Generics and Loops

Announcements

- Homework #11 due today
- Homework #12 due after break – last HW
- Final Exam
- SRATE
- Reading assignment
    - Ch. 2 sect 6, Ch. 8 sect 4, Ch. 6 sect 6

Generics

- Generics allow a design entity to be described so that, for each use of the design entity, its structure and/or behavior can be altered by the choice of generic values
- Similar to constants except their default values can be overridden from the outside when the component is instantiated.

Generic Constants

- Used to develop a module/function that can take on different parameters in different applications
  - Example parameters:
    - Prop delay
    - Bit widths for address, registers, counters, etc.
  - Default value assigned to generic when module/function is developed
  - True value is assigned during component instantiation
    - This value can be different than the default value

Ex. Generic Register

```
Entity reg is
    generic (wide : integer := 2);
    port(
                        clk, reset_n : in std_logic;
                        data_in       : in std_logic_vector (wide - 1 downto 0);
                        data_out      : out std_logic_vector(wide -1 downto 0));
    end reg;
Architecture behavioral of reg is
Begin
    store: process(reset_n, clk)
      begin
            if (reset_n = ‘0’) then
                 data_out <= (others => ‘0’);
            elseif (rising_edge(clk)) then
                 data_out <= data_in;
            end if;
     end process;
End behavioural;
```

Ex. 8-bit reg using generic

```
ENTITY eight_bit_reg IS
    PORT( clk, reset_n  : IN STD_LOGIC;
            data_in       : IN STD_LOGIC_VECTOR(7 downto 0);
            data_out      : OUT STD_LOGIC_VECTOR(7 downto 0));
    END eight_bit_reg;

ARCHITECTURE structural OF eight_bit_reg IS
    COMPONENT reg is
            GENERIC (wide : integer := 2);
            PORT( clk, reset_n : IN STD_LOGIC;
                    data_in       : IN STD_LOGIC_VECTOR (width - 1 downto 0);
                    data_out      : OUT STD_LOGIC_VECTOR(width -1 downto 0));
    end COMPONENT;
    BEGIN
    REG8: reg GENERIC MAP(
                PORT MAP(
```

Multiple Generics

- A module can have more than one generic constant
- Example : a sizable n-bit register with programmable prop delay

```
Entity reg_pd is
    generic (t_pd : time := 10 ns;
                 wide : integer := 2);
    port ( clk, reset_n   :   in std_logic;
             data_in          :   in std_logic_vector(wide-1 downto 0);
             data_out        :   out std_logic_vector (wide-1 downto 0));
End entity reg_pd;
```

Example (con’t)

```
Architecture behavioral of reg_pd
    begin
          store: process(clk, reset_n)
             begin
                if (reset_n = ‘0’) then
                      data_out <= (others => ‘0’);
                elsif (rising_edge(clk)) then
                      data_out <= data_in after t_pd;
               end if;
            end process;
    end behavioral;
```

Will this compile? Why or why not?

Example Instantiation -8 bit register with prop delay

```
Entity eight_bit_reg_pd is
   port( clk, reset_n  : in std_logic;
         data_in       : in std_logic_vector (7 downto 0);
         data_out      : out std_logic_vector(7 downto 0));
   end eight_bit_reg_pd;

Architecture structural of eight_bit_reg_pd is
```

Loops

Loop Statements

- Contains a sequence of sequential statements that can be repeatedly executed, zero or more times
- Execution continues until terminated by:
  - Completion of the iteration scheme
  - Execution of an exit statement
  - Execution of a next statement that specifies a label outside of the loop

Loop Statements

- Iteration Scheme
  - Optional
  - Loop without it is infinite – cannot be synthesized
  - Two iteration schemes
    - For loops - synthesizable
    - While loops
      - Not synthesizable
      - Useful in testbenches

For loop

- Syntax of for loop

  [loop_label:] **for** identifier **in** range **loop**

  sequence_of_statements

  End loop [loop_label];

- Loops synthesize to combinatorial logic
  - If put in a synchronous process, the entire loop completes in one clock cycle
  - You cannot embed a clock in the loop (remember the only thing that can go above a clock)
  - Why can't you make a shift reg with a loop?

