Introduction to Electrical Engineering

Course Code: EE 103

Department: Electrical Engineering

Instructor Name: B. G. Fernandes

E-mail id: bgf@ee.iitb.ac.in

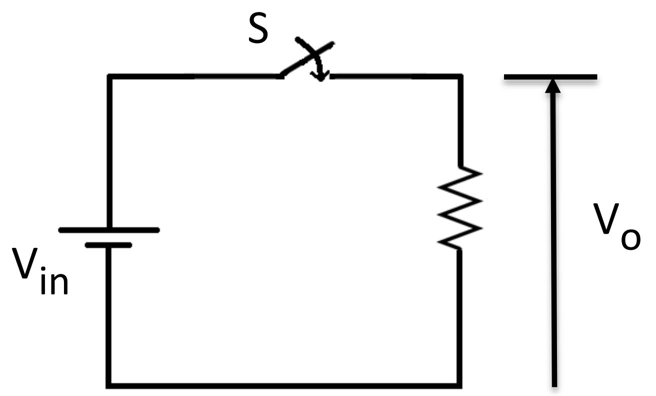
Digital Electronics

- Two ways of representing the numerical values of any quantity
- **Analog** → most of the physical quantities
  → vary over a continuous range of values
- **Digital** → Discrete → digital clock
  → reliability is more
  → easier to design
- **Output is 0 or 1**

'S' closed $V_{in} \approx V_o$

'S' open $V_o = 0$

We need only the ``range'' and not the ``exact'' value of V or I



Number System	**Base**	**Characters**
Binary	2	0 and 1
Octal	8	0 – 7
Decimal	10	0 – 9
Hexadecimal	16	0 – 9, A – F

➢ Boolean Algebra and Identities

- AND, OR and NOT are the basic functions

OR	AND	NOT
$A + 0 = A$	$A0 = 0$	$A + \bar{A} = 1$
$A + 1 = 1$	$A1 = A$	$A\bar{A} = 0$
$A + A = A$	$AA = A$	$\bar{\bar{A}} = A$
$A + \bar{A} = 1$	$A\bar{A} = 0$	

Binary addition			
a	**b**	**sum**	**carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

➢ Realization of Logic Function using Logic Gates

Treat OR as Union of sets and AND as Intersection of sets.
0 as empty set and 1 as universal set.

- AB + AC can be realized using 2 AND & 1 OR gate → 3 gates
- Alternatively, AB + AC → A(B + C) ➔ 1 OR and 1 AND gate → 2 gates
- Reduce following logic expression using Boolean Algebra

$$(A + \bar{B} + \bar{C})(A + \bar{B} + C) = A\,[1 + \bar{B} + C + \bar{B} + \bar{C}] + \bar{B}[1 + C + \bar{C}]$$

$$= A + \bar{B}$$

- Many a times, it is not convenient to use Boolean Algebra

Karnaugh Map (K – Map)

- Means of showing a relationship between logic input and desired output
- Squares are labelled so that horizontally adjacent squares differ only in one variable. Similarly, vertically adjacent squares differ only in one variable.

A	B	$Output$
0	0	1
0	1	1
1	0	0
1	1	0

→

	$\bar{B}$	B
$\bar{A}$	1 (00)	1 (01)
A	0 (10)	0 (11)

→

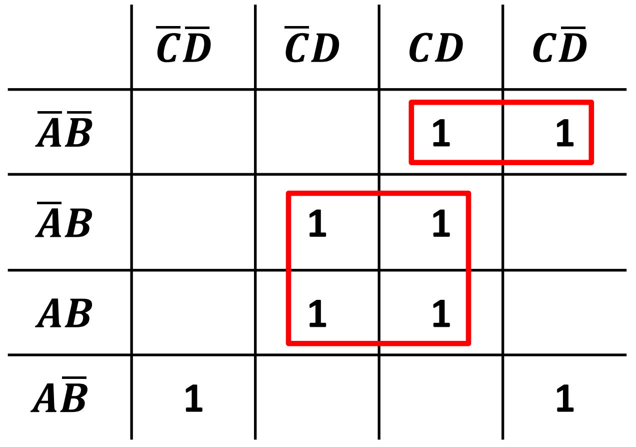
$$\bar{A}\bar{B} + \bar{A}B$$
$$= \bar{A}(\bar{B} + B)$$
$$= \bar{A}$$

- Different from K-map: as B changes and $\bar{A}$ does not change → Drop the variable which has changed

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}$	1	0	0	1
C	0	1	0	0

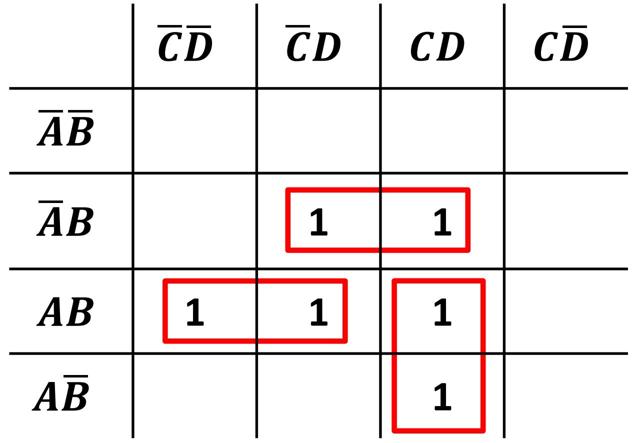
→

$$X = \bar{B}\bar{C} + \bar{A}BC$$



	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$			1	1
$\bar{A}B$		1	1	
AB		1	1	
$A\bar{B}$	1			1

