`timescale 1ns / 1ps

module aoi(

input a, b, c, d,

output e

);

assign e = ~((a&b)|(c&d));

endmodule

`timescale 1ns / 1ps

module nandga(

input a, b, c, d,

output e

);

assign e = ~(a&b&c&d);

endmodule

module nandgb(

input a, b, c, d,

output e, f, g

);

assign e = ~(a&b);

assign f = ~(e&c);

assign g = ~(f&d);

endmodule

`timescale 1ns / 1ps

module norga(

input a, b, c, d,

output e

);

assign e = ~(a|b|c|d);

endmodule

module norgb(

input a, b, c, d,

output e, f, g

);

assign e = ~(a|b);

assign f = ~(e|c);

assign g = ~(f|d);

endmodule

`timescale 1ns / 1ps

module xorga(

input a, b, c, d,

output e

);

assign e = a^b^c^d;

endmodule

module xorgb(

input a, b, c, d,

output e, f, g

);

assign e = a^b;

assign f = e^c;

assign g = f^d;

endmodule

`timescale 1ns / 1ps

module boolf1A(

input a, b, c,

output out

);

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

endmodule

module boolf1B(

input a, b, c,

output out

);

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

endmodule

module boolf2A(

input a, b, c,

output out

);

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

endmodule

module boolf2B(

input a, b, c,

output out

);

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

endmodule

`timescale 1ns / 1ps

module cmp(

input a, b,

output eq, neq, lg, le

);

assign eq = (~a&~b)|(a&b);

assign neq = (~a&b)|(a&~b);

assign lg = a&~b;

assign le = ~a&b;

endmodule

`timescale 1ns / 1ps

module dm1A(

input a, b,

output out

);

assign out = ~(a|b);

endmodule

module dm1B(

input a, b,

output out

);

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

endmodule

module dm2A(

input a, b,

output out

);

assign out = ~(a&b);

endmodule

module dm2B(

input a, b,

output out

);

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

endmodule

`timescale 1ns / 1ps

module half\_adder(

input a, b,

output s, c

);

assign s = a^b;

assign c = a&b;

endmodule

module full\_adder(

input a, b, cin,

output s, cout

);

assign s = a^b^cin;

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

endmodule

`timescale 1ns / 1ps

module converter(

input a, b, c, d,

output e, f, g, h

);

assign e = a|(b&d)|(b&c);

assign f = a|(b&c)|(b&~d);

assign g = (b&~c&d)|a|(~b&c);

assign h = d;

endmodule

module converterPOS(

input a, b, c, d,

output e, f, g, h

);

assign e = (a|c|d)&(a|b);

assign f = (a|b)&(a|c|~d);

assign g = (a|b|c)&(a|c|d)&(~b|~c);

assign h = d;

endmodule

`timescale 1ns / 1ps

module half\_sub(

input a, b,

output borrow, d

);

assign borrow = ~a&b;

assign d = a^b;

endmodule

module full\_sub(

input a, b, bin,

output bout, d

);

assign bout = (bin&(~a|b))|(~a&b);

assign d = a^b^bin;

endmodule

`timescale 1ns / 1ps

module ParityGen(

input a, b, c, d,

output p

);

assign p = (a^b)^(c^d);

endmodule

module ParityCheck(

input a, b, c, d, p,

output pec

);

assign pec = a^b^c^d^p;

endmodule

module cmp(

input a, b, c, d,

output lg, eq, le

);

assign lg = (a&~c)+(b&~c&~d)+(a&b&~d);

assign eq = ~((b^d)|(a^c));

assign le = (~a&c)+(~b&c&d)+(~a&~b&d);

endmodule

`timescale 1ns / 1ps

module segment(

input a, b, c, d,

output ao, bo, co, do, eo, fo, go, dp, digit

);

assign ao = (~b&~d)|(a&~d)|(a&~b&~c)|(~a&b&d)|(~a&c)|(b&c);

