Hacettepe University

Computer Engineering Department

BM233 Logic Design Lab - 2020 Fall

Experiment 4 - Combinational Circuits in Verilog

December 5, 2020

Student name:
Deniz Gönenç

Student Number:
b21946146

1 Problem Definition

Designing and simulating 3x8 and 4x8 decoders using Verilog HDL.

A decoder is a combinational circuit that converts binary information from n binary inputs to a maximum of 2n unique output lines. A decoder provides the 2n minterms of n input variables.

Decoders are combinational circuits. Combinational circuits are circuits whose outputs, at any instant of time, depend only on the present inputs (the combinational circuits do not use any memory elements). That is, the previous inputs or state of the circuit do not have any effect on its present state.

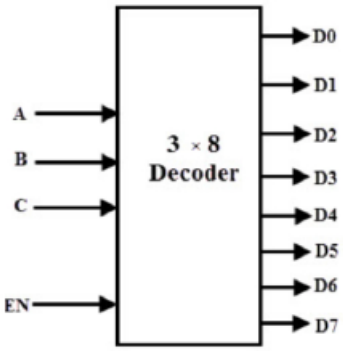

Figure 1: 3x8 decoder

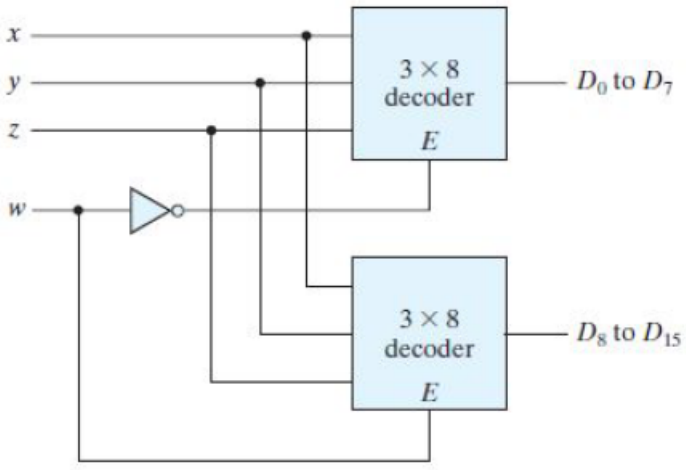

Figure 2: 4x16 decoder

2 Solution Implementation

2.1 3x8 decoder

I used four different variables for the inputs: a, b, c and en for enable. I used a net vector for the 8 outputs and assigned values to them with their respective boolean functions.

The code for the 3x8 decoder:

```
module decoder_3x8(a,b,c,en,d);
    // Declare input and output ports
    input a,b,c,en;
    output [7:0] d;

    // assign values to the outputs
    assign d[0] = en & ~a & ~b & ~c;
    assign d[1] = en & ~a & ~b & c;
    assign d[2] = en & ~a & b & ~c;
    assign d[3] = en & ~a & b & c;
    assign d[4] = en & a & ~b & ~c;
    assign d[5] = en & a & ~b & c;
    assign d[6] = en & a & b & ~c;
    assign d[7] = en & a & b & c;

endmodule
```

2.2 4x16 decoder

I used four different variables for the inputs: a, b, c and w. I used a net vector for the 16 outputs. I created two instances of 3x8 decoders.

The code for the 4x16 decoder:

```
`include "decoder_3x8.v"

module decoder_4x16(a,b,c,w,d);
    //  Declare  input  and  output  ports
    input a,b,c,w;
    output [15:0] d;

    // Instantiate two 3x8 decoders
    decoder_3x8 C1(.a(a), .b(b), .c(c), .en(w), .d(d[15:8]));
    decoder_3x8 C2(.a(a), .b(b), .c(c), .en(~w), .d(d[7:0]));

endmodule
```

3 Testbench Implementation

3.1 3x8 decoder

The first state I am testing for is when the enable is 0. I gave a, b and c arbitrary values for this one since we don't need to test them all.
The rest of the states go by binary order. We start with 000 and end with 111. For all of these states, enable is 1.

