NAME

irsim – An event-driven logic-level simulator for MOS circuits

SYNOPSIS

irsim *[-s] prm_file sim_file ... [+hist_file] [-cmd_file ...]*

DESCRIPTION

IRSIM is an event-driven logic-level simulator for MOS (both N and P) transistor circuits. Two simulation models are available:

switch

Each transistor is modeled as a voltage-controlled switch. Useful for initializing or determining the functionality of the network.

linear

Each transistor is modeled as a resistor in series with a voltage-controlled switch; each node has a capacitance. Node values and transition times are computed from the resulting RC network, using Chorng-Yeoung Chu's model. Chris Terman's original model is not supported any more.

If the **-s** switch is specified, 2 or more transistors of the same type connected in series, with no other connections to their common source/drain will be *stacked* into a compound transistor with multiple gates.

The **prm_file** is the electrical parameters file that configure the devices to be simulated. It defines the capacitance of the various layers, transistor resistances, threshold voltages, etc... (see presim(1)). If *prm_file* does not specify an absolute path then IRSIM will search for the *prm_file* as follows (in that order):
1) *./<prm_file>* (in the current directory).
2) **~cad/lib/***<prm_file>*
3) **~/cad/lib/***<prm_file>***.prm**

If **~cad/** does not exist, IRSIM will try to use directory **/projects/cad**. The default search directory (~cad) can be overriden by setting the environment variable CAD_HOME to the appropriate directory prior to running IRSIM (i.e. setenv CAD_HOME /usr/beta/mycad).

IRSIM first processes the files named on the command line, then (assuming the exit command has not been processed) accepts commands from the user, executing each command before reading the next.

File names NOT beginning with a '-' are assumed to be sim files (see sim(5)), note that this version does not require to run the sim files through presim. These files are read and added to the network database. There is only a single name space for nodes, so references to node "A" in different network files all refer to the same node. While this feature allows one to modularize a large circuit into several network files, care must be taken to ensure that no unwanted node merges happen due to an unfortunate clash in names.

File names prefaced with a '-' are assumed to be command files: text files which contain command lines to be processed in the normal fashion. These files are processed line by line; when an end-of-file is encountered, processing continues with the next file. After all the command files have been processed, and if an "exit" command has not terminated the simulation run, IRSIM will accept further commands from the user, prompting for each one like so:

irsim>

The **hist_file** is the name of a file created with the *dumph* command (see below). If it is present, IRSIM will initilize the network to the state saved in that file. This file is different from the ones created with the ">" command since it saves the state of every node for all times, including any pending events.

This version supports changes to the network through the **update** command. Also, the capability to incrementally re-simulate the network up to the current time is provided by the **isim** command.

