Overview of Digital Design with Verilog HDL

1

1.1 Evolution of Computer-Aided Digital Design

Digital circuit design has evolved rapidly over the last 25 years. The earliest digital circuits were designed with vacuum tubes and transistors. Integrated circuits were then invented where logic gates were placed on a single chip. The first integrated circuit (IC) chips were SSI (*Small Scale Integration*) chips where the gate count was very small. As technologies became sophisticated, designers were able to place circuits with hundreds of gates on a chip. These chips were called MSI (*Medium Scale Integration*) chips. With the advent of LSI (*Large Scale Integration*), designers could put thousands of gates on a single chip. At this point, design processes started getting very complicated, and designers felt the need to automate these processes. *Electronic Design Automation* (EDA)[1] techniques began to evolve. Chip designers began to use circuit and logic simulation techniques to verify the functionality of building blocks of the order of about 100 transistors. The circuits were still tested on the breadboard, and the layout was done on paper or by hand on a graphic computer terminal.

With the advent of VLSI (*Very Large Scale Integration*) technology, designers could design single chips with more than 100,000 transistors. Because of the complexity of these circuits, it was not possible to verify these circuits on a breadboard. Computer-aided techniques became critical for verification and design of VLSI digital circuits. Computer programs to do automatic placement and routing of circuit layouts also became popular. The designers were now building gate-level digital circuits manually on graphic terminals. They would build small building blocks and then derive higher-

1. The earlier edition of the book used the term *CAD tools*. Technically, the term *Computer-Aided Design (CAD) tools* refers to back-end tools that perform functions related to place and route, and layout of the chip. The term *Computer-Aided Engineering (CAE) tools* refers to tools that are used for front-end processes such HDL simulation, logic synthesis, and timing analysis. Designers used the terms CAD and CAE interchangeably. Today, the term Electronic Design Automation is used for both CAD and CAE. For the sake of simplicity, in this book, we will refer to all design tools as *EDA tools*.

level blocks from them. This process would continue until they had built the top-level block. Logic simulators came into existence to verify the functionality of these circuits before they were fabricated on chip.

As designs got larger and more complex, logic simulation assumed an important role in the design process. Designers could iron out functional bugs in the architecture before the chip was designed further.

1.2 Emergence of HDLs

For a long time, programming languages such as FORTRAN, Pascal, and C were being used to describe computer programs that were sequential in nature. Similarly, in the digital design field, designers felt the need for a standard language to describe digital circuits. Thus, *Hardware Description Languages* (HDLs) came into existence. HDLs allowed the designers to model the concurrency of processes found in hardware elements. Hardware description languages such as *Verilog® HDL* and VHDL became popular. Verilog HDL originated in 1983 at Gateway Design Automation. Later, VHDL was developed under contract from DARPA. Using both Verilog and VHDL simulators to simulate large digital circuits quickly gained acceptance from designers.

Even though HDLs were popular for logic verification, designers had to manually translate the HDL-based design into a schematic circuit with interconnections between gates. The advent of logic synthesis in the late 1980s changed the design methodology radically. Digital circuits could be described at a *register transfer level (RTL)* by use of an HDL. Thus, the designer had to specify how the data flows between registers and how the design processes the data. The details of gates and their interconnections to implement the circuit were automatically extracted by logic synthesis tools from the RTL description.

Thus, logic synthesis pushed the HDLs into the forefront of digital design. Designers no longer had to manually place gates to build digital circuits. They could describe complex circuits at an abstract level in terms of functionality and data flow by designing those circuits in HDLs. Logic synthesis tools would implement the specified functionality in terms of gates and gate interconnections.

HDLs also began to be used for system-level design. HDLs were used for simulation of system boards, interconnect buses, FPGAs (Field Programmable Gate Arrays), and PALs (Programmable Array Logic). A common approach is to design each IC chip, using an HDL, and then verify system functionality via simulation.

Today, Verilog HDL is an accepted IEEE standard. In 1995, the original standard IEEE 1364-1995 was approved. IEEE 1364-2001 is the latest Verilog HDL standard that made significant improvements to the original standard.

1.3 Typical Design Flow

A typical design flow for designing VLSI IC circuits is shown in Figure 1-1. Unshaded blocks show the level of design representation; shaded blocks show processes in the design flow.

