intusoft

(310) 833-0710

Personal Computer Circuit Design Tools

N E W S L E T T E R

An update package is being prepared for release later this year. Included will be changes and additions that you have requested. Please keep the comments coming so that we can make the programs do what you need. This issue of the newsletter is devoted almost entirely to modeling. Our models rely on the PRE_SPICE parameter evaluation capability.

In This Issue

Run Time Parameter Evaluation:

When you see expressions in curly braces, they must be evaluated prior to running SPICE. If you have our PRE_SPICE program they will be evaluated automatically by turning on the parameter evaluation switch in the ICAPS circuit simulation menu. If you don't have PRE_SPICE, then you must evaluate the expressions by hand. The models shown here can be added to your existing libraries.

Get a Jump on the Next Update:

The models described in this newsletter should be placed in the SYS.LIB library. The model names and symbols used here will be in the next SPICE_NET update. You can get a jump on the updates by entering the information from this newsletter now.

New Products:

Watch for our new SPICE manual and IS_SPICE update. We expect to be shipping both in June. We are out of many old newsletters; the new book will have the applications and Spice tips from the old newsletters. Look at pp 2-8 of this newsletter for a preview.

Phase Locked Loop Models

Phase locked loops, PLL's, are used as filters that automatically track a carrier signal and filter out unwanted noise. Their performance in the presence of noise is degraded. The main parameter affected by noise is the lock time. Gardner [1] shows the results for a PLL with compensation of the form A + B/S. This configuration is among the simplest, yet it involves the solution of a second order nonlinear differential equation. Adding component offsets or higher order compensation changes the results. The difficulty in producing an analysis requires either simulation or breadboard evaluation.

Using IS_SPICE for PLL simulations requires the addition of a voltage controlled oscillator, VCO, and a random noise source to the PRE_SPICE 2.0 libraries. The PLL phase detector converts two input signals, that may be of different frequency, into an error voltage proportional to phase. This nonlinear operation makes analysis a difficult task.

The most interesting nonlinearities occur in communication applications that use multipliers or clipped signals with exclusive or phase detectors. Both detectors can be simulated using the existing library elements.

Making a VCO:

A VCO is fairly straight forward and is shown in figure 1. The VCO uses two integrators connected with negative feedback and no damping to make the oscillator. It may be necessary to specify the optional parameter, TMAX, in the .TRAN statement in order to prevent the oscillator from decaying.

Oscillation frequency is proportional to the square of loop gain, while amplitude depends on the initial conditions assigned to the capacitors. Each integrator is formed using a voltage controlled current source, VCCS. With the poly(n) option, the VCCS's are made into multipliers, using the extra controlling inputs for frequency control. The sum of the square of capacitor voltage will be constant. The capacitors initial voltages, therefore, control amplitude.

The integrating capacitors must be initialized with the IC=...

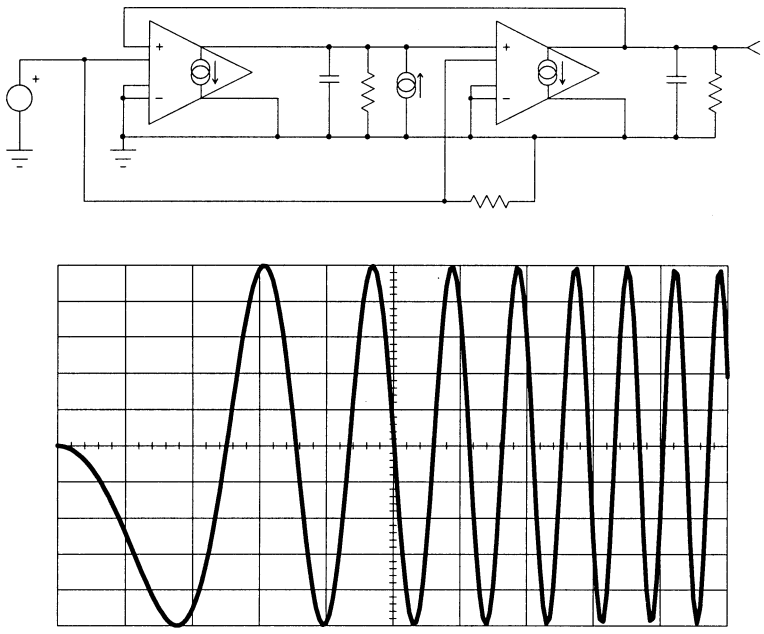


Figure 1, Basic VCO driven by a ramp of control voltage

syntax appended to the capacitor definition. Pulse initialization only works when the VCO control voltage is zero.

