**UNIVERSITY OF PORT HARCOURT**

**PORT HARCOURT**

**FACULTY OF SCIENCE**

**DEPARTMENT OF COMPUTER SCIENCE**

**ASSIGNMENT:**

***DISCUSS THE DIFFERENT BEHAVIOURS OF THE AND, OR, XOR, NOT, NAND, NOR AND XNOR GATES, AND WITH TRUTH TABLES, SHOW THEIR OPERATIONS WITH THE FLIP-FLOPS, COUNTERS & REGISTERS, AND THE LAWS THAT GOVERN THEIR OPERATIONS.***

**BY:**

**CHOKO BENJAMIN**

**U2017/5570247**

**COURSE TITLE:**

***COMPUTER ARCHITECT I***

**COURSE CODE:**

**CSC 382.1**

**COURSE LECTURER:**

***DR. F. EGBONO***

**MAY, 2021.**

**THE LOGIC GATES IN THE COMPUTER SYSTEM.**

Basically, there are various operations performed by different logic gates that include addition, multiplication, inverse, etc. These operations distinct the different logic gates. The operation performed by the logic gates is referred as a **boolean expression.** Here in this article, we will discuss the factors that differentiate these logic gates from each other.

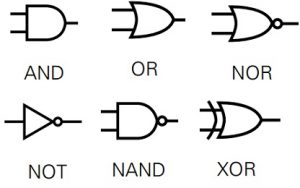
A logic gate is a basic building block of a digital circuit that has two inputs and one output. The relationship between the i/p and the o/p is based on a certain logic. These gates are implemented using electronic switches like transistors, diodes. But, in practice, basic logic gates are built using CMOS technology, FETS, and [MOSFET(Metal Oxide Semiconductor FET)s](https://www.elprocus.com/mosfet-as-a-switch-circuit-diagram-free-circuits/). Logic gates are [used in microprocessors, microcontrollers](https://www.elprocus.com/microprocessor-and-microcontroller/), embedded system applications, and in electronic and [electrical project circuits](https://www.elprocus.com/top-electrical-project-ideas-for-engineering-students/). The basic logic gates are categorized into seven: AND, OR, XOR, NAND, NOR, XNOR, and NOT. These logic gates with their logic gate symbols and truth tables are explained below.



Basic Logic Gates Operation

### Types of Logic Gates

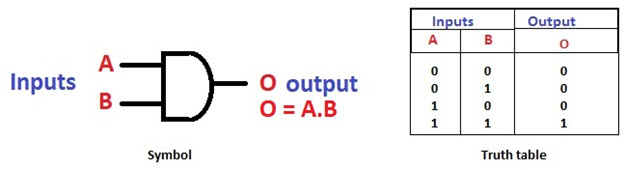
The different types of logic gates and symbols with truth tables are discussed below.



Basic Logic Gates

#### **AND Gate**

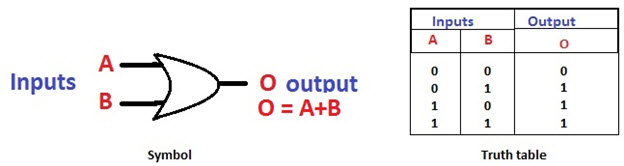
The AND gate performs logical conjunction based on the combinations of its inputs. **The output of this gate is true only when all the inputs are true.** When one or more inputs of the AND gate’s inputs are false, then only the output of the AND gate is false. The truth table with two inputs is shown blow.



AND Gate and its Truth Table

#### **OR Gate**

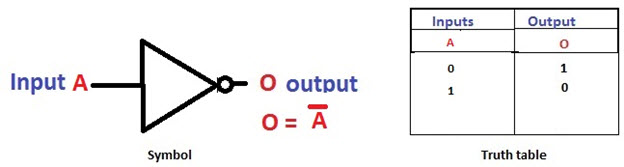
The OR gate is a digital logic gate that performs logical conjunction based on the combinations of its inputs. **The output of the OR gate is true only when one or more inputs are true.** If all the **inputs** of the gate are false, then only the output of the OR gate is false.



