Department of Computer and Software Engineering – ITU

SE201L: Digital Logic Design Lab

Course Instructor: Ms Aqsa Khalid	**Dated: 25-09-2024**
Lab Engineer: Muhammad Kashif	**Semester: Fall 2024**
Batch: BSSE 23 A	

LAB 6 Behavioral Modeling in Verilog: Implementation of Decoders

Name	Roll Number	Lab Marks

Checked on: _____

Signature: _____

Behavioral Modeling in Verilog: Implementation of Decoders

6.1. Introduction

This lab exercise introduces students to an important standardized combinational logic circuit: the *decoder*. The implementation of Boolean functions using decoders is detailed in theory and practiced in lab tasks using *behavioral modeling* style of Verilog. *Cascading* decoders to build larger sized decoders is also covered in one lab task.

6.2. Objectives

This lab exercise will enable students to achieve the following:

- Familiarize with the decoder structure and understand what is meant by the size of a decoder
- Design larger size decoders by cascading small size decoders
- Relate the output of decoders to the already learnt concepts of minterms and thus use decoders with a few extra gate circuitries to implement Boolean functions
- Learn how to use behavioral modeling style of Verilog

6.3. Conduct of Lab

1. This lab experiment has to be performed using the ModelSim PE Student Edition, installed in Embedded Lab PCs.
2. Bring printout of this lab manual when you come to perform the lab.
3. You can work and get evaluated in groups of two. However, manual submission has to be separate.
4. If there is difficulty in understanding any aspect of the lab, please seek help from the lab engineer or the TA.
5. If a lab task contains an instruction to show working to lab engineer, make sure that the lab engineer evaluates and marks on your manual for that task. If your manual is unmarked for this task, it can result in marks deduction.
6. Complete the lab within the allocated time. Late submissions will be marked zero.
7. Print the codes and screenshots of simulation results of all tasks and attach these to your lab manual. Submit complete manual to the lab engineer no later than 24 hours after the lab.

6.4. Theory and Background

6.4.1. Decoders

An n-bit binary number can be used to represent 2^n values. A binary decoder is a multiple-output combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If there are unused combinations in n-bit coded input, then there can be fewer than 2^n output lines.

Table 6.1 shows the truth table of the outputs of a 3-to-8 ($=2^3$) decoder that is shown in Figure 6.1:

Table 6.1: Truth table of a 3-to-8 decoder

Inputs			**Outputs**							
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

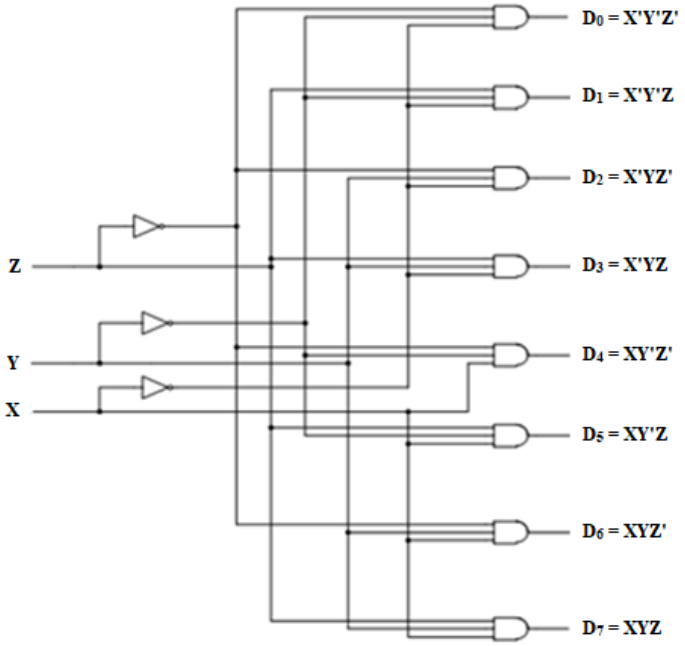


Figure 6.1: A 3-to-8 line decoder and its truth table

Only one output line of a binary decoder can be 1, while all other output lines remain zero. An n-to-2^n decoder actually provides all the minterms of the n-bit input. Outputs of a combinational logic circuit can be represented in sum-of-minterms form; hence decoder outputs (that are actually minterms) can be used to implement any combinational circuit by ORing the required minterms. Hence a combinational circuit with n-bit input and m-bit output can be implemented using an n-to-2^n decoder and m number of OR gates with multiple inputs.

