U N I T - 3

Combinational Logic Circuits: Introduction – Adders – The Half Adder – The Full Adder – Subtractors – BCD Adder – Multiplexers – Demultiplexers – Decoders – Encoders – Sequential Logic Circuits: Flip Flops – RS Flip Flop – Clocked RS Flip Flop – D Flip Flop – JK Flip Flop – T Flip Flop –Master Slave Flip Flop Registers: Counters – Asynchronous or Ripple Counter – Ring Counter – Shift Registers.

COMBINATIONAL LOGIC CIRCUITS

The combinational logic circuits are the circuits that contain different types of logic gates. Simply, a circuit in which different types of logic gates are combined is known as a **combinational logic circuit**.

The output of the combinational circuit is determined from the present combination of inputs, regardless of the previous input.

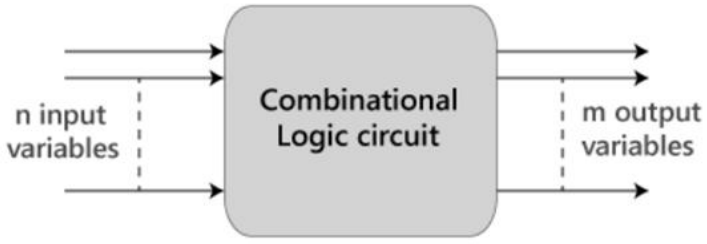
The input variables, logic gates, and output variables are the basic components of the combinational logic circuit.

There are different types of combinational logic circuits:

- Adder
- Subtractor
- Decoder
- Encoder
- Multiplexer
- De-multiplexer.

There are the following characteristics of the combinational logic circuit:

- At any instant of time, the output of the combinational circuits depends only on the present input terminals.
- The combinational circuit doesn't have any backup or previous memory. The present state of the circuit is not affected by the previous state of the input.
- The n number of inputs and m number of outputs are possible in combinational logic circuits.



Block diagram of a combinational circuit

- **Half Adder:** The half adder is used to add two 1-bit numbers. The half adder is a basic building block having two inputs and two outputs. The adder is used to perform OR operation of two single bit binary numbers. The **carry** and **sum** are two output states of the half adder.
- **Full Adder:** The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.
- **Half Subtractors** The half subtractor is also a building block of subtracting two binary numbers. It has two inputs and two outputs. This circuit is used to subtract two single bit binary numbers A and B. The '**difference**' and '**borrow**' are the two output state of the half adder.
- **Full Subtractors:** The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are **minuend, subtrahend**, and **borrow,** respectively. The full subtractor has three input states and two output states i.e., diff and borrow.
- **Multiplexers:** The multiplexer is a combinational circuit that has n-data inputs and a single output. It is also known as the **data selector** which selects one input from the inputs and routes it to the output.
- **De-multiplexers:** A De-multiplexer performs the reverse operation of a multiplexer. The de-multiplexer has only one input, which is distributed over several outputs.
- **Decoder:** A decoder is a combinational circuit having n inputs and to a maximum of $m = 2^n$ outputs.
- **Encoder:** The encoder is used to perform the reverse operation of the decoder. An encoder having 2^n number of inputs and n number of outputs.

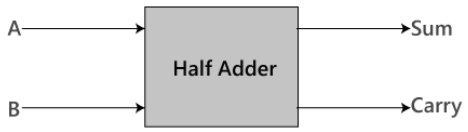
HALF ADDER

The Half-Adder is a basic building block of adding two 1-bit numbers as two inputs and produce out two outputs.

The adder is used to perform OR operation of two single bit binary numbers.

The **augent** and **addent** bits are two input states, and '**carry**' and '**sum** 'are two output states of the half adder.

Block diagram



Construction of Half Adder Circuit:

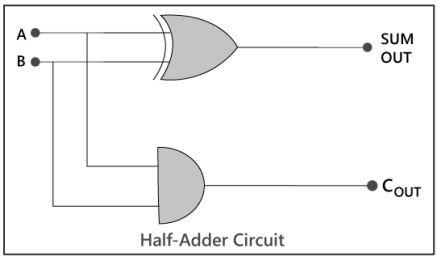
In the block diagram, we have seen that it contains two inputs and two outputs. The **augent** and **addent** bits are the input states, and **carry** and **sum** are the output states of the half adder.

The half adder is designed with the help of the following two logic gates:

1. 2-input AND Gate.
2. 2-input Exclusive-OR Gate or Ex-OR Gate

The Half Adder is designed by combining the 'XOR' and 'AND' gates and provide the sum and carry.

Logic Diagram



Truth Table

Inputs		Outputs	
A	B	Sum	Carry
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

In the above table,

1. 'A' and' B' are the input states, and 'sum' and 'carry' are the output states.
2. The carry output is 0 in case where both the inputs are not 1.
3. The least significant bit of the sum is defined by the 'sum' bit.

The Boolean expression of the sum and carry are as follows:

Sum = x'y+xy'

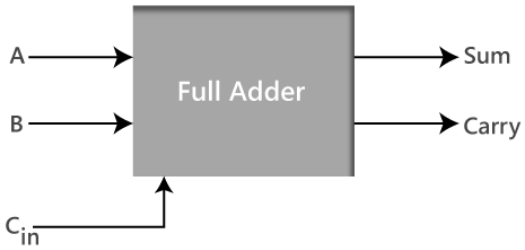
Carry = xy

FULL ADDER

The half adder is used to add only two numbers. To overcome this problem, the full adder was developed.

The full adder is used to add three 1-bit binary numbers A, B, and carry C. The full adder has three input states and two output states i.e., sum and carry.

Block diagram



Truth Table

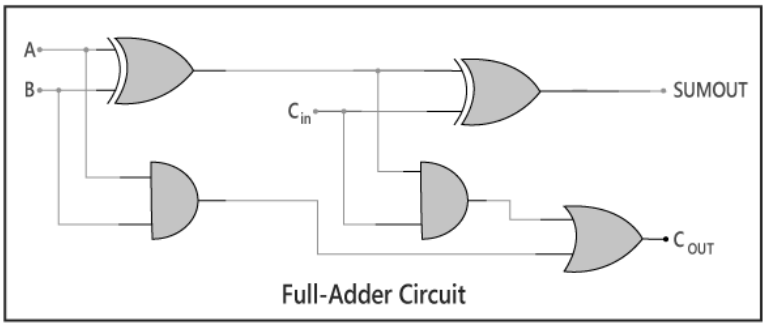
Inputs			Outputs	
A	B	C_{in}	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

In the above table,

1. 'A' and' B' are the input variables. These variables represent the two significant bits which are going to be added

2. 'C_{in}' is the third input which represents the carry. From the previous lower significant position, the carry bit is fetched.
3. The 'Sum' and 'Carry' are the output variables that define the output values.
4. The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

The full adder logic circuit can be constructed using the **'AND'** and **the 'XOR' gate** with an **OR gate**



Full-Adder Circuit

The actual logic circuit of the full adder is shown in the above diagram. The full adder circuit construction can also be represented in a Boolean expression.

Sum:

- Perform the XOR operation of input A and B.
- Perform the XOR operation of the outcome with carry. So, the sum is (A XOR B) XOR C_{in} which is also represented as: $(A \oplus B) \oplus C_{in}$
- Sum = A' B' C+A' BC+AB' C'+ABC

Carry:

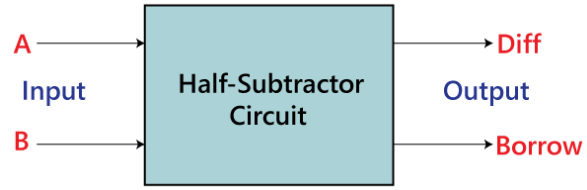
1. Perform the 'XOR' operation of input A and B.
2. Perform the XOR operation of the result with C_{in}.
3. Perform the 'AND' operation of input A and B.
4. Perform AND operation of $(A \oplus B)$ and C_{in}.
5. Perform the 'OR' operations of both the outputs that come from the previous two steps. So the 'Carry' can be represented as: $A.B + (A \oplus B) C_{in}$
6. Carry = AB+AC+BC

HALF SUBTRACTOR

The half subtractor is also a building block for subtracting two binary numbers. It has two inputs and two outputs.

This circuit is used to subtract two single bit binary numbers A and B. The **'diff'** and **'borrow'** are two output states of the half subtractor.

Block diagram



Truth Table

Inputs		Outputs	
A	B	Diff	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

The Boolean expression of the **Diff** and **Borrow** is as follows:

Diff= A'B+AB' (A⊕B)

Borrow = A'B

In the above table,

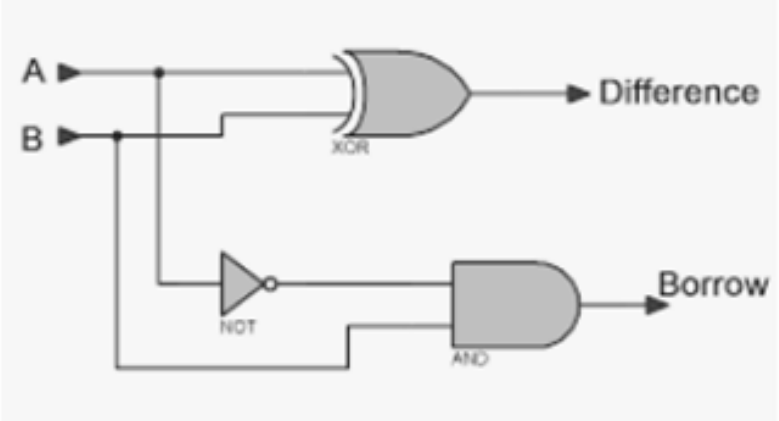
- 'A' and 'B' are the input variables whose values are going to be subtracted.
- The 'Diff' and 'Borrow' are the variables whose values define the subtraction result, i.e., difference and borrow.
- The first two rows and the last row, the difference is 1, but the 'Borrow' variable is 0.
- The third row is different from the remaining one. When we subtract the bit 1 from the bit 0, the borrow bit is produced.