**OR Gate and its Truth Table**

#### **NOT Gate**

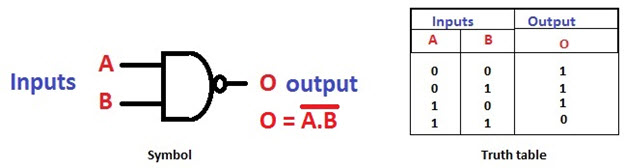
The NOT gate is a digital logic gate with one input and one output that **operates an inverter** operation of the input. The output of the NOT gate is the r**everse** of the input. ***When the input of the NOT gate is true then the output will be false and vice versa.*** By using this gate, we can implement NOR and NAND gates



**NOT Gate and Its Truth Table**

#### **NAND Gate**

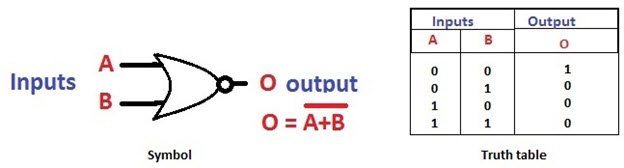
The NAND gate is a digital logic gate that performs the operation of the AND gate followed by the operation of the NOT gate. **NAND gate is designed by combining the AND and NOT gates**. If the input of the NAND gate is high, then the output of the gate will be low.



**NAND Gate and its Truth Table**

#### **NOR Gate**

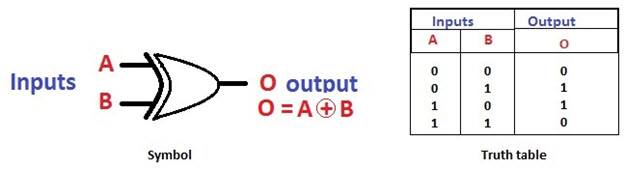
The NOR gate is designed by combining the OR and NOT gate. **When any one of the inputs of the NOR gate is true, then the output of the NOR gate will be false**.



**NOR Gate and Its Truth Table**

#### **Exclusive-OR Gate**

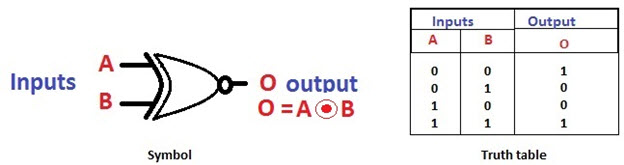
The Exclusive-OR gate is a digital logic gate with two inputs and one output. The short form of this gate is Ex-OR. It performs based on the operation of the OR gate. **If any one of the inputs of this gate is high, then the output of the EX-OR gate will be high.**



**EX-OR gate and Its Truth Table**

#### **Exclusive-NOR Gate**

The Exclusive-NOR gate is a digital logic gate with two inputs and one output. The short form of this gate is Ex-NOR. It performs based on the operation of the NOR gate. **When both the inputs of this gate are high, then the output of the EX-NOR gate will be high**. But, if any one of the inputs is high (but not both), then the output will be low.



**EX-NOR Gate and Its Truth Table**

**THE FUNCTION OF BASIC LOGIC GATES**

* For AND Gate – If both the inputs are high then the output is also high
* For OR Gate – If a minimum of one input is high then the output is High
* For XOR Gate – If the minimum of one input is high then only the output is high
* NAND Gate – If the minimum of one input is low then the output is high
* NOR Gate – If both the inputs are low then the output is high.

**RS FLIP FLOP AND LOGIC GATES**

A **Flip Flop** is a bi-stable device. There are three classes of flip flops they are known as **Latches**, **pulse-triggered** flip-flop, **Edge- triggered** flip flop. In this set word means that the output of the circuit is equal to 1 and the word reset means that the output is 0.

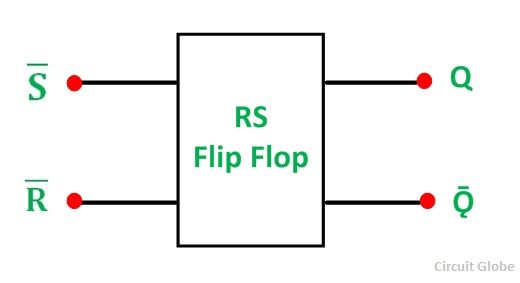
There are two types of flip flop one is **RS Flip Flop** and **JK Flip Flop**.

The RS Flip Flop is considered as one of the most basic sequential logic circuits. The Flip Flop is a one-bit memory bi-stable device.

