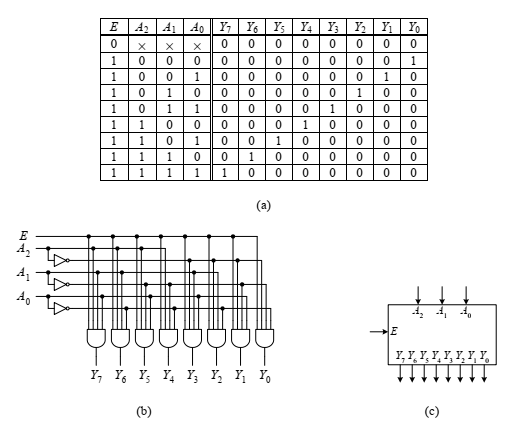
**Assignment 3**

**Note: All Verilog questions is developed basing on Structural modelling**

Phạm Quang Vinh - ITITIU21347

1. Write the Verilog module to implement 3-to-8 decoder, then develop the testbench to simulate and test this module. Show your simulation results



A 3-to-8 decoder: (a) truth table; (b) circuit; (c) logic symbol.

**Verilog Module for 3-to-8 Decoder:**

module decoder\_3to8 (

input wire [2:0] in, // 3-bit input

output reg [7:0] out // 8-bit output

);

always @ (in) begin

case (in)

3'b000: out = 8'b00000001;

3'b001: out = 8'b00000010;

3'b010: out = 8'b00000100;

3'b011: out = 8'b00001000;

3'b100: out = 8'b00010000;

3'b101: out = 8'b00100000;

3'b110: out = 8'b01000000;

3'b111: out = 8'b10000000;

default: out = 8'b00000000;

endcase

end

endmodule

**Testbench for the 3-to-8 Decoder :**

module tb\_decoder\_3to8;

// Declare inputs as regs and outputs as wires

reg [2:0] in;

wire [7:0] out;

// Instantiate the Unit Under Test (UUT)

decoder\_3to8 uut (

.in(in),

.out(out)

);

initial begin

// Apply test vectors

$display("Input | Output");

in = 3'b000; #10; $display(" %b | %b", in, out);

in = 3'b001; #10; $display(" %b | %b", in, out);

in = 3'b010; #10; $display(" %b | %b", in, out);

in = 3'b011; #10; $display(" %b | %b", in, out);

in = 3'b100; #10; $display(" %b | %b", in, out);

in = 3'b101; #10; $display(" %b | %b", in, out);

in = 3'b110; #10; $display(" %b | %b", in, out);

in = 3'b111; #10; $display(" %b | %b", in, out);

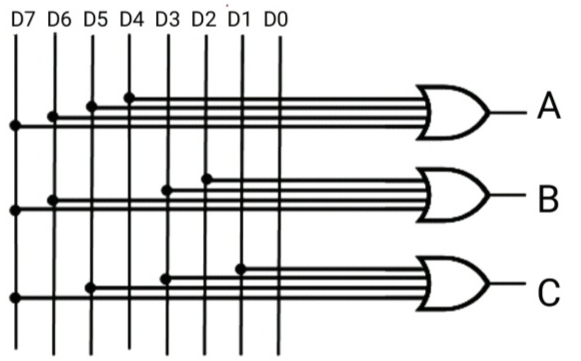
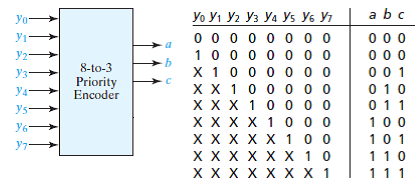
// Finish simulation

$finish;

end

endmodule

1. Write the Verilog module to implement the 8-to-3 Priority encoder, then develop the testbench to simulate and test this module.



