FPGA Design
EECE 277

More VHDL Constructs

Dr. William H. Robinson
February 11, 2005

http://eecs.vanderbilt.edu/courses/eece277/

Topics

“What's in a name? That which we call a rose by any other name would smell as sweet.”

– William Shakespeare
from Romeo and Juliet (Act II, Scene II)

• Administrative stuff
  – Homework Assignment #3 (due Monday, February 14)
  – Remember, Exam I is next Wednesday

• Very High Speed Integrated Circuit (VHSIC)
  Hardware Description Language (HDL)

• VHDL for simulation

• VHDL for synthesis

VHDL Statements

• Some VHDL constructs:
  – Signal Assignment: A <= B;
  – Comparisons
    = (equal), > (greater than), < (less than), etc.
  – Boolean operations AND, OR, NOT, XOR
  – Concurrent statements (when-else)
  – Sequential statements (CASE, IF, FOR)
    • Must be enclosed in a PROCESS block

• VHDL Reference
  – Refer to Appendix A in your textbook
  – VHDL Language Guide on class web page

VHDL Combinational Template

• Every VHDL model is composed of an entity and at least one architecture

• Entity describes the interface to the model (inputs, outputs)

• Architecture describes the behavior of the model

• Can have multiple architectures for one entity
A VHDL Template

entity model_name is
port (list of inputs and outputs);
end model_name;
architecture arch_name of model_name is
begin
concurrent statement 1;
concurrent statement 2;
... concurrent statement N;
end arch_name;

• All of the text in blue are VHDL keywords
• VHDL is NOT case sensitive
  – (ENTITY is same as entity is same as EnTiTy)

Concurrent vs. Sequential Statements

• The statements we have looked at so far are called concurrent statements
  – Each concurrent statement will synthesize to a block of logic

• Another class of VHDL statements are called sequential statements
  – Sequential statements can ONLY appear inside of a process block
  – A process block is considered to be a single concurrent statement.
  – Can have multiple process blocks in an architecture
  – Usually use process blocks to describe complex combinational or sequential logic

Example: Priority Circuit

library ieee;
use ieee.std_logic_1164.all;
entity priority is
port (y1, y2, y3, y4, y5, y6, y7 : in std_logic;
dout: out std_logic_vector(2 downto 0));
end priority;
architecture ifels of priority is
begin
-- priority circuit, Y7 highest priority input
-- Y1 is lowest priority input
process (y1, y2, y3, y4, y5, y6, y7)
begin
if (y7 = '1') then dout <= "111";
elsif (y6 = '1') then dout <= "110";
elsif (y5 = '1') then dout <= "101";
elsif (y4 = '1') then dout <= "100";
elsif (y3 = '1') then dout <= "011";
elsif (y2 = '1') then dout <= "010";
elsif (y1 = '1') then dout <= "001";
else dout <= "000";
end process;
end ifels;

• This priority circuit has 7 inputs
  – Y7 is highest priority
  – Y0 is lowest priority

• Three-bit output should indicate the highest priority input that is a ‘1’
  – i.e. if ‘Y6 =’1’, Y4 = ‘1’, then output should be “110”
  – If no input is asserted, output should be “000”

Comments on Priority Example

• This is the first example that used a bus. The DOUT signal is a 3-bit output bus.
  – std_logic_vector(2 downto 0) describes a 3-bit bus
  – std_logic_vector(0 to 2) is also a 3-bit bus, but dout(0) is least significant bit
  – std_logic_vector(0 to 2) is also a 3-bit bus, but dout(0) is MSB, dout(2) is LSB
  – ALWAYS use ‘downto’ in this class
  – What is the comparable notation on I/O pins from Quartus II?
Comments on Priority Example

- A bus assignment can be done in many ways:
  - \( \text{dout} \leq "110" \); assigns all three bits
  - \( \text{dout}(2) \leq '1' \); assigns only bit #2
  - \( \text{dout}(1 \text{ downto } 0) \leq "10" \); assigns two bits of the bus

