Theory of Logic Circuits

Laboratory manual

Exercise 11

Implementing Logic Functions Using MSI Multiplexers and Demultiplexers

1. Multiplexer and Demultiplexer Overview

1.1. Multiplexer

A Multiplexer (MUX) is a circuit that is used to direct one out of 2^n inputs to a single output. It is also called Data Selector because n-input select lines are used to select one of the input signals and direct it to the output. A block diagram of 4-to-1 line Multiplexer with enable input is shown in Fig. 1 and truth table for it in Fig. 2.

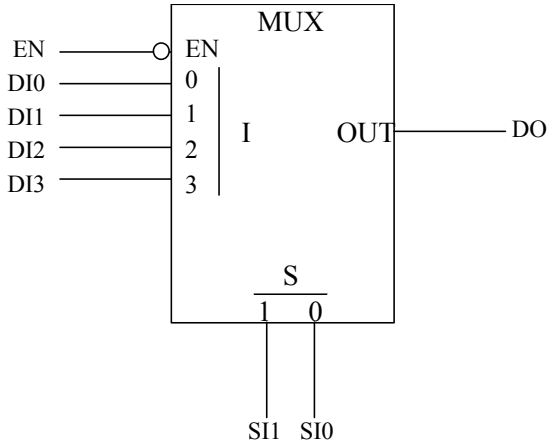

Fig. 1. Block diagram of a 4-to-1 line Multiplexer

EN	DI 0	DI 1	DI 2	DI 3	SI1	SI0	DO
1	X	X	X	X	X	X	0
0	0	X	X	X	0	0	0
0	1	X	X	X	0	0	1
0	X	0	X	X	0	1	0
0	X	1	X	X	0	1	1
0	X	X	0	X	1	0	0
0	X	X	1	X	1	0	1
0	X	X	X	0	1	1	0
0	X	X	X	1	1	1	1

Fig. 2. Truth table of a 4-to-1 line Multiplexer

Since the enable input EN has negation symbol at its input in the block diagram, 0 enables device and 1 disables the device. Since the output line has no negation symbol, when the device is disabled, the output signal DO is 0. When the device is enabled, the device output signal DO follows the data input signal selected by select lines SI1 and SI0. Fig. 3 shows the functional logic diagram for 4-to-1 MUX with enable input.

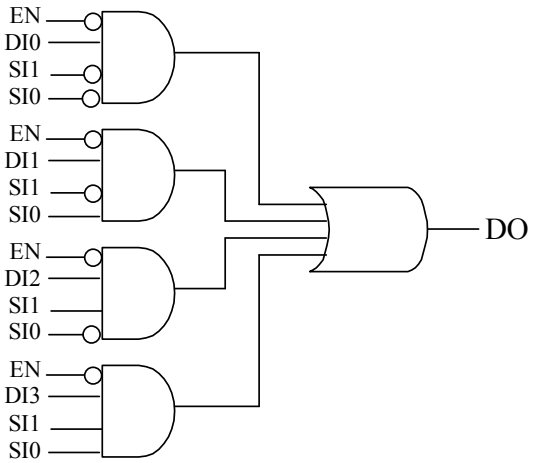

Fig. 3. Functional logic diagram for a 4-to-1 MUX with an enable input

Smaller Multiplexers can be connected together to obtain larger configuration. Fig. 4 illustrates in block diagram form how five 4-to-1 Multiplexers can be connected to obtain 16-to-1 Multiplexer.