**Verilog Module for 8-to-3 Priority Encoder**

module priority\_encoder\_8to3 (

input wire [7:0] in, // 8-bit input

output reg [2:0] out, // 3-bit output

output reg valid // Output valid flag

);

always @ (in) begin

valid = 1;

casez (in)

8'b10000000: out = 3'b111;

8'b?1000000: out = 3'b110;

8'b??100000: out = 3'b101;

8'b???10000: out = 3'b100;

8'b????1000: out = 3'b011;

8'b?????100: out = 3'b010;

8'b??????10: out = 3'b001;

8'b???????1: out = 3'b000;

default: begin

out = 3'bxxx; // Invalid output

valid = 0;

end

endcase

end

endmodule

**Testbench for the 8-to-3 Priority Encoder**

module tb\_priority\_encoder\_8to3;

// Declare inputs as regs and outputs as wires

reg [7:0] in;

wire [2:0] out;

wire valid;

// Instantiate the Unit Under Test (UUT)

priority\_encoder\_8to3 uut (

.in(in),

.out(out),

.valid(valid)

);

initial begin

// Apply test vectors

$display("Input | Output | Valid");

in = 8'b00000001; #10; $display(" %b | %b | %b", in, out, valid);

in = 8'b00000010; #10; $display(" %b | %b | %b", in, out, valid);

in = 8'b00000100; #10; $display(" %b | %b | %b", in, out, valid);

in = 8'b00001000; #10; $display(" %b | %b | %b", in, out, valid);

in = 8'b00010000; #10; $display(" %b | %b | %b", in, out, valid);

in = 8'b00100000; #10; $display(" %b | %b | %b", in, out, valid);

in = 8'b01000000; #10; $display(" %b | %b | %b", in, out, valid);

in = 8'b10000000; #10; $display(" %b | %b | %b", in, out, valid);

in = 8'b00000000; #10; $display(" %b | %b | %b", in, out, valid);

// Finish simulation

$finish;

end

endmodule

1. Write the Verilog module to implement An 2-to-1 multiplexer, develop the testbench to simulate and test this module.

Verilog Module for 2-to-1 Multiplexer:

module mux\_2to1 (

input wire a, // First input

input wire b, // Second input

input wire sel, // Select input

output wire out // Output

);

assign out = (sel) ? b : a;

endmodule

**Testbench for the 2-to-1 Multiplexer:**

module tb\_mux\_2to1;

// Declare inputs as regs and outputs as wires

reg a;

reg b;

reg sel;

wire out;

// Instantiate the Unit Under Test (UUT)

mux\_2to1 uut (

.a(a),

.b(b),

.sel(sel),

.out(out)

);

initial begin

// Apply test vectors

$display("a b sel | out");

a = 0; b = 0; sel = 0; #10; $display("%b %b %b | %b", a, b, sel, out);

a = 0; b = 1; sel = 0; #10; $display("%b %b %b | %b", a, b, sel, out);

a = 1; b = 0; sel = 0; #10; $display("%b %b %b | %b", a, b, sel, out);

a = 1; b = 1; sel = 0; #10; $display("%b %b %b | %b", a, b, sel, out);

a = 0; b = 0; sel = 1; #10; $display("%b %b %b | %b", a, b, sel, out);

a = 0; b = 1; sel = 1; #10; $display("%b %b %b | %b", a, b, sel, out);

a = 1; b = 0; sel = 1; #10; $display("%b %b %b | %b", a, b, sel, out);

a = 1; b = 1; sel = 1; #10; $display("%b %b %b | %b", a, b, sel, out);

// Finish simulation

$finish;

end

endmodule

1. Write the Verilog module to implement the 4-to-1 multiplexer from three 2-to-1 multiplexer, then develop the testbench to simulate and test this module.

**Testbench for the 4-to-1 Multiplexer:**

module tb\_mux\_4to1;

// Declare inputs as regs and outputs as wires

reg [3:0] in;

reg [1:0] sel;

wire out;

// Instantiate the Unit Under Test (UUT)

mux\_4to1 uut (

.in(in),

.sel(sel),

.out(out)

);

initial begin

// Apply test vectors

$display(" in sel | out");

in = 4'b0001; sel = 2'b00; #10; $display("%b %b | %b", in, sel, out);

in = 4'b0010; sel = 2'b01; #10; $display("%b %b | %b", in, sel, out);

in = 4'b0100; sel = 2'b10; #10; $display("%b %b | %b", in, sel, out);

in = 4'b1000; sel = 2'b11; #10; $display("%b %b | %b", in, sel, out);

in = 4'b0110; sel = 2'b00; #10; $display("%b %b | %b", in, sel, out);

in = 4'b0110; sel = 2'b01; #10; $display("%b %b | %b", in, sel, out);

in = 4'b0110; sel = 2'b10; #10; $display("%b %b | %b", in, sel, out);

in = 4'b0110; sel = 2'b11; #10; $display("%b %b | %b", in, sel, out);

