SmartSpice Training Program

Analog/Mixed-Signal Simulation

Transitioning to SmartSpice
Part 1: Getting Started
At the end of this SmartSpice training session you should understand:

- Various operating modes and their advantages
- How to start & run the tools in these various modes
- How to optimize your designs for and achieve the maximum performance of SmartSpice in parallel/multi-threading/multi-core mode
- How to run SmartSpice simulations when originating from (HSPICE, PSPICE) decks
- How to analyse output results using SmartView
- What to look out for and how to address convergence issues
- How and when to utilize the SmartSpice Optimizer to improve design productivity by eliminating manual design iteration
- How and when to utilize the Verilog-A modeling language as part of your design flow to improve design productivity and expand simulation coverage
SmartSpice Training Outline

- Part 1 SmartSpice Getting Started (9:00-9:45)
  - Silvaco Design Flows
  - Silvaco PDK-Based AMS/RF Design Flow
  - What is SmartSpice?
  - Introduction to SmartSpice
  - SmartSpice supported platforms, design environments & operating modes
  - Basics of SmartSpice
  - Output
  - Summary of SmartSpice Analysis Techniques
  - Commonly Used Options
  - Other Commonly Used Functions
  - Special Functions
  - Special Functions (cont’d)
  - Transient Noise Analysis
  - Noise Analysis
  - Sensitivity Analysis
  - Monte Carlo Analysis
  - SmartSpice Invocation Syntax
  - SmartSpice Invocation Syntax (output)
  - SmartSpice Invocation Syntax (silent batchmode)
  - SmartSpice Invocation Syntax (rawfile)
  - SmartSpice Invocation Syntax (windows)
  - SmartSpice Initialization Scheme

- MODULE BREAK / Q&A (9:45-10:00)
  - HSPICE Compatibility Mode
  - Measurement Results (HSPICE mode)
  - .ALTER and *.mt# Files
  - PSPICE to SmartSpice conversion (part I)
  - PSPICE to SmartSpice conversion (part II)
  - PSPICE to SmartSpice conversion (part III)
  - PSPICE to SmartSpice conversion (part IV)
  - Use ‘Spiceserver’ to Cut on License Check-out Time (for UNIX and LINUX only) (part I)
  - Use ‘Spiceserver’ to Cut on License Check-out Time (for UNIX and LINUX only) (part II)
  - Parallel SmartSpice (Parallel Computing)
  - Parallelization Models in SmartSpice
  - Task Parallelization in SmartSpice
  - Data Parallelization in SmartSpice
  - Data Parallelization in SmartSpice (cont’d)
  - Parallelization User Requirements
  - Parallelization Processes in SmartSpice
  - SmartSpice Special Functions
  - SmartSpice Encryption Engine
  - SmartSpice Encryption Engine (cont’d)

- MODULE BREAK / Q&A (10:45-11:00)
SmartSpice Training Outline (cont)

- SmartSpice Training Outline (cont)
  - Part 2 SmartVIEW (11:00-11:10)
  - What is SmartVIEW?
  - Signal Selection and Graphing Options
  - Interactive & Composite Measurements
  - Advanced Concepts & Post-Processing Options

- Part 3 SmartSpice Convergence Tips (11:10-11:20)
  - Basics to Convergence
  - Auto Convergence Algorithm Process – 5 levels
  - DC / OP Analysis Convergence Aids
  - Transient Analysis Options
  - Aids in Overcoming Non-Convergence
  - Notes and Restrictions for Using Control Options

- Part 4 SmartSpice Optimizer (11:20-11:45)
  - What is SmartSpice Optimizer?
  - Optimizer Using .MODIF
  - SmartSpice Optimizer Syntax
  - SmartSpice Optimizer Syntax (cont’d)
  - SmartSpice Optimizer Input Deck
  - Timing Optimization Using Bisection
  - SmartSpice Bisection Optimization Syntax
  - SmartSpice Bisection Optimization Syntax (cont’d)
  - HSPICE Compatibility Syntax
  - HSPICE Compatibility Syntax (cont’d)

- MODULE BREAK / Q&A (11:45-12:00)

- Part 5 SmartSpice Verilog-A (12:00-12:45)
  - What is SmartSpice Verilog-A?
  - Introduction to Verilog-A in SmartSpice
  - Representing a System in Verilog-A
  - The SmartSpice Verilog-A Simulation Flow
  - Verilog-A System Types
  - Verilog-A Inverter Model
  - Verilog-A Inverter SPICE Test Bench
  - Verilog-A NAND Model
  - Verilog-A DFF Model
  - Verilog-A LPF
  - Verilog-A HPF
  - Verilog-A Active Analog Circuits
  - Verilog-A OPAMP
  - Verilog-A Sample & Hold
  - Verilog-A DAC (part I)
  - Verilog-A DAC (part II)
  - Verilog-A ADC (part I)
  - Verilog-A ADC (part II)

- LUNCH BREAK (12:45-1:30)

- Part 6 SmartSpice Training Lab
SmartSpice Training Program

Analog/Mixed-Signal Simulation

Getting Started
What is SmartSpice?

