macros are expanded in the netlist according to their definition. The only restriction is that after the macro reference is replaced with a textual expansion, the description must form a valid statement or statements.

Hierarchical Instantiation

HSIM supports hierarchical instantiation of Verilog-A models and SPICE primitives inside Verilog-A models. During simulation, these instances are flattened in the netlist up to the level of the parent instance.

[Example 53](#) shows a simple netlist that instantiates two inverter models.

Chapter 19: Using Verilog-A with HSIM

Hierarchical Instantiation

Example 53

```
.param HSIMVERILOGA="inv.va"

vin ref1 gnd PWL(0 0 10n 0 11n 4 20n 4 21n 0)
x1 ref1 out inverter
x2 out out2 inverter

.tran 0.1n 30n
.printf v(*)
.ends
```

The inv.va file is shown in the following syntax sample. In the sample, the inverter module instantiates two resistors (SPICE Primitives) and one fgopamp module (Verilog-A).

```
`include "../discipline.h"
`include "../constants.h"

module inverter(in,out);
inout in,out;
electrical in, out;
electrical gnd,minus;
parameter real gain=1e5;
parameter real R=1e5;
resistor # (.r(2k)) r1(in,minus);
resistor # (.r(1K)) r2(minus,out);
fgopamp # (.gain(1e6)) op(gnd,minus,out);
endmodule

module fgopamp(,minus,out);
inout minus,out;
electrical minus,out;
parameter real gain=1e5;

analog
      V(out) <+ gain*V(,minus);
Endmodule
```

Note: If the Verilog-A model has no behavior, but only hierarchy inside, that module will be removed during simulation. The instances of this module will be instantiated regularly. For the syntax sample above, there will be only one Verilog-A model in the circuit inventory and that will be fgopamp. The inverter model will be removed from the circuit after the devices inside are instantiated.

Simulation Speed Limitations

HSIM simulation speed will be affected adversely if the design employs a large number of Verilog-A models. Designs that include a large number of transient models written in Verilog-A will run slower than simulations with built-in models.

Partitioning Verilog-A Modules

Use the following commands for partitioning: [HSIMVAPARTITION](#) on page 189. By default, partitioning is enabled.

Isomorphic Matching Verilog-A Modules

Use the following commands for partitioning: Use the following command for isomorphic matching: [HSIMVACCBMATCH](#) on page 188. By default, isomorphic matching is not enabled.

Verilog-A Interactive Commands

This section lists the Verilog-A interactive commands:

- [ev, iev](#)
 - [nc, inc](#)
-

ev, iev

The ev and iev commands are enhanced for Verilog-A elements. Ports, port connections, and branch equation information can be accessed using these commands. An example of the ev command follows:

```
HSIM > ev xr2
xr2 (2) - TERM#0:2 (2), TERM1:0 (0)
TYPE PVAMOD
```

nc, inc

The nc and inc commands are new etypes used with Verilog-A elements. To get a list of Verilog-A elements connected to a node, enter the following command:

```
nc vdd -etype y
```

Encrypted Verilog A Modules

HSIM is able to parse Verilog A modules that were encrypted with the metaencrypt utility in HSPICE.

Limitations and Unsupported Features

Unsupported features and limitations include the following:

- Only transient simulation is supported. AC analysis is not supported.
- Bus width declarations must be constant expressions and cannot be overridden from instantiating circuits.
- Numerical scheme tolerances set by reltol/abstol or specified in natures are ignored. The error controlling parameter [HSIMALLOWEDDV](#) on page 59 of the HSIM kernel overrides the setting.
- Nature statements:
 - ldt_nature
 - ddt_nature
- Discrete disciplines
- Indirect branch assignment statement
- Noise functions such as the following are ignored:
 - White noise

- Flicker noise
- PVA does not support static CircuitCheck

If you specify a static CircuitCheck command in the netlist, HSIM uses the legacy Verilog A engine and issues a warning.

Legacy Verilog-A Compiler Support

The legacy Verilog-A compiler can be selected by setting the following environment variable:

```
setenv HSIM_LEGACY_VA 1
```

To go back to the default PVA:

```
unsetenv HSIM_LEGACY_VA
```

Compiler Option

In addition to interpreted evaluation, HSIM provides a compile option for Verilog-A. The Verilog-A parser provides two types of compilation for a Verilog-A module:

Batch: Batch compilation is enabled by setting the environment variable HSIM_COMPILE_VERILOGA to 1, y, or Y. This option creates a compiled library for all parsed Verilog-A modules.

Module-based: Module-based compilation is specified by including the hsim_compile_module in a module definition command line. This option creates a compiled library for a specific module that can be evaluated at a later time. For example:

```
module aaa(a,b,...)
// Put the specifier in the module definition,
// such as the following line,
// 'hsim_compile_module
inout a, b;
...
endmodule
```

Supported Features

Features supported in HSIM Verilog-A include:

- AC analysis
 - ac_stim functions
 - z-transform
-

Supported HSIM-VA Commands

The following table shows the full list of existing HSIM-VA commands and their support in HSIM-PVA.

HSIM VA Commands	Legacy VA Support	PVA Support
HSIMVERILOGA on page 193	Yes	Yes
HSIMUSEVA on page 184	Yes	Yes
HSIMPREFERVERILOGA on page 128	Yes	Yes
HSIMVAPRINTVAR on page 189	Yes	No
HSIMVACCBMATCH on page 188	Yes, default 1	Yes, default 0
HSIMVAPARTITION on page 189	Yes	Yes
HSIMUSEVATABLE on page 185	Yes	No
HSIMVATBLERANGE on page 190	Yes	No
HSIMVATABLESIZE on page 190	Yes	No

HSIM VA Commands	Legacy VA Support	PVA Support
HSIMVACROSSTOL on page 188	Yes	No
HSIMVACROSSVTOL on page 188	Yes	No

Cross Statements

To enforce the simulation time step falling inside the t_tol and v_tol box in a cross function, use [HSIMVACROSSTOL on page 188](#) and [HSIMVACROSSVTOL on page 188](#) to control the time step selection.

Using Tables for Verilog-A models

For simple Verilog-A models, HSIM has an option to use table models instead of evaluation during simulation. Using tables improves simulation speed. However, only simple models are read from the table: larger models are still evaluated. The following parameters are used for activating table models: [HSIMUSEVATABLE on page 185](#), [HSIMVATBLERANGE on page 190](#) and [HSIMVATABLESIZE on page 190](#).

Printing Variables in Verilog-A Instances

To print variables in Verilog-A instances, use [HSIMVAPRINTVAR on page 189](#).

Encrypted Verilog A Modules

HSIM is able to parse Verilog A modules that were encrypted using the in HSPICE. When encrypting a Verilog A module, the entire encrypted module must reside in one file before you use the metaencrypt utility. The legacy Verilog A engine does not support partially encrypted Verilog A modules.

Feedback

Chapter 19: Using Verilog-A with HSIM

Legacy Verilog-A Compiler Support

Ferroelectric Capacitor (FeCap) Model

Describes the ferroelectric capacitor (FeCap) model, FeCap elements, and FeCap model parameter extraction.

- [Overview of the Ferroelectric Capacitor \(FeCap\) Model](#)
- [FeCap Elements](#)
- [FeCap Model](#)
- [FeCap for Model Parameter Extraction](#)

Overview of the Ferroelectric Capacitor (FeCap) Model

HSIM supports the ferroelectric capacitor (FeCap) model. Ferroelectric random access memory (FRAM) is nonvolatile memory featuring fast read/write operation as low power battery-backed SRAM. FRAM eliminates the need for a battery.

Ferroelectric material is distinguished from other materials by the P-V hysteresis loop. When voltage is externally applied to a ferroelectric capacitor, sweeping from a large negative to a large positive voltage and back to the original negative voltage, the corresponding polarization (which is directly related to the charge quantity for circuit design purpose) changes from a negative value to a positive value along the lower characteristic curve and then back to the original negative value along an upper characteristic curve.

The hysteresis loop of a ferroelectric film is not fixed. It depends on several factors including:

Chapter 20: Ferroelectric Capacitor (FeCap) Model**FeCap Elements**

- Film history
- Peak and frequency of the applied voltage
- Operating temperature

The hysteresis loop corresponding to the maximum peak voltage in one application is called the major hysteresis loop, while the other loops are called the sub-loops (or minor loops).

In the FRAM operation, the changes in the operating point switch among the loops during the transient period according to the switching conditions determined by the following factors:

- Ferroelectric capacitor history
- Current state
- Voltage applied

FeCap Elements

The FeCap element can be handled in HSIM as a capacitor with model parameter level=6.

Syntax

```
Caa n1 n2 model_name area=val1 p0=val2 v0=val3
```

Parameters

FeCap parameters are described in [Table 97 on page 590](#).

Table 97 FeCap Element Parameters

Parameter	Description
Caa	Capacitor element name, which must begin with the character C.
n1	Positive node name.
n2	Negative node name.
model_name	Capacitor model name.
area	Area of the ferroelectric capacitor (units in square meter). The default value is 1.0e-12.

Table 97 FeCap Element Parameters

Parameter	Description
p0	Polarization (units in uC/cm ²). The default value is 0.
v0	Initial voltage (units in volt). The default value is 0.

Example

```
c1 1 2 frmc area=3p p0=0 v0=0
```

FeCap Model

The hysteresis loops of the FeCap model from Ramtron are symmetric with respect to the origin point in a P-V plot.

Syntax

```
.model model_name c level=6 <parameter=val1>
```

Parameters

FeCap model parameters are shown in [Table 98](#).

Table 98 FeCap Model Parameters

Parameter	Default Value	Description
vmax	5.0	Maximum voltage used in measured data
pmax	30.27	Maximum polarization at vmax in uC/cm ²
kpmx	-3.47e-2	Temperature coefficient of pmax
vsat	2.0	Saturation voltage
vcr	0.5	Minimum voltage for domain switching on unpolarized cap only
as1	2.5000e-1	Curve fitting parameter for a saturated loop
bs1	0.12390e1	Curve fitting parameter for a saturated loop
cs1	0.13483e1	Curve fitting parameter for a saturated loop

Chapter 20: Ferroelectric Capacitor (FeCap) Model
FeCap Model

Table 98 FeCap Model Parameters

Parameter	Default Value	Description
as2	5.6726e-2	Curve fitting parameter for a saturated loop
bs2	2.4269e-1	Curve fitting parameter for a saturated loop
cs2	-3.8183e-2	Curve fitting parameter for a saturated loop
ds0	5.0536e-1	Curve fitting parameter for a saturated loop
au1	5.0152e-1	Curve fitting parameter for a unsaturated loop
bu1	1.4908	Curve fitting parameter for a unsaturated loop
cu1	1.0950	Curve fitting parameter for a unsaturated loop
au2	7.4715e-2	Curve fitting parameter for a unsaturated loop
bu2	1.3517e-7	Curve fitting parameter for a unsaturated loop
cu2	-8.2528	Curve fitting parameter for a unsaturated loop
du0	2.1247e-3	Curve fitting parameter for a unsaturated loop
kas1	-1.0107e-3	Temperature coefficient for a saturated loop
kbs1	1.4579e-3	Temperature coefficient for a saturated loop
kcs1	-3.0895e-3	Temperature coefficient for a saturated loop
kas2	4.8937e-4	Temperature coefficient for a saturated loop
kbs2	6.2892e-4	Temperature coefficient for a saturated loop
kcs2	7.2631e-3	Temperature coefficient for a saturated loop
kds0	9.7153e-5	Temperature coefficient for a saturated loop
kau1	-9.5819e-4	Temperature coefficient for an unsaturated loop
kbu1	2.6618e-3	Temperature coefficient for an unsaturated loop
kcu1	-6.1257e-3	Temperature coefficient for an unsaturated loop

Table 98 FeCap Model Parameters

Parameter	Default Value	Description
kau2	3.7970e-4	Temperature coefficient for an unsaturated loop
kbu2	2.7597e-4	Temperature coefficient for an unsaturated loop
kcu2	1.0525	Temperature coefficient for an unsaturated loop
kdu0	-3.4559e-2	Temperature coefficient for an unsaturated loop

Example

```
.model frmcc level=6
```

FeCap for Model Parameter Extraction

There are 18 parameters in the ferroelectric capacitor model, FeCap obtained from curve fitting of a saturated loop or an unsaturated loop as shown in [Table on page 593](#).

FeCap Model Parameters from Curve-fitting Saturated & Unsaturated Loops

Table 99

Saturated Loop	Unsaturated Loop
as1	au1
bs1	bu1
cs1	cu1
as2	au2
bs2	bu2
cs2	cu2
ds0	du0

Chapter 20: Ferroelectric Capacitor (FeCap) Model

FeCap for Model Parameter Extraction

Each of these parameters has temperature coefficients to estimate its variation with temperature. Vmax is the maximum voltage in the parameter extraction measurement. Usually, Vmax is the maximum V_{DD} in the application of the film.

Pmax is the polarization when Vmax is applied on the ferroelectric capacitor.

Vsat is the criteria voltage to distinguish saturated and unsaturated loops.

When a peak voltage applied on the capacitor is higher than Vsat, the parameters for saturated loops will be used; otherwise, the parameters for unsaturated loops will be used. Vsat should be larger than the coercive voltage of the film. The coercive voltage is defined as the voltage corresponding to the zero polarization point when the applied voltage sweeps from $-V_{max}$ to V_{max} .

Vcr is the criteria voltage for domain switching. When a voltage applied on an unpolarized ferroelectric capacitor is lower than Vcr, we assume no domain switching and the capacitor is a linear capacitor. When the voltage is higher than Vcr, domain switching starts and the P-V curve exhibits hysteresis loops. Usually, Vcr is about 0.5V for 5V films. Vcr should be less than the coercive voltage of the film.

Measuring Ferroelectric Hysteresis Loops

All model parameters are extracted from hysteresis loops that are measured with a Sawyer-tower circuit, as shown in [Figure 27 on page 595](#).

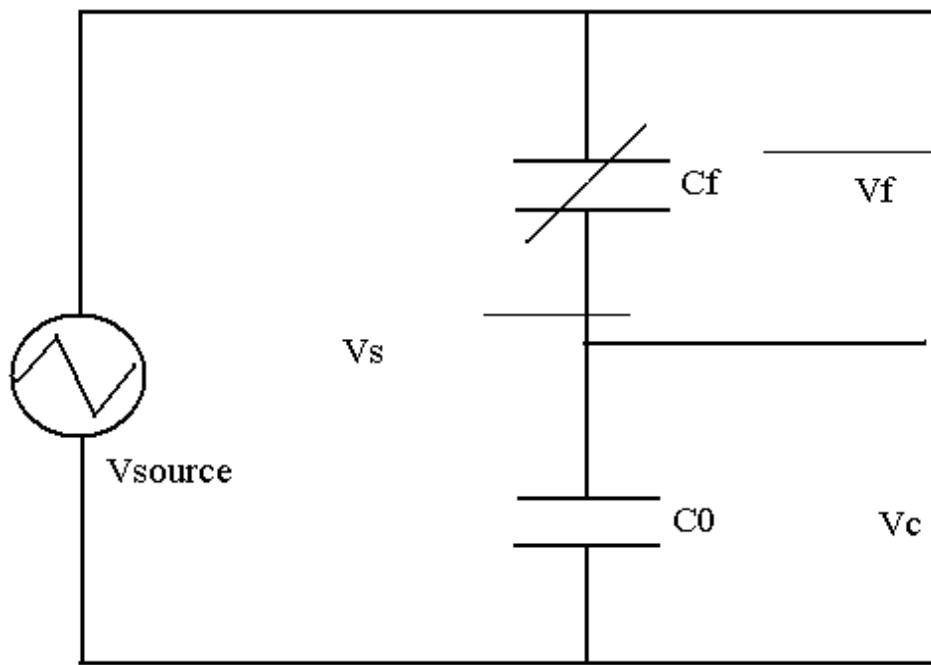


Figure 27 Sawyer-Tower Circuit for Hysteresis Loop Measurement

In [Figure 27 on page 595](#):

- **Vsource** is a function generator that generates either a sinusoidal wave or a triangular wave. Typically, a lower frequency such as 2KHz is used.
- **C0** is a high precision linear capacitor.
- **Cf** is a ferroelectric capacitor.

C0 is usually chosen to be much larger than **Cf** so that most of **Vs** will be dropped on **Cf** and the hysteresis loops can be conveniently monitored with an oscilloscope.

In a ferroelectric film with an area of 2500um^2 , the maximum value of **Cf** is about 500pF . We choose **C0** to be about 5000pF . In the measurement, **Vs** and **Vc** are sampled and stored. **Vf** is obtained from **Vs** - **Vc**.

The polarization in the ferroelectric capacitor is calculated using the following equation:

$$P = C_0 \times V_c / A_f$$

where A_f is the area of the ferroelectric capacitor.

Chapter 20: Ferroelectric Capacitor (FeCap) Model

FeCap for Model Parameter Extraction

If a plot of P vs. V_f , or V_c vs. V_f is conducted, a loop is observed as V_f goes from $-V_{max}$ to V_{max} and then returns to $-V_{max}$. A typical ferroelectric hysteresis loop is shown in [Figure 28 on page 596](#).

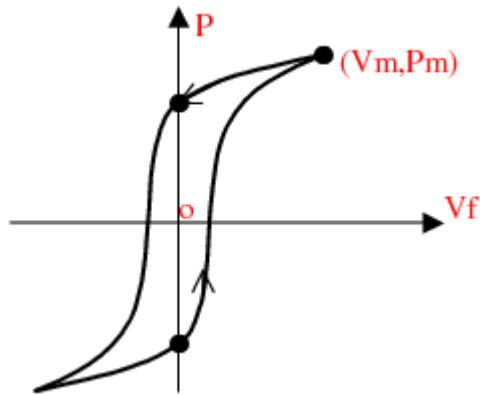


Figure 28 Ferroelectric Hysteresis Loop Example

Two hysteresis loops are used for parameter extraction. One is a saturated loop which corresponds to $V_{max}=V_{DDmax}$, where V_{DDmax} is the maximum source voltage in the application. The other one is an unsaturated loop, which corresponds to $V_{max} @ V_{DD}/2$. Actually, V_{max} should be slightly larger than the coercive voltage of the ferroelectric film in this case. An unsaturated loop is used to obtain parameters, a_{u1} , b_{u1} , c_{u1} , a_{u2} , b_{u2} , c_{u2} , and d_{u0} .

For a 5V film, $V_{max}=5.5V$. A saturated loop is used to obtain parameters, a_{s1} , b_{s1} , c_{s1} , a_{s2} , b_{s2} , c_{s2} and d_{s0} .

FeCap Model Parameter Extraction

The saturated parameters, a_{s1} , b_{s1} , c_{s1} , a_{s2} , b_{s2} , c_{s2} and d_{s0} , are extracted from a saturated hysteresis loop.

The steps are as follows:

1. Obtain a saturated hysteresis loop from measurement data.
2. Remove the upper-branch of the hysteresis loop. Since the hysteresis loop is assumed to be symmetric about the origin, only the lower-branch is needed, as shown in [Figure 29 on page 597](#).

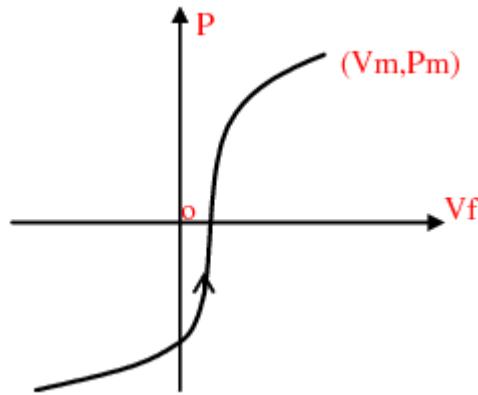


Figure 29 Lower Branch of a Hysteresis Loop

3. Normalize the lower branch of the hysteresis loop, as shown in [Figure 30 on page 597](#). For each data point (V, P) , it is normalized by:

$$Y = (P - P_{min}) / (P_{max} - P_{min})$$

where P_{max} and P_{min} are the maximum and the minimum values of P between $-V_{max}$ and V_{max} .

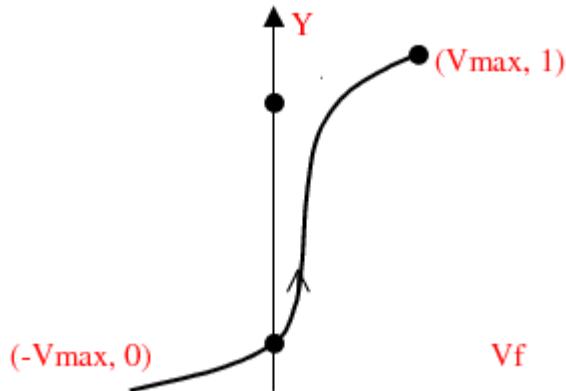


Figure 30 Normalized Lower Branch of a Hysteresis Loop

4. Curve-fit the normalized lower-branch to the following equation to obtain the parameters, $a_1, b_1, c_1, a_2, b_2, c_2$ and d_0 , which are the values of $as1, bs1, cs1, as2, bs2, cs2$ and $ds0$, respectively.

$$Y = d_0 + (a_1/b_1) \times \arctan(b_1(x - c_1)) + (a_2/b_2) \times \arctan(b_2(x - c_2))$$

Chapter 20: Ferroelectric Capacitor (FeCap) Model

FeCap for Model Parameter Extraction

The initial values for curve fitting are: $c1=Vsat/2$, and $c2=2*Vsat$.

The steps for extracting the unsaturated parameters $au1$, $bu1$, $cu1$, $au2$, $bu2$, $cu2$, and $du0$ are the same, except when an unsaturated hysteresis loop is used.

FeCap Model Temperature Coefficients

Saturated and unsaturated hysteresis loops are measured at different temperatures. The parameters, $Pmax$, $as1$, $bs1$, $cs1$, $as2$, $bs2$, $cs2$, $ds0$, $au1$, $bu1$, $cu1$, $au2$, $bu2$, $cu2$, and $du0$, are extracted for each temperature point. Then, temperature coefficients for each of these parameters are obtained by linear fitting.

The temperature coefficient $kas1$ of $as1$ is obtained by curve-fitting the following equation:

$$as1@T = as1@25C + kas1 \times (T - 25)$$

The temperature coefficient for $Pmax$ is obtained from the saturated $Pmax$ vs. Temperature measurements.

User Model Interface (UMI)

Provides information on HSIM proprietary MOSFET models. It describes how to implement user-defined models in a dynamic library file using the user model interface (UMI).

- [UMI Models](#)
 - [User Files](#)
-

UMI Models

HSIM allows the use of proprietary MOSFET models. While built-in device models are included with HSIM, a user-defined model must be described in a dynamic library file.

The user model interface (UMI) models are specified with the model parameter level. The levels used in the examples are as follows:

- Level 102 MOSFET Level 2 Model for Meyer capacitances illustration purpose
- Level 108 BSIM3v3 model for charge-based capacitances illustration purpose

Note: The level of the user-defined MOSFET model must be between 100 and 200.

In the example, model b3 will be compiled. The files below are required.

- UMI.h
- UMI.c
- b3defs.h

Chapter 21: User Model Interface (UMI)

User Files

- b3assigngeometry.c
- b3load.c
- b3main.c
- b3readmodel.c
- b3set.c
- b3temp.c

The corresponding files are described in the following sections.

Building a Dynamic Library

This section describes how to build the appropriate dynamic library for one or multiple models.

The file buildfmod, located in \$HSIM_HOME/bin, can be used to compile the user-defined model.

The one-line command to compile the model m115 follows:

```
%buildfmod user.so UMI.c m115assigngeometry.c m115load.c  
m115main.c m115readmodel.c m115set.c m115temp.c
```

This command compiles the C files into the dynamic library file user.so.

If two models b3 and m2 are compiled together, then the one-line command is as follows:

```
%buildfmod user.so UMI.c b3assigngeometry.c b3load.c b3main.c  
b3readmodel.c b3set.c b3temp.c m2assigngeometry.c m2load.c  
m2main.c m2readmodel.c m2set.c m2temp.c
```

User Files

This section describes the UMI user files:

- [Header File UMI.h](#)
- [Interface File UMI.c](#)
- [Header File b3defs.h](#)
- [Primary File for Model b3main.c](#)
- [Model Parameter Processing File b3readmodel.c](#)

- Model Default Value Setting File b3setmodel.c
- Geometry Assignment File b3assigngeometry.c
- Temperature Updating File b3temp.c
- Model Evaluation and Load File b3load.c

Header File UMI.h

Caution: Do not alter the file UMI.h.

The header file UMI.h defines the communication protocol between HSIM and the dynamic library user.so. The initialize() function is first called by HSIM. The TEMPS structure includes two elements: tnom for the nominal temperature and temp for the simulation temperature. The UMI_VAR struct contains elements to pass information between HSIM and the dynamic library user.so. The biasing voltages, vds, vgs, vbs, and device mode are sent from HSIM to the dynamic library.

The calculated results are returned from the dynamic library user.so to HSIM. The calculated results can include threshold voltage von, saturation voltage vdsat, drain current ids, output conductance gds, transconductance gm, gate-source overlap capacitance cgso, gate-drain overlap capacitance cgdo, gate-bulk overlap capacitance cgbo, substrate-source junction diode current ibs, substrate-drain junction diode current ibd, substrate-source junction diode capacitance capbs, and substrate-drain junction diode capacitance capbd.

If charge-based capacitance model equations are used, the returned capop value shall be 13 and the following can also be returned: gate charge qg, drain charge qd, source charge qs, and capacitance values cggb, cgdb, cgsb, cbgb, cbdb, cbsb, cdgb, cddb, and cdsb.

If the Meyer capacitance model equations are used, the returned capop value shall be 0, and positive gate-source capacitance capgs, gate-drain capacitance capgd, and gate-bulk capacitance capgb are returned.

Note: The cgso and cgdo capacitances are multiplied by the channel width. The cgbo capacitance is multiplied by the channel length.

The USER_MOSDEF struct contains information to assist HSIM to allocate adequate amount of memory space. It also contains function names for HSIM to call. The pModel address helps link the model struct specified in the dynamic library user.so back to HSIM. The value modelsize notifies HSIM about the

Chapter 21: User Model Interface (UMI)

User Files

required memory size for the particular model struct. The value instsize notifies HSIM the required memory size for the transistor instance struct.

In addition to initialize() function, HSIM calls other functions such as:

- initial_model()
- initial_geometry()
- read_model()
- assign_geometry()
- set_model()
- temp_geometry()
- model_load()

Two functions are optional:

- start()
- conclude()

The UMI.h file shall be kept intact without any modification because a copy exists inside HSIM to facilitate data communication between HSIM and the dynamic library user.so. Content of the UMI.h file is listed below.

Note: Contact a Synopsys Application Engineer for updates to the content of the UMI.h file.

Example 54 UMI.h File Example

```
UMI.h
/****************************************************************************
 * Copyright 1998 - 2004/
 * Synopsys Corporation      */
/* Module       : UMI.h      */
/* Last Update : Jan.2004*/
/* Description : USER Model Interface struct*/
/*                  */
/* DO NOT CHANGE THIS HEADER FILE*/
#ifndef UMI_H
#define UMI_H
#if defined(WIN32)
__declspec(dllexport) char * initialize();
#else
char * Initialize();
#endif

/* USER device type */
#ifndef NMOS
#define NMOS 1
#define PMOS -1
#endif

typedef struct TEMPS {
    double tnom; /* nominal temperature */
    double temp; /* simulation temperature */
} TEMPS;

/* USER MODEL interface variables */
typedef struct UMI_VAR {
    /* device input formation */
    int mode; /* device mode */
    double vds; /* vds bias */
    double vgs; /* vgs bias */
    double vbs; /* vbs bias */

    double von; /* threshold voltage (i.e., turn-on
                 voltage) */
    double vdsat; /* saturation voltage */
    double ids; /* drain dc current */
    double gds; /* output conductance (dIds/dVds) */
    double gm; /* transconductance (dIds/dVgs) */

    double cgso; /* gate-source overlap capacitance
                   (already multiplied by width) */
    double cgdo; /* gate-drain overlap capacitance
                   (already multiplied by width) */
}
```

Chapter 21: User Model Interface (UMI)

User Files

```
                                (already multiplied by width) */
double cgbo; /* gate-bulk overlap capacitance
                (already multiplied by length) */

int      capop; /* capacitor selector */
               /* capop can have following values
                  13: charge-based model; 0: Meyer
                  model */

/* intrinsic terminal charges and trans-capacitances,
   used for capop=13 */
/* NOTE: these are intrinsic capacitance ONLY,
   not including overlap value */
double qg; /* gate charge */
double qd; /* drain charge */
double qs; /* source charge */
double cggb;
double cgdb;
double cgsb;
double cbgb;
double cbdb;
double cbsb;
double cdgb;
double cddb;
double cdsb;
/* next: Meyer capacitances: intrinsic capacitance + overlap
   capacitance */
double capgs; /* Meyer model gate-source capacitance
                 (dQg/dVgs + cgso) */
double capgd; /* Meyer model gate-drain capacitance
                 (dQg/dVds + cgdo) */
double capgb; /* Meyer model gate-bulk capacitance
                 (dQg/dVbs + cgbo) */ /* substrate-
                                         junction information */
double ibs; /* substrate-source junction diode
               current */
double ibd; /* substrate-drain junction diode
               current */
double gbs; /* substrate source
               junction-conductance */
double gbd; /* substrate drain junction
               conductance */
double capbs; /* substrate-source junction diode
                 capacitance */
double capbd; /* substrate-drain junction diode
                 capacitance*/
/* additional*/
double ibd; /* drain-to-bulk static current through
               CHANNEL REGION not counting substrate
```

```
junction. Please add isub
(substrate current)to this term */
double isb; /* source-to-bulk static current through
CHANNEL REGION, not counting
substrate junction */
double igd; /* gate-to-drain static current */
double igs; /* gate-to-source static current */
double igb; /* gate-to-bulk static current */
} UMI_VAR;

/* USER interface functions */
typedef struct USER_MOSDEF {
#ifndef __STDC__
char ModelName[80];
char *pModel;
int modelsize;
int instsize;
void (*initial_model)(char*,int);
void (*initial_geometry)(char*);
void (*read_model)(char*,char*,double);
void (*assign_geometry)(char*,char*,double);
void (*set_model)(TEMPS*,char*);
void (*temp_geometry)(TEMPS*,char*,char*);
void (*model_load)(UMI_VAR *,char*,char*);
void (*start)();/* Optional */
void (*conclude)();/* Optional */
#else
char ModelName[80];
char *pModel;
int modelsize;
int instsize;
void (*initial_model)();
void (*initial_geometry)();
void (*read_model)();
void (*assign_geometry)();
void (*set_model)();
void (*temp_geometry)();
void (*model_load)();
void (*start)();/* Optional */
void (*conclude)();/* Optional */
#endif
} USER_MOSDEF;

#ifndef NULL
#define NULL 0
#endif
#endif /* end of UMI.h file */
/*********************************************
```

Interface File UMI.c

The interface file UMI.c contains one interface function initialize(). It is the first function inside the dynamic library user.so that is called by HSIM. HSIM sends two pieces of information to this function: type to be 1 for n-MOSFET and -1 for p-MOSFET, and level between 100 and 200. UMI.c returns the address of the device model if it finds a corresponding one with the same model level number. It can support multiple user-defined models. In addition, the initialize() function also calls the initial_model() function to grab the memory space for the matched model and also to store the device type value. An example of the UMI.c file is listed below.

Note: Boldface text identifies code that must not to be changed.

Example 55 UMI.c File Example

```
*****  
/* Copyright 1998 - 2004*/  
/* Synopsys Corporation*/  
/* Module      : UMI.c*/  
/* Last Update : Jan.2004*/  
/* Description : General Interface function of user-defined  
model*/  
/*  
** Sign Convention for MOSFET Current:  
** -----  
** positive current direction:  
** channel current : from drain to source terminal  
** junction-diode current : from bulk to source/drain  
**  
** At the bias conditions for characterization  
** NMOS : vds >=0,then ids >=0  
**          vbd, vbs < 0,then ibs <=0 & ibd <=0  
  
** PMOS : vds <=0,then ids <=0  
**          vbd, vbs > 0,then ibs >=0 & ibd >=0  
**  
** -----  
** derivatives are defined as:  
**          gm   is derivative of ids w.r.t. vgs  
**          gds is derivative of ids w.r.t. vds  
**  
** Meyer model capacitances (for capop=0) and p-n junction  
** capacitances always have positive values for both NMOS  
** and PMOS:  
** -----*/  
  
#include <stdio.h>  
#include <string.h>  
#include <math.h>  
#include "UMI.h"  
  
extern USER_MOSDEF *p_m2;  
extern USER_MOSDEF *p_b3;  
  
char *  
#ifdef __STDC__  
initialize(  
int type,  
int level,  
char *name)  
#else  
initialize(type,level,name)
```

Chapter 21: User Model Interface (UMI)

User Files

```
int      type;
int      level;
char *name;
#endif
{
int length;
char *pDevice=NULL;

/* select model level */

switch(level) {
/* you can add multiple model levels here */
case 102: /* "MOS Level-2 Model" */
pDevice=(char *)p_m2;
break;
case 108: /* "BSIM3 Model" */
pDevice=(char *)p_b3;
break;
default:
(void)fprintf(stderr, "\n[Error]: level %d model not
supported\n",      level);
return NULL;
}

length=strlen(name);

if(length < 79)
strcpy(((USER_MOSDEF*)pDevice)->ModelName, name);
else {
strncpy(((USER_MOSDEF*)pDevice)->ModelName, name, 79);
((USER_MOSDEF*)pDevice)->ModelName[79]=
(char)NULL;
}
((USER_MOSDEF*)pDevice)->initial_model(((USER_MOSDEF*)
pDevice)->pModel, type);

return pDevice;
}           /* end of UMI.c file */
*****
```

Header File b3defs.h

The header file b3defs.h contains the struct definitions for the particular user-defined model and the corresponding instance. If a different model name is to be used, then "b3" shall be replaced by the new name. A small example is listed below.

Example 56 b3defs.h Header File Example

```
*****  
#ifndef B3defs_h  
#define B3defs_h  
  
/* declaration for MOSFETs */  
/* information needed for each instance */  
  
typedef struct B3instance {  
/* add the actual content at this location */  
  
} B3instance;  
  
typedef struct B3model {  
/* add the actual content at this location */  
} B3model;  
  
#endif /* end of b3defs.h file */  
*****
```

Primary File for Model **b3main.c**

This is the primary source file for a particular user-defined model. It contains the actual implementation of the `b3_initialmodel()` function (which corresponds to `initial_model()` function described in `UMI.h` header file), the `b3_initialgeometry()` function (which corresponds to `initial_geometry()` function), the optional `b3_start()` function (which corresponds to `start()` function) and the optional `b3_conclude()` function (which corresponds to `conclude()` function). Content of the `b3main.c` file is listed below. Boldface is used to identify words that are not to be changed. Comments are printed at small font size.

Chapter 21: User Model Interface (UMI)

User Files

Example 57 b3main.c File Example

```
/*****************************************/
/*                                         */
/*      Copyright 1998 - 2004          */
/*      Synopsys Corporation          */
/*                                         */
/* Module       : b3main.c           */
/* Last Update : Jan.2004           */
/* Description : Unique Interface function for BSIM3 model */

#include <stdio.h>
#include <string.h>
#include <memory.h>
#include "UMI.h"
#include "m2defs.h"

#ifndef __STDC__
void b3_initialmodel(char*,int,int);
void b3_initialgeometry (char*);
void b3_read_model(char*, char*, double);
void b3_assign_geometry(char*, char*, double);
void b3_set_model(TEMPS*, char*);
void b3_temp_geometry(TEMPS*, char*, char*);
void b3_model_load(UMI_VAR*, char*, char*);
void b3_start();
void b3_conclude();
extern void b3_readmodel(B3model*, char*, double);
extern void b3_assigngeometry(B3instance*, char*, double);
extern void b3_setmodel(TEMPS*,B3model*);
extern void b3_tempgeometry(TEMPS*,B3model*,B3instance*);
extern void b3_load(UMI_VAR*,B3model*,B3instance*);
#else
void b3_initialmodel();
void b3_initialgeometry();
void b3_read_model();
void b3_assign_geometry();
void b3_set_model();
void b3_temp_geometry();
void b3_model_load();
void b3_start();
void b3_conclude();
extern void b3_readmodel();
extern void b3_assigngeometry();
extern void b3_setmodel();
extern void b3_tempgeometry();
extern void b3_load();
#endif
```

```
/* local */
static B3model      _B3Model;

static USER_MOSDEF bsim3def={
    (char*)&_B3Model,
    sizeof(B3model),
    sizeof(B3instance),
    b3_initialmodel,
    b3_initialgeometry,
    b3_read_model,
    b3_assign_geometry,
    b3_set_model,
    b3_temp_geometry,
    b3_model_load,
    b3_start,          /* Optional, replace with NULL if not used */
    b3_conclude,       /* Optional, replace with NULL if not used */
};

USER_MOSDEF *p_b3=&bsim3def;
/*** PARTICULAR MODEL INTERFACE SUBROUTINE ***

void
#ifdef __STDC__
b3_initialmodel(
    char *model,
    int   type)
#else
b3_initialmodel(model, type)
char *model;
int   type;
#endif
{
/* initialization */
(void)memset(model, 0, sizeof(B3model));
if(type== -1) {
    ((B3model*)model)->B3type=-1;
    ((B3model*)model)->B3typeGiven=1;
}
return;
}

void
#ifdef __STDC__
b3_initialgeometry(
    char *here)
#else
b3_initialgeometry(here)
char *here;
```

Chapter 21: User Model Interface (UMI)

User Files

```
#endif
{
(void)memset(here, 0, sizeof(B3instance));
return;
}

void
#ifdef __STDC__
b3_read_model(
char *model,
char *name,
double value)
#else
b3_read_model(model,name,value)
char *model;
char *name;
double value;
#endif
{
b3_readmodel((B3model*)model,name,value);
return;
}

void
#ifdef __STDC__
b3_assign_geometry(
char *here,
char *name,
double value)
#else
b3_assign_geometry(here,name,value)
char *here;
char *name;
double value;
#endif
{
b3_assigngeometry((B3instance*)here,name,value);
return;
}

void
#ifdef __STDC__
b3_set_model(
TEMPS *ckt_temp,
char *model)
#else
b3_set_model(ckt_temp,model)
TEMPS *ckt_temp;

```

```
char    *model;
#endif
{
    b3_setmodel(ckt_temp, (B3model*)model);
    return;
}

void
#ifdef __STDC__
b3_temp_geometry(
TEMPS   *ckt_temp,
char    *model,
char    *here)
#else
b3_temp_geometry(ckt_temp,model,here)
TEMPS   *ckt_temp;
char    *model;
char    *here;
#endif
{
/* temperature-updated parameters */

b3_tempgeometry(ckt_temp, (B3model*)model, (B3instance*)here);
return;
}
void
#ifdef __STDC__
b3_model_load(
UMI_VAR *custom,
char    *model,
char    *here)
#else
b3_model_load(custom,model,here)
UMI_VAR *custom;
char    *model;
char    *here;
#endif
{
b3_load(custom, (B3model*)model, (B3instance*)here);
return;
}

void
#ifdef __STDC__
b3_start()
#else
b3_start()
#endif
```

Chapter 21: User Model Interface (UMI)

User Files

```
{  
/* Use of this function is optional */  
printf("Start: Use of this function is optional.\n");  
return;  
}  
void  
#ifdef __STDC__  
b3_conclude()  
#else  
b3_conclude()  
#endif  
{  
/* Use of this function is optional */  
printf("Conclusion for UMI: Use of this function is optional.\n");  
return;  
} /* end of b3main.c file */  
*****
```

Model Parameter Processing File b3readmodel.c

This file contains the b3_readmodel() function which corresponds to the read_model() function. For each pair of the model parameter and its associated value, HSIM calls this function once. If the model parameter name is recognized by this function, the value is stored. Otherwise, this function prints a Warning message. A small example is listed below.

Example 58 b3_readmodel() File Example

```
*****  
/* */  
/* Copyright 1998 - 2004 */  
/* Synopsys Corporation */  
/* */  
/* Module : b3readmodel.c */  
/* Last Update : Jan.2004 */  
/* Description : get BSIM3 Model parameters */  
  
#include <stdio.h>  
#include <string.h>  
#include <math.h>  
#include "UMI.h"  
#include "b3defs.h"  
  
void  
#ifdef __STDC__  
b3_readmodel(  
B3model *model,  
char *pname,  
double value)  
#else  
b3_readmodel(model,pname,value)  
B3model *model;  
char *pname;  
double value;  
#endif  
{  
/* process every model parameter value specified by the user */  
return;  
} /* end of b3readmodel.c file */  
*****
```

Model Default Value Setting File b3setmodel.c

This file contains the b3_setmodel() function which corresponds to the set_model() function in the UMI.h header file. HSIM calls this function to assign default value to a model parameter if you do not provide input value for that particular model parameter. A small example is listed below.

Chapter 21: User Model Interface (UMI)

User Files

Example 59 b3_setmodel() File Example

```
*****  
/* Copyright 1998 - 2004 */  
/* Synopsys Corporation */  
/* */  
/* Module : b3set.c */  
/* Last Update : Jan.2004 */  
/* Description : assign default BSIM3 model */  
/* parameter values */  
  
#include <stdio.h>  
#include <string.h>  
#include <math.h>  
#include "UMI.h"  
#include "b3defs.h"  
  
void  
#ifdef __STDC__  
b3_setmodel(  
TEMPS *ckt_temp,  
B3model *model)  
#else  
b3_setmodel(ckt_temp,model)  
TEMPS *ckt_temp;  
B3model *model;  
#endif  
{  
/* assign default values to those model parameters  
that do not have input values */  
  
return;  
} /* end of b3setmodel */  
*****
```

Geometry Assignment File b3assigngeometry.c

This file contains the b3_assigngeometry() function which corresponds to the assign_geometry() function in the UMI.h header file. The geometric information about a transistor instance includes width w, length l, drain-diode area ad, source-diode area as, drain-diode periphery pd., and source-diode periphery ps. To pass each piece of geometric information into the dynamic library user.so, HSIM will call this function once. A small example is listed below.

Example 60 b3_assigngeometry() File Example

```
*****  
/* Copyright 1998 - 2004 */  
/* Synopsys Corporation */  
/* */  
/* Module      : b3assigngeometry.c      */  
/* Last Update : Jan.2004      */  
/* Description : assign BSIM3 geometry parameters      */  
  
#include <stdio.h>  
#include <string.h>  
#include <math.h>  
#include "UMI.h"  
#include "b3defs.h"  
  
void  
#ifdef __STDC__  
b3_assigngeometry(  
B3instance *here,  
char *name,  
double value)  
#else  
b3_assigngeometry(here, name, value)  
B3instance *here;  
char *name;  
double value;  
#endif  
{  
/* geometric information to be used in model equation  
   calculation is processed at this function */  
return;  
} /* end of b3_assigngeometry.c file */  
*****
```

Temperature Updating File b3temp.c

This file contains the b3_tempgeometry() function which corresponds to the temp_geometry() function. The information about nominal temperature t_{nom} and simulation temperature $temp$ are passed from HSIM. Model parameter values and transistor instance parameter values are updated for the given simulation temperature. A small example is listed below.

Chapter 21: User Model Interface (UMI)

User Files

Example 61 b3_tempgeometry() File Example

```
*****  
/* Copyright 1998 - 2004*/  
/* Synopsys Corporation */  
/* */  
/* Module      : b3temp.c */  
/* Last Update : Jan.2004*/  
/* Description : temperature-updated model parameters*/  
  
#include <stdio.h>  
#include <string.h>  
#include <math.h>  
#include <stdlib.h>  
#include "UMI.h"  
#include "b3defs.h"  
  
void  
#ifdef __STDC__  
b3_tempgeometry(  
    TEMPS      *ckt_temp,  
    B3model    *model,  
    B3instance *here)  
#else  
b3_tempgeometry(ckt_temp, model, here)  
    TEMPS      *ckt_temp;  
    B3model    *model;  
    B3instance *here;  
#endif  
{  
    /* temperature-related parameter values are updated at this  
    function for the given simulation temperature */  
    return;  
} /* end of b3temp.c file */  
*****
```

Model Evaluation and Load File b3load.c

The file b3load.c contains the b3_load() function which corresponds to the model_load() function in the [Header File UMI.h on page 601](#).

Biassing voltages vds, vgs, vbs, and device mode are passed from HSIM. The calculated results are returned from the dynamic library user.so to HSIM. The results can include the following: threshold voltage von, drain current ids, output conductance gds, transconductance gm, gate-source overlap capacitance cgso, gate-drain overlap capacitance cgdo, gate-bulk overlap capacitance cgbo, substrate-source junction diode current ibs, substrate-drain junction

diode current ibd, substrate-source junction diode capacitance capbs, substrate-drain junction diode capacitance capbd.

Charge-Based and Meyer Capacitance Models

If Charge-Based Capacitance Model equations are used as described in [Charge-Based Capacitance Model Example on page 619](#), the following can also be returned: gate charge qg, drain charge qd, source charge qs, and capacitance values cggb, cgdb, cgsb, cbgb, cbdb, cbsb, cdgb, cddb, and cdsb.

If the Meyer Capacitance Model equations are used as described in [Meyer Capacitance Model Example on page 622](#), the returned capop value is 0, and positive gate-source capacitance capgs, gate-drain capacitance capgd, and gate-bulk capacitance capgb are also returned. The capacitances cgso and cgdo have been multiplied by the channel width, and capacitance cgbo has already been multiplied by channel length.

Charge-Based Capacitance Model Example

Content of the b3load.c follows.

Chapter 21: User Model Interface (UMI)

User Files

Example 62 b3load.c File Example

```
/*****************************************/
/* Copyright 1998 - 2004 */
/* Synopsys Corporation */
/*
 * Module      : b3load.c */
/* Last Update : Jan.2004 */
/* Description : evaluate BSIM3 current, the derivatives and
   capacitances */

#include <stdio.h>
#include <string.h>
#include <math.h>
#include "UMI.h"
#include "b3defs.h"

void
#ifdef __STDC__
b3_load(
UMI_VAR    *custom,
B3model   *model,
B3instance *here)
#else
b3_load(custom,model,here)
UMI_VAR    *custom;
B3model   *model;
B3instance *here;
#endif
{
VgsExt=custom->vgs;
VdsExt=custom->vds;
VbsExt=custom->vbs;
DevMode=custom->mode;

/* NOTE: If the model code is SPICE-like,
/* keep the following three lines in order
   to handle PMOS properly*/

vgs=model->M2type*VgsExt;
vds=model->M2type*VdsExt;
vbs=model->M2type*VbsExt;

vbd=vbs - vds;
vgd=vgs - vds;
vgb=vgs - vbs;

if (vds >=0.0) {/* normal mode */
   here->B3mode=1;
```

```
Vds=vds;
Vgs=vgs;
Vbs=vbs;
}
else /* inverse mode */
{
    here->B3mode=-1;
    Vds=-vds;
    Vgs=vgd;
    Vbs=vbd;
}
here->B3cbs/* RETURN RESULT */
here->B3cbd/* RETURN RESULT */
here->B3gds/* RETURN RESULT */
here->B3gm/* RETURN RESULT */
here->B3von/* RETURN RESULT */
here->B3cd/* RETURN RESULT */

here->B3capbs/* RETURN RESULT */
here->B3capbd/* RETURN RESULT */

here->B3cgso/* RETURN RESULT */
here->B3cgdo/* RETURN RESULT */
here->B3cgbo/* RETURN RESULT */
here->B3qgate/* RETURN RESULT */
here->B3gdrn/* RETURN RESULT */
here->B3gbulk/* RETURN RESULT */
here->B3cggb/* RETURN RESULT */
here->B3cgdb/* RETURN RESULT */
here->B3cgsb/* RETURN RESULT */
here->B3cbgb/* RETURN RESULT */
here->B3cbdb/* RETURN RESULT */
here->B3cbsb/* RETURN RESULT */
here->B3cdgb/* RETURN RESULT */
here->B3cddb/* RETURN RESULT */
here->B3cdsb/* RETURN RESULT */

/*****************/
/* return result back to interface function */
/*****************/

custom->von=here->B3von; /* threshold voltage */
custom->vdsat=here->B3vdsat; /* saturation voltage */
custom->ids=here->B3cd; /* drain current */
custom->gds=here->B3gds; /* output conductance */
custom->gm=here->B3gm; /* transconductance */

/* overlap capacitances */
custom->cgso=here->B3cgso; /* multiplied by width already */
```

Chapter 21: User Model Interface (UMI)

User Files

```
custom->cgdo=here->B3cgdo; /* multiplied by width already */
custom->cgbo=here->B3cgbo; /* multiplied by length already */

/* capop: 0 for Meyer model;    13 for charge-based model */
custom->capop=13;
custom->qg=here->B3qgate; /* gate charge */
custom->qd=here->B3qdrn; /* drain charge */
custom->qs=- (here->B3qgate+here->qdrn+here->B3qbulk);           /
 * source charge */
custom->cggb=here->B3cggb;
custom->cgdb=here->B3cgdb;
custom->cgsb=here->B3cgsb;
custom->cbgb=here->B3cbgb;
custom->cbdb=here->B3cbdb;
custom->cbsb=here->B3cbsb;
custom->cdgb=here->B3cdgb;
custom->cddb=here->B3cddb;
custom->cdsb=here->B3cdsb;

/* substrate diode */
custom->ibs=here->B3cbs; /* source-substrate diode current*/
custom->ibd=here->B3cbd; /* drain-substrate diode current */
custom->capbs=here->B3capbs; /* source-substrate diode
                                capacitance*/
custom->capbd=here->B3capbd; /* drain-substrate diode
                                capacitance */

custom->idb=
custom->isb=
custom->igd=
custom->igs=
custom->igb=
return;
} /* end of b3load.c file */
*****
```

Meyer Capacitance Model Example

The last portion of the file m2_load() is listed below and includes the HSIM return values.

Example 63 m2_load() File Example

```
*****  
/* return result back to interface function */  
*****  
  
custom->von=here->M2von; /* threshold voltage */  
custom->vdsat=here->M2vdsat; /* saturation voltage */  
custom->ids=here->M2cd; /* drain current */  
custom->gds=here->M2gds; /* output conductance */  
custom->gm=here->M2gm; /* transconductance */  
  
/* overlap capacitances */  
custom->cgso=here->M2cgso; /* multiplied by width already */  
custom->cgdo=here->M2cgdo; /* multiplied by width  
already */  
custom->cgbo=here->M2cgbo; /* multiplied by length  
already */  
  
/* capop: 0 for Meyer model; 13 for charge-based model */  
custom->capop=0;  
custom->capgs=here->M2capgs; /* Meyer model gate-source  
capacitance ( $dQg/dVgs + cgso$ ) */  
  
custom->capgd=here->M2capgd; /* Meyer model gate-drain  
capacitance ( $dQg/dVds + cgdo$ ) */  
custom->capgb=here->M2capgb; /* Meyer model gate-bulk capacitance  
( $dQg/dVbs + cgbo$ ) */  
/* substrate diode */  
custom->ibs=here->M2cbs; /* source-substrate diode  
current */  
custom->ibd=here->M2cbd; /* drain-substrate diode  
current */  
custom->capbs=here->M2capbs; /* source-substrate diode  
capacitance */  
custom->capbd=here->M2capbd; /* drain-substrate diode  
capacitance */  
return;  
} /* end of m2load.c file */  
*****
```

Feedback

Chapter 21: User Model Interface (UMI)

User Files

Flash Core Cell Model

Provides data about support of flash and MRAM core cells. Flash is a non-volatile memory technology based on a floating-gate device. The HSIM® flash core cell model permits simulation of most flash design styles including NOR, NAND structures and multi-level cells (MLC). MRAM architectures integrate magnetic devices within standard CMOS microelectronics.

- [Flash Cell Core Models](#)
- [Flashlevel = 1 Parameters and Operation](#)
- [Flashlevel = 2 Parameters and Operation](#)
- [MRAM Core Cell Models](#)

Flash Cell Core Models

The HSIM flash core cell model provides an efficient way to model NMOS-based flash memory cells. This compact modeling capability lets you verify the peripheral circuitry and its routing to the array simultaneously. The verification methodology can be applied at the pre-layout and post-layout stage. The initial Vth can be set on a cell by cell basis so that the simulation can start with the memory array in a desired state (see [HSIMDELVTO on page 80](#) and [fcc on page 481](#) interactive mode command). An optional fifth terminal is available to model the Nwell in twin-well processes.

Flash memory designers typically have access to a MOS model for the floating-gate device. The model used could be BSIM3 or BSIM4, for example. This device accurately models loading on word and bit lines as well as Ids current, but it usually does not model the floating gate and therefore does not provide a way to simulate programming or erasing events.

Chapter 22: Flash Core Cell Model

Flash Cell Core Models

The HSIM flash core cell model is an extension of the MOS model described above. It allows some level of modeling of the floating gate by continuously adjusting the threshold voltage of the device based on the conditions applied to its terminals. The programming and erasing conditions can be specified in terms of minimum or maximum terminal voltages. A built-in model is provided for the V_{th} change as a function of time, terminal voltages and built-in parameters. See the *Using the HSIM User Flash Interface (UFI)* application note on Solvnet for custom V_{th} change equations.

Floating Gate Modeling

The floating gate modeling is basic. It provides enough flexibility to model the functionality of NOR and NAND bit-cells, as well as Multi-level cells (MLC). It can be used for floating gate devices as long as the effects used for programming and erasing can be described in terms of minimum or maximum terminal voltages, which is the case for various technologies, including Channel Hot-Electron (CHE) programming and Fowler-Nordheim (F-N) tunneling. Since the model is not physics-based, there are limitations in terms of the effects that can be modeled. For example, the charge flowing from and to the floating gate is not seen on the device terminals.

Defining and Instantiating a Flash Model

The flash cell model is an extension of a base NMOS transistor model. BSIM3, BSIM4, MOS1, MOS9, MOS11 and EKV models are supported as base model for flash cell modeling extension. The base NMOS model card is defined using the .model statement (see [MOSFET on page 249](#)).

The flash extension consists of an additional set of model parameters. It is enabled when the flashlevel parameter is set to 1 or 2. See [Flashlevel = 1 Parameters and Operation on page 628](#) and [Flashlevel = 2 Parameters and Operation on page 632](#) sections for detailed description. The additional set of parameters of the flash extension can be added directly to the base NMOS model card or defined in a separate model card using the .model statement and then added to the base NMOS model card using the .appendmodel statement using the following syntax:

```
.appendmodel source_model destination_model  
where source_model is the model card to be appended to the  
destination_model.
```

The following example shows a flash model definition and instantiation.

```
*Base nmos model card
.model mos1 nmos level=1 vto=2.5 kp=25u gamma=0 lambda=0.1 phi=0

*Flash model extension model card
.model flash flashcell flashlevel=1

*Adding the flash extension to the base nmos model
.appendmodel flash mos1

*Device instantiation
m1 nd ng ns nb mos1 L=1u W=1u
```

Optional Nwell Terminal

For twin-well processes, you can specify the Nwell terminal using the optional B2 parameter. If the Nwell is specified using the B2 instance parameter, a simple level=1 diode is inserted between this terminal and the pwell bulk (fourth terminal). Note that this diode is only taken into account by HSIM only when HSIMBMOS or HSIMABMOS is set. See [HSIMBMOS on page 70](#) or [HSIMABMOS on page 52](#).

The following example uses the optional Nwell terminal:

```
* Device instantiation
m1 nd ng ns nb mos1 L=1u W=1n B2=nb2
```

Conventions for HSIM Flash Core Cell Models

In this chapter, programming normally refers to an increase of the threshold voltage of the NMOS transistor, while erasing normally refers to a decrease of the threshold voltage. A flash device enters a program or an erase event when its terminals meet the program or erase conditions respectively. The event ends when the conditions are no longer met. Also, V(S) always refers to the lowest voltage between terminal 1 and terminal 3. Conversely, V(D) always refers to the highest voltage between terminal 1 and terminal 3. Effectively, source and drain are swapped during simulation such that these definitions remain true.

Initializing and Probing Flash Core Cell Models

You can use the delvto instance parameter available for the base model to initialize flash core cells. Alternately, you can use [HSIMDELVTO on page 80](#).

You can plot transient waveforms for the threshold voltage and the threshold voltage change due to program and erase events using LV9 and LV0 element templates, respectively. For example:

```
print lv9(x1.m*)
* Plots Vth for mosfets in x1

print lv0(x1.m*)
* Plots Vth shift due to program/erase for mosfets in x1
```

Flashlevel = 1 Parameters and Operation

Flashlevel =1 is a simple model that targets NOR flash technology. The programming and erasing conditions are fixed in terms of maximum or minimum voltage on the device terminals. All Vth programming and erasing behavior is controlled by built-in equations.

Program Event

A flash device enters a program event when all of the following conditions are met:

Drain: $V(D) > VDPGMMIN$

Gate: $V(G) > VPGMMIN$

Source: $V(S) < VSPGMMAX$

Bulk: $V(B) < VPWPGMMAX$

Nwell (if B2 is defined):

- If no VNWPBM* parameter is set or VNPGMMAX is set, $V(B2) < VNWPBMMAX$.
- If VNWPBMMIN only is set, $V(B2) > VNWPBMMIN$.
- If VNWPBMMAX and VNWPBMMIN are both set, $V(B2) > VNWPBMMIN$.

Erase Event

An erase event occurs when all of the following conditions are met:

Drain: $V(D) > VDERSMIN$

Gate: $V(G) < VGERSMAX$

Source: $V(S) > VSERSMIN$

Bulk: $V(B) > VPWERSMIN$

Nwell (if B2 is defined):

- If no VNWERS* parameter is set or VNWERSMIN is set, $V(B2) > VNWERSMIN$.
 - If VNWERSMAX only is set, $V(B2) < VNWERSMAX$.
 - If both VNWERSMIN and VNWERSMAX are set, $V(B2) < VNWERSMAX$.
-

Flashlevel = 1 Voltage Threshold Change

Let VTH_{init} be the VTH at the time a program or erase event begins. During a program event VTH increase for TPGMSTEP period of time is given by:

$$KPGM * (V(G) - VTPGMSAT - VTH) * (V(D) - VDPGMMIN)$$

This VTH increase is prorated to the time step taken by the simulator. The maximum for VTH is the minimum of:

$V(G)$

$VTH_{init} + \text{abs}(VTPGMMAXSHIFT)$

$VTHIGH - VTPGMSAT$

During an erase event the VTH decrease for TERSSSTEP period of time is given by:

$$KERS * (VTH - V(G) - VTERSSAT) * (V(S) - VSERSMIN)$$

This VTH decrease is prorated to the time step taken by the simulator. The minimum for VTH is the maximum of:

$V(G)$

$VTH_{init} - \text{abs}(VTERSSMAXSHIFT)$

$VTLOW + VTERSSAT$

Flashlevel = 1 Parameter List and Default Values

Parameters	Default Setting
FLASHLEVEL	Must be set to 1.
VDPGMMIN	3.0
VGPGMMIN	8.0
VSPGMMAX	1.0
VPWPGMMAX	1.0
VNWPGMMAX	2.0
VNWPGMMIN	No default, active when set.
VDERSMIN	7.0
VGERSMAX	-8.0
VSERSMIN	-8.0
VPWERSMIN	7.0
VNWERSMIN	7.0
VNWERSMAX	No default, active when set.
TPGMSTEP	1n
KPGM	1m
VTPGMSAT	0.0
VTPGMMAXSHIFT	5.0
VTHIGH	7.0
TERSSTEP	1n
KERS	1m

Parameters	Default Setting
VTERSSAT	0.0
VTERSMAXSHIFT	5.0
VTLOW	0.0

Flashlevel = 1 Single Cell Simulation Example

The following example shows the flashlevel=1 default behavior.

```
*signal printing section
.print lv9(m*) lv0(m*) v(*) i(*)

*nmos model definition section
.model mos1 nmos level=1 vto=2.5 kp=25u gamma=0 lambda=0.1 phi=0

*flash model definition section
.model flash flashcell flashlevel=1
.appendmodel flash mos1

*device instantiation section
m1 nd ng ns nb mos1 L=1u W=1u B2=nb2

*stimuli section
Vng ng 0
+pwl(0.5u 5.5 0.6u 9 1u 9 1.1u 5.5 1.5u 5.5 1.6u -9 2u -9 2.1u 5.5)
Vnd nd 0
+pwl(0.5u 0.9 0.6u 4.5 1u 4.5 1.1u 0.9 1.5u 0.9 1.6u -6 2u -62.1u
0.9)
Vns ns 0 pwl(1u 0 1.1u 0 1.5u 0 1.6u 8 2u 8 2.1u 0)
Vnb nb 0 pwl(0.5u -8 0.6u 1 1u 1 1.1u -8 1.5u -8 1.6u 8 2u 8 2.1u -8)
Vnb2 nb2 0 10

*simulation setup section
.tran 10n 2.5u
.param hsimsncts=0
.end
```

Flashlevel = 2 Parameters and Operation

Based on the same concept as flashlevel = 1, flashlevel = 2 offers more flexibility, which makes it suitable to model NAND technology for example. The flexibility is increased by:

- Allowing the programming and erasing condition on each terminal to be a minimum or a maximum.
- Allowing the use of custom equations for the V_{th} change in programming and erasing conditions through the user flash interface (UFI).

This section documents the default flashlevel=2 operation, which uses a built-in equation for V_{th} change. See the *Using the HSIM Flash User Interface (UFI)* application note on SolvNet for information about using custom equations for V_{th} change.

Flashlevel = 2 Program and Erase Events

A flash device enters a program or an erase event when its terminals meet the program or erase conditions respectively. The event ends when the conditions are no longer met. In flashlevel=2, a minimum or a maximum can be defined for each terminal, but not both. If both a minimum and maximum are defined for a given terminal, HSIM errors out.

Program Event

A program event occurs when the following program conditions are met simultaneously:

Drain:

If VDPGMMIN is set

V(D) > VDPGMMIN

else

V(D) < VDPGMMAX

Gate:

If VGPGMMAX is set

V(G) < VGPGMMAX
else
V(G) > VGPGMMIN

Source:

If VSPGMMIN is set
V(S) > VSPGMMIN
else
V(S) < VSPGMMAX

Bulk:

If VPWPGMMIN is set
V(B) > VPWPGMMIN
else
V(B) < VPWPGMMAX

N-well (if B2 is defined):
If VNWPGMMIN is set
V(B2) > VNWPGMMIN
else
V(B2) < VNWPGMMAX

Erase Event

An erase event occurs when the following erase conditions are met simultaneously:

Drain:

If VDERSMAX is set
V(D) < VDERSMAX
else
V(D) > VDERSMIN

Chapter 22: Flash Core Cell Model

Flashlevel = 2 Parameters and Operation

Gate:

If VGERSMIN is set

V(G) > VGERSMIN

else

V(G) < VGERSMAX

Source:

If VSERSMAX is set

V(S) < VSERSMAX

else

V(S) > VSERSMIN

Bulk:

If VPWERSMAX is set

V(B) < VPWERSMAX

else

V(B) > VPWERSMIN

Nwell (if B2 is defined):

If VNWPGBMAX is set

V(B2) < VNWPGBMAX

else

V(B2) > VNWPGBMIN

Flashlevel=2 Threshold Voltage Change

This section describes the default behavior for flashlevel=2. If the HSIMUFI parameter is defined, the V_{th} change equation are defined by the user. See the *Using the HSIM Flash User Interface (UFI)* application note on Solvnet for information about using custom equations for V_{th} change.

Let VTH_init be the VTH at the time program or erase event begins. During a program event:

VTH increase for TPGMSTEP period of time is given by

If VDPGMMIN is set

$$KPGM * (V(G) - VTPGMSAT - VTH) * (V(D) - VDPGMMIN)$$

else

$$KPGM * (V(G) - VTPGMSAT - VTH) * (VDPGMMAX - V(D))$$

This VTH increase is prorated to the time step taken by the simulator. The maximum for VTH is the minimum of:

V(G)

$$VTH_init + \text{abs}(VTPGMMAXSHIFT)$$

$$VTHIGH - VTPGMSAT$$

During an erase event:

VTH decrease for TERSSSTEP period of time is given by

If VPWERSMAX is set

$$KERS * (VTH - V(G) - VTERSSAT) * (VPWERSMAX - V(B))$$

else

$$KERS * (VTH - V(G) - VTERSSAT) * (V(B) - VPWERSMIN)$$

This VTH decrease is prorated to the time step taken by the simulator. The minimum for VTH is the maximum of:

V(G)

$$VTH_init - \text{abs}(VTERSMAXSHIFT)$$

$$VTLOW + VTERSSAT$$

Flashlevel=2 Parameter List and Default Values

Parameters	Default Setting
FLASHLEVEL	Must be set to 2.
VDPGMMIN	No default, active when set.
VDPGMMAX	3.0
VGPGMMIN	15.0
VGPGMAX	No default, active when set.
VSPGMMIN	No default, active when set.
VSPGMMAX	1.0
VPWPGMMIN	No default, active when set.
VPWPGMAX	1.0
VNWPGMMIN	No default, active when set.
VNWPGMAX	1.0
VDPGMMIN	15.0
VDPGMMAX	No default, active when set.
VGPGMMIN	No default, active when set.
VGPGMAX	1.0
VSPGMMIN	15.0
VSPGMMAX	No default, active when set.
VPWPGMMIN	15.0
VPWPGMAX	No default, active when set.
VNWPGMMIN	15.0

Parameters	Default Setting
VNWPGMMAX	No default, active when set.
TPGMSTEP	1n
KPGM	1m
VTPGMSAT	0.0
VTPGMMAXSHIFT	5.0
VTHIGH	7.0
TERSSTEP	1n
KERS	1m
VTERSSAT	0.0
VTERS MAXSHIFT	5.0
VTLOW	0.0

MRAM Core Cell Models

MRAM architectures integrate magnetic devices within standard CMOS microelectronics. Unlike other memory technologies, MRAM data is stored as a magnetic state, rather than electrical charge. The elements are formed from two ferromagnetic plates, each of which can hold a magnetic field, separated by a thin insulating layer.

The read (or “sensing”) of the magnetic state is accomplished by measuring the electrical resistance of the cell. Due to the magnetic tunnel effect, the electrical resistance of the cell changes due to the orientation of the fields in the two magnetic plates. By measuring the current flowing through the cell, the resistance inside a cell can be determined. Typically, if the two magnetic plates have the same polarity (parallel) this is considered to mean a "0" state (or R_{MIN}), while if the two plates are of opposite polarity (anti-parallel) the resistance will be higher (R_{MAX}) meaning a "1" state.

Chapter 22: Flash Core Cell Model

MRAM Core Cell Models

Due to the various different MRAM architectures, MRAM core cells differ in structure, functionality and write/read cycles. There are several types of MRAM core cells supported within HSIM. Users may choose to use one of the supported MRAM core cell models to properly define the correct functionality of their MRAM designs.

Currently, HSIM supports three MRAM core cell architectures with 0, 1, and 2 word lines:

- Spin-Torque-Transfer MRAM - MRES0
 - Bidirectional symmetrical write
 - Write driver has bidirectional current flow (behaves as a current source AND current sink)
 - Parallelizing-direction read
- Dual 'Active' Layer MRAM - MRES1
 - Contains two 'active' magnetic layers whose polarity can be switched
 - Symmetrical writing current and time between "0" and "1"
 - Two-cycle read
- Toggle MRAM - MRES2
 - Same pulse sequence used to write "0" to "1" or "1" to "0"
 - Toggle current magnetic state to the opposite state with each execution

Spin-Torque-Transfer (STT) MRAM Core Cell Model (MRES0)

Spin-torque-transfer (STT) MRAM uses spin-aligned ("polarized") electrons to directly torque the magnetic domains. Specifically, if the electrons flowing into a layer have to change their spin, this will develop a torque that will be transferred to the nearby layer.

A stream of conducting electrons moving through the fixed magnetic layer are spin polarized (that is, most of the electrons' spins become aligned to that of the fixed layer). When these spin-polarized electrons pass through the free layer, they become re-polarized. In re-polarizing, the free layer magnet experiences a torque associated with the change in angular momentum resulting from the rotation of the spins. This torque pumps enough energy to reverse the orientation of the free layer magnet.

This spin-alignment of the electrons is ultimately achieved by changing the direction of the write current requiring a bidirectional and symmetrical write current.

The magnetic tunnel junction (MTJ) or magnetic device within the STT MRAM core cell consists of two ferromagnetic layers, one free to change polarity (sense layer) and the other fixed to some predetermined polarity (reference layer).

Graphically, the STT MRAM core cell can be represented as a 2-terminal device, consisting of a bidirectional bit line (BL) shown in Figure 31.

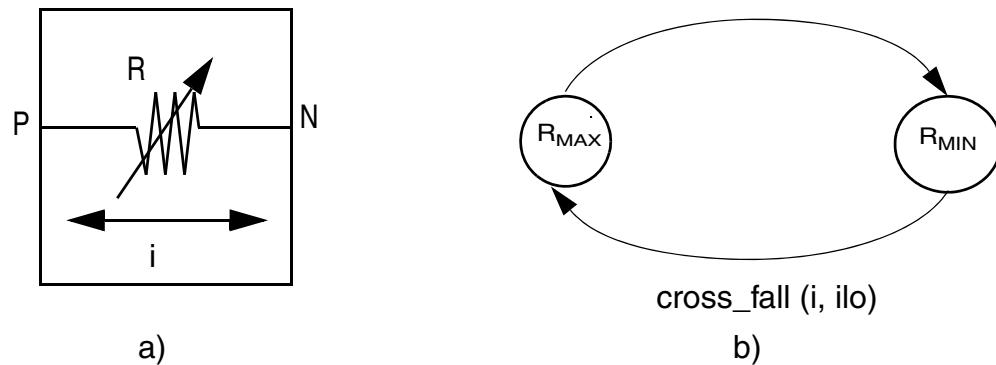


Figure 31 STT MRAM 2-terminal device (MRES0) and b) STT MRAM state sequence and current requirements

MRES0 - STT MRAM Core Cell Definition

Gxx p n MRES0

- + rmin=<val>
- + rmax=<val>
- + ihi=<val>
- + ilo=<val>
- + [ic=<val>]
- + [iwin=<val>]

MRES0 - Supported Parameter Description

rmin

minimal resistance of 'parallel' state of the MRAM cell dipoles

Chapter 22: Flash Core Cell Model

MRAM Core Cell Models

max

maximal resistance of 'anti-parallel' state of the MRAM cell dipole

ihi

i upper threshold

ilo

i lower threshold

ic

initial state of MRAM cell; default=0 <Boolean> (0 ~ rmin; 1 ~ rmax)

iwin

 time duration window which current must be above threshold (ihi | ilo);
 default=0**Note:** This current threshold condition must always be followed: ihi > ilo iwin - Check when current crosses threshold, the current remains
 above (or below) that threshold for a certain period of time
 (specified by iwin) in order to cause a switch. If the current falls
 before meeting the iwin criteria, the Examtime data is reset.**MRES0 Instantiation Example**

G0 p n MRES0 rmin=1k rmax=1.2k ihi=1m ilo=-1m

MRES0 - STT MRAM Core Cell Functionality

```
if (cross_rise (i, ihi)) {  
    state = rmin ;  
} elseif (cross_fall (i, ilo)) {  
    state = rmax ;  
}
```

Limitations and Assumptions for the MRES0 Core Model

Current thresholds (ihi and ilo)

- i is bi-directional current, which can flow into or out of the MRAM cell
- The bi-directional functionality determines the state to which the cell resolves
- The current flowing may either be 'positive' (flowing in one direction) or 'negative' (flowing in the opposite direction)

Therefore, ihi and ilo are typically close in magnitude but opposite in sign:
 $ihi \gg ilo$.

Dual-Active Layer (DAL) MRAM (MRES1)

Similar to the Toggle and STT MRAM architectures, the MTJ within the Dual-Active Layer (DAL) MRAM core cell also consists of two ferromagnetic layers. However, within the DAL MRAM both magnetic layers are free to change polarity. For naming differentiation from the other architectures, they are referred to here as the soft (S) and hard (H) layers.

The word line (WL) is bidirectional, therefore the direction and magnitude of the WL current will determine the resulting polarity of the two dipoles. The S layer dipole is controlled via the WL current $i1 > i1shi \parallel i1 < i1slo$. At this current value ($i1shi \parallel i1slo$), the H layer dipole is not affected, as it is thicker. The H layer dipole is controlled via $i1 > i1hhi \parallel i1 < i1hlo$. $i1shi < i1hhi \&& i1slo > i1hlo$, therefore changing the H layer dipole may also cause a change in the S layer dipole.

In order to determine the state of the cell or orientation of the H layer, a two-cycle read operation is required. The H layer dipole direction is determined by flipping the S layer only ($i1shi < i1 < i1hhi$) first in one direction (shaping the S layer dipole in a fixed orientation) and then reversing the direction if the $i1$ ($i1hlo < i1 < i1slo$) current (shaping the S layer dipole in the opposite orientation).

The difference in resistance measured between the two read cycles determines the orientation of the H layer and thus the state of the cell. If the S and H layer dipoles are parallel (or aligned), the measured resistance will be lower (R_{MIN}) than if the dipoles of the S and H layers were in an anti-parallel (or unaligned) position (R_{MAX})

The model also requires a sense line (SL) running in between the two magnetic layers used to help change the polarity of the magnetic layers as well as help determine resistance shifts.

Feedback

Chapter 22: Flash Core Cell Model

MRAM Core Cell Models

Graphically, the DAL MRAM core cell can be represented as a 4-terminal device, consisting of a bidirectional word line (WL) and a unidirectional sense line (SL) and a two-state structure (one for each magnetic layer) shown in Figure 32.

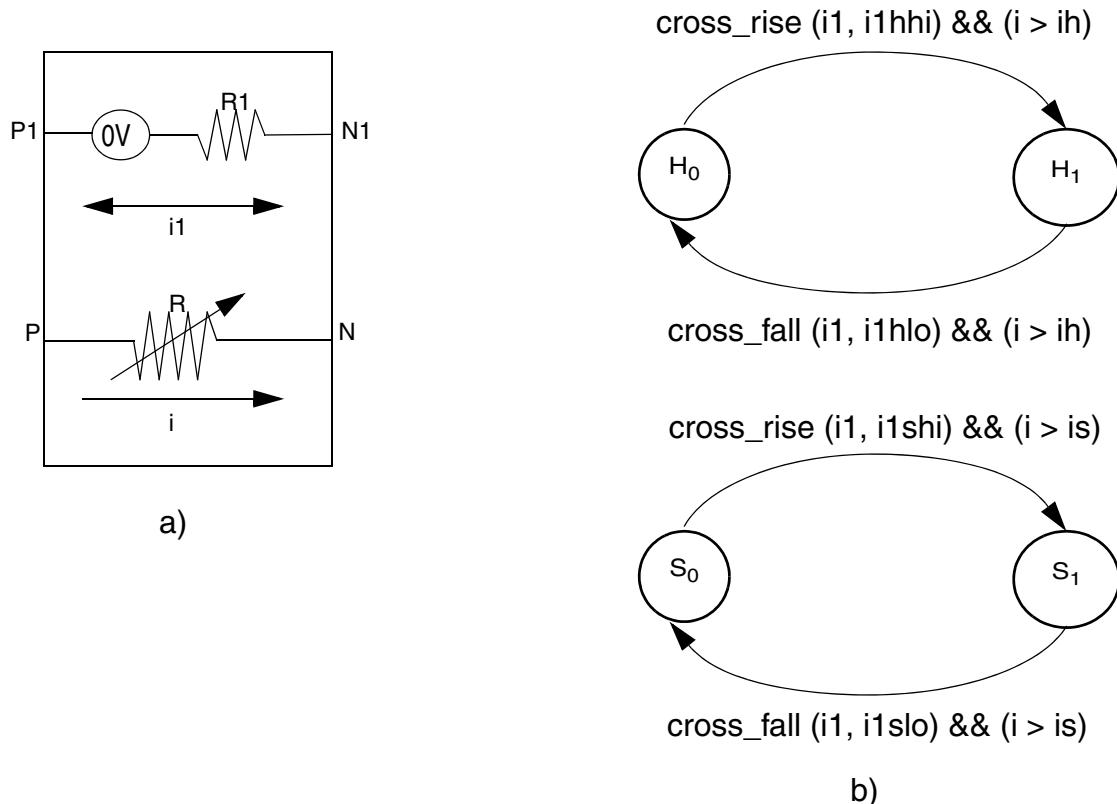


Figure 32 DAL MRAM 4-terminal device (MRES1) and DAL MRAM state sequence and current requirements

MRES1 - DAL MRAM Core Cell Definition

```
Gxx p n MRES1 p1 n1  
+ rmin=<val>  
+ rmax=<val>  
+ ih=<val>  
+ is=<val>  
+ i1hh=<val>
```

+ i1hlo=<val>
+ i1shi=<val>
+ i1slo=<val>
+ [r1=<val>]
+ [ich=<val>]
+ [ics=<val>]

MRES1 - Supported Parameter Description

min

minimal resistance of 'parallel' state of the MRAM cell dipoles

max

maximal resistance of 'anti-parallel' state of the MRAM cell dipoles

i_h

i threshold for magnetic hard (H) layer

i_s

i threshold for magnetic soft (S) layer

i1hh_i

i1 upper threshold for magnetic hard (H) layer

i1hlo

i1 lower threshold for magnetic hard (H) layer

i1shi

i1 upper threshold for magnetic soft (S) layer

i1slo

i1 lower threshold for magnetic soft (S) layer

r₁

additional internal line resistance (optional and defaults to 10mΩ ('10m'))

ich

initial state of MRAM hard layer; default=0 <Boolean>

Chapter 22: Flash Core Cell Model

MRAM Core Cell Models

ics

initial state of MRAM soft layer; default=0 <Boolean>

Note: These current threshold conditions must always be followed:

ih > is

i1hhi > i1shi > i1slo > i1hlo

MRES1 Instantiation Example

```
G1 p n MRES1 p1 n1 rmin=1k rmax=1.2k
+ ih=3m is=2m i1hhi=15m i1hlo=-15m i1shi=10m i1slo=-10
```

MRES1 - DAL MRAM Core Cell Functionality

```
if (cross_rise (i1, i1hhi)) {
    if (i > ih) {
        stateH = 1 ;
    }
} elseif (cross_fall (i1, i1hlo)) {
    if (i > ih) {
        stateH = 0 ;
}

if (cross_rise (i1, i1shi)) {
    if (i > is) {
        stateS = 1 ;
}
} elseif (cross_fall (i1, i1slo)) {
    if (i > is) {
        stateS = 0 ;
}

if (stateH == stateS) {
    R = rmin ;
} elseif (stateH != stateS) {
R = rmax ;
}
```

Limitations and Assumptions for the MRES1 Core Model

Current thresholds for i1:

- i1 is bi-direction current which flow into or out of the MRAM cell
- The bi-directional functionality determines the state to which the cell resolves
- The current flowing may either be 'positive' (flowing in one direction) or 'negative' (flowing in the opposite direction)

Therefore, ihi and ilo are typically close in magnitude but opposite in sign:

i1hhi >> i1hlo && i1shi >> i1slo

ih > is

i1hhi > i1shi > i1slo > i1hlo

Toggle MRAM Core Cell Model (MRES2)

The Toggle MRAM architecture obtains its name from its use of the same pulse sequence when writing a "0" to "1" or "1" to "0" state. Each time the write current sequence is executed, the magnetic device changes from its current magnetic state to the opposite magnetic state.

Similar to the STT MRAM architecture, the MTJ within the Toggle MRAM core cell also consists of two ferromagnetic layers, one free to change polarity and the other fixed to some predetermined polarity.

By applying a current pulse sequence through two independent write lines, a rotating magnetic field is generated which moves the free, or sense magnetic layer, from one state to the other. Since the write sequence toggles the bit to its opposite state regardless of its existing state, a pre-read must be performed to determine if a write is required.

Graphically, the toggle MRAM core cell can be represented as a 6-terminal device, consisting of unidirectional write-through bit line (BL) and word lines (WL1 and WL2) shown in [Figure 33 on page 646](#).

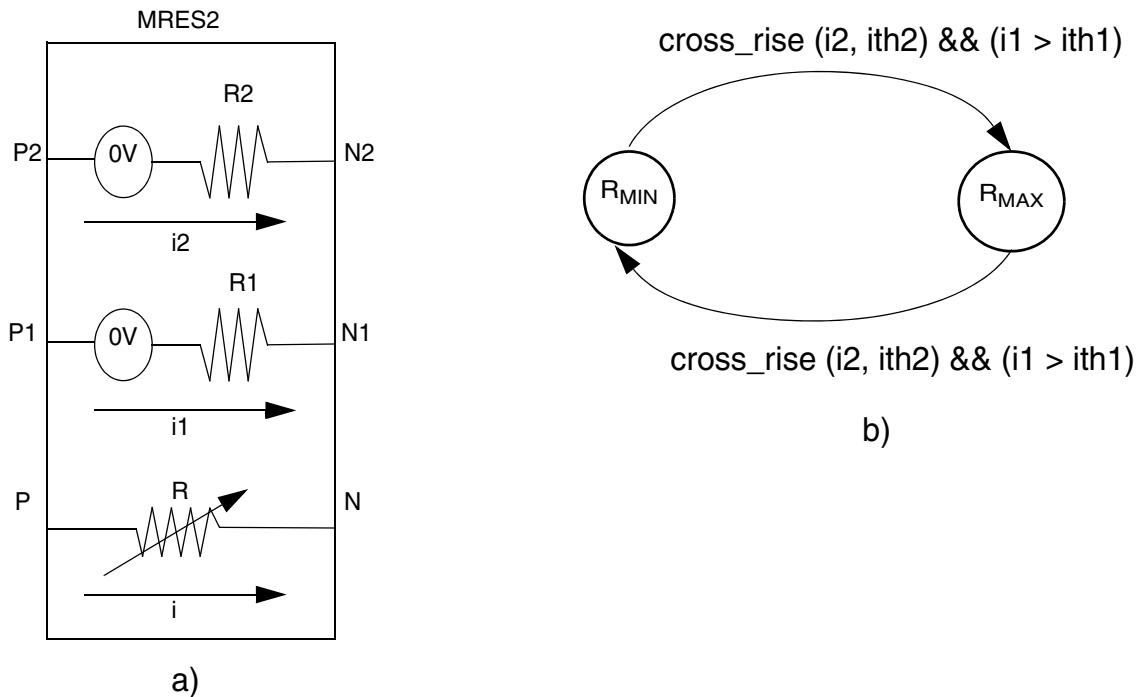


Figure 33 Toggle MRAM 6-terminal device (MRES2) and Toggle MRAM state sequence and current requirements

Based on the same symmetrical current write sequence, the state of the Toggle MRAM core cell will toggle from one state to the other. The memory cell has two states of resistance values corresponding to the direction of the magnetization in the free layer with respect to that of the fixed layer. When the two magnetic layers are parallel, the cell is in the low resistance state (R_{MIN}) and when the two magnetic layers are anti-parallel, the cell is in the high resistance state (R_{MAX}). See [Figure 33 on page 646](#).

MRES2 - Toggle MRAM Core Cell Definition

```
Gxx p n MRES2 p1 n1 p2 n2
+ rmin=<val>
+ rmax=<val>
+ ith1=<val>
+ ith2<val>
+ [r1=<val> r2=<val>]
```

+ [ic=<val>]

MRES2 - Supported Parameter Description

min

minimal resistance of 'parallel' state of the MRAM cell dipoles

max

maximal resistance of 'anti-parallel' state of the MRAM cell dipole

ith1

i1 threshold value of current flowing through word line 1 (WL1)

ith2

i2 threshold value of current flowing through word line 2 (WL2)

r1

additional internal word line (WL1) resistance (optional and defaults to 10mΩ ('10m'))

r2

additional internal word line (WL2) resistance (optional and defaults to 10mΩ ('10m'))

ic

initial state of MRAM cell; default=0 <Boolean> (0 ~ rmin; 1 ~rmax)

MRES2 Instantiation Example

```
G2 p n MRES2 p1 n1 p2 n2 rmin=1k rmax=1.2k ith1=3m ith2=2m
```

MRES2 - Toggle MRAM Core Cell Functionality

```
if (cross_rise (i2, ith2) && (i1 > ith1)) {
    state = ~state;
}
```

Limitation of MRES Elements

To improve simulation performance, HSIM takes advantage of latent partitions and does not simulate such circuits considered latent or 'asleep.' Latency within

Chapter 22: Flash Core Cell Model

MRAM Core Cell Models

HSIM is globally controlled using the [HSIMSNCS on page 148](#) command. HSIM by default does, however, automatically consider any partitions to be latent if the current of its entire boundary and internal nodes are less than [HSIMSTEADYCURRENT on page 172](#).

MRES core cell elements are current-based voltage controlled resistors where the current across the device is sensed in order to determine the cell's state or functionality.

Therefore, if the current thresholds for the MRES cells are set to a number less than HSIMSTEADYCURRENT, HSIM will consider them latent and not simulate them.

Monitoring MRAM Array State Conditions

HSIM has the ability to monitor and report the initial state conditions and/or state transition information of any MRAM bit cells when using the MRES built-in proprietary MRAM models. See [HSIMMONITORMRES on page 108](#).

C Language Functional Model

Describes HSIM functional element support written in C language and the electrical elements defined in the SPICE netlist. This chapter provides information about C language functional models. The instantiation of each C language functional model is listed in the SPICE netlist file and is defined as a regular electrical subcircuit instantiation. The chapter includes a list of API functions that perform various tasks, reference information for compiling and linking the C language functional models, and example code for A/D and D/A converters and RAM.

- [Model Flow Overview](#)
- [Windows NT/2000/XP Requirements](#)
- [HP-UX Requirements](#)
- [Model Definition](#)
- [Model Interfaces](#)
- [Modeling Memory Core](#)
- [Examples](#)

Model Flow Overview

A functional model written in standard C language can be compiled and linked with HSIM. This requires a C compiler in the system. The file buildfmod, in directory \$HSIM_HOME/bin, is provided to compile the C functional model.

```
% buildfmod abc.so a1.c a2.c a3.c
```

buildfmod compiles C files a1.c, a2.c and a3.c into the library file abc.so.

HSIM supports the functional elements described in C language as well as the electrical elements defined in the SPICE netlist. While all electrical elements are defined in the SPICE netlist file, the C functional model must be described in a separate source code .c file. This chapter provides information about functional models in the C language.

- The instantiation of each C functional model is listed in SPICE netlist file and is defined as regular electrical subcircuit instantiation. Defining the model is described in [Model Definition on page 651](#).
- A list of API functions that perform various tasks is provided in [Model Interfaces on page 660](#).
- Compiling and linking the C functional models are described in [Model Flow Overview on page 649](#).
- Example code for A/D and D/A converters and RAM are provided in [Examples on page 698](#).

Windows NT/2000/XP Requirements

If using the Windows NT or Windows 2000 platform, Visual C++ must be installed and used to run the file VCVARS32.BAT, which is provided by Visual C++. This bat file will set up the correct path to use nmake as shown in the following example:

```
% nmake /f c:/hsim2.0/etc/include/buildfmod.mak SOURCES="a1.c  
a2.c a3.c" LIBNAME=abc.dll
```

This one line command compiles C files a1.c, a2.c, and a3.c into the library file abc.dll. This example assumes that the name of the directory for HSIM installation in your machine is hsim2.0 in the C drive. If necessary, replace with the suitable directory name.

HP-UX Requirements

If using HP-UX, a compiler option is required to generate a model library with position-independent code.

The library file can be linked to the circuit netlist by adding [HSIMFMODLIB on page 90](#) code in the netlist, as follows:

```
.param HSIMFMODLIB=". /abc.so"
```

Model Definition

Each C functional model you write must be defined within an HSIM functional model interface function `hsimC_model`. The following model definition APIs are provided to define the model name, interface ports, and internal states.

Signal direction at the interface ports can be defined as one of the following:

`hmInput`

Defines the port to be an input port such that the functional model element is driven by this input port.

`hmOutput`

Defines the port to be an output port such that the functional model element drives this output port.

`hmBiput`

Defines the port to be a bidirectional (biput) port such that the functional model element can drive the port at some time period, and can be driven by the outside driver connecting to the port at some other time period.

Each port defined by the functions listed in [Model Port APIs on page 653](#) connects a physical node in the circuit. Each port definition is similar to the argument in a C function.

Local variables can be defined within the model function. This allows each local variable, although not present in the circuit database, to retain value after exiting the mode. Also, each local variable can be retrieved when the same model is re-evaluated later.

Local variables are defined as the state of the model function. Functionally, this is similar to the static local variable within the C function—the content is not lost after exiting from the function.

There is a difference between the state of model function and the static variable with the C function. There is only one static variable storage such that the same storage is used in each C function evaluation. In case of model state, the state storage is instance based such that each instance of the model function has its own storage of the state.

Model Creation APIs

This section lists the model creation APIs:

- [hmCreateModel](#)
- [hmSetModelAttr](#)

hmCreateModel

```
pHMMODEL hmCreateModel (model_name, function_name)  
model_name  
char  
function_name  
pFUNC
```

`hmCreateModel` creates a pointer link, which has the data type of `pHMMODEL`, between a functional model in the netlist and a C functional model you write. The pointer serves as the ID number for any later reference to this model. To create the model, the model name defined in the netlist is given in the string `model_name`. The function name is specified by `function_name` which has a data type of pointer to a function.

Example 64

```
pRam=hmCreateModel ("ram_model", ram_function);
```

In [Example 64](#), the pointer `pRam` is created to link the reference between a functional model `ram_model` and the C function `ram_function()`.

hmSetModelAttr

```
void hmSetModelAttr(hdl, attr, val)  
pHMMODEL hdl;  
HMModelAttr attr;  
int val;
```

HMNOTVOUT

`hmSetModelAttr` defines a value `val` for the attribute `attr` on the `hdl` functional model. Only the `HMNOTVOUT` attribute is currently used. If its value is set to a non-zero number, the `hdl` functional model is partitioned into a block during simulation. This helps to speed up the simulation however, only functional

models with a large number of I/O ports should set this attribute. It is not advisable to define this attribute for all functional models. Furthermore, hmSetModelAttr should be used immediately after a functional model is created; before port creation. In [Example 65](#), only the pRam functional model needs be specified.

Example 65

```
hsimC_Model()
{
    pHMMODEL pAdder, pRam;
    pAdder=hmCreateModel ("8-bit_adder", adder8);
    hmDefAnalogPort (pAdder, "C_IN", hmInput);
    hmDefAnalogPortBus (pAdder, "Ain", 0, 7, hmInput);
    hmDefAnalogPortBus (pAdder, "Bin", 0, 7, hmInput);
    hmDefDigitalPortBus (pAdder, "Sum", 0, 7, hmOutput);
    hmDefDigitalPort (pAdder, "C_OUT", hmOutput);
    hmDefAnalogPort (pAdder, "dummy", hmOutput);
    pRam=hmCreateModel ("sram_512K", ram_func);
    hmSetModelAttr(pRam, HMNOTVOUT,1);
    hmDefAnalogPortBus (pRam, "ADDR", 256, 0, hmInput);
    hmDefAnalogPortBus (pRam, "DATA", 0, 1024, hmBiput);
    hmDefAnalogStateVec (pRam, "CORE", 0, 4095);
    hmDefAnalogPort (pRam, "dummy", hmOutput);
}
```

Note: HMNOTVOUT can only be applied to models with ports consisting of non-time varying output, such as voltage input ports, disabled ports, or constant ports.

Model Port APIs

This section lists the model port APIs:

- [hmDefAnalogPort](#)
- [hmDefAnalogPortBus](#)
- [hmDefDigitalPort](#)
- [hmDefDigitalPortBus](#)
- [hmDefVarAnalogPortBus](#)
- [hmDefVarDigitalPortBus](#)
- [hmDefCurrentPort](#)

hmDefAnalogPort

```
int hmDefAnalogPort (model_pointer, port_name, port_dir)
PHMMODEL model_pointer;
char *port_name;
HMPortDir port_dir;
```

hmDefAnalogPort defines an analog port terminal in a functional model referenced by the pointer model_pointer which was created earlier by hmCreateModel. The analog port implies the signal value at the port is represented by analog voltage. The port name is given in the string port_name.

The signal direction at the port is specified by port_dir which can be one of the following: hmInput, hmOutput, and hmBiput.

Example 66

```
hmDefAnalogPort (pRam, "READ", hmInput);
hmDefAnalogPort (pRam, "WRITE", hmInput);
hmDefAnalogPort (pRam, "DATA[0]", hmBiput);
```

In Example 58, the analog ports READ and WRITE are input ports driving the functional model element referenced by pRam. The port DATA[0] can be either input or output port of the functional model element.

hmDefAnalogPortBus

```
int hmDefAnalogPortBus (model_pointer, port_bus, start, end,
port_dir)
PHMMODEL model_pointer;
char *port_bus;
int start, end;
HMPortDir port_dir;
```

hmDefAnalogPortBus defines a group of analog port terminals in a functional model referenced by the pointer model_pointer that was created earlier by hmCreateModel. The signal value at each port is represented by analog voltage. The ports are specified by a bus representation port_bus[start:end]. The name of the port bus is specified by the string port_bus, start and end are integers representing the starting bit number and the ending bit number of the bus, respectively.

The signal direction at the port is specified by port_dir which can be one of the following: hmInput, hmOutput, and hmBiput as shown in [Example 67](#).

Example 67

```
hmDefAnalogPortBus (pRam, "DATA", 0, 15, hmBiput);
Each analog port DATA[i], i=0, 1, ..., 15, in the analog port bus
DATA[0:15] is defined as bidirectional.
```

hmDefDigitalPort

```
int hmDefDigitalPort (model_pointer, port_name, port_dir)
pHMMODEL model_pointer;
char *port_name;
HMPortDir port_dir;
```

hmDefDigitalPort defines a digital port terminal in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The digital port implies the signal value at the port is represented by digital logic state. The port name is given in the string port_name. The signal direction at the port is specified by port_dir, which can be one of the following: hmInput, hmOutput, and hmBiput as shown in [Example 68](#).

Example 68

```
hmDefDigitalPort (pAdder, "C_OUT", hmOutput);
```

The digital port C_OUT is defined as the output port of an adder referenced by the pointer pAdder.

hmDefDigitalPortBus

```
int hmDefDigitalPortBus (model_pointer, port_bus, start, end,
port_dir)
pHMMODEL model_pointer;
char *port_bus;
int start, end;
HMPortDir port_dir;
```

hmDefDigitalPortBus defines a group of digital port terminals in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The signal value at each port is represented by digital logic state. The ports are specified by a bus representation port_bus[start:end] where the name of the port is specified by the string port_bus, start and end are integers representing the starting bit number and the ending bit number of the bus.

The signal direction at the port is specified by port_dir, which can be one of the following: hmInput, hmOutput, and hmBiput as shown in [Example 69](#).

Chapter 23: C Language Functional Model

Model Definition

Example 69

```
hmDefDigitalPortBus (pAdder, "SUM", 0, 7, hmOutput);
```

Each digital port SUM[i], i=0, 1, ..., 7, in the digital bus port SUM[0:7] of an adder, referenced by pAdder, is defined to be an output port.

hmDefVarAnalogPortBus

```
int hmDefVarAnalogPortBus (model_pointer, port_bus, port_dir,  
size_param_name)  
pHMMODEL model_pointer;  
char *port_bus, *size_param_name;  
HMPortDir port_dir;
```

hmDefVarAnalogPortBus is similar to hmDefAnalogPortBus, except the port bus size defined by hmDefVarAnalogPortBus is not specified when the model function is written. Instead, the size is determined after the circuit netlist is compiled: the bus size is set to the value specified in the instantiation call to the functional model. This allows the same functional model to be instantiated by modules with different bus sizes.

For example, generic RAM model can be referenced by 16-bit address modules and 32-bit address modules. The address bus size is not specified. If the address port addr is defined as a variable bus using hmDefVarAnalogPortBus, the string size_param_name defines the port bus size and the parameter name. The parameter name must be specified in the instantiation call.

```
hmDefVarAnalogPortBus (pRam, "ADDRESS", hmInput, "addr_size");
```

Each analog port ADDRESS[i], i=0, 1, ..., m is defined to be an input port. The bus size m+1 is determined by the instantiation call. An example follows:

```
Xram1 ..... ADDR0 ADDR1 ADDR2 ADDR3 ADDR4 ADDR5 ADDR6 ADDR7 ....  
+ ..... addr_size=8
```

The analog port bus ADDRESS is defined to have a size of 8 through addr_size value in the instantiation call.

hmDefVarDigitalPortBus

```
int hmDefVarDigitalPortBus (model_pointer, port_bus, port_dir,  
size_param_name)  
pHMMODEL model_pointer;  
char *port_bus, *size_param_name;  
HMPortDir port_dir;
```

`hmDefVarDigitalPortBus` is similar to `hmDefDigitalPortBus` except the port bus size defined by `hmDefVarDigitalPortBus` is not specified when the model function is written. The size is determined after the circuit netlist is compiled. The port bus size is set to the value specified in the instantiation call to the functional model. This allows the same functional model to be instantiated by modules with different bus sizes as shown in the following two examples.

Examples

In a generic RAM model the address bus size is not specified. If the address port “addr” is defined as a variable bus using `hmDefVarDigitalPortBus`, the RAM model can be referenced by 16-bit address modules and also by 32-bit address modules. The string `size_param_name` defines the parameter name which defines the port bus size. The parameter name must be specified in the instantiation call.

```
hmDefVarDigitalPortBus (pRam, "ADDRESS", hmInput, "addr_size");
```

Each digital port `ADDRESS[i]`, $i=0, 1, \dots, m$, is defined to be an input port. The bus size $m+1$ is determined by the instantiation call. An example follows.

```
Xram1 ..... ADDR0 ADDR1 ADDR2 ADDR3 ADDR4 ADDR5 ADDR6 ADDR7 ...
+ ..... addr_size=8
```

This defines the digital port bus `ADDRESS` to have a size of 8 through `addr_size` value in the instantiation call.

hmDefCurrentPort

```
int hmDefCurrentPort (port_pointer, port_name, port_dir)
pHMMODEL port_pointer;
char *port_name;
HMPortDir port_dir;
```

`hmDefCurrentPort` defines a current port in a functional model referenced by the pointer `port_pointer`, which was created earlier by `hmCreateModel`. The port name is given in the string `port_name`. The signal direction at the port is specified by `port_dir`, which should be `hmOutput` that defines the port to be an output port for the functional model. To disable the Current Port, set the current value to 0 (zero). Current is measured in amperes. Refer to [Example 70](#).

Example 70

```
hmDefCurrentPort (pPort, "OUT", hmOutput);
```

The current port OUT is driven by the functional model element referenced by pPort.

Internal State APIs

This section lists the internal state APIs:

- [hmDefAnalogState](#)
- [hmDefAnalogStateVec](#)
- [hmDefDigitalState](#)
- [hmDefDigitalStateVec](#)

hmDefAnalogState

```
int hmDefAnalogState (model_pointer, state_name)
pHMMODEL model_pointer;
char *state_name;
```

hmDefAnalogState defines an analog state variable in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The name of the state variable is specified by the string state_name.

hmDefAnalogStateVec

```
int hmDefAnalogStateVec (model_pointer, state_vec, start, end)
pHMMODEL model_pointer;
char *state_vec;
int start, end;
```

hmDefAnalogStateVec defines a vector of analog state variables in a functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel. The state vector is represented by state_vec[start:end], where start is the starting bit of the vector and end is the ending bit of the vector.

hmDefDigitalState

```
int hmDefDigitalState (model_pointer, state_name)
pHMMODEL model_pointer;
char *state_name;
```

`hmDefDigitalState` defines a digital state variable in a functional model referenced by the pointer `model_pointer`, which was created earlier by `hmCreateModel`. The name of the state variable is specified by the string `state_name`.

hmDefDigitalStateVec

```
int hmDefDigitalStateVec (model_pointer, state_vec, start, end)
pHMMODEL model_pointer;
char *state_vec;
int start, end;
```

`hmDefDigitalStateVec` defines a vector of digital state variables in a functional model referenced by the pointer `model_pointer`, which was created earlier by `hmCreateModel`. The state vector is represented by `state_vec[start:end]` in which `start` is the starting bit of the vector and `end` is the ending bit of the vector. Refer to [Example 71](#).

Example 71 Model Definition Example.

```
hsimC_Model()
{
    pHMMODEL          pAdder, pRam;
    pAdder=hmCreateModel ("8-bit_adder", adder8);
    hmDefAnalogPort (pAdder, "C_IN", hmInput);
    hmDefAnalogPortBus (pAdder, "Ain", 0, 7, hmInput);
    hmDefAnalogPortBus (pAdder, "Bin", 0, 7, hmInput);
    hmDefDigitalPortBus (pAdder, "Sum", 0, 7, hmOutput);
    hmDefDigitalPort (pAdder, "C_OUT", hmOutput);
    hmDefAnalogPort (pAdder, "dummy", hmOutput);

    pRam=hmCreateModel ("sram_512K", ram_func);
    hmDefAnalogPortBus (pRam, "ADDR", 9, 0, hmInput);
    hmDefAnalogPortBus (pRam, "DATA", 0, 15, hmBiput);
    hmDefAnalogStateVec (pRam, "CORE", 0, 4095);
    hmDefAnalogPort (pRam, "dummy", hmOutput);
}

void adder8 ()
{
-----
}
void ram_func ()
{
-----
}
```

Model Interfaces

This section describes the model interfaces:

- [Simulation Interface APIs](#)
 - [Name Interface APIs](#)
 - [Analog Port Interface APIs](#)
 - [Digital Port Interface APIs](#)
 - [Current Port APIs](#)
 - [Port Capacitance APIs](#)
 - [Enable/Disable Port APIs](#)
 - [Set Port APIs](#)
 - [Port Delay/Strength APIs](#)
 - [Port Sensitivity APIs](#)
 - [Port Change APIs](#)
 - [Port Bus Size APIs](#)
 - [Internal State Interface APIs: Analog State APIs](#)
 - [Miscellaneous Interface APIs](#)
-

Simulation Interface APIs

- [hmSimStage](#)
- [hmPresentTime](#)
- [hmWakeUpModel](#)
- [hmQuitSim](#)
- [hmIntrSim](#)

hmSimStage

HMPHASE hmSimStage ()

`hmSimStage` returns a value with data type of `HMPHASE` which, depending on the HSIM simulation phase at which `hmSimStage` is called, can be one of the following:

- `HMSTART`: The HSIM simulation is in the pre-processing stage which covers the phases from parsing the netlist until the beginning of DC initialization.
- `HMDCINIT`: The HSIM simulation is in DC initialization stage.
- `HMSIM`: The HSIM simulation is in transient simulation stage.
- `HMEND`: HSIM will wake up every model instance at the end of simulation with an `HMEND` simulation phase.

It is recommended to call those API functions during the `HSIMSTART` stage if the API functions are needed either only once or at the beginning of simulation. Examples include:

- Memory allocation
- Static variable initialization

hmPresentTime

```
double hmPresentTime ()
```

`hmPresentTime` returns the present time of simulation. The time is represented by a double precision value in the unit of second.

hmWakeUpModel

```
int hmWakeUpModel (wake_up_time)
double    wake_up_time;
```

`hmWakeUpModel` forces an evaluation of the functional model at the specified time `wake_up_time`, which has the unit of second.

hmQuitSim

```
void hmQuitSim()
```

`hmQuitSim` interprets the simulation and enters the interactive mode after the current simulation step.

hmIntrSim

```
void hmIntrSim()
```

hmIntrSim interprets the simulation and enters the interactive mode after the current simulation step.

Name Interface APIs

This section lists the name interface APIs:

- [hmPortName2PortId](#)
- [hmPortId2PortName](#)
- [hmStateName2StatId](#)
- [hmStatId2StateName](#)
- [hmPortName2CktNodeName](#)
- [hmPortId2CktNodeName](#)
- [hmModelInstName](#)
- [hmModelFuncName](#)

hmPortName2PortId

```
int hmPortName2PortId (port_name)  
char *port_name;
```

hmPortName2PortId returns an integer number which represents the ID number of the port specified by the string port_name. Using the ID number instead of the port name is recommended. This eliminates the time to search for the port index which is required when the port name is specified in an API function.

hmPortId2PortName

```
char *hmPortId2PortName (port_id)  
int port_id;
```

hmPortId2PortName returns a pointer to the string that represents the port name with the port ID number port_id.