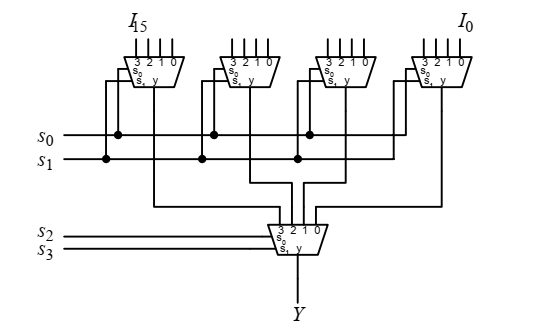
// Finish simulation

$finish;

end

endmodule

1. Write the Verilog module to implement 16-to-1 multiplexer using only 4-to-1 multiplexers, then develop the testbench to simulate and test this module.



**16-to-1 Multiplexer Module using 4-to-1 Multiplexers:**

16-to-1 Multiplexer Module using 4-to-1 Multiplexer module mux\_16to1 (

input wire [15:0] in, // 16-bit input

input wire [3:0] sel, // 4-bit select input

output wire out // Output

);

wire [3:0] mux\_out; // Intermediate outputs from the first layer of multiplexers

// Instantiate the first layer of 4-to-1 multiplexers

mux\_4to1 mux0 (

.in(in[3:0]),

.sel(sel[1:0]),

.out(mux\_out[0])

);

mux\_4to1 mux1 (

.in(in[7:4]),

.sel(sel[1:0]),

.out(mux\_out[1])

);

mux\_4to1 mux2 (

.in(in[11:8]),

.sel(sel[1:0]),

.out(mux\_out[2])

);

mux\_4to1 mux3 (

.in(in[15:12]),

.sel(sel[1:0]),

.out(mux\_out[3])

);

// Instantiate the second layer 4-to-1 multiplexer

mux\_4to1 mux4 (

.in(mux\_out),

.sel(sel[3:2]),

.out(out)

);

endmodule

**Testbench for the 16-to-1 Multiplexer:**

module tb\_mux\_16to1;

// Declare inputs as regs and outputs as wires

reg [15:0] in;

reg [3:0] sel;

wire out;

// Instantiate the Unit Under Test (UUT)

mux\_16to1 uut (

.in(in),

.sel(sel),

.out(out)

);

initial begin

// Apply test vectors

$display(" in sel | out");

in = 16'b0000\_0000\_0000\_0001; sel = 4'b0000; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0000\_0000\_0010; sel = 4'b0001; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0000\_0000\_0100; sel = 4'b0010; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0000\_0000\_1000; sel = 4'b0011; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0000\_0001\_0000; sel = 4'b0100; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0000\_0010\_0000; sel = 4'b0101; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0000\_0100\_0000; sel = 4'b0110; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0000\_1000\_0000; sel = 4'b0111; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0001\_0000\_0000; sel = 4'b1000; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0010\_0000\_0000; sel = 4'b1001; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_0100\_0000\_0000; sel = 4'b1010; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0000\_1000\_0000\_0000; sel = 4'b1011; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0001\_0000\_0000\_0000; sel = 4'b1100; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0010\_0000\_0000\_0000; sel = 4'b1101; #10; $display("%b %b | %b", in, sel, out);

in = 16'b0100\_0000\_0000\_0000; sel = 4'b1110; #10; $display("%b %b | %b", in, sel, out);

in = 16'b1000\_0000\_0000\_0000; sel = 4'b1111; #10; $display("%b %b | %b", in, sel, out);

// Finish simulation

$finish;

end

endmodule

1. Write the Verilog module to implement the 8-to-1 multiplexer using 3-to-8 decoder method and using seven 2-to-1 multiplexers method. Develop the testbench to simulate and compare the operation of these two modules.