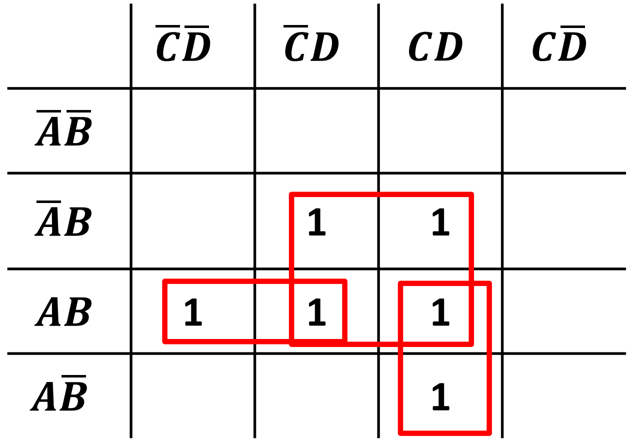
$$X = \bar{A}\bar{B}C + BD + A\bar{B}\bar{D}$$



	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$				
$\bar{A}B$		1	1	
AB	1	1	1	
$A\bar{B}$			1	

$$X = AB\bar{C} + \bar{A}BD + ACD$$

Instead



	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$				
$\bar{A}B$		1	1	
AB	1	1	1	
$A\bar{B}$			1	

$$X = BD + AB\bar{C} + ACD$$

Always choose the notation in which least number of variables are used.

Suppose in 3rd row all are 1s $\rightarrow$ $X = AB$

2nd row and 3rd row are all 1s $\rightarrow$ $X = B$

Suppose 2nd and 3rd rows are all 1s and 1st row 2nd column is also 1 $\rightarrow$ $X = B + \bar{A}\bar{B}\bar{C}D$

Instead loop the isolated 1 with 2nd row 2nd column $\rightarrow$ $X = B + \bar{A}\bar{C}D$

Minterms

Variable (A) = 1 and Complement of variable (A bar) = 0

Minterms are a way to easily represent Boolean (0 or 1) functions using Decimal notation.

- It is convenient to express a Boolean function in its sum-of-minterms
- $F = \bar{A}\,\bar{B}C + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$
- It is sometimes convenient to express the function in the following brief notation:

$$\mathrm{F}(A, B, C) = \sum(1, 4, 5, 6, 7)$$

where the numerals correspond to the minterms (with ‘1’ for true and ‘0’ for complement)

- $\sum$ stands for OR-ing of the terms

Truth Table

A	B	C	Decimal equivalent	F
0	0	0	0	0
0	0	1	1	1
0	1	0	2	0
0	1	1	3	0
1	0	0	4	1
1	0	1	5	1
1	1	0	6	1
1	1	1	7	1

Boolean Function Notation/Representations

Consider the function:

$$F(A, B, C) = \sum(1, 4, 5, 6, 7)$$

A	B	C	Decimal equivalent	F
0	0	0	0	0
0	0	1	1	1
0	1	0	2	0
0	1	1	3	0
1	0	0	4	1
1	0	1	5	1
1	1	0	6	1
1	1	1	7	1

F	$\bar{B}\bar{C}$	$\bar{B}C$	BC	$B\bar{C}$
$\bar{A}$	(000) 0 0	(001) 1 1	(011) 3 0	(010) 2 0
A	(100) 4 1	(101) 5 1	(111) 7 1	(110) 6 1

$$F = A + \bar{B}C$$

K-map Minimization

$$F(A,B,C,D) = \sum(1,3,4,11,12,13,14,15)$$

Always type the variables in a K map in such a way that only one variable changes in the next line.

AB \ CD	00	01	11	10
00	0	1	1	0
01	1	0	0	0
11	1	1	1	1
10	0	0	1	0

And not 10

Steps to be followed

- Construct the K map and place 1s in that square corresponding to the 1s in truth table. Place 0s in other squares.
- Loop those 1s which are not adjacent to any other 1s. These are isolated 1s. Next, look for those 1s which are adjacent to only one other 1. Loop any pair containing such a 1.
- Loop any octet even if some of 1s have already been looped.
- Loop any quad that contain one or more 1s which have not yet been looped.
- Loop any pairs necessary to include any 1s that have not yet been looped. Make sure to use the minimum number of loops.
- Form the OR sum of all terms.

Half Adder

Computers perform addition operation on 2 binary numbers at a time, each binary number can have several binary digits.

