***Assignment 1***

2\*4 DECOEDER

code : Data flow

module decoder(d0,d1,d2,d3,e,a,b);

input e,a,b;

output d0,d1,d2,d3;

assign d0=e&~a&~b;

assign d1=e&~a&b;

assign d2=e&a&~b;

assign d3=e&a&b;

endmodule

**Behavioural:**

module Decoder2x4\_Behavioral(

input wire [1:0] A,

output wire [3:0] Y

);

assign Y = (A == 2'b00) ? 4'b0001 :

(A == 2'b01) ? 4'b0010 :

(A == 2'b10) ? 4'b0100 :

(A == 2'b11) ? 4'b1000 : 4'b0000;

Endmodule

**Gate-Level:**

module Decoder2x4\_GateLevel(

input wire A0,

input wire A1,

output wire Y0,

output wire Y1,

output wire Y2,

output wire Y3

);

wire NA0, NA1;

not #(.delay\_model("gate") ) not0 (NA0, A0);

not #(.delay\_model("gate") ) not1 (NA1, A1);

and #(.delay\_model("gate") ) and0 (Y0, NA0, NA1);

and #(.delay\_model("gate") ) and1 (Y1, NA0, A1);

and #(.delay\_model("gate") ) and2 (Y2, A0, NA1);

and #(.delay\_model("gate") ) and3 (Y3, A0, A1);

endmodule

testbench

module testbench();

reg t\_e, t\_a, t\_bhb;

wire t\_d0, t\_d1, t\_d2, t\_d3;

decoder dut(.e(t\_e), .a(t\_a), .b(t\_b), .d0(t\_d0), .d1(t\_d1), .d2(t\_d2), .d3(t\_d3));

initial begin

$dumpfile("testbench.vcd");

$dumpvars(0, testbench);

end

initial begin

t\_e = 0; t\_a = 1; t\_bhb = 1;

#100;

t\_e = 1; t\_a = 0; t\_bhb = 0;

#100;

t\_e = 1; t\_a = 0; t\_bhb = 1;

#100;

t\_e = 1; t\_a = 1; t\_bhb = 0;

#100;

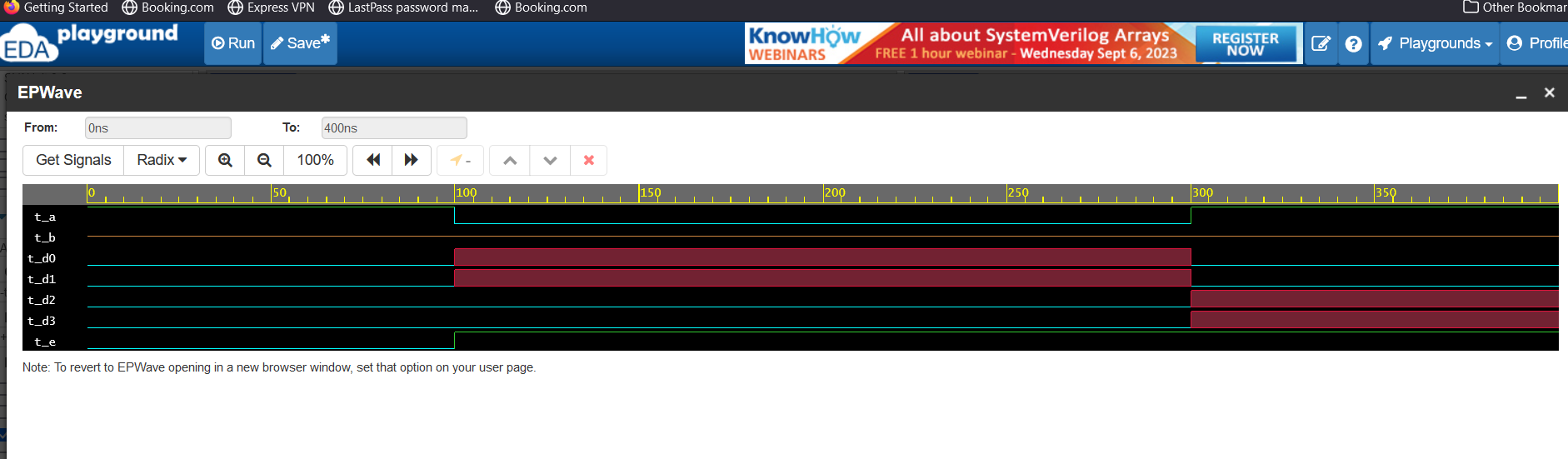
t\_e = 1; t\_a = 1; t\_bhb = 1;

