**module modul\_comparator**(a, b, mic, egal, mare);

input a, b;

output mic, egal, mare;

wire w1, w2;

not(w1, a);

not(w2, b);

and(mic, w1, b);

xnor(egal, a, b);

and(mare, a, w2);

endmodule

module test\_comparator();

reg a, b;

wire mic, egal, mare;

initial begin

a=0;b=0;

#10 a=0;b=1;

#10 a=1;b=0;

#10 a=1;b=1;

end

modul\_comparator COMP (a,b,mic,egal,mare);

endmodule

**module comparator\_4biti\_comp**(a, b, mic, egal, mare);

input [3:0]a, b;

output reg mic, egal, mare;

always @(a or b)

if(a<b) begin mic = 1; egal = 0; mare = 0; end

else if (a == b)

begin mic = 0; egal = 1; mare = 0; end

else begin mic = 0; egal = 0; mare = 1; end

endmodule

module test\_comparator\_4biti\_comp();

reg [3:0]a, b;

wire mic, egal, mare;

initial begin

a=0; b=2;

#10 a=10;b=13;

#10 a=7; b=7;

#10 a=12;b=12;

#10 a=15;b=2;

#10 a=8; b=6;

end

comparator\_4biti\_comp COMPARATOR\_4biti\_comp(a,b,mic,egal,mare);

endmodule

**module mux\_struct**(out, a, b, sel); //cu primitive

input a, b, sel;

output out;

wire w1,w2,w3;

not(w1, sel);

and(w2, a, w1);

and(w3, b, sel);

or(out, w2, w3);

endmodule

------------------------------------------------------------

**module mux\_comp**(out, a, b, sel); //comportamental

input a, b, sel;

output out;

assign out = (a & (~(sel))) | (b & sel);

endmodule

------------------------------------------------------------

**module mux\_cond**(out, a, b, sel); //conditional

input a, b, sel;

output out;

assign out = (sel == 0)? a: b;

endmodule

------------------------------------------------------------

**module modul\_multiplexor**(i0,i1,i2,i3,a0,a1,e,y);

input i0,i1,i2,i3,a0,a1,e;

output y;

wire wa0, wa1, we, y0,y1,y2,y3;

not(wa0, a0);

not(wa1, a1);

not(we, e);

and(y0, i0, wa0, wa1, we);

and(y1, i1, a0, wa1, we);

and(y2, i2, wa0, a1, we);

and(y3, i3, a0, a1, we);

or(y, y0,y1,y2,y3);

endmodule

module test\_multiplexor();

reg i0,i1,i2,i3,a0,a1,e;

wire y;

initial begin

a1=0; a0=0; e=0; i3=0; i2=0; i1=0; i0=1;

#10 a1=0; a0=1; e=0; i3=0; i2=0; i1=1; i0=0;

#10 a1=1; a0=0; e=0; i3=0; i2=1; i1=0; i0=0;

#10 a1=1; a0=1; e=0; i3=1; i2=0; i1=0; i0=0;

end

modul\_multiplexor mux(i0,i1,i2,i3,a0,a1,e,y);

endmodule

------------------------------------------------------------

**module mux\_4biti\_comp**(a, b, c, d, sel, out);

input [3:0]a, b, c, d;

input [1:0]sel;

output reg [3:0]out;

always @(a or b or c or d or sel)

if (sel==0) out <= a;

else if (sel==1) out <= b;

else if (sel==2) out <= c;

else out <= d;

endmodule

//testare comportamental cu porti logice

module test\_mux\_4biti\_comp();

reg [3:0] a, b, c, d;

reg [2:0] sel;

wire [3:0] out;

initial begin

a<=0; b<=1; c<=2; d<=4; sel <= 0;

#5 a<=0; b<=1; c<=2; d<=4; sel <= 1;

#5 a<=0; b<=1; c<=2; d<=4; sel <= 2;

#5 a<=0; b<=1; c<=2; d<=4; sel <= 3;

end

mux\_4biti\_comp MUX\_4biti (a, b, c, d, sel, out);

endmodule

**module demultiplexor**(d, e, a, y);

input d, e;

input [2:0]a;

output reg [7:0]y;

integer i;

always @ ( \* )

for(i=0; i<8; i=i+1)

if(a==i && e==0) y[i]=d;

else y[i]=0;

endmodule

module test\_demultiplexor();

reg d, e;

reg [2:0]a;

wire[7:0]y;

initial begin

d=1; e=0; a=0;