**8-to-1 Multiplexer using 3-to-8 Decoder:**

module mux\_8to1\_decoder (

input wire [7:0] in, // 8-bit input

input wire [2:0] sel, // 3-bit select input

output wire out // Output

);

wire [7:0] decoder\_out;

decoder\_3to8 decoder (

.sel(sel),

.out(decoder\_out)

);

assign out = (decoder\_out[0] & in[0]) |

(decoder\_out[1] & in[1]) |

(decoder\_out[2] & in[2]) |

(decoder\_out[3] & in[3]) |

(decoder\_out[4] & in[4]) |

(decoder\_out[5] & in[5]) |

(decoder\_out[6] & in[6]) |

(decoder\_out[7] & in[7]);

endmodule

**8-to-1 Multiplexer using Seven 2-to-1 Multiplexers:**

module mux\_8to1\_seven\_2to1 (

input wire [7:0] in, // 8-bit input

input wire [2:0] sel, // 3-bit select input

output wire out // Output

);

wire [3:0] mux1\_out;

wire [1:0] mux2\_out;

mux\_2to1 mux1\_0 (.a(in[0]), .b(in[1]), .sel(sel[0]), .out(mux1\_out[0]));

mux\_2to1 mux1\_1 (.a(in[2]), .b(in[3]), .sel(sel[0]), .out(mux1\_out[1]));

mux\_2to1 mux1\_2 (.a(in[4]), .b(in[5]), .sel(sel[0]), .out(mux1\_out[2]));

mux\_2to1 mux1\_3 (.a(in[6]), .b(in[7]), .sel(sel[0]), .out(mux1\_out[3]));

mux\_2to1 mux2\_0 (.a(mux1\_out[0]), .b(mux1\_out[1]), .sel(sel[1]), .out(mux2\_out[0]));

mux\_2to1 mux2\_1 (.a(mux1\_out[2]), .b(mux1\_out[3]), .sel(sel[1]), .out(mux2\_out[1]));

mux\_2to1 mux3\_0 (.a(mux2\_out[0]), .b(mux2\_out[1]), .sel(sel[2]), .out(out));

endmodule

**Testbench for 8-to-1 Multiplexers**

module tb\_mux\_8to1;

reg [7:0] in;

reg [2:0] sel;

wire out\_decoder;

wire out\_seven\_2to1;

// Instantiate the Unit Under Test (UUT) - 8-to-1 MUX using decoder

mux\_8to1\_decoder uut\_decoder (

.in(in),

.sel(sel),

.out(out\_decoder)

);

// Instantiate the Unit Under Test (UUT) - 8-to-1 MUX using seven 2-to-1 MUXes

mux\_8to1\_seven\_2to1 uut\_seven\_2to1 (

.in(in),

.sel(sel),

.out(out\_seven\_2to1) );

initial begin

// Apply test vectors

$display(" in sel | out\_decoder out\_seven\_2to1");

in = 8'b00000001; sel = 3'b000; #10; $display("%b %b | %b %b", in, sel, out\_decoder, out\_seven\_2to1);

in = 8'b00000010; sel = 3'b001; #10; $display("%b %b | %b %b", in, sel, out\_decoder, out\_seven\_2to1);

in = 8'b00000100; sel = 3'b010; #10; $display("%b %b | %b %b", in, sel, out\_decoder, out\_seven\_2to1);

in = 8'b00001000; sel = 3'b011; #10; $display("%b %b | %b %b", in, sel, out\_decoder, out\_seven\_2to1);

in = 8'b00010000; sel = 3'b100; #10; $display("%b %b | %b %b", in, sel, out\_decoder, out\_seven\_2to1);

in = 8'b00100000; sel = 3'b101; #10; $display("%b %b | %b %b", in, sel, out\_decoder, out\_seven\_2to1);

in = 8'b01000000; sel = 3'b110; #10; $display("%b %b | %b %b", in, sel, out\_decoder, out\_seven\_2to1);

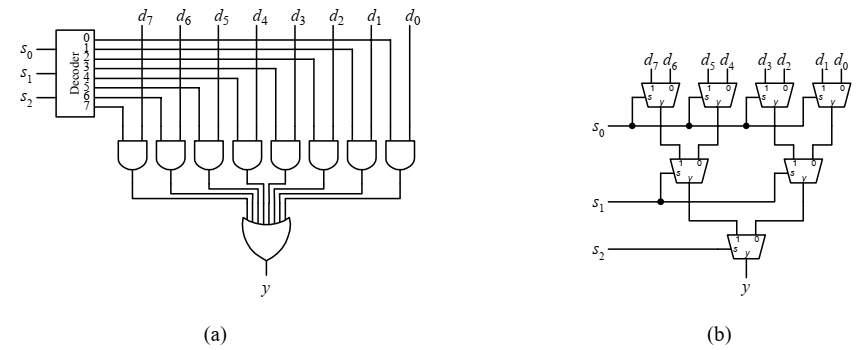
in = 8'b10000000; sel = 3'b111; #10; $display("%b %b | %b %b", in, sel, out\_decoder, out\_seven\_2to1);