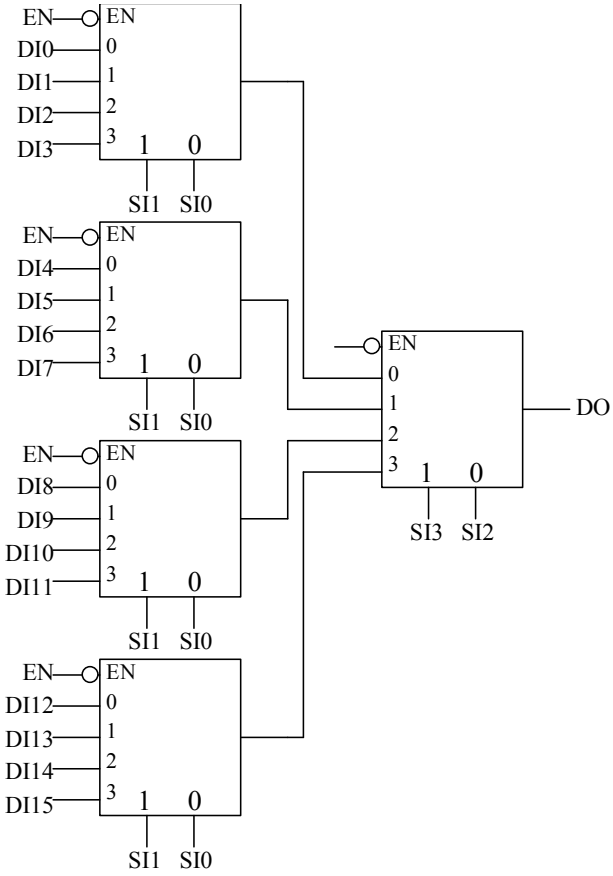

Fig. 4. A 16-to-1 line Multiplexer constructed from five 4-to-1 line Multiplexers

1.2. Demultiplexer

A Demultiplexer (DMUX) is a circuit that receives a signal on a single input line and directs that signal to one of 2^n possible output lines. If the enable input is active, we can also call this circuit a n-to2^n Decoder. A block diagram of 1-to-4 line Demultiplexer is shown in Fig. 5a. A functional logic diagram for the device is shown in Fig. 5b and truth table for it is shown in Fig. 6.

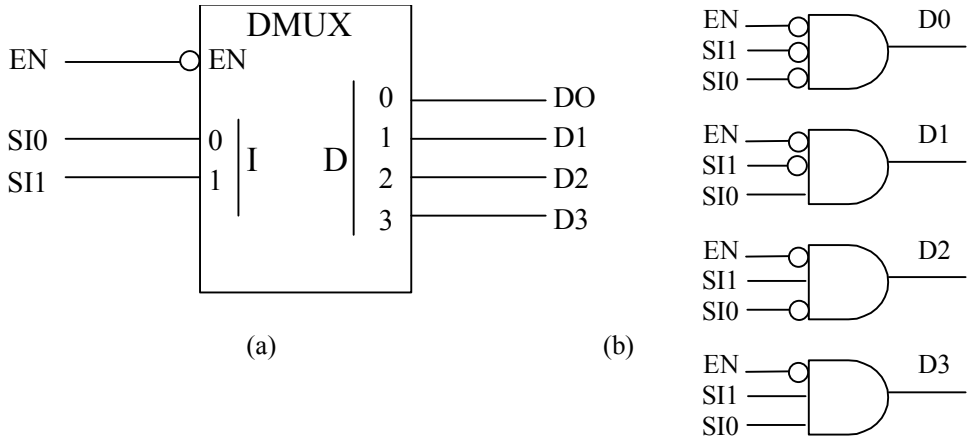

Fig. 5. Block diagram for a 1-to-4 line Demultiplexer (a) and its functional logic diagram (b)

EN	SI1	SI0	D0	D1	D2	D3
1	X	X	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1