COMMAND SUMMARY

@ *filename*
       take commands from command file
? *wnode...*   **print info about node's source/drain connections**
! *wnode...*   **print info about node's gate connections**
< *filename*   **restore network state from file**
> *filename*   **write current network state to file**
<< *filename*   **same as "<" but restores inputs too**
| *comment...*   **comment line**
activity *from [to]*   **graph circuit activity in time interval**
ana *wnode...*   **display nodes in analyzer window**
analyzer *wnode...*   **display nodes in analyzer window**
assert *wnode [m] val*   **assert that** *wnode* **equals** *value*
back *[time]*   **move back to** *time*
c *[n]*   **simulate for** *n* **clock cycles (default:1)**
changes *from [to]*   **print nodes that changed in time interval**
clock *[node [val]]*   **define value sequence for clock node**
clear   clear analyzer window (remove signals)
d *[wnode]...*   **print display list or specified node(s)**
debug *[debug_level...]*
       set debug level (default: off)
decay *[n]*   **set charge decay time (0 => no decay)**
display *[arg]...*   **control what gets displayed when**
dumph *filename...*   **write net history to file**
exit *[status]*   **return to system**
flush *[time]*   flush out history up to *time* (default: now)
h *wnode...*   **make node logic high (1) input**
has_coords   print YES if transistor coordinates are available
inputs   print current list of input nodes
ires *[n]*   **set incremental resolution to** *n* **ns**
isim *[filename]*   **incrementally resimulate changes form** *filename*
l *wnode...*   **make node logic low (0) input**
logfile *[filename]*   **start/stop log file**
model *[name]*   **set simulation model to** *name*
p   step clock one simulation step (phase)
path *wnode...*   **display critical path for last transition of a node**
print *comment...*   **print specified text**
printp   print a list of all pending events
printx   print all undefined (X) nodes
q   terminate input from current stream
R *[n]*   **simulate for** *n* **cycles (default:longest sequence)**
readh *filename*   **read history from** *filename*
report*[level]*   **set/reset reporting of decay events**
s *[n]*   **simulate for** *n* **ns. (default: stepsize)**
stepsize *[n]*   **set simulation step size to** *n* **ns.**
set *vector value*   **assign** *value* **to** *vector*
setlog*[file|off]*   **log net changes to file (***off* **-> no log)**
"setpath" "*[path...]***"**
       set search path for cmd files
stats   print event statistics
t *[-]wnode...*   **start/stop tracing of specified nodes**
tcap   print list of shorted transistors
"time *[command]* **print resource utilization summary**
u *wnode...*   **make node undefined (X) input**
unitdelay *[n]*   **force transitions to take** *n* **ns. (0 disables)**
update *filename*   **read net changes from file**
V *[node [value...]]*   **define sequence of inputs for a node**
vector *label node...*   **define bit vector**
w *[-]wnode...*   **add/delete nodes from display list**
wnet *[filename]*   **write network to file**
x *wnode...*   **remove node from input lists**
Xdisplay*[host:n]*   **set/show X display (for analyzer)**

COMMAND DESCRIPTIONS

Commands have the following simple syntax:

cmd *arg1 arg2 ... argn* **<newline>**

where **cmd** specifies the command to be performed and the *argi* are arguments to that command. The arguments are separated by spaces (or tabs) and the command is terminated by a **<newline>.**

If **cmd** is not one of the built-in commands documented below, IRSIM appends ".cmd" to the command name and tries to open that file as a command file (see "@" command). Thus the command "foo" has the same effect as "@ foo.cmd".

Notation:

> **...**
> **indicates zero or more repetitions**

[ ] enclosed arguments are optional

node name of node or vector in network

wnode name of node or vector in network, can include **'*'** wildcard which matches any sequence of zero or more characters. The pair of characters **'{'** and **'}'** denote iteration over the limits enclosed by it, for example: **name{1:10}** will expand into *name1, name2 ... name10.* A 3rd optional argument sets the stride, for example: **name{1:10:2}** will expand into *name1, name3, ... name7, name9.*

| comment...
> Lines beginning with vertical bar are treated as comments and ignored -- useful for comments or temporarily disabling certain commands in a command file.

Most commands take one or more node names as arguments. Whenever a node name is acceptable in a command line, one can also use the name of a bit vector. In this case, the command will be applied to each node of the vector (the "**t**" and "**d**" treat vectors specially, see below).

vector *label node...*
> **Define a bit vector named "label" which includes the specified nodes. If you redefine a bit vector, any special attributes of the old vector (e.g., being on the display or trace list) are lost. Wild cards are not accepted in the list of node names since you would have no control over the order in which matching nodes would appear in the vector.**

The simulator performs most commands silently. To find out what's happened you can use one of the following commands to examine the state of the network and/or the simulator.

set *vector value*
> **Assign** *value* **to** *vector.* For example, the following sequence of commands:

> **vector** BUS bit.1 bit.2 bit.3 **set** BUS 01x

> The first command will define *BUS* to be a vector composed of nodes *bit.1, bit.2,* and *bit.3.* The second command will assign the following values:

bit.1 = 0
bit.2 = 1
bit.3 = X

> Value can be any sequence of [0,1,h,H,l,L,x,X], and must be of the same length as the bit vector itself.

d *[wnode]...*
> **Display. Without arguments displays the values all nodes and bit vectors currently on the display list (see w** command). With arguments, only displays the nodes or bit vectors specified. See also the "display" command if you wish to have the display list printed out automatically at the end of certain simulation commands.

