LECTURE 6: DIGITAL LOGIC CIRCUITS

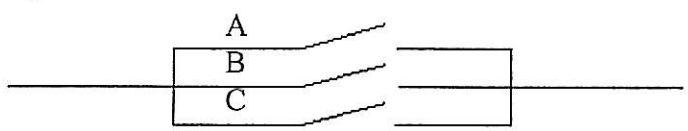
When using digital circuits one runs across strange titles like "OR" circuit, "NOR" circuic, "NAND" circuit, "J-K Flip-Flop", etc. With a modest introduction to the terms, most of this turns out to be relatively simple.

Logic functions can often be realized with simple switches, so we will use such switches to demonstrate some logic functions. Logic functions may depend on binary bits of information. Such a bit can take on only two values or can be in one of two states. Thus a simple switch is either open or closed. The bits could be represented by a yes or a no or perhaps by a transistor that is ON (conducting) or OFF (not conducting). Whatever the two states might be, we will call one of the states a "1" and the other a "0." It doesn't matter which we call "1" and which we call "0" if we are consistent.

Let us call the closed or conducting state of a switch a "1" and the open or non-conducting state a "0." Similarly if a network of switches conducts we will say the output is in the "1" state. If it doesn't conduct, we will say that the output is in the "0" state.

OR Circuit

A parallel system of switches makes an "OR" circuit.



That is, if any one or more of the switches A, B, or C is in state "1", the system is in state "1" (conducting).

It is customary to construct a "truth table." The truth table specifies the state of the system for each state of the switches:

Truth Table for 3-Input (3-Switch) OR Circuit

A	B	C	System
0	0	0	0
1	0	0	1
0	1	0	1
0	0	1	1
1	1	0	1
1	0	1	1
0	1	1	1
1	1	1	1

The extension to two, four, or more switches is obvious.

AND Circuit

A series system of switches makes an "AND" circuit.

That is, all switches must be in the "1" state for the system to be in the "1" state (conducting).

The truth table for the system follows:

Truth Table for 3-Input (3-Switch) AND Circuit

A	B	C	System
0	0	0	0
1	0	0	0
0	1	0	0
0	0	1	0
1	1	0	0
1	0	1	0
0	1	1	0
1	1	1	1

Exercise 1

Construct a truth table for a system of three parallel switches with inverted logic (open or non-conducting is "1" and closed or conducting is "0"). Justify the statement that, if one inverts the meaning of "1" and "0", an "OR" circuit becomes an "AND" circuit and vice-versa.

(Ans. The table above.)

Typically, with the logic circuits used in this course there will be two voltages, V_{DD} and V_{SS} ($V_{DD} > V_{SS}$). With the particular integrated circuits we will use, a one "1" will typically be a voltage near V_{DD}, while a logical "0" will be a voltage near V_{SS}. This is termed positive logic. The exact nature of V_{DD} and V_{SS} is explained in the next section. Two of the most common devices we use will be the "NOR" and the "NAND". These devices are just like "OR" and "AND" except that the outputs are inverted, i.e., in the truth table each output "1" becomes a "0" and each output "0" becomes a "1". The initial "N" stands for NOT or negation.

Two Input NOR Truth Table

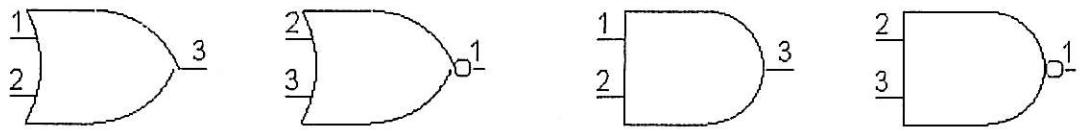
A	B	Output
0	0	1
1	0	0
0	1	0
1	1	0

Two Input NAND Truth Table

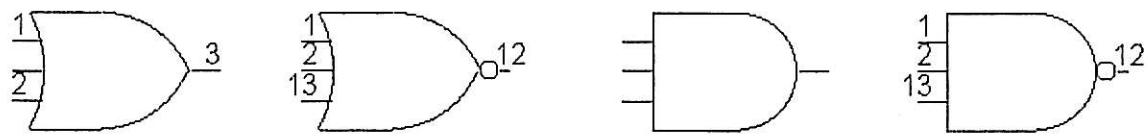
A	B	Output
0	0	1
1	0	1
0	1	1
1	1	0

The following symbols are used for any logic circuit, whether they are made of switches, electronic circuits, fluidic control devices, etc.

