Combinational Circuit Background

Please send corrections (or suggestions) to youngwlim@hotmail.com.

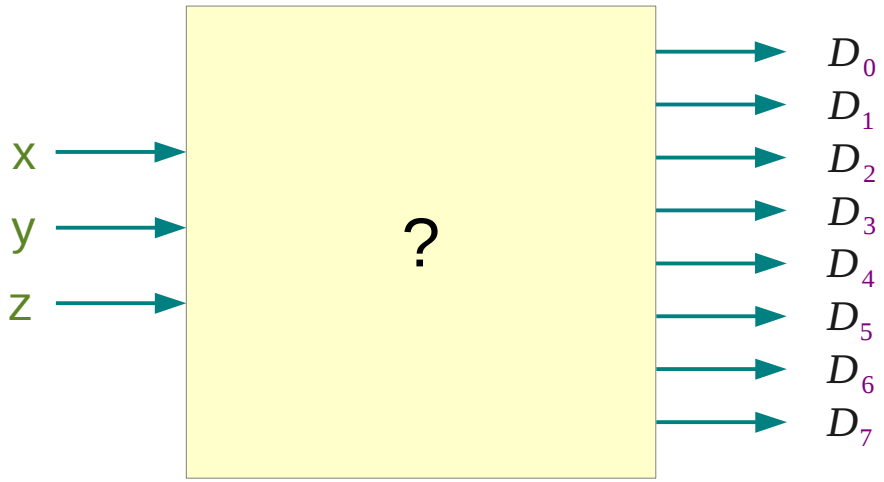
This document was produced by using OpenOffice and Octave.

Decoder

Decoder Truth Table

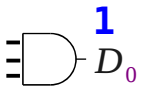
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

inputs output

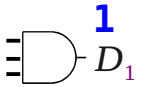


Truth Table and minterms

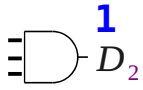
x	y	z	D_0
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



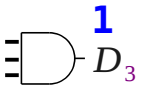
x	y	z	D_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



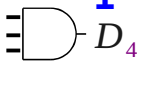
x	y	z	D_2
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



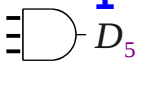
x	y	z	D_3
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0



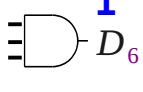
x	y	z	D_4
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0



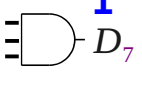
x	y	z	D_5
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



x	y	z	D_6
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0



x	y	z	D_7
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



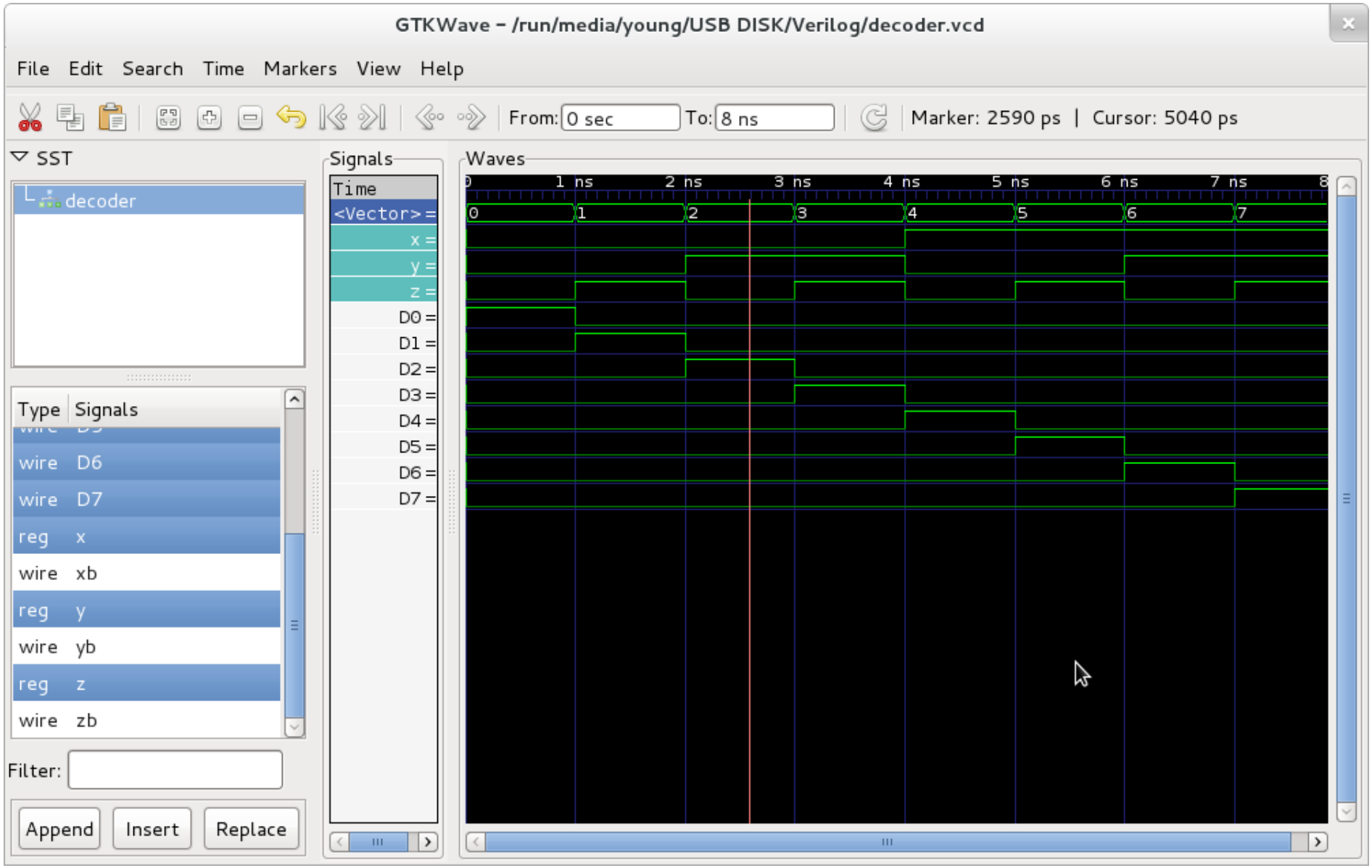
$$\bar{x}\,\bar{y}\,\bar{z} = D_0$$
$$\bar{x}\,\bar{y}\,z = D_1$$
$$\bar{x}\,y\,\bar{z} = D_2$$
$$\bar{x}\,y\,z = D_3$$
$$x\,\bar{y}\,\bar{z} = D_4$$
$$x\,\bar{y}\,z = D_5$$
$$x\,y\,\bar{z} = D_6$$
$$x\,y\,z = D_7$$

decoder.v

```
`timescale 1ns/100ps


module decoder();
  wire  D0, D1, D2, D3, D4, D5, D6, D7;
  reg x, y, z;


  not (xb, x);
  not (yb, y);
  not (zb, z);

  and (D0, xb, yb, zb);  // D0 = minterm m0
  and (D1, xb, yb, z );  // D1 = minterm m1
  and (D2, xb, y , zb);  // D2 = minterm m2
  and (D3, xb, y , z );  // D3 = minterm m3
  and (D4, x , yb, zb);  // D4 = minterm m4
  and (D5, x , yb, z );  // D5 = minterm m5
  and (D6, x , y , zb);  // D6 = minterm m6
  and (D7, x , y , z );  // D7 = minterm m7
```

```
// Testbench Code goes here
initial begin
  $dumpfile("decoder.vcd");
  $dumpvars(0, decoder);

  $monitor ("[x y z] = %b%b%b [D0, D1, D2, D3]
                = %b%b%b%b%b%b%b%b",
               x, y, z, D0, D1, D2, D3, D4, D5, D6, D7);

  x = 0;
  y = 0;
  z = 0;


  #8 $finish;
end



always #4 x = ~x;
always #2 y = ~y;
always #1 z = ~z;


endmodule
```