Least Significant Bits - least weightage (rightmost ones)

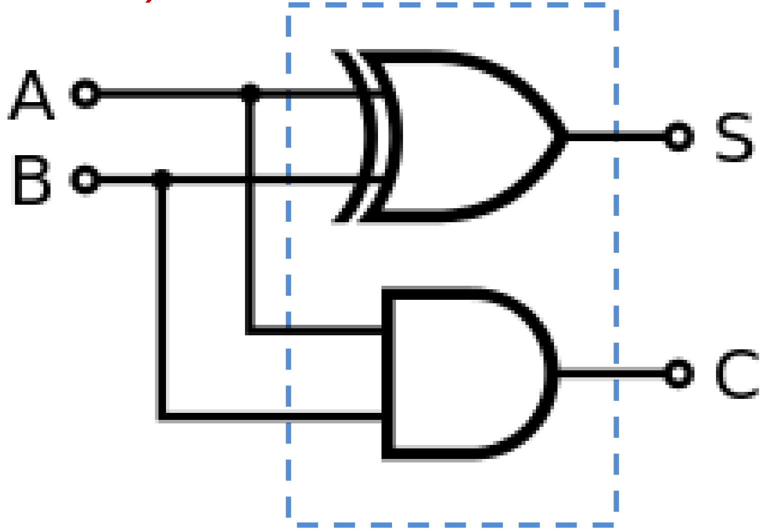
- Process starts by adding LSBs.
- Adds 2 bits (A & B), generate 'sum' and carry.
- Half adder

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$$S = A\bar{B} + \bar{A}B$$

$$C = AB$$

Sum S = | A-B |



Half adder

- While adding the next bit, carry generated (C_i) in the previous stage has to be added.

Full Adder

Sum = A + B + C_i

A	B	C_i	Sum	C_o
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Ci = Previous Carry

Sum S = | A - B - Ci |

For Carry to be 1, at least two of (A, B, previous carry) have to be 1.

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}_i$	0 (000)	$\boxed{1}$ (001)	0 (011)	$\boxed{1}$ (010)
C_i	$\boxed{1}$ (100)	0 (101)	$\boxed{1}$ (111)	0 (110)

	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}_i$			1	
C_i		1	1	1

$$Sum = \bar{A}B\bar{C}_i + A\bar{B}\bar{C}_i + ABC_i + \bar{A}\bar{B}C_i$$

$$= \bar{C}_i(\bar{A}B + A\bar{B}) + C_i\,(AB + \bar{A}\bar{B})$$

$$= \bar{C}_i Z + C_i\bar{Z}$$

Where, Z = o/p of half adder with A & B as inputs.

Sum → o/p of another half adder with C_i & Z as inputs.

C_i → Carry of previous stage

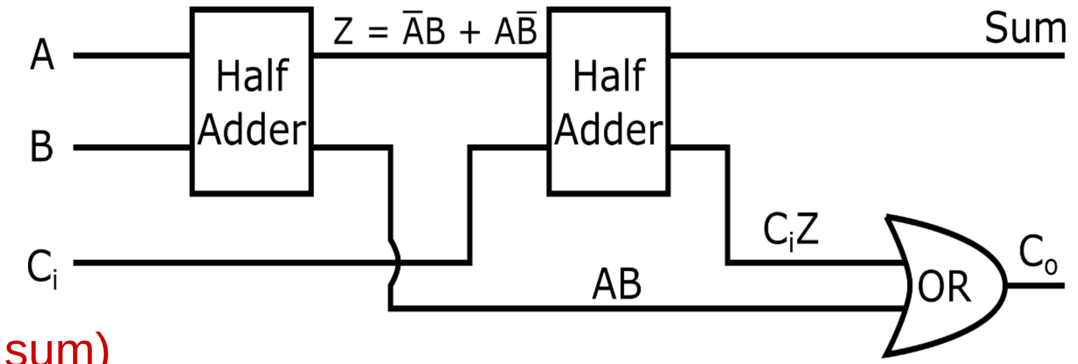
= AB

$C_o = C_i Z + AB$

$$C_o = AB + C_iB + C_iA$$

Instead

$$C_o = AB + C_i\,\underbrace{(\bar{A}B + A\bar{B})}_{Z}$$



New carry = AB + (old carry)(old sum)

Don't care condition

Some logic circuits can be designed so that there are certain input conditions for which there are no specified output levels.

- Because their input conditions will never occur
- Certain combinations of input, where we 'don't care' whether the o/p is high or low

- Example: Logic circuit that control an elevator door of a 3 story building
- 4 Inputs
  - M ⇒ 0 when stopped
    ⇒ 1 when moving
  - F_1, F_2, F_3 ⇒ Generally low
    ⇒ Go high only when the elevator is positioned at a level of that particular floor.

∴ only one of them can be high

M	F_1	F_2	F_3	Door (open)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	X
0	1	0	0	1
0	1	0	1	X
0	1	1	0	X
0	1	1	1	X
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	X
1	1	0	0	0
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

$MF_1 \backslash F_2F_3$	00	01	11	10
00	0	1	X	1
01	1	X	X	X
11	0	X	X	X
10	0	0	X	0

What’s best for minimizing the output expression?

