CHAPTER-2
2.1 Overview 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 [23]. 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 oriented in 1983 at Gateway Design Automation. Later VHDL was developed under contract from DARPA. Using both 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 designers had to specify how the data flows between interconnections to implement the circuit were automatically extracted by logic synthesis tools from the RTL description [24]. 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 the 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.

A Hardware description language that describes the hardware of digital systems in a textual form. It resembles a programming language, but is specially oriented to describing hardware structures and behavior. It can be used to represent logic diagrams, Boolean expressions, and other more complex digital circuits. As a documentation language HDL is used to represent and document digital systems in a form that can be read by both humans and computers and is suitable as an exchange language between designers. The language content can be stored and retrieved easily and processed by computer software in an efficient manner. An HDL provides the framework for the complete logical design of the ASIC. There are two applications of HDL processing: Simulation and synthesis[25].

**Simulation:** A primary use of HDLs is the simulation of designs before the designer must commit to fabrication. Logic simulation is the representation of the structure and behavior of a digital logic system through the use of a computer. A simulator interprets the HDL description and produces readable output, such as a timing diagram, that predicts how the hardware will behave before it is actually fabricated. Simulation allows the detection of functional errors in a design without having to physically create the circuit. The stimulus that test the functionality of a design is called the Test bench.

**Synthesis:** Logic synthesis is the process of deriving a list of components and their interconnections (called net-list) from the model of a digital system described in HDL. Logic synthesis is similar to compiling a program in conventional high-level language. The difference is that instead of producing an object code, logic synthesis produces a database with instructions on how to
fabricate a physical piece of digital hardware that implements the statements described by the HDL code.

2.2 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 simplify input the RTL description to the logic synthesis tool and create a new gate-level net-list, 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 net-list or physical layout is minimized.

- Designing with HDLs is analogous to compare 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 say with rapidly increasing complexities of digital circuits and increasingly sophisticated EDA tools, HDLs are now the dominant method for large digital designs. No digital circuits designer can afford to ignore HDL-based design.
2.3 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 designers 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 net-lists from RTL 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.

- Formal verification and assertion checking techniques have emerged. Formal verification applies formal mathematical techniques to verify the correctness of the HDLs descriptions and to establish equivalency between RTL and gate-level net-lists. 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 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 any HDL languages. They simply boost the productivity of the verification process.

- For very high-speed and timing-critical circuits like microprocessors, the gate-level net-list 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 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.

### 2.4 Design flow in HDLs:

The design flow shown in figure 2.1 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. 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 RTL description to a gate-level net-list. A gate level net-list is a description of the circuit in terms of gates and
connections between them. Logic synthesis tools ensure that the gate-level
net-list meets timing, area, and power specifications. The gate-level net-list 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. EDA tools will help the designer convert the behavioral description to a final IC chip.

Figure 2.1: A Hierarchical design flow

2.5 Types of HDLs:

There are many proprietary HDLs that are supported by companies that design, or help in the design of the integrated circuits. There are two standard HDLs that are supported by IEEE (Institute of Electrical and Electronics
Engineers). VHDL and verilog HDL, which are most commonly used HDLs today. Both have constructs with which the design can be fully described at all the levels. There are additional constructs available to facilitate setting up of the test bench, spelling out test vectors for them and "observing" the outputs from the designed unit. IEEE has brought out Standards for the HDLs, and the software tools conform to them. Verilog as an HDL was introduced by Cadence Design Systems; they placed it into the public domain in 1990. It was established as a formal IEEE Standard in 1995. The revised version has been brought out in 2001. However, most of the simulation tools available today conform only to the 1995 version of the standard.

**HDL modeling capability:**

Hardware structure can be modeled equally effectively in both VHDL and Verilog. Figure 2.2 shows the modeling capability of HDL.

![Figure 2.2: HDL modeling capability](image)

When modeling abstract hardware, the capability of VHDL can sometimes only be achieved in Verilog when using the PLI. The choice of which is not only based solely on technical capability but also on personal preferences and EDA tool availability.
2.5.1 Verilog HDL:

Verilog is one of the two major Hardware Description Languages (HDL) used by hardware designers in industry and academia. From the modest beginning in early 1984 at Gateway Design Automation, the verilog has become an industry standard as a result of extensive use in the design of integrated circuit chips and digital systems. Until May 1990, with the formation of Open Verilog International (OVI), Verilog HDL was a proprietary language of Cadence. Cadence was motivated to open the language to the public Domain with the expectation that the market for Verilog HDL related software products would grow more rapidly with broader acceptance of the language. Cadence realized that Verilog HDL users wanted other software and service companies to embrace the language and develop Verilog supported design tools. Verilog came into being as a proprietary language supported by a simulation environment that was the first to support mixed-level design representations comprising switches, Gates, RTL, and higher levels of abstractions of digital circuits. Verilog HDL allows a hardware designer to describe designs at a high level of abstraction such as at the architectural or behavioral level as well as the lower implementation levels (i.e., gate and switch levels) leading to Very Large Scale Integration (VLSI) Integrated Circuits (IC) layouts and chip fabrication[22]. This handout does not cover all of Verilog HDL but focuses on the use of Verilog HDL at the architectural or behavioral levels. The handout emphasizes design at the Register Transfer Level (RTL). The simulation environment provided a powerful and uniform method to express digital designs as well as tests that were meant to verify such designs. Verilog is very C-like and linked by electrical and computer engineers as most learn the C-language in college[26].

