2) Incorporate Build-in-Self-Test (BIST) modules for memory blocks, enabling the identification and containment of errors.

3) Develop remediation circuits to rectify identified errors.

5. **Consideration of Timing:**

   1) When developing RTL code, it's crucial to take timing considerations into account. Practical outcomes might be different from RTL simulation results due to timing issues.

   2) During design specification, identify the anticipated operational clock frequency and the longest critical path.

6. **Memory:** Avoid reading memory before writing to it. Some FPGAs may reset all memory blocks to zeros when powered up.

Exercises

Problem 4.1. What are the differences between asynchronous and synchronous reset?

Problem 4.2. What are the differences between non-blocking and blocking assignments in Verilog HDL designs?

Problem 4.3. Identify the ten bugs present in the Verilog code below, which include seven different types of design simulation warnings/errors: 1) signal declaration errors, 2) bit mismatch warnings, 3) incomplete trigger lists, 4) incomplete *case* – *endcase* statements, 5) errors in non-blocking and blocking assignments, 6) incomplete *begin* – *end* statements, and 7) incomplete *module* – *endmodule* blocks.

```
1  module design_bugs (input       [7:0] a, b, c,
2                      input       [1:0] sel,
3                      input             rst, clk,
4                      output reg        d      );
5  assign d = a & b | c;
6
7  wire [7:0] e;
8  always @ (a, b) begin
9    case (sel)
```

```
10      2'h0 : e <= a;
11      2'h1 : e <= b;
12    endcase
13  end
14
15  wire [7:0] f;
16  always @ (posedge clk, negedge rst) begin
17    if(~rst) begin
18      f = 8'h0;
19    end else begin
20      f = e   ;
21    end
22
23  wire [7:0] g;
24  initial begin
25    g = 4'h0;
26    #10; g = f;
27  end
```

Problem 4.4. Consider the following two designs, labeled as Design #1 and Design #2, both using *always* blocks.
1) Which one of the designs is level sensitive, and which one is edge sensitive?
2) Do they use non-blocking or blocking assignments?
3) In Design #1, is the data type of "c" a *reg* or *wire*? In Design #2, is the data type of "c" a *reg* or *wire*?

```
1  // Design #1
2  always (a, b) begin
3      c <= a & b;
4  end
5
6  // Design #2
7  always (posedge clk) begin
8      c <= a & b;
9  end
```

Problem 4.5. In designs #1 and #2 below, which design uses asynchronous reset, and which design uses synchronous reset?

```
1  // Design #1
```

```
always (posedge clk, negedge rst) begin
  if(~rst) begin
    c <= 1'b0 ;
  end else begin
    c <= a & b;
  end
end

// Design #2
always (posedge clk) begin
  if(~rst) begin
    c <= 1'b0 ;
  end else begin
    c <= a & b;
  end
end
```

Problem 4.6. Figure 4.7 illustrates a Verilog design representing a 4-bit 2-to-1 multiplexer. The design comprises two 4-bit inputs labeled as "bus_a[3:0]" and "bus_b[3:0]", along with a 4-bit output labeled as "bus_c[3:0]". A single-bit signal denoted as "sel" serves as the selection input. When "sel" is set to 0, the output "bus_c[3:0]" is driven by the input "bus_a[3:0]". Conversely, when "sel" is set to 1, the output "bus_c[3:0]" is determined by the input "bus_b[3:0]".

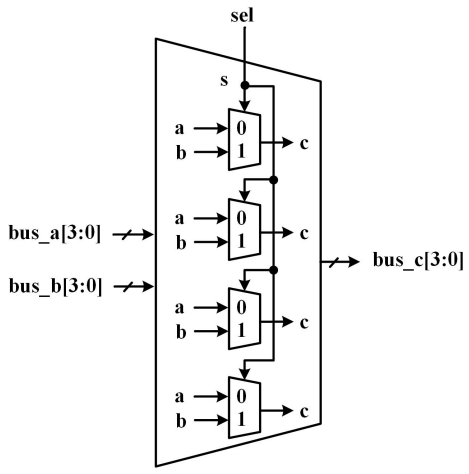

FIGURE 4.7
Design Instantiation (Four Single-Bit Mux into a 4-Bit Mux)

The design utilizes four individual single-bit 2-to-1 multiplexers. These

submodules' IOs connections are established with the top module's IOs, progressing from MSB to LSB. Specifically, the IOs of the first submodule, designated as "a-c", are interconnected with the MSB signals of "bus_a[3]", "bus_b[3]", and "bus_c[3]" within the top module. The second submodule's IOs are correspondingly linked to the intermediate bits "bus_a[2]", "bus_b[2]", and "bus_c[2]" of the main module, and this pattern continues until the LSB is reached. Furthermore, the selection signals labeled as "s" in the submodules are integrated together with the top module's selection signal "sel".

