SpectreRF Tutorial

SpectreRF provides about a dozen new simulation modes targeted primarily for RF designers but which also have applications in, for example, sample-and-hold circuits and switched-capacitor filters. In this tutorial we focus on the periodic steady state (*pss*), periodic ac (*pac*), and periodic noise (*pnoise*) analyses.

First, we will provide a simple example to motivate the need for these tools. Suppose you need to know the transfer function of a simple RC low-pass filter (Figure 1a). As you know from previous courses, you would run a *dc* simulation to calculate the operating point and then an *ac* simulation to calculate the transfer function. The *dc* simulation is a large-signal simulation, and the *ac* simulation is a small-signal simulation which is calculated around that operating point.

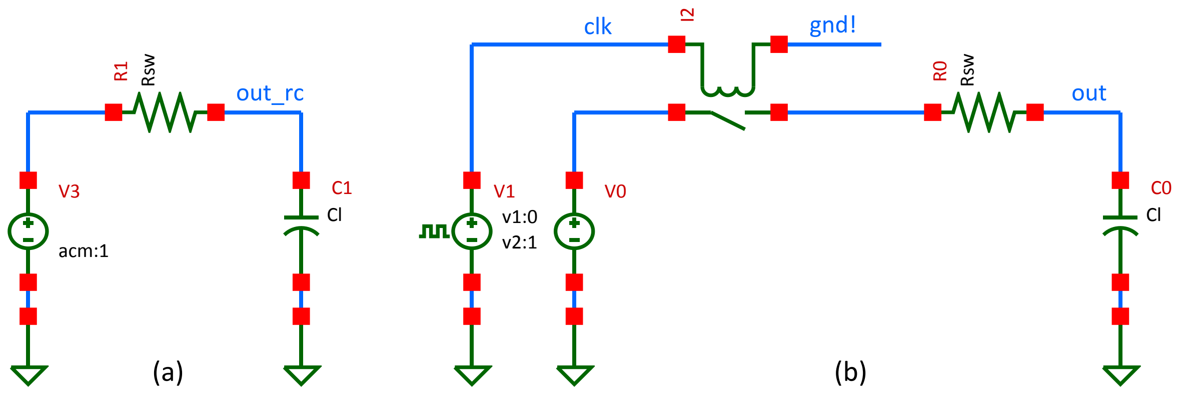


Figure 1. (a) Low-pass RC schematic. (b) Track and hold circuit schematic

Now suppose that you need to simulate a track and hold circuit. Schematically, we have just added an ideal switch in series with the resistor which is clocked at the sampling frequency (Figure 1b). The output voltage of this circuit is clearly dependent on the time-varying characteristic of the switch, an effect which is not captured in a *dc* operating point simulation. In fact, the *dc* simulation will either assume that the switch is always open or closed depending on the default value of the clock.

Instead, we need to use the *pss* and *pac* simulations. These simulation modes are analogous to *dc* and *ac* simulation, respectively, but are used for periodically varying circuits. First, the *pss* simulation calculates the periodic large-signal operating point including the periodically time-varying (and therefore frequency-translating) characteristics of the switch. Then, the *pac* simulation uses this periodic operating point to calculate the ac response from any input harmonic to any output harmonic, and we can sample this response at the clock frequency. Similarly, the *pnoise* simulation is analogous to the *noise* simulation but is applied to periodically-varying circuits and therefore accounts for frequency-translation (such as noise folding) and time-varying noise sources.

This document first walks through the use of *pss*, *pac*, and *pnoise* simulations on a simple circuit example. Then, it provides some insight into how the simulator actually works. Some familiarity with the Cadence Virtuoso software is expected, at least at the level of the EE315 Virtuoso tutorial (creating a schematic, adding schematic cells and modifying their properties, running simple simulations in ADE L).

Simulation Example

For this example, we will use the same track and hold circuit shown in Figure 1b on the previous page. This schematic is provided in the ee315 library as *example_pss* so you must first copy this to your own library. If you haven't made a new library yet, "ee315_work" would suffice for this and future homework assignments.

Open the schematic and look at the properties of each device. Specifically, notice that the clock source (V1) is set to a variable frequency (we'll set this in ADE L), and the switch is nearly ideal (1 mOhm and 1 TOhm on and off resistances, respectively). Also, compare the two voltage sources. Notice that the input source V0 is only given a (small-signal) PAC magnitude, whereas the clock source V1 is a large-signal source. If you made the input a sinusoid, for example (using the *vsin* cell), that would be a large-signal source which would mistakenly be accounted for during the *pss* simulation. During *pss*, we want to find a periodic operating point which means the input signal of interest needs to be set to small-signal so it is ignored. These signals are taken into account during the *pac* simulation.

In this example we have parameterized the device parameters around the settling time constant N as discussed in class. As a brief review, recall that N=(Ts/2)/RC and that N=4.6 corresponds to about 1% settling and N=9.2 corresponds to about 0.01% settling. By using this parameterization we can easily vary N in order to understand its effects on both the transfer function and sampled noise.

Launch ADE L and copy the variables from the schematic (Variables → Copy From Cellview). Set the period to 1 ns for a 1 GHz sampling frequency, the capacitor to 1 pF, and N=10. Later, we will investigate what happens when we decrease N.

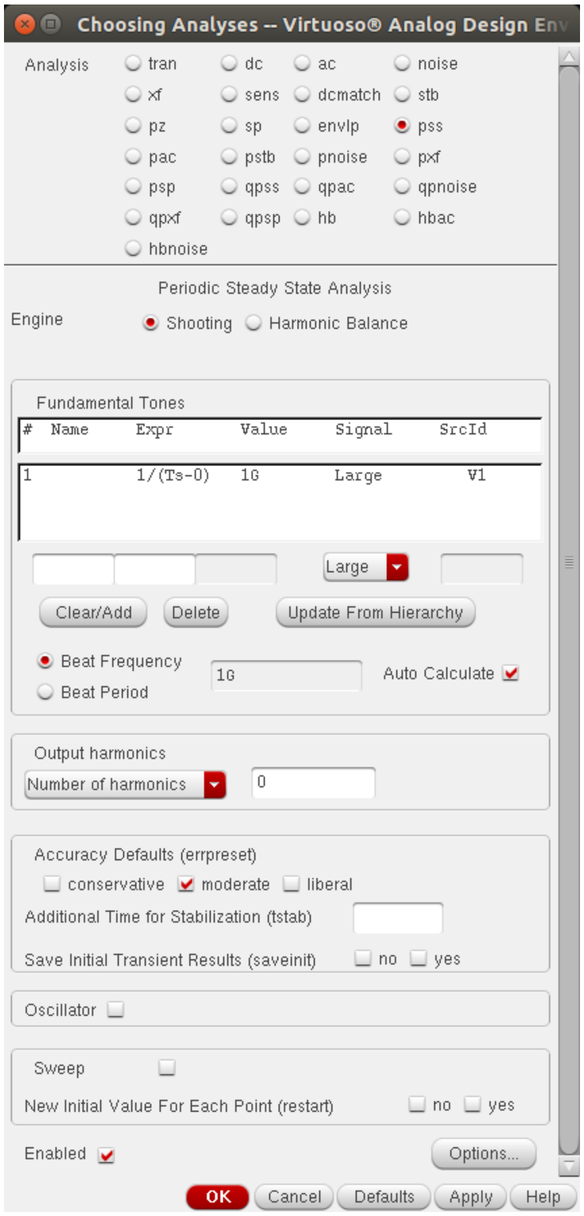
How to run a *pss* simulation

Add a new analysis (Analyses → Choose…). Select *pss* since this is the first simulation that we will have to run. Keep the Engine set to *Shooting*. The shooting method uses a time-domain simulation to solve for the periodic steady-state solution, as opposed to harmonic balance which works in the frequency domain. We will only work with the shooting method in this document.

