AC 2007-1050: PEDAGOGIC CONSIDERATIONS FOR TEACHING DIGITAL SYSTEM DESIGN USING VHDL

Chia-Jeng Tseng, Bucknell University

Chia-Jeng Tseng is with the Department of Electrical Engineering at Bucknell University. His current research focuses on the development of digital system design methodologies and digital signal processing algorithms.

Pedagogic Considerations for Teaching Digital System Design Using VHDL

Abstract

Over the last four years, system-level design methodologies have been taught in an "Advanced Digital Design" course at Bucknell University. VHDL is used to define the functions and structures of a digital system. The writing of a hardware description is very different from writing a program for software applications. Effective teaching of a hardware description language such as VHDL is a challenging task. To improve the effectiveness of teaching digital system design using VHDL, numerous pedagogic considerations have been taken into account. In this paper major pedagogic considerations including course organization and materials are described. Student feedback was collected and analyzed; the effectiveness of each course module is reviewed. Common mistakes and general guidelines of writing VHDL descriptions for synthesis are also presented.

1. Introduction

Two digital design courses are offered at Bucknell University: one is entitled "Digital System Design" and the other is called "Advanced Digital Design." Both courses consist of three hours of lectures and laboratories weekly. Digital System Design, offered to the junior class, focuses on logic synthesis; schematic capture is used for design entry. "Advanced Digital Design," offered to senior and graduate students, addresses system-level design methodologies; the detailed breakdown consists of VHDL, register-transfer-level design methodologies, advanced topics in logic synthesis, and technology mapping. This paper addresses the pedagogic considerations of teaching "Advanced Digital Design" using VHDL.

The design description of a digital system may contain a number of combinational logic blocks, flip-flops, counters, finite state machines, embedded finite state machines, and register-transfer-level function blocks such as registers, multiplexers as well as arithmetic and logic units. The VHDL description of a module can be written in dataflow, behavioral, or structural style. These module descriptions can be bundled together and randomly placed in a design description.

Based on the basic digital components, the issues of writing a VHDL description to specify a digital system are addressed in Section 2. Section 3 discusses system-level design issues. Section 4 describes laboratory and project assignments for students to practice digital design methodologies using VHDL. Section 5 presents common mistakes of and general guidelines for writing VHDL descriptions for synthesis. Finally, the results of course assessment and concluding remarks are presented in Section 6.

2. Major VHDL Lecture Topics

The "Advanced Digital Design" course begins with an introduction to VHDL, focusing on writing synthesizable VHDL descriptions[1,2]. The history and grammar of VHDL is first reviewed. The process of translating a high-level description into intermediate representation resembling an assembly program is then described. The functions of traditional software compilation including lexical scanning, parsing, and code generation are discussed[3]. To enable students to consider the implications of each VHDL statement, inference rules used by some synthesis tools to derive a hardware structure from a VHDL description are reviewed[2]. The methodologies of digital design using micro-architectural modeling, which are the main focus of this course, are addressed separately after the completion of VHDL discussion. The following is a list of major VHDL-specific topics covered in lecture classes.

2.1 Language Basics

The basics of the VHDL language including identifiers, key words, data types, data operators, attributes, generics are presented. Data objects such as signals, variables, ports, and constants as well as their roles in hardware specification are discussed in detail. The concepts of delays including inertia, transport, and delta delays are demonstrated.

Two views involved in a digital system design include the specification of input and output interface and the description of internal functions. A VHDL description generally consists of two sections: one is entity declaration and the other describes system architecture. The entity declaration identifies the name, input ports, and output ports of a digital circuit. The architecture section defines the function or structure of a digital circuit in terms of a number of processes and circuit modules. Other important VHDL topics are elaborated in subsequent subsections.

2.2 Dataflow Description

A dataflow expression consists of a number of signal or variable objects and data operators. For example, let the statement "X <= A0 + B0 + C0" be given. The expression on the right-hand side of the signal assignment operator (<=) defines a function of adding the three objects A0, B0, and C0. The resultant sum is assigned to the signal object on the left-hand side of the assignment operator. A dataflow description may be comprised of several dataflow statements. These statements may be related to one another; they may also be independent from each other.

