University of Technology

Laser and Optoelectronics Engineering Department

Fourth Year

Digital Design

Prepared by
Dr. Sinan Majid
M. Sc. Eman Yousif Nasir

Textbooks:

1. Digital Design, Morris M. Mano, (3rd Edition), Prentice Hall, 2002
2. Digital Fundamentals, Thomas L. Floyd, (9th Edition), Prentice Hall, 2006
3. Microprocessor Architecture, Programming, and Applications with the 8085, by R. Gaonkar.

Overview

- The design of computers
  - It all starts with numbers
  - Building circuits
  - Building computing machines
- Digital systems
- Understanding decimal numbers
- Binary and octal numbers
  - The basis of computers!
- Conversion between different number systems

Digital Systems

- Digital systems consider *discrete* amounts of data.

Examples

  - •26 letters in the alphabet
  - •10 decimal digits

- Larger quantities can be built from discrete values:
  - •Words made of letters
  - •Numbers made of decimal digits (e.g. 239875.32)

- Computers operate on *binary* values (0 and 1)

- Easy to represent binary values electrically
  - •Voltages and currents.
  - •Can be implemented using circuits
  - •Create the building blocks of modern computers

Understanding Decimal Numbers

•Decimal numbers are made of decimal digits: (0,1,2,3,4,5,6,7,8,9)

•But how many items does a decimal number represent?

$8653 = 8*10^3+ 6*10^2 + 5*10^1 + 3*10^0$

•What about fractions?

$97654.35 = 9*10^4+ 7*10^3 + 6*10^2 + 5*10^1 + 4*10^0+ 3*10^{-1} + 5*10^{-2}$

•In formal notation -> $(97654.35)_{10}$

Understanding Binary Numbers

•Binary numbers are made of binary digits (bits): 0 and 1

•How many items does a binary number represent?

$(1011)2= 1*2^3+ 0*2^2 + 1*2^1 + 1*2^0 = (11)_{10}$

•What about fractions?

$(110.10)2= 1*2^2 + 1*2^1 + 0*2^0+ 1*2^{-1} + 0*2^{-2}$

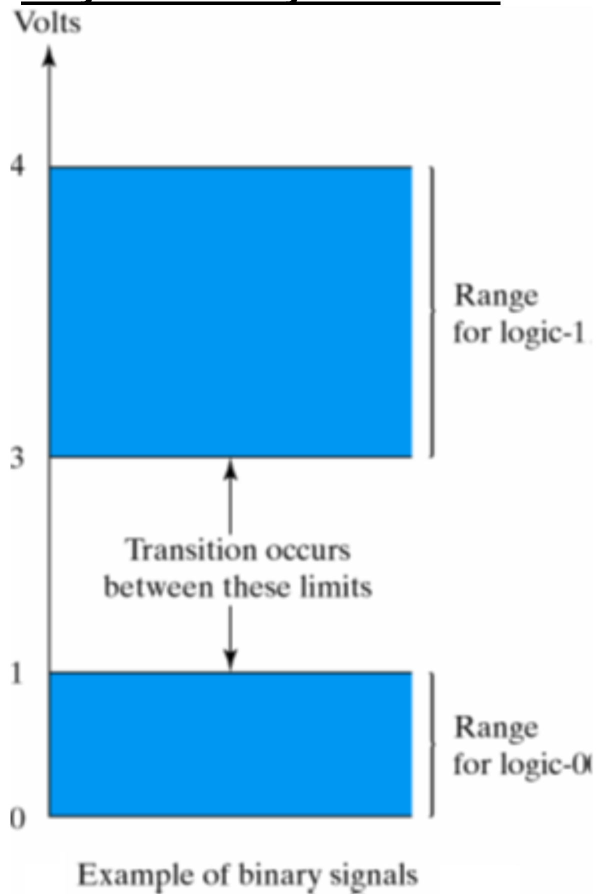
•Groups of eight bits are called a *byte*

$(11001001)_2$

•Groups of four bits are called a *nibble.*

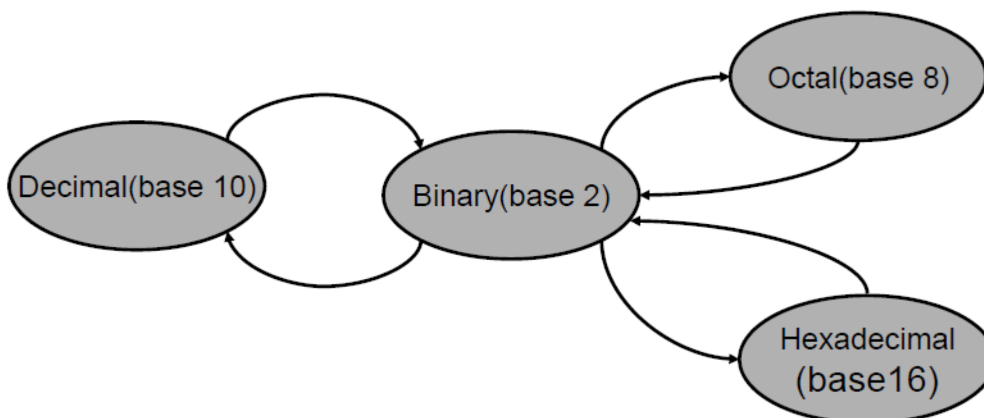
$(1101)_2$

Why Use Binary Numbers?



- Easy to represent 0 and 1 using electrical values.
- Possible to tolerate noise.
- Easy to transmit data
- Easy to build binary circuits.

Conversion Between Number Bases



- Learn to convert between bases.
- Already demonstrated how to convert from binary to decimal.

Convert an Integer *from* Decimal *to a*nother Base

For each digit position:

1. Divide decimal number by the base (e.g. 2)
2. The remainder is the lowest-order digit3.Repeat first two steps until no divisor remains.

Example for $(13)_{10}$:

	Integer Quotient		Remainder	Coefficient
13/2 =	6	+	½	$a_0 = 1$
6/2 =	3	+	0	$a_1 = 0$
3/2 =	1	+	½	$a_2 = 1$
1/2 =	0	+	½	$a_3 = 1$

Answer $(13)_{10} = (a_3\, a_2\, a_1\, a_0)_2 = (1101)_2$

Convert an Fraction *from* Decimal *to a*nother Base1.

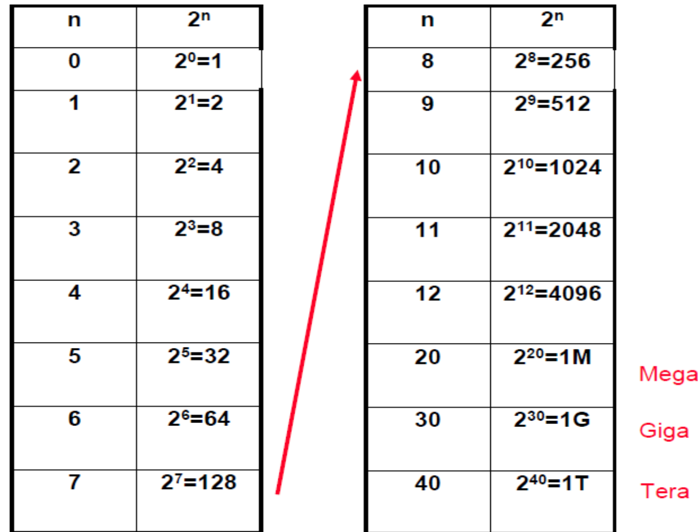
1. Multiply decimal number by the base (e.g. 2)
2. The *integer* is the highest-order digit
3. Repeat first two steps until fraction becomes zero.

Example for $(0.625)_{10}$:

	Integer		Fraction	Coefficient
0.625 x 2 =	1	+	0.25	$a_{-1} = 1$
0.250 x 2 =	0	+	0.50	$a_{-2} = 0$
0.500 x 2 =	1	+	0	$a_{-3} = 1$

Answer $(0.625)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_2 = (0.101)_2$

The Growth of Binary Numbers



n	2^n
0	2^0=1
1	2^1=2
2	2^2=4
3	2^3=8
4	2^4=16
5	2^5=32
6	2^6=64
7	2^7=128

n	2^n	
8	2^8=256	
9	2^9=512	
10	2^{10}=1024	
11	2^{11}=2048	
12	2^{12}=4096	
20	2^{20}=1M	Mega
30	2^{30}=1G	Giga
40	2^{40}=1T	Tera

Binary Addition

- Binary addition is very simple.
- This is best shown in an example of adding two binary numbers…

```
  1  1  1  1  1  1 <------- carries
     1  1  1  1  0  1
+       1  0  1  1  1
---------------------
  1  0  1  0  1  0  0
```

Binary Subtraction

- We can also perform subtraction (with borrows in place of carries).
- Let's subtract $(10111)_2$ from $(1001101)_2$…

```
       1          10     <------- borrows
    0  10  10  0  0  10

    1  0  0  1  1  0  1    (first six digits struck through)
-         1  0  1  1  1
------------------------
       1  1  0  1  1  0
```

Binary Multiplication

Binary multiplication is much the same as decimal multiplication, except that the multiplication operations are much simpler…

```
            1  0  1  1  1
X              1  0  1  0
-------------------------
            0  0  0  0  0
         1  0  1  1  1
      0  0  0  0  0
   1  0  1  1  1
-------------------------
   1  1  1  0  0  1  1  0
```

Understanding Octal Numbers

- Octal numbers are made of octal digits: (0,1,2,3,4,5,6,7)
- How many items does an octal number represent?
  - $(4536)_8 = 4x8^3 + 5x8^2 + 3x8^1 + 6x8^0 = (1362)_{10}$
- What about fractions?
  - $(465.27)_8 = 4x8^2 + 6x8^1 + 5x8^0 + 2x8^{-1} + 7x8^{-2}$
- Octal numbers don't use digits 8 or 9

Convert an Integer *from* Decimal *to* Octal

1. Divide decimal number by the base (8)
2. The *remainder* is the lowest-order digit
3. Repeat first two steps until no *divisor* remains.

Example for $(175)_{10}$:

	Integer Quotient		Remainder	Coefficient
175/8 =	21	+	7/8	$a_0 = 7$
21/8 =	2	+	5/8	$a_1 = 5$
2/8 =	0	+	2/8	$a_2 = 2$

Answer $(175)_{10} = (a_2\, a_1\, a_0)_2 = (257)_8$

Convert an Fraction *from* Decimal *to* Octal

1. Multiply decimal number by the base (e.g. 8)
2. The *integer* is the highest-order digit
3. Repeat first two steps until fraction becomes zero.

Example for $(0.3125)_{10}$:

	Integer		Fraction	Coefficient
0.3125 x 8 =	2	+	5	$a_{-1} = 2$
0.5000 x 8 =	4	+	0	$a_{-2} = 4$

Answer $(0.3125)_{10} = (0.24)_8$

Understanding Hexadecimal Numbers

- **Hexadecimal numbers are made of 16 digits:**
  - (0,1,2,3,4,5,6,7,8,9,A, B, C, D, E, F)
- **How many items does an hex number represent?**
  - $(3A9F)_{16} = 3x16^3 + 10x16^2 + 9x16^1 + 15x16^0 = 14999_{10}$
- **What about fractions?**
  - $(2D3.5)_{16} = 2x16^2 + 13x16^1 + 3x16^0 + 5x16^{-1} = 723.3125_{10}$
- **Note that *each* hexadecimal digit can be represented with four bits.**
  - $(1110)_2 = (E)_{16}$
- **Groups of four bits are called a *nibble.***
  - $(1110)_2$

Putting It All Together

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

- **Binary, octal, and hexadecimal similar**
- **Easy to build circuits to operate on these representations**
- **Possible to convert between the three formats**

Converting Between Base 16 and Base 2

$3A9F_{16}$ = 0011 1010 1001 1111_2

3 A 9 F

°Conversion is easy!

- Determine 4-bit value for each hex digit

°Note that there are 24= 16 different values of four bits

°Easier to read and write in hexadecimal.

°Representations are equivalent!

Converting Between Base 16 and Base 8

$3A9F_{16}$ = 0011 1010 1001 1111_2

3 A 9 F

↓

35237_8 = 011 101 010 011 111_2

3 5 2 3 7

1. Convert from Base 16 to Base 2
2. Regroup bits into groups of three starting from right
3. Ignore leading zeros
4. Each group of three bits forms an octal digit.

Signed Numbers

How to Represent Signed Numbers

•Plus and minus sign used for decimal numbers: 25 (or +25), -16, etc.

•For computers, desirable to represent everything as *bits*.

•Three types of signed binary number representations: signed magnitude, 1's complement, 2's complement.

•In each case: left-most bit indicates sign: positive (0) or negative (1).

Consider *signed magnitude*:

One's Complement Representation

•The one's complement of a binary number involves inverting all bits.

•1's comp of 00110011 is 11001100

•1's comp of 10101010 is 01010101

•For an n bit number N the 1's complement is $(2^n-1) – N$.

•Called diminished radix complement by Mano since 1's complement for base (radix 2).

•To find negative of 1's complement number take the 1's complement.

Two's Complement Representation

•The two's complement of a binary number involves inverting all bits and adding 1.

•2's comp of 00110011 is 11001101

•2's comp of 10101010 is 01010110

•For an n bit number N the 2's complement is $(2^n-1) - N + 1$.

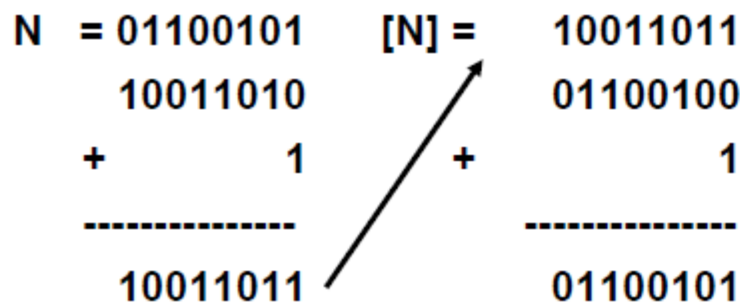
•Called radix complement by Mano since 2's complement for base (radix 2).

•To find negative of 2's complement number take the 2's complement.

Two's Complement Shortcuts

°Algorithm –Simply complement each bit and then add 1 to the result.

•Finding the 2's complement of (01100101)2and of its 2's complement…



1's Complement Addition

°Using 1's complement numbers, adding numbers is easy.

°For example, suppose we wish to add $+(1100)_2$ and $+(0001)_2$.

°Let's compute $(12)_{10} + (1)_{10}$.

•$(12)_{10} = +(1100)_2 = 01100_2$ in 1's comp.

•$(1)_{10} = +(0001)_2 = 00001_2$ in 1's comp.

Step 1: Add binary numbers
Step 2: Add carry to low-order bit

```
                0 1 1 0 0
Add       +     0 0 0 0 1
          -----------------
          0     0 1 1 0 1
Add carry └───────────→ 0
          -----------------
Final
Result          0 1 1 0 1
```

°Adding the carry bit, the sign bit is seen to be zero, indicating a positive result, $(01101)_2 = +(1101)_2 = +(13)_{10}$

1's Complement Subtraction

°Using 1's complement numbers, subtracting numbers is also easy.

°For example, suppose we wish to subtract $+(0001)_2$ from $+(1100)_2$.

° **Let's compute $(12)_{10} - (1)_{10}$.**

- **$(12)_{10} = +(1100)_2 = 01100_2$ in 1's comp.**
- **$(-1)_{10} = -(0001)_2 = 11110_2$ in 1's comp.**

Step 1: Take 1's complement of 2nd operand
Step 2: Add binary numbers
Step 3: Add carry to low order bit

```
                0 1 1 0 0
          -     0 0 0 0 1
          -----------------
1's comp
                0 1 1 0 0
Add       +     1 1 1 1 0
          -----------------
          1     0 1 0 1 0
Add carry └───────────→ 1
          -----------------
Final
Result          0 1 0 1 1
```

°Adding the carry bit, the sign bit is seen to be zero, indicating a positive result, $(01011)_2 = +(1011)_2 = +(11)_{10}$

2’s Complement Addition

°Using 2’s complement numbers, adding numbers is easy.

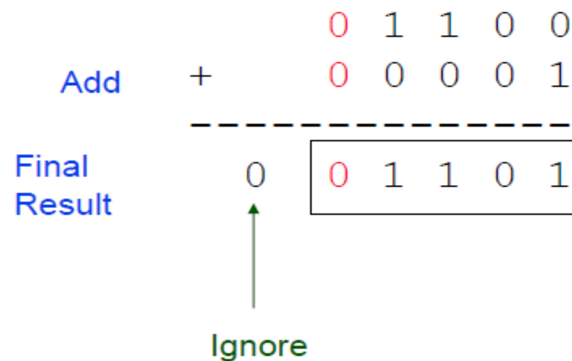
°For example, suppose we wish to add $+(1100)_2$ and $+(0001)_2$.

°Let’s compute $(12)_{10} + (1)_{10}$.

•$(12)_{10} = +(1100)_2 = 01100_2$ in 2’s comp.

•$(1)_{10} = +(0001)_2 = 00001_2$ in 2’s comp.

Step 1: Add binary numbers
Step 2: Ignore carry bit



°Discarding the carry bit, the sign bit is seen to be zero, indicating a positive result, $(01101)_2 = +(1101)_2 = +(13)_{10}$

2’s Complement Subtraction

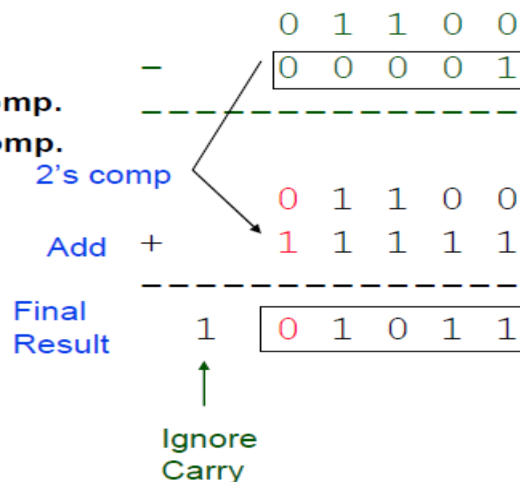
°Using 2’s complement numbers, follow steps for subtraction

°For example, suppose we wish to subtract $+(0001)_2$ from $+(1100)_2$.

° **Let’s compute $(12)_{10} - (1)_{10}$.**

- **$(12)_{10} = +(1100)_2 = 01100_2$ in 2’s comp.**
- **$(-1)_{10} = -(0001)_2 = 11111_2$ in 2’s comp.**

Step 1: Take 2’s complement of 2nd operand
Step 2: Add binary numbers
Step 3: Ignore carry bit



°Discarding the carry bit, the sign bit is seen to be zero, indicating a positive result, $(01011)_2 = +(1011)_2 = +(11)_{10}$

2's Complement Subtraction: Example #2

°Let's compute $(13)_{10}-(5)_{10}$.

•$(13)_{10} = +(1101)_2 = (01101)_2$

•$(-5)_{10} = -(0101)_2 = (11011)_2$

°Adding these two 5-bit codes…

```
                      0 1 1 0 1
carry            +    1 1 0 1 1
                 --------------
              -->  1  0 1 0 0 0
```

°Discarding the carry bit, the sign bit is seen to be zero, indicating a positive result, $(01000)_2 = +(1000)_2 = +(8)_{10}$

2's Complement Subtraction: Example #3

°Let's compute $(5)_{10}-(12)_{10}$.

•$(-12)_{10} = -(1100)_2 = (10100)_2$

•$(5)_{10} = +(0101)_2 = (00101)_2$

°Adding these two 5-bit codes…

```
       0 0 1 0 1
+      1 0 1 0 0
----------------
       1 1 0 0 1
```

°Here, there is no carry bit and the sign bit is 1. This indicates a negative result, which is what we expect. $(11001)_2 = -(7)_{10}$.

Binary Coded Decimal

Digit	BCD Code	Digit	BCD Code
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

- **Binary coded decimal (BCD) represents each decimal digit with four bits**
  - **Ex.** 0011 0010 1001 = 329_{10}

    3 2 9

- **This is *NOT* the same as 001100101001_2**

Putting It All Together

Decimal	Binary	Octal	Hexadecimal	BCD
0	0	0	0	0000
1	1	1	1	0001
2	10	2	2	0010
3	11	3	3	0011
4	100	4	4	0100
5	101	5	5	0101
6	110	6	6	0110
7	111	7	7	0111
8	1000	10	8	1000
9	1001	11	9	1001
10	1010	12	A	0001 0000
11	1011	13	B	0001 0001
12	1100	14	C	0001 0010
13	1101	15	D	0001 0011
14	1110	16	E	0001 0100
15	1111	17	F	0001 0101

- **BCD not very efficient**
- **Used in early computers (40s, 50s)**
- **Used to encode numbers for seven-segment displays.**
- **Easier to read?**

Gray Code

Digit	Binary	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

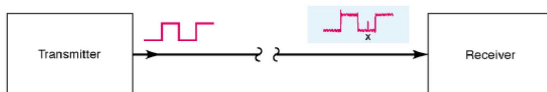
- Gray code is not a number system.
  - It is an alternate way to represent four bit data
- Only one bit changes from one decimal digit to the next
- Useful for reducing errors in communication.
- Can be scaled to larger numbers.

ASCII Code

- American Standard Code for Information Interchange
- ASCII is a 7-bit code, frequently used with an 8th bit for error detection (more about that in a bit).

Character	ASCII (bin)	ASCII (hex)	Decimal	Octal
A	1000001	41	65	101
B	1000010	42	66	102
C	1000011	43	67	103
…				
Z				
a				
…				
1				
‘				

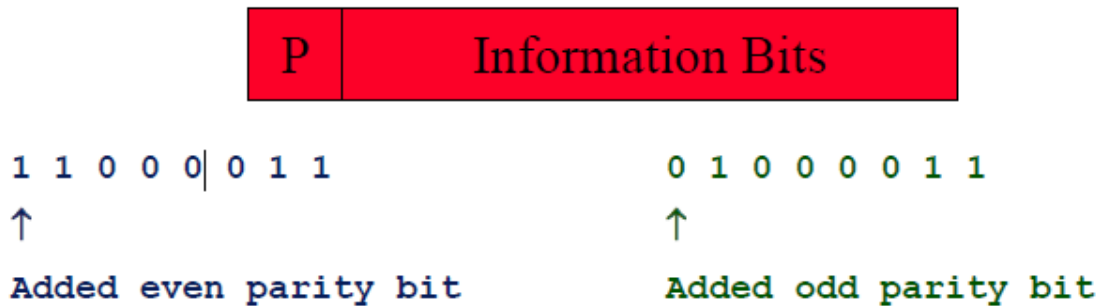
ASCII Codes and Data Transmission



- ASCII Codes
  - A – Z (26 codes), a – z (26 codes)
  - 0-9 (10 codes), others (@#$%^&*….)

Parity Codes

°Parity codes are formed by concatenating a *parity bit, P* to each code word of C.
°In an *odd-parity code*, the parity bit is specified so that the total number of ones is odd.
°In an even-*parity code*, the parity bit is specified so that the total number of ones is even.



