Chapter 2: Digital Components

2.1 Logic Gates

Digital Computers

Digital electronics, digital circuits, and digital technology are electronics that are operated on digital signals. Digital circuits are made from a large collection of logic gates and a simple electronic representation of the Boolean logic function.

A Digital computer can be considered as a digital system that performs various computational tasks. The first electronic digital computer was developed in the late 1940s and was used primarily for numerical computations. By convention, the digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit.

Digital electronic circuits operate with voltages of two logic levels namely Logic Low and Logic High. The range of voltages corresponding to Logic Low is represented with '0'. Similarly, the range of voltages corresponding to Logic High is represented with '1'.The basic digital electronic circuit that has one or more inputs and single output is known as **Logic gate.**

Basic Gates

Logic gates play an important role in circuit design and digital systems. It is a building block of a digital system and an electronic circuit that always has only one output. These gates can have one input or more than one input, but most of the gates have two inputs.

These gates are the AND, OR, NOT, NAND, NOR, EXOR and EXNOR gates. The basic operations are described below with the aid of truth tables.

AND gate

An AND gate is a digital circuit that has two or more inputs and produces an output, which is the logical AND of all those inputs. The AND operation is represented by a dot (.) sign.

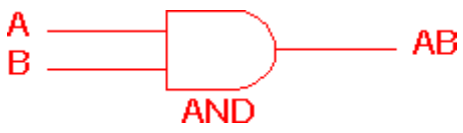


Figure 2.1: Circuit of AND gate

Table 2.1: Truth table of AND gate

2 Input AND gate		
A	B	A.B
0	0	0
0	1	0
1	0	0
1	1	1

OR gate

An OR gate is a digital circuit that has two or more inputs and produces an output, which is the logical OR of all those inputs. The operation performed by an OR gate is represented by a plus (+) sign.

Figure 2.2: Circuit of OR gate

Table 2.2: Truth table of OR gate

2 Input OR gate		
A	B	A+B
0	0	0
0	1	1
1	0	1
1	1	1

NOT gate

The NOT gate is an electronic circuit which produces an inverted version of the input at its output. It is also known as an Inverter.

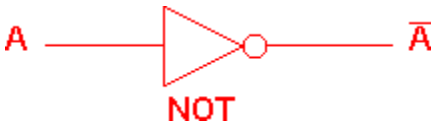


Figure 2.3: Circuit of NOT gate

Table 2.3: Truth table of NOT gate

NOT gate	
A	$\overline{A}$
0	1
1	0

NAND gate

This is a NOT-AND gate which is equal to an AND gate followed by a NOT gate. The outputs of all NAND gates are high if any of the inputs are low. The symbol is an AND gate with a small circle on the output. The small circle represents inversion.

Figure 2.4: Circuit of NAND

Table 2.4: Truth table of NAND gate

2 Input NAND gate		
A	B	$\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate

This is a NOT-OR gate which is equal to an OR gate followed by a NOT gate. The outputs of all NOR gates are low if any of the inputs are high. The symbol is an OR gate with a small circle on the output. The small circle represents inversion.

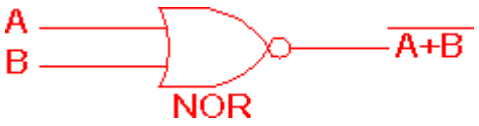


Figure 2.5: Circuit of NOR

Table 2.5: Truth table of NOR gate

2 Input NOR gate		
A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

EXOR gate

The 'Exclusive-OR' gate is a circuit which will give a high output if either, but not both, of its two inputs are high. An encircled plus sign (⊕) is used to show the EOR operation.

Figure 2.6: circuit of

Table 2.6: Truth table of NAND gate

2 Input EXOR gate		
A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

EXNOR gate

The 'Exclusive-NOR' gate circuit does the opposite to the EOR gate. It will give a low output if either, but not both, of its two inputs are high. The symbol is an EXOR gate with a small circle on the output. The small circle represents inversion

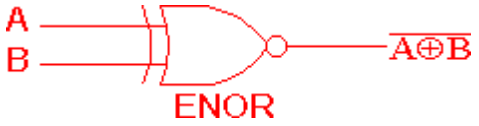


Figure 2.7: Circuit of EXNOR

Table 2.7: Truth table of ENOR gate

2 Input EXNOR gate		
A	B	$\overline{A \oplus B}$
0	0	1
0	1	0
1	0	0
1	1	1

2.2 Simplification of Boolean Functions

Boolean algebra

Boolean algebra can be considered as an algebra that deals with binary variables and logic operations. Boolean algebraic variables are designated by letters such as A, B, x, and y. The basic operations performed are AND, OR, and complement.

The Boolean algebraic functions are mostly expressed with binary variables, logic operation symbols, parentheses, and equal sign. For a given value of variables, the Boolean function can be either 1 or 0. For instance, consider the Boolean function:

$$F = x + y'z$$

The logic diagram for the Boolean function F = x + y'z can be represented as:

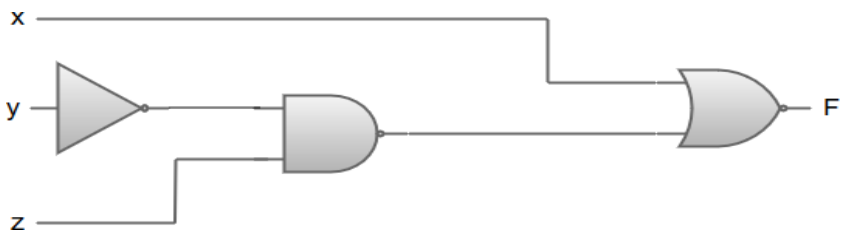


Figure 2.19: F = x + y'z

The truth table for the Boolean function F = x + y'z can be represented as:

Table 2.8: Truth table of F== x + $y'z$

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Laws of Boolean algebra

There are six types of Boolean algebra laws. They are Commutative law, Associative law, Distributive law, AND law, OR law and Inversion law

Commutative Law

- **A.B = B. A**
- **A + B = B + A**

Associative Law

- **(A. B). C = A. (B.C)**
- **(A + B) + C = A + (B + C)**

Distributive Law

- **A. (B + C) = (A. B) + (A. C)**
- **A + (B. C) = (A + B).(A + C)**

AND Law

- **A.0 = 0**
- **A.1 = A**
- **A. A = A**
- **A.$\bar{A}$ = 0**

OR Law

- **A + 0 = A**
- **A + 1 = 1**
- **A + A = A**
- **A+$\bar{A}$ = 1**

Inversion Law

- $\bar{\bar{A}}$ = A

Boolean algebra Theorems

De Morgan's First Law:

- **De Morgan's First Law states that (A.B)' = A'+B'.**

De Morgan's Second Law:

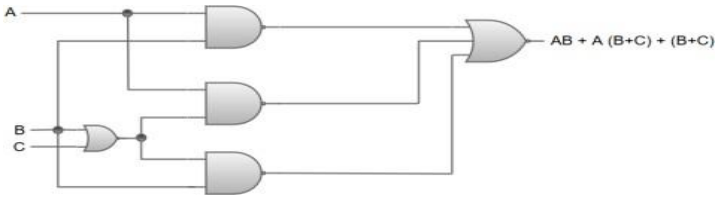
- **De Morgan's Second law states that (A+B)' = A'. B'.**

Example Questions

1. Simplify the following expression, create a truth table, and draw circuit

Let us consider an example of a Boolean function: **AB+A (B+C) + B (B+C)**

The logic diagrams for the Boolean function AB+A (B+C) + B (B+C) can be represented as:



We will simplify this Boolean function based on rules given by Boolean algebra.