Construction of Half Subtractor Circuit

In the block diagram, we have seen that it contains two inputs and two outputs. The **diff** and **borrow** are the output states of the half subtractor.

The half subtractor is designed with the help of the following logic gates:

1. 2-input AND gate.
2. 2-input Exclusive-OR Gate or Ex-OR Gate
3. NOT or inverter Gate



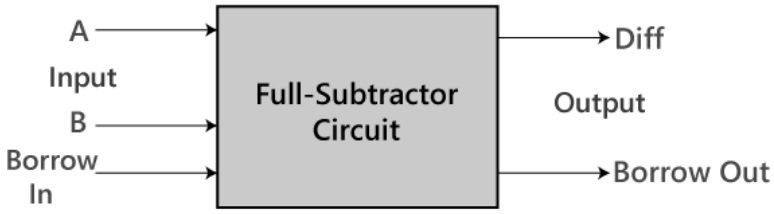
FULL SUBTRACTOR

The Half Subtractor is used to subtract only two numbers.

To overcome this problem, a full subtractor was designed. The full subtractor is used to subtract three 1-bit numbers A, B, and C, which are minuend, subtrahend, and borrow, respectively.

The full subtractor has three input states and two output states i.e., diff and borrow.

Block diagram



Truth Table

Inputs			Outputs	
A	B	$Borrow_{in}$	Diff	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

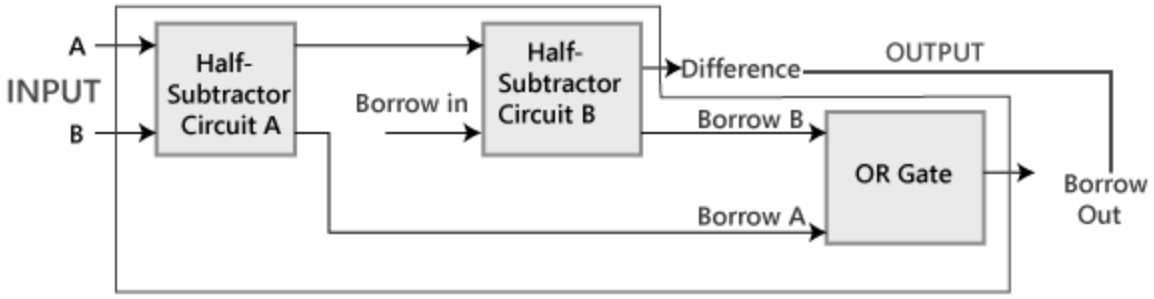
In the above table,

- 'A' and' B' are the input variables. These variables represent the two significant bits that are going to be subtracted.
- '$Borrow_{in}$' is the third input which represents borrow.
- The 'Diff' and 'Borrow' are the output variables that define the output values.
- The eight rows under the input variable designate all possible combinations of 0 and 1 that can occur in these variables.

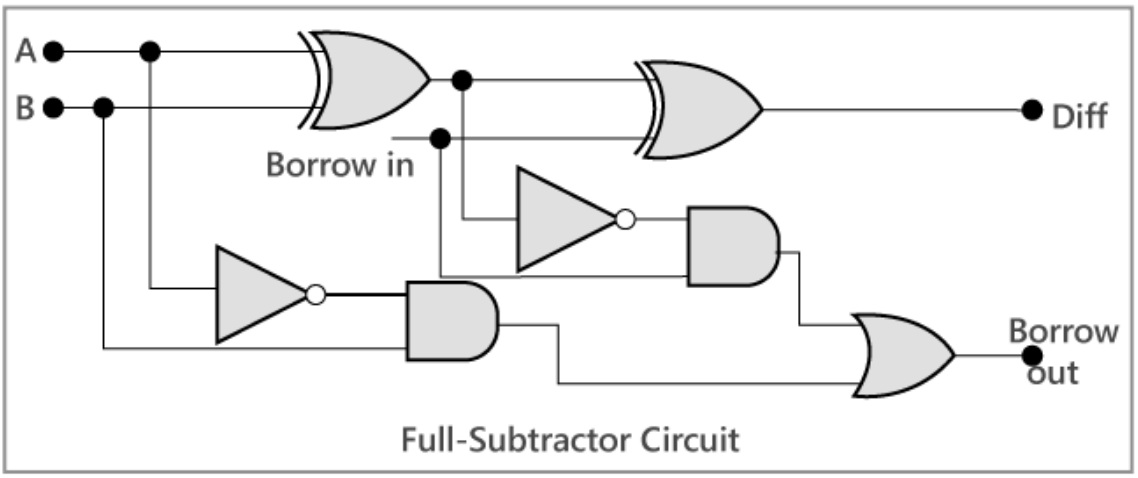
Diff=AB' C'+A' B' C+ABC+A'BC'

Borrow=A'C+A' B+BC

Construction of Full Subtractor Circuit:



The above block diagram describes the construction of the Full subtractor circuit. In the above circuit, there are two half adder circuits that are combined using the OR gate.



The actual logic circuit of the full subtractor is shown in the above diagram. The full subtractor circuit construction can also be represented in a Boolean expression.

Diff:

- Perform the XOR operation of input A and B.
- Perform the XOR operation of the outcome with 'Borrow'. So, the difference is (A XOR B) XOR '$Borrow_{in}$' which is also represented as: $(A \oplus B) \oplus 'Borrow_{in}'$

Borrow:

- Perform the 'AND' operation of the inverted input A and B.
- Perform the 'XOR' operation of input A and B.
- Perform the 'OR' operations of both the outputs that come from the previous two steps. So the 'Borrow' can be represented as: $A'.B + (A \oplus B)'$

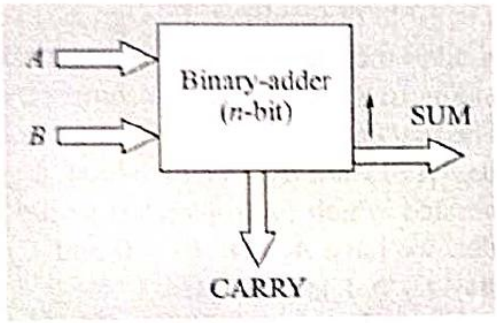
PARALLEL BINARY ADDER

The **Parallel binary adder** is a **combinational circuit** consists of various full adders in parallel structure so that when more than 1-bit numbers are to be added, then there can be full adder for every column for the addition.

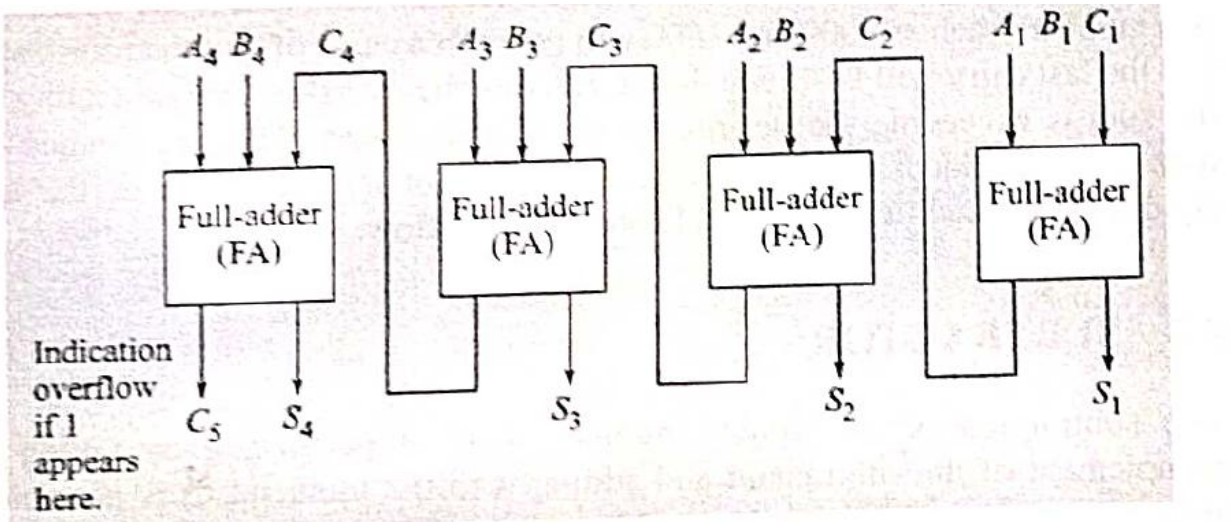
The number of full adders in a parallel binary adder depends on the number of bits present in the number for the addition.

If 4-bits numbers are to be added, then there will be 4-full adder in the parallel binary adder.

Block Diagram:



The circuit diagram of binary parallel adder is shown below:



C	1	1	1	0	
A	0	1	1	1	
B	0	1	0	1	+
	1	1	0	0	

BCD Adder

BCD adder refers to a 4-bit binary adder that can add two 4-bit words of BCD format. The output of the addition is a BCD-format 4-bit output word, which defines the decimal sum of the addend and augend and a carry that is created in case this sum exceeds a decimal value of 9. Therefore, BCD adders can implement decimal addition.

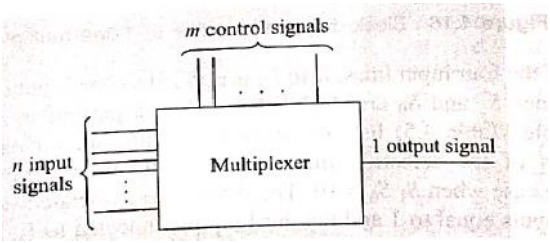
MULTIPLEXER

A multiplexer is a combinational circuit that has 2^n input lines and a single output line.

