**Combinational circuit**

**Introduction**

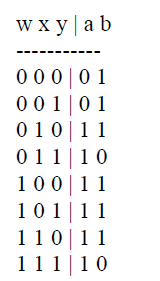
The combinational circuit consist of logic gates whose outputs at any time is determined directly from the present combination of input without any regard to the previous input. A combinational circuit performs a specific information processing operation fully specified logically by a set of Boolean functions. A *combinatorial circuit* is a generalized gate. In general such a circuit has *m* inputs and *n* outputs. Such a circuit can always be constructed as *n* separate combinatorial circuits, each with exactly one output. For that reason, some texts only discuss combinatorial circuits with exactly one output. In reality, however, some important *sharing of intermediate signals* may take place if the entire *n*-output circuit is constructed at once. Such sharing can significantly reduce the number of gates required to build the circuit. When we build a combinatorial circuit from some kind of specification, we always try to make it *as good as possible*. The only problem is that the definition of "as good as possible" may vary greatly. In some applications, we simply want to minimize the number of gates (or the number of transistors, really). In other, we might be interested in as short a *delay* (the time it takes a signal to traverse the circuit) as possible, or in as low *power consumption* as possible. In general, a mixture of such criteria must be applied.

**Describing existing circuits using Truth tables**

To specify the exact way in which a combinatorial circuit works, we might use different methods, such as logical expressions or *truth tables*. A truth table is a complete enumeration of all possible combinations of input values, each one with its associated output value. When used to describe an existing circuit, output values are (of course) either 0 or 1. Suppose for instance that we wish to make a truth table for the following circuit:

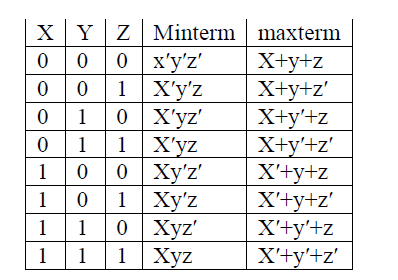


All we need to do to establish a truth table for this circuit is to compute the output value for the circuit for each possible combination of input values. We obtain the following truth table:



**Canonical form(Minterns and Maxterms )**

A binary variable may appear either in it normal form or in it complement form . consider two binary variables x and y combined with AND operation. Since each variable may appears in either form there are four possible combinations: x′y′, x′y, xy′,xy. Each of the term represent one distinct area in the Venn diagram and is called minterm or a standard product. With n variable, 2n minterms can be formed. In a similar fashion, n variables forming an OR term provide 2n possible combinations called maxterms or standard sum. Each maxterm is obtained from an OR term of the n variables, with each variable being primed if the corresponding bit is 1 and un-primed if the corresponding bit is 0. Note that each maxterm is the complement of its corresponding minterm and vice versa.



**Examples:** (X′+Y)(Y+XZ′)′+X(YZ)′ The equation is neither in sum of product nor in product of sum. The implementation is as follow



**Simplicity of logic expressions**

There are many logic expressions (and therefore many circuits) that correspond to a certain truth table, and therefore to a certain function computed. For instance, the following two expressions compute the same function: X(Y+Z) XY+XZ The left one requires two gates, one *and*-gate and one *or*-gate. The second expression requires two *and*-gates and one *or*-gate. It seems obvious that the first one is preferable to the second one. However, this is not always the case. It is not always true that the number of gates is the only way, nor even the best way, to determine simplicity. We have, for instance, assumed that gates are ideal. In reality, the signal takes some time to propagate through a gate. We call this time the gate *delay*. We might be interested in circuits that minimize the total gate delay, in other words, circuits that make the signal traverse the fewest possible gates from input to output. Such circuits are not necessarily the same ones that require the smallest number of gates.

**Circuit minimization**

The complexity of the digital logic gates that implement a Boolean function is directly related to the complexity of the algebraic expression from which the function is implemented. Although the truth table representation of a function is unique, it can appear in many different forms when expressed algebraically.