6.4.1.1. Enable input and decoder cascading

A decoder may be provided with an additional input to enable or disable its operation. This additional input is called "enable" input. When this input is not enabled, all of the 2^n outputs of an n-bit decoder are 0. Decoders with enable inputs can be combined to build a larger decoder circuit. For example, two 2-to-4 line decoders can be connected together to give a 3-to-8 line decoder, as shown below:

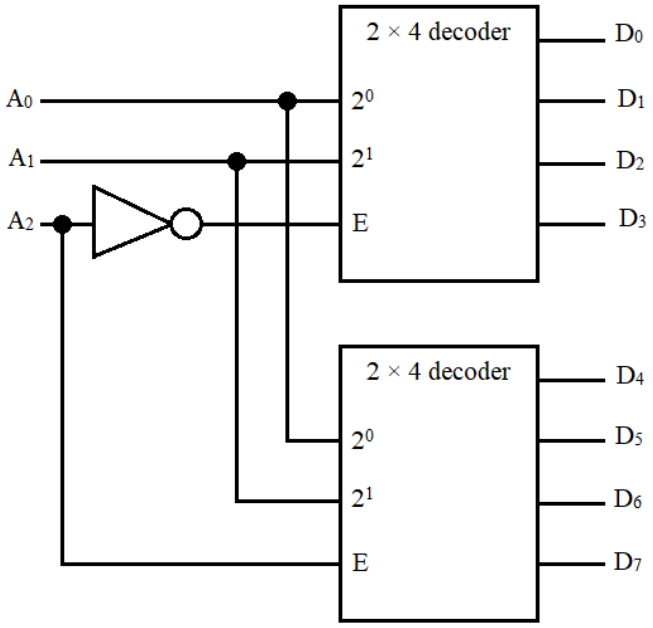


Figure 6.2: 3-to-8 line decoder using two 2-to-4 line decoder

6.4.2. Implementation of a decoder using dataflow modeling

A decoder can be implemented using dataflow modeling using the '?' conditional operator. The syntax of the conditional operator "?" is as follows:

```
condition ? value_if_true : value_if_false
```

Let's look at each line of the following code of a 2-to-4 decoder based on dataflow modeling:

```
module Decoder_2_to_4(X, Y, En, D);
input X, Y, En;
output [3:0] D;
assign D = En ? (X ? (Y ? 4'h8 : 4'h4) :(Y ? 4'h2 : 4'h1)): 4'h0;
endmodule
```

1. *module Decoder_2_to_4 (X, Y, En, D);* This line declares the module of 2-to-4 line decoder that has 4 ports X, Y, En and D.
2. *input X, Y, En;* This line declares X, Y and En as inputs of the decoder. X and Y are the inputs to be decoded and En is the enable of the decoder.
3. *output [3:0] D;* This line declares D as a vector output of size 4. This is the 4-bit decoded output.
4. *assign D = En ? (X ? (Y ? 4'h8 : 4'h4) :(Y ? 4'h2 : 4'h1)): 4'h0;* This line uses the conditional operator ? to check various conditions and assigns binary values to D accordingly. The first condition that is checked is the value of En input. If this is equal to 0, all bits of the output D are assigned 0. If En is equal to 1, then in another conditional operation, the value of X is checked for 0 or 1. For both values of X, Y is checked again and D is assigned 4-bit value accordingly. Note that the values assigned to D are written in hex, specified by the use of 'h. However, the number of bits that precedes 'h is still written as 4.
5. *endmodule* This line marks the end of this module.

