VERILOG BASIC PROGRAMS USING GATE LEVEL DATA FLOW AND STRUCTURAL MODELLING

K-VLSI PROGRAM

IIIT-BANGLORE

PREPARED BY

SHRUTHI NAIK

KVLSI2501091

P1). Gate level code for 2:1 MUX

Design code:

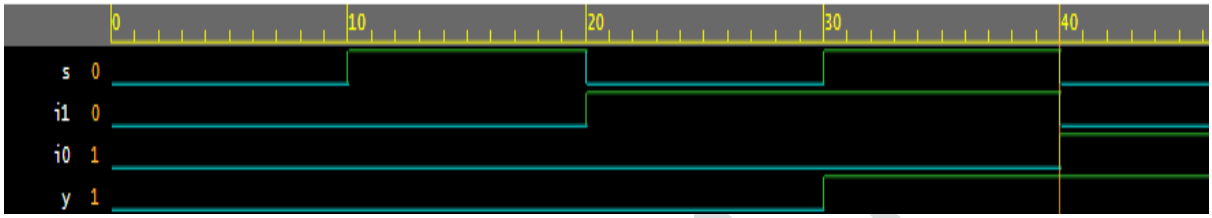
```
module mux(i1, i0, s, y);
input i1, i0, s;
output y;
wire w1, w2, w3;
not n1 (w1, s);
and al (w2, i0, w1);
and a2 (w3, s, i1);
or o1 (y, w2, w3);
endmodule
```

Test bench code:

```
module mux_test;
  reg i0, i1, s;
  wire y;
 mux21 dut(.i0(i0), .i1(i1), .s(s), .y(y));

  initial begin
         i0=0; i1=0; s=0;
    #10 i0=0; i1=0; s=1;
    #10 i0=0; i1=1; s=0;
    #10 i0=0; i1=1; s=1;
    #10 i0=1; i1=0; s=0;
    #10 i0=1; i1=0; s=1;
    #10 i0=1; i1=1; s=0;
    #10 i0=1; i1=1; s=1;
  end

initial begin
     $monitor("Simulation time = %0t, i0 =
    %b, i1 =   %b, s = %b, y = %b", $time, i0,
    i1, s, y);
end
initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, i0, i1, s, y);
end
endmodule
```

OUTPUT:

```
Simulation time = 0, i0 = 0, i1 =   0, s = 0, y = 0
Simulation time = 10, i0 = 0, i1 =   0, s = 1, y = 0
Simulation time = 20, i0 = 0, i1 =   1, s = 0, y = 0
Simulation time = 30, i0 = 0, i1 =   1, s = 1, y = 1
Simulation time = 40, i0 = 1, i1 =   0, s = 0, y = 1
Simulation time = 50, i0 = 1, i1 =   0, s = 1, y = 0
Simulation time = 60, i0 = 1, i1 =   1, s = 0, y = 1
Simulation time = 70, i0 = 1, i1 =   1, s = 1, y = 1
```

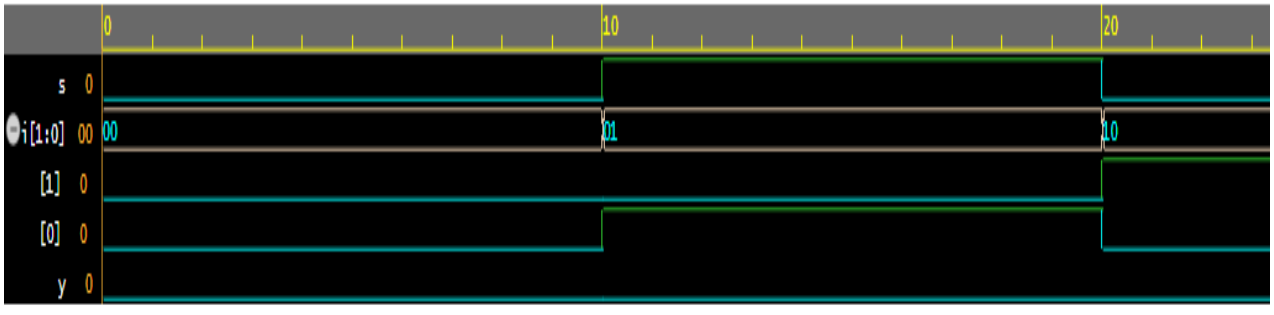


P2). Gate level code for 2:1 MUX using vector and concatenation

Design code:	*Test bench code:*
module mux(i, s, y); input [1:0]i; input s; output y; wire w[3:1]; not n1 (w[1], s); and al (w[2], i[0], w[1]); and a2 (w[3], s, i[1]); or o1 (y, w[2], w[3]); endmodule	module mux_test; reg [1:0]i; reg s; wire y; mux dut(.i(i), .s(s), .y(y)); initial begin {i, s} = 0; #10 {i, s} = 3; #10 {i, s} = 6; #10 {i, s} = 12; end initial begin $monitor("Simulation time= %0t, i=%b, s=%b, y=%b", $time, i, s, y); end initial begin $dumpfile("dump.vcd"); $dumpvars(0, i, s, y); end endmodule

OUTPUT:

```
Simulation time= 0, i=00, s=0, y=0
Simulation time= 10, i=01, s=1, y=0
Simulation time= 20, i=11, s=0, y=1
Simulation time= 30, i=10, s=0, y=0
```



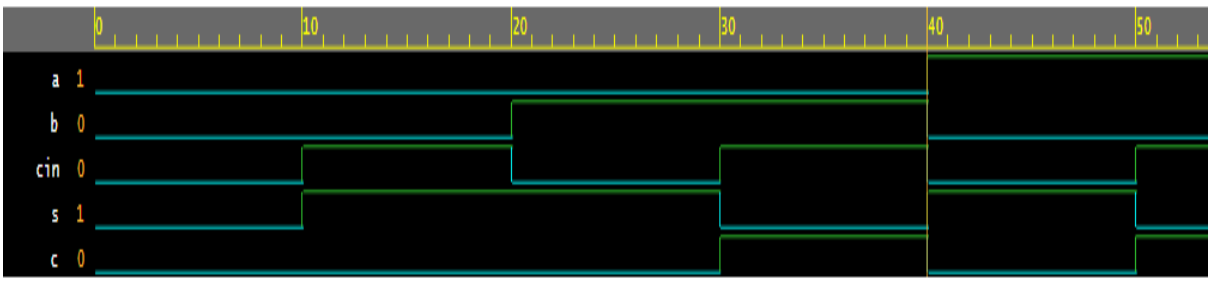
P3).Gate level code for full adder

Design code:	*Test bench code:*

module full_adder(a, b, cin, s, c);
input a, b, cin;
output s, c;
wire w1, w2, w3;
xor x1(s, a, b, cin);
and a1(w1, a, b);
and a2(w2, b, cin);
and a3(w3, a, cin);
or o1(c, w1, w2, w3);
endmodule
	module full_adder_test;
	reg a,b,cin;
	wire s,c;
	full_adder dut(.a(a), .b(b), .cin(cin), .s(s), .c(c));
	initial begin
	a=0; b=0; cin=0;
	#10 a=0; b=0; cin=1;
	#10 a=0; b=1; cin=0;
	#10 a=0; b=1; cin=1;
	#10 a=1; b=0; cin=0;
	#10 a=1; b=0; cin=1;
	#10 a=1; b=1; cin=0;
	#10 a=1; b=1; cin=1;
	end
	initial begin
	$monitor("Simulation time = %0t, a = %b, b = %b, cin = %b, s = %b, c=%b", $time, a, b, cin, s, c);
	end
	initial begin
	$dumpfile("dump.vcd");
	$dumpvars(0, a, b, cin, s, c);
	end
	endmodule