assign bo = (~b&~c)|(~b&~d)|(~a&~c&~d)|(~a&c&d)|(a&~c&d);

assign co = (~a&b)|(a&~b)|(~c&d)|(~a&~c)|(~a&d);

assign do = (~b&~c&~d)|(~b&c&d)|(~a&c&~d)|(b&~c&d)|(a&b&~d);

assign eo = (~b&~d)|(c&~d)|(a&b)|(a&c);

assign fo = (~c&~d)|(b&~d)|(~a&b&~c)|(a&c)|(a&~b);

assign go = (~a&~b&c)|(c&~d)|(~a&b&~c)|(a&d)|(a&~b);

assign dp = 1;

assign digit = 1;

endmodule

`timescale 1ns / 1ps

module ah\_decoder(

input in1, in2,

output out1, out2, out3, out4

);

assign out1 = ~in1&~in2;

assign out2 = ~in1&in2;

assign out3 = in1&~in2;

assign out4 = in1&in2;

endmodule

module al\_decoder(

input in1, in2,

output out1, out2, out3, out4

);

assign out1 = ~(~in1&~in2);

assign out2 = ~(~in1&in2);

assign out3 = ~(in1&~in2);

assign out4 = ~(in1&in2);

endmodule

module encoder (

input a, b, c, d,

output e0, e1

);

assign e0 = a|b;

assign e1 = a|c;

endmodule

module bcd2dec(

input a3, a2, a1, a0,

output o0, o1, o2, o3, o4, o5, o6, o7, o8, o9

);

assign o0 = ~a3&~a2&~a1&~a0;

assign o1 = ~a3&~a2&~a1&a0;

assign o2 = ~a3&~a2&a1&~a0;

assign o3 = ~a3&~a2&a1&a0;

assign o4 = ~a3&a2&~a1&~a0;

assign o5 = ~a3&a2&~a1&a0;

assign o6 = ~a3&a2&a1&~a0;

assign o7 = ~a3&a2&a1&a0;

assign o8 = a3&~a2&~a1&~a0;

assign o9 = a3&~a2&~a1&a0;

endmodule

module mux41(

input a, b, c, d, s1, s0,

output q

);

assign q = (a&~s0&~s1)|(b&s0&~s1)|(c&~s0&s1)|(d&s0&s1);

endmodule

module mux81(

input a, b, c, d, e, f, g, h, s2, s1, s0,

output q

);

assign q = (a&~s0&~s1&~s2)|(b&s0&~s1&~s2)|(c&~s0&s1&~s2)|

(d&s0&s1&~s2)|(e&~s0&~s1&s2)|(f&s0&~s1&s2)|(g&~s0&s1&s2)|(h&s0&s1&s2);

endmodule

module demux14(

input s1, s0, f,

output a, b, c, d

);

assign a = f&(~s1&~s0);

assign b = f&(~s1&s0);

assign c = f&(s1&~s0);

assign d = f&(s1&s0);

endmodule

module decoder416(

input s3, s2, s1, s0,

output o0, o1, o2, o3, o4, o5, o6, o7, o8, o9, o10, o11, o12, o13, o14, o15

);

wire en0, en1, en2, en3;

demux14 dm(s3, s2, 1, en0, en1, en2, en3);

demux14 dm0(s1, s0, en0, o0, o1, o2, o3);

demux14 dm1(s1, s0, en1, o4, o5, o6, o7);

demux14 dm2(s1, s0, en2, o8, o9, o10, o11);

demux14 dm3(s1, s0, en3, o12, o13, o14, o15);

endmodule

`timescale 1ns / 1ps

module full\_adder(a, b, cin, sum, cout);

input a, b, cin;

output sum, cout;

