ICT 2103 Digital Logic Design

Lecture

Prof. Dr. Muhammad Shahin Uddin

Email: shahin.uddin@mbstu.ac.bd

Textbook:

Digital Design

Morris Mano and Michae D. Ciletti, 2012, 5th Edition Pearson Education, ISBN-13: 978-0-13-277420-8

Reference Books:

1. Digital Design: Principles and Practices, J F Wakerly, 4thed, Prentice Hall, 2005
2. Digital Logic Techniques, T J Stonham, 3rded, Chapman & Hall, 1996

Digital Logic Design

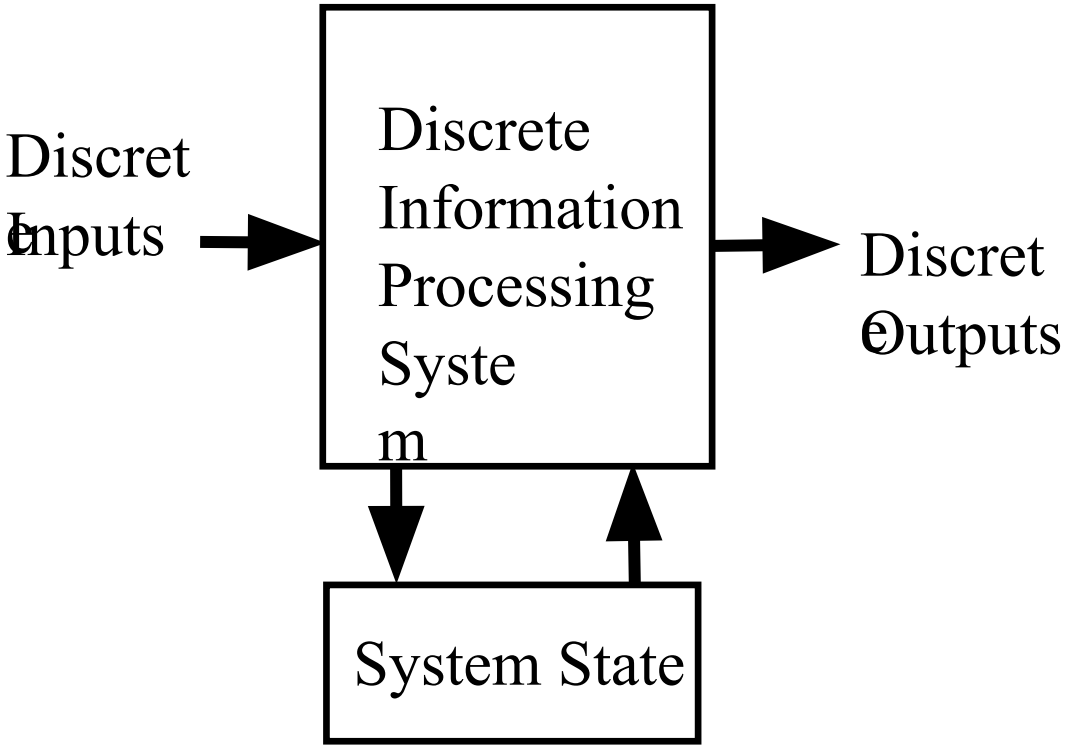
- Let me ask you few questions first…..(**prerequisite knowledge?**)
  - Did you take course on electrical circuits ??
  - Did you take course on electronics (Analog) ??
  - Do you know about signals (Analog/digital) ??

Digital Systems and Numbers

- Digital Systems
- Binary Numbers
- Number-base Conversions
- Octal and Hexadecimal Numbers
- Complements
- Signed Binary Numbers
- Binary Codes

Digital Systems

Takes a set of discrete information inputs and discrete internal information (system state) and generates a set of discrete information outputs.



Digital Systems

- Application of Digital System
  - Communication
  - Business transactions
  - Traffic control
  - Spacecraft guidance
  - Medical treatment
  - Weather monitoring
  - Internet
  - Telephone switching exchanges
  - Digital camera
  - Electronic calculators, PDA's
  - Digital TV
  - Digital telephones
  - Digital televisions
  - Digital versatile discs
  - Digital cameras
  - Handheld devices, and
  - Digital computers
  - Many other commercial, industrial, and scientific enterprises.

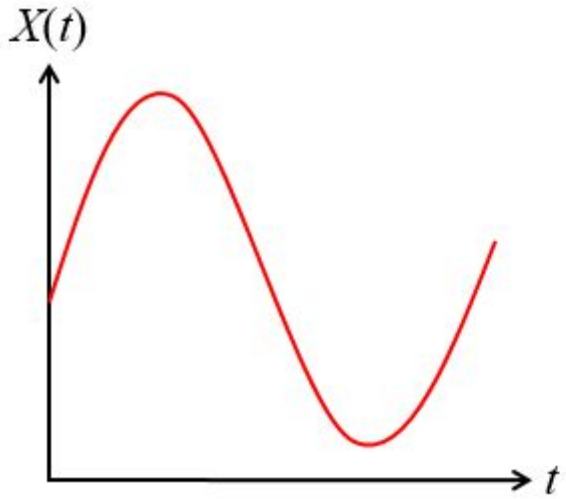
Analog and Digital Signal

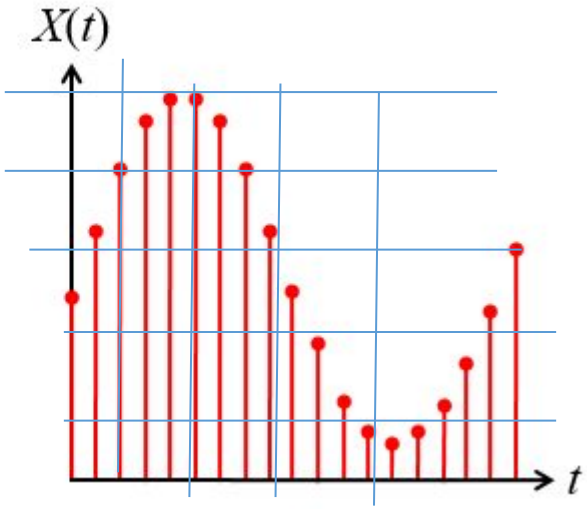
❖Analog system

- The physical quantities or signals may vary continuously over a specified range.

❖Digital system

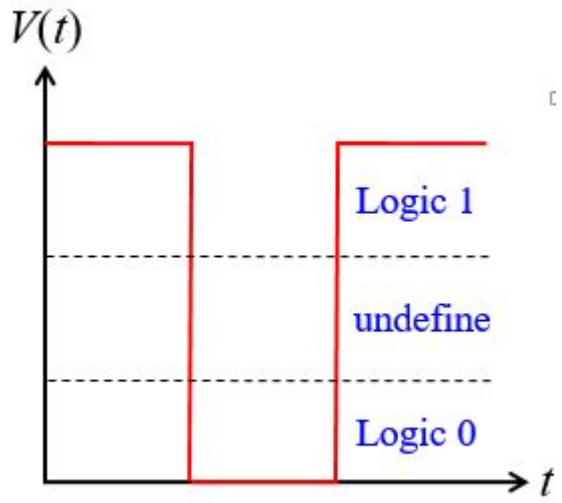
- The physical quantities or signals can assume only discrete values.
- Greater accuracy





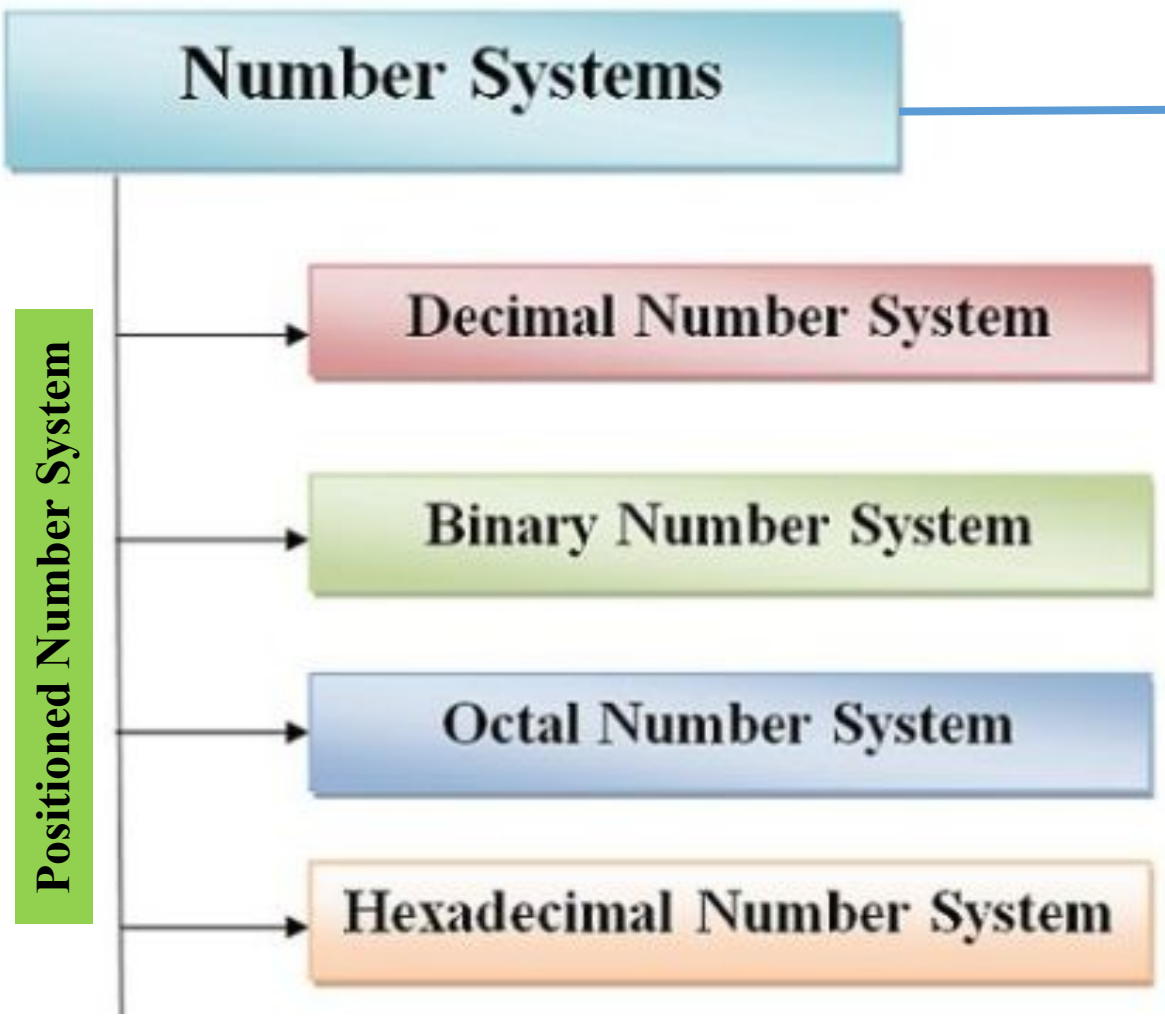
Binary Digital Signal

- An information variable represented by physical quantity.
- For digital systems, the variable takes on discrete values.
  - Two level, or binary values are the most prevalent values.
- Binary values are represented abstractly by:
  - Digits 0 and 1
  - Words (symbols) False (F) and True (T)
  - Words (symbols) Low (L) and High (H)
  - And words on and off
- Binary values are represented by values or ranges of values of physical quantities.



Binary digital signal

Number System



Non-Positioned Number System

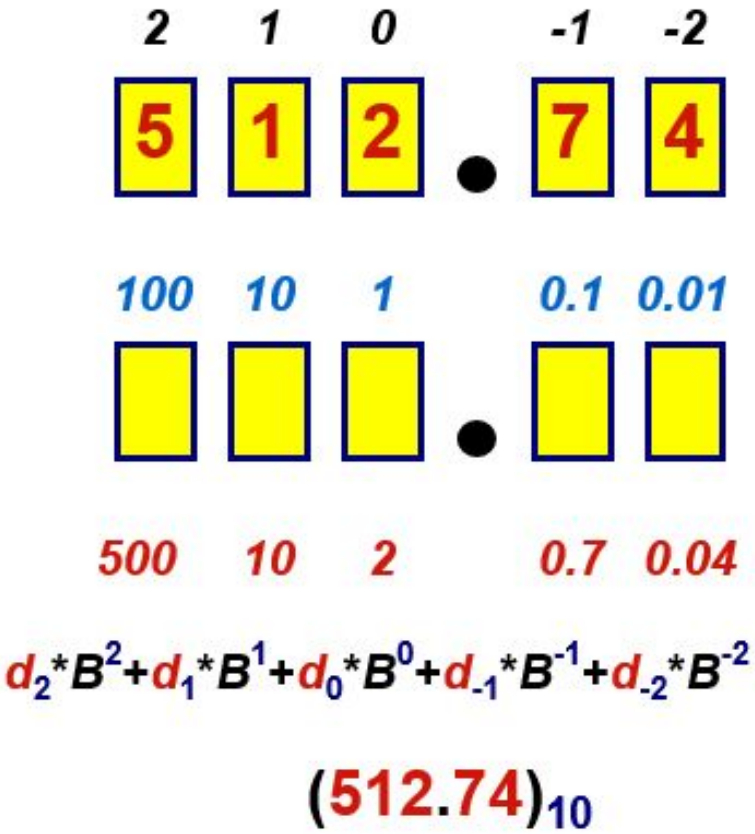
Roman Number

I = 1, V = 5, X = 10, L = 50, C = 100, D = 500, M = 1000.

1999 is MCMXCIX, or M (1000) + CM (1000-100) + XC (100-10) + IX (10-1).

Decimal Number System

- Base (also called radix) = 10
  - 10 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 }
- Digit Position
  - Integer & fraction
- Digit Weight
  - Weight = $(Base)^{Position}$
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation



Octal Number System

- Base = 8
  - 8 digits { 0, 1, 2, 3, 4, 5, 6, 7 }
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

64	8	1		1/8	1/64
5	1	2	•	7	4
2	1	0		-1	-2

$$5 * 8^{2} + 1 * 8^{1} + 2 * 8^{0} + 7 * 8^{-1} + 4 * 8^{-2}$$

$$=(330.9375)_{10}$$

$$(512.74)_{8}$$

Binary Number System

- Base = 2
  - 2 digits { 0, 1 }, called *b*inary dig*its* or "*bits*"
- Weights
  - Weight = (*Base*) ^{Position}
- Magnitude
  - Sum of "*Bit* x *Weight*"
- Formal Notation
- Groups of bits

8 bits = *Byte*

4	2	1		1/2	1/4
1	0	1	.	0	1
2	1	0		-1	-2

$$1 * 2^{2} + 0 * 2^{1} + 1 * 2^{0} + 0 * 2^{-1} + 1 * 2^{-2}$$

$$=(5.25)_{10}$$

$$(101.01)_{2}$$

1 0 1 1

1 1 0 0 0 1 0 1

Binary Number

Powers of Two

n	2^n	n	2^n	n	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024 (1K)	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096 (4K)	20	1,048,576 (1M)
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

Hexadecimal Number System

- Base = 16
  - 16 digits { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F }
- Weights
  - Weight = $(Base)^{Position}$
- Magnitude
  - Sum of "*Digit* x *Weight*"
- Formal Notation

256	16	1		1/16	1/256
1	E	5	•	7	A
2	1	0		-1	-2

$$1 * 16^{2} + 14 * 16^{1} + 5 * 16^{0} + 7 * 16^{-1} + 10 * 16^{-2}$$

$$= (485.4765625)_{10}$$

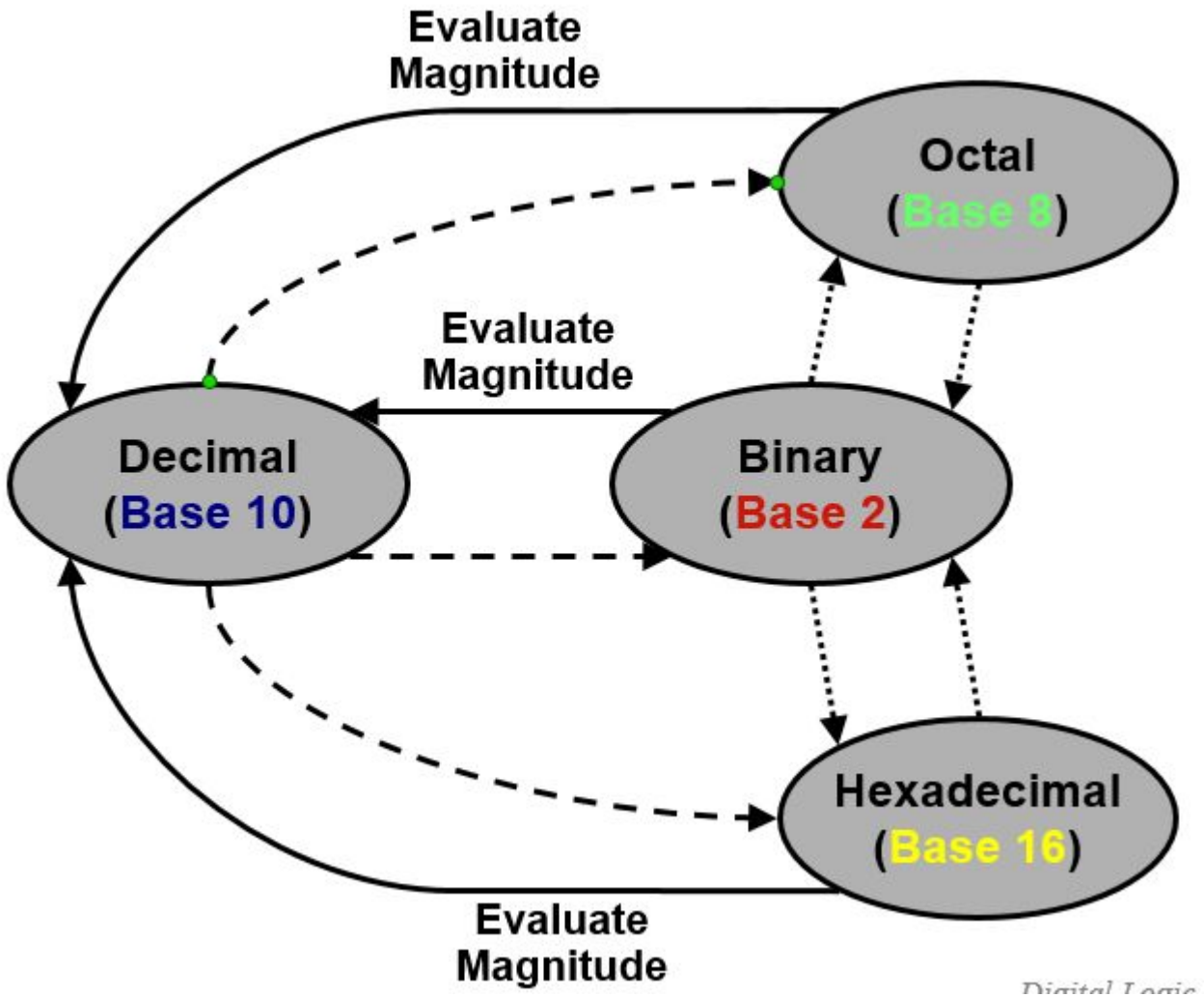
$$(1E5.7A)_{16}$$

The Power of 2

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	$2^{10}=1024$	Kilo
11	$2^{11}=2048$	
12	$2^{12}=4096$	
20	$2^{20}=1M$	Mega
30	$2^{30}=1G$	Giga
40	$2^{40}=1T$	Tera

Number Base Conversions



Decimal (*Integer*) to Binary Conversion

- Divide the number by the 'Base' (=2)
- Take the remainder (either 0 or 1) as a coefficient
- Take the quotient and repeat the division
- Example:

$(13)_{10}$

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

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

MSB LSB

Decimal (*Fraction*) to Binary Conversion

- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division
- Example: $(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

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

MSB LSB

Decimal (*Fraction*) to Binary Conversion

- Multiply the number by the 'Base' (=2)
- Take the integer (either 0 or 1) as a coefficient
- Take the resultant fraction and repeat the division
- Example:

$(0.625)_{10}$

		Integer		Fraction	Coefficient
0.625	* 2 =	1	.	25	$a_{-1} = 1$
0.25	* 2 =	0	.	5	$a_{-2} = 0$
0.5	* 2 =	1	.	0	$a_{-3} = 1$

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

MSB LSB

Decimal to Octal Conversion

Example: $(175)_{10}$

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

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

Example: $(0.3125)_{10}$

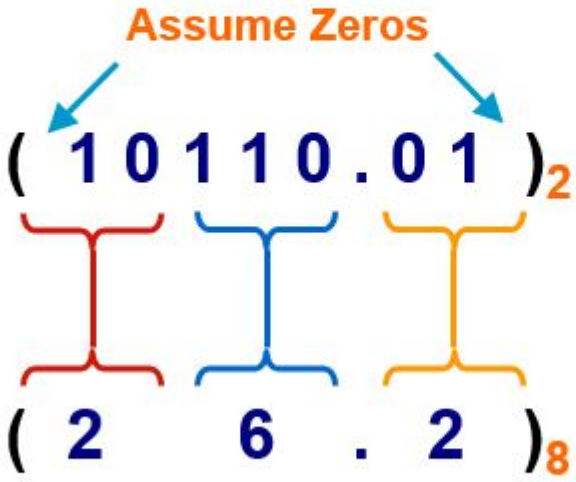
	Integer		Fraction	Coefficient
0.3125 * 8 =	2	.	5	$a_{-1} = 2$
0.5 * 8 =	4	.	0	$a_{-2} = 4$

Answer: $(0.3125)_{10} = (0.a_{-1}\, a_{-2}\, a_{-3})_8 = (0.24)_8$

Binary– Octal Conversion

- $8 = 2^3$
- Each group of 3 bits represents an octal digit

Example:



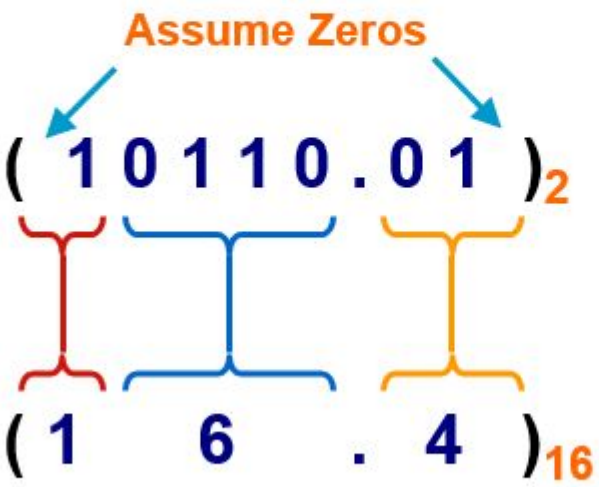
Octal	Binary
0	0 0 0
1	0 0 1
2	0 1 0
3	0 1 1
4	1 0 0
5	1 0 1
6	1 1 0
7	1 1 1

Works both ways (*Binary* to *Octal* & *Octal* to *Binary*)

Binary–Hexadecimal Conversion

- $16 = 2^4$
- Each group of 4 bits represents a hexadecimal digit

Example:



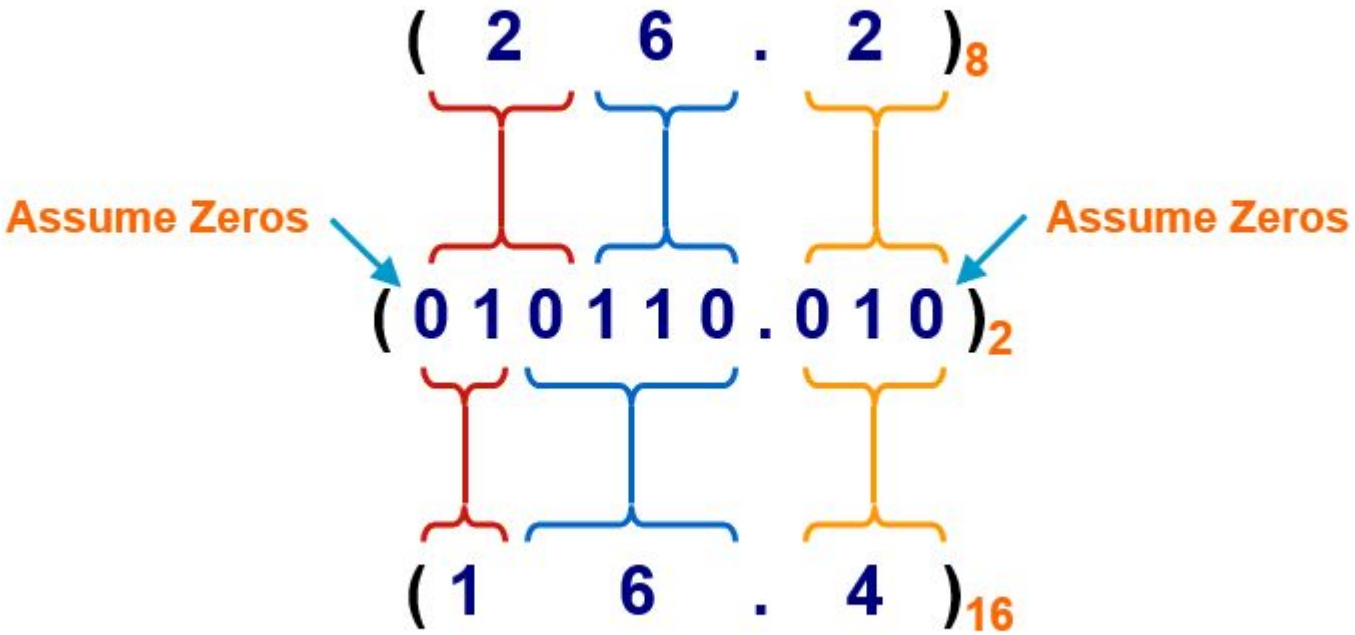
Hex	Binary
0	0 0 0 0
1	0 0 0 1
2	0 0 1 0
3	0 0 1 1
4	0 1 0 0
5	0 1 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
A	1 0 1 0
B	1 0 1 1
C	1 1 0 0
D	1 1 0 1
E	1 1 1 0
F	1 1 1 1

Works both ways (*Binary* to *Hex* & *Hex* to *Binary*)

Octal−Hexadecimal Conversion

Convert to Binary as an intermediate step

Example:



Works both ways (*Octal* to *Hex* & *Hex* to *Octal*)

Decimal, Binary, Octal and Hexadecimal

Decimal	Binary	Octal	Hex
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	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

Thanks

Addition

$$
\begin{array}{rrrl}
1 & 1 & & \leftarrow \text{Carry} \\
 & 5 & 5 & \\
+ & 5 & 5 & \\
\hline
1 & 1 & 0 &
\end{array}
$$

Binary Addition

	1	1	1	1	1	1	
		1	1	1	1	0	1	= 61
+			1	0	1	1	1	= 23
	1	0	1	0	1	0	0	= 84

Binary Subtraction

Borrow a "Base" when needed

		1			2			$= (10)_2$
	0	~~2~~	2	0	0	2		
	~~1~~	0	0	~~1~~	~~1~~	0	1	= 77
−			1	0	1	1	1	= 23
	0	1	1	0	1	1	0	= 54

Binary Multiplication

Bit by bit

			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

Base Conversion

- **Examples**:
  - $(11010)_2 = (\ ?\ )_{10}$
  - $(627)_8 = (\ ?\ )_{10}$

Base Conversion

- $(11010)_2 \rightarrow (\ ?\ )_{10}$

- $(627)_8 \rightarrow (\ ?\ )_{10}$

Examples of Base Conversion (3)

- $(315)_{10} = (473)_{8}$

- $(315)_{10} = (13B)_{16}$

Base Conversion (5)

- $(0.479)_{10} = (0.3651\ldots)_8$

MSD	$3.832 \leftarrow 0.479 \times 8$	
	$6.656 \leftarrow 0.832 \times 8$	
	$5.248 \leftarrow 0.656 \times 8$	
LSD	$1.984 \leftarrow 0.248 \times 8$	
	…	

- $(0.479)_{10} = (0.0111\ldots)_2$

MSD	$0.9580 \leftarrow 0.479 \times 2$	
	$1.9160 \leftarrow 0.9580 \times 2$	
	$1.8320 \leftarrow 0.9160 \times 2$	
LSD	$1.6640 \leftarrow 0.8320 \times 2$	
	…	

Example 1

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The coefficients of the desired binary number are obtained from the remainders as follows:

Example 1

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The coefficients of the desired binary number are obtained from the remainders as follows:

	Integer Quotient		Remainder	Coefficient
$41/2 =$	20	+	$\frac{1}{2}$	$a_0 = 1$
$20/2 =$	10	+	0	$a_1 = 0$
$10/2 =$	5	+	0	$a_2 = 0$
$5/2 =$	2	+	$\frac{1}{2}$	$a_3 = 1$
$2/2 =$	1	+	0	$a_4 = 0$
$1/2 =$	0	+	$\frac{1}{2}$	$a_5 = 1$

Therefore, the answer is $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$.

Example 1

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The coefficients of the desired binary number are obtained from the remainders as follows:

The arithmetic process can be manipulated more conveniently as follows:

Integer	Remainder
41	
20	1
10	0
5	0
2	1
1	0
0	1 101001 = answer

Conversion from decimal integers to any base-*r* system is similar to this example, except that division is done by *r* instead of 2.

Example 2

Convert decimal 153 to octal. The required base r is 8. First, 153 is divided by 8 to give an integer quotient of 19 and a remainder of 1. Then 19 is divided by 8 to give an integer quotient of 2 and a remainder of 3. Finally, 2 is divided by 8 to give a quotient of 0 and a remainder of 2. This process can be conveniently manipulated as follows:

Example 2

Convert decimal 153 to octal. The required base r is 8. First, 153 is divided by 8 to give an integer quotient of 19 and a remainder of 1. Then 19 is divided by 8 to give an integer quotient of 2 and a remainder of 3. Finally, 2 is divided by 8 to give a quotient of 0 and a remainder of 2. This process can be conveniently manipulated as follows:

153	
19	1
2	3
0	$2 = (231)_8$

The conversion of a decimal *fraction* to binary is accomplished by a method similar to that used for integers. However, multiplication is used instead of division, and integers instead of remainders are accumulated. Again, the method is best explained by example.

Complements

- r’s complements
- (r-1)’s complements

	r’s	(r-1)’s
r=10	10’s com	9’s
r=2	2’s com	1’s
r=8	8’s com	7’s
r=16	16’s com	15’s

Complements

- Complement number system:
  - An alternative representation of numbers in a fixed-radix number system.
- Why complement of a number is used?
  - Simplify the subtraction operation
  - Logical manipulations.
- For each radix-r system (radix r represents base of number system) there are two types of complements.

Binary Subtraction using 1's complement

- Step 1: Convert number to be subtracted to it's 1's complement.
- Step 2: Perform addition
- If the final carry is 1 then add it to the results obtained in step 2 is negative in step 2
- Example: Perform $(1100)_2 - (0101)_2$

A=1100 and B=0101

1's complement of B =1010=-B

```
  1 1 0 0
  1 0 1 0
---------
101 1 0
      1
---------
 0 1 1 1
```

Complements

- There are two types of complements for each base-*r* system: the radix complement and diminished radix complement.
- **Diminished Radix Complement - (r-1)'s Complement**
  - Given a number *N* in base *r* having *n* digits, the (*r–1*)'s complement *of N* is defined as:

    $$(r^n - 1) - N$$
- **Example for 6-digit <u>decimal</u> numbers**:
  - 9's complement is $(r^n - 1) - N = (10^6 - 1) - N = 999999 - N$
  - 9's complement of 546700 is 999999–546700 = 453299
- **Example for 7-digit <u>binary</u> numbers:**
  - 1's complement is $(r^n - 1) - N = (2^7 - 1) - N = 1111111 - N$
  - 1's complement of 1011000 is 1111111–1011000 = 0100111
- **Observation:**
  - Subtraction from $(r^n - 1)$ will never require a borrow
  - Diminished radix complement can be computed digit-by-digit
  - For binary: 1 – 0 = 1 and 1 – 1 = 0

Complements

1's Complement (*Diminished Radix* Complement)

- All '0's become '1's
- All '1's become '0's

Example $(10110000)_2$

⇨ $(01001111)_2$

If you add a number and its 1's complement …

$$
\begin{array}{r}
1\,0\,1\,1\,0\,0\,0\,0 \\
+\;0\,1\,0\,0\,1\,1\,1\,1 \\
\hline
1\,1\,1\,1\,1\,1\,1\,1
\end{array}
$$

Complements

Radix Complement:

The r's complement of an n digit number in base r is defined as (r^n-N) for $N\neq0$ and as 0 for N=0. Comparing the (r-1)'s complement, we note that the r's complement is obtained by adding 1 to the (r-1)'s complement, since $(r^n-N)=[(r^n-1)-N]+1$.

Example: Base-10

The 10's complement of 012398 is 987602
The 10's complement of 246700 is 753300

Example: Base-2

The 2's complement of 1101100 is 0010100
The 2's complement of 0110111 is 1001001

Complements

Radix Complement:

The r's complement of an n digit number in base r is defined as (r^n-N) for $N\neq0$ and as 0 for N=0. Comparing the (r-1)'s complement, we note that the r's complement is obtained by adding 1 to the (r-1)'s complement, since $(r^n-N)=[(r^n-1)-N]+1$.

Example: Base-10

The 10's complement of 012398 is 987602
The 10's complement of 246700 is 753300

1000000-1=9999999
999999-012398=987601+1

Example: Base-2

The 2's complement of 1101100 is 0010100
The 2's complement of 0110111 is 1001001

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

Complements

2's Complement (*Radix* Complement)

- Take 1's complement then add 1

OR

- Toggle all bits to the left of the first '1' from the right

Example:

Number:

1's Comp.:

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

Complements

❖ Subtraction with Complements

- The subtraction of two n-digit unsigned numbers $M - N$ in base r can be done as follows:

1. Add the minuend M to the r's complement of the subtrahend N. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geqq N$, the sum will produce and end carry r^n, which can be discarded; what is left is the result $M - N$.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the r's complement of $(N - M)$. To obtain the answer in a familiar form, take the r's complement of the sum and place a negative sign in front.

Complements

Example

- Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction (a) $X - Y$; and (b) $Y - X$, by using 2's complement.

(a)

X =	1010100
2's complement of Y =	+0111101
Sum =	10010001
Discard end carry 2^7 =	−10000000
Answer. X − Y =	0010001

(b)

Y =	1000011
2's complement of X =	+ 0101100
Sum =	1101111

There is no end carry. Therefore, the answer is Y − X = − (2's complement of 1101111) = − 0010001.

Complements

Subtraction of unsigned numbers can also be done by means of the $(r-1)$'s complement. Remember that the $(r-1)$'s complement is one less then the r's complement.

Example

- Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$X =$	1010100
1's complement of $Y =$	± 0111100
Sum =	10010000
End-around carry =	+ 1
Answer. $X - Y =$	0010001

(b) $Y - X = 1000011 - 1010100$

$Y =$	1000011
1's complement of $X =$	+ 0101011
Sum =	1101110

There is no end carry, Therefore, the answer is Y – X = – (1's complement of 1101110) = – 0010001.

Signed Binary Numbers

To represent negative integers, we need a notation for negative values.

It is customary to represent the sign with a bit placed in the leftmost position of the number since binary digits.

The convention is to make the sign bit 0 for positive and 1 for negative.

Example:

Signed-magnitude representation:	10001001
Signed-1's-complement representation:	11110110
Signed-2's-complement representation:	11110111

Table lists all possible four-bit signed binary numbers in the three representations.

Signed Binary Numbers

Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

Signed Binary Numbers

Arithmetic addition

- The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. **If the signs are the same**, we add the two magnitudes and give the sum the common sign. **If the signs are different**, we subtract the smaller magnitude from the larger and give the difference the sign if the larger magnitude.
- The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits.
- A carry out of the sign-bit position is discarded.

Example:

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+ 19	00010011	+ 7	00000111
+ 6	00000110	− 6	11111010
−13	11110011	−13	11110011
− 7	11111001	− 19	11101101

Binary Codes

BCD Code

- In this **code** each decimal digit is represented by a 4-bit binary number.**BCD** is a way to express each of the decimal digits with a binary **code**
- A number with k decimal digits will require 4k bits in BCD.
- Decimal 396 is represented in BCD with 12bits as 0011 1001 0110, with each group of 4 bits representing one decimal digit.
- The binary combinations 1010 through 1111 are not used and have no meaning in BCD.

Binary-Coded Decimal (BCD)

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

Binary Code

Example:

- Consider decimal 185 and its corresponding value in BCD and binary:

$$(185)_{10} = (0001\ 1000\ 0101)_{BCD} = (10111001)_2$$

BCD addition

4	0100	4	0100	8	1000
+5	+0101	+8	+1000	+9	+1001
9	1001	12	1100	17	10001
			+0110		+0110
			10010		10111

Binary Code

Example:

- Consider the addition of 184 + 576 = 760 in BCD:

BCD	1	1		
	0001	1000	0100	184
	+0101	0111	0110	+576
Binary sum	0111	10000	1010	
Add 6	____	0110	0110	____
BCD sum	0111	0110	0000	760

Binary Codes

- Other Decimal Codes – weighted representation of decimal numbers in binary.

Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused	1011	0110	0001	0010
bit	1100	0111	0010	0011
combi-	1101	1000	1101	1100
nations	1110	1001	1110	1101
	1111	1010	1111	1110

Gray Code

- **Definition**: Gray Code is the minimum-change code category of coding in which, the two consecutive values changes by only a single bit. More specifically we can say, it is a binary number system where while moving from one step to the next, only a single bit shows variation.

- The advantage is that only bit in the code group changes in going from one number to the next.
    - Error detection.
    - Representation of analog data.
    - Low power design

Gray Code

The conversion process from binary code to a gray involves the following steps:

- Firstly, record the most significant bit or MSB or the leftmost bit of the given
- binary data as it is, to have MSB of gray equivalent.
- Now, proceed towards adding the adjacent bits of the binary data starting from MSB with its adjacent bit to LSB. While adding, put the summation obtained in place of next bit and ignoring the carry.
- Repeat the same process for all the bits in the sequence till LSB.
- This is how the binary code is converted into gray equivalent.

Decimal Value	Binary Code	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
A	1010	1111

BCD-to-Excess-3 Code converter

- BCD is a code for the decimal digits 0-9
- Excess-3 is also a code for the decimal digits

Decimal Digit	Input BCD	Output Excess-3
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0

❑ Specification of BCD-to-Excess3

- **Inputs:** a BCD input, A,B,C,D with A as the most significant bit and D as the least significant bit.
- **Outputs:** an Excess-3 output W,X,Y,Z that corresponds to the BCD input.
- **Internal operation** – circuit to do the conversion in combinational logic.

❑ Formulation of BCD-to-Excess-3

- Excess-3 code is easily formed by adding a binary 3 to the binary or BCD for the digit.
- There are 16 possible inputs for both BCD and Excess-3.
- It can be assumed that only valid BCD inputs will appear so the six combinations not used can be treated as don’t cares.

ASCII Code

- **ASCII** is the acronym for the American Standard **Code** for Information Interchange. It is a **code** for representing 128 English **characters** as numbers, with each letter assigned a number from 0 to 127. For **example**, the **ASCII code** for uppercase M is 77.
- **ASCII**, a standard data-transmission **code** that is **used by** smaller and less-powerful computers to represent both textual data (letters, numbers, and punctuation marks) and noninput-device commands (control **characters**).

ASCII Code

- American Standard Code for Information Interchange (ASCII) Character Code

American Standard Code for Information Interchange (ASCII)

	$b_7b_6b_5$							
$b_4b_3b_2b_1$	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	`	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	–	=	M	]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	–	o	DEL

ASCII Code

ASCII Character Code

Control characters

NUL	Null	DLE	Data-link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End-of-transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

ASCII Character Codes

- American Standard Code for Information Interchange
- A popular code used to represent information sent as character-based data.
- It uses 7-bits to represent:
  - 94 Graphic printing characters.
  - 34 Non-printing characters.
- Some non-printing characters are used for text format (e.g. BS = Backspace, CR = carriage return).

ASCII Properties

- ASCII has some interesting properties:
  - Digits 0 to 9 span Hexadecimal values 30_{16} to 39_{16}
  - Upper case A-Z span 41_{16} to $5A_{16}$
  - Lower case a-z span 61_{16} to $7A_{16}$
    - Lower to upper case translation (and vice versa) occurs by flipping bit 6.

Extended ASCII Code/ Error Detection Code

- To detect errors in data communication and processing, an eighth bit is sometimes added to the ASCII character to indicate its parity.
- A parity bit is an extra bit included with a message to make the total number of 1's either even or odd.
- Consider the following two characters and their even and odd parity:

	With even parity	With odd parity
ASCII A = 1000001	01000001	11000001
ASCII T = 1010100	11010100	01010100

- The parity bit is helpful in detecting errors during the transmission of information.
- Handled by generating an even parity bit at the sending end for each character.
- The eight-bit characters that include parity bits are transmitted to their destination.
- The parity of each character is then checked at the receiving end.
- If the parity of the received character is not even, then at least one bit has changed value during the transmission.

Binary Codes

- In the coding, when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded.
- The group of symbols is called as a code.
- The digital data is represented, stored and transmitted as group of binary bits.
- This group is also called as binary code. The binary code is represented by the number as well as alphanumeric letter.

Advantages of Binary Code

- Binary codes are suitable for the computer applications.
- Binary codes are suitable for the digital communications.
- Binary codes make the analysis and designing of digital circuits if we use the binary codes.
- Since only 0 & 1 are being used, implementation becomes easy.