Parity Code Example

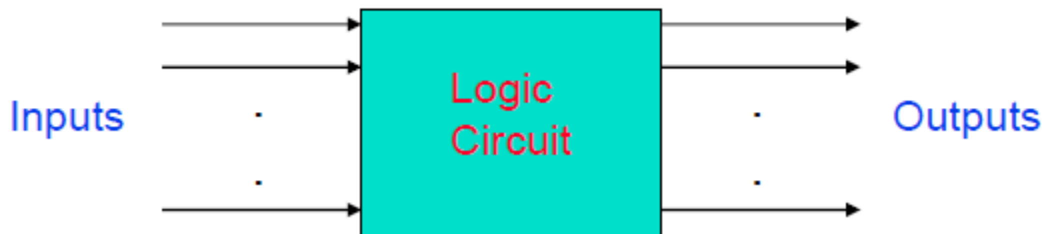
°Concatenate a parity bit to the ASCII code for the characters 0, X, and = to produce both odd-parity and even-parity codes.

Character	ASCII	Odd-Parity ASCII	Even-Parity ASCII
0	0110000	10110000	00110000
X	1011000	01011000	11011000
=	0111100	10111100	00111100

Boolean Algebra and Logic Gates

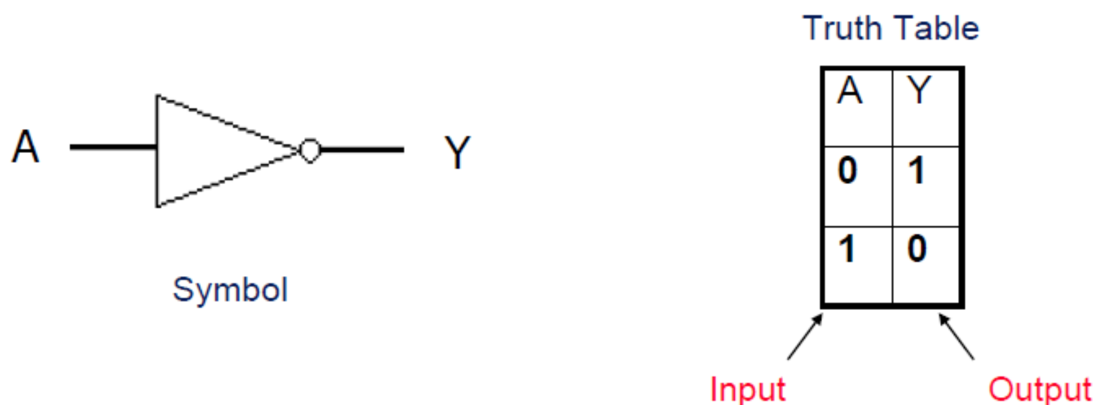
Combinational Circuit: The outputs at any instance of time are entirely dependent upon the inputs present at that time.

°Analysis problem:



•Determine binary outputs for each combination of inputs
°Design problem: given a task, develop a circuit that accomplishes the task
•Many possible implementation
•Try to develop "best" circuit based on some criterion (size, power, performance, etc.)

Describing Circuit Functionality: Inverter



°Basic logic functions have symbols.
°The same functionality can be represented with truth tables.
•Truth table completely specifies outputs for all input combinations.
°The above circuit is an inverter.
•An input of 0 is inverted to a 1.
•An input of 1 is inverted to a 0.

The AND Gate

- This is an AND gate.
- So, if the two inputs signals are asserted (high) the output will also be asserted. Otherwise, the output will be deasserted (low).

Truth Table

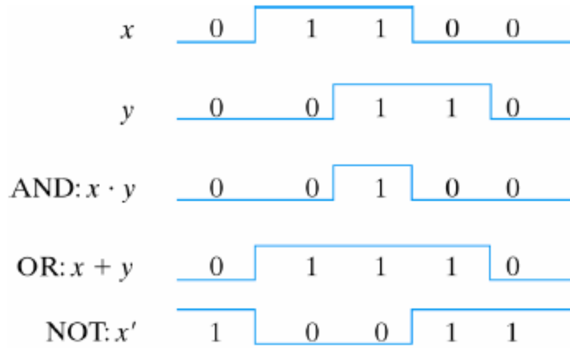
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

The OR Gate

- This is an OR gate.
- So, if either of the two input signals are asserted, or both of them are, the output will be asserted.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Describing Circuit Functionality: Waveforms



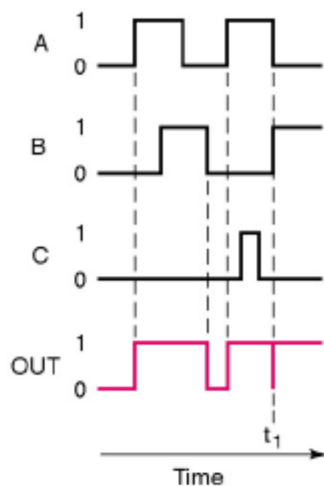
Input-output signals for gates

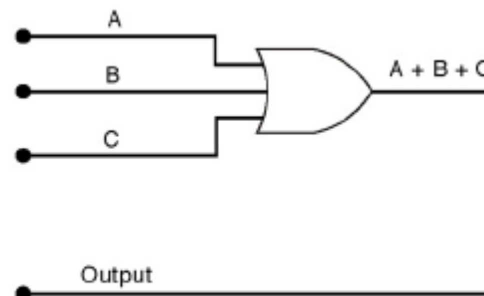
AND Gate

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

°Waveforms provide another approach for representing functionality.
°Values are either high (logic 1) or low (logic 0).
°Can you create a truth table from the waveforms?

Consider three-input gates





3 Input OR Gate

A	B	C	x = A + B + C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Boolean Algebra

•A *Boolean algebra* is defined as a closed algebraic system containing a set K or two or more elements and the two operators, . and +.

•Useful for identifying and *minimizing* circuit functionality

•Identity elements

$a + 0 = a$

$a \,.\, 1 = a$

•0 is the identity element for the + operation.

•1 is the identity element for the . operation.

Commutativity and Associativity of the Operators

•The Commutative Property:

For every a and b in K,

$a + b = b + a$

$a \,.\, b = b \,.\, a$

•The Associative Property:

For every a, b, and c in K,

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

$a \,.\, (b \,.\, c) = (a \,.\, b) \,.\, c$

Distributivity of the Operators and Complements

•The Distributive Property:

For every a, b, and c in K,

$a + (b \,.\, c) = (a + b) \,.\, (a + c)$

$a \,.\, (b + c) = (a \,.\, b) + (a \,.\, c)$

•The Existence of the Complement:

For every a in K there exists a unique element called a'(*complement of a*) such that,

$a + a' = 1$

$a \,.\, a' = 0$

•To simplify notation, the . operator is frequently omitted. When two elements are written next to each other, the AND (.) operator is implied…

$a + b \,.\, c = (a + b) \,.\, (a + c)$

$a + bc = (a + b)(a + c)$

Duality

- The principle of *duality* is an important concept. This says that if an expression is valid in Boolean algebra, the dual of that expression is also valid.
- To form the dual of an expression, replace all + operators with . operators, all . operators with + operators, all ones with zeros, and all zeros with ones.
- Form the dual of the expression

  $a + (bc) = (a + b)(a + c)$
- Following the replacement rules…

  $a(b + c) = ab + ac$
- Take care not to alter the location of the parentheses if they are present.

Involution

- This theorem states:

  $a'' = a$

  Remember that $aa' = 0$ and $a + a' = 1$.
- Therefore, a’ is the complement of a and a is also the complement of a’.
- As the complement of a’ is unique, it follows that $a'' = a$.

Taking the double inverse of a value will give the initial value.

Absorption

- This theorem states:

  $a + ab = a$ $\quad\quad a(a+b) = a$
- To prove the first half of this theorem:

$$
\begin{aligned}
a + ab &= a \cdot 1 + ab \\
&= a(1 + b) \\
&= a(b + 1) \\
&= a(1) \\
&= a
\end{aligned}
$$

DeMorgan’sTheorem

•A key theorem in simplifying Boolean algebra expression is DeMorgan’sTheorem. It states:

$(a + b)' = a'b'$ $(ab)' = a' + b'$

•Complement the expression $a(b + z(x + a'))$ and simplify.

$$
\begin{aligned}
(a(b+z(x + a')))' &= a' + (b + z(x + a'))' \\
&= a' + b'(z(x + a'))' \\
&= a' + b'(z' + (x + a')') \\
&= a' + b'(z' + x'a'') \\
&= a' + b'(z' + x'a)
\end{aligned}
$$

•Basic logic functions can be made from AND, OR, and NOT (invert) functions
•The behavior of digital circuits can be represented with waveforms, truth tables, or symbols
•Primitive gates can be combined to form larger circuits
•Boolean algebra defines how binary variables can be combined
•Rules for associativity, commutativity, and distribution are similar to algebra
•DeMorgan’srules are important.
Will allow us to reduce circuit sizes.

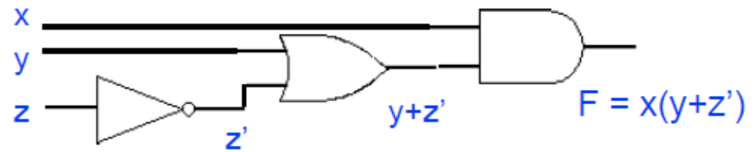
The following table lists the most basic relations of Boolean algebra. All the relations can be proven by means of truth tables:

X+0=X	**X.0=0**
X+1=1	**X.1=X**
X+X=X	**X.X=X**
X+X’=1	**X.X’=0**
X+Y=Y+X	**XY=YX**
X+(Y+Z)=(X+Y)+Z	**X(YZ)=(XY)Z**
X(Y+Z)=XY+XZ	**X+YZ=(X+Y)(X+Z)**
(X+Y)’=X’Y’	**(XY)’=X’+Y’**
X’’=X	**X+X’Y=X+Y**
X+XY=X	**X(X+Y)=X**
XY+XY’=X	**(X+Y)(X+Y’)=X**

Boolean Functions

•Boolean algebra deals with binary variables and logic operations.
•Function results in binary 0 or 1

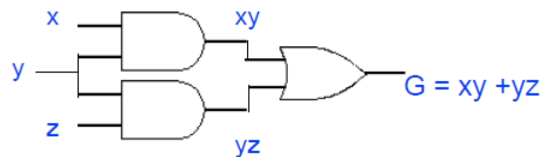
x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1



F = x(y+z')

•Boolean algebra deals with binary variables and logic operations.
•Function results in binary 0 or 1

x	y	z	xy	yz	G
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	0	0
1	1	0	1	0	1
1	1	1	1	1	1



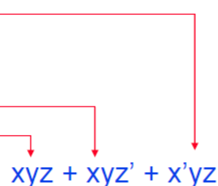
We will learn how to transition between equation, symbols, and truth table.

Truth Table to Expression

•Converting a truth table to an expression
•Each row with output of 1 becomes a product term
•Sum product terms together.

x	y	z	G
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Any Boolean Expression can be represented in sum of products form!

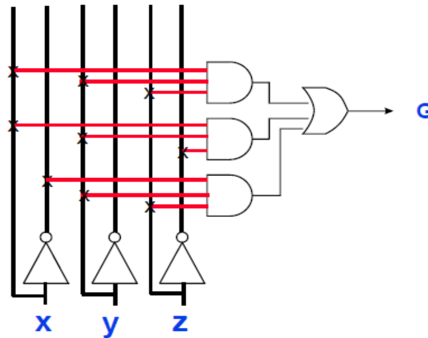


Equivalent Representations of Circuits

•All three formats are equivalent

•Number of 1's in truth table output column equals AND terms for Sum-of-Products (SOP)

x	y	z	G
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



G = xyz + xyz' + x'yz

Reducing Boolean Expressions

•Is this the smallest possible implementation of this expression? No!

•Use Boolean algebra rules to reduce complexity while preserving functionality.

•Step 1: Use Theorum1 (a + a = a)

So xyz + xyz'+ x'yz= xyz + xyz + xyz'+ x'yz

•Step 2: Use distributive rule a(b + c) = ab+ ac

So xyz+ xyz+ xyz'+ x'yz= xy(z+ z') + yz(x+ x')

•Step 3: Use Postulate 3 (a + a'= 1)

So xy(z+ z') + yz(x+ x') = xy.1 + yz.1

•Step 4: Use Postulate 2 (a . 1 = a)

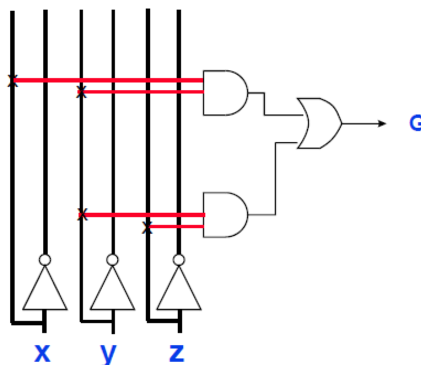
So xy.1 + yz.1 = xy+ yz= xyz + xyz'+ x'yz

Reduced Hardware Implementation

•Reduced equation requires less hardware!

•Same function implemented!

x	y	z	G
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



G = xyz + xyz' + x'yz = xy + yz

Sum of Product (SOP) and Product of Sum (POS)

Minterms and Maxterms

•Each variable in a Boolean expression is a literal
•Boolean variables can appear in normal (x) or complement form (x')
•Each AND combination of terms is a minterm
•Each OR combination of terms is a maxterm

For example:
Minterms

x	y	z	Minterm	
0	0	0	x'y'z'	m_0
0	0	1	x'y'z	m_1
…				
1	0	0	xy'z'	m_4
…				
1	1	1	xyz	m_7

For example:
Maxterms

x	y	z	Maxterm	
0	0	0	x+y+z	M_0
0	0	1	x+y+z'	M_1
…				
1	0	0	x'+y+z	M_4
…				
1	1	1	x'+y'+z'	M_7

Representing Functions with Minterms

•Minterm number same as row position in truth table (starting from top from 0)
•Shorthand way to represent functions

x	y	z	G
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

G = xyz + xyz' + x'yz

$G = m_7 + m_6 + m_3 = \Sigma(3, 6, 7)$

This format is called Sum Of Product (SOP)

Complementing Functions

- Minterm number same as row position in truth table (starting from top from 0)
- Shorthand way to represent functions

x	y	z	G	G'
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	1
1	1	0	1	0
1	1	1	1	0

G = xyz + xyz' + x'yz

G' = (xyz + xyz' + x'yz)'

Can we find a simpler representation?

Complementation Example

- Find complement of F = x'z+ yz   This format is called sum of product
  F'= (x'z+ yz)'
- DeMorgan's
  F'= (x'z)'(yz)'
- DeMorgan's
  F'= (x''+z')(y'+z')
- Reduction -> eliminate double negation on x
  F'= (x+z')(y'+z') This format is called product of sums

Conversion Between Canonical Forms

- Easy to convert between minterm and maxterm representations
- For maxterm representation, select rows with 0's

x	y	z	G	
0	0	0	0	←
0	0	1	0	←
0	1	0	0	←
0	1	1	1	
1	0	0	0	←
1	0	1	0	←
1	1	0	1	
1	1	1	1	

G = xyz + xyz' + x'yz

↓

$G = m_7 + m_6 + m_3 = \Sigma(3, 6, 7)$

↓

$G = M_0M_1M_2M_4M_5 = \Pi(0,1,2,4,5)$

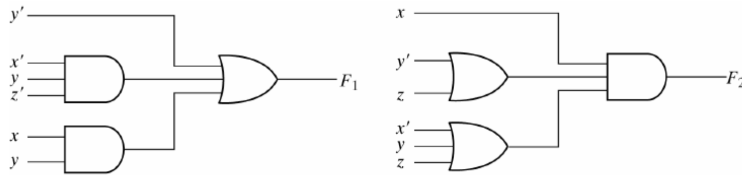
↓

G = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z)(x'+y+z')

This format is called Product Of Sum (POS)

Representation of Circuits

- All logic expressions can be represented in 2-level format
- Circuits can be reduced to minimal 2-level representation
- Sum of products representation most common in industry.



(a) Sum of Products (b) Product of Sums

Two-level implementation

Q) Assume the output (1,0,1,0, 1,0,1,0):

1- Construct the truth table

2- Find the Expression using sum of product then simplify the expression and draw the logic circuit diagram

3- Find the Expression using product of sum

1)

Inputs			Output		
A	B	C	F		
0	0	0	1	A'B'C'	m_0
0	0	1	0		m_1
0	1	0	1	A'BC'	m_2
0	1	1	0		m_3
1	0	0	1	AB'C'	m_4
1	0	1	0		m_5
1	1	0	1	ABC'	m_6
1	1	1	0		m_7

2) F= A'B'C' + A'BC' + AB'C' + ABC' $=m_0+m_2+m_4+m_6=\sum(0,2,4,6)$

=A'C'(B'+B)+AC'(B'+B)= A'C'(1)+AC'(1)=A'C'+AC'=C'(A'+A)= C'(1)=C'

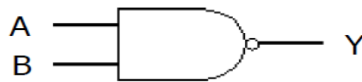
3)

Inputs			Output		
A	B	C	F		
0	0	0	1		M_0
0	0	1	0	A+B+C'	M_1
0	1	0	1		M_2
0	1	1	0	A+B'+C'	M_3
1	0	0	1		M_4
1	0	1	0	A'+B+C'	M_5
1	1	0	1		M_6
1	1	1	0	A'+B'+C'	M_7

F= (A+B+C') (A+B'+C') (A'+B+C') (A'+B'+C')$=M_1M_3M_5M_7=\prod(1,3,5,7)$

NAND and NOR Gates

The NAND Gate



- This is a NAND gate. It is a combination of an AND gate followed by an inverter. Its truth table shows this...
- NAND gates have several interesting properties...
  - NAND(a,a)=(aa)' = a' = NOT(a)
  - NAND'(a,b)=(ab)'' = ab = AND(a,b)
  - NAND(a',b')=(a'b')' = a+b = OR(a,b)

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

The NAND Gate

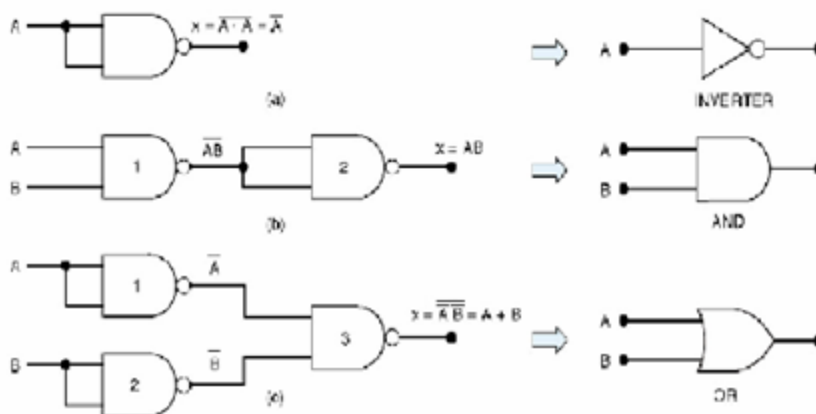
•These three properties show that a NAND gate with both of its inputs driven by the same signal is equivalent to a NOT gate

•A NAND gate whose output is complemented is equivalent to an AND gate, and a NAND gate with complemented inputs acts as an OR gate.

•Therefore, we can use a NAND gate to implement all three of the *elementary operators* (AND,OR,NOT).

•Therefore, ANY switching function can be constructed using only NAND gates. Such a gate is said to be *primitive* or *functionally complete*.

Universality of NAND gates



The NOR Gate

- This is a NOR gate. It is a combination of an OR gate followed by an inverter. It's truth table shows this…
- NOR gates also have several interesting properties…
  - NOR(a,a)=(a+a)' = a' = NOT(a)
  - NOR'(a,b)=(a+b)'' = a+b = OR(a,b)
  - NOR(a',b')=(a'+b')' = ab = AND(a,b)

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Functionally Complete Gates

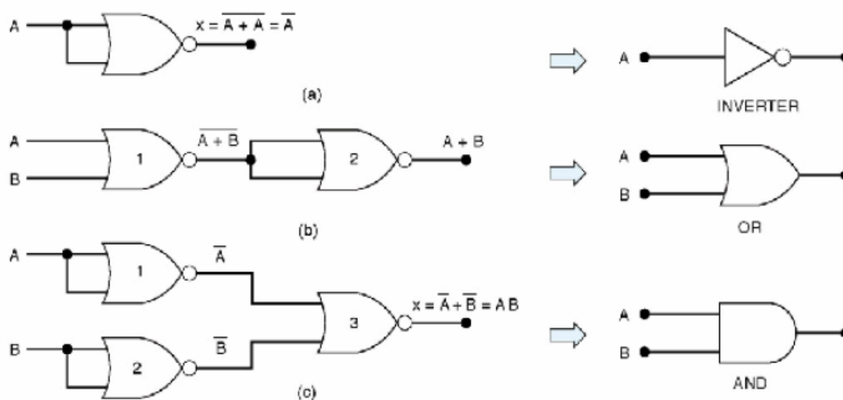
•Just like the NAND gate, the NOR gate is functionally complete…any logic function can be implemented using just NOR gates.

•Both NAND and NOR gates are very valuable as any design can be realized using either one.

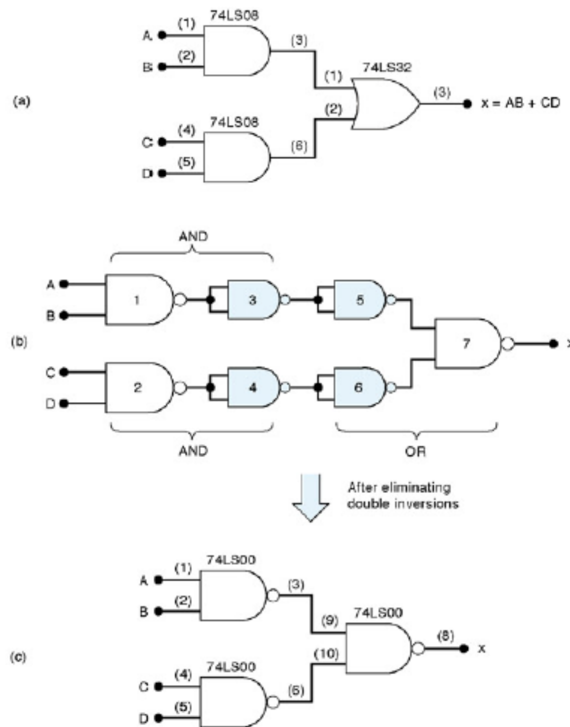
•It is easier to build an IC chip using all NAND or NOR gates than to combine AND,OR, and NOT gates.

•NAND/NOR gates are typically faster at switching and cheaper to produce.

