Chapter – 3/4/5

Combinational Logic Circuits

Boolean Algebra

- In 1845, George Boole developed Boolean Algebra.
- Boolean Algebra is an algebraic structure defined on a set of element B that takes one of two values, 0 and 1, together with two binary operators + (OR operator) and . (AND operator) provided the following *postulates (axioms)* are satisfied:

1. (a) The structure is closed with respect to the operator +.
   (b) The structure is closed with respect to the operator ·.
2. (a) The element 0 is an identity element with respect to +; that is, $x + 0 = 0 + x = x$.
   (b) The element 1 is an identity element with respect to ·; that is, $x \cdot 1 = 1 \cdot x = x$.
3. (a) The structure is commutative with respect to +; that is, $x + y = y + x$.
   (b) The structure is commutative with respect to ·; that is, $x \cdot y = y \cdot x$.
4. (a) The operator · is distributive over +; that is, $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
   (b) The operator + is distributive over ·; that is, $x + (y \cdot z) = (x + y) \cdot (x + z)$.
5. For every element $x \in B$, there exists an element $x' \in B$ (called the *complement* of x) such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.
6. There exist at least two elements $x, y \in B$ such that $x \neq y$.

Basic Theorems/ Rules of Boolean Algebra

Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Boolean Functions

- A binary variable can take the value of 0 an 1.
- A Boolean Function is an expression formed with binary variables, the two binary operations OR and AND, the unary operator NOT, parentheses and equal sign.
- For a given value of the variable, the function can be 0 and 1.
- Example:
  - F1 = xyz'
  - F2 = x + y'z
  - F3 = x'y'z' + y'yz + xy'

Algebric Manipulation

- A literal is a primed or unprimed variable.
- When a Boolean function is implemented with logic gates, each literal in the function designates an input to a gate, and each term is implemented with a gate.
- The minimization of the number of literals and the number of terms results in a circuit with less hardware, thus reduces the cost and complexity.
- Boolean functions can be minimized by algebraic manipulation.

Simplify the following Boolean functions to a minimum number of literals:

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$
2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$
3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$
4. $$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$, by duality from function 4.

Duality

- The important property of Boolean algebra.
- It states that every algebraic expressions of Boolean algebra remains valid if the operations and identity elements are interchanged.
- Dual of an algebraic expression is obtained by interchanging OR into AND, and AND into OR operations, and replace 1's by 0's and 0's by 1's.

Complement of a Function

- The complement of a function F is F$^{|}$ is obtained from an interchange of 0's and 1's and 1's for 0's in the value of F.
- The complement of a function may be derived algebraically through De-Morgan Theorem.
- Also, the complement of a function is derived by taking dual of the function and complementing each literal.

Find the complement of functions F_1 and F_2.

1. $F_1 = x'yz' + x'y'z.$
   The dual of F_1 is $(x' + y + z')(x' + y' + z).$
   Complement each literal: $(x + y' + z)(x + y + z') = F_1'.$
2. $F_2 = x(y'z' + yz).$
   The dual of F_2 is $x + (y' + z')(y + z).$
   Complement each literal: $x' + (y + z)(y' + z') = F_2'.$

The operator precedence

The operator precedence for evaluating Boolean Expression is: i) Parentheses, ii) NOT, iii) AND, and iv) OR

Canonical AND Standard Forms

- A binary variable may appear either in its normal form (x) or in complement form (x$^{|}$)
- For two binary variables x and y, combined with AND operations, there are four possible operations, x$^{|}$y$^{|}$, x$^{|}$y, xy$^{|}$, and xy. Each this term is called ***minterms*** or a standard product.
- For n variables, we can form 2^n minterms.
- In similar manner, n variables forming OR term, with each variable being primed or unprimed, provide 2^n possible combinations called ***maxterms***, or standard sums.
- Each minterm is complement of its maxterm and vice-versa.
- ***A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms***.

