SPICE 2: Subcircuits and Op Amps

ECE 3410, Utah State University

Subcircuits in SPICE

Complex circuits use **hierarchy** to organize the netlist into modules or subcircuits. This is useful both for design clarity and for repeating multiple copies of the same circuit.

To declare a subcircuit, SPICE uses this syntax:

```
.subckt <subckt type name> <list of I/O ports>
  * devices, nodes, and connections with local scope
.ends
```

All declared devices and connections have local scope except for the ground node 0. In NGSpice, the node `gnd` also has global scope and can be referenced anywhere.

Instantiating a Subcircuit

After declaring a subcircuit, it can be placed in a netlist by using the `X` device type:

```
X<name> <port list> <subcircuit type name>
```

Exercise 1: voltage divider subcircuit

This simple example implements a resistive voltage divider:

```
.subckt voltageDivider a b
  R1 a b 1k
  R2 b 0 2k
.ends
```

Notice that within the subcircuit the ports `a` and `b` are treated like node names. This circuit should work as a voltage divider with `v(b)=v(a)*R1/(R1+R2) = v(a)/3`.

Save these lines in a file called `netlists/voltageDivider.sp`

Exercise 1: instantiating the voltage divider

Now make a new file called `netlists/circuit1.sp` with these lines:

```
* circuit1: instantiate and simulate a voltage divider

.include netlists/voltageDivider.sp

V1 n1 0 DC 1V
X1 n1 n2 voltageDivider

.control
  op
  print all
.endc
.end
```

Explanation: this circuit connects node `n1` to the voltage divider's port `a`, and node `n2` is connected to the voltage divider's port `b`. At `n1` the voltage is 1V, so at `n2` we expect to see (1/3)V.

Exercise 1: run the simulation

Run the simulation using this command:

```
ngspice netlists/circuit1.sp | tee -a log.txt
```

Your output should include these lines:

```
No. of Data Rows : 1
n1 = 1.000000e+00
n2 = 6.666667e-01
v1#branch = -3.33333e-04
```

The lines report the voltages at nodes `n1` and `n2`, and lastly the current sourced through `v1`.

Subcircuits with Parameters

As with other hardware description languages, SPICE allows parametric subcircuits. For example, suppose we want to define a **voltage divider with adjustable resistor values.** This would be much more useful than a fixed voltage divider.

Parameters and their default settings are defined as follows:

```
.subckt voltageDivider a b PARAMS: r1=1k r2=2k
  R1 a b {r1}
  R2 b 0 {r2}
.ends
```

The parameters can be individually defined for each instance like this:

```
X1 n1 n2 voltageDivider PARAMS: r1=2k r2=4k
```

If no parameters are defined on the instance line, then the default values are used.

Exercise 2

- Add the parameters `r1` and `r2` to the `voltageDivider.sp` subcircuit.
- Simulate `circuit1.sp` to verify that you get the same result.
- Copy `circuit1.sp` to file `circuit2.sp` and change the parameters so that R1=2k and R2=8k.
- Simulate `circuit2.sp` and verify that you see the expected results:

```
No. of Data Rows : 1
n1 = 1.000000e+00
n2 = 8.000000e-01
v1#branch = -1.00000e-04
```

Using a Vendor Op Amp Model

Electronic device manufacturers usually provide SPICE models for their products. For complex devices like an op amp, the models are usually given as subcircuits.

An example model for the **ua741 op amp** is provided in `models/ua741.md` (here the file type `md` means "model" and not "markdown"). **Open the file and look it over.** Pay special attention to the header:

```
*--------------------------------------------------------------------------------
*
* 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
*
```

An Inverting Summer Schematic

An inverting weighted summer schematic is shown below, followed by its SPICE netlist. (You may need to scroll down to see the whole netlist). In the next few slides, we will explain each part of the netlist.