Classification of binary codes

- The codes are broadly categorized into following four categories.
  Weighted Codes
  Non-Weighted Codes
  Binary Coded Decimal Code
  Alphanumeric Codes
  Error Detecting Codes
  Error Correcting Codes

Weighted Codes

- Weighted binary codes are those binary codes which obey the positional weight principle. Each position of the number represents a specific weight. Several systems of the codes are used to express the decimal digits 0 through 9. In these codes each decimal digit is represented by a group of four bits.

Non-Weighted Codes

In this type of binary codes, the positional weights are not assigned. The examples of non-weighted codes are **Excess-3** code and **Gray code.**

EXCESS-3 CODE

The Excess-3 code is also called as XS-3 code. It is non-weighted code used to express decimal numbers. The Excess-3 code words are derived from the 8421 BCD code words adding (0011)2 or (3)10 to each code word in 8421. The excess-3 codes are obtained as follows

Boolean Algebra

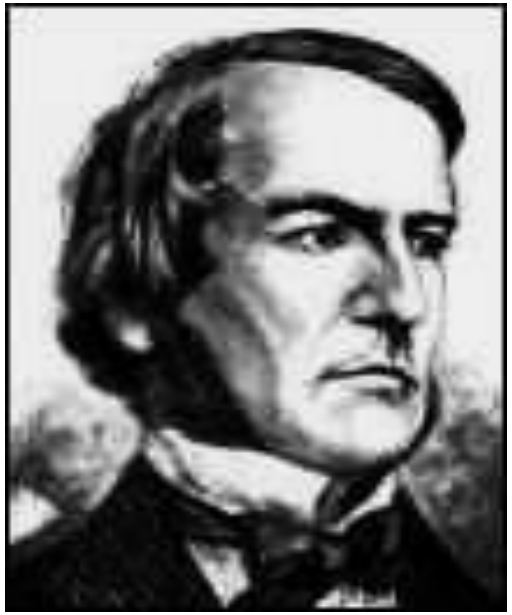
Algebras

- What is an algebra?
  - Mathematical system consisting of
    - Set of elements
    - Set of operators
    - Axioms or postulates
- Why is it important?
  - Defines rules of "calculations"
- Example: arithmetic on natural numbers
  - Set of elements: $N = \{1,2,3,4,\ldots\}$
  - Operator: +, –, * /
  - Axioms: associativity, distributivity, closure, identity elements, etc.
- Note: operators with two inputs are called *binary*
  - Does not mean they are restricted to binary numbers!
  - Operator(s) with one input are called *unary*

George Boole

- Father of Boolean Algebra

- He came up with a type of linguistic algebra, the three most basic operations of which were (and still are) AND, OR and NOT. It was these three functions that formed the basis of his premise, and were the only operations necessary to perform comparisons or basic mathematical functions.
- Boole's system (detailed in his 'An Investigation of the Laws of Thought, on Which Are Founded the Mathematical Theories of Logic and Probabilities', 1854) was based on a binary approach, processing only two objects - the yes-no, true-false, on-off, zero-one approach.
- Surprisingly, given his standing in the academic community, Boole's idea was either criticized or completely ignored by the majority of his peers.
- Eventually, one bright student, Claude Shannon (1916-2001), picked up the idea and ran with it



George Boole (1815 - 1864)

Binary Logic and Gates

- **Binary variables** take on one of two values.
- **Logical operators** operate on binary values and binary variables.
- Basic logical operators are the **logic functions** AND, OR and NOT.
- **Logic gates** implement logic functions.
- **Boolean Algebra:** a useful mathematical system for specifying and transforming logic functions.
- We study Boolean algebra as foundation for designing and analyzing digital systems!

Binary Variables

- Recall that the two binary values have different names:
  - True/False
  - On/Off
  - Yes/No
  - 1/0
- We use 1 and 0 to denote the two values.
- Variable identifier examples:
  - A, B, y, z, or X_1 for now
  - RESET, START_IT, or ADD1 later

Logical Operations

- The three basic logical operations are:
  - AND
  - OR
  - NOT
- AND is denoted by a dot (·).
- OR is denoted by a plus (+).
- NOT is denoted by an overbar (¯), a single quote mark (') after, or (~) before the variable.

Boolean Functions

- A Boolean function
  - Binary variables
  - Binary operators OR and AND
  - Unary operator NOT
  - Parentheses
- Examples
  - $F_1 = x.\ y.\ z'$
  - $F_2 = x + y'.z$
  - $F_3 = x'.y'.z + x'.y.z + x.y'$
  - $F_4 = (x.\ y' + x'.z)$

Properties of Boolean Algebra

Property	AND	OR
Commutative	AB=BA	A+B=B+A
Associative	(AB)C=A(BC)	(A+B)+C=A+(B+C)
Distributive	A(B+C)=AB+AC	A+(BC)=(A+B)(A+C)
Identity	A1=A	A+0=A
Complement	A(A')=0	A+A'=1
DeMorgan's Law	(AB)'=A' + B'	(A+B)'=A'B'

Boolean Algebra: Basic Theorems

- The theorems and postulates listed are the most basic relationships in Boolean algebra.
- Postulates and Theorems of Boolean Algebra are:

Postulate 2	(a)	$x + 0 = x$	(b)	$x \cdot 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b)	$x \cdot x' = 0$
Theorem 1	(a)	$x + x = x$	(b)	$x \cdot x = x$
Theorem 2	(a)	$x + 1 = 1$	(b)	$x \cdot 0 = 0$
Theorem 3, involution		$(x')' = x$		
Postulate 3, commutative	(a)	$x + y = y + x$	(b)	$xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b)	$x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b)	$x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b)	$(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = x$	(b)	$x(x + y) = x$

Proof

THEOREM 1(a): $x + x = x.$

Statement	Justification
$x + x = (x + x) \cdot 1$	postulate 2(b)
$= (x + x)(x + x')$	5(a)
$= x + xx'$	4(b)
$= x + 0$	5(b)
$= x$	2(a)

Proof

THEOREM 1(b): $x \cdot x = x.$

Statement	Justification
$x \cdot x = xx + 0$	postulate 2(a)
$= xx + xx'$	5(b)
$= x(x + x')$	4(a)
$= x \cdot 1$	5(a)
$= x$	2(b)

Proof

THEOREM 2(a): $x + 1 = 1.$

Statement	Justification
$x + 1 = 1 \cdot (x + 1)$	postulate 2(b)
$= (x + x')(x + 1)$	5(a)
$= x + x' \cdot 1$	4(b)
$= x + x'$	2(b)
$= 1$	5(a)

THEOREM 2(b): $x \cdot 0 = 0$ by duality.

Proof

THEOREM 3: $(x')' = x$. From postulate 5, we have $x + x' = 1$ and $x \cdot x' = 0$, which together define the complement of x. The complement of x' is x and is also $(x')'$.

Therefore, since the complement is unique, we have $(x')' = x$. The theorems involving two or three variables may be proven algebraically from the postulates and the theorems that have already been proven. Take, for example, the absorption theorem:

THEOREM 6(a): $x + xy = x$.

Statement	Justification
$x + xy = x \cdot 1 + xy$	postulate 2(b)
$= x(1 + y)$	4(a)
$= x(y + 1)$	3(a)
$= x \cdot 1$	2(a)
$= x$	2(b)

THEOREM 6(b): $x(x + y) = x$ by duality.

Example:

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x + y) = xx + xy = 0 + xy = xy$.
2. $x + xy = (x + x)(x + y) = 1(x + y) = x + y$.
3. $(x + y)(x + y) = x + xy + xy + yy = x(1 + y + y) = x$.
4. $xy + xz + yz = xy + xz + yz(x + x) = xy + xz + xyz + xyz = xy(1 + z) + xz(1 + y) = xy + xz$.
5. $(x + y)(x + z)(y + z) = (x + y)(x + z)$, by duality from function 4.

Example:

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x + y) = xx + xy = 0 + xy = xy$.
2. $x + xy = (x + x)(x + y) = 1(x + y) = x + y$.
3. $(x + y)(x + y) = x + xy + xy + yy = x(1 + y + y) = x$.
4. $xy + xz + yz = xy + xz + yz(x + x) = xy + xz + xyz + xyz = xy(1 + z) + xz(1 + y) = xy + xz$.
5. $(x + y)(x + z)(y + z) = (x + y)(x + z)$, by duality from function 4.

Complement of a Function

- The complement of a function F is F’ and is obtained from an interchange of 0’s for 1’s and 1’s for 0’s in the value of F.
- DeMorgan’s theorems can be extended to three or more variables. The three-variable form of the first DeMorgan’s theorem is derived as follows:

$$(A+B+C)'=(A+x)'=A'x'=A'(B+C)'= A'B'C'$$

DeMorgan’s theorems can be generalized as follows:

$(A + B + C + D + G + F)' = A'B'C'D'\ldots..F'$

$(ABCDF)' = A'+ B'+ C'+D'+G'+F'$

The generalized form of DeMorgan’s theorems states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal.

Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$.

$$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)'$$
$$= (x + y' + z)(x + y + z')$$

$$F_2' = [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)'$$
$$= x' + (y + z)(y' + z')$$
$$= x' + yz' + y'z$$

The operators and postulates have the following meanings:

The binary operator $+$ defines addition.

The additive identity is 0.

The additive inverse defines subtraction.

The binary operator $\cdot$ defines multiplication.

The multiplicative identity is 1.

For $a \neq 0$, the multiplicative inverse of $a = 1/a$ defines division (i.e., $a \cdot 1/a = 1$).

The only distributive law applicable is that of $\cdot$ over $+$:

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

Notation Examples

- Examples:
  - Y =A.B is read "Y is equal to A AND B."
  - Z=X+Y is read "Z is equal to X OR Y."
  - X=A' is read "X is equal to NOT A."

- Note: The statement:

  1 + 1 = 2 (read "one <u>plus</u> one equals two")

  is not the same as

  1 + 1 = 1 (read "1 <u>or</u> 1 equals 1").

Logic Gates

- **Gate**

  A device that performs a basic operation on electrical signals

- **Boolean Expressions**

  Uses Boolean algebra, a mathematical notation for expressing two-valued logic

- **Logic Diagrams**

  A graphical representation of a circuit; each gate has its own symbol

- **Truth Tables**

  A table showing all possible input value and the associated output values

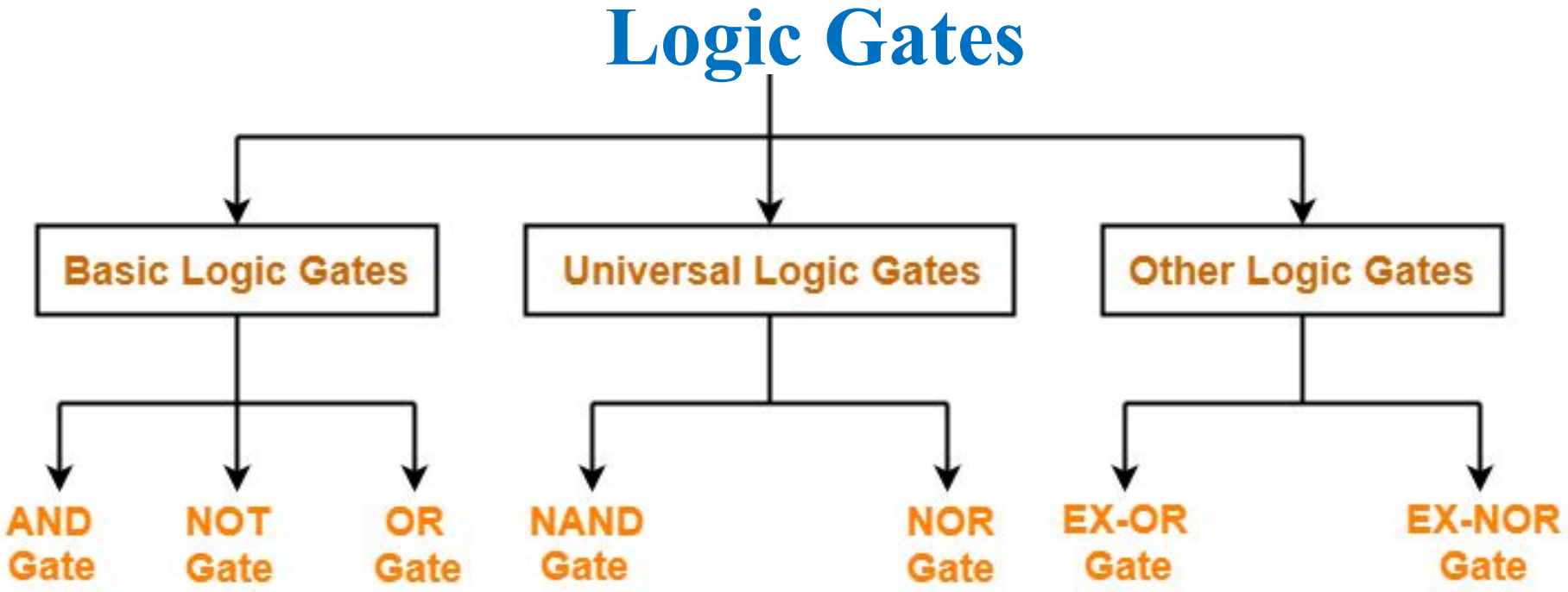
- **Timing Diagram**

Logic Gates

- Logic gates:
  - NOT
  - AND
  - OR
  - EX-OR
  - EX-NOR
  - NAND
  - NOR

Typically, logic diagrams are black and white with gates distinguished only by their shape

Types of Logic Gates

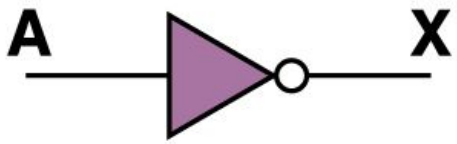


NOT Gate

- A NOT gate accepts one input signal (0 or 1) and returns the opposite signal as output
- The output of NOT gate is high (‘1’) if its input is low (‘0’).
- The output of NOT gate is low (‘0’) if its input is high (‘1’).
- It is clear that NOT gate simply inverts the given input.
- Since NOT gate simply inverts the given input, therefore it is also known as **Inverter Gate**.

Boolean Expression **Logic Diagram Symbol**

$$X = A'$$



Truth Table

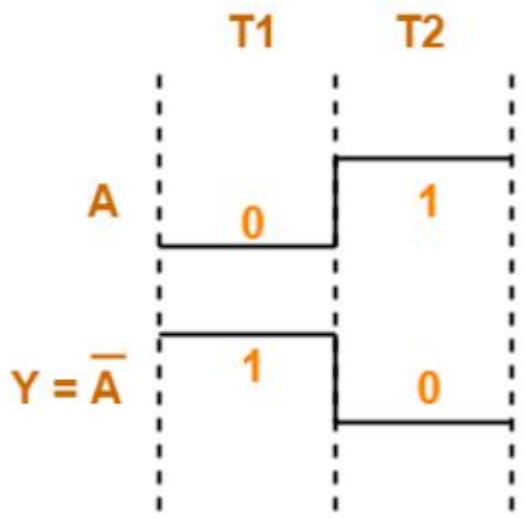
Truth Table for NOT Gate is as shown below-

Truth Table

A	X
0	1
1	0

Timing Diagram-

The timing diagram for NOT Gate is as shown below-



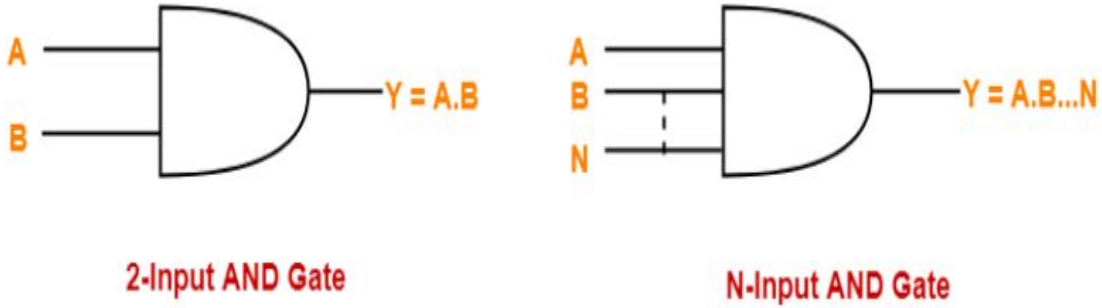
AND Gate

- An AND gate accepts two/ more than two input signals
- The output of AND gate is high (‘1’) if all of its inputs are high (‘1’).
- The output of AND gate is low (‘0’) if any one of its inputs is low (‘0’).

Boolean Expression

$$X = A \cdot B$$

Logic Diagram Symbol



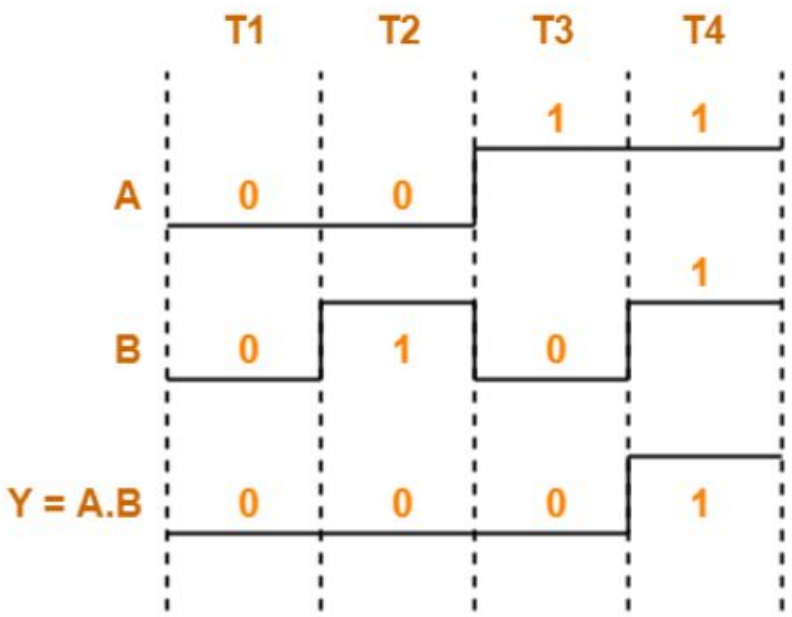
Truth Table-

The truth table for AND Gate is as shown below-

A	B	Y = A.B
0	0	0
0	1	0
1	0	0
1	1	1

Timing Diagram-

The timing diagram for AND Gate is as shown below-

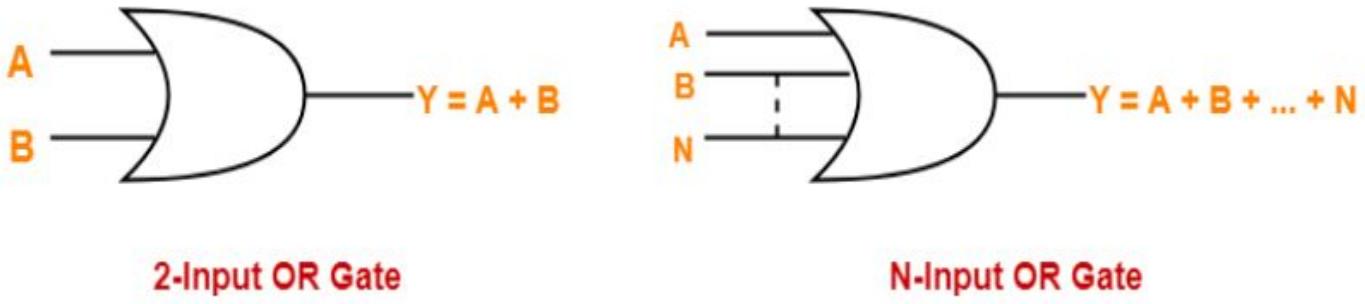


OR Gate

The output of OR gate is high (‘1’) if any one of its inputs is high (‘1’).
The output of OR gate is low (‘0’) if all of its inputs are low (‘0’).

Logic Symbol-

The logic symbol for OR Gate is as shown below-



Truth Table-

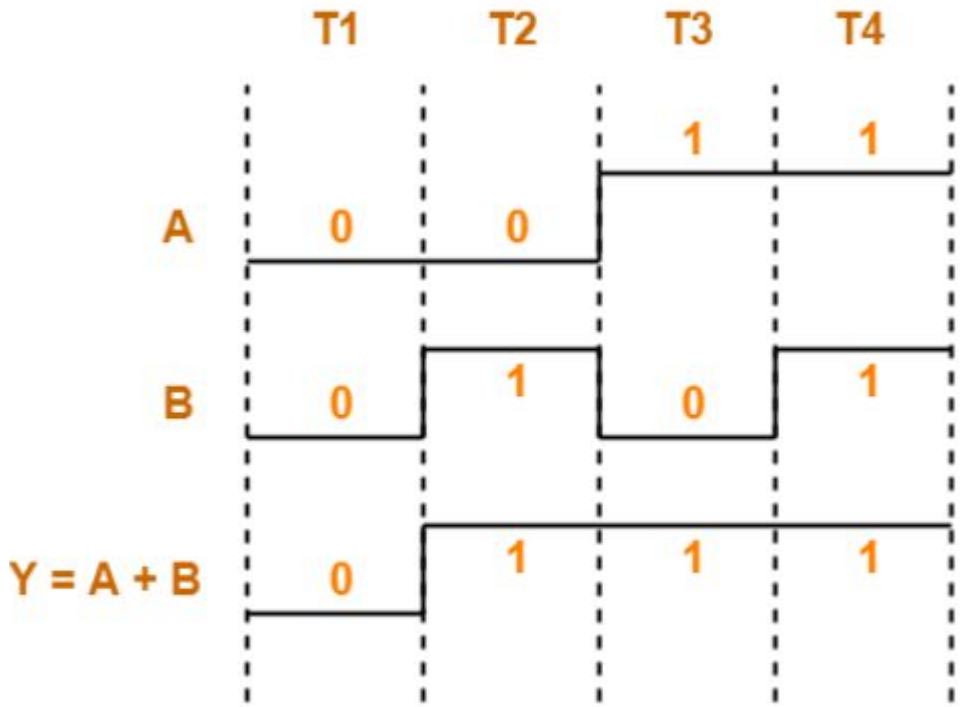
The truth table for OR Gate is as shown below-

A	B	Y = A + B
0	0	0
0	1	1
1	0	1
1	1	1

Truth Table

Timing Diagram-

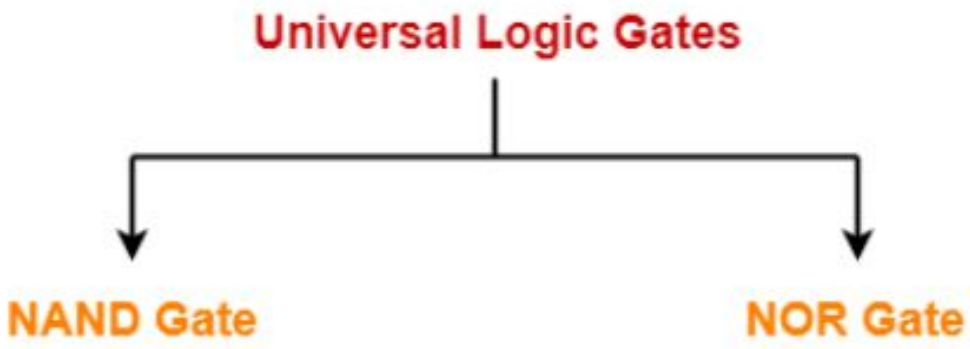
The timing diagram for OR Gate is as shown below-



Universal Logic Gates-

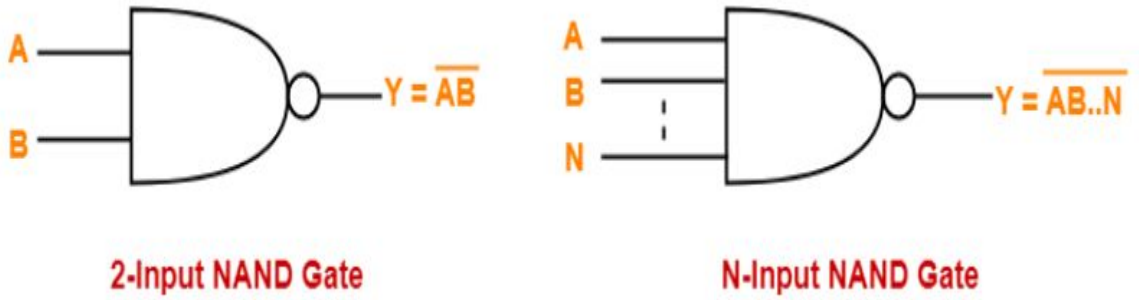
- Universal logic gates are the logic gates that are capable of implementing any Boolean function without requiring any other type of gate.

- They are called as "**Universal Gates**" because- They can realize all the binary operations.
- All the basic logic gates can be derived from them
- They have the following properties-
- Universal gates are not associative in nature.
- Universal gates are commutative in nature.
- There are following two universal logic gates-



Universal Logic Gates: NAND Gate-

- ❖ A NAND Gate is constructed by connecting a NOT Gate at the output terminal of the AND Gate.
- ❖ The output of NAND gate is high (‘1’) if at least one of its inputs is low (‘0’).
- ❖ The output of NAND gate is low (‘0’) if all of its inputs are high (‘1’).
- ❖ Logic Symbol-
- ✔ The logic symbol for NAND Gate is as shown below-



Truth Table-

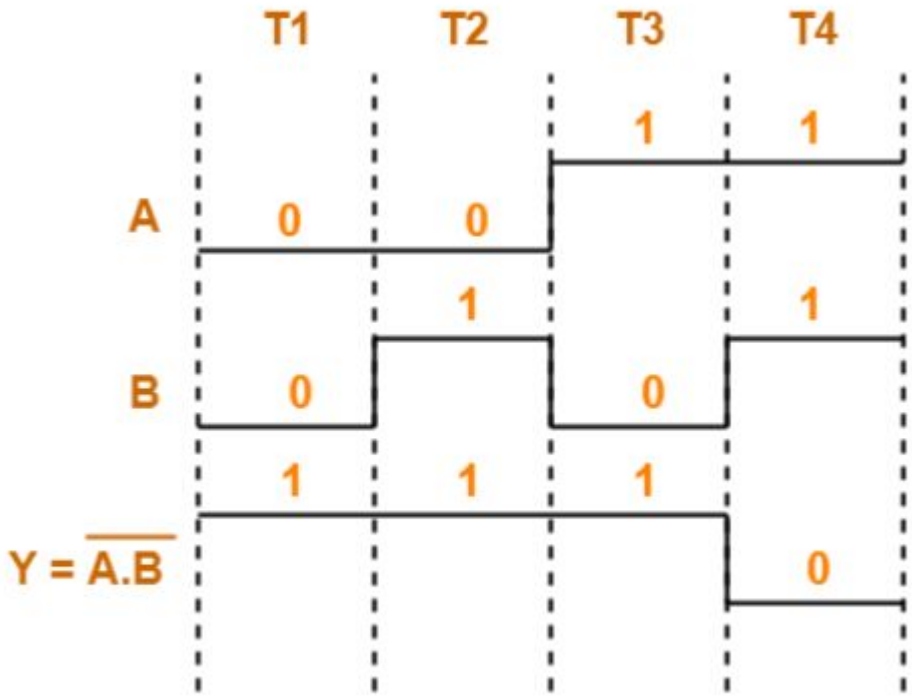
The truth table for NAND Gate is as shown below-

A	B	Y = (A.B)'
0	0	1
0	1	1
1	0	1
1	1	0

Truth Table

Timing Diagram-

The timing diagram for NAND Gate is as shown below-

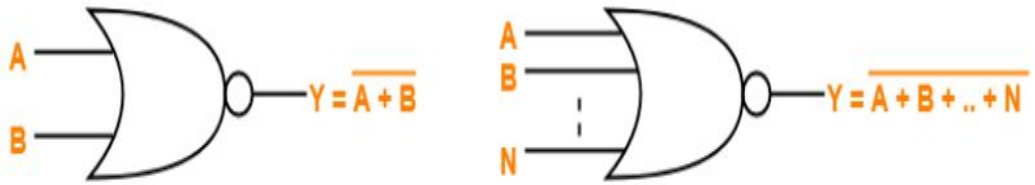


Universal Logic Gates: NOR Gate

- A NOR Gate is constructed by connecting a NOT Gate at the output terminal of the OR Gate.
- The output of OR gate is high (‘1’) if all of its inputs are low (‘0’).
- The output of OR gate is low (‘0’) if any of its inputs is high (‘1’).

Logic Symbol-

The logic symbol for NOR Gate is as shown below-



2-Input NOR Gate

N-Input NOR Gate

Truth Table-

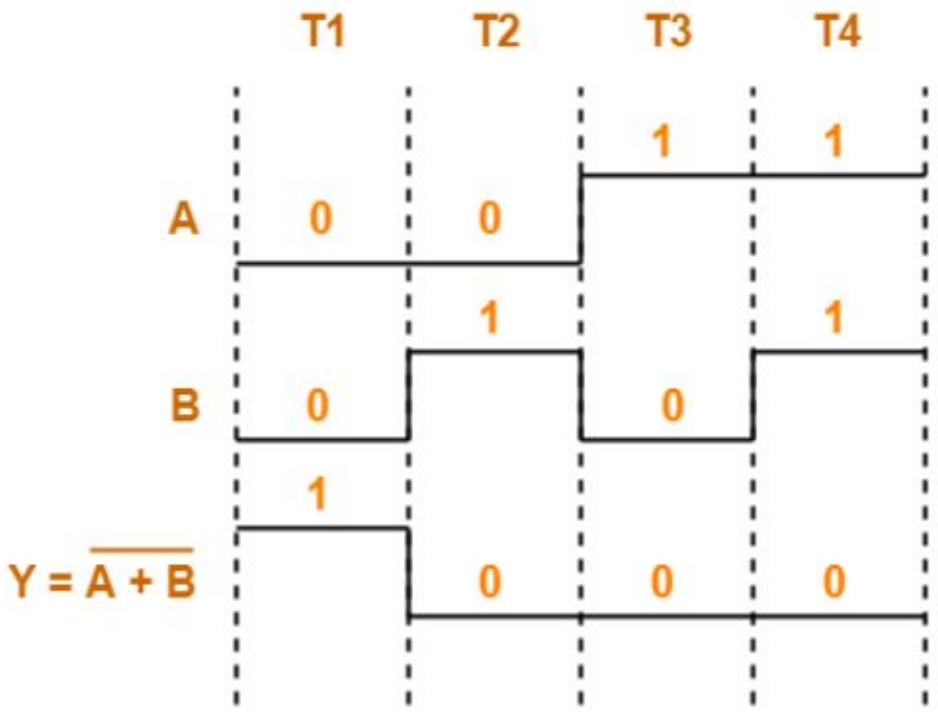
The truth table for NOR Gate is as shown below-

A	B	Y = A + B
0	0	1
0	1	0
1	0	0
1	1	0

Truth Table

Timing Diagram-

The timing diagram for NOR Gate is as shown below-



XOR Gate

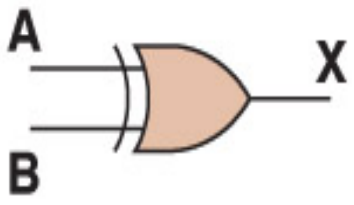
An XOR gate accepts two input signals

If both are the same, the output is 0; otherwise, the output is 1

Boolean Expression

$$X = A \oplus B$$

Logic Diagram Symbol



Truth Table

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

EX-OR Gate

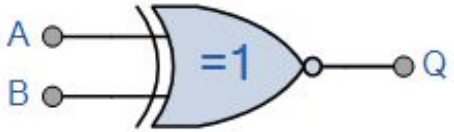
Note the difference between the XOR gate
and the OR gate; they differ only in one input situation

When both input signals are 1, the OR gate produces a 1 and the XOR produces a 0

XOR is called the exclusive OR

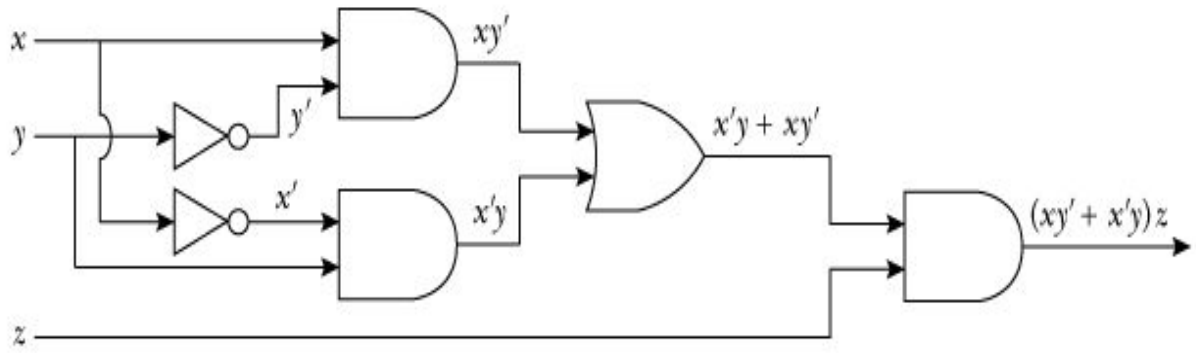
EX-NOR Gate

EXNOR gate means exclusive NOR gate and is a combination of **X-OR** and **NOT** gate. It also has two inputs and one output. When both the inputs are at 0, the output of the EXNOR gate will be high i.e.1. If one of the input is 0 and the other is 1 the output will be 0, and the gate is known as **Coincidence gate**.

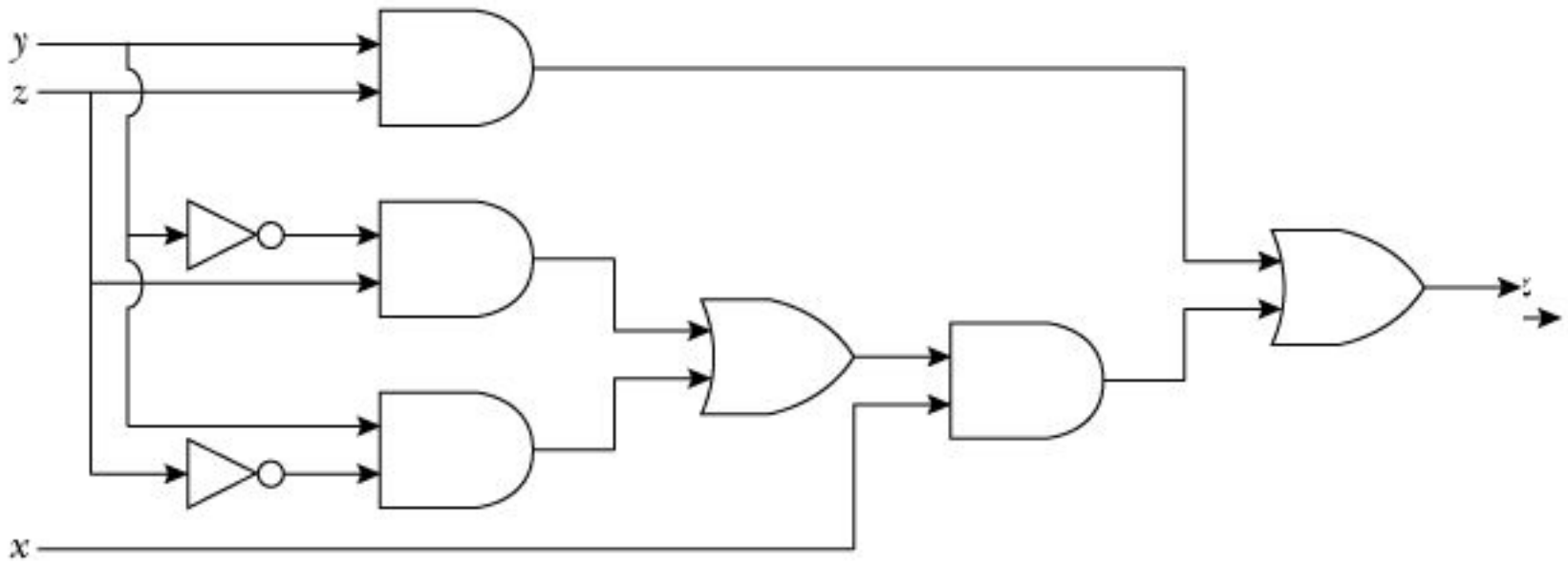
Symbol	Truth Table		
 2-input Ex-NOR Gate	B	A	Q
	0	0	1
	0	1	0
	1	0	0
	1	1	1
Boolean Expression $Q = \overline{A \oplus B}$	Read if A **AND** B the **SAME** gives Q		

- Draw a circuit diagram for $f = (xy' + x'y)z$.

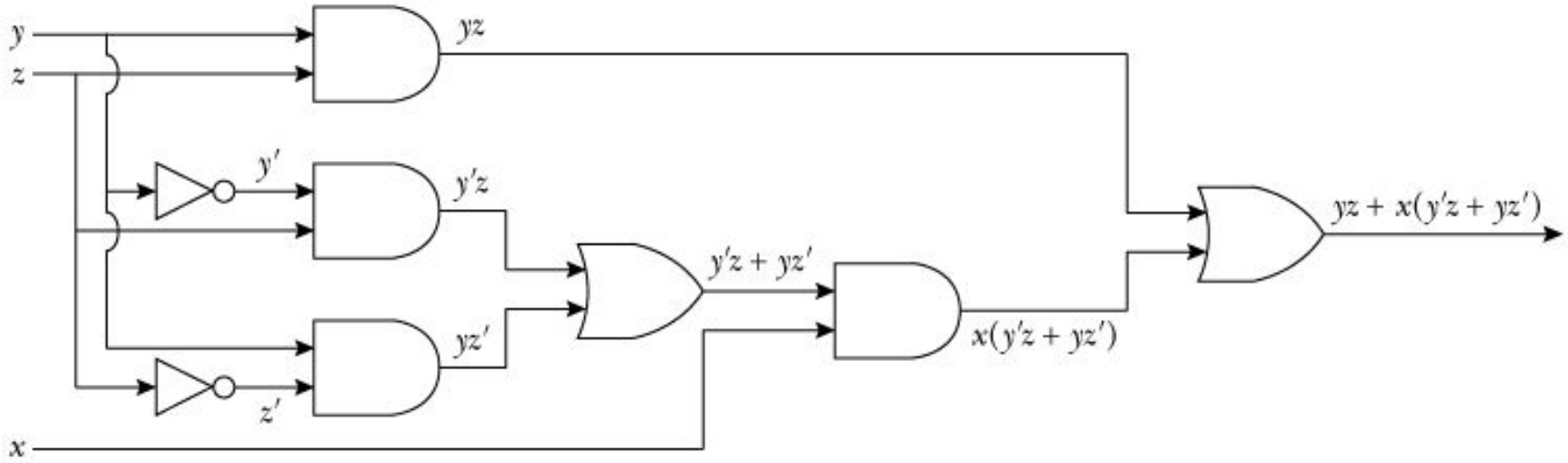
- Draw a circuit diagram for f= (xy' + x'y)z.



- Find the Boolean expression for the following circuit



- The Boolean expression that represent the circuit in the following Figure is yz+x(yz'+y'z)

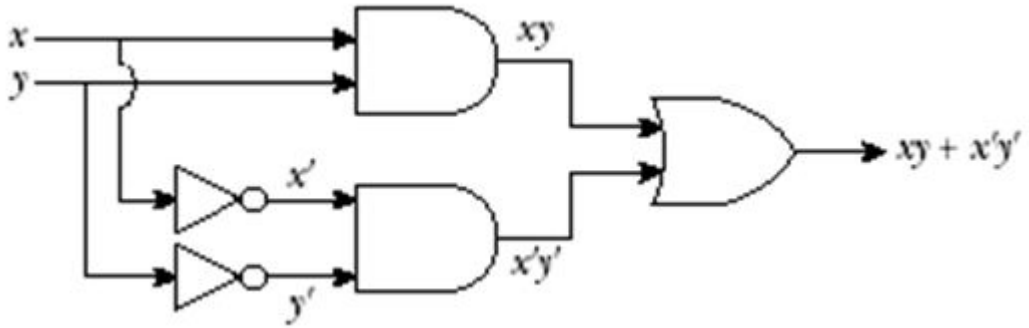


Example:

In this example, we construct a circuit that represents the following input-output table.

x	y	Output
1	1	1
1	0	0
0	1	0
0	0	1

It can be shown that the Boolean expression $xy + x'y'$ represents this input-output table. Hence, the circuit corresponding to this input-output table is as shown in



A circuit for two light switches

EXAMPLE

F(x,y)=1 when the light is on

- F(x,y)=0 when the light is off
- When both switches are closed, the light is on: F(1,1)=1, this implies
- When we open one switch, the light is off: F(1,0)=F(0,1)=0
- When the other switch is also open, the light is on: F(0,0)=1

Thus, we get:

x	y	F(x,y)
1	1	1
1	0	0
0	1	0
0	0	1

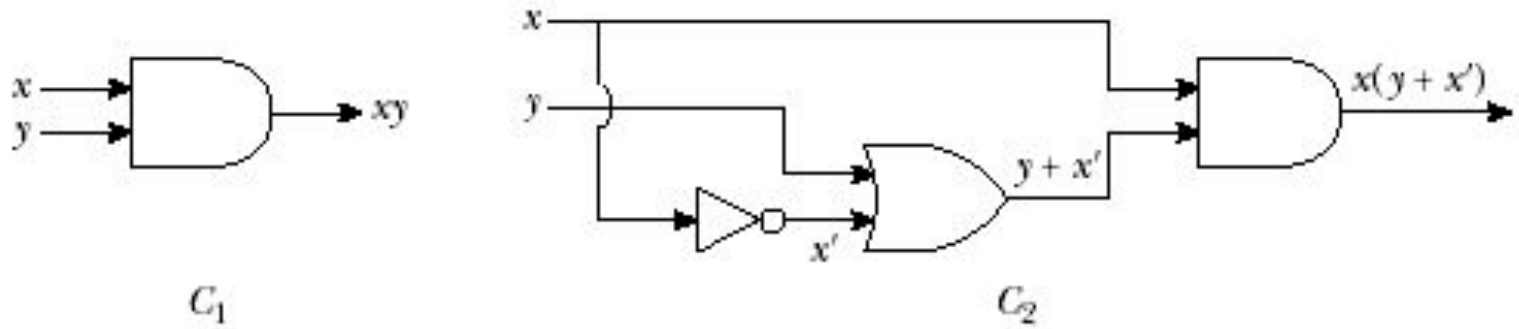
Which Boolean expression is given by F?