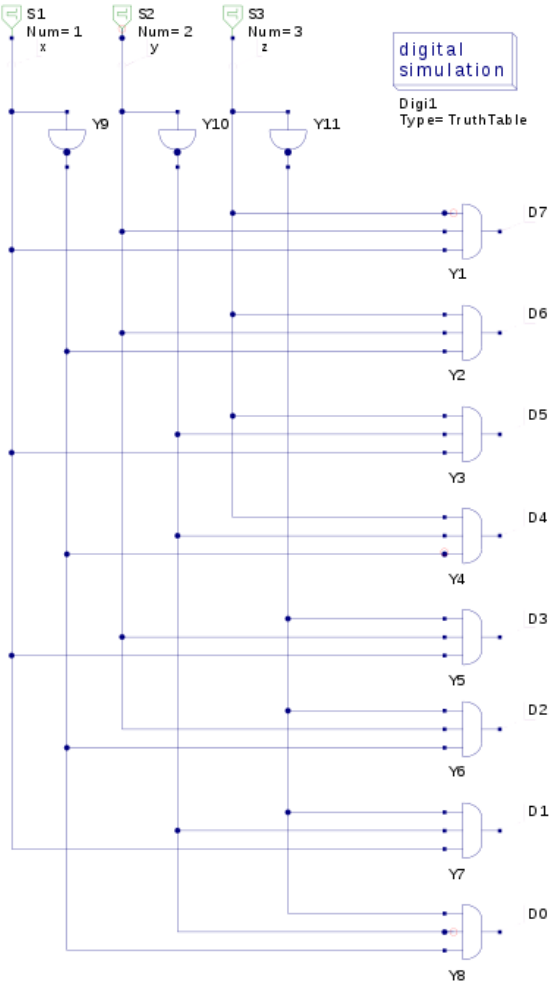
decoder.v simulation



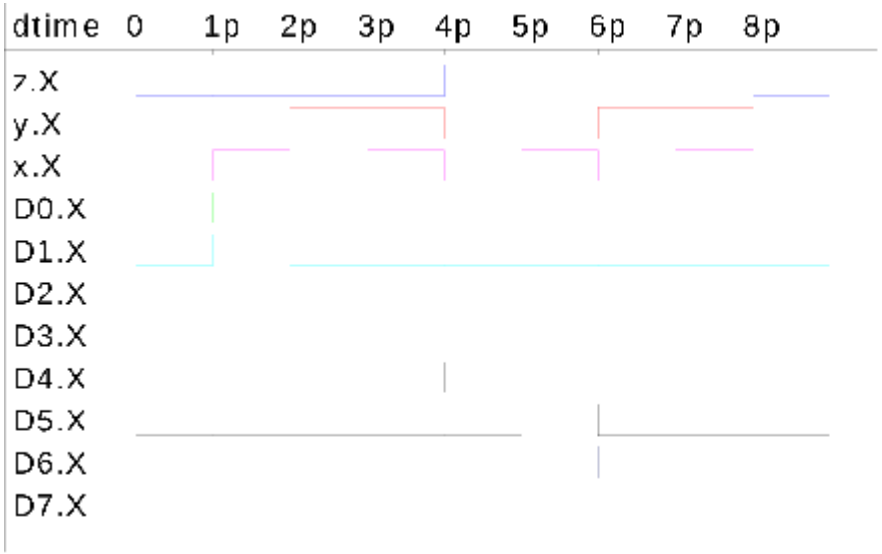
Behavioral Decoder

```
`timescale 1ns/100ps


module decoder(A, D);
  input [2:0] A;
  output [7:0] D;
  reg      [7:0] D;

   always @(A)
      begin
          case (A)
              0: D = 8'b00000001;
              1: D = 8'b00000010;
              2: D = 8'b00000100;
              3: D = 8'b00001000;
              4: D = 8'b00010000;
              5: D = 8'b00100000;
              6: D = 8'b01000000;
              7: D = 8'b10000000;
              default: D = 8b'X;
         endcase
      end

endmodule
```

Decoder Schematic



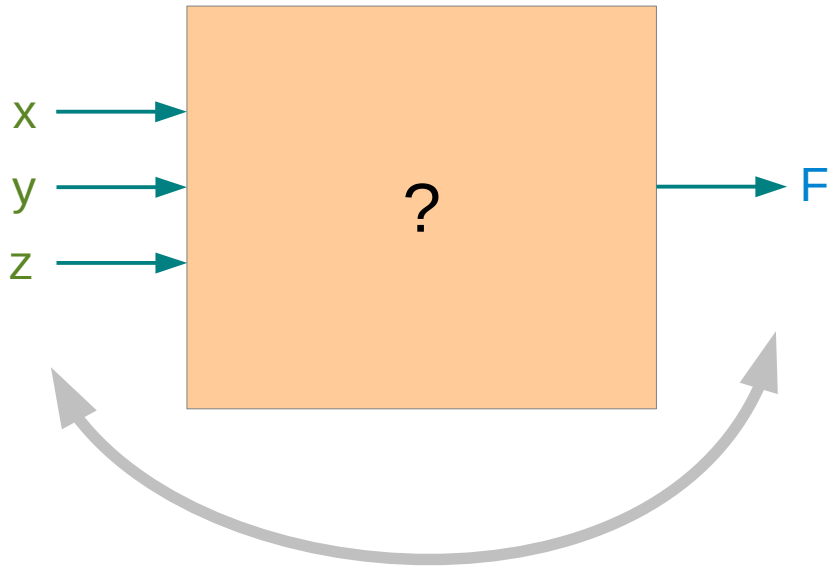
Decoder Wavefpr,



	z.X	y.X	x.X	D0.X	D1.X	D2.X	D3.X	D4.X	D5.X	D6.X	D7.X
0000	0	0	0	1	0	0	0	0	0	0	0
0001	0	0	1	0	1	0	0	0	0	0	0
0010	0	1	0	0	0	1	0	0	0	0	0
0011	0	1	1	0	0	0	1	0	0	0	0
0100	1	0	0	0	0	0	0	1	0	0	0
0101	1	0	1	0	0	0	0	0	1	0	0
0110	1	1	0	0	0	0	0	0	0	1	0
0111	1	1	1	0	0	0	0	0	0	0	1
1000	0	0	0	1	0	0	0	0	0	0	0

Encoder

Truth Table

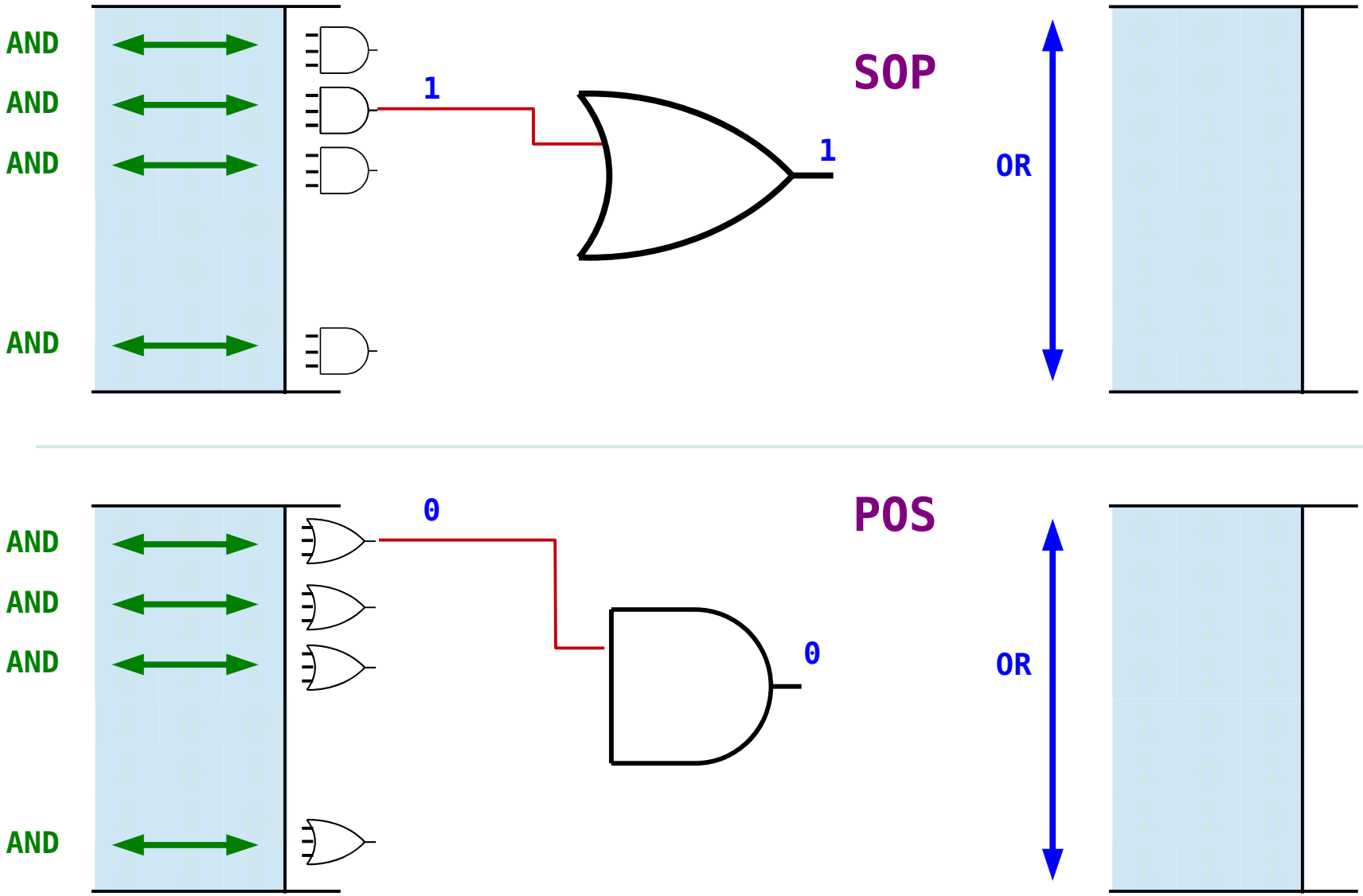


I/O relationship

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

inputs output

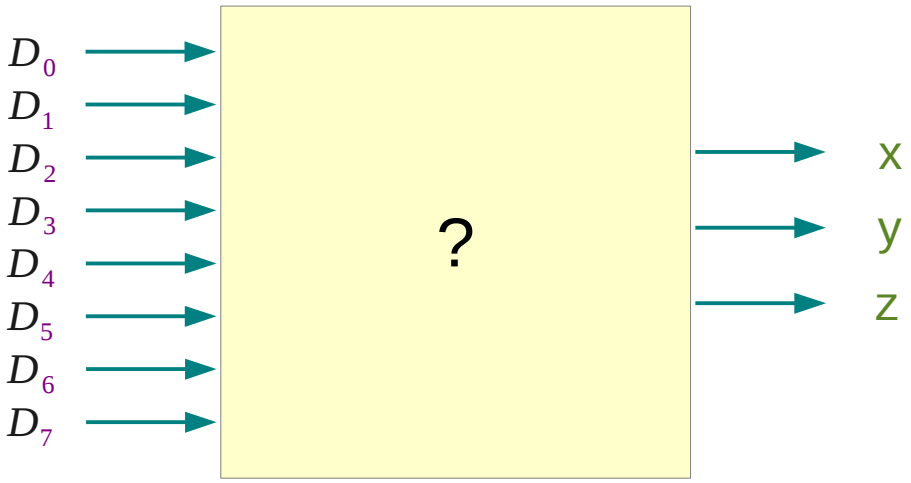
SOP and POS



Truth Table

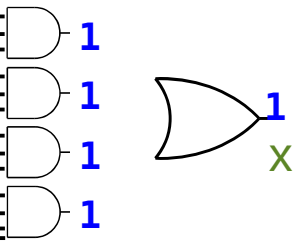
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

inputs output

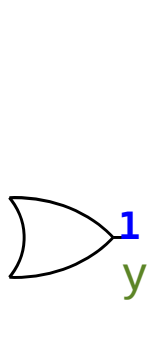


Truth Table and minterms

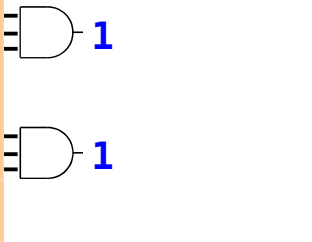
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1



D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	y
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	0	0	1	0	0	0	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1



D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	z
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1