AB + A (B+C) + B (B+C)

AB + AB + AC + BB + BC {Distributive law; A (B+C) = AB+AC, B (B+C) = BB+BC}

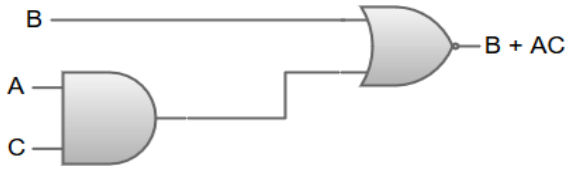
AB + AB + AC + B + BC {Idempotent law; BB = B}

AB + AC + B + BC {Idempotent law; AB+AB = AB}

AB + AC +B {Absorption law; B+BC = B}

B + AC {Absorption law; AB+B = B}

Hence, the simplified Boolean function will be B + AC. The logic diagram for Boolean function B + AC can be represented as:



2. Simplify the following expression

- $(\overline{\overline{A}}\overline{B})$

  Using De Morgan's and double negation: $\overline{\overline{A}} + \overline{\overline{B}}$

  AB

- $A.B+A.\overline{B}$

  Using distributive Rule: $A.(B+\overline{B})$

  $B+\overline{B}=1$

  A.1 = A

- A.(B+1)

  B+1 = 1

  A.1 = A

- $\overline{\overline{A}}\overline{B}+\overline{A}\overline{B}$

  Using De Morgan's Rule $(\bar{A}+\overline{B})+(\bar{A}+B)$

  $\bar{A}+\bar{A}=\bar{A}$

  $\bar{A}+B+\overline{B}$

  $\bar{A}+1$

  $\bar{A}$ + 1 = 1

- $(X+Y).(X+\overline{Y})$

  Using distributive Rule: we get X + (Y + $\overline{Y}$)

  Using Rule 1: this gives us X + 0

  Rule 1 says that X + 0 = X

2.3 Digital Circuit

A digital circuit is **a, electric circuit where the signal is either of the two discrete levels – ON / OFF or 0 / 1 or True / False**. Transistors are used to create logic gates perform Boolean logic. Software like Electronic Design Automation (EDA / ECDA) is used to design digital circuits.

Types of Digital Circuit

Logic circuits for digital systems may be combinational logic or sequential logic. Digital Circuits are divided into following 2 Types:

1. Combinational Circuits

A Circuit that gives the same output when given the same inputs. It is represents a set of logic functions.

Examples: Multiplexers, De-multiplexers, Encoders, Decoders, Full and Half Adders etc.

2. Sequential Circuits

A sequential circuit is a combinational circuit with some of the outputs fed back as inputs. These circuits perform a sequence of operations.

Examples: Shift Registers, Counter, Flip-Flop

Both Combinational Circuits and Sequential Circuits can further be classified into smaller types. But for ease of understanding I am not mentioning them here or things will get confusing.

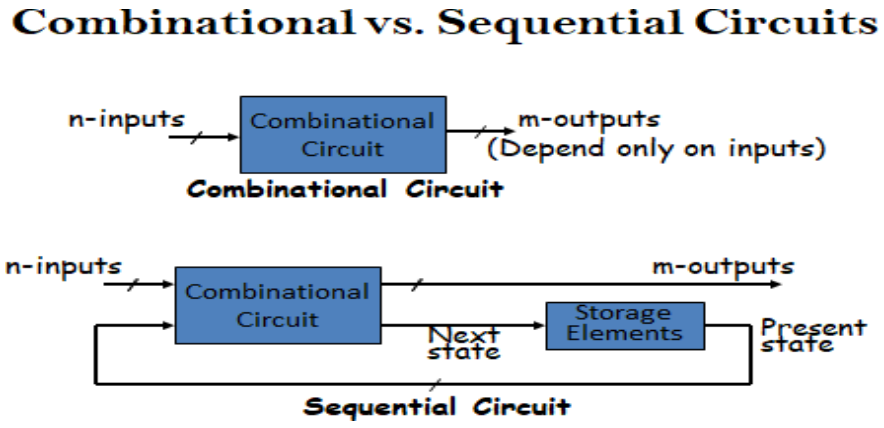


Figure 2.38: Block diagram Combinational Circuits and Sequential

2.3.1 Combinational Circuits

- Combinational circuits are memory-less. Thus, the output value depends ONLY on the current input values.
- A combinational circuit consists of input variables, logic gates, and output variables.
- where the outputs are determined solely by the current states of the inputs
- Has no memory
- Output depends only on current input

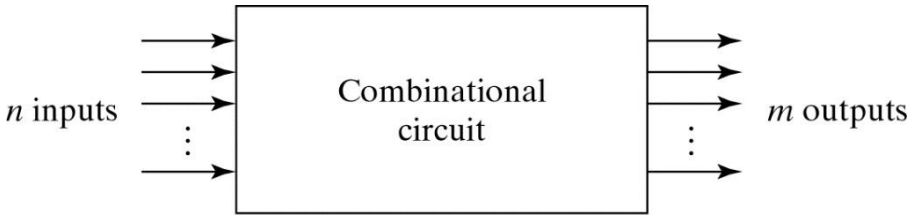


Figure 2.39: Block diagram Combinational Circuits

1. Hence, a combinational circuit can be described by:

- A truth table that lists the output values for each combination of the input **(n)** variables, ***m*** Boolean functions, one for each output variable.

  Common type of combinational circuit
  - Half and Full Adders,
  - Multiplexers and De-multiplexers, and Encoders, and Decoders, etc

The design of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram.

- The procedure involves the following steps:
  1. The problem is stated.
  2. The input and output variables are assigned letter symbols.
  3. The truth table that defines the relationship between inputs and outputs is derived.
  4. The simplified Boolean functions for each output are obtained.
  5. The logic diagram is drawn.

A. Half Adders

- Half adder is a combinational logic circuit with two inputs and two outputs.
- The half adder circuit is designed to add two single bit binary number A and B.
- It is the basic building block for addition of two single bit numbers.
- This circuit has two outputs carry and sum.

Figure 2.40: Block diagram of half adder

Step 1. Truth table

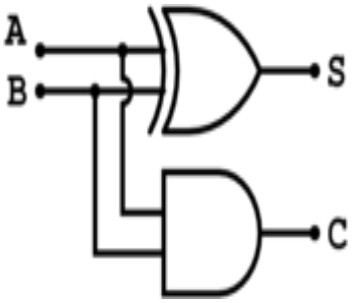
A + B = S with C

Where S= sum and C= carry

Inputs		Output	
A	B	S	C
0	0	0	0
0	1	1	0
0	1	1	0
1	1	0	1

Table 2.9: Truth table of half adder

Step 2. Circuit Diagram



Circuit diagram of half adder

B. Full adder

- Full adder developed to overcome the drawback of Half Adder circuit.
- It can add two one-bit numbers A and B, and carry c.
- The full adder is a three input and two output combinational circuit.
- It used to add more than 1 bit numbers
- It known as cascading half adder

- IC / Integrate circuit for 4-bits and 8-bits full adder.