Simply, the multiplexer is a multi-input and single-output combinational circuit. The binary information is received from the input lines and directed to the output line.

On the basis of the values of the selection lines, one of these data inputs will be connected to the output.

Block Diagram:



There is a total of 2^N possible combinations of inputs. A multiplexer is also treated as **Mux**.

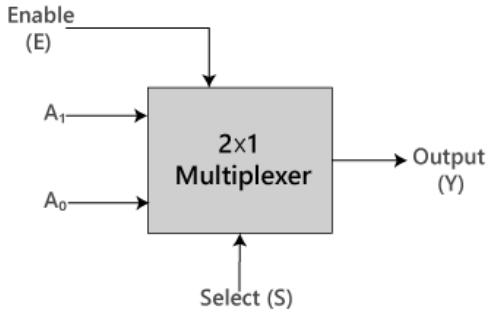
2×1 Multiplexer:

In 2×1 multiplexer, there are only two inputs, i.e., A_0 and A_1, 1 selection line, i.e., S_0 and single outputs, i.e., Y.

On the basis of the combination of inputs which are present at the selection line S^0, one of these 2 inputs will be connected to the output.

The block diagram and the truth table of the 2×1 multiplexer are given below.

Block Diagram



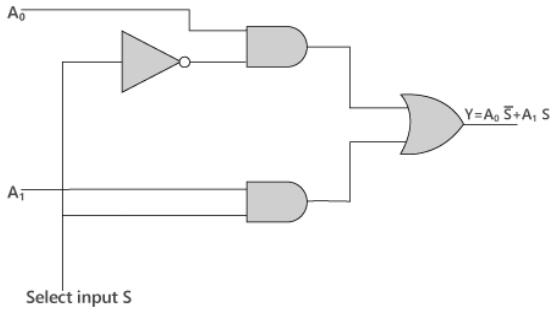
Truth Table

INPUTS	Output
S_0	Y
0	A_0
1	A_1

The logical expression of the term Y is as follows:

$Y=S_0'.A_0+S_0.A_1$

Logical circuit of the above expression is given below:



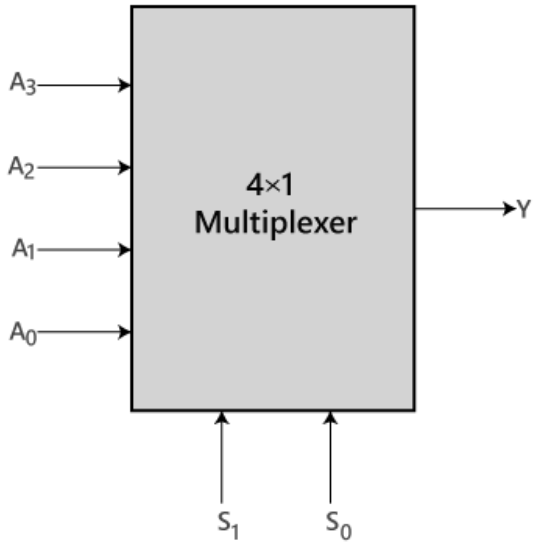
4×1 Multiplexer:

In the 4×1 multiplexer, there is a total of four inputs, i.e., A_0, A_1, A_2, and A_3, 2 selection lines, i.e., S_0 and S_1 and single output, i.e., Y.

On the basis of the combination of inputs that are present at the selection lines S^0 and S_1, one of these 4 inputs are connected to the output.

The block diagram and the truth table of the 4×1 multiplexer are given below.

Block Diagram



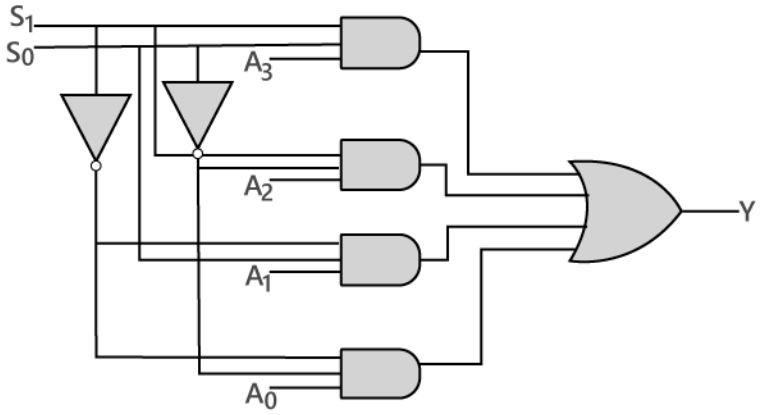
Truth Table

INPUTS		Output
S_1	S_0	Y
0	0	A_0
0	1	A_1
1	0	A_2
1	1	A_3

The logical expression of the term Y is as follows:

$Y=S_1' S_0' A_0+S_1' S_0 A_1+S_1 S_0' A_2+S_1 S_0 A_3$

Logical circuit of the above expression is given below:



DE-MULTIPLEXER

A De-multiplexer is a combinational circuit that has only 1 input line and 2^N output lines.

Simply, the multiplexer is a single-input and multi-output combinational circuit. The information is received from the single input lines and directed to the output line.

On the basis of the values of the selection lines, the input will be connected to one of these outputs. De-multiplexer is opposite to the multiplexer.

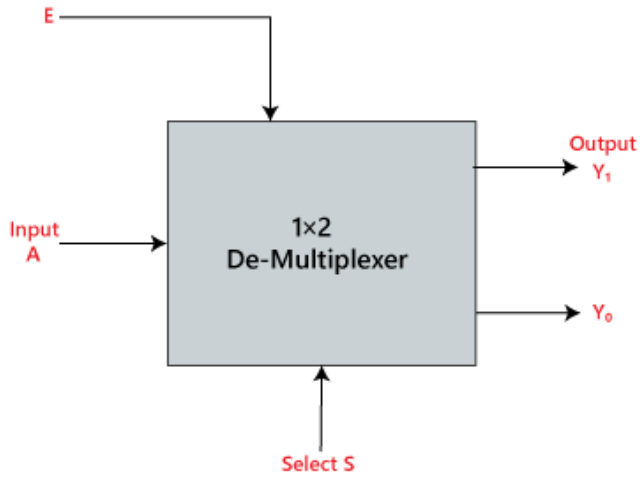
Unlike encoder and decoder, there are n selection lines and 2^n outputs. So, there is a total of 2^n possible combinations of inputs. De-multiplexer is also treated as **De-mux**.

1×2 De-multiplexer:

In the 1 to 2 De-multiplexer, there are only two outputs, i.e., Y_0, and Y_1, 1 selection lines, i.e., S_0, and single input, i.e., A.

On the basis of the selection value, the input will be connected to one of the outputs. The block diagram and the truth table of the 1×2 multiplexer are given below.

Block Diagram



Truth Table

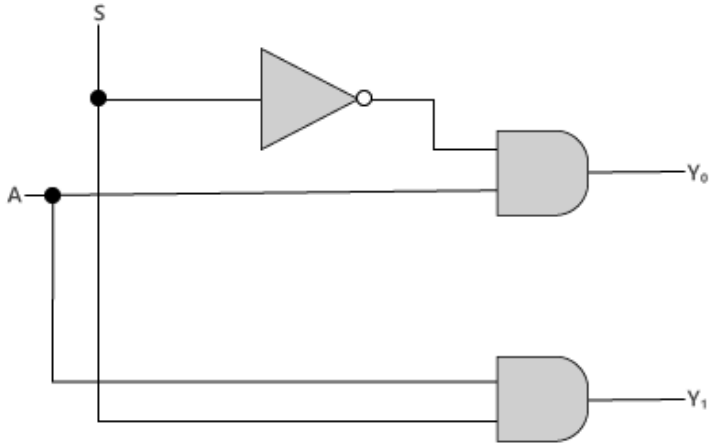
INPUTS	Output	
S_0	Y_1	Y_0
0	0	A
1	A	0

The logical expression of the term Y is as follows:

$Y_0=S_0'.A$

$Y_1=S_0.A$

Logical circuit of the above expressions is given below:

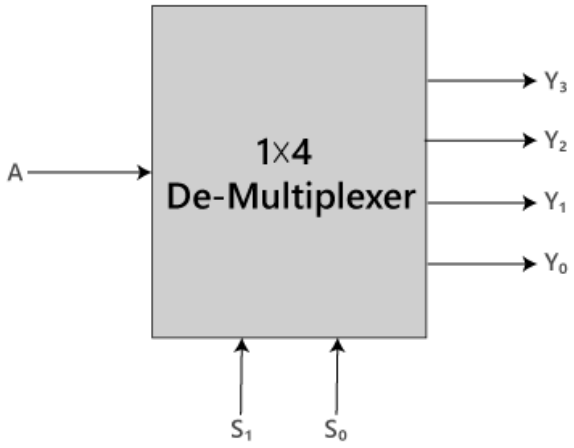


1×4 De-multiplexer:

In 1 to 4 De-multiplexer, there are total of four outputs, i.e., Y_0, Y_1, Y_2, and Y_3, 2 selection lines, i.e., S_0 and S_1 and single input, i.e., A.

On the basis of the combination of inputs which are present at the selection lines S_0 and S_1, the input be connected to one of the outputs. The block diagram and the truth table of the 1×4 multiplexer are given below.