- SmartSpice is a general purpose, nonlinear, transistor level, circuit simulator for analyzing the discrete and distributed DC, transient & AC electrical behavior of complex SPICE format netlists and their physical parasitics
- SmartSpice supports a wide variety of industry standard active and passive analytical device models plus Verilog-A (an analog behavioral description language) for user-defined compact model generation of devices and circuit blocks for top-down/bottom-up system-level definition and verification methods
- SmartSpice’s optimizer provides a powerful means to aid the designer in finding the best circuit device parameters (W & L, etc) to achieve the desired performance goal thereby enhancing productivity
What is SmartSpice? (con’t)

- SmartSpice’s main out-of-the-box performance advantages over other SPICE simulators are:
  - More robust circuit convergence - due to advanced model parameter error detection, trapping, reporting, and out-of-range compensation algorithms, plus a library of 3 direct and 2 iterative matrix-solvers to handle even the most complex of circuit topologies
  - Greater accuracy over a wider operating range – due to improved device modeling equations proven through TCAD physics-based process and device numerical modeling tests and probed wafer data
  - Faster simulation speed – due to advanced 64-bit, multi-threaded, dynamic load-balanced parallel processing algorithms
  - Ability to simulate a wider selection of devices without macromodeling - due to an advanced proprietary device model library
  - Greater capacity for complex circuits – due to 64-bit processing technology and efficient memory management techniques
Introduction to SmartSpice

- Complete Analog Circuit Simulator
- Multi-threaded Simulation Flow
- HSPICE™ Compatible: Input & output files
  (Batch-mode: -hspice  GUI:set simulator=hspice )
- Input Compatible - Eldo™, Pspice™
- The SmartSpice Verilog-A Interface
- SmartSpice Special Functions
  - Superior convergence & Accuracy
  - Accurate modeling, including many foundry models
  - Optimizer for circuit optimization to models and cells, with incremental or
    simultaneous multi-parameter optimization in AC, DC, and transient analyses
    (.MODIF, .Measure .ST)
  - Monte Carlo and worst-case analyses design support
Linux Red Hat Enterprise 4, 5 and 6 64bit and windows XP, Win7 Professional 32bit or 64 bit PCs

The size of the circuit that can be simulated is limited only by the computer hardware and operating system. (active devices of 400K on 32-bit windows and 8M+ on 64-bit Linux w/64GB RAM)

SmartSpice can be run from under the Silvaco Gateway schematic design environment or standalone

SmartSpice also supports running under the Cadence Analog Design Environment (ADE) (formerly Analog Artist) behaving as a Spectre replacement

SmartSpice supports both GUI driven & command line batch processing

SmartSpice supports HSPICE and PSPICE netlist operating modes through parser control switches on invocation (startup)
**Input deck:** Netlist: include file(model file library file include files(ex. Verilog –A & Digital vector files...), suffix: .sp .in .inc ,...

**output file:** simulation result or data (raw file format), suffix: .raw or HSPICE output files : .tr0 .lis .pa0 .st0,...,measurement output file or other specific files (save-bias file , bias-check file...).And while (parse the circuit) source: warning & error message (from input deck or include file ...)

**batch-mode & GUI**
Command line in batch-mode:

**Ex:**
```
```

(-r, if post option isn’t specified)

**Viewer**

- SmartView or other compatible viewer for all analyses results files
SmartSpice generates a .raw file to store the simulation result, which can be opened using SmartView.

**Output Commands**

- `.PRINT`, `.PLOT`, `.IPLOT`, `.GRAPH`, `.PROBE`, and `.MEASURE`

  Each statement specifies:
  - output variable
  - simulation result to be displayed
    - `.IPLOT` for interactive plotting during simulation
    - `.GRAPH` sends hardcopy to printer automatically
    - `.PROBE` can limit raw file size (it requires `.options probe`)
    - `.MEASURE` has several special forms
    - `.Save` command can save vectors into specific raw file
Summary of SmartSpice Analysis Techniques

- DC Analysis
- AC Small-Signal Analysis
- Transient Analysis
- Transfer Function
- Network Analysis
- Sensitivity Analysis
- Noise Analysis
- Distortion Analysis
- Fourier Analysis
- Forward and Reverse FFT
- Monte Carlo/Worst-Case Analysis
- Pole-Zero Analysis
### Commonly Used Options

#### Simulation Control Options
- **Acout=1**
- **acct=2**
- **expert=777**
- **nomode**
- **Temp**
- **post probe**
- **Stopcont**
- **rawpts=300~500**
- **LIST BRIEF**
- **dccap**