$finish();

end

endmodule

output wave from



code for Full subtractor. Data flow

module FullSubtractor (

input wire A,

input wire B,

input wire BorrowIn,

output wire DiffOut,

output wire BorrowOut

);

assign DiffOut = A ^ B ^ BorrowIn;

assign BorrowOut = (~A & B & BorrowIn) | (A & ~B & ~BorrowIn);

endmodule

**Behavioural:**

module FullSubtractor\_Behavioral(

input wire A,

input wire B,

input wire Bin,

output wire Diff,

output wire Bout

);

assign Diff = A ^ B ^ Bin;

assign Bout = (~A & B) | (Bin & (A ^ B));

endmodule

**Gate-Level:**

module FullSubtractor\_GateLevel(

input wire A,

input wire B,

input wire Bin,

output wire Diff,

output wire Bout

);

wire D1, B1, D2;

xor #(.delay\_model("gate") ) xor0 (D1, A, B);

xor #(.delay\_model("gate") ) xor1 (Diff, D1, Bin);

and #(.delay\_model("gate") ) and0 (B1, ~A, B);

and #(.delay\_model("gate") ) and1 (D2, Bin, D1);

or #(.delay\_model("gate") ) or0 (Bout, B1, D2);

endmodule

testbench

module Testbench\_FullSubtractor;

// Instantiate the FullSubtractor module

FullSubtractor dut (

.A(A),

.B(B),

.BorrowIn(BorrowIn),

.DiffOut(DiffOut),

.BorrowOut(BorrowOut)

);

// Declare signals for testbench

reg A, B, BorrowIn;

wire DiffOut, BorrowOut;

// Clock generation

reg clk = 0;

always #5 clk = ~clk;

// Stimulus generation

initial begin

$dumpfile("full\_subtractor.vcd");

$dumpvars(0, Testbench\_FullSubtractor);

// Test Case 1

A = 1'b0;

B = 1'b0;

BorrowIn = 1'b0;

#10;

// Test Case 2

A = 1'b1;

B = 1'b0;

BorrowIn = 1'b0;

#10;

// Test Case 3

A = 1'b0;

B = 1'b1;

BorrowIn = 1'b0;

#10;

// Test Case 4

A = 1'b1;

B = 1'b1;

BorrowIn = 1'b0;

#10;

// Test Case 5

A = 1'b0;

B = 1'b0;

BorrowIn = 1'b1;

#10;

// Test Case 6

A = 1'b1;

B = 1'b0;

BorrowIn = 1'b1;

#10;

// Test Case 7

A = 1'b0;

B = 1'b1;

BorrowIn = 1'b1;

#10;

// Test Case 8

A = 1'b1;

B = 1'b1;

BorrowIn = 1'b1;

#10;

$finish;

end

// Monitor for displaying outputs

always @(posedge clk) begin

$display("Time: %t, A: %b, B: %b, BorrowIn: %b, DiffOut: %b, BorrowOut: %b", $time, A, B, BorrowIn, DiffOut, BorrowOut);

end

endmodule

wave from

A screen shot of a computer

Description automatically generated

code for 2-bit comparator. **Behavioural:**

module Comparator2Bit (

input wire [1:0] A,

input wire [1:0] B,

output wire Equal,

output wire AGreaterThanB,

output wire ALessThanB

);

assign Equal = (A == B);

assign AGreaterThanB = (A > B);

assign ALessThanB = (A < B);

endmodule

**Gate-Level:**

module Comparator2bit\_GateLevel(

input wire A0,

input wire A1,

input wire B0,

input wire B1,

output wire A\_greater,

output wire B\_greater,

output wire Equal

);

