## Exercises

**Problem 3.1.** What are the differences between level-sensitive and edge-sensitive *always* blocks?

**Problem 3.2.** What are the differences between *reg* and *wire* data types?

**Problem 3.3.** The following examples demonstrate the use of signal assignment with mismatched bit widths. Assuming an 8-bit signal "wire [7:0] a = 8'h89;", what values will the signals "b" and "c" have in each example?

```
wire [7 :0] a = 8'h89;
wire [15:0] b = a    ;
wire [3 :0] c = a    ;
```

**Problem 3.4.** Assuming we have two 4-bit signals "a" and "b" with the values of binary 4'b0101 and 4'b1100, respectively, what are the resulting values of signals "c", "d", "e", "f", and "g" when performing bitwise operations on them?

```
wire [3:0] a = 4'b0101 ;
wire [3:0] b = 4'b1100 ;

wire [3:0] c = a & b   ; // AND
wire [3:0] d = a | b   ; // OR
wire [3:0] e = a ^ b   ; // XOR
wire [3:0] f = ~(a & b); // NAND
wire [3:0] g = ~(a | b); // NOR
```

**Problem 3.5.** Assuming we have a 4-bit signal "a" with the value of binary 4'b0011, what are the resulting values of signals "b", "c", "d", "e", and "f" when using reduction operations?

```
wire [3:0] a = 4'b0011;

wire b = &a ;    // Reduction AND
wire c = |a ;    // Reduction OR
```

```
5  wire d = ^a ;     // Reduction XOR
6  wire e = ~&a;     // NOT Reduction AND
7  wire f = ~|a;     // NOT Reduction OR
```

---

**Problem 3.6.** Assuming we have a 128-bit signal "a", how can we use a reduction operation in Verilog to determine the statement "if(a==128'hffff_ffff_ffff_ffff_ffff_ffff_ffff_ffff)", which checks if all the 128 bits of the signal "a" are all binary ones. Similarly, express the statement "if(a==128'h0)" with a reduction operation, which checks if all the 128 bits are all binary zeros.

Additionally, use reduction operations to express the following two statements: "if(a!=128'h0)" and "if(a!=128'hffff_ffff_ffff_ffff_ffff_ffff_ffff_ffff)", which check if not all the 128 bits are binary zeros and ones, respectively.

**Problem 3.7.** Assume that we have two 8-bit signals "a" and "b" with the hexadecimal values 8'h12 and 8'h34, respectively. What are the resulting values of signals "vec1", "vec2", "vec3", "vec4", and "vec5" when using concatenation operations below?

---

```
1  wire [7:0] a = 8'h12;
2  wire [7:0] b = 8'h34;
3
4  wire [15:0] vec1 = {a, b}            ;
5  wire [7:0]  vec2 = {a[3:0], b[7:4]} ;
6  wire [15:0] vec3 = {4{a[3:0]}}       ;
7  wire [15:0] vec4 = {8'h0, a}         ;
8  wire [15:0] vec5 = {b, 8'h0}         ;
```

---

**Problem 3.8.** Assume that we have an 8-bit signal "a" with the hexadecimal values 8'h17. What are the values of 10-bit signals "lls2" and "lrs2" when performing logical left and right shift operations below?

---

```
1  wire [7:0] a = 8'h17  ;
2  wire [9:0] lls2 = a<<2;
3  wire [9:0] lrs2 = a>>2;
```

---

## Training: Vim, Simulation Env, Siemens ModelSim

Prior to commencing the projects, it is advisable to peruse **Appendix A** and **Appendix B**. These sections provide essential guidance on tasks such as editing and inserting Verilog code using the Vim editor, simulating Verilog designs using Siemens Modelsim, and setting up a fundamental simulation environment through the creation of a TCL script, a filelist, and a testbench. Following this preparation, you can proceed to gain hands-on experience by working on the projects outlined below.

## PBL 1: AND-OR Combinational Circuit

1) Design a combinational circuit using a concurrent *assign* block. Refer to Figure 3.5 for the design structure, which includes two AND gates and one OR gate. The circuit should have single-bit inputs "a", "b", "c", and "d", and a single-bit output "e".

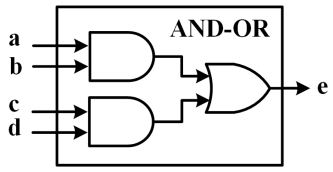


**FIGURE 3.5**
Design Structure of AND-OR Circuit

2) Create a basic testbench with the following test cases.

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

## PBL 2: AND-MUX Combinational Circuit

1) Design a combinational circuit using a conditional *assign* block. Refer to Figure 3.6 for the design structure, which includes two AND gates and one multiplexer. The circuit should have single-bit inputs "a", "b", "c", and "d", a single-bit select input "sel", and a single-bit output "e".

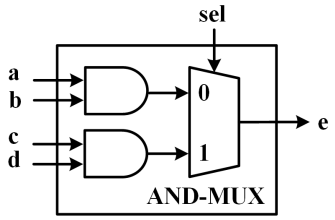


**FIGURE 3.6**
Design Structure of AND-MUX Circuit

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

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

## PBL 3: AND-OR-Tri Combinational Circuit

1) Design a combinational circuit using a conditional *assign* block. Refer to Figure 3.7 for the design structure, which includes two AND gates, one OR gate, and one tri-state buffer. The circuit should have single-bit inputs "a", "b", "c", and "d", a single-bit enable input "en", and a single-bit output "f".
2) Create a testbench featuring the following test cases and verify the design functionality.

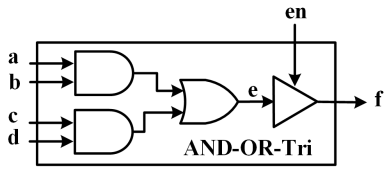


**FIGURE 3.7**
Design Structure of AND-OR-Tri Circuit

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