Verilog

Full adder

module fulladder (a, b, cin, sum, cout);

input a, b, cin;

output sum, cout;

assign sum = a^b^cin;

assign cout = (a&b)|(b&cin)|(a&cin);

endmodule

**TESTBENCH**

module fulladder\_tb;

reg a;

reg b;

reg c;

wire sum;

wire cout;

fa uut (  .a(a),   .b(b),.c(c),.sum(sum),.cout(cout)  );

initial begin

#10 a=1’b0;b=1’b0;c=1’b0;

#10 a=1’b0;b=1’b0;c=1’b1;

#10 a=1’b0;b=1’b1;c=1’b0;

#10 a=1’b0;b=1’b1;c=1’b1;

#10 a=1’b1;b=1’b0;c=1’b0;

#10 a=1’b1;b=1’b0;c=1’b1;

#10 a=1’b1;b=1’b1;c=1’b0;

#10 a=1’b1;b=1’b1;c=1’b1;

#10$stop;

end

endmodule

**ENCODER**

module encoder (I,D);

input [7:0] I;

output [2:0] D;

reg [2:0] D;

always @ (I)

begin

if (I[7]==1'B1) D=3'B111;

else if (I[6]==1'B1) D=3'B110;

else if (I[5]==1'B1) D=3'B101;

else if (I[4]==1'B1) D=3'B100;

else if (I[3]==1'B1) D=3'B011;

else if (I[2]==1'B1) D=3'B010;

else if (I[1]==1'B1) D=3'B001;

else if (I[0]==1'B1) D=3'B000;

else D=3'BZZZ;

end

endmodule

**Encoder( without priority)TestBench**

module encoder\_tb;

reg [7:0] I;

wire [2:0] D;

encoder uut(

.D(D),

.I(I)

);

initial begin

// Initialize Inputs

#10 I = 8'b10000000;

#10 I = 8'b01000000;

#10 I = 8'b00100000;

#10 I = 8'b00010000;

#10 I= 8'b00001000;

#10 I= 8'b00000100;

#10 I = 8'b00000010;

#10 I = 8'b00000001;

#10;

end

initial begin

$monitor("time=",$time, "I=%b : D=%b ",I,D);

end

endmodule

**with priority(Replace this in encoder testbench)**

I = 8'b00000001;

I= 8'b0000001x;

I = 8'b000001xx;

I = 8'b00001xxx;

I= 8'b0001xxxx;

I = 8'b001xxxxx;

I = 8'b01xxxxxx;

I = 8'b1xxxxxxx;

**Verilog**

**Decoder**

module decoder(I,D);

input [1:0] I;

output [3:0] D;

reg [3:0] D;

always @(I)

begin

if (I==2'B00) D=4'B0001;

else if (I==2'B01) D=4'B0010;

else if (I==2'B10) D=4'B0100;

else if (I==2'B11) D=4'B1000;

else D=4'BZZZZ;

end

endmodule

**Decoder Testbench**

module decoder\_tb;

reg [1:0] I;

wire [3:0] D;

decoder uut(

.D(D),

.I(I)

);

initial begin

// Initialize Inputs

I=2'b00; #20;

I = 2'b01; #20;

I = 2'b10; #20;

I = 2'b11; #20;

end

endmodule

**ALU**

module alu(

input [7:0] A,B, // ALU 8-bit Inputs

input [3:0] ALU\_Sel,// ALU Selection

output [7:0] ALU\_Out, // ALU 8-bit Output

output CarryOut // Carry Out Flag

);

reg [7:0] ALU\_Result;

wire [8:0] tmp;

assign ALU\_Out = ALU\_Result; // ALU out

assign tmp = {1'b0,A} + {1'b0,B};

assign CarryOut = tmp[8]; // Carryout flag

always @(\*)

begin

case(ALU\_Sel)

4'b0000: // Addition

ALU\_Result = A + B ;

4'b0001: // Subtraction

ALU\_Result = A - B ;

4'b0010: // Multiplication

ALU\_Result = A \* B;

4'b0011: // Division

ALU\_Result = A/B;

4'b0100: // Logical shift left

ALU\_Result = A<<1;

4'b0101: // Logical shift right

ALU\_Result = A>>1;

4'b0110: // Rotate left

ALU\_Result = {A[6:0],A[7]};

4'b0111: // Rotate right

ALU\_Result = {A[0],A[7:1]};

4'b1000: // Logical and

ALU\_Result = A & B;

4'b1001: // Logical or

ALU\_Result = A | B;

4'b1010: // Logical xor

ALU\_Result = A ^ B;

4'b1011: // Logical nor

ALU\_Result = ~(A | B);

4'b1100: // Logical nand

ALU\_Result = ~(A & B);

4'b1101: // Logical xnor

ALU\_Result = ~(A ^ B);