Since this is a periodic simulation we need to set the fundamental period of the circuit, called the beat frequency (this term is borrowed from acoustics). You can set this manually, but in our case we'll check the *Auto Calculate* box. Additionally, set Number of harmonics to 0. Select the *moderate* accuracy setting, which has a good balance between speed and accuracy.

Now click on the Options button and navigate to the Accuracy tab. The *maxacfreq* parameter sets the maximum frequency that will be taken into account for small-signal simulations (*pac*, *pnoise*). By default this value is set to 40x the beat frequency which will not be enough for the *pnoise* simulation due to noise folding, so you should set it to *200G* (200 GHz).

Press OK to finish. The properly configured *pss* window is shown in Figure 2.



Analysis: tran, dc, ac, noise, xf, sens, dcmatch, stb, pz, sp, envlp, [✓ pss], pac, pstb, pnoise, pxf, psp, qpss, qpac, qpnoise, qpxf, qpsp, hb, hbac, hbnoise

Periodic Steady State Analysis

Engine: [✓ Shooting] Harmonic Balance

Fundamental Tones

#	Name	Expr	Value	Signal	SrcId
1		1/(Ts-0)	1G	Large	V1

Large

[✓ Beat Frequency] 1G Auto Calculate [✓]

Beat Period

Output harmonics: Number of harmonics 0

Accuracy Defaults (errpreset): conservative [✓ moderate] liberal

Additional Time for Stabilization (tstab)

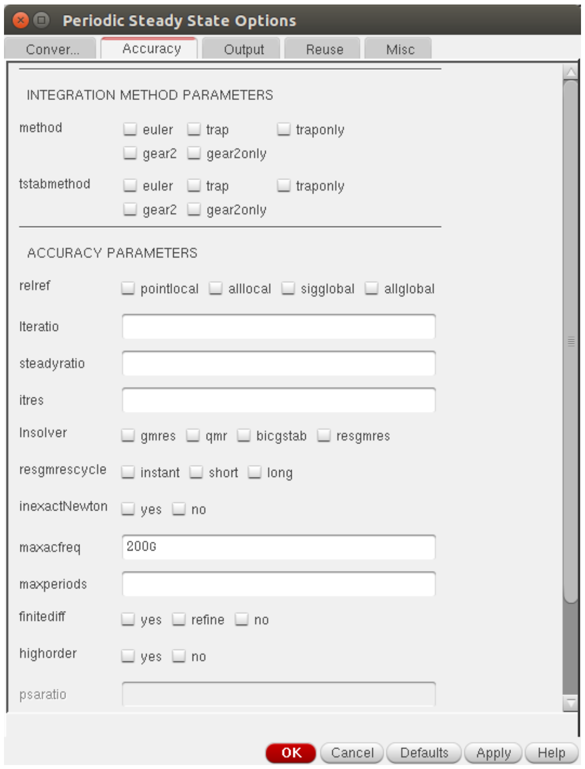
Save Initial Transient Results (saveinit): no yes

Oscillator

Sweep

New Initial Value For Each Point (restart): no yes

Enabled [✓]



INTEGRATION METHOD PARAMETERS

method: euler, trap, traponly, gear2, gear2only

tstabmethod: euler, trap, traponly, gear2, gear2only

ACCURACY PARAMETERS

relref: pointlocal, alllocal, sigglobal, allglobal

lteratio

steadyratio

itres

lnsolver: gmres, qmr, bicgstab, resgmres

resgmrescycle: instant, short, long

inexactNewton: yes, no

maxacfreq: 200G

maxperiods

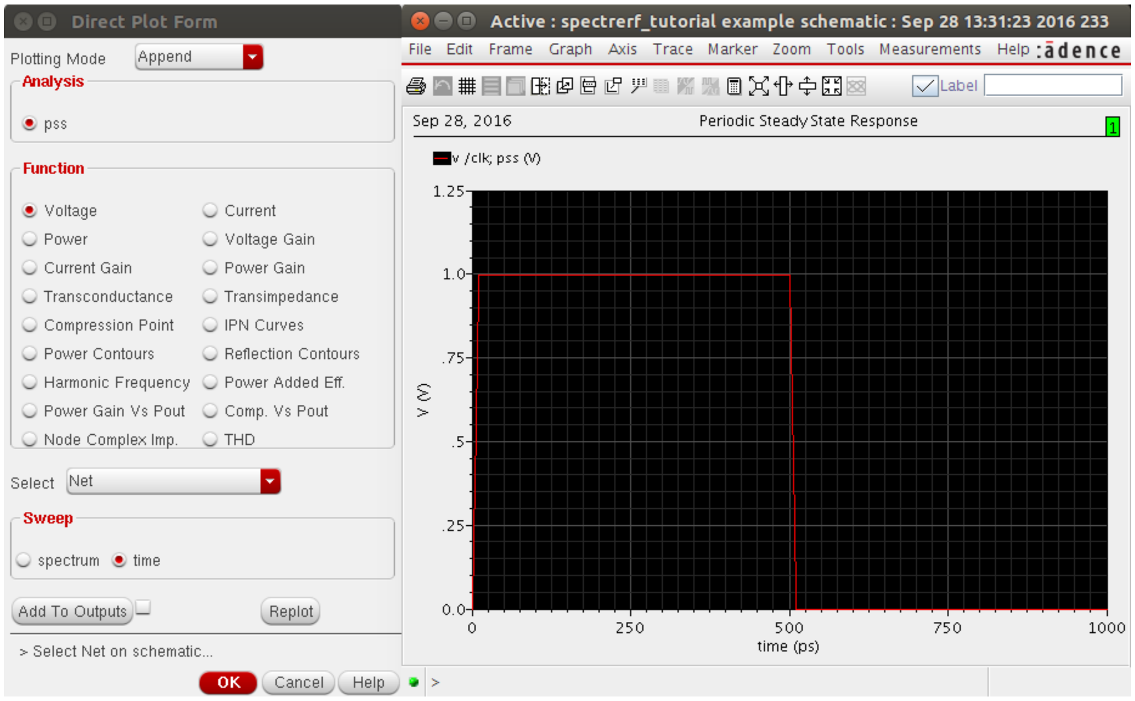
finitediff: yes, refine, no

highorder: yes, no

psaratio

Figure 2. Analysis window configured for *pss* simulation

Click Netlist and Run to run the simulation. There isn't much to look at yet, but we can check the clock waveform. To do this, go to Results → Direct Plot → Main Form from the ADE L window. Configure the form as shown in Figure 3 below, and then click on the *clk* net on the schematic to plot it. The expected output is also shown in Figure 3, which is a 50% duty cycle square wave with 1V amplitude.