2 input 'OR' 2 input 'NOR' 2 input 'AND' 2 input 'NAND'

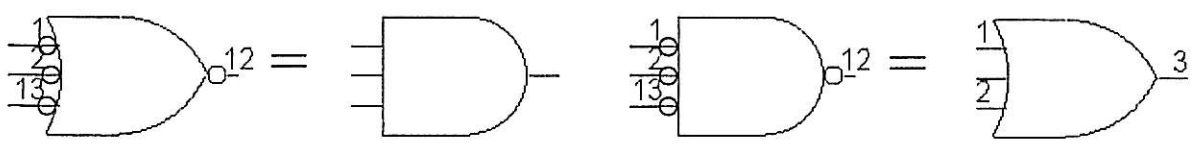


3 input 'OR' 3 input 'NOR' 3 input 'AND' 3 input 'NAND'



Note that a small circle on the output indicates that the output is inverted.

The results of Exercise 1 can be shown schematically as:



Algebraic notation is often used to represent logic operations with the "+" operator indicating "OR", and "•" indicating "AND". A line or bar over a letter indicates the inverse.

$A + B = C$	(A OR B gives C)
$A + B = \overline{C}$	(A OR B gives NOT C)
$A \cdot B = C$	(A AND B gives C)
$C = \overline{A \cdot B}$	(C is NOT (A AND B))

THE ALGEBRA OF LOGIC

Simple Relations

$0 + 0 = 0$	$0 \cdot 1 = 0$	$A + A = A$	$0 + A = A$
$0 \cdot 0 = 0$	$0 + 1 = 1$	$A \cdot A = A$	$0 \cdot A = 0$
$1 + 1 = 1$	$\overline{0} = 1$	$\overline{A} \cdot A = 0$	$1 + A = 1$
$1 \cdot 1 = 1$	$\overline{1} = 0$	$\overline{A} + A = 1$	$1 \cdot A = A$

Commutative Law	Associative Law	Distributive Law
$A + B = B + A$	$(A + B) + C = A + (B + C)$	$(A \cdot B) \cdot C = A \cdot (B \cdot C)$
$A \cdot B = B \cdot A$	$A \cdot (B + C) = A \cdot B + A \cdot C$	

Simplification Rules

$A + A \cdot B = A$ $\qquad$ $A + \overline{A} \cdot B = A + B$ $\qquad$ $A \cdot (A + B) = A$

DeMorgan's Theorem

$\overline{A \cdot B} = \overline{A} + \overline{B}$ $\qquad$ $\overline{A + B} = \overline{A} \cdot \overline{B}$

Unless otherwise indicated, "•" before "+", e.g., $A \cdot (B+C) = A \cdot B + A \cdot C$ means $(A \cdot B) + (A \cdot C)$.

Exercise 2

Prove that $A + \overline{A} = 1$ (construct an appropriate truth table).

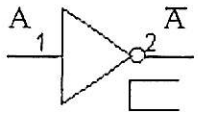
Exercise 3

Prove that $A \cdot (B + C) = (A \cdot B) + (A \cdot C) = A \cdot B + A \cdot C$.

Exercise 4

Prove that $\overline{A + B} = \overline{A} \cdot \overline{B}$.

Another very useful symbol is that of the inverter, having the truth table shown.



A	Out = $\overline{A}$
0	1
1	0

Logic Circuit Families

There are a variety of logic circuit families, identified by their initials: RTL (resistor-transistor logic), DTL (diode-transistor logic), ECL (emitter-coupled logic), TTL (transistor-transistor logic), and CMOS (complementary symmetry metal-oxide-semiconductor). It is easiest to understand how RTL functions internally. ECL is the fastest. DTL has lost popularity. TTL is very commonly used, being both the least expensive and nearly as fast as ECL. Low Power Schottky TTL is very popular in computer applications because it consumes less power than TTL while retaining TTL's high speed. Low Power Schottky devices can be identified by the LS prefix to the device number. CMOS is nearly as popular as TTL. CMOS is inexpensive, has very low power dissipation, has good noise immunity, has high input resistance, and can be run from a very wide range of power supply voltages, typically 5 to 15 volts. Luckily it is rather easy to interface between the two most popular logic families, CMOS and TTL.