4'b1110: // Greater comparison

ALU\_Result = (A>B)?8'd1:8'd0 ;

4'b1111: // Equal comparison

ALU\_Result = (A==B)?8'd1:8'd0 ;

default: ALU\_Result = A + B ;

endcase

end

endmodule

**TESTBENCH**

`timescale 1ns / 1ps

module tb\_alu;

//Inputs

reg[7:0] A,B;

reg[3:0] ALU\_Sel;

//Outputs

wire[7:0] ALU\_Out;

wire CarryOut;

// Verilog code for ALU

integer i;

alu test\_unit(

A,B, // ALU 8-bit Inputs

ALU\_Sel,// ALU Selection

ALU\_Out, // ALU 8-bit Output

CarryOut // Carry Out Flag

);

initial begin

// hold reset state for 100 ns.

A = 8'h0A;

B = 4'h0B;

ALU\_Sel = 4'h0;

for (i=0;i<=15;i=i+1)

begin

ALU\_Sel = ALU\_Sel + 8'h01;

#10;

end

end

endmodule

**Synchronous Counter**

module counter\_behav ( count,reset,clk);

input wire reset, clk;

output reg [3:0] count;

always @(posedge clk) if (reset)

count <= 4'b0000; else

count <= count + 4'b0001;

endmodule

module mycounter\_t ;

wire [3:0] count;

reg reset,clk; initial

clk = 1'b0; always

#5 clk = ~clk;

counter\_behav m1 ( count,reset,clk);

initial

begin

reset = 1'b0 ;

#15 reset =1'b1;

#30 reset =1'b0;

#300 $finish; end

initial

$monitor ($time, "Output count = %d ",count );

Endmodule

**Asynchronous Counter**

module ripple\_counter (clock, toggle, reset, count); input clock, toggle, reset;

output [3:0] count;

reg [3:0] count;

wire c0, c1, c2;

assign c0 = count[0],

c1 = count[1],

c2 = count[2];

always @ (posedge reset or posedge clock) if (reset == 1'b1) count[0] <= 1'b0;

else if (toggle == 1'b1) count[0] <= ~count[0]; always @ (posedge reset or negedge c0)

if (reset == 1'b1) count[1] <= 1'b0;

else if (toggle == 1'b1) count[1] <= ~count[1]; always @ (posedge reset or negedge c1)

if (reset == 1'b1) count[2] <= 1'b0;

else if (toggle == 1'b1) count[2] <= ~count[2]; always @ (posedge reset or negedge c2)

if (reset == 1'b1) count[3] <= 1'b0;

else if (toggle == 1'b1) count[3] <= ~count[3];

endmodule

**TESTBENCH**

module ripple\_counter\_t ; reg clock,toggle,reset; wire [3:0] count ;

ripple\_counter r1 (clock,toggle,reset,count); initial

clock = 1'b0; always

#5 clock = ~clock; initial

begin

reset = 1'b0;toggle = 1'b0;

#10 reset = 1'b1; toggle = 1'b1;

#10 reset = 1'b0;

#190 reset = 1'b1;

#20 reset = 1'b0;

#100 reset = 1'b1;

#40 reset = 1'b0;

#250 $finish; end

initial

$monitor ($time, " output q = %d", count);

Endmodule

**SRFlipflop**

module SR\_ff(q,qbar,s,r,clk); output q,qbar;

input clk,s,r; reg tq;

always @(posedge clk or tq) begin

if (s == 1'b0 && r == 1'b0) tq <= tq;

else if (s == 1'b0 && r == 1'b1) tq <= 1'b0;

else if (s == 1'b1 && r == 1'b0) tq <= 1'b1;

else if (s == 1'b1 && r == 1'b1) tq <= 1'bx;

end

assign q = tq; assign qbar = ~tq;

endmodule

**TESTBENCH**

module SR\_ff\_test;

reg clk,s,r;

wire q,qbar; wire s1,r1,clk1;

SR\_ff sr1(q,qbar,s,r,clk); assign s1=s;

assign r1=r; assign clk1=clk; initial

clk = 1'b0; always

#10 clk = ~clk; initial

begin

s = 1'b0; r = 1'b0;

#30 s = 1'b1;

#29 s = 1'b0;

#1 r = 1'b1;

#30 s = 1'b1;

#30 r = 1'b0;

#20 s = 1'b0;

#19 s = 1'b1;

#200 s = 1'b1; r = 1'b1;

#50 s = 1'b0; r = 1'b0;

#50 s = 1'b1; r = 1'b0;

