1.DECODER:

module Decoder(

input wire [2:0] B,

input wire E,

output reg [7:0] D

);

// // Method -> Assignment operator

// assign D[0] = ~B[0] & ~B[1] & ~B[2] & E;

// assign D[1] = B[0] & ~B[1] & ~B[2] & E;

// assign D[2] = ~B[0] & B[1] & ~B[2] & E;

// assign D[3] = B[0] & B[1] & ~B[2] & E;

// assign D[4] = ~B[0] & ~B[1] & B[2] & E;

// assign D[5] = B[0] & ~B[1] & B[2] & E;

// assign D[6] = ~B[0] & B[1] & B[2] & E;

// assign D[7] = B[0] & B[1] & B[2] & E;

// // Method -> If-Else Statement

// always@(\*) begin

// if(E == 1'b0)

// D = 8'b00000000;

// else if(B == 3'b000)

// D = 8'b00000001;

// else if(B == 3'b001)

// D = 8'b00000010;

// else if(B == 3'b010)

// D = 8'b00000100;

// else if(B == 3'b011)

// D = 8'b00001000;

// else if(B == 3'b100)

// D = 8'b00010000;

// else if(B == 3'b101)

// D = 8'b00100000;

// else if(B == 3'b110)

// D = 8'b01000000;

// else if(B == 3'b111)

// D = 8'b10000000;

// else

// D = 8'b00000000;

// end

// Method -> Case Statement

always@(\*) begin

case({E, B})

4'b1000: D = 8'b00000001;

4'b1001: D = 8'b00000010;

4'b1010: D = 8'b00000100;

4'b1011: D = 8'b00001000;

4'b1100: D = 8'b00010000;

4'b1101: D = 8'b00100000;

4'b1110: D = 8'b01000000;

4'b1111: D = 8'b10000000;

default: D = 8'b00000000;

endcase

end

endmodule

Test---🡪

`timescale 1ns/1ps

module Decoder\_Tb;

reg [2:0] B;

reg E;

wire [7:0] D;

Decoder cir1(

.B(B),

.D(D),

.E(E)

);

initial begin

$dumpfile("./out/Decoder\_gtk.vcd");

$dumpvars;

E = 0;

#10;

E = 1;

B = 3'b000;

#10;

B = 3'b001;

#10;

B = 3'b010;

#10;

B = 3'b011;

#10;

B = 3'b100;

#10;

B = 3'b101;

#10;

B = 3'b110;

#10;

B = 3'b111;

#10;

end

initial begin

$monitor("E = %b, B = %3b | D = %8b", E, B, D);

end

endmodule

DEMULTIPLEXER:

module DeMultiplexer(

input wire Y,

input wire [1:0] S,

input wire E,

output reg [3:0] I

);

// assign I[0] = E & ~S[0] & ~S[1] & Y;

// assign I[1] = E & S[0] & ~S[1] & Y;

// assign I[2] = E & ~S[0] & S[1] & Y;

// assign I[3] = E & S[0] & S[1] & Y;

// always @(\*) begin

// if(E == 1'b0)

// I = 4'b0000;

// else if(S == 2'b00) begin

// I = 4'b0000;

// I[0] = Y;

// end

// else if(S == 2'b01) begin

// I = 4'b0000;

// I[1] = Y;

// end

// else if(S == 2'b10) begin

// I = 4'b0000;

// I[2] = Y;

// end

// else if(S == 2'b11) begin

// I = 4'b0000;

// I[3] = Y;

// end

// else

// I = 4'b0000;

// end

always @(\*) begin

case({E, S})

3'b100 : begin

I = 4'b0000;

I[0] = Y;

end

3'b101 : begin

I = 4'b0000;

I[1] = Y;

end

3'b110 : begin

I = 4'b0000;

I[2] = Y;

end

3'b111 : begin

I = 4'b0000;

I[3] = Y;

end

default : I = 4'b0000;

endcase

end

endmodule

Test-🡪

`timescale 1ns/1ps

module DeMultiplexer\_Tb;

reg Y;

reg [1:0] S;

reg E;

wire [3:0] I;

DeMultiplexer circuit(Y, S, E, I);

initial begin

$dumpfile("./out/DeMultiplexer\_gtk.vcd");

$dumpvars(0, DeMultiplexer\_Tb);

E = 0;

#10;

E = 1;

Y = 1'b1;

#10;

S = 2'b00;

#10;

S = 2'b01;

#10;

S = 2'b10;

#10;

S = 2'b11;

#10;

end

initial

$monitor("E = %b, Y = %b, S = %2b | I = %4b", E, Y, S, I);

Endmodule

MULTIPLEXER:

module Multiplexer(

input wire [3:0] I,

input wire [1:0] S,

input wire E,

output reg Y

);

// assign Y = E & ((~S[0] & ~S[1] & I[0]) + (S[0] & ~S[1] & I[1]) + (~S[0] & S[1] & I[2]) + (S[0] & S[1] & I[3]));