Direct Plot Form
Active : spectrerf_tutorial example schematic : Sep 28 13:31:23 2016 233
File Edit Frame Graph Axis Trace Marker Zoom Tools Measurements Help
cadence
Label
Sep 28, 2016
Periodic Steady State Response
v /clk; pss (V)
1.25
1.0
.75
.5
.25
0.0
V (V)
0
250
500
750
1000
time (ps)
Add To Outputs
Replot
OK
Cancel
Help

Plotting Mode: Append

Analysis

[✓ pss]

Function

[✓ Voltage]	Current
Power	Voltage Gain
Current Gain	Power Gain
Transconductance	Transimpedance
Compression Point	IPN Curves
Power Contours	Reflection Contours
Harmonic Frequency	Power Added Eff.
Power Gain Vs Pout	Comp. Vs Pout
Node Complex Imp.	THD

Select: Net

Sweep

spectrum [✓ time]

> Select Net on schematic...

Figure 3. Configuration and plot of clock time domain waveform from *pss* simulation

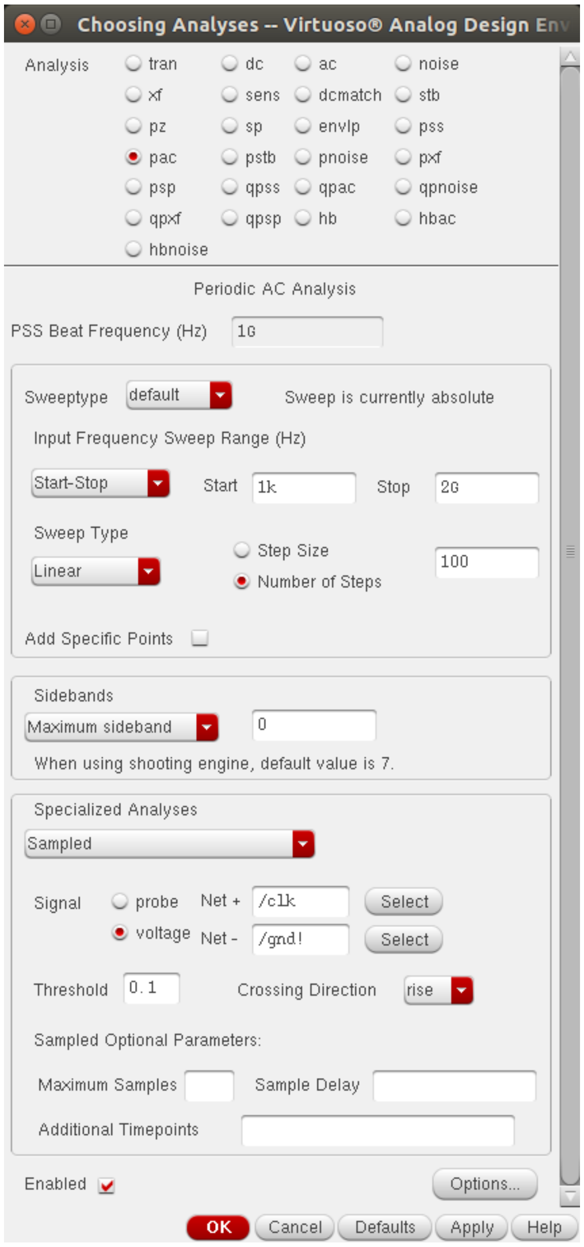
How to run a *pac* simulation

Now that you have configured and run *pss*, we can use that result to run small-signal analyses. First, we will run a *pac* simulation to look at the small-signal transfer function from the input to output.

A brief aside: You may notice a simulation mode called *pxf*, which stands for periodic transfer function. Since we're calculating a transfer function here, you might wonder why we aren't using that. Generally, *pac* is used to calculate the transfer function from one input to any output node, whereas *pxf* is used to calculate the transfer function from any number of inputs to a single output. Since we have one input and one output here, either tool would work.

To set this up, add a new analysis and select *pac*. Set the input frequency to *Start-Stop* from 1k to 2G and Maximum sideband to 0. Also, change the Sweep Type to *linear* and set it to 100 total points. Under Specialized Analyses, pick *Sampled*. We will trigger this on the rising edge of the clock, so set the Signal to a *voltage* between */clk* and */gnd!* with a Threshold of 0.1 and Crossing Direction of *rise*.

This works because of how the switch cell is configured. The switch element that we use has upper and lower thresholds. Above the upper threshold, the switch is on (Rsw = 1 mOhm). Below the lower threshold, the switch is off (Rsw = 1 TOhm). In between, the resistance is linearly interpolated. Since the upper and lower thresholds in schematic are set to 0.499 V and 0.501 V respectively, by triggering at 0.1 V we are sampling at an instant right before the switch turns back on again. (Alternately, you could set the threshold to an arbitrary value and manually add an Additional Timepoint at, say, 750 ps, in the middle of the off period. Or, create a second, delayed version of the clock to use as the trigger. There are many valid methods.)



Analysis: [✓ pac]

Periodic AC Analysis

PSS Beat Frequency (Hz) 1G

Sweeptype default   Sweep is currently absolute

Input Frequency Sweep Range (Hz)

Start-Stop   Start 1k   Stop 2G

Sweep Type

Linear   Step Size   [✓ Number of Steps]   100

Add Specific Points ☐

Sidebands

Maximum sideband   0

When using shooting engine, default value is 7.

Specialized Analyses

Sampled

Signal   probe   [✓ voltage]   Net + /clk   Net - /gnd!

Threshold 0.1   Crossing Direction rise

Sampled Optional Parameters:

