CS225 Switching Theory

S. Tripathy
IIT Patna

Previous Class

Minimization/ Simplification **of Switching Functions**

K-map (SOP)

Quine-McCluskey (Tabular) Minimization

This Class

Quine-McCluskey (Tabular) Minimization

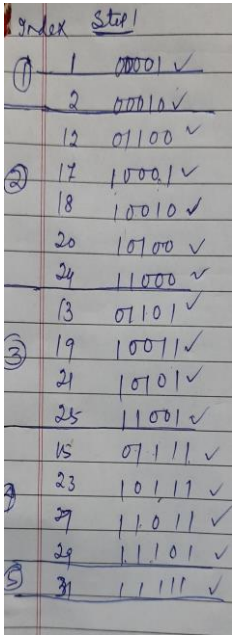
Combinational Circuit logic design

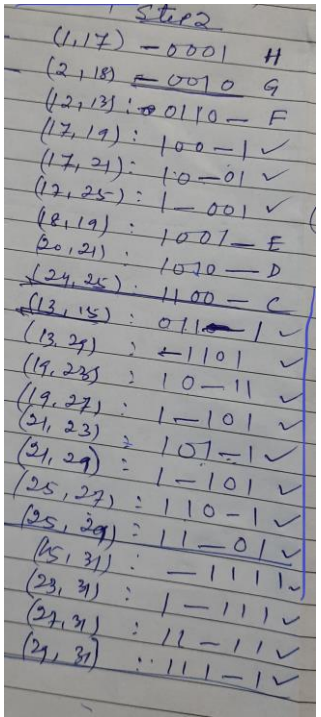
Quine-McCluskey Tabulation Procedure
Step 1: to Obtain the Set of All Prime implicants

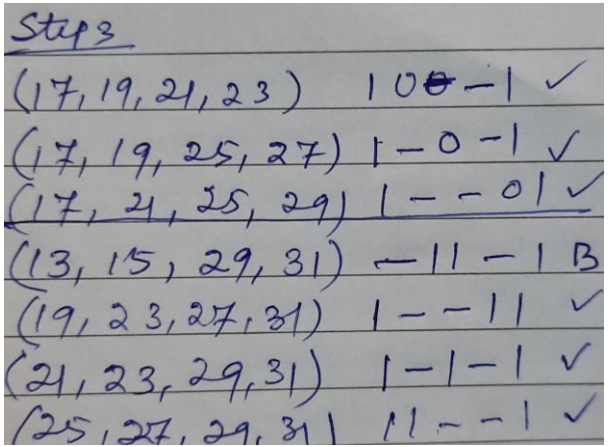
Example: with don't care condition

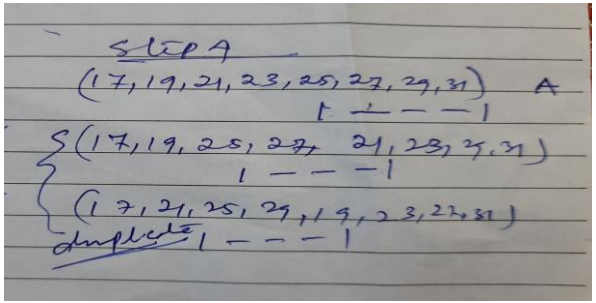
Ex.: $f_3(v,w,x,y,z) = \sum(13,15,17,18,19,20,21,23,25,27,29,31) + \sum_{\phi}(1,2,12,24)$

(a) (b) (c) (d)







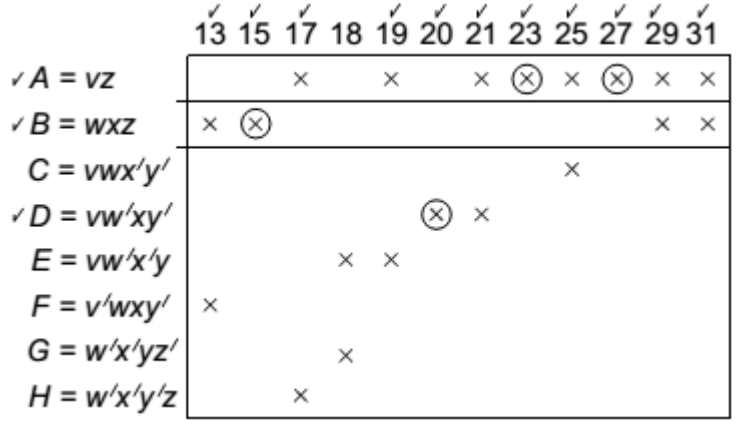


P= {vz, wxz, vwx'y', vw'xy', vw'x'y, v'wxy', w'x'yz', w'x'y'z'}

Step 2: Find the minimal expression(s)

