

# Electronic Computer-Aided Design

Supplement to PHYS3360/AEP3630 Laboratory Manual

Ivan V. Bazarov  
Physics Department  
Cornell University

October 6, 2009

---

## Contents

---

<b>1</b>	<b>Introduction To Electronic Computer-Aided Design</b>	<b>3</b>
1.1	About SPICE and LTSPICE . . . . .	5
1.2	Getting Started with LTSPICE . . . . .	7
1.2.1	Example 1: High-Pass Filter . . . . .	7
1.2.2	Example 2: Simplistic Op-Amp . . . . .	15
1.2.3	Example 3: Using External Models . . . . .	21
1.2.4	Example 4: Creating Custom Components . . . . .	26
1.3	Practice Problems . . . . .	32
<b>2</b>	<b>Optional Experiments Using LTSPICE</b>	<b>35</b>
2.1	Chapter 8 Experiments . . . . .	36
2.2	Chapter 9 Experiments . . . . .	38
2.3	Chapter 10 Experiments . . . . .	42
2.4	Chapter 11 Experiment . . . . .	44
2.5	Chapter 12 Experiments . . . . .	49
	<b>Bibliography</b>	<b>56</b>

---

## Introduction To Electronic Computer-Aided Design

---

Electronic circuit simulation programs use mathematical models to replicate the behavior of an actual electronic device or a circuit. Simulating a circuit's behavior prior to building it can greatly improve its efficiency, allow to catch potentially costly “bugs”, as well as to provide additional insights into the circuit's performance by exploring various “what-if” scenarios. Such simulations are particularly important when routine circuit prototyping using a breadboard is not easily available, e.g. as in a design of Integrated Circuit devices. More generally, most new circuits under development, except for the simplest kind, can benefit from such electronic computer-aided design (ECAD).

There is a large number of circuit simulators available both commercially and for free. A list of some of the most popular ones is given in Table 1.1. Depending on which circuit type needs to be simulated, a particular code may be geared towards treatment of either analog or digital signals. However, most of the present-day ECAD programs support some form of *mixed-mode* — a mode that allows to simulate both analog and digital circuits.

Part 1 will introduce a fully-featured circuit simulator LTSPICE (also known as SWITCHERCAD) and will teach you how to simulate electronic circuits that use various analog and digital components. The simulation program allows to perform time transient analysis of signals, display waveforms of voltages and currents using a virtual scope utility, find quiescent point, create custom components for use in future circuits, and do much more. Having learned how to use LTSPICE will enable you to perform simulations for real-life electronic circuit projects in the future using either this or another ECAD program.

A word of caution is in order: no ECAD software, no matter how sophisticated or feature-packed, can serve as a substitute for intuitive understanding of circuit



Table 1.1: List of some popular ECAD programs.

Name	URL & comments
B2 SPICE	<a href="http://www.beigebag.com">http://www.beigebag.com</a> ; commercial, large library of components, free demo available
CIRCUITCREATOR	<a href="http://www.advancedmsinc.com">http://www.advancedmsinc.com</a> ; commercial, printed circuit board design, free demo available
EDS LITE	<a href="http://www.mccad.com">http://www.mccad.com</a> ; free, limitation on circuit size, includes printed circuit board design
ICAP	<a href="http://www.intusoft.com">http://www.intusoft.com</a> ; commercial, large library of components
LOGICWORKS	<a href="http://www.logicworks5.com">http://www.logicworks5.com</a> ; commercial, geared towards digital circuitry, demo available
MACSPICE	<a href="http://www.macspice.com">http://www.macspice.com</a> ; free, console interface without schematic capture
PSPICE	<a href="https://www.cadence.com">https://www.cadence.com</a> ; commercial, large software package, demo CD available
HSPICE	<a href="http://www.synopsys.com">http://www.synopsys.com</a> ; commercial, large software package
SIMETRIX/SIMPLIS	<a href="http://www.catena.uk.com">http://www.catena.uk.com</a> ; commercial, free trial version available
SUPERSPICE	<a href="http://www.anasoft.co.uk">http://www.anasoft.co.uk</a> ; commercial, free demo available
LTSPICE	<a href="http://www.linear.com/software">http://www.linear.com/software</a> ; free, includes schematic capture
TOPSPICE	<a href="http://penzar.com/topspice/topspice.htm">http://penzar.com/topspice/topspice.htm</a> ; commercial, free demo available
VISUALSPICE	<a href="http://www.islandlogix.com">http://www.islandlogix.com</a> ; commercial, free demo available
SPICE	The original simulator from U. of California at Berkeley for analog circuits; the “engine” behind many of the simulators listed above; uses command line interface; available in public domain
XSPICE	An extension to SPICE by Georgia Inst. of Technology that allows simulation of mixed-signal circuits and systems; available in public domain

behavior if a good grasp of fundamental electronics principles is missing in the first place. Nor can a reliance on such a software replace proper experimental practices in the lab. As a matter of fact, a premature embracement of a powerful simulation tool can have quite an opposite effect of “cementing” certain deficiencies in understanding of the subject matter. To avoid this unbecoming scenario, the use of the software described here is discouraged until the students find themselves well into the course, more specifically, in its second half right after the last lecture on analog circuits (Field Effect Transistors).

## 1.1 About SPICE and LTSPICE

Most of the ECAD simulation codes employ SPICE (Simulation Program with Integrated Circuit Emphasis) as their “engine” to find actual numerical solution of a mathematical model representing the circuit being simulated. SPICE is an analog electronic circuit simulator, which was developed at the Electronics Research Laboratory of the University of California at Berkeley. SPICE3 is the most current version. The first two versions were written in FORTRAN, and SPICE3 was written in C and subsequently released into the public domain. Later, a code named XSPICE, an extension to SPICE3 developed by Georgia Institute of Technology, has expanded the simulation capabilities to include modeling of the mixed-signal circuits (both analog and digital). These two codes, oftentimes substantially modified, form the core for most state-of-the-art circuit simulation programs, both free and commercial<sup>1</sup>.

The original SPICE has an entirely text-based interface. SPICE requires a text file called `netlist` that describes the circuit to be simulated. In a way, SPICE `netlist` syntax can be looked at as an interpreted language, which contains all the necessary details the program needs to know about the particular circuit. This syntax is quite powerful albeit not very intuitive for an untrained eye. As an example, Figure 1.1 shows a simple AC circuit, and the corresponding `netlist` that describes this circuit for SPICE.

Each line of the `netlist` file has a specific meaning:

```
1 * Demo of a simple AC circ.
2 v1 1 0 ac 12 sin ; v1 is an AC source of 12V amp.
3 r1 1 2 30 ; r1 is 30ohm between nodes 1 and 2
4 c1 2 0 100u ; c1 is 100uF between nodes 2 and 0
5 .ac lin 1 60 60 ; directive to perform AC analysis
6 .print ac v(2) ; print out voltage at node 2
7 .end ; anything after .end will be ignored
```

Any line with ‘\*’ as a first character is treated as a comment, as is anything following a semicolon ‘;’. The first line is always assumed to be a circuit description

---

<sup>1</sup>LTSPICE uses its own proprietary version of modified SPICE3 instead of XSPICE to enable mixed-mode simulations.

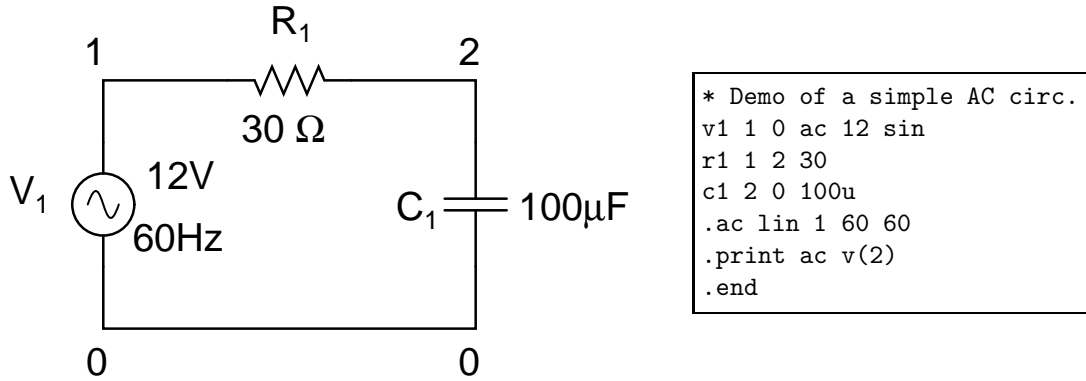


Figure 1.1: Simple AC circuit and the corresponding SPICE netlist.

(comment). Lines 2 through 4 in this example represent the actual circuit description. Node 0 is always treated as ground and must be present in any circuit. The first letter in each of these lines tells SPICE whether the element is a voltage source, a resistor, or a capacitor. The following numbers indicate what nodes a particular element is connected to and the component value. Units are implicit for each component type, and, thus, need not be supplied except for a decimal prefix as applicable. Lines 5 and 6 are examples of SPICE directives — particular instructions to perform certain tasks: `.ac lin ...` instructs SPICE to perform AC analysis of the circuit with different frequencies on a linear interval (in this case simply a single frequency value of 60 Hz), while `.print ac v(2)` instructs the program to printout AC voltage at node 2. The case of letters in a SPICE netlist can be freely changed as the syntax is not case sensitive. Running SPICE on this netlist will produce the following output:

```

--- AC Analysis ---
frequency: 60 Hz
V(2):  mag:  7.94876      phase:  -48.5171      voltage

```

LTSPICE/SWITCHERCAD is an example of a general-purpose SPICE-based simulator developed by Linear Technologies. It is a fully functional code that can be downloaded from <http://www.linear.com/designtools/software>. Some of the most salient features of this program include:

- Mixed-mode support that allows simulation of both analog and digital devices.
- Use of *schematic capture* Graphical User Interface (GUI) to create new schematics or modify already existing circuits, with subsequent automatic netlist generation.
- Waveform viewer that allows to plot voltages and currents after the simulation by clicking the mouse on the nodes and devices in the schematic.


- It includes an extensive library of Linear Technology devices. Additional devices can also be added or constructed with some knowledge of electronics and SPICE language. A large collection of device components, digital ICs in particular, is available from the user community [1].
- It is one of the most widely used circuit simulators at this time. Many professionals consider it to be superior to some commercial simulators.
- It's totally free!



The program is distributed for MS Windows operating system only. However, LTSPICE can be successfully run on Linux and Mac OS X computers with WINE package installed [2].

## 1.2 Getting Started with LTSPICE

We will start introducing basic functionality of LTSPICE by building and analyzing several simple circuits. The program's help and example files contain plenty of additional interesting information.

To get started, open the program and choose **File**→**New Schematic** from the Menu or press **Ctrl** + **N**. For easy circuit drawing, you can enable the grid by selecting **View**→**Show Grid** or toggle the grid with **Ctrl** + **G**. Next, you can place various electronic components by selecting them from the Menu **Edit**→{**Component Name**}, the Toolbar, or by pressing the appropriate 'hot' key. See Figure 1.2.

Pressing **F2** key or the corresponding Toolbar icon  brings a window that allows to choose a symbol from a long list of components, such as voltage or current sources, transistors, op-amps, logic gates, predefined Linear Technology products, or user-defined components.

Once the component is selected and placed on the screen, you can rotate and mirror it by pressing **Ctrl** + **R** and **Ctrl** + **E**, respectively (Figure 1.3). Other useful commands for manipulating component placement are moving (**F7** key or  Toolbar icon) and dragging (**F8** or ). The difference between the two is that dragging allows to change the position of the component without breaking its electrical contact (by expanding the wires).

### 1.2.1 Example 1: High-Pass Filter

Let's build and analyze our first simple circuit in LTSPICE.

**Goals:** exercise basic schematic capture ; set up generating function for voltage source; perform time transient analysis `.tran`; use waveform plot utility to view voltage and current signals; assign specific diode model; basic anatomy of automatically generated `netlists`.

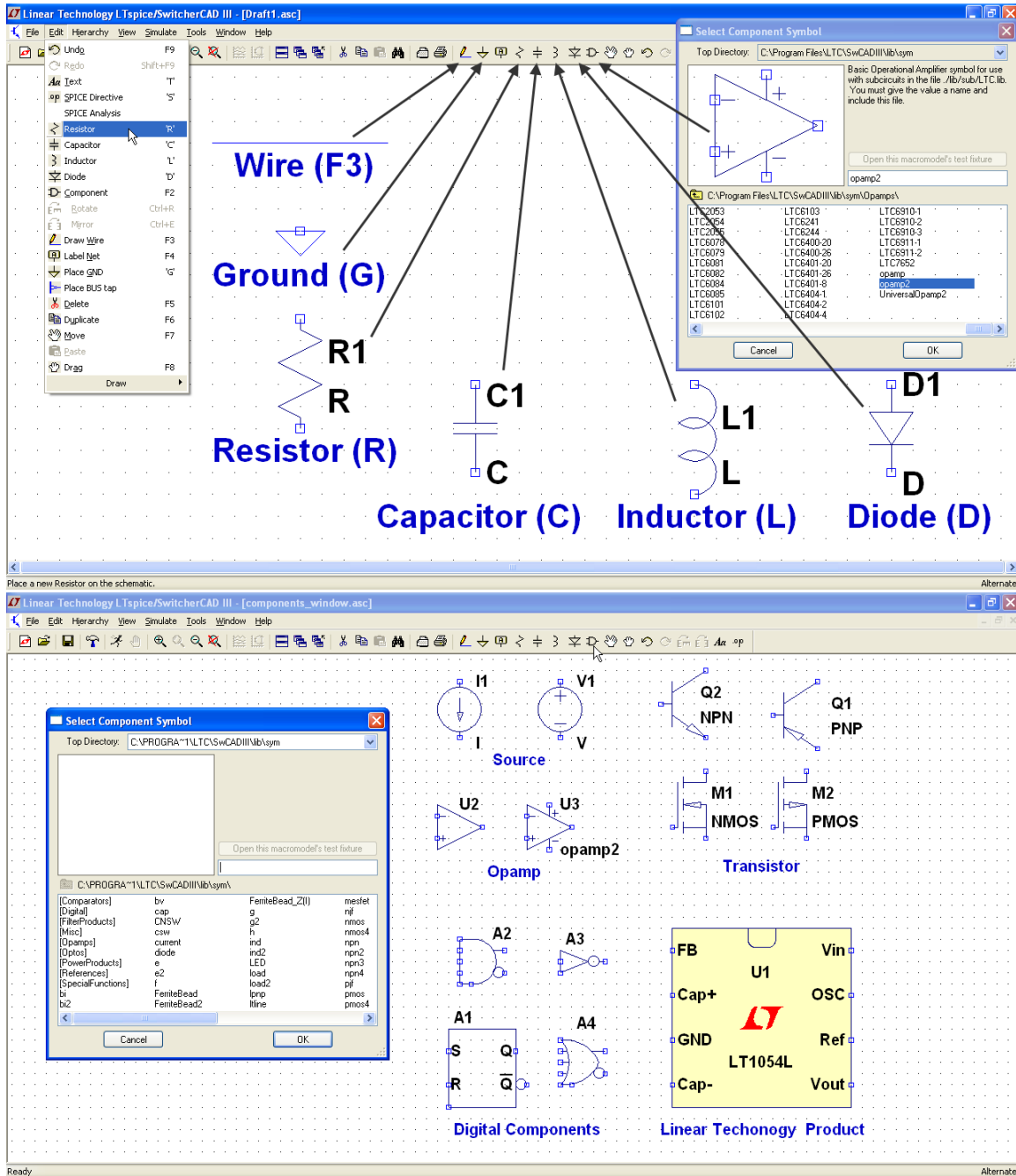


Figure 1.2: Placement of various components in LTSPICE schematic capture. Pressing **[F2]** brings up the selection window for additional components.



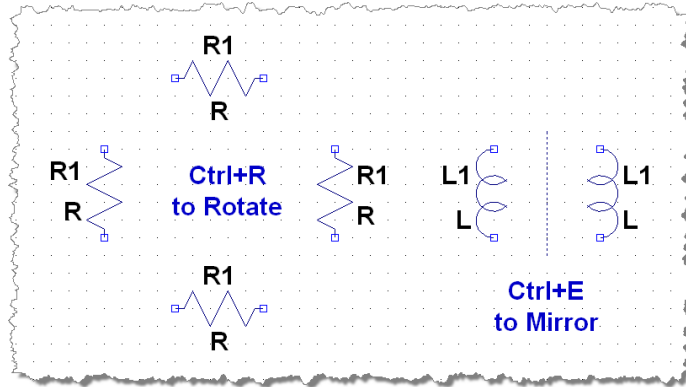



Figure 1.3: Orientation options prior to the component's placement.

First, create a circuit shown in Figure 1.4 and assign appropriate values to the capacitor and the resistor. You can find a voltage source symbol (called `voltage`) in the list of components after pressing `[F2]` key. To edit component properties, place the cursor on top of the element so that it is transformed into a hand shape  and then right-click with the mouse.

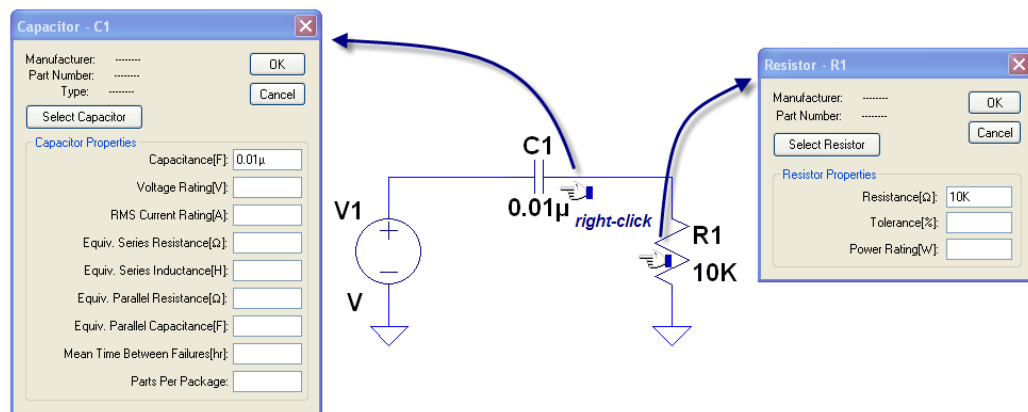



Figure 1.4: Assigning component values.

SPICE automatically assumes appropriate SI units for all of its component values. The prefix for the unit can be specified as follows:

$$\begin{array}{lll}
 T = \text{Terra} = 10^{12} & K = \text{Kilo} = 10^3 & N = \text{nano} = 10^{-9} \\
 G = \text{Giga} = 10^9 & M = \text{milli} = 10^{-3} & P = \text{pico} = 10^{-12} \\
 \text{MEG} = \text{Mega} = 10^6 & U = \text{micro} = 10^{-6} & F = \text{femto} = 10^{-15}
 \end{array}$$

For example, if you want a  $10\text{K}\Omega$  resistor, simply type `10K` for the resistance value.



Note: SPICE commands are not case sensitive. For example, `10M` typed in for a resistance value is the same as `10m`, which is  $10\text{m}\Omega$  (10 milliohm)! If what you really intended was  $10\text{M}\Omega$ , you should use `10MEG` (or `10meg`) instead. 

For now, enter 10K for resistance and 0.01u for capacitance. Note that the value of the components will appear where previously there were letters R and C. You can also right-click on these letters to open up a simpler version of the component value editor window, which too can be used to specify the component value. Letters R1, C1 and V1 are the name of the elements that SPICE uses in their reference. You can right-click and change them to whatever you want as long as these names remain unique.