#### Output Format Options
- **solver=speeds**
- **FAST**
- **bypass=1**
- **autostop**

#### Speed Options
- **LIST**
- **BRIEF**

#### Accuracy Options & Algorithm Options
- **Abstol reltol**
- **accurate**
- **accurate**
- **Logic=2**

#### Convergence Options
- **Changessolver**
- **Expert=777**
- **GMIN= DCGMIN=**
- **GNODE CNODE=**
- **DCPATH=**

#### Model Options
- **Scale**
- **nowarn**
- **TNOM=**

#### Special Feature Options
- **biasstoptime=**
- **D IBIS=**

*For complete options see “2.14”*
Other Commonly Used Functions

- .BIASCHK
- .CONTROL(Control block)
- .IPLT
- .OVERSHOOT
- .VEC
- .TF
- .TEMP
- .WCASE
- .PARAM
- .INC
- .LET
- .PROBE
- .SAVE HIER1(V) HIER2(I)
From simulation result, SmartSpice can allow users to see the effects of:
- Element parameter variation
- Model parameter variation

By using:
- **.ST** – Sweep Variables for all analyses
  - Ex: `.ST VIN -2.0 5.0 0.25`
  - Ex: `.ST DEC QNL(BF) 10 100 10`
  - Ex: `.ST LIST TEMP 0 20 27 50 -20`

- **.ALTER** – Alter simulation run on changing input deck by sequence
  - Switch to different process corner libraries and repeat simulation
  - Ex: `.tran 10ps 10ns SWEEP MONTE = 500`
  - .measure tran OIN1 AVG i(VN1) FROM=0ns TO=10n
  - .alter
  - .option seed=29001
Special Functions (con’t)

- **.WCASE**  – Worst Case Corner analysis
  Component value limits are known, measured, and implemented as skew parameters
  
  **Ex:**  `.WCASE TRAN V(1) FALL_CROSS (0.001) LIST OUTPUT ALL`
  **Ex:**  `.WCASE DC V(1,2) MAX LIST RANGE (*,5) VARY ALL`

- **.MC**  – Standard Statistical Analysis
  Statistical standard deviations of component values are known.
  Automated over a random range of values

- **.MEASURE**

- **.MODIF**
- Transient Noise Analysis -

- Noise (=val) in .TRAN statement
- Causes SMARTSPICE to perform Transient Noise Analysis and produce the noise variances of the circuit variables as a function of time, including thermal, shot, and flicker noise from devices in the circuit

Ex: .TRAN 1ps 100ns NOISE Freq=20 Freqh=2kHz
Ex: .TRAN 1ps 50ns NOISE=5 Freq=20 Freqh=20megHz
Ex: .TRAN 1ns 20us NOISE Freq=2kHz NOISE_REL_TOL=0.02
**Noise Statement**

- This statement performs a noise analysis of the circuit. The analysis calculates noise spectral density at the output port due to noise sources of each device in the circuit, and the equivalent input noise.

  Ex: .NOISE v(2) vin
  Ex: .NOISE v(op,on) vip DEC 10 1 10G
Sensitivity Analysis

.SNS Statement

- This statement calculates the sensitivity of any basic output variable to the most input parameters at any DC value or time-points

Ex: .SNS DC V(VN) V(VP) to ARG=5.5 ARG2=6.0 MP1(W) M1(W)
Ex: .SNS TRAN V(5) to M7(W) ARG=6NS ARG2=26NS
How to use Monte Carlo Analysis

- Analysis must be in .AC, .DC, .Tran or .MC
- Value of MONTE is the number of randomizations

Ex: .DC v1 START=0 STOP=0.8499 STEP=0.01 sweep monte=val
Ex: .AC LIN 101 0.01MEG 10G sweep monte=300
Ex: .Tran 10ps 10ns SWEEP MONTE = 500
Ex: .MC 10 TRAN V(1) FALL_CROSS (0.001) LIST OUTPUT ALL
Ex: .MC 30 DC V(1,2) MAX LIST RANGE (*,5) SEED=135
Ex: .MC 50 AC VM(10) YMAX OUTPUT RUNS 5, 10, 20, 30, 40
% smartspice [-n] [-u] [-v] [-P n_cpus [-nodist]][-i|-c|-pp|-b [-sb] infile [-o outfile][-r rawfile][-hspice][-pspice][-startupfile]

- **b**: (batchmode) SmartSpice parses the input file infile and performs the required analyses.
- **c**: (command mode) On the UNIX platforms, a CSH-like command line user interface is used to run SmartSpice interactively. For a summary of the available SmartSpice commands see Chapter 5: “Commands.”
- For displaying text information in batchmode, by default, SmartSpice uses the standard output/error streams. By using UNIX shell commands, it’s possible to redirect these output streams to file. For an example of C shell, see the following the command line:
  ```
  smartspice -b exl.in > & result.out
  ```
- **i**: (window mode) This option forces SmartSpice to run with a graphical user interface, and is Qt-based on UNIX platforms.
- **n**: (don’t read startup file) On UNIX, the ~/SmartSpice.ini is read when SmartSpice starts up before reading any input deck, and its contents are treated like the contents of a .CONTROL block. When this flag is specified, the file is not read.
- **nodist**: (do not distribute for .ALTER) See the description of the .ALTER statement for details of this flag.
- **-o:** *(output file)* During batchmode simulation, messages are printed on the screen by default. This option redirects messages to the file .out