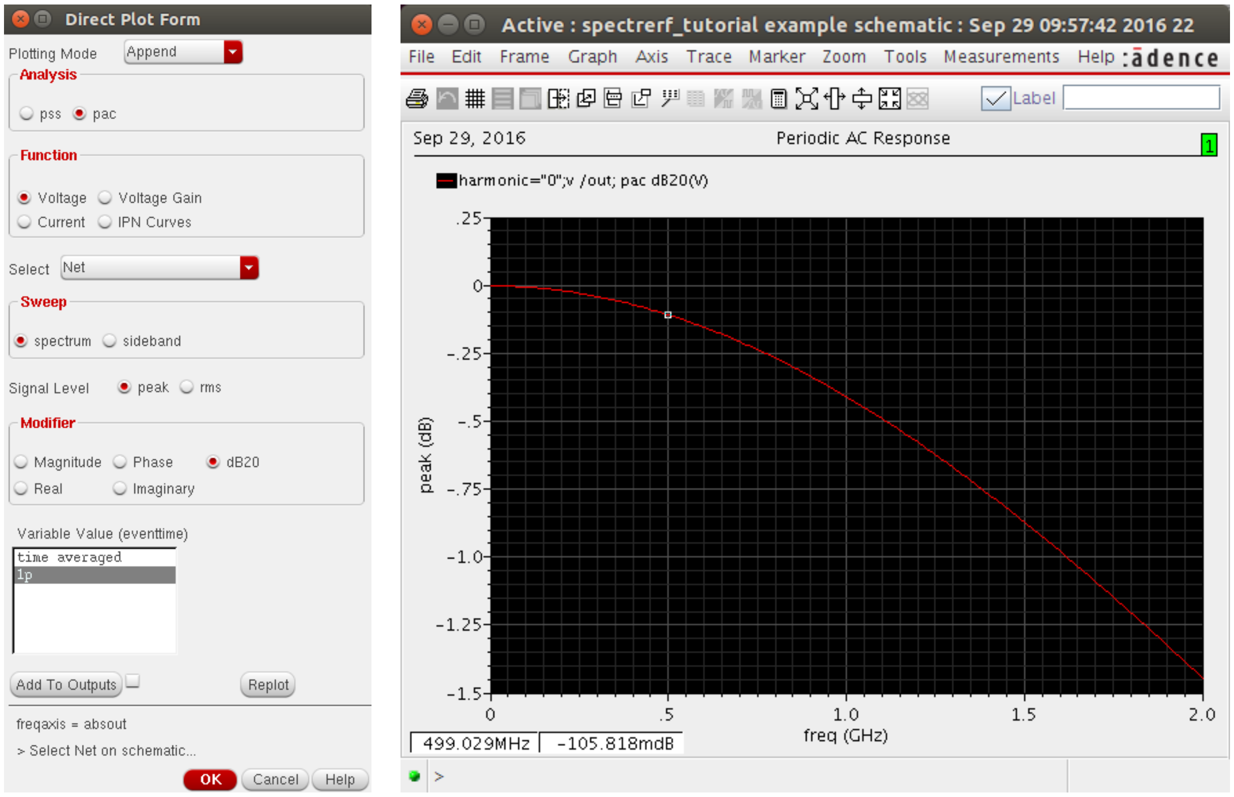
Maximum Samples   Sample Delay

Additional Timepoints

Enabled ✓

Figure 4. Analysis window configured for *pac* simulation

Press OK, and then hit Netlist and Run to start the simulation. To look at the *pac* output, open the Direct Plot window again and configure as shown in Figure 5. Click on the output node, and you will see the transfer function from input to output.


Direct Plot Form

Plotting Mode: Append

Analysis: pss [✓ pac]

Function: [✓ Voltage] Voltage Gain Current IPN Curves

Select: Net

Sweep: [✓ spectrum] sideband

Signal Level: [✓ peak] rms

Modifier: Magnitude Phase [✓ dB20] Real Imaginary

Variable Value (eventtime):
```
time averaged
1p
```
freqaxis = absout
> Select Net on schematic...

Active : spectrerf_tutorial example schematic : Sep 29 09:57:42 2016 22

Sep 29, 2016 Periodic AC Response

harmonic="0";v /out; pac dB20(V)

499.029MHz -105.818mdB

Figure 5. Configuration and plot of transfer function using *pac*

By choosing *eventtime*=1p we are looking at a sampled *pac* output which has very little loss up to the Nyquist frequency. If you plot the time averaged output instead, you would be looking at the continuous-time output of the system which has a loss of about 1.6 dB at Nyquist, versus 0.1 dB for the sampled output. This discrepancy is clear to understand if you look at this in the time domain (Figure 6). The sampled data points can reproduce the input waveform without loss, whereas the continuous-time output is distorted resulting in power loss at the input frequency.

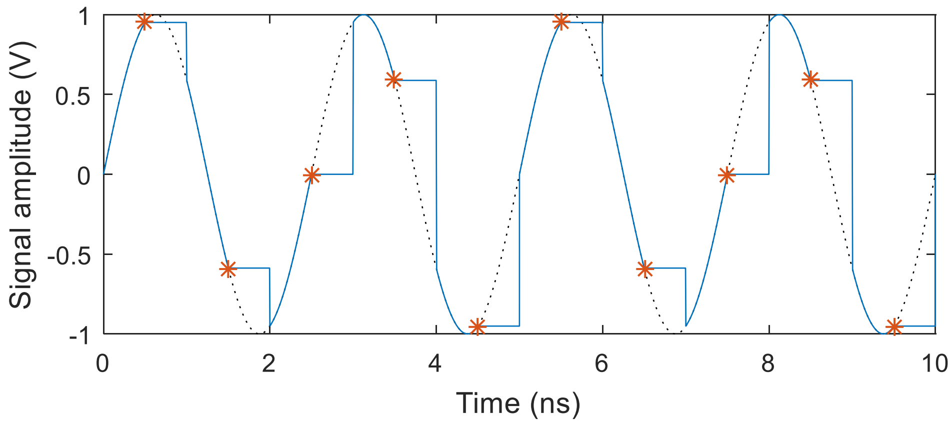


Figure 6. Difference between sampled and continuous circuit output from an ideal (N=∞) track-and-hold with 400 MHz input signal and 1 GHz sample rate

The *pac* plot in Figure 5 can also tell us about the behavior of aliased signals. If you put a 1.75 GHz signal into a 1 GHz sampled system, we know that it will alias down to 250 MHz. According to our simulations, it should see a gain of roughly -1.15 dB. This can be validated (but you don't need to do this) by running a transient simulation with a 1.75 GHz input signal and taking the FFT of the result (Figure 7). Notice that the FFT amplitude at 250 MHz is equal to the *pac* amplitude at 1.75 GHz.

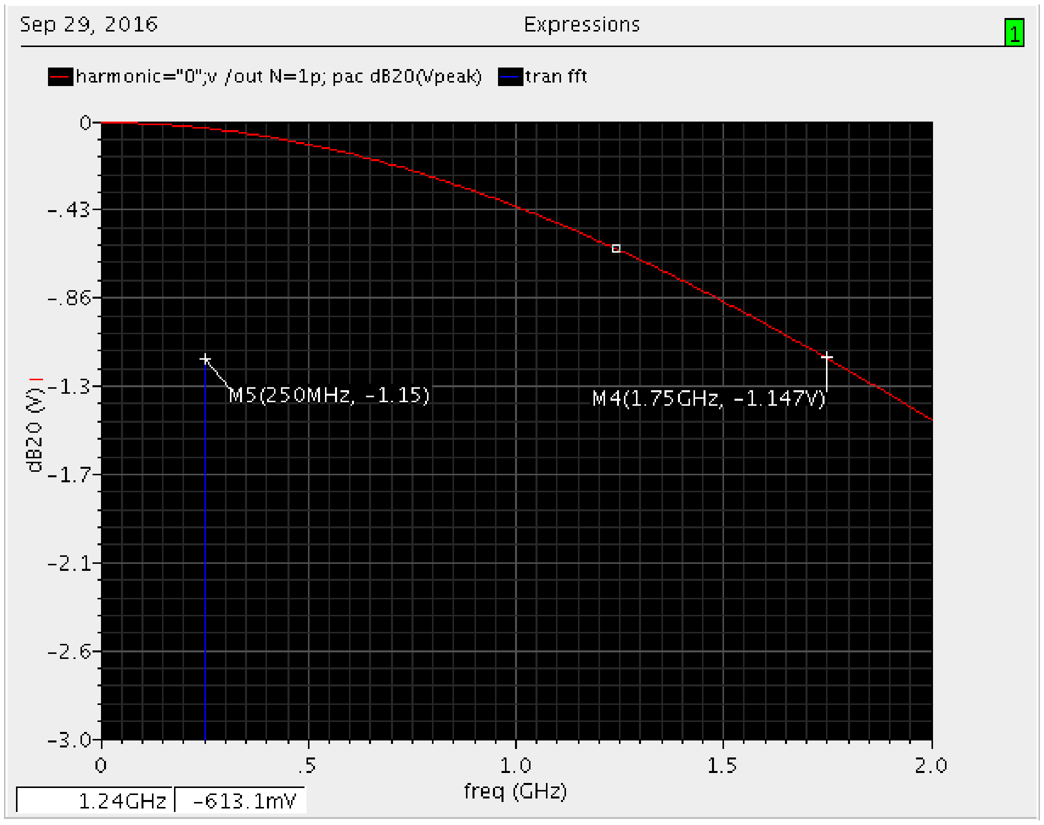
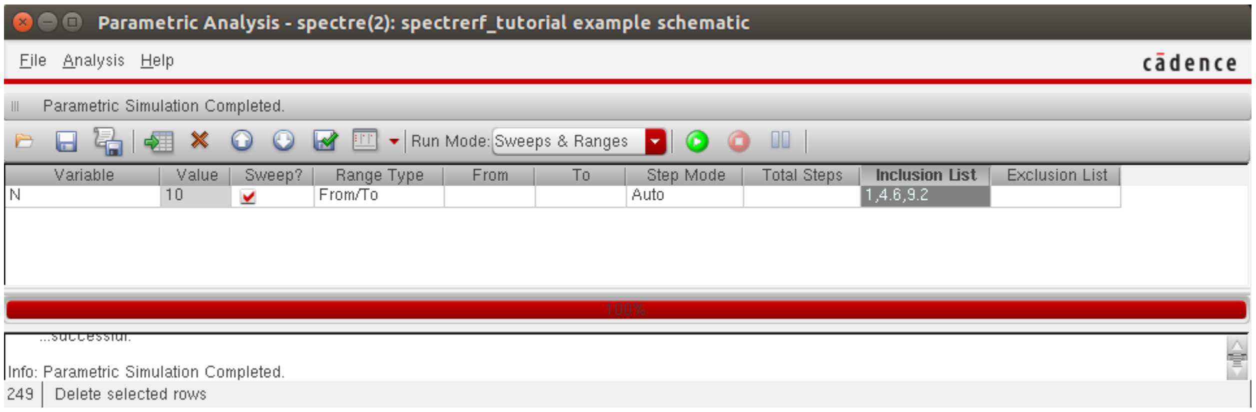


