

CS 240 Laboratory 3 Notes

Boolean Algebra and Basic Digital Circuits

□

Laboratory 3 uses truth tables and the laws of Boolean algebra to prove circuit equivalence and to experiment with the operation of the Exclusive OR function.

A boolean function with a given truth table may be implemented in more than one way. In other words, different circuits can be is **equivalent** (have the same truth table), even though the basic gates used in the circuits are not the same.

It is possible to prove circuit equivalency by showing the truth tables for two circuits are the same. It is also possible to prove it by using the identities of Boolean algebra which have been introduced in lecture.

When two functions are equivalent, one function may be converted to the other equivalent function by using the identities of Boolean algebra.

A complicated function (one which used many gates) can often be simplified by using these identities, so that the function may be implemented more efficiently with the simpler equivalent circuit

Also, sometimes it is desirable to implement a circuit with a single type of gate. NAND and NOR gates are defined as **universal** gates, because all of the basic logic functions

(AND, OR, NOT) can be produced by a NAND or NOR gate. Any Boolean function can be defined by a combination of AND, OR, and NOT operations. Therefore, any function can also be defined by only NAND gates, or only NOR gates.

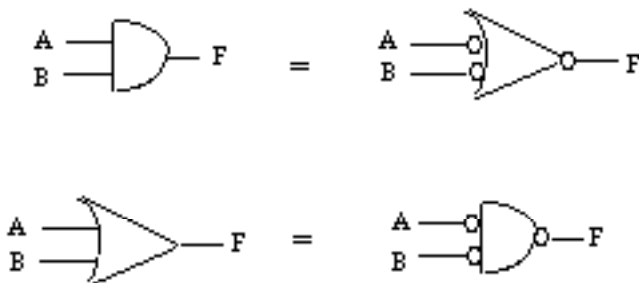
The Boolean identities and **DeMorgan's Law** are used to simplify a function, or to convert it to universal gates:

$$AB = (A' + B)'$$

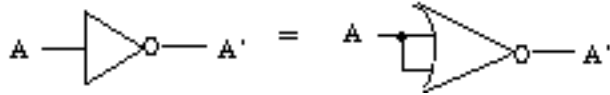
$$A + B = (A'B)'$$

DeMorgan's Law states that any AND gate can be converted to an OR (and vice versa) by changing the AND to an OR, complementing each input to the gate, and complementing the output of the gate:

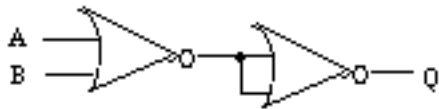
The following diagram is a graphical representation of DeMorgan's Law. The open bubble at the inputs is a shorthand way of showing that a NOT gate is used to produce A' and B'.



When using NOR as a universal gate, you can implement the NOT gate as follows:



and the OR gate by performing a NOT NOR:



$$A + B = Q$$

$$A + B = ((A + B)')' = Q$$

EXAMPLE: Implement the function $Q=(AB)'B'$ using only NOR gates.

To do so, apply DeMorgan's Law to each AND in the expression until all ANDs are converted to ORs:

$$Q = (AB)'B'$$

$$= (A' + B')B'$$

$$= ((A'+B')' + B)'$$

The final expression can be built using only NOR gates.

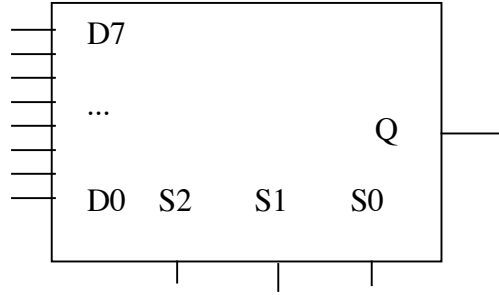
□

MULTIPLEXER

A multiplexer typically has n select lines, 2^n input lines, and 1 output. Only one of the inputs is gated through to the output, based on the value of the n select lines.

In the following 8x1 multiplexer, you are given 3 control lines (S_2, S_1, S_0) and $2^3 = 8$ input lines ($D_0..D_7$). Only one of the 8 inputs $D_0..D_7$ is gated through to the output Q . The output is selected based upon the value of the control lines $S_2 S_1 S_0$:

<u>S2</u>	<u>S1</u>	<u>S0</u>	<u>Q</u>
0	0	0	D0
0	0	1	D1
0	1	0	D2
0	1	1	D3
1	0	0	D4
1	0	1	D5
1	1	0	D6
1	1	1	D7

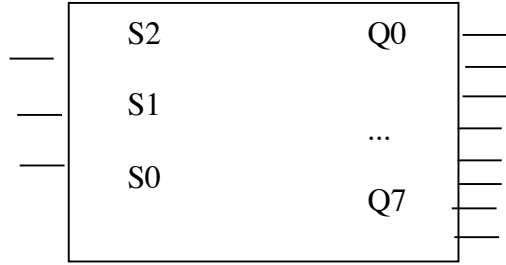


Multiplexers can be used to solve any type of **selection** problem.

Multiplexers can also be used to implement any given function of n inputs. Connect the n inputs to the select lines of the multiplexer, and connect the 2^n possible output values of the function to the inputs of the multiplexer.

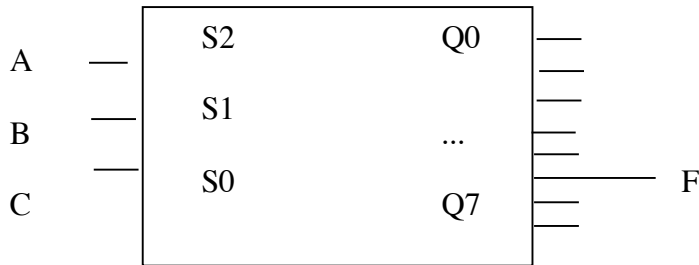
EXAMPLE: $F = ABC' + AB'$

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

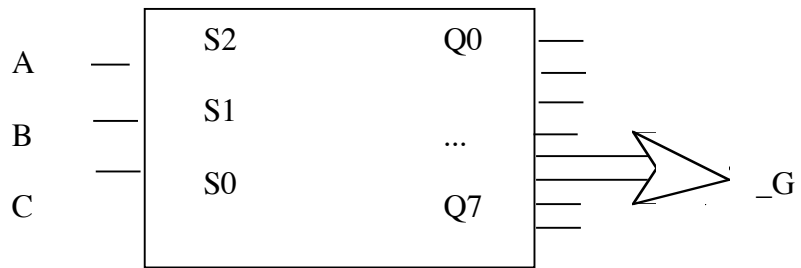


Decoders can be used as code detectors.

EXAMPLE: Using a 3x8 decoder, design a code detector for $F = AB'C$:



How about $G = AB'C + ABC'$?



You can do it with one additional OR gate!