Laboratory Development for a VHDL Design Course

George H. Zion
Electrical, Computer, and Telecommunication Engineering Technology
Rochester Institute of Technology
Rochester, NY 14623

Abstract

Due to the proliferation of highly integrated programmable logic devices, (PLD, CPLD, and FPGA), the traditional methods for performing digital logic design has given way to a development process that involves extensive use hardware descriptive languages. In industry, the two languages that have become prominent are VHDL and Verilog.

This paper provides a brief overview of the VHDL hardware descriptive languages and discusses the course and laboratory development for a VHDL design course for the Computer Engineering Technology program at Rochester Institute of Technology.

VHDL – A Brief History

First, what is VHDL? VHDL stands for Very High Speed Integrated Circuit (VHSIC) Hardware Descriptive Language. VHDL was developed in the late 1970's early 1980's under the direction of the Department of Defense as a means to document complex logic designs. Later, VHDL evolved into a simulation language for large designs targeted for ASICs. Finally, with the proliferation of low cost, quick time-to-market programmable logic devices (PLD, CPLD, FPGA) VHDL has become an industrial standard for logic synthesis. VHDL was standardized in 1987 (IEEE-1076) and has since been updated twice, first in 1993 (IEEE-1164) and again in 1996 (IEEE-1076.3). The 1996 update standardized VHDL as a synthesis language. Today, VHDL, along with Verilog, are industry standards for logic simulation and syntheses.

VHDL Design Process

Prior to delving into an overview of the VHDL language, it is important to first understand the overall VHDL design process. A flow chart for the design process is shown in figure #1. The process begins with the design entry, which, for this discussion will be limited to VHDL source code. Many development tools also support schematic entry and state graph editor design entry. Following the design entry is the compiler. Like any programming language, the compile performs a syntactical check of the source code. The output of the compiler is used as input to the synthesizer as well as a pre-layout simulator. The pre-layout simulator is a very effective tool in validating VHDL design early in the design process. Pre-layout simulation is most beneficial in large VHDL designs where the end hardware will be an ASIC and the impact to cost and scheduling due to a design error can be catastrophic. The benefits of pre-layout simulation for small to medium size designs where the end hardware will be a programmable logic device are limited because post-layout simulation will still need to be performed to validate the hardware implementation. The next step in the design process is to synthesize the VHDL source code into a gate-level netlist, which is then implemented via a fitter (PLD/CPLD) or via a place & route process (FPGA). Following implementation, a post-layout simulation is used to validate the

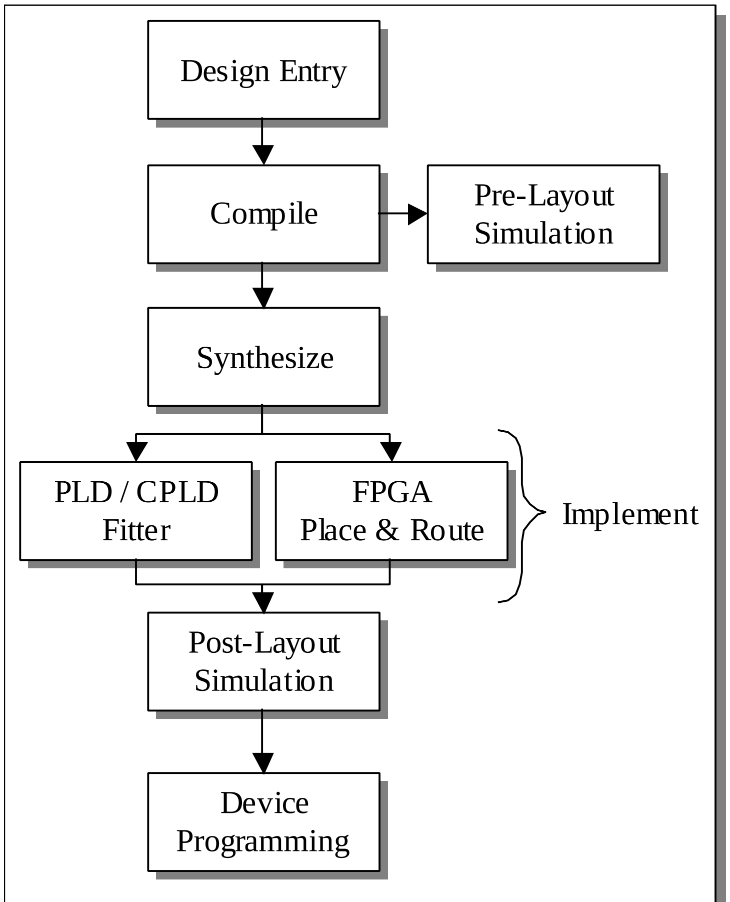


Figure #1 : VHDL Design Process

hardware implementation of the VHDL design. The final step in the design process is to program the programmable logic device.

