1 Introduction

1 Hardware Description Languages

Hardware description languages (HDLs) exist to describe hardware. In this they differ from traditional programming languages, which generally exist to describe algorithms. Programming languages such as C grew up with computers that were constructed with a Von Neumann architecture, meaning that they had a central processing unit (CPU) connected to memory that held both instructions and data. The CPU also controlled peripheral elements such as displays, keyboards, long-term storage, networking, etc. The fact that there was one CPU meant that programming languages developed to describe procedures that consist of a sequence of operations performed in a serial manner to the data in memory or on the peripheral elements. Contrast this with typical hardware systems where there are many individual components that all operate simultaneously. To properly describe hardware, one must be able to describe both the behavior of the individual components as well as how they are interconnected.

Hardware description languages have two primary applications: simulation and synthesis. With simulation, one applies various stimuli to an executable model that is described using the HDL in order to predict how it will respond. Simulation allows you to understand how complex systems behave before you incur the time and expense of implementing them. Synthesis is the process of actually implementing the hardware. Here the assumption is that the HDL is used to describe the hardware at an abstract level using component models that do not yet have a physical implementation, and that synthesis is the act of creating a new refined description with equivalent behavior at the inputs and outputs that uses components that do have a physical implementation. The goal for HDLs used for simulation is expressiveness: they should be able to describe a wide variety of behaviors easily. The goal for HDLs used for synthesis is realizability: they should only allow those behaviors that can be converted into an implementation to be described. As such, if a single language is used for both simulation and synthesis, then generally synthesis only supports a relatively constrained subset of the language.

Currently only digital finite-state machines are automatically synthesized. In this case, the desired behavior is described at the register-transfer level (RTL) using a well-defined subset of an HDL. Synthesis then converts the RTL description to an optimized gate-level description. Implementations of the gates are available from a library of standard cells.

Automated synthesis of analog or mixed-signal systems from a description of its desired behavior has not progressed to the point where it is practical except in a few very restricted cases. Furthermore, it is not clear that it will ever reach this point. For this reason, automated synthesis is not discussed in this book. Rather, the focus is on manual synthesis, the process undertaken by designers to convert high-level design requirements to an implementation that meets those requirements. This process, also known as the design process, is not one that traditionally uses hardware description languages when it involves the design of analog or mixed-signal systems. However, as mixed-signal systems become more complex there comes a time where it becomes impractical to design them without using abstraction. It is this point where use of HDLs becomes necessary as they are used to express the abstraction.

There are currently two HDLs available for describing mixed-signal hardware: Verilog-AMS and VHDL-AMS. As the names imply, they are extensions to the traditional Verilog and VHDL digital HDLs that are intended to support modeling of analog and mixed-signal systems. Though these languages have different strengths and weaknesses, they are intended to be used on the same types of circuits, in the same ways, to produce the same results. As such, they are competitors. To a large degree, the choice between them is currently determined by what language is being used for the digital part of the system. However, in the future the simulators supporting the HDLs are expected to fully support both languages, allowing the various components of a single system to be described with which either language is convenient. At that point, one language may begin to dominate over the other. In the mean time, both languages need supporting material that teaches designers how to use them. This book is intended to fulfill that role for Verilog-AMS, with other books doing the same for VHDL-AMS [15, 24].

2 The Verilog Family of Languages

Verilog-AMS is a modeling language for mixed-signal systems. It is primarily designed to support simulation of mixed-signal systems by allowing the system to be described to the simulator. However, mixed-signal systems represents a very broad class of systems and must support a wide variety of situations. As such, Verilog-AMS is a language that has a diverse range of capabilities.

The term “mixed-signal” suggests systems made up of parts that process digital signals and parts that process analog signals. As such, Verilog-AMS is a language that supports the description of both digital and analog components. Verilog-AMS is the merger and extension of two languages, Verilog-HDL and Verilog-A. These three languages currently make up the Verilog® family of languages.† Verilog-HDL allows the description of digital components and Verilog-A allows the description of analog. Verilog-AMS combines these two languages and adds additional capability to allow the description of mixed-signal components. The term Verilog-AMS will be used when referring to just the full AMS extensions and Verilog-A/MS when referring to both Verilog-A and Verilog-AMS.