// Finish simulation

$finish;

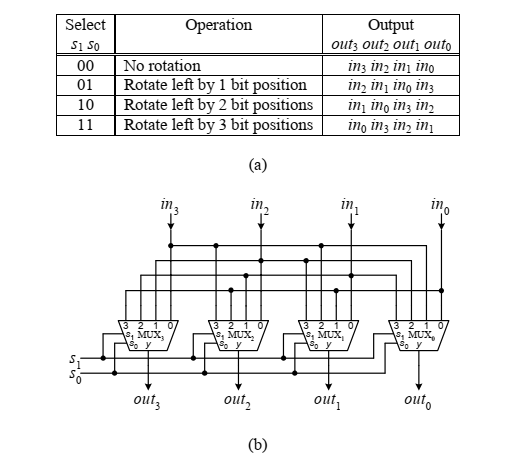
end

endmodule



An 8-to-1 multiplexer implemented using: (a) a 3-to-8 decoder; (b) seven 2-to-1 multiplexers.

1. Write the Verilog module to implement 4-bit barrel shifter for the rotate left operation, then develop the testbench to simulate and test this module.



A 4-bit barrel shifter for the rotate left operation: (a) operation table; (b) circuit.

**Verilog Module for 4-bit Barrel Shifter (Rotate Left):**

module barrel\_shifter\_4bit(

input [3:0] in, // 4-bit input

input [1:0] shift, // 2-bit shift amount

output reg [3:0] out // 4-bit output

);

always @\*

begin

case(shift)

2'b00: out = in; // No shift

2'b01: out = {in[2:0], in[3]}; // Rotate left by 1 bit

2'b10: out = {in[1:0], in[3:2]}; // Rotate left by 2 bits

2'b11: out = {in[0], in[3:1]}; // Rotate left by 3 bits

default: out = in; // Default to no shift

endcase

end

endmodule

**Testbench for Barrel Shifter (Rotate Left):**

module tb\_barrel\_shifter\_4bit;

reg [3:0] in;

reg [1:0] shift;

wire [3:0] out;

// Instantiate the barrel shifter module

barrel\_shifter\_4bit uut (

.in(in),

.shift(shift),

.out(out)

);

// Initialize inputs and apply test vectors

initial begin

// Test case 1: No shift

in = 4'b1010;

shift = 2'b00;

#10;

$display("Input: %b, Shift: %b, Output: %b", in, shift, out);

// Test case 2: Rotate left by 1 bit

in = 4'b1010;

shift = 2'b01;

#10;

$display("Input: %b, Shift: %b, Output: %b", in, shift, out);

// Test case 3: Rotate left by 2 bits

in = 4'b1010;

shift = 2'b10;

#10;

$display("Input: %b, Shift: %b, Output: %b", in, shift, out);

// Test case 4: Rotate left by 3 bits

in = 4'b1010;

shift = 2'b11;

#10;

$display("Input: %b, Shift: %b, Output: %b", in, shift, out);

// Additional test cases can be added here

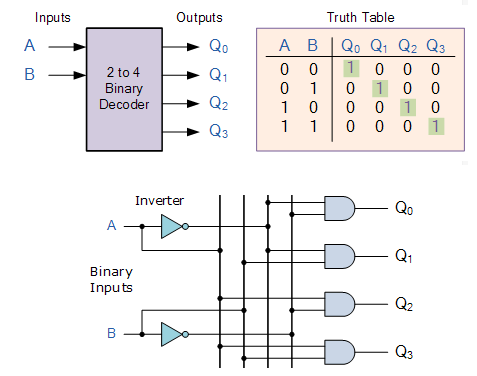
// Finish simulation

$finish;

end

endmodule

1. Write the Verilog module to implement 2-to-4 decoder, then develop the testbench to simulate and test this module.



**Verilog Module for 2-to-4 Decoder:**

module decoder\_2to4(

input [1:0] in, // 2-bit input

output reg [3:0] out // 4-bit output

);

always @\*

begin

case(in)