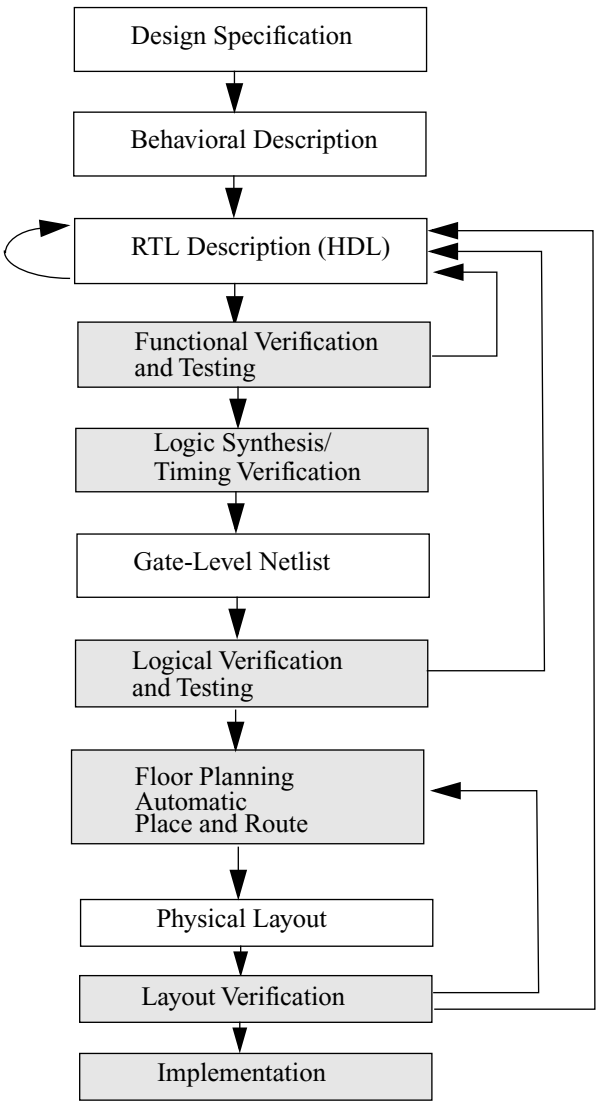

Figure 1-1 Typical Design Flow

The design flow shown in Figure 1-1 is typically used by designers who use HDLs. In any design, specifications are written first. Specifications describe abstractly the functionality, interface, and overall architecture of the digital circuit to be designed. At this point, the architects do not need to think about how they will implement this circuit. A behavioral description is then created to analyze the design in terms of functionality, performance, compliance to standards, and other high-level issues. Behavioral descriptions are often written with HDLs.[2]

The behavioral description is manually converted to an RTL description in an HDL. The designer has to describe the data flow that will implement the desired digital circuit. From this point onward, the design process is done with the assistance of EDA tools.

Logic synthesis tools convert the RTL description to a gate-level netlist. A gate-level netlist is a description of the circuit in terms of gates and connections between them. Logic synthesis tools ensure that the gate-level netlist meets timing, area, and power specifications. The gate-level netlist is input to an Automatic Place and Route tool, which creates a layout. The layout is verified and then fabricated on a chip.

Thus, most digital design activity is concentrated on manually optimizing the RTL description of the circuit. After the RTL description is frozen, EDA tools are available to assist the designer in further processes. Designing at the RTL level has shrunk the design cycle times from years to a few months. It is also possible to do many design iterations in a short period of time.

Behavioral synthesis tools have begun to emerge recently. These tools can create RTL descriptions from a behavioral or algorithmic description of the circuit. As these tools mature, digital circuit design will become similar to high-level computer programming. Designers will simply implement the algorithm in an HDL at a very abstract level. EDA tools will help the designer convert the behavioral description to a final IC chip.

It is important to note that, although EDA tools are available to automate the processes and cut design cycle times, the designer is still the person who controls how the tool will perform. EDA tools are also susceptible to the "*GIGO : Garbage In Garbage Out*" phenomenon. If used improperly, EDA tools will lead to inefficient designs. Thus, the designer still needs to understand the nuances of design methodologies, using EDA tools to obtain an optimized design.

2. New EDA tools have emerged to simulate behavioral descriptions of circuits. These tools combine the powerful concepts from HDLs and object oriented languages such as C++. These tools can be used instead of writing behavioral descriptions in Verilog HDL.

1.4 Importance of HDLs

HDLs have many advantages compared to traditional schematic-based design.