Block Diagram



Truth Table

INPUTS		Output			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

The logical expression of the term Y is as follows:

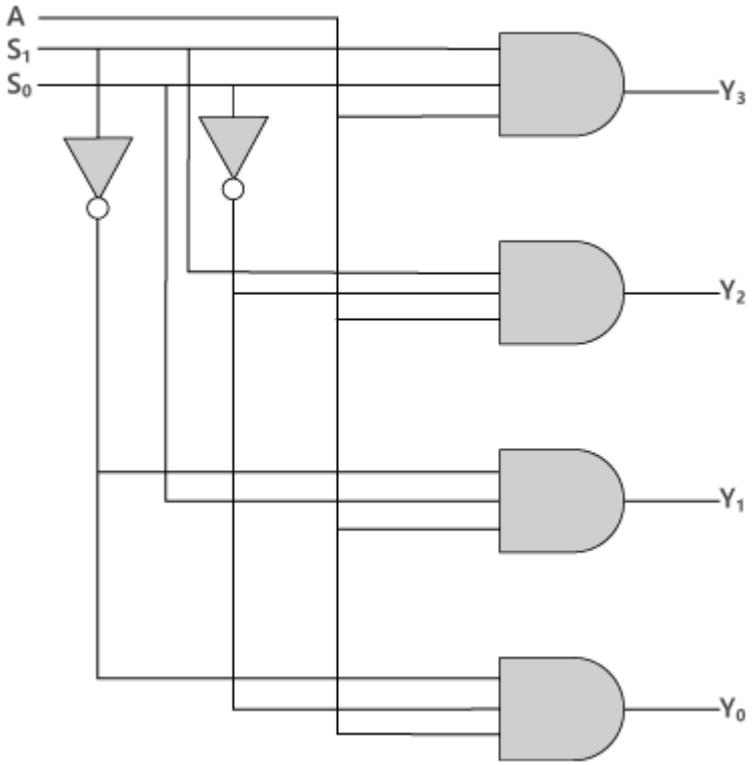
$Y_0=S_1' S_0' A$

$y_1=S_1' S_0 A$

$y_2=S_1 S_0' A$

$y_3=S_1 S_0 A$

Logical circuit of the above expressions is given below:

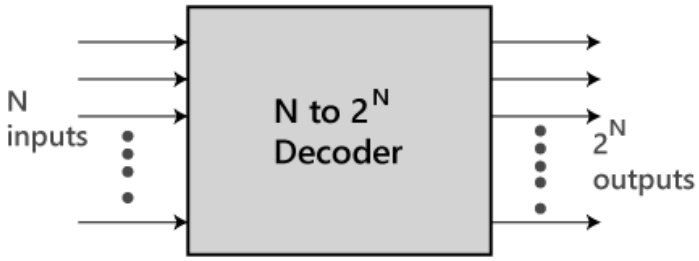


DECODER

The combinational circuit that change the binary information into 2^N output lines is known as **Decoders.**

The binary information is passed in the form of N input lines. The output lines define the 2^N-bit code for the binary information.

The produced 2^N-bit output code is equivalent to the binary information.



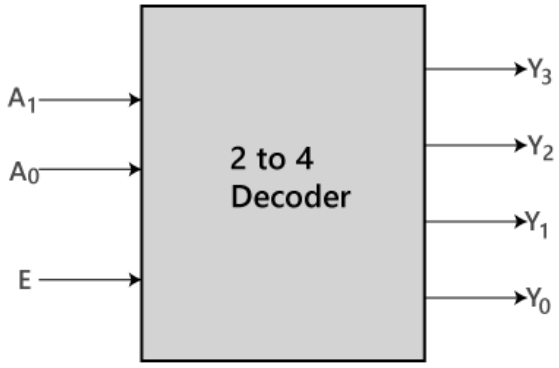
2 to 4 line decoder:

In the 2 to 4 line decoder, there is a total of three inputs, i.e., A_0, and A_1 and E and four outputs, i.e., Y_0, Y_1, Y_2, and Y_3.

For each combination of inputs, when the enable 'E' is set to 1, one of these four outputs will be 1.

The block diagram and the truth table of the 2 to 4 line decoder are given below.

Block Diagram:



Truth Table:

Enable	INPUTS		OUTPUTS			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	X	X	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

The logical expression of the term Y0, Y0, Y2, and Y3 is as follows:

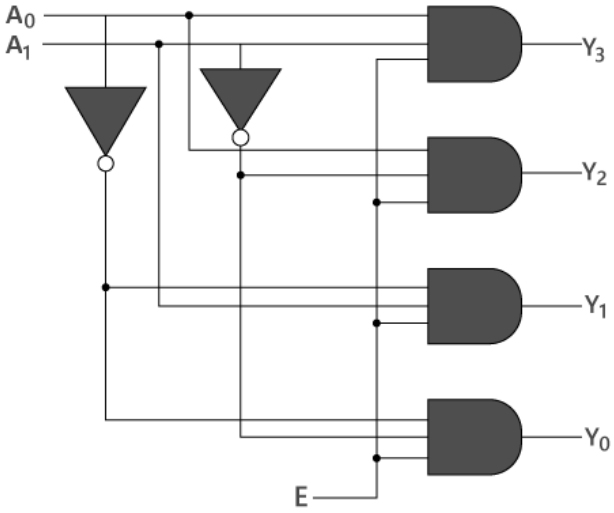
$Y_3=E.A_1.A_0$

$Y_2=E.A_1.A_0'$

$Y_1=E.A_1'.A_0$

$Y0=E.A_1'.A_0'$

Logical circuit of the above expressions is given below:



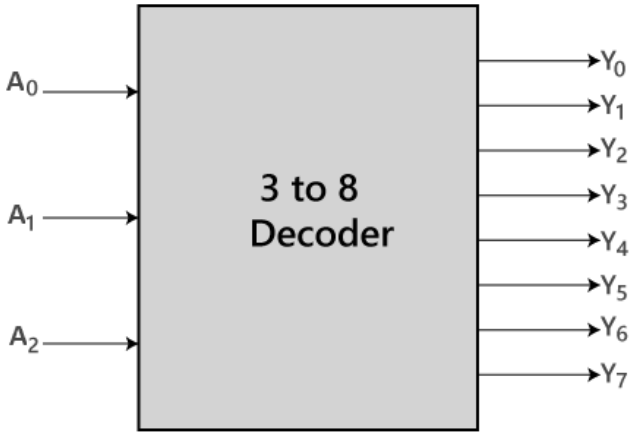
3 to 8 line decoder:

The 3 to 8 line decoder is also known as **Binary to Octal Decoder**. In a 3 to 8 line decoder, there is a total of eight outputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 and three outputs, i.e., A_0, A1, and A_2.

This circuit has an enable input 'E'. Just like 2 to 4 line decoder, when enable 'E' is set to 1, one of these four outputs will be 1.

The block diagram and the truth table of the 3 to 8 line encoder are given below.

Block Diagram:



Truth Table:

Enable	INPUTS			Outputs							
E	A_2	A_1	A_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

The logical expression of the term Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 is as follows:

$Y_0=A_0'.A_1'.A_2'$

$Y_1=A_0.A_1'.A_2'$

$Y_2=A_0'.A_1.A_2'$

$Y_3=A_0.A_1.A_2'$

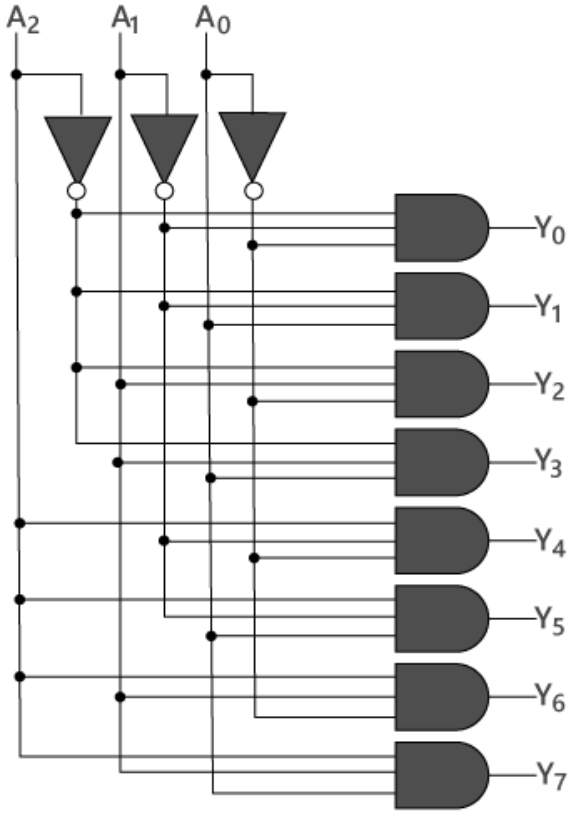
$Y_4=A_0'.A_1'.A_2$

$Y_5=A_0.A_1'.A_2$

$Y_6=A_0'.A_1.A_2$

$Y_7=A_0.A_1.A_2$

Logical circuit of the above expressions is given below:

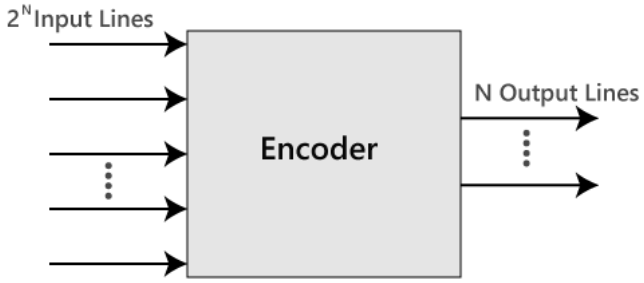


ENCODERS

The combinational circuits that change the binary information into N output lines are known as **Encoders**.

The binary information is passed in the form of 2^N input lines. The output lines define the N-bit code for the binary information.

At a time, only one input line is activated for simplicity. The produced N-bit output code is equivalent to the binary information.