It has two inputs, one is called**“SET”** which will set the device (output = 1) and is labelled S and another is known as**“RESET”** which will reset the device (output = 0) labelled as R. The RS stands for **SET/RESET.**

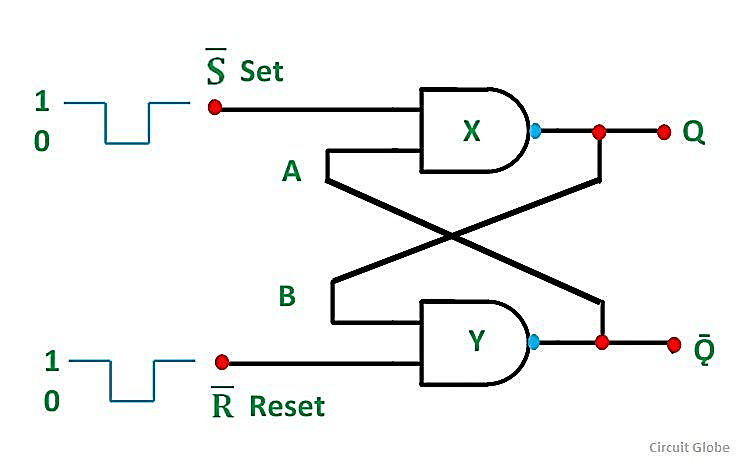
The flip-flop is reset back to its original state with the help of RESET input and the output is Q that will be either at logic level “1” or logic”0”. It depends upon the set/reset condition of the flip-flop. Flip flop word means that it can be**“FLIPPED”** into one logic state or **“FLOPPED”** back into another.

The basic NAND gate RS Flip Flop circuit is used to store the data and thus provides feedback from both of its outputs again back to its inputs. The RS Flip Flop actually has three inputs, SET, RESET and its current output Q relating to its current state.

The symbol of the RS Flip-Flop is shown below:  
[](https://circuitglobe.com/wp-content/uploads/2015/12/RS-FLIP-FLOP-FIG-1-compressor.jpg)

**The NAND Gate RS Flip Flop**

A pair of cross-coupled 2 unit NAND gates is the simplest way to make any basic one-bit set/reset RS Flip Flop. It forms Set/Reset bi-stable or an active LOW RS NAND gate latch. The feedback is fed from each output to one of the other NAND gate input.

The device consists of two inputs; one is known as SET, (S) and the other is called as RESET, (R).The two outputs are Q and Q bar as shown in the figure below:  
[](https://circuitglobe.com/wp-content/uploads/2015/12/RS-FLIP-FLOP-FIG-2-compressor.jpg)  
**The Set State**

Considering the above circuit. If the input R is at logic level “0” (R = 0) and input S is at the logic “1” (S = 1), the NAND gate Y has, at least, one of its inputs at a logic “0”. Therefore, its output Q must be at a logic level “1” (NAND gate principles). The Output (Q) is fed back to the input “A”. Both the inputs of the NAND gates X are at logic “1”, and therefore, its output Q must be at the logic level”0”.

The reset input R changes its state, and goes HIGH to logic “1” with S constant at logic “1”. The NAND gate Y input are now (R = 1) and (B = 0). The output at Q remains at HIGH or at logic level “1” as one of its inputs is still at logic level “0”.

As a result, there is no change in state. Therefore, the flip-flop circuit is said to be “LATCHED” or “SET” with Q = 1 and Ǭ = 0.

**The Reset State**

In this second stable state, Q is at logic level ‘0” and its inverse output Q is at logic level “1”. And is given by (R = 1) and (S = 0). As gate X has one of its inputs at a logic “0” its output Q must equal logic level “1”. (According to the NAND gate principle). The output Q is fed to input B, so both the inputs to NAND gate Y are at logic “1”, therefore, Q = 0.

If the set input S now changes the state to logic “1” with the input R remaining at logic “1”, the output Q still remains LOW at logic level “0”. And there is no change in the state.

Therefore, the flip-flop circuits “RESET” state has been latched.

The **truth table** of the Set/Reset is given below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **State** | **S** | **R** | **Q** | **Ǭ** | **Description** |
| **SET** | 1 | 0 | 1 | 0 | Set Q >>1 |
| 1 | 1 | 1 | 0 | No Change |
| **RESET** | 0 | 1 | 0 | 1 | Reset Q >>0 |
| 1 | 1 | 0 | 1 | No Change |
| **INVALID** | 0 | 0 | 0 | 1 | Memory with Q = 0 |
| 0 | 0 | 1 | 0 | Memory with Q = 1 |

From the truth table, it is clear that when both the inputs S = 1 and R =1 the outputs Q, and Ǭ can be at either logic level ‘1’ or “0” depending upon the state of the inputs.