#5 a=1; #5 a=2; #5 a=3; #5 a=4; #5 a=5; #5 a=6;

#5 a=7; #5 e=1;

end

demultiplexor gate\_demultiplexor(d, e, a, y);

endmodule

**module codificator8**(in, c);

input [7:0] in;

output reg[2:0]c;

integer i;

always @ ( \* )

for(i=0; i<8; i = i+1)

if(in[i] == 1) c = i;

endmodule

module test\_codificator();

reg [7:0] in;

wire[2:0]c;

initial begin

in = 8'b10000000;

#5 in = 8'b01000000;

#5 in = 8'b00100000;

#5 in = 8'b00010000;

#5 in = 8'b00001000;

#5 in = 8'b00000100;

#5 in = 8'b00000010;

#5 in = 8'b00000001;

end

codificator8 gate\_codificator(in, c);

endmodule

**module modul\_decodificator**(a, b,y0,y1,y2,y3);

input a, b;

output y0,y1,y2,y3;

wire w1, w2;

not(w1, a);

not(w2, b);

and(y0, w1, w2);

and(y1, w1, b);

and(y2, w2, a);

and(y3, a,b);

endmodule

module test\_decodificator();

reg a, b;

wire y0,y1,y2,y3;

initial begin

a=0;b=0;

#10 a=0;b=1;

#10 a=1;b=0;

#10 a=1;b=1;

end

modul\_decodificator DECOD (a,b,y0,y1,y2,y3);

endmodule

-----------------------------------------------------------------**module dec\_3biti**(in, ac, y );

input [2:0]in;

input ac;

output reg [7:0]y;

always @(ac or in)

if (ac == 0) y = 0;

else

case (in)

0: y[0] = 1;

1: y[1] = 1

2: y[2] = 1;

3: y[3] = 1;

4: y[4] = 1;

5: y[5] = 1;

6: y[6] = 1;

7: y[7] = 1;

default: y = 1;

endcase

endmodule

module test\_dec\_3biti();

reg [2:0]in;

reg ac;

wire [7:0]y;

initial begin

in=7; ac=0; // 0 tot

#10 in=0; ac=1; // 1000 0000

#5 ac=0;

#10 in=4; ac=1; // 0000 1000

#5 ac=0;

#10 in=7; ac=1; // 0000 0001

end

dec\_3biti DEC\_3Biti(in, ac, y );

endmodule

**module sum\_comp\_concat**(cout, out, a, b, cin);

input a, b, cin;

output cout, out;

assign {cout, out} = a + b + cin;

endmodule

module test\_sum\_comp\_concat();

reg a, b, cin;

wire cout, out;

initial begin

a=0;b=0;cin=0;

#5 a=0;b=0;cin=1;

#5 a=0;b=1;cin=0;

#5 a=0;b=1;cin=1;

#5 a=1;b=0;cin=0;

#5 a=1;b=0;cin=1;

#5 a=1;b=1;cin=0;

#5 a=1;b=1;cin=1;

end

sum\_comp\_concat sum\_concat(cout, out, a, b, cin);

endmodule

module test\_sum\_4biti();

reg [3:0]a, b;

reg cin;

wire [3:0]out;

wire cout;

initial begin

a=10;b=5;cin=0;

#5 a=10;b=5;cin=1;

end

sum\_comp\_concat s0(w1, out[0], a[0], b[0], cin);

sum\_comp\_concat s1(w2, out[1], a[1], b[1], w1);

sum\_comp\_concat s2(w3, out[2], a[2], b[2], w2);

sum\_comp\_concat s3(cout, out[3], a[3], b[3], w3);

endmodule

-----------------------------------------------------------------**module sumator\_4biti**(a, b, cin, out, cout);

input [3:0]a, b;

input cin;

output reg cout;

output reg [3:0]out;

always @(a or b or cin) begin

if(a + b + cin <=15) begin

out = a + b + cin; cout = 0;

end

else begin

out = a + b + cin; cout = 1;

end

end

endmodule

module test\_sumator\_4biti();

reg [3:0]a, b;

reg cin;

wire [3:0]out;

wire cout;

initial begin

a<=10; b<=4; cin<=1;

#5 a<=8; b<=2; cin<=0;

#5 a<=5 ; b<=10; cin<=0;

#5 a<=15; b<=2; cin<=0;