Note: The content of the pointer must not be modified, and the pointer must not be freed.

hmStateName2StateId

```
int hmStateName2StateId (state_name)
char *state_name;
```

`hmStateName2StateId` returns an integer number that represents the ID number of the internal state variable name specified by the string `state_name`. Using the ID number instead of the port name is recommended. This eliminates the time needed to search for the state index that is required when the state name is specified in the API function.

hmStateId2StateName

```
char *hmStateId2StateName (state_id)
int state_id;
```

`hmStateId2StateName` returns a pointer to the string which represents the state variable name with the state ID number `state_id`.

Note: The content of the pointer must not be modified, and the pointer must not be freed.

hmPortName2CktNodeName

```
char *hmPortName2CktNodeName (port_name)
char *port_name;
```

`hmPortName2CktNodeName` returns the full hierarchy path name of the circuit node that connects to the port specified by the string `port_name`.

Note: The string returned by `hmPortName2CktNodeName` is from a temporary buffer. The content of the pointer should not be modified, and the pointer should not be freed. However, the contents of the pointer can be changed by future API calls.

hmPortId2CktNodeName

```
char *hmPortId2CktNodeName (port_id)
int *port_id;
```

Chapter 23: C Language Functional Model

Model Interfaces

`hmPortId2CktNodeName` returns the full hierarchy path name of the circuit node that connects to the port specified by the ID number `port_id`.

Note: The string returned by `hmPortId2CktNodeName` is from a temporary buffer. The content of the pointer should not be modified, and the pointer should not be freed. However, the contents of the pointer can be changed by future API calls.

hmModelInstName

```
char *hmModelInstName ()
```

`hmModelInstName` returns a pointer to the string which represents the full hierarchy path name of the functional model instance.

Note: The string returned by `hmModelInstName` is from a temporary buffer. The content of the pointer should not be modified, and the pointer should not be freed. However, the contents of the pointer can be changed by future API calls. Refer to [Example 72](#).

Example 72

```
hsimC_Model()
{
    pHMMODEL          pAdder, pRam;
    pAdder=hmCreateModel ("8-bit_adder", adder8);
    .....
}

void adder8 ()
{
    char           *model_name, *inst_name, *node1, *node2;
    .....
    model_name=hmModelFuncName ();
    /* model_name is "8-bit_adder"      */
    inst_name=hmModelInstName ();
    /* inst_name is Xalu.Xadd3         */
    node1=hmPortName2CktNodeName ("SUM");
    /* the node which connects to SUM */
    node2=hmPortName2CktNodeName ("C_OUT");
    /* the node which connects to C_OUT */
    .....
}
```

Note: The model_name will be 8-bit_adder when hmModelName is called from the model function adder8(). The instance name, however, could change in different occasions depending on which instance of 8-bit_adder invokes the function call of adder8().

hmModelFuncName

```
char *hmModelFuncName ()
```

hmModelFuncName returns a pointer to the string that represents the functional model name.

Note: The content of the pointer must not be modified, and the pointer must not be freed.

Analog Port Interface APIs

- [hmAnalogPortValue](#)
- [hmAnalogPortValueById](#)
- [hmAnalogPortBusValue](#)
- [hmAnalogPortBusValueById](#)
- [hmSetAnalogPortValue](#)
- [hmSetAnalogPortValueById](#)
- [hmSetAnalogPortBusValue](#)
- [hmSetAnalogPortBusValueById](#)

hmAnalogPortValue

```
double hmAnalogPortValue (port_name)  
char *port_name;
```

hmAnalogPortValue returns the voltage value of the analog port specified by the string port_name. The result is represented by a double precision value and has the unit of volt.

hmAnalogPortValue("dummy") returns the port voltage of dummy in the model adder8 if it is called within the function adder8(). hmAnalogPortValue("dummy")

returns the port voltage of dummy in the model ram_func if it is called from the function ram_func().

hmAnalogPortValueById

```
double hmAnalogPortValueById (port_id)
int port_id;
```

hmAnalogPortValueById returns the voltage value of the analog port specified by the ID number port_id. The result is represented by a double precision value and has the unit of volt.

hmAnalogPortBusValue

```
double *hmAnalogPortBusValue (port_bus, start, end)
char *port_bus;
int start, end;
```

hmAnalogPortBusValue returns a pointer to an array of port voltages for a group of analog ports represented by the bus port_bus[start:end], where start is the starting bit of the bus, and end is the ending bit of the bus. Each voltage value is represented by a double precision value and has the unit of volt. Since the array space is allocated within the function hmAnalogPortBusValue, it is required to free the array within the calling function when the array is no longer needed or before exiting the calling function. Refer to [Example 73](#).

Example 73

```
test_func ()
{
    double      *a, *b;

    a=hmAnalogPortBusValue ("addr", 4, 7);
    /* a[0]=addr[4], a[1]=addr[5], a[2]=addr[6],
       a[3]=addr[7] */

    b=hmAnalogPortBusValue ("data", 7, 0);
    /* b[0]=data[7], b[1]=data[6], ....,
       b[6]=data[1], b[7]=data[0] */

    .....
    free (a);
    free (b);
}
```

hmAnalogPortBusValueById

```
double *hmAnalogPortBusValueById (port_bus_id, start, end)
int port_bus_id;
int start, end;
```

hmAnalogPortBusValueById returns a pointer to an array of port voltages for a group of analog ports represented by the bus port_bus[start:end], where the port bus name is specified by the ID number port_bus_id, start is the starting bit of the bus, and end is the ending bit of the bus. Each voltage value is represented by a double precision value and has the unit of volt. Since the array space is allocated within the function hmAnalogPortBusValueById, it is required to free the array within the calling function when the array is no longer needed, or before exiting the calling function.

hmSetAnalogPortValue

```
int hmSetAnalogPortValue (port_name, voltage)
char *port_name;
double voltage;
```

hmSetAnalogPortValue sets the voltage of analog port, specified by the string port_name, to the value specified by voltage. voltage is a double precision number and has the unit of Volt.

hmSetAnalogPortValueById

```
int hmSetAnalogPortValueById (port_id, voltage)
int port_id
double voltage;
```

hmSetAnalogPortValueById sets the voltage of analog port, specified by the ID number port_id, to be the value specified by voltage, which is a double precision number and has the unit of Volt.

hmSetAnalogPortBusValue

```
int hmSetAnalogPortBusValue (port_bus, start, end, voltage_array)
char *port_bus;
int start, end;
double *voltage_array;
```

hmSetAnalogPortBusValue sets the voltages of analog port bus, specified by port_bus[start:end], to be the array values voltage_array[0:abs(start-end)], where abs(x) represents the absolute value of x, and voltage_array is a pointer

to a double array. Refer to [Example 74](#).

Example 74

```
write_state ()  
{  
    double          x[16];  
    .....  
    hmSetAnalogPortBusValue ("DATA", 15, 0. x);  
    /* DATA[15]=x[0], DATA[14]=x[1], ..... DATA[0]=x[15] */  
    .....  
}
```

hmSetAnalogPortBusValueById

```
int hmSetAnalogPortBusValueById (port_bus_id, start, end,  
voltage_array)  
int      port_bus_id;  
int      start, end;  
double   *voltage_array;
```

`hmSetAnalogPortBusValueById` sets the voltages of analog port bus, specified by `port_bus_id[start:end]`, to be the array values `voltage_array[0:abs(start-end)]`, where `abs(x)` represents the absolute value of `x`, the port bus name is specified by its ID number `port_bus_id`, and `voltage_array` is a pointer to a double array.

Digital Port Interface APIs

This section lists the digital port interface APIs:

- [hmDigitalPortValue](#)
- [hmDigitalPortValueById](#)
- [hmSetDigitalPortBusDelay](#)
- [hmSetDigitalPortBusDelayById](#)
- [hmDigitalPortBusValue](#)
- [hmDigitalPortBusValueById](#)
- [hmSetDigitalPortDelay](#)
- [hmSetDigitalPortDelayById](#)
- [hmSetDigitalPortValue](#)

- [hmSetDigitalPortValueById](#)
- [hmSetDigitalPortBusValue](#)
- [hmSetDigitalPortBusValueById](#)

hmDigitalPortValue

```
HMLogic hmDigitalPortValue (port_name)
char *port_name;
```

hmDigitalPortValue returns the digital state of the digital port specified by the string port_name. The state value has the data type of hmLogic.

hmDigitalPortValueById

```
HMLogic hmDigitalPortValueById (port_id)
Int port_id;
```

hmDigitalPortValueById returns the digital state of the digital port specified by the ID number port_id. The state value has the data type of hmLogic.

hmSetDigitalPortBusDelay

```
hmSetDigitalPortBusDelay (port_bus, start, end,
rising_delay_time_array, falling_delay_time_array)
char *port_bus;
int start, end;
double *rising_delay_time_array;
double *falling_delay_time_array;
```

hmSetDigitalPortBusDelay sets the intrinsic rising and falling delay times on the digital port bus specified by port_bus[start:end]. Delay times should be double precision arrays corresponding to port_bus[start:end] and in the unit of second.

hmSetDigitalPortBusDelayById

```
hmSetDigitalPortBusDelayById(port_bus_id, start, end,
rising_delay_time_array, falling_delay_time_array)
int port_bus_id;
int start, end;
double *rising_delay_time_array;
double *falling_delay_time_array;
```

hmSetDigitalPortBusDelayById sets the intrinsic rising and falling delay times on the digital port bus specified by port_bus_id[start:end]. Delay times should be double precision arrays corresponding to port_bus_id[start:end] and in the unit of second.

hmDigitalPortBusValue

```
HMLogic *hmDigitalPortBusValue (port_bus, start, end)
char *port_bus;
int start, end;
```

hmDigitalPortBusValue returns a pointer to an array of port states for a group of digital ports represented by the bus port_bus[start:end], where start is the starting bit of the bus, and end is the ending bit of the bus. Each state value has the data type of hmLogic.

Since the array space is allocated within the function hmDigitalPortBusValue, it is required to free the array within the calling function when the array is no longer needed, or before exiting from the calling function.

hmDigitalPortBusValueById

```
HMLogic *hmDigitalPortBusValueById (port_bus_id, start, end)
int port_bus_id;
int start, end;
```

hmDigitalPortBusValueById returns a pointer to an array of port states for a group of digital ports represented by the bus port_bus[start:end], where the port bus is specified by the ID number port_bus_id, start is the starting bit of the bus, and end is the ending bit of the bus. Each state value has the data type of hmLogic.

Since the array space is allocated within the function hmDigitalPortBusValue, it is required to free the array within the calling function when the array is no longer needed, or before exiting from the calling function.

hmSetDigitalPortDelay

```
hmSetDigitalPortDelay(port_name, rising_dalay_time,
falling_delay_time)
    char *port_name;
    double rising_dalay_time;
    double falling_delay_time;
```

hmSetDigitalPortDelay sets the intrinsic rising and falling delay times on the digital port specified by port_name. Both delay times should be double precision numbers and in the unit of second. It should be noted that these delay times are in addition to the RC delays on the port.

hmSetDigitalPortDelayById

```
SetDigitalPortDelayById(port_id, rising_dalay_time,  
falling_delay_time)  
int port_id;  
double rising_dalay_time;  
double falling_delay_time;
```

SetDigitalPortDelayById sets the intrinsic rising and falling delay times on the digital port specified by port_id. Both delay times should be double precision numbers and in the unit of second.

hmSetDigitalPortValue

```
int hmSetDigitalPortValue (port_name, state)  
char *port_name;  
HMLogic state;
```

hmSetDigitalPortValue sets the logic state of digital port, specified by the string port_name, to be the value specified by state. The state value has the data type of hmLogic.

hmSetDigitalPortValueById

```
int hmSetDigitalPortValueById (port_id, state)  
int port_id;  
HMLogic state;
```

hmSetDigitalPortValueById sets the logic state of digital port, specified by the ID number port_id, to be the value specified by state. The state value has the data type of hmLogic.

hmSetDigitalPortBusValue

```
int hmSetDigitalPortBusValue (port_bus, start, end, state_array)  
cha *port_bus;  
int start, end;  
HMLogic *state_array;
```

hmSetDigitalPortBusValue sets the logic states of digital port bus, specified by port_bus[start:end], to be the array values specified by the array state_array[0:abs(start-end)], where abs(x) represents the absolute value of x and state_array is a pointer to a hmLogic array. The state value has the data type of hmLogic.

hmSetDigitalPortBusValueById

```
int hmSetDigitalPortBusValueById (port_bus_id, start, end,
state_array)
int      port_bus_id;
int      start, end;
HMLogic *state_array;
```

hmSetDigitalPortBusValueById sets the logic states of digital port bus, specified by port_bus[start:end], to be the array values specified by the array state_array[0:abs(start-end)], where the port bus name is specified by its ID number port_bus_id, abs(x) represents the absolute value of x and state_array is a pointer to a hmLogic array. The state value has the data type of hmLogic.

Current Port APIs

This section lists the current port APIs:

- [hmCurrentPortValue](#)
- [hmCurrentPortValueById](#)
- [hmSetCurrentPortValue](#)
- [hmSetCurrentPortValueById](#)

hmCurrentPortValue

```
double hmCurrentPortValue (port_name)
char *port_name;
```

hmCurrentPortValue returns the current value of the current port specified by the string port_name. It is intended to work with a port that is defined as an Analog Port. If the port direction of the Analog Port is Input or Disabled, the returned value is 0. If the port is defined as Current Port and the function is used to obtain the current value, the most recent value of the current set by the

function hmSetCurrentPortValue is returned. The current value is represented by a double precision value and has the unit of ampere.

hmCurrentPortValueById

```
double hmCurrentPortValueById (port_id)
int port_id;
```

hmCurrentPortValueById returns the current value of the current port specified by the ID number port_id. The usage is similar to [hmCurrentPortValue on page 672](#) except for port_id.

hmSetCurrentPortValue

```
int hmSetCurrentPortValue (port_name, current)
char *port_name;
double current;
```

hmSetCurrentPortValue sets the electrical current value of current port, specified by the string port_name, to be the value specified by current. Current flow is a double precision number measured in Amperes as shown in [Table on page 673](#).

Current Flow Direction

Table 100

Current Flow	Flow Direction
POSITIVE value	OUT of the functional model port
NEGATIVE value	INTO the functional model port

hmSetCurrentPortValueById

```
int hmSetCurrentPortValueById (port_id, current)
int port_id
double current;
```

hmSetCurrentPortValueById sets the current of current port, specified by the ID number port_id, to be the value specified by current. Current direction is shown in [Table on page 673](#).

Port Capacitance APIs

This section describes the port capacitance APIs:

- [hmPortCap](#)
- [hmPortCapById](#)
- [hmPortBusCap](#)
- [hmPortBusCapById](#)
- [hmSetPortCap](#)
- [hmSetPortCapById](#)
- [hmSetPortBusCap](#)
- [hmSetPortBusCapById](#)

hmPortCap

```
double hmPortCap (port_name)
char *port_name;
```

hmPortCap returns node capacitance value of the port specified by the string port_name. The capacitance value is in double precision and has the unit of Farad.

hmPortCapById

```
double hmPortCapById (port_id)
int      port_id;
```

hmPortCapById returns node capacitance value of the port specified by the ID number port_id. The capacitance value is in double precision and has the unit of Farad.

hmPortBusCap

```
double *hmPortBusCap (port_bus, start, end)
char    *port_bus;
int      start, end;
```

hmPortBusCap returns a pointer to a double array which stores the node capacitances of port_bus[start:end], where the name of the port bus is specified by the string port_bus, start is the starting bit of the bus, and end is

the ending bit of the bus. The capacitance value is in double precision and has the unit of Farad.

hmPortBusCapById

```
double *hmPortBusCapById (port_bus_id, start, end)
int    port_bus_id;
int    start, end;
```

hmPortBusCapById returns a pointer to a double array which stores the node capacitances of port_bus[start:end], where the name of the port bus is specified by its ID number port_bus_id, start is the starting bit of the bus, and end is the ending bit of the bus. The capacitance value is in double precision and has the unit of Farad.

hmSetPortCap

```
int hmSetPortCap (port_name, cap)
char *port_name;
double cap;
```

hmSetPortCap sets the node capacitance of the port, specified by the string port_name, to be the value cap. The capacitance value is in double precision and has the unit of Farad.

hmSetPortCapById

```
int hmSetPortCapById (port_id, cap)
int    port_id;
double cap;
```

hmSetPortCapById sets the node capacitance of the port, specified by the ID number port_id, to be the value cap. The capacitance value is in double precision and has the unit of Farad.

hmSetPortBusCap

```
int hmSetPortBusCap (port_bus, start, end, cap_array)
char *port_bus;
int    start, end;
double *cap_array;
```

hmSetPortBusCap sets the node capacitance of each port in the bus port_bus[start:end] to be the value corresponding to the capacitor array element, where the name of the bus port is specified by the string port_bus,

start is the starting bit of the bus and end is the ending bit of the bus. The pointer to the capacitor array is given by cap_array. The capacitance value is in double precision and has the unit of Farad.

hmSetPortBusCapById

```
int hmSetPortBusCapById (port_bus_id, start, end, cap_array)
int      port_bus_id;
int      start, end;
double   *cap_array;
```

hmSetPortBusCapById sets the node capacitance of each port in the bus port_bus[start:end] to be the value corresponding to the capacitor array element, where the name of the bus port is specified by its ID number port_bus_id, start is the starting bit of the bus and end is the ending bit of the bus. The pointer to the capacitor array is given by cap_array. The capacitance value is in double precision and has the unit of Farad. Refer to [Example 75](#).

Example 75 Port Capacitance Usage Example

```
define_capacitance ()
{
    double   *a, b[16];

    .....
    a=hmPortBusCap ("data", 7, 0);
    /* a[0]=capacitance at data[7]
     * a[1]=capacitance at data[6]
     * .....
     * a[7]=capacitance at data[0] */

    hmSetPortBusCap ("addr", 11, 8, b);
    /* capacitance at addr[11]=b[0]
     * capacitance at addr[10]=b[1]
     * capacitance at addr[9]=b[2]
     * capacitance at addr[8]=b[3] */

    .....
    free(a);
}
```

Enable/Disable Port APIs

This section lists the enable/disable port APIs:

- [hmDisablePortDrive](#)
- [hmDisablePortDriveById](#)
- [hmEnablePortDrive](#)
- [hmEnablePortBusDriveById](#)
- [hmDisablePortBusDrive](#)
- [hmDisablePortBusDriveById](#)
- [hmEnablePortBusDrive](#)
- [hmEnablePortBusDriveById](#)
- [hmDisableAllPorts](#)
- [hmEnableAllPorts](#)

hmDisablePortDrive

```
int hmDisablePortDrive (port_name)
char *port_name;
```

`hmDisablePortDrive` disables the output driver of a bidirectional or output port in a functional model. The port name is specified by the string `port_name`. After the output driver is disabled, the port direction becomes `hmHiZ` if the port is an output port; the port direction becomes `hmInput` if the port is a bidirectional port.

hmDisablePortDriveById

```
int hmDisablePortDriveById (port_id)
int port_id;
```

`hmDisablePortDriveById` disables the output driver of a bidirectional or output port in a functional model. The port ID number is specified by the integer `port_id`. After the output driver is disabled, the port direction becomes `hmHiZ` if the port is an output port; the port direction becomes `hmInput` if the port is a bidirectional port.

hmEnablePortDrive

```
int hmEnablePortDrive (port_name)
char *port_name;
```

hmEnablePortDrive enables the output driver to a bidirectional or output port in a functional model. The port name is specified by the string port_name.

hmEnablePortDriveById

```
int hmEnablePortDriveById (port_id)
int port_id;
```

hmEnablePortDriveById enables the output driver to a bidirectional or output port in a functional model. The port ID number is specified by the integer port_id.

hmDisablePortBusDrive

```
int hmDisablePortBusDrive (port_bus, start, end)
char *port_bus;
int start, end;
```

hmDisablePortBusDrive disables the output driver to each output or bidirectional port in the bus port_bus[start:end].

hmDisablePortBusDriveById

```
int hmDisablePortBusDriveById (port_bus_id, start, end)
int port_bus_id;
int start, end;
```

hmDisablePortBusDriveById disables the output driver to each output or bidirectional port in the bus port_name[start:end], where the ID number for port_name is port_bus_id.

hmEnablePortBusDrive

```
int hmEnablePortBusDrive (port_bus, start, end)
char *port_bus;
int start, end;
```

hmEnablePortBusDrive enables the output driver to each output or bidirectional port in the bus port_bus[start:end].

hmEnablePortBusDriveById

```
int hmEnablePortBusDriveById (port_bus_id, start, end)
int port_bus_id;
int start, end;
```

hmEnablePortBusDriveById enables the output driver to each output or bidirectional port in the bus port_name[start:end], where the ID number for port_name is port_bus_id.

hmDisableAllPorts

```
int hmDisableAllPorts (port_dir)
HMPortDir port_dir;
```

hmDisableAllPorts disables the output driver to each model port which has the port direction port_dir. The direction must be either hmOutput or hmBiput.

hmEnableAllPorts

```
int hmEnableAllPorts (port_dir)
HMPortDir port_dir;
```

hmEnableAllPorts enables the output driver to each model port which has the port direction port_dir. The direction must be either hmOutput or hmBiput.

Set Port APIs

The following APIs enable model ports to be set to active or idle. These APIs include:

- [hmSetPortActive](#)
- [hmSetPortActiveByID](#)
- [hmSetPortBusActive](#)
- [hmSetPortBusActiveByID](#)
- [hmSetPortIdle](#)
- [hmSetPortIdleByID](#)
- [hmSetPortBusIdle](#)
- [hmSetPortBusIdleByID](#)

hmSetPortActive

```
hmSetPortActive (port_name)
char *port_name;
```

hmSetPortActive enables the model port so that the model will be evaluated when the port changes.

hmSetPortActiveByID

```
hmSetPortActiveByID (port_id)
char *port_id;
```

hmSetPortActiveByID enables a model port specified with an ID so that the model will be evaluated when the port changes.

hmSetPortBusActive

```
hmSetPortBusActive (port_name, start, end)
char *port_name;
int start, end;
```

hmSetPortBusActive enables the model port bus so that the model will be evaluated when the port changes.

hmSetPortBusActiveByID

```
hmSetPortBusActiveByID (port_id, start, end)
char *port_id;
int start, end;
```

hmSetPortBusActiveByID enables a model port bus specified with an ID so that the model will be evaluated when the port changes.

hmSetPortIdle

```
hmSetPortIdle (port_name)
int port_name;
```

hmSetPortIdle disables the model port so that the model will not be evaluated when the port changes.

hmSetPortIdleByID

```
hmSetPortIdleByID (port_id)
int port_id;
```

`hmSetPortIdleByID` disables a model port specified with an ID so that the model will not be evaluated when the port changes.

hmSetPortBusIdle

```
hmSetPortBusIdle (port_bus_name, start, end)
int port_bus_name;
int start, end;
```

`hmSetPortBusIdle` disables the model port bus so that the model will not be evaluated when the port changes.

hmSetPortBusIdleByID

```
hmSetPortBusIdleByID (port_bus_id, start, end)
int port_bus_id;
int start, end;
```

`hmSetPortBusIdleByID` disables a model port bus specified with an ID so that the model will not be evaluated when the port changes.

Port Delay/Strength APIs

The following four functions specify the delay time (τ) for a signal outputted by a port or ports given by one of the following:

- Name
- ID
- Bus name
- Bus ID

Implementing the delay is accomplished using an RC circuit in which $R*C=\tau$. In addition to τ , the value of R (default is 1 Ohm) can also be specified so that the value of capacitance will be τ/R . For this implementation to be accurate, R and C should be chosen as follows:

- R should be much smaller than the loading impedance of the port.
- C should be much larger than the loading capacitance of the port.

hmSetPortDelayRes

```
int hmSetPortDelayRes (port_name, delay, res)
char      *port_name;
double    delay, res;
```

hmSetPortDelayRes sets the delay time delay and driver impedance res to the output or bidirectional port specified by the string port_name. The delay time is given by a double precision number and has the unit of second.

The driver impedance is represented by a double precision value and has the unit of ohm. Any voltage or logic-state change at the specified port will take a delay time to react the change. The port is driven by a voltage source through a resistance of res.

hmSetPortDelayResByID

```
int hmSetPortDelayResByID (port_name, delay, res)
char      *port_name;
double    delay, res;
```

hmSetPortDelayResByID sets the delay time delay and driver impedance res to the output or bidirectional port specified by the string port_name. The delay time is given by a double precision number and has the unit of second.

The driver impedance is represented by a double precision value and has the unit of ohm. Any voltage or logic-state change at the specified port will take a delay time to react the change. The port is driven by a voltage source through a resistance of res.

hmSetPortBusDelayRes

```
int hmSetPortBusDelayRes (port_bus, start, end, delay, res)
char      *port_bus;
int       start, end;
double    delay, res;
```

hmSetPortBusDelayRes sets the delay time delay and driver impedance res to each output or bidirectional port of the bus port_bus[start:end]. The delay time is given by a double precision number and has the unit of second.

The driver impedance is represented by a double precision number and has the unit of ohm. Any voltage or logic-state change at the specified port will take a delay time to react the change. The port is driven by a voltage source through a resistance of res.

hmSetPortBusDelayResById

```
int hmSetPortBusDelayResById (port_bus_id, start, end, delay,
res)
int      port_bus_id, start, end;
double   delay, res;
```

hmSetPortBusDelayResById sets the delay time delay and driver impedance res to each output or bidirectional port of the bus port_bus[start:end], where port_bus is the name of the bus with ID number port_bus_id. The delay time is given by a double precision number and has the unit of second.

The driver impedance is represented by a double precision number and has the unit of ohm. Any voltage or logic-state change at the specified port will take a delay time to react the change. The port is driven by a voltage source through a resistance of res.

Port Sensitivity APIs

This section lists the port sensitivity APIs:

- [hmSetPortEventDv](#)
- [hmSetPortEventDvById](#)
- [hmSetPortBusEventDv](#)
- [hmSetPortBusEventDvById](#)

hmSetPortEventDv

```
int hmSetPortEventDv (port_name, event_dv)
char      *port_name;
double    event_dv;
```

hmSetPortEventDv sets the event threshold voltage event_dv to the port specified by the string port_name. The event threshold voltage defines when the functional model needs to be re-evaluated. The functional model is scheduled and evaluated when the difference between the new port voltage and the last port voltage, when the functional model was evaluated last time, exceeds the event threshold voltage event_dv.

The value of event_dv adjusts the trade-off between performance and precision: Larger event_dv value reduces the number of functional model evaluations at the cost of less precision. The default event threshold voltage is

0.3V and is suggested to be 0.1V or smaller value in case of higher precision requirements.

hmSetPortEventDvById

```
int hmSetPortEventDvById (port_id, event_dv)
int      port_id;
double   event_dv;
```

hmSetPortEventDvById sets the event threshold voltage event_dv to the port specified by the ID number of port_id. The event threshold voltage defines when the functional model needs to be re-evaluated. The functional model is scheduled and evaluated when the difference between the new port voltage and the last port voltage, when the functional model was evaluated last time, exceeds the event threshold voltage event_dv.

The value of event_dv adjusts the trade-off between performance and precision: Larger event_dv value reduces the number of functional model evaluations at the cost of less precision. The default event threshold voltage is 0.3V and is suggested to be 0.1V or smaller value in case of higher precision requirements.

hmSetPortBusEventDv

```
int hmSetPortBusEventDv (port_bus, start, end, event_dv_array)
char    *port_bus;
int      start, end;
double  *event_dv_array;
```

hmSetPortBusEventDv sets the event threshold voltage of each port in the bus port_bus[start:end] to be the corresponding value in the array element, where event_dv_array is the pointer to the array, start is the starting bit of the bus and end is the ending bit of the bus. The event threshold voltage defines when the functional model needs to be re-evaluated. The functional model is scheduled and evaluated when the difference between the new port voltage and the last port voltage, when the functional model was evaluated last time, exceeds the event threshold voltage.

The value of event threshold voltage adjusts the trade-off between performance and precision: Larger event threshold voltage reduces the number of functional model evaluations at the cost of less precision. The default event threshold voltage is 0.3V and is suggested to be 0.1V or smaller value in case of higher precision requirements.

hmSetPortBusEventDvById

```
int hmSetPortBusEventDvById (port_bus_id, start, end,
event_dv_array)
int      port_bus_id, start, end;
double   *event_dv_array;
```

hmSetPortBusEventDvById sets the event threshold voltage of each port in the bus port_bus[start:end] to be the corresponding value in the array element, where the bus is specified by its ID number port_bus_id, event_dv_array is the pointer to the array, start is the starting bit of the bus and end is the ending bit of the bus. The event threshold voltage defines when the functional model needs to be re-evaluated. The functional model is scheduled and evaluated when the difference between the new port voltage and the last port voltage, when the functional model was evaluated last time, exceeds the event threshold voltage.

The value of event threshold voltage adjusts the trade-off between performance and precision: Larger event threshold voltage reduces the number of functional model evaluations at the cost of less precision. The default event threshold voltage is 0.3V and is suggested to be 0.1V or smaller value in case of higher precision requirements. Refer to [Example 76](#).

Example 76

```
set_event_voltage ()
{
    int      i;
    double ev[16];mfs

    for (i=0; i<16; i++)
        ev[i]=(i > 7)? 0.1: 0.05;
    hmSetPortBusEventDv ("addr", 15, 0, ev);
    /* event threshold voltage at addr[15]=ev[0]
       event threshold voltage at addr[14]=ev[1]
       ..... .
       event threshold voltage at addr[0]=ev[15] */
    .....
}
```

Port Change APIs

This sections lists the port change APIs:

Chapter 23: C Language Functional Model

Model Interfaces

- [hmPortChange](#), [hmPortChangeById](#)
- [hmPortBusChange](#), [hmPortBusChangeById](#)
- [hmAnyPortChange](#)

hmPortChange, hmPortChangeById

```
int hmPortChange(port_name)
char *port_name;
int hmPortChangeById(port_id)
int port_id;
```

hmPortChange and hmPortChangeById return 1 if the digital port specified by port_name or port_id has value change since the last time this model is evaluated. This function should be used for digital ports only. For analog ports, the functions will always return 1.

hmPortBusChange, hmPortBusChangeById

```
int hmPortBusChange(port_bus, start, end)
char *port_bus;
int start;
int end;
int hmPortBusChangeById(port_bus_id, start, end)
int port_bus_id;
int start;
int end;
```

hmPortBusChange and hmPortBusChangeById returns 1 if any of the specified group of digital ports represented by the bus port_bus[start:end] has value change since the last time this model is evaluated. For analog ports, the functions will always return 1.

hmAnyPortChange

```
int hmAnyPortChange(dir)
HMPortDir dir;
```

hmAnyPortChange returns 1 if any digital port matches the port direction specified by dir has a value change since the last time this model is evaluated. dir can be hmInput or hmBiput.

Port Bus Size APIs

This section lists the port bus size APIs:

- [hmPortBusSize](#)
- [hmPortBusSizeById](#)
- [hmPortDir. hmPortDirByID](#)

hmPortBusSize

```
int hmPortBusSize (port_bus)
char *port_bus;
```

`hmPortBusSize` returns the bus size of the port specified by the string `port_size`.

If the bus is defined by `hmDefAnalogPortBus` or `hmDefDigitalPortBus`, the size is equal to `abs(start-end+1)` in which start and end are the starting and ending bit numbers of the port bus, respectively.

If the bus is defined by `hmDefVarAnalogPortBus` or `hmDefVarDigitalPortBus`, the size is equal to the parameter value specified in the instance call. The parameter name is defined by the string specified in `hmDefVarAnalogPortBus` or `hmDefVarDigitalPortBus`.

hmPortBusSizeById

```
int hmPortBusSizeById (port_id)
int port_id;
```

`hmPortBusSizeById` returns the bus size of the port specified by its ID number `port_id`.

If the bus is defined by `hmDefAnalogPortBus` or `hmDefDigitalPortBus`, the size is equal to `abs(start-end+1)` in which start and end are the starting and ending bit numbers of the port bus, respectively.

If the bus is defined by `hmDefVarAnalogPortBus` or `hmDefVarDigitalPortBus`, then the size is equal to the parameter value specified in the instance call. The parameter name is defined by the string specified in `hmDefVarAnalogPortBus` or `hmDefVarDigitalPortBus`.

hmPortDir, hmPortDirByID

```
HMPortDir hmPortDir (port_name)
char *port_name;
```

```
HMPortDir hmPortDirById(port_id)
int port_id;
```

`hmPortDir` checks and returns the signal flow direction at the port specified by the string `port_name`.

`hmPortDirById` checks and returns the signal flow direction at the port specified by the ID number `port_id`.

The direction has the data type of `hmPortDir` or `hmPortDirById` and can be one of the following:

hmInput

There are two possibilities for `hmInput`:

1. The port has been defined as `hmInput` in the function `hmDefAnalogPort` or `hmDefDigitalPort`.
2. The port has been defined as `hmBiput` and has also been disabled from the output driver by `hmDisablePortDrive`.

hmOutput

There are two possibilities for `hmOutput`:

1. The port has been defined as `hmOutput` and has also been enabled to the output driver by `hmEnablePortDrive`;
2. The port has been defined as `hmBiput` and has also been enabled to the output driver by `hmEnablePortDrive`.

hmHiZ

`hmHiZ` has been defined as `hmOutput` and the port has also been disabled from the output driver by `hmDisablePortDrive`.

Internal State Interface APIs: Analog State APIs

This section lists the internal analog state APIs:

- [hmAnalogStateValue](#)
- [hmAnalogStateValueById](#)
- [hmAnalogStateVecValue](#)
- [hmAnalogStateVecValueById](#)
- [hmSetAnalogStateValue](#)
- [hmSetAnalogStateValueById](#)
- [hmSetAnalogStateVecValue](#)
- [hmSetAnalogStateVecValueById](#)

hmAnalogStateValue

```
double hmAnalogStateValue (state_name)
char *state_name;
```

hmAnalogStateValue returns the numerical value of the internal state variable specified by the string state_name. Double-precision value is used.

hmAnalogStateValueById

```
double hmAnalogStateValueById (state_id)
int state_id;
```

hmAnalogStateValueById returns the numerical value of the internal state variable specified by the ID number state_id. Double-precision value is used.

hmAnalogStateVecValue

```
double *hmAnalogStateVecValue (state_vec, start, end)
char *state_vec;
int start, end;
```

hmAnalogStateVecValue returns a pointer to an array of internal state values for a group of analog states represented by the vector state_vec[start:end], where start is the starting bit of the vector, and end is the ending bit of the vector. A double precision value is used. Since the array space is allocated within the hmAnalogStateVecValue function, it is required to free the array within the calling function when the array is no longer needed, or before exiting the calling function.

hmAnalogStateVecValueById

```
double *hmAnalogStateVecValueById (state_vec_id, start, end)
int state_vec_id;
int start, end;
```

hmAnalogStateVecValueById returns a pointer to an array of internal state values for a group of analog states represented by the vector state_vec[start:end], where the state vector name is specified by the ID number state_vec_id, start is the starting bit of the vector, and end is the ending bit of the vector. A double precision value is used. Since the array space is allocated within the function hmAnalogStateVecValueById, it is required to free the array within the calling function when the array is no longer needed, or before exiting from the calling function.

hmSetAnalogStateValue

```
int hmSetAnalogStateValue (state_name, value)
char *state_name;
double value;
```

hmSetAnalogStateValue sets the state value of an internal state variable, specified by the string state_name, to a double precision value.

hmSetAnalogStateValueById

```
int hmSetAnalogStateValueById (state_id, value)
int state_id;
double value;
```

hmSetAnalogStateValueById sets the state value of an internal analog state variable to a double precision value. The name of the state variable is specified by its ID number state_id.

hmSetAnalogStateVecValue

```
int hmSetAnalogStateVecValue (state_vec, start, end, value_array)
char *state_vec;
int start, end;
double *value_array;
```

hmSetAnalogStateVecValue sets the values of internal analog state vector, specified by state_vec[start:end], to be the array values value_array[0:abs(start-end)], where abs(x) represents the absolute value of x, and value_array is a pointer to a double-type array.

hmSetAnalogStateVecValueById

```
int hmSetAnalogStateVecValueById (state_vec_id, start, end,
value_array)
int      state_vec_id;
int      start, end;
double   *value_array;
```

`hmSetAnalogStateVecValueById` sets the values of internal analog state vector, specified by `state_vec[start:end]`, to be the array values `value_array[0:abs(start-end)]`, where `abs(x)` represents the absolute value of `x`, and `value_array` is a pointer to a double-type array. The vector name is specified by its ID number `state_vec_id`.

Internal State Interface APIs: Digital State APIs

This section lists the internal digital state APIs:

- [hmDigitalStateValue](#)
- [hmDigitalStateValueById](#), [hmDigitalStateVecValue](#), [hmDigitalStateVecValueById](#)
- [hmSetDigitalStateValue](#)
- [hmSetDigitalStateValueById](#)
- [hmSetDigitalStateVecValue](#)
- [hmSetDigitalStateVecValueById](#)

hmDigitalStateValue

```
HMLogic hmDigitalStateValue (state_name)
char*state_name;
```

`hmDigitalStateValue` returns the digital state value of the internal state variable specified by the string `state_name`. The state value has the data type of `hmLogic`.

hmDigitalStateValueById, hmDigitalStateVecValue, hmDigitalStateVecValueById

```
HMLogic hmDigitalStateValueById (state_id)
int  state_id;
```

Chapter 23: C Language Functional Model
Model Interfaces

```
HMLogic *hmDigitalStateVecValue (state_vec, start, end)
char *state_vec;
int start, end;

HMLogic *hmDigitalStateVecValueById (state_vec_id, start, end)
char state_vec_id;
int start, end;
```

hmDigitalStateValueById returns the digital state value of the internal state variable specified by the ID number state_id.

hmDigitalStateVecValue returns a pointer to an array of internal state values for a group of digital states end is the ending bit of the vector.

hmDigitalStateVecValueById returns a pointer to an array of internal state values for a group of digital states represented by the vector state_vec[start:end]

,

where the state vector name is specified by its ID number state_vec_id, start is the starting bit of the vector, and end is the ending bit of the vector.

The state value has the data type of hmLogic and can be one of the following:

- hmZero: Represents logic 0 state.
- hmOne: Represents logic 1 state.
- hmU: Represents the port is in transition between logic 0 and logic 1 states.
- hmX: Represents unknown state which could be in either logic 0 or logic 1 state.
- hmZ: Implies the port is in floating or high-impedance state.
- hmZL: Implies the port is floating and also in logic 0 state.
- hmZH: Implies the port is floating and also in logic 1 state.

Since the array space is allocated within any of the functions, it is required to free the array within the calling function when the array is no longer needed, or before exiting the calling function.

hmSetDigitalStateValue

```
int hmSetDigitalStateValue (state_name, state)
char *state_name;
HMLogic state;
```

hmSetDigitalStateValue sets the logic state of an internal digital state variable, specified by the string state_name, to be the value specified by state. The state value has the data type of hmLogic.

hmSetDigitalStateValueById

```
int hmSetDigitalStateValueById (state_id, state)
int state_id;
HMLogic state;
```

hmSetDigitalStateValueById sets the logic state of an internal digital state variable to be the value specified by state. The state variable name is specified by its ID number state_id. The state value has the data type of hmLogic.

hmSetDigitalStateVecValue

```
int hmSetDigitalStateVecValue (state_vec, start, end,
state_array)
char *state_vec;
int start, end;
HMLogic *state_array;
```

hmSetDigitalStateVecValue sets the logic states of internal digital state vector state_vec[start:end] to be the array values specified by the array state_array[0:abs(start-end)], where abs(x) represents the absolute value of x and state_array is a pointer to a hmLogic array. The state value has the data type of hmLogic.

hmSetDigitalStateVecValueById

```
int hmSetDigitalStateVecValueById (state_vec_id, start, end,
state_array)
int state_vec_id;
int start, end;
HMLogic *state_array;
```

hmSetDigitalStateVecValueById sets the logic states of internal digital state vector state_vec[start:end] to be the array values specified by the array state_array[0:abs(start-end)], where abs(x) represents the absolute value of x and state_array is a pointer to a hmLogic array. The state vector name is

specified by its ID number state_vec_id. The state value has the data type of hmLogic.

Miscellaneous Interface APIs

This section lists the miscellaneous interface APIs:

- [hmModelInstParamValue](#)
- [hmModelInstStrParamValue](#)
- [hmSimOptValue](#)
- [hmFree](#)
- [hmMsg, hmWarn, hmError](#)

hmModelInstParamValue

```
char hmModelInstParamValue (param_name, param_value)
char *param_name;
double *param_value;
```

hmModelInstParamValue returns the parameter value of the specified param_name for the current model instance. If the function returns 1, param_name is defined in the model instance and the value is returned in param_val. If the function returns 0, the parameter cannot be found in the model instance, and no value is written to param_val.

Example 77

```
* netlist
x1 1 2 3 a2d_model start=10n

/* C model */
if (hmModelInstParamValue("start", &start_val)) {
    ...
    ...
}
else {
    printf("parameter '\'start\'' is not defined\n");
}
```

hmModelInstStrParamValue

```
char hmModelInstStrParamValue (param_name, str_param_value)
char *param_name;
char **str_param_value;
```

hmModelInstStrParamValue returns the string parameter value of the specified param_name for the current model instance. If the function returns 1, param_name is defined in the model instance and the value is returned in str_param_val. If the function returns 0, the parameter cannot be found in the model instance, and no value is written to str_param_val. Refer to [Example 78](#).

Example 78

```
* netlist
x1 1 2 3 a2d_model key="abc"

/* C model */
if (hmModelInstStrParamValue(key, &sptr) ) {
    if (!strcmp(sptr, "abc")) {
        ...
        ...
    }
} else {
    printf("parameter key is not defined\n");
}
```

hmSimOptValue

```
double hmSimOptValue(opt)
HMSimOpt opt;
```

hmSimOptValue returns the value of a simulation parameter specified by opt. Simulation parameter can be one of the followings:

- HMTEMP - circuit temperature
- HMSTOP - transient time

hmFree

```
void hmFree (mem_pointer)
void *mem_pointer;
```

hmFree frees the memory allocated by functions like hmDigitalPortBusValueById, hmDigitalPortBusValue which allocate memories and return them during runtime. mem_pointer specifies the memory location.

hmMsg, hmWarn, hmError

```
void hmMsg(fmt, ...)
char *fmt;
void hmWarn(fmt, ...)
char *fmt;
void hmError(fmt, ...)
char *fmt;
```

hmMsg, hmWarn and hmError are 3 functions that print messages to both the screen and the HSIM log file. The usage is similar to C library printf(). hmMsg will print the formatted message to both the screen and logoff.

hmWarn will print Warning: and then the formatted message to both the screen and log file.

hmError will print Error: and then the formatted message to both the screen and log file and abort.

Modeling Memory Core

This section lists the modeling memory core commands.

- [hmDefMemCore](#)
 - [hmInitMemCore](#)
 - [hmReadMemCore](#)
 - [hmWriteMemCore](#)
-

hmDefMemCore

```
int hmDefMemCore (model_pointer, mem_name)
pHMMODEL model_pointer;
char *mem_name;
```

hmDefMemCore defines a memory core within the functional model referenced by the pointer model_pointer, which was created earlier by hmCreateModel.

The name of memory core is specified by the string mem_core.

hmDefMemCore, as the functions listed in [Chapter 17, Timing and Power Analysis, Hold Time Check on page 510](#), can be called only from the model interface function hsimC_Model. Each core cell state has a data type of hmLogic.

hmInitMemCore

```
hmInitMemCore (mem_name, addr_size, data_size, def_state,  
data_file)  
char      *mem_core, *data_file, *default_state;  
int       addr_size, data_size;
```

hmInitMemCore allocates and initializes the memory core specified by the string mem_name that was declared by the function hmDefMemCore. The memory size is defined by the number of address bits, addr_size, and by the number of data bits, data_size. Refer to [Example](#).

A total of $2^{10}=1024$ words of memory core cells are defined if addr_size=10. Each word is 8-bit wide if data_size=8.

The default initial state for each memory word is specified by a hexadecimal string default_state.

Each memory word has the default initial state 0xffff0 if default_state is "ffff0" and assuming data_size=16. The exception is the initial state is overwritten by the bit pattern defined in the file with the filename specified by the string data_file. The format of the initial state in the file is as follows.

```
addr1      data11 data12 data13 .....  
addr2      data21 data22 data23 .....  
.....
```

This sets the initial state at address addr1 to be data11, the initial state at address addr1+1 to be data12, the initial state at address addr1+2 to be data13, and so forth. Similarly, the initial state at address addr2 is data21, and data22 at address addr2+1, data23 at address addr2+2. Each address or data value is represented as a hexadecimal number (see the next example).

If addr_size=8 and data_size=12, then the initial data file may appear as follows.

```
00 21f 59a 3be  
39 8c2 3d2 107 29f
```

This initializes the data to be 21f at address 0, 59a at address 1, 3be at address 2, 8c2 at address 39, 3d2 at address 3a, 107 at address 3b, and 29f at address 3c.

hmReadMemCore

```
hmReadMemCore (mem_name, addr, data)
char *mem_name;
HMLogic*addr, *data;
```

hmReadMemCore reads the memory core, specified by the string mem_name, and sets the bit pattern at the array data to be the memory core content at address addr.

hmWriteMemCore

```
hmWriteMemCore (mem_name, addr, data)
char *mem_name;
hmLogic *addr, *data;
```

hmWriteMemCore writes the bit pattern at the array data into the memory core addressed by addr.

Examples

This section provides examples for the following applications:

- [A/D and D/A Converter Examples on page 698](#)
- [Port Delay Examples on page 702](#)
- [RAM Example on page 704](#)
- [Event Handling Example on page 708](#)

A/D and D/A Converter Examples

The A/D and D/A converter example contains a 16-bit A/D converter and a 16-bit D/A converter. A 200-KHz sinusoidal input voltage source vin drives the A/D converter which has the 16-bit digital output bus b0, b1, ..., b15. The digital output bus from the A/D converter is applied to the 16-bit D/A converter that has the analog output port vout. The A/D conversion occurs at positive clock edge while the D/A conversion occurs at negative clock edge. The analog output voltage at vout should follow the analog input at vin.

```

* SPICE netlist file
* 16-bit A/D and D/A converters
.param HSIMFMODLIB=".a.so"
x1 vin vref clk b0 b1 b2 b3 b4 b5 b6 b7
+ b8 b9 b10 b11 b12 b13 b14 b15 a2d size=16
x2 vout vref clk b0 b1 b2 b3 b4 b5 b6 b7
+ b8 b9 b10 b11 b12 b13 b14 b15 d2a size=16
vclk clk gnd pulse 0 3.0 0 .2n .2n 2n 5n
vin vin gnd sin(1.5 1.5 2e5)
vref vref gnd 3.0
.tran 1n 10u
.print v(*) level=1
.end
***** C functional model file *****
#include <math.h>
#include <stdio.h>
#include "hm.h"
/* digital to analog converter */
void d2a()
{
    static int data_size, d_id, vout_id, clk_id, conv_id,
vref_id;
    int
        i;
    HMLogic
        clk, conv, *dd;
    double
        vref, vout, vinc;
    double
        *cap_array;
    if (hmSimStage() == HMSTART) {
        /* convert port name to port id; using port id */
        /* is more efficient than using the name since */
        /* it eliminates the time needed to find the */
        /* port index when the port name is used in */
        /* each later API reference */
        d_id = hmPortName2PortId("d");
        vout_id = hmPortName2PortId("vout");
        vref_id = hmPortName2PortId("vref");
        clk_id = hmPortName2PortId("clk");
        conv_id = hmPortName2PortId("conv");
        /* the data bus size is given by the instantiation call */
        data_size = hmPortBusSizeById(d_id);
        hmSetAnalogPortValueById(vout_id, 0.0);
        hmSetDigitalStateValue("conv", hmZero);
        /* set data bus capacitance */
        cap_array = (double *) calloc (data_size,
sizeof(double));
        for (i=0; i<data_size; i++)
            cap_array[i] = 1.0E-13; /* 100fF */
        hmSetPortBusCapById (d_id, 0, data_size-1, cap_array);
    }
}

```

Chapter 23: C Language Functional Model

Examples

```
else if (hmSimStage() == HMSIM) {
    clk=hmDigitalPortValueById (clk_id);
    conv=hmDigitalStateValueById (conv_id);
    /* no data conversion until the next clock toggle */
    if (clk==hmOne) {
        hmSetDigitalStateValueById (conv_id, hmZero);
        return;
    }
    /* data conversion occurs at negative clk edge */
    if (clk==hmZero && conv==hmZero) {
        vout=0.0;
        vref=hmAnalogPortValueById (vref_id);
        /* digital to analog data conversion */
        vinc=0.5* vref;
        dd=hmDigitalPortBusValueById (d_id, 0, data_size-
1);
        for (i=0; i<data_size; i++) {
            vout +=dd[data_size-i-1]*vinc;
            vinc *=0.5;
        }
        hmSetAnalogPortValueById (vout_id, vout);
        hmSetDigitalStateValueById (conv_id, hmOne);
        /* data converted */
        /* the array space of dd is allocated in
        hmDigitalPortBusValue */
        /* and is required to avoid memory leakage */
        hmFree (dd);
    }
}
/* analog to digital converter */
void a2d()
{
    int
    mt=2;
    static int
    dout_id, clk_id, conv_id;
    static HMLogic
    HMLogic
    double
    vdiff;
    if (hmSimStage() == HMSTART) {
        data_size=hmPortBusSize("dout");
        dout_id=hmPortName2PortId("dout");
        conv_id=hmPortName2PortId("conv");
        clk_id=hmPortName2PortId("clk");
        dout=(HMLogic *) calloc(data_size,
        sizeof(HMLogic));
        hmSetDigitalPortBusValueById(dout_id, 0,
```

```

        data_size-1, dout);
                hmSetDigitalStateValueById(conv_id, hmZero);
                return;
            }
            if(hmSimStage() == HMSIM) {
                clk=hmDigitalPortValueById(clk_id);
                conv=hmDigitalStateValueById(conv_id);
                /* data conversion occurs at rising clock edge */
                if (clk==hmZero) {
                    hmSetDigitalStateValueById(conv_id, hmZero);
                    return;
                }
                vin=hmAnalogPortValue("vin");
                vref=hmAnalogPortValue("vref");
                if(clk==hmOne && conv==hmZero) {/* rising clock
edge */
                    for (i=0; i<data_size; i++) {
                        vdiff=vin - vref/mt;
                        dout[data_size-i-1]=(vdiff > 0.0)? hmOne:
hmZero;
                        vin=(vdiff > 0.0)? vdiff : vin;
                        mt +=mt;
                    }
                    hmSetDigitalPortBusValueById(dout_id, 0,
data_size - 1, dout);
                    hmSetDigitalStateValueById(conv_id, hmOne);
                }
            }
        void hsimC_model()
{
    pHMMODEL pf, pg;
    pf=hmCreateModel("a2d",a2d);
    hmDefAnalogPort(pf,"vin", hmInput); /* analog input voltage
*/
    hmDefAnalogPort(pf, "vref", hmInput); /* supply voltage */
    hmDefDigitalPort(pf, "clk", hmInput); /* clock input */
    hmDefVarDigitalPortBus(pf, "dout", hmOutput, "size");
/* digital output */
    hmDefDigitalState(pf,"conv"); /* clock edge recorder */
    pg=hmCreateModel("d2a", d2a);
    hmDefAnalogPort(pg, "vout", hmOutput); /* analog output
voltage */
    hmDefAnalogPort(pg, "vref", hmInput); /* supply voltage */
    hmDefDigitalPort(pg, "clk", hmInput); /* clock input */
    hmDefVarDigitalPortBus(pg, "d", hmInput, "size"); /* digital
input */
    hmDefDigitalState(pg,"conv"); /* clock edge recorder */
}

```

Port Delay Examples

This section has the following port delay examples:

- [Port Delay Example - C Functional Model File](#)
- [Port Delay Example - HSIM Netlist File](#)

Port Delay Example - C Functional Model File

```
/*
 *      Synopsys Corporation
 */
/*
 *      Port Delay example
 *      C Functional Model File
*/
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "hm.h"
void inv() /* just invert */
{
    static in_id, out_id;
    HMLogic in;
    if (hmSimStage()==HMSTART) {
        /* convert port name to port id; using port id*/
        /* is more efficient than using the name since*/
        /* it eliminates the time needed to find the*/
        /* port index when the port name is used in*/
        /* each later API reference           */
        in_id=hmPortName2PortId("in");
        out_id=hmPortName2PortId("out");
    }
    in=hmDigitalPortValueById (in_id);
    if (in==hmOne) {
        hmSetDigitalPortValueById (out_id, hmZero);
        return;
    }
    else if (in==hmZero) {
        hmSetDigitalPortValueById (out_id, hmOne);
        return;
    }
}
void dinv() /* delayed inversion */
{
    static in_id, out_id;
    HMLogic in;
    if (hmSimStage()==HMSTART) {
        /* convert port name to port id; using port id*/
        /* is more efficient than using the name since*/
        /* it eliminates the time needed to find the*/
        /* port index when the port name is used in*/
        /* each later API reference           */
        in_id=hmPortName2PortId("in");
        out_id=hmPortName2PortId("out");
        /* Set the port rising delay time 15ns and falling
delay time 25ns */
        hmSetDigitalPortDelayById(in_id, 15e-9, 25e-9);
    }
}
```

Chapter 23: C Language Functional Model

Examples

```
        }
        in=hmDigitalPortValueById (in_id);
        if (in==hmOne) {
            hmSetDigitalPortValueById (out_id, hmZero);
        return;
        }
        else if (in==hmZero) {
            hmSetDigitalPortValueById (out_id, hmOne);
        return;
        }
    }
void hsimC_model()
{
    pHMMODEL pf, pg;
    pf=hmCreateModel("inv",inv);
    hmDefDigitalPort(pf,"in", hmInput); /* input */
    hmDefDigitalPort(pf,"out", hmOutput); /* output */
    pg=hmCreateModel("dinv", dinv);
    hmDefDigitalPort(pg,"in", hmInput); /* input */
    hmDefDigitalPort(pg,"out", hmOutput); /* output */
}
```

Port Delay Example - HSIM Netlist File

```
*
*      Synopsys Corporation
*
*      Port Delay Example
*      hsim netlist file
*
*.param HSIMFMODLIB=".fmdq_debug.dll"
.param HSIMFMODLIB=".fmdq.lib"
.param HSIMSPEED=5
.param HSIMUSEHM="inv dinv"
x1 vin vout inv
x2 vin dvout dinv
*vin vin 0 pwl 0 0 0.8u 0 0.81u 3v 1.5u 3v 1.51u 0v
vin vin 0 pulse 0v 3v 100n 10n 10n 100n 200n
.tran 1n 2000n
.print v(*) level=1
.end
```

RAM Example

The RAM example models the READ/WRITE operations of a RAM with 10-bit address and 8-bit data buses. The initial state of some memory core cells,

other than the default state setting, is defined in the file core_state. The address and data inputs come from the vector file ram_vec and the simulation result is checked by the pattern specified in ram_out.vec.

Chapter 23: C Language Functional Model

Examples

```
* SPICE netlist file
* memory with 10-bit address and 8-bit data buses
.param HSIMOUTPUT=fsdb
.param HSIMFMDLIB=".//a.so"
.param hsimvectorfile=ram.vec
.param hsimvectorfile=ram_out.vec
x1 CE READ_RAM ADDR9 ADDR8 ADDR7 ADDR6 ADDR5
+ ADDR4 ADDR3 ADDR2 ADDR1 ADDR0
+ DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0 hmram
+ addr_size=10 data_size=8
.print v(*) level=1
.tran 1n 2u
.end
; input vector file ram.vec
signal CE READ_RAM ADDR9 ADDR8 ADDR7 ADDR6 ADDR5
+ ADDR4 ADDR3 ADDR2 ADDR1 ADDR0
+ DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
radix          11      244      44
io             ii      iii      bb
0              00      000      00
100            01      000      a5
200            10      137      37
300            10      2f4      2d
400            10      0ec      c6
500            11      2f4      zz
600            11      137      zz
700            11      15a      zz
800            11      2f4      zz
900            11      0ec      zz
1000           11      326      zz
; output vector comparison file ram_out.vec
signal DATA7 DATA6 DATA5 DATA4 DATA3 DATA2 DATA1 DATA0
io oo
radix 44
;DATA
0          XX
100         XX
200         XX
300         XX
400         XX
505         2d
605         37
705         f0
805         2d
905         c6
1005        cf
***** C functional model file *****/
#include <stdio.h>
```

```

#include "hm.h"
void hmram()
{
    static int          DATAid, ADDRid, CEid, RWid;
    int                ADDR_COUNT, DATA_COUNT;
    HMLogic            *DATA, *ADDR, CE, READ, DATA_BUF[100];
    ADDR_COUNT=hmPortBusSize("ADDR"); /* address bus size */
    DATA_COUNT=hmPortBusSize("DATA"); /* data bus size */
    if(hmSimStage()==HMSTART) {
        /* using id instead of name is more */
        /* efficient in each API reference */
        DATAid=hmPortName2PortId("DATA");
        ADDRid=hmPortName2PortId("ADDR");
        CEid=hmPortName2PortId("CE");
        RWid=hmPortName2PortId("RW");
        /*           allocate a core memory of ADDR_COUNT words
     */
        /*
         *           each word has DATA_COUNT bits*/
        /*
         *           the default state of each core cell is f0*/
        /*
         *           except for those cells defined in file
         */
        core_state
            hmInitMemCore("core", ADDR_COUNT, DATA_COUNT,
"f0",
            "core_state");
        }
        ADDR=hmDigitalPortBusValueById(ADDRid, ADDR_COUNT-1,
0);
        CE=hmDigitalPortValueById(CEid);
        READ=hmDigitalPortValueById(RWid);
        if (CE==hmZero) /* RAM disabled */
            hmDisablePortBusDriveById(DATAid, 0, DATA_COUNT-
1);
        }
        else if (CE==hmOne) {
            if (READ==hmOne) /* read operation */
                hmEnablePortBusDriveById(DATAid, DATA_COUNT-
1, 0);
            hmReadMemCore("core", ADDR, DATA_BUF);

            hmSetDigitalPortBusValueById(DATAid, DATA_COUNT-1, 0, DATA_BUF);
        }
        else if (READ==hmZero) /* write operation */

            hmDisablePortBusDriveById(DATAid, 0, DATA_COUNT-1);
            DATA=hmDigitalPortBusValueById(DATAid,
DATA_COUNT-1, 0);
            hmWriteMemCore("core", ADDR, DATA);
        /* the DATA array is allocated in
        hmDigitalPortBusValue      */
    }

```

```
        /* need to free this array otherwise memory leakage
occurs */
                hmFree (DATA) ;
            }
        }
        /* need to free this array to avoid memory leakage */
        hmFree(ADDR) ;
    }
hsimC_model()
{
    pHMMODEL      pf;
    pf=hmCreateModel("hmram",hmram); /* RAM model definition */
    hmDefDigitalPort(pf,"CE",hmInput); /* enable control */
    hmDefDigitalPort(pf,"RW",hmInput); /* Read and Write control
*/
    hmDefVarDigitalPortBus(pf, "ADDR", hmInput, "addr_size");
    /* address */
    /* bidirectional data bus; the bus size is determined by */
    /* parameter data_size in the instantiation call */
    hmDefVarDigitalPortBus(pf, "DATA", hmBiput, "data_size");
    /* memory core definition */
    hmDefMemCore(pf, "core");
}
```

Event Handling Example

This section describes the following event handling example:

- [hmEventHandle](#)

hmEventHandle

```
HMEventHandle hmSchedulePortEvent(char *portname, double time,
void *userdata); HMEventHandle hmSchedulePortEventById(int
portid, double time, void *userdata); HMEventHandle
hmGetCurrentEvent(void);
int hmDescheduleEvent(HMEventHandle eventhandle);
int hmGetEventPortId(HMEventHandle eventhandle);
void * hmGetEventData(HMEventHandle eventhandle);
```

Note: Allocating and freeing memory for user-defined data is your responsibility.

Note: Do *not* de-schedule an event that has been processed.

```
-----fmev.c-----
#include <stdlib.h>
#include <math.h>
#include <stdio.h>
#include "hm.h"
void inv()
{
    static int in_id, out_id, id1, id2, id3;
    HMLogic in;
    HMEventHandle *ev;
    int id;
    int d;
    if (hmSimStage() == HMSTART) {
        /* convert port name to port id; using port id*/
        /* is more efficient than using the name since*/
        /* it eliminates the time needed to find the*/
        /* port index when the port name is used in*/
        /* each later API reference */
        in_id=hmPortName2PortId("in");
        out_id=hmPortName2PortId("out");
        id1=hmPortName2PortId("out1");
        id2=hmPortName2PortId("out2");
        id3=hmPortName2PortId("out3");
        hmSchedulePortEventById(id1, 1e-10, (void*)0);
        hmSchedulePortEventById(id2, 2e-10, (void*)0);
        hmSchedulePortEventById(id3, 3e-10, (void*)0);
        hmSetDigitalPortValueById(out_id, hmZero);
    }
    else if (hmSimStage() == HMSIM) {
        in=hmDigitalPortValueById (in_id);
        while(ev=hmGetCurrentEvent ()) {
            id=hmGetEventPortId(ev);
            if(id==id1) {
                d=(int)hmGetEventData(ev);
                if(d==1) {
                    hmSetDigitalPortValueById(id1, hmZero);
                    hmSchedulePortEventById(id1, hmPresentTime() + 1e-10, (void*)0);
                }
            }
            else {
                hmSetDigitalPortValueById(id1, hmOne);
                hmSchedulePortEventById(id1, hmPresentTime() + 1e-10, (void*)1);
            }
        }
        if(id==id2) {
            d=(int)hmGetEventData(ev);
            if(d==1) {
                hmSetDigitalPortValueById (id2, hmZero);
                hmSchedulePortEventById (id2, hmPresentTime() + 2e-10, (void*)0);
            }
        }
    }
}
```

Chapter 23: C Language Functional Model

Examples

```
        }
    else {
        hmSetDigitalPortValueById (id2, hmOne);
        hmSchedulePortEventById (id2, hmPresentTime() + 2e-10, (void*)1);
    }
}
if(id==id3) {
    d=(int)hmGetEventData(ev);
    if(d==1) {
        hmSetDigitalPortValueById (id3, hmZero);
        hmSchedulePortEventById (id3, hmPresentTime() + 3e-10, (void*)0);
    }
    else {
        hmSetDigitalPortValueById (id3, hmOne);
        hmSchedulePortEventById (id3, hmPresentTime() + 3e-10, (void*)1);
    }
}
/*
no data conversion until the next clock toggle */
if (in == hmOne) {
    hmSetDigitalPortValueById (out_id, hmZero);
    return;
}
else if (in == hmZero) {
    hmSetDigitalPortValueById (out_id, hmOne);
    return;
}
}
void hsimC_model()
{
PHMMODEL pf;
pf=hmCreateModel("inv",inv);
hmDefDigitalPort(pf,"in", hmInput); /* clock input */
hmDefDigitalPort(pf,"out", hmOutput); /* clock input */
hmDefDigitalPort(pf,"out1", hmOutput); /* clock input */
hmDefDigitalPort(pf,"out2", hmOutput); /* clock input */
hmDefDigitalPort(pf,"out3", hmOutput); /* clock input */
}
*****
```

HSIM Three-Dimensional Integrated Circuit (3D-IC) Modularization

This chapter describes the HSIM® solution to modularize IC chips inside a 3D-IC to be simulated as a single circuit.

Overview of 3D-IC Simulation Netlist

A three-dimensional integrated circuit is a single chip that integrates two or more layers of active electronic components into a single circuit. All components on the layers communicate using on-chip signaling, whether vertically or horizontally. The layers in the 3D-IC are IC chips that might be fabricated using different technology processes and operate at different sets of parameters, including temperature and geometry scaling. The HSIM solution modularizes these IC chips and enables the simulation of the 3D-IC as a single circuit.

HSIM supports HSPICE syntax for 3D-IC netlists as described in the *HSPICE User Guide: Basic Simulation and Analysis*, Three-Dimensional Integrated Circuit (3D-IC) Modularization chapter.

3D-IC Netlist Construct and Usage

The 3D-IC netlist construct and usage in HSIM follow the same rules as described in the *HSPICE User Guide: Basic Simulation and Analysis*, Three-Dimensional Integrated Circuit (3D-IC) Modularization chapter, 3D-IC Netlist Construct and Usage section.

Transient Analysis and Alters Simulation Features

The current 3D-IC solution applies to transient analysis only. It supports sweeps for temperature and parameters, as well as .ALTER features as described in *HSPICE User Guide: Basic Simulation and Analysis*, Three-Dimensional Integrated Circuit (3D-IC) Modularization chapter, Transient Analysis and Alters Simulation Features section.

Full Circuit Example

A full circuit example of a 3D-IC is described in *HSPICE User Guide: Basic Simulation and Analysis*, Three-Dimensional Integrated Circuit (3D-IC) Modularization chapter, Full Circuit Example section.

Module-Based HSIM Commands

In a regular netlist, HSIM supports many local commands at the subcircuit level. With the introduction of modules in a 3D-IC netlist, HSIM extends the support of selected local commands to the module level. In addition, some global commands in a regular netlist are now supported as local commands at the module level in a 3D-IC netlist. These commands are global in a module scope, meaning they cannot be localized to subcircuit levels. See [Table 101](#).

Table 101 HSIM Command Module and Subcircuit Support

Command	Module Support	Subcircuit Support
HSIMVERILOGA	x	
HSIMUSEVA	x	
HSIMSPEED	x	x
HSIMANALOG	x	x
HSIMSPICE	x	x
HSIMVDD	x	x

Table 101 HSIM Command Module and Subcircuit Support

Command	Module Support	Subcircuit Support
HSIMALLOWEDDV	x	x
HSIMSNCS	x	x
HSIMSTEADYCURRENT	x	x
HSIMGCSC	x	x
HSIMPOSTL	x	x
HSIMVHTH, HSIMVLTH	x	

For example:

```
.module mod1
.param HSIMSPICE=2
...
subckt inv in out
mp out in vdd vdd nch l=0.09u w=0.1u
mn out in gnd gnd pch l=0.09u w=0.2u
.ends
...
endmodule
```

In this example, `HSIMSPICE` is set to 2 for the `mod1` module, which means model equation is used in the simulation of all subcircuits defined within the scope of `mod1`, including `inv`.

3D-IC Post-Layout Back-Annotation

The HSIM tool supports two DPF back-annotation flows for a 3D-IC netlist:

- Flat IC module DPFs with a separate DPF for silicon interposer.
- Full 3D-IC flat extracted DPF.

Flat IC Module DPFs with a Separate DPF for Silicon Interposer

This flow lets you reuse the extracted DPFs of individual IC modules. In the so-called 2.5D integration, in which IC modules are extracted and verified individually before assembly using a silicon interposer that is extracted separately. The DPF back-annotation steps in this flow are:

- Extract each IC module with a separate DPF/DSPF file. Each DPF/DSPF file must contain the IC module top subcircuit definition.
- Extract the silicon interposer layer as a flat DPF/DSPF file.
- Set the DPF/DSPF file reference in the appropriate module.

For example:

```
Top netlist – top.sp
*simulation setup
...
*silicon interposer netlist
X1 ... IC_1::top1
X2 ... IC_2::top2
.param hsimdpf="silicon_interposer.dspf"
*IC modules
.module IC_1
.param hsimdpf="IC_1_flat.dspf"
.subckt top1 ...
...
.ends
.endmodule IC_1
.module IC_2
.param hsimdpf="IC_2_flat.dspf"
.subckt top2 ...
...
.ends
.endmodule IC_2
```

DPF/DSPF file for module "IC_1" – IC_1_flat.dpsf

```
.subckt top1
*Extracted RLC nets
* |NET ...
...
*INSTANCE
M1... w=0.3u l=0.2u ...
...
.ends
```

DPF/DSPF file for module "IC_2" – IC_2_flat.dpsf

```
.subckt top2
*Extracted RLC nets
* | NET ...
...
*INSTANCE
M1... w=0.3u l=0.2u ...
...
.ends
```

Full 3D-IC Flat Extracted DPF

In this flow the full 3D-IC is extracted. This could be the case for true 3D-IC integration, where the IC modules are stacked on top of each other and their layouts must be extracted together. The DPF back-annotation in this flow is as follows:

- You provide the fully extracted device parameters in the flat DPF files.
- Usage is the same as the single IC module flat DPF back-annotation flow.
- The DPF file is not referenced to any of the .module scope.

For example:

```
Top netlist – top.sp
*simulation setup
...
*silicon interposer netlist
X1 ... IC_1::top
...
.param hsimdpf="full_MTS_flat.dspf"
*IC modules
.module IC_1
.subckt top
...
.ends \
.endmodule IC_1
```

DPF file – full_MTS_flat.dpsf

```
*Extracted RLC nets
* |NET ...
...
*INSTANCE
MX1.M1... w=0.3u l=0.2u ...
MX1.M2... w=0.3u l=0.2u ...
```

3D-IC Parasitic (SPF) Back-Annotation

Similar to DPF back-annotation, two SPF back-annotation flows are supported for a 3D-IC netlist:

- Flat IC module SPFs with a separate SPF for silicon interposer.
- Full 3D-IC flat extracted SPF.

Flat IC Module SPFs with a Separate SPF for Silicon Interposer

The SPF back-annotation steps in this flow are as follows:

- Extract each IC module with a separate SPF file. Each SPF file must contain the IC module top subcircuit definition.
- Extract the silicon interposer layer as a flat SPF file.
- Set the SPF file reference in the appropriate module.

For example:

Top netlist – top.sp

```
*simulation setup
...
*silicon interposer netlist
X1 ... IC_1::top1
X2 ... IC_2::top2
.param hsimspf="silicon_interposer.dspf"
*IC modules
.module IC_1
.param hsimspf="IC_1_flat.dspf"
.subckt top1 ...
...
.ends
.endmodule IC_1
.module IC_2
.param hsimspf="IC_2_flat.dspf"
.subckt top2 ...
...
.ends
.endmodule IC_2
```

DSPF file for module "IC_1" - IC_1_flat.dpsf

```
.subckt top1
*Extracted RLC nets
* | NET ...
...
.ends
```

DSPF file for module "IC_2" - IC_2_flat.dpsf

```
.subckt top2
*Extracted RLC nets
* | NET ...
...
.ends
```

Full 3D-IC Flat Extracted SPF

The SPF back-annotation steps in this flow are as follows:

- You provide the fully extracted device parameters in the flat SPF files.
- The usage is the same as the single IC module flat SPF back-annotation flow.
- The SPF file is not referenced to any of the .module scope.

For example:

Feedback

Chapter 24: HSIM Three-Dimensional Integrated Circuit (3D-IC) Modularization 3D-IC Post-Layout Back-Annotation

Top netlist – top.sp

```
*simulation setup
...
*silicon interposer netlist
X1 ... IC_1::top
...
.param hsimspf="full_MTS_flat.dspf"
*IC modules
.module IC_1
.subckt top

...
.ends
.endmodule IC_1
```

SPF file – full_MTS_flat.dpsf

```
*Extracted RLC nets
* |NET ...
...
```

Part 4: HSIM Appendices

Device Model Reference

Lists sources of reference information for device models.

Reference Sources for Device Models

The information shown in the table below is provided to assist users in locating valuable background information not covered in this manual.

Table 102 Device Model References

Models	Reference
BSIM3 & BSIM4	http://www-device.eecs.berkeley.edu/~bsim3/
BSIM3SOI & BSIM4SOI	http://www-device.eecs.berkeley.edu/~bsimsoi/
MOS9 & MOS11	http://www.semiconductors.philips.com/phili..._Models
EKV	http://legwww.epfl.ch/ekv
HiSIM	http://www.starc.jp/kaihatu/pdgr/hisim
UCB SPICE	http://www-cad.eecs.berkeley.edu/Software/software.html
VBIC BJT	http://www.fht-esslingen.de/institute/iafgp/neu/VBIC/
MEXTRAM BJT	http://www.semiconductors.philips.com/phili..._Models
HICUM BJT	http://www.iee.et.tu-dresden.de/iee/eb/comp_mod.html
JUNCAP2	http://www.semiconductors.philips.com/Philips_Models/additional/juncap/index.html

Feedback

Appendix A: Device Model Reference

Reference Sources for Device Models

Table 102 Device Model References (Continued)

Models	Reference
PSP	http://pspmodel.ee.psu.edu/

Custom Output Interface

Provides information on how to accommodate a specific output format not compatible with existing design flows using a template library to construct a generator of a waveform that will be suitable for that format.

- Overview of Output Formats
- Call Custom Library Mechanism
- Functional Specification
- Common Include File
- Create and Initialize the Waveform File Function
- Create a Header Function
- Create a Waveform Into a Waveform File
- Create a Duplicate Waveform Into a Waveform File
- Add Waveform Values Into a Waveform File
- AddNextAnalogValueChange
- End the Waveform File Process
- Building a Waveform File Generator Library

Overview of Output Formats

HSIM supports the following output formats.

- FSDB Output Format
- WSF Output Format
- EPIC ASCII Output Format

Appendix B: Custom Output Interface

Call Custom Library Mechanism

- HSIM Output Format
- WDF Sandwork Output Format

On some occasions the regular output formats may not be compatible with an existing design flow. To accommodate a specific output format, a template library can be used to construct a generator of a waveform that will be suitable for that format.

For HSIM to load the template library, a dynamic library loader, such as dlopen on the Solaris system, is required.

The examples in this appendix illustrate the use of this interface only, and include the following:

- The header file coi.h which contains a list of common data shared by HSIM and your waveform generator library. The coi.h is available in \$HSIM_HOME/etc/include after the completion of HSIM installation.
- The generation library myformat.c which illustrates a simple implementation of the interface.

Note: In operation, the detailed content of the library is developed by you.

Call Custom Library Mechanism

The output file format is specified by the following HSIM command.

```
.param HSIMOUTPUT=out_format
```

To specify a particular output format, the name of the corresponding format must be set with [HSIMOUTPUT on page 119](#). An additional HSIM parameter is used to specify the full path of the waveform file generator library.

```
.param HSIMCOILIB="full path of the waveform file generator library"
```

The full path should include the library name. [HSIMCOILIB on page 74](#) is optional. If there is no HSIMCOILIB specified, HSIM searches for the lib<out_format>.so in the following directories, in the order shown:

- Run directory
- \$HOME directory

- \$HSIM_HOME/platform/<port>/bin
 - LD_LIBRARY_PATH on Solaris and on Linux, and SHLIB_PATH on HP
- where <output_format> is the value of [HSIMOUTPUT](#) on page 119 and all the letters in <output_format> have to be in upper case.

Functional Specification

For flexibility, the interface specification handles the following items:

- Common include file
- Creation and initialization of a waveform file function
- Creation of a header function
- Creation of a waveform into a waveform file function
- Creation of a duplicate waveform into a waveform file function
- Addition of values to one or more waveforms into a waveform file function
- Completion of the waveform file creation function

A procedural interface is provided for you to prepare a waveform file generator library. For transient analysis, the HSIMAPI calling sequence is in the following order:

```
out_format_CreateWaveFile()  
out_format_CreateHeader()  
out_format_BeginCreateWave()  
out_format_CreateWave()  
out_format_EndCreateWave()  
out_format_AddNextDigitalValueChange() [optional; see note below]  
out_format_AddNextAnalogValueChange() [optional; see note below]  
out_format_CloseWaveFile()
```

For DC analysis, the HSIMAPI calling sequence is in the following order:

```
out_format_CreateWaveFileDC()  
out_format_CreateHeaderDC()  
out_format_BeginCreateWaveDC()  
out_format_CreateWaveDC()  
out_format_EndCreateWaveDC()  
out_format_AddNextAnalogValueChangeDC()  
out_format_CloseWaveFileDC()
```

Appendix B: Custom Output Interface

Common Include File

Note: The API out_format serves as a name holder. out_format is specified in the command .param HSIMOUTPUT=out_format. A different name can be used, such as MYFORMAT. Except for steps 6 and 7 , all other steps are called only once by HSIM. Steps 6 and 7 are called when a signal value changes.

HSIM invokes these functions only when a value change occurs to a specific signal at the current simulation time. At a particular simulation time, HSIM calls these functions exclusively for those signals with value changes. The generated waveform file is value-change based instead of tabular based.

For AC analysis, HSIM API calling sequence is in the following order:

```
out_format_CreateWaveFileAC()
out_format_CreateHeaderAC()
out_format_BeginCreateWaveAC()
out_format_CreateWaveAC()
out_format_EndCreateWaveAC()
out_format_AddNextAnalogValueChangeAC()
out_format_format_CloseWaveFileAC()
```

Common Include File

The common include (coi.h) header file describes the common information used by HSIM and the user libraries to generate waveform files. The header file must contain the following declarations.

- Declaration of structs
- Declaration of enumerations

Note: The coi.h header file must remain intact and be used in building the waveform file generator library.

For more information, refer to [Building a Waveform File Generator Library on page 739](#). [Example 79](#) shows the code for the header file coi.h.

Example 79 The header file coi.h

```

/*
 * Synopsys Corporation
 */
#ifndef COI_H
#define COI_H
/* Structures and Enumerations
*****
enum SignalType {
    VOLTAGE=0,
    CURRENT,
    LOGIC,
    POWER,
    DEFAULT
};
***** DON'T CHANGE the following hsimparam data structure. Use
it as it is *****/
typedef struct hsimparam
{
    double StopTime; /* stopTime as in the */
                      /* .TRAN, .DC, or .AC */
    double TimeScale; /* time scale factor */
    double VoltageScale; /* voltage scale factor */
    double CurrentScale; /* current scale factor */
    double Temperature; /* temperature as in .TEMP */
    int NumOfSweep; /* total number of sweep */
                     /* step in .TRAN ... SWEEP */
    int NumOfTemperature; /* total number of */
                          /* temperatures in .TEMP */
    int NumOfAlter; /* total number of .ALTER */
    int CurrOfSweep; /* current number of SWEEP */
    int CurrOfTemperature; /* current number of */
                          /* Temperature */
    int CurrOfAlter; /* current number of .ALTER */
    int CurrOfIteration; /* current number of */
                         /* iteration generated */

    double Vres; /* voltage resolution */
    double Ires; /* current resolution */
***** USED BY HSIM ONLY *****
    char *HierId; /* used by HSIM only */
    double TopSupply; /* used by HSIM only */
    double OutFileSplit; /* used by HSIM only */
    double Cfg_Banner; /* used by HSIM only */
    double Cfg_Xout; /* used by HSIM only */
    double Cfg_OutputFsdbSize; /* used by HSIM only */
    double Cfg_OutputWdfSize; /* used by HSIM only */
    double Cfg_OutputWdfComperess; /* used by HSIM only */
***** */
}

```

Appendix B: Custom Output Interface

Create and Initialize the Waveform File Function

```
double    StartTime; /* startTime as in the .DC */
            /* or .AC sweep */
char*     SweepVarName; /* Sweep Varialbe Name as */
            /* in the .DC or .AC */
            /* sweep */
char*     SweepType; /* For .DC and .AC sweep */
            /* type, LIN, DEC, OCT, */
            /* OTHERS */
int       NumOfMonteCarlo; /* total number of */
            /* MONTECARLO */
int       CurrOfMonteCarlo; /* current number of */
            /* MONTECARLO */
char*     SweepVarName2; /* external Sweep Varialbe */
            /* Name as in the .DC or */
            /* .AC */
} hsimparam;
#endif
```

Create and Initialize the Waveform File Function

The following section provides examples of creating and initializing waveform file functions using the CreateWaveFile function.

CreateWaveFile

Returns a handle and creates the waveform file. When necessary, multiple waveform files are created to store all the waveform information.

Syntax

For Transient Analysis

```
void * Hdl=void * out_format_CreateWaveFile(char *BaseName,
                                             hsimparam *hsimparam);
```

For DC Analysis

```
void *Hdl=void * out_format_CreateWaveFileDC(char
                                              *BaseName, hsimparam *hsimparam);
```

For AC Analysis

```
void *Hdl=void * out_format_CreateWaveFileAC(char
                                              *BaseName, hsimparam *hsimparam);
```

Parameters

BaseName

The string name of the base name for waveform information.

hsimparam

Contains parameters for file initialization.

Return Values

Hdl

The handle attached to the file descriptors, and so on.

-1

Indicates an error occurred.

Note: The out_format is the name specified in .param
HSIMOUTPUT=out_format.

Description

The waveform information is stored in HSIM format in the .ctrl and the .out files. The function creates the files, stores the file descriptors into the customer-specific structure, and returns the reference of that structure. The rest of the APIs use the reference as a handler to dump out the waveform information. The base name is the prefix for those waveform files.

HSIM does not need to know the customer specific structure; the handler is a pass-through token for the rest of APIs.

Example

In this example, hsimparam is defined as follows:

Appendix B: Custom Output Interface

Create a Header Function

```
typedef struct hsimparam {  
    double StopTime;  
    double TimeScale;  
    double VoltageScale;  
    double CurrentScale;  
    double Temperature;  
    int NumOfSweep;  
    int NumOfTemperature;  
    int NumOfAlter;  
    int CurrOfSweep;  
    int CurrOfTemperature;  
    int CurrOfAlter;  
    int CurrOfIteration;  
    double Vres;  
    double Ires;  
} hsimparam;
```

Create a Header Function

The following section provides an example of the header function.

CreateHeader

Adds a header to the waveform file.

Syntax

For Transient Analysis

```
int out_format_CreateHeader(void *Hdl);
```

For DC Analysis

```
int out_format_CreateHeaderDC(void *Hdl);
```

For AC Analysis

```
int out_format_CreateHeaderAC(void *Hdl);
```

Parameters

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

Return Values

0

No errors occurred.

-1

An error occurred.

Create a Waveform Into a Waveform File

The following sections provides examples for creating a waveform into a waveform file. The functions include creating and ending a wave.

BeginCreateWave

Can be used for additional initialization when starting to add new signal(s) into the waveform file.

Syntax

For Transient Analysis

```
int out_format_BeginCreateWave(void *Hdl);
```

For DC Analysis

```
int out_format_BeginCreateWaveDC(void *Hdl);
```

For AC Analysis

```
int out_format_BeginCreateWaveAC(void *Hdl);
```

Parameters

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

Return Values

0

No errors occurred.

Appendix B: Custom Output Interface

Create a Waveform Into a Waveform File

-1

An error occurred.

Description

BeginCreateWave may be used for additional initialization when starting to add new signal(s) into the waveform file. If no special setup takes place before the new waveform is created, BeginCreateWave only needs to return 0 (zero).

CreateWave

Creates a new waveform handle.

Syntax

For Transient Analysis

```
void * WaveHdl=void * out_format_CreateWave( 
void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type);
```

For DC Analysis

```
void * WaveHdl=void * out_format_CreateWaveDC( 
void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type);
```

For AC Analysis

```
void * WaveHdl=void * out_format_CreateWaveAC( 
void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type);
```

Parameters

The input parameters include the following:

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

FullPathScopeName

The full path of the hierarchy, except the signal name.

SignalName

The name of the signal.

ScopeSeparator

The separating character used in FullPatchScopenName.

Type

The waveform type.

Return Values**WaveHdl**

The signal handle.

-1

An error occurred.

Description

CreateWave creates a new waveform handle. The handle is used by the following APIs to print the value when a value change occurs to this signal waveform handle.

Example

In this example, SignalType is defined as follows:

```
enum SignalType {  
    VOLTAGE=0,  
    CURRENT,  
    LOGIC,  
    POWER,  
    DEFAULT  
};
```

Note: LOGIC type is not available to DC or AC analysis.

EndCreateWave

Can be used at the end of creating wave.

Syntax

For Transient Analysis

```
int out_format_EndCreateWave (void *Hdl);
```

For DC Analysis

```
int out_format_EndCreateWaveDC (void *Hdl);
```

For AC Analysis

```
int out_format_EndCreateWaveAC (void *Hdl);
```

Description

EndCreateWave may be used at the end of creating wave. If no special setup takes place before the new waveform is created, EndCreateWave only needs to return 0 (zero).

Parameters

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

Return Values

0

No errors occurred.

-1

An error occurred.

Create a Duplicate Waveform Into a Waveform File

CreateDuplWave is used to create a duplicate waveform and insert it into a waveform file for transient analysis. The values of the duplicate wave form are copied from a master wave to which it is aliased.

CreateDuplWave

Creates a duplicate waveform.

Syntax

For Transient Analysis

```
void * WaveHdl=void * out_format_CreateDuplWave (
    void *Hdl,
    char *FullPathScopeName,
    char *SignalName,
    char ScopeSeparator,
    enum SignalType Type
    void *MasterWave);
```

Parameters

The input parameters include the following:

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

FullPathScopeName

The full path of the hierarchy, except the signal name.

SignalName

The name of the signal.

ScopeSeparator

The separating character used in FullPatchScopenName.

Type

The waveform type.

MasterWave

The MasterWave provides the values for creating a duplicate wave.

Return Values

WaveHdl

The signal handle.

-1

An error occurred.

Appendix B: Custom Output Interface
Add Waveform Values Into a Waveform File**Description**

CreateDuplWave is used to create a duplicate waveform and insert it into a waveform file for transient analysis. The values of the duplicate wave form are copied from a master wave to which it is aliased.

Add Waveform Values Into a Waveform File

This section provides examples for adding digital and analog values into a waveform file for transient analysis, and for adding analog values into a wave file for DC and AC analysis.

AddNextDigitalValueChange

Stores digital data of a waveform into a waveform file.

Syntax

```
int out_format_AddNextDigitalValueChange (
    void *Hdl,
    void *WaveHdl,
    double Time,
    int Value
);
```

Parameters

The input parameters include the following:

Hdl

The handle returned from the CreateWaveFile()

WaveHdl

The handle returned from the CreateWave() for transient analysis, CreateWaveDC() for DC analysis, or CreateWaveAC() for AC analysis.

Time

The current simulation time

b

The waveform data at the current time

Return Values

0

No errors occurred.

-1

An error occurred.

Description

AddNextDigitalValueChange stores digital data of a waveform into a waveform file.

AddNextAnalogValueChange

Stores analog values of a waveform into a waveform file.

Syntax**For Transient Analysis**

```
int out_format_AddNextAnalogValueChange (
    void *Hdl,
    void *WaveHdl,
    double Time,
    double X_Value
);
```

For DC Analysis

```
int out_format_AddNextAnalogValueChangeDC (
    void *Hdl,
    void *WaveHdl,
    double Time,
    double X_Value
);
```

For AC Analysis

```
int out_format_AddNextAnalogValueChangeAC (
    void *Hdl,
    void *WaveHdl,
    double Time,
    double X_Value
);
```

Feedback

Appendix B: Custom Output Interface

End the Waveform File Process

Parameters

The input parameters include:

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

WaveHdl

The handle returned from the CreateWave() for transient analysis, CreateWaveDC() for DC analysis, or CreateWaveAC() for AC analysis.

Time

The current simulation *x_Value*.

Value

The waveform value at the current *x_Value*.

Return Values

0

No errors occurred.

-1

An error occurred.

Description

AddNextAnalogValueChange stores analog values of a waveform into a waveform file.

End the Waveform File Process

This section provides an example for closing a waveform.

CloseWaveFile

Closes a waveform file at the end of the simulation.

Syntax

For Transient Analysis

```
int out_format_CloseWaveFile(void *Hdl);
```

For DC Analysis

```
int out_format_CloseWaveFileDC(void *Hdl);
```

Description

CloseWaveFile closes a waveform file at the end of the simulation.

Parameters

Hdl

The handle returned from the CreateWaveFile() for transient analysis, CreateWaveFileDC() for DC analysis, or CreateWaveFileAC() for AC analysis.

Return Values

0

No errors occurred.

-1

An error occurred.

Building a Waveform File Generator Library

This section provides an example waveform file generator library. The sample code generates a MYFORMAT waveform file format.

The following files must be prepared for the MYFORMAT waveform file generator library:

- myformat.c: Source file of the waveform file generator library.
- myformat.h: Header file of the waveform file generator library.
- makefile: Makefile for building the file generator library.

To build this library, enter make as shown in [Example 80](#).

Appendix B: Custom Output Interface
Building a Waveform File Generator Library*Example 80*

```
<<< myformat.h >>>

/* Copyright 1998 - 2004 */
/* Synopsys Corporation */
/* */

#ifndef MYFORMAT_H
#define MYFORMAT_H
#ifdef WIN32
#include <windows.h>
#endif COI_EXPORTS
#define COI_API __declspec(dllexport) extern
#define COI_API __declspec(dllexport) extern
#define COI_API __declspec(dllexport) extern
#else
#define COI_API __declspec(dllimport) extern
__declspec(dllexport) void *MYFORMAT_CreateWaveFile (char
*,HsimParam*);
__declspec(dllexport) int MYFORMAT_CreateHeader (void *);
__declspec(dllexport) int MYFORMAT_BeginCreateWave (void *);
__declspec(dllexport) int MYFORMAT_EndCreateWave (void *);
__declspec(dllexport) void *MYFORMAT_CreateWave (void *, char
*,char *, char, enum SignalType);
__declspec(dllexport) void *MYFORMAT_CreateDuplWave (void *, char
*,char *, char, enum SignalType, void *MasterWave);
__declspec(dllexport) int MYFORMAT_AddNextDigitalValueChange
(void *, void *, double time, int);
__declspec(dllexport) int MYFORMAT_AddNextAnalogValueChange
(void *, void*, double, double);
__declspec(dllexport) int MYFORMAT_CloseWaveFile (void *);
#endif
#else
#define COI_API extern
#endif
#endif

<<< myformat.c >>>
/* */
/* Copyright 1998 - 2004 */
/* Synopsys Corporation */
/* */
#include "coi.h"
#include <stdio.h>
#include "myformat.h"
static int numOfWave=0;
int *myFileDialog;
int *myWaveId;
void *
```

```
MYFORMAT_CreateWaveFile(char *BaseName, struct HsimParam
*hsimparam)
{
fprintf(stdout, "MYFORMAT/CreateWaveFile\n");
fprintf(stdout, "BaseName=%s\n", BaseName);
fprintf(stdout, "StopTime=%g\n", hsimparam->StopTime);
fprintf(stdout, "Temperature=%g\n", hsimparam->Temperature);
myFileId=(int *)malloc(sizeof(int));
*myFileId=1;
return(myFileId);
}
int
MYFORMAT_CreateHeader(void *Hdl)
{
int *hdl=(int *)Hdl;
fprintf(stdout, "MYFORMAT/CreateHeader\n");
fprintf(stdout, "Hdl=%d\n", *hdl);
return(0);
}
void *
MYFORMAT_CreateWave(void *Hdl, char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type)
{
int *hdl=(int *) Hdl;
fprintf(stdout, "MYFORMAT/CreateWave\n");
fprintf(stdout, "Hdl=%d\n", *hdl);
fprintf(stdout, "FullPathScopeName=%s\n", FullPathScopeName);
fprintf(stdout, "SignalName=%s\n", SignalName);
fprintf(stdout, "ScopeSeparator=%c\n", ScopeSeparator);
fprintf(stdout, "Type=%s\n", (Type==VOLTAGE) ?VOLTAGE: ((Type==CURRENT) ?CURRENT:LOGIC));
myWaveId=(int *)malloc(sizeof(int));
*myWaveId=numOfWave++;
return(myWaveId);
}
void *
MYFORMAT_CreateDuplWave(void *Hdl,
char *FullPathScopeName,
char *SignalName,
char ScopeSeparator,
enum SignalType Type,
void *MasterWave)
{
int *hdl=(int *) Hdl;
fprintf(stdout, "MYFORMAT/CreateDuplWave\n");
fprintf(stdout, "Hdl=%d\n", *hdl);
```

Appendix B: Custom Output Interface
Building a Waveform File Generator Library

```
fprintf(stdout, "FullPathScopeName=%s\n",FullPathScopeName) ;
fprintf(stdout, "SignalName=%s\n",SignalName) ;
fprintf(stdout, "ScopeSeparator=%c\n",ScopeSeparator) ;
fprintf(stdout, "Type=%s\n", (Type==VOLTAGE)?VOLTAGE:(Type==CURRENT)?CURRENT:LOGIC));
myWaveId=(int *)malloc(sizeof(int)) ;
*myWaveId=numOfWave++;
return(myWaveId) ;
}

int
MYFORMAT_BeginCreateWave(void *Hdl)
{
int *hdl;
hdl=(int *)Hdl;
fprintf(stdout,"MYFORMAT/BeginCreateWave\n");
fprintf(stdout,"Hdl=%d\n",*hdl);
return(0);
}
int
MYFORMAT_AddNextDigitalValueChange(void *Hdl,
void *WaveHdl,
double Time,
int Value)
{
int *hdl1=(int *) Hdl;
int *hdl2=(int *) WaveHdl;
fprintf(stdout,"MYFORMAT/AddNextDigitalValueChange\n");
fprintf(stdout,"Hdl=%d\n",*hdl1);
fprintf(stdout,"WaveHdl=%d\n",*hdl2);
fprintf(stdout,"Time=%g\n",Time);
fprintf(stdout,"Value=%d\n",Value);
return(0);
}
int
MYFORMAT_AddNextAnalogValueChange(void *Hdl,
void *WaveHdl,
double Time,
double Value)
{
int *hdl1=(int *) Hdl;
int *hdl2=(int *) WaveHdl;
fprintf(stdout,"MYFORMAT/AddNextAnalogValueChange\n");
fprintf(stdout,"Hdl=%d\n",*hdl1);
fprintf(stdout,"WaveHdl=%d\n",*hdl2);
fprintf(stdout,"Time=%g\n",Time);
fprintf(stdout,"Value=%g\n",Value);
return(0);
}
```

```
}

int
MYFORMAT_CloseWaveFile(void *Hdl)
{
    int *hdl=(int *)Hdl;
    fprintf(stdout,"MYFORMAT/CloseWaveFile\n");
    fprintf(stdout,"Hdl=%d\n",*hdl);
    return(0);
}
int
MYFORMAT_EndCreateWave(void *Hdl)
{
    int *hdl=(int *)Hdl;
    fprintf(stdout,"MYFORMAT/EndCreateWave\n");
    fprintf(stdout,"Hdl=%d\n",*hdl);
    return(0);
}

<<< makefile >>>

#!/bin/sh
=====
BIN=.
=====
# --- Makefile rules
# -c C program
# -g Debug option
# -O Optimization
=====
CC=cc -fPIC -c -g -O -D_DEBUG
LINK=cc -o
SHARED=ld -G -o
=====
# --- libraries used for linking
SYSLIBS=-lc -ldl
=====
# -- Make the executable program
default:
    make libMYFORMAT.so
=====
myformat.o : myformat.c myformat.h
    $(CC) myformat.c
libMYFORMAT.so : myformat.o
    $(SHARED) $(BIN)/libMYFORMAT.so myformat.o
```

Feedback

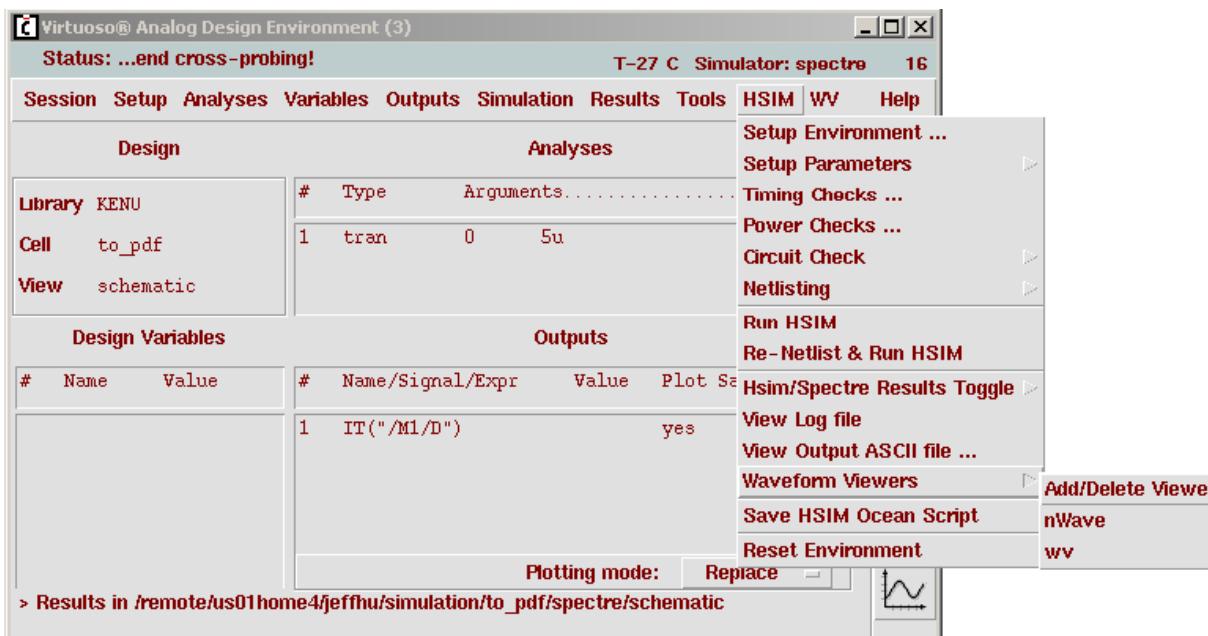
Appendix B: Custom Output Interface

Building a Waveform File Generator Library

Waveform Viewer Customization

This appendix provides details on the customer features in nWave and WaveView Analyzer.

When installed, the Waveform Viewers option is displayed in the HSIM menu:



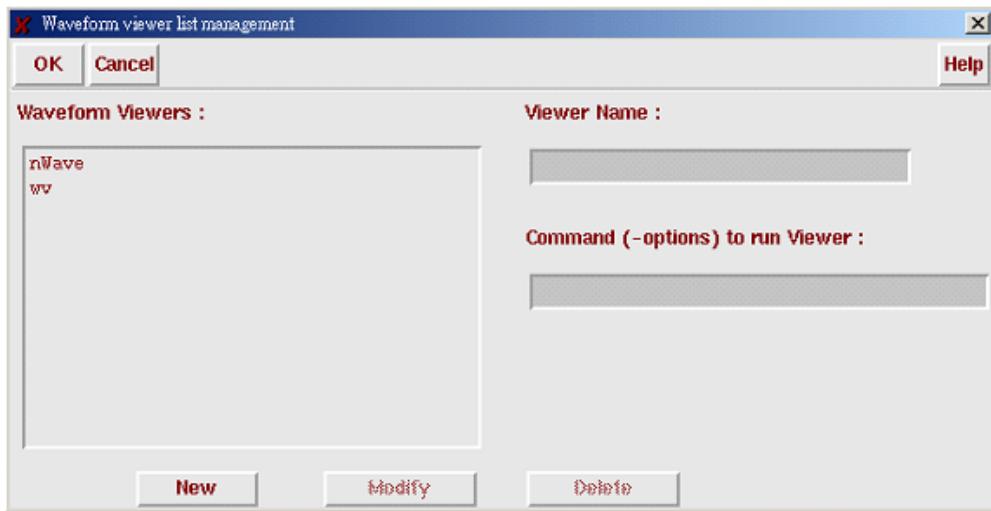
© 2006, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 34 Waveform Viewers Menu

nWave and wv are listed in the waveform viewer list by default. The list is customizable by using the Add/Delete Viewer option. Click on Add/Delete Viewer and a customization form pops up as shown in [Figure 35 on page 746](#).

Appendix C: Waveform Viewer Customization

New



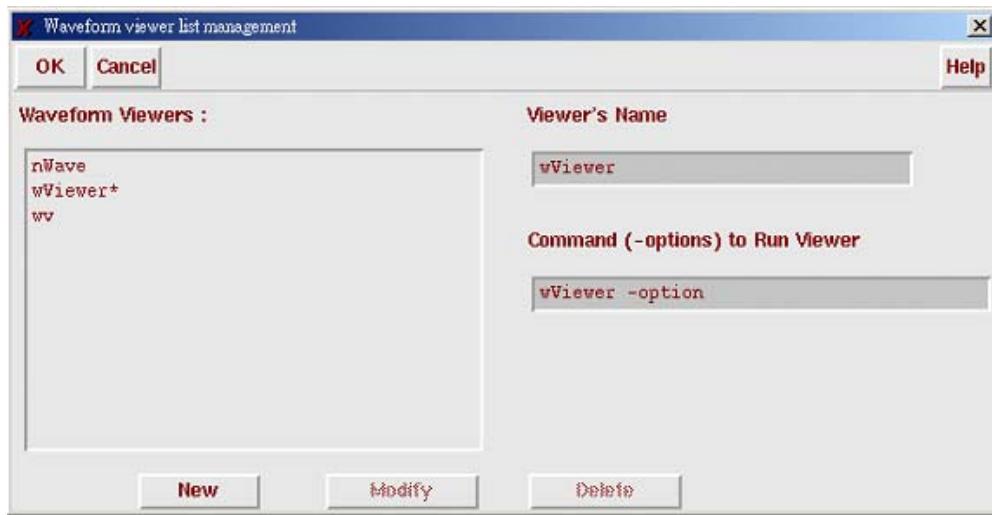
© 2006, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 35 Waveform Viewer List Management Screen

In the beginning, only the [New] button is activated and both the Viewer name and the Command (-options) to run Viewer fields are disabled. A new waveform viewer can be added to the list by pressing the [New] button or by selecting an existing waveform viewer and modifying its commands and options.

New

Clicking the [New] button makes both the Viewer Name and Command option fields editable and transforms the [New] button into the [Add] button. After entering the new viewer's name, pressing the [Add] button adds the new viewer to the list as shown in [Figure 36 on page 747](#).



© 2006, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 36 Waveform Viewer List

Note: The asterisk * after the name indicates the viewer is newly added but not activated yet. Newly added viewers will not be activated and can not be invoked from the drop down menu under HSIM until the next Virtuoso session is opened.

Modify, Delete

Viewer functions are affected by using the following steps:

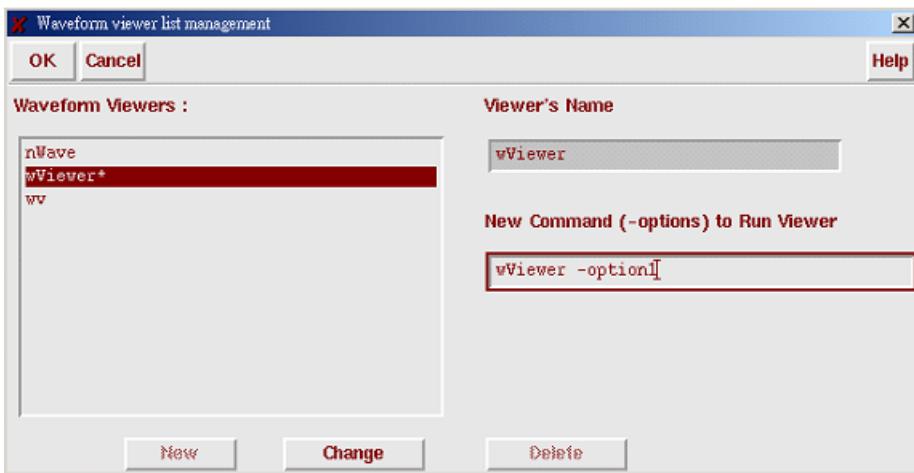
- Select one of the existing viewers from the list to display the viewer's settings.
- Press the [Modify] and [Delete] button to disable the Viewer's Name and Command (-options) fields.
- Press the [Delete] button to remove the selected viewer from the list.
- Press the [Modify] button to make the Command (-options) field editable and change the label text. Additionally, [Modify] button transforms into the [Change] button and the [New] and [Delete] buttons are disabled.
- Once the command field is modified, press the [Change] button to complete the process.

Feedback

Appendix C: Waveform Viewer Customization

OK

The Viewer's Name can not be modified. To change the viewer name the current viewer must first be deleted and a new viewer added as shown in [Figure 37 on page 748](#).