Next, let's specify the voltage source. LTSPICE allows to assign various generating functions to its voltage and current sources, including DC, sinusoidal AC, exponential, triangular, square forms, etc. It even allows to load arbitrarily complex waveforms from .wav files (more on that later). For now, right-click on the voltage source. When you do so for the first time, it brings up a simple window that allows to specify a DC source voltage and its series resistance. For other options click on **Advanced** button, which will bring another window with more options. To specify a time-dependant voltage source outputting repetitive pulses of square, triangle, sawtooth, etc. shape, it is convenient to use PULSE option. After selecting this option, enter the values as shown in Figure 1.5. This will produce a square-wave with a 1kHz repetition rate with high and low voltage values of  $\pm 5$  V. Leaving **Ncycles** entry blank means the pulse will be repeating endlessly for the whole duration of the simulation. Most of the entries are self-explanatory. One precautionary remark relates to **Trise** and **Tfall** entries. Setting **Trise** = **Tfall** = 0 or leaving these entries blank will lead to a default behavior where LTSPICE will use a value of 10% of **Ton** or **Toff**, whichever is smaller, for the rise and fall times. While a convenient choice for the default behavior, it may be confusing to the user who explicitly supplied 0 to these entries expecting to get a sharp edge. Instead, for a sharp edge, one should supply a sufficiently small but nonzero value. In this example, we specify **Trise** = **Tfall** = 1n for 1ns rise and fall times. Press OK button. The specified PULSE options should now appear next to the voltage source symbol.



LTSPICE has six different types of analyses: time-transient, small signal AC, DC operating point (Q-point), DC source sweep, small signal DC transfer function, and intrinsic noise analysis. We will go over the first 3 in this tutorial using various examples.

To do time-transient analysis click **Simulate**→**Edit Simulation Cmd**. Select **Transient** tab, and enter 5m for **Stop Time**, then press **OK**. See Figure 1.6. This will produce a text line `.tran 5m`, a SPICE directive — a command with additional instructions for the simulator. In this particular case, it instructs LTSPICE to perform time-transient analysis of the circuit for 5ms. Place the command anywhere in your schematic. When you become familiar with SPICE `netlist` syntax, you can insert SPICE directives directly by pushing  icon on the Toolbar or pressing  key. Once the appropriate string is created, just place it anywhere in your schematic.

There are many SPICE directives performing various functionalities during or after the simulation. Some of them will be introduced in this tutorial. Refer to the

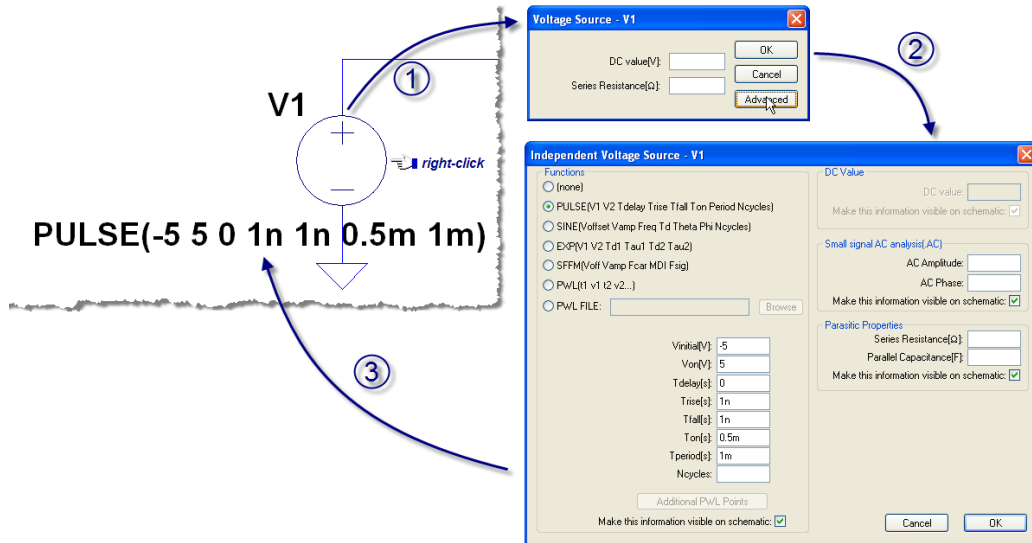


Figure 1.5: Assigning square wave to the voltage source.

program's help files for a full description of the syntax for possible directives (also known as dot commands). Alternatively, you can specify the string to be treated as comment — this can be used to disable certain commands, or to add descriptive text to your circuit. Another way to add such text comments is by pressing **T** key or **Au** icon on the Toolbar, and then placing the string in your schematic. Such strings have no effect on simulations.

SPICE directive `.tran` contains several useful entries. In particular, **Time to Start Saving Data** can be used to ignore time-transients of the circuit by specifying appropriate waiting time. This is commonly done when one is only interested in a steady-state behavior of the circuit.

Click on Run **R** icon or select **Simulate**→**Run** from the Menu bar. If the simulation has successfully finished, a new window will appear where you can display simulation results. Additionally, LTSPICE writes several files to the disk. A text file with extension `.log` contains messages pertaining to the simulation status with any warnings or errors. Another text file `.net` is the automatically generated **netlist** of the circuit. Finally, a binary file with extension `.raw` contains the actual results of the simulation such as voltages for all nodes, etc. You can choose to have LTSPICE automatically delete these temporary files after the program exit by going to the Control Panel **T** (Tools→Control Panel) and then selecting **Operation** tab.

To measure the voltage on the resistor, click at the wire above it when the cursor changes its shape to a red probe **P**. This voltage measurement is relative to the ground. The lower-left corner of the main window displays interactive messages about the position of the cursor and description of the action to be taken upon clicking. See Figure 1.7.

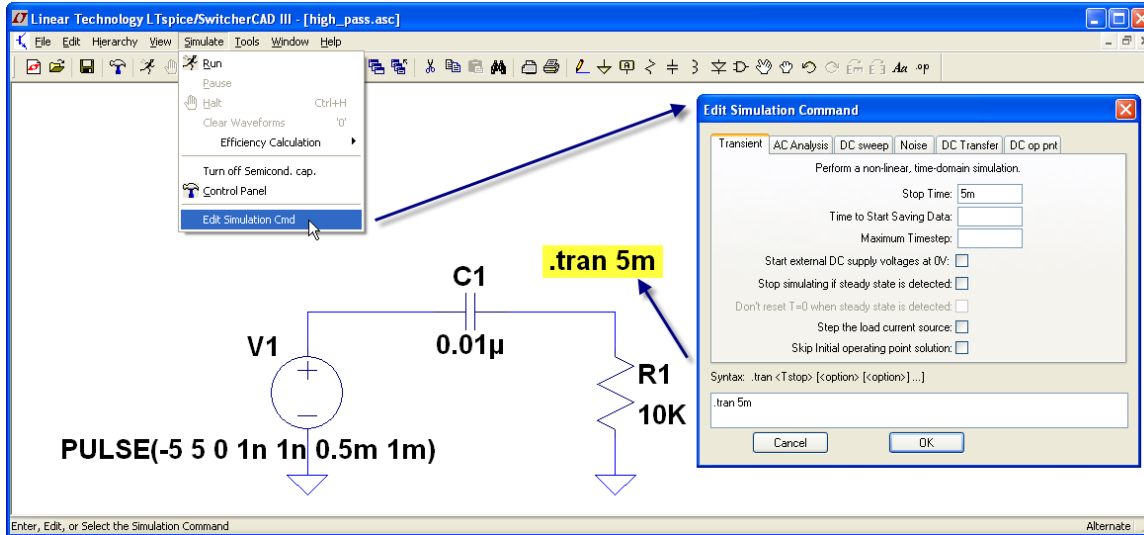











Figure 1.6: Setting up time-transient analysis.

LTSPICE waveform plotting utility is very flexible and you should spend some time familiarizing yourself with its many options. Some of the most commonly used operations are listed below.

- To measure voltage across two different nodes, hold the left mouse button pointing the cursor at the position of the first node. The probe  will “detach” and stay at that location. Drag the probe  to the other node until it turns black . Release the button for the measurement.
- To measure current flowing through a component simply place the cursor over it. The probe will change its shape into a current probe . To force for a current measurement in a location of a connecting wire, hold down  key, then click to plot the current.
- To measure power dissipated in an element, click on it while holding down  key. The probe should transform into a thermometer-like shape .
- Various zooming options are available in the plot window. Simply select the region of interest to zoom in.  +  returns the display to its default view.
- Selecting new signals will add the corresponding waveforms to the same plot pane. Selecting the same trace twice, removes all other traces from that pane.
- Add or remove additional plot panes from Plot Settings Menu bar or by right-clicking on the plot window.

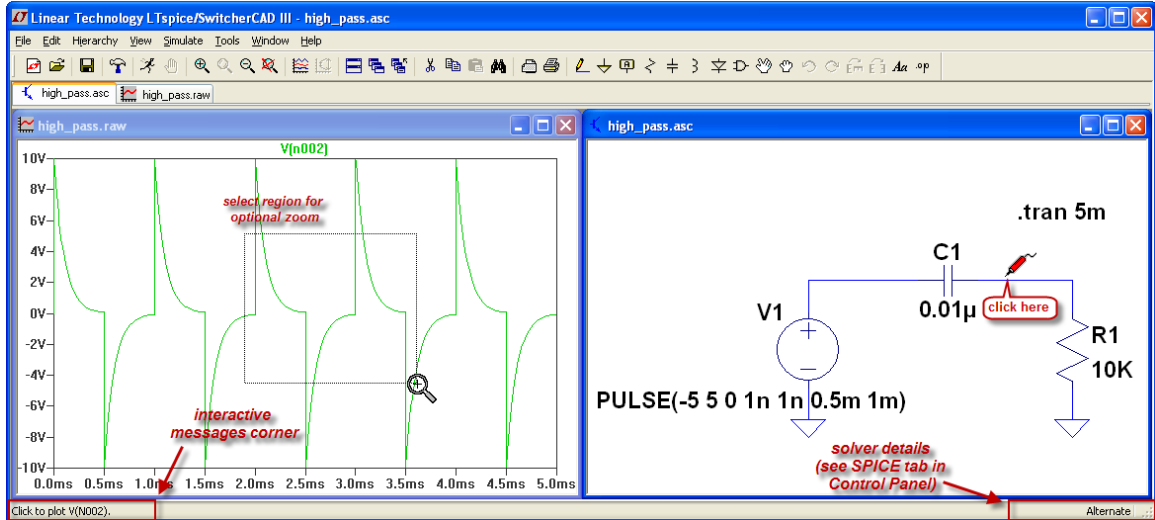


Figure 1.7: Use of plotting utility to view the simulation results.

- Edit mathematical expression being plotted, or plot one signal vs. another. For example,  $V(N001, N002) * I(C1)$  plots the voltage between nodes N001 and N002 times the current flowing through C1 (the dissipated power).  $V(N002) - V(N001)$  plots the voltage difference between the two nodes, as does  $V(N002, N001)$ . Refer to *Waveform Arithmetic* in LTSPICE help files for full description. The low-left corner of the main window displays the node number when the cursor is placed over a particular area in the circuit.
- Change axes to display data either on linear or logarithmic scale.
- Use up to two cursors on each trace to read out abscissa and ordinate values at locations of interest in the plot, such as corner points. These cursors function in a similar manner as with conventional oscilloscope.
- Find the average, rms, or the integrated values of a trace. Figure 1.8 shows an example of how to use cursors and perform trace integration.
- Fast Fourier Transform (FFT).

As a matter of exercise, create two plot panes. Use them to display power deposited in the resistor as a function of time in one pane, and power deposited in the capacitor in the other pane. Find the average values for dissipated power in the two elements. Does the average value for power dissipated in the capacitor agree with your expectations? After that, keeping only one pane, plot voltage across the capacitor vs. the current flowing through it. What is the shape of this trace?

Next, modify the circuit by including a diode in parallel with the resistor. Right-click on the diode, then click on Pick New Diode button. This will bring a list of

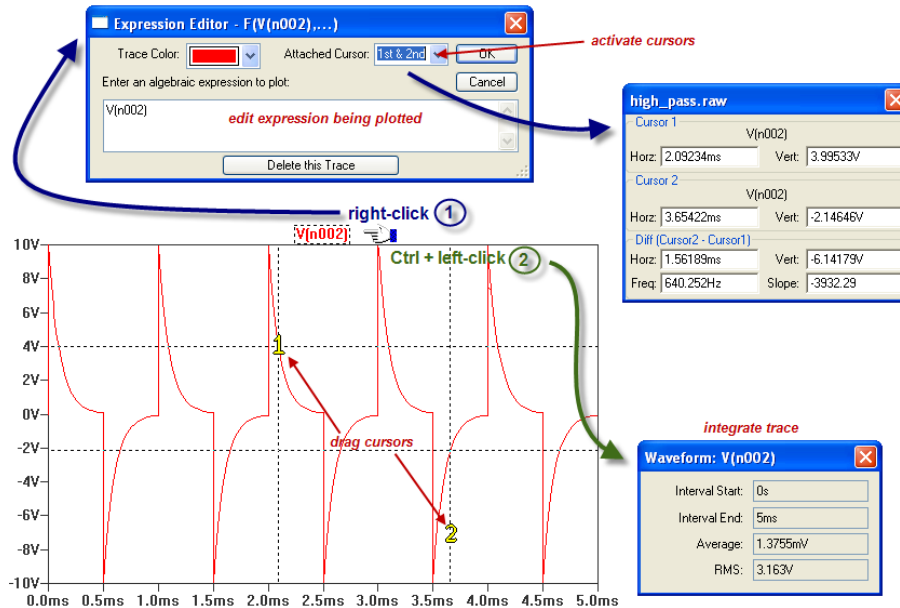


Figure 1.8: Use of cursors and trace integration in the plotting utility.

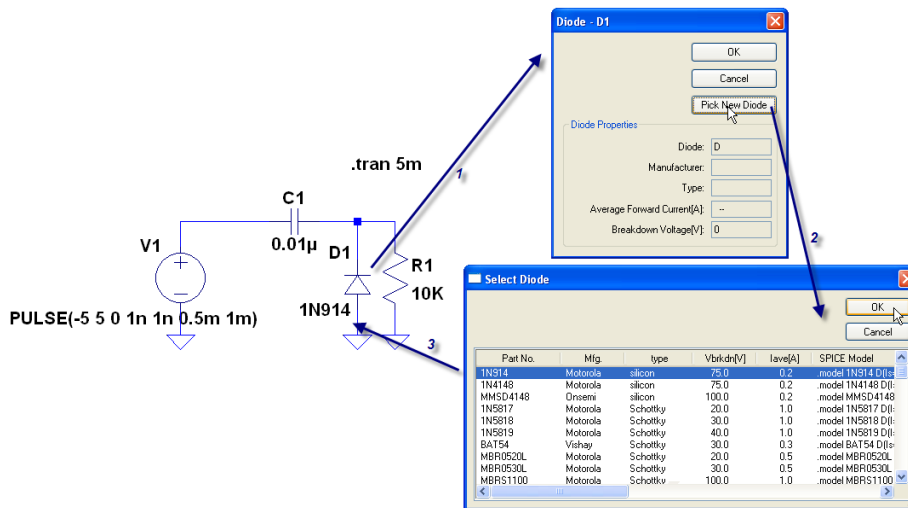


Figure 1.9: Adding 1N914 diode to the circuit.

diode models available in LTSPICE library. Select 1N914 diode. See Figure 1.9. Rerun the simulation and plot the voltage across the resistor. Do you see what you have expected?

By selecting View→SPICE Netlist , you can bring up netlist for the circuit you have built. This netlist can be used in other SPICE-based programs, or special utility programs, such as printed circuit board design tools [3]. The netlist for your circuit should look something like:

```

1 * Path to where you saved .asc (LTspice) file of the circuit
2 R1 N002 0 10K
3 C1 N001 N002 0.01
4 V1 N001 0 PULSE(-5 5 0 1n 1n 0.5m 1m)
5 D1 0 N002 1N914
6 .model D D
7 .lib C:\Program Files\LTC\SwCADIII\lib\cmp\standard.dio
8 .tran 5m
9 .backanno
10 .end

```



The meaning of most of the lines can be recognized in the light of what we have already seen. The diode on line 5 has 1N914 in place of its value. Line 6 informs SPICE that a particular diode model should be used (in this trivial form, this command is redundant and can be removed), whereas the actual parameters for 1N914 diode are contained in the library file specified using the `.lib` directive on line 7. Line 9 is an LTSPICE-specific command inserted in every automatically generated `netlist` and does not affect the actual simulation flow (it is only used in postprocessing with the plotting utility).

## 1.2.2 Example 2: Simplistic Op-Amp

Here we will use a simple op-amp example (Problem 6.3 from the Lab Manual) to demonstrate other capabilities of LTSPICE.

**Goals:** use of Net Labels; quiescent point analysis using `.op` directive; use of parameters (`.param` directive); stepping a parameter value (`.step` directive); using measurement command `.meas`; small gain AC analysis using `.ac` directive.

To get started, draw the circuit as shown in the Lab Manual. For transistors choose two `pnp` and one `npn` symbols from the list of components. You will have to rotate or mirror the symbols for their optimal placement. Selecting a specific model for transistors is done in the same way as assigning a model for diodes — right-click on the transistor, press **Pick New Transistor**, then choose 2N3906 for PNP and 2N3904 for NPN. Specify values for all of the components. Also, rename the components to follow the schematic. See Figure 1.10 for what your circuit should look like at this stage.

Next, use a feature called Label Net to rename the nodes from their default names, as shown in Figure 1.11. Node relabeling is done by pressing either  or the Toolbar icon , typing a new name, and then attaching it to the circuit. Each node is then referred by its (new) name, which must be unique for the circuit. If two or more identical net names are present, SPICE considers them to be electrically connected (even if there is no wire present in the schematic capture). This feature is

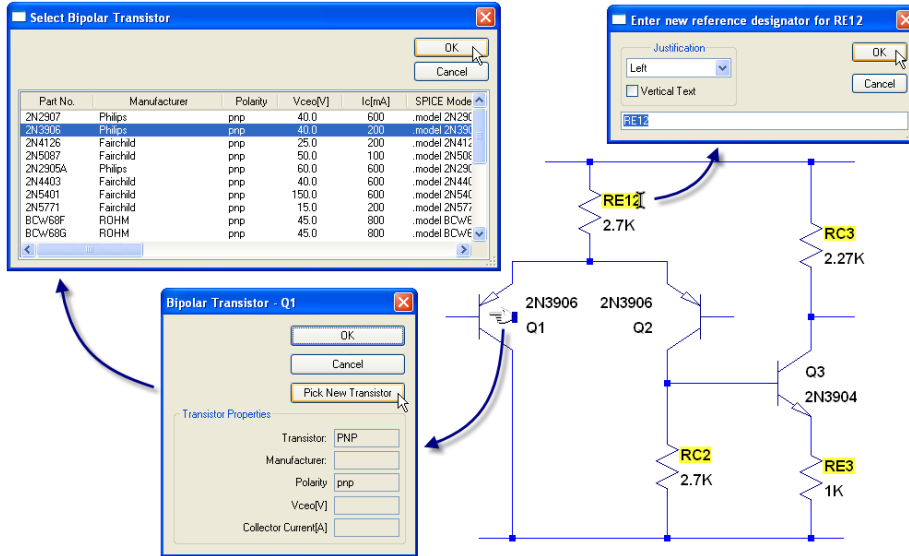


Figure 1.10: Creating a simple op-amp circuit.

particularly convenient to use in a circuit with many interconnections to avoid clutter in the schematic. Here we will use this feature to “attach” voltage sources: two DC ( $\pm 15V$ ) and two SINE sources. A sinusoidal voltage sources can be created in analogy to PULSE source of the previous example. Here they are given amplitudes of 100mV and 95mV for inverting and non-inverting inputs of the differential amplifier,  $V_a$  and  $V_b$  respectively. The frequency is set to 1kHz.