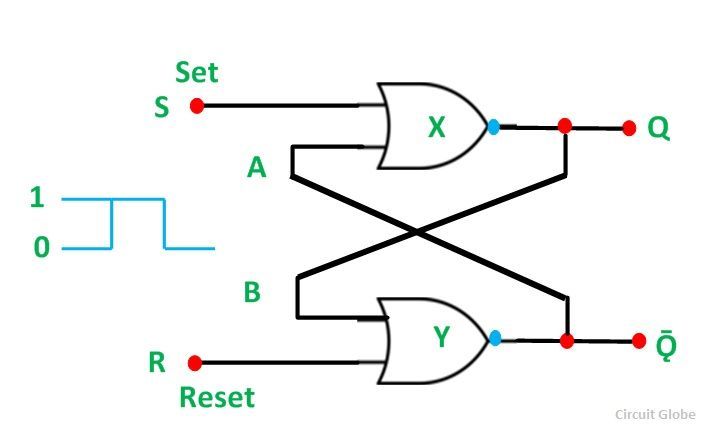
When the input state R = 0 and S = 0 is an invalid condition and must be avoided because this will give both outputs Q and Ǭ at logic level “1” at the same time and the necessary condition is that Q to be the inverse of Ǭ.

The flip-flop goes to an unstable state as both the output goes LOW. This unstable condition arises when the LOW input is switched to HIGH. The flip-flop switches to one state or the other and any one output of the flip-flop switches faster than the other. This unstable condition is known as Meta- stable state.

The bi-stable RS flip flop is activated or set at logic “1” applied to its S input and deactivated or reset by a logic “1” applied to R. The RS flip-flop is said to be in an invalid condition if both the set and reset inputs are activated simultaneously.

**The NOR Gate RS Flip Flop**

The circuit diagram of the NOR gate flip-flop is shown in the figure below:

[](https://circuitglobe.com/wp-content/uploads/2015/12/RS-FLIP-FLOP-FIG-3-compressor1.jpg)

A simple one bit RS Flip Flops are made by using two cross-coupled NOR gates connected in the same configuration. The circuit will work similar to the NAND gate circuit.

The **truth table** of the **NOR gate RS Flip Flop**:

| **S** | **R** | **Q** | **Ǭ** |
| --- | --- | --- | --- |
| 0 | 0 | No Change | No Change |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |

The inputs are active HIGH and the invalid condition exists when both its inputs are at logic level ‘1’.

**COUNTERS IN DIGITAL LOGIC**

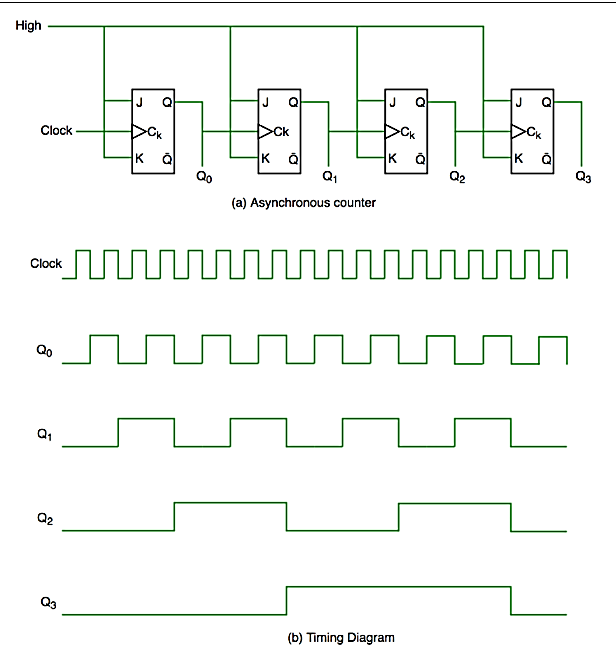
A C**ounter** is a device which stores (and sometimes displays) the number of times a particular event or process has occurred, often in relationship to a clock signal. Counters are used in digital electronics for counting purpose, they can count specific event happening in the circuit. For example, in UP counter a counter increases count for every rising edge of clock. Not only counting, a counter can follow the certain sequence based on our design like any random sequence 0,1,3,2… .They can also be designed with the help of flip flops.

