Chapter 1: VHDL Overview

1.1 Introduction

In the early part of the 1990’s there was an increase in the demand for electronic components in the continual growth of personal computers, cellular phones and high-speed data communications. Electronic vendors are providing devices with increasingly greater functionality, higher performance, lower cost, smaller packaging and lower power consumption. The trend for electronic design is continuing to provide complex designs and systems with more capability using fewer devices that take up less area on printed circuit boards (PCBs). Problems were encountered with increasingly complex electronic systems using the existing Electronic Design Automation (EDA) tools coupled with the accelerated time-to-market schedules. High-Density Programmable Logic Devices (HDPLDs) and Very High Speed Integrated Circuit (VHSIC) Hardware Descriptive Language (VHDL) became the key element for developing methodologies in handling complex electronic design.

PLDs include Complex Programmable Logic Devices (CPLDs) and Field Programmable Gate Arrays (FPGAs). These devices are used to integrate very complex designs with large amounts of logic into a single Integrated Circuit (IC). Semi-custom and full-custom Application Specific Integrated Circuits (ASICs) are also very useful in integrating digital, analog, mixed signal or system-on-a-chip (SOC) designs but are very costly and not schedule friendly. CPLDs and FPGAs provide more flexibility, cost less for development and can accommodate a design without compromising a schedule.

VHDL is an excellent tool for complex electronic designs using PLDs. With very large complex electronic system designs and large capacity CPLDs and FPGAs, it is very
difficult and almost impossible to design using Boolean equations or schematic capture containing gate level descriptions. VHDL provides high-level language constructs for design engineers to describe large circuits or systems. VHDL allows electronic designs or modules of the design to be stored as libraries allowing for use and portability to other designs. It is a standard language, IEEE Standard 1076 that is portable between simulation and synthesis tools. VHDL allows the design to be device-independent, targeting the technology and architecture towards the end of the design process and development. It allows for smoother transition of the design from PLD to ASIC.

It is very difficult to use traditional design methods, such as Boolean equations or schematic capture, on very large complex electronic designs. Generating equations and tracking errors using Boolean functions is time consuming and prone to making mistakes. Schematic capture does have some advantages. It provides a graphical view of the design and divide-and-conquer techniques with the use of schematic hierarchy. Schematic capture has the following drawbacks:

- Control logic must be designed with traditional techniques.
- Schematics can be difficult to maintain because the intent of the design gets lost in the complexity and detail.
- Schematics must be accompanied with documentation to scribe the functionality of the design with respect to the application.
- Portability is sometimes a problem with schematic capture. It is difficult going from one project using one design tool and importing that to another tool on another project. VHDL is tool independent.
Simulation environments for schematic capture design may not be the same for system level electronics making the verification of performance difficult.

Designing using VHDL increases the efficiency of the designer. VHDL designs facilitate capturing, understanding, maintaining and providing well-defined designs. VHDL is a standard and can be used on many different EDA environments and modules can be reused from design to design. It supports hierarchy, gate-level designs and system level designs and is used for description, simulation and synthesis.

There are currently two main Hardware Descriptive Languages, VHDL and Verilog. Verilog syntax is not as complicated as VHDL and is less verbose. It lacks features and capabilities that VHDL can provide. Verilog is easier to grasp and understand and its constructs are based on 50% C programming and 50% ADA. EDA environments support them both with documentation, simulation and synthesis capabilities. VHDL for this design was an excellent choice.

VHDL was started by the VHSIC program, which was funded by the Department Of Defense (DOD) in the 1970s and 1980s. It was developed to document complicated circuits so that one contractor could understand the designs of another. It was also developed as a modeling tool so software programs could be used to simulate complicated designs. In 1987, VHDL was established as the IEEE 1076 standard. In 1988, MilStd454 stated that all ASICs that were to be delivered to DOD be designed using VHDL. In 1993, IEEE standard was updated and VHDL standard, IEEE 1164 was initiated. Lastly, in 1996, IEEE 1076.3 became the VHDL synthesis standard. VHDL is the industry standard for electronic design description, simulation and synthesis. The
driving force behind the growth of VHDL is synthesis. Synthesis is the reduction of a VHDL design description to a lower level of circuit representation.

