EXPERIMENT I: MSI COMPONENTS

DECODER

W	X	Y	Z	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14	M15
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Figure 1 4x16 Decoder truth table

```
`timescale 1ns / 1ps
module DECODER(input [3:0]IN, output reg [15:0]OUT);
always @(*)
begin
case(IN)
4'b0000 : OUT=16'b0000000000000001;
4'b0001 : OUT=16'b0000000000000010;
4'b0010 : OUT=16'b0000000000000100;
4'b0011 : OUT=16'b0000000000001000;
4'b0100 : OUT=16'b0000000000010000;
4'b0101 : OUT=16'b0000000000100000;
4'b0110 : OUT=16'b0000000001000000;
4'b0111 : OUT=16'b0000000010000000;
4'b1000 : OUT=16'b0000000100000000;
```

```
4'b1001 : OUT=16'b0000001000000000;
4'b1010 : OUT=16'b0000010000000000;
4'b1011 : OUT=16'b0000100000000000;
4'b1100 : OUT=16'b0001000000000000;
4'b1101 : OUT=16'b0010000000000000;
4'b1110 : OUT=16'b0100000000000000;
4'b1111 : OUT=16'b1000000000000000;
endcase
end
endmodule

`timescale 1ns / 1ps
module top_module( sw , btn, led, seg, dp,  an);
input [7:0]sw;
input [3:0]btn;
output [7:0] led;
output [6:0] seg;
output dp;
output [3:0] an;
DECODER decoder1(  sw[3:0], {dp, seg, led} );
assign an = 4'b1110;
endmodule
```