w *[-]wnode...*
> **Watch/unwatch one or more nodes. Whenever a "d" command is given, each watched node will displayed like so:**
>
> **node1=0 node2=X ...**
>
> To remove a node from the watched list, preface its name with a '-'. If *wnode* is the name of a bit vector, the values of the nodes which make up the vector will be displayed as follows:
>
> **label=010100**
>
> where the first 0 is the value of first node in the list, the first 1 the value of the second node, etc.

assert *wnode [mask] value*
> **Assert that the boolean value of the node or vector** *wnode* is *value*. If the comparison fails, an error message is printed. If *mask* is given then only those bits corresponding to zero bits in *mask* take part in the comparison, any character other than 0 will skip that bit. The format of the error message is the following:

(tty, 3): assertion failed on 'name' 10X10 (1010X)

> Where *name* is the name of the vector, followed by the actual value and the expected value enclosed in parenthesis. If a *mask* is specified, then bits that were not compared are printed as '-'.

ana *wnode...*
> **This is a shorthand for the analyzer command (described below).**

analyzer *wnode...*
> **Add the specified node(s)/vector(s) to the analyzer display list (see irsim-analyzer(3) for a detailed explanation). If the analyzer window does not exist, it will be created. If no arguments are given and the analyzer window already exists, nothing happens.**

Xdisplay *[host:display]*
> **You must be able to connect to an X-server to start the analyzer. If you haven't set up the DISPLAY** environment variable properly, the analyzer command may fail. If this is the case you can use the **Xdisplay** command to set it from within the simulator. With no arguments, the name of the current X-server will be printed.

clear Removes all nodes and vectors from the analyzer window. This command is most useful in command scripts for switching between different signals being displayed on the analyzer.

"**?**" and "**!**" allow the user to go both backwards and forwards through the network. This is a useful debugging aid.

? *wnode...*
> **Prints a synopsis of the named nodes including their current values and the state of all transistors that affect the value of these nodes. This is the most common way of wandering through the network in search of what went wrong. The output from the command** *? out* looks like

out=0 (vl=0.3 vh=0.8) (0.100 pf) is computed from:
n-channel phi2=0 out=0 in=0 [1.0e+04, 1.3e+04, 8.7e+03]
pulled down by (a=1 b=1) [1.0e+04, 1.3e+04, 8.8e+03]
pulled up [4.0e+04, 7.4e+04, 4.0e+04]

> The first line gives the node's name and current value, its low and high logic thresholds, user-specifed low-to-high and high-to-low propagation delays if present, and its capacitance if nonzero.

4

Succeeding lines list the transistor whose sources or drains connect to this node: the transistor type ("pulled down" is an n-channel transistor connected to gnd, "pulled up" is a depletion pullup or p-channel transistor connected to vdd), the values of the gate, source, and drain nodes, and the modeling resistances. Simple chains of transistors with the same implant type are collapsed by the _–s_ option into a single transistor with a "compound" gate; compound gates appear as a parenthesized list of nodes (e.g., the pulldown shown above). The three resistance values -- static, dynamic high, dynamic low -- are given in Kilo-ohms.

Finally, any pending events for a node are listed after the electrical information.

! _wnode..._
> **For each node in the argument list, print a list of transistors controlled by that node.**

tcap　Prints a list of all transistors with their source/drain shorted together or whose source/drain are connected to the power supplies. These transistors will have no effect on the simulation other than their gate capacitance load. Although transistors connected across the power supplies are real design errors, the simulator does not complain about them.

Any node can be made an input -- the simulator will not change an input node's value until it is released. Usually on specific nodes -- inputs to the circuit -- are manipulated using the commands below, but you can fool with a subcircuit by forcing values on internal nodes just as easily.

h _wnode..._
> **Force each node on the argument list to be a high (1) input. Overrides previous input commands if necessary.**

l _wnode..._
> **Like "h" except forces nodes to be a low (0) input.**

u _wnode..._
> **Like "h" except forces nodes to be a undefined (X) input.**

x _wnode..._
> **Removes nodes from whatever input list they happen to be on. The next simulation step will determine the correct node value from the surrounding circuit. This is the default state of most nodes. Note that this does not force nodes to have an "X" value -- it simply removes them from the input lists.**

inputs
> prints the high, low, and undefined input lists.