- **-e:** *(error file)* On Windows, during batchmode simulation, error messages are printed to a file

- **Note:** The following options affect the data output by SmartSpice: BRIEF, NOMOD, FORMAT, LARGE, ACCT, and NUMDGT

- **-P n_cpus:** *(run in parallel on several CPUs)* On multi-threaded platforms, for certain models, SmartSpice can distribute over the requested number of processors *(n_cpu)*

- **-pp:** *(post-processor mode)* This option invokes a post-processor only version of SmartSpice. It is only used when a full license for SmartSpice exists, however a license will not be used. The post-processor version cannot perform any simulation

- **-r:** *(rawfile)* After batchmode simulation, output variables saved during simulation are stored in a rawfile. The option `-r` is used to specify the name of the file where the results are stored. SmartSpice loads rawfiles from prior simulations, compares results, generates plots, and so forth
**SmartSpice Invocation Syntax (silent batchmode)**

- **Note**: The following option is used to work with a remote computer with installed Unix/Linux operating systems without X server

- **-sb**: *(silent batchmode)* SmartSpice parses the input file *infile* and performs the required analysis. The differences between batchmode and silent batchmode are as follows:
  - Silent batchmode works without GUI features (windows, message boxes, plots and so forth)
  - Silent batchmode redirects all the output information to a file that is specified using the command line option [-o *outfile*], or automatically to the file `<input_deck_name> SBout` if [-o *outfile*] commands are absent
  - Silent batchmode doesn't draw in SmartVIEW.

**Example**

smartspice -sb circuit.in -o alloutput.out

SmartSpice parses the input file circuit.in and performs the required analysis. In the first example, SmartSpice will redirect all output information in an alloutput.out file.

The second example will redirect all output information in `<input_deck_name> SBout` file in the input deck’s directory automatically.

The default parallel scheme in the batchmode (-b) and silent batchmode (-sb) has changed from create/join (shell variable *mt_scheme=0*) to pool of threads (shell variable *mt_scheme=1*)

The parallel scheme pool of threads shows the best performance in combination with -b and -sb modes. GUI and command mode (-c) use create/join algorithm by default.
**SmartSpice Invocation Syntax (rawfile)**

- **Note**: The following options affect rawfile generation: `POST`, `PROBE`, `RAWPTS`, `NUMDGT` and `SAVEMEASURES`.

- **-u**: *(show usage)* SmartSpice prints a list of the allowed command line flags when this option is used.

- **-v**: *(print version number)* This option causes SmartSpice to print the version number as well as the hostname of the machine on which SmartSpice is running.

- **-hspice**: *(HSPICE compatibility mode)* This option turns on the HSPICE compatibility mode.

- **-pspice**: *(PSPICE compatibility mode)* This option turns on the PSPICE compatibility mode, and causes the same SmartSpice reaction as the command set simulator=pspice in the startup file.

- **-startupfile**: *(specify alternative startup file)* This option allows you to specify the startup file instead of the default `SmartSpice.ini` (UNIX) or `mspice.set` (WIN32).

- **Example** `smartspice -startupfile MyInitialization.ini`
If SmartSpice is invoked with no options, it starts in the Windows mode

**Note**: The following option only applies to the Windows operating system

- **foreground**: This option applies only to the batchmode operation (-b), and is used as the first switch after SmartSpice to change the operation from background to foreground (i.e. the prompt only returns after the simulation is complete). This can be used to start jobs in a sequence run for maximum tool license use

**Example** `smartspice -foreground -b circuit.in -r Rawfile.raw`
The initialization file (or startup file) is read when SmartSpice starts, before reading any input deck, and its contents are treated like the contents of a .CONTROL block.

The initialization scheme defines the name of the initialization file and its location.

In the old initialization scheme, the name of the initialization file was different on Unix and Windows, and SmartSpice could read this file (or files) from many locations.

Now SmartSpice supports a new initialization scheme.
The default name of the initialization file is SmartSpice.ini, and is the same on Unix and Windows. The algorithm of loading the initialization file is:

- When SmartSpice starts with the option -startupfile filename on the command line, SmartSpice will source filename. If sourcing is successful, the initialization phase is finished, otherwise the step 2 will be done.
- SmartSpice sources default SmartSpice.ini from the program (executable) directory first. If sourcing is successful, the initialization phase is finished, otherwise SmartSpice sources the default SmartSpice.ini from the home directory.
- Note: For Unix the home directory is usually set to /home/username, where username is the login name.
  - For Windows, the home directory is usually set to USERPROFILE, where USERPROFILE is the Windows environment variable for the current user. For example, user kon
    USERPROFILE = “WinPartition(C:):\Documents and Settings\kon”

For compatibility with the previous version of SmartSpice, the old initialization scheme can be activated with command line option -oldini. This applies to SmartSpice version 2.15.0 and above.