Figure 7. Comparison of *pac* and *tran* output spectrum

Lastly, we can investigate the effects of varying N. To do this, we will go to Tools → Parametric Analysis. This tool allows you to run the same simulation multiple times but with swept variable values. Select the variable N and set the inclusion list to 1, 4.6, and 9.2. This will allow us to compare almost no settling, 1% settling, and 0.01% settling, respectively.


Parametric Analysis - spectre(2): spectrerf_tutorial example schematic
File Analysis Help
cādence
Run Mode: Sweeps & Ranges
100%

Parametric Simulation Completed.

Variable	Value	Sweep?	Range Type	From	To	Step Mode	Total Steps	Inclusion List	Exclusion List
N	10	✓	From/To			Auto		1,4.6,9.2	

```
...successful.
Info: Parametric Simulation Completed.
249 | Delete selected rows
```

Figure 8. Parametric analysis window configured to set specific values of N

Hit run and plot the same output node again. The plot you see should be similar to Figure 9, emphasizing the importance of proper settling in sampled circuits in order to obtain a flat frequency response.

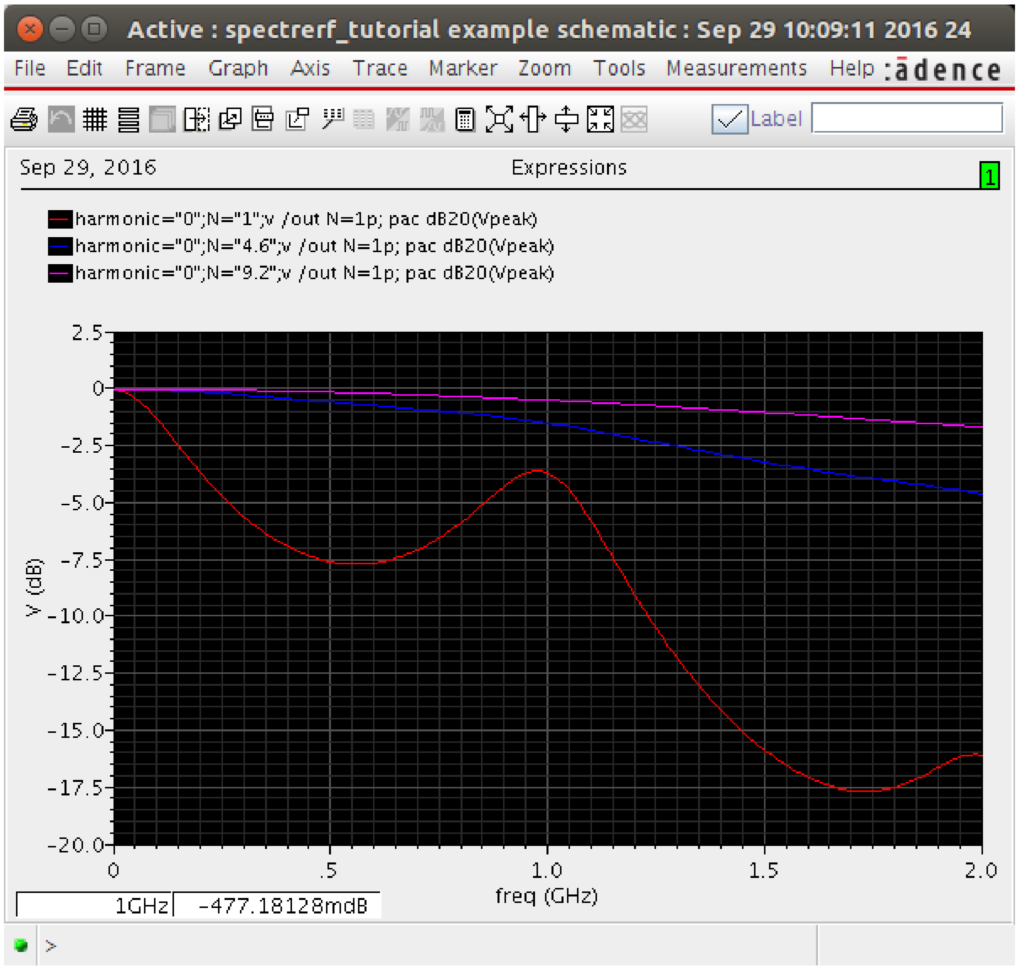
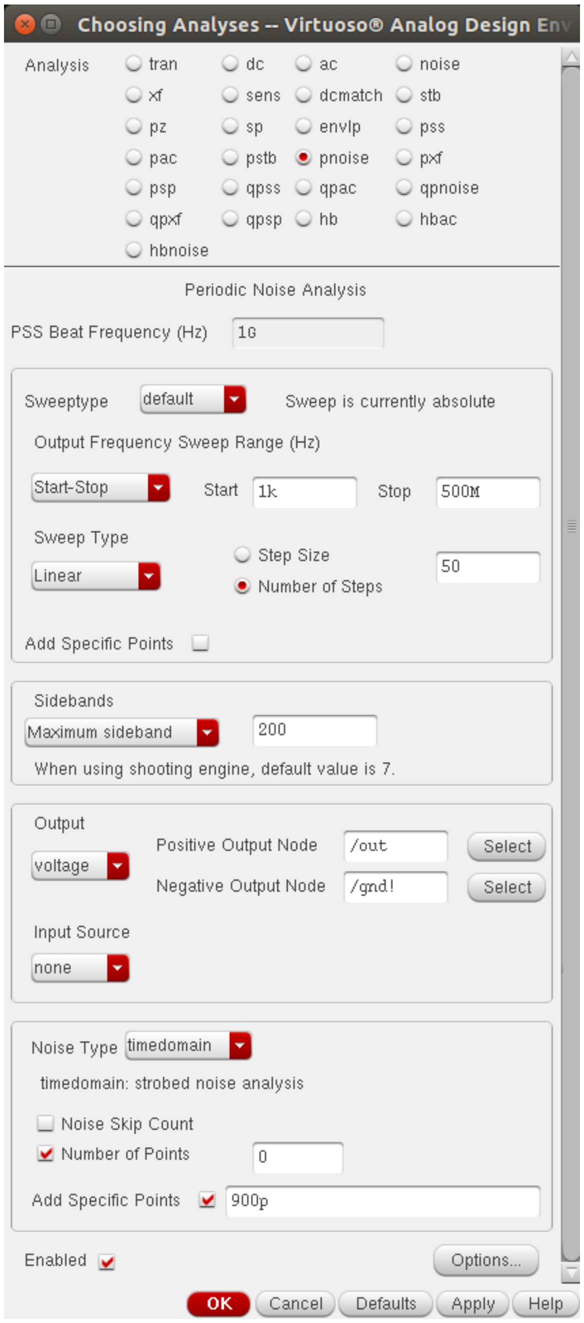