TTL logic circuits are so commonly used that everyone must know a bit about them. TTL requires a 5-volt power supply. While changing states the TTL logic circuits briefly draw a large current. To supply this current without generating undue noise and interference, a 0.01 μF capacitor with short leads should be connected from 5 volts to ground near each group of chips (perhaps every 4 chips if they are close together). TTL inputs are designed to accept 0.8 volts or less as low or logical zero and 2.0 volts or more as high or logical one. The region between 0.8 and 2.0 is undefined and to be avoided. TTL outputs are designed to be at 0.4 volts or less for a logical zero and at 2.4 volts or greater for a logical one. The difference between the input and output specifications leaves a "noise margin" of 0.4 volts. One TTL output can only successfully drive 10 standard TTL inputs. The number of inputs an output can drive is called the "fanout" of the IC. TTL has a fanout of 10. The output of a TTL gate need only provide (source) 40 μA to bring the input of the next gate high, but it must accept (sink) 1.6 mA to pull the following input low. This sourcing of current by a TTL input has two important related consequences. If an input is grounded through a resistor to provide a logical zero, the maximum resistance allowed is 500 Ω since 1.6 mA through 500 Ω results in a voltage rise of 0.8 volts, the maximum allowed "zero" voltage. Also a disconnected input will rise to a logical one, though it is not a good practice from a noise sensitivity standpoint to use such "floating" ones.

It should be noted that not all TTL inputs are "standard" TTL inputs. Some inputs are twice as hard to drive and load as two standard TTL inputs. Data sheets should be consulted for any device of interest.

We will use CMOS in the lab. It has a wide choice of logic functions available. Although slightly more expensive than TTL, in most cases the most popular CMOS packages are only 5¢ to 15¢ more than the equivalent TTL package. There are two numbering systems, the 74C00 series, which is a pin for pin, function for function equivalent of the 7400 series TTL family. The other series is the 4000 series. We are using the 4000 series. Data sheets for all the digital IC's you have are in Appendix 1b. Within either series there are two types: the "A" and "B" types. The "B" types have additional circuitry that improves some of the electrical characteristics at the expense of somewhat slower responses. For the circuits you will be building in the lab, you need not know whether you have "A" or "B". For those interested in the details, more can be found in the "CMOS Cookbook" by Don Lancaster (Howard Sams Publication).

In CMOS the power supplies are called V_{DD} and V_{SS}. Typically, V_{SS} is connected to ground and V_{DD} is some value between +5 and +15 volts. In our laboratory work we will use V_{DD} = +5 volts, although +9, +10, +12 and +15 V are also very commonly used values. A CMOS input will recognize as a "0" any potential between -0.5 volts and 0.45 V_{DD}. It will recognize as a "1" a potential between 0.55 V_{DD} and V_{DD} + 0.5 volts. Voltages between 0.45 V_{DD} and 0.55 V_{DD} do not give a well defined response. To be safe it is best to avoid input voltages between 0.3 V_{DD} and 0.7 V_{DD}. It should be noted that, except during transitions between the "1" and "0" states, the input current is almost always less than one nanoampere.

All unused inputs should be tied to either ground or V_{DD}. Failure to do so may result in unwanted power dissipation and an output that changes rapidly and unpredictably. In order that the "0" output lies between 0 and 0.5 volts, and the "1" lies between V_{DD} and V_{DD} - 0.5 volts, the output should not have to source (supply) or sink (accept) more than 100 μamps (However, this specification varies greatly from one device to another. Thus you should check the specification sheet if output current is of concern).

An output can supply voltage to almost a limitless number of inputs to other CMOS gates since they normally draw less than 1 nanoamp. Usually, however, it is recommended that 1 output drive no more than 50 inputs. Driving more than 50 inputs may result in unwanted extra power dissipation.

A few rules are useful when working with CMOS, since CMOS can be sensitive to static electricity shocks.