Note: smspice.set is still available for Window’s users to apply to older versions.
The command line option -hspice can be used to turn on the HSPICE compatibility mode.

In general, SmartSpice maintains a high degree of compatibility with other SPICE simulators.

However, there are occasions when a standard SmartSpice behavior has been in use for some time, and conflicts with other SPICEs.

This flag is useful in distinguishing between the two behaviors.

If the input deck is specified in the -hspice batchmode as a link, rawfiles will then be created in the link directory.

SmartSpice supports redirection of rawfiles, which were created in HSPICE compatibility mode, using the option -r.

You can use this feature to collect all rawfiles in one location, which has more free space and other advantages.

On previous SmartSpice versions, rawfiles were always created in the same directory, and with the same file name as the input deck.
Example:

```
smartspice -hspice -b circuit.in -r Rawfiles_Location/Results.raw
```
- The rawfile Results.raw will be created in the directory Rawfiles_Location

```
smartspice -hspice -b circuit.in -r RawfilesDirectory
```
- The rawfile circuit.raw will be created in the directory Rawfiles_Directory

Now, if the variable simulator is set to -hspice, the default value of the parameters:
- TNOM: The normal temperature
- TEMP: The Circuit operating temperature, will be set to 25 degrees C by default
Measurement Results (HSPICE mode)

- When the option -hspice is specified on the command line, and a measurement fails due to an out of interval error, a warning is issued and a zero result is returned for that variable.
- This contrasts with the standard SmartSpice behavior in which an error is issued and no result is returned for that variable.
.ALTER and *.mt# Files

- While running in batchmode with the option -hspice, SmartSpice correctly outputs the header information in each .mt# file using alter# as a header for the right most column when the deck contains a .ALTER statement, and index# otherwise.

- The data in the right most column is the alter index or vector index, respectively.
If you have PSPICE files and would like to use SmartSpice to run the simulation, you will discover that SmartSpice is not performing the same analysis as PSPICE. The reason is that there is some PSPICE syntax in input files that SmartSpice can't recognize.

The following is a sample of PSPICE deck format:

```
.NODESET V(N_0001)=1
.NODESET V(N_0002)=1
.IC V(Vxxx_1)=0
.IC V(Vxxx_2)=0
.IC V(Vxxx_3)=0
.IC V(Vxxx_4)=0
** Analysis setup **
.tran 1.520ms 1.520ms 0 2us
.OPTIONS RELTOL=0.001
.OPTIONS PREORDER
.TEMP 27* From [PSPICE NETLIST] section of pspice91.ini:
.lib "library1.lib"
.lib "library2.lib"
.lib "library3.lib"
.lib "library4.lib"
.inc "include1.inc"
.inc "include2.inc"
.inc "circuit1.cir"
.INC "test_case.net"
.INC "test_case.als"
.INC "test_case__probe.inc"
.OPTIONS POST
.OPTIONS PROBE
.OPTIONS RAWPTS=100
.probe I(VIN)
.probe I(L1)
.probe V(SW)
.probe V(vout)
.END
```
To resolve this conversion issue two steps are needed. This first step is to use `set simulator=pspice` and `set use_syntax0_libs=true` before running a SmartSpice simulation. These two commands will change parsing to improve PSPICE compatibility.

The second step is to improve the SmartSpice compatibility of the input files. The following are procedures needed to be taken before sourcing the deck in the prior example:

- **Step 1**
  1. Type `set simulator=pspice` in the SmartSpice main command window.
  2. Type `set use_syntax0_libs=true` in the SmartSpice main command window.

- **Step 2**
  1. Put `.LIB` entry name and `.ENDL` inside each library file.
  2. Remove all `$` in the spice deck because `$` is used for comment statements.
  3. Remove the spaces of `+` inside the include file.
  4. Globally change `{` to `(` and `}` to `)` inside the include file.
  5. Comment out the alias file in the spice deck.
  6. Change the LEVEL to be 8 in the library file.
  7. Change to `.tran 1.520us 1.520ms 0 2us UIC` in the `.CIR` file.
  8. Change to `.probe I(V_VIN)` in the `.CIR` file.
In GUI mode in UNIX, LINUX, and Windows, you can also open Set window under System in the main SmartSpice window

- You can type simulator=pspice and then press Set
- Also, type use_syntax0_libs=true and then press Set
- Once you run the simulation, you can immediately go into PSPICE mode

In batchmode you can save these two by setting set simulator=pspice and set use_syntax0_libs=true

- For UNIX or LINUX place these in SmartSpice.ini
- For Windows, place these in smspice.set
- Once you type a SmartSpice command, you can immediately go into PSPICE mode

Inability to use an include file.

