International Journal of Scientific Trends- (IJST)
ISSN: 2980-4299
Volume 4, Issue 9, September - 2025
Website: https://scientifictrends.org/index.php/ijst
Open Access, Peer Reviewed, Scientific Journal

Hardware Description Languages in Modern Digital Design

Shadiyeva Marjona
Inha University in Tashkent / Department of Hardware Engineering
marjonashu@icloud.com

Abstract
Hardware Description Languages also known as HDLs have become a breakthrough in the world of engineering by transforming the way digital systems are designed and implemented. As technology becomes increasingly complex, traditional schematic-based design methods are no longer sufficient to describe complex hardware systems. HDLs have emerged as tools that address this problem since they provide efficient and structured ways to describe circuits at various abstraction levels and enable the synthesis of these descriptions into real hardware. This paper aims to demonstrate the importance of using HDLs in designing modern digital systems by outlining types of HDLs, their use in hierarchical and modular design, verification, simulation and synthesis processes. The examination of the role of HDLs demonstrates that they have become crucial in digital design due to their ability to provide a framework for transforming mere descriptions into real-life circuits.

Keywords:

Introduction

In our cutting-edge era, when technologies grow more and more complex, so do the schematics of circuits based on which these technologies are implemented. This creates a need for textual and structured methods for describing circuits, since traditional schematic design has become less relevant due to its inability to represent and manage all parts of complex circuits. Schematics of complex hardware can contain an enormous number of logical gates, hence, drawing them has become impractical and time-consuming. Other problems were lack of abstraction and poor reusability of traditional schematic-based drawing, since they represent circuit only at the level of wires and making any changes would require redrawing the whole scheme.

It is HDLs that replaced traditional design and provided means to model complex systems at various levels of abstraction, such as behavioral, register-transfer and gate levels. They enabled simulation of circuits before their physical production and synthesis of circuit description into actual physical hardware. HDLs eliminated the need to manually draw interconnection of gates

International Journal of Scientific Trends- (IJST)
ISSN: 2980-4299
Volume 4, Issue 9, September - 2025
Website: https://scientifictrends.org/index.php/ijst
Open Access, Peer Reviewed, Scientific Journal

and wires by providing simple code that can effectively represent any circuit. Their hierarchical design gives opportunity to test each sub-module separately and increases reusability.
The purpose of this paper is to examine the role that Hardware Description Languages have in designing digital devices, by firstly exploring all abstraction levels they use and their different types. Then, it discusses main features of hierarchical and modular design, verification process and various categories of simulation: functional, timing and post-synthesis simulations. Finally, it explains synthesis and implementation using FPGA and ASICs.

2. Fundamentals of Hardware Description Languages

2.1 Definition

Hardware Description Language is a computer language, but unlike programming languages like Pascal and C that are used to describe computer programs, Hardware Description Languages are used in the field of digital design to describe digital circuits. If programming languages follow sequential execution, HDLs are concurrent. That is to say, they allow engineers to model the concurrency of processes that occur in hardware elements (Palnitkar, 2003)

2.2 Abstraction levels

HDLs allow engineers to describe circuits on various abstraction levels. The most essential abstraction levels are:
1**. Behavioral level**. This is the highest level of abstraction provided by HDLs and it describes the overall behavior of a circuit. A module can be implemented in terms of the desired design algorithm without concern for the hardware implementation details. Designing at this level is very similar to C programming. (Palnitkar, 2003)

2**. Dataflow level**. At this level, the module is designed by specifying the data flow. It is important that how data flows between hardware registers and how it is processed within the design should be understood by designer. (Palnitkar, 2003)

3. **Gate level.** The module is implemented in terms of logic gates (AND, NOT, OR, NAND etc) and how these gates are interconnected. Design at this level is similar to describing a design in terms of a gate-level logic diagram. (Palnitkar ,2003)

4. **Register-transfer level** (RTL). This level represents a combination of behavioral and dataflow modellings – it describes circuits in terms of data flow at register level. RTL is the most common level used for synthesis, as it provides a good balance between abstraction and applicability. (Palnitkar, 2003)

2.3 Types of HDLs

When learning about HDLs it is crucial to know about different types of HDLs. Nowadays there are three main languages that are most commonly used:
1.**VHDL**-which stands for VHSIC (Very High-Speed Integrated Circuit) Hardware Description Language, was developed in the early 1980s by the U.S. Department of Defense (DoD) as part of

the VHSIC program. The program's primary goal was to create a standardized language for the design and verification of digital systems, particularly for military applications. (Sufyan, 2024)

2. **Verilog** – introduced by Gateway Design Automation in 1984 as a proprietary language for simulating and verifying digital circuits. It was one of the first hardware description languages developed from 1983 to 1984. In 1995 The first official standard for Verilog, known as IEEE Standard 1364-1995, was published. (Sufyan, 2024)

3.**SystemVerilog** - started with the donation of the Superlog language to Accellera in 2002 by the startup company Co-Design Automation. In 2005, SystemVerilog was adopted as IEEE Standard 1800-2005. In 2009, the standard was merged with the base Verilog (IEEE 1364-2005) standard, creating IEEE Standard 1800-2009(Wikipedia contributors, n.d.).