Universality of NOR gate



Example



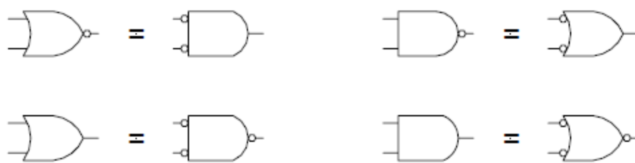
NAND and XOR Implementations Combinational Design Procedure

NAND-NAND & NOR-NOR Networks

DeMorgan's Law:

$(a + b)' = a' b'$ $(a b)' = a' + b'$

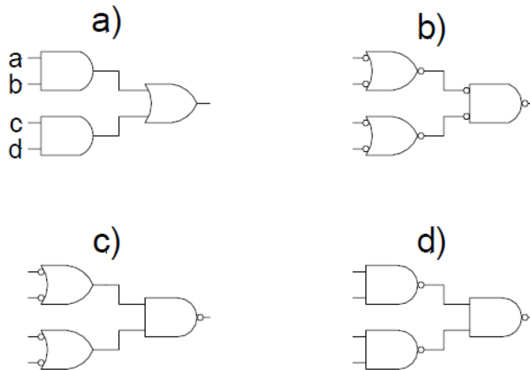
$a + b = (a' b')'$ $(a b) = (a' + b')'$



push bubbles **or** ***introduce in pairs*** **or** ***remove pairs.***

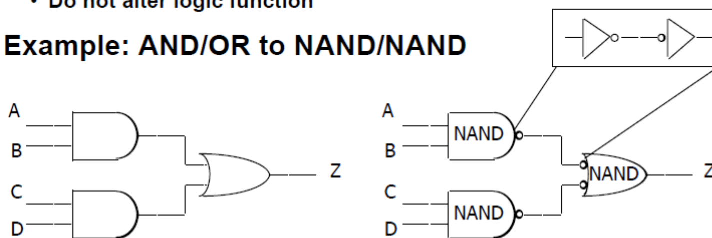
NAND-NAND Networks

- Mapping from AND/OR to NAND/NAND



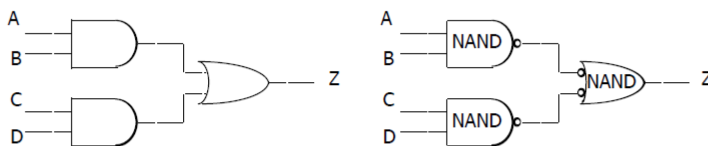
Conversion Between Forms

- **Convert from networks of ANDs and ORs to networks of NANDs and NORs**
  - Introduce appropriate inversions ("bubbles")
- **Each introduced "bubble" must be matched by a corresponding "bubble"**
  - Conservation of inversions
  - Do not alter logic function
- **Example: AND/OR to NAND/NAND**



Conversion Between Forms (cont'd)

- **Example: verify equivalence of two forms**



$$
\begin{aligned}
Z &= [\,(A \cdot B)' \cdot (C \cdot D)'\,]' \\
&= [\,(A' + B') \cdot (C' + D')\,]' \\
&= [\,(A' + B')' + (C' + D')'\,] \\
&= (A \cdot B) + (C \cdot D) \checkmark
\end{aligned}
$$

Q) Convert to NAND gate only then to NOR gate only

1) X=(B+C)(A+D)

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

$=\overline{\overline{\overline{\overline{(B+C)}}\;\overline{\overline{(A+D)}}}}$

$=\overline{\overline{\overline{(\overline{B}\,\overline{C})}\;\overline{(\overline{A}\,\overline{D})}}}$

$=\overline{\overline{\overline{(\overline{BB}\,\overline{CC})}\;\overline{(\overline{AA}\,\overline{DD})}}}$

X=(B+C)(A+D)

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

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

2) $F=(\overline{A}B)+(\overline{C}D)$

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

$=\overline{\overline{(\overline{A}B)}\;\overline{(\overline{C}D)}}$

$=\overline{\overline{(\overline{AA}B)}\;\overline{(\overline{CC}D)}}$

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

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

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

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

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

3) K=(AB)+(C+D)

$$=\overline{\overline{(AB)+(C+D)}}$$

$$=\overline{\overline{(AB)}\,\overline{(C+D)}}$$

$$=\overline{\overline{(AB)}\,\overline{\overline{\overline{(C+D)}}}}$$

$$=\overline{\overline{(AB)}\,\overline{\overline{(\overline{C}\,\overline{D})}}}$$

$$=\overline{\overline{(AB)}\,\overline{\overline{(\overline{CC}\,\overline{DD})}}}$$

K=(AB)+(C+D)

$$=\overline{\overline{(AB)+(C+D)}}$$

$$=\overline{\overline{\overline{\overline{(AB)}}+\overline{\overline{(C+D)}}}}$$

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

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

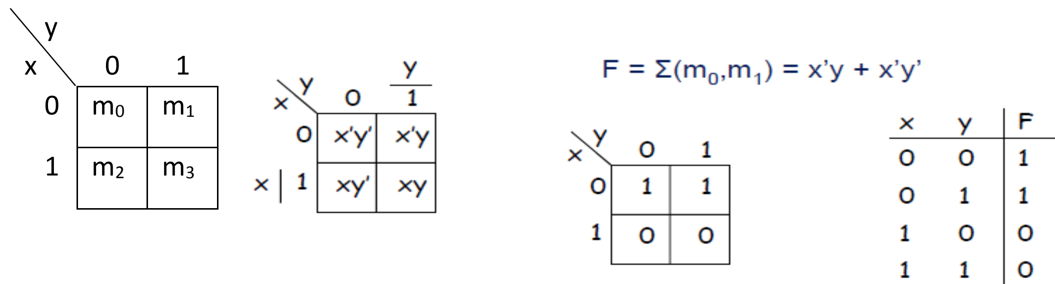
Minimization with Karnaugh Maps

Karnaugh maps

- Alternate way of representing Boolean function
  - •All rows of truth table represented with a square
  - •Each square represents a minterm
- Easy to convert between truth table, K-map, and SOP
  - •Unoptimized form: number of 1's in K-map equal number of minterms (products) in SOP
  - •Optimized form: reduced number of minterms

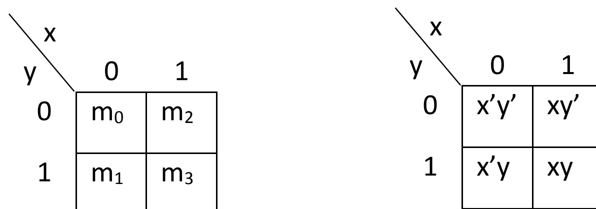
Two variable maps

It can be represented as follow:

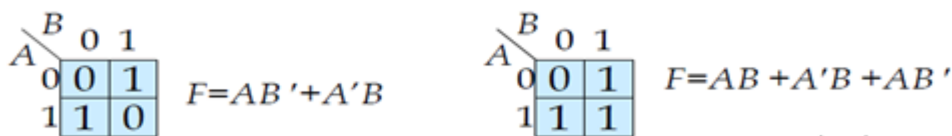


x	y	F
0	0	1
0	1	1
1	0	0
1	1	0

Or represented as follow:



•A Karnaugh map is a graphical tool for assisting in the general simplification procedure.



Three variable maps

It can be represented as follow:

A \ BC	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

A \ BC	00	01	11	10
0	A’B’C’	A’B’C	A’BC	A’BC’
1	AB’C’	AB’C	ABC	ABC’

Or represented as follow:

BC \ A	0	1
00	m_0	m_4
01	m_1	m_5
11	m_3	m_7
10	m_2	m_6

BC \ A	0	1
00	A’B’C’	AB’C’
01	A’B’C	AB’C
11	A’BC	ABC
10	A’BC’	ABC’

A \ BC	00	01	11	10
0	0	1	0	1
1	1	1	1	1

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

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

Rules for K-Maps

- We can reduce functions by circling 1's in the K-map
- Each circle represents minterm reduction
- Following circling, we can deduce minimized and-or form.

Rules to consider

- Every cell containing a 1 must be included at least once.
- The largest possible "power of 2 rectangle must be enclosed.
- The 1's must be enclosed in the smallest possible number of rectangles.

A Karnaugh map is a graphical tool for assisting in the general simplification procedure.

Two variable maps.

A \ B	0	1
0	0	1
1	1	0

$F = AB' + A'B$

A \ B	0	1
0	0	1
1	1	1

$F = AB + A'B + AB'$

$F = A + B$

Three variable maps.

A \ BC	00	01	11	10
0	0	1	0	1
1	1	1	1	1

$F = A + B'C + BC'$

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

Karnaugh Maps for Four Input Functions

- Represent functions of 4 inputs with 16 minterms
- Use same rules developed for 3-input functions
- Note bracketed sections shown in example.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

wx \ yz	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

(y spans columns 11 and 10; z spans columns 01 and 11; x spans rows 01 and 11; w spans rows 11 and 10)

'or represented as follow:

YZ \ WX	00	01	11	10
00	m_0	m_4	m12	m_8
01	m_1	m_5	m_{13}	m_9
11	m_3	m_7	m_{15}	m_{11}
10	m_2	m_6	m_{14}	m_{10}

YZ \ WX	00	01	11	10
00	W'X'Y'Z'	W'XY'Z'	WXY'Z'	WX'Y'Z'
01	W'X'Y'Z	W'XY'Z	WXY'Z	WX'Y'Z
11	W'X'YZ	W'XYZ	WXYZ	WX'YZ
10	W'X'YZ'	W'XYZ'	WXYZ'	WX'YZ'

Karnaugh map: 4-variable example

$F(A,B,C,D) = \Sigma m(0,2,3,5,6,7,8,10,11,14,15)$

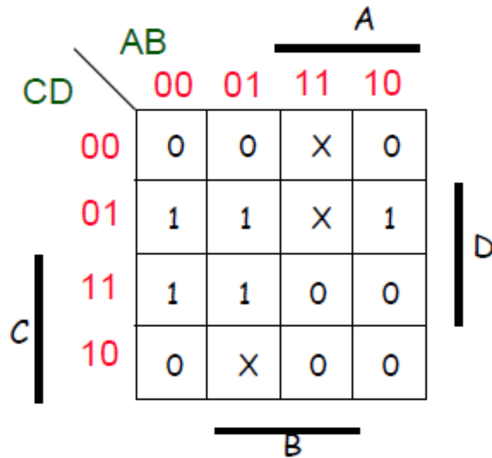
Inputs				F
A	B	C	D	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	0	0	1	1
10	1	0	1	1

F=C+A'BD+B'D'

Karnaugh maps: Don't cares

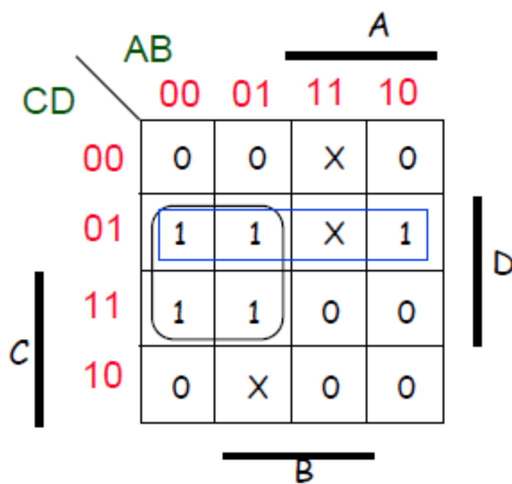
- In some cases, outputs are undefined
- We "don't care" if the logic produces a 0 or a 1
- This knowledge can be used to simplify functions.



- Treat X's like either 1's or 0's
- Very useful
- OK to leave some X's uncovered

- $f(A,B,C,D) = \Sigma\, m(1,3,5,7,9) + d(6,12,13)$
  - **without don't cares**
    - **f =**

A'D + C'D



A	B	C	D	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	X
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	X
1	1	0	1	X
1	1	1	0	0
1	1	1	1	0

Don’t Care Conditions

- In some situations, we don’t care about the value of a function for certain combinations of the variables.
  - •these combinations may be impossible in certain contexts
  - •or the value of the function may not matter in when the combinations occur
- In such situations we say the function is *incompletely specified* and there are multiple (completely specified) logic functions that can be used in the design.
  - •so we can select a function that gives the simplest circuit
- When constructing the terms in the simplification procedure, we can choose to either cover or not cover the don’t care conditions.

Map Simplification with Don’t Cares

AB \ CD	00	01	11	10
00	0	1	0	0
01	x	x	x	1
11	1	1	1	x
10	x	0	1	1

$F = A'C'D + B + AC$

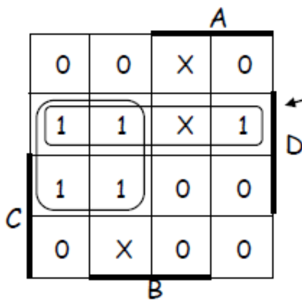
•Alternative covering.

AB \ CD	00	01	11	10
00	0	1	0	0
01	x	x	x	1
11	1	1	1	x
10	x	0	1	1

$F = A'B'C'D + ABC' + BC + AC$

- **f(A,B,C,D) = Σ m(1,3,5,7,9) + d(6,12,13)**
  - **f = A'D + B'C'D** **without don't cares**
  - **f =** **with don't cares**

A'D + C'D

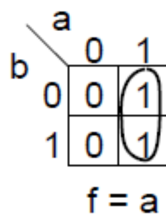


by using don't care as a "1"
a 2-cube can be formed
rather than a 1-cube to cover
this node

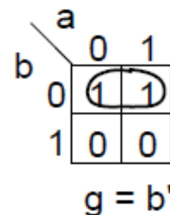
don't cares can be treated as 1s or 0s depending on which is more advantageous

More KarnaughMap Examples

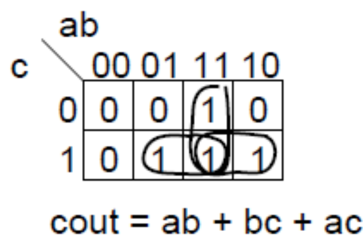
- **Examples**



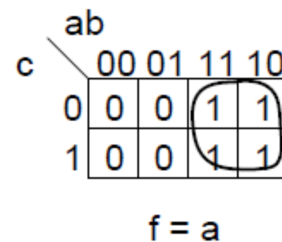
f = a



g = b'



cout = ab + bc + ac



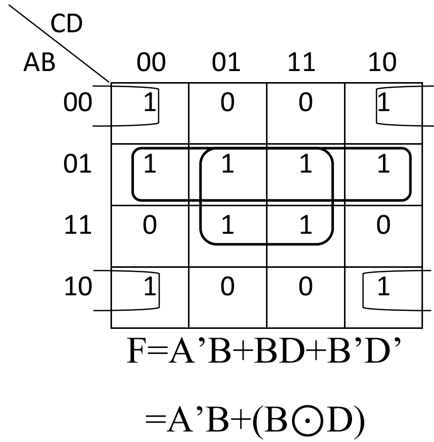
f = a

1. Circle the largest groups possible.
2. Group dimensions must be a power of 2.
3. Remember what circling means!

Q) Use a Karnaugh map to reduce the expression to a minimum SOP form

F(A,B,C,D)= ∑ (0,2,4,5,6,7,8,10,13,15)

Inputs				F
A	B	C	D	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1



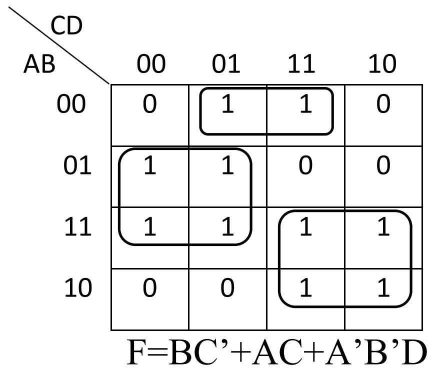
F=A’B+BD+B’D’

=A’B+(B⊙D)

Q) Use a Karnaugh map to reduce the expression to a minimum SOP form

F(A,B,C,D)=∑(1,3,4,5,10-15)

Inputs				F
A	B	C	D	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

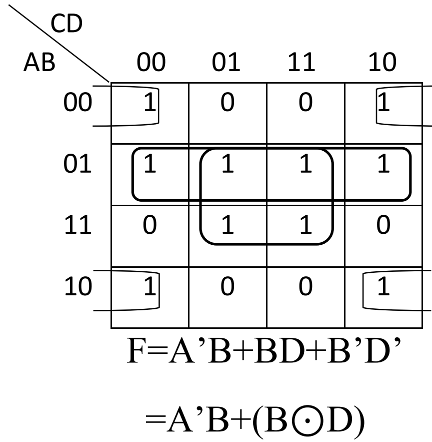


F=BC’+AC+A’B’D

Q) Use a Karnaugh map to reduce the expression to a minimum SOP form

F(A,B,C,D)= $\prod(1,3,9,11,12,14)$

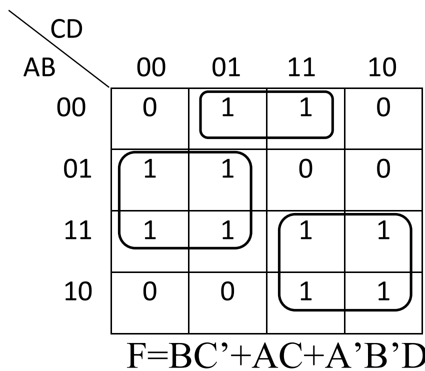
Inputs				F
A	B	C	D	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1



Q) Use a Karnaugh map to reduce the expression to a minimum SOP form

F(A,B,C,D)= $\prod(0,2,6\text{-}9)$

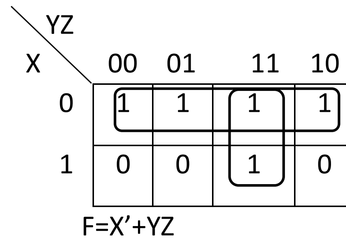
Inputs				F
A	B	C	D	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



Q) Use a Karnaugh map to reduce the expression to a minimum SOP form

F(X,Y,Z)= X'Y'+YZ+X'YZ'

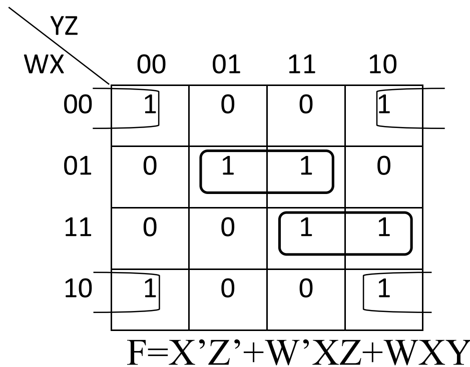
Inputs			Output
X	Y	Z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1



Q) Use a Karnaugh map to reduce the expression to a minimum SOP form

F(W,X,Y,Z)= WXY+X'Z'+W'XZ

Inputs				F
W	X	Y	Z	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

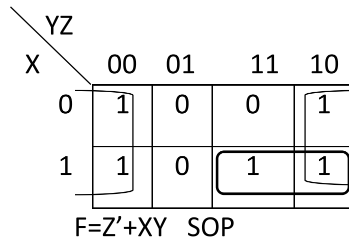


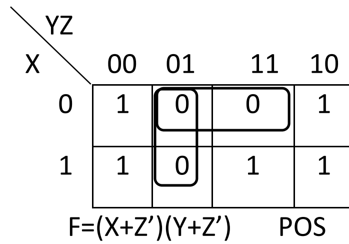
F=X'Z'+W'XZ+WXY

Q) Use a Karnaugh map to reduce the expression to a minimum SOP and minimum POS form

F(X,Y,Z)= X’Z’+Y’Z’+YZ’+XY

Inputs			Output
X	**Y**	**Z**	**F**
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

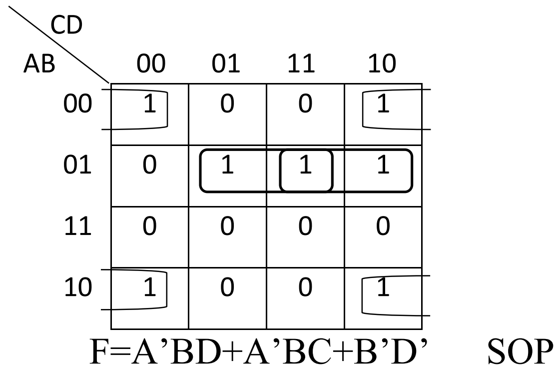


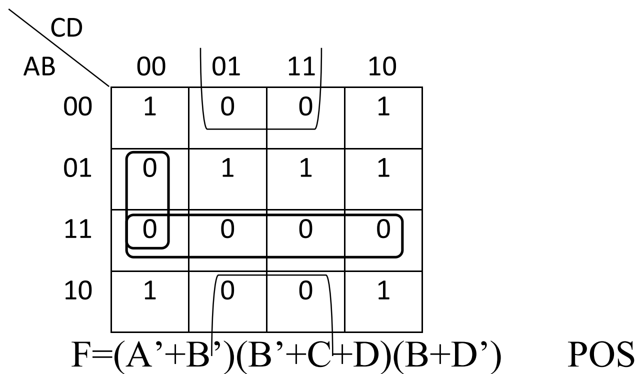


Q) Use a Karnaugh map to reduce the expression to a minimum SOP and minimum POS form, then draw both logic circuit

F(A,B,C,D)= $\sum(0,2,5,6,7,8,10)$

Inputs				F
A	**B**	**C**	**D**	
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

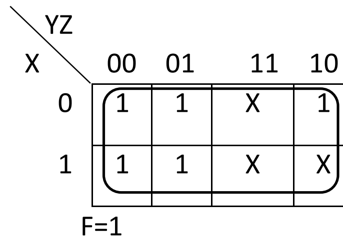




Q) Use a Karnaugh map to reduce the expression to a minimum SOP form

F(X,Y,Z)= ∑m(0,1,2,4,5)+ d(3,6,7)

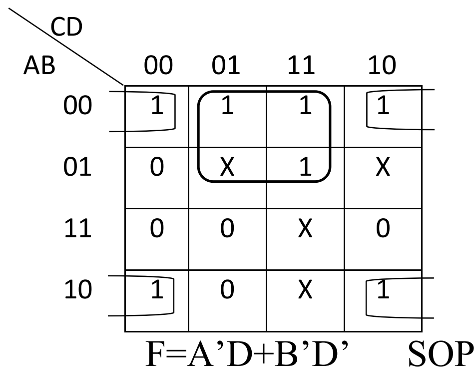
Inputs			Output
X	**Y**	**Z**	**F**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X



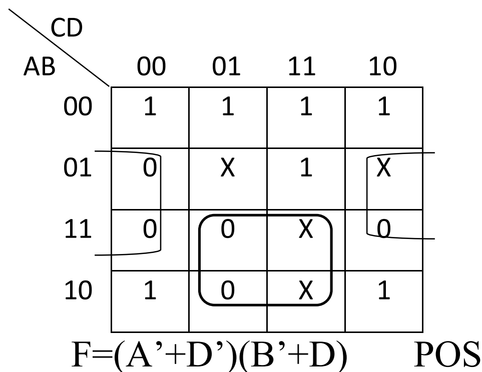
Q) Use a Karnaugh map to reduce the expression to a minimum SOP and minimum POS form, then draw both logic circuit

F(A,B,C,D)= ∑m(0,1,2,3,7,8,10)+d(5,6,11,15)

Inputs				F
A	B	C	D	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	X
0	1	1	0	X
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	X
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	X



F=A’D+B’D’ SOP



F=(A’+D’)(B’+D) POS

Exclusive-OR and Exclusive-NOR Circuits

Exclusive-OR (XOR) produces a HIGH output whenever the two inputs are at opposite levels.

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

A	B	x
0	0	0
0	1	1
1	0	1
1	1	0

(a)

XOR gate symbols

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

(b)

$x = A \oplus B$

(c)

Controlled Inverter:

The XOR gate can be used as a "NOT" gate by connecting one of the inputs to the logic (1), for this reason it can be used to complement the 1st input by using the 2nd input as control line, when control signal is logic (0) then, X = A. When control signal is logic (1) then, **X** = **A'**

Control Signal

Exclusive-NOR (XNOR): Exclusive-NOR (XNOR) produces a HIGH output whenever the two inputs are at the same level.