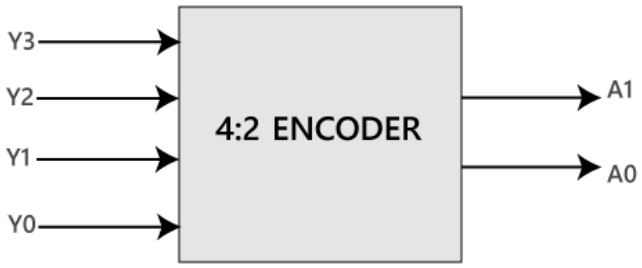
4 to 2 line Encoder:

In 4 to 2 line encoder, there are total of four inputs, i.e., Y_0, Y_1, Y_2, and Y_3, and two outputs, i.e., A_0 and A_1.

In 4-input lines, one input-line is set to true at a time to get the respective binary code in the output side.

Below are the block diagram and the truth table of the 4 to 2 line encoder.

Block Diagram:



Truth Table:

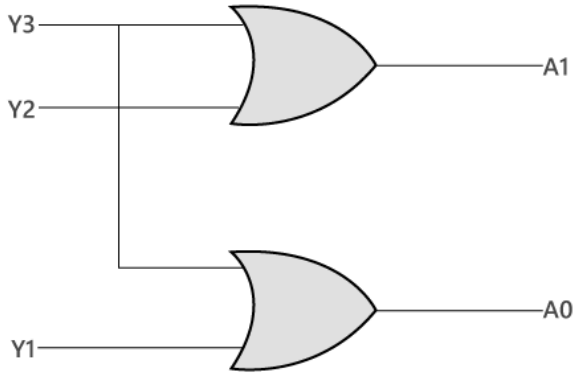
INPUTS				OUTPUTS	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
1	0	0	0	0	0
0	1	0	0	0	1
0	0	1	0	1	0
0	0	0	1	1	1

The logical expression of the term A0 and A1 is as follows:

$A_1=Y_3+Y_2$

$A_0=Y_3+Y_1$

Logical circuit of the above expressions is given below:



8 to 3 line Encoder:

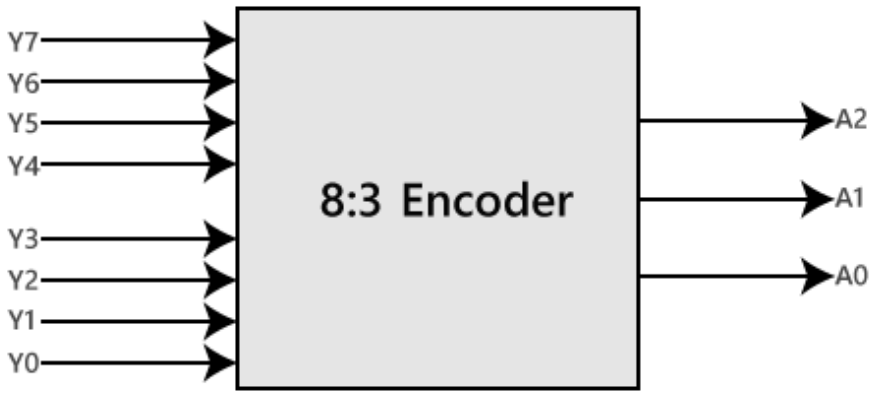
The 8 to 3 line Encoder is also known as **Octal to Binary Encoder**.

In 8 to 3 line encoder, there is a total of eight inputs, i.e., Y_0, Y_1, Y_2, Y_3, Y_4, Y_5, Y_6, and Y_7 and three outputs, i.e., A_0, A1, and A_2.

In 8-input lines, one input-line is set to true at a time to get the respective binary code in the output side.

Below are the block diagram and the truth table of the 8 to 3 line encoder.

Block Diagram:



Truth Table:

INPUTS								OUTPUTS		
Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0	A_2	A_1	A_0
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

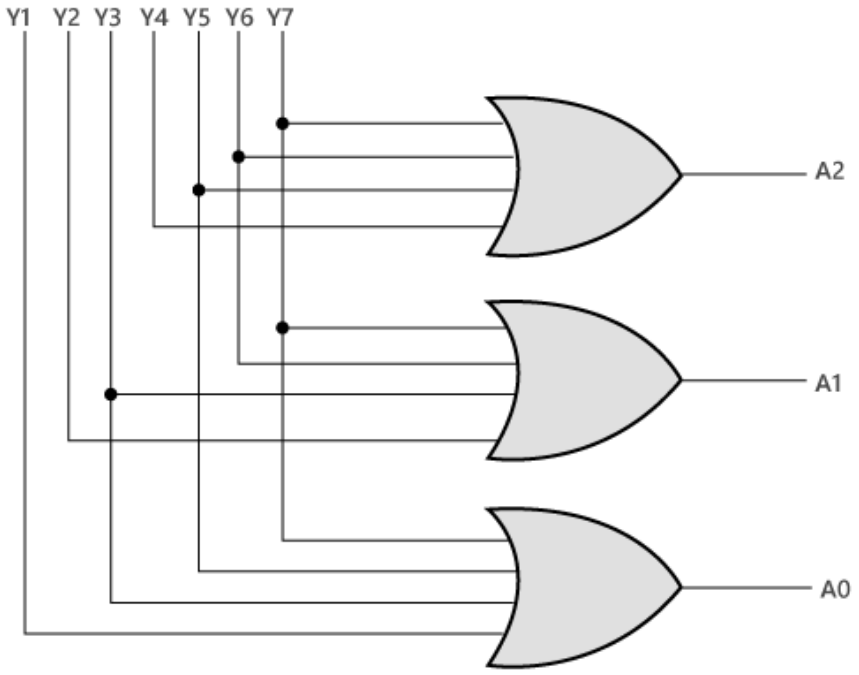
The logical expression of the term A0, A1, and A2 are as follows:

$A_2=Y_4+Y_5+Y_6+Y_7$

$A_1=Y_2+Y_3+Y_6+Y_7$

$A_0=Y_7+Y_5+Y_3+Y_1$

Logical circuit of the above expressions is given below:



SEQUENTIAL LOGIC CIRCUITS

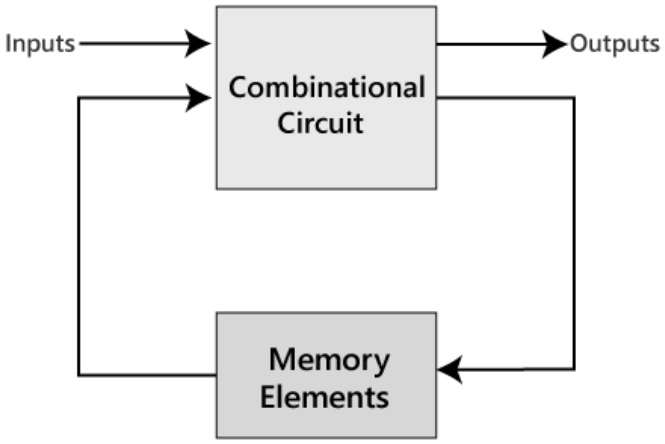
The combinational circuits have set of outputs, which depends only on the present combination of inputs.

The sequential circuit is a special type of circuit that has a series of inputs and outputs. The outputs of the sequential circuits depend on both the combination of present inputs and previous outputs.

The previous output is treated as the present state. So, the sequential circuit contains the combinational circuit and its memory storage elements.

A sequential circuit doesn't need to always contain a combinational circuit. So, the sequential circuit can contain only the memory element.

Below is the block diagram of the synchronous logic circuit.



Difference between the combinational circuits and sequential circuits are given below:

	Combinational Circuits	**Sequential Circuits**
1)	The outputs of the combinational circuit depend only on the present inputs.	The outputs of the sequential circuits depend on both present inputs and present state(previous output).
2)	The feedback path is not present in the combinational circuit.	The feedback path is present in the sequential circuits.
3)	In combinational circuits, memory elements are not required.	In the sequential circuit, memory elements play an important role and require.
4)	The clock signal is not required for combinational circuits.	The clock signal is required for sequential circuits.
5)	The combinational circuit is simple to design.	It is not simple to design a sequential circuit.

Types of Sequential Circuits

- **Asynchronous sequential circuits:** The clock signals are not used by the **Asynchronous sequential circuits**. The asynchronous circuit is operated through the pulses. So, the changes in the input can change the state of the circuit. The asynchronous circuits do not use clock pulses. The internal state is changed when the input variable is changed. The un-clocked flip-flops or time-delayed are the memory elements of asynchronous sequential circuits. The asynchronous sequential circuit is similar to the combinational circuits with feedback.
- **Synchronous sequential circuits:** In synchronous sequential circuits, synchronization of the memory element's state is done by the clock signal. The output is stored in either flip-flops or latches(memory devices). The synchronization of the outputs is done with either only negative edges of the clock signal or only positive edges.

FLIP FLOP

A circuit that has two stable states is treated as a **flip flop**. These stable states are used to store binary data that can be changed by applying varying inputs.

The flip flops are the fundamental building blocks of the digital system. Flip flops and latches are examples of data storage elements.

In the sequential logical circuit, the flip flop is the basic storage element.

SR FLIP FLOP

The SR flip flop is a 1-bit memory bistable device having two inputs, i.e., SET and RESET.

The SET input 'S' set the device or produce the output 1, and the RESET input 'R' reset the device or produce the output 0.

The SET and RESET inputs are labeled as **S** and **R**, respectively.

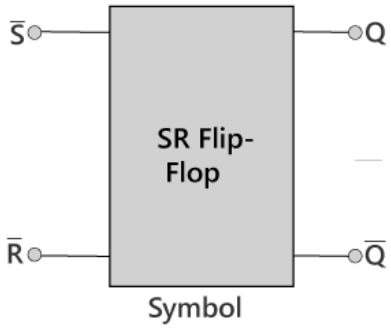
The SR flip flop stands for "Set-Reset" flip flop.