#10 ;

end always

#5 $display($time," clk=%b s=%b r=%b ",clk,s,r); initial

#500 $finish;

endmodule

**TFlipflop**

module t\_ff(q,qbar,clk,tin,rst); output q,qbar;

input clk,tin,rst; reg tq;

always @(posedge clk or negedge rst) begin

if(!rst)

tq <= 1'b0; else

begin if (tin)

tq <= ~tq; end

end

assign q = tq; assign qbar = ~q;

endmodule

TESTBENCH

module t\_ff\_test; reg clk,tin,rst; wire q,qbar;

t\_ff t1(q,qbar,clk,tin,rst); initial

clk = 1'b0; always

#10 clk = ~clk; initial

begin

rst = 1'b0; tin = 1'b0;

#30 rst = 1'b1;

#10 tin = 1'b1;

#205 tin = 1'b0;

#300 tin = 1'b1;

#175 tin = 1'b0;

#280 rst = 1'b0;

#20 rst = 1'b1;

#280 tin = 1'b1;

#10 ;

end initial

#2000 $finish;

endmodule

**Serial adder**

module serialadder(clk,rst,pload,adata,bdata,enable,pout);

input clk,rst,pload,enable;

input [7:0] adata, bdata;

output [7:0] pout;

wire shiftrega\_lsb, shiftregb\_lsb;

reg [7:0] shiftrega, shiftregb, shiftregc;

wire sum,cout, sum\_del, cout\_del;

reg holdc;

// instantiated the full adder ( combo circuit)

full\_adder\_1bit bit\_adder\_inst(shiftrega[0],shiftregb[0],holdc,sum,cout);

assign pout=shiftregc;

assign sum\_del = sum & enable;

assign cout\_del = cout & enable;

// sequential block -- shift registers and shifting logic

always@(posedge clk or rst )

begin

if (rst) begin

shiftrega<=8'd0;

shiftregb<=8'd0;

shiftregc<=8'd0;

end else if(pload ) begin

shiftrega<=adata;

shiftregb<=bdata;

shiftregc<=8'b0;

end else if(enable) begin

shiftrega <=shiftrega>>1;

shiftrega[7]<=shiftrega\_lsb;

shiftregb<=shiftregb>>1;

shiftregb[7] <=shiftregb\_lsb;

shiftregc <=shiftregc>>1;

shiftregc[7] <=sum\_del;

end

end

// combo logic

assign shiftrega\_lsb=shiftrega[0];

assign shiftregb\_lsb=shiftregb[0];

always@(posedge clk or rst)

begin

if(rst) begin

holdc <=1'b0;

end else if(enable)

holdc<=cout\_del;

else

holdc<=1'b0;

end

endmodule

**TESTBENCH**

module serialadder\_tb;

reg clk,rst,pload,enable;

reg [7:0] adata,bdata;

wire [7:0]pout;

// DUT serial adder instance

serialadder DUT\_serial\_adder(clk,rst,pload,adata,bdata,enable,pout);

// clock generation 100 Mhz frequency

always

#5 clk=~clk;

initial

begin

clk=1'b0;

rst=1'b1;

pload=1'b0;

enable = 1'b0;

#10

rst=1'b0;

#10

pload=1'b1; enable=1'b0;

$display ($time, " On Reset : adata=%0h, bdata=%0h, pout=%0h", adata, bdata, pout);

adata=8'd1;bdata=8'd2;

$display ($time, " First Data - Data Inputs loaded: adata=%0h, bdata=%0h, pout=%0h", adata, bdata, pout);

#10

pload=1'b0;enable=1'b1;

#200

$display ($time, " First Data Serial addition completed: adata=%0h, bdata=%0h, pout=%0h", adata, bdata, pout);

rst=1'b0;

#10

pload=1'b1; enable=1'b0;

adata=8'd2;bdata=8'd3;

#10

$display ($time, " Second Data - Data Inputs loaded: adata=%0h, bdata=%0h, pout=%0h", adata, bdata, pout);

pload=1'b0;enable=1'b1;

#200

$display ($time, " Second Data Serial addition completed: adata=%0h, bdata=%0h, pout=%0h", adata, bdata, pout);

$finish;

end

// dump waveforms in cadence format

initial begin

`ifdef CAD\_DUMP\_ON

$recordfile("adder.trn");

$recordvars("depth=0");

`endif

`ifdef VCD\_DUMP\_ON

$dumpfile("adder.vcd");

$dumpvars("depth=0");