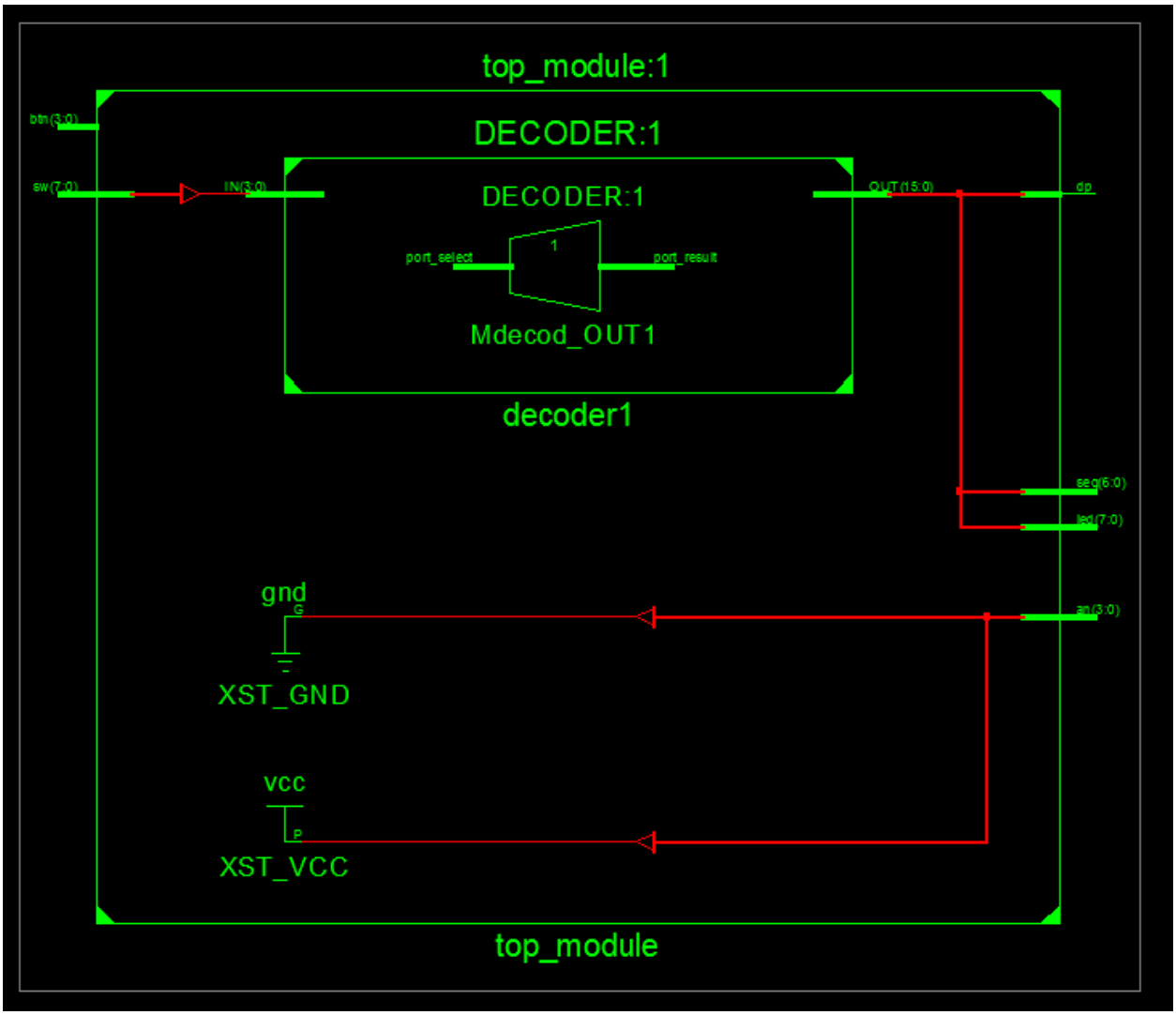

Figure 2 Decoder RTL Schematics

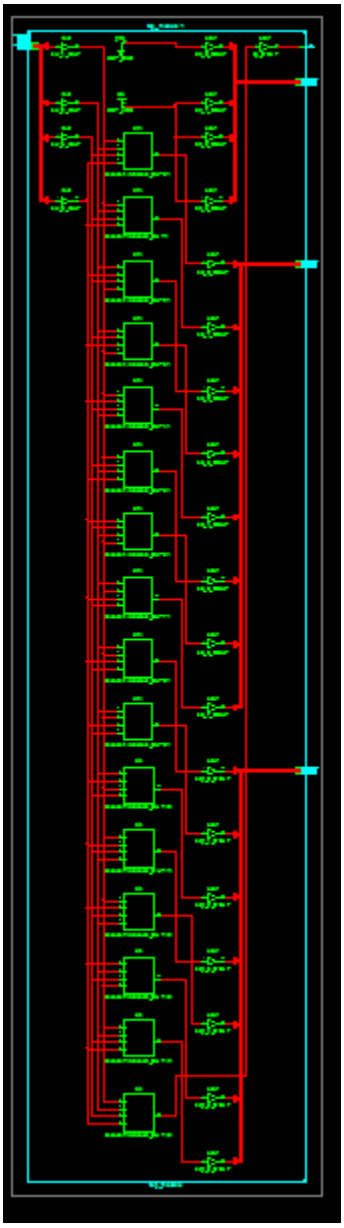

Figure 3 Technology Schematics of decoder as whole

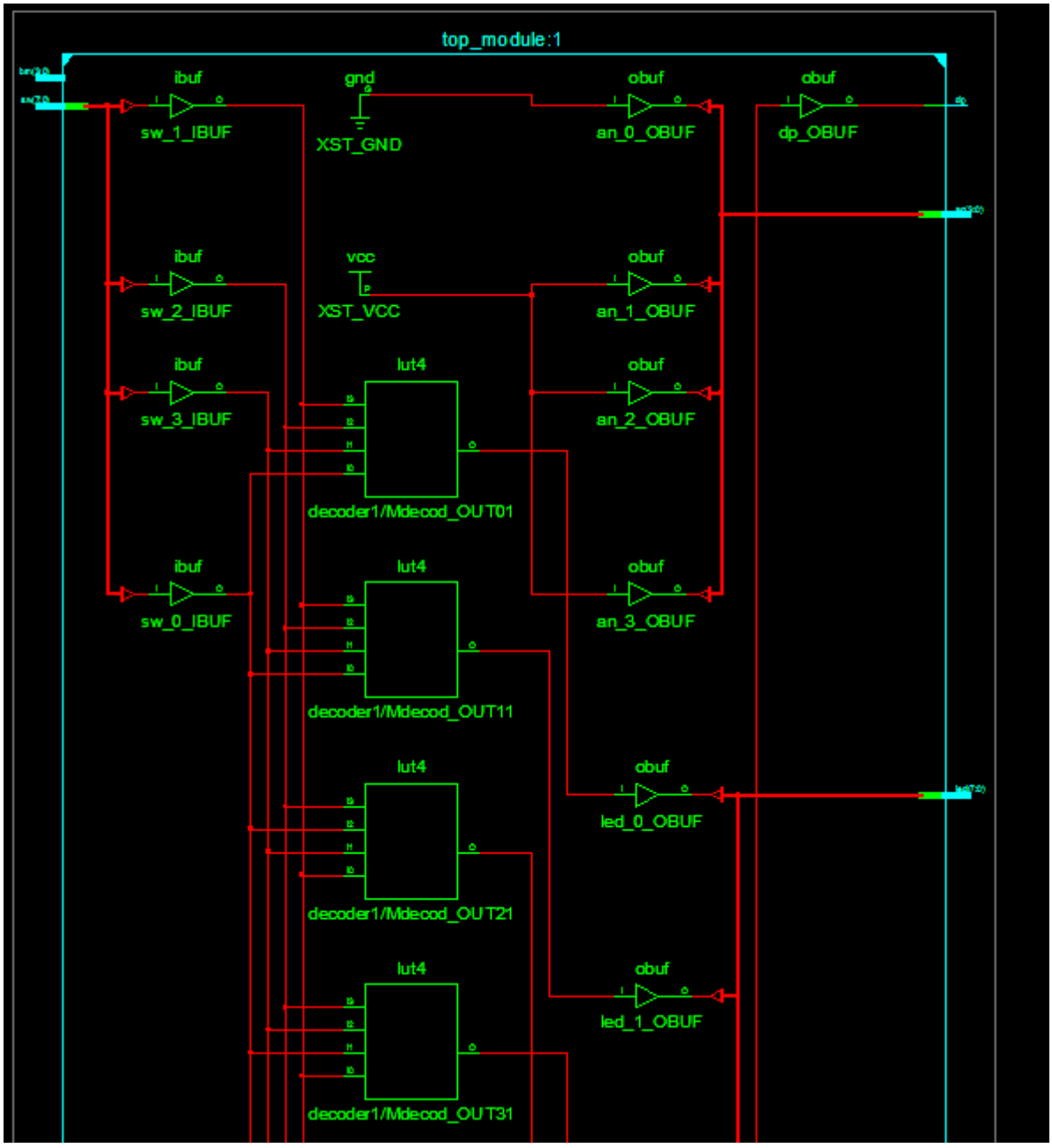

Figure 4 A part of Technology Schematics of decoder

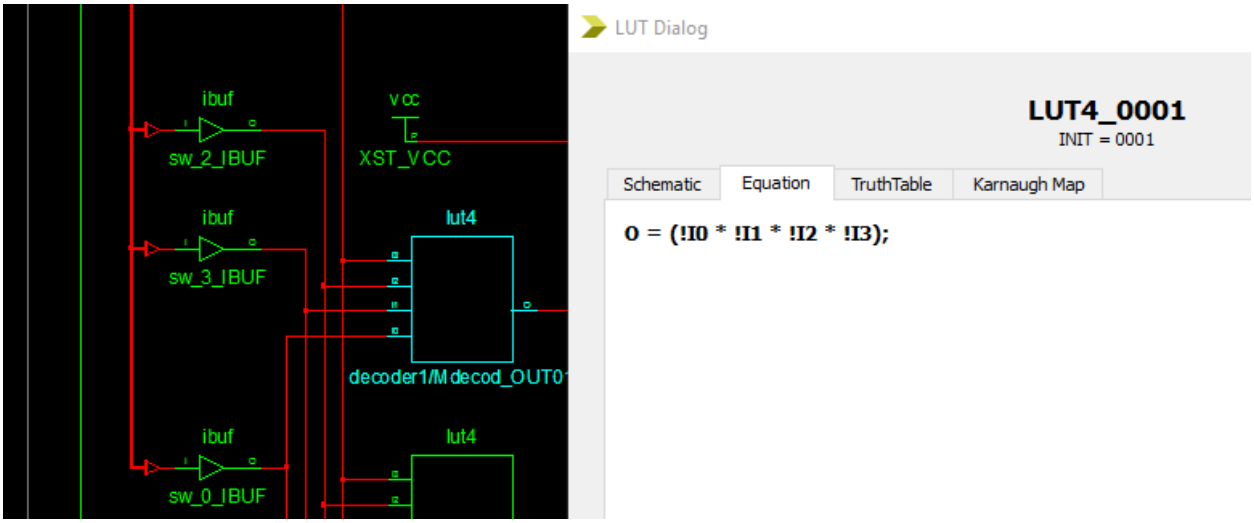

Figure 5 LUT equation of Decoder

It used 16 LUTs to complete decoder. When we change the switch, the inputs are changing. It uses switches to give our output.

sw<2>	|led<4>	8.297|
sw<2>	|led<5>	7.106|
sw<2>	|led<6>	10.456|
sw<2>	|led<7>	12.393|
sw<2>	|seg<0>	7.239|
sw<2>	|seg<1>	7.238|
sw<2>	|seg<2>	8.673|

Figure 6 Part of delay table

Greatest delay is from switch 3 to led 7 which is 12.393 ns.

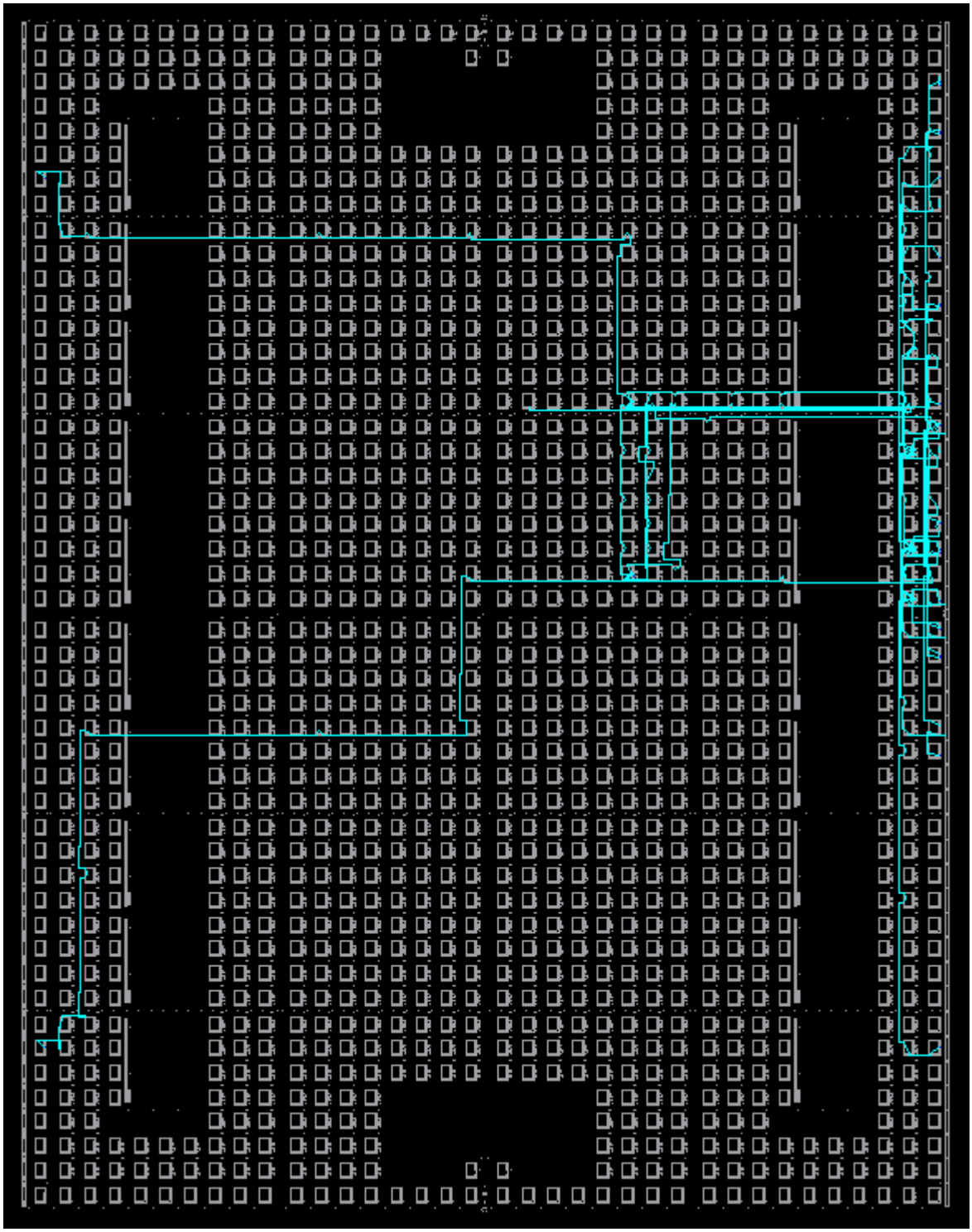

Figure 7 Layout of decoder on FPGA Editor

After adding time constraint 10 ns, greatest delay is from switch 3 to led 6 with 9.809 ns.