$F(x,y) = xy + x'y'$

Draw a circuit for F,
i.e., a circuit to control two light switches.

EXAMPLE:

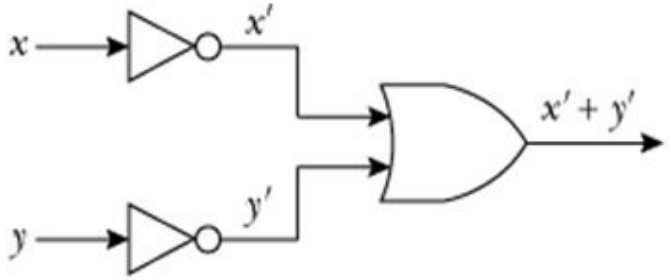
Consider the Boolean expressions $\alpha = xy$ and $\beta = x(y + x')$. Let C_1 denote the circuit for α and C_2 denote the circuit for β. These circuits are shown in Figure



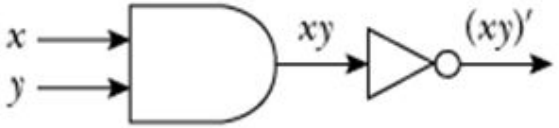
Note that $\beta = x(y + x') = xy + xx' = xy + 0 = xy$. Hence, the Boolean expressions α and β are equal, i.e., $\alpha = \beta$. However, note that circuit C_1 requires only one gate while circuit C_2 requires three gates. It follows that it is cheaper to construct C_1 than C_2. Moreover, we say that circuit C_1 is simpler than circuit C_2.

EXAMPLE:

Two combinatorial circuits, C_1 and C_2, having inputs $x_1, x_2, \ldots, x_n$ and a single output are said to be **equivalent** if whenever the circuits receive the same inputs, they give the same output.



C_1



C_2

The input-output table for circuits C_1 and C_2 is:

Input x	Input y	Output
1	1	0
1	0	1
0	1	1
0	0	1

Hence, these two circuits are equivalent.

Also, note that the Boolean expression for circuit C_1 is $x' + y'$ and the Boolean expression for circuit C_2 is $(xy)'$. From Boolean algebra we know that $(xy)' = x' + y'$.

Theorem 12.3.12: Two combinatorial circuits, C_1 and C_2, having inputs $x_1, x_2, \ldots, x_n$ and a single output are equivalent if and only if the Boolean expression representing C_1 is equal to the Boolean expression representing C_2.

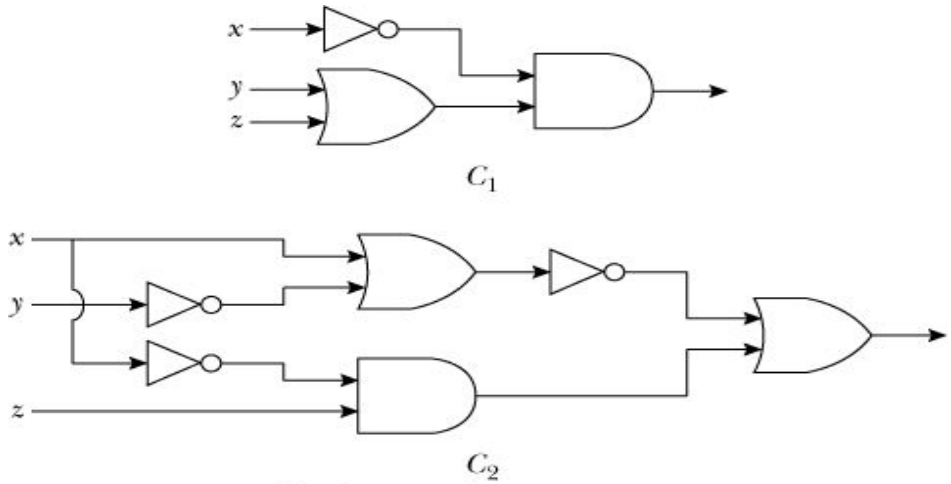


FIGURE 12.24 Circuits C_1 and C_2

The Boolean expression representing circuit C_1 is $x'(y+z)$, and the Boolean expression representing circuit C_2 is $(x+y')' + x'z$.

Now

$$(x+y')' + x'z = x'y + x'z \quad \text{because } (x+y')' = x'(y')' \text{ and } (y')' = y$$

$$= x'(y+z) \quad \text{by the distributive law}$$

Hence, the Boolean expression representing C_1 is equal to the Boolean expression representing C_2. Therefore, these two circuits are equivalent.

One circuit is said to be **simpler** than another circuit if the first circuit contains fewer gates than the second circuit.

In Figure 12.24, Example 12.3.13, we see that C_1 contains three gates, whereas C_2 contains six gates. Hence, the first circuit is simpler than the second circuit. Here we find that the effect of both circuits is the same but the first circuit is simpler than the second circuit.

Typically, we use the properties of Boolean algebra to simplify complex circuits used in digital computers and many other electronic devices.

The problem of finding a simpler circuit is called the **minimization problem**.

The symbol used for the Boolean expression $(xy)'$ is called a **NAND gate**.

The diagram in Figure 12.26 represents the NAND gate.

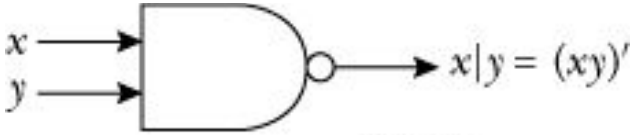


FIGURE 12.26 NAND gate

The input-output table for a NAND gate is as follows.

Input x	Input y	Output for NAND gate
1	1	0
1	0	1
0	1	1
0	0	1

In Boolean algebra, the expression $(xy)'$ is also denoted as $x \mid y$. The binary operation symbol $\mid$ used in $x \mid y$ is called the **Scheffer stroke**.

If we draw the circuit for the Boolean expression $(xy)'$ using NOT and AND gates, we obtain the circuits shown in Figure 12.27.

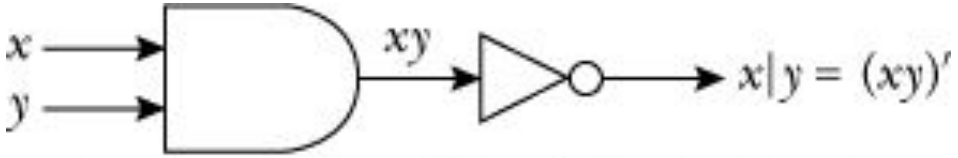


FIGURE 12.27 Circuit for $(xy)'$ using NOT and AND gates

As we can see, the circuit in Figure 12.27 requires two gates. However, we can draw the same circuit using only one NAND gate.

Let us again consider the Boolean expression $(xy)'$. Now $(xy)' = x' + y'$. The circuit diagram for $x' + y'$ is shown in Figure 12.28.

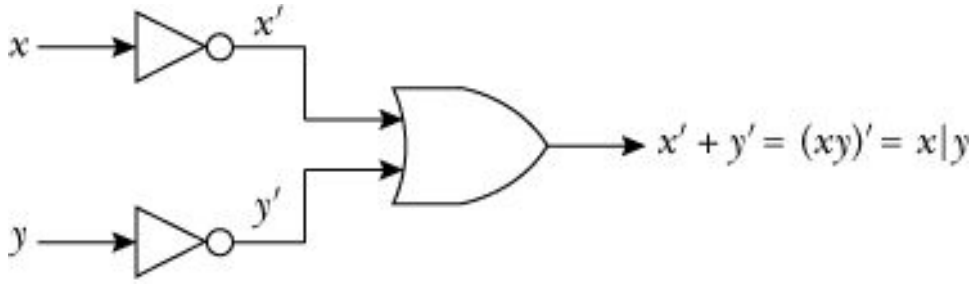


FIGURE 12.28 Circuit for $x' + y'$

Notice that this circuit contains three gates. We can replace this circuit by one NAND gate. Thus, we see that in a minimization problem the NAND gate plays a very useful role.

The symbol that is used for the Boolean expression $(x + y)'$ is called a **NOR gate**.

The diagram in Figure 12.29 represents the NOR gate.

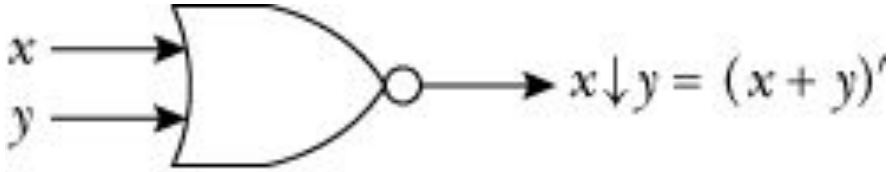


FIGURE 12.29 NOR gate

The input-output table for the NOR gate is as follows.

Input x	Input y	Output for NOR gate
1	1	0
1	0	0
0	1	0
0	0	1

In Boolean algebra, the expression $(x + y)'$ is also denoted by $x \downarrow y$. The binary operation symbol $\downarrow$ used in $x \downarrow y$ is called the **Peirce arrow**.

If we draw the circuit for the Boolean expression $(x + y)'$ using NOT and OR gates, we obtain the circuit shown in Figure 12.30.

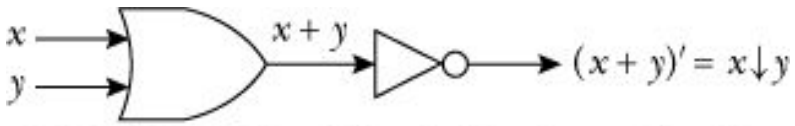


FIGURE 12.30 Circuits for $(x + y)'$ using NOT and OR gates

As we can see, the circuit in Figure 12.30 requires two gates. However, we can draw the same circuit using only one NOR gate. Let us again consider the Boolean expression $(x + y)'$. Now $(x + y)' = x'y'$. The circuit diagram for $x'y'$ is shown in Figure 12.31.

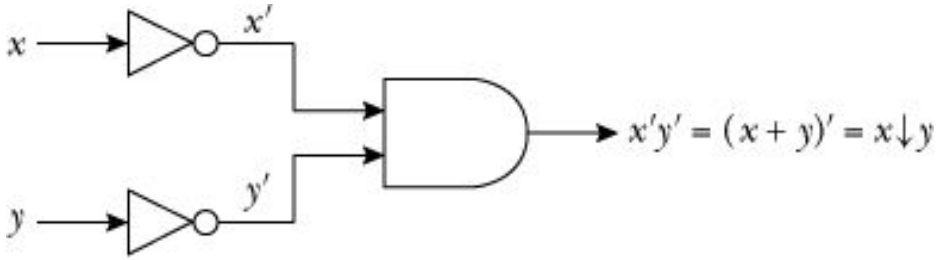
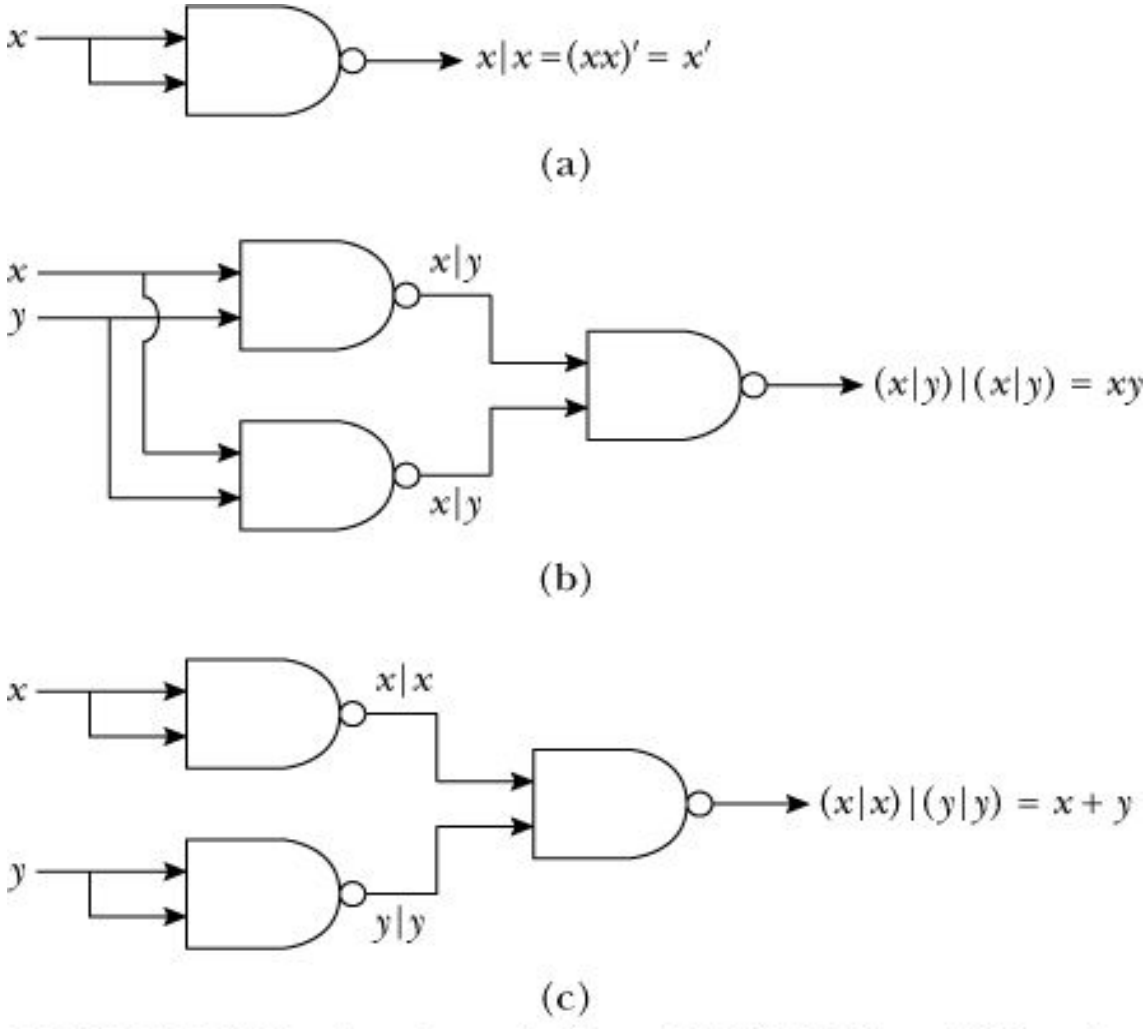


FIGURE 12.31 Circuits for $(x + y)' = x'y' = x \downarrow y$

Notice that this circuit contains three gates. We can replace this circuit by only one NOR gate. Thus, in this minimization problem the NOR gate also plays a useful role.

Logical Gates and Combinatorial Circuits



- A NOT gate can be implemented using a NAND gate (a).
- An AND gate can be implemented using NAND gates (b).
- An OR gate can be implemented using NAND gates (c).

Canonical and Standard Form : Minterms and Maxterms

- A binary variable may appear either in its normal form (x) or in its complement form (x).
- Now consider two binary variables x and y combined with an AND operation.
- Since each variable may appear in either form, there are four possible combinations: xy, x'y, xy', and x'y'.
- Each of these four AND terms is called a minterm, or a standard product.
- In a similar manner, n variables can be combined to form 2^n minterms.
- The 2^n different minterms may be determined by a method similar to the one shown in a Table (next slide) for three variables.

Canonical and Standard Form : Minterms and Maxterms

- The binary numbers from 0 to 2^n - 1 are listed under the n variables.
- Each minterm is obtained from an AND term of the n variables, with each variable being primed if the corresponding bit of the binary number is a 0 and unprimed if a 1.
- A symbol for each minterm is also shown in the table and is of the form mj , where the subscript j denotes the decimal equivalent of the binary number of the minterm designated.
- A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.

$$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$$

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

Canonical and Standard Form : Minterms and Maxterms

If we take the complement of f_1', we obtain the function f_1:

$$f_1 = (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z)$$
$$= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

Similarly, it is possible to read the expression for f_2 from the table:

$$f_2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)$$
$$= M_0 M_1 M_2 M_4$$

Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.

Canonical and Standard Form : Minterms and Maxterms

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Canonical and Standard Form : Minterms and Maxterms

Functions of Three Variables

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Sum of Minterms

- The minterms whose sum defines the Boolean function are those which give the 1's of the function in a truth table.
- Function can be either 1 or 0 for each minterm, and since there are 2^n minterms
- Convenient to express a Boolean function in its sum-of-minterms form.
- If the function is not in this form, it can be made so by first expanding the expression into a sum of AND terms.
- Each term is then inspected to see if it contains all the variables.
- If it misses one or more variables, it is ANDed with an expression such as x + x, where x is one of the missing variables.
- The next example clarifies this procedure.

Example

Express the Boolean function $F = A + B'C$ as a sum of minterms. The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

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

This function is still missing one variable, so

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

The second term $B'C$ is missing one variable; hence,

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

Combining all terms, we have

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

But $AB'C$ appears twice, and according to theorem 1 ($x + x = x$), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned} F &= A'B'C + AB'C + AB'C + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

It is sometimes convenient to express the function in the following brief notation.

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

Alternative procedure for deriving the minterms

Consider the Boolean function given in previous Example

$$F = A + B'C$$

The truth table can be derived directly from the algebraic expression by listing the eight binary combinations under variables A, B, and C and inserting 1's under F for those combinations for which A = 1 and BC = 01.

From the truth table, we can then read the five minterms of the function to be 1, 4, 5, 6, and 7.

Truth Table for $F = A + B'C$

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

Alternative procedure for deriving the minterms

Consider the Boolean function given in previous Example

$$F = A + B'C$$

The truth table can be derived directly from the algebraic expression by listing the eight binary combinations under variables A, B, and C and inserting 1's under F for those combinations for which A = 1 and BC = 01.

From the truth table, we can then read the five minterms of the function to be 1, 4, 5, 6, and 7.

Truth Table for $F = A + B'C$

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

Alternative procedure for deriving the minterms

Consider the Boolean function given in previous Example

$$F = A + B'C$$

The truth table can be derived directly from the algebraic expression by listing the eight binary combinations under variables A, B, and C and inserting 1's under F for those combinations for which A = 1 and BC = 01.

From the truth table, we can then read the five minterms of the function to be 1, 4, 5, 6, and 7.

Truth Table for F = A + B'C

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

Example of Maxterms

Express the Boolean function $F = xy + x'z$ as a product of maxterms. First, convert the function into OR terms by using the distributive law:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &= (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

$$\begin{aligned} x' + y &= x' + y + zz' = (x' + y + z)(x' + y + z') \\ x + z &= x + z + yy' = (x + y + z)(x + y' + z) \\ y + z &= y + z + xx' = (x + y + z)(x' + y + z) \end{aligned}$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z') \\ &= M_0 M_2 M_4 M_5 \end{aligned}$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

The product symbol, Π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

Conversion between Canonical Forms

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms which make the function equal to 1, whereas its complement is a 1 for those minterms for which the function is a 0. As an example, consider the function

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

This function has a complement that can be expressed as

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Now, if we take the complement of F' by DeMorgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3 = \Pi(0, 2, 3)$$

The last conversion follows from the definition of minterms and maxterms as shown in Table 2.3. From the table, it is clear that the following relation holds:

$$m_j' = M_j$$

Conversion between Canonical Forms

- We now state a general conversion procedure:
- To convert from one canonical form to another, interchange the symbols Σ and Π and list those numbers missing from the original form.
- In order to find the missing terms, one must realize that the total number of minterms or maxterms is 2^n , where n is the number of binary variables in the function.
- A Boolean function can be converted from an algebraic expression to a product of maxterms by means of a truth table and the canonical conversion procedure.
- Consider, for example, the Boolean expression F = xy + x'z
- First, we derive the truth table of the function
- The 1's under F in the table are determined from the combination of the variables for which xy = 11 or xz = 01.
- The minterms of the function are read from the truth table to be 1, 3, 6, and 7.
- The function expressed as a sum of minterms is $F(x, y, z) = \Sigma(1, 3, 6, 7)$

Conversion between Canonical Forms

Since there is a total of eight minterms or maxterms in a function of three variables, we determine the missing terms to be 0, 2, 4, and 5. The function expressed as a product of maxterms is

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

the same answer as obtained in Previous Example.

Conversion between Canonical Forms

Truth Table for $F = xy + x'z$

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

Minterms

Maxterms

Canonical Form: Sum of Minterms

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

$= X'Y'(Z+Z')+Z(X+X')(Y+Y')$

$= X'Y'Z+X'Y'Z'+(ZX+ZX')(Y+Y')$

$= X'Y'Z'+X'Y'Z+X'YZ+XY'Z+XYZ$

$= m0+m1+m3+m5+m7$

$= \sum(m0+m1+m3+m5+m7)$

$= \sum m(0,1,3.5,7)$

Product of minterms 0,1,3.5.7

Canonical Form: Product of Maxterms

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

$= Z+X'Y'$

$= (Z+X')(Z+Y')$ Using Distributed law

$=(X'+YY'+Z)(XX'+Y'+Z)$

$=(X'+Y+Z)(X'+Y'+Z)\ (X+Y'+Z)(X'+Y+Z)$

$=(X'+Y+Z)(X'+Y'+Z)\ (X+Y'+Z)$

$=M_4M_6M_2$

$=\prod(M_4M_6M_2)$

$=\prod M\ (2,4,6)$

Product of Maxterm 2, 4, 6

Canonical Forms

- **Example in slide 3 and** 4 $\sum m(0,1,3.5,7) = \prod M\ (2,4,6)$

In general $\sum m(\{x\}) = \prod M\ (\{y\})$

Where $\{x\} \cup \{y\} = \{0,1,2, \ldots\ 2^{n-1}\}$ and

$\{x\} \cap \{y\} = \{\emptyset\}$

- A function expressed as a sum of minterms can also be expressed as the product of the opposite maxterms

Canonical Forms: Minterms and Maxterms

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The coefficients of the desired binary number are obtained from the remainders as follows:

Canonical/Standard Forms

- There are two types of standard forms:
  - Sum of products (SOP) and
  - Products of sums (POS).

Sum of Product (SOP)

Truth Table for $F = xy + x'z$

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

SOP form

POS form

Directly written from Truth Table

Each minterm is having all the variable or complemented form

Canonical/ Standard form

F=x'yz'+xy'z'+xy'z+xyz'+xyz

= x'yz'+xy'[z'+z]+xy[z'+z]

=x'yz'+xy'+xy

=x'yz'+x[y'+y]

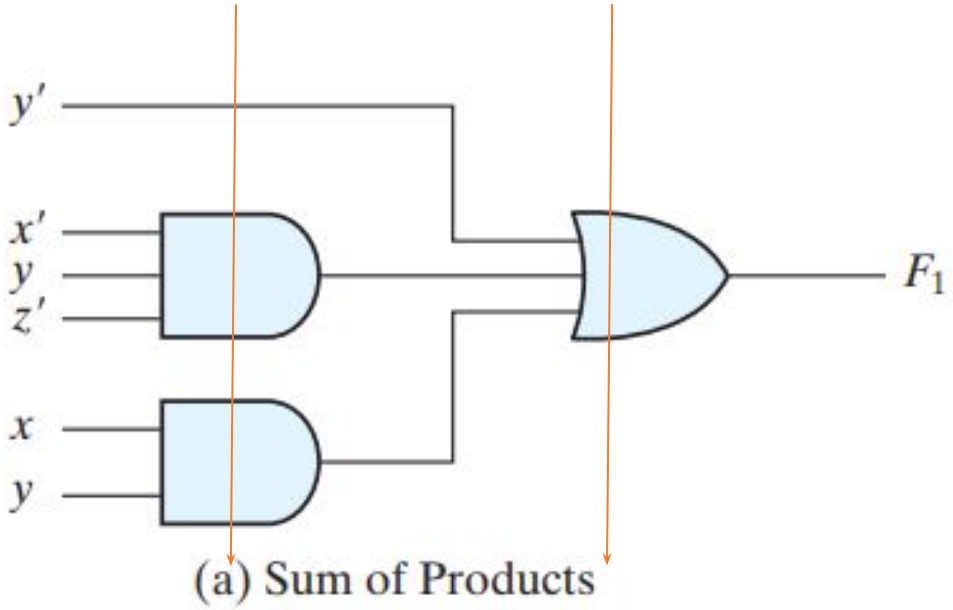
= x'yz'+x

= x+yz' [Distributed law A+A'B=A+B)]

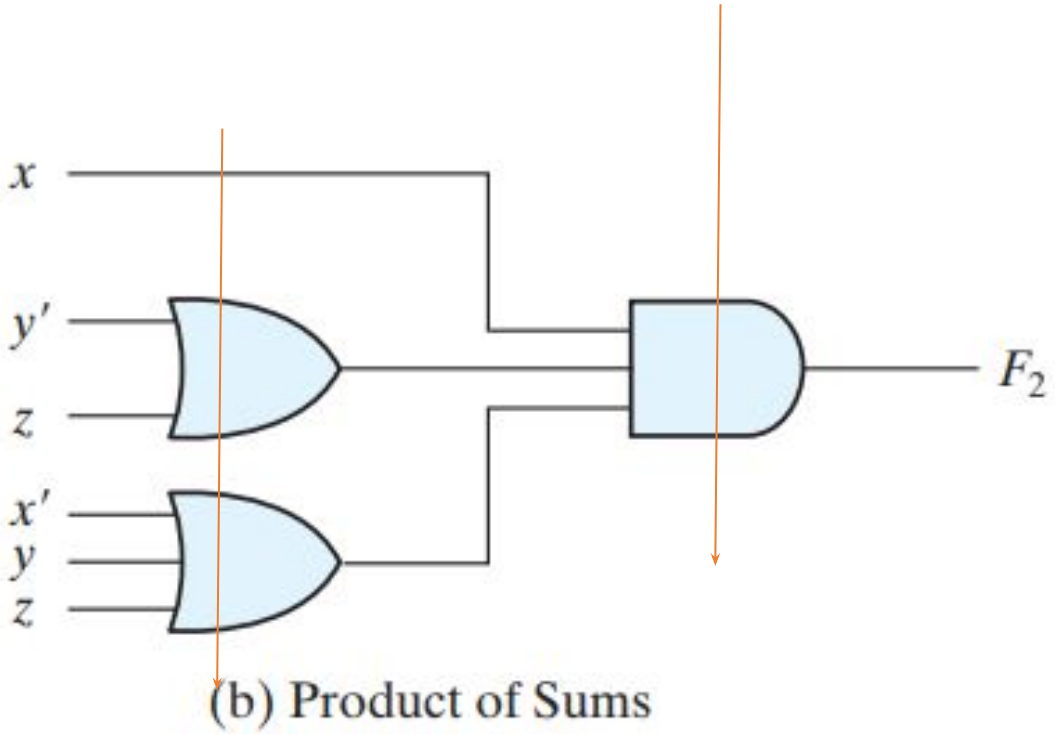
Minimal SOP form

Each minterm does not have all the variables in normal or complemented form

Standard Forms: Two-level implementation



(a) Sum of Products

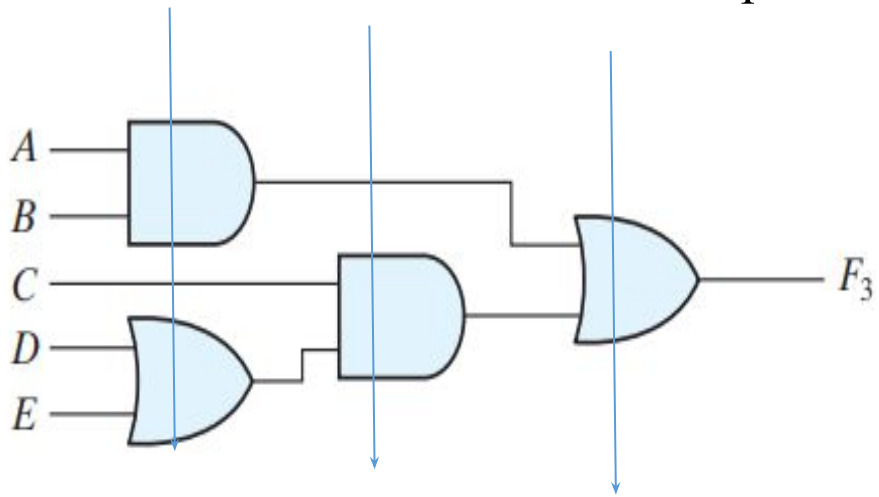


(b) Product of Sums

y'+x'yz'+xy

Standard Forms: Conversion from other form

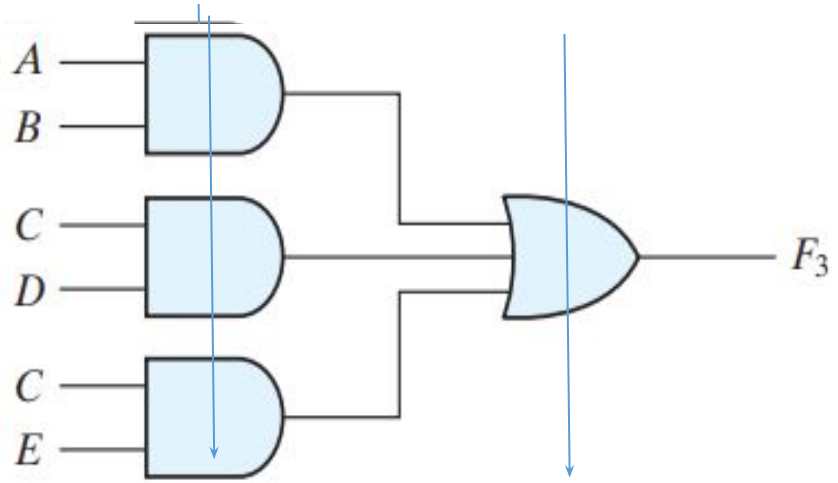
- A Boolean function may be expressed in a nonstandard form
- Example: $F_3 = AB + C(D + E)$
  - ✔ Neither in sum-of-products nor in product-of-sums form.



(a) $AB + C(D + E)$

3-> 2 level

Sum of Product



(b) $AB + CD + CE$

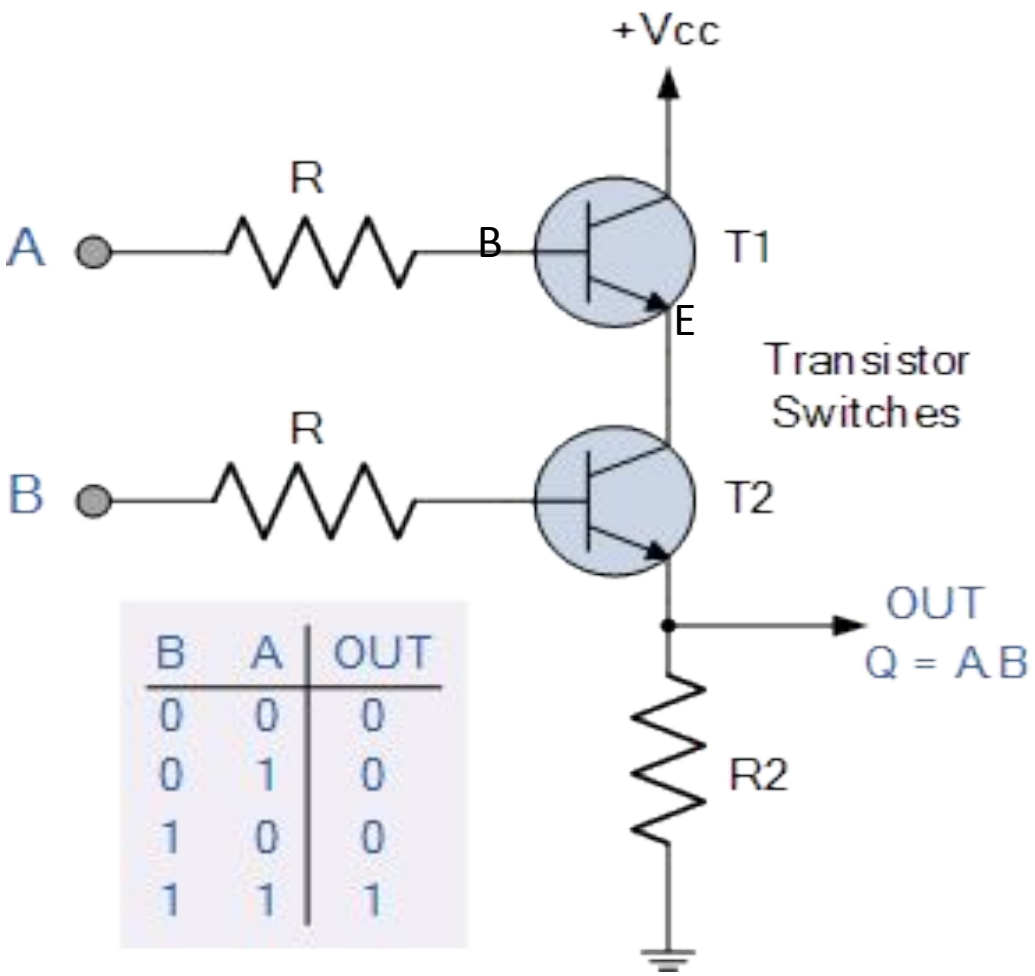
- ✔ Two-level produces the least amount of delay through the gates when the signal propagates from the inputs to the output.

Constructing Gates

Transistor

- A device that acts either as a wire that conducts electricity or as a resistor that blocks the flow of electricity, depending on the voltage level of an input signal
- A transistor has no moving parts, yet acts like a switch
- It is made of a semiconductor material, which is neither a particularly good conductor of electricity nor a particularly good insulator

How is the behavior implemented



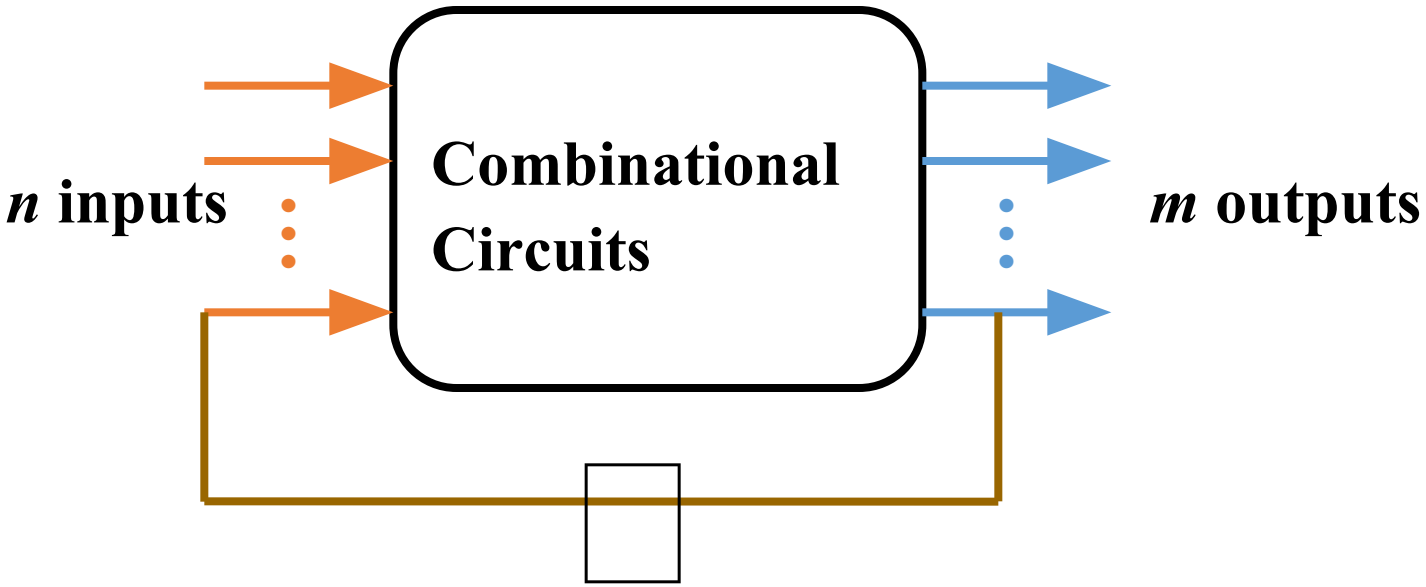
B	A	OUT
0	0	0
0	1	0
1	0	0
1	1	1

Circuits

- **Combinational Circuit**
  - The input values explicitly determine the output
- **Sequential Circuit**
  - The output is a function of the input values and the existing state of the circuit

Combinational Circuits

- Output is function of input only
  - i.e. no feedback



When input changes, output may change (after a delay)

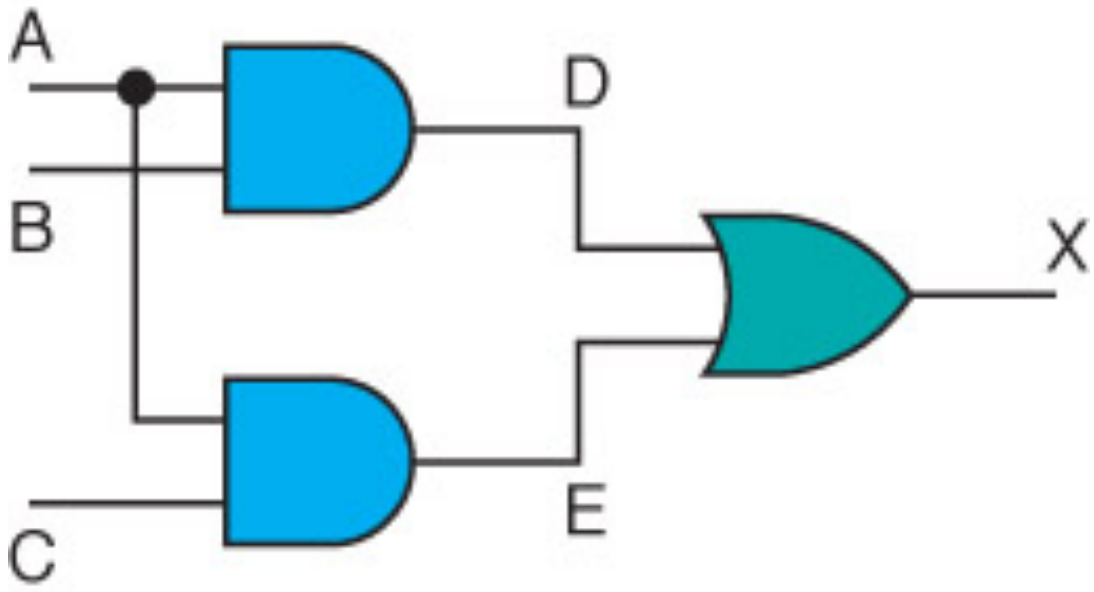
Combinational Circuits

Example:

- Adder
- Subtractor
- Decoder
- MUX
- DeMUX
- Code Converters
- Comparator
- ROM
- PLA
- PAL

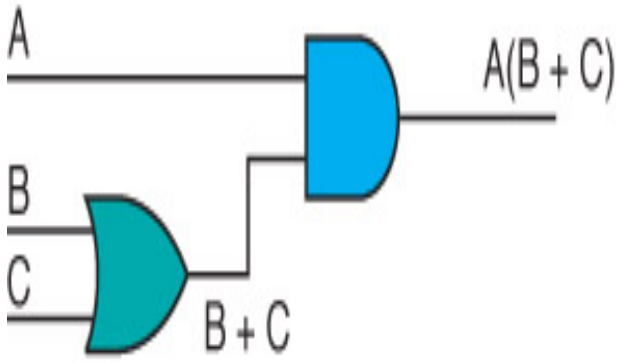
Combinational Circuits

- Gates are combined into circuits by using the output of one gate as the input for another



Combinational Circuits

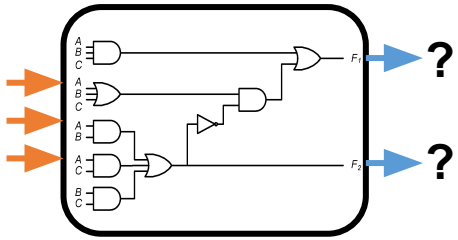
❖ Consider the following Boolean expression A(B + C)



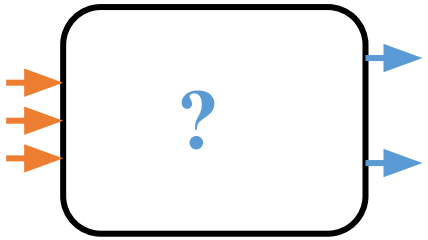
A	B	C	B + C	A(B + C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

Combinational Circuits

- Analysis
  - Given a circuit, find out its *function*
  - Function may be expressed as:
    - Boolean function
    - Truth table