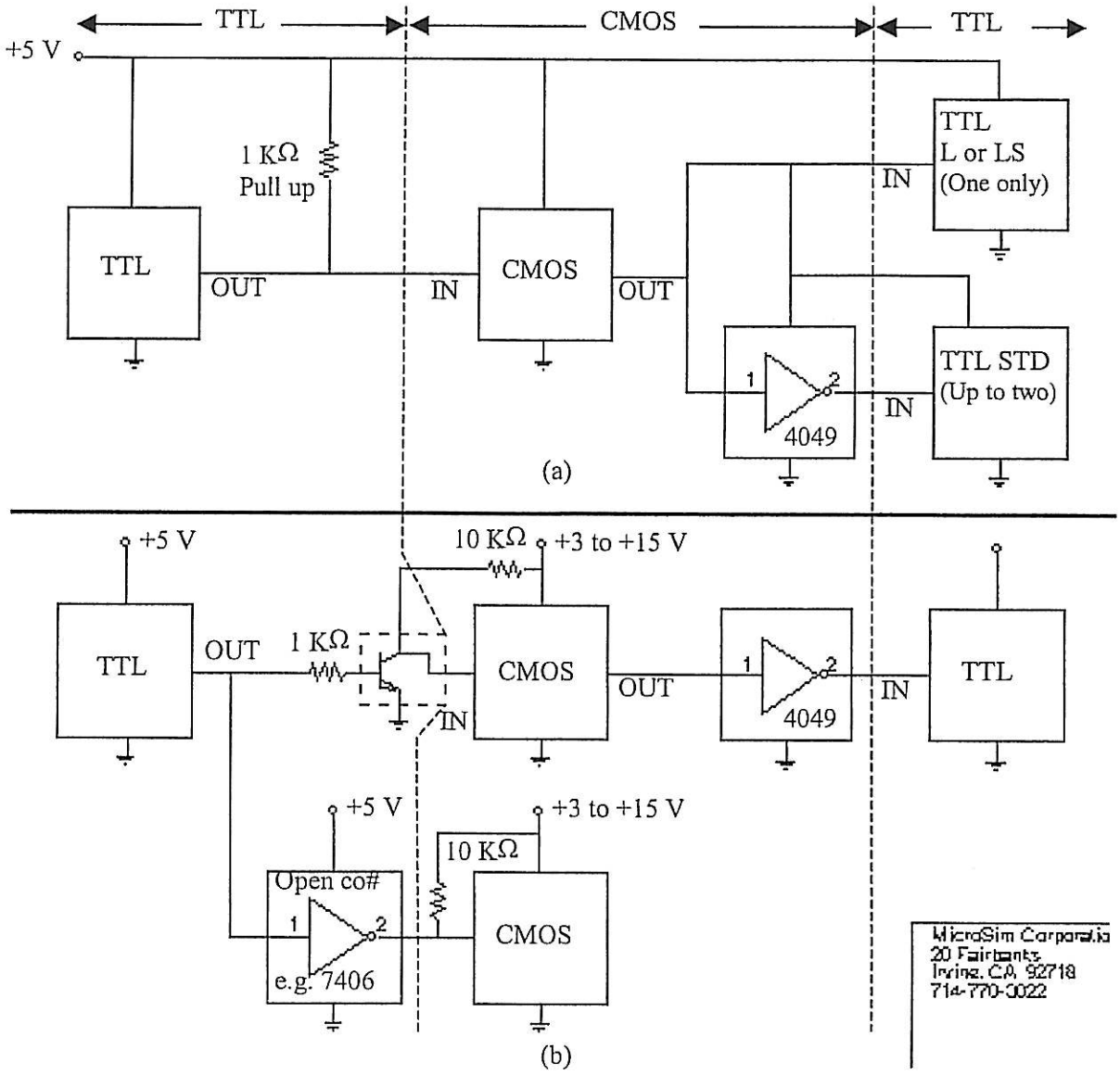
1. Always keep the IC in its protective foam until you are ready to use it.

2. Ground yourself before inserting or removing an IC from its foam. Ground yourself to the circuit before inserting the IC into the circuit.

3. As with any IC, always check to see that the power supply polarities are correct before turning on the power.

4. Input voltages should never be greater than V_{DD} + 0.5 volts or less than V_{SS} - 0.5 volts.

The figure below (adapted from R.J. Higgins' "Electronics with Digital and Analog Integrated Circuits") shows how to connect TTL outputs to CMOS inputs or CMOS outputs to TTL inputs.

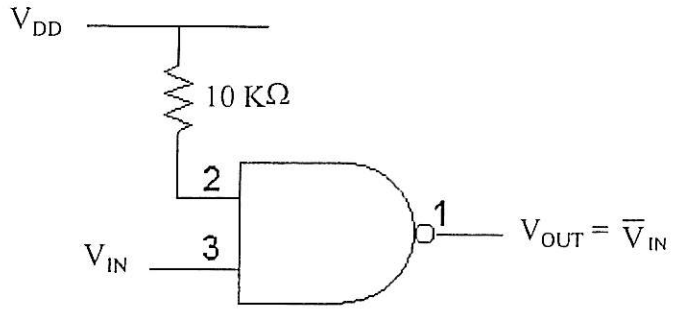


Interfacing between CMOS and TTL logic families. (a) Both CMOS and TTL running on 5 V power supplies; (b) CMOS running on independent power supply.

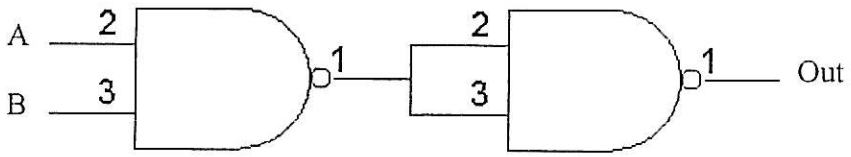
Let us note some simple uses of two-input NAND gates.

1. INVERTER

If one input is connected to V_{DD}, the output is just the inverse of the other input.