sw<2>	|led<5>	7.373|
sw<2>	|led<6>	9.809|
sw<2>	|led<7>	8.799|
sw<2>	|seg<0>	7.538|
sw<2>	|seg<1>	7.156|

Figure 8 Part of delay table

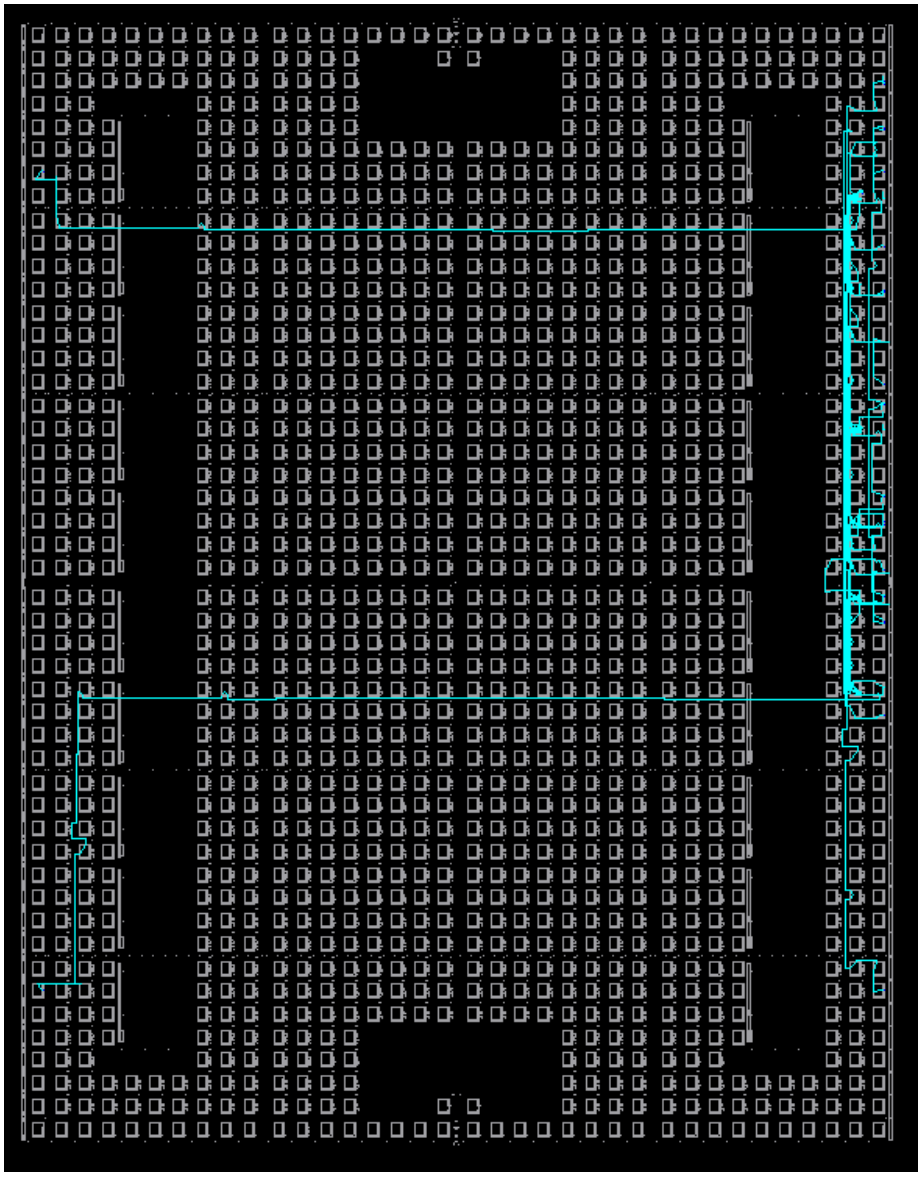

Figure 9 Layout of decoder with time constraint on FPGA Editor

Layout became more orderly to accomplish the 10ns constraint. The decoder without time constraint used any place on FPGA.

PRIORITY ENCODER

Inputs				Outputs		
D_0	D_1	D_2	D_3	Y_1	Y_0	V
0	0	0	0	×	×	0
1	0	0	0	0	0	1
×	1	0	0	0	1	1
×	×	1	0	1	0	1
×	×	×	1	1	1	1

Figure 10 Priority encoder truth table

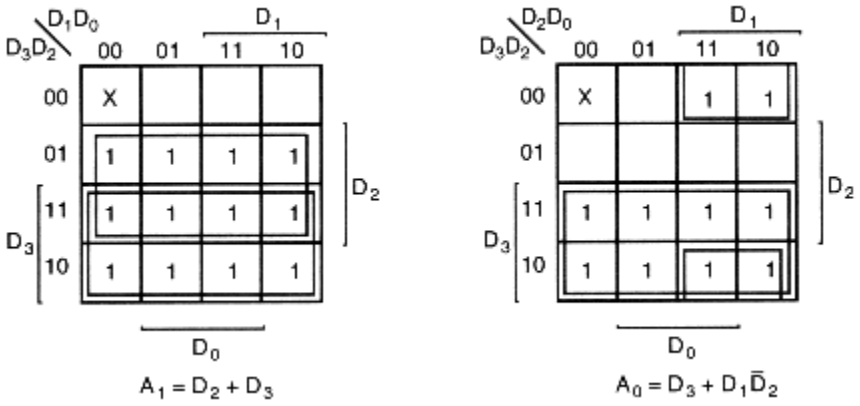

Figure 11 Priority encoder Karnaugh map and output

```
module ENCODER(IN, E,OUT);
input [3:0]IN;
output [1:0]OUT;
output E;
wire in2_not,d1xd2_not;
NOT not1(IN[2],in2_not);
```

```
AND and1(IN[1],in2_not,d1xd2_not);
OR or1(IN[3],d1xd2_not,OUT[0]);
OR or2(IN[2],IN[3],OUT[1]);
wire y0,y1;
OR or3(IN[0],IN[1],y0);
OR or4(IN[2],IN[3],y1);
OR or5(y0,y1,E);
endmodule
```