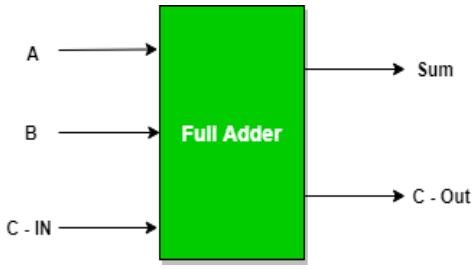


Figure 2.42: block diagram of Full adder

Table 2.10: Truth table of full adder

Inputs			Outputs	
A	B	C – IN	Sum	C - Out
0	0	0	0	0
0	0	1	1	0
0	1	0	1	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

Step 1: Logical Expression for SUM:

- A’ B’ C-IN + A’ B C-IN’ + A B’ C-IN’ + A B C-IN
- C-IN (A’ B’ + A B) + C-IN’ (A’ B + A B’)
- C-IN XOR (A XOR B)
- (1,2,4,7)

Step2: Logical Expression for C-OUT

- A’ B C-IN + A B’ C-IN + A B C-IN’ + A B C-IN
- A B + B C-IN + A C-IN
- (3,5,6,7)

Step3: Another form: in which C-OUT can be implement

- A B + A C-IN + B C-IN (A + A’)
- A B C-IN + A B + A C-IN + A’ B C-IN
- A B (1 +C-IN) + A C-IN + A’ B C-IN
- A B + A C-IN + A’ B C-IN
- A B + A C-IN (B + B’) + A’ B C-IN
- A B C-IN + A B + A B’ C-IN + A’ B C-IN
- A B (C-IN + 1) + A B’ C-IN + A’ B C-IN
- A B + A B’ C-IN + A’ B C-IN
- AB + C-IN (A’ B + A B’)

Therefore COUT = AB + C-IN (A EX – OR B)

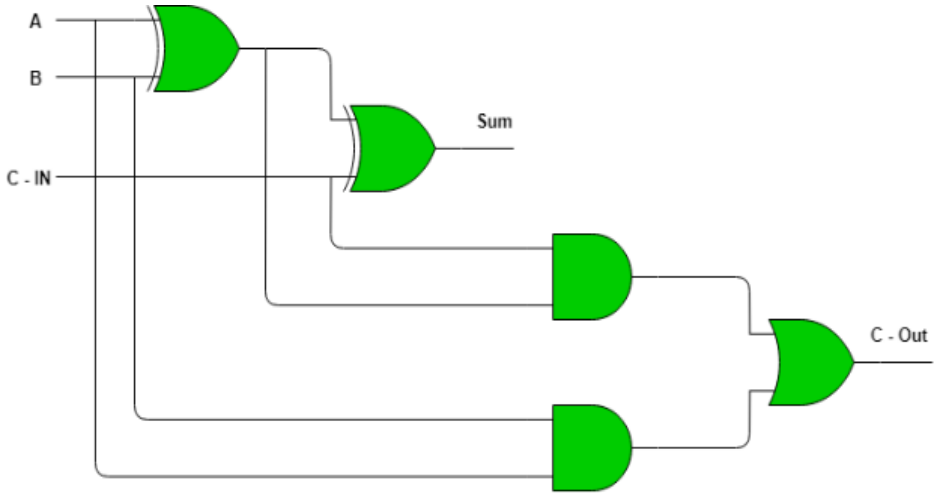


Figure 2.43: Circuit diagram of Full adder

Implementation of Full Adder using Half Adders

- Two Half Adders and OR gate is required to implement a Full Adder.

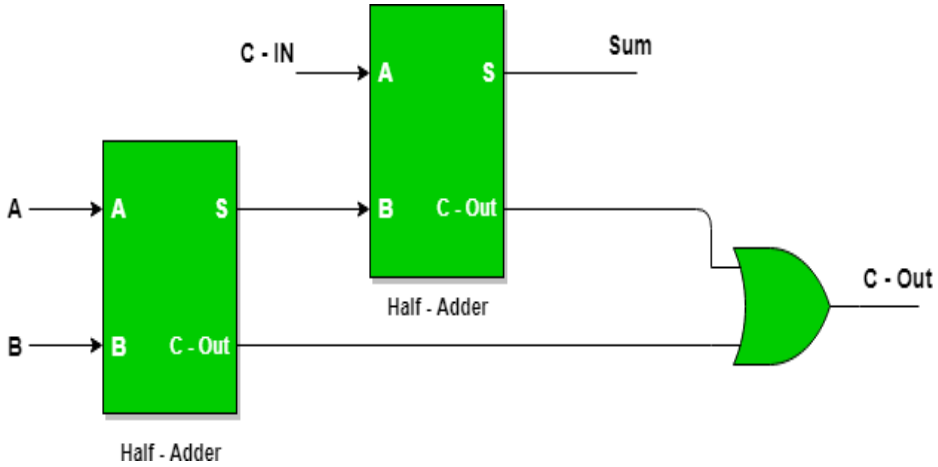


Figure 2.44: Circuit diagram of Full and half adder

- With this logic circuit, two bits can be added together, taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude.

Encoders and Decoders in Digital Logic

Binary code of N digits can be used to store 2^N distinct elements of coded information. This is what encoders and decoders are used for. **Encoders** convert 2^N lines of input into a code of N bits and **Decoders** decode the N bits into 2^N lines.

1. Encoder

An encoder is a combinational circuit that converts binary information in the form of a 2^N input lines into N output lines, which represent N bit code for the input. For simple encoders, it is assumed that only one input line is active at a time.

As an example, let's consider **Octal to Binary** encoder. As shown in the following figure, an octal-to-binary encoder takes 8 input lines and generates 3 output lines.

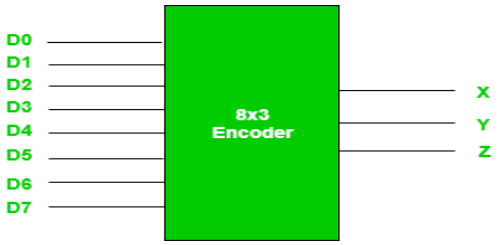


Figure 2.47: Block diagram of Encoder

Truth Table

Table 2.11: Truth table of Encoder

D7	D6	D5	D4	D3	D2	D1	D0	X	Y	Z
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

As seen from the truth table, the output is 000 when D0 is active; 001 when D1 is active; 010 when D2 is active and so on.