Counters are broadly divided into two categories

1. Asynchronous counter
2. Synchronous counter

1. **Asynchronous Counter**

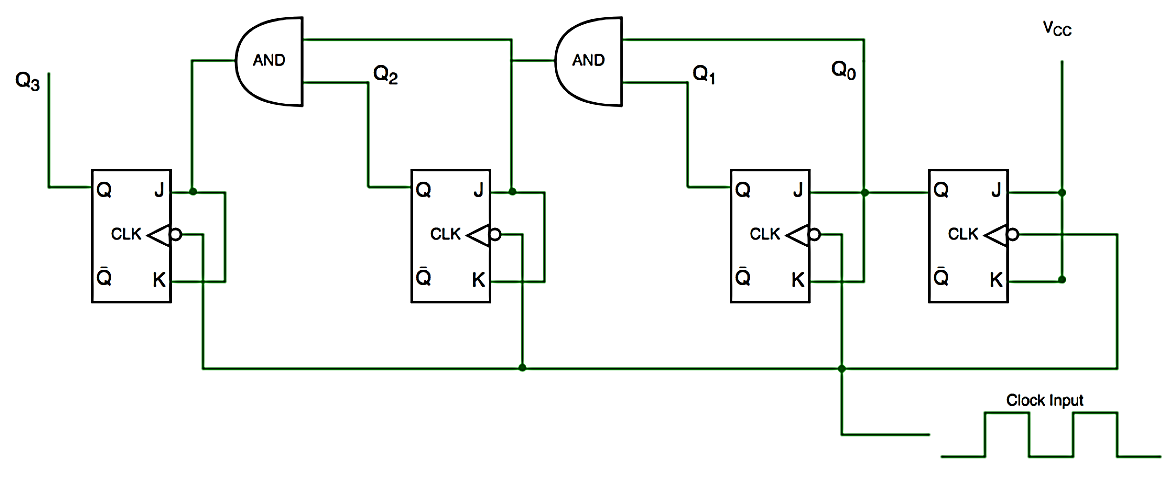
In asynchronous counter we don’t use universal clock, only first flip flop is driven by main clock and the clock input of rest of the following flip flop is driven by output of previous flip flops. We can understand it by following diagram-



It is evident from timing diagram that Q0 is changing as soon as the rising edge of clock pulse is encountered, Q1 is changing when rising edge of Q0 is encountered(because Q0 is like clock pulse for second flip flop) and so on. In this way ripples are generated through Q0,Q1,Q2,Q3 hence it is also called **RIPPLE counter.**

2. **Synchronous Counter**

Unlike the asynchronous counter, synchronous counter has one global clock which drives each flip flop so output changes in parallel. The one advantage of synchronous counter over asynchronous counter is, it can operate on higher frequency than asynchronous counter as it does not have cumulative delay because of same clock is given to each flip flop.



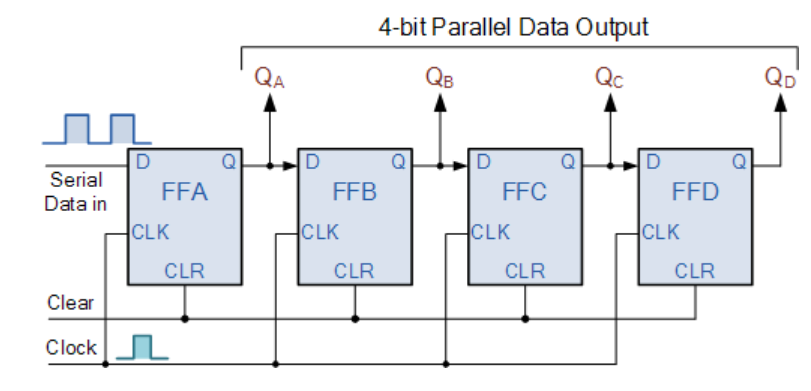
From circuit diagram we see that Q0 bit gives response to each falling edge of clock while Q1 is dependent on Q0, Q2 is dependent on Q1 and Q0 , Q3 is dependent on Q2,Q1 and Q0.

**REGISTERS AND LOGIC GATES**

A **Register** is a device which is used to store such information. It is a group of flip flops connected in series used to store multiple bits of data.

The information stored within these registers can be transferred with the help of **shift registers**. Shift Register is a group of flip flops used to store multiple bits of data. The bits stored in such registers can be made to move within the registers and in/out of the registers by applying clock pulses. An n-bit shift register can be formed by connecting n flip-flops where each flip flop stores a single bit of data.  
The registers which will shift the bits to left are called “Shift left registers”.

**Serial-in to Parallel-out (SIPO) Shift Register**



### 4-bit Serial-in to Parallel-out Shift Register

The operation is as follows. Lets assume that all the flip-flops ( FFA to FFD ) have just been RESET ( CLEAR input ) and that all the outputs QA to QD are at logic level “0” ie, no parallel data output.

