Using VHDL to Design Digital Circuits – Part 1

- Entity and architecture
- Signal assignments
- Processes and if-then-else
- Separation principle
- Avoiding unintended storage

Jon Turner
Hardware Description Languages

- HDLs allow designers to work at a higher level of abstraction than logic gates
- As with programming languages, HDL descriptions are compiled into a lower level representation
  - low level form can be simulated for logical correctness
  - and, can be converted to a circuit specification using a library of primitive components and a set of constraints that affect circuit cost and performance
- But don’t confuse hardware design with software
  - software is translated into sequential instruction sequence
  - HDL descriptions must translate to physical circuits
  - circuits are inherently parallel with many things going on at once
Binary Coded Decimal Incrementer

```
-- Binary Coded Decimal incrementer
-- a3..a0 is 4 bit input value
-- x3..x0 is 4 bit output value
-- x3..x0 = a3..a0 + 1

entity bcdIncrement is
  port (a3,a2,a1,a0 : in std_logic;
        x3,x2,x1,x0 : out std_logic;
        carry : out std_logic);
end bcdIncrement;

architecture al of bcdIncrement is
  signal z : std_logic;
begin
  carry = a3*a0;
  z <= a1 and a0;
  x0 <= not a0;
  x1 <= a1 xor a0;
  x2 <= a2 xor z;
  x3 <= a3 or (a2 and z);
end al;
```

<table>
<thead>
<tr>
<th>decimal</th>
<th>a3..a0</th>
<th>x3..x0</th>
<th>carry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0001</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>0010</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>0011</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>0100</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>0101</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>0110</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>0111</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>1000</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>1001</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>1010</td>
<td>1</td>
</tr>
</tbody>
</table>
Comments in VHDL

- Add a comment to a line of VHDL, using two dashes; e.g. -- this is a comment
- Every source code file should start with a short comment explaining its role
  - good idea to include your name and date, as well
- Every VHDL module (entity-architecture pair) should be preceded by a comment
  - explain what the module does
  - explain the role of the input and output ports
- Comment signal, function, procedure declarations
- Comment major sections of architecture
  - help reader understand purpose of different code sections
  - don’t just re-state the obvious
Exercises

1. Consider a version of the BCD incrementer that has a carryIn signal. This circuit behaves the same way as the original when carryIn=1, but when carryIn=0, \( x_i = a_i \) for all \( i \) and the carry output is 0. Write the logic equations for this circuit.

2. Draw a block diagram of a four digit BCD increment circuit that uses the circuit from exercise 1 as a building block.

3. Draw a circuit that implements the following VHDL code fragment, assuming \( a-d \) are all single bit signals.
   
   \[
   \begin{align*}
   x & := a \text{ and } b; \\
   y & := x \text{ or } (c \text{ and } d); \\
   z(0 \text{ to } 1) & := (a \text{ and } c) \& (x \text{ or } c);
   \end{align*}
   \]

4. Write a VHDL code segment that is implemented by the circuit at right.

\[\text{[Diagram of circuit with labels A, B, C, D, X, Y]}\]
Vector Signal Assignments

- Assignments using vectors can be viewed as a set of parallel single bit assignments
  a <= "011"; is same as a(2)<='0'; a(1)<='1'; a(0)<='1';

- The bit assignments can be explicit.
  x(4 downto 1) <= y(3 downto 0) and z(2 to 5);

- Assignments that omit explicit ranges use the ranges in the signal declarations.
  architecture foo of bar is
  signal u, v: std_logic_vector(3 downto 0);
  signal w: std_logic_vector(3 to 6);
  begin
    u <= v or w;
    -- u(3 downto 0) <= v(3 downto 0) or w(3 to 6)
  end;
Conditional Signal Assignment

- The conditional signal assignment can make logic equations easier to understand (and write)

- Example:

  ```
  c <= x when a /= b else
  y when a = '1' else
  z;
  -- c <= ((a /= b) and x) or
  -- (not(a /= b) and (a='1') and y) or
  -- (not(a /= b) and not(a='1') and z);
  ```

- General form

  ```
  x <=
  v_1 when condition_1 else
  v_2 when condition_2 else
  v_3 when condition_3 else
  ... else
  v_N
  ```
Conditional Assignments with Vectors

\[ \begin{align*} 
    c(3 \text{ downto 0}) & \leq \begin{cases} 
    "0100" & \text{when } a \neq b \\
    "1011" & \text{when } a = '1' \\
    "1001" & \text{else}
    \end{cases} \\
\end{align*} \]

is equivalent to

\[ \begin{align*} 
    c(3) & \leq \begin{cases} 
    "0" & \text{when } a \neq b \\
    "1" & \text{when } a = '1' \\
    "1" & \text{else}
    \end{cases} \\
    c(2) & \leq \begin{cases} 
    "1" & \text{when } a \neq b \\
    "0" & \text{when } a = '1' \\
    "0" & \text{else}
    \end{cases} \\
    c(1) & \leq \begin{cases} 
    "0" & \text{when } a \neq b \\
    "1" & \text{when } a = '1' \\
    "0" & \text{else}
    \end{cases} \\
    c(0) & \leq \begin{cases} 
    "0" & \text{when } a \neq b \\
    "1" & \text{when } a = '1' \\
    "1" & \text{else}
    \end{cases} \\
\end{align*} \]
Selected Signal Assignment

- Selected signal assignment can be viewed as special case of conditional signal assignment
- Example:

```vhdl
with x select
    c <= a when "00",
    b when "01" | "10",
    a+b when others;
```

- can be implemented using a single multiplexor stage

- Resulting circuit is slightly more compact and faster than circuit produced by conditional signal assignment
Exercises

1. In the circuit defined by the architecture shown below, what is the value of x(2)?
   What is the value of y(5 downto 3)?
   