Implementation –

From the truth table, the output line Z is active when the input octal digit is 1, 3, 5 or 7. Similarly, Y is 1 when input octal digit is 2, 3, 6 or 7 and X is 1 for input octal digits 4, 5, 6 or 7. Hence, the Boolean functions would be:

X = D4 + D5 + D6 + D7

Y = D2 +D3 + D6 + D7

Z = D1 + D3 + D5 + D7

Hence, the encoder can be realized with OR gates as follows:

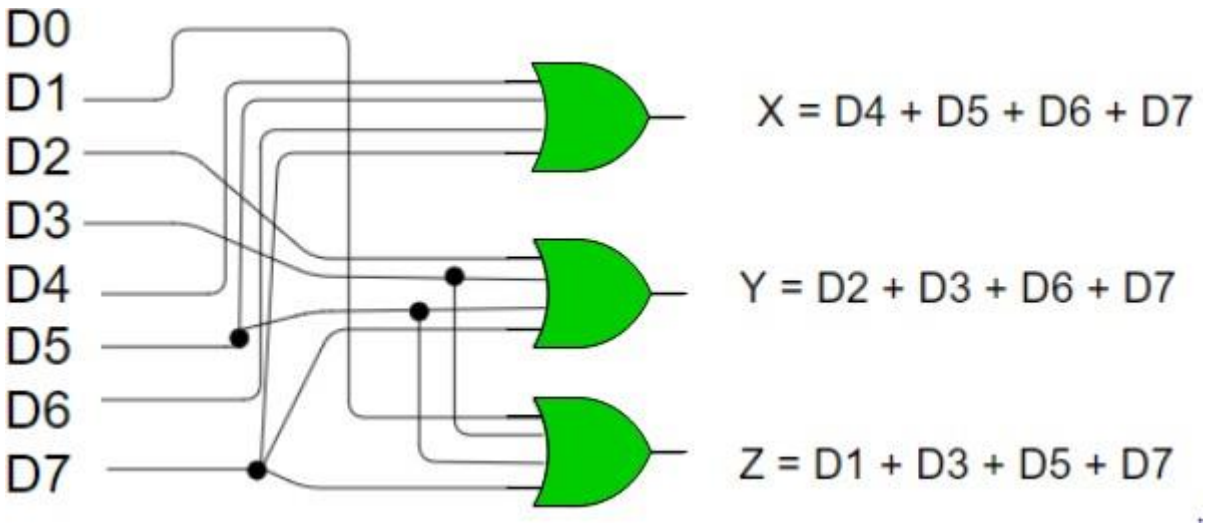


Figure 2.48: Circuit diagram of Encoder

One limitation of this encoder is that only one input can be active at any given time. If more than one inputs are active, then the output is undefined. For example, if D6 and D3 are both active, then, our output would be 111 which is the output for D7. To overcome this, we use Priority Encoders.

Another ambiguity arises when all inputs are 0. In this case, encoder outputs 000 which actually is the output for D0 active.

In order to avoid this, an extra bit can be added to the output, called the valid bit which is 0 when all inputs are 0 and 1 otherwise.

Priority Encoder –

A priority encoder is an encoder circuit in which inputs are given priorities. When more than one inputs are active at the same time, the input with higher priority takes precedence and the output corresponding to that is generated.

Let us consider the 4 to 2 priority encoder as an example. From the truth table, we see that when all inputs are 0, our V bit or the valid bit is zero and outputs are not used. The x's in the table show the don't care condition, i.e, it may either be 0 or 1. Here, D3 has highest priority, therefore, whatever be the other inputs, when D3 is high, output has to be 11. And D0 has the lowest priority, therefore the output would be 00 only when D0 is high and the other input lines are low. Similarly, D2 has higher priority over D1 and D0 but lower than D3 therefore the output would be 010 only when D2 is high and D3 are low (D0 & D1 are don't care).

Truth Table –

Table 2.12: Truth table of Encoder priority

D3	D2	D1	D0	X	Y	V

D3	D2	D1	D0	X	Y	V
0	0	0	0	x	x	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Implementation –

It can clearly be seen that the condition for valid bit to be 1 is that at least any one of the inputs should be high. Hence,

V = D0 + D1 + D2 + D3

For X:

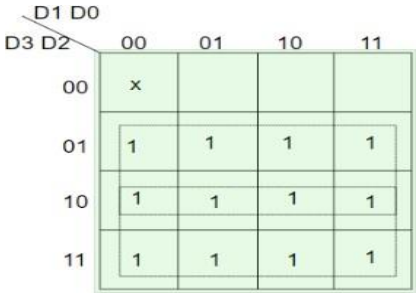


Figure 2.49.1: K-map diagram of Encoder

X = D2 + D3

For Y:

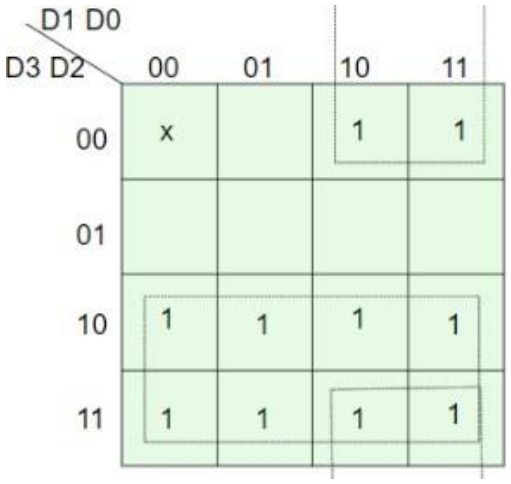


Figure 2.49.2: K-map diagram of Encoder

$$Y = D1\ D2' + D3$$

Hence, the priority 4-to-2 encoder can be realized as follows:

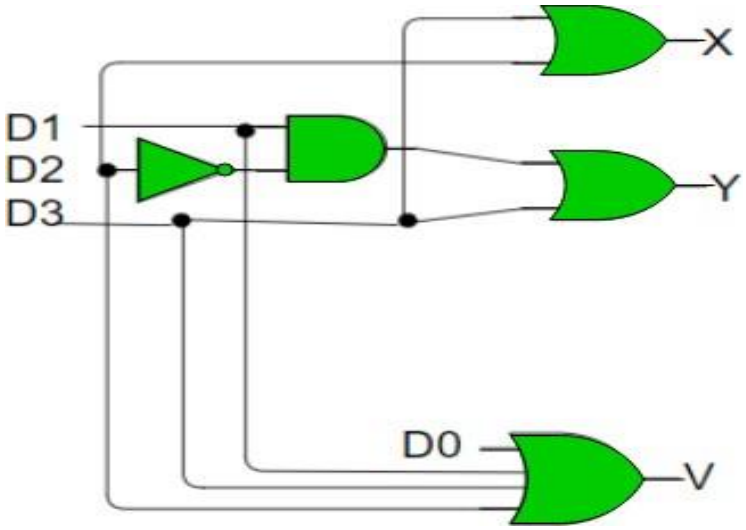


Figure 2.50: Circuit diagram of Encoder

Y = D1 D2' + D3

Decoders

A decoder does the opposite job of an encoder. It is a combinational circuit that converts n lines of input into 2^n lines of output.

Let's take an example of 3-to-8 line decoder.