Figure 9. Track-and-hold voltage gain is a function of RC settling

How to run a *pnoise* simulation

Next, we will run *pnoise* analysis to measure the noise sampled on the capacitor. Add a new analysis and choose *pnoise*. Set a 50-point linear sweep from 1 kHz to 500 MHz with Maximum sideband set to 200 to match *maxacfreq* (200 GHz is 200x the fundamental). Set the Output to Voltage, and select the output and ground as the positive and negative nodes, respectively. Set the Input Source to *none*.

We will use a Noise Type of *timedomain*, because we are interested in the noise sampled at a specific point in time. Check Number of Points and set it to 0, and then add a specific point at 900p. Our clock period is 1000 ps long, with the switch conducting until 500 ps and off after that. By sampling at 900 ps, we are sampling the noise at a time when the switch is open and therefore the sample is being held. The correct analysis settings are shown in Figure 10.



Analysis: [✓ pnoise]

Periodic Noise Analysis

PSS Beat Frequency (Hz) 1G

Sweeptype default   Sweep is currently absolute

Output Frequency Sweep Range (Hz)

Start-Stop   Start 1k   Stop 500M

Sweep Type Linear   Step Size [✓ Number of Steps] 50

Add Specific Points ☐

Sidebands

Maximum sideband 200

When using shooting engine, default value is 7.

Output voltage   Positive Output Node /out   Negative Output Node /gnd!

Input Source none

Noise Type timedomain

timedomain: strobed noise analysis

☐ Noise Skip Count

✓ Number of Points 0

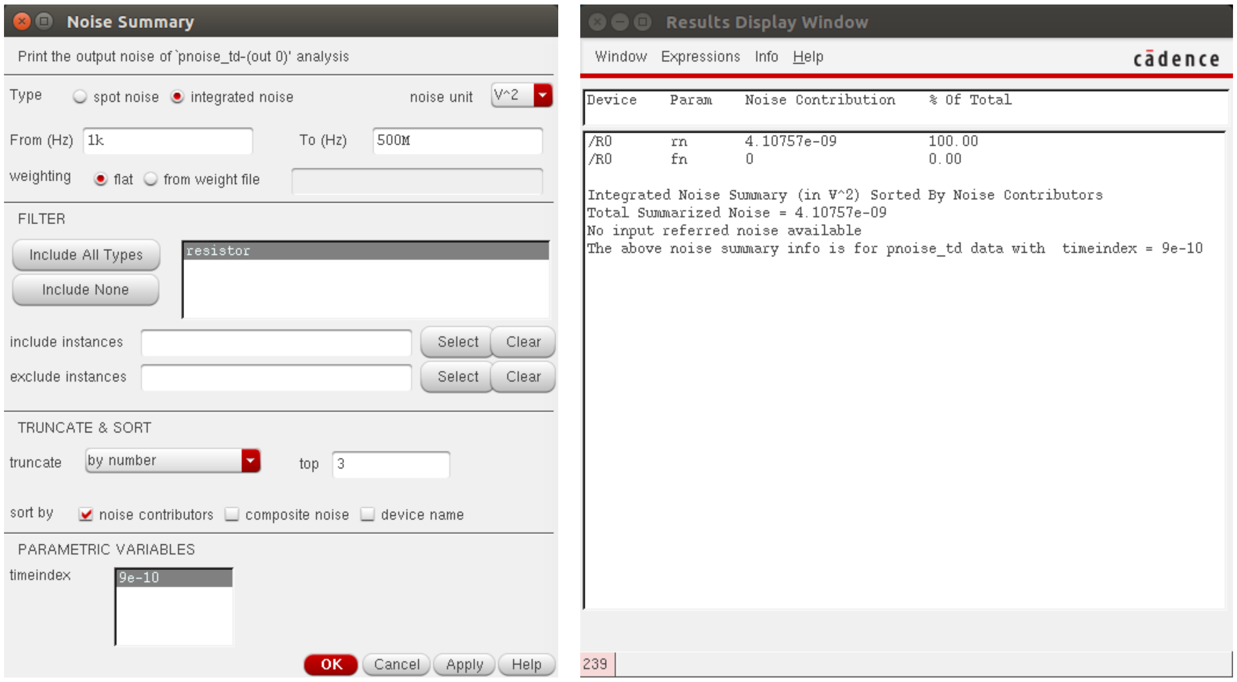
Add Specific Points ✓ 900p

Enabled ✓

Figure 10. Configuration of *pnoise* analysis

Press OK, and then Netlist and Run to run the simulation. We are interested in measuring the total integrated noise sampled at the output to make sure that it matches the theoretical value of kT/C.

There are a couple ways to measure the total integrated noise at the output. The first way is going to Results → Print → Noise Summary. Set the Type to *integrated noise* from 1 kHz to 500 MHz, click *Include All Types* to account for all noise sources (in this case, there's just a resistor), and select the 900p *timeindex*. Then, press Apply. A new window will open which shows the total summarized noise of 4.108e-9 V^2, with 100% of that coming from the switch resistance R0. The Noise Summary configuration and output are shown in Figure 11.