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

A	B	x
0	0	1
0	1	0
1	0	0
1	1	1

(a)

XNOR gate symbols

$x = \overline{A \oplus B} = AB + \bar{A}\bar{B}$

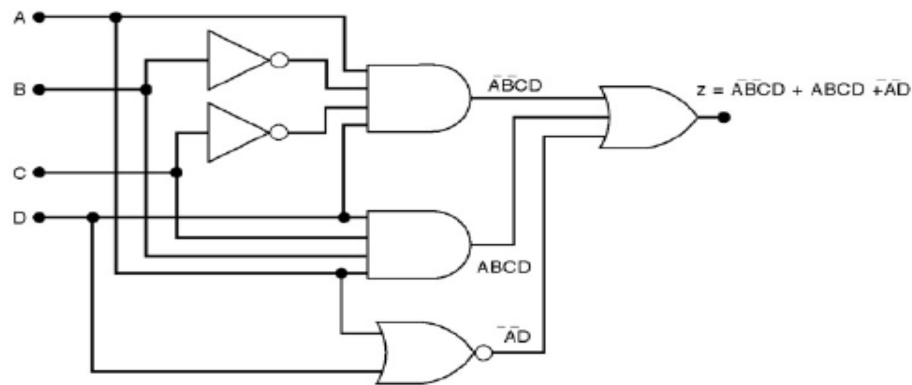
(b)

$x = \overline{A \oplus B}$

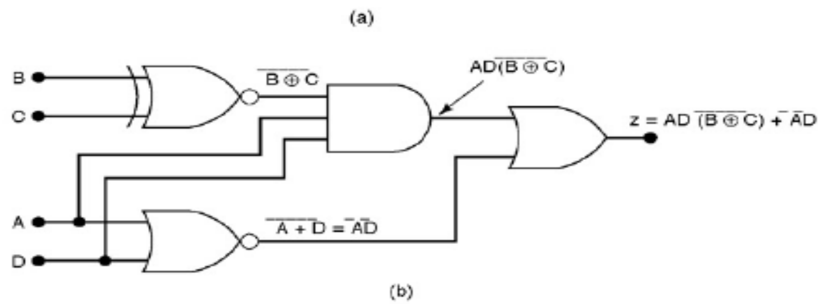
(c)

Exclusive-NOR Circuits

XNOR gate may be used to simplify circuit implementation.



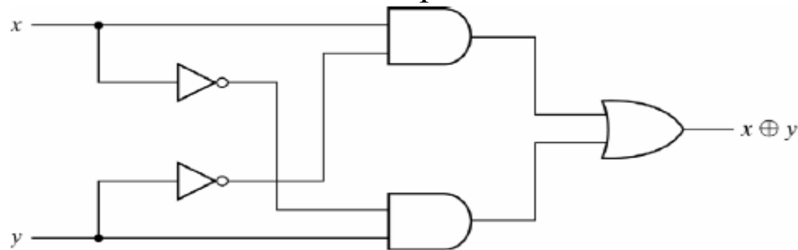
(a)



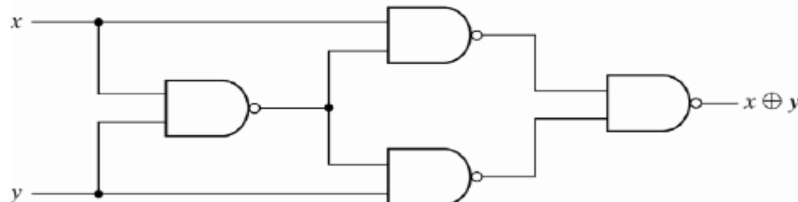
(b)

XOR Function

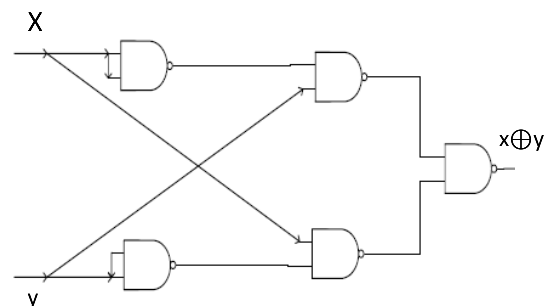
XOR function can also be implemented with AND/OR gates (also NANDs).



(a) With AND-OR-NOT gates



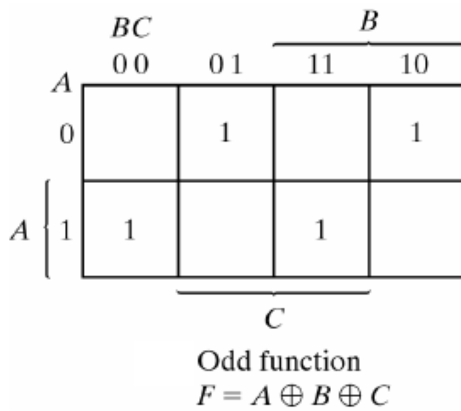
(b) With NAND gates



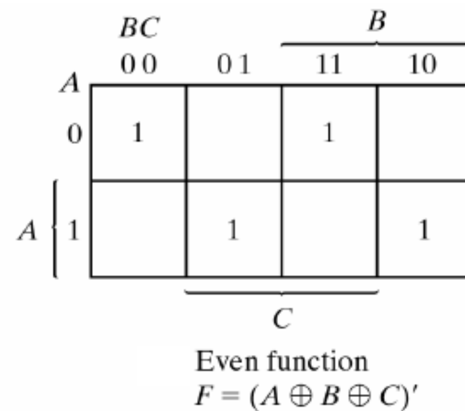
XOR Function

- Even function –even number of inputs are 1, the output will be 1.
- Odd function –odd number of inputs are 1, the output will be 1.

A	B	C	Even Function	Odd Function
0	0	0	1	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	0	1



Odd function
$F = A \oplus B \oplus C$



Even function
$F = (A \oplus B \oplus C)'$

Map for a Three-variable Exclusive-OR Function

Odd Function:

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

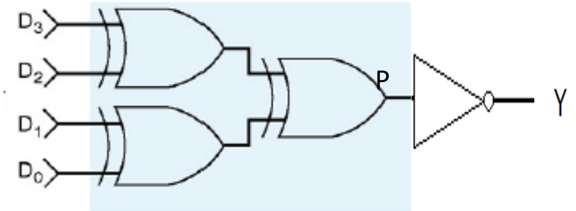
=C(A'B'+AB)+C'(A'B+AB')

$=C(A \oplus B)'+C'(A \oplus B)$

$=A \oplus B \oplus C$

Q) Design even and odd parity

- Even parity – even number of inputs are 1, the output will be 0.
- Odd parity –odd number of inputs are 1, the output will be 0.



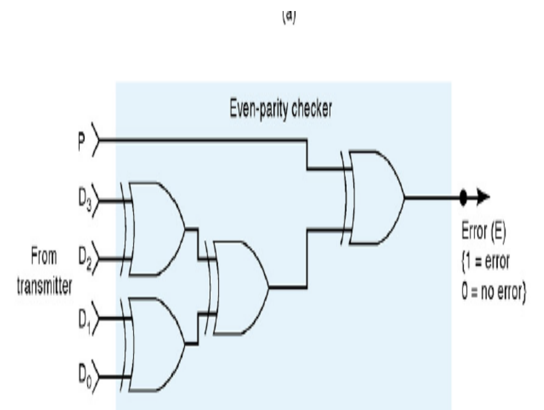
D_3	D_2	D_1	D_0	Even parity P	Odd parity Y
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

$P = D'_3D'_2D'_1D_0 + D'_3D'_2D_1D'_0 + D'_3D_2D'_1D'_0 + D'_3D_2D_1D_0 + D_3D_2D'_1D_0 + D_3D_2D_1D'_0 + D_3D'_2D'_1D'_0 + D_3D'_2D_1D_0$

$= (D_3 \oplus D_2) \oplus (D_1 \oplus D_0)$

$Y = P'$

D_3	D_2	D_1	D_0	Even parity P	Even Parity Checker E
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0



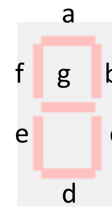
$E=P\oplus((D_3\oplus D_2)\oplus(D_1\oplus D_0))$

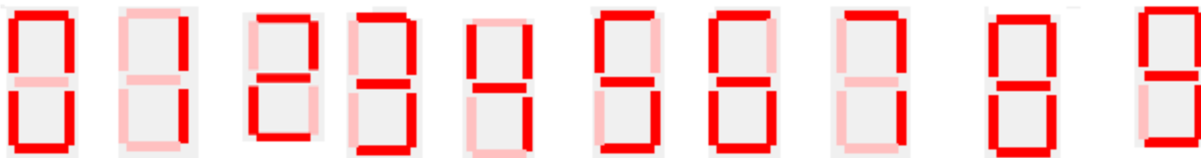
BCD to Seven Segment

- Used to display binary coded decimal (BCD) numbers using seven illuminated segments.
- BCD uses 0's and 1's to represent decimal digits 0 -9. Need four bits to represent required 10 digits.
- Binary coded decimal (BCD) represents each decimal digit with four bits

List the segments that should be illuminated for each digit.

0: a,b,c,d,e,f
1: b,c
2: a,b,d,e,g
3: a,b,c,d,g
4: b,c,f,g
5: a,c,d,f,g
6: a,c,d,e,f,g
7: a,b,c
8: a,b,c,d,e,f,g
9: a,b,c,d,f,g





- **Derive the truth table for the circuit. Each output column in one circuit.**

No.	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

- Find minimal sum-of-products representation for each output

For segment "a":

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11				
10	1	1		

Note: Have only filled in ten squares, corresponding to the ten numerical digits we wish to represent.

- Fill in don't cares for undefined outputs. Leads to a reduced implementation
- Note that these combinations of inputs should never happen.

Put in "X" (don't care), and interpret as either 1 or 0 as desired ….

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

- Circle biggest group of 1's and Don't Cares. Leads to a reduced implementation. All 1's should be covered by at least one implicant
- Put all the terms together
- Generate the circuit

AB \ CD	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	X	X	X	X
10	1	1	X	X

a=A+C+BD+B'D'

=A+C+(B⊙D)

AB \ CD	00			
00	1	1	1	1
01	1	0	1	0
11	X	X	X	X
10	1	1	X	X

b=B'+C'D'+CD

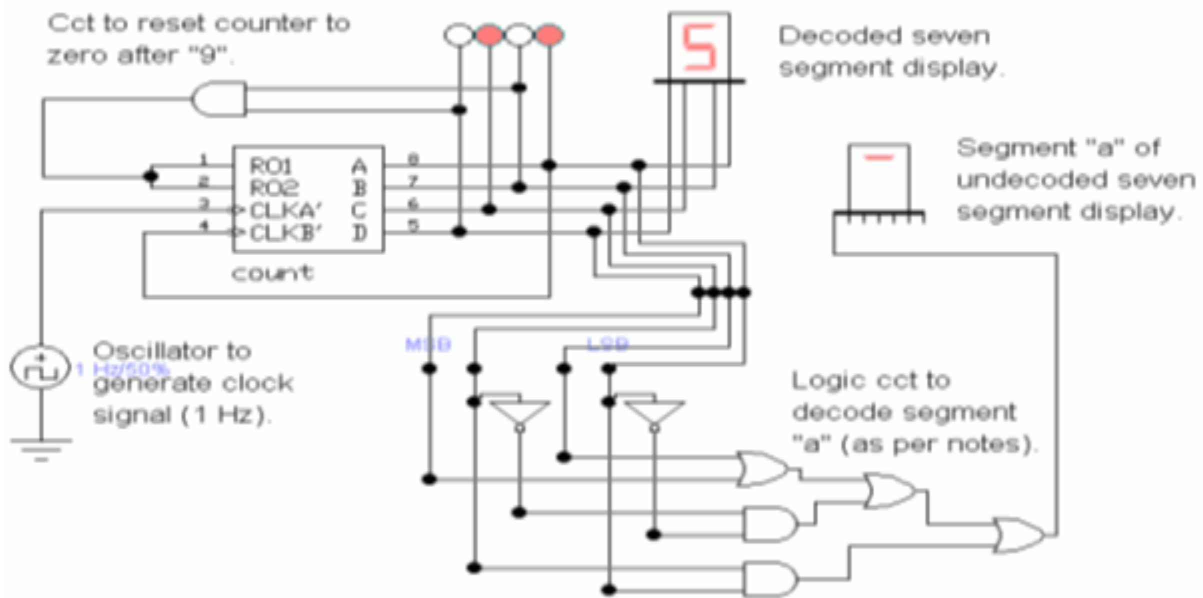
=B'+(C⊙D)

AB \ CD	00	01	11	10
00	1	1	1	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

c=C'+D+B

Homework

Design circuits for segments d, e, f, and g of seven segments to display BCD numbers



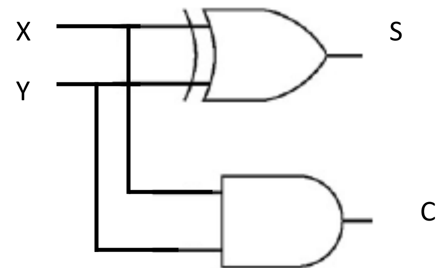
Binary Addition and Subtraction

Half adder

Add two binary numbers

X,Y: single bit inputs

S: single bit sum, C: carry out



X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$S = X'Y + XY' = (X \oplus Y)$

C: XY

Full adder

Full adder includes carry in C_{in}

A	B	C_{in}	C_{out}	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

$$Cout = A'BC_{in} + AB'C_{in} + ABC'_{in} + ABC_{in}$$
$$= A'BC_{in} + ABC_{in} + AB'C_{in} + ABC_{in} + ABC'_{in} + ABC_{in}$$
$$= (A' + A)BC_{in} + (B' + B)AC_{in} + (C'_{in} + C_{in})AB$$
$$= 1 \cdot BC_{in} + 1 \cdot AC_{in} + 1 \cdot AB$$
$$= BC_{in} + AC_{in} + AB$$

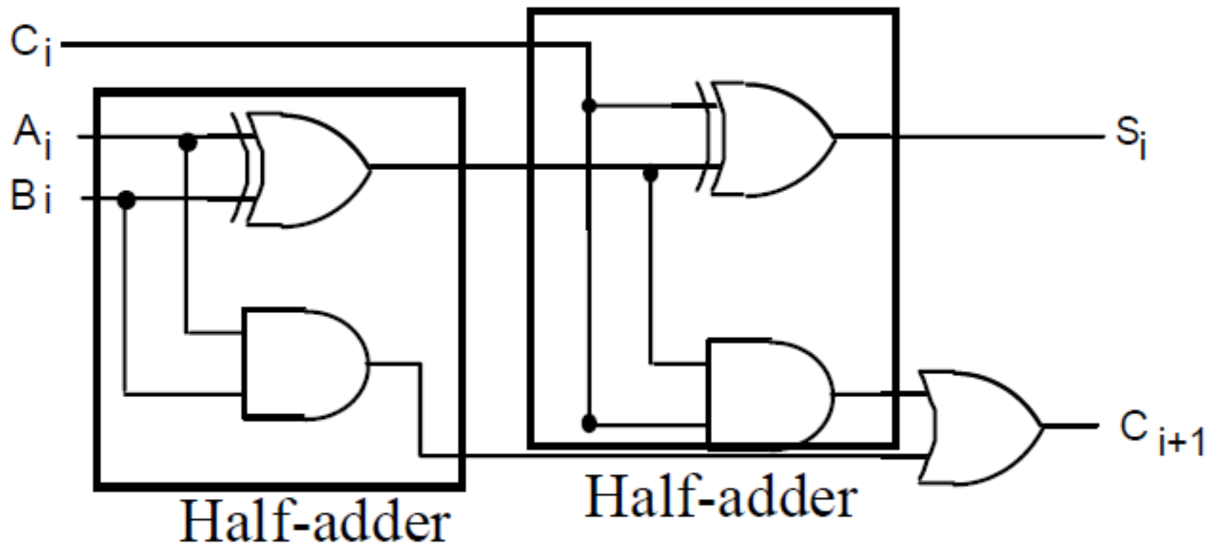
A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$S = A'B'C_{in} + A'BC'_{in} + AB'C'_{in} + ABC_{in}$$
$$= C_{in}(A'B' + AB) + C'_{in}(A'B + AB')$$
$$= C_{in}(A \oplus B)' + C'_{in}(A \oplus B)$$
$$= C_{in} \oplus A \oplus B$$

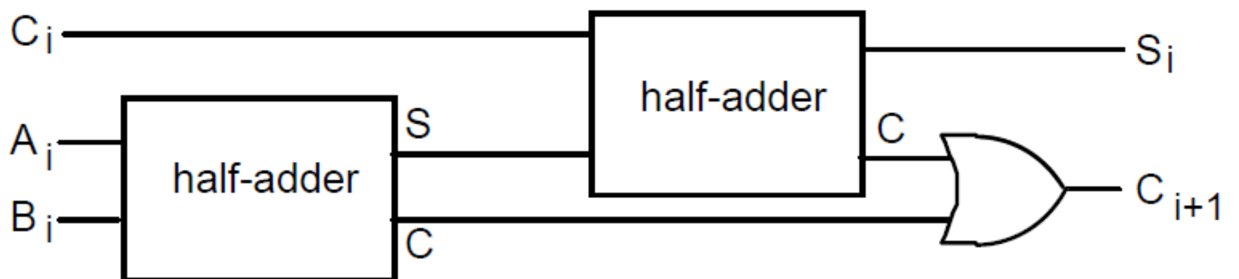
A \ BC	00	01	11	10
0	0	0	1	0
1	0	1	1	1

$$C_{out} = AC_{in} + AB + BC_{in}$$

- Full adder made of several half adders

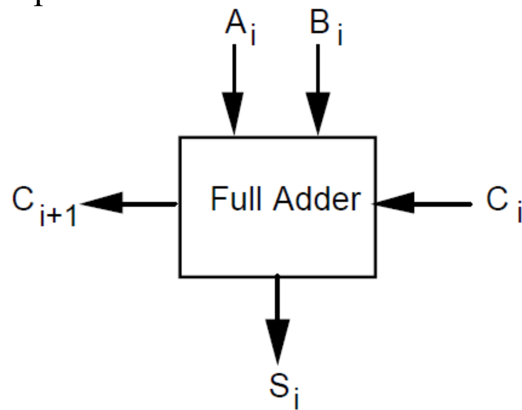


- Hardware repetition simplifies hardware design



A full adder can be made from two half adders (plus an OR gate).

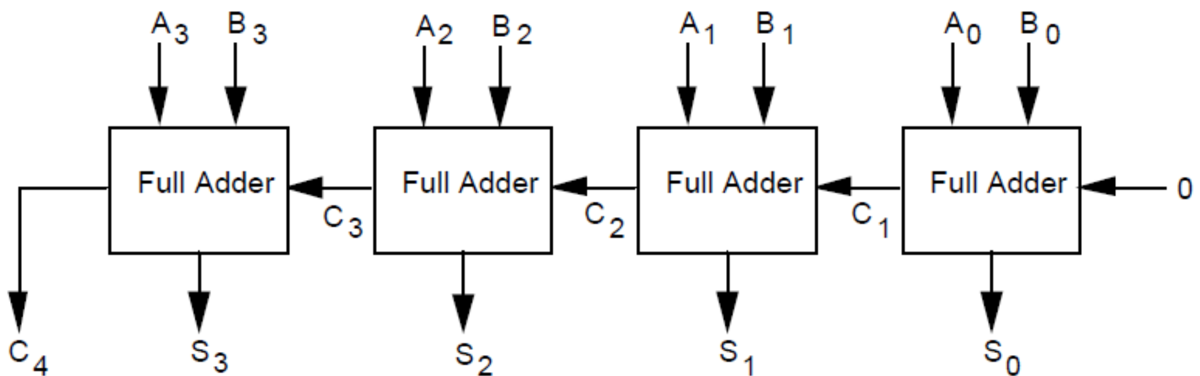
- Putting it all together
  - •Single-bit full adder
  - •Common piece of computer hardware



Block Diagram

4-Bit Adder

Chain single-bit adders together.
What does this do to delay?



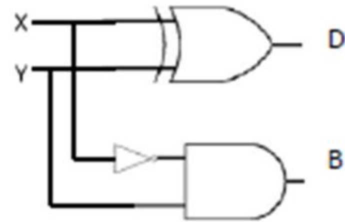
C	1	1	1	0
A	0	1	0	1
B	0	1	1	1
S	1	1	0	0

Half Subtractor

Subtract two binary numbers

X,Y: single bit inputs, D: single bit difference, B: barrow

X	Y	D	B
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



D= X'Y+XY'=(X⊕Y)

B: X'Y

Full subtractor

Full subtractor include barrow: Bin

X	Y	B_{in}	D	B_{out}
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

X \ YB	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$D = X'Y'B_{in} + X'YB'_{in} + XY'B'_{in} + XYB_{in}$

$= B_{in}(X'Y'+XY) + B'_{in}(X'Y+XY')$

$= B_{in}(X \oplus Y)' + B'_{in}(X \oplus Y)$

$= B_{in} \oplus X \oplus Y$

X \ YB	00	01	11	10
0	0	1	1	1
1	0	0	1	0

$B_{out} = X'B_{in} + X'Y + YB_{in}$

Negative Numbers –2’s Complement.

