Teaching Top-down Design Using VHDL and CPLD

Morris Chang
Computer Engineering Program
Department of Computer Science
Illinois Institute of Technology, 10 west 31st street
Chicago, IL 60616
Internet: chang@charlie.iit.edu

Abstract

This paper presents a teaching experience in using VHDL and CPLD in the senior digital design course. The course focus on teaching the top-down design methodology through hands-on experiments. The industrial available tools — Maxplus2, made possible through Altera’s University Program, provide our students a smooth transition from academic concepts to industrial practice. VHDL, the industrial standard language (IEEE-1076), is used as the design entry. Thus, the students are forced to learn the practical aspect of writing a synthesizable VHDL code. The hands-on weekly projects are exercised on the integrated CPLD design tool which has VHDL compiler, logic synthesizer, functional and timing simulator, floor plan editor and programmer. With the help of programmable devices, students can bypass the waiting period for IC fabrication and obtain ASIC designs after the devices have been programmed.

The VHDL design entry in Maxplus2 is ideal for teaching top-down design methodology. Translating from a given Algorithmic State Machine (ASM) chart to a synthesizable and efficient VHDL code is presented. Exploiting the VHDL constructs to make a design reusable is demonstrated through examples. In this course, students learn how to partition a complex design into small components and focus on higher level of abstraction and hierarchy in design description which have become desirable to digital systems designers.

Introduction

In recent years, CPLDs (Complex Programmable Logic Devices) have increased dramatically in capacity and complexity. CPLDs with 100K gates are available in today’s technology. To cope with the complex design, higher level of abstraction and hierarchy in design description have become desirable to digital systems designers. Today, system-level logic design is most likely a team work, forcing modular and hierarchical design approach. Moreover, exploiting many technology (implementation) options without translation of the source design description is an increasingly important requirement. The IEEE-standard (i.e. IEEE-1076) VHDL language addresses these needs.

VHDL can be used to describe hardware from the abstract to the concrete level. Many of the EDA (Electronic Design Automation) vendors are standardizing on VHDL as input and output from their tools. These tools include simulation tools, synthesis tools, layout tools, testing tools, etc. Due to the recent advances in high-level synthesis tools, the text-based design entry has gained increasing popularity in the ASIC design. In our senior design course, we introduce VHDL as the design entry in the CPLD design environment. This course focuses on how to write VHDL code that can be processed by synthesis tools [2].

The course begins with a brief summary of the syntax of VHDL and is followed by several examples of hardware modelling. The examples include simple combinational logic, sequential models and finite state machines. Applications of VHDL to top down design methodology are presented. Specific design trade-offs in CPLD for performance and efficiency are also discussed.

We employ the industry standard language, VHDL, and a commercially available CAD tool (Maxplus2[1] from Altera) in a set of hands-on weekly laboratory experiments. These labs, in concert with a semester design project, provide our students with a smooth transition from the abstractions of academe to the realities of engineering practice.
Course Description

Following topics are addressed in this senior design course.

1. Introduction to the syntax and basic structure of VHDL—data types, operators; signal assignments; components instantiation; modeling styles: behavioral, dataflow and structural; subprograms; packaging parts and utilities.

2. The synthesizable issues under current technology—register (latches) inference; state machines.

3. The VHDL portability issues in the synthesis tools.

4. Introduction to the architecture of CPLDs.

5. Design trade-offs in CPLD.

6. Labs on: design, verification, synthesis, timing simulation and program CPLDs.

7. Lab assignments: multiplexor, adder, decoder/encoder, round-up circuit, multiplier, counter/shift registers, Finite State Machine, digital comparator, Arithmetic and Logic Unit, and a simple microprocessor.

The top-down designs that exploiting the VHDL

The VHDL design entry in Maxplus2 is ideal for teaching top-down design methodology. Instead of emphasizes the microscopic gate aspects of the digital design, we emphasize a macroscopic view of digital systems by starting from the original problem. This result in a top-down approach to design. Once the problem is analyzed and partitioned to a further design, students can focus on the design aspect and exploit the logic synthesis tool for the minimization and implementation details. In this section, we will show two type of top-down design examples that would fit well into the VHDL design.