2'b00: out = 4'b1110; // Output: E3 E2 E1 E0

2'b01: out = 4'b1101; // Output: E3 E2 E1 E0

2'b10: out = 4'b1011; // Output: E3 E2 E1 E0

2'b11: out = 4'b0111; // Output: E3 E2 E1 E0

default: out = 4'b1111; // Default case (should not occur)

endcase

end

endmodule

**Testbench for 2-to-4 Decoder :**

module tb\_decoder\_2to4;

reg [1:0] in; // Input

wire [3:0] out; // Output

// Instantiate the decoder\_2to4 module

decoder\_2to4 uut (

.in(in),

.out(out)

);

// Initialize inputs and apply test vectors

initial begin

// Test case 1: in = 2'b00

in = 2'b00;

#10;

$display("Input: %b, Output: %b", in, out);

// Test case 2: in = 2'b01

in = 2'b01;

#10;

$display("Input: %b, Output: %b", in, out);

// Test case 3: in = 2'b10

in = 2'b10;

#10;

$display("Input: %b, Output: %b", in, out);

// Test case 4: in = 2'b11

in = 2'b11;

#10;

$display("Input: %b, Output: %b", in, out);

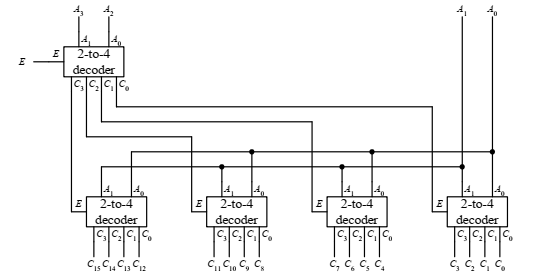
// Additional test cases can be added here

// Finish simulation

$finish;

end

endmodule

1. Write the Verilog module to implement 4-to-16 decoder using only 2-to-4 decoders, then develop the testbench to simulate and test this module.

the circuit for the 4-to-16 decoder using only 2-to-4 decoders.

**Verilog Module for 4-to-16 Decoder using 2-to-4 Decoders**

module decoder\_4to16(

input [3:0] in, // 4-bit input

output reg [15:0] out // 16-bit output

);

wire [3:0] in1, in2;

wire [3:0] in3, in4;

wire [15:0] out1, out2, out3, out4;

decoder\_2to4 d1 (

.in(in[1:0]),

.out(in1)

);

decoder\_2to4 d2 (

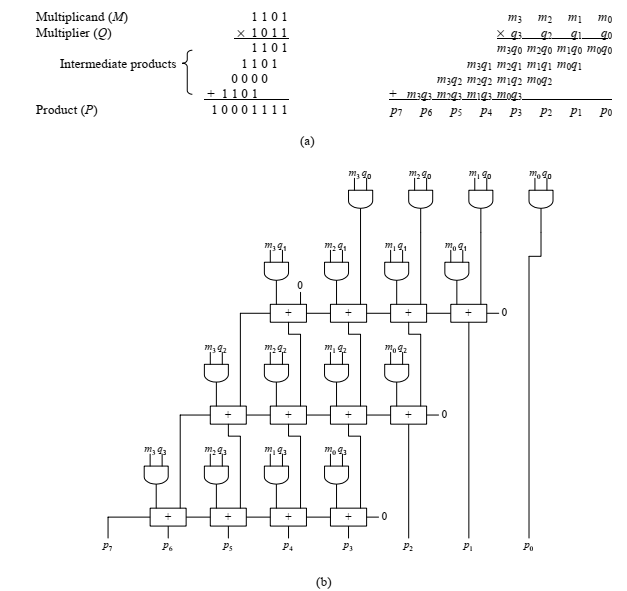
.in(in[3:2]),

.out(in2)

);

assign out1 = 16'h0001;

assign out2 = d1 d2

1. Write the Verilog module to implement 4 bit Multiplication circuit, then develop the testbench to simulate and test this module.

Multiplication: (a) method; (b) circuit.

**Verilog Module for 4-bit Multiplication Circuit :**

module multiplication\_4bit (

input [3:0] a, // 4-bit input A

input [3:0] b, // 4-bit input B

output reg [7:0] result // 8-bit output for the product

);

reg [7:0] temp\_result;

always @\*

begin

temp\_result = a \* b; // Multiplication operation

end

assign result = temp\_result; // Assigning the result to output

endmodule