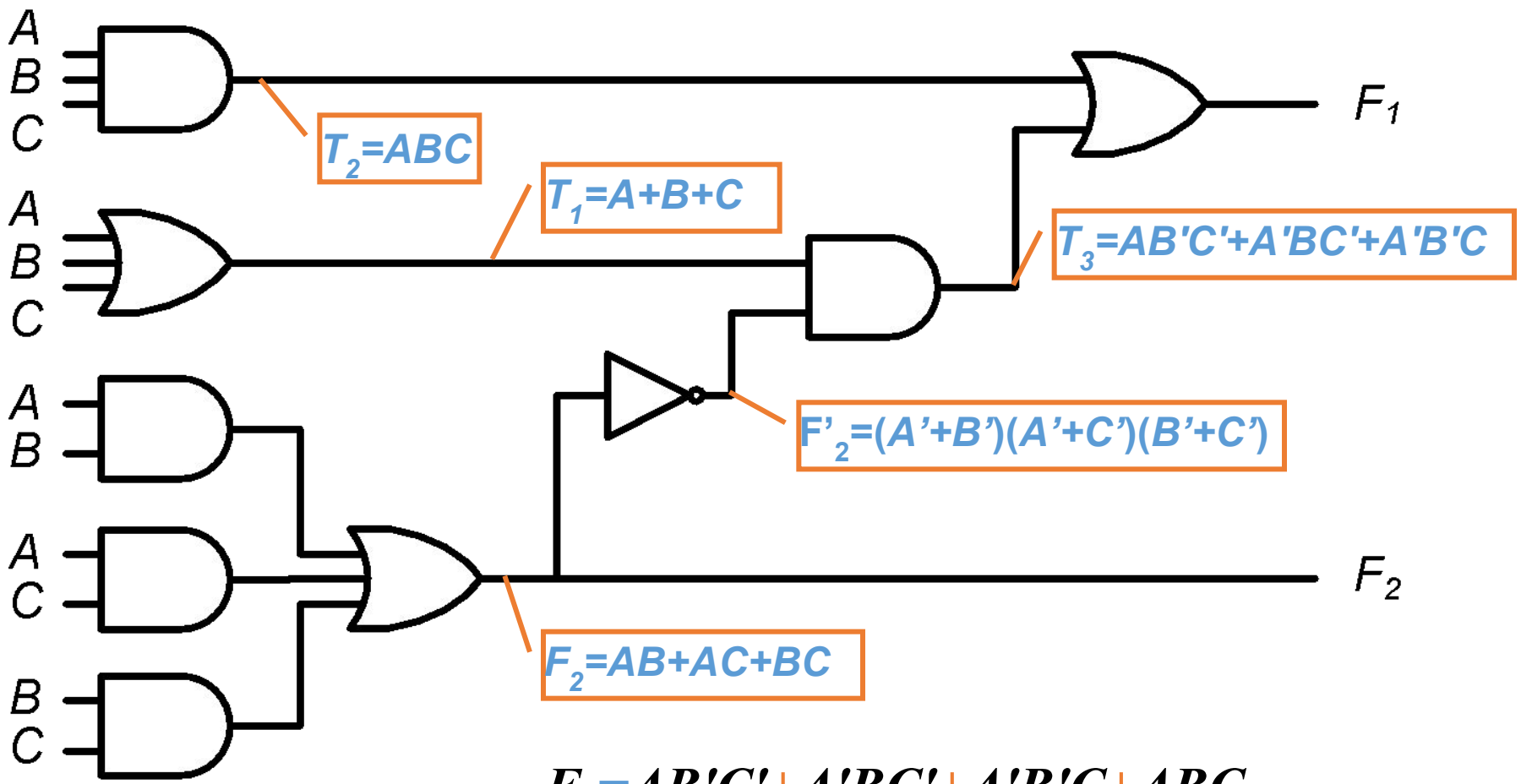


- Design
  - Given a desired function, determine its *circuit*
  - Function may be expressed as:
    - Boolean function
    - Truth table



Analysis Procedure

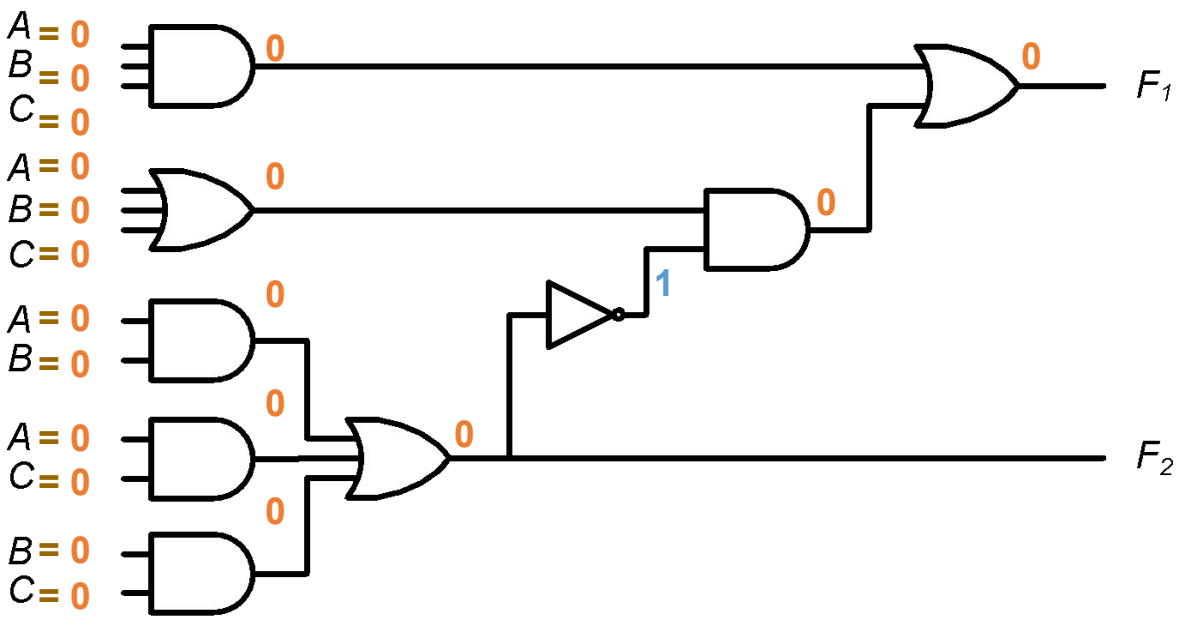
- Boolean Expression Approach



$$F_1 = AB'C' + A'BC' + A'B'C + ABC$$
$$F_2 = AB + AC + BC$$

Analysis Procedure

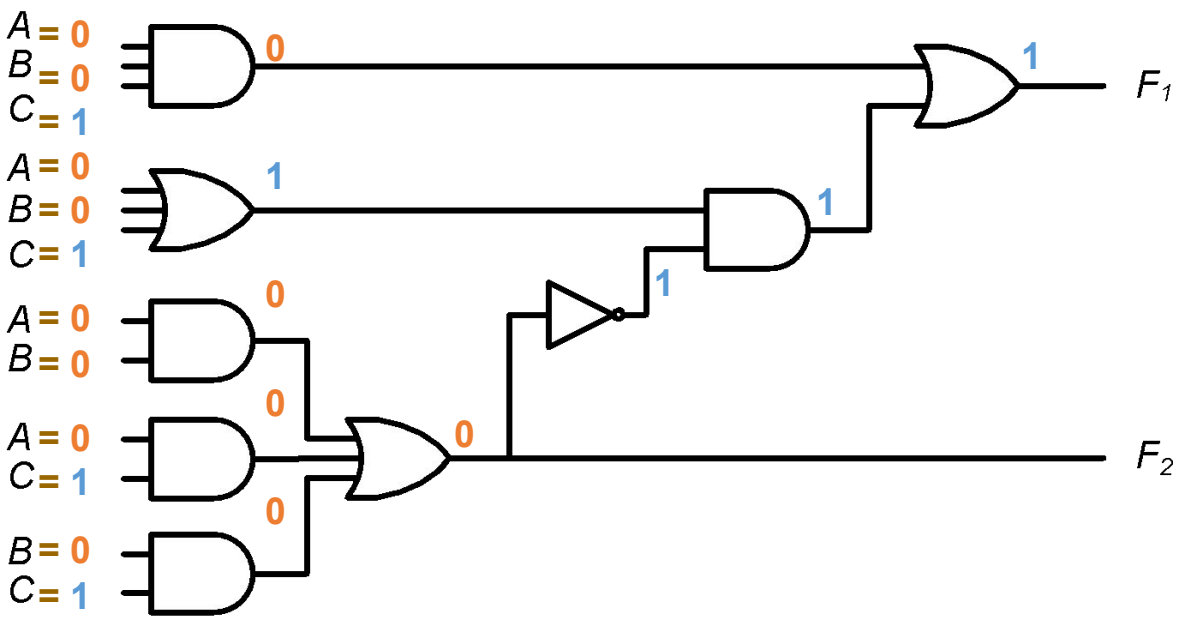
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0

Analysis Procedure

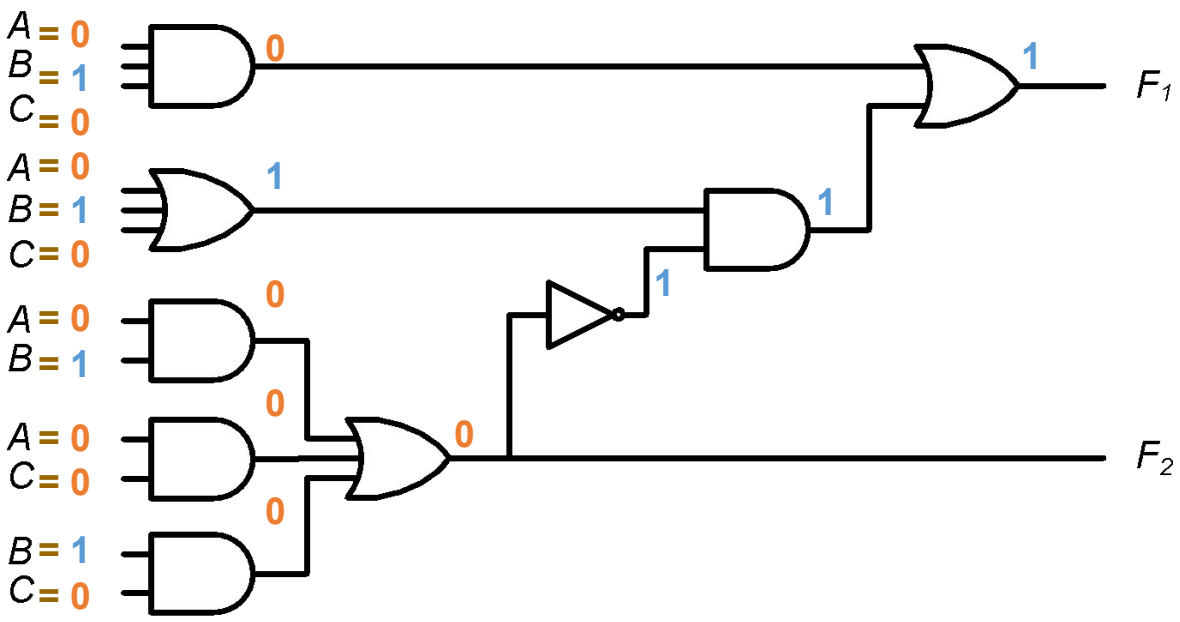
- Truth Table Approach



A B C	F_1	F_2
0 0 0	0	0
0 0 1	1	0

Analysis Procedure

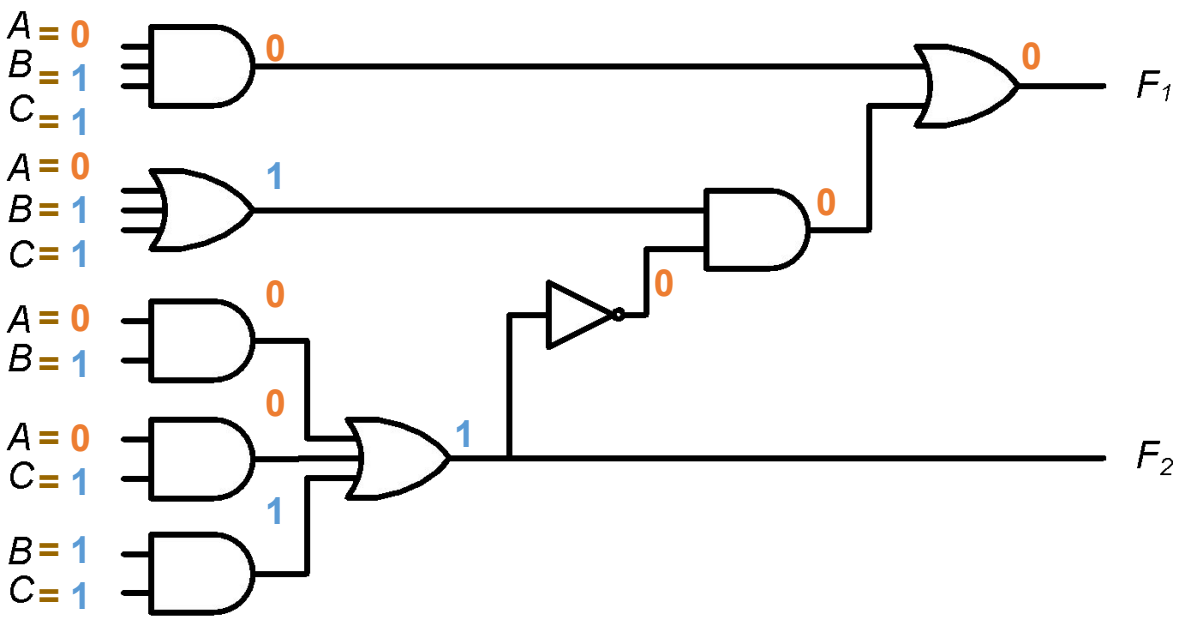
- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0

Analysis Procedure

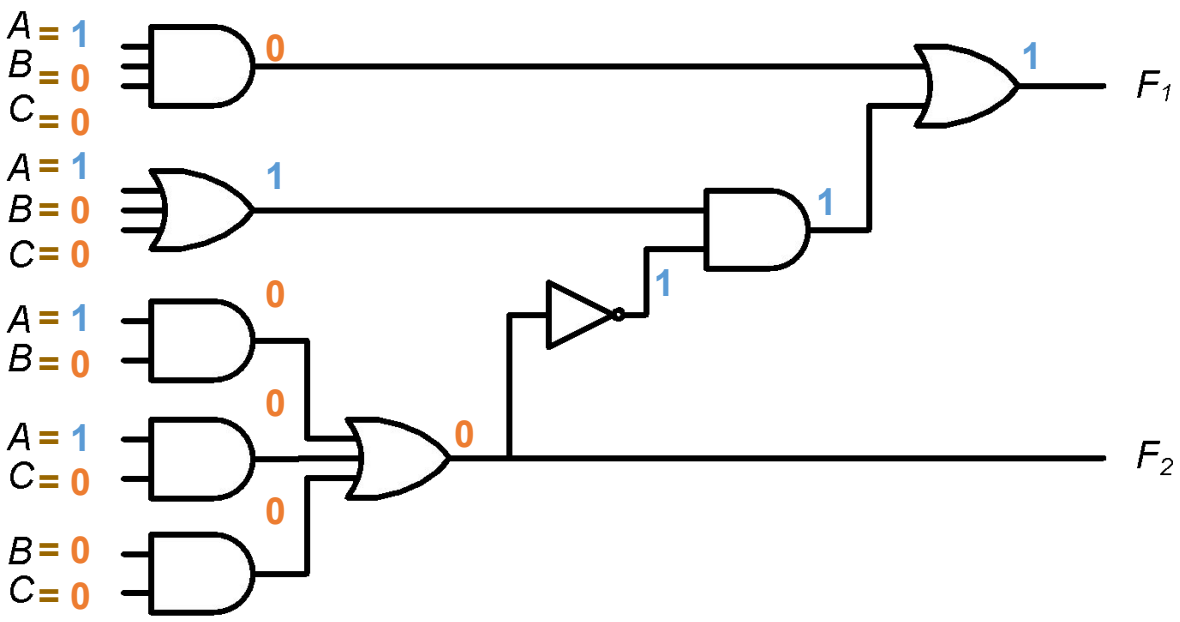
- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1

Analysis Procedure

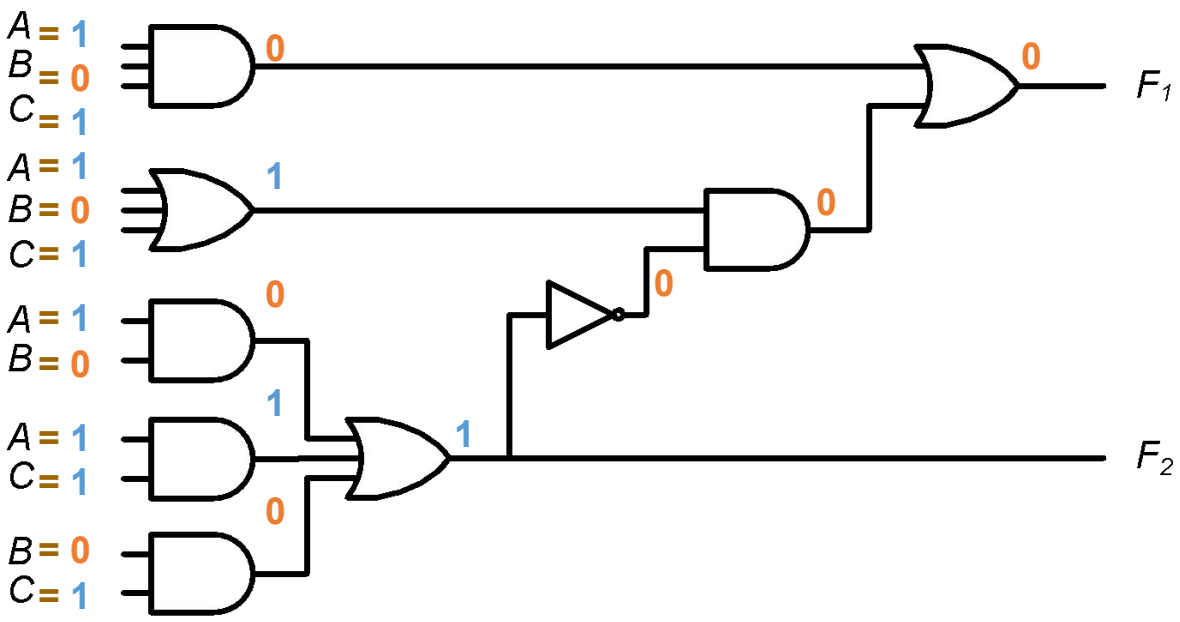
- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0

Analysis Procedure

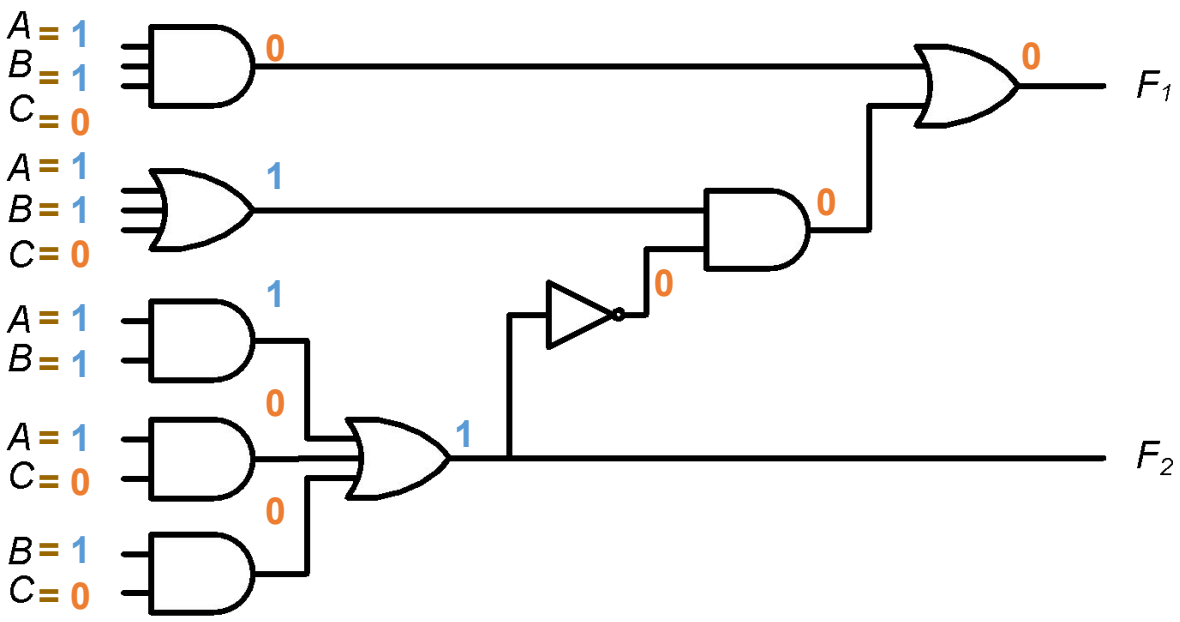
- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1

Analysis Procedure

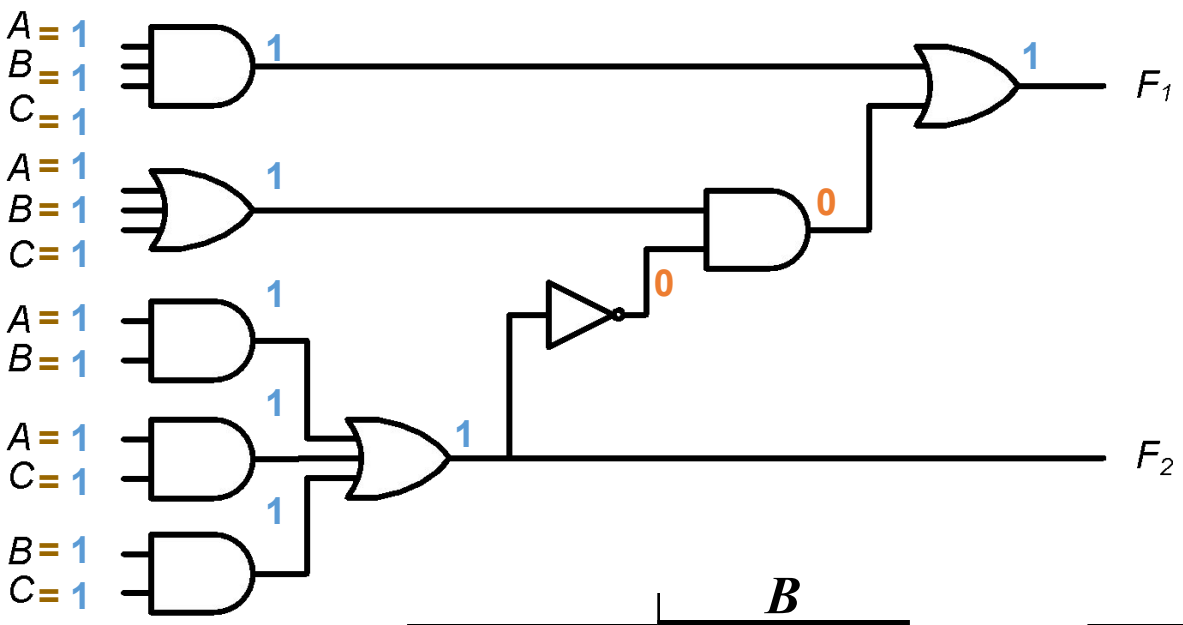
- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1

Analysis Procedure

- Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A \ B,C	00	01	11	10
0	0	1	0	1
1	1	0	1	0

A \ B,C	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Minimization using K-Map Discuss later ....

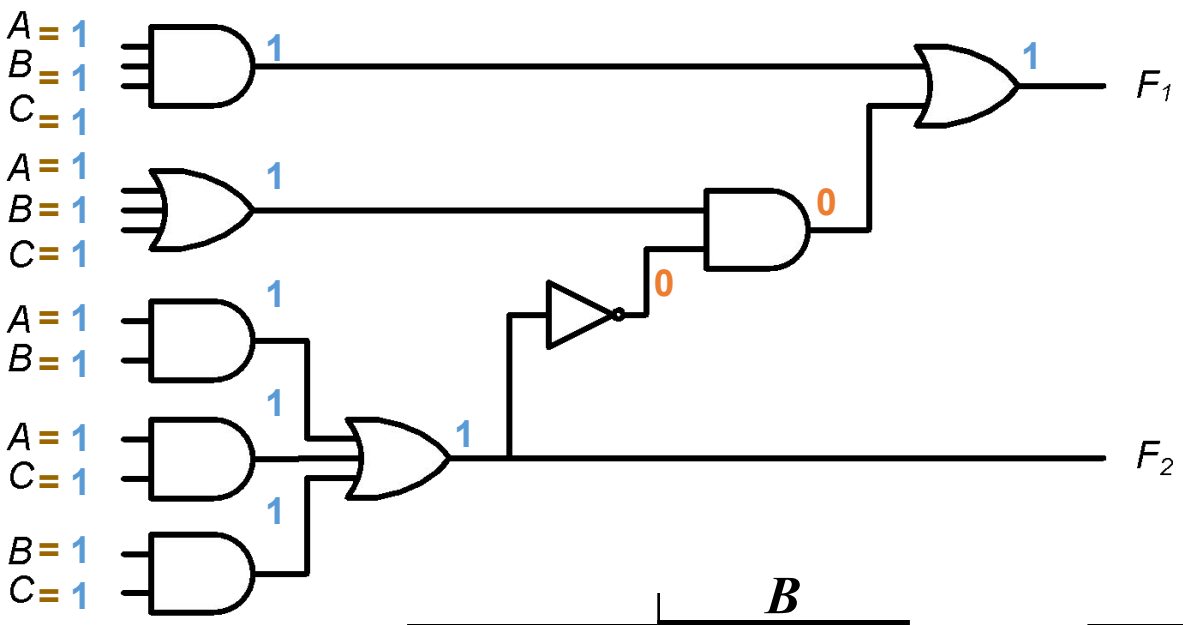
$$F_1 = AB'C' + A'BC' + A'B'C + ABC \qquad F_2 = AB + AC + BC$$

Design Procedure

- Given a problem statement:
  - Determine the number of *inputs* and *outputs*
  - Derive the truth table
  - Simplify the Boolean expression for each output
  - Produce the required circuit

Analysis Procedure

Truth Table Approach



A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

A \ B,C	00	01	11	10
0	0	1	0	1
1	1	0	1	0

A \ B,C	00	01	11	10
0	0	0	1	0
1	0	1	1	1

Minimization using K-Map

$F_1 = AB'C' + A'BC' + A'B'C + ABC$

$F_2 = AB + AC + BC$

Gate-Level Minimization

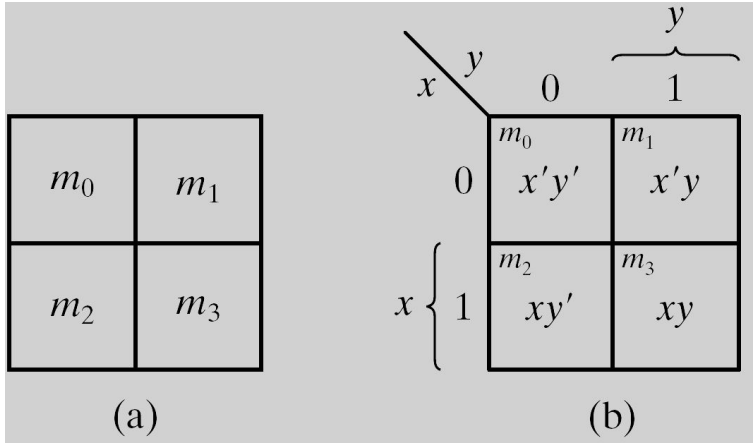
- Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit.
- This task is well understood, but is difficult to execute by manual methods when the logic has more than a few inputs

The Map Method

- ❖ The complexity of the digital logic gates
  - The complexity of the algebraic expression
- ❖ Logic minimization
  - Algebraic approaches: lack specific rules
  - The Karnaugh map
    - ✔ A simple straight forward procedure
    - ✔ A pictorial form of a truth table
    - ✔ Applicable if the # of variables < 7
- ❖ A diagram made up of squares
  - Each square represents one minterm

The Map Method

- Boolean expressions may be simplified by algebraic means.
- It is also known as the Karnaugh map or K-map .
  - A simple straight forward procedure
  - A pictorial form of a truth table
  - Applicable if the # of variables < 7
- A diagram made up of squares
  - Each square represents one minterm/maxterms



- Boolean function can be expressed as a sum of minterms.
- The simplified expressions produced by the map are two standard forms:
  - Sum of products (SoP)
  - Product of sums (PoS)
- The expression produces a circuit diagram with
  - a minimum number of gates and
  - the minimum number of inputs to each gate.

Two-Variable Map

- A two-variable map (Fig. 1)
  - Four minterms
  - Four squares, one for each minterm.
  - $x' =$ row 0; $x =$ row 1
  - $y' =$ column 0; $y =$ column 1
- The map is redrawn in Fig.1(b):
  - To show the relationship between the squares and the variables x and y .
- The 0 and 1 marked in each row and column designate the values of variables.

Fig. 1 Two-variable Map

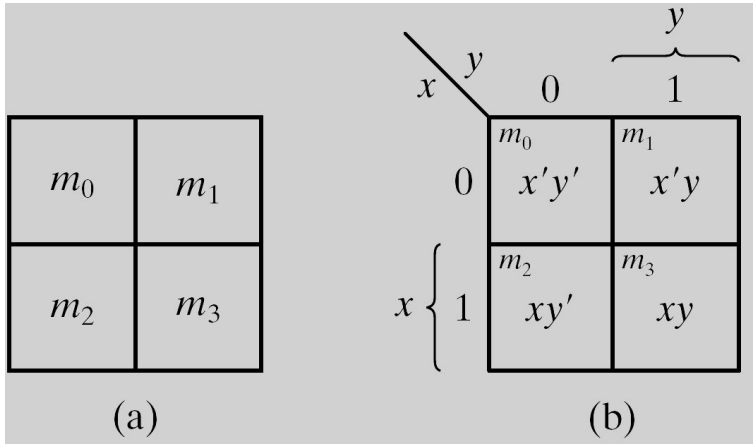
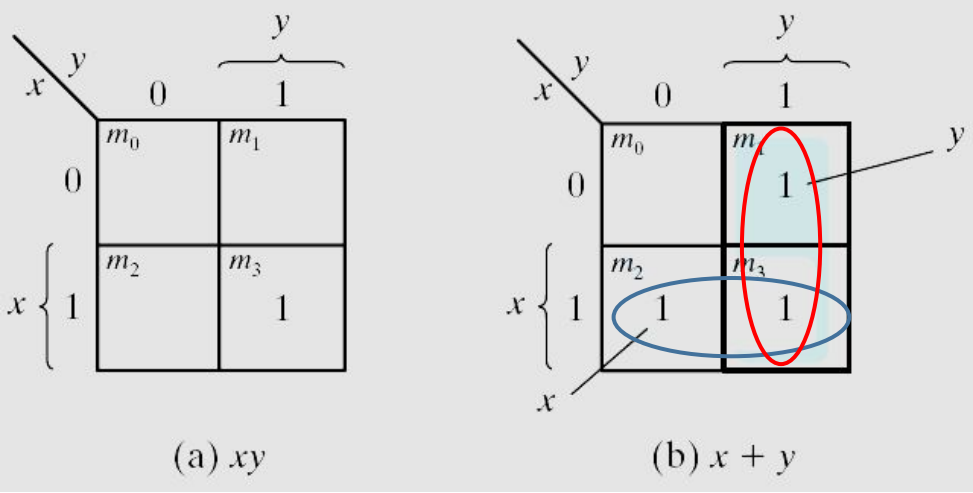


Fig. 2 Representation of functions in the map



A truth table in square diagram

Fig. 2(a): $xy = m_3$

Fig. 2(b): $x+y = x'y+xy' +xy= m_1+m_2+m_3$

Three-variable Map

❖ A three-variable map
- Eight minterms
- The Gray code sequence
- Any two adjacent squares in the map differ by only on variable
  - Primed in one square and unprimed in the other
  - e.g., m_5 and m_7 can be simplified

x'yz+x'yz'=x'y(z+z')=x'y.1=x'y

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

$x \backslash yz$	00	01	11	10
0	m_0 $x'y'z'$	m_1 $x'y'z$	m_3 $x'yz$	m_2 $x'yz'$
1	m_4 $xy'z'$	m_5 $xy'z$	m_7 xyz	m_6 xyz'

Three-variable Map

Three-variable Map

- m_0 and m_2 (m_4 and m_6) are adjacent
- $m_0 + m_2 = x'y'z' + x'yz' = x'z'(y'+y) = x'z'$
- $m_4 + m_6 = xy'z' + xyz' = xz'(y'+y) = xz'$

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

x \ xz	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
x 1	$xy'z'$	$xy'z$	xyz	xyz'

y

z

Three-variable Map

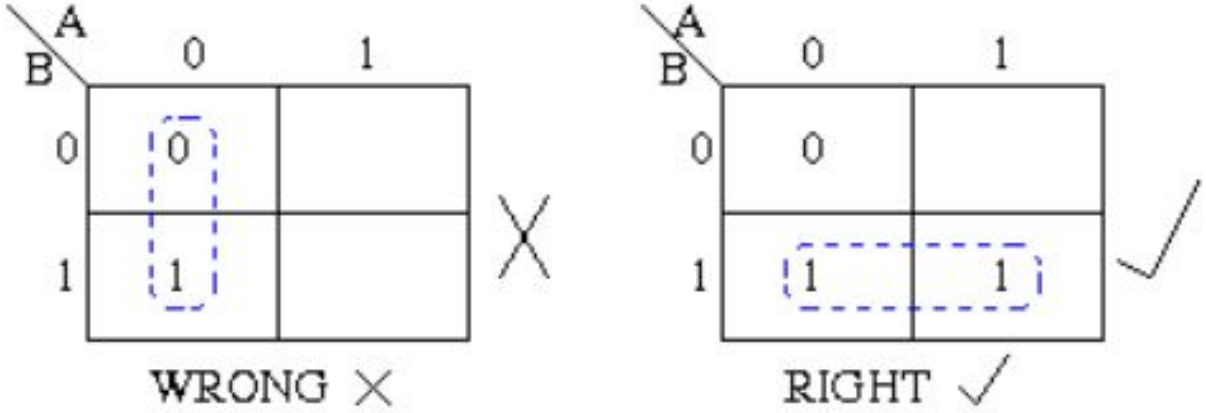
Karnaugh Maps - Rules of Simplification

❑ Summary:

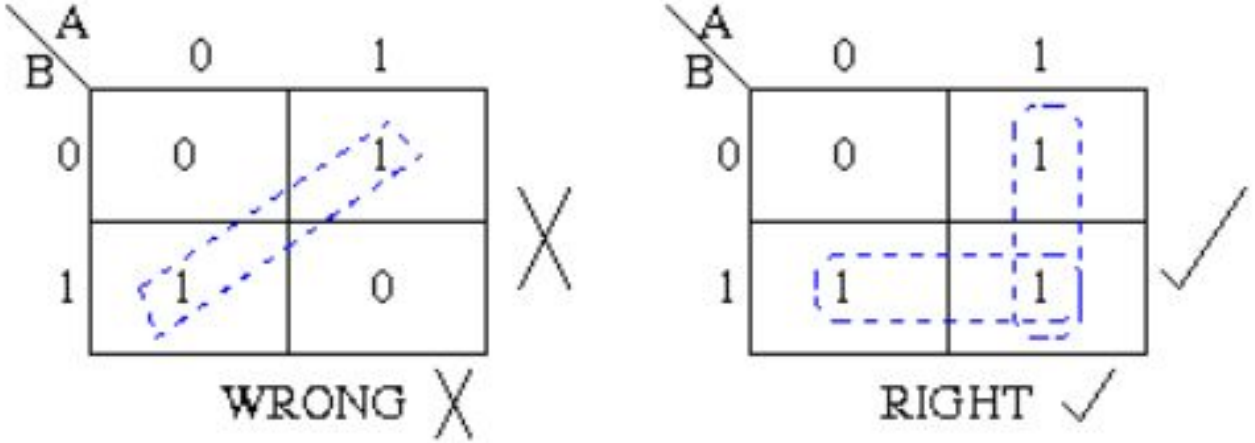
- ❖ No zeros allowed.
- ❖ No diagonals.
- ❖ Only power of 2 number of cells in each group.
- ❖ Groups should be as large as possible.
- ❖ Every one must be in at least one group.
- ❖ Overlapping allowed.
- ❖ Wrap around allowed.
- ❖ Fewest number of groups possible.