2. GATING



Exercise 5

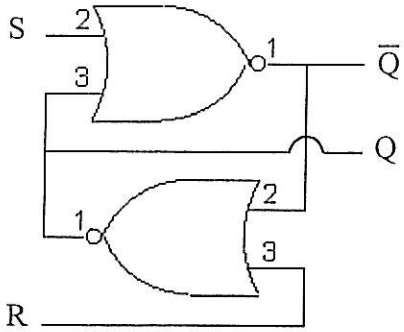
Make a truth table for the above circuit. Why can A be said to "gate" B? (It opens the gate to B or closes it to B.)

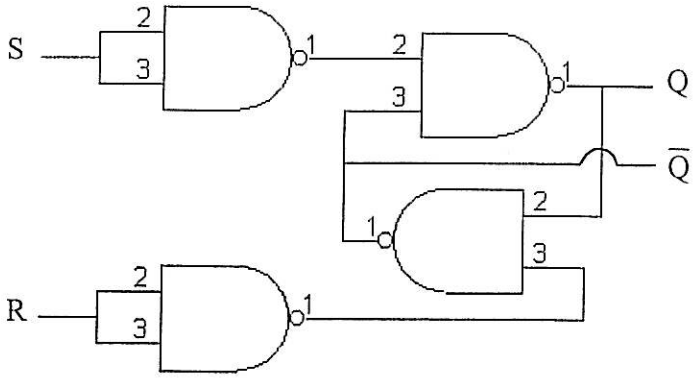
Ans.

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

If A = 1 the output = B.
If A = 0 the output is zero independent of B.

3. R-S FLIP-FLOP





Assume both inputs R and S are at "0". The output can remain stable at either Q = 1, $\overline{Q}$ = 0, or at Q = 0, $\overline{Q}$ = 1. Assume Q = 0 and $\overline{Q}$ = 1. If S (the "SET" input) goes to the 1 state, Q goes to 1 and $\overline{Q}$ to 0. After S returns to 0, the output remains with Q = 1, $\overline{Q}$ = 0. Now if the R input ("RESET" input) goes to 1, the output reverses.

4. EXCLUSIVE OR

This circuitry is to have the truth table:

$in_1 = A$	$in_2 = B$	$out = C$
1	1	0
0	1	1
1	0	1
0	0	0

Exercise 6

Prove that a logical statement of the above truth table is: $C = (A+B)\cdot\overline{(A\cdot B)}$

Exercise 7

Using DeMorgan's Theorem, show that: $C = \overline{(\overline{A}\cdot\overline{B})}\cdot\overline{(A\cdot B)}$

Exercise 8

Show that: $C = (A+B)\cdot(\overline{A}+\overline{B})$

Exercise 9

Show that: $C = (A\cdot\overline{B})+(\overline{A}\cdot B)$

In many design problems we have determined what truth table we need. The question then is what gates do we use to create that truth table? For example, consider this truth table:

A	B	Out
0	0	1
0	1	0
1	0	1
1	1	1

We want to build a circuit that will give us this truth table. As a starting point, let us look at the truth tables for the NAND and NOR:

A NAND B

A	B	Out
0	0	1
0	1	1
1	0	1
1	1	0

A NOR B

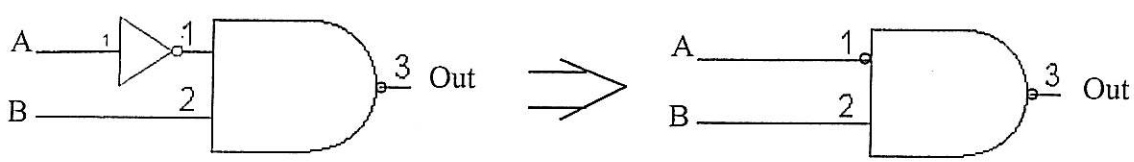
A	B	Out
0	0	1
0	1	0
1	0	0
1	1	0

Noting that the truth table we want has three 1's and one 0, we see that the NAND has the same number of 1's and 0's but in a different order. To try to change that order, let us invert A first, and then perform the NAND operation.

A NAND B

A	$\overline{A}$	B	Out
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1

Thus we have the truth table that we wanted, ignoring the $\overline{A}$ column (which was an intermediate step). Schematically, we would have:



Recall that we rejected the NOR as a starting point because it didn't have the right number of 1's and 0's. But note that, if we invert the output, we would have three 1's and one 0. To get them into the right order we need to invert B (convince yourself of this). Stated algebraically, we have:

$$\text{Out} = A + \overline{B} \text{ (A OR NOT B)}$$

Similarly, the previous solution can be stated as:

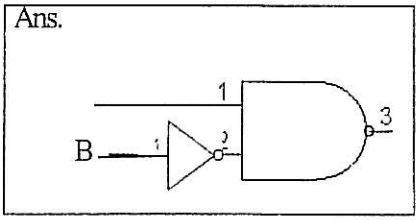
$$\text{Out} = \overline{\overline{A} \cdot B} \text{ (NOT A NAND B)}$$

Exercise 10

Design a circuit to realize:

A	B	Out
0	0	1
0	1	1
1	0	0
1	1	1

Ans.