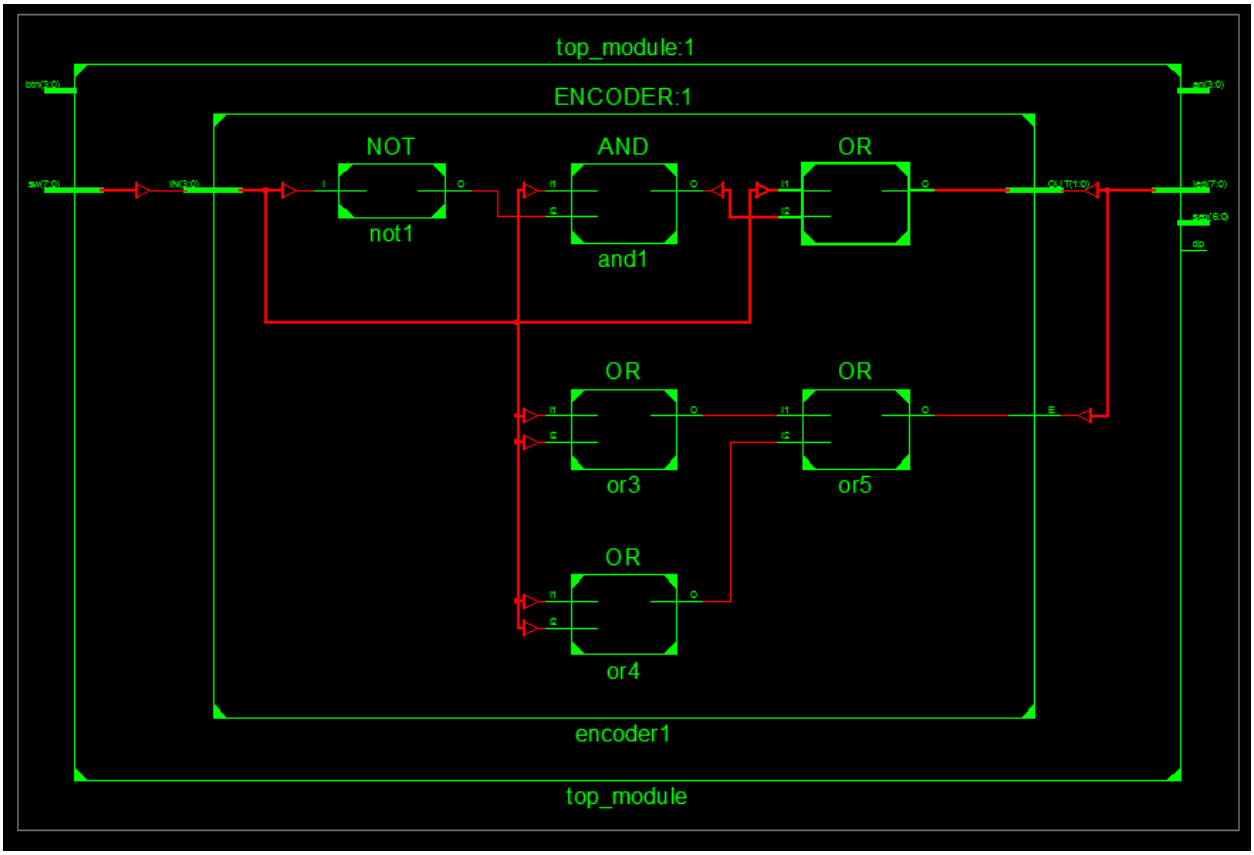

Figure 12 RTL schematics of priority encoder

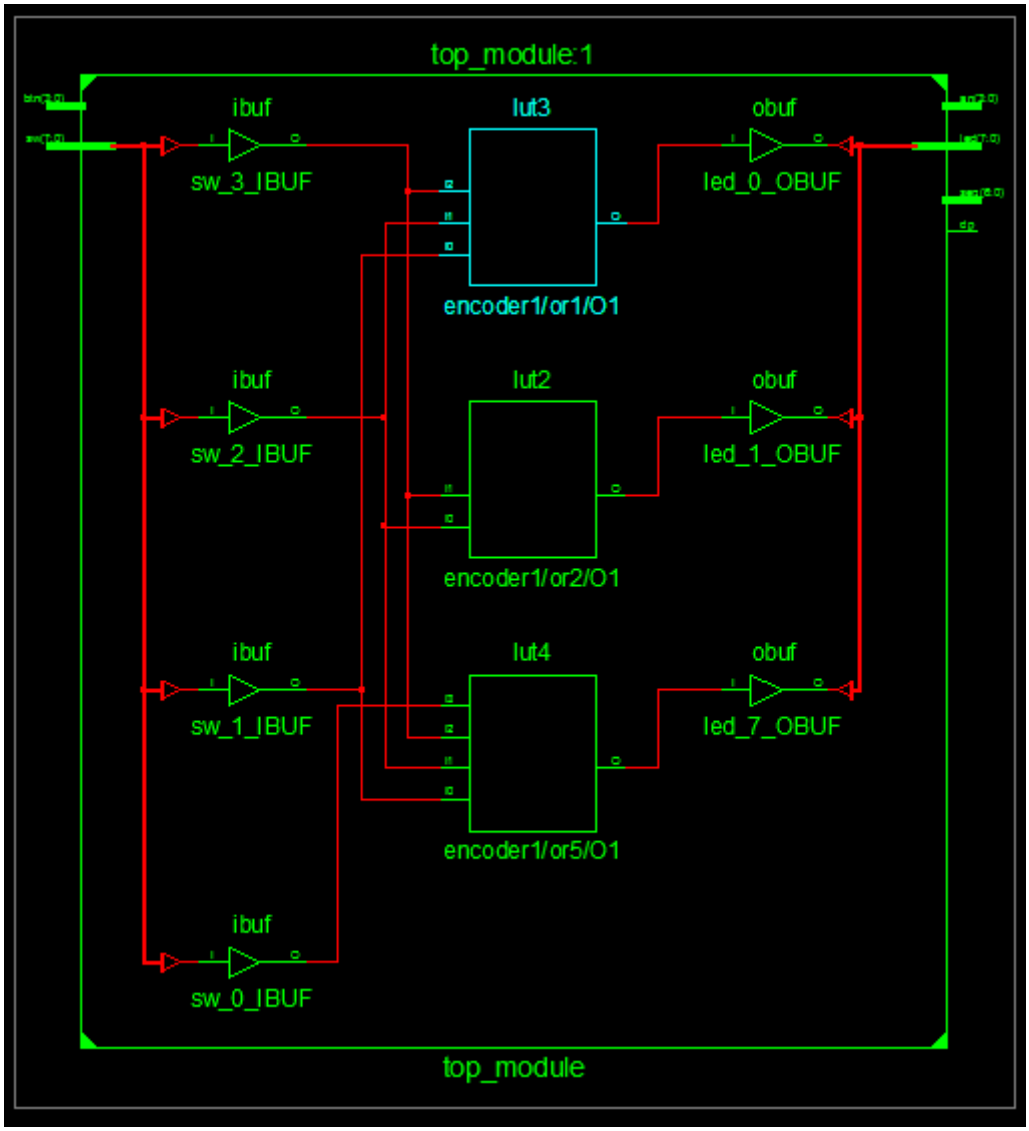

Figure 13 Technology schematics of priority encoder

Technology schematics have 3 LUTs but RTL schematics only have gates.

