SPICE 2: Subcircuit Models and Testbenches

Chris Winstead

January 27, 2015

Preparing for the Exercises

In this session, we will explore more complex designs that are split into multiple files. Make a new directory for this exercise, so that your project directory tree looks like this:

```
3410/
└── spice/
    ├── lab1/
    └── lab2/
```

Obtain the Example Files

In this lab, you will download a collection of files:

- `lab_parts.md` — Contains complex model files for the 741 op amp and other components we will use this semester.
- `exercise1.sp` and `exercise2.sp` — Partially completed SPICE file describing the circuit and testbench for Lab 1 exercises.

Obtain the Example Files

Download these files and place them in these locations:

```
3410/
└── spice/
    ├── lab_parts.md
    ├── lab1/
    │   └── [files from last time]
    └── lab2/
        ├── exercise1.sp
        └── exercise2.sp
```

Study Exercise 1

Navigate to your lab2 directory and open the exercise1.sp file. This file is significantly more complex than the examples from Lab 1. We'll step through it piece-by-piece.

At the top of the file, you should notice a .include statement:

```
Lab 2, Exercise 1, ECE 3410
***************************
* By Chris Winstead
***************************

* Include the model file:
.include ../lab_parts.md
```

This tells SPICE to load all the contents of the lab_parts.md file and process them as if they were part of the current circuit description.

Inside the lab_parts File

The lab_parts.md file contains model definitions for every device that we will use this semester. Open it and scroll through its contents until you find the model for the ua741 op amp:

```
*-------------------------------------------------------------------------------
*
* To use a subcircuit, the name must begin with 'X'.  For example:
* X1 1 2 3 4 5 uA741
*
* connections:   non-inverting input
*                |  inverting input
*                |  |  positive power supply
*                |  |  |  negative power supply
*                |  |  |  |  output
*                |  |  |  |  |
.subckt uA741    1  2  3  4  5
*
  c1   11 12 8.661E-12
  c2    6  7 30.00E-12
  dc    5 53 dx
  de   54  5 dx

  ...

.ends
```

Defining Subcircuit Models

Subcircuits are like modules. They describe complex circuits that can be reused.

To define a subcircuit, the first line specifies the name and I/O ports:

```
.subckt <subckt name> <port 1> <port 2> ...

<local devices and connections go here>

.ends
```

This behavioral model was provided by the 741 manufacturer. It mimicks all of the data sheet specifications without revealing the true design.

Using Subcircuit Models

The devices placed inside the subcircuit cannot be directly accessed by the rest of your circuit. You connect to the subcircuit through its I/O ports. To use a subcircuit, it is declared as an 'X' device:

```
X<name> <port1> <port2> ... <subckt type>
```

In `exercise1.sp`, the 741 is placed using this line:

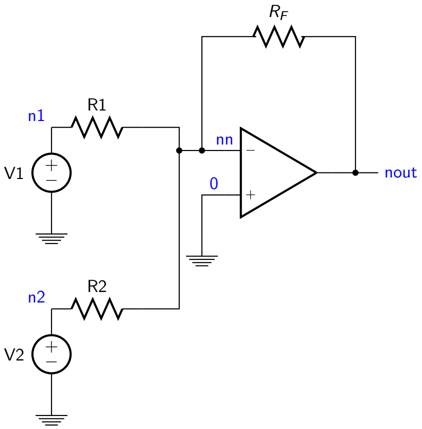
```
* Op Amp Model
X1 0 nn ndd nss nout uA741
```

This says to connect the non-inverting terminal to ground (node 0), the inverting input to `nn`, the power supplies to `ndd` and `nss`, respectively, and the output to `nout`, for a subcircuit of type `ua741`.

The Circuit in Exercise 1

The code below describes a weighted summer circuit. Study the code and schematic and verify that they match. Complete the description by entering your values for RF, R1 and R2. Then run the exercise in NGSpice, and record the text output in a file called `log.txt`.

```
* Power supplies:
VDD ndd 0 DC 15V
VSS nss 0 DC -15V

* The input voltage sources
V1 n1 0 DC 1V
V2 n2 0 DC 5V

* Resistors
R1 n1 nn    *** YOUR VALUE
R2 n2 nn    *** YOUR VALUE
RF nn nout *** YOUR VALUE

* Op Amp Model
X1 0 nn ndd nss nout uA741
```



The Testbench

The `exercise1.sp` file contains several different simulation commands used to evaluate the circuit. A collection of tests is called a testbench. It allows for automated verification of your design objectives:

```
* Control Commands:
.control
op
print all

echo "Result for Part A:"
print v(nout)

alter V2 DC 0V

dc V1 1V 2V 0.05V
plot v(nout)

echo "Results for Part B:"

* Print out specific data points:
meas dc vo1 FIND v(nout) AT=1
meas dc vo2 FIND v(nout) AT==1.25
meas dc vo3 FIND v(nout) AT==1.5
meas dc vo4 FIND v(nout) AT==1.75
meas dc vo5 FIND v(nout) AT==2.0

let gain='(vo5 - vo1)/(2.0-1.0)'
echo The measured gain is $&gain

.endc
```

