# HARDWARE DESCRIPTION

**Mehran M. Massoumi,**
*HDL Research & Development, Averant Inc., USA*

**Keywords:** HDL, Hardware Language, Digital Design, Logic Design, RTL, Register Transfer, VHDL, Verilog, VLSI, Electronic CAD.

**Contents**





## 1. Introduction

Throughout the decades of digital computer history, various notations have been developed for capturing the logical behavior of digital circuits at different levels of abstraction. The primary purpose of these notations has been to deemphasize the electronic and fabrication aspects of the design and therefore facilitate documentation, analysis and synthesis of large complex systems. Schematics, Boolean equations, timing charts, state transition tables, block diagrams, and hardware description languages are examples of such notations. Although most of these approaches are being used in practice today, hardware description languages have gained significant acceptance starting from late1980s.

A Hardware Description Language (HDL henceforth) is a set of notations, similar to software programming languages, used for modeling the logical function of digital circuits and systems. Compared to alternate forms of design capture, it has been shown, in practice, that the use of HDLs shortens the design cycle and yields more robust realizations. Many concede that without HDLs, the design of today's complex circuits would not be possible in a reasonable amount of time. A hardware description can serve as a principal means of communication between members of a design team. The conciseness and readability of HDLs minimize the need for any natural language, and

more error prone, discussion of the design. Furthermore, a hardware description can be used as an input to a variety of analysis and synthesis tools. These tools greatly facilitate the verification and realization of the described circuits. Similar to a software programming language where the target machine code is hidden from the programmer, HDLs are independent of any particular target circuit technology. This feature contributes to improved readability and design management. It should be noted that an HDL is a language and only a language and has no apparent algebraic structure in terms of guiding the user to a minimal implementation. However, with practice, the designer will readily arrive at modeling techniques that yield more efficient realizations.

This writing is an introduction to hardware description and languages. No prior exposure is assumed, although proficiency in Boolean Algebra, combinational logic circuits, and sequential circuits is a prerequisite. Furthermore, knowledge of at least one software programming language helps in better understanding of the HDL concepts. To begin with, a short historical note on the evolution of description languages and the standardization efforts will be presented. Subsequently, the levels of abstraction and the fundamental elements of a description language will be discussed. The standard language Verilog will be used as a vehicle to present various modeling concepts and techniques, including concurrency, procedural descriptions, and modeling of various hardware components. Verilog has been selected for a number of reasons: (1) it is a widely used standard language, (2) it is simple and intuitive and therefore does not overwhelm the reader with esoteric semantic details, and (3) it contains a wide repertoire of language features that allow description of complex systems. However, no attempt will be made in this writing to cover the Verilog language in its entirety. A complete description can be found in the references [IEE96, Nav99, Tho96].

## 2. A Historical Note

The concept of a hardware description language as a medium for design capture was first introduced in the1950s, but wide adoption by the design community did not start until after 1985. Historically, the development of software programming languages stimulated the evolution of HDLs. One example, among many, is the programming language APL [Ive62] which was used as a form of design entry for a logic automation system developed at IBM in the early 1960s. The notational conventions of APL were later used by researchers at the University of Arizona to design AHPL (A Hardware Programming Language) [Hil74]. Since its introduction in the early 1970s, AHPL was hardly used in non-academic applications but served as an effective teaching tool [Hil87] in classroom environments. In the three decades starting from 1960 many HDLs were introduced including DDL[Dul69], ISPS[Bar81], and Zeus[Lie83]. However, the use of these languages rarely exceeded research and academic applications.

Partly in reaction to the proliferation of HDLs and partly due to its own needs, in 1980, the U.S. Department of Defense initiated the development of VHDL as part of its VHSIC program (Very High Speed Integrated Circuits; The name VHDL is derived from VHSIC Hardware Description Language). The overall objective of this effort was to design a single language that will allow the design, documentation, and analysis of hardware at various levels of abstraction. Moreover, the intend was to make VHDL the standard HDL in all DoD design projects and use the language as a means of

communication between various teams or departments. Even though none of the existing HDLs met the requirements of the VHSIC program, the features and shortcomings of the following eight HDLs were studied before designing the new language: AHPL, CDL, CONLAN, IDL, ISPS, TEGAS, TI-HDL, and Zeus. In December of 1987 VHDL was adopted by the IEEE [IEE88, IEE94] as a standard description language for modeling digital circuits and systems. The emergence of such industry standard marked the beginning of widespread adoption of HDLs by the design community.

