Appendix B
Spectre Netlist Language

B.1 Introduction

In addition to accepting SPICE netlists, Spectre also supports a simple, powerful, and extensible language for describing netlists. This appendix describes the basics of Spectre's netlist language only to the level of detail needed to allow you to understand the netlists given in this book.

B.2 The Language

When reading a netlist, SPICE determines the type of a component by the first letter in its name. For example, R1 must be a resistor and Vin must be a voltage source. This approach follows engineering convention and fits well with SPICE's original goal of being a simulator for integrated circuits. However, this convention has two unfortunate limitations. First, it limits the simulator to only supporting only 25 component types (X is reserved for subcircuits). Second, it does not allow the representation type of the components to change. For example, Q1 must refer to a built-in model for a bipolar transistor. It might also be convenient use a macromodel for the transistor. Indeed, integrated transistors often have other junctions near-by that can form parasitic transistors. The built-in model does not include parasitic transistors, but it is easy to construct a macromodel using a subcircuit that contains the original transistor

along with the parasitic transistor. However, because of the constraints on the first letter, one cannot change the type of a transistor from built-in model to macromodel without changing the transistors name. This might not seem like a serious problem until you consider a circuit with thousands of transistors. If you would like to change just the NPN transistors from the built-in model to the subcircuit macromodel that includes the parasitic transistor, you have to make perhaps thousands of hand edits on the netlist.

Spectre's native netlist language is modeled after the SPICE language, but is more uniform. It provides several important features that are unavailable in SPICE due to limitations of its language, including the ability to support an arbitrary number of primitives and the ability to switch representation type.

There are three basic types of primary statements, along with some secondary control statements. The primary statements include component instance, component model, and analysis statements. The control statements include those for specifying initial conditions, nodesets, outputs, etc, and are not discussed further here. In addition, Spectre provides statments used to define parameterized subcircuits.

Basic Language Attributes The Spectre parser has two modes. By default, Spectre is configured to expect SPICE netlists unless told otherwise. When it reads

```
simulator lang=spectre
```

it switches off SPICE compatibility. When doing so, it makes the following changes:

1. Spectre no longer accepts SPICE statements and constructs.
2. The language becomes case-sensitive, with all keywords being lower case.
3. Standard SI scale-factors are used as a convenient way of specifying either very large or very small numbers. The SI scale

P = 10^{15}	T = 10^{12}	G = 10^{9}	M = 10^{6}	K = 10^{3}
k = 10^{3}	_ = 1	% = 10^{-2}	c = 10^{-2}	m = 10^{-3}
u = 10^{-6}	n = 10^{-9}	p = 10^{-12}	f = 10^{-15}	a = 10^{-18}

Table B.1: SI scale factors.

t = 10^{12}	g = 10^{9}	meg = 10^{6}	k = 10^{3}	m = 10^{-3}
mil = 25.4×10^{-6}	u = 10^{-6}	n = 10^{-9}	p = 10^{-12}	f = 10^{-15}

Table B.2: SPICE scale factors.

factors are different than those used by SPICE. For example, with SI, m = 10^{-3} and M = 10^{6}, whereas with SPICE both m and M = 10^{-3}, while MEG = 10^{6}. The SI scale factors are detailed in Table B.1 while the SPICE scale-factors are shown in Table B.2.

To switch back to accepting SPICE netlists, use

```
simulator lang=spice
```

When Spectre includes one file from another using the `include` statement, it automatically switches to SPICE mode at the beginning of the new file, and returns to the previously active mode when it finishes reading it. For this reason, all Spectre netlists must begin with a `lang=spectre` statement.

Model Statements Often, certain characteristics are common to a large number of instances of components of the same type. For example, the saturation current of a diode is a function of the process used to construct the diode and also of the area of the diode. Rather than describing the process on each diode instantiation, that

description is done once in a model statement and many diode instances refer to it. The area, which can be different for each component, is included on each instance statement. Though it is possible to have several model statements for a particular type of component, each instance can only reference at most one model. Not all types of components support model statements.

Model statements have the form

```
model dnp diode is=3.1e-10 n=1.12 cjo=3.1e-8 vj=.914
```

In this example, *dnp* is the model name, and *diode* is the primitive name. The primitive name is either the name of a built-in primitive or a user designed behavioral model.

Instance Statements The instance statement consists of an instance name, the nodes to which the terminals of the instance are connected, the name of the master, and the parameters. It takes the form,

```
M1 (4 pin 1 1) nmos w=402.4u l=7.6u
```

where *M1* is the instance name, '*(4 pin 1 1)*' is the node list (parentheses are optional), *nmos* is the master name, in this case a model name, and '*w=402.4u l=7.6u*' are the parameters. The master field contains either the name of a built-in component type (for example, capacitor), the name of a user-defined behavioral model, a model name, or a subcircuit definition name. The nodes must appear in the order defined by the master. Unlike SPICE, the first character in the instance names is not fixed to any particular value by the type field.

The list of Spectre's available built-in components is shown in Table B.3 on the facing page and Table B.4 on page 362. Further information on these components and their parameters is found by using Spectre's `-help` command-line option.