|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Verilog | Testbench | Design | | Result |
| Half adder | module ha( a, b, cout, sum);  input a, b;  output cout, sum;  assign cout = a & b;  assign sum = a ^ b;  // xor g1 (sum, a, b);  // and g2 (cout, a, b);  endmodule | `timescale 1ns / 1ns  module ha\_tb;  reg a, b;  wire cout, sum;  ha u1 ( a, b, cout, sum );  initial begin  {a, b} = 2'b00;  #10 {a, b} = 2'b01;  #10 {a, b} = 2'b10;  #10 {a, b} = 2'b11;  #10 $stop;  end  initial $monitor($time, "ns, a=%b, b=%b, cout=%b, sum=%b", a, b, cout, sum);  endmodule |  |  | | |
| Full Adder | module fa( a, b, cin, cout, sum);  input a, b, cin;  output cout, sum;  wire m, n, p;  ha g1 (.cout(n), .sum(m), .a(a), .b(b) );  ha g2 (.cout(p), .sum(sum), .a(cin), .b(m) );  assign cout = p | n;  endmodule | `timescale 1ns / 1ns  module fa\_tb;  reg a, b, cin;  wire cout, sum;  fa u1 ( a, b, cin, cout, sum );  initial begin  {a, b, cin} = 3'b000;  #10 {a, b, cin} = 3'b001;  #10 {a, b, cin} = 3'b01\_0;  #10 {a, b, cin} = 3'b011;  #10 {a, b, cin} = 4;  #10 {a, b, cin} = 5;  #10 {a, b, cin} = 6;  #10 {a, b, cin} = 7;  #10 $stop;  end  initial $monitor($time, "ns, a=%b, b=%b, cin = %b, cout=%b, sum=%b", a, b, cin, cout, sum);  endmodule |  |  | | |
| Ripple Carry Adder (3) | module rca3( a, b, cin, cout, sum);  input [2:0] a, b;  input cin;  output [2:0] sum;  output cout;  wire [1:0] m;  fa g1 (.cout(m[0]), .sum(sum[0]), .a(a[0]), .b(b[0]), .cin(cin));  fa g2 (.cout(m[1]), .sum(sum[1]), .a(a[1]), .b(b[1]), .cin(m[0]) );  fa g3 (.cout(cout), .sum(sum[2]), .a(a[2]), .b(b[2]), .cin(m[1]) );  endmodule | `timescale 1ns / 1ns  module rca3\_tb;  reg[2:0] a, b;  reg cin;  wire [2:0] sum;  wire cout;  wire [3:0] res;  assign res = { cout, sum };  rca3 uu ( a, b, cin, cout, sum );  initial begin  a = 2; b= 4; cin = 0;  #10 a = 3; b = 3; cin = 1;  #10 a= 5; b =6; cin = 1;  #10 a= 7; b = 7; cin = 1;  #10 $stop;  end  initial $monitor($time, "ns, a=%d, b=%d, cin = %d, addition result = %d", a, b, cin, res);  endmodule |  |  | | |
| 2 to 4 Decoder | module dec2\_4(a, b,en);  output [3:0] a;  input [1:0] b;  input en;  wire [1:0] bb;  not g1(bb[1], b[1]), (bb[0], b[0]);  and g2 (a[0], en, bb[1], bb[0]);  and g3 (a[1], en, bb[1], b[0]);  and g4 (a[2], en, b[1], bb[0]);  and g5 (a[3], en, b[1], b[0]);  endmodule | module dec2\_4\_tb;  wire [3:0] a;  reg [1:0] b;  reg en;  dec2\_4 uut(a, b, en);  initial  begin  { b, en } = 3'b000;  #2 { b, en } = 3'b001;  #2 {b, en } = 3'b011;  #2 {b, en } = 3'b101;  #2 {b, en } = 3'b111;  end  initial  $monitor ($time, " output a=%b, input b=%b, input en=%b", a, b, en);  endmodule | C:\Users\Fluff\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\B738A374.tmp  https://lh6.googleusercontent.com/uke82zap4uUwl_ZXTLD9xWmKTlUvb7H5S_ZIo2ymEarbyKukKPr98wQGe13fLiIq3X9InbVozeKjKRaGoFGz2ySxzQSb8vVdorUkbEaQGKUXrSkr5Um4h9M3NcCOQrOTPMJ4kfZFNdFLBSGTvg |
| Muxb | module muxb( d0, d1, s, y );  input [3:0] d0, d1;  input s;  output [3:0] y;  reg [3:0] y;  always@(d0 or d1 or s)  begin  if ( s == 1 ) y = d1;  else y = d0;  end  endmodule | module muxb\_tb;  reg [1:0] d;  reg s;  wire y;  mux ul(.d(d), .s(s), .y(y));  intial  begin  d=2'b10; s=0;  #10; d=2'b01; s=0;  #10;d=2'b10; s=1;  #10;d=2'b01; s=1;  #20 $stop;  end  endmodule |  |