Functions x, y, z

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

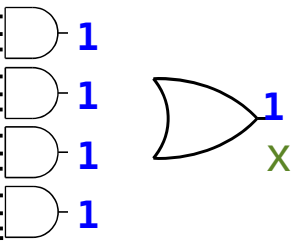
$$\bar{D}_0\bar{D}_1\bar{D}_2\bar{D}_3D_4\bar{D}_5\bar{D}_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2\bar{D}_3\bar{D}_4D_5\bar{D}_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2\bar{D}_3\bar{D}_4\bar{D}_5D_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2\bar{D}_3\bar{D}_4\bar{D}_5\bar{D}_6D_7 = x$$

$$\bar{D}_0\bar{D}_1D_2\bar{D}_3\bar{D}_4\bar{D}_5\bar{D}_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2D_3\bar{D}_4\bar{D}_5\bar{D}_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2\bar{D}_3\bar{D}_4\bar{D}_5D_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2\bar{D}_3\bar{D}_4\bar{D}_5\bar{D}_6D_7 = y$$

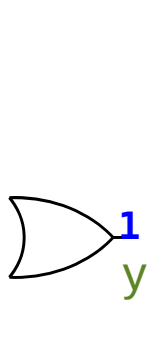
$$\bar{D}_0D_1\bar{D}_2\bar{D}_3\bar{D}_4\bar{D}_5\bar{D}_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2D_3\bar{D}_4\bar{D}_5\bar{D}_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2\bar{D}_3\bar{D}_4D_5\bar{D}_6\bar{D}_7 + \bar{D}_0\bar{D}_1\bar{D}_2\bar{D}_3\bar{D}_4\bar{D}_5\bar{D}_6D_7 = z$$

Another Functions x, y, z

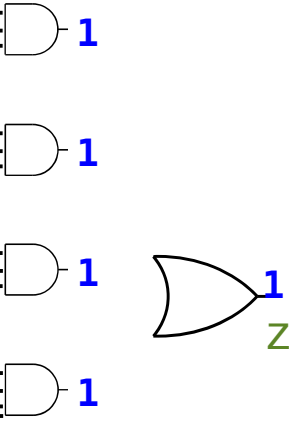
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1



D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	y
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1
0	0	0	1	0	0	0	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	1
0	0	0	0	0	0	0	1	1



D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	z
1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	0	0	1
0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	1



$$D_4 + D_5 + D_6 + D_7 = x$$

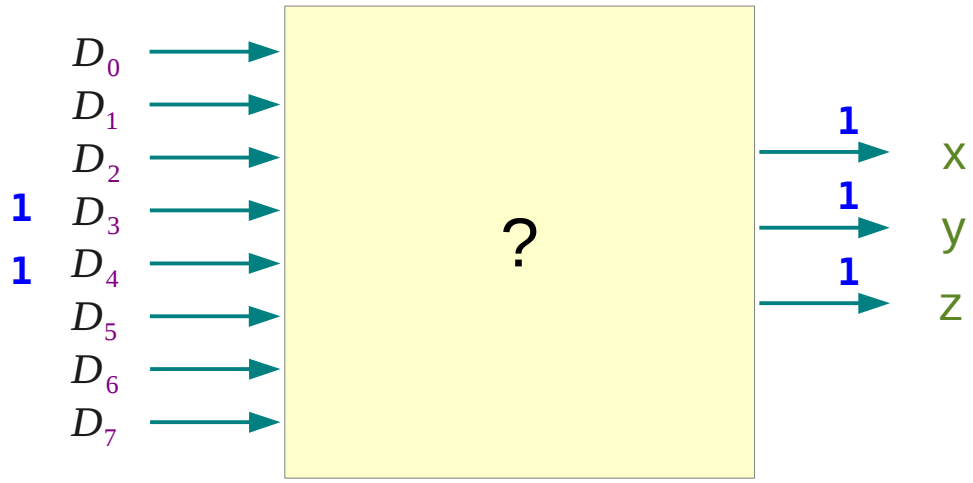
$$D_2 + D_3 + D_6 + D_7 = y$$

$$D_1 + D_3 + D_5 + D_7 = z$$

Problems

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

inputs output



Priority Encoder: Truth Table

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z	v
1	0	0	0	0	0	0	0	0	0	0	1
X	1	0	0	0	0	0	0	0	0	1	1
X	X	1	0	0	0	0	0	0	1	0	1
X	X	X	1	0	0	0	0	0	1	1	1
X	X	X	X	1	0	0	0	1	0	0	1
X	X	X	X	X	1	0	0	1	0	1	1
X	X	X	X	X	X	1	0	1	1	0	1
X	X	X	X	X	X	X	1	1	1	1	1
0	0	0	0	0	0	0	0	X	X	X	0

inputs

output

Priority Encoder: Truth Table

D_0	D_1	D_2	D_3	x	y	v
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1
0	0	0	0	X	X	0

```
if (d3) xy=11
else if (d2) xy = 10;
else if (d1) xy = 01;
else if (d0) xy = 00;
```

D_0	D_1	D_2	D_3	x	y	v
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0	0	1	1
0	0	1	0	1	0	1
0	1	1	0	1	0	1
1	0	1	0	1	0	1
1	1	1	0	1	0	1
0	0	0	1	1	1	1
0	0	1	1	1	1	1
0	1	0	1	1	1	1
0	1	1	1	1	1	1
1	0	0	1	1	1	1
1	0	1	1	1	1	1
1	1	0	1	1	1	1
1	1	1	1	1	1	1
0	0	0	0	X	X	0

D_0	D_1	D_2	D_3	x	y	v
0	0	0	0	X	X	0
0	0	0	1	1	1	1
0	0	1	0	1	0	1
0	0	1	1	1	1	1
0	1	0	0	0	1	1
0	1	0	1	1	1	1
0	1	1	0	1	0	1
0	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	0	1	1	1	1
1	0	1	0	1	0	1
1	0	1	1	1	1	1
1	1	0	0	0	1	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

Priority Encoder: Truth Table

X	1	1	1
0	1	1	1
0	1	1	1
0	1	1	1

$$x = D_2 + D_3$$

X	1	1	0
1	1	1	0
1	1	1	0
0	1	1	0

$$y = D_3 + D_1 \bar{D}_2$$

0	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1

$$v = D_0 + D_1 + D_2 + D_3$$

D_0	D_1	D_2	D_3	x	y	v
0	0	0	0	X	X	0
0	0	0	1	1	1	1
0	0	1	0	1	0	1
0	0	1	1	1	1	1
0	1	0	0	0	1	1
0	1	0	1	1	1	1
0	1	1	0	1	0	1
0	1	1	1	1	1	1
1	0	0	0	0	0	1
1	0	0	1	1	1	1
1	0	1	0	1	0	1
1	0	1	1	1	1	1
1	1	0	0	0	1	1
1	1	0	1	1	1	1
1	1	1	0	1	0	1
1	1	1	1	1	1	1

pencoder.v

```
`timescale 1ns/100ps


module pencoder();
  reg  D0, D1, D2, D3;
  wire P;

  //Invert the sel signals
  not (D2b, D2);

  // 3-input AND gate

  and (P, D1, D2b);

  or (x, D2, D3);
  or (y, D3, P);
  or (v, D0, D1, D2, D3);
```

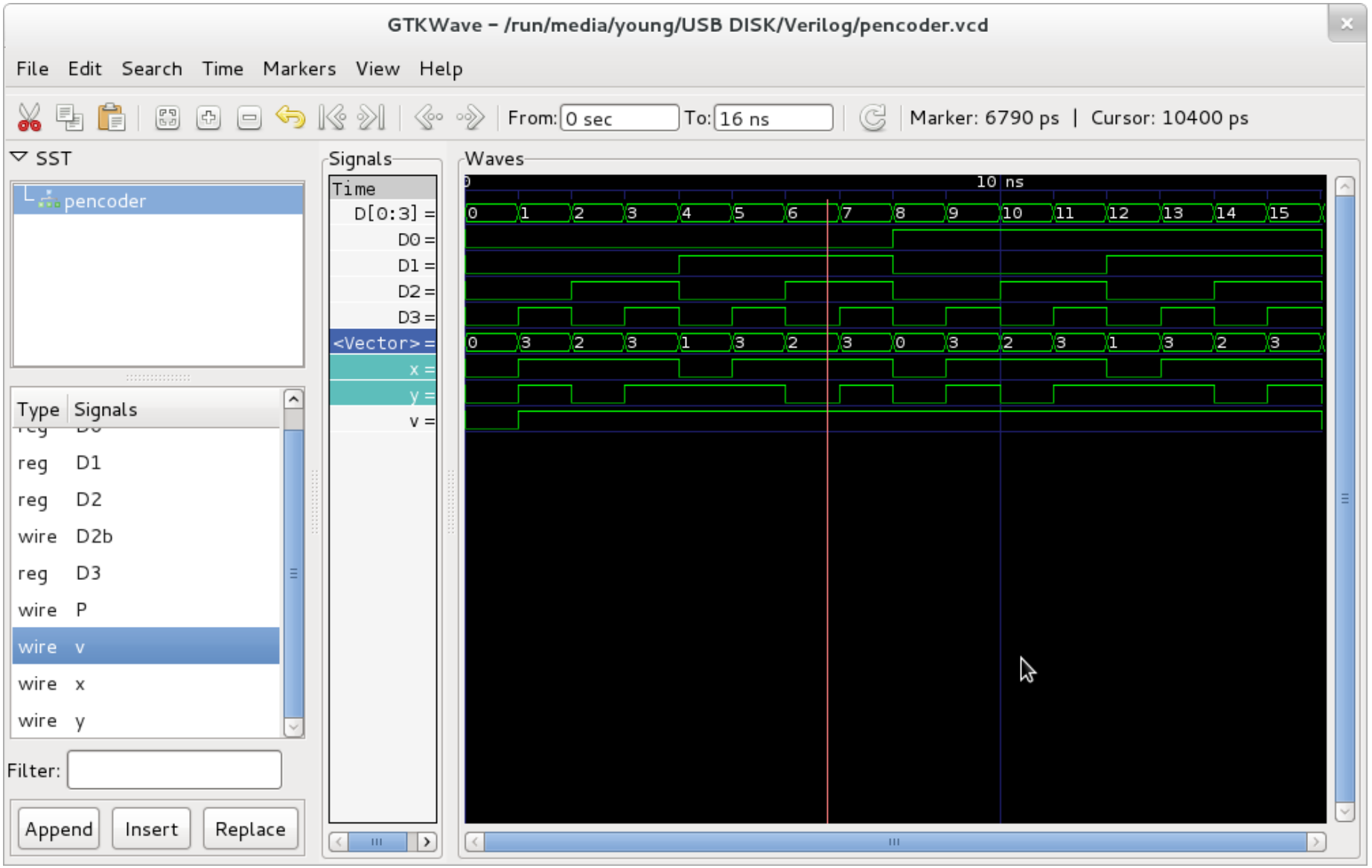
```
// Testbench Code goes here
initial begin
  $dumpfile("pencoder.vcd");
  $dumpvars(0, pencoder);

  $monitor ("[D0, D1, D2, D3] = %b %b %b %b [x y] = %b %b v = %b",
   D0, D1, D2, D3, x, y, v);

  D0 = 0;
  D1 = 0;
  D2 = 0;
  D3 = 0;



  #16 $finish;
end


always #8 D0 = ~D0;
always #4 D1 = ~D1;
always #2 D2 = ~D2;
always #1 D3 = ~D3;


endmodule
```