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Example: Function of three variables

Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	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

Here,

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Now consider the complement of a Boolean function. It may be read from the truth table by forming a minterm for each combination that produces a 0 in the function and then ORing those terms. The complement of f_1 is read as

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

If we take the complement of f_1', we obtain the function f_1:

$$\begin{aligned} f_1 &= (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z) \\ &= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6 \end{aligned}$$

Similarly, it is possible to read the expression for f_2 from the table:

$$\begin{aligned} f_2 &= (x + y + z)(x + y + z')(x + y' + z)(x' + y + z) \\ &= M_0 M_1 M_2 M_4 \end{aligned}$$

- Any Boolean function can be expressed as a product of maxterms i.e. form a maxterm for each combination of the variables which produces 0 in the function, and then form the AND of all those maxterms.
- **Boolean function expressed as a sum of minterms or product of maxterms are said to be in *canonical form*.**

Example:

Express the Boolean function $F = A + B'C$ as a sum of minterms. The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

The second term $B'C$ is missing one variable; hence,

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

Combining all terms, we have

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

But $AB'C$ appears twice, and according to theorem 1 ($x + x = x$), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned} F &= A'B'C + AB'C + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

■

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

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

The summation symbol Σ stands for the ORing of terms; the numbers following it are the indices of the minterms of the function. The letters in parentheses following F form a list of the variables in the order taken when the minterm is converted to an AND term.

Example: Express the Boolean Function F = xy + x$^{|}$z in a product of maxterm form.

Express the Boolean function $F = xy + x'z$ as a product of maxterms. First, convert the function into OR terms by using the distributive law:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0M_2M_4M_5 \end{aligned}$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

The product symbol, Π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

Conversion between Canonical Form

Example:

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

This function has a complement that can be expressed as

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Now, if we take the complement of F' by DeMorgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

The last conversion follows from the definition of minterms and maxterms as shown in Table 2.3. From the table, it is clear that the following relation holds:

$$m_j' = M_j$$

That is, the **maxterm with subscript *j* is a complement of the minterm with the same subscript *j* and vice versa.**

Standard Forms

- Boolean function can be expressed in Standard form.
- In standard form, the terms that form the function may contain one, two or any number of literals.
- There are two types of standard forms:
  i. The sum of products (SOP)
  ii. The Product of Sum (SOP)

SOP

- SOP of a Boolean expression containing AND terms, called product terms, of one or more literals.
- The sum denotes the ORing of these terms.
- For example:

$$F_1 = y' + xy + x' y' z'$$

POS

- A POS is a Boolean expression containing OR terms, called sum terms of one or more literals each. The product denotes the ANDing of these terms.
- For example:

$$F_2 = x(y' + z)\,(x' + y + z')$$

Non-Standard Form

- A Boolean function may be expressed in a non-standard form.
- Example: $F3 = (AB + CD)(A' B' + C' D')$

Simplification of Boolean Functions: The Map Method

- Algebric method lacks specific rules to predict each succeeding step in the manipulative process.
- The map methods provide a simple straightforward procedure for minimzing Boolean functions.
- It is a pictorial form of a truth table.
- It is first proposed by Veitch and slightly modified by Karnaugh, so known as "Veitch Diagram" or "Karnaugh Map".

- The map is a diagram made up of squares.

- Each square represent one minterm.

- Since any Boolean function can be expressed as a sum of min terms, it follows that a Boolean function is recognized graphically in the map from the area enclosed by those squares whose midterms are included in the function.

Type of K-map

1. 2 variable k-map
2. 3 variable k-map
3. 4 variable k-map
4. 5 variable k-map
5. 6 variable k-map

Rule of simplification using k-map

1. Enter a 1 on the Karnaugh map for each fundamental product that produces a 1 output in the truth table. Enter 0s elsewhere.
2. Encircle the octets, quads, and pairs. Remember to roll and overlap to get the largest groups possible.
3. If any isolated 1's remain, encircle them.
4. Eliminate any redundant group.
5. Write the Boolean equation by ORing the products corresponding to encircled groups.

Don't care conditions

- In some digital systems, certain input conditions never occur during normal operation; therefore, the corresponding output never appears. Since the output never appears, it is indicated by an ***X*** in the truth table.

K-map with don't care

1. Given the truth table, draw the Karnaugh map with 0s, 1s and don't cares.
2. Encircle the actual 1's on the Karnaugh map in the largest groups you can find by treating the don't cares as 1's.
3. After the actual 1's have been included in groups, disregard the remaining don't cares by visualizing them as 0's.

