CS207 · Digital Logic

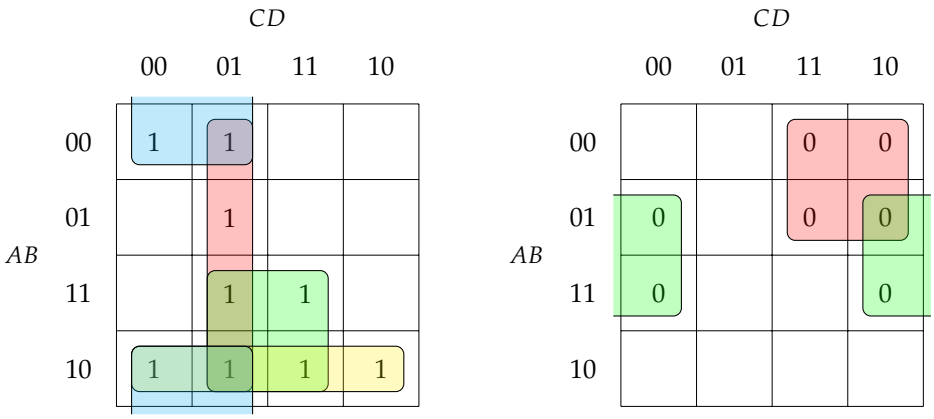
何泽安 (He Zean)
12011323

Assignment 3

Dec. 6, 2021

1 DIGITAL DESIGN THEORY

Q. 1



(a) $F(A,B,C,D) = B'C' + C'D + AD + AB'$

(b) $F(A,B,C,D) = (B+C)' + (C+D')' + (A'+D')' + (A'+B)'$

(c) $F(A,B,C,D) = ((B'C')'(C'D)'(AD)'(AB')')'$

(d) $F(A,B,C,D) = (B'+D)(A+C')$

(e) $F(A,B,C,D) = ((B'+D)' + (A+C')')'$

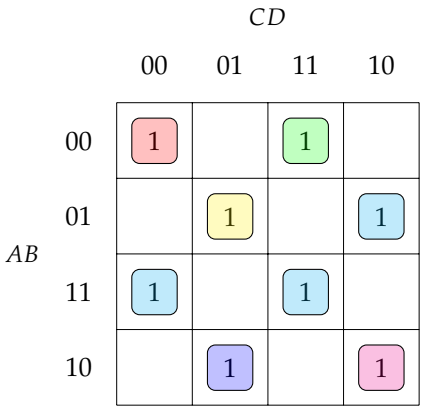
(f) $F(A,B,C,D) = (BD')'(A'C)'$

Q. 2

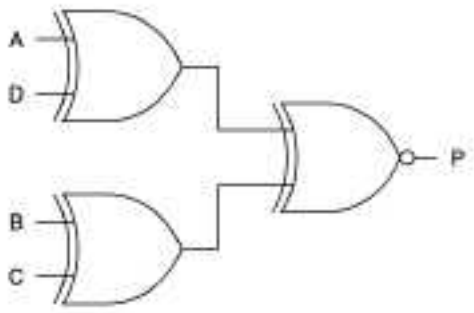
Four-bit Parity Generator

Input 4-bit message				Odd parity generator
A	B	C	D	P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0

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



$$
\begin{aligned}
F(A,B,C,D) &= A'B'C'D' + A'B'CD + A'BC'D + A'BCD' + ABC'D' + ABCD + AB'C'D + AB'CD' \\
&= A'D'(BC + B'C') + A'D(B'C + BC') + AD'(BC' + B'C) + AD(BC + B'C') \\
&= A'D'(B \oplus C)' + A'D(B \oplus C) + AD'(B \oplus C) + AD(B \oplus C)' \\
&= (A'D' + AD)(B \oplus C)' + (A'D + AD')(B \oplus C) \\
&= (A \oplus D)'(B \oplus C)' + (A \oplus D)(B \oplus C) \\
&= ((A \oplus D) \oplus (B \oplus C))'
\end{aligned}
$$