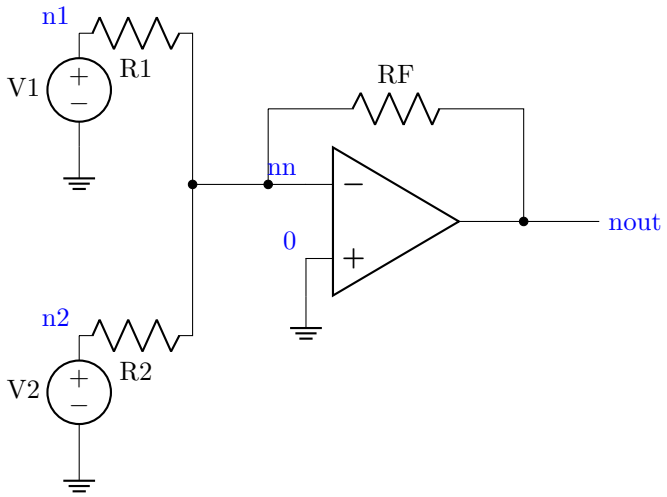

Figure 1: Schematic. Node names are indicated in blue.

```
* Inverting Weighted Summer netlist

* Use parameters to set resistor values:
* (change these to match your experiment)
.param r1=1k
.param r2=1k
.param rf=1k

* Import the op amp subcircuit:
.include models/ua741.md

* Op amps need power supplies:
VDD ndd 0 DC 15V
VSS nss 0 DC -15V

* Next, the input voltages:
V1 n1 0 DC 1V
V2 n2 0 DC 5V

* Then the resistor network:
R1 n1 nn {r1}
R2 n2 nn {r2}
```

```
RF nout nn {rf}

* Finally, place the op amp:
*  non-inverting input
*    |  inverting input
*    |  |  positive power supply
*    |  |  |  negative power supply
*    |  |  |  |    output
*    |  |  |  |    |      model type
*    |  |  |  |    |      |
X1   0 nn ndd nss nout    ua741
```

SPICE `.param` Statement

In addition to subcircuit parameters, SPICE allows top-level netlist parameters using the `.param` statement:

```
* Use parameters to set resistor values:
* (change these to match your experiment)
.param r1=1k
.param r2=1k
.param rf=1k
```

Power Supplies

When we import the op amp subcircuit model, we also need to set power supplies, since they are not built-into the model.

```
* Import the op amp subcircuit:
.include models/ua741.md

* Op amps need power supplies:
VDD ndd 0 DC 15V
VSS nss 0 DC -15V

* Next, the input voltages:
V1 n1 0 DC 1V
V2 n2 0 DC 5V
```

Using Netlist Parameters

When the resistors are placed, their values are defined by the `.param` statements. We reference those parameters as follows:

```
* Then the resistor network:
R1 n1 nn {r1}
R2 n2 nn {r2}
RF nout nn {rf}
```

Placing the Op Amp

The op amp itself is placed using the subcircuit syntax:

```
* Finally, place the op amp:
*  non-inverting input
*    |  inverting input
*    |  |  positive power supply
*    |  |  |  negative power supply
*    |  |  |  |    output
*    |  |  |  |    |      model type
*    |  |  |  |    |      |
X1   0 nn ndd nss nout    ua741
```

Exercise 3: Save the Weighted Summer

Make a file named `netlists/summer.sp` and enter the netlist explained in the preceding slides.

Exercise 4: Weighted Summer Testbench

Make a file named `tests/summer_op.sp` and perform an operating point test:

```
* Weighted Summer operating point test

.include netlists/summer.sp

.control
  op
  print n1 n2 nout
.endc
```

Ve3rify that the results match these lines:

```
No. of Data Rows : 1
n1 = 1.000000e+00
n2 = 5.000000e+00
nout = -5.99976e+00
```

Exercise 5: Simulate the First Experiment

In `summer.sp`, change the `.param` statements to match the resistor values you calculated in Pre-lab Exercise 5. Then create a testbench named `tests/exercise5.sp` to perform these actions in your `.control` script, in this order:

1. Operating point simulation; print voltages at `n1`, `n1`, and `nout`.
2. Change `V2` to zero volts using the `alter` command, like this:

```
alter V2 DC 0V
```

3. DC sweep `V1` from 0 to 5 volts in steps of 1V.
4. Print the results using `print n1 n2 nout`
5. Measure the gain using the `deriv` function. Since we are sweeping `V1`, the function `deriv(v(nout))` will calculate the derivative of `Vout` with respect to `V1`:

```
print deriv(v(nout))
```

This derivative is designed to be -1V/V, so the calculations should be close to that.

6. Generate plots of `v(nout)` and the error, `{v(nout) - (-v(n1)-2*v(n2))}`. Save the plots as `plots/dc_output.svg` and `plots/dc_error.svg`.

Relating Simulation to Experiment

Note that these are the same procedures you will be performing in the physical experiment, Procedures 1(A), 1(B), and 1(C).

When you conduct the physical experiment, the simulation should give you an idea of whether your experiment is setup properly. The physical results should roughly correspond to what you saw in the simulation.