This code is simulated using the following test bench from which the timing diagrams of Figure 6.3 are obtained.

```
module Test_decoder();
reg X, Y, En;
wire [3:0] D;
Decoder_2_to_4 my_dec(X, Y, En, D);
initial begin
En = 0; X = 0; Y = 0; #100
En = 0; X = 0; Y = 1; #100
En = 1; X = 1; Y = 0; #100
En = 1; X = 1; Y = 1;
end
endmodule
```

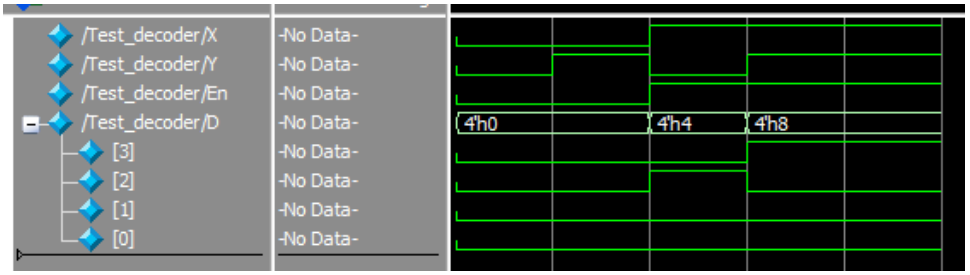


Figure 6.3: Timing diagram of 2-to-4 decoder

6.4.3. Behavioral modeling in Verilog

Behavioral modeling style lies at the highest level of abstraction in Verilog. It is similar to dataflow modeling in syntax. The only difference is the introduction of the notion of time and the sequence of code execution. Let's look at the following code that implements a 2-to-4 line decoder using behavioral modeling in Verilog:

```
module Decoder_2_to_4(X, Y, En, D);
input X, Y, En;
```

```
output reg [3:0] D;
always @ (X, Y, En) begin
case ({X,Y,En})
3'b001: D = 4'h1;
3'b011: D = 4'h2;
3'b101: D = 4'h4;
3'b111: D = 4'h8;
default: D = 4'h0;
endcase
end
endmodule
```

1. *module Decoder_2_to_4(X, Y, En, D);* This is the module declaration.
2. *input X, Y, En;* This line declares X, Y and En as scalar inputs.
3. *output reg [3:0] D;* This line declares a vector output D of 4 bits. Notice that the data type of output is not a net but a register. This has been done so that D can be updated inside an 'always' block. Only reg or integer data types can be stored with new values inside always or initial blocks.
4. *always @ (X, Y, En) begin* An 'always' block has been created in this line. An always block, like the initial block, is a timing block that is used in behavioral modeling. The always block runs all the time in contrast to the initial block that you have used previously in test benches, which runs only once. The always blocks needs to have trigger signals to let it know when to run. In our case there are three signals, X, Y and En. The always block is expected to be executed completely whenever one of these signals changes value. The list of trigger signals is called the sensitivity list of the always block. Similar to an initial block, the always block cannot drive wire data types either; only registers (reg) or integer data types can be driven inside an always block.
5. *case ({X, Y, En})* This line marks the start of a case statement inside our always block. Inside the parenthesis following the case keyword, is the expression that will be check against the different conditions listed inside this case statement. This expression can be an n-bit variable. In our case, we have used the braces to concatenate the three 1-bit variables X, Y and En. The order of values of X, Y and En will be retained in the concatenated expression.
6. *3'b001: D = 4'h1;* This line and the next three lines define four cases of the input expression {X, Y, En} and assign values to D accordingly.
7. *default: D = 4'h0;* This line defines the default case which will be executed when none of the above four have been found to match the input expression {X, Y, En}.
8. *endcase* This keyword marks the end of case statement.
9. *end* This keyword marks the end of the always block. This is the Verilog equivalent of the closing brace } in C and other programming languages.
10. *endmodule* This line marks the end of the module.

When tested with the test bench provided in 6.4.2, the simulation results are the same as in Figure 6.3.