OUTPUT:

```
Simulation time = 0, a = 0, b =   0, cin = 0, s = 0, c=0
Simulation time = 10, a = 0, b =  0, cin = 1, s = 1, c=0
Simulation time = 20, a = 0, b =  1, cin = 0, s = 1, c=0
Simulation time = 30, a = 0, b =  1, cin = 1, s = 0, c=1
Simulation time = 40, a = 1, b =  0, cin = 0, s = 1, c=0
Simulation time = 50, a = 1, b =  0, cin = 1, s = 0, c=1
Simulation time = 60, a = 1, b =  1, cin = 0, s = 0, c=1
Simulation time = 70, a = 1, b =  1, cin = 1, s = 1, c=1
```



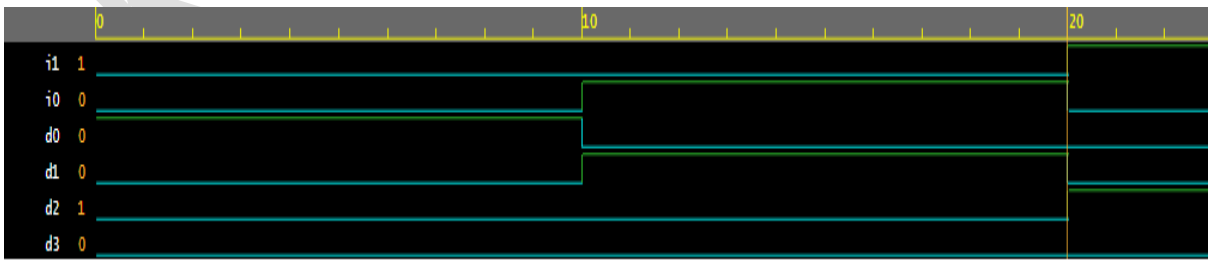
P4). Gate level code for 2x4 decoder

Design code:	*Test bench code:*

module decoder(i1, i0, d0, d1, d2, d3);
input i1, i0;
output d0, d1, d2, d3;
wire w1, w2;
not n1(w1, i0);
not n2(w2, i1);
and a1(d0, w2, w1);
and a2(d1, w2, i0);
and a1(d2, i1, w1);
and a1(d3, i1, i0);
endmodule
	module decoder_test;
	reg i1, i0;
	wire d0, d1, d2,d3,d4;
	
	decoder dut(.i1(i1), .i0(i0), .d0(d0), .d1(d1), .d2(d2), .d3(d3));
	
	initial begin
	i1=0; i0=0;
	#10 i1=0; i0=1;
	#10 i1=1; i0=0;
	#10 i1=1; i0=1;
	end
	
	initial begin
	$monitor("Simulation time = %0t, i1 = %b, i0 = %b, d0 = %b, d1 = %b, d2=%b d3=%b", $time, i1, i0, d0, d1, d2, d3);
	end
	initial begin
	$dumpfile("dump.vcd");
	$dumpvars(0, i0, i1, d0, d1, d2, d3);
	end
	endmodule

OUTPUT:

```
Simulation time = 0, i1 = 0, i0 =   0, d0 = 1, d1 = 0, d2=0 d3=0
Simulation time = 10, i1 = 0, i0 =  1, d0 = 0, d1 = 1, d2=0 d3=0
Simulation time = 20, i1 = 1, i0 =  0, d0 = 0, d1 = 0, d2=1 d3=0
Simulation time = 30, i1 = 1, i0 =  1, d0 = 0, d1 = 0, d2=0 d3=1
xmsim: *W,RNQUIE: Simulation is complete.
```



P5). Data flow code for 2x4 decoder

Design code:

```
module deco24(I, D);
  input [1:0]I;
  output [3:0]D;

  assign D[0]=(~I[1])&(~I[0]);
  assign D[1]=(~I[1])&(I[0]);
  assign D[2]=(I[1])&(~I[0]);
  assign D[3]=(I[1])&(I[0]);

endmodule
```

Test bench code:

```
module deco24_test;

  reg [1:0]I;
  wire [3:0]D;

  deco24 dut(I, D);

  initial begin
      I=2'b00;
    #5 I=2'b01;
    #5 I=2'b10;
    #5 I=2'b11;

  end
  initial begin
    $monitor("SIM TIME= %0t, I=%b,
D=%b ", $time, I, D);
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0,D,I);
  end
endmodule
```

OUTPUT:

```
SIM TIME= 0, I=00, D=0001
SIM TIME= 5, I=01, D=0010
SIM TIME= 10, I=10, D=0100
SIM TIME= 15, I=11, D=1000
xmsim: *W,RNQUIE: Simulation is complete.
```

P6). Data flow code for 4x1 mux using vector and concatenation

Design code:

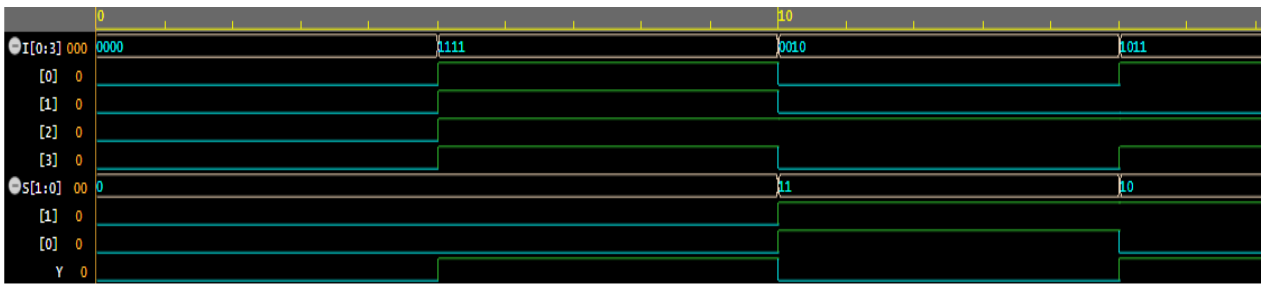
```
module mux41vector(I, S, Y);


  input [0:3]I;
  input [1:0]S;
  output Y;


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


endmodule
```

Test bench code:

```
module mux41vector_test;
  reg [0:3]I;
  reg [1:0]S;
  wire Y;

  mux41vector dut(I, S, Y);

  initial begin
        {I,S}=0;
    #5  {I,S}=1;
    #5  {I,S}=8;
    #5  {I,S}=20;
    #5  {I,S}=30;
  end

  initial begin
    $monitor("Sim time=%0t, I=%b, S=%b,
Y=%b", $time, I, S, Y);
  end
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, I, S, Y);
  end
endmodule
```

OUTPUT:

```
Sim time=0, I=0000, S=00, Y=0
Sim time=5, I=1111, S=00, Y=1
Sim time=10, I=0010, S=11, Y=0
Sim time=15, I=1011, S=10, Y=1
Sim time=20, I=0111, S=10, Y=1
xmsim: *W,RNQUIE: Simulation is complete.
```



P7). Dataflow code for 2x1 mux

Design code:

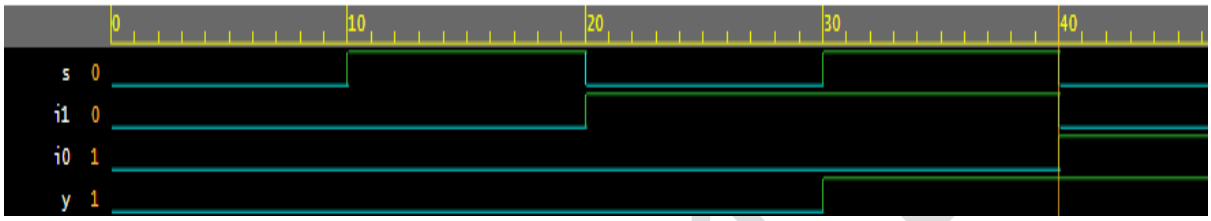
```
module mux21 (i0, i1, s, y);
input i0, i1, s;
output y;
assign y= ((~s)&i0) | (s&i1);
endmodule
```