The reset input is used to get back the flip flop to its original state from the current state with an output 'Q'. This output depends on the set and reset conditions, which is either at the logic level "0" or "1".

The NAND gate SR flip flop is a basic flip flop which provides feedback from both of its outputs back to its opposing input.

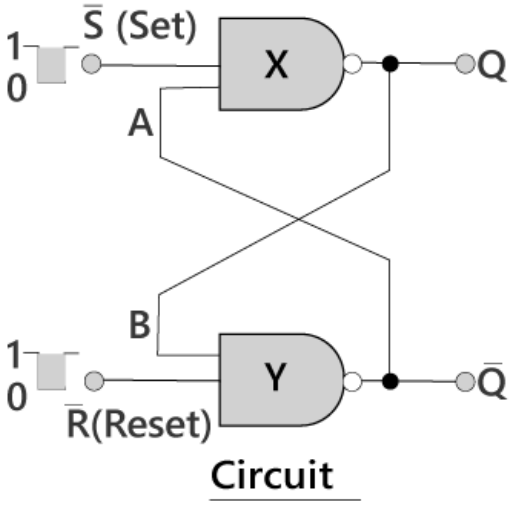
This circuit is used to store the single data bit in the memory circuit. So, the SR flip flop has a total of three inputs, i.e., 'S' and 'R', and current output 'Q'.

This output 'Q' is related to the current history or state. The term "flip-flop" relates to the actual operation of the device, as it can be "flipped" to a logic set state or "flopped" back to the opposing logic reset state.



Symbol

Circuit Diagram:



Circuit

Truth Table:

S	R	Q(t+1)
0	0	Q(t) **(No Change)**
0	1	0
1	0	1
1	1	Indeterminate

Characteristic Equation

Q(t+1) = R'Q + S

SR = 0

D Flip Flop

In **SR NAND Gate Bistable** circuit, the undefined input condition of SET = "0" and RESET = "0" is forbidden. It is the drawback of the SR flip flop. This state:

1. Override the feedback latching action.
2. Force both outputs to be 1.

3. Lose the control by the input, which first goes to 1, and the other input remains "0" by which the resulting state of the latch is controlled.

We need an **inverter** to prevent this from happening. We connect the inverter between the Set and Reset inputs for producing another type of flip flop circuit called **D flip flop**,

Delay flip flop, D-type Bistable, D-type flip flop.

The D flip flop is the most important flip flop from other clocked types. It ensures that at the same time, both the inputs, i.e., S and R, are never equal to 1.

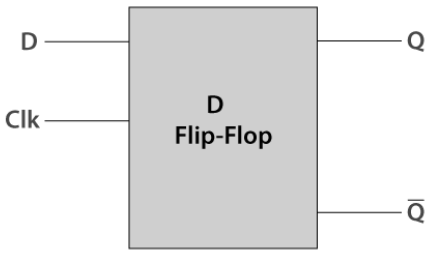
The Delay flip-flop is designed using a gated SR flip-flop with an inverter connected between the inputs allowing for a single input D(Data).

This single data input, which is labeled as "D" used in place of the "Set" input and for the complementary "Reset" input, the inverter is used.

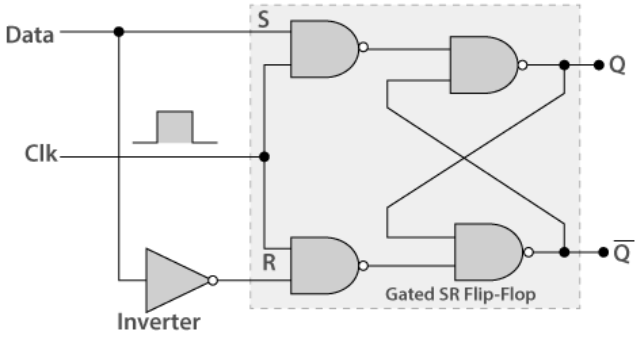
Thus, the level-sensitive D-type or D flip flop is constructed from a level-sensitive SR flip flop.

So, here S=D and R= ~D(complement of D)

Block Diagram



Circuit Diagram



Truth Table

D	Q(t+1)
0	0
1	1

Characteristic Equation: Q(t+1) = D(t)

JK Flip Flop

The SR Flip Flop or Set-Reset flip flop has lots of advantages. But, it has the following switching problems:

- When Set 'S' and Reset 'R' inputs are set to 0, this condition is always avoided.
- When the Set or Reset input changes their state while the enable input is 1, the incorrect latching action occurs.

The JK Flip Flop removes these two drawbacks of SR Flip Flop.

The JK flip flop is one of the most used flip flops in digital circuits. The JK flip flop is a universal flip flop having two inputs 'J' and 'K'. In SR flip flop, the 'S' and 'R' are the shortened abbreviated letters for Set and Reset, but J and K are not. The J and K are themselves autonomous letters which are chosen to distinguish the flip flop design from other types.

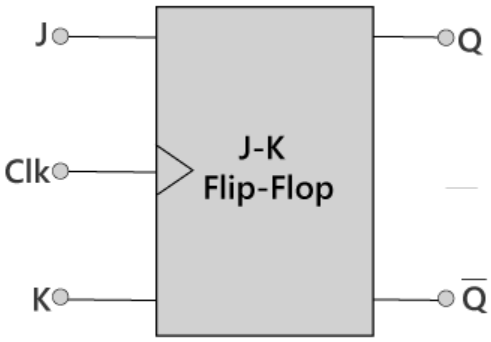
The JK flip flop work in the same way as the SR flip flop work. The JK flip flop has 'J' and 'K' flip flop instead of 'S' and 'R'.

The only difference between JK flip flop and SR flip flop is that when both inputs of SR flip flop is set to 1, the circuit produces the invalid states as outputs, but in case of JK flip flop, there are no invalid states even if both 'J' and 'K' flip flops are set to 1.

The JK Flip Flop is a gated SR flip-flop having the addition of a clock input circuitry. The invalid or illegal output condition occurs when both of the inputs are set to 1 and are prevented by the addition of a clock input circuit.

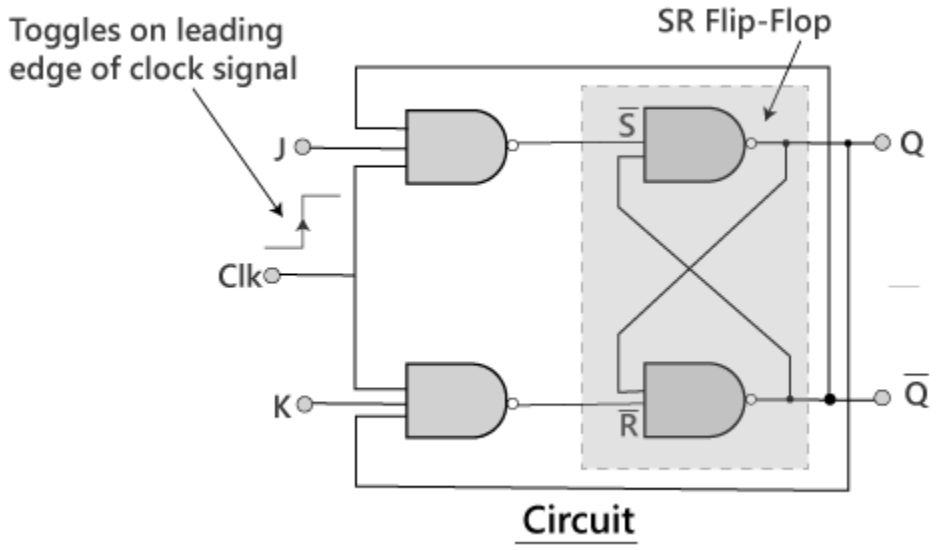
So, the JK flip-flop has four possible input combinations, i.e., 1, 0, "no change" and "toggle". The symbol of JK flip flop is the same as **SR Bistable Latch** except for the addition of a clock input.

Block Diagram:



Symbol

Circuit Diagram:



Truth Table

J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	Q'(t)

Characteristic Equation

$$Q(t+1) = K'(t)Q(t) + J(t)Q'(t)$$

T Flip Flop

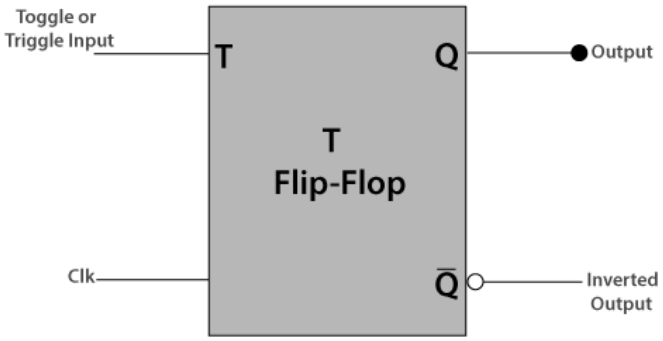
In T flip flop, "T" defines the term "Toggle". In SR Flip Flop, we provide only a single input called "Toggle" or "Trigger" input to avoid an intermediate state occurrence.

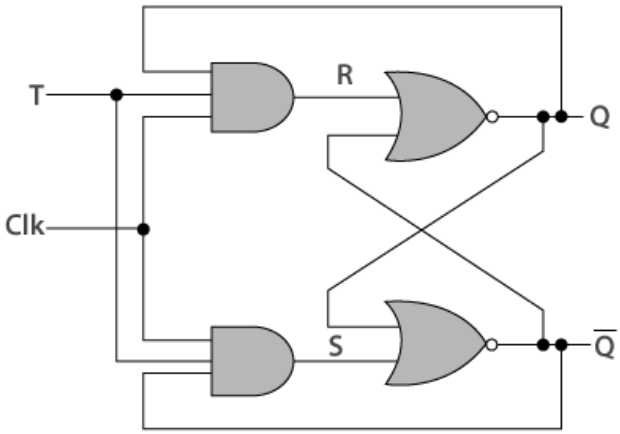
Now, this flip-flop work as a Toggle switch. The next output state is changed with the complement of the present state output. This process is known as "Toggling"'.