Operating Point Simulation

The first testbench command is op, which requests a DC operating point simulation. This is followed by the `print all` command, which dumps every result into the terminal:

```
No. of Data Rows : 1
a$poly$e.x1.egnd#branch_1_0 = 1.099852e-03
h.x1.hlim#branch = 2.200018e-12
n1 = 1.000000e+00
n2 = 5.000000e+00
ndd = 1.500000e+01
nn = 7.529301e-05
nout = -1.09989e+01
nss = -1.50000e+01
v.x1.vb#branch = 1.054484e-08
v.x1.vc#branch = 2.499970e-11
v.x1.ve#branch = 3.002344e-12
v.x1.vlim#branch = -1.09990e-03
v.x1.vln#branch = -3.89037e-11
v.x1.vlp#branch = -4.11007e-11
v1#branch = -9.99925e-05
v2#branch = -9.99985e-04
vdd#branch = -1.66694e-03
vss#branch = 1.667142e-03
x1.6 = 1.054484e-03
...and so on...
```

Most results are voltages. The current through each voltage source is reported with a `#branch` postfix. For example, `v1#branch` reports the current passing through source `V1`.

All of the internal results for the `X1` subcircuit are also reported (you can ignore any line with `x1` in it).

Relating Simulations to Experiment

The next part of the code prints out the operating point for v_{OUT}. In this simulation, the circuit is configured to match the experiment described in Proc. 1, Part A of the Lab 2 assignment.

These lines print out a prediction of the value you are asked to measure in the experiment:

```
echo "Result for Part A:"
print v(nout)
```

When you are finished with Lab 2, you will want to compare this simulation result with your measurement.

The Alter Command

A good testbench should simulate a variety of different cases. The `alter` command is useful for changing simulation conditions:

```
alter V2 DC 0V
```

This command changes V2 to a DC source with value 0V. This matches the conditions requested in Proc. 1, Part B of the Lab 2 assignment. After this change, another DC simulation is performed to match the requested measurement:

```
dc V1 1V 2V 0.05V
```

This requests a simulation sweeping V1 from 1V up to 2V in steps of 0.05V.

More Measurements

For convenience, these lines print out the precise numerical measurements requested in Lab 2, Proc. 1, Part B:

```
* Print out specific data points:
meas dc vo1 FIND v(nout) AT=1
<...and so on...>
```

Each line requests a measurement from the DC simulation results. The general syntax is

```
* Print out specific data points:
meas <type> <result_name> FIND <signal_to_measure> AT=<sweep_value>
```

In this simulation, the type is “dc”, since we are doing a DC simulation. The result will be stored in a variable named “vo1”, and SPICE will measure the voltage at node `nout` when V1=1.

Expressions

The final lines evaluate a mathematical expression enclosed in single quotes:

```
let gain='(vo5 - vo1)/(2.0-1.0)'
echo The measured gain is $&gain
```

Here, the let syntax is used to declare a control variable named `gain`. The expression is a simple rise-over-run calculation for $G = v_{\mathrm{OUT}}/v_1$.

To reference a vector (like the one named “`gain`” in the above statements) in an `echo` command, it must be prefixed with $&.

What is the Rhyme or Reason about $&?

In SPICE, sometimes we reference variables with $, sometimes with $&, and sometimes with no symbol at all. What's going on?

SPICE has two types of information:

- Vectors — Data that can be processed with type checking. All internal SPICE calculations (voltage, current, etc) are performed on vectors. Vectors can be defined by using the `let` command.
- Variables — These are processed without type checking, and are used for higher-level control purposes like the `foreach` command. Variables can be defined by using the `set` command.

The distinction is admittedly confusing.

Print vs Echo

For the most part, use `print` to display vectors, and use `echo` to display variables. The example below defines a variable `x` and a vector `y`, and shows different syntax options for printing them to the terminal:

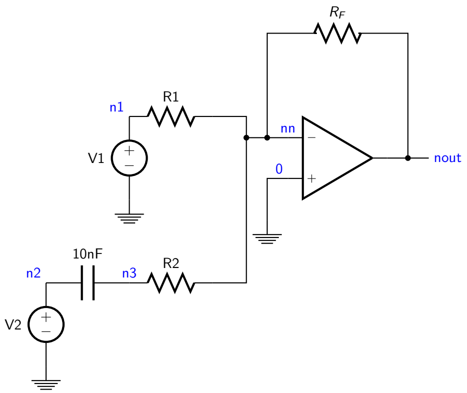
```
set x=1
let y=2*unitvec(3)
echo x=$x
echo y=$&y
print y
```

Try typing these commands into the NGSpice command terminal (access the interpretr by running `ngspice` in the terminal with no filename).