#5 a<=11; b<=4; cin<=1;

end

sumator\_4biti SUMATOR\_4biti( a, b, cin, out, cout);

endmodule

**module alu4**(a, b, f, cin, m, o, cout, egal);

input [3:0]a, b, f;

input cin, m;

output reg [3:0]o;

output reg cout, egal;

always @(\*)

begin

if (a == b) egal=1;

else if(m==1) //operatii logice

begin

if (f == 0) o = ~(a);

else if (f == 1) o = ~(a | b);

else if (f == 2) o = (~a) & b;

else if (f == 3) o = 0;

else if (f == 4) o = ~(a & b);

else if (f == 5) o = ~b;

else if (f == 6) o = a ^ b;

else if (f == 7) o = a & (~b);

else if (f == 8) o = (~a) | b;

else if (f == 9) o = ~(a ^ b);

else if (f == 10) o = b;

else if (f == 11) o = a & b;

else if (f == 12) o = 15;

else if (f == 13) o = a | (~b);

else if (f == 14) o = a | b;

else if (f == 15) o = a;

end

else if (m == 0) //operatii aritmetice

begin

if (f == 0) {cout, o} = a + cin;

else if (f == 1) {cout, o} = a | b + cin;

else if (f == 2) {cout, o} = a | (~b) + cin;

else if (f == 3) {cout, o} = -1 + cin;

else if (f == 4) {cout, o} = a + a | (~b) + cin;

else if (f == 5) {cout, o} = (a | b) + (a & (~b)) + cin;

else if (f == 6) {cout, o} = a - b + cin - 1;

else if (f == 7) {cout, o} = a | (~b) - 1 + cin ;

else if (f == 8) {cout, o} = a + (a & b) + cin;

else if (f == 9) {cout, o} = a + b + cin;

else if (f == 10) {cout, o} = (a | (~b)) + (a & b) + cin;

else if (f == 11) {cout, o} = (a & b) - 1 + cin;

else if (f == 12) {cout, o} = a + a + cin;

else if (f == 13) {cout, o} = (a | b) + a + cin;

else if (f == 14) {cout, o} = (a | (~b)) + a + cin;

else if (f == 15) {cout, o} = a - 1 + cin;

end

end

endmodule

module test\_alu4();

reg [3:0]a, b, f;

reg cin, m;

wire [3:0]o;

wire cout, egal;

initial begin

a=9;b=6;f=0;m=1; //not a = 0110

#5 a=9;b=6;f=1;m=1; //not (a sau b) = 0000

#5 a=9;b=6;f=2;m=1; //(not a) si b = 0110

#5 a=9;b=6;f=31;m=1; //0 = 0000

#5 a=9;b=6;f=4;m=1; //not (a si b) = 1111

#5 a=9;b=6;f=5;m=1; //not b = 1001

#5 a=9;b=6;f=6;m=1; //a xor b = 1111

#5 a=9;b=6;f=7;m=1; //a si (not b) = 1001

#5 a=9;b=6;f=8;m=1; //(not a) sau b = 0110

#5 a=9;b=6;f=9;m=1; //xnor(a, b) = 0000

#5 a=9;b=6;f=10;m=1; //b = 0110

#5 a=9;b=6;f=11;m=1; //a si b = 0000

#5 a=9;b=6;f=12;m=1; //1 = 1111

#5 a=9;b=6;f=13;m=1; //a sau (b negat) = 1001

#5 a=9;b=6;f=14;m=1; //a sau b = 1111

#5 a=9;b=6;f=15;m=1; //a = 1001

#10

a=9; b=6; f=0; m=0; cin = 1; // a + cin = 1010

#5 a=9; b=6; f=1; m=0; cin = 0; // a | b + cin = 1111

#5 a=9; b=6; f=2; m=0; cin = 1; // a | (~b) + cin =

#5 a=9; b=6; f=3; m=0; cin = 0; // -1 + cin =

#5 a=9; b=6; f=4; m=0; cin = 1; // a + a | (~b) + cin

#5 a=9; b=6; f=5; m=0; cin = 0; // (a | b) + (a & (~b))

#5 a=9; b=6; f=6; m=0; cin = 1; // a - b + cin – 1 =

#5 a=9; b=6; f=7; m=0; cin = 0; // a | (~b) - 1 + cin =

#5 a=9; b=6; f=8; m=0; cin = 1; // a + (a & b) + cin =

#5 a=9; b=6; f=9; m=0; cin = 0; // a + b + cin =

#5 a=9; b=6; f=10; m=0; cin = 1; // (a | (~b)) + (a & b)