Fig. 6. Truth table of a 1-to-4 line Demultiplexer

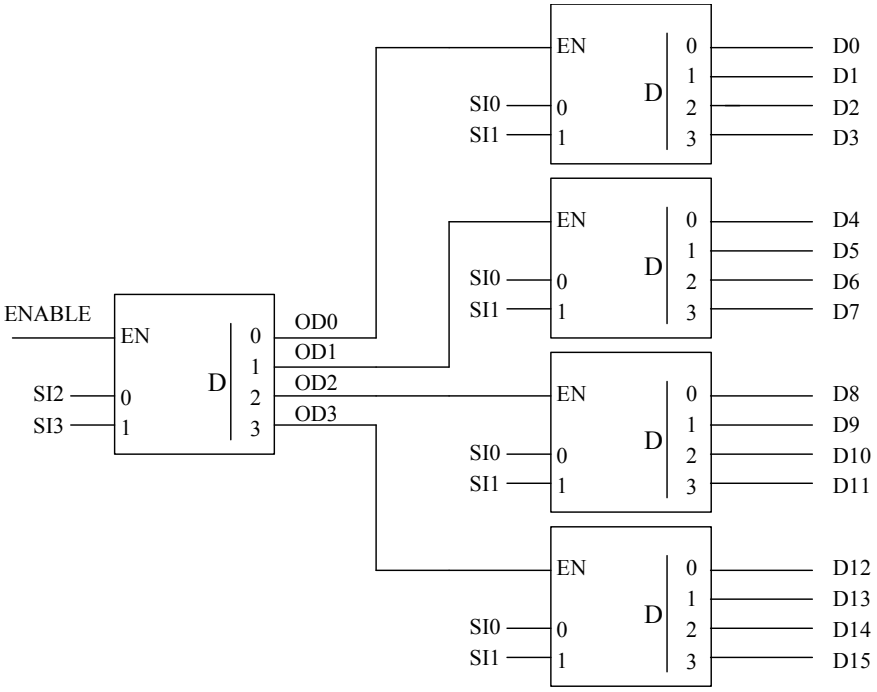

Fig. 7. A 1-to-16 line Demultiplexer constructed from five 1-to-4 line Demultiplexers

Like Multiplexers, smaller Demultiplexers can also be connected together to obtain larger configuration. Fig. 7 illustrates in block diagram form how five 1-to-4 line Demultiplexers can be connected to obtain a 1-to-16 line Demultiplexer. Observe that ENABLE signal is activated when input line is high (instead of Demultiplexer shown in Fig. 5).

2. Implementing Logic Functions Using MSI Multiplexers

When a Multiplexer is used to implement a logic function, that function does not need to be minimized in the normal manner; however, a minimized function consisting of only a single literal or a single product term would be more cost-effective using a gate-lever design. Usually logic designers use truth tables to implement logic functions with Multiplexers. To illustrate the process, the following function will be implemented using a Multiplexer:

$$F(a,b,c,d)=\Sigma(4,5,6,7,10,14)_{abcd}$$

2.1. Implementing for Type 0.

A type 0 MUX design require no signal in the truth table representing the function to be partitioned off. This means, that all signals are applied to the select inputs. Also the characteristic numbers of the function are applied to the data inputs. For the specified function listed above, the characteristic numbers are $f_0=0$, $f_1=0$, $f_2=0$, $f_3=0$, $f_4=1$, $f_5=1$, $f_6=1$, $f_7=1$, $f_8=0$, $f_9=0$, $f_{10}=1$, $f_{11}=0$, $f_{12}=0$, $f_{13}=0$, $f_{14}=1$, $f_{15}=0$ (see Fig. 9a).