Karnaugh Maps

WHAT THEY ARE AND HOW TO USE THEM

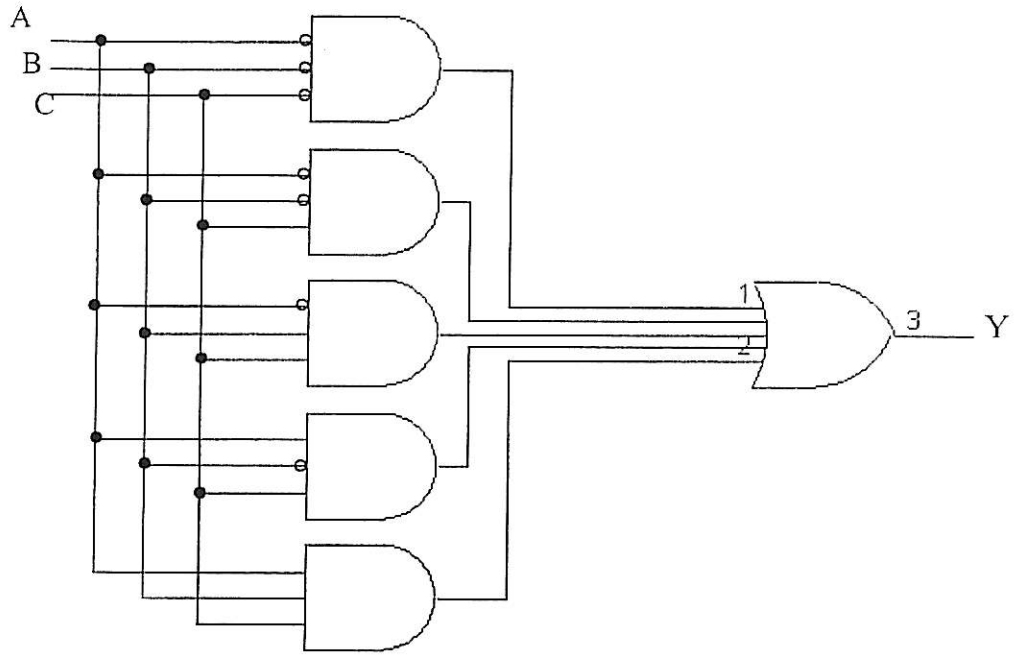
In many situations you want to implement a truth table or logic equations using digital electronic circuitry. It is usually not obvious how one begins the design process. Consider the following 3 variable truth table:

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

The easiest way to implement this (or any) truth table with logic gates is the "brute force" method. In this method, one uses an AND gate to define each Y = 1 state. The inputs of the AND gate are such that it will only turn on for only one particular combination of A, B and C. For each Y = 1 state, a single 3-input AND gate is used. The outputs of all the AND gates are then OR'd together. The output of this "big OR" is the desired variable Y. For instance, this truth table is equivalent to:

$$Y = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,\overline{B}C + \overline{A}BC + A\overline{B}C + ABC$$

This circuit would then be implemented by the following logic gate circuit:



This design, although lacking a certain elegance, will certainly do the job. The brute force method will always work with very little design effort. This method will also usually use the *maximum* number of gates possible to implement the design.

One objective of a good design should be to minimize the number of logic gates used in the circuit, as this improves reliability (i.e., 10 logic gates will fail twice as often as five logic gates) and also make it easier to build the circuit (less wiring means there are fewer chances for making a mistake). One can reduce the above logic equation using Boolean logic relationships to reduce the number of gates. However, this can be tedious mathematical work and one is not guaranteed that the final logic equation will contain the minimum gate implementation.

A more powerful design tool would be useful to reduce the effort needed in designing the circuit. This is the purpose of Karnaugh maps. Karnaugh maps reduce the Boolean algebra manipulations to a purely visual task. Using one's eyes, the designer can see hidden symmetries that one might have missed when manipulating the logical equations. The optimal logic gate implementation can be performed in only a minute or two, even for a four variable logic equation. In summary, the Karnaugh map will tell you immediately the minimum number of gates necessary for the implementation of the logic function or truth table, and it also tells you directly how to implement the circuit.

The Karnaugh map is most useful for logical equations and truth tables which have 3 or 4 logic variables (inputs). It becomes difficult to use the Karnaugh map efficiently for 5 or more variables.

