3 Analog Modeling

In this chapter, Verilog-A, the analog-only subset of Verilog-AMS, will be introduced using a series of practical examples, one example per section. In the beginning the examples will be simple, but they will be useful as is. As the chapter progresses the examples will become more advanced. Once the example is given, all aspects of it will be discussed. As new ideas are presented, they will be set in ***bold italics*** to make them easier to find and to call your attention to them as important points. Once an example is covered in detail, straight forward extensions to the concepts introduced by the example will be covered. Finally, pointers will be given to the language reference where more information can be found. These references appear like this (5§2.3p157), which includes the chapter number, the section number, and finally the page number. In this way, the language will be covered with a fair degree of completeness.

1 Resistor

One of the simplest models that can be described by Verilog-A is a resistor. In general, a resistor is a relationship between voltage and current, as in

$$f(v, i) = 0, \tag{1}$$

where v represents the voltage across the resistor, i represents the current through the resistor, and f is an arbitrary function of two arguments. This is the most general definition of a resistor and so covers what people commonly refer to as a resistor (more precisely termed a linear resistor) as well as nonlinear resistors such as the intrinsic part of diodes and transistors. The resistance of a resistor is the derivative of the voltage with respect to current.

The equation for a simple linear resistor is

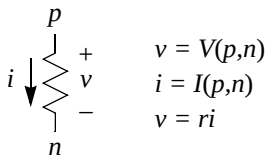
$$v = ri, \tag{2}$$

where r is the resistance. A model for a linear resistor is given in Listing 1. This model uses only Verilog-A constructs and so can be used with both Verilog-A and Verilog-AMS simulators.

LISTING 1 *Verilog-A/MS description of a linear resistor.*

```
// Linear resistor (resistance formulation)
`include "disciplines.vams"
module resistor (p, n);
    parameter real r=0;    // resistance (Ohms)
    inout p, n;
    electrical p, n;

    analog
        V(p,n) <+ r * I(p,n);
endmodule
```



The first line of this model is

```
// Linear resistor (resistance formulation)
```

The // characters begin a ***comment*** (5§1.1p149), which extends to the end of the line. Comments are meant to explain the model to any person that might be trying to understand the model. They are ignored by whatever program is reading the model. In this book, comments will be placed in italics to make them easier to distinguish from the other parts of the model. Comments can also be written inline by using '/*' to start the comments, and '*/' to end them. Inline comments are rare, but this form is often use to write multi-line comments, such as

```
/*
 * RESISTOR
 * A linear resistor that uses the resistance formulation: v = ri
 */
```

Verilog-A/MS is a language that supports multiple disciplines. A ***discipline*** is a collection of related physical signal types, which in Verilog-A/MS are referred to as ***natures***. For example, the *electrical* discipline consists of voltages and currents, where both voltage and current are natures. Verilog-A/MS by itself defines only one discipline, the empty discipline, and it defines no natures. Thus, in order for the language to be able to describe models that operate on physical signals, the disciplines and natures associated with those signals must be defined. A collection of common disciplines and natures are defined in a file *disciplines.vams* (5§2.4p159) that is provided with all implementations of Verilog-A/MS. That file is included into this model by writing

```
`include "disciplines.vams"
```

The tick (`) that precedes the word *include* indicates this is a preprocessor directive (5§1.4p151). This line is replaced by the language preprocessor with the contents of

the file *disciplines.vams* before being passed to the compiler. It defines the names *electrical*, *V*, and *I*, which are used later in the model. It also defines other disciplines and natures, but those are not used in this model. The name *include* is a keyword of the Verilog-A/MS language. Being a keyword, it is not a name that you can choose, both the name and its meaning are specified by the language itself (5§1.3p150). All keywords in listings are set in bold text.

It is not necessary to use *disciplines.vams*. You are free to create your own natures and disciplines. How to do so is described later in Section 3.1 on page 51.