2.3 Control Flow Specification

The algorithm of a digital design often requires conditional checking and repetition of a set of statements. In VHDL these features can be described by control flow operators such as *if-then-else*, *when-else*, *case-when*, *select-when*, *repeat*, *for-loop*, *for-generate*, etc. These VHDL control flow constructs are introduced. The implications of data and control flow on hardware implementation are also discussed.

2.4 Commonly Used Designs

As it was pointed out in Section 1, the basic components contained in a VHDL architecture section may contain design units like combinational logic functions, latches and flip-flops. Composite modules may also be included; some examples are register-transfer-level blocks such as registers, counters, arithmetic and logic units, adders, multiplexers, finite state machines, and embedded circuit modules.

The description of a combinational circuit specifies the logic function as well as the input and output ports of the circuit. The description of a latch which is a level-sensitive device often consists of a condition for value setting. A flip-flop consists of more than one stage of latches internally; a clock signal is required for defining a flip-flop. A conventional finite state machine contains only bit-wise functional specification. An embedded register-transfer-level (RTL) module differs from a conventional finite state machine such that RTL logic blocks are embedded in the control flow. Modeling considerations and inference rules for synthesis of these modules are first discussed; typical ways of describing these modules are then described. The structures and access mechanisms of memory blocks are also investigated. The styles of VHDL descriptions including dataflow, behavioral, structural, and mixed are compared as well.

2.5 VHDL Design Modules

In a VHDL description, *functions*, *processes*, *procedures*, *blocks*, *components*, and *generics* (parameterized definition) are convenient tools for defining a complex circuit module. The syntax and semantics of these infrastructures are discussed in detail.

2.6 Design Library

To support design reuse, existing VHDL descriptions and design standards can be organized using a *library*, *package*, and *configuration*. These features as well as other VHDL concepts such as *files* and *test benches* are also discussed.

3. System-Level Design Issues

Several issues of digital design at the system level are addressed in this course, including system partitioning, design space exploration, clocking schedule, input and output interface, module design using micro-architectural modeling, and design integration.

3.1 System Partitioning

A complex system can often be partitioned into several loosely coupled modules. System partitioning refers to the process of isolating tightly coupled functions as a module. The function of each module can then be designed separately. The criteria for design partitioning include functionality, connectivity, clocking, etc.[4] The main purpose of partitioning is to reduce the complexity of digital design. Each module is comprised of a

sequence of events. These events can be specified using VHDL. The configuration of the entire system can be defined through interface signals as well as the schedule of events and clocks among these modules.

It is sometimes convenient to allocate each process to a unique operating mode. Tasks dispatch can be achieved through operating-mode switching[4]. Depending on design requirements, mode setting can be done either synchronously or asynchronously.

3.2 Design Space Exploration

Exploring design alternatives is an important task for digital design. At the system level, a different partitioning constitutes a unique domain of designs. At the module level, the speed and area of a design may result from the algorithm describing the module as well as the implementation scheme of the module functions.

3.3 Module Description

A module resulted from system partitioning may be a combinational function, a set of registers, a finite state machine, a register-transfer-level function, an algorithmic function or procedure. The logic of a functional or procedural process can be synthesized using a micro-architectural modeling method[5] or as an embedded finite state machine.

3.4 Clock Generation

A synchronous sequential circuit needs clock signals to drive its operations. For example, a finite state machine, in addition to input signals, requires a clock signal to trigger state transition. Clock signal generation[6] and clocking schedule[4] are important issues to address.

3.5 Input and Output Interface

The signal of an input port to a digital system may not be properly synchronized with its internal logic. A latch or flip-flop may be needed to ensure the integrity of a module. In many cases the number of input and output ports is limited, efficient allocation of external interface resources[6] may also be a major concern. Finally, the external inputs to a digital system may be analog signals. The design of analog-to-digital and digital-to-analog converters is also an important task.

3.6 Design Integration

The task of integrating several modules into a working system is generally a bottom-up process. To support efficient design partitioning and integration, interface connections and protocols must be carefully considered. A strobe signal can be used to initiate a process being called. An acknowledgement signal may then be generated by the called module. If the frequencies of the two clock signals are different, a more sophisticated coordination protocol may be required to ensure seamless coordination.

