SECTION A

A1.

(i) State what is stored in the variables b and c after the following initial block runs:

```verilog
logic [7:0] a,b;
logic [15:0] c;
initial
begin
  a = 8'hFF;
  b = 8'hA1;
  c = 8;
  {b[5:3],c[11:7]} = a ^ {b[3:0], c[15:12]};
end
```

This question tests the knowledge of vector slices, logic literal syntax and application of exclusive or. After the initial block runs, b = 8'b1011_1001 and c = 16'b0000_0111_1000_1000.

(ii) Consider the following two SystemVerilog code fragments:

(1) ```verilog
always_ff @(posedge clk or negedge nreset)
  if (~nreset)
    q <= 1'b0;
  else
```

q <= d;

(2) always_ff @(posedge clk or nreset)
    if (~nreset)
        q <= 1'b0;
    else
        q <= d;

Explain how these two fragments differ in terms of the behaviour they describe. Be specific and illustrate your answer with sample waveforms.

The behaviour differs in that block (2) is also triggered on a rising edge of nreset. This may have an undesirable effect from the point of view of synthesis as each rising edge of nreset stores the input d in the flip-flop output q, not just clk. This difference is illustrated by the scenario shown in the waveform diagram below.

Here input d is 1 at all times and nreset changes from 1 to 0 at 12ns (this resets both q1 and q2) and then changes back to 1 at 29ns. At that point Block (2) stores the input d into the q2, i.e. between clock ticks. So Block (2) does not map to a standard flip-flop while Block (1) describes standard D-type flip-flop behaviour with async active-low reset.

[15 marks]

A2.
(i) Explain the difference between blocking and non-blocking assignments in SystemVerilog. Give an example of a simple code fragment with statements using blocking assignments such that a different behaviour would result if the blocking assignments were replaced with non-blocking ones.
A synthesisable combinational logic block is required with the output logic [4:0] y and the input logic [7:0] x. The value of y must be equal to the number of bits in x which are equal to 1. Thus, if \( x = 8'b1010_1000' \), \( y = 3 \). Write an `always_comb` block to implement the above specification. Hint: use a `for` loop and blocking assignments.

```verilog
always_comb
begin
    y = 5'b00000; // initial value
    // immediate target updates are needed hence = is essential
    for (int i = 0; i<=7; i++)
        if (x[i])
            y = y+1;
end
```

Note: to synthesise correctly the above code in a module where y is an output port, a modification would be required, as output
Signals cannot be appear in assignments on the right hand side. E.g.

```systemverilog
module q2ii (input logic [7:0] x, output logic [4:0] y);

logic [4:0] tmp; // temp variable used in always_comb specification
always_comb
begin
    tmp = 5'd0;
    for (int i = 0; i <= 7; i++)
        if (x[i])
            tmp++;
end

assign y = tmp; // concurrent block assigns tmp to y
endmodule
```

A3. A synthesisable SystemVerilog module is required for a synchronous 12-bit register with one 12-bit logic output port Y, four single-bit logic input ports E, R, L and CLK and one 12-bit logic input port X. The module has the following specification. If E=1'b1, the output Y increments on the next rising clock edge: Y <= Y + 1'b1; Otherwise, if the input L=1'b1, the machine loads the input X into Y: Y <= X. The signal R is an active-high synchronous reset and CLK represents the clock.

(i) Provide a SystemVerilog description of the module.

```systemverilog
module pc (input logic clk, R, E, L, input logic [11:0] X, output logic [11:0] Y);

always_ff @ (posedge clk)
if (R) // sync reset must appear first in the if conditions
    Y <= 12'b0;
else if (E) // increment must appear 2nd according to spec above
    Y <= Y + 1'b1;
else if (L) // parallel load
    Y <= X;
endmodule // module pc
```
(ii) Provide a testbench which will test the following scenario. On the first edge of the clock the register is reset and the reset signal remains asserted for the next 3 clock periods. For the next 5 clock periods the register \( Y \) increments. Then the value of 3 is written into the register, followed by an increment.

```verilog
module pctest;
logic clk, R, E, L;
logic [11:0] X, Y;