pencoder.v

```
`timescale 1ns/100ps


module pencoder();
  reg  D0, D1, D2, D3;
  wire P;

  //Invert the sel signals
  not (D2b, D2);

  // 3-input AND gate

  and (P, D1, D2b);

  or (x, D2, D3);
  or (y, D3, P);
  or (v, D0, D1, D2, D3);
```

```
/*
  #1 {D0,D1,D2,D3} = 4'b0000; // 0
  #1 {D0,D1,D2,D3} = 4'b0001; // 1
  #1 {D0,D1,D2,D3} = 4'b0010; // 2
  #1 {D0,D1,D2,D3} = 4'b0011; // 3
  #1 {D0,D1,D2,D3} = 4'b0100; // 4
  #1 {D0,D1,D2,D3} = 4'b0101; // 5
  #1 {D0,D1,D2,D3} = 4'b0110; // 6
  #1 {D0,D1,D2,D3} = 4'b0111; // 7
  #1 {D0,D1,D2,D3} = 4'b1000; // 8
  #1 {D0,D1,D2,D3} = 4'b1001; // 9
  #1 {D0,D1,D2,D3} = 4'b1010; // 10
  #1 {D0,D1,D2,D3} = 4'b1011; // 11
  #1 {D0,D1,D2,D3} = 4'b1100; // 12
  #1 {D0,D1,D2,D3} = 4'b1101; // 13
  #1 {D0,D1,D2,D3} = 4'b1110; // 14
  #1 {D0,D1,D2,D3} = 4'b1111; // 15
*/
```

Priority Encoder: Truth Table



binary encoder

```
module encoder(D, E);

input [7:0] D;
output [2:0] E;
reg [2:0] E;

  always @(D)
    begin
      if (D == 8'b00000001) E = 0;
      else if (D == 8'b00000010) E = 1;
      else if (D == 8'b00000100) E = 2;
      else if (D == 8'b00001000) E = 3;
      else if (D == 8'b00010000) E = 4;
      else if (D == 8'b00100000) E = 5;
      else if (D == 8'b01000000) E = 6;
      else if (D == 8'b10000000) E = 7;
      else E = 3'bX;
    end

endmodule
```

priority encoder

```
module pencoder(D, E);

input [7:0] D;
output [2:0] E;
reg [2:0] E;

  always @(D)
    begin
      if (D[0]) E = 0;
      else if (D[1]) E = 1;
      else if (D[2]) E = 2;
      else if (D[3]) E = 3;
      else if (D[4]) E = 4;
      else if (D[5]) E = 5;
      else if (D[6]) E = 6;
      else if (D[7]) E = 7;
      else E = 3'bX;
    end

endmodule
```

Multiplexer

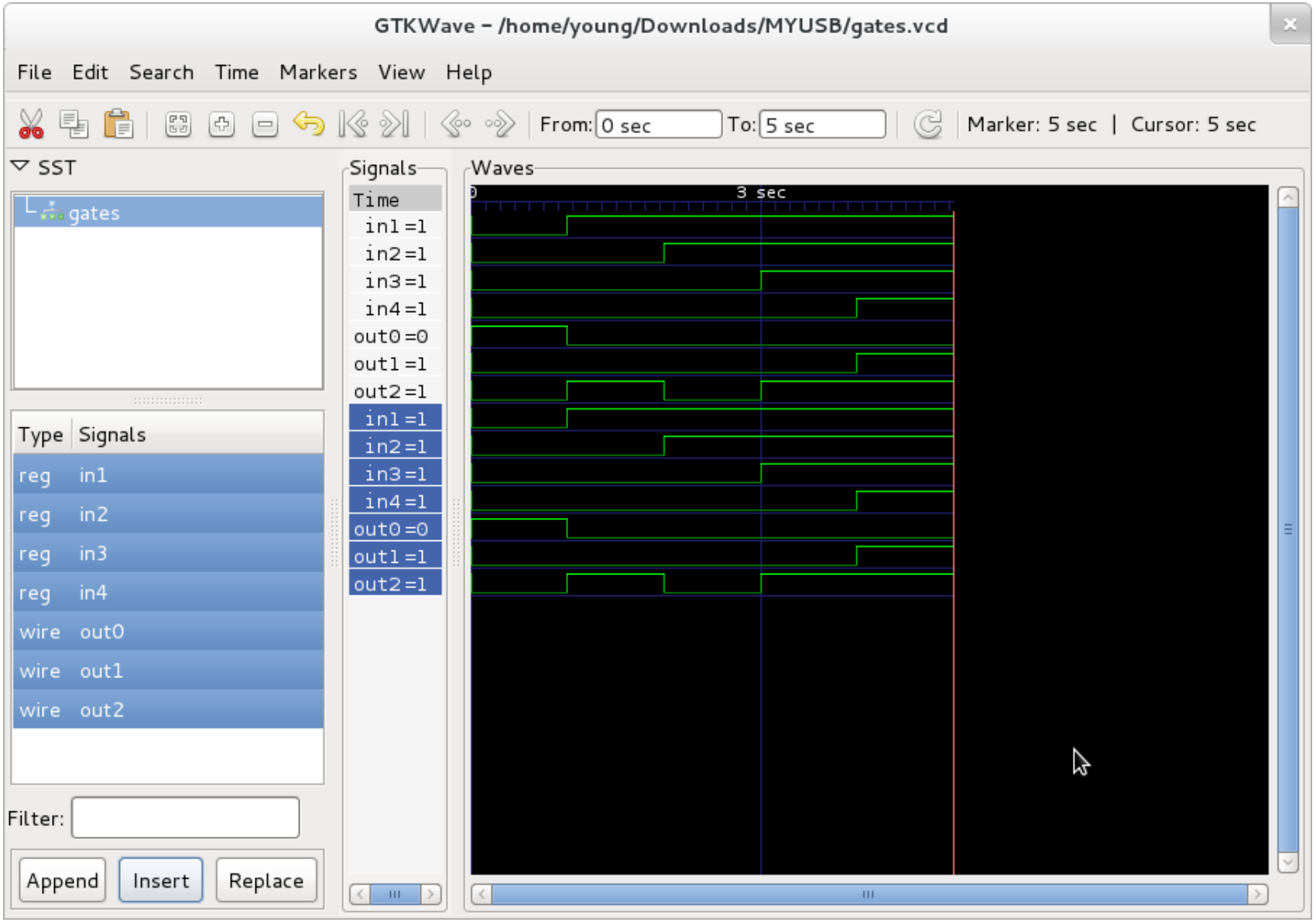
Simple Gates

```
module gates();

  wire out0;
  wire out1;
  wire out2;

  reg  in1, in2, in3, in4;

  not U1 (out0, in1);

  and U2 (out1, in1, in2, in3, in4);

  xor U3 (out2, in1, in2, in3);
```

```
initial begin
  $dumpfile("gates.vcd");
  $dumpvars(0, gates);

  $monitor(
  "in1=%b in2=%b in3=%b in4=%b out0=%b out1=%b out2=%b",
   in1,in2,in3,in4, out0,out1,out2);

  in1 = 0;
  in2 = 0;
  in3 = 0;
  in4 = 0;

#1 in1 = 1;
#1 in2 = 1;
#1 in3 = 1;
#1 in4 = 1;
#1

  $finish;
end

endmodule
```