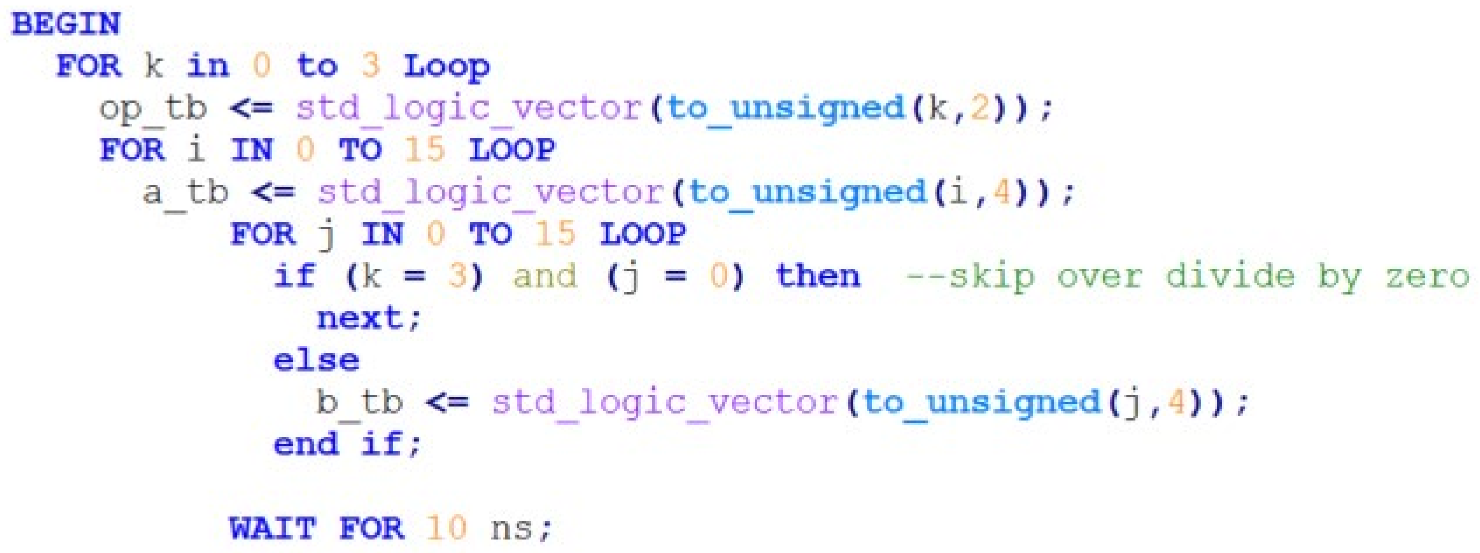
For loop

- Identifier or loop parameter follows **for**
  - Implicitly declared
  - Only exists while loop is being executed
  - Not visible outside of the loop
  - Range must be in one of the following forms:
    - *integer_expression* **to** *integer_expression*
    - *integer_expression* **downto** *integer_expression*
  - Treated as a constant in the loop
    - Can be read
    - Cannot be written
  - Loop is executed once for each value in the range

Next Statement

- Used to terminate current iteration and go to the next iteration
- *Next;*
  - Starts the next iteration of the immediately enclosing loop
- *Next loop_label;*
  - Used in nested loops to indicate for which loop to complete the iteration
- *Next when condition;*
  - Only jump to next iteration if condition is true
- *Next loop_label when condition;*
  - Combines previous 2 statements

Example of Next



```
BEGIN
  FOR k in 0 to 3 Loop
    op_tb <= std_logic_vector(to_unsigned(k,2));
    FOR i IN 0 TO 15 LOOP
      a_tb <= std_logic_vector(to_unsigned(i,4));
          FOR j IN 0 TO 15 LOOP
            if (k = 3) and (j = 0) then  --skip over divide by zero
              next;
            else
              b_tb <= std_logic_vector(to_unsigned(j,4));
            end if;

          WAIT FOR 10 ns;
```

Exit Statement

- Used to terminate the execution of an enclosing loop statement
- When executed, any remaining statements in the loop are skipped
- Control is transferred to the statement after the end loop keywords
- Can contain a condition

```
If condition then
    Exit
End if;
```

same as:

```
exit when condition
```

Example with EXIT and Generic (magnitude comparator)

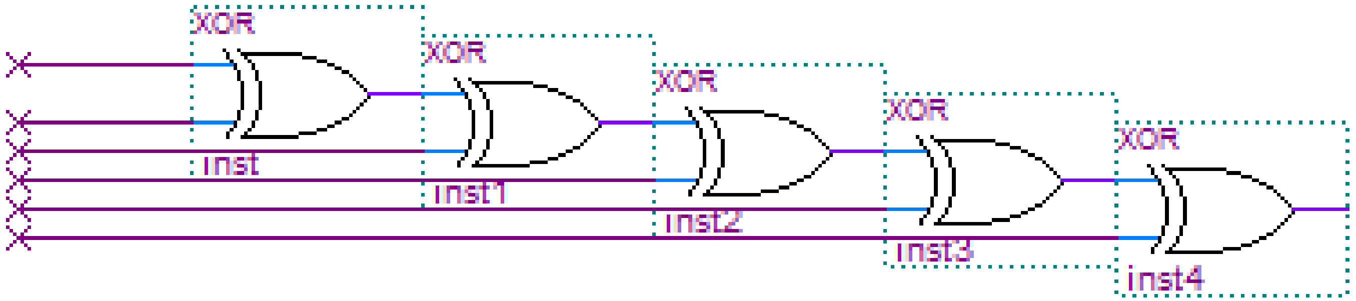
```vhdl
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY mag_comp IS
   GENERIC ( wide : integer := 4);
   PORT( p, q : IN STD_LOGIC_VECTOR(wide - 1 DOWNTO 0);
         p_gt_q, p_eq_q, p_lt_q : OUT STD_LOGIC);
END mag_comp;

ARCHITECTURE behavior OF mag_comp IS
BEGIN
   comp: PROCESS(p, q)
   BEGIN
      p_gt_q <= '0';  --default values
      p_eq_q <= '0';
      p_lt_q <= '0';
      FOR i IN wide - 1 DOWNTO 0 LOOP
         IF ((p(i) = '1') AND (q(i) = '0')) THEN
            p_gt_q <= '1';
            EXIT;
         ELSIF ((p(i) = '0') AND (q(i) = '1')) THEN
            p_lt_q <= '1';
            EXIT;
         ELSIF i = 0 THEN
            p_eq_q <= '1';
         END IF;
      END LOOP;
   END PROCESS;
END behavior;
```

Example with Loop and Generic

- Create an N-bit XOR gate. The generic constant is the width of the std_logic_vector input whose bits will all be XOR'ed together. The output is a std_logic.


XOR
inst
XOR
inst1
XOR
inst2
XOR
inst3
XOR
inst4