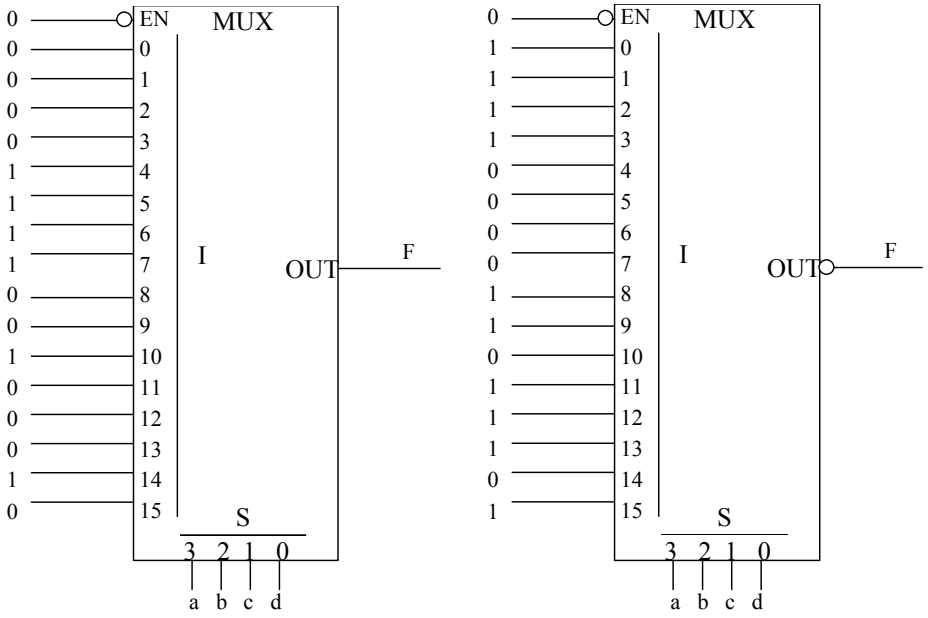

Fig. 9. Implementation for type 0 16-to-1 Multiplexer design for function F

If a Multiplexer has negation on the output an Invertor must be included on the output of the Multiplexer or characteristic numbers applied to data inputs must be complemented (see Fig. 9b).

2.2. Implementing for Type 1.

A type 1 MUX design requires one signal to be partitioned off. Continuing with the same function F, the truth table is now drawn and partitioned as shown in Fig. 10.The select inputs are independent signals a,b,c, and the data inputs are the values (the subfunctions) in the truth table. The reduced output column in Fig. 10 represents the output states as function of the partitioned-off input signal d. The reduced outputs in the truth table consist of a set of subfunctions of the signal that is partitioned off in the truth table. In this case, eight subfunctions are required where each subfunction is written as function of d, that is, F(d). The implementation for type 1 Multiplexer design is shown in Fig. 11.

a	b	c	d	F	F
0	0	0	0	0	0
0	0	0	1	0	
0	0	1	0	0	0
0	0	1	1	0	
0	1	0	0	1	1
0	1	0	1	1	
0	1	1	0	1	1
0	1	1	1	1	
1	0	0	0	0	0
1	0	0	1	0	
1	0	1	0	1	$\overline{d}$
1	0	1	1	0	
1	1	0	0	0	0
1	1	0	1	0	
1	1	1	0	1	$\overline{d}$
1	1	1	1	0	

Fig. 10. Truth table partitioned for a type 1 MUX design.

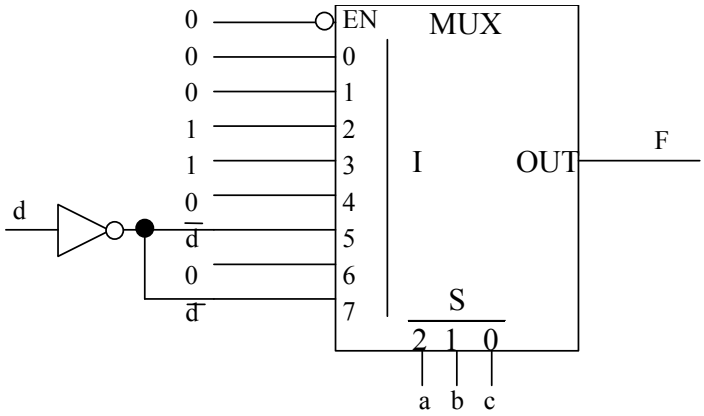

Fig. 11. Implementation for a type 1 MUX design.

2.3. Implementing for Type 2 and 3.

Fig. 12a and b illustrate truth table partitioning for a type 2 and 3 Multiplexer design respectively. The two signals c and d are partitioned off for a type 2 MUX design, while the three signals b, c, and d are partitioned off for a type 3 MUX design. Fig. 13 a and b show the implementation for a type 2 and type 3 MUX design.

a	b	c	d	F	F
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	0
0	0	1	1	0	
0	1	0	0	1	
0	1	0	1	1	
0	1	1	0	1	1
0	1	1	1	1	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	$c\bar{d}$
1	0	1	1	0	
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	1	$c\bar{d}$
1	1	1	1	0	

(a)

a	b	c	d	F	F
0	0	0	0	0	
0	0	0	1	0	
0	0	1	0	0	
0	0	1	1	0	
0	1	0	0	1	b
0	1	0	1	1	
0	1	1	0	1	
0	1	1	1	1	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	
1	0	1	1	0	
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	1	$c\bar{d}$
1	1	1	1	0	

(b)

Fig. 12. Truth table partitioned for a type 2 (a) and a type 3 (b) MUX design.