4. Project Assignments

VHDL lectures teach students the basic knowledge and skills of digital design using VHDL. Along with lectures, students were assigned several projects to study and practice digital design methodologies. An FPGA demo board was given to each student to demonstrate digital design capabilities. There were three types of assignments: a case study, three mini assignments, and several design projects. This section presents the motivation, considerations, and evolution of developing these assignments.

4.1 Case Study

The FPGA demo board given to each student contained an FPGA chip, eight toggle switches, four push buttons, eight bar-graph Light Emitting Devices (LEDs), four seven-segment displays, and other devices[7]. To efficiently utilize the output pins of the FPGA chip, each set of four corresponding LED elements of the seven-segment displays share an output pin of the FPGA chip. Each seven-segment display, however, has its own activation pin.

A case study is an efficient way for learning a new programming language. A VHDL description which involves efficient resource utilization of the demo board was given to students as a case study to expedite the VHDL learning process. The VHDL routines described four decimal counters driven by four separate push-button switches. Each push-button switch used a different processing scheme for incrementing a four-bit counter. One directly applied the push-button signal to its counter and the second one used a flip-flop as a buffer. Each of the remaining two cases used a finite state machine to handle switch signals: in one case the counter was embedded in the finite-state-machine description and in the other case the counter was described as a separate process. The outputs of the four decimal counters were displayed on four seven-segment LEDs. These LEDs shared the same ports for data inputs; however, each had its own activation port. A scanner driven by a two-bit counter was used to control the display of the four counters. Students used an oscilloscope to study the waveforms generated by each switch for incrementing its counter. The interface protocol between each switch and its counter was investigated. The implication of vision persistence on hardware implementation of LED display was analyzed. Students studied the impacts of clock speeds on data multiplexing and module coordination. Finally, the issues related to clock signal generation were also examined. Students analyzed the VHDL description and learned VHDL-relevant design skills from the assignment.

In summary, the following intricate issues were demonstrated in the case study:
1. Essentials of a VHDL description.
2. Clock signal generation.
3. Methods of handling push-buttons.
4. Data multiplexing and the design of a hardware scanner.
5. Timing, handshaking, and coordination protocols.

In the next subsection, mini VHDL assignments are described.

4.2 Mini Assignments

Several mini VHDL assignments were defined, including a sequential to Gray code converter, a Hamming code transmitter, a Hamming code receiver, a clock generator, and a design for input capture and output display of multiple-digit decimal numbers[6]. Students practiced VHDL coding using these simple projects and in-class discussion was conducted to help students learn important design skills using VHDL. Students indicated that the mini VHDL assignments were very conducive to their learning. The requirements of these mini assignments are summarized in this subsection.

4.2.1 Hamming Code Transmitter and Receiver

A Hamming code generator and a Hamming code receiver were assigned to students for them to learn how to specify a combinational circuit using VHDL. The two VHDL programs required students to consider all the input combinations for logic completeness. Some of the lessons that students were able to derive from this simple assignment are listed below:

- The problem of simultaneously writing multiple data sources to a combinational output.
- The specification and applications of parity functions.
- The difference between logic minimization and logic partitioning.

4.2.2 Clock Signal Generator

In this assignment students practiced how to produce clock signals of different frequencies. These clock signals were derived from the same source so that some of these clock signals could be periodically synchronized. The students were requested to analyze the duty cycle of each clock signal using an oscilloscope. Also, they studied how to generate a clock signal with 50% duty cycle.

4.2.3 Micro-architectural Modeling Methodology

This assignment involved reading several binary-coded-decimal numbers, storing them into a memory, and reading memory words one by one and displaying each on a seven-segment LED. This project allowed students to practice the specification of memories as well as input capture and output display functions. They studied the methodologies of micro-architectural modeling and efficient scheduling of clock signals as well. These methodologies enable students to produce a working design for computer algorithms that can be described by a high-level procedural language.

4.3 Design Projects

In addition to the mini VHDL assignments, two or three of the projects described in this subsection were assigned to students each semester. These projects included a motion

guide[4], an alarm clock, a discrete cosine transform[5], and a sorting network[8]. In defining a project, the factors of design complexity, potential applications, and its relevancy to major design considerations were taken into account.