6.5. Lab Tasks

6.5.1. Task 1: Cascading decoders [Marks: 15]

1. Write Verilog code for a 1-to-2-line decoder with enable input using the "?" operator of dataflow modeling. [2]

```
module decoder1to2 (
   input wire enable,
   input wire A,
   output wire Y0, Y1
);
   assign Y0 = (enable && ~A) ? 1'b1 : 1'b0;
   assign Y1 = (enable && A) ? 1'b1 : 1'b0;
endmodule
```

2. Write Verilog code for a 3-to-8-line decoder with enable input using behavioral modeling. [3]

```
module decoder3to8 (
   input wire enable,
   input wire [2:0] A,
   output reg [7:0] Y
);
   always @(enable, A) begin
      if (enable) begin
         case (A)
            3'b000: Y = 8'b00000001;
            3'b001: Y = 8'b00000010;
            3'b010: Y = 8'b00000100;
            3'b011: Y = 8'b00001000;
            3'b100: Y = 8'b00010000;
            3'b101: Y = 8'b00100000;
            3'b110: Y = 8'b01000000;
            3'b111: Y = 8'b10000000;
            default: Y = 8'b00000000;
         endcase
      end
      else
         Y = 8'b00000000;
   end
endmodule
```

3. Draw the design diagram of a 4-to-16-line decoder using two 3-to-8-line decoders and one 1-to-2-line decoder in the space given below: [2]

4. Write Verilog code that implements your design of 4-to-16-line decoder with enable input using the already coded 3-to-8-line and 1-to-2-line decoders in a hierarchical pattern. [2]

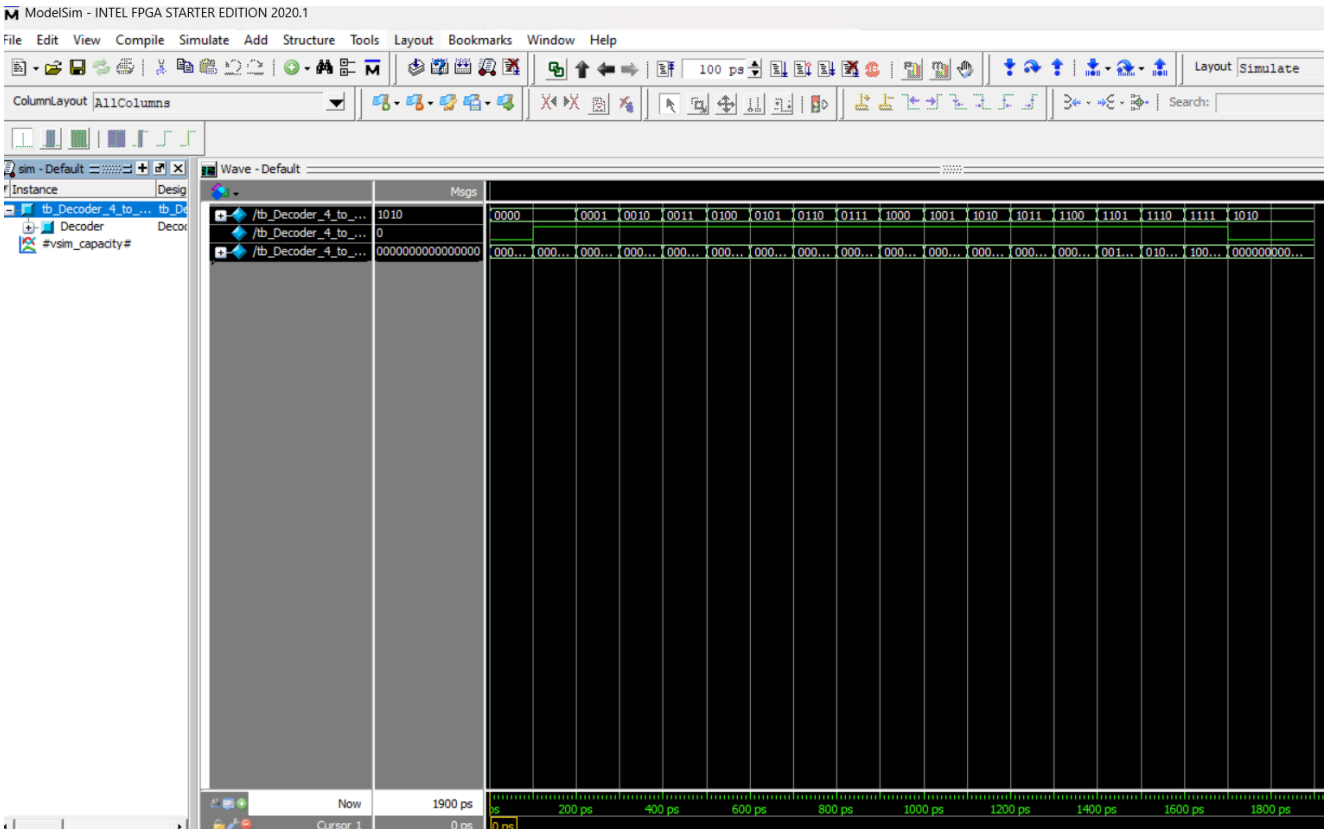
```
module Decoder_4_to_16(A, En, D);
  input [3:0] A;
  input En;
  output [15:0] D;
  wire En1, En2;
  decoder1to2 U1 (.A(A[3]), .En(En), .D({En2, En1}));
  decoder3to8 U2 (.A(A[2:0]), .En(En1), .D(D[7:0]));
  decoder3to8 U3 (.A(A[2:0]), .En(En2), .D(D[15:8]));
endmodule
```