It is possible to define a sequence of values for a node, and then cycle the circuit as many times as necessary to input each value and simulate the network. A similar mechanism is used to define the sequence of values each clock node goes through during a single cycle.

Each value is a list of characters (with no intervening blanks) chosen from the following:
> 1, h, H　logic high (1)
> 0, l, L　logic low (0)
> u, U　undefined (X)
> x, X　remove node from input lists

Presumably the length of the character list is the same as the size of the node/vector to which it will be assigned. Blanks (spaces and tabs) are used to separate values in a sequence. The sequence is used one value at a time, left to right. If more values are needed than supplied by the sequence, IRSIM just restarts the sequence again.

V _[node [value...]]_
> **Define a vector of inputs for a node. After each cycle of an "R" command, the node is set to the next value specified in the sequence.**

With no arguments, clears all input sequences (does not affect clock sequences however). With one argument, "node", clears any input sequences for that node/vector.

clock *[node [value...]]*
> **Define a phase of the clock. Each cycle, each node specified by a clock command must run through its respective values. For example,**

clock phi1 1 0 0 0
clock phi2 0 0 1 0

> defines a simple 4-phase clock using nodes *phi1* and *phi2*. Alternatively one could have issued the following commands:
> vector clk phi1 phi2
> clock clk 10 00 01 00

> With no arguments, clears all clock sequences. With one argument, "node", clears any clock sequences for that node/vector.

After input values have been established, their effect can be propagated through the network with the following commands. The basic simulated time unit is 0.1ns; all event times are quantized into basic time units. A simulation step continues until *stepsize* ns. have elapsed, and any events scheduled for that interval are processed. It is possible to build circuits which oscillate -- if the period of oscillation is zero, the simulation command will not return. If this seems to be the case, you can hit **<ctrl-C>** to return to the command interpreter. Note that if you do this while input is being taken from a file, the simulator will bring you to the top level interpreter, aborting all pending input from any command files.

When using the linear model (see the "**model**" command) transition times are estimated using an RC time constant calculated from the surrounding circuit. When using the switch model, transitions are scheduled with unit delay. These calculations can be overridden for a node by setting its tplh and tphl parameters which will then be used to determine the time for a transition.

s *[n]* **Simulation step. Propogates new values for the inputs through the network, returns when** *n* **(default:** *stepsize*) ns. have passed. If *n* is specified, it will temporarily override the *stepsize* value. Unlike previous versions, this value is NOT remembered as the default value for the *stepsize* parameter. If the display mode is "automatic", the current display list is printed out on the completion of this command (see "display" command).

c *[n]* **Cycle** *n* times (default: 1) through the clock, as defined by the "**clock**" command. Each phase of the clock lasts *stepsize* ns. If the display mode is "*automatic*", the current display list is printed out on the completion of this command (see "**display**" command).

p Step the clock through one phase (or simulation step). For example, if the clock is defined as above
clock phi1 1 0 0 0
clock phi2 0 0 1 0

> then "**p**" will set phi1 to 1 and phi2 to 0, and then propagate the effects for one simulation step. The next time "**p**" is issued, phi1 and phi2 will both be set to 0, and the effects propagated, and so on. If the "**c**" command is issued after "**p**" has been used, the effect will be to step through the next 4 phases from where the "**p**" command left off.

R *[n]* **Run the simulator through** *n* cycles (see the "**c**" command). If *n* is not present make the run as long as the longest sequence. If display mode is automatic (see "**display**" command) the display is printed at the end of each cycle. Each "**R**" command starts over at the beginning of the sequence defined for each node.

back *time*
> **Move back to the specified time. This command restores circuit state as of** *time*, **effectively undoing any changes in between. Note that you can not move past any previously flushed out history (see flush command below) as the history mechanism is used to restore the network state. This command can be useful to undo a mistake in the input vectors or to re-simulate the circuit with a different debug level.**

path *wnode...*
>   **display critical path(s) for last transition of the specified node(s). The critical path transistions are reported using the following format:**

node **->** *value* **@** *time* (*delta*)