In your lab report, you should compare simulation and measurement results. For example, you should plot measured and simulated results for Vout together on the same plot, and see if they differ in any specific way. You can also compare your simulated and measured derivatives of Vout vs V1 and analyze any significant differences.

Exercise 6: Simulate the Second Experiment

- Copy the file `netlists/summer.sp` to a new file named `netlists/highpass_summer.sp`.
- In `highpass_summer.sp` change the definition of `V2` to a sinusoid, i.e. the line should look like this:

```
V2 n2 0 SIN ({VO} {VA} {freq})
```

- Declare parameters `VO=0V`, `VA=1V`, and `freq=1k`
- Create an AC testbench named `tests/highpass_summer_ac.sp`
- Create a *transient simulation* testbench named `tests/highpass_summer_tran.sp` to perform the experiment described in Procedure 2(A).
- The testbench details are discussed in the following slides.

Testbench with Parameters

Since our netlist uses parameters, it would be convenient to use them in the testbench. NGSpice `control` scripts have a different parameter scope, so they can't "see" the netlist parameters unless we explicitly pass them through like this:

```
.include netlists/highpass_summer.sp

* pass some parameters to control script:
.csparam freq=freq
.csparam VA=VA
```

Then in the control script, the parameters can be accessed in expressions. In this example we use the `freq` parameter to set the timescale for transient simulation:

```
.control
  * Calculate transient simulation settings
  * (Here the csparam can be directly used
  * in expressions with the "let" command)
  let period={1/freq};
  let tstop = 10*period;
  let tstep = 0.1*period;
```

Use Vectors to Save Data

Procedure 2(A) requires testing several different values for V1, and several different DC offsets for the sinusoid V2. To efficiently simulate all of those cases, we can place the experimental settings in vectors and use a `foreach` loop to test each case:

```
  * Initialize output data vectors:
  let gain  = 0*unitvec(6)
  let outofs= 0*unitvec(6)
```

In the above code, two vectors are declared: `gain` and `outofs`, which will store measurement results for the gain and output offset, respectively. The `unitvec(6)` function initializes a vector of length 6 containing all 1s. We multiply this vector by zero to make them all 0s.

Next we use the `compose` function to specify the values for V1 and V2:

```
* Define experimental settings as indicated in
* Procedure 2(A)(i--iv)
* negative values need to be in parentheses
compose v1dc  values 0 0 0 1 (-1) (-2)
compose v2ofs values 0 1 2 0 0  0
```

Use a `foreach` loop for multiple cases

```
* Use a FOREACH LOOP to run each case
foreach idx 0 1 2 3 4 5

   * Grab parameters from their vectors:
   set VI=v1dc[$idx]
   set VO=v2ofs[$idx]

   * Change the settings of sinusoidal source V2,
   * and DC source V1, and repeat the simulation:
   alter V1 DC $VI
   alter @V2[sin] = [ $VO $&VA $&freq ]

   * Run transient simulation
   * (here the "$&" expansion is used so the
   * settings are given as strings)
   tran $&tstep $&tstop
```

Comment about `set` and `let`

There are two kinds of variables in NGSpice:

- `set` declares and defines a **unitless numerical variable**
  - When used in expressions, must be referenced with a `$` prefix.
  - The `idx` loop variable is of this variety.
- `let` declares and defines a **vector** which may have physical units.
  - Can be referenced directly in expressions, but needs `$&` prefix to convert value to string.
  - `$&` can convert a whole vector or array into a strings.
  - `$&` cannot always be used with array indices like `[idx]`, so we need to sometimes do multiple steps of conversion.

Examples:

```
* These all work:
set x=12
set y=$x
```

```
let a=12
let b=a

set x=a
let a=$x

compose x lin=4
echo $&x

* This fails:
echo $&x[2]

* But this works:
let y=x[2]
echo $&y
```

Measuring and Saving the Results

To measure the gain, we use the `PP` (i.e. peak-to-peak) `meas` function to obtain the amplitudes at the output and input. **The gain is the ratio of amplitudes.** To measure the output offset voltage, we use the `AVG` (i.e. average) `meas` function. The results are stored in the output vectors at the current index:

```
* Measure gain:
meas tran vopp PP v(nout) from=0 to=$&tstop;
meas tran vipp PP v(n2) from=0 to=$&tstop;

* Measure output offset:
meas tran voavg AVG v(nout) from=0 to=$&tstop;

* Save the measurements in the output data vectors:
let gain[$idx]  = vopp/vipp
let outofs[$idx]= voavg
```

Plotting the Results

Since we are producing several cases, we will want to title the plots using the voltage settings for each case, and we will want to save each hardcopy to a unique filename to avoid overwriting results.

```
* Plot results:
let a=v1dc[$idx]
let b=v2ofs[$idx]
set fname=plots/proc2a_{$idx}.svg
```

```
    plot v(nout) v(n2) title 'V1=$&a,V2(offset)=$&b'
    hardcopy $fname v(nout) v(n2) title 'V1=$&a,V2(offset)=$&b'
```

Terminating the Loop and Printing Results

The `foreach` loop is concluded by an `end` statement.