wire A0\_B0, A1\_B1, A0\_eq\_B0, A1\_eq\_B1;

xor #(.delay\_model("gate")) xor0 (A0\_B0, A0, B0);

xor #(.delay\_model("gate")) xor1 (A1\_B1, A1, B1);

xor #(.delay\_model("gate")) xor2 (A0\_eq\_B0, A0, B0);

xor #(.delay\_model("gate")) xor3 (A1\_eq\_B1, A1, B1);

not #(.delay\_model("gate")) not0 (A\_greater, A1\_B1);

not #(.delay\_model("gate")) not1 (B\_greater, A0\_B0);

not #(.delay\_model("gate")) not2 (Equal, A0\_eq\_B0, A1\_eq\_B1);

endmodule

testbench

module Testbench\_Comparator2Bit;

// Instantiate the Comparator2Bit module

Comparator2Bit dut (

.A(A),

.B(B),

.Equal(Equal),

.AGreaterThanB(AGreaterThanB),

.ALessThanB(ALessThanB)

);

// Declare signals for testbench

reg [1:0] A, B;

wire Equal, AGreaterThanB, ALessThanB;

// Clock generation

reg clk = 0;

always #5 clk = ~clk;

// Stimulus generation

initial begin

$dumpfile("comparator\_2bit.vcd");

$dumpvars(0, Testbench\_Comparator2Bit);

// Test Case 1

A = 2'b00;

B = 2'b00;

#10;

// Test Case 2

A = 2'b01;

B = 2'b00;

#10;

// Test Case 3

A = 2'b00;

B = 2'b01;

#10;

// Test Case 4

A = 2'b10;

B = 2'b11;

#10;

// Test Case 5

A = 2'b11;

B = 2'b10;

#10;

$finish;

end

// Monitor for displaying outputs

always @(posedge clk) begin

$display("Time: %t, A: %b, B: %b, Equal: %b, AGreaterThanB: %b, ALessThanB: %b",

$time, A, B, Equal, AGreaterThanB, ALessThanB);

end

endmodule

wave from

++A screen shot of a computer

Description automatically generated

code for 3 bit binary to gray convertor: **Behavioural:**

module BinaryToGray3Bit (

input wire [2:0] binary,

output wire [2:0] gray

);

assign gray[2] = binary[2];

assign gray[1] = binary[2] ^ binary[1];

assign gray[0] = binary[1] ^ binary[0];

endmodule

**Gate-Level:**

module BinaryToGray3bit\_GateLevel(

input wire Bin0,

input wire Bin1,

input wire Bin2,

output wire Gray0,

output wire Gray1,

output wire Gray2

);

wire G0, G1;

xor #(.delay\_model("gate")) xor0 (G0, Bin0, Bin1);

xor #(.delay\_model("gate")) xor1 (G1, Bin1, Bin2);

xor #(.delay\_model("gate")) xor2 (Gray0, G0, Bin2);

xor #(.delay\_model("gate")) xor3 (Gray1, G1, Bin0);

xor #(.delay\_model("gate")) xor4 (Gray2, G1, G0);

endmodule

**Data Flow:**

module BinaryToGray3bit\_DataFlow(

input wire [2:0] Bin,

output wire [2:0] Gray

);

assign Gray[0] = Bin[0] ^ Bin[1];

assign Gray[1] = Bin[1] ^ Bin[2];

assign Gray[2] = Bin[2];

endmodule

testbench

module Testbench\_BinaryToGray3Bit;

// Instantiate the BinaryToGray3Bit module

BinaryToGray3Bit dut (

.binary(binary),

.gray(gray)

);

// Declare signals for testbench

reg [2:0] binary;

wire [2:0] gray;

// Clock generation

reg clk = 0;

always #5 clk = ~clk;

// Stimulus generation

initial begin

$dumpfile("binary\_to\_gray\_3bit.vcd");

$dumpvars(0, Testbench\_BinaryToGray3Bit);

// Test Case 1

binary = 3'b000;

#10;

// Test Case 2

binary = 3'b001;

#10;

// Test Case 3

binary = 3'b010;

#10;