`timescale 1ns/1ns

module dff\_e1\_tb;

reg clk, d;

wire q;

dff\_e1 uut( clk, d, q);

always #10 clk = ~ clk;

initial

begin

clk = 0;

d = 1;

#5 d = 0;

#20 d = 1;

#20 d = 0;

#15 $stop; end

endmodule

/////////////////////////////

`timescale 1ns/1ns

module dff\_e2\_tb;

reg clk, rst, d;

wire q;

dff\_e1 uut( clk, rst, d, q);

always #10 clk = ~ clk;

initial

begin

clk = 0;

rst = 1;

d = 1;

#6 rst = 0;

#5 d = 0;

#20 d = 1;

#20 d = 0;

#15 $stop; end

endmodule

/////////////////////////////

`timescale 1ns/1ns

module dff\_e3\_tb;

Professor Pang

CPE166 Advanced Logic Design Handout

reg clk, rst;

reg [3:0] d;

wire [3:0] q;

dff\_e1 uut( clk, rst, d, q);

always #10 clk = ~ clk;

initial

begin

clk = 0;

rst = 1;

d = 4'b1100;

#6 rst = 0;

#5 d = 8;

#20 d = 10;

#20 d = 4;

#15 $stop; end

endmodule

/////////////////////////////

module dffb\_tb;

reg clk, clr, load, sft;

reg [3:0] db;

wire [3:0] qb;

Professor Pang

CPE166 Advanced Logic Design Handout

dffb uut (clk, clr, load, sft);

initial clk = 0;

always #10 clk = ~ clk;

initial

begin

clr = 1; load = 0; sft = 0; db = 4'b1011;

#22 clr=0;

load = 1; sft = 0;

#20 load = 0; sft = 1;

#60 $stop; end

Endmodule.20.

……………………………………………

3333333

30.2

**1. General D Flip-Flop**

module dff\_e1 ( clk, d, q);

input clk, d;

output q;

reg q;

always @ ( posedge clk )

begin

q <= d;

end

endmodule

/////////////////////////////////////

**2. Rising Edge Triggered D Flip-Flop with Asynchronous High Active Reset**

module dff\_e2 ( clk, rst, d, q);

input clk, d, rst;

output q;

reg q;

always @ ( posedge clk or posedge rst)

begin

if(rst )

q <= 0;

else

q <= d;

Professor Pang

CPE166 Advanced Logic Design Handout

end

endmodule

/////////////////////////////////////

**3. 4-bit Registers with High Active Asynchronous Reset**

module dff\_e3 ( clk, rst, d, q);

input clk, rst;

input [3:0] d;

output [3:0] q;

reg [3:0] q;

always @ ( posedge clk or posedge rst)

begin

if(rst )

q <= 0;

else

q <= d;

end

endmodule

////////////////////////////////////

**4. More Features for Registers**

module dffb (clk, clr, load, sft, db, qb);

input clk, clr, load, sft;

input [3:0] db;

output [3:0] qb;

reg [3:0] qb;

always@(posedge clr or posedge clk)

begin

if(clr) qb <= 0;

else if (load)

qb <= db;

else if (sft)

qb <= { 1'b0, qb[3:1] };

// qb[3] <= 1'b0;

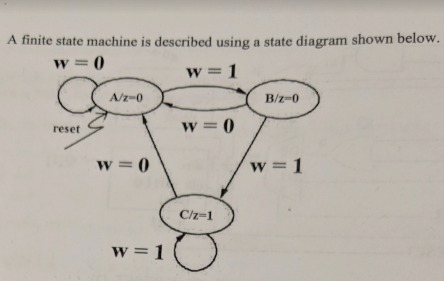
// qb[2] <= qb[3];