Test bench code:

```
module mux21_test;
  reg i0, i1, s;
  wire y;
 mux21 dut(.i0(i0), .i1(i1), .s(s), .y(y));

  initial begin
          i0=0; i1=0; s=0;
    #10 i0=0; i1=0; s=1;
    #10 i0=0; i1=1; s=0;
    #10 i0=0; i1=1; s=1;
    #10 i0=1; i1=0; s=0;
    #10 i0=1; i1=0; s=1;
    #10 i0=1; i1=1; s=0;
    #10 i0=1; i1=1; s=1;
  end

initial begin
     $monitor("Simulation time = %0t, i0 =
    %b, i1 =   %b, s = %b, y = %b", $time, i0,
    i1, s, y);
end
initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, i0, i1, s, y);
end
endmodule
```

OUTPUT:

```
Simulation time = 0, i0 = 0, i1 =   0, s = 0, y = 0
Simulation time = 10, i0 = 0, i1 =   0, s = 1, y = 0
Simulation time = 20, i0 = 0, i1 =   1, s = 0, y = 0
Simulation time = 30, i0 = 0, i1 =   1, s = 1, y = 1
Simulation time = 40, i0 = 1, i1 =   0, s = 0, y = 1
Simulation time = 50, i0 = 1, i1 =   0, s = 1, y = 0
Simulation time = 60, i0 = 1, i1 =   1, s = 0, y = 1
Simulation time = 70, i0 = 1, i1 =   1, s = 1, y = 1
```



P8).Gate level code for 4x1 mux

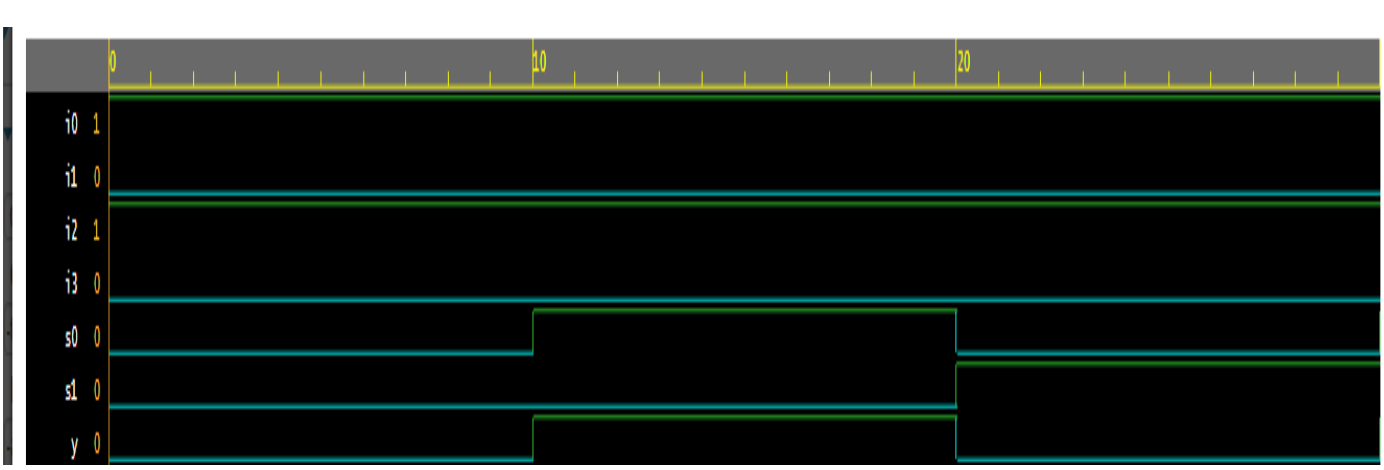
Design code:

```
module mux41(s0, s1, i0, i1, i2, i3, y);
  input s0, s1;
  input i0, i1, i2, i3;
  output y;
  wire x1, x2, w1, w2, w3, w4;

  not n1(x1, s1);
  not n2(x2, s0);
  and a1(w1, x1, x2, i0);
  and a2(w2, x1, s0, i1);
  and a3(w3, s1, x2, i2);
  and a4(w4, s1, s0, i3);
  nor n3(y, w1, w2, w3, w4);

endmodule
```

Test bench code:

```
module mux41_tb;
reg s0, s1;
reg i0, i1, i2, i3;
wire y;

mux41 dut(.s0(s0), .s1(s1), .i0(i0), .i1(i1), .i2(i2),
.i3(i3), .y(y));

 initial begin
   i0=1; i1=0; i2=1; i3=0;
       s1=0; s0=0;
   #10 s1=0; s0=1;
   #10 s1=1; s0=0;
   #10 s1=1; s0=1;
 end

 initial begin
   $monitor("simulation time=%0t, s0=%b, s1=%b,
i0=%b, i1=%b, i2=%b, i2=%b, y=%b", $time, s0,
s1, i0, i1, i2, i3, y);
 end

 initial begin
   $dumpfile("dump.vcd");
   $dumpvars(0, s0, s1,i0, i1, i2, i3, y);
 end
endmodule
```

```
simulation time=0, s0=0, s1=0, i0=1, i1=0, i2=1, i2=0, y=0
simulation time=10, s0=1, s1=0, i0=1, i1=0, i2=1, i2=0, y=1
simulation time=20, s0=0, s1=1, i0=1, i1=0, i2=1, i2=0, y=0
simulation time=30, s0=1, s1=1, i0=1, i1=0, i2=1, i2=0, y=1
xmsim: *W,RNQUIE: Simulation is complete.
```



P9).Data flow code for 4x1 mux

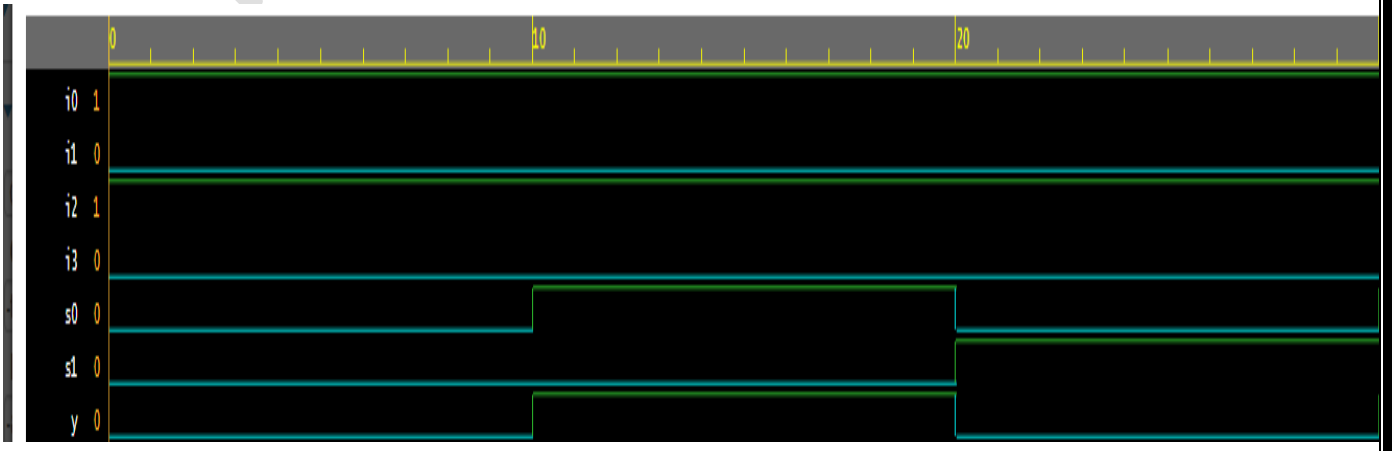
Design code:

```
module mux41(s0, s1, i0, i1, i2, i3, y);
  input s0, s1;
  input i0, i1, i2, i3;
  output y;

  assign
y=((~s1)&(~s0)&i0)|((~s1)&(s0)&i1)|((s1)
&(~s0)&i2)|((s1)&(s0)&i3);

endmodule
```