// always @(\*) begin

// if(E == 1'b0) Y = 1'b0;

// else if(S == 2'b00) Y = I[0];

// else if(S == 2'b01) Y = I[1];

// else if(S == 2'b10) Y = I[2];

// else if(S == 2'b11) Y = I[3];

// else Y = 1'b0;

// end

always @(\*) begin

case({E, S})

3'b100 : Y = I[0];

3'b101 : Y = I[1];

3'b110 : Y = I[2];

3'b111 : Y = I[3];

default : Y = 1'b0;

endcase

end

endmodule

TEST🡪

`timescale 1ns/1ps

module Multiplexer\_Tb;

reg [3:0] I;

reg [1:0] S;

reg E;

wire Y;

Multiplexer circuit(I, S, E, Y);

initial begin

$dumpfile("./out/Multiplexer\_gtk.vcd");

$dumpvars(0, Multiplexer\_Tb);

E = 0;

#10;

E = 1;

I = 4'b0101;

#10;

S = 2'b00;

#10;

S = 2'b01;

#10;

S = 2'b10;

#10;

S = 2'b11;

#10;

end

initial

$monitor("E = %b, I = %4b, S = %2b | Y = %b", E, I, S, Y);

Endmodule

PRIORITY ENCOER:

module Priority\_Encoder(

input wire E,

input wire [3:0] D,

output reg [1:0] B

);

// assign B[1] = ((D[2] & ~D[3]) + D[3]) & E;

// assign B[0] = ((D[1] & ~D[2] & ~D[3]) + D[3]) & E;

// always @(\*) begin

// if(E == 1'b0)

// B = 2'b00;

// else if(D[0] == 1'b1 && D[3:1] == 3'b000)

// B = 2'b00;

// else if(D[1] == 1 && D[3:2] == 2'b00)

// B = 2'b01;

// else if(D[2] == 1'b1 && D[3] == 1'b0)

// B = 2'b10;

// else if(D[3] == 1'b1)

// B = 2'b11;

// else B = 2'b00;

// end

always @(\*) begin

casex({E, D})

5'b10001: B = 2'b00;

5'b1001x: B = 2'b01;

5'b101xx: B = 2'b10;

5'b11xxx: B = 2'b11;

default: B = 2'b00;

endcase

end

endmodule

Test🡪

`timescale 1ns/1ps

module Priority\_Encoder\_Tb;

reg E;

reg [3:0] D;

wire [1:0] B;

Priority\_Encoder circuit(E, D, B);

initial begin

$dumpfile("./out/Priority\_encoder\_gtk.vcd");

$dumpvars(0, Priority\_Encoder\_Tb);

E = 0;

#10;

E = 1;

D = 4'b0001;

#10;

D = 4'b0010;

#10;

D = 4'b0100;

#10;

D = 4'b1000;

#10;

D = 4'b0011;

#10;

D = 4'b0110;

#10;

D = 4'b1100;

#10;

D = 4'b1010;

#10;

end

initial

$monitor("E = %b, D = %4b | B = %2b", E, D, B);

Endmodule

Counter:

// Problem - 5

module DownCounter(

input clk,

input reset,

output [3:0] q

);

// Memory Logic

wire [3:0] r\_next;

reg [3:0] r\_reg;

always @(posedge clk, posedge reset) begin

if(reset == 1'b1)

r\_reg <= 4'b0000;

else

r\_reg <= r\_next;

end

// Next State Logic

assign r\_next = r\_reg - 1;;

// Outptu Logic

assign q = r\_reg;

// always @(\*) begin

// q <= r\_reg;

// end

Endmodule

TESt🡪

// Problem - 5

`timescale 1ns/1ns

module DownCounter\_Tb;

reg clk;

reg reset;

wire [3:0] q;

DownCounter counter(clk, reset, q);

always begin

#10 clk = ~clk;

end

initial begin

$dumpfile("./out/DownConter1Module\_gtk.vcd");

$dumpvars(0, DownCounter\_Tb);

clk <= 0;

reset <= 0;

#20;

reset <= 1;

#20;

reset <= 0;

#20;

#20;

#20;

#20;

#20;

reset <= 1;

#20;

$finish;

end

initial

$monitor("clk = %b, reset = %b, q = %4b", clk, reset, q);

Endmodule

Syn counter:  
// Problem - 4

module Register4\_Synchronous (

input wire [3:0] D,

input wire clk,

input wire reset,

output wire [3:0] Q

);

wire [3:0] r\_next;

reg [3:0] r\_reg;

// Memory

always @(posedge clk, posedge reset) begin

if(reset == 1'b1)

r\_reg <= 4'b0000;

else

r\_reg <= r\_next;

end

// Next State Logic

assign r\_next = D;

// Output Logic

assign Q = r\_reg;

endmodule

module subOne(

input wire [3:0] I,

output wire [3:0] O

);

assign O = I - 1;

endmodule

module DownCounter4bit\_Synchronous(

input wire clk,

input wire reset,

output wire [3:0] q

);