1.2 Advantages of Using VHDL

Because VHDL is an industry standard, it is critical that every electronic design engineer learns and uses it to implement designs. This allows the designer to keep up with the current electronic design methodologies, and allows companies to be efficient and competitive. VHDL allows designers to quickly describe and synthesize a design that may occupy up to 20 thousand gates in a PLD in a matter of minutes, where it would take several weeks or months to perform the same task using schematic capture or any other method. The following are a set of advantages of why designers should use VHDL:

- **Power and Flexibility**

  VHDL is a very powerful language that provides constructs that are compact expressions describing very complex circuits without having any wasted words in the code. VHDL requires precisely coded models using defined and matching data types. It provides for multiple levels of design description allowing it to be partitioned for implementation and control. It supports design libraries containing designed modules and user designed modules that can be reused in other designs on other projects very efficiently. VHDL provides one language for design and simulation. A VHDL test bench can be used to simulate the design for functional and performance verification.

- **Device Independent**

  VHDL allows the designer to create the design without having a device selected. You can create the design and simulate it prior to ever selecting a CPLD, FPGA or ASIC technology. One design description can be targeted to any device architecture.
Defining and intimately understanding the device architecture is not critical in determining how well the design will perform or how it will utilize resources. VHDL also provides multiple styles of design description, which include netlists, Boolean equations, concurrent statements and sequential statements.

- **Portability**
  VHDL allows the designer to simulate the same set of code that gets synthesized to the specific device architecture. Simulating a design is very useful in finding errors in the code and can save a considerable amount of time if done prior to the design implementation stage. VHDL allows the electronic design to be described, simulated and synthesized on many different EDA tools and software packages. VHDL designs can be used on multiple projects and is not out of date if the design tool that created it is no longer available. The skills the designer gains on one tool can be used or transferred to other tools, because the design methodology is the same with minor differences. Figure 1.1 illustrates the portability of VHDL in the EDA environment.

- **ASIC Migration**
  The efficiency of VHDL allows a design to be implemented very quickly in a CPLD or an FPGA. VHDL allows for easy transition from a CPLD or FPGA design to an ASIC design. Many times the same VHDL code that was used in the CPLD or FPGA design is the same one implemented in an ASIC. Because VHDL is a well-defined language, ASIC vendors will deliver a functional device meeting the expected and predetermined specifications and performance requirements.
Figure 1.1  VHDL Portability Illustration

- **VHDL for Low Cost and Tight Schedule**

  VHDL, programmable logic devices and the EDA tools available these days allow for an efficient fast design cycle. VHDL permits very large complex designs to be completed quickly. PLDs reduce Nonrecurring Engineering (NRE) costs and allow for quick turn-around with multiple design iterations. Synthesis allows the two to be joined for a fast and effective methodology.

1.3 **VHDL Shortcomings**

  Designers have three concerns about implementing a design using VHDL and they are as follows:

  1. The designer does not have control of the gate level implementation of the design due to high level abstract constructs.

  2. The logic implementation of the design by synthesis tools is sometimes inefficient.

  3. The quality of the synthesis tools varies from one tool to another.
The very first concern is hard to get around, because the intent of VHDL was to relieve designers from having to specify gate level implementation of a circuit. Compilers that synthesize logic are designed to optimize most constructs eliminating the need for the designer to specify. Many synthesis tools allow the designer to use synthesis directives providing limited control over the implementation and optimization of the logic. The synthesis directives force the compiler to optimize the design in terms of area-efficiency or a speed-efficiency. Many synthesis tools allow the designer to specify gate level implementation that is technology specific based upon the internal architecture of logic cells and routing resources.

The second problem is that logic synthesis is inefficient. Synthesis tools do not always offer the optimal solution for a logic function because the solution depends on the design objectives. Compilers use algorithms to implement logic in a standard way not always looking at the design problem in a specific way. One solution to the problem is to code the design in a way that the designer has control over the implementation process. Inefficient VHDL code results in redundant logic resulting in a decrease in performance.

The last shortcoming for VHDL addresses the quality of the synthesis tools. Fortunately, VHDL synthesis technology for CPLD and FPGA designs is becoming standard and well beyond the infant stage. This shortcoming is becoming less of an issue.