The output vectors can be printed in a table using the `print` function as shown below.

```
  end * foreach

  * Print the output vectors:
  print v1dc v2ofs outofs gain

.endc

.end
```

The `control` script ends with `.endc` and the testbench ends with `.end`

Example Output Table

The table of output data should look something like this:

```
--------------------------------------------------------------------------------
Index   v1dc            v2ofs           outofs          gain
--------------------------------------------------------------------------------
0   0.000000e+00    0.000000e+00    1.463937e-02    1.993725e+00
1   0.000000e+00    1.000000e+00    1.463937e-02    1.993725e+00
2   0.000000e+00    2.000000e+00    1.463937e-02    1.993725e+00
3   1.000000e+00    0.000000e+00    -9.84697e-01    2.000365e+00
4   -1.00000e+00    0.000000e+00    1.013977e+00    1.996517e+00
5   -2.00000e+00    0.000000e+00    2.014064e+00    1.991924e+00
```

You should be able to use these results to make direct comparisons with your physical experiments.

Exercise 7: AC Testbench

- In `highpass_summer.sp` change the definition of `V2` to add an AC magnitude of 1, i.e. the line should look like this:

```
V2 n2 0 AC 1 SIN ({VO} {VA} {freq})
```

- Create an AC testbench named `tests/highpass_summer_ac.sp`

AC Testbench: Simulate and Plot

Perform an `AC` simulation with a logarithmic (i.e. `dec`) frequency sweep. Use the `mag` function to obtain the gain, and the `vdb` function to obtain the gain in decibels. Then plot the results versus the frequency and save plots.

```
ac dec 10 1k 4Meg
let gain=mag(v(nout))
let gain_dB=vdb(nout)

* Plot magnitude response:
plot gain    vs frequency
plot gain_dB vs frequency

* Save plots to SVG files:
hardcopy plots/Proc2B_dB.svg gain_dB vs frequency
hardcopy plots/Proc2B.svg    gain    vs frequency
```

AC Testbench: Measure Bandwidth

Use the `meas` command with `MAX` and `WHEN` functions to measure the peak gain (in dB), and the two -3dB cutoff frequencies. Then report the results to the console.

These measurements serve as a prediction for your physical experiment.

```
* Measure peak gain at mid-band
meas ac max_dB MAX gain_dB

let drop3dB=max_dB-3

* Find lower and upper cutoff frequencies
meas ac f_low  WHEN gain_dB=drop3dB CROSS=1
meas ac f_high WHEN gain_dB=drop3dB CROSS=2

let bw=f_high-f_low

echo "Frequency range from $&f_low to $&f_high with total bandwidth $&bw"
```

Exercise 8: Simulate Procedure 3

- Copy the `netlists/highpass_summer.sp` netlist to a new file named `netlists/two_stage_summer.sp`, then modify the new file to implement the schematic shown in Fig. 2.3 of the pre-lab.
- Copy the `tests/highpass_summer_ac.sp` file to a new testbench named `tests/two_stage_summer_ac.sp`.

- Adapt the new testbench to simulate the experiment described in Procedure 3 of the lab manual. Change the SVG plot filenames so they refer to Proc3 instead of Proc2. Record the measurement results to compare with your physical experiment.

Important Note: The low end of the AC sweep should be 1Hz, not 500Hz. The experimental procedure should be revised **both for the simulation and the physical experiment.**

Summary of Exercises

1. Subcircuit (voltage divider)
2. Parametric subcircuit (voltage divider)
3. Weighted Summer netlist
4. Weighted Summer testbench
5. Simulate Procedure 1A
6. Simulate Procedure 2A
7. Simulate Procedure 2B
8. Simulate Procedure 3

Turning in Your Work

The preferred way to turn in your work is to use `git`. From the Linux terminal:

```
git add *
git commit -a -m "Submitting SPICE 2 assignment"
git push origin master
```

Alternatively you can upload a ZIP file to Canvas containing all your assignment files.