The code for 3x8 decoder's testbench:

```
1  `timescale 1ns / 1ps
2  `include "decoder_3x8.v"
3
4  module decoder_3x8_tb;
5      reg a,b,c,en; // Declaring inputs as regs
6      wire [7:0] d; // Declaring outputs as nets
7      decoder_3x8 UUT (.a(a), .b(b), .c(c), .en(en), .d(d)); //Instantiate UUT
8
9      initial // Generating stimuli
10         begin
11         en=0; a=1; b=1; c=1;
12         #100 en=1; a=0; b=0; c=0;
13         #100 en=1; a=0; b=0; c=1;
14         #100 en=1; a=0; b=1; c=0;
15         #100 en=1; a=0; b=1; c=1;
16         #100 en=1; a=1; b=0; c=0;
17         #100 en=1; a=1; b=0; c=1;
18         #100 en=1; a=1; b=1; c=0;
19         #100 en=1; a=1; b=1; c=1;
20         #100 $finish;
21     end
22 endmodule
```

3.2 4x16 decoder

The first 8 states I am testing for is when w is 0. The other inputs follow binary order. The last 8 states I am testing for is when w is 1. The other inputs follow binary order.

The code for 4x16 decoder's testbench:

```
`timescale 1ns / 1ps
`include "decoder_4x16.v"

module decoder_4x16_tb;
    reg a,b,c,w; // Declaring  inputs  as regs
    wire [15:0] d; // Declaring  outputs  as nets

    decoder_4x16 UUT (.a(a), .b(b), .c(c), .w(w), .d(d)); // Instantiate  UUT

    initial // Generate stimuli
        begin
            a=0; b=0; c=0; w=0;
            #25 a=0; b=0; c=1;
            #25 a=0; b=1; c=0;
            #25 a=0; b=1; c=1;
            #25 a=1; b=0; c=0;
            #25 a=1; b=0; c=1;
            #25 a=1; b=1; c=0;
            #25 a=1; b=1; c=1;
            #25 a=0; b=0; c=0; w=1;
            #25 a=0; b=0; c=1;
            #25 a=0; b=1; c=0;
            #25 a=0; b=1; c=1;
            #25 a=1; b=0; c=0;
            #25 a=1; b=0; c=1;
            #25 a=1; b=1; c=0;
            #25 a=1; b=1; c=1;
            #25 $finish;
    end

endmodule
```