$MF_1 \backslash F_2F_3$	00	01	11	10
00	0	1	1	1
01	1	1	1	1
11	0	0	0	0
10	0	0	0	0

Open=$\bar{M}F_1 + \bar{M}F_3 + \bar{M}F_2 = \bar{M}(F_1 + F_2 + F_3)$

Don't care Example: EXAMPLE 3.8

Simplify the Boolean function

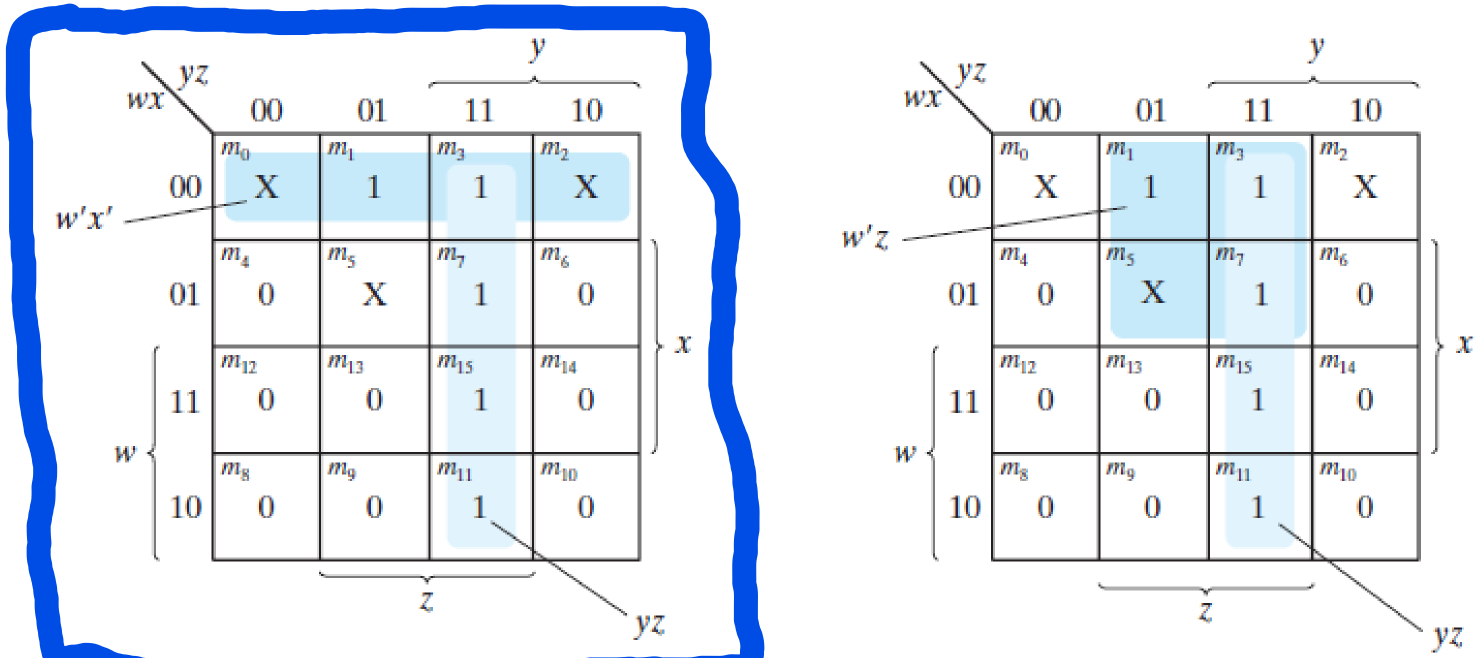
$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

which has the don't-care conditions

$$d(w, x, y, z) = \Sigma(0, 2, 5)$$

By taking suitable values for X, make the simplest expressions.

Learn the decimal notation for boxes.