from www.asic-world.com

Simple Gates



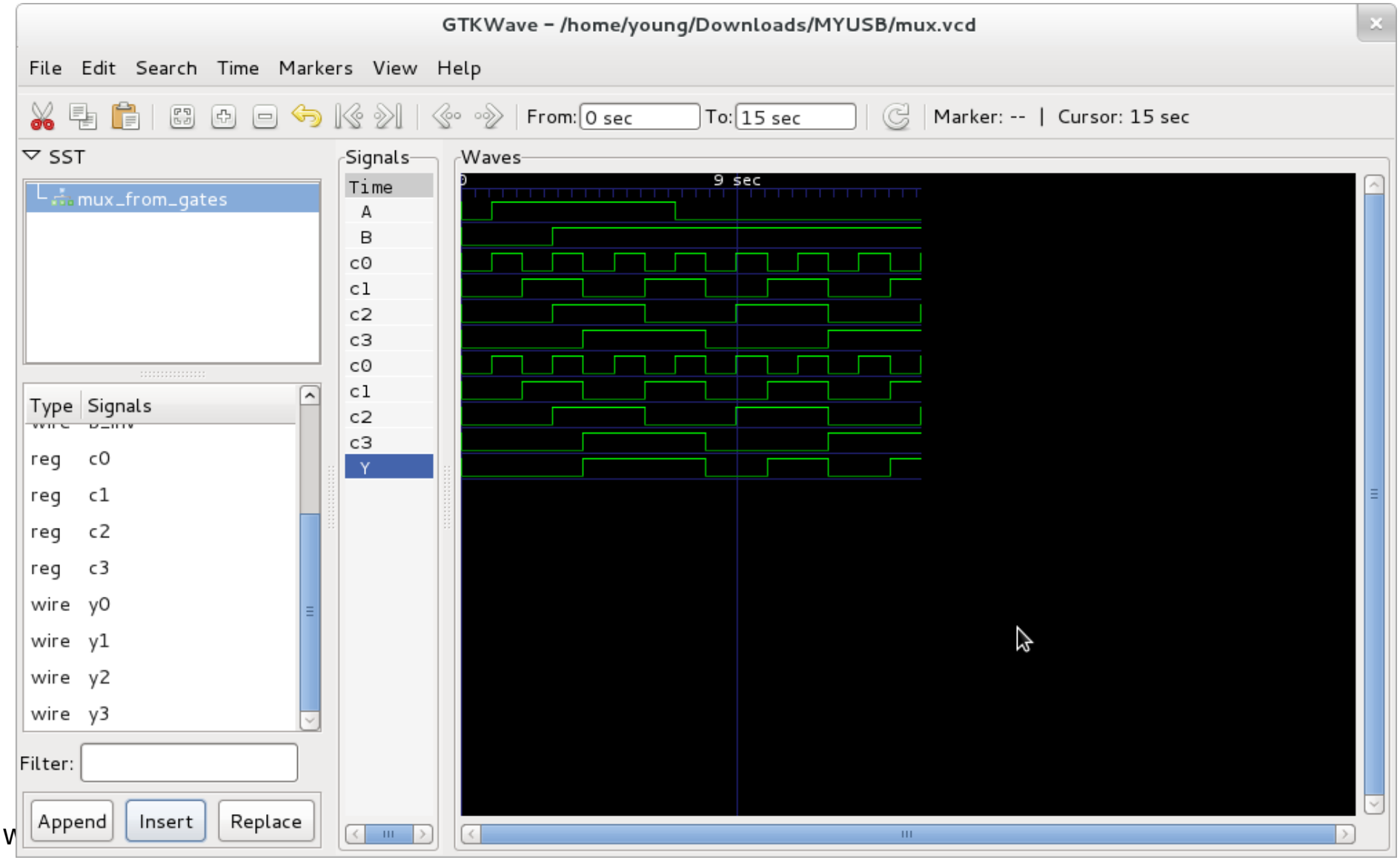
Multiplexer

```
module mux_from_gates ();
  reg c0,c1,c2,c3, A,B;
  wire Y;

  //Invert the sel signals
  not (a_inv, A);
  not (b_inv, B);


  // 3-input AND gate
  and (y0, c0, a_inv, b_inv);  // 00
  and (y1, c1, a_inv, B);      // 01
  and (y2, c2, A,     b_inv);  // 10
  and (y3, c3, A,     B);      // 11


  // 4-input OR gate
  or (Y, y0, y1, y2, y3);
```

```
initial begin
  $dumpfile("mux.vcd");
  $dumpvars(0,mux_from_gates);

  $monitor (
   "c0 = %b c1 = %b c2 = %b c3 = %b A = %b B = %b Y = %b",
   c0, c1, c2, c3, A, B, Y);

  c0 = 0;
  c1 = 0;
  c2 = 0;
  c3 = 0;
  A = 0;
  B = 0;

  #1 A  = 1;
  #2 B  = 1;
  #4 A  = 0;
  #8 $finish;
end

always #1 c0 = ~c0;
always #2 c1 = ~c1;
always #3 c2 = ~c2;
always #4 c3 = ~c3;
endmodule
```

from www.asic-world.com

Multiplexer



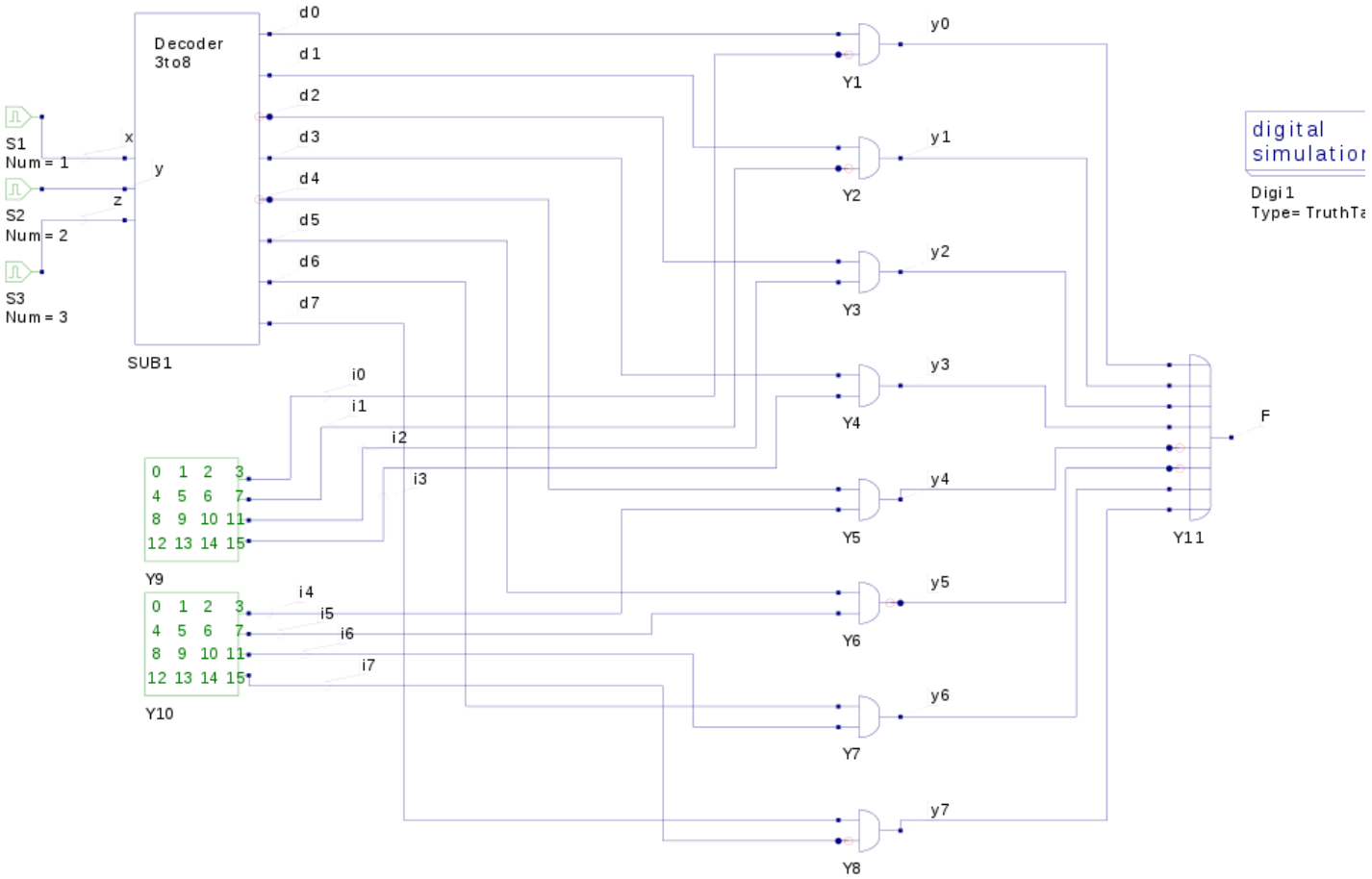
from w

Multiplexer

```
module mux(sel, D0, D1, D2, D3, Y)
  input [1:0] sel;
  input D0, D1, D2, D3;
  output Y;
  reg Y;

  always @(sel or D0 or D1 or D2 or D3)
     case (sel)
         2'b00 : Y = D0;
         2'b01 : Y = D1;
         2'b10 : Y = D2;
         2'b11 : Y = D3;
        default: Y = D0;
     endcase

endmodule
```

from www.asic-world.com

Multiplexer Schematic

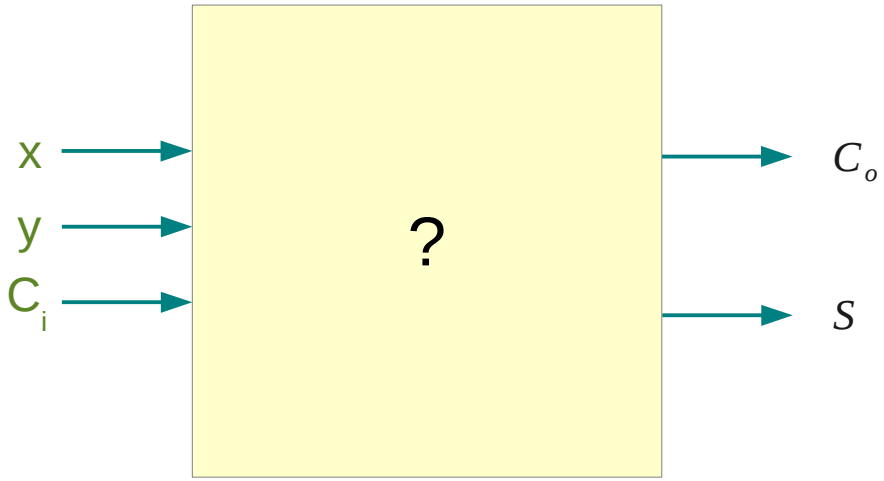


Adder (RCA)