Register4\_Synchronous register(r\_next, clk, reset, r\_reg);

subOne subtractor(r\_reg, r\_next);

// Output Logic

assign q = r\_reg;

endmodule

TESt🡪

// Problem - 4

`timescale 1ns/1ns

module DownCounter4bit\_Synchronous\_Tb;

reg clk;

reg reset;

wire [3:0] q;

DownCounter4bit\_Synchronous counter(clk, reset, q);

always begin

#10 clk = ~clk;

end

initial begin

$dumpfile("./out/Counter4bit\_Synchronous\_gtk.vcd");

$dumpvars(0, DownCounter4bit\_Synchronous\_Tb);

clk <= 0;

reset <= 0;

#20;

reset <= 1;

#20;

reset <= 0;

#20;

#20;

#20;

#20;

#20;

reset <= 1;

#20;

$finish;

end

initial

$monitor("clk = %b, reset = %b, q = %4b", clk, reset, q);

Endmodule

DOWN COUNTER:

// Problem - 3

module Register4 (

input wire [3:0] D,

input wire clk,

input wire reset,

output reg [3:0] Q

);

always @(posedge clk, posedge reset) begin

if(reset == 1'b1)

Q <= 4'b0000;

else

Q <= D;

end

endmodule

module subOne(

input wire [3:0] I,

output wire [3:0] O

);

assign O = I - 1;

endmodule

module DownCounter4bit(

input wire clk,

input wire reset,

output wire [3:0] q

);

wire [3:0] r\_reg, r\_next;

Register4 register(r\_next, clk, reset, r\_reg);

subOne subtractor(r\_reg, r\_next);

assign q = r\_reg;

endmodule

Test🡪

// Problem - 3

`timescale 1ns/1ns

module DownCounter4bit\_Tb;

reg clk;

reg reset;

wire [3:0] q;

DownCounter4bit counter(clk, reset, q);

always begin

#10 clk = ~clk;

end

initial begin

$dumpfile("./out/Counter4bit\_gtk.vcd");

$dumpvars(0, DownCounter4bit\_Tb);

clk <= 0;

reset <= 0;

#20;

reset <= 1;

#20;

reset <= 0;

#20;

#20;

#20;

#20;

#20;

reset <= 1;

#20;

$finish;

end

initial

$monitor("clk = %b, reset = %b, q = %4b", clk, reset, q);

Endmodule

REGISTER:  
// Problem - 1

module Register16 (

input wire [15:0] D,

input wire clk,

input wire reset,

output reg [15:0] Q

);

always @(posedge clk, posedge reset) begin

if(reset == 1'b0)

Q <= D;

else

Q <= 16'b0000\_0000\_0000\_0000;

end

endmodule

TEST🡪

// Problem - 1

`timescale 1ns/1ps

module Register16\_Tb;

reg [15:0] D;

reg clk;

reg reset;

wire [15:0] Q;

Register16 register(D, clk, reset, Q);

always begin

#10 clk = ~clk;

end

initial begin

$dumpfile("./out/Register16\_gtk.vcd");

$dumpvars(0, Register16\_Tb);

clk <= 0;

reset <= 0;

D <= 16'b1111\_0000\_0000\_1111;

#20;

reset <= 1;

#20;

D <= 16'b0000\_0000\_1111\_0000;

#20;

D <= 16'b1111\_0000\_0000\_0000;

#20;

reset <= 0;

#20;

D <= 16'b0000\_1111\_0000\_0000;

#20;

reset <= 1;

#20;

D <= 16'b0000\_0000\_1111\_0000;

#20;

$finish;

end

initial

$monitor("clk = %b, reset = %b, D = %16b", clk, reset, D);

Endmodule

7Bit Register:

// Problem - 2

module Register7Enable (

input wire [6:0] D,

input wire clk,

input wire reset,

input wire enable,

output reg [6:0] Q

);

always @(posedge clk, posedge reset) begin

if(reset == 1'b1)

Q <= 7'b0000\_000;

else if(enable == 1'b1)

Q <= D;

else

Q <= Q;

end

endmodule

TESt🡪

// Problem - 2

`timescale 1ns/1ns

module Register7Enable\_Tb;

reg [6:0] D;

reg clk;

reg reset;

reg enable;

wire [6:0] Q;

Register7Enable register(D, clk, reset, enable, Q);

always begin

#10 clk = ~clk;

end

initial begin

$dumpfile("./out/Register7Enable\_gtk.vcd");

$dumpvars(0, Register7Enable\_Tb);

clk <= 0;

reset <= 0;

D <= 16'b1111\_000;

#20;

reset <= 1;

#20;

D <= 16'b0000\_001;

#20;

D <= 16'b1111\_000;

#20;

reset <= 0;

#20;

enable <= 0;

D <= 16'b0000\_111;

#20;

reset <= 1;

#20;

D <= 16'b0000\_001;

#20;

$finish;

end

initial

$monitor("clk = %b, reset = %b, enable = %b, D = %7b", clk, reset, enable, D);

endmodule