**Birzeit University – Faculty of Information Technology**

**Computer Systems Engineering Department**

**Digital Lab ENCS 211**

**(Report EXP. No. 3)**

**Encoders , Decoders , Multiplexers and**

**Demultiplexers**

**1) Name: Ahmad Mafarjah ID: 1080429**

**2) Name: Laith Samara ID: 1080758**

# Instructor: Dr. Iyad Tumar.

# Assistant: Eng. Anas Abd-elrazeq.

**Procedure:-**

**A. Constructing 4-to-2 Line Encoder with Basic Gates**

An Encoder is a digital circuit that accepts one or multiple inputs and generates a specific output code. Only one input can be active at a time. If tow inputs are active, it will produce an undefined output. The 4-to-2 Line Encoder is shown below in Figure 8.

**Figure 8: 4-to-2 Line Encoder with Basic Gates.**

After we applied +5 VDC from the fixed power and connect the required inputs and outputs connection and followed the input sequence in the following table, we obtained on these output states.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **D** | **C** | **B** | **A** | **F9** | **F8** |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 |

As we noticed, the circuit displayed result when only one input active. When two or multiple inputs were active, the result was undefined.

**B. Constructing 4-to-2 Priority Encoder**

A priority Encoder is an encoder circuit but with a priority function. The priority encoder works such that if two or more inputs are equal to 1, the input having the highest priority will be active. Sometimes, in addition to the tow outputs, the circuit had a third output which is called a valid bit indicator that is set to 1 when one or more inputs are equal to 1. A common use of priority encoders is for interrupt controllers. The following figures show 4-to-2 priority encoder and its implementation using NAND gates only.



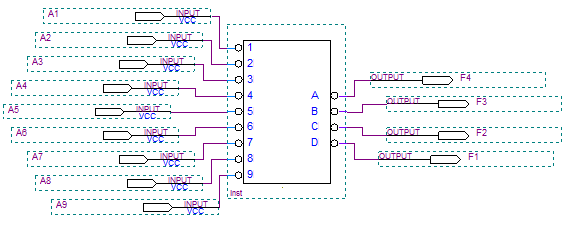
**4-to-2 Priority Encoder**

**4-to-2 Priority Encoder using NAND gates only**

After we implemented the circuit using NAND gates only and applied the following input sequence, we obtained these results (shown in the table below).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| D0 | D1 | D2 | D3 | X | Y |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

**C. Constructing 10-to-4 Line Encoder with TTL IC**



**(74147) BCD Priority Encoder**

After we connected the previous circuit and applied the following inputs , we get these outputs (Shown in the table below).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **A9** | **A8** | **A7** | **A6** | **A5** | **A4** | | **A3** | **A2** | | **A1** | | **F4** | **F3** | | **F2** | **F1** |
| 0 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | | 1 | | 0 | 1 | | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | | 1 | 1 | | 1 | | 0 | 1 | | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 1 | | 0 | | 0 | 1 | | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | | 1 | 0 | | 0 | | 1 | 0 | | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | | 0 | 1 | | 1 | | 0 | 0 | | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | | 0 | 0 | | 0 | | 1 | 1 | | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | | 1 | 1 | | 1 | | 0 | 1 | | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | | 0 | 1 | | 1 | | 0 | 1 | | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 1 | | 1 | 0 | | 0 | | 1 | 0 | | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | | 1 | 1 | | 0 | | 0 | 0 | | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | | 1 | 1 | | 1 | | 0 | 0 | | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | | 0 | 1 | | 1 | 1 | 1 | | 1 | 1 | | 0 |

**D. Constructing 2-to-4 Line Decoder with Basic Gates**

A decoder is a logic circuit that converts a parallel binary number (input) to a binary signal (output) that indicates absence of that specific number. The AND gate can be used as a basic decoder circuit (As in the following figure 9).