pc pc1 (.*) // create an instance of the tested module

initial // clk period 10ns
begin
    clk = 0;
    forever #5 clk = ~clk;
end

initial
begin
    R = 1; // this will reset pc on 1st clock tick at 5ns
    X = 0, E=0; L = 0;
    // hold R=1 for 3 clock ticks, i.e. 15, 25, 35 ns.
    #40 E= 1; R = 0; // increments will follow
    // wait next 5 ticks, (45, 55, 65, 75, 85) i.e 50ns
    #50 E=0; L=1; X =3; // write 3 into pc
    #50 E=1, L=0; // increment
end
```

[15 marks]
SECTION B

B1. (i) Explain, with examples, the difference between a defect and a fault? Why do we look for faults, and not defects, in digital circuits? [5 marks]

A defect is a physical error in a manufactured circuit. A fault is the electrical manifestation of that defect. We look for faults because they can be enumerated – there is an infinite number of possible defects.

(ii) Write down the fault list for the circuit shown in Figure B1.

![Figure B1](image)

Derive a test for B/0 for the circuit shown in Figure B1. What other faults are covered by this test? Hence, derive a test for B/1. What other faults are covered by this test? You must show all your working to obtain full marks for this question. [10 marks]

Test for B/0 => B=1
To transmit to F, C=1, D=1
To transmit to Y => E=0, G=0
E=0 => A=0 or C=1
G=0 => A=0 or C=0 or D=1
So a test is 0111/1

Also covers Y/0, F/0, C/0, D/0

Test for B/1 => B=0
Hence, a test is 0011/0, by analogy

Also covers Y/1, E/1, F/1, G/1

(iii) Show that the circuit of Figure B1 contains a static hazard. Show that including redundant logic removes that static hazard. Show that a test for one stuck fault in this redundant logic cannot be found. What are the implications for digital design of your answer? [10 marks]

Logic function is \( Y = \overline{A} \cdot \overline{C} + B \cdot C \cdot D + A \cdot C \cdot \overline{D} \)

If \( A=B=D=1 \) and \( C \) changes from 1 to 0, will get a hazard. Can remove this by adding a redundant term: \( H = A \cdot B \cdot (Y=E+F+G+H) \). Can't now test for \( H/0 \), because we have to set \( H=1 \) and \( E=F=G=0 \), which is inconsistent.

Implication is that we can't create testable redundant logic. So it might as well not be there. The alternative is to use synchronous sequential logic, so hazards don't matter.
Figure B2 shows a circuit configured to perform a built-in self-test. The LFSR and the MISR both have 3 bits. For both registers, the feedback takes the exclusive OR of the two lowest bits to create the next value of the highest bit.

(i) Assuming the LFSR starts in state 111, derive the counting sequence for the LFSR. Which state is not part of the sequence? [5 marks]

(ii) Draw the circuit diagram of the MISR. [5 marks]

(iii) Assuming that both the LFSR and MISR start in state 111, that the circuit under test (CUT) implements the function \( Q = A \cdot (B + \overline{C}) \), that \( Q \) is connected to the lowest bit input of the MISR, with the other two inputs held at 0, and that the LFSR and MISR have the same clock, tabulate the values of the MISR as the LFSR counts. What is the signature in the MISR when the LFSR returns to state 111? [8 marks]

(iv) What is the signature in the MISR, if \( A \) in the CUT is stuck at 1? Assume the fault is confined to the CUT and that the behaviour of the LFSR is not affected. Comment on your answer. [7 marks]
i) 

\[
\begin{array}{ccc}
  a & b & c \\
  1 & 1 & 1 \\
  0 & 1 & 1 \\
  0 & 0 & 1 \\
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  1 & 0 & 1 \\
  1 & 1 & 0 \\
  1 & 1 & 1 \\
\end{array}
\]

No state 000

ii) 

Outputs from Circuit Under Test

![Circuit Diagram]

iii) 

\[
\begin{array}{cccccccc}
  a & b & c & q & x & y & z \\
  1 & 1 & 1 & 1 & 1 & 1 & 1 \\
  0 & 1 & 1 & 0 & 0 & 1 & 1 \\
  0 & 0 & 1 & 0 & 0 & 0 & 1 \\
  1 & 0 & 0 & 1 & 1 & 0 & 1 \\
  0 & 1 & 0 & 0 & 1 & 1 & 0 \\
  1 & 0 & 1 & 0 & 1 & 1 & 1 \\
  1 & 1 & 0 & 1 & 0 & 1 & 0 \\
  1 & 1 & 1 & 1 & 1 & 0 & 0 \\
\end{array}
\]

So signature is 100

iv) 

\[
\begin{array}{cccccccc}
  a & b & c & q & x & y & z \\
  1 & 1 & 1 & 1 & 1 & 1 & 1 \\
  0 & 1 & 1 & 1 & 0 & 1 & 0 \\
  0 & 0 & 1 & 0 & 1 & 0 & 1 \\
  1 & 0 & 0 & 1 & 1 & 1 & 1 \\
  0 & 1 & 0 & 1 & 1 & 0 & 1 \\
  1 & 0 & 1 & 0 & 1 & 0 & 1 \\
  1 & 1 & 0 & 1 & 1 & 1 & 1 \\
  1 & 1 & 1 & 1 & 0 & 1 & 0 \\
\end{array}
\]

Probability of aliasing is 0.125

END OF PAPER