3. Hierarchical and modular design

Hardware description languages use a hierarchical design, due to which they became popular and widely used. Hierarchical design addresses some of the problems which traditional written schematics had- it provides a way to cover all elements without ambiguity and enables reusability. There are two main hierarchical methodologies: top-down, where high-level functionality is divided into smaller sub-blocks, and bottom-up, where small blocks are designed first and integrated into larger systems. These hierarchical modeling concepts can be related to HDLs. HDLs provide a concept of a module- which is the basic building block and can be an element or a collection of lower-level design blocks. A module provides the necessary functionality to the higher-level block through its port interface (inputs and outputs), but hides the internal implementation- this concept is called encapsulation. This concept enables modifying and testing module internals and separate blocks without affecting the rest of the design (Palnitkar, 2003).
In Verilog, a module is declared by the keyword module. A corresponding keyword endmodule must appear at the end of the module definition (Fig 1). For VHDL, it's "entity-architecture" construct (Palnitkar, 2003).

```
module <module_name> (<module_terminal_list>);

...
<module internals>
...
...
endmodule
```

Fig 1. Syntax of Verilog for modules

Similarly, using modular design enables reusability- once the module is written, it can be reused again without modification. Thus, this mirrors software engineering principles of encapsulation and modularity.

4.Verification and Simulation

Hardware circuits and chips require a huge amount of time and resources. In the case of smallest mistake or bug it will cost month of work and money, since change must be implemented in original design, and new chip should be manufactured. To avoid such troublesome cases, verification process must be done.

Verification is a critical process in hardware engineering that ensures that system or component meets its specified requirements and performs intended functions accurately. It checks whether design implementations match the original design intent before manufacturing. Not only does verification help to catch functional bugs, but also checks performance and security vulnerabilities, which ensures robustness and reliability of a product (Khan, 2025). There is a wide range of techniques using which verification is done, but for now only the most widely used technique will be discussed- simulation.

Simulation is the verification technique which involves executing the design under the test (DUT) in a software environment using testbenches (or simulation blocks) that imitate real-world scenarios. DUT is run on a computer or simulator, and it is observed how it reacts to different inputs such as test vectors. This process reveals functional errors and unexpected behaviors (Khan, 2025). Simulations use HDLs code and execute them like a program. They understand HDL code and allow writing testbenches which provide stimuli like inputs and expected outputs.

There are three main types of simulations:

1**. Functional simulation**. Functional simulation is used to verify the logical correctness of the design and performed in the very early stages. It ensures that design behaves and functions as intended logically. However, it does not consider the timing delays of the physical implementation, and for that reason it is generally faster than timing simulation. It is performed using simulators like Xilinx Vivado Simulator (Vemeko, 2025).

2.**Timing simulation.** It is used to verify that the design meets timing requirements after considering the physical implementation. Without it design may fail timing constraints, even if it functions correctly, so timing simulation ensures design works considering real-world delays. It is performed after design has been synthesized, and it is generally slow due to timing analysis. Performed using Xilinx Vivado and other tools (Vemeko, 2025).

3.**Post-synthesis simulation**. This is simulation that is done after design synthesis into gate-level netlist. The earlier simulation at a higher level of abstraction does not account for specific implementations of the hardware components that the design is using. For that reason post-synthesis simulation is done to ensure that implementation of design did not break its functionality. Also performed using Xilinx Vivado simulators (Roth et al., 2016).

5. Synthesis and implementation

The next two consecutive stages in designing hardware are synthesis and implementation. These stages are performed after writing HDL code and verification and functional simulation processes and logically follow design flow (Fig 2).

In the synthesis process, HDL codes are compiled and translated into netlist by a program called a synthesis tool. In detail, synthesis tools convert the RTL descriptions to a gate-level netlist which is a description of the circuit in terms of gates and connections between them. Synthesis tools ensure that the gate-level netlist meets all required specifications, such as timing, area and power.

International Journal of Scientific Trends- (IJST)
ISSN: 2980-4299
Volume 4, Issue 9, September - 2025
Website: https://scientifictrends.org/index.php/ijst
Open Access, Peer Reviewed, Scientific Journal

Hence, input in this stage is RTL HDL code, output: gate-level netlist (Mano & Kime, 2001; RealDigital, n.d.).
Next, gate-level netlist is passed as an input to an Automatic Place and Route (APR) tool. It creates a layout which is verified and then fabricated on a chip. This stage is called implementation. Implementation process maps the synthesized design onto FPGA or ASIC chip targeted by design. Output of this stage is physical layout for ASIC or configuration for FPGA (Mano & Kime, 2001; RealDigital, n.d.).
After completing these stages, design is ready and usually post-implementation verification and timing analysis are carried out to ensure that design works as expected under physical constraints and meets all requirements.