- This problem is caused by a difference in syntax between PSPICE and SmartSpice for the .probe command
- To resolve this issue the .probe N(x) needs to be changed to .probe V(x)
- This change will allow the probe.inc file to be imported into the simulation
After taking the conversion procedures, the input deck would be something like in the following:

```
.NODESET V(N_0001)=1
.NODESET V(N_0002)=1
.IC V(Vxxx_1 )=0
.IC V(Vxxx_2 )=0
.IC V(Vxxx_3 )=0
.IC V(Vxxx_4 )=0
** Analysis setup **
.tran 1.520us 1.520ms 0 2us UIC
.OPTIONS RELTOL=0.001
.OPTIONS PREORDER
.TEMP 27* From [PSPICE NETLIST] section of pspice91.ini:
.lib "library1.lib" TT
.lib "library2.lib" TT
.lib "library3.lib" TT
.lib "library4.lib" TT
.inc "include1.inc"
.inc "include2.inc"
.inc "circuit1.cir"
.INC "test_case.net"
.INC "test_case.als"
.INC "test_case__probe.inc"
.OPTIONS POST
.OPTIONS PROBE
.OPTIONS RAWPTS=100
.probe I(V_VIN)
.probe I(L_L1)
.probe V(SW)
.probe V(vout)
.END
```

Remember to use set simulator=pspice and set use_syntax0_libs=true. These two commands will make PSpice to SmartSpice conversion easier, and produce a more accurate and faster run time than PSpice.
spiceserver is a simple program to improve the license check out time between two consecutive batchmode simulations. This program contains three tasks:

1. Start a license.
   ```bash
   spiceserver -start [-n <project name>] [-h <host>] [-v <smartspice version>]
   ```
2. Run simulation.
   ```bash
   spiceserver [-run] -f <deck file> [-o <output file>] [-e <error file>] [-r <result file>] [-n <project name>]
   ```
3. Return license(s).
   ```bash
   spiceserver -stop [-n <project name>]
   OR
   spiceserver -stopall
   ```

**Usage:**

1. -start: Check out the license
   ```bash
   -n: Specifies project name. Each project name will check out one SMARTSPICE license. If -n is not specified, spiceserver will only allow one license to start
   -h: Specifies where to run the simulation. If -h is not specified, the simulation will run on the local machine
   -v: Choose a version of SMARTSPICE. If -v is not specified, SMARTSPICE will use the default version
   ```
2. -run: Start the simulation
   ```bash
   -f: Specifies input file name
   -o: Specifies output file name
   -e: Specifies the error file name
   -r: Specifies the rawfile name
   -n: Simulation will run on specified project if applicable. Otherwise, SMARTSPICE will run on the default project
   ```
3. -stop: Return the license. OR -stopall: Return all licenses
   ```bash
   Note: If -stop or -stopall is not specified, all the license(s) that have been checked out will remain active
   ```
Example:

A script can be written to automate these processes as follows:

- `spiceserver -start`
- `spiceserver -f inputdeck.in -o outputfile.out -r rawfile.raw`
- `spiceserver -f inputdeck2.in -o output2.out -r rawfile2.raw`
- `spiceserver -stop`

The first statement starts a project and checks out a license.

The second statement simulates `inputdeck.in`, and generates `outputfile.out` and `rawfile.raw`.

The third statement simulates `inputdeck2.in`, and generates `output2.out` and `rawfile2.raw`.

The fourth statement returns the license.
Shortening design cycles through a design methodology can be achieved using many tools and techniques.

In the case of analog or mixed-signal design and verification, especially in the IC segment, SPICE simulation is a recognized bottleneck.

The throughput of a design team can be hampered by very long device-level SPICE simulations.

In the physical verification arena for example, complex chips tend to yield millions of parasitic elements that are grafted into an extracted post layout netlist.

SPICE simulations on this type of circuit typically last days per run.

For all practical purposes, SPICE simulators today can only handle partial chip level netlists, like a critical path or a clock tree within a multi-million transistor chip.

Techniques like netlist reduction or table lookup methods to speed up SPICE simulations are common place now.

Furthermore, parallel computing has emerged as a viable add-on to shave off some more simulation time.

Using multi-CPU computers to run simulations is spreading quickly in design companies for three main reasons:

1. Multi-CPU computers have become affordable to even small startup companies.
2. Few software vendors have produced commercial-grade parallel versions of their software.
3. A multi-CPU computer tends to pack more memory than a single processor computer, which in turn speeds up single-CPU simulation runs by avoiding paging.

Simucad has pioneered the EDA industry in the field of parallel computing, jumping on the bandwagon at an early stage during the nineties.