Exercise 2

After you've finished studying the first exercise, open exercise2.sp and examine its contents. You will need to modify this circuit to add the 10nF capacitor as shown. Then run the simulation and store the text output in `log.txt`.

```
Lab 2, Exercise 2, ECE 3410
***************************
* By Chris Winstead
***************************

* Include the model file:
.include ../lab_parts.md

*===== CIRCUIT DESCRIPTION ======
* Power supplies:
VDD ndd 0 DC 15V
VSS nss 0 DC -15V

* The input voltage sources
V1 n1 0 DC 1V
V2 n2 0 DC 0V AC 1 SIN(0V 0.5V 50kHz)

* Resistors
R1 n1 nn    *** YOUR VALUE
R2 n3 nn    *** YOUR VALUE
RF nn nout *** YOUR VALUE
*** ADD CAPACITOR HERE

* Op Amp Model
X1 0 nn ndd nss nout uA741
```



A More Complex Testbench

Exercise 2 contains a sophisticated testbench that introduces new techniques. At the top are some variable declarations:

```
let ofs_1=unitvec(6)
let ofs_2=unitvec(6)
let ofs_out=unitvec(6)
let idx=0
```

These lines create some vectors of length 6. We will use these vectors to store measurement results.

Looping Over Cases

Next we perform a batch of simulations using a nested `foreach` statement:

```
*----- START OF LOOP ----*
foreach vo1 1 2
foreach vo2 0.5 1.0 1.5

<...simulation commands here...>

end
end
*----- END OF LOOP ----*
```

This loop assigns variables `vo1` and `vo2` across six distinct cases. This will perform several simulations that correspond to the experiment in Proc. 2, Part A of Lab 2.

Altering Multiple Parameters

At the start of each `foreach` loop, we alter V1 and V2 and perform a new transient simulation:

```
* Alter V1 and V2:
alter V1 DC $vo1
alter @V2[sin] = [ $vo2 0.5 50k ]
* ^^^Quirk: make sure to put spaces around
* the brackets in the alter statement

* Do a transient simulation
tran 1u 100u
plot v(n2) v(nout)
```

Here the V1 and V2 sources are altered so that their DC values correspond to the `vo1` and `vo2` variables, respectively. In this example, the V2 source has been altered using an extended syntax that allows modifying multiple parameters in the SIN source.

Collecting Measurements

Next we will measure the DC offset of v_{OUT}. The results are stored in the vectors declared earlier. The vector position is stored in an index variable called `idx`, which is incremented at the end of the loop:

```
* Measure the output offset:
meas tran ofs1 AVG v(nout)

* Record results from this simulation:
let ofs_1[idx]=$vo1
let ofs_2[idx]=$vo2
let ofs_out[idx]=ofs1

* Increment the loop index:
let idx=idx+1
```

Printing Vectors

At the end of the control block, the saved vectors are printed with a simple statement:

```
* Print out the reults:
print ofs_1 ofs_2 ofs_out
```

This will output a table of results, like this:

```
--------------------------------------------------------------------------------
Index   ofs_1           ofs_2           ofs_out
--------------------------------------------------------------------------------
0       1.000000e+00    5.000000e-01    -9.72123e-01
1       1.000000e+00    1.000000e+00    -9.72123e-01
2       1.000000e+00    1.500000e+00    -9.72123e-01
3       2.000000e+00    5.000000e-01    -1.99881e+00
4       2.000000e+00    1.000000e+00    -1.99881e+00
5       2.000000e+00    1.500000e+00    -1.99881e+00
```

Save these and other measurement outputs in your `log.txt` file. This is another set of results that you will want to compare with your experimental measurements when you conduct Lab 2.

AC Simulation

The last part of this testbench performs an AC simulation that corresponds to Proc. 2, Part B.

```
*----- AC SIMULATION -----*
ac dec 100 1 1e6

plot vdb(nout)-vdb(n2)

meas ac Av0  FIND vdb(nout) AT=1k
let A3dB=Av0-3
print A3dB

meas ac fhigh WHEN vdb(nout)=$&A3dB FALL=LAST
meas ac flow  WHEN vdb(nout)=$&A3dB RISE=1
meas ac ft    WHEN vdb(nout)=0
```

These measurements locate both the 3dB cutoff frequencies (low and high), and the unity gain frequency. The `FALL=LAST` directive requests a measurement where `vdb(nout)` is falling. Similarly the `RISE=1` directive requests measurement when `vdb(nout)` is rising.

Exercise 3

Now create a new file called `exercise3.sp`. Perform the following tasks:

- Copy the circuit description from `exercise2.sp` and modify it so that it implements the circuit in Fig. 3 of the Lab 2 assignment.
- Create a testbench to perform an AC simulation, sweeping from 100Hz up to 10MHz. Measure the lower and upper 3dB frequencies and the unity-gain frequency.
- Place the AC simulation within a `foreach` loop, in which RF is altered for three different values:
  - Your original value, RF
  - Double your original value, 2RF
  - Four times the value, 4RF.

Exercise 3

- Plot the magnitude response (in dB) for each case. Note what happens to the mid-band gain, the upper 3dB cutoff frequency, and the unity-gain frequency in each case. In what ways do the simulations differ, and what features do they have in common? Write your answers in your `log.txt` file.
- You should use data from `log.txt` to assist your experimental measurements, and you should refer to these results in your lab report.

What to Turn In

Make a zip file containing `exercise3.sp` and `log.txt`. Have the TA verify your prelab solutions for R1, R2 and RF.