### Q-point analysis

SPICE directive `.op` can be used to perform Q-point analysis. It computes the DC operating point treating capacitances as open circuits and inductances as short circuits. This command is run implicitly by SPICE prior to performing other types of analyses such as small-signal AC gain calculations. We will invoke this command explicitly to determine Q-point of this circuit. SINE voltage sources assume their DC offset values for the Q-point analysis, which are zero in our case for  $V_a$  and  $V_b$  sources. Create and place `.op` command in the circuit either by selecting **Simulate**→**Edit Simulation Cmd** from the Menu bar and then going to **DC op pnt** tab or typing it in directly after pressing **[S]** key. Completed circuit ready for simulation is shown in Figure 1.11.

Let’s briefly examine `netlist` by going to **View**→**SPICE Netlist** Menu. It should look like the following (comments added manually). Note the use of new node names in the description of the circuit.

```
* Path to the saved .asc (LTspice) file with the circuit
V1 VEE 0 15 ;;;;;;;;;;;;;;
V2 VCC 0 -15 ; circuit description ;
```



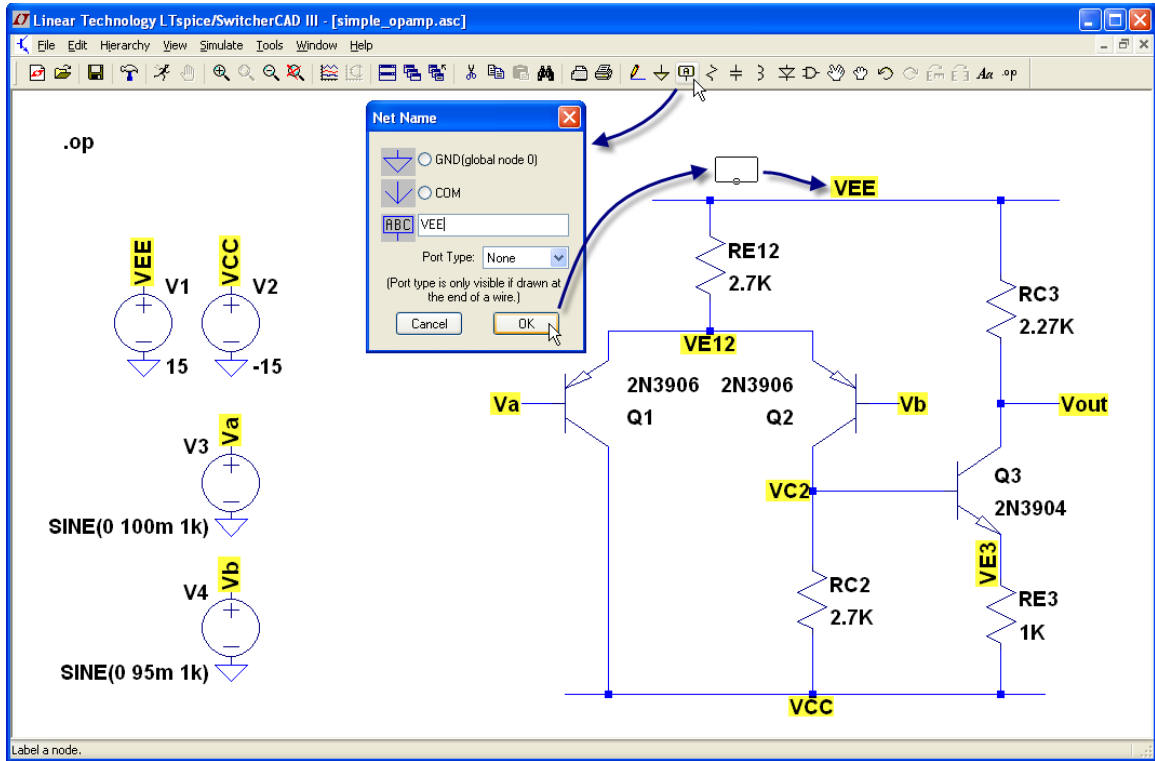


Figure 1.11: Circuit with renamed nodes and .op SPICE directive added.

```

Q1 VCC Va VE12 0 2N3906 ; Q for a BJT ;
Q2 VC2 Vb VE12 0 2N3906 ; ;
RC2 VC2 VCC 2.7K ; ;
RE12 VEE VE12 2.7K ; ;
V3 Va 0 SINE(0 100m 1k) ; ;
Q3 Vout VC2 VE3 0 2N3904 ; ;
RE3 VE3 VCC 1K ; ;
RC3 VEE Vout 2.27K ; ;
V4 Vb 0 SINE(0 95m 1k) ;;;;;;;;;;;;;;
.model NPN NPN ;;;;;;;;;;;;;;
.model PNP PNP ; SPICE directives ;
.lib C:\PROGRA~1\LTC\SwCADIII\lib\cmp\standard.bjt
.op ; ;
.backanno ; ;
.end ;;;;;;;;;;;;;;

```

Running simulation of this example brings up a window with Q-point values of all voltages and currents:

--- Operating Point ---

```

V(vee):      15          voltage
V(vcc):      -15         voltage
...
Ic(Q3):      0.00614037  device_current
Ib(Q3):      1.90069e-005 device_current
Ie(Q3):      -0.00615938 device_current
...

```

You can access these values after closing the window by pointing the cursor to the circuit. A short description and the calculated value will appear in the lower-left corner of the main window. Pointing to various components also reports dissipated power for the DC operating point.

It is possible to replace any of the component values with a named variable — a parameter — which can be assigned a numerical value elsewhere. For example, right-click on RC3 resistance value, 2.27K, and type-in a variable name in curly braces {RC3val} in its place. Next, create a SPICE directive

```
.param Rval 2.27K
```

(without curly braces!) and rerun the simulation. You should get the same result as before.

Note that the Q-point for V(Vout) is offset from zero. You can change RC3val to bring it closer to zero, which would be more along the lines of a desired op-amp behavior. In a real-life implementation of this circuit, such details will depend on the actual transistors used, which can differ significantly from their typical specifications used by the program. There is a way to scan a parameter using .step directive. Create a SPICE command

```
.step param RC3val 2.2K 2.7K 10
```

and place it somewhere in the schematic. This command instructs the program to repeat calculation while scanning the parameter RC3val from 2.2k $\Omega$  to 2.7k $\Omega$  with a step of 10 $\Omega$ . Refer to LTSPICE help files for additional information about .step command syntax. Run the example again, then click on Vout node to plot Q-point voltage as a function of the scanned parameter. What value of RC3 produces V(Vout) = 0? You can find the relevant RC3val value with a help of a trace cursor in the plotting utility by clicking on the legend. Another way to determine the RC3 value is through a powerful .meas[ure] command. We will gradually introduce some of its functionalities. Refer to LTSPICE help for a complete description. This command allows to evaluate a user-defined quantity derived from the results of simulations. Create the following SPICE directive:

```
.meas op RC3ideal find RC3val when V(Vout)=0
```

and place it on your schematic. This command instructs SPICE to find RC3val value when V(Vout)=0 and label the result RC3ideal. The second (optional) word “op” in this command specifies which type of analysis .meas was applied to (e.g., it would be .tran for time-transient analysis). By now all of your SPICE directives (an exact

order is irrelevant) should look like:

```
.op
.step param RC3val 2.2K 2.7K 10
.meas op RC3ideal find RC3val when V(Vout)=0
```

Run the simulation again. Choose **View**→**SPICE Error Log** from the Menu bar. Find a string in the log file that looks like this:

```
rc3ideal: rc3val={actual value} at ...
```

and substitute this value for RC3 resistance. Run the example again using `.op` command (remove or comment out all other directives) to verify that indeed  $V_{out} = 0$  as expected.

Save this circuit to `simple_opamp.asc` file for future work.

### Small-signal AC gain

Here we will determine a small-signal AC gain of the simple op-amp using several different methods. First, let's calculate AC gain using time-transient analysis. Simulate the circuit for 10ms by including `.tran 10m` command. Next, create two plot panes and display the input signal  $v_{diff} = V_b - V_a$  on one and  $V_{out}$  on the other, see Figure 1.12. Activate two trace cursors for each of the plot, and “measure” peak-to-peak voltage (the difference between the cursors) for each case. Find the AC gain  $G = \Delta V_{out} / \Delta v_{diff}$ .

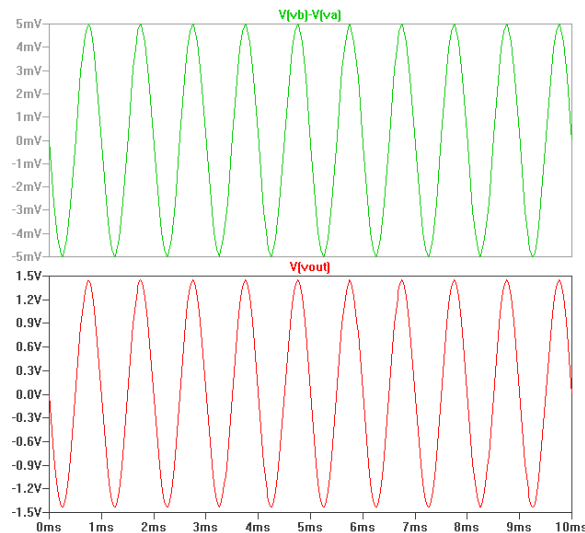


Figure 1.12: Displaying results of time-transient analysis in two plot panes.

Another way to extract this information is to use `.meas` commands. The following three lines achieve the desired result.

```
1 .meas tran Vout_pp pp V(Vout)
2 .meas tran vdiff_pp pp V(vb)-V(va)
```

```
3 .meas tran G param Vout_pp/vdiff_pp
```

Line 1 finds the peak-to-peak (**pp** keyword) value of  $V_{out}$  and labels the result **Vout\_pp**. Other possible keywords are **avg** (average), **max**, **min**, **rms**, and **integ** (integrate the expression following it). Line 2 does a similar thing for  $V_b - V_a$ . The last line performs some basic arithmetic on parameters **Vout\_pp** and **vdiff\_pp** and stores the result in variable **G**. Run the simulation with these 3 lines included as SPICE directives and view the computed value of **G** in the **.log** file. Compare the result to the previously calculated value.

A dedicated SPICE command to calculate small-signal AC gain is **.ac**. Choose **Simulate**→**Edit Simulation Cmd**, then select **AC Analysis** tab. Select **Decade** for a type of sweep, with 10 points per decade starting from 1Hz to 50MHz. This type of analysis requires an AC stimulus — a voltage source that will be “driven” at different frequencies. For example, to turn  $V_a$  source into an AC stimulus, bring up its voltage source window, and type in 1 into the **AC Amplitude** field. A label **AC 1** will appear near the voltage source. Now you can run the simulation and display Bode plot of the gain vs. frequency by clicking on **Vout** node. See Figure 1.13. A solid line shows the magnitude, and a dashed line the phase. By clicking on the magnitude axis you can choose for the gain to be displayed on a linear or logarithmic scale. By clicking on the phase axis, you can choose to alternatively plot *group delay* (a measure of the transit time of a signal through the device) associated with the complex gain. Note  $180^\circ$  phase at low frequencies, which indicates  $V_a$  to be an inverting input. Compare the gain at 1kHz with what you have found above using time-transient analysis.

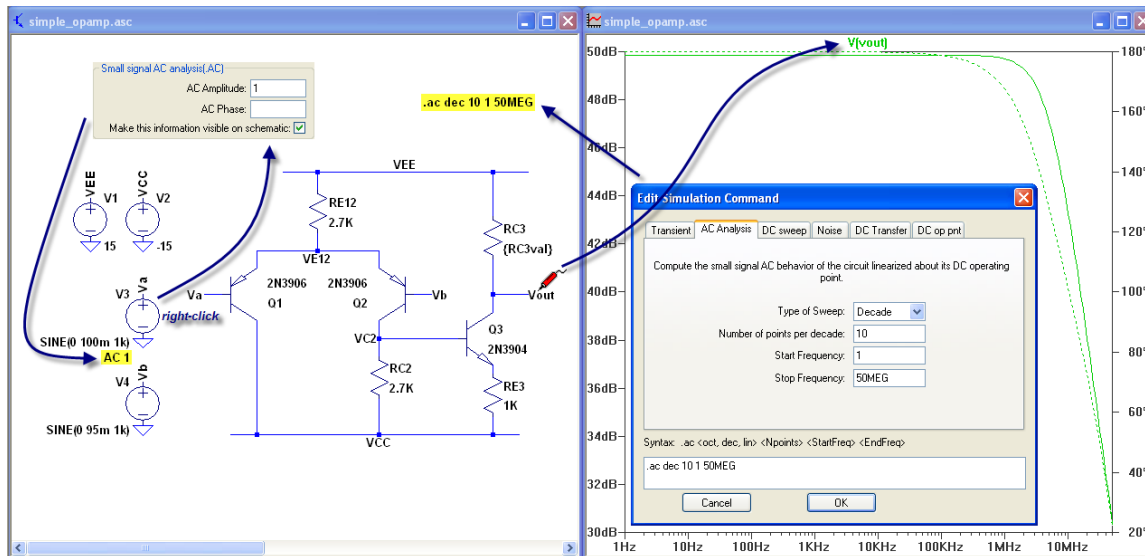


Figure 1.13: Setting up **.ac** analysis of the circuit.

### 1.2.3 Example 3: Using External Models

Here we will consider 3 short examples: a voltage follower, an astable multivibrator, and a ring oscillator. In these examples we will explore how to incorporate external models (not part of LTSPICE) into your simulations.

Goals: assign manufacturer's SPICE model to op-amp; introduce `.subckt` directive; set initial conditions with `.ic` directive; use of `.lib` and `.include` statements.

#### Stability Of Voltage Follower

In this example, we will investigate the stability of a voltage follower built with LM741 op-amp. LM741 is not a Linear Technology product and does not come with LTSPICE library, but this and similar models can be easily added to your circuits. Go to <http://www.national.com> and search for "LM741 model" (or search on the Internet for "LM741 SPICE model"). There, you will find a file called LM741.MOD. This model file is available on your lab computer in SPICE\external\_components directory. Download or copy the file to your local directory where you save your LTSPICE examples.

LM741.MOD is a SPICE model for LM741 op-amp distributed by its manufacturer, National Semiconductor. Most manufacturers provide SPICE models of many of their products, which can be used in LTSPICE simulations. LM741.MOD header contains the following:

```
*/*****
* (C) National Semiconductor, Inc.
* Models developed and under copyright by:
* National Semiconductor, Inc.
*/*****
* Legal Notice: This material is intended for free software support.
* The file may be copied, and distributed; however, reselling the
* material is illegal
*/*****
* For ordering or technical information on these models, contact:
* National Semiconductor's Customer Response Center
*           7:00 A.M.--7:00 P.M. U.S. Central Time
*           (800) 272-9959
* For Applications support, contact the Internet address:
*   amps-apps@galaxy.nsc.com
*/*****
*LM741 OPERATIONAL AMPLIFIER MACRO-MODEL
*/*****
*
* connections:      non-inverting input
*                   |   inverting input
*                   |   |   positive power supply
*                   |   |   |   negative power supply
*                   |   |   |   |   output
*                   |   |   |   |   |
*                   |   |   |   |   |
* .SUBCKT LM741/NS  1  2  99  50  28
* ..
```

.SUBCKT command starts a description of the actual model. This directive can be used as an aid in defining a circuit through inclusion of repetitive circuitry contained in a subcircuit definition. Before the simulation runs, the circuit is expanded to a flat netlist by replacing each invocation of a subcircuit with the circuit elements in the subcircuit definition. The end of a subcircuit definition must be a .ends directive.

Create a circuit of the voltage follower driving a capacitive load C1. Use generic opamp2 symbol found in the component selection window, see Figure 1.14. “Power” the op-amp with  $\pm 15\text{V}$  voltage supplies. Then setup a square-wave voltage source with  $\pm 1\text{V}$  amplitude and  $1\text{kHz}$  frequency and connect it to the voltage follower. Use labels as shown in Figure 1.14. To study the behavior of the output for different C1 capacitance values, use parameter {C} in place of its capacitance. Save this circuit to the same directory as LM741.MOD file.

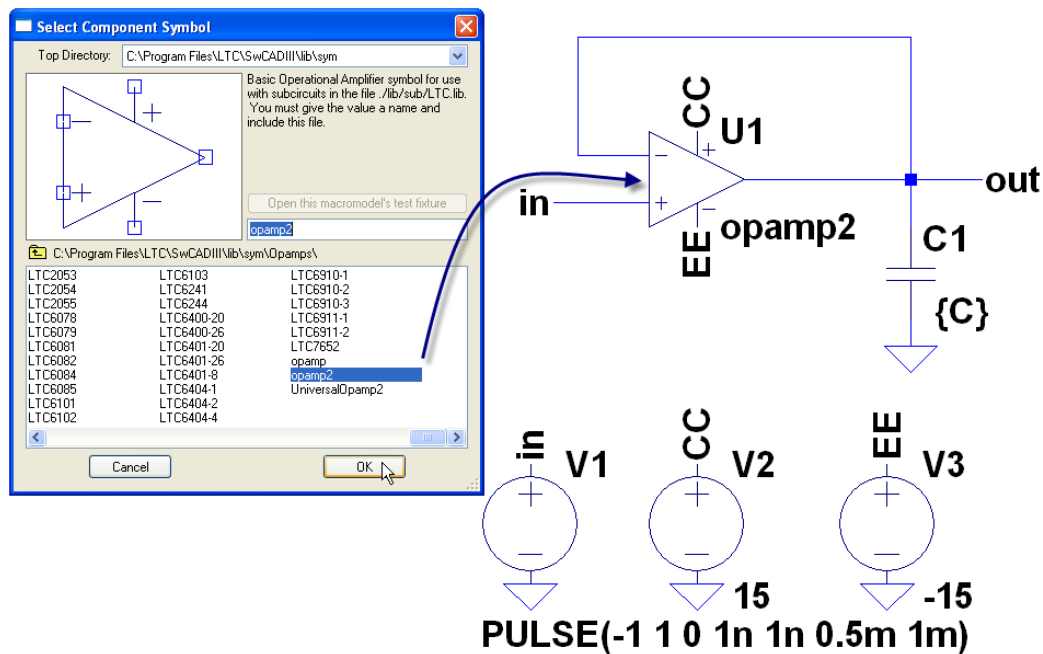


Figure 1.14: Voltage follower circuit.