   ```vhdl
   architecture foo of bar is
   signal a, x: std_logic_vector(5 downto 0);
   signal b, y: std_logic_vector(7 downto 2);
   begin
       a <= "010111"; b <= "101000";
       x <= a xor b; y <= a(2 downto 1) & b(6 downto 3);
   end;
   ```

2. Draw a diagram of a circuit that implements the VHDL code segment shown below. Assume that a, b and c are all single bit signals.
   
   ```vhdl
   x <= a and b;
   y <= x xor c;
   z <= "100" when x = '1' else
         "010" when y = '0' else
         "001" when a < b else
         "111";
   ```

3. Write three ordinary signal assignments that are equivalent to the assignment to z in exercise 3.

4. Write a conditional signal assignment that is equivalent to the selected signal assignment on slide 9.
Processes and if-then-else

Example:

```vhdl
entity foo is port(
    a, b: in std_logic;
    c, d: out std_logic_vector(3 downto 0));
end foo;
architecture bar of foo is begin
    process (a, b) begin
        if a /= b then
            c <= "0010"; d <= "1100";
        elsif a = '1' then
            c <= "1101"; d <= a & b & "01";
        else
            c <= "0100"; d <= "10" & b & a;
        end if;
    end process;
    end bar;
```

- process block enables use of complex statement types
- sensitivity list for combinational circuits must include all signals whose values are used in process
- note that c, d defined under all possible input conditions - REQUIRED
Separation Principle

- A VHDL code segment defining several signals can be re-written to separate the different signals

```
x <= x"0000";
y <= x"abcd";
if a = b then
    x <= y; z <= b;
elsif a > c then
    y <= b; z <= a;
else
    z <= x + y;
end if;
```

-- code segment defining x
```
x <= x"0000";
if a = b then x <= y; end if;
```

-- code segment defining y
```
y <= x"abcd";
if a /= b and a > c then
    y <= b;
end if;
```

-- code segment defining z
```
if a = b then z <= b;
elsif a > c then z <= a;
else z <= x + y;
end if;
```

» statement order matters within each segment, but order of segments does not matter
"Conflicting" Assignments

- The code segment
  
  ```vhdl
  architecture bar of foo is begin
  a <= '1'; b <= a; a <= '0';
  end bar;
  ```

  is incorrect, because the two assignments to a conflict

- However, the following is allowed
  
  ```vhdl
  architecture bar of foo is begin
  process(a) begin
  a <= '1'; b <= a; a <= '0'; -- note: b = '0'
  end process;
  end bar;
  ```

  in such situations, the first assignment is ignored

- Within process, assignments that come later in text, logically replace earlier assignments to same signal
  
  - such replacement may be conditional on where assignments appear (e.g. in statement list of an if-then-else)
Avoiding Unintended Storage

- Within a process, if value of a signal is not specified for some input condition, it means that signal is unchanged
- Example

  ```vhdl
  process(a, b) begin
    if a = 'l' then
      x <= '0';
    elsif b = 'l' then
      x <= '1';
    end if; -- x retains its value when a=b=0
  end process;
  ```

  Storage elements are required to implement circuit with the specified behavior
  - if one accidentally omits a condition for a signal, unintended storage elements are synthesized

- Easy way to avoid unintended storage is to start process with assignment of default values to *all signals assigned a value inside the process*
Default Values

Example:

entity foo is port(
  a, b: in std_logic;
  c, d: out std_logic_vector(3 downto 0));
end foo;
architecture bar of foo is begin
  process (a, b) begin
    c <= "0100"; d <= "10" & b & a;
    if a /= b then
      c <= "0010"; d <= "1100";
    elsif a = '1' then
      c <= "1101"; d <= a & b & "01";
    end if;
  end process;
end bar;
Exercises

1. Rewrite the process below to separate the code for the different outputs.
   
   ```
   process (...) begin -- see exercise 2
   x <= "01";
   if a > b then y <= "110"; z <= "111";
   elsif c = d then x <= "11" end if;
   z <= "101";
   if a < b then y <= "001";
   elsif c > d then y <= "000"; z <= "011";
   end if;
   end process;
   ```

2. What signals should be included in the sensitivity list for exercise 1? Suppose the first signal in your list is left out. Draw a waveform diagram showing an incorrect simulation result that might be produced as a consequence of this.

3. Does the process in exercise 1 define a combinational circuit? Why or why not?

4. Draw a diagram of the circuit implemented by the process in exercise 1.