© 2006, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

*Figure 37 Selecting wViewer**

OK

Clicking the [OK] button updates the Waveform Viewers list and saves it to the .waveViewerTable file. The modification does not immediately become effective in the HSIM->Waveform Viewers menu. The change will becomes valid in the next Virtuoso Analog Design Environment session.

.waveViewerTable

HSIM->Waveform Viewers content is stored in the .waveViewerTable file. At start up, Native Netlist Integration first looks for this file and read its content using the following rules:

Rule 1

Search the current working directory, if there is one available, and use that directory.

Rule 2

If the current working directory is not available, search your home directory, if there is one available, and use that directory.

Rule 3

If neither the current working directory or your home directory are available, Native Netlist Integration searches \$HSIM_ARTISTIF/install/.waveViewerTable and use that directory.. If this file is read and a user made changes to the list, then a new .waveViewerTable file will be created under user's home directory and this file will be used in the following sessions.

The \$HSIM_ARTISTIF/install/.waveViewerTable file is distributed through the Native Netlist Integration package. The system administrator normally has authority to customize this file via the GUI interface or manually and permits all users share the same file from a central location.

Two reference files are provided in the \$HSIM_ARTIST_IF/install directory:

- .waveViewerTable_nWave_WaveView: This file directly accesses nWave and WaveView. It is the default for .waveViewerTable. Deleted items may also be added into the list and saved using the GUI form.
- .waveViewerTable_empty: Only allows a viewer to be added or deleted by copying the preferred viewer into .waveViewerTable.

Note: It is possible to toggle between the two settings at any time.

Rule 4

If the .waveViewerTable file is not available using Rules 1 through 3, Native Netlist Integration automatically creates a .waveViewerTable in the home directory. The default .waveViewerTable contains the following menu items contained in the HSIM drop down menu:

Feedback

Appendix C: Waveform Viewer Customization

.waveViewerTable

- Add/Delete Viewer
- nWave
- WV

HSIM/Eldo Compatibility

Provides information on HSIM's support for Mentor Graphic's Eldo SPICE simulator.

- Starting HSIM in the Eldo Mode
- Transistor Model Compatibility
- HSIM-Supported Eldo-Specific Syntax
- Eldo Syntax Not Supported by HSIM
- Eldo Commands Partially or Fully Supported by HSIM
- HSIM-Supported Eldo Options
- Eldo Commands Not Supported in HSIM
- HSIM Supported Eldo Macromodels

Starting HSIM in the Eldo Mode

This Appendix provides information on HSIM's support for Mentor Graphic's Eldo SPICE simulator. The information is provided as a guide to known Eldo features only. This appendix does not cover Eldo features that may exist but are not documented, or functions that will be introduced in future releases.

To use HSIM in the Eldo mode, use the following syntax:

```
Hsim -eldo my_netlist.cir
```

Note: HSIM always considers a GND=0. Eldo does not default to GND=0.

Appendix D: HSIM/Eldo Compatibility

Transistor Model Compatibility

Note: All Eldo syntax equations must be defined between curly brackets {}. Refer to the Mentor Graphics Eldo documentation for specifics.

Transistor Model Compatibility

[Table 103](#) lists the HSIM/ELDO transistor model compatibility.

Table 103 HSIM/ELDO Transistor Model Compatibility

Transistor Model	HSIM	Eldo
BSIM3	Level = 53 or 49	Level = 53
BSIM4	Level = 60	Level = 60
MM9	Level = 50	Level = 50
BSIM3SO	Level = 57	Level = 55
EKV2.6	Level = 55	Level = 44

Note: ST proprietary devices such as diodes, resistors, capacitors, MOSFETs, and Bipolar device models are integrated into HSIM to support Bipolar CMOS DMOS (BCD) technologies. These components use the same level as Eldo, but the stver option is required in the netlist.

HSIM-Supported Eldo-Specific Syntax

[Table 104](#) lists the HSIM-supported ELDO-specific syntax definitions.

Table 104 HSIM Supported Eldo-Specific Syntax Definitions

Command	Functional Description
!	Defines the end of a comment line(s).

Table 104 HSIM Supported Eldo-Specific Syntax Definitions (Continued)

Command	Functional Description
TEMPER	The variable used in equations to indicate the simulation temperature specified by .temp. The HSIM equivalent TEMP.
FREQ	The variable used to indicate current frequency. When FREQ=1 for DC and TRAN simulations, it reflects the current frequency during AC analysis.
#com	Defines the beginning of a comment line(s).
#endcom	Defines the end of a comment line(s).
Yxx	Defines lossy transmission lines in Eldo as W elements in HSIM.
Yxx	Verilog-A model instantiations are supported with the same syntax as Eldo.

Eldo Syntax Not Supported by HSIM

[Table 105](#) lists the Eldo syntax not supported by HSIM.

Table 105 Eldo syntax Not Supported by HSIM

Command	Functional Description
Jxx	MESFET
NOISE	Any NOISE sources
Yxx	No Specific Eldo macro-models (Analog, Digital, Amplifier, converters, switch capacitor models, etc.)

Appendix D: HSIM/Eldo Compatibility

Eldo Commands Partially or Fully Supported by HSIM

Eldo Commands Partially or Fully Supported by HSIM

Table 106 lists the Eldo commands partially or fully supported by HSIM.

Table 106 Eldo Commands Partially or Fully Supported by HSIM

Command	Function
.AC	Full support
.ALTER	Generalized re-run facility. HSIM will create only one .log file and one output waveform file per run. These are named sequentially for all runs with the following extension(s): a0, a1..., aN.
.CHRENT	Piecewise linear source
.CHECKSOA	Check safe operating area limits
.CONNECT	Connects two nodes. HSIM only accepts .connect between a voltage/current source and a node.
.CONSO	Current used by a circuit
.DATA	Parameter sweep
.DC	DC analysis
.DEFMAC	Macro definition
.DEFWAV	Waveform definition
.DEFMOD	Define a model alias
.DEL	Remove a library name
.DSPF_INCLUDE	Include a DSPF file
.END	End netlist
.ENDL	End a library variant description
.ENDS	End a subcircuit description

Table 106 Eldo Commands Partially or Fully Supported by HSIM (Continued)

Command	Function
.EXTRACT	Extract waveform characteristics
.HIER	Changing the hierarchy separator
.FOUR FFT	Select waveform
.GLOBAL	Global node allocation
.INCLUDE	Include a file in an input netlist
.IC	Initial transient analysis conditions
.LIB	Insert circuit information from a library file
.MC	Monte Carlo analysis
.MCMOD LOT & DEV	Variation specification on model parameters
.MEAS	Measure waveform characteristics
.MODEL	Device model description
.NODESET DC	Analysis conditions
.OP	DC operating point calculation
.OPTION	Simulator configuration
.OPTFOUR	FFT post-processor options
.PARAM	Global declarations
.PLOT	Plotting of simulation results
.PRINT	Prints results. Same behavior as .plot command. Beware it is different than Eldo.
.PROBE	Output short form. Same behavior as .plot command.
.probe VTOP	Displays all the top level node voltages.
.probe VN	Probe only node names which are not numbers

Appendix D: HSIM/Eldo Compatibility

HSIM-Supported Eldo Options

Table 106 Eldo Commands Partially or Fully Supported by HSIM (Continued)

Command	Function
.probe vi	HSIM-supported
.probe isub	HSIM-supported
.SETBUS	Create bus
.SETSOA	Set safe operating area
.SIGBUS	Set bus signal
.STEP	Parameter sweep (supported only with .TRAN)
.SUBCKT	Sub-circuit definition
.TEMP	Set circuit temperature
.TOPCELL	Select the TOP cell subcircuit
.TRAN	Transient analysis. In HSIM, the first parameter is ignored, possibility to have a .tran sweep (see documentation)

HSIM-Supported Eldo Options**Table 107** lists the HSIM-supported Eldo options.*Table 107 HSIM-Supported Eldo Options*

Command	Function
.OPTION TNOM=val	Sets the model temperature.
.OPTION	Model for binning parameters
.OPTION SCALE=va	Multiplier for MOSFET w, l, as and ad
.OPTION SEARCH=va	Search path definition for subckt files
.OPTION HISTAUMAX=val	Refer to HSIMTAUMAX on page 176

Table 107 HSIM-Supported Eldo Options (Continued)

Command	Function
.OPTION HMIN=val	Refer to HSIMTIMESCALE on page 177
.OPTION FREQSMP=val	Refer to HSIMOUTPUTTSTEP on page 122
.OPTION TUNING=val	Refer to HSIMSPEED on page 150
.OPTION DSCGLOB=global	Works the same as in Eldo

Eldo Commands Not Supported in HSIM

[Table 108](#) lists the Eldo commands not supported in HSIM.

Table 108 Eldo Commands Not Supported in HSIM

Command	Function
.A2D	Analog-to-digital converter
.ADDLIB	Insert a model or sub circuit file
.CALL_TCL	Call a TCL program
.CHECKBUS	Check bus values. This command can be replaced by an automatic output check in the digital vector file used by HSIM.
.CHRSIM	Input from a prior simulation. This command can be replaced by tools such as: cou2fsdb.
.COMCHAR	Define the new comment character
.D2A	Digital to analog converter
.DISTRIB	User-defined distributions (Monte Carlo Analysis)
.DSP	Digital signal processing computation
.DSPMOD	Power spectral density computation
.FFILE S, Y, Z	Parameter output file specification

Appendix D: HSIM/Eldo Compatibility

Eldo Commands Not Supported in HSIM

Table 108 Eldo Commands Not Supported in HSIM (Continued)

Command	Function
.GUESS	Initial DC analysis conditions
.INIT	Initial digital circuit conditions
.LIBFAS	Specification of a library containing FAS models
.LOAD	Use previously simulated results
.LOOP	Insert a feedback loop
.LSTB	Loop stability analysis
.LOTGROUP	Share distributions
.MODDUP	Aspire/SimPilot command
.MODLOGIC	Digital model definition
.MPRUN	Multiprocessor simulation
.NET	Network analysis
.NEWPAGE	Control page layout for Xelga
.NOCOM	Suppress comment lines from output file
.NOISE	Noise analysis
.NOISETRAN	Transient noise analysis
.NOTRC	Suppress netlist from an output file
.NWBLOCK	Partition netlist into Newton blocks
.OPTPWL	Accuracy by time window
.OPTWIND	Accuracy by time window
.OPTIMIZE	Circuit optimization
.OPTNOISE AC	Noise analysis

Table 108 Eldo Commands Not Supported in HSIM (Continued)

Command	Function
.PLOTBUS	Plotting of bus signals
.PROTECT	Netlist protection
.PZ	Pole & zero analysis
.RAMP	Automatic ramping
.RESTART	Restart simulation. The command in HSIM is .restore with the HSPICE syntax
.SAVE	Save simulation run. The command in HSIM is .store with the HSPICE syntax
.SENS	Sensitivity analysis
.SINUS	Sinusoidal voltage source
.SNF	Spot noise figure
.SOLVE	Sizing Facility. Identical function in HSIM, but with the HSPICE syntax.
.SUBDUP	Sub-circuit duplicate parameters
.TABLE	Value tables
.TITLE	Set title of cou file
.TF	Transfer function
.TVINCLUDE	Test vector files
.UNPROTECT	Netlist protection
.USE	Use previously simulated results
.USE_TCL	Load a TCL file
.WCASE	Worst case analysis
.WIDTH	Set printer paper width

Appendix D: HSIM/Eldo Compatibility

HSIM Supported Eldo Macromodels

HSIM Supported Eldo Macromodels

Table 109 lists the Eldo macromodels when PVA is activated by default:

Table 109 Supported Eldo Macromodels

Macromodel	Function
COMP	Comparator (Single Output)
COMPD	Comparator (Differential Output)
DIF	Differentiator
DIV	Divider
MULT	Multiplier
ADD	Adder
DEL	Delay
ADC1...ADC32	Analog to Digital Converter
OPAMP0	Op-amp (Linear) (Single Output)
OPAMP0D	Op-amp (Linear) (Differential Output)
OPAMP1	Op-amp (Linear 1-pole) (Single Output)
OPAMP1D	Op-amp (Linear 1-pole) (Differential Output)
OPAMP2	Op-amp (Linear 2-pole) (Single Output)
OPAMP2D	Op-amp (Linear 2-pole) (Differential Output)

The unified syntax for Eldo Macromodels is supported:

```
Yxx NAME [PIN:] P1 P2 ... [[PARAM:] K1=val1 K2=val2 ...] [MODEL:  
NMANE]
```

For example:

```
Y1 MULT PIN: in2_1 in2_2 out_mult PARAM: VMAX=12 VMIN=0 VSATP=9  
VSATN=1 PSLOPE=1 NSLOPE=1
```

E

HSIM-ADMS Integration

Please contact Synopsys support for documentation on the single-kernel integration of HSIM into Mentor's multilanguage AAdvanced Mixed-signal Simulator(ADMS).

Feedback

Appendix E: HSIM-ADMS Integration

HSIM-Virtuoso Analog Design Environment Interface

This chapter describes how to integrate HSIM into the Cadence® Virtuoso® Analog Design Environment using the HSIM-Virtuoso Interface. The approaches described in this chapter can be tightly coupled to a Cadence database that requires intensive data preparation or they can be loosely integrated while maintaining the signal cross probing functionality.

Note: Cadence's Analog Artist product is now known as Virtuoso Analog Design Environment, so the HSIM interface is now referred to as the HSIM-Virtuoso Interface, or for simplification purposes, the Interface. In previous releases, the interface was referred to as the Analog Artist Interface (AAI).

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

Synopsys provides an All-In-One Package that permits installation of one or multiple installation approaches listed below:

- AANNI (Native Netlist Integration)
- AAI HSIM (HSIM Socket Interface)
- AAI HSIMD (HSIM Direct Interface)
- AACoSim (CoSim Integration)

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

The HSIM Virtuoso Analog Design Environment Interface has been developed on Cadence Design Framework II 4.4.5. and has been tested on the following versions:

4.4.5

4.4.6

5.0.0

5.1.41

The Interface is supported on the following platforms:

- Solaris
- HPUX
- Linux

All-In-One Package (AAIM)

AAIM Installation & Setup

Synopsys provides an installation tar file. The tar file can be used to install any or all of the Interface options: AANNI, AAI HSIM Socket, AAI HSIMD, and/or AACoSim.

To install the All-In-One Package, use the following steps:

1. Obtain the tar file.

Obtain the following installation tar file from the Synopsys ftp site:

AAIM-<version>-mmddyear.tar.gz

2. Set up HSIM_ARTISTIF.

Unzip and untar the tar file then, set the following environment variable:

```
setenv HSIM_ARTISTIF <AAI_installation_dir>/AAIM-<version>-mmddyear
```

3. Add the nsdaAAIMPkgList and load statements.

Add the following two statements in the .cdsinit file:

```
nsdaAAIMPkgList='("hsimD" "hsim" "AACoSim" "AANNI")
load("<AAI_installation_dir>/AAIM-<version>-mmddyear/
install/nsdaAAIMInvoker.il")'
```

Important: nsdaAAIMPkgList must be entered as shown and placed prior to the load statement.

All of the package(s) specified in nsdaAAIMPkgList are installed. The syntax shown above installs the following:

- AANNI
- AAI HSIM
- AAI HSIMD
- AACoSim

To add the Sandwork's WaveView Analyzer link to the Interface, add WV in the nsdaAAIMPkgList statement, as shown in the following syntax:

```
nsdaAAIMPkgList='("hsimD" "hsim" "AACoSim" "AANNI" "WV")
```

AANNI is the default package if nsdaAAIMPkgList is an empty list, as shown in the following syntax:

```
nsdaAAIMPkgList='()
```

The menu files are based on the specific integration approaches as shown in the following:

- AACoSim: spectreVerilog.menus, spectreSVerilog.menus
- AANNI: simui.menus
- AAI HSIM: hsim.menus
- AAI HSIMD: hsimD.menus

All the menu files are located under the AAIM-<version>-*mmddyear*/menus subdirectory.

4. Locate the menu file subdirectory.

Locate the menu file in user's working environment. It can be either of the following directories:

- <user_working_dir>/menus
- <user_home_dir>/menus

5. Merge the menu files.

The provided menu file can be merged with the menu file in the Cadence tree, as shown in the following example:

```
<CDS_install_dir>/tools/dfII/etc/tools/menus/simui.menus
```

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

6. Set up the executable.

Set the Interface to point to the appropriate HSIM executable as shown in the following command syntax:

```
setenv HSIM_HOME <your_HSIM_installation_dir>
```

7. Obtain an Interface license.

Obtaining the hmartld license daemon is required before conducting any interface flows. The license daemon resides in the HSIM release tree such as \$HSIM_HOME/bin/hmartld. Without the Interface license daemon properly located, the following error message will be displayed in the Cadence central information window and the Interface flow will not proceed.

```
.....
Checking out Synopsys license ...
*Error* ipcWriteChild: Attempt to write to expired process -
ipc:-1
.....
```

AAIM Uninstallation

To uninstall any part of the AAIM software, delete any unwanted software package name in the nsdaAAIMPkgList statement as shown in the following syntax example:

```
nsdaAAIMPkgList='("hsim" "AANNI")'
```

Note: Using the syntax above, only AAI HSIM socket and AANNI are installed. AAI HSIMD and AAICoSim will not be installed.

Native Netlist Integration (AANNI)

Native Netlist Integration Installation & Setup

This section describes how to use the HSIM-Virtuoso Interface to integrate HSIM's simulation engine into the Virtuoso Analog Design Environment. The Native Netlist Integration software has a different integration approach than traditional approaches such as HSIM socket and HSIMD direct. The key features are described below.

Native Netlist Integration Features

Native Netlist Integration has the following features:

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

- HSIM parameters and command line options specification.
- Top-level netlist creation.
- HSIM simulation.
- Browse and view ASCII output files.
- Signal cross-probing and waveform display.
- Save/load state.
- Does not conduct design host netlisting.
- Does not require a special view and simInfo from device libraries.
- Does not require an additional Cadence license for operation.

Basic Native Netlist Integration Flow

[Figure 38](#) displays the typical Native Netlist Integration process flow.

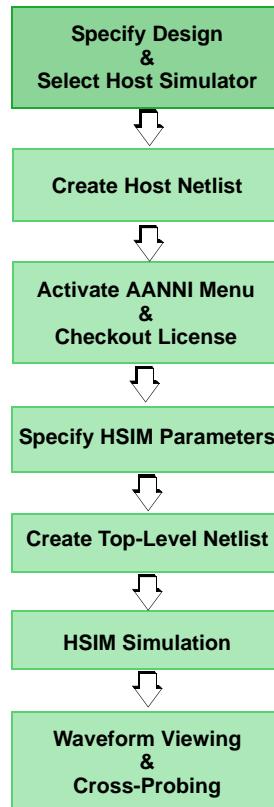


Figure 38 Process Flow

Porting the Existing Design

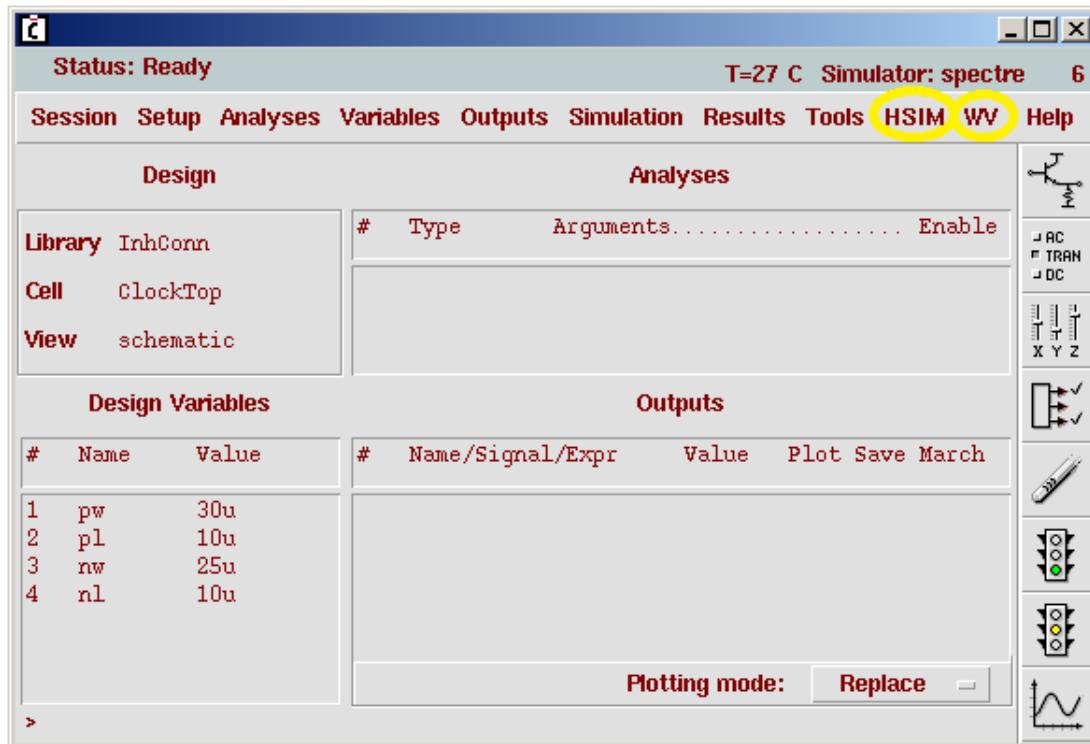
Since Native Netlist Integration does not conduct host netlist creation nor require special view and simInfo from the device libraries, there are no major porting issues for existing designs when migrating from traditional socket and direct integration approaches.

Native Netlist Integration Window and Pull-Down Menus

With Native Netlist Integration properly setup and installed in the Virtuoso Analog Design Environment window shown in [Figure 39](#), the pull-down menu appears. It is located on the RIGHT hand side of the form. Also, if WV is also specified in the nsdaAAIMPkgList statement, there will also be a WV menu next to the menu header.

Caution: The window may need to be enlarged to view the menu if there are a large number of existing pull-down menus displayed from the local environment setup.

Appendix F: HSIM-Virtuoso Analog Design Environment Interface
HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



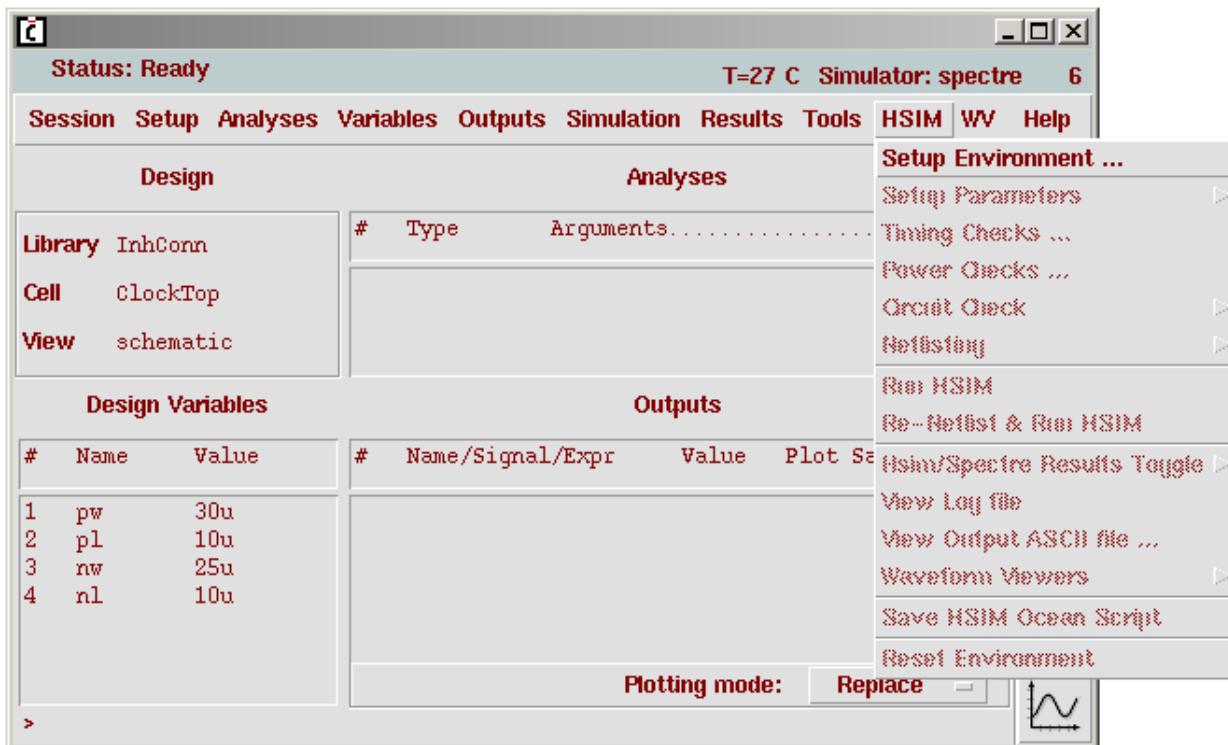
© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 39 HSIM-Virtuoso Design Environment Interface Window

Initially, all items in the menu are disabled and appear as grayed out text as shown in [Figure 40](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 40 HSIM-Virtuoso Design Environment Interface Window & Pull-down Menu

A design and designated host simulator such as spectre or hspice must be specified before the pull-down menu commands become active. If they are not specified, a warning message is displayed as shown in [Figure 41](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

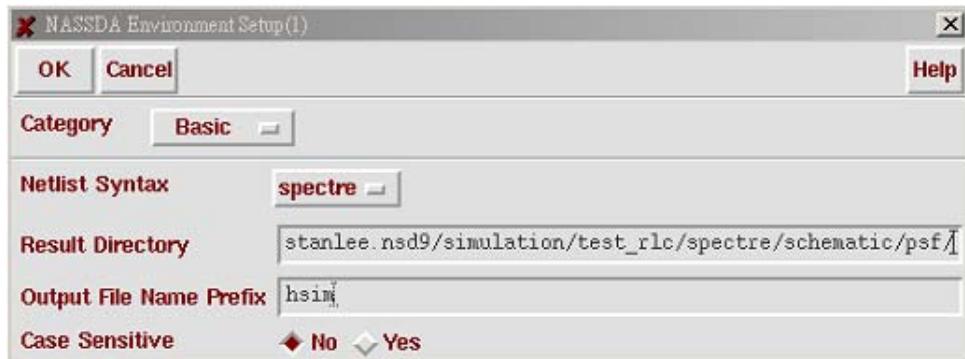


© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 41 Warning Message Window

Environment Setup

After specifying a design and simulator, select Setup Environment in the pull down menu. The Environment Setup(1) window containing the Environment Options form shown in [Figure 42](#) will appear.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 42 Setup Options Form Window

Basic Setup

The options displayed on the basic form include the following:

- [Basic]: If the [Basic] screen button is displayed, the setup form only displays the most frequently used buttons and entry fields.
- Netlist Syntax: Indicates what kind of netlist syntax the host netlister is going to generate.

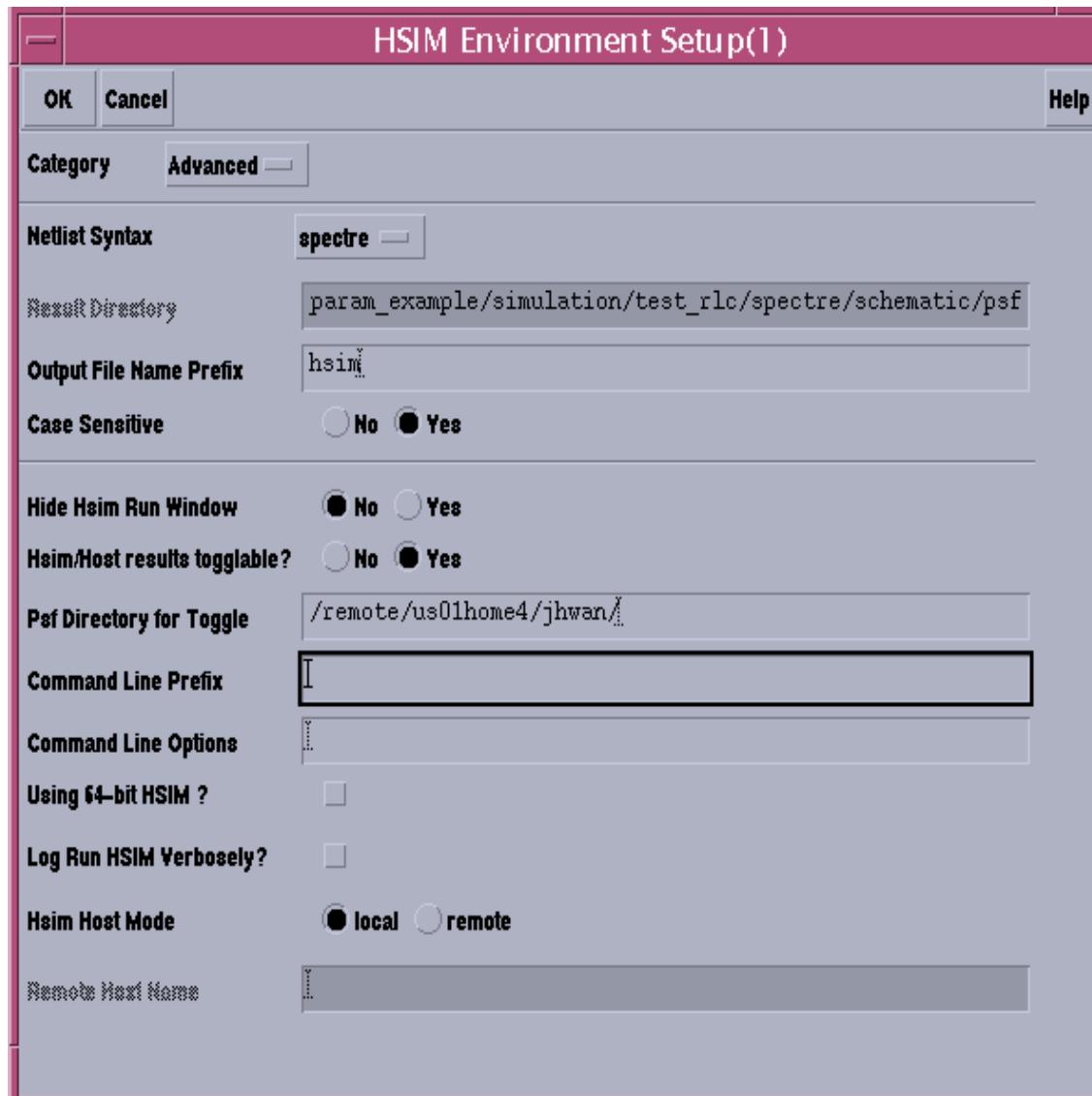
Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

- Result Directory: Allows user to specify directory for storing output data.
- Output File Name Prefix: Is equivalent to the -o option of HSIM,
- Case Sensitive: Is equivalent to the -case option of HSIM

Advanced Setup

To perform more advanced environment setups, click on the [Basic] screen button and select Advanced from the options displayed. The form enlarges and provides additional setup options as shown in [Figure 43](#).



© 2007, Cadence Design Systems, Inc. All rights reserved
worldwide. Printed with permission.

Figure 43 Advanced Setup Options Form Window

The advanced options displayed on this form include the following:

- **HSIM/Host results togglable?:** The default is Yes, which allows AANNI to preserve a copy of the HSIM simulation result in case you would like to conduct HSIM-based or host simulator-based waveform probing/viewing. The default location for the preserved HSIM simulation copy is your home

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

directory. You can change the location through the "Psf Directory for Toggle" entry field. Refer to [Toggle between Spectre and HSIM Simulation Database for Waveform Probing on page 780](#) for more details.

- Command Line Prefix: Whatever specified in this field will be attached in the beginning of the HSIM command line.
- Command Line Options: Allows user to specify additional HSIM command line options if applicable.
- Using 64-bit HSIM?: If turned on, it is equivalent to set the following in your working environment: `setenv HSIMPLUS_64 1`.
- HSIM HOST MODE with local and remote radio buttons: The remote radio button permits AANNI users to run HSIM installed on a remote machine. To run HSIM through remote mode, select the remote radio button from HSIM HOST MODE and fill in the remote host name. Since the operation is done via the rsh Unix command, make sure that the .rhosts file is set up and the HSIM executable is in the search path on the remote machine.

When the [OK] screen button is selected, the Synopsys artistIF license is automatically checked out. If the license checkout is successful, all of the menu commands will be activated and a licence checkout confirmation will be displayed as shown in [Figure 44](#).

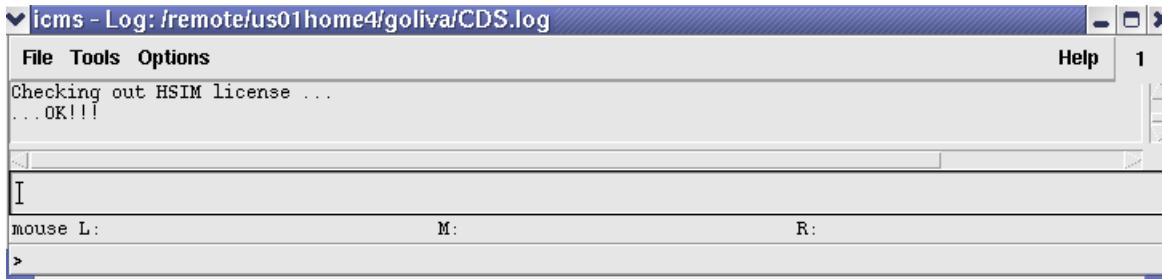


Figure 44 License Checkout Confirmation Window

Setup Parameters

Native Netlist Integration provides several forms to allow specification of HSIM parameter and command options. This works similarly with HSIM socket and HSIMD direct integrations.

- HSIM -> Setup Parameters -> Basic: Basic specifies parameters for HSIM commands including HSIMVDD, HSIMSPEED, HSIMOUTPUT, etc. Refer to the *HSIM Simulation Reference Manual: Chapter 6, Simulation Parameters*

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

for details on HSIMVDD and HSIMSPEED. Refer to the *HSIM Simulation Reference Manual: Chapter 8, Simulation Output* for information on HSIMOUTPUT and other output formats.

- HSIM -> Setup Parameters -> Advanced: Advanced specifies less frequently used parameters. These parameters are classified into different categories based on their specific functionality.

Basic and Advanced HSIM parameters in the form have the same default values as HSIM.

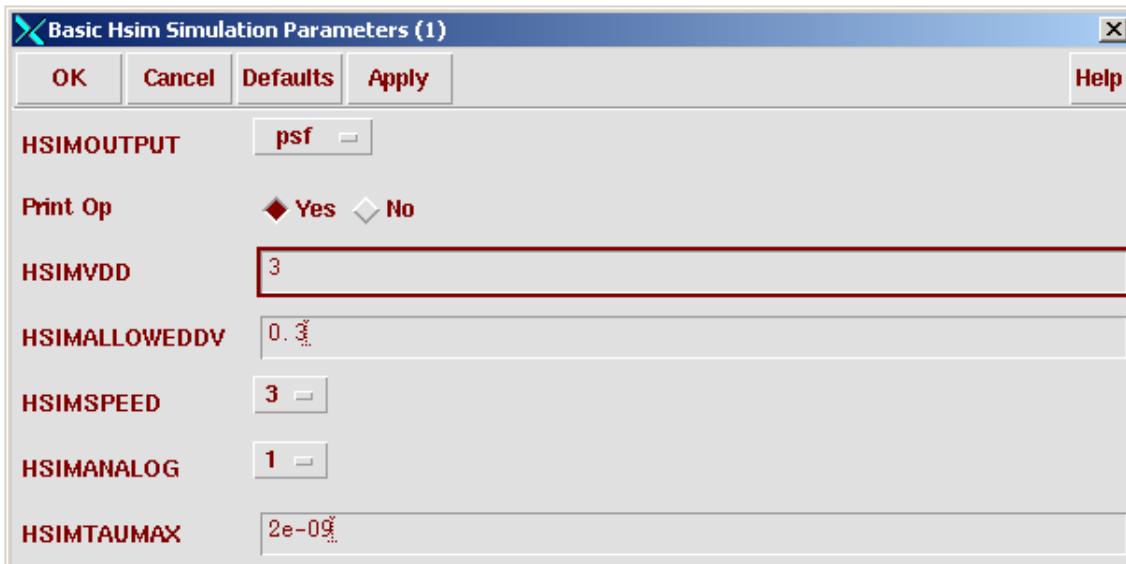


Figure 45 HSIM Basic Parameters Form

- HSIM -> Setup Parameters -> Manual
Netlist options or statements can be manually keyed-in and then included in the top (final) netlist. All content specified in this form must follow the legal

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

netlist syntax shown in [Figure 46](#).

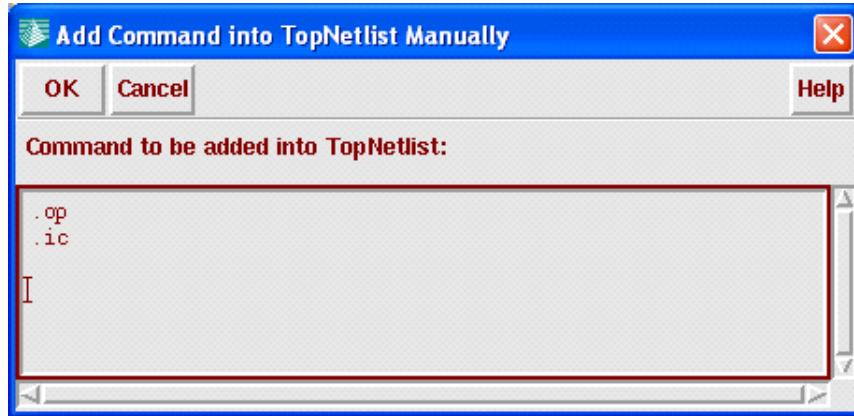


Figure 46 Manual Command Entry Window

Netlisting

Native Netlist Integration netlisting provides the following functionality:

- Create Top Netlist
- Edit Top Netlist
- Create Top & Host Netlist
- View Host Netlist

Create Top Netlist

HSIM -> Netlisting -> Create Top Netlist

By default, Create Top Netlist only creates the top-level netlist content. If there is no existing host netlist, Native Netlist Integration automatically creates the host netlist before making the top-level netlist. The top-level netlist can then be created which includes the following:

- Host netlist: Automatically generated by the host netlister.
- HSIM parameters and options: HSIM Basic and Advanced parameters.
- Manually generated netlist statements or options: HSIM Manual parameters.

Edit Top Netlist

HSIM -> Netlisting -> Edit Top Netlist

Edit Top Netlist allows manual editing of the top-level netlist generated by Create Top Netlist with any text editor. To set up the editor for opening the top netlist, set and export the HSIM_FAVORITE_EDITOR shell environment variable as shown in the following syntax example:

```
export HSIM_FAVORITE_EDITOR=xedit
```

In this syntax example, xEdit is used as the default editor so that Native Netlist Integration reads in this environment variable and automatically set the editor accordingly in icfb or icms. Users can temporarily switch to and from various editors at any time entering the following command directly in the Command Interpreter Window (CIW).

```
editor="commandToInvokeEditor"
```

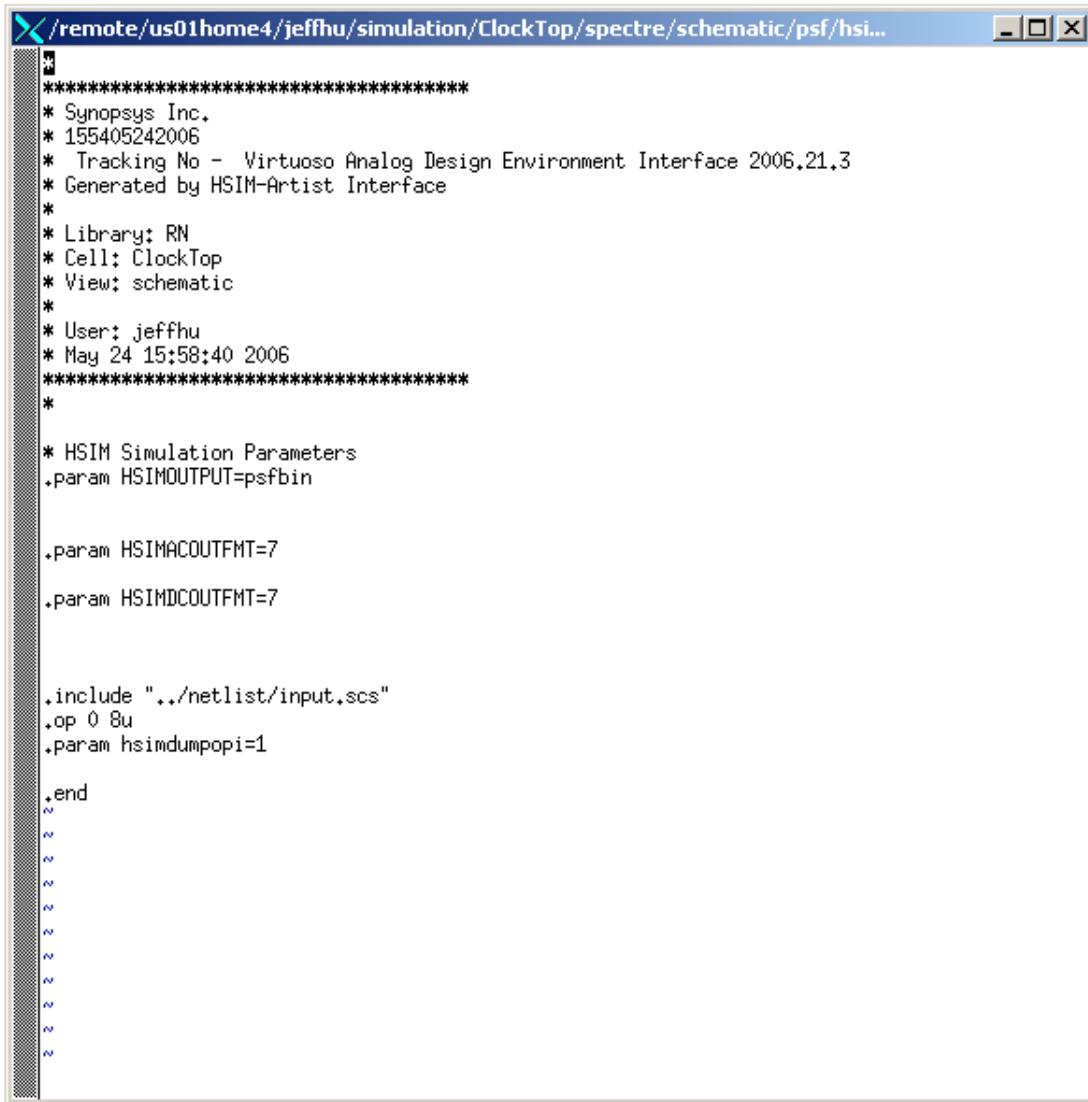
To set the editor to vi, enter the following in the CIW window:

```
editor="vi"
```

[Figure 47](#) shows a typical top-level netlist.

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



The screenshot shows a terminal window with the following content:

```
X:/remote/us01home4/jeffhu/simulation/ClockTop/spectre/schematic/psf/hsi...
[Minimize] [Close]

*****
* Synopsys Inc.
* 155405242006
* Tracking No - Virtuoso Analog Design Environment Interface 2006.21.3
* Generated by HSIM-Artist Interface
*
* Library: RN
* Cell: ClockTop
* View: schematic
*
* User: jeffhu
* May 24 15:58:40 2006
*****
*
* HSIM Simulation Parameters
+param HSIMOUTPUT=psfbins

+param HSIMACOUTFMT=7
+param HSIMDCOUTFMT=7

.include "../netlist/input.scs"
+op 0 8u
+param hsimdumpopi=1

+end
^
^
^
^
^
^
^
^
^
^
^
^
```

Figure 47 Typical Top-Level Netlist

Create Top & Host Netlist

HSIM -> Netlisting -> Create Top & Host Netlist

Create Top & Host Netlist creates the top-level netlist and forces the host simulator to update or regenerate the host netlist.

View Host Netlist

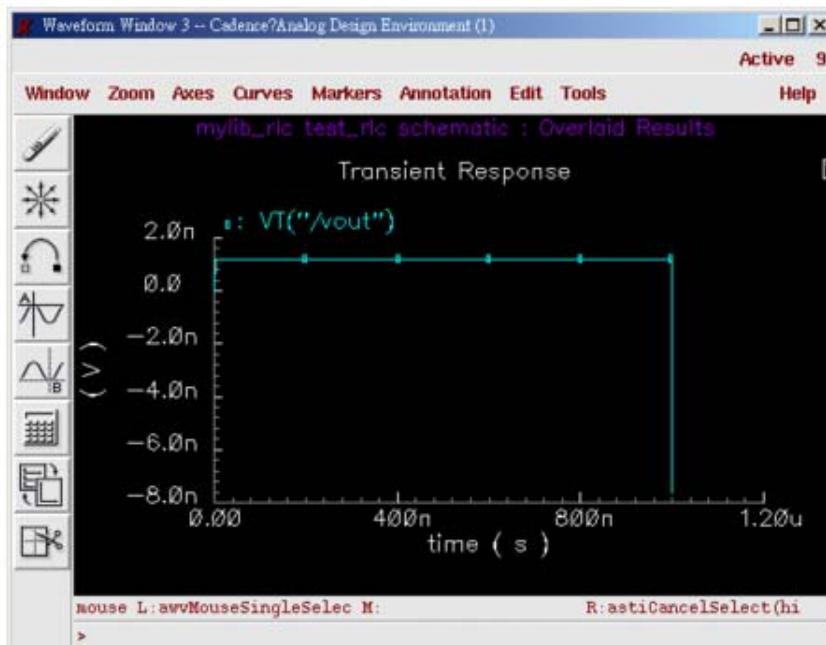
HSIM -> Netlisting -> View Host Netlist

View Host Netlist opens the host netlist file for viewing.

Run HSIM

HSIM -> Run HSIM

- HSIM -> Run HSIM runs HSIM simulation on the top-level netlist with command line options specified using the Setup Environment option form, when applicable.
- After running HSIM, all output files are stored in the directory specified under the Result Directory in the Setup Environment form.
- A Waveform window similar to the one shown in [Figure 48](#) will automatically display for pre-selected signals.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 48 Waveform Window

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

- Cross probing of results is also supported.
- If parameter settings are modified and HSIM -> Run HSIM is run directly without updating the top netlist, Native Netlist Integration displays a confirmation dialog box asking whether the Top Netlist File should be recreated when running HSIM as shown in [Figure 49](#). Use the [Yes] or [No] screen buttons to tell the system whether to update the Top Netlist File or not.

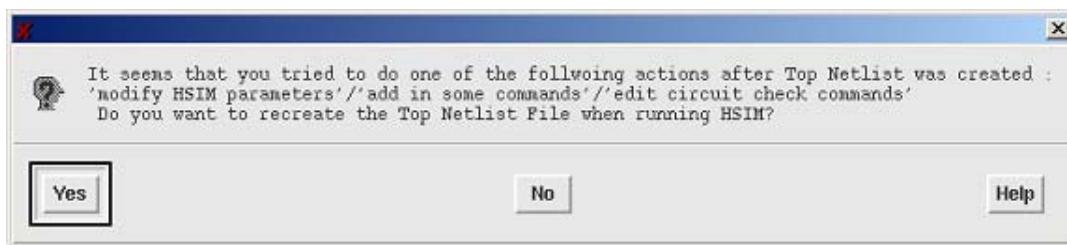


Figure 49 Top Netlist File Regeneration Confirmation Window

Regenerate the Netlist and Run HSIM

HSIM -> Re-Netlist & Run HSIM

Re-Netlist & Run HSIM forces the host netlist or top netlist to be updated and/or regenerated and run HSIM.

Toggle between Spectre and HSIM Simulation Database for Waveform Probing

HSIM -> Hsim/Spectre Results Toggle

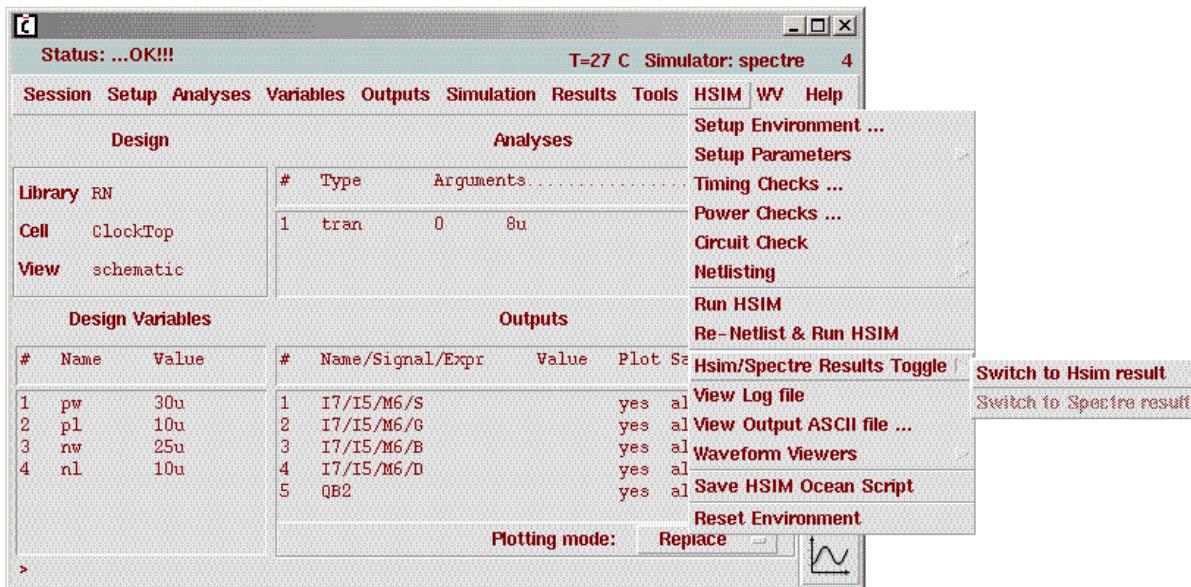
If you want to perform Spectre® simulation and HSIM simulation within the same Virtuoso Analog Design Environment session you can display the associated waveform through the standard plotting and cross-probing features by switching between the Spectre simulation results and the HSIM simulation results. After setting up the HSIM environment, the program points to the HSIM simulation database for waveform display by default. If you are interested in HSIM simulation only through the whole Virtuoso Analog Design Environment session you can ignore this feature.

The default setting for case sensitivity and database toggle can be specified in either a .cdsinit file or .aannienv file. The .cdsinit file can reside in your current working directory or your home directory. If you specify the setting in the .aannienv file, the interface program searches for it first in the current working

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

directory and then in the home directory. If the setting resides in both of these files, the .aannienv setting has the higher priority (and in this case AAI also issues a warning message).



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 50 HSIM-Spectre Results Toggle

There are two database toggle flows that are recommended to if you need to perform Spectre simulation as well as HSIM simulation:

Toggle Flow I:

Start a Virtuoso Analog Design Environment session and complete the following steps:

1. Setup -> Design
2. Setup -> Simulator -> Spectre
3. Setup -> Model Libraries...
4. Analyses -> Choose...
5. Outputs -> To Be Plotted...
6. Simulation->Netlist and Run (conduct Spectre simulation)

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

7. Results -> Direct Plot (plot Spectre simulation result)
8. HSIM -> Setup Environment
9. HSIM -> Re-Netlist & Run HSIM
10. Results -> Direct Plot (plot HSIM simulation result)
11. HSIM -> Hsim/Spectre Results Toggle -> Switch to Spectre result
12. Results -> Direct Plot (plot Spectre simulation result)
13. HSIM -> Hsim/Spectre Results Toggle -> Switch to Hsim result
14. Results -> Direct Plot (plot HSIM simulation result)

Toggle Flow II:

Start a Virtuoso Analog Design Environment session and complete the following steps:

1. Setup -> Design
2. Setup -> Simulator -> Spectre
3. HSIM -> Setup Environment
4. Setup -> Model Libraries...
5. Analyses -> Choose...
6. Outputs -> To Be Plotted...
7. HSIM -> Re-Netlist & Run HSIM
8. Results -> Direct Plot (plot HSIM simulation result)
9. HSIM -> Reset Environment
10. Simulation -> Netlist and Run (conduct Spectre simulation)
11. Results -> Direct Plot (plot Spectre simulation result)
12. HSIM -> Setup Environment
13. HSIM -> Re-Netlist & Run HSIM
14. HSIM -> Hsim/Spectre Results Toggle -> Switch to Spectre result
15. Results -> Direct Plot (plot Spectre simulation result)
16. HSIM -> Hsim/Spectre Results Toggle -> Switch to Hsim result
17. Results -> Direct Plot (plot HSIM simulation result)

Note: The above recommended flows assume that Spectre simulation and HSIM simulation are each conducted once respectively. If you intend to conduct more than one Spectre run within the same HSIM-Virtuoso session you must to first select HSIM -> Reset Environment immediately before running the Spectre simulation. This ensures that the data access function points to the correct simulator. Otherwise, the plotting may not be able to map to the appropriate waveform database.

CircuitCheck in the HSIM-Virtuoso Interface Environment

HSIM -> CircuitCheck

The HSIM-Virtuoso Interface integrates the CircuitCheck (CCK) features into the Virtuoso Analog Design Environment. Specific CCK commands or the entire CCK command file can be specified using the Native Netlist Integration GUI as shown in [Figure 51](#). The interface combines CCK-related information into the top-level netlist for HSIM to analyze.

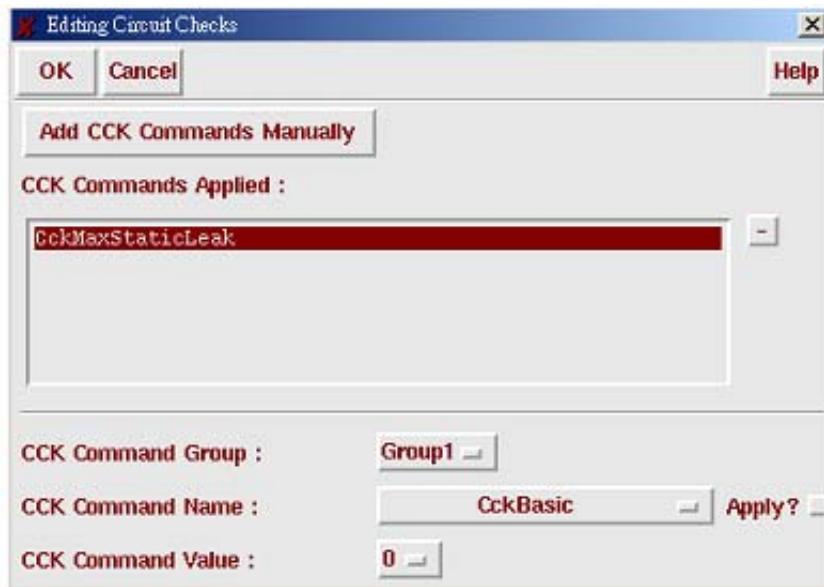


Figure 51 Editing CircuitChecks Window

View Log File

HSIM -> View Log File

View Log File displays the HSIM log file.

View Output ASCII Files

HSIM -> View Output ASCII File

View Output ASCII File brings up the file browser containing all the ASCII results stored in the result directory as shown in [Figure 52](#).

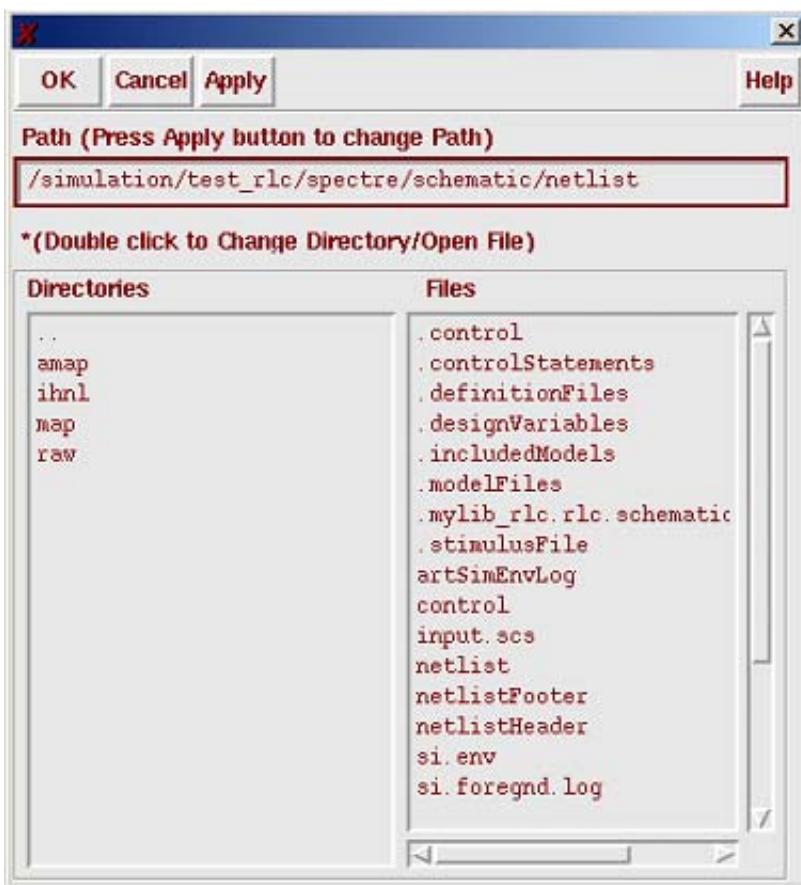


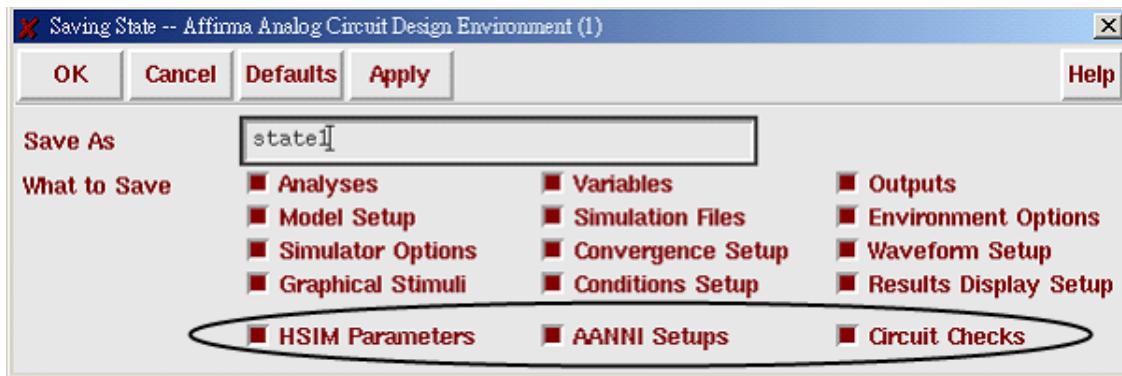
Figure 52 File Browser Window

Save/Load States

Native Netlist Integration supports the Virtuoso Analog Design Environment saving and loading states feature that contains HSIM Parameters, AANNI Setups, and CircuitChecks options. These options are added to the Virtuoso Analog Design Environment Saving State window as shown in [Figure 53](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

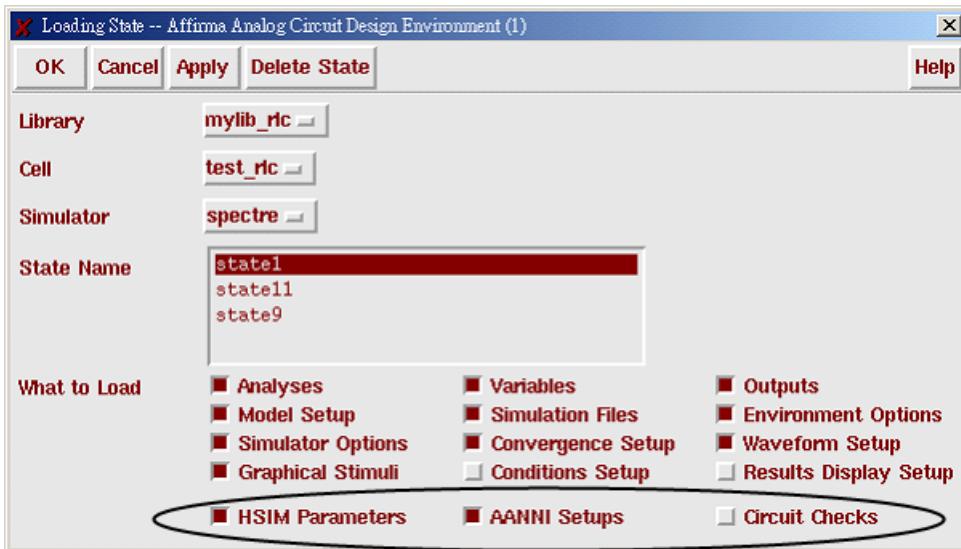
Figure 53 Saving State Form Window

These options have the following functions:

- HSIM Parameters: HSIM Parameters is used to save or load states specified through Setup Parameters->Basic.
- AANNI Setups: This option is used for miscellaneous related parameters to the Native Netlist Integration flow.
- CircuitChecks: CircuitChecks is primarily used for commands specified through the Editing CircuitCheck form.
- Saved states will be stored in files under the directory specified in the Session-> Options...->State Save Directory field.
- Once states for Native Netlist Integration are saved, they can be loaded using the Session->Load States... form as shown in [Figure 54](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 54 Loading State Form Window

Note: Before loading Native Netlist Integration states, the HSIM -> Setup Environment must first be activated or a warning message is displayed as shown in [Figure 55](#). Loading save state without first activating Native Netlist Integration results in missing parameters settings.

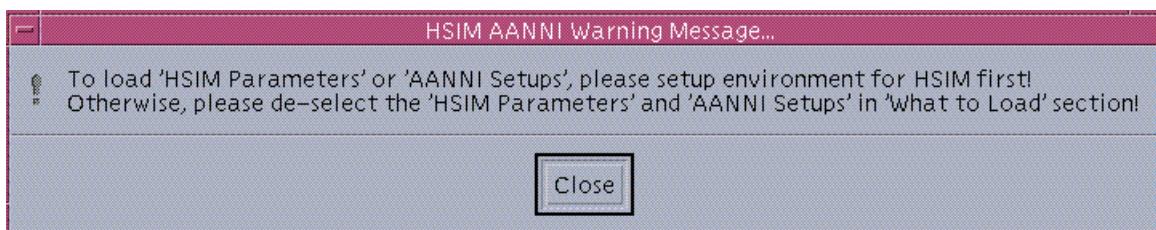


Figure 55 HSIM Native Netlist Integration Warning Message Window

Check in Synopsys License

When the Virtuoso Analog Design Environment session is terminated, the artistlF license is automatically checked in and a message is generated Command Interpreter Window as shown in [Figure 56](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

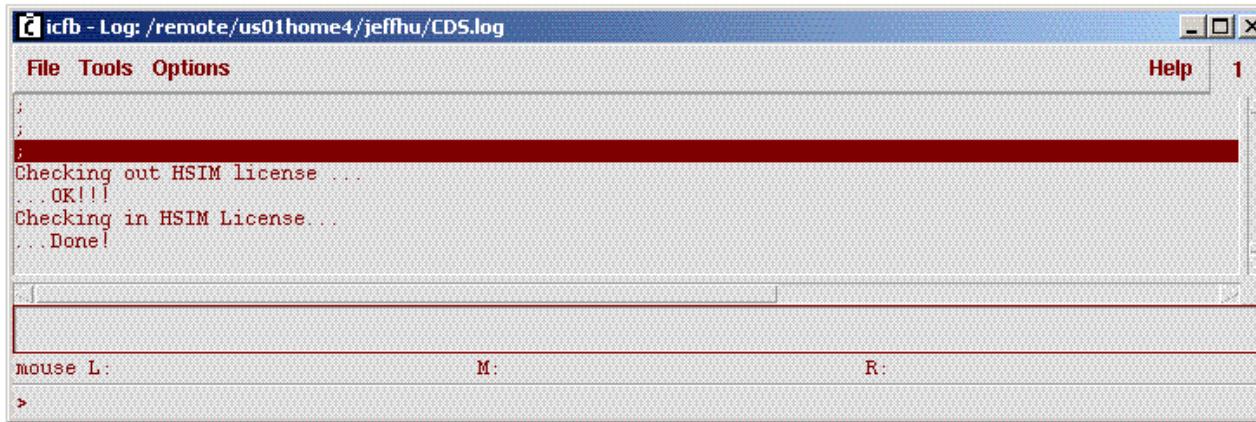


Figure 56 Log Window with Synopsys Licence Check Confirmation

Note: The HSIM -> Reset Environment command is also located under the HSIM pull-down menu for the following reasons:

- The artistIF license is forced to be checked in so that even a current session is active, the commands under the HSIM pull down menu will be inactivated and the released artistIF license can be used by other users.
- Native Netlist Integration releases ownership of the data access function back to the host simulator. This is required to perform special features such as signal cross probing. It is possible for users to switch back-and-forth between the host simulator and HSIM within the same Virtuoso Analog Design Environment session to conduct simulation and then cross probing.

Caution: It is strongly recommended that users issue the HSIM -> Reset Environment command before switching back to the host simulator to eliminate associated host simulator signal cross-probing failures.

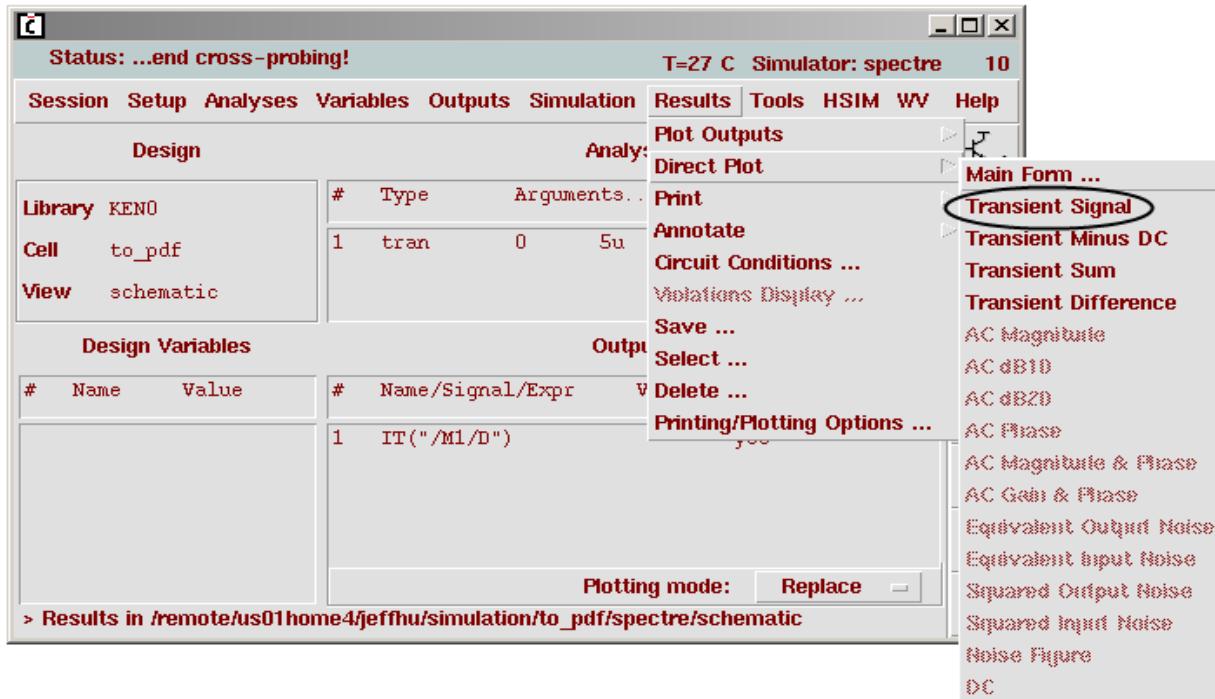
Cadence Cross-probing

Native Netlist Integration works with Cadence® cross-probing functionality with the following steps:

1. After simulation has completed, select Results > Direct Plot > Transient Signal, as shown in [Figure 57](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



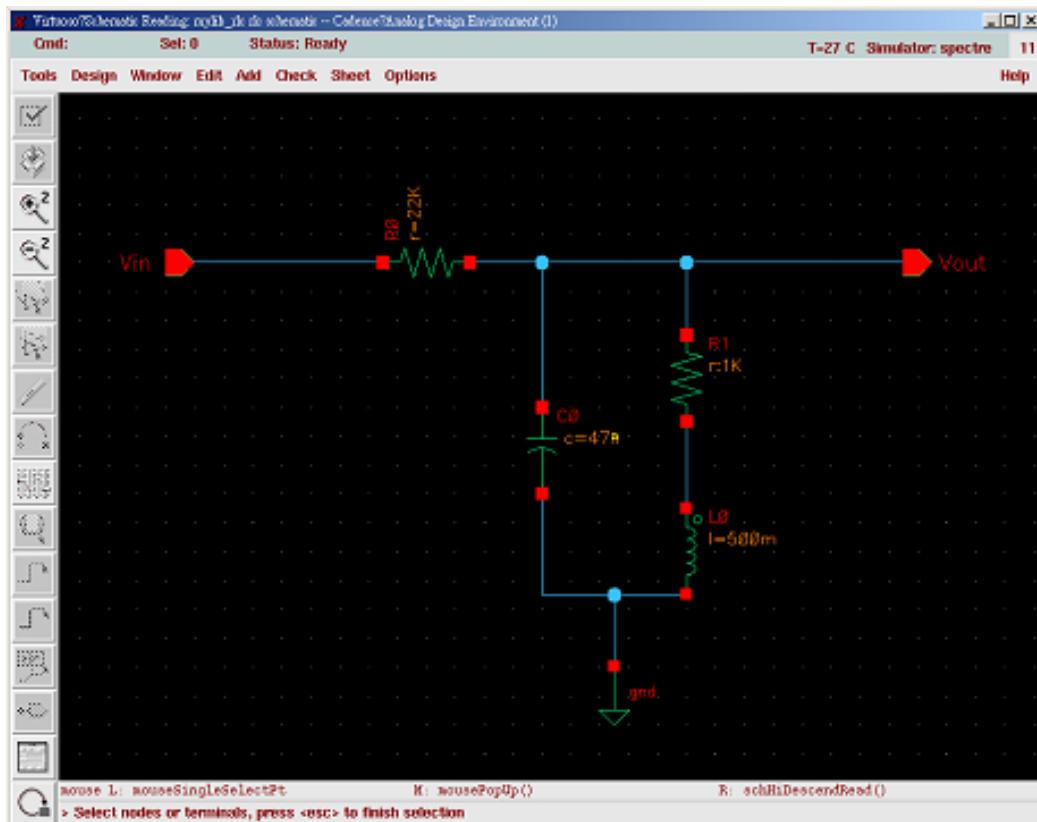
© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 57 Virtuoso Analog Design Environment Window with the Results->Direct Plot->Transient Signal Pull-down Windows

When Transient Signal is selected, the Virtuoso Schematic Editor window shown in [Figure 58](#) is automatically displayed.

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

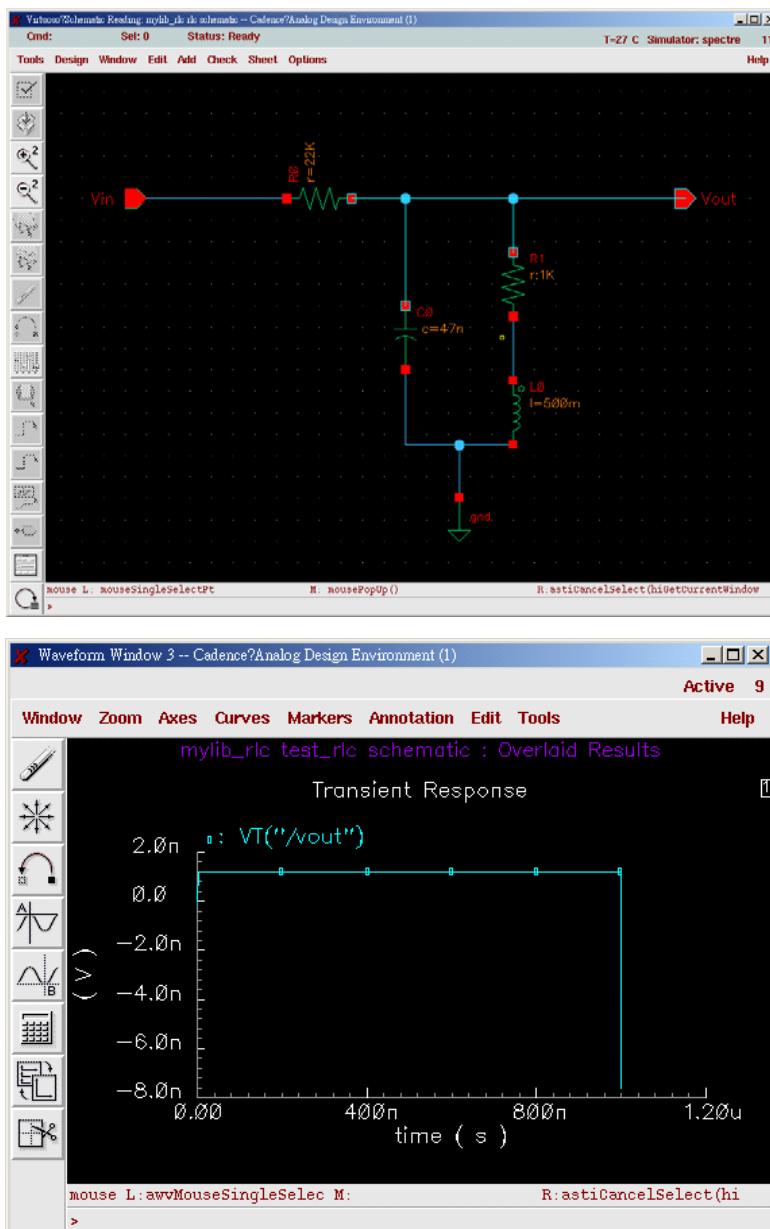
Figure 58 Virtuoso Schematic Editor Window

2. Go to the desired level of design by using Design->Hierarchy->Descend Edit (or Read) in the Virtuoso Schematic Editor window as shown in [Figure 58](#).
3. Select the signals to probe by LEFT clicking with the mouse. Signals can be either node voltage or instance current. Press [Esc] to terminate the signal selection process. The corresponding signal wave form will be displayed on the Waveform window as shown in [Figure 59](#).

Feedback

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 59 Virtuoso Schematic Editor Window & Corresponding Waveform Window

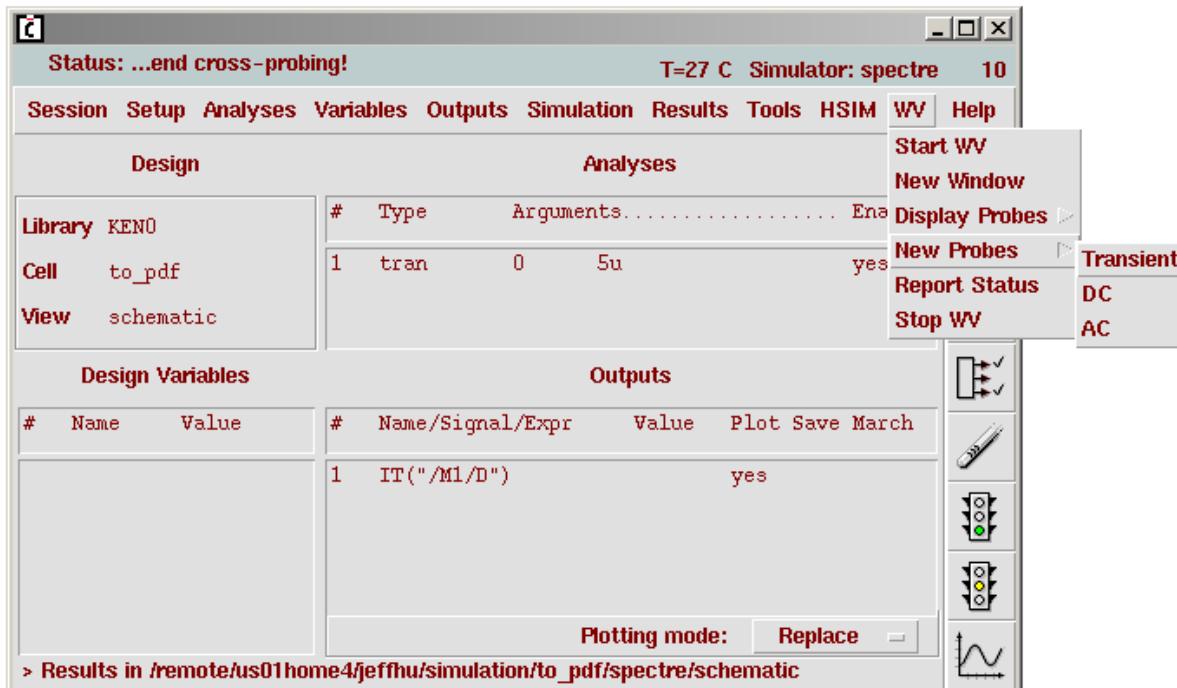
WaveView Analyzer Cross-probing

In addition to supporting the Cadence Waveform viewer, Native Netlist Integration also supports cross-probing functions with Sandwork's WaveView Analyzer (WV). Currently, WaveView Analyzer only partly supports hierarchical cross-probing.

Note: Cross-hierarchy signal probing is not supported.

To use WaveView Analyzer for cross probing, use the following steps.

1. Select WV->start WV.
2. In the Virtuoso Schematic Editor window, select the desired circuit hierarchy level.
3. Select WV->New probe->Transient, as shown in [Figure 60](#).



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

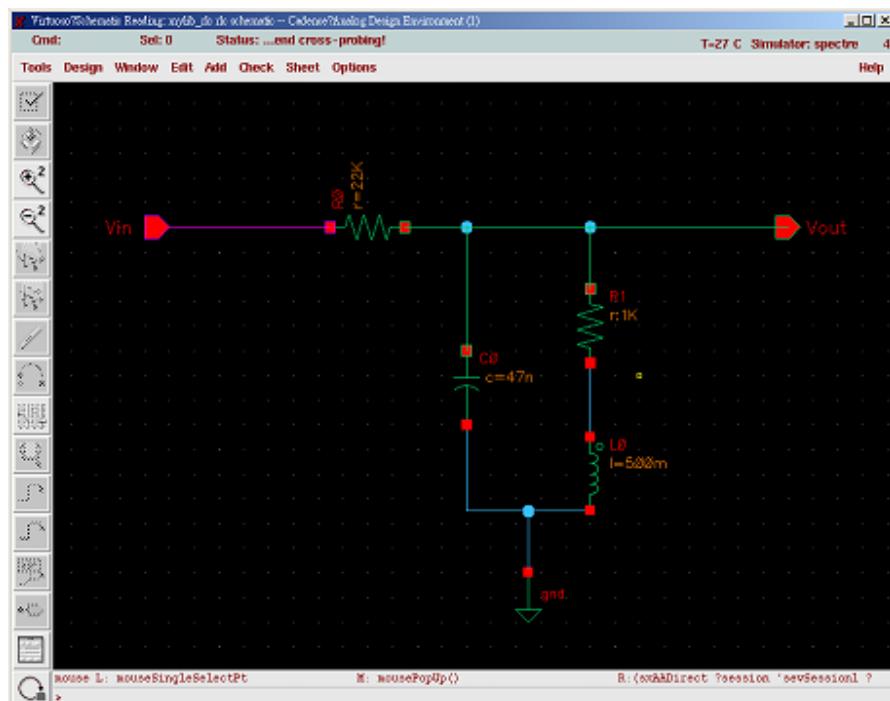
Figure 60 Cross-probing with WaveView Analyzer

4. Select the signal to be plotted by clicking VIN and then press Escape to display the waveform as shown in [Figure 61](#).

Feedback

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

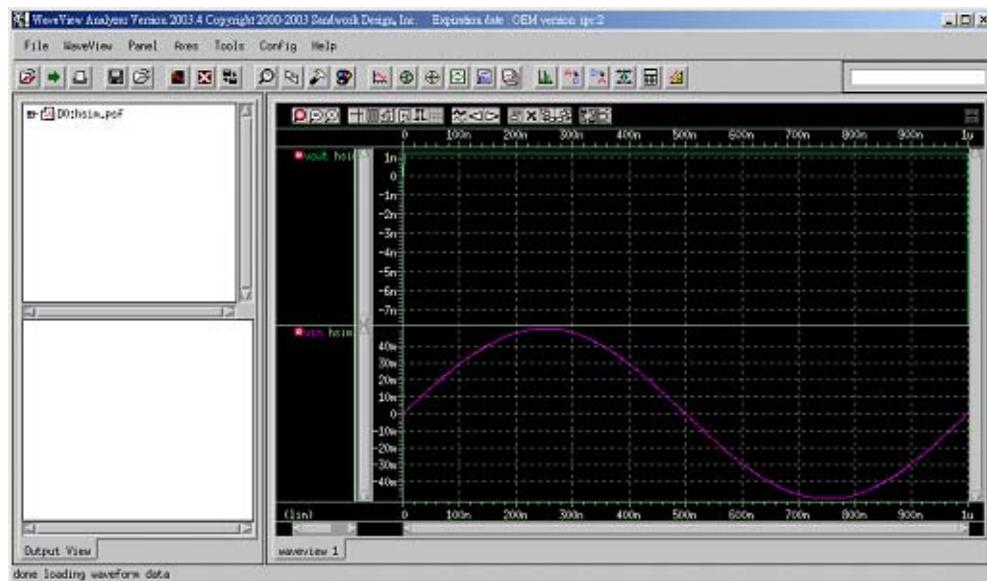


Figure 61 Virtuoso Schematic Editor Window Showing a Selected Signal Plotted in WaveView Analyzer

5. Repeat [Step 1](#) through [Step 4](#) in this section for additional signals.

CoSim (AACoSim) Integration

The purpose of CoSim integration is to integrate the Synopsys co-simulation technology into Cadence Virtuoso Analog Design Environment, the integration approach is very similar to Native Netlist Integration (AANNI), which utilizes the host simulator (such as spectreVerilog or hspiceSVerilog) to generate the host netlist (digital as well as analog parts), then make the top-level netlist with the cosim configuration file to conduct HSIM Verilog co-simulation. CoSim shares the same installation procedures with Native Netlist Integration with additional UNIX environment setup.

Note: During the simulation cycle for CoSim (AACoSim) the hsim-cosim license is required, in addition to the artistIF license.

UNIX Setup

To set up CoSim for the Unix environment, perform the following tasks.

1. Set up the search path for Verilog simulator executable.

```
set path=(/usr/local/vendors/cadence/ldv40/tools/bin $path)
```

2. Set up \$LD_LIBRARY_PATH

```
setenv LD_LIBRARY_PATH ${LD_LIBRARY_PATH} :  
<vpi_library_location>
```

Note: The vpi library is normally located in the HSIM release tree;
e.g., “/version/06-23-2004/hsimplus/platform/sunos_57/bin”
The vpi library is required to perform the co-simulation

CoSim Installation

To install CoSim, you must first receive the AAIM package, then set the HSIM_HOME and HSIM_ARTISTIF environment variables to where HSIM and AAIM packages are located, respectively.

Next, perform the following steps:

1. Copy \$HSIM_ARTISTIF/menus/spectreVerilog.menus to either user's \$HOME/menus/ or <working_directory>/menus directory
2. Use the following two lines in user's .cdsinit file to invoke CoSim :

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

```
nsdaAAIMPkgList='("AACoSim")
load("<AAI_installation_directory>/install/
nsdaAAIMInvoker.il")'
```

Basic CoSim Flow

Here is the basic CoSim flow:

1. User starts icms or icfb session, brings up the Virtuoso Analog Design Environment window.
2. Select spectreVerilog as the simulator, specify library and design.
3. HSIM pull-down menu shows on the RIGHT hand side of the Virtuoso window.
4. Conduct spectreVerilog netlisting, generate digital and analog host netlist.
5. Select HSIM -> Setup Environment, [OK] to check out HSIM artistIF license and activate all the CoSim commands under HSIM pull-down menu.
6. Similar to Native Netlist Integration, use HSIM -> Setup Parameters to specify desired HSIM parameters.
7. Select HSIM -> Netlisting -> Create Top Netlist to generate the top-level netlist (cosim.scs).
8. Select HSIM -> Run HSIM CoSim to start the co-simulation.

HSIM-Virtuoso CircuitCheck Integration**Native Netlist CircuitCheck**

Native Netlist CircuitCheck is the integration interface between Virtuoso and the CCK commands built into HSIM. This approach and the GUI interface provide an easy to use mechanism for setting up all kinds of CCK commands to perform various circuit checking.

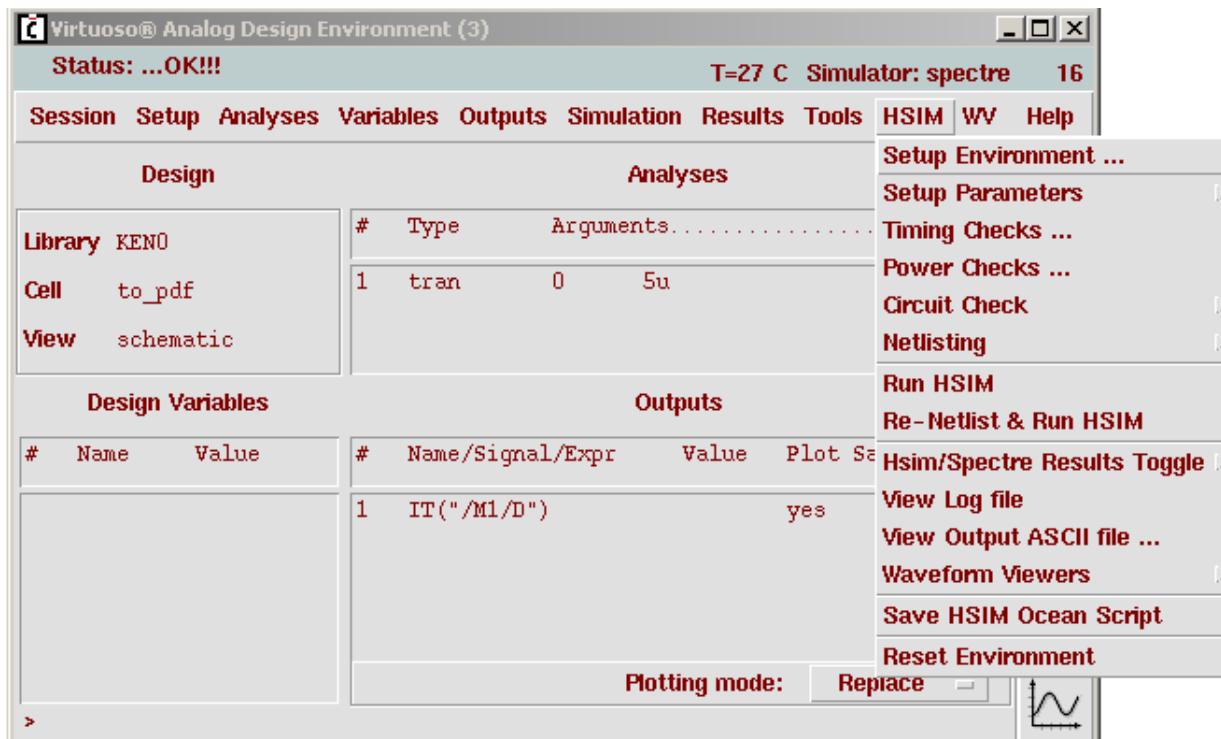
Note: Although Native Netlist Integration does not require a CCK license, a separate CCK license is required in addition to the HSIM license during the simulation cycle. This allows the CCK commands to be set and edited however, these commands will not take effect when HSIM is run unless a CCK license is available.

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

Perform the following steps to open the CCK GUI form.

1. Check out an artistIF license from the Virtuoso Analog Design Environment window shown in [Figure 62](#) as follows:
2. Choose HSIM->Setup Environment.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 62 Virtuoso Analog Design Environment Window

3. Choose HSIM->CircuitCheck to open the Editing CircuitChecks window shown in [Figure 63](#).

Feedback

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

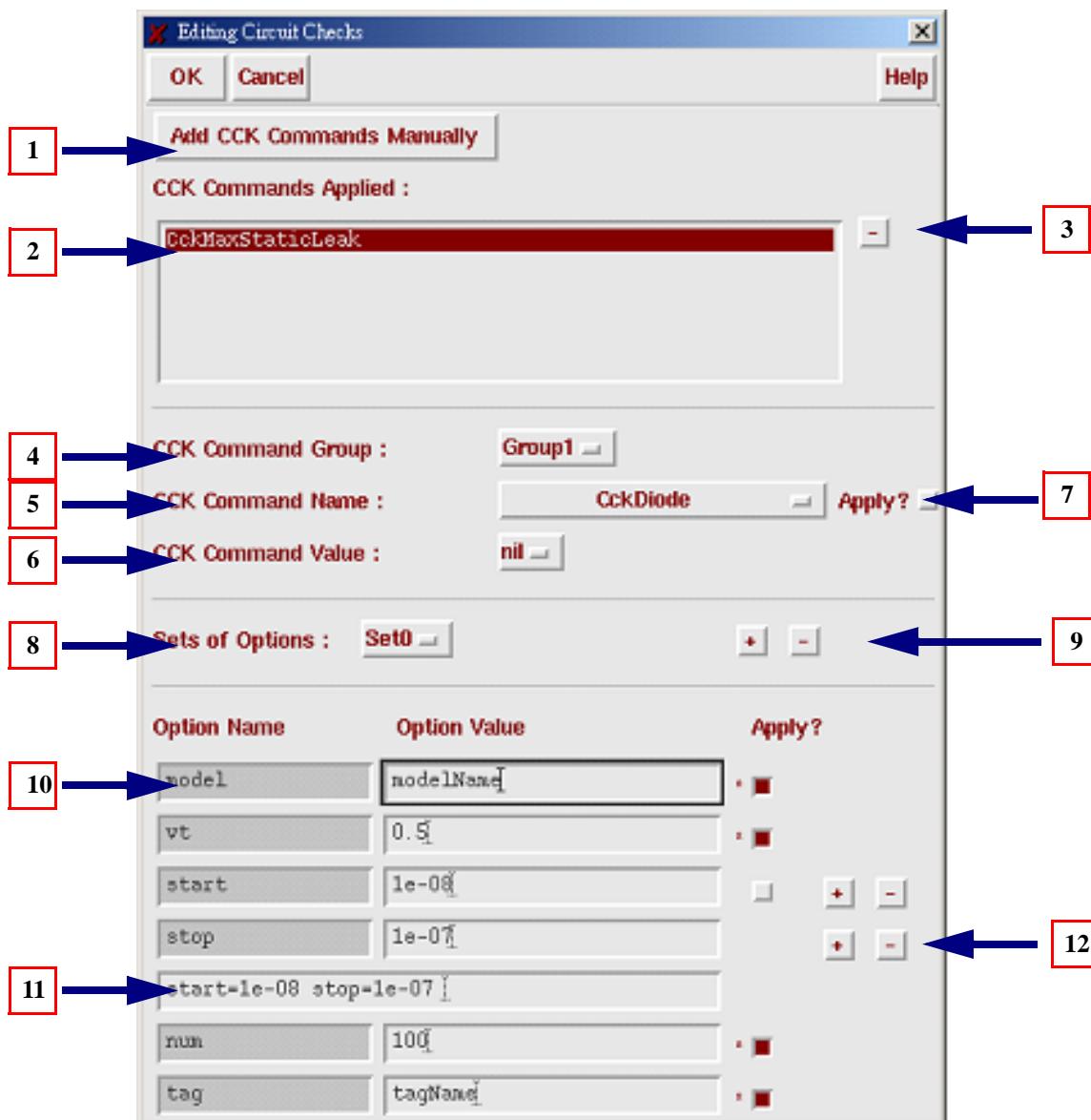


Figure 63 Editing CircuitChecks

The following are descriptions of the Editing CircuitChecks window features shown in Figure 63.

1 [Add{{CCK{{Commands{{Manually}}}}

Most CCK commands can be covered and selected through the cyclic selection on the form. New commands or those with complex parameter configurations can also be entered manually. [Figure 64](#) displays the window used to manually enter commands.

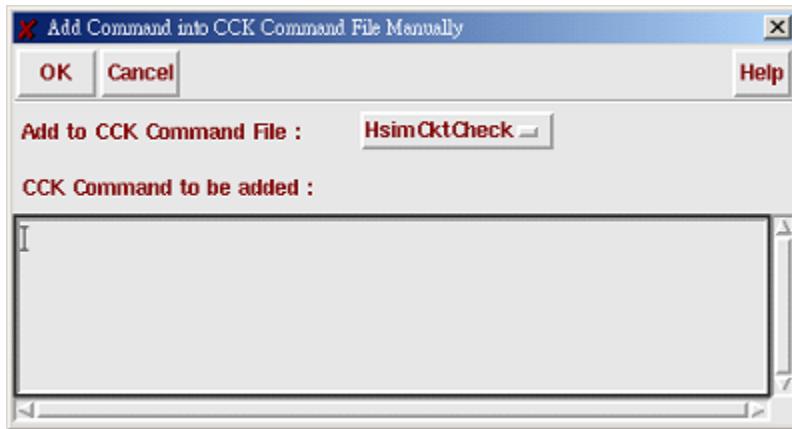


Figure 64 Add Command into CCK Command File Manually Window

Commands added using this window are included in the file generated via the Edit CircuitChecks form as described in the following sections.

2 CCK Command Applied

CCK Command Applied shows the list of CCK commands selected for application. To apply a command, press any associated Apply? button on the form.

The name of the selected command is listed in the CCK Command Applied: list box of the Editing CircuitChecks window shown in [Figure 63 on page 796](#). Only the commands listed in the list box will be output into a CCK command file. These will then be included in the HSIM simulation netlist.

3 [{-{]}

The [{-{]} screen button to the RIGHT of the CCK Command Applied list box is used to deselect an applied command. To deselect a command, select and highlight the command from the list of commands in the CCK Command Applied: list box and press the [{-{]} screen button.

4 CCK Command Groups

Describes all CCK commands, which are classified into 3 groups having the following classifications:

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

- Group 1
- Group 2
- Group 3

Note: There is no Group 2 displayed in the drop down list shown in [Figure 65](#) since Group 2 commands are mainly used in the interactive mode.



Figure 65 CCK Command Group Drop-Down List

5 CCK Command Name

This drop down menu accommodates all CCK commands within the selected command group; for example, Group 1 commands are shown in [Figure 66 on page 799](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

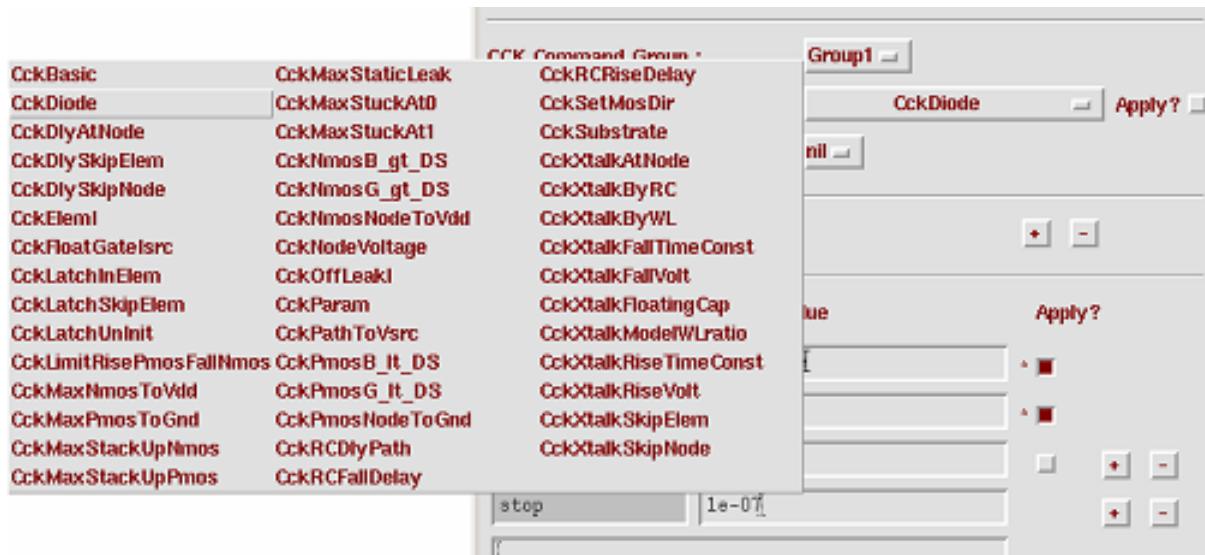


Figure 66 CCK Command List

In the list shown in Figure 67, there is only one Group 3 command; however, it has numerous options listed.

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

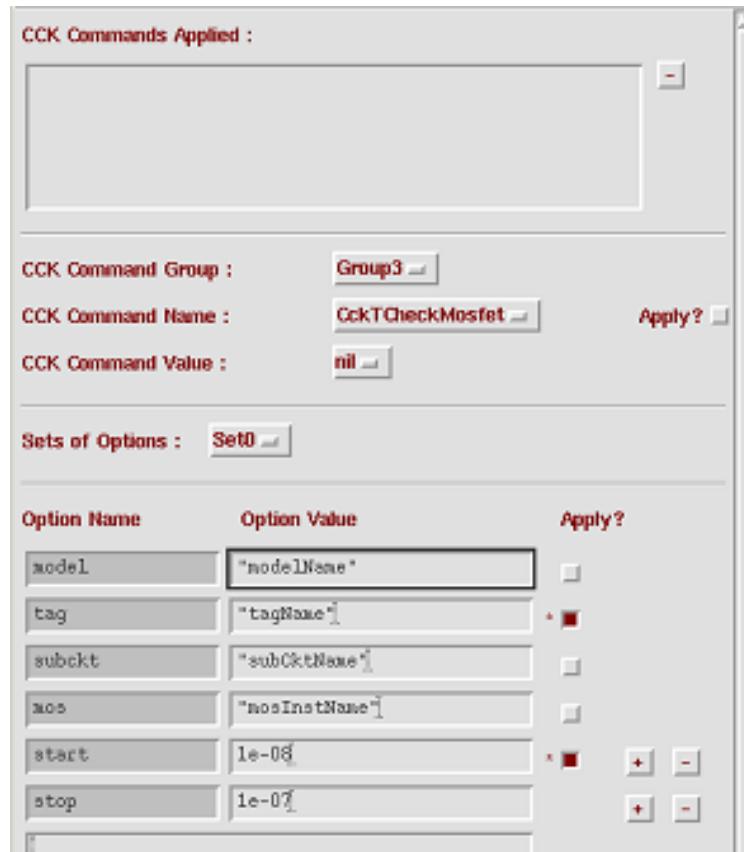


Figure 67 CCK Commands Applied: Window

6 CCK Command Value

CCK Command Value contains a list of commands with assigned values selected from a pre-determined list. A cyclic value button permits selecting from the command list.

Some commands themselves have values but without options.

7 Apply?

Apply? applies the command selected from the CCK Command Name: drop down menu. The name of selected command is listed in the CCK Command Name: list box when activated. If Apply? is deselected, then the corresponding command name will be removed from the applied command list box.

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

8 Sets of options

Certain commands allow multiple sets of options, e.g. different CCK command sets may be specified with different models. To setup multiple options, perform the following, Select [+]. The [+] screen button is located on the RIGHT of the window.

Select [+] refreshes the bottom portion of the Editing CircuitChecks form to reflect a new option set with the new set number. The default option set values are set as shown in [Figure 68](#):

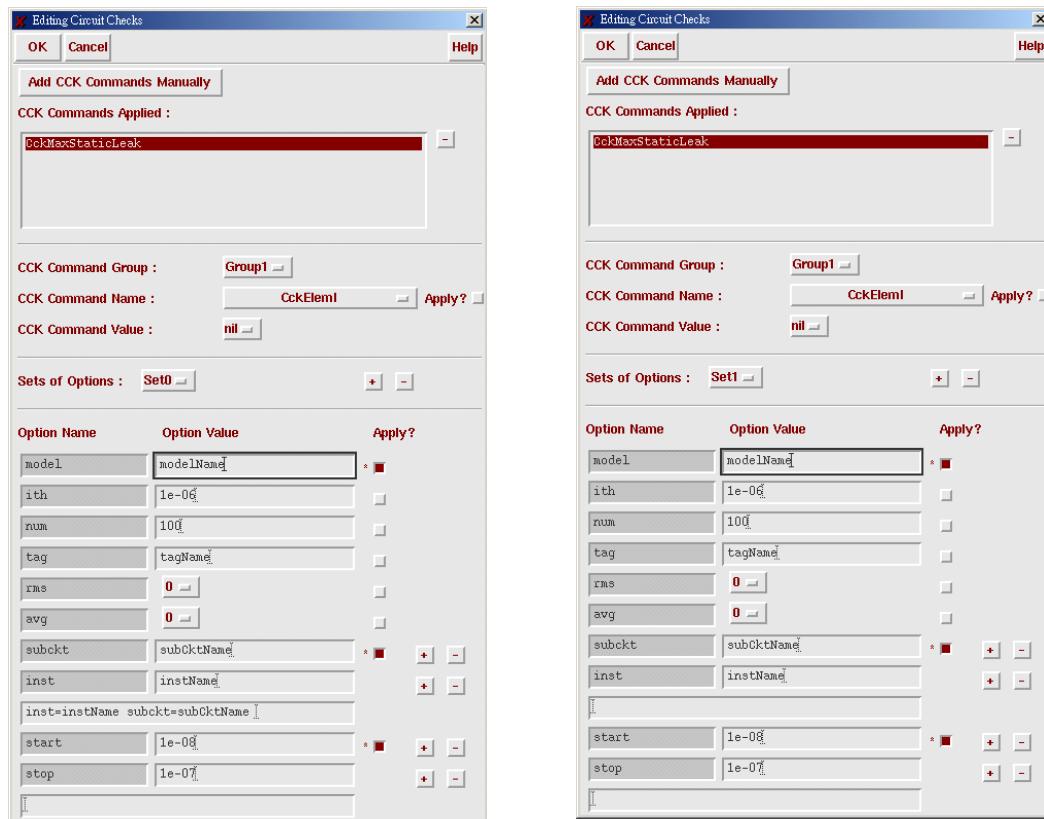


Figure 68 Editing CircuitChecks Window

If any command has multiple option sets, each set is presented on an individual command line in the output file.

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

9 Sets of options

Use the [{+{]} and the [{-{]} screen buttons to add or delete an option set from an option set. The [{+{]} and the [{-{]} screen buttons function as follows:

- [{+{]} adds a new set of options.
- [{-{]} deletes the selected set of options from the current selection shown on the menu.

10 Command option

The Command option specifies the command option name, especially when multiple option sets are applied in the CCK command.