ASM chart to VHDL

Algorithmic State Machine (ASM) chart [4] has been used to describe the finite state machine for more than two decades. The ASM chart can be used to describe either Mealy machine or Moore machine of traditional sequential circuit theory. An ASM with only unconditional outputs is equivalent the traditional Moore machine; the traditional mealy machine has conditional outputs.

Translating a given ASM chart into a synthesizable VHDL code can be done in a systematic fashion. The next figure presents a simple ASM chart. In this ASM, there are five states, namely J, K, L, M and N; three inputs: A, B, and C; three outputs: W, X, and Y.

This ASM can be implemented as following VHDL code.

```
entity asm2 is -- Specify the inputs and outputs
  port( clock, reset, a, b, c :in bit;
       x_out, y_out, w_out :out bit;
       state :out bit_vector(4 downto 0) );
end asm2;

architecture statemach of asm2 is
  type state_type is (j,k,l,m,n);
  signal mstate, next_state : state_type;
  begin
    state_register: process(clock,reset)
    begin
      if reset = '1' then
        mstate <= j;
      elsif clock'event and clock = '1' then
        mstate <= next_state;
      end if;
    end process;

    state_logic: process(mstate,a,b,c)
    begin
      if reset = '1' then
        mstate <= j;
      elsif clock'event and clock = '1' then
        mstate <= next_state;
      end if;
    end process;

  end statemach;
```


begin
  x_out <= '0';
  y_out <= '0';
  w_out <= '0';
CASE mstate IS
  WHEN j =>
    next_state<= k;
  WHEN k =>
    x_out <= '1';
    y_out <= '1';
    IF a = '1' THEN
      next_state<= l;
    ELSIF (a = '0' and b = '1') THEN
      next_state<= n;
    ELSEIF (a = '0' and b = '0') THEN
      next_state<= n;
    END IF;
  WHEN l =>
    x_out <= '1';
    IF b = '0' THEN w_out <= '1';
    ELSE w_out <= '0'; END IF;
    IF b = '1' THEN
      next_state<= m;
    ELSE
      next_state<= n;
    END IF;
  WHEN m =>
    w_out <= '1';
    IF b = '1' THEN
      next_state<= m;
    ELSIF (b = '0' and c = '1') THEN
      next_state<= k;
    ELSIF (b = '0' and c = '0') THEN
      next_state<= j;
    END IF;
  WHEN n =>
    next_state<= m;
  END CASE;
END process;
with mstate select
state <= "00001" when j,
      "00010" when k,
      "00100" when l,
      "01000" when m,
      "10000" when n;
END statemach;

This VHDL code has two processes; one for the registers and one for the combinatorial logic. The state encoding is decoded and connected to output ports for testing purpose.

System level partition

Partition a complex design into small components allows students to focus on higher level of abstraction and hierarchy in design description which have become desirable to digital systems designers. For instance, an 8-bit ripple adder can be implemented by a cascade of 8 full-adder stages. The VHDL generate statement can instantiate small components (e.g. 1-bit full adder) in an iterative fashion. In addition, using the VHDL constant declaration this 8-bit adder can be modified easily to a n-bit adder. Such approach also results in a reusable design.

However, some of the cases can be less intuitive than the adder. The digital comparator which can be found in [3] is one of such cases. The students need to know the detail algorithm that employs N 1-bit comparators in a “message passing” fashion.

One of our favorite examples is a round-up system. This system should round up a given n-bit binary data (as input) to its next higher binary data (as output) which is 2^n. Example (in decimal):

<table>
<thead>
<tr>
<th>input</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 (a special case)</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>31</td>
<td>32</td>
</tr>
<tr>
<td>58</td>
<td>64</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

It is worth noting that this system will need n+1 bits output for the input with n bits. Students are asked to come up an algorithm (a combinatorial one is preferred) and partition it into small components. Again, the design is asked to be a reusable one that can be customized easily for any length of the data. With an approach that similar to the comparator, students can quickly see the advantages of using VHDL in digital design (verse the Boolean expression.)