Overview of VHDL Model

All VHDL models consist of an entity and architecture pair (see figure #2). The entity declaration describes the interface signals along with their corresponding data type and mode (i.e. in, out, inout, buffer). The entity describes the circuit as it is seen from the outside with no reference to how the model is implemented. The architecture body describes the functionality of the model. The architecture body can be implemented with one or more of the three VHDL modeling techniques. These three modeling techniques are *structural*, *dataflow* and *behavioral*. For the *structural* modeling technique, as the name implies, the structure of the final design is captured in the VHDL code. For example, if the final design contained a D flip flop, the VHDL code would include a structure of a D flip flop. Structural modeling, though closest to the hardware, does not allow the designer to take advantage of many of the language constructs that makes VHDL the powerful tool that it is. The second modeling technique is *dataflow*. In dataflow modeling, the VHDL code describes the circuit in terms of how data is transferred from register to register as the data is moved from the input to the output. Dataflow modeling is often referred to as *register transfer logic*, and is very effective for implementing combinational logic. The final modeling technique is *behavioral*. Using behavioral modeling, the designer is not concerned with correlating the VHDL code with the final hardware, but rather capturing the behavior of the high level design. Behavioral modeling is the most powerful modeling technique yet is the most abstract and distant from the final hardware

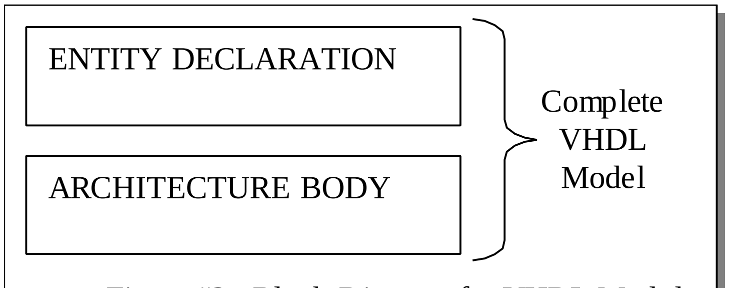


Figure #2 : Block Diagram for VHDL Model

A Design Example

Figure #3 shows a simple logic circuit consisting of a NAND gate, a D Flip-Flop and Tri-State output buffer. This circuit will be used to illustrate the development of a complete VHDL model. The dotted line placed around the components is an effective way to show the division of a VHDL entity and its accompanying architecture.