#5 a=9; b=6; f=11; m=0; cin = 0; // (a & b) - 1 + cin =

#5 a=9; b=6; f=12; m=0; cin = 1; // a + a + cin =

#5 a=9; b=6; f=13; m=0; cin = 0; // (a | b) + a + cin =

#5 a=9; b=6; f=14; m=0; cin = 1; // (a | (~b)) + a + cin

#5 a=9; b=6; f=15; m=0; cin = 0; // a - 1 + cin =

#5 a=9; b=9;

end

alu4 gate\_alu4(a, b, f, cin, m, o, cout, egal);

endmodule

**NBCD3**

**module sumator\_4biti(a, b, cin, out, cout);**

**module comparator\_4biti(a, b, mic, egal, mare);**

**module porti\_and**(s, mic, y);

input [3:0]s;

input mic;

output [3:0] y;

and (y[0], s[0], mic);

and (y[1], s[1], mic);

and (y[2], s[2], mic);

and (y[3], s[3], mic);

endmodule

module test\_nbcd();

reg [3:0]n, b1, b2;

reg cin;

wire [3:0]s, y;

wire cout, mic, mare, egal;

initial begin

//de la 10 la 15 iesirea este 0 /de la 0 la 9 iesirea este 1

n=10; b1=3; b2 = 10; cin =1; //10 + 3 + 1 = 14 14 > 10 af. 0

#10 n=15; b1=3; b2 = 10; cin =0; //15+3=18 15 > 10 af. 0

#10 n=1; b1=3; b2=10; cin =1; //1 + 3 + 1 = 5 1 < 10 af. 5

#10 n=7; b1=3; b2=10; cin =0; //7 + 3 = 10 7 < 10 af. 10

end

sumator\_4biti suma4(n, b1, cin, s, cout);

comparator\_4biti comp4(n, b2, mic, egal, mare);

porti\_and final(s, mic, y);

endmodule

**module numarator**(data, up\_down, enable, load, clock, reset, result, terminal\_count);

parameter k = 4;

input up\_down, enable, load, clock, reset;

input [k-1:0] data;

output reg terminal\_count;

output reg[k-1:0] result;

always @(posedge clock) begin //1

if(reset == 0) result = 0; //resetare numerator

else begin //2

if (load == 1) result = data; //incarc valoarea

else begin //3

if(enable == 1) begin //validare numarare 4

terminal\_count = 0;

if(up\_down == 1) begin //5

result = result + 1;

if(result == 15) terminal\_count = 1;

end //5

else begin //5

result = result - 1;

if(result == 0) terminal\_count = 1;

end //5

end //4

end //3

end //2

end //1

endmodule

module test\_numarator();

parameter k = 4;

reg up\_down, enable, load, clock, reset;

reg [k-1:0] data;

wire terminal\_count;

wire [k-1:0] result;

initial begin

clock = 0;

forever #5 clock = ~clock;

end

initial begin

reset=0;

#15 reset = 1; enable = 1; load = 0; up\_down = 0;

#35 load = 1;

data = 5;

#15 load = 0;

end

numarator counter( data, up\_down, enable, load, clock, reset, result, terminal\_count);

endmodule

**RAM**

**module RAM(Dout, address, clk, cs, oe, we);**

input[3:0] address;

input clk, cs, oe, we;

inout [3:0] Dout;

reg [3:0] memorie [15:0];

reg [3:0] data\_Read;

assign Dout = data\_Read;

always@(posedge clk)

begin

if(cs == 1) begin

if(we == 1)

begin

data\_Read <= 4'bz;

memorie[address] <= Dout;

end

else begin if (oe == 1) data\_Read<=memorie[address]; end

end

else begin

data\_Read<=4'bz;

end

end

endmodule

module test\_RAM();

reg[3:0] address;

reg clk, cs, oe, we;

wire [3:0] Dout;

reg[3:0] data\_Write;

integer i;

assign Dout = data\_Write;

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

cs= 1; oe = 1; address = 0; data\_Write = 2;

//scriere in memorie din reg data\_write

for(i=0; i<16; i = i+1) begin

we=1;

#20 address = address + 1;

data\_Write = data\_Write + 1;

end

//citire din memorie

data\_Write = 4'bz;

for(i=0; i<16; i = i+1) begin

we=0;