These shortcomings do not take away form the advantages of implementing a design using VHDL. Design engineers need to keep in mind the design objectives and not to be concerned with the details of the design implementation. The design objectives are to fulfill the design requirements, within cost and schedule.
1.4 VHDL Design Process

The VHDL design process for FPGAs and CPLDs consists of six steps. The first step is to identify the design requirements for the system or subsystem. The detailed design specifications follow from the requirements outlining the circuit performance, implementation and interfaces. The second step is to describe the design, which includes formulating and coding the design. Simulating the code is the next step. This aids in the elimination of errors in the code and provides the first assessment on the performance of the system and how well it has met the specifications. Synthesis, optimization and fitting the design are the next step followed by post layout simulation. Post layout simulation allows the designer to verify the functionality of the design and the critical timing. Finally, the device is ready to program. Figure 1.2 contains a block diagram of the VHDL design process.

1.4.1 Define the Design Requirements

It is important when designing an electronic system to have a clear idea of the design objectives. The design objectives include the overall function of the design, the required internal setup time, maximum operating frequency, interface requirements, clock to output times and critical paths. Having an idea of the requirements helps determine the partitioning of the design and preliminary device architecture.
1.4.2 Describe the Design in VHDL

The first part in describing the VHDL design is formulating the design. Formulating the design is deciding the design methodology. The methodologies include top-down, bottom-up or flat. The first two methodologies involve creating design hierarchies and the later is a monolithic design. Designing the methodology aids in writing fast efficient code that can be synthesized for optimal logic. The top-down approach requires that the design be broken down into functional components, each with specific inputs and outputs. The top-level design module ties the lower designs together.
to implement the logic for a specific function. Once the top-level is designed, the lower level components are the designed. The bottom-up approach involves designing the individual components and connecting them together to form the overall system. The flat design contains the functional components at the same level as the interconnection of those components. The flat methodology works well for smaller designs, where getting caught up in the functional details of the components does not distract from the system level design. Hierarchy works well for large complex designs partitioning them into multiple functional components. It is important not to partition the design into too many components, because it increases the complexity of the inter-module connections.

Once the design methodology has been chosen the code can be written, using as a reference, block, state, or data-flow diagrams. Most engineers use existing code that can be modified to implement the new functional component. This saves time and reduces on syntax mistakes. When writing the VHDL code it is critical to think in terms of the hardware, trying to understand how the synthesis compiler will realize your code. This will help to increase the efficiency of the design and to decrease the unneeded, repetitive or non-optimal logic.

1.4.3 Simulate the Design

For very large and complex designs it is time efficient to simulate the code. This allows for errors to be detected early in the design process so that corrections can be made with the least possible impact to the schedule. Hierarchical designs can consist of many sub-designs that can all be simulated and functionally verified before integrating into the entire function or system. Designers should not spend much time on initial simulation because post synthesis simulation is the final stage in determining if the
design performance has been met. If it has not, the design will need to be modified and simulated again.

1.4.4 VHDL Design Synthesis

Synthesis is the reduction of the code description to a lower level circuit representation. The synthesis process creates netlists or equations form the abstract design description. VHDL synthesis software converts the VHDL description into a technology specific netlist or set of equations. The tool usually provides a set of equations for a CPLD targeted technology and a netlist for a FPGA.

Synthesis is technology specific because each PLD vendor contains devices with different architectures including the logic module, I/O modules and the routing resources. Figure 1.3 illustrates the VHDL synthesis process. Before synthesis occurs, a software toll must first check the code for syntax errors. The synthesis process converts the abstract design into a Register-Transfer Level (RTL) description. The device architecture determines the specific RTL elements. Some of the synthesis tools will identify operators and their operands and determine if they can be replaced with technology specific optimized components. The rest of the code that is not converted to the known technology specific components will be expressed in terms of Boolean equations and not yet optimized.

1.4.5 Optimization

The optimization process depends on the form of the Boolean equations, type of resources available and the user defined synthesis constraints. Some forms of expressions can be converted into logical resources more efficiently than others. For example, minimal sum of products is efficiently mapped into PAL architecture and
canonical sum of products are best suited for a RAM-based architecture. These architectures will be discussed in Chapter two, Programmable Logic Devices. User constraints are used to optimize expressions for the available resources. Constraints are used to reduce signal loading, reduce fan-in or reduce the number of terms in an expression.