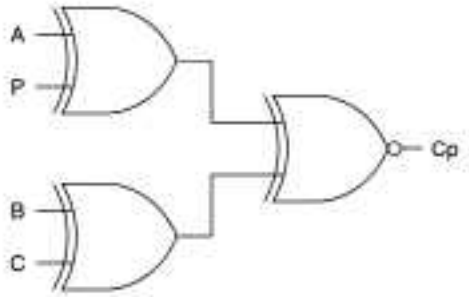
Three-bit Parity Checker

Input (3+1)-bit				Odd parity checker
A	B	C	P	C_P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0

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

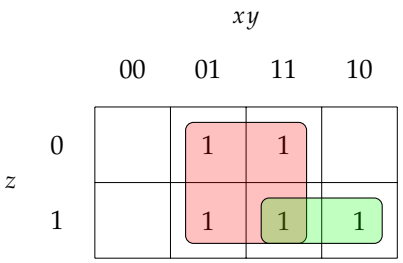
AB \ CP	00	01	11	10
00	1		1	
01		1		1
11	1		1	
10		1		1

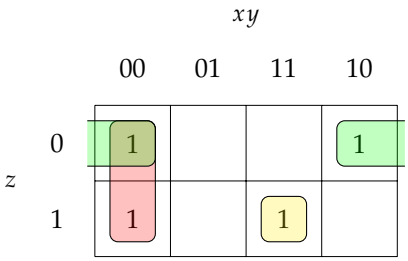
$$
\begin{aligned}
F(A,B,C,P) &= A'B'C'P' + A'B'CP + A'BC'P + A'BCP' + ABC'P' + ABCP + AB'C'P + AB'CP' \\
&= A'P'(BC + B'C') + A'P(B'C + BC') + AP'(BC' + B'C) + AP(BC + B'C') \\
&= A'P'(B \oplus C)' + A'P(B \oplus C) + AP'(B \oplus C) + AP(B \oplus C)' \\
&= (A'P' + AP)(B \oplus C)' + (A'P + AP')(B \oplus C) \\
&= (A \oplus P)'(B \oplus C)' + (A \oplus P)(B \oplus C) \\
&= ((A \oplus P) \oplus (B \oplus C))'
\end{aligned}
$$

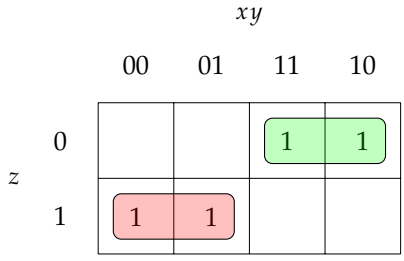


Q. 3

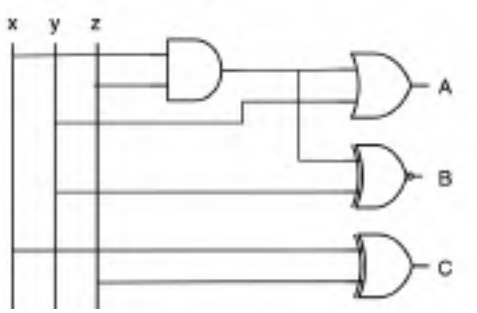
Input			Output		
x	y	z	A	B	C
0	0	0	0	1	0
0	0	1	0	1	1
0	1	0	1	0	0
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	0







$$
\begin{aligned}
A &= y + xz \\
B &= x'y' + y'z' + xyz \\
&= (x' + z')y' + (xz)y \\
&= (xz)'y' + (xz)y \\
&= ((xz) \oplus y)' \\
C &= xz' + x'z \\
&= x \oplus z
\end{aligned}
$$



Q. 4

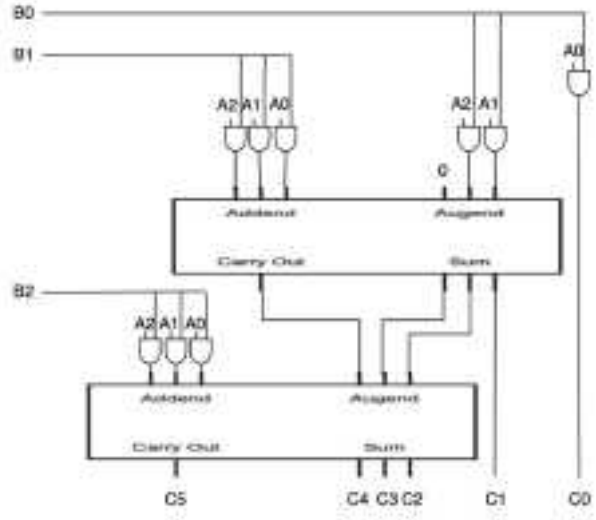
			A_2	A_1	A_0
$\times$			B_2	B_1	B_0
			A_2B_0	A_1B_0	A_0B_0
		A_2B_1	A_1B_1	A_0B_1	
	A_2B_2	A_1B_2	A_0B_2		
C_5	C_4	C_3	C_2	C_1	C_0