// qb[1] <= qb[2];

// qb[0] <= qb[1];

end

endmodule

 **Moore FSM**

**Use Verilog to design a finite state machine to recognize the sequence 0110.**

module fsm\_detector(reset, clk, a, y);

input reset, a, clk;

output y;

reg y;

parameter s\_idle = 3'b000, s1=3'b001, s2=3'b010, s3=3'b011, s4=3'b100;

reg [2:0] cs, ns;

always@(posedge clk or posedge reset)

begin

if(reset) cs <= s\_idle;

else cs <= ns;

end

always@(cs or a)

begin

case(cs)

s\_idle: if(a) ns = s\_idle;

else ns = s1;

s1: if(a) ns = s2;

else ns = s1;

s2: if(a) ns = s3;

else ns = s1;

s3: if(~a) ns = s4;

else ns = s\_idle;

s4: if(a) ns = s2;

else ns = s1;

default: ns = s\_idle;

endcase

end

Hierarchical Design

module my\_block (in1, in2, dout);

input in1, in2;

output dout;

. . .

. . .

endmodule

module top (DI\_1, DI\_2, DI\_3,DI\_4,

DOUT1, DOUT2);

input DI\_1, DI\_2, DI\_3, DI\_4;

output DOUT1, DOUT2;

my\_block inst1 (

.in1(DI\_1),

.in2(DI\_2),

.dout(DOUT)

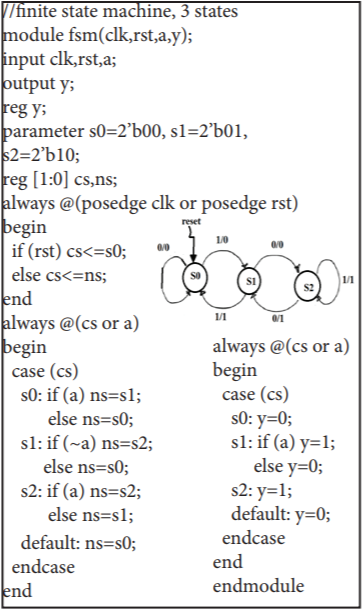
);

my\_block inst2(DI\_3,DI\_4, DOUT);

. . .

. . .

endmodule



always@(cs or a)

begin

case(cs)

s\_idle: y = 0;

s1: y = 0;

s2: y = 0;

s3: if(~a) y = 1;

else y = 0;

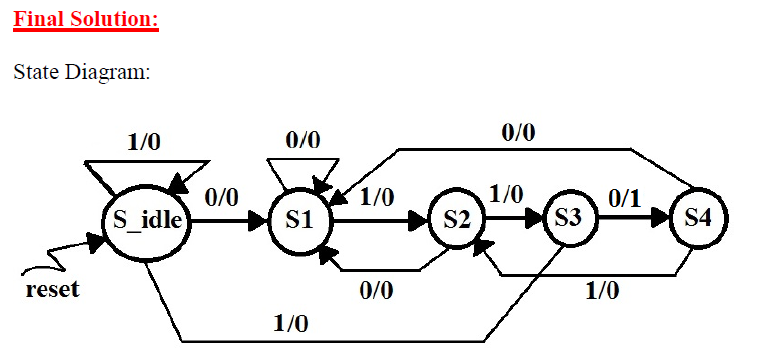
s4: y = 0;

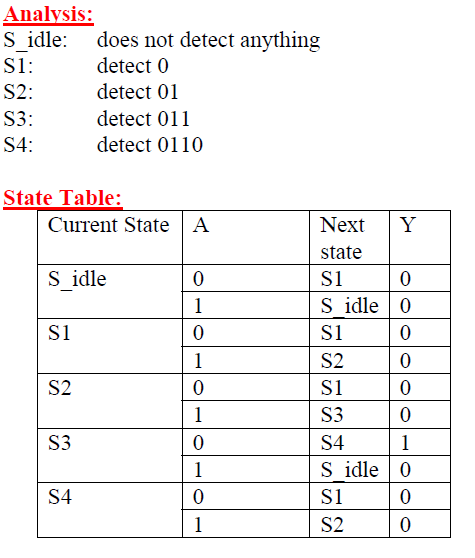
default: y = 0;