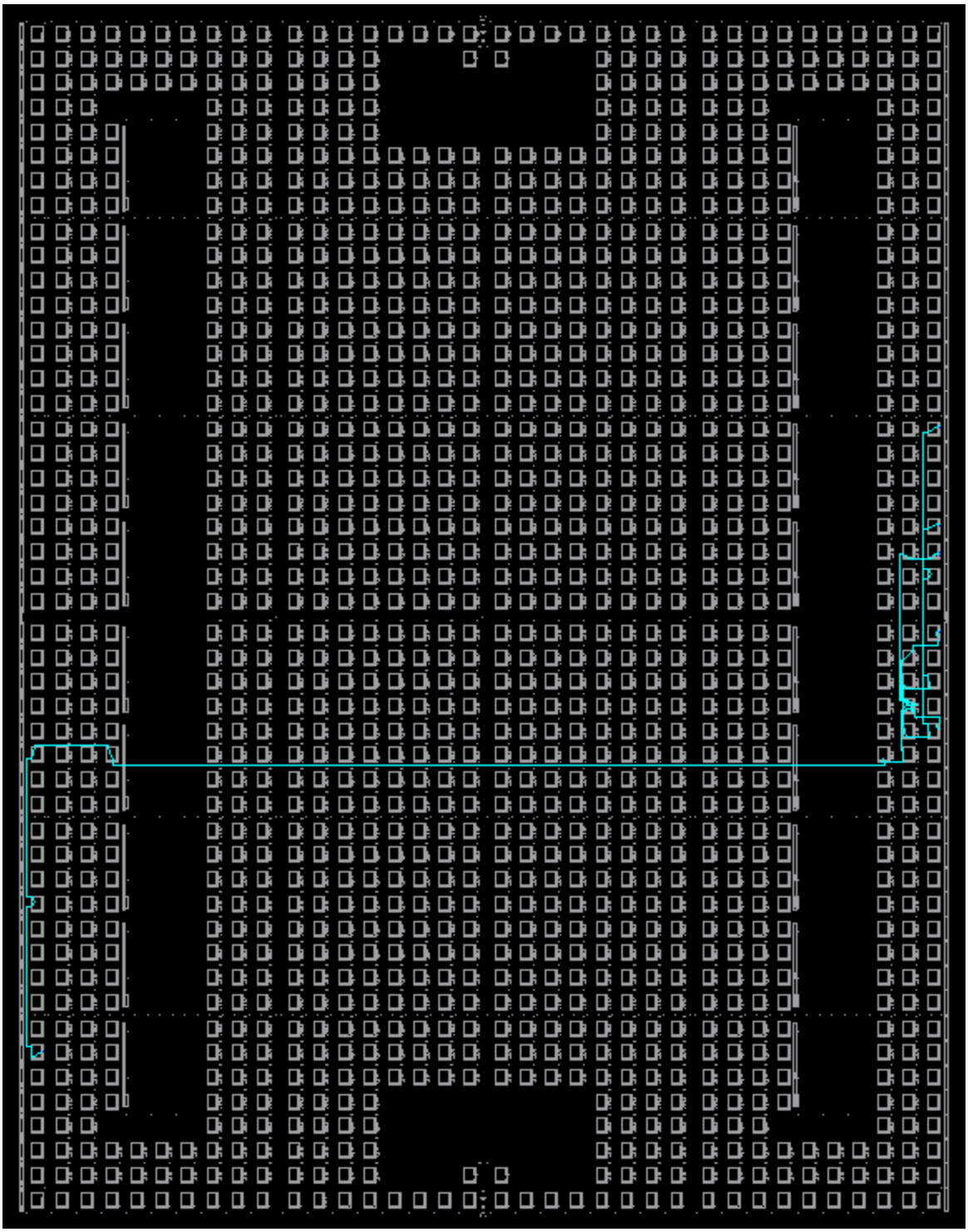

Figure 14 Layout of priority encoder on FPGA Editor

MULTIPLEXER

```
module MUX(D,S,O);
input [3:0]D;
input [1:0]S;
output O;
wire w0,w1,w2,w3;
assign w0 = (~S[0])&(~S[1])&D[0];
assign w1 = S[0]&(~S[1])&D[1];
assign w2 = (~S[0])&S[1]&D[2];
assign w3 = S[0]&S[1]&D[3];
assign O = w0|w1|w2|w3;
endmodule
```

