# Hints, Tips and Solutions

Colin Shaw, Applications and Support Engineer

## Remote .ALTER processing

A new parallelization method has been implemented into ***SmartSpice***. Now .ALTERs can be destributed not only over several CPUs (by using -P option), but over a network of computers as well.

Remote .ALTER processing works the following way:

When it is invoked (by using -remote command line option), ***SmartSpice*** will read the input deck and check for .ALTER statements in it. If there are no .ALTER statements in the input deck, ***SmartSpice*** will just continue simulating the given netlist in batch mode. If .ALTERs are found, ***SmartSpice*** will extract parts of the netlist which form entire circuits and write out each circuit as separate files (.ALTER files). Resulting files containing one altered circuit each are named by adding the suffix -n to the composite basical netlist file name and have no extension. Number of produced files equals to the amount of .ALTER statements in the composite netlist plus one (deck without .ALTERs).

***SmartSpice*** will then collect hosts from ralter_hosts list variable set in SmartSpice.ini file. This data consists of host names to distribute .ALTERs onto and number of CPUs to use on each host. On the next step ***SmartSpice*** will try to launch one simulation on each host (filling all hosts if number of .ALTERs is greater than amount of hosts specified). If the simulator on certain host is launched successfully, parent ***SmartSpice*** will check for amount of available CPUs source child with according .ALTER file and start simulation. Then, if there are not processed .ALTERs remaining, the parent will launch additional simulators on this given host (up to number CPUs specified by user, but never exceeding actual number of CPUs present).

If host fails for some reason, the .ALTER file it was processing will be released, and parent ***SmartSpice*** will retry to launch the simulator on the failed host again. If the host fails for ralter_numretries times, the parent ***SmartSpice*** will stop retrying to launch the remote simulator on it. If all hosts fail, but unprocessed .ALTERs still remain, ***SmartSpice*** will exit with error an message.

When remote ***SmartSpice*** finishes simulation, it signals the parent that it became idle and is immediately sourced with another unprocessed .ALTER file. If there are no unprocessed .ALTERs left, remote ***SmartSpice*** will terminate.

Output files (raw, out, err, etc.) are created for each separate circuit file.

When all separate files are processed, parent ***SmartSpice*** removes these files and terminates. Parent does not do any simulation - it just monitors remote simulators and acts as the server manager.

***Usage***:

```
smartspice <input-file> -remote
```

Variables that can be set in SmartSpice.ini file:

**ralter_hosts = ( *hostname<=numCPUs>***
**  *<<hostname<=numCPUs>> ...* > )**

This list variable specifies the list of hosts to destribute .ALTERs onto. Syntax is important. There has to be space after opening parenthesis, and before closing parenthesis. When number of CPUs to use is specified, there should not be spaces around '='. If the number of CPUs is not specified, all available CPUs on the remote host will be used.

`ralter_hosts` variable is mandatory because if it is not set, ***SmartSpice*** will not be able to destribute .ALTERs over the network, and will continue simulation localy in batch mode.

***Example***:

```
ralter_hosts = ( hostone hosttwo=4 host-
three=2 )
```

**ralter_outpath = *"val"***

Path to store output files. Default: path were original composite netlist is located.

**ralter_timeout = *val***

Timeout in seconds after which remote host is considered failed. Default: 30 seconds.

**ralter_numretries = *val***

Number of retries to launch simulator on remote host. Default: 5

**Note:** TMP environmental variable must be set to location for temporary files.

All computers in network must have the same operatng system. Remote simulators are invoked by using invocation line used for parent ***SmartSpice*** , thus if parent was launched on a Linux platform, it will be unable to launch remote simulator on a Unix platform.

Remote .ALTER processing is not currently supported for Windows platforms.

`-remote` option also forces ***SmartSpice*** to run in batch mode.

If specified in command line, the startup file must include a full path. Otherwise remote ***SmartSpice*** will fail to load it.

## .OVERSHOOT Statement Improved

.OVERSHOOT statement has been improved. In previous versions of ***SmartSpice*** this statement allowed the user to check all nodes on every timepoint during the simulation, and report nodes with voltages exceeding limits specified by ***vmin*** and ***vmax***. Report contained node names along with time points whenever a violation occurs.