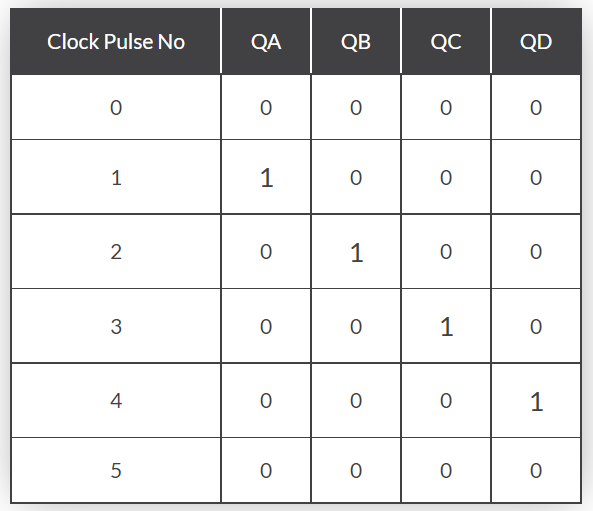
If a logic “1” is connected to the DATA input pin of FFA then on the first clock pulse the output of FFA and therefore the resulting QA will be set HIGH to logic “1” with all the other outputs still remaining LOW at logic “0”. Assume now that the DATA input pin of FFA has returned LOW again to logic “0” giving us one data pulse or 0-1-0.

The second clock pulse will change the output of FFA to logic “0” and the output of FFB and QB HIGH to logic “1” as its input D has the logic “1” level on it from QA. The logic “1” has now moved or been “shifted” one place along the register to the right as it is now at QA.

When the third clock pulse arrives this logic “1” value moves to the output of FFC ( QC ) and so on until the arrival of the fifth clock pulse which sets all the outputs QA to QD back again to logic level “0” because the input to FFA has remained constant at logic level “0”.

The effect of each clock pulse is to shift the data contents of each stage one place to the right, and this is shown in the following table until the complete data value of  0-0-0-1 is stored in the register. This data value can now be read directly from the outputs of QA to QD.

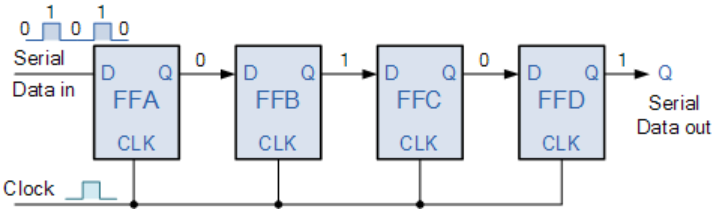
**Basic Data Movement Through A Shift Register**



## Serial-in to Serial-out (SISO) Shift Register

This **shift register** is very similar to the SIPO above, except were before the data was read directly in a parallel form from the outputs QA to QD, this time the data is allowed to flow straight through the register and out of the other end. Since there is only one output, the DATA leaves the shift register one bit at a time in a serial pattern, hence the name **Serial-in to Serial-Out Shift Register** or **SISO**.

### 4-bit Serial-in to Serial-out Shift Register

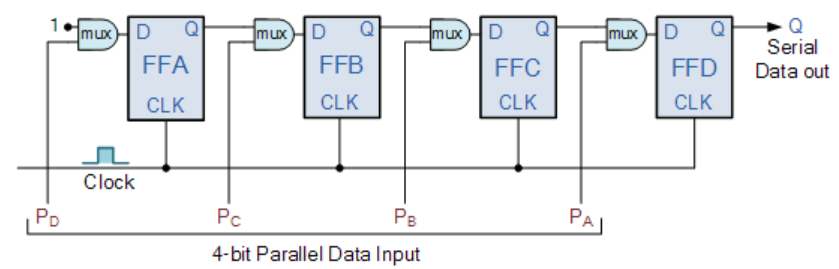


## Parallel-in to Serial-out (PISO) Shift Register

The Parallel-in to Serial-out shift register acts in the opposite way to the serial-in to parallel-out one above. The data is loaded into the register in a parallel format in which all the data bits enter their inputs simultaneously, to the parallel input pins PA to PD of the register. The data is then read out sequentially in the normal shift-right mode from the register at Q representing the data present at PA to PD.

This data is outputted one bit at a time on each clock cycle in a serial format. It is important to note that with this type of data register a clock pulse is not required to parallel load the register as it is already present, but four clock pulses are required to unload the data.