Parallel with VHDL efforts, the Verilog HDL originated in 1983 at the Gateway Design Automation and was introduced into the market in 1985. Verilog was designed to address the requirements of circuit designers and at the same time to be intuitive and as simple as possible. As a result, the language has fewer constructs than VHDL, its semantics is not as complex, and development of Verilog-based tools are more readily realizable. Verilog became an IEEE standard [IEE96] in 1995.

Today, both Verilog and VHDL are used extensively, and perhaps exclusively, by circuit designers everywhere. Many simulation and synthesis tools have been developed and marketed that automate the analysis and realization of these HDL models.

## 3. Levels of Abstraction

Levels of abstraction in a hardware description refers to the amount of design decisions that have been specified in the description of the circuit; The fewer the design decisions, the higher the abstraction level. Three distinct levels of abstraction have been recognized in hardware descriptions: Gate Level (or structural), Register Transfer Level (also known as, RTL or dataflow), and Behavioral Level.

The lowest and most detailed level of abstraction is the gate level where specific gates and components and their interconnections are specified in the description. A gate level model is very specific in terms of the design architecture and what gates are to be used for the final implementation. The corresponding function of the hardware is not evident from such descriptions unless the function of each constituent component is known. One can think of a gate level description as a textual representation of the circuit schematic.

Register transfer, or dataflow, models are more abstract and describe a controlled flow of data between buses and registers in the design. Unlike the gate level models, no specific components are specified in dataflow models since that decision is abstracted out in the interest of efficiency, simplicity, and better design management. Even though the mapping between a dataflow description and a hardware architecture is one to many, the hardware correspondence of each statement is well defined. Furthermore, certain architectural decisions, such as the scheduling of data transfers in relation to the system clocks, are specified in the dataflow models.

The behavioral level is the most abstract level and is used to describe the function of the design without providing any implementation details. Furthermore, hardware correspondence is not well defined and in most cases is not evident from the model

description. Behavioral models present an input to output mapping according to the data sheet specification and can serve as a concise documentation medium. Compared to gate level and dataflow counterparts, behavioral models often require less effort to develop and are useful for simulation and functional analysis during early stages of the design.

The motivation for recognizing various levels of design abstraction is to provide ways to manage and analyze any given design at all stages of the design cycle. Top-down design methodologies often start by first contriving more abstract models and then proceed to more detailed descriptions. Bottom-up design approaches often start by evaluating existing models and composing them hierarchically with new models in order to achieve the desired functionality.

In this writing, the initial focus will be dataflow and gate level models. Since these levels have closer hardware correspondence, it will be a more natural transition for readers with logic design background. A treatment of behavioral descriptions will appear in later sections.

## 4. Fundamental Characteristics of a Description Language

A general hardware description language should support all levels of abstraction and a number of other characteristics listed below:

- Computations in hardware occur in parallel and this implies that an HDL should support concurrency or description of parallel actions. Additionally, procedural constructs should be supported in order to facilitate description of hardware algorithms that occur within a specific time unit. These procedural constructs are similar to software languages and improve readability.
- Carriers and operators are an integral part of any HDL. A carrier is either a wire or memory, depending on the latency of information stored in them. Operators take their operands from carriers, perform their function, and return the computed outputs onto carriers.
- HDLs should support a variety of data types that a carrier can assume. However, the most elementary type is the binary digit or bit. Aggregate types are formed by hierarchically grouping bits into vectors and arrays. Often times operators are applied to arrays or vectors of bits and therefore any viable HDL should provide support for such types and operations.
- Partitioning a large design based on functional coherency, and connecting these partitions hierarchically is common practice among designers. Such approach improves design management and allows design reuse. Therefore, HDLs should support component instantiations and hierarchical configuration of designs. Moreover, parameterization of modules in the context of hierarchical design allows description of generic modules which can be customized when instantiated.
- Another commonly desired feature is the ability to express the dichotomy between data and control. Since different optimization techniques are applied to data and control sections, such separation provides the needed design control.
- Hardware independence is a notable characteristic of any HDL. The language notations should be independent of any specific circuit technology.