Now .OVERSHOOT functionality has been enhanced making it more powerfull and more convinient to use. It is now possible to specify not only voltage thresholds, but also minimum spike duration to detect, and nodes list that have to be excluded from checking.

Output format has also changed. Previous report contained node names and timepoints when a spike occured, and was sorted by timepoints. Now report is sorted by nodes. If spike occured on any certain node, first there's a line stating that v(*x*) > *vmax* or v(*x*) < *vmin* and then is followed by listing of spikes detected. This list contains spike start time, spike end time, spike duration and peak voltage (maximum voltage in case of >*vmax* spike or minimum voltage in <*vmin* spike case).

It is now possible to specify more than one statement for single analysis. But like in previous versions, the only analysis supported is TRAN.

***Syntax***

```
.OVERSHOOT filename="val" <vmin=val>
<vmax=val> <duration=val>

+ <excludenodes=nodename <nodename
<...<nodename>>>
```

**filename:** The name of the file where overshoot information is saved. This file contains comment lines that identify the circuit name and type, the type and date of the simulation, specified params and report on detected spikes.

**vmin=val, vmax=val:** These value specify voltage limits. At least one of them has to be present.

**duration=val:** Specifies minimum duration of spike to detect. If omitted, ***SmartSpice*** will detect spikes of any duration.

**excludenodes:** List of nodes to exclude from checking. Wildcards can be used. If not specified, ***SmartSpice*** will check all nodes.

***Example***:

```
.OVERSHOOT filename="o.ost" vmin=-0.2 vmax=0.9
duration=1p + excludenodes="0 va* x3.*"
```

## New RNOISE Parameter for Resistor

A new optional device parameter, specified resistance value for noise analysis (which can be different from DC/TRAN or AC analysis resistance values) was added for linear resistor specification:

**RNOISE|NOISE=val:** Resistance for noise simulation. Default is AC or R value.

***Example***:

```
R 3 7 R=0.001 AC=1e10 RNOISE=0.001
```

In this example resistor R5, connected between nodes 3 and 7, for DC and Transient Analysis has a resistance value R=0.001 Ohm, Noise Analysis resistance is Rnoise=0.001Ohm, and AC analysis resistance is AC=1e10 Ohm.

## Multiple nested .DC, .AC, .TRAN sweep

Now, ***SmartSpice*** allows the user to use the multiple nested parametric sweep for .DC, .AC, .TRAN analyses.

***Syntax***

.TRAN ... **SWEEP** *VAL1* start1 stop1 step1 **SWEEP** *VAL2* start2 stop2 step2 ...

.DC ... **SWEEP** *VAL1* start1 stop1 step1 **SWEEP** *VAL2* LIST nplist val1 val2 ...

.AC ... **SWEEP** *VAL1* DEC np start stop **SWEEP** *VAL2* MODIF=VAL1 <PRTBL>...

The multiple nested parametric sweep works the following way:

At first ***SmartSpice*** saves the current amounts and makes the start initialization of the all sweep variables.

After that, ***SmartSpice*** performs the specified analysis, changing settings for the first sweep variable. When the loop of the first sweep variable is finished ***SmartSpice*** makes the start initialization for the first sweep variable, and next step for the second sweep variable. ***SmartSpice*** repeat the above mentioned actions until the loop of the second sweep variable will not be finished.

The multiple sweep uses nested loops like:

```
for (, ,)
    for (, ,)
        for(, ,)
```

Finally, ***SmartSpice*** restores the amounts of sweep variables.

The multiple nested parametric sweep is an extension for the standard sweep ideology. Sweep blocks have the same syntax and must be started with keyword "SWEEP".

**Call for Questions**

If you have hints, tips, solutions or questions to contribute, please contact our Applications and Support Department
Phone: (408) 567-1000 Fax: (408) 496-6080
e-mail: support@silvaco.com

**Hints, Tips and Solutions Archive**

Check our our Web Page to see more details of this example plus an archive of previous Hints, Tips, and Solutions
www.silvaco.com