Don't-cares: not listed as column headings in the prime implicant chart

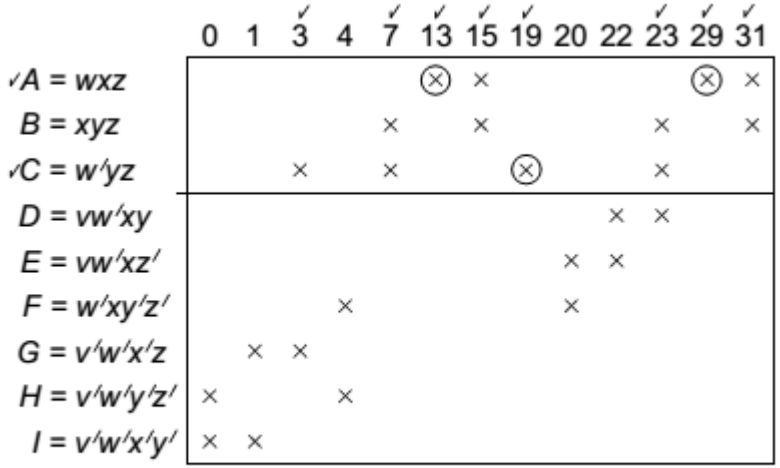
Example: $f_3(v,w,x,y,z) = \sum(13,15,17,18,19,20,21,23,25,27,29,31) + \sum_{\phi}(1,2,12,24)$



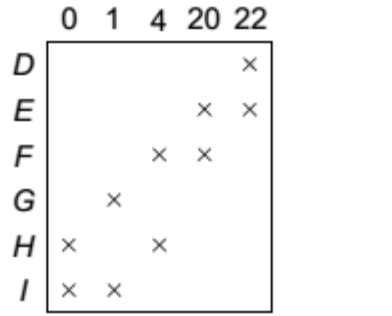
Selection of nonessential prime implicants facilitated by listing prime implicants in decreasing order of the number of minterms they cover

Determining the Set of All Irredundant Expressions

Deriving the minimal sum-of-products through prime implicant chart inspection: difficult for more complex cases *Example:* $f_4(v,w,x,y,z) = \sum(0,1,3,4,7,13,15,19,20,22,23,29,31)$



(*a*) Prime implicant chart.



(*b*) Reduced prime implicant chart.

While every irredundant expression must contain A and C, none of them may contain B since it covers minterms already covered by A and C. The reduced chart, obtained after removing A, B, and C, has two x's in each column

Example (Contd.)

Use propositional calculus: define prime implicant function p to be 1 if each

column is covered by at least one of the chosen prime implicants, and 0 if not

$$p = (H + I)(G + I)(F + H)(E + F)(D + E)$$
$$= EHI + EFI + DFI + EGH + DFGH$$

Since all prime implicants in the reduced chart have the same literal count, there are four minimal sum-of-products:

$$f_4(v,w,x,y,z) = A + C + E + H + I = wxz + w'yz + vw'xz' + v'w'y'z' + v'w'x'y'$$
$$f_4(v,w,x,y,z) = A + C + E + F + I = wxz + w'yz + vw'xz' + w'xy'z' + v'w'x'y'$$
$$f_4(v,w,x,y,z) = A + C + D + F + I = wxz + w'yz + vw'xy + w'xy'z' + v'w'x'y'$$
$$f_4(v,w,x,y,z) = A + C + E + G + H = wxz + w'yz + vw'xz' + v'w'x'z + v'w'y'z'$$

Reduction of the Chart

Aim: find just one minimal expression rather than all such expressions

Example: $f_5(v,w,x,y,z) = \sum(1,3,4,5,6,7,10,11,12,13,14,15,18,19,20,21,22,23,25,26,27)$

	1	3	4	5	6	7	10	11	12	13	14	15	18	19	20	21	22	23	25	26	27
✓ $A = w'x$			×	×	×	×									⊗	⊗	×	×			
✓ $B = v'x$			×	×	×	×			⊗	⊗	×	×									
$C = vx'y$													×	×						×	×
$D = vw'y$													×	×			×	×			
$E = wx'y$							×	×												×	×
$F = v'wy$							×	×			×	×									
$G = x'yz$		×						×						×							×
$H = w'yz$		×				×								×				×			
$I = v'yz$		×				×		×				×									
✓ $J = v'w'z$	⊗	×		×		×															
✓ $K = vwx'z$																			⊗		×

(*a*) Prime implicant chart.