Print the output noise of `pnoise_td-(out 0)' analysis

Type: spot noise [✓ integrated noise] noise unit: V^2

From (Hz): 1k To (Hz): 500M

weighting: [✓ flat] from weight file

FILTER: resistor

include instances:

exclude instances:

TRUNCATE & SORT

truncate: by number top: 3

sort by: [✓ noise contributors] composite noise device name

PARAMETRIC VARIABLES

timeindex: 9e-10

```
Device    Param    Noise Contribution    % Of Total

/R0       rn       4.10757e-09           100.00
/R0       fn       0                     0.00

Integrated Noise Summary (in V^2) Sorted By Noise Contributors
Total Summarized Noise = 4.10757e-09
No input referred noise available
The above noise summary info is for pnoise_td data with  timeindex = 9e-10
```

Figure 11. Using the Noise Summary form to determine the total integrated noise

A second way is to use the calculator (which you should be familiar with from the Virtuoso tutorial). From ADE L, go to Tools → Calculator. From the calculator, go to Tools → Browser and go into *pnoise-timedomain.pnoise*. Right click on the *out* signal and select *Calculator* to send the waveform to the calculator. This is the noise voltage at the output, so we need to square it to get noise power and then integrate it to get total integrated noise. In the lower section of the calculator, select *All* from the dropdown menu and then click on *pow*. The Base should be the noise voltage expression, and Exponent should be 2. Press OK.

Then, select *integ* in the bottom menu with the Initial and Final Values set to 1k and 500M, respectively. Hit OK again. The calculator should now read the following:

```
integ(pow(getData("/out" ?result "pnoise-timedomain.pnoise") 2) 1k 500M)
```

If you hit the plot button now, it will create a plot with a single data point to indicate the noise at 900 ps. Since we only have the one data point, you can just take the average of this one data point to get a scalar value. To do that, click on the *average* function in the bottom menu.

Now go back to ADE L. Go to Outputs → Setup and click Get Expression to get the result from the calculator. Give it a name and press Add. Lastly, press the Plot button on the right side of ADE L. You should now see, in the Outputs section, that the output noise is 4.108n.

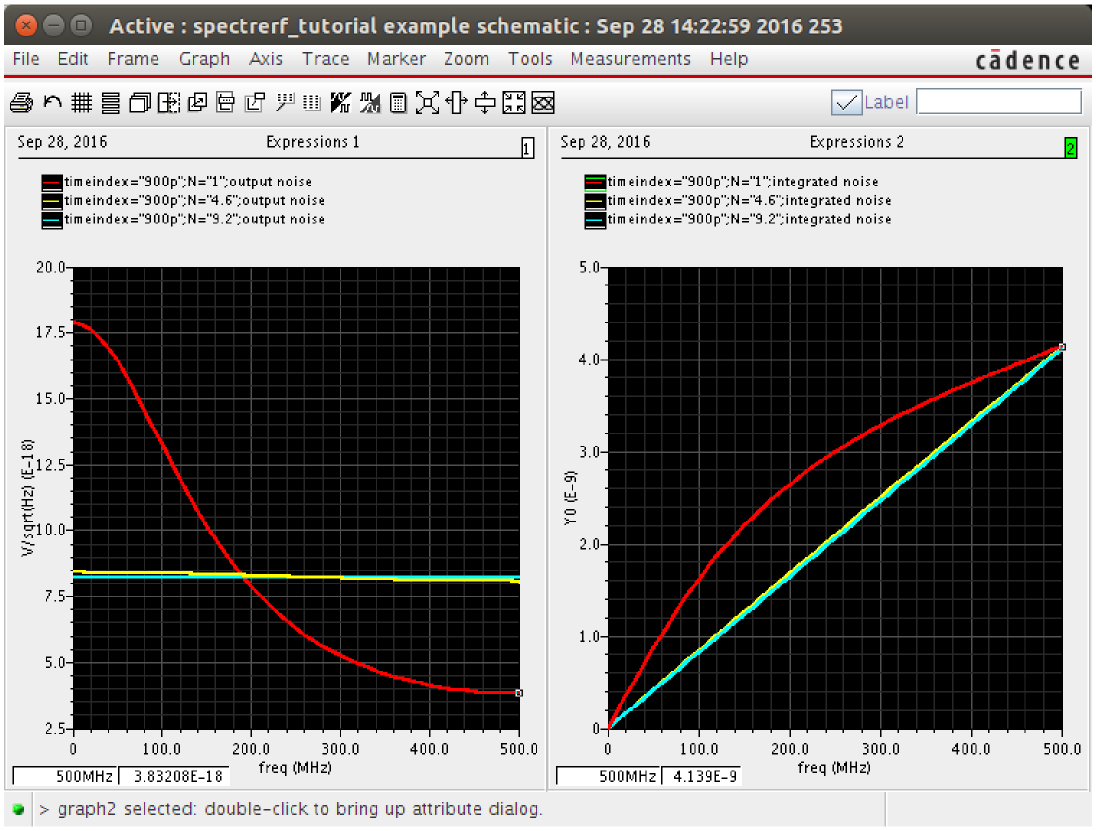
To recap, we have simulated a total integrated output noise of 4.108e-9 V^2. This is very close to the expected value of kT/C, which for a 1 pF capacitor at 300 K is 4.14e-9 V^2.

The roughly 1% error is because we have limited the sidebands to 200, and the total kT/C comes from noise folding from higher frequencies. In practice, we typically would reduce sidebands and allow as much as 5% noise error in order to reduce simulation time, with the understanding that the simulation will always (slightly) underestimate noise. For example, with 200 sidebands the *pnoise* simulation had 1% error and took 5.4 seconds to run. If we allow for 5% error and use 40 sidebands, simulation time drops to under 1.5 seconds. On larger circuits, this could save hours of simulation time.

In class we have assumed that this sampled noise is white in color and has integral kT/C. We have verified the integral, but we should also verify the color of the noise. We will look at the output noise PSD at multiple values of *N* to consider settling effects. For this, open the parametric analysis window and use the same settings as for the *pac* analysis. Hit Run. Then, open the Direct Plot Form and configure it for Analysis: *tdnoise*, Function: *Output Noise*, Select *Total Noise*, Sweep: *spectrum*, and Modifier: *Magnitude*. Click Plot.

Notice that with *N* large the noise is white, but with *N=1* the noise has some other color. This is due to correlation between successive noise samples when the circuit does not completely settle, resulting in a higher noise density at low frequencies.

At the right of the figure we are showing a running integral of the noise power. (If you want to create this plot, get the output noise spectrum from Tools → Results Browser → pnoise-timedomain.pnoise → Right click on "out" → Calculator. Then, use the *pow* function to square it and the *iinteg* function to generate a running integral.) Despite the widely different noise PSDs, all three integrals are equal to kT/C.



> graph2 selected: double-click to bring up attribute dialog.

Figure 12. Output noise PSD and running integral. Despite the different shapes, the total integrated noise is always equal to kT/C.

Bibliography (and suggested further reading)

Kundert, Ken. "Simulating Switched-Capacitor Filters with SpectreRF," The Designer's Guide Community, July 2006. Available: http://www.designers-guide.org/analysis/sc-filters.pdf

Murmann, Boris. "Thermal noise in track-and-hold circuits: analysis and simulation techniques," IEEE Solid-State Circuits Magazine, vol.4, no.2, pp. 46054, June 2012. Available: http://ieeexplore.ieee.org/document/6218338/?arnumber=6218338

SpectreRF User Guide. This is only available within Cadence Virtuoso software. From any window (CIW, Library Manager, ADE L, etc) click Help and then Cadence Documentation Library. In the Documentation Browser in the left, navigate down to Spectre RF/Virtuoso Spectre Circuit Simulator RF User Guide.