Making the VCO into a general element requires that key parameters be evaluated at run time. Parameters in table 1 that must be evaluated are VPK, the peak voltage and FREQ, the output frequency per volt of input control. Figure 1 shows the VCO output for a 0 to 10 volt control voltage ramp.

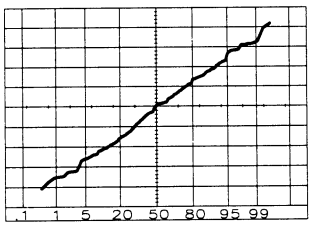
Table 1, Extended Spice Syntax Listing for a VCO

```
.SUBCKT VCO  3  2
RE1 2 0 1K
E1 2 0 22 0 1
G1 1 0 POLY(2)  22  0  3
+ 0  0  0  0  0  -1
G2 22 0 POLY(2)  1  0  3  0
+ 0  0  0  0  1
R2 1 0 1E9
I1 0 1 PULSE {1E-9*VPK} 0
C1 22 0 {.15915/FREQ} IC=0
R1 22 0 1E9
R3 0 3 1MEG
C2 1 0 {.15915/FREQ} IC={VPK}
.ENDS
```

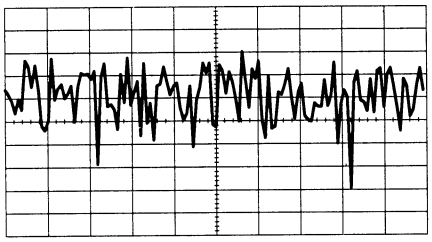
Building a Random Noise Generator:

Random noise can be generated algorithmically or by using the PWL function to enter tabular noise data. . The minimum circuit realization for algorithmic noise generation uses far more computational resources than a PWL table. For this reason, the tabular function was selected as the best choice.

Random numbers are formed by taking the sum of n uniformly distributed random samples. According to the central limit theorem of probability, the resulting series of numbers tends to have a Guassian Distribution as n approaches infinity. The resulting RMS magnitude is the peak magnitude of each sample times the square root of n/12. The choice of n=12 samples eliminates the need for a square root computation, making a very efficient computer algorithm. It can be seen by inspection that the distribution function is truncated, the largest possible number is 12 Sigma. The resulting distribution function will also have non-uniform frequency domain characteristics, namely the higher frequency components will be attenuated. Figure 2 shows the probability distribution of the time sequence and the magnitude as a function of frequency for numbers generated with this method.



Cummulative Probability. Straight line represents a Gaussian Distribution.



Noise, 5dB/Div vs Frequency shows slight loss at high frequency

Figure 2, Gaussian Noise Generator Test Results, 256 points

Our next PRE_SPICE update will have several random generator tables. If you need something now, you can use MONTE.EXE to generate random numbers and Intu_Scope to make the PWL table. We parameterized the noise generator using RMS amplitude (MAG) and simulation time span (TIM) so that the noise genrator can be used for many different applications.

Simulation Results:

Finally, simulations were run with and without noise with results shown in figure 3. When noise was included, many extra iterations were required, extending the run time and memory utilization considerably. This simulation stretches the capability of the PC, however, improvements on the horizon in both software and hardware will make IS_SPICE a viable tool for studying PLL behavior in the presence of noise. Without noise, the time domain solutions run efficiently with current generation hardware and software; they provide basic design data and sensitivity to real world component imperfections.