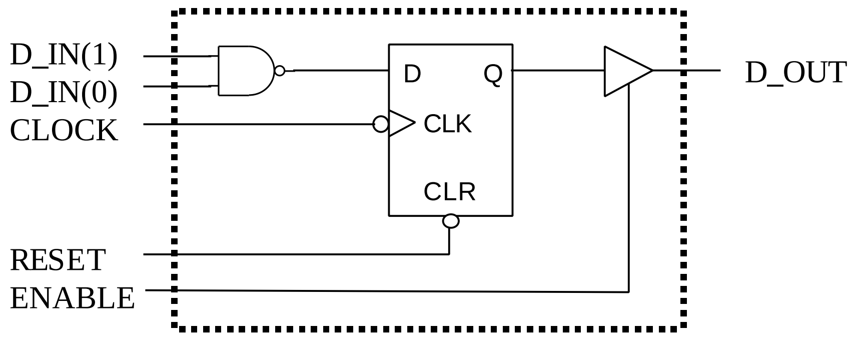


Figure #3 : Design Example Block Diagram

All the signal names around the parameter of the dotted line are the interface signal of the entity declaration (see figure #4). This entity declaration is relatively self-explanatory, all the inputs are of mode in, the output is of mode out, and all the signals are of type ***STD_LOGIC*** or ***STD_LOGIC_VECTOR***.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity EXAMPLE is
   port (D_IN      : in   STD_LOGIC_VECTOR (1 downto 0);
         CLOCK     : in   STD_LOGIC;
         RESET     : in   STD_LOGIC;
         ENABLE    : in   STD_LOGIC;
         D_OUT     : out  STD_LOGIC);
end EXAMPLE;
```

Figure #4 : VHDL Entity

But what is ***STD_LOGIC*** ? As previously mentioned, VHDL was originally standardized in 1987 (IEEE-1076), and updated in 1993 (IEEE-1164). One of the most significant changes made in the 1164 update was to add the base data type ***STD_LOGIC***. In the original 1076 standard, signals of type ***BIT*** could only be assigned a value of '0' or '1'. For anything other than very simple designs, this was inadequate. Along with, '0' & '1', the ***STD_LOGIC*** data type allows signals to be assigned other values including tri-state ('Z'), un-initialized ('U') and don't care ('-'). In order to have access to the STD_LOGIC data type, the IEEE.STD_LOGIC_1164 library must be include prior to the entity declaration. For the architecture body, the functionality of the logic within the dotted line needs to be captured (see figure #5). For the purpose of edification, all three modeling techniques were used for this example. First, the NAND gate is implemented structurally; next the tri-state buffer is implemented with the dataflow technique, and finally the flip-flop is implemented via the behavioral technique. Note, because all statements within a VHDL architecture body are concurrent, the order that they appear in the architecture does not matter. Shown in figure #6 is an annotated timing analysis for this VHDL design implemented in a Xilinx XC9536-5 PC44.

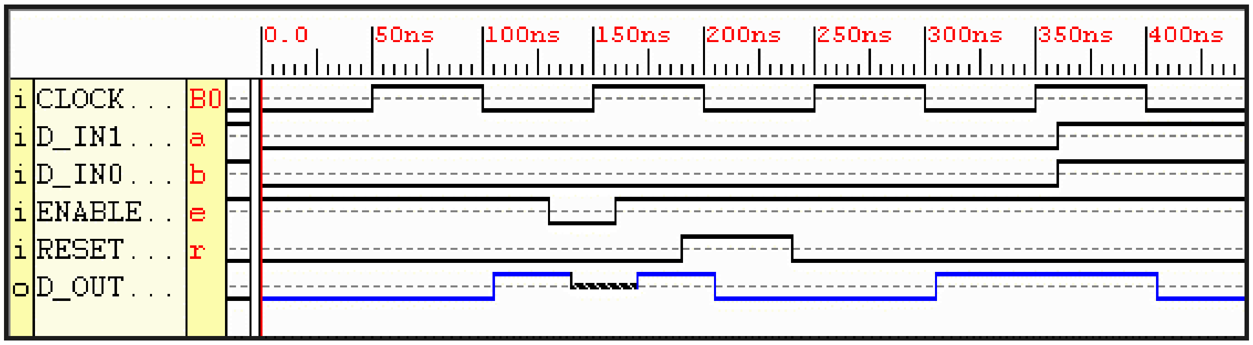
```
architecture EXAMPLE_ARCH of EXAMPLE is
   signal TEMP_IN, TEMP_OUT : STD_LOGIC;
begin
   -- Code for input NAND gate
   TEMP_IN <= D_IN(1) NAND D_IN (0);

   -- Code for Tri-State output buffer
   D_OUT <= TEMP_OUT when ENABLE = '1' else 'Z';

   -- Code for Negative Edge Trigger Flip-Flop
   -- with active high asynchronous reset.
   process (CLOCK, RESET)
   begin
     if RESET='1' then
        TEMP_OUT <= '0';
     elsif (falling_edge(CLOCK)) then
        TEMP_OUT <= TEMP_IN;
     end if;
   end process;

