Using VHDL to Design Digital Circuits – Part 2

- constants and enumeration types
- for and case statements
- synchronization conditions
- structural VHDL

Defining Constants

- Symbolic constants can be declared within architecture
  ```vhdl
c => constant numBits: integer := 4;
end
```
- To define constants for use by multiple entities, use separate package (typically stored in a separate file)
  ```vhdl
package commonConstants is
  constant wordSize: integer := 8;
end package commonConstants;
```

Using VHDL to Design Digital Circuits – Part 2

Enumerated Types and Subtypes

- These declarations
  ```vhdl
type color is (red, green, blue, black, white);
subtype primary is color range red to blue;
signal c1, c2, c3: color;
signal p: primary;
```
- named values help clarify intention of signals with symbolic significance
- allowed values for subranges are limited; so assignments are not allowed
  ```vhdl
  p <= black; p <= c2;
```

For-loops

- Compute 2’s complement of input value
  ```vhdl
  process(x, foundOne) begin
  foundOne(0) = '0';
  for i in 0 to wordSize-1 loop
    x_neg(i) <= foundOne(i) xor x(i);
    foundOne(i+1) <= foundOne(i) or x(i);
  end loop;
  end process;
```
- allowed values for subranges are limited; so assignments are not allowed

Review Questions

1. Given the declarations
   ```vhdl
type shape is (square, rectangle, trapezoid, triangle, pentagon);
subtype quadrilateral is shape range square to trapezoid;
signal s1, s2: shape;
signal q1, q2: quadrilateral;
```
- which of the following assignments are allowed?
  ```vhdl
  s1 <= square; s2 <= q1; q2 <= s2; q1 <= s1; q1 <= pentagon;
  ```

2. Draw a circuit diagram for a circuit that implements a four-bit version of the negate module on slide 4.

Enumerated Type Example

- this declaration
  ```vhdl
type command is (clear, load, add, noop);
```
- allows us to write
  ```vhdl
  entity calc is
  Port ( clk : in  STD_LOGIC;
         cmd : in command;
         din : in  STD_LOGIC_VECTOR (15 downto 0);
         dout : out  STD_LOGIC_VECTOR (15 downto 0));
  end calc;
  ```
Enumerated Type Example (cont.)
-- this declaration
type command is (clear, load, add, noop);
-- allows us to write
architecture Behavioral of calc is
signal dreg: std_logic_vector(15 downto 0);
begin
  process (clk) begin
    if rising_edge(clk) then
      if cmd = clear then
        dreg <= (others => '0');
      elsif cmd = load then
        dreg <= din;
      elsif cmd = add then
        dreg <= dreg + din;
      end if;
      end if;
      dout <= dreg;
  end process;
end Behavioral;

Understanding VHDL
- VHDL developed for circuit modeling & simulation
  - allows specification of hardware behavior independent of implementation
  - synthesis tools developed later
- currently, two primary uses of VHDL
  - circuit specifications define circuit components and how they are connected
  - testbenches define conventional programs that are used to generate input signals for circuit simulations
- Signals are an abstraction of the wires in real circuits
  - different meaning than variables in sequential programming
  - VHDL includes variables, as well as signals
    - common case: loop indexes are variables, not signals
    - precise meaning of variables in circuit specifications is not always obvious; best to use them sparingly

Understanding VHDL
- Signal assignments define logic circuits
  - signals on left side of assignment change as signals on right side change
  - on clock edge, in case of synchronous assignments
  - not like sequential program execution
- Strong typing in VHDL
  - signal types in expressions must match exactly
    - no automatic type conversions
  - bit and integer are built-in types
  - supports user-defined types, such as std_logic
    - std_logic defines 9 values, including 0, 1 and undefined
    - additional values mostly useful for simulation