## 5. Hardware Description and Concurrency

All hardware elements or gates in a logic circuit operate in parallel. Each element responds to changes or events on its immediate inputs and updates its outputs which could in turn cause events on the inputs of other elements, depending on the circuit connectivity. This may start a chain of switching activity in the entire circuit until all events have been propagated and a steady state is reached. To illustrate this point, let us compose a dataflow Verilog description of a single full adder.

A general template for describing a hardware unit in Verilog. A module description begins with the keyword **module** and is followed by the list of input/ output ports, declarations, specification of behavior, and the keyword **endmodule**.

The ports of a module represent the set of carriers that allow the module to communicate with the outside world. A port can be an input, output, or bi-directional inout. The declarations of a module specify the type of each carrier as well as whether the carrier is a port or a signal used for intermediate computations. The remaining part is the specification of the behavior where a mapping between the inputs and outputs is described. What follows is a Verilog description of a full adder. All keywords are shown in bold letters.

**module** full_adder(a, b, carry_in, carry_out, sum);
**input** a, b, carry_in;
**output** carry_out, sum;
**wire** carry_out, sum, temp;
**assign** temp = a ^ b;
**assign** sum = temp ^ carry_in;
**assign** carry_out = (temp & carry_in) | (a & b);
**endmodule**



Any carrier that has a driver within the body of a Verilog module must be declared as a **wire**, **wor**, **wand**, **reg**, or a number of other carrier kinds, discussed later. In the *full_adder* model, both *carry_out* and *sum* are single-bit output ports and are also declared as **wire**. Moreover, the intermediate signal *temp* which is not a port is declared as a **wire**. As the name implies, a wire is simply a carrier that is continuously driven and has no memory. In Verilog, a wire is of data type bit and can take four possible values 0, 1, X, and Z. The X value is used for modeling unknown or logic don't care and the Z value is used for three state modeling. Both X and Z will be discussed later and for this example the focus is on the binary values 0 and 1.

The operators used in the *full_adder* description are xor (i.e., ^), bit-wise and (i.e., &), and bit-wise OR (i.e., |). The body of the module consists of three continuous assignments describing the full adder function. All three assignments are concurrent and therefore the order in which they are written does not affect the behavior. Prior to delving into order of evaluation of concurrent statements, it is helpful to look at how time is modeled in terms of simulating a description.

-
-
-



**Bibliography**

M. R. Barbacci, "A Comparison of Register Transfer Languages for Describing Computers and Digital Systems", IEEE Transactions on Computers, Vol. C-24, No. 2, January 1975.

M. R. Barbacci, "Instruction Set Processor Specifications (ISPS): The notation and its applications", IEEE Transactions on Computers, Vol. C-30, No. 1, January 1981.

J. R Duley, D. L. Dietmeyer, "A Digital System Design Language (DDL)", IEEE Transactions on Computers, Vol. C-17, pp 850-861, September 1969.

F. J. Hill, "Introducing AHPL", IEEE Computer, 1974.

F. J. Hill, G. R. Peterson, "Digital Systems, Hardware Organization and Design ", 3rd Edition, John Wiley & Sons, 1987.

IEEE Inc., "IEEE Standard VHDL Language Reference Manual", IEEE Std 1076-1987, IEEE 1988.

IEEE Inc., "IEEE Standard VHDL Language Reference Manual", ANSI/IEEE Std 1076-1993, IEEE 1994.

IEEE Inc., "IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language", IEEE Std 1364-1995, ANSI/IEEE 1996.

K. E. Iversosn, "A Programming Language", Wiley, New York, 1962.

K. J. Lieberherr, S. E. Knudsen, "Zeus: A Hardware Description Language for VLSI", Proceedings of the 20th ACM/IEEE Design Automation Conference, 1983.

Z. Navabi, "VHDL: Analysis and Modeling of Digital Systems", 2nd Edition, McGraw- Hill Publishing, New York, 1998.

Z. Navabi, "Verilog Digital System Design," McGraw-Hill Publishing, New York, 1999.

D. E. Thomas, and P. R. Moorby, " The Verilog Hardware Description Language," 3rd Edition, Kluwer Academic Publishers, Norwell, MA, 1996.