// Test Case 4

binary = 3'b011;

#10;

// Test Case 5

binary = 3'b100;

#10;

// Test Case 6

binary = 3'b101;

#10;

// Test Case 7

binary = 3'b110;

#10;

// Test Case 8

binary = 3'b111;

#10;

$finish;

end

// Monitor for displaying outputs

always @(posedge clk) begin

$display("Time: %t, Binary: %b, Gray: %b", $time, binary, gray);

end

endmodule

A computer screen shot of a person

Description automatically generated

code for BCD to excess 3 convertors: data flow.

module BCDtoExcess3 (

input wire [3:0] bcd,

output wire [3:0] xs3

);

assign xs3[3] = bcd[3] ^ bcd[2];

assign xs3[2] = bcd[2] ^ bcd[1];

assign xs3[1] = bcd[1] ^ bcd[0];

assign xs3[0] = bcd[0];

endmodule

**Behavioural:**

**module BCDtoExcess3\_Behavioral(**

**input wire [3:0] BCD,**

**output wire [3:0] Excess3**

**);**

**assign Excess3 = BCD + 4'b0011;**

**endmodule**

**Gate-Level:**

module BCDtoExcess3\_GateLevel(

input wire BCD0,

input wire BCD1,

input wire BCD2,

input wire BCD3,

output wire Excess30,

output wire Excess31,

output wire Excess32,

output wire Excess33

);

wire BCD0\_xor, BCD1\_xor, BCD2\_xor, BCD3\_xor;

wire BCD0\_and, BCD1\_and, BCD2\_and, BCD3\_and;

xor #(.delay\_model("gate")) xor0 (BCD0\_xor, BCD0);

xor #(.delay\_model("gate")) xor1 (BCD1\_xor, BCD1);

xor #(.delay\_model("gate")) xor2 (BCD2\_xor, BCD2);

xor #(.delay\_model("gate")) xor3 (BCD3\_xor, BCD3);

and #(.delay\_model("gate")) and0 (BCD0\_and, BCD1\_xor, BCD2\_xor);

and #(.delay\_model("gate")) and1 (BCD1\_and, BCD0, BCD2\_xor, BCD3\_xor);

and #(.delay\_model("gate")) and2 (BCD2\_and, BCD0, BCD1, BCD3\_xor);

and #(.delay\_model("gate")) and3 (BCD3\_and, BCD0, BCD1, BCD2);

xor #(.delay\_model("gate")) xor4 (Excess30, BCD0\_and, BCD3\_xor);

xor #(.delay\_model("gate")) xor5 (Excess31, BCD1\_and, BCD2\_and);

xor #(.delay\_model("gate")) xor6 (Excess32, BCD0\_and, BCD1\_and, BCD3\_xor);

xor #(.delay\_model("gate")) xor7 (Excess33, BCD0\_and, BCD1\_and, BCD2\_and);

endmodule

testbench

module Testbench\_BCDtoExcess3;

// Instantiate the BCDtoExcess3 module

BCDtoExcess3 dut (

.bcd(bcd),

.xs3(xs3)

);

// Declare signals for testbench

reg [3:0] bcd;

wire [3:0] xs3;

// Clock generation

reg clk = 0;

always #5 clk = ~clk;

// Stimulus generation

initial begin

$dumpfile("bcd\_to\_xs3.vcd");

$dumpvars(0, Testbench\_BCDtoExcess3);

// Test Case 1

bcd = 4'b0000;

#10;

// Test Case 2

bcd = 4'b0001;

#10;

// Test Case 3

bcd = 4'b0101;

#10;

// Test Case 4

bcd = 4'b1100;

#10;

// Test Case 5

bcd = 4'b1010;

#10;

$finish;

end

// Monitor for displaying outputs

always @(posedge clk) begin

$display("Time: %t, BCD: %b, XS-3: %b", $time, bcd, xs3);

end

endmodule

A screen shot of a video

Description automatically generated

***Assignments-2***

Design 4-bit Ripple Carry Adder with the help of 1-bit adder

module OneBitAdder (

input wire A,

input wire B,

input wire Cin,

output wire Sum,

output wire Cout

);

assign Sum = A ^ B ^ Cin;

assign Cout = (A & B) | (B & Cin) | (A & Cin);

endmodule

module FourBitRippleCarryAdder (

input wire [3:0] A,

input wire [3:0] B,

output wire [3:0] Sum,

output wire Cout

);

wire [3:0] sum\_intermediate;

wire carry1, carry2, carry3;