**4-bit Parallel-in to Serial-out Shift Register**



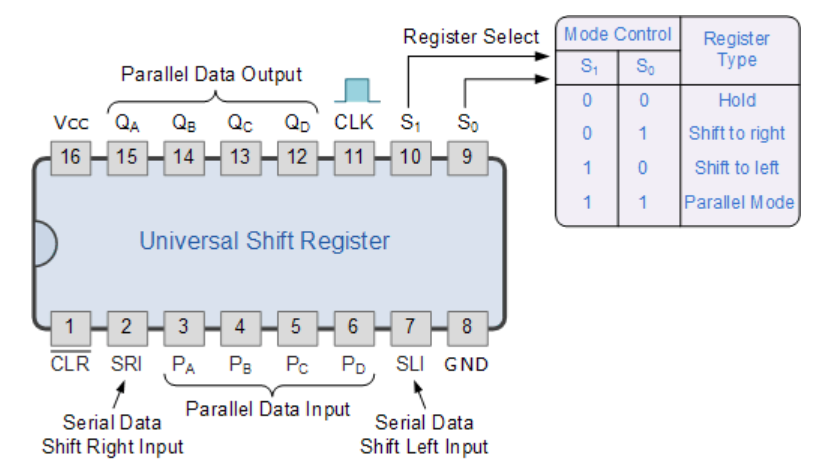
As this type of shift register converts parallel data, such as an 8-bit data word into serial format, it can be used to multiplex many different input lines into a single serial DATA stream which can be sent directly to a computer or transmitted over a communications line. Commonly available IC’s include the 74HC166 8-bit Parallel-in/Serial-out Shift Registers.

## Universal Shift Register

Today, there are many high speed bi-directional “universal” type **Shift Registers** available such as the TTL 74LS194, 74LS195 or the CMOS 4035 which are available as 4-bit multi-function devices that can be used in either serial-to-serial, left shifting, right shifting, serial-to-parallel, parallel-to-serial, or as a parallel-to-parallel multifunction data register, hence their name “Universal”.

These universal shift registers can perform any combination of parallel and serial input to output operations but require additional inputs to specify desired function and to pre-load and reset the device. A commonly used universal shift register is the TTL 74LS194 as shown below.

### 4-bit Universal Shift Register 74LS194



## Summary Shift Register

Then to summarise a little about **Shift Registers**

* A simple **Shift Register** can be made using only D-type flip-Flops, one flip-Flop for each data bit.
* The output from each flip-Flop is connected to the D input of the flip-flop at its right.
* Shift registers hold the data in their memory which is moved or “shifted” to their required positions on each clock pulse.
* Each clock pulse shifts the contents of the register one bit position to either the left or the right.
* The data bits can be loaded one bit at a time in a series input (SI) configuration or be loaded simultaneously in a parallel configuration (PI).
* Data may be removed from the register one bit at a time for a series output (SO) or removed all at the same time from a parallel output (PO).

# DEMORGAN’S THEOREM

DeMorgan´s Theorem and Laws can be used to find the equivalency of the NAND and NOR gates

As we have seen previously, Boolean Algebra uses a set of laws and rules to define the operation of a digital logic circuit with “0’s” and “1’s” being used to represent a digital input or output condition. Boolean Algebra uses these zeros and ones to create truth tables and mathematical expressions to define the digital operation of a logic AND, OR and NOT (or inversion) operations as well as ways of expressing other logical operations such as the XOR (Exclusive-OR) function.

While George Boole’s set of laws and rules allows us to analyise and simplify a digital circuit, there are two laws within his set that are attributed to **Augustus DeMorgan** (a nineteenth century English mathematician) which views the logical NAND and NOR operations as separate NOT AND and NOT OR functions respectively.

But before we look at **DeMorgan’s Theory** in more detail, let’s remind ourselves of the basic logical operations where A and B are logic (or Boolean) input binary variables, and whose values can only be either “0” or “1” producing four possible input combinations, 00, 01, 10, and 11.

### Truth Table for Each Logical Operation

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Input Variable** | | **Output Conditions** | | | |
| A | B | AND | NAND | OR | NOR |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 |

The following table gives a list of the common logic functions and their equivalent Boolean notation where a “.” (a dot) means an AND (product) operation, a “+” (plus sign) means an OR (sum) operation, and the complement or inverse of a variable is indicated by a bar over the variable.

|  |  |
| --- | --- |
| **Logic Function** | **Boolean Notation** |
| AND | A.B |
| OR | A+B |
| NOT | A |
| NAND | A .B |
| NOR | A+B |

## Two Basic Laws of DeMorgan’s Theory

DeMorgan’s Theorems are basically two sets of rules or laws developed from the Boolean expressions for AND, OR and NOT using two input variables, A and B. These two rules or theorems allow the input variables to be negated and converted from one form of a Boolean function into an opposite form.

**DeMorgan’s first theorem** *states that two (or more) variables NOR´ed together is the same as the two variables inverted (Complement) and AND´ed,* while the

**second theorem states** *that two (or more) variables NAND´ed together is the same as the two terms inverted (Complement) and OR´ed*. That is replace all the OR operators with AND operators, or all the AND operators with an OR operators.