- This architecture used the 'elsif' form of the 'if' statement
  - Note that it is 'elsif', NOT 'elseif'
  - This called an elsif chain

Priority Circuit Using \textit{if} Statements

- By reversing the order of the assignments, we can accomplish the same as the elsif priority chain

- In a process, the \textbf{LAST} assignment to the output is what counts

Priority Circuit Using \textit{when-else} Statements

- No process
  - Just one concurrent \textit{when-else} statement

A Bad Attempt at a Priority Circuit

- By reversing the order of the assignments, we can accomplish the same as the elsif priority chain

- In a process, the \textbf{LAST} assignment to the output is what counts

- No process
  - Just one concurrent \textit{when-else} statement
Comments on “bad” Priority Circuit

• There are multiple things wrong with this description

• There are multiple concurrent statements driving the DOUT signal
  – This means MULTIPLE GATE output are tied to dout signal!
  – Physically, this will create an unknown logic condition on the bus

Example: 4-to-1 Mux with 8-bit Data Paths

library ieee;
use ieee.std_logic_1164.all;
entity mux4to1_8 is
  port ( a,b,c,d : in std_logic_vector(7 downto 0);
         sel: in std_logic_vector (1 downto 0);
         dout: out std_logic_vector(7 downto 0) );
end mux4to1_8;
architecture whenelse of mux4to1_8 is
begin
  dout <= b when (sel = "01") else
c when (sel = "10") else
d when (sel = "11") else
  a; -- default
end whenelse;

Comments on “bad” Priority Circuit

• The writer seems to think that the order of the concurrent statements makes a difference
  – i.e. the last concurrent statement just assigns a '000'

• The order in which you arrange concurrent statements MAKES NO DIFFERENCE
  – The synthesized logic will be the same

• Ordering of statements only makes a difference within a process
  – This is why statements within a process are called “sequential” statements
  – The logic synthesized reflects the statement ordering (only for assignments to the same output)

Comments on Mux Example

• This is one way to write a mux, but is not the best way

• The when-else structure is actually a priority structure
  – A mux has no priority between inputs, just a simple selection
  – The synthesis tool has to work harder than necessary to understand that all possible choices for sel are specified and that no priority is necessary

• Just want a simple selection mechanism
4-to-1 Mux Using select Statement

architecture select_statement of mux4to1_8 is begin
with sel select
  dout <= b when "01",
  c when "10",
  d when "11",
  a when others;
end select_statement;

• Some synthesis tools will automatically recognize this structure as a mux
  – Will find a more efficient implementation than using a when-else or if statement structure
  – when-else and if structures define a priority structure

4-to-1 Mux Using case Statement

architecture select_statement of mux4to1_8 is begin
process (a, b, c, d, sel) begin
  case sel is
    when "01" => dout <= b;
    when "10" => dout <= c;
    when "11" => dout <= d;
    when others => dout <= a;
  end case;
end process;
end select_statement;

• There can be multiple statements for each case
• Only one statement is needed for each case in this example

Example: Logical Shift Left by 1

library ieee;
use ieee.std_logic_1164.all;
entity lshift is
  port (din : in std_logic_vector(7 downto 0);
         shift_en: in std_logic;
         dout: out std_logic_vector(7 downto 0));
end lshift;
architecture brute_force of lshift is begin
process (din, shift_en) begin
  dout <= din; -- default case
  if (shift_en = '1') then
    dout(0) <= '0'; -- shift a zero into LSB
    dout (1) <= din(0);
    dout (2) <= din(1);
    dout (3) <= din(2);
    dout (4) <= din(3);
    dout (5) <= din(4);
    dout (6) <= din(5);
    dout (7) <= din(6);
  end if;
end process;
architecture brute_force of lshift is begin