OneBitAdder adder0 (.A(A[0]), .B(B[0]), .Cin(1'b0), .Sum(Sum[0]), .Cout(carry1));

OneBitAdder adder1 (.A(A[1]), .B(B[1]), .Cin(carry1), .Sum(Sum[1]), .Cout(carry2));

OneBitAdder adder2 (.A(A[2]), .B(B[2]), .Cin(carry2), .Sum(Sum[2]), .Cout(carry3));

OneBitAdder adder3 (.A(A[3]), .B(B[3]), .Cin(carry3), .Sum(Sum[3]), .Cout(Cout));

endmodule

testbench

module Testbench\_FourBitRippleCarryAdder;

// Instantiate the FourBitRippleCarryAdder module

FourBitRippleCarryAdder dut (

.A(A),

.B(B),

.Sum(Sum),

.Cout(Cout)

);

// Declare signals for testbench

reg [3:0] A, B;

wire [3:0] Sum;

wire Cout;

// Clock generation

reg clk = 0;

always #5 clk = ~clk;

// Stimulus generation

initial begin

$dumpfile("ripple\_carry\_adder.vcd");

$dumpvars(0, Testbench\_FourBitRippleCarryAdder);

// Test Case 1

A = 4'b0000;

B = 4'b0000;

#10;

// Test Case 2

A = 4'b0101;

B = 4'b0011;

#10;

// Test Case 3

A = 4'b1111;

B = 4'b0001;

#10;

// Test Case 4

A = 4'b1100;

B = 4'b1011;

#10;

$finish;

end

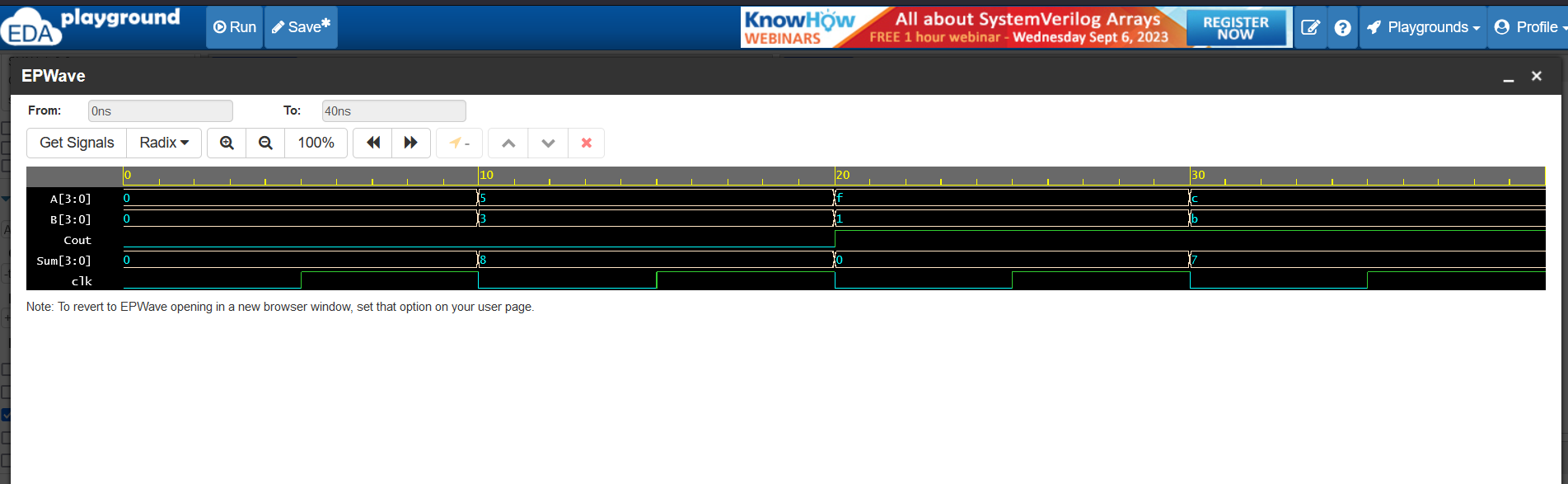
// Monitor for displaying outputs

always @(posedge clk) begin

$display("Time: %t, A: %b, B: %b, Sum: %b, Cout: %b", $time, A, B, Sum, Cout);