Symbol opamp2 does not have any model associated with it and this circuit will not work as is. To use LM741/NS with this symbol you must perform two steps. 1) Right-click on the op-amp to open Component Attribute Editor window, and modify Attribute Value so that it reads LM741/NS. This string should match the name of the subcircuit found on .SUBCKT line in the LM741.MOD file. 2) Add .include LM741.MOD command to your circuit. This directive includes the named file as if that file had been typed into the netlist in place of the .include command. An absolute path name may be entered for the filename. If a relative path is provided, LTSPICE looks first in the directory <SwCADIII>\lib\sub and then in the directory that contains the calling netlist (the local directory). <SwCADIII> is the directory containing the scad3.exe

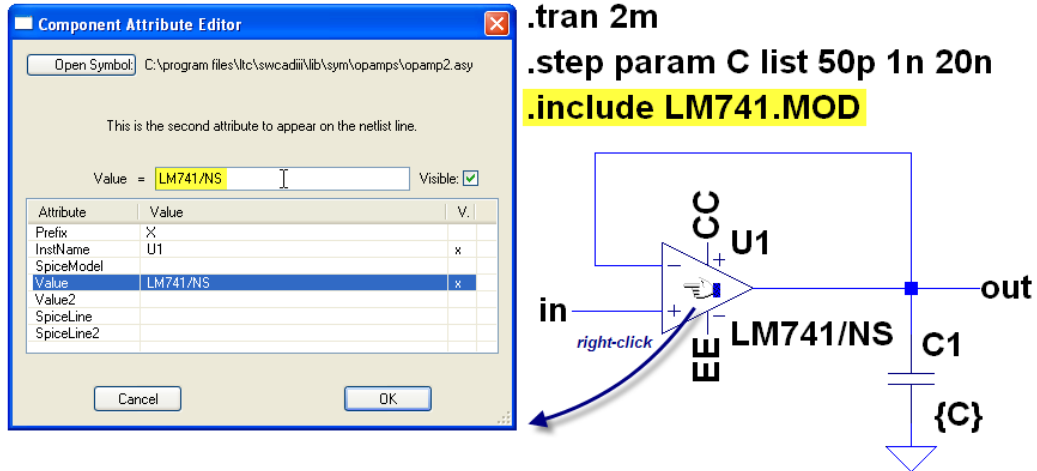


Figure 1.15: Setting up LM741 op-amp.

(LTSPICE) executable, typically installed in C:\Program Files\LTC\SwCADIII. Add a time-transient analysis command, and a command for stepping parameter C so that the directives included to your schematic read (see Figure 1.15):

```
.tran 2m
.step param C list 50p 1n 20n
.include LM741.MOD
```

The second line instructs parameter C to adopt its values from a list of arbitrary values following list keyword (here 50pF, 1nF, and 20nF).

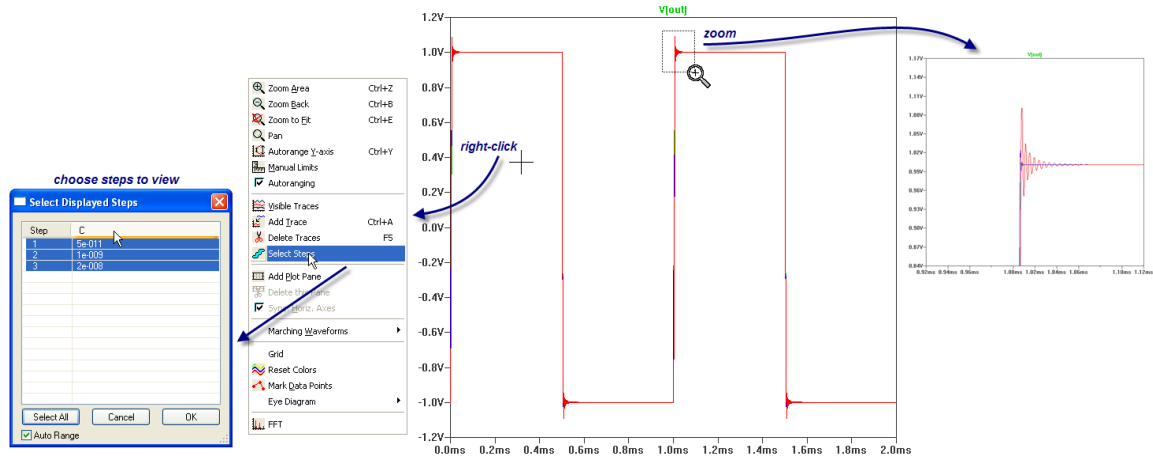


Figure 1.16: Viewing the output of the voltage follower.

Run the simulation and plot V(out). Zoom in on the rising or falling edge of the pulse. See Figure 1.16. The ringing that can be seen indicates that the circuit is marginally unstable. The circuit is more unstable with the larger capacitor due to

a smaller phase and gain margins of the loop gain. A problem at the end of Part 1 asks you to find phase and gain margins for different capacitor values through `.ac` calculations of the loop gain of the circuit. When using `.step` command, the plotting utility displays all traces corresponding to all steps of the parameter. You can also choose to include specific steps in the graph by right-clicking in the plot window and choosing **Select Steps** from the Menu.

Save this circuit to `voltage_follower.asc` file for future work.

## Astable Multivibrator


You can simulate circuits in LTSPICE without explicit voltage sources such as an astable multivibrator. There may be a problem with starting oscillations in a circuit that employs positive feedback, which we will demonstrate now. Build an astable multivibrator using LM741 op-amp and perform time-transient analysis for 10ms. See Figure 1.17. As you can see from `V(out)` output, the oscillations do not start until after 5ms into the simulation. As you recall, a positive feedback circuit may have an unstable equilibrium point where all voltages are zero so that in an idealistic case (e.g., in simulations where voltages may be *exactly* zero, with no input offset voltage in the op-amp, etc.) the self-induced oscillations may take considerable time to start, or may never start at all. To resolve this issue, you can include `.ic` statement to your circuit:

```
.ic V(inv)=1m
```

This statement specifies initial conditions for time-transient analysis. It sets the initial voltage on the inverting input of the op-amp to be 1mV. Add this statement and rerun the example. Observe how the output has changed.

Save this circuit to `multivibrator.asc` file for future work.

## Ring Oscillator

You can also build a multivibrator using digital components. Digital circuitry is introduced in the second half of the course. Here, you will learn how to build a simple ring oscillator using NOT gates, and in particular how to include integrated circuit models that are not part of LTspice for a later use in the course. Very briefly: digital gates are devices that operate on two-state voltage signals, for example 0V and 5V (voltages need not to be exact). One state will be known as TRUE, and the other one as FALSE. An inverter (NOT) gate is the simplest digital component with only one input and one output that inverts the digital signal supplied to its input. It performs a Boolean logic operation known as negation. Its symbol  is usually drawn as a triangle (buffer) with an inverting circle (bubble).

Next, consider a circuit in which the output of a NOT gate is connected to its input. Such a configuration would be meaningless for an ideal NOT gate, because the operation of negation can never be satisfied in such a case. In reality, all gates have a finite *propagation delay*, which can be defined as a time period starting from



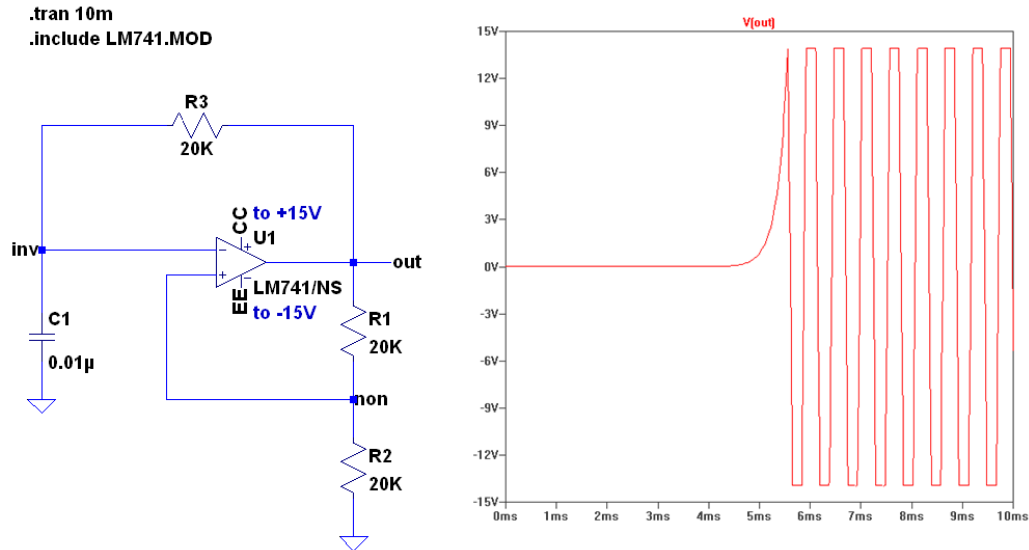


Figure 1.17: Astable multivibrator circuit (on the left) that displays a delay in starting the oscillations (on the right).

when the input to a logic gate becomes stable and valid to when the output of that logic gate is stable and valid (“valid” here means that it is one of the two possible voltage levels, TRUE or FALSE). For example, 74HCT04 NOT gate from Phillips is specified to have a typical propagation delay of 10ns under usual operating conditions. Connecting the gate’s output to its input will lead to oscillations between the two voltage levels (FALSE and TRUE), with a period equal to twice the propagation delay — about 20ns or 50MHz. A similar situation occurs when any *odd* number of NOT gates are connected together, except the period of oscillations becomes longer by the number of gates being used. This configuration is known as *ring oscillator*.

You can find a selection of digital components for LTSPICE in the directory `SPICE\external_components` on your laboratory computer. These files are also available from the course web-site. Copy `74HCT.LIB` file containing SPICE models of various gates from this digital family (found in `Digital_74HCTxxx` directory) and a NOT gate circuit symbol `74hct04.asy` (found in `Digital_74HCTxxx\74HCT`) to your local directory. Create a new schematic in LTSPICE and save the file as `ring_oscillator.asc` to the *same* directory as the other files. Next, insert the inverter gate symbol in your schematic by bringing up **Select Component Symbol** window (press `F2`), and changing **Top Directory** to your local directory. See Figure 1.18. To use the component you also need to include the library file using `.lib 74HCT.LIB` or `.include 74HCT.LIB` directive. The difference between `.lib` and `.include` is that the former inserts the contents of the file only between `.subckt` and `ends` commands, while ignoring other parts of the file. This command is convenient when you want to reuse model definitions from another circuit file or `netlist`, without including the

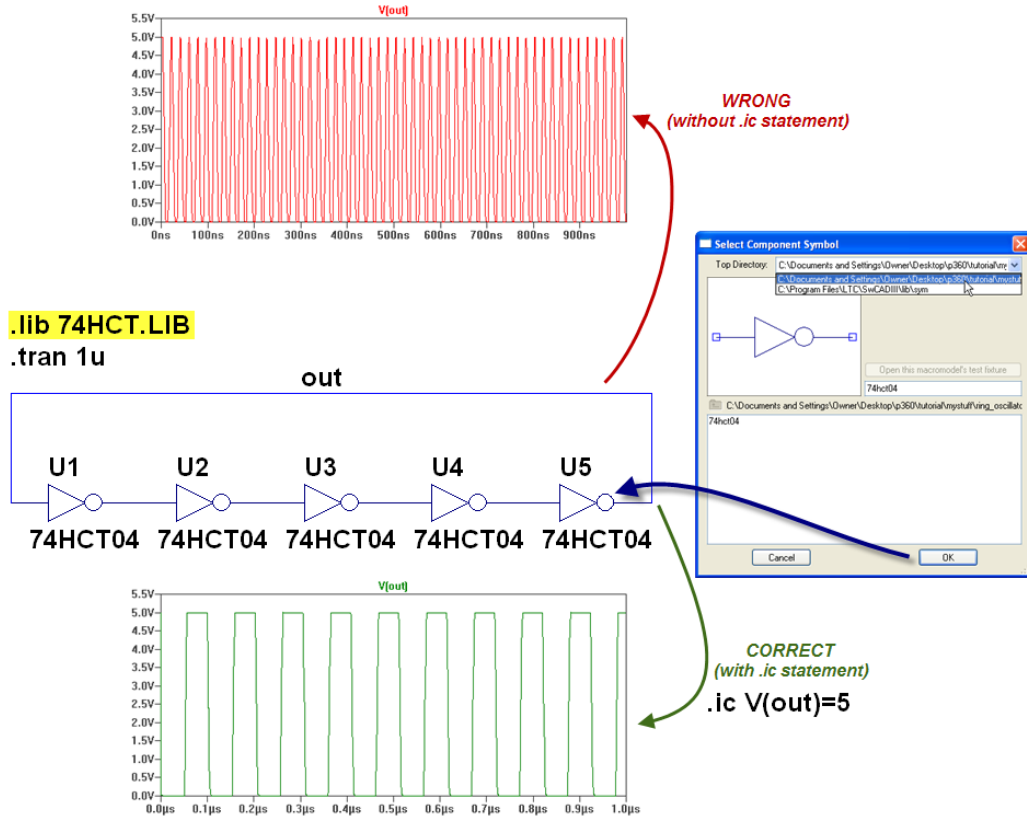


Figure 1.18: Oscillator ring using 5 inverter gates 74HCT04.

entire circuit. Build the circuit as shown in Figure 1.18 and perform time-transient analysis for  $1\mu\text{s}$ . If you do not include the initial condition statement `.ic V(out)=5`, the circuit will be simulated incorrectly. In particular, the frequency of oscillations will be the same as if only a single inverter gate were present, whereas a real-life version of this circuit would tend to produce a frequency  $\times 5$  lower. A closer look at the circuit reveals that all 5 NOT gates are performing oscillations in a perfect unison — a scenario possible only in the ideal world of simulations! Real-life gates have slightly different propagation delays, which precludes such behavior from happening. The initial condition statement ensures that such pathological case does not occur in the simulation.



## 1.2.4 Example 4: Creating Custom Components

Goals: create custom components and circuit blocks for inclusion in other circuits.

In addition to the library of components in LTSPICE, models for many different types of analog and digital devices may be found on the Internet [4]. Nevertheless, occasionally one needs to create a custom component because a particular device model

has not been implemented. This ability is also useful for a hierarchical organization of large circuits — it may be convenient to create a new component that represents a block (a subcircuit), which either repeats several times in the circuit or represents a unit with a well-pronounced functionality. Schematics organized in such a fashion facilitate their reading and understanding.

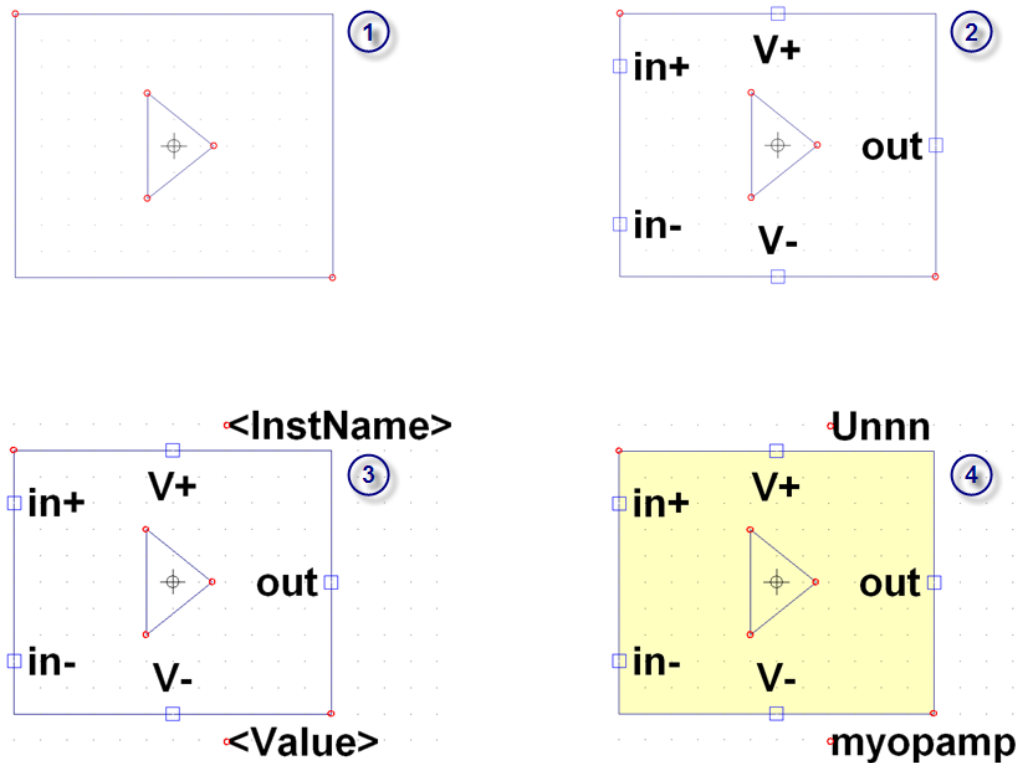


Figure 1.19: Steps to creating a custom symbol.

### Custom Component Symbol

To create a new component, one needs a circuit symbol and a SPICE model (`netlist`) of a device or a circuit block. In this example, we will go through these steps. First, you have to draw the component's symbol that will be used in LTSPICE schematic capture or, alternatively, you can reuse an already existing one. To begin from scratch, select `File`→`New Symbol` from the pull-down Menu. Once the program is in the symbol editing mode, an additional `Draw Menu` provides options to draw a line, rectangle, circle, arc, and to add text to your symbol. Draw a simple symbol for an op-amp as shown in Figure 1.19.

Step 1. Create a symbol, as shown in Figure 1.19, consisting of a rectangle and a triangle.

Step 2. Select **Edit**→**Add Pin/Port** from the Menu to add pins to the symbol in the following order: **in+**, **in-**, **V+**, **V-**, and **out**.

Step 3. Select **Edit**→**Attributes**→**Attribute Window** from the Menu. Select **InstName** attribute to add to the schematic. It will display the actual value of the attribute in your schematic. Repeat this step for **Value** attribute.

Step 4. Select **Edit**→**Attributes**→**Edit Attributes** and enter **X** for **Prefix**, **myopamp** for **Value**, and some description message such as ‘alternative op-amp symbol’ for **Description**.

The most important attribute is called **Prefix**. It determines the basic type of a symbol. If the symbol is intended to represent a SPICE *primitive*, the symbol should have an appropriate prefix: **R** for resistor, **C** or capacitor, **M** for MOSFET, etc. The prefix should be **X** if you want to use the symbol to represent a subcircuit that incorporates an external **netlist** model (e.g. LM741). The rectangle in your schematic will be changed to a filled shape.

The next significant attribute is **Value**. As you recall from the LM741 example, it should match the string after the **.subckt** command in the external file. For now we set this value to **myopamp**, but it may have to be replaced with whatever string on the **.subckt** command line in the model file to be used with this symbol.



The order of pins is significant — it must match that of the nodes from the **.subckt** line. The specified order of pins was chosen so that the symbol can be used with standard model files for op-amps, such as **LM741.MOD**. The order of the nodes can be seen from the file content:

```

...
*////////////////////////////////////
*LM741 OPERATIONAL AMPLIFIER MACRO-MODEL
*////////////////////////////////////
*
* connections:      non-inverting input
*                   |   inverting input
*                   |   |   positive power supply
*                   |   |   |   negative power supply
*                   |   |   |   |   output
*                   |   |   |   |   |
*                   |   |   |   |   |
*                   |   |   |   |   |
.SUBCKT LM741/NS 1 2 99 50 28
...

```

Double-check that the order of the pins in your symbol matches the order found in **LM741.MOD** file by bringing up **View**→**Pin Table** from the Menu. See Figure 1.20. Save the symbol to **myopamp.asy** file.

Because this symbol is pin-compatible with the other op-amp symbol **opamp2** you have been using, you can use these two symbols interchangeably (for example, with **LM741.MOD** file). To use the new symbol, make sure **myopamp.asy** file is placed in the same directory as your **.asc** schematic file. Bring up **Select Component Symbol** window and change **Top Directory** to your local directory. You should see the new symbol in the window available for insertion into your schematic.