Multiple-stage milestones were considered for most projects. For instance, three-stage milestones were set for a clock project, including a wall clock, an alarm clock, and an alarm clock with multiple alarms and special wakeup features. Students were expected to complete at least the first-stage objective of each project. The feasibility of projects stimulated enormous interest for students to complete these assignments. The alarm clock allowed students to compare direct output of time data using 42 pins and the scheme of applying parallel-to-serial, serial-to-parallel, high-speed data transfer and buffering techniques for output management.

The discrete cosine transform provided students an opportunity to study number systems, digital organization for numeric operations, internal data conversion techniques, and various multiplication schemes. The project also provides a rich set of tradeoffs for students to explore, including the selection of data width, the positioning of binary points, quantization errors, alternative methods for input data capture and output data display.

The design of sorting networks encouraged students to study the impacts of writing efficient algorithms, dataflow scheduling for exploring design alternatives, and the methods of describing a memory block.

5. Common Mistakes and General Guidelines

In this section several commonly seen errors in writing VHDL descriptions for synthesis are described. These errors are subtle; students often had difficulty to track the root causes and correct the errors.

5.1 Combinational Logic Description

The following are three errors often seen for defining the function of a combinational function:

- Using an input port as the target variable of a statement.
- Using an output port as the source variable of a statement.
- Writing data from multiple sources to a variable simultaneously.

The Hamming code receiver described in Section 4.2 constitutes an interesting design for presenting students a scenario of experiencing with these mistakes. For single-bit error correction, a seven-bit Hamming codeword contains four data and three parity bits. Depending on a bit is in error or not, its corresponding output bit is either identical to or inversion of the input bit. Some students would unconditionally assign the output signal to be equal to the input signal. At the same time, the output is also defined to be the inversion of the input under the condition that the corresponding input error is detected. As a result, the error of assigning two different values to the output bit under that condition would be detected. Another error in the description of a Hamming code

receiver is to unconditionally feed input port to an output port and then use the output port internally for inversion.

5.2 Register Driven by Multiple Clocks

A clock signal is required to control a flip-flop or register. Routine-1 depicts a typical format for defining a flip-flop or register variable. In Routine-1, `A` is used as input and `X` is defined as a rising-edge triggered output register. Relating multiple clocks to a flip-flop or register is a common mistake. This happens if a signal or variable is defined in several processes.

The general guideline is that the definition of a clock-driven signal should be centralized; the use of the variable can however spread over several processes. A process governed by a unique clock signal should be used to define the variable. Flags can be introduced to support writing the variable from multiple data sources. A flag is included in each process writing data to the variable. Different flag values are assigned in a process to select the expected data sources. It may be essential to include a flag value which allows the target variable to retain its value in the previous cycle. As illustrated in Routine-2, the writing of a register variable may be controlled by several flag variables.

Routine-1: A clock-driven VHDL process

```
if (clk'event and clk='1') then
  ...
  X <= A;
  ...
end if;
```

Routine-2: Writing a register from multiple data sources

```
if (clk'event and clk='1') then
  ...
  if (flag0 = "01") then
    reg_var <= in_var_a;
  elsif (flag0 = "11") then
    reg_var <= in_var_b;
  elsif (flag1 = '1') then
    reg_var <= in_var_c;
  ...
  end if;
end if;
```

5.3 Asynchronous versus Synchronous Reset for A Counter

The description of a counter is generally driven by a clock signal. A clear input is often needed for a counter. The clear input can be written either inside or outside the context of the if-clause of a clock signal. Students tend to overlook the implication of these two

scenarios. A clear statement outside the context of the if-clause results in an asynchronous reset while one defined within the domain of a clock's if-clause implies a synchronous clear. The content of a counter is reset in a cycle that the asynchronous reset is asserted. A counter is not reset until the next clock cycle arrives for a synchronous reset. Many students are not aware of the one-cycle difference for the range spanned by counters generated by the two reset schemes.