Truth Table

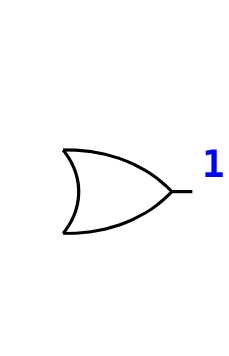
x	y	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

inputs output

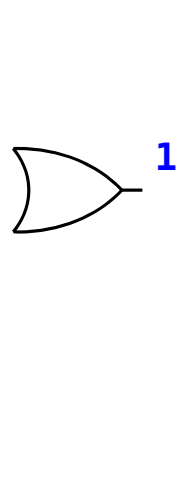


SOP

x	y	C_i	C_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



x	y	C_i	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

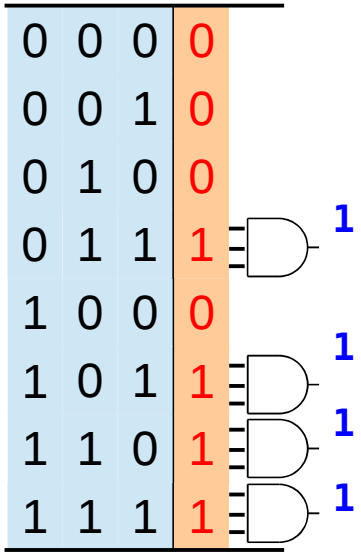


$$C_o = \bar{x}\,y\,C_i + x\,\bar{y}\,C_i + x\,y\,\bar{C}_i + x\,y\,C_i$$

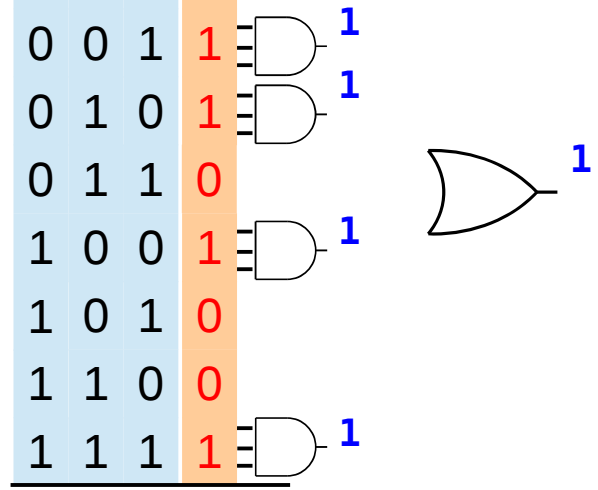
$$S = \bar{x}\,\bar{y}\,C_i + \bar{x}\,y\,\bar{C}_i + x\,\bar{y}\,\bar{C}_i + x\,y\,C_i$$

K-Map

x	y	C_i	C_o
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

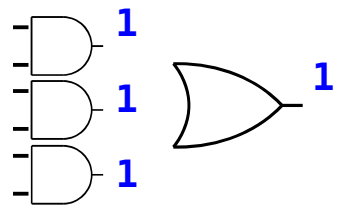


x	y	C_i	S
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



x \ y	0	1
0	0	1
1	1	1

C_i



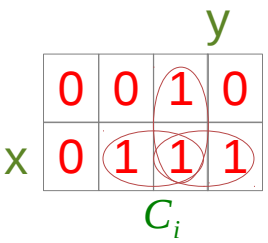
x \ y	0	1
0	0	1
1	1	0

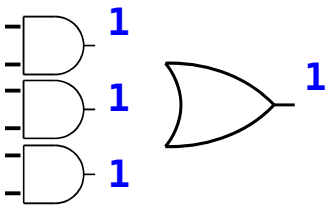
C_i

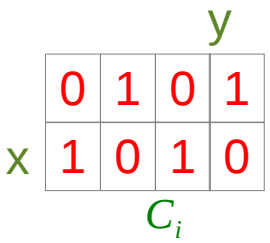
$$C_o = y C_i + x C_i + x y$$

$$S = \bar{x}\,\bar{y}\,C_i + \bar{x}\,y\,\bar{C}_i + x\,\bar{y}\,\bar{C}_i + x\,y\,C_i$$

Boolean Algebra







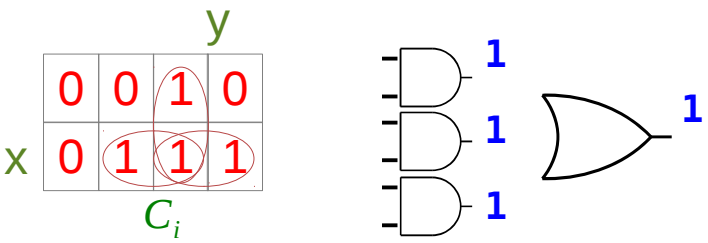
$$C_o = y C_i + x C_i + x y$$

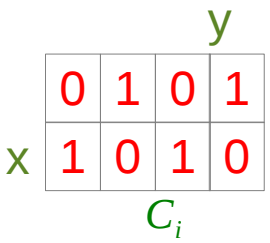
$$\begin{aligned} C_o &= (x + y) C_i + x y \\ &= (\bar{x} y + x \bar{y} + xy) C_i + x y \\ &= (\bar{x} y + x \bar{y}) C_i + xy (C_i + 1) \\ &= (x \oplus y) C_i + xy \end{aligned}$$

$$S = \bar{x}\,\bar{y}\,C_i + \bar{x}\,y\,\bar{C}_i + x\,\bar{y}\,\bar{C}_i + x\,y\,C_i$$

$$\begin{aligned} S &= (\bar{x}\,\bar{y} + xy) C_i + (\bar{x}\,y + x\,\bar{y}) \bar{C}_i \\ &= \overline{(x \oplus y)} C_i + (x \oplus y) \bar{C}_i \\ &= (x \oplus y) \oplus C_i \end{aligned}$$

Boolean Algebra



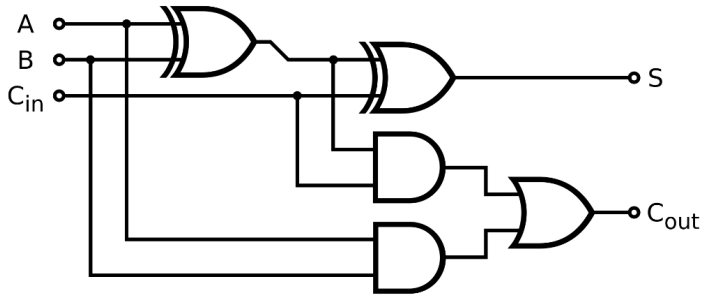


$$C_o = y C_i + x C_i + x y$$

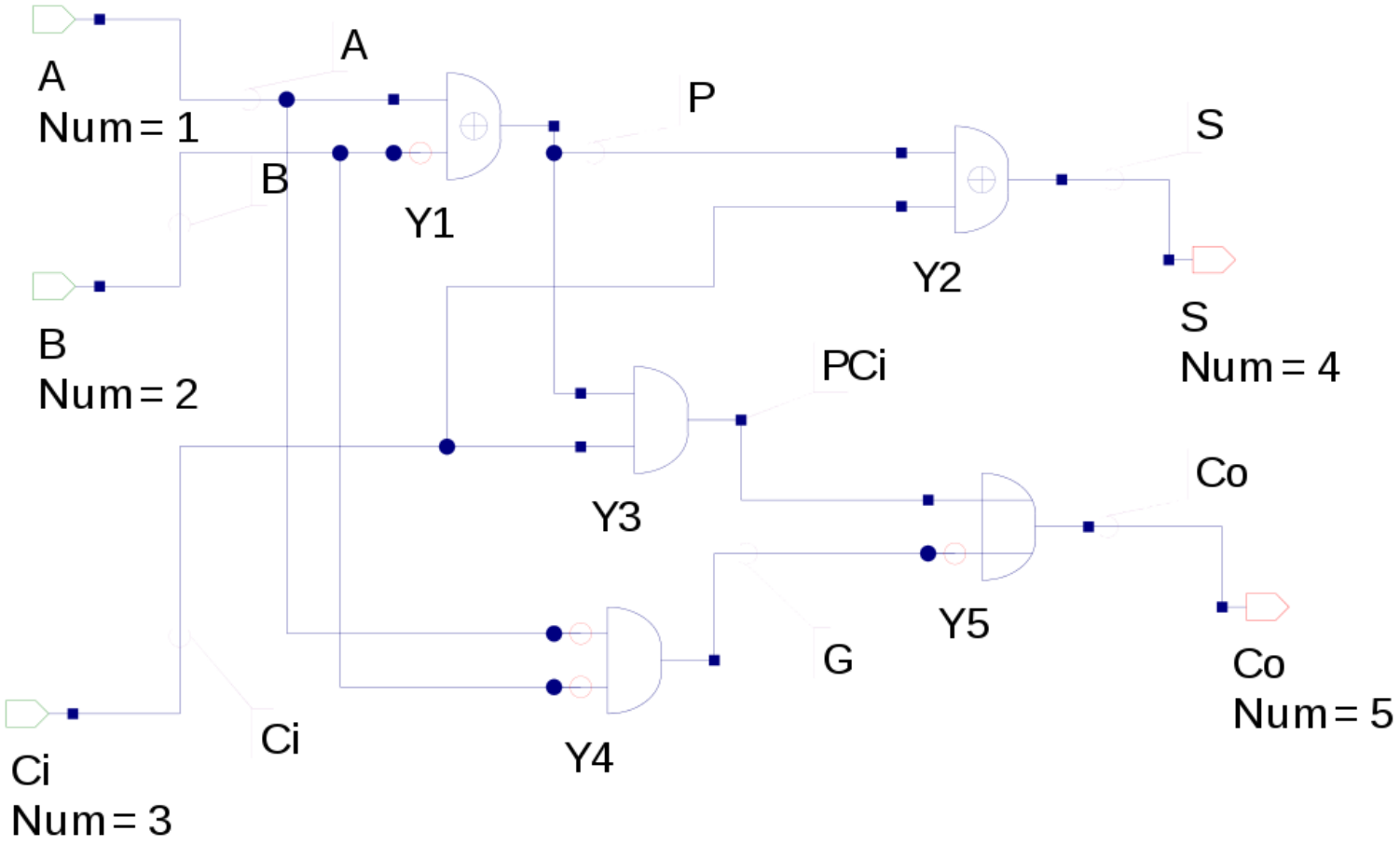
$$\begin{aligned} C_o &= (x + y) C_i + x y \\ &= (\bar{x} y + x \bar{y} + xy) C_i + x y \\ &= (\bar{x} y + x \bar{y}) C_i + xy (C_i + 1) \\ &= (x \oplus y) C_i + xy \end{aligned}$$

$$S = \bar{x}\,\bar{y}\,C_i + \bar{x}\,y\,\bar{C}_i + x\,\bar{y}\,\bar{C}_i + x\,y\,C_i$$