Starting with the traditional vertical multiplication, we found out that: ① a 3-bit multiplication is made up of three partial products; ② each partial product, is a 1-bit × 3-bit, while bits can only be 0 or 1, this multiplication doesn't have carry out, and thus can simply use three; *and gate* to denote them ③ the partial products are added up to give the final result. Formally, we say:

$$
\begin{aligned}
C_0 &= A_0B_0 && \text{(no carry out)} \\
\{\text{Carry}_1, C_1\} &= A_1B_0 + A_0B_1 && \text{LHS} = 2 \times \text{Carry}_1 + C_1 \\
\{\text{Carry}_2, C_2\} &= A_2B_0 + A_1B_1 + A_0B_2 + \text{Carry}_1 \\
&\cdots \\
\{\text{Carry}_4, C_4\} &= A_2B_2 + \text{Carry}_3 \\
C_5 &= \text{Carry}_4
\end{aligned}
$$

A.k.a., since multiply takes *AND* operation of the all binary combination from the bits of A and B and add the result with shifting *i+j (i,j denotes the indices of bits in A and B)* operations. To express the final result

in binary form, we only take the least bit of output sum and feed the rest bits of output sum together with the carry out bit into the augend input of the next full adder.



The circuit can be represented as the following logic formulas:

$$C_0 = A_0B_0$$

$$C_1 = (A_1B_0) \oplus (A_0B_1)$$

$$C_2 = (A_2B_0) \oplus (A_1B_1) \oplus (A_0B_2) \oplus (A_1B_0A_0B_1)$$

$$C_3 = (A_2B_1) \oplus (A_1B_2) \oplus (A_2B_0A_1B_1A_0B_2B_0)$$

$$C_4 = (A_2B_2) \oplus (A_2B_0A_1B_1A_0B_2B_0)$$

$$C_5 = A_2B_0A_1B_1A_0B_2B_0$$

Q. 5

(a) $F(A, B, C, D) = \Sigma(1, 3, 5, 8, 10, 14)$

D1, D3, D5, D8, D10, D14 should be connected to *signal 1*, while the left ports (D0, D2, D4, D6, D7, D9, D11, D12, D13, D15) should be connected to *signal 0*.

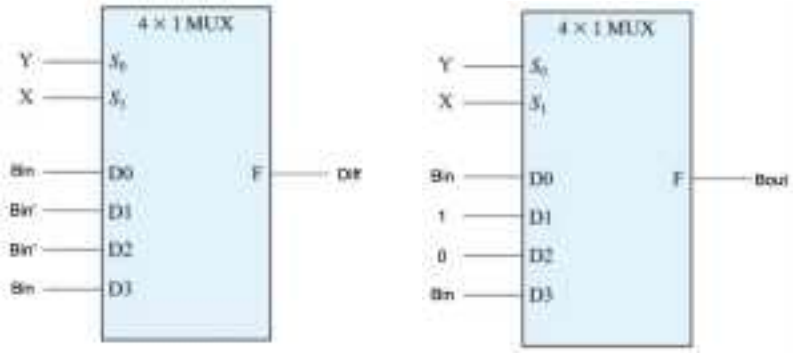
(b) $F(A, B, C, D) = \Pi(4, 7, 11) = \Sigma(0, 1, 2, 3, 5, 6, 8, 9, 10, 12, 13, 14, 15)$

D4, D7, D11 should be connected to *signal 0*, while the left ports (D0, D1, D2, D3, D5, D6, D8, D9, D10, D12, D13, D14, D15) should be connected to *signal 1*.

