**Md. Maniruzzaman**

**ID: 155009**

**IV. Exercise**

***A\_01: Answer***

**Problem was:**

* Incomplete sensitivity list

**Correct Code:**

module q1(

  input  wire A, B, D, E,

  output reg  C, F

);

  always @\* begin

    if (A)

      C = B ^ A;

    else

      C = D & E;

    F = C | A;

  end

endmodule

***A\_02: Answer***

**Problem was:**

* Multiple procedural assignments (race condition) to the same variable C.

**Correct Code:**

module q2(

  input  wire [7:0] B, E,

  output reg        C

);

  always @\* begin

    C = ^E;   // Reduction XOR of E

  end

endmodule

***A\_03: Answer***

**Problem was:**

* Sequential and combinational logic separation
* No initialization of Q
* Simulation-specific issues
* Clock example

**Correct code:**

`timescale 1ns/1ps

`default\_nettype none

module q3(

input wire clock,

input wire A,

input wire D,

input wire E,

output reg Q,

output reg F

);

// Sequential block (use non-blocking)

always @(posedge clock) begin

if (A) Q <= D;

end

// Combinational block (proper begin/end and sized literal)

always @\* begin

case (Q)

1'b0: F = E;

default: F = 1'b1;

endcase

end

endmodule

***A\_04: Answer***

**Problem was:**

* Multiple drivers on one net
* Incorrect port declaration syntax
* Undeclared inputs in top

**Correct code:**

`timescale 1ns/1ps

`default\_nettype none

module top(

input wire A,

input wire C,

output wire B

);

wire B1, B2;

bar u1 (.D(A), .E(B1));

bar u2 (.D(C), .E(B2));

assign B = B1 | B2; // single resolved driver

endmodule

module bar(

input wire D,

output wire E

);

assign E = ~D;

endmodule

***A\_05: Answer***

**Problem was:**

* Incorrect port declaration syntax
* Undeclared signal
* Combinational loop
* Sequential assignment style

**Correct code:**

`timescale 1ns/1ps

`default\_nettype none

module foo(

input wire clock,

input wire A,

input wire B,

output reg E

);

// Break the C–D combinational loop and fix ports/clock/non-blocking style

wire C, D;

assign C = A ^ B; // from inputs only

assign D = A | B; // (A ^ B) | B simplifies to A | B

always @(posedge clock) begin

E <= B & D; // sequential: use non-blocking

end

endmodule

**Question-B:** *According to the modules IfMux8 and CaseMux8, draw the corresponding circuits and compare them.*

**Answer:**

Circuit Schematic

A screenshot of a computer

AI-generated content may be incorrect.

**Comparison of IfMux8 and CaseMux8**

* **Functional equivalence**:

Both IfMux8 and CaseMux8 describe the same 8-to-1 multiplexer. When the conditions are equality checks on the select input, the synthesis result is identical.

* **Priority vs. parallel intent**:

The if/else if style normally implies priority. But, if the conditions are disjoint (sel == 3’dk), the tool reduces it to a standard multiplexer. The case(sel) form directly shows a multi-way choice, which makes the designer’s intention clearer.

* **Readability and style**:

Case statements are shorter, easier to read, and avoid accidental priority. They also match the multiplexer schematic more clearly, which is preferred in grading and reviews.

* **Synthesis result**:

Both forms synthesize to nearly the same structure: a decoder with AND gates feeding into an OR tree, or a multiplexer (mux) tree. The timing and area are practically identical in clean cases.

* **Corner cases**:

An incomplete if/else chain may cause unintended priority. A case statement without a default may infer a latch. To avoid problems, designers often use a unique case or a priority case in SystemVerilog or ensure that all possibilities are covered with a default.