Test bench code:

```
module mux41_tb;
reg s0, s1;
reg i0, i1, i2, i3;
wire y;

mux41 dut(.s0(s0), .s1(s1), .i0(i0), .i1(i1), .i2(i2),
.i3(i3), .y(y));

 initial begin
   i0=1; i1=0; i2=1; i3=0;
       s1=0; s0=0;
    #10 s1=0; s0=1;
    #10 s1=1; s0=0;
    #10 s1=1; s0=1;
 end

 initial begin
   $monitor("simulation time=%0t, s0=%b, s1=%b,
i0=%b, i1=%b, i2=%b, i2=%b, y=%b", $time, s0,
s1, i0, i1, i2, i3, y);
 end

 initial begin
   $dumpfile("dump.vcd");
   $dumpvars(0, s0, s1,i0, i1, i2, i3, y);
 end
endmodule
```

```
simulation time=0, s0=0, s1=0, i0=1, i1=0, i2=1, i2=0, y=0
simulation time=10, s0=1, s1=0, i0=1, i1=0, i2=1, i2=0, y=1
simulation time=20, s0=0, s1=1, i0=1, i1=0, i2=1, i2=0, y=0
simulation time=30, s0=1, s1=1, i0=1, i1=0, i2=1, i2=0, y=1
xmsim: *W,RNQUIE: Simulation is complete.
```



P10). Gate level code for xor using nand

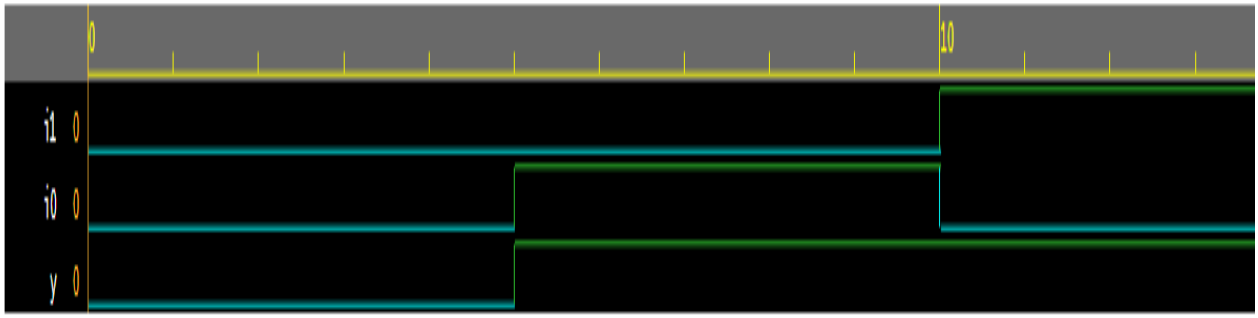
Design code:

```
module xornand(i1, i0, y);
  input i1, i0;
  output y;
  wire w1, w2, w3;

  nand n1(w1, i1,i0);
  nand n2(w2, i1, w1);
  nand n3(w3, i0, w1);
  nand n4(y, w2, w3);

endmodule
```

Test bench code:

```
module xornand_tb;
  reg i1, i0;
  wire y;

  xornand dut(.i1(i1), .i0(i0), .y(y));

  initial begin
      i1=0; i0=0;
    #5 i1=0; i0=1;
    #5 i1=1; i0=0;
    #5 i1=1; i0=1;
  end

  initial begin
    $monitor("simulation time=%0t, i1=%b, i0=%b,
y=%b", $time, i1, i0, y);
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, i0, i1, y);
  end
endmodule
```

```
simulation time=0, i1=0, i0=0, y=0
simulation time=5, i1=0, i0=1, y=1
simulation time=10, i1=1, i0=0, y=1
simulation time=15, i1=1, i0=1, y=0
xmsim: *W,RNQUIE: Simulation is complete.
```



P11). Data flow code for xor using nand

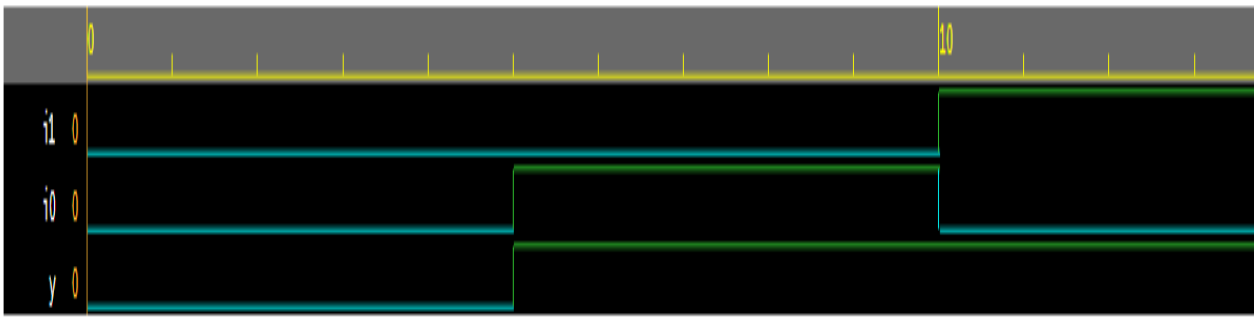
Design code:

```
module xornand(i1, i0, y);
  input i1, i0;
  output y;
  wire w1, w2, w3;

  assign w1 = ~(i1 & i0);
  assign w2 = ~(i1 & w1);
  assign w3 = ~(i0 & w1);
  assign y = ~(w2 & w3);

endmodule
```

Test bench code:

```
module xornand_tb;
  reg i1, i0;
  wire y;

  xornand dut(.i1(i1), .i0(i0), .y(y));

  initial begin
      i1=0; i0=0;
    #5 i1=0; i0=1;
    #5 i1=1; i0=0;
    #5 i1=1; i0=1;
  end

  initial begin
    $monitor("simulation time=%0t, i1=%b, i0=%b,
y=%b", $time, i1, i0, y);
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, i0, i1, y);
  end
endmodule
```

```
simulation time=0, i1=0, i0=0, y=0
simulation time=5, i1=0, i0=1, y=1
simulation time=10, i1=1, i0=0, y=1
simulation time=15, i1=1, i0=1, y=0
xmsim: *W,RNQUIE: Simulation is complete.
```



P12). Data flow code for 2x1 mux using nand

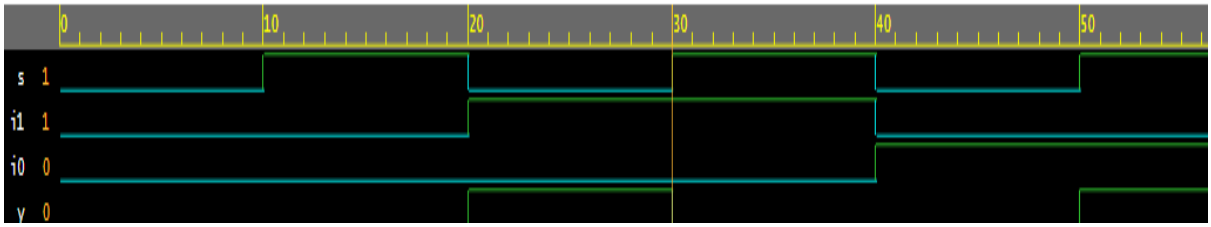
Design code:

```
module muxnand(i1, i0, s, y);
  input i1, i0, s;
  output y;
  wire w1, w2, w3;

  assign w1 = ~(i0 & s);
  assign w2 = ~(s&s);
  assign w3 = ~(i1 & w2);
  assign y = ~(w1 & w3);

endmodule
```