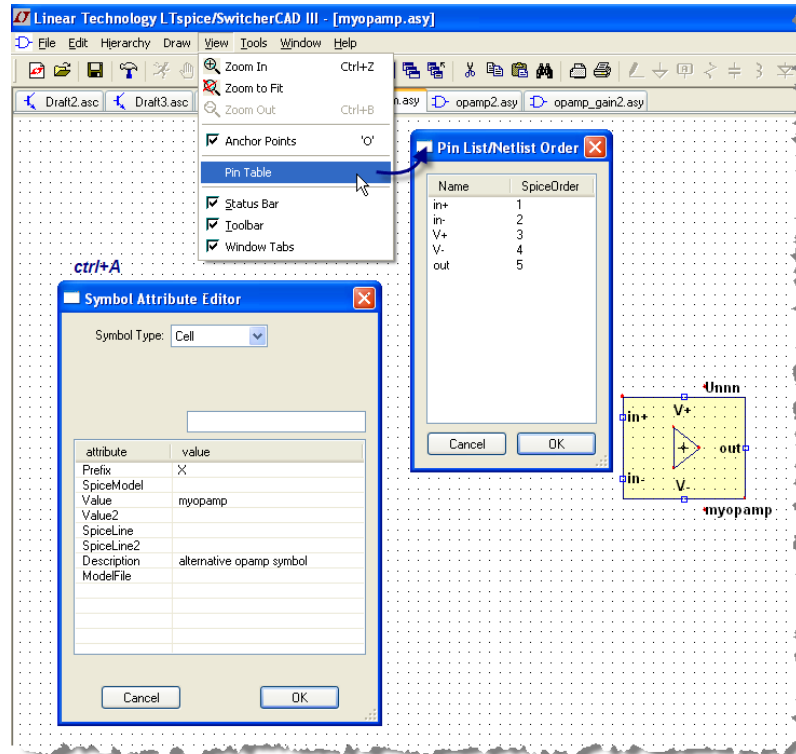


Figure 1.20: Attributes and correct pin order for the op-amp symbol.

```

1  * Path to the saved .asc (LTspice) file with the circuit
2  V1 VEE 0 15
3  V2 VCC 0 -15
4  Q1 VCC Va VE12 0 2N3906
5  Q2 VC2 Vb VE12 0 2N3906
6  RC2 VC2 VCC 2.7K
7  RE12 VEE VE12 2.7K
8  V3 Va 0 SINE(0 100m 1k)
9  Q3 Vout VC2 VE3 0 2N3904
10 RE3 VE3 VCC 1K
11 RC3 VEE Vout 2.27K
12 V4 Vb 0 SINE(0 95m 1k)
13 .model NPN NPN
14 .model PNP PNP
15 .lib C:\PROGRA~1\LTC\SwCADIII\lib\cmp\standard.bjt
16 .op
17 .backanno
18 .end

```

Figure 1.21: Raw SPICE file automatically generated for the three-transistor op-amp.

## Custom SPICE Model

Next, we will create an external SPICE model containing the simple op-amp from the earlier Example 2. Generally, creating a new model requires knowledge of SPICE

```

* myopamp.txt: model for a three-transistor op-amp
.SUBCKT myopamp Vb Va VEE VCC Vout
Q1 VCC Va VE12 0 2N3906
Q2 VC2 Vb VE12 0 2N3906
RC2 VC2 VCC 2.7K
RE12 VEE VE12 2.7K
Q3 Vout VC2 VE3 0 2N3904
RE3 VE3 VCC 1K
RC3 VEE Vout 2.27K
.model NPN NPN
.model PNP PNP
.lib C:\PROGRA~1\LTC\SwCADIII\lib\cmp\standard.bjt
.backanno
.ends

```

Figure 1.22: Contents of `myopamp.txt` SPICE model file.

language. However, with LTSPICE automatically creating `netlists`, this job largely amounts to copying and pasting.

Open the file `simple_opamp.asc`, which you saved earlier. Copy the contents of SPICE `netlist` (from `View`→`SPICE Netlist Menu`) and paste it into a text editor (e.g., Notepad). See Figure 1.21. Examine the file. It may be slightly different than what is shown in Figure 1.21. Note that lines 2, 3, 8 and 12 contain definitions for voltage sources, and that line 16 contains a SPICE directive for circuit analysis. Delete these lines as they do not belong in the subcircuit body. Next, place the following line at the beginning of the file (after the first comment line):

```
.SUBCKT myopamp Vb Va VEE VCC Vout
```

and change `.end` to `.ends`. Note that the order of the pins must follow that of the symbol: non-inverting input, inverting input, positive power supply, negative power supply, and the output. The modified file should look like what is shown in Figure 1.22.

Save the model file to `myopamp.txt`. Now you can use this model file along with `myopamp` symbol by setting `Value` to `myopamp` and adding `.include myopamp.txt` statement to your circuit.

## Blocks And Hierarchical Organization Of Large Circuits

There exists a convenient way in LTSPICE to organize large circuits using *blocks*. Each block is a component symbol (stored in `some_name.asy` file), which is linked to a circuit schematic with identical filename except for the file extension (stored in `some_name.asc` file). Node association is achieved through matching names of pins in the symbol file and net labels in the schematic file. Right-clicking on such component (also known as block) allows one to open and edit its schematic. While the use of SPICE models through `.include` statement fits best when representing an existing device or a component, circuit blocks are a convenient way to organize large circuits under development, or, when the internal workings of a lower-hierarchy device need to be looked at and possibly tweaked.

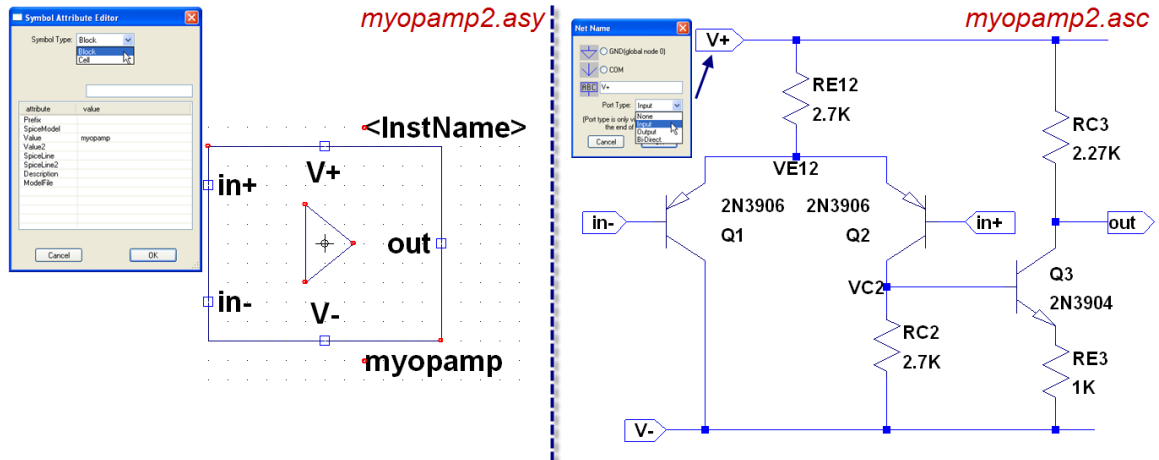


Figure 1.23: Circuit block: symbol (on the left) and schematic (on the right).

We will create a circuit block using the same simple op-amp (Example 2). Copy `myopamp.asy` and `simple_opamp.asc` files to a single directory giving them the new names: `myopamp2.asy` and `myopamp2.asc`, respectively. Open `myopamp2.asy` symbol file. Bring up a window with the symbol's attributes (`[Ctrl] + [A]`) and choose **Block** for the **Symbol Type**. Next, make sure all attributes are cleared (**Prefix** in particular) because a block does not support attributes. See Figure 1.23. Save the file.

Open `myopamp2.asc` file with LTSPICE for editing. The schematic does not need voltage sources and SPICE directives for analysis because they will be supplied externally. The names of the nodes that will interface the block in a higher hierarchy circuit must match those of the pins in the symbol file. You can choose the appropriate **Port Type** symbols when assigning Net Labels to help visualize which nodes are meant to be inputs and which ones outputs. Simplify the circuit as shown in Figure 1.23 and save the file.

A block can be used in another circuit similar to any other component. No `.include` statement is necessary, but both the symbol and schematic files must be in the same directory. When you right-click on the block, you are given a choice of whether to open its symbol or its schematic for viewing and/or editing. If you commit any changes to the schematic, they will propagate into the higher level circuit that uses it. An arbitrary number of blocks can be used in the same circuit.

You can also view voltages and/or currents in subcircuits. By default LTspice does not save this information for blocks. However, you can enable it by choosing **Tools**→**Control Panel** from the Menu, then under **Save Defaults** tab select **Save Subcircuit Node Voltages** and **Save Subcircuit Device Currents**.

## 1.3 Practice Problems

1. Build an astable multivibrator using the SPICE model of the simple three-transistor op-amp and its symbol that you made earlier (with `.include` statement) using  $\pm 15\text{V}$  power supply. Print out few cycles of the output. Determine the peak-to-peak voltage of the output (use `.meas` directive). Is the output symmetric around zero? Explain why.
2. An enhancement n-channel MOSFET can be connected in a diode configuration, see Figure 1.24. Obtain I-V curve of such a “diode” for 2N7000 n-channel enhancement MOSFET scanning the voltage from  $-0.9$  to  $12\text{V}$  and print the result. (Hint: use `.op` and `.step` directives). Find the incremental resistance from I-V curve at  $5\text{V}$  drain-source voltage. What is the dissipated power at that point? Use 2N7000 SPICE model from the library files available from the course web-site. This model requires a 4-input symbol, which can be found along with the model file in `SPICE\external_components\2N7000` directory. Make sure that `SpiceModel` attribute of the component is set to 2N7000 and that the appropriate `.include` statement is present. The 4th input is used to specify temperature for SPICE model of this MOSFET. The temperature input needs a dummy voltage source whose DC value in volts is interpreted as temperature in  $^{\circ}\text{C}$ . Set the temperature to be  $20^{\circ}\text{C}$ . How does the incremental resistance at  $5\text{V}$  change when the temperature is  $80^{\circ}\text{C}$ ?

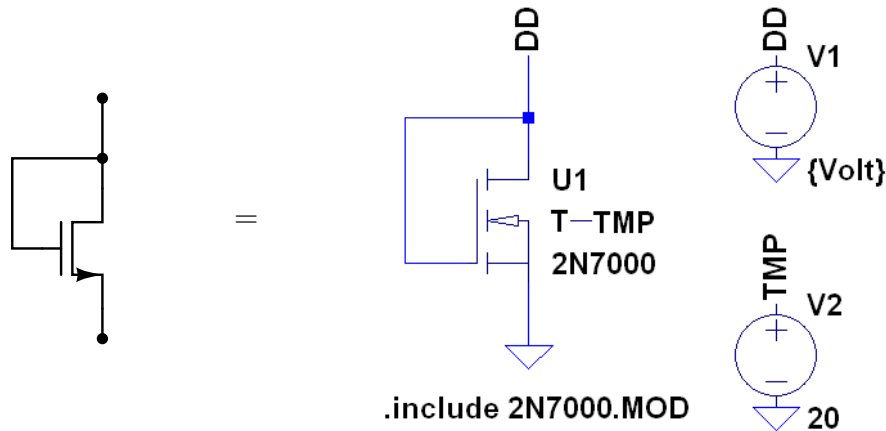


Figure 1.24: MOSFET in diode configuration (Problem 2).

3. The stability margin of a voltage follower driving a capacitive load depends on the capacitor’s value. Use LTSPICE to calculate *gain margin* and *phase margin* — useful criteria of the circuit’s degree of (in)stability — for this circuit for  $50\text{pF}$ ,  $1\text{nF}$ , and  $20\text{nF}$   $C_1$  capacitance (`voltage_follower.asc` file, see Figure 1.15). Fill out the following table (indicate the units).



C	$f_1 = f_{0dB}$	$G_{L1}$	$\phi_{L1}$	$f_2 = f_{-180^\circ}$	$G_{L2}$	$\phi_{L2}$	$G_M$	$\phi_M$	$f_0$
50pF		0dB				-180°			
1nF		0dB				-180°			
20nF		0dB				-180°			

Here,  $f_1 = f_{0dB}$  and  $f_2 = f_{-180^\circ}$  are the frequencies for which the loop gain  $G_L = 0dB$  and the loop phase  $\phi_L = -180^\circ$ , respectively.  $G_M$  is the gain margin:  $G_M = G_{L2} - G_{L1}$ .  $\phi_M$  is the phase margin:  $\phi_M = \phi_{L2} - \phi_{L1}$ .  $f_0$  is the frequency of oscillations you are observing in time-transient analysis in LTSPICE after a steep rise or fall of the input signal.

Note: in one of the earlier examples you have obtained Bode plot of the *closed loop gain* of this circuit, but to determine phase and gain margin you should find Bode plot of the *loop gain*. These are two distinct gains. Recall that the loop gain is determined from the signal gain traveling around the loop formed by the op-amp and the negative feedback. To find the loop gain, insert AC stimulus in the path of the negative feedback, while grounding the input of the voltage follower. See Figure 1.25. The loop gain is given by the expression  $-V(out)/V(in)$ , which you should plot after running an `.ac` analysis. Time-transient analysis on the original circuit is also necessary to determine the frequency of induced oscillations,  $f_0$ .

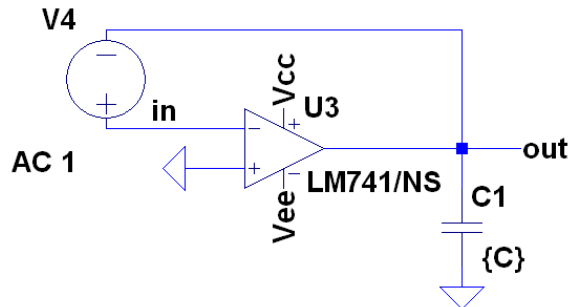


Figure 1.25: Setup to determine loop gain of the voltage follower driving a capacitor.

- Figure 1.26 shows a center-tapped transformer implementation in LTSPICE. Inductor component used for this circuit is `ind2`, which is found among the predefined components in LTSPICE. There are three inductors: L1 corresponding to the primary coil, and L2 and L3 corresponding to the secondary coil. Note SPICE directive `K1 L1 L2 L3 1` which defines *mutual* inductance with coupling coefficient  $K1=1$ . It means that all three coils fully induce each other (an analog to them being wound on an iron core with a very high permeability). The relationship between the voltages for a pair of coils with a coupling coefficient of 1 is  $|V_1/V_2| = \sqrt{L_1/L_2}$ . L1 is specified to be 10H.  $5\Omega$  in series with L1 represents the primary coil resistance (you can also specify this and other parameters by right-clicking on the inductors). Build a full-wave rectifier using this center-tapped

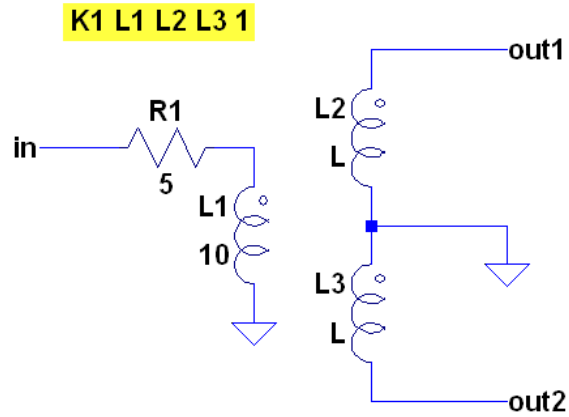


Figure 1.26: Center-tapped transformer in LTSPICE.

transformer that converts 120VAC 60Hz voltage to 10V DC driving 10k $\Omega$  load with 1% ripple. Show calculations for the component values. Then double-check your calculations by observing the output of your rectifier in LTSPICE. Use 1N4007 rectifier diode model found in the files (in SPICE\external\_components directory) available for download from the course web-site. Move 1n4007.txt file to the same directory as your schematic .asc file. Enter d1n4007 for Value in the diode's symbol and add .include 1n4007.txt statement. Finally, determine how much power is dissipated on average in all of the diodes.

---


## Optional Experiments Using LTSPICE

---

You will find a shortcut to **SPICE** directory on your lab computers. This directory contains symbols and model files for some of the components, which are not available in LTSPICE, but are used in the experiments outlined here. It is recommended that you create a new directory for each new experiment and copy only needed symbol and model files for inclusion into your schematic `.asc` file saved in the same directory. Please make sure you save your files to your own storage medium between lab sessions — the lab computers have a shared usage and any files you leave on their hard-drives will be eventually overwritten.



The main directory (**SPICE**) has two subdirectories: **external\_components** with component definitions, and **experiments** that contains circuit schematics files for use with some of the experiments outlined in Part 2. The description for each experiment will indicate whether additional files from **SPICE** directory may be needed.

Directory **external\_components** contains an additional folder named **dview** or digital view. It contains two “elements” useful for visualizing multiple digital traces overlapping in time without having to open a new plot pane each time another signal is being added to the plot. The two symbol files in that folder are **dview5.asy** and **dview10.asy**, each allowing up to 5 or 10 traces to be viewed simultaneously. Simply connect the nodes you wish to be viewed to **dview5** or **dview10** inputs and then view them by clicking on the corresponding outputs of the “element” with the probe . Each new trace viewed this way will be scaled to fit a single plot pane. As a result, the vertical axis reading does not correspond to the actual voltage of the signal. To use the digital view “elements”, copy both `.asy` and `dview.lib` files to your current directory. Insert the symbol to your circuit and add `.include dview.lib` SPICE directive.

## 2.1 Chapter 8 Experiments

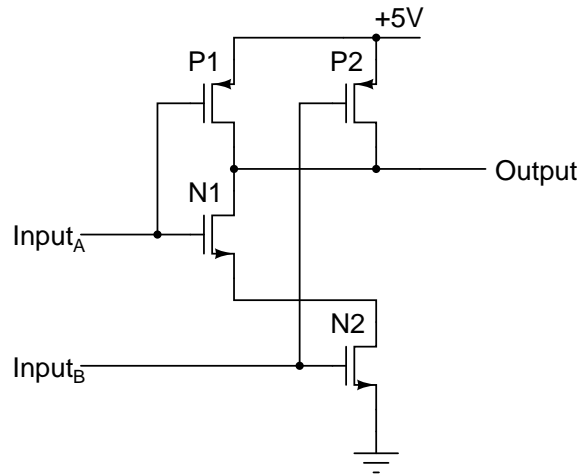


Figure 2.1: 2-input CMOS NAND gate.

**Exp (LTSPICE 8.1). 3-input CMOS NAND gate.** Figure 2.1 shows a 2-input complementary metal-oxide-semiconductor (CMOS) NAND gate. It consists of two n-channel MOSFETs, N1 and N2, connected in series, and two p-channel MOSFETs, P1 and P2, connected in parallel. This gate can be expanded to more inputs by appropriately adding additional pairs of complementary MOSFETs.

Expand this gate to a 3-input NAND and implement it in LTSPICE. Use default MOSFET transistor models associated with symbols `nmos` and `pmos` in LTSPICE. Fill out and verify the truth table. Indicate what state (ON/OFF) each of the transistors is in. See the table below. Ignoring switching glitches (current spikes during rapid changes of the digital signal state), what is the maximum power required to operate this gate? (Hint: to verify the truth table, you may find it useful to setup 3 square pulse voltage sources (0 to 5V) with frequency ratios 1:2:4 to drive inputs A, B, and C.)