end

endmodule



Design D-flipflop and reuse it to implement 4- bit Johnson Counter.

module DFlipFlop (

input wire D,

input wire clk,

input wire reset,

output reg Q

);

always @(posedge clk or posedge reset) begin

if (reset)

Q <= 1'b0;

else

Q <= D;

end

endmodule

module JohnsonCounter4Bit (

input wire clk,

input wire reset,

output wire [3:0] Q

);

wire [3:0] d\_values;

assign d\_values[0] = Q[3] ^ 1'b1;

assign d\_values[1] = d\_values[0] ^ 1'b1;

assign d\_values[2] = d\_values[1] ^ 1'b1;

assign d\_values[3] = d\_values[2] ^ 1'b1;

DFlipFlop ff0 (.D(d\_values[0]), .clk(clk), .reset(reset), .Q(Q[0]));

DFlipFlop ff1 (.D(d\_values[1]), .clk(clk), .reset(reset), .Q(Q[1]));

DFlipFlop ff2 (.D(d\_values[2]), .clk(clk), .reset(reset), .Q(Q[2]));

DFlipFlop ff3 (.D(d\_values[3]), .clk(clk), .reset(reset), .Q(Q[3]));

Endmodule

module Testbench\_JohnsonCounter4Bit;

// Instantiate the JohnsonCounter4Bit module

JohnsonCounter4Bit dut (

.clk(clk),

.reset(reset),

.Q(Q)

);

// Declare signals for testbench

reg clk = 0;

reg reset = 0;

wire [3:0] Q;

// Clock generation

always #5 clk = ~clk;

// Reset generation

initial begin

reset = 1;

#10;

reset = 0;

end

// Stimulus generation

initial begin

$dumpfile("johnson\_counter.vcd");

$dumpvars(0, Testbench\_JohnsonCounter4Bit);

repeat (20) begin

#10;

end

$finish;

end

// Monitor for displaying outputs

always @(posedge clk) begin

$display("Time: %t, Q: %b", $time, Q);

end

endmodule

A screen shot of a computer

Description automatically generated

Reuse 2:1 Mux code to implement 8:1 Mux.

module Mux2to1 (

input wire S,

input wire D0,

input wire D1,

output reg Y

);

always @\* begin

case (S)

1'b0: Y = D0;

1'b1: Y = D1;

default: Y = 1'bx; // Handle undefined case

endcase

end

endmodule

module Mux8to1 (

input wire [2:0] S,

input wire [7:0] D,

output reg Y

);

wire [1:0] sel1, sel2;

wire [3:0] sel3;

Mux2to1 mux0 (.S(S[0]), .D(D[0], D[1]), .Y(sel1));

Mux2to1 mux1 (.S(S[0]), .D(D[2], D[3]), .Y(sel2));

Mux2to1 mux2 (.S(S[0]), .D(D[4], D[5]), .Y(sel3[0]));

Mux2to1 mux3 (.S(S[0]), .D(D[6], D[7]), .Y(sel3[1]));

Mux2to1 mux4 (.S(S[1]), .D(sel1, sel2), .Y(sel3[2]));

Mux2to1 mux5 (.S(S[1]), .D(D[8], D[9]), .Y(sel3[3]));

Mux2to1 mux6 (.S(S[2]), .D(sel3[0], sel3[1]), .Y(Y));

Mux2to1 mux7 (.S(S[2]), .D(sel3[2], sel3[3]), .Y(Y));

endmodule

testbench

module Testbench\_8to1Mux;