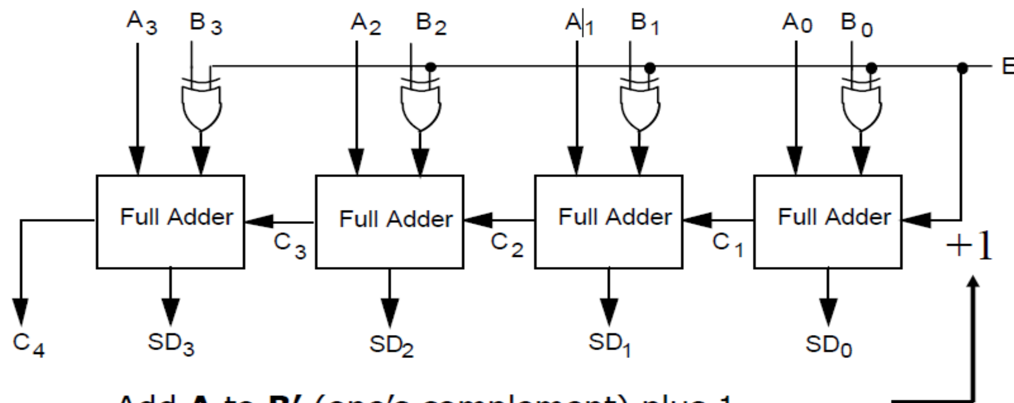
Subtracting a number is the same as:
1. Perform 2’s complement
2. Perform addition

°If we can augment adder with 2’s complement hardware?

$$1_{10} = 01_{16} = 00000001$$
$$-1_{10} = FF_{16} = 11111111$$

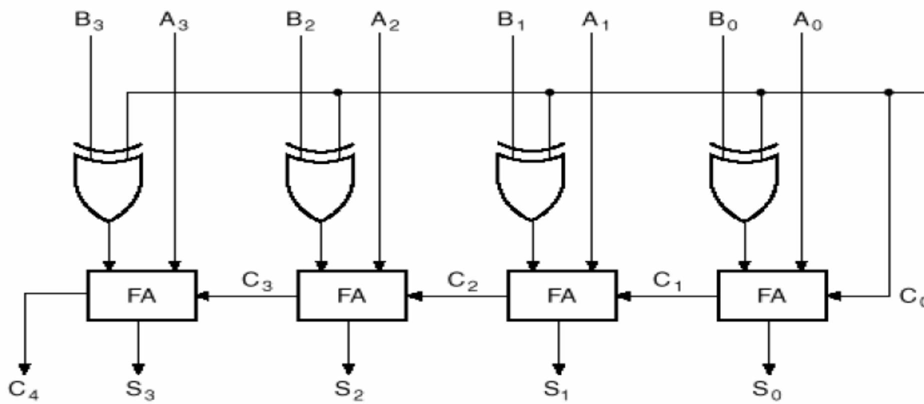
$$128_{10} = 80_{16} = 10000000$$
$$-128_{10} = 80_{16} = 10000000$$

4-bit Subtractor: E = 1



Add **A** to **B’** (one’s complement) plus 1
That is, add **A** to two’s complement of **B**
D = **A** - **B**

Adder-Subtractor Circuit



Note:-

S=0: Addition

S=1: Subtraction

Digital Comparator

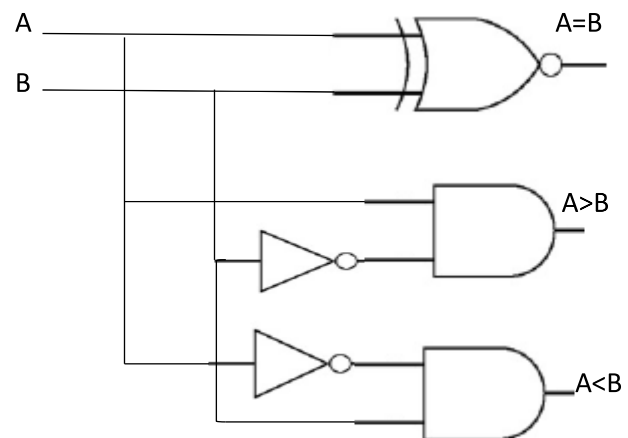
Q) Compare two numbers each number has 1 bit

A	B	A=B	A>B	A<B
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

A=B : $A'B'+AB=(A\oplus B)'=A\odot B$

A>B: AB'

A<B: $A'B$



Q) Compare two numbers each number has 2 bits

A_2	A_1	B_2	B_1	A=B	A>B	A<B
0	0	0	0	1	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	0	1	0
0	1	0	1	1	0	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	1	0
1	0	0	1	0	1	0
1	0	1	0	1	0	0
1	0	1	1	0	0	1
1	1	0	0	0	1	0
1	1	0	1	0	1	0
1	1	1	0	0	1	0
1	1	1	1	1	0	0

$A=A_2A_1$, $B=B_2B_1$

A=B: $(A_2=B_2)$ and $(A_1=B_1)$ $\quad =(A_2\odot B_2).(A_1\odot B_1)$

A>B: $(A_2>B_2)$ or $((A_2=B_2)$ and $(A_1>B_1))$ $\quad =(A_2B'_2)+((A_2\odot B_2).(A_1B'_1))$

A<B: $(A_2<B_2)$ or $((A_2=B_2)$ and $(A_1<B_1))$ $\quad =(A'_2B_2)+((A_2\odot B_2).(A'_1B_1))$

A=B: $A'_2A'_1 B'_2 B'_1+ A'_2A_1 B'_2 B_1+ A_2A'_1 B_2 B'_1+ A_2A_1 B_2 B_1$

$= A'2B'2(A'1B'1+ A_1B_1)+A_2B_2(A'_1B'_1+ A_1B_1)$

$= A'_2B'_2(A_1\odot B_1)+ A_2B_2(A_1\odot B_1)= (A_1\odot B_1)(A'_2B'_2+ A_2B_2)= (A_1\odot B_1) (A_2\odot B_2)$

A<B: $A'_2A'_1B'_2B_1+A'_2A'_1 B_2B'_1+A'_2A'_1B_2B_1+A'_2A_1B_2B'_1+A'_2A_1B_2B_1+A_2A'_1 B_2B_1$

$= A'_2B_2(A'_1B'_1+A'_1B_1+A_1B'_1+A_1B_1)+ A'_2A'_1B'_2B_1+A_2A'_1 B_2B_1$

$= A'_2B_2(A'_1(B'_1+B_1)+A1(B'_1+B_1))+ A'_1B_1(A'_2B'_2+ A_2B_2)$

$= A'_2B_2(A'_1(1)+A1(1))+ A'_1B_1(A_2\odot B_2)$

$= A'_2B_2(A'_1 +A_1) + A'_1B_1(A_2\odot B_2)$

$= A'_2B_2(1) + A'_1B_1(A_2\odot B_2) = (A'_2B_2) + ((A'_1B_1).(A_2\odot B_2))$

A_2A_1 \ B_2B_1	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

A=B: $A'_2A'_1\, B'_2\, B'_1 + A'_2A_1\, B'_2\, B_1 + A_2A'_1\, B_2\, B'_1 + A_2A_1\, B_2\, B_1$

$= A'2B'2(A'1B'1 + A_1B_1) + A_2B_2(A'_1B'_1 + A_1B_1)$

$= A'_2B'_2(A_1 \odot B_1) + A_2B_2(A_1 \odot B_1)$

$= (A_1 \odot B_1)(A'_2B'_2 + A_2B_2)$

$= (A_1 \odot B_1)(A_2 \odot B_2)$

A_2A_1 \ B_2B_1	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

A>B: $A_2\, B'_2 + A_1\, B'_2\, B'_1 + A_2A_1B'_1$

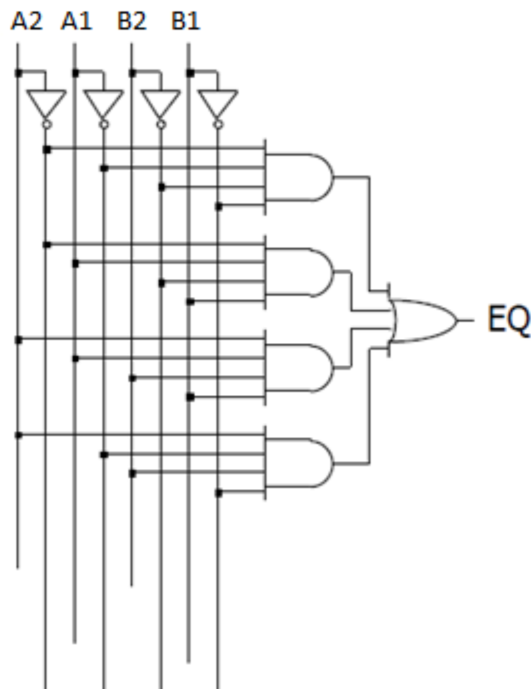
$= A_2\, B'_2 + A_1\, B'_1(B'_2 + A_2)$

A_2A_1 \ B_2B_1	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

A<B: $A'_2\, B_2 + A'_1\, B_2\, B_1 + A'_2A'_1B_1$

$= A'_2\, B_2 + A'_1B_1(B_2 + A'_2)$

Physical Implementation



- **Step 1: Truth table**
- **Step 2: K-map**
- **Step 3: Minimized sum-of-products**
- **Step 4: Physical implementation with gates**

Encoder, Decoder, Multiplexer, and Demultiplexer

Encoders

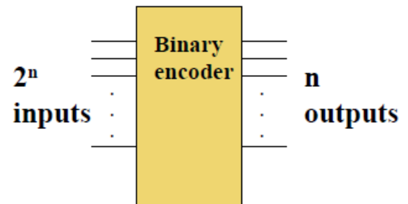
If the decoder's output code has fewer bits than the input code, the device is usually called an encoder.

e.g. 2^n-to-n

The simplest encoder is a 2^n-to-n binary encoder

•One of 2^n inputs = 1

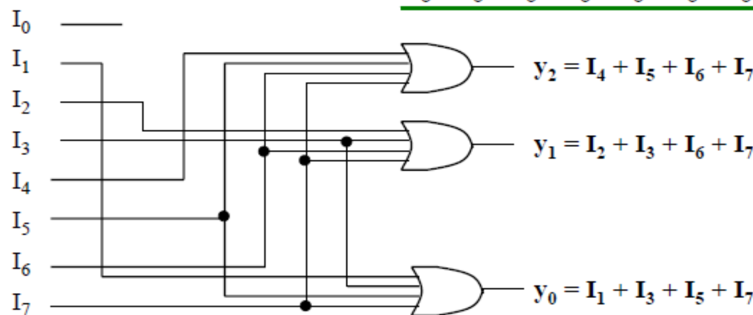
•Output is an n-bit binary number



8-to-3 Binary Encoder

At any one time, only one input line has a value of 1.

Inputs								Outputs		
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	y_2	y_1	y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1



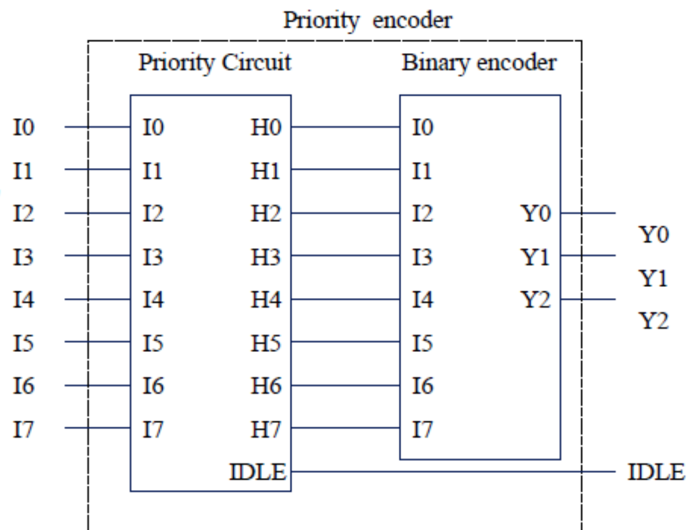
8-to-3 Priority Encoder

- **What if more than one input line has a value of 1?**
- **Ignore "lower priority" inputs.**
- **Idle indicates that no input is a 1.**
- **Note that polarity of Idle is opposite from Table in Mano**

Inputs								Outputs			
I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	y_2	y_1	y_0	Idle
0	0	0	0	0	0	0	0	x	x	x	1
1	0	0	0	0	0	0	0	0	0	0	0
X	1	0	0	0	0	0	0	0	0	1	0
X	X	1	0	0	0	0	0	0	1	0	0
X	X	X	1	0	0	0	0	0	1	1	0
X	X	X	X	1	0	0	0	1	0	0	0
X	X	X	X	X	1	0	0	1	0	1	0
X	X	X	X	X	X	1	0	1	1	0	0
X	X	X	X	X	X	X	1	1	1	1	0

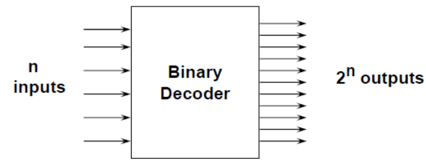
Priority Encoder (8 to 3 encoder)

- Assign priorities to the inputs
- When more than one input are asserted, the output generates the code of the input with the highest priority
- Priority Encoder :
  - H7=I7 (Highest Priority)
  - H6=I6.I7'
  - H5=I5.I6'.I7'
  - H4=I4.I5'.I6'.I7'
  - H3=I3.I4'.I5'.I6'.I7'
  - H2=I2.I3'.I4'.I5'.I6'.I7'
  - H1=I1. I2'.I3'.I4'.I5'.I6'.I7'
  - H0=I0.I1'. I2'.I3'.I4'.I5'.I6'.I7'
  - IDLE= I0'.I1'. I2'.I3'.I4'.I5'.I6'.I7'
- Encoder
  - Y0 = I1 + I3 + I5 + I7
  - Y1 = I2 + I3 + I6 + I7
  - Y2 = I4 + I5 + I6 + I7



Binary Decoder

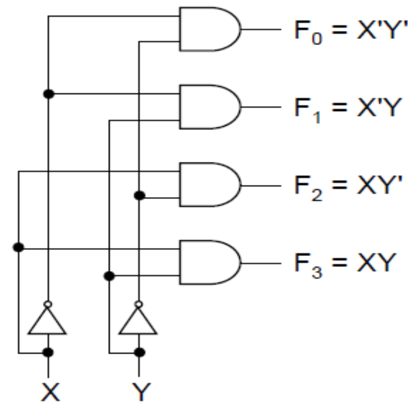
- Black box with n input lines and 2^n output lines
- Only one output is a 1 for any given input



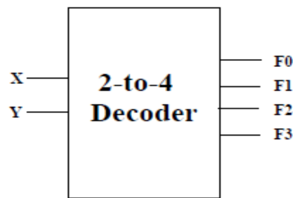
2-to-4 Binary Decoder

Truth Table:

X	Y	F_0	F_1	F_2	F_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



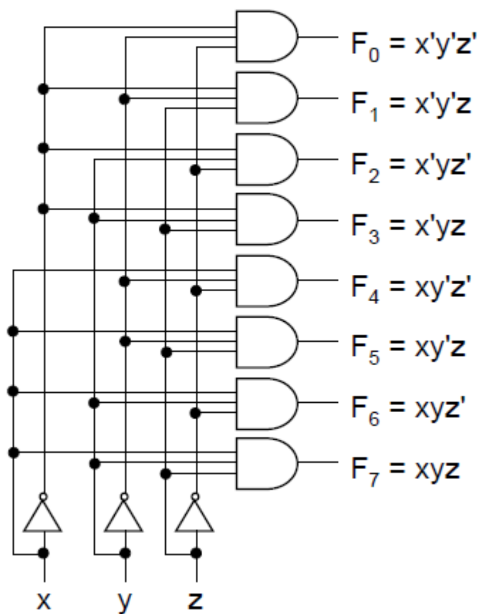
- **From truth table, circuit for 2x4 decoder is:**
- **Note: Each output is a 2-variable minterm (X'Y', X'Y, XY' or XY)**

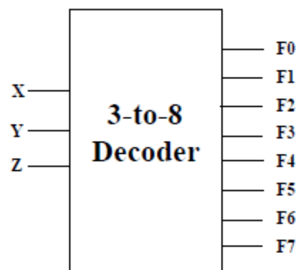


3-to-8 Binary Decoder

Truth Table:

x	y	z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1





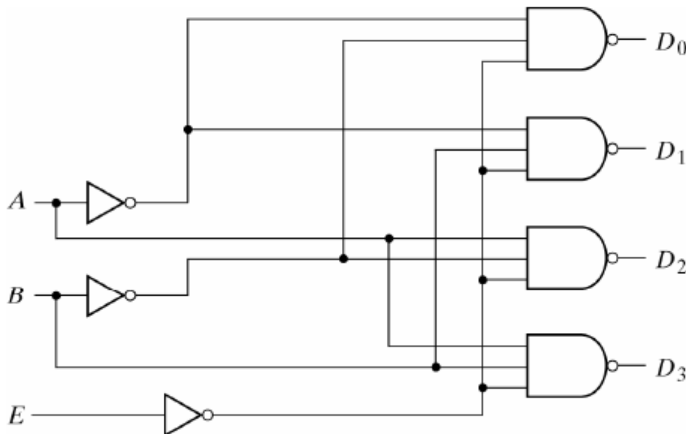
Building a Binary Decoder with NAND Gates

- Start with a 2-bit decoder
  - Add an enable signal (E)

Note: use of NANDs

only one 0 active!

if E = 0



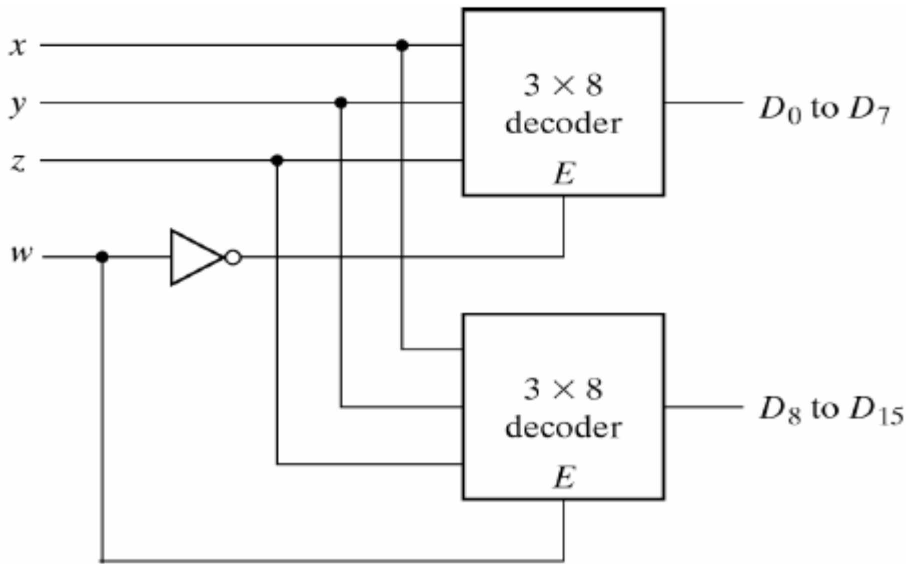
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(a) Logic diagram (b) Truth table

2-to-4-Line Decoder with Enable Input

Use two 3 to 8 decoders to make 4 to 16 decoder

- Enable can also be active high
- In this example, only one decoder can be active at a time.
- x, y, z effectively select output line for w

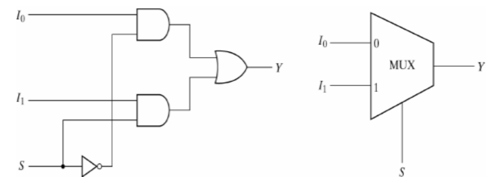


4 × 16 Decoder Constructed with Two 3 × 8 Decoders

Multiplexer

A multiplexer is a network that has many inputs and one output, and the value of the output will be the value of one of inputs which will be decided by some select lines. The simplest type of multiplexer is the two line to one line data multiplexer. Let A be one of the inputs and B is the other input and Y is the output, and S is the select line, then Y = A if Select = 0, Y = B if Select = 1.

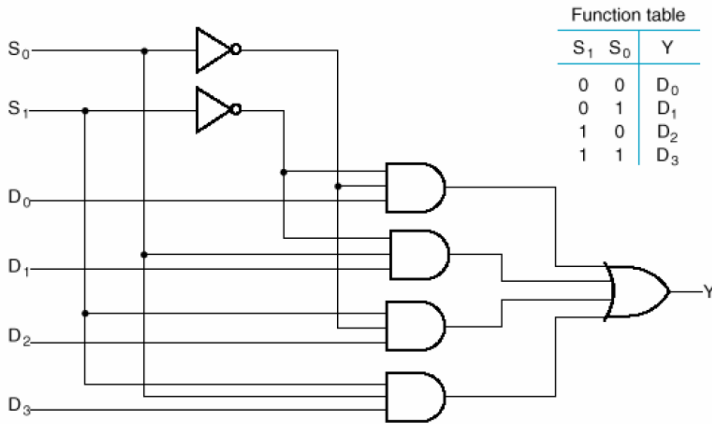
- Select an input value with one or more select bits
- Use for transmitting data
- Allows for conditional transfer of data
- Sometimes called a mux



(a) Logic diagram (b) Block diagram

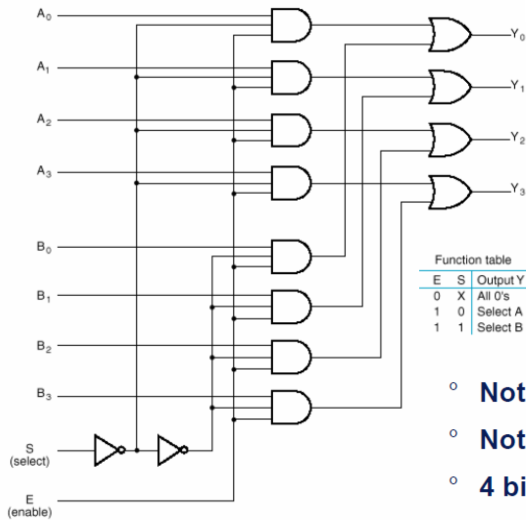
2-to-1-Line Multiplexer

4– to– 1- Line Multiplexer



S_1	S_0	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Quadruple 2–to–1-Line Multiplexer



E	S	Output Y
0	X	All 0's
1	0	Select A
1	1	Select B

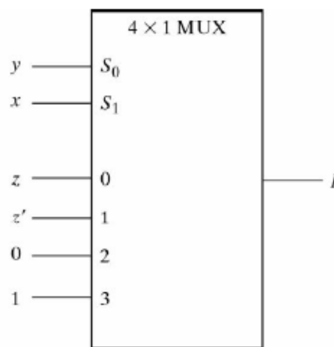
- Notice enable bit
- Notice select bit
- 4 bit inputs

Multiplexer as combinational modules

- **Connect input variables to select inputs of multiplexer *(n-1 for n variables)***
- **Set data inputs to multiplexer equal to values of function for corresponding assignment of select variables**
- **Using a variable at data inputs reduces size of the multiplexer**

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table

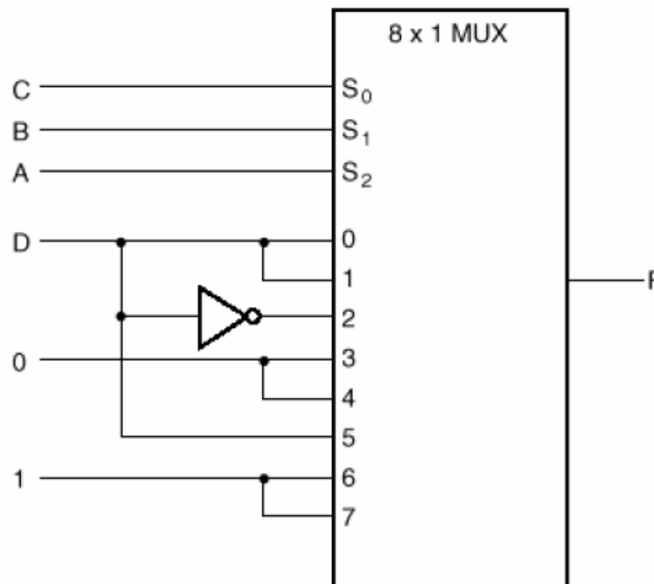


(b) Multiplexer implementation

Implementing a Boolean Function with a Multiplexer

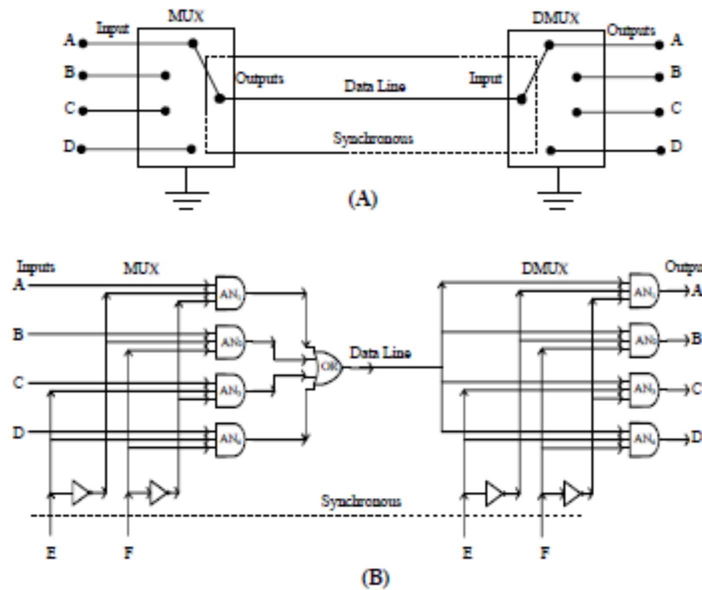
Implementing a Four- Input Function with a Multiplexer

A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	$F = \overline{D}$
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	



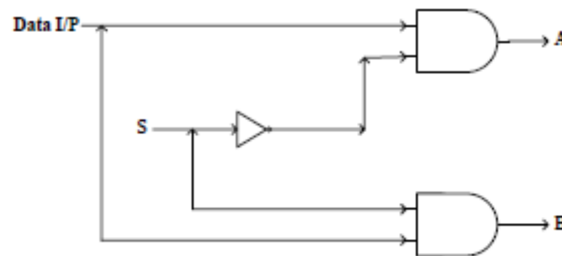
Demultiplexer

A demultiplexer basically reverses the multiplexing function. It is take data from one line and distribute them to given number of output lines. The following figure shows a one to four line demultiplexer circuit. The input data line goes to all of the AND gates. The two select lines enable only one gate at a time and the data appearing on the input line will pass through the selected gate to the associated output line.