Example (Contd.)

	10	11	18	19	26
C			×	×	×
D			×	×	
E	×	×			×
F	×	×			
G		×		×	
H				×	
I		×			

Reduced prime implicant chart

	10 ✓	11 ✓	18 ✓	19 ✓	26 ✓
✓C			⊗	×	×
✓E	⊗	×			×
G		×		×	

Final chart

Dominated row: row U of the chart dominates row V if U covers every column covered by V. If U does not have more literals than V then V can be deleted from the chart.

Example: I is dominated by G, D is dominated by C and F is dominated by E,H is dominated by G, so they can be deleted

From the final chart: $f_5(v,w,x,y,z) = A + B + J + K + C + E$

Dominating Column (Alternative choice)

	10	11	18	19	26
C			×	×	×
D			×	×	
E	×	×			×
F	×	×			
G		×		×	
H				×	
I		×			

Reduced prime implicant chart

Dominating column: column i of the chart dominates column j if i has an x in every row in which j has an x. Hence, dominating column i can be deleted.

Example: column 11 dominates column 10. In order to cover column 10, either E or F must be selected, whereby column 11 will also automatically be covered. Similarly, since column 19 covers column 18, column 19 can be deleted.

Final solution is still: $f_5(v,w,x,y,z) = A + B + J + K + C + E$

Design with Basic Logic Gates

Logic gates: perform logical operations on input signals

Positive (negative) logic polarity: constant 1 (0) denotes a high voltage and constant 0 a low (high) voltage

Synchronous circuits: driven by a clock that produces a train of equally spaced pulses

Asynchronous circuits: are almost free-running and do not depend on a clock; controlled by initiation and completion signals

Fanout: number of gate inputs driven by the output of a single gate

Fanin: bound on the number of inputs a gate can have

Propagation delay: time to propagate a signal through a gate

Logic Design with Integrated Circuits

Small scale integration (SSI): integrated circuit packages containing a few gates; e.g., AND, OR, NOT, NAND, NOR, XOR

Medium scale integration (MSI): packages containing up to about 100 gates; e.g., code converters, adders

Large scale integration (LSI): packages containing thousands of gates; arithmetic unit

Very large scale integration (VLSI): packages with millions of gates

Combinational Logic Design Process

Step		Description
Step 1: Capture behavior	Capture the function	Create a truth table or equations, ***whichever is most natural for the given problem***, to describe the desired behavior of each output of the combinational logic.
Step 2: Convert to circuit	2A: Create equations	This substep is only necessary if you captured the function using a truth table instead of equations. Create an equation for each output by ORing all the minterms for that output. Simplify the equations if desired.
	2B: Implement as a gate-based circuit	For each output, create a circuit corresponding to the output's equation. (Sharing gates among multiple outputs is OK optionally.)

Analysis of Combinational Circuits

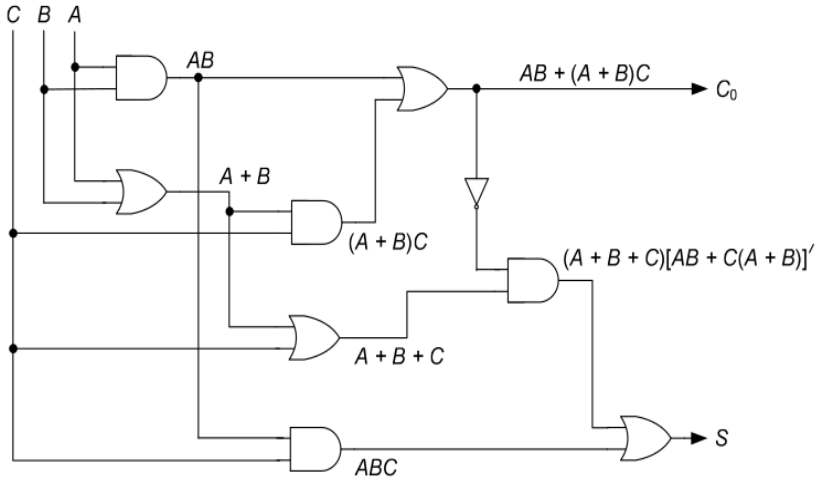
Circuit analysis: determine the Boolean function that describes the circuit