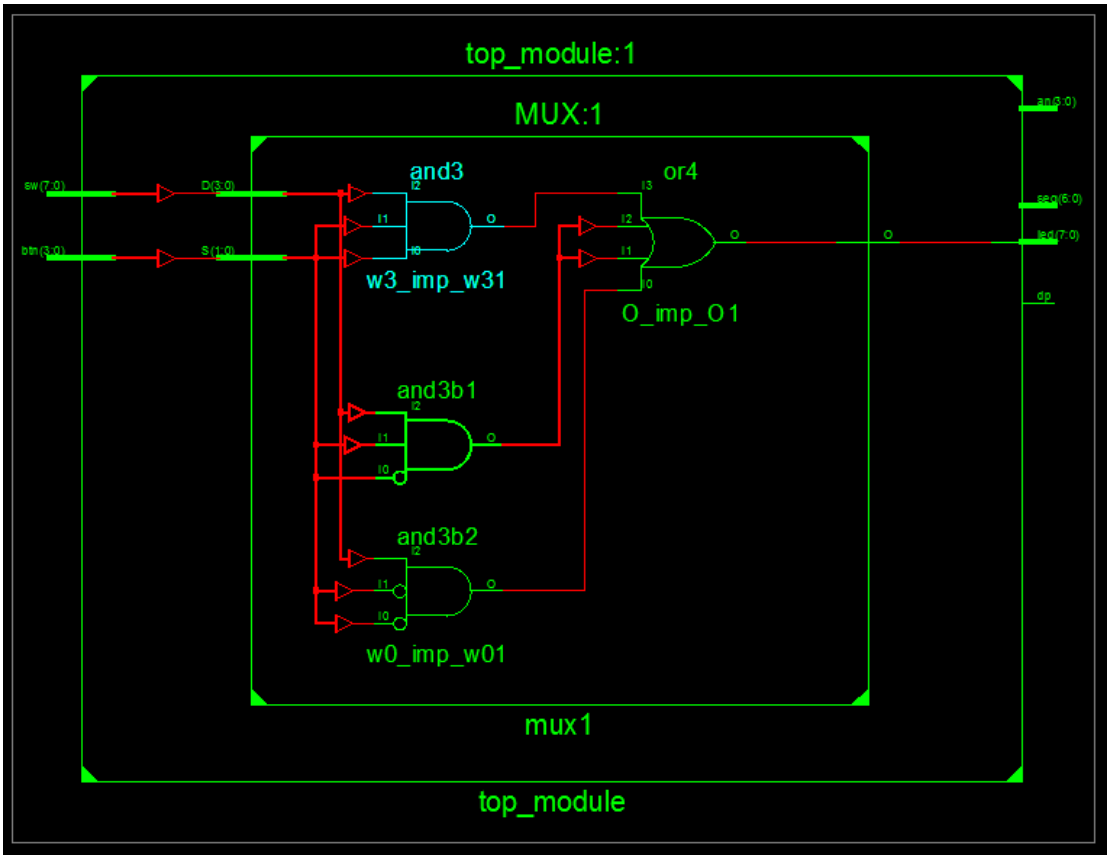

Figure 15 RLT schematics of multiplexer

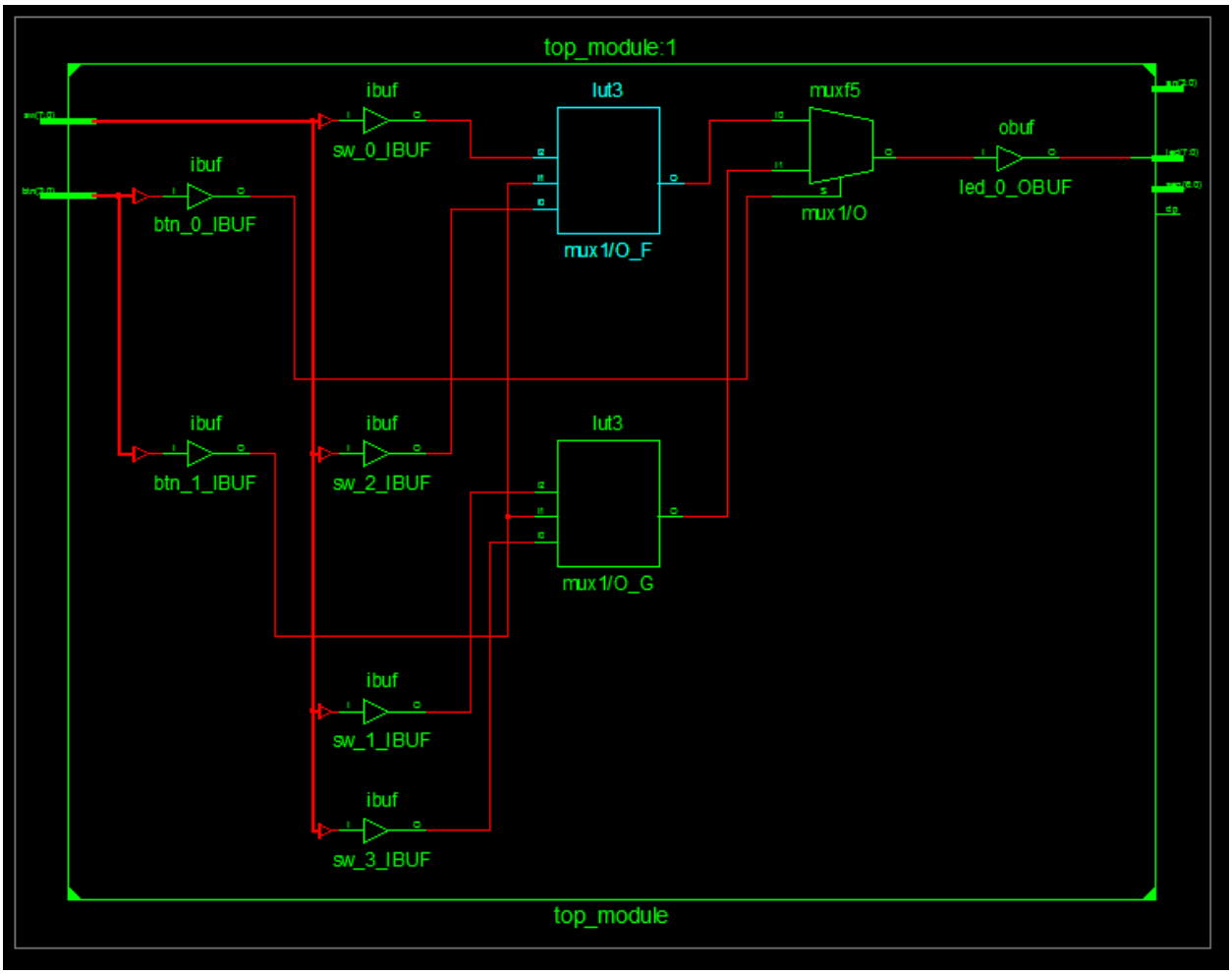

Figure 16 Technology schematics of multiplexer

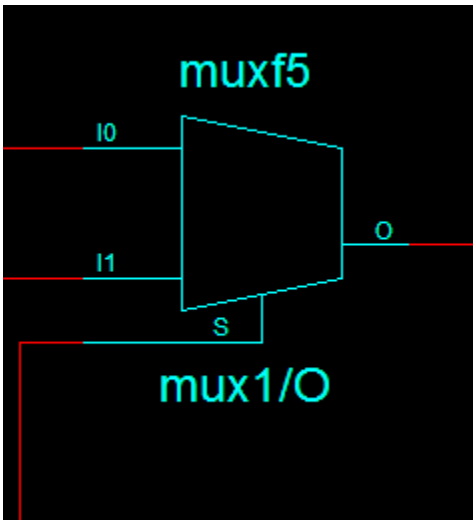

Figure 17 Multiplexer part in the technology schematics

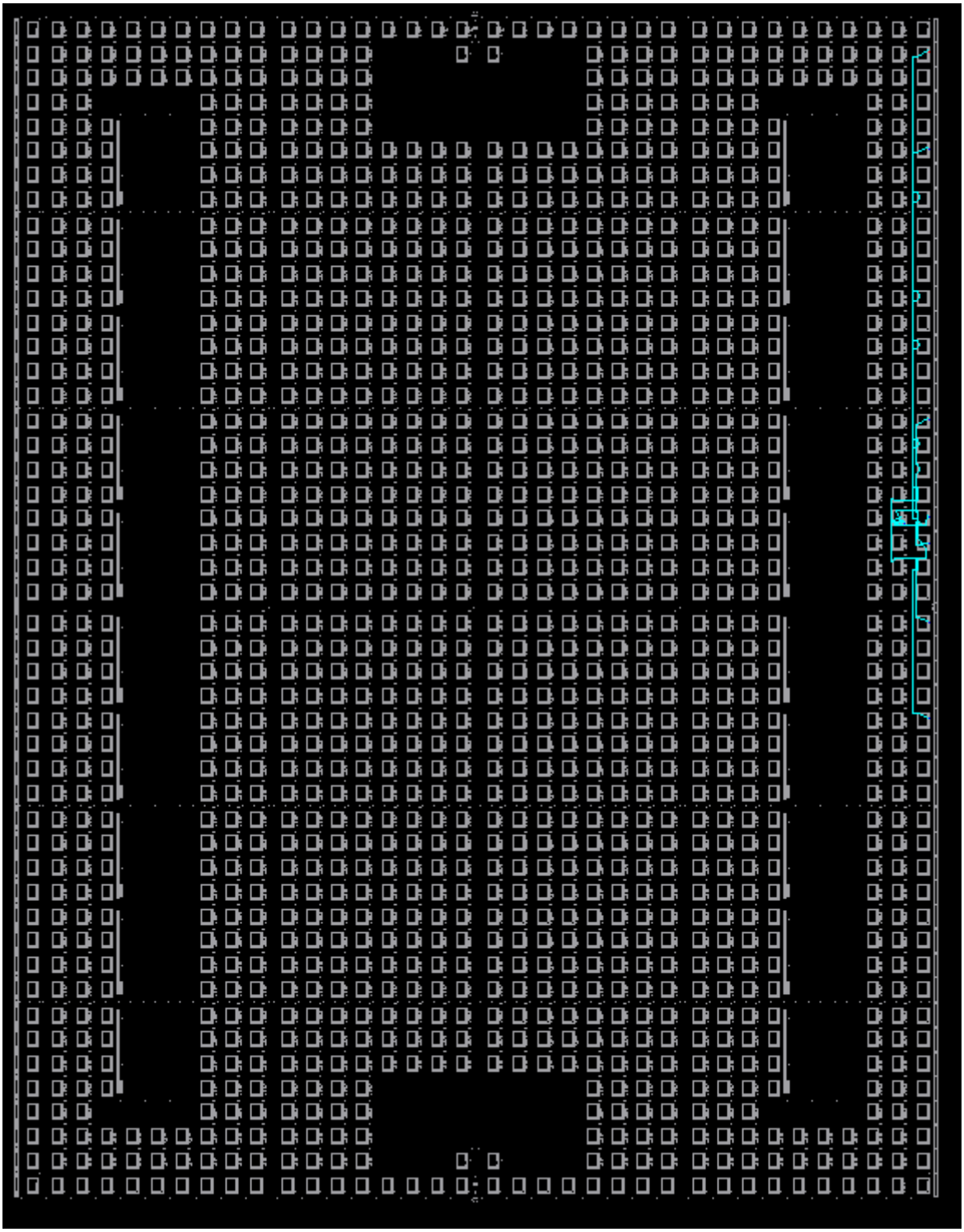

Figure 18 Layout of multiplexer in FPGA Editor

DEMULTIPLEXER

```
module DEMUX(D,S,O);
input D;
input [1:0]S;
output [3:0]O;
wire n0,n1,o0,o1,o2,o3,z0,z1,z2,z3;
NOT not1(S[0],n0);
NOT not2(S[1],n1);
AND and1(n0,n1,z0);
AND and2(S[0],n1,z1);
AND and3(n0,S[1],z2);
AND and4(S[0],S[1],z3);
TRI trI1(D,z0,O[0]);
TRI trI2(D,z1,O[1]);
TRI trI3(D,z2,O[2]);
TRI trI4(D,z3,O[3]);
endmodule
```