Several Silvaco TCAD driven CAD tools support parallel computing, including SmartSpice, its industry standard analog simulator.

For many years, Simucad customers have been able to run SmartSpice on multiprocessors, be it Sun servers, HP servers, or multiprocessor PCs running Windows or Linux.

Nevertheless, performance of a parallel application is still highly misunderstood. “Why isn't my application going four times faster using 4 CPUS?” is a frequently asked question.
Cray vector computers feature multiple adders and multipliers within a CPU. No matter how your application software was written, Cray would make it go faster, as long as your application dealt mainly with arrays of data and performed vector operations.

Thinking Machine (TMC) Computers using a two-dimensional array of ALUs used the same approach. The same instruction of code was executed at the same time on a two-dimensional array of data. This is known as the SIMD (Single Instruction, Multiple Data) computing model.

Unlike vector and SIMD computers, today's multiprocessor computers are built using commercial off-the-shelf microprocessors. The application programmer is left with two basic models of parallel computing:

1. Task parallelism: You have many unrelated tasks to perform and you distribute them among the available processors.
2. Data parallelism: You only have one task to perform on a large data set. Distribute the data on the processors, and each will perform the same task on its subset of data.
Task Parallelism in SmartSpice

- Task parallelism
- Well suited to DOE (Design Of Experiment) type scenarios where you run multiple unrelated simulations.
- In this case, an instance of the simulator is forked for each data set, and they all run in parallel on the same machine.
- This is the pinnacle of parallel computing.
- Barring shared resource contentions, like memory and I/Os, using 4 CPUs to run 4 simulations would effectively yield a 4x speedup compared to running all 4 simulations on one CPU.
- This is the approach taken by SmartSpice when a .ALTER statement is encountered in the command input file.
- A .ALTER statement directs the simulator to rerun a similar simulation while varying one or more parameters of the design.
- Reading in and parsing the input file could be a task consuming operation depending on the size and hierarchy of the design.
- Therefore, the time spent parsing the netlist, which is always done on one CPU, is an overhead as far as parallel computing is concerned.
- Only the numerical simulation itself is parallelized.
- Unfortunately, the effect of a relatively small portion of the overall execution time that remains serial can be devastating to the global speed improvement.
- Even if only 10% of the 1 CPU runtime remains sequential in the parallel run, using 8 CPUs will only yield a maximum speedup of 4.7.
- This is assuming the optimal case of the parallel execution being at 100% performance, which is seldom the case in reality.
- Most SPICE simulations require a steady state evaluation in DC mode followed by a transient analysis over a certain period of time.
- The time stepping approach requires the knowledge of node voltages and branch currents at each time step before continuing on to the next time step.
- So by nature, the calculations are serial between time steps, and no parallelism can be invoked at this level.
Data Parallelization in SmartSpice

- Data parallelism
- The above situation is why SmartSpice relies on data parallelism within each time step calculation, in all simulations not involving a .ALTER statement
- This means that the parallelization needs to be effective at each time step calculation
- Therefore, if a simulation takes a long time to execute because it performs a very large number of time steps, there is no guarantee that parallel SmartSpice will be able to reduce that execution time
- It all depends on what happens within each time step, as will be explained below
- Data parallelism has a major drawback; it is only effective if the calculations on the different data subsets are independent
  - Otherwise, synchronization is required to ensure data integrity so that:
    - No CPU is using an out of date value of a variable
    - No CPU is using a variable value that has not yet been calculated
    - No 2 CPUs are attempting to update the same variable at the same time
  - In a real execution, these constraints translate into a stop-and-go execution by all the CPUs involved
  - CPUs stop calculating to wait for other CPUs to calculate needed values or access shared resources
  - In particular, parallel SmartSpice exhibits this behavior at two levels:
    - 1. Element model evaluation: For any device model (transistor, resistor, capacitor, etc.), SmartSpice evaluates the model equations and submits results into a global admittance matrix This matrix is shared among all the CPUs, and therefore a synchronization is required whenever one of the CPUs tries to update the content of this matrix. This is an overhead
    - 2. Linear solver: SmartSpice relies on an LU factorization to solve the linear system of equations within each time step. Without going into details, a lot of stop-and-go happens at this stage
Data Parallelization in SmartSpice (cont)

- Data parallelism (cont’d)
  - Usually, the element model evaluation (also known as Load) parallelizes fairly well.
  - As a general rule, the parallel efficiency is better when:
    - The circuit has a lot of devices (thousands at least).
    - There are more active devices (transistors). It doesn't help having too many RCs.
    - The device models used are more complicated.
    - It is not unusual that the efficiency of the model calculation reaches 90% or more of CPU utilization on a multiprocessor computer.
  - On the other hand, the efficiency of the linear solver is not very well characterized, although a few comments can be made. In general, the parallel solver's efficiency deteriorates when:
    - The circuit is small.
    - The circuit's topology is almost a linear array of devices.
    - A lot of extracted parasitics with small values are present.
    - In SmartSpice, these two phases (element model evaluation & linear solving) of the calculation can be monitored using the .OPTIONS ACCT=2 statement.
As shown in the following example, the Load time reflects the time spent evaluating the device models, and the L-U decomposition time reflects the time spent calculating the LU decomposition in parallel.