5.4 Finite State Machine Design

Traditionally, a finite state machine is defined by a number of symbolic states. The next state and output functions are then defined in terms of each symbolic state and input function combination. In this case, a state identifier is defined as an enumeration data type; state assignment can then be done separately. In VHDL an identifier of integer or logic-vector type can be used as a state variable. A constant can be directly assigned to the state variable. Different values are then freely used to represent different states in a description. A state transition can be defined by assigning a new value to the state variable. A common error is to define the state transitions without using a clock's if-clause. As a result, an asynchronous sequential circuit is specified. Hazards may occur due to signal races. Also, a state variable can be shared in several blocks or sections of a VHDL description, which results in several finite state machines sharing the same state variable. Design errors could be easily introduced from inadvertent definition and use of the state variable.

5.5 Memory Design

A memory contains a number of words and each word contains a fixed number of bits. Two major concerns for defining a memory block include easy specification of memory structure and efficient methods for memory accesses. The type declarations of "word" and "memory" shown in Routine-3 are convenient facilities for a memory specification. The type declarations depicted in Routine-3 define a memory block as an array of ten-bit "words" indexed by an integer. For the purpose of synthesis, type conversion between integer and logic vector must be resolved. Many students have difficulty to write a working VHDL description for a memory block.

An efficient method of defining a memory block begins with defining a small memory primitive. For example, Routine-4 presents a VHDL description for a memory block of eight "words." The mapping between an integer index and a logic vector represented by the identifier "`adr`" is explicitly specified using a "`case`" statement in the VHDL description. The primitive description can then be identified as a VHDL component. Using the VHDL component, structural VHDL statements can be used to define a larger memory block. For example, as depicted in Routine-5, the read-access network of a 64-word memory block can be defined using nine instances of the 8-to-1 multiplexer. The read-access network of a 4096-word memory block can be defined using nine instances of the 64-to-1 multiplexing network.

Routine-3: Type declarations for memory

```
subtype word is std_logic_vector(0 to 9);
type memory is array (integer range <>) of word;
```

Routine-4: A VHDL description for read access of an 8-word memory block

```
entity mux8to1 is
  port(
        adr: in std_logic_vector(0 to 2);
        idatum: in memory (0 to 7);
        odatum: out word
      );
end mux8to1;

architecture mux8to1_arch of mux8to1 is
begin
  process(adr, idatum)
  begin
    case adr is
      when "000"  => odatum <= idatum(0);
      when "001"  => odatum <= idatum(1);
      when "010"  => odatum <= idatum(2);
      when "011"  => odatum <= idatum(3);
      when "100"  => odatum <= idatum(4);
      when "101"  => odatum <= idatum(5);
      when "110"  => odatum <= idatum(6);
      when others => odatum <= idatum(7);
    end case;
  end process;
end mux8to1_arch;
```

Routine-5: A VHDL description for read access of a 64-word memory block

```
entity mux64to1 is
  port (
         xadr: in std_logic_vector(0 to 5);
         xi: in memory (0 to 63);
         xo: out word
       );
end mux64to1;

architecture mux64to1_arch of mux64to1 is
  signal ybuf: memory (0 to 7);
begin
  mux8to1_0: mux8to1 port map (xadr(0 to 2), xi(0 to 7),   ybuf(0));
  mux8to1_1: mux8to1 port map (xadr(0 to 2), xi(8 to 15),  ybuf(1));
  mux8to1_2: mux8to1 port map (xadr(0 to 2), xi(16 to 23), ybuf(2));
```

```
  mux8to1_3: mux8to1 port map (xadr(0 to 2), xi(24 to 31), ybuf(3));
  mux8to1_4: mux8to1 port map (xadr(0 to 2), xi(32 to 39), ybuf(4));
  mux8to1_5: mux8to1 port map (xadr(0 to 2), xi(40 to 47), ybuf(5));
  mux8to1_6: mux8to1 port map (xadr(0 to 2), xi(48 to 55), ybuf(6));
  mux8to1_7: mux8to1 port map (xadr(0 to 2), xi(56 to 63), ybuf(7));

  mux8to1_a: mux8to1 port map (xadr(3 to 5), ybuf(0 to 7), xo);
end mux64to1_arch;
```

5.6 Coordination Protocols

As described in Section 3.6, a strobe and an acknowledgement pulses can be issued by a calling module and a called process, respectively, for the coordination between two modules. If the two modules share the same clock signal, proper handshaking can be easily accomplished. If the calling process needs to wait for the results generated by the called module, a counter may be embedded in the VHDL description for the purpose. If the two modules are driven by different clocks, sophisticated schemes are required to generate a pulse of appropriate width to ensure a seamless coordination. Improper coordination specification is probably the main reason for system malfunction. Extensive design experience is instrumental for gaining insights to resolve this type of problems.