MUX/DMUX System: (A). Switch Analog. (B). Logic Gate Circuit .

The simplest type of demultiplexer is the one to two lines DMUX.



One to Two Lines Demultiplexer

Q) Design majority voting using 4*1 multiplexer

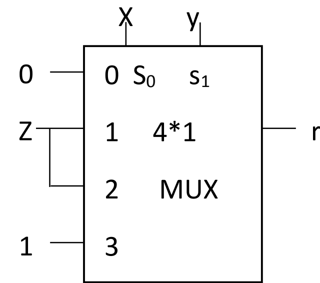
X	Y	Z	r
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

r=0

r=Z

r=Z

r=1



Q) Design 3-bit even Parity using 4*1 multiplexer

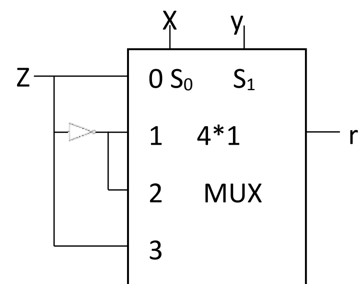
X	Y	Z	r
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

r=Z

r=Z’

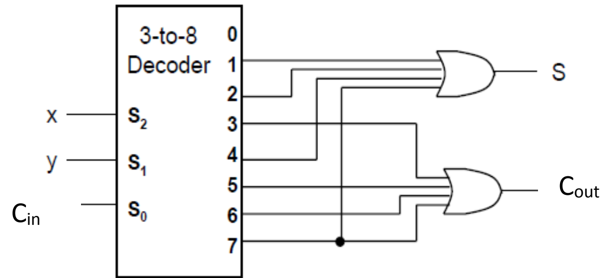
r=Z’

r=Z



Q) Design the full adder using 3*8 decoder

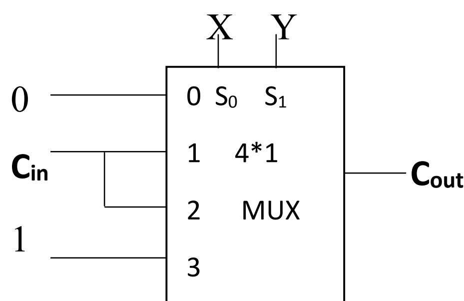
X	Y	C_{in}	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

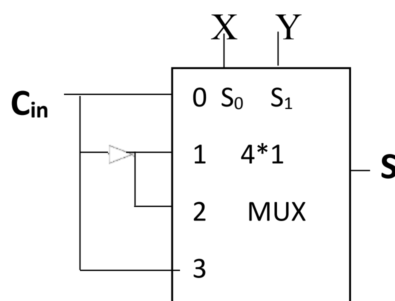


$S(x, y, C_{in}) = \Sigma(1,2,4,7)$
$C_{out}(x, y, C_{in}) = \Sigma(3,5,6,7)$

Q) Design the full adder using 4*1 multiplexer

X	Y	C_{in}	C_{out}		S	
0	0	0	0	$C_{out}=0$	0	$S= C_{in}$
0	0	1	0		1	
0	1	0	0	$C_{out}= C_{in}$	1	$S= C'_{in}$
0	1	1	1		0	
1	0	0	0	$C_{out}= C_{in}$	1	$S= C'_{in}$
1	0	1	1		0	
1	1	0	1	$C_{out}= 1$	0	$S= C_{in}$
1	1	1	1		1	





Q) Design the circuit using 4*1 multiplexer Z=f(A,B,C)=A'B'C'+A'B+ABC'+AC

1-1)

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Z=C'

Z=1

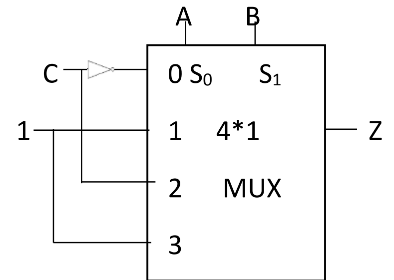
Z=C

Z=1

when the Select Lines: A, B

then the Function Table

S_0=A	S_1=B	Z
0	0	C'
0	1	1
1	0	C
1	1	1



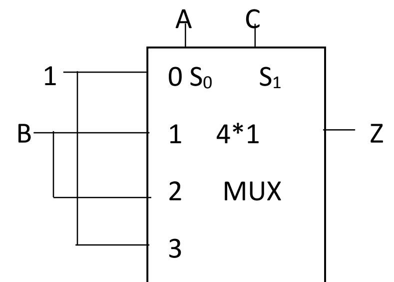
1-2)

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

when the Select Lines: A, C

then the Function Table

S_0=A	S_1=C	Z
0	0	1
0	1	B
1	0	B
1	1	1



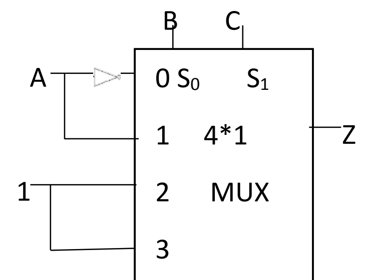
1-3)

A	B	C	Z
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

When the Select Lines: A, C

then the Function Table

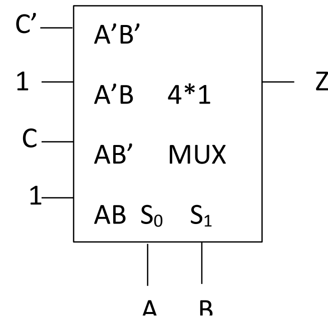
S_0=B	S_1=C	Z
0	0	A'
0	1	A
1	0	1
1	1	1



2)

C / AB	0	1	
00	1	0	Z=C' C=0
01	1	1	Z=1 both 1
11	1	1	Z=1 both 1
10	0	1	Z=C C=1

can be drawn like this



3)

Z=f(A,B,C)=A'B'C'+A'B+ABC'+AC

=A'B'C'+A'B(C+C')+ABC'+AC(B+B')

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

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

=A'B'(C')+A'B(1)+ AB'(C)+ AB(1)

4) Z=f(A,B,C)=A'B'C'+A'B+ABC'+AC= A'B'C'+A'BC+A'BC'+ABC'+ABC+AB'C

Index(No. of 1)	0	2	1	2	3	2
Value	0	3	2	6	7	5

Value according to index	A	B	C
0	0	0	0
2	0	1	0
3	0	1	1
5	1	0	1
6	1	1	0
7	1	1	1

1	2	4	weight
C	B	A	
2,3	0,2	2,6	(the difference between numbers
6,7	5,7	3,7	have different index=weight)

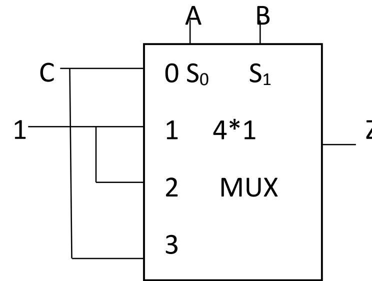
All variable have the same no. of pairs then take any one like C

4	2	0	1	data value
A	B	C'	C	
0	0	(0)	1	C'
0	1	(2)	(3)	1
1	0	4	(5)	C
1	1	(6)	(7)	1

Q) Design the circuit using 4*1 multiplexer Z=f(A,B,C)=A'B+B'C+BC+AB'C'

1)

A	B	C	Z	
0	0	0	0	Z=C
0	0	1	1	
0	1	0	1	Z=1
0	1	1	1	
1	0	0	1	Z=1
1	0	1	1	
1	1	0	0	Z=C
1	1	1	1	



2)

AB \ C	0	1
00	0	1
10	1	1
11	0	1
10	1	1

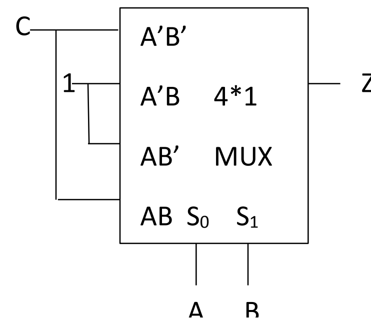
Z=C C=1

Z=1 both 1

Z=C C=1

Z=1 both 1

Or can be drawn like this



3) Z=f(A,B,C)= A'B+B'C+BC+AB'C'

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

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

=A'B'C+A'BC+A'BC'+AB'C+AB'C'+ABC=A'B'C+A'B(C+C')+AB'(C+C')+ABC

=A'B'(C)+A'B(1)+ AB'(1)+ AB(C)

4) Z=f(A,B,C)= A'B+B'C+BC+AB'C'= A'BC+A'BC'+AB'C+A'B'C+ABC+AB'C'

Index(No. of 1)	2	1	2	1	3	1
Value	3	2	5	1	7	4

Value according to index	A	B	C
1	0	0	1
2	0	1	0
4	1	0	0
3	0	1	1
5	1	0	1
7	1	1	1

1	2	4	
C	B	A	
2,3	1,3	1,5	(the difference between numbers
4,5		3,7	have different index=weight)

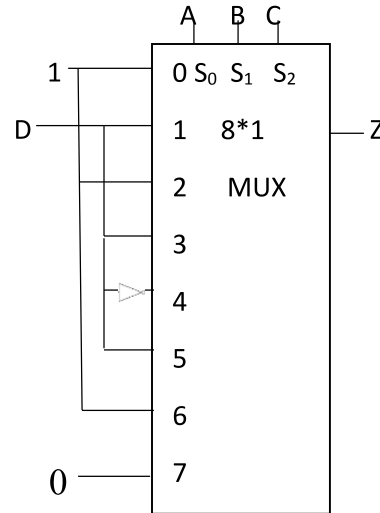
A,C have the same no. of pairs then take any one like C

4 A	2 B	0 C'	1 C	data value
0	0	0	(1)	C
0	1	(2)	(3)	1
1	0	(4)	(5)	1
1	1	6	(7)	C

Q) Design the circuit using 8*1 multiplexer Z=f(A,B,C,D)=∑(1,3,4,7,12,13) with don't care (0,5,8,11)

1)

A	B	C	D	Z	
0	0	0	0	X	Z=1;D
0	0	0	1	1	
0	0	1	0	0	Z=D
0	0	1	1	1	
0	1	0	0	1	Z=1;D'
0	1	0	1	X	
0	1	1	0	0	Z=D
0	1	1	1	1	
1	0	0	0	X	Z=0;D'
1	0	0	1	0	
1	0	1	0	0	Z=0;D
1	0	1	1	X	
1	1	0	0	1	Z=1
1	1	0	1	1	
1	1	1	0	0	Z=0
1	1	1	1	0	



2) Z=f(A,B,C,D)=∑(1,3,4,7,12,13) with don't care (0,5,8,11)

Index(No. of 1) 1 2 1 3 2 3 0 2 1 3

Value according to index	A	B	C	D
0	0	0	0	0
1	0	0	0	1
4	0	1	0	0
8	1	0	0	0
3	0	0	1	1
5	0	1	0	1
12	1	1	0	0
7	0	1	1	1
11	1	0	1	1
13	1	1	0	1

1	2	4	8
D	C	B	A
0,1	1,3	0,4	0,8
4,5	5,7	1,5	4,12
12,13		8,12	3,11
		3,7	5,13

(the difference between numbers have different index=weight)

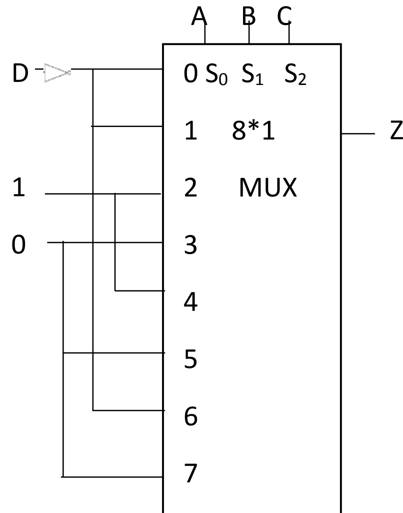
A,B have the same no. of pairs then take any one like A

4	2	1	0	8	data value
B	C	D	A'	A	
0	0	0	0	8	1
0	0	1	1	9	A'
0	1	0	2	10	0
0	1	1	3	11	1
1	0	0	4	12	1
1	0	1	5	13	1
1	1	0	6	14	0
1	1	1	7	15	A'

Q) Design the circuit using 8*1 multiplexer

Z=f(A,B,C,D)=A'C'D'+BC'D'+AB'C'+A'BC'D+A'B'CD'

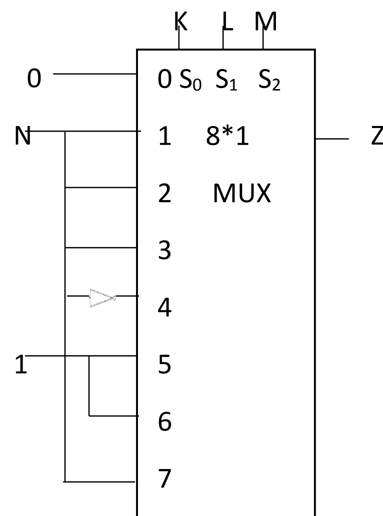
A	B	C	D	Z	
0	0	0	0	1	Z=D'
0	0	0	1	0	
0	0	1	0	1	Z=D'
0	0	1	1	0	
0	1	0	0	1	Z=1
0	1	0	1	1	
0	1	1	0	0	Z=0
0	1	1	1	0	
1	0	0	0	1	Z=1
1	0	0	1	1	
1	0	1	0	0	Z=0
1	0	1	1	0	
1	1	0	0	1	Z=D'
1	1	0	1	0	
1	1	1	0	0	Z=0
1	1	1	1	0	



Q) Design the circuit using 8*1 multiplexer

Z=f(K,L,M,N)=KL'N'+KLM'+LMN+K'L'MN with don't care (K'LM'N,KL'MN)

K	L	M	N	Z	
0	0	0	0	0	Z=0
0	0	0	1	0	
0	0	1	0	0	Z=N
0	0	1	1	1	
0	1	0	0	0	Z=0;N
0	1	0	1	X	
0	1	1	0	0	Z=N
0	1	1	1	1	
1	0	0	0	1	Z=N'
1	0	0	1	0	
1	0	1	0	1	Z=1;N'
1	0	1	1	X	
1	1	0	0	1	Z=1
1	1	0	1	1	
1	1	1	0	0	Z=N
1	1	1	1	1	



Homework

Q1) Design the circuit using 8*1 multiplexer

$Z=f(A,B,C,D)=\sum(0,1,2,3,5,7,8,10,12,13,15)$

Q2) Design the circuit using 8*1 multiplexer $Z=f(A,B,C)=\sum(2,3,5,6,7)$

Q3) Design the circuit using 4*1 multiplexer $Z=f(A,B,C)=\sum(2,3,5,6,7)$

Q4) Design the comparator to compare 2 numbers each has 2 bits using 4*16 decoder

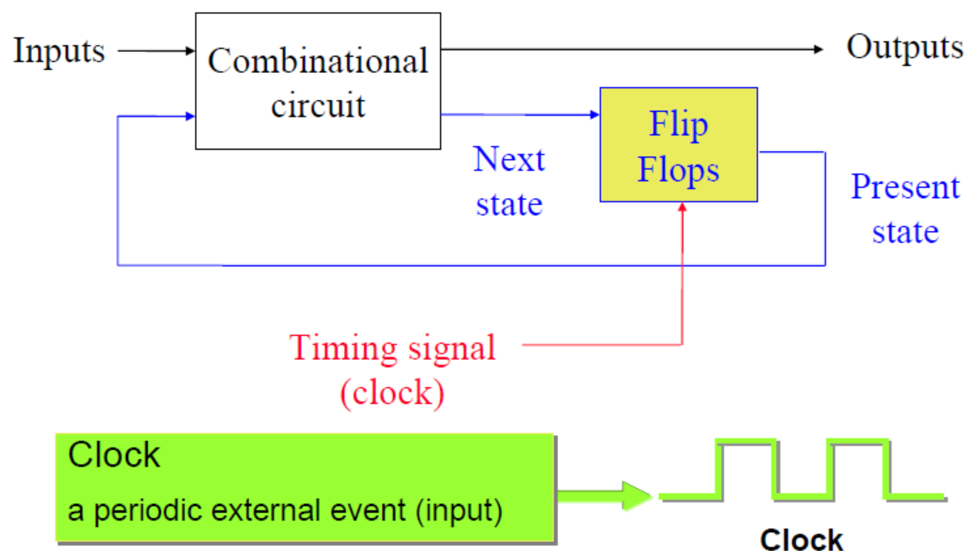
Q5) Design the comparator to compare 2 numbers each has 2 bits using 8*1 multiplexer

Q6) Design the circuit using 4*1 multiplexer $Z=f(A,B,C)=A'B+BC+A'C$

Sequential Circuits

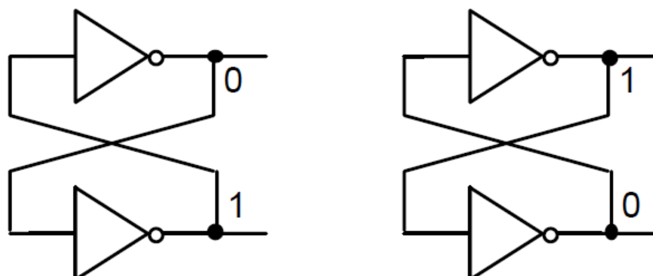
Sequential Circuits: Consist of a combinational circuit to which memory elements are connected to form a feedback path. The memory elements (Flip-Flops) are devices capable of storing binary information within them. This binary information at any given time defines the state of the sequential circuit.

- Outputs depend on inputs and previous values of outputs
- Outputs depend on previous state of the circuit
- State is stored in memory elements (registers, latches, flip flops)



Cross-Coupled Invertor

A stable value can be stored at inverter outputs

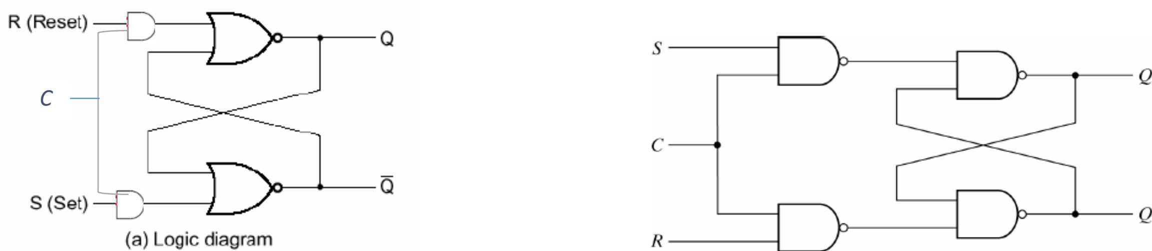


Flip-Flop

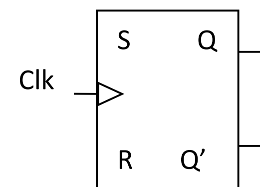
1-S-R Flip-Flop

S-R latch made from cross-coupled NORs

Logic circuit diagram of Simple S-R

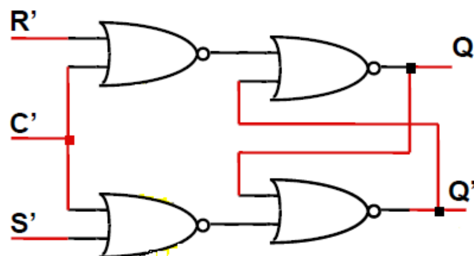


Logic circuit diagram of Clock S-R



Graphic Symbol

- Occasionally, desirable to avoid latch changes
- C = 0 disables all latch state changes
- Control signal enables data change when C = 1



NOR S-R Latch with Control Input
Latch is level-sensitive, in regards to C
Only stores data if C'= 0

Inputs			Output
S	R	Q_t	$Q_{(t+1)}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	X
1	1	1	X

Truth table

S	R	$Q_{(t+1)}$
0	0	Q_t No change
0	1	0 Reset
1	0	1 Set
1	1	Unpredictable

Characteristic table

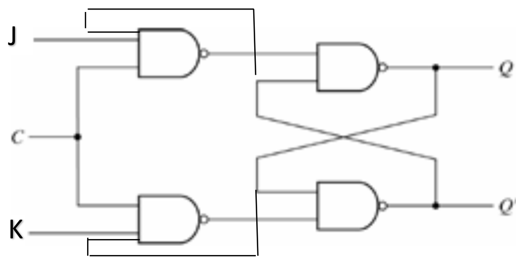
Q_t	$Q_{(t+1)}$	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Excitation table

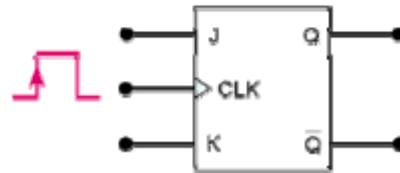
$$Q_{(t+1)} = S + R'Q_t$$

$$S.R = 0$$

2-J-K Flip-Flop



Logic circuit diagram of Clock J-K



Graphic Symbol

Inputs			Output
J	K	Q_t	$Q_{(t+1)}$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Truth table

J	K	$Q_{(t+1)}$
0	0	Q_t No change
0	1	0 Reset
1	0	1 Set
1	1	Q'_t Toggle

Characteristic table

Q_t	$Q_{(t+1)}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	x	0

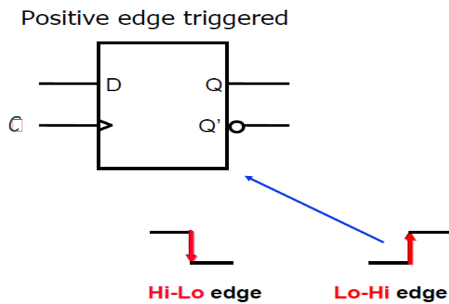
Excitation table

$$Q_{t+1} = J\overline{Q_t} + \overline{K}Q_t$$

- Two data inputs, J and K
- J -> set, K -> reset, if J=K=1 then toggle output

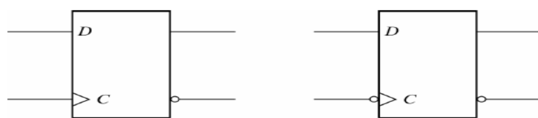
3-D Flip-Flop

- Q0 indicates the previous state (the previously stored value)
- Stores a value on the positive edge of C (D gets latched to Q on the rising edge of the clock, Input changes at other times have no effect on output)