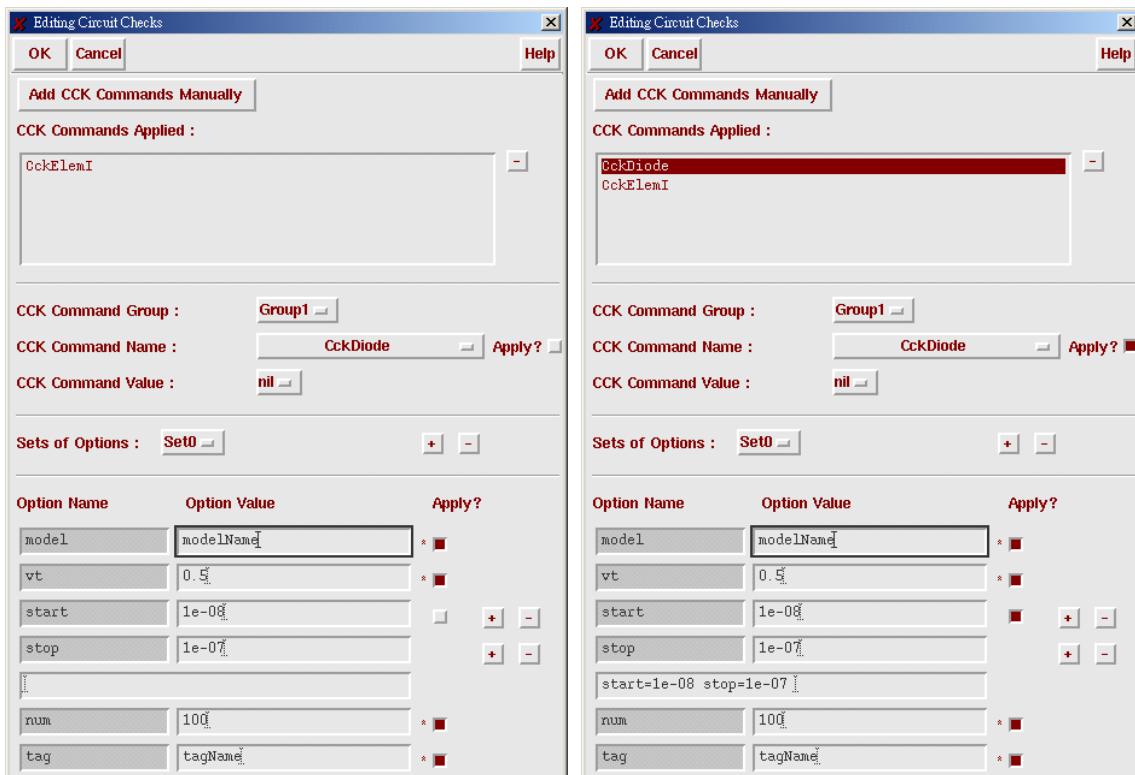


Figure 69 Editing CircuitChecks Window with Commands

11 Combinations of related options

Some related CCK command options are used in combination such as start/stop, subckt/inst, subckt/node, skipsub/skipinst, etc. These combinations can be specified in multiple sets having different option values. The string field displayed at item [11]. displays the current user specified setting. The controls described in [12] to are used to modify this field.

12 [{+{]} and [{-{}} for combined options

The [{+{]} and [{-{}} screen buttons are used to control the option as follows:

- [{+{]} adds the corresponding option=value pair into the string field described in [11].
- [{-{}} deletes the last added option=value pair.

Viewing Commands When Multiple Commands Are Applied

When more than one command is applied, it is possible to navigate between each of the commands by double clicking on a command name listed in the applied command list box as shown in [Figure 70](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support

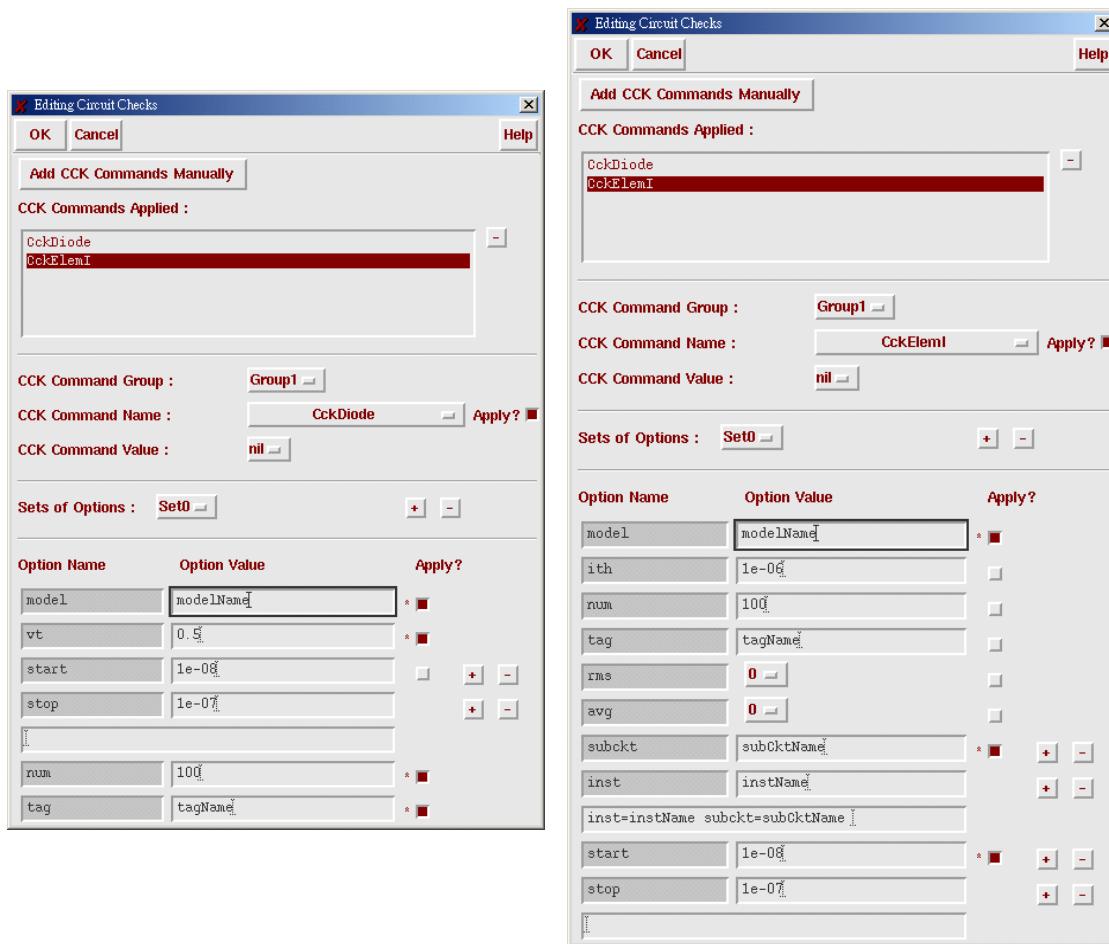


Figure 70 Editing CircuitChecks Window

cckCommandFile, cckDeviceVFile

Selecting the [OK] screen button on the CircuitCheck form generates two files in the same directory as the top netlist file.

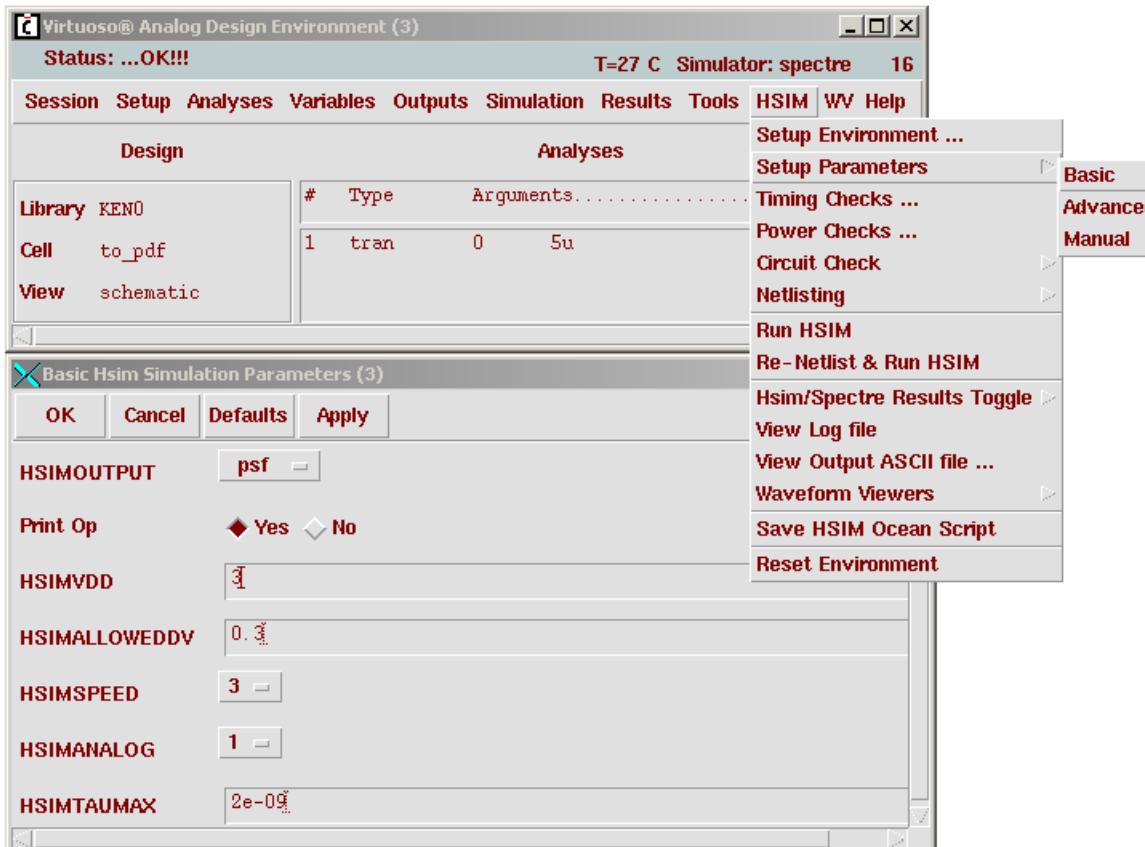
- cckCommandFile lists all applied Group 1 commands.
- cckDeviceVFile lists all applied Group 3 commands.

The top netlist file should always be recreated to include these CCK command files.

WaveView Analyzer Integration

To use WaveView, do the following (use Native Netlist Integration as an example):

1. Set HSIMOUTPUT to wdf as shown in [Figure 71](#).



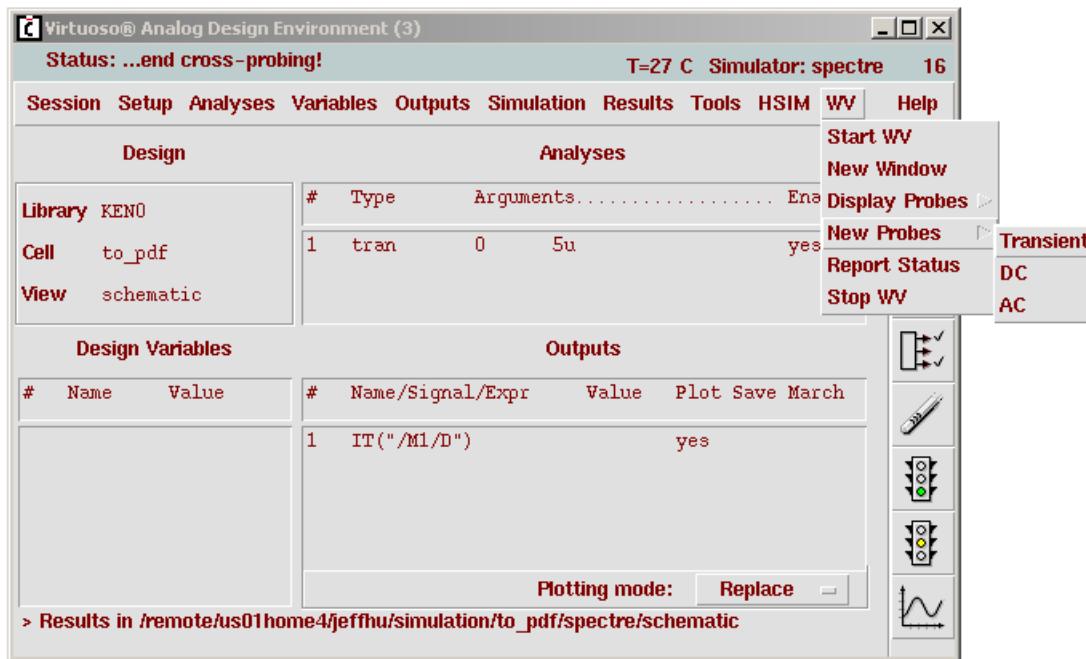
© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 71 Basic HSIM Simulator Parameters Form

2. Generate the netlist.
3. Run the simulation.
4. Choose WV -> New Probes -> Transient to cross-probe the waveform using WaveView Analyzer, as shown in [Figure 72](#).

Appendix F: HSIM-Virtuoso Analog Design Environment Interface

HSIM Virtuoso Analog Design Environment Interface Package Options and Platform Support



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 72 Cross-probing Window

5. Select the desired signals from the schematic.
6. Press [Esc] to display the waveforms to WaveView Analyzer.

See the *WaveView v2004.5 User's Guide* for details of the WaveView Analyzer features.

HSIM-Virtuoso Interface Advanced Topics

Provides information on creating an hsim/hsimD view as a stop view for the Cadence® database traversing scheme. It details the hsim/hsimD SimInfo requirements for providing information for the netlister from primitive cells required by Cadence software.

Generating hsim/hsimD View and SimInfo

An hsim/hsimD view can be created as a stop view for Cadence's database traversing scheme. hsim/hsimD view is required to indicate the stop point of database traversing for the hsim/hsimD netlister. Cadence software requires this name to ensure proper database traversing during netlisting. hsim/hsimD SimInfo is required for primitive cells to provide cell information for the hsim/hsimD netlister.

Three functions can be used to create an hsim/hsimD view and hsim/hsimD SimInfo:

```
nsdCreateView(l_libnames @optional(target "hsim") (sources nil))
nsdCreateSimInfo(l_libnames @optional(target "hsim") (sources nil))
nsdCreateSimInfoAndView(l_libnames @optional(target "hsim") (sources nil))
```

Note: An optional argument in Skill function provides users the ability to either ignore the argument or explicitly give the value of the argument to override its default value.

The following example creates hsim view for the analogLib where the value of the original target is hsim:

```
nsdCreateView("analogLib")
```

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Modifying hsim/hsimD SimInfo

The following example creates hsimD view for the libraries lib1 and lib2 that contain the value of target that are overridden to hsimD.

```
nsdCreateView('("lib1" "lib2") "hsimD")
```

Note: The value of target can be either hsim or hsimD as described in the following:

- If the value is hsim, it copies from the hspiceS view if sources is not specified.
- If the value is hsimD, it copies from the Spectre® view if sources is not specified.

Examples

```
nsdCreateView("liba")
```

Creates hsim view for liba from the HSPICE view.

```
nsdCreateView('("liba" "libb") )
```

Creates hsim view for liba and libb from HSPICES view.

```
nsdCreateView("liba" "hsimD" ("Spectre" "SpectreS" "hspices"))
```

Creates hsimD view for liba from Spectre, SpectreS, and HSPICES, with priority in the same order in which they are listed.

```
nsdCreateSimInfo("liba" "hsimD")
```

Creates hsimD simInfo for liba from Spectre siminfo.

```
nsdCreateSimInfoAndView("mylib")
```

Creates hsim view and simInfo for mylib.

Modifying hsim/hsimD SimInfo

The Interface relies on the Component Description Format hsim/hsimD SimInfo fields for netlisting. The Component Description Format editor is used to modify the fields. To use this editor, launch icms from a shell prompt. In the graphic environment, select the following in order:

Tools->CDF->Edit

Cells and libraries can be browsed and edited using the editor. User's can select from four alternatives for working with an analog component primitive:

- Component Parameters
- Simulation Information
- Interpreted Labels Information
- Other Information

After successfully installing the HSIM-Virtuoso Interface, an [hsim/hsimD] screen button appears in the Choose Simulator field as shown in [Figure 73 on page 810](#).

Feedback

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Modifying hsim/hsimD SimInfo



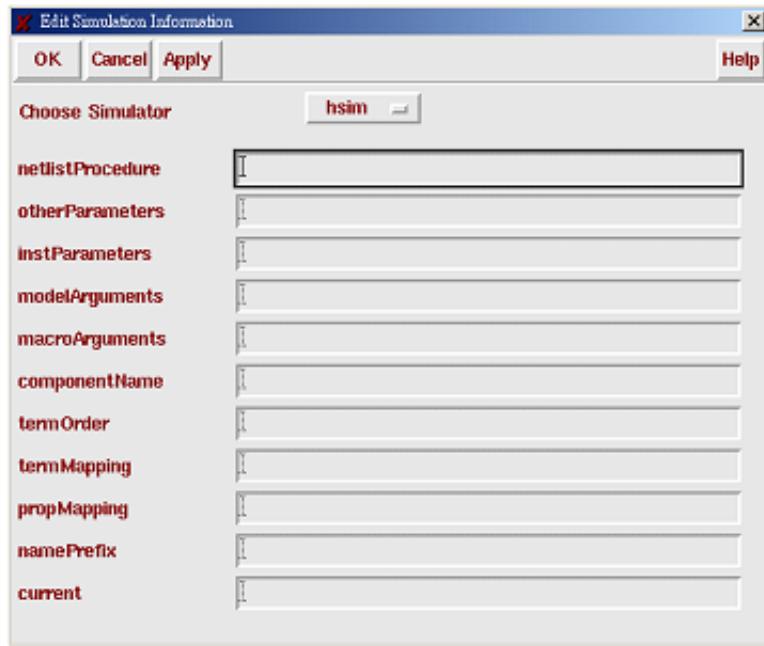
© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 73 Edit Simulation Information Window

To edit simulation information, complete the following steps:

1. Select the [Edit] screen button under Simulation Information menu bar. The Edit Simulation Information window is displayed as shown in [Figure 74 on page 811](#).

2. Select the [hsim/hsimD] screen button in the Edit Simulation Information Window. The hsim/hsimD simulation information will be displayed in the form as shown in [Figure 74 on page 811](#).



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 74 Edit Simulation Information Window

The form fields and their HSIM-specific descriptions are shown in [Table 110 on page 812](#). There are several fields required for the interface, including:

- netlistProcedure
- instParameters
- componentName
- termOrder
- propMapping

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Modifying hsim/hsimD SimInfo

- namePrefix

Table 110 Form Field Names and Descriptions

Field Name	Description
netlistProcedure	The HSIM-Virtuoso Interface provides a set of netlist subroutines for hsim/ hsimD. Refer to Table 111 on page 814 for details.
otherParameters	No specific meaning for HSIM.
instParameters	List of parameters which will be included in the netlist file associated with HSIM.
modelArguments	No specific meaning for HSIM.
macroArguments	No specific meaning for HSIM.
componentName	The type of component created.
termOrder	List of terminals that defines the net order for the component. Programmable nodes are supported.
refTermOrder	Reference term order. It is only used by netlist procedures for controlled sources, i.e. HSIMCompPrimRef.
termMapping	No specific meaning for HSIM.
propMapping	List which defines the map of users own Component Description Format parameter name and the name HSIM recognizes. The fields have to be in a format as shown below: nil <hsim parameter name 1> <user's component description format name 1> <hsim parameter name 2> <user's CDF name 2> <hsim parameter name 3> <user's CDF name 3> ...
namePrefix	The prefix for the device in the netlist file.
current	No specific meaning for HSIM.

3. Using the Simulator cyclic button, select HSIM.

4. For netlist to traverse the design database, the following must be set:
 - stopViewList
 - switchViewList
 In icms, select the following
 - Setup
 - Environment
 The following features are described below:
 - stopViewList: A list of cellview names that specifies the leaf cells
 - switchViewList: A list of cellview names that binds the netlist procedures in the design hierarchy
 Typical setting is as follows:


```
stopViewList=hsim hspices
switchViewList=hsim hspices schematic
```

Removing hsim/hsimD SimInfo

Commands for removing hsim/hsimD SimInfo fields from the database of primitives are also provided. They are:

```
nsdRemoveSimInfo(libName @ Optional (target "hsim"))
```

The argument target in the procedure is optional where hsim is the default. If another argument replaces hsim, it will be used instead.

The following command removes hsimD SimInfo from the libName library:

```
nsdRemoveSimInfo("libName" "hsimD")
```

The following command removes hsim SimInfo for the liba library:

```
nsdRemoveSimInfo("liba")
```

Netlist Procedures for Component Primitives

The netlistProcedure field defines which netlist procedure to call for the primitive. The HSIM-Virtuoso Interface supplies the five classes of netlist procedures for component primitives that use fields in the hsim/hsimD SimInfo

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Netlist Procedures for Component Primitives

section of each of the primitives shown in [Table 111 on page 814](#).

Table 111 HSIM Netlist Primitives

Class	Netlist Primitive	Explanation
I.	HSIMCompPrim	For components which a device model is not necessary; such as, resistors, capacitors, and inductors.
II.	HSIMDevPrim	For components which requires a device model, such as bjt, diode, MOSFET etc. Warning messages are printed if no correspondent models found in the design.
III.	HSIMCompPrimRef	For controlled source components, that is, vccs, vcvs, cccs, ccvs, vcres and vccap.

Table 111 HSIM Netlist Primitives (Continued)

Class	Netlist Primitive	Explanation
IV.	HSIMSrcPrim_XXX	<p>This is a set of procedures for independent sources, where XXX is the name of the source.</p> <p>Examples:</p> <p>The netlist procedure for a DC current source (idc) is HSIMSrcComp_idc. The HSIM-Virtuoso Interface supports the following independent sources:</p> <p>HSIMSrcPrim_idc</p> <p>HSIMSrcPrim_ipulse</p> <p>HSIMSrcPrim_isin</p> <p>HSIMSrcPrim_isffm</p> <p>HSIMSrcPrim_iam</p> <p>HSIMSrcPrim_iexp</p> <p>HSIMSrcPrim_ipwl</p> <p>HSIMSrcPrim_ipwlf</p> <p>HSIMSrcPrim_vdc</p> <p>HSIMSrcPrim_vpulse</p> <p>HSIMSrcPrim_vsin</p> <p>HSIMSrcPrim_vsffm</p> <p>HSIMSrcPrim_vam</p> <p>HSIMSrcPrim_vexp</p> <p>HSIMSrcPrim_vpwl</p> <p>HSIMSrcPrim_vpwlf</p> <p>HSIMSrcPrim_vpwlz</p> <p>HSIMSrcPrim (for vsource & isource)</p>

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Netlist Procedures for Component Primitives

Table 111 HSIM Netlist Primitives (Continued)

Class	Netlist Primitive	Explanation
V.	HSIMSubcktCall	For netlist macro primitives. The primitive is defined as a user-supplied netlist. The namePrefix and componentName should be set as X and sub-circuit. The name of the correspondent macro is defined in the CDF parameter macro of the primitive.

The procedure selected depends on the primitive characteristics to be netlisted as described in the table.

HSIMD Netlist Procedures for Component Primitives

Unlike the socket interface, most primitives do not need to be assigned with a netlist procedure, except primitives shown in [Table 112 on page 816](#).

Table 112 HSIMD Netlist Primitives

Lib	Cell	HSIMD Netlist Procedure
analogLib	vcss, vccs	hsimDVCPPrim
	ccvs, cccs	hsimDCCPrim
	dc, pulse, sin, exp	hsimDSrcPrim
	pwl	hsimDPwlSrcPrim

instParameters Field

The instParameters field is a list of parameters that are included in the netlist with the primitive. The parameter name in the list is the Component Description Format name used by the primitive. It is not necessarily the name recognized by HSIM. If it is not the same name, additional information must be added to map the unrecognized name to the one that HSIM recognizes in propMapping. Refer to [propMapping Field](#) below.

componentName Field

The netlister refers to the componentName field for the type of component specified.

termOrder Field

The termOrder field is a list of terminals that define the terminal order of the primitive being netlisted. Terminal names are defined in the symbol view of the primitive.

propMapping Field

The propMapping field records the map between the Component Description Format parameter name and the HSIM recognized name. It has the form of:

```
nil <hsim parameter name 1> <user's CDF name1>
<hsim parameter name 2> <user's CDF name2>
<hsim parameter name 3> <user's CDF name3>...
```

The netlister uses this map to translate Cadence parameters to HSIM parameters.

namePrefix Field

The namePrefix field defines the netlist primitive prefix. HSIM uses this to determine which component type is in the netlist file.

Setting up hsim SimInfo fields for a user defined component:

In this example, a capacitor named ucap is built. Assume that ucap has Cadence parameter Cap, TC_1, TC_2 where the following occur:

Cap

 Capacitance

TC_1

 Temperature coefficient 1

TC_2

 Temperature coefficient 2 ucap has two terminals: plus (+) and minus (-).

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Netlist Procedures for Component Primitives

Refer to the Virtuoso Analog Design Environment user documentation for additional component setup details.

Since the device model for a capacitor is optional in HSIM, HSIMCompPrim is selected as the netlistProcedure. It is assumed that only c and tc_1 must be included in the netlist therefore, instParameters are c and tc_1.

Since ucap is a capacitor so the componentName is cap. Plus is the leading terminal for this component hence, termOrder (Terminal Order) is Plus Minus. Since Cap and TC_1 are not the parameter names that HSIM recognizes, a propMapping field is required. The translation pair for the parameters are Cap<->c and TC_1<->tc1, therefore, propMapping is nil c Cap tc1 TC_1.

Finally, this component is netlisted using namePrefix C as it is a capacitor to HSIM. The hsim SimInfo fields for this component are shown in [Table 113 on page 818](#).

Table 113 Example HSIM SimInfo Fields

netlist name	hsim/hsimD SimInfo name
netlistProcedure	HSIMCompPrim
instParameters	c tc_1
componentName	cap
termOrder	Plus Minus
propMapping	nil c Cap tc1 TC_1
namePrefix	C

For hsimD, keep netlistProcedure empty since hsimD uses the default netlist procedures provided by Cadence for primitive elements like capacitor, resistor, etc.

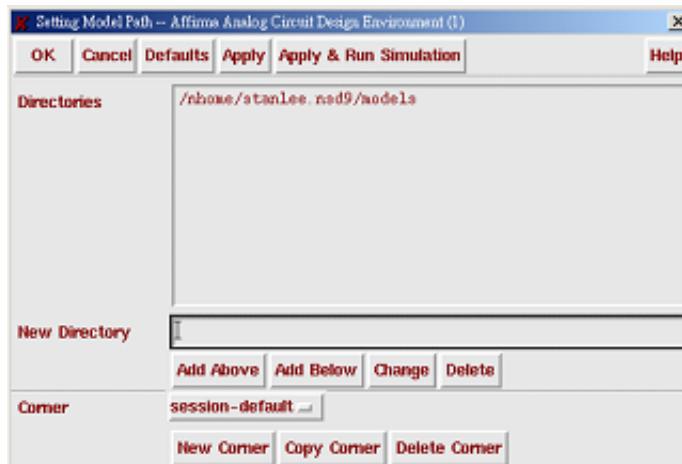
namePrefix Field

A set of component primitives is supported by the interface and installed into the default Cadence analogLib library. The installation script generates the necessary hsim/hsimD SimInfo field for them.

Models, Macros, and Include Files

Models

Device models in a design are specified in individual files and appended to the final netlist. For hsim, the netlist procedures will search these files accordingly in a user-defined list of model paths. For hsimD, the netlister will search the definition of models in a user-defined list of model libraries. hsim (socket) specifies the model paths by adding the paths where these models are located. To set the model path, select Setup-> Model Path from the screen shown in the HSIM-Virtuoso Interface banner menu.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 75 Set Model Path Window

Using this approach, each model file can contain only one device model. The file name uses a fixed naming convention that adds a .m to the end of the models name.

A model n-MOSFET is named nmos.m. A model file is a plain text file that can be edited by a common editor such as UNIX vi. To edit a model file, select the following in order:

Setup -> Simulation Files -> Edit Model File

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Models, Macros, and Include Files

The file is written in native HSIM model syntax with minor modifications as shown in the following syntax example:

```
*  
* The comments start with an *  
*.model &1 NMOS $After '$' is an in-line component  
+Level=49 $which will be dropped in the final netlist  
+Tnom=27.0& $An & at the end of a line is for line continuation  
+Nch=2.498+E17 & $$An & at the end of a line is for line continuation  
+Tox-9E-09 $A + at the beginning of a line is  
+Lint=9.36e-8 $continuation from the previous.  
+Vth0=.6322 K1=.756 K2=-3.83e-2 K3=-2.612  
+Dvt0= 2.812 Dvt1=0.462 Dvt2=-9.17e-2  
+Nlx=3.523E-08 $The rest are omitted from this example
```

In this example, comments and line continuation syntax are remarked. To avoid model name clashes in a design hierarchy during netlisting, the model name is replaced with the special tag &1.

hsimD (Direct)

Instead of specifying the paths to where model files are located, the model libraries to be used for designs should be specified. To specify a model library, complete the following steps:

1. Select Setup > Model Libraries from the Banner Menu.
2. Set Model Library Window.
3. In the Model Library Window, specify the library files and the section to be used for a design.

Macros

Macros are user-supplied sub-circuit definitions for component primitives in socket flow. Macros are not used by the direct flow of hsimD. The netlist procedure for these primitives are HSIMSubcktCall.

The macro definitions to be included in the final netlist. HSIMSubcktCall gathers the name of the macro definition from the Component Description Format parameter macro of the primitive being netlisted. The procedure searches for the definition file in the same paths specified for model files. For the details setting up the path, refer to n in [Figure 75 on page 819](#).

Each model file can contain only one macro or sub-circuit definition. The file name uses a fixed naming convention that adds a .s to the end of the models name such that a model nmoscap is named nmoscap.s. A model file is plain text file that can be edited by common editor such as vi in Solaris.

The file is written in native HSIM netlist syntax with minor modifications as shown in [Example 81](#). Attention is drawn to the replacement macro name with the special tag &1. This is to avoid macro name clashes in a design hierarchy during netlisting.

Example 81 Macro Definition

```
* This is an example of a macro definition
* This subcircuit is called nmoscap.s
* This macro simulates a capacitor using an nmos .
.subckt &1 PLUS MINUS B
.param w='20u' l='20u' as='5p' ad='5p' ps='15u'
+ pd='15u'
*declaration of the parameters is for parameter override.
MC0 MINUL PLUS MINUL B nch w='w' l='l' as='as'
+ ad='ad' ps='ps' pd='pd'
.ends &1
**CAUTION* The model name for devices in macro definition
*will not be manipulated
* Macro/subcircuit definition shares the same line
* continuation and comments style as model files.
```

Include File

The include file is only used by hsim. Apart from giving individual definitions for models and sub-circuits, all predefined definitions can be lumped into a single include file. Users must confirm that the name of the models and sub-circuits are cross referenced in both the include file and design schematics. A graphical file browser is provided to make inputting the include file list as described in [Environment Setup on page 771](#).

Net Name Conversion Macro

The interface attempts to keep the following user-defined names as identical as possible in both the schematic drawing and netlist.

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Assigning HSIM Parameters

- Net names
- Instance names
- Model names

In some circumstances, the names MUST be changed due to ambiguities. These changes are referred to as name mapping. If the names are specified in both the macro and include files, correct connection may be lost. When this occurs, a name conversion macro is required to ensure correct name mapping. The net to be mapped in the macro and include files must be bracketed as shown in the following example:

[# and]

The macro then has a specific [#netname] format that forces the interface net netname to be filtered by the name mapping procedure.

Expansion of pPar, iPar

Expansion of pPar and iPar is accomplished as follows:

pPar(x)

Refers to the value of the parameter from the parent instance of the current instance. To ensure that the resulting netlist is well ordered, the HSIM-Virtuoso Interface utilizes the HSIM parser parameter override mechanism so that after checking the x parameter exists in the parent instance, x replaces pPar(x).

iPar(x)

Is evaluated as the parameter x value of for the instance being netlisted. the HSIM-Virtuoso Interface replaces the iPar(x) function with the value of x.

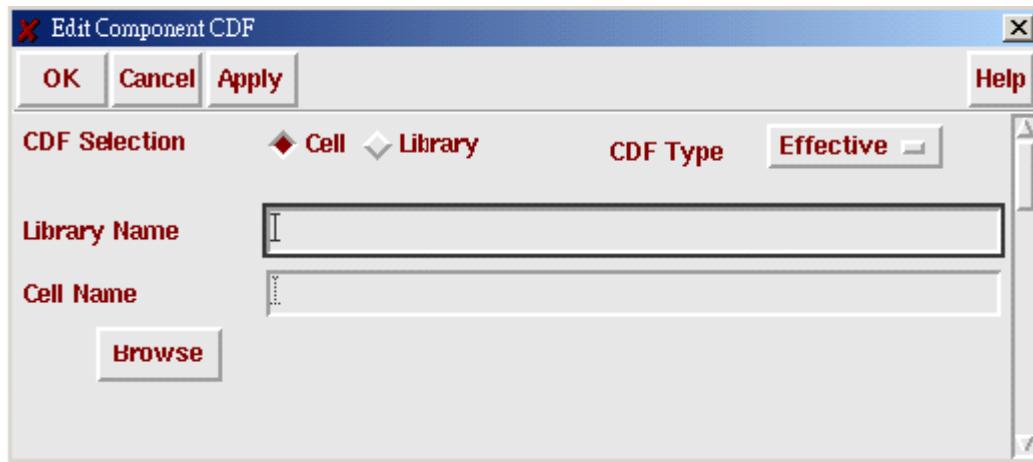
Assigning HSIM Parameters

HSIM parameters can be assigned using any of following methods:

- Sub-circuit
- Instance
- To instance whose sub-circuit has been assigned with same HSIM parameter in CDF.

Assigning HSIM Parameters for Subcircuit

Assigning HSIM parameters for a sub-circuit is accomplished through the Component Description Format as shown in the figure below.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 76 Component Description Format Window

To assign HSIM parameters for a subcircuit, follow these steps:

1. In CIW, select Tools -> CDF -> Edit to bring up the Edit Component CDF window.
2. Select one of the following to specify where the HSIM parameter is to be assigned in the same window for the target sub-circuit cell:
 - Library
 - Cell
3. Select the CDF Type to specify the following information:
 - Effective: The change is effective for this instance only.
 - User: The change will be effective for one user only.
 - Base: The change will be effective for all users.
4. In the same window, select Add for component parameters. An Add CDF Parameter window appears.

For example, if HSIMSPEED=5 is assigned for this specific sub-circuit, the following changes should be made:

Appendix G: HSIM-Virtuoso Interface Advanced Topics

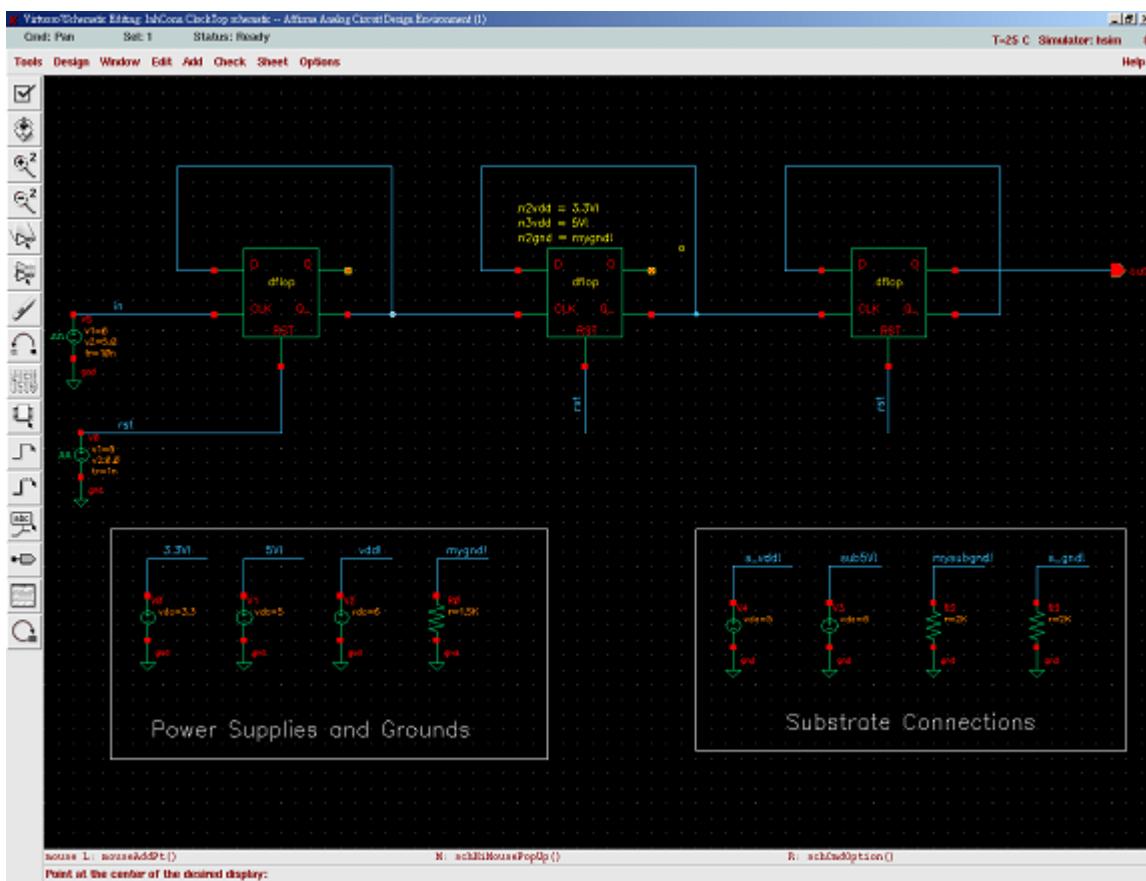
Assigning HSIM Parameters

- paramType: is changed to int
 - name: is changed to HSIMSPEED
 - defValue: is changed to 5
5. Click [OK] or [Apply] for both windows. The netlist is displayed as follows:

```
.subckt [Subckt Name] <port> HSIMSPEED=5  
....  
.ends
```

Assigning an HSIM Instance Parameter

Assigning an HSIM parameter for an instance must be done using the Virtuoso Schematic Editor. For the example illustrated in [Figure 77](#), the ClockTop design in the InhConn library is used.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 77 Virtuoso Schematic Editor Window

Figure 77 on page 825, shows the ClockTop design in the Virtuoso Schematic Editor. I3 of the dflop will be assigned to the parameter described later in this section. This is located on the LEFT of the ClockTop cell.

To assign an HSIM instance parameter, follow these steps:

1. Select I3 using the LEFT mouse button.
2. Either click the [Property] button in Equalizer shape cell or select the following in the order shown:

Edit -> Properties -> Objects...

An Edit Object Properties window appears.

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Assigning HSIM Parameters

3. In the Edit Object Properties window, select Add to display the Add Property window.
4. In the Add Property window, enter the HSIM parameter to be added.

For example, if HSIMABMOS=1 is to be added, enter the following in the Add Property window:

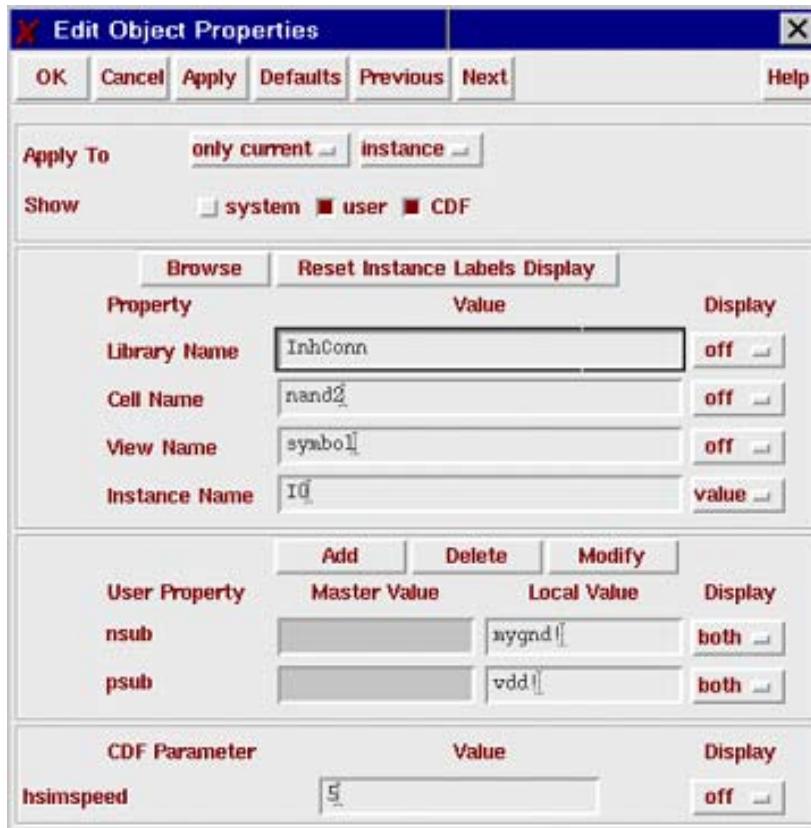
- Name HSIMABMOS
- Type int
- Value 1
- [OK] or [Apply] For both Add Property and Edit Object Property windows

The netlist will be displayed as follows:

```
Xname <portlist> <subcktname> HSIMABMOS=1
```

Assigning an HSIM Parameter to an Instance with a Subcircuit (Cell) Assigned the Same HSIM Parameter in Component Description Format

This method is similar to the method described in [Assigning an HSIM Instance Parameter on page 824](#) for an instance shown above however, a parameter does not need to be added. The parameter already exists when the Edit Object Properties window appears. This window is shown in [Figure 78](#).



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 78 Edit Object Properties Window

The properties for I0 (nand2) in I7 (dflop) of the top cell. Its defValue of HSIMSPEED=5, as set in Step 4 in [Assigning HSIM Parameters for Subcircuit on page 823](#). The value can be changed to any other value through the instance base Edit Object Properties window.

Naming Conventions

The following naming limitations apply to nets and elements:

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Naming Conventions

1. The net name can not be 0 (zero). This is to avoid ambiguity with the nomenclature for a ground. The HSIM-Virtuoso Interface maps zero as follows:

Net_Named_0

It will not be treated as ground in HSIM. Specify the ground net using the following SPICE command:

vxx ground_net gnd 0

2. When the first character of a net name is unacceptable to HSIM, it is mapped to another character. For example, '+' is mapped to 'n'. In [Table 114 on page 828](#), first characters of net names are mapped to 'n'.

Table 114 First Characters of Net Names Mapped to 'n'

Character	Definition	Character	Definition
+	Plus sign	\$	Dollar sign
,	Comma	.	Period
(Open parenthesis)	Close parenthesis
[Open bracket	Close bracket	

3. If the first character of an element name is one of those shown in [Table 115 on page 828](#), it is automatically mapped to 'x':

Table 115 First Characters of Element Names Mapped to 'x'

Character	Definition	Character	Definition
+	Plus sign	\$	Dollar sign
,	Comma	.	Period
(Open parenthesis)	Close parenthesis
[Open bracket]	Close bracket
<	Open (left) angle bracket	>	Close (right) angle bracket
!	Exclamation mark		

4. If the first character of a net names is an asterisk (*), it is removed.
5. If the first character of an element is an underline (_) symbol, it is automatically mapped to x_.
6. The characters shown in [Table 116 on page 829](#) are not allowed in either a net or element name. If they appear they will be mapped to “ ”.

Table 116 Characters Not Allowed in Net or Element Names Mapped to “ ”

Character	Definition	Character	Definition
,	Comma	\$	Dollar sign
(Open parenthesis)	Close parenthesis
[Open bracket]	Close bracket
<	Open (left) angle bracket	>	Close (right) angle bracket
!	Exclamation mark	*	Asterisk
.	Period	:	Colon

7. If an element/net name has a pipe (|) or number/pound (#) character, it is mapped to the underscore (_) character.
8. If the first character of a model name is one of those shown in [Table 117 on page 829](#), it is automatically mapped to ‘m’:

Table 117 First Characters in Model Names Mapped to ‘m’

Character	Definition	Character	Definition
+	Plus sign	\$	Dollar sign
,	Comma	.	Period
(Open parenthesis)	Close parenthesis
[Open bracket]	Close bracket
<	Open (left) angle bracket	>	Close (right) angle bracket
!	Exclamation mark		

Appendix G: HSIM-Virtuoso Interface Advanced Topics
HSIM-Virtuoso Interface Ocean Script Command Usage

9. If the first character of a model name is one of those shown in [Table 118 on page 830](#), it is automatically added as the second character of the model name with 'm' as the first character: for example, 1model would become m1model.

Table 118 First Characters Added as the First Character of a Model Name

Character	Definition	Character	Definition
_	Underscore	1 ... 9	Numerals 1 through 9
0	Zero		

10. All uppercase letters will be mapped to lowercase letters in the model name.
11. The characters shown in [Table 119 on page 830](#), will be mapped to " " if they appear in the model name.

Table 119 Model Name Characters Mapped to " "

Character	Definition	Character	Definition
+	Plus sign	!	Exclamation mark
,	Comma	\$	Dollar sign
(Open parenthesis)	Close parenthesis
[Open bracket]	Close bracket
<	Open (left) angle bracket	>	Close (right) angle bracket

HSIM-Virtuoso Interface Ocean Script Command Usage

The “HSIMOcean” package can be applied to all integration approaches such as “hsim”, “hsimD”, “aanni” and “aaCosim”. To enable HSIMOcean, specify the HSIM-Virtuoso Interface initialization setup in your local .oceanrc file as follows:

```
path = getShellEnvVar("HSIM_ARTISTIF")
nsdaAAIMPkgList='("aanni")
load("$HSIM_ARTISTIF/install/nsdaAAIMInvoker.il")'
```

Public APIs for “HSIMOcean” package

There are seven Application Interface (API) functions available for public use:

Note: The order of these functions in the ocean script file is important.
Please issue these API functions in the same order as they are listed below.

1. `nsdOcnEnvSetup()`

This function performs the following environment setup tasks:

- a. checks out an AANNI license,
- b. initializes the HSIM environment variables, and
- c. initialize the HSIM parameters.

2. `nsdOcnSetHsimParam(symName value)`

This function sets user-defined values for HSIM parameters and environment variables. All of the HSIM parameters in the existing Interface GUI and the following HSIM environment variables are supported in this API:

- `netlistSyntax`
- `hsimOutputFileNamePrefix`
- `isHsimCaseSensitive`
- `hsimCommandLinePrefix`
- `hsimCommandOptions`
- `hsimUsing64Bit`
- `hsimCktCheck`
- `hsimDeviceV`

If an HSIM parameter is set to a value different than its default value, then HSIMOcean writes it out as part of the Ocean script.

3. `nsdOcnCreateHostNetlist()`

This function determines if host netlist exists and tries to create it if it is missing.

4. `nsdOcnCreateTopNetlist()`

Appendix G: HSIM-Virtuoso Interface Advanced Topics

HSIM-Virtuoso Interface Ocean Script Command Usage

This function creates a top level netlist (`hsim.netlist`) for HSIM according to the setup of the HSIM parameters and environment variables.

Note: The host netlist is included in this top level netlist file

5. `nsdOcnCreateNetlist()`

This is a group function that consists of `nsdOcnCreateHostNetlist()` and `nsdOcnCreateTopNetlist()`. If you want to perform both of these functions at the same time, use `nsdOcnCreateNetlist()`.

6. `nsdOcnRunHsim()`

This function creates and executes the simulation run file, `runHsim`.

7. `nsdOcnFinishing()`

This function finishes up after simulation by checking-in the AANNI license.

Ocean Script Example

A complete example of HSIMOcean script is shown in [Example 82](#).

Note: The ocean script is modified from the default script generated by CDS. Some of CDS function calls have been replaced with `nsdOcn-` function calls.

Example 82 HSIMOcean Script

```

ocnWaveformTool( 'wavescan' )
;----- Choose Host Simulator ----->>>
simulator( 'spectreVerilog' )

;----- Setup Design/Analysis/ResultsDir/etc ----->>>
design( "/remote/us01home4/jeffhu/simulation/top/
spectreVerilog/configSpectreVerilog/netlist/analog/netlist" )
resultsDir( "/remote/us01home4/jeffhu/simulation/top/
spectreVerilog/configSpectreVerilog" )
modelFile(
    '("/remote/us01home4/jeffhu/Jason/aacosim_demo/models/
spectre/pll.scs" "")'
)

analysis('tran ?stop "5u"   ')
desVar(  "c1" 100p)
desVar(  "c2" 85f)
desVar(  "r1" 47K)
desVar(  "r2" 4.7K)
desVar(  "wgain" 40u)
option('temp  "27.0"  )

;----- Save results ----->>>
save( 'v "/load"  )
temp( 27.0  )

;----- Setup AANNI environment ----->>>
nsdOcnEnvSetup()

;----- Setting up Hsim parameters/Env. variables ----->>>
nsdOcnSetHsimParam('hsimresultdir "/remote/us01home4/jeffhu/
simulation/top/spectreVerilog/configSpectreVerilog/psf/"')
nsdOcnSetHsimParam('hostsimulator "spectreVerilog"')
nsdOcnSetHsimParam('plotitemtable '('/load net'))')
nsdOcnSetHsimParam('digitalsimulator "NC-Sim"')
nsdOcnSetHsimParam('ishsimcasesensitive "no"')
nsdOcnSetHsimParam('hsimoutput "psfbin"')

;----- Create Host/Top Netlist ----->>>
nsdOcnCreateNetlist()

;----- Start simulation ----->>>
nsdOcnRunHsim()

;----- Open results ----->>>
openResults("/remote/us01home4/jeffhu/simulation/top/
spectreVerilog/configSpectreVerilog/psf")
```

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Socket (HSIM) and Direct (HSIMD) Integration

```
;<<<----- Select results ----->>>
selectResult( 'tran' )

;<<<----- Plot results ----->>>
plot(nsdGetData("/load"))

;<<<----- Finishing up ----->>>
nsdOcnFinishing()
```

Socket (HSIM) and Direct (HSIMD) Integration

This section describes the installation and usage of HSIM socket and HSIMD direct integration, which are the traditional ways to integrate HSIM with Cadence's Virtuoso Analog Design Environment. The interface is developed on Cadence Design Framework II 4.4.5. and has been tested on the following versions:

- 4.4.5
- 4.4.6
- 5.0.0
- 5.1.41

Note: Only the major elements of HSIM and HSIMD installation and usage are described in this section. Refer to [Appendix G, HSIM-Virtuoso Interface Advanced Topics](#) for detailed information.

Installing Socket (HSIM) and Direct (HSIMD) Interfaces

The installation process requires that the HSIM/HSIMD Simulation Information (SimInfo) fields be added. Site administrators are advised to make backups of the Cadence Software hierarchy before proceeding. Customers must have the appropriate write permissions to modify the Cadence environment.

As with the Native Netlist Integration installation, you must first obtain the AAIM package and complete the following steps:

1. Ensure the HSIM executable and Cadence software are in the search path.
2. Set HSIM_HOME environment to the root of HSIM release tree.

3. Set the HSIM_ARTISTIF environment to the root of the AAIM release tree.
4. Add hsim and/or hsimd into the list nsdaAAIMPkgList in .cdsinit file, as in this example:

```
nsdaAAIMPkgList='("hsim" "hsimd" "aanni" ...)
```

5. Create the menus directory. It can be either of the following:

- <user_home_directory>/menus
- <user_working_directory>/menus

6. Copy menus from the following to the directory created in [Step 4](#):

For HSIM Socket:

```
cp <AAIM-Install-Tree-Root>/menus/hsim.menu
```

For hsimd Direct:

```
cp <AAIM-Install-Tree-Root>/menus/hsimd.menu
```

7. Create hsim/hsimd view and Component Description Format hsim/hsimd SimInfo for the primitive libraries.

hsim/hsimd SimInfo is required for each primitive in order to perform design netlisting in the HSIM-Virtuoso Interface. Refer to Modifying hsim/hsimd SimInfo in [Appendix G, HSIM-Virtuoso Interface Advanced Topics](#), for instructions on generating the information required for individual primitives.

Note: It is recommended that the menu files and .cdsenv for hsim/hsimd can be placed in your CDS tree. Menu files should be placed under <CadenceDir>/etc/tools/menus and .cdsenv should be placed under either <CadenceDir>/etc/tools/hsim or <CadenceDir>/etc/tools/hsimd. Both files are included in the AAIM installation package.

8. Confirm that installation is successful.

Successful installation can be confirmed through Cadence log. A log window is displayed upon successfully registering the Interface as shown in [Figure 79](#).

Feedback

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Socket (HSIM) and Direct (HSIMD) Integration

```
ifb - Log: /home/stanlee/msd9/CDS.log
File Tools Options Technology File Help 1
COPYRIGHT © 1992-1999 CADENCE DESIGN SYSTEMS INC. ALL RIGHTS RESERVED.
© 1992-1999 UNIX SYSTEMS Laboratories INC..
Reproduced with permission.
This Cadence Design Systems program and online documentation are
proprietary/confidential information and may be disclosed/used only
as authorized in a license agreement controlling such use and disclosure.
RESTRICTED RIGHTS NOTICE (SHORT FORM)
Use/reproduction/disclosure is subject to restriction
set forth at FAR 1252.227-19 or its equivalent.
Program:          e(#$)SCDS: ifcb.exe version 4.4.5 11/16/1999 15:06 (cds11182) $
Sub version:     sub-version 4.4.5.42
Loading PRshare.cxt
Loading auCore.cxt
Loading schView.cxt
Loading selectSv.cxt
END OF SITE CUSTOMIZATION
Loading ./ cdsinit init file from the site init file.
***-----***  
HSIM-AAI : Loading common procedures and declarations...
Loading oasis.cxt
Loading analog.cxt
Loading cosspacei.cxt
HSIM-AAI : REGISTRATION, Ver. 3.0-135901182004
* Tracking No - Analog Artist Interface 2004.02.0.
HSIM-AAI : End of loading
***-----***  
LOADING SX-Cadence Link Package for Analog Artist...
    --- VV-CDS LINK. 2003.2 BUILD 1054 (04/07/2003)    ---
    --- COPYRIGHT 2001-2003 SANDWORK DESIGN INC.    ---
END OF LOADING SX-Cadence Link Package for Analog Artist
***-----***  
HSIMD-AAI : Loading common procedures and declarations...
Loading asimenv.cxt
Loading spectra...
HSIMD-AAI : REGISTRATION, Ver. 3.0-135901182004
* Tracking No - Analog Artist Interface 2004.02.0.
HSIMD-AAI : End of loading
***-----***  
I  
MOUSE L: M: R:  
>
```

Figure 79 Registration Confirmation Window

Porting Existing Design

The outline for exporting existing design hierarchy to the HSIM-Virtuoso Interface is described below. Refer to [Appendix G, HSIM-Virtuoso Interface Advanced Topics](#), for detailed procedures.

To export an existing design, use the following steps:

1. Collect all primitives for the design including, but not limited to, the following components:
 - Resistors
 - MOSFETs

- BJTs
 - Voltage sources (controlled)
 - Current sources (controlled)
2. Verify that hsim/hsimD view exists for primitives listed above. If not, create the view by using the nsdCreateView procedure. Refer to Generating hsim/hsimD View and SimInfo in [Appendix G, HSIM-Virtuoso Interface Advanced Topics](#) for information using nsdCreateView.
Note: Simulator views from other vendors can also be specified using the AAI switch/stop view feature.
 3. Edit the hsim/hsimD SimInfo fields for these primitives. Refer to [Appendix G, HSIM-Virtuoso Interface Advanced Topics](#) for information on editing hsim/hsimD SimInfo.
 4. Verify the netlist.

Netlist and Simulation

For using HSIM socket and direct interface to run simulation, you must generate a netlist from the schematic design. Data preparation procedures need to be conducted for each primitive device to have required simInfo, such as netlistProcedure, ready for the netlisting process.

The Interface adopts the default Cadence schema as closely as possible. Refer to Cadence's Virtuoso Analog Design Environment user documentation for information on the schema.

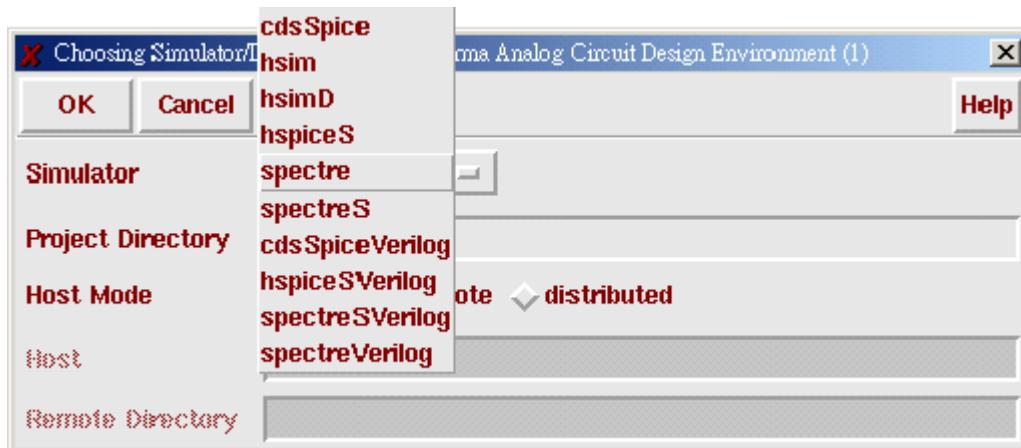
Starting the GUI and Selecting HSIM

To start the GUI, follow these steps:

1. Select the following in order as shown in [Figure 80](#):
Tools -> Analog Environment -> Simulation
2. Select one of the following:
 - hsim: as the simulator for socket integration.
 - hsimD: as the simulator for direct integration.

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Socket (HSIM) and Direct (HSIMD) Integration



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 80 Choosing Simulation Directory Host Window

Specifying a Host Machine

The HSIM-Virtuoso Interface permits selection of a local or remote host to run HSIM. Special attention should be used if the remote mode is selected. The following issues must be considered when choosing to run under a remote host:

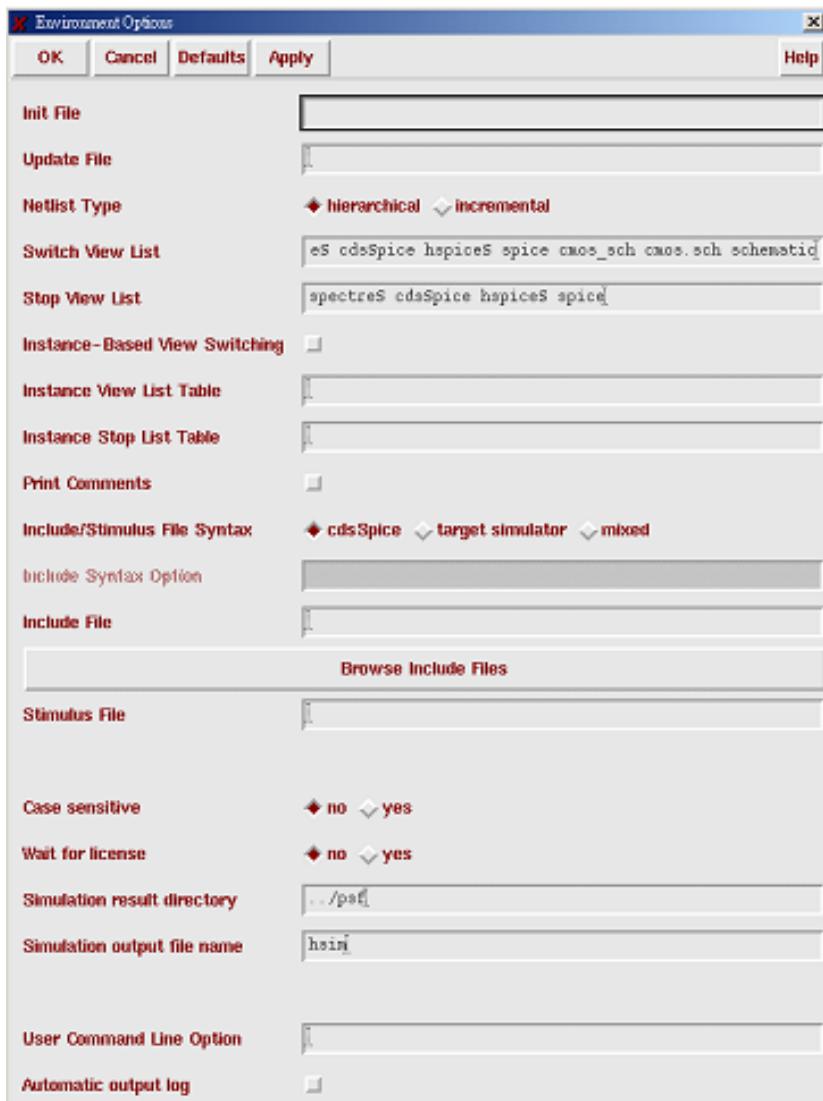
- The remote machine must be setup to grant permissions from the local machine without authentication. For example: .rhosts
- The remote machine must see an identical file hierarchy under the directory path entered in it's Remote Directory as the local machine sees under it's Project Directory.

Setup Environment

To display the Environment Options window, select the following in order:

Setup -> Environment

The GUIs shown in [Figure 81](#) and [Figure 82](#) is used to set up the Interface environment for HSIM and HSIMD respectively.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 81 Environment Options Window for HSIM Socket

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Socket (HSIM) and Direct (HSIMD) Integration



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 82 Environment Options Window for HSIM Direct

Netlister Settings

For the netlister to traverse the design database correctly, it is important that the following options be correctly set:

- Stop View List: A list of cellview names that specify the leaf cells.
- Switch View List: A list of cellview names that bind the netlist procedures in the design hierarchy.

Typical HSIM socket settings include the following:

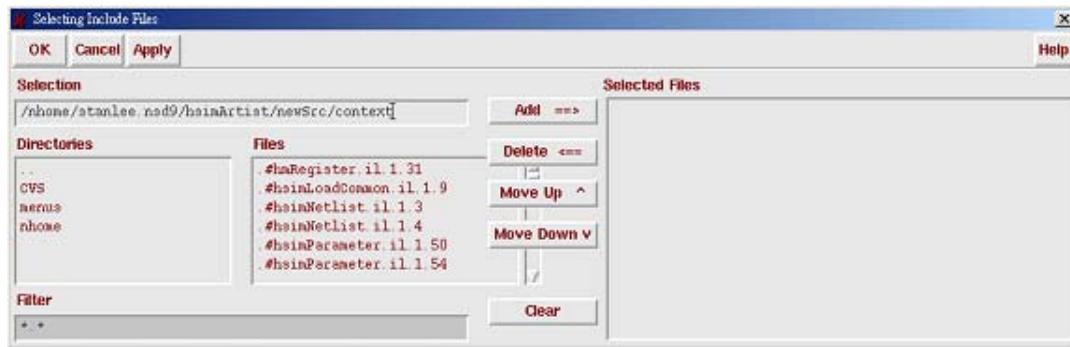
```
Stop View List: hsim hspiceS  
Switch View List: hsim hspiceS schematic
```

In [Figure 81](#), the Include/Stimulus File Syntax field setting specifies the syntax of the include stimulus, macro, and model files. The Interface's netlister performs a syntax check if target simulator is chosen. Special attention should be paid if cdsSpice is selected because the syntax check is handled by cdsSpice and is should conform with the cdsSpice rules.

Files listed in both the Include and Stimulus files are treated equally. They are inserted line-by-line into the final netlist and the net name macro is expanded. Refer to [Appendix G, HSIM-Virtuoso Interface Advanced Topics, Net Name](#)

[Conversion Macro on page 821.](#)

Select the [Browse Include Files] button and a graphical file browser is displayed as shown in [Figure 83](#). The files to be included can be browsed and selected from this window.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
 Printed with permission.

Figure 83 Selecting Include Files Window

Note: Different result directories and output file names for multiple simulations can be set up. They are specified in the following parts of the Environment Options form:

- Simulation Results directory
- Simulation output file name fields

The Results Browser can be used to inspect the results of different simulations. The associate netlist is also backed up in the Simulation Results directory.

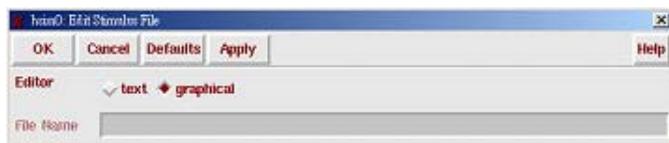
Graphically Editing Stimulus Files

Stimulus files can be graphically edited using the GUI shown in [Figure 84](#). Signals for Stimulus are automatically extracted when the radio button in the Edit Stimulus File window is activated.

Feedback

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Socket (HSIM) and Direct (HSIMD) Integration



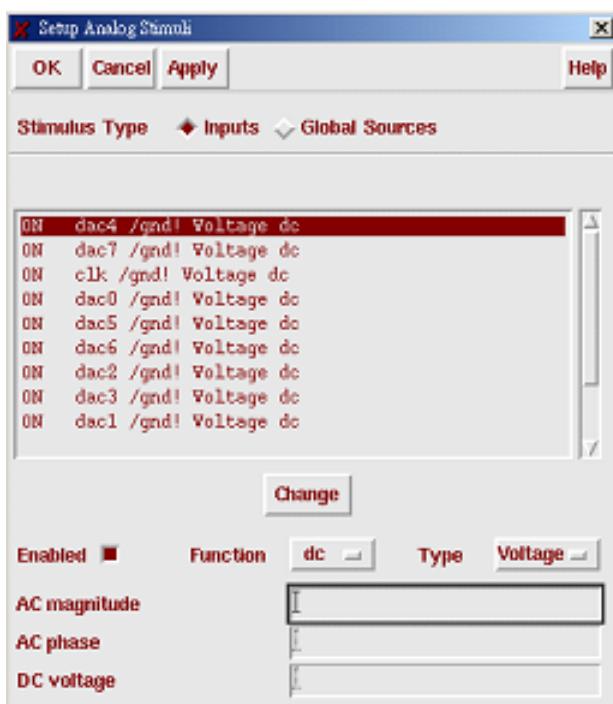
© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 84 Edit Stimulus File Window

To display the Setup Analog Stimuli window, select the following in order:

Setup -> Stimulus -> Edit Analog

The available signals are listed in the Graphical Stimulus editor shown in the figure below.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

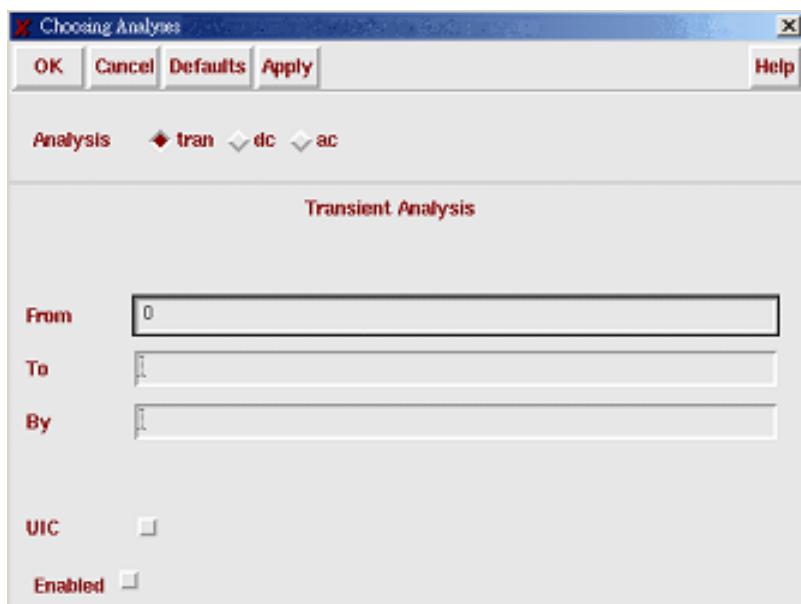
Figure 85 Setup Analog Stimuli Window

Stimulus functions can be changed using the cyclic button of Function. The following Stimulus functions are not supported:

- pwlf
- sffm

Analysis

hsim/hsimD supports AC, DC, and Transient analysis. Selecting the desired analysis process is accomplished by selecting the analysis type in the Virtuoso Analog Design Environment window shown in [Figure 86](#).



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 86 Choosing Analysis - Virtuoso Analog Design Environment Window

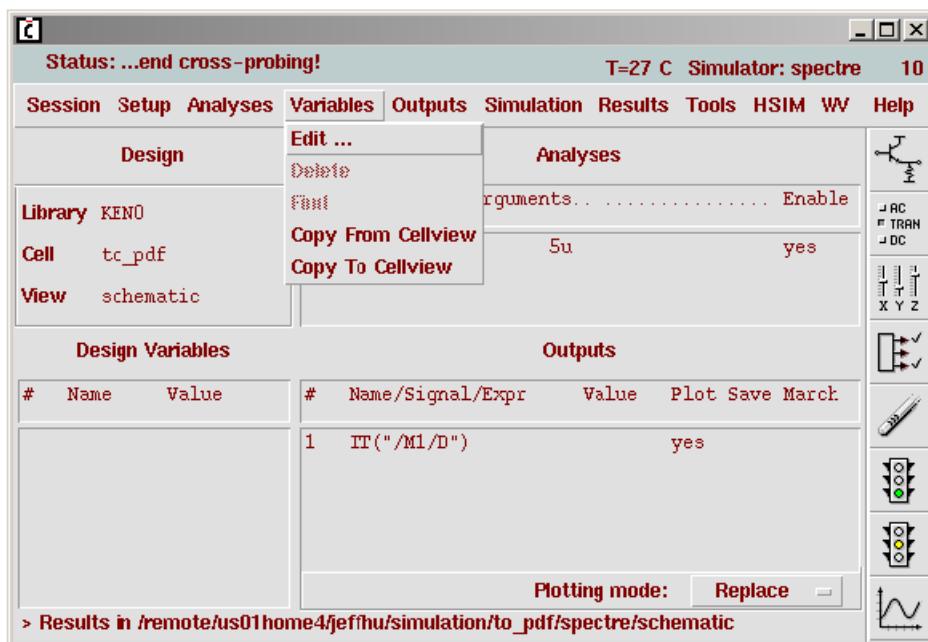
To select transient analysis, perform the following steps:

1. Navigate to the appropriate page by choosing Analysis > Choose > .
2. Click on the tran radio button.
3. Specify the simulation end time in the To field.
4. Specify the simulation time step in the By field.

Appendix G: HSIM-Virtuoso Interface Advanced Topics
Socket (HSIM) and Direct (HSIMD) Integration

Design Variables

Design variables are defined in the Component Description Format parameters through the design hierarchy. Design variables can be specified for each simulator run. Design variables are obtained using the GUI shown in [Figure 87](#).



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 87 Virtuoso Analog Design Environment(1) Window

Perform the following steps:

1. Select Variables.
2. Copy from Cellview: Existing design variables are displayed in the Design Variables list box.
3. Select any variable: A new pop-up window will be displayed as shown in [Figure 88](#).



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 88 Editing Design Variables Window

4. Set the desired design variable values

Simulator Options

The simulator options are mostly HSPICE compatible. Setup the Interface options using the GUI shown in [Figure 89](#).

Feedback

Appendix G: HSIM-Virtuoso Interface Advanced Topics Socket (HSIM) and Direct (HSIMD) Integration

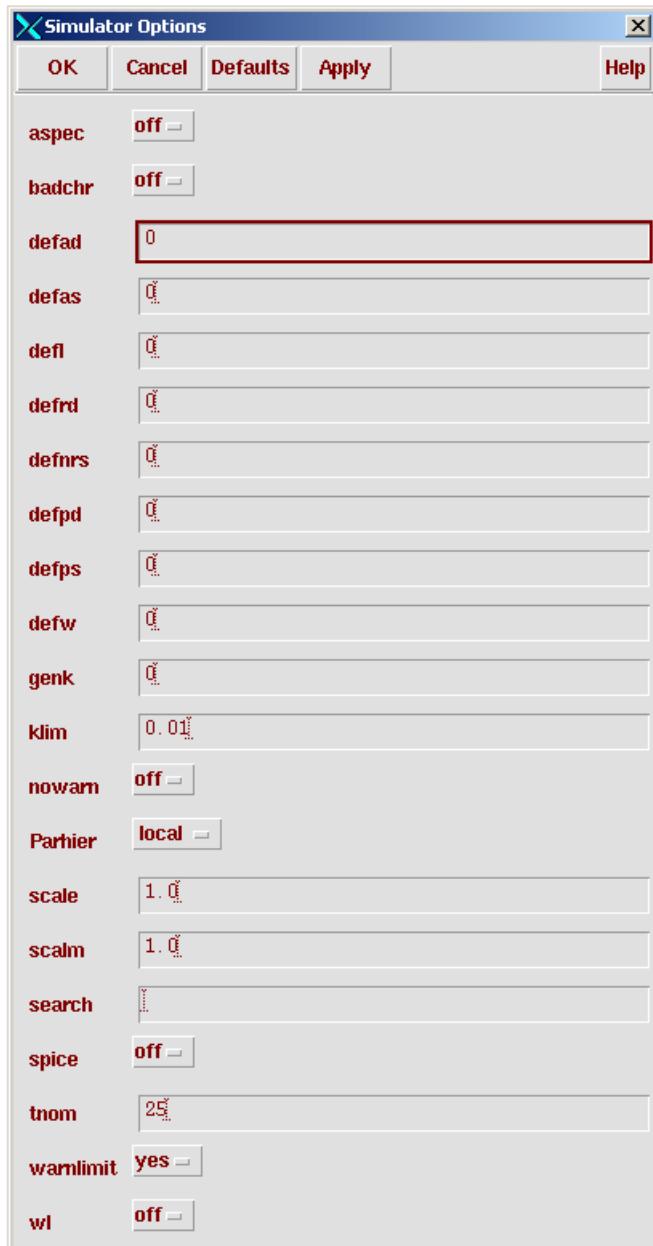


Figure 89 Simulator Options Window

To set up options, perform the following steps:

1. Navigate to: Simulation -> Options -> Analog...; a popup form is displayed.

2. Enter the desired values. The filled (filled or (filled-in or entered)) options are netlisted as .option statements in the final netlist.
3. Save or load option settings using the Save State Simulator Options/Load State option in the Session menu.

HSIM Parameters

The HSIM-Virtuoso Interface provides a large set of HSIM-specific parameters to fine tune HSIM performance. For efficiency and user convenience, the HSIM parameters are separated into two groups:

- Basic HSIM Parameters contain only the most frequently used commands.
- Advanced HSIM Parameters include the rest of the commands and are categorized by functionality.

Feedback

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Socket (HSIM) and Direct (HSIMD) Integration

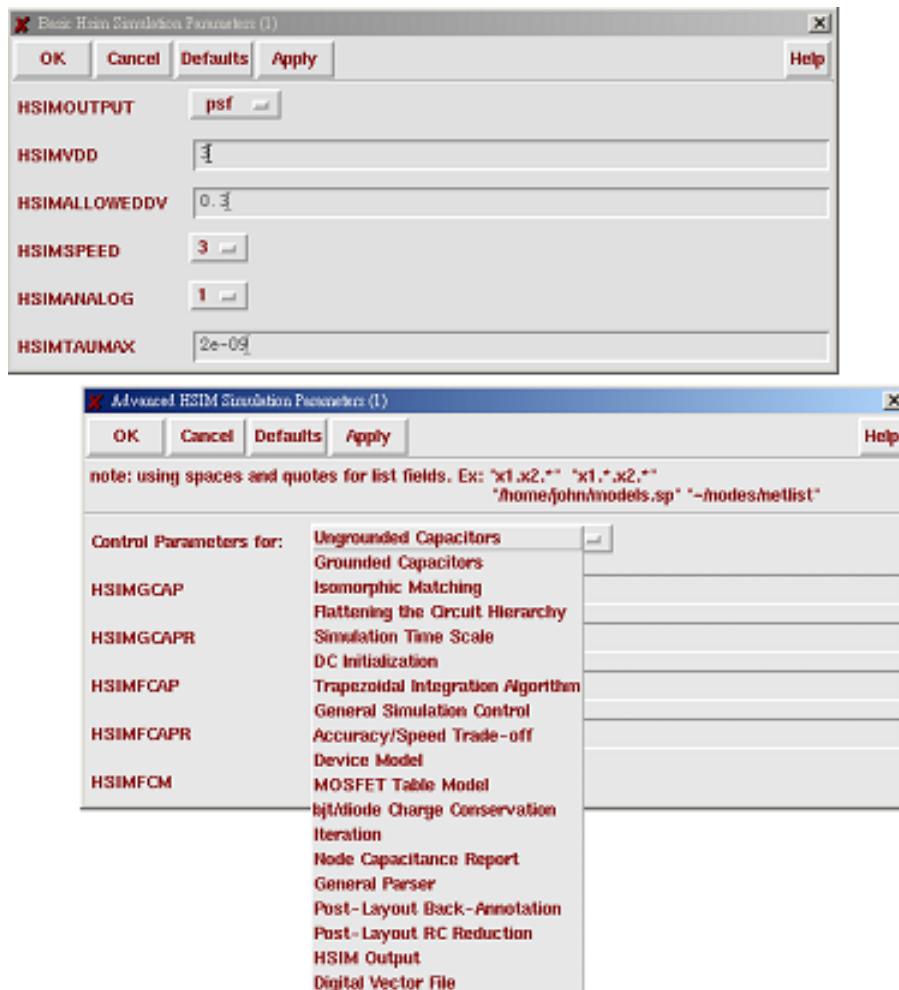


Figure 90 Basic and Advanced HSIM Parameters Forms

To set up set parameters, select the following in order:

1. Navigate to: Simulation -> HSIM Parameters -> Basic; for basic HSIM parameters or, Simulation -> HSIM Parameters -> Advanced; for advanced HSIM parameters.
2. Enter the desired parameters The parameters are netlisted as .param statements in the final netlist.

The GUI in Figure 90 only sets HSIM parameters for the top cell. It is possible to specify parameters in the sub-hierarchy using the netlisting Component Description Format parameters associated with sub-circuits.

Refer to [Appendix G, HSIM-Virtuoso Interface Advanced Topics](#) for details.

3. Save or load HSIM specific parameter settings using the Save State Simulator Parameters/Load State option in the Session menu.

Selecting Data to Save or Plot

Refer to the Cadence Virtuoso Analog Design Environment user documentation for details of this operation.

To select data to save or plot, perform the following steps:

1. Navigate to either of the following to display the Virtuoso Schematic Editor window:
 - Outputs -> To Be Saved -> On schematic
 - Outputs -> To Be Plotted -> On schematicThe Virtuoso Schematic Editor window is displayed.
2. Use the LEFT mouse button to select the desired nets or nodes. The signals selected to be plotted will be displayed automatically when the simulation is completed.
3. View saved signals using the Tools menu in the Results Browser.

Timing and Power Checks

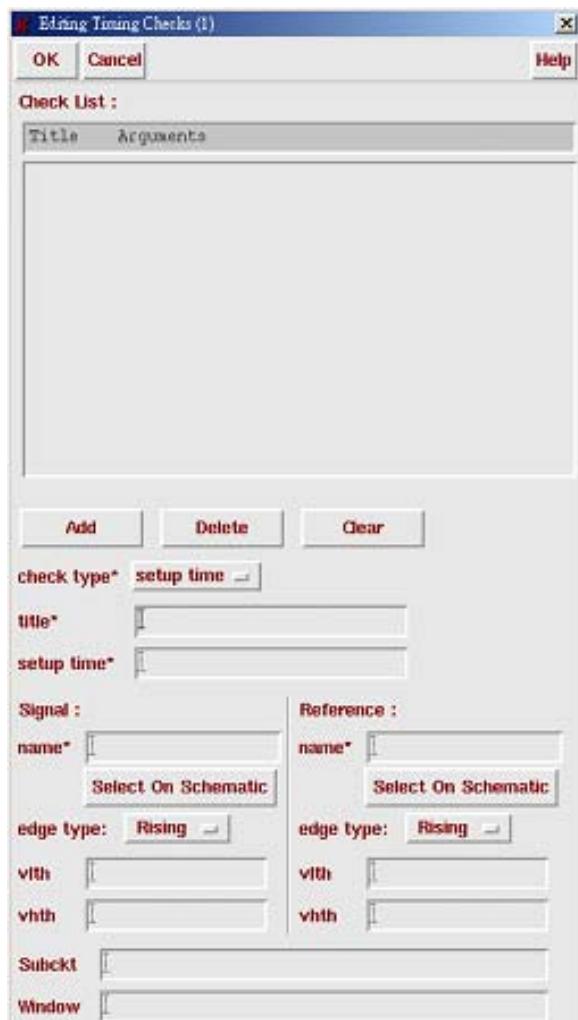
To set timing and power checks, choose Select Outputs -> Timing Checks or Power Checks. Depending on which option is selected, one of the GUI's described in the following figures is displayed:

- [Figure 91](#)
- [Figure 92](#)

Feedback

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Socket (HSIM) and Direct (HSIMD) Integration



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 91 Editing Timing Checks Window



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 92 Editing Power Checks Window

Power and timing check settings can be saved for future use or loaded from the previous session using load/save state facilities. Refer to [Load and Save Sessions on page 855](#).

Appendix G: HSIM-Virtuoso Interface Advanced Topics

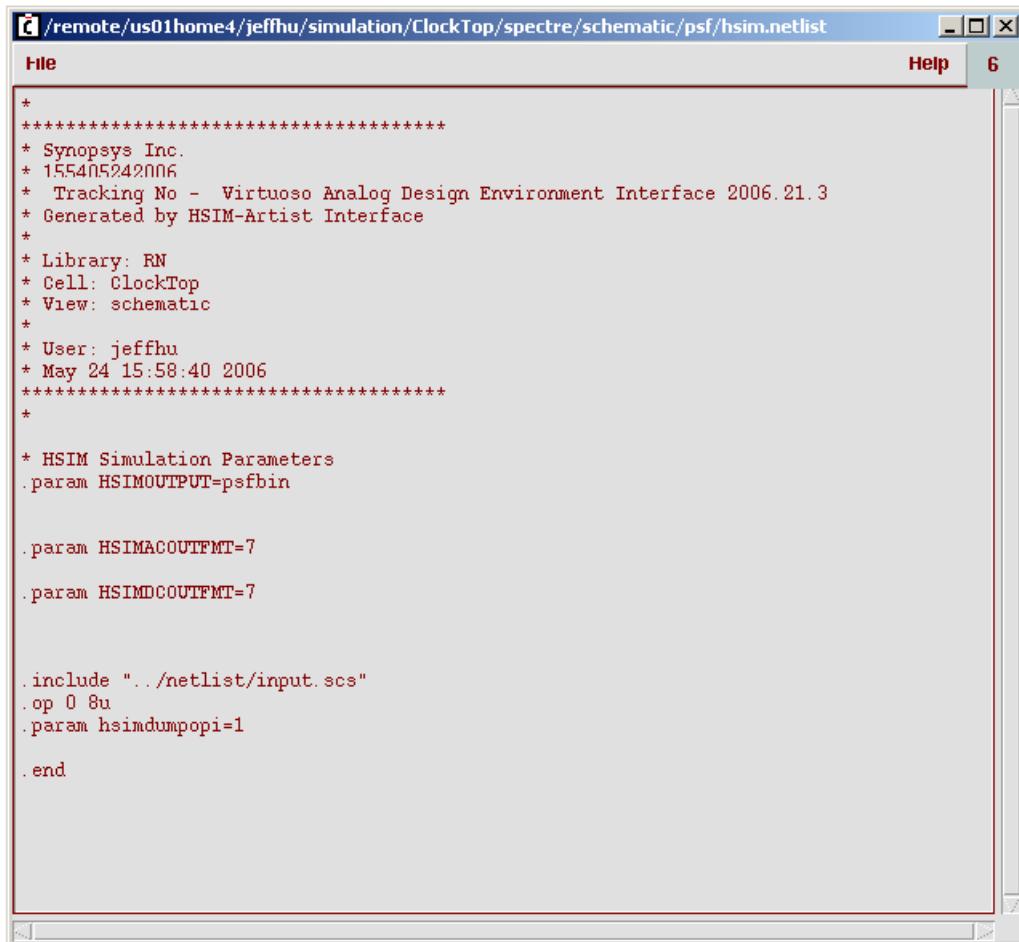
Socket (HSIM) and Direct (HSIMD) Integration

Generating Netlists

To generate a netlist, follow these steps:

1. Select Simulation -> Netlist.
2. Select one of the following:
 - Re-Netlist: Performs the entire netlisting procedure.
 - Create Netlist: Updates the netlist result incrementally based on environment or parameter changes.
 - Display Netlist: Displays the netlist.

The netlist will be displayed in the window shown in [Figure 93](#).



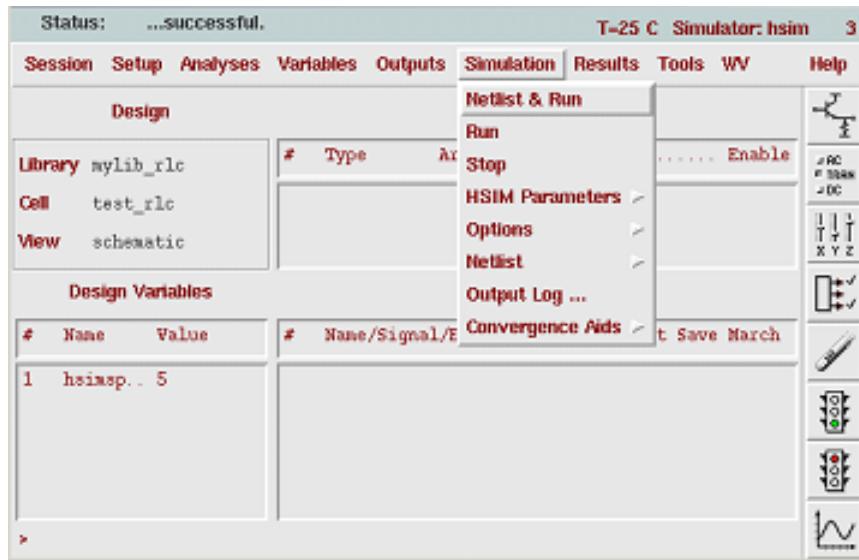
A screenshot of a Windows command-line window titled '/remote/us01home4/jeffhu/simulation/ClockTop/spectre/schematic/psf/hsim.netlist'. The window contains a text-based netlist script. The script includes header information such as the Synopsys tracking number (155405242006), the Virtuoso Analog Design Environment version (2006.21.3), and the generation source (HSIM-Artist Interface). It also specifies simulation parameters like HSIMOUTPUT=psfbins, HSIMACOUTFMT=7, and HSIMDCOUTFMT=7. The script concludes with an include directive for 'input.scs', an op command with parameters 0 8u, and a param hsimdumppop=1, followed by an .end command.

```
*  
*****  
* Synopsys Inc.  
* 155405242006  
* Tracking No - Virtuoso Analog Design Environment Interface 2006.21.3  
* Generated by HSIM-Artist Interface  
*  
* Library: RN  
* Cell: ClockTop  
* View: schematic  
*  
* User: jeffhu  
* May 24 15:58:40 2006  
*****  
  
* HSIM Simulation Parameters  
.param HSIMOUTPUT=psfbins  
  
.param HSIMACOUTFMT=7  
.param HSIMDCOUTFMT=7  
  
.include "../netlist/input.scs"  
.op 0 8u  
.param hsimdumppop=1  
  
.end
```

Figure 93 Netlist Result Window

Running Simulations

The GUI shown in [Figure 94](#) is used to invoke simulation.



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
 Printed with permission.

Figure 94 Running Simulations Window

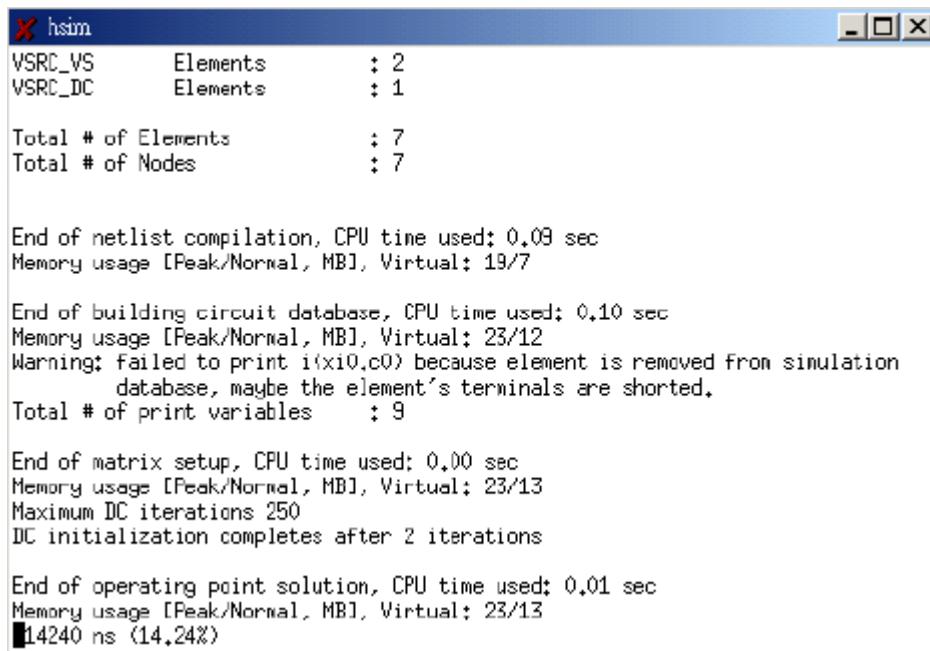
To invoke a simulation, select either of the following:

- GO 
- (Green) traffic light to run the simulation.
- Simulation -> Run

An xterm window is displayed where simulation status can be monitored as shown in [Figure 95](#). From this window, the HSIM simulation process can be interrupted using [Ctrl] [C], which enters the interactive mode.

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Socket (HSIM) and Direct (HSIMD) Integration



```
hsim
VSRC_VS      Elements      : 2
VSRC_DC      Elements      : 1

Total # of Elements      : 7
Total # of Nodes         : 7

End of netlist compilation, CPU time used: 0.09 sec
Memory usage [Peak/Normal, MB], Virtual: 19/7

End of building circuit database, CPU time used: 0.10 sec
Memory usage [Peak/Normal, MB], Virtual: 23/12
Warning: Failed to print i(x10,c0) because element is removed from simulation
          database, maybe the element's terminals are shorted.
Total # of print variables   : 9

End of matrix setup, CPU time used: 0.00 sec
Memory usage [Peak/Normal, MB], Virtual: 23/13
Maximum DC iterations 250
DC initialization completes after 2 iterations

End of operating point solution, CPU time used: 0.01 sec
Memory usage [Peak/Normal, MB], Virtual: 23/13
14240 ns (14.24%)
```

Figure 95 XTERM Status Window

Simulation can be aborted by selecting either of the following:

- STOP

- (Red) traffic light to Stop the simulation.
- Simulation -> Stop

Viewing Results

Refer to the Cadence Virtuoso Analog Design Environment user documentation for details of this operation.

Waveforms

Waveform results are automatically displayed in a pop-up window for signals saved as plotted when a simulation finishes. Saved signals can also be viewed by the Results Browser under the Tools menu to check waveforms for signals marked as saved. The HSIM-Virtuoso Interface saves results from multiple

simulations. The Results Browser can also be used to switch among the multiple results.

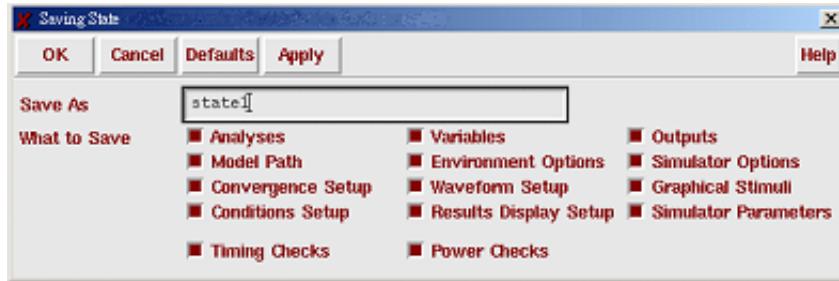
Annotation

Simulation results can be annotated to schematics. Refer to Cadence user documentation for details on annotating simulation results.

Note: Annotation is only possible for the most recent simulation results.

Load and Save Sessions

The HSIM-Virtuoso Interface permits a session to be saved for future reference using the GUI shown in [Figure 96](#).



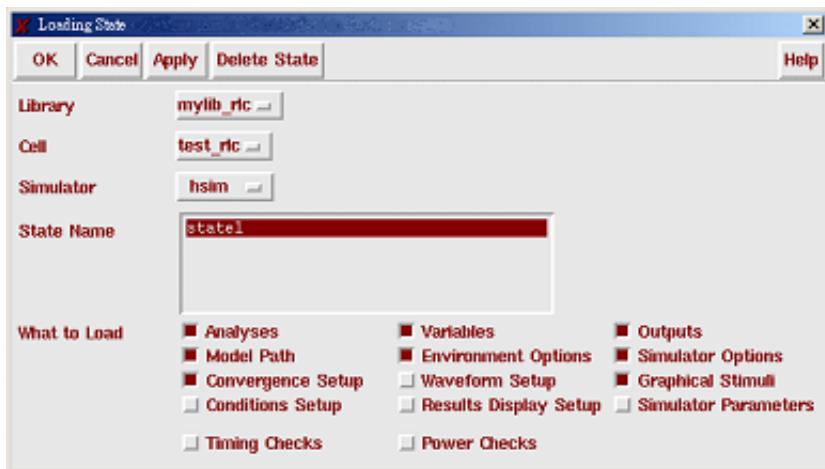
© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 96 Save a Session Window

To save a session, choose Session -> Save State. Sessions can be loaded using the GUI shown in [Figure 97](#).

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Monte Carlo Analysis



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 97 Load a Session Window

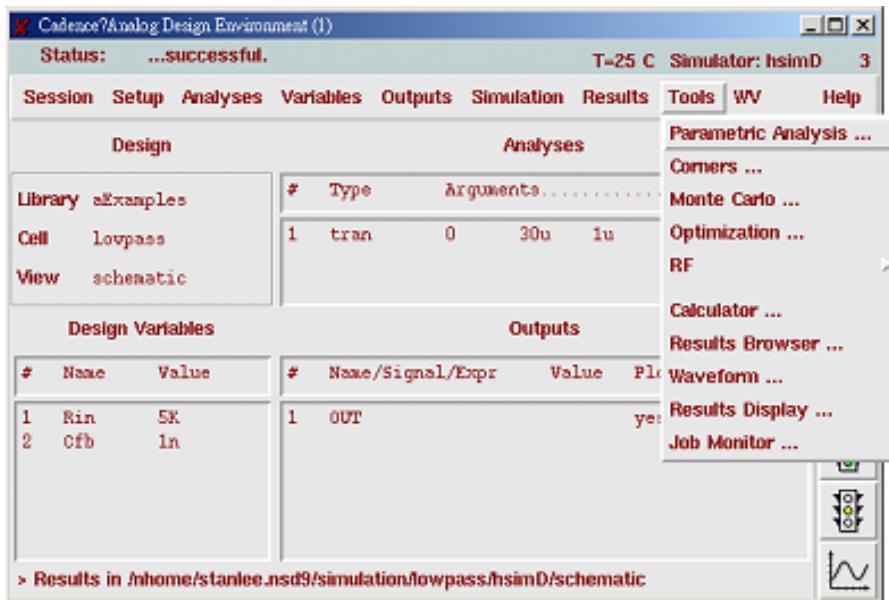
To load a session, choose Session -> Load State.

Monte Carlo Analysis

HSIM's Monte Carlo Analysis function is currently supported only using direct integration (HsimD).

To use Monte Carlo analysis, perform the following steps:

1. Select HsimD as the simulator.
2. Select Tools->Monte Carlo, as shown in [Figure 98](#).



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

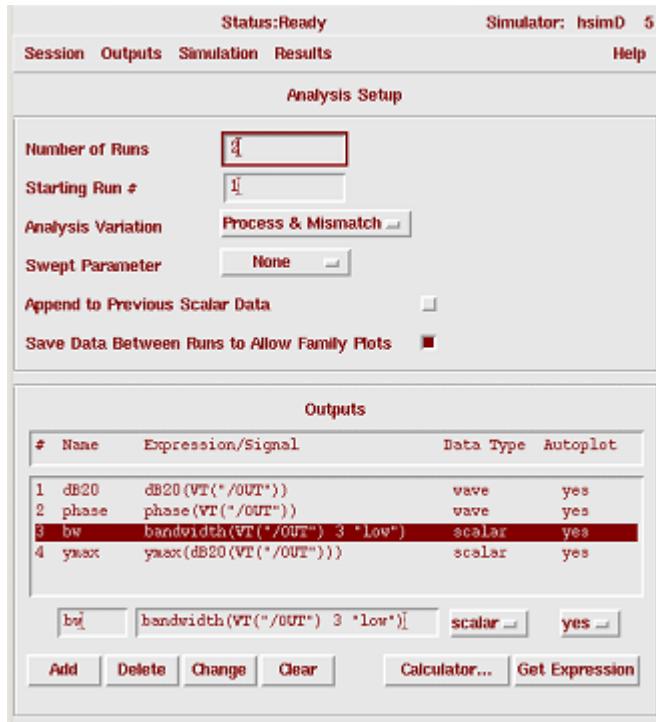
Figure 98 Virtuoso Analog Design Environment Window

This displays the Virtuoso Statistical Analysis form. Usages and setups on this form are inherited from the generic Cadence form shown in [Figure 99](#).

Feedback

Appendix G: HSIM-Virtuoso Interface Advanced Topics

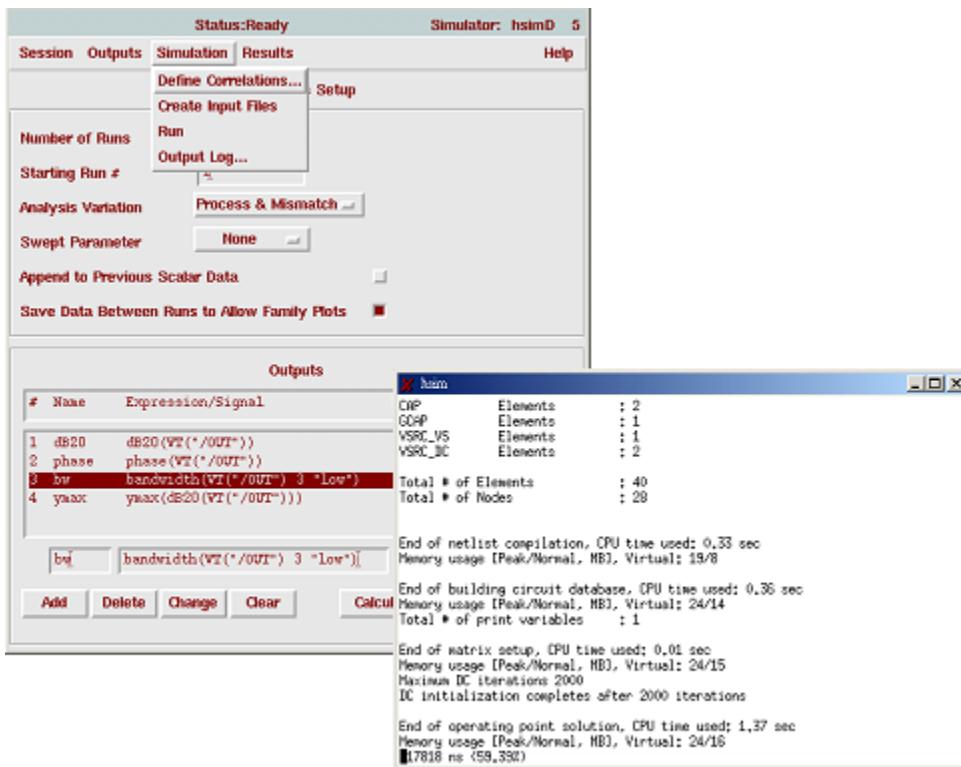
Monte Carlo Analysis



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 99 Virtuoso Analog Statistical Analysis Window

3. Invoke simulation using Statistical Analysis->Simulation->Run. A shell console will pop up as shown in [Figure 100](#).
4. Once simulation completes, the selected output data is displayed as shown in [Figure 101](#).



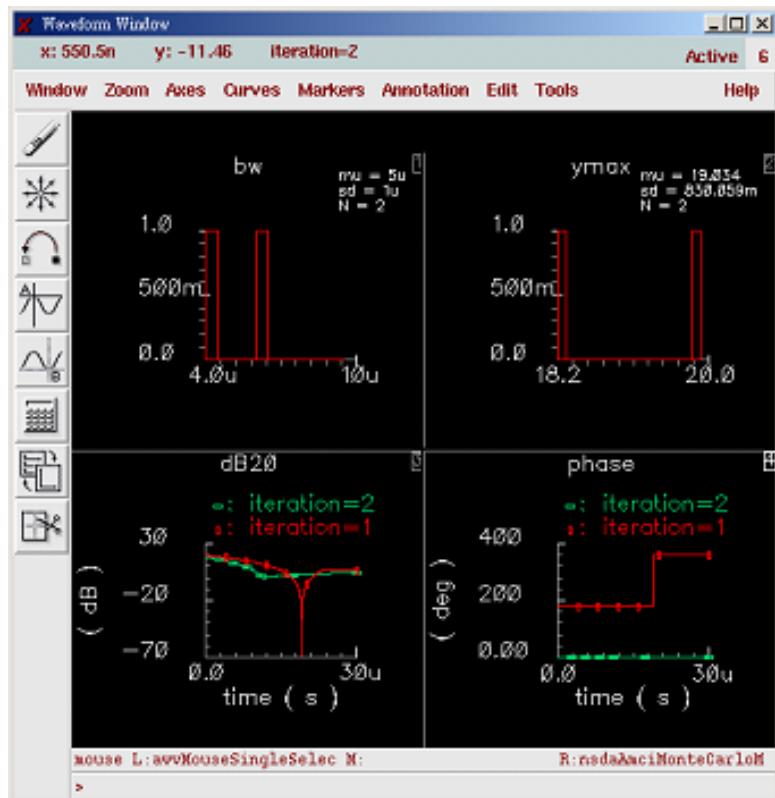
© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 100 Simulation Window

Feedback

Appendix G: HSIM-Virtuoso Interface Advanced Topics

Monte Carlo Analysis



© 2007, Cadence Design Systems, Inc. All rights reserved worldwide.
Printed with permission.

Figure 101 Waveform Output Window for Monte Carlo Analysis

HSIM-Virtuoso Interface Netlist Properties

The appendix provides a list of the HSIM-Virtuoso Interface netlist properties and provides syntax examples and explanations.

HSIM Netlist Properties

The table below describes the netlist properties associated with HSIM.