Test bench code:

```
module muxnand_test;
  reg i1, i0, s;
  wire y;
  muxnand dut(.i1(i1), .i0(i0), .s(s), .y(y));

  initial begin
        i1=0; i0=0; s=0;
   #10 i1=0; i0=0; s=1;
   #10 i1=0; i0=1; s=0;
   #10 i1=0; i0=1; s=1;
   #10 i1=1; i0=0; s=0;
   #10 i1=1; i0=0; s=1;
   #10 i1=1; i0=1; s=0;
   #10 i1=1; i0=1; s=1;
  end

initial begin
 $monitor("Simulation time = %0t, i0 = %b, i1 =
%b, s = %b, y = %b", $time, i0, i1, s, y);
end
initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, i0, i1, s, y);
end
endmodule
```

```
xcelium> run
Simulation time = 0, i0 = 0, i1 =   0, s = 0, y = 0
Simulation time = 10, i0 = 0, i1 =   0, s = 1, y = 0
Simulation time = 20, i0 = 1, i1 =   0, s = 0, y = 0
Simulation time = 30, i0 = 1, i1 =   0, s = 1, y = 1
Simulation time = 40, i0 = 0, i1 =   1, s = 0, y = 1
Simulation time = 50, i0 = 0, i1 =   1, s = 1, y = 0
Simulation time = 60, i0 = 1, i1 =   1, s = 0, y = 1
Simulation time = 70, i0 = 1, i1 =   1, s = 1, y = 1
xmsim: *W,RNQUIE: Simulation is complete.
```



P13). Gate level code for 2x1 mux using nand

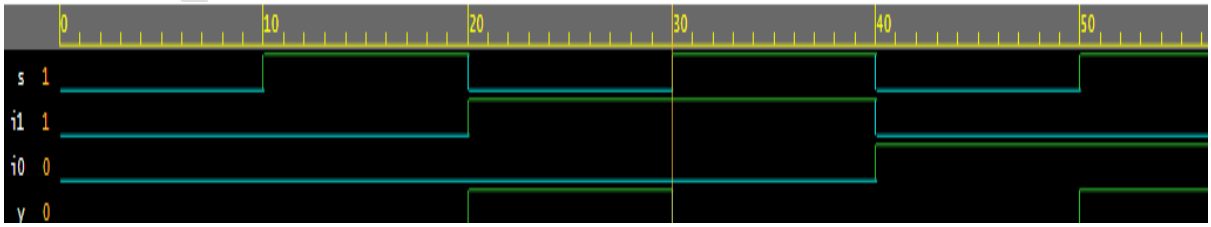
Design code:

```
module mux21nand(i1, i0, s, y);
  input i1, i0,s;
  output y;
  wire w1,w2,w3;

  nand n1(w1,i0,s);
  nand n2(w2,s);
  nand n3(w3, i1);
  nand n4(y,w1,w3);

endmodule
```

Test bench code:

```
module muxnand_test;
  reg i1, i0, s;
  wire y;
  muxnand dut(.i1(i1), .i0(i0), .s(s), .y(y));

  initial begin
        i1=0; i0=0; s=0;
   #10 i1=0; i0=0; s=1;
   #10 i1=0; i0=1; s=0;
   #10 i1=0; i0=1; s=1;
   #10 i1=1; i0=0; s=0;
   #10 i1=1; i0=0; s=1;
   #10 i1=1; i0=1; s=0;
   #10 i1=1; i0=1; s=1;
  end

initial begin
  $monitor("Simulation time = %0t, i0 = %b, i1 =
%b, s = %b, y = %b", $time, i0, i1, s, y);
end
initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, i0, i1, s, y);
end
endmodule
```

```
xcelium> run
Simulation time = 0, i0 = 0, i1 =   0, s = 0, y = 0
Simulation time = 10, i0 = 0, i1 =   0, s = 1, y = 0
Simulation time = 20, i0 = 1, i1 =   0, s = 0, y = 0
Simulation time = 30, i0 = 1, i1 =   0, s = 1, y = 1
Simulation time = 40, i0 = 0, i1 =   1, s = 0, y = 1
Simulation time = 50, i0 = 0, i1 =   1, s = 1, y = 0
Simulation time = 60, i0 = 1, i1 =   1, s = 0, y = 1
Simulation time = 70, i0 = 1, i1 =   1, s = 1, y = 1
xmsim: *W,RNQUIE: Simulation is complete.
```



P14). Gate level code for 2x4 decoder

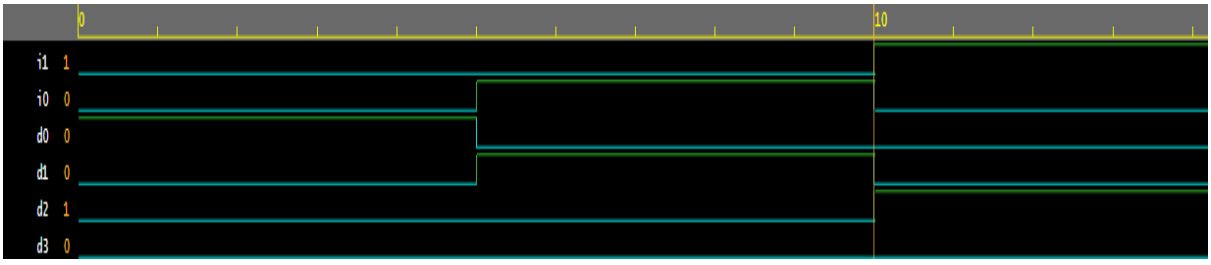
Design code:

```
module decoder24(i1, i0, d0, d1, d2, d3);
input i1, i0;
output d0, d1, d2, d3;
wire w1, w2;

        not n1(w1, i0);
        not n2(w2, i1);
        and a1(d0, w2, w1);
        and a2(d1, w2, i0);
        and a3(d2, i1, w1);
        and a4(d3, i1, i0);

endmodule
```

Test bench code:

```
module decoder24_test;
  reg i1, i0;
  wire d0, d1, d2, d3;

  decoder24 dut(.i1(i1), .i0(i0), .d0(d0), .d1(d1),
.d2(d2), .d3(d3));

  initial begin
    i1=0; i0=0;
    #5 i1=0; i0=1;
    #5 i1=1; i0=0;
    #5 i1=1; i0=1;
  end

  initial begin
    $monitor("Simulationtime=%0t, i1=%b, i0=%b,
d0=%b, d1=%b, d2=%b, d3=%b", $time, i1, i0,
d0,d1, d2,d3);
  end

  initial begin
     $dumpfile("dump.vcd");
     $dumpvars(0, i1, i0, d0,d1,d2,d3);
   end
endmodule
```

```
Simulationtime=0, i1=0, i0=0, d0=1, d1=0, d2=0, d3=0
Simulationtime=5, i1=0, i0=1, d0=0, d1=1, d2=0, d3=0
Simulationtime=10, i1=1, i0=0, d0=0, d1=0, d2=1, d3=0
Simulationtime=15, i1=1, i0=1, d0=0, d1=0, d2=0, d3=1
xmsim: *W,RNQUIE: Simulation is complete.
```



P15). Data flow code for 2x4 decoder

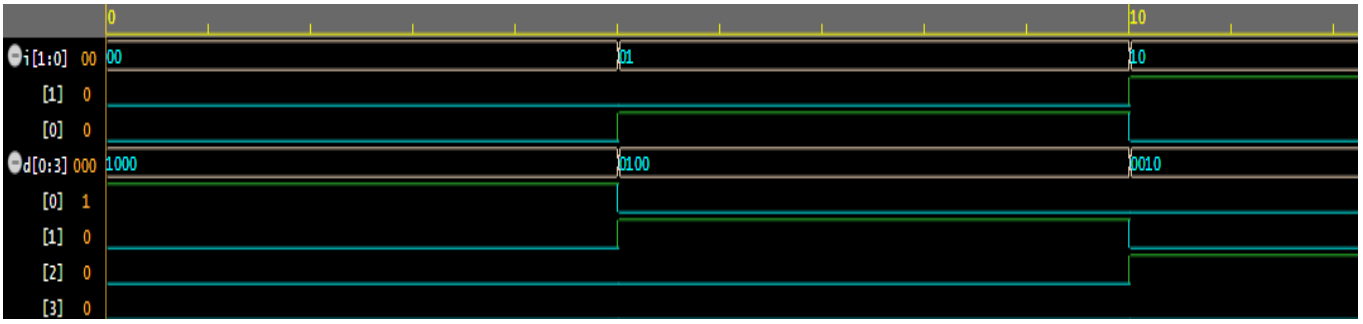
Design code:

```
module decoder24(i, d);
  input [1:0]i;
  output [0:3]d;

  assign d[0]= (~i[1]) & (~i[0]);
  assign d[1]= (~i[1]) & (i[0]);
  assign d[2]= (i[1]) & (~i[0]);
  assign d[3]= (i[1]) & (i[0]);

endmodule
```