// Instantiate the Mux8to1 module

Mux8to1 dut (

.S(S),

.D(D),

.Y(Y)

);

// Declare signals for testbench

reg [2:0] S;

reg [7:0] D;

wire Y;

// Clock generation

reg clk = 0;

always #5 clk = ~clk;

// Stimulus generation

initial begin

$dumpfile("mux\_8to1.vcd");

$dumpvars(0, Testbench\_8to1Mux);

// Test Case 1

S = 3'b000;

D = 8'b11001100;

#10;

// Test Case 2

S = 3'b101;

D = 8'b10101010;

#10;

// Test Case 3

S = 3'b010;

D = 8'b00011101;

#10;

$finish;

end

// Monitor for displaying outputs

always @(posedge clk) begin

$display("Time: %t, S: %b, D: %b, Y: %b", $time, S, D, Y);

end

endmodule

output wave from:

A screenshot of a computer

Description automatically generated

Design a Full Subtractor with Gate Level Modeling Style.(use primitive gates)

module FullSubtractor(

input wire A,

input wire B,

input wire Bin,

output wire Diff,

output wire Bout

);

wire X1, X2, X3, X4, X5, X6;

// XOR gates

xor gate1 (X1, A, B);

xor gate2 (X2, X1, Bin);

// NOR gate

nor gate3 (X3, X1, Bin);

// NAND gates

nand gate4 (X4, A, B);

nand gate5 (X5, X1, Bin);

// OR gate

or gate6 (X6, X4, X5);

assign Diff = X2;

assign Bout = X6;

endmodule

testbench

module Testbench\_FullSubtractor;

reg A, B, Bin;

wire Diff, Bout;

FullSubtractor dut (

.A(A),

.B(B),

.Bin(Bin),

.Diff(Diff),

.Bout(Bout)

);

initial begin

$dumpfile("full\_subtractor.vcd");

$dumpvars(0, Testbench\_FullSubtractor);

// Test Case 1

A = 0; B = 0; Bin = 0;

#10;

$display("Test Case 1: A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);

// Test Case 2

A = 0; B = 0; Bin = 1;

#10;

$display("Test Case 2: A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);

// Test Case 3

A = 0; B = 1; Bin = 0;

#10;

$display("Test Case 3: A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);

// Test Case 4

A = 0; B = 1; Bin = 1;

#10;

$display("Test Case 4: A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);

// Test Case 5

A = 1; B = 0; Bin = 0;

#10;

$display("Test Case 5: A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);

// Test Case 6

A = 1; B = 0; Bin = 1;

#10;

$display("Test Case 6: A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);

// Test Case 7

A = 1; B = 1; Bin = 0;

#10;

$display("Test Case 7: A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);

// Test Case 8

A = 1; B = 1; Bin = 1;

#10;

$display("Test Case 8: A=%b, B=%b, Bin=%b, Diff=%b, Bout=%b", A, B, Bin, Diff, Bout);

$finish;

end

endmodule

A screen shot of a computer

Description automatically generated

Design a 2X4 decoder using gate level modelling

module Decoder2x4(

input wire A,

input wire B,

output wire Y0,

output wire Y1,

output wire Y2,

output wire Y3

);

wire NA, NB;

// Inverters

not gate1 (NA, A);

not gate2 (NB, B);

// AND gates

and gate3 (Y0, NA, NB);

and gate4 (Y1, NA, B);

and gate5 (Y2, A, NB);

and gate6 (Y3, A, B);

endmodule

testbench

module Testbench\_Decoder2x4;

reg A, B;

wire Y0, Y1, Y2, Y3;

Decoder2x4 dut (

.A(A),

.B(B),

.Y0(Y0),

.Y1(Y1),

.Y2(Y2),

.Y3(Y3)

);

initial begin

$dumpfile("decoder\_2x4.vcd");

$dumpvars(0, Testbench\_Decoder2x4);

// Test Case 1

A = 0; B = 0;

#10;

$display("Test Case 1: A=%b, B=%b, Y0=%b, Y1=%b, Y2=%b, Y3=%b", A, B, Y0, Y1, Y2, Y3);

// Test Case 2

A = 0; B = 1;