There were three factors that drove the acceptance and dominance of verilog in the marketplace. First, the introduction of the programming language interface (PLI) permitted users of verilog to literally extended and customize the simulation environment. Since then, users have exploited the PLI and their success at adapting verilog to their environment has been a real winner for verilog.
Reasons for using Verilog HDL for description:
Verilog is the top HDL used by over 10,000 designers at such hardware vendors as Sun Microsystems, Apple Computer and Motorola.

1. The Verilog language provides the digital designers with a means of describing a digital system at a wide range of levels of abstraction, and at the same time, provides access to computer-aided design tools to aid in the design process at these levels.

2. Verilog allows hardware designers to express their design with behavioral constructs, deterring the details of implementation to a later stage of design in the design. An abstract representation helps the designer explore architectural alternatives through simulations and to detect design bottlenecks before detailed design begins.

Through the behavioral level of Verilog is a high level description of a digital system, it is still a precise notation.

2.5.2 VHDL (Very High Speed Integrated Circuit HDL):

VHDL, an acronym of VHSIC hardware description language is originated in the early 1980s and is now accepted as one of the most important standard languages for specifying and designing electronics. Today a number of high-tech companies work exclusively with VHDL for digital system. Universities run VHDL courses for their students and there are several VHDL groups to support new users. The requirements of the language were first generated in 1981 under the VHSIC program. In this program, a number of U.S. companies were involved in designing VHSIC chips for the Department of Defense (DoD) [23]. It was American defense department which initiated the development of VHDL in the early 1980s because the US military needed a standardized method of describing electronic systems. A team of three companies IBM, Texas Instruments and Intermetrics, were first awarded the contract by the DoD to develop a version of the language in 1983. Version 7.2 of VHDL was developed and released to the public in 1985. VHDL was standardized in 1987 by IEEE and ratified in December 1987 as IEEE1076-1987. The next standard, VHDL-93
doesn't contain any major changes from the old VHDL-87 standard except new VHDL commands and attributes for VHDL modeling. The great success of VHDL is due to the fact that it is the only hardware language which has been standardized.[27,28]  

**Reasons for using VHDL for description:**

Designing in VHDL offers advantages over traditional schematic design techniques.  

1. VHDL supports various development methods, top-down, bottom-up or any mix. The VHDL also supports modifiability, as the language is easy to read hierarchical and structured i.e the language is more flexible in terms of describing hardware.  

2. The language supports hierarchies reusable components error management and verification. This also supports concurrent and sequential language constructions.  

3. The design of VHDL components can be technology independent or more or less technology independent for a technical family. The components can be stored in a library for reuse in several different designs.  

4. The code for a VHDL component can be verified functionally in a simulator. The input signals are defined either in VHDL or in simulators language. When the VHDL code is simulated, functional verification takes place. At a later stage time verification of the design is also possible.  

5. In the case of traditional schematic design, the designer has to keep a manual check on technology-specific factors such as timing area, driving strength, component choice and fan-out.  

6. One of the great advantages of designing in VHDL is that the designer can concentrate on function i.e. implement the requirement specification and doesn't need to devote time and energy to technology-specific factors which do not affect function.  

7. There are four levels of abstraction such as Behavioral, structural(dataflow level), gate level and switch level to represent the same module offers flexibility of using all these abstractions in VHDL.
8. VHDL is standardized and this makes it possible to move code between different development systems for simulation. For a design it is harder to move the code because there is no standard.

VHDL is designed to fill a number of needs in the design process. Firstly it allows description of the structure of a design i.e. how it is decomposed in to sub-designs and how those sub designs are interconnected. Secondly it allows the specification of design using familiar programming language forms. Thirdly as a result, it allows a design to be simulated before being manufactured so that designers can quickly compare alternatives and test for correctness without the delay and expose of hardware prototyping.[29,30]

2.5.3 Design units:

VHDL description consists of primary design units and secondary design units. The primary design units are the Entity and package. The secondary design units are the architecture and the package body. Secondary design units are always related to a primary design unit. Libraries are collections of primary and secondary design units. A typical design usually contains one or more libraries of design units.

Entity: A VHDL entity specifies the name of the entity, the ports of the entity and entity related information. All designs are created using one or more entities i.e. the entity describes the interface to the VHDL model.

Package: A package declaration is used to store set of common declarations, such as components, types, procedures, and functions. These declarations can then be imported into other design units using a use clause.

Architecture: The architecture describes the underlying functionality of the entity and contains the statement that model the behavior of the entity. Architecture is always related to an entity and describes the behavior of the entity.

Package body: A package body is used to store the definitions of functions and procedures that were declared in the corresponding package declaration, and also the complete constant declarations for any deferred constants that appear in the package declaration. Therefore, a package body is always associated with a package declaration.