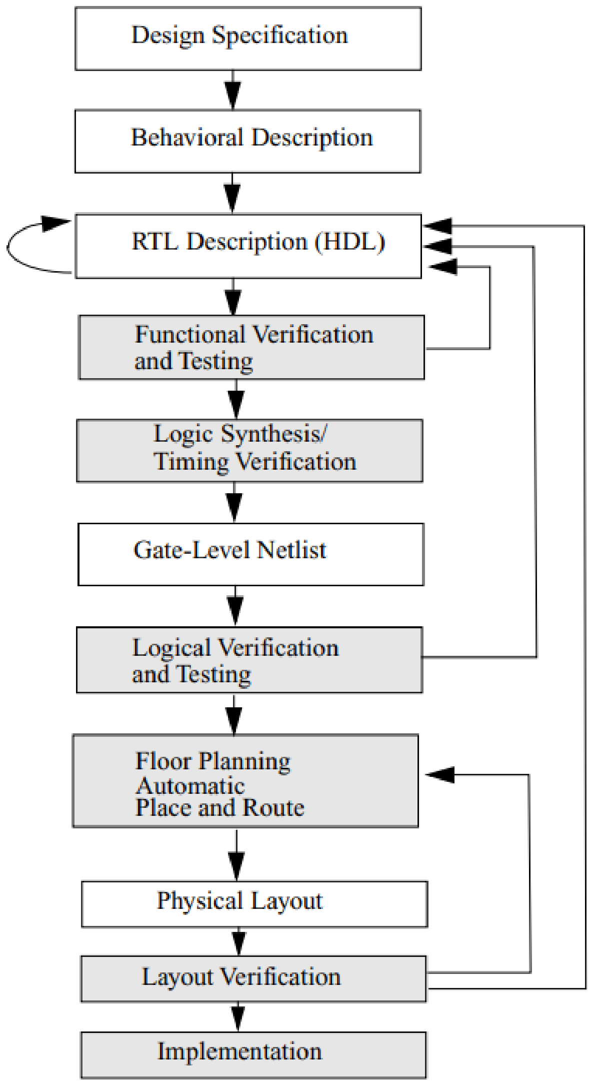
Design Specification
Behavioral Description
RTL Description (HDL)
Functional Verification and Testing
Logic Synthesis/ Timing Verification
Gate-Level Netlist
Logical Verification and Testing
Floor Planning Automatic Place and Route
Physical Layout
Layout Verification
Implementation

Fig 2. Typical design workflow
(Mano & Kime, 2001)

6. Conclusion

Hardware Description Languages have become a pivotal tool in designing digital systems. They are essential in the design workflow and contribute to almost all stages of design development and implementation. They provide different levels of abstraction and help to describe a circuit in a concise and efficient way. HDLs allow engineers to avoid costly mistakes and provide methods to test a circuit at various stages of development and consider all possible scenarios. This is done through verification and simulation processes. Moreover, they are directly used in synthesis and implementation stages which are responsible for converting code into real-life circuits. Implementation using FPGA and ASIC demonstrates that HDLs are tools that bridge abstract descriptions and physical hardware.

As hardware systems grow even further in their complexity, the future of design will not rely solely on HDLs, but rather on engineers' ability to integrate them with upcoming tools and methodologies. This will ensure that digital design remains reliable and efficient.

References

1. Khan, K. (2025). What is verification? Synopsys. https://www.synopsys.com/glossary/what-is-verification.html
2. Mano, M. M., & Kime, C. R. (2001). Logic and computer design fundamentals (2nd ed.). Prentice Hall.
3. Palnitkar, S. (2003). Verilog HDL: A guide to digital design and synthesis (2nd ed.). Prentice Hall.
4. RealDigital. (n.d.). Introduction to logic simulation. RealDigital. https://www.realdigital.org/doc/bd6a53089056fc9e2888deabdfcb2a66
5. Roth, C. H., John, L. K., & Lee, B. K. (2016). Digital systems design using Verilog (2nd ed.). Cengage Learning.
6. Sufyan, M. (2024, April 8). Verilog vs VHDL: A comprehensive comparison. Wevolver. https://www.wevolver.com/article/verilog-vs-vhdl-a-comprehensive-comparison
7. Vemeko. (2025, February 7). What's the difference between functional simulation and timing simulation? Vemeko Blog. https://www.vemeko.com/blog/67138.html
8. Wikipedia contributors. (n.d.). SystemVerilog. In Wikipedia. Retrieved September 15, 2025, from https://en.wikipedia.org/wiki/SystemVerilog.