Test bench code:

```
// Code your testbench here
// or browse Examples
          module decoder24_test;
      reg [1:0]i;
      wire [0:3]d;

      decoder24 dut(.i(i), .d(d));

      initial begin
          i=2'b00;
        #5 i=2'b01;
        #5 i=2'b10;
        #5 i=2'b11;
      end

      initial begin
        $monitor("Simulation time=%0t, i=%b,
d=%b", $time, i, d);
      end

      initial begin
        $dumpfile("dump.vcd");
        $dumpvars(0, i, d);
      end
    endmodule
```

```
xcelium> run
Simulation time=0, i=00, d=1000
Simulation time=5, i=01, d=0100
Simulation time=10, i=10, d=0010
Simulation time=15, i=11, d=0001
xmsim: *W,RNQUIE: Simulation is complete.
```



P16). Gate level Even parity detector using 2x1 mux

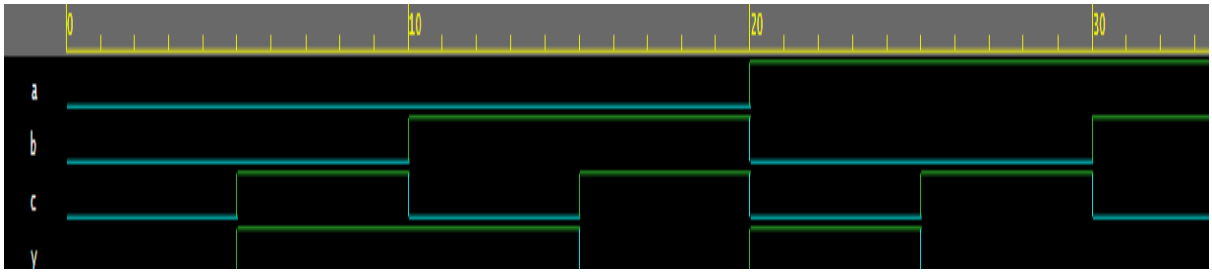
Design code:

```
module EPUMUX(a,b,c,y);
  input a,b,c;
  output y;
  wire w1, w2, w3;
  wire [1:0]i;

  xor x1(i[1], b, c);
  xnor x2(i[0], b, c);
  not n1(w1, a);
  and a1(w2,w1,i[1]);
  and a2(w3, a, i[0]);
  or o1(y, w2, w3);

endmodule
```

Test bench code:

```
// Code your testbench here
// or browse Examples
module EPUMUX_test;
  reg a,b,c;
  wire y;

  EPUMUX dut(a,b,c,y);

  initial begin
      a=0; b=0; c=0;
    #5         c=1;
    #5     b=1; c=0;
    #5         c=1;
    #5 a=1; b=0; c=0;
    #5         c=1;
    #5     b=1; c=0;
    #5         c=1;
  end

  initial begin
    $monitor("simulation time=%0t, a=%b, b=%b,
c=%b, y=%b", $time, a, b, c,y);
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, a, b, c, y);
  end
endmodule
```

```
simulation time=0, a=0, b=0, c=0, y=0
simulation time=5, a=0, b=0, c=1, y=1
simulation time=10, a=0, b=1, c=0, y=1
simulation time=15, a=0, b=1, c=1, y=0
simulation time=20, a=1, b=0, c=0, y=1
simulation time=25, a=1, b=0, c=1, y=0
simulation time=30, a=1, b=1, c=0, y=0
simulation time=35, a=1, b=1, c=1, y=1
xmsim: *W,RNQUIE: Simulation is complete.
```



P17). Data flow Even parity detector using 2x1 mux

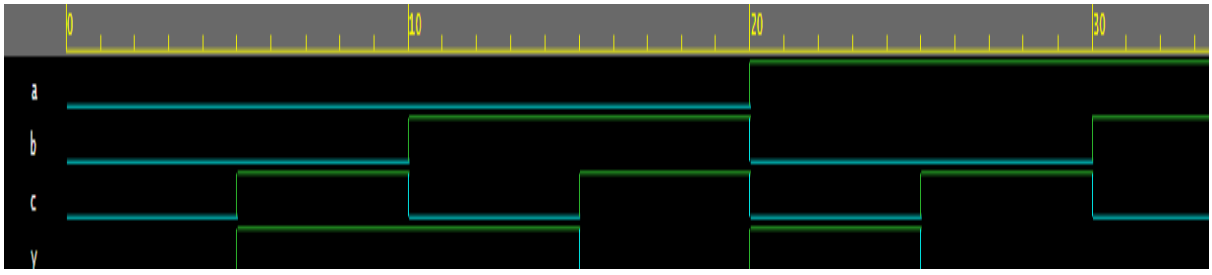
Design code:

```
module EPUMUX(a,b,c,y);
  input a,b,c;
  output y;

  assign y= ((~a)&(b^c))+((a)&(~(b^c)));

endmodule
```

Test bench code:

```
// Code your testbench here
// or browse Examples
module EPUMUX_test;
  reg a,b,c;
  wire y;

  EPUMUX dut(a,b,c,y);

  initial begin
      a=0; b=0; c=0;
    #5          c=1;
    #5      b=1; c=0;
    #5          c=1;
    #5 a=1; b=0; c=0;
    #5          c=1;
    #5      b=1; c=0;
    #5          c=1;
  end

  initial begin
    $monitor("simulation time=%0t, a=%b, b=%b,
c=%b, y=%b", $time, a, b, c,y);
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, a, b, c, y);
  end
endmodule
```

```
simulation time=0, a=0, b=0, c=0, y=0
simulation time=5, a=0, b=0, c=1, y=1
simulation time=10, a=0, b=1, c=0, y=1
simulation time=15, a=0, b=1, c=1, y=0
simulation time=20, a=1, b=0, c=0, y=1
simulation time=25, a=1, b=0, c=1, y=0
simulation time=30, a=1, b=1, c=0, y=0
simulation time=35, a=1, b=1, c=1, y=1
xmsim: *W,RNQUIE: Simulation is complete.
```



P18). Code for 2x1 mux using ternary operator

Design code:

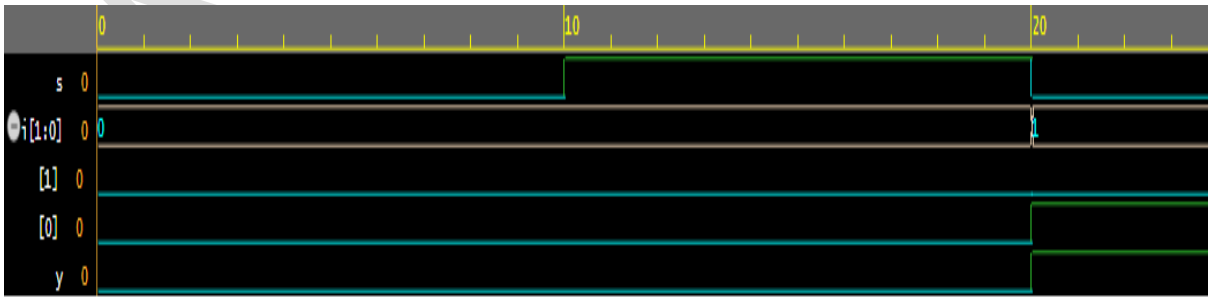
```
module mux21 (i, s, y);
  input [1:0]i;
  input s;
  output y;

  assign y= (s==1'b0) ? i[0] : i[1];

endmodule
```