- Done by tracing the output of each gate, starting from circuit inputs and continuing towards each circuit output

Example: a multi-level realization of a full binary adder

$$C_0 = AB + (A + B)C$$
$$= AB + AC + BC$$

$$S = (A + B + C)[AB + (A + B)C]' + ABC$$
$$= (A + B + C)(A' + B')(A' + C')(B' + C') + ABC$$
$$= AB'C' + A'BC' + A'B'C + ABC$$
$$= A \oplus B \oplus C$$



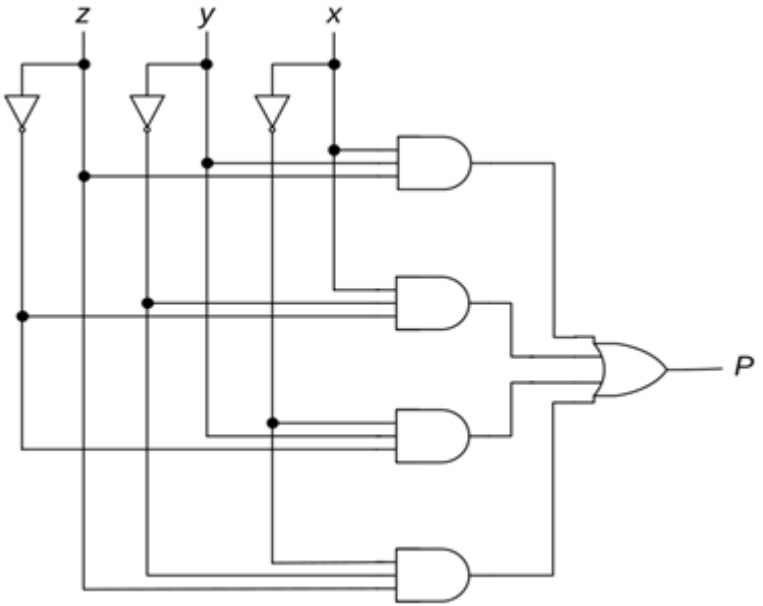
Simple Design Problems

Parallel parity-bit generator: produces output value 1 if and only if an odd number of its inputs have value 1

K-Map

z \ xy	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Implementation

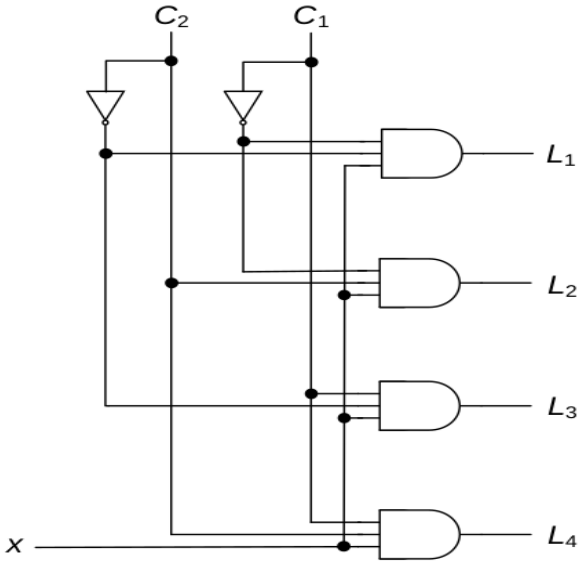


P = X′Y′Z + X′YZ′ + XY′Z′ + XYZ

Simple Design Problems (Contd.)

Serial-to-parallel converter: distributes a sequence of binary digits on a serial input to a set of different outputs, as specified by external control signals

Control		Output lines				Logic equations
C1	C2	L1	L2	L3	L4	
0	0	x	0	0	0	$L_1 = xC1'C2'$
0	1	0	x	0	0	$L_2 = xC1'C2$
1	0	0	0	x	0	$L_3 = xC1C2'$
1	1	0	0	0	x	$L_4 = xC1C2$



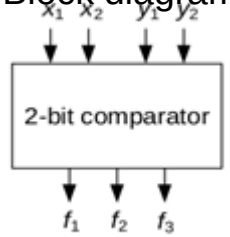
Comparators

n-bit comparator: compares the magnitude of two numbers X and Y, and has three outputs f 1 , f 2 , and f 3