Table 120 HSIM Netlist Properties

Primitive	Function	Default Parameter
res	netlistProcedure:	HSIMCompPrim
	instParameters:	r tc1 tc2 scale m dtemp l w
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	res
	namePrefix:	R
presistor	netlistProcedure:	HSIMCompPrim
	instParameters:	r tc1 tc2 scale m dtemp l w
	termOrder	PLUS MINUS
	propMapping:	
	componentName:	presistor

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIM Netlist Properties*Table 120 HSIM Netlist Properties (Continued)*

Primitive	Function	Default Parameter
	namePrefix:	R
cap	netlistProcedure:	HSIMCompPrim
	instParameters:	c tc1 tc2 scale m dtemp ic l w
	termOrder:	PLUS MINUS
	propMapping:	cap
	componentName:	
	namePrefix:	C
pcapacitor	netlistProcedure:	HSIMCompPrim
	instParameters:	c tc1 tc2 scale m dtemp ic l w
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	pcapacitor
	namePrefix:	C
ind	netlistProcedure:	HSIMCompPrim
	instParameters:	l tc1 tc2 scale m dtemp ic
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	ind
	namePrefix:	L
mind	netlistProcedure:	HSIMCompPrim
	instParameters:	k

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termOrder:	
	propMapping:	
	componentName:	mind
	namePrefix:	K
vdc	netlistProcedure:	HSIMSrcPrim_vdc
	instParameters:	vdc
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	vsrc
	namePrefix:	V
idc	netlistProcedure:	HSIMSrcPrim_idc
	instParameters:	idc m
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I
vpulse	netlistProcedure:	HSIMSrcPrim_vpulse
	instParameters:	v1 v2 td tr tf pw per
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	vsrc

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIM Netlist Properties*Table 120 HSIM Netlist Properties (Continued)*

Primitive	Function	Default Parameter
	namePrefix:	V
ipulse	netlistProcedure:	HSIMSrcPrim_ipulse
	instParameters:	i1 i2 td tr tf pw per
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I
vpwl	netlistProcedure:	HSIMSrcPrim_vpwl
	instParameters:	rpt td tvpairs t1 v1 t2 v2 t3 v3
	termOrder:	PLUS MINUS
		t4 v4 ... t50 v50
	propMapping:	
	componentName:	vsrc
	namePrefix:	V
ipwl	netlistProcedure:	HSIMSrcPrim_ipwl
	instParameters:	rpt td tvpairs t1 i1 t2 i2 t3 i3
		t4 i4 ... t50 v50
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
vsin	netlistProcedure: instParameters: termOrder: propMapping: componentName: namePrefix:	HSIMSrcPrim_vsin vo va freq td theta PLUS MINUS vsr V
isin	netlistProcedure: instParameters: termOrder: propMapping: componentName: namePrefix:	HSIMSrcPrim_isin io ia freq td theta PLUS MINUS isr I
vexp	netlistProcedure: instParameters: termOrder: propMapping: componentName: namePrefix:	HSIMSrcPrim_vexp v1 v2 td1 td2 tau1 tau2 PLUS MINUS vsr V
iexp	netlistProcedure: instParameters:	HSIMSrcPrim_iexp i1 i2 td1 td2 tau1 tau2

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIM Netlist Properties

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
vcvs	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	isrc
	namePrefix:	I
	netlistProcedure:	HSIMCompPrimRef
	instParameters:	csType hegain maxv minv delta htd xypairs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x18 x20 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	PLUS MINUS
	refTermOrder:	NC+ NC-
	propMapping:	nil max maxv min minv td htd
	componentName:	vcvs
	namePrefix:	E
vccs	netlistProcedure:	HSIMCompPrimRef
	instParameters:	csType hggain maxi mini delta xypairs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x18 x20 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
ccvs		y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	PLUS MINUS
	refTermOrder:	NC+ NC-
	propMapping:	nil max maxi min mini
	componentName:	vccs
	namePrefix:	G
	netlistProcedure:	HSIMCompPrimRef
	instParameters:	csType hhgain vref maxv minv delta xypairs
		x1 x2 x3 x4 x5 x6 x7 x8 x9 x10
		x11 x12 x13 x14 x15 x16 x17 x18 x18 x20
cccs		y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
		y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	PLUS MINUS
	propMapping:	nil max maxv min minv
	componentName:	cccs
	namePrefix:	H
	netlistProcedure:	
	instParameters:	HSIMCompPrimRef delta xypairs
		x1 x2 x3 x4 x5 x6 x7 x8 x9 x10

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIM Netlist Properties*Table 120 HSIM Netlist Properties (Continued)*

Primitive	Function	Default Parameter
		x11 x12 x13 x14 x15 x16 x17 x18 x18 x20
		y1 y2 y3 y4 y5 y6 y7 y8 y9 y10
		y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	PLUS MINUS
	propMapping:	nil max maxi min mini
	componentName:	cccs
	namePrefix:	F
tline	netlistProcedure:	HSIMCompPrim
	instParameters:	z0 td len f nl
	termOrder:	IN+ IN- OUT+ OUT-
	propMapping:	nil l len
	componentName:	tline
	namePrefix:	T
diode	netlistProcedure:	HSIMDevPrim
	instParameters:	area w l pj m ic
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	diode
	namePrefix:	D
pdiode	netlistProcedure:	HSIMDevPrim
	instParameters:	area w l pj m ic

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
zener	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	diode
	namePrefix:	D
	netlistProcedure:	HSIMDevPrim
	instParameters:	area w l pj m ic
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	diode
	namePrefix:	D
schottky	netlistProcedure:	HSIMDevPrim
	instParameters:	area w l pj m ic
	termOrder:	PLUS MINUS
	propMapping:	nil vd Vd
	componentName:	diode
	namePrefix:	D
	netlistProcedure:	HSIMDevPrim
	instParameters:	area areab areac m
	termOrder:	C B E
	propMapping:	
npn	componentName:	npn

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIM Netlist Properties*Table 120 HSIM Netlist Properties (Continued)*

Primitive	Function	Default Parameter
	namePrefix:	Q
pnp	netlistProcedure:	HSIMDevPrim
	instParameters:	area areab areac m
	termOrder:	C B E
	propMapping:	
	componentName:	npn
	namePrefix:	Q
njfet	netlistProcedure:	HSIMDevPrim
	instParameters:	area l w m dtemp
	termOrder:	D G S progn(bn)
	propMapping:	nil vds Vds vgs Vgs vgbs Vgbs
	componentName:	njfet
	namePrefix:	J
pjfet	netlistProcedure:	HSIMDevPrim
	instParameters:	area l w m dtemp
	termOrder:	D G S progn(bn)
	propMapping:	nil vds Vds vgs Vgs vgbs Vgbs
	componentName:	njfet
	namePrefix:	J
nmos	netlistProcedure:	HSIMDevPrim
	instParameters:	l w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
pmos	termOrder:	D G S progn(bn)
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
	netlistProcedure:	HSIMDevPrim
	instParameters:	I w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S progn(bn)
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
nmos4	netlistProcedure:	HSIMDevPrim
	instParameters:	I w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S B
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
	netlistProcedure:	HSIMDevPrim
	instParameters:	I w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S B
	propMapping:	

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIM Netlist Properties

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	componentName:	mosfet
	namePrefix:	M
nbsim	netlistProcedure:	HSIMDevPrim
	instParameters:	I w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S progn(bn)
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
pbsim	netlistProcedure:	HSIMDevPrim
	instParameters:	I w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S progn(bn)
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
nbsim4	netlistProcedure:	HSIMDevPrim
	instParameters:	I w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S B
	propMapping:	
	componentName:	mosfet
	namePrefix:	M

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
pbsim4	netlistProcedure:	HSIMDevPrim
	instParameters:	I w ad as pd ps nrd nrs m geo rdc rsc delvto dtemp
	termOrder:	D G S B
	propMapping:	
	componentName:	mosfet
	namePrefix:	M
bcs	netlistProcedure:	HSIMCompPrim
	instParameters:	cur
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	bcs
	namePrefix:	G
bvs	netlistProcedure:	HSIMCompPrim
	instParameters:	vol
	termOrder:	PLUS MINUS
	propMapping:	
	componentName:	bvs
	namePrefix:	E
vcres	netlistProcedure:	HSIMCompPrimRef

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIM Netlist Properties

Table 120 HSIM Netlist Properties (Continued)

Primitive	Function	Default Parameter
	instParameters:	csType transfactor delta xypairs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x18 x20 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	n+ n-
	refTermOrder:	in+ in-
	propMapping:	
	componentName:	vgres
	namePrefix:	G
vccap	netlistProcedure:	HSIMCompPrimRef
	instParameters:	csType tc1 tc2 scale delta xypairs x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12 x13 x14 x15 x16 x17 x18 x18 x20 y1 y2 y3 y4 y5 y6 y7 y8 y9 y10 y11 y12 y13 y14 y15 y16 y17 y18 y18 y20
	termOrder:	n+ n-
	refTermOrder:	in+ in-
	propMapping:	
	componentName:	vccap
	namePrefix:	G

HSIMD Netlist Properties

The table below describes the netlist properties associated with HSIMD.

Table 121 HSIMD Netlist Properties

Primitive	Function	Default Parameter
res	netlistProcedure:	
	instParameters:	r l w m scale dtemp tc1 tc2
	otherParameters:	model
	componentName:	resistor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise
	namePrefix:	R
presistor	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(“PLUS\”))))”)
	netlistProcedure:	
	instParameters:	r l w m scale dtemp tc1 tc2
	otherParameters:	model
	componentName:	resistor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise
cap	namePrefix:	R
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(“PLUS\”))))”)
	netlistProcedure:	
	instParameters:	c l w m scale dtemp ic tc1 tc2

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIMD Netlist Properties

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	otherParameters:	model
	componentName:	capacitor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise
	namePrefix:	C
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(\"PLUS\"))))”)
pcapacitor	netlistProcedure:	
	instParameters:	c l w m scale dtemp ic tc1 tc2
	otherParameters:	model
	componentName:	capacitor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise
	namePrefix:	C
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(\"PLUS\"))))”)
ind	netlistProcedure:	
	instParameters:	I m dtemp ic
	otherParameters:	model
	componentName:	inductor
	termOrder	PLUS MINUS
	proMapping	nil dtemp trise

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
mind	namePrefix:	L
	termMapping	(nil PLUS \:1 MINUS “(FUNCTION minus(root(“PLUS\”))))”)
	netlistProcedure:	hsimDMindPrim
	instParameters:	coupling
	otherParameters:	ind1 ind2
	componentName:	mutual_inductor
	termOrder	
	proMapping	nil coupling k
	namePrefix:	K
	termMapping	
vdc	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type
	otherParameters:	noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	vsource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acptype srcType
	namePrefix:	V
	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(“PLUS\”))))”)
	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type
idc		

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIMD Netlist Properties

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
vpulse	otherParameters:	noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	nil dc idc mag acm phase acp type srcType
	namePrefix:	I
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(“PLUS\”))))”)
	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type va10 val1 period delay rise fall width
	otherParameters:	fundname noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	vsource
ipulse	termOrder	PLUS MINUS
	proMappin	nil dc vdc mag acm phase acp va10 v1 val1 v2 period per delay td rise tr fall tf width pw type srcType
	namePrefix:	V
	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(“PLUS\”))))”)
	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type va10 val1 period delay rise fall width
	otherParameters:	

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
vpwl	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	(nil dc vdc mag acm phase acp va10 v1 val1 v2 period per delay td rise tr fall tf width pw type srcType
	namePrefix:	I
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(“PLUS\”))))”)
	netlistProcedure:	hsimDPwlsrcPrim
	instParameters:	dc type mag phase delay offset scale stretch pwlperiod twidth
	otherParameters:	tvpairs t1 v1 t2 v2 t3 v3 t4 v4 t5 v5 t6 v6 t7 v7 t8 v8 t9 v9 t10 v10 t11 v11 t12 v12 t13 v13 t14 v14 t15 v15 t16 v16 t17 v17 t18 v18 t19 v19 t20 v20 t21 v21 t22 v22 t23 v23 t24 v24 t25 v25 t26 v26 t27 v27 t28 v28 t29 v29 t30 v30 t31 v31 t32 v32 t33 v33 t34 v34 t35 v35 t36 v36 t37 v37 t38 v38 t39 v39 t40 v40 t41 v41 t42 v42 t43 v43 t44 v44 t45 435 t46 v46 t47 v47 t48 v48 t49 v49 t50 v50 noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	vsource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acp delay td offset vo type srcType
	namePrefix:	V
	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(“PLUS\”))))”)

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIMD Netlist Properties

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
ipwl	netlistProcedure:	hsimDPwlsrcPrim
	instParameters:	dc type mag phase delay offset scale stretch pwlp period twidth
	otherParameters:	tipairs t1 i1 t2 i2 t3 i3 t4 i4 t5 i5 t6 i6 t7 i7 t8 i8 t9 i9 t10 i10 t11 i11 t12 i12 t13 i13 t14 i14 t15 i15 t16 i16 t17 i17 t18 i18 t19 i19 t20 i20 t21 i21 t22 i22 t23 i23 t24 i24 t25 i25 t26 i26 t27 i27 t28 i28 t29 i29 t30 i30 t31 i31 t32 i32 t33 i33 t34 i34 t35 i35 t36 i36 t37 i37 t38 i38 t39 i39 t40 i40 t41 i41 t42 i42 t43 i43 t44 i44 t45 435 t46 i46 t47 i47 t48 i48 t49 i49 t50 i50 noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	nil dc idc mag acm phase acp delay td offset io type srcType
	namePrefix:	V
	termMapping	(nil PLUS \:sink MINUS “FUNCTION minus(root(“PLUS\”)))”)
	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type delay sinedc ampl sinephase freq damp
	otherParameters:	
	componentName:	vsource
vsin	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acp delay td sinedc vo amp1 damp theta type srcType

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
isin	namePrefix:	V
	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(“PLUS\”))))”)
	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type delay sinedc amp1 sinephase freq damp
	otherParameters:	fundname fundname2 noisefile FNpairs F1 N1 F2 N2 F3 N3 F4 N4 F5 N5 F6 N6 F7 N7 F8 N8 F9 N9 F10 N10
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acp delay td sinedc vo ampl damp theta type srcType
	namePrefix:	I
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(“PLUS\”))))”)
vexp	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type delay val0 val1 td1 tau1 td2 tau2
	otherParameters:	
	componentName:	vsource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acp delay td val0 v1 val1 v2 type srcType
	namePrefix:	V

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIMD Netlist Properties

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
ixep	termMapping	(nil PLUS \:P MINUS “(FUNCTION minus(root(\"PLUS\"))))”)
	netlistProcedure:	hsimDSrcPrim
	instParameters:	dc mag phase type delay val0 val1 td1 tau1 td2 tau2
	otherParameters:	
	componentName:	isource
	termOrder	PLUS MINUS
	proMapping	nil dc vdc mag acm phase acp delay td val0 v1 val1 v2 type srcType
	namePrefix:	I
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\"PLUS\"))))”)
	netlistProcedure:	hsimDVCPPrim
vcvs	instParameters:	gain
	otherParameters:	
	componentName:	vcvs
	termOrder	PLUS MINUS NC\+ NC\-
	proMapping	nil gain egain
	namePrefix:	E
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(\"PLUS\")))” NC\+ “(FUNCTION zero(root(\"PLUS\")))” NC\“(FUNCTION zero(root(\"PLUS\")))”)
	netlistProcedure:	hsimDVCPPrim

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	instParameters:	gm m
	otherParameters:	
	componentName:	vccs
	termOrder	PLUS MINUS NC\+ NC\-
	proMapping	nil gm ggain
	namePrefix:	G
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(“PLUS\”)))” NC\+ “(FUNCTION zero(root(“PLUS\”)))” NC\- “(FUNCTION zero(root(“PLUS\”))))”
ccvs	netlistProcedure:	hsimDCCPrim
	instParameters:	rm
	otherParameters:	vref
	componentName:	ccvs
	termOrder	PLUS MINUS
	proMapping	nil rm hgain probe vref
	namePrefix:	H
	termMapping	(nil PLUS \:p MINUS “(FUNCTION minus(root(“PLUS\”))))”
cccs	netlistProcedure:	hsimDCCPrim
	instParameters:	gain m
	otherParameters:	vref
	componentName:	cccs

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIMD Netlist Properties

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termOrder	PLUS MINUS
	proMapping	nil gain fgain probe vref
	namePrefix:	F
	termMapping	(nil PLUS \:sink MINUS “(FUNCTION minus(root(“PLUS\”))))”)
tline	netlistProcedure:	
	instParameters:	z0 td f nl vel len m
	otherParameters:	model
	componentName:	tline
	termOrder	IN\+ IN\‐ OUT\+ OUT\‐
	proMapping	ni1 z0 zo f freq
	namePrefix:	T
	termMapping	(nil IN\+ \:t1 IN\‐ \:b1 OUT\+ \:t2 OUT\‐ \:b2)
diode	netlistProcedure:	
	instParameters:	area perim 1 w m scale trise region
	otherParameters:	model
	componentName:	
	termOrder	PLUS MINUS
	proMapping	
	namePrefix:	D
	termMapping	(nil PLUS \:a MINUS “(FUNCTION minus(root(“PLUS\”))))”)

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
pdiode	netlistProcedure: instParameters: otherParameters: componentName: termOrder proMapping namePrefix: termMapping	area perim 1 w m scale trise region model PLUS MINUS (nil PLUS \:a MINUS “(FUNCTION minus(root(“PLUS\”))))”)
zener	netlistProcedure: instParameters: otherParameters: componentName: termOrder proMapping namePrefix: termMapping	area perim 1 w m scale trise region model PLUS MINUS (nil PLUS \:a MINUS “(FUNCTION minus(root(“PLUS\”))))”)
schottky	netlistProcedure: instParameters: otherParameters: componentName:	area perim 1 w m scale trise region model

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIMD Netlist Properties

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	termOrder	PLUS MINUS
	proMapping	
	namePrefix:	D
	termMapping	(nil PLUS \:a MINUS “(FUNCTION minus(root(\"PLUS\"))))”)
npn	netlistProcedure:	
	instParameters:	area m trise region
	otherParameters:	model
	componentName:	
	termOrder	C B E S
	proMapping	
	namePrefix:	Q
	termMapping	(nil C \:c B \:b E \:e S \:s)
pnp	netlistProcedure:	
	instParameters:	area m trise region
	otherParameters:	model
	componentName:	
	termOrder	C B E S
	proMapping	
	namePrefix:	Q
	termMapping	(nil C \:c B \:b E \:e S \:s)
njfet	netlistProcedure:	

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
pjfet	instParameters:	area m region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	J
	termMapping	(nil D \:d G \:g S \:s B \:b)
	netlistProcedure:	
	instParameters:	area m region
	otherParameters:	model
nmos	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	J
	termMapping	(nil D \:d G \:g S \:s B \:b)
	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs Id ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIMD Netlist Properties

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
pmos	namePrefix:	M
	termMapping	(nil D \:d G \:g S \:s B \:b)
	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs Id ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	M
	termMapping	(nil D \:d G \:g S \:s B \:b)
nmos4	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs Id ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	M
	termMapping	(nil D \:d G \:g S \:s B \:b)
	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs Id ls m trise region
pmos4	otherParameters:	model

Table 121 HSIMD Netlist Properties (Continued)

Primitive	Function	Default Parameter
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	M
	termMapping	(nil D \:d G \:g S \:s B \:b)
nbsim	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	S
	termMapping	(nil D \:d G \:g S \:s B \:b)
pbsim	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	S
	termMapping	(nil D \:d G \:g S \:s B \:b)

Appendix H: HSIM-Virtuoso Interface Netlist Properties
HSIMD Netlist Properties*Table 121 HSIMD Netlist Properties (Continued)*

Primitive	Function	Default Parameter
nbsim4	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	S
	termMapping	(nil D \:d G \:g S \:s B \:b)
	netlistProcedure:	
	instParameters:	w l as ad ps pd nrd nrs ld ls m trise region
pbsim4	otherParameters:	model
	componentName:	
	termOrder	D G S B
	proMapping	
	namePrefix:	S
	termMapping	(nil D \:d G \:g S \:s B \:b)

HSIM CCK

HSIM CCK capabilities include: reporting potential circuit problems during the early stage of simulation, detecting incorrect input data or tool usage, analyzing latch conditions before simulation starts, monitoring node voltages, generating reports, comparing waveforms to determine when and why a circuit node is triggered to change its value, computing static rise and fall delays to transistor gates, and analyzing crosstalk noise voltage caused by coupling capacitors.

- Overview of CCK Option
- Using CCK Commands
- Running Circuit Check Operations
- Passing Parameters Into CCK Commands
- Running Parametric Checks
- Running Static Analysis for Devices
- Running Post-layout RC Checking
- Checking Design and Electrical Rules
- Running Device Voltage Analysis for Transistors
- Running Device Voltage Analysis for BJT Devices
- Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices
- Running Subcircuit-Based Voltage Analysis Using the Static Approach
- Running Diode Forward Bias Analysis
- Running Element Current Analysis
- Running a Reference Check for Instances and Subcircuits
- Detecting Excessive Current Path
- Checking for Floating Gates and Analyzing Current Sources

Appendix I: HSIM CCK

- Checking NMOS Bulk Connections
- Finding Potentially Conducting NMOS Devices
- Checking NMOS Node to VDD Connection
- Checking Node Voltages
- Checking Paths to Voltage Sources
- Checking PMOS Bulk Connections
- Finding Potentially Conducting PMOS Devices
- Checking PMOS Node To GND Connection
- Running a Safe Operating Area Check
- Evaluating Signal Degradation or Correct Biasing
- Checking Substrate Forward Bias
- Checking for Unprotected Antenna Nodes
- Checking Static Voltage Propagation Sharing
- Running Digital Logic and Memory Diagnostics
- Checking Stack-up Transistors
- Checking and Classifying the Stuck Nodes
- Running Interactive Circuit Debugging Command for a Tracking Circuit
- Finding a Node First State Change After a Specified Time
- Running Timing Checks
- Running Static RC Delay Analysis – Estimate Slew Rate
- Running a Dynamic Device Voltage Check
- Running a Post-process Device Voltage Check
- Running Signal Integrity Checks
- Signal Edge Characteristics
- Crosstalk Checking Methods
- Static Crosstalk Noise Analysis: Estimating Noise Glitches
- Detecting Leakage Current
- Detecting Power-Down Floating Gates
- Running Static Analysis

- [Checking for a Static DC Path](#)
 - [CCK Utilities](#)
 - [Running the CCK Tutorial](#)
-

Overview of CCK Option

The CCK option detects design problems that may lead to lengthy simulation times and automates circuit debugging and diagnostics checks. Its capabilities include:

- Reporting potential problems in the circuit before running a simulation.
- Scanning the netlist for geometric and electrical parameter errors.
- Identifying uninitialized latches before simulation starts.
- Monitoring node voltages during a simulation and generating a report when the constraints set by the user are detected.
- Tracing circuit paths to find the source of state changes.
- Computing the static rise and fall delays to transistor gates.
- Analyzing the crosstalk noise voltage caused by coupling capacitors.
- Locating leakage paths and monitoring standby conditions.

CCK commands are executed in different stages when running HSIM and provide various reports, warning, or error messages. Unless otherwise specified, the CCK commands are invoked by adding a list of CCK commands in a single file. The CCK command file must be specified in the HSIM input file using the HsimCktCheck parameter as illustrated below:

```
.param hsimCktCheck=cck_cmd_file
```

In this example, `cck_cmd_file` is the name of the CCK command file.

CCK Tutorial

For a comprehensive CCK Tutorial, see the [Running the CCK Tutorial on page 1091](#).

Conventions

The following conventions are used for CCK commands:

- The terms VDD (voltage source) and GND (ground) are used as generic terms. CCK does not look for VDD and GND, but rather the values they represent.
- In CCK only, a back slash character (\) indicates a continuation of the code onto the next line. When entering this code string, the backslash is omitted.
- The /* and */ comment delimiters are used in the CCK command language. The beginning is indicated by a /* and the end of a comment is indicated by a */. CCK treats everything between the /* and */ delimiters as part of the comment.

Using CCK Commands

CCK commands provide a wide array of functions designed to support various simulation, processing, and testing options. The CCK commands have been developed in functional groups to provide various types of testing. Some of the commands are expected to be used in a group, and some provide standalone functions. The commands are presented in the following categories and in the following sections.

- [Running Parametric Checks on page 904](#)
- [Checking Design and Electrical Rules on page 917](#)
- [Running Digital Logic and Memory Diagnostics on page 1003](#)
- [Running Timing Checks on page 1023](#)
- [Running a Dynamic Device Voltage Check on page 1041](#)
- [Running Signal Integrity Checks on page 1051](#)
- [Detecting Leakage Current on page 1066](#)
- [CCK Utilities on page 1082](#)

The following tables list the commands most likely to be used for the listed applications. The commands listed in the tables are not always exclusive to that application and may be used for a variety of applications.

Design and Electrical Rules Check

cckAntGate on page 999	cckCapV on page 933
cckDiode on page 946	cckDioV on page 936
cckElemI on page 948	cckMatchSub on page 950
cckExiPath on page 951	cckFloatGateIsrc on page 953
cckMosV on page 918	cckNmosG_gt_DS on page 960
cckNmosB_gt_DS on page 956	cckNmosNodeToVDD on page 965
Checking Node Voltages on page 967	cckPathToVsrc on page 969
cckPmosB_It_DS on page 973	cckPmosG_It_DS on page 977
cckPmosNodeToGnd on page 984	cckResV on page 939
cckSOA on page 985	cckSubstrate on page 996
tcheck mosv on page 1041 ¹	

1. This command and its parameters are used for Dynamic Device Voltage Check during simulation.

Note: For more information on design and electrical rules checks, see [Checking Design and Electrical Rules on page 917](#).

Digital Logic and Memory Diagnostics

cckFlashcore on page 1003	cckLatchUnInit on page 1004
cckLatchInElem on page 1005	cckMaxStackUpNmos on page 1007
cckLatchSkipElem on page 1006 ¹	
cckMaxStackUpPmos on page 1008	cckMaxStuckAt on page 1009

Appendix I: HSIM CCK

Using CCK Commands

Digital Logic and Memory Diagnostics

[cckToggleCount on page 1010](#) [ntrig on page 1018](#)

-
1. *These are sub-commands for cckLatchUnInit*

Note: For more information on digital logic and memory diagnostics, see [Running Digital Logic and Memory Diagnostics on page 1003](#).

Timing Checks

[cckMaxNmosToVdd on page 1023](#) [cckMaxPmosToGnd on page 1024](#)[cckMaxStackUpNmos on page 1007](#) [cckMaxStackUpPmos on page 1008](#)[cckMeasPathDelay on page 1025](#) [cckNodeMaxRF on page 1027](#)[cckParasiticRC on page 916](#) [cckRCDlyPath on page 1029¹](#)

-
1. *See cckRCDlyPath for a description of the group of commands needed to support this function.*

Note: For more information on timing checks, see [Running Timing Checks on page 1023](#).

Parametric Checks

[cckParam on page 904](#) [cckParasiticRC on page 916](#)

Note: For more information on parametric checks, see [Running Parametric Checks on page 904](#).

Signal Integrity Checks

[cckDXtalk on page 1052¹](#) [cckParasiticRC on page 916](#)

-
1. *This is the Dynamic Crosstalk command.*

Note: For more information on signal integrity checks, see [Running Signal Integrity Checks on page 1051](#).

Leakage Current Detection

cckAnalogPDown on page 1073	cckAnalogPDownlth on page 1075
cckEleml on page 948	cckEleml on page 948
cckMaxStaticLeak on page 1066	cckOffLeakl on page 1067
cckStaticHzNode on page 1078	cckStaticDCPath on page 1080

Note: For more information on leakage current detection, see [Detecting Leakage Current on page 1066](#).

CCK Utilities

cckBasic on page 1082	cckCompareOp on page 1083
cckMatchSub on page 1085	cckPatternMatch on page 1086
cckPatternConstraint on page 1087	cckSetMosDir on page 1035
cckTgPair on page 1089	

Note: For more information on CCK utilities, see [CCK Utilities on page 1082](#).

Table 122 lists commonly used commands for basic design, electrical parameters, and logic check as sorted by analysis type.

Table 122 Basic Design, Electrical Parameters, and Logic Check

Static Analysis	Dynamic Analysis	Multiple Mode
cckAntGate	cckEleml	cckDiode
cckCapV	cckExiPath	cckSubstrate
cckDioV	cckFlashCore	
cckFloatGatelsrc	cckLatchInElem	

Appendix I: HSIM CCK

Using CCK Commands

Table 122 Basic Design, Electrical Parameters, and Logic Check

Static Analysis	Dynamic Analysis	Multiple Mode
cckMatchSub	cckLatchSkipElem	
cckMaxPmosToGnd	cckLatchUninit	
cckMaxStackUpNmos	cckMeasPathDelay	
cckMaxStackUpPmos	cckNodeVoltage	
cckMaxStuckAt	cckSOA	
cckMosV	cckToggleCount	
cckNmosB_gt_DS	Intrig	
cckNmosG_gt_DS	Ntrig	
cckNmosNodeToVdd	Tcheck mosv	
cckParasiticRC		
cckPathToVsrc		
cckPmosB_lt_DS		
CckPmosG_lt_DS		
cckPmosNodeToGnd		
cckRCDlypath		
cckRCDlypath		
cckMaxNmosToVdd		

Table 123 lists commonly used commands for device dependent characteristics, as sorted by analysis type.

Table 123 Device Dependent Characteristics

Static	Dynamic
cckBasic	cckAnalogPdown
cckMaxStaticLeak	cckAnalogPDownlth
cckParam	cckCompareOp
cckparasiticRC	cckDXtalk
cckPatternMatch	cckExiPath
cckSetMosDir	cckOffLeakI
cckStaticXtalk_GroupC	
cckTgPair	

Specifying Circuit Checks in Command Files

CCK allows you to specify static or dynamic check commands to conduct specific analysis throughout the simulation. To use this capability, you need to group all the commands in a file and add the following command in the netlist:

```
.param hsimCktCheck=<cck_cmd_filename>
```

For example, suppose you want to conduct a CCK static floating gate check. Create a `cck.cmd` CCK command file and add the following content to the file:

```
cckFloatGateIsrc 1
```

Then in the netlist, add the following statement for the CCK command file specification:

```
.param hsimCktCheck=cck.cmd
```

In addition to the static and dynamic CCK commands, there is a special group of commands that conduct dynamic device voltage checks (see the [Running a Dynamic Device Voltage Check on page 1041](#) for details). You must group all the related dynamic device voltage check commands in a command file and specify it in the netlist with the following statement:

Appendix I: HSIM CCK

Running Circuit Check Operations

```
.param hsimDeviceV=<deviceV_cmd_filename>
```

For example, suppose you want to conduct a CCK dynamic `mosv` check. Create a `deviceV.cmd` CCK command file with the following content in the file:

```
.tcheck test1 mosv model=* lvgs=-0.6
```

Then in the netlist, add the following statement for the CCK command file specification:

```
.param hsimDeviceV=deviceV.cmd
```

Running Circuit Check Operations

Most of the static CCK commands that conduct analyses are based on netlist connectivity data and do not require DC initialization or transient simulation results. By default, HSIM always conducts full DC initialization and transient simulation whether CCK commands are used or not. However, sometimes you only want to focus on static CCK checks, so a `cckNoSimu` control parameter (described in [cckNoSimu](#)) is available to allow HSIM to skip the DC initialization and transient steps.

cckNoSimu

Accelerates the static CCK commands by skipping DC initialization and transient simulation.

Syntax

```
cckNoSimu [0|1] level=[0|1|2]
```

Arguments

Argument	Description
level=0	Runs both DC initialization and transient simulation (same as cckNoSimu 0).
level=1	Disables transient simulation (same as cckNoSimu 1).
level=2	Disables DC initialization, transient simulation, and partitioning (building the circuit database).

Examples

The following CCK command file conducts static floating gate analysis based on the netlist connectivity database and skips transient simulation:

```
cckNoSimu level=1
cckFloatGateIsrc 1
or
cckNoSimu level=1
cckFloatGateIsrc 1
```

The following CCK command file conducts static floating gate analysis based on the netlist connectivity database and goes directly to the end of the simulation process by skipping DC initialization, transient simulation, and partitioning (building the circuit database).

```
cckNoSimu level=2
cckFloatGateIsrc 1
```

Note: The `cckNoSimu` command has no effect with the `cckStaticXTalk` and `cckRCDlyPath` static CCK commands, which conduct static crosstalk glitch analysis and static path delay analysis, respectively. If you use one or both commands in the CCK command file, HSIM automatically disables transient simulation even without the `cckNosimu` command. Do not use the `cckStaticXTalk` and `cckRCDlyPath` commands with other static CCK commands to try to save memory and runtime. For example:

```
cckNoSimu level=2
cckFloatGateIsrc 1
cckRCDlyPath 1
```

Appendix I: HSIM CCK

Passing Parameters Into CCK Commands

Disables transient simulation only because cckRCDlyPath is specified, but still runs DC initialization and builds the circuit database. So these steps do not achieve the original goal to save memory and runtime.

You need to properly specify cckNoSimu to avoid design flow problems. For example, if you conduct a static floating gate check and dynamic device voltage check at the same and include cckNoSimu, HSIM runs but does not perform a dynamic check because DC initialization and transient simulation are disabled.

Passing Parameters Into CCK Commands

To provide the freedom to organize HSIM inputs and CCK command files, the parameter-passing capabilities in both HSIM and CCK are provided in the CCK Command groups.

Parameter values are passed into CCK command files by specifying them in either the input netlist file or CCK command files.

Parameters specified in the netlist or CCK command file are subject to the following constraints:

1. The `.param` statement in the HSIM input netlist or the CCK Command file must follow SPICE format, so a `.param` is only assigned a value or an expression. However, a string is not allowed in a parameter specification.
2. If the same parameter is specified in the netlist file, via `.param HSIMCKTCHECK=cck_cmd_File`, or via `.param HSIMDEVICEV=dev_v_file`, there are priority rules for CCK to pick the values:
 - In non-post-process cases:
`.param in HSIM netlist > .param HSIMCKTCHECK=cck_cmd_File >`
`.param HSIMDEVICEV=dev_v_file`
 - In post-process cases:
`.param in HSIM netlist > .param HSIMDEVICEV=dev_v_file >`
`.param HSIMCKTCHECK=cck_cmd_File`

When the following parameter definitions are added into the HSIM input file, the CCK command files use these values and not the CCK command values.

Example 83 Parameter Definitions

```
.param min=1.0
.param base='min + 0.5'
.param max='2.0*min + base'
.param duration='3*(-min+max)'
.param HsimCktCheck=CircuitCheck.cck
```

Where `min` is an assigned value, and `base`, `max`, and `duration` are all given with an expression. These expressions are converted to values by the HSIM parser.

CircuitCheck.cck Command File

In this CCK command file example, the parameters `vg` and `vs` are defined using the parameter values passed from the netlist file. For example:

Example 84 CCK Command File

```
.param vg=base+min
.param vs=-min+2*max
cckMosv tag model=p vhth=min lvd=base uvd=max
cond= '(uvg<vg || lvd>=vs)'
```

If the same parameter is specified in multiple places, priority rules of parameter-passing apply, and a warning message is issued in the log file, such as:

```
WARNING '.param vdd1' has already been defined, ignore!!
```

Note: In the `cond` expression, combinations of parameters such as `base+min`, `-min+2*max` are also supported.

In the above example, the parameter definitions are specified in either netlist file or CCK command file.

Include Statements

The include statement can be used to include files into CCK command files. Multiple include statements are allowed in a CCK command file. For example:

Appendix I: HSIM CCK

Running Parametric Checks

```
/* file: cck_cmd */
include cck_cmd1
cckPmosG_lt_DS

/* file: cck_cmd1 */
cckNmosG_gt_DS
include cck_cmd2

/* file: cck_cmd2 */
cckMosv plv_vgs1 model=plv cond='lvg-uv< -1.7' rpttrace=1
```

Running Parametric Checks

These CCK commands are designed to help check min/max device dimensions, temperature, post-layout parasitic effects, and so on.

Checking Electrical Parameters

The `cckParam` command checks for:

cckParam

The `cckParam` command checks for:

- Normal element size.
- Correct model usage.
- Reasonable temperature usage.

CCK commands are executed in different stages when running HSIM.

[Figure 102 on page 905](#) graphically illustrates the high and low limits where warning or error messages are generated and when required, CCK stops. When data being checked exceeds the CCK soft-upper bound, a warning message is issued. As the data abnormality exceeds the CCK absolute upper bound, an error message is generated and CCK stops.



Figure 102 CCK Checking Methodology

When a very large MOSFET width is detected, such as 1 meter which is above the absolute upper bound, it is considered an error and simulation stops. If the width appears to be larger than ordinary, but has not reached the absolute upper bound, a warning message is generated and checking continues.

CCK checks the following device parameters:

<erMaxCap=v>	<waMaxCap=v>
<erMaxMosW=v>	<waMaxMosW=v>
<erMinMosW=v>	<waMinMosW=v>
<erMaxMosL=v>	<waMaxMosL=v>
<erMinMosL=v>	<waMinMosL=v>
<erMaxMosAD=v>	<waMaxMosAD=v>
<erMaxMosAS=v>	<waMaxMosAS=v>
<erMaxMosPD=v>	<waMaxMosPD=v>
<erMaxMosPS=v>	<waMaxMosPS=v>
<erMaxMosTox=v>	<waMaxMosTox=v>
<erMinMosTox=v>	<waMinMosTox=v>
<erMaxTemp=v>	<waMaxTemp=v>
<erMinTemp=v>	<waMinTemp=v>

Appendix I: HSIM CCK

Running Parametric Checks

```
<erMaxDiodeW=v>      <waMaxDiodeW=v>  
<erMaxDiodeL=v>      <waMaxDiodeL=v>  
<erMaxDiodeA=v>      <waMaxDiodeA=v>  
<erMinDiodeW=v>      <waMinDiodeW=v>  
<erMinDiodeL=v>      <waMinDiodeL=v>  
<erMinDiodeA=v>      <waMinDiodeA=v>  
<erMinMfactor=v>      <waMinMfactor=v>  
  
<model=m>
```

Note: Resistor size is checked by using the HSIMRMIN and HSIMRMAX commands.

CCK parameters are described in the following sections.

Capacitor Values

Cap (Capacitor) has two bounds:

- erMaxCap: Absolute upper bound.
- waMaxCap: Soft-upper bound.

The results of a capacitor value causes one of the following to occur:

- HSIM stops: If a capacitor exceeds the absolute upper bound.
- Prints a warning message: If the capacitor is larger than the soft-upper bound, but not the absolute upper bound.

The error message and warning message are reported in hsim.cck.

MOSFET Width

There are four bounds associated with MOSFET width:

- erMaxMosW: Absolute upper bound.
- waMaxMosW: Soft-upper bound.

- erMinMosW: Absolute lower bound.
- waMinMosW: Soft-lower bound.

MOSFET Width: Upper Bounds

If the width of a MOSFET is larger than erMaxMosW, an error message is reported and HSIM is stopped. If the width is between waMaxMosW and erMaxMosW, a warning message is printed.

MOSFET Width: Lower Bounds

If the width of a MOSFET is smaller than erMinMosW, an error message is reported and HSIM is stopped. If the width of a MOSFET is between waMinMosW and erMinMosW, a warning message is printed.

MOSFET Length

MOSFET length has the following bounds:

- erMaxMosL: Absolute upper bound.
- waMaxMosL: Soft-upper bound.
- erMinMosL: Absolute lower bound.
- waMinMosL: Soft-lower bound.

MOSFET Length: Upper Bounds

If MOSFET length exceeds erMaxMosL, HSIM is stopped. If the length is between waMaxMosL and erMaxMosL, a warning message is printed.

MOSFET Length: Lower Bounds

If the length of a MOSFET is smaller than erMinMosL, HSIM is stopped. If the width of a MOSFET is between waMinMosL and erMinMosL, a warning message is printed.

MOSFET Drain/Source Area and Drain/Source Perimeter

MOSFET Drain Area has the following bounds:

- erMaxMosAD: Absolute upper bound.
- waMaxMosAD: Soft-upper bound.

MOSFET Source Area has the following bounds:

Appendix I: HSIM CCK

Running Parametric Checks

- erMaxMosAS: Absolute upper bound.
- waMaxMosAS: Soft-upper bound.

MOSFET Drain Perimeter has the following bounds:

- erMaxMosPD: Absolute upper bound.
- waMaxMosPD: Soft-upper bound.

MOSFET Source Perimeter has the following bounds:

- erMaxMosPS: Absolute upper bound.
- waMaxMosPS: Soft-upper bound.

MOSFET Gate Oxide Thickness

The MOSFET Tox is checked. The upper and lower bounds are set to detect design problems as follows:

- erMaxMosTox: Absolute upper bound.
- waMaxMosTox: Soft-upper bound.
- erMinMosTox: Absolute lower bound.
- waMinMosTox: Soft-lower bound.

Diode Width, Length, and Area

The twelve values associated with diode are:

- erMaxDiodeW: Absolute upper bound for diode width.
- waMaxDiodeW: Soft-upper bound for diode width.
- erMaxDiodeL: Absolute upper bound for diode length.
- waMaxDiodeL: Soft-upper bound for diode length.
- erMaxDiodeA: Absolute upper bound for diode area.
- waMaxDiodeA: Soft-upper bound for diode area.
- erMinDiodeW: Absolute lower bound for diode width.
- waMinDiodeW: Soft-lower bound for diode width.
- erMinDiodeL: Absolute lower bound for diode length.
- waMinDiodeL: Soft-lower bound for diode length.

- erMinDiodeA: Absolute lower bound for diode area.
- waMinDiodeA: Soft-lower bound for diode area.

Simulation Run Temperature

Simulation run temperature is checked. The upper and lower bounds are set to detect simulation problems as follows:

- erMaxTemp: Absolute upper bound.
- waMaxTemp: Soft-upper bound.
- erMinTemp: Absolute lower bound.
- waMinTemp: Soft-lower bound.

If the temperature exceeds the absolute upper bound defined as erMaxTemp, HSIM is stopped.

Model

Allow different parameter values for different models. Each line contains all the specific values for each individual model.

The default values are defined in [Table 124](#):

Table 124 Model Default Parameter Values

Default Value	Measurement Unit
erMaxCap=0.001	F
waMaxCap=1.e-8	F. This value is 10000 pF.
erMaxMosW=0.01	Meter. This value is 10000 um.
waMaxMosW=0.001	Meter. This amounts to 1000 um.
erMinMosW=1.e-8	Meter. This amounts to 0.01 um.
waMinMosW=1.e-7	Meter. This amounts to 0.1um.
erMaxMosL=0.01	Meter
waMaxMosL=0.001	Meter. This amounts to 1000 um.
erMinMosL=1.e-8	Meter. This value is 0.01 um.

Appendix I: HSIM CCK
Running Parametric Checks*Table 124 Model Default Parameter Values*

Default Value	Measurement Unit
waMinMosL=1.e-7	Meter. This amounts to 0.1 um.
erMaxMosAD=0.0001	Meter square
waMaxMosAD=1.e-6	Meter square
erMaxMosAS=0.0001	Meter square
waMaxMosAS=1.e-6	Meter square
erMaxMosPD=0.01	Meter. This value is 10000 um.
waMaxMosPD=0.001	Meter
erMaxMosPS=0.01	Meter
waMaxMosPS=0.001	Meter
erMaxMosTox=5.e-8	Meter
waMaxMosTox=3e-8	Meter
erMinMosTox=5.e-10	Meter
waMinMosTox=5.e-9	Meter
erMaxDiodeW=1.e-2	Meter
waMaxDiodeW=1.e-3	Meter
erMaxDiodeL=1.e-2	Meter
waMaxDiodeL=1.e-3	Meter
erMaxDiodeA=1.e-4	Meter square
waMaxDiodeA=1.e-6	Meter square
erMinDiodeW=1.e-8	Meter
waMinDiodeW=1.e-7	Meter

Table 124 Model Default Parameter Values

Default Value	Measurement Unit
erMinDiodeL=1.e-8	Meter
waMinDiodeL=1.e-7	Meter
erMinDiodeA=1.e-16	Meter square
waMinDiodeA=1.e-14	Meter square
erMaxTemp=500	Celsius
waMaxTemp=100	Celsius
erMinTemp=-200	Celsius
waMinTemp=-100	Celsius
erMinMfactor=-0.001	
waMinMfactor=0	

The mode parameter for cckParam command is:

mode=0 | 1

if mode is specified as 1, the check will report geometry out-of-bound devices, and the min/max values within the associated device group.

For example: `cckparam wamaxmostox=10.1 mode=1` will flag as warning the MOSFET devices with tox value greater than 10.1, and will also report the min/max tox value among all the MOSFET devices.

Default value of mode is 0.

These defaults may be overridden by using the commands listed above.

The following is an example of the cckParam command.

Example 85 cckParam Command

```
cckParam erMaxCap=2000p waMaxCap=1.e-9
cckParam erMaxMosW=0.1 waMaxMosW=0.03 erMinMosW=0.01u waMinMosW=0.1u
```

A capacitor with these characteristics is reported as follows:

Appendix I: HSIM CCK

Running Parametric Checks

- An error message is reported for a value > 2000 pF (picofarad).
- A warning message is printed for capacitor values between 1000 pF and 2000 pF.

A MOSFET with these widths is reported as follows:

- HSIM stops and an error message is reported if its width is one of the following:
 - $> 0.1 \text{ m}$
 - $< 0.01 \text{ um}$.
- A warning message is printed if its width is between either of the following:
 - 0.03 meter and 0.1 meter.
 - 0.01 micron and 0.1 micron.

The following is a sample output (hsim.cck or out_file.cck) resulting from the command shown in the previous example.

Example 86 Output File

```
*****
* Check Elem Size, Model, Temperature
*****
Warning: find a mos (mi8x) with unusual width (0.04 M) in subckt (fuse)
Error: can't allow mos (mc2, width 0.155 M) in subckt (fuse10)
Warning: find a mos (mi8) with unusual width (0.05 M) in subckt (cgdv)
Error: can't allow too large a capacitor (cext, 3e-6 F) in subckt (TLC)
```

In the sample file shown above, the following definitions apply:

- mi8x: Element name.
- fuse: Subcircuit name.
- cext: Capacitor name.
- TLC: Top-Level Circuit.

Limiting the Number of Violations Reported

The "num" parameter limits the number of violations reported, to avoid generating lengthy output files. For example, `cckParam num=10` allows up to 10 warnings or errors in the violation report. The default number is 300.

M-factor

If there are elements with a zero or negative m-factor, warnings are reported. The following is a sample .cck output:

```
Warning: find a mos (xt.xcore.x256m4) with unusual m factor (0) in subckt (epju)
Warning: find a mos (xt.xbist.xm4) with unusual m factor (0) in subckt (enll)
```

When HSIM checks parameters, it creates a default template containing all the default values, as indicated in [Table 124 on page 909](#). This template is used to examine all the transistors. If any default needs to be changed, simply specify a cckParam statement without any model keyword, as in the following example:

```
cckParam erMaxMosW=0.05
cckParam model=nch1 erMaxMosW=0.06
```

For example, the first statement of cckParam overwrites the value of erMaxMosW to 0.05 meter. The second statement has a keyword model=nch1, which creates a new template consisting of the default values, except that erMaxMosW is changed to 0.06 meter for model nch1.

It is suggested to group the parameters for general-purpose checking first. Those values overwrite the default settings. Then, create a new statement with model and the associated values. If the line is too long, a backslash (\) is used to indicate the continuation of the line. Parameters for each model must be on a separate line.

Note: cckParam also takes effect from the .options scale in the netlist.
For example, in an input netlist such as:

```
.options
+ scale = 0.08
+ ...
```

With cckParam syntax such as:

```
cckParam model=hpmos erMinMosW=0.65u waMinMosW=0.66u
cckParam model=hmmos erMinMosW=0.65u waMinMosW=0.66u
```

The actual values of erMinMosW and waMinMosW used in the cckParam checks are the specified values scaled by a factor specified by the scale option. In the examples above, they are 0.65u*0.08 and 0.66u*0.08, respectively.

In addition, the following options are scaled by scale^2 (squared) because they are area-related: erMaxMosAD, waMaxMosAD, erMaxMosAS, waMaxMosAS.

Specifying Resistor Boundaries

There are two commands to specify resistor boundaries:

Appendix I: HSIM CCK

Running Static Analysis for Devices

- erMaxRes=v
Specifies the absolute upper boundary.
 - waMaxRes=v
Specifies the soft-upper boundary.
- The resistor value results cause one of the following conditions to occur:
- If a resistor exceeds the absolute upper boundary, HSIM stops.
 - If a resistor exceeds the soft-upper boundary, but not the absolute upper boundary, an error and warning message are printed in the `hsim.cck` file.

Running Static Analysis for Devices

An enhancement of the `cckParam` capabilities, the `cckDevParam` command conducts static analysis to scan qualified devices and reports violations when the associated geometric information satisfies the user-defined constraint expression. This expression provides a flexible and intuitive way to specify the constraints information, plus width/length detection for resistors and capacitors.

cckDevParam

Runs static analysis for qualified devices

Syntax

```
cckDevParam tag constraint='(<expression>,
    <lower_bound_value>, <upper_bound_value>, severity)' |
    '<logical_expression>'
    scope='(<subckt>, <instance>, <model>)'
    rmscope='(<subckt>, <instance>, <model>)'
```

Argument	Description
tag	Specifies a label in the report file to make searching multiple <code>cckDevParam</code> statements easier.

Argument	Description
constraint	<p>The constraint expression can be $[+ - * / \&& > < =]$, or a combination of reserved keywords for device geometry check. <code>cckDevParam</code> evaluates the expression and returns a plain value or logic value (true: 1, false: 0). Once the returned value is either lower than the <code><lower_bound_value></code> or greater than the <code><upper_bound_value></code>, <code>cckDevParam</code> considers the violation <code><severity></code>. If set to 0 (default), <code>cckDevParam</code> issues a warning regarding the devices with undesired geometric information. Once set it to 1, <code>cckDevParam</code> errors out if a violation occurs.</p> <p>The reserved keywords are:</p> <ul style="list-style-type: none"> ■ w for Mos/Res/Cap device width ■ l for Mos/Res/Cap device length ■ m for M factor ■ tox for MOSFET Gate Oxide Thickness ■ ad/as for MOSFET Drain/Source Area ■ pd/ps for MOSFET Drain/Source Perimeter ■ c/r for Capacitor/Register Value ■ a for Diode Area
scope	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
rmscope	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Examples

```
cckDevParam test1 constraint ='(w, 1e-6, 1e-5)' scope='(,,rpp1)'
```

Performs a static device geometry check on resistor devices applying the rpp1 model. A warning is issued as long as the qualified devices have widths less than 1 micron or greater than 10 microns.

```
cckDevParam test2 constraint='(w>1e-6 && l>1e-5, 0, 0.5, 1)' scope='(adder)'
```

Performs a static device geometry check for devices in the adder subcircuit. `cckDevParam` errors out once there is a qualified device with its width greater than 1 micron and its length greater than 10 micron.

Running Post-layout RC Checking

You use the `cckParasiticRC` command to run post-layout RC checking.

cckParasiticRC

Reports data about the parasitic RC during netlist loading to aid in post-layout circuit debugging.

Syntax

```
cckParasiticRC <rmin=rmin_value> <cmin=cmin_value>  
    <ccmin=ccmin_value> <numr=num_of_r> <numc=num_of_c>  
    <numcc=num_of_cc> <outFile=output_file_name>
```

Argument	Description
rmin=rmin_value	Minimum value of parasitic resistors to report. The default is rmin=0.
cmin=cmin_value	Minimum value of parasitic of grounded capacitors to report. The default is cmin=0.
ccmin=ccmin_value	Minimum value of parasitic coupling capacitors to report. The default is ccmin=0.
numr=num_of_r	Maximum number of resistors to report. The default is numr=100.
numc=num_of_c	Maximum number of grounded capacitors to report. The default is numc=100.
numcc=num_of_cc	Maximum number of coupling capacitors to report. The default is numcc=100.
outFile=output_file_name	Output file name for saving warning messages. The default is parasiticRC.cck.

Description

cckParasiticRC reports data about the parasitic RC during netlist loading to aid in post-layout circuit debugging. If the resistance, coupling capacitance, and node capacitance are greater than the predetermined values specified in the rmin, cmin, and ccmin, respectively, cckParasiticRC reports the nodes along with the corresponding resistor or capacitor values. This report is stored in the file, *output file name*, which is specified in the outFile. The default output file name is parasiticRC.cck.

Examples

The following is an example of adding cckParasiticRC to the CCK command file:

```
cckParasiticRC rmin=1 cmin=0.1p ccmin=0.1p numr=100 numc=200 numcc=200
outFile="tt.out"
```

After simulation is completed, HSIM reports the resistors, capacitors, and coupling capacitors with values greater than 1 Ohm, 0.1pF, and 0.1pF, respectively, as shown in the following report. The number of resistor nodes and capacitor nodes are limited to 100 and 200, respectively, due to the `numr`, `numc`, and `numcc` setup. The statistical results are stored in the file, `tt.out`. A portion of the report from the above example is as follows.

```
* Check parasitic R
r3215 n_14:5944 n_14:f258 r=143.889
r1197 n_10:2332 n_10:f524 r=143.889
* Check parasitic C
c44127 vdd:4600 vdd:4522 c=3.49101e-017
c40397 vdd:8035 vdd:7957 c=3.49101e-017
* Check parasitic CC
cg8188 n_9:1473 0 cc=3.64899e-017
cg32864 n_15:7189 0 cc=3.70265e-017
cckParasiticRC rmin=1 cmin=0.1p ccmin=0.1p numr=100 numc=200 numcc=200
outFile="tt.out"
```

Checking Design and Electrical Rules

These CCK commands are designed to help check over-voltage conditions, excessive device currents, forward-biased diodes, shorts to VDD or GND, missing VDD and GND connections, floating gates, and voltage-domain interface errors.

Static Device Voltage Analysis

The concept of static analysis is the ability to propagate characteristics of the design, in this case voltages, throughout the whole design without the need of vectors or transient simulation. To propagate voltages through the design it is necessary to define rules associated with the propagation. These propagation rules are used to determine if a voltage (or voltage range) can be seen by a node through a network of design elements like MOSFETs, resistors, capacitors, and diodes. Using these rules HSIM, along with the CCK Option, can analyze voltage biasing conditions and report user-defined violations, with a minimum of a design netlist and supply voltages defined.

By performing static voltage propagation, you can apply electrical rule checks to determine if damaging voltage biases exist. Because the voltage propagation is performed statically, the coverage is greater than what can be

Appendix I: HSIM CCK

Running Device Voltage Analysis for Transistors

achieved via dynamic simulation, ensuring more violations are reported. Ultimately more corrections can be implemented to ensure design correctness.

Running Device Voltage Analysis for Transistors

You use the `cckMosV` command to perform static device checking on user-specified transistors.

cckMosV

`cckMosV` performs a static check on user-specified transistors. If a violation occurs, a warning is printed to the `hsim.cckmosv` or `output.cckmosv` report file. Also, refer to [Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices on page 932](#).

Syntax

```
cckMosV tag <model=model_name> <subckt=subckt_name>
    <inst=inst_name> <rmSub=subckt_name>
    <rmInst=inst_name> <skipSub=suckt_name>
    <skipInst=inst_name> <limitMos=num> <vlth=v1> <vhth=vh>
    <vnth=v1> <vpth=v2> <lvd=v7> <uvd=v8> <lvg=v9> <uvg=v10>
    <lvs=v11> <uvs=v12> <lvb=v13> <uvb=v14>
    <cond='expression'...> <num=n> <separate_file=[0|1]>
    <rptTrace=[0|1|2]> <risePmosFallNmos=[0|1]>
    <subinfo=[0|1]> <filterAlert=[0|1]> <pwl_time=time>
    <fvsrc='(e_name,vmin,vmax)'>
    <fvsrcnd='(vsr_node_name,vmin,vmax)'>
    <connSub=subckt_name> <connInst=inst_name> <rmax4init>
    <connNode=node_name> <autoFvsrcnd=[0|1]> <vio_only =
    [0|1|2]>
```

Argument	Description
<code>tag</code>	Specifies a label in the report file to make searching easier.
<code>model=model_name</code>	Causes CCK to go through all instances of <code>model_name</code> during path tracing and does not consider other model devices. The model option only applies to the adjacent subcircuit/instance options behind it, and non-scoping option is not allowed in between. If no adjacent subcircuit/instance option is found behind the model option, an implicit <code>inst=*</code> is added.

Argument	Description
inst=inst_name subckt=subckt_name	Examines all instances of <i>inst_name</i> in all of the instances of the subcircuit defined by <i>subckt_name</i> . Instance names can contain wildcards; the subcircuit name cannot.
rmSub=subckt_name rmInst=inst_name	If one or both of these options is used, elements that satisfy the defined conditions are not checked. This option prevents devices that reside in the specified subcircuit or instance from being reported. Both options allow the use of wildcards.
skipSub=suckt_name skipInst=inst_name	If one or both options are used, subcircuits matching subcircuits and instances prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
limitMos=num	Defines the expansion limit from the given voltage source that is not to exceed the specified number of transistors. You can specify this option with global settings.
vlth=vl	Specifies the low- voltage threshold such that CCK traces from the voltage sources with values less than or equal to <i>vl</i> . You can specify this option with global settings.
vhth=vh	Specifies the high-voltage threshold such that CCK traces only from sources with values equal to or greater than <i>vh</i> . You can specify this option with global settings.
vnth=v1	Specifies the n-MOSFET threshold voltage. The <i>v1</i> value is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v2	Specifies the p-MOSFET threshold voltage. The <i>v2</i> value is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.
lvd=v7	Specifies the lower bound of drain node voltage to be <i>v7</i> . If a device drain terminal is less than <i>v7</i> , a warning is issued. You can specify this option with global settings.
uvd=v8	Specifies the drain node voltage upper boundary. If a device's drain terminal is greater than <i>v8</i> , a warning is issued. You can specify this option with global settings.
lvg=v9	Specifies the gate node voltage lower boundary. If a device gate terminal is less than <i>v9</i> , a warning is issued. You can specify this option with global settings.

Appendix I: HSIM CCK

Running Device Voltage Analysis for Transistors

Argument	Description
uvg=v10	Specifies the gate node voltage upper boundary. If a device gate terminal is greater than <i>v10</i> , a warning is issued. You can specify this option with global settings.
lvs=v11	Specifies the source node voltage lower boundary. If a device source terminal is less than <i>v11</i> , a warning is issued. You can specify this option with global settings.
uvs=v12	Specifies the source node voltage upper boundary. If a device source terminal is greater than <i>v12</i> , a warning is issued. You can specify this option with global settings.
lvb=v13	Specifies the bulk node voltage lower boundary. If a device bulk terminal is less than <i>v13</i> , a warning is issued. You can specify this option with global settings.
uvb=v14	Specifies the bulk node voltage upper boundary. If a device bulk terminal is greater than <i>v14</i> , a warning is issued. You can specify this option with global settings.
cond='expression' ...	Specifies that a voltage constraint can be a Boolean expression such as: cond ='(uvp < 1 lvn >= 0.5)' or mathematical operations (+,-,* , /, and abs()). Operators, +,-,* , / and abs() are allowed for both sides of inequality operations. An expression can contain the following parameters: lvp, uvp, lvn, uvn, lvd, uvd, lvg, uvg, lvs, uvs, lvb, uvb, w (Mos width), l (Mos length), && ('and' operator), and ('or' operator). You can specify multiple expressions. Each expression is evaluated, checked, and reported. If any one of the conditions is not met, it is reported.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
separate_file=[0 1]	Specifies the default that causes all warning messages to be reported in the <i>hsim_output_prefix.cck</i> file. If <i>separate_file</i> =1, warning messages are reported in a separate <i>hsim_output_prefix.cckmosv_command_tag</i> file. You can specify this option with global settings.
rptTrace=[0 1 2]	Indicates whether (1) or not (0) CCK reports a conductive path for a transistor that leads from its nodes to a voltage source. If you set a value of 2, CCK also indicates the conductive path information as in 1, but also reports the additional vt-drop information in the violation report. The default value is 0. You can specify this option with global settings.

Argument	Description
<code>risePmosFallNmos= [0 1]</code>	Indicates whether (1) or not (0) device voltage analysis propagation applies restrictions with positive, zero, or negative voltage sources relative to element type. If <code>risePmosFallNmos</code> is set to 1, positive voltage sources can only be propagated through p-MOSFET devices, and negative or zero voltage sources can only be propagated through n-MOSFET devices. The default value is 0. You can specify this option with global settings.
<code>subinfo= [0 1]</code>	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
<code>1= [0 1]</code>	Indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.
<code>pwl_time=time</code>	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (<code>v(time)</code>) used for propagation is defined by the pwl voltage source. This voltage value (<code>v(time)</code>) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
<code>fvsrc='(e_name,vmin,vmax)</code> <code>x)</code>	Propagates from voltage source element <code>e_name</code> with values <code>vmin</code> and <code>vmax</code> . If <code>vhth</code> is set, constant voltage sources greater than or equal to the value set for <code>vhth</code> are used as a starting point for propagation. If <code>vlth</code> is set, constant voltage sources less than or equal to the value set for <code>vlth</code> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
<code>fvsrcnd='(vsr_node_name</code> <code>e,vmin,vmax)</code>	Propagates from voltage source node <code>vsr_node_name</code> with values <code>vmin</code> and <code>vmax</code> . If <code>vhth</code> is set, constant voltage sources greater than or equal to the value set for <code>vhth</code> are used as a starting point for propagation. If <code>vlth</code> is set, constant voltage sources less than or equal to the value set for <code>vlth</code> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
<code>connSub=subckt_name</code>	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified subcircuit name.
<code>connInst=inst_name</code>	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.
<code>connNode=node_name</code>	Limits elements to be reported to those that have a direct connection to the specified node name.

Appendix I: HSIM CCK

Running Device Voltage Analysis for Transistors

Argument	Description
autoFvsrcnd= [0 1]	If set to 1, the static CCK command automatically looks for any pwl or pulse supply voltage in the netlist, scans the supply waveform, and identifies its minimum and maximum peak value (Vmin, Vmax), respectively. The identified values form a voltage range (Vmin, Vmax) and are carried throughout the static voltage propagation process. The default is 0, which means that the automation capability is disabled. In addition, if autoFvsrcnd=1 is applied together with fvsrcnd=() with a designated voltage range on a particular Vsrc node (means the user-defined voltage range is different than (Vmin, Vmax) identified by autoFvsrcnd=1), the particular Vsrc node propagates twice, once with the (Vmin, Vmax) by autoFvsrcnd=1, and the other by the user-defined range by fvsrcnd=().
rmax4init = value	Specifies whether a resistor is considered to be open in the initialization propagation stage. During the initialization propagation stage, if the resistance R of a resistor r is greater than or equal to value ($R \geq value$), then r is considered as OPEN in the initialization propagation stage.
vio_only [0 1 2]	This option is valid when the output is in the standard output format and the rptTrace option is set to 1. See Examples: Using the vio_only and cond Options . <ul style="list-style-type: none"> ▪ If you specify vio_only 0, all paths from all the terminals (Drain, Gate, Source, and Base) to the higher and lower voltage sources will be reported. ▪ If you specify vio_only 1, it reports only the violation path defined in cond option. ▪ If you vio_only 2, all paths from all the terminals (Drain, Gate, Source, and Base) to the higher and lower voltage sources will be reported with the tag, violate that indicates the violated path. <p>The default is 1.</p>

Description

Among the connSub, connInst, and connNode options, if you apply more than one or mix options there is no scoping relationship and CCK operates based on a pure OR Boolean operation in reporting the violations collected through each of the specified arguments. These three arguments can use wildcard characters.

vlth and vhth act like filters; not like checking criteria. Their purpose is to filter out other voltage sources so that cckMosV starts tracing only with voltage sources lower than vlth or higher than vhth. The actual checking criteria are specified in lvg, uvg, lvd, uvd, lvs, and uvs.

Specifying the Resistance Threshold

In CCK, there are several static commands that perform the *initialization propagation* process to get the circuit ready with a working condition. This

heuristic procedure imitates the DC process but without the cost of DC analysis or simulation. As the whole procedure is heuristic, to obtain a more reasonable result from such *initialization propagation* process, CCK opens several windows to control the initialization process, that requires the users to specify specific rules to conduct the propagation process. You can now request to have another control point on the open resistors in the netlist during the *initialization propagation* process.

Note: This feature is also applicable for the `cckdiov`, `cckbjtv`, `cckcapv`, and `cckresv` commands.

You can use the option to specify the resistance threshold. For example:

```
cckMosv rmax4init=5M
```

During the *initialization propagation* stage, if you specify a positive value for the `rmax4init`, CCK will check whether the resistance (R) of a resistor is greater than or equal to (\geq) `rmax4init`. If $R \geq rmax4init$, then CCK will treat the resistor as open and it will stop the propagation when that resistor is encountered.

Such a resistor check against `rmax4init` will be performed only during the *initialization propagation* stage.

By default, if `rmax4init` is not specified, then there will be no resistor checks against `rmax4init`.

Note: The `rmax` option is used through all stages of the propagation and is not just limited to the *initialization propagation* stage. When both `rmax` and the `rmax4init` options are specified, the `rmax4init` option takes higher priority over the `rmax` option.

The behavior of the `rmax` and `rmax4init` options are described below:

- During *initialization propagation* stage, when `rmax4init` is given, CCK performs the resistor check against `rmax4init` in spite of whether or `rmax` is given or not. If $R \geq rmax4init$, the resistor is considered as open, else the resistor is considered as conducting.
- During the *initialization propagation* stage, when `rmax4init` is not specified and `rmax` is specified, CCK performs the resistor check against `rmax`. If $R \geq rmax$, the resistor is considered as open, else the resistor is considered as conducting.

Appendix I: HSIM CCK

Running Device Voltage Analysis for Transistors

- During the *initialization propagation* stage, when both `rmax4init` nor `rmax` are specified, CCK does not perform the resistor check and all resistors are considered as conducting.
- For other propagation stages (other than *initialization propagation* stage) only `rmax` (if specified) is considered for the resistor checks. `rmax4init` will have no effect during these stages.

Table 125 describes the behavior of the `rmax` and the `rmax4init` options during the initialization and other stages of propagation.

Table 125 Behavior of R, rmax, and rmax4init options

R, rmax4init, rmax	Initialization Stage	Other Stages
R=4M, rmax = 3M	R >= rmax open	R >= rmax open
R=4M, rmax4init=5M, rmax=3M	R < rmax4init not open	R >= rmax open
R=4M, rmax4init=3M, rmax=5M	R >= rmax4init open	R < rmax not open

Examples

An example of passing user-defined parameters into a CCK command is as follows:

```
.param vxx=2.5
cckMosV tagName inst=inv cond='(lvd*3 - uvs >= -vxx)'
```

The following example shows the usage of the `cond=` options:

```
(a) cond='(lvd - 0.5*uvs >= 1.5)'
(b) cond='(lvd >= 0.5*uvg + 1.5)'
(c) cond ='(uvd-lvs*0.5 > 1.5 *2)'
(d) cond ='(uvd-abs(lvs-lvg) > 1.5 *2)'
```

The following are the static device voltage checking examples:

```
cckMosv tn subckt=a0tc inst=* uvg=2.9 limitMos=2 risePmosFallNmos=1 skipInst=aa*
```

In the above example:

- All instances of `a0tc` are checked because `subckt=a0tc inst=*`.
- Since `risePmosFallNmos=1`, when a voltage source is traced, then:
 - Positive: Then go through p-MOSFET, only go through p-MOSFETS.
 - GND or Negative: Only go through n-MOSFET.

- Since `skipInst=aa*`, when tracing, instances with the prefix aa are not gone through.
- Since `limitMos=2`, only nodes reached from a voltage source by going through no more than 2 transistors are considered.
- Transistor node voltages are then checked to see if their gate node exceeds 2.9 V. Transistors that meet this criteria are reported.

```
cckMosV tn subckt=b1tr inst=* vlth=0.4 uvg=2.9
```

In the previous example, since `subckt=b1tr` and `inst=*`, all transistors in all the instantiations of `b1tr` are checked. Since `vlth=0.4`, the power supplies with less than or equal to 0.4 V are traced. Since `limitMos` is not set, any node can be traced. If a transistor gate node exceeds 2.9 V, then it is reported.

```
cckMosV tag inst=xpb1* uvg=3.5
```

In the previous example, because `inst=xpb1*`, all transistors that reside in instances of names beginning with `xpb1` are checked. If the transistor gate node exceeds 3.5 V, it is reported.

```
cckMosV tag subckt=mux inst=xadder* uvg=3.5
```

In the previous example, because `subckt=mux` and `inst=xadder*`, `cckMosV` applies both `subckt` and `inst` parameters for the limited scoping check. This means that `cckMosV` only looks for all transistors in the instance of name beginning with `xadder` and such instances need to be defined under the `subckt mux`. For example:

```
.subckt mux in1 in2 in3 out
xadder in1 in2 out adder
xinv in2 cout inv
...
.ends

.subckt mux1 in1 in2 in3 out
xadder in1 in2 out adder
inv in2 cout inv
...
.ends

x1 in1 in2 in3 out mux
x2 in4 in5 in6 out1 mux1
```

In this case, `cckMosV` looks for all transistors inside `x1.xadder(mux/adder)`, but not `x2.xadder(mux1/adder)`.

```
cckMosV tag model=nch cond='(uvg>=0.4 || lvd <-1.8)'
```

Appendix I: HSIM CCK

Running Device Voltage Analysis for Transistors

In the previous example, the syntax checks all the nch transistors. If a nch MOSFET has gate node voltage larger or equal to 0.4 V or drain voltage smaller than -1.8 V, then it is reported.

```
cckMosV tag model=nch vlth=0.5 cond='(uvg >=0.4 || lvd <-1.8)'
```

In the previous example, the syntax traces from nch MOSFET to see if it reaches any logic-0 voltage sources. If the upper bound of voltage source reached by gate node is larger than 0.4 V or the lower bound of voltage sources seen by a drain node is less than -1.8 V, then it is reported.

```
cckMosV tpc model=pch inst=* vhth=0.7 cond='(lvg <=1.1 || uvd >1.8)'
```

In the previous example, the syntax checks pch MOSFETs to find the logic-1 voltage source range using `vhth=0.7` V as seen from the gate node. If the lower bound is less than or equal to 1.1 V, it is reported. Similarly, the upper voltage source bound seen from the drain node can be checked. If it is larger than 1.8 V, then it is reported.

```
cckMosV tn model=nch inst=* vlth=0.2 uvg=0.3 limitMos=2 risePmosFallNmos=1 skipInst=xram.xwr2_7*
```

In the previous example, nch transistors are checked to determine if their nodes can reach logic-low voltage source with values less than 0.2 V. When tracing, the process can only go through two MOSFETS and must go through a nch MOSFET. If a gate node voltage is greater than 0.3 V, it is reported.

Note: Tracing can not go through any MOSFETS named `xram.xwr2_7*`.

The following example assumes this Vsrc definition in the netlist:

```
...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vddx      0 pvdd
vvde vvdex     0 pwl 0ns 0 0.1ns pvdd
vshift shift    0 pwl 0ns 0 0.1ns pvdd2
...
```

And also assumes the following command set

```
cckMosv vhth=0.5 vnth=0.3 vt=0.1 autoFvsrccnd=1
```

CCK operates equivalently with the following syntax:

```
cckMosv vhth=0.5 vnth=0.3 vt=0.1 fvsrccnd=(vvdex, 0, 1.11) fvsrccnd=(shift, 0, 0.8)
```

The report header is updated to include the `autoFvsrccnd` setting:

```
*****
* PMOS gate vsrc valu
e less than D/S vsrc:
*
*   vnth=0.300 vpth=-0.500
*   vhth=0.500
*
*   vt=0.10 num=300
*
*   autoFvsrcnd=1
*   format: instName gate_volt d/s_volt
*****
```

Examples: Using the `vio_only` and `cond` Options

The following script describes an example output when the `cond` option set to `(uvg>1.5)` and the `vio_only` option is set as 0. As the `vio_only` option is set to 0, all paths are printed.

```
Dev #1: x1.mn
Subinfo: x1.mp2
Attributes: Model=nmos, W=3e-06, L=3e-07
Connectivity:
G: b, V=(0.00 - 1.90)
D: c, V=(1.90 - 2.80)
S: d, V=(1.90 - 2.80)
B: gnd, V=0.00
Data:
Path #1: from vdd1 to b
    Attributes: V=1.90
    Status: UVG
    Dev #1: xin2.mp
        Subinfo: c
        Attributes: Model=pmos
        Connectivity:
            D: b
            S: vdd1

Path #2: from gnd to b
    Attributes: V=0.00
    Status: LVG
    Dev #1: xin2.mm
        Subinfo: c
        Attributes: Model=nmos
        Connectivity:
            D: b
            S: gnd
```

Appendix I: HSIM CCK

Running Device Voltage Analysis for Transistors

```
Path #3: from vdd0 to c
    Attributes: V=2.80
    Status: UVD
    Dev #1: x1.mp1
        Subinfo: sub
        Attributes: Model=pmos
        Connectivity:
        D: vdd0
        S: c
Path #4: from vdd0 to c
    Attributes: V=1.90
    Status: LVD
    Dev #1: x1.mp2
        Subinfo: sub
        Attributes: Model=pmos
        Connectivity:
        D: vdd0
        S: d
    Dev #2: x1.mn
        Attributes: Model=nmos
        Connectivity:
        D: c
        S: d
Path #5: from vdd0 to d
    Attributes: V=2.80
    Status: UVS
    Dev #1: x1.mp2
        Subinfo: sub
        Attributes: Model=pmos
        Connectivity:
        D: vdd0
        S: d
Path #6: from vdd0 to d
    Attributes: V=1.90
    Status: LVS
    Dev #1: x1.mp1
        Subinfo: sub
        Attributes: Model=pmos
        Connectivity:
        D: vdd0
        S: c

    Dev #2: x1.mn
        Subinfo: sub
        Attributes: Model=nmos
        Connectivity:
        D: c
        S: d
Path #7: from gnd to gnd
    Attributes: V=0.00
    Status: UVB
Path #8: from gnd to gnd
    Attributes: V=0.00
    Status: LVB
```

If you set the `vio_only` option to 1, in the same example, only the following path is reported.

```

Dev #1: x1.mn
Subinfo: x1.mp2
Attributes: Model=nmos, W=3e-06, L=3e-07
Connectivity:
  G: b, V=(0.00 - 1.90)
  D: c, V=(1.90 - 2.80)
  S: d, V=(1.90 - 2.80)
  B: gnd, V=0.00
Data:
  Path #1: from vdd1 to b
    Attributes: V=1.90
    Status: UVG
  Dev #1: xin2.mp
    Subinfo: c
    Attributes: Model=pmos
    Connectivity:
      D: b

```

If you set the `vio_only` option is set as 2, all the paths are reported along with the `violate` tag indicating the violation path. This case is described in the following example.

```

Dev #1: x1.mn
Subinfo: x1.mp2
Attributes: Model=nmos, W=3e-06, L=3e-07
Connectivity:
  G: b, V=(0.00 - 1.90)
  D: c, V=(1.90 - 2.80)
  S: d, V=(1.90 - 2.80)
  B: gnd, V=0.00
Data:

  Path #1: from vdd1 to b
    Attributes: V=1.90
    Status: UVG, violate
  Dev #1: xin2.mp
    Subinfo: c
    Attributes: Model=pmos
    Connectivity:
      D: b
      S: vdd1

  Path #2: from gnd to b
    Attributes: V=0.00
    Status: LVG
  Dev #1: xin2.mn
    Subinfo: c
    Attributes: Model=nmos
    Connectivity:
      D: b
      S: gnd

```

Appendix I: HSIM CCK

Running Device Voltage Analysis for Transistors

```
Path #3: from vdd0 to c
    Attributes: V=2.80
    Status: UVD
    Dev #1: x1.mp1
        Subinfo: sub
        Attributes: Model=pmos
        Connectivity:
        D: vdd0
        S: c
Path #4: from vdd0 to c
    Attributes: V=1.90
    Status: LVD
    Dev #1: x1.mp2
        Subinfo: sub
        Attributes: Model=pmos
        Connectivity:
        D: vdd0
        S: d
    Dev #2: x1.mn
        Attributes: Model=nmos
        Connectivity:
        D: c
        S: d

Path #5: from vdd0 to d
    Attributes: V=2.80
    Status: UVS
    Dev #1: x1.mp2
        Subinfo: sub
        Attributes: Model=pmos
        Connectivity:
        D: vdd0
        S: d

Path #6: from vdd0 to d
    Attributes: V=1.90
    Status: LVS
    Dev #1: x1.mp1
        Subinfo: sub
        Attributes: Model=pmos
        Connectivity:
        D: vdd0
        S: c
    Dev #2: x1.mn
        Subinfo: sub
        Attributes: Model=nmos
        Connectivity:
        D: c
        S: d
Path #7: from gnd to gnd
    Attributes: V=0.00
    Status: UVB
Path #8: from gnd to gnd
    Attributes: V=0.00
    Status: LVB
```

Running Device Voltage Analysis for BJT Devices

You use the `cckBjtV` command to run device voltage analysis for bjt devices.

cckBjtV

Runs device voltage analysis for bjt devices.

Syntax

```
cckBjtV tag <model=m> <inst=instName> <subckt=subckt_name>
    <rmSub=subckt_name> <rmInst=instName>
    <skipSub=subckt_name> <skipInst=instName>
    <vnth=v1> <vpth=v2> <lvb=v3> <uvb=v4> <lve=v5> <uve=v6>
    <lv<sub>c</sub>=v7> <uv<sub>c</sub>=v8> <cond='expression' ...>
    <separate_file=[0|1]> <risePmosFallNmos=[0|1]>
    <rptTrace=[0|1|2]>
```

Argument	Description
<code>tag</code>	Specifies a label in the report file to make searching easier.
<code>model=m</code>	Causes CCK to go through all instances of model m during path tracing. CCK does not go through other model devices.
<code>inst=instName</code> <code>subckt=subckt_name</code>	Examines all instances of <code>inst_name</code> or <code>subckt_name</code> in all of the instances of <code>subckt_name</code> . Instance names can contain wildcards. The subcircuit name cannot.
<code>rmSub=subckt_name</code> <code>rmInst=instName</code>	Suppresses the specified subcircuits and instances.
<code>skipSub=subckt_name</code> <code>skipInst=instName</code>	If one or both options are used, subcircuits matching <code>subckt_name</code> and instances matching <code>inst_name</code> prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
<code>vnth=v1</code>	Specifies the n-MOSFET threshold voltage to be v1. This option is required if there are MOSFET devices on the tracing paths. The default is 0.5 V.
<code>vpth=v2</code>	Specifies the p-MOSFET threshold voltage to be v2. This parameter is also required on the tracing paths that have MOSFET devices. The default is -0.5 V.

Appendix I: HSIM CCK

Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices

Argument	Description
lvb=v3	Specifies the lower bound of the base node voltage to be v3. Devices with voltages less than v3 cause a warning to be printed in the report file.
uvb=v4	Specifies the upper bound of the base node voltage. Nodes with voltages higher than v4 cause a warning to be printed in the report.
lve=v5 uve=v6 lvc=v7 uvc=v8	Specifies the lower and upper boundaries of the emitter and collector node voltages. Nodes in these bjt elements with voltages less than lv or higher than uv cause a warning to be printed in the report file.
cond='expression' ... separate_file=[0 1] risePmosFallNmos=[0 1] rptTrace=[0 1 2]	<p>A voltage constraint can be a Boolean expression such as: cond='(upv < 1 lvn >= 0.5) or mathematical operations (+,-,* , /, and abs()). Operators, +,-,* , / and abs(), are allowed for both side of inequality operations. <expression> can contain the parameters lvp, upv, lvn, uvn, lvd, uvd, lvg, uvg, lvs, uvs, lvb, uvb, && ('and' operator), and, ('or' operator). When this expression is true, a warning is generated.</p> <p>You can specify multiple expressions. Each expression is evaluated, checked, and reported. If any one of the conditions is not met, it is reported.</p> <p>Specifies the default that causes all warning messages to be reported in the <i>hsim_output_prefix.cck</i> file. If separate_file=1, warning messages are reported in a separate <i>hsim_output_prefix.cckmosv_command_tag</i> file. You can specify this option with global settings.</p> <p>The default is 0, which sets no restrictions on the types of devices checking goes through. If risePmosFallNmos=1, path tracing from a positive voltage source goes through p-MOSFET devices, and if path tracing is from a ground or negative voltage source, checking goes through n-MOSFET devices.</p> <p>When rptTrace=1, if the voltage of a specified node exceeds the specified limits, both the instance relating to that node and the paths leading to such a voltage source are reported. Multiple paths are reported if the node has more than one path to that voltage source. The default value is 0.</p>

Examples

```
cckBjtV bjt_report model=BMOD inst=* uvb=2.0 uvc=2.0 vnth=1 vpth=1 rptTrace=1
```

Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices

With some analog designs, it is important to check the voltage for 2-terminal devices such as capacitors, resistors, and diodes. *cckCapV* performs a static voltage check on user-specified capacitors similar to the tests conducted by

`cckMosV`. A warning message is printed to the `hsim.cckcapv` or `output.cckcapv` report file if a violation occurs.

`cckResV` and `cckDioV` perform static voltage checking for specified resistors and diodes. A warning message is reported in either the `.cckResV` or `.cckDioV` report files if a violation occurs.

The syntax and parameters for `cckCapV`, `cckDioV`, and `cckResV` are described in this section.

cckCapV

Performs analysis checks on capacitors. Voltage propagation results are compared against your defined rules.

Syntax

```
cckCapV tag <model=m> <inst=instName> <subckt=subckt_name>
    <rmSub=subckt_name> <rmInst=instName>
    <skipSub=subckt_name> <skipInst=instName>
    <limitMos=num> <vhth=vh> <vlth=vl> <vnth=v1>
    <vpth=v2> <lvp=v3> <uvp=v4> <lvn=v5> <uvn=v6>
    <cond='expression' ...> <num=num> <separate_file=[0|1]>
    <risePmosFallNmos=[0|1]> <rptTrace=[0|1|2]>
    <subinfo=[0|1]> <filterAlert=[0|1]>
    <pwl_time=time> <fvsrcc='(e_name,vmin,vmax)'>
    <fvsrcnd='(vsrcc_node_name,vmin,vmax)'>
    <autoFvsrccnd=[0|1]>
```

Argument	Description
<code>tag</code>	Specifies a label in the report file to make searching easier.
<code>model=m</code>	Causes CCK to go through all instances of model m during path tracing. CCK does not go through other model devices.
<code>inst=instName</code> <code>subckt=subckt_name</code>	Examines all instances of <code>inst_name</code> or <code>subckt_name</code> in all of the instances of <code>subckt_name</code> . Instance names can contain wildcards. The subcircuit name cannot.
<code>e</code>	
<code>rmSub=subckt_name</code> <code>rmInst=instName</code>	Suppresses the specified subcircuits and instances.

Appendix I: HSIM CCK

Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices

Argument	Description
skipSub=subckt_name me skipInst=instName	If one or both options are used, subcircuits matching <i>subckt_name</i> and instances matching <i>inst_name</i> prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
limitMos=num	Defines the expansion limit from the given voltage source that is not to exceed specified number transistors. You can specify this option with global settings.
vhth=vh	Specifies the high-threshold CCK traces only from sources with values equal to or greater than the specified <i>vh</i> value. You can specify this option with global settings.
vlth=vl	Specifies the low-threshold CCK traces only from sources with values equal to or less than the specified <i>vh</i> value. You can specify this option with global settings.
vnth=v1	Specifies the n-MOSFET threshold voltage to be used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v2	Specifies the p-MOSFET threshold voltage to be used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.
lvp=v3	Specifies the lower bound of the anode voltage. If a resistor, capacitor, or diode anode voltage is less than the specified value, a warning is issued. You can specify this option with global settings.
uvp=v4	Specifies the upper bound of the anode voltage. If a resistor, capacitor, or diode anode voltage is greater than the specified value, a warning is issued. You can specify this option with global settings.
lvn=v5	Specifies the lower bound of the cathode voltage. If a resistor, capacitor, or diode cathode voltage is less than the specified value, a warning is issued. You can specify this option with global settings.
uvn=v6	Specifies the upper bound of the cathode voltage. If a resistor, capacitor, or diode cathode voltage is greater than the specified value, a warning is issued. You can specify this option with global settings.
cond='expression' ...	A voltage constraint can be a Boolean expression such as: cond='(uvp < 1 lvn >= 0.5) or mathematical operations (+,-,* , /, and abs()). Operators, +,-,* , / and abs(), are allowed for both side of inequality operations. <expression> can contain the parameters lvp, uvp, lvn, uvn, lvd, uvd, lvg, uvg, lvs, uvs, lvb, uvb, && ('and' operator), and, ('or' operator). When this expression is true, a warning is generated. You can specify multiple expressions. Each expression is evaluated, checked, and reported. If any one of the conditions is not met, it is reported.

Argument	Description
num=num	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
separate_file=[0 1]	Specifies the default that causes all warning messages to be reported in the <i>hsim_output_prefix.cck</i> file. If <i>separate_file</i> =1, warning messages are reported in a separate <i>hsim_output_prefix.cckmosv_command_tag</i> file. You can specify this option with global settings.
risePmosFallNmos=[0 1]	The default is 0, which sets no restrictions on the types of devices checking goes through. If <i>risePmosFallNmos</i> =1, path tracing from a positive voltage source goes through p-MOSFET devices, and if path tracing is from a ground or negative voltage source, checking goes through n-MOSFET devices.
rptTrace=[0 1 2]	When <i>rptTrace</i> =1, if the voltage of a specified node exceeds the specified limits, both the instance relating to that node and the paths leading to such a voltage source are reported. Multiple paths are reported if the node has more than one path to that voltage source. The default value is 0 if <i>rtpTrace</i> is not specified
subinfo=[0 1]	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
filterAlert=[0 1]	Indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.
pwl_time=time	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (<i>v(time)</i>) used for propagation is defined by the <i>pwl</i> voltage source. This voltage value (<i>v(time)</i>) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
fvsrc='(e_name, vmin, vmax)'	Propagates from voltage source element <i>e_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
fvsrcnd='(vsr_node_name, vmin, vmax)'	Propagates from voltage source node <i>vsr_node_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.

Appendix I: HSIM CCK

Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices

Argument	Description
autoFvsrcnd= [0 1]	If set to 1, the static CCK command automatically looks for any pwl or pulse supply voltage in the netlist, scans the supply waveform, and identifies its minimum and maximum peak value (Vmin, Vmax), respectively. The identified values form a voltage range (Vmin, Vmax) and are carried throughout the static voltage propagation process. The default is 0, which means that the automation capability is disabled. In addition, if autoFvsrcnd=1 is applied together with fvsrcnd=() with a designated voltage range on a particular Vsrc node (means the user-defined voltage range is different than (Vmin, Vmax) identified by autoFvsrcnd=1), the particular Vsrc node propagates twice, once with the (Vmin, Vmax) by autoFvsrcnd=1, and the other by the user-defined range by fvsrcnd=().

cckDioV

Performs analysis checks on diodes. Voltage propagation results are compared against your defined rules.

Syntax

```
cckDioV tag <model=m> <inst=instName> <subckt=subckt_name>
<rmSub=subckt_name> <rmInst=instName>
<skipSub=subckt_name> <skipInst=instName>
<limitMos=num> <vhth=vh> <vlth=vl> <vnth=v1>
<vpth=v2> <lvp=v3> <uvp=v4> <lvn=v5> <uvn=v6>
<cond='expression' ...> <num=num> <separate_file=[0|1]>
<risePmosFallNmos=[0|1]> <rptTrace=[0|1|2]>
<subinfo=[0|1]> <filterAlert=[0|1]>
<pwl_time=time> <fvsrc='(e_name, vmin, vmax)'>
<fvsrcnd='(vsrc_node_name, vmin, vmax)'>
<autoFvsrcnd=[0|1]>
```

Argument	Description
tag	Specifies a label in the report file to make searching easier.
model=m	Causes CCK to go through all instances of model m during path tracing. CCK does not go through other model devices.
inst=instName subckt=subckt_name e	Examines all instances of <i>inst_name</i> or <i>subckt_name</i> in all of the instances of <i>subckt_name</i> . Instance names can contain wildcards. The subcircuit name cannot.

Argument	Description
rmSub=subckt_name rmInst=instName	Suppresses the specified subcircuits and instances.
skipSub=subckt_name skipInst=instName	If one or both options are used, subcircuits matching <i>subckt_name</i> and instances matching <i>inst_name</i> prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
limitMos=num	Defines the expansion limit from the given voltage source that is not to exceed specified number transistors. You can specify this option with global settings.
vhth=vh	Specifies the high-threshold CCK traces only from sources with values equal to or greater than the specified <i>vh</i> value. You can specify this option with global settings.
vlth=vl	Specifies the low-threshold CCK traces only from sources with values equal to or less than the specified <i>vh</i> value. You can specify this option with global settings.
vnth=v1	Specifies the n-MOSFET threshold voltage to be used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v2	Specifies the p-MOSFET threshold voltage to be used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.
lvp=v3	Specifies the lower bound of the anode voltage. If a resistor, capacitor, or diode anode voltage is less than the specified value, a warning is issued. You can specify this option with global settings.
uvp=v4	Specifies the upper bound of the anode voltage. If a resistor, capacitor, or diode anode voltage is greater than the specified value, a warning is issued. You can specify this option with global settings.
lvn=v5	Specifies the lower bound of the cathode voltage. If a resistor, capacitor, or diode cathode voltage is less than the specified value, a warning is issued. You can specify this option with global settings.
uvn=v6	Specifies the upper bound of the cathode voltage. If a resistor, capacitor, or diode cathode voltage is greater than the specified value, a warning is issued. You can specify this option with global settings.

Appendix I: HSIM CCK

Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices

Argument	Description
cond='expression' ... num=num	A voltage constraint can be a Boolean expression such as: cond='(uvp < 1 lvn >= 0.5) or mathematical operations (+,-,* , /, and abs()). Operators, +,-,* , / and abs(), are allowed for both side of inequality operations. <expression> can contain the parameters lvp, uvp, lvn, uvn, lvd, uvd, lvg, uvg, lvs, uvs, lvb, uvb, && ('and' operator), and, ('or' operator). When this expression is true, a warning is generated. You can specify multiple expressions. Each expression is evaluated, checked, and reported. If any one of the conditions is not met, it is reported.
separate_file=[0 1]	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
risePmosFallNmos=[0 1]	Specifies the default that causes all warning messages to be reported in the <i>hsim_output_prefix.cck</i> file. If separate_file=1, warning messages are reported in a separate <i>hsim_output_prefix.cckmosv_command_tag</i> file. You can specify this option with global settings.
rptTrace=[0 1 2]	The default is 0, which sets no restrictions on the types of devices checking goes through. If risePmosFallNmos=1, path tracing from a positive voltage source goes through p-MOSFET devices, and if path tracing is from a ground or negative voltage source, checking goes through n-MOSFET devices.
subinfo=[0 1]	When rptTrace=1, if the voltage of a specified node exceeds the specified limits, both the instance relating to that node and the paths leading to such a voltage source are reported. Multiple paths are reported if the node has more than one path to that voltage source. The default value is 0 if rptTrace is not specified
filterAlert=[0 1]	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
pwl_time=time	Indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.
	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (v(time)) used for propagation is defined by the pwl voltage source. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.

Argument	Description
<code>fvsrccnd='(e_name,vmin,vmax)'</code>	Propagates from voltage source element <code>e_name</code> with values <code>vmin</code> and <code>vmax</code> . If <code>vhth</code> is set, constant voltage sources greater than or equal to the value set for <code>vhth</code> are used as a starting point for propagation. If <code>vlth</code> is set, constant voltage sources less than or equal to the value set for <code>vlth</code> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
<code>fvsrccnd='(vsrc_node_name,vmin,vmax)'</code>	Propagates from voltage source node <code>vsrc_node_name</code> with values <code>vmin</code> and <code>vmax</code> . If <code>vhth</code> is set, constant voltage sources greater than or equal to the value set for <code>vhth</code> are used as a starting point for propagation. If <code>vlth</code> is set, constant voltage sources less than or equal to the value set for <code>vlth</code> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
<code>autoFvsrccnd=[0 1]</code>	If set to 1, the static CCK command automatically looks for any pwl or pulse supply voltage in the netlist, scans the supply waveform, and identifies its minimum and maximum peak value (<code>Vmin</code> , <code>Vmax</code>), respectively. The identified values form a voltage range (<code>Vmin</code> , <code>Vmax</code>) and are carried throughout the static voltage propagation process. The default is 0, which means that the automation capability is disabled. In addition, if <code>autoFvsrccnd=1</code> is applied together with <code>fvsrccnd=()</code> with a designated voltage range on a particular Vsrc node (means the user-defined voltage range is different than (<code>Vmin</code> , <code>Vmax</code>) identified by <code>autoFvsrccnd=1</code>), the particular Vsrc node propagates twice, once with the (<code>Vmin</code> , <code>Vmax</code>) by <code>autoFvsrccnd=1</code> , and the other by the user-defined range by <code>fvsrccnd=()</code> .

cckResV

Performs analysis checks on resistors. Voltage propagation results are compared against your defined rules.

Syntax

```
cckResV tag <model=m> <inst=instName> <subckt=subckt_name>
    <rmSub=subckt_name> <rmInst=instName>
    <skipSub=subckt_name> <skipInst=instName>
    <limitMos=num> <vhth=vh> <vlth=vl> <vnth=v1>
    <vpth=v2> <lvp=v3> <uvp=v4> <lvn=v5> <uvn=v6>
    <cond='expression' ...> <num=num> <separate_file=[0|1]>
    <risePmosFallNmos=[0|1]> <rptTrace=[0|1|2]>
    <subinfo=[0|1]> <filterAlert=[0|1]>
    <pwl_time=time> <fvsrccnd='(e_name,vmin,vmax)'>
    <fvsrccnd='(vsrc_node_name,vmin,vmax)'>
    <autoFvsrccnd=[0|1]>
```

Appendix I: HSIM CCK

Running Device Voltage Analysis for Capacitor, Resistor, and Diode Devices

Argument	Description
tag	Specifies a label in the report file to make searching easier.
model=m	Causes CCK to go through all instances of model <i>m</i> during path tracing. CCK does not go through other model devices.
inst=instName subckt=subckt_name e	Examines all instances of <i>inst_name</i> or <i>subckt_name</i> in all of the instances of <i>subckt_name</i> . Instance names can contain wildcards. The subcircuit name cannot.
rmSub=subckt_name rmInst=instName	Suppresses the specified subcircuits and instances.
skipSub=subckt_name me skipInst=instName	If one or both options are used, subcircuits matching <i>subckt_name</i> and instances matching <i>inst_name</i> prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
limitMos=num	Defines the expansion limit from the given voltage source that is not to exceed a specified number of transistors. You can specify this option with global settings.
vhth=vh	Specifies the high-threshold CCK traces only from sources with values equal to or greater than the specified <i>vh</i> value. You can specify this option with global settings.
vlth=vl	Specifies the low-threshold CCK traces only from sources with values equal to or less than the specified <i>vl</i> value. You can specify this option with global settings.
vnth=v1	Specifies the n-MOSFET threshold voltage to be used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v2	Specifies the p-MOSFET threshold voltage to be used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.
lvp=v3	Specifies the lower bound of the anode voltage. If a resistor, capacitor, or diode anode voltage is less than the specified value, a warning is issued. You can specify this option with global settings.
uvp=v4	Specifies the upper bound of the anode voltage. If a resistor, capacitor, or diode anode voltage is greater than the specified value, a warning is issued. You can specify this option with global settings.

Argument	Description
lvn=v5	Specifies the lower bound of the cathode voltage. If a resistor, capacitor, or diode cathode voltage is less than the specified value, a warning is issued. You can specify this option with global settings.
uvn=v6	Specifies the upper bound of the cathode voltage. If a resistor, capacitor, or diode cathode voltage is greater than the specified value, a warning is issued. You can specify this option with global settings.
cond='expression' ... cond='expression'	A voltage constraint can be a Boolean expression such as: cond='(uvp < 1 lvn >= 0.5) or mathematical operations (+,-,* , /, and abs()). Operators, +,-,* , / and abs(), are allowed for both side of inequality operations. <expression> can contain the parameters lvp, uvp, lvn, uvn, lvd, uvd, lvg, uvg, lvs, uvs, lvb, uvb, && ('and' operator), and, ('or' operator). When this expression is true, a warning is generated. You can specify multiple expressions. Each expression is evaluated, checked, and reported. If any one of the conditions is not met, it is reported.
num=num	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
separate_file=[0 1]	Specifies the default that causes all warning messages to be reported in the <i>hsim_output_prefix.cck</i> file. If separate_file=1, warning messages are reported in a separate <i>hsim_output_prefix.cckmosv_command_tag</i> file. You can specify this option with global settings.
risePmosFallNmos=[0 1]	The default is 0, which sets no restrictions on the types of devices checking goes through. If risePmosFallNmos=1, path tracing from a positive voltage source goes through p-MOSFET devices, and if path tracing is from a ground or negative voltage source, checking goes through n-MOSFET devices.
rptTrace=[0 1 2]	When rptTrace=1, if the voltage of a specified node exceeds the specified limits, both the instance relating to that node and the paths leading to such a voltage source are reported. Multiple paths are reported if the node has more than one path to that voltage source. The default value is 0 if rptTrace is not specified
subinfo=[0 1]	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
filterAlert=[0 1]	Indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.

Appendix I: HSIM CCK

Running Subcircuit-Based Voltage Analysis Using the Static Approach

Argument	Description
<code>pwl_time=time</code>	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (<code>v(time)</code>) used for propagation is defined by the pwl voltage source. This voltage value (<code>v(time)</code>) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
<code>fvsrc='(e_name, vmin, vmax)'</code>	Propagates from voltage source element <code>e_name</code> with values <code>vmin</code> and <code>vmax</code> . If <code>vhth</code> is set, constant voltage sources greater than or equal to the value set for <code>vhth</code> are used as a starting point for propagation. If <code>vlth</code> is set, constant voltage sources less than or equal to the value set for <code>vlth</code> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
<code>fvsrcnd='(vsr_node_name, vsrc_no, de_name, vmin, vmax)'</code>	Propagates from voltage source node <code>vsr_node_name</code> with values <code>vmin</code> and <code>vmax</code> . If <code>vhth</code> is set, constant voltage sources greater than or equal to the value set for <code>vhth</code> are used as a starting point for propagation. If <code>vlth</code> is set, constant voltage sources less than or equal to the value set for <code>vlth</code> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
<code>autoFvsrcnd=[0 1]</code>	If set to 1, the static CCK command automatically looks for any pwl or pulse supply voltage in the netlist, scans the supply waveform, and identifies its minimum and maximum peak value (<code>Vmin</code> , <code>Vmax</code>), respectively. The identified values form a voltage range (<code>Vmin</code> , <code>Vmax</code>) and are carried throughout the static voltage propagation process. The default is 0, which means that the automation capability is disabled. In addition, if <code>autoFvsrcnd=1</code> is applied together with <code>fvsrcnd=()</code> with a designated voltage range on a particular Vsrc node (means the user-defined voltage range is different than (<code>Vmin</code> , <code>Vmax</code>) identified by <code>autoFvsrcnd=1</code>), the particular Vsrc node propagates twice, once with the (<code>Vmin</code> , <code>Vmax</code>) by <code>autoFvsrcnd=1</code> , and the other by the user-defined range by <code>fvsrcnd=()</code> .

Running Subcircuit-Based Voltage Analysis Using the Static Approach

You use the `cckSubV` command to perform a subcircuit-based voltage analysis using the static approach.

cckSubV

Performs a static subcircuit port voltage check, which can be considered an extension of the existing static `cckMosV` command. However, unlike `cckMosV`, which performs a voltage check at the device (for example, MOSFET) level,

cckSubV provides the ability to check the port/pin voltage in a subcircuit with a regular expression rather than a device terminal-specific pattern.

Syntax

```
cckSubV tag <subckt=subckt_name> <inst=instName>
    <rmSub=subckt_name> <rmInst=instName>
    <skipInst=instName> <limitMos=num> <vhth=vh>
    <vlth=vl> <vnth=v2> <vpth=v3> <cond='expression'>
    <num=n> <separate_file=[0|1]> <rptTrace=[0|1|2]>
    <risePmosFallNmos=[0|1]> <subinfo=[0|1]>
    <filterAlert=[0|1]> <pwl_time=time>
    <fvsrc='(e_name,vmin,vmax)'>
    <fvsrcnd='(vsr_node_name,vmin,vmax)'>
```

Argument	Description
tag	Specifies a label in the report file to make searching easier.
inst=instName subckt=subckt_name e	Enforces that the node analyzed is associated with the specified <i>inst_name</i> or <i>subckt_name</i> . If you do not specify a <i>subckt_name</i> , cckSubV looks for all subcircuits in the netlist. Instance names can contain wildcards. The subcircuit name cannot.
rmSub=subckt_name rmInst=instName	Prevents nodes that reside in the specified <i>subckt_name</i> or <i>inst_name</i> from being reported. Instance names can contain wildcards. The subcircuit name cannot.
skipInst=instName	If one or both options are used, subcircuits matching <i>subckt_name</i> and instances matching <i>inst_name</i> prevent voltage propagation from occurring through elements within their coverage. This option allows the use of wildcards.
limitMos=num	Defines the expansion limit from the given voltage source that is not to exceed specified number transistors. You can specify this option with global settings.
vhth=vh	Specifies the high-threshold CCK traces only from sources with values equal to or greater than the specified <i>vh</i> value. You can specify this option with global settings.
vnth=v2	Specifies the n-MOSFET threshold voltage to be used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v3	Specifies the p-MOSFET threshold voltage to be used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.

Appendix I: HSIM CCK

Running Subcircuit-Based Voltage Analysis Using the Static Approach

Argument	Description
cond='expression'	You can use a Boolean expression to define a constraint as violation reporting criteria. If the Boolean value returned is logically true, it means the analysis satisfies the reporting criteria and the violation is reported. An expression can be formed by voltage of the node using <code>lv(<node_name>)</code> or <code>uv(<node_name>)</code> , which means the lower range and upper range of node voltage respectively.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <code>n</code> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
separate_file=[0 1]	Specifies the default that causes all warning messages to be reported in the <code>hsim_output_prefix.cck</code> file. If <code>separate_file=1</code> , warning messages are reported in a separate <code>hsim_output_prefix.cckmosv_command_tag</code> file. You can specify this option with global settings.
rptTrace=[0 1 2]	Indicates whether (1) or not (0) CCK reports a conductive path for a transistor that leads from its nodes to a voltage source. If you set a value of 2, CCK also indicates the conductive path information as in 1, but also reports the additional Vt-drop information in the violation report. The default is 0. You can specify this option with global settings.
risePmosFallNmos=[0 1]	Indicates whether (1) or not (0) device voltage analysis propagation applies restrictions with positive, zero, or negative voltage sources relative to element type. If <code>risePmosFallNmos</code> is set to 1, positive voltage sources can only be propagated through p-MOSFET devices, and negative or zero voltage sources can only be propagated through n-MOSFET devices. The default is 0. You can specify this option with global settings.
subinfo=[0 1]	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
filterAlert=[0 1]	Indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.
pwl_time=time	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (<code>v(time)</code>) used for propagation is defined by the pwl voltage source. This voltage value (<code>v(time)</code>) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
fvsrc='(e_name, vmin, vmax)'	Propagates from voltage source element <code>e_name</code> with values <code>vmin</code> and <code>vmax</code> . If <code>vhth</code> is set, constant voltage sources greater than or equal to the value set for <code>vhth</code> are used as a starting point for propagation. If <code>vlth</code> is set, constant voltage sources less than or equal to the value set for <code>vlth</code> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.

Argument	Description
fvsrcnd='(vsrc_no de_name,vmin,vmax)'	Propagates from voltage source node <i>vsrc_node_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.

Examples

```
cckSubV sub1 subckt=tg cond='lv(in) <0 || uv(in) > 1' rptTrace=1
```

Performs a subcircuit voltage check with the tag name *sub1*. According to the *subckt=tg* and the “*cond=*” expression, it looks for the voltage of node *in* defined within the *tg* subcircuit. If this node has its lower end voltage less than 0 or upper end voltage greater than 1 V, a violation is reported.

```
cckSubV sub3 subckt=tg cond='lv(in)<1 || uv(out)>3' rptTrace=1
```

The above example reports that the violations in the lower end of voltage of node *in* is less than 1 V or the upper end of voltage of node *out* is greater than 3 V.

The following example shows a *cckSubV* report:

```
* Static Device-V Analysis
* command= cckSubV
* tag=tag2
* cond='lv(out)<0 || uv(out) > 2'
* vpth=-0.500 vnth=0.500
* rptTrace=1 limitMos=0 separate_file=0
* risePmosFallNmos=0 subinfo=0 FilterAlert=0

*
* subckt= tg
*
* format: tag instName constraint-violated-node-name (minv - maxv)
tag2 x27
(2) n73 (0.00 - 3.00)
    thruI x27.mn1 (V=3)
    thruI x25.mp1 (V=3)
frmNd vdd (constant node
```

Running Diode Forward Bias Analysis

You use the *cckDiode* command to perform forward bias checks on diodes.

Appendix I: HSIM CCK

Running Diode Forward Bias Analysis

cckDiode

Performs forward bias checks on diodes.