NAND and NOR Implementation

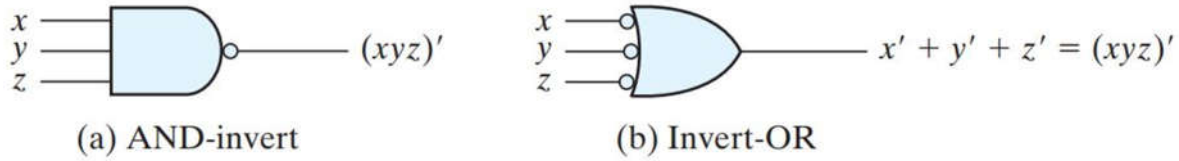


Figure: Two graphic symbols for a three input NAND gate

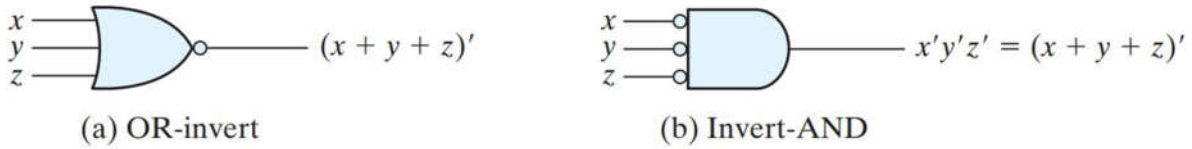


Figure: Two graphic symbols for a three input NOR gate

NAND Implementation example

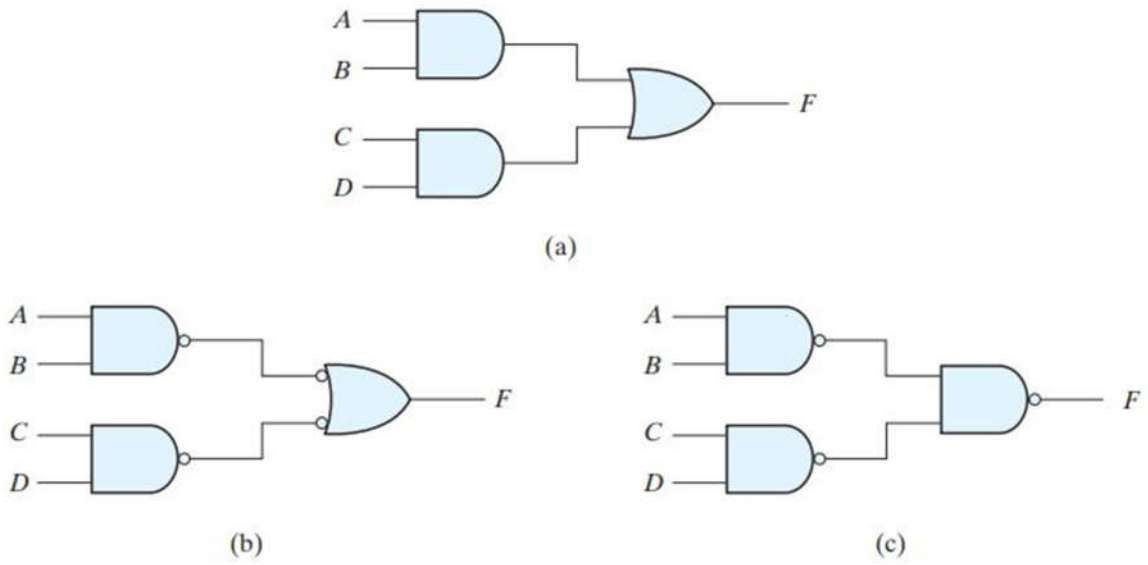


Figure: Three way to implement the function F = AB + CD

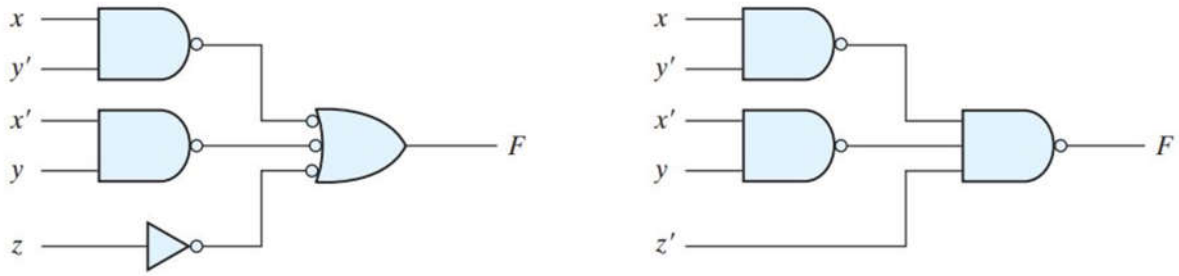
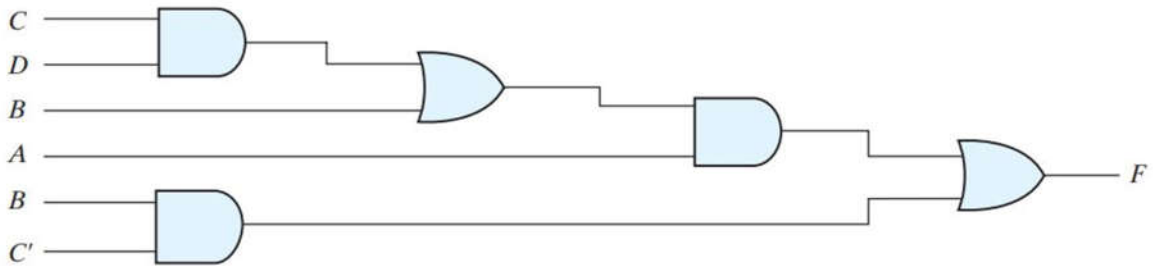
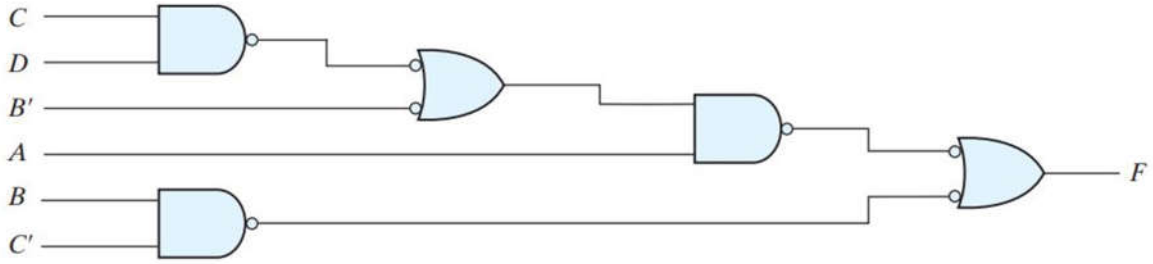


Figure: NAND implementation of the function F = xy' + x'y + z



(a) AND–OR gates



(b) NAND gates

Figure: Multilevel NAND implementation of the function F = A(CD + B) + BC'

Combinational Circuit

- Combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination of inputs without regard to previous inputs.