Figure 1.3  VHDL Synthesis Process

Optimization for CPLDs consists of reducing the logic to a minimum sum of products. This can further be optimized for a minimum literal count. This approach reduces the product term utilization and the number of inputs to a specific logic block required for an expression. These optimized equations are then passed to the fitter. Optimization of an FPGA requires equations that are factored based on device specific resources. Factors are evaluated for the most efficient implementation. Criteria is used
to determine the existing set of factors or to factor the system of equations differently. The criteria consist of factor sharing.

1.4.6 Fitting or Place and Route

Fitting is the process of taking the output produced by the synthesis tool and placing it into the design. Fitting is usually the term used for CPLD-based architectures and placing and routing is used in FPGA logic structures, where logic cells are placed in optimal locations and then routed between each other and to the I/O logic cells.

Fitting in a CPLD is difficult because there are many ways in which a design or logic function can be placed in the device. Before fitting occurs the logic equations are further optimized depending upon the available logic, I/O and interconnection resources. Once the equations are transformed, the expressions that share resources are grouped together. The shared resources can be resets, presets, clocks, output enables and inputs to macrocells. Pin assignments can also determine what expressions get grouped together. The group of expressions is then evaluated to see if they can fit into a logic block. If they cannot, then the expressions are adjusted until successful. Once the expressions are optimized into the logic blocks, an initial placement is attempted. If it fails, the grouping will have to be done again. If the placement of the logic blocks is successful, then the routing can take place. Routing occurs between the inputs and the outputs of the logic blocks and the I/O blocks. If the routing cannot be completed it may have to go back and try a different grouping. If it is still not successful then the design may have to be targeted to a larger device more capable of handling the design.

Place and route tools have a significant effect on how well the design will perform. Total Propagation Delays (Tpd) depending largely on the routing efficiency.
good place and route will group the time critical components together in order to optimize the delays. Place and route tools use algorithms, directives, user constraints, and performance estimates to perform the initial placement. Once the initial placement is done, algorithms can fine tune to meet the performance requirements of the design. Routing can then begin with global interconnections for long signals or signals having high fan-out. Local routing is then performed to connect logic cells to I/O logic cells.

There are many synthesis tools available. Some include Synopsys, Cadence, Mentor Graphics, Viewlogic, Data I/O and Synplicity. Many of the FPGA and CPLD vendors provide their own tools with proprietary algorithms for synthesizing logic. This results in implementation varying from one tool to the next.

1.4.7 Simulate the Post-Layout Design Model

Even if simulation was done before synthesis, it is important to perform post-layout simulation to verify that the design is functioning and meeting its timing requirements. If the design is not meeting the timing specifications, it may have to be redesigned or synthesized and fitted to a new logic device with a faster speed grade. It may also require reviewing the VHDL code to verify that it was written efficiently.

1.4.8 Programming the Device

Once the design has been formulated, simulated, synthesized, optimized, fitted and re-simulated it is ready to be programmed into a device. The synthesis, optimization and fitting software provides a file as an output to program the device.

1.5 VHDL Chapter Summary

VHDL provides high-level language constructs for design engineers to describe very large complex circuits for implementation into CPLDs, FPGAs, and ASICs, while
providing capability for the design to migrate to ASICs. It is an industry standard for electronic design description, simulation and synthesis and the driving force for current design methodologies in Electronic Design Automation (EDA) tools. VHDL advantages include; power and flexibility to use one source for design and simulation; device independent allowing the design to be functionally completed prior to the device being selected; portability from one EDA tool to the next and reduces NRE costs while providing quick turn-around time on products. VHDL does have a few problems, however, the problems do not take away from the efficiency that VHDL provides in meeting the design objectives, fulfilling the design requirements and meeting cost and schedule. The design process for implementing a VHDL design in FPGAs and CPLDs include defining the requirements, simulating the design, synthesizing the design, post layout simulation, and finally programming the targeted device. The detailed syntax and VHDL constructs will not be outlined. Chapter 6 contains the VHDL design methodology and a detailed description of how the code is designed to meet the requirements. Chapter 7 revisits the VHDL design process and how it relates to the actual tools used to implement and verify the design.