• This is one way to do it; surely there is a better way?
Example: Logical Shift Left by 1

architecture better of lshift is
begin
process (din, shift_en)
begin
    dout <= din; -- default case
    if (shift_en = '1') then
        dout(0) <= '0'; -- shift a zero into LSB
        dout (7 downto 1) <= din(6 downto 0);
    end if;
    end process;
    end better;
end lshift;

- This illustrates the assignment of a segment of one bus to another bus segment
- The bus ranges on each side of the assignment statement must be the same number of bits (each 6 bits in this case)

Example: 4-Bit Ripple-Carry Adder

• Want to write a VHDL model for a 4-bit ripple-carry adder
• Logic equation for each full adder is:
  sum <= a xor b xor ci;
  co <= (a and b) or (ci and (a or b));

Full Adder Logic Minimization

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c_i</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

(a) Truth table

\[ s = a + b + \overline{c_i} \]

(b) Karnaugh maps

Example: 4-Bit Ripple-Carry Adder

library ieee;
use ieee.std_logic_1164.all;
entity adder4bit is
port ( a, b : in std_logic_vector(3 downto 0);
      cin : in std_logic;
      cout: out std_logic;
      sum: out std_logic_vector(3 downto 0));
end adder4bit;
architecture bruteforce of adder4bit is
-- temporary signals for internal carries
signal c : std_logic_vector(4 downto 0);
begin
    process (a, b, cin, c)
    begin
        c(0) <= cin;
        -- full adder 0
        sum(0) <= a(0) xor b(0) xor c(0);
        c(1) <= (a(0) and b(0)) or (c(0) and (a(0) or b(0)));
        -- full adder 1
        sum(1) <= a(1) xor b(1) xor c(1);
        c(2) <= (a(1) and b(1)) or (c(1) and (a(1) or b(1)));
        -- full adder 2
        sum(2) <= a(2) xor b(2) xor c(2);
        c(3) <= (a(2) and b(2)) or (c(2) and (a(2) or b(2)));
        -- full adder 3
        sum(3) <= a(3) xor b(3) xor c(3);
        c(4) <= (a(3) and b(3)) or (c(3) and (a(3) or b(3)));
        cout <= c(4);
    end process;
end bruteforce;

• Straight-forward implementation
• Nothing wrong with this
• However, is there an easier way?
Example: 4-Bit Ripple-Carry Adder Using for Statement

architecture forloop of adder4bit is
    signal c : std_logic_vector(4 downto 0); -- temporary signals for internal carries.
    begin
        process (a, b, cin, c)
        begin
            c(0) <= cin;
            for i in 0 to 3 loop
                -- all four full adders
                sum(i) <= a(i) xor b(i) xor c(i);
                c(i+1) <= (a(i) and b(i)) or (c(i) and (a(i) or b(i)));
            end loop;
            cout <= c(4);
        end process;
    end forloop;

Comments on for-loop Statement

- The for-loop can be used to repeat blocks of logic
- The loop variable i is implicitly declared for this loop; does not have to be declared anywhere else
- To visualize what logic is created, 'unroll' the loop by writing down each loop iteration with loop indices replaced hard numbers

VHDL Code for Full Adder

LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY fulladd IS
    PORT ( Cin, a, b : IN STD_LOGIC ;
            s, Cout : OUT STD_LOGIC ) ;
END fulladd ;

ARCHITECTURE LogicFunc OF fulladd IS
BEGIN
    s <= a XOR b XOR Cin ;
    Cout <= (a AND b) OR (Cin AND a) OR (Cin AND b) ;
END LogicFunc ;

- VHDL can be used to describe the structure as well

4-Bit Ripple-Carry Adder

LIBRARY ieee ;
USE ieee.std_logic_1164.all ;

ENTITY adder4 IS
    PORT ( Cin : IN STD_LOGIC ;
            a3, a2, a1, a0 : IN STD_LOGIC ;
            b3, b2, b1, b0 : IN STD_LOGIC ;
            s3, s2, s1, s0 : OUT STD_LOGIC ;
            Cout : OUT STD_LOGIC ) ;
END adder4 ;