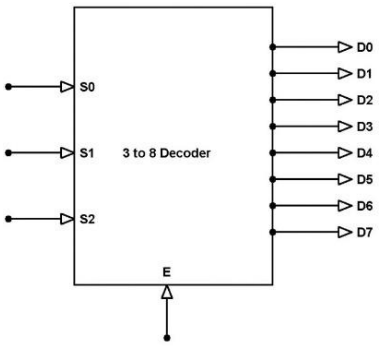


Figure 2.51: Circuit diagram of Encoder 3 to 8

Truth Table –

Table 2.13: Truth table of Decoder

X	Y	Z	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

Implementation –

- D0 is high when X = 0, Y = 0 and Z = 0. Hence,
- D0 = X’ Y’ Z’

Similarly,

- D1 = X’ Y’ Z
- D2 = X’ Y Z’

- D3 = X’ Y Z
- D4 = X Y’ Z’
- D5 = X Y’ Z
- D6 = X Y Z’
- D7 = X Y Z

Hence,

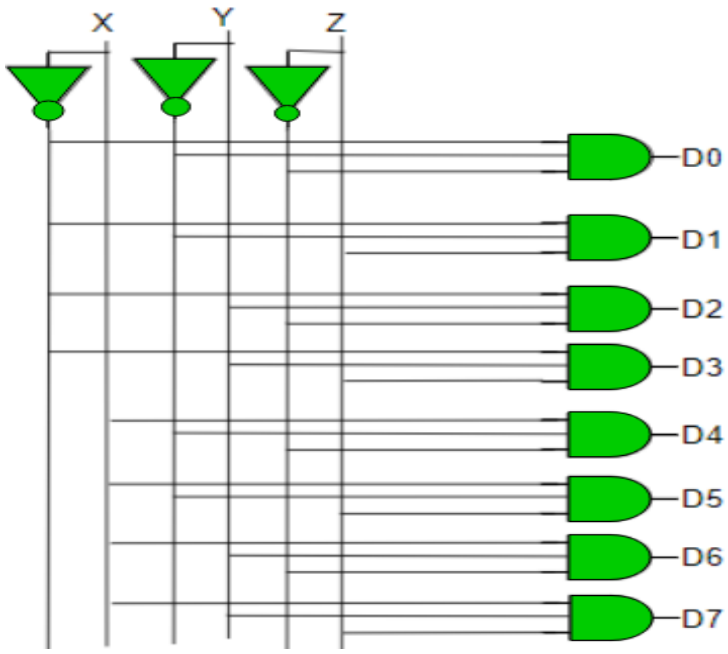


Figure 2.52: Circuit diagram of Decoder

Multiplexer

It is a combinational circuit which has many data inputs and single output depending on control or selects inputs. For N input lines, log n (base2) selection lines, or we can say that for 2^n input lines, n selection lines are required. Multiplexers are also known as **“Data n selector, parallel to serial convertor, many to one circuit, universal logic circuit”**. Multiplexers are mainly used to increase amount of the data that can be sent over the network within certain amount of time and bandwidth.

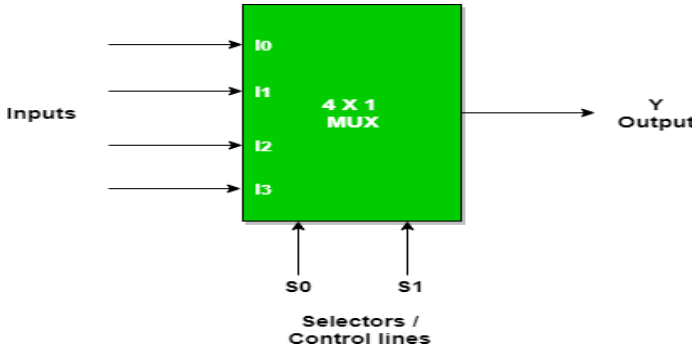


Figure 2.53: Block diagram of Multiplexer

Now the implementation of 4:1 Multiplexer using truth table and gates.

Table 2.14: Truth table of 4:1 Multiplexer

S0	S1	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

So, final equation, Y=S0'.S1'.I0+S0'.S1.I1+S0.S1'.I2+S0.S1.S3

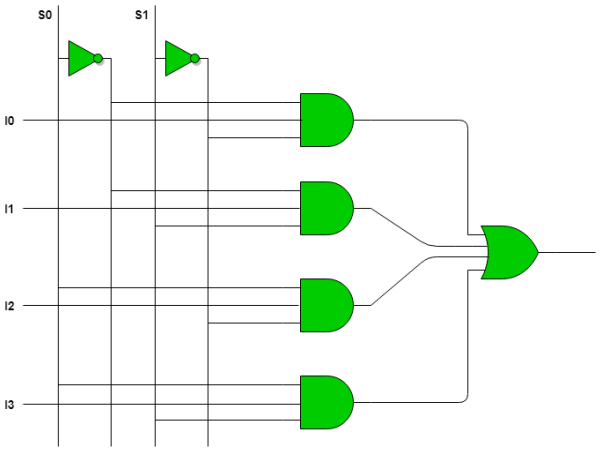
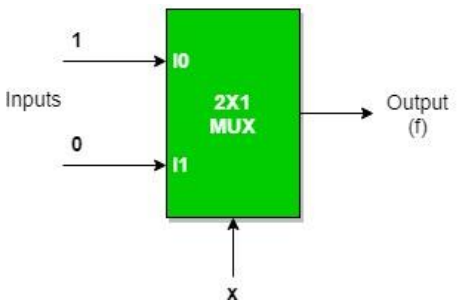


Figure 2.54: Circuit diagram of 4:1 Multiplexer

Multiplexer can act as universal combinational circuit. All the standard logic gates can be implemented with multiplexers.

Example1: **Implementation of NOT gate using 2 : 1 Mux**

- **NOT Gate :**



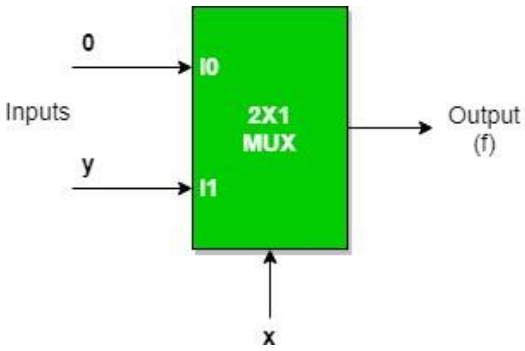
Truth Table

x	f
0	1
1	0

We can analyze it
Y = x'.1 + x.0 = x'
It is NOT Gate using 2:1 MUX.
The implementation of NOT gate is done using "n" selection lines. It cannot be implemented using "n-1" selection lines. Only NOT gate cannot be implemented using "n-1" selection lines.

Example 2: Implementation of AND gate using 2 : 1 Mux

- **AND GATE**



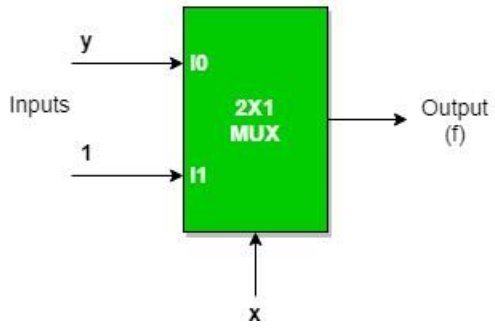
Truth Table

x	y	f	f→y
0	0	0	f = 0
0	1	0	
1	0	0	f = y
1	1	1	

This implementation is using "n-1" selection lines.

Example 3: Implementation of OR gate using 2 : 1 Mux using "n-1" selection lines.