The basic building blocks of Verilog-A/MS are ***modules***. Modules are descriptions of individual components (5§9.1p226). In Verilog-A/MS modules are a block of statements that begin with the keyword *module*, which is then followed by the name of the module and the list of ports. The statement is terminated with a semicolon.

```
module resistor (p, n);
```

A ***parameter*** is specified for the module using the *parameter* statement (5§2.3p157).

```
parameter real r=0;
```

In this case, a real valued parameter *r* is defined that can be specified when the module is instantiated (more about this later). The parameter is given a default value of 0, meaning that if the value is not specified when the module is instantiated, it will assume a value of 0. Thus with no value specified, the resistor will act as a perfect short circuit. All parameters must be given default values. However, specifying the type, in this case *real*, is optional. If not given, the parameter will take the type of the default value.

Ports are the points where connections can be made to the component (5§2.5p164). In this case, they are the terminals for the resistor. So far, the ports have only been given names, but have not been described in any other way. That is done in the two subsequent lines.

```
inout p, n;
electrical p, n;
```

These two lines describe the direction and the type of the ports. The ***port direction*** is given by the statement that begins with the keyword inout. There are three directions possible, input, output, and bidirectional as designated by the *input*, *output*, and *inout* keywords. Each port should be given a direction. Input ports can sense the signals that they are connected to, but cannot affect them; output ports can affect the signals, but cannot sense them; and inout ports can both sense and affect the signals. Since *inout* can do everything that both *input* and *output* ports can do, one might wonder why *input* and *output* ports are needed. In fact, they are not strictly needed. However, using

input and output ports are considered a good practice because it provides clarity of intent. Labeling a port as either input or output at the top of the module makes the behavior of the module clearer. It also allows for extra error checking by whatever tool is reading the module.

The type of the ports is specified by the second of the two lines in which the name of a discipline is followed by a list of ports. In this case the *p* and *n* ports are defined to be *electrical*, meaning the signals associated with the ports are expected to be voltage and current.

The actual behavior of the module is defined in the next two lines.

```
analog
    V(p,n) <+ r * I(p,n);
```

The ***analog*** keyword introduces an ***analog process*** (5§6.1p196). An analog process is used to describe continuous time behavior. Syntactically, it is the analog keyword followed by a statement that describes the relationship between signals. This relationship must be true at all times. In this case, the statement that defines the relationship between the signals on the ports is a contribution statement. A ***contribution statement*** takes the form of a branch signal on the left side of a contribution operator, '<+', followed by an expression on the right side (5§3.2p169). The branch signal on the left side is forced to be equal to the value of the expression. The branch signal on the left is *V*(*p*,*n*), it is the voltage across the implicit branch between the *p* and *n* ports. The expression on the right is *r***I*(*p*,*n*), the product of the parameter *r* and the branch signal *I*(*p*,*n*), which is the current that is flowing through the implicit branch between the *p* and *n* ports. Thus, the contribution statement establishes a relationship between the branch voltage and the branch current that models a linear resistor with resistance *r*.

The signals *V*(*p*,*n*) and *I*(*p*,*n*) are the voltage across and the current through the ***implicit or unnamed branch*** between the nodes *p* and *n* (5§2.6p167). An implicit branch is referenced using its end points, in this case *p* and *n*. The signals associated with the branch are accessed using the ***access functions*** that are given in the definition of the discipline in *disciplines.vams* for the branch. An implicit branch inherits its discipline from its endpoints, both of which must have equivalent disciplines. In this case, the discipline of the end points *p* and *n* are *electrical*, and so the discipline of the implicit branch is *electrical*. The *electrical* discipline defines *V* as the access function for the potential of the branch and *I* as the access function for the flow through the branch. As such, the *V* in *V*(*p*,*n*) accesses the voltage across *p* and *n*, and the *I* in *I*(*p*,*n*) accesses the current that flows between *p* and *n*.

Finally the module definition is terminated with the **endmodule** keyword. Any statements that follow it are not associated with this module.