We can construct the "T Flip Flop" by making changes in the "JK Flip Flop". The "T Flip Flop" has only one input, which is constructed by connecting the input of JK flip flop.

This single input is called T. In simple words, we can construct the "T Flip Flop" by converting a "JK Flip Flop". Sometimes the "T Flip Flop" is referred to as single input "JK Flip Flop".

Block diagram of the "T-Flip Flop" is given where T defines the "Toggle input", and CLK defines the clock signal input.





Truth Table

T	Q(t+1)
0	Q(t)
1	Q'(t)

Characteristic Equation

$$Q(t+1) = T'(t)Q(t) + T(t)Q'(t) = T(t) \oplus Q(t)$$

Master-Slave JK Flip Flop

In "JK Flip Flop", when both the inputs and CLK set to 1 for a long time, then Q output toggle until the CLK is 1.

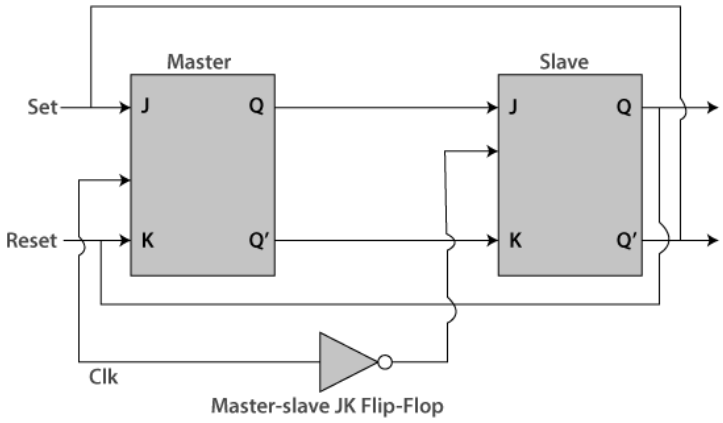
Thus, the uncertain or unreliable output produces. This problem is referred to as a race-round condition in JK flip-flop and avoided by ensuring that the CLK set to 1 only for a very short time.

The master-slave flip flop is constructed by combining two JK flip flops. These flip flops are connected in a series configuration. In these two flip flops, the 1st flip flop work as "master", called the master flip flop, and the 2nd work as a "slave", called slave flip flop.

The master-slave flip flop is designed in such a way that the output of the "master" flip flop is passed to both the inputs of the "slave" flip flop. The output of the "slave" flip flop is passed to inputs of the master flip flop.

In "master-slave flip flop", apart from these two flip flops, an inverter or NOT gate is also used. For passing the inverted clock pulse to the "slave" flip flop, the inverter is connected to the clock's pulse.

In simple words, when CP set to false for "master", then CP is set to true for "slave", and when CP set to true for "master", then CP is set to false for "slave".



Master-slave JK Flip-Flop

Working:

- When the clock pulse is true, the slave flip flop will be in the isolated state, and the system's state may be affected by the J and K inputs. The "slave" remains isolated until the CP is 1. When the CP set to 0, the master flip-flop passes the information to the slave flip flop to obtain the output.
- The master flip flop responds first from the slave because the master flip flop is the positive level trigger, and the slave flip flop is the negative level trigger.
- The output Q'=1 of the master flip flop is passed to the slave flip flop as an input K when the input J set to 0 and K set to 1. The clock forces the slave flip flop to work as reset, and then the slave copies the master flip flop.
- When J=1, and K=0, the output Q=1 is passed to the J input of the slave. The clock's negative transition sets the slave and copies the master.
- The master flip flop toggles on the clock's positive transition when the inputs J and K set to 1. At that time, the slave flip flop toggles on the clock's negative transition.
- The flip flop will be disabled, and Q remains unchanged when both the inputs of the JK flip flop set to 0.

Timing Diagram of a Master Flip Flop:

REGISTERS

A **Register** is a collection of flip flops.

A flip flop is used to store single bit digital data. For storing a large number of bits, the storage capacity is increased by grouping more than one flip flops. If we want to store an n-bit word, we have to use an n-bit register containing n number of flip flops.

The register is used to perform different types of operations. For performing the operations, the CPU use these registers. The faded inputs to the system will store into the registers. The result returned by the system will store in the registers.

Shift Register

A group of flip flops which is used to store multiple bits of data and the data is moved from one flip flop to another is known as **Shift Register**.

The bits stored in registers shifted when the clock pulse is applied within and inside or outside the registers. To form an n-bit shift register, we have to connect n number of flip flops.

So, the number of bits of the binary number is directly proportional to the number of flip flops. The flip flops are connected in such a way that the first flip flop's output becomes the input of the other flip flop.

A **Shift Register** can shift the bits either to the left or to the right. A **Shift Register**, which shifts the bit to the left, is known as **"Shift left register"**, and it shifts the bit to the right, known as **"Right left register"**.

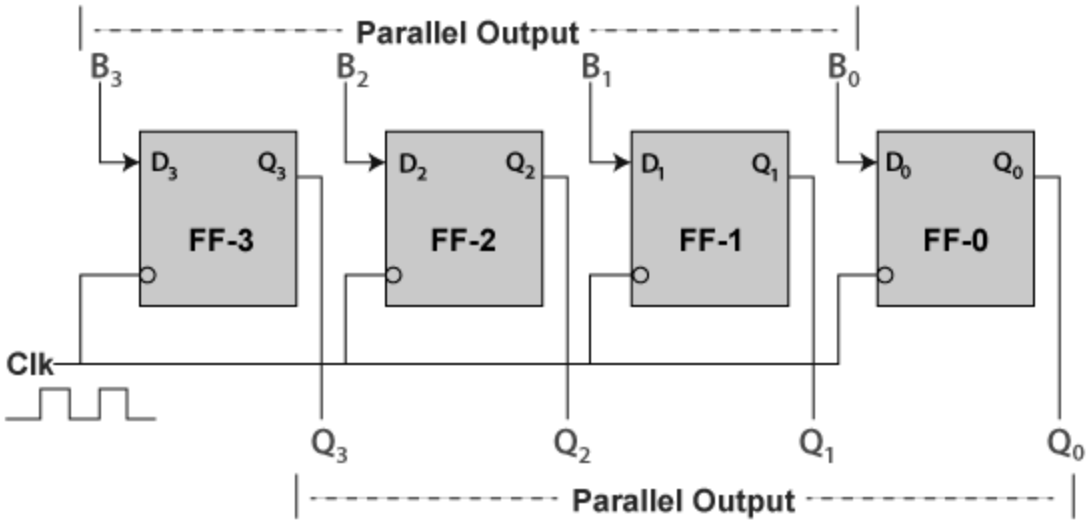
Parallel IN Parallel OUT

In **"Parallel IN Parallel OUT"**, the inputs and the outputs come in a parallel way in the register.

The inputs A_0, A_1, A_2, and A_3, are directly passed to the data inputs D_0, D_1, D_2, and D_3 of the respective flip flop. The bits of the binary input is loaded to the flip flops when the negative clock edge is applied.

The clock pulse is required for loading all the bits. At the output side, the loaded bits appear.

Block Diagram



Bidirectional Shift Register

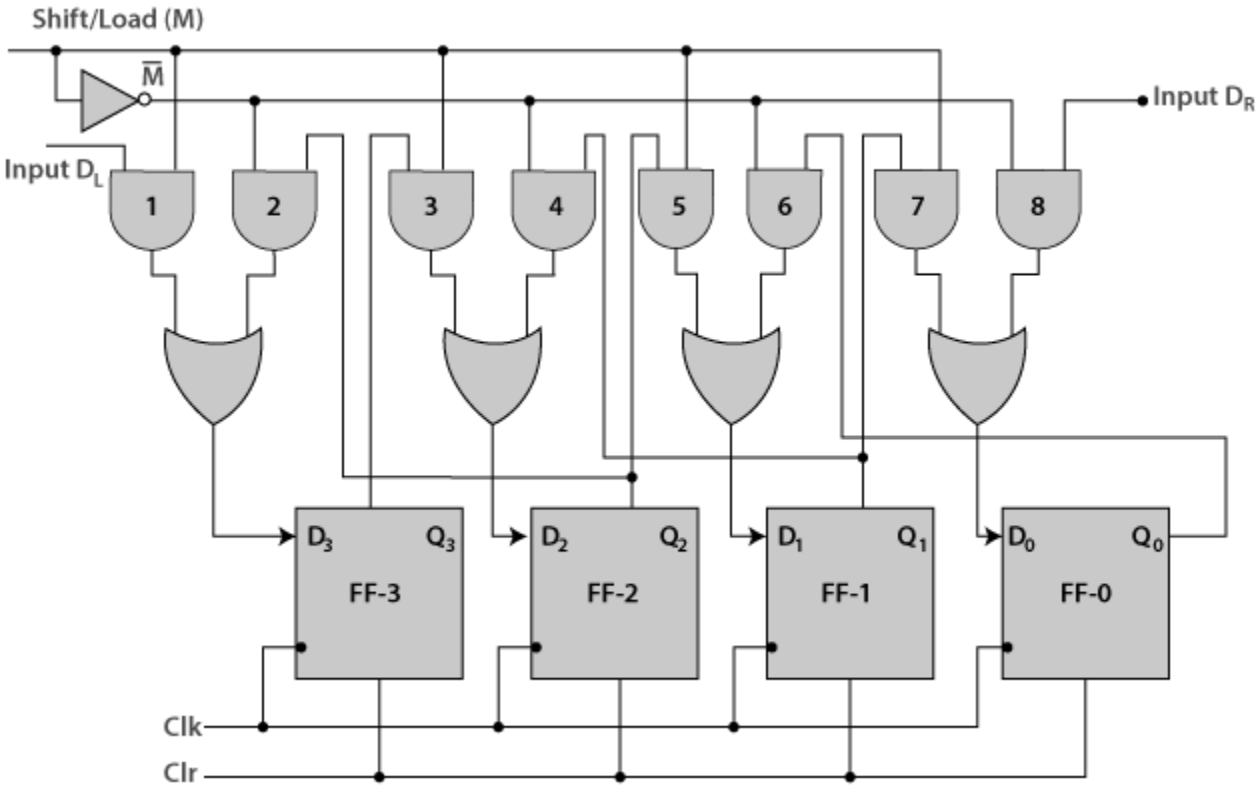
The binary number after shifting each bit of the number to the left by one position will be equivalent to the number produced by multiplying the original number by 2.