end EXAMPLE_arch;
```

Figure #5 : VHDL Architecture



Time	Event
0 nSec	D_IN1 and D_IN0 set to ‘0’
100 nSec	Falling edge of clock
105 nSec	D_OUT changes to ‘1’; Delay equal to typical t_{PD} of 5 nSec for CPLD
130 nSec	ENABLE set to ‘0’ (inactive state)
140 nSec	D_OUT changes to ‘Z’ (tri-state); Delay equal to typical t_{POD} of 10 nSec for CPLD
160 nSec	ENABLE set to ‘1’ (active state)
170 nSec	D_OUT changes to ‘1’; Delay equal to typical t_{POE} of 10 nSec for CPLD
190 nSec	RESET set to ‘1’ (active state)
205 nSec	D_OUT changes to ‘0’; Delay equal to typical t_{POR} of 15 nSec for CPLD
200 nSec	RESET set to ‘0’ (inactive state)
300 nSec	Falling edge of clock
305 nSec	D_OUT changes to ‘1’; Delay equal to typical t_{PD} of 5 nSec for CPLD
360 nSec	D_IN1 set to ‘1’ & D_IN0 set to ‘1’
400 nSec	Falling edge of clock
405 nSec	D_OUT changes to ‘0’; Delay equal to typical t_{PD} of 5 nSec for CPLD

Figure #6 : Timing Analysis of Design Example

Course Material

The RIT design course that uses VHDL, titled Principles of Electronic Design Automation (PEDA), has been taught since 1996. The course is required for all Computer and Electrical Engineering Technology students and is taken at the start of their third year, prior to their first CO-OP work block. Prior to taking this course the students are required to take two quarters of digital electronics, one quarter each of C++ programming and an introductory microprocessor course. Subsequently, the Computer Engineering Technology students are required to take a three-course sequence in Embedded Systems Design where VHDL is heavily utilized. The Electrical Engineering Technology students can take the Embedded sequence as their technical electives.

The underlying philosophy of this course is simple, we are teaching the design process for the development of quality logic designs where a hardware descriptive language (HDL) is being used as a means to this end. The HDL and the specific development tool being used are not really that important, understanding the design process and methodologies is. Hopefully you have noticed that throughout this paper no VHDL development tool[1] has been mentioned and Verilog was mentioned only in passing. This was not unintentional. We could just as well have implemented this course using Verilog with any number of development tools. When this course was first developed, the rationale used for selecting VHDL over Verliog had less to do with the merits or features of the languages then they did with the availability of low cost development tools. Unlike many courses and textbooks that focus on creating VHDL simulation models, this course focus on VHDL development for programmable logic synthesis with a heavy emphasis placed on the quality of the hardware created.

Topic	Time
Introduction to VHDL	One Week
• History of HDL / Rational For Use	
• Overview of CPLD/FPGA	
• VHDL Structure and Syntax	
• VHDL Design Process	
Combinational & Synchronous Logic Design	Two Weeks
• Behavioral	
• Dataflow	
• Structural	
State Machine Design	One Week
• Design Methodology	
• Synchronous & Asynchronous Inputs	
• Resource Allocation	
Design Hierarchy	One Week
• Components & Packages	
• Function & Procedures	
Design Examples	One Week
CPLD vs. FPGA Design Implementations	One Week
Design Optimizing & Verification	Two Weeks
• Pipelining	
• Resource Sharing	
• Test Vectors	
• Algorithm	
Examinations, Homework and Laboratory Review	One Week

Table #1 : Course Outline

The current course topics and time allocations are given in table #1. Since its inception, this content has remained relatively consistent with only minor changes. This consistency cannot be said for the focus and content of the laboratories. Initially, the weekly laboratory assignments where used to reinforce specific VHDL features or hardware principles and then were followed by a multi-week quarter ending project. This bottom-up approach was met with limited success.

[1] For the record, Xilinx's Foundation Series version 3.1 is currently being used.

Many students who could easily perform the weekly assignments, had difficulty synthesizing the concepts as needed to implement a design of any significant magnitude. Greater success has been achieved via a combination of weekly homework assignments and several multi-week laboratory projects. The homework assignments allow for specific VHDL concepts and hardware principles to be emphasized and the larger laboratory projects repeatedly force the application of these concept and principles. The laboratory projects, which in the past included a home burglar security system, multi-lane traffic light controller, and a simple ALU, are constantly changing, but the underlying principles that they stress are always present. These principles are :

- Process : Languages and development tools come and go, the underlying process stays the same.
- Hardware Focus: Hardware utilization needs to be stressed for every design example and laboratory project. Projects need to push the limits of the hardware device to force the issues of design efficiency.
- Software NON-Focus: Students need to be constantly reminded that they are designing hardware, NOT writing software.
- Design Verification: All designs need to be thoroughly tested through software simulation and hardware implementation.

Summary

Hardware descriptive languages, particularly VHDL and Verilog, have become the industrial standard for implementing digital logic using programmable logic devices. Students educated in the use of an HDL where the emphasis is placed on the design process rather than the language have found, and will continue to find, great success in industry.

Bibliographic Information

[1] Skahill, Kevin; VHDL for Programmable Logic, Addison-Wesley, Menlo Park CA, 1996. ISBN 0-201-89573-0

[2] Pellerin, David; VHDL Made Easy !, Prentice-Hall PTR, Upper Saddle River NY, 1997. ISBN 9-13-650763-8

[3] Armstrong, James; VHDL Design Representation and Synthesis, Prentice-Hall PTR, Saddle River NY, 2000.

Biographical Information

George Zion is the Program Chair of and a Professor in the Computer Engineering Technology program in the College of Applied Science and Technology at Rochester Institute of Technology. He received a BSEET in 1984 and an MSCS in 1988 from Rochester Institute of Technology. His teaching interests are VHDL, embedded systems design and embedded C/C++ programming. He is a member of ASEE and the Rochester Engineering Society.