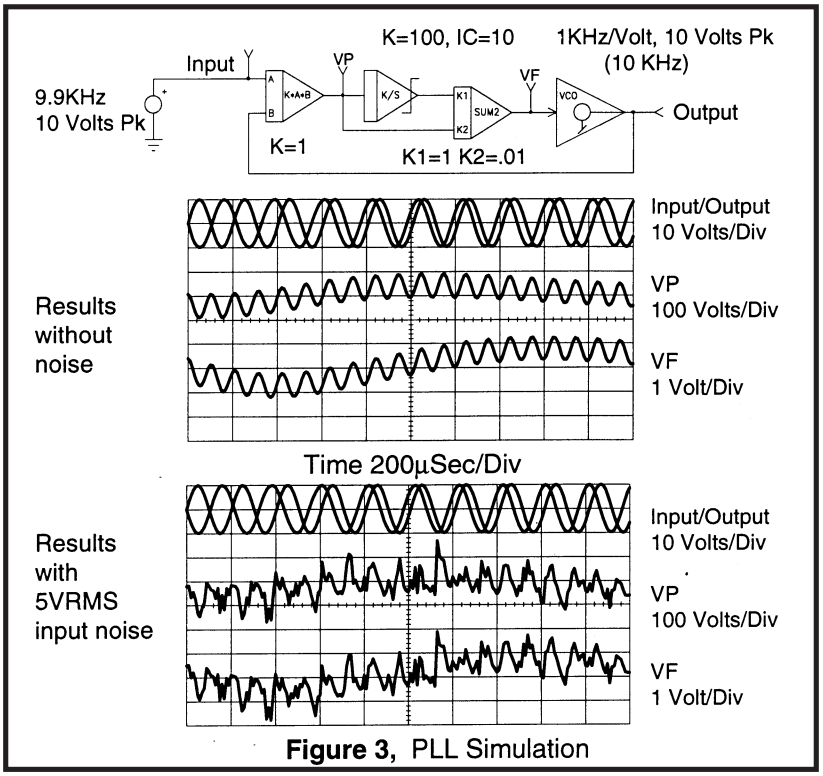


Figure 3, PLL Simulation

PHASE LOCKED LOOP LISTING

```
.OPTIONS ITL5=20000 ACCT LVLTIM=1
*SPICE_NET
.TRAN 5U 1M 0 5U UIC
*INCLUDE SYS.LIB
*INCLUDE NOISE.LIB
.PRINT TRAN  V(4)  V(6)  V(2)  V(5)
.PRINT TRAN  V(1)
X1 5 4 6 MUL  {K=1 }
X3 3 6 2 SUM2  {K1=1 K2=.01 }
X4 2 4 VCO  {VPK=10 FREQ=1K }
V1 5 1 SIN 0 10 9.9K
X7 1 NOISE  {MAG=5 TIM=1M }
X5 6 3 NTGR8  {K=100 NLIM=0 PLIM=15 INIT=10 }
.END
```

Using SPICE Polynomials

Background:

Polynomial nonlinearities give SPICE subcircuits the capability of representing devices such as vacuum tubes that are not built-in to the basic code. Capacitors, inductors and controlled sources allow polynomial nonlinearities of up to the 20th order. In addition, the controlled sources allow multiple inputs to be included. The syntax is complex, however, it reduces to a simpler form for specific cases.

Syntax Review:

Syntax for polynomial functions may appear awkward. However, it was developed in a manner that limits the number of entries. When several sources are involved, the number of sources, N, must be specified in using the POLY(N) keyword. For each exponent, all combinations of the controlling sources are specified before coefficients for the next exponent are entered. The polynomial is expanded for a given order, m, as follows:

Let m be the order of the nonlinearity
and n the number of controlling sources
with P1, P2, ... being the values of coefficients
and f1, f2, ... fn the source values

Then

$$\text{Output} = \sum_{J=1} [P_J * f1^{K1} * f2^{K2} * f3^{K3} * ... * fn^{KN}]$$

where: K1, K2, ... , KN are selected by algorithm for each coefficient, Pj

Table 2 shows how the first 20 coefficients control the exponent values. Shown next are examples for a 2 input summer followed by a multiplier.

2 Input Summer

```
.SUBCKT SUM2 1 2 3
* INPUT NODES ARE 1, 2; OUTPUT IS 3
E1 3 0 POLY(2) 1 0 2 0  0   1   1
                        P1  P2  P3
*
.ENDS
```

A 2 Input Multiplier:

```
.SUBCKT MUL2 1 2 3
* INPUT NODES ARE 1, 2; OUTPUT IS 3
E1 3 0 POLY(2) 1 0 2 0  0  0  0  0  1
                        P1 P2 P3 P4 P5
*
.ENDS
```