5. Write test bench module to simulate your 4-to-16-line decoder for an arbitrary set (at least eight in number) of input combinations. Make sure to test the enable of your decoder. [2]

```
module tb_decoder4to16;
  reg [3:0] A;
  reg En;
  wire [15:0] D;
  decoder4to16 DUT (.A(A), .En(En), .D(D));
  initial begin

    En = 0; A = 4'b0000; #10;
    En = 1; A = 4'b0000; #10;
    En = 1; A = 4'b0001; #10;
    En = 1; A = 4'b0010; #10;
    En = 1; A = 4'b0011; #10;
    En = 1; A = 4'b0100; #10;
    En = 1; A = 4'b0101; #10;
    En = 1; A = 4'b0110; #10;
    En = 1; A = 4'b0111; #10;
    En = 1; A = 4'b1000; #10;
    En = 1; A = 4'b1001; #10;
    En = 1; A = 4'b1010; #10;
    En = 1; A = 4'b1011; #10;
    En = 1; A = 4'b1100; #10;
    En = 1; A = 4'b1101; #10;
    En = 1; A = 4'b1110; #10;
    En = 1; A = 4'b1111; #10;
    En = 0; A = 4'b1010; #10;
  end
endmodule
```

6. Show your code and simulation waveforms to the lab engineer to obtain credit. [2]
7. Take clear screenshots of the code of all modules of this task and the simulation timing waveforms, and paste them in a WORD file such that all fit on no more than two pages. Make sure that screenshots are legible.[2]



6.5.2. Task 2: Boolean function implementation using decoders [Marks: 15]

1. You are given a 4-variable Boolean function F.

$$F(A, B, C, D) = \sum m\,(0,\ 1,\ 3,\ 5,\ 7,\ 12,\ 11,\ 14,\ 15)$$

2. Implement this function in Verilog using the 4-to-16 line decoder constructed in Task 1. [6]

```
module Function_F(
    input wire [3:0] A,
    input wire En,
    output wire F
);
    wire [15:0] D;
    Decoder_4_to_16 decoder (
        .A(A),
        .En(En),
        .D(D)
    );
    assign F = D[0] | D[1] | D[3] | D[5] | D[7] | D[11] | D[12] | D[14] | D[15];
endmodule
```

3. Write a test bench module to simulate the above module for all input combinations of F. Complete the truth table of F in Table 6.2. [2]

```
module tb_Function_F;
    reg [3:0] A;
    reg En;
   wire F;
```