A simple VHDL design (not optimized one) for the 4-bit round-up system is described in the next figure. This design takes similar approach as the digital comparator described earlier. A one-bit round-up system (i.e. the roundup entity) is designed first. Then, a structural architecture, which takes advantage of VHDL construct — generate and constant, is implemented (i.e. rndup_n entity). A 4-bit design using a Boolean expression is also included for comparison. Apparently, it would be much difficult and error prone to design a 32-bit case by using the Boolean expression approach.
entity roundup is
  port( bit_in, set1_in, any1_in, leading0_in: in BIT;
       bit_out, set1_out, any1_out, leading0_out: out BIT);
end roundup;

architecture dataflow of roundup is
begin
  bit_out <= leading0_in and ((set1_in and not bit_in) or 
                          (bit_in and not any1_in));
  set1_out <= bit_in and any1_in;
  any1_out <= any1_in or bit_in;
  leading0_out <= leading0_in and not bit_in;
end dataflow;

architecture iterative of rndup_n is
component roundup port (bit_in, set1_in, any1_in, leading0_in: in bit;
                      bit_out, set1_out, any1_out, leading0_out: out bit);
end component;
for all : roundup use entity work roundup(dataflow);
constant n : integer := 4;
signal set1_im, any1_im, leading0_im: bit_vector(0 to n);
begin
  b0: roundup port map b_in(0), set1_in, any1_in, leading0_in, bit_out, set1_out, any1_out, leading0_out;
  bn_1: roundup port map b_in(n-1), set1_im(n-2), any1_im(n-2),
       leading0_in, b_out(n-1), set1_im(n-1), any1_out, leading0_out;
  b_out(n) <= set1_im(n-1);
  set1_out <= set1_im(n-1);
  b_rest: for i in 1 to n-2 generate
    rest: roundup port map b_in(i), set1_im(i-1), any1_im(i-1),
         leading0_im(i), b_out(i), set1_im(i), any1_im(i),
         leading0_im(i-1);  
    end generate;
end iterative;

The Boolean expression approach

\[
\begin{align*}
  b_{out}(0) &= b_{in}(3)b_{in}(2)b_{in}(1)b_{in}(0) \\
  b_{out}(1) &= b_{in}(3)b_{in}(2)b_{in}(1)b_{in}(0) \\
  b_{out}(2) &= b_{in}(3)b_{in}(2)b_{in}(1)b_{in}(0) \\
              &+ b_{in}(3)b_{in}(2)b_{in}(1)b_{in}(0) \\
  b_{out}(3) &= b_{in}(3)b_{in}(2)b_{in}(1)b_{in}(0) \\
              &+ b_{in}(3)b_{in}(2)b_{in}(1)b_{in}(0) \\
  b_{out}(4) &= b_{in}(3)b_{in}(2)b_{in}(1)b_{in}(0)
\end{align*}
\]

Conclusions

Using commercially available VHDL tool to design CPLD in a senior design course has following advantages for our students:

- to have a quick turn around time for their design; students can bypass the waiting period for IC fabrication and obtain ASIC designs after the devices have been programmed.
- to focus on the top-down design methodology; students can leave the minimization and detail implementations to logic synthesizer.
- to gain an edge in the job marketplace; students learn the practical aspect of writing a synthesizable VHDL code through hands-on projects.

Maxplus2 supports only a subset (the synthesizable part) of VHDL to be used as CPLD design entry. We plan to cover the non-synthesizable part of the VHDL, which can be used for modelling and simulation, through other VHDL tools.

Acknowledgments

The University Program at Altera Corporation provides us Maxplus2-VHDL and hardware at Rochester Institute of Technology and Illinois Institute of Technology. Partial support for this work was provided by the National Science Foundation’s Division of Undergraduate Education through grant DUE-9650347.

References