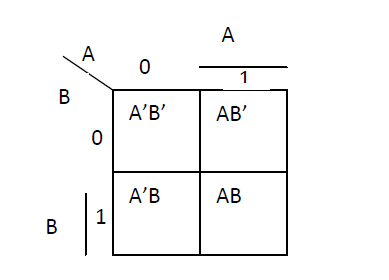
**Simplification through algebraic manipulation**

A Boolean equation can be reduced to a minimal number of literal by algebraic manipulation as stated above. Unfortunately, there are no specific rules to follow that will guarantee the final answer. The only methods is to use the theorem and postulate of Boolean algebra and any other manipulation that becomes familiar e.g. simplify x+x′y x+x′y=(x+x′)(x+y)=x+y simplify x′y′z+x′yz+xy′ x′y′z+x′yz+xy′=x′z(y+y′)+xy′ =x′z+xy′

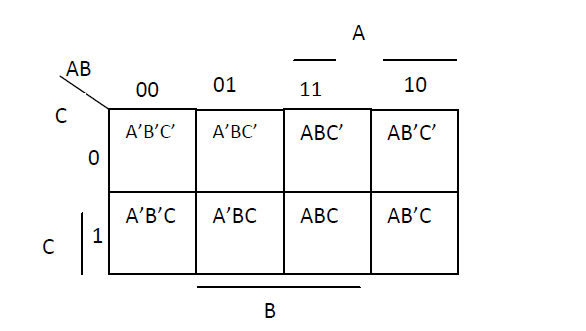
Simplify xy +x′z+yz xy +x′z+yz= xy +x′z+yz(x+x′) xy +x′z+yzx+yzx′ xy(1+z) +x′z(1+y) =xy+x′z

**Karnaugh map**

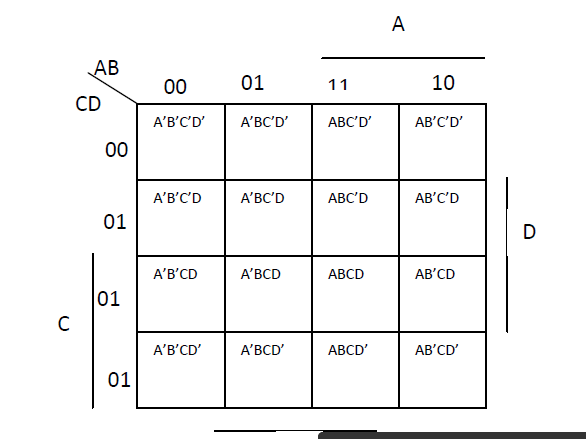
The Karnaugh map also known as Veitch diagram or simply as K map is a two dimensional form of the truth table, drawn in such a way that the simplification of Boolean expression can be immediately be seen from the location of 1’s in the map. The map is a diagram made up of squares , each sqare represent one minterm. Since any Boolean function can be expressed as a sum of minterms, it follows that a Boolean function is recognised graphically in the map from the area enclosed by those squares whose minterms are included in the function. A two variable Boolean function can be represented as follow



A three variable function can be represented as follow



A four variable Boolean function can be represented in the map bellow



To simplify a Boolean function using karnaugh map, the first step is to plot all ones in the function truth table on the map. The next step is to combine adjacent 1’s into a group of one, two, four, eight, sixteen. The group of minterm should be as large as possible. A single group of four minterm yields a simpler expression than two groups of two minterms\. In a four variable karnaugh map, 1 variable product term is obtained if 8 adjacent squares are covered 2 variable product term is obtained if 4 adjacent squares are covered 3 variable product term is obtained if 2 adjacent squares are covered 1 variable product term is obtained if 1 square is covered A square having a 1 may belong to more than one term in the sum of product expression The final stage is reached when each of the group of minterms are ORded together to form the simplified sum of product expression The karnaugh map is not a square or rectangle as it may appear in the diagram. The top edge is adjacent to the bottom edge and the left hand edge adjacent to the right hand edge. Consequent, two squares in karnaugh map are said to be adjacent if they differ by only one variable