Test bench code:

```
module mux21_test;
  reg [1:0]i;
  reg s;
  wire y;

  mux21 dut(.i(i), .s(s), .y(y));

  initial begin
        {i, s} = 0;
    #10 {i, s} = 1;
    #10 {i, s} = 2;
    #10 {i, s} = 3;
  end
  initial begin
    $monitor("sim time=%0t, i=%b, s=%b, y=%b",
$time, i, s, y);
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, i, s, y);
  end
endmodule
```

OUTPUT:

```
sim time=0, i=00, s=0, y=0
sim time=10, i=00, s=1, y=0
sim time=20, i=01, s=0, y=1
sim time=30, i=01, s=1, y=0
```



P19). Full adder using half adder(structural)

Design code:

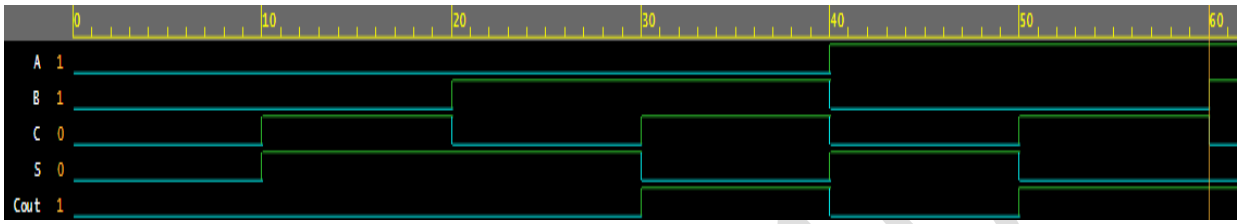
```
//Half adder
module HA(A, B, S, C);
input A, B;
output S, C;
xor x1(S, A, B);
and a1(C, A, B);
endmodule


//Full adder
`include "HA.v"
module FA(A, B, C, S, Cout);
  input A, B, C;
  output S, Cout;
  wire w1, w2, w3;
  HA H1(A, B, w1, w2);
  HA H2(w1, C, S, w3);
  assign Cout=w2|w3;
endmodule
```

Test bench code:

```
module FA_TB;
reg A, B, C;
wire S, Cout;
FA dut(.A(A), .B(B), .C(C), .S(S), .Cout(Cout));
initial begin
          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;
end
initial begin
  $monitor("Simulationtime = %0t, A=%b, B=%b,
C=%b, S=%b, Cout=%b", $time, A, B, C, S,
Cout);
end
initial begin
$dumpfile("dump.vcd");
  $dumpvars (0, A, B, C, S, Cout);
end
endmodule
```

OUTPUT:

```
Simulationtime = 0, A=0, B=0, C=0, S=0, Cout=0
Simulationtime = 10, A=0, B=0, C=1, S=1, Cout=0
Simulationtime = 20, A=0, B=1, C=0, S=1, Cout=0
Simulationtime = 30, A=0, B=1, C=1, S=0, Cout=1
Simulationtime = 40, A=1, B=0, C=0, S=1, Cout=0
Simulationtime = 50, A=1, B=0, C=1, S=0, Cout=1
Simulationtime = 60, A=1, B=1, C=0, S=0, Cout=1
Simulationtime = 70, A=1, B=1, C=1, S=1, Cout=1
```



P20). 4-bit adder using full adder and half adder

Design code:

```
//Half adder
module half_adder(A, B, Sum, Carry);
  input A, B;
  output Sum, Carry;
  assign Sum = A ^ B;
  assign Carry = A & B;
endmodule

//Full adder
module full_adder(A, B, Cin, Sum, Cout);
  input A, B, Cin;
  output Sum, Cout;
  wire w1, w2, w3;

  half_adder HA1(.A(A), .B(B), .Sum(w1),
.Carry(w2));
  half_adder HA2(.A(w1), .B(Cin), .Sum(Sum),
.Carry(w3));
  assign Cout = w3| w2;
endmodule

//4 bit adder
`include "ha.v"
`include "fa.v"
module four_bit_adder(A, B, Cin, Sum, Cout);
  input [3:0] A;
  input [3:0] B;
  input Cin;
  output [3:0] Sum;
  output Cout;
```

Test bench code:

```
// Code your testbench here
// or browse Examples
module four_bit_adder_TB;
  reg [3:0] A, B;
  reg Cin;
  wire [3:0] Sum;
  wire Cout;

  four_bit_adder UUT (
    .A(A), .B(B), .Cin(Cin),
    .Sum(Sum), .Cout(Cout)
  );

  initial begin
    for (integer i = 0; i < 512; i = i + 1)
begin
      {A, B, Cin} = i;
      #5;
    end
  end

  initial begin
    $monitor("Time=%0t | A=%b, B=%b,
Cin=%b => Sum=%b, Cout=%b",
             $time, A, B, Cin, Sum, Cout);
  end

initial begin
    $dumpfile("four_bit_adder.vcd");
    $dumpvars(0, A, B, Cin, Sum, Cout);
```

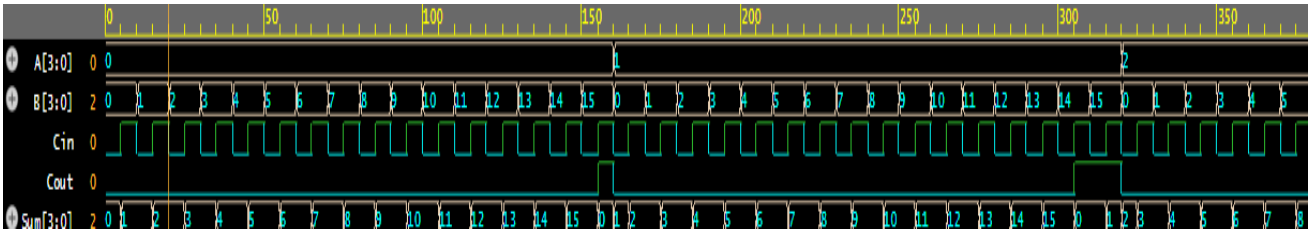
```
  wire [3:1]C;

  full_adder FA0(.A(A[0]), .B(B[0]), .Cin(Cin),
.Sum(Sum[0]), .Cout(C1));
  full_adder FA1(.A(A[1]), .B(B[1]), .Cin(C1),
.Sum(Sum[1]), .Cout(C2));
  full_adder FA2(.A(A[2]), .B(B[2]), .Cin(C2),
.Sum(Sum[2]), .Cout(C3));
  full_adder FA3(.A(A[3]), .B(B[3]), .Cin(C3),
.Sum(Sum[3]), .Cout(Cout));
endmodule
```