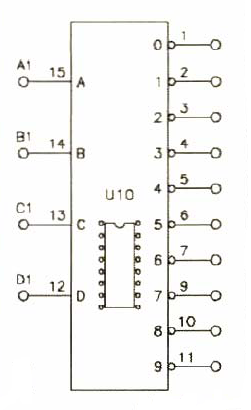


**Figure 9: 2-to-4 Decoder**

After we connected the required connection and applied the following sequence , we obtained these results (Shown in the following table).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **B** | **A** | **F1** | **F2** | **F3** | **F4** |
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 |

**E. Constructing 4-to-10 Line Decoder with TTL IC**



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **BCD** | **D** | **C** | **B** | **A** | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 2 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 6 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 7 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

**F. Constructing BCD-to-7-Segment Decoder**

A 7-segment decoder is used to display any the decimal digits (from 0 to 9). When the LT (Lamp Test) set to ‘0’, all the output ‘8’.LT is an active low input , so when the LT set to ‘0’ , the output will be ‘8’ for all inputs , which indicates that all the segments of the display are in a good status and working.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **D** | **C** | **B** | **A** | **Display Pattern** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | X |
| 1 | 0 | 1 | 1 | X |
| 1 | 1 | 0 | 0 | X |
| 1 | 1 | 0 | 1 | X |
| 1 | 1 | 1 | 0 | X |
| 1 | 1 | 1 | 1 | X |

After number 9 , the 7-Segment displayed an undefined output (or don’t care).

**G. Constructing 2-to-1 Line Multiplexer with Basic Gates**

A multiplexer , or MUX , is a logic circuit that select binary information from many input lines and send them to a single output. One of the multiple inputs is selected by the selector gates (The number of these gates determine the MUX capacity) and is routed to the single output.

A 2-to-1-Line multiplexer has two data input lines , one output line , and one selection line. (See figure 10)



**Figure 10: 2-to-1 Line Multiplexer**

When C = 0, the input B determines the output.

When C = 1, the input A determines the output.

|  |  |  |  |
| --- | --- | --- | --- |
| **C** | **B** | **A** | **F3** |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

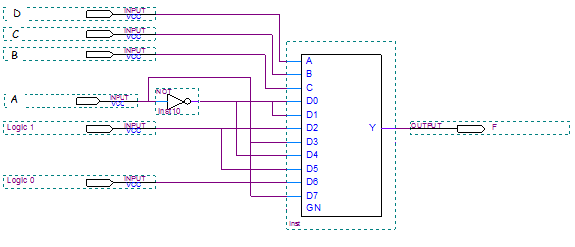
**H. Using Multiplexer to Create Function**

It’s a more efficient method to implement a Boolean function of 4 variables with a multiplexer that has 3 selection lines. The first 3 variables of the function are connected to the selection inputs of the multiplexer. The remaining single variable of the function is used for the data inputs. We denoted A as a single variable and obtained the following results.

**We implemented this function in the experiment :- F (D , C , B , A) = Σ(0,2,4,5,7,8,10,11,15)**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **D** | **C** | **B** | **A** | **F** | |
| 0 | 0 | 0 | 0 | 1 | **A’** |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | **A’** |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | **1** |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | **A** |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | **A’** |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | **1** |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | **0** |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | **A** |
| 1 | 1 | 1 | 1 | 1 |

The following diagrame represent the required MUX to create the previous function .

**Multiplexer to implement the function : F (D,C,B,A) = Σ(0,2,4,5,7,8,10,11,15)**

**I. Constructing 8-to-1 Line Multiplexer with TTL IC**

A Demultiplexers, or DMUX, is a logic circuit that is opposite of multiplexer. DMUX has a single input and multiple outputs. The input signal generates to the output through the selector terminal.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **C** | **B** | **A** | **Y** | **F** |
| 0 | 0 | 0 | D0 | D0’ |
| 0 | 0 | 1 | D1 | D1’ |
| 0 | 1 | 0 | D2 | D2’ |
| 0 | 1 | 1 | D3 | D3’ |
| 1 | 0 | 0 | D4 | D4’ |
| 1 | 0 | 1 | D5 | D5’ |
| 1 | 1 | 0 | D6 | D6’ |
| 1 | 1 | 1 | D7 | D7’ |

