

Embedded System Design with FPGAs using HDLs (Lessons learned and pitfalls to be avoided)

R James Duckworth
Electrical & Computer Engineering
Worcester Polytechnic Institute
Worcester, MA 01609
rjduck@wpi.edu

Abstract

This paper describes the authors experience with teaching VHDL (and more recently, Verilog) to undergraduate and graduate students at WPI and to engineers through various short courses in industry. The courses have concentrated on logic synthesis targeting CPLDs and FPGAs. All the courses had a major lab component where students could use simulation tools to test their design using test benches and also synthesis tools to synthesize, implement, and download the configuration bit-stream to evaluation boards. The paper concentrates on some of the problems that students encounter when they are trying to design their digital systems using HDLs. In addition to the conceptual issues with using a language to describe hardware behavior, the newer higher capacity devices provide additional challenges as more functions, including embedded processors, are added to the device.

1. Introduction

This paper summarizes the main issues that seem to cause problems for students when they are learning to use VHDL or Verilog to describe hardware for the first time. While there have been many books written that cover VHDL (and to a lesser extent Verilog), most seem to confuse students more than help them and invariably end up mixing constructs that are only suitable for synthesis with other language capabilities that should only be used for simulation. Having used VHDL on many research projects and taught many courses over the years the author has adopted a plan to try to alleviate the most common problems for students. In the examples given below it should be noted that reference is made to VHDL, however the principles apply just as well to other HDLs such as Verilog.

2. Synthesis or modeling first?

Most text books start out by describing the use of VHDL for modeling digital components and systems by using many features of the language including adding in time related information. Instead of this approach, the author has always started by introducing and only covering the subset of VHDL suitable for synthesis. This approach is further helped by providing standard templates and vendor recommendations for describing combinational and sequential logic, including state machines. Only after VHDL for synthesis has been thoroughly explained, and multiple examples and designs attempted, are the additional keywords and constructs covered that are appropriate for testing, simulation and modeling. This approach seems to minimize confusion and stops students from trying to use inappropriate statements when they are using VHDL for synthesis.

3. VHDL is not a programming language.

When introducing VHDL, it is very important to keep emphasizing the fact that the VHDL code is only *describing* the required behavior of the digital circuit or system and is not being executed in some way by a 'hidden' interpreter or microprocessor on the FPGA. A clear distinction needs to be made between a VHDL description that will be *synthesized* by the tools to produce a hardware implementation and the same description that will be *simulated* on a conventional computer. It is useful to describe the hardware features of the target device (for FPGAs, this would be look-up tables for combinational logic, and flip-flops for sequential circuits) and to explain that the VHDL description will be analyzed, compiled, and synthesized to these hardware components, and the bit-stream sent to the device is just a configuration file. Unfortunately, with the recent capability to routinely

add microprocessors to an FPGA as part of the design, the potential for major confusion is possible!

4. Why doesn't the design work?

Another problem that frequently arises is with trying to debug a design that doesn't work. Many students, having written many programs using conventional languages such as 'C' or Pascal, can usually modify (or hack if we are being brutally honest about what really happens) a program until it performs the required function. With VHDL for synthesis this is not an option. In fact the more one tweaks, or modifies, a VHDL description, the more likely the synthesis tools will complain, essentially producing error messages that they don't have a clue what type of hardware is trying to be created. In many cases it is best to totally scrap an approach that is not working and to start the design description over. However, being able to put this statement into practice can require substantial maturity as an engineer!

There are a number of reasons for this problem. If a syntax error is found in a 'C' program then the compiler can usually point straight to the specific construct or statement line in error. However when writing VHD for synthesis, it is not only necessary to meet the VHDL syntax rules, but also to make sure that the statements are carefully applied so that the tools can infer the correct logic. This usually means that multiple VHDL statements are analyzed together to deduce if the necessary logic is a simple combinational circuit, a sequential circuit, or some type of state machine. When a problem is identified, the analyzer can usually only provide a general indication of what or where the problem is. Students may spend many hours fruitlessly trying to find a problem with a statement on one particular line when in fact the error or errors may lie elsewhere.

Another common problem is when students try to place their design into the target device. They create the necessary pin constraint file to map the signals to the device pins (in Xilinx this is a user constraint file - UCF). However when the design is analyzed, simplified, and subsequently implemented by the tools it is possible that due to omissions in describing the required behavior the tools correctly deduce that a particular output is not connected and so is optimized away. Although this is usually reported in a warning message it is not always obvious what the cause is. Again students may look for the error in totally the wrong place.

A third common misunderstanding is with defining combinational circuits. First it is sometimes necessary to remind students what a combinational circuit is and the fact that it should not contain any latches or flip-flops. It is quite common for students to fail to specify the output values for all possible input combinations resulting in unexpected latches and incorrect behavior. A possible remedy is to ask the students to put themselves in the place of the compiler – would they be able to correctly design a digital circuit based on the provided VHDL code?

It should also be stressed that VHDL does not provide a magic short-cut to designing digital systems. It is still essential that engineers understand the basic principles of digital logic design and they have a good understanding of the type of logic that should be produced. When writing a 'C' program, most programmers do not worry about or even look at the intermediate assembler or final machine code produced. However it is essential that designers review resource usage when using VHDL to make sure that the resulting design matches expectations and no silly mistakes were made. The author has always made it a requirement of lab reports that students document the resources used for their design. For example, they need to justify and explain why their design resulted in a particular number of flip-flops. It is also helpful to review the RTL schematic representation to check that the general hardware structure matches expectations.

5. System on an FPGA

With the introduction of the lower cost but larger capacity FPGAs it is now possible to add embedded microcontrollers to the FPGA along with other more conventional digital logic. This makes the development and management and debugging of the device even more challenging. Using a very simple embedded processor such as the Xilinx Picoblaze 8-bit microcontroller seems like the appropriate starting point. At the same time it is also becoming easier to add DSP functions in the form of customizable building blocks to the FPGA allowing the concept of a system on-a-chip solution to be realizable for many student projects.

ACKNOWLEDGMENTS

We gratefully acknowledge the donation of software and hardware from Xilinx over many years that has enabled us to educate our students in this important area of digital system design.