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

assign sum = (a^b)^cin;

endmodule

module rca\_4bit(a, b, sign, s, c4);

input [3:0] a, b;

output [3:0] s;

input sign;

output c4;

wire c1, c2, c3;

full\_adder fa0(a[0], b[0]^sign, sign, s[0], c1);

full\_adder fa1(a[1], b[1]^sign, c1, s[1], c2);

full\_adder fa2(a[2], b[2]^sign, c2, s[2], c3);

full\_adder fa3(a[3], b[3]^sign, c3, s[3], c4);

endmodule

module bcd\_adder(a, b, sum, cout);

input [3:0] a, b;

output [3:0] sum, cout;

wire carryOut;

wire [3:0] binarySum;

rca\_4bit rca0(a, b, 0, binarySum, carryOut);

assign cout = carryOut|(binarySum[3]&binarySum[2])|(binarySum[3]&binarySum[1]);

rca\_4bit rca1(binarySum, 4'b0110 & {4{cout[0]}}, 0, sum);

endmodule

`timescale 1ns / 1ps

module SRLatch(

en, s, r, q, qbar

);

input en, s, r;

output q, qbar;

reg q;

assign qbar = ~q;

always @(en or s or r) begin

if(en) begin

if((s==1)&&(r==1))

q <= 1'bx;

else if((s==1)&&(r==0))

q <= 1'b1;

else if((s==0)&&(r==1))

q <= 1'b0;

else

q <= q;

end

end

endmodule

module srlatch (

input S,

input R,

input En,

output reg Q,

output reg Qc

);

always @(\*) begin

if (En) begin

Q = ~(R | Qc);

Qc = ~(S | Q);

end

end

endmodule : srlatch

module SRFlipFlop(

clk, s, r, q, qbar

);

input clk, s, r;

output q, qbar;

reg q;

assign qbar = ~q;

always @(posedge clk) begin

if((s==1)&&(r==1))

q <= 1'bx;

else if((s==1)&&(r==0))

q <= 1'b1;

else if((s==0)&&(r==1))

q <= 1'b0;

else

q <= q;

end

endmodule

module DLatch(

en, d, q, qbar

);

input en, d;

output q, qbar;

reg q;

assign qbar = ~q;

always @(en or d) begin

if(en)

q <= d;

end

endmodule

module DFlipFlop(

clk, d, q, qbar

);

input clk, d;

output q, qbar;

reg q;

assign qbar = ~q;

always @(posedge clk) begin

q <= d;

end

endmodule

module JKFlipFlop(

clk, j, k, q, qbar

);

input clk, j, k;

output q, qbar;

reg q;

assign qbar = ~q;

always @(posedge clk) begin

if((j==1)&&(k==1))

q <= ~q;

else if((j==1)&&(k==0))

q <= 1'b1;

else if((j==0)&&(k==1))

q <= 1'b0;

else

q <= q;

end

endmodule

`timescale 1ns / 1ps

module two\_bit\_counter(

clk, rst, out

);

input clk, rst;

output[1:0] out;

reg[1:0] out;

initial out = 2'b00;

always @(posedge clk) begin

if(rst) begin

out <= 2'b00;

end

else begin

out <= out + 1;

end

end

endmodule

module decade\_counter(

clk, rst, out

);

input clk, rst;

output[3:0] out;

reg[3:0] out;

initial out = 4'b0000;

always @(posedge clk) begin

if(rst) begin

out <= 4'b0000;

end

else if(out >= 4'b1001) begin

out <= 4'b0000;

end

else begin

out <= out + 1;

end

end

endmodule

module bcd\_to\_2421(

in, out

);

input[3:0] in;

output[3:0] out;

assign out[3] = in[3]|(in[2]&in[0])|(in[2]&in[1]);

assign out[2] = in[3]|(in[2]&in[1])|(in[2]&~in[0]);

assign out[1] = (in[2]&~in[1]&in[0])|in[3]|(~in[2]&in[1]);

assign out[0] = in[0];

endmodule

module code2421\_counter(

clk, rst, out

);

input clk, rst;

output[3:0] out;

wire[3:0] connecter;

decade\_counter dc(

.clk(clk),

.rst(rst),

.out(connecter)

);

bcd\_to\_2421 bt2(

.in(connecter),

.out(out)