Inp. A	Inp. B	Inp. C	Output	P1	P2	P3	N1	N2	N3
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0							
1	1	1							

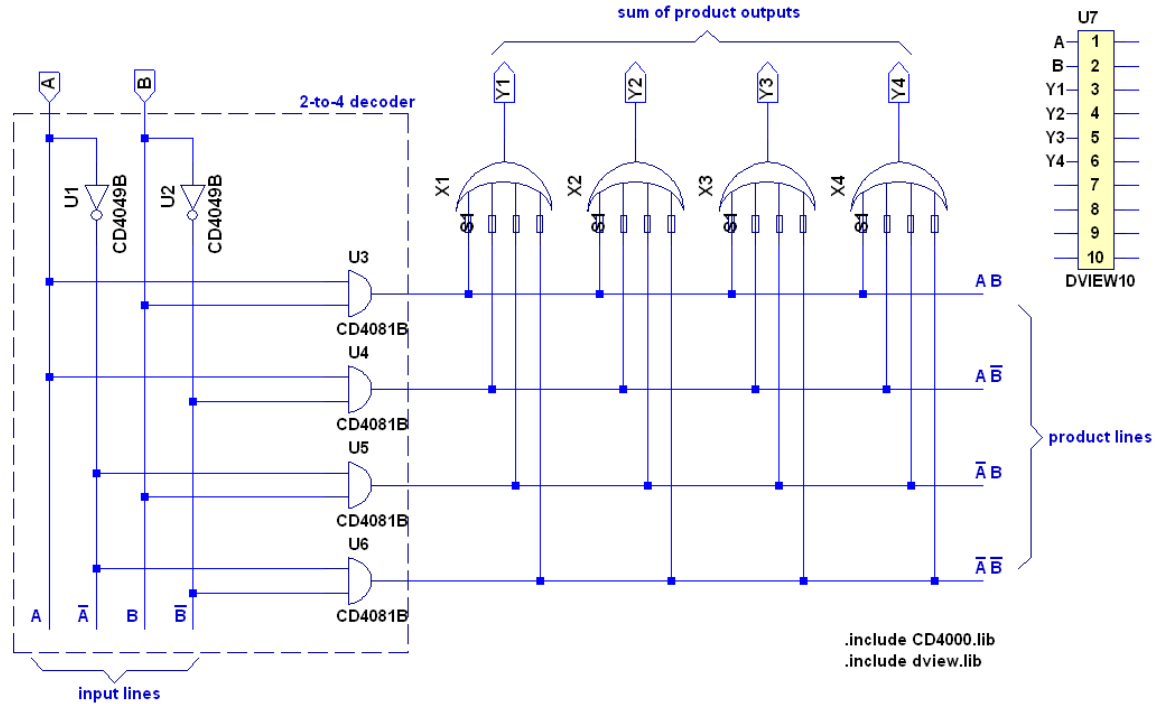


Figure 2.2: Programmable logic device.

**Exp (LTSPICE 8.2). Programmable Logic Device.** Programmable Logic Device (PLD) is an integrated circuit that contains a large number of gates (and other types of basic digital elements, such as flip-flops and registers) interconnected on a chip. A PLD can be configured by the user to perform an arbitrary logic function. Inside a Programmable Read-Only Memory (PROM), a type of PLD, many of the connections are fusible links which can be selectively destroyed using a special device called PROM programmer to achieve some desired functionality. Other types of read-only memory employ different mechanisms of connecting or breaking links allowing for Erasable Programmable Read-Only Memory (EPROM), i.e., with a possibility of resetting broken links and allowing to reprogram the device. One of the main advantages of a PLD is that it greatly reduces the number of ICs otherwise necessary to implement the same functionality, resulting in reduced printed board space, lower power requirements, and overall higher reliability.

Figure 2.2 shows an example of implementation for a simple PLD. It consists of an array of AND gates and an array of OR gates. Each input provides its negated version in addition to the signal itself forming *input lines*. These in turn are connected to AND gates to form so-called *product lines*. Each product line is connected to one of the inputs of the OR gates via a fusible link allowing for a summation of certain product terms depending on whether a particular fuse is broken or not. As a result, this device can be programmed to have an arbitrary truth table for the two inputs.

The example on Figure 2.2 has 2 input lines and 4 output lines. Implement this circuit in LTSPICE. To do so, copy the contents of SPICE\experiments\8.2 to your local directory. It contains: symbols for CMOS gates CD4049B (NOT), CD4072B (4-input OR), CD4081B (AND), and CD4000.lib SPICE library file with their definitions (also found in SPICE\external\_components\Digital\_CD4000 directory); a circuit block files (fuse4or.asy and fuse4or.asc) that implement 4-input OR gate with “fusible” links; a single-pole double-throw switch symbol spdt.asy and its model switches.sub (also available in switches directory in SPICE\external\_components); and a “component” to simultaneously view digital traces (dview10.asy and dview.lib), which can also be found in SPICE\external\_components\dview directory. Build the circuit using 2 NOT gates, 4 AND gates, and 4 fusible 4-input OR gates (use fuse4or, not CD4072B).

Right-click on one of the fusible 4-input OR gates to see how this subcircuit is implemented. The 4 switches represent “fuses”. By default, these “fuses” connect the OR gate to its inputs. A “fuse” can be “blown” by passing a parameter to the subcircuit. From your PLD circuit, right-click on one of the four OR gates, then enter, for example, S1=0 S3=0 in PARAMS field. A corresponding text will appear next to the gate’s symbol if you place a checkmark next to PARAMS. These parameters will “blow” S1 and S3 “fuses” producing 0 logic on the corresponding inputs of the OR gate, regardless of the input. The position of S1 input is shown in the symbol to help you identify how the input pins are connected to the corresponding “fuses”: S1, S2, S3, and S4.

Implement the following Boolean functions by “blowing” the appropriate “fuses”: A XOR B on Y1; NOT A on Y2 (regardless of B); A OR B on Y3; A NOR B on Y4. Write down each of these functions as they correspond to their implementation in the circuit (i.e., in terms of sums of products). Supply square 0-5V pulses on inputs A and B (1:2 frequency ratio) to verify the truth table for each of the 4 functions by viewing the corresponding outputs.

## 2.2 Chapter 9 Experiments

**Exp (LTSPICE 9.1). Master-slave RS and JK flip-flops.** Figure 2.3 shows an implementation of a negative-edge triggered RS flip-flop. It employs a master-slave configuration, in which two identical clocked RS latches are used. An individual clocked latch performs as a regular RS latch when the clock input CLK is logic 1. If CLK level is logic 0, S and R inputs have no effect on the latch, and it retains its old value.

Note that the two CLK inputs are connected via an inverter. This ensures that the two sections will be enabled during opposite half-cycles of the clock signal. This is key to realizing an edge-triggered flip-flop.

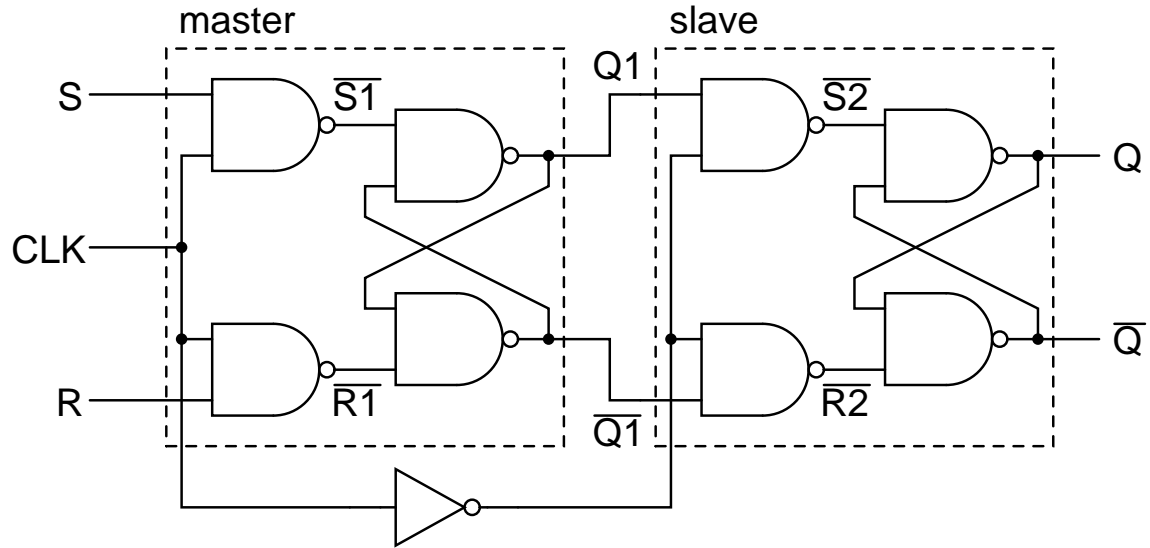


Figure 2.3: Master-slave configuration for RS flip-flop.

Consider what happens when CLK input starts out at logic 0 level. The S and R inputs are disconnected from the input (master) latch. Therefore, any changes in the input signals cannot affect the state of the final outputs. When the CLK signal goes 1, the S and R inputs are able to control the master latch. However, at the same time, the inverted CLK signal applied to the slave latch prevents the master latch from having any effect on it. This way any changes in the R and S input signals are propagated to  $Q1$  and  $\overline{Q1}$  by the master latch while CLK is at 1, but are not reflected in the  $Q$  and  $\overline{Q}$  outputs.

When CLK falls back to 0, the S and R inputs are again isolated from the master latch. At the same time, the inverted CLK signal now allows the current state of the master latch ( $Q1$  and  $\overline{Q1}$ ) to reach the output latch. Thus, the  $Q$  and  $\overline{Q}$  outputs can only change state when the CLK signal falls from logic 1 to 0. This is known as negative (falling) edge triggered RS flip-flop.

Construct this circuit in LTSPICE using 74HCT00 NAND and 74HCT04 inverter gates (found in SPICE\external\_components\Digital\_74HCTxxx). You can draw wires diagonally in LTSPICE by holding down **Ctrl** key. 74HCT is a popular digital family that uses CMOS technology but is also compatible with TTL signal levels. Be sure to include the appropriate library file 74HCT.LIB. Supply voltage signals (0-5V levels) to S, R, and CLK, as shown in Figure 2.4 (use PWL option to specify piece-wise linear functions for S and R inputs — accessible after right-clicking on the voltage source, then pressing *Advanced* button).

To avoid the circuit's going into a "race" (rapid oscillations of the outputs with steady inputs), supply the following initial conditions:

```
.ic V(Q)=0 V(_Q)=5 V(Q1)=0 V(_Q1)=5
```

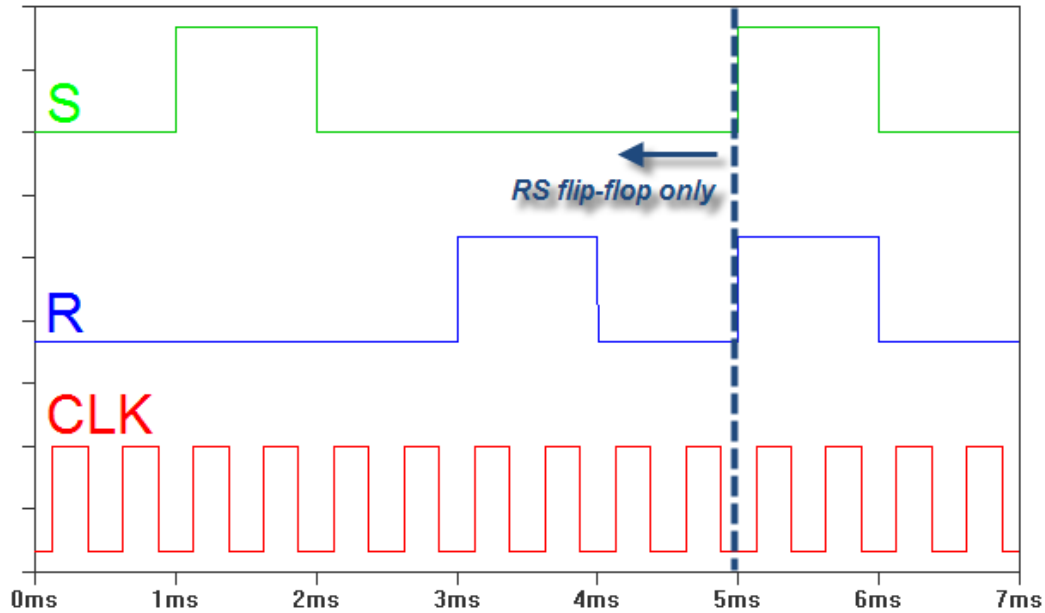


Figure 2.4: Signals (0-5V range) for LTSPICE Exp. 9.1.

where Q,  $\bar{Q}$ , Q1, and  $\bar{Q}1$  are the node labels for Q,  $\bar{Q}$ , Q1, and  $\bar{Q}1$ , respectively. This sets both RS latches to 0 initial state.

Fill out the following table:

S	R	CLK	Q
0	0	⌋	0
1	0	⌋	
0	0	⌋	
0	1	⌋	
0	0	⌋	

You can also send this circuit into a “race” by supplying  $S = R = 1$ . Such undefined behavior is an intrinsic limitation of RS flip-flops. You can turn this circuit into a JK flip-flop that does not have such a problem. *First, you need to replace the two NAND gates at the input with two 3-input NANDs.* Use 74HCT10 3-input NAND gates. To deal with the problematic  $S = R = 1$  condition, one should discretely disable this condition from propagating into the master latch while keeping all other combinations unaltered. The basic idea is that one has to disable S input when and only when Q is 1 (S input’s job is done, no need to reassert it continuously), and also to disable R input when  $\bar{Q} = 0$  (R input’s job is done, no need to reassert it continuously). One can do so by connecting the remaining 3rd input of each 74HCT10 NAND to the appropriate output.

Turn your original circuit into a JK flip-flop. Verify that the circuit works by running it with inputs as shown in Figure 2.4.



Fill out the following table:

S	R	CLK	Q
0	0	⌋	0
1	0	⌋	
0	0	⌋	
0	1	⌋	
0	0	⌋	
1	1	⌋	
1	1	⌋	
0	0	⌋	

**Exp (LTSPICE 9.2). Domino effect in a ripple counter.**

Build two different 8-bit counters using the appropriate HCT family components: an asynchronous one using 8 D flip-flops 74HCT74 and a synchronous one using 2 4-bit binary counters 74HCT161. As usual, copy the symbols and the corresponding library file from SPICE\external\_components\Digital\_74HCTxxx to your local directory. Supply a clock CLK signal (0-5V) of 100kHz frequency to both counters. Name synchronous counter bits S0 through S7, and asynchronous counter bits A0 through A7.

To visualize a difference between the two counters, you can perform a digital-to-analog “conversion” on their outputs: convert the binary representation of the counter’s state into a signal of varying amplitude that can be visualized. One simple way to accomplish it in LTSPICE is to add a special voltage source to your circuit, a so-called *behavioral voltage source*. The output of such a source in LTSPICE can be programmed with an arbitrary function. To insert this special component, select **bv** symbol from **Select Component Symbol** window after pressing **F2**. Insert two such sources (one for each counter). Right-click on each of the components and enter the following strings in their respective **Value** fields:

$$V=(V(S0)+2*V(S1)+4*V(S2)+8*V(S3)+16*V(S4)+32*V(S5)+64*V(S6)+128*V(S7))/255$$

and

$$V=(V(A0)+2*V(A1)+4*V(A2)+8*V(A3)+16*V(A4)+32*V(A5)+64*V(A6)+128*V(A7))/255$$

If the counter bit outputs are strictly 0 or 5V, these functions will convert the binary values represented by these bits (digital) to a voltage in a 0-5V range (analog).

Run the simulation on these two counters for 4ms. Plot analog signals obtained from the digital-to-analog conversion of the outputs for each case. You should see a sawtooth pattern in both cases. How does that pattern differ between the synchronous and asynchronous versions? Zoom in on the largest “glitch”, which occurs half-way between 0 and 5V. Write down binary values that the counter goes through during that “glitch” and their corresponding voltage values.

## 2.3 Chapter 10 Experiments

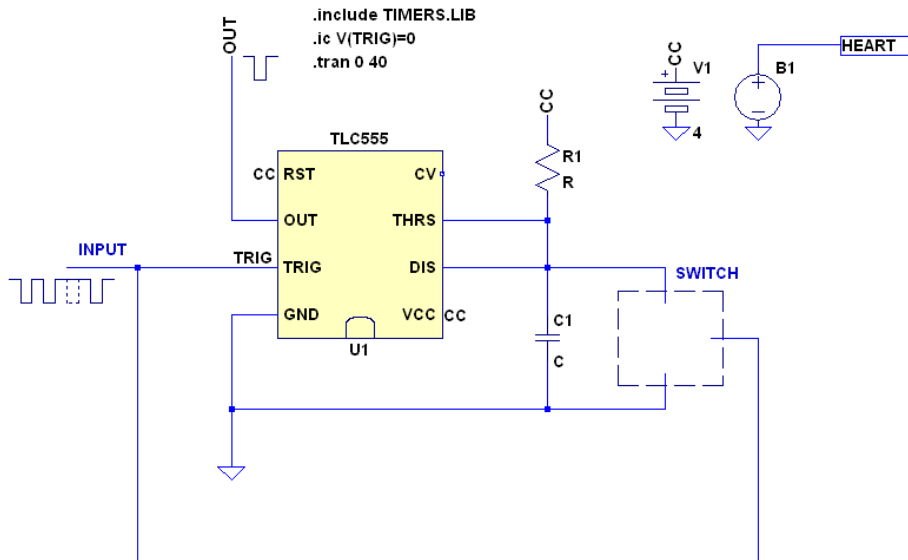


Figure 2.5: Circuit for LTSPICE Exp. 10.1: Missing Pulse Detector.

**Exp (LTSPICE 10.1). Missing pulse detector.** Figure 2.5 shows an incomplete circuit for a missing pulse detector. This circuit is found in `SPICE\experiments\10.1` directory. Copy its contents to your local directory. This basic circuit utilizes a 555 timer in a one-shot configuration. Used as is, this circuit will generate pulses of a given duration (triggered by a negative edge on the input) as determined by  $R1$  and  $C1$  values (left unspecified in the circuit). A falling edge of the input signal, when its level drops below  $V(CC)/3$ , produces  $S = 1$  logical signal on the internal RS latch. As a result,  $OUT = 1$  logic, the internal transistor switch is OFF leaving DIS pin essentially disconnected (floating), and  $C1$  capacitor is charging up through  $R1$  resistor. Refer to the 555 timer internal schematic found in the Lab Manual. Any subsequent falling edge pulses at the input will not have any effect on the internal latch, and therefore on  $OUT$  and  $DIS$  pins, until the THRS voltage becomes larger than  $2V(CC)/3$  generating  $R = 1$  digital level on the RS latch (LTSPICE symbol for 555 timer uses THRS for the same input designated in the Lab Manual as  $2/3$  COMP). This puts the internal transistor switch in ON state, discharging the capacitor  $C1$ .  $OUT$  becomes logic 0, and the RS latch is again sensitive to the falling edge of TRIG input.

A missing pulse detector essentially acts as a retriggerable one-shot. With each subsequent pulse arriving within the duration specified by  $R1$  and  $C1$  values, the capacitor  $C1$  is discharged through the addition of an external switch (shown as a dashed box in Figure 2.5). While the internal transistor switch remains OFF, the external switch should be ON each time TRIG input goes low. This prevents the

voltage capacitor and the THRS pin from crossing  $2V(CC)/3$  threshold, and keeps OUT at 1 logic level. When the TRIG input goes high, both switches are OFF, and if TRIG stays high long enough to allow C1 capacitor to charge to the  $2V(CC)/3$  level, the output OUT will go low ( $OUT = 0$ ). Any subsequent low input at TRIG will again change OUT to be logic 1.

Complete the circuit to make it a functioning missing pulse detector. The initial condition specified in the file ensures that the internal transistor switch is OFF prior to the simulation. Use an appropriate BJT transistor (2N3906 or 2N3904 — only one is correct!) to form the external switch as explained above.