Karnaugh Maps - Rules of Simplification

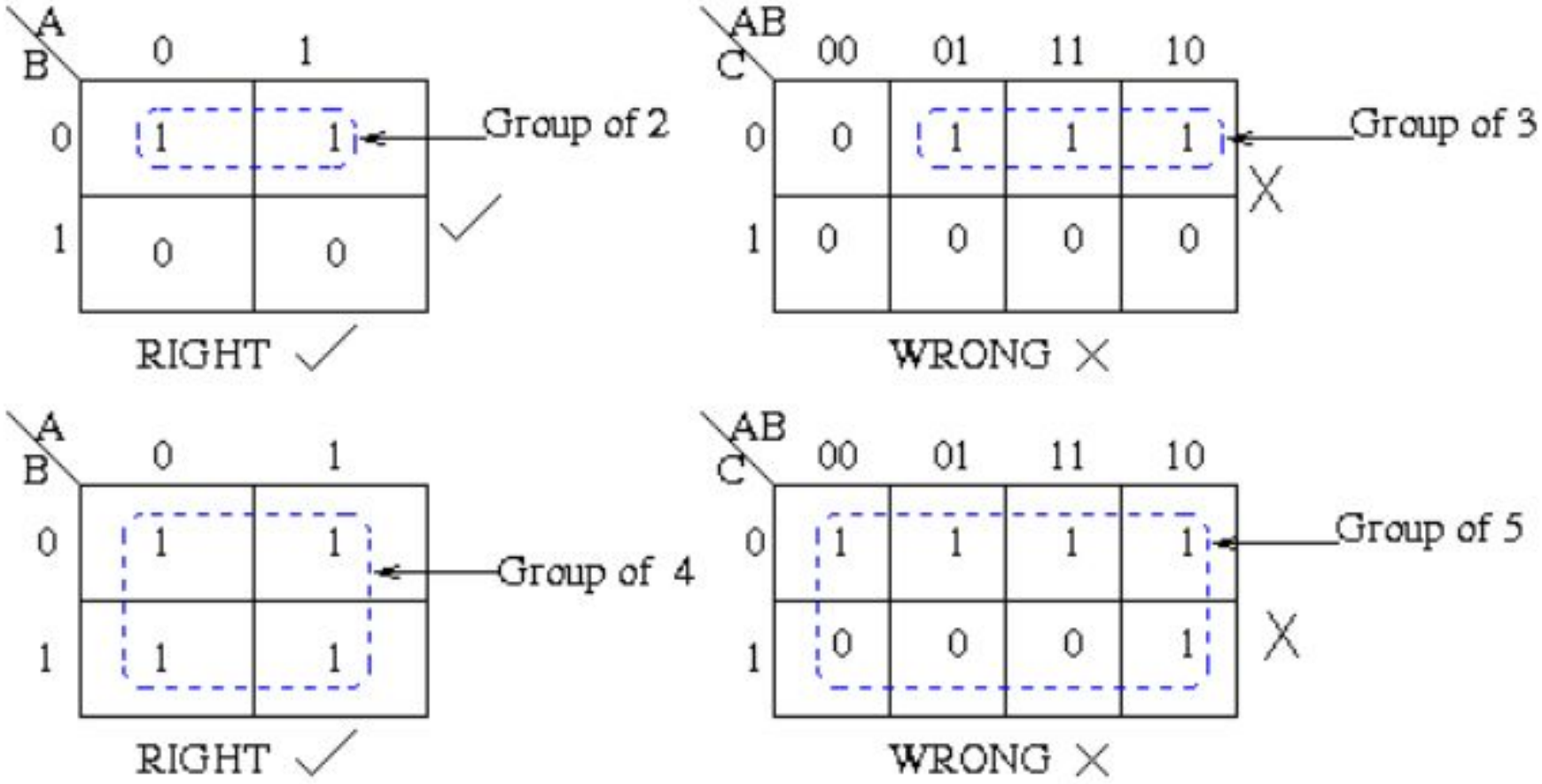
- **Groups may not include any cell containing a zero**



- **Groups may be horizontal or vertical, but not diagonal.**

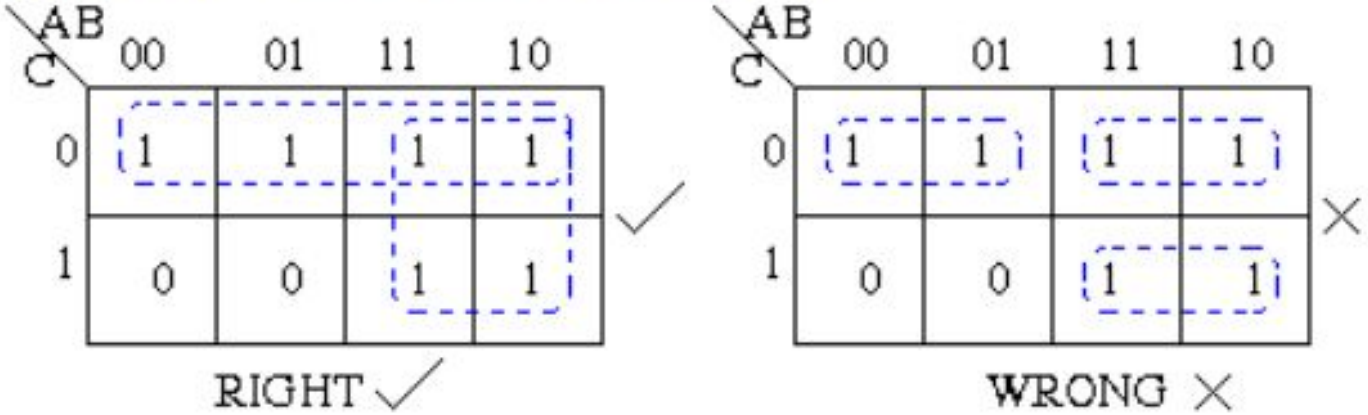


Karnaugh Maps - Rules of Simplification

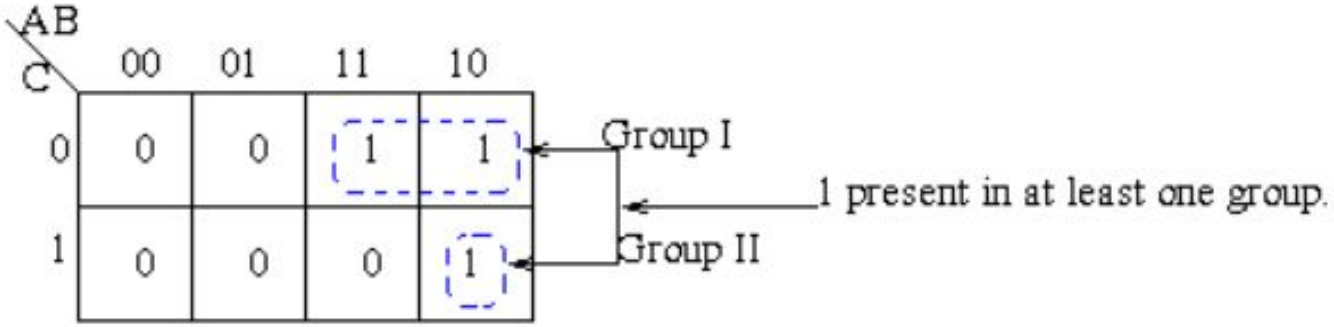


Karnaugh Maps - Rules of Simplification

- **Each group should be as large as possible.**

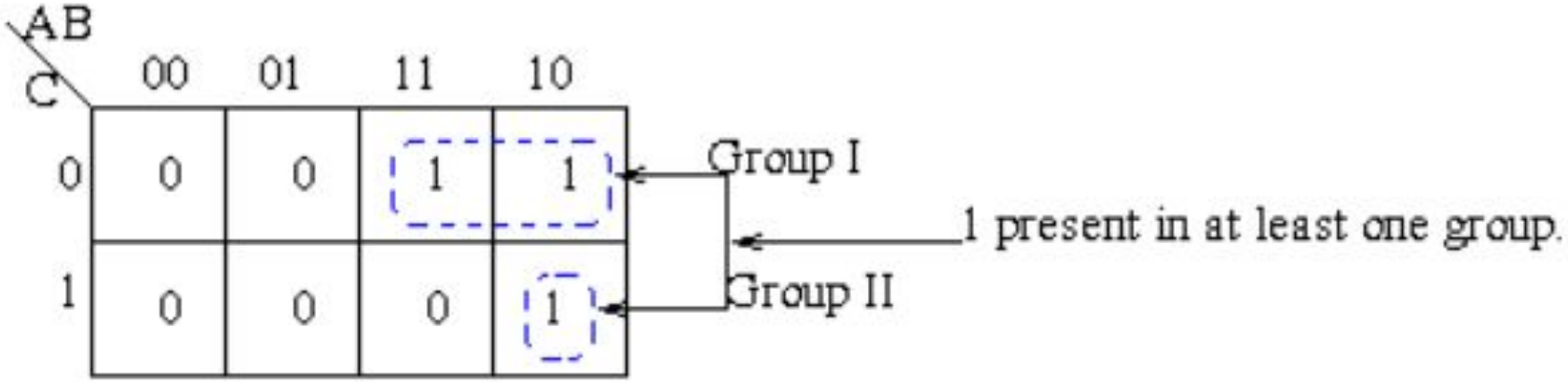


- **Each cell containing a *one* must be in at least one group.**



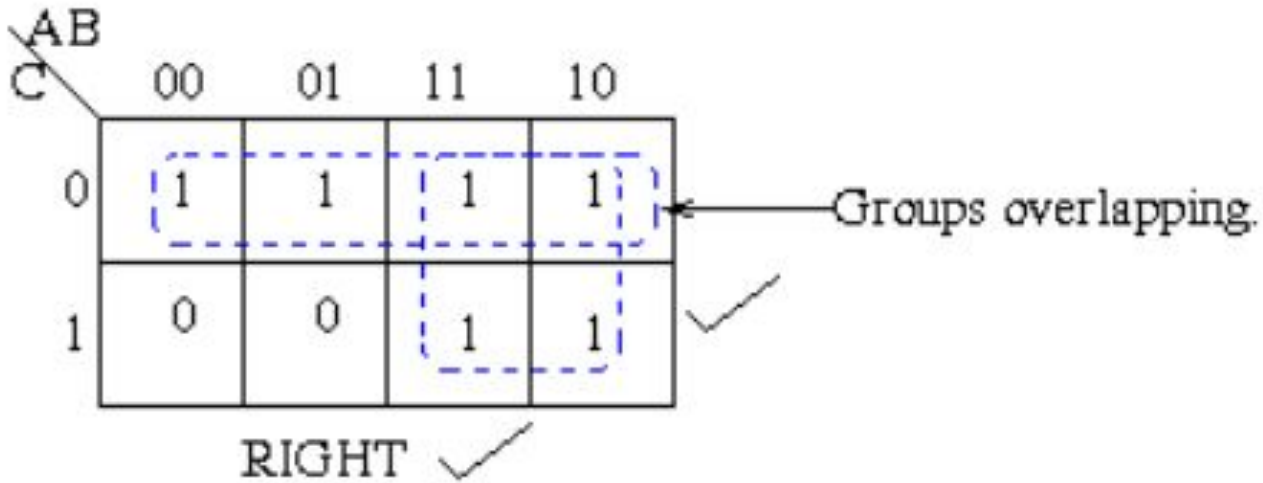
Karnaugh Maps - Rules of Simplification

- **Each cell containing a *one* must be in at least one group.**

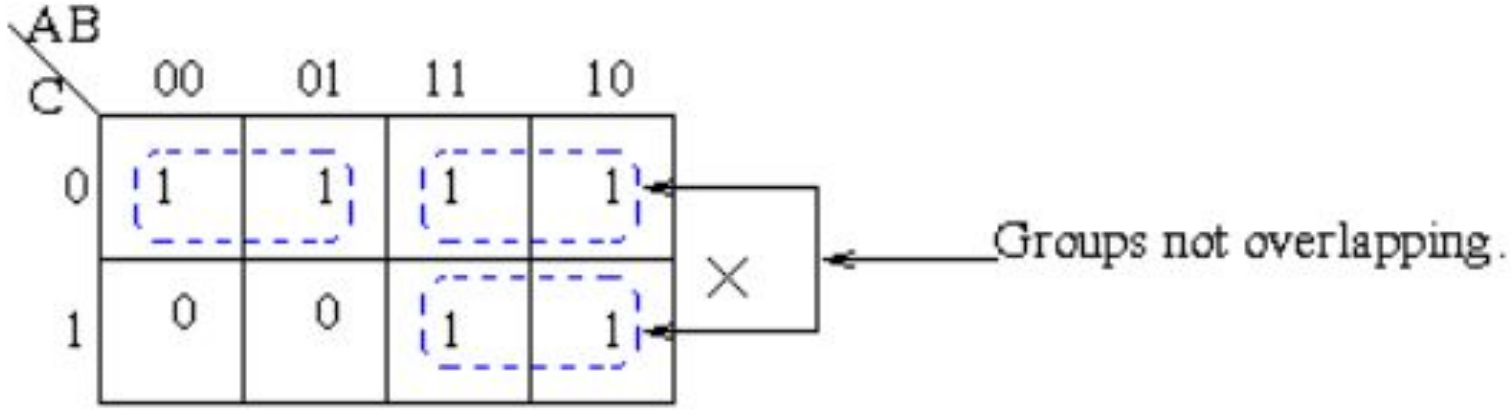


Karnaugh Maps - Rules of Simplification

- **Groups may overlap.**



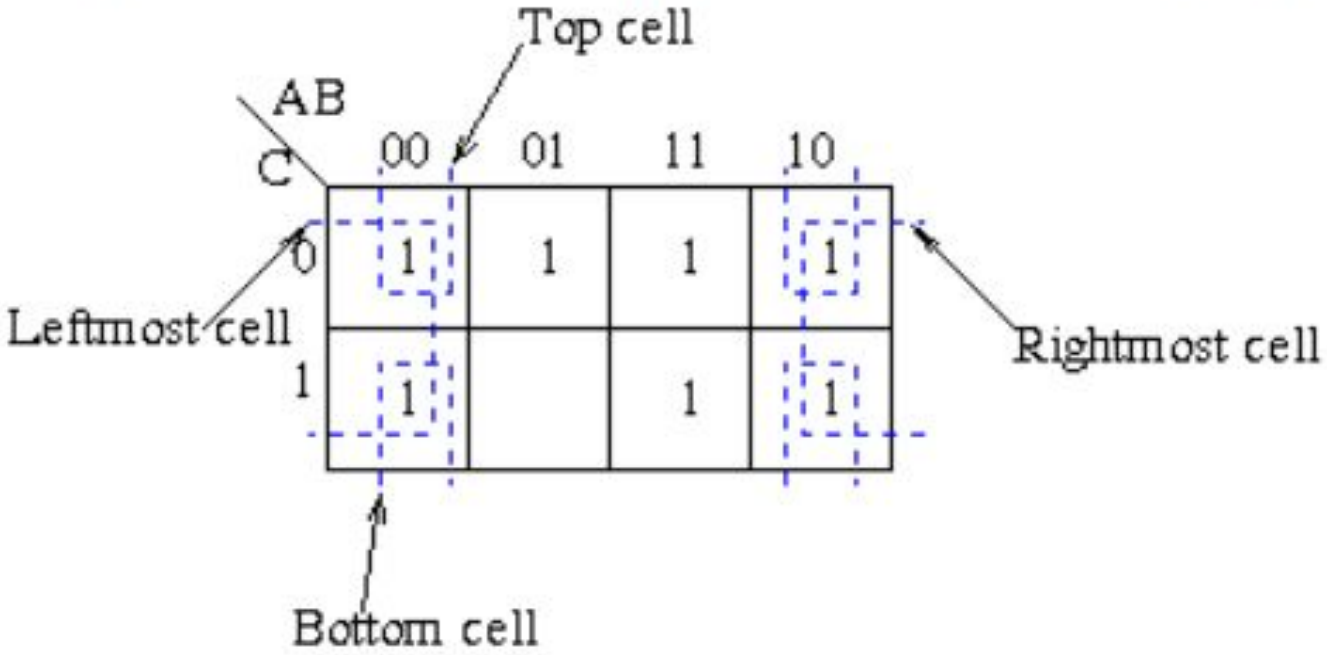
RIGHT ✓



WRONG ✗

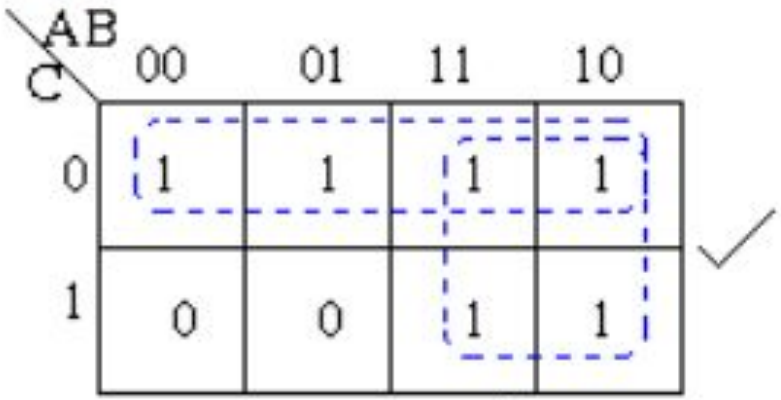
Karnaugh Maps - Rules of Simplification

- **Groups may wrap around the table. The leftmost cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.**

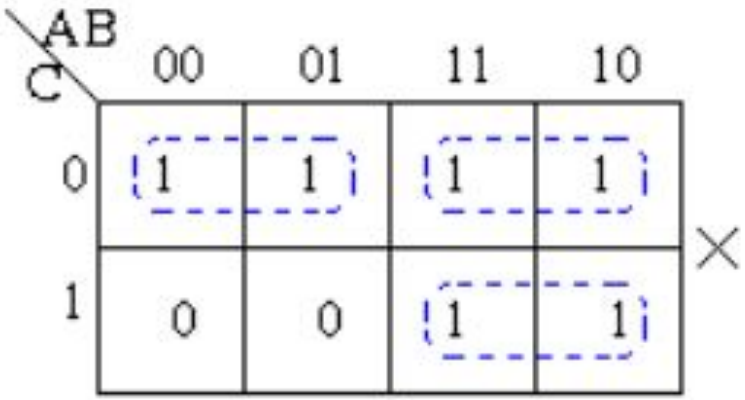


Karnaugh Maps - Rules of Simplification

- **There should be as few groups as possible, as long as this does not contradict any of the previous rules.**



RIGHT ✓

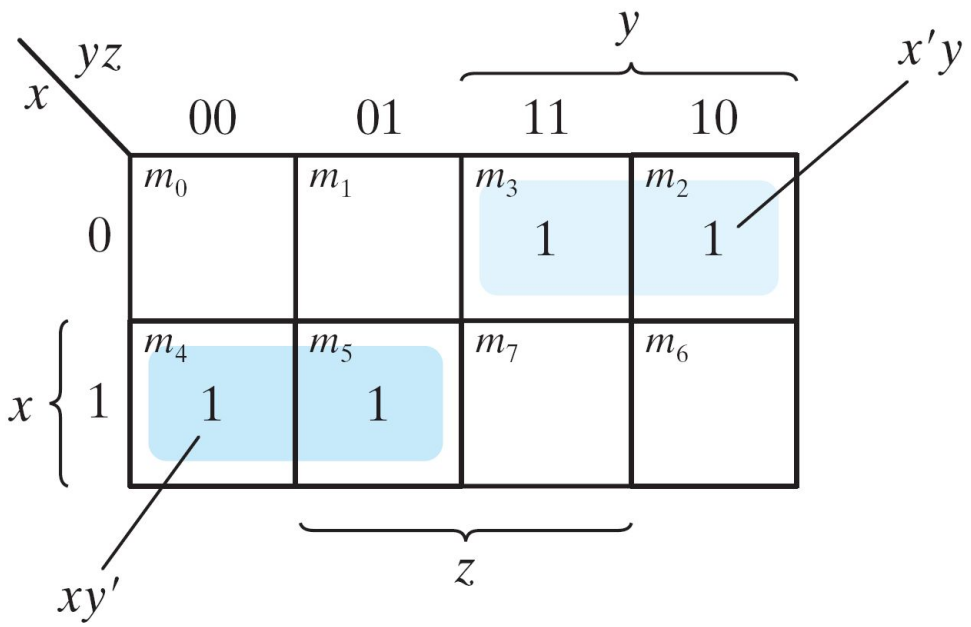


WRONG ✗

Example 1

Simplify the Boolean function F(x, y, z) = ∑(2, 3, 4, 5)

- $F(x, y, z) = \sum(2, 3, 4, 5) = x'y + xy'$



$$F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$$

Example 2

Simplify $F(x, y, z) = \sum(3, 4, 6, 7)$

- $F(x, y, z) = \sum(3, 4, 6, 7) = yz + xz'$

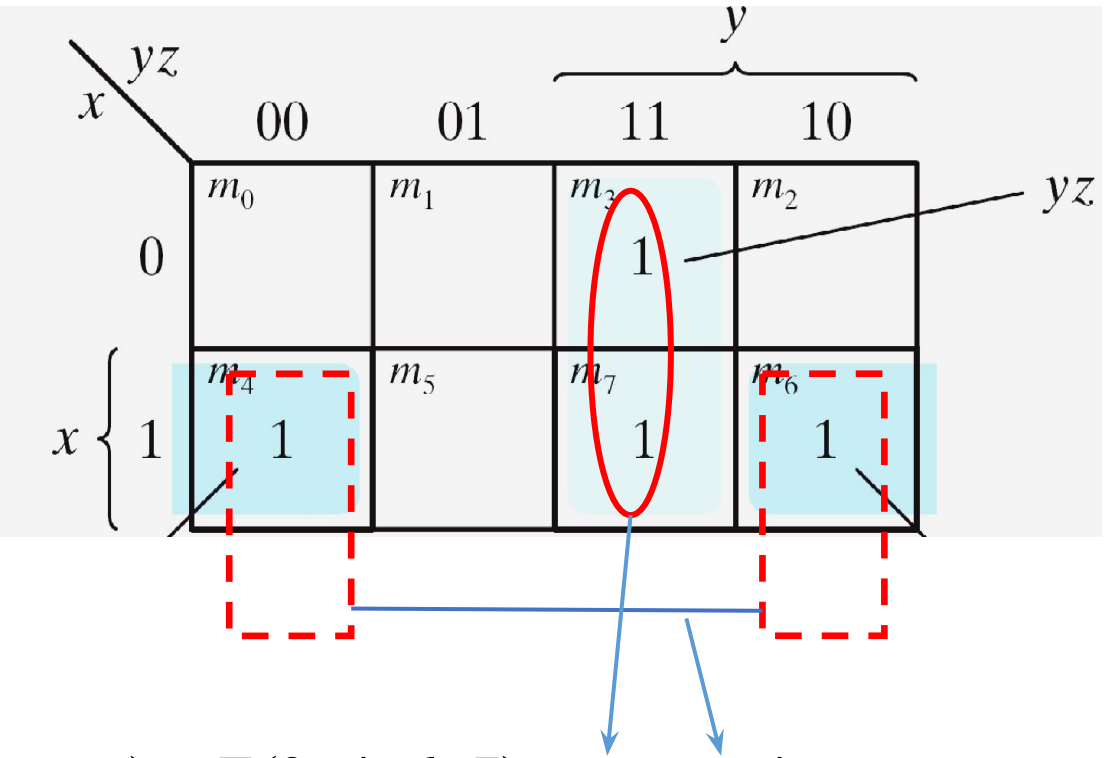
$F(x, y, z)$ =x'yz+xy'z'+xyz'+xyz

=x'yz+xz'(y'+y)+xyz

=x'yz+xz'+xyz

=yz(x'+x)+xz'

=yz+xz'

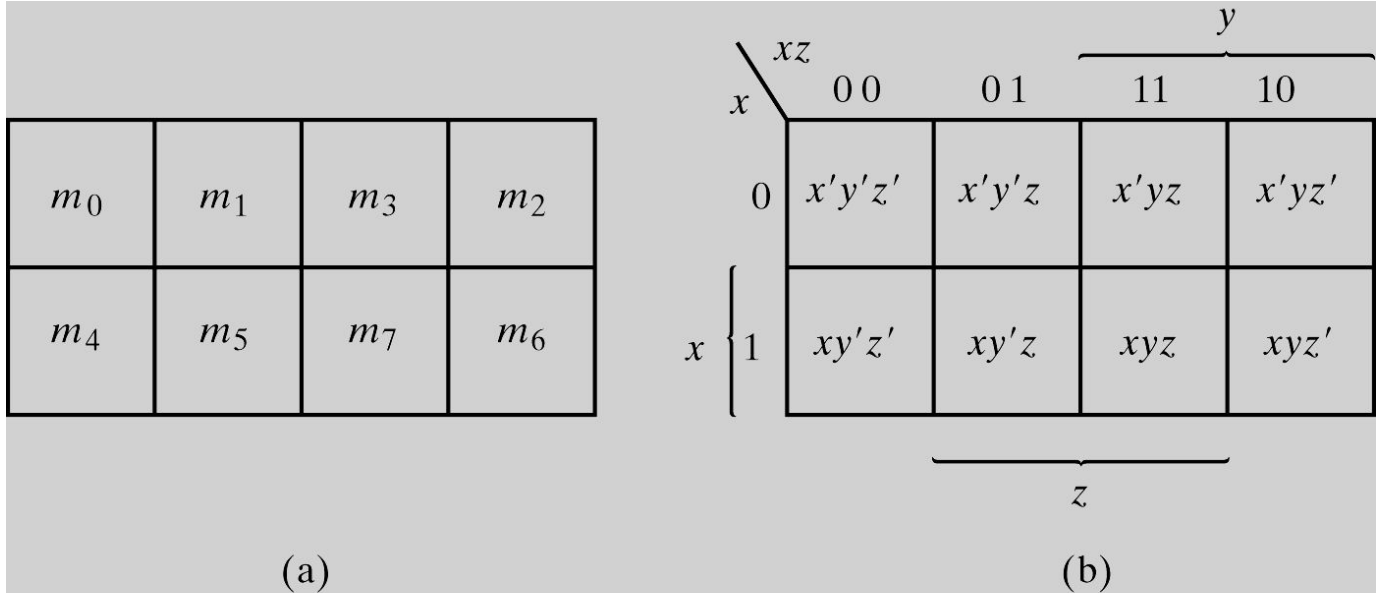


$F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

Four Adjacent Squares

0, 2, 4, and 8 squares

- $m_0+m_2+m_4+m_6 = x'y'z'+x'yz'+xy'z'+xyz' = x'z'(y'+y) +xz'(y'+y) = x'z' + xz' = z'$
- $m_1+m_3+m_5+m_7 = x'y'z+x'yz+xy'z+xyz =x'z(y'+y) + xz(y'+y) =x'z + xz = z$

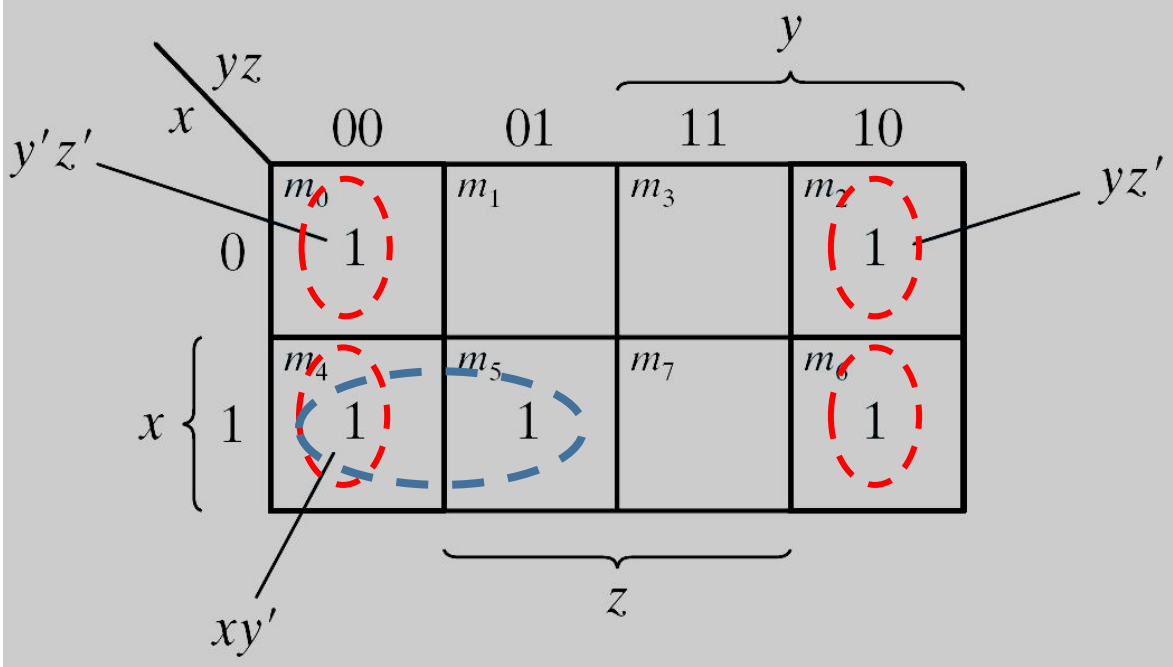


Three-variable Map

Example 3

Simplify $F(x, y, z) = \sum(0, 2, 4, 5, 6)$

- $F(x, y, z) = \sum(0, 2, 4, 5, 6) = z' + xy'$



$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$$

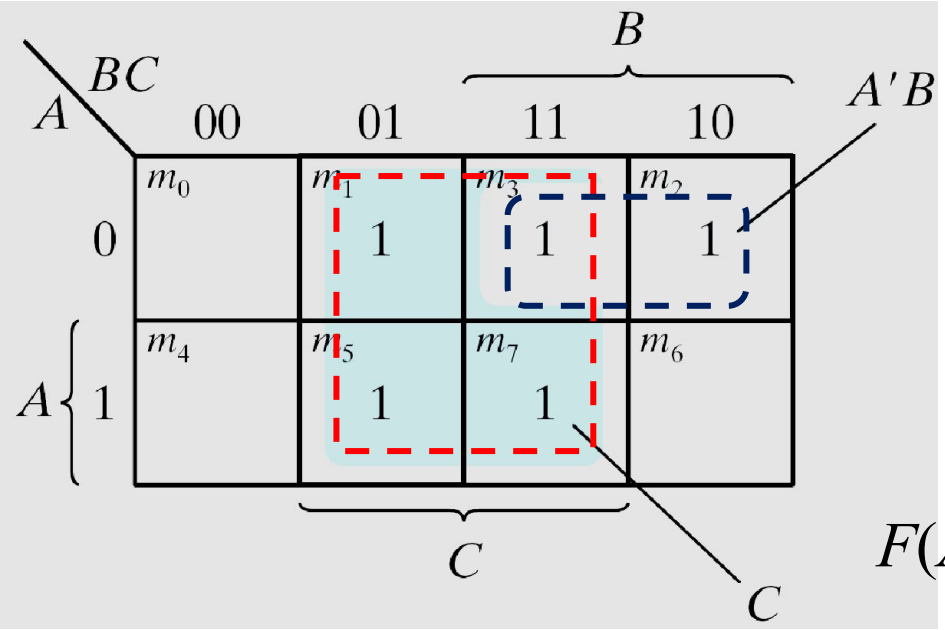
Example 4

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

a) Express it in sum of minterms.

b) Find the minimal sum of products expression.

$$F(A, B, C) = \sum(1, 2, 3, 5, 7) = C + A'B$$



$F(A, B, C) = \boldsymbol{C + A'B}$

Four-Variable Map

❖ The map
- 16 minterms
- Combinations of 2, 4, 8, and 16 adjacent squares

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}

(a)

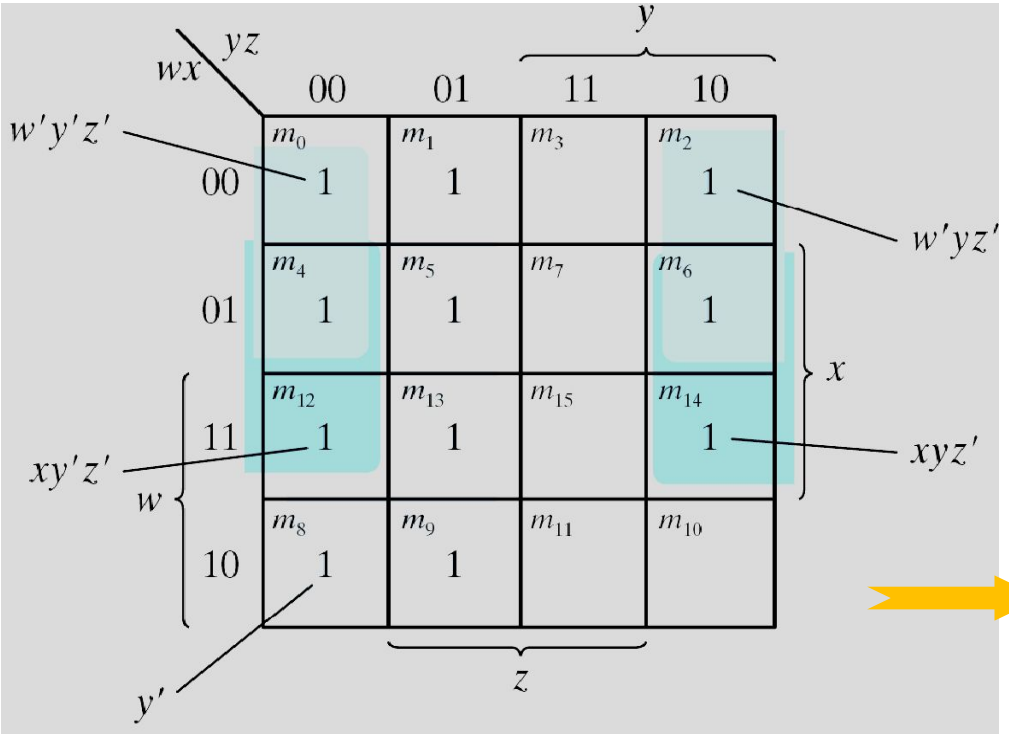
$wx \backslash yz$	00	01	11	10
00	$w'x'y'z'$	$w'x'y'z$	$w'x'yz$	$w'x'yz'$
01	$w'xy'z'$	$w'xy'z$	$w'xyz$	$w'xyz'$
11	$wxy'z'$	$wxy'z$	$wxyz$	$wxyz'$
10	$wx'y'z'$	$wx'y'z$	$wx'yz$	$wx'yz'$

(b)

Four-variable Map

Example 5

❖ Simplify $F(w, x, y, z) = S(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)$

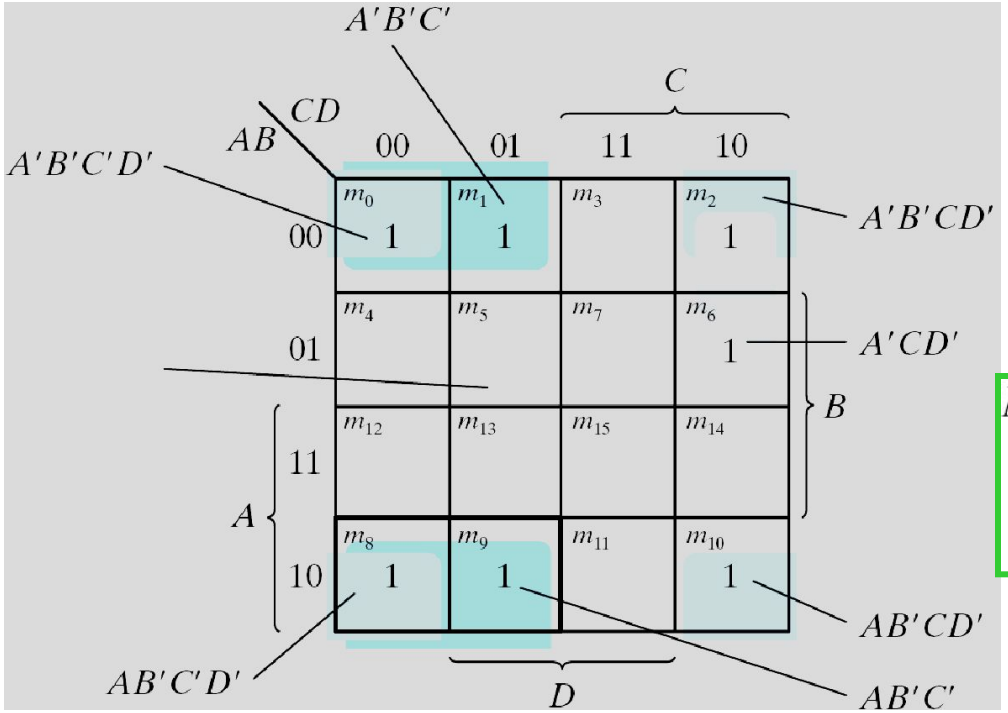


$F = y' + w'z' + xz'$

$F(w, x, y, z) = \Sigma(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14) = \boldsymbol{y' + w'z' + xz'}$

Example 6

- Simplify $F = A'B'C' + B'CD' + A'BCD' + AB'C'$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
$AB'C'D' + AB'CD' = AB'D'$
$A'B'D' + AB'D' = B'D'$
$A'B'C' + AB'C' = B'C'$

$$A'B'C' + B'CD' + A'B'C'D' + AB'C' = B'D' + B'C' + A'CD'$$

Five-Variable Map

❖ Map for more than four variables becomes complicated

- Five-variable map: two four-variable map (one on the top of the other).

$A = 0$

BC \ DE	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

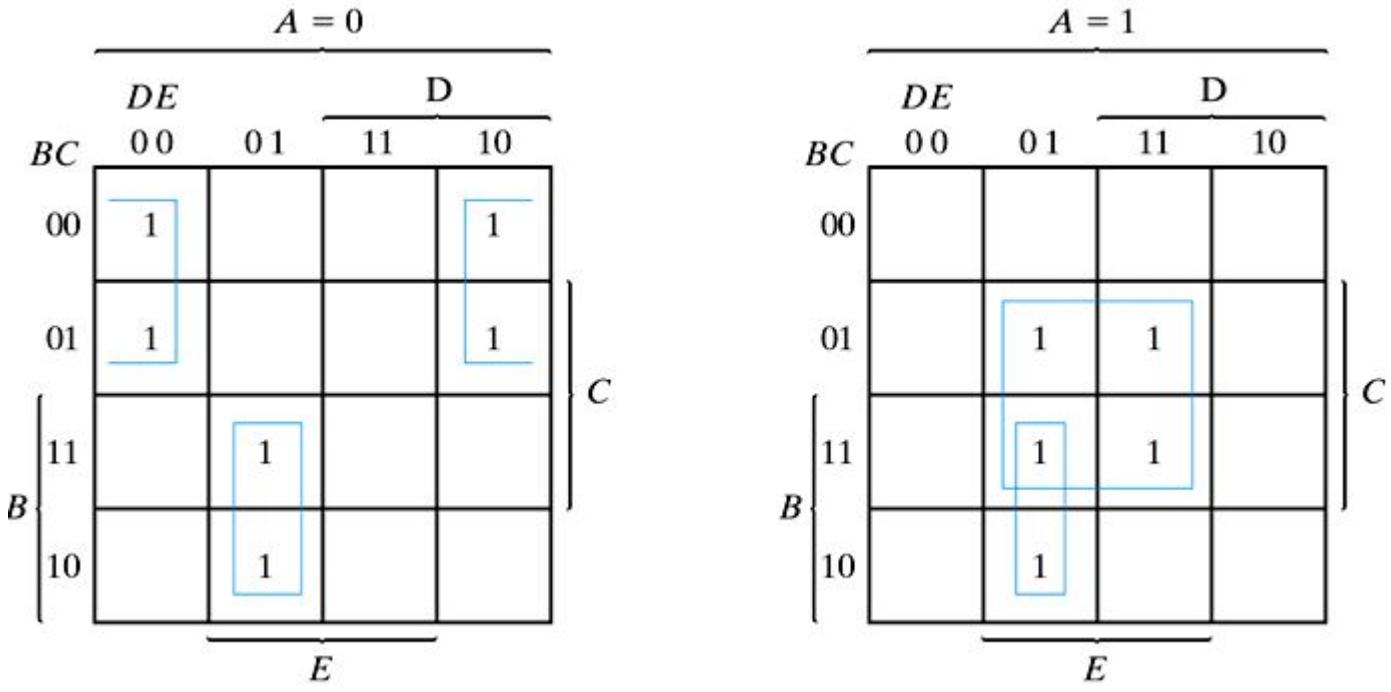
$A = 1$

BC \ DE	00	01	11	10
00	16	17	19	18
01	20	21	23	22
11	28	29	31	30
10	24	25	27	26

Five-variable Map

Example 7

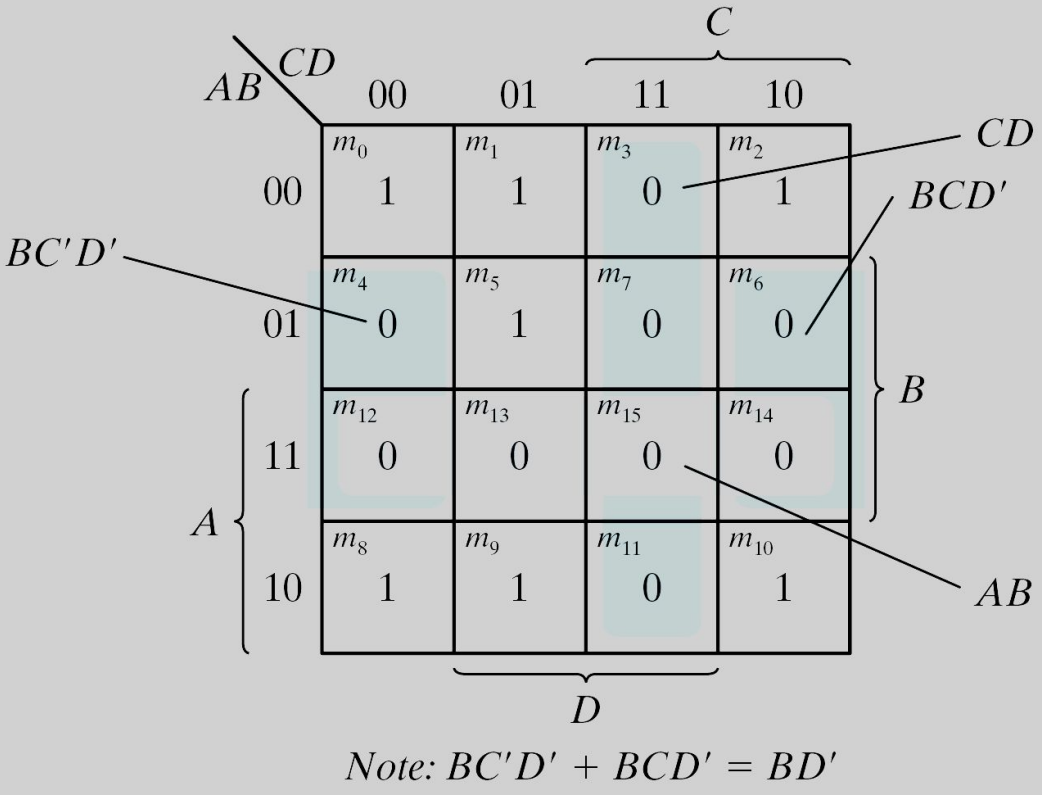
- Simplify $F = \sum(0, 2, 4, 6, 9, 13, 21, 23, 25, 29, 31)$



$F = A'B'E' + BD'E + ACE$

Example 8

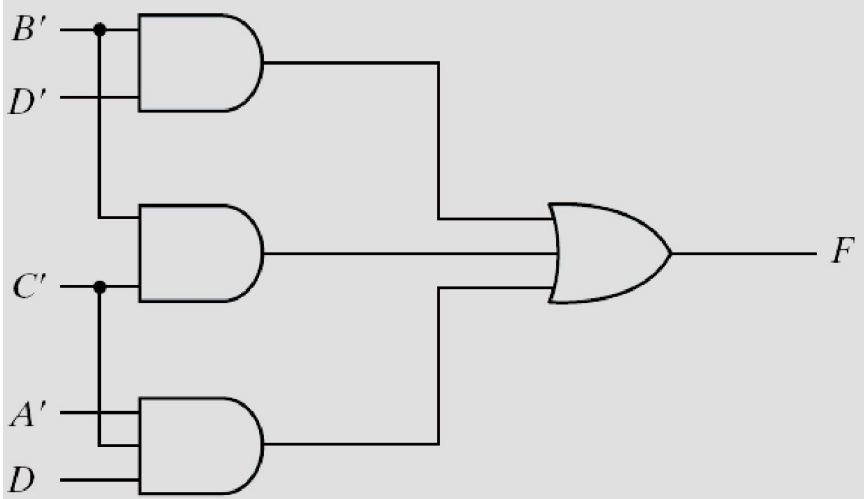
- Simplify $F = \sum(0, 1, 2, 5, 8, 9, 10)$ into sum-of-products form



$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10) = B'D' + B'C' + A'C'D$

Example 8 (cont.)

- Gate implementation of the function of Example 8



(a) $F = B'D' + B'C' + A'C'D$

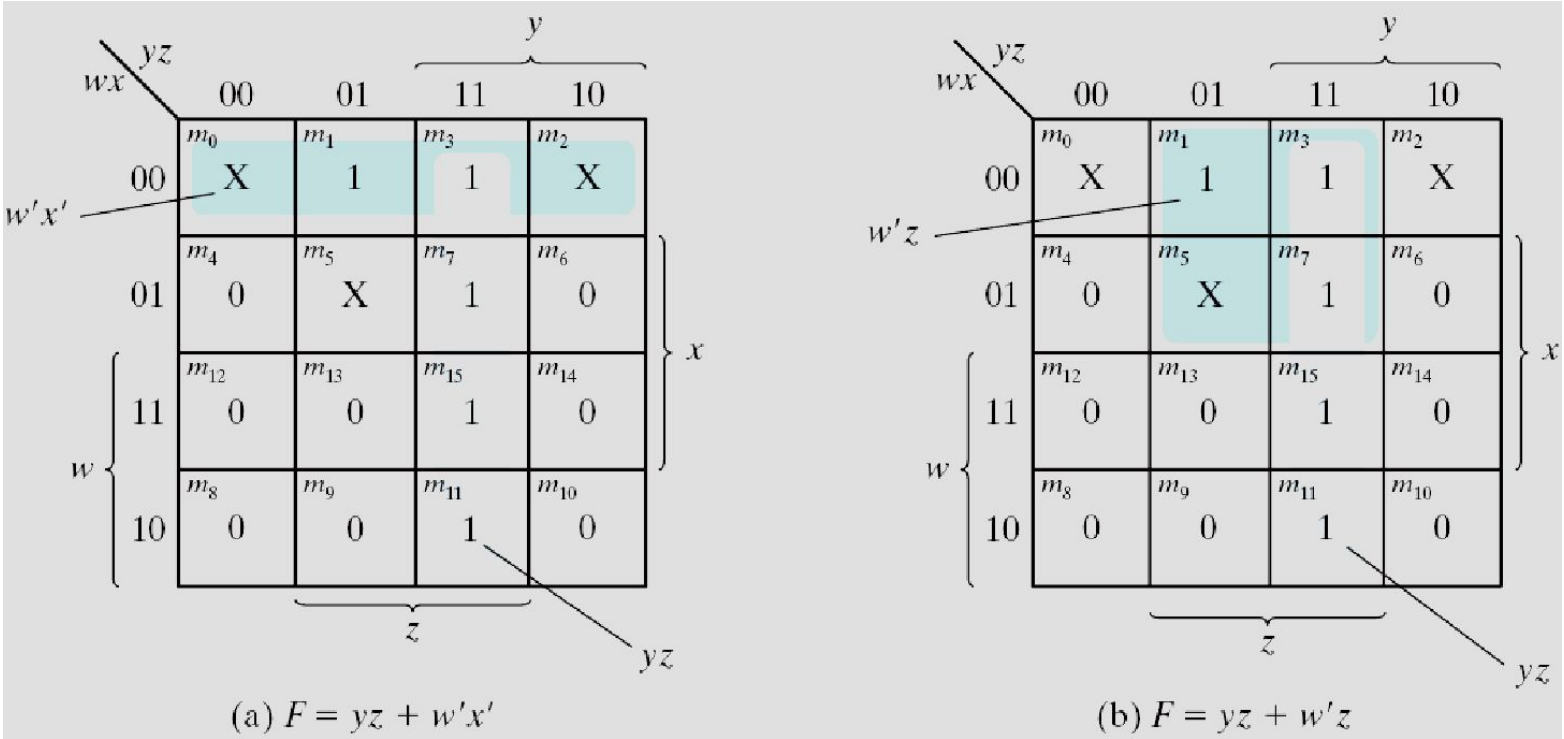
Don't-Care Conditions

- ❖ The value of a function is not specified for certain combinations of variables
- ❖ The don't-care conditions can be utilized in logic minimization
  - Can be implemented as 0 or 1

Example 9

❖Simplify $F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$ which has the don't-care conditions $d(w, x, y, z) = \sum(0, 2, 5)$.

