1. Consider the following 3-input, 2-output truth table:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>otherwise</td>
<td>0</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

where - denotes don’t care

Implement the table as a module using the following variations. Synthesize each version and submit a screen-capture of the RTL view.

(a) using behavioral code with if statements
(b) using concurrent continuous assignment statement(s)
(c) using behavioral code with case statements

2. Create a proper synthesizable module that takes a 1-bit input and outputs a version of it delayed by four clock cycles using an always block.

3. Create a proper synthesizable module that takes a 1-bit input and outputs two delayed versions of it, one version that is delayed by 2 clock cycles and another that is delayed by 5 cycles using an always block.

4. Create a synthesizable combinatorial module that accepts a 4-bit input \( x \) and generates a single-bit output \( u \) based on \( x \) and a selector. The selector should select the reduction operator to be applied to \( x \) in order to produce \( u \)

- 0: and
- 1: or
- 2: xor
- 3: nand
- 4: nor

5. Create a synthesizable sequential module that accepts two 4-bit inputs \( x \) and \( y \) and generates a 4-bit output \( q \) based on \( x \) and a selector. The selector should select the bit-size operator to be applied to \( x \) and \( y \) in order to produce \( q \). The output \( q \) should be a registered output, and so the module should accept clk signal \( clk \). There should be no unesssissyary cycle delays from input to the output.

- 0: and
- 1: or
- 2: xor
- 3: nand
- 4: nor

6. Leveraging the concatenation operator, create a module that accepts a 64-bit input \( x \), and swaps pairs of even-and odd bytes to produce a 64-bit output \( u \). The output should be registered using a clock signal \( clk \).
7. Create a test bench to demonstrate the functionality of the following code in which \( a \) and \( b \) are 1-bit inputs. The test bench should produce a table using the \$strobe task. Your calls should print time using \%0t and \$time. Provide your testbench and the output. Use the output to produce a condensed table with comments explaining the operation of the circuit.

```verilog
module DUT(y, z, a, b, clk );
input a, b, clk;
output reg [1:0] y, z;
always@(posedge clk) begin
  y=2'b00;
  if ( a == 1 ) begin
    y[1] = 1'b1;
y[0] = 1'bx;
z = 2'b01;
  end
  else if (b == 1 ) begin
    y[0] = 1'b0;
y[1] = 1'b0;
z = 2'b10;
  end
end
endmodule
```

8. Add \$display calls to the above code after every assignment statement and resimulate to show that multiple assignments are observable in simulation whereas \$strobe only prints one final result per change event. Both your \$display and \$strobe calls should print time using \%0t and \$time.

9. Repeat (8) with with blocking statements converted to non-blocking statements.

10. Now, perform a post-synthesis simulation using the module in (7). Instructions are provided in the appendix. By observing the times printed using a \$monitor call in your testbench, show that the output \( y \) and \( z \) change with some delay after the clk edge. Be sure to create a 50 MHz clock from your testbench. Provide your output and discuss what other change you see in the output as compared to what you saw in (7).

11. Implement the linear feedback shift register shown on https://en.wikipedia.org/wiki/Linear_feedback_shift_register

   ![](https://en.wikipedia.org/wiki/Linear_feedback_shift_register)

   The core 16-bit register must be implemented with a 16-bit signal. The shift should be implemented using a concatenation operator to right-shift with the left-most bit loaded from the output of the xor gates. Verify that linear-feedback shift-register produces all possible states of a 16-bit register excluding the zero state before repeating its sequence.

   To do this, use a Verilog Test Bench to save the output to a file every clock cycle for at least two full cycles of the pattern. Write C/Python/Java code to verify that the pattern repeats identically at least twice and that no number is repeated or missed during one cycle of the pattern. You should submit all code and output files.

12. Implement as a strictly combinatorial module an ALU that will add or subtract two 5-bit two-complement values \( a \) and \( b \) based on a control signal \( \text{sub} \). The outputs should be a 5-bit two's-complement result \( y \) and an over-/under-flow bit \( c \). Implement two variations of this module using
   (a) structurally using full 1-bit adders that are in turn created using Verilog primatives
   (b) using behavioural code with arithmetic operators

13. If reg [1:0] \( A \) and reg [3:0] \( B \) store the value -2, what is the result of \$display(“%d %d”,A,B)?

14. If \( H \) is an 8-bit variable what is the value stored after the assignment \( H = ’bzx1 \) executes?

15. If 4-bit variables \( A \) and \( B \) have the values -4 and -6 respectively, what is the result stored in memory for a 4-bit register \( C = B-A \) and a 4-bit register \( D = A-B \)?

16. What is the result of the following assignments to the intergers \( A \) and \( B \): \( A = -10/2 \), \( B = ’d10/2 \)?

---

1 Q12-Q15 are taken directly or modified from Chapter 4 problems of Michael D. Ciletti, Modeling, Synthesis, and Rapid Prototyping with the VERILOG (TM) HDL 1st Edition, Prentice Hall
Appendix – Post-Synthesis Simulation

- Perform post Place and Route Simulation using steps as follows
- After successfully Implement Design, explore place and route option
  - Click on Generate Post-Place & Route Simulation Model, it should now run post Place and route simulation
  - If successful you should see something similar to screen capture shown here:

![Generate Post-Place & Route Simulation Model](image1)

- Go to simulation radiobutton/tab and select Post Route simulation from the dropdown, as shown here:

![Post Route Simulation](image2)