Positive and Negative Edge D Flip-Flop

- D flops can be triggered on positive or negative edge
- Bubble before *Clock (C)* input indicates negative edge trigger



(a) Positive-edge (a) Negative-edge

Graphic Symbol for Edge-Triggered *D* Flip-Flop

Inputs		Output
D	Q_t	$Q_{(t+1)}$
0	0	0
0	1	0
1	0	1
1	1	1

Truth table

D	Q_t
0	0 Reset
1	1 Set

Characteristic table

Q_t	$Q_{(t+1)}$	D
0	0	0
0	1	1
1	0	0
1	1	1

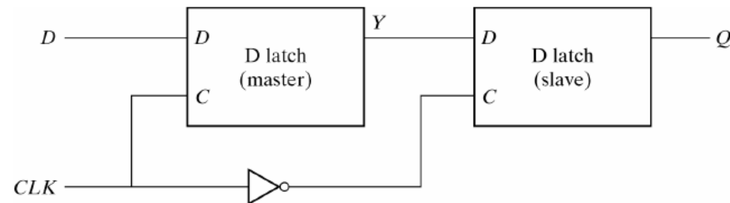
Excitation table

$$Q_{t+1} = D$$

- Input value D is passed to output Q when C is high
- Input value D is ignored when C is low

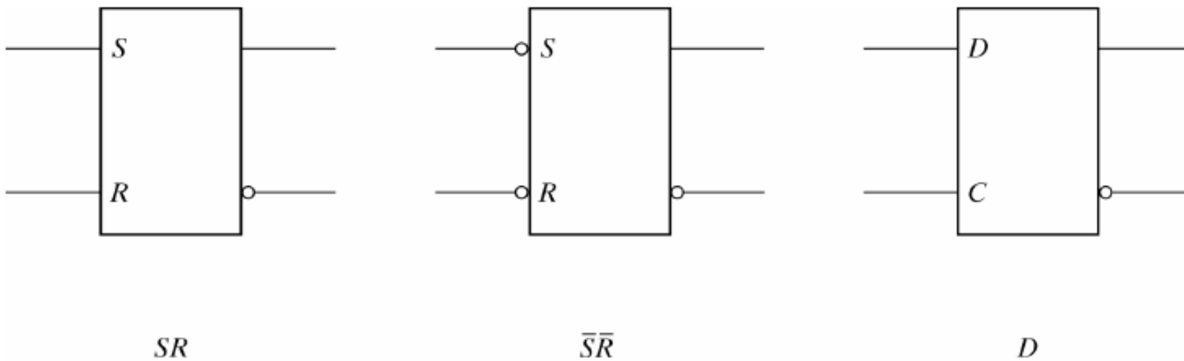
Master-Slave D Flip Flop

- **Consider two latches combined together**
- **Only one *C* value active at a time**
- **Output changes on falling edge of the clock**



Master-Slave *D* Flip-Flop

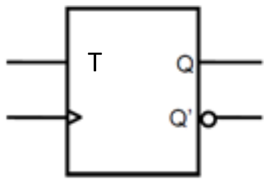
Symbols for Latches



Graphic Symbols for Latches

- SR latch is based on NOR gates
- S'R' latch based on NAND gates
- D latch can be based on either.
- D latch sometimes called transparent latch

4-T Flip-Flop



Graphic Symbol

Inputs		Output
T	Q_t	$Q_{(t+1)}$
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

T	$Q_{(t+1)}$
0	Q_t No change
1	Q'_t Toggle

Characteristic table

Q_t	$Q_{(t+1)}$	T
0	0	0
0	1	1
1	0	1
1	1	0

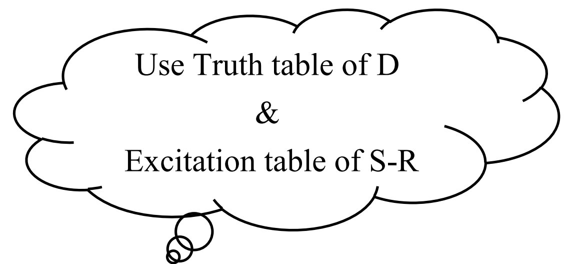
Excitation table

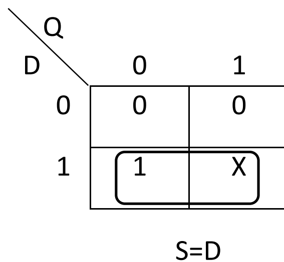
$$Q_{t+1} = T\overline{Q_t} + \overline{T}Q_t = T \oplus Q_t$$

Convert from one Flip-Flop (F.F) to another

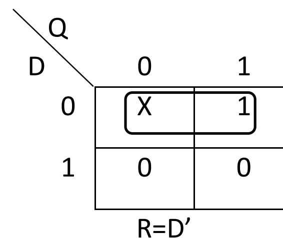
1- S-R F.F → D F.F

D	Q_t	$Q_{(t+1)}$	S	R
0	0	0	0	x
0	1	0	0	1
1	0	1	1	0
1	1	1	x	0

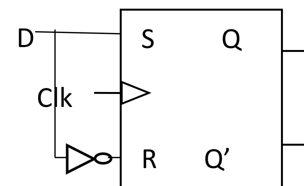


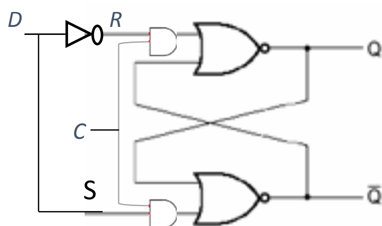


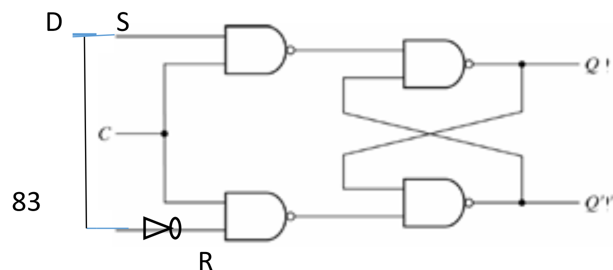
S=D



R=D'



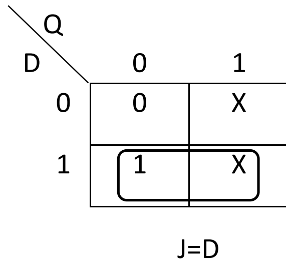




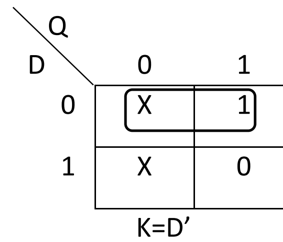
2- J-K F.F → D F.F

D	Q_t	$Q_{(t+1)}$	J	K
0	0	0	0	x
0	1	0	X	1
1	0	1	1	X
1	1	1	x	0

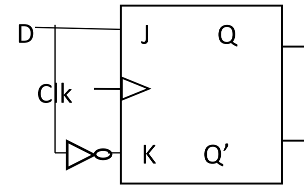
Use Truth table of D
&
Excitation table of J-K

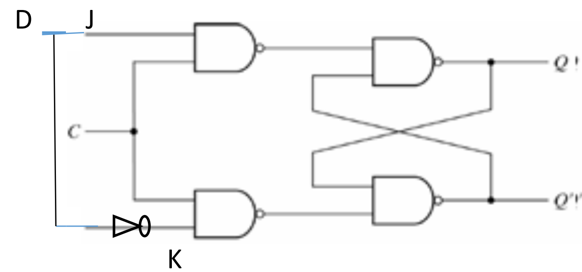


J=D



K=D'

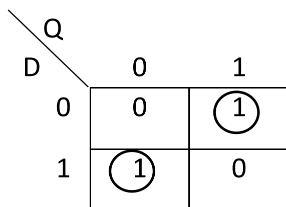




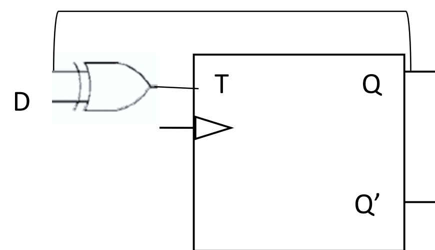
3- T F.F → D F.F

D	Q_t	$Q_{(t+1)}$	T
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Use Truth table of D
&
Excitation table of T



$T = DQ' + D'Q = D \oplus Q$



4) S-R F.F → T F.F

T	Q_t	$Q_{(t+1)}$	S	R
0	0	0	0	x
0	1	1	X	0
1	0	1	1	0
1	1	0	0	1

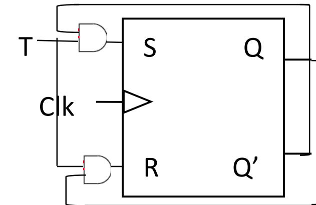
Use Truth table of T
&
Excitation table of S-R

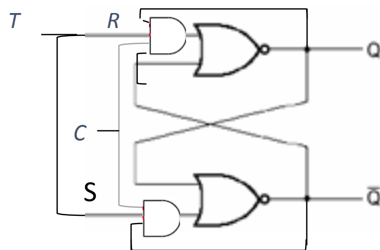
T \ Q	0	1
0	0	X
1	(1)	0

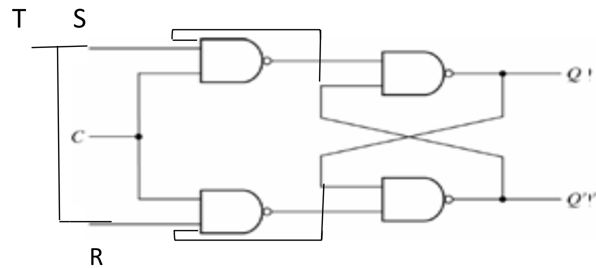
$S = T.\overline{Q}$

T \ Q	0	1
0	X	0
1	0	(1)

R=T.Q







5) J-K F.F→ T F.F

T	Q_t	$Q_{(t+1)}$	J	K
0	0	0	0	x
0	1	1	X	0
1	0	1	1	X
1	1	0	x	1

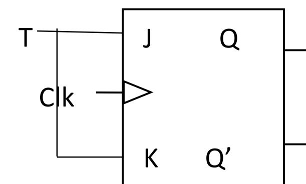
Use Truth table of T
&
Excitation table of J-K

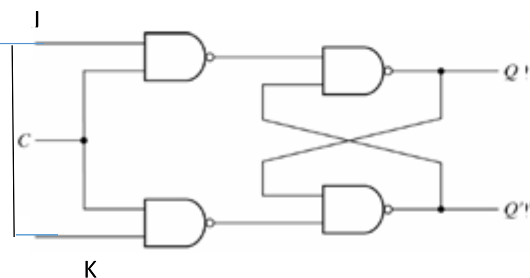
T \ Q	0	1
0	0	X
1	(1	X)

J=T

T \ Q	0	1
0	X	0
1	(X	1)

K=T





6- J-K F.F → S-R F.F

S	R	Q_t	$Q_{(t+1)}$	J	K
0	0	0	0	0	x
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	x	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	0	X	X	X
1	1	1	X	X	X

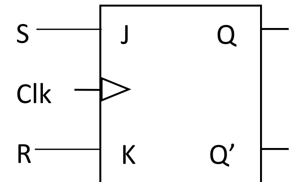
Use Truth table of S-R
&
Excitation table of J-K

S \ RQ	00	01	11	10
0	0	X	X	0
1	1	X	X	X

J=S

S \ RQ	00	01	11	10
0	X	0	1	X
1	X	0	X	X

K=R



Homework

1) Convert D F.F → T F.F
2) Convert D F.F → S-R F.F
3) Convert T F.F → J-K F.F
4) Convert D F.F → J-K F.F
5) Convert S-R F.F → J-K F.F
6) Convert T F.F → S-R F.F

Counter

°Counters are important components in computers

- The increment or decrement by one in response to input

°Two main types of counters

- Ripple (asynchronous) counters
- Synchronous counters

°Ripple counters

- Flip flop output serves as a source for triggering other flip flops

°Synchronous counters

- All flip flops triggered by a clock signal

°Synchronous counters are more widely used in industry.

°Counter: A register that goes through a prescribed series of states

°Binary counter

- Counter that follows a binary sequence
- N bit binary counter counts in binary from 0 to 2^n-1

°Ripple counters triggered by initial Count Signal

°Applications:

- Watches
- Clocks
- Alarms
- Web browser refresh

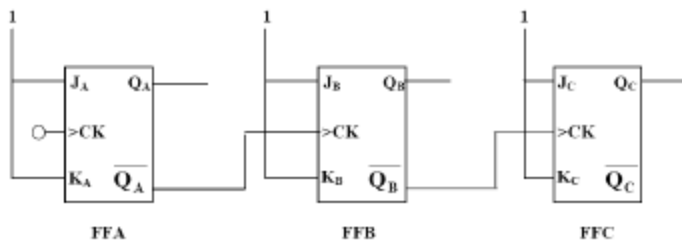
Counter: A sequential circuit that goes through a prescribed sequence of states upon the application of input pulses. Which used for counting the number of occurrences of an event and are useful for generating timing sequences to control operations in a digital system.

Asynchronous binary counter

A three-stage asynchronous binary counter is shown in the following figure. It has eight states due to its three states.

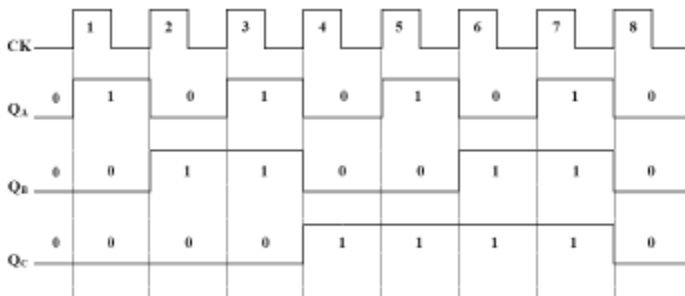
Clock Pulse	QC	QB	QA
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

State Table for a Three-Stage Binary Counter.



Three-Stages Asynchronous Counter.

A timing diagram appears in the following figure for eight clock pulse.



Timing Diagram of Three-Stage Asynchronous Counter.

Notes:-

- Reset signal sets all outputs to 0
- Count signal toggles output of low-order flip flop
- Low-order flip flop provides trigger for adjacent flip flop
- Not all flops change value simultaneously
  •Lower-order flops change first
- Each FF output drives the CLK input of the next FF.
- FFs do not change states in exact synchronism with the applied clock pulses.
- There is delay between the responses of successive FFs.
- Ripple counter due to the way the FFs respond one after another in a kind of rippling effect.

Synchronous binary counter

The synchronous counter is also called a parallel counter because the clock line is connected in parallel to each Flip-Flop. Notice that an arrangement different from that for the asynchronous counter. The following figure shows a four-stage binary counter and its equivalent logic symbol.

Q) Design 4 bit counter using J-K F.F

Present State Q_t				Next State Q_{t+1}				J_D	K_D	J_C	K_C	J_B	K_B	J_A	K_A
D	C	B	A	D	C	B	A								
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	1	0	1	0	X	0	0	X	1	X	X	1
1	0	1	0	1	0	1	1	X	0	0	X	X	0	1	X
1	0	1	1	1	1	0	0	X	0	1	X	X	1	X	1
1	1	0	0	1	1	0	1	X	0	X	0	0	X	1	X
1	1	0	1	1	1	1	0	X	0	X	0	1	X	X	1
1	1	1	0	1	1	1	1	X	0	X	0	X	0	1	X
1	1	1	1	0	0	0	0	X	1	X	1	X	1	X	1

Excitation table of J-K F.F

Q_t	$Q_{(t+1)}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	x	0

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	X	X	X	X

$J_D = Q_C . Q_B . Q_A$

DC \ BA	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	0	0	1	0
10	0	0	0	0

$k_D = Q_C . Q_B . Q_A$

DC \ BA	00	01	11	10
00	0	0	1	0
01	X	X	X	X
11	X	X	X	X
10	0	0	1	0

$J_C = Q_B . Q_A$

DC \ BA	00	01	11	10
00	X	X	X	X
01	0	0	1	0
11	0	0	1	0
10	X	X	X	X

$k_C = Q_B . Q_A$

DC \ BA	00	01	11	10
00	0	X	1	X
01	0	1	X	X
11	0	1	X	X
10	0	1	X	X

$J_B = Q_A$

DC \ BA	00	01	11	10
00	X	1	X	0
01	X	X	1	0
11	X	X	1	0
10	X	X	1	0

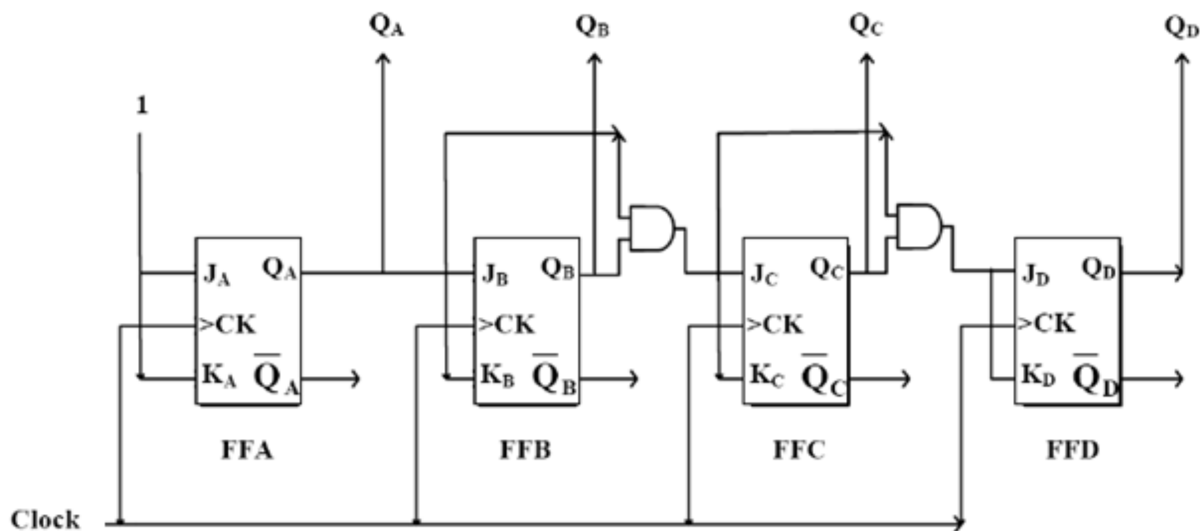
$K_B = Q_A$

DC \ BA	00	01	11	10
00	1	X	X	1
01	1	X	X	1
11	1	X	X	1
10	1	X	X	1

$J_A = 1$

DC \ BA	00	01	11	10
00	X	1	1	X
01	X	1	1	X
11	X	1	1	X
10	X	1	1	X

$K_A = 1$



Four-stage synchronous binary counter

Notes:-

- Synchronous (parallel) counters
  - •All of the FFs are triggered simultaneously by the clock input pulses.
  - •All FFs change at same time
- Remember
  - •If J=K=0, flop maintains value
  - •If J=K=1, flop toggles
- Most counters are synchronous in computer systems.

Decade Counter

Decade counters are very important category of digital counter because of their wide application, a decade counter has ten states in its sequence that is, it has modulus of ten. It consist of four stages and can have any given sequence of states as long as there are ten. A very common type of decade counter is the BCD (8421) counter, which exhibits a binary-coded-decimal sequence as shown in Table (2).

As you can see, the BCD decade counter goes through a straight binary sequence through the binary 9 state, rather than going to the binary 10 state, it recycles to the 0 state. A synchronous BCD decade counter is shown in Fig. (6).