Syntax

```
cckDiode <mode=0|-1|1|2> <num=n> <separate_file=0|1>
    <subinfo=[0|1]> <vt=vt0> <start=t1> <stop=t2>
    <model=model_name> <tag=t> <pwl_time=time>
    <scope='(subckt_name, inst_name, model_name)'>
    <skipScope='(subckt_name, inst_name, model_name)'>
    <rmScope='(subckt_name, inst_name, model_name)'>
```

Argument	Description
mode=0 -1 1 2	<p>Determines the analysis mode:</p> <ul style="list-style-type: none"> ■ 0 performs a static check. ■ -1 performs a static forward bias diode check, with filtering of negative Vsrc anode diodes. ■ 1 performs a dynamic check. ■ 2 performs both static and dynamic checks. <p>When performing static check, a diode is considered in a forward-bias condition whenever:</p> <ul style="list-style-type: none"> ■ Its anode is connected directly, or through a serial resistor/inductor, to a positive constant Vsrc. ■ Its cathode is connected directly, or through a serial resistor/inductor, to a zero or negative Vsrc. <p>You can specify this option with global settings.</p>
num=n	<p>Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If n is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.</p>
separate_file=0 1	<p>Causes all warning messages to be reported in the <i>hsim_output_prefix.cck</i> file. If separate_file=1, warning messages are reported in a separate <i>hsim_output_prefix.cckDiode</i> file. This option is valid only in static mode (mode=0). The default is 0. You can specify this option with global settings.</p>
subinfo=[0 1]	<p>Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.</p>
vt=vt0	<p>After voltage propagation, the vt0 values determines if a violation is warranted based on the following equation:</p> $V(\text{anode}) \geq V(\text{cathode}) + <\text{vt0}>$ <p>The default is 0.5. You can specify this option with global settings.</p>

Argument	Description
start=t1	Specifies the start time for vt checks.
stop=t2	Specifies the stop time for vt checks.
model=model_name	Causes CCK to go through all instances of <i>model_name</i> during path tracing. It does not consider other model devices.
tag=t	Prints the specified tag name in the report file at the beginning of a line to indicate a forward-biased diode.
pwl_time=time	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (<i>v(time)</i>) used for propagation is defined by the pwl voltage source. This voltage value (<i>v(time)</i>) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds. Note that this option only works in static mode (<i>mode=0</i>).
scope='(subckt_name, inst_name, model_name)'	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
skipScope='(subckt_name, inst_name, model_name)'	Specifies a filtering control to prevent voltage propagation from occurring through devices or nodes.
rmScope='(subckt_name, inst_name, model_name)'	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Description

In transient simulation, it is necessary to determine if any diode becomes forward-biased. If an anode voltage is greater than a cathode voltage by a threshold value, a warning message is printed. The default threshold is overwritten with the *vt* option.

Warning messages are written to the *hsim.cck* or *out_file.cck* file.

Examples

```
cckDiode mode=2 num=100 start=100n stop=500n model=pdio tag=t1 separate_file=1
```

Checks anode and cathode voltages in static mode and dynamic mode. While in dynamic mode, the checks occur between 100 ns and 500 ns for diode model *pdio*. If the anode voltage is larger than the cathode voltage by 0.5 V, a warning message is reported in the *hsim.cck* or *out_file.cck* file. When

Appendix I: HSIM CCK

Running Element Current Analysis

the number of warnings exceeds 100, the checking is stopped. The following example shows sample output:

```
*****
* Diode forward-bias checking before DC
*
* tag=t1
* model=pdio
* vt=0.5
* ith=5e-05
* seperate_file = 1
* num = 100
* start=0 stop=0
*
*****
Diode (d1) is forward-biased (anode (dvdd_o, 3.3 volt) cathode (net1, 0 volt)), via serial res/ind.
```

The following example is from dynamic mode.

```
*****
* MOS-Bulk/Diode Forward Bias and Node Voltage Check
* diode forward bias: tag=t1 model=pdio vt=0.5
* start=1e-07 stop=5e-07
*****
(t1) @100.00n, diode (d1) forward biased
      v(anode)=3.3  v(cathode)=1.67559
```

In this output sample derived from the previous example, at time 100 ns, a diode (d1) becomes forward-biased with the following parameters.

- Anode voltage: 3.3 V
- Cathode voltage: 1.67559 V

Running Element Current Analysis

You use the `cckElemI` command to run element current analysis.

cckElemI

In transient simulation, CCK monitors the current through each element. If the absolute value of the current exceeds the threshold, CCK reports the element name, current, and time. If you specify a tag, the warning is prefixed with the tag.

Syntax

```
cckElemI <start=t1> <stop=t2> <tag=t> <subckt=s> <inst=name>
    <iTh=v> <model=m> <rms=[0|1]> <avg=[0|1]> <num=n>
    <iThAbs=[0|1]>
```

Argument	Description
start=t1	Specifies the start time for vt checks.
stop=t2	Specifies the stop time for vt checks.
tag=t	Prints the specified t in the report file at the beginning of a line to indicate a warning.
subckt=s	Specifies a current check for the specified subcircuit.
inst=name	Specifies a current check for the specified instance.
iTh=v	Specifies the current threshold value.
model=m	Causes CCK to go through all instances of <i>model_name</i> during path tracing. It does not consider other model devices.
rms=0 1	Specifies checking the root-mean-square current of the specified element during a given time window (<i>t₁</i> , <i>t₂</i>). If this rms current exceeds the threshold (<i>iTh=v</i>), then the element is reported.
avg=0 1	Checks the average current in an element during a given time window. If the average current is larger than the i threshold (<i>iTh=v</i>), the element is printed.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
iThAbs=0 1	Reports the current with sign value instead of reporting the absolute value. CCK reports absolute current values by default.

Description

In transient simulation, CCK monitors the current through each element. If the absolute value of the current exceeds the threshold, CCK reports the element name, current, and time. If you specify a tag, the warning is prefixed with the tag.

Examples

```
cckElemI start=10n stop=76n tag=t1 subckt=aa inst=* iTh=1.e-6 model=pch rms=1
```

Appendix I: HSIM CCK

Running a Reference Check for Instances and Subcircuits

Within a specified time window from 10 n to 76 n, all instances in all instantiations of the aa subcircuit are examined. The root-mean-square current of each pch transistor is then computed. If the current is greater than 1 uAmp, the element is reported using a t1 tag.

Running a Reference Check for Instances and Subcircuits

You use the `cckMatchSub` command to run a reference check for instances associated with a subcircuit.

cckMatchSub

Conducts a reference check and lists all instances associated with a specified subcircuit.

Syntax

```
cckMatchSub <subckt=subcircuit_name> <ReptHierNode=[0|1]>
```

Argument	Description
<code>subckt=subcircuit_name</code>	Specifies the subcircuit name as the matching target.
<code>ReptHierNode=0 1</code>	Reports both the matched nodes and associated upstream hierarchical nodes.

Description

`cckMatchSub` conducts a reference check and lists all instances associated with a specified subcircuit and its port mapping information. While `cckPatternMatch` performs detailed pattern matching by traversing the design hierarchy or flattening certain levels during the operation, its intensive approach might require a significant amount of computing resources. `cckMatchSub` supplements `cckPatternMatch` to gain early and final design-phase matching results.

Detecting Excessive Current Path

You use the `cckExiPath` command to detect excessive current paths.

cckExiPath

Detects excessive current paths from a power supply to the ground. You can specify both current and time duration thresholds.

Syntax

```
cckExiPath <ith=cur> <tth=time> <from=vsrc1> <from=vsrc2
...> <to=gnd <to=vss ...>> <start=t1> <stop=t2>
<scope='(subckt_name, inst_name, model_name)'>
<skipScope='(subckt_name, inst_name, model_name)'>
<rmScope='(subckt_name, inst_name, model_name)'>
```

Argument	Description
<code>ith=cur</code>	Specifies the current threshold value. All the elements in a path must have current larger than the <i>i</i> th value.
<code>tth=time</code>	Specifies the time threshold value. All the elements in a path must have current larger than the <i>i</i> th value. The excessive current path must exist for more than <i>tth</i> time interval.
<code>from=vsrc1</code> <code>from=vsrc2 ...</code>	Defines the power supply. You can specify multiple nodes.
<code>to=gnd</code> <code>to=vss ...</code>	Defines the ending nodes. You can specify multiple nodes. Note that if the <code>from</code> and <code>to</code> options are not specified, <code>cckExiPath</code> searches the DC paths with all combinations of all voltage sources.
<code>start=t1</code>	Specifies the start time for vt checks.
<code>stop=t2</code>	Specifies the stop time for vt checks.
<code>scope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
<code>skipScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control to prevents voltage propagation from occurring through devices or nodes.

Appendix I: HSIM CCK

Checking for Floating Gates and Analyzing Current Sources

Argument	Description
<code>rmScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Examples

Given a netlist with VSRC nodes, VDD, VCC and VSS, if `from` and `to` are not specified, `cckExiPath` searches the paths "from VDD to VSS" and "from VCC to VSS" and from "VDD to VCC".

An example of excessive current path detection syntax is as follows:

```
cckExiPath ith=10u tth=0.6n from=vdd from=xam.d1 to=gnd start=10n stop=100n
```

`cckExiPath` starts monitoring from 10 ns to 100 ns. If a path exists from VDD or node `xam.d1` to GND, and all the elements in this path have current greater than 10 uAmp and last for more than 0.6 ns, then this path is reported. The output of the warning goes to file `hsim.cckexipath`.

The following example shows a sample `cckExiPath` output:

```
*****
* cckExiPath ith=1e-005 tth=6e-010 stop=1e-007
* from=xam.d1
* from=vdd
* to=gnd
*****
path 0: from xam.x8.n to gnd, time 5e-009 - 6e-008 (duration 5.5e-008)
    xam.xcr8.mn (eid=25508) I=0.000197307
        node gnd
path 1: from vdd to gnd, time 3.2384e-008 - 4.0598e-008 (duration 8.214e-009)
    xam.xrd.mul (eid=50071) I=0.000146329
        node xam.xu8.p1
    xam.xr.mu2 (eid=50072) I=0.000141375
        node xam.npf<1>
    xam.xrd1.mn (eid=49985) I=4.14077e-005
        node gnd
```

Checking for Floating Gates and Analyzing Current Sources

You use the `cckFloatGateIsrc` command to check for floating gates.

cckFloatGateIsrc

Checks for floating gates.

Syntax

```
cckFloatGateIsrc [0|1] <listall=0|1> <num=n>
    <subinfo=0|1> <skipport=0|1> <xdummy=[0|1|2]>
    <subckt=subckt_name> <inst=inst_name>
    <rmSub=subckt_name> <rmInst=inst_name><filterAlert=0|1>
    <scope='(subckt_name, inst_name, model_name)'>
    <skipScope='(subckt_name, inst_name, model_name)'>
    <rmScope='(subckt_name, inst_name, model_name)'>
    <diode_conducting=0|1>
```

During voltage analysis, the behavior of the diode in a circuit check device is explained in the table given below:

Table 126 Diode Conducting Rules

	Conducting Forward Bias Condition (On)	Nonconducting Reverse Bias Condition (Off)
diode	Always On unless defined as Off.	(Vanode and Vcathode are connected to the <i>conducting constant</i> voltage nodes && Vanode <= Vcathode) (Vanode is connected to a <i>conducting constant</i> voltage node && Vanode <= 0) (Vcathode is connected to a <i>conducting constant</i> voltage node && Vcathode > 0)

Appendix I: HSIM CCK

Checking for Floating Gates and Analyzing Current Sources

Arguments

Argument	Description
0 1	Turns floating gate checking off or on.
listall=0 1	Reports only one representative floating gate per gate node, and whether it is a PMOS or NMOS, when set to 0. When set to 1, reports both NMOS and PMOS devices when their gates are connected together and are floating. The default is 0. This parameter can be specified with global settings. You can specify this option with global settings.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
subinfo=0 1	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default is 0. This parameter can be specified with global settings. You can specify this option with global settings.
skipport=0 1	Indicates whether (1) or not (0) all top-level ports and nodes connected to top-level ports through resistors or inductors are excluded from the floating node report. A top-level port is defined as a subcircuit port in which the subcircuit is defined as the top-level instance using either the <code>-top</code> HSIM command line or the option <code>HSIMTOP</code> command. When <code>skipport=1</code> , <code>cckFloatgateIstc</code> requires either <code>-top</code> or <code>HSIMTOP</code> to locate the top-level ports.
xdummy= [0 1 2]	Specifies how to report floating gate MOSFETs when a MOSFET qualifies as "dummy". The values you can specify are: <ul style="list-style-type: none"> ■ 0 (default) for when a dummy MOSFET has no effect on how a floating gate MOSFET is reported. ■ 1 does the filtering for floating gate MOSFET when both its drain/source are floating, or one of its drain/source is floating while the other is non-floating nor non-dangling. ■ 2 does the filtering of <code>xdummy=1</code> and, filters out floating gate MOSFET if one of its drain/source is dangling.
subckt=subckt_name inst=inst_name	If you specify either a subcircuit or instance name, or both, only gate nodes that reside in the defined conditions are checked. You can use a wildcard character in an instance name, but not in a subcircuit name.
rmSub=subckt_name rmInst=inst_name	If you specify either a subcircuit or instance name, or both, gate nodes that reside in the defined conditions are not checked. You can use a wildcard character in both a subcircuit and instance name.

Argument	Description
filterAlert=0 1	filterAlert=1 checks against the matching result by the subcircuit and instance names specified in the same cckFloatGateIsrc command line. The cckFloatGateIsrc check terminates and errors out if there is no matching subcircuit and instance. The default is 1.
scope='(subckt_name, inst_name, model_name)'	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
skipScope='(subckt_name , inst_name, model_name)'	Specifies a filtering control to prevent voltage propagation from occurring through devices or nodes.
rmScope='(subckt_name, inst_name, model_name)'	Specifies a filtering mechanism to prevent devices or nodes from being reported.
diode_conducting = 0 1	Specifies whether the diodes are in conducting or non-conducting state. If you specify 0, the diodes are considered to be in non-conducting state. If you specify 1, the conducting rules of the diodes described in Table 126 on page 953 are evaluated. If the conducting rules are satisfied, the diodes are considered to be in conducting state. The default is 1.

Examples

```
cckFloatGateIsrc 1
```

The following output example is the floating gate nodes and floating isrc nodes (hsim.cck or out_file.cck) resulting from the previous example. Because the listall option is not used, only representative devices are reported.

```
*****
* Floating gate nodes
*****
mos (x1.mn) gate node (x1.p) floats
*****
* Floating isrc nodes
*****
current src node (x2.i) floats
```

The HSIMCCK script given below indicates that the diodes are in non-conducting state.

```
cckFloatGateIsrc diode_conducting=0
```

An intuitive example for listall is that three inverters, inv1, inv2, and inv3, have their gates connected together to an input node, but nothing drives the input.

Appendix I: HSIM CCK

Checking NMOS Bulk Connections

- If `listall=0` (default), `cckFloatGateIsrc` reports only one representative MOS device from the `inv1`, `inv2`, and `inv3` as a floating gate.
- If `listall=1`, `cckFloatGateIsrc` reports all the MOS devices in `inv1`, `inv2`, and `inv3` as floating gates.

Checking NMOS Bulk Connections

You use the `cckNmosB_gt_DS` command to perform analysis check on n-MOSFET transistors.

cckNmosB_gt_DS

Performs analysis checks on n-MOSFET transistors.

Syntax

```
cckNmosB_gt_DS <model=model_name> <subckt=subckt_name>
    <inst=inst_name> <rmSub=subckt_name>
    <rmInst=inst_name> <skipSub=suckt_name>
    <skipInst=inst_name> <vlth=vl> <vt=vt0>
    <vnth=v1> <vpth=v2> <num=n> <rptv=node_name>
    <rptTrace=[0|1|2]> <subinfo=[0|1]>
    <filterAlert=[0|1]> <pwl_time=time>
    <fvsrc='(e_name,vmin,vmax)'>
    <fvsrcnd='(vsr_node_name,vmin,vmax)'>
    <connSub=subckt_name> <connInst=inst_name>
    <connNode=node_name> <autoFvsrcnd=[0|1]>
    <scope='(subckt_name, inst_name, model_name)'>
    <skipScope='(subckt_name, inst_name, model_name)'>
    <rmScope='(subckt_name, inst_name, model_name)'>
```

Argument	Description
<code>model=model_name</code>	Causes CCK to go through all instances of <code>model_name</code> during path tracing and does not consider other model devices. The <code>model</code> option only applies to the adjacent subcircuit/instance options behind it, and non-scoping option is not allowed in between. If no adjacent subcircuit/instance option is found behind the <code>model</code> option, an implicit <code>inst=*</code> is added.

Argument	Description
subckt=subckt_name inst=inst_name	Examines all instances of <i>inst_name</i> in all of the instances of the subcircuit defined by <i>subckt_name</i> . Instance names can contain wildcards; the subcircuit name cannot.
rmSub=subckt_name rmInst=inst_name	If one or both of these options is used, elements that satisfy the defined conditions are not checked. This option prevents devices that reside in the specified subcircuit or instance from being reported. Both options allow the use of wildcards.
skipSub=suckt_name skipInst=inst_name	If one or both options are used, subcircuits matching subcircuits and instances prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
vlth=v1	Specifies the low- voltage threshold such that CCK traces from the voltage sources with values less than or equal to <i>v1</i> . You can specify this option with global settings.
vt=vt0	After voltage propagation the specified <i>vt0</i> value is used to determine if a violation is warranted based on the following equation: $\max(V_b) \geq \min(V_d/s) + <vt0>$ The default value is 0.3. You can specify this option with global settings.
vnth=v1	Specifies the n-MOSFET threshold voltage. The <i>v1</i> value is used during voltage propagation to determine whether a full voltage, <i>vt</i> dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v2	Specifies the p-MOSFET threshold voltage. The <i>v2</i> value is used during voltage propagation to determine whether a full voltage, <i>vt</i> dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings
rptv=node_name	Reports the voltage ranges of the specified nodes. You can use wildcard characters in the node name.

Appendix I: HSIM CCK

Checking NMOS Bulk Connections

Argument	Description
rptTrace=[0 1 2]	Indicates whether (1) or not (0) CCK reports a conductive path for a transistor that leads from its nodes to a voltage source. If you set a value of 2, CCK also indicates the conductive path information as in 1, but also reports the additional vt-drop information in the violation report. The default value is 0. You can specify this option with global settings.
subinfo=[0 1]	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
filterAlert=[0 1]	Indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.
pwl_time=time	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (v(time)) used for propagation is defined by the pwl voltage source. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
fvsrc='(e_name,vmin,vmax)'	Propagates from voltage source element <i>e_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
fvsrcnd='(vsr_node_name,e,vmin,vmax)'	Propagates from voltage source node <i>vsr_node_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
connSub=subckt_name	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified subcircuit name.
connInst=inst_name	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.
connNode=node_name	Limits elements to be reported to those that have a direct connection to the specified node name.

Argument	Description
autoFvsrCnd= [0 1]	If set to 1, the static CCK command automatically looks for any pwl or pulse supply voltage in the netlist, scans the supply waveform, and identifies its minimum and maximum peak value (Vmin, Vmax), respectively. The identified values form a voltage range (Vmin, Vmax) and are carried throughout the static voltage propagation process. The default is 0, which means that the automation capability is disabled. In addition, if autoFvsrCnd=1 is applied together with fvsrCnd=() with a designated voltage range on a particular Vsrc node (means the user-defined voltage range is different than (Vmin, Vmax) identified by autoFvsrCnd=1), the particular Vsrc node propagates twice, once with the (Vmin, Vmax) by autoFvsrCnd=1, and the other by the user-defined range by fvsrCnd=().
scope='(subckt_name, inst_name, model_name)'	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
skipScope='(subckt_name , inst_name, model_name)'	Specifies a filtering control to prevent voltage propagation from occurring through devices or nodes.
rmScope='(subckt_name, inst_name, model_name)'	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Description

cckNmosB_gt_DS performs analysis checks on n-MOSFET transistors specifically to see if the maximum bulk voltage (maxVb) is greater than the minimum drain/source voltages (minVd/s). Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared. If `maxVb >= minVd/s + vt` warnings are generated. The num option is used to limit the number of warnings that are reported.

Examples

The following example assumes this voltage source in the netlist:

```
cckNmosB_gt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrCnd=1
```

And also assumes the following command:

```
cckNmosB_gt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrCnd=1
```

The report header is updated to include the `autoFvsrCnd` setting:

Appendix I: HSIM CCK

Finding Potentially Conducting NMOS Devices

```
*****
* PMOS gate vsrc value
less than D/S vsrc:
*
*   vnth=0.300 vpth=-0.500
*   vhth=0.500
*
*   vt=0.10 num=300
*
*   autoFvsrCnd=1
*   format: instName gate_volt d/s_volt
*****
```

Finding Potentially Conducting NMOS Devices

You use the `cckNmosG_gt_DS` command to find potentially conducting NMOS devices.

cckNmosG_gt_DS

Performs analysis checks to find potentially conducting NMOS devices.

Syntax

```
cckNmosG_gt_DS <model=model_name> <subckt=subckt_name>
    <inst=inst_name> <rmSub=subckt_name>
    <rmInst=inst_name> <skipSub=suckt_name>
    <skipInst=inst_name> <vlth=vl> <vt=vt0>
    <vnth=v1> <vpth=v2> <num=n> <rptv=node_name>
    <rptTrace=[0|1|2]> <subinfo=[0|1]>
    <filterAlert=[0|1]> <pwl_time=time>
    <fvsrc='(e_name,vmin,vmax)'>
    <fvsrcCnd='(vsrc_node_name,vmin,vmax)'>
    <connSub=subckt_name> <connInst=inst_name>
    <connNode=node_name> <autoFvsrCnd=[0|1]>
    <skipPcap=0|1> <scope='(subckt_name, inst_name,
model_name)'> <skipScope='(subckt_name, inst_name,
model_name)'> <rmScope='(subckt_name, inst_name,
model_name)'>
```

Argument	Description
model=model_name	Causes CCK to go through all instances of <i>model_name</i> during path tracing and does not consider other model devices. The model option only applies to the adjacent subcircuit/instance options behind it, and non-scoping option is not allowed in between. If no adjacent subcircuit/instance option is found behind the model option, an implicit <i>inst=*</i> is added.
subckt=subckt_name inst=inst_name	Examines all instances of <i>inst_name</i> in all the instances of the subcircuit defined by <i>subckt_name</i> . Instance names can contain wildcards; the subcircuit name cannot.
rmSub=subckt_name rmInst=inst_name	If one or both of these options is used, elements that satisfy the defined conditions are not checked. This option prevents devices that reside in the specified subcircuit or instance from being reported. Both options allow the use of wildcards.
skipSub=suckt_name skipInst=inst_name	If one or both options are used, subcircuits matching subcircuits and instances prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
vlth=vl	Specifies the low- voltage threshold such that CCK traces from the voltage sources with values less than or equal to <i>vl</i> . You can specify this option with global settings.
vt=vt0	After voltage propagation, the specified <i>vt0</i> value is used to determine if a violation is warranted based on the following equation: $\max(V_b) \geq \min(V_d/s) + <vt0>$ <p>The default value is 0.3. You can specify this option with global settings.</p>
vnth=v1	Specifies the n-MOSFET threshold voltage. The <i>v1</i> value is used during voltage propagation to determine whether a full voltage, <i>vt</i> dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v2	Specifies the p-MOSFET threshold voltage. The <i>v2</i> value is used during voltage propagation to determine whether a full voltage, <i>vt</i> dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings

Appendix I: HSIM CCK

Finding Potentially Conducting NMOS Devices

Argument	Description
rptv=node_name	Reports the voltage ranges of the specified nodes. You can use wildcard characters in the node name.
rptTrace=[0 1 2]	Indicates whether (1) or not (0) CCK reports a conductive path for a transistor that leads from its nodes to a voltage source. If you set a value of 2, CCK also indicates the conductive path information as in 1, but also reports the additional vt-drop information in the violation report. The default value is 0. You can specify this option with global settings.
subinfo=[0 1]	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
filterAlert=[0 1]	indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.
pwl_time=time	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (v(time)) used for propagation is defined by the pwl voltage source. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
fvsrc='(e_name,vmin,vmax)' x)	Propagates from voltage source element <i>e_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
fvsrcnd='(vsr_node_name e,vmin,vmax)' x)	Propagates from voltage source node <i>vsr_node_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
connSub=subckt_name	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified subcircuit name.
connInst=inst_name	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.

Argument	Description
connNode=node_name	Limits elements to be reported to those that have a direct connection to the specified node name.
autoFvsrcnd=[0 1]	If set to 1, the static CCK command automatically looks for any pwl or pulse supply voltage in the netlist, scans the supply waveform, and identifies its minimum and maximum peak value (Vmin, Vmax), respectively. The identified values form a voltage range (Vmin, Vmax) and are carried throughout the static voltage propagation process. The default is 0, which means that the automation capability is disabled. In addition, if autoFvsrcnd=1 is applied together with fvsrcnd() with a designated voltage range on a particular Vsrc node (means the user-defined voltage range is different than (Vmin, Vmax) identified by autoFvsrcnd=1), the particular Vsrc node propagates twice, once with the (Vmin, Vmax) by autoFvsrcnd=1, and the other by the user-defined range by fvsrcnd().
skipPcap=0 1	If you set this option to 1, the CCK commands skip the violation reporting on related MOSFETs with their drain and source shorted. The default is 0.
scope='(subckt_name, inst_name, model_name)'	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
skipScope='(subckt_name, inst_name, model_name)'	Specifies a filtering control to prevents voltage propagation from occurring through devices or nodes.
rmScope='(subckt_name, inst_name, model_name)'	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Description

cckNmosG_gt_DS performs analysis checks on n-MOSFET transistors specifically to see if the maximum bulk voltage (maxVb) is greater than the minimum drain/source voltages (minVd/s). Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared. If $\text{maxVb} \geq \text{minVd/s} + \text{vt}$, warnings are generated. The num option is used to limit the number of warnings that are reported. See [Figure 103](#).

Appendix I: HSIM CCK

Finding Potentially Conducting NMOS Devices

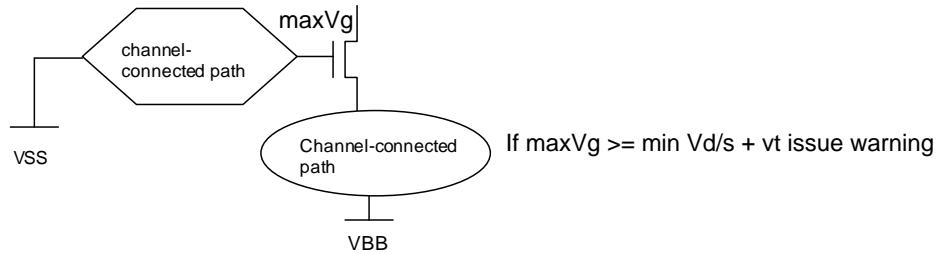


Figure 103 Report Warning Circuit

Examples

```
cckNmosG_gt_DS vith=0.4 vt=0.3 inst=xam*
```

The following example shows the .cck file output sample resulting from the command in the previous example:

```
*****
* NMOS gate vsrc value greater than D/S vsrc: inst=xam*
* vith=0.40    vt=0.30
* format: instName gate_volt d/s_volt
*****
xam.xctl.mu30      Vg (0.6) Vs (0)
xam.xu22.mu2       Vg (0.6) Vs (0)
xam.xi14.mu30      Vg (0.6) Vs (0)
```

The following example assumes this voltage source definition in the netlist:

```
...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vddx      0 pvdd
vvde vvdex     0 pwl 0ns 0 0.1ns pvdd
vshift shift    0 pwl 0ns 0 0.1ns pvdd2
...
```

The following example checks the terminal voltage of capacitor subckt, cap1. If the voltage difference ($V_{\text{positive}} - V_{\text{negative}}$) is outside the range (-2.0, 5.51) with the time period longer than 1 nanosecond, a violation is reported.

```
cckSoa label="test_cap"    subckt=cap1 constraint='(vdip, -2.0, 5.51, 1)'
```

And also assumes the following command set:

```
cckNmosG_gt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrncnd=1
```

CCK operates equivalently to the following syntax:

```
ckNmosG_gt_DS vhth=0.5 vnth=0.3 vt=0.1 fvsrrnd=(vvdex, 0, 1.11) fvsrrnd=(shift,
0, 0.8)
```

The report header is updated to include the autoFvsrnd setting:

```
*****
* PMOS gate vsrc valu
e less than D/S vsrc:
*
* vnth=0.300 vpth=-0.500
* vhth=0.500
*
* vt=0.10 num=300
*
* autoFvsrnd=1
* format: instName gate_volt d/s_volt
*****
```

Checking NMOS Node to VDD Connection

You can use the cckNmosNodeToVdd command to check NMOS terminals connected to a logic-high voltage source.

cckNmosNodeToVDD

Checks NMOS terminals (drain/source/gate/bulk). If any terminal is connected to logic-high voltage source (whose value is larger than vhth), CCK reports it. You can specify multiple cckNmosNodeToVdd commands in a CCK command file.

Note: The second syntax has a simplified version of the node specification.

Syntax

```
cckNmosNodeToVdd <tag=tag_name> <num=n> <vhth=v>
    <inst=inst_name> <model=model_name> <node=drain>
    <node=source> <node=gate> <node=bulk>
    <vsrcnd=vsrc_node_name>

or

cckNmosNodeToVdd <tag=tag_name> <num=n> <vhth=v>
    <inst=inst_name> <model=model_name>
    <node='drain|source|bulk|gate'> <vsrcnd=vsrc_node_name>
```

Appendix I: HSIM CCK

Checking NMOS Node to VDD Connection

Argument	Description
tag=tag_name	Prints the specified tag in the report file at the beginning of a line to indicate a warning.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
vhth=v	Specifies the threshold voltage to check. The default is 0.7 V.
inst=inst_name	Specifies the instance to check.
model=model_name	Specifies the model to check.
<node=drain> <node=source> <node=gate> <node=bulk> node='drain source bulk gate	Specifies the nodes to check. The "*" and "all" keywords are supported for checking on four terminals of a MOS.
vsrcnd=vsrc_node_name	Specifies the node name of the voltage source and conducts the check from these voltage sources. By default, all voltage sources are added as start nodes. This option supports wildcard characters.

Examples

The following examples are equivalent. They check the node voltage inside instance, x03 with model=nmos. If the voltages of gate node and drain node are higher than 0.7 V, they are reported

```
cckNmosNodeToVdd tag=tag2 vhth=0.7 node=gate node=drain inst=x03.* model=nmos
cckNmosNodeToVdd tag=tag2 vhth=0.7 node='gate| drain' inst=x03.* model=nmos
```

The following example checks the four terminals of the NMOS device in a design because node=* is specified.

```
cckNmosNodeToVdd tag=tag4 vhth=0.7 node=*
```

The following example checks the four terminals of the NMOS device inside the x03 instance with model name, nmos, because the keyword, all, is used in the node= control parameter.

```
cckNmosNodeToVdd tag=tag5 vhth=0.7 node=all inst=x03.* model=nmos
```

Checking Node Voltages

You use the `cckNodeVoltage` command to check if nodes exceed specified limits.

`cckNodeVoltage`

Checks if the voltage of a node exceeds specified limits.

Syntax

```
cckNodeVoltage <num=n> <vmax=v1> <vmin=v2> <start=t1>
    <stop=t2> <model=m> <tag=t> <lvdb=v> <uvdb=v> <lvbd=v>
    <uvbd=v> <lvds=v> <uvds=v> <lvsd=v> <uvsd=v> <lvdg=v>
    <uvdg=v> <lvgd=v> <uvgd=v> <lvgs=v> <uvgs=v> <lvsg=v>
    <uvsg=v> <lvbs=v> <uvbs=v> <lvcs=v> <uvcs=v> <lvsc=v>
    <uvsc=v> <lves=v> <uves=v> <lvse=v> <uvse=v> <lvbc=v>
    <uvbc=v> <lvcb=v> <uvcb=v> <lvbe=v> <uvbe=v> <lveb=v>
    <uveb=v> <lvce=v> <uvce=v> <lvec=v> <uvce=v> <lvsb=v>
    <uvsb=v> <lvgb=v> <uvgb=v> <lvbg=v> <uvbg=v> <lvac=v>
    <uvac=v> <node=name>
```

Description

In a transient simulation, CCK monitors node voltage. If any node voltage is greater than the maximum value (`v1`), or smaller than the minimum value (`v2`), then CCK reports the following:

- Node name
- Voltage
- Time

`num=m`: If the number of warnings exceeds a predetermined threshold (`num=n`), CCK stops printing. The time window to monitor the node voltage can also be specified.

`model=m`: The type of devices used to check node voltage can be selected. The following three model types are supported:

- MOSFET
- BJT
- Diode

Appendix I: HSIM CCK
Checking Node Voltages

`cckNodeVoltage` compares the voltage of a pair of terminals to user-defined thresholds. For example, MOSFET elements have four terminals:

- Drain
- Source
- Gate
- Bulk

For each of these four MOSFET nodes, the following bounds can be applied:

- `lvdb`: Lower bound of drain-to-bulk voltage difference.
- `uvdb`: Upper bound of drain-to-bulk voltage difference.
- `lvgs`: Lower bound of gate-to-source voltage difference.
- `uvgs`: Upper bound of gate-to-source voltage difference.

Examples

The following example represents a sample device used to check node voltage:

```
cckNodeVoltage vmax=10 vmin=-3 model=nch
```

In the following example, starting from 500 ns the gate-to-source voltage difference of `ph1a` MOSFET is checked. If it is smaller than 0.5 V, it is reported and output tag `ph1a` is placed at the beginning of the line. The drain-to-bulk voltage difference is also checked to see if it is greater than 1.3 V.

If the number of warnings exceeds a predetermined threshold (`num=n`), CCK stops printing. The time window to monitor the node voltage can also be specified.

```
cckNodeVoltage model=ph1a tag=ph1a lvgs=0.5 uvdb=1.3 start=500n
```

In the following example, starting from 100ns until end of simulation, CCK monitors node voltage. If the voltage is > 5 volt, or < -2.3 volt, a warning is issued.

```
cckNodeVoltage num=100 vmax=5 vmin=-2.3 start=100n
```

The following is the MOSFET bulk/diode forward bias and node voltage check (from `hsim.cck` or `out_file.cck`) output sample resulting from the command example above.

```
***** ****
* MOS-Bulk/Diode Forward Bias and Node Voltage Check
* node voltage: vmax=5, vmin=-2.3
***** ****
@20.4n, node (xx1.pg.a1) voltage (6.71) exceeds vmax
@100.7n, node (x1.asrc.p1) voltage (-3.4) is below vmin
```

In the following example, nodes in different areas have different voltage ranges. Node names are allowed in the command. CCK examines those specific nodes to see if their voltages meet the constraint. Otherwise, a warning is created. For example, nodes in `x1` and `x2` instances are limited to be within -1 V and 3 V. The nodes in `x4` instance need to be within -2 V to 4 V.

```
cckNodeVoltage    vmax=3      vmin=-1  node=x1*  node=x2*
cckNodeVoltage    vmax=4      vmin=-2  node=x4*
```

Checking Paths to Voltage Sources

You can use the `cckPathToVsrc` command to check for a node path to reach VDD, GND, both, or specified nodes.

cckPathToVsrc

Checks whether a node has a path to reach VDD, GND, both, or specified nodes, and reports if no such path is found.

Syntax

```
cckPathToVsrc <fanout=0|1> <num=n> <node=node_name>
    <bjton=0|1> <bjtonc2e=0|1> <bjtonb2e=0|1> <bjtonb2c=0|1>
    <scope='(subckt_name, inst_name, model_name)'>
    <skipScope='(subckt_name, inst_name, model_name)'>
    <rmScope='(subckt_name, inst_name, model_name)'>
```

or

```
cckPathToVsrc <fanout=0|1]> <num=n> <node=node_name>
    <ckt_node=internal_node_name>
    <vsrc_node=vsrc_node_name> <bjton=0|1> <bjtonc2e=0|1>
    <bjtonb2e=0|1> <bjtonb2c=0|1>
```

Argument	Description
<code>fanout=0 1</code>	To only check nodes that have a direct connection to the transistor gate, set <code>fanout=1</code> . To check all nodes, set <code>fanout=0</code> . The default is 0.

Appendix I: HSIM CCK

Checking Paths to Voltage Sources

Argument	Description
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
node=node_name	Reports only the specified matching nodes.
vsrc_node=vsrc_node_name	Specifies the names of the voltage source nodes. <i>cckPathToVsrc</i> traces to find out if the specified nodes in <i>node=</i> can reach the <i>vsrc_node</i> . The <i>vsrc_node</i> must be a voltage source. If non-voltage source nodes are chosen, warnings are issued in the log file.
bjton=0 1	Includes all BJT devices in the checking. The default is 0, which excludes all BJT devices.
bjtonc2e=0 1	Includes collector to emitter paths for all BJT devices. The default is 0.
bjtonb2e=0 1	Includes base to emitter paths for all BJT devices. The default is 0.
bjtonb2c=0 1	Includes base to collector paths for all BJT devices in the checking. The default is 0.
scope='(subckt_name, inst_name, model_name)'	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
skipScope='(subckt_name, inst_name, model_name)'	Specifies a filtering control to prevents voltage propagation from occurring through devices or nodes.
rmScope='(subckt_name, inst_name, model_name)'	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Description

cckPathToVsrc checks whether a node has a path to reach VDD, GND, both, or specified nodes, and reports if no such path is found.

There are two uses for *cckPathToVsrc*, and the reporting styles are slightly different in both. When you use the first syntax, nodes are reported if they cannot reach a constant voltage source, ground, or both. The report has three categories:

- No path to any generic voltage sources or ground nodes
- No path to any generic voltage sources
- No path to any ground nodes

The generic voltage source refers to all of the possible voltage sources in the design or netlist, regardless of the names and the voltage level of the voltage sources.

The second syntax is more advanced. When you use this version, it checks if a node has a path to voltage source, ground source, or specific nodes. The report contains two categories:

- No path to the `vsrc_node specified_vsrc_node_name`
- No path to the circuit node `specified_ckt_node_name`

The `specified_vsrc_node_name` and `specified_ckt_node_name` parameters are the node names specified in the control parameters, `vsrc_node=` and `ckt_node=` respectively. The second syntax is more beneficial in a design with multiple voltage sources. For example, when a design has multiple voltage sources (VDD1, VDD2, etc.), the particular VSRC nodes can be reported in the banner if there are violations.

If a node has a resistive path to the current source, it is not reported. The total number of nodes cannot exceed 30.

Examples

In the following example, all nodes with fanout are checked to see if constant voltage sources can be reached. The number of warnings for each category cannot exceed 300.

```
cckPathToVsrc num=300 node=*
```

The following is the check path to voltage sources (from `hsim.cck` or `out_file.cck`) output sample resulting from the previous command example.

Appendix I: HSIM CCK

Checking Paths to Voltage Sources

```
*****
* Check Path to Voltage Sources
*   fanout = 0
*   num     = 300
*   node    = *
*****
No path to any generic voltage sources or ground node(s) :
  node (to_inv3) has no path to any voltage sources or ground node(s)

No path to any generic voltage sources:
  node (low) has no path to any generic voltage sources

No path to ground node(s):
  node (high) has no path to ground node(s)
~
```

~

Note: The report style of version (1) and version (2) are slightly different. See the variations in the following report examples.

When the command in the example above is changed to the following:

```
cckPathToVsrc node=* vsrc_node=vdd vsrc_node=vss ckt_node=to_inv3
```

The report looks like this:

```
No path to circuit node 'to_inv3':
  node (high) has no path to to_inv3
  node (low) has no path to to_inv3
  node (out) has no path to to_inv3
  node (to_inv3) has no path to to_inv3

No path to specified vsrc_node 'vss':
  node (high) has no path to vss
  node (to_inv3) has no path to vss

No path to specified vsrc_node 'vdd':
  node (low) has no path to vdd
  node (to_inv3) has no path to vdd
```

In the third example, all transistor nodes including BJTs with `fanout=1` are checked to determine if constant voltage sources can be reached. The number of warnings for each category does not exceed 10000.

```
cckPathToVsrc fanout=1 node=* num=10000 BJTON=1
```

The check path to voltage sources (from `hsim.cck` or `out_file.cck`) output sample resulting from the previous command example is as follows:

```
*****
* Check Path to Voltage Sources
*   fanout      = 1
*   num         = 10000
*   bjton       = 1
*   bjtonb2c    = 1
*   bjtonb2e    = 1
*   bjtonc2e    = 1
*   node        = *
*****
No path to any generic voltage sources or ground node(s) :
node(xids3234a1.xosc.biassource.xosc_biasres.net26) has no path to
any voltage sources or ground node(s)
node(xids3234a1.xosc.biassource.xosc_biasres.net28) has no path to
any voltage sources or ground node(s)
node (xids3234a1.xpads_east.dinr) has no path to any voltage sources or ground
node(s)
~
~
```

Note: The report includes BJTs on in path checking. You can apply any of the combinations of `bjtonc2e/bjtonb2e/bjtonb2c=0` in this example. However, CCK issues a warning when the combination of `bjtonc2e/bjtonb2e/bjtonb2c=1` is used with `bjton=0`.

Checking PMOS Bulk Connections

You can use the `cckPmosB_It_DS` command to check p-MOSFET transistors specifically to see if the minimum bulk voltage (`minVb`) is less than the maximum drain/source voltages (`maxVd/s`).

cckPmosB_It_DS

Performs analysis checks on p-MOSFET transistors specifically to see if the minimum bulk voltage (`minVb`) is less than the maximum drain/source voltages (`maxVd/s`).

Syntax

```
cckPmosB_lt_DS <model=model_name> <subckt=subckt_name>
  <inst=inst_name> <rmSub=subckt_name>
  <rmInst=inst_name> <skipSub=suckt_name>
  <skipInst=inst_name> <vhth=vh> <vt=vt0>
  <vnth=v1> <vpth=v2> <num=n> <rptv=node_name>
  <rptTrace=[0|1|2]> <subinfo=[0|1]>
```

Appendix I: HSIM CCK

Checking PMOS Bulk Connections

```
<filterAlert=[0|1]> <pwl_time=time>
<fvsrc='(e_name,vmin,vmax)'>
<fvsrcnd='(vsrc_node_name,vmin,vmax)'>
<connSub=subckt_name> <connInst=inst_name>
<connNode=node_name> <autoFvsrcnd=[0|1]>
<skipPcap=0|1>
```

Argument	Description
model=model_name	Causes CCK to go through all instances of <i>model_name</i> during path tracing and does not consider other model devices. The model option only applies to the adjacent subcircuit/instance options behind it, and non-scoping option is not allowed in between. If no adjacent subcircuit/instance option is found behind the model option, an implicit <i>inst=*</i> is added.
subckt=subckt_name inst=inst_name	Examines all instances of <i>inst_name</i> in all of the instances of the subcircuit defined by <i>subckt_name</i> . Instance names can contain wildcards; the subcircuit name cannot.
rmSub=subckt_name rmInst=inst_name	If one or both of these options is used, elements that satisfy the defined conditions are not checked. This option prevents devices that reside in the specified subcircuit or instance from being reported. Both options allow the use of wildcards.
skipSub=suckt_name skipInst=inst_name	If one or both options are used, subcircuits matching subcircuits and instances prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
vhth=vh	Specifies the high-voltage threshold such that CCK traces from the voltage sources with values greater than or equal to <i>v1</i> . The default is 0.7. You can specify this option with global settings.
vt=vt0	After voltage propagation, the specified <i>vt0</i> value is used to determine if a violation is warranted based on the following equation: $\max(V_b) \geq \min(V_d/s) + <vt0>$ The default value is 0.3. You can specify this option with global settings.
vnth=v1	Specifies the n-MOSFET threshold voltage. The <i>v1</i> value is used during voltage propagation to determine whether a full voltage, <i>vt</i> dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v2	Specifies the p-MOSFET threshold voltage. The <i>v2</i> value is used during voltage propagation to determine whether a full voltage, <i>vt</i> dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.

Argument	Description
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings
rptv=node_name	Reports the voltage ranges of the specified nodes. You can use wildcard characters in the node name.
rptTrace=[0 1 2]	Indicates whether (1) or not (0) CCK reports a conductive path for a transistor that leads from its nodes to a voltage source. If you set a value of 2, CCK also indicates the conductive path information as in 1, but also reports the additional vt-drop information in the violation report. The default value is 0. You can specify this option with global settings.
subinfo=[0 1]	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
filterAlert=[0 1]	Indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.
pwl_time=time	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (v(time)) used for propagation is defined by the pwl voltage source. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
fvsrc='(e_name,vmin,vmax)'	Propagates from voltage source element <i>e_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
fvsrcnd='(vsr_node_name,e,vmin,vmax)'	Propagates from voltage source node <i>vsr_node_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
connSub=subckt_name	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified subcircuit name.

Appendix I: HSIM CCK

Checking PMOS Bulk Connections

Argument	Description
connInst=inst_name	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.
connNode=node_name	Limits elements to be reported to those that have a direct connection to the specified node name.
autoFvsrcnd=[0 1]	If set to 1, the static CCK command automatically looks for any pwl or pulse supply voltage in the netlist, scans the supply waveform, and identifies its minimum and maximum peak value (Vmin, Vmax), respectively. The identified values form a voltage range (Vmin, Vmax) and are carried throughout the static voltage propagation process. The default is 0, which means that the automation capability is disabled. In addition, if autoFvsrcnd=1 is applied together with fvsrcnd=() with a designated voltage range on a particular Vsrc node (means the user-defined voltage range is different than (Vmin, Vmax) identified by autoFvsrcnd=1), the particular Vsrc node propagates twice, once with the (Vmin, Vmax) by autoFvsrcnd=1, and the other by the user-defined range by fvsrcnd=().
skipPcap=0 1	If you set this option to 1 the CCK commands skip the violation reporting on related MOSFETs with their drain and source shorted. The default is 0.

Description

Based on the rules defined by this command, voltage propagation is performed and the resulting voltages are compared. If `minVb <= maxVd/s - vt`, warnings are generated.

Note: Among the connSub, connInst, and connNode arguments, if you apply more than one or mix arguments there is no scoping relationship and CCK operates based on a pure OR Boolean operation in reporting the violations collected through each of the specified arguments. These three arguments can use wildcard characters.

Examples

The following example assumes this Vsrc definition in the netlist:

```
...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vwdx      0 pvdd
vvde vvdex     0 pwl 0ns 0 0.1ns pvdd
vshift shift    0 pwl 0ns 0 0.1ns pvdd2
...
```

And also assumes the following command set

```
cckPmosB_lt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrrnd=1
```

CCK operates equivalently to following syntax:

```
cckPmosB_lt_DS vhth=0.5 vnth=0.3 vt=0.1 fvsrrnd=(vvdex, 0, 1.11) fvsrrnd=(shift, 0, 0.8)
```

The report header is updated to include the `autoFvsrrnd` setting:

```
*****
* PMOS gate vsrc value less than D/S vsrc:
*
*   vnth=0.300 vpth=-0.500
*   vhth=0.500
*
*   vt=0.10 num=300
*
*   autoFvsrrnd=1
*   format: instName gate_volt d/s_volt
*****
```

Finding Potentially Conducting PMOS Devices

You can use the `cckPmosG_lt_DS` command to examine every p-MOSFET channel-connected path from drain/source and gate.

cckPmosG_lt_DS

Examines every p-MOSFET channel-connected path from drain/source and gate.

Syntax

```
cckPmosG_lt_DS <model=model_name> <subckt=subckt_name>
<inst=inst_name> <rmSub=subckt_name>
<rmInst=inst_name> <skipSub=suckt_name>
<skipInst=inst_name> <vhth=vh> <vt=vt0>
<vnth=v1> <vpth=v2> <num=n> <rptv=node_name>
<rptTrace=[0|1|2]> <subinfo=[0|1]>
<filterAlert=[0|1]> <pwl_time=time>
<fvsrc='(e_name,vmin,vmax)'>
<fvsrcnd='(vsrc_node_name,vmin,vmax)'>
```

Appendix I: HSIM CCK

Finding Potentially Conducting PMOS Devices

```
<connSub=subckt_name> <connInst=inst_name>
<connNode=node_name> <autoFvsrCnd=[0|1]>
<skipPcap=0|1>
```

Argument	Description
model=model_name	Causes CCK to go through all instances of <i>model_name</i> during path tracing and does not consider other model devices. The model option only applies to the adjacent subcircuit/instance options behind it, and non-scoping option is not allowed in between. If no adjacent subcircuit/instance option is found behind the model option, an implicit <i>inst=*</i> is added.
subckt=subckt_name inst=inst_name	Examines all instances of <i>inst_name</i> in all of the instances of the subcircuit defined by <i>subckt_name</i> . Instance names can contain wildcards; the subcircuit name cannot.
rmSub=subckt_name rmInst=inst_name	If one or both of these options is used, elements that satisfy the defined conditions are not checked. This option prevents devices that reside in the specified subcircuit or instance from being reported. Both options allow the use of wildcards.
skipSub=subckt_name skipInst=inst_name	If one or both options are used, subcircuits matching subcircuits and instances prevent voltage propagation from occurring through elements within their coverage. Both options allow the use of wildcards.
vhth=vh	Specifies the high-voltage threshold such that CCK traces from the voltage sources with values greater than or equal to <i>v1</i> . The default is 0.7. You can specify this option with global settings.
vt=vt0	After voltage propagation the specified <i>vt0</i> value is used to determine if a violation is warranted based on the following equation: $\max(V_b) \geq \min(V_d/s) + <vt0>$ The default value is 0.3. You can specify this option with global settings.
vnth=v1	Specifies the n-MOSFET threshold voltage. The <i>v1</i> value is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across an n-MOSFET transistor channel. The default value is 0.5. You can specify this option with global settings.
vpth=v2	Specifies the p-MOSFET threshold voltage. The <i>v2</i> value is used during voltage propagation to determine whether a full voltage, vt dropped voltage, or no voltage is passed across a p-MOSFET transistor channel. The default value is -0.5. You can specify this option with global settings.

Argument	Description
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings
rptv=node_name	Reports the voltage ranges of the specified nodes. You can use wildcard characters in the node name.
rptTrace=[0 1 2]	Indicates whether (1) or not (0) CCK reports a conductive path for a transistor that leads from its nodes to a voltage source. If you set a value of 2, CCK also indicates the conductive path information as in 1, but also reports the additional vt-drop information in the violation report. The default value is 0. You can specify this option with global settings.
subinfo=[0 1]	Indicates whether (1) or not (0) subcircuit information about the violation node path is included in the warnings. The default value is 0. You can specify this option with global settings.
filterAlert=[0 1]	Indicates whether (1) or not (0) CCK produces an error message and the entire process is terminated when the scope of a given command, defined by the model, subcircuit and/or instance parameters, is found to be empty. The default is 1. You can specify this option with global settings.
pwl_time=time	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (v(time)) used for propagation is defined by the pwl voltage source. This voltage value (v(time)) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
fvsrc='(e_name,vmin,vmax)'	Propagates from voltage source element <i>e_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
fvsrcnd='(vsr_node_name,e,vmin,vmax)'	Propagates from voltage source node <i>vsr_node_name</i> with values <i>vmin</i> and <i>vmax</i> . If <i>vhth</i> is set, constant voltage sources greater than or equal to the value set for <i>vhth</i> are used as a starting point for propagation. If <i>vlth</i> is set, constant voltage sources less than or equal to the value set for <i>vlth</i> are used as a starting point for propagation. By default, CCK only propagates from constant voltage sources.
connSub=subckt_name	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified subcircuit name.

Appendix I: HSIM CCK

Finding Potentially Conducting PMOS Devices

Argument	Description
connInst=inst_name	Limits elements to be reported to those that have a direct connection to the ports (except power/ground) inside of the specified instance name. Note that the specified instance name only applies to subcircuit instance name, and the instance name specified by this argument must use a full hierarchical naming convention.
connNode=node_name	Limits elements to be reported to those that have a direct connection to the specified node name.
autoFvsrcnd=[0 1]	If set to 1, the static CCK command automatically looks for any pwl or pulse supply voltage in the netlist, scans the supply waveform, and identifies its minimum and maximum peak value (Vmin, Vmax), respectively. The identified values form a voltage range (Vmin, Vmax) and are carried throughout the static voltage propagation process. The default is 0, which means that the automation capability is disabled. In addition, if autoFvsrcnd=1 is applied together with fvsrcnd=() with a designated voltage range on a particular Vsrc node (means the user-defined voltage range is different than (Vmin, Vmax) identified by autoFvsrcnd=1), the particular Vsrc node propagates twice, once with the (Vmin, Vmax) by autoFvsrcnd=1, and the other by the user-defined range by fvsrcnd=().
skipPcap=0 1	If you set this option to 1 the CCK commands skip the violation reporting on related MOSFETs with their drain and source shorted. The default is 0.

Description

If there is a possible channel-connected path from source or drain to a logic-high voltage source VPP, cckPmosG_lt_DS checks the channel-connected path from gate to another logic-high voltage source VDD. If the minimum of gate voltage, minVg, is less than the maximum of D/S voltage, maxVd/s, CCK reports a warning about this p-MOSFET and the resulting VPP and VDD, because the p-MOSFET may cause a leakage path. See [VDD < \(VPP – vt\) = Report Warning](#).

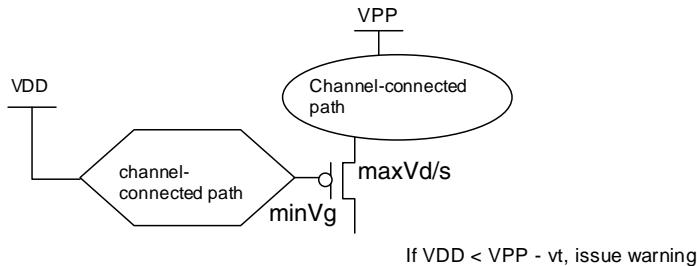


Figure 104 $VDD < (VPP - vt) = \text{Report Warning}$

The following values are used in voltage drop calculations along with path tracing:

- $v_{n\text{th}}$: n-MOSFET turn-on threshold.
- $v_{p\text{th}}$: p-MOSFET turn-on threshold.

This command allows some portions of the design to be examined by using the `subckt` and `inst` options. During the channel-connect path tracing, it allows some specific instances to be skipped by using `skipSub` and `skipInst`. In the reporting, some instances are repressed using `rmSub` and `rmInst`.

When `rptTrace=1`, two paths of transistors are included for each device to be printed. The first one is a path from a reported device gate node to a voltage source; the second list is a path from its drain or source node to lead to another power supply.

`cckPmosG_lt_DS` and related commands are used immediately after HSIM netlist parsing as described in the following:

- Operation parsing is static based and vt in the model file is not used.
- $v_{n\text{th}}$ and $v_{p\text{th}}$ in the command option are used for voltage drop consideration along the path tracing process.

Note: Among the `connSub`, `connInst`, and `connNode` arguments, if you apply more than one or mix arguments there is no scoping relationship and CCK operates based on a pure OR Boolean operation in reporting the violations collected through each of the specified arguments. These three arguments can use wildcard characters.

Appendix I: HSIM CCK

Finding Potentially Conducting PMOS Devices

Examples

The following example checks all the p-MOSFETs in instance `xcm`. It finds the channel-connected paths from drain or source to any high-voltage source whose value is larger than 0.8 V. Let the highest voltage of source node be `maxVd/s`. Then, look for the channel-connected path from gate to a positive voltage source whose value is larger than `vhth=0.8v`. Let the lowest voltage source value seen from gate be `minVg`. If `maxVg` is smaller than `minVd/s` by 0.3 V, issue a warning.

```
cckPmosG_lt_DS vhth=0.8 inst=xcm* vt=0.3 vnth=0.6 vpth=-0.5
```

An output sample resulting from the previous command example is as follows:

```
*****
* PMOS gate vsrc value less than D/S vsrc: inst=xcm
* vhth=0.80      vt=0.30
* format: instName gate_volt d/s_volt
*****
xcm.xdec.xu1.mp1          Vg (1) Vs (1.65)
xcm.xdec.xu6.mp2          Vg (1) Vs (1.65)
xcm.xctl.mu2              Vg (1) Vs (1.65)
```

In the following example, all of the p-MOSFET in subcircuit `cpu` and the p-MOSFET whose names starting with `x1` and `x2` in `fifo` subcircuit are checked. The command traces to all the logic-high voltage sources, whose values are larger than 0.8 V. During the traversing, it skips all the transistors in the `mux2` subcircuit.

```
cckPmosG_lt_DS vhth=0.8 vt=0.3 vnth=0.6 vpth=-0.5 subckt=cpu inst=* subckt=fifo
inst=x1* inst=x2* skipSub=mux2 skipInst=* rmInst=x4*
```

When the tracing is completed, it compares the gate node voltage with those of drain and source. If the gate voltage is smaller than drain or source by 0.3 V, it reports this device, unless this device's name is prefixed with `x4`.

The following example uses the `rptTrace` option:

```
cckPmosG_lt_DS subckt=aa inst=* vhth=0.7 vnth=0.5 vpth=-0.6 vt=0.3 rptTrace=1
```

For each transistor reported, a transistor path is printed which leads its gate node to a voltage source.

A sample of the above command example's output is:

```

xi.mp1      Vg (2.4) Vs (2.8)
(G) wctrl
    thruI xi10.m2 (propagate 2.400)
    thruI xi10.r3 (propagate 2.400)
    thruI xial.mp1 (propagate 2.400)
    frmNd vdd
(S) wa
    thruI xi9.m1 (propagate 2.8)
    frmNd exvdd

```

The gate node of transistor `xi.mp1` has a potential voltage of 2.4 volt, which is less than the source node's potential voltage (2.8 volt). So this device can be partially conducting and cause large leakage current. To facilitate debugging the design, a path is shown connecting its gate node to 2.4 volt; a path connecting its source node to 2.8 volt. In this output, it shows that the gate node `wctrl` can reach VDD of 2.4 volt through `xi10.m2`, `xi10.r3`, and `xial.mp1`. How voltage is propagated is also shown. If there is a voltage drop, it is printed. For its source node `wa`, it is connected to `exvdd` which is at 2.8 volt through `xi9.m1`. By going through these two lists, the cause for potential large leakage current can be found.

The following example assumes this `Vsrc` definition in the netlist:

```

...
.param pvdd = 1.11v
.param pvdd2 = 0.8v
vvdd vddx      0 pvdd
vvde vvdex      0 pwl 0ns 0 0.1ns pvdd
vshift shift     0 pwl 0ns 0 0.1ns pvdd2
...

```

And also assumes the following command set:

```
cckPmosG_lt_DS vhth=0.5 vnth=0.3 vt=0.1 autoFvsrclnd=1
```

CCK operates equivalently to following syntax:

```
cckPmosG_lt_DS vhth=0.5 vnth=0.3 vt=0.1 fvsrclnd=(vvdex, 0, 1.11) fvsrclnd=(shift, 0, 0.8)
```

The report header is updated to include the `autoFvsrclnd` setting:

```
*****
* PMOS gate vsrc valu
e less than D/S vsrc:
*
*   vnth=0.300 vpth=-0.500
*   vhth=0.500
*
*   vt=0.10 num=300
*
*   autoFvsrclnd=1
*   format: instName gate_volt d/s_volt
*****
```

Appendix I: HSIM CCK

Checking PMOS Node To GND Connection

Checking PMOS Node To GND Connection

You use the `cckPmosNodeToGnd` command to check the four p-MOSFET nodes (drain/source/gate/bulk).

cckPmosNodeToGnd

Checks four p-MOSFET nodes (drain/source/gate/bulk) to find out if they are connected to ground or logic-low voltage source (whose value is less than the specified threshold), and if so report it.

Note: The second syntax has a simplified version of the node specification.

Syntax

```
cckPmosNodeToGnd <tag=tag_name> <num=n> <vlth=v>
    <inst=inst_name> <model=model_name> <node=drain>
    <node=source> <node=gate> <node=bulk>
    <vsrncnd=vsrnc_node_name>

or

cckPmosNodeToGnd <tag=tag_name> <num=n> <vlth=v>
    <inst=inst_name> <model=model_name>
    <node='drain|source|bulk|gate'> <vsrncnd=vsrnc_node_name>
```

Argument	Description
<code>tag=tag_name</code>	Prints the specified tag in the report file at the beginning of a line to indicate a warning.
<code>num=n</code>	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <code>n</code> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
<code>vlth=v</code>	Specifies the threshold voltage to check. The default is 0.4 V.
<code>inst=inst_name</code>	Specifies the instance to check.
<code>model=model_name</code>	Specifies the model to check.

Argument	Description
<node=drain> <node=source> <node=gate> <node=bulk> node=' drain source bulk gate	Specifies the nodes to check. The "*" and "all" keywords are supported for checking on four terminals of a MOS.
vsrnd=vsrd_node_name	Specifies the node name of the voltage source and conducts the check from these voltage sources. By default, all voltage sources are added as start nodes. This option supports wildcard characters.

Examples

The following examples are equivalent. They check the node voltage inside instance, x03 with model=pmos. If the voltages of gate node and drain node are less than 0.4 V, they are reported

```
cckPmosNodeToGnd tag=tag2 vlth=0.7 node=gate node=drain inst=x03.* model=nmos
cckPmosNodeToGnd tag=tag2 vlth=0.7 node='gate| drain' inst=x03.* model=nmos
```

The following example checks the four terminals of the PMOS device in a design because node=* is specified and reports the violations if the terminal voltage is less than 0.3 V.

```
cckPmosNodeToGnd tag=tag4 vlth=0.7 node=*
```

The following example checks the four terminals of the PMOS device inside the x03 instance with model name, nmos, because the keyword, all, is used in the node= control parameter.

```
cckPmosNodeToGnd tag=tag5 vlth=0.7 node=all inst=x03.* model=nmos
```

Running a Safe Operating Area Check

You use the `cckSOA` command to run a safe operating area (SOA) check.

cckSOA

Performs a safe operating area (SOA) check.

Syntax

```
<inst=instanceScope> <label=labelName>
  <model=typeOfModel| VA_model_name>
  <num=numberOfViolations><numw=numberOfViolationWindows>
```

Appendix I: HSIM CCK

Running a Safe Operating Area Check

```
<constraint=ConstraintExpression>
<twindow='(start_time,stop_time,step_time)'>
<filterAlert 0|1> <subckt= subckt_name> <inst= inst_name>
<autoFlush 0|1> <scope='(subckt_name, inst_name,
model_name)'> <skipScope='(subckt_name, inst_name,
model_name)'> <rmScope='(subckt_name, inst_name,
model_name)'> <instparam='(width_length_expression)'>
```

Argument	Description
inst=instanceScope	Specifies the scope of devices in the circuit within which the cckSOA check is applied. See the Example to know more.
label=labelName	Specifies a label/tag to be used in the report file for distinguishing each different cckSOA command. See the Example to know more.
model=typeOfModel VA_mo del_name	<p>Specifies the devices such that the check is applied on the devices with the same model as specified in this option.</p> <p>You need not specify <code>inst=*</code> explicitly, as the <code>model</code> option implies it. See the Example to know more.</p> <p>You can also specify a Verilog-A module name, which means cckSOA performs a port voltage check on the instances instantiated from the matched Verilog-A module.</p>
num=numberOfViolations	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <code>n</code> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings
numw=numberOfViolationW indows	Specifies the maximum number of violation windows to report for a constraint. The default is 1. If <code>numw</code> is less than or equal to 0, there is no limit to the number of violation windows. Note that when you specify the <code>twindow</code> option with <code>step_time</code> , the <code>numw</code> has no effect.
constraint=ConstraintEx pression	<p>You can specify a constraint expression to perform analysis on a targeted Verilog-A module port. For example:</p> <pre>cckSOA label=test model="resVA1" constraint='(v(in1_b) < 2.0, 0, 0, 5n)'</pre> <p>In this example, cckSOA looks for voltage of <code>in1_b</code> port defined in <code>resVA1</code> Verilog-A module. If the port voltage value is less than 2 v and the condition lasts longer than 5 ns, it is reported as violation.</p> <p>Use '(and)' to delimit the constraint expression. Where <code>condition_expression</code> can be any [+ - * /] combination of cckSOA constraint functions. See the Description section for this command for the supported cckSOA constraint functions.</p>

Argument	Description
<code>twindow='(start_time,stop_time,step_time)'</code>	Specifies the start time, stop time, and step time for the SOA check. The times can be optional. For example, you can specify <code>twindow='(4n,,1n)' </code> to perform the SOA check at every 1ns starting from 4ns to the end of the simulation cycle. You can also specify <code>twindow='(1n,20n)' </code> to conduct the check starting at 1n up to 20n.
<code>filterAlert 0 1</code>	If set to 0 issues a warning if there are no matched devices. If set it to 1 (the default), issues an error message and terminates the CCK analysis if there are no matched devices.
<code>subckt= subckt_name inst= inst_name</code>	Specifies the scope of devices in the circuit within which the subcircuit is subject to the <code>cckSOA</code> check. When both <code>subckt</code> and <code>inst</code> options are applied, the scoping relationship is formed, for example: <code>cckSOA label="test" subckt=adder inst=x1.* constraint='(vgs, -1.5, 0.0, 10n)</code> This means the violated MOSFETs must be or reside in an instance with name starting with <code>x1</code> . In addition, such a qualified instance needs to be defined inside the <code>adder</code> subcircuit.
<code>autoFlush 0 1</code>	If set to 1, <code>cckSOA</code> enables the automatic report flushing capability. During the <code>cckSOA</code> analysis cycle, the analysis creates a temporary file with intermediate violation information. Because of the intermediate data dumping, the violations in the temporary file may not be sorted in desired order. In addition, the criteria in the intermediate violations might not satisfy the constraint expression (Boolean condition, time duration), and the violation state change triggers the flushing operation. The default is 0, which means automatic report flushing is disabled.
<code>scope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
<code>skipScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control to prevent voltage propagation from occurring through devices or nodes.
<code>rmScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Appendix I: HSIM CCK

Running a Safe Operating Area Check

Argument	Description
instparam='(width_length_expression)'	Specifies the MOSFET width/length filtering criteria to report qualified violated MOSFET devices. When the given <i>condition_expression</i> is found to be lower than the <i>lower_bound</i> or higher than the <i>upper_bound</i> and it lasts for a time period longer than the <i>time_duration</i> then this time window of violation is reported. If this kind of violation happens frequently and exceeds the option <i>num</i> specified, then <i>cckSOA</i> ceases to check or report more time windows of violation.

Description

Use '(and)' to delimit the constraint expression. The *condition_expression* can be any [+,-,*,/] combination of *cckSOA* constraint functions listed below. The *cckSOA* constraint functions presently supported include:

- I: get current of a DIODE
- IB: get BULK current of MOSFET or BASE current of BJT
- IC: get COLLECTOR current of BJT
- ID: get DRAIN current of MOSFET or JFET
- IE: get EMITTER current of BJT
- IG: get GATE current of MOSFET/JFET
- IS: get SOURCE current of MOSFET/JFET/BJT
- VB: get BULK/BASE voltage of MOSFET/BJT
- VBC: get BASE/COLLECTOR voltage difference of BJT
- VBD: get BULK/DRAIN voltage difference of MOSFET
- VBE: get BASE/EMITTER voltage difference of BJT
- VBS: get BULK(BASE)/SOURCE voltage difference of MOSFET(BJT)
- VC: get COLLECTOR voltage of BJT
- VCE: get COLLECTOR/EMITTER voltage difference of BJT
- VCS: get COLLECTOR/SOURCE voltage difference of BJT
- VD: get DRAIN voltage of MOSFET/JFET
- VDIP: get anode/cathode (or positive/negative) node voltage difference of Diode (or resistor/capacitor)
- VDS: get DRAIN/SOURCE voltage difference of MOSFET/JFET
- VE: get EMITTER voltage of BJT

- VES: get EMITTER/SOURCE voltage difference of BJT
- VG: get GATE voltage of MOSFET/JFET
- VGB: get GATE/BULK voltage difference of MOSFET
- VGD: get GATE/DRAIN voltage difference of MOSFET/JFET
- VGS: get GATE/SOURCE voltage difference of MOSFET/JFET
- VNEG: get CATHODE voltage of DIODE
- VPOS: get ANODE voltage of DIODE
- VS : get SOURCE voltage of MOSFET/JFET/BJT

Examples

The following example specifies that the `cckSOA` command, with and without `inst=*` gives identical results:

```
cckSOA      label="nch_vgs"      model=nch      inst=*      constraint='(vgs,-2,2,1n)'
```

and

```
cckSOA      label="nch_vgs"      model=nch      constraint='(vgs,-2,2,1n)'
```

gives the same output, therefore, `inst=*` is not needed.

The following example specifies a special feature of the `cckSOA` command, in which, the different order of the `cckSOA` parameters deliver different effective scoping range. Since the `cckSOA` command allows you to specify multiple and flexible scoping expressions, it uses `label` and `model` as internal delimiters to break up the complex scoping expression to form effective scoping range.

Example 1:

```
cckSOA ... model=m1 inst=i1 model=m2 subckt=sub2 inst=i2
```

This statement breaks down the scoping expression into two groups (Check1 and Check2) with logical “OR” relationship, and effectively the check operates based on the following scoping range:

Check1: `model=m1 && inst=i1`

Check2: `model=m2 && subckt=sub2 && inst=i2`

Therefore "Check1 || Check2" is:

`(model=m1 && inst=i1) || (model=m2 && subckt=sub2 && inst=i2)`

Example 2:

```
cckSOA ... model=m1 subckt=sub1 label="xyz" inst=i1 model=p2.....
```

Appendix I: HSIM CCK

Running a Safe Operating Area Check

This statement breaks down the scoping expression into three groups (Check1, Check 2, and Check3) with logical “OR” relationship, and effectively the check operates based on the following scoping range:

Check1: model=m1 && subckt=sub1

Check2: inst=i1

Check3: model=p2

Therefore "Check1|| Check2|| Check3" is:

(model=m1 && subckt=sub1) || (inst=i1) || (model=p2)

Example 3:

```
cckSOA    inst=*    label="nch_vgs"    model=nch    constraint='(vgs,-2,2,1n)'
```

This statement breaks down the scoping expression into two groups (Check1 and Check2) with logical "OR" relationship, and the scoping range of the check is effectively “Check1 || Check2”.

Check1 (label="nch_vgs"): inst==* && constraint='(vgs, -2,2,1n)'

Check2 (label="nch_vgs"): inst==* && model =nch && constraint='(vgs,-2,2,1n)'

But with order adjustment of the cckSOA parameters, for example:

```
cckSOA    label="nch_vgs"    model=nch    inst=*
constraint='(vgs,-2,2,1n)'
```

the scoping expression effectively equals to Check2.

The following example specifies that in addition to complying with the constraint specified by the constraint expression, the cckSOA command reports up to 100 device violations, and for each individual device it can report up to 2 violation windows (duration).

```
cckSOA label=vgd_test constraint='(vgd, 0, 1, 3n)' num=100 numw=2
```

The following commands check the device element:

x3m.x0.x0.xsap.x1.mp1 with model type p with the constraint '(vgs, -1.5, 0.0, 15n)' expression; check if the function value vgs (voltage difference between gate and source) is outside the range (-1.5, 0.0) consecutively for a time period longer than 15 nanoseconds. If such a violation occurs, it is reported. It also checks and reports occurrences of the violation up to 2 times then ceases checking. At the same time the commands also check on the same device with constraint expression '(id, -500u, 5u, 15n)' which focuses its drain current.

```
cckSOA inst=x3m.x0.x0.xsap.x1.mp1 Label="vgs" model=p num=2 \
constraint = '(vgs, -1.5, 0.0, 15n)'
cckSOA inst=x3m.x0.x0.xsap.x1.mp1 Label="id" model=p num=2 \
constraint = '(id, -500u, 5u, 15n)'
```

The following example is a violation report:

```
***-----***  
***----- vgs -----***  
Element: (x3m.x0.x0.xsap.x1.mp1) of model: 'p',  
violates constraint: '(vgs, -1.5, 0, 15)'  
@ the following time window(s):  
{  
    Time Window #1:(121.8300, 140.3690)--> time span=18.5390 ns  
    Peak constraint values: low @(-2.4094), high @(0.0000)  
}  
{  
    Time Window #2:(172.0070, 190.3690)--> time span=18.3620 ns  
    Peak constraint values: low @(-2.4094), high @(0.0000)  
}  
***-----***  
***----- id -----***  
Element: (x3m.x0.x0.xsap.x1.mp1) of model: 'p',  
violates constraint: '(id, -0.0005, 5e-006, 15)'  
@ the following time window(s):  
{  
    Time Window #1:(120.8400, 140.5300)--> time span=19.6900 ns  
    Peak constraint values: low @(0.0000), high @(0.0010)  
}  
{  
    Time Window #2:(170.8420, 190.5300)--> time span=19.6880 ns  
    Peak constraint values: low @(0.0000), high @(0.0010)  
}
```

The following example checks for MOSFET gate voltages either less than 0.5 V or greater than 1.5 V at the times between 10 ns, 11 ns, 12 ns, 13 ns, 14 ns, and 15 ns. Note that because there is a periodical check at every 1 ns between 10 ns and 15 ns, the duration of 2 ns becomes ineffective.

```
ccksoa label="test1" constraint='(vg, 0.5, 1.5, 2n)' twindow='(10n,15m,1n)'
```

Currently, the safe operating area check, cckSOA, is a dynamic check that must be performed during simulation. Sometimes, especially when the circuit is large and the circuit states change violently in simulation, the cckSOA check slows down simulation. In these circumstances, a post-simulation check capability of cckSOA can be helpful. Also, with one single set of simulation results (waveform file), you can perform a series of post-simulation checks without running simulation multiple times of simulation.

To trigger post-simulation CCK checks, it is necessary to read in waveform data to do CCK analysis. Use the `cckPost` command to enable this capability.

```
cckSOA ... constraint='(vgd, 0, 1, 3n)' instparam='(l>1u && w>=10u)'
```

Appendix I: HSIM CCK

Running a Safe Operating Area Check

Indicates that in addition to the vgd bias constraint specified by the constraint expression, the reported device needs to also qualify the geometry filtering criteria specified by instparam. In this example, the violated device needs to have its length longer than 1 u and width equal or longer than 10 u in order to be reported in the cckSOA report file.

cckPost

Performs post-simulation analysis.

Syntax

```
cckPost <dump=0 | 1 | 2> <waveform=file_name> <format=fsdb | wdf>
```

Argument	Description
<dump=0 1 2>	Specify one of the following values: <ul style="list-style-type: none">■ 0 (default) specifies that the user-defined waveform file is needed for post-simulation analysis of cckSoa checks.■ 1 generates the waveform file automatically according to the specified cckSoa commands. The waveform file contains necessary data for post-simulation analysis of related cckSoa commands. After the waveform file is generated, HSIM stops. You must run HSIM again to trigger the post-simulation cckSoa analysis using the same CCK command file used to generate waveform file, but now with dump=0.■ 2 specifies to dump the necessary waveform file exactly as with dump=1. However, HSIM now starts the post-simulation cckSoa analysis automatically after the waveform is generated.
waveform=file_name	Specifies the waveform file name.
format=fsdb wdf	Specifies either the fsdb (default) or wdf format.

Description

Due to the nature of postprocessing when applying `cckPost` in a dynamic SOA check, several control options are disabled and not supported: `skipModel`, `scope`, `skipScope`, `rmScope`, `numw`, and `autoflush`.

Evaluating Signal Degradation or Correct Biasing

You can use the `cckDynSubV` command to evaluate the signal degradation or correct biasing, such as a signal propagating along parasitic nets or a chain of diodes.

cckDynSubV

The `cckDynSubV` command lets you evaluate the signal degradation or correct biasing, such as a signal propagating along parasitic nets or a chain of diodes. It also provides the flexibility to check or monitor the voltage difference on multiple specified nodes during transient simulation.

Syntax

```
cckDynSubV tag <subckt=subckt_name> <inst=inst_name>
    constraint='expression' num=<n> duration=<val>
    start=<time> stop=<time> <separate_file=<0|1>
    <filterAlert 0|1>
```

Argument	Description
<code>tag</code>	Each <code>cckDynSubV</code> command must have a unique <code>tag</code> name as its first argument. When several <code>cckDynSubV</code> commands are in one command file, these unique tags distinguish violations reported from the different commands. In the output file, each <code>tag</code> name is the leading string of each line, so you can easily identify which line is output by which command. Also, the <code>tag</code> name is referenced as part of output filename if <code>separate_file</code> is set to 1. For example: <code>cckDynSubV sub1_chk subckt= sub1 constraint= '...' ...</code> <code>cckDynSubV inst1_chk inst= subInst1 constraint= '...' ...</code>
<code>subckt=subckt_name</code>	Specifies the subcircuit name in which the node voltages are to be checked. If a subcircuit containing the nodes to be checked is instantiated more than once, and if violations occur, these violations are all reported. This argument does not support a “*” wildcard character. If a subcircuit name matches the specified subcircuit, and the constraint expression of nodes is applicable, the expression in constraint is evaluated to determine if there is a violation.
<code>inst=inst_name</code>	Specifies the instance of a subcircuit or the instantiation of a specified subcircuit in nodes to be checked. You can specify a hierarchical name of instances delimited by a “.” character. This argument does not support a “*” wildcard. If an instance matches with this instance scope, and the constraint expression of node voltages specified is applicable to it, the expression in constraint is evaluated to determine if there is a violation.

Appendix I: HSIM CCK

Evaluating Signal Degradation or Correct Biasing

Argument	Description
constraint='expression'	An expression is formed by voltages of nodes, logical operators (<code>&&</code> , <code> </code> , <code>></code> , <code><</code> , <code>>=</code> , <code><=</code> , <code>==</code> , <code>!=</code>) and mathematical operators (<code>+</code> , <code>-</code> , <code>*</code> , <code>/</code> , and <code>abs()</code>). A simple example is: <code>constraint='((v(nodeA)-v(nodeC)) < v(nodeB) (v(nodeB) >= 0.5 && v(nodeD)<1.0))'</code> . When this expression is logically true, a violation is reported. The argument of “V” is a “node name”, where “node name” can be one of the following: <ul style="list-style-type: none"> ▪ An absolute (hierarchical) node name if you do not specify a <code>subckt</code> or <code>inst</code>. For example, <code>x0.x1.x2.x3.net4</code>. You cannot specify a wildcard in such an absolute node name expression. ▪ A (hierarchical) node name relative to <code>subckt</code> or <code>inst</code>. For example, if given <code>inst=x0.x1.x2.*</code> and there are lower-level subcircuit instances <code>x3</code> and <code>x4</code>, then <code>V(nd1)</code> means check node <code>nd1</code> of subcircuit instance <code>x0.x1.x2.x3</code> and <code>nd1</code> of subcircuit instance <code>x0.x1.x2.x4</code>. For example, if given <code>subckt=sub0</code> then <code>V(nd2)</code> means check voltage at <code>nd2</code> under the <code>sub0</code> subcircuit. ▪ When <code>inst= x0.x1.x2.*</code> is given as the pattern, it matches with subcircuit instances like <code>x0.x1.x2.x3</code> as well as <code>x0.x1.x2.x5.x6</code>. That is, the wildcard “<code>*</code>” matches <code>x3</code> and <code>x5.x6</code> as a regular expression would normally do. Note that an expression must be quoted by single quotation marks “‘‘“.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <code>n</code> is set to less than or equal to 0, the warnings are unlimited.
duration=val	Specifies the time duration that the constraint expression sustains the logically true condition to trigger an error report.
start=time stop=time	Specifies the start and stop times to perform checking.
separate_file=0 1	When set to 1, specifies that errors to be reported are directed to separate files for each tag in <code>cckDynSubV</code> command statement. The default value is 0. When <code>separate_file=1</code> , output files are named according to the following rules: <ul style="list-style-type: none"> ▪ If the <code>-o</code> option (<code>output_prefix</code>) is issued in the HSIM command line, it is used as the output file prefix. Otherwise, <code>hsim</code> is used as the default output file prefix. ▪ <code>nodev</code> is the first portion of default extension for <code>cckDynSubV</code> output files. ▪ When <code>separate_file=1</code>, an additional <code>tag</code> is attached at the end of output file name.
filterAlert 0 1	If set to 0, issues a warning if there are no matched devices. If set it to 1 (the default), issues an error message and terminates the analysis if there are no matched devices.

Examples

```
cckDynSubV test subckt=inv constraint='( (V(nd1)-V(nd2)>3) || (V(nd1)-V(nd2)<1)' duration=1n start=5n stop=10n
```

Performs a dynamic node voltage check from transient time 5 ns to 10 ns. The violations are reported if the voltage difference ($V(nd1) - V(nd2)$) is greater than 3 V or less than 1 V and such conditions remains longer than 1 ns. The target nodes `nd1` and `nd2` are defined inside subckt `inv`.

```
cckDynSubV test1 inst=hsio* constraint='(V(nd1) >=0 && V(nd2)-V(nd3) > 0.5)' duration=2n start=5n stop=10n num=200 separate_file=1
```

Performs a dynamic node voltage check from transient time 5 ns to 10 ns. A violation occurs if the constraint expression: “ $V(nd) \geq 0V$ AND $V(nd2)-V(nd3) < 0.5$ ” is satisfied. Only subckt instances with names starting with “`hsio`” are checked and reported. The total number of violations to be reported is limited to 200. The violations are in a separate file with extension `nodev_test1`.

```
cckDynSubV test2 subckt=fulladd inst=xful* constraint='(v(o1)> 0 && v(c1)-v(c2) > 0.2)' duration=3n num=100 separate_file=1
```

Performs a dynamic voltage check throughout the entire transient simulation cycle. The expression looks for signal names such as `o1`, `c1`, and `c2` defined within subcircuit `fulladd`. The checking targets are on subcircuit `fulladd` instantiation with its instance name starting with `xful`.

The following example shows a `cckDynSubV` report:

Appendix I: HSIM CCK

Checking Substrate Forward Bias

```
* -----
* Dynamic Node Voltage Check
* tag      =test2
* subckt   =fulladd
* inst     =xful*
* constraint= '(v(o1)> 0 && v(c1)-v(c2) > 0.2)'
* duration =3n
* num      = 100
* separate_file = 1
* twindow   = (0, max)

* -----
Instance: xful2
Violation Data:
Duration: (41.000, 51.961) --> time span = 10.961 ns
at 41.000 ns: xful2.o1(v=4.95532) xful2.c1(v=2.4666) xful2.c2(v=2.2018)
at 51.961 ns: xful2.o1(v=0.861179) xful2.c1(v=5.12874)
xful2.c2(v=4.92276)

Instance: xful3
Violation Data:
Duration: (42.395, 52.203) --> time span = 9.808 ns
at 42.395 ns: xful3.o1(v=4.05138) xful3.c1(v=4.61114) xful3.c2(v=4.37931)
at 52.203 ns: xful3.o1(v=0.801633) xful3.c1(v=5.0937) xful3.c2(v=4.87613)

Instance: xful2
Violation Data:
Duration: (80.967, 91.961) --> time span = 10.994 ns
at 80.967 ns: xful2.o1(v=4.98497) xful2.c1(v=2.40586) xful2.c2(v=2.17932)
at 91.961 ns: xful2.o1(v=0.864535) xful2.c1(v=5.12876) xful2.c2(v=4.9235)

** Total number of violations = 3
```

Checking Substrate Forward Bias

You can use the `cckSubstrate` command to check whether a MOSFET substrate becomes forward-biased.

cckSubstrate

Checks whether a MOSFET substrate becomes forward-biased.

Syntax

```
cckSubstrate <mode=2|1|0> <num=n> <vt=v> <ith=iv> <start=t1>
<stop=t2> <model=m> <tag=t> <pwl_time=time> <xhz=0|1>
<scope='(subckt_name, inst_name, model_name)'>
<skipScope='(subckt_name, inst_name, model_name)'>
<rmScope='(subckt_name, inst_name, model_name)'>
```

Argument	Description
mode=2 1 0	<p>Controls when to start checking:</p> <ul style="list-style-type: none"> ■ 0 specifies that every MOSFET substrate connection is checked before DC analysis. A warning message is issued if a MOSFET bulk is connected as shown in the following: for p-MOSFET ground or negative constant voltage source or n-MOSFET positive constant voltage source. The warning message is saved in <code>hsim.cck</code> by default or <code>out_file.cck</code> if you use the <code>-o</code> command line option. ■ 1 specifies that a MOSFET substrate is checked to see if it becomes forward-biased during simulation. ■ 2 specifies checking in DC analysis and during simulation.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <code>n</code> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
vt=v	Set the voltage threshold (applicable to dynamic check only) to avoid printing out an excessive number of messages. If the forward bias voltage exceeds the <code>vt</code> value, a warning message is printed. The default is 0.5 V.
ith=iv	Computes the bulk current under the forward-bias condition. If the absolute value of this bulk current is greater than <code>ith</code> , the element is reported. The default is 0 Amp.
start=t1	Specifies the start time for <code>vt</code> checks.
stop=t2	Specifies the stop time for <code>vt</code> checks.
model=m	Specifies the MOSFET type to be checked.
tag=t	Prints the specified tag in the report file at the beginning of a line to indicate a warning.
pwl_time=time	Propagates time-specific voltage values from piece-wise-linear (pwl) voltage sources. The voltage value (<code>v(time)</code>) used for propagation is defined by the pwl voltage source. This voltage value (<code>v(time)</code>) is then propagated under the same restrictions/application as a constant voltage source. By default, CCK only propagates from constant voltage sources. All time units are in seconds.
xhz=0 1	This option only works with the dynamic <code>ckSubstrate</code> setting (<code>mode=1</code> or <code>2</code>). When <code>xhz=1</code> , the <code>HSIMHZ</code> operation is also set internally, which means whenever you conduct a dynamic <code>cckSubstrate</code> analysis. The identified dynamic forward-bias bulk nodes are checked against the <code>HSIMHZ</code> result to see whether the identified bulk node is also a dynamic hz node. If it is, the bulk node is filtered out without appearing in the dynamic forward-bias violation report. The static forward-bias is not affected by this option. It continues following its original static forward-bias criteria as usual.

Appendix I: HSIM CCK

Checking Substrate Forward Bias

Argument	Description
<code>scope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
<code>skipScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control to prevent voltage propagation from occurring through devices or nodes.
<code>rmScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Description

The `cckSubstrate` command checks whether a MOSFET substrate becomes forward-biased. This check is performed before DC initialization and during transient simulation. In simulation, once a MOSFET substrate becomes forward-biased by more than a threshold, a warning message is printed.

Examples

```
cckSubstrate mode=2 num=300 vt=0.8 start=10n stop=50n start=90n stop=120n
```

Because `mode=2`, the entire MOSFET substrate is checked before performing DC initialization and during transient simulation.

Note: There is a limit of 300 warning messages that can be specified as shown in the example above.

During simulation, if a substrate becomes forward-biased by more than 0.8V, a warning message is printed. The following explanation illustrates when a warning message is printed.

At 10 ns, a p-MOSFET drain is 3 V and its substrate is 2.1 V. This p-MOSFET is forward-biased by $3-2.1=0.9$ V, which is larger than `vt=0.8` V. So a warning message is printed. Checking is accomplished between 10 ns and 50 ns and between 90 ns and 120 ns.

The following is the MOSFET substrate checking before DC output sample resulting from the previous command example.

```
*****
* MOS Substrate Checking Before DC
*****
in subckt (TLC), pmos (xi.pg)'s bulk (gnd) is connected to 0.00 volt
in subckt (TLC), nmos (xi2.mi10)'s bulk (vdd) is connected to 3.00 volt
```

In the previous example, the following definitions apply:

- xi.pg: p-MOSFET element name
- gnd: Node name
- xi2.mi10: Element name
- vdd: Node name
- TLC: Top-level circuit

The following is the MOSFET bulk and diode forward-biased in simulation output sample resulting from the same command example:

```
***** ****
* MOS Bulk and Diode forward-biased In Simulation
*      bulk forward bias: model=n1a, threshold=0.8 tag=n1
*****
(n1)@20.86n, nmos (xu5.mn) bulk forward-biased
v(b)=0.0000 v(d)=-1.51 v(s)=0.0000
```

This sample reads: At 20.86 ns, n-MOSFET xu5.m bulk becomes forward-biased with the following parameters:

- Bulk voltage (v(b)) is 0 V
- Drain voltage n1a(v(d)) is -1.51 V.

This element is reported since the difference derived by the following formula is greater than the default of 0.8 V.

$$(0 - (-1.51)) = 1.51V$$

Note: In transient simulation, both the substrate and diode forward bias are checked at each time interval and shows both results under the same header.

Checking for Unprotected Antenna Nodes

You can use the `cckAntGate` command to check if an antenna node is protected by diode.

cckAntGate

Checks if an antenna node is protected by a diode.

Appendix I: HSIM CCK

Checking Static Voltage Propagation Sharing

Syntax

```
cckAntGate <num=n> <scope='(subckt_name, inst_name,  
model_name)'> <skipScope='(subckt_name, inst_name,  
model_name)'> <rmScope='(subckt_name, inst_name,  
model_name)'>
```

Argument	Description
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
scope='(subckt_name, inst_name, model_name)'	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
skipScope='(subckt_name, inst_name, model_name)'	Specifies a filtering control to prevents voltage propagation from occurring through devices or nodes.
rmScope='(subckt_name, inst_name, model_name)'	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Description

This command checks if an antenna node is protected by a diode. If an antenna node has no connection to any reversed-bias diode, a warning is issued. An antenna node is defined as the primary input at top level, driven by ideal voltage sources, with a direct connection or through resistor/inductor to the transistor gate.

Checking Static Voltage Propagation Sharing

Multiple commands can share the same static voltage propagation. The commands that can share static voltage propagation are:

- cckMosV
- cckCapV
- cckResV

- cckDioV
 - cckPmosG_It_DS
 - cckNmosB_gt_DS
 - cckPmosB_It_DS
 - cckNmosG_gt_DS
-

Propagation Parameters

Propagation parameters are the command parameters that affect the result of the static voltage propagation. The propagation parameters are:

- limitmos
 - risepmosfallnmos
 - vpth
 - vnth
 - off_vpth
 - off_vnth
 - vlth
 - vhth
-

Propagation Sharing

Among all of the commands in the command file, groups of command that meet all the following conditions can share the same propagation:

1. Have the same command name
2. Use identical propagation parameters

Note: Command sharing does not depend on the ordering of the commands in the command file.

Example

Consider the following CCK command file:

Appendix I: HSIM CCK

Checking Static Voltage Propagation Sharing

```
cckStaticDCPPath nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1
cckPmosG_lt_DS inst=* vhth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000 rpttrace=1
cckNmosG_gt_DS inst=* vlth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000 rpttrace=1
cckPmosB_lt_DS inst=* vhth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000
cckNmosB_gt_DS inst=* vlth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000
cckDiode mode=0 subinfo=1 num=1000000
cckfloatgateisrc 1 subinfo=1
cckStaticHZNode nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1 fanout=3 pcap=1
cckStaticHZNode nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1 fanout=3 pcap=1
cckMosV mn1 model=n7* inst=* vnth=0 vpth=0 uvg=3.3 uvs=3.3 subinfo=1
cckMosV mn2 model=n_* inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mn3 model=na inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mn4 model=n inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckNmosG_gt_DS inst=* vlth=0.5 vt=0.02 vnth=0 vpth=0 subinfo=1 num=1000000 rpttrace=1
cckMosV mp1 model=p7* inst=* vnth=0 vpth=0 uvg=3.3 uvs=3.3 subinfo=1
cckMosV mp2 model=p_* inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mp4 model=p inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
```

There are 17 commands. Ten separate static voltage propagations are necessary in this case.

Propagations 1 to 5 (commands not supported for voltage propagation sharing):

```
cckStaticDCPPath nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1
cckDiode mode=0 subinfo=1 num=1000000
cckfloatgateisrc 1 subinfo=1
cckStaticHZNode nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1 fanout=3 pcap=1
cckStaticHZNode nmosOn=0 pmosOn=1.2 separate_file=1 subinfo=1 fanout=3 pcap=1
```

Propagations 6 to 8 (supported, but different command names are used):

```
cckPmosG_lt_DS inst=* vhth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000 rpttrace=1
cckPmosB_lt_DS inst=* vhth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000
cckNmosB_gt_DS inst=* vlth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000
```

Propagation 9:

```
cckNmosG_gt_DS inst=* vlth=0.5 vt=0.01 vnth=0 vpth=0 subinfo=1 num=1000000 rpttrace=1
cckNmosG_gt_DS inst=* vlth=0.5 vt=0.02 vnth=0 vpth=0 subinfo=1 num=1000000 rpttrace=1
```

Propagation 10:

```
cckMosV mn1 model=n7* inst=* vnth=0 vpth=0 uvg=3.3 uvs=3.3 subinfo=1
cckMosV mn2 model=n_* inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mn3 model=na inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mn4 model=n inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mp1 model=p7* inst=* vnth=0 vpth=0 uvg=3.3 uvs=3.3 subinfo=1
cckMosV mp2 model=p_* inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
cckMosV mp4 model=p inst=* vnth=0 vpth=0 uvg=1.2 uvs=1.2 subinfo=1
```

Running Digital Logic and Memory Diagnostics

These CCK commands are designed to help check stuck-at nodes, uninitialized latches, series MOSFET stack-up, flash memory over-erase conditions, toggle count, and trace event triggers. These commands are presented in alphabetic order to make it easier to find them.

cckFlashcore

Automatically detects the over-erasing condition in a flash memory cell and helps prevent involuntary operations during the read cycle.

Syntax

```
cckFlashcore <vtlow=vtlow_value> <vdsrdmax=vdsrdmax_value>
```

Argument	Description
vtlow=vtlow_value	Checks if $V_{th} < V_{tlow}$ during the phase of erasing. Otherwise the check is disabled. If $V_{th} < V_{tlow}$, a warning message is reported.
vdsrdmax=vdsrdmax_value	Checks if $V_{ds} > V_{dsrdmax}$ during the phase of reading. Otherwise the check is disabled. If $V_{ds} < V_{dsrdmax}$, a warning message is reported. If you use <code>vtlow</code> and <code>vdsrdmax</code> together, both checks launch simultaneously. If you do not specify <code>vtlow</code> or <code>vdsrdmax</code> , the corresponding check is disabled.

Description

This command is especially helpful in flash memory cell design. It provides an automatic detection of over-erasing condition in flash memory cell and ensures that there is no risk of involuntary operation during the read cycle. With this command, CCK generates warning messages and reports the flash core cell instance if V_{th} becomes less than a specified threshold during the erasing cycle or if V_{ds} is greater than a threshold during read cycle.

Note: The warning messages are stored in the `hsim.cck` or `file_name.cck`, where `file_name` is the output file name.

Examples

When you add the following command in the CCK command file after simulation is completed, a warning message is issued and stored in the .cck file if $V_{ds} > 4.5$ V.

```
cckFlashcore vdsrdmax=4.5
```

Appendix I: HSIM CCK

Running Digital Logic and Memory Diagnostics

The following is a typical warning message:

```
*  
*  
High Vds during reading.  
    model mos1, vds=4.911000, time=1.500004e-005.  
High Vds during reading.  
    model mos1, vds=5.060000, time=1.500005e-005.  
High Vds during reading.  
    model mos1, vds=5.805000, time=1.500005e-005.  
High Vds during reading.  
    model mos1, vds=6.550000, time=1.500005e-005.  
High Vds during reading.  
    model mos1, vds=6.848000, time=1.500006e-005.
```

Note: In the warning message shown above, Vt check is disabled because the parameter, vtlow, is not specified.

cckLatchUnInit

Finds initialized latches.

Syntax

cckLatchUnInit 0 | 1

Argument	Description
0	Turns off checking for uninitialized latches.
1	Detects uninitialized latches (default).

Description

A latch usually has a circuit loop. [Figure 105 on page 1005](#) shows two circuits. The first circuit has a pass gate in the loop and the second circuit does not. C and C are signals connecting to GATE nodes of pass gate transistors. The initial conditions of the A and B nodes in the loop affect the simulation result. If no node is driven at the initial stage, CCK reports it.

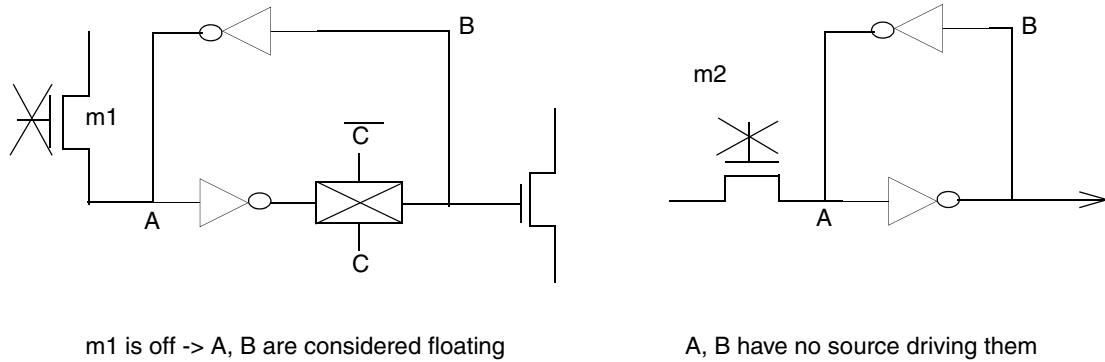


Figure 105 cckLatchUnInit: Signals connecting the Pass Gate through Nodes A & B

cckLatchUnInit checking is performed after DC initialization. In the first circuit shown in [Figure 105 on page 1005](#), if m1 is OFF and the pass gate is ON due to C/C. A and B have no other elements driving them. Therefore, they are considered to be floating.

cckLatchUnInit has a group of sub-commands which are described in the following sections:

- [cckLatchInElem on page 1005](#)
- [cckLatchSkipElem on page 1006](#)

Examples

`cckLatchUnInit 1`

Sample output from `cckLatchUnInit`:

```
*****
* Un-initialized Latch
*****
isolated latch node (xicg.xeset.n2)
isolated latch node (xicg.xeset.n1)
```

cckLatchInElem

Directs the `cckLatchUnInit` command to only check latches in a portion of the design.

Appendix I: HSIM CCK

Running Digital Logic and Memory Diagnostics

Syntax`cckLatchInElem <subckt=s> <inst=e>`

Argument	Description
subckt=s	Checks latches only in the specified subcircuit.
inst=e	Checks latches only in the specified instance.

Examples

The following command directs `cckLatchUnInit` to check latches in only a portion of the design, instances `xm` and `xcpu`:

`cckLatchInElem <inst=xm*> <inst=xcpu*>`

In the following command, `cckLatchUnInit` checks latches in all the instances of subcircuits `xLatch1` and `xLatch2`:

`cckLatchUnInit 1
cckLatchInElem subckt=xLatch1 subckt=xLatch2`

In the following command, the `subckt` and `inst` specify instances instantiated from a specific subcircuit for `cckLatchUnInit` to check their latches.

`cckLatchUnInit 1
cckLatchInElem <subckt=s> <inst=e>`

cckLatchSkipElem

Directs the `cckLatchUnInit` command to check all latches in a design, except for the specified portion.

Syntax`cckLatchSkipElem <subckt=s> <inst=e>`

Argument	Description
subckt=s	Checks latches only in the specified subcircuit.
inst=e	Checks latches only in the specified instance.

Description

This command checks all the latches, except for the specified portion. The `subckt` and `inst` options specify specific instances instantiated from a specific subcircuit for `cckLatchUnInit` to skip during its checking operation.

Examples

The following example checks all the other specified latches except those in instance xram.

```
cckLatchSkipElem inst=xram.*  
cckLatchUnInit 1
```

Checking Stack-up Transistors

CCK can verify the number of MOSFETs in series. If it exceeds the stack-up length, CCK reports the MOSFET stack path.

cckMaxStackUpNmos

Reports any path of more than predetermined n-MOSFET transistors in series.

Syntax

```
cckMaxStackUpNmos <length=len> <num=n>  
  <scope='(subckt_name, inst_name, model_name)'>  
  <skipScope='(subckt_name, inst_name, model_name)'>  
  <rmScope='(subckt_name, inst_name, model_name)'>
```

Argument	Description
length=len	Specifies the stack-up limit.
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
scope='(subckt_name, inst_name, model_name)'	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
skipScope='(subckt_name, inst_name, model_name)'	Specifies a filtering control to prevents voltage propagation from occurring through devices or nodes.

Appendix I: HSIM CCK

Checking Stack-up Transistors

Argument	Description
<code>rmScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Examples

The following CCK command reports any path of more than three n-MOSFET transistors in series.

```
cckMaxStackUpNmos length=3
```

The following is the NMOS stack-up limit output sample resulting from the previous command example.

```
*****
* NMOS stack up limit: 3
*****
Largest nmos stack: 4
    (1) stack length: 4
        xam.xd.xu_0.mn1
        xam.xd.xu_0.mn2
        xam.xd.xu_0.mn3
        xam.xd.xu_0.mn4
```

Note: These four NMOS transistors are in series.

cckMaxStackUpPmos

Reports any path of more than predetermined p-MOSFET transistors in series.

Syntax

```
cckMaxStackUpPmos <length=len> <num=n>
    <scope='(subckt_name, inst_name, model_name)'>
    <skipScope='(subckt_name, inst_name, model_name)'>
    <rmScope='(subckt_name, inst_name, model_name)'>
```

Argument	Description
<code>length=len</code>	Specifies the stack-up limit.
<code>num=n</code>	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <code>n</code> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.

Argument	Description
<code>scope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
<code>skipScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control to prevent voltage propagation from occurring through devices or nodes.
<code>rmScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Examples

```
cckMaxStackUpPmos length=2
```

The following is the PMOS stack-up limit output sample resulting from the previous command example.

```
*****
* PMOS stack up limit: 2
*****
Largest pmos stack: 3
    (1) stack length: 3
        xm.xu_6.xu51.mp
        xm.xwr_6.xu.mnoen
        xm.xwr_6.xu.mpi
```

Checking and Classifying the Stuck Nodes

CCK reports any node stuck at a certain voltage. If the stuck value is positive, CCK treats it as stuck-at-1. Otherwise, it is stuck-at-0. This type of warning is reported in the `.cckstuck0` and `.cckstuck1` files.

cckMaxStuckAt

Reports any node stuck at a certain voltage.

Syntax

```
cckMaxStuckAt0 <num=n> <node=nd> <skipSub=skip_sub_name>
    <skipnode=skip_node_name>
```

Appendix I: HSIM CCK

Checking and Classifying the Stuck Nodes

```
cckMaxStuckAt1 <num=n> <node=nd> <skipSub=skip_sub_name>
<skipnode=skip_node_name>
```

Argument	Description
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.
node=nd	Defines the portion of the design to be checked for stuck-at nodes.
skipSub=skip_sub_name	Skips the specified subcircuit.
skipnode=skip_node_name	Skips the specified node.

Examples

```
cckMaxStuckAt0 <num=300> <node=xm.*>
```

Checks the stuck-at nodes in instance xm.

cckToggleCount

Checks every node in the design.

Syntax

```
cckToggleCount <vref=v> <skipSub=skip_sub_name>
<skipnode=skip_node_name> <start=start+tim1
<stop=stop_tim1 <start=start_tim2 <stop=stop_tim2...
<tc0=0|1>>>
```

Argument	Description
vref=v	Specifies the reference voltage.
skipSub=skip_sub_name	Skips the specified subcircuit.
skipnode=skip_node_name	Skips the specified node.

Argument	Description
start=start+tim1 stop=stop_tim1	Specifies the time windows in which cckToggleCount checks. You can specify multiple start and stop parameters.
tc0=0 1	A value of 0 (the default) disables the additional zero-toggling report, so only the active signals are reported. A value 1 enables the additional zero-toggling report.

Examples

```
cckToggleCount vref=5.0 skipsub=cnand start=20n stop=40n start=60n stop=80n
```

If any node crosses the vref=v reference voltage, the toggle count at this node is increased by 1. At the end of the simulation, the toggle count is reported for every node. This report is used to compute the toggle frequency at each node and then multiply with the node capacitance to estimate the power consumption. The output is located in either the hsim.ccktoggle or output.ccktoggle file.

Example 87 Sample hsim.ccktoggle File Output.

```
*Synopsys Corporation.  
*HSIM Win32 Debug Version 5.0 - 175607242004  
*Tracking No - HSIM 2004.30.3  
*Copyright (C) 1998 - 2004. All rights reserved.  
*  
*  
* cckToggleCount vref=0.7  
*   format: node_name toggle_count  
aa<0> 1  
aa<12> 4  
aa<16> 1  
dd<0> 1  
xic5m29f016b_verilog.xibaaps.i25_d 1  
xic5m29f016b_verilog.xibaaps.i26_d 2
```

cckConnReport

Performs a high connectivity node check.