`endif

end

endmodule

module full\_adder\_1bit(a,b,ci,s,co);

input a,b,ci;

output s,co;

assign s = (a^b^ci);

assign co = ((a&b) | (b&ci) | (ci & a));

endmodule

**Moore Sequential Machine**

**0**

**1**

**1**

**one1 [0]**

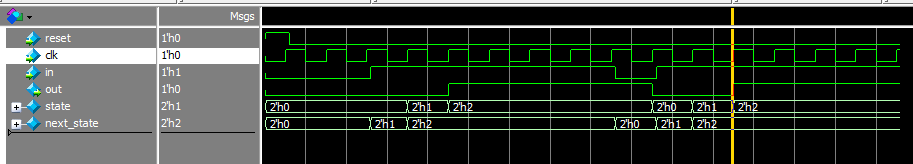
**two1s [1]**

**zero [0]**

**0**

**0**

**1**



**module** state\_machine\_moore**(**clk**,** reset**,** in**,** out**);**

**parameter** zero**=**0**,** one1**=**1**,** two1s**=**2**;**

**output** out**; input** clk**,** reset**,** in**;**

**reg** out**; reg** **[**1**:**0**]** state**,** next\_state**;**

// Implement the state register

**always** **@(posedge** clk **or** **posedge** reset**)** **begin**

**if** **(**reset**)**

state **<=** zero**;**

**else**

state **<=** next\_state**;**

**end**

**always** **@(**state **or** in**)** **begin**

**case** **(**state**)**

zero**:** **begin** //last input was a zero out = 0;

**if** **(**in**)**

next\_state**=**one1**;**

**else**

next\_state**=**zero**;**

**end**

one1**:** **begin** //we've seen one 1 out = 0;

**if** **(**in**)**

next\_state**=**two1s**;**

**else**

next\_state**=**zero**;**

**end**

two1s**:** **begin** //we've seen at least 2 ones out = 1;

**if** **(**in**)**

next\_state**=**two1s**;**

**else**

next\_state**=**zero**;**

**end**

**default:** //in case we reach a bad state out = 0;

next\_state**=**zero**;**

**endcase**

**end**

// output logic

**always** **@(**state**)** **begin**

**case** **(**state**)**

zero**:** out **<=** 0**;**

one1**:** out **<=** 0**;**

two1s**:** out **<=** 1**;**

**default** **:** out **<=** 0**;**

**endcase**

**end**

**endmodule**

**Test Bench**

timescale 1ns**/**1ps

`include "state\_machine\_moore.v"

**module** state\_machine\_moore\_tb **;**

**reg** clk**,** reset**,** in**;**

**wire** out**;**

// instantiate state machine

state\_machine\_moore DUT **(**clk**,** reset**,** in**,** out**);**

**initial**

**forever** **#**5 clk **=** **~**clk**;**

**initial** **begin**

reset **=** 1'b1**;**

clk **=** 1'b0**;**

in **=** 0**;**

**#**6**;**

reset **=** 1'b0**;**

**for** **(integer** i**=**0**;** i**<** 10**;** i**=**i**+**1**)** **begin**

**@(negedge** clk**);** #1;

in **=** $random**;**

**if** **(**out **==** 1'b1**)**

$display **(**"PASS : Sequence 11 detected \n"**);**

**end**

**#**50**;**

$finish**;**

**end**

**endmodule**

**0/0 1/0**

**0/0**

**1/1**

**one1**

**zero**

**Mealy state Machine**

**module** state\_machine\_mealy **(**clk**,** reset**,** in**,** out**);**

**input** clk**,** reset**,** in**;**

**output** out**;**

**reg** out**,** state**,** next\_state**;**

**parameter** zero**=**0**,** one**=**1**;**

//Implement the state register

**always@(posedge** clk, **posedge** reset**) begin**

**if** **(**reset**)**

state **<=** zero**;**

**else**

state **<=** next\_state**;**

**End**

**always** **@(**in **or** state**)**

**case** **(**state**)**

zero**:** **begin**

// last input was a zero

out**=**0**;**

**if** **(**in**)**

next\_state **=** one**;**

**else**

next\_state **=** zero**;**

**end**

one**:** **begin** //seen one

**if** **(**in**)** **begin**

next\_state **=** one**;**

out**=**1**;**

**end** **else** **begin**

next\_state **=** zero**;**

out**=**0**;**

**end**

**end**

**endcase**

**endmodule**

**Test bench**

`timescale 1ns**/**1ps

`include "state\_machine\_mealy.v"

**module** state\_machine\_mealy\_tb **;**

**reg** clk**,** reset**,** in**;**

**wire** out**;**

// instantiate state machine

state\_machine\_mealy DUT **(**clk**,** reset**,** in**,** out**);**

**initial**

**forever** **#**5 clk **=** **~**clk**;**

**initial** **begin**

reset **=** 1'b1**;**

clk **=** 1'b0**;**

in **=** 0**;**

**#**6**;**

reset **=** 1'b0**;**

**for** **(integer** i**=**0**;** i**<** 10**;** i**=**i**+**1**)** **begin**

**@(negedge** clk**);** #1;

in **=** $random**;**

**if** **(**out **==** 1'b1**)**

$display **(**"PASS : Sequence 11 detected i=%d\n“, i**);**

**end**

**#**50**;**

$finish**;**

**end**