HEART output represents a signal derived from the heart beat of an ailing patient. Arrange for the circuit to detect a missing heart beat if the input stays low (no heart beat signal) for more than about 2 seconds. Note: the missing pulse detector explained above generates  $OUT = 0$  when the input stays high longer than it's supposed to. Therefore, you have to modify the circuit to detect a missing pulse when the input stays low. Feel free to use any of the gates from 74HC digital family (found in SPICE\external\_components\Digital\_74HCxxx). This is a modern replacement for CD4000 CMOS family of gates. It features much higher speed than CD4000, higher sink and source currents (20mA), and can work with the power supply as low as 2V.

Next, use a D flip-flop 74HC74 to produce an ALARM signal. This signal should go logic 1 when a first missing heart beat is detected, and stay this way until manually reset. Document the schematic in your log book and understand how it works.

Imagine this circuit is to be powered by 3 AA batteries, 1.33V each. Each battery has an estimated 500mA-hours lifetime before the battery power starts to droop off significantly (a conservative value). Estimate when one would have to replace the batteries if the device is to stay on all the time.

**Exp (LTSPICE 10.2). Wailing alarm.** LTSPICE has a capability to read and write complicated waveforms from and to .wav files. In this experiment you will use this capability to hear the actual output of electronically generated alarm sound.

Figure 2.6 shows a circuit that generates a wailing alarm siren sound of the type used in some police vehicles. The sound frequency is being modulated by changing the control voltage (CV) input of the timer on the right. A loudspeaker would be connected to the output  $V_{OUT}$  along with a series capacitor.

Implement this circuit in LTSPICE. What would be the frequency of the sound if the transistor is to be removed from the circuit? What would be the value of CV for the 555 timer on the right? Estimate these quantities and then compare to what you observe in simulations. Assume a three-100k $\Omega$  voltage divider inside the

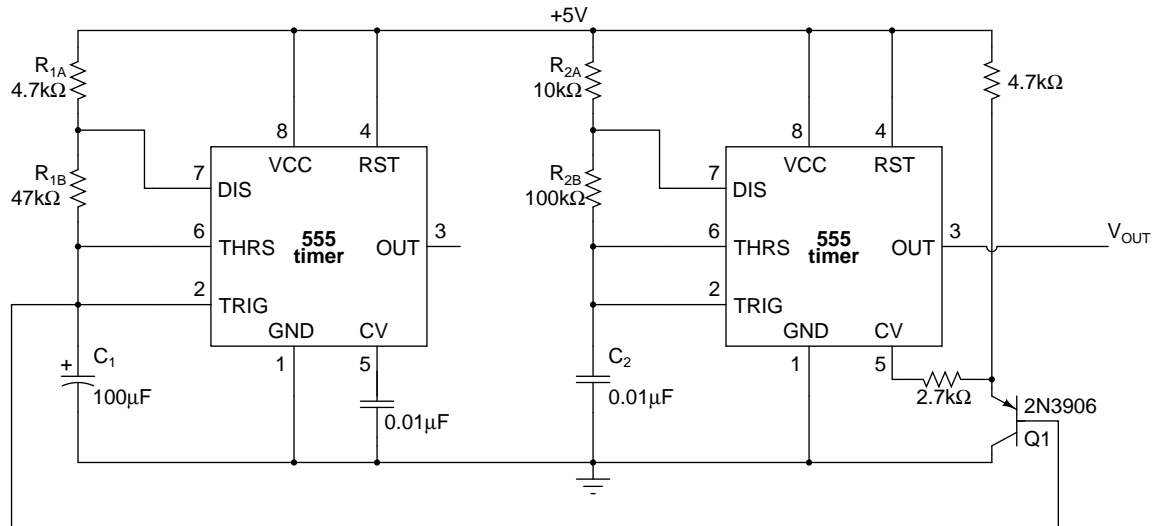


Figure 2.6: Circuit generating wailing alarm siren sound.

555 instead of  $5k\Omega$ . With the transistor in place, note the maximum and minimum voltages at CV. When is the pitch higher: when CV is high or low?

To generate a .wav file, add the following SPICE directive to your circuit:

```
.wave "somename.wav" 16 22.05K V(OUT)
```

This line instructs LTSPICE to write the waveform of V(OUT) to a file `somename.wav` with 16 sampling bits at the sampling frequency of 22.05kHz (more on what that means later in the course). The .wav analog to digital converter in LTSPICE has a full scale range of  $\pm 1V$ . For best results the signal to be written to a .wav file has to match that range, otherwise, the converted signal will end up being truncated.

The output of the circuit is in 0-5V range. Build a simple voltage divider at the output of the circuit to transform the signal to  $\pm 1V$  range (you will also need a voltage source to shift the signal). Move OUT node label to the new output location with  $\pm 1V$  signal (in real life you could drive a speaker connected through a series capacitor directly from the output in Figure 2.6). Simulate the circuit for about 15s and listen to the sound that it produces by playing the .wav file (it will appear in the same directory as your .asc file with the schematic). Try few different values of C1 in the range from 1 to  $100\mu F$ . Observe the change in the sound by recording and playing different .wav files.

## 2.4 Chapter 11 Experiment

**Exp (LTSPICE 11.1). Signal encryption using a sequence of pseudo-random numbers.** Random numbers that lack any order have many uses in computational

physics, statistics, cryptography, etc. Efficient generation of random numbers on deterministic systems such as computers is not a trivial task. One common way to implement a random number generator is through the use of a so-called Linear Feedback Shift Register (LFSR). By tapping output bits in a shift register at specific places and feeding those bits through a tree of XOR-gates back into the serial data input line, a LFSR of  $N$  bits in length can generate a sequence of  $2^N - 1$  pseudo-random values. This sequence is called *pseudo-random* because it will repeat itself after  $2^N - 1$  clock pulses, e.g., an 8-bit LFSR will generate 255 long sequence, whereas 32-bit LFSR will start to repeat itself after 4,294,967,295 clock ticks. A basic LFSR of 8 bits is shown in Figure 2.7. The shift register — a series-connected group of D flip-flops — uses XOR feedback to scramble the basic data input bit. XOR gates must be connected to appropriate taps in order to generate the longest sequence of numbers before they repeat. The taps for maximum-length pseudo-random sequences of LFSRs of a certain number of bits are given in Table 2.1. Other tap configurations are possible to achieve sequences of maximum length. When more than 2 inputs are specified, XOR operation can be achieved by combining 2-input XORs, e.g., in the case of 4-input XOR:  $\text{XOR}(A, B, C, D) = \text{XOR}(\text{XOR}(A, B), \text{XOR}(C, D))$ .

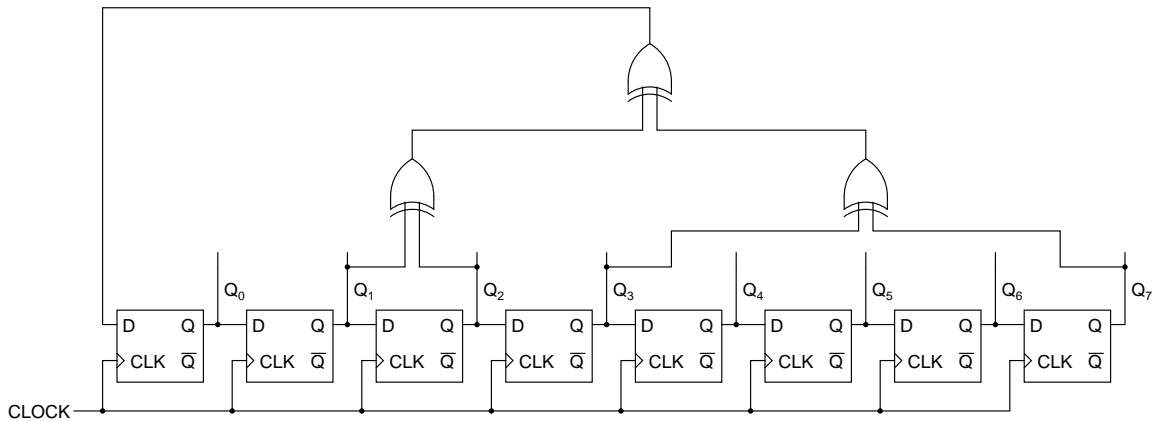


Figure 2.7: 8-bit shift register in a pseudo-random number configuration.

Note that if all stages of the shift register are logic 0, the pseudo-random number generator will get “stuck” with all zeros since  $\text{XOR}(0, 0) = 0$ . To prevent this from happening, at least some of D flip-flops have to be “preloaded” with nonzero D inputs on the first clock tick. The value corresponding to the initial binary bit pattern  $(D_N D_{N-1} \dots D_1 D_0)_2$  can be any number as long as it is not 0. This number is known as a *seed* of the pseudo-random number generator. The sequence of generated numbers will always be identical if the same seed is used more than once.

Depending on a particular application, the approach to seeding the LFSR will be different. For example, if a “truly” random distribution is desired, a “random” seed would be ideal (of course if a truly random seed is available in the first place there would be no need for a random-number generator!). For example, a seed can

Table 2.1: Shift register taps for maximum-length pseudo-random sequences.

$N$	Loop length	Tap numbers
4	15	(0,3)
8	255	(1,2,3,7)
12	4095	(0,3,5,11)
16	65,535	(1,2,4,15)
20	1,048,575	(2,19)
24	16,777,215	(0,2,3,23)
28	268,435,455	(2,27)
32	4,294,967,295	(1,5,6,31)

be chosen from a quickly varying real-time clock of the system. You can read the current time, mask off some portions of its bit pattern and use that as a seed. With an analog-to-digital converter, a seed can be derived from AC power line voltage, some sensor position or even amplified intrinsic noise from a Zener diode (a common practice in cryptography). The random sequence so obtained is further manipulated (e.g., converted to an integer and/or scaled) to form a desired random result.

In other cases, however, it is precisely the deterministic nature of pseudo-random numbers that is of significance. One example is encryption. Here, the basic problem is how to transmit some information in a way that it is made unreadable to anyone except for a designated reader possessing a special knowledge (a password or a key). Encryption stages involve a cypher (used to scramble the information), and a decipher, which makes the information readable again (decryption).

For example, a plain-text message can be viewed as a long string of bits. It can be encrypted using a predefined sequence of numbers (bits) that may otherwise appear random. Before each bit is transmitted, it is XOR'ed with the output bit of the pseudo-random number generator. Upon receipt, the bit is decrypted by using XOR again with the output of an *identical* pseudo-random number generator with the *same seed*. In this case, the seed becomes the *encryption key*, which must be known by both the sender and receiver. A transmitted message may appear totally random or scrambled, and it can be sent without fear that it will be read by an “enemy” (unless the enemy has taken PHYS3360/AEP3630 and has somehow figured out the key). Using longer pseudo-random sequences makes it less likely that the enemy will be able to crack the encryption through some brute-force technique.

Because LTSPICE is not well suited to encode/decode a plain-text into/from bit streams, you will decipher a scrambled digital 8-bit version of an analog signal. A pair of analog signals is digitized and sent into a hostile territory in hopes of reaching a “friend” with the knowledge of the encryption key. These two signals, once plotted against each other, represent a secret drawing (parametric curve) intended for the friend. One of the signals was encoded using an 8-bit pseudo-random number generator. To see the drawing one needs to have both signals available to him, therefore,

encrypting only one signal is sufficient to scramble the information being transmitted.

There are several subtleties to this simple encryption method. First, in addition to having to use *identical* pseudo-random number generator, the clocks used to encrypt and decipher the signal must be *identical* as well. The sequence of pseudo-random numbers must overlap exactly with the encrypted signal, which inevitably brings up the issue of synchronization. As such, the technique described here can be reliably used only on relatively short messages (as limited by the clock accuracy). Second, even if the encrypted signal and the proper sequence of pseudo-random numbers overlap, special attention is still required for the edges as determined by the clock signal. For example, the propagation delay in XOR gates used for decryption may cause “glitches” at the edges of the signal once converted to analog form (similar to the domino effect in the case of asynchronous counter). One simple way to fix this problem is to pass all the bits of the signal as a final stage through D flip-flops clocked with a timing signal edge sufficiently delayed with respect to the master clock active edge to ensure that all transients have died out by that time. For example, if the shift register is triggered by the positive edge of the master clock, you can use the negative edge of the clock to clean up the signal. Doing so effectively delays the signal. If there is another signal which must be synchronous with the first, it too should be passed through D flip-flops triggered by the same clock signal.

Decipher a secret message which was encrypted using a sequence of pseudo-random 8-bit numbers. Copy the contents of SPICE\experiments\11.1 to your local directory. The files include some relevant devices from the 74HC digital family (symbols 74hc04.asy, 74hc86.asy, 74hc374.asy, and the library file 74hc.lib). There are two behavioral components to be used in the circuit: an 8-port buffer to boost the level of digital signals to 5V (buf8.asy symbol and buf8.asc schematic files), and a component which performs digital (8-bit) to analog ( $\pm 1V$  default range) conversion (dac8.asy file and dac8.asc schematic files). These components are called behavioral because they only act like their real-life counterparts, while they are implemented using behavioral voltage sources in LTSPICE, as opposed to electronic components, such as transistors, capacitors, etc. The directory also contains the schematic file encryption.asc and two traces containing the “transmitted” information: xtrace\_encrypted.wav and ytrace.wav. Open encryption.asc file with LTSPICE. See Figure 2.8.

There are several areas in this schematic (partially filled): Digital Input, Clock Signal, Pseudo-Random Number Generator (RNG), Cypher/Decipher, Edge Clean-Up, and Digital to Analog. In the Digital Input area, you will find two traces in 8-bit digital form loaded from the .wav files: encrypted x-trace and y-trace signal. When y-trace is plotted against x-trace, it will display the secret drawing. You can see that each bit is imported from an appropriate channel in .wav file. Note: LTSPICE can read and write .wav files which may not be playable for sound because of the number of channels being used, the number of sampling bits or

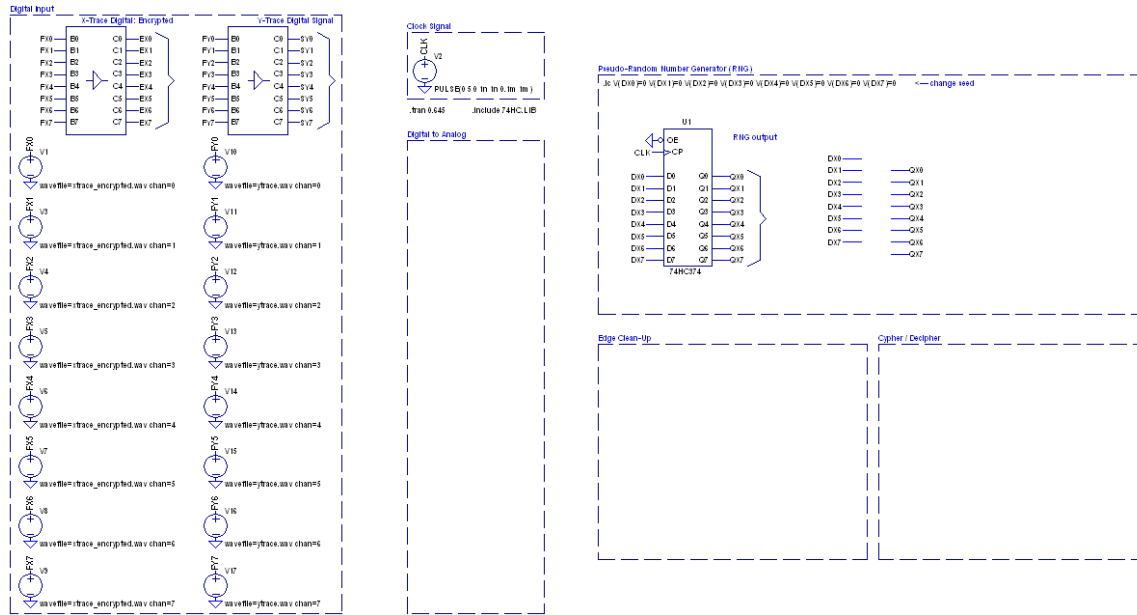



Figure 2.8: Unfinished schematic for LTSPICE Exp. 11.1: Signal Encryption.

the sampling rate. These files, however, are useful to store complicated signals for LTSPICE simulations.

The two signals in 8-bit digital form are available at nodes EX0 (LSB) through EX7 (MSB) for the encrypted x-trace, and SY0 through SY7 for the y-trace. To visualize these traces, place two digital to analog converters (dac8) in Digital to Analog area, one for each trace. Connect appropriate inputs of DACs to the 8 bits of each signal. (Hint: use node labels for connections!) Plot the resultant analog signals against each other and observe a scrambled pattern. The job is now to decipher the x-trace represented by  $(EX_7EX_6EX_5EX_4EX_3EX_2EX_1EX_0)_2$  binary value.

First, construct a LFSR random number generator. In Pseudo-Random Number Generator (RNG) area, connect the pins of 74HC374 — a positive edge-triggered octal D flip-flop — to form a shift register. Use small value resistors (e.g.  $1\Omega$ ) to connect the appropriate pins on the right of the 74HC374. The reason for using small resistors is just a matter of convenience: LTSPICE allows only a single label per a set of nodes connected with wires, whereas we would prefer to address the input and output digital pins of the flip-flop by different labels: DX0 (LSB) through DX7 (MSB) and QX0 through QX7, respectively. Verify that the shift register works by supplying logic 1 (5V) to DX0 (use the .ic line in the schematic) and observing the value propagating along the output bits. Note: the full simulation loading entire trace signals may take several minutes. To verify that various subcomponents work properly, it is sufficient to simulate the circuit only for a small fraction of the



entire time span. This can be accomplished through pressing Halt icon  after the simulation has been started. Next, setup XOR gates as shown in Figure 2.7 or given in Table 2.1. Observe that the LFSR is working by supplying 1 logic to one of DX inputs. You may hook up QX outputs of RNG to DAC and observe the analog representation of the output.

Second, build a decipher by XOR'ing the bits of the RNG output QX and the encrypted signal EX.

Third, perform edge clean-up by passing *both* decrypted x-trace and y-trace bits through a pair of 74HC374's triggered off the negative edge of CLK clock signal. Then connect both 8-bit digital outputs to DACs.

You are now ready to decrypt the secret message. An instruction in place of a key simply states: “last two digits of your Alma Mater founding date”.

Use the hint provided to decrypt the message. Convert the last two digits in the founding date of the University to binary form and use it in place of the seed for LFSR (.ic line in the schematic). Display y-trace vs. decrypted x-trace. You may choose *Mark Data Points* from *Plot Settings* drop-down Menu to better see individual points on the plot.

The operations of encryption/decryption are reversible in this case. E.g. you can use this circuit to encrypt the signal with your own choice of the seed such as the year of your graduation.

## 2.5 Chapter 12 Experiments

**Exp (LTSPICE 12.1). Dual-Slope Analog to Digital Converter.** Integrating A/D converters (ADCs) employ a very different principle compared to converters that require a “frozen” input, such as a flash or successive approximation ADC. As their name implies, the output of integrating ADCs represents the integral or average of an input voltage over a fixed period of time. This results in a good immunity to high frequency noise (relative to the measurement period), albeit at the expense of a reduced speed of the A/D conversion. Integrating ADCs are ideal for use in measuring devices, such as digital voltmeters and panel meters. In fact, the overall usage of integrating converters exceeds the combined total of all other conversion methods.