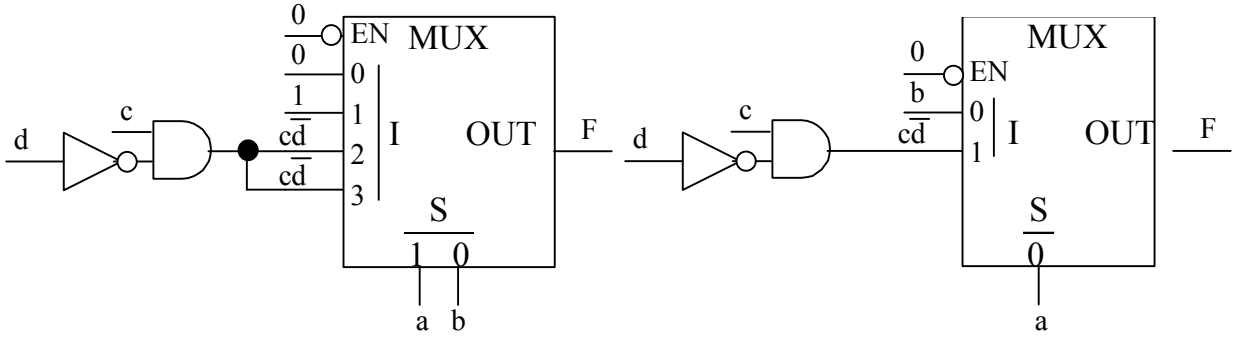

Fig. 13. Implementation for a type 2 (a) and type 3 (b) MUX design.

3. Implementing Logic Functions Using MSI Demultiplexers

Since Demultiplexers do not suffer from a lack of outputs like Multiplexers do, Demultiplexer implementations are normally carried out for Boolean functions with multiple outputs. Using a Demultiplexer with n select inputs with an external OR elements connected to the appropriate outputs of a Demultiplexer any Boolean function of n variables can be implemented. If k Boolean functions are to be implemented using a Demultiplexer, k external OR elements can be required.

3.1. Example

Design a 4-bit binary-to-gray code converter with the truth table shown in Fig. 14.

b3	b2	b1	b0	g3	g2	g1	g0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Fig. 14. The truth table a 4-bit binary-to-gray code converter.

We can write the Boolean functions for the outputs of the code converter in terms of the inputs as follows.

$$g3=\Sigma(8,9,10,11,12,13,14,15)_{b3b2b1b0}$$
$$g2=\Sigma(4,5,6,7,8,9,10,11)_{b3b2b1b0}$$
$$g1=\Sigma(2,3,4,5,10,11,12,13)_{b3b2b1b0}$$
$$g0=\Sigma(1,2,5,6,9,10,13,14)_{b3b2b1b0}$$

Since there are four bits of input information required to determine the minterms, a 1-to-16 line Demultiplexer is utilized as shown in Fig. 15 (note that OR element symbols shown in the figure are the DeMorgan equivalent symbol for NAND gates).