Q) Design BCD decade counter using J-K F.F

Present State Q_t				Next State Q_{t+1}				J_D	K_D	J_C	K_C	J_B	K_B	J_A	K_A
D	C	B	A	D	C	B	A								
0	0	0	0	0	0	0	1	0	X	0	X	0	X	1	X
0	0	0	1	0	0	1	0	0	X	0	X	1	X	X	1
0	0	1	0	0	0	1	1	0	X	0	X	X	0	1	X
0	0	1	1	0	1	0	0	0	X	1	X	X	1	X	1
0	1	0	0	0	1	0	1	0	X	X	0	0	X	1	X
0	1	0	1	0	1	1	0	0	X	X	0	1	X	X	1
0	1	1	0	0	1	1	1	0	X	X	0	X	0	1	X
0	1	1	1	1	0	0	0	1	X	X	1	X	1	X	1
1	0	0	0	1	0	0	1	X	0	0	X	0	X	1	X
1	0	0	1	0	0	0	0	X	1	0	X	0	X	X	1

Excitation table of J-K F.F

Q_t	$Q_{(t+1)}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	X	X	X	X
10	X	X	X	X

$J_D = Q_C.Q_B.Q_A$

DC \ BA	00	01	11	10
00	X	X	X	X
01	X	X	X	X
11	X	X	X	X
10	0	1	X	X

$k_D = Q_D.Q_A$

DC \ BA	00	01	11	10
00	0	0	1	0
01	X	X	X	X
11	X	X	X	X
10	0	0	X	X

$J_C = Q_B.Q_A$

DC \ BA	00	01	11	10
00	X	X	X	X
01	0	0	1	0
11	X	X	X	X
10	X	X	X	X

$k_C = Q_B.Q_A$

DC \ BA	00	01	11	10
00	0	1	X	X
01	0	1	X	X
11	X	X	X	X
10	0	0	X	X

$J_B = Q'_D .Q_A$

DC \ BA	00	01	11	10
00	X	X	1	0
01	X	X	1	0
11	X	X	X	X
10	X	X	X	X

$K_B = Q'_D .Q_A$

DC \ BA	00	01	11	10
00	1	X	X	1
01	1	X	X	1
11	X	X	X	X
10	1	X	X	X

$J_A = 1$

DC \ BA	00	01	11	10
00	X	1	1	X
01	X	1	1	X
11	X	X	X	X
10	X	1	X	X

$K_A = 1$

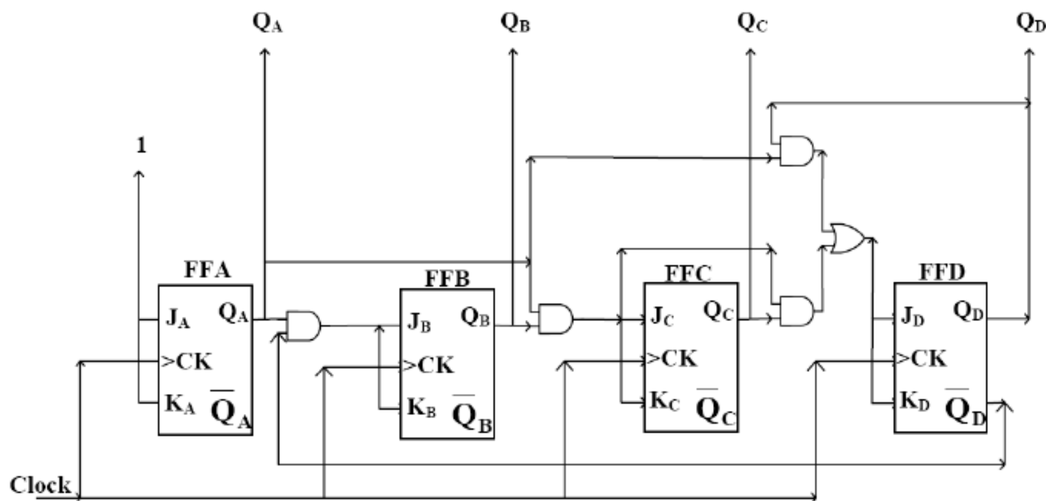


Fig. (6)
BCD Synchronous Decade Counter

Q) Design 3 bit counter using J-K F.F

Present State Q_t			Next State Q_{t+1}			J_C	K_C	J_B	K_B	J_A	K_A
C	B	A	C	B	A						
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	1	0	1	X	0	0	X	1	X
1	0	1	1	1	0	X	0	1	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	0	0	0	X	1	X	1	X	1

Excitation table of J-K F.F

Q_t	$Q_{(t+1)}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	x	0

C \ BA	00	01	11	10
0	0	0	1	0
1	x	x	x	x

$J_C = Q_B\ Q_A$

C \ BA	00	01	11	10
0	x	x	x	x
1	0	0	1	0

$K_C = Q_B\ Q_A$

C \ BA	00	01	11	10
0	0	1	x	x
1	0	1	x	x

$J_B = Q_A$

C \ BA	00	01	11	10
0	x	x	1	0
1	x	x	1	0

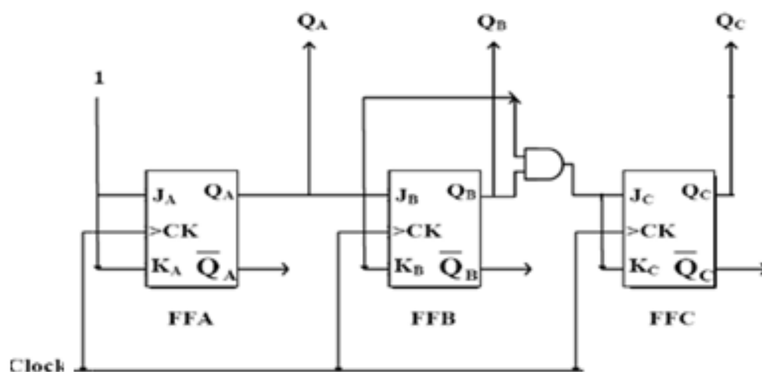
$K_B = Q_A$

C \ BA	00	01	11	10
0	1	x	x	1
1	1	x	x	1

$J_A = 1$

C \ BA	00	01	11	10
0	X	1	1	x
1	x	1	1	x

$K_A = 1$



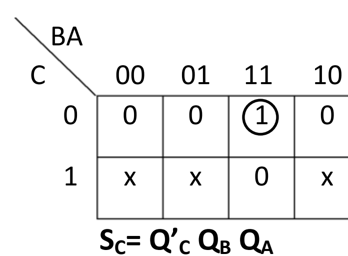
3 bit synchronous binary counter

Q) Design 3 bit counter using S-R F.F

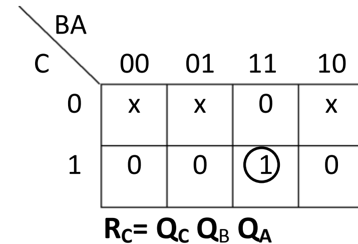
Present State Q_t			Next State Q_{t+1}			S_C	R_C	S_B	R_B	S_A	R_A
C	B	A	C	B	A						
0	0	0	0	0	1	0	X	0	X	1	0
0	0	1	0	1	0	0	X	1	0	0	1
0	1	0	0	1	1	0	X	X	0	1	0
0	1	1	1	0	0	1	0	0	1	0	1
1	0	0	1	0	1	X	0	0	X	1	0
1	0	1	1	1	0	X	0	1	0	0	1
1	1	0	1	1	1	X	0	X	0	1	0
1	1	1	0	0	0	0	1	0	1	0	1

Excitation table of S-R F.F

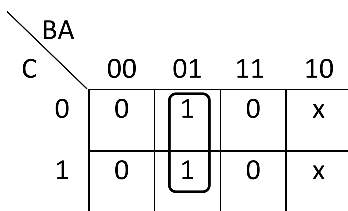
Q_t	$Q_{(t+1)}$	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	x	0

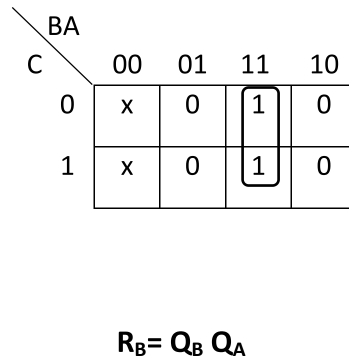


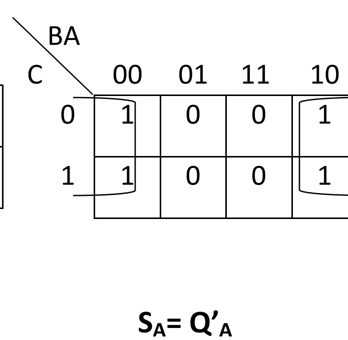
$S_C = Q'_C Q_B Q_A$

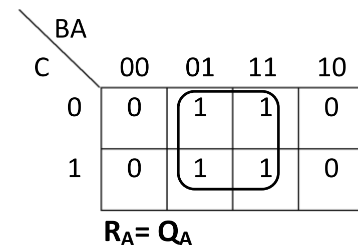


$R_C = Q_C Q_B Q_A$









$R_A = Q_A$

$S_A = Q'_A$

$S_B = Q'_B Q_A$

$R_B = Q_B Q_A$

Q) Design 3 bit counter using T F.F

Present State Q_t			Next State Q_{t+1}			T_C	T_B	T_A
C	B	A	C	B	A			
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	1
0	1	0	0	1	1	0	0	1
0	1	1	1	0	0	1	1	1
1	0	0	1	0	1	0	0	1
1	0	1	1	1	0	0	1	1
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Excitation table of T F.F

Q_t	$Q_{(t+1)}$	T
0	0	0
0	1	1
1	0	1
1	1	0

C \ BA	00	01	11	10
0	0	0	1	0
1	0	0	1	0

$T_C = Q_B Q_A$

C \ BA	00	01	11	10
0	0	1	1	0
1	0	1	1	0

$T_B = Q_A$

C \ BA	00	01	11	10
0	1	1	1	1
1	1	1	1	1

$T_A = 1$

Q) Design 3 bit counter using D F.F

Present State Q_t			Next State Q_{t+1}			D_C	D_B	D_A
C	B	A	C	B	A			
0	0	0	0	0	1	0	0	1
0	0	1	0	1	0	0	1	0
0	1	0	0	1	1	0	1	1
0	1	1	1	0	0	1	0	0
1	0	0	1	0	1	1	0	1
1	0	1	1	1	0	1	1	0
1	1	0	1	1	1	1	1	1
1	1	1	0	0	0	0	0	0

Excitation table of DF.F

Q_t	$Q_{(t+1)}$	D
0	0	0
0	1	1
1	0	0
1	1	1

C \ BA	00	01	11	10
0	0	0	1	0
1	1	1	0	1

$D_C = Q'_C Q_B Q_A + Q_C Q'_B + Q_C Q'_A$

C \ BA	00	01	11	10
0	0	1	0	1
1	0	1	0	1

$D_B = Q'_B Q_A + Q_B Q'_A = Q_B \oplus Q_A$

C \ BA	00	01	11	10
0	1	0	0	1
1	1	0	0	1

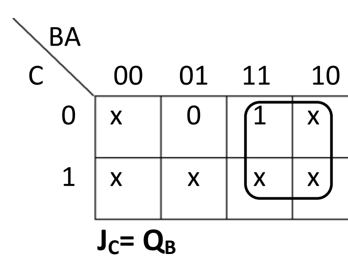
$D_A = Q'_A$

Q) Design 3 bit odd counter (1→3→5→7→1) using J-K F.F

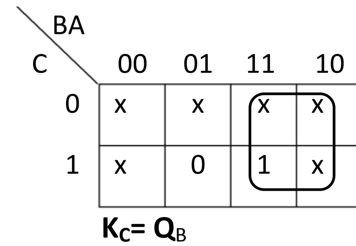
Present State Q_t			Next State Q_{t+1}			J_C	K_C	J_B	K_B	J_A	K_A
C	B	A	C	B	A						
0	0	1	0	1	1	0	X	1	X	x	0
0	1	1	1	0	1	1	X	x	1	x	0
1	0	1	1	1	1	X	0	1	X	x	0
1	1	1	0	0	1	X	1	x	1	x	0

Excitation table of J-K F.F

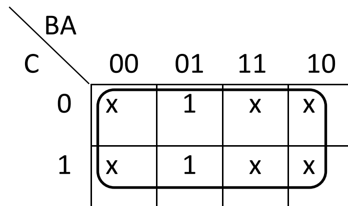
Q_t	$Q_{(t+1)}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	x	0



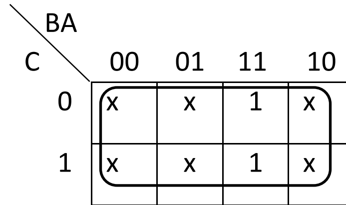
$J_C = Q_B$



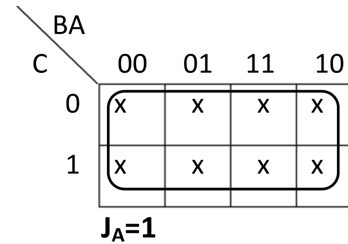
$K_C = Q_B$



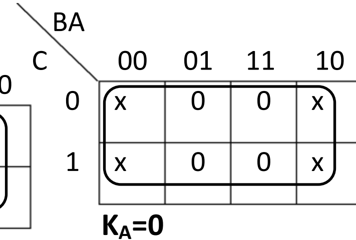
$J_B = 1$



$K_B = 1$



$J_A = 1$



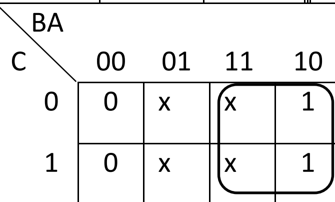
$K_A = 0$

Q) Design 3 bit even counter (0→2→4→6→0) using T F.F

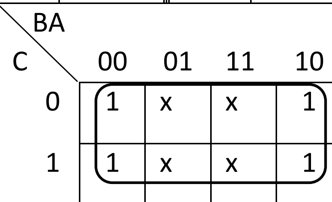
Present State Q_t			Next State Q_{t+1}			T_C	T_B	T_A
C	B	A	C	B	A			
0	0	0	0	1	0	0	1	0
0	1	0	1	0	0	1	1	0
1	0	0	1	1	0	0	1	0
1	1	0	0	0	0	1	1	0

Excitation table of T F.F

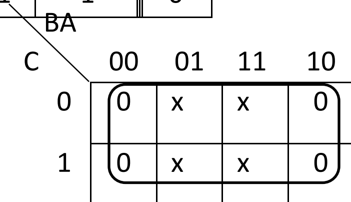
Q_t	$Q_{(t+1)}$	T
0	0	0
0	1	1
1	0	1
1	1	0



$T_C = Q_B$



$T_B = 1$



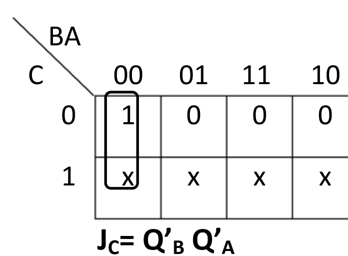
$T_A = 0$

Q) Design 3 bit down counter using J-K F.F

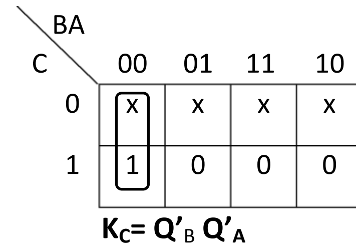
Present State Q_t			Next State Q_{t+1}			J_C	K_C	J_B	K_B	J_A	K_A
C	B	A	C	B	A						
1	1	1	1	1	0	X	0	X	0	X	1
1	1	0	1	0	1	X	0	X	1	1	X
1	0	1	1	0	0	X	0	0	X	X	1
1	0	0	0	1	1	X	1	1	X	1	X
0	1	1	0	1	0	0	X	X	0	X	1
0	1	0	0	0	1	0	X	X	1	1	X
0	0	1	0	0	0	0	X	0	X	X	1
0	0	0	1	1	1	1	X	1	X	1	X

Excitation table of J-K F.F

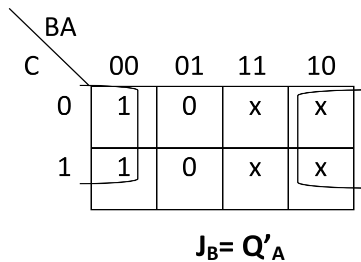
Q_t	$Q_{(t+1)}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	x	0



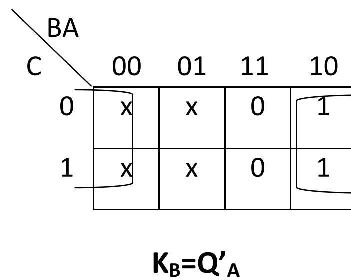
$J_C = Q'_B Q'_A$



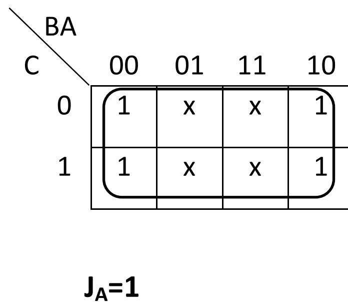
$K_C = Q'_B Q'_A$



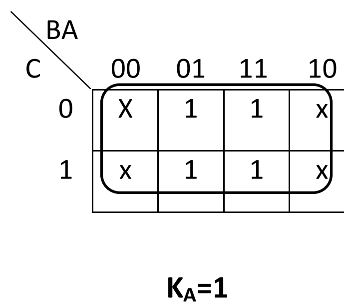
$J_B = Q'_A$



$K_B = Q'_A$



$J_A = 1$



$K_A = 1$

Q) Design 3 bit up-down counter using T F.F

When x=0 up counter: (000→111), when x=1 down counter: (111→000)

Present State Q_t			Up when x=0 Next State Q_{t+1}			Down when x=1 Next State Q_{t+1}			Up x=0			Down x=1		
C	B	A	C	B	A	C	B	A	T_C	T_B	T_A	T_C	T_B	T_A
0	0	0	0	0	1	1	1	1	0	0	1	1	1	1
0	0	1	0	1	0	0	0	0	0	1	1	0	0	1
0	1	0	0	1	1	0	0	1	0	0	1	0	1	1
0	1	1	1	0	0	0	1	0	1	1	1	0	0	1
1	0	0	1	0	1	0	1	1	0	0	1	1	1	1
1	0	1	1	1	0	1	0	0	0	1	1	0	0	1
1	1	0	1	1	1	1	0	1	0	0	1	0	1	1
1	1	1	0	0	0	1	1	0	1	1	1	0	0	1

Excitation table of T F.F

Q_t	$Q_{(t+1)}$	T
0	0	0
0	1	1
1	0	1
1	1	0

CB \ AX	00	01	11	10
00	0	1	0	0
01	0	0	0	1
11	0	0	0	1
10	0	1	0	0

$T_C = Q_B Q_A X' + Q_B Q'_A X$

$= Q_B(Q_A X' + Q_B Q'_A X)$

$= Q_B(Q_A \oplus X)$

CB \ AX	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$T_B = Q_A X' + Q'_A X$

$= Q_A \oplus X$

CB \ AX	00	01	11	10
00	1	1	1	1
01	1	1	1	1
11	1	1	1	1
10	1	1	1	1

$T_A = 1$

Homework

Q1) Design 3 bit odd counter using T F.F

Q2) Design 3 bit even counter using J-K F.F

Q3) Design 3 bit down counter using T F.F

Q4) Design 3 bit down odd counter using J-K F.F

Q5) Design 3 bit down odd counter using T F.F

Q6) Design 3 bit down even counter using J-K F.F

Q7) Design 3 bit down even counter using T F.F

Q8) Design 3 bit gray code (000→001→011→010→110→111→101→100→000) counter using J-K F.F, then using T F.F)

Shift Register

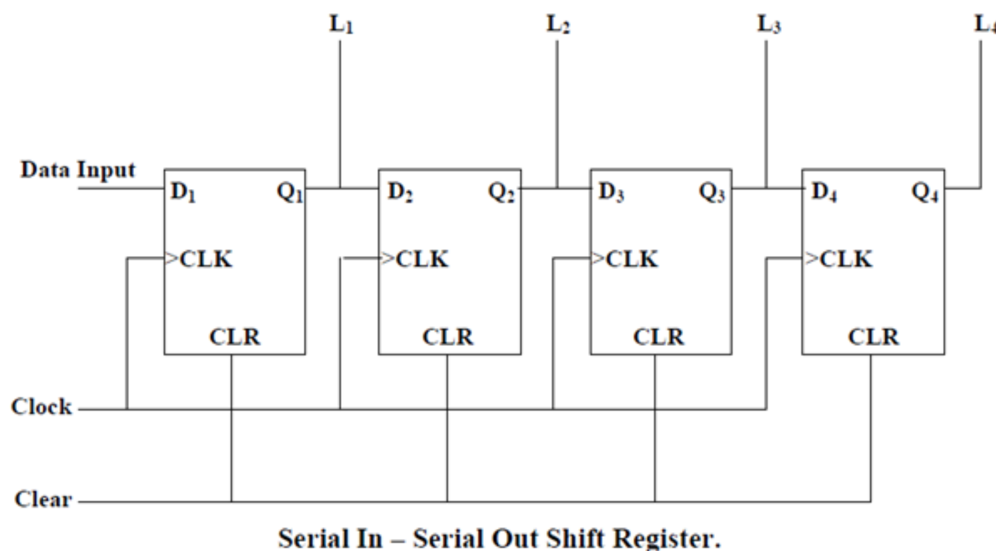
Register: Is a group of binary cells suitable for holding binary information.

Any binary machine is said to have a particular "Word Length". These terms defines the number of bits required to represent data,

In other words, a machine which said to have a four-bit word length has its flip flops arranged in groups of four. The group of flip flops are consider as a single unit called a "Register".

The binary number is "Shifted" one bit at time from one flip flop to the next. The device used in this type of transfer operation it called a "Shift Register"

A shift register is a series of interconnected flip flops used for temporary storage of data as shown in Fig. (1). The output of one flip flop becomes the input of another, all the flip flops in the shift register have a common clock signal connection and all can be set or reset at the same time. Because the data were loaded to the circuit one bit after another and the shift register shifted them from one flip flop to another, this sequence is referred to as serial data loading and the circuit is called a "4-BIT SERIAL IN-SERIAL OUT SHIFT REGISTER" as shown in the following figure.

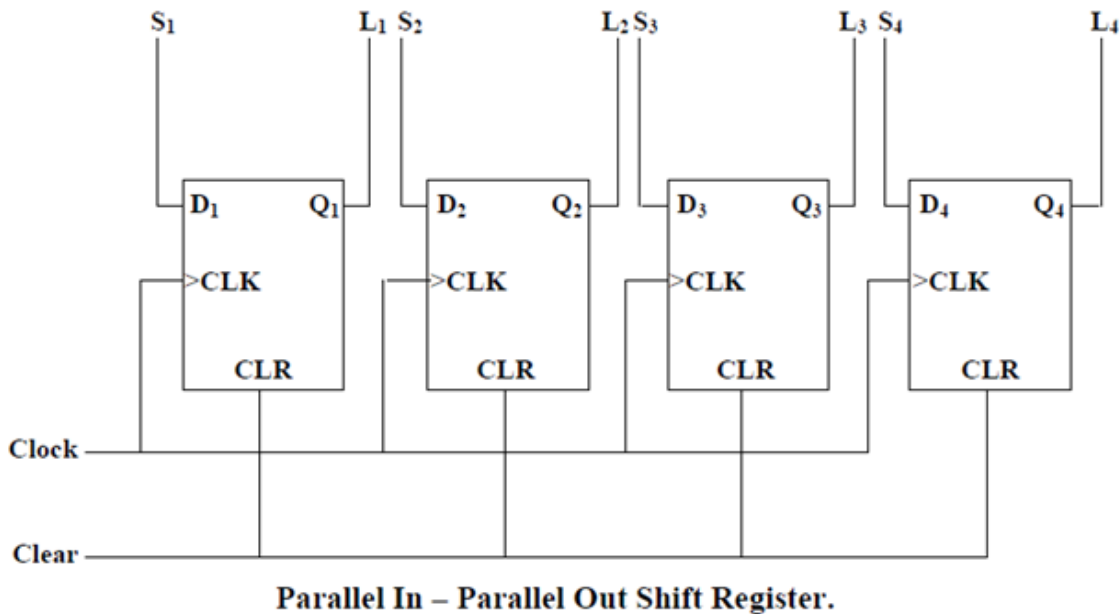


Serial In – Serial Out Shift Register.

Clock	D_1	D_2	D_3	D_4	Q_1	Q_2	Q_3	Q_4
1	1	0	0	0	1	0	0	0
2	1	1	0	0	1	1	0	0
3	1	1	1	0	1	1	1	0
4	1	1	1	1	1	1	1	1

This type of shift register accepts digital data serially that is one bit at the time on one line. It produces the stored information on its output also in serial form.

The alternative to serial loading of the shift register is parallel loading, for a register with parallel data input, the bits are entered simultaneously into their respective stages on parallel-lines, rather than on a bit-by-bit basis on one line as with serial data inputs. The following figure Shows a "4 BIT PARALLEL IN-PARALLEL OUT REGISTER". In the parallel output register the output of each stage is available, once the data are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis as with the serial output.



Parallel In – Parallel Out Shift Register.

Clock	D_1	D_2	D_3	D_4	Q_1	Q_2	Q_3	Q_4
1	1	0	1	1	1	0	1	1
2	0	1	0	1	0	1	0	1
3	1	1	1	0	1	1	1	0

If the data are loaded serially and read out in parallel, the shift register is functioning as a "SERIAL-TO-PARALLEL CONVERTER". If the data is loaded in parallel and shifted out serially, the shift register is functioning as a "PARALLEL-TO-SERIAL CONVERTER". Some shift registers are configured to allow shifting the data in both the right and left direction. These shift registers are usually called "Universal Shift Register", because they can shift data in either right or left direction, can load data either serially or in parallel and can output data either serially or in parallel.

Notes:-

- A register is a digital electronic device capable of storing several bits of data:
  - Normally made from D-type flip-flops.
  - Multiple flip flops can be combined to form a data register Shift registers allow data to be transported one bit at a time.
  - Registers also allow for parallel transfer, many bits transferred at the same time.
- Operation
  - Data input is stored in the flip-flop on the +/- ve edge of the clock.
  - The data can be read from the Q outputs
  - New data can be reloaded by re-CLOCKing the register