Q. 6

Input			Output	
X	Y	B_{in}	Diff	B_{out}
0	0	0	0	0
0	0	1	1	1

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



2 DIGITAL DESIGN LAB

Task 1

Design File

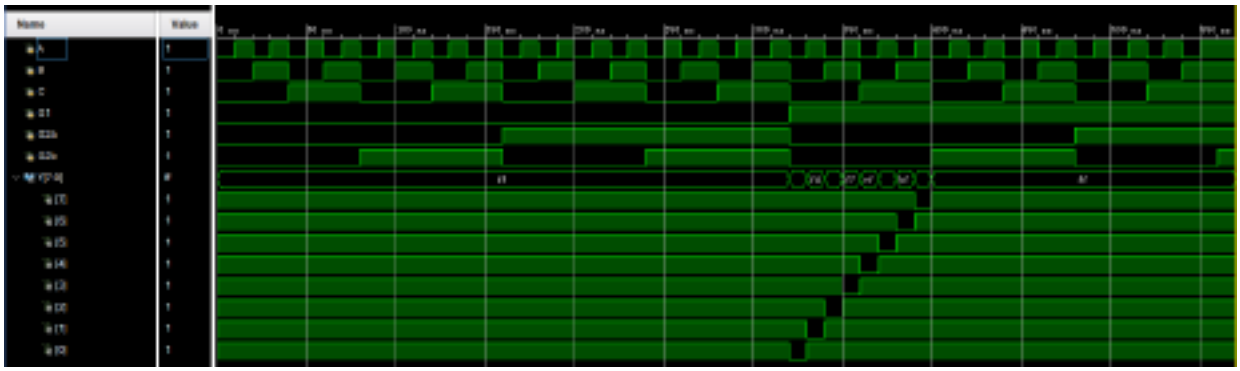
```
`timescale 1ns/1ps

module decode_74138(
    input A, B, C,
    input G1, G2a, G2b,
    output reg [7:0] Y
);
    always @ * begin
        if ({G1, G2a, G2b} == 'b100) begin
            case ({C, B, A})
                'd0: Y <= ~'b0000_0001;
                'd1: Y <= ~'b0000_0010;
                'd2: Y <= ~'b0000_0100;
                'd3: Y <= ~'b0000_1000;
                'd4: Y <= ~'b0001_0000;
                'd5: Y <= ~'b0010_0000;
                'd6: Y <= ~'b0100_0000;
                'd7: Y <= ~'b1000_0000;
            endcase
        end
        else begin
            Y = ~'b0000_0000;
        end
    end
endmodule
```

```
26
27
28  module decode_4_16(
29      input A, B, C, D,
30      input dec_en,
31      output [15:0] Y
32  );
33      reg [2:0] lo_en = 'b000;
34      reg [2:0] hi_en = 'b000;
35      decode_74138 lo(A, B, C, lo_en[2], lo_en[1], lo_en[0], Y[7:0]);
36      decode_74138 hi(A, B, C, hi_en[2], hi_en[1], hi_en[0], Y[15:8]);
37
38          always @ * begin
39          if (dec_en) begin
40              if (!D) begin
41                  lo_en <= 'b100;
42                  hi_en <= 'b000;
43              end
44              else begin
45                  lo_en <= 'b000;
46                  hi_en <= 'b100;
47              end
48          end
49          else begin
50              lo_en <= 'b000;
51              hi_en <= 'b000;
52          end
53      end
54  endmodule
```