>   where *node* is the name of the node, *value* is the value to which the node transitioned, *time* is the time at which the transistion occurred, and *delta* is the delay through the node since the last transition. For example:

critical path for last transition of Hit_v1:
phi1-> 1 @ 2900.0ns , node was an input
PC_driver-> 0 @ 2900.4ns   (0.4ns)
PC_b_q1-> 1 @ 2904.0ns   (3.6ns)
tagDone_b_v1-> 0 @ 2912.8ns   (8.8ns)
tagDone1_v1-> 1 @ 2915.3ns   (2.5ns)
tagDone1_b_v1-> 0 @ 2916.0ns   (0.7ns)
tagDone_v1-> 1 @ 2918.4ns   (2.4ns)
tagCmp_b_v1-> 0 @ 2922.1ns   (3.7ns)
tagCmp_v1-> 1 @ 2923.0ns   (0.9ns)
Vbit_b_v1-> 0 @ 2923.2ns   (0.2ns)
Hit_v1-> 1 @ 2923.5ns   (0.3ns)

activity *from_time [to_time]*
>   **print histogram showing amount of circuit activity in the specified time inteval. Actually only shows number of nodes which had their most recent transition in the interval.**

changes *from_time [to_time]*
>   **print list of nodes which last changed value in the specified time interval.**

printp
>   print list of all pending events sorted in time. The node associated with each event and the scheduled time is printed.

printx
>   print a list of all nodes with undefined (X) values.

Using the trace command, it is possible to get more detail about what's happening to a particular node. Much of what is said below is described in much more detail in "Logic-level Simulation for VLSI Circuits" by Chris Terman, available from Kluwer Academic Press. When a node is traced, the simulator reports each change in the node's value:

>   [event #100] node out.1: 0 -> 1 @ 407.6ns

The event index is incremented for each event that is processed. The transition is reported as

old value **->** *new value* **@** *report time*

Note that since the time the event is processed may differ from the event's report time, the report time for successive events may not be strictly increasing.

Depending on the debug level (see the "**debug**" command) each calculation of a traced node's value is reported:
[event #99] node clk: 0 -> 1 @ 400.2ns
final_value(Load) V=[0.00, 0.04] => 0
..compute_tau(Load)
{Rmin=2.2K Rdom=2.2K Rmax=2.2K} {Ca=0.06 Cd=0.17}
tauA=0.1 tauD=0.4 ns
[event #99: clk->1] transition for Load: 1 -> 0 (tau=0.5ns, delay=0.6ns)

In this example, a calculation for node *Load* is reported. The calculation was caused by event 99 in which node clk went to 1. When using the linear model (as in this example) the report shows

current value -> final value

The second line displays information regarding the final value (or dc) analysis for node "Load"; the minimun and maximum voltages as well as the final logical value (0 in this case).

The next three lines display timing analysis information used to estimate the delays. The meaning of the variables displayed can be found Chu's thesis: "Improved Models for Switch-Level Simulation".

When the *final value* is reported as "D", the node is not connected to an input and may be scheduled to decay from its current value to X at some later time (see the "**decay**" *command).*

"tau" is the calculated transition time constant, "delta" is when any consequences of the event will be computed; the difference in the two times is how IRSIM accounts for the shape of the transition waveform on subsequent stages (see reference given above for more details). The middle lines of the report indicate the Thevenin and capacitance parameters of the surrounding networks, i.e., the parameters on which the transition calculations are based.

debug *[ev dc tau taup tw spk][off][all]*
> **Set debugging level. Useful for debugging simulator and/or circuit at various levels of the computation. The meaning of the various debug levels is as follows:**

> **ev** display event enqueueing and dequeueing.

> **dc** display dc calculation information.

> **tau** display time constant (timing) calculation.

> **taup** display second time constant (timing) calculation.

> **tw** display network parameters for each stage of the tree walk, this applies to **dc, tau**, and **taup**. This level of debugging detail is usually needed only when debugging the simulator.

> **spk** displays spike analysis information.

> **all** This is a shorthand for specifying all of the above.

> **off** This turns off all debugging information.