KARNAUGH MAPS: HOW TO MAKE ONE

Example: Implement the previous 3 Variable truth table using a Karnaugh map.

Step #1: Set up the map. Draw a matrix with 4 columns and 2 rows. Label the horizontal axes using two variables (i.e., A and B). Label the vertical axis with one variable (C). At the top of each column, enter all the logic permutations of the first two variables (for example, AB = 00, 01, 11, 10). *Make sure there is only one bit changing between adjacent columns!* At the beginning of the first row print a '0'; print a '1' at the beginning of the second row. You will have the following map or chart:

C\AB	00	01	11	10
0				
1				

Step #2 Using the given truth table, fill in the blanks with the values of Y which correspond to each combination of A, B and C according to the columns and rows you have just labeled. For example, the third row of the truth table will generate an entry in the Karnaugh map in the following manner:

A	B	C	Y
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

C\AB	00	01	11	10
0		0		
1				

The final filled in Karnaugh map would therefor look like:

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

Step # 3 Identify the symmetries of the map by encircling or making a loop around rows, columns, and blocks of entries which have identical logic values. The circles you draw can only include 1, 2, 4, or 8 (a power of two) of the entries. Diagonal loops are not allowed.

You begin by choosing to only encircle either the 1's or the 0's. Let's assume that we choose to encircle only the 1's. You find the biggest loop that can encircle the most 1's. For our example, this would be a row of 4 (a quad loop).

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

Next, since there is no other unique quad loop (a loop which encircles another set of four 1's which are not identical to the previous loop), we create loops of entry pairs to pick up additional 1's which were not previously encircled.

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

The process continues until each of the 1's in the map have at least one loop around them. You may have to make a loop around a single entry if it is not adjacent to another 1.

Here are some examples of the loops you are allowed to make (for some Karnaugh maps other than the above example):

An 8 entry loop:

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

A quad entry loop:

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

A quad block loop:

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

Pair loops:

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

Single loops:

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

Some additional rules:

Loops are allowed to overlap.

Edges of the map 'wrap-around' and touch each other so this quad loop is valid:

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

Sometimes it may be easier to define the map using the zeroes instead of the 1's.

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

Try to use the *minimum* number of loops necessary to encircle all the desired entries. This will automatically minimize the number of gates needed for the logic circuit implementation.

If a single (lone) loop can be extended (overlapped) to form a double or quad entry loop you must extend it.

If a double can be extended (or overlapped) to form a quad you must do it.

Step #4: Each loop represents a logical term in the logic equation of the truth table. The total output Y is the OR of the individual loop terms.

For three variable Karnaugh maps, loops with one entry circled depend upon three variables (e.g. $A\overline{B}C$). Loops with two elements circled depend upon only two variables (e.g. $\overline{A}\overline{B}$). Loops with 4 elements circled depend upon only one variable (e.g. A).

In our original example, we have two loops:

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

$C = 1$ row

↑ $AB=00$ or $\overline{A}\,\overline{B} = 11$ column

The quad loop is independent of A and B (has the same value no matter what A and B are). It depends only upon C, and Y is 1 whenever C is 1. The pair loop is independent of C and therefore can only depend upon A and B. In this loop, the output is 1 when A = 0 and B = 0, thus this is the $\overline{A}\overline{B}$ term. The total logical function Y is the sum of the individual loop terms:

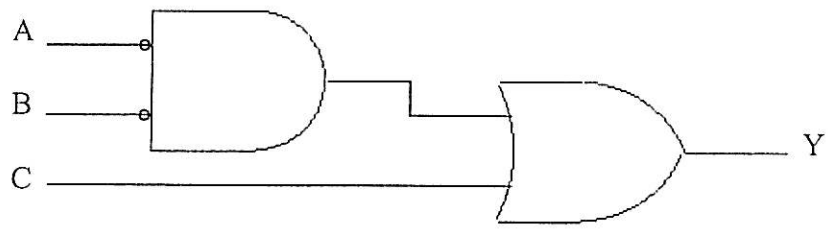
$$Y = C + \overline{A}\,\overline{B}$$

Notice that if we ignored the overlap between the quad and the pair, the logical equation would have been:

$$Y = \overline{A}\,\overline{B}\,\overline{C} + C$$

which is substantially more complicated than the expression we have derived.

Step # 5: Now use the 'brute force' method to implement your circuit, with a large OR gate adding up each of the individual AND terms. The result in this case would be:



4 VARIABLE KARNAUGH MAPS

The 4 variable Karnaugh map proceeds in a similar manner to the three element Karnaugh map with only a few modifications.