Digital signals are discrete-event signals with discrete values. In other words, they are signals that are constant for a period of time, and then abruptly change to a new value. With digital signals there are generally only a small number of possible signal values, typically two, designated *true* and *false*, *high* and *low*, or *zero* and *one*. The Verilog-HDL language was designed to handle such signals, and the systems that generate them. This language has been available for many years. It is both well known and well documented, and so will not be discussed in depth in this book. If you wish more information on Verilog-HDL, try picking up one of the many books available that focus on it exclusively [1, 5, 23, 27].

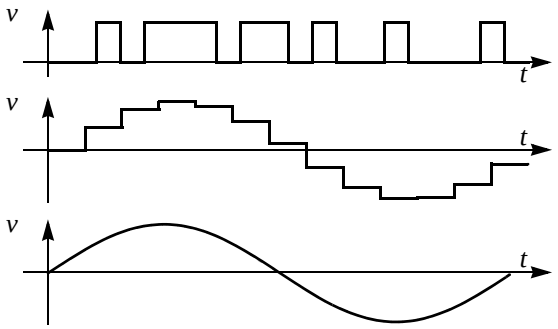
Analog signals are signals that vary continuously, meaning that the value of the signal at any point may be any value from within a continuous range of values. There are two ways in which this typically occurs, as shown in Figure 1. Either the signal is piecewise constant versus time, meaning that it holds its value for a period of time before jumping to a new value, or it is continuous versus time, meaning that its value varies smoothly as a function of time. The former signals are referred to as being analog discrete-event signals and the latter are continuous-time signals. The figure shows the analog discrete-event signal jumping between values at regular intervals, but this is not necessary. Both the value, and the time at which the jump-events occur can be irregular.

Verilog-A is designed to allow modeling of systems that process continuous-time signals. While it can also handle systems that process the other types of signals, it is not efficient for doing so. Verilog-A has been around for many years, though not nearly as many as Verilog-HDL. The documentation available is either incomplete [8] or hard to find [28], and so Verilog-A is presented in depth in Chapter 3.

Since Verilog-AMS combines Verilog-HDL and Verilog-A, as shown in Figure 2, it inherits the ability to handle systems that process both digital and continuous-time

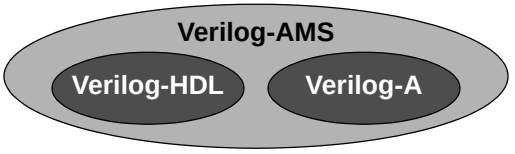
† Verilog is a registered trademark of Cadence Design Systems licensed to Accellera.

FIGURE 1 *Digital, analog discrete-event and analog continuous-time signals.*



analog signals. It also adds the ability to efficiently support systems that process analog discrete-event signals. Verilog-AMS is the subject of Chapter 4,

FIGURE 2 *The relationship between Verilog-AMS, Verilog-A and Verilog-HDL.*



Verilog-AMS is expected to have a big impact on the design of mixed-signal systems because it provides a single language and a single simulator that is shared between analog and digital designers, and between block designers and system designers. It will be much easier to provide a single design flow that naturally supports analog, digital and mixed-signal blocks, making it simpler for these designers to work together.

Verilog-AMS makes it substantially more straight-forward to write behavioral models for mixed-signal blocks, and brings strong event-driven capabilities to analog simulation, allowing analog event-driven models to be written that perform with the speed and capacity inherited from the digital engines. This is very important, because most of the analog and mixed-signal models used in high-level simulations are naturally written using event-driven constructs. For example, blocks like ADCs, DACs, PLLs, ΣΔ converters, discrete-time filters (switched-capacitor), etc. are easily and very efficiently modeled using the analog event-driven features of the AMS languages.