### DeMorgan’s First Theorem

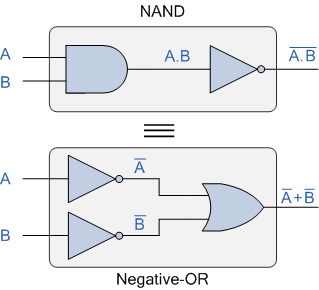
DeMorgan’s First theorem proves that when two (or more) input variables are AND’ed and negated, they are equivalent to the OR of the complements of the individual variables. Thus the equivalent of the NAND function will be a negative-OR function, proving that A.B = A+B. We can show this operation using the following table.

### Verifying DeMorgan’s First Theorem using Truth Table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Inputs** | | **Truth Table Outputs For Each Term** | | | | | |
| **B** | **A** | **A.B** | **A.B** | **A** | **B** | **A + B** |  |
| 0 | 0 | 0 | **1** | 1 | 1 | **1** |  |
| 0 | 1 | 0 | **1** | 0 | 1 | **1** |  |
| 1 | 0 | 0 | **1** | 1 | 0 | **1** |  |
| 1 | 1 | 1 | **0** | 0 | 0 | **0** |  |

We can also show that A.B = A+B using logic gates as shown.

### DeMorgan’s First Law Implementation using Logic Gates



The top logic gate arrangement of: A.B can be implemented using a standard NAND gate with inputs A and B. The lower logic gate arrangement first inverts the two inputs producing A and B. These then become the inputs to the OR gate. Therefore the output from the OR gate becomes: A+B

Then we can see here that a standard OR gate function with inverters (NOT gates) on each of its inputs is equivalent to a NAND gate function. So an individual NAND gate can be represented in this way as the equivalency of a NAND gate is a negative-OR.

### DeMorgan’s Second Theorem

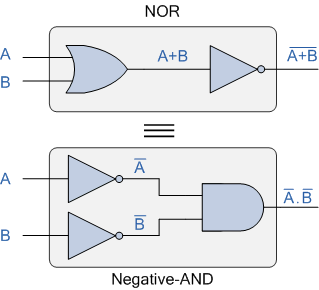
DeMorgan’s Second theorem proves that when two (or more) input variables are OR’ed and negated, they are equivalent to the AND of the complements of the individual variables. Thus the equivalent of the NOR function is a negative-AND function proving that A+B = A.B, and again we can show operation this using the following truth table.

**Verifying DeMorgan’s Second Theorem using Truth Table**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Inputs** | | **Truth Table Outputs For Each Term** | | | | |
| **B** | **A** | **A+B** | **A+B** | **A** | **B** | **A . B** |
| 0 | 0 | 0 | **1** | 1 | 1 | **1** |
| 0 | 1 | 1 | **0** | 0 | 1 | **0** |
| 1 | 0 | 1 | **0** | 1 | 0 | **0** |
| 1 | 1 | 1 | **0** | 0 | 0 | **0** |

We can also show that A+B = A.B using the following logic gates example.

### DeMorgan’s Second Law Implementation using Logic Gates



The top logic gate arrangement of: A+B can be implemented using a standard NOR gate function using inputs A and B. The lower logic gate arrangement first inverts the two inputs, thus producing A and B. Thus then become the inputs to the AND gate. Therefore the output from the AND gate becomes: A.B

Then we can see that a standard AND gate function with inverters (NOT gates) on each of its inputs produces an equivalent output condition to a standard NOR gate function, and an individual NOR gate can be represented in this way as the equivalency of a NOR gate is a negative-AND.

Although we have used DeMorgan’s theorems with only two input variables A and B, they are equally valid for use with three, four or more input variable expressions, for example:

For a 3-variable input

A.B.C = A+B+C

and also

A+B+C = A.B.C

For a 4-variable input

A.B.C.D = A+B+C+D

and also

A+B+C+D = A.B.C.D

## DeMorgan’s Equivalent Gates

We have seen here that by using DeMorgan’s Theorems we can replace all of the AND (.) operators with an OR (+) and vice versa, and then complements each of the terms or variables in the expression by inverting it, that is 0’s to 1’s and 1’s to 0’s before inverting the entire function.

Thus to obtain the DeMorgan equivalent for an AND, NAND, OR or NOR gate, we simply add inverters (NOT-gates) to all inputs and outputs and change an AND symbol to an OR symbol or change an OR symbol to an AND symbol as shown in the following table.

### DeMorgan’s Equivalent Gates