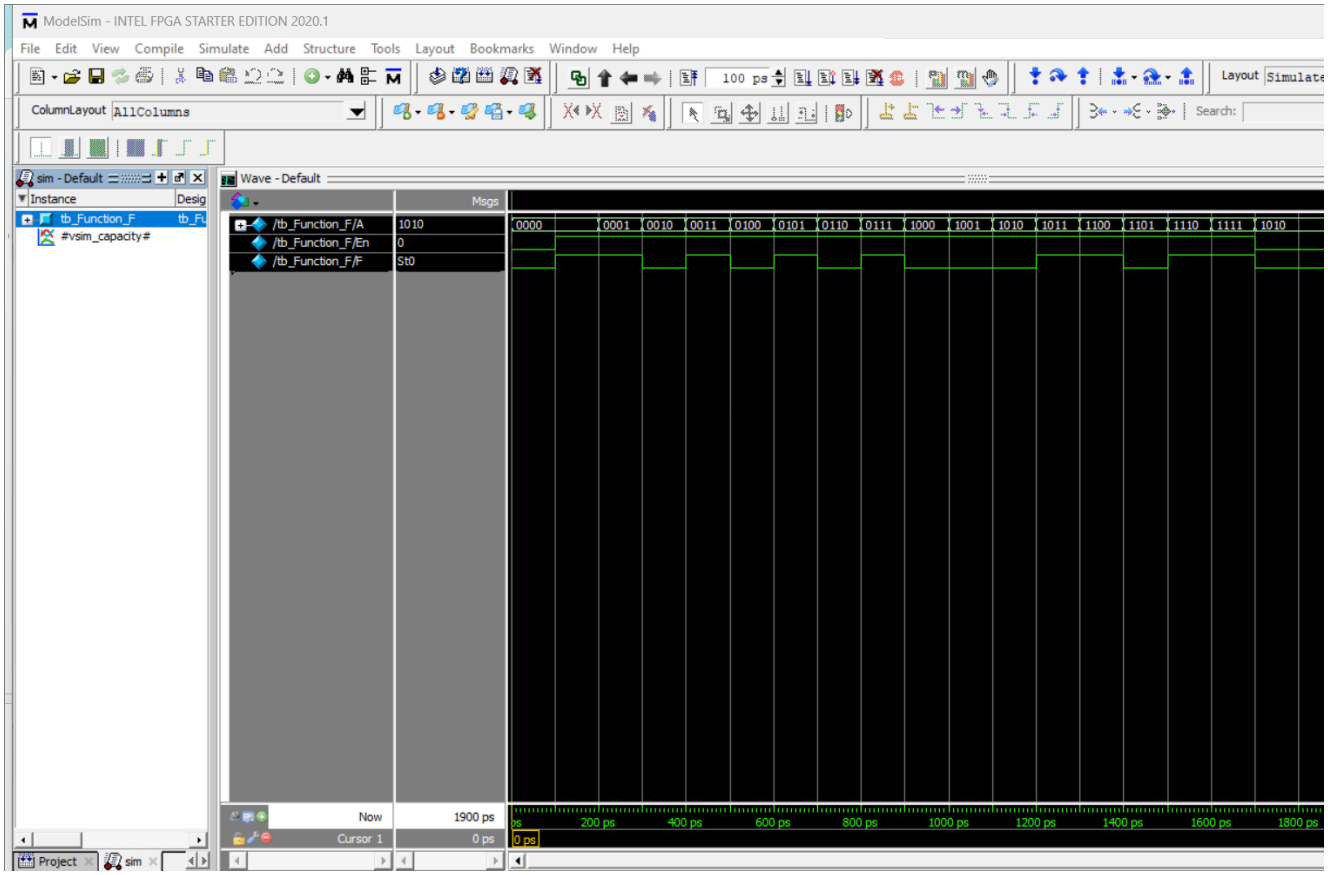
```
    Function_F DUT (.A(A), .En(En), .F(F));
    initial begin
        En = 0; A = 4'b0000; #100;
        En = 1; A = 4'b0000; #100;
        En = 1; A = 4'b0001; #100;
        En = 1; A = 4'b0010; #100;
        En = 1; A = 4'b0011; #100;
        En = 1; A = 4'b0100; #100;
        En = 1; A = 4'b0101; #100;
        En = 1; A = 4'b0110; #100;
        En = 1; A = 4'b0111; #100;
        En = 1; A = 4'b1000; #100;
        En = 1; A = 4'b1001; #100;
        En = 1; A = 4'b1010; #100;
        En = 1; A = 4'b1011; #100;
        En = 1; A = 4'b1100; #100;
        En = 1; A = 4'b1101; #100;
        En = 1; A = 4'b1110; #100;
        En = 1; A = 4'b1111; #100;
        En = 0; A = 4'b1010; #100;
    end
endmodule
```