Simulation File

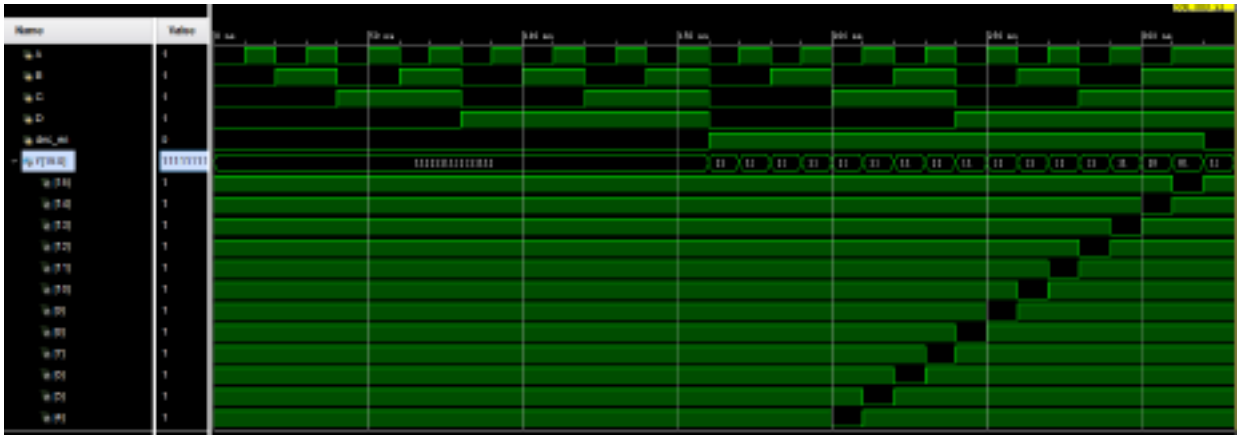
```
1   `timescale 1ns/1ps
2
3   module decode_74138_test();
4       reg A, B, C;
5       reg G1, G2a, G2b;
6       wire [7:0] Y;
7
8       decode_74138 dec(A, B, C, G1, G2a, G2b, Y);
9
10      initial begin
11          {G1, G2a, G2b} = 'b000;
12
13          while ({G1, G2a, G2b} < 'b111) begin
14              {C, B, A} = 'b000;
15
16              while ({C, B, A} < 'b111) begin
17                  #10 {C, B, A} = {C, B, A} + 1;
18              end
```

```
            #10 {G1, G2a, G2b} = {G1, G2a, G2b} + 1;
        end
        #10 $finish();
    end
endmodule
```



The 74138 decoder has three enable ports and should only be enabled when **{G1, G2a, G2b} = 'b100**, when the decoder is not enabled, no matter what the input is, the output keeps 'b1111_1111. Notice the part of waveform where **{G1, G2a, G2b} = 'b100**, we have {C, B, A} be the selection signals and the outputs are all as expected.

```
`timescale 1ns/1ps

module decode_4_16_test();
    reg A, B, C, D;
    reg dec_en;
    wire [15:0] Y;

    decode_4_16 dec(A, B, C, D, dec_en, Y);

    initial begin
        dec_en = 'b0;
        repeat(2) begin
            {D, C, B, A} = 'b0000;

            while ({D, C, B, A} < 'b1111) begin
                #10 {D, C, B, A} = {D, C, B, A} + 1;
            end

            #10 dec_en = dec_en + 1;
        end
        #10 $finish();
    end
endmodule
```



Similarly, when the decoder is not enabled, the output keeps 1111_1111_1111_1111, otherwise it fits the result we expected (one-hot).

Constraint File

```
set_property IOSTANDARD LVCMOS33 [get_ports {Y[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[8]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[9]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[10]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[11]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[12]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[13]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[14]}]
set_property IOSTANDARD LVCMOS33 [get_ports {Y[15]}]
set_property IOSTANDARD LVCMOS33 [get_ports A]
set_property IOSTANDARD LVCMOS33 [get_ports B]
set_property IOSTANDARD LVCMOS33 [get_ports C]
set_property IOSTANDARD LVCMOS33 [get_ports D]
set_property IOSTANDARD LVCMOS33 [get_ports dec_en]
set_property PACKAGE_PIN U5 [get_ports dec_en]
set_property PACKAGE_PIN T5 [get_ports D]
set_property PACKAGE_PIN T4 [get_ports C]
set_property PACKAGE_PIN R4 [get_ports B]
set_property PACKAGE_PIN W4 [get_ports A]
set_property PACKAGE_PIN A21 [get_ports {Y[0]}]
set_property PACKAGE_PIN E22 [get_ports {Y[1]}]
set_property PACKAGE_PIN D22 [get_ports {Y[2]}]
set_property PACKAGE_PIN E21 [get_ports {Y[3]}]
set_property PACKAGE_PIN D21 [get_ports {Y[4]}]
set_property PACKAGE_PIN G21 [get_ports {Y[5]}]
set_property PACKAGE_PIN G22 [get_ports {Y[6]}]
```

```
set_property PACKAGE_PIN F21 [get_ports {Y[7]}]
set_property PACKAGE_PIN J17 [get_ports {Y[8]}]
set_property PACKAGE_PIN L14 [get_ports {Y[9]}]
set_property PACKAGE_PIN L15 [get_ports {Y[10]}]
set_property PACKAGE_PIN L16 [get_ports {Y[11]}]
set_property PACKAGE_PIN K16 [get_ports {Y[12]}]
set_property PACKAGE_PIN M15 [get_ports {Y[13]}]
set_property PACKAGE_PIN M16 [get_ports {Y[14]}]
set_property PACKAGE_PIN M17 [get_ports {Y[15]}]
```

The left fig. shows when decoder is not enabled (the 5^{th} switch counting from right, a.k.a. dec_en is set as 0 now), outputs are all 1. The right one, dec_en = 1, A = B = C = D = 0, then Y[0] acts as the one-hot, since the output ports of 74138 is low-enabled, Y[0] = 0 and the rest 15 bits are 1.