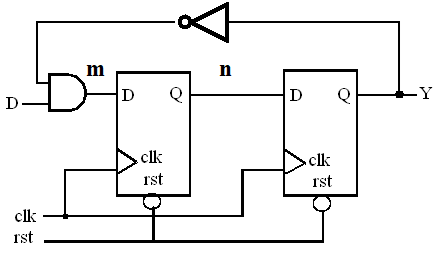
endcase

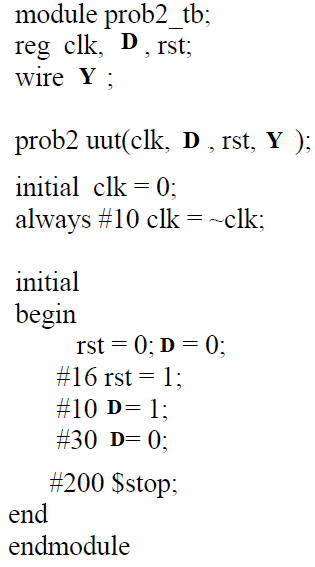
end

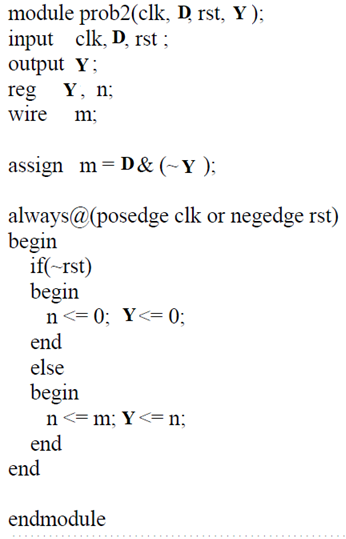
endmodule





****





module mealy\_fsm(reset, a, clk, y);

input    reset, a, clk;

output   y;

reg      y;

// Binary State Enconding

parameter  s0 = 2'b00, s1 = 2'b01, s2 = 2'b10;

reg [1:0]    cs, ns;

// sequential state register block

always@(posedge reset or posedge clk)

begin

    if( reset )  cs <= s0;

    else         cs <= ns;

end

// combinational next state block

always@(cs or a)

begin

    case(cs)

    s0:   if(a) ns = s1;

          else  ns = s0;

    s1:   if(a) ns = s1;

          else  ns = s0;

    s2:   if(a) ns = s2;

          else  ns = s3;

    s3:   if(a) ns = s4;

          else  ns = s0;

    s4:   if(a) ns = s1;

          else  ns = s3;

    default:  ns = s0;

    endcase

end

//combinational output block

always@(cs or a)

begin

    case(cs)

    s0:   if(a) y = 0;

          else  y = 1;

    s1:   y = 0;

    s2:   if(a) y = 0;

          else  y = 1;

    s3:   if(a) y = 1;

          else  y = 0;

    s4:   if(a) y = 1;

          else  y = 0;

    default:  y = 0;

    endcase

end

module aft16(clk, din, reset, dout);

input clk, din, reset;

output [6:0]dout;

reg [6:0]dout;

always @(posedge reset or negedge clk)

begin

if(reset)

dout<=0;

else

dout<={din, Q[15:1]};

end

always @(posedge rest)

begin

d<=7’b00000000

end

endmodule

module circuit\_tb

reg A, B, CLK;

wire [1:0]Y;

circuit uut(A, B, CLK, Y);

initial clk=1;

always#5

clk=~clk

initial

begin

A=1; B=0;

#10;

A=0; B=1;

#10

A=1; B=1;

#10

A=0; B=0;

#200 $stop;end

endmodule

module priority (sel, code);

input [7:**0**] sel;

output [2:0] code;

reg [2:0] code;

always @(sel)

begin

if (sel[0]) code = 3'b000;

else if (sel[1]) code = 3'b001;

else if (sel[2]) code = 3'b010;

else if (sel[3]) code = 3'b011;

else if (sel[4]) code = 3'b100;

else if (sel[5]) code = 3'b101;