);

endmodule

`timescale 1ns / 1ps

module shift\_register(shr, rst, shr\_in, clk, Q);

input clk, rst, shr\_in, shr;

output[3:0] Q;

reg [3:0] Q;

always @(posedge clk) begin

if(rst) begin

Q <= 4'b0000;

end

else if(shr) begin

Q[0] <= Q[1];

Q[1] <= Q[2];

Q[2] <= Q[3];

Q[3] <= shr\_in;

end

end

endmodule

module ring\_counter(clk, rst, Q);

input clk, rst;

output [3:0] Q;

reg [3:0] Q;

always @(posedge clk) begin

if(rst)

Q = 4'b1000;

else begin

Q[0] <= Q[1];

Q[1] <= Q[2];

Q[2] <= Q[3];

Q[3] <= Q[0];

end

end

endmodule

module up\_down\_counter(up, clk, Q, ss, digit);

input up, clk;

output [3:0] Q;

output [6:0] ss;

output digit;

assign digit = 1'b1;

reg[3:0] Q;

reg[6:0] ss;

initial Q <= 4'b0000;

always @(posedge clk) begin

if(up) begin

Q <= Q + 4'b0001;

ss <= 7'b0111110;

end

else begin

Q <= Q - 4'b0001;

ss <= 7'b0111101;

end

end

endmodule

module shif(shr,rst,shr\_in,clk,Q);

input shr , shr\_in;

input clk,rst;

output [3:0] Q;

reg [3:0] Q;

always @(posedge clk) begin

if(rst == 1)

R<=4'b0000;

else if (shr ==1) begin

R[3] <= shr\_in;

R[2] <= R[3];

R[1] <= R[2];

R[0] <= R[1];

end

end

//assign Q =R;

endmodule

module inv\_sim;

reg clk;

reg rst;

reg shr,shr\_in;

wire [3:0] out;

inv aoi(shr,rst,shr\_in,clk,out);

always #5 clk = ~clk;

always #3 shr = ~shr;

always #20 shr\_in = ~shr\_in;

initial begin

shr <= 0;

shr\_in <= 1;

clk <= 0;

rst <= 0;

#20 rst <=0;

#80 rst <=1;

#50 rst <=0;

end

endmodule

module ring(clk,clr,Q);

input clk;

input clr;

output [3:0] Q;

reg [3:0] Q;

always @(posedge clk or posedge clr)

begin

if(clr==1)

Q<=1;

else

begin

Q[3] <= Q[0];

Q[2:0] <= Q[3:1];

end

end

endmodule

`timescale 1ns / 1ps

module inv\_sim;

//x,clk,preset,clear,outz

reg clk,clr;

wire [3:0] Q;

inv aoi(clk,clr,Q);

always #5 clk = ~clk;

initial begin

clk <= 0;

clr <= 0;

#20 clr <=0;

#80 clr <=1;

#50 clr <=0;

end

endmodule

//(UP/DOWN counter)

`timescale 1ns / 1ps

module inv(clk,up,clr,Q,x,seg);

input clk,up;

input clr;

output [3:0] Q;

output x;

output [6:0] seg;

reg [3:0] Q;

reg [6:0] seg;

always @(posedge clk or posedge clr)

begin

if(~clr)

begin

Q <=0;

seg = 7'b1111111;

end

else if(up)

begin

seg = 7'b0111110;

Q=Q+1;

end

else

begin

seg = 7'b0111101;

Q=Q-1;

end

end

assign x = 1;

endmodule

`timescale 1ns / 1ps

module inv\_sim;

reg clk,clr,up;

wire [3:0] Q;

wire x;

wire [6:0] seg;

inv aoi(clk,up,clr,Q,x,seg);

always #5 clk = ~clk;

always #20 up = ~up;

initial begin

up<=1;

clk <= 0;

clr <= 0;

#20 clr <=0;

#80 clr <=1;

#40 clr <=0;

#20 clr <=1;

#30 clr <=0;

end

endmodule