> If a debug switch is on then during a simulation step, each time a watched node is encounted in some event, that fact is indicated to the user along with some event info. If a node keeps appearing in this prinout, chances are that its value is oscillating. Vice versa, if your circuit never settles (ie., it oscillates) , you can use the "**debug**" and "**t**" commands to find the node(s) that are causing the problem.
> Without any arguments, the debug command prints the current debug level.

t *[-]wnode...*
> **set trace flag for node. Enables the various printouts described above. Prefacing the node name with '-' clear its trace flag. If "wnode" is the name of a vector, whenever any node of that vector changes value, the current time and the values of all traced vectors is printed. This feature is useful for watching the relative arrival times of values at nodes in an output vector.**

System interface commands:

> *filename*
> **Write current state of each node into specified file. Useful for making a breakpoint in your simulation run. Only stores values so isn't really useful to "dump" a run for later use, i.e., the current input lists, pending events, etc. are NOT saved in the state file.**

< *filename*
> **Read from specified file, reinitializing the value of each node as directed. Note that network**

must already exist and be identical to the network used to create the dump file with the ">" command. These state saving commands are really provided so that complicated initializing sequences need only be simulated once.

<< *filename*
> **Same as "<" command, except that this command will restore the** *input* **status of the nodes as well.** It does not, however, restore pending events.

dumph *[filename]*
> **Write the history of the simulation to the specified file, that is; all transistions since time = 0. The resulting file is a machine-independent binary file, and contains all the required information to continue simulation at the time the dump takes place. If the filename isn't specified, it will be constructed by taking the name of the sim_file (from the command line) and appending ".hist" to it.**

readh *filename*
> **Read the specified history-dump file into the current network. This command will restore the state of the circuit to that of the dump file, overwriting the current state.**

flush *[time]*
> **If memory consumption due to history maintanance becomes prohibitive, this command can be used to free the memory consumed by the history up to the time specified. With no arguments, all history up to the current point in the simulation is freed. Flushing out the history may invalidate an incremental simulation and the portions flushed will no longer appear in the analyzer window.**

setpath *[path...]*
> **Set the search-path for command files.** *Path* **should be a sequence of directories to be searched for ".cmd" files, "." meaning the current directory. For eaxmple:**

setpath . /usr/me/rsim/cmds /cad/lib/cmds

> With no arguments, it will print the current search-path. Initially this is just ".".

print *text...*
> **Simply prints the text on the user's console. Useful for keeping user posted of progress through a long command file.**

logfile *[filename]*
> **Create a logfile with the specified name, closing current log file if any; if no argument, just close current logfile. All output which appears on user's console will also be placed in the logfile. Output to the logfile is cleverly formatted so that logfiles themselves can serve as command files.**

setlog *[filename | off]*
> **Record all net changes, as well as resulting error messages, to the specified file (see "update" command). Net changes are always appended to the log-file, preceding each sequence of changes by the current date. If the argument is** *off* **then net-changes will not be logged.** With no arguments, the name of the current log-file is printed.
> The default is to always record net changes; if no filename is specified (using the "**setlog**" command) the default filename *irsim_changes.log* will be used. The log-files are formatted so that log-files may themselves be used as net-change files.

wnet *[filename]*
> **Write the current network to the specified file. If the filename isn't specified, it will be constructed by taking the name of the sim_file (from the command line) and appending ".inet" to it. The resulting file can be used in a future simulation run, as if it were a sim file. The file produced is a machine independent binary file, which is typically about 1/3 the size of the sim file and about 8 times faster to load.**

time *[command]*
> **With no argument, a summary of time used by the simulator is printed. If arguments are given**

the specified command is timed and a time summary is printed when the command completes. The format of the time summary is *U***u** *S***s** *E P***%** *M***, where:**

U => User time in seconds
S => System time in seconds
E => Elapsed time, minutes:seconds
P => Percentage of CPU time (((U + S)/E) * 100)
M => Median text, data, and stack size use

q Terminate current input stream. If this is typed at top level, the simulator will exit back to the system; otherwise, input reverts to the previous input stream.

exit *[n]*
        Exit to system, *n* is the reported status (default: 0).