- $f_1 = 1$ iff $X > Y$ $f_2 = 1$ iff $X = Y$ $f_3 = 1$ iff $X < Y$

2-Bit Comparator

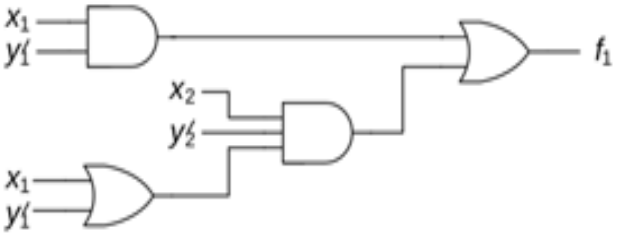
Block diagram



K-Map

y_1y_2 \ x_1x_2	00	01	11	10
00	2	1	1	1
01	3	2	1	1
11	3	3	2	3
10	3	3	1	2

Logic circuit diagram



Logic Expression

$f_1 = X_1X_2Y_2' + X_2Y_1'Y_2' + X_1Y_1'$

$= (X_1 + Y_1')X_2Y_2' + X_1Y_1'$

$f_2 = X_1'X_2'Y_1'Y_2' + X_1'X_2Y_1'Y_2$

$+ X_1X_2'Y_1Y_2' + X_1X_2Y_1Y_2$

$= X_1'Y_1'(X_2'Y_2' + X_2Y_2)$

$+ X_1Y_1(X_2'Y_2' + X_2Y_2)$

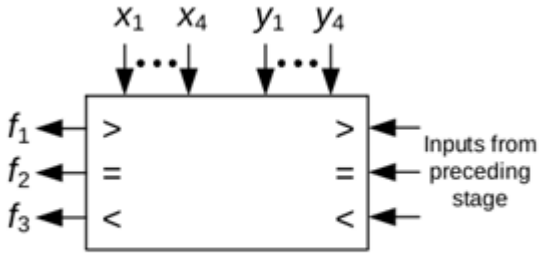
$= (X_1'Y_1' + X_1Y_1)(X_2'Y_2' + X_2Y_2)$

$f_3 = X_2'Y_1Y_2 + X_1'X_2'Y_2 + X_1'Y_1$

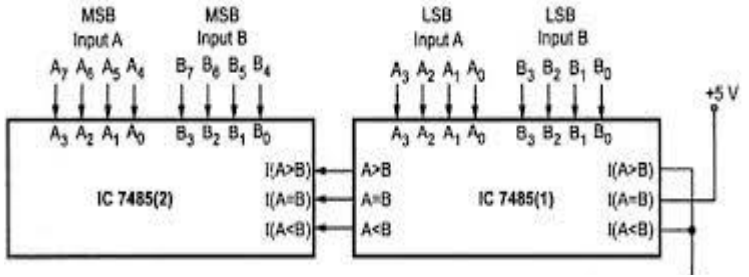
$= X_2'Y_2(Y_1 + X_1') + X_1'Y_1$

4-bit/12-bit Comparators

Four-bit comparator: 11 inputs (four for X, four for Y, and three connected to outputs f 1 , f 2 and f 3 of the preceding stage)

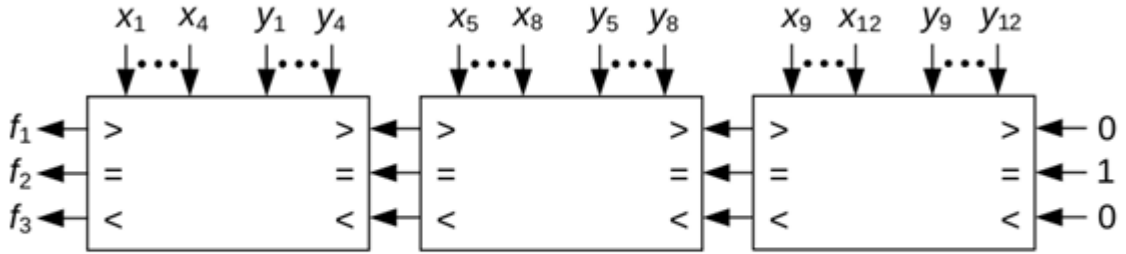


(a) A 4-bit comparator.



12-bit Comparator:

:



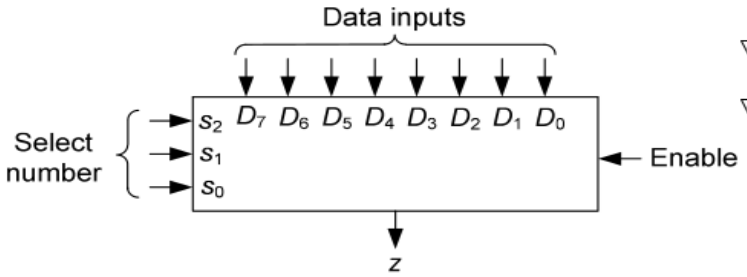
(b) A 12-bit comparator.