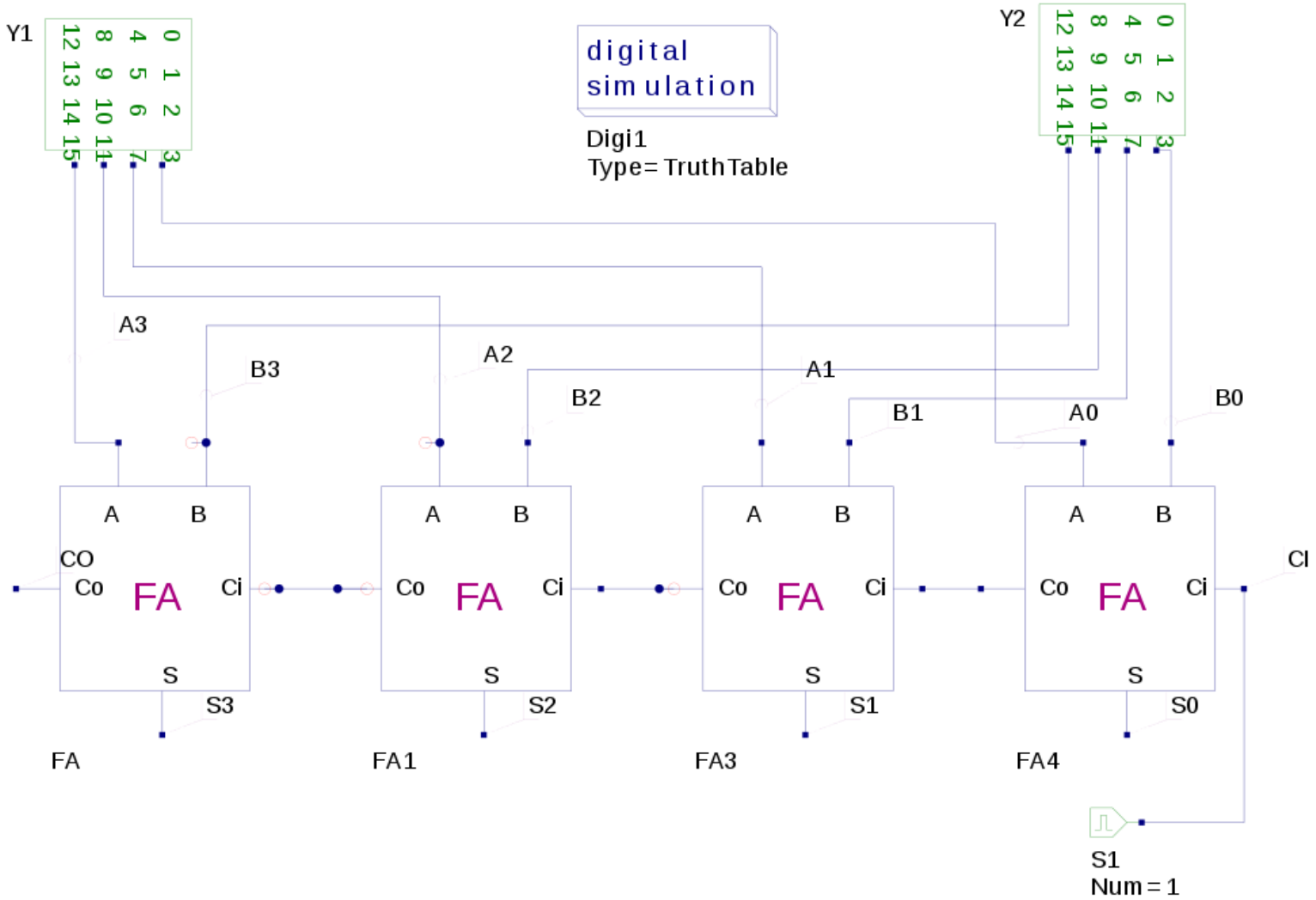
$$\begin{aligned} S &= (\bar{x}\,\bar{y} + xy) C_i + (\bar{x}\,y + x\,\bar{y}) \bar{C}_i \\ &= \overline{(x \oplus y)} C_i + (x \oplus y) \bar{C}_i \\ &= (x \oplus y) \oplus C_i \end{aligned}$$



Full Adder in Qucs



4-Bit Adder in Qucs



adder_tb.v

```
`timescale 1ns/100ps

module adder_tb;

reg signed [3:0] i, j, k, flag;

wire signed [3:0] S;


adder4 A4 (i, j, 1'b0, Co, S);

initial
begin
    $dumpfile("test.vcd");
    $dumpvars(0, adder_tb);
end



initial
begin
  i = -4'd8;
  j = -4'd8;
  repeat (16)
  begin
    repeat (16)
    begin

      k = i + j;

      flag = {i[3], j[3], k[3]};
      #1

      if ((flag == 3'b110) || (flag == 3'b001))
              $display("i=%d j=%d k=%d S=%d OV", i , j, k, S);
      else
              $display("i=%d j=%d k=%d S=%d", i , j, k, S);


      j = j + 4'b1;
    end
    i = i + 4'b1;
  end

end // initial begin

endmodule
```

adder4.v, fa_gate.v fa_flow.v

adder4.v

```
module adder4(A, B, Ci, Co, S);
  input [3:0] A, B;
  input Ci;

  output Co;
  output [3:0] S;

  wire[3:1] C;

  fa  fa0 (A[0], B[0], Ci,   C[1], S[0]);
  fa  fa1 (A[1], B[1], C[1], C[2], S[1]);
  fa  fa2 (A[2], B[2], C[2], C[3], S[2]);
  fa  fa3 (A[3], B[3], C[3], Co,   S[3]);


endmodule
```

iverilog adder_tb.v adder4.v fa_gate.v
vvp a.out

iverilog adder_tb.v adder4.v fa_flow.v
vvp a.out

fa_gate.v

```
module fa(a, b, c, Co, S);
  input a, b, c;
  output Co, S;

  wire P, S, G, PC;

  xor #0.1 U1 (P, a, b);
  xor #0.1 U2 (S, P, c);
  and #0.1 U3 (G, a, b);
  and #0.1 U4 (PC, P, c);
  or  #0.1 U5 (Co, G, PC);
endmodule
```

fa_flow.v

```
module fa(a, b, c, Co, S);
  input a, b, c;
  output Co, S;

  wire P, S, G, PC;

   assign #0.1 P = a ^ b;
   assign #0.1 S = P ^ c;
   assign #0.1 G = a & b;
   assign #0.1 PC = P & c;
   assign #0.1 Co = G | PC;
endmodule
```

adder4.v, fa_behav.v, fa_arith.v

adder4.v

```
module adder4(A, B, Ci, Co, S);
  input [3:0] A, B;
  input Ci;

  output Co;
  output [3:0] S;

  wire[3:1] C;

  fa  fa0 (A[0], B[0], Ci,   C[1], S[0]);
  fa  fa1 (A[1], B[1], C[1], C[2], S[1]);
  fa  fa2 (A[2], B[2], C[2], C[3], S[2]);
  fa  fa3 (A[3], B[3], C[3], Co,   S[3]);


endmodule
```

iverilog adder_tb.v adder4.v fa_behav.v
vvp a.out

iverilog adder_tb.v adder4.v fa_arith.v
vvp a.out

fa_behav.v

```
module fa(a, b, c, Co, S);
  input a, b, c;
  output Co, S;

  reg P, S, G, PC, Co;

  always @(a or b)
    #0.1 P = a ^ b;

  always @(P or c)
    #0.1 S = P ^ c;

  always @(a or b)
    #0.1 G = a & b;

  always @(P or c)
    #0.1 PC = P & c;

  always @(G or PC)
    #0.1 Co = G | PC;


endmodule
```

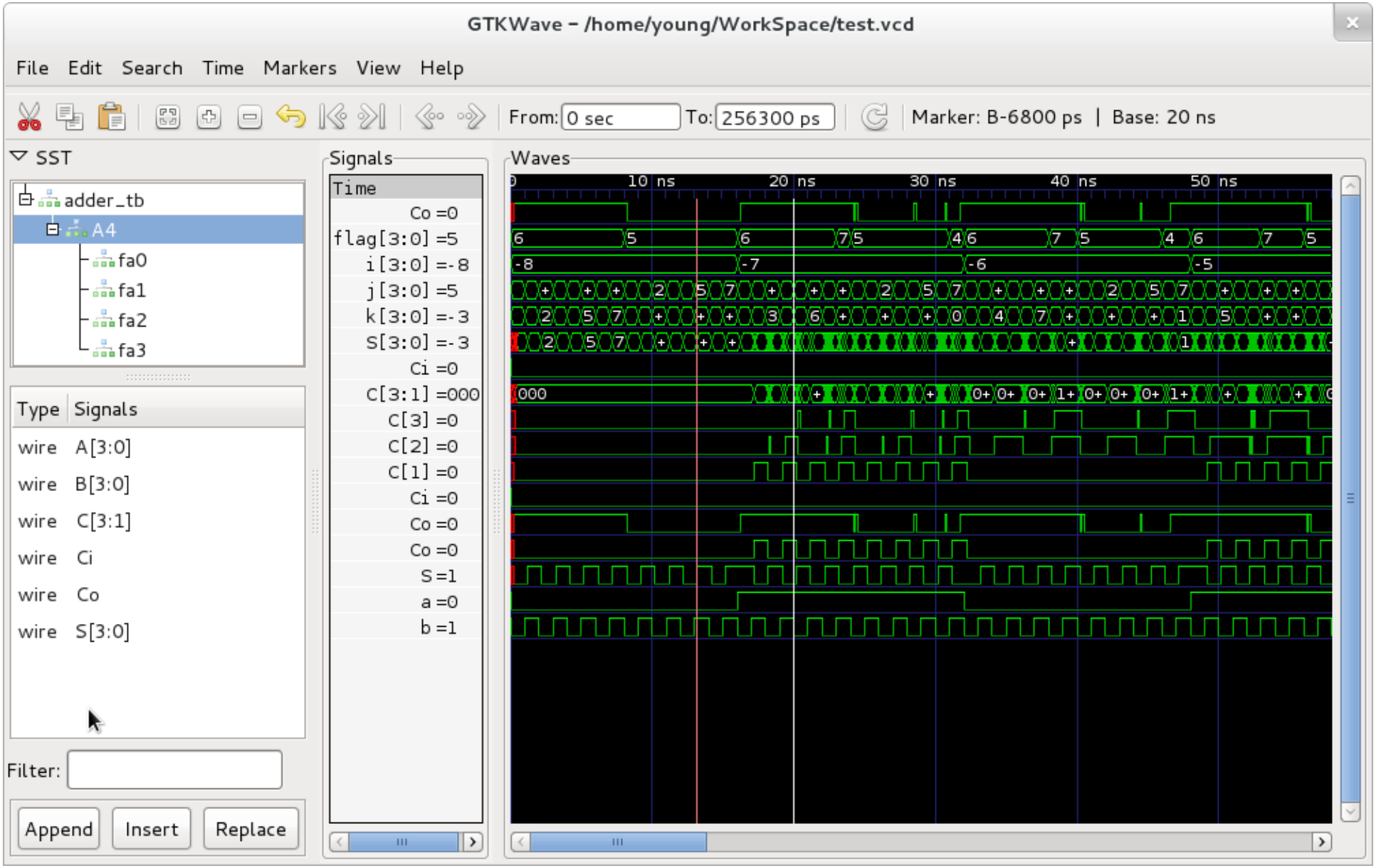
fa_arith.v

```
module fa(a, b, c, Co, S);
  input a, b, c;
  output Co, S;

  reg S, Co;

  always @(a or b or c)
    #0.1 {Co, S} = a + b + c;

endmodule
```