(a) input as 0101, output as Y[5] = 0, rest are 1;

(b) input as 0110, output as Y[6] = 0, rest are 1;

(c) input as 0010, output as Y[2] = 0, rest are 1.

Task 2

Design File

```
module mux_74151(
    input mux_en,
    input s2, s1, s0,
    input d0, d1, d2, d3, d4, d5, d6, d7,
    output reg Y,
    output W
```

```
);
    always @ * begin
        if (!mux_en) begin
            case ({s2, s1, s0})
                'd0: Y <= d0;
                'd1: Y <= d1;
                'd2: Y <= d2;
                'd3: Y <= d3;
                'd4: Y <= d4;
                'd5: Y <= d5;
                'd6: Y <= d6;
                'd7: Y <= d7;
            endcase
        end
        else begin
            Y <= 'b0;
        end
    end

    assign W = ~Y;
endmodule
```

As the same of standard 74151, the enable signal is low-enabled, and there should have two output ports, where one is the selected input and the other one is its negation.

```
module func_df(
    input A, B, C, D,
    output Y
);
    assign Y = (~A & ~B & ~C & ~D) | (B & D) | (A & C) | (~B & C & ~D) | (~A & ~B & ~D);
endmodule
```

CD \ AB	00	01	11	10
00	1			
01		1	1	
11		1	1	1
10	1		1	1

From the K-map above (actually this is quite similar to checking the truth-table), we conclude:

```
case ({A, B, C})
    'b000: Y = ~D;
```

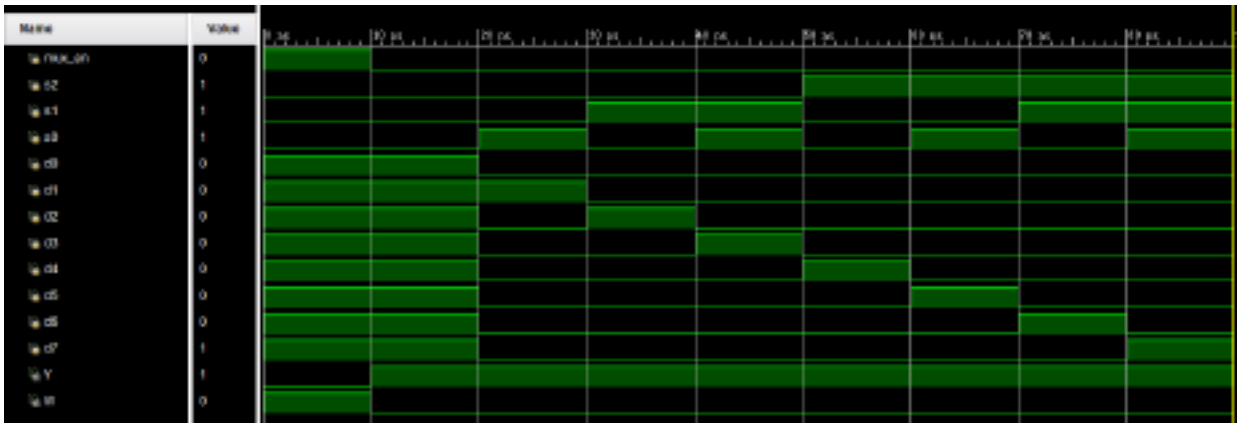
```
    'b001: Y = ~D;
    'b010: Y = D;
    'b011: Y = D;
    'b110: Y = D;
    'b111: Y = 1;
    'b100: Y = 0;
    'b101: Y = 1;
endcase
```