A particular representation of this class of ADCs is a dual-slope converter. Figure 2.9 illustrates its working principle. Dual-slope A/D converter involves a 2-stage process. First, S1 switch is closed to ensure capacitor C is discharged. The counter is reset. S2 switch is then connected to  $V_{in}$  input voltage about to be measured.  $V_{in}$  is integrated for a *fixed time*  $t_m = m/f_0$ , where  $m$  is the number of clock ticks allowed

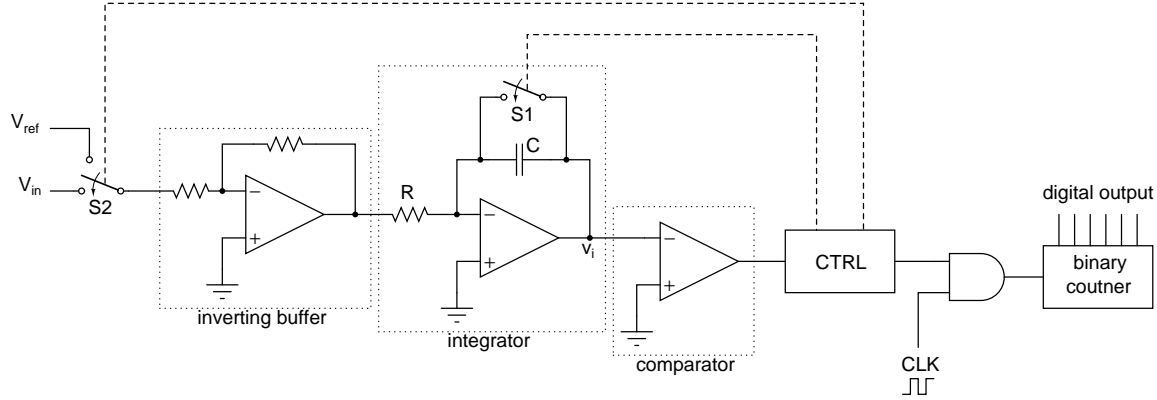


Figure 2.9: Dual slope A/D converter.

for a measurement, and  $f_0$  is the reference clock frequency. The voltage at the output of the integrator after time  $t_m$  is then given by

$$v_i = \frac{1}{RC} \int V_{in} dt = \frac{t_m}{RC} \langle V_{in} \rangle.$$

For the second stage, S2 switch is thrown immediately after time  $t_m$  to a well controlled reference voltage  $-V_{ref}$  of the opposite sign to  $V_{in}$  (alternatively, one of these two signals can be connected to the integrator via an inverter). At the same time the counter starts to count. After some time  $t_r$ ,  $v_i$  integrates back to zero, the comparator goes HIGH stopping the counter. The counter reading represents the digital value of  $\langle V_{in} \rangle$ . The slopes of the integrator in these two stages are proportional to the voltage supplied, namely  $V_{in}$  and  $-V_{ref}$ . Hence the name of the method. So, the two time periods, the fixed measurement time  $t_m$  when S2 switch is connected to  $V_{in}$ , and the integrator's capacitor discharge time  $t_r$  from the reference voltage  $-V_{ref}$  are related by

$$\frac{t_m}{RC} \langle V_{in} \rangle = \frac{t_r}{RC} V_{ref}.$$

If  $r$  is the number of clock ticks counted by the counter during the discharge time of the capacitor,  $t_r = r/f_0$ ,  $V_{in}$  is simply given by

$$\langle V_{in} \rangle = \frac{r}{m} V_{ref}.$$

This is an elegant result. The digital representation of the measured voltage ends up being independent from the integrator's time constant  $RC$  and the clock frequency  $f_0$ ! This means the design is extremely forgiving with respect to tolerances of the component values. As a drawback of the dual-slope converter, one may point out the long time required for a single A/D conversion, which ends up being at least  $(r + m)/f_0$ . Nevertheless, a good fraction of this time is used to *average* the input signal  $V_{in}$ , improving the signal-to-noise ratio of the measurement.

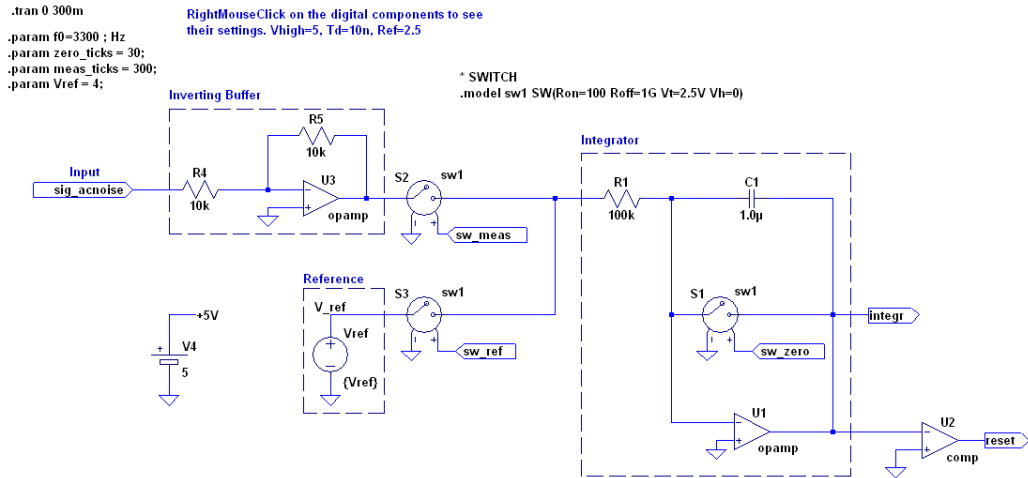


Figure 2.10: Circuit for LTSPICE Exp. 12.1: Dual-Slope A/D Converter.

Copy the file from SPICE\experiments\12.1 to your local directory. This file contains an implementation of a dual-slope ADC. It uses several parameters: `zero_ticks` refers to how many clock cycles the capacitor stays shorted out at the beginning of the measurement; `meas_ticks` specifies how many cycles the integrator is to be connected to the (inverse) input voltage. The clock frequency `f0` and the reference voltage `Vref` specify additional parameters for the ADC. Spend some time familiarizing yourself with how this circuit works. Plot various switch voltages: `sw_zero`, `sw_meas`, and `sw_ref` versus time. The circuit contains several sample input signals: +3V DC voltage (`sig_clean`), 3V with 1V 60Hz AC signal superimposed on top (`sig_acnoise`), and 3V signal with some white noise of 2V peak-to-peak amplitude (`sig_whitenoise`). Connect `sig_acnoise` to the input of the ADC, and observe the voltage at the input and output of the integrator. The circuit also contains a gated counter implemented from 8 D flip-flops. View the voltage at the counter's node `gated_clk` during the various stages of A/D conversion.

Next, connect the 3V input signal `sig_clean`. The digitized output of the counter is converted back to analog domain to ease its visualization. View the voltage on `sum_cleaned` node. Note that the output had to be cleaned to get rid of the edge "glitches" (compare this to the unrefined voltage signal at `sum` node). Change values of `R`, `C`, and `f0` by 10% to double-check that the output of the dual-slope converter does not depend on these values.

Connect the noisy signal `sig_whitenoise` to the converter's input. Find the RMS value of the input signal and compare it to the error of the measured voltage (relative to 3V).

Finally, connect `sig_acnoise` back to the ADC input. Note the error in the

measured voltage relative to 3V. A certain choice of  $f_0$  frequency can be used to totally reject 60Hz AC line noise. Find the expression for such frequency values. (Hint: it should contain both  $m$  and 60Hz). Verify that the noise is fully suppressed when you assign  $f_0$  a value from your formula.

**Exp (LTSPICE 12.2). Noise analysis in LTSPICE.** Noise comes in many ways and forms. By noise we understand any unwanted electrical disturbance. Such disturbances can be classified either as intrinsic noise or as interference. Interferences are introduction of *external* sources of noise or other signals to the electric circuit through various coupling mechanisms, such as mechanical, electromagnetic, etc. These types of noise can be largely suppressed through a proper design, shielding techniques, and so on. Intrinsic noise, on the other hand, is a property of individual components, which cannot be completely eliminated. Understanding the origin of different types of noise, their identification, characterization, and subsequent mitigation often requires considerable knowledge and experience. Some additional information is available in the following books [5, 6, 7]. Here we will briefly touch on intrinsic noise types and use `.noise` analysis available in LTSPICE to estimate a noise figure of a simple common emitter amplifier.

The following three types of intrinsic noise are being considered: i) Johnson (thermal) noise, ii) shot noise, and iii) flicker noise.

Johnson noise is an intrinsic noise, which arises due to thermal fluctuations of the number of current carriers in two halves of a resistor. Any resistor, no matter how expensive, displays this type of noise. The rms *voltage* that appears on the resistor  $R$  is given by

$$v_{n,rms} = \sqrt{4k_B T R \Delta f},$$

where Boltzmann’s constant  $k_B = 1.38 \times 10^{-23} \text{J/K}$ ,  $T$  is the resistor’s temperature in Kelvin, and  $\Delta f$  is the measurement bandwidth (or the response of the circuit) in Herz. For example, Johnson noise for  $R = 20\text{k}\Omega$  in audio frequency range ( $\Delta f \approx 20\text{kHz}$ ) at  $T = 300\text{K}$  is  $v_{n,rms} = 2.5\mu\text{V}$ . Johnson noise power spectrum (power per unit bandwidth) is independent of frequency, and such types of noise are oftentimes referred to as “white” (various frequencies are equally represented). Because of the  $\propto \sqrt{\Delta f}$  dependance of the voltage on the bandwidth for “white” noise, it is common to specify this and other types of noise as *voltage spectral noise density* in units of  $\text{V}/\sqrt{\text{Hz}}$ . To find the rms voltage equivalent, one has to multiply the spectral noise density by the square root of the bandwidth (provided the spectral noise density is a constant).

The shot noise appears due to discrete charge of the electron, and the corresponding statistical fluctuations in current (also known as “rain on a tin roof” noise). Poisson statistics tells us that fluctuations in a stochastic (random) quantity, such as a number of radioactive decays per second, number of cars in traffic passing an observer per unit time, etc., are proportional to the square root of the mean number of events. Similarly, the rms *current* fluctuation due to shot noise is proportional to the square root of the signal current  $i_s$ , which is given by

$$i_{n,rms} = \sqrt{2ei_s \Delta f}.$$

Here  $e = 1.6 \times 10^{-19} \text{C}$  is the electron’s charge and  $\Delta f$  is the measurement bandwidth. Similar to Johnson noise, shot noise has a “white” spectrum.

Flicker noise or “1/f” noise has power spectrum that follows 1/f trend, dominating the noise spectrum at low frequencies. Unlike the two other types of intrinsic noise, flicker noise does not have a single physical mechanism behind it, but is instead a cumulative term that encompasses a number of noise generating mechanisms, such as carrier density fluctuations in semiconductor devices. Flicker noise dominates at low frequencies over Johnson and shot noise, and it depends on such properties of the component’s material as purity, dopant levels, etc.

To characterize noise magnitude relative to the signal’s strength, it is common to use *signal-to-noise ratio* — a ratio of power contained in the signal over that of the noise:

$$\text{SNR}_{\text{dB}} = 10 \log_{10}(P_s/P_n) = 20 \log_{10}(V_s/V_n).$$

Another parameter being used (especially in RF circuitry) is *noise figure*, which specifies how much *additional* noise is being introduced by the circuit ( $\text{NF}_{\text{dB}} > 0$  always):


$$\text{NF}_{\text{dB}} = 20 \log_{10} \frac{(V_s/V_n)_{\text{in}}}{(V_s/V_n)_{\text{out}}} = \text{SNR}_{\text{dB}}^{\text{in}} - \text{SNR}_{\text{dB}}^{\text{out}}.$$

SPICE (and LTSPICE) provides a capability of modeling circuit performance in the presence of intrinsic noise. The analysis is limited to frequency domain. Components, such as diodes and transistors, have model parameters that specify their flicker noise behavior (usually it is necessary to double-check that these parameters have been properly assigned in each specific SPICE model before performing noise analysis). The command for noise analysis has the following syntax:

```
.noise V(<out>[,<ref>]) <src> <oct,dec,lin> <Npoints> <StartFreq> <EndFreq>
V(<out>[,<ref>])
```

is the node at which the total output noise is calculated. It can be expressed as V(n1,n2) to represent the voltage between two nodes. <src> is the name of an independent source (a noiseless input signal) to which input noise is referred. The parameters <oct,dec,lin>, <Nsteps>, <StartFreq>, and <EndFreq> define the frequency range of interest and resolution in the manner used in the .ac directive. For example:

```
.noise V(OUT) Vin dec 10 1K 1MEG
```

instructs SPICE to calculate noise at OUT node using frequency sweep with 10 points per decade starting from 1kHz to 1MHz. The corresponding input noise will be referred to Vin source. The analysis produces output data trace V(onoise) accessible for plotting by clicking on OUT node. V(onoise) is a noise spectral voltage density referenced to the OUT node specified as the output in the above syntax. It will have units of V/√Hz. To find total rms voltage noise, one needs to integrate spectral voltage density V(onoise) over a range of frequencies of interest (i.e., the bandwidth). It can be done by pressing  key and left-clicking on V(onoise) legend in the plot window. Additionally, the analysis produces trace V(inoise), which represents the input-referred noise voltage density. It is simply V(onoise) divided by the AC gain of OUT relative to the input source Vin.

By default, the temperature of 27°C is assumed for Johnson noise. You can change this parameter using `.temp <temp_in_celcius>` SPICE directive in the circuit.

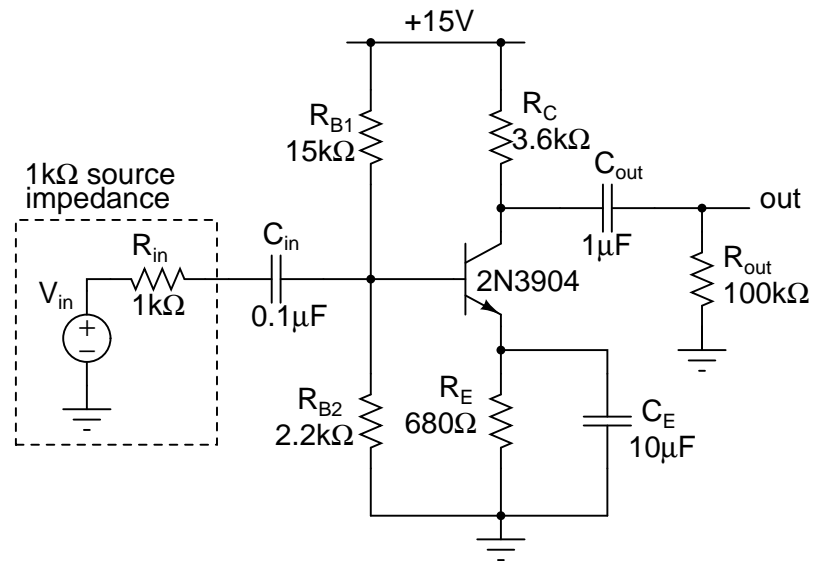


Figure 2.11: Common emitter amplifier for LTSPICE Exp. 12.2: Noise Analysis.

Implement a common-emitter amplifier in LTSPICE as shown in Figure 2.11. Perform noise analysis on OUT node for a frequency range 100Hz to 5MHz. Observe the noise spectral voltage  $V(\text{onoise})$ .

Next, obtain a plot of the noise figure vs. frequency for this circuit. To do so, enter  $20 \cdot \log_{10}(V(\text{onoise})/V(\text{Rin}))$  expression in the plotting utility window. This compares noise levels at  $R_{in}$  input resistor to that of the output OUT, which is the definition of noise figure. Observe  $1/f$  behavior in the noise figure at low frequencies. What do you think is the cause of this behavior?

Next, determine  $v_{n,rms}$  that you would measure from the output of this circuit with an ideal scope that has a perfect response from DC to 5MHz frequency (in the absence of additional noise sources and interference). Determine the signal-to-noise ratio expressed in dB for a signal that has an rms voltage of  $v_{s,rms} = 0.5V$ .

---

## Bibliography

---

- [1] <http://tech.groups.yahoo.com/group/LTspice>
- [2] <http://www.winehq.org>
- [3] For example, a free software project gEDA contains a utility program PCB for producing printed circuit boards <http://geda.seul.org>. A number of other free PCB utilities exist that support netlists to help ensure correct connections, see for example <http://www.freepcb.com>, <http://www.expresspcb.com>, and <http://www.pad2pad.com>.
- [4] Most manufacturers' websites provide SPICE models for their analog devices. Also refer to [1] for a large collection of user-created components (both digital and analog)
- [5] G. Vasilescu, *Electronic Noise and Interfering Signals: Principles and Applications*, Springer, 2005.
- [6] P. Horowitz, W. Hill, *The Art of Electronics*, 2nd Edition, Cambridge University Press, 1989. Sections 7.11-25 and 15.12-15.
- [7] R.E. Simpson, *Introductory Electornics*, 2nd Edition, Allen and Bacon, 1987. Chapter 8.



- .ac, 20
- .ic, 24
- .include, 25
- .lib, 25
- .meas, 18–19
- .noise, 53
- .op, 16
- .param, 18
- .step, 18, 24
- .subckt, 22, 30
- .tran, 10–11
- .wave, 44
- ADC
  - dual slope, 49
- analysis
  - noise, *see* .noise
  - Q-point, *see* .op
  - small gain, *see* .ac
  - time-transient, *see* .tran
- astable multivibrator, 24
- behavioral voltage source, 41
- Bode plot, 20, 33
- circuit block, 30
  - viewing signals in, 31
- CMOS, 36
- common-emitter amplifier, 55
- component
  - placement, 7
  - selection, 7
- custom component, 29
  - model, 29
  - symbol, 27
  - assigning ports in, 28, 30
- DAC
  - 8-bit, 47
  - using behavioral voltage source, 41
- digital family
  - 74HC, 43
  - 74HCT, 39
  - CD4000, 43
  - specifying components, 25
- diode
  - specifying, 13, 34
- dview, 35
- ECAD, 3
- Electr. Comp.-Aided Design, *see* ECAD
- external models, 21
- flip-flop
  - JK, 40
  - master-slave configuration, 38
  - RS, 38
- high-pass filter, 7
- LFSR, 45
- library files, 25, 35
  - digital components, 25, 35

Linear Feedback Shift Reg., *see* LFSR  
 LTSPICE, 3, 7
 

- Control Panel, 11, 31
- help on, 7
- plotting utility, 12
- waveform viewer, 12

missing pulse detector, 42  
 mixed mode of simulation, 3

**netlist**, 5–6, 14, 16  
 noise, 53
 

- figure of, 54
- flicker, 54
- Johnson, 53
- shot noise, 53
- thermal, *see* Johnson noise

op-amp
 

- custom model, 30
- model file for LM741, 21
- specifying, 21

PLD, 37
 

- EPROM, 37
- PROM, 37

Programmable Logic Device, *see* PLD  
 pseudo-random number generator, 45

rectifier
 

- full-wave, 33

ring oscillator, 25

ripple counter
 

- asynchronous, 41
- domino effect, 41
- synchronous, 41

schematic capture, 6, 7  
 signal encryption, 44  
 signal-to-noise ratio, 54  
 SPICE, 5
 

- directives, 6, 10–15, 21
- SPICE3, 5
- XSPICE, 5

SWITCHERCAD, *see* LTSPICE

timer 555
 

- specifying, 42

transformer
 

- center-tapped, 33
- specifying, 33

transistor
 

- specifying, 15, 32

units, 9
 

- decimal prefixes, 9

voltage follower, 21
 

- closed loop gain, 33
- loop gain, 33
- stability margin, 32

voltage source
 

- specifying, 10

waveform arithmetic, 13