(b) $F = yz + w'z$

FIGURE 3.15
Example with don't-care conditions

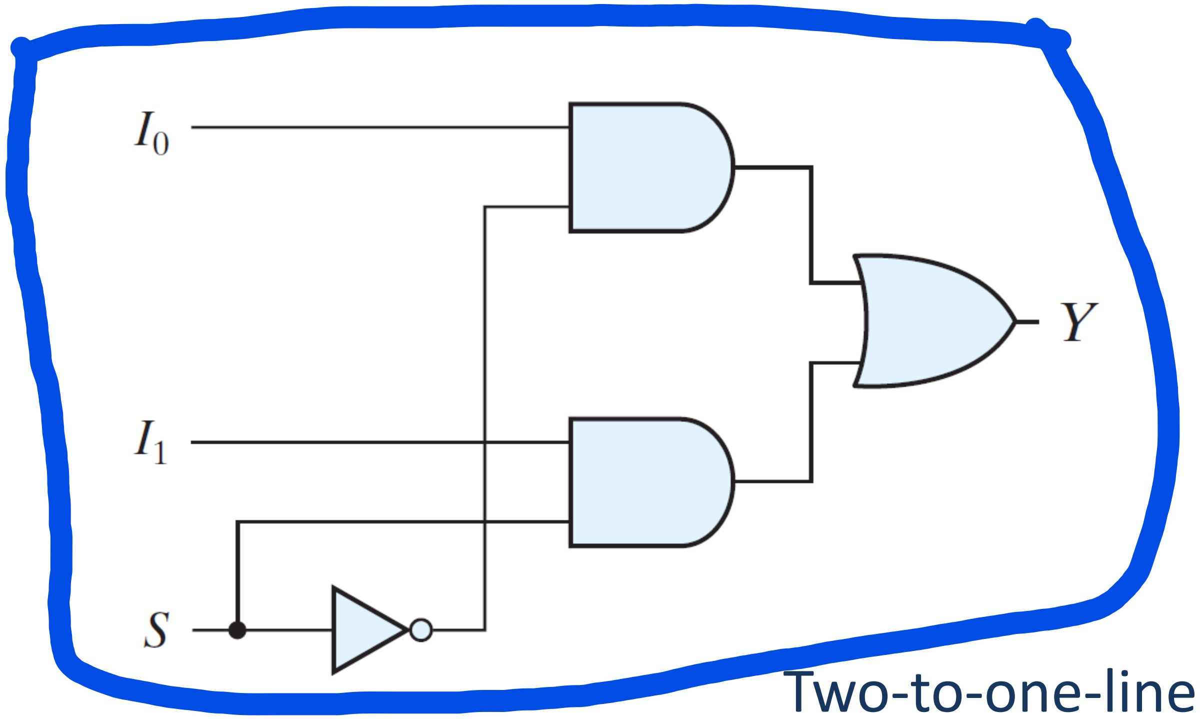
Advantage of using Don't cares:

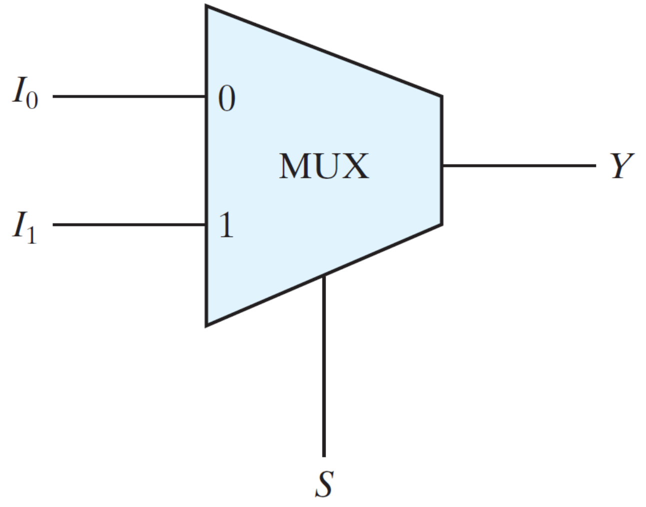
simpler expressions

Source: Morris Mano & M. Ciletti: Digital Design, 5 Ed., Pearson

Multiplexer (MUX)

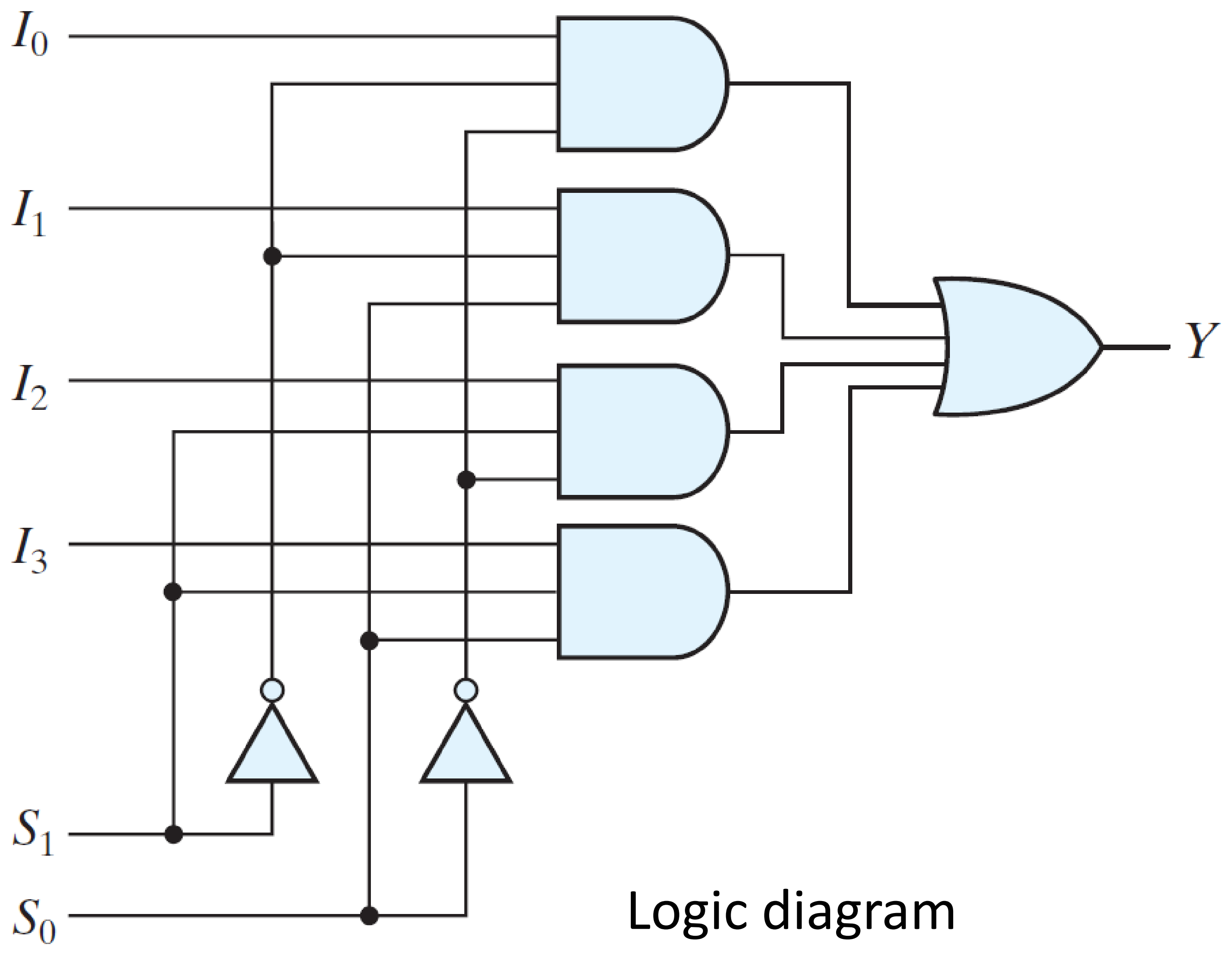
- Logic circuit accepts several data inputs. But allows only one of them at a time to get through the o/p.
- Routing of the desired input to the o/p is done by 'select' inputs or 'address' inputs.