In the same way, the binary number after shifting each bit of the number to the right by one position will be equivalent to the number produced by dividing the original number by 2.

For performing the multiplication and division operation using the shift register, it is required that the data should be moved in both the direction, i.e., left or right in the register. Such registers are called the **"Bidirectional"** shift register.

Below is the diagram of 4-bit **"bidirectional"** shift register where $\mathbf{D_R}$ is the **"serial right shift data input"**, $\mathbf{D_L}$ is the **"left shift data input"**, and M is the **"mode select input"**.

Block Diagram



Operations

1) Shift right operation(M=1)

- The first, third, fifth, and seventh AND gates will be enabled, but the second, fourth, sixth, and eighth AND gates will be disabled.
- The data present on the data input **DR** is shifted bit by bit from the fourth flip flop to the first flip flop when the clock pulse is applied. In this way, the shift right operation occurs.

2) Shift left operation(M=0)

- The second, fourth, sixth and eighth AND gates will be enabled, but the AND gates first, third, fifth, and seventh will be disabled.
- The data present on the data input DR is shifted bit by bit from the first flip flop to the fourth flip flop when the clock pulse is applied. In this way, the shift right operation occurs.

Counters

A special type of sequential circuit used to count the pulse is known as a counter, or a collection of flip flops where the clock signal is applied is known as counters.

The counter is one of the widest applications of the flip flop. Based on the clock pulse, the output of the counter contains a predefined state. The number of the pulse can be counted using the output of the counter.

Truth Table

Clock	Counter output		State number	Decimal counter output
	Q_B	Q_A		
Initially	0	0	-	0
1st	0	1	1	1
2nd	1	0	2	2
3rd	1	1	3	3
4th	0	0	4	0

There are the following types of counters:

- Asynchronous Counters
- Synchronous Counters

RIPPLE COUNTER

Ripple counter is a special type of **Asynchronous** counter in which the clock pulse ripples through the circuit.

The n-MOD ripple counter forms by combining n number of flip-flops. The n-MOD ripple counter can count 2n states, and then the counter resets to its initial value.

Features of the Ripple Counter:

- Different types of flip flops with different clock pulse are used.
- It is an example of an asynchronous counter.
- The flip flops are used in toggle mode.
- The external clock pulse is applied to only one flip flop. The output of this flip flop is treated as a clock pulse for the next flip flop.
- In counting sequence, the flip flop in which external clock pulse is passed, act as LSB.

Based on their circuitry design, the counters are classified into the following types:

Up Counter

The up-counter counts the states in ascending order.

Down Counter

The down counter counts the states in descending order.

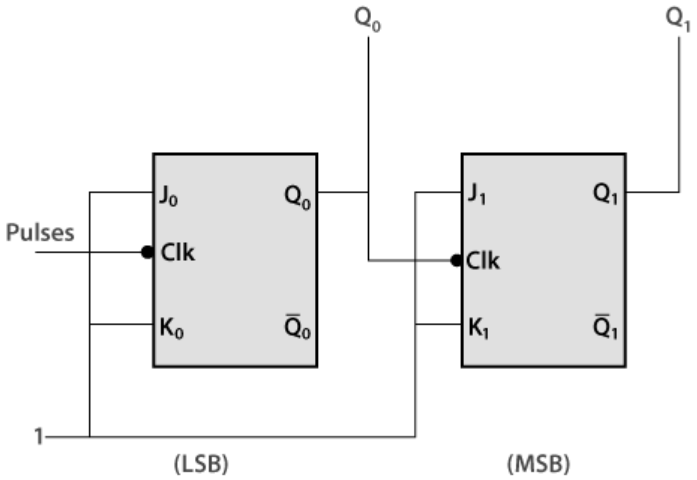
Up-Down Counter

The up and down counter is a special type of bi-directional counter which counts the states either in the forward direction or reverse direction. It also refers to a reversible counter.

Binary Ripple Counter

A **Binary counter** is a **2-Mod counter** which counts up to 2-bit state values, i.e., 22 = 4 values. The flip flops having similar conditions for toggling like T and JK are used to construct the **Ripple counter**. Below is a circuit diagram of **a binary ripple counter**.

In the circuit design of the binary ripple counter, two JK flip flops are used. The high voltage signal is passed to the inputs of both flip flops. This high voltage input maintains the flip flops at a state 1. In JK flip flops, the negative triggered clock pulse use.



The outputs Q_0 and Q_1 are the LSB and MSB bits, respectively. The truth table of JK flip flop helps us to understand the functioning of the counter.

J_n	K_n	Qn+1
0	0	Q_n
1	0	1
0	1	0
1	1	Q_n

When the high voltage to the inputs of the flip flops, the fourth condition is of the JK flip flop occurs. The flip flops will be at the state 1 when we apply high voltage to the input of the flip-flop. So, the states of the flip flops passes are toggled at the negative going end of the clock pulse. In simple words, the flip flop toggle when the clock pulse transition takes place from 1 to 0.

Ring Counter

A ring counter is a special type of application of the **Serial IN Serial OUT** Shift register.

The only difference between the shift register and the ring counter is that the last flip flop outcome is taken as the output in the shift register. But in the ring counter, this outcome is passed to the first flip flop as an input.

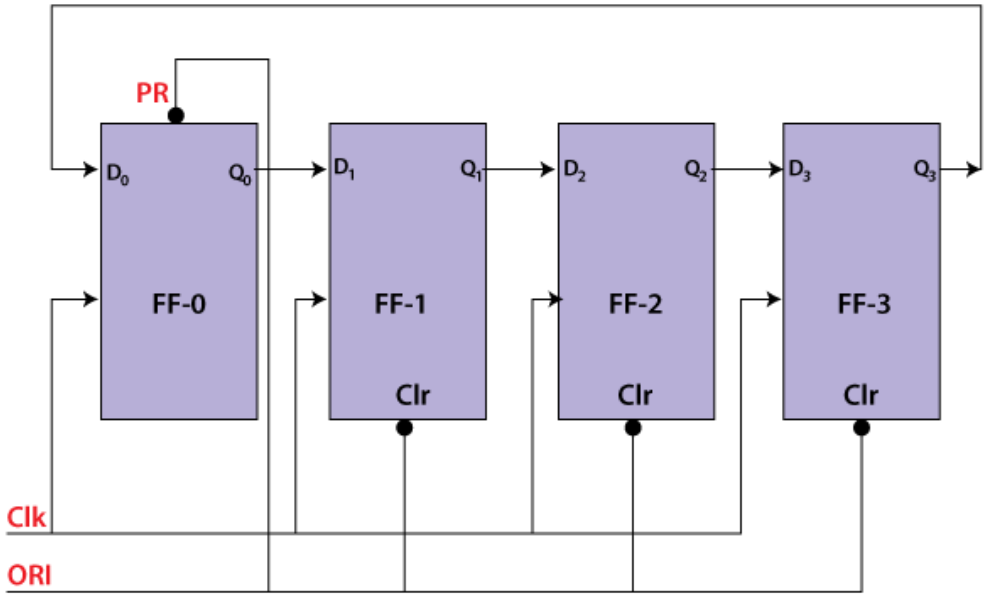
All of the remaining things in the ring counter are the same as the shift register.

In **the Ring counter**

No. of states in Ring counter = No. of flip-flop used

Below is the block diagram of the 4-bit ring counter. Here, we use 4 **D flip flops**. The same clock pulse is passed to the clock input of all the flip flops as a synchronous counter. The **Overriding input(ORI)** is used to design this circuit.

The Overriding input is used as **clear** and **pre-set**.



The output is 1 when the pre-set set to 0. The output is 0 when the clear set to 0. Both PR and CLR always work in value 0 because they are active low signals.

1. PR = 0, Q = 1
2. CLR = 0, Q = 0

These two values(always fixed) are independent with the input D and the Clock pulse (CLK).

Working

The ORI input is passed to the PR input of the first flip flop, i.e., FF-0, and it is also passed to the clear input of the remaining three flip flops, i.e., FF-1, FF-2, and FF-3. The pre-set input set to 0 for the first flip flop. So, the output of the first flip flop is one, and the outputs of the remaining flip flops are 0. The output of the first flip flop is used to form the ring in the **ring counter** and referred to as **Pre-set 1**.

ORI	Clk	Q_0	Q_1	Q_2	Q_3
low	X	1	0	0	0
high	low	0	1	0	0
high	low	0	0	1	0
high	low	0	0	0	1
high	low	1	0	0	0

In the above table, the highlighted 1's are **pre-set 1**.