- Designs can be described at a very abstract level by use of HDLs. Designers can write their RTL description without choosing a specific fabrication technology. Logic synthesis tools can automatically convert the design to any fabrication technology. If a new technology emerges, designers do not need to redesign their circuit. They simply input the RTL description to the logic synthesis tool and create a new gate-level netlist, using the new fabrication technology. The logic synthesis tool will optimize the circuit in area and timing for the new technology.
- By describing designs in HDLs, functional verification of the design can be done early in the design cycle. Since designers work at the RTL level, they can optimize and modify the RTL description until it meets the desired functionality. Most design bugs are eliminated at this point. This cuts down design cycle time significantly because the probability of hitting a functional bug at a later time in the gate-level netlist or physical layout is minimized.
- Designing with HDLs is analogous to computer programming. A textual description with comments is an easier way to develop and debug circuits. This also provides a concise representation of the design, compared to gate-level schematics. Gate-level schematics are almost incomprehensible for very complex designs.

HDL-based design is here to stay.[3] With rapidly increasing complexities of digital circuits and increasingly sophisticated EDA tools, HDLs are now the dominant method for large digital designs. No digital circuit designer can afford to ignore HDL-based design.

1.5 Popularity of Verilog HDL

Verilog HDL has evolved as a standard hardware description language. Verilog HDL offers many useful features for hardware design.

- Verilog HDL is a general-purpose hardware description language that is easy to learn and easy to use. It is similar in syntax to the C programming language. Designers with C programming experience will find it easy to learn Verilog HDL.

3. New tools and languages focused on verification have emerged in the past few years. These languages are better suited for functional verification. However, for logic design, HDLs continue as the preferred choice.

- Verilog HDL allows different levels of abstraction to be mixed in the same model. Thus, a designer can define a hardware model in terms of switches, gates, RTL, or behavioral code. Also, a designer needs to learn only one language for stimulus and hierarchical design.
- Most popular logic synthesis tools support Verilog HDL. This makes it the language of choice for designers.
- All fabrication vendors provide Verilog HDL libraries for postlogic synthesis simulation. Thus, designing a chip in Verilog HDL allows the widest choice of vendors.
- The Programming Language Interface (PLI) is a powerful feature that allows the user to write custom C code to interact with the internal data structures of Verilog. Designers can customize a Verilog HDL simulator to their needs with the PLI.

1.6 Trends in HDLs

The speed and complexity of digital circuits have increased rapidly. Designers have responded by designing at higher levels of abstraction. Designers have to think only in terms of functionality. EDA tools take care of the implementation details. With designer assistance, EDA tools have become sophisticated enough to achieve a close-to-optimum implementation.

The most popular trend currently is to design in HDL at an RTL level, because logic synthesis tools can create gate-level netlists from RTL level design. Behavioral synthesis allowed engineers to design directly in terms of algorithms and the behavior of the circuit, and then use EDA tools to do the translation and optimization in each phase of the design. However, behavioral synthesis did not gain widespread acceptance. Today, RTL design continues to be very popular. Verilog HDL is also being constantly enhanced to meet the needs of new verification methodologies.

Formal verification and *assertion checking* techniques have emerged. Formal verification applies formal mathematical techniques to verify the correctness of Verilog HDL descriptions and to establish equivalency between RTL and gate-level netlists. However, the need to describe a design in Verilog HDL will not go away. Assertion checkers allow checking to be embedded in the RTL code. This is a convenient way to do checking in the most important parts of a design.

New verification languages have also gained rapid acceptance. These languages combine the parallelism and hardware constructs from HDLs with the object oriented nature of C++. These languages also provide support for automatic stimulus creation,

checking, and coverage. However, these languages do not replace Verilog HDL. They simply boost the productivity of the verification process. Verilog HDL is still needed to describe the design.

For very high-speed and timing-critical circuits like microprocessors, the gate-level netlist provided by logic synthesis tools is not optimal. In such cases, designers often mix gate-level description directly into the RTL description to achieve optimum results. This practice is opposite to the high-level design paradigm, yet it is frequently used for high-speed designs because designers need to squeeze the last bit of timing out of circuits, and EDA tools sometimes prove to be insufficient to achieve the desired results.

Another technique that is used for system-level design is a mixed bottom-up methodology where the designers use either existing Verilog HDL modules, basic building blocks, or vendor-supplied core blocks to quickly bring up their system simulation. This is done to reduce development costs and compress design schedules. For example, consider a system that has a CPU, graphics chip, I/O chip, and a system bus. The CPU designers would build the next-generation CPU themselves at an RTL level, but they would use behavioral models for the graphics chip and the I/O chip and would buy a vendor-supplied model for the system bus. Thus, the system-level simulation for the CPU could be up and running very quickly and long before the RTL descriptions for the graphics chip and the I/O chip are completed.