Signal assignments define logic circuits
- signals on left side of assignment change as signals on right side change
  - on clock edge, in case of synchronous assignments
  - not like sequential program execution
- Strong typing in VHDL
  - signal types in expressions must match exactly
    - no automatic type conversions
  - bit and integer are built-in types
  - supports user-defined types, such as std_logic
    - std_logic defines 9 values, including 0, 1 and undefined
    - additional values mostly useful for simulation

Case Statement
- Case statement provides convenient way to express alternatives that depend only on value of a single signal
  - architecture al of foo is
    - begin
      process(c,d,e) begin
        b <= '1'; -- provide default value for b
        case e is
          when "00"   => a <= c; b <= d;
          when "01"   => a <= d; b <= c;
          when "10"   => a <= c xor d;
          when others => a <= '0';
        end case;
      end process;
    end arch;
  - Can produce more efficient circuits than equivalent if-then-else

Circuit Implementing Case Statement
- Case can be implemented with decoder and selection logic to choose among alternatives
  - decoder with \(n\) inputs generates \(2^n\) outputs, one for each of the distinct input values
  - one output is high, the rest are low
  - the input value determines which output is high

Synchronization Conditions
- Case statement provides convenient way to express alternatives that depend only on value of a single signal
  - architecture al of foo is
  - begin
    - process(c,d,e) begin
      - b <= '1'; -- provide default value for b
      - case e is
        - when "00"   => a <= c; b <= d;
        - when "01"   => a <= d; b <= c;
        - when "10"   => a <= c xor d;
        - when others => a <= '0';
      - end case;
    - end process;
    - end arch;
  - Can produce more efficient circuits than equivalent if-then-else

Synchronization Conditions
- Case statement provides convenient way to express alternatives that depend only on value of a single signal
  - architecture al of foo is
    - begin
      - process(c,d,e) begin
        - b <= '1'; -- provide default value for b
        - case e is
          - when "00"   => a <= c; b <= d;
          - when "01"   => a <= d; b <= c;
          - when "10"   => a <= c xor d;
          - when others => a <= '0';
        - end case;
      - end process;
    - end arch;
  - Can produce more efficient circuits than equivalent if-then-else

Synchronization Conditions
- Case statement provides convenient way to express alternatives that depend only on value of a single signal
  - architecture al of foo is
    - begin
      - process(c,d,e) begin
        - b <= '1'; -- provide default value for b
        - case e is
          - when "00"   => a <= c; b <= d;
          - when "01"   => a <= d; b <= c;
          - when "10"   => a <= c xor d;
          - when others => a <= '0';
        - end case;
      - end process;
    - end arch;
  - Can produce more efficient circuits than equivalent if-then-else
Understanding Synchronization Conditions

- Signal assigned value in scope of synchronization condition is connected to flip flop output
  - so, cannot assign to same signal outside a sync condition
  - and cannot assign same signal a value in scope of sync condition involving a different clock signal
  - cannot assign same signal in different processes
  - and usually, cannot assign same signal on both rising and falling edges of same clock signal
- Changes to signals assigned within scope of sync condition are delayed until clock occurs
  - to make signal change immediately in response to another signal, place signal assignment outside sync condition
- Be careful with "mixed" processes
  - synchronous assignments require clock in sensitivity list
  - asynchronous assignments require purely synchronous process

3. Write a VHDL module that is implemented by the circuit shown below.

4. Draw a circuit that implements the following VHDL module.

5. Write a VHDL module that is implemented by the circuit shown below.

Example of Mixed Process

- changes to signals assigned within scope of sync condition are delayed until clock occurs
- to make signal change immediately in response to another signal, place signal assignment outside sync condition
- Be careful with "mixed" processes
  - synchronous assignments require clock in sensitivity list
  - asynchronous assignments require purely synchronous process
\[ a(0) \lt (0) \lt (1) \lt (2) \lt (3) \lt (4) \]

Diagram of a circuit showing the comparison of two binary numbers, \( a \) and \( b \), with each bit being checked for greater than, equal to, or less than the corresponding bit in the other number.