4 Results

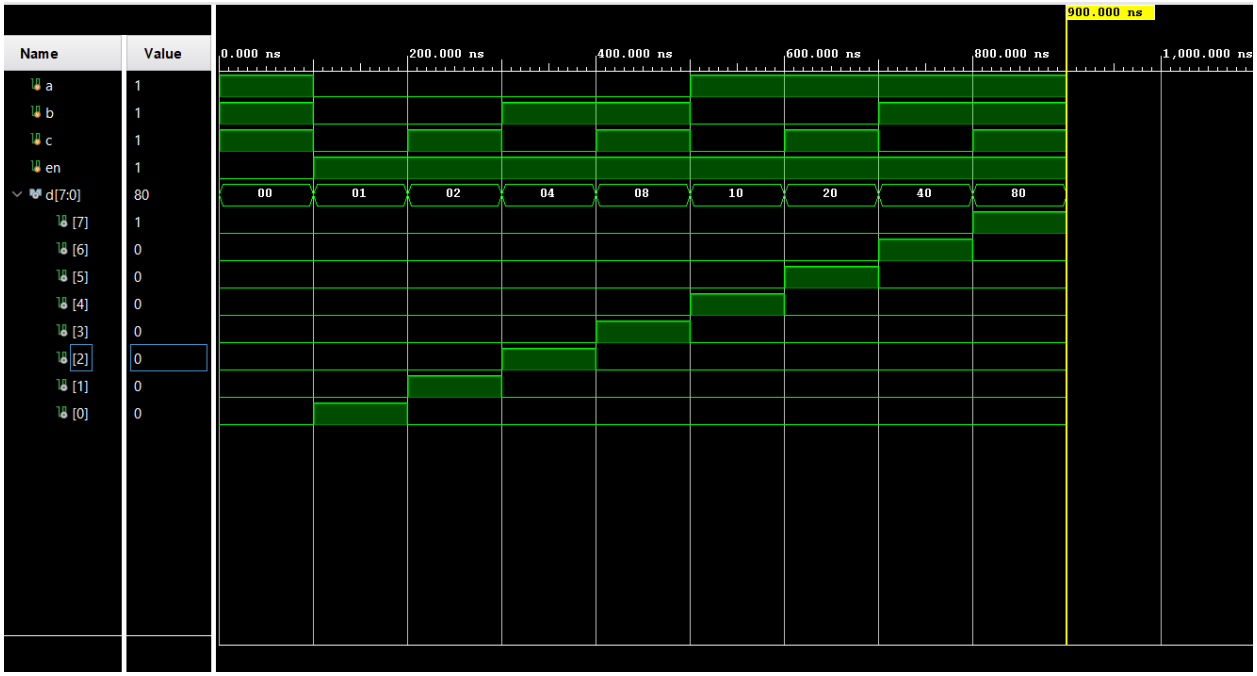

Figure 3: Resulting Waveform for the 3x8 decoder

The very first part we see is the case when enable is 0. When enable is 0, all outputs should be 0, as is here.
The next 8 parts are the 8 cases that follow binary order, while enable is 1.
My results are correct because it represents the truth table of the circuit.

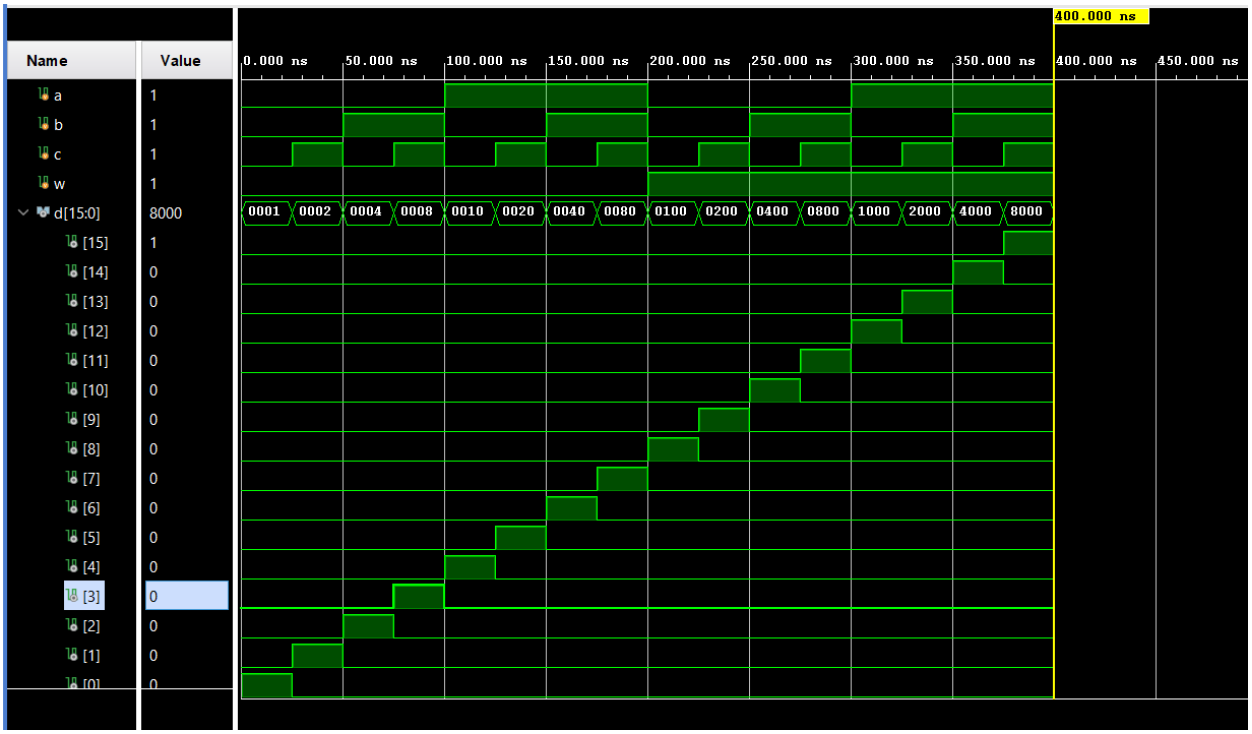

Figure 4: Resulting Waveform for the 4x16 decoder

The first 8 parts are the cases when w is 0, as we can see from its line. The other three inputs follow binary order.
The last 8 parts are the cases when w is 1. The other three inputs follow binary order.
My results are correct because it represents the truth table of the circuit.

5 Notes

By binary order I mean this: 000,001,010,011,100,101,110,111