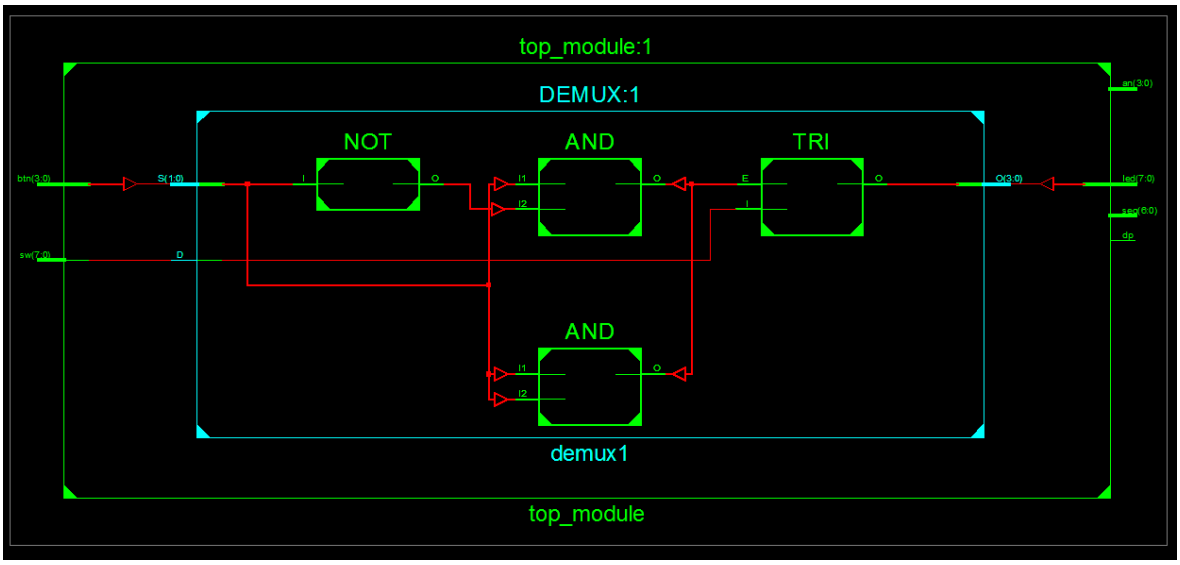

Figure 19 RTL schematics of demultiplexer

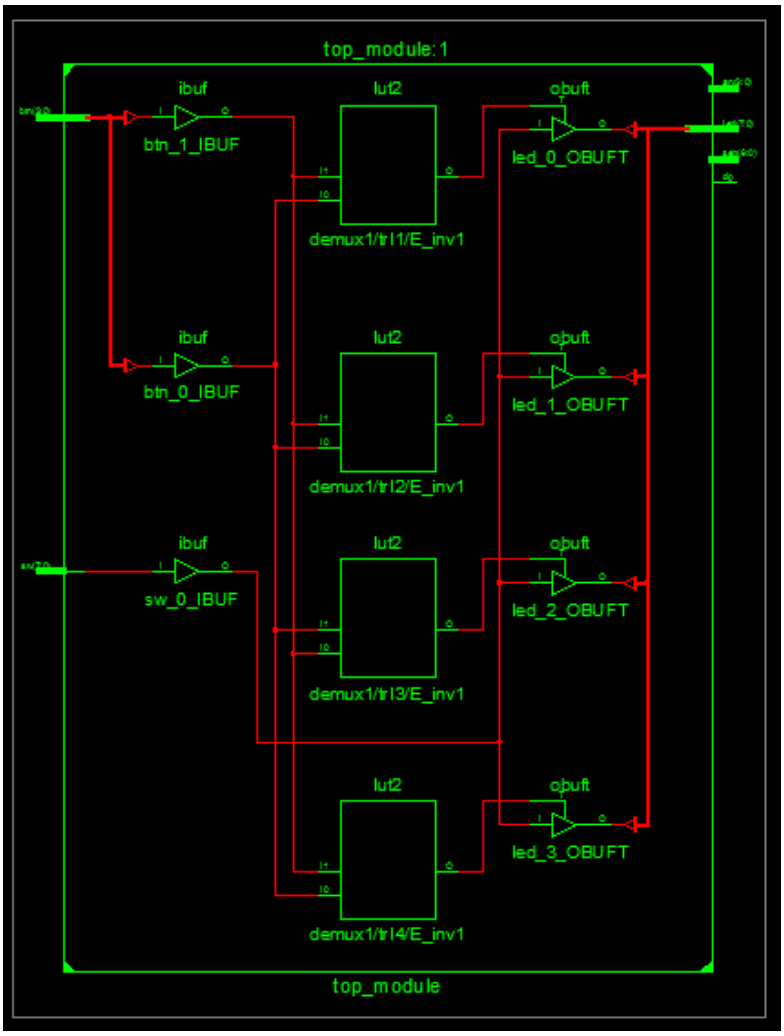

Figure 20 Technology schematics of demultiplexer

LUTs in the technology have OR gates and NOT gates in them. We pick which LUT will work with buttons on FPGA.

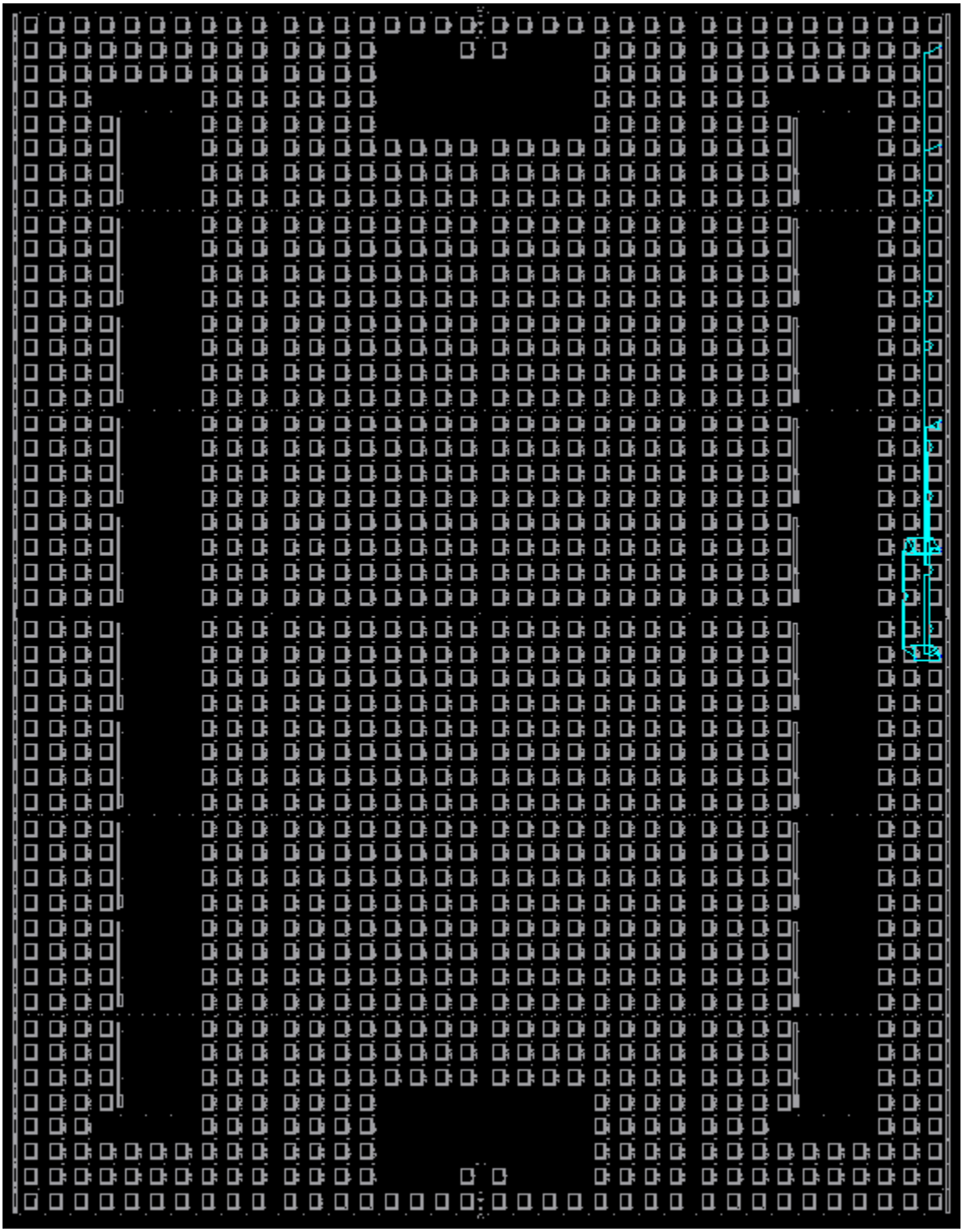

Figure 21 Layout of demultiplexer in FPGA Editor

Demultiplexer’s outputs have faster response rate than others. Because its inputs are coming from near. However, its slew rate is slow.