- $F = yz + w'x'$; $F = yz + w'z$
- $F = \sum(0, 1, 2, 3, 7, 11, 15)$; $F = \sum(1, 3, 5, 7, 11, 15)$
- Either expression is acceptable

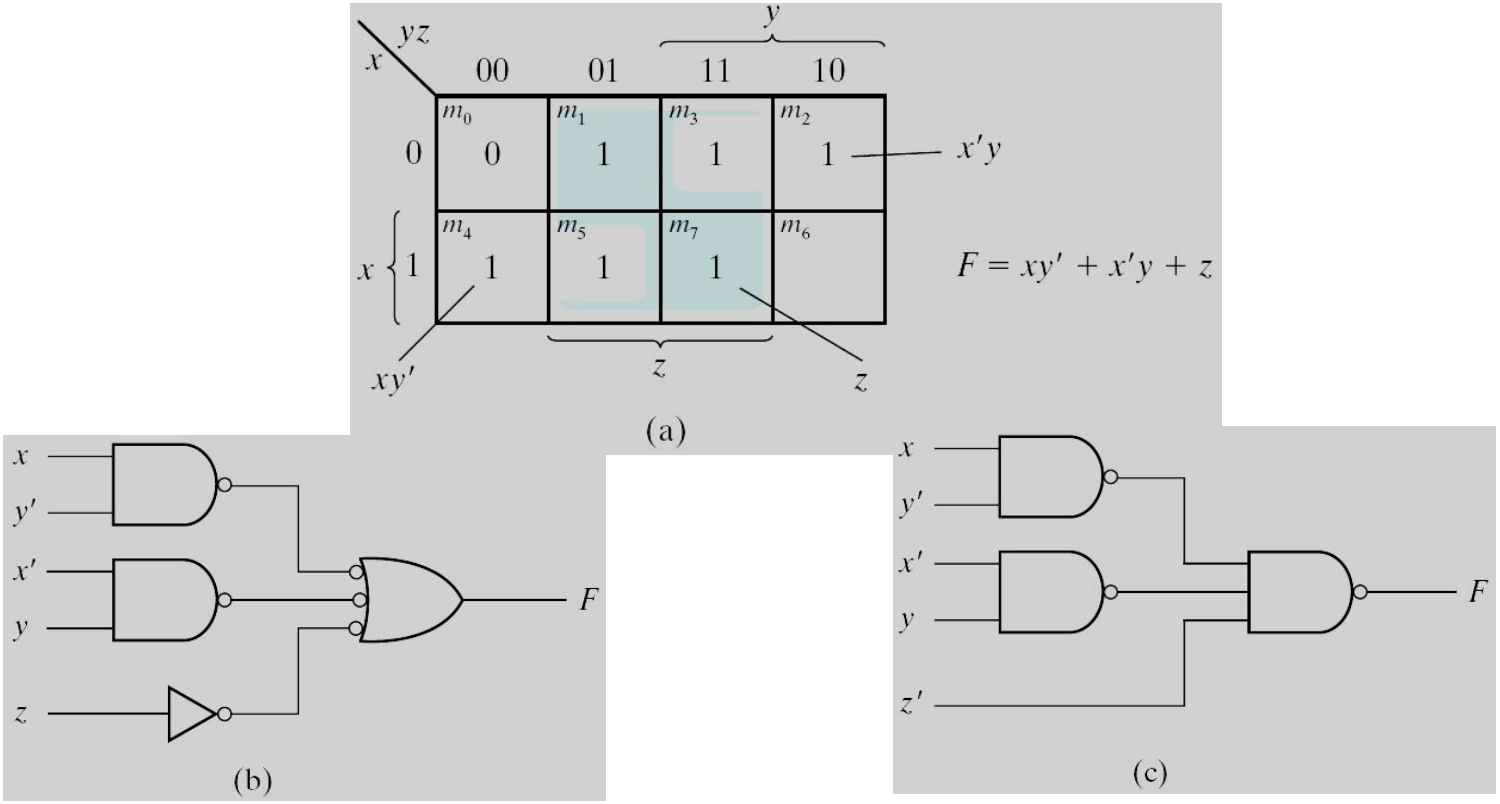


Example with don't-care Conditions

Example 10

- Implement $F(x, y, z) = \sum(1, 2, 3, 4, 5, 7)$

$$F(x,y,z) = \sum(1,2,3,4,5,7) \longrightarrow F(x,y,z) = xy' + x'y + z$$



Solution to Example10

Karnaugh Maps - Rules of Simplification : Summary

- No zeros allowed.
- No diagonals.
- Only power of 2 number of cells in each group.
- Groups should be as large as possible.
- Every one must be in at least one group.
- Overlapping allowed.
- Wrap around allowed.
- Fewest number of groups possible.

Implicant:

- Implicants: The group's of 1's is implicants
- Prime Implicants: It largest possible group of 1's.
- Essential Prime Implicants: At least, there is single 1 which can not be combined in other way.
- Redundant Prime Implicants: The prime implicants for which each of its minterm is covered by some essential prime implicant are **redundant prime implicants (RPI)**.
- Selective Prime Implicants: The prime implicants for which are neither essential nor redundant prime implicants are called **selective prime implicants (SPI)**.

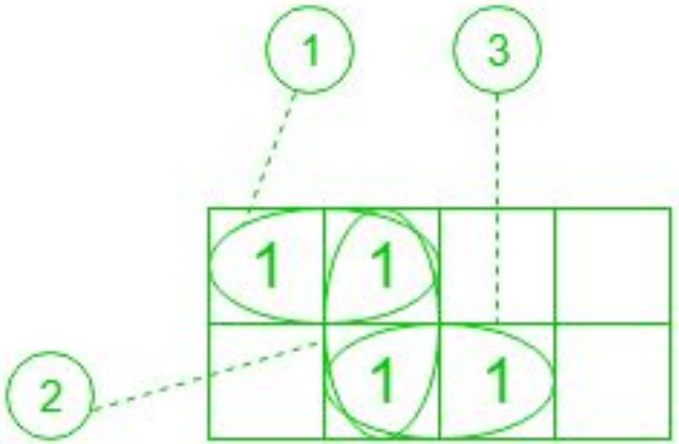
Prime Implicants

- **Prime Implicants**
  - All the minterms are covered.
  - Minimize the number of terms.
  - A prime implicant: a product term obtained by combining the maximum possible number of adjacent squares (combining all possible maximum numbers of squares).
  - Essential P.I.: a minterm is covered by only one prime implicant.
  - The essential P.I. must be included.

Prime Implicant

A group of square or rectangle made up of bunch of adjacent minterms which is allowed by definition of K-Map are called **prime implicant (PI)** i.e. all possible groups formed in K-Map.

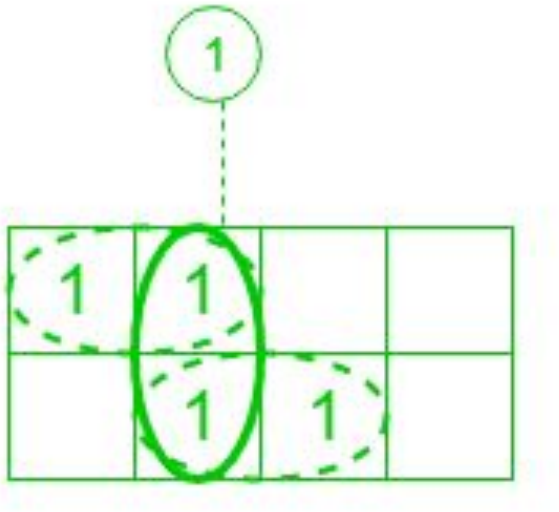
Example:



Number of Prime Implicants=3

Redundant Prime Implicants

The prime implicants for which each of its minterm is covered by some essential prime implicant are **redundant prime implicants (RPI)**. This prime implicant never appears in final solution.
Example:

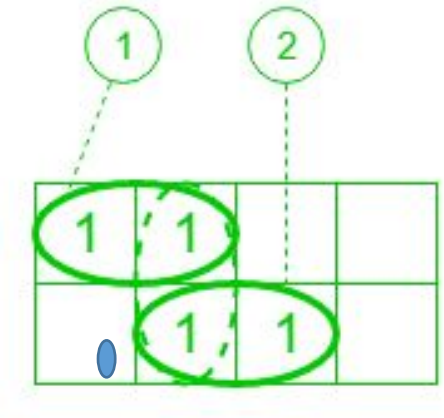


Number of Redundant Prime Implicant=1

Essential Prime Implicants

Essential Prime Implicants – These are those subcubes(groups) which cover atleast one minterm that can't be covered by any other prime implicant. **Essential prime implicants(EPI)** are those prime implicants which always appear in final solution.

Example:

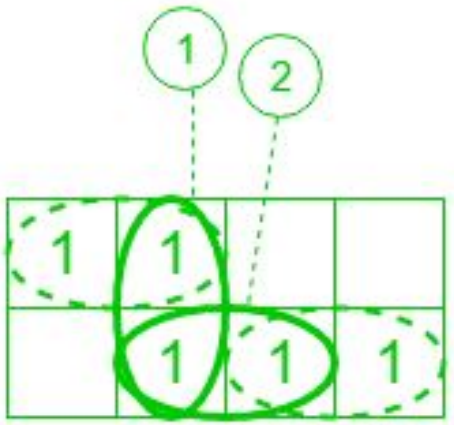


Number of Essential Prime Implicants=2

Selective Prime Implicants

The prime implicants for which are neither essential nor redundant prime implicants are called **selective prime implicants(SPI)**. These are also known as non-essential prime implicants. They may appear in some solution or may not appear in some solution.

Example:



Number of Selective Prime Implicants=2

Example-11

Given $F = \sum(1, 5, 6, 7, 11, 12, 13, 15)$, find number of implicant, PI, EPI, RPI and SPI.

Example-11

Given F = ∑(1, 5, 6, 7, 11, 12, 13, 15), find number of implicant, PI, EPI, RPI and SPI.

No. of Implicants =
No. of Prime Implicants(PI) =
No. of Essential Prime Implicants(EPI) =
No. of Redundant Prime Implicants(RPI) =
No. of Selective Prime Implicants(SPI) =

Example-11

Given F = ∑(1, 5, 6, 7, 11, 12, 13, 15), Find number of implicant, PI, EPI, RPI and SPI.

No. of Implicants = 8
No. of Prime Implicants(PI) =
No. of Essential Prime Implicants(EPI) =
No. of Redundant Prime Implicants(RPI) =
No. of Selective Prime Implicants(SPI) =

Example-11

Given F = ∑(1, 5, 6, 7, 11, 12, 13, 15), Find number of implicant, PI, EPI, RPI and SPI.

No. of Implicants = 8
No. of Prime Implicants(PI) = 5
No. of Essential Prime Implicants(EPI) =
No. of Redundant Prime Implicants(RPI) =
No. of Selective Prime Implicants(SPI) =

Example-11

Given F = ∑(1, 5, 6, 7, 11, 12, 13, 15), Find number of implicant, PI, EPI, RPI and SPI.

No. of Implicants = 8
No. of Prime Implicants(PI) = 5
No. of Essential Prime Implicants(EPI) = 4
No. of Redundant Prime Implicants(RPI) =
No. of Selective Prime Implicants(SPI) =

Example-11

Given F = ∑(1, 5, 6, 7, 11, 12, 13, 15), Find number of implicant, PI, EPI, RPI and SPI.

No. of Implicants = 8
No. of Prime Implicants(PI) = 5
No. of Essential Prime Implicants(EPI) = 4
No. of Redundant Prime Implicants(RPI) = 1
No. of Selective Prime Implicants(SPI) =

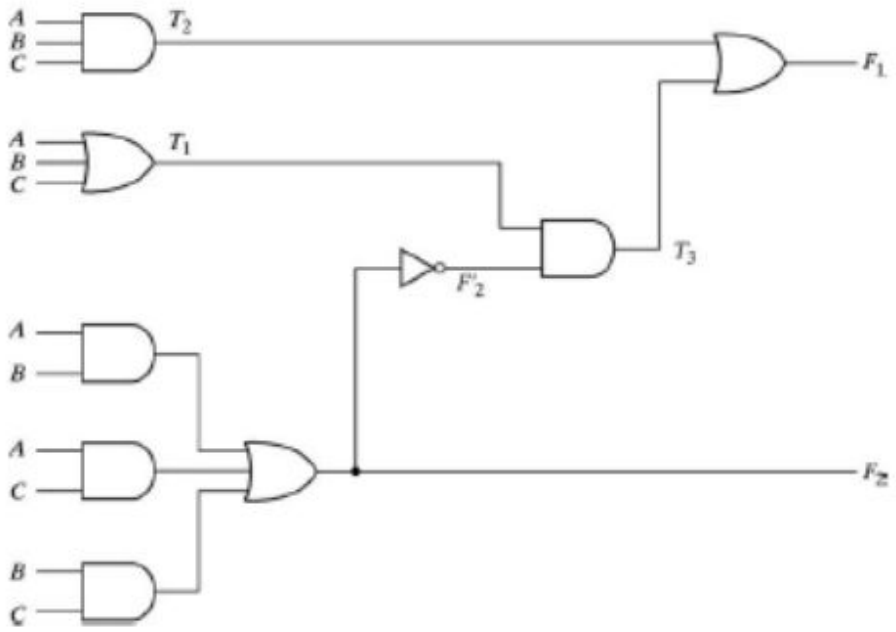
Example-11

Given F = ∑(1, 5, 6, 7, 11, 12, 13, 15), Find number of implicant, PI, EPI, RPI and SPI.

No. of Implicants = 8
No. of Prime Implicants(PI) = 5
No. of Essential Prime Implicants(EPI) = 4
No. of Redundant Prime Implicants(RPI) = 1
No. of Selective Prime Implicants(SPI) = 0

Design Procedure

- Design Procedure:
  - Input: the specification of the problem
  - Output : the logic circuit diagram (or Boolean function)
- **Step1**: determine the required number of inputs and outputs from the specification
- **Step2**: derive the truth table that defines the required relationship between inputs and outputs
- **Step3**: obtain the simplified Boolean function for each output as a function of the input variables
- **Step4**: draw the logic diagram and verify the correctness of the design



Design of BCD to Excess -3 Code Converter Circuit

- Convert from BCD code to Excess-3 code
- The 6 input combinations not listed are don't cares
- These values have no meaning in BCD
- We can arbitrary assign them to 1 or 0

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

- <u>BCD to Excess-3 Code</u>

  - 0 0 0 1 =1
  - 0 0 1 1 =3
  - + 0 1 0 0 =4

Maps for Code Conversion…. Example

The six don't care minterms (10-15) are marked with x

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

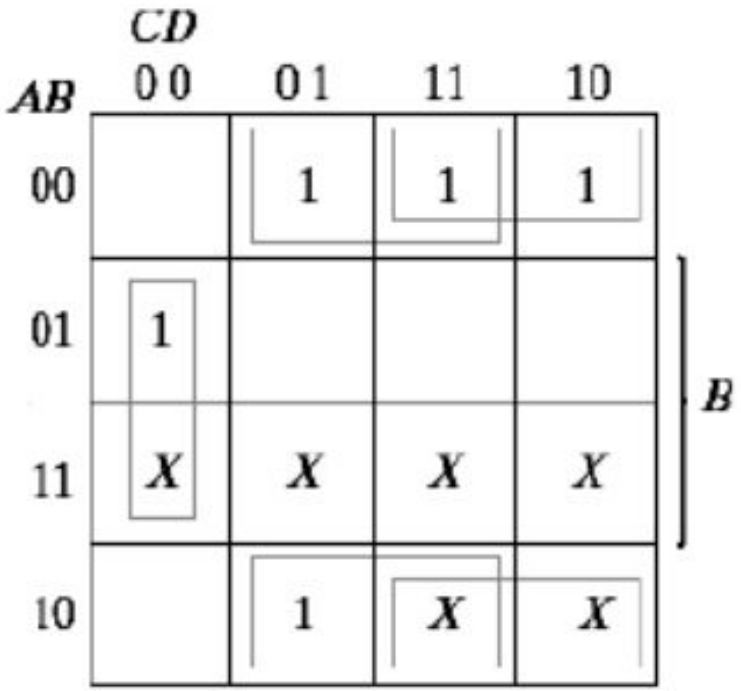
z=D'

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

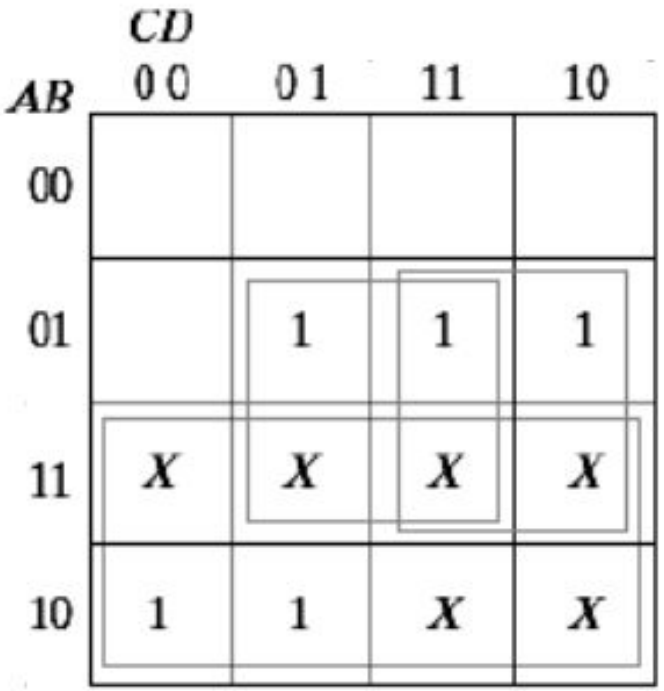
C

y=CD+C'D'

Maps for Code Conversion…. Example



x=B’C+B’D+BC’D’



w=A+BC+BD

Logic Diagram for the Converter

- There are various possibilities for a logic diagram that implements a circuit
- A two-level logic diagram may be obtained directly from the Boolean expressions derived by the maps
- The expressions may be manipulated algebraically to use common gates for two or more outputs
- Reduce the number of gates used

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

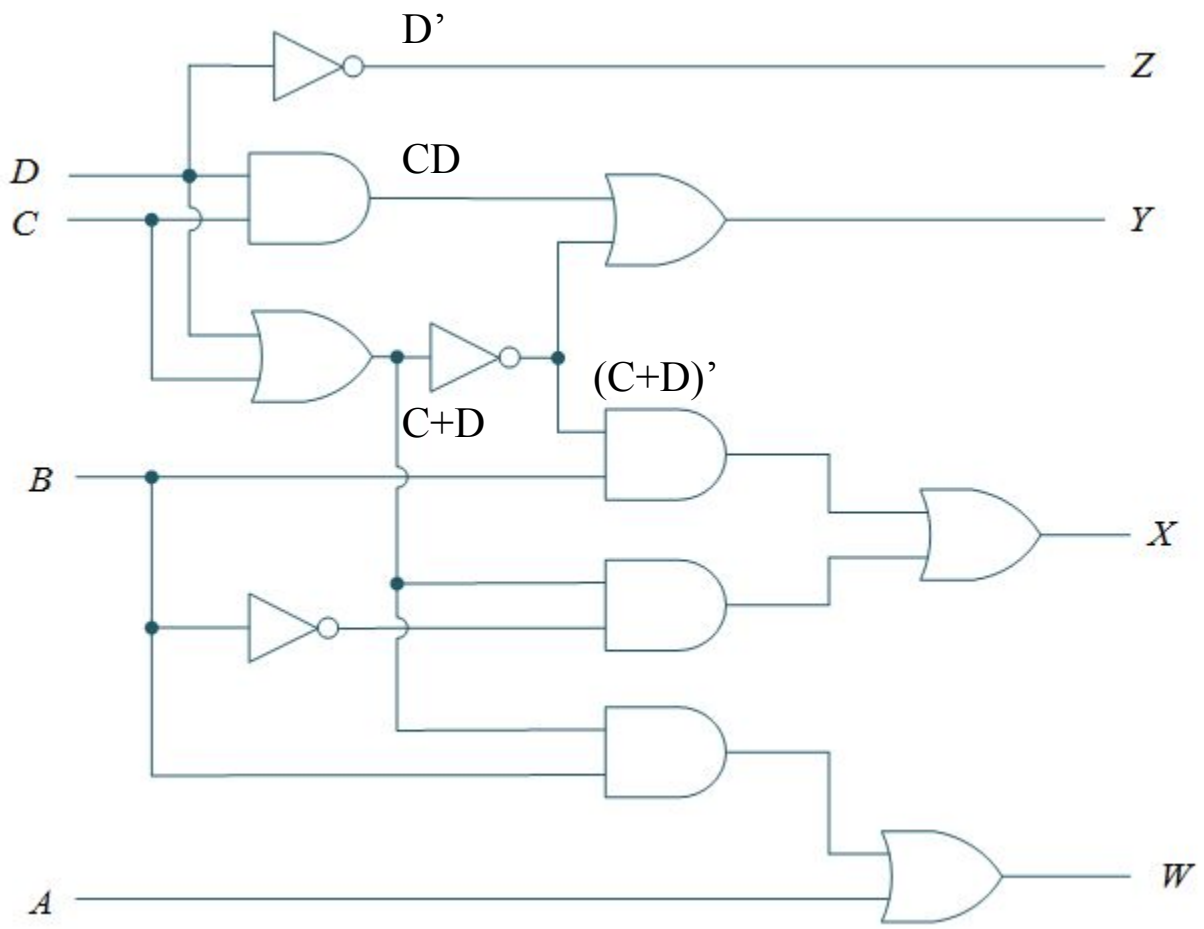
$$W = A + BC + BD$$

$$X = \bar{B}C + \bar{B}D + B\bar{C}\bar{D}$$

$$Y = CD + \bar{C}\bar{D}$$

$$Z = \bar{D}$$

Logic Diagram for the Converter



Application of Combinational Circuit

Parity Bit:

- The parity generating technique is one of the most widely used error detection techniques for the data transmission.
- In digital systems, when binary data is transmitted and processed, data may be subjected to noise so that such noise can alter 0s (of data bits) to 1s and 1s to 0s.
- Hence, parity bit is added to the word containing data in order to make number of 1's either even or odd.

Parity Generator

- A parity generator is a combinational logic circuit that generates the parity bit in the transmitter
- On the other hand, a circuit that checks the parity in the receiver is called parity checker

Parity Generation and Checking

Even-Parity-Generator Truth Table

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
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

Parity Generation and Checking

Even-Parity-Checker Truth Table

Four Bits Received				Parity Error Check
x	*y*	*z*	*P*	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Even Parity Generator

3-bit message			Even parity bit generator (P)
A	B	C	Y
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

K-map simplification for 3-bit message even parity generator is

K-Map Simplification

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

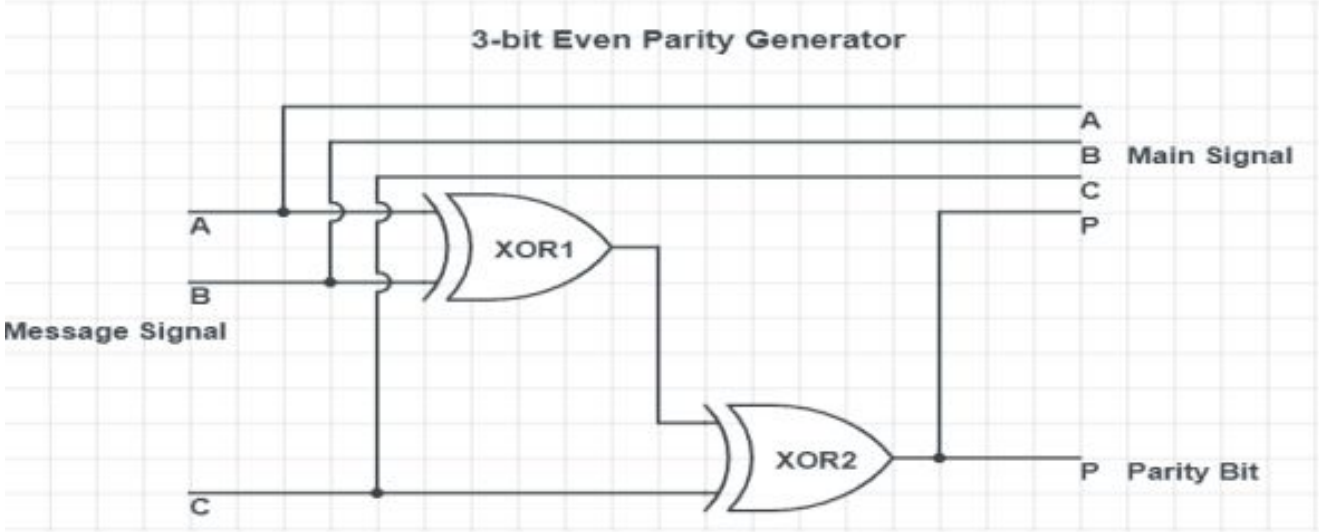
From the above truth table, the simplified expression of the parity bit can be written as

$$P = \bar{A}\,\bar{B}\,C + \bar{A}\,B\,\bar{C} + A\,\bar{B}\,\bar{C} + A\,B\,C$$

$$= \bar{A}\,(\bar{B}\,C + B\,\bar{C}) + A\,(\bar{B}\,\bar{C} + B\,C)$$

$$= \bar{A}\,(B \oplus C) + A\,(\overline{B \oplus C})$$

$$P = A \oplus B \oplus C$$

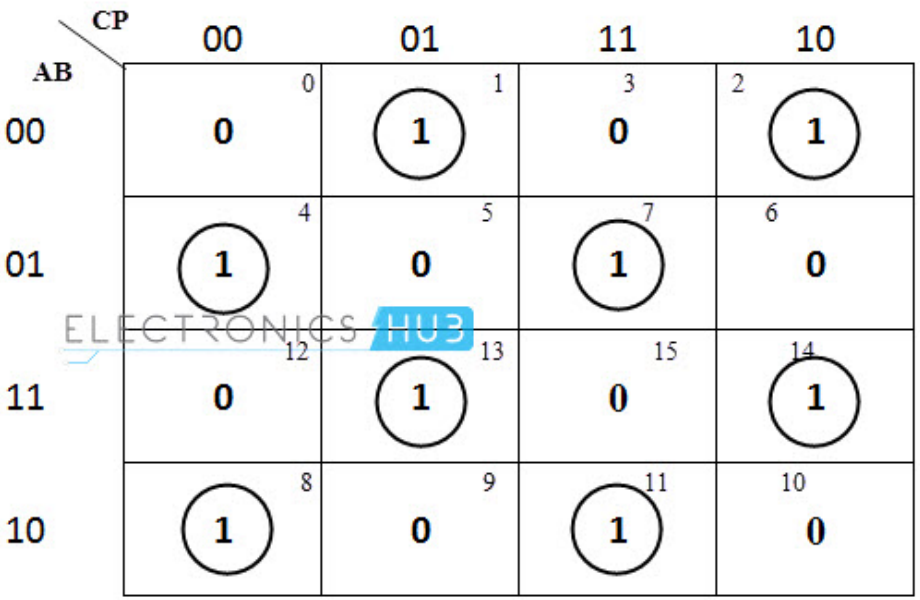


Even Parity Checker

❖ **The below table shows the truth table for the even parity checker :**

- PEC = 1 if the error occurs, i.e., the four bits received have odd number of 1s and
- PEC = 0 if no error occurs, i.e., if the 4-bit message has even number of 1s.

4-bit received message				Parity error check C_p
A	B	C	P	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0



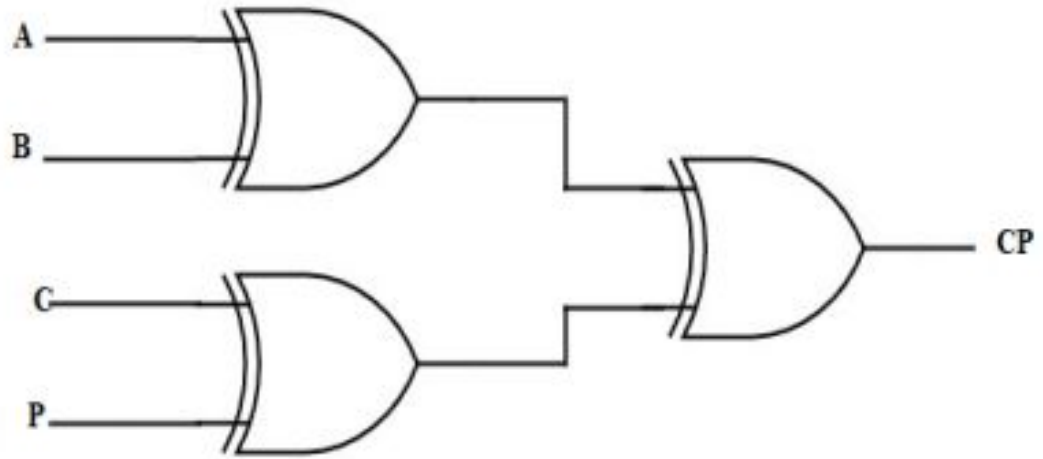
Even Parity Checker

$$CP = \bar{A}\,\bar{B}\,(\bar{C}\,D + C\,\bar{D}) + \bar{A}\,B\,(\bar{C}\,\bar{D} + C\,D) + A\,B\,(\bar{C}\,D + C\,\bar{D}) + A\,\bar{B}\,(\bar{C}\,\bar{D} + C\,D)$$

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

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

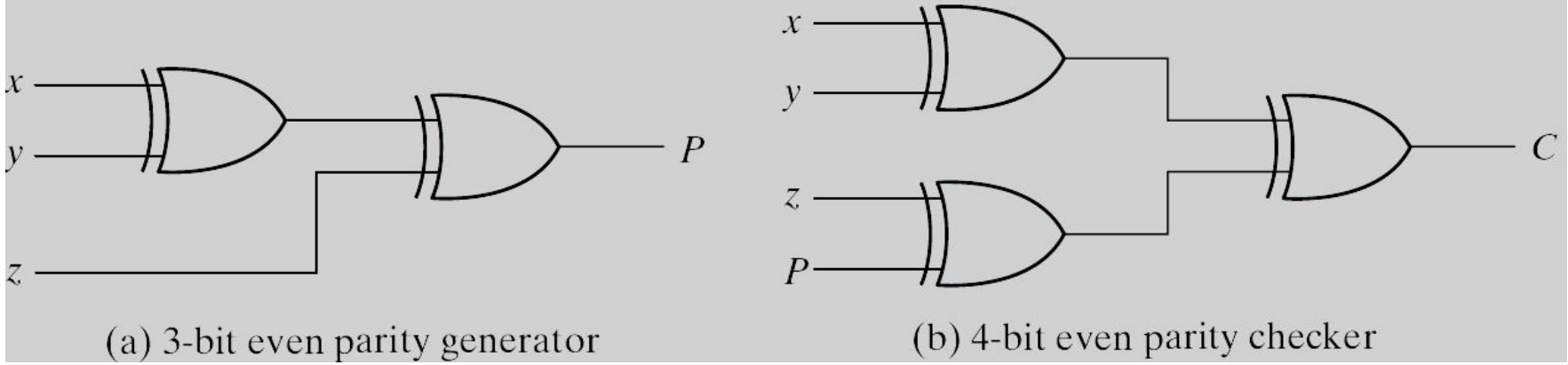
$$= (A \oplus B) \oplus (C \oplus D)$$



The above logic expression for the even parity checker can be implemented by using three Ex-OR gates as shown in figure. If the received message consists of five bits, then one more Ex-OR gate is required for the even parity checking.

Parity Generation and Checking

- Parity Generation and Checking
  - A parity bit: P = $x \oplus y \oplus z$
  - Parity check: C = $x \oplus y \oplus z \oplus P$
    - C=1: one bit error or an odd number of data bit error
    - C=0: correct or an even # of data bit error



Logic Diagram of a Parity Generator and Checker

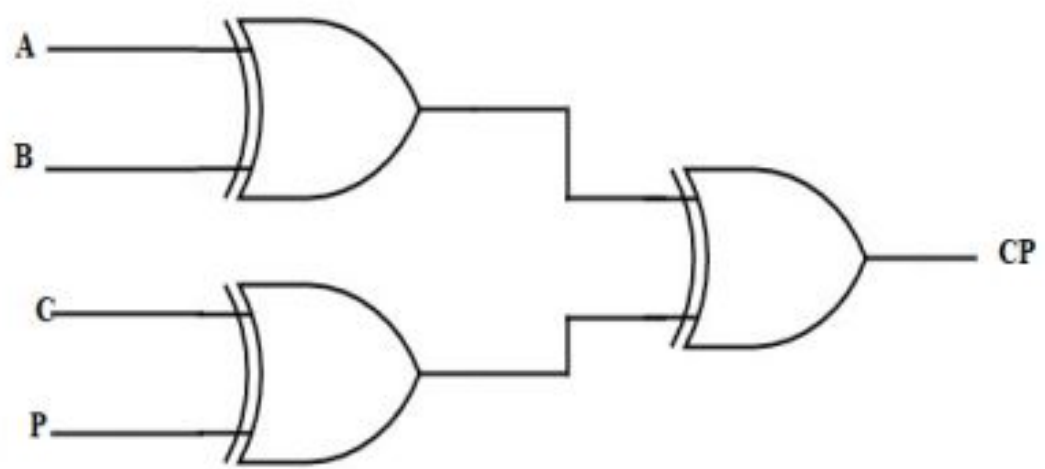
Even Parity Checker

$$CP = \bar{A}\,\bar{B}\,(\bar{C}\,D + C\,\bar{D}) + \bar{A}\,B\,(\bar{C}\,\bar{D} + C\,D) + A\,B\,(\bar{C}\,D + C\,\bar{D}) + A\,\bar{B}\,(\bar{C}\,\bar{D} + C\,D)$$

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

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

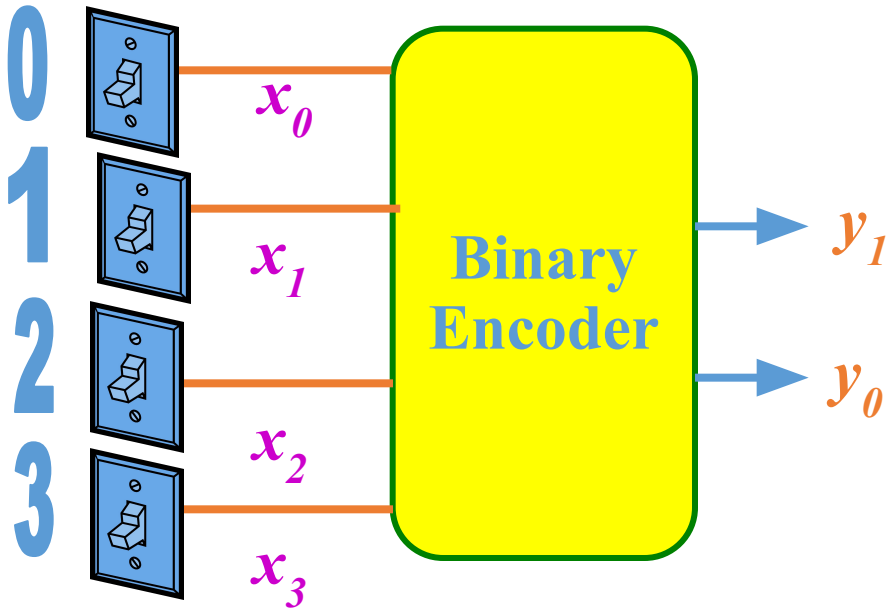
$$= (A \oplus B) \oplus (C \oplus D)$$



The above logic expression for the even parity checker can be implemented by using three Ex-OR gates as shown in figure. If the received message consists of five bits, then one more Ex-OR gate is required for the even parity checking.