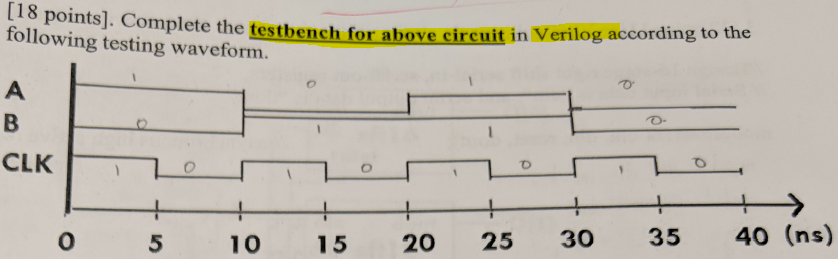
else if (sel[6]) code = 3'b110;

else if (sel[7]) code = 3'b111;

else code = 3'bxxx;

end

endmodule



***2 to 1 Multiplexer***

module mux2to1(d0, d1, s, y);

input d0, d1, s;

output y;

wire m,n;

assign m=(~s) &d 0;

assign n = s & d1;

assign y= m | n;

endmodule

`timescale 1ns / 1ns

module mux2to1\_tb;

reg       d1, d0, s;

wire     y;

mux2to1   u1 (d1, d0, s, y);

initial begin

          d1 = 0;  d0 = 0; s = 0;

   #10  d1 = 0;  d0 = 0; s = 1;

   #10  d1 = 0;  d0 = 1; s = 0;

   #10  d1 = 0;  d0 = 1; s = 1;

   #10  d1 = 1;  d0 = 0; s = 0;

   #10  d1 = 1;  d0 = 0; s = 1;

   #10  d1 = 1;  d0 = 1; s = 0;

   #10  d1 = 1;  d0 = 1; s = 1;

   #10 $stop;

end

initial $monitor($time, "ns, d1=%b, d0=%b, s = %b,  y=%b", d1, d0, s, y);

endmodule

module circuit(A, B, C, D, E, F, G, SEL, CLK, REST, Q);

input A, B, C, D, E, F, G, SEL, CLK, REST;

output [1:0] Q;

wire m, n, p, w;

assign m=A&B;

assign p = m^C^D ;

or g2(E,F);

mux2to1 g3(n, G, SEL, w);

SFT16 g0(clk, p, RESET, Q[0]);

Sft16 g1(clk, w, RESET, Q[1]);

endmodule

module circuit(A, B, CLK, Y);

input A, B, CLK;

output [1:0]Y;

reg [1:0]Y;

wire N,M;

assign N=A^ ~B;

assign M=~(Y[1] \* Y[0] \* B);

always @(negedge clk)

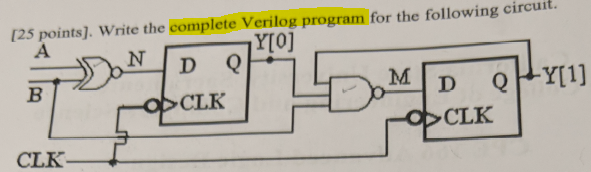
begin

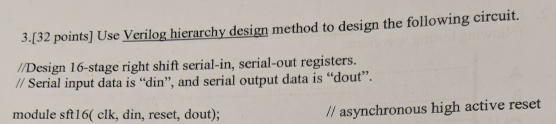
Y[0]<=N;

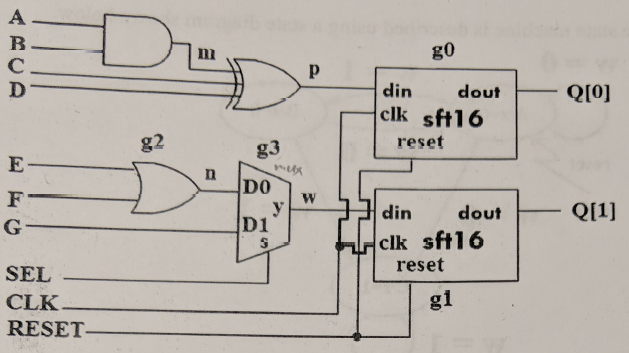
Y[1]=M;

End

endmodule







module moorefsm (clk, reset, w, z);

input clk, reset, w;

output z;

reg z;

parameter A=0; B=1, C=0;

reg cd, ns;

always @(posedge clk or posedge clk)

begin

if(reset) cs <=A;

else cs <=ns;

end

always@(cs or w)

begin

case (cs);

A: if(w) ns=B;

Else ns=A;

B: if(~w) ns =A;

Else ns=c;

C: if(~w) ns=A;

Else ns = c;

Default:

Ns=A;

Endcase

end

endmodule