Syntax

```
cckConnReport <connth=value> <subinfo=[0 | 1]>  
    <dev_Type_Term='(device_type, terminal_type)'>  
    <rmDev='(subckt_name, instance_name, <model_name>)'>  
    <report='conn'>
```

Appendix I: HSIM CCK

Checking and Classifying the Stuck Nodes

Argument	Description
connth=value	Specifies the device count threshold value. CCK reports node names with connected devices greater than the specified threshold value. The default is 500. If you specify other options such as dev_Type_Term or rmDev, connth is evaluated after those options.
subinfo=[0 1]	If set to 1, specifies that the additional subcircuit information is printed along with the device full hierarchy name. The default is 0.
dev_Type_Term='(device_type, terminal_type)'	Conducts the connectivity count based on specified device type and terminal type symbol. The device_type can be: MOS (mosfet), DIO (diode), RES (resistor), CAP (capacitor), IND (inductor), or BJT (bipolar).The terminal_type_symbol can be: 1, 2, 3, 4, 5, 6, 7, 8, or 9.
rmDev='(subckt_name, instance_name, <model_name>)'	Provides a connectivity count filtering mechanism based on the specified subcircuit, instance, or model name. Wildcards are supported for instance name and model name expressions, but not for a subcircuit name expression.
report='conn'	Dumps the status to indicate when a node is a constant voltage source node or a variable voltage source node. Additional connectivity information for voltage source nodes is dumped when you specify report='conn'.

Description

You can specify the appropriate device_type with the terminal_type_symbol to obtain the desired connectivity count criteria. See [Table 127](#).

Table 127 device_type References

term dev_type	MOS	RES IND CAP	DIO	BJT
1	drain	Plus	anode	collector
2	gate	minus	cathode	base
3	source			emitter
4	bulk			substrate
5	drain && gate			collector && base
6	source && gate			emitter && base

Table 127 device_type References

term dev_type	MOS	RES IND CAP	DIO	BJT
7	drain && bulk			collector && substrate
8	source && bulk			emitter && substrate
9	drain && source			collector && emitter

Note: If a node is connected to any terminals of the MOSFET devices, such as source and bulk as shown in [Figure 106](#), it is counted as one device.

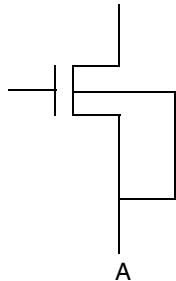


Figure 106 MOSFET Example

The source and bulk of the MOSFET are connected with the A node.

```
cckConnReport connth=150 dev_type_term='(res, 1)'
```

The previous example reports nodes that have a connectivity count over 150. The connectivity count only considers the connection to resistor plus node.

```
cckConnReport connth=150 dev_type_term='(mos)'
```

The previous example reports nodes that have a connectivity count over 150. The connectivity is counted whenever a node is connected to any terminal of MOSFET devices. By omitting the terminal_type_symbol specification, the connectivity count criteria becomes the connection to any terminal of the specified device_type.

Appendix I: HSIM CCK

Checking and Classifying the Stuck Nodes

```
cckConnReport connth=150 dev_type_term='(mos, 2)' dev_type_term='(mos, 3)'  
dev_type_term='(mos, 4)'
```

The previous example reports nodes that have a connectivity count over 150. The connectivity is counted whenever the node is connected to gate, source, or bulk of MOSFET devices.

```
cckConnReport connth=150 dev_type_term='(mos)' dev_type_term='(dio, 1)'
```

The previous example reports nodes that have a connectivity count over 150. The connectivity is counted whenever the node is connected to any terminal of MOSFET devices or anode of diode devices. In the previous two examples, note that you can specify multiple dev_type_term expressions with an “OR” relationship in between.

The rmDev option provides a connectivity count filtering mechanism based on the specified subcircuit, instance, or model name. Wildcards are supported for instance name and model name expressions, but not for a subcircuit name expression.

```
cckConnReport connth=150 rmDev='(adder1, x1.*, ph1)'
```

In the previous example, the connectivity count skips devices of the model ph1 with an x1. instance naming prefix that resides in the adder1 subcircuit. The expression in rmDev has an “AND” relationship.

```
cckConnReport connth=150 rmDev='(adder1,,)' rmDev='(adder2,,)'
```

In the previous example, the connectivity count skips devices that reside in the adder1 subcircuit as well as the ones in the adder2 subcircuit.

```
cckConnReport connth=150 rmDev='(,pmos1)' rmDev='(,nmos1)'
```

In the previous example, the connectivity count skips devices of model pmos1 and nmos1. In the previous two examples, multiple rmDev specifications have an “OR” relationship.

Consider a node, A, where M1, M2, M3, and M4 are connected as shown in [Figure 107](#), where n1 and p1 are model names:

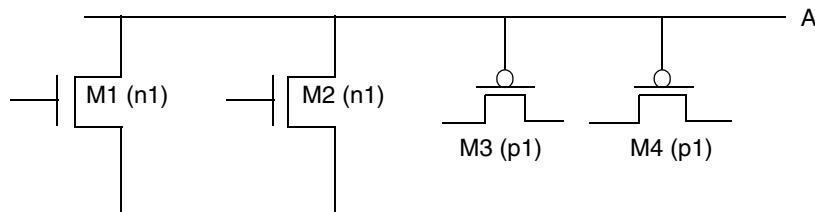


Figure 107 n1 and p1 Model Names

```
cckConnReport connth=3 dev_type_term='(mos)'
```

In the previous example, node A is reported because it has 4 MOSFETs connected to it.

```
cckConnReport connth=3 dev_type_term='(mos,2)'
```

In the previous example, node A is not reported because there are 2 MOSFETs connected to it with gate.

```
cckConnReport connth=3 rmDev='(,p1)'
```

In the previous example, node A is not reported because there are 2 MOSFETs that instantiate the p1 model connected to it.

Assume that there is a vdd voltage source node tied with a 5 V constant VVdd voltage supply. When you specify report='conn' there is an additional connectivity section printed in the cckConnReport report for this voltage source node. For example, the following example does not include report='conn':

```
cckConnReport connth=150 subinfo=1

Node #2: vdd
Subinfo: TLC
Status: constant voltage source node
Data :
Connections: 3439841
...
```

The next example has a report='conn' argument:

```
cckConnReport connth=150 subinfo=1 report='conn'

Node #2: vdd
Subinfo: TLC
Status: constant voltage source node
Connectivity:
    VVdd : plus
    Subinfo: TLC
    Attributes: type=vdc, V= 5v
    Data :
Connections: 3439841
...
```

The additional connectivity information triggered by report='conn' includes the associated Vsrc supply name (VVdd), the terminal type (plus), and its voltage value (5 V).

Assume there is a Vvsupply33 pwl voltage source connecting to the a vsupply33 node. Then there is a dump of "Status: voltage source node" to indicate that the node is a variable voltage source node. In addition, with

Appendix I: HSIM CCK

Checking and Classifying the Stuck Nodes

report='conn', there is a dump of the voltage source device connecting to this node. For example, the following example does not include report='conn':

```
cckConnReport connth=150 subinfo=1

Node #2: vsupply33
Subinfo: TLC
Status: voltage source node
Data :
Connections: 3439841
...
```

The next example has a report='conn' argument:

```
cckConnReport connth=150 subinfo=1 report='conn'

Node #2: vsupply33
Subinfo: TLC
Status: voltage source node
Connectivity:
    VVsupply33 : plus
        Subinfo: TLC
        Attributes: type=vpwl
    Data :
Connections: 3439841
...
```

Note that the Status information can be printed as well as the Data information by default.

Note: The reported node names are in the full hierarchy naming convention with the corresponding subcircuit name information (if subinfo=1 is set), and the node names are sorted in descending order by the associated connected device count value.

Examples

```
cckConnReport connth=1000
```

CCK reports the node name with a connected device count greater than 1000 and generates the following report:

```

*
* CCK Command : cckConnReport
* connth=1000 subinf
o=1
*
*
Node #1: vss
Subinfo : TLC
Data :
    Connections: 50064
Node #2: vdd
Subinfo : TLC
Data :
    Connections: 31965
Node #3: vcci
Subinfo : TLC
Data :
    Connections: 4780
Node #4: xids3234a1.vdd_tempcore
Subinfo : d32a34a1_g171
Data :
    Connections: 4210
Node #5: xids3234a1.xcontrol.net138
Subinfo : d32a34a1_g171 control_g167
Data :
    Connections: 3320

```

Running Interactive Circuit Debugging Command for a Tracking Circuit

CCK traces the cause of a digital state change at a given node. Refer to the HSIM Simulation Reference Manual: *Interactive Debugging, List of Interactive Mode Commands* for information on obtaining the interactive debugging environments.

Note: It is strongly recommended that all the logic signals are printed out before using this feature. This is accomplished by using the `lprint v(*)` command. The `HSIMVDD` value must also be set to specify the low and high voltage logic thresholds. The commands used to track circuits are based on FSDB files and must be specified as FSDB format to use the commands.

Note: In a large design, the FSDB file can be created by first running the HSIM simulation. At a later time in the process, enter the Interactive Mode and use the previously created FSDB files for the `ntrig` commands. This reduces the number of times that simulations for `ntrig` commands must be rerun.

Appendix I: HSIM CCK

Finding a Node First State Change After a Specified Time

Finding a Node First State Change After a Specified Time

ntrig

Finds node state changes.

Syntax

```
ntrig node_name <-t time> <-f file1> <-mt time_interval>
```

Examples

Example 88

```
HSIM> ntrig -t 9      m188:gate
time=9.000000 ns
fsdb file #1=hsim.fsdb.
signal=m188(10)
Only 1 possibility.
possibility: #1
    m188:gate from 0 to 1 in [9.3260n, 9.3550n]
<- m177(1) from 1 to 0 in [9.2870n, 9.3090n]
<- m189(2) from 0 to 1 in [9.2470n, 9.2760n]
<- m429(3) from 1 to 0 in [9.2070n, 9.2290n]
<- m432(4) from 0 to 1 in [9.1680n, 9.1970n]
<- n_276(5) from 1 to 0 in [9.1280n, 9.1500n]
<- m173(6) from 0 to 1 in [9.0870n, 9.1180n]
<- i_22221.m31(7) from 1 to 0 in [9.0380n, 9.0750n]
<- i_22221.m28:gate from 0 to 1 in [8.9790n, 9.0200n]
i_22221.m28:gate is a voltage source!
```

Example 89

```
HSIM > ntrig out -t 50 -mt 12
Node out (4):
Total number of possibilities: 2

possibility: #1
    out (4) from 1 to 0 in [52.06n, 52.57n]
<- s1 (6) from 0 to 1 in [51.03n, 51.62n]
<- in2 (2) from 1 to 0 in [50.29n, 50.70n]
in2 is a voltage source!

possibility: #2
    out (4) from 1 to 0 in [52.06n, 52.57n]
<- s3 (8) from 0 to 1 in [43.37n, 44.22n]
<- s2 (7) from 1 to 0 in [41.60n, 43.08n]
<- in3 (3) from 0 to 1 in [40.29n, 40.70n]
in3 is a voltage source!
```

The out node changed its state from 52.06 ns to 52.57 ns. Since -mt is specified, the search travels backward to 40.67 ns (=52.57 – 12). It finds nodes s1 and s3 changed their states at 51.03 ns and 43.37 ns, respectively. These two nodes can be reported. Also node s1 was triggered to change by node in2 at 50.29 ns. Since in2 is a voltage source, the tracing from s1 is stopped. The other possibility is from s3 to trace to s2 and finally to in3.

In addition, the ntrig command traces the circuit for the cause of a state change according to one FSDB file and retrieve the corresponding signals of another user-specified FSDB file for comparison.

```
ntrig <-t time> <-f file1> <-cf file2> node_name
```

Note: file1 and file2 are fsdb files.

file1 is used to find the first state change for the node after the specified time. If -f is not set, the FSDB file of the current run is the default. If -t is not set, the default time is 0, file2 needs to be specified in order to compare values from two FSDB files.

Example 90

```
HSIM> ntrig -t 3000      -cf try.fsdb d0
      time=3000.000000 ns
      fsdb file #1=hsim.fsdb.
      fsdb file #2=try.fsdb.
      signal=d0
      possibility: #1
      d0
          file 1: 1 to 0 in [3053.8080n, 3053.8410n]
          file 2: 1 to 0 in [3053.8060n, 3053.8390n]
      <- xi259.net86
          file 1: 0 to 1 in [3053.7510n, 3053.7900n]
          file 2: 0 to 1 in [3053.7490n, 3053.7880n]
      <- xi259.net71
          file 1: 1 to 0 in [3053.7020n, 3053.7250n]
          file 2: 1 to 0 in [3053.7000n, 3053.7230n]
      <- xi259.net75
          file 1: 0 to 1 in [3053.6560n, 3053.6760n]
          file 2: 0 to 1 in [3053.6540n, 3053.6740n]
      <- doi
          file 1: 1 to 0 in [3053.6370n, 3053.6550n]
          file 2: 1 to 0 in [3053.6350n, 3053.6530n]
doi is a voltage source.
```

A compare.rc file is created each time the ntrig command is used to assist to bring up all the identified digital signals in the nWave environment. The compare.rc file is used as follows:

- Select File > Restore Signals in nWave
- Choose compare.rc

Appendix I: HSIM CCK

Finding a Node First State Change After a Specified Time

Sometimes it is useful to find out why at a certain time specified by t1, a node is at a specific digital state while in another simulation run, the same node is at a different state. It is valuable to know what other signals caused this node to be at different states at a given time. This is achieved using ntrig -ic, as follows:

```
ntrig -ic <-t t1> <-f file1> <-cf file2> node_name
```

This feature compares two files (file1 and file2) to find out which other signals may cause this node to have different states at time t1. Time is measured in nanoseconds. If -t is not set, the default time is 0. If -f is not set, the current FSDB is a default file. The second FSDB file (file2) needs to be specified for comparison. In this case, the program only examines at time t1 to determine why the given node was at different states.

Another way to probe a node is to provide the node ID. The usage and functionalities are same as ntrig, except that the node hierarchical ID is used, instead of node names.

```
intrig node_id <-t time> <-f file1> <-mt time1>
intrig node_id <-t time> <-f file1> <-cf file2>
intrig node_id -ic <-t time> <-f file1> <-cf file2>
intrig node_id -diff <-t time> <-f file1> <-cf file2>
```

The following is an example of an interactive circuit debugging command for tracking a circuit. [Figure 108 on page 1020](#) illustrates this example.

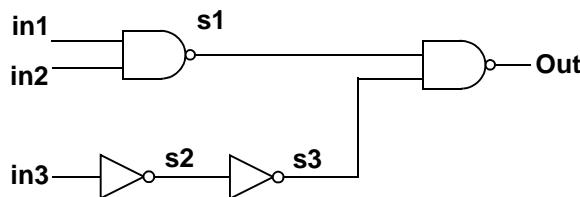


Figure 108 intrig Circuit

This example is stored in directory /\$HSIM_HOME/tutorial/logic. The three primary inputs are in1, in2, and in3. Different choice of input pattern may lead to different output voltage, v(out). For this circuit, there are two different input vectors.

In the first run of testb.sp, the following syntax is used:

```
Vvdd vdd 0 5
r1 vdd in1 1
Va2 in2 0 pwl (0 5 5e-08 5 5.1e-08 0 10e-08 0)
Va3 in3 0 pwl (0 0 4e-08 0 4.1e-08 5 10e-08 5)
```

In the second run of testw.sp, the following syntax is used:

```
Vvdd vdd 0 5
r1 vdd in1 1
Va2 in2 0 pwl (0 5 5e-08 5 5.1e-08 0 10e-08 0)
Va3 in3 0 pwl (0 0 4e-08 0 4.1e-08 0 10e-08 0)
```

The only difference in the two cases is the in3 signal. Follow these steps to specify different options in ntrig command.

1. Run the runw script.

Run script runw (top netlist is testw.sp). An FSDB file called w.fsdb is generated. This file is required for later use.

2. Enter the interactive mode.

At the command line, type the following:

```
hsim testb.sp
```

HSIM stops at 80 ns and enters the interactive mode.

3. Find the latest path.

When bringing up the waveform for the out node in the hsim.fsdb file, it shows that v(out) goes from 1 to 0 at around 52 ns. To find out the cause, type the following command at the interactive simulation mode.

```
HSIM> ntrig -t 50 out
```

The following signal list is displayed:

```
out (4) has only one possibility:
possibility: #1
out (4) from 1 to 0 in [52.0600n, 52.5760n]
<- s1 (6) from 0 to 1 in [51.0370n, 51.6290n]
<- in2 (2) from 1 to 0 in [50.2990n, 50.7000n]
in2 is a voltage source!
```

4. Find multiple paths.

Based on the schematics in [Figure 108 on page 1020](#), the state change of the out may be influenced by both s1 and s3. The previous ntrig command only shows one path. In order to find out both paths triggering v(out) to digital low stage after 50 ns, the -mt option is required. Type the following command at the interactive mode:

```
HSIM> ntrig -t 50 -mt 10 out
```

Appendix I: HSIM CCK

Finding a Node First State Change After a Specified Time

The following messages are displayed:

```
out (4) has two possibilities:  
possibility: #1  
out (4) <2> from 1 to 0 in [52.0600n, 52.5760n]  
<- s1 (6) from 0 to 1 in [51.0370n, 51.6290n]  
<- in2 (2) from 1 to 0 in [50.2990n, 50.7000n]  
in2 is a voltage source!  
possibility: #2  
out (4) <2> from 1 to 0 in [52.0600n, 52.5760n]  
<- s3 (8) from 0 to 1 in [43.3740n, 44.2230n]  
<- s2 (7) from 1 to 0 in [41.6020n, 43.0830n]  
<- in3 (3) from 0 to 1 in [40.2990n, 40.7000n]  
in3 is a voltage source.
```

Both paths are displayed. s3 changes its state at about 9 ns earlier than out.
If the following command is issued and only one path is listed:

```
HSIM> ntrig -t 50 -mt 8
```

5. Trace the difference in a node.

V(out) is at different states at time 60 ns in the first run testw.sp and current run testb.sp. To find out why they are different, type the following command at the interactive simulation mode:

```
HSIM> ntrig -t 60 out -diff -cf w.fsdb
```

The following messages are displayed:

```
out (4) has only one possibility:  
possibility: #1  
out (4)  
    file 1: 1 to 0 in [52.0600n, 52.5760n]  
    file 2: 1 to 1 in [0.0000n, 52.5760n]  
<- s3 (8)  
    file 1: 0 to 1 in [43.3740n, 44.2230n]  
    file 2: 0 to 0 in [0.0000n, 44.2230n]  
<- s2 (7)  
    file 1: 1 to 0 in [41.6020n, 43.0830n]  
    file 2: 1 to 1 in [0.0000n, 43.0830n]  
<- in3 (3)  
    file 1: 0 to 1 in [40.2990n, 40.7000n]  
    file 2: 0 to 0 in [0.0000n, 40.7000n]  
in3 is a voltage source.
```

CCK outputs a comparison list of signals back-to-back.

6. Find the difference in two initial conditions.

Type the following command at the interactive simulation mode:

```
HSIM> ntrig -t 60 -ic -cf w.fsdb out
```

The following messages are displayed:

```

out (4) has only one possibility:
possibility: #1
out (4) voltage: 0 vs
<- s3 (8) voltage: 1 vs 0.
<- s2 (7) voltage: 0 vs 1.
<- in3 (3) voltage: 1 vs 0.
in3 is a voltage source.
  
```

Running Timing Checks

These CCK commands are designed to help check RC parasitic delay estimation, charge/discharge path delays, input slew rate, inter-nodal delays, and event times.

cckMaxNmosToVdd

Checks NMOS limit on charging-up path to VDD.

Syntax

`cckMaxNmosToVdd <length=leng> <num=n>`

Argument	Description
<code>length=leng</code>	Specifies the length threshold. The default is 0.
<code>num=n</code>	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <code>n</code> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.

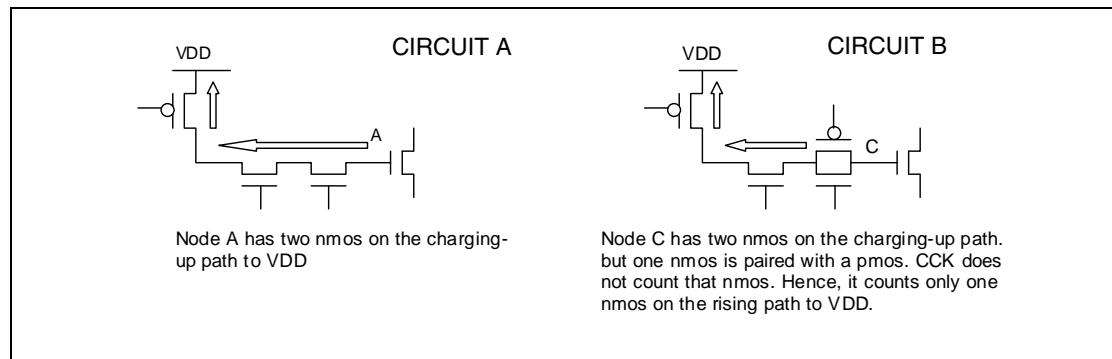
Description

Check the number of n-MOSFETs on every rising path to VDD. If it is larger than the specified length (`leng`), CCK issues a warning until the number of warnings exceeds `num=n`. This command is to make sure the charging-up paths do not have too many n-MOSFETs.

In [Figure 109 on page 1024](#), if `length=1`, a warning is issued for Circuit A. Circuit B has a p-MOSFET accompanying an n-MOSFET, CCK does not count that particular n-MOSFET. Hence, no warning is reported for Circuit B.

Appendix I: HSIM CCK

Running Timing Checks

*Figure 109 Example p-MOSFET and n-MOSFET Circuits***Examples**

```
cckMaxNmosToVdd length=2 num=30
```

The following is the NMOS limit on charging-up path output sample resulting from the command example given above:

```
*****
* NMOS limit on charging up path: 2, num=30
*****
Largest number of nmos on rising path: 4
(1) number of nmos on path: 4
    end node (xam.xrd.nout)
        xam.xrd.xu8.mn
        xam.xrd.xu6.mn
        xam.xrd.xu7.mn
        xam.xrd.xu9.mn
        xam.xrd.xu24.mp
from node (vdd) with 1.65 volt
```

cckMaxPmosToGnd

Checks PMOS limit on discharging path to GND.

Syntax

```
cckMaxPmosToGnd <length=leng> <num=n>
```

Argument	Description
length=leng	Specifies the length threshold. The default is 0.

Argument	Description
num=n	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.

Description

Check number of p-MOSFETs on discharging path to ground. If it exceeds the specified limit length=leng, a warning is reported, until the number of warnings reaches num=n. Similar to the previous command, if a p-MOSFET is paired with another n-MOSFET, this p-MOSFET is not counted. See [Figure 110 on page 1025](#).

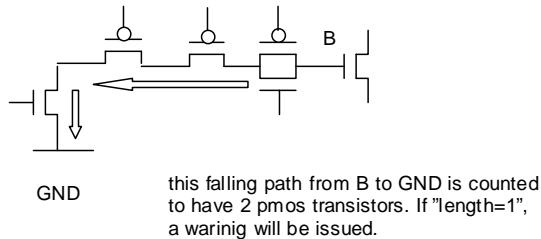


Figure 110 Example p-MOSFET Circuit

cckMaxStackUpNmos

cckMaxStackUpNmos checks the number of n-MOSFET transistors in series. If it exceeds the stack-up length, CCK reports the MOSFET stack path. See [cckMaxStackUpNmos on page 1007](#) for a detailed description.

cckMaxStackUpPmos

cckMaxStackUpPmos checks the number of p-MOSFET transistors in series. If it exceeds the stack-up length, CCK reports the MOSFET stack path. See [cckMaxStackUpPmos on page 1008](#) for a detailed description.

cckMeasPathDelay

Traces the path delay between the source node and target node.

Appendix I: HSIM CCK

Running Timing Checks

Syntax

```
cckMeasPathDelay <source=source_node_name>
    <target=target_node_name> <outFile=output_file_name>
    <srcEdge=[1|-1|0]> <targEdge=[1|-1|0]>
    <start=start_time> <stop=stop_time>
    <post=fsdb_file_name>
```

Argument	Description
source=source_node_name	Specifies the source node name, permits wildcards in the expression, and searches all trigger nodes satisfying the parameter setting. For *, CCK reports all path triggering the target nodes.
target=target_node_name	Specifies the target node name. Wildcards are permitted in the expression. However, they cannot be as simple as a single asterisk (*). Wildcards must be specified similar to the following example: target=data*.
outFile=output_file_name	Specifies the output file name for storing check results. If it is not provided, hsim.cck is the default file name.
srcEdge=[1 -1 0]	Specifies one of the following values: <ul style="list-style-type: none"> ■ 1 looks for rising trigger nodes only. ■ -1 looks for falling trigger nodes only. ■ 0 looks for both rising and falling trigger nodes.
targEdge=[1 -1 0]	Specifies one of the following values: <ul style="list-style-type: none"> ■ 1 looks for rising trigger nodes only. ■ -1 looks for falling trigger nodes only. ■ 0 looks for both rising and falling trigger nodes.
start=start_time	Specifies the time span allowed for cckMeasPathDelay to perform its checks. If you do not specify start and stop, it checks the entire transient time. If you specify a value for start, cckMeasPathDelay checks from specified start_time time to the end of transient simulation.
stop=stop_time	Specifies the time span allowed for cckMeasPathDelay to perform its checks. If you do not specify start and stop, it checks the entire transient time. If you specify a value for stop, cckMeasPathDelay checks from 0 to the specified stop.
post=fsdb_file_name	The analysis is conducted in the post-process mode. This means that the program directly analyzes the path measurement through the specified existing HSIM FSDB output file instead of looking into the current simulation data. When using cckMeasPathDelay, you must specify lprint in the netlist file because cckMeasPathDelay uses the FSDB file to generate the report.

Description

This command traces the path delay between the source node and target node. Differing from the .measure command that only measures the delay between

source node and target node, `cckMeasPathDelay` has the path tracking capability that reports all delay paths between the source node and target nodes.

When it is necessary to measure the path between node A and node D, the source measures incremental paths between node A to node B, then node B to node C, and finally node C to node D. This segmented path A to B, B to C, and C to D from the source to the target is reported.

Examples

If the `cckMeasPathDelay` is added to the CCK command file as shown in the following syntax:

```
cckMeasPathDelay source=a_* target=out_* outFile=result.out
```

This command reports the delay paths with the node name beginning with `a_` to the target node name with the prefix, `out_`. Because the `srcEdge` is not specified, it traces both the rising edge and falling edge. The result is written into the file called `result.out`. The example below shows a typical report.

```
* Measurement of Path Delay.
* source nodes: "a_*".
* target nodes: "out_*".
* source edge: don't care.
* target edge: don't care.
* Path delay measurement for target node: (out_a)
Path: #1
    out_a (525) rising in [1721.0000n, 1721.0200n]
    <- zz_i1/1 (4995) falling in [1720.9700n, 1721.0100n]
    <- 146 (12) rising in [1720.9200n, 1720.9800n]
    <- m2_i1/7 (3516) <2> falling in [1720.8400n, 1720.8800n]
    <- 147 (13) rising in [1720.7900n, 1720.8300n]
    <- div_i0/4 (1903) falling in [1720.1500n, 1720.1800n]
    <- 144 (10) rising in [1720.0800n, 1720.1500n]
    <- cp_i1/1 (1406) falling in [1720.0100n, 1720.0500n]
    <- a_a (4023) rising in [1719.9900n, 1720.0000n]
a_a is a voltage source!
* total number of path found=1.
```

cckNodeMaxRF

Performs a quick estimate of the rising and falling delay at a given node.

Syntax

```
cckNodeMaxRF <node=fullPathnodeName> <tth=t1>
cckNodeMaxRF <subckt=subcktName> <node=nodeNameInSub>
    <tth=t1>
cckNodeMaxRF <skipNode=fullnodeName1>
    <skipNode=fullnodeName2> <tth=t1>
```

Appendix I: HSIM CCK

Running Timing Checks

Argument	Description
<code>node=fullPathNode Name</code>	Specifies the node to check for rising and falling delay.
<code>tth=t1</code>	Specifies the time threshold. If a node has a rise/fall delay longer than the specified using t_1 value, it is reported.
<code>subckt=subcktName</code>	Specifies the subcircuit nodes to check for rising and falling delay.
<code>skipNode=fullNode Name1</code> <code>skipNode=fullNode Name2</code>	Specifies the nodes to skip in the check for rising and falling delay.

Description

`cckNodeMaxRF` performs a quick estimate of the rising and falling delay at a given node. It examines all the adjacent elements of the node to find the most resistive elements and calculate its effective resistance (R_{max}), then find its node capacitance (C_{node}), which is multiplied by R_{max} . The resulting product is an estimate of rising and falling delay to this node, since very likely, both the rising and falling paths go through the most resistive element. This check does not trace from this node to VDD or ground.

When the estimated delay is larger than the time threshold (t_1), it reports this node and its capacitance, its most resistive adjacent element, and the estimated delay. The output file is in either the `hsim.cck` or `output.cck` files. The nodes are selected by their full path names or by subcircuit nodes. Also, `cckNodeMaxRF` checks all the nodes, except the skipped nodes.

The following command examines the nodes in the `xam` module and all `wordline*` nodes in all the instantiations of subcircuit `atc`. Estimated delays larger than 4ns are reported.

Examples

```
cckNodeMaxRF node=xam.* subckt=atc node=wordline* tth=4n
```

Sample `cckNodeMaxRF` output content generated using the syntax given in the above example.

```
*****
* estimate node max rise/fall delay
* tth=4e-009
* subckt=atc node=wordline*
* node=xam.*
* format: node_name est_delay(ns) (elem_with_max_R ohm)
* (note: if an elem is always off, its ohm is 1e15)
*****
xam.xctl.nlined2<5> 30.2 ns (max_R xcam.xctl.xi6_5.mu2 1e+5 ohm)
xam.xctl.nonseqc2 40.5 ns (max_R xcam.xctl.xi10.mu1 3e+5 ohm)
```

Running Static RC Delay Analysis – Estimate Slew Rate

Static RC Delay Analysis finds the RC delay to the GATE nodes of MOSFETs. The charging paths from VDD and the discharging paths from GND to each GATE node is found, and their RC delays are then computed. This analysis provides the following information for each signal:

- Circuit loading
- Slew rate

cckRCDlyPath

Performs static RC delay analysis. Note that when you run this command, CCK automatically disables HSIM transient simulation.

Syntax

```
cckRCDlyPath [0|1] fanoutonly=[1|0] <ndCapThreshold=value>
or
cckRCDlyPath [0|1] <fanoutonly=[1|0]> <static_init=[0|1]>
<pmosOn=value> <nmosOn=value> <pwl_time=time>
<inst=instance_name> <subckt=subckt_name>
<skipInst=instance_name> <skipSub=subckt_name>
<limitRisePmosFallNmos=[0|1]> <fallDly='(value1 value2
0|1)'> <riseDly='(value1 value2 0|1)'>
```

Argument	Description
[0 1]	Specify 1 to perform RC delay analysis after DC operating point analysis. If you specify the second type of syntax for <code>cckRCDlyPath</code> , on top of any other filtering control the analysis only targets nodes with connections to MOSFET gate or BJT base. The default is 0.

Appendix I: HSIM CCK

Running Static RC Delay Analysis – Estimate Slew Rate

Argument	Description
<code>fanoutonly=[1 0]</code>	Set to 1 to report the RC delay path to the gate node. When set to 0, the RC delay path can reach to either the gate or output node. The default is 1.
<code>ndCapThreshold=value</code>	Acts as an additional filter to the existing <code>cckRCDlyPath</code> checking criteria. The <code>cckRCDlyPath</code> command originally reports all the rising/falling paths with path delay values greater than 0 and less than 100 n. The new <code>ndcapThreshold=7e-10</code> argument specifies an additional reporting criteria, not only the reported path delay value within the 0n to 100 n range, but also that the ending node of the path (normally gate node) that has a node capacitance greater than 7e-10F.
<code>static_init=[0 1]</code>	If you specify 1, the static voltage initialization procedure propagates the <code>Vsrc</code> constant value in the circuit as per different device type: <ul style="list-style-type: none"> ■ Resistor: full propagation. ■ Inductor: full propagation. ■ Capacitor: stop propagation. ■ Diode: stop propagation. ■ BJT: stop propagation. ■ n-MOSFET: if V_g is a constant value and $V_g > nmosOn$ then full propagation, otherwise stop propagation. ■ p-MOSFET: if V_g is a constant value and $V_g < pmosOn$ then full propagation, otherwise stop propagation. By default (<code>static_init=0</code>), this initialization procedure is disabled and only the constant <code>Vsrc</code> nodes can have the constant voltage value.
<code>pmosOn=value</code>	Specifies the user-defined conducting threshold value for a P type MOSFET. If p-MOSFET has its V_g as a constant voltage value and $V_g > pmosOn$, then the static voltage initialization procedure conducts full voltage propagation from MOSFET drain (source) to source (drain). This option is only effective when <code>static_init=1</code> is also set. Upon its effectiveness its default value is Max (constant <code>Vsrc</code> value in design). If the design has no constant <code>Vsrc</code> , the default value is set equal to the <code>HSIMVDD</code> value.
<code>nmosOn=value</code>	Specifies the user-defined conducting threshold value for an N type MOSFET. If an n-MOSFET has its V_g as a constant voltage value and $V_g > nmosOn$, the static voltage initialization procedure conducts full voltage propagation from MOSFET drain (source) to source (drain). This option is only effective when <code>static_init=1</code> is also set. Upon its effectiveness its default value is 0.
<code>pwl_time=time</code>	Specifies that all the pwl and pulse <code>Vsrc</code> are internally converted to a constant <code>Vsrc</code> assigned by the constant value at the specified time. This option is only effective when <code>static_init=1</code> is also set.
<code>inst=instance_name</code>	Defines the instance name such that the associated delay path is reported if the partial or entire path is within the specified instance.

Argument	Description
subckt=subckt_name	Defines the subcircuit name such that the associated delay path is reported if the partial or entire path is within the specified subcircuit.
skipInst=instance_name	Defines the instance such that the associated delay path is not reported if the partial or entire path is within the specified instance name.
skipSub=subckt_name	Defines the subcircuit such that the associated delay path is not reported if the partial or entire path is within the specified subcircuit name.
limitRisePmosFall Nmos=[0 1]	Setting a value of 1 (default) means, when tracing a rising path, cckRCDlyPath only considers a p-MOSFET as a legitimate MOSFET device on the path. A falling path is considered an n-MOSFET. Setting a value of 0 means the rising or falling path does not apply the MOSFET validation check.
fallDly='(value1 value2 0 1)'	The falling path is reported if the path delay is less than <code>value1</code> or greater than <code>value2</code> . For non-default mode (when the third argument value is 1), only the falling path with the delay value less than <code>value2</code> and greater than <code>value1</code> is reported.
riseDly='(value1 value2 0 1)'	The rising path is reported if the path delay is less than <code>value1</code> or greater than <code>value2</code> . For non-default mode (when the third argument value is 1), only the rising path with the delay value less than <code>value2</code> and greater than <code>value1</code> is reported.

Description

The `cckRCDlyPath` command performs static RC delay analysis to find the RC delay to the GATE nodes of MOSFETs. It finds the charging paths from VDD and the discharging paths from GND to each GATE node and computes their RC delays. This analysis provides the following information for each signal:

- Circuit loading
- Slew rate

Examples

```
cckRCDlyPath 1 fanoutOnly=1 subckt=mux1
```

The previous example conducts static RC delay path analysis based on the generic algorithm (no static voltage initialization or specialized transistor conducting criteria are applied). It analyzes nodes that have a connection to the MOSFET gate or BJT base terminals. The reported path needs to be either partially or completely within the `mux1` subcircuit.

```
cckRCDlyPath 1 static_init=1 pmosOn=3.5 nmosOn=0.5 pwL_time=1n
```

Appendix I: HSIM CCK

Running Static RC Delay Analysis – Estimate Slew Rate

The previous example conducts the static RC delay path analysis by activating the static voltage initialization to propagate the constant `Vsrc` through qualified conducting devices, particularly for p-MOSFET which is in a conducting condition if its `Vg` has a constant voltage value and $Vg < \text{pmosOn}(3.5v)$. An n-MOSFET is in a conducting condition if its `Vg` has a constant voltage value and $Vg > \text{nmosOn}(0.5v)$.

Similarly, if there is a pwl or pulse `Vsrc`, CCK internally converts them into a constant `Vsrc` with the value based on the pwl(pulse) function at time=1 ns.

The initialization and voltage propagation procedure delivers a more practical device on/off landscape in the design. The analyzed delay path containing off device is considered a false path and is not reported.

Table 128 on page 1032 shows the mapping between the existing static delay analysis commands and the equivalent consolidated `cckRCDlyPath` commands.

Table 128 Mapping Static Delay Analysis and cckRCDlyPath Commands

Static Delay Analysis Command	Equivalent Consolidated cckRCDlyPath Command
<code>cckRCDlyPath 1</code>	<code>cckRCDlyPath 1</code>
<code>cckLimitRisePmosFallNmos 0 1</code>	<code>limitRisePmosFallNmos=0 1</code>
<code>cckRCFallDelay min=value1 max=value2 inside=0 1</code>	<code>fallDly='(value1 value2 0 1)</code>
<code>cckRCRiseDelay min=value1 max=value2 inside=0 1</code>	<code>riseDly='(value1 value2 0 1)</code>

cckDlyAtNode

Analyzes the nodes specified in the specified subcircuit and node.

Syntax

`cckDlyAtNode <subckt=s> <node=nd>`

Argument	Description
<code>subckt=s</code>	Specifies the subcircuit that contains nodes to analyze.

Argument	Description
node=nd	Specifies the node to analyze.

Examples

```
cckDlyAtNode node=xcpu.*
```

Analyzes only the nodes in the instance.

```
cckDlyAtNode subckt=xdp node=*
```

Analyzes only the nodes in the instances of the xdp subcircuit.

cckDlySkipElem

Directs cckRCDlypath to skip the elements specified in subcircuit, instance, and pattern.

Syntax

```
cckDlySkipElem <subckt=s> <inst=e> pattern=p
```

Argument	Description
subckt=s	Specifies the subcircuit that contains nodes to skip and not analyze.
inst=e	Specifies the node to skip and not analyze.
pattern=p	Specifies the element pattern to skip and not analyze.

Examples

```
CckDlySkipElem inst=xram.*
```

Analyses all the nodes except those in the xram instance.

cckDlySkipNode

Skips the nodes specified in the subcircuit and node.

Syntax

```
cckDlySkipNode <subckt=s> <inst=e>
```

Appendix I: HSIM CCK

Running Static RC Delay Analysis – Estimate Slew Rate

Argument	Description
subckt=s	Specifies the subcircuit that contains nodes to skip and not analyze.
inst=e	Specifies the node to skip and not analyze.

cckLimitRisePmosFallNmos

Controls path tracing.

Syntax

```
cckLimitRisePmosFallNmos [1|0]
```

Description

When the rising paths to VDD are traced, the default is to go through p-MOSFETs only. CCK ignores n-MOSFET elements, except for the transmission gates consisting of both p-MOSFET and n-MOSFET. When the falling path to GND is traced, the default is to limit the search within n-MOSFET elements only. Set `cckLimitRisePmosFallNmos` to 0 to change the default (1) and expand the search through both p-MOSFET and n-MOSFET.

cckRCFallDelay

Directs `cckRCDlyPath` to check the falling delay.

Syntax

```
cckRCFallDelay <min=dd3> <max=dd4> <inside=[0|1]>
```

Description

For falling paths, the following are reported to the `.cckfall` file:

- Paths to ground.
- Falling paths with a delay greater than `max` or less than `min`.

A report on paths with delays within the `<min, max>` ($=\langle dd3, dd4 \rangle$) range is obtained by setting `inside=1`.

cckRCRiseDelay

Directs `cckRCDlyPath` to check the rising delay.

Syntax

```
cckRCFallDelay <min=dd3> <max=dd4> <inside=[0|1]>
```

Description

For rising delay, the RC delays are computed along the transistor paths to VDD. The default is to report a rising path with a delay greater than `max` or smaller than `min`. If a report on rising paths with delays within a range such as `<min, max> (=<dd1, dd2>)` is required, set `inside=1`.

cckSetMosDir

Computes MOS delay and sets the signal or current flow direction through MOSFETs in order to perform other static and dynamic tests.

Syntax

```
cckSetMosDir [1|0]
```

Description

Since RC delay analysis is a static circuit analysis, the methods used to eliminate false paths are:

- Transistor direction
- Inverter relationship to trim the paths

The default direction of MOSFETs must be determined before tracing the paths to compute the delay such as `cckSetMosDir 1`. [Figure 111 on page 1035](#) shows the direction. Transistors with hard-to-find directions are set as undefined.

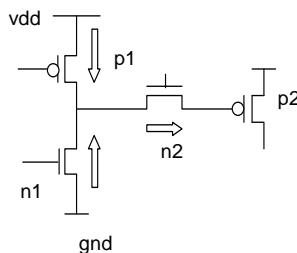


Figure 111 MOSFET Direction

In [Figure 111 on page 1035](#), the transistor paths are:

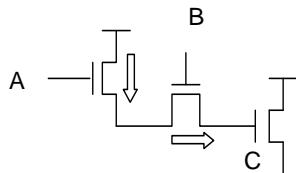
Appendix I: HSIM CCK

Running Static RC Delay Analysis – Estimate Slew Rate

- p-MOSFET (p1): Leaving VDD
- n-MOSFET (n1): Leaving GND
- Pass gate (n2): Entering the p2 gate

The paths may become invalid and ignored if they fall into one of the following criteria:

1. If two pass transistors are in the same subcircuit with an inverted relationship on the same path. [Figure 112 on page 1036](#) illustrates this condition.



If B is an inverted off^A then a path from vdd-> A-> B-> C is not valid. Ignore this path.

Figure 112 Transistors with an Inverted Relationship

If both pass transistors reside in the same subcircuit structure, and B is the reverse of A, then both transistors have an inverted relationship causing the path to be ignored

2. One or more pass transistors on the path is considered statically off.
For statically off pmos: if the pmos gate is connected to a constant voltage and its $V_g \geq \text{Max}(\text{constant voltage source in the circuit})$. If not applicable then the value of HSIMVDD is used as the threshold.
For statically off nmos: if the nmos gate is connected to a constant voltage and its $V_g \leq 0$.

Computing the Resistance of MOSFET

When CCK computes the resistance of a MOSFET it considers the parallel transistors and treats them as parallel resistors. This makes their effective resistance smaller in computing the delay. [Figure 113 on page 1037](#) illustrates a parallel transistor circuit.

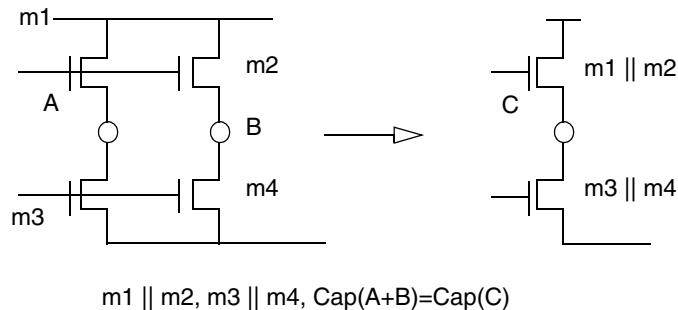
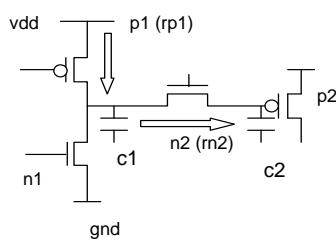


Figure 113 MOSFET Resistance

In [Figure 113 on page 1037](#), m1 and m2 are in parallel since they share the same input gate and drain node. m3 and m4 are also parallel. In addition, CCK also computes the effective capacitance at C. The capacitance value of C is the sum of the capacitance of nodes A and B.

Rising and Falling Path Delays

Based on the Elmore Delay Model, the rising and falling delays are computed as illustrated in [Figure 114 on page 1037](#) and [Figure 115 on page 1038](#).

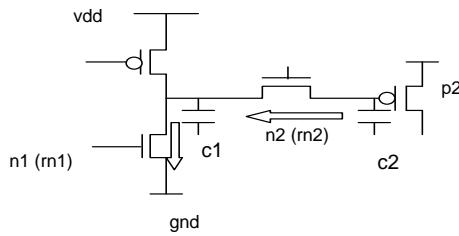


$$\text{rising delay} = rp1 * (c1 + c2) + rn2 * c2$$

Figure 114 Rising Delay

Appendix I: HSIM CCK

Running Static RC Delay Analysis – Estimate Slew Rate



$$\text{falling delay} = rn1 * (c1 + c2) + rn2 * c2$$

Figure 115 Falling Delay

Explanation of Rising Path Report

Rising delay paths are sorted with descending delay values. Each rising path indicates the following:

- Ending node
- Node capacitance
- Rising delay to ending node

CCK lists the elements on a path from VDD to the ending node and the following element characteristics are printed:

- Element name
- Element type
- Transistor length
- Transistor width

If there is a parallel element, an asterisk (*) is printed before the element name.

Examples

```
cckRCDlyPath      1
cckRCRiseDelay    min=0.8n max=3.e-9
cckRCFallDelay    min=0.4n max=3.2e-9
cckLimitRisePmosFallNmos1
cckSetMosDir      1
```

This check finds rising/falling paths with the following delays:

- Rising: Delays > 3 ns or < 0.8 ns
- Falling: Delays > 3.2 ns or < 0.4 ns.

Example 91 Rising Delay

```

; Path format
; Serial_num Node_name Node_capacitance(pf) Rise_time(ns)
;           Elem_name Elec_type L/W or Resistance

Total number of paths=2

User defined value:
    rise_time: Ton-Time < 0.8ns || 3ns < Ton-Time

S.N.  Node_name C(pf)Rise_time(ns)
1     Node: xic.xib.fr880.1164.37
        xi10.mp1pmos2.6/8.2
        xi32.mp1pmos2.2/8.2
        * xi32.mn1nmos1.55/3.1
        x88inst.mi22pmos1.6/8.2

2     Node: xiaba.xi27.w160b0.1750.68
        xic.xiba.xib.mi49pmos1.8/8
        xic5m.ri50res38.4

```

Referring to line item 1 in [Example 91](#), rising paths are sorted with the descending delay values:

Node: xic.xib.fr88 0.116 4.37

This means that node xic.xib.fr88 has 0.116 pF capacitance. The rising delay from VDD to this node is 4.37 ns. The path is as follows:

```

xi10.mp1      pmos  2.6/8.2
xi32.mp1      pmos  2.2/8.2
* xi32.mn1    nmos  1.55/3.1
x88inst.mi22pmos 1.6/8.2

```

Note: When an asterisk (*) is the first character in the line, it indicates it is a parallel transistor.

From VDD, the path goes through a p-MOSFET xi10.mp1 with the following dimensions:

- Length: 2.6 um
- Width: 8.2 um

p-MOSFET xi10.mp1 goes through p-MOSFET xi32.mp1 and n-MOSFET parallel transistor xi32.mn1 with the following dimensions:

- Length: 2.2 um
- Width: 8.2 um

From p-MOSFET xi10.mp1, the path goes through n-MOSFET xi32.mn1 with the following dimensions:

Appendix I: HSIM CCK

Running Static RC Delay Analysis – Estimate Slew Rate

- Length: 1.55 um
- Width: 3.1 um

Note: These two transistors usually form a transmission gate.

From n-MOSFET xi32.mn1, the path goes through p-MOSFET x88inst.mi22 with the following dimensions:

- Length: 1.6 um
- Width: 8.2 um

The signal then reaches the ending node.

Referring to line item 2 in [Example 91](#), the second path goes through a p-MOSFET xic.xiba.xib.mi49 whose dimensions are:

- Length: 1.8 um
- Width: 8.0 um

After passing through p-MOSFET xic.xiba.xib.mi49, the path goes through a 38.4 ohm resistor to reach this ending node. The ending node capacitance is 0.175 pF and the rising delay is 0.68ns.

Example 92 Falling Delay

```
; Path format
; Serial_num Node_name Node_capacitance(pf) Fall_time(ns)
; Elel_name Elel_type L/W or Resistance
Total number of paths=2
User defined value:
fall_time: Toff-Time < 0.4ns || 3.2ns < Toff-Time

      S.N.    Node_name        C(pf)    Fall_time(ns)
1 Node:  xiba_out          0.150     4.15
           xi3.mn1          nmos      0.55/6.3
           xi25.mn1          nmos      0.55/11.58
2 Node:  xic5m2.i12_out    0.150     3.65
           xc.xi99.xi11.mn1  nmos      0.55/6.3
           xc.xim4.xi10.mn1  nmos      0.55/11.58
```

Falling delay paths in [Example 92](#) are sorted with descending delay values similar to the rising delay report. Each falling path indicates the following:

- Ending node
- Node capacitance
- Falling delay to ending node

CCK lists the elements on a path from GND to the ending node and the following element characteristics are printed:

- Element name
- Element type
- Transistor length
- Transistor width

If there is a parallel element, an asterisk (*) is printed before element name.

The first path ending node is xiba_out) and has a capacitance of 0.15 pF.

Starting from the GND node, the element is traced through n-MOSFET xi3.mn1) with the following dimensions:

- Length: 0.55 um
- Width: 6.3 um

From n-MOSFET xi3.mn1), the path goes through n-MOSFET xi25.mn1) with the following dimensions:

- Length: 0.55 um
- Width: 11.58 um

Running a Dynamic Device Voltage Check

A single command `.tcheck` followed by numerous parameters is used to check device voltages during the simulation. When a condition is met, the device voltage is recorded and reported later. These commands are stored in a command file, which is invoked by adding the following to an input file of HSIM:

```
.param hsimDeviceV=dev_v_file
.param hsimDeviceV=dev_v_file
```

tcheck mosv

Checks the MOSFET device voltage.

Syntax

```
.tcheck tag_name mosv <model=name> <subckt=subckt_name>
<mos=inst_name> <lvgd=inst_name> <uvgd=val> <lvds=val>
<uvds=val> <lvdb=val> <uvdb=val> <lvgs=val> <uvgs=val>
<lvgb=val> <uvgb=val> <lvsb=val> <uvsb=val> <minL=val>
<maxL=val> <cond='expression'> <report=#> <time=val>
```

Appendix I: HSIM CCK

Running a Dynamic Device Voltage Check

```
<parallel=0|1> <separate_file=0|1>
<sort=tag|t1|el_name|node1|node2|err_v>
<start=time> <stop=time> <step=time>
```

Argument	Description
tag_name	Specifies a label in the report file to make searching multiple checking results easier.
mosv	Specifies MOSFET node voltages checking.
model=name	Specifies the model to be checked.
subckt=subckt_name	Specifies the subcircuit to be checked.
mos=inst_name	Specifies the instance to be checked.
lvgd=val	Reports an error to the output file when MOSFET vgd is less than the value specified in lvgd=val. vgd is the voltage value between gate and drain.
uvgd=val	Reports an error to the output file when MOSFET vgd is greater than the value specified in uvgd=val. vgd is the voltage value between gate and drain.
lvds=val	Reports an error to the output file when MOSFET vds is less than the value specified in lvds=val. vds is the voltage value between drain and source.
uvds=val	Reports an error to the output file when MOSFET vds is greater than the value specified in uvds=val. vds is the voltage value between drain and source.
lvdb=val	Reports an error in the output file when MOSFET vdb is less than the value specified in lvdb=val. vdb is the voltage value between drain and bulk.
uvdb=val	Reports an error in the output file when MOSFET vdb is greater than the value specified in uvdb=val. vdb is the voltage value between drain and bulk.
lvgs=val	Reports an error in the output file when MOSFET vgs is less than the value specified in lvgs=val. vgs is the voltage value between gate and source.
uvgs=val	Reports an error in the output file when MOSFET vgs is greater than the value specified in uvgs=val. vgs is the voltage value between gate and source.
lvgb=val	Reports an error in the output file when MOSFET vgb is less than the value specified in lvgb=val. vgb is the voltage value between gate and bulk.
uvgb=val	Reports an error in the output file when MOSFET vgb is greater than the value specified in uvgb=val. vgb is the voltage value between gate and bulk.
lvsb=val	Reports an error in the output file when MOSFET vsb is less than the value specified in lvsb=val. vsb is the voltage value between source and bulk.

Argument	Description
<code>uvsb=val</code>	Reports an error in the output file when MOSFET <code>vsb</code> is greater than the value specified in <code>uvsb=val</code> . <code>vsb</code> is the voltage value between source and bulk.
<code>minL=val</code>	Defines the minimum MOSFET length that <code>mosv</code> checks. The default value is 0.
<code>maxL=val</code>	Defines the maximum MOSFET length that <code>mosv</code> checks. The default value is infinite.
<code>cond='expression'</code>	Specifies a user-defined conditional expression to control checking.
<code>report=#</code>	Specifies how many times elements with errors are reported for the associated tag. If not limited, CCK reports all violations by default.
<code>time=val</code>	Specifies the time duration that the checking condition sustains before triggering an error report.
<code>parallel=0 1</code>	Reduces the number of errors reported for parallel devices.
<code>separate_file=0 1</code>	Reports errors in a separate file.
<code>sort=tag t1 el_name node1 node2 error</code>	Determines how to sort output. Table 129 shows the sorting keys and their tag names.
<code>r_v</code>	
<code>start=time</code>	Specifies the time span in which to perform checking.
<code>stop=time</code>	
<code>step=time</code>	Specifies the time step to use for checking.

Description

You can use any combination of the `mos`, `model`, or `subckt` arguments to specify the elements to be checked. The output of a device voltage check is reported in either of the following files:

- `hsim.mosv`
- `out_file.mosv` (if `hsim -o out_file` is used)

[Table 129](#) shows the sorting keys for the `sort` argument.

Table 129 Output Sorting Keys

Parameter	Description
<code>tag</code>	Tag Name

Appendix I: HSIM CCK

Running a Dynamic Device Voltage Check

Table 129 Output Sorting Keys

Parameter	Description
t1	The time an error occurs
el_name	Element name
node1	First node name
node2	Second node name
err_v	Error voltage value

Note that the following precedence rules apply to the `start`, `stop`, and `step` options:

1. When you use the `start` option, CCK checks if there is an existing time window object, (`start`, `step`, `stop`). If not, CCK generates a new time window and assigns the `start` time. The default `stop` time is the maximum simulation time and the default `step` time is 0. Otherwise CCK replaces the original `start` time.
2. When you assign the `stop` option, CCK checks if there is an existing time window object (`start`, `step`, `stop`). If not, CCK generates a new time window and assigns the `stop` time. The default `start` time is 0 and the default `step` time is 0. Otherwise CCK replaces the original `stop` time.
3. When you assign the `step` option, CCK checks if there is an existing time window object (`start`, `step`, `stop`). If not, CCK generates a new time window and assigns the `step` time. The default `start` time is 0 and the default `stop` time is the maximum simulation time. Otherwise CCK replaces the original `step` time.
4. After these three steps CCK checks the time window objects. Once there is no `step` option assigned (0) in the time window object, CCK assigns the `step` time as "`MAX(1n, (stop-start)/100)`". If CCK assigns the `step` time, it adds a (`step set by prog`) indicator in the header of the violation report.

Based on these rules, `tcheck` supports multiple time windows.

Note: If the time window expression contains a single start followed by multiple step or stop specifications, only the last step or stop value is considered and the entire expression is considered as a single time window.

Examples

```
.tcheck tag1 mosv lvgs=1
```

Device voltages with a gate-to-source voltage difference less than 1 V contain a tag1 prefix in the hsim.mosv file. For example, the hsim.mosv file has entries similar to those given below:

```
tag1: 1 40.00n 40.91n n lvgs x4.x3.mn2 b[0] x4.bn 0.50 1.00
tag1: 2 53.00n 60.91n n lvgs x4.x3.mn2 b[0] x4.bn 0.70 1.00
```

```
.tcheck tag1 mosv model=nch lvgs=1 uvgs=5 lvgb=3 uvds=10
```

Checks all the MOSFETs using the nch model. An error message is reported to one of the following files:

- hsim.mosv
- out_file.mosv

when any of the following conditions is met:

- vgs < 1 V
- vgs > 5 V
- vgb < 3 V
- vds > 10 V

```
.tcheck t1 mosv model=nch cond='(vgs < -3 || vgs > 7) || (vdb < 1 && vdb > -1)'
```

Checks every n-MOSFET to see if its length is within [1.1 micron, and 5.0 microns]. If the length is within these constraints, CCK continues to check vgs and vgd to see if vgs <-1 or vgs > 1 or vgd <-1 or vgd > 2.

```
.tcheck 7 mosv model=p cond='(l <=2e-6 && vds > 7) || (l > 2e-6 && vds > 10)'
```

Checks whether vds is greater than 7 V when MOSFET length is less than or equal to 2 microns, or if vds is greater than 10 V when MOSFET length is greater than 2 microns.

```
.tcheck tag mosv model=p minL=0 maxL=2e-6 uvds=7
.tcheck tag mosv model=p minL=2e-6 maxL=3e-6 uvds=8
.tcheck tag mosv model=p minL=3e-6 uvds=10
```

Appendix I: HSIM CCK

Running a Dynamic Device Voltage Check

The voltage difference depends on the transistor length. For example, if a transistor length is smaller than or equal to 2 um, the voltage difference between drain and source cannot be larger than 7 V. Otherwise, a warning is issued. If the length is between 2 um and 3 um, the max vds is 8 V. If the length is larger than 3 um, the maximum vds is 10 V. minL and maxL specify the range of transistor length.

Voltage and transistor length constraints are specified in one conditional expression, as shown in the example below.

```
.tcheck 7 mosv model=p cond='(l <=2e-6 && vds > 7) ||  
(l > 2e-6 && l <=3e-6 && vds > 8) ||  
(l > 3e-6 && vds > 10)'
```

The previous expression requires that when the length is less than 2 um, and vds is greater than 7 V, or if the length is greater than 3 um and vds is greater than 10 V, a warning is issued.

```
.tcheck tag1 mosv subckt=inv lvgs=1
```

Checks all the MOSFETs inside the inv subcircuit. If there are two inv instances in the design, this check examines every MOSFET inside those two inv instances. lvgs is defined as the lower bound of the voltage difference between a gate and a source.

```
.tcheck tag1 mosv mos=x1.* lvgs=1
```

Checks all the MOSFETs inside the x1 instance.

```
.tcheck tag1 mosv model=nch lvgs=1
```

Check all the MOSFETs using nch as a model.

```
.tcheck tag1 mosv mos=x1.* model=nch lvgs=1
```

Checks all the MOSFETs inside the x1 instance using nch as a model.

```
.tcheck tag1 mosv subckt=inv model=nch lvgs=1
```

Checks all the MOSFETs inside the inv subcircuit using nch as a model.

```
.tcheck tag1 mosv mos=x1.* lvgs=1 report=2
```

Checks all the MOSFETs inside x1. If vgs is less than 1 V an error is reported to either the hsim.mosv or out_file.mosv file. Only the first two errors are reported for tag1, even if there are additional errors.

```
.tcheck tag1 mosv mos=x1.* lvgs=1 time=5n
```

Checks all the MOSFETs inside x1. If vgs is less than 1 V for more than 5 ns, an error is reported to either the hsim.mosv or out_file.mosv files.

```
.tcheck tag1 mosv mos=x1.* lvgs=1 parallel=1
```

Checks all the MOSFETs inside x1. If vgs is less than 1 V, an error is reported to either the `hsim.mosv` or `out_file.mosv` file. Only one error is reported if there are parallel devices having the same error. An asterisk (*) is added after the element name to indicate there are parallel MOSFETs.

```
.tcheck tag2 mosv mos=x1.* lvgs=1 separate_file=1
```

Errors for this check go to `hsim.tag2` or `out_file.tag2`.

```
.tcheck tag2 mosv mos=x1.* lvgs=1 sort=node1|tag|err_v
```

The error output is sorted in the following order:

1. Node name
2. Tag name
3. Error voltage

```
.tcheck tag2 mosv mos=x1.* lvgs=1 start=10n stop=20n start=100n
```

In the previous example, checking is between 10 ns and 20 ns and repeated after 100 ns.

```
.tcheck tag2 mosv mos=x1.* lvgs=1 start=10n stop=120n step=30n
```

Checking starts at 10 ns and is repeated every 30 ns thereafter until 120 ns. You must use the `start` and `stop` arguments with the `step` argument.

Note: Time duration is not tracked when using `step` if the condition is true. Only a particular time point is reported. Therefore, time parameter is ignored.

tcheck diodev

Performs a diode device voltage check that is very similar to `.tcheck mosv`.

Note: Conditional expressions are not supported in `diodev`.

Syntax

```
.tcheck tag_name diodev <model=name> <subckt=subckt_name>
    <diode=inst_name> <lvac=val> <uvac=val> <report=#>
    <time=val> <parallel=0|1> <separate_file=0|1>
    <sort=tag|t1|el_name|node1|node2|err_v> <start=time>
    <stop=time> <step=time>
```

Appendix I: HSIM CCK

Running a Dynamic Device Voltage Check

Argument	Description
tag_name	Specifies a label in the report file to make searching multiple checking results easier.
diodev	Specifies diode voltage checking.
model=name	Specifies the model to be checked.
subckt=subckt_name	Specifies the subcircuit to be checked.
diode=inst_name	Specifies the diode instance to be checked.
lvac=val	When diode vac (the voltage value between anode and cathode) is less than the value specified in lvac, CCK reports an error to the output file.
uvac=val	When diode vac (the voltage value between anode and cathode) is greater than the value specified in uvac, CCK reports an error to the output file.
report=#	Specifies how many times elements with errors are reported for the associated tag. If not limited, CCK reports all violations by default.
time=val	Specifies the time duration that the checking condition sustains before triggering an error report.
parallel=0 1	Reduces the number of errors reported for parallel devices.
separate_file=0 1	Reports errors in a separate file.
sort=tag t1 el_name node1 node2 error_v	Determines how to sort output. Table 129 shows the sorting keys and their tag names.
start=time	Specifies the time span in which to perform checking.
stop=time	
step=time	Specifies the time step to use for checking.

tcheck capv

Performs a capacitor device voltage check.

Note: Conditional expressions are not supported in diodev.

Syntax

```
.tcheck tag_name capv <subckt=subckt_name> <cap=inst_name>
<lvpn=val> <uvpn=val> <report=#> <time=val>
<separate_file=0|1>
<sort=tag|t1|el_name|node1|node2|err_v> <start=time>
<stop=time> <step=time>
```

Argument	Description
tag_name	Specifies a label in the report file to make searching multiple checking results easier.
capv	Specifies capacitor device voltage checking.
subckt=subckt_name	Specifies the subcircuit to be checked.
cap=inst_name	Specifies the capacitor device instance to be checked.
lvpn=val	Check if the absolute voltage value between positive and negative nodes is less than the value specified in lvpn and reports any errors in the output file.
uvpn=val	Check if the absolute voltage value between positive and negative nodes is less than the value specified in uvpn and reports any errors in the output file.
report=#	Specifies how many times elements with errors are reported for the associated tag. If not limited, CCK reports all violations by default.
time=val	Specifies the time duration that the checking condition sustains before triggering an error report.
separate_file=0 1	Reports errors in a separate file.
sort=tag t1 el_name node1 node2 err_v	Determines how to sort output. Table 129 shows the sorting keys and their tag names.
start=time	Specifies the time span in which to perform checking.
stop=time	
step=time	Specifies the time step to use for checking.

Appendix I: HSIM CCK

Running a Post-process Device Voltage Check

Running a Post-process Device Voltage Check

The post-process device voltage check is accomplished with two methods.

Method 1

The first method for post-process device voltage check uses the following steps:

1. Add a device voltage check command in any of input file using the following syntax:

```
.param hsimDeviceV=cmdFile
```

2. Add the following syntax to the cmdFile:

```
.tcheck post=1
```

A command file example for post-process voltage check is as follows:

```
.tcheck post=1
.tcheck jt11 mosv model=pch mos=xram* lvdb=-1.6 uvdb=-1
.tcheck jt12 mosv model=pch mos=xram* lvgs=-0.6 uvds=1
.tcheck jt13 mosv model=nch mos=* cond='(vds < -1.0 || vgs < -0.75)'
```

3. Run HSIM. An FSDB file is created such as hsim.fsdb. The file contains all the print nodes and all the nodes required in comparing device voltage.

4. Run HSIM with a new option:

```
-post_devv fsdbFile
```

HSIM reads in netlist and device voltage commands. It then reads the nodes needed for device voltage comparison from the FSDB and outputs the results.

Note: Partial results are written throughout the process.

Method 1 Example:

Run hsim -o pp. A pp.fsdb file is created.

Run hsim -o pp1 -post_devv pp.fsdb. Signals are read from the fabd file and compared.

Method 2

The second method for post-process device voltage check uses the following steps:

1. Add a device voltage check command in any of input file using the following syntax:

```
.param hsimDeviceV=cmdFile
```

2. Add the following syntax to the cmdFile:

```
.tcheck post=2
```

When post=2 appears, the voltage continues to be read in the fsb file using the same process. The following is a command file example for post-process voltage check:

```
.tcheck post=2
.tcheck jt11 mosv model=pch mos=xram* lvdb=-1.6 uvdb=-1
.tcheck jt12 mosv model=pch mos=xram* lvgs=-0.6 uvds=1
```

3. Run HSIM – an FSBD or WDF waveform file like hsim.fsdb is created. When simulation is finished, it automatically uses the created waveform file to perform the post-process voltage check.

Note: When running post=2, HSIM automatically performs the post-process device voltage check after the waveform file is created.

Running Signal Integrity Checks

These CCK commands are designed to help check for static and dynamic crosstalk and noise-sensitivity estimation. These commands are presented in alphabetic order to make it easier to find them.

Dynamic crosstalk refers to the parasitic effects which lead to distortion of digital signals in the post-layout netlist. The effect is measured according to user-specified thresholds for change of characteristics of the signals. Violations are reported in the CCK result file.

To illustrate the usage concept, consider a digital signal at a node of interest in [Figure 116 on page 1052](#). There are two overlapped waveforms of the signal. The leading one is the waveform in the pre-layout netlist. The second is the waveform found in the post-layout netlist. It somewhat differs from the first due

Appendix I: HSIM CCK

Running Signal Integrity Checks

to parasitic effect. Refer to [Figure 116 on page 1052](#).

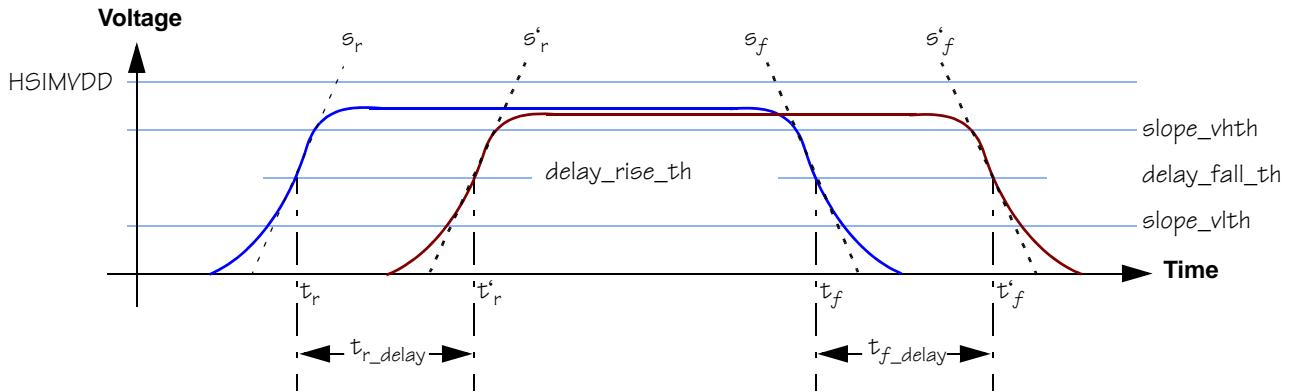


Figure 116 Pre-Post-layout Waveform Node Differences

cckDXtalk

Performs dynamic crosstalk analysis.

Note: The values or numbers specified in the cckDXtalk syntax are the default values if you do not specify corresponding values.

Syntax

```
cckDXtalk [ccFile=<ccfile_name>]
[noccFile=<noccfile_name>]
[nodeListFile=<nodeList_name>] [spfFile=<spf_file_name>]
[node=<node_name>] * [skipnode=<nodeName>] *
[slope_rel=10] [slope_abs=1e-9] [slope_vhth=80]
[slope_vlth=20] [delay_rise_th=50] [delay_rise_abs=1e-9]
[delay_fall_th=50] [delay_fall_abs=1e-9]
[separate_file=[0|1]] [extract_signals=[0|1]]
[bc_wc=[0|1]] [start] [stop]
```

Argument	Description
ccFile=<ccfile_name>	Specifies the post-layout fstdb file name. See the Crosstalk Checking Methods on page 1055 section.

Argument	Description
<code>noccFile=<noccfile_name></code>	Specifies the pre-layout fsdb file name.
<code>nodeListFile=<nodeList_name></code>	Specifies a file that contains a list of node names to analyze.
<code>spfFil=<spf_file_name></code>	Specifies the \SPF file name.
<code>node=<node_name></code>	Specifies the node names to analyze.
<code>skipnode=<nodeName></code>	Specifies the nodes to skip and not analyze.
<code>slope_rel=10</code>	Specifies the relative slope in percentage. The default is 10%.
<code>slope_abs=1e-9</code>	Specifies the absolute slope. The default is 1. Units are in V/ns.
<code>slope_vhth=80</code>	Specifies the high-threshold voltage of output. The default is 80, meaning that 80% of HSIMVDD.
<code>slope_vlth=20</code>	Specifies the low-threshold voltage of output. The default is 20, meaning that 20% of HSIMVDD.
<code>delay_rise_th=50</code>	Specifies the rising threshold in percentage. The default is 50, which means 50% of HSIMVDD.
<code>delay_rise_abs=1e-9</code>	Specifies the absolute value of rise time. The default is 1ps. Units are in seconds.
<code>delay_fall_th=50</code>	Specifies the falling threshold in percentage. The default is 50, which means 50% of HSIMVDD.
<code>delay_fall_abs=1e-9</code>	Specifies the absolute value of fall time. The default is 1ps. Units are in seconds.
<code>separate_file=[0 1]</code>	Reports errors in a separate file.
<code>extract_signals=[0 1]</code>	If set to 1, extract signals that violate cckDXtalk rules from ccFile and noccFile into a separate fsdb file. The default is 0.
<code>bc_wc=[0 1]</code>	If set to 1, report best/worst violations of a signal in the simulations. The default is 0.
<code>start</code>	Specifies the start time for dynamic cross talk check.

Appendix I: HSIM CCK
Signal Edge Characteristics

Argument	Description
stop	Specifies the stop time for dynamic cross talk check.

Signal Edge Characteristics

The signal edge characteristics for the waveform shown in [Figure 116 on page 1052](#) are described in the following sections.

Thresholds

`delay_rise_th` and `delay_fall_th`

Thresholds for the rising and falling edges. Normally these two values are 50% of VDD. Their simulation time at the point where the signal crosses the thresholds are tr and tf .

`slope_vhth` and `slope_vlth`

Defines the signal edges. The default value of `slope_vhth` is 80% of VDD and 20% of VDD for `slope_vlth`.

Rising Slope

`srising`

The positive average slope of a signal crossing from `slope_vlth` to `slope_vhth` as shown by the following syntax:

```
srising=(slope_vhth-slope_vlth)/(tr_end-tr_start)
```

Falling Slope

`sfalling`

The negative average slope of a signal crossing from `slope_vhth` to `slope_vlth` as shown by the following syntax:

```
sfalling=(slope_vlth-slope_vhth) (tf_end-tf_start)
```

Validation

Validation is measured based on four criteria:

Rising Edge Delay

Rising edge delay measures the time delay of a rising edge. It measures the difference of tr, that is, tr_delay between pre-layout and post-layout waveforms. If the difference, in second, is greater than the value specified by delay_rise_abs, a warning is issued into the result file.

Falling Edge Delay

Falling edge delay measures the time delay of a falling edge. It measures the difference of tf, that is, tf_delay between pre-layout and post-layout waveforms. If the difference, in second, is greater than the value specified by delay_fall_abs, a warning is issued into the result file.

Slope Relative Error

Slope relative error measures the relative error of rising/falling slopes between pre-layout and post-layout waveforms. The error is calculated using the formula $(\text{slope}_{\text{post_layout}} - \text{slope}_{\text{pre_layout}}) * 100\%$. If the absolute value of the error is greater than the value specified by slope_rel, a warning is issued into the result file.

Slope Absolute Error

Slope absolute error measures the difference of rising/falling slopes between pre-layout and post-layout waveforms. If the difference, in V/ns, is greater than the value specified by slope_abs, a warning is issued into the result file.

Crosstalk Checking Methods

Various methods of executing dynamic crosstalk checking are:

Method 1

The first method of executing dynamic crosstalk checking is to execute HSIM separately. Using this approach, simulation results must be prepared for both pre-layout and post-layout netlists in advance. The result files are denoted as noccFile.fsdb and ccFile.fsdb respectively. The nodes of interest have to be output using .print command for both cases.

Appendix I: HSIM CCK
Crosstalk Checking Methods

The analysis needs an optional node list file such as nodelist.txt. The node list file is a text file containing a list of analysis nodes itemized one node name per line. If the file is not given, the node specifier in `cckDXtalk` specifies which nodes are to be analyzed.

Dynamic crosstalk analysis is eventually driven by adding `cckDXtalk` into the CCK command file of the netlist. This is either a pre-layout or post-layout netlist file and run HSIM with this revised netlist. [Figure 117 on page 1056](#) shows the detailed analysis flow.

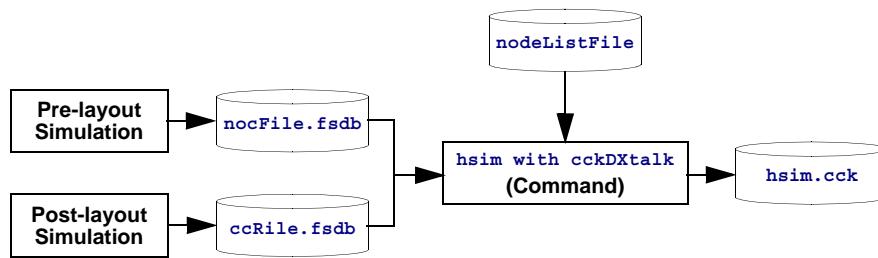


Figure 117 Dynamic Cross Talk Detailed Analysis Flow

Method 2

The second method of executing dynamic crosstalk checking is to run `cckDXtalk` in batch mode. Using this flow, only `spfFile` must be specified while not specifying `ccFile` and `noccFile`. The command automatically runs HSIM twice.

- The first run simulates the pre-layout netlist.
- The second run simulates the post-layout netlist with the specified SPF file and carries out the analysis.

Nodes found that are to be analyzed are automatically added to output files as shown in the following examples:

Examples

When `ccFile` and `noccFile` are both given, add `cckDXtalk` to the CCK command file using the following syntax:

```
cckDXtalk ccFile=post.fsdb noccFile=pre.fsdb nodeListFile=nodelist.txt \
delay_rise_th=45 delay_rise_abs=1.12n \
slope_vlth=14 slope_vhth=80 slope_abs=1 slope_rel=10 \
delay_fall_th=60 delay_fall_abs =0.001n \
separate_file=1
```

Following is a typical report sample from cckDXtalk:

```
* -----
* Dynamic Cross Talk Check
* noccFile = pre.fsdb
* ccFile = post.fsdb
* nodeListFile = nodelist.txt
* referenced vdd = 1.5 V
* slope_rel = 10%
* slope_abs = 1 V/ns
* slope_vlth = 14%
* slope_vhth = 80%
* delay_rise_th = 55%
* delay_rise_abs = 1 ns
* delay_fall_th = 60%
* delay_fall_abs = 0.001 ns
* separate_file = 1
* -----
Signalsim time(ns) delay time(ns) slope rel(%) slope \
abs(v/ns)
v(up) 0.375- -28.3 3.69
v(up) 3.96 0.232(F) -47.7 8.73
v(up) 7.36 --34 4.49
v(up) 11 0.232(F)-47.7 8.72
v(out_b) 0.28 - -34 4.49
v(out_b) 2.71 0.232(R) -47.7 8.72
.....
.....
      find 24 rise delay violation(s).
      find 89 fall delay violation(s).
      find 174 absolute slope violation(s).
      find 174 relative slope violation(s).
Total of 437 dynamic cross talk violation(s).
Total of 6 node checked.
```

Using cckDXtalk when the spfFile is given using the batch mode method. When spfFile is given, add cckDXtalk into the CCK command file using the following syntax:

```
cckDXtalk spfFile=cc.spf nodeListFile=nodelist.txt \
delay_rise_th=45 delay_rise_abs=1.12n \
slope_vlth=15 slope_vhth=75 slope_abs=4 slope_rel=19 \
delay_fall_th=35 delay_fall_abs =3.12n \
separate_file=1
```

Appendix I: HSIM CCK

Static Crosstalk Noise Analysis: Estimating Noise Glitches

Static Crosstalk Noise Analysis: Estimating Noise Glitches

Due to the ever increasing density in a chip design, the coupling capacitors between signal nets is an important topic to study. Aggressor nets can induce noise on victim nets. If the noise glitch is too large, it triggers the circuit to change state unexpectedly. Furthermore, with the decrease of power supply voltage (VDD) such as from 3 V to 2.5 V to 1.8 V, the noise glitch requires more attention and needs to be addressed.

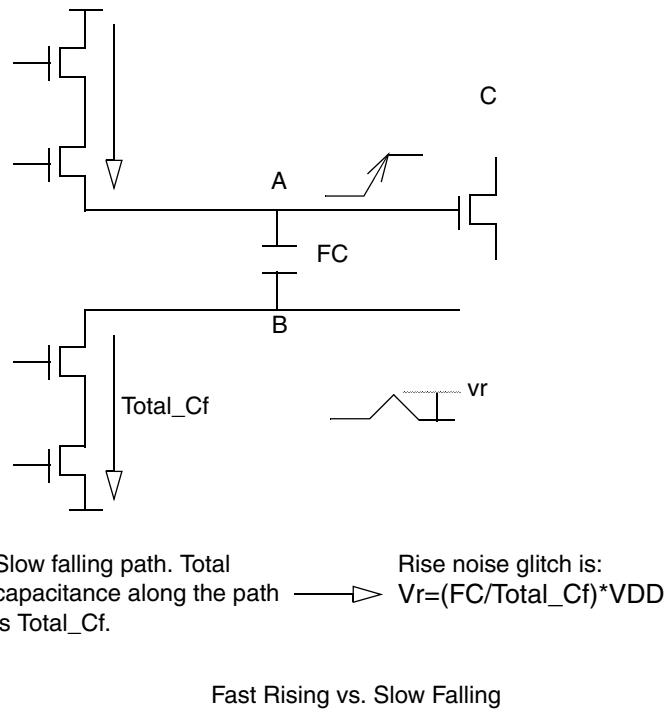


Figure 118 Fast Rising vs. Slow Falling Path]

If FC is a floating capacitor with a value exceeding a specific threshold; and from node A, the fast rising paths are located; from node B, slow falling paths are located. The fast rising at A creates a rising noise glitch at node B. The noise bump is approximated as $(FC / Total_Cf) * VDD$, where FC is the coupling capacitance; Total_Cf is the total capacitance on the slow falling path.

Similarly, fast falling paths and slow rising paths may exist at the two ends of a floating capacitor. The noise glitch on the slow path needs to be computed.

Note: When `cckStaticXtalk` is issued, CCK automatically disables HSIM transition simulation. This means that there is no transition simulation even if users have `.tarn` in the HSIM control file or netlist file.

slow rising path. total capacitance on the path is `Total_Cr`.

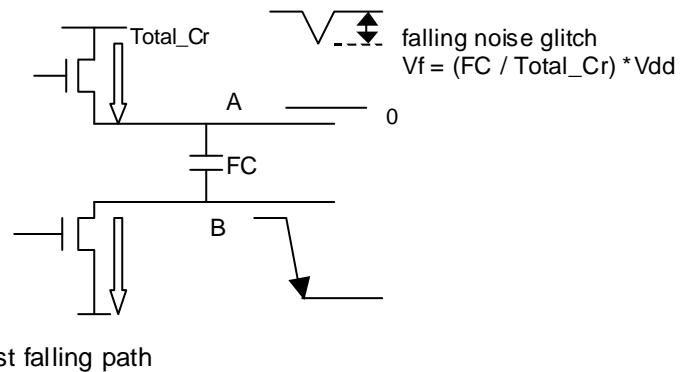


Figure 119 Fast Falling vs. Slow Rising Path

cckXtalk

Runs crosstalk glitch analysis.

Syntax

```
cckStaticXtalk fCapTh=fcap_threshold
    vrth=rise_noise_glitch_threshold
    vfth=fall_noise_glitch_threshold
    [method=1|2] [modelWLRatio=(model_name, min,
    max)] [RCTime=(riseMin, riseMax, fallMin,
    fallMax)] [fCapRptRatio= value] [rptAggrNode= node_name]
    [subckt=subcircuit_name] [node=node_name]
    [skipSub=subcircuit_name] [skipNode=node_name]
    [skipElem=element_name] [debug= 0|1] [victim='([subckt],
    [node])'* ... [skipVictim='([subckt], [node])'* ...
    [aggressor='([subckt], [node])'* ...
    [skipAggressor='([subckt], [node])'* ...
```

Appendix I: HSIM CCK

Static Crosstalk Noise Analysis: Estimating Noise Glitches

Argument	Description
fCapTh=fcap_thres hold	Floating capacitors with values larger than the specified threshold value are considered for crosstalk analysis. The value is in farads.
vrth=rise_noise_g litch_threshold	Reports nodes if they have rise glitch larger than the specified threshold value. The value is in volts.
vfth=fall_noise_g litch_threshold	Reports nodes if they have fall glitch larger than the specified threshold value. The value is in volts.
method=1 2	A value of 1 (default) identifies the fast/slow path by the transistor W/L ratio. Specify a value of 2 to identify the fast/slow path by the path RC delay. The method values are mutually exclusive in the same static crosstalk command set.
modelWLRatio=mode l_name min max	This argument applies only to method=1. For the specified model name, the path is a fast path if the smallest W/L ratio is greater than the specified maximum value. The path is a slow path if the greatest W/L ratio is less than the specified minimum value.
RCTime=riseMin riseMax fallMin fallMax	This argument applies only to method=2. The path RC delay calculation is based on the approach used by the cckRCDlyPath on page 1029 command. If the path delay is less than the specified minimum value, the path is treated as a fast path. If the path delay is greater than the specified maximum value, the path is treated as a slow path.
fCapRptRatio value	This argument works only with the rptAggrNote argument. If specified, the cckStaticXtalk command uses the fCapRpt ratio value ($0 \leq \text{value} \leq 1$) to launch a quick check of coupling capacitors only. The other end of coupling capacitors is reported when its $(\text{CC}/\text{C}_{\text{total}})$ is greater than the fCapRpt ratio value. If you do not use this argument, the coupling capacitors check does not run. There is no default.
rptAggrNode node_name	This argument works only with the fCapRptRatio argument. It reports the qualified aggressor signals across the related coupling capacitors through the specified nodes. You can use a wildcard character in the node name.
subckt=subcircuit _name	Specifies to conduct a static crosstalk analysis within the specified subcircuit domain.
node=node_name	Specifies a static crosstalk analysis on the specified nodes. You can use this argument with the subckt argument.
skipSub=subcircuit _name	Runs static crosstalk analysis without considering the nodes in the specified subcircuit domain.

Argument	Description
skipNode=node_name	Runs static crosstalk analysis without considering the specified nodes.
skipElem=element_name	Runs static crosstalk analysis without considering the specified elements.
debug 0 1	Specify a value of 1 to output all valid paths to a gzipped file with a .xtkdebug extension. The default is 0 (off).
victim='([subckt], [node])'	Limits the victim nodes in the same way as the scope option used by many other CCK commands, such as cckDiode on page 946 .
skipVictim='([subckt], [node])'	Excludes victim nodes from the victim options in the same way as the scope option used by many other CCK commands.
aggressor='([subckt], [node])'	Limits the aggressor nodes.
skipAggressor='([subckt], [node])'	Excludes aggressor nodes from the aggressor options.

Description

Wildcard support applies to node but not for subckt. Either sub-options (subckt and node) can be ignored. However, they cannot be ignored simultaneously. An ignored subcircuit implies searching nodes from the top, while an ignored node implies searching all nodes in the specified subcircuit.

Examples

```
cckStaticXtalk fCapTh=1p vrth=0.01 vfth=0.02 method=1 modelWLRatio=(nx, 12, 18)
modelWLRatio=(px, 16, 24)
```

In the previous example, the transistor W/L ratio determines the fast/slow paths in the design. The analysis only considers the capacitors larger than 1pF. A path with an nx type transistor with a minimum W/L ratio greater than 18 is most likely a fast path. A path with a px type transistor with a minimum W/L ratio greater than 24 is most likely a fast path. The `cckStaticXtalk` command collects and reports the violations with rise glitch greater than 0.01 v and fall glitch greater than 0.02 v.

```
cckStaticXtalk fCapTh=1p vrth=0.01 vfth=0.01 method=2 RCTime=(0.5n 3n 0.4n 4n)
```

In the previous example the path RC delay (RC time constant) determines the fast/slow paths in the design. The analysis only considers the capacitors larger

Appendix I: HSIM CCK

Static Crosstalk Noise Analysis: Estimating Noise Glitches

than 1 pF. The `cckStaticXtalk` command collects and reports the violations with rise glitch greater than 0.01 v and fall glitch greater than 0.02 v. A rising path with a delay less than 0.5 n is considered a fast path. If the delay is larger than 3 n, it is considered a slow path. A falling path with a delay less than 0.4 n is considered a fast path. If the delay is larger than 4 n, it is considered a slow path.

The following three files are created:

- `hsim.cckxtk` or `out_file.cckxtk`: These files show path-pairs on the interested floating capacitors. In each pair, one path is fast and the other is slow.
- `hsim.cckvr` or `out_file.cckvr`: Path-pairs that cause large rising noise glitch are reported in these files.
- `hsim.cckvf` or `out_file.cckvf`: These files contain path-pairs that cause large falling glitch, hence smaller vf.

The `hsim.cckxtk` output sample contains pairs of slow and fast paths as shown in the following:

```
; Report format:
;   Serial_num Capacitor_name Cap_1st_node_name Cap_2nd_node_name Cap_value \
;   minWL_mos_name_on_low_imp_path minWL_mos_name_on_high_imp_path
;   Low_imp_path:
;     Vsrc_node_name
;     Elec_name Elec_type L/W or Resistance
;   High_imp_path:
;     Vsrc_node_name
;     Elec_name Elec_type L/W or Resistance
Total number of low and high impedance path pairs for
floating capacitors=2912
User defined values:
Floating capacitance threshold=1e-012 (F)
Use MOSFET WL ratio to find low/high impedance paths.
nx      min=12, max=18
px      min=16, max=24
px2     min=10, max=20
1       c6x gx1 wl8 4.51e-012 mx7|mn1 mx55|mi30
low_imp_path:
  gnd
    mx7|mn1nmos0.4/5
  * mx7|xi38|mn1@3 nmos0.4/5
  * mx7|xi38|mn1@2 nmos0.4/5
  * mx7|xi38|mn1@5 nmos0.4/5
  * mx7|xi38|mn1@4 nmos0.4/5
high_imp_path:
  gnd
    mx55|mi30 nmos 0.4/3
```

Note: MOSFET mx7|mn1 has parallel transistors. The sum of their W/L ratios is documented. So the effective W/L is $(5 * 5/0.4) = 62.5 > (\max=18)$. It is a fast path.

The hsim.cckvf output sample is shown in the following:

```
Total number of Vf errors=115
User defined values:
Floating capacitance threshold:1e-012 (F)
Vf threshold:0.3 (V)
Use mos WL ratio to find low/high impedance paths.
nx: min=12, max=18
px: min=16, max=24
px2: min=10, max=20
S.N. Cap_name Node1 Node2 Ctotal(F) Cfloat(F) Vf(V) Low_imp_mos High_imp_mos
1 c201 xi5|io4 xi4|io4b 6.7e-012 2.3e-012 0.66 mx54|mi79 mx59|mi54
  low_imp_path:
    gnd
    mx54| mi79 nmos 0.4/25
  high_imp_path:
    vdd
    mx59|mi54 pmos 0.5/5
```

Note: The floating capacitor is c201. The total capacitance on the slow path is 6.7 pF. The falling glitch results in vf=0.66 V. The smallest MOSFET on the low impedance path is mx54|mi79. The smallest MOSFET on the high impedance path is mx59|mi54.

Example 93

```
cckXtalkFloatingCap      1.e-12
cckXtalkRiseVolt        0.01
cckXtalkFallVolt        0.3
cckXtalkByRC            1
cckXtalkRiseTimeConst   min=0.5n max=3n
cckXtalkFallTimeConst   min=0.4n max=4n
```

Use RC-delay to decide the fast and slow paths. The floating capacitors threshold is 1 pF. Any coupling capacitors value exceeding 1 pF is considered. From those capacitors' two ends, trace to VDD and GND to find rising and falling paths. For a rising path, if its RC-delay is smaller than 0.5 ns, it is a fast path. If its delay is larger than 3 ns, it is a slow path. This also applies to falling paths. Fast and slow path-pairs are located at each interested floating capacitor. Then, compute the coupling impact. If the rising noise glitch is larger than 0.01 V, this path-pair is reported in hsim.cckvr. If the falling glitch is large enough to make Vf smaller than 1.79V, then this path is in hsim.cckvf.

Floating capacitor c6x has two end nodes: gx1 and wl8. A fast falling path, whose MOSFET with smallest W/L is mx7|mn1, consists of parallel transistors.

Appendix I: HSIM CCK

Static Crosstalk Noise Analysis: Estimating Noise Glitches

For the high impedance path, its smallest MOSFET is mx55|mi30, whose W/L is 3/0.4 < (min=12).

The hsim.cckxtk output sample is shown in the following:

```
;      Report format:
;          Serial_num Capacitor_name Cap_1st_node_name Cap_2nd_node_name
Cap_value \
;      Time_constant_on_low_imp_path Time_constant_on_high_imp_path
;      Low_imp_path:
;          Vsrc_node_name
;          Elel_name Elel_type L/W or Resistance
;      High_imp_path:
;          Vsrc_node_name
;          Elel_name Elel_type L/W or Resistance
Total number of low and high impedance path pairs for floating capacitors=2647
User defined values:
Floating capacitance threshold1e-012(F)
Use RC delay to find low/high impedance paths.
Rise time-constant:min=5e-010, max=3e-009
Fall time-constant:min=4e-010, max=4e-009
S.N. Cap_name Node1     Node2     Cap_value(F)   Min_time_constant
Max_time_constant
1 c249 xite<0 rd<9 7.6e-012 3.9933e-010           1.43892e-008
low_imp_path:
    gnd
        mxer|xi8|mn1 nmos 0.4/10
high_imp_path:
    vdd
        mxer|xi9|mi54 pmos 0.5/20
        mxer|xi9|mi56 pmos 0.5/20
```

The hsim.cckvr output sample is shown in the following:

```
Total number of Vr errors=4
User defined values:
Floating capacitance threshold:1e-014(F)
Vr threshold:0.01(V)
Use RC delay to find low/high impedance paths.
Rise time-constant: min=5e-010, max=3e-009
Fall time-constant: min=4e-010, max=4e-009S.N. Cap_name Node1 Node2 Ctotal(F)
Cfloat(F) Vr(V) Min_time_constant Max_time_constant
1 c81 xik2|i5 xik4|io2b 5.87e-012 2.4e-012 0.2.42 3.68e-010 5.0e-009
    low_imp_path:
        gnd
        xi43|mi9 nmos 0.4/25
    high_imp_path:
        vdd
        xi13|mi5 pmos 0.5/5
```

The hsim.cckvf output sample is shown in the following:

```
Total number of Vf errors=1501 (1501)
User defined values:
Floating capacitance threshold:1e-012 (F)
Vf threshold:0.2 (V)
Use RC delay to find low/high impedance paths.
Rise time-constant: min=5e-010, max=3e-009
Fall time-constant: min=4e-010, max=4e-009
S.N. Cap_name Node1 Node2 Ctotal(F) Cfload(F) Vf(V) Min_time_constant
Max_time_constant
1 c1081 xik1|io2 xik1|io4b 5.487e-012 2.04e-012 0.73042 3.68e-010 5.0e-009
    low_imp_path:
        gnd
        xi313|mi79                                         nmos0.4/25
    high_imp_path:
        vdd
        xi313|mi54                                         pmos0.5/5
```

Note: The `cckStaticXtalk` command replaces the `cckStaticXtalk_GroupCmd` command. [Table 130](#) shows how the `cckStaticXtalk` command arguments map to the `cckStaticXtalk_GroupCmd` commands.

Table 130

cckStaticXtalk argument	Equivalent cckStaticXtalk_GroupCmd command
fCapTh	<code>cckXtalkFloatingCap</code>
vrth	<code>cckXtalkRiseVolt</code>
vftth	<code>cckXtalkFallVolt</code>
method	<code>cckXtalkByWL</code> and <code>cckXtalkByRC</code>
modeWLRatio	<code>cckXtalkModelWLratio</code>
RCTime	<code>cckXtalkRiseTimeConst</code> and <code>cckXtalkFallTimeConst</code>
fCapRptRatio	<code>cckXtalkReportCouplingCapRatio</code>
rptAggrNode	<code>cckXtalkReportAggressorNode</code>
subckt	<code>cckXtalkAtNode</code>
node	<code>cckXtalkAtNode</code>
skipSub	<code>cckXtalkSkipNode</code>
skipElem	<code>cckXtalkSkipElem</code>

Detecting Leakage Current

These CCK commands detect leakage-current paths and estimate the contribution of the nodes to the leakage current.

cckMaxStaticLeak

Reports any path leaking current from one voltage source to another source statically. For example, there is a conducting path in [Figure 120 on page 1066](#) from VDD to GND.

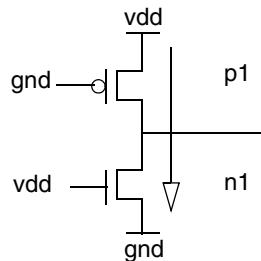


Figure 120 Conducting Path

Note: `cckMaxStaticLeak` is a legacy command to be phased out. Instead, apply the `cckStaticDCPath` feature as a static leakage path analysis solution, which accommodates technology to cope with complex modern design infrastructure.

Syntax

`cckMaxStaticLeak <num=n>`

Argument	Description
<code>num=n</code>	Limits the number of warnings in the <code>.cckleak</code> file to the specified value

Examples

`cckMaxStaticLeak num=10`

The following is the static leakage path between voltage sources output sample resulting from the command example given above:

```
*****
* Static leakage path between voltage sources
*****
A static leakage path to
  node (vdd, v=1.65)
  from node (rmaddr<63>) thru (xm.xyy2.mp)
  from node (0, v=0) thru (xm.rz)
```

cckOffLeakI

Detects the OFF transistors at the standby time and computes the leakage current.

Syntax

```
cckOffLeakI <tag=tagName> <subckt=subName> <inst=instName>
  <vsrvc=vsrvcName> <vt=threshold> <time=standByTime>
  <ioff=ioffCmd> <hzStage=hzLimit> <detail=[0|1]>
  <separate_file=[0|1]>
```

Argument	Description
tag=tagName	Prints the specified <i>tagName</i> in the report file at the beginning of a line to indicate a warning.
subckt=subName	Specifies a leakage current check for the specified subcircuit.
inst=instName	Specifies a leakage current check for the specified instance.
vsrvc=vsrvcName	Voltage source name from which OFF devices are traced.
vt=threshold	Voltage threshold used to decide if a device has a leakage current. The default is 0.1 V.
time=standByTime	Specifies the standby time when checking is performed.
ioff=ioffCmd	Specifies the file name that contains the current ratios per unit length for each model.
hzStage=hzLimit	A series of OFF devices make some nodes to be in high-impedance mode. In these cases, each OFF device has a high-z stage number, based on the distance from the non-high-z nodes. The default is 2.
detail=[0 1]	If set to 1, all the OFF transistors and the high-z stage for each instance are printed.
separate_file=[0 1]	If set to 1, the summary is printed to a separate file.

Appendix I: HSIM CCK

Detecting Leakage Current

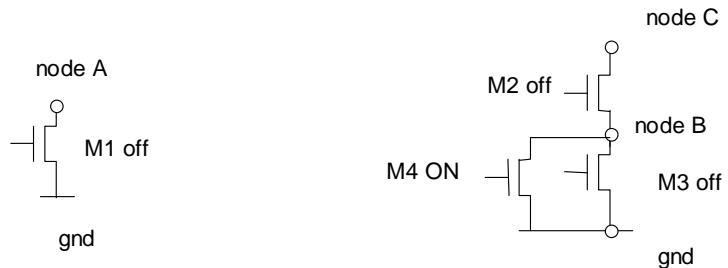
Description

The leakage current of an OFF device depends on its length, width, and state of its drain and source nodes. For example, if a node is in high impedance mode, each group is detailed in a summary report.

Leakage Current in OFF Transistor

The leakage current of an OFF device depends on the voltage of its drain (D) and source (S) node. If a trace is run from a given voltage source node to an OFF device and reaches the S node first, then if $V(D) > V(S) + vt$, this OFF device has a leakage current.

Tracing starts from the given voltage source specified in the command. For example in [Figure 121 on page 1068](#), a trace to node A is begun from a GND node with zero voltage, where M1 is OFF. Then the voltage at node A is verified. If $V_a > gnd + vt$, M1 has a leakage current.



if node A's voltage
 $V_a > V_{gnd} + vt$, M1 is
 considered to have leakage
 current.

M2 and M3 are off. Since M4 is
 ON, node B is not a high-
 impedance node. If the voltage of
 node B is greater than ($gnd + vt$),
 M3 has a leak. For M2, we
 compare the voltage difference
 between nodes C and B. if it is
 larger than vt , then M2 has a
 leakage.

Figure 121 OFF Device Leakage Current

From the GND node, OFF device M3 can be traced continuously until it reaches M2. To consider the leakage current in M3, the voltage at node B and GND are compared. If $V_b > gnd + vt$, M3 has a leakage. At this point, $V_b=0$ and $V_{gnd}=0$, so M3 has no leakage. For M2, which is also OFF, the voltages of nodes C and B are compared. If $V_c > V_b + vt$, M2 has a leakage.

If some nodes are in a high-impedance (high-Z) state, the effects on adjacent nodes must be considered. Since the voltage of a high-z node is unknown, tracing must be continued to find the node voltage of the next stage. Before proceeding, however, the high-z stage must be defined.

High-Z Stage Definition

A high-z stage is the distance from an OFF device with high-z node to a non-high-z node. [Figure 123 on page 1070](#) shows an example of a definition of high-z stage.

In [Figure 123 on page 1070](#), tracing starts at a given voltage source such as GND. For one branch, OFF devices such as M3, M2, and M1 are tested and stop at node A, which is VDD or non-high-z node. Another tracing starts from GND where two branches of OFF devices are found. One branch is M8 -> M7 -> M6 and the other is M8 -> M7 -> M4 -> M5. Nodes D and E are either power supply or non-high-z nodes.

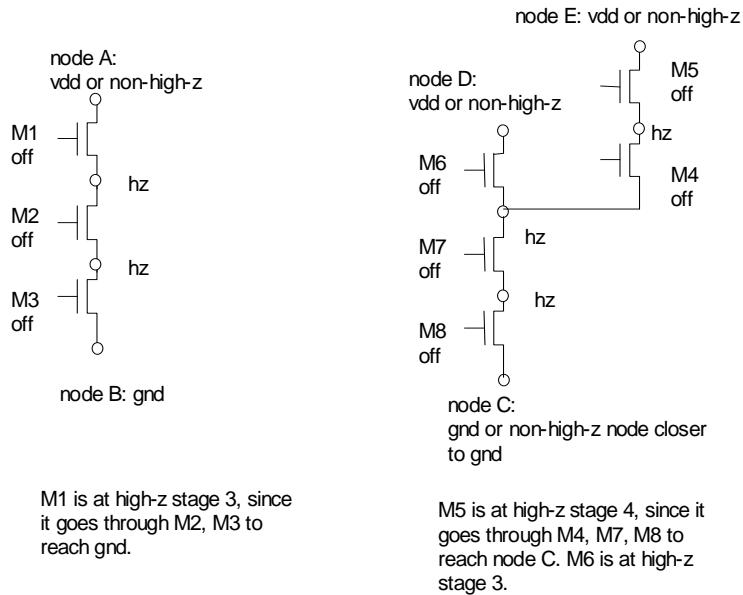


Figure 122 OFF Device Leakage Current with a High Impedance Node

Appendix I: HSIM CCK

Detecting Leakage Current

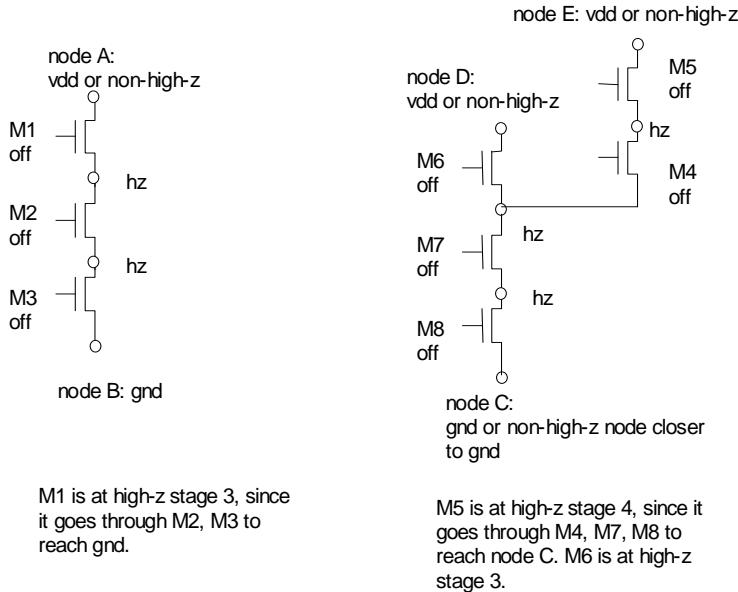


Figure 123 OFF Device Leakage Current with a High Impedance Node

When deciding the leakage current, only the top device in each branch is considered. In [Figure 123 on page 1070](#), only M1, M6, and M5 are considered since they are the last element in each branch starting from GND node.

[Figure 123 on page 1070](#) illustrates the following:

- M1 is at high-z stage 3
- M6 is at high-z stage 3
- M5 is at stage 4.

The leakage current at a large high-z stage is smaller than that at the small high-z stage as illustrated by the measurement. Therefore, the current ratios at different high-z stages need to be specified.

Leakage Current Ratio

The ratio in the current loffCmd file is the leakage current per unit width for each model and length.

For example, in the following command:

```
nch 0.18u 2e-4 z2 1.5e-4 z3 1e-4
```

nch is the model name. 0.18u is the length of the transistor. The leakage current per unit width is 2e-4 (Amp/meter) for an OFF device with high-z stage 0 or 1. If an OFF device is at high-z stage 2, the unit width leakage current is 1.5e-4 (Amp/meter). If a device is at high-z stage 3, then its leakage current is 1e-4 (Amp/meter). The higher the high-z stage, the less its leakage current.

- Determine which OFF devices in [Figure 123 on page 1070](#) have leaks:
 - In [Figure 123 on page 1070](#), for M1, nodes A and B are considered. The nodes between A and B are not considered since they are at high-z mode. Tracing begins from node B (that is, node B is closer to GND node). If $V_a > V_b + V_t$, where V_t is the threshold specified in the command, M1 has a leak. If M1 is a nch device of $L=0.18u$, its leakage current is the current ratio of z3 (1e-4 Amp/meter) multiplied by the width of M1.
 - For M6, nodes D and C are considered. If $V_d > V_c + v_t$, M6 has a leakage current. Otherwise, it does not have a leak. If M6 is a nch device with $L=0.25u$, then its leakage current is the current ratio z3 (0.5e-4 Amp/meter) multiplied by the width of M6.
 - For M5 at high-z stage 4, nodes E and C are considered. If $V_e > V_c + v_t$, M5 has a leak. If only the OFF device with maximum high-z stage 3 is to be considered, M5 is not considered. Now for the remaining OFF devices, for example, M4, since it is not the last element in the branch, it is considered to have no leakage.
- The total leakage current is I (leak in M1) + I (leak in M6).
- If the current ratio is not given, the total width of the OFF devices with the same model and length is added together.
- The length in the current ratio file may have a range. For example, 0.18u to 4u means the devices with length from 0.18u to 4u, inclusively.