**J. Constructing 1-to-2 Line Demultiplexers with Basic logic gates**



|  |  |  |  |
| --- | --- | --- | --- |
| **A** | **C** | **F1** | **F2** |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**K. Constructing 1-to-8 Line Demultiplexers with CMOS IC**

****

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **C** | **B** | **A** | **Y0** | **Y1** | **Y2** | **Y3** | **Y4** | **Y5** | **Y6** | **Y7** |
| 0 | 0 | 0 | D | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | D | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | D | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | D | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | D | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | D | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | D | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | D |

**------------------------- CONCLUSION -------------------------**

At the End, we understood the operating principles of Encoders, Decoders, Multiplexers and Demultiplexers. We concluded that we can construct Encoders, Decoders, Multiplexers and Demultiplexers using basic gates and IC. Also we can design some simple circuits and use them in designing other complex circuits.

If we have a decoder with n inputs, then we can get 2^n outputs with only one active output (one of them is ‘1’).

We can implement any function using an appropriate decoder , mux and other circuits.

**3.6 PROBLEM:-**

Design a Majority Circuit; a circuit takes 4 inputs and outputs 1 output. Its output equals 1 when 3 or 4 of the inputs are 1. **You can only use two 4\*1 multiplexers.**

**Solution:**

Let A, B, C and D be the 4 inputs, F is the output. When 3 or 4 of these inputs are 1, the output F will be 1. Let’s we see the truth table for this problem.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **F** |  |
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | CD |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | CD |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | C+D |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

From the truth table, we conclude that D0 = 0, D1 = D2 = CD, D3 = C+D and A, B are selection lines. Now, we can built the circuit using tow multiplexers, AND gate and 2 OR gates. The Design is shown below.



**Pre Lab Solution:-**

**Question #1:-** Design, construct, and test a circuit which uses an SN74151 to implement a sum-of-products expression.

**a)** Convert the following expression into summation: **Y = F (A, B, C) = AB’ + B’C**

**Answer:** F (A, B, C) = AB’ + B’C

= AB’ (C+C’) + B’C (A+A’)

= AB’C + AB’C’ + AB’C + A’B’C

= A’B’C + AB’C’ + AB’C

= ∑ (1, 4, 5)

**b)** Sketch on 8-to-1 Multiplexerthe input connections necessary to implement the function in (**a).**

**Answer:**



**c)** Refer to function in **(a)** to fill in the table below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **INPUTS**  **A B C** | | | **OUTPUTS**  **Y Y’** | |
| L | L | L | 0 | 1 |
| L | L | H | 1 | 0 |
| L | H | L | 0 | 1 |
| L | H | H | 0 | 1 |
| H | L | L | 1 | 0 |
| H | L | H | 1 | 0 |
| H | H | L | 0 | 1 |
| H | H | H | 0 | 1 |

**Question #2:- Design**, construct, and test a circuit which uses an SN74138 Demultiplexers to implement a sum-of-products expression.

**a)** Convert the following expression into summation: **Y = F(A , B , C) = A’BC + BC’**

**Answer:** F (A, B, C) = A’BC + BC’

= A’BC + BC’ (A+A’)

= A’BC +ABC’ + A’BC’

= ∑ (2, 3, 6)

**b)** The Demultiplexers output is selected, and will go low, by the address on inputs A, B, and C when the IC is enabled. Therefore, we can create the output function Y by summing together the outputs indicated by the summation form of Expression 1.2. Since the outputs of the Demultiplexers are active- low, this is done with a NAND gate. Connect each of the TRUE minterms outputs of the Demultiplexers in Figure.7 (indicated by the summation equation) to an input of the NAND gate. Connect all unused NAND inputs to logic 1.