Two-to-one-line multiplexer

Multiplexer: Four-to-one-line multiplexer

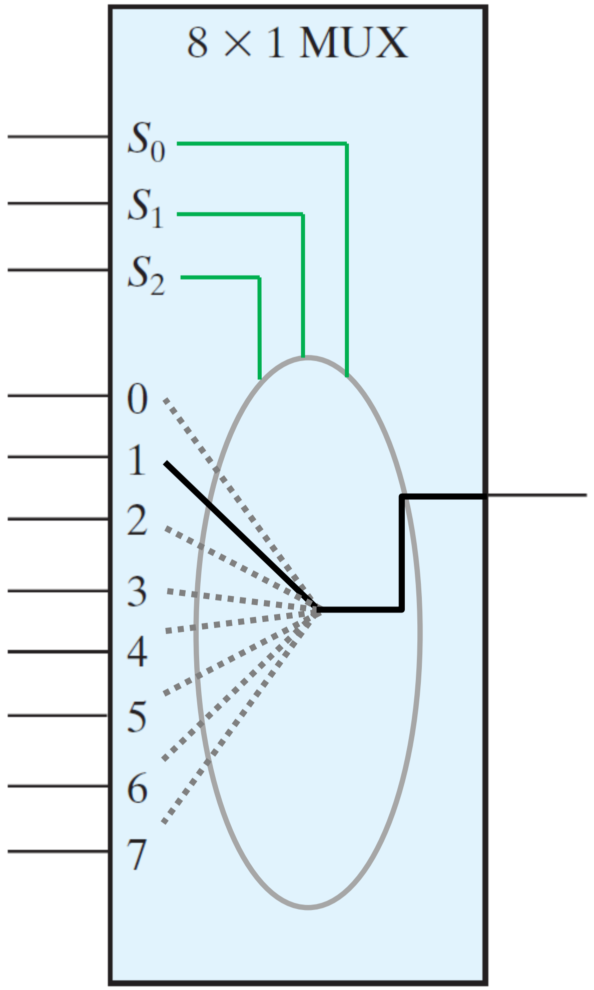


Logic diagram

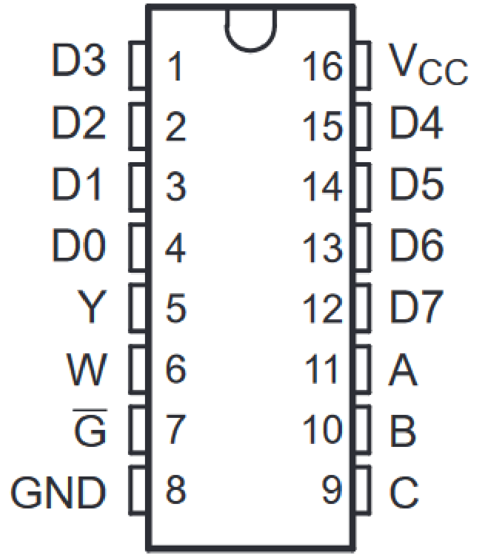
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