The Verilog design of the submodule is presented in lines 1-5, followed by the declaration of the top module name and IOs in lines 7-16. Write the instantiation code below the comments (lines 12-14).

```
1  // Submodule Design: Single-Bit 2-to-1 Mux
2  module mux2to1_1bit (input  a, b, s,
3                       output c      );
4  assign c = ~s ? a : b;
5  endmodule
6
7  // Top Module Instantiation: 4-bit 2-to-1 Mux
8  module mux2to1_4bit (input  [3:0] bus_a, bus_b;
9                       input         sel         ,
10                      output [3:0] bus_c       );
11 //--------- Instantiation --------------------
12
13 endmodule
```

PBL 4: AND-OR-Latch Sequential Circuit

1) Design a sequential circuit according to the illustration in Figure 4.8. The combinational section, housing two AND gates and one OR gate, can be succinctly described using a concurrent *assign* block. For the sequential aspect, incorporating a latch, can be succinctly described employing an conditional *assign* block or an level-triggered *always* block.

The circuit has six single-bit IO ports: four inputs labeled "a-d", one input "en" signal (for latch enable), and one output "f". An internal signal "e" is used to connect the combinational and sequential parts.

2) Create a testbench featuring the following test cases and verify the design functionality.

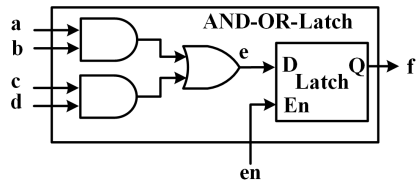

FIGURE 4.8
Design Structure of AND-OR-Latch Circuit

```
initial begin
  en = 1'b0; a = 1'b0; b = 1'b0; c = 1'b0; d = 1'b0;
  #10; en = 1'b1; a = 1'b1; b = 1'b1; c = 1'b1; d = 1'b0;
  #10; en = 1'b0; a = 1'b1; b = 1'b0; c = 1'b0; d = 1'b1;
  #10; en = 1'b1; a = 1'b0; b = 1'b1; c = 1'b1; d = 1'b1;
  #10; en = 1'b0; a = 1'b1; b = 1'b1; c = 1'b0; d = 1'b1;
end
```

PBL 5: AND-OR-Reg Sequential Circuit

1) Design a sequential circuit according to the illustration in Figure 4.9. The combinational section, housing two AND gates and one OR gate, can be succinctly described using a concurrent *assign* block. For the sequential aspect, incorporating a register, can be succinctly described employing an edge-triggered *always* block. It's important to note that the register responds to both a falling-edge reset (indicated as "negedge rst") and a rising-edge clock (indicated as"posedge clk").

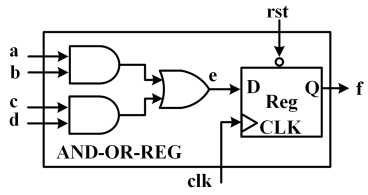

FIGURE 4.9
Design Structure of AND-OR-REG Circuit

The IO ports are all single-bit signals, including four inputs "a-d", one

input “clk”, one input “rst” signal, and one output “f”. The internal signal “e” is used for connecting the combinational and sequential parts.
2) Create a testbench featuring the following test cases and verify the design functionality.

```
initial begin
  rst=1'b0; clk=1'b0;
  a = 1'b0; b = 1'b0; c = 1'b0; d = 1'b0;
  #10 rst=1'b1;
  #10; a = 1'b1; b = 1'b1; c = 1'b0; d = 1'b0;
  #10; a = 1'b0; b = 1'b1; c = 1'b0; d = 1'b1;
  #10; a = 1'b0; b = 1'b1; c = 1'b1; d = 1'b1;
  #10; a = 1'b0; b = 1'b1; c = 1'b0; d = 1'b1;
end

always #5 clk=~clk;
```