Simulator parameters are set with the following commands. With no arguments, each of the commands simply prints the current value of the parameter.

decay *[n]*
        Set decay parameter to *n* ns. (default: 0). If non-zero, it tells the number of ns. it takes for charge on a node to decay to X. A value of 0 implies no decay at all. You cannot specify this parameters separately for each node, but this turns out not to be a problem. See "**report**" command.

display *[-][cmdfile][automatic]*
        set/reset the display modes, which are

        cmdfile commands executed from command files are displayed to user before executing. The default is *cmdfile = OFF.*

        automatic print out current display list (see "**d**" command) after completion of "**s**" or "**c**" command. The default is *automatic = ON.*

        Prefacing the previous commands with a "-" turns off that display option.

model *[name]* **Set simulation model to one of the following:**

        switch
                Model transistors as voltage controlled switches. This model uses interval logic levels, without accounting for transistor resistances, so circuits with fighting transistors may not be accuratelly modelled. Delays may not reflect the *true* speed of the circuit as well.

        linear
                Model transistors as a resistor in series with a voltage controlled switch. This model uses a single-time-constant computed from the resulting RC network and uses a two-time-constant model to analyze charge sharing and spikes.

        The default is the **linear** model. You can change the simulation model at any time -- even with events pending -- as only new calculations are affected. Without arguments, this command prints the current model name.

report *[level]*
        When level is nonzero, report all nodes which are set to X because of charge decay, regardless on whether they are being traced. Setting level to zero disables reporting, but not the decay itself (see "decay" command).

stepsize *[n]*
        Specify duration of simulation step or clock phase. *n is specified* in ns. (nanoseconds). Floating point numbers with up to 1 digit past the decimal point are allowed. Further decimals are trucated (i.e. 10.299 == 10.2).

unitdelay *[n]*
> **When nonzero, force all transitions to take** *n* ns. Setting the parameter to zero disables this feature. The resolution is the same as for the "**stepsize**" command.

stats Print event statitistics, as follows:
changes = 26077
punts (cns) = 208 (34)
punts = 0.79%, cons_punted = 16.35%
nevents = 28012; evaluations = 27972

> Where *changes* is the total number of transistions recorded, *punts* is the number of punted events, *(cns)* is the number of consecutive punted events (a punted event that punted another event). The penultimate line shows the percentage of punted events with respect to the total number of events, and the percentage of consecutive punted events with respect to the number of punted events. The last line shows the total number of events (nevents) and the number of net evaluations.

Incremental simulation commands:

Irsim supports incremental changes to the network and resimulation of the resulting network. This is done incrementally so that only the nodes affected by the changes, either directly or indirectly, are re-evaluated.

update *filename*
> **Read net-change tokens from the specified file. The following net-change commands are available:**

> **a**dd type gate source drain length width [area]
> **d**elete type gate source drain length width [area]
> **m**ove type gate source drain length width [area] g s d
> **c**ap node value
> **N** node metal-area poly-area diff-area diff-perim
> **M** node M2A M2P MA MP PA PP DA DP PDA PDP
> **t**hresh node low high
> **D**elay node tplh tphl

> For a detailed dscription of this file see netchange(5). Note that this is an experimental interface and is likely to change in the future.

> Note that this command doesn't resimulate the circuit so that it may leave the network in an inconsistent state. Usually this command will be followed by an **isim** command (see below), if that is not the case then it's up to the user to initilize the state of the circuit. This command exists only for historical reasons and will probably disappear in the future. It's use is discouraged.

isim *[filename]*
> **Read net-change tokens from the specified file (see netchange(5)) and incrementally resimulate the circuit up to the current simulation time (not supported yet).**

ires *n* **The incremental algorithm keeps track of nodes deviating from their past behavior as recorded in the network history. During resimulation, a node is considered to deviate from its history if it's new state is found to be different within** *n* ns of its previous state. This command allows for changing the incremental resolution. With no arguments, it will print the current resolution. The default resolution is 0 ns.

SEE ALSO

presim(1) (now obsolete) rsim(1) irsim-analyzer(3) sim(5) netchange(5)