- **OR GATE**



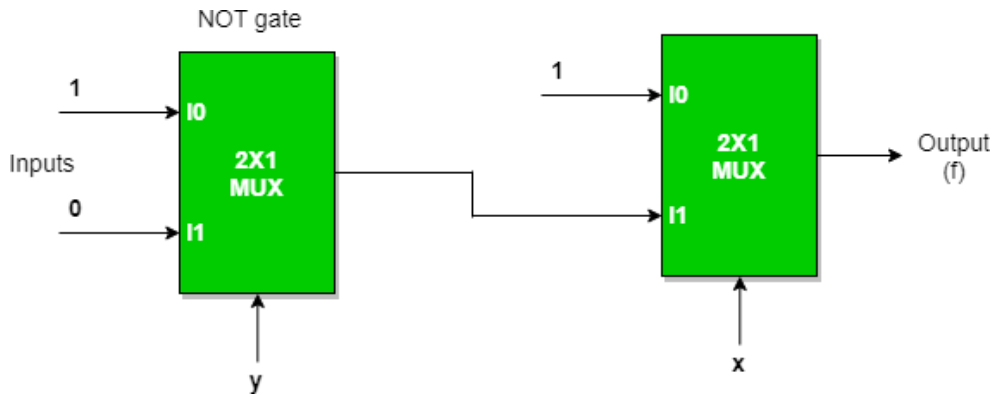
Truth Table

x	y	f	f→y
0	0	0	f = y
0	1	1	
1	0	1	f = 1
1	1	1	

Implementation of NAND, NOR, XOR and XNOR gates requires two 2:1 Mux. First multiplexer will act as NOT gate which will provide complemented input to the second multiplexer.

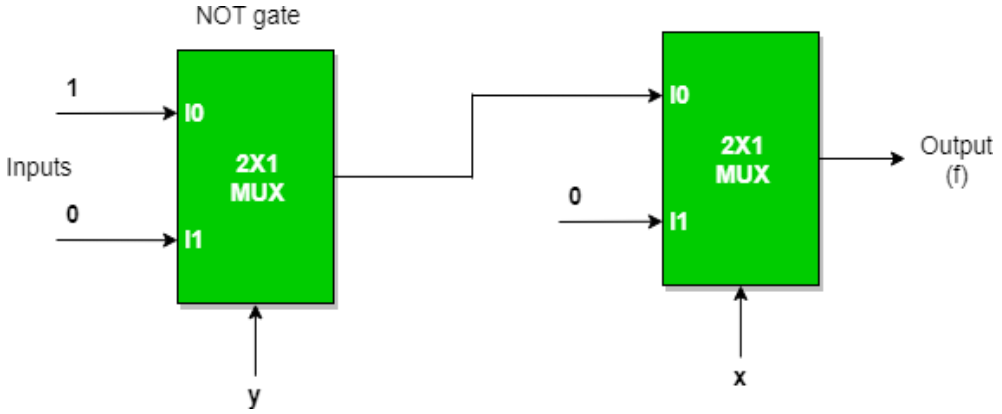
Example 4: Implementation of NAND gate using 2 : 1 Mux

- **NAND GATE**



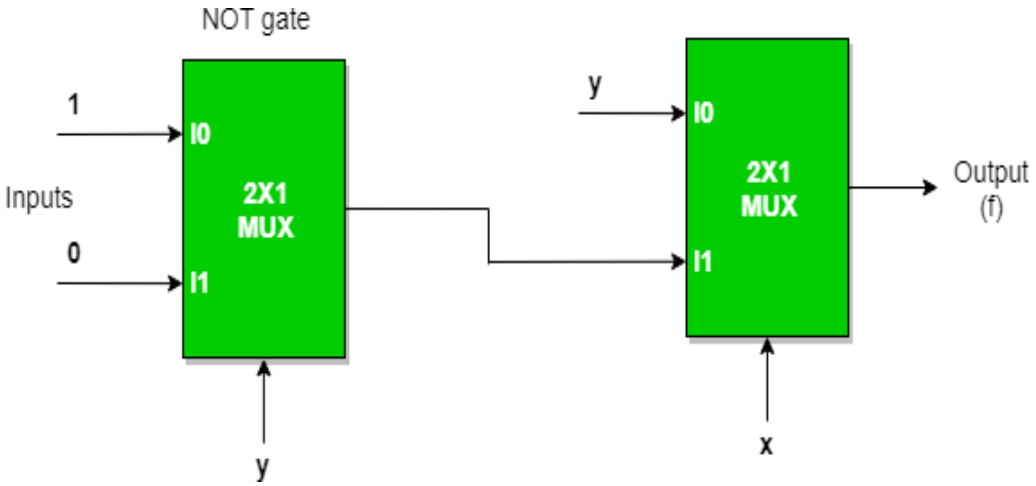
Example 5: Implementation of NOR gate using 2 : 1 Mux

- **NOR GATE**



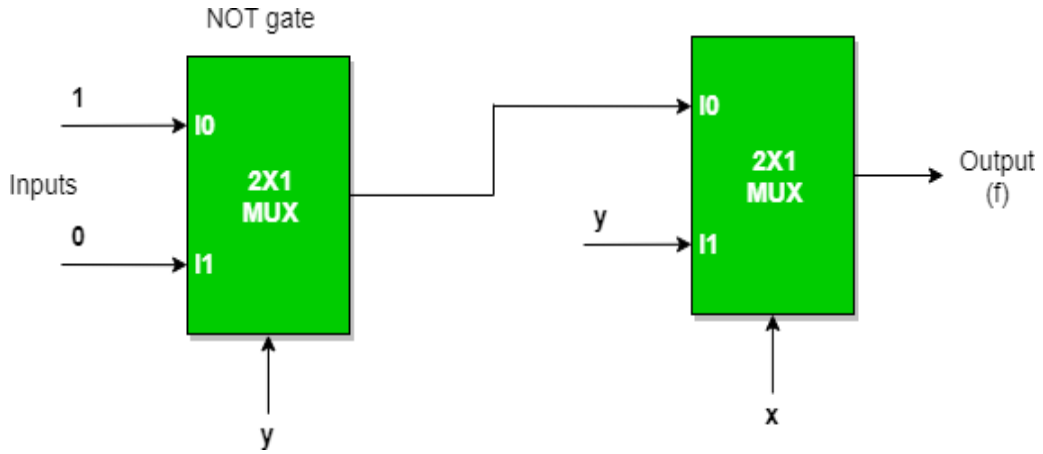
Example 6: Implementation of EX-OR gate using 2 : 1 Mux

- **EX-OR GATE**



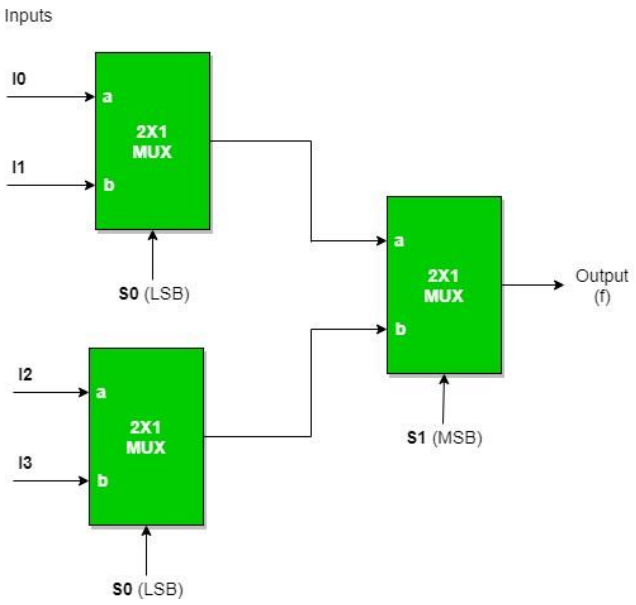
Example 7: Implementation of EX-NOR gate using 2 : 1 Mux

- **EX-NOR GATE**



Example 8: Implementation of Higher order MUX using lower order MUX (4 : 1 MUX using 2 : 1 MUX)

- 2 : 1 MUX are required to implement 4 : 1 MUX.