Table 2, Polynomial Coefficients

POLY	1	2		3			4			
Coeff.	k1	k1	k2	k1	k2	k3	k1	k2	k3	k4
P1	0	0	0	0	0	0	0	0	0	0
P2	1	1	0	1	0	0	1	0	0	0
P3	2	0	1	0	1	0	0	1	0	0
P4	3	2	0	0	0	1	0	0	1	0
P5	4	1	1	2	0	0	0	0	0	1
P6	5	0	2	1	1	0	2	0	0	0
P7	6	3	0	1	0	1	1	1	0	0
P8	7	2	1	0	2	0	1	0	1	0
P9	8	1	2	0	1	1	1	0	0	1
P10	9	0	3	0	0	2	0	2	0	0
P11	10	4	0	3	0	0	0	1	1	0
P12	11	3	1	2	1	0	0	1	0	1
P13	12	2	2	2	0	1	0	0	2	0
P14	13	1	3	1	2	0	0	0	1	1
P15	14	0	4	1	1	1	0	0	0	2
P16	15	5	0	1	0	2	3	0	0	0
P17	16	4	1	0	3	0	2	1	0	0
P18	17	3	2	0	2	1	2	0	1	0
P19	18	2	3	0	1	2	2	0	0	1
P20	19	1	4	0	0	3	1	2	0	0

Key to shaded areas

Coefficients used for summation

Coefficients used for multiplication

Division:

A divider is made as shown in Figure 4. The divider works by using the multiplier, G2, in a feedback system to solve the following set of equations.

VOUT = E1∗VX
V1∗G1 ≅ V2∗G2∗VX
if R2 approaches infinity
and E1=G1=G2=1,
Then
VOUT = V1/V2

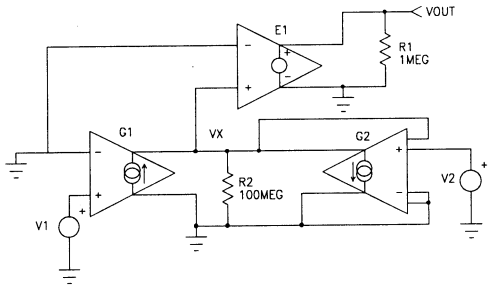


Figure 4, A Divider

The following listing produces a divider library element. If the divisor goes to zero, the simulation will abort. Unlike many real circuit implementations, the divider will work for all real inputs except when the divisor is in the neighborhood of zero.

```
.SUBCKT DIVIDE 1 2 4
* V4 = V1 / V2
R1 1 0 1MEG
R2 2 0 1MEG
R4 4 0 1MEG
G1 0 3 1 0 1
G2 3 0 POLY(2) 2 0 3 0 0 0 0 0 1
R3 3 0 100MEG
E1 4 0 3 0 1
.ENDS
```

Special Report

Using Intusoft Products with Microsoft® Windows/386®

The full line of Intusoft PC design tools have been run using the new Windows 386 operating environment. Comparitive results for an IS_SPICE simulation using the multitasking operating system are shown.

The INTU_SCOPE post processor and SPICE_NET schematic capture programs were run in the FULL SCREEN mode where performance degradation as compared to operation in the DOS environment was only about 10% - 20%. When run in a FOREGROUND WINDOW mode, SPICE_NET's and INTU_SCOPE's productivity were severely hampered by Windows/386. Both programs ran at about 1/2 their normal speed because of the excessive amount of time needed to update the graphic window display.

IS_SPICE, on the other hand, ran extremely well in both the FULL SCREEN mode and the FOREGROUND WINDOW mode. In the FOREGROUND WINDOW mode only a 30% - 40% speed penalty was revealed for short simulations and a 20% - 30% penalty for longer simulations. In the FULL SCREEN mode almost no simulation speed degradation was observed.

IS_SPICE ran from 6 to 15 times slower than normal when invoked as a BACKGROUND process. More importantly there was little or no apparent performance degradation induced in the foreground application because windows/386 gives the foreground application a much higher priority than the background application. This fact indicates that WINDOWS 386 could be a productive tool if you run your IS_SPICE simulations in the background mode and have some other windows oriented work to perform, provided you're not in a hurry for your simulation results.

COMPAQ 386 16 Meg Hz 80387 Coprocessor
Sample.Cir - 15 Nodes 14 Elements, AC & Transient Analysis

MODE	FULL SCREEN	IN A WINDOW	DOS
FOREGROUND BACKGROUND	0:27.2 N/A	0:29.2 2:01* 5:12**	0:21.0

* No Foreground activity.
** Normal Foreground activity using Aldus® Pagemaker®.

Times are in Minutes:Seconds.