User Requirements
To run the simulation on multiple CPUs detected by SmartSpice you need one full SmartSpice license and a number of thread licenses (or a multi-core license).

For example, if SmartSpice detects four CPUs, you will require one SmartSpice license and three thread licenses (or a multi-core license).

Example
Running on 1 CPU:
smartspice -b <input-file> (NOTE: smartspice now defaults to parallel mode if license available)
equations (Circuit Equations) = 2540
loadtime (Load time) = 186.83
lutime (L-U decomposition time) = 310.83

Running on 2 CPUs:
Smartspice -b -P 2 <input-file>
equations (Circuit Equations) = 2540
loadtime (Load time) = 98.29
lutime (L-U decomposition time) = 212.83
If the deck contains one or more .ALTER statements and you specified -P n (and not -nodist) in the command line, where n is the number of CPUs, SmartSpice’s internal functions will read the deck, extract part of the netlist which form entire circuits, and write out each circuit in the same directory where the composite netlist is situated.

It also files the names of the .ALTER statements, which are contained in the table pointed to by names built up from the names of the composite basic netlist with the suffixes -n (n is the number of the CPU which is taken by extra process) at the right side.

The name of the file does not contain an extension.

After running SmartSpice, you will create an extra child process.

Each child process will handle a separate circuit, which is situated in a separate file without an extension.

The parent process waits for the children to finish. If we specified --P 4, then 4 extra processes will be created by the parent process of SmartSpice.

The total amount of SmartSpices in the memory is 5, including 4 child processes and 1 parent.

After finishing the simulation, the child will remove its own file-circuit for you.

The .out files, .raw files, and .err files will have the suffix --n, which says what part of the basic netlist is described.

Four .raw, .out, and .err files are created, one for each subnetlist from the composite altered netlist.

The above mechanism is a powerful tool for increasing the speed of the simulation, and obeys the following rules:

We want to spawn additional SmartSpice processes, subject to the following constraints:

1. The number of subprocesses spawned should at no time exceed the number of specified CPUs on the command line, with option -P. In addition, this should not exceed the actual number of CPUs on the machine. Therefore, on a single-CPU machine, no subprocesses will be spawned.

2. If there are more .ALTERs than CPUs, the .ALTERs should be queued until CPUs are freed up.

3. One (full) license should be used for the first SmartSpice, and one multi-threaded license for each subprocess. OR One Multicore license.

Parallel alter will be canceled if you specify the option --nodist in the command line of SmartSpice.
SmartSpice Special Functions

- **.ST statement**: Allows repeated calculation while stepping model parameter values
- **.MEASURE statement**: Measures user-specified circuit activity, thus reducing the volume of output data and minimizing calculation time
- **.MODIF statement**: Allows you to investigate the behavior of a circuit for the same sets of parameters as they are modified over a user specified range of values
A secure and robust encryption engine is available to encode everything from the device model level up through subcircuits to any level, including the full SmartSpice input deck.

- Encrypted model and circuit libraries are treated as "black boxes" providing you with access only to terminal functions.
- You cannot view or print the encrypted netlists, internal node voltages, or parameters.
- The simulation process flow is exactly the same as without encryption so that you do not have to change their design and verification flows.

This program comes in 4 versions to cover the 4 major supported operating systems.

The programs are located in the contents of the .zip package supplied:

- readme: References a PDF file
- encryption.pdf: PDF Guide to Encryption operation
- Folders containing the relevant executable: hp 700-hpux 901, i386-linux, sparc-solaris 2, x86-64-linux, or x86-nt
- This is a one-way process, no De-Encrypt program is provided so it is important to keep the source file safe and not discard this.

1. Run the encryption program “encrypt input_file output_file e”
2. In your Spice deck use “.include output_file”
This allows SmartSpice to run the encrypted file, but if anyone tries to print the file they will get a “jumbled mess” that is not understandable.

Encryption can be done at any level of the SmartSpice input deck structure.

This means right from the main netlist to included files and model libraries down to individual models file contents can be protected and unreadable.

This feature has already found applications, such as joint company ventures, where netlist circuit blocks can be used for simulation, but Intellectual properties are confidential.

Since these encrypted files are only usable inside SmartSpice, the program recognizes the contents should be protected and all normal functions that would allow viewing of the contents are automatically disabled.

This ensures the protected circuit can be evaluated for electrical performance but the contents remain confidential.

Because the encryption process is made so secure there is no way to reconstruct the original contents.

It is recommended to keep 2 versions of the files as:

- Encrypted file for 3rd party use
- The original readable file