6. Course Assessment and Conclusion

Numerous new materials have been developed for the "Advanced Digital Design" course over the last three years. Course organization has also been improved each time it was offered. The case study describes in Section 4.1 provides a VHDL model for students to obtain a good head-start. Other assignments provided students opportunities to practice digital design methodologies. While students worked on these projects, they often struggled in the beginning. The instructor would then provide students with clues and guidance. Students were able to learn important design skills through practicing.

To assess the effectiveness of the course, students were asked to answer the following questions.

1. The VHDL case study was useful for learning VHDL.
2. Mini assignment #1 (Hamming code transmitter and receiver) was useful for learning VHDL.
3. Mini assignment #2 (clock signal generator) was useful for learning VHDL.
4. Mini assignment #3 (input capture and display of binary-coded-decimal numbers) was useful for learning VHDL.
5. Mini-assignment in-class discussions were useful for learning VHDL.
6. Alarm clock lab assignment was useful for learning.
7. Sorting network lab assignment was useful for learning.
8. The VHDL lecture classes were useful for learning.
9. The lecture classes of using micro-architectural modeling for register-transfer-level design (data-path and controller design) were useful for learning.
10. The lecture classes on logic synthesis (Quine-McCluskey method for two-level logic minimization, multiple-level logic optimization, technology mapping) were useful for learning.

Students were asked to select the best answer from the five choices listed below for each question. The weight of each answer is shown on the right.

- Agree Strongly: 5
- Agree: 4
- Neutral/Mixed: 3
- Disagree: 2
- Disagree Strongly: 1

Table 1 presents the results of the survey conducted in the fall semester of 2006. The scores indicated that students were generally happy with the course design. This was confirmed by the fact that most students were able to complete the projects assigned to them. Question 2 had the lowest score, which showed that the combinational circuit design of Hamming code transmitter and receiver might be too easy for them. Questions 4 and 6 had the best scores, which indicated that the mini assignments of the "input and output of multiple-digit decimal numbers" and "alarm clock design" were the most useful assignments to the students.

Table 1: Statistical results of student survey

Question Index	1	2	3	4	5	6	7	8	9	10
Mean	4.13	3.88	4.5	4.75	4.25	4.88	4.13	4.38	4.25	4.13
Standard Deviation	0.35	0.64	0.53	0.46	0.46	0.35	0.35	0.52	0.46	0.64

References

1. Peter J. Ashenden, "The Designer's Guide to VHDL", Morgan Kaufmann Publishers, San Francisco, California, 2002.

2. J. Bhasker, "A VHDL Synthesis Primer," Star Galaxy Publishing, 1998.

3. A. V. Aho, R. Sethi, and J. D. Ullman, "Compilers – Principles, Techniques, and Tools," Addison-Wesley Publishing Company, 1986.

4. C. J. Tseng, "Clocking Schedule and Writing VHDL Programs for Synthesis," Proceedings of The 2004 ASEE Annual Conference & Exposition, Session 1532, Salt Lake City, Utah, June 2004.

5. C. J. Tseng and M. F. Aburdene, "Digital Signal Processing and Digital System Design Using Discrete Cosine Transform," Proceedings of The 2005 IEEE International Conference on Acoustics, Speech, and Signal Processing, Philadelphia, March 2005.

6. C. J. Tseng, "Efficient Resource Allocation for FPGA Demo Board Based Digital Laboratories," Proceedings of The 2005 ASEE Annual Conference & Exposition, Session 3532, Portland, Oregon, June 2005.

7. Xilinx, "Spartan-3 Starter Kit Board User Guide," May 2005.

8. C. J. Tseng, “Dataflow Scheduling and Exploring Digital System Design Alternatives,” Proceedings of The 2006 ASEE Annual Conference & Exposition, Session 3232, Chicago, Illinois, June 2006.