Table 6.2: Truth table of F as observed in decoder implementation

A	B	C	D	F
0	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	0	0	0	
1	0	0	1	
1	0	1	0	
1	0	1	1	
1	1	0	0	
1	1	0	1	
1	1	1	0	
1	1	1	1	

4. Show your code and simulation waveforms to the lab engineer to obtain credit. [4]

5. Take clear screenshots of the code of all modules of this task and the simulation timing waveforms, and paste them in your WORD file such that all fit on no more than two pages. Make sure that screenshots are legible. [3]



6.5.3. Analysis [Marks: 5]

1. How is the size of a decoder specified? Give examples. [1]

8. Search the internet for two ICs of 7400 series that implement a decoder. Also mention the decoder ICs along with their names (numbers). [1]

9. How can you use a 2-to-4 decoder as a 1-to-2 decoder? Draw circuit diagram to support your answer. [2]

10. How many 4-to-16 decoders will be required to implement a 6-to-64 decoder? [1]

Assessment Rubric

Method:
Lab report evaluation and instructor observation during lab sessions.

Performance	CLO	Able to complete the tasks over 80% (4-5)	Able to complete the tasks 50 – 80% (2-3)	Tasks completion below 50% (0-1)	Marks
1. Teamwork	1	Actively engages and cooperates with other group members in an effective manner	Cooperates with other group members in a reasonable manner	Distracts or discourages other group members from conducting the experiments	
2. Laboratory safety and disciplinary rules	1	Observes lab safety rules; handles the development board and other components with care and adheres to the lab disciplinary guidelines aptly	Observes safety rules and disciplinary guidelines with minor deviations	Disregards lab safety and disciplinary rules	
3. Realization of experiment	3	Conceptually understands the topic under study and develops the experimental setup accordingly	Needs guidance to understand the purpose of the experiment and to develop the required setup	Incapable of understanding the purpose of the experiment and consequently fails to develop the required setup	
4. Conducting experiment	3	Sets up hardware/software properly according to the requirement of experiment and examines the output carefully	Makes minor errors in hardware/software setup and observation of output	Unable to set up experimental setup, and perform the procedure of experiment	
5. Data collection	3	Completes data collection from the experiment setup by giving proper inputs and observing the outputs, complies with the instructions regarding data entry in manual	Completes data collection with minor errors and enters data in lab report with slight deviations from provided guidelines	Fails at collecting data by giving proper inputs and observing output states of experiment setup, unable to fill the lab report properly	

6. Data analysis	3	Analyzes the data obtained from experiment thoroughly and accurately verifies it with theoretical understanding, accounts for any discrepancy in data from theory with sound explanation, where asked	Analyzes data with minor error and correlates it with theoretical values reasonably. Attempts to account for any discrepancy in data from theory	Unable to establish the relationship between practical and theoretical values and lacks the theoretical understanding to explain any discrepancy in data	
7. Computer use	3	Successfully uses lab PC and internet to look for relevant datasheets, carry out calculations, or verify results using simulation	Requires assistance in looking for IC datasheets and carrying out calculation and simulation tasks	Does not know how to use computer to look up datasheets or carry out calculation and simulation tasks	
				Total (out of 35)	