- It consists of input variables, logic gates and output variable.
- The logic gates signals from the inputs and generates signals to the outputs.
- For 'n' input variables, there are 2n possible combinations of binary input values.
- For each possible input combination, there is one and only possible output combinations.
- A combinational circuit can be described by 'm' Boolean functions one for each output variable.
- Each output is expressed in terms of the 'n' input variables.

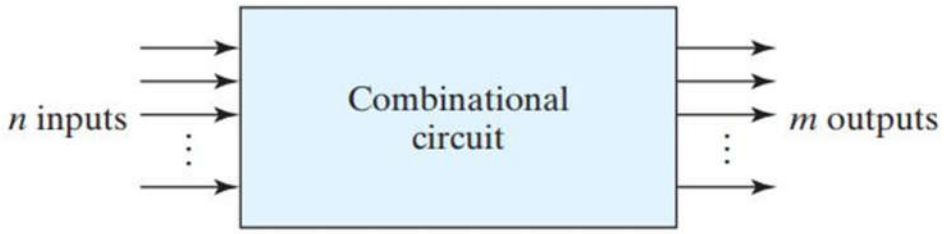


Figure: Block Diagram of Combinational Circuit

Design Procedure

1. The problem is stated.
2. The number of available input variables and required variables are determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean expression for each output is obtained.
6. The logic diagram is drawn.