ARCHITECTURE Structure OF adder4 IS
BEGIN
    stage0: fulladd PORT MAP ( Cin, a0, b0, s0, c1 ) ;
    stage1: fulladd PORT MAP ( c1, a1, b1, s1, c2 ) ;
    stage2: fulladd PORT MAP ( c2, a2, b2, s2, c3 ) ;
    stage3: fulladd PORT MAP ( Cin => c3, Cout => Cout, a => a3, b => b3, s => s3 ) ;
END Structure ;

- Can use individual bits instead of a bus
- This version mimics the structural connections of full adders within the 4-bit adder
- Component port maps set the internal connections
Libraries and Packages

- Libraries are logical units that are mapped to physical directories
- Packages are repositories for type definitions, procedures, and functions

### Delay Models in VHDL

- **Inertial delay**
  - Default delay model
  - Suitable for modeling delays through devices such as gates

- **Transport Delay**
  - Model delays through devices with very small inertia, e.g., wires
  - All input events are propagated to output signals

- **Delta delay**
  - What about models where no propagation delays are specified?
  - Infinitesimally small delay is automatically inserted by the Simulator to preserve correct ordering of events

### Signal Assignment With Delay

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity full_adder is
    port (a, b, c_in : in std_logic;
         sum, c_out : out std_logic);
end entity full_adder;

architecture dataflow of full_adder is
    signal s1, s2, s3 : std_logic;
    constant gate_delay : time := 5 ns;
    begin
        L1: s1 <= (a xor b) after gate_delay;
        L2: s2 <= (c_in and s1) after gate_delay;
        L3: s3 <= (a and b) after gate_delay;
        L4: sum <= (s1 xor c_in) after gate_delay;
        L5: c_out <= (s2 or s3) after gate_delay;
    end architecture dataflow;
```

### Example: Inertial Delay

- For **Out 1**, pulses less than 8ns are rejected
- For **Out 2**, pulses less than 2ns are rejected

```vhdl
signal <= reject time-expression inertial value-expression after time-expression;
```
Example: Transport Delay

```vhdl
architecture transport_delay of half_adder is
  signal s1, s2 : std_logic := '0';
begin
  s1 <= (a or b) after 2 ns;
  s2 <= (a and b) after 2 ns;
  sum <= transport s1 after 4 ns;
  carry <= transport s2 after 4 ns;
end architecture transport_delay;
```

Example: Delta Delay

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
entity combinational is
  port (in1, in2 : in std_logic;
        z : out std_logic);
end entity combinational;
architecture behavior of combinational is
begin
  s1 <= not in1;
  s2 <= not in2;
  s3 <= not (s1 and in2);
  s4 <= not (s2 and in1);
  z <= not (s3 and s4);
end architecture behavior;
```

Summary

- There are many different ways to write VHDL synthesizable models for combinational logic
- There is no 'best' way to write a model
  - For now, just use the statements/style that you feel most comfortable with and can get to work (of course!)
- READ THE BOOK!!!!!!!
  - There is NO WAY that we can cover all possible examples in class. The book has many other VHDL examples.
  - I have intentionally left out MANY, MANY language details. You can get by with what I have shown you, but feel free to experiment with other language features that you see discussed in the book or elsewhere.
Summary (cont)

- **SEARCH THE INTERNET!!!!**
  - The Internet is full of VHDL examples, tutorials, etc.

- **TRY IT OUT!!!!**
  - If you have a question about a statement or example, try it out in the Altera Quartus II package and see what happens!

- **This course is about FPGA DESIGN, not VHDL**
  - VHDL is only a means for efficiently implementing your design - it is not interesting by itself.
  - You will probably learn multiple synthesis languages in your design career - it is the digital design techniques that you use that will be common to your designs, not the synthesis language.