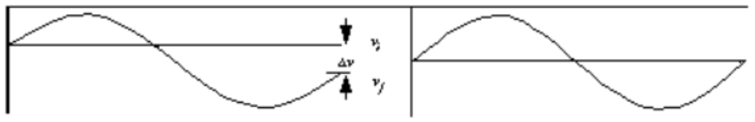
How These Tools Work

The theory behind *pss* and the small-signal analyses is described in detail in the Cadence Help documentation (see Bibliography). This section is largely summarized from that documentation and is intended to provide a basic understanding of the simulator's operation.

Theory behind the *pss* simulation

As described previously, the *pss* simulation calculates a large-signal periodic operating point from which small-signal analyses such as *pac* and *pnoise* can run, similar to how the standard *ac* and *noise* analyses run using the results from a *dc* operating point simulation. Like in a *dc* operating point simulation, for a *pss* simulation all small-signal and noise sources are turned off.

The periodic steady-state simulation is a time-domain simulation which uses the shooting method to determine the steady-state response of the system. (As in the example simulation, we will only discuss the shooting method here. The alternative harmonic balance engine operates in the frequency domain.) At its most basic level, the shooting method works by iteratively solving a transient simulation until it finds an initial condition which results in a steady-state solution. A steady-state solution is one in which the initial and final node currents and voltages are identical, within some mathematical tolerance. This is why the beat frequency is a required parameter: it sets how long each transient simulation will run. Figure 13 illustrates the concept of finding a steady-state solution.



The signal on the left starts at point v_i that does not result in periodicity.

For the signal on the right, the starting point was adjusted by the shooting method to directly result in periodic steady state.

Figure 13. Shooting method achieving a steady-state solution (from Cadence Help docs)

This analysis method makes two fundamental assumptions: periodicity and linearity. Periodicity is clear: all stimuli and responses must be periodic in the beat frequency; otherwise you would never expect the shooting method to find a solution. Edge cases exist when subharmonics are generated, in which case the beat frequency should be set to this subharmonic, or in circuits with chaotic responses, which are not periodic and should be simulated using other analyses. The linearity assumption requires that there is a near-linear relationship between initial and final points in the fundamental period. This requirement allows the shooting method to quickly converge on a steady-state solution.

The *pss* simulation has three important phases, illustrated in Figure 14 and described here:

First, if the circuit requires some time to stabilize on start-up then an (optional) initial transient analysis should be run. This provides a more accurate initial guess for the iterative solution which speeds up convergence and in some cases is necessary for the *pss* simulation to ever converge. The length of this initial transient analysis is set using the *tstab* parameter, as seen (but not set) in Figure 2. Notice that you can also save the initial transient simulation which can be useful for plotting in the time-domain if the *pss* simulation is not converging.

Second, a single period is simulated in the time domain using the node voltages and currents from the initial transient (if *tstab* was set). This is used as the initial guess for the *pss* simulation. The simulator calculates the final state as well as the sensitivity of the final state with respect to each component in the initial state. Here, the "state" refers to node voltages and currents.

Third, the initial guess is tweaked slightly using the information about the final state and sensitivities and step 2 is repeated. This tweaking involves slightly changing the initial node voltages and currents in order to get to a point where the initial node voltages and currents match the final conditions, and is why the linearity assumption is important. It typically takes about five iterations to reach the steady-state, though in the case of the simple example circuit it only took two iterations. By default the simulation will fail after about 20 iterations, but this can be changed using the *maxiters* parameter under Options → Convergence.

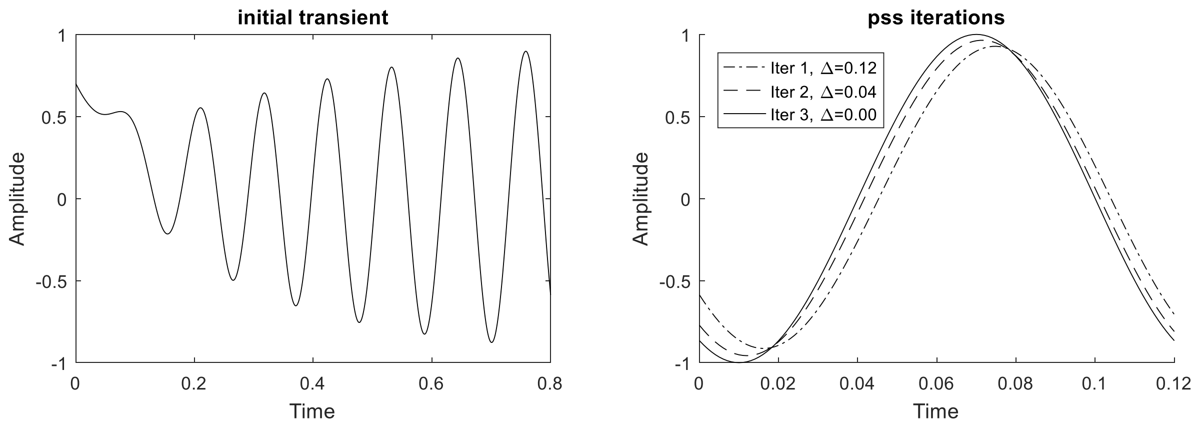


Figure 14. Illustration of pss initial settling and iterative solution

Theory behind the small-signal analyses (*pac*, *pnoise*)

During the *pss* simulation, the circuit is linearized around a periodic operating point much like in a *dc* simulation. This linearization is then used for the small-signal analyses. Two fundamental assumptions are made in these small-signal analyses which must be considered. First, since this is a small-signal analysis it is assumed that the system will have a linear response to the *pac* and *pnoise* inputs. If the inputs are large, these simulations will not agree with physical device measurements. Second, because the *pss* simulation works in the time-domain it is important that the largest time-step used in that simulation is much shorter than the small-signal input frequency. This is why we had to set the *maxacfreq* in the tutorial to avoid numerical errors in the *pnoise* simulation.

The key difference between *pac* and *ac* simulation is that *ac* computes a single transfer function from input to output. On the other hand, *pac* calculates a set of transfer functions, from an infinite number of input harmonics to an infinite number of output harmonics. In the simulator, these harmonics are referred to as sidebands where the input frequency is the 0th sideband. In the simple example in this document, we were only interested in the 0th sideband input to the 0th sideband output which is why we set it to 0 sidebands (Figure 4). In a down-conversion mixer, for example, you are likely interested in frequency translation effects such as from RF at the input (sideband 0) to DC at the output (sideband -1).

Similarly, the *pnoise* simulation calculates frequency-translational effects which are ignored in a standard *noise* analysis. This includes noise folding, which is critical to proper noise analysis in a track-and-hold circuit. The *pnoise* simulation also accounts for noise source that are bias-dependent, since in a *pss* simulation the bias conditions are periodically varying. Using the frequency-translational relationships similar to in the *pac* analysis, the *pnoise* analysis computes the sum of all noise components at each output frequency as translated from all input sidebands. By default the output is the time-averaged noise at the output at each frequency; in our example, we sampled the noise at a specific point in time.

Bibliography (and suggested further reading)

The Cadence Help manuals on SpectreRF provide a more detailed explanation of how these tools work. This must be accessed from within Cadence. From any window (CIW, Library Manager, ADE L, etc) click Help and then Cadence Documentation Library. In the Documentation Browser in the left, navigate down to Spectre RF/Virtuoso Spectre Circuit Simulator RF Analysis Theory.