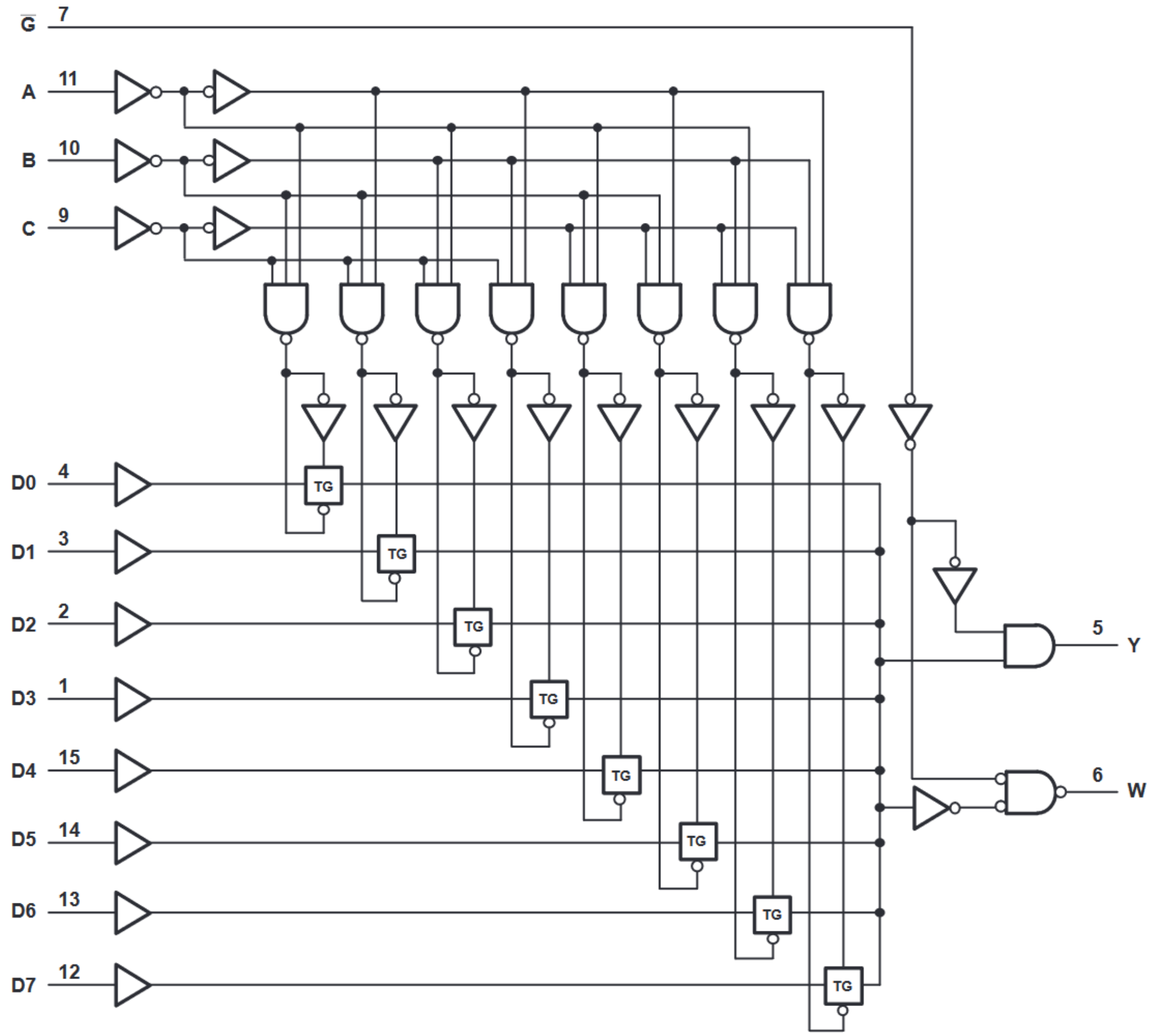
Function table

8-1 Multiplexer

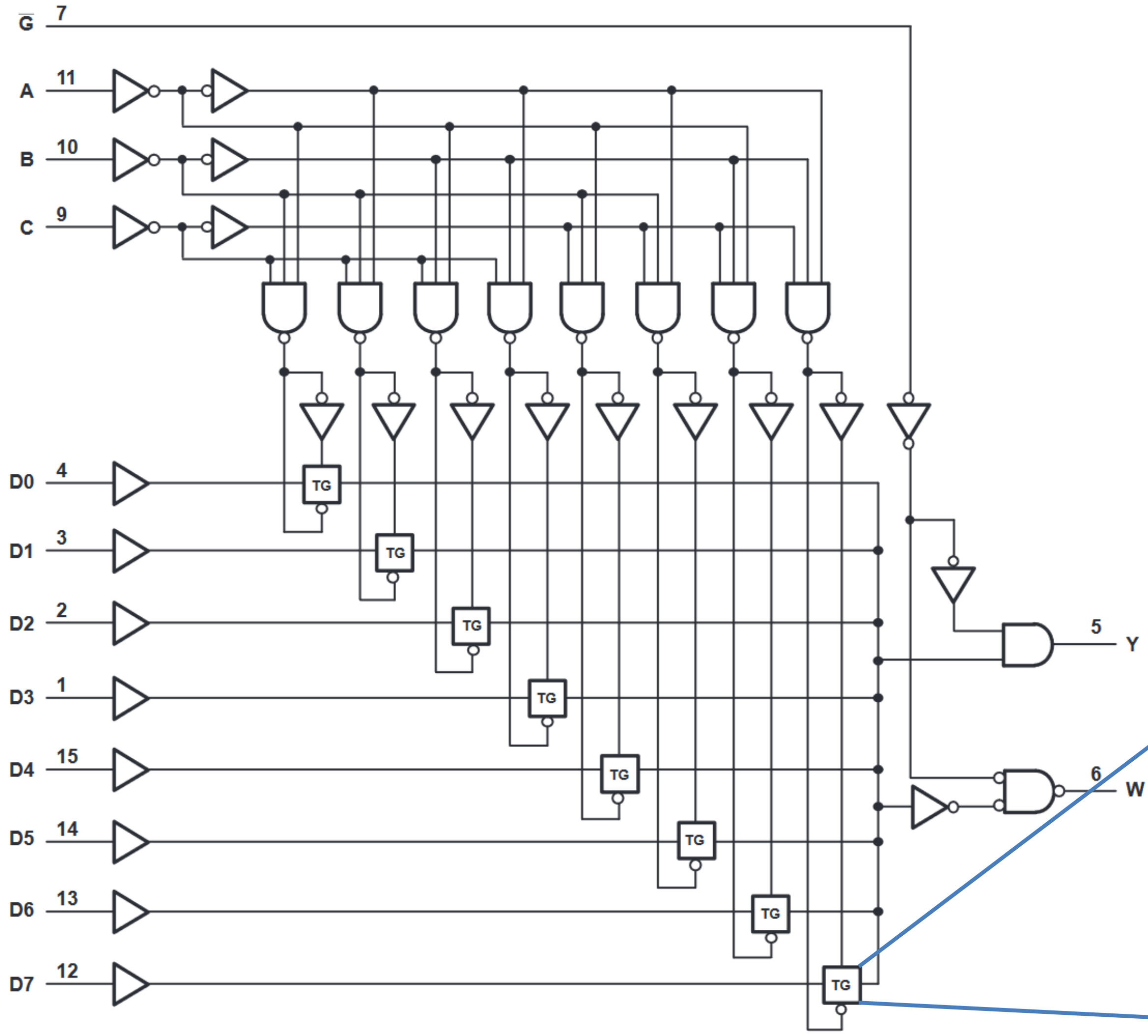


8-1 Mux: IC 74151

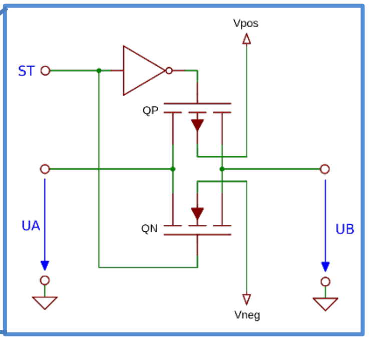




8-1 Multiplexer vs Discrete Gates for Logic Realization: Footprint comparison

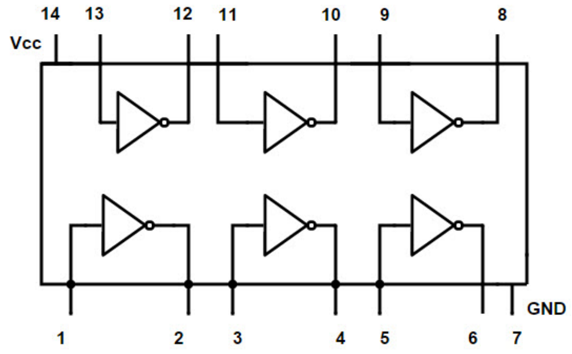


8-1 Mux: IC 74151

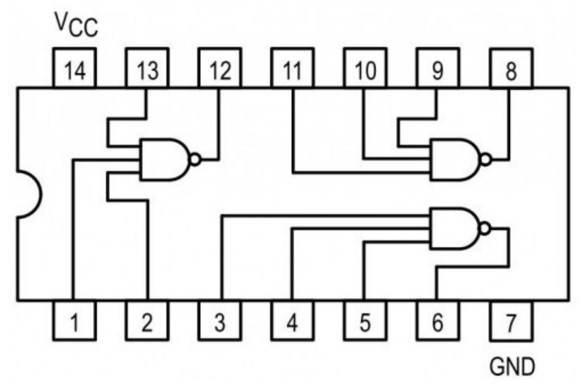


TG: Transmission Gate

NOT Gate: IC 7404



3 input NAND Gate: IC 7410



Implementation of Boolean expressions using Multiplexers (MUX)

- In a practical circuit, the number of ICs should be minimized; K-map solution requires a combination of gates, with differing number of inputs - which in most cases will not result in the minimum number of ICs.

Another Solution:

- A 4-to-1 MUX can directly implement the Truth Table of a TWO variable function using its TWO selection inputs and Input lines.
- Similarly, an 8-to-1 MUX can directly implement the Truth Table of a THREE variable function using its THREE selection inputs;
- A 16-to-1 MUX can implement the Truth table of a FOUR variable function using its FOUR selection inputs.
- It is more efficient to implement a Boolean function of *n* variables using a MUX that has *(n-1)* selection inputs.