ALWAYS CASES

```
module MUX(D,S,O);
input [3:0]D;
input [1:0]S;
output O;
reg O;
always @( D )
  begin
   case ( S )
     2'd0 : O = D[0];
     2'd1 : O = D[1];
     2'd2 : O = D[2];
     2'd3 : O = D[3];
     default : O = 1'bx;
   endcase
 end
endmodule
```

```
Data Sheet report:
-----------------
All values displayed in nanoseconds (ns)

Pad to Pad
---------------+---------------+---------+
Source Pad     |Destination Pad|   Delay  |
---------------+---------------+---------+
btn<0>         |led<0>         |    8.145|
btn<1>         |led<0>         |    7.894|
sw<0>          |led<0>         |    7.423|
sw<1>          |led<0>         |    6.926|
sw<2>          |led<0>         |    7.344|
sw<3>          |led<0>         |    7.562|
---------------+---------------+---------+
```

Figure 22 Delays of pads in mutiplexer with always and case

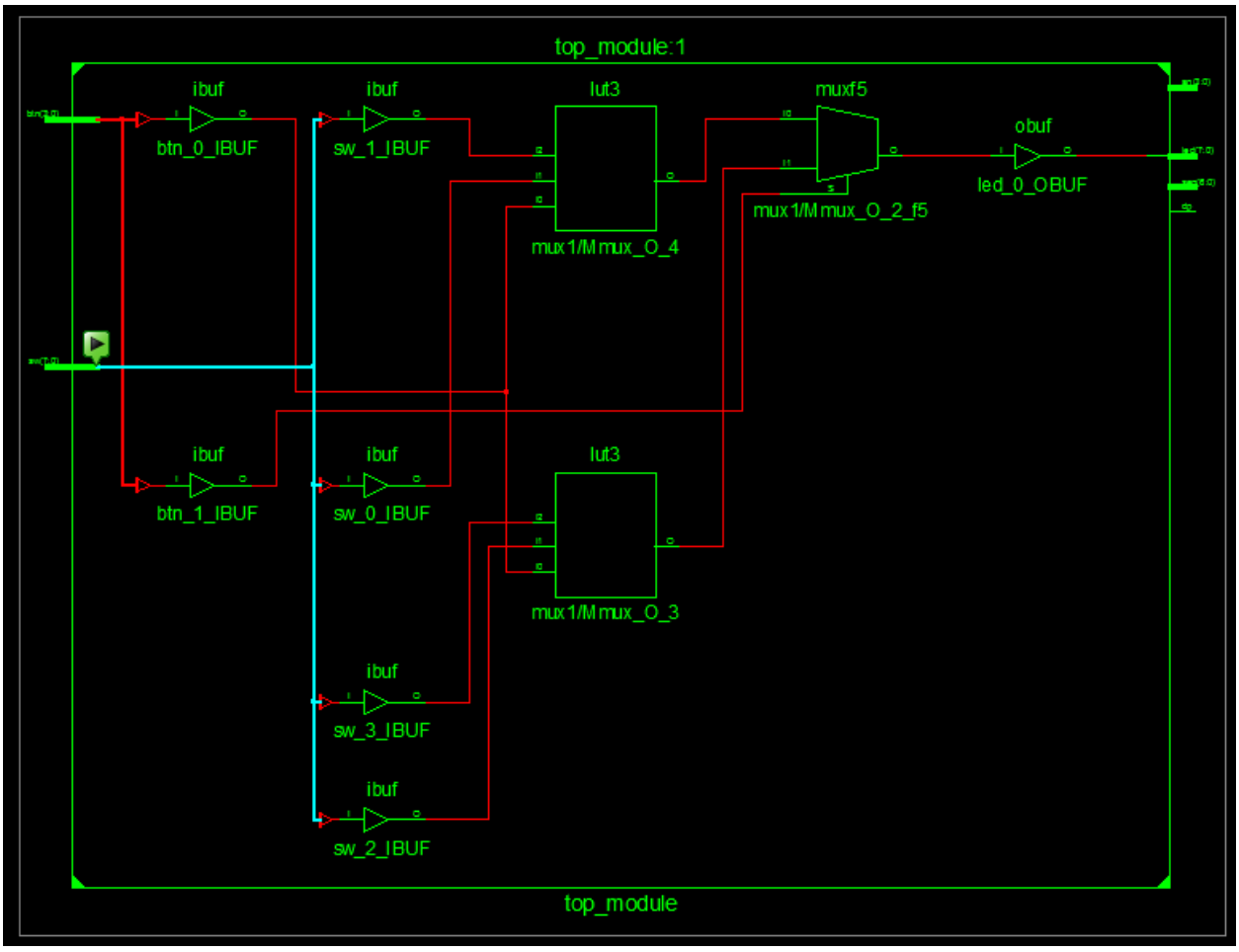

Figure 23 Technology schematics of multiplexer with always and case

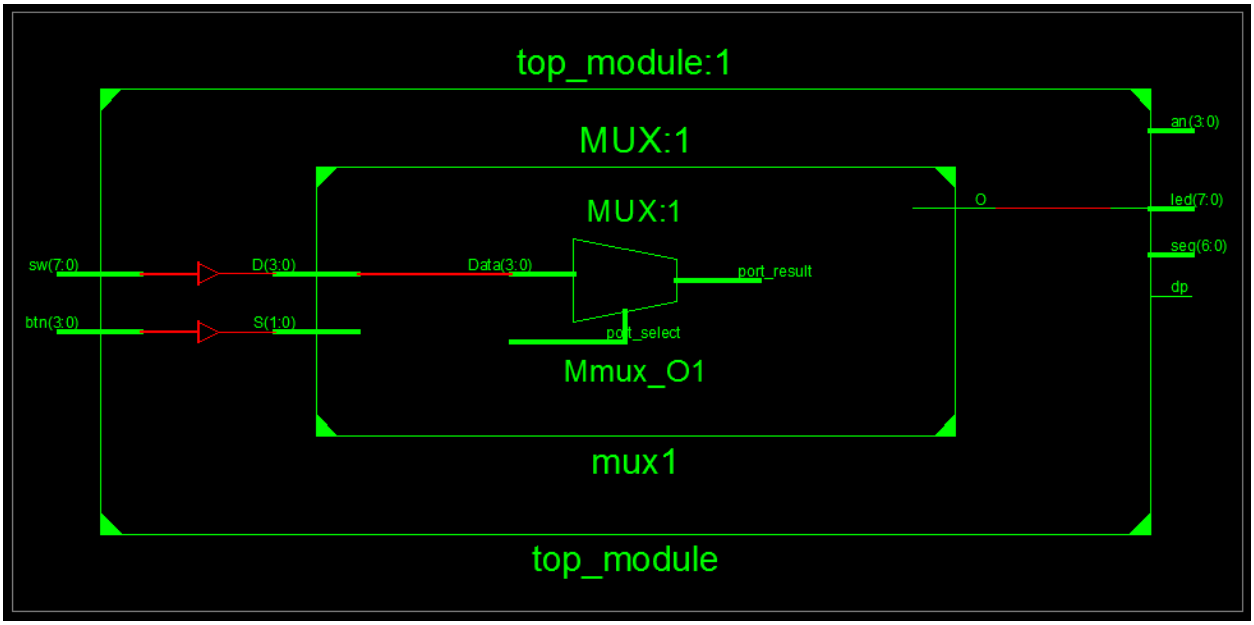

Figure 24 RTL schematics of multiplexer with always and case

Even though, technology schematics are same, in RTL schematics it used normal multiplexer instead of gates.

```
Data Sheet report:
------------------
All values displayed in nanoseconds (ns)

Pad to Pad
---------------+---------------+---------+
Source Pad     |Destination Pad|  Delay  |
---------------+---------------+---------+
btn<0>         |led<0>         |    8.167|
btn<1>         |led<0>         |    7.887|
sw<0>          |led<0>         |    7.423|
sw<1>          |led<0>         |    6.865|
sw<2>          |led<0>         |    7.407|
sw<3>          |led<0>         |    7.562|
---------------+---------------+---------+
```

Figure 25 delay table with gates

Delays are same, because we still use same places for inputs and outputs.

```
module ENCODER(IN, E,OUT);
input [3:0]IN;
output [1:0]OUT;
output E;
reg[1:0]OUT;
reg E;
always @(IN)
begin
case(IN)
        4'b1XXX: OUT= 2'b11;
        4'b01XX: OUT= 2'b10;
        4'b001X: OUT= 2'b01;
        4'b0001: OUT= 2'b00;
        default: OUT=2'bx;

endcase
end
endmodule
```