Truth Table

S0	S1	f
0	0	I0
0	1	I1
1	0	I2
1	1	I3

De-Multiplexers

A De-multiplexer (De-Mux) can be described as a combinational circuit that performs the reverse operation of a Multiplexer.
A De-multiplexer has a single input, 'n' selection lines and a maximum of 2^n outputs.
The following image shows the block diagram of a 1 * 4 De-multiplexer.

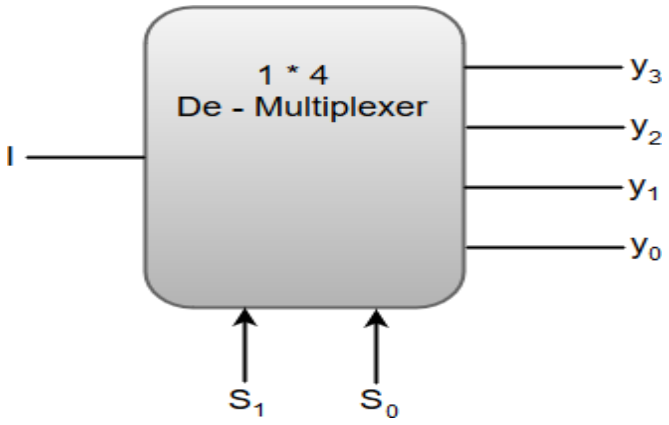


Figure 2.55: Block diagram of 1:4 De-Multiplexer

The function table for a 1 * 4 De - Multiplexer can be represent as:

Table 2.15: Truth table of 1:4 De- Multiplexer

S1	S0	y3	y2	y1	y0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

From the above function table, we can write the Boolean function for each output as:

y3 = S1S0 I, y2 = S1S0' I, y1 = S1' S0 I, y0 = S1'S0' I

The above equations can be implemented using inverters and three-input AND gates.

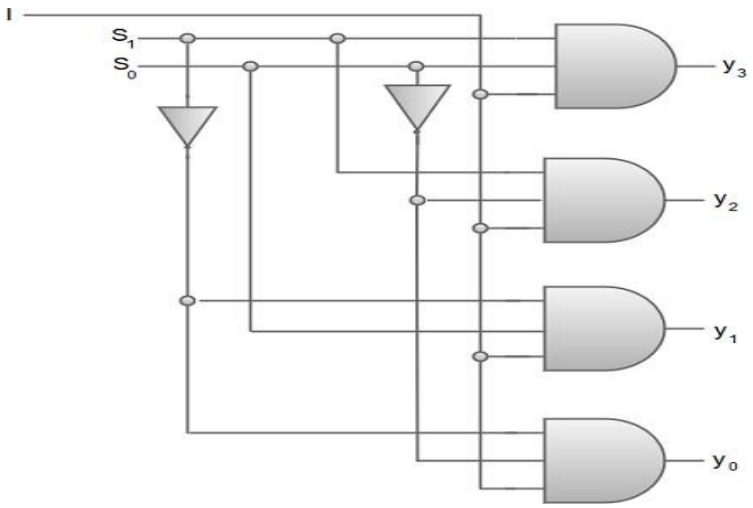


Figure 2.56: Circuit diagram of 1:4 De- Multiplexer

We can also implement higher order De-multiplexers using lower order De-multiplexers. For instance, let us implement a 1 * 8 De-multiplexer using 1 * 2 De-multiplexer in the first stage followed by two 1 * 4 De-multiplexers in the second stage.

The function table for a 1 * 8 De-multiplexer can be represent as:

Table 2.16: Truth table of 1:8 De- Multiplexer

S2	S1	S0	y7	y6	y5	y4	y3	y2	y1	y0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

The block diagram for a 1 * 8 De-multiplexer can be represent as:

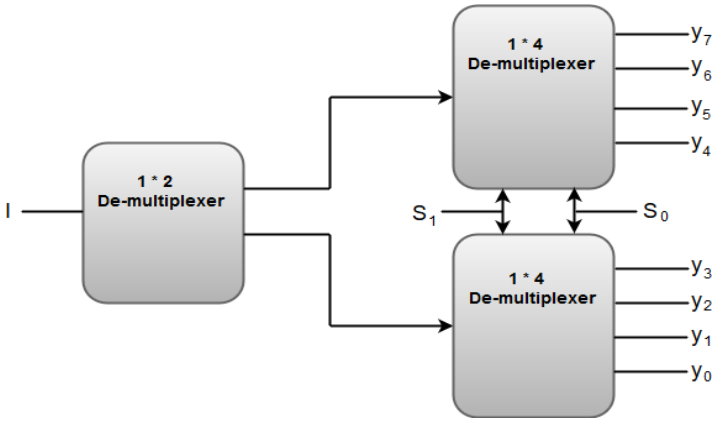


Figure 2.57: Circuit diagram of 1:8 De- Multiplexer

The Selection lines 'S1' and 'S0' are common for both of the 1 * 4 De-multiplexers.

2.3.2 Sequential Circuits

Sequential circuits have memory (i.e., remember the past).The current state is "held" in memory and the next state is computed based the current state and the current inputs. In synchronous systems, the clock signal orchestrates the sequence of events

Sequential Logic:

- where the outputs are determined not only by the current inputs but also by the sequence of inputs that led to the current state
- Output depends not only on current input but also on past input values, e.g., design a counter
- Need some type of memory to remember the past input values
- Outputs from the system are "feedback" as new inputs

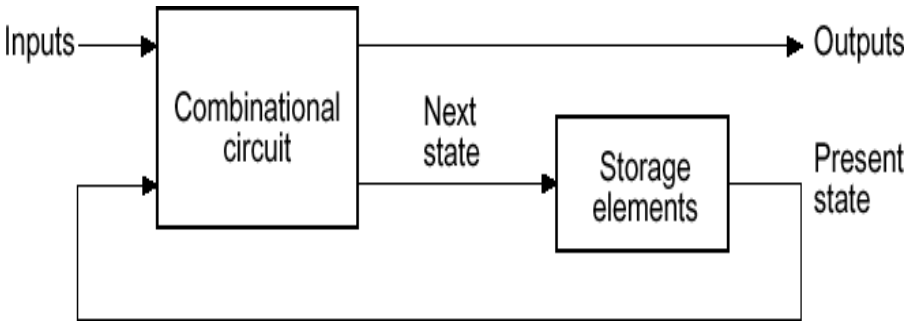


Figure 2.58: Block diagram of Sequential

The most common type of Sequential circuits

- Flip-Flop
- Counter and
- Shift Registers

Flip-Flop

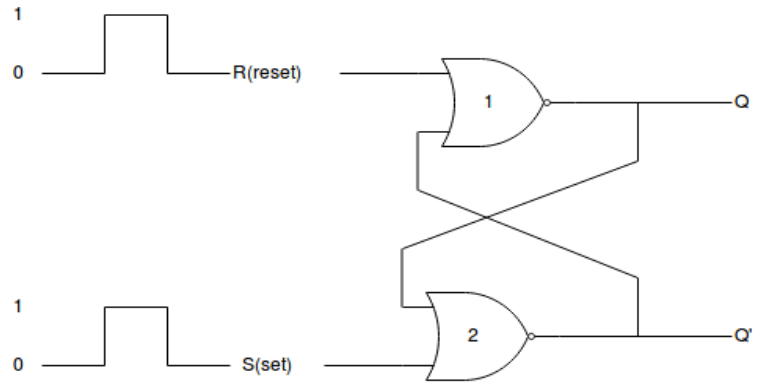
Flip flops are an application of logic gates. A flip-flop circuit can remain in a binary state indefinitely (as long as power is delivered to the circuit) until directed by an input signal to switch states.