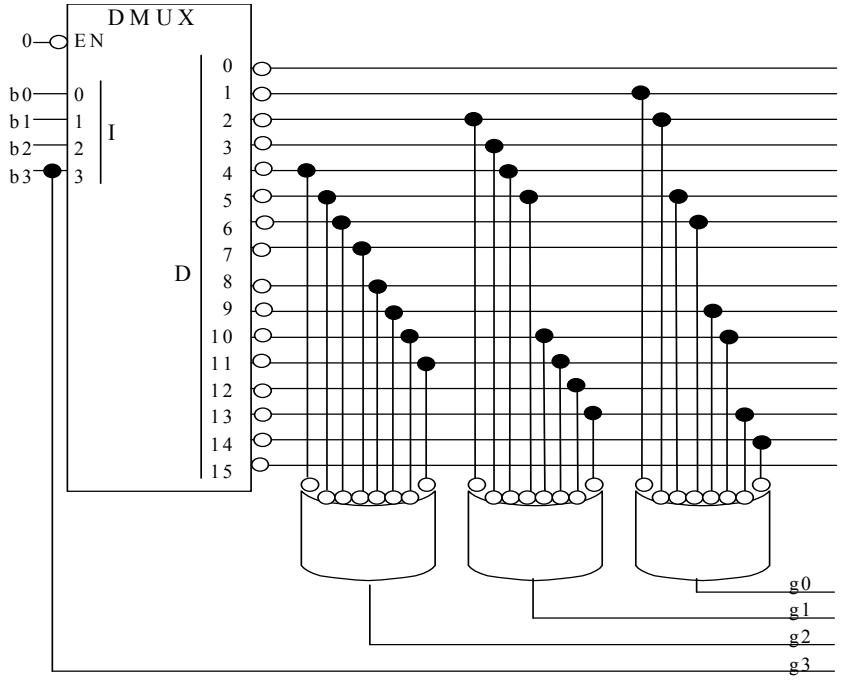

Fig. 15. Design a 4-bit binary-to-gray code converter using 1-to-16 Demultiplexer and three NAND elements.

Notice in the design in example above that there are the same number of 1s in each gray code function as there are 0s. When designing with Demultiplexers, it is important to count the number of 1s and 0s in function. If there are fewer 1s, the function is best implemented using the miniterm compact form for the 1s of the function; however, if there are fewer 0s, the function is best implemented using miniterm compact form for the 0s. A circuit implemented with fewer minterms requires gates with less input lines and takes less wiring.

3.2. Example

Obtain the functional logic diagram for the following functions using one 1-to-16 Demultiplexer without output inverters.

$F_1 = \Sigma(0,1,5,8,13)_{abcd}$

$F_2 = \Pi(3,7,8,12,14,15)_{abcd}$

$F_3 = \Sigma(1,3,4,5,7,8,11,12,13,14,15)_{abcd}$

$F_4 = \Pi\ (0,2,3,6,7,8,10,12,13,15)_{abcd}$

Solution

Because functions F_2 and F_4 are presented in maxterm compact form, we must convert them to miniterm form.

$F_2 = \Sigma\ (0,1,2,4,5,6,9,10,11,13)_{abcd}$

$F_4 = \Sigma\ (1,4,5,9,11,14)_{abcd}$

In the second step we count 1s and 0s in each function: functions F_2 and F_3 have less 0s than 1s, so for this functions are better implemented using miniterm compact form for the 0s (see Fig. 16).

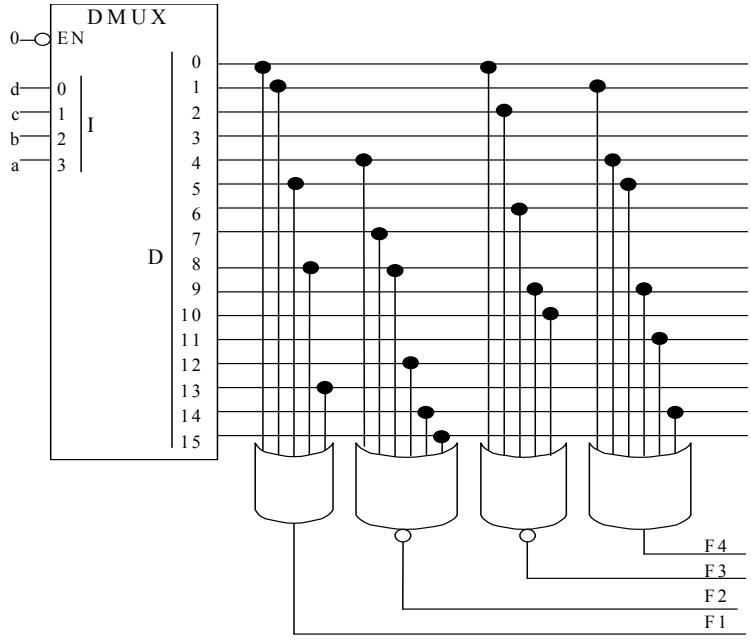

Fig. 16. Design functions F_1, F_2, F_3, and F_4 using 1-to-16 Demultiplexer.