Thus we map this into our module mux 74151:

```
module func_1mux(
    input A, B, C, D,
    output Y
);
    mux_74151 mux(0, A, B, C,
                  ~D, ~D, D, D, 0, 1, D, 1,
                  Y);  // we don't use <W>, aka. ~Y
endmodule


module func_2mux(
    input A, B, C, D,
    output reg Y
);
    reg en1 = 1;
    reg en2 = 1;
    wire y1, y2;

    always @ * begin
        if (D) begin
            en1 <= 0;
            en2 <= 1;
            Y <= y1;
        end
        else begin
            en1 <= 1;
            en2 <= 0;
            Y <= y2;
        end
    end

    mux_74151 mux1(en1, A, B, C,
                   0, 0, 1, 1, 0, 1, 1, 1,
                   y1);  // D == 1

    mux_74151 mux2(en2, A, B, C,
                   1, 1, 0, 0, 0, 1, 0, 1,
                   y2);  // D == 0
endmodule
```

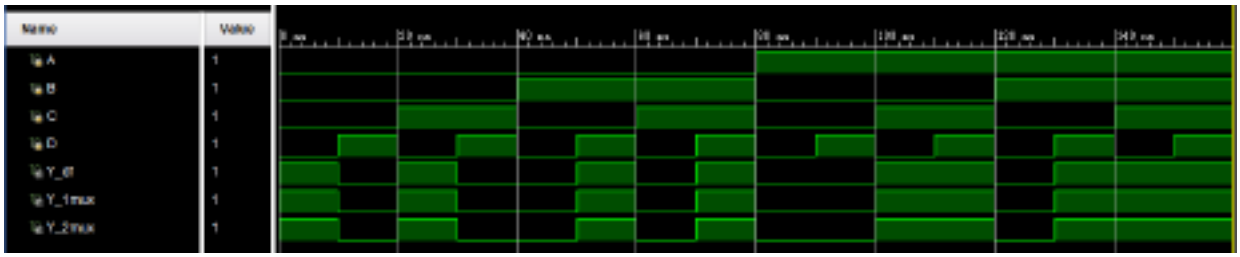
Simulation File

```
`timescale 1ns/1ps

module mux_74151_test();
    reg mux_en;
    reg s2, s1, s0;
    reg d0, d1, d2, d3, d4, d5, d6, d7;
    wire Y, W;

    mux_74151 mux(mux_en,
                  s2, s1, s0,
                  d0, d1, d2, d3, d4, d5, d6, d7,
                  Y, W);

    initial begin
        mux_en = 'b1;
        {s2, s1, s0} = 'b000;
        {d7, d6, d5, d4, d3, d2, d1, d0} = 'b1111_1111;

        #10 mux_en = 'b0;
        while ({s2, s1, s0} < 'b111) begin
            #10 {s2, s1, s0} = {s2, s1, s0} + 1;
            {d7, d6, d5, d4, d3, d2, d1, d0} = 'b0000_0001 << {s2, s1, s0};
        end
        #10 $finish();
    end
endmodule
```



When testing the mux itself, we just take 1111_1111 as an example input when it is not enabled (mux_en == 1). When it is enabled, since the mux will output the cooresponding input d$(x) where x = {s2, s1, s0}, and in the testbench we've let it to be one (while the others be zero), thus the output of mux, Y, should always be 1, and its negation, W, is always 0.

```
`timescale 1ns/1ps

module mux_func_test();
    reg A, B, C, D,
    wire Y_df, Y_1mux, Y_2mux;

    func_df df(A, B, C, D, Y_df);
    func_1mux one_mux(A, B, C, D, Y_1mux);
    func_2mux two_mux(A, B, C, D, Y_2mux);

    initial begin
        {A, B, C, D} = 'b0000;
        while ({A, B, C, D} < 'b1111) begin
            #10 {A, B, C, D} = {A, B, C, D} + 1;
        end
        #10 $finish();
    end
endmodule
```



First we notice that three ways to implement the function have the same output, then we check the truth-table and found all the outputs are right.

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1

1	1	1	1	1

Problems & Solutions

(1) At first I couldn't understand why 74138-decoder should has three enable ports, until detailed research *"The three enable pins of chip (in which Two active-low and one active-high) reduce the need for external gates or inverters when expanding."* (Components101, 2018)

(2) After happening the dilemma that variables went to be X again in the simulation, I have a better understanding of blocking and non-blocking assignments.