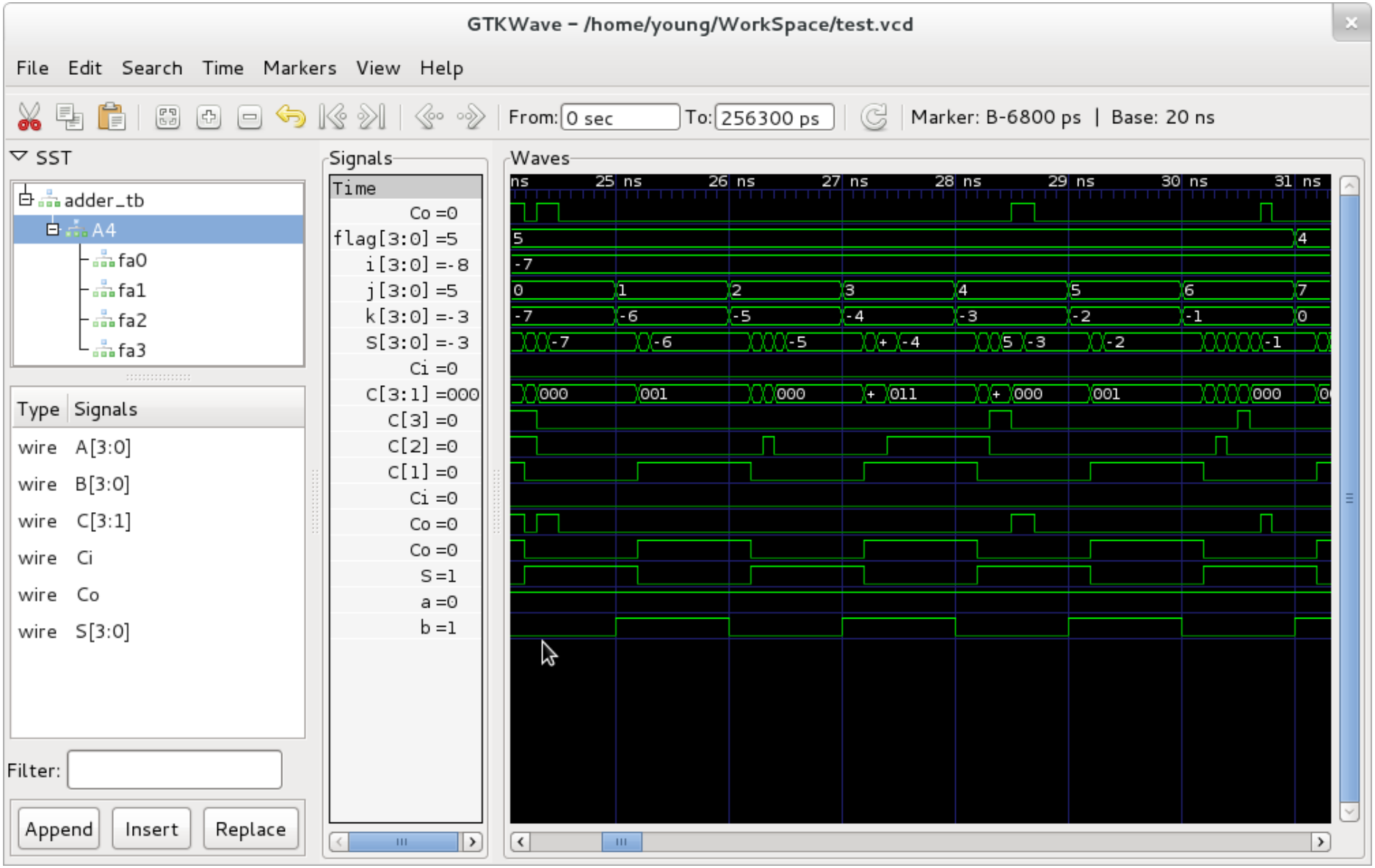
Output

```
[young@ubook WorkSpace]$ iverilog adder_tb.v adder4.v fa_gate.v
[young@ubook WorkSpace]$ vvp a.out
VCD info: dumpfile test.vcd opened for output.
i=-8 j=-8 k= 0 S= 0 OV
i=-8 j=-7 k= 1 S= 1 OV
i=-8 j=-6 k= 2 S= 2 OV
i=-8 j=-5 k= 3 S= 3 OV
i=-8 j=-4 k= 4 S= 4 OV
i=-8 j=-3 k= 5 S= 5 OV
i=-8 j=-2 k= 6 S= 6 OV
i=-8 j=-1 k= 7 S= 7 OV
i=-8 j= 0 k=-8 S=-8
i=-8 j= 1 k=-7 S=-7
i=-8 j= 2 k=-6 S=-6
i=-8 j= 3 k=-5 S=-5
i=-8 j= 4 k=-4 S=-4
i=-8 j= 5 k=-3 S=-3
i=-8 j= 6 k=-2 S=-2
i=-8 j= 7 k=-1 S=-1
i=-7 j=-8 k= 1 S= 1 OV
i=-7 j=-7 k= 2 S= 2 OV
i=-7 j=-6 k= 3 S= 3 OV
i=-7 j=-5 k= 4 S= 4 OV
i=-7 j=-4 k= 5 S= 5 OV
i=-7 j=-3 k= 6 S= 6 OV
i=-7 j=-2 k= 7 S= 7 OV
i=-7 j=-1 k=-8 S=-8
i=-7 j= 0 k=-7 S=-7
i=-7 j= 1 k=-6 S=-6
i=-7 j= 2 k=-5 S=-5
i=-7 j= 3 k=-4 S=-4
i=-7 j= 4 k=-3 S=-3
i=-7 j= 5 k=-2 S=-2
i=-7 j= 6 k=-1 S=-1
i=-7 j= 7 k= 0 S= 0
i=-6 j=-8 k= 2 S= 2 OV
i=-6 j=-7 k= 3 S= 3 OV
i=-6 j=-6 k= 4 S= 4 OV
i=-6 j=-5 k= 5 S= 5 OV
i=-6 j=-4 k= 6 S= 6 OV
i=-6 j=-3 k= 7 S= 7 OV
i=-6 j=-2 k=-8 S=-8
i=-6 j=-1 k=-7 S=-7
i=-6 j= 0 k=-6 S=-6
i=-6 j= 1 k=-5 S=-5
i=-6 j= 2 k=-4 S=-4
i=-6 j= 3 k=-3 S=-3
i=-6 j= 4 k=-2 S=-2
```

Waveform (1)


GTKWave – /home/young/WorkSpace/test.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 256300 ps
Marker: B-6800 ps | Base: 20 ns
SST
adder_tb
A4
fa0
fa1
fa2
fa3
Type Signals
wire A[3:0]
wire B[3:0]
wire C[3:1]
wire Ci
wire Co
wire S[3:0]
Filter:
Append
Insert
Replace
Signals
Time
Co =0
flag[3:0] =5
i[3:0] =-8
j[3:0] =5
k[3:0] =-3
S[3:0] =-3
Ci =0
C[3:1] =000
C[3] =0
C[2] =0
C[1] =0
Ci =0
Co =0
Co =0
S =1
a =0
b =1
Waves
10 ns
20 ns
30 ns
40 ns
50 ns

Waveform (2)


GTKWave – /home/young/WorkSpace/test.vcd
File Edit Search Time Markers View Help
From: 0 sec
To: 256300 ps
Marker: B-6800 ps | Base: 20 ns
SST
adder_tb
A4
fa0
fa1
fa2
fa3
Type Signals
wire A[3:0]
wire B[3:0]
wire C[3:1]
wire Ci
wire Co
wire S[3:0]
Filter:
Append
Insert
Replace
Signals
Time
Co =0
flag[3:0] =5
i[3:0] =-8
j[3:0] =5
k[3:0] =-3
S[3:0] =-3
Ci =0
C[3:1] =000
C[3] =0
C[2] =0
C[1] =0
Ci =0
Co =0
Co =0
S =1
a =0
b =1
Waves
ns 25 ns 26 ns 27 ns 28 ns 29 ns 30 ns 31 ns

$dumpvars()

$dumpfile("filename.vcd")
$dumpvar dumps all variables in the design.
$dumpvar(1, top) dumps all the variables in module top, but not modules instantiated in top.
$dumpvar(2, top) dumps all the variables in module top and 1 level below.
$dumpvar(n, top) dumps all the variables in module top and n-1 levels below.
$dumpvar(0, top) dumps all the variables in module top and all level below.
$dumpon initiates the dump.
$dumpoff stop dumping.

vvp

NAME

vvp - Icarus Verilog vvp runtime engine

SYNOPSIS

vvp [-sv] [-Mpath] [-mmodule] [-llogfile] inputfile [extended-args...]

DESCRIPTION

vvp is the run time engine that executes the default compiled form generated by Icarus Verilog. The output from the iverilog command is not by itself executable on any platform. Instead, the vvp program is invoked to execute the generated output file.

References

[1] http://en.wikipedia.org/
[2] M. M. Mano, C. R. Kime, “Logic and Computer Design Fundamentals”, 4th ed.
[3] J. Stephenson, Understanding Metastability in FPGAs. Altera Corporation white paper. July 2009.