Differences between combinational circuit and sequential circuit.

Combinational Circuit	Sequential Circuit
1. In this output depends only upon present input.	1. In this output depends upon present as well as past input.
2. Speed is fast.	2. Speed is slow.
3. It is designed easy.	3. It is designed tough as compared to combinational circuits.
4. There is no feedback between input and output.	4. There exists a feedback path between input and output.
5. This is time independent.	5. This is time dependent.
6. Elementary building blocks: Logic gates	6. Elementary building blocks: Flip-flops
7. Used for arithmetic as well as boolean operations.	7. Mainly used for storing data.
8. Combinational circuits don't have capability to store any state.	8. Sequential circuits have capability to store any state or to retain earlier state.
9. As combinational circuits don't have clock, they don't require triggering.	9. As sequential circuits are clock dependent they need triggering.
10. These circuits do not have any memory element.	10. These circuits have memory element.
11. It is easy to use and handle.	11. It is not easy to use and handle.
Examples – Encoder, Decoder, Multiplexer,	**Examples –** Flip-flops, Counters

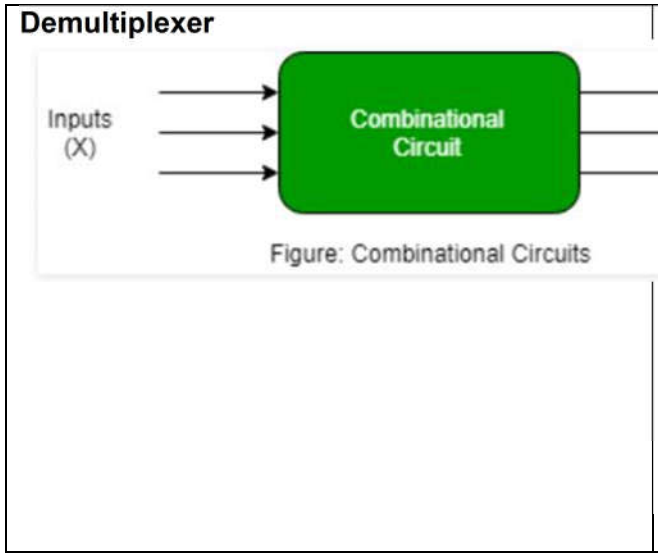


Figure: Combinational Circuits

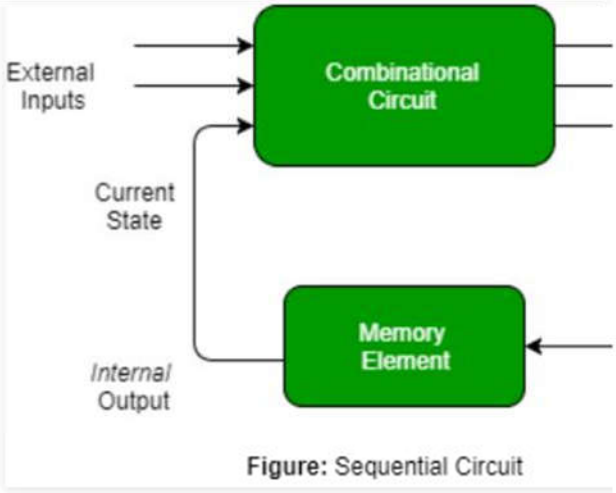


Figure: Sequential Circuit

Following Combinational Circuits Design will be discussed:

1. Adder
   i. Half Adder
   ii. Full Adder
   iii. Binary Parallel Adder
   iv. Serial Adder
   v. Carry-Look Ahead Adder
2. Subtractor
   i. Half Subtractor
   ii. Full Subtractor
3. Adder-Subtractor Circuit
4. Decoder
5. Encoder
6. Multiplexer
7. De-Multiplexer
8. Code Converter
9. Magnitude Converter
10. Error Detector Circuit
11. Read Only Memory8
12. Programmable Array Logic
13. Programmable Logic Array