Data Selectors

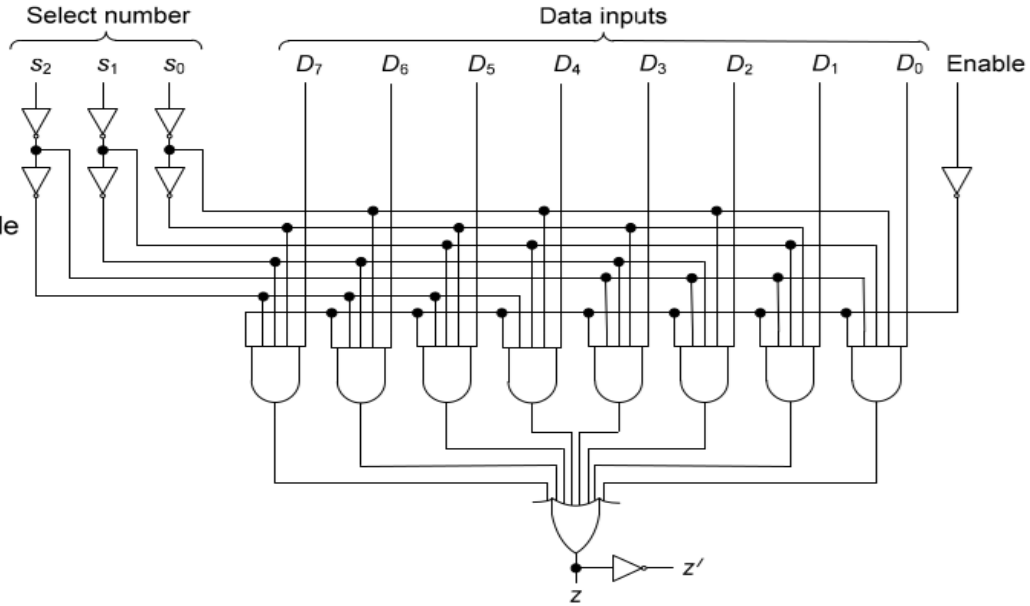
Multiplexer: electronic switch that connects one of n inputs to the output

Data selector: application of multiplexer

- n data input lines, D_0 , D_1 , ..., D_{n-1}
- m select digit inputs S_0 , S_1 , ..., s_{m-1}
- 1 output



(*a*) Block diagram.



(*b*) Logic diagram.

Implementing Switching Functions with Data Selectors

Data selectors: can implement arbitrary switching functions

Example: implementing two-variable functions

$z = sD_1 + s'D_0$

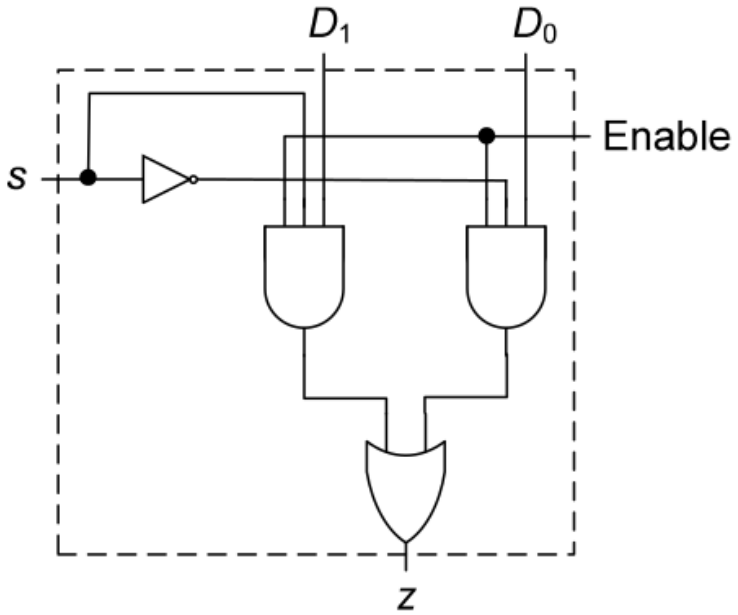
$z = A \oplus B$

If $s = A$, $D_0 = B$, and $D_1 = B'$

$z = A' + B'$.

If $s = A$, $D_0 = 1$, and $D_1 = B'$,

As $AB' + A' = A' + B'$



Thanks