4 Mixed-Signal Modeling

1 Mixed Signal Models

Both analog and digital functionality as well as the interaction between the two domains are described in mixed signal models. The Verilog-AMS language allows combining analog and digital behavior into a single model. That means that such a model can contain both mixed signal behavioral descriptions or it could instantiate a collection of analog, digital and mixed signal modules. In most cases these models have both analog and digital pins. But this is not always the case; occasionally a model looks digital in terms of its pin types, but analog behavioral descriptions are used inside. Or it may happen that a model looks purely analog when considering the type of its pins, but inside digital constructs are used, perhaps to speed up the model.

When using a top-down design methodology for developing a mixed-signal IC there are various needs for mixed-signal models. During the architectural phase, when blocks are represented abstractly, a single block often includes both complex analog and digital functionality. It is too early to divide this block into purely analog or digital sub-blocks in this phase. Verilog-AMS is used to describe this mixed-signal functionality. At this high level of abstraction the modeling is mostly done in such a way that the disciplines of the pins match. However, as the implementation process proceeds, the blocks are refined to the point where there may be multiple versions of the same block, and those different versions may have different disciplines for the same pin. This implies that as the different versions are used, conflicts may arise between the discipline of a net, and the pins connected to that net. In these cases, ***interface components*** (or connect modules as per the Verilog-AMS language definition) would be needed to resolve the conflict. As the design proceeds to lower levels of detail, these interface components are needed more often. Verilog-AMS provides mechanisms to allow these interface components to be inserted automatically based on the disciplines of the pins and the nets. In doing so, Verilog-AMS not only provides the ability to naturally describe mixed-signal behavioral models, but it also allows mixed-signal models to be built-up from purely analog and digital blocks, and those models to be freely interconnected with Verilog-AMS automatically performing the signal conversion.

The goal of this chapter is to introduce basic mixed signal behavioral and structural modeling techniques, and to help the user to understand the concepts of interface components and their automatic insertion. The following section gives an overview of modeling in the digital domain for the reader not familiar with Verilog-HDL. The understanding of basic digital constructs is a pre-requisite for the mixed signal modeling concepts presented in Section 3 and Section 4.

2 Modeling Discrete Behavior

2.1 Language Basics

The ability provided by Verilog-HDL for describing discrete behavior is fully contained as a subset of Verilog-AMS. That means that every Verilog-HDL model can be legally used in a Verilog-AMS context.

2.1.1 Disciplines

Consider the simple inverter of Listing 1.

LISTING 1 *Verilog-HDL description of an inverter.*

```
module inverter (q, a);
    output q;
    input a;
    wire a, q;      // digital net type (declaration optional)

    assign q=~a;
endmodule
```

The module header in Listing 1 looks similar to the examples provided for Verilog-A. In comparison to these Verilog-A module examples we notice that there is no discipline declaration provided for *a* and *q* (*wire* is not a discipline). Disciplines are a concept that is not part of Verilog-HDL. Disciplines first appeared as part of Verilog-A for continuous-time signals and Verilog-AMS extended the concept to also cover discrete-time signals, however it was made optional for these signals so that Verilog-HDL models can be used by a Verilog-AMS simulator without modification. The default discrete-time discipline is *logic,* which is defined in the *disciplines.vams* file and is shown in Listing 2.

LISTING 2 *The declaration of the discrete discipline ‘logic’.*

```
discipline logic
    domain discrete;
enddiscipline
```

The user may define several discrete disciplines to distinguish between different logic families, different semiconductor processes, different supply voltages, etc. Neverthe-less the inverter example shown in Listing 1 could be used directly without discipline definition and the discipline for *a* and *q* would default to *logic*. Listing 3 adds the explicit discipline declaration to Listing 1, which will later allow more control over the interface component insertion process.

LISTING 3 *The inverter of Listing 1 enhanced with a declaration of wire discipline.*

```
`include “disciplines.vams”

module inverter (q, a);
    output q;
    input a;
    wire a, q;      // digital net type (declaration optional)
    logic a, q;

    assign q = ~a;
endmodule
```

2.1.2 Wires

Now consider the details of the inverter module. The input and output statements are already known from Verilog-A.

```
    wire a, q;
```

defines that *a* and *q* are scalar wires. A ***wire*** is one type of a digital net. A scalar wire can carry one bit of information, and that bit can take one of the four values shown in Table 1 (5§2.5p164).

A wire is the logical representation within Verilog of a physical wire. As such, it can be connected to many things. In particular, there may be more than one thing driving it. An important aspect of the semantics of a wire is how it responds when driven by multiple outputs (drivers). If all of the drivers connected to a wire output a 0, the value of the wire is 0; if they all output a 1, then the value of the wire is 1. However, if the outputs produced by the drivers conflict, the wire resolves to *x* or unknown. The value *x* is interpreted as “either 0 or 1 or in a state of change”. Any drivers that output a *z*

TABLE 1 *Verilog logic values.*

Name	Description	Literal Constant
0	Zero, low, or false.	0 or 1'b0
1	One, high, or true.	1 or 1'b1
x or *X*	Unknown or uninitialized.	1'bx
z or *Z*	High impedance (floating).	1'bz

(high impedance or disconnected) are ignored, unless all drivers output a *z*, in which case the wire resolves to a *z*.†

A wire is not the only type of net available in Verilog (5§2.5p164). There are also those that perform the equivalent of a *wired-or* or *wired-and* operation, those that are equivalent to connecting to the supply or to ground, and one, *trireg*, that mimics a net that holds its value due to charge storage.

It is also possible to declare a bus (a vector wire) by adding a range specification to a wire declaration (5§2.5p164). The range consists of the integer indices for the first and last members of the bus. For example,

```
wire [7:0] data;
```

declares an 8-bit bus, where the members can be accessed with *data*[*i*]; and *i* can range from 7 to 0.

2.1.3 Continuous Assignment

With the exception of the *trireg*, wires do not store values. Rather, they only transmit values that are driven on to them. To continuously transmit a value, it must be continuously driven. One way to do this is with a ***continuous assignment statement*** (5§7.4.2p213), which is the last new statement given in Listing 1.

```
assign q = ~a;
```

indicates that *q* is driven at all times to the value that is the inverse of *a*: the state of *q* changes directly with any change of *a*. The operator '~' is the bitwise invert operator, see (5§4.1p172) for a list of all operators available in Verilog-A/MS.

† In certain cases, resolution rules beyond the scope of this book are triggered that cause the result to be somewhat different. For details, see the Verilog and Verilog-AMS LRMs [16,28].