#20 address = address - 1;

end

#50 oe = 0;

end

initial #1000 $stop;

RAM memorieRAM(Dout, address, clk, cs, oe, we);

Endmodule

**LATCH\_D**

**module latch\_D(clk, data, q, q\_neg);**

input clk, data;

output reg q, q\_neg;

always @(posedge clk) begin

q = data;

q\_neg = ~data;

end

endmodule

module test\_latch();

reg clk, data;

wire q, q\_neg;

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

data = 0;

#15 data = 1;

end

latch\_D latchD(clk, data, q, q\_neg);

endmodule

**REGISTRU DE DEPLASARE**

**module latch\_D(q, q\_neg, data, clk);**

**module reg\_depl(q, q\_neg, serial\_in, clk);**

input clk, serial\_in;

output [3:0]q, q\_neg;

latch\_D D0(q[0], q\_neg[0], serial\_in, clk);

latch\_D D1(q[1], q\_neg[1], q[0], clk);

latch\_D D2(q[2], q\_neg[2], q[1], clk);

latch\_D D3(q[3], q\_neg[3], q[2], clk);

endmodule

module test\_reg\_depl();

reg clk, serial\_in;

wire [3:0]q, q\_neg;

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

serial\_in = 0;

#15 serial\_in = 1;

#15 serial\_in = 0;

#15 serial\_in = 1;

end

reg\_depl reg\_depl4(q, q\_neg, serial\_in, clk);

endmodule

**REGISTRU DE DEPLASARE BIDIRECTIONAL**

**module reg\_depl\_bidir(clk, rst, sd, sin, din, data, q);**

input clk,rst,sin,din;

input [1:0]sd;

input [3:0]data;

output reg [3:0]q;

always @(negedge clk or rst) begin

if (rst == 1) q<=0;

else begin

if (sd == 0) q<=q;

if (sd == 1) q<={din, q[3:1]};

if (sd == 2) q<={q[2:0], sin};

if (sd == 3) q<=data;

end

end

endmodule

module test\_reg\_bidir();

reg clk,rst,sin,din;

reg [1:0]sd;

reg [3:0]data;

wire [3:0]q;

initial begin

clk = 0;

forever #5 clk = ~clk;

end

initial begin

rst = 1;

data = 12; din=1; sin=1; sd=0;

#15 rst = 0;

#15 sd=1;

#15 sd=2;

#15 sd=3;

#20 rst=1;

end

reg\_depl\_bidir depl\_bidir(clk, rst, sd, sin, din, data, q);

endmodule

**STIVA**

**module stiva(data, push, pop, reset, full, empty, err, sp);**

input push, pop, reset;

output reg full, empty, err;

output reg[2:0] sp;

inout [3:0] data;

reg[3:0] data\_out, stack[7:0];

assign data=data\_out;

always @(posedge push or posedge pop or posedge reset) begin

if(reset == 1) begin

full <= 0;

empty <= 0;

err <= 0;

sp <= 0;

data\_out <= 4'bz;

end

if(push == 1) begin

if(empty == 1) begin

stack[sp] <=data;

empty <= 0;

err <= 0;

end

if (full == 1) begin err <= 1; end

else begin

sp <= sp + 1;

stack[sp] <= data;

if(sp == 7) begin

stack[sp] <= data;

full <= 1;

end

end

end

if (pop == 1) begin

if(sp == 0 && empty == 0) begin

data\_out <= stack[sp];

empty <= 1;

end

else

if (empty == 1) begin err <=1; end

else begin

data\_out <=stack[sp];

sp <= sp - 1;

full <= 0;

err <= 0;

end

end

end

always @(negedge pop) begin

data\_out <= 4'bz;

end

endmodule

module test\_stiva();

reg push, pop, reset;

reg [3:0] data\_in;

wire full, empty, err;

wire [2:0] sp;

wire [3:0] data\_io;

integer i;

assign data\_io = data\_in;

initial begin

reset = 1;

#2 reset = 0; pop = 1;

#1 pop = 0; data\_in = 0;

for (i = 0; i< 7; i = i + 1) begin

#2 push = 1;

#1 push = 0;

data\_in = data\_in + 1;

end

#1 data\_in = 4'bz;

for (i = 0; i< 7; i = i + 1) begin

#2 pop = 1;

#1 pop = 0;

end

end

stiva stack\_st(data\_io, push, pop, reset, full, empty, err, sp);

endmodule