Type of Flip-flops

1. SR Flip-Flop
2. D Flip-Flop
3. JK Flip-Flop
4. T Flip-Flop

Design and implement sequentialcircuit

1. **S-R flip-flop stands for SET-RESET flip-flops**
   The SET-RESET flip-flop consists of two NOR gates and two NAND gates. These flip-flops are called S-R Latch. The design of these flip-flops also includes two inputs, called the SET [S] and RESET [R]. There are also two outputs, Q and Q'.

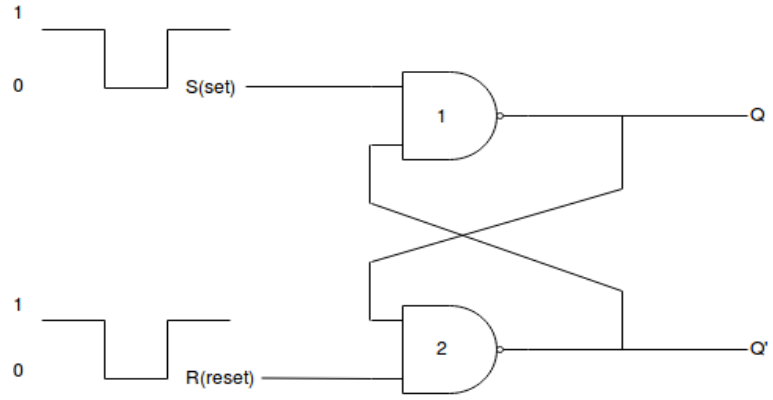


a) Logic diagram

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

b) Truth table

fig: Basic flip-flop circuit with NOR gates



a) Logic diagram

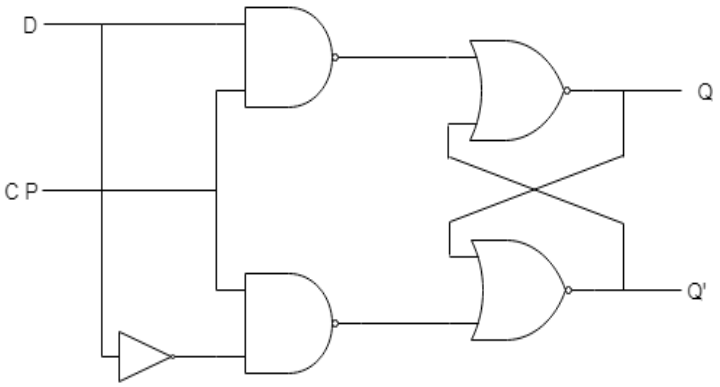
S	R	Q	Q'
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0
0	0	1	1

b) Truth table

fig: Basic flip-flop circuit with NAND gates

2. D Flip-Flop

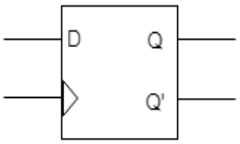
D flip-flop is a slight modification of clocked SR flip-flop.



(a) Logic diagram with Nand gates

Q D	Q (t + 1)
0 0	0
0 1	1
1 0	0
1 1	1

(c) Transition table



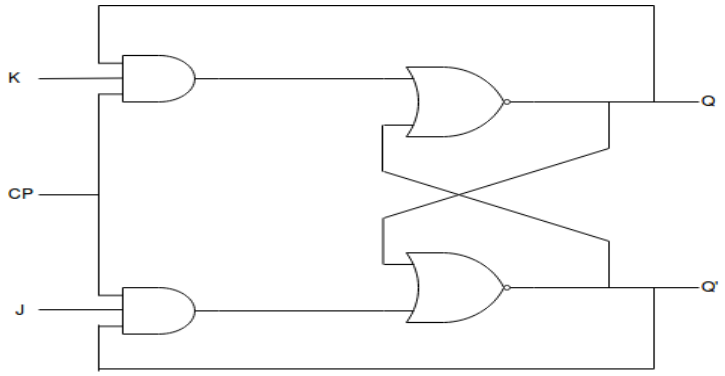
(b) Graphic Symbol

fig. Clocked D flip flop

From the above figure, you can see that the D input is connected to the S input and the complement of the D input is connected to the R input. When the value of CP is '1' (HIGH), the flip-flop moves to the SET state if it is '0' (LOW), the flip-flop switches to the CLEAR state.

3. J-K Flip-Flop

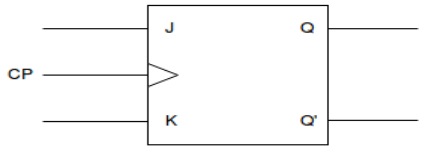
J-K flip-flop can be considered as a modification of the S-R flip-flop. The main difference is that the intermediate state is more refined and precise than that of an S-R flip-flop.



a) Logic diagram

Q	J	K	Q(t+1)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

b) Transition table



c) Graphical symbol

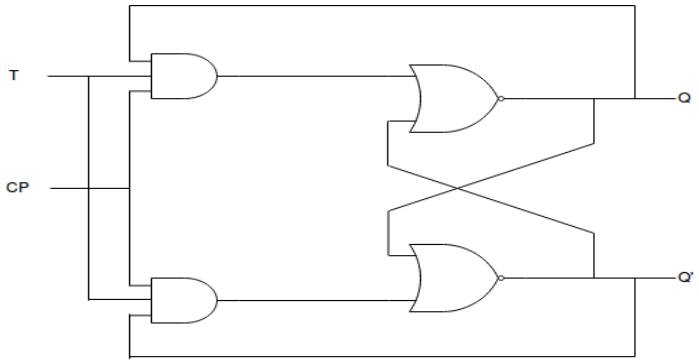
fig. Clocked JK flip flop

- The characteristic of inputs ‘J’ and ‘K’ is same as the 'S' and 'R' inputs of the S-R flip-flop.
- J stands for SET, and 'K' stands for CLEAR.
- When both the inputs J and K have a HIGH state, the flip-flop switches to the complement state, so, for a value of Q = 1, it switches to Q=0, and for a value of Q = 0, it switches to Q=1.

4. T Flip-Flop

T flip-flop is a much simpler version of the J-K flip-flop.

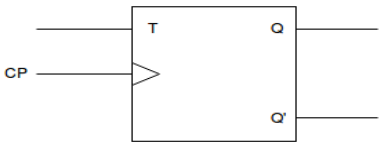
Toggle flip flop is basically a JK flip flop with J and K terminals permanently connected together. It has only input denoted by **T** as shown in the Symbol Diagram



a) Logic diagram

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

b) Transition table



c) Graphical symbol

fig. Clocked T flip flop

Both the J and K inputs are connected and are also called as a single input J-K Flip-flop