4. Implementing Logic Functions Using Demultiplexers and Multiplexers

A type 2 (and higher) MUX design requires two or more signal to be partitioned off. Using this signals are created set of functions which are connected to input lines of Multiplexer. Example below presents this problem and its solution.

4.1. Example

Obtain the functional logic diagram for the following function using one 1-to-4 Demultiplexer and 8-to-1 Multiplexer.

$F= \Sigma(2,3,4,6,9,10,12,16,17,18,19,21,23,25,26,30,31)_{abcde}$

Solution

Karnaugh table for function F is shown in Fig. 17 (signals d and e are partitioned off in the Karnaugh map) and implementation for this function in Fig. 18.

		de				
abc		00	01	11	10	F
000	0	0	0	1	1	d
001	1	1	0	0	1	~e[1]
011	3	1	0	0	0	~d~e
010	2	0	1	0	1	d⊕e
110	6	0	1	0	1	d⊕e
111	7	0	0	1	1	d
101	5	0	1	1	0	e
100	4	1	1	1	1	1

Fig. 17. Karnaugh table for example 3

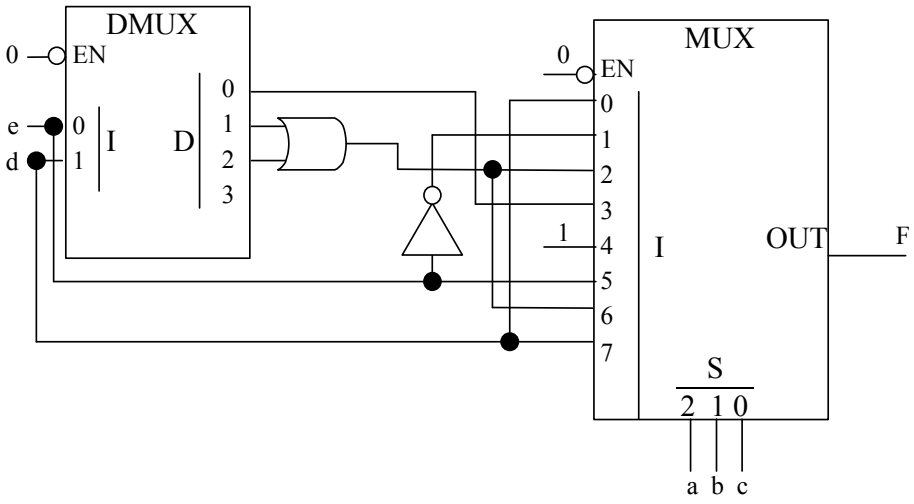

Fig. 18. Implementation for function F

[1] Where ~ sign stands for complementary operator.

5. Tasks to be performed during laboratory

1. Obtain function F_1, F_2, F_3, F_4 using one 1-to-16 Demultiplexer (with invertors on output lines) and NAND elements.
   $F_1 = \Pi\ (1,5,6,9,12)_{abcd}$
   $F_2 = \Sigma\ (2,6,9,10,11,12)_{abcd}$
   $F_3 = \Sigma(1,2,3,5,6,8,9,10,12,14,15)_{abcd}$
   $F_4 = \Pi\ (1,2,5,6,7,9,10,11,14,15)_{abcd}$

2. Obtain function Z=ab(c+(~c)d)+~d((~a)(~c)+bc)+(~b)c(~d) using:
   a. one 16-to-1 line Multiplexer (using a type 0 design)
   b. one 8-to-1 line Multiplexer and inverter element (using a type 1 design)
   c. one 4-to-1 line Multiplexer and NAND or Inverter elements (using a type 2 design)
   d. one 2-to-1 line Multiplexer and NAND or Inverter elements (using a type 3 design)
   e. one 4-to-1 line Multiplexer and 1-to-4 line Demultiplexer and NAND or Inverter elements (using a type 2 design with Demultiplexer)

6. Instructions to follow

1. Solve all tasks before the exercise.
2. Implement the circuits specified by your supervisor (using given elements).
3. Present working circuits to your supervisor for acceptance.