#10;

$display("Test Case 2: A=%b, B=%b, Y0=%b, Y1=%b, Y2=%b, Y3=%b", A, B, Y0, Y1, Y2, Y3);

// Test Case 3

A = 1; B = 0;

#10;

$display("Test Case 3: A=%b, B=%b, Y0=%b, Y1=%b, Y2=%b, Y3=%b", A, B, Y0, Y1, Y2, Y3);

// Test Case 4

A = 1; B = 1;

#10;

$display("Test Case 4: A=%b, B=%b, Y0=%b, Y1=%b, Y2=%b, Y3=%b", A, B, Y0, Y1, Y2, Y3);

$finish;

end

endmodule

A screen shot of a computer

Description automatically generated

Design a 4x1 mux using operators. (use data flow)

module Mux4x1(

input wire [3:0] S,

input wire [3:0] D,

output wire Y

);

assign Y = (S[3]) ? D[3] :

(S[2]) ? D[2] :

(S[1]) ? D[1] :

D[0];

endmodule

testbench

module Testbench\_Mux4x1;

reg [3:0] S;

reg [3:0] D;

wire Y;

Mux4x1 dut (

.S(S),

.D(D),

.Y(Y)

);

initial begin

$dumpfile("mux\_4x1.vcd");

$dumpvars(0, Testbench\_Mux4x1);

// Test Case 1

S = 4'b0000;

D = 4'b1010;

#10;

$display("Test Case 1: S=%b, D=%b, Y=%b", S, D, Y);

// Test Case 2

S = 4'b0001;

D = 4'b1100;

#10;

$display("Test Case 2: S=%b, D=%b, Y=%b", S, D, Y);

// Test Case 3

S = 4'b0010;

D = 4'b0011;

#10;

$display("Test Case 3: S=%b, D=%b, Y=%b", S, D, Y);

// Test Case 4

S = 4'b0011;

D = 4'b0110;

#10;

$display("Test Case 4: S=%b, D=%b, Y=%b", S, D, Y);

$finish;

end

endmodule

output wave from

A screen shot of a computer

Description automatically generated

Design a Full adder using half adder.

module HalfAdder(

input wire A,

input wire B,

output wire Sum,

output wire Carry

);

assign Sum = A ^ B;

assign Carry = A & B;

endmodule

module FullAdder(

input wire A,

input wire B,

input wire Cin,

output wire Sum,

output wire Cout

);

wire H1\_Sum, H1\_Carry, H2\_Carry;

HalfAdder HA1 (.A(A), .B(B), .Sum(H1\_Sum), .Carry(H1\_Carry));

HalfAdder HA2 (.A(H1\_Sum), .B(Cin), .Sum(Sum), .Carry(H2\_Carry));

assign Cout = H1\_Carry | H2\_Carry;

endmodule

testbench

module Testbench\_FullAdder;

reg A, B, Cin;

wire Sum, Cout;

FullAdder dut (

.A(A),

.B(B),

.Cin(Cin),

.Sum(Sum),

.Cout(Cout)

);

initial begin

$dumpfile("full\_adder.vcd");

$dumpvars(0, Testbench\_FullAdder);

// Test Case 1

A = 1; B = 0; Cin = 0;

#10;

$display("Test Case 1: A=%b, B=%b, Cin=%b, Sum=%b, Cout=%b", A, B, Cin, Sum, Cout);

// Test Case 2

A = 1; B = 1; Cin = 0;

#10;

$display("Test Case 2: A=%b, B=%b, Cin=%b, Sum=%b, Cout=%b", A, B, Cin, Sum, Cout);

// Test Case 3

A = 0; B = 0; Cin = 1;

#10;

$display("Test Case 3: A=%b, B=%b, Cin=%b, Sum=%b, Cout=%b", A, B, Cin, Sum, Cout);

// Test Case 4

A = 1; B = 0; Cin = 1;

#10;

$display("Test Case 4: A=%b, B=%b, Cin=%b, Sum=%b, Cout=%b", A, B, Cin, Sum, Cout);

$finish;

end

endmodule

A screen shot of a computer

Description automatically generated