Encoders

- Put "*Information*" into code
- Binary Encoder
  - Example: 4-to-2 Binary Encoder



Only *one* switch should be activated at a time

x_3	x_2	x_1	y_1	y_0
0	0	0	0	0
0	0	1	0	1
0	1	0	1	0
1	0	0	1	1

Encoders

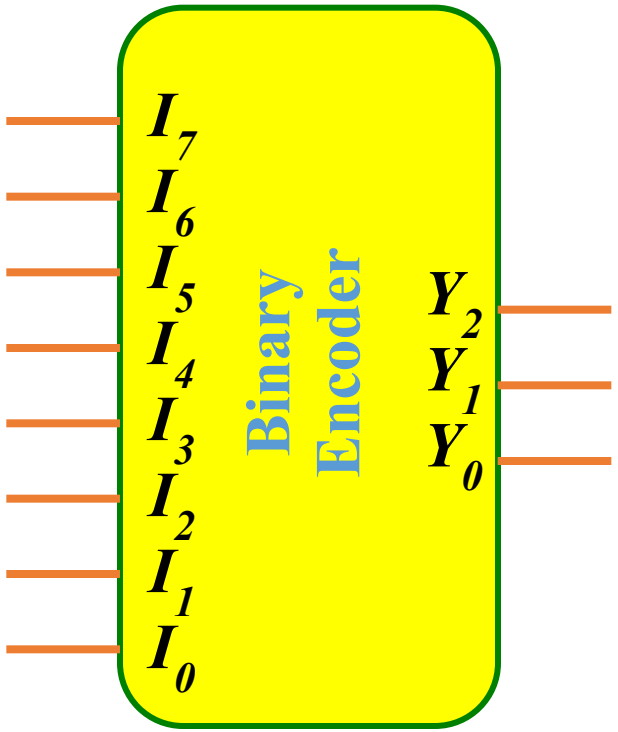
- Octal-to-Binary Encoder (8-to-3)

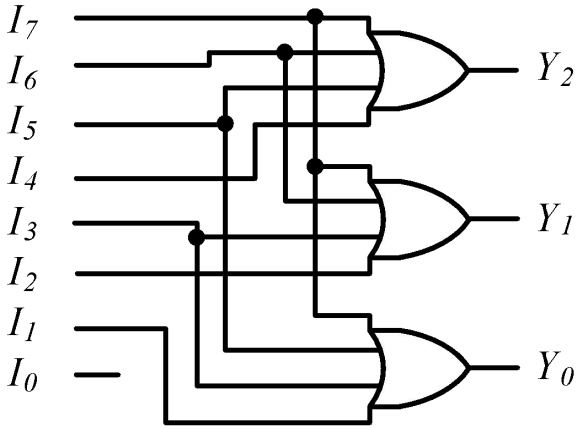
I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

$$Y_2 = I_7 + I_6 + I_5 + I_4$$

$$Y_1 = I_7 + I_6 + I_3 + I_2$$

$$Y_0 = I_7 + I_5 + I_3 + I_1$$

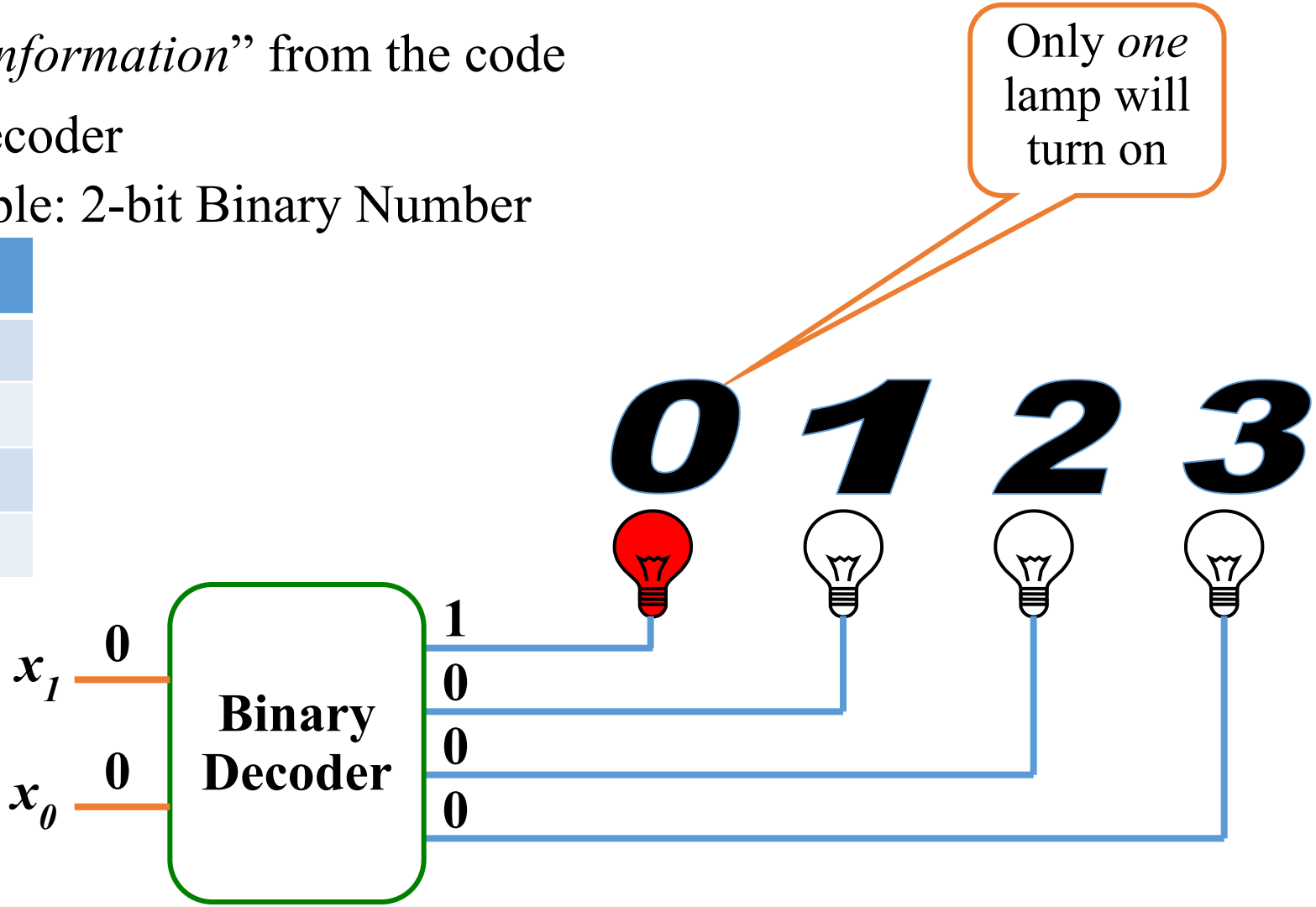




Decoders

- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

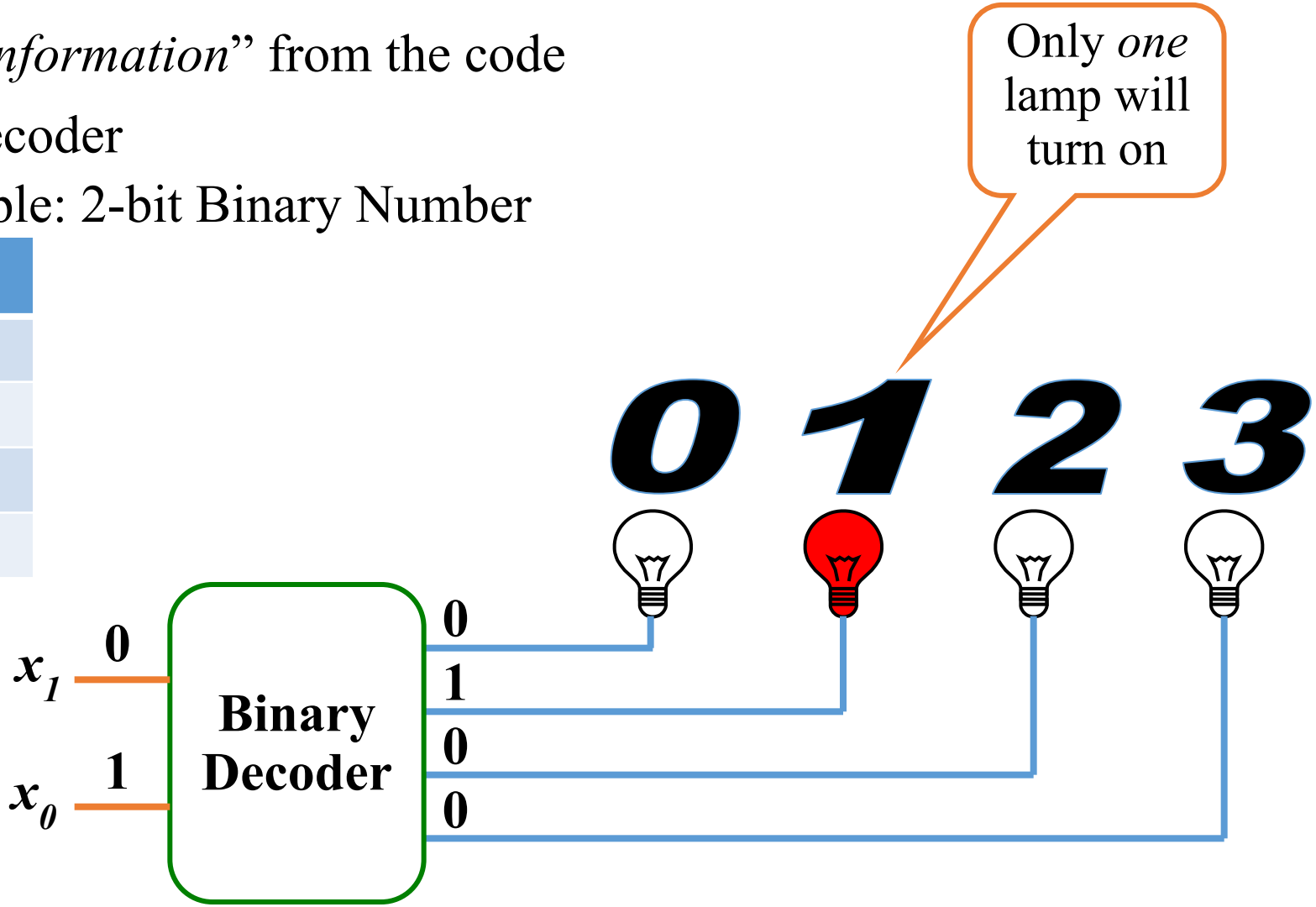
x_1	x_0	O/P
0	0	0



Decoders

- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

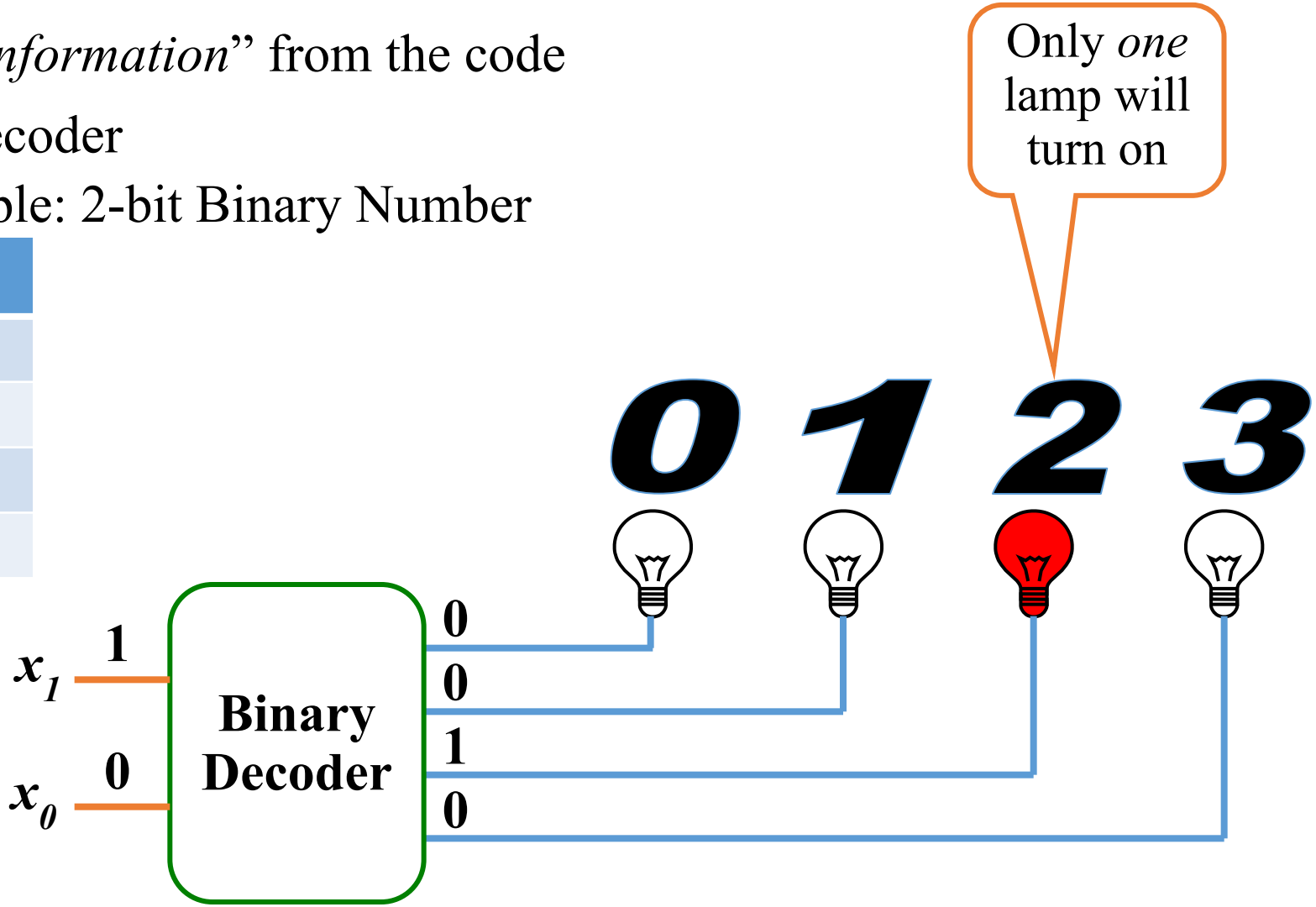
x_1	x_0	O/P
0	0	0
0	1	1



Decoders

- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

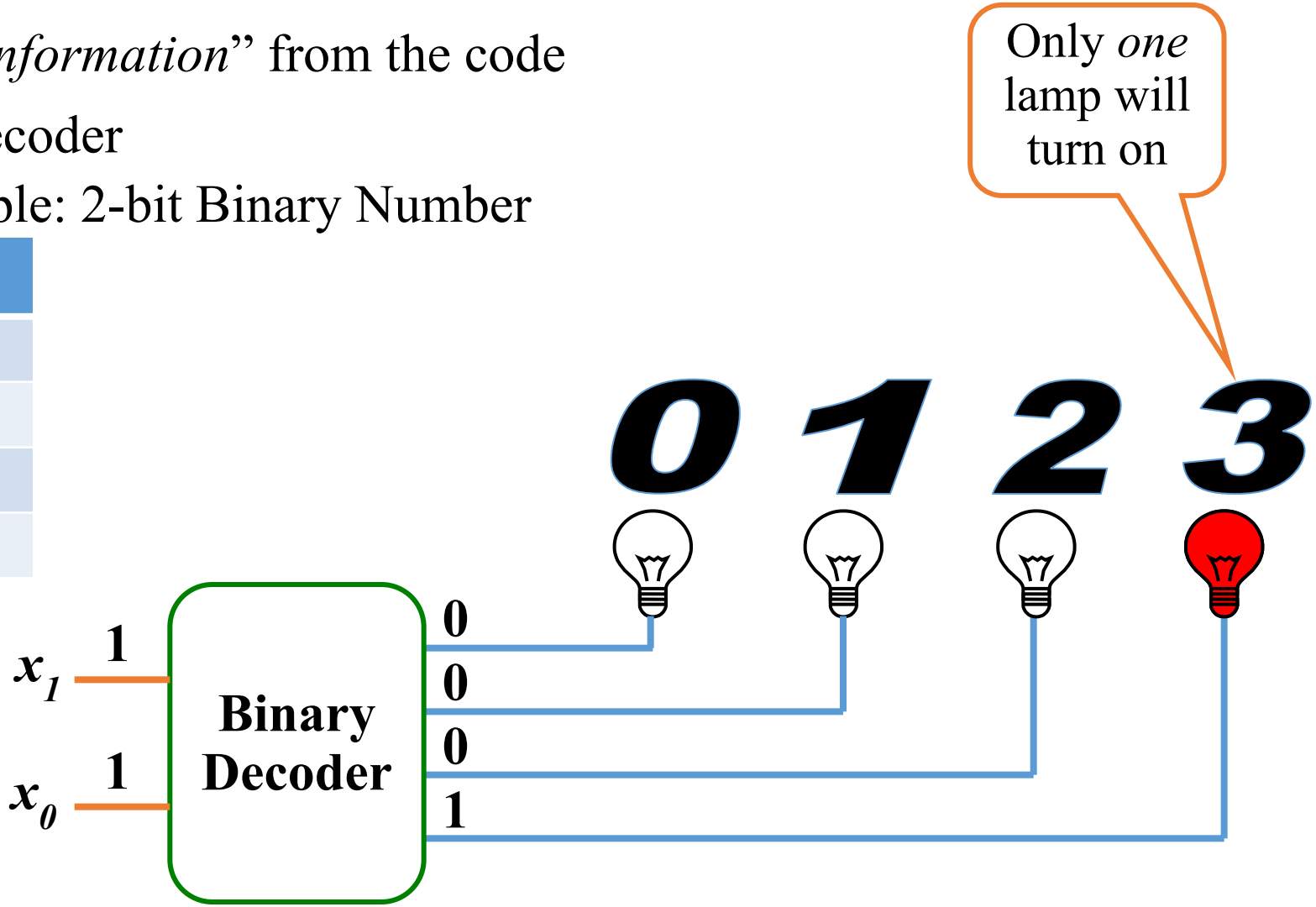
x_1	x_0	O/P
0	0	0
0	1	1
1	0	2



Decoders

- Extract "*Information*" from the code
- Binary Decoder
  - Example: 2-bit Binary Number

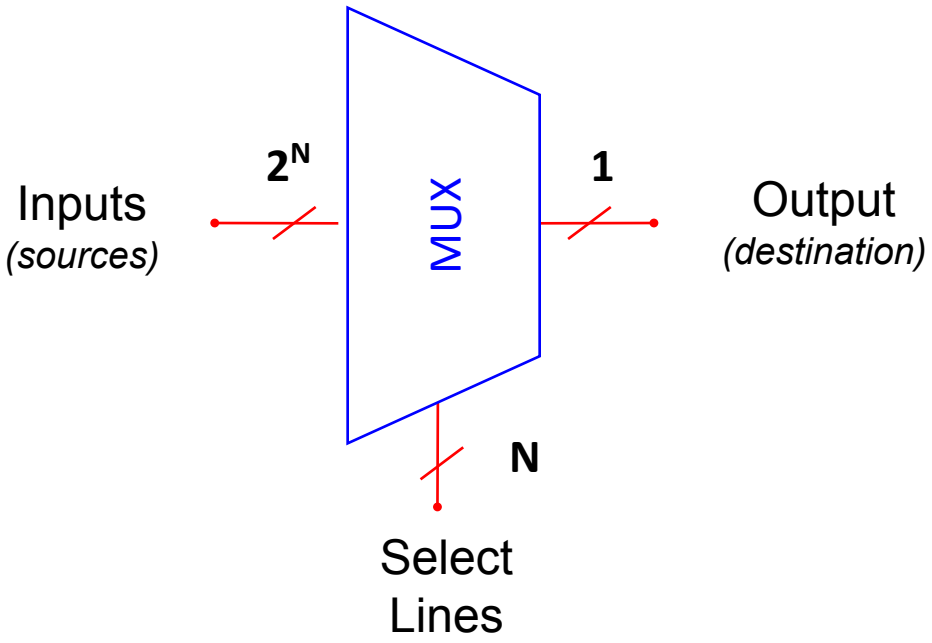
x_1	x_0	O/P
0	0	0
0	1	1
1	0	2
1	1	3



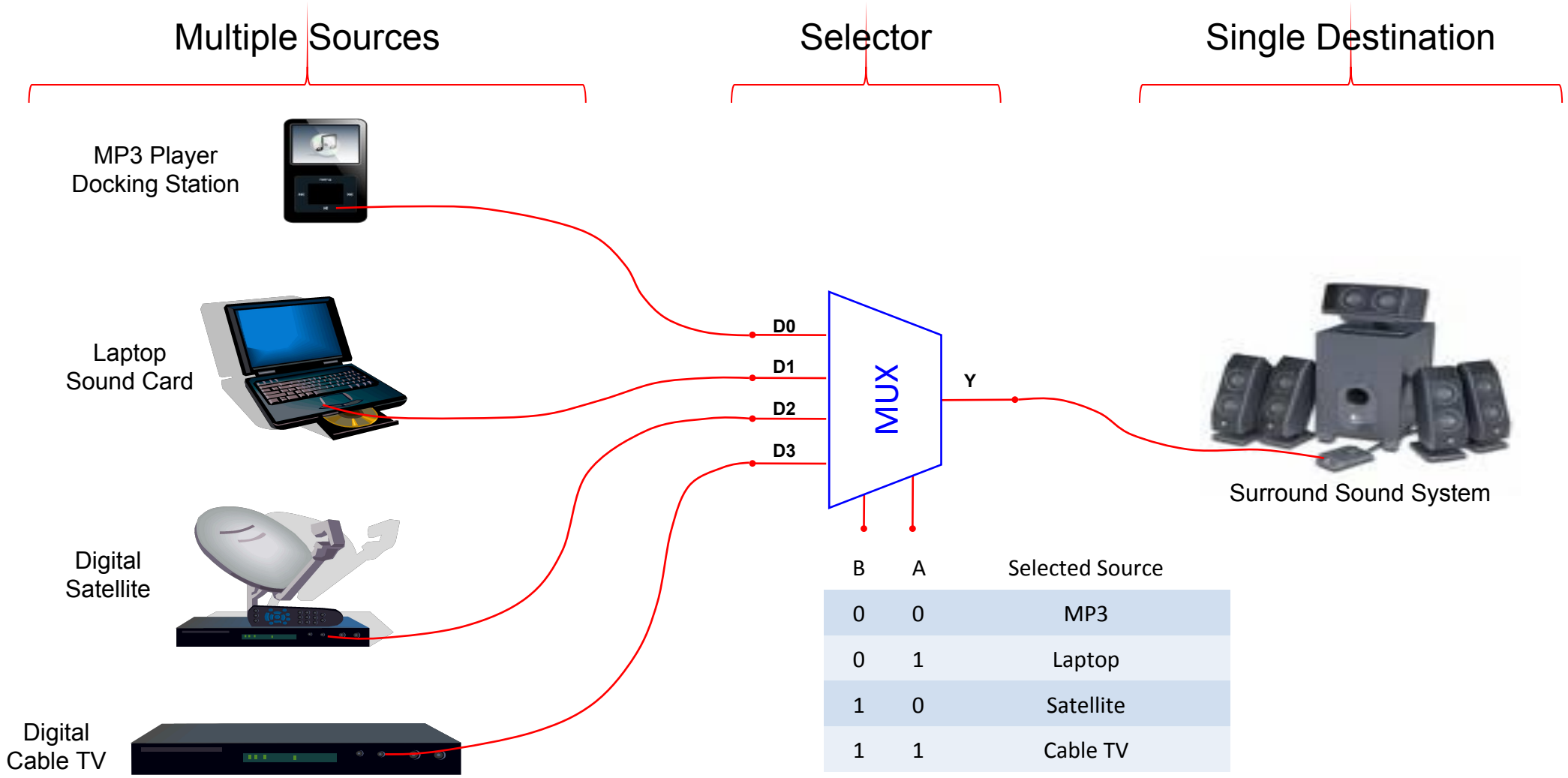
What is a Multiplexer (MUX)?

- A MUX is a digital switch that has multiple inputs (sources) and a single output (destination).
- The select lines determine which input is connected to the output.
- MUX Types
  - 2-to-1 (1 select line)
  - 4-to-1 (2 select lines)
  - 8-to-1 (3 select lines)
  - 16-to-1 (4 select lines)

Multiplexer
Block Diagram

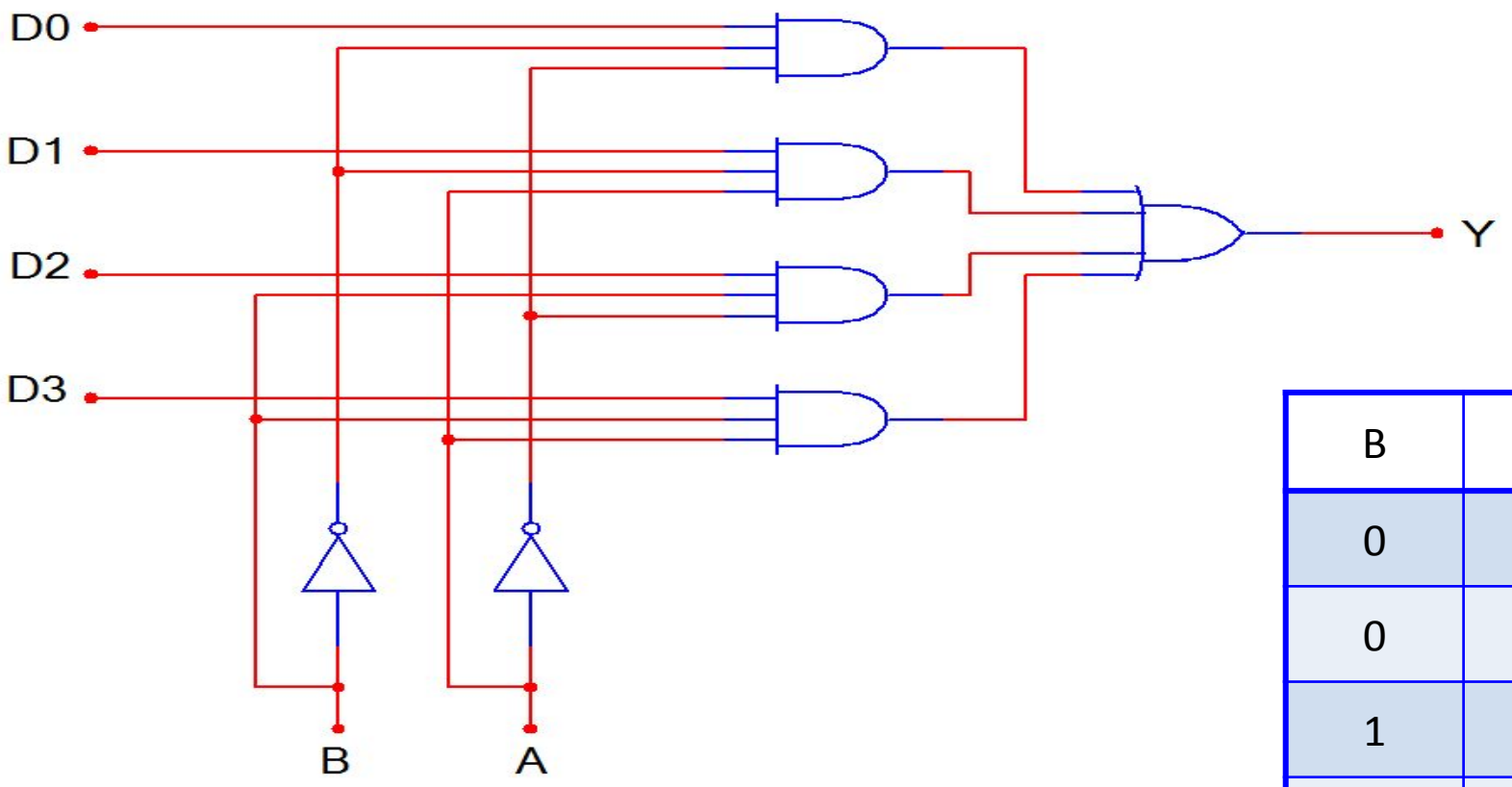


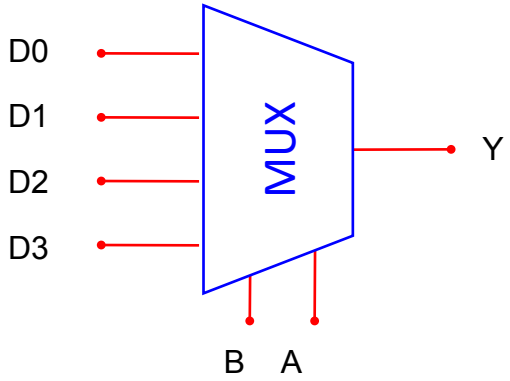
Typical Application of a MUX



B	A	Selected Source
0	0	MP3
0	1	Laptop
1	0	Satellite
1	1	Cable TV

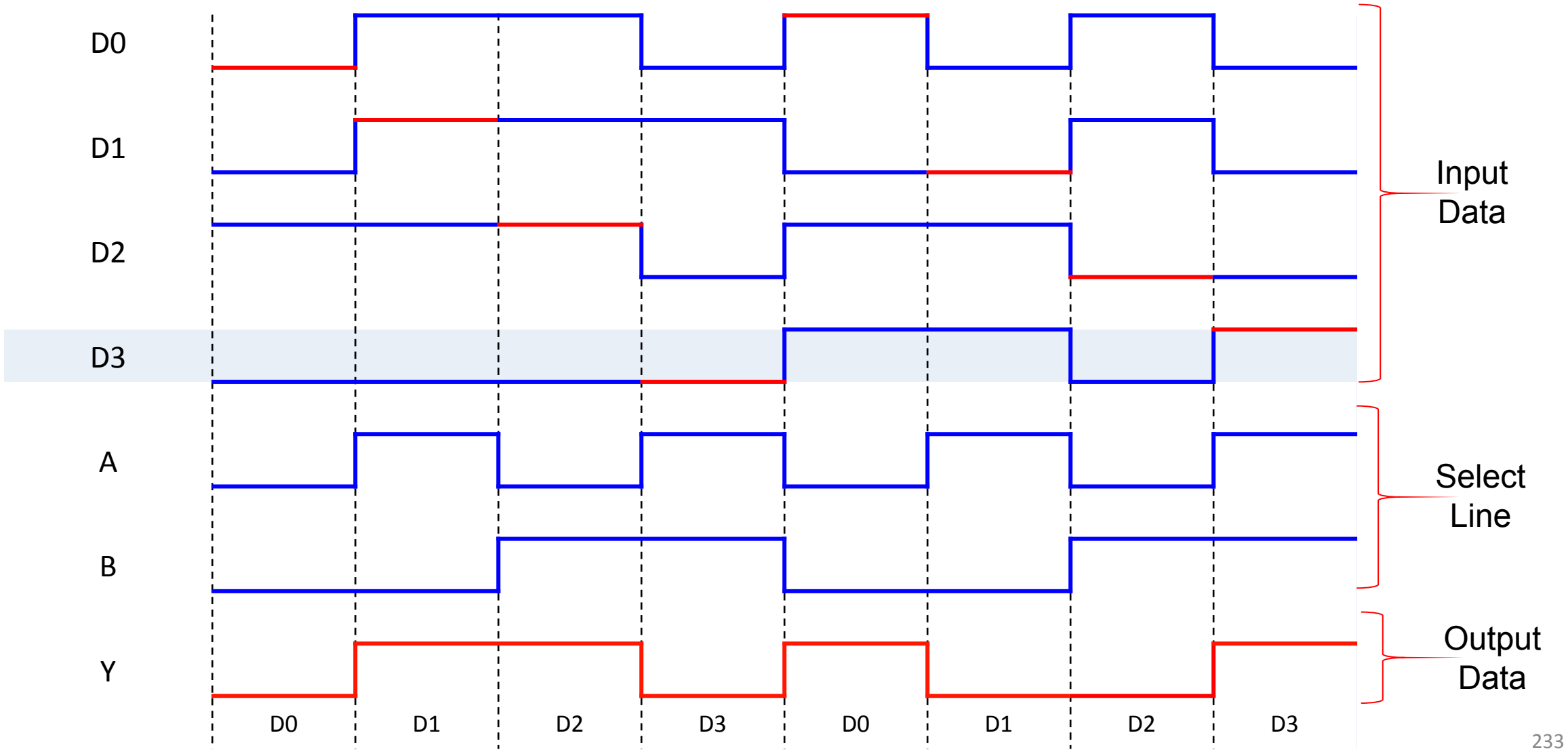
4-to-1 Multiplexer (MUX)





B	A	Y
0	0	D0
0	1	D1
1	0	D2
1	1	D3

4-to-1 Multiplexer Waveforms



Implementation of Boolean function using multiplexer

❑ Type #1

- ❖ Let us solve some problems on implementing the Boolean expressions using a multiplexer.
- ❖ In this method, 3 variables are given(say P, Q, R), which are the selection inputs for the mux.
- ❖ For three selection inputs, the mux to be built was $2^n = 2^3 = 8 : 1$.
- ❖ So, in this method, the **type of mux can be decided by the given number of variables.**

❑ <u>How to solve?</u>

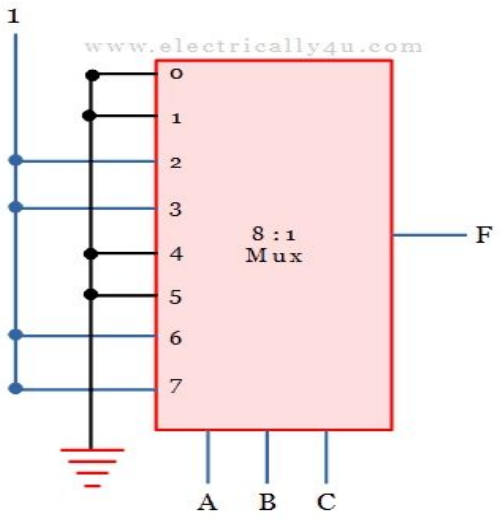
- ❖ The first step is to select the multiplexer. If the given expression has n variables, then determine the multiplexer using the formula $2^n : 1$.
- ❖ Connect the inputs, that correspond to the given minterms to logic 1.
- ❖ Connect all the other inputs to the ground(logic 0).
- ❖ Connect the input variables(P, Q, R) as the selection lines.

❑ Solved example #1

- ❖ **Implement the Boolean expression $F(A, B, C) = \sum m(2, 3, 6, 7)$ using a multiplexer.**

❑ Solution:

- There are 3 variables in the given expression, hence $2^n = 2^3 = 8 : 1$ multiplexer.
- So, the mux has 8 input lines, 3 selection lines, and one output.
- The inputs, corresponding to the minterms (2, 3, 6, 7) are connected to the logic 1 and remaining terms to the logic 0(grounded).
- The given input variables are connected as three selection lines.

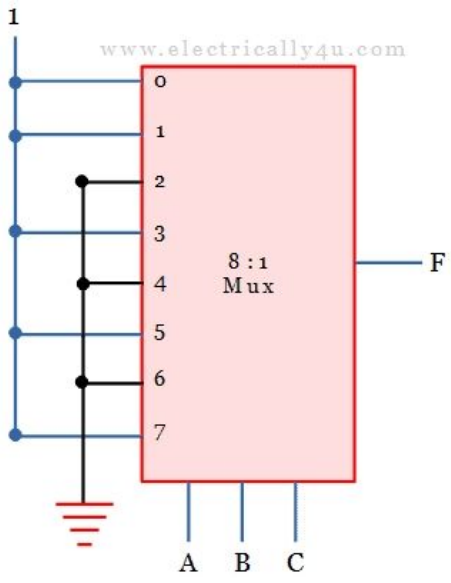


❑ Example #2

Implement the Boolean expression F(A, B, C) = ∑ m(0, 1, 3, 5, 7) using a multiplexer.

❑ Solution:

- Similar to the above problem, there are 3 variables and hence 8 : 1 multiplier is used to solve the expression.
- The three input variables(A, B, C) are connected as three selection lines.
- The inputs, corresponding to the minterms (0, 1, 3, 5, 7) are connected to the logic 1 and remaining terms to the logic 0(grounded).



Implementation of Boolean Function in Multiplexer

❑ Type #2

- If the boolean expression has n+1 variables, we take only n variables as the selection line of the multiplexer.
- The remaining single variable is used as the inputs of the mux.
- In this way, any Boolean expression having n variables can be implemented with 2^{n-1} to 1 multiplexer.

❑ <u>How to solve?</u>

- From the given set of n+1 input variables, the n least significant variables are used as selection line inputs.
- The 2^n inputs for 2^n : 1 multiplexer are derived by using an implementation table.
- The implementation table has all the inputs(D_0, D_1, D_2, D_3,…) for the multiplexer, under which, all the minterms are listed in two rows.
- The first row consists of all minters where A is complemented and the second row has the remaining minterms where A is in uncomplemented form.
- The minterms in the given Boolean expression alone are circled.

Look at the example of implementation table here,

		D_0	D_1	D_2	D_3
Row 1	$\bar{A}$	0	1	2	3
Row 2	A	4	5	6	7
		A	$\bar{A}$	0	1

- Each column is analyzed separately as follows:
- If both the minterms in the two rows are circled, 1 is applied to the corresponding multiplexer input.*(check 4th column in the above table)*
- If both the minterms in the two rows are not circled, 0 is applied to the corresponding multiplexer input.*(check 3rd column in the above table)*
- If the minterm in the first row alone is circled, A is applied to the corresponding multiplexer input.*(check 2nd column in the above table)*
- If the minterm in the second row alone is circled, $\bar{A}$ is applied to the corresponding multiplexer input.*(check 1st column in the above table)*

Solved example #3 (Type#2)

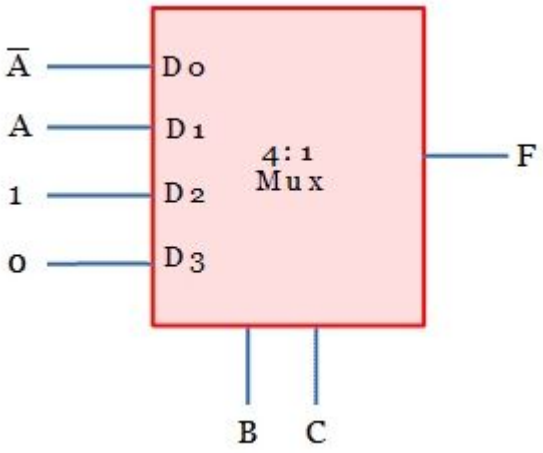
- **Implement the boolean expression F(A, B, C) = ∑ m(0, 2, 5, 6) using 4 : 1 multiplexer.**

Solution:

- In the given Boolean expression, there are 3 variables.
- We should use $2^3 : 1 = 8 : 1$ multiplexer. But as per the question, it is to be implemented with 4 : 1 mux.
- For 4 : 1 multiplexer, there should be 2 selection lines.
- So from the given 3 variables, the 2 least significant variables(B, C) are used as selection line inputs.

- Let us derive the four inputs of 4 : 1 multiplexer using the implementation table.
- The four inputs are listed in column-wise and all the minterms are written under the four inputs in 2 rows as shown below.

		D_0	D_1	D_2	D_3
Row 1	$\overline{A}$	0	1	2	3
Row 2	A	4	5	6	7
		$\overline{A}$	A	1	0



- The minterms given in the boolean expression is circled and analyzed.
- After analyzing, the input values of 4 : 1 mux is obtained as $\overline{A}$, A, 1, 0.
- Thus the circuit can be drawn as below.

❑ Solved example #4 (Type#2)

❖ **Implement F(A, B, C, D) = ∑ m(0, 1, 5, 6, 8, 10, 12, 15) using 8 : 1 multiplexer.**

❖ Solution:

❖ In the given boolean expression, there are 4 variables.

❖ We should use $2^4 : 1 = 16 : 1$ multiplexer.

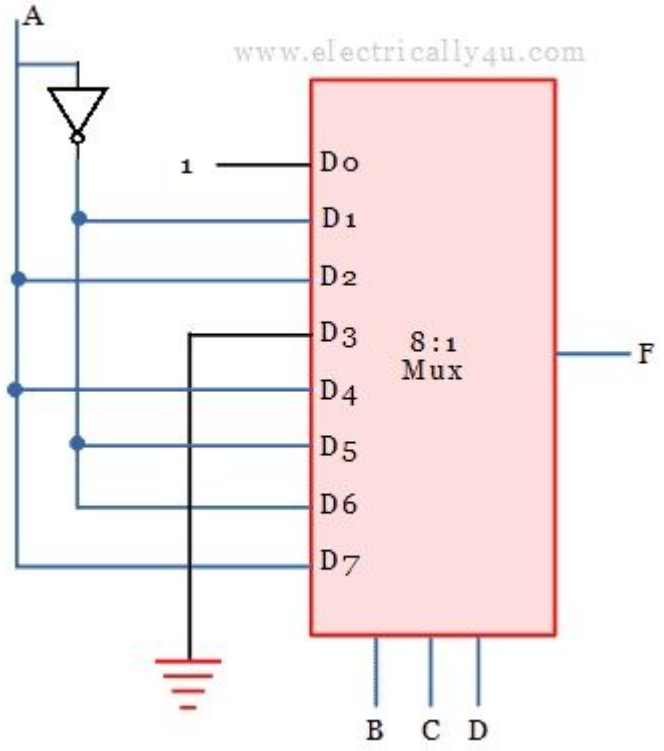
❖ But as per the question, it is to be implemented with 8 : 1 mux.

❖ For 8 : 1 multiplexer, there should be 3 selection lines.

❖ So from the given 4 variables, the 3 least significant variables(B, C, D) are used as selection line inputs.