Examples

Example 94

```
cckOffLeakI tag=t1 inst=* vsrc=0 vt=0.4 time=10n time=20n \
time=41n time=66n detail=1 separate_file=1 ioff=ioff.cmd \
hzStage=3
cckOffLeakI tag=t2 inst=* vsrc=vbb vt=0.4 time=6n time=8n \
time=15n time=56n detail=1 separate_file=1 ioff=ioff.cmd \
hzStage=3
```

Add the ioff.cmd syntax to the current ratio file as follows:

Appendix I: HSIM CCK

Detecting Leakage Current

```
/* leakage current for every size L
 * syntax:
 *
 * model_name L(meter) off_current(Amp/meter) highZCnt
 * off_current
 *
 * length unit=meter
 * current unit=Amp
 *
 * unit must be specified. for instance,
 * L=2u (2e-6 memter)
 * the unit for off_current ratio is Amp/meter
 * if length has range, use "to"
 */
nch 0.18u 2e-4 z2 1.5e-4 z3 1e-4
nch 0.19u to 4u 1e-4 z2 0.8e-4 z3 0.5e-4
pch 0.18u 1.5e-4 z2 1.1e-4 z3 0.7e-4
```

Producing the command output is accomplished as follows:

1. The options specified are listed in the command as follows:

```
*****
* tag=t1 vt=0.400 hzStage=3 separate_file=1 detail=1
* vsrc=0 time=10n time=20n time=41n time=66n
* inst=*
* ioff=ioff.cmd
* leakage current spec is:
*      model length offI(A/meter) highZ_stage offI(A/m)
*      pch 0.18u      0.00015 z2 0.00011 z3 7e-005
*      nch 0.19u to 4u 0.0001 z2 8e-005 z3 5e-005
*      nch 0.18u      0.0002 z2 0.00015 z3 0.0001
*****
```

2. All the OFF devices of model nch with length 0.18u are detected. These nch devices are further separated into different high-z stages.
 - For devices at high-z stage 0 or 1, its unit width leakage current is 2e-4 Amp/meter. Therefore the total width of nch devices of L=0.18u are computed.
 - For nch device at high-z stage 2, the unit width ratio 1.5e-4 is used.
 - For nch device at high-z stage 3, the unit width ratio 1e-4 is used.
 - For nch device at high-z stage 4 and above, the leakage current is not computed since it is too small.
 - Similarly, a nch device with L=0.19u to L=4u is considered. It uses a different current ratio.
 - For a pch device, there are different ratios.
 - When an OFF device is found and its current ratio is not specified in the current spec file, only the total width for each given length is computed.

A sample output summary is shown in the following example:

Example 95 :

```
Summary @ time 10n from Vsrc (0) tag (t1):
    model length totWidth OFF_I(Amp)
pch L=0.48u
    W=2u (pch device with L=0.48u is not specified in the spec
    file. we simply compute the total width.)
pch L=0.18u
    W=4u Ioff=6e-010(@0.00015)
    W=301.7u Ioff=3.3187e-008 (@0.00011) HiZ=2
nch L=0.19u to 4u
    W=0u Ioff=0 (@0.0001)
    W=0u Ioff=0 (@8e-005) HiZ=2
    W=0u Ioff=0 (@5e-005) HiZ=3
nch L=0.18u
    W=183.2u Ioff=3.664e-008 (@0.0002)
    W=130u Ioff=1.95e-008 (@0.00015) HiZ=2
    W=0u Ioff=0 (@0.0001) HiZ=3
```

In the detailed mode, every OFF device is printed and the high-z stage is indicated as shown in the following example:

Example 96 :

```
xram.xr2.xr0.xu8.mn      : nch L=1.8e-007 W=8e-007
xram.xr2.xr18.xu10.xu8.mn: nch L=1.8e-007 W=8e-007
xram.xr2.xr6.xu10.xu8.mn: nch L=1.8e-007 W=8e-007
xcam.xh_30.xi7.mn        : nch L=1.8e-007 W=1e-006 HiZ=2
xcam.xh_22.xi11.mn       : nch L=1.8e-007 W=1e-006
xcam.xh_54.xi7.mn        : nch L=1.8e-007 W=1e-006 HiZ=2
xcam.xh_31.xi7.mn        : nch L=1.8e-007 W=1e-006 HiZ=2
xcam.xh_54.xi9.mn        : nch L=1.8e-007 W=1e-006
xcam.xh_59.xi9.mn        : nch L=1.8e-007 W=1e-006
xcam.xh_16.xi9.mn        : nch L=1.8e-007 W=1e-006
```

Detecting Power-Down Floating Gates

CCK detects floating gates by extending the existing HiZ (high-impedance) check to detect the leakage path during the power down mode. HiZ node voltage may be about one-half of the VDD voltage causing its fanout elements to partially conduct leakage current.

cckAnalogPDown

Detects HiZ nodes and marks their associated fanout transistors to be ON.

Appendix I: HSIM CCK

Detecting Power-Down Floating Gates

Syntax

```
cckAnalogPDown <tag=name> <time=check_time_1
    <time=check_time_2 ...> <vsr=source
        destination_vsr_name 1 <vsr=source
        destination_vsr_name 2 ... >> <<model=model_name1>
            dvt=dvt_value1> <<model=model_name2> dvt=dvt_value2>...
        >> <num=value>
```

Argument	Description
tag=name	Prints the specified <i>name</i> in the report file at the beginning of a line to indicate a warning.
time=check_time_1 time=check_time_2 ...	Specifies the time interval to in which to run cckAnalogPDown. You can specify multiple check times.
vsr=source destination_vsr_name 1 vsr=source destination_vsr_name 2 ...model=model_name 1	Specifies the voltage source (vsr) names that form the leakage path. cckAnalogPDown only looks for the conducting paths associated with the specified names.
dvt=dvt_value1 model=model_name2 dvt=dvt_value2	When the model name of a MOS element matches the <i>model_name</i> number, the pairing <i>dvt_value</i> number applies in the conducting rule for the matched MOSFET element. The <i>model_name</i> is an optional value, with wildcard support, and the <i>dvt_value</i> has a default value of 0 and can apply to any model if it is not paired with a specific model. When you specify a dvt value, the effect on the cckAnalogPdown conducting criteria is as follows: For PMOS: It is ON if min(vgs, vgd) < vth +dvt For NMOS: It is ON if max(vgs, vgd) > vth -dvt
num=value	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited.

Description

If the operation forms any conducting path with devices driven by HiZ nodes, or a DC path with the leakage current larger than the threshold value, then these paths are reported. The list can be checked to isolate the problematic HiZ nodes, so they can be clamped to either VDD or GND which avoids the leakage problem.

Examples

```
cckAnalogPDown .... model=nmos_low_dvt dvt=0.1 model=pmos_high_dvt dvt=0.8
```

Assigns a dvt value of 0.1 to the nmos_low_dvt model and a dvt value 0.8 to the pmos_high_dvt model.

```
cckAnalogPDown .... model=*_dvt dvt=0.1
```

All the model names matched with *_dvt have a dvt value 0.1

```
cckAnalogPDown tag=no_mname time=50n vsrc=vdd! vsrc=0 dvt=0.5
```

All the models have a dvt value 0.5.

cckAnalogPDownIth

Determines the paths listed in the cckAnalogPDown report.

Syntax

```
cckAnalogPDownIth <current_threshold_value>
```

Argument	Description
current_threshold_value	If the leakage current on the DC path detected by cckAnalogPDown is larger than the specified threshold value, the paths are listed in the cckAnalogPDown report.

Examples

```
cckAnalogPDown tag=chk_1 time=5n time=7n vsrc=vdd vsrc=gnd
```

Conducts a power-down leakage-path check at both 5.0 ns and 7.0 ns. It then reports DC paths with leakage currents larger than the default 1uA current threshold flowing from VDD to GND, or the conducting paths with devices driven by a HiZ node. See the following example.

```
cckAnalogPDown tag=chk_2 time=5n time=7n vsrc=vdd vsrc=gnd
cckAnalogPDownIth 10u
```

Using the previous example, except changing the default 1uA to 10uA, the report appears as follows:

Appendix I: HSIM CCK

Running Static Analysis

```
* -----
* DC Path Check (analog power-down check)
* pdownIth = 10e-006 Amp
* vsrc = vdd
* vsrc = gnd
* title = chk_2
* at = 0.000000005
* at = 0.000000007
* sort = 0
* separate_file = 1
* -----
path 1 from vdd to 0 @ 0.000000005s
  (gateIsHiZ)           xls.xli146.mp1 (PMOS: Ids=0)
    drain                xlsda1.n1n152 5.25
    source               vdd 5.25
    gate                 xlsda1.rstd -0.0155812
  (gateIsHiZ)           xlsda1.xli146.mnl
  (NMOS: Ids=1.50143e-012)
    drain                xlsda1.n1n152 5.25
    source               0 0
    gate                 xlsda1.rstd -0.0155812
```

Note: If the value for HSIMSTEADYCURRENT is greater than the value for cckAnalogPDownIth incorrect results might occur. Ensure that HSIMSTEADYCURRENT is less than cckAnalogPDownIth.

Running Static Analysis

The concept of static analysis is the ability to propagate characteristics of the design, in this case static logic 0 and static logic 1, throughout the whole design without the need of vectors or transient simulation. To propagate static logic 0 and static logic 1 throughout the design, it is necessary to define rules associated with the propagation. These propagation rules are used to determine if a static logic state (0 or 1) can be seen by a node through a network of design elements like MOSFETs, resistors, capacitors, and diodes. CCK can analyze the design state to identify static high impedance (HiZ) nodes and static DC paths, with a minimum of a design netlist and supply voltages defined.

A node is considered static 0 if it has a static conducting path to a qualified GND node and no possibility of a static conducting path to a qualified VDD node. See the following definitions of qualified GND and VDD nodes relative to this static 0 and static 1 node concept. Similarly a node is considered static 1 if it has a static conducting path to a qualified VDD node and no possibility of a static conducting path to a qualified GND node.

A qualified VDD node is defined as a node directly connected to a constant voltage source with a value greater than 0 volts.

A qualified GND node is defined as a node directly connected to a constant voltage source with a value equal to 0 volts.

The definition of a static conducting path is a series path consisting of resistors, inductors, 'on' n-MOSFET or p-MOSFET transistor drain/source channels, or a combination of these.

Static HiZ node and static DC path analysis requires MOSFETs to be either considered conducting (On), non-conducting (Off), or 'unknown'. The definitions/analysis used to determine the state of a MOSFET are described in [Example 97](#). To help facilitate more representative results, the concept of MOSFET threshold is necessary. MOSFET threshold is defined using the command parameters nmosOn and pmosOn, for n-MOSFET and p-MOSFET transistors, respectively. For example:

Example 97 MOSFET Conducting (On)/Non-Conducting (Off) Rule

```
nmosOn and pmosOn :user defined command parameters used in the On/Off mosfet
analysis with units of volts
(example: nmosOn=0.0 pmosOn=3.0).

If (nmos gate node is static 1 && its' gate voltage > 'nmosOn')
    the mosfet is considered 'on'
else if nmos gate node is static 0 the mosfet is considered 'off'
else the mosfet is considered 'unknown'

If (pmos gate node is static 0 && its' gate voltage < 'pmosOn')
    the mosfet is considered 'on'.
else if pmos gate node is static 1 the mosfet is considered 'off'.
else the mosfet is considered 'unknown'
```

By performing the static logic state propagation, it is possible to check for static HiZ nodes and static DC paths in order to improve design functionality and to improve power consumption. Because the propagation is performed statically, the coverage is greater than that achieved via dynamic simulation, ensuring more design errors are reported and ultimately more corrections can be implemented to ensure design correctness.

A node is said to be a Static High Impedance Node, if it has no HiZ qualified static conducting path to either a qualified VDD or GND node.

A HiZ qualified static conducting path is defined as a series path consisting of resistors, inductors, 'on' or 'unknown' n-MOSFET transistor drain/source channels, 'on' or 'unknown' p-MOSFET transistor drain/source channels, or a combination of these.

Appendix I: HSIM CCK

Running Static Analysis

cckStaticHzNode

Runs a static high impedance node check.

Syntax

```
cckstaticHZNode <pmosOn=val0> <nmosOn=val1>
    <fanout=0|1|2|3|4> <pCap=0|1> <separate_file=0|1>
    <pwl_time=time> <xdummy=[0|1|2]> <scope='(subckt_name,
    inst_name, model_name)'> <skipScope='(subckt_name,
    inst_name, model_name)'> <rmScope='(subckt_name,
    inst_name, model_name)'>
```

Argument	Description
pmosOn=val0	Defines the voltage threshold value that determines the 'on' state of p-MOSFET transistors.
nmosOn=val1	Defines the voltage threshold value that determines the 'on' state of n-MOSFET transistors.
fanout=0 1 2 3 4	Specifies the fanout values to report: <ul style="list-style-type: none"> ■ 0 reports all qualified HiZ nodes. ■ 1 reports only nodes connected to MOSFET gates. ■ 2 reports only nodes connected to MOSFET bulks. ■ 3 reports only nodes connected to either MOSFET gates or bipolar transistor (BJT) bases. This is the default value. ■ 4 reports only nodes connected to either MOSFET drain/source or BJT emitter/collector.
pCap=0 1	Defines the nodes to report when pseudo-capacitors are encountered. A pseudo-capacitor is defined as a MOSFET with drain and source shorted (or a bi-polar transistor with emitter and collector shorted). The possible values are: <ul style="list-style-type: none"> ■ 0: pseudo-capacitors have no affect on how HiZ nodes are reported. This is the default value. ■ 1: if a node is considered HiZ, and only pseudo-capacitors are connected via their MOSFET gates (or bi-polar transistor bases), the node is not reported. <p>The pCap option has a higher priority than the fanout option.</p>
separate_file=0 1	Outputs static high impedance node warnings into the <i>hsim output prefix.cck</i> file when separate_file is set to 0. If separate_file is set to 1, the results are output to the <i>hsim output prefix.SHZNd</i> file.
pwl_time=time	Converts the input vector value (pwl) to a constant DC value per user-specified time point.

Argument	Description
<code>xdummy= [0 1 2]</code>	<p>Defines dummy HiZ MOSFET and BJT devices. The definition of a dummy device is a MOSFET with HZ gate-and-drain, gate-and-source, or gate-and-drain-and-source; or a BJT with HZ base-and-collector, base-and-emitter, or base-and-collector-and-emitter.</p> <p>Specify 2 which does the filtering of <code>xdummy=1</code> and, filters out dummy MOSFETs with floating gate either with a dangling source or drain terminal.</p> <p>Specify 1 to not report a node defined as HiZ and all its gate (or base) connection devices defined as dummy HZ MOSFET (BJT).</p> <p>Specify 0 so that dummy HiZ devices have no effect on how HiZ nodes are reported (the default).</p>
<code>scope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
<code>skipScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering control to prevent voltage propagation from occurring through devices or nodes.
<code>rmScope='(subckt_name, inst_name, model_name)'</code>	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Description

By performing the static logic state propagation, it is possible to check for static HiZ nodes and static DC paths in order to improve design functionality and to improve power consumption. Because the propagation is performed statically, the coverage is greater than that achieved via dynamic simulation, ensuring more design errors are reported and ultimately more corrections can be implemented to ensure design correctness.

A node is said to be a Static High Impedance Node if it has no HiZ qualified static conducting path to either a qualified VDD or GND node.

A HiZ qualified static conducting path is defined as a series path consisting of resistors, inductors, ‘on’ or ‘unknown’ n-MOSFET transistor drain/source channels, ‘on’ or ‘unknown’ p-MOSFET transistor drain/source channels, or a combination of these.

Note: You can specify all of the `cckStaticHZNode` options with global settings.

Examples

```
cckstaticHZNode pmosOn=3.0 nmosOn=0.0
```

Appendix I: HSIM CCK

Checking for a Static DC Path

Checking for a Static DC Path

A static DC path exists if there is a DC qualified static conducting path between a qualified VDD node and another qualified VDD or GND node.

cckStaticDCPath

A DC qualified static conducting path is a series path consisting of resistors, inductors, ‘on’ NMOS transistor drain/source channels, and ‘on’ PMOS transistor drain/source channels.

Note: You can specify all of the `cckStaticDCPath` options with global settings.

Syntax

```
cckStaticDCPath pmosOn=val0 nmosOn=val1 <report='conn'>
    <separate_file=0|1> <diodeOn=vt_value> <num=value>
    <pwl_time=time> <subckt=subckt_name> <model=model_name>
    <von=value> <scope='(subckt_name, inst_name,
    model_name)'> <skipScope='(subckt_name, inst_name,
    model_name)'> <rmScope='(subckt_name, inst_name,
    model_name)'>
```

Argument	Description
<code>pmosOn=val0</code>	Defines the voltage threshold value that determines the ‘on’ state of p-MOSFET transistors.
<code>nmosOn=val1</code>	Defines the voltage threshold value that determines the ‘on’ state of n-MOSFET transistors.
<code>report='conn'</code>	Prints out additional connectivity and associated R/L value information in the path report, when the output is in standard output format. See Example to know more.
<code>separate_file=0 1</code>	Outputs static high impedance node warnings into the <code>hsim output prefix.cck</code> file when <code>separate_file</code> is set to 0. If <code>separate_file</code> is set to 1, the results are output to the <code>hsim output prefix.SDCPath</code> file.
<code>diodeOn=vt_value</code>	If $V(\text{anode}) - V(\text{cathode}) \geq vt_value$, the diode is considered as a conducting device if both its anode/cathode have a constant voltage value. Otherwise the diode device is considered to be non-conducting.

Argument	Description
num=value	Limits the number of warnings from the analysis to the specified value. The default is to show all the warnings.
pwl_time=time	Allows CCK to automatically convert the input vector value (pwl) to a constant DC value per user-specified time point.
subckt=subckt_name	Only the DC path that has one or more of its associated devices in the specified subcircuit, is reported. This option acts as a path reporting filter.
model=model_name von=value	You can use this option with pmosOn/nmosOn options to allow CCK to use the model-based threshold value specification method. For example, if you specify: cckStaticDCPath pmoson= 3 nmoson=0.2 model=hvmd von= 0.1 model=lvmd von=2.5 For mos model hvmd (nmos), this example specifies that the conducting condition is v>von, and for lvmd mos (pmos) the conducting condition is v<von. For other MOS devices that do not satisfy the condition, CCK uses the value specified by the pmosOn and nmosOn arguments.
scope='(subckt_name, inst_name, model_name)'	Specifies a filtering control mechanism for the devices or nodes to be reported if there are violations.
skipScope='(subckt_name, inst_name, model_name)'	Specifies a filtering control to prevent voltage propagation from occurring through devices or nodes.
rmScope='(subckt_name, inst_name, model_name)'	Specifies a filtering mechanism to prevent devices or nodes from being reported.

Examples

```
cckStaticDCPath pmosOn=3.0 nmosOn=0.0
```

When output is in the standard output format and the option report is set to conn (report='conn'), the command will print out additional connectivity and the associated R/L value information in the path report, like the following example:

Appendix I: HSIM CCK

CCK Utilities

```
...
Path #1: from vdd to gnd
Attributes: V(vdd)=3.00
Dev #1: mp1
    Status: Conducting
    Connectivity:
D: net1
    S: net2
Dev #2: R1
    Attributes: R=300.00
    Status: Conducting
    Connectivity:
P: net2
    N: net3
Dev #3: mn2
    Status: Conducting
    Connectivity:
D: net3
    S: net4
...
...
```

CCK Utilities

These CCK commands are designed to check various circuit functions that are not covered in one of the other categories. These commands include:

- `cckBasic`
 - `cckCompareOp`
 - `cckPatternMatch`
 - `cckPatternConstraint`
 - `cckSetMosDir`
 - `cckTgPair`
-

cckBasic

Performs a basic check for valid parameters, substrate connections, node paths to voltage sources, and floating gates. Warnings are reported when errors or inconsistencies are found.

Syntax

`cckBasic [0|1] num=<n>`

Argument	Description
0 1	To enable this command, add <code>cckBasic 1</code> in the CCK command file. No other commands should be added to this file. CCK then checks for: <ul style="list-style-type: none"> ▪ Parameters ▪ Substrate connection before DC ▪ Whether nodes have path to voltage sources ▪ Floating gates
num=<n>	Limits the number of warnings generated by a particular analysis command defining it. Only the maximum specified number of warnings are generated for a particular analysis command as defined by that command. The default value is 300. If <i>n</i> is set to less than or equal to 0, the warnings are unlimited. You can specify this option with global settings.

Examples

`cckBasic 1 num=2000`

Enables `cckBasic` and sets the maximum number of warnings to 2000.

cckCompareOp

Provides an automatic DC comparison of HSIM results with results in a given reference file.

Syntax

```
cckCompareOp <refFile=fileName> <format=[eldo|hsim]>
    <subckt=subckt_name> <node=node_name>
    <skipsub=skip_sub_name> <skipnode=skipnode_node_name>
    <time=doComparisionAtThisTime>
    <outFile=output_file_name>
```

Argument	Description
<code>refFile=fileName</code>	Specifies the reference file name.
<code>format=[eldo hsim]</code>	Specifies an Eldo or HSIM format netlist.
<code>subckt=subckt_name</code>	Specifies the subcircuit or nodes to be filtered out of the reference file. Only nodes specified in the scope of the subcircuit or node names are compared.
<code>node=node_name</code>	

Appendix I: HSIM CCK

CCK Utilities

Argument	Description
skipsub=skip_sub_name	Skips nodes in the given scope when comparing reference nodes.
skipnode=skipnode_node_name	
time=doComparisonAtThisTime	DC time at which comparison is conducted. The default is 0.
outFile=output_file_name	Compares results file to the reference file. If not specified, the output is written to the <i>prefix.cck</i> file, where <i>prefix</i> is given in the -o command line option of HSIM.

Description

The `cckCompareOp` comparison automatically finds the nodes within the given file, and then compares HSIM results with the reference data at those specified nodes.

The output is an ASCII report file containing the following fields:

- Node_Name compares Eldo and HSIM hierarchical names
- HSIM_DC_OP(V) for HSIM-provided DC op condition
- REFERENCE_DC_OP(V) for Eldo-provided DC op condition
- Relative difference in percentage relative to HSIMVDD

The output file is sorted by relative difference with the highest value on top. The reference DC OP format is ELDO.

Examples

```
cckCompareOp refFile=inv1_0.iic time=0n format=eldo \
outFile=hsim_cmpop.log skipsub=inv10 skipnode=*
cckCompareOp refFile=inv1_100n.iic time=100n format=eldo \
outFile=hsim100_cmpop.log node=*
```

The following is a typical `cckCompareOp` output file.

```
*****
* Comparison of operating points.
*
* Spectre OP file: inv1_100n.iic
* transient time: 0
*****
***-----Start Compare Op----- ***
***-----Hsim_Vdd(%)-----
Node_NameHSIM_DC_OP(V)Eldo_DC_OP(V)(HSIM-Eldo)_DC_OP/
x3.5 +5.00000000000000-0.0000017426983195+166.667
x5.9 +5.00000000000000-0.0000042658173509+166.667
x5.7 +5.00000000000000-0.0000006782416429+166.667
x5.5 +5.00000000000000-0.0000027092119370+166.667
x1.3 +5.00000000000000-0.0000007398899505+166.667
x5.3 +5.00000000000000-0.0000001779232473+166.667
x1.5 +5.00000000000000-0.0000001967053253+166.667
x5.1 +5.00000000000000-0.0000024511264310+166.667
x1.7 +5.00000000000000-0.0000024611588271+166.667
x4.10 +5.00000000000000-0.0000000221863606+166.667
x1.9 +5.00000000000000-0.0000001658784807+166.667
x4.8 +5.00000000000000-0.0000029060007400+166.667
x4.6 +5.00000000000000-0.0000022913118397+166.667
x2.2 +5.00000000000000-0.0000027995864791+166.667
x4.4 +5.00000000000000-0.0000017390747854+166.667
***-----Compare Op End----- ***
***-----
```

cckMatchSub

Reports all the instances of the specified subcircuits to the output file *prefix.ccksub* by traversing the hierarchical netlist.

Syntax

```
cckMatchSub <subckt=subckt_name1,subckt_name2, . . .>
             <ReptHierNode=0 | 1>
```

Argument	Description
subckt=subckt_name1, subckt_name2, ...	Specifies the subcircuit names to be matched.
ReptHierNode=0 1	If you set this option to 1, it reports all the hierarchical node names in the output file <i>prefix.ccksub</i> . The default is 0.

Examples

```
cckMatchSub subckt=buf1 ReptHierNode=1
```

The output is shown in the following example:

Appendix I: HSIM CCK

CCK Utilities

```
Instances of Sub buf1:  
Instance 1: x1  
* Port *  
in  
-- x1.in  
-- tx1  
out  
-- x1.out  
-- tx2  
vdd  
-- x1.vdd  
-- v3  
gnd  
-- x1.gnd  
-- v5  
Instance 2: x2  
* Port *  
in  
-- x2.in  
-- tx2  
out  
-- x2.out  
-- tx3  
vdd  
-- x2.vdd  
-- v2  
gnd  
-- x2.gnd  
-- v4  
2 Instances of Sub buf1 in Total
```

cckPatternMatch

Enables pattern matching to designated devices while running selected CCK commands. Compatible commands include:

- cckTgPair
- cckDlySkipElem
- cckXtalkSkipElem

Syntax

```
cckPatternMatch <file=pattern_file_name>  
    <subckt=subckt_name> <ReptHierNode=0|1>  
    <IgnorePattern=pattern_names> <noOverlap=1|0>  
    <MatchModel=0|1> <ReptParallel=0|1>
```

Argument	Description
file=pattern_file_name	Specifies the pattern file name containing the patterns to be selected for matching to the circuit.
subckt=subckt_name	Defines the specified subcircuit as the scope to apply pattern matching.
ReptHierNode=0 1	If you set this option to 0 (default), CCK only reports the matched node. If you set it to 1, the pattern matched node and the up-stream hierarchical nodes are reported as well.
IgnorePattern=pattern_names	Specifies pattern names that are excluded during pattern matching. You can use wildcard characters in pattern names.
noOverlap=1 0	If you specify 1 (default), MOSFET devices inside the circuit are matched only once during pattern matching. Specify 0 to indicate that the MOSFET devices inside the circuit are matched multiple times during pattern matching. Use this option only if there is an overlap between mappings of the pattern matching result.
MatchModel=0 1	Set this option to 1 to require that the models of the mappings of the pattern matching results must also match the models of the patterns specified in the pattern file. The default is 0.
ReptParallel=0 1	Set this option to 1 to report all parallel devices in the <i>prefix.cckpat_match</i> output file.

Examples

```
cckPatternMatch file=pattern IgnorePattern=p1 p2 p12*
```

The previous example shows multiple pattern names in the `IgnorePattern` option.

cckPatternConstraint

Points to a file that contains the pattern to be matched.

Syntax

```
cckPatternConstraint pattern=<pattern_name>
    Wlratio=<logical_expression>
```

Appendix I: HSIM CCK

CCK Utilities

Argument	Description
pattern=<pattern_name>	Specifies the pattern to be matched.
Wlratio=<logical_expression>	Specifies the width/length ratio expression.

Examples

```
cckPatternMatch file=aa subckt=s32_d1lat_0
cckDlyskipElem pattern=1 inst=m3 inst=m4 subckt=s32_d1lat_0
cckPatternConstraint pattern=<pattern_name> Wlratio=<logical_expression>
cckNoSimu 1
```

In the previous example `file=aa` in `cckPatternMatch` points to the file that contains the pattern to be matched as shown in the following example:

```
** 1st line must be comment or empty (same as spice file) **
.subckt 1 n3 n5
m3 n5 n3 vdd p
m4 n5 n3 gnd n
m5 n3 n5 vdd p
m6 n3 n5 gnd n
.ends
```

Note: Each pattern in the file must be described in SPICE format, except where size information is not required. All the patterns are grouped in the same file specified in the file option.

Pattern matching is enabled and applied for the `cckDlySkipElem` command to look for the designated pattern named 1 in the scope of subckt `s32_d1lat_0`. If there is a match, the corresponding `m3` and `m4` transistors are classified as qualified candidates to be skipped. Because the command file also contains `cckPatternConstraint`, which is used to specify additional matching criteria, the pattern 1 in this example must meet the matching criteria of the `WLratio='m3>m5'`.

When `cckNoSimu=1` is set, no simulation is performed. For example:

```
*****  
In sub (s32_d1lat_0) match 1 (1) patterns  
Mapping 0:  
* Mos *  
m3 -- xia2.mpl  
m4 -- xia2.mnl  
m5 -- xia4.mpl  
m6 -- xia4.mnl  
* Net *  
vdd -- vdd  
gnd -- gnd  
n3 -- zo  
n5 -- o  
*****
```

Note: Nodes connected to voltage sources are treated differently. The following example shows two patterns that are not the same.

```
<pattern 1>  
.subckt clamppnmos11 vdd nb  
MMa vdd nc nb nb NY  
RRa nb nc $[RF]  
.ends  
<pattern 2>  
.subckt clamppnmos11 na nb  
MMa na nc nb nb NY  
RRa nb nc $[RF]  
.ends
```

In pattern 1, there are nodes connected to the global VDD signal, which is a voltage source node. In pattern 2, the global VDD signal is a non-voltage source node, na. Because these two patterns are not the same, pattern matching does not treat them the same.

cckTgPair

Specifies the method for setting the transistor direction at either the gate or circuit level.

Syntax

```
cckTgPair <dirN=[D2S|S2D]> <subckt=subcktname>  
<inst=nmosName> <pattern=patternname>
```

Argument	Description
dirN=[D2S S2D]	Defines the drain-to-source (D2S) or source-to-drain (S2D). You can use this option to set the direction of an n-MOSFET device. Only the n-MOSFET direction needs to be set since other p-MOSFETs are found in this pair and assigned the same direction as in n-MOSFET.

Appendix I: HSIM CCK

CCK Utilities

Argument	Description
subckt=subcktname	Defines the specified subcircuit as the scope to apply pattern matching.
inst=nmosName	Specifies the MOSFET instance name as the starting point to define the direction.
pattern=patternname me	Specifies the pattern to be matched.

Description

This command works with pattern matching technology to identify the designated transistors based on patterns defined using `cckPatternMatch`. Pattern names selected from those definitions are matched. The direction of the transistors to be matched is limited to those with the direction specified in the `dirN` option.

Examples*Example 98 Method 1: Setting a User-Specified Transistor Gate Transistor Direction*

It is not easy to determine the correct direction for a transfer gate. Therefore, transistor direction in a transfer gate may be specified using the following syntax:

```
tgPair <dirN=[D2S|S2D]> <subckt=subcktName> <inst=nmosName>
```

`tgPair` defines the transfer gate pair, including an n-MOSFET and a p-MOSFET forming a transfer gate.

`dirN` specifies that the parameters are defined as drain-to-source (D2S) and source-to-drain (S2D). The direction of an n-MOSFET device is set using this syntax.

Note: Only the n-MOSFET direction needs to be set since other p-MOSFETs are found in this pair and assigned the same direction as in n-MOSFET.

```
tgPair dirN=S2D subckt=aa inst=mn1
```

`subckt=aa` finds an n-MOSFET transistor `mn1` and assigns its direction as S2D. All the instantiations of `aa` have this direction for `mn1`.

The node name may also be used as the starting point to define the direction. For example, a MOSFET device has a drain node called `dd` and source node

called ss. If this transistor is from drain to source, then node dd sets the direction in mn1.

```
tgPair <fromNode=nd1> <subckt=aa> <inst=mn1>
```

where `fromNode=nd1`. The `mn1` instance in `subckt aa` is first found. Node `nd1` must be either a drain or source node of `mn1`. If `nd1` is `mn1`'s drain node, this syntax command sets the `mn1` direction to be drain-to-source. If `nd1` is `mn1`'s source node, then the `mn1` direction is source-to-drain.

Example 99 Method 2: Defining Transistor Direction in a Circuit

```
.param <hsimMosDir=[1|2|3]> <subckt=aa> <inst=bb>
```

Argument	Description
<code>hsimMosDir=[1 2 3]</code>	<ul style="list-style-type: none">▪ 1 for source-to-drain (S2D).▪ 2 for drain-to-source (D2S).▪ 3 for bidirectional (BID).
<code>subckt=aa</code>	Specifies that in the <code>aa</code> subcircuit, the direction selected by the option is assigned to the <code>bb</code> transistor. The <code>bb</code> transistors in all the instantiations of <code>aa</code> have the same direction.

Running the CCK Tutorial

This tutorial provides information and examples for using CCK commands in HSIM. The commands are specified in an input netlist file as follows.

Running CCK

In the main input file - (`top.sp`), add the following lines so that HSIM reads in the two CCK and device files.

```
.param hsimCktCheck=cck_cmd_file
```

The `tcheck mosv` commands with its parameters is used to monitor the node voltages. The set can be invoked by adding the following syntax to an HSIM input file.

```
.param hsimDeviceV=dev_v_file
```

Appendix I: HSIM CCK

Running the CCK Tutorial

The ntrig and intrig commands are used in the interactive mode to find the first state change for the node after a specified time in nanoseconds. ntrig specifies the node name while intrig specifies the node ID. The report lists the elements causing the node voltage to change. For example:

```
ntrig node_name <-t time> <-f file1> <-mt time1>
intrig node_id <-t time> <-f file1> <-mt time1>
```

Test Case

Typical commands used in CCK include the following from the CCK file.

Note: For comments: /* (start comment) and */ (end comment) are used. /* must always be placed at the beginning of a line before simulation; for example, immediately following netlist parsing. For syntax: \" denotes a line continuation.

```

cckParam erMaxCap=0.001 waMaxCap=1.e-8
cckParam erMaxMosW=0.01 waMaxMosW=1000u
cckParam erMaxMosL=0.01 waMaxMosL=1000u
cckParam erMaxMosAD=0.0001 waMaxMosAD=1.e-6
cckParam erMaxMosPD=0.01 waMaxMosPD=0.001
cckParam erMaxMosTox=5.e-8 waMaxMosTox=3e-8
cckParam num=100
/* To check a particular model, add one of the following:
 * cckParam model=name, erMaxMosW=..., and waMaxMosW=...
 * These parameters contain the specific values shown in
 * this test case.*/
cckParam erMaxMosW=2
cckParam model=m1 erMaxMosW=4
/* In this test case, template A0 is created to keep the
 * default parameter values.
 * Line 1.Does not have any model, overwrites the
 * maxMosW to be 2 in template A0, which is used for general
 * purpose checking.
 * Line 2.Creates a new template A1 that contains the
 * default parameter values and the new maxMosW=4.
 * Therefore, complete the following steps:
 * Define all the parameters' values for general purpose
 * checking.
 * Start to define the special values for different models;
 * one model per line where each line contains the new
 * values for this model.
 */
/* Substrate check. Forward bias threshold is 0.2 */
cckSubstrate num=500 vt=0.2 mode=2 start=10n stop=50n \
start=200n stop=300n
/* Diode check. Forward bias threshold is 0.4 */
cckDiode num=500 vt=0.4 start=10n stop=50n
/* Floating gate check */
cckFloatGateIsrc 1
/* Static leakage path check */
cckMaxStaticLeak num=100
/* Check whether a node is stuck at 0 or 1. Limit the node
 * checking to some area of the design indicated by an asterisk (*) */
cckMaxStuckAt0 num=5000 node=xcam* node=xram*
cckMaxStuckAt1 num=5000 node=xcam*
/* Check each pmos to determine which logic-high power
 * supply its nodes can reach.
 * a.) The (min, max) voltage sources reached from gate,
 *      drain, src.
 * b.) If the gate node is less than drain/src by 0.3 volts,
 *      it is reported.
 * c.) Since voltage drop is considered through nmos and
 *      pmos, specify vnth=0.6 for nmos and vpth=-0.5 for pmos
 */
cckPmosG_lt_DS vhth=0.9 vnth=0.6 vpth=-0.5 inst=* vt=0.3
/* Check each pmos. If a gate has a higher voltage than D/S,
 * it is reported. */
cckNmosG_gt_DS vlth=0.7 vnth=0.6 vpth=-0.5 inst=* vt=0.2
/* Pmos substrate check. Trace from D/S and Bulk nodes. If a
 * bulk node reaches a lower power supply, it is reported. */
cckPmosB_lt_DS vhth=0.9 vnth=0.6 vpth=-0.5 inst=* vt=0.1 \
num=200
/* similar for nmos's bulk checking */
cckNmosB_gt_DS vlth=0.7 vnth=0.6 vpth=-0.5 inst=* vt=0.1 \
num=200

```

Appendix I: HSIM CCK

Running the CCK Tutorial

```
/* Path to vdd/gnd check*/
cckPathToVsrc num=900 node=* fanout=1
/* Add the following syntax immediately following the DC
 * operating point: */
/* To check for un-initialized latches. Focus on the
 * instances in subckt=q1latchsd1 */
cckLatchUnInit 1
cckLatchInElem subckt=q1latchsd1 inst=*
cckLatchSkipElem subckt=a920traddrbuf2 inst=xu5*
cckLatchSkipElem inst=xram.xpipe_5*
/* To get a static path delay analysis, specify: */
cckRCDlyPath 1
cckRCRiseDelay min=0.05n max=3e-9 inside=0
cckRCFallDelay min=0.04n max=3.2e-9 inside=0
cckLimitRisePmosFallNmos 1
cckSetMosDir 0
cckDlySkipElem subckt=p2sticky inst=mpwk
inst=mnwk
/* To do static crosstalk analysis, specify: */
cckXtalkFloatingCap 1.e-14
cckXtalkRiseVolt 0.04
cckXtalkFallVolt 1.5
cckXtalkByWL 0
cckXtalkmodelWLratio model=pch min=12 max=20
cckXtalkmodelWLratio model=nch min=12 max=15
cckXtalkByRC 1
cckXtalkRiseTimeConst min=0.5n max=3n
cckXtalkFallTimeConst min=0.4n max=4n
/* See the description of cckRCDlyPath
 * See the description of cckXtalkByRC, cckXtalkByWL, and
 * cckStaticXtalk_GroupCmd */
/* To skip some node or look at some node, then specify: */
cckXtalkAtNode node=xcam* --> examine these nodes
cckXtalkAtNode node=xram* --> examine these nodes
cckXtalkSkipNode node=xram* --> don't go thru these nodes
cckXtalkSkipElem inst=xram.x* --> don't go thru these \ elements
/* NOTE: At this time Skip and AtNode are mutually exclusive.
 * If Skip is used, then AtNode cannot be specified.
 * Conversely, if the AtNode is chosen,
 * then Skip can't be used. */
/* during simulation, fast check node voltage, elem current
 * and leakage current */
/* in simulation, check node min and max voltage */
cckNodeVoltage num=100 vmax=2.0 vmin=-0.3
cckNodeVoltage num=100 vmax=1.64 vmin=-0.3 model=nch
cckNodeVoltage num=100 lvgs=-1.66 lvgd=-1.66 model=* \
start=60n
cckNodeVoltage num=100 lvgs=-1.64 lvgd=-1.64 model=pch \
start=20n stop=50n
/* (Note: in this comment an asterisk (*) is used to
 * represent a number.) To check the node voltage of
 * transistors connecting to constant * power supply,
 * use ...*. The following command checks the tp018
 * transistors
 * connecting through * resistors to constant power supply
 * 1.8V. If the D/S node voltage * is less than 1.79, it is
 * reported. This command can do a quick check * on
 * IR-drop */
cckNodeVoltage model=tp018 constV=1.8 vmin=1.79
```

```
/* Check the current through element in simulation */
cckElemI ith=1.e-6 model=pch constV=1.65
```

Test Case Example for tcheck mosv

Typical commands used in CCK for device voltage checking include the following from the tcheck mosv file.

```
.tcheck jtlv mosv model=pch mos=xram* lvdb=-1.6 uvdb=-1 \ lvsb=-2.6 uvsb=-1.5
stop=30n start=60n
.tcheck jtl2 mosv model=pch \ mos=xram* lvgs=-0.6 uvds=1 .tcheck jtl1 mosv
model=nch mos=* cond=' \
(vds < -1.0 || vgs < -0.75)'
/* to do post-process device voltage check, add */
.tcheck post=1
.tcheck post=2
```

When post=2 appears, voltage continues to be read in the fsdb file using the same process.

Note: If post=1, then two processes are run:

- Standard HSIM
- HSIM -post_devv

Feedback

Appendix I: HSIM CCK

Running the CCK Tutorial

Index

Symbols

/\$HSIM_HOME/tutorial/fadd8 356
/\$HSIM_HOME/tutorial/gscap 360
/\$HSIM_HOME/tutorial/logic 1020
#alias 311
#edge_shift 313
#format 307
#i_delay 313
#o_delay 313
#outz 313
#scale 313
#scope 315
#slope 314
#fall 314
#rise 314
#triz 314
#vih 314
#vil 314
#voh 315
#vol 315
\$AMOS 60, 73
\$CC 73
\$CC=1 73
\$display 580
\$error 580
\$fstrobe 580
\$HSIM_HOME/etc 206
\$HSIM_HOME/etc/include 724
\$HSIM_HOME/tutorial 213
\$HSIM_HOME/tutorial/isocap 358
\$MODEL 137, 138
\$monitor 580
\$SPICE 73
\$strobe 580
\$warn 580

Numerics

12-digit representation 62

16-bit address modules 657
16-bit D/A converter 698
32-bit address modules 657
64 Bit executable 208

A

A/D converter 61, 73, 171, 650, 698
AANNI - Native Netlist Integration 793
aborting netlist compilation 206
absolute upper bound 906
abstol 584
AC 3, 446
.ac 447
AC analysis 4, 230, 446, 447, 448, 449, 451, 452
AC component 446
AC drop support 584
AC grounded 347
AC Small-Signal Analysis 445
.ac.a 450
.ach list file 100
.acheck 538, 539
active components 20
active element drivers 4
active loading elements 4
ad 250
additional drain resistance 250
additional source resistance 250
AddNextAnalogValueChange 738
AddNextDigitalValueChange 737
address decoder 219
adonly 414, 419
agauss 547
aggression levels 23
ako 251
.alter 325, 327, 450
.alter statement 458
alter statement 450, 458
AM source Function 275

Feedback

Index

B

analog bus 568
analog circuit 60, 61, 150, 152, 171
analog port
 READ 654
 WRITE 654
analog waveform shape 297
analog/memory design styles 18
analysis 429
analysis functions
 579
 analysis("ic") 579
 analysis("static") 579
 analysis("tran") 579
-anan 394
ap 471
API 729
API calling sequence
 725
 transient analysis 725
API functions 650
application-specific integrated circuit 17
as 250
ASCII 10, 11, 540
ASCII format 408, 458
ASCII input netlist 9
ASCII output file 404
ASIC 17
assign_geometry 602, 616
at 527
aunif 547
automatic calculation of second-order mutual
 inductance for multiple coupled inductors 335

B

b3_assigngeometry 616, 617
b3_conclude 609
b3_initialgeometry 609
b3_initialmodel 609
b3_load 618
b3_readmodel 614, 615
b3_setmodel 615, 616
b3_start 609
b3_tempgeometry 617, 618
b3assigngeometry.c 600, 616
b3defs.h 599, 608, 609
b3load.c 600, 618, 619, 620
b3main.c 600, 609, 610
b3readmodel.c 600, 614
b3set.c 600
b3temp.c 600, 617
back-annotation 9
back-annotating DPF 20
backend verification and optimization schedules 23
Backus Naur Form 405
Bartlett triangular window 429
BaseName 729
BeginCreateWave 731, 732
behavioral modelling 554
bias point value 445
.BIASCHK command 46
biasing voltage 601, 618
bidirectional input/output (I/O) port 221
binary 10
bipolar junction transistor 228
bipolar transistors 3
biput 651
bisection 523
bisection optimization 230
bit-line 126, 220
BJT 228, 446
BJT element name 257
BJT Model
 G.-P. Model 232
 Gummel-Poon BJT model 258
 HICUM 2.2.1 258
 HICUM 2.3.1 259
 HICUM0 259
 HICUM2.1 258
 HICUM2.2 258
 Mextram-503 258
 Mextram-504 258
 Mextram-504 model 232
 VBIC model 232
 VBIC-1.15 258
 VBIC-1.2 258
BJT parameters 257, 258
Blackman window 429
Blackman-Harris window 429
block characterization 326
block current assessment 17
block optimization 326

block statement 572
block-level flow 15
block-level simulations 7
BNF 405
bound 578
bound_step 578
branch 568
branch contribution 571
branch current waveform 412
branches 568
buffer size 226
buildfmod 600, 649
built-in functions
 abs(x) 323
 acos(x) 323
 asin(x) 323
 atan(x) 323
 atan2(x) 323
 ceil(x) 324
 cos(x) 323
 cosh(x) 323
 db(x) 324
 dmax(x,y) 324
 dmin(x,y) 324
 exp(x) 324
 floor(x) 324
 int(x) 324
 ln(x) 324
 log(x) 324
 log10(x) 324
 max(x,y) 324
 min(x,y) 324
 pow(x,y) 323
 pwr(x,y) 324
 sgn(x) 324
 sign(x,y) 324
 sign(x) 324
 sin(x) 323
 sinh(x) 323
 sqrt(x) 323
 tan(x) 323
 tanh(x) 323
 trunc(x) 324
bulk node name 249
bulk terminals 52, 70, 171
bus syntax 307
Bus width declaration support 584

C
C behavioral description 229
C compiler 649
C function evaluation 651
C functional model 649, 650
C language 649, 650
C_OUT digital port 655
Caa 590
Cadence
 Design Framework II 834
 Virtuoso Analog Design Environment 834
CAM 153
capacitance matrix 242
capacitance report 488
capacitance value keyword 236
capacitance values
 cbdb 601, 619
 cbgb 619
 cbsb 601, 619
 cddb 601, 619
 cdgb 601, 619
 cdsb 601, 619
 cgdb 601, 619
 cggb 601, 619
 cgbs 601, 619
capacitance-only back-annotation 157, 168
capacitor 446
 voltage-controlled 287
capacitor element name 235, 590
capacitor length 236
capacitor model name 236, 590
capacitor parameters 235
capacitor terminals 346, 348
capacitor width 236
capbd 601, 619
capbs 601, 619
capgb 601, 619
capgd 601, 619
capgs 601, 619
capop 601, 619
carrier frequency 275
case 572
case sensitivity in Verilog-A 565
cbgb 601
CCCS 229, 446
CCCS parameters 283

Feedback

Index

C

- CCCS syntax statements 281, 282
- CCK 904, 905, 906, 968, 1004, 1009, 1010, 1025, 1038, 1040
- CCK's absolute upper boundary 904
- CCK's soft upper boundary 904
- ckkDioV 933
- ckkExiPath 952
- ckkFloatGateSrc 1093
- ckkMosV 918, 925, 926, 953, 956, 960, 973, 977
- ckkMosv 924, 925
- ckkNodeMaxRF 1028
- ckkToggleCount 1010, 1011
- CCVS 229, 446
- CCVS parameters 285
- CCVS syntax statements 284
- cell characterization 326
- cell optimization 326
- cobo 601, 618, 619
- codo 601, 618
- cgso 601, 618, 619
- change simulation control parameters 465
- characteristic impedance 239
- charge conservation feature 232
- charge-based capacitance model equations 601
- check
 - timing 506
- check commands 6
- check_window 293, 294
- circuit
 - simulation example 213
- circuit admittance matrix 346
- circuit characteristics 20
- circuit debugging environment 4
- circuit element 547
- circuit element connectivity 226
- circuit element instances 545
- circuit latency 148
- circuit node voltages 416
- circuit reliability 20
- circuit simulators 6
- circuit topology description 226
- CircuitCheck 1017
- CircuitCheck Commands
 - ckkLatchInElem 1006
 - ckkLatchSkipElem 1007
 - ckkLatchUnInit 1005
 - ckkLimitRisePmosFallNmos 1038
 - ckkMaxStackUpNmos 1008
 - ckkMaxStackUpPmos 1009
 - ckkMaxStuckAt0 1009, 1010
 - ckkMaxStuckAt1 1010
 - ckkNodeVoltage 968, 969
 - ckkOffLeakI 1071
 - ckkParam 904, 911, 913
 - ckkPathToVsrc 971, 972
 - ckkPmosG_It_DS 982
 - ckkRCDlyPath 1029, 1032, 1033, 1034, 1035, 1038
 - ckkRCFallDelay 1038
 - ckkRCRiseDelay 1038
 - ckkResV 933
 - ckkSetMosDir 1038
 - ckkXtalkByRC 1063
 - ckkXtalkFallTimeConst 1063
 - ckkXtalkFallVolt 1063
 - ckkXtalkFloatingCap 1063
 - ckkXtalkRiseTimeConst 1063
 - ckkXtalkRiseVolt 1063
- CircuitCheck commands
 - ckkParam 904
- clock net analysis (jitter and skew) 17
- cloelog 473
- CloseWaveFile 739
- coarse optimization of timing, voltage, and current 12
- coercive voltage 594, 596
- COI interface 408
- coi.h 726
- command descriptions 483
- command history 465
- commands
 - #alias 311
 - #edge_shift 313
 - #format 307
 - #i_delay 313
 - #o_delay 313
 - #outz 313
 - #scale 313
 - #scope 315
 - #slope 314
 - #tfall 314
 - #trise 314
 - #triz 314

#vih 314
#vil 314
#voi 315
#vol 315
.BIASCHK 46
.PROTECT 203
.UNPROTECT 204
Complex BJT Device Parameter
HSIMHASCOMPLEXBJT 93
complex numbers 451
complex phasor 58, 451
conclude 602, 609
conditional interruption 499
constant node voltage status 483
constant voltage source 346
constant voltage source node 347
cont 471
continue 471
control and data file 404
control file 405
controlled current source voltage 278
controlled heuristics 23
controlled source negative node name 278
controlled source positive node name 278
controlling current through vc source 284, 285, 287, 288
converting NOF to tabular data output 405
core cell 22
corresponding output current of xk 284, 288
corresponding output current of xk. 285, 287
coupling capacitance 5, 18
coupling effect 52, 60, 70, 171, 346, 348, 349, 361
CPU time 221
CreateHeader 730
CreateWave 733, 736, 738
CreateWaveDC 736, 738
CreateWaveFile 728, 730, 731, 733, 734, 735, 736, 738, 739
CreateWaveFileDC 730, 731, 733, 734, 735, 738, 739
cross 574
cross-coupling 7
cross-coupling capacitors 17, 360
cross-talk 17
crosstalk analysis 1051
crosstalk assessment 17
cross-talk noise simulation 5
current gain factor 283
current output keyword 279
current source 446, 450, 453, 454, 455
 current-controlled current source 446
 voltage-controlled current source 446
current sources statement 449
current through resistor - imaginary part 451
current through resistor - real part 451
current through source terminal magnitude 451
current-controlled current source 229, 283
current-controlled current source keyword 283
current-controlled current source name 283
current-controlled switches 445
current-controlled voltage source 229
cycle time 298

D

D/A converter 61, 73, 171, 650, 698
damping factor 274
.data 326
data 457
data file 405
data file, include 331
data R/W operation 222
data syntax 448
DC analysis 3, 230, 453, 545, 728, 730, 731, 733, 734, 735, 736, 738, 739, 998, 1005
DC component 446
DC convergence 99
DC current 475
DC current path 474, 475, 527, 528, 529
DC current path between vcc and vss 529
DC current path between vdd and GND 529
DC current source 449
DC current source value 272
DC initialization 10, 333, 351
DC initialization algorithm 88
DC initialization parameters
 HSIMDCI 352
 HSIMDCINIT 76, 77, 352
 HSIMDCITER 76, 352
 HSIMDCSTEP 352
 HSIMDCV 77, 352

Feedback

Index

D

- HSIMENHANCEDC 352
- HSIMKEEPNODESET 352
- HSIMMULTIDC 352
- DC Interactive Mode Commands
 - option 459
- DC interactive mode commands
 - dccont 459
 - pt 459
 - quit 459
 - quitdc 459
- DC interactive mode debugging commands
 - HSIMDCSTOPAT 458
- DC iteration 88, 99
- DC operating point 230, 333
- DC path 474
- DC path check parameters
 - at 528
 - delay 528
 - ith 528
 - node1, node2 .. nodeN 528
 - period 528
 - start 528
 - stop 528
 - title_name 528
- DC path search 474
- DC path search result 474
- DC source value key word 271, 272
- .dc statement 454
- .dc temp 456
- DC voltage source 271
- DC voltage source node 475
- .dc_mc file 110
- .dc.a 458
- dccont 464
- dcpath 474, 527
- Debugging Command 1017
- dec 454
- decibel 451, 452
- default binary output format 10
- default temperature 339
- define
 - sub-circuit 338
 - temperature 339
 - transient analysis 339
- define compiler directives 581
- .del param 327
- delay 294, 297, 527
- delay element 281
- delay statement 294
- delay time 275, 277, 281
- delvto 250
- derivative measurements 424
- design optimization 23
- detailed standard parasitic format (DSPF) 9
- device geometric data 364
- device mode 601, 606, 618
- device model 547
- device model control parameters 73
- Device Model Parameter
 - HSIMABMOS 73, 81
 - HSIMAMOS 73
 - HSIMBMOS 70, 73, 81
 - HSIMCC 73
 - HSIMDIODECURRENT 81
 - HSIMDIODEVT 82
 - HSIMENHANCEDIDDQ 88
 - HSIMIDDQ 99
- device models 545
 - BJT 205
 - diode 205
 - JFET 205
 - MOSFET 205
- Device Parameter Format 21
- device parameter format 364
- diagonal entry value 346
- dielectric-loss conductance matrix 242
- differential amplifier 220
- digital vector file 291
- digital vector input and output files 192, 291
- diode 446
- diode element name 255
- diode forward bias 999
- diode model level selector 256
- diode model name 255, 256
- diode model parameters 256
- diode parameters 255
- disciplines 567, 581
- disciplines.h 561
- disciplines.h file 560
- discrete disciplines 584
- distributed signal nets 22
- distribution function 547
- dlink 326

domain switching 594
 double precision numbers 566
 double sweeps 456
 double-type array 690
 dp 471
 DPF 21, 82, 364
 DPF annotation-related parameters
 HSIMDPFHIERID 83
 HSIMDPFPFX 84
 DPF back-annotation 83
 Dplot 12, 405
 drain capacitance 359
 drain charge 601, 619
 drain current 601, 618
 drain node name 249
 drain resistance calculation source diffusion squares 250
 drain terminals 52, 70, 171
 drain-bulk junction diode area 335
 drain-bulk junction permieter 250
 drain-diode area 616
 drain-diode periphery 616
 DRAM 5
 DRC 20
 DSP 5
 DSPF 9, 21, 22
 DSPF back-annotation 22
 DSPF file for back-annotation 154, 155
 DSPF/SPEF annotation-related parameters
 HSIMSPFTRIPLET 170
 DSPF/SPEF file 160
 DSPF/SPEF netlist 364
 dttemp 250
 dynamic circuit data 463
 dynamic crosstalk analysis 1051
 dynamic library loader 724
 dynamic logic 17

E

edge 466, 500, 522
 EEPROM 5
 elem2 530
 element
 BJT transistor 256
 capacitor 235

diode 254
 JFET transistor 259
 lossless transmission line 239
 lossy transmission line 240
 MESFET transistor 259
 MOS transistor (MOSFET) 249
 mutual inductor 238
 resistor 234
 self-inductor 237
 element branch current waveforms 6
 element capacitance 4
 element conductance 4
 element current 4
 element index number 479
 element model parameter values 226
 element name 829
 element parameter variation 547
 element scale parameter 234, 236, 238, 288, 336
 element statements 233
 element terminal nodes 4
 elements of AC analysis 446
 encmode 251
 encryption
 metaencrypt 231
 triple DES 231
 Verilog-A 231
 .end 116, 227, 328
 EndCreateWave 734
 .endl 328, 332
 .ends 328
 enhanced design margins 20
 EPROM 5
 ERC 20
 error controlling parameter 584
 ev 479
 excessive current check parameters
 ith 531
 start_time1, stop_time1 . . . start_timeN,
 stop_timeN 531
 title_name 531
 tth 531
 excessive current checks 4
 excessive element-current check 530
 excessive rise/fall time check 532
 excessive rise/fall time check parameters
 fall 533
 fanout 533

Feedback

Index

F

- node1, node2 ...nodeN 533
- rise 533
- title_name 533
- utime 534
- vhth 534
- vlth 534
- exi 530
- exponential source function 276
- exponential source function keyword 276
- exponential source initial value 276
- exrf 532
- exrf node1 532
- external loop variable 458
- external parameter 447
- external sweep 447
- external value 447
- extracted circuit netlist 21
- extracted geometric data 364

- F**
- fall 297, 301
- falling edge delay time 276
- falling edge time constant 276
- fan-in 4
- fan-out 4
- fanout 532
- fanout transistors 1073
- fast fourier analysis 410
- fast fourier transform 230, 428
- FeCap Element 590
 - area 590
 - model_name 590
 - n1 590
 - n2 590
 - p0 591
 - v0 591
- FeCap element 590
- FeCap model parameters 591
 - as1 591
 - as2 592
 - au1 592
 - au2 592
 - bs1 591
 - bs2 592
 - bu1 592
 - bu2 592
 - cs1 591
 - cs2 592
 - cu1 592
 - cu2 592
 - ds0 592
 - du0 592
 - kas1 592
 - kas2 592
 - kau1 592
 - kau2 593
 - kbs1 592
 - kbs2 592
 - kbu1 592
 - kbu2 593
 - kcs1 592
 - kcs2 592
 - kcu1 592
 - kcu2 593
 - kds0 592
 - kdu0 593
 - kpmax 591
 - Pmax 591
 - vcr 591
 - vsat 591
- FeCap parameters 590
- feedback coupling 61, 62
- ferroelectric capacitor 589, 595
- ferroelectric capacitor (FeCap) 589
- ferroelectric capacitor (FeCAP) model 232
- ferroelectric capacitor area 590
- ferroelectric capacitor history 590
- ferroelectric capacitor polarization 595
- ferroelectric film 589, 595, 596
- ferroelectric hysteresis loop 596
- ferroelectric random access memory (FRAM) 589
- FFT 428
 - .fft 207
 - FFT analysis 428, 429
 - FFT analysis points 429
 - FFT output variable 428
 - fgopamp 582
 - files
 - ocean.rc 830
 - fill-ins 346
 - film history 590
- fine optimization of timing, voltage, and current 12

first coupled inductor name 238
 first node name 234
 first sweep variable 457
 first transient simulation 327
 first-order numerical integration algorithm 353
 flash core cells
 flashlevel=1 628
 flash memory 5
 flashelevel=1 628
 flat netlist file 21
 flat parasitic netlists 22
 flattening back-annotation 22
 for loop 573
 .four 428
 fourier analysis 410
 fourier transform 230, 428
 Fowler-Nordheim diode 256
 -fpost 394
 frequency domain 446
 frequency domain analysis 446
 frequency range 446
 frequency response analysis 445
 fromNode 1091
 FSDB 10, 325, 328, 339, 404, 408, 421, 435,
 438, 472, 484, 723, 1017, 1019, 1020,
 1021
 fsdb 120, 436, 437, 438, 523
 ftemperature effect first-order coefficient 234
 full-chip designs 7
 full-chip functionality and timing simulation flow 15
 FullPatchScopenName 733, 735
 FullPathScopeName 733, 735
 fv 498

G

G, voltage-controlled current source 278
 G, voltage-controlled resistor 286
 gain 446
 gain response 445
 gate charge 601, 619
 gate node name 249
 gate-bulk capacitance 601, 619
 gate-bulk overlap capacitance 601, 618
 gate-drain capacitance 601, 619
 gate-drain overlap capacitance 601, 618

gate-level representation 17
 gate-source overlap capacitance 601, 618
 gate-to-drain terminals 91, 92
 gate-to-source terminals 91, 92
 gauss 547
 Gauss window 429
 gds 601, 618
 GDSII layout database 20
 General HSIM Parameters
 HSIMMQS 346
 HSIMSPISCIUNITERR 172
 .palias 346
 generate 573, 574
 geo 250
 .global 330
 global nodes 330
 global-based implementations 13
 globally coupled power nets 22
 .graph 410, 411, 419
 ground bounce 17
 Guassian window 429

H

Hamming window 429
 Hanning window 429
 HdI 729, 730, 731, 733, 734, 735, 738, 739
 .hdl 558
 Help system, opening 208
 hierarchical back-annotation 22, 23
 hierarchical instantiation 581
 hierarchical simulation database 10
 hierarchical simulation engine 22
 hierarchical technology 6
 high impedance node check parameters
 fanout 536
 node1 .. nodeN 536
 title_name 536
 ztime 537
 high precision capacitor 595
 high-sensitivity analog circuit 152, 171
 high-sensitivity analog circuits 354
 high-speed analog circuit simulation 5
 high-speed functionality 91
 high-speed nanometer circuits 7
 high-speed standard cells 17

hmAnalogPortBusValue 666
hmAnalogPortBusValueById 667
hmAnalogPortValue 665
hmAnalogPortValueById 666
hmAnalogStateValueById 689
hmAnalogStateVecValue 689
hmAnalogStateVecValueById 690
hmBiput 651, 654, 655, 679, 680, 681, 688
hmCreateModel 652, 654, 655, 657, 658, 659, 696
hmDefAnalogPort 688
hmDefAnalogPortBus 654, 656, 687
hmDefAnalogState 658
hmDefAnalogStateVec 658
hmDefCurrentPort 657
hmDefDigitalPort 655, 688
hmDefDigitalPortBus 655, 657, 687
hmDefDigitalState 659
hmDefDigitalStateVec 659
hmDefMemCore 696, 697
hmDefVarAnalogPortBus 656, 687
hmDefVarDigitalPortBus 657, 687
hmDigitalPortBusValue 670, 695
hmDigitalPortBusValueById 670, 695
hmDigitalPortValue 669
hmDigitalPortValueById 669
hmDigitalStateValue 691
hmDigitalStateValueById 692
hmDigitalStateVecValue 692
hmDigitalStateVecValueById 692
hmDisableAllPorts 679
hmDisablePortBusDrive 678
hmDisablePortBusDriveById 678
hmDisablePortDrive 677, 688
hmDisablePortDriveById 677
hmEnableAllPorts 679
hmEnablePortBusDrive 678
hmEnablePortBusDriveById 679
hmEnablePortDrive 678, 688
hmEnablePortDriveById 678
hmFree 695
hmGetCurrentPortValue 672
hmGetCurrentPortValueById 673
hmHiZ 677, 688
hmInitMemCore 697
hmInput 654, 655, 677, 688
hmLogic 669, 670, 671, 672, 691, 692, 693, 694, 696
hmLogic aray 672
hmLogic array 672
hmModelFuncName 665
hmModelInstName 664
hmModelInstParamValue 694
hmModelName 665
HMNOTVOUT 652
hmOutput 651, 654, 655, 679, 680, 681, 688
hmPortBusCap 674
hmPortBusCapById 675
hmPortBusSize 687
hmPortBusSizeById 687
hmPortCap 674
hmPortCapById 674
hmPortDir 688
hmPortDirById 688
hmPortId2CktNodeName 664
hmPortId2PortName 662
hmPortName2CktNodeName 663
hmPortName2PortId 662
hmPresentTime 661
hmReadMemCore 698
hmSetAnalogPortBusValue 667
hmSetAnalogPortBusValueById 668
hmSetAnalogPortValue 667
hmSetAnalogPortValueById 667
hmSetAnalogStateValueById 690
hmSetAnalogStateVecValue 690
hmSetAnalogStateVecValueById 691
hmSetCurrentPortValue 673
hmSetCurrentPortValueById 673
hmSetDigitalPortBusValue 672
hmSetDigitalPortBusValueById 672
hmSetDigitalPortValue 671
hmSetDigitalPortValueById 671
hmSetDigitalStateValue 693
hmSetDigitalStateValueById 693
hmSetDigitalStateVecValue 693
hmSetDigitalStateVecValueById 693
hmSetModelAttr 652, 653
hmSetPortBusCap 675
hmSetPortBusCapById 676

hmSetPortBusDelayRes 682
hmSetPortBusDelayResById 683
hmSetPortBusEventDv 684
hmSetPortBusEventDvById 685
hmSetPortCap 675
hmSetPortCapById 675
hmSetPortDelayRes 682
hmSetPortEventDv 683
hmSetPortEventDvById 684
hmSimStage 661
hmStatId2StateName 663
hmStateName2StatId 663
hmWakeUpModel 661
hmWriteMemCore 698
hold 511, 513
hold time check parameters
 edge_type 511
 hold_time 512
 logic_high_voltage 512
 logic_low_voltage 512
 ref_edge_type 511
 ref_logic_high_voltage 513
 ref_logic_low_voltage 513
 ref_name 511
 sig_name 511
 title_name 511
 window_limit 512
hold time error 513
hs2tbl utility 12
HSIM 3
HSIM built-in functions 323
HSIM Verilog-A implementation 554
HSIM_PARSE_ERROR_LIMIT 206
HSIMABMOS 73, 81
hsim.ac_mt 452
HSIMACFREQSCALE 54
HSIMALLOWDDV 584
HSIMALLOWEDDV 60, 93, 343, 354, 356, 576
HSIMAMOS 70, 81
HSIMANALOG 13, 61, 62, 344
HSIMANALOG=1 61, 62
HSIMANALOG=2 61, 62
HSIMBISECTION 66, 67
HSIMBMOS 70, 81
HSIMBUSDELIMITER 71
hsimC_model 651
HSIMCC 73
hsim.cckcapv 933
hsim.cckmosv file 918
hsim.ccktoggle file 1011
hsim.chk 505
HSIMCKTCHECK 893
HSIMCMIN 74
HSIMCOILIB 724
HSIMCONNCHECK 75
HSIMCONNCHECK=0 75
HSIMCONNCHECK=1 75
HSIMCONNCHECK=2 75
hsim.dc 458
hsim.dc_mt 458
HSIMDCI 352
HSIMDCINIT 76, 77, 352
HSIMDCITER 76, 352
HSIMDCOUTFMT 59, 78
HSIMDCSTEP 352
HSIMDCSTOPAT 464
HSIMDCV 77, 352
HSIMDELAYNTLRRMIN 79
HSIMDELVDT 467
HSIMDEVICEV 1041, 1050, 1051
HSIMDIODECURRENT 81
HSIMDPFHIERID 83
HSIMDPFPFX 84
HSIMDPFSCALE=1u 84
HSIMENHANCEDC 13, 352
HSIMENHANCEMOSIV 89
HSIMFCAP 356, 360, 361
HSIMFCAPR 356, 360, 361
HSIMFLAT 89, 355
HSIMFLAT=1 89
HSIMMODLIB 650, 651, 699, 706
hsim.fsdb 328, 1021
HSIMFSDBDOUBLE 404
HSIMGCAP 356, 360, 361
HSIMGCAPR 356, 360, 361
HSIMGCSC 62, 73, 91, 354
HSIMHIERID 94
HSIMHZ 95, 419
HSIMHZALL 96
hsim.ic 325, 328, 339
HSIMIDDQ=1 99

Feedback

Index

H

hsim.ini 206
hsim.ini file 206
HSIMKEEPNODESET 352
hsim.log 216
HSIMLOGICHV 296
HSIMLOGICLV 296
HSIMLUMPCAP=0 105
HSIMLUMPCAP=1 105
HSIMMODELSTAT 108
HSIMMONTECARLO 112
HSIMMQS 346, 354
HSIMMULTIDC 352
HSIMNODECAP 115
HSIMNTLFMT 117
HSIMNTRMIN 79
HSIMNVBS 117, 118
HSIMOcean 830
HSIMOcean API 831
 nsdOcnCreateHostNetlist() 831
 nsdOcnCreateNetlist() 832
 nsdOcnCreateTopNetlist() 831
 nsdOcnEnvSetup() 831
 nsdOcnFinishing() 832
 nsdOcnRunHsim() 832
 nsdOcnSetHSIMParam() 831
HSIMOPTSEARCHEXT 119
HSIMOUTPUT 59, 78, 403, 404, 405, 407, 408, 724, 729
HSIMOUTPUTFLUSH 120
HSIMOUTPUTIRES 414
HSIMOUTPUTt 726
HSIMOUTPUTTBL 409
HSIMOUTPUTVRES 414
HSIMPARAM 12
HsimParam 729
.hsimparam 344, 482
HSIMPORTCR 356, 358, 359
HSIMPORTV 351
HSIMRASPFMODXY 132
HSIMRCRKEEPMODEL 137, 138
HSIMREDEFSUB 139
HSIMREDEFSUB=1 139
hsimrmax 906
HSIMSAMPLERATE 179
HSIMSBAHIERID 390
HSIMSBAMSGLEVEL 389
HSIMSBAMSGLIMIT 390
HSIMSBAPFX 391
HSIMSBASFX 390
HSIMSCALE 147
HSIMSNCs 354
HSIMSPEED 13, 91, 354, 355, 357
HSIMSPEED=0 357
HSIMSPEED=1 357
HSIMSPEED=4 91, 153, 354, 355, 356, 357
HSIMSPEED=5 357
HSIMSPF 155
HSIMSPFTRIPLET 170
HSIMSPF 161, 364
HSIMSPFFDELIM 157
HSIMSPFFEEDTHRU 159
HSIMSPFHLEVEL 159
HSIMSPFMERGEPIN 162
HSIMSPFNETIPIN 162
HSIMSPICE 73, 354, 356
HSIMSTEADYCURRENT 149, 172, 354, 356
HSIMTAUMAX 173, 354, 356
HSIMTIMESCALE 93, 178
HSIMTIMESCALE=0.1 178
HSIMTIMESCALE=10 178
HSIMTIMESCALE=100 178
HSIMTLINEDV 179
HSIMTRAPEZOIDAL=1 353
HSIMTSTEP2TAUMAX 173
HSIMUSEHM 181, 183
HSIMV2S 187
HSIMVDD 60, 419
HSIMVECTORFILE 193, 291
HSIMVERILOGA 558
HSIMVHTH 304, 419, 420
HSIMVLTH 304, 419, 420
HSIMXSW 412
hspice 117
HSPICE netlists
 parse encrypted 231
HSPICE simulator 227
hysteresis loop 589, 590
 lower branch 597
 lower-branch 596
 upper-branch 596

hysteresis loops 591, 594

I

I, current source 272

I/O statement 295

I/O statements 136

.ibs 601, 618

.ic 136, 330

IDDQ 99

IDDQ current 99

IDDQ measurement 99

identical sub-circuit instances 350

.ids 601, 618, 1020

.iev 479

if-else netlist syntax 261

model binning 270

model card definition 268

nested blocks 269

parameter value definition 267

subcircuit definition 266

if-else statement 572

iInductor 446

imaginary part 451

Implicit Backward Euler Algorithm 353

In 435

.include 205, 226, 331, 345, 505

independent current source name 272, 447

independent voltage source name 271, 447

independent voltage sources

229

index number 479

indirect branch assignment statement 584

inductance matrix 242

inductance value key word 237

initial condition, define (.IC) 330

initial current flowing through the inductor 238

initial voltage 591

initial voltage across the capacitor 236

initial_geometry 602, 609

initial_model 602, 606, 609

initial_step 574

initialize 601, 602, 606

input netlist 904

input netlist file 206

input node name 239

input signal reference node name 239, 240

input stimuli 205

instsize 602

integer variables 567

intelligent cross-coupling capacitance handling method 361

interactive circuit analysis 4

interactive circuit debugging environment 463

interactive circuit diagnosis 4

interactive mode 230, 465

interactive mode ap 471, 472

interactive mode command records 473

interactive mode commands

alias 471

ap 471, 479, 483, 484

closelog 472, 473, 497

cont 473

dcon 473, 494

dcpath 474, 476, 488

dn 477

dp 478, 479, 484

eid 479, 480

ename 479

ev 479, 480

exi 480, 481

fmeta 482

fv 482, 483, 497

help 483

iap 473, 483, 484

idp 484

iev 479, 484

inc 485

inv 487

lx 488

matche 488, 489

matchn 489

nc 490, 492

nctr 492

ni 494

nid 494

nname 495

nv 495

op 496

openlog 473, 496, 497

pt 497

quit 497

rcf 497, 498

restart 498

Feedback

Index

J

- rv 482, 483, 498
- savesim 498, 499
- stop 500
- stop -at 500
- stop -delete 500, 501
- stop -list 500, 501
- trace_thru_on 469, 501
- tree 502
- interactive mode debugging command
 - alias 466
 - ap 467
 - closelog 467
 - cont 467
 - dcon 467
 - dcpath 467
 - dn 467
 - dp 467
 - eid 467
 - ename 467
 - ev 467
 - exi 467
 - fcc 467
 - fmeta 467
 - fv 467
- HSIMALLOWEDDV 467
- HSIMDELVDT 467
- HSIMSPICE 467
- HSIMSTEADYCURRENT 467
- iap 467
- idp 467
- iev 467
- inv 468
- lx 468
- matche 468
- matchn 468
- nc 468
- nctr 468
- ni 468
- nid 468
- nm 468
- nname 468
- nv 468
- op 468
- openlog 468
- pt 468
- quit 468
- rcf 468
- restart 468
- rv 468
- savesim 468
- stop 468, 469
- tree 469
- vni 469
- interactive mode dp 471, 472
- interactive mode pt 472
- inter-block interconnects 15
- interface specification, custom output 725
- internal frequency sweep 448
- intiger parameter declarations 566
- inverter model 582
- inv.va file 582
- io 295, 296
- IP cores 5
- IR drop 7, 18
- isomorphic matching techniques 6
- iteration count 10
- iterations (simulation runs) 546
- iterative layout improvement process 23
- ith 527, 530

J

- J, JFET transistor 259
- JFET 228, 446
- JFET model 259, 260
- JFET model parameters 260
- JFET parameters 259
- Juncap capacitance model from Phillips 254
- junction field-effect transistor 228
- junction field-effect transistors 3

K

- Kaiser window 429
- Kaiser-Bessel window 429
- kas1 598
- keyword 448, 449
- keyword sweep 447
- Kirchoff's Flow Law 567

L

- l 249
- L, inductor 237
- laplace transform function

- 579
large circuit blocks 7
large hierarchical resistive ladder 215
large interacting circuit blocks 7
latency setting 149
layout parasitics 18, 20
level 251, 413, 419, 539
.lib 205, 226, 331, 332
libraries
 private 203
 protecting 203
limit 547
Limit distribution using absolute variation 547
limitMos 918, 924, 925, 926
linear analysis 445
linear capacitor 594
linear circuit solution 445, 446
linear devices 445
linear scale 448
list of points 448
listing, suppressing 203
LLTL 228
local feedback coupling 62
localization algorithms 13
lock-in state 355
.log 337
logarithmic scale 448
logic threshold voltage 305
logichv 296, 297, 304
logiclv 296, 297, 304
long interconnects 17
lossless transmission line 228, 239
lossless transmission line element name 239
lossless transmission line parameters 239
lossy multi-conductor transmission lines 240
lossy transmission line 228, 240
lossy transmission line data 243
lossy transmission line element name 240
lossy transmission line model key word 242
lossy transmission line model parameters 242
lossy transmission line parameters 240
lower triangular format 243
low-frequency applications 91
LPE 10
.lprint 419, 489
LVS 20
LYTL 228
- M**
- m 250
M, MOS transistor (MOSFET) 249
m2_load 622, 623
maa 249
macro parameter 354
macro primitives
 bcs 873
 bvs 873
 cap 862
 cccs 867
 ccvs 867
 diode 868
 idc 863
 iexp 865
 ipwl 864
 isin 865
 mind 862
 nbsim 872
 nbsim4 872
 njfet 870
 nmos 870
 nmos4 871
 npn 869
 pbsim 872
 pbsim4 873
 p capacitor 862
 pdiode 868
 pjfet 870
 pmos 871
 pmos4 871
 pnp 870
 presistor 861
 res 861, 875
 schottky 869
 tline 868
 vccap 874
 vccs 866
 vcres 873
 vdc 863
 vpulse 863
 vpwl 864
 zener 869
macro primitives/ind 862

Feedback

Index

M

- macro primitives/ipearl 864
- macro primitives/vcvs 866
- macro primitives/vexp 865
- macro primitives/vsin 865
- Magnitude 451
- magnitude 451, 452
- major hysteresis loop 590
- makefile 739
- .alias 332
- mask 297
- matchport 413, 414, 419
- maximum current value 278, 283
- maximum number of parse errors 206
- maximum voltage value 280
- .mc file 110
- MCA 545, 546
- MCA histogram 111
- .meas 427
- .measure 137, 207, 423, 426, 427, 466, 499, 523, 524
- measure 230
- .measure ac 452
- .measure commands 110
- measure commands 110
- measure statement 111
- measured output 446
- memory core cell port 126
- memory usage 221
- memory utilization 221
- MESFET 228
- MESFET model 260
- MESFET parameters 259
- metaencrypt 231
- metal-oxide-semiconductor field-effect transistor 228
- metal-semiconductor field-effect transistor 228
- meta-stable conditions 482
- Meyer Capacitance Model 619
- microprocessor 5
- Miller Effect 91
- minimum current value 279, 283
- minimum mutual inductance threshold 336
- minimum voltage value 280
- minor loops 590
- mixed-signal circuit simulation 5
- .model 227, 523, 525
- model 524
- b3main.c model 609
- B3SOIPD2.0 252
- B3SOIPD2.2.1 252
- B3SOIPD2.2.2 252
- BJT model 232, 257, 258, 967
- BJT transistor 257
- BSIM1 254
- BSIM1 MOSFET model 231
- BSIM3 model 721
- BSIM3 MOSFET model 231
- BSIM3SOI model 231, 721
- BSIM3v2 252
- BSIM3v3 model 599
- BSIM3v3.0 252
- BSIM3v3.1 252
- BSIM3v3.24 252
- BSIM4 model 721
- BSIM4 MOSFET model 232
- BSIM4.0.0 252
- BSIM4.2.0 252
- BSIM4.2.1 252
- BSIM4.3 252
- BSIM4.4 252
- BSIM4.5 252
- BSIM4SOI 253, 721
- BSIM4SOI Model 232
- C functional model 651, 652
- capacitor model 236, 261
- charte-based capacitance model 619
- Curtice model 260
- device model 545
- diode 256
- diode model 232, 255, 256, 967
- EKV model 253
- EKV MOSFET model 232
- element models 226
- Elmore Delay model 1037
- gate capacitance model 232
- Gummel-Poon model 258
- HICUM 258
- HICUM0 258
- HiSIM MOSFET 232
- HiSIM1.0.0 253
- JFET model 232, 260
- Level-6 SSIM model 254
- lossy transmission line 241, 242

lossy transmission line model 241, 242
MESFET model 259
MEXTRAM BJT model 721
Mextram model 258
Meyer capacitance model 601
meyer capacitance model 622
MOS11 model 721
MOS9 model 721
MOSFET 251
MOSFET DC model 171
MOSFET Gate Capacitance model 171
MOSFET Level 2 model 599
MOSFET model 249, 251, 599, 967
MOSFET models 252
MOSFET table model 170, 171
Motorola SSIM model 253
Motorola SSIM SOI 254
n-JFET model 260
npn transistor model 258
Philips MOS11 model Level 1100 253
Philips MOS11 model Level 1101 253
Philips MOS9 model 253
Philips MOSFET models 232
p-JFET model 260
pnp transistor model 258
precise model 52, 60, 70, 349
PSP 253
PSP model 232
RAM model 657
resistor model 234
RPI TFT 253
simple model 349
SPICE Level 2 model 254
SPICE Level 3 model 254
SPICE Level-3 MOSFET model 232
SPICE model 232
Statz model 260
table model 171
UCB SPICE model 721
UMI model 599
VBIC BJT model 721
VBIC model 258
model b3 600
model format selector 242
model library 331
model m115 600
model m2 600
model name 829, 830

model optimization reference name 525
model parameter 251
model parameter variation 547
model scaling factor 336
.model statement 523
model_load 602, 618
model_name 249, 251
models
 private 203
 protecting 203
modelsize 601
modify parameter values 326
modulation frequency 275
modulation index 275
modules using the same name 581
Monte Carlo analysis 3, 545
MOS transistor 451
MOS transistors 491, 492
MOS11 model 254
MOS9 model 254
MOSFET 446
MOSFET capacitances 91
MOSFET channel length 335
MOSFET default channel width 335
MOSFET drain bulk junction diode perimeter 335
MOSFET drain parameter 908
MOSFET drain resistor squares 335
MOSFET drain terminals 127
MOSFET drain/source sharing selector 250
MOSFET element geometry order changes 337
MOSFET element name 249
MOSFET gate capacitance 91
MOSFET gate length 249
MOSFET gate width 249
MOSFET model 170
MOSFET Model evaluation 171
MOSFET model level selector 251
MOSFET model name 249, 251
MOSFET model parameters 251
 ako 251
 encmode 251
 level 251
 model_name 251
 nmos 251
 parameter 251

Feedback

Index

N

pmos 251
MOSFET parameters
ad 250
as 250
delvto 250
dtemp 250
geo 250
l 249
m 250
maa 249
model_name 249
nbu 249
ndr 249
nga 249
nrd 250
nrs 250
nso 249
off 250
pd 250
ps 250
rdc 250
rsc 250
w 249
MOSFET source area 907
MOSFET source bulk junction diode perimeter 335
MOSFET source parameter 908
MOSFET source resistor squares 335
MOSFET source terminals 127
MOSFET table model 171
MOSFET table modell 171
MOSFETdrain area 907
MOSPRECISION 13
Motorola 254
MPEG 5
.mt 523
multiple references to the same module 581
multiwire lossy transmission line model 241
mutual coupling coefficient value key word 238
mutual inductor 446
mutual inductor element name 238
mutual inductor parameters 238
MYFORMAT 726, 739
myformat.c 739
myformat.h 739

N

name1 419
nanometer effects 7
nassda control and data file 408
Nassda Output Format 11, 405
Native Netlist Integration (AANNI) 793
nature statements
 584
 ddt_nature 584
 ldt_nature 584
natures 567, 581
nbu 249
ndr 249
negative node name 236, 237, 255, 271, 272,
 278, 280, 283, 285, 286, 288, 590
net name 828, 829
netlist syntax
 asterisk symbol 226
 back slash 226
 blank spaces 233
 buffer size 226
 characters 226
 commas 233
 comment line 226
 control lines 227
 dollar symbol 226
 double back slash 226
 element lines 227
 if-else 261
 key identification character 226
 line continuation 233
 line continuation symbols 226
 line placement 116, 227
 line sequence 227
 period 227
 period symbol 227
 plus sign 226, 233
 title line 226
nga 249
nm 495
nmos 251
n-MOSFET key word 251
nodal analog voltage waveforms 6
nodal digital logic-state waveforms 6
node 568
node capacitance 4, 149
node index 494

node voltage 4, 451
 node voltage waveform 411
 node voltages 416
 .nodeset 136, 333
 NOF 405
 NOF formatted output files 408
 noise analysis 230
 noise functions
 584
 flicker noise 585
 white noise 584
 nominal temperature 601, 617
 non-constant expression error messages 579
 nonlinear devives 445
 non-quasi-static (NQS) 254
 non-quasi-static (NQS) model 254
 non-synthesizable logic 17
 Non-zero AC values 449
 normalized electrical length 239
 normalized lower-branch curve-fit parameters
 a1 597
 a2 597
 b1 597
 b2 597
 c1 597
 c2 597
 d0 597
 NQSMOD=1 254
 nrd 250
 nrs 250
 nso 249
 ntrig 1017
 Numerical scheme tolerance support 584
 nWave 6, 10, 404, 478, 489, 1019
 nWave waveform display tool 404
 n-wire transmission line behavior equations 241

O

occ 72
 Ocean script 830
 .oceanrc file 830
 oct 454
 odel 525
 off 250
 off-diagonal entries 346

offset value 275
 online help, opening 208
 onset ramp delay time 273
 .op 333
 openlog 473
 operating point 446
 operating systems
 Windows 2000 222
 operator
 ddt 576
 delay 577
 idt 576
 transition 577
 opt 524
 optimization goals 20
 optimization parameter reference name 524
 .option 335
 .option post 411
 option statement parameters
 335
 optxxx 524
 OPTxxx parameter 524
 .optz file 523
 oscilloscope 595
 otc 72
 .out 408, 523, 729
 out 408
 out_file.chk 505, 532
 out_file.fsdb 207
 outer sweep 449
 outer sweeps 449, 450
 outfile.ac 450
 outfile.ac_mt 452
 outfile.dc 458
 outfile.dc_mt if 458
 output conductance 601, 618
 output format
 ASCII 723
 FSDB 10, 325, 328, 339, 404, 408, 421, 435,
 438, 472, 484, 723, 1017, 1019, 1020,
 1021
 Nassda Output Format 724
 WDF Sandwork Output Format 724
 WSF 723
 output node name 239
 output signal reference node name 239, 240

Feedback

Index

P

output specification 205
output syntax 451
output variable 451
output variable modifier 451
output variables 450
output voltage resolution 123
output.cckmosv file 918
output.ccktogle file 1011
outputs 414
ov1 419
ov2 419

P

.param 71, 227, 337, 524
PARAM statement 337
.param statement 523
parameter 251
remove setting 327
repeat values 325
parameter extraction measurement 594
parameter name 448
parameter value 453
parasitic capacitors 364
parasitic database 21
parasitic extraction process 21
parasitic RC reduction 9
parasitic RC values 157, 170
parasitic RCs 363
parasitic resistors 364
parasitics 15, 17, 21, 22
PARPRECISION 13
partitioned sub-circuit 21
pass gate transistors 1004
passfail 523
pause simulation 499
.pcheck 505, 527, 530, 532
.pcheck window 538
pd 250
performance/precision trade-offs 12
period 298, 301, 527
phase 449, 451, 452
phase delay 274
phase response 445
Philips MOSFET model
MOS9 232

Philips MOSFET models
MOS11 232
pHMMODEL 652
physical layout verification 20
piecewise constant (PWC) 345
piecewise constant values 356
piecewise linear controlling function 279, 281, 283, 288
piecewise linear controlling functionn 285, 287
piecewise linear corner control parameter 279, 281, 283, 285, 287, 288
piecewise linear function 450
piecewise linear source function keyword 277
PLL 345
PLL circuit 61, 345
PLL circuit lock-in stage. 345
PLL circuit simulation 355
.plot 410, 411, 419
Pmax 594, 597, 598
Pmin 597
pModel 601
pmos 251
p-MOSFET key word 251
poi 454
points per decade 448
polarization 591, 594
pole-zero analysis 230
polynomial coefficients 279, 281, 285, 286
polynomial controlling function 279, 281, 283, 285, 286
polynomial dimension for controlling sources 279, 281, 283, 285, 286
port capacitance 351
port control parameters
port capacitance
HSIMPORTCR 351
port current tolerance
HSIMPORTI 351
port voltage tolerance
HSIMPORTV 351
port voltage toleranceHSIMPORTV 351
port definition 651
port voltage tolerance 351
positive gate-source capacitance 601, 619
positive node name 236, 237, 255, 271, 272, 280, 283, 285, 286, 287, 590

positive or negative controlling node 278, 280, 285, 286, 288
post layout design flow 18
post layout indicator 12
post-layout design flow 15, 18, 20
post-layout netlist types 364
post-layout parasitics 5
post-layout simulation 363
-posttop 394
power analysis 230
power bus sizing 17
power characterization 5
power net interconnect parasitics 23
power net IR drop 5
power net reduction 22
power net topology 22
pRam 652, 654
precision degradation 222
precision-speed trade-off 356
pre-layout design flow 15, 20
pre-layout hierarchical netlist 20, 364
pre-layout schematic netlist 10
.print 136, 410, 411, 413, 414, 416, 419, 523
.print ac 450, 451
.print dc statement 458
print out variables/parameters 457
.print window 430
printout
 suppressing 203
.probe 410, 411, 419
procedural assignment statements 571
procedural interface 725
proprietary MOSFET models 599
.PROTECT command 203
protecting data 203
ps 250
pt 472
pulse fall time 273, 276
pulse peak value 273, 276
pulse period 273
pulse rise time 273
pulse source function 273
pulse source initial value 273
pulse width 273

pulse width check parameters
 high_max_time 516
 high_max_voltage 516
 high_min_time 514, 516
 logic_high_voltage 515
 logic_low_voltage 515
 low_max_time 515
 low_min_time 515
 low_min_voltage 516
 sig_name 515
 title_name 515
pulse width violation error 517
P-V curve 594
P-V hysteresis loops 589
PWC 345
pwc 345, 356
PWL 276, 435, 438
.pwl 438
pwl 356
PWL (piecewise linear) source function 276, 277
PWL voltage source inputs 214
PWLZ 277
PWLZ element 229
PWLZ voltage source 221

Q

qd 601, 619
qg 601, 619
qs 601, 619
query commands 479

R

radix 298, 299
RAM 650
RAM example 704
ram_function 652
random number distribution 546
.rc 478
RC back-annotation 157, 168
RC in DSPF format 154, 155
RC in SPEF format 154, 155
RC nodes 136
RC reduction 9, 21, 22, 127, 136, 137, 138, 145
rdc 250
read_model 602, 614

Feedback

Index

S

real part 451
real variables 567
reference node transition time 506
reference voltage 304
reltol 584
repeat 573
repeat function keyword 277
repeat function start time 277
reports
 functionality 5
 power analysis 5
 timing 5
reserved words
 425
 AND 279, 281, 284, 285, 287
 integ 425
 max 425
 min 425
 NAN 281, 284, 285, 287
 NOR 279, 281, 284, 285, 287
 OR 279, 281, 284, 285, 287
 pp 425
 rms 425
resistance 299
resistance matrix 242
resistance value keyword 234
resistor 446
 voltage-controlled 286
resistor element name 234
resistor ladder 213
resistor ladder test case
 tutorial/2br script 215
resistor ladder test results 216
resistor length 235
resistor model name 234
resistor parameters 234
resistor reduction 22
resistor width 235
resistor.va 560
result 524
rise 297, 301, 532
risePmosFallNmos 918, 924, 926
rising edge delay time 276
rising edge time constant 276
RLC elements 136, 137, 138
RLGC file 243

RLGC model 243
RMIN elimination 22
ROM 5, 153
rptTrace 981, 982
.rs# 498
rsc 250

S

samp 344, 416
saturated hysteresis loop 596
saturated loop 594, 596
saturated loop parameters
 as1 596, 597, 598
 as2 596, 597, 598
 bs1 596, 597, 598
 bs2 596, 597, 598
 cs1 596, 597, 598
 cs2 596, 597, 598
 ds0 596, 597, 598
saturated parameters
 596
saturation voltage 601
Sawyer-tower circuits 594
scalar branches 568
scalar parameters 566
ScopeSeparator 733, 735
scripts
 HSIMOcean 830
 ocean 830
search path for included files 336
search path for model libraries 336
second coupled inductor name 238
second node name 234
second-order numerical integration algorithm 179, 353
segmentation resolution 21
self inductor parameters 237
self-contained netlist
 back-annotation netlist
 parasitic RC netlist 364
self-inductor element name 237
semi-custom designs 17
sense amplifier operations 219
sensitivity analysis 230
set_model 602, 615
setup time 506

setup time check 507, 510
 setup time check parameters
 edge_type 507
 logic_high_voltage 509
 logic_low_voltage 509
 ref_edge_type 508
 ref_logic_high_voltage 509
 ref_logic_low_voltage 509
 ref_name 507
 setup_time 508
 sig_name 507
 title_name 507
 window_limit 508
 SFFM source function 274
 shunt conductance matrix 242
 signal 299, 300
 signal amplitude 275
 signal edges 1054
 signal frequency 275
 signal information file 306
 signal information file example 319
 signal information file, VCD 306
 signal net partitions 22
 signal node 297
 signal node transition time 506
 signal nodes 297, 346
 signal, falling slope 1054
 signal, rising slope 1054
 signal|vname 299, 300
 SignalName 733, 735
 SignalType 733
 Simple rectangular truncation window 429
 simplified MOSFET table model 354
 Simulation and Control Statements
 .ends 328
 .force 329, 338
 .global 330
 .ic 330
 .include 331
 .lib 331
 .malias 332
 .nodeset 333
 .op 333
 .option 334, 335
 .param 337
 .release 329, 337, 338
 .subckt 338
 .temp 339
 .tran 339
 simulation control parameters 226
 simulation file global parameters
 check_window 292
 delay 291
 fall 291
 logichv 292
 logiclv 292
 period 292
 resistance 292
 rise 291
 slope 292
 tunit 292
 vhth 292
 simulation flow 9, 10
 simulation log 581
 simulation reference temperature 336
 simulation results output files 207
 simulation runs (iterations) 546
 simulation speed limitations 583
 simulation statistics log files 207
 simulation temperature 601, 617
 SimWave 6
 single-frequency FM source function 274
 sinusoidal source function 273, 274
 skin-effect resistance matrix 242
 skipInst 918, 924, 925, 926, 956, 960, 973, 977
 slew 577
 slope 297, 301
 small-signal response 445
 SoC 22
 soft-upper bound 906
 sorting keys
 el_name 1044
 err_v 1044
 node1 1044
 node2 1044
 t1 1044
 source
 current-controlled current source 282, 283
 current-controlled voltage source 284
 independent current 272
 independent voltage 271
 voltage-controlled current source 278
 voltage-controlled voltage source 279, 280
 source amplitude value 274, 275

Feedback

Index

S

- source capacitance 359
- source diffusion area 250
- source frequency 274
- source function
 - amplitude modulation 275
 - exponential 276
 - piecewise linear 276, 277
 - piecewise linear high-impedance 277
 - pulse 273
 - single-frequency FM 274
 - sinusoidal 273, 274
- source magnitude 446, 449
- source node name 249
- source offset value 274, 275
- source phases 449
- source terminals 52, 70, 171
- Source Value 453
- source value 453, 456, 457
- source-bulk junction diode area 335
- source-bulk junction perimeter 250
- source-diode area 616
- source-diode periphery 616
- specifying parameters
 - attach parameter to a port list in the sub-circuit
 - definition 343
 - external files 341, 345
 - global parameter 342
 - global parameters 344
 - local parameter 342
 - setting a parameter at global and sub-circuit
 - levels 344
- sub-circuit 341
- top level netlist 341, 342
- top level netlist file 341
- SPEF 9, 21, 22, 71
 - parasitic RC values 157, 170
- SPEF file for back-annotation 155
- SPF file 364
- SPICE 430
- SPICE netlist 71, 650
- SPICE netlist file 181, 183, 650
 - converters 699
 - RAM 706
- SPICE parser 345
- spice primitives 582
- SPICE simulation options
 - 334
- aspec 334, 335
- badchr 334, 335
- defad 334, 335
- defas 334, 335
- defl 334, 335
- defnrd 334, 335
- defnrs 334, 335
- defpd 334, 335
- defps 334, 335
- defw 334, 335
- genk 334, 335
- global 336
- klim 334, 336
- nowarn 334, 336
- parhier 334
- Parhier=global 336
- Parhier=local 336
- scale 334, 335
- SCALE option 336
- scalm 334, 335, 336
- search 334
- search=dir_path 336
- spice 334
- SPICE option 336
- tnom 334, 336
- warnlimit 334, 337
- wl 334, 337
- SPICE simulator 227, 342, 345
- SPICE-compatible netlists 205
- SRAM 5, 221, 589
- SRAM cell 62
- SRAM circuit 213
- SRAM circuit construction 219
- SRAM circuit elements
 - memory cells 219
 - read/write (R/W) 219
 - sense amplifiers 219
- SRAM test case 219
- SRAM tutorial
 - tororial/SRAM directory 220
- standard parasitic exchange format (SPEF) 9
- STARC/Hiroshima University 254
- start 523, 527, 530, 532, 539, 602, 609
- start, stop 534, 537
- statements
 - .BIASCHK 46
 - .PROTECT 203
 - .UNPROTECT 204

- steady current tolerance 149
 steady state convergence 10
 steptime 524
 stimulus input sources 226
 stimulus output specifications 226
 stop 523, 527, 530, 532, 539
 stop_at_error 301
 stop= 532
 stoptime 524
 .store 420, 498
 store simulation control parameters 465
 sub_name 344
 sub-circuit definition 338
 sub-circuit library 17
 subcircuit_instance_name 344
 subcircuit-based implementations 13
 .subckt 227, 338
 subckt 137, 186, 328, 330, 333, 344, 352, 354, 355, 413, 419, 442, 443, 444, 474, 502, 503, 507, 510, 511, 512, 515, 519
 sub-loops 590
 substrate forward bias 999
 substrate-drain junction diode capacitance 601, 619
 substrate-drain junction diode current 601, 618
 substrate-source junction diode capacitance 601, 619
 substrate-source junction diode current 601, 618
 sweep 447, 448, 454, 457
 sweep syntax 339
 sweep variable 457
 sweep voltage source 457
 syntax of a specific command 483
 synthesizable logic 17
 synthesizable logic flow 17
 synthsizable logic 17
 systems-on-chip 22
- T**
- T, lossless transmission line 239
 Table Model Control Parameter
 HSIMAUTOVDD 60, 420
 HSIMNVDS 117
 HSIMVBSEND 118
 HSIMVDD 60, 420
- HSIMVDSEND 118
 HSIMVGSEND 118
 tabular format 320
 tag 1043
 target voltage value 466, 499
 .tbl 436
 .tcheck 505
 .tcheck window 523
 TCL 465
 TCL commands
 !! 465
 !# 465
 tdelay 294
 .temp 339
 temp 601, 617
 temp_geometry 602, 617
 temperature effect first-order coefficient 236, 237, 288
 temperature effect second-order coefficient 234, 236, 238, 288
 temperature value 454
 temperature, define 339
 template library 724
 terminals 346
 tfall 301
 THD 429
 The 357
 threshold current 528
 threshold voltage 601, 618
 threshold voltages, digital 304
 time delay between two specified nodes 518
 time step adjustment 113
 time stepsize 149
 time wheel event 999
 time window 293
 timer 575
 timestep 574
 timing analysis 230
 timing behavior assessment 17
 timing characterization 5
 timing edge check parameters
 edge_type 519
 logic_high_voltage 520
 logic_low_voltage 520
 max_time 520
 min_time 520

Feedback

Index

U

ref_edge_type 520
ref_logic_high_voltage 520
ref_logic_low_voltage 520
ref_name 519
sig_name 519
title_name 519
trigger value 521
window_limit 520
timing shift 294
timing-based sub-flow 22
tnom 601, 617
Tool Command Language 465
top level stimuli 20
top-level full-chip simulation 20
top-level input stimuli 15
topological connectivity 21
top.spi 444
total harmonic distortion 429
touchstone format 248
.tran 255, 326, 339, 429, 523, 524
tran command 173
.tran statement 523
.tran statement parameters 339
transconductance 601, 618
transfer gate 1090
transient analysis 110, 230, 451, 452, 458, 545
transient analysis, define 339
transient component 446
transient function 449
transient simulation 10, 99, 113, 347, 999
transient simulation requirements
 216
 average memory usage 216
 peak memory usage 216
 time 216
transient simulation support 584
transient source function 271, 272
transistor instance struct 602
transistor netlist 364
 extracted netlist 364
 pre-layout hierarchical netlist 364
transistor-level representation 17
transistors 364
transistor-specific parameters 345
transition error messages 577
transmission delay time 239
transmission line
 lossless 239
 lossy 240
transmission line input node name 240
transmission line length 239, 240
transmission line model name 240, 242
transmission line normalized electrical length 239
transmission line output node name 240
Trapezoidal Algorithm 179, 353
trise 301
tristate impedance 302
triz 302
tskip 298
tth 530
tunit 193, 301, 302, 303

U

UIC keyword 340
UMI 599
 HSIMUSERLIB 184
UMI approach 254
UMI.c 599, 606
UMI.c file 606, 607
UMI.h 599, 601, 615, 616
UMI.h file 602, 603
UMI.h header 609
UMI.h header file 615, 616
ungrounded capacitor terminals 346
unif 547
Uniform distribution using absolute variation 547
Uniform distribution using relative variation 547
.UNPROTECT command 204
unsaturated loop 594
unsaturated loop parameters
 au1 596, 598
 au2 596, 598
 bu1 596, 598
 bu2 596, 598
 cu1 596, 598
 cu2 596, 598
 du0 596, 598
unsupported features 584
user model interface (UMI) 599
USER_MOSDEF 601
user-defined model 608, 609

- user-defined models 606
- user.so 606
- user.so dynamic library 184, 600, 601, 602, 606, 616, 618
- utf 409
- utime 532
- V**
- v2s 186, 442
- variables 567
- vbs 601, 618
- VCC 229
- VCC parameters 287
- VCCS 229, 446
- VCCS parameters 278
- VCCS syntax statements 278
- VCD
 - bus syntax 307
 - signal information file 306
- VCD direct read 306
- VCD file 306
- VCD file example 317
- VCD file, signal information 306
- VCO 61
- VCR 229
- Vcr 594
- VCR parameters 286
- Vcr, 594
- VCVS 229, 446
- VCVS syntax statements 279
- VDDmax 596
- vds 601, 618
- vdsat 601
- VEC CHECK 137
- vector data section 291
 - signal states at specified times 291
- vector definition section 291
 - cycle period 291
 - driving strength 291
 - rise/fall time 291
 - vector stimulus 291
 - vector type 291
- VECTOR input element 229
- vector node 568
- VECTOR output automatic comparison 229
- Verilog input files 442
- Verilog to Spice conversion 442
- Verilog-A
 - case sensitivity 565
- Verilog-A compiler 554
- Verilog-A Interactive Command
 - ev 583
 - iev 583
 - inc 584
 - nc 584
- Verilog-A supported statement types
 - 570
 - block 570
 - branch contribution 570
 - case 571
 - for loop 571
 - generate 571
 - if-else 571
 - procedural assignment 570
 - repeat loop 571
 - while loop 571
- very large MOSFET width 905
- very large system integration 15
- vgs 601, 618
- VHth 419
- vhth 297, 303, 304, 517, 532
- vih 296
- vil 296
- Virtuoso Analog Design Environment 834
- VLSI design 15
- VLth 419
- vlth 292, 297, 303, 304, 419, 517, 532
- Vmax 594
- Vmax 594, 596, 597
- vmax 591
- vname 299, 412
- voh 303, 304
- vol 303, 304
- voltage check, static 917
- voltage control oscillators 61
- voltage controlled capacitor keyword 288
- voltage divider 215
- voltage divider resistor branches 215
- voltage gain factor 280
- voltage node 466, 499
- voltage output keyword 281

Feedback

Index

W

voltage source 446, 450, 451, 453, 454, 455
 current-controlled voltage source 446
 voltage-controlled voltage source 446
voltage source elements 435
voltage source name for the controlling current to
 flow 283
voltage source node 105, 347
voltage source node current 347
voltage sources statement 449
voltage-controlled capacitor 229
voltage-controlled capacitor name 287
voltage-controlled current source 229
voltage-controlled current source name 278
voltage-controlled resistor 229
voltage-controlled resistor keyword 285, 286
voltage-controlled switches 445
voltage-controlled voltage source 229, 280
voltage-controlled voltage source keyword 280
voltage-controlledresistor name 285, 286
voltage-to-current transfer factor 278
von 601, 618
vref 304
Vsat 594
Vsource 595
Vsrc flag 474
vth 305

W

w 249
W, lossy transmission line 240
warning limiter 337
warning message 457
waveform display tool 215
waveform display tools 12
waveform file
 analog data 737, 738
 close 738
 format 739
 functions 728
 header 730

waveform handle 733
waveform viewers 6
 nWave 6, 10, 404, 478, 489, 1019
 SimWave 6
 Waveview 6
WaveHdl 733, 735
Waveview 6
WDF 407
wdf 407
.wdf output file 407
-webhelp option 208
while 573
Window
 Bartlett triangular window 429
 Blackman window 429
 Blackman-Harris window 429
 Gaussian window 429
 Hamming window 429
 Hanning window 429
 Kaiser-Bessel window 429
 Simple rectangular truncation window 429
Windows NT/Windows 2000 12
wire capacitors 17
write cycle 221
WSF 408
wsf 408
.wsf output file 408

X

X state 419
X windows 12
X_Value 738
Xgraph 12, 405
xsubckt 536, 537

Z

zero polarization point 594
zero-bias threshold voltage shift 250