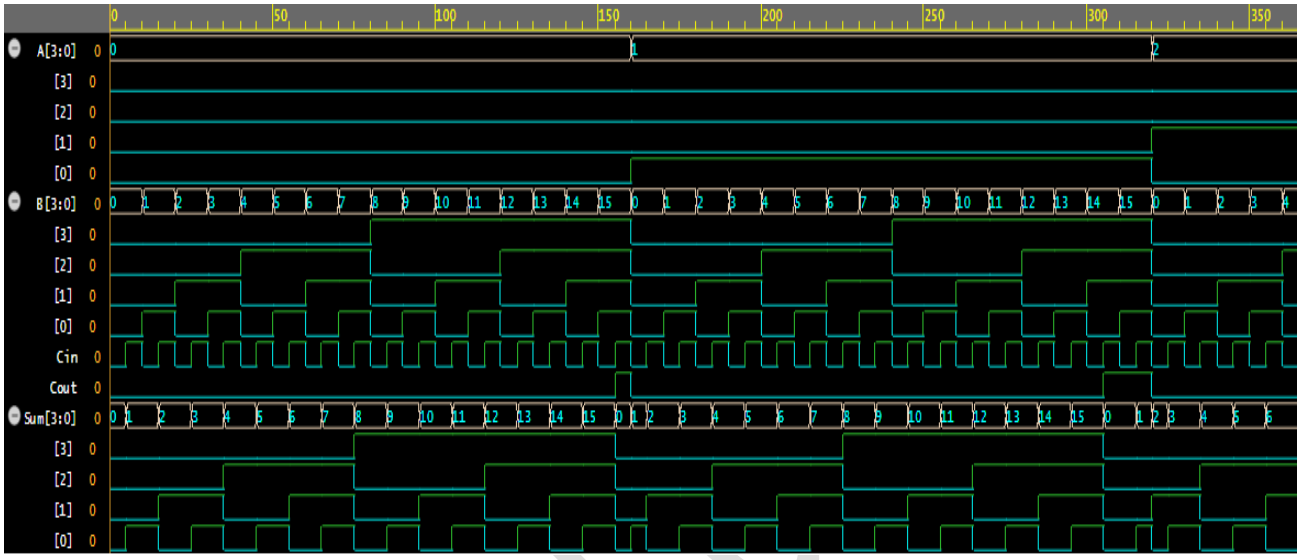
```
  end
endmodule
```

OUTPUT:

```
Time=0 | A=0000, B=0000, Cin=0 => Sum=0000, Cout=0
Time=5 | A=0000, B=0000, Cin=1 => Sum=0001, Cout=0
Time=10 | A=0000, B=0001, Cin=0 => Sum=0001, Cout=0
Time=15 | A=0000, B=0001, Cin=1 => Sum=0010, Cout=0
Time=20 | A=0000, B=0010, Cin=0 => Sum=0010, Cout=0
Time=25 | A=0000, B=0010, Cin=1 => Sum=0011, Cout=0
Time=30 | A=0000, B=0011, Cin=0 => Sum=0011, Cout=0
Time=35 | A=0000, B=0011, Cin=1 => Sum=0100, Cout=0
Time=40 | A=0000, B=0100, Cin=0 => Sum=0100, Cout=0
Time=45 | A=0000, B=0100, Cin=1 => Sum=0101, Cout=0
Time=50 | A=0000, B=0101, Cin=0 => Sum=0101, Cout=0
Time=55 | A=0000, B=0101, Cin=1 => Sum=0110, Cout=0
Time=60 | A=0000, B=0110, Cin=0 => Sum=0110, Cout=0
Time=65 | A=0000, B=0110, Cin=1 => Sum=0111, Cout=0
Time=70 | A=0000, B=0111, Cin=0 => Sum=0111, Cout=0
Time=75 | A=0000, B=0111, Cin=1 => Sum=1000, Cout=0
Time=80 | A=0000, B=1000, Cin=0 => Sum=1000, Cout=0
Time=85 | A=0000, B=1000, Cin=1 => Sum=1001, Cout=0
Time=90 | A=0000, B=1001, Cin=0 => Sum=1001, Cout=0
Time=95 | A=0000, B=1001, Cin=1 => Sum=1010, Cout=0
Time=100 | A=0000, B=1010, Cin=0 => Sum=1010, Cout=0
Time=105 | A=0000, B=1010, Cin=1 => Sum=1011, Cout=0
Time=110 | A=0000, B=1011, Cin=0 => Sum=1011, Cout=0
Time=115 | A=0000, B=1011, Cin=1 => Sum=1100, Cout=0
Time=120 | A=0000, B=1100, Cin=0 => Sum=1100, Cout=0
Time=125 | A=0000, B=1100, Cin=1 => Sum=1101, Cout=0
Time=130 | A=0000, B=1101, Cin=0 => Sum=1101, Cout=0
Time=135 | A=0000, B=1101, Cin=1 => Sum=1110, Cout=0
Time=140 | A=0000, B=1110, Cin=0 => Sum=1110, Cout=0
```

```
Time=2415 | A=1111, B=0001, Cin=1 => Sum=0001, Cout=1
Time=2420 | A=1111, B=0010, Cin=0 => Sum=0001, Cout=1
Time=2425 | A=1111, B=0010, Cin=1 => Sum=0010, Cout=1
Time=2430 | A=1111, B=0011, Cin=0 => Sum=0010, Cout=1
Time=2435 | A=1111, B=0011, Cin=1 => Sum=0011, Cout=1
Time=2440 | A=1111, B=0100, Cin=0 => Sum=0011, Cout=1
Time=2445 | A=1111, B=0100, Cin=1 => Sum=0100, Cout=1
Time=2450 | A=1111, B=0101, Cin=0 => Sum=0100, Cout=1
Time=2455 | A=1111, B=0101, Cin=1 => Sum=0101, Cout=1
Time=2460 | A=1111, B=0110, Cin=0 => Sum=0101, Cout=1
Time=2465 | A=1111, B=0110, Cin=1 => Sum=0110, Cout=1
Time=2470 | A=1111, B=0111, Cin=0 => Sum=0110, Cout=1
Time=2475 | A=1111, B=0111, Cin=1 => Sum=0111, Cout=1
Time=2480 | A=1111, B=1000, Cin=0 => Sum=0111, Cout=1
Time=2485 | A=1111, B=1000, Cin=1 => Sum=1000, Cout=1
Time=2490 | A=1111, B=1001, Cin=0 => Sum=1000, Cout=1
Time=2495 | A=1111, B=1001, Cin=1 => Sum=1001, Cout=1
Time=2500 | A=1111, B=1010, Cin=0 => Sum=1001, Cout=1
Time=2505 | A=1111, B=1010, Cin=1 => Sum=1010, Cout=1
Time=2510 | A=1111, B=1011, Cin=0 => Sum=1010, Cout=1
Time=2515 | A=1111, B=1011, Cin=1 => Sum=1011, Cout=1
Time=2520 | A=1111, B=1100, Cin=0 => Sum=1011, Cout=1
Time=2525 | A=1111, B=1100, Cin=1 => Sum=1100, Cout=1
Time=2530 | A=1111, B=1101, Cin=0 => Sum=1100, Cout=1
Time=2535 | A=1111, B=1101, Cin=1 => Sum=1101, Cout=1
Time=2540 | A=1111, B=1110, Cin=0 => Sum=1101, Cout=1
Time=2545 | A=1111, B=1110, Cin=1 => Sum=1110, Cout=1
Time=2550 | A=1111, B=1111, Cin=0 => Sum=1110, Cout=1
Time=2555 | A=1111, B=1111, Cin=1 => Sum=1111, Cout=1
xmsim: *W,RNQUIE: Simulation is complete.
```





P21). Full adder using half adders(structural)

Design code:

```
//Half adder
module HA(A, B, S, C);
input A, B;
output S, C;
xor x1(S, A, B);
and a1(C, A, B);
endmodule


//Full adder
`include "HA.v"
module FA(A, B, C, S, Cout);
  input A, B, C;
  output S, Cout;
  wire w1, w2, w3;
  HA H1(A, B, w1, w2);
  HA H2(w1, C, S, w3);
  assign Cout=w2|w3;
endmodule
```

Test bench code:

```
module FA_TB;
reg A, B, C;
wire S, Cout;
FA dut(.A(A), .B(B), .C(C), .S(S), .Cout(Cout));
initial begin
          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;
end
initial begin
  $monitor("Simulationtime = %0t, A=%b, B=%b,
C=%b, S=%b, Cout=%b", $time, A, B, C, S,
Cout);
end
initial begin
$dumpfile("dump.vcd");
  $dumpvars (0, A, B, C, S, Cout);
end
endmodule
```

OUTPUT: