“Digital Design and Boolean Algebra”

Dr. Cahit Karakuş, February-2019

Analogue and Digital Signals

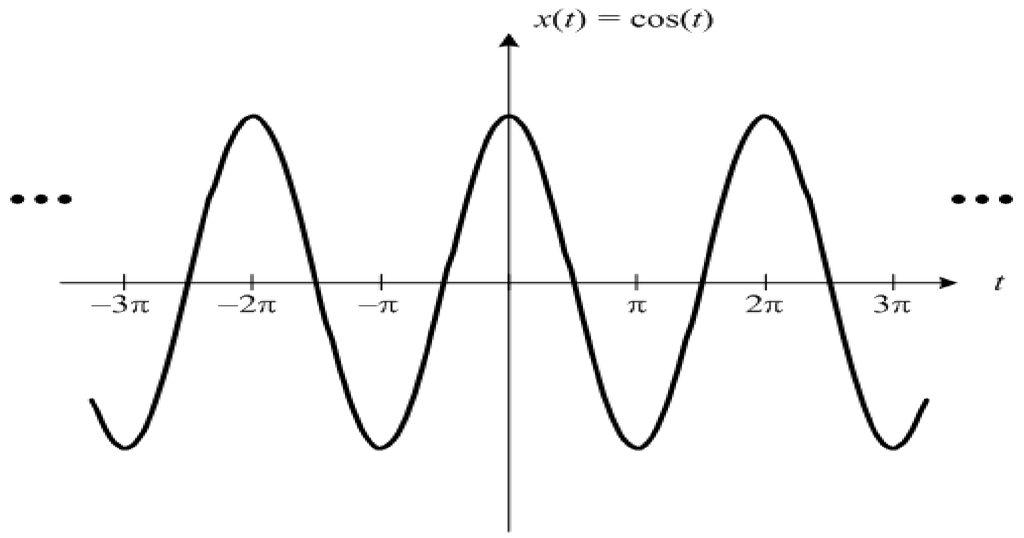
How is data represented inside the computer?

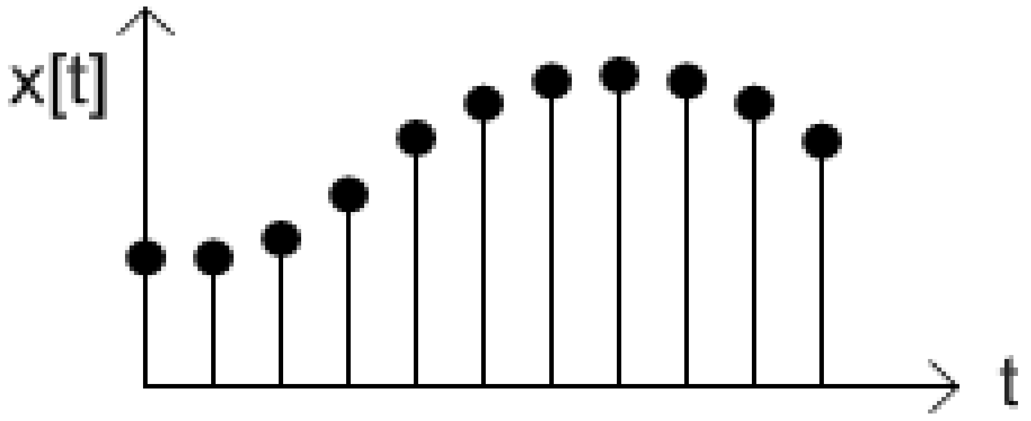
- Inside the computer, data is represented as a sequence of 1s and 0s.
- Since data in the computer system is represent as digital signals (bit: 1/0) while arithmetic and logical operations are performed, stored in memories, and transferred between relevant units, so analog signals must be converted into digital signals.
- Signals spread far away in the form of waves. Waves carry messages in signals.
- Signals: Acoustic signals, Seismic Signals, Electrical signals, Electromagnetic signals, Heat, Vibration, ...
- Messages/Symbols: Numbers (positive, negative, integer, float, fraction), Characters, Texts, Picture, Image, Keyboard keys
- Analog signals: These are signals whose amplitude, frequency and phase change over time. $X(t)=A(t)*Sin(wt+\phi(t))$;
  - $w=2\pi f$, f: frequency (Hz=1/sec), ϕ: phase (degrees or radians), A: Amplitude (unit, volt, ampere)

Classification of signals

Signals are basically classified into two different types as follows.

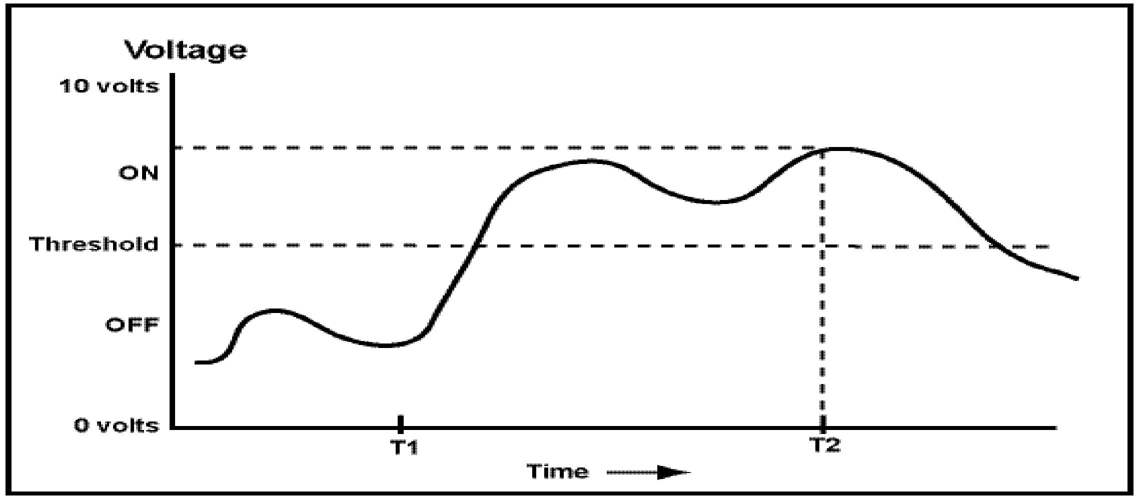
- Continuous time signals, Analog signals
- Discrete (Ayrık) time signals: It is the name given to the output signal obtained by measuring the input value at certain intervals or levels. The discrete-time signal is derived from the input signal by the sampling process. Samples taken from the discrete-time signal are converted into a digital signal by quantization.

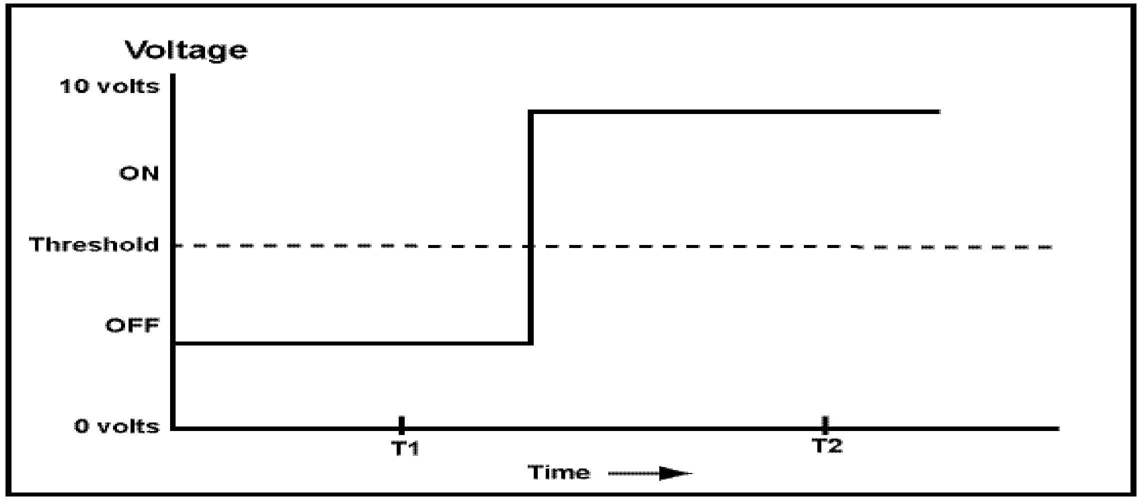




Analog to Binary Signal

- A voltage below the *threshold*
  - Off (0)
- A voltage above the *threshold*
  - On (1)



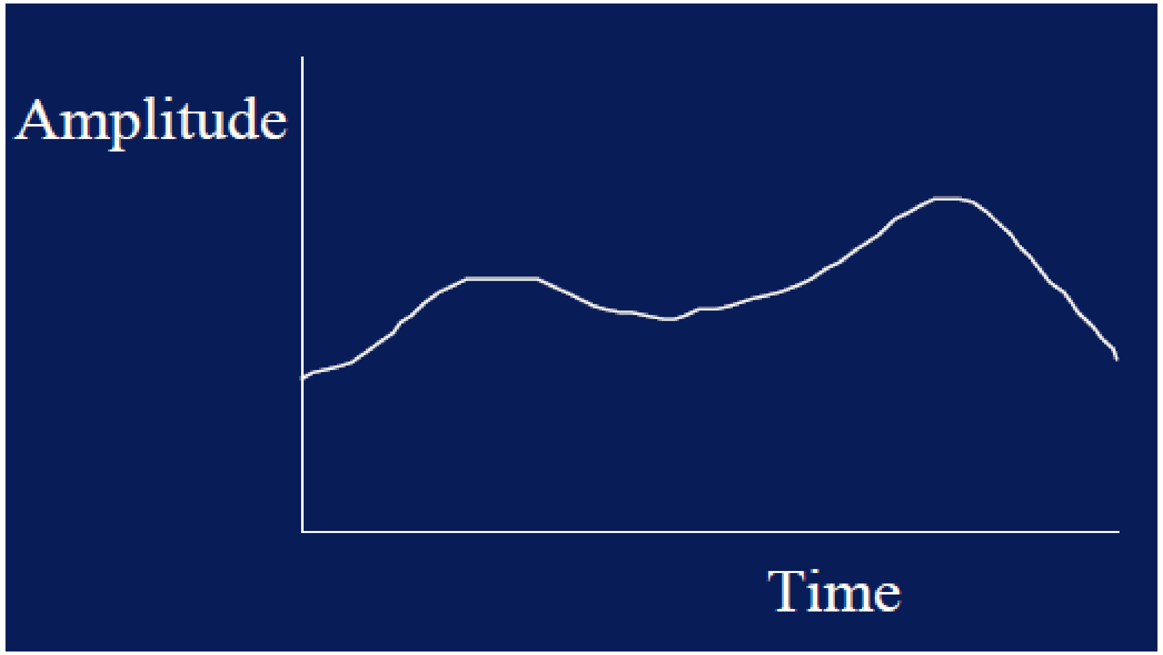


Analogue and Digital Signals

- Calculation with numbers is usually done in base 10 arithmetic
- Easier to effect machine computation in base 2 or binary notation
- We can also use base 2 or binary notation to represent logic values: TRUE and FALSE
- Manipulation of these (digital) logic values is subject to the laws of logic as set out in the formal rules of Boolean algebra
- An analogue signal can have any value within certain operating limits
- For example, in a (common emitter) amplifier, the output (O/P) can have any value between 0v and 10v.
- A digital signal can only have a fixed number of values within certain tolerances

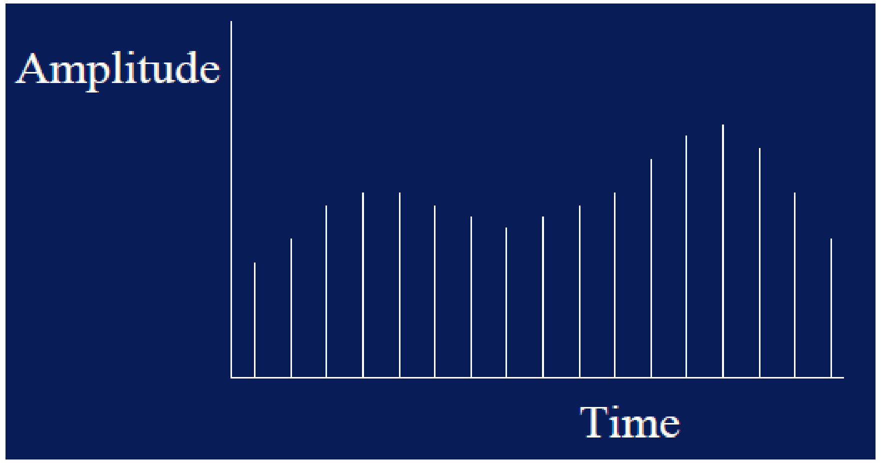
Analogue Signals

- The amplitude is defined at all moments in time



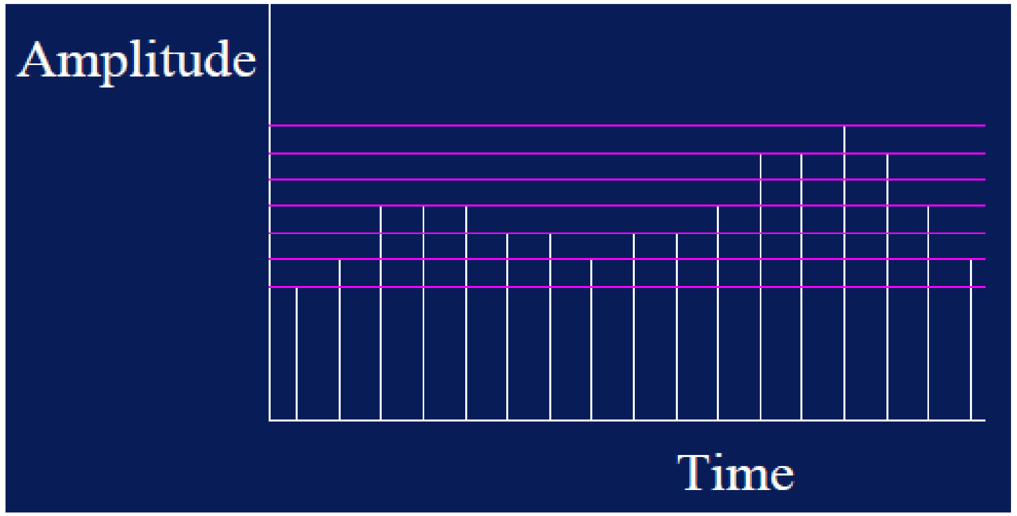
First step of sampling: Obtain a discrete signal from analogue signal

- It is a sampled version of the analogue signal
- Only defined at certain discrete times
- DISCRETE TIME SIGNAL
- A digital signal is a sampled version of the analogue signal
- Analog signal sampling interval=1/fmax, [sec]. Here fmax=maximum frequency of the analog signal.



The second step of sampling: Quantization

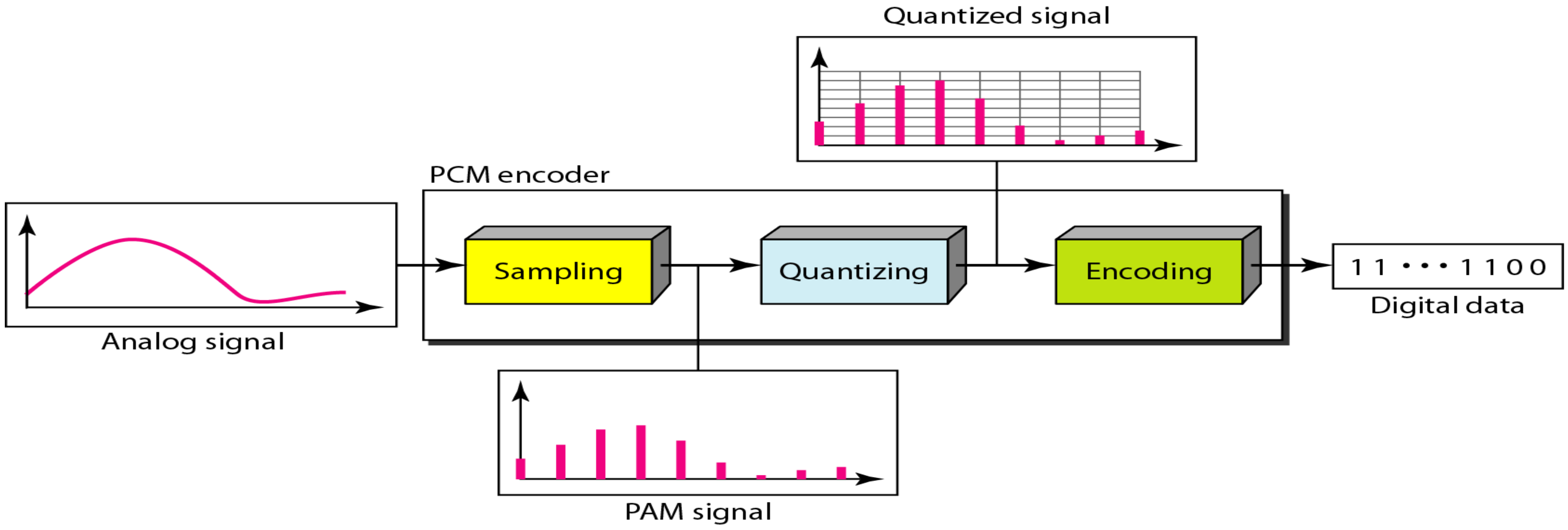
- The amplitude may also be restricted to take on discrete values only
- In which case it is said to be quantized
- Every level of amplitute represent 8 bits binary numbering system.
- There are 2^8 levels in an analogue signal.
- Quantization introduces errors which depend on the step size or the resolution
- Signals (voltages or currents) which are samples and quantized are said to be DIGITAL
- They can be represented by a sequence of binary numbers



The third step of sampling: Encoding

- The messages produced by all the components that make up the universe are transmitted via analog signals; In this way, they are in interactive communication with each other.
- While the analog signal is converted into a digital signal, samples are taken from the amplitude and phase values at certain time intervals (sampling frequency). This process is called the sampling function. When sampling, the sampling frequency must be greater than or equal to twice the maximum frequency of the analog signal.
- An analog signal has a lot of frequencies. Also, there are maksimum and minumum frequency. Bandwidth, BW: fmax – fmin. Sampling frequency, fs >= 2* BW >= fmax
- The values of the samples taken from the analog signal are assigned to discrete values within the amplitude scaling range. This process is called quantization. During this assignment process, quantization errors occur depending on the sampling time interval, quantization values, translation and resolution.
- Representing discrete values with a certain number of binary number systems is called encoding in the binary number system. Each discrete amplitude value is represented by a certain number of binary (0/1) numbers. Thus, digital signals are obtained. For example, 8 bit can be taken.

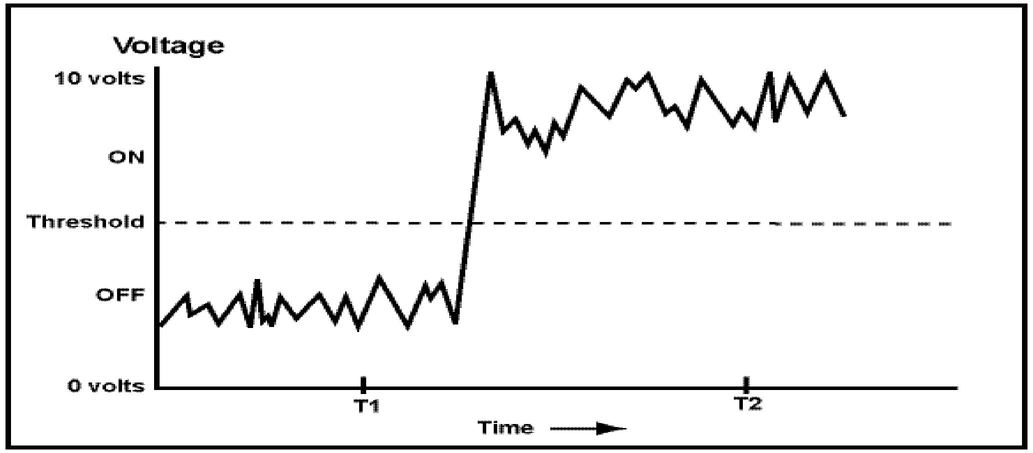
Digitization of analog signal

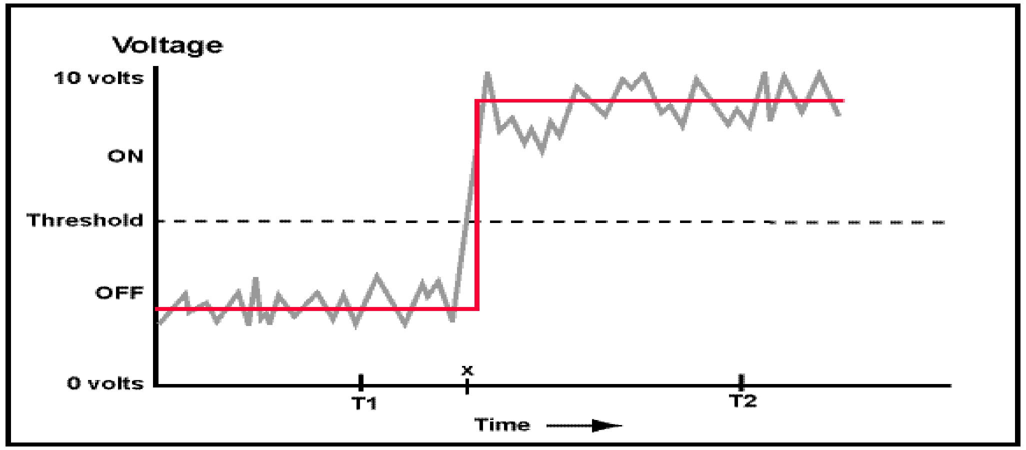


- The values of the samples taken from the analog signal are assigned to discrete values in the amplitude scaling range. This process is called quantization. During this assignment process, quantization errors occur depending on the sampling time interval, quantization values, translation and resolution.Representing discrete values with a certain number of binary number systems is called encoding in the binary number system. Each discrete amplitude value is represented by a certain number of binary (0/1) numbers. Thus, digital signals are obtained.

Noise on Transmission

- When the signal is transferred it will pick up noise from the environment
- Even when the noise is present the binary values are transmitted without error
- Recovery - Filtering





Nyquist Sampling Theorem

When an analog signal is converted to digital signal and trasfered then converted back to an analog signal, how can obtain the same analog signal. The sampling frequency must be equal to or greater than wice the bandwidth of the signal in order to obtain the same signal.

$$f_s \geq 2^* f_{max,\ if\ fmin=0}$$
$$f_s \geq 2^* BW$$

- Here fs is the sampling frequency and fmax is the maximum frequency in the signal; BW is the bandwidth of signal. BW=fmax – fmin.
- Frequency is the number of periods in one second in an analogue sigal. Frequency is also the number of vibration in an one second. f=1/T. T: period [second], f: frequency [Hz=1/second]In telephone communication, bandwidth is B= 4KHz (Understanding, Recognition, Feeling) and fs=2*B. Then the sampling interval is T=1/8KHz=125microseconds.

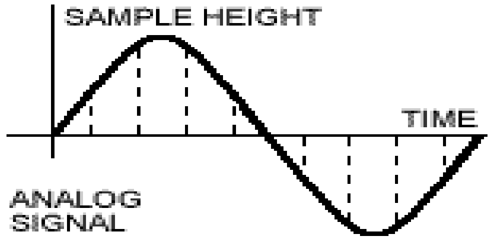
Example

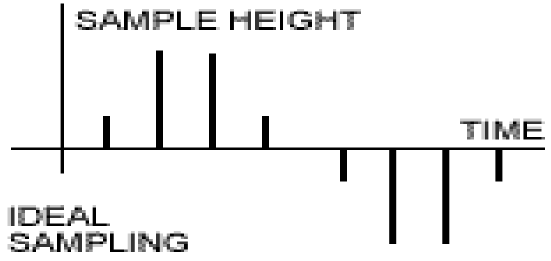
- F1=4KHZ, f2=3200Hz, f3=6KHz, f4=200Hz, f5=8KHz, f6=10KHz
- Find bandwidth (BW), find fmin and fmax. Note: you must conert all units to basic unit.

- F1=4000Hz
- F2=3200Hz
- F3=6000Hz
- F4=200Hz
- F5=8000Hz
- F6=10000Hz
- Fmin=f4=200Hz
- Fmax=10000Hz
- BW=fmax-fmin=10000Hz-200Hz=9800Hz

The Sampling

- Sampling is converting a continuous time signal into a discrete time signal
- Sampling is usually done at equal time intervals; This interval is called the sampling interval. The reciprocal of the sampling interval is called sampling frequency or sampling rate. The unit of sampling rate is Hz.
- In accordance with the sampling theorem, telephone voice signals frequency ranges from 300 Hz to 3400 Hz. It is taken as 4KHz, the sampling frequency is taken as greater than or equal to 8000 Hz.
- If 8000 samples are taken from an analog signal at equal intervals per second and one sample is represented by 8 bits, how many bits are taken per second? 8000 x 8=64 000 bit/sec=64Kbit/sec.
- Minimum capacity of a channel in data communication = 64Kbit/s.
- If each sample is represented by 8 bits, the number of sampling intervals (Quantization) is 28 = 256.
- 256 quantization ranges (layers) are obtained.
- After the sound waves are converted to an analog signal, a sample is taken at 125 µsec. Sampling interval, T=1/8000= 0.000125 sec. = 125 µs, where T is the sampling interval.

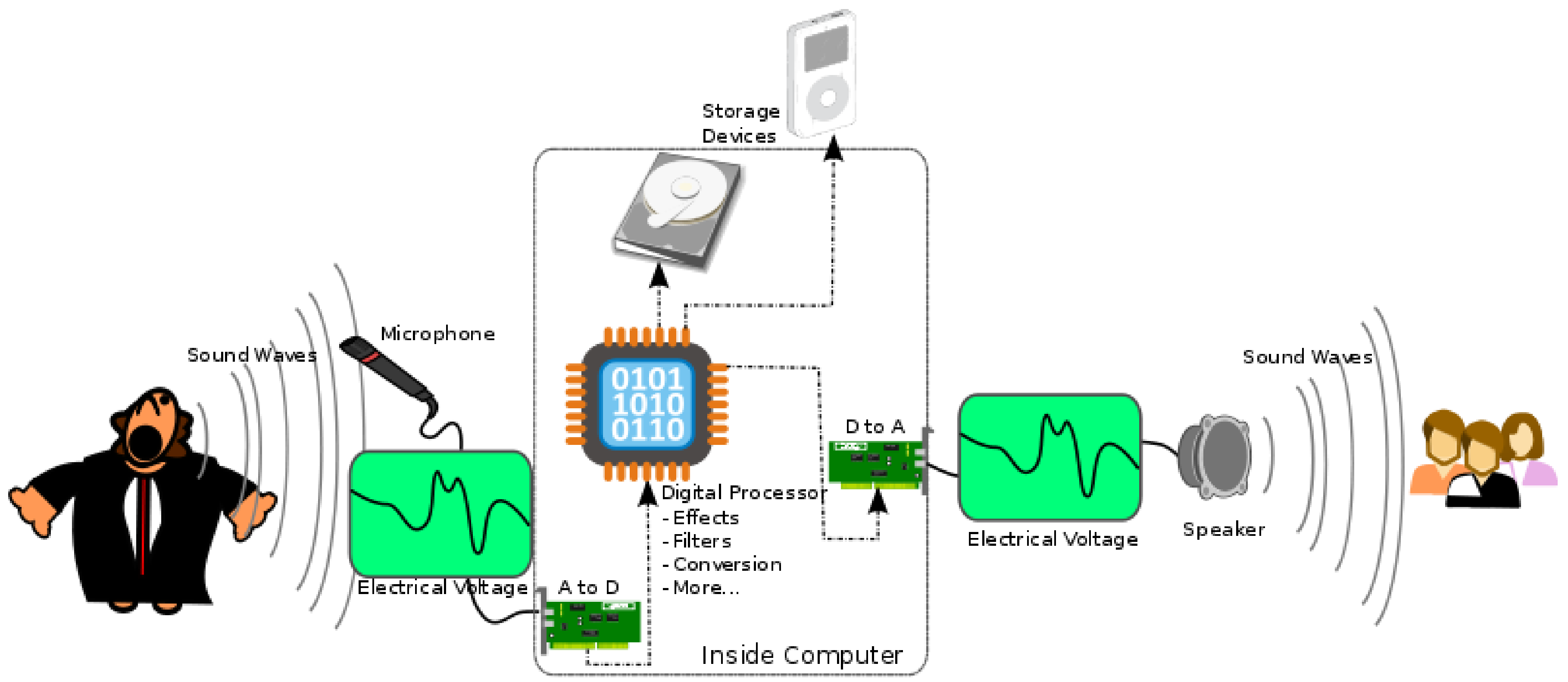




Ideal Sampling and Aliasing

- Sampled signal is discrete in time domain with spacing T_s
- Spectrum will repeat for every f_s Hz
- Aliasing (spectral overlapping – Loss of an analog signal within another analog signal) if f_s is too small ($f_s < 2f_m$)
- Nyquist sampling rate $f_s = 2f_m$
- Generally oversampling is done $\rightarrow f_s > 2f_m$

Lifecycle from Sound to Digital to Sound



Source: http://en.wikipedia.org/wiki/Digital_audio

Digital Design and Boolean Algebra

Fundamentals of Digital Logic

- A binary number system is used to represent digital logic values: 1/0 (True/False, Good/Bad, Day/Night, 0V/5V)
- Mathematical operations of digital logic values are governed by the laws specified in the rules of Boolean algebra.
- The mathematical inputs and outputs of the laws specified in the rules of Boolean algebra are represented by the binary number (1/0) system.
- Logical gates that make up the hardware of computer systems
    - AND, OR, NOT, NAND, NOR, XOR, …
- Logic gates are created using transistors.
    - NOT gate can be implemented by a single transistor
    - AND gate requires 3 transistors
- Transistors are the basic circuit elements of computer systems.
    - Pentium consists of 3 million transistors
    - Compaq Alpha consists of 9 million transistors
    - Now we can build chips with more than 100 million transistors

Design Hierarchy

Many digital systems can be divided into three design levels that form a well-defined hierarchy:

- The Architecture Level: High-level concerned with overall system management
- The Logic Level: Intermediate level concerned with the technical details of the system
- The Physical Level: Low level concerned with the details needed to manufacture or assemble the system
- We have already studied the architecture level
- Now we will address the logic level
- At the logic level, there are two classes of digital system
  - Combinational - digital systems without memory
  - Sequential - digital systems with memory

Boolean Algebra (Mantıksal devre)

Aa logic variable x can have only one of two possible values or states

- x = TRUE
- x = FALSE

In binary notation, we can say

- x = TRUE = 1
- x = FALSE = 0
- This is called positive logic or high-true logic

Electrically, – 1 is represented by a more positive voltage than zero and – 0 is represented by zero volts

- x = TRUE = 1 = 5 volts
- x = FALSE = 0 = 0 volts

Boolean Algebra

- Sayı sistemi, ikili olduğundan tüm değişkenler 0 ya da 1 değerini alır.
- Ve (*), veya (+), değil (tersi) kavramları üzerine oturmaktadır. Aritmetiksel işlem yoktur.
- Veya
    - 0 veya (+) 0=0
    - 1 veya (+) 0=1
    - 0 veya (+) 1=1
    - 1 veya (+) 1=1
- Ve
    - 0 ve (*) 0 = 0
    - 0 ve (*) 1 = 0
    - 1 ve (*) 0 = 0
    - 1 ve (*) 1 = 1
- Değil
    - 0 tersi 1
    - 1 tersi 0

Boolean Algebra Theorems

1. a) $a+b=b+a$ Commutativity

   b) $a \cdot b = b \cdot a$

2. a) $a+b+c=a+(b+c)$ Merger Feature (Birleşme)

   b) $a \cdot b \cdot c = a \cdot (b \cdot c)$

3. a) $a+b \cdot c=$ $(a+b) \cdot (a+c)$ Dispersion Feature (Dağılma)

   b) $a \cdot (\ b+c\ )=\ a \cdot b + a \cdot c$

   c) a(b+c)=ab+ac

4. a) $a+a=a$ Idempotency(Değişkende Fazlalık Özelliği)

   b) $a \cdot a = a$

5. a) $a+a.b=a$ Swallowing Feature (Yutma)

   b) $a \cdot (a+b) = a$

6. a) (a)^n $=a$ Redundancy Feature in transaction (işlemde Fazlalık Özelliği)

   b) (a x n) $=a$

7. a) $\overline{(a+b)}=\bar{a} \cdot \bar{b}$ De Morgan Rule

8. a) $0+a=a$ Ineffectiveness Feature (Etkisizlik Özelliği)

   b) $1 \cdot a = a$

9. a) $a+\bar{a}$ =1 Fixed Feature (Sabit Özelliği)

   b) $a \cdot \bar{a}$ =0

10. a) $1+a$=1 Devourer Fixed Feature (Yutan Sabit Özelliği)

    b) $0 \cdot a$=0

11. a) $(a+b\ ) \cdot b = a \cdot b$

    b) $a \cdot b + b = a + b$

12. a) $a+b \cdot a+c \cdot b+c = a+b \cdot (a+c)$

) b) $a \cdot b + a \cdot c + b \cdot c = a \cdot b + a \cdot c$

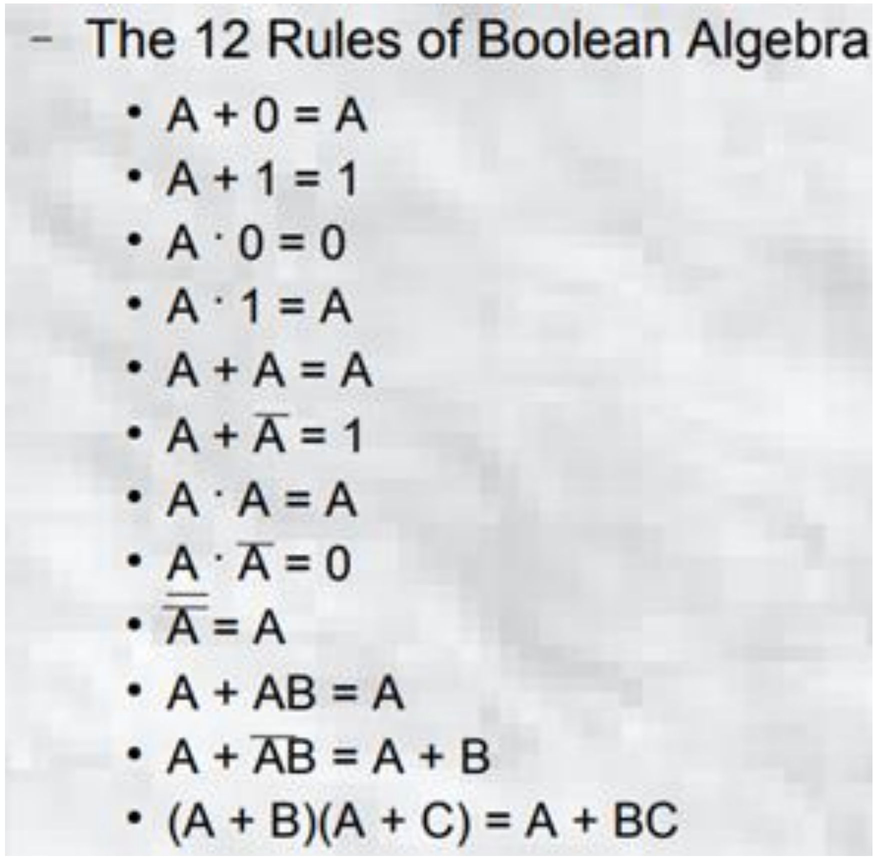
13. a) $a+b \cdot a+c = a \cdot c + a \cdot b$

    b) $a \cdot b + a \cdot c = a+c \cdot (a+b)$

14. a) $f\ a,b,c,d,\cdots = [a+f(0,b,c,d,\cdots)] \cdot [a+f(1,b,c,d,\cdots)]$ Shannon Teoremi

    b) $f\ a,b,c,d,\cdots = a \cdot f\ 1,b,c,d,\cdots + [a \cdot f(0,b,c,d,\cdots)]$

- **Boolean Algebra**: rules for rewriting Boolean functions
  - Useful for simplifying Boolean functions
    - Simplifying = reducing gate count, reducing gate "levels"
  - Rules: similar to logic (0/1 = F/T)
    - **Identity**: A1 = A, A+0 = A
    - **0/1**: A0 = 0, A+1 = 1
    - **Inverses**: (A')' = A
    - **Idempotency**: AA = A, A+A = A
    - **Tautology**: AA' = 0, A+A' = 1
    - **Commutativity**: AB = BA, A+B = B+A
    - **Associativity**: A(BC) = (AB)C, A+(B+C) = (A+B)+C
    - **Distributivity**: A(B+C) = AB+AC, A+(BC) = (A+B)(A+C)
    - **DeMorgan's**: (AB)' = A'+B', (A+B)' = A'B'



– The 12 Rules of Boolean Algebra

- $A + 0 = A$
- $A + 1 = 1$
- $A \cdot 0 = 0$
- $A \cdot 1 = A$
- $A + A = A$
- $A + \overline{A} = 1$
- $A \cdot A = A$
- $A \cdot \overline{A} = 0$
- $\overline{\overline{A}} = A$
- $A + AB = A$
- $A + \overline{A}B = A + B$
- $(A + B)(A + C) = A + BC$

Rules and Laws of Boolean Algebra

- Operations on Boolean variables are defined by rules and laws, the most important of which are presented here
- Commutative Law

  A . B = B . A

  A + B = B + A
- This states that the order of the variables is unimportant
- Associative Law

  A . (B . C) = A . (B . C)

  A + (B + C) = A + (B + C)
- This states that the grouping of the variables is unimportant
- Distributive Law: A . (B + C) = A . B + A . C
- This states that we can remove the parenthesis by ‘multiplying through’
- The above laws are the same as in ordinary algebra, where ‘+’ and ‘.’ are interpreted as addition and multiplication

Rules and Laws of Boolean Algebra

- Basic rules involving one variable:

$A + 0 = A$	$A . 0 = 0$
$A + 1 = 1$	$A . 1 = A$
$A + A = A$	$A . A = A$
$A + A' = 1$	$A . A' = 0$

- It should be noted that $A'' = A$
- An informal proof of each of these rules is easily accomplished by taking advantage of the fact that the variable can have only two possible values
- For example, rule 2: $A + 1 = 1$

  If $A = 0$ then $0 + 1 = 1$

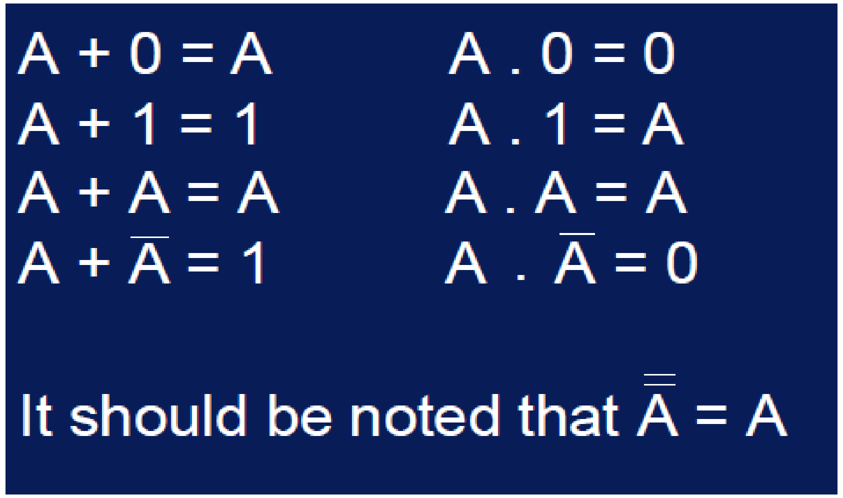
  If $A = 1$ then $1 + 1 = 1$

Rules and Laws of Boolean Algebra

Basic rules of single variable

- A proof of each of the rules and laws of Boolean algebra can be easily proved by taking advantage of the fact that a variable can only have two bits (0/1) of value.
- Note: A=0 or 1.
- A +A+A+A+A....+A+ 1 = 1 ; In the OR gate, if any of the inputs is 1, the output is one. The other method of proof is to search for accuracy by giving values of 1 and 0.
- A+A+A+ ... + A=A (Why? Bivariate 0 or 1 inputs are available)
- AAA ... A=A
  - If A = 0 then 0 + 1 = 1
  - If A = 1 then 1 + 1 = 1

Rules and Laws of Boolean Algebra



$A + 0 = A$	$A . 0 = 0$
$A + 1 = 1$	$A . 1 = A$
$A + A = A$	$A . A = A$
$A + \overline{A} = 1$	$A . \overline{A} = 0$

It should be noted that $\overline{\overline{A}} = A$

In logic circuits and mathematics, there are two numbers: 0 and 1.

Question: Perform the following operation using Boolean algebra. A=9, A+1=?

a)0 b)1 c)10 d) 8 e)none

Question: What values does A take in Boolean algebra?

a) 0 b)1 c) 0/1 d) 0,1,2, …, 9 d)Any value e)No value

Question: If A=1 in Boolean algebra, A+A+A+A+A=?

A)1 B)0 C)A D)5 D)5A E) None

Question: In Boolean Algebra, A*A*A*A=? A)A B)A^4

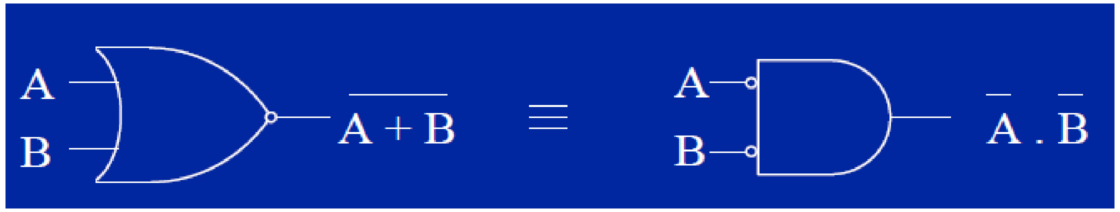
DeMorgan’s Laws

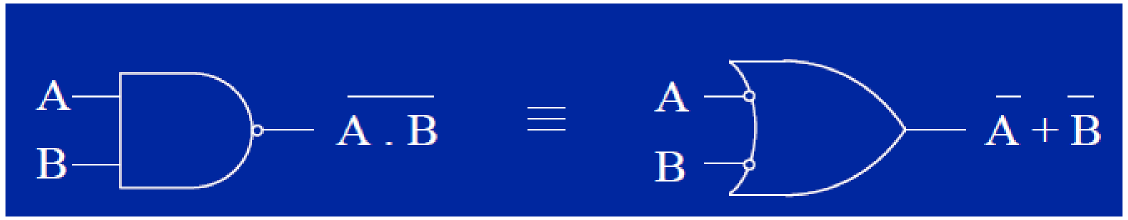
$$\overline{A + B} = \overline{A} \,.\, \overline{B}$$

$$\overline{A \,.\, B} = \overline{A} + \overline{B}$$

A	B	$A+B$	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}\,.\,\overline{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

A	B	$A\,.\,B$	$\overline{A\,.\,B}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0



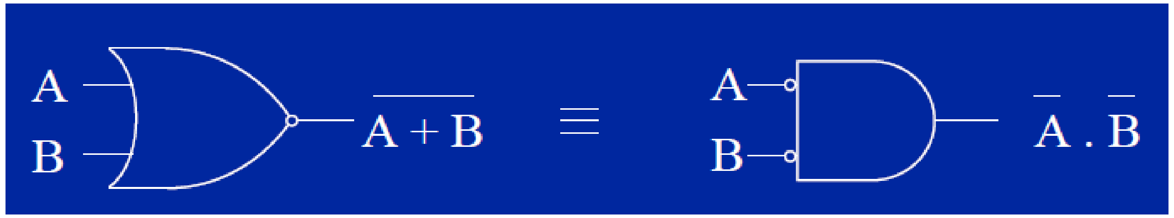


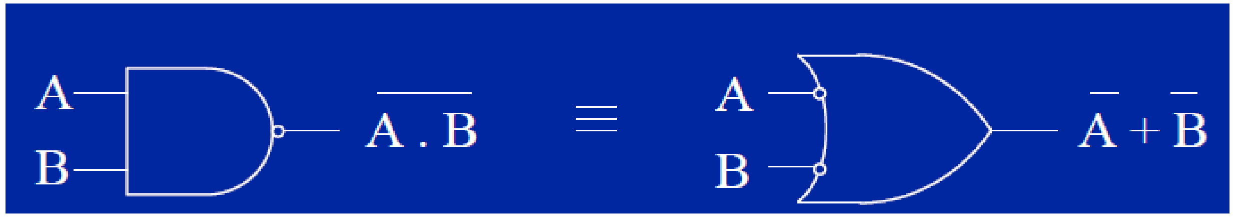
Rules and Laws of Boolean Algebra

De Morgan's Laws are particularly useful when dealing with NAND and NOR logic.

$$\overline{A + B} = \overline{A} \, . \, \overline{B}$$

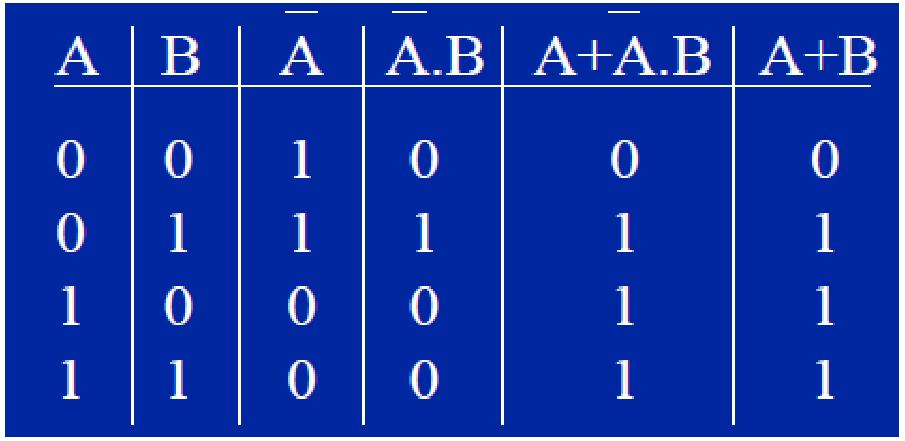
$$\overline{A \, . \, B} = \overline{A} + \overline{B}$$





Some useful theorems

- A+A.B=A.(1+B)=A,   1+B=1
- A+A’.B=(A+A’).(A+B)=A+B,   A+A’=1
- A.B+AB’=A(B+B’)=A,   B+B’=1
- A.(A+B)=A.A+AB=A+AB=A(1+B)=A,   A.A=A, 1+B=1
- A(A’+B)=A.A’+AB=AB;A.A’=0
- (A+B)(A+B’)=AA+AB’+AB+BB’=A+AB’+AB=A(1+B+B’)=A;   A.A=1, B.B’=0, 1+B+B’=1
- A + A’.B = A+B



A	B	$\bar{A}$	$\bar{A}.B$	$A+\bar{A}.B$	$A+B$
0	0	1	0	0	0
0	1	1	1	1	1
1	0	0	0	1	1
1	1	0	0	1	1

The result of logical operations is always 1 or 0.

- a + a + a + a + ... + a =a
- a*a*a* .... *a =a
- 1+a+b+c+ ... + z=1
- ab’c + ab’c=ab’c; The sum of similar expressions always equals a similar one.

Boolean Algebra

- Developed by George Boole in 19th Century
  - Algebraic representation of logic
    - Encode “True” as 1 and “False” as 0

And

- **A&B = 1 when both A=1 and B=1**

&	0	1
0	0	0
1	0	1

Or

- **A|B = 1 when either A=1 or B=1**

|	0	1
0	0	1
1	1	1

Not

- **~A = 1 when A=0**

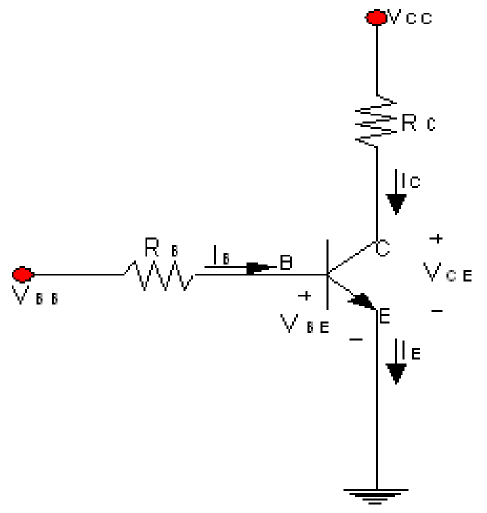
~	
0	1
1	0

Exclusive-Or (Xor)

- **A^B = 1 when either A=1 or B=1, but not both**

^	0	1
0	0	1
1	1	0

Gates and Transistors



$$V_{CC} = R_C * I_C + V_{CE}$$

$$V_{BB} = R_B * I_B + V_{BE}$$

$$I_C = \beta * I_B$$

$$I_{C\ SAT} = \frac{V_{CC}}{R_C}$$

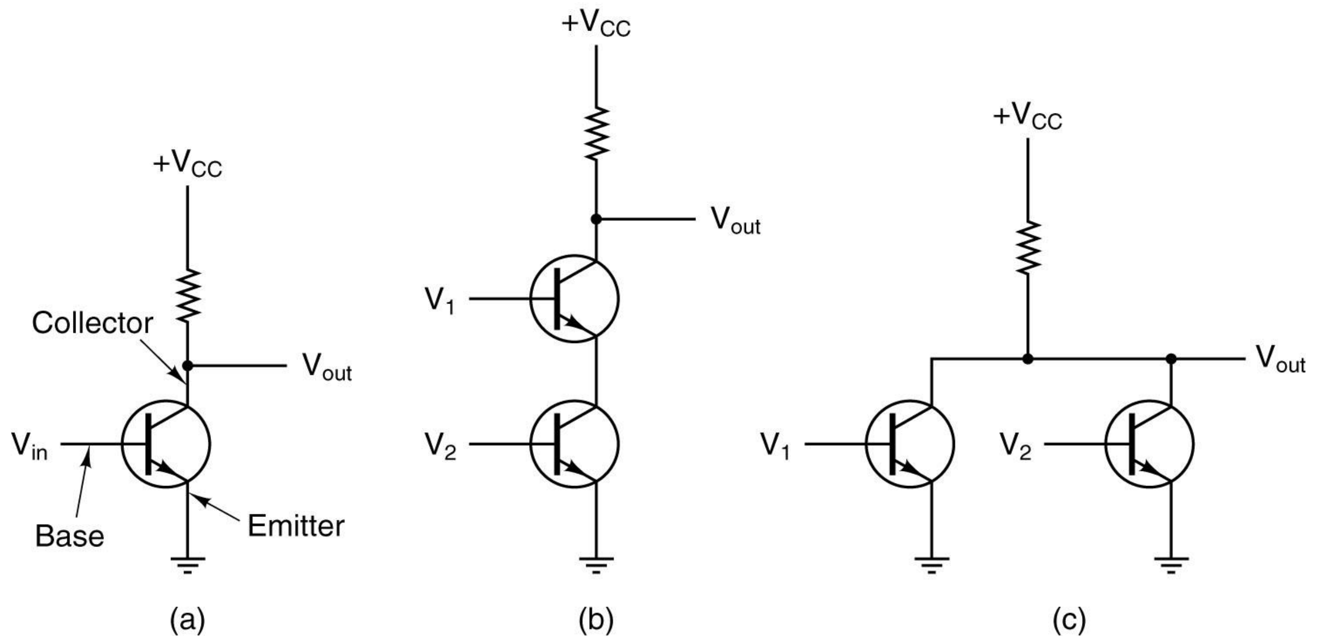
$$I_B = \frac{V_{BB} - V_{BE}}{R_B}$$

$V_{CE} \leq 0\ V$ *ya da* $I_c \geq I_{c\ SAT}$; *Saturasyon* $V_{CE} = 0V$ *Olur*

$I_B \leq 0\ A$ *ise* ; *Kesmede* $I_B = I_C = 0\ A$ *Olur*

VCE=VCC kesme durumunda

Transistor is a circuit element produced in semiconductor technology that controls the flow of electrons.



(a) A transistor inverter.

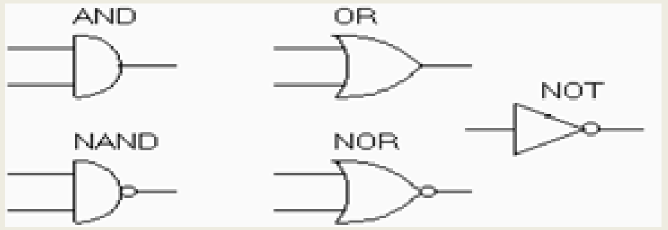
(b) A NAND gate.

(c) A NOR gate.

Transistor

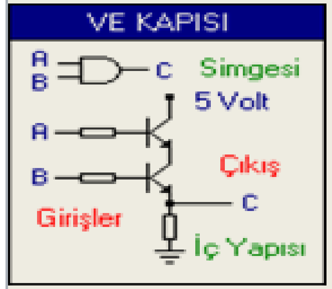
- Semiconductor circuit element that controls the flow of electrons.
- Subatomic particles (Quantum Mechanics): Proton, Neutron, Electron, Phototron
- Current is created from the flow of electrons.
- The transistor memory element stores the bit (0/1) state on it. Performs Switching. Or it strengthens the signal.
- Transistor is the most used electronic circuit element in the world.
- The smallest basic electronic circuit element of a microprocessor is the transistor.
- The CPU's basic function cycles occur at a dizzying speed as transistors turn on and off millions or even billions of times per second..

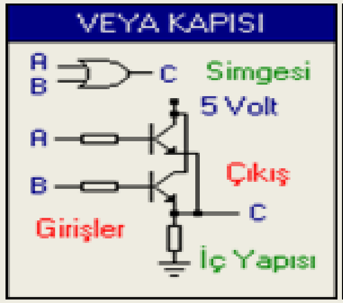
Gates

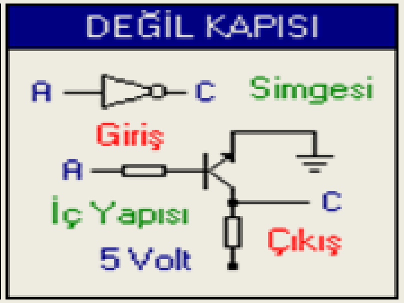


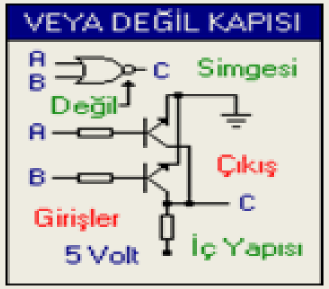
Doğruluk tablosu:

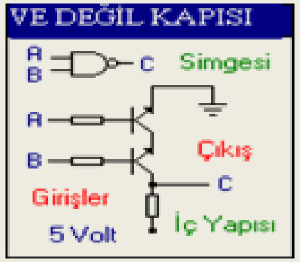
A	B	OR	AND	NOT	NOR	NAND	EXOR
		A+B	A*B	A'	(A+B)'	(A*B)'	(A')*B+A*(B')
0	0	0	0	1	1	1	0
0	1	1	0	1	0	1	1
1	0	1	0	0	0	1	1
1	1	1	1	0	0	0	0











Formüller	0 Değeri Verildiğinde	1 Değeri Verildiğinde
A . 0 = 0	A = 0 ise, 0 . 0 = 0	A = 1 ise, 1 . 0 = 0
A . 1 = A	A = 0 ise, 0 . 1 = 0	A = 1 ise, 1 . 1 = 1
A + 0 = A	A = 0 ise, 0 + 0 = 0	A = 1 ise, 1 + 0 = 1
A + 1 = A	A = 0 ise, 0 + 1 = 1	A = 1 ise, 1 + 1 = 1
A . A = A	A = 0 ise, 0 . 0 = 0	A = 1 ise, 1 . 1 = 1
A + A = A	A = 0 ise, 0 + 0 = 0	A = 1 ise, 1 + 1 = 1
A . A' = 0	A = 0 ise, 0 . 1 = 0	A = 1 ise, 1 . 0 = 0
A + A' = 1	A = 0 ise, 0 + 1 = 1	A = 1 ise, 1 + 0 = 1
(A')' = A	A = 0 ise, A' = 1, (A')' = 0	A = 1 ise, A' = 0, (A')' = 1

Sadeleştirmeler

(A + B) = (B + A)

A + B) + C = A + (B + C) = A + B + C

(A . B) . C = A . (B . C) = A . B . C

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

(A' . B) + (A . B') = A ⊕ B

(A + B)' = A' . B'

(A . B) = (B . A)

(A' . B') + (A . B) = (A ⊕ B)'

(A . B)' = A' + B'

Digital Logic Basics

- Hardware consists of a few simple building blocks
    - These are called *logic gates*
        - AND, OR, NOT, …
        - NAND, NOR, XOR, …
- Logic gates are built using transistors
        - NOT gate can be implemented by a single transistor
        - AND gate requires 3 transistors
- Transistors are the fundamental devices
        - Pentium consists of 3 million transistors
        - Compaq Alpha consists of 9 million transistors
        - Now we can build chips with more than 100 million transistors

Temel Kavramlar -1

- Number of functions
  - With N logical variables, we can define
    2^{2^N} functions
  - Some of them are useful
    - AND, NAND, NOR, XOR, …
  - Some are not useful:
    - Output is always 1
    - Output is always 0
  - “Number of functions” definition is useful in proving completeness property

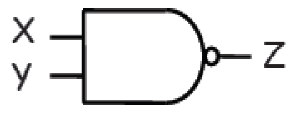
Temel Lojik Kapılar -1

- Simple gates
  - AND
  - OR
  - NOT
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination
- Precedence
  - NOT > AND > OR
  - F = A B + A B

    = (A (B)) + ((A) B)

Gate | **Symbol** | **Truth-Table** | **Expression**

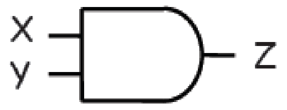
NAND



X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

$$Z = \overline{X \bullet Y}$$

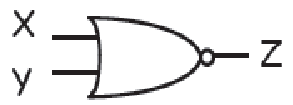
AND



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

$$Z = X \bullet Y$$

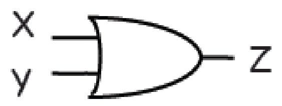
NOR



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

$$Z = \overline{X + Y}$$

OR



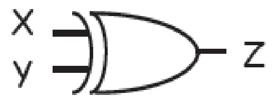
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

$$Z = X + Y$$

Temel Lojik Kapılar -2

- Additional useful gates
  - NAND
  - NOR
  - XOR
- NAND = AND + NOT
- NOR = OR + NOT
- XOR implements exclusive-OR function
- NAND and NOR gates require only 2 transistors
  - AND and OR need 3 transistors!

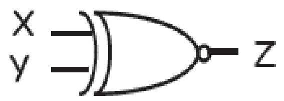
XOR
$(X \oplus Y)$



X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

$Z = X\overline{Y} + \overline{X}Y$
X or Y but not both
("inequality", "difference")

XNOR
$\overline{(X \oplus Y)}$



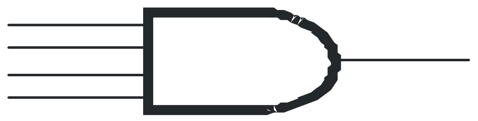
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

$Z = \overline{X}\,\overline{Y} + XY$
X and Y the same
("equality")

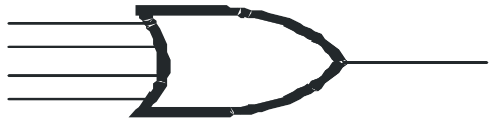
Widely used in arithmetic structures such as adders and multipliers

Basic Logic Functions

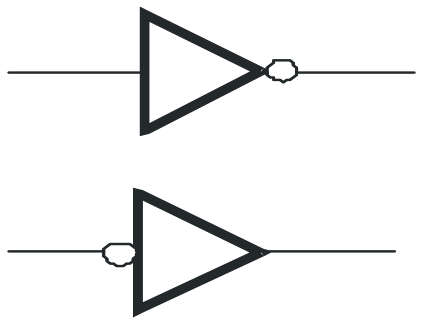
AND True only if ***all*** input conditions are true.



OR True only if ***one or more*** input conditions are true.



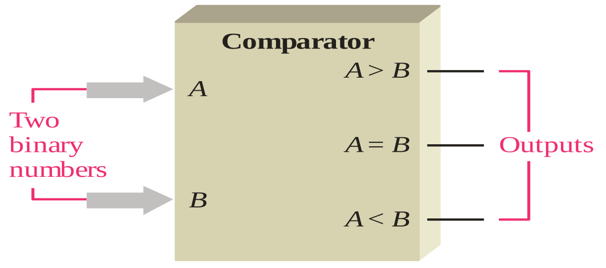
NOT Indicates the ***opposite*** condition.



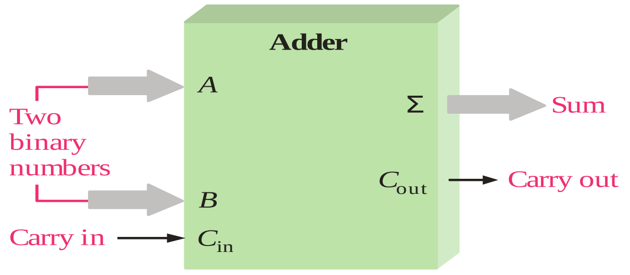
Basic System Functions

And, **or**, and **not** elements can be combined to form various logic functions. A few examples are:

The comparison function



Basic arithmetic functions



Logic Functions

- Logical functions can be expressed in several ways:
  - Truth table
  - Logical expressions
  - Graphical form
- Example:
  - Majority function
    - Output is one whenever majority of inputs is 1
    - We use 3-input majority function

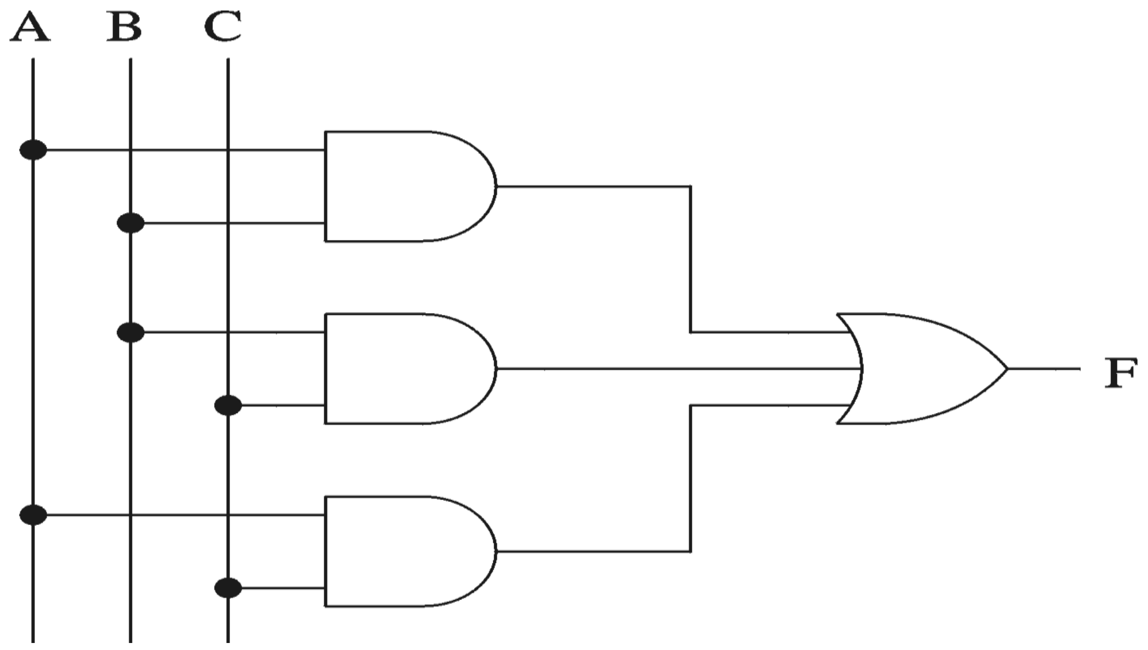
Logic Functions

3-input majority function

A	**B**	**C**	**F**	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	
1	0	0	0	
1	0	1	1	
1	1	0	1	
1	1	1	1	

- Logical expression form

$$F = A B + B C + A C$$



Boole Cebri Teoremleri

1. a) $a+b=b+a$ Değişme Özelliği

   b) $a \cdot b=b \cdot a$

2. a) $a+b+c= a+b +c=a+(b+c)$ Birleşme Özelliği

   b) $a \cdot b \cdot c= a \cdot b \cdot c=a \cdot (b \cdot c)$

3. a) $a+b \cdot c= a+b \cdot (a+c)$ Dağılma Özelliği

   b) $a \cdot b+c = a \cdot b +(a \cdot c)$

   c) a(b+c)=ab+ac

4. a) $a+a=a$ Değişkende Fazlalık Özelliği

   b) $a \cdot a=a$

5. a) $a+a.b=a$ Yutma Özelliği

   b) $a \cdot (a+b)=a$

6. a) $(a) =a$ işlemde Fazlalık Özelliği

   b) $(a) =a$

7. a) $(a+b+c+\cdots) =a \cdot b \cdot c \cdots$ De Morgan Kuralı

   b) $(a \cdot b \cdot c \cdots) =a +b +c +\cdots$

8. a) $a+a =1$ Sabit Özelliği

   b) $a \cdot a =0$

9. a) $0+a=a$ Etkisizlik Özelliği

   b) $1 \cdot a=a$

10. a) $1+a=1$ Yutan Sabit Özelliği

    b) $0 \cdot a=0$

11. a) $(a+b) \cdot b=a \cdot b$

    b) $a \cdot b +b=a+b$

12. a) $a+b \cdot a +c \cdot b+c = a+b \cdot (a +c)$

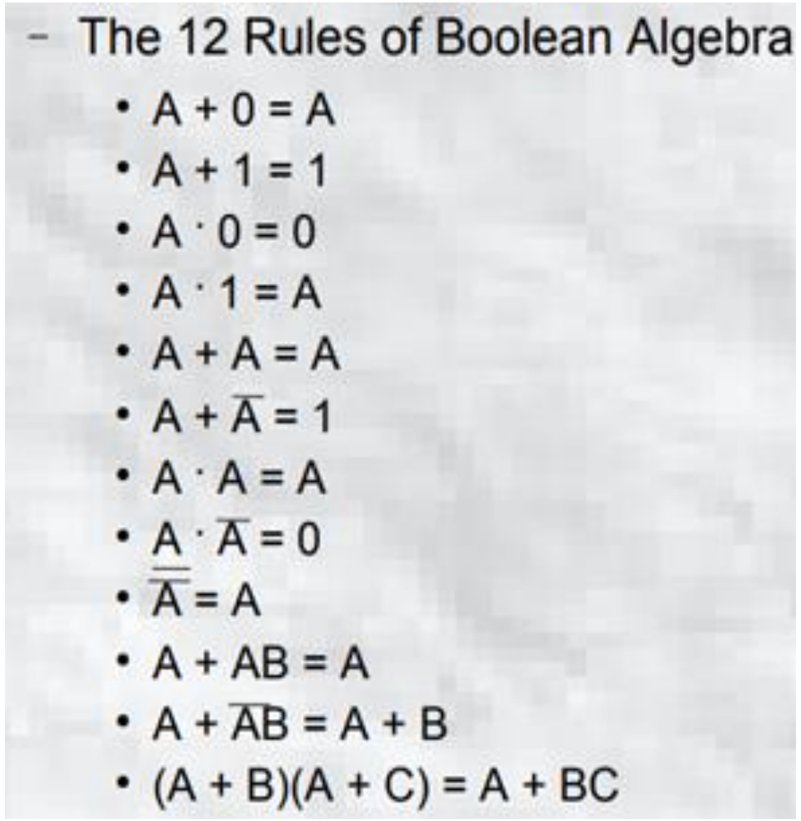
    b) $a \cdot b+a \cdot c+b \cdot c=a \cdot b+a \cdot c$

13. a) $a+b \cdot a +c =a \cdot c+a \cdot b$

    b) $a \cdot b+a \cdot c= a+c \cdot (a +b)$

14. a) $f\ a,b,c,d,\cdots =[a+f(0,b,c,d,\cdots)] \cdot [a +f(1,b,c,d,\cdots)]$ Shannon Teoremi

    b) $f\ a,b,c,d,\cdots = a \cdot f\ 1,b,c,d,\cdots +[a \cdot f(0,b,c,d,\cdots)]$



– The 12 Rules of Boolean Algebra

- $A + 0 = A$
- $A + 1 = 1$
- $A \cdot 0 = 0$
- $A \cdot 1 = A$
- $A + A = A$
- $A + \overline{A} = 1$
- $A \cdot A = A$
- $A \cdot \overline{A} = 0$
- $\overline{\overline{A}} = A$
- $A + AB = A$
- $A + \overline{A}B = A + B$
- $(A + B)(A + C) = A + BC$

Standard Forms

- Sum of Products (SOP)

$$F = \overline{A}\,\overline{B}C + A\overline{B}\,\overline{C} + A\overline{B}C + ABC$$

$$A\overline{B}(\overline{C} + C)$$

$$= A\overline{B}(1)$$

$$= A\overline{B}$$

$$AC(\overline{B} + B)$$

$$= AC$$

$$\overline{B}C(\overline{A} + A)$$

$$= \overline{B}C$$

$$F = \overline{B}C\cancel{(\overline{A} + A)} + A\overline{B}\cancel{(\overline{C} + C)} + AC\cancel{(\overline{B} + B)}$$

$$F = \overline{B}C + A\overline{B} + AC$$

Boolean Algebra

- We can use Boolean identities to simplify the function:

  as follows:

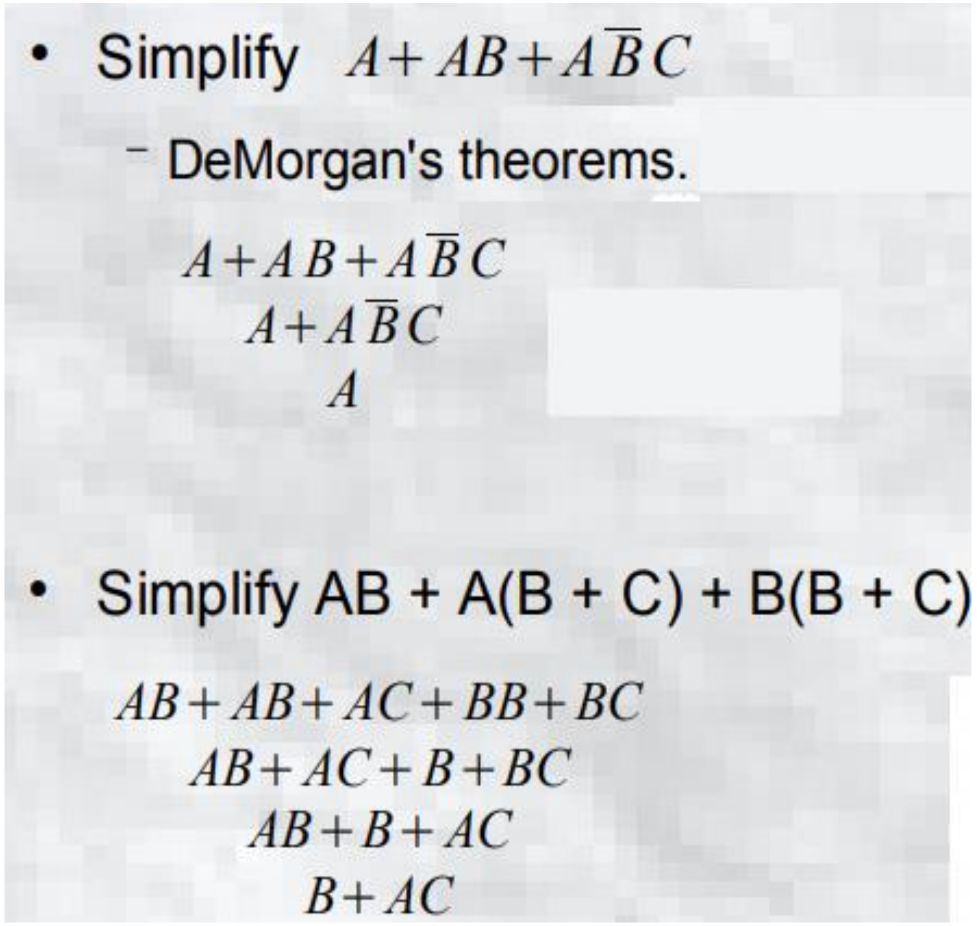
$$F(X,Y,Z) = (X+Y)(X+\overline{Y})(\overline{X\overline{Z}})$$

$(X+Y)(X+\overline{Y})(\overline{X\overline{Z}})$	Idempotent Law (Rewriting)
$(X+Y)(X+\overline{Y})(\overline{X}+Z)$	DeMorgan's Law
$(XX+X\overline{Y}+XY+Y\overline{Y})(\overline{X}+Z)$	Distributive Law
$((X+Y\overline{Y})+X(Y+\overline{Y}))(\overline{X}+Z)$	Commutative & Distributive Laws
$((X+0)+X(1))(\overline{X}+Z)$	Inverse Law
$X(\overline{X}+Z)$	Idempotent Law
$X\overline{X}+XZ$	Distributive Law
$0+XZ$	Inverse Law
XZ	Idempotent Law

Logic simplification

- Example:
- Z = A'BC + AB'C' + AB'C + ABC' + ABC
  - = A'BC + AB'(C' + C) + AB(C' + C) distributive
  - = A'BC + AB' + AB complementary
  - = A'BC + A(B' + B) distributive
  - = A'BC + A complementary
  - = BC + A absorption #2 Duality

  (X •Y')+Y=X+Y with X=BC and Y=A



- Simplify $A+AB+A\overline{B}C$
  - DeMorgan's theorems.

$$A+AB+A\overline{B}C$$
$$A+A\overline{B}C$$
$$A$$

- Simplify AB + A(B + C) + B(B + C)

$$AB+AB+AC+BB+BC$$
$$AB+AC+B+BC$$
$$AB+B+AC$$
$$B+AC$$

$$Y = A.B.C + A.\overline{B}.C + \overline{B}.\overline{C} + \overline{A}.\overline{B}$$

Using Boolean algebra:

$Y = A.B.C + A.\overline{B}.C + \overline{B}.\overline{C} + \overline{A}.\overline{B}$

$Y = A.B.C + A.\overline{B}.C + A.\overline{B}.\overline{C} + \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.\overline{B}.\overline{C}$ **(Expanding all terms by multiplying by 1. i.e. $(\mathrm{A} + \overline{\mathrm{A}})$)**

$Y = A.B.C + \overline{B}.(A.C + A.\overline{C} + \overline{A}.\overline{C} + \overline{A}.C + \overline{A}.\overline{C})$ **(Take out the common factor)**

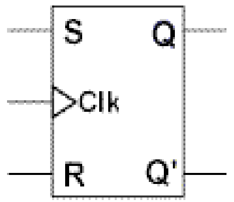
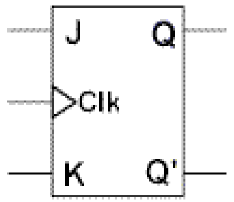
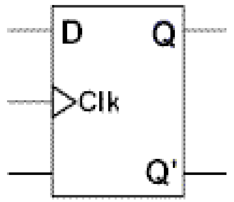
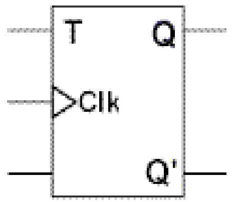
$Y = A.B.C + \overline{B}.\left(A.\left(C + \overline{C}\right) + \overline{A}.\left(\overline{C} + C + \overline{C}\right)\right)$ **(Group terms and take out the common factors)**

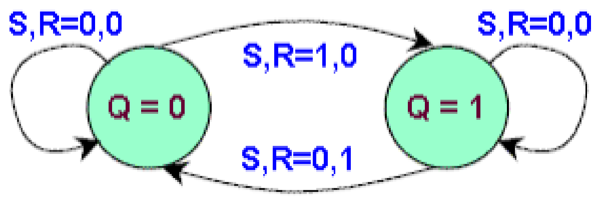
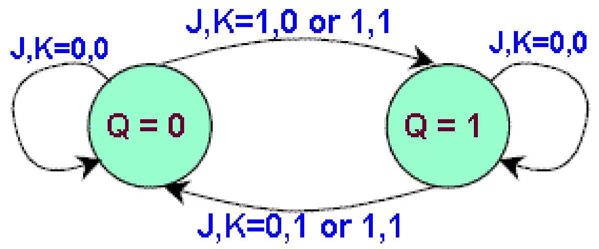
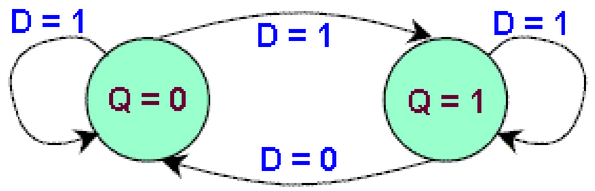
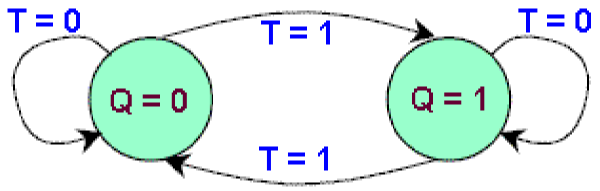
$Y = A.B.C + \overline{B}\left(A + \overline{A}\right)$ **(Simplify)**

$\underline{Y = A.B.C + \overline{B}}$

Sequential Logic

- Has memory; the circuit stores the result of the previous set of inputs. The current output depends on inputs in the past as well as present inputs.
    - The basic element in sequential logic is the bistable latch or flip-flop, which acts as a memory element for one bit of data.

FLIP-FLOP NAME	FLIP-FLOP SYMBOL	CHARACTERISTIC EQUATION	EXCITATION TABLE			
SR		**$Q_{(next)} = S + R'Q$** **$SR = 0$**	**Q**	**$Q_{(next)}$**	**S**	**R**
			0	0	0	X
			0	1	1	0
			1	0	0	1
			1	1	X	0
JK		**$Q_{(next)} = JQ' + K'Q$**	**Q**	**$Q_{(next)}$**	**J**	**K**
			0	0	0	X
			0	1	1	X
			1	0	X	1
			1	1	X	0
D		**$Q_{(next)} = D$**	**Q**	**$Q_{(next)}$**	**D**	
			0	0	0	
			0	1	1	
			1	0	0	
			1	1	1	
T		**$Q_{(next)} = TQ' + T'Q$**	**Q**	**$Q_{(next)}$**	**T**	
			0	0	0	
			0	1	1	
			1	0	1	
			1	1	0	

NAME	STATE DIAGRAM
SR	
JK	 J,K=1,0 or 1,1 J,K=0,0 J,K=0,0 Q = 0 Q = 1 J,K=0,1 or 1,1
D	
T	

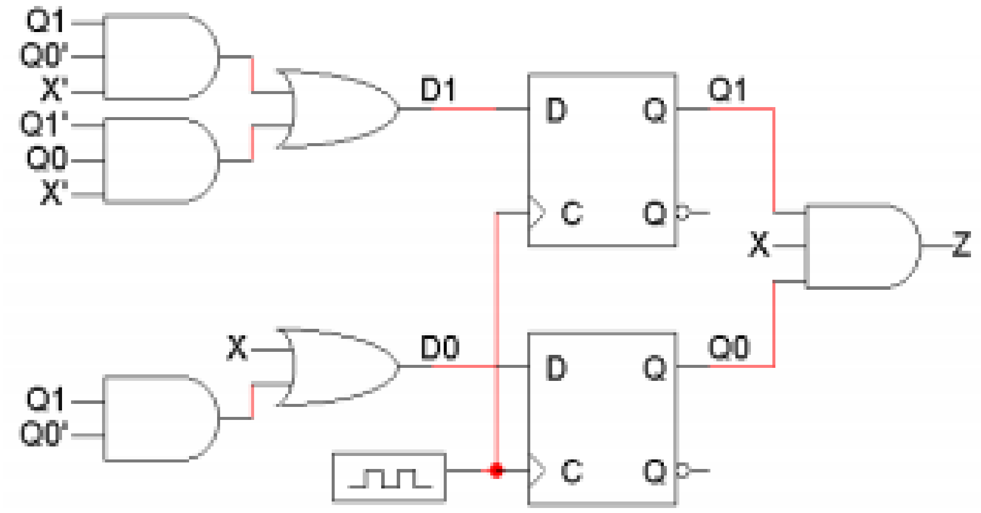
- D FF karakteristik tablosu:

Q(t)	Q(t+1)	D	İşlem
0	0	0	Reset
0	1	1	Set
1	0	0	Reset
1	1	1	Set

Şimdiki durum		Giriş	Gelecek durum		Flip flop girişleri		Çıkış
Q_1	Q_0	X	Q_1	Q_0	D_1	D_0	Z
0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	1	0	1	0	1	0	0
0	1	1	0	1	0	1	0
1	0	0	1	1	1	1	0
1	0	1	0	1	0	1	0
1	1	0	0	0	0	0	0
1	1	1	0	1	0	1	1

Karnaugh diyagramı yardımıyla aynı denklemleri bulabiliriz:

$$D_1 = Q_1\, Q_0'\, X' + Q_1'\, Q_0\, X'$$
$$D_0 = X + Q_1\, Q_0'$$
$$Z = Q_1\, Q_0\, X$$



Binary Logic

- ♣ Binary logic consists of binary variables and a set of logical operations.
- ♣ The variables are designated by letters of the alphabet, such as A, B, C, x, y, z, etc, with each variable having two and only two distinct possible values: 1 and 0.
- ♣ There are three basic logical operations: AND, OR, and NOT.

AND: represented by a dot or by the absence of an operator.

- $x \cdot y = z$ or $xy = z$ is read "x AND y is equal to z."
- $z = 1$ if and only if $x = 1$ and $y = 1$; otherwise $z = 0$. (Remember that x, y, and z are binary variables and can be equal either to 1 or 0, and nothing else.)

OR: represented by a plus sign.

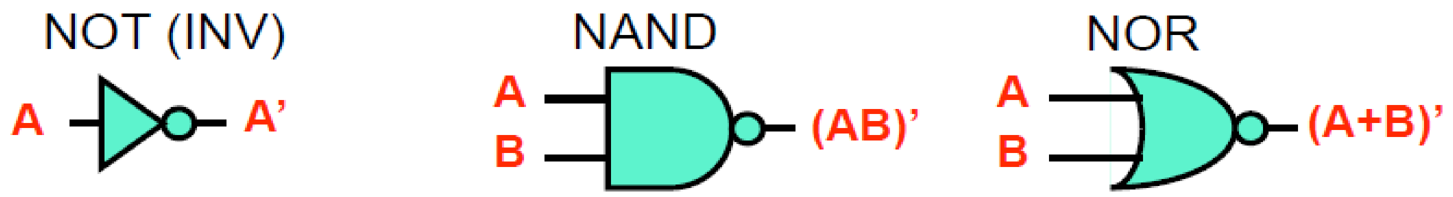
- $x + y = z$ is read "x OR y is equal to z," meaning that $z = 1$ if $x = 1$ or if $y = 1$ or if both $x = 1$ and $y = 1$.
- If both $x = 0$ and $y = 0$, then $z = 0$.

NOT: represented by a prime (sometimes by an overbar).

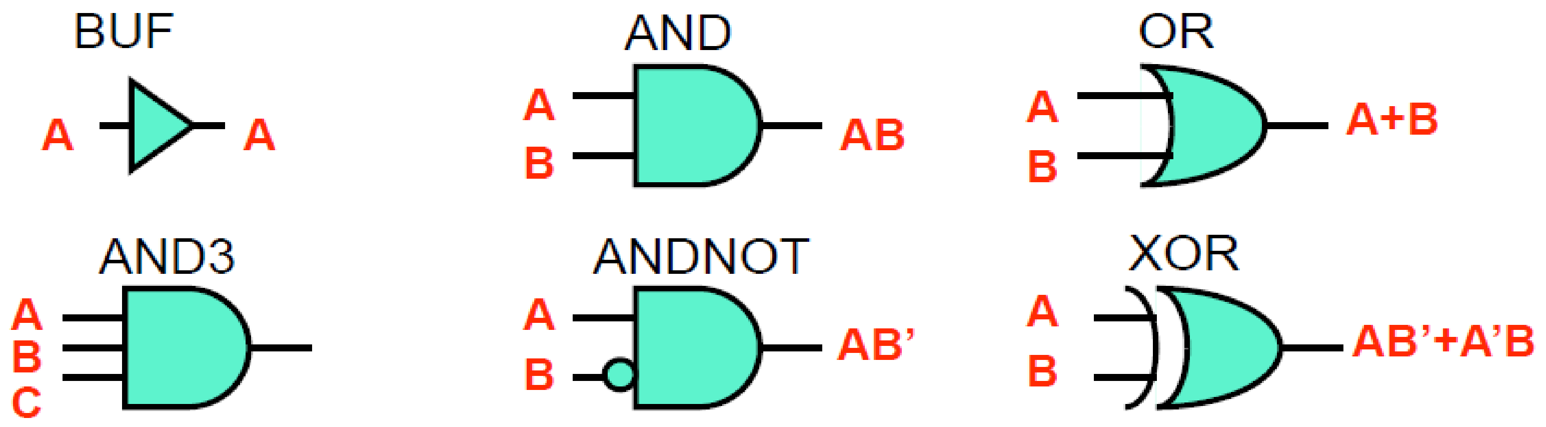
- $x' = z$ (or $x = \overline{z}$) is read "not x is equal to z," meaning that z is what x is not.
- If $x = 1$, then $z = 0$, but if $x = 0$, then $z = 1$.
- also referred to as the *complement* operation, since it changes a 1 to 0 and a 0 to 1.

Logic Gates

- **Logic gates:** implement Boolean functions
  - Basic gates: NOT, NAND, NOR
    - Underlying CMOS transistors are naturally inverting (● = NOT)

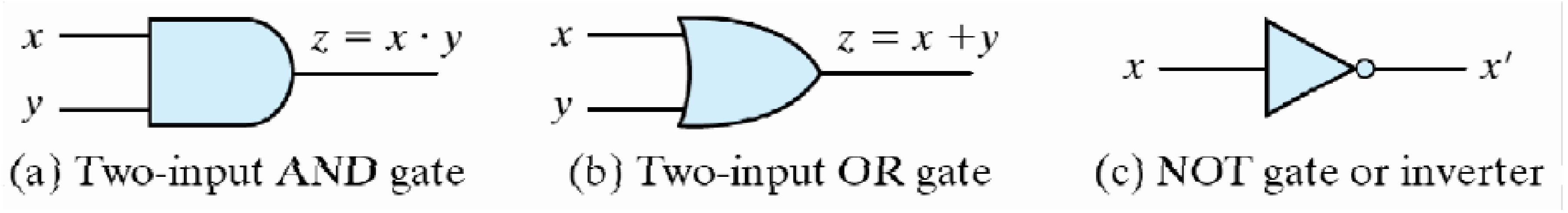


  - NAND, NOR are "Boolean complete"



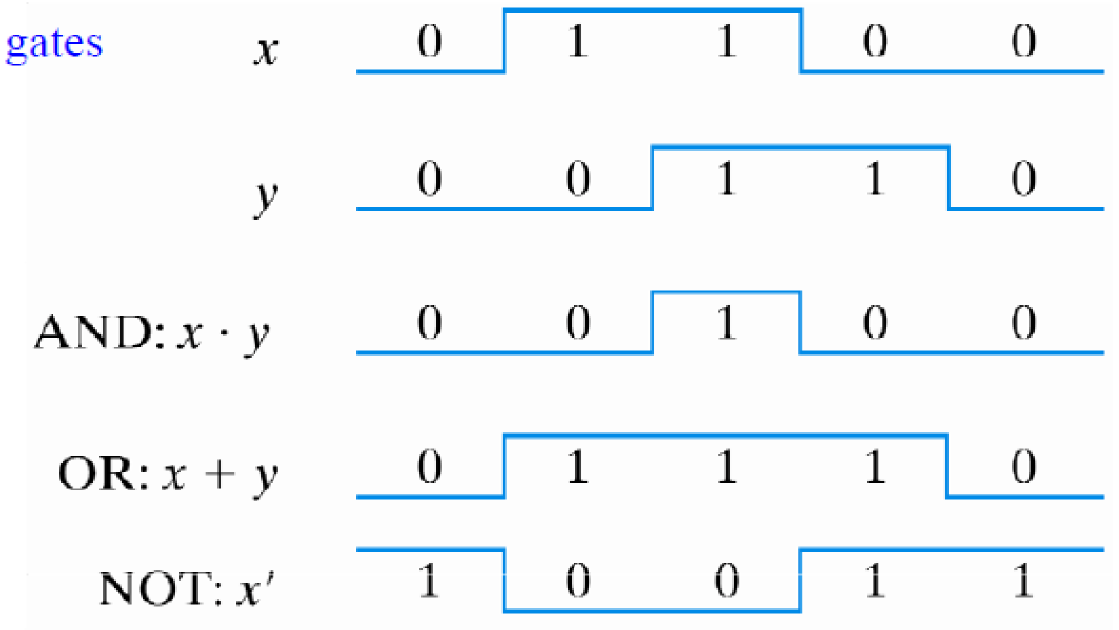
Basic Logic Gates

♣ Graphic Symbols and Input-Output Signals for Logic gates:



(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

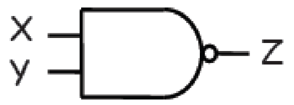
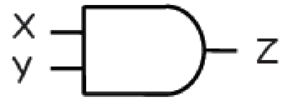
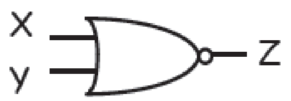
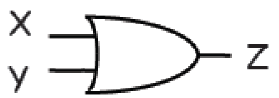
➢Input-Output signals for logic gates



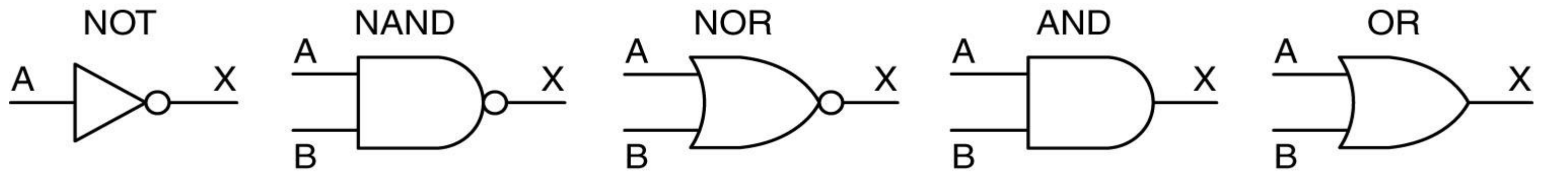
Temel Lojik Kapılar -1

- Simple gates
  - AND
  - OR
  - NOT
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination
- Precedence
  - NOT > AND > OR
  - F = A B + A B

    = (A (B)) + ((A) B)

Gate	Symbol	Truth-Table	Expression
NAND		X Y Z 0 0 1 0 1 1 1 0 1 1 1 0	$Z = \overline{X \bullet Y}$
AND		X Y Z 0 0 0 0 1 0 1 0 0 1 1 1	$Z = X \bullet Y$
NOR		X Y Z 0 0 1 0 1 0 1 0 0 1 1 0	$Z = \overline{X + Y}$
OR		X Y Z 0 0 0 0 1 1 1 0 1 1 1 1	$Z = X + Y$

Symbols and functional behavior for Logic Gates



A	X
0	1
1	0

(a)

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

(b)

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

(c)

A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

(d)

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

(e)

AND Gate: If any of the inputs is 0, the output is 0. If all inputs are 1, the output is 1.

OR Gate: If any of the inputs is 1, the output is 1. If all inputs are 0, the output is 0.

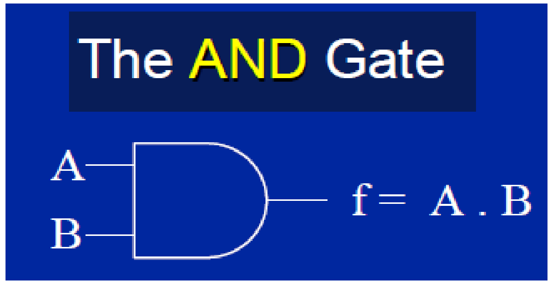
NOT Gate: transposes the input.

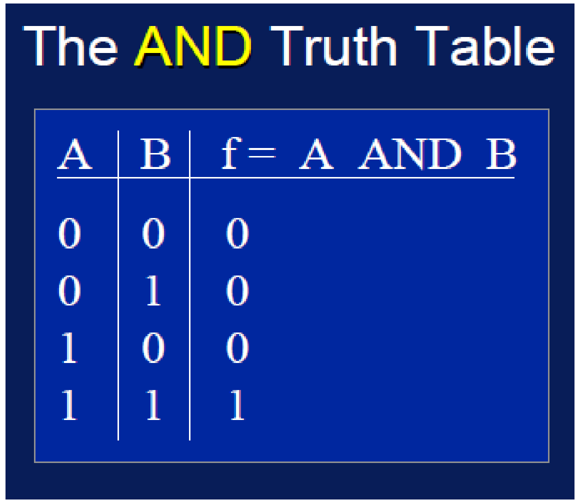
A logic gate is an idealized or physical circuit that implements a Boolean function, that is, it performs a logical operation on one or more logic inputs and produces a single logic output.

Logic AND Gates

- Logic gates are switching circuits that perform certain simple operations on binary signals
- These operations are chosen to facilitate the implementation of useful functions

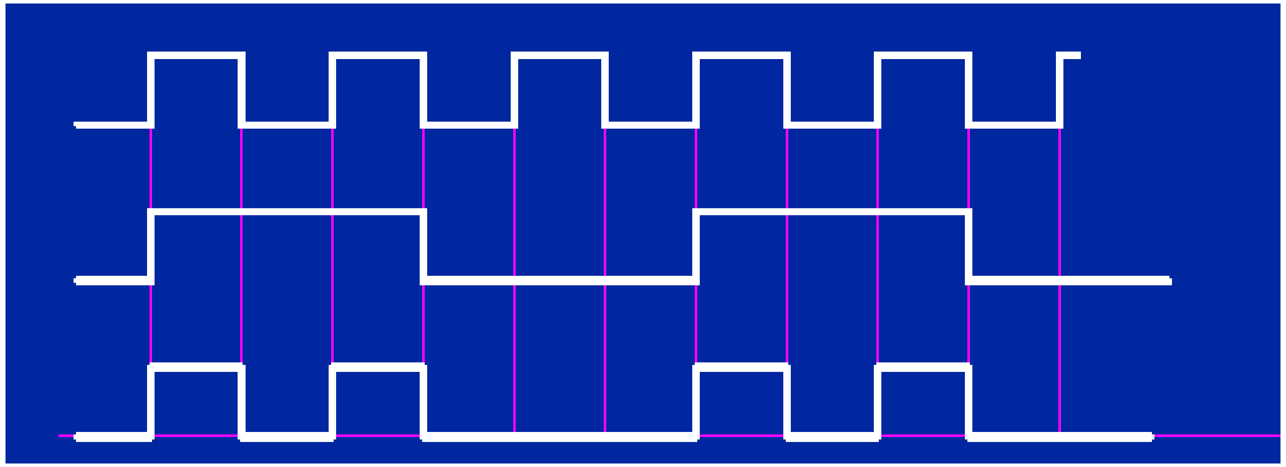
- The AND Gate - Determine the output waveform when the input waveforms A and B are applied to the two inputs of an AND gate
- A and B are variables and note the use of the . to denote AND
- Giriş dalga formları A ve B bir mantık kapısının iki girişine uygulandığında çıkış dalga formu belli ise bu kapının türünü belirleyiniz. (AND Kapısı)





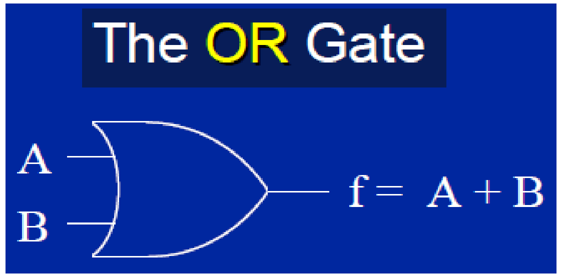
The AND Truth Table

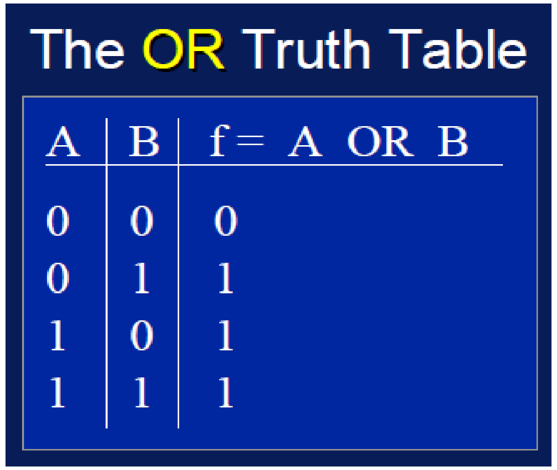
A	B	f= A AND B
0	0	0
0	1	0
1	0	0
1	1	1



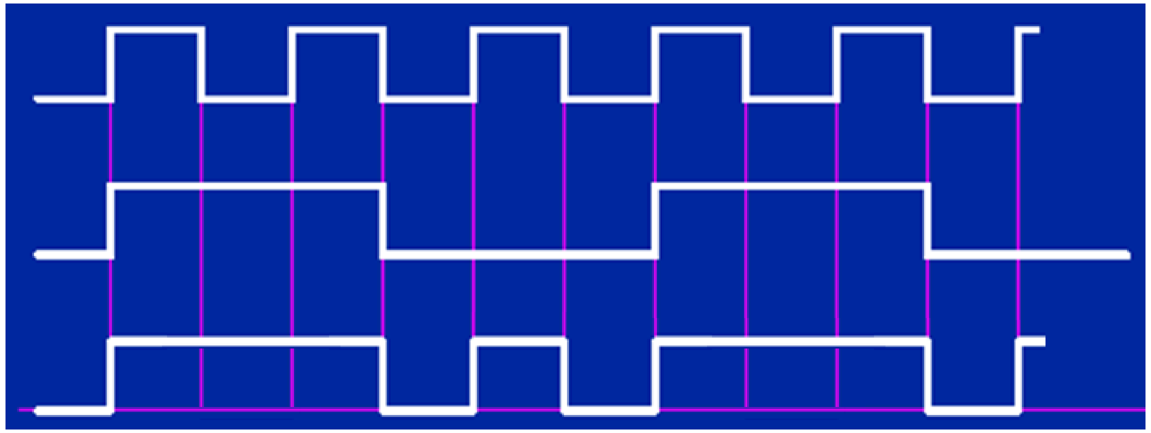
Logic OR Gates

- A and B are variables and note the use of the + to denote OR



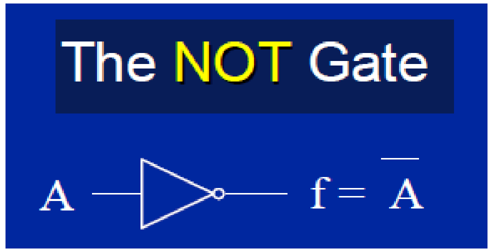

The OR Truth Table

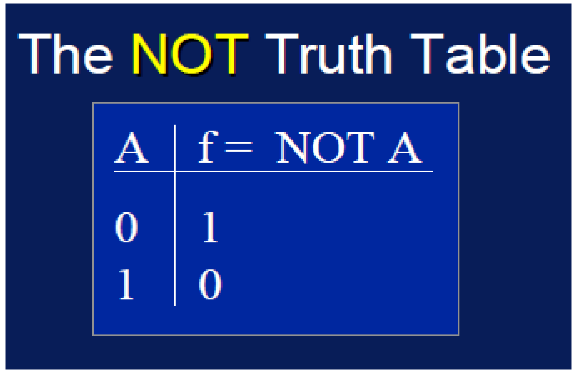
A	B	f = A OR B
0	0	0
0	1	1
1	0	1
1	1	1



Logic NOT Gates

- Note the use of the bar over the A to denote NOT

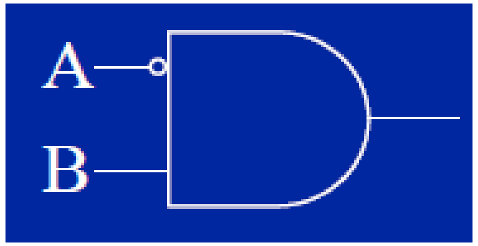


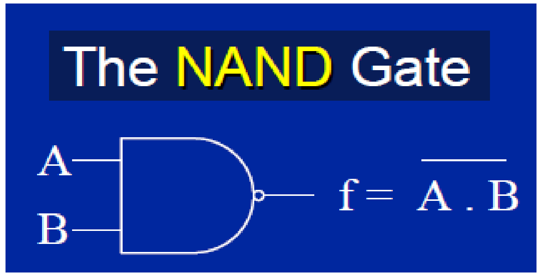

The NOT Truth Table

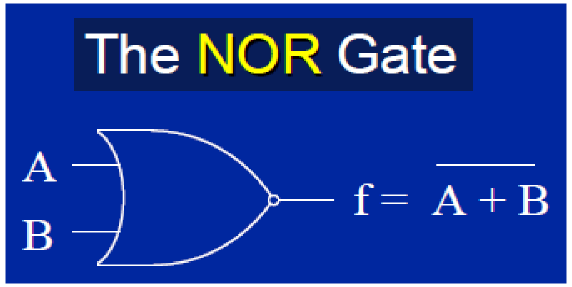
A	f = NOT A
0	1
1	0

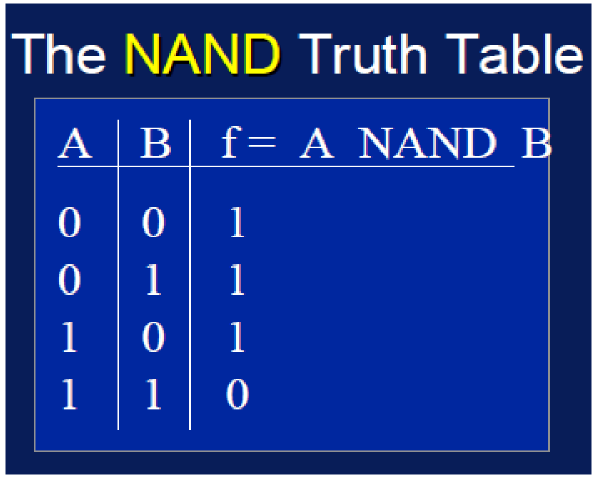
Logic Gates

- Sometimes a ‘bubble’ is used to indicate Inversion
- In fact it is simpler to manufacture the combination NOT AND and NOT OR than it is to deal with AND and OR
- NOT AND becomes NAND
- NOT OR becomes NOR



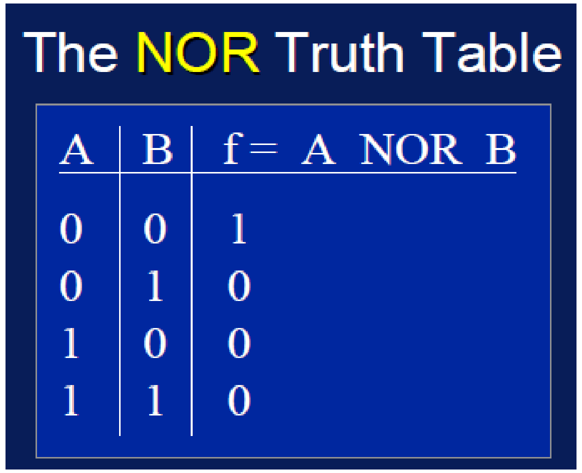






The NAND Truth Table

A	B	f = A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

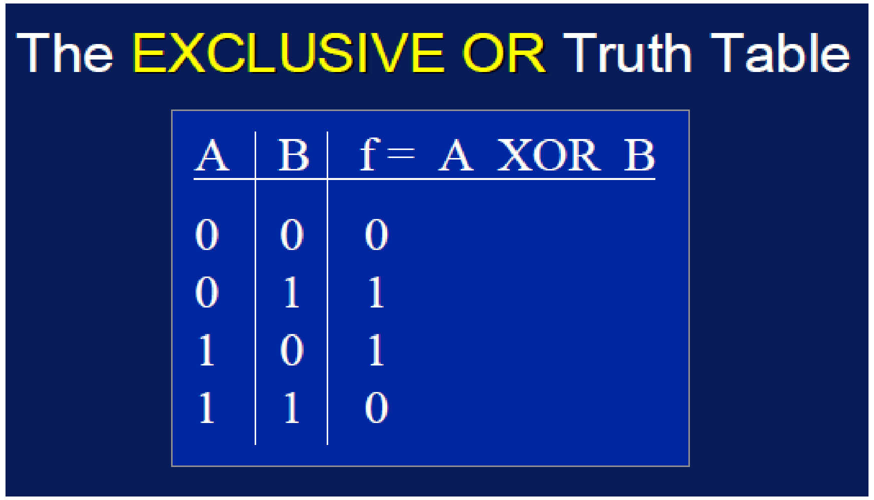


The NOR Truth Table

A	B	f = A NOR B
0	0	1
0	1	0
1	0	0
1	1	0

Logic Gates

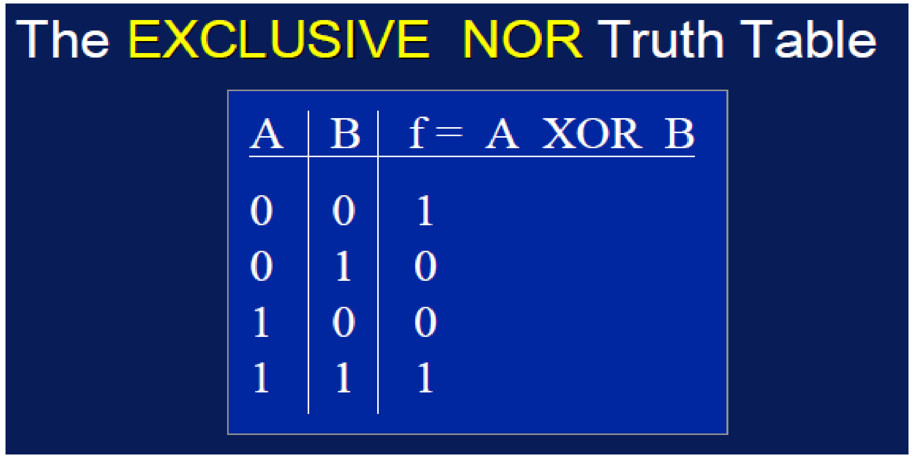
Toplama



The EXCLUSIVE OR Truth Table

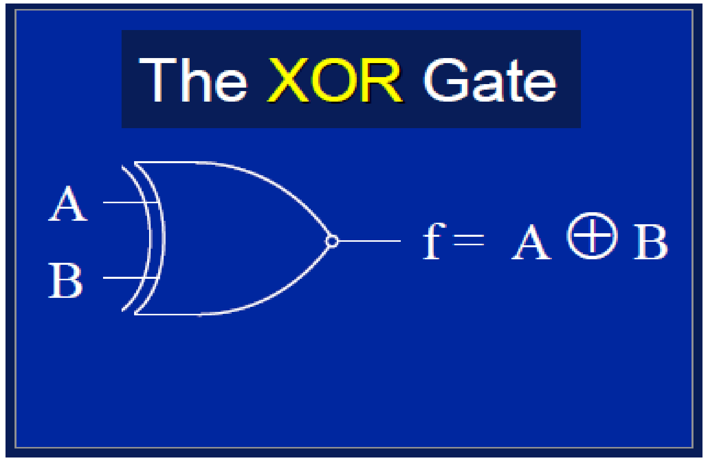
A	B	f= A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Karşılaştırma

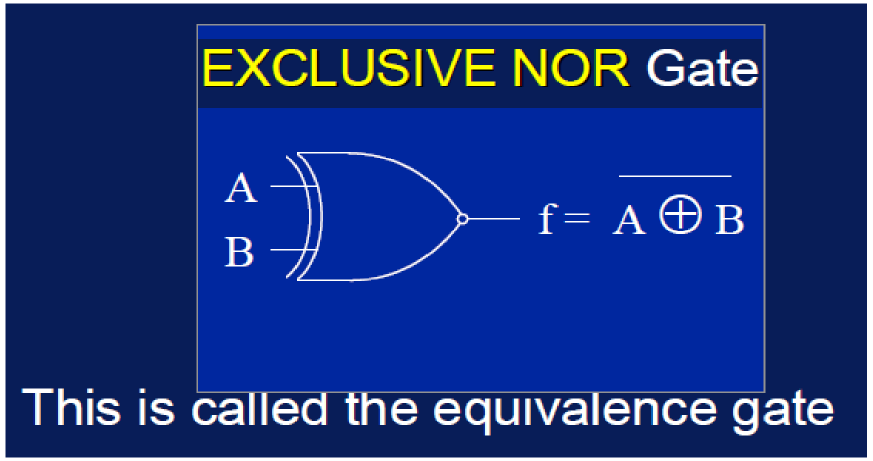


The EXCLUSIVE NOR Truth Table

A	B	f= A XOR B
0	0	1
0	1	0
1	0	0
1	1	1



The XOR Gate

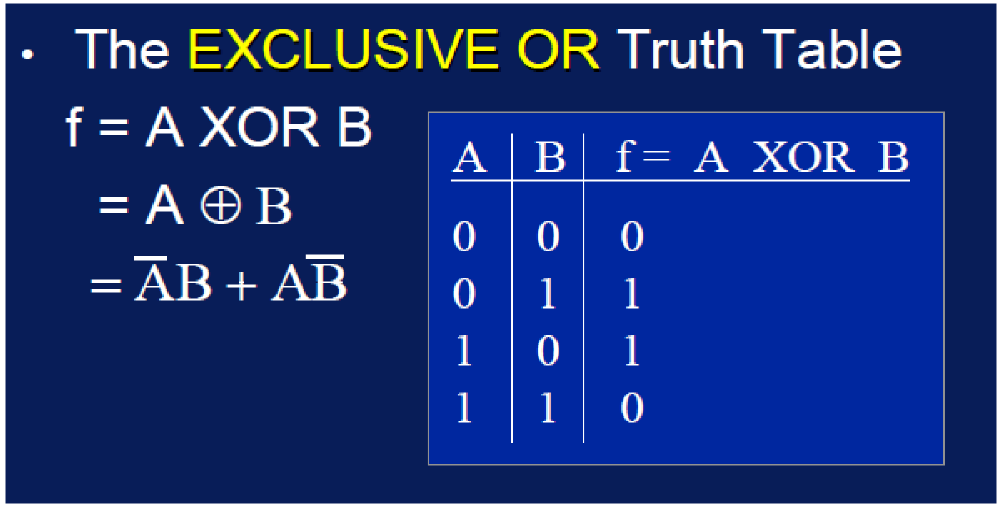


EXCLUSIVE NOR Gate

This is called the equivalence gate

XOR gates are used in comparison and arithmetic addition operations.If all inputs are equal (0 or 1) and the output is zero, it is an XOR gate; if the output is 1, it is an XNOR gate.

The XOR – XNOR Gates



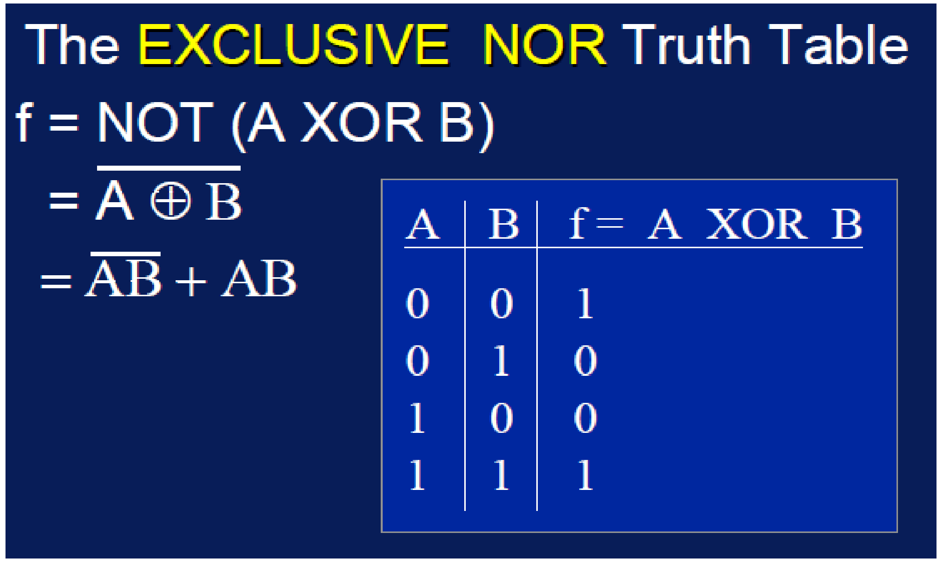
- The EXCLUSIVE OR Truth Table

f = A XOR B

$= A \oplus B$

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

A	B	f = A XOR B
0	0	0
0	1	1
1	0	1
1	1	0



The EXCLUSIVE NOR Truth Table

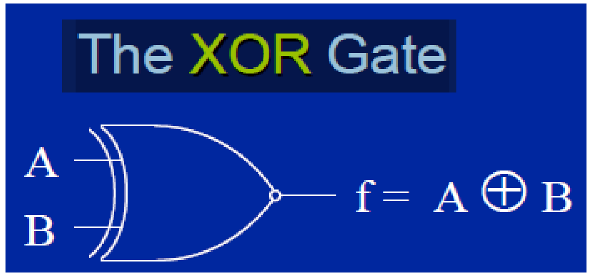
f = NOT (A XOR B)

$= \overline{A \oplus B}$

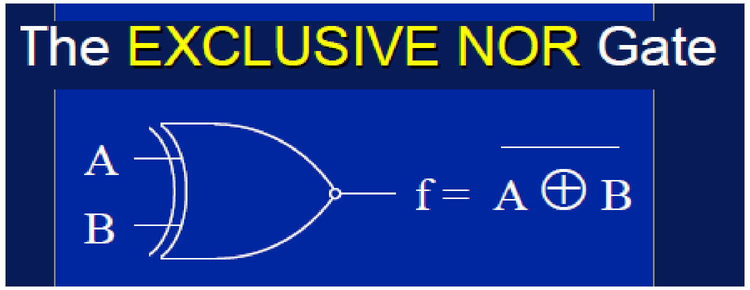
$= \overline{AB} + AB$

A	B	f = A XOR B
0	0	1
0	1	0
1	0	0
1	1	1

The XOR Gate



The EXCLUSIVE NOR Gate

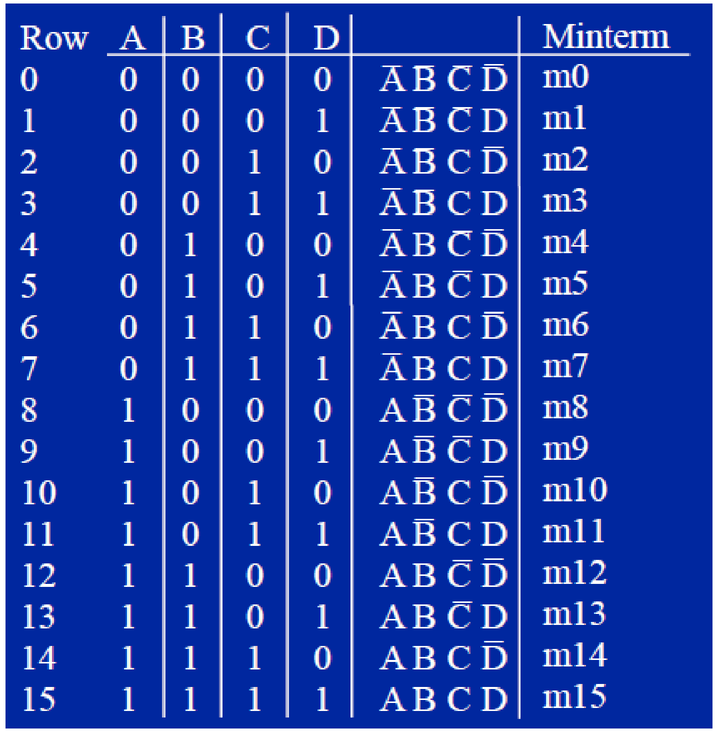


Örnek:

- Sound the alarm when A=1 and B=1 or C=1 and D=1.

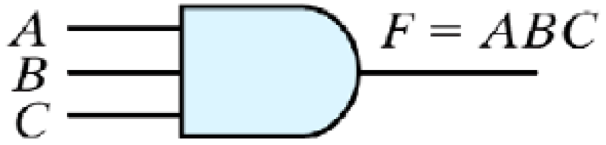
F=AB+CD

In case of AB: m12, m13, m14, m15In case of CD: m3,m7,m11, m15

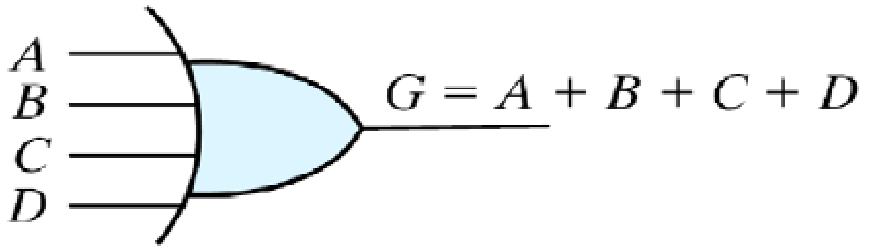


Row	A	B	C	D		Minterm
0	0	0	0	0	$\bar{A}\bar{B}\bar{C}\bar{D}$	m0
1	0	0	0	1	$\bar{A}\bar{B}\bar{C}D$	m1
2	0	0	1	0	$\bar{A}\bar{B}C\bar{D}$	m2
3	0	0	1	1	$\bar{A}\bar{B}CD$	m3
4	0	1	0	0	$\bar{A}B\bar{C}\bar{D}$	m4
5	0	1	0	1	$\bar{A}B\bar{C}D$	m5
6	0	1	1	0	$\bar{A}BC\bar{D}$	m6
7	0	1	1	1	$\bar{A}BCD$	m7
8	1	0	0	0	$A\bar{B}\bar{C}\bar{D}$	m8
9	1	0	0	1	$A\bar{B}\bar{C}D$	m9
10	1	0	1	0	$A\bar{B}C\bar{D}$	m10
11	1	0	1	1	$A\bar{B}CD$	m11
12	1	1	0	0	$AB\bar{C}\bar{D}$	m12
13	1	1	0	1	$AB\bar{C}D$	m13
14	1	1	1	0	$ABC\bar{D}$	m14
15	1	1	1	1	$ABCD$	m15

Multiple-Input Gates

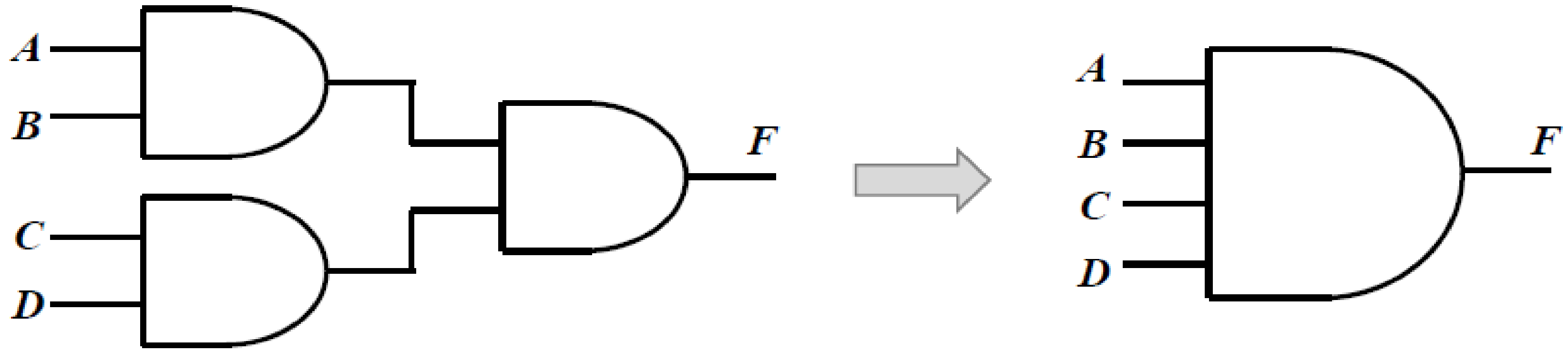


(a) Three-input AND gate



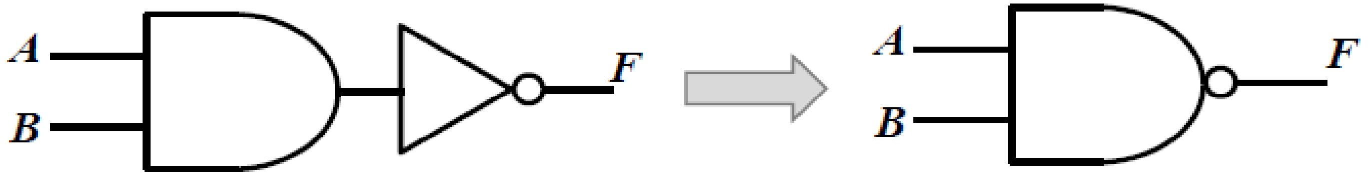
(b) Four-input OR gate

$F = (AB)(CD) = ABCD$



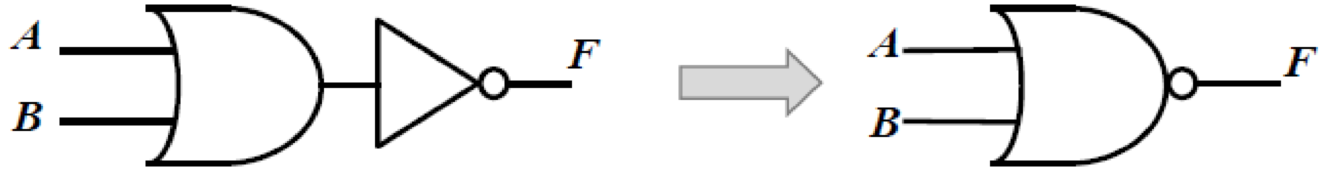
Inverting Gates

NOT + AND = NAND



A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

NOT + OR = NOR

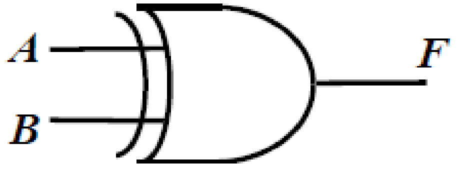


A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive OR/NOR Gates

XOR

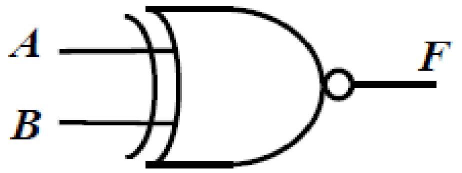
$F = A \oplus B$



A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

$F = A \odot B$



A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Truth Table of Logic Operation

Truth Tables of Logical Operations

AND

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

OR

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

NOT

x	x'
0	1
1	0

A logic variable is always either 1 or 0.

Logic Functions

Truth Table: 3-input majority function

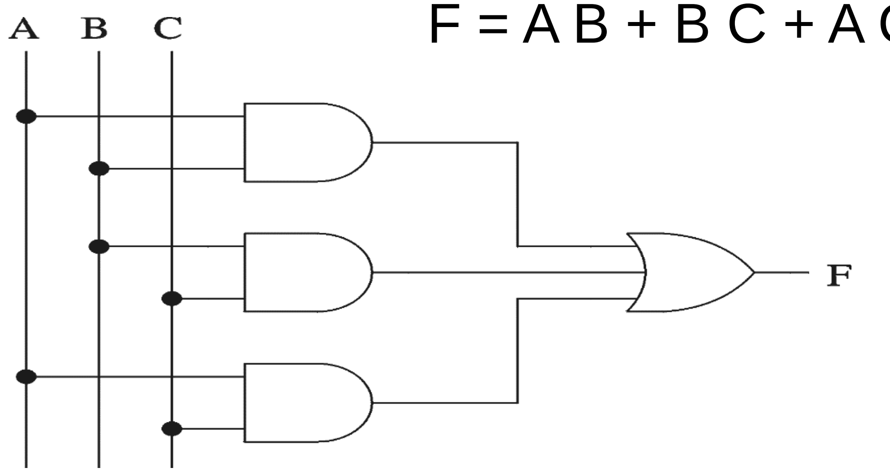
A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

- **Logical functions can be expressed in several ways:**
  - **Truth Table**
  - **Logical Expression**
  - **Impression**

- **Logical expression form**

F=A'BC+AB'C+ABC'+ABC

F = A B + B C + A C



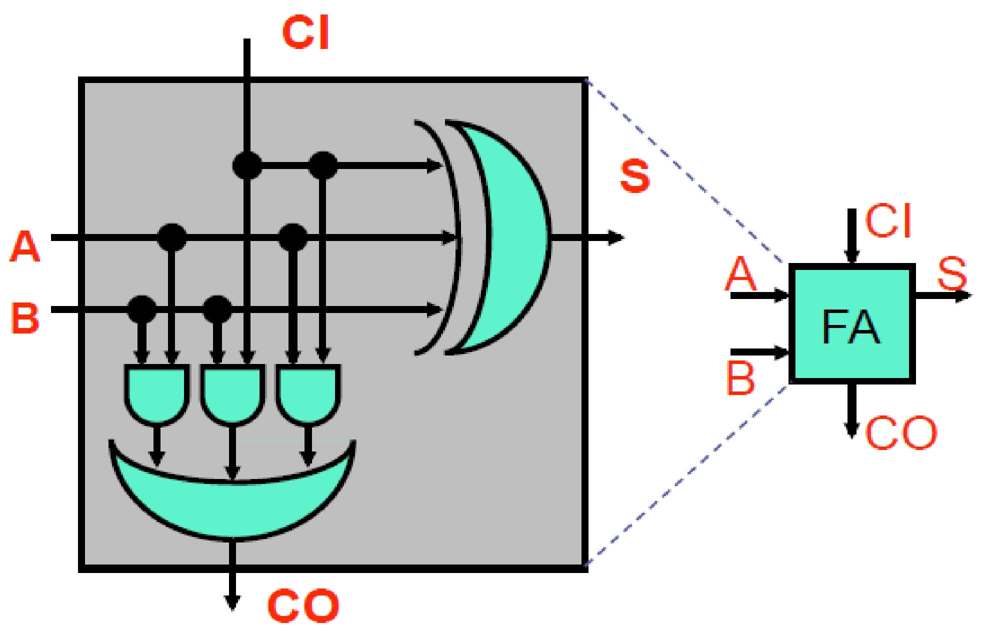
- The sum of the variables in the given equations gives the number of inputs.
- In logic gates, the result of a logical equation is 1 or 0.

If the number of entries is m, how many different states are there? There are S=2^m states. The reason why it has base 2 is due to the binary number system: 0 or 1, bit

Full Adder

- What is the logic for a full adder?
  - Look at truth table

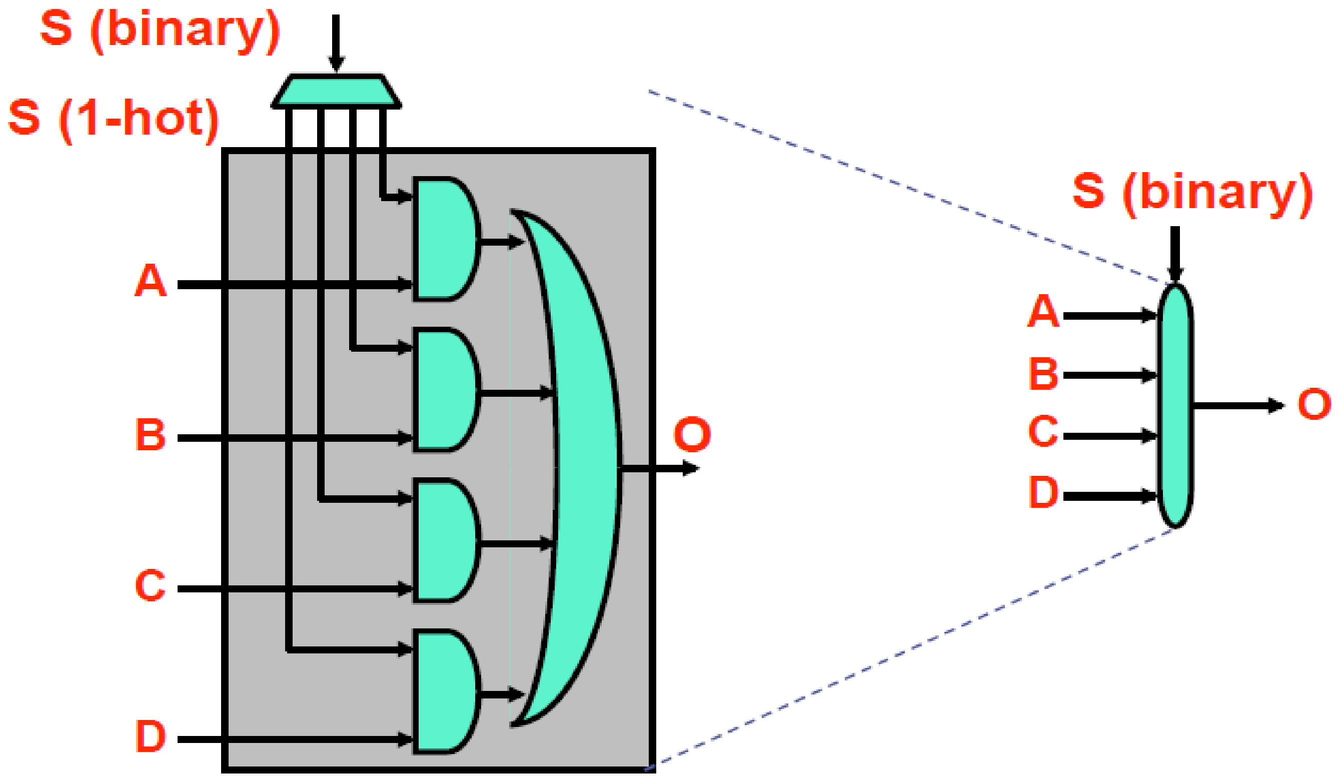
CI	A	B	→	CO	S
0	0	0	→	0	0
0	0	1	→	0	1
0	1	0	→	0	1
0	1	1	→	1	0
1	0	0	→	0	1
1	0	1	→	1	0
1	1	0	→	1	0
1	1	1	→	1	1



- **S = C′A′B + C′AB′ + CA′B′ + CAB = C ^ A ^ B**
- **CO = C′AB + CA′B + CAB′ + CAB = CA + CB + AB**

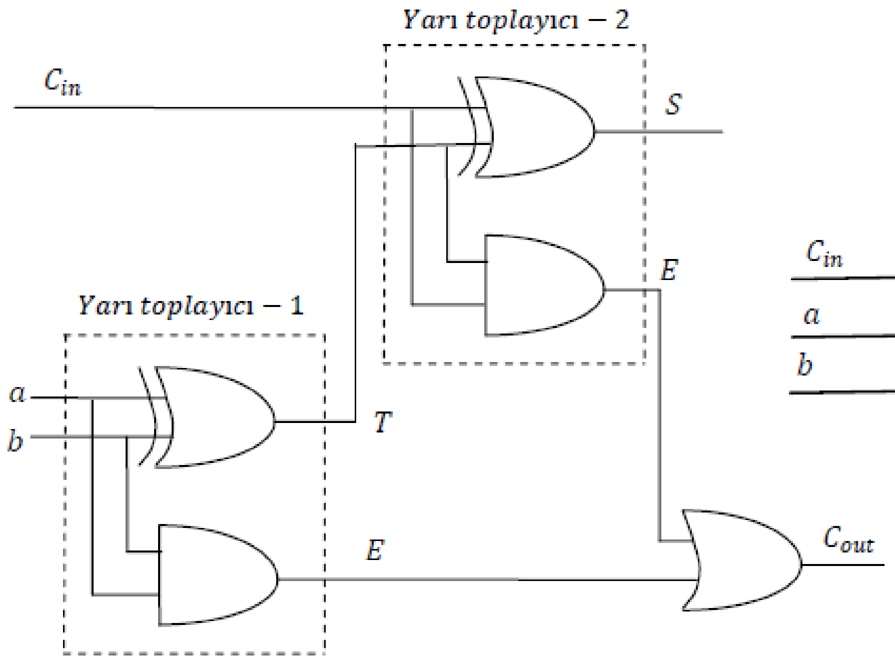
Multiplexer (mux): selects output from N inputs

- Example: 1-bit 4-to-1 mux
- Not shown: N-bit 4-to-1 mux = N 1-bit 4-to-1 muxes + 1 decoder

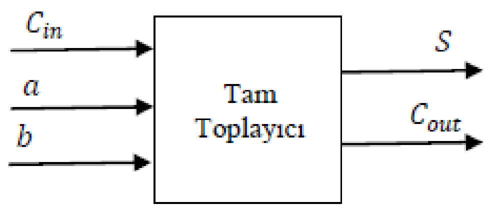


Tam toplayıcı (Full Adder)

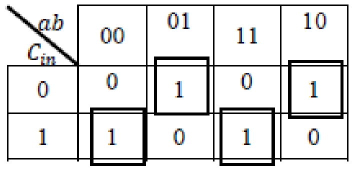
- It is a combinational circuit where the carry bit at the input is added together with two one-bit numbers.



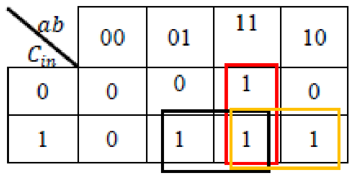
Tam toplayıcı devresi



C_{in}	a	b	Toplam S	Elde 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



C_{in} \ ab	00	01	11	10
0	0	1	0	1
1	1	0	1	0



C_{in} \ ab	00	01	11	10
0	0	0	1	0
1	0	1	1	1

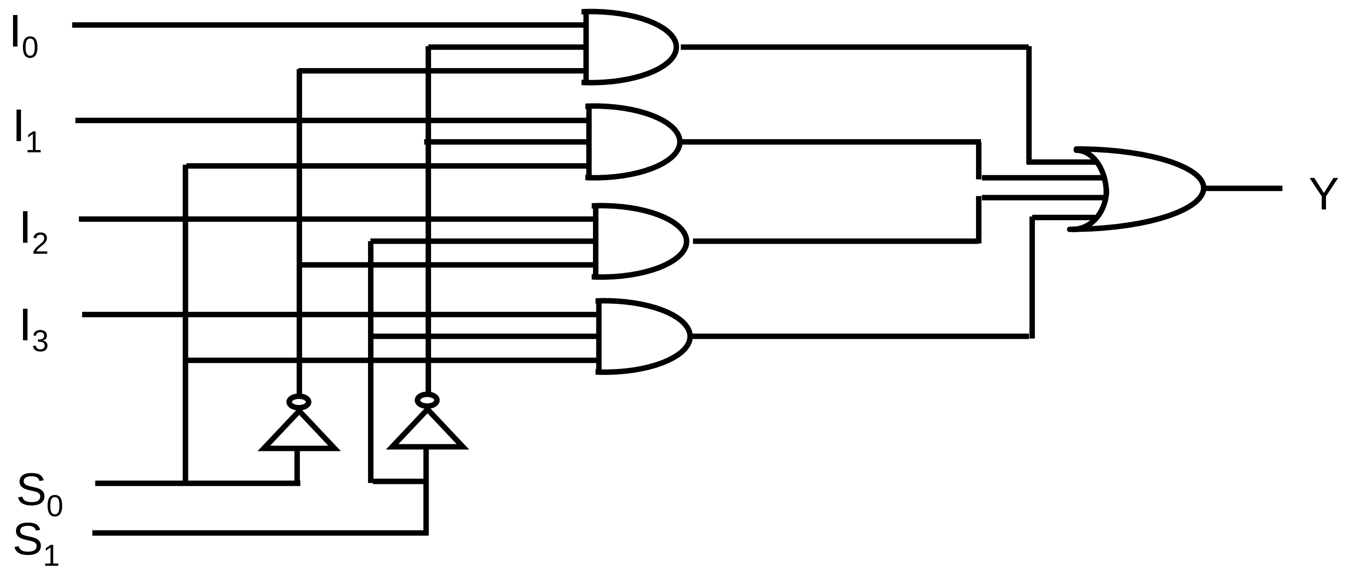
$$T = C_{in}\bar{a}\bar{b} + \overline{C_{in}}\bar{a}b + C_{in}ab + \overline{C_{in}}a\bar{b} = \overline{C_{in}}(\bar{a}b + a\bar{b}) + C_{in}(ab + \bar{a}\bar{b}) \Longrightarrow T = a \oplus b \oplus C_{in}$$

$$C_{out} = C_{in}b + C_{in}a + ab$$

MULTIPLEXER

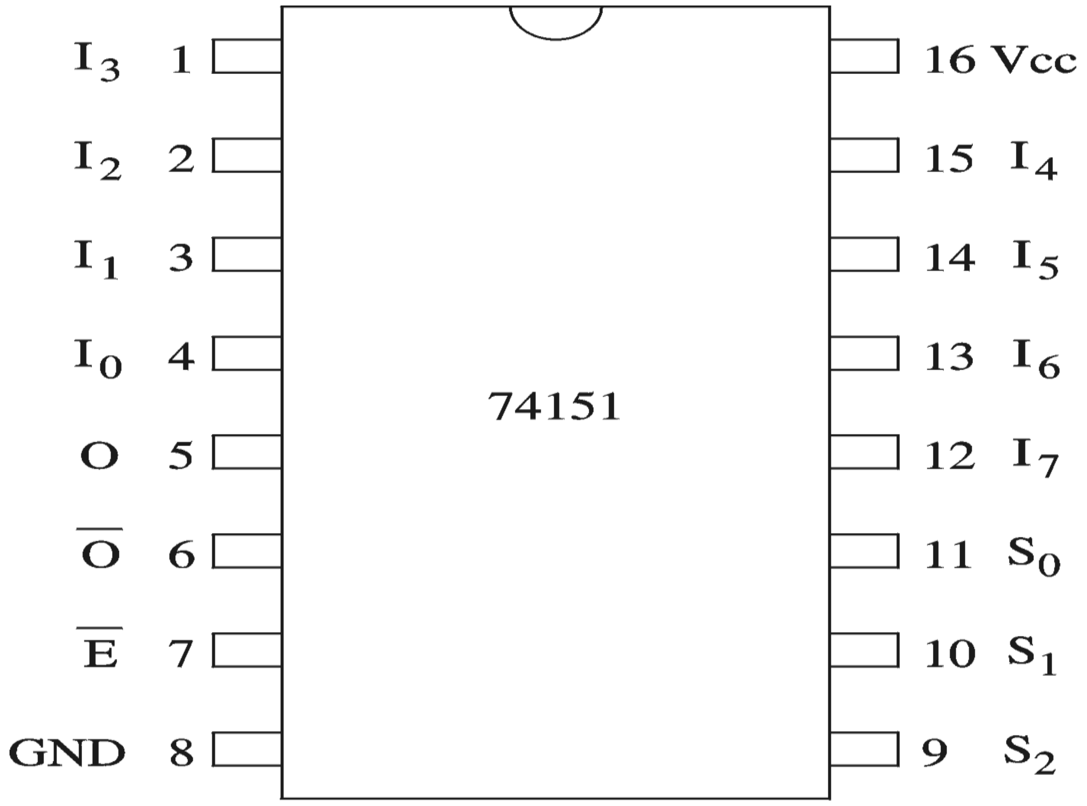
4-to-1 Multiplexer

Select		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

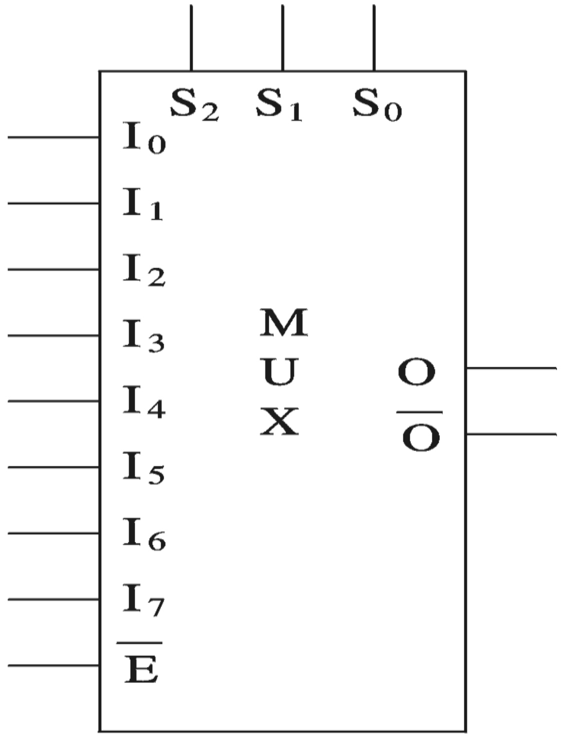


Multiplexers

Example chip: 8-to-1 MUX



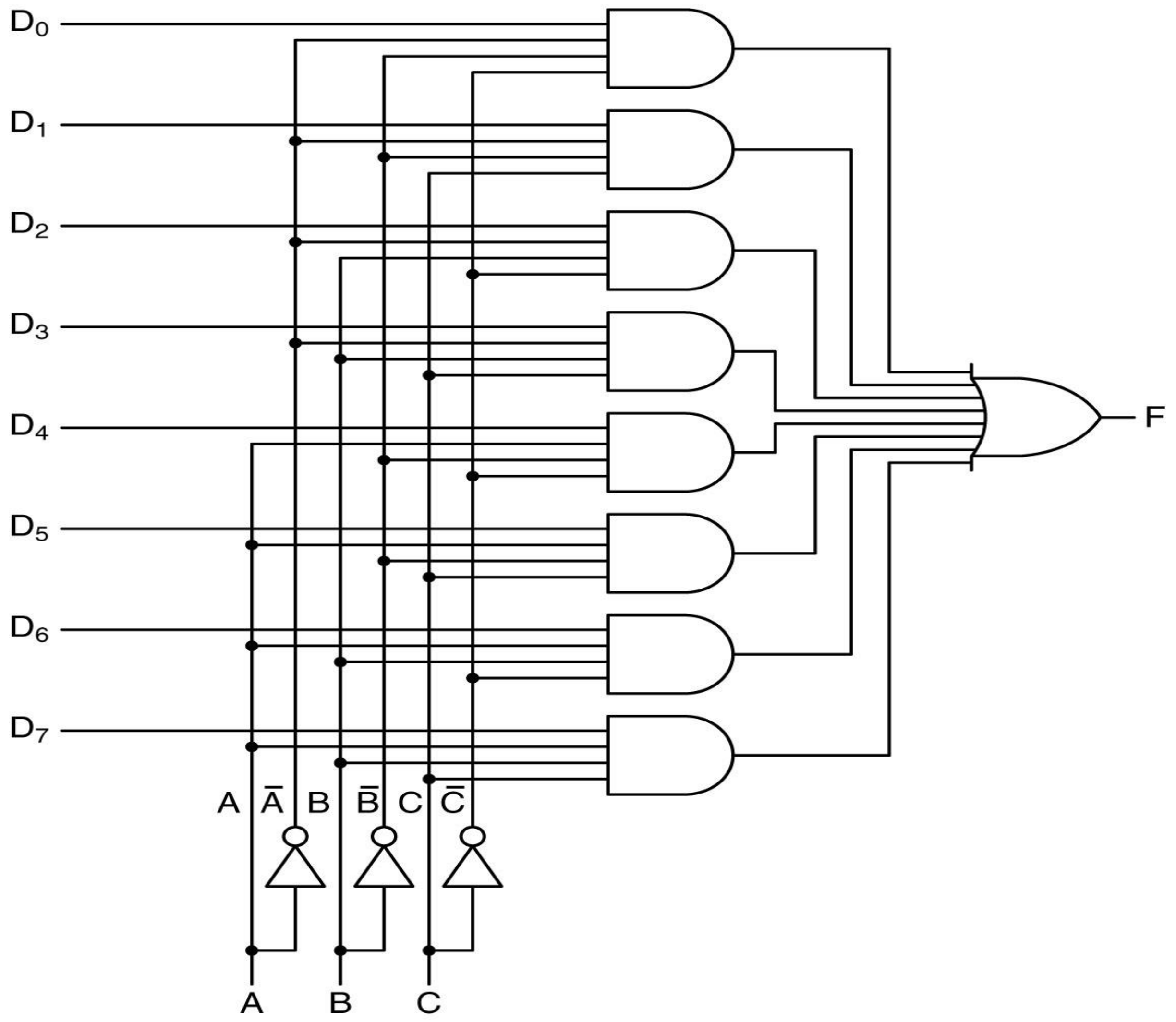
(a) Connection diagram



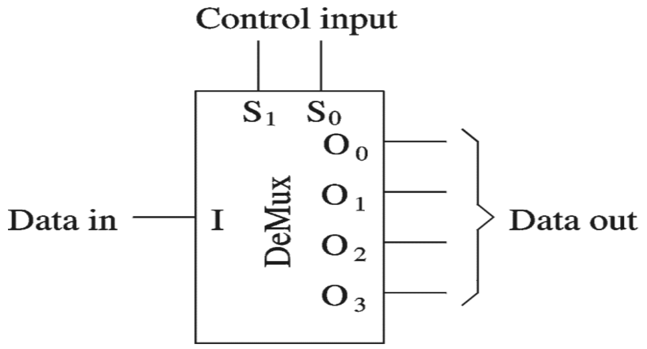
(b) Logic symbol

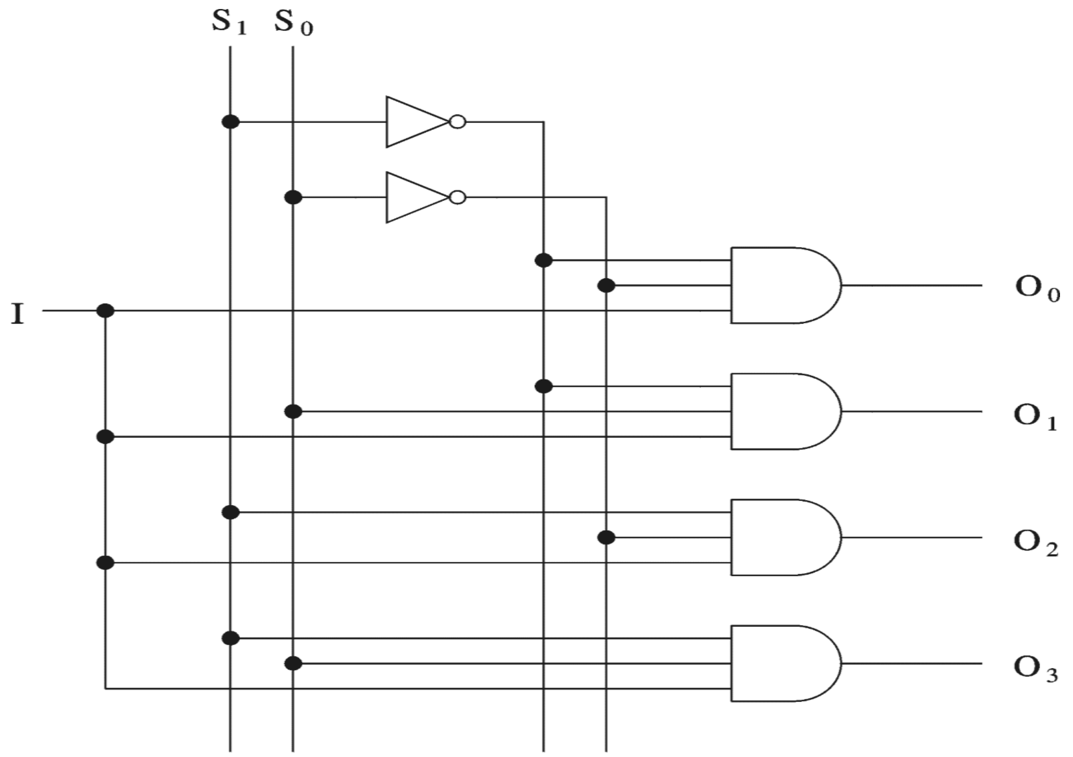
Multiplexers

An eight-input multiplexer circuit.



Demultiplexer (DeMUX)

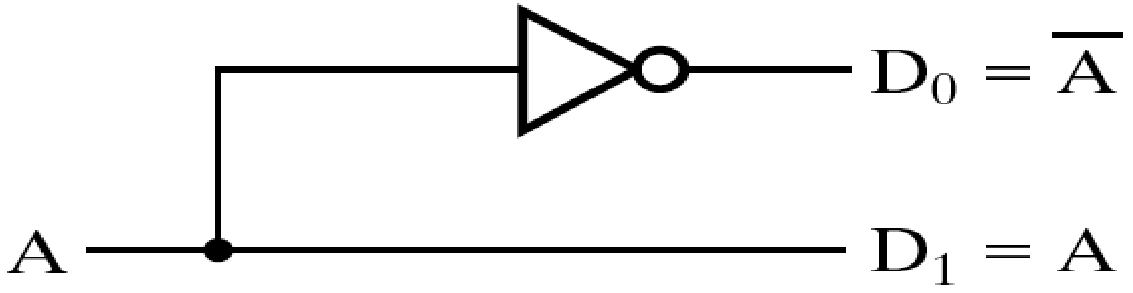




1-2 Decoder

A	D_0	D_1
0	1	0
1	0	1

(a)

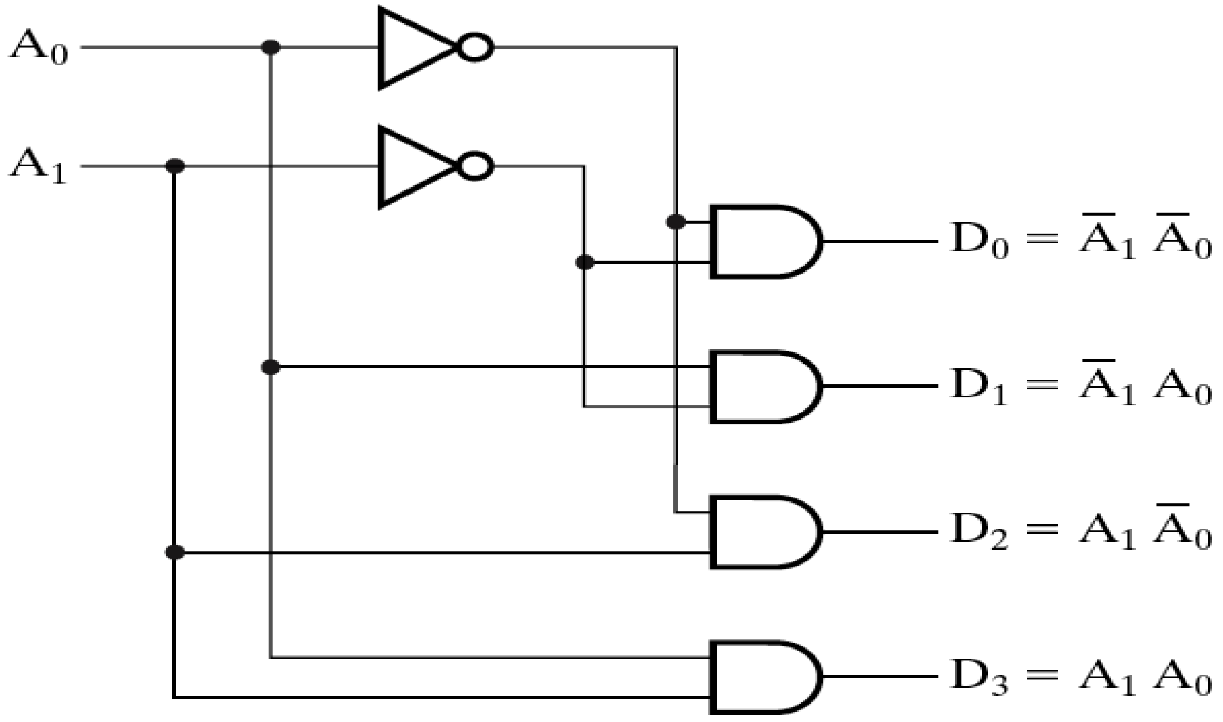


(b)

2-to-4 Decoder

A_1	A_0	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

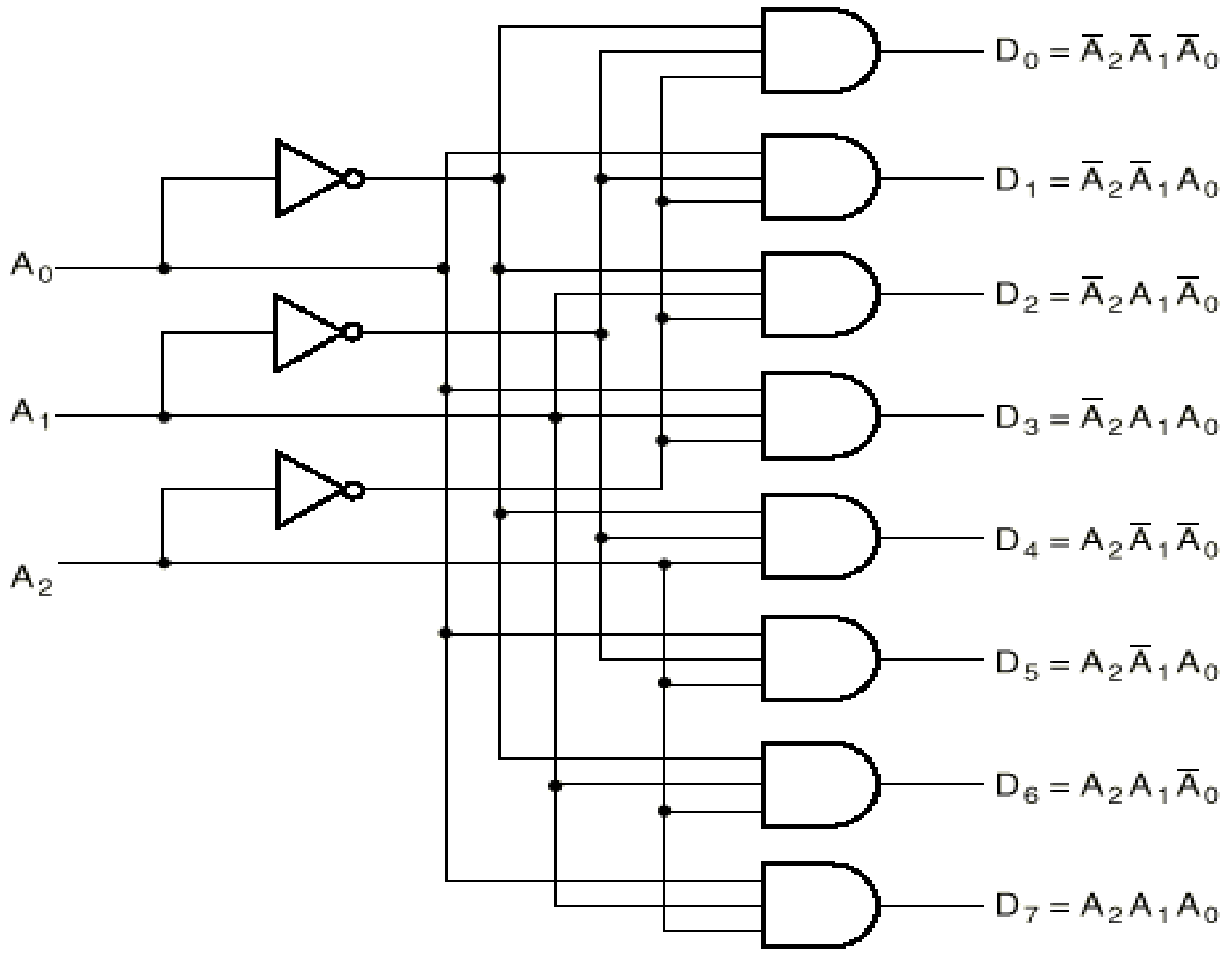
(a)


A_0
A_1
$D_0 = \overline{A}_1 \overline{A}_0$
$D_1 = \overline{A}_1 A_0$
$D_2 = A_1 \overline{A}_0$
$D_3 = A_1 A_0$

(b)

3-to-8 Decoder

- Decoding circuit is used to select memory.

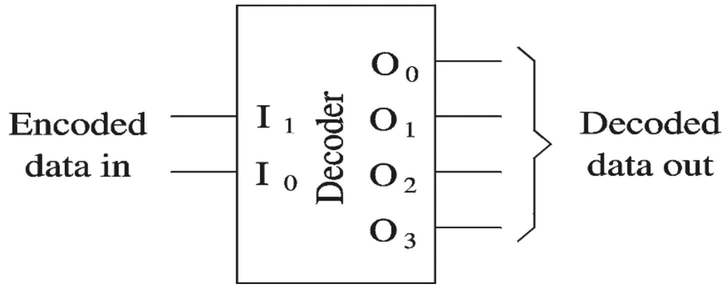


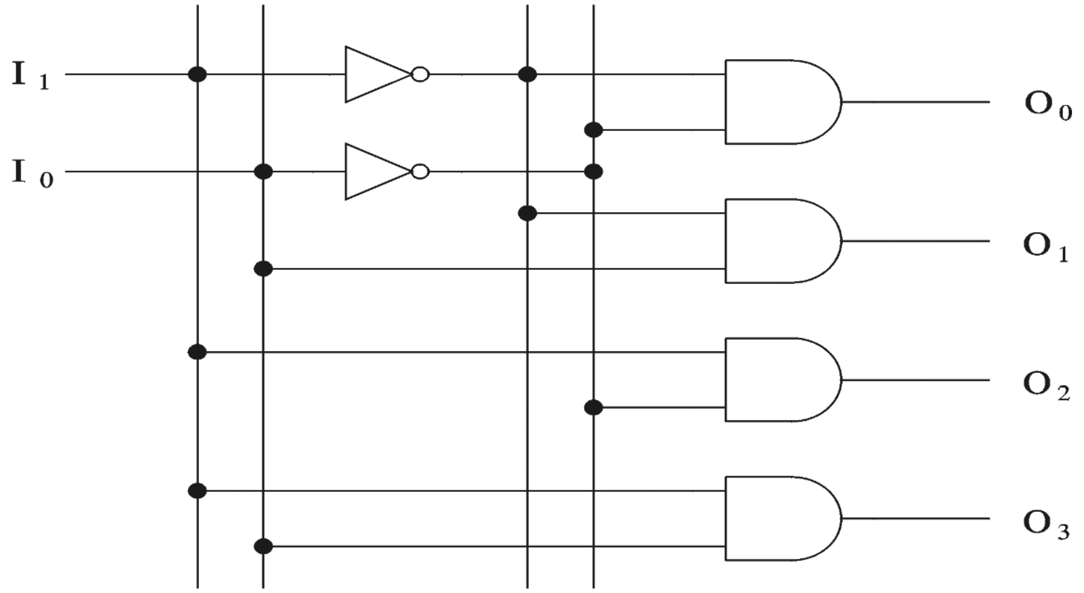
A_0
A_1
A_2
$D_0 = \overline{A}_2\overline{A}_1\overline{A}_0$
$D_1 = \overline{A}_2\overline{A}_1A_0$
$D_2 = \overline{A}_2A_1\overline{A}_0$
$D_3 = \overline{A}_2A_1A_0$
$D_4 = A_2\overline{A}_1\overline{A}_0$
$D_5 = A_2\overline{A}_1A_0$
$D_6 = A_2A_1\overline{A}_0$
$D_7 = A_2A_1A_0$

Decoders

- Decoder selects one-out-of-N inputs

I_1	I_0	O_3	O_2	O_1	O_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

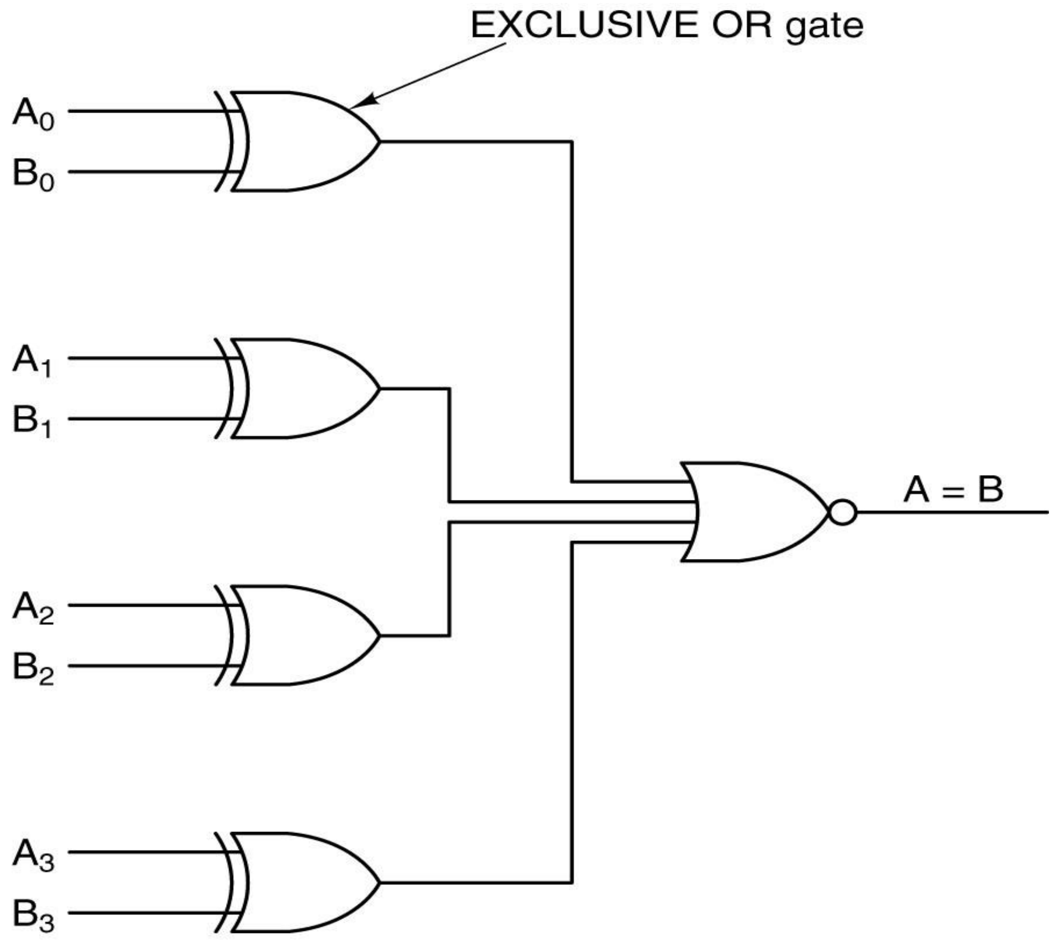




Comparators

A simple 4-bit comparator.

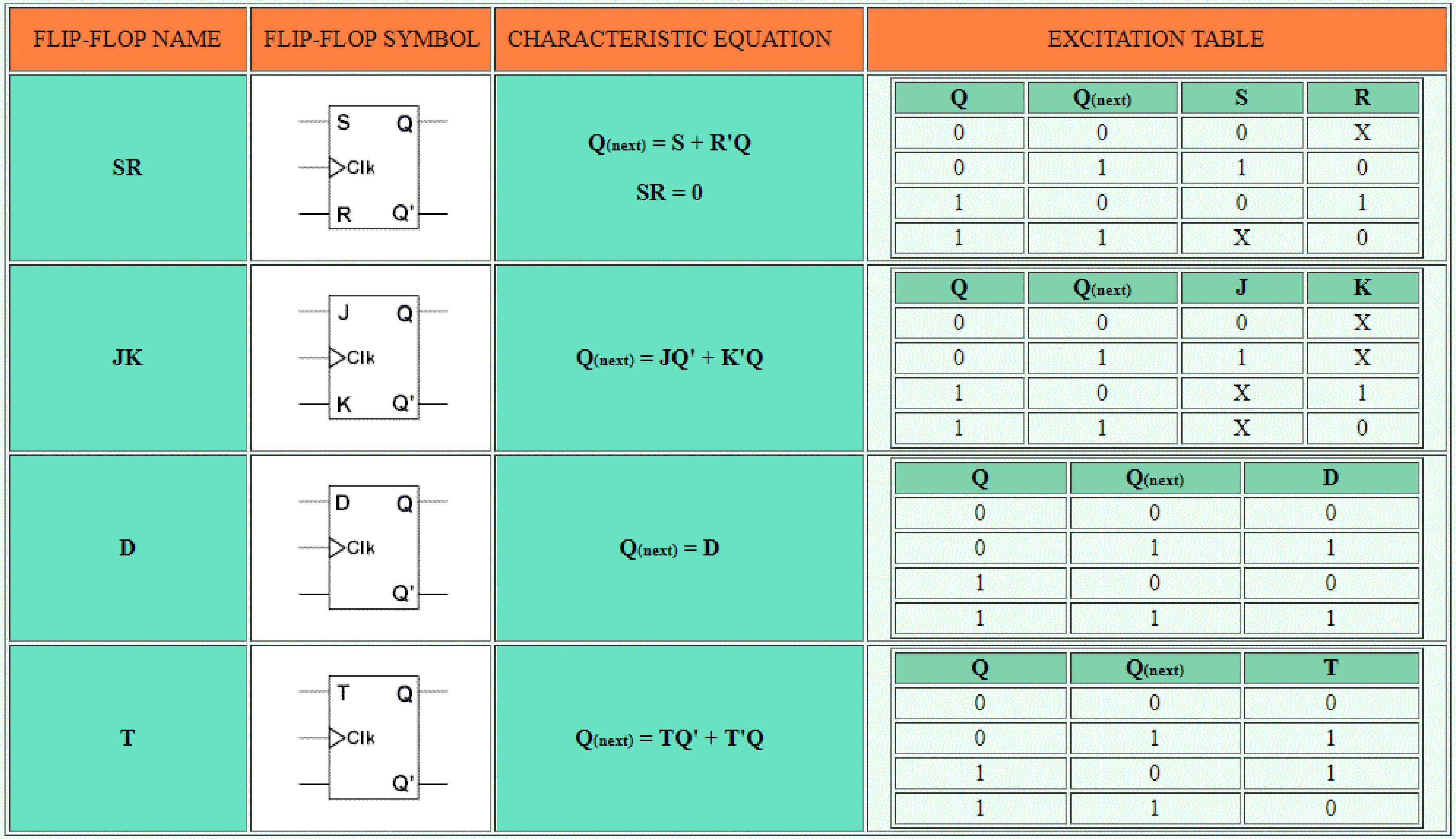
Used to implement comparison operators (= , > , < , ≥ , ≤)

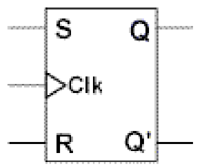
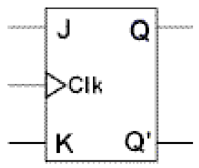
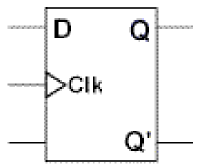
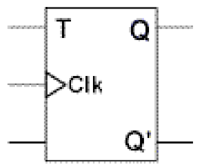


Sıralı Mantık (Sequential Logic)

- Sequential logic has memory; The circuit stores the result of the previous set of inputs. The output depends on current inputs as well as past inputs.
- The basic element in sequential logic is the two-state latch or flip-flop circuit that serves as the memory element for one bit of data.

Sequential Logic (Bellek özelliğine sahiptir.)

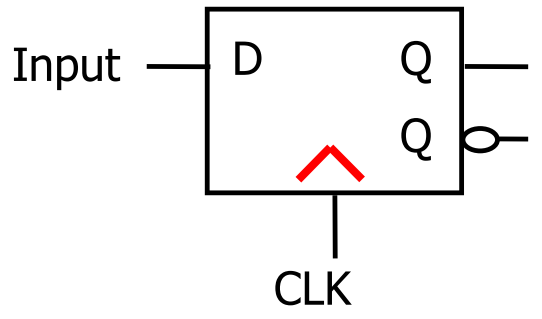


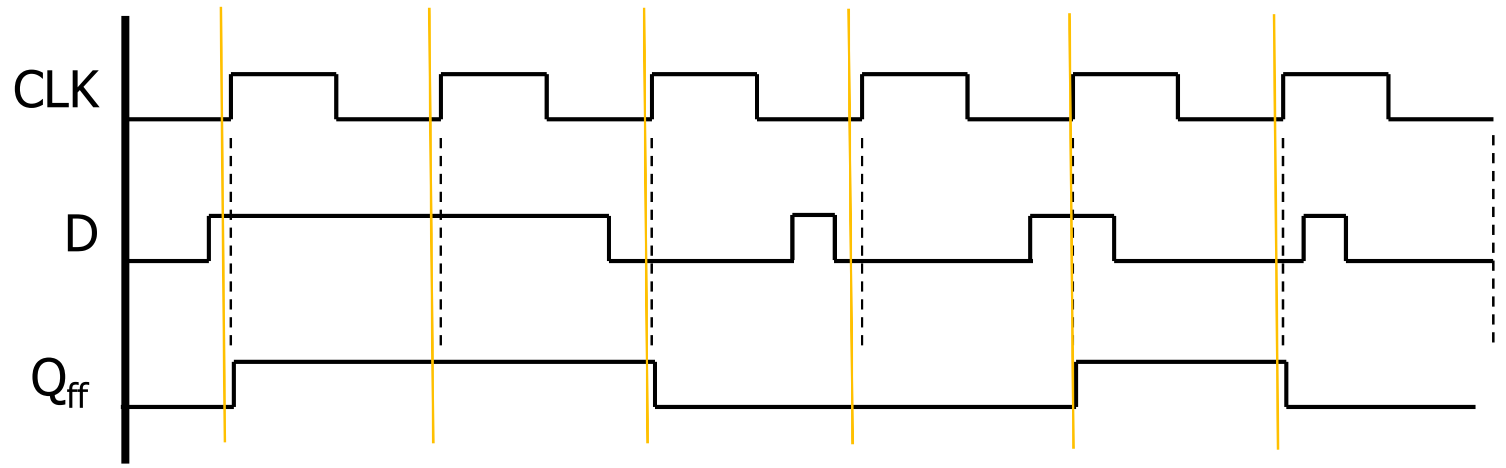
FLIP-FLOP NAME	FLIP-FLOP SYMBOL	CHARACTERISTIC EQUATION	EXCITATION TABLE			
SR		$Q_{(next)} = S + R'Q$ $SR = 0$	**Q**	**Q(next)**	**S**	**R**
			0	0	0	X
			0	1	1	0
			1	0	0	1
			1	1	X	0
JK		$Q_{(next)} = JQ' + K'Q$	**Q**	**Q(next)**	**J**	**K**
			0	0	0	X
			0	1	1	X
			1	0	X	1
			1	1	X	0
D		$Q_{(next)} = D$	**Q**	**Q(next)**	**D**	
			0	0	0	
			0	1	1	
			1	0	0	
			1	1	1	
T		$Q_{(next)} = TQ' + T'Q$	**Q**	**Q(next)**	**T**	
			0	0	0	
			0	1	1	
			1	0	1	
			1	1	0	

It is triggered by the rising edge of the Clok and the output takes the input value. Its state does not change until the next clok rising edge arrives. This is called memory feature.

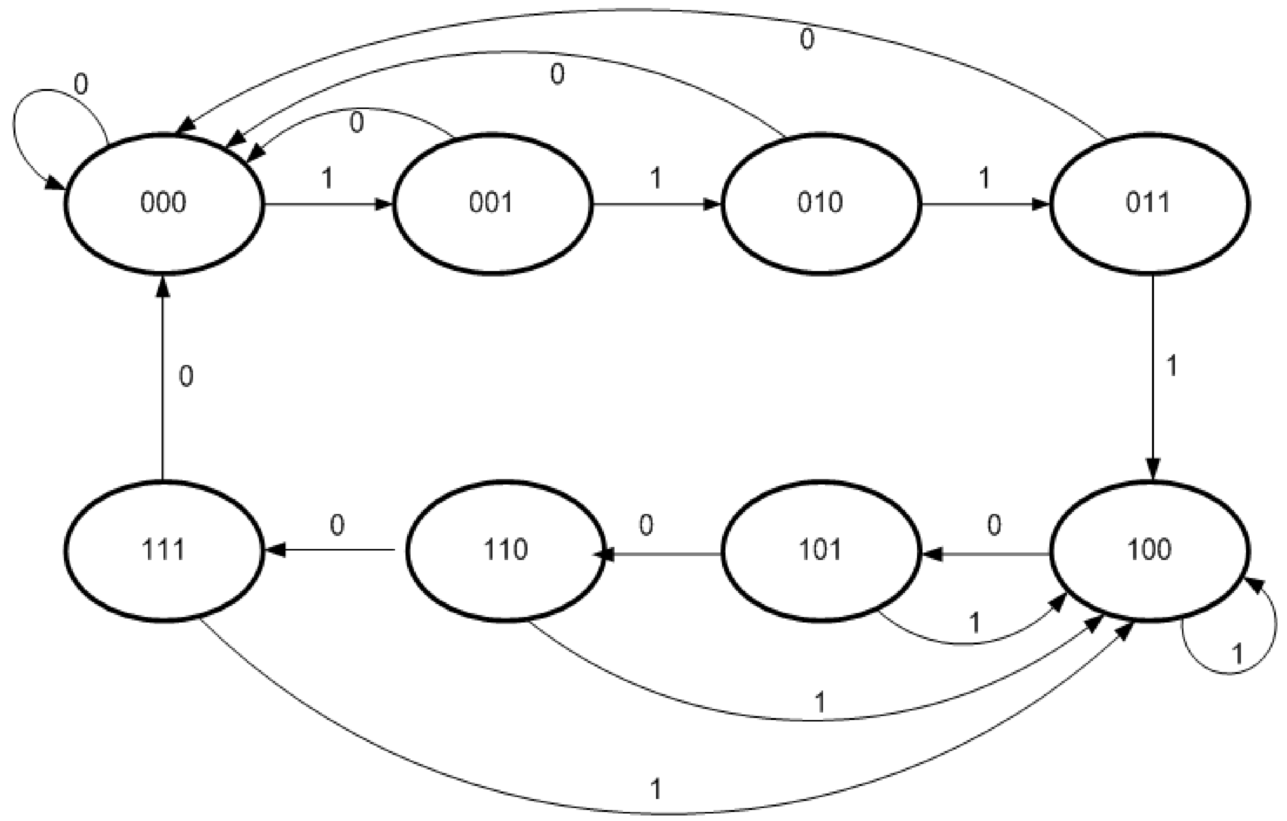
The D flip-flop

- Input sampled at clock edge
  - Rising edge: Input passes to output
  - Otherwise: Flip-flop holds its output
- Flip-flops can be rising-edge triggered or falling-edge triggered
- On the rising edge of the clok signal, the output becomes equal to the input (Q=D). In all other cases of the clok signal, the output remains unchanged.
- The current state (Q) and the next state (D) are considered together.





State Diagram

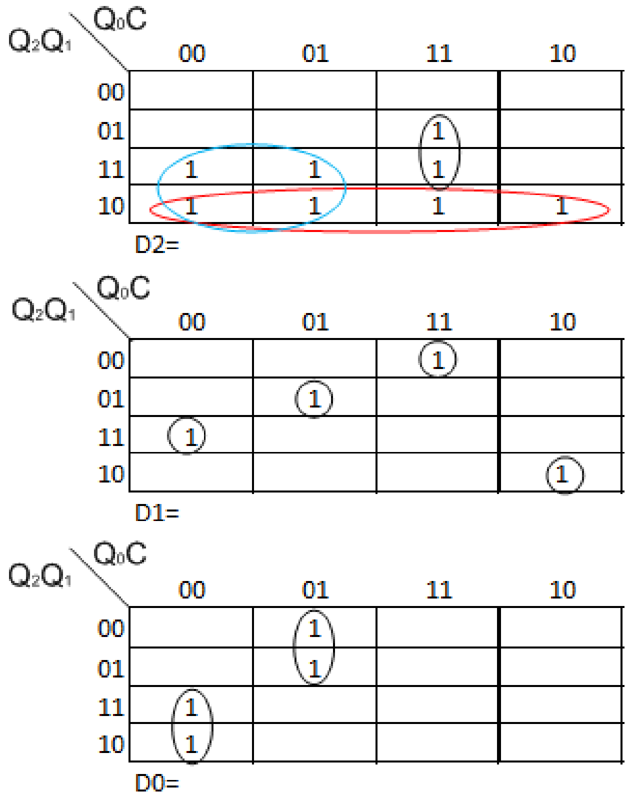


- The number of binary circuits is found according to the current and next states
- Number of pairs = 3 pieces. Because in the State diagram, all states vary between 0 and 7. Total number of states = 8 = 2^3.
- The current states are found at the Q outputs of the D-binary circuit. The next situation is at the D inputs of the D-binary circuit.
- When the D-binary circuit is triggered by the rising edge of the Clok, the Q-outputs become equal to the D-inputs.

Creating the State Table and reducing it with the help of Karnaugh diagram

Current situation — Next situation

Şu anki durum			Giriş	Bir sonraki durum		
Q2	Q1	Q0	C	D2	D1	D0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	0
0	0	1	1	0	1	0
0	1	0	0	0	0	0
0	1	0	1	0	1	1
0	1	1	0	0	0	0
0	1	1	1	1	0	0
1	0	0	0	1	0	1
1	0	0	1	1	0	0
1	0	1	0	1	1	0
1	0	1	1	1	0	0
1	1	0	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	0	0	0
1	1	1	1	1	0	0



Q_2Q_1 \ Q_0C	00	01	11	10
00				
01			1	
11	1	1	1	
10	1	1	1	1

D2=

Q_2Q_1 \ Q_0C	00	01	11	10
00			1	
01		1		
11	1			
10				1

D1=

Q_2Q_1 \ Q_0C	00	01	11	10
00		1		
01		1		
11	1			
10	1			

D0=

- D2 = Q2Q1’ + Q2Q0’ + Q1Q0C
- D1 = Q2Q1Q0’C’ + Q2’Q1Q0’C + Q2’Q1’Q0C + Q2Q1’Q0C’
- D0 = Q2’Q0’C’ + Q2Q0’C

SUMMARY

- A binary number is a weighted number in which the weight of each whole number digit is a positive power of 2 and the weight of each fractional digit is a negative power of 2.
- The 1’s complement of a binary number is derived by changing 1s to 0s and 0s to 1s
- The 2’s complement of a binary number can be derived by adding 1 to the 1’s complement.
- The octal number system consists of eight digits, 0 through 7.
- The hexadecimal number system consists of 16 digits and characters, 0 through 9 followed by A through F.
- The ASCII is a 7-bit alphanumeric code that is widely used in computer systems for input/output of information.
- The output of an inverter is the complement of its input
- The output of an AND gate is high only if all the inputs are high
- The output of an OR gate is high if any of the inputs is high
- The output of an NOR gate is low if any of the inputs is high
- The output of an NAND gate is low only if all the inputs are high
- The output of an exclusive-OR gate is high when the inputs are not the same