Example: Convert the following 4 variable truth table to a logic circuit using Karnaugh mapping:

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
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
1	1	1	1	0

Step # 1: Set up the map. The horizontal axis is the same as in the 3 variable Karnaugh map. The vertical (row) axis will have 2 variables, similar to the horizontal axis. Change only 1 bit as you cycle through the logic permutations on each axis!

CD\AB	00	01	11	10
00				
01				
11				
10				

Step # 2 Fill in the rows and columns of the map from the truth table. A typical entry would be calculated as follows:

A	B	C	D	Y
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
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
1	1	1	1	0

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

The final Karnaugh map would therefore look like:

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

Step # 3: Circle singles, pairs, quads, and octets. Cover all the selected entries with the minimum number of loops:

Single $\bar{A}B\bar{C}\bar{D}$

$A\bar{B}\bar{C}$ pair

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

wrap around pair $\bar{B}\bar{C}D$

Quad $C\bar{D}$

Step # 4: Build the total logical expression using the OR of the individual loop terms as was done previously. In the 4 variable Karnaugh map:

Single element loops need a term with 4 variables (e.g. $\overline{A}B\overline{C}\overline{D}$).

Pair loops represent three variable terms (e.g. $A\overline{B}\overline{C}$).

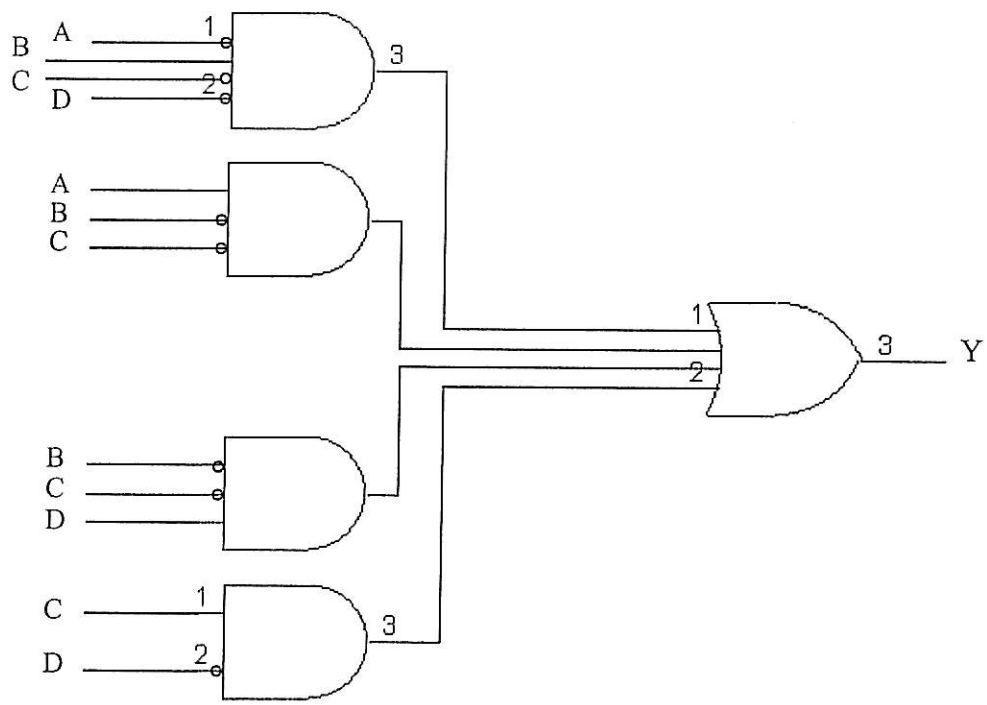
Quad loops represent terms with two variables (e.g. $C\overline{D}$)

Octet loops represent terms with only 1 variable (e.g. A).

The total logical function is the OR of the individual loop terms:

$$Y = \overline{A}B\overline{C}\overline{D} + A\overline{B}\overline{C} + \overline{B}\overline{C}D + C\overline{D}$$

Step # 5: Build the logic circuit using the AND-OR strategy. The circuit would look like the following:



Some additional hints on 4 variable Karnaugh maps:

Again the top and bottom edges 'wrap-around' and touch, as do the left and right sides also wraparound and touch. As a consequence, there are some very strange looking loops:

This is a valid quad loop!:

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

The next two are valid octet loops:

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

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

Remember that you can choose to loop the zeroes instead of the 1's if it will require fewer loops.

You should try interchanging variables on the map axes as you may uncover other simpler hidden symmetries:

Using our above truth table example, you could try:

BC\AD	00	01	11	10
00	0	1	1	1
01	1	0	0	1
11	1	0	0	1
10	1	0	0	0

or

BD\AC	00	01	11	10
00	0	1	1	1
01	1	0	0	1
11	0	0	0	0
10	1	1	1	0