**PART 1**

**Answer 2**

The design contains an inferred latch. On each case of sel input the design needs to remember the previous value of g0, g1, g2 and g3. Hence each of these outputs get converted to latches. The solution to this is for each value of sel in case block all the outputs should be defined. Below is an example of synthesizable circuit.

module mod1(sel, g0, g1, g2, g3, a);

input [1:0] sel;

input a;

output logic g0, g1, g2, g3;

always\_comb begin

case(sel)

2'b00: begin

g0 = a; g1 = 0; g2 = 0; g3 = 0;

end

2'b01: begin

g1 = a; g3 = 0; g2 = 0; g0 = 0;

end

2'b10: begin

g2 = a; g1 = 0; g0 = 0; g3 = 0;

end

2'b11: begin

g3 = a; g1 = 0; g2 = 0; g0 = 0;

end

endcase

end

endmodule

**Answer 3**

Variable b is driven in two different always\_comb blocks. To solve this problem b = e ^ a; should be removed or the value of e ^ a should be assigned to another output.

module mod2(a, b, c, d, e);

input a, c;

output logic b, d, e;

always\_comb begin

if (c == 1) begin

e = a;

b = c;

end

else begin

e = 0;

b = a;

end

end

always\_comb begin

d = a | c;

end

endmodule

**PART 2**

**Answer 2**

Testing approach: The corner test cases were done manually. We did exhaustive testing using automated test cases. Test cases were generated by C program which wrote randomized value of a (input data) and valid\_in (input). These values were stored in a text file which was used as an input for the testbench. The output was calculated using the C code and outputting it in a text file. The testbench also records the outputs in a text file. The two output files were compared for diff. The automated testbench was run for 100000 test cases.

**Answer 3**

**Answer 4**

a) The testbench which we designed has a few test cases which are written manually. Rest testcases are automated. The automated testcases contain random input values for input data and valid\_in. This testcases is run for 100000 values. This covers a lot of scenarios like overflows, valid\_in being 0 for multiples cycles etc.

b) In order to get the accumulator to overflow we gave largest valid input (i.e. in this case 255) for a clock cycle and therefore the result got so large that it was not possible to represent the output using a 20 bit register. In order to detect overflow we can pad an extra bit to the output bits as MSB bit so that whenever the padded bit is high it means that the output has overflown. Basically we increasing the output register size from 20 bit to 21 bit where the 20th bit indicate whether the result has overflown.

c)

d) Frequency vs Power Graph

The Frequency vs Power graph is mostly linearly increasing. This is due to the fact while static power consumption is not dependent on frequency but dynamic power (Pdynamic = ½ CV2 F) is directly proportional to frequency. Hence the Total Power consumption increases as the frequency is increased.

Frequency vs Area Graph

Frequency vs Area is also a generally linearly increasing with increase in frequency. However, there are few outliers in the graph as the synthesis tool uses different optimizations for different clock frequencies.