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	1	$\bar{A}$	A	0	A	$\bar{A}$	$\bar{A}$	A

From the derived input, 8 : 1 multiplexer can be drawn as belo

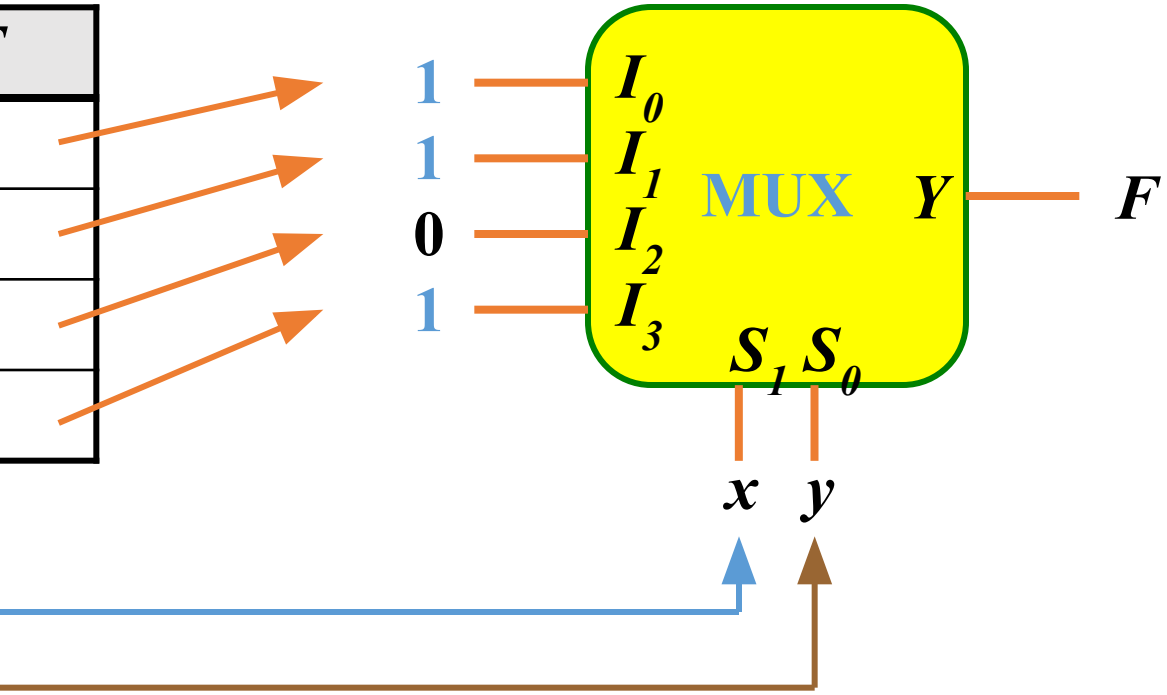


More Example: Implementation Using Multiplexers

Implementation Using Multiplexers

- Example $F(x, y) = \sum(0, 1, 3)$

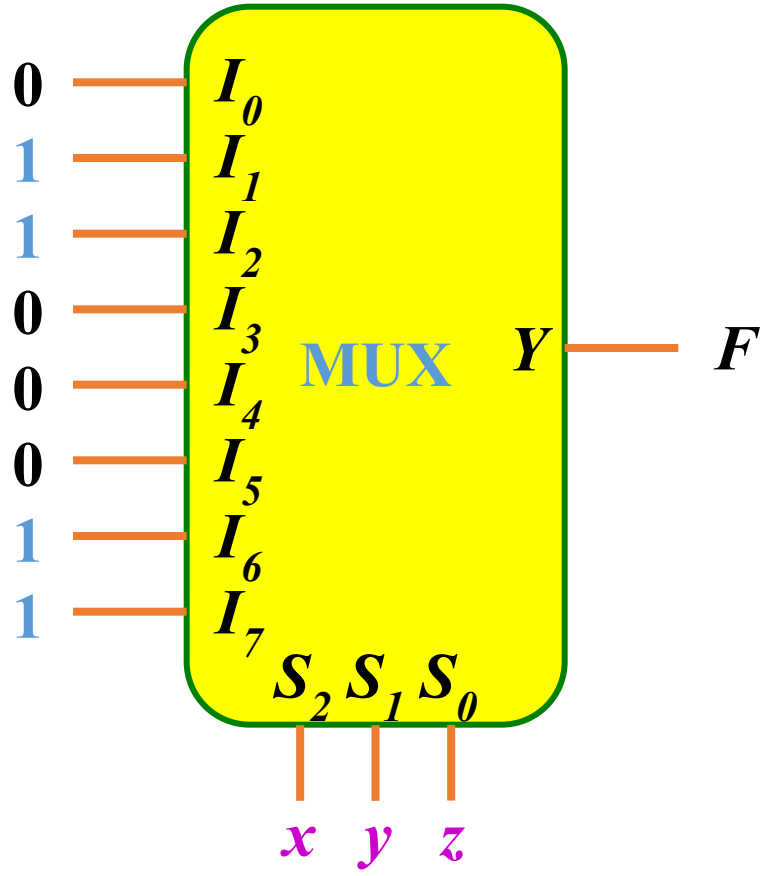
x	y	F
0	0	1
0	1	1
1	0	0
1	1	1



Implementation Using Multiplexers

- $F(x, y, z) = \sum(1, 2, 6, 7)$

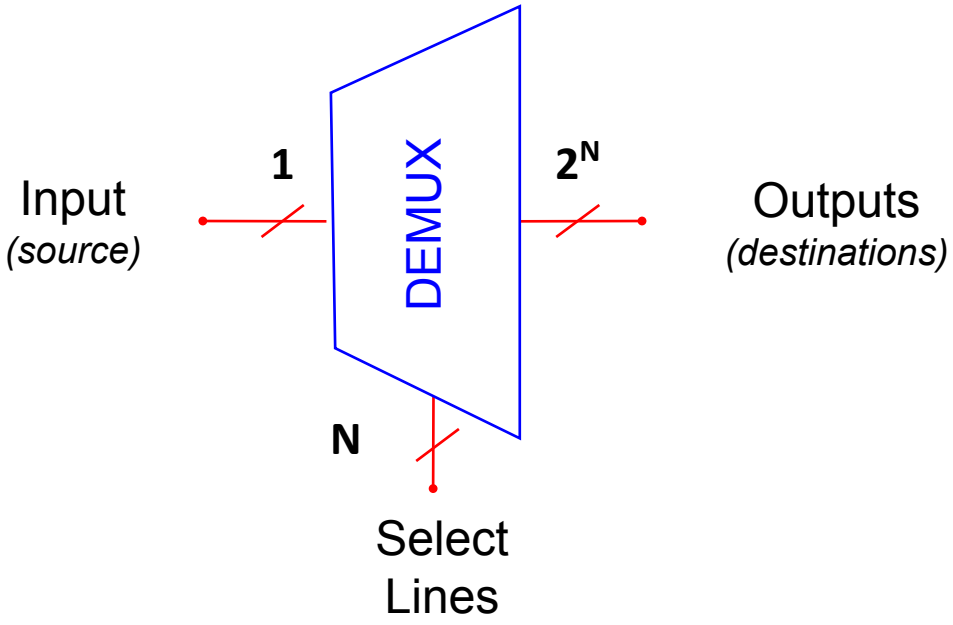
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1



De-multiplexer (DEMUX)

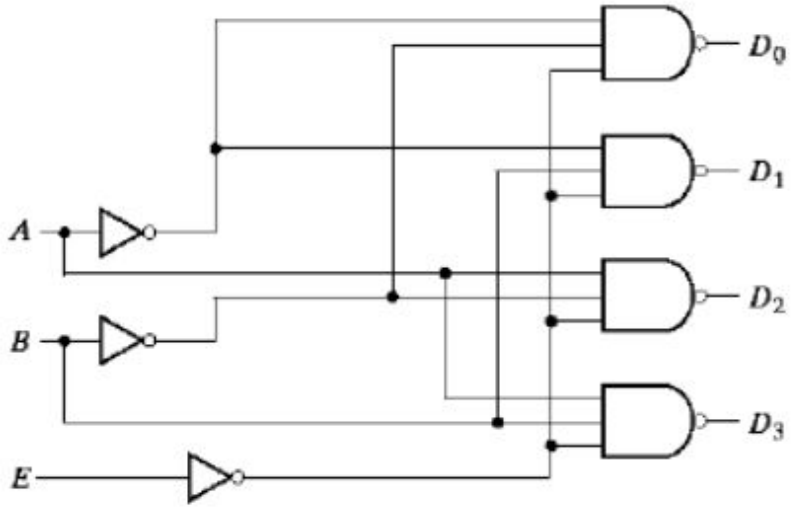
- A DEMUX is a digital switch with a single input (source) and a multiple outputs (destinations).
- The select lines determine which output the input is connected to.
- DEMUX Types
  - 1-to-2 (1 select line)
  - 1-to-4 (2 select lines)
  - 1-to-8 (3 select lines)
  - 1-to-16 (4 select lines)

Demultiplexer
Block Diagram



De-multiplexer

- A circuit that receives information from a single line and directs it to one of 2^n possible output lines
- A decoder with enable input can function as a demultiplexer
  - Often referred to as a ***decoder/demultiplexer***

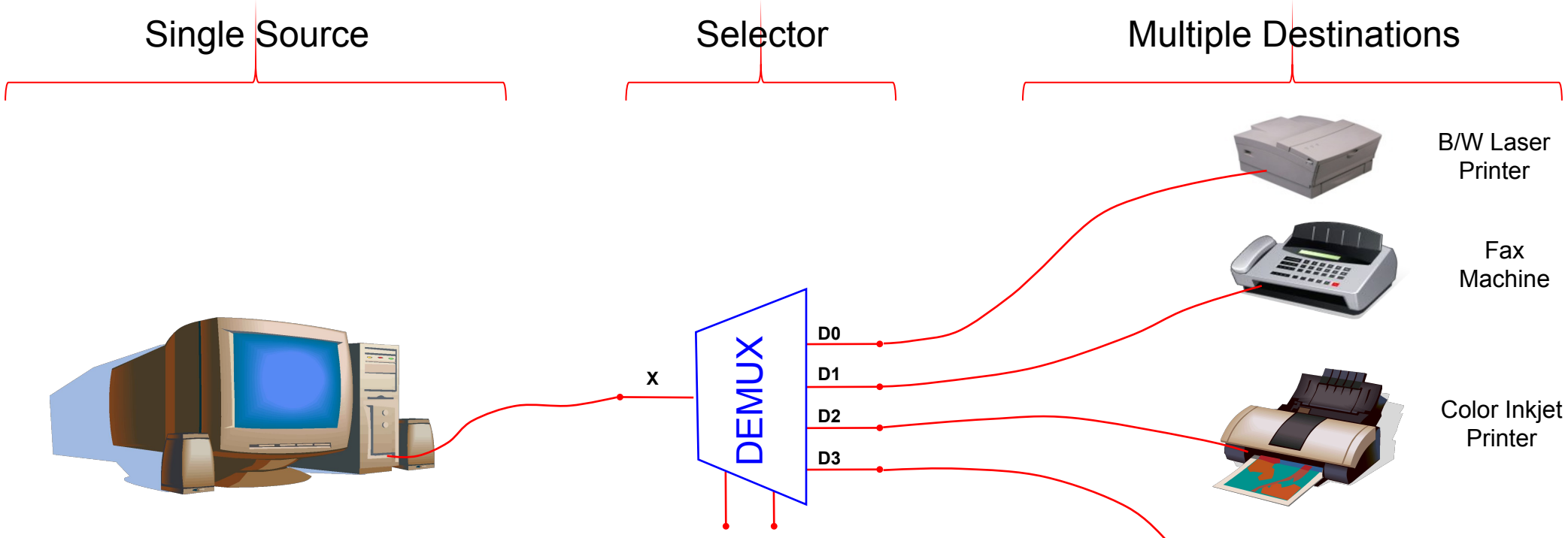


data input

selection

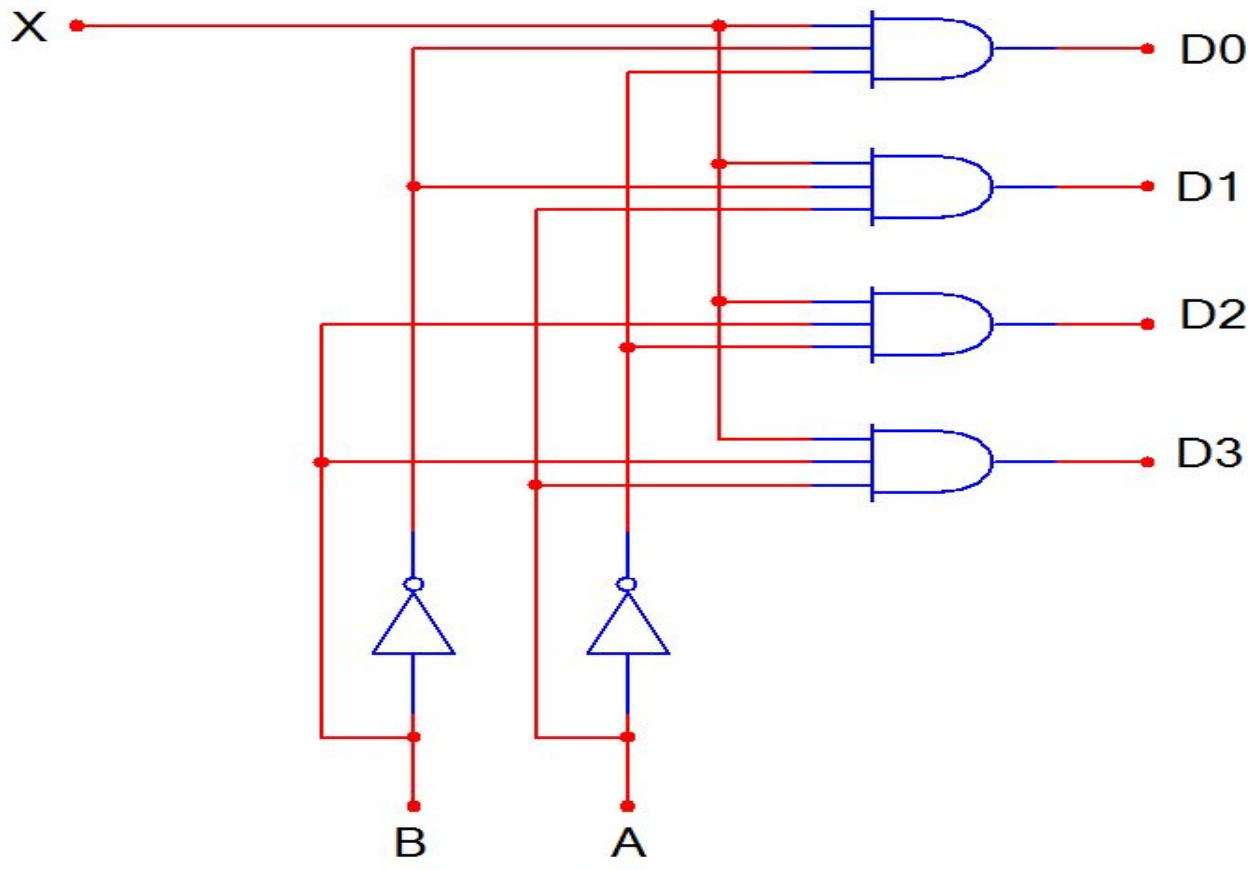
E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

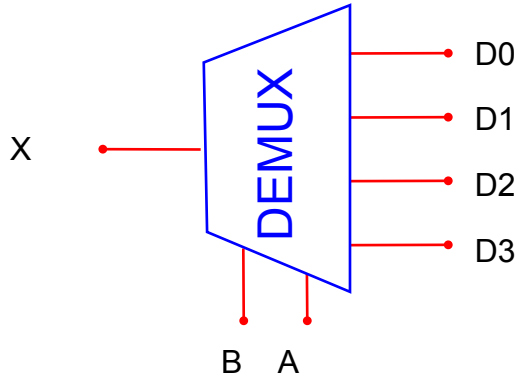
Typical Application of a DEMUX



B	A	Selected Destination
0	0	B/W Laser Printer
0	1	Fax Machine
1	0	Color Inkjet Printer
1	1	Pen Plotter

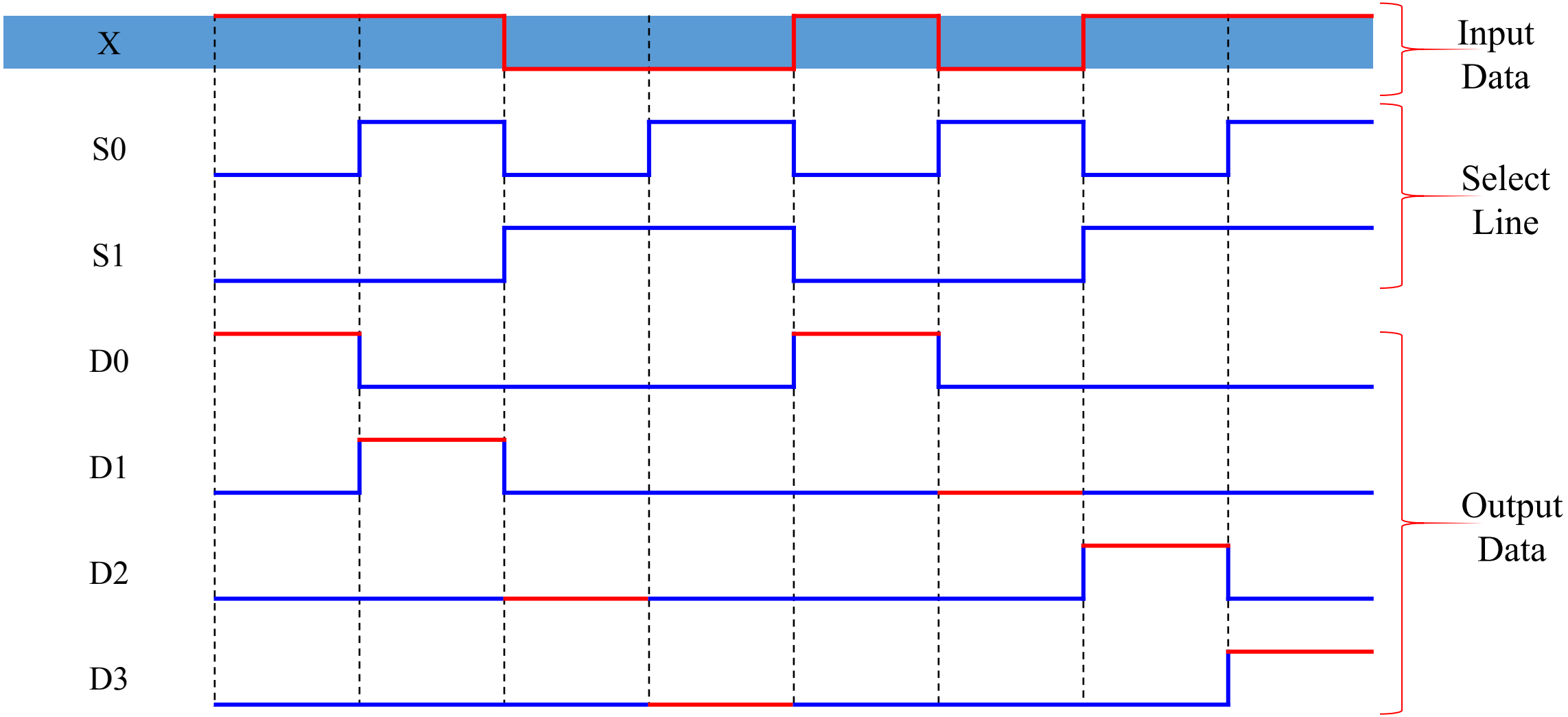
1-to-4 De-Multiplexer (DEMUX)



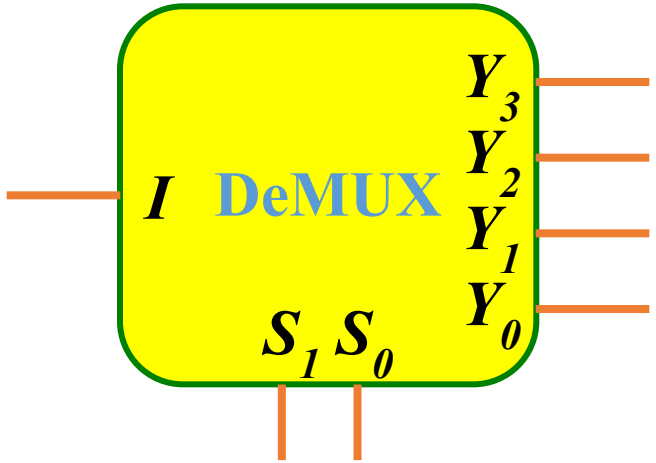


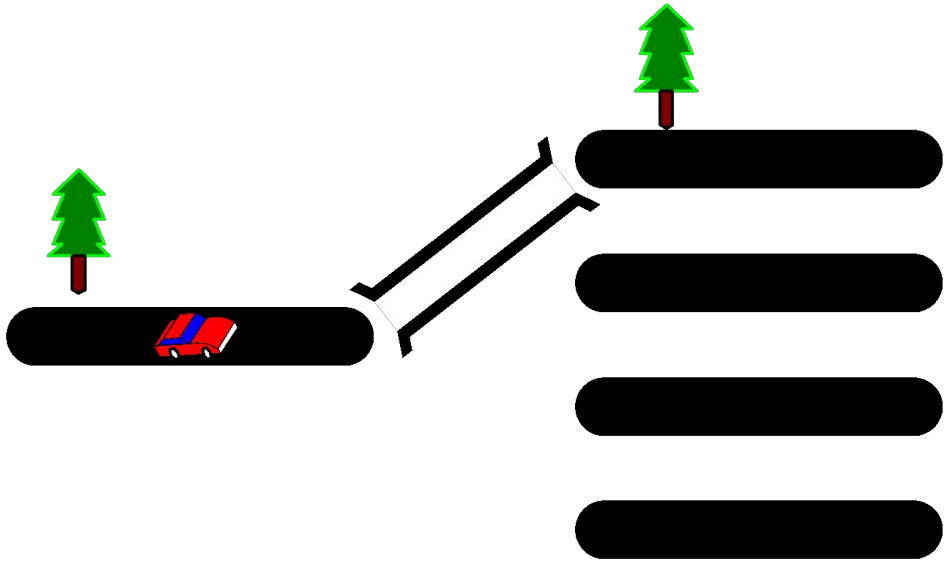
B	A	D0	D1	D2	D3
0	0	X	0	0	0
0	1	0	X	0	0
1	0	0	0	X	0
1	1	0	0	0	X

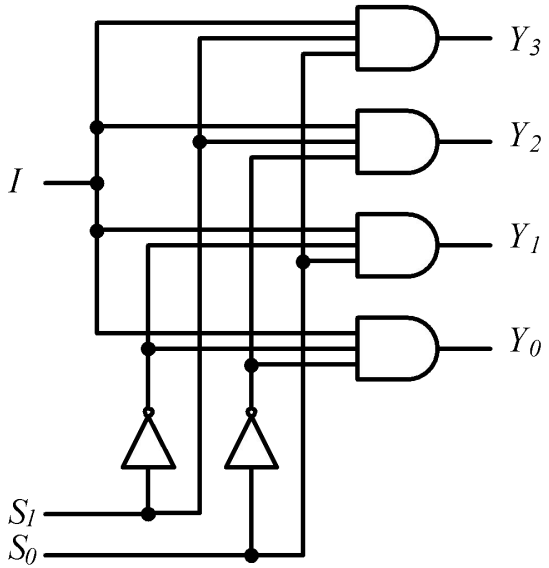
1-to-4 De-Multiplexer Waveforms



De-multiplexer







S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

Seven Segment Display

b_3	b_2	b_1	b_0		a	b	c	d	e	f	g
0	0	0	0	(0)	1	1	1	1	1	1	0
0	0	0	1	(1)	0	1	1	0	0	0	0
0	0	1	0	(2)	1	1	0	1	1	0	1
0	0	1	1	(3)	1	1	1	1	0	0	1
0	1	0	0	(4)	0	1	1	0	0	1	1
0	1	0	1	(5)	1	0	1	1	0	1	1
0	1	1	0	(6)	1	0	1	1	1	1	1
0	1	1	1	(7)	1	1	1	0	0	0	0
1	0	0	0	(8)	1	1	1	1	1	1	1
1	0	0	1	(9)	1	1	1	1	0	1	1

$b_3b_2 \backslash b_1b_0$	00	01	11	10
00	1	0	1	1
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

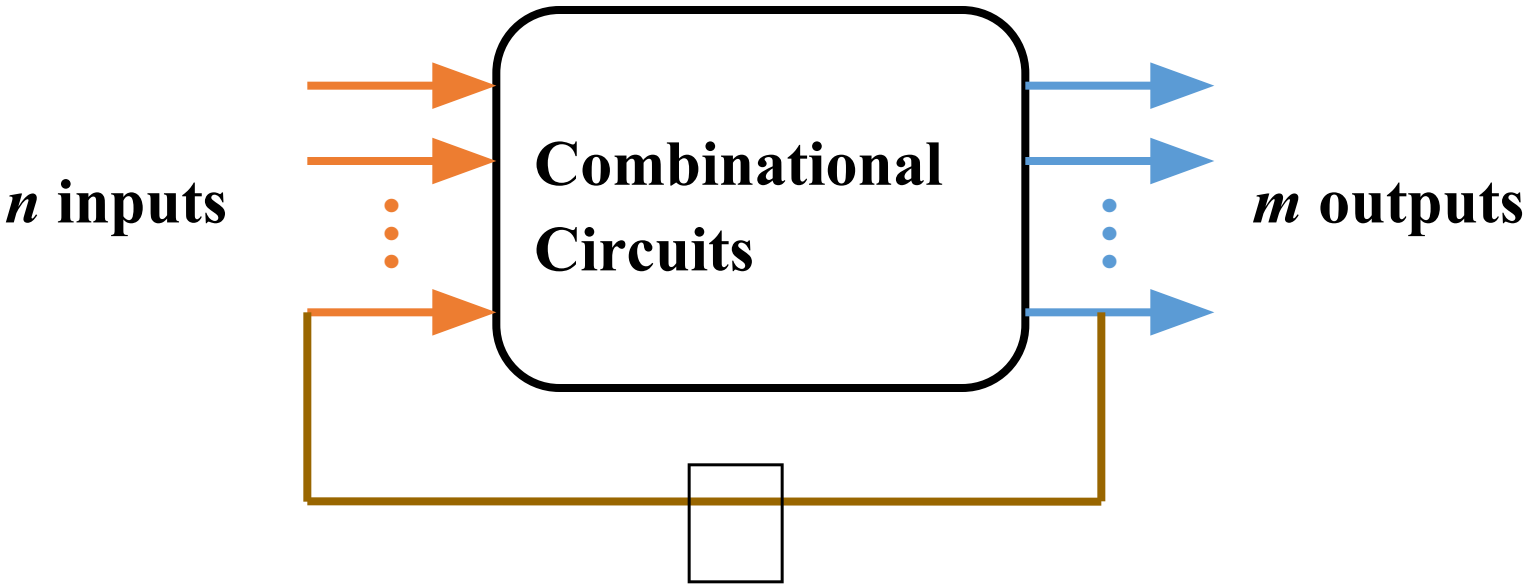
$b_3b_2 \backslash b_1b_0$	00	01	10	
00				
01				
10				

$b_3b_2 \backslash b_1b_0$	00	01	10	
00				
01				
10				

b_3	b_2	b_1	b_0		a	b	c	d	e	f	g
0	0	0	0	(0)	1	1	1	1	1	1	0
0	0	0	1	(1)	0	1	1	0	0	0	0
0	0	1	0	(2)	1	1	0	1	1	0	1
0	0	1	1	(3)	1	1	1	1	0	0	1
0	1	0	0	(4)	0	1	1	0	0	1	1
0	1	0	1	(5)	1	0	1	1	0	1	1
0	1	1	0	(6)	1	0	1	1	1	1	1
0	1	1	1	(7)	1	1	1	0	0	0	0
1	0	0	0	(8)	1	1	1	1	1	1	1
1	0	0	1	(9)	1	1	1	1	0	1	1

Combinational Circuits

- Output is function of input only
  - i.e. no feedback



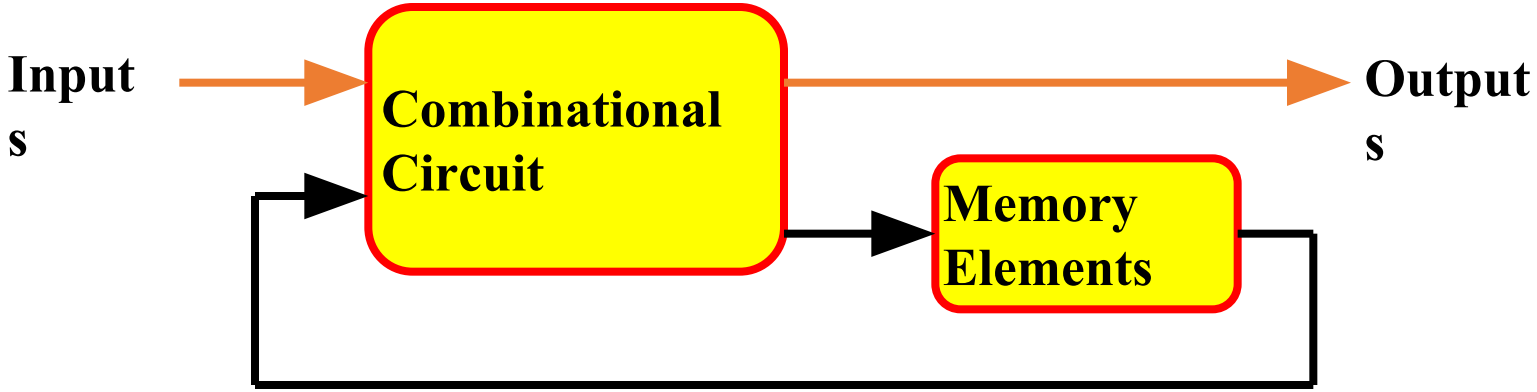
When input changes, output may change (after a delay)

Combinational Circuit Vs. Sequential Circuit

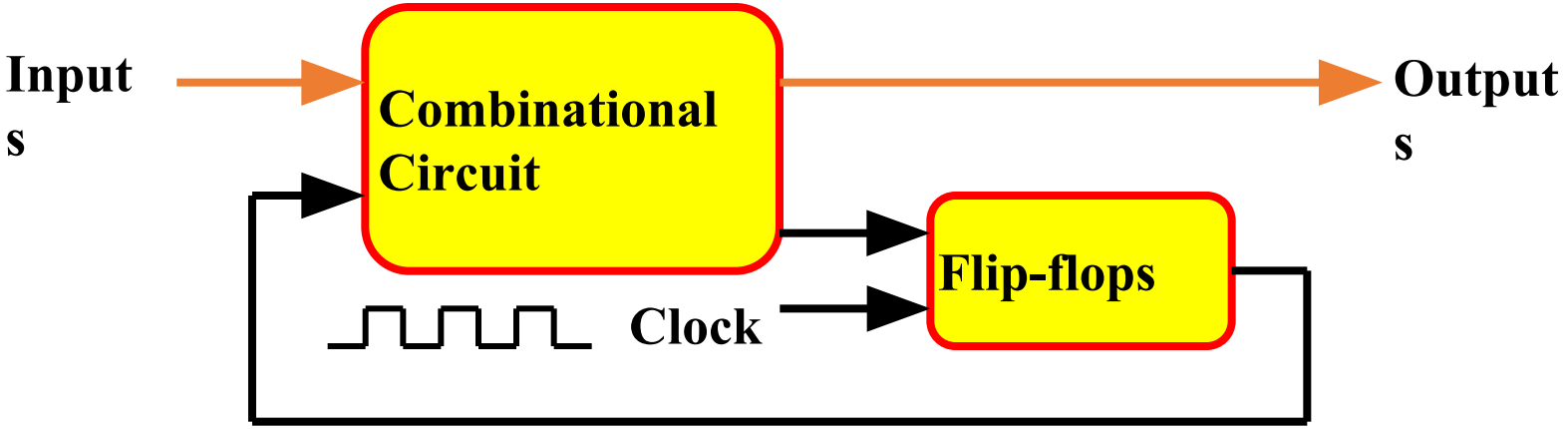
Combinational Circuit	Sequential Circuit
Output depends only upon present input	Output depends upon present as well as past input
Speed is fast.	Speed is slow.
It is designed easy.	It is designed tough as compared to combinational circuits.
There is no feedback between input and output.	There exists a feedback path between input and output.
This is time independent.	This is time dependent.
Elementary building blocks: Logic gates	Elementary building blocks: Flip-flops
Used for arithmetic as well as Boolean operations.	Mainly used for storing data.
Combinational circuits don't have capability to store any state.	Sequential circuits have capability to store any state or to retain earlier state.
As combinational circuits don't have clock, they don't require triggering.	As sequential circuits are clock dependent they need triggering.
These circuits do not have any memory element.	These circuits have memory element.
It is easy to use and handle.	It is not easy to use and handle.
Examples: Encoder, Decoder, and Multiplexer, Demultiplexer	Examples: Flip-flops, and Counters

Sequential Circuits

- Asynchronous

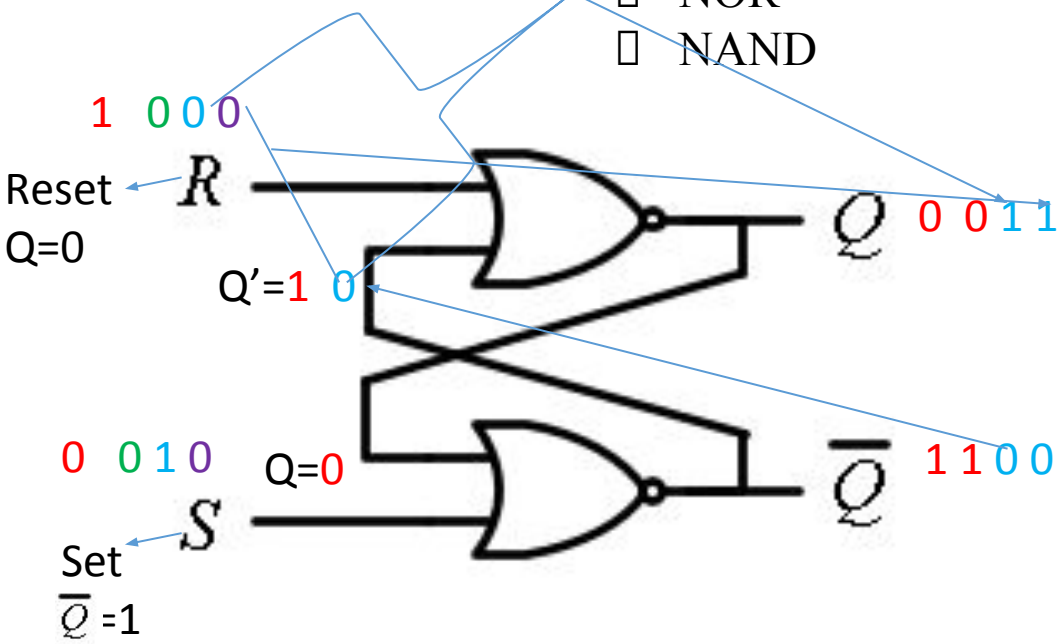


- Synchronous



SR Latch :

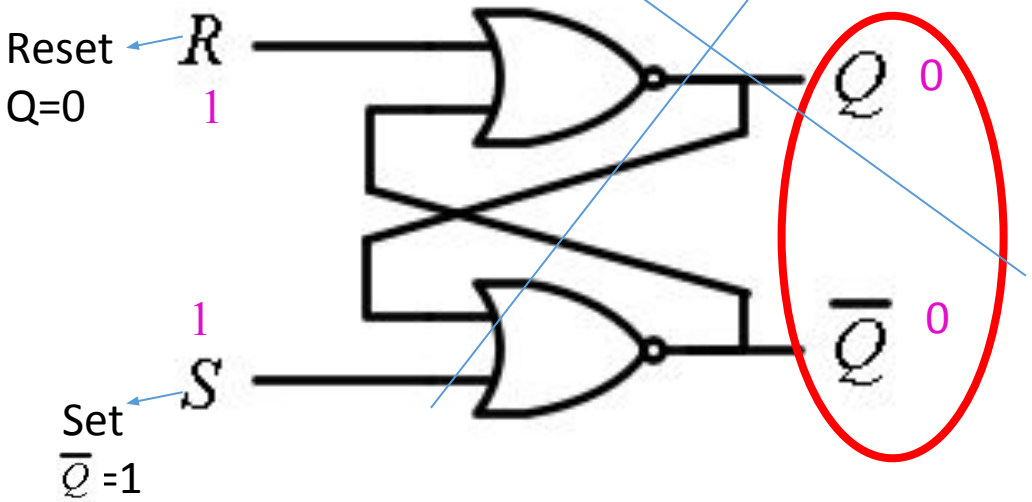
- NOR
- NAND



Case 3:

S=1, R=1 Q=0 Q'=0

S=0, R=0 Q=0 Q'=0



Case 1: S=0, R=1 Q= 0 $\overline{Q}$ =1

S=0, R=0 Q= 0 $\overline{Q}$ =1

No Change: (Memory)

Case 2: S=1, R=0 Q= 1 $\overline{Q}$ =0

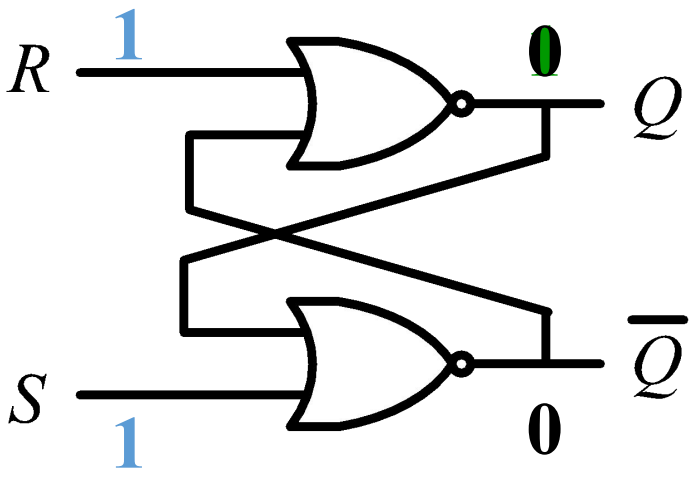
S=0, R=0 Q= 1 $\overline{Q}$ =0

No Change: Memory

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

Latches

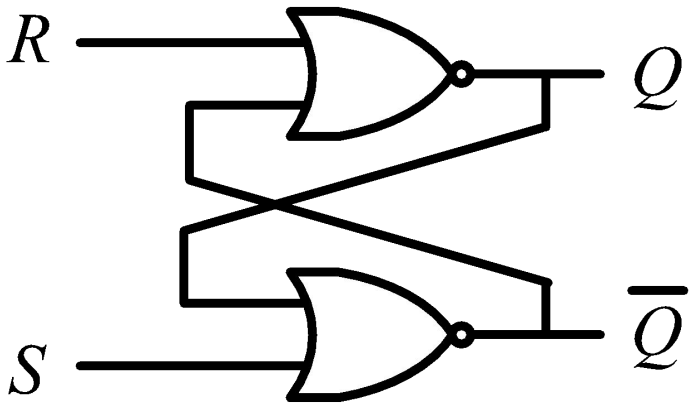
- *SR* Latch



S	R	Q_0	Q	Q'	
0	0	0	0	1	$Q = Q_0$
0	0	1	1	0	
0	1	0	0	1	$Q = 0$
0	1	1	0	1	
1	0	0	1	0	$Q = 1$
1	0	1	1	0	
1	1	0	0	0	$Q = Q'$
1	1	1	0	0	$Q = Q'$

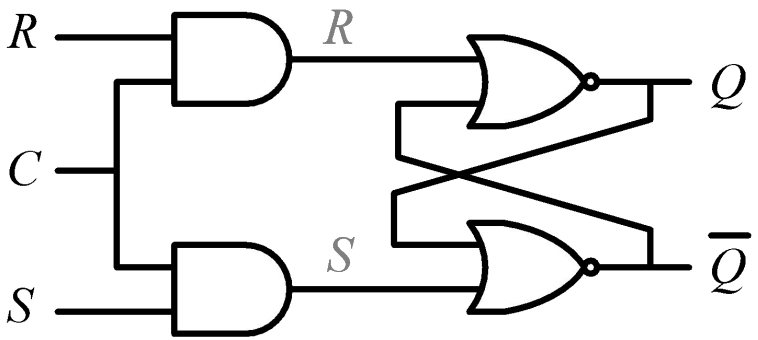
Latches

❖ SR Latch

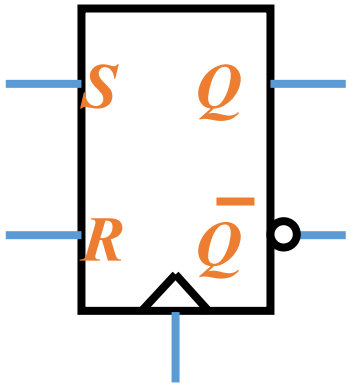


S R	Q	
0 0	Q_0	No change
0 1	0	Reset
1 0	1	Set
1 1	$Q=Q'=0$	Invalid

✔ SR Latch with Control Input

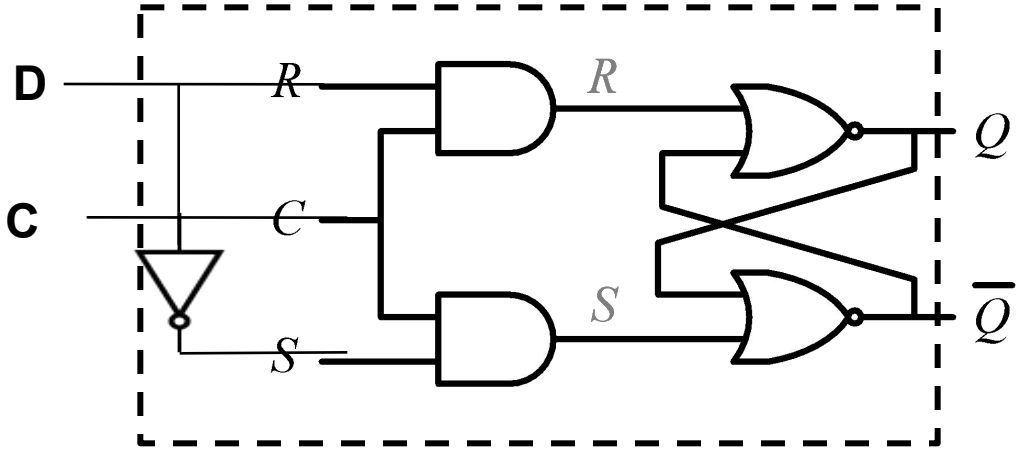


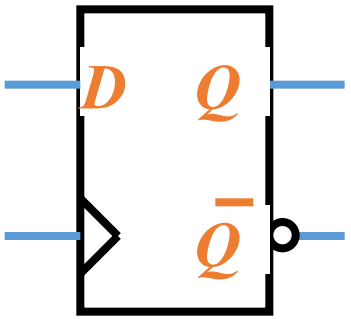
C S R	Q	
0 x x	Q_0	No change
1 0 0	Q_0	No change
1 0 1	0	Reset
1 1 0	1	Set
1 1 1	$Q=Q'$	Invalid



Controlled Latches

- D Latch (D = Data)

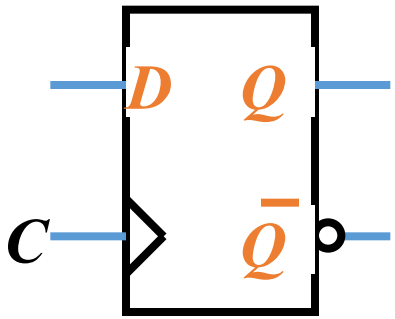




C D	Q	
0 x	Q_0	No change
1 0	0	Reset
1 1	1	Set

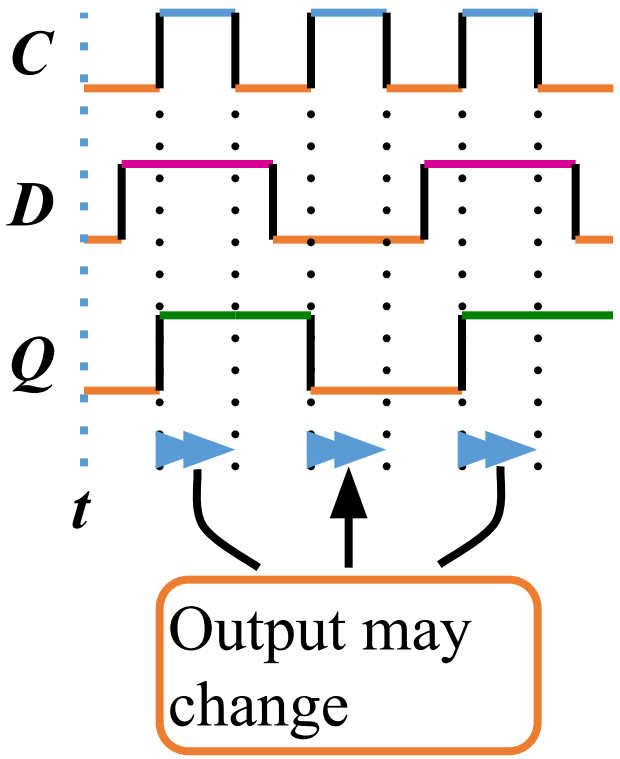
Controlled Latches

- *D* Latch (*D* = *Data*)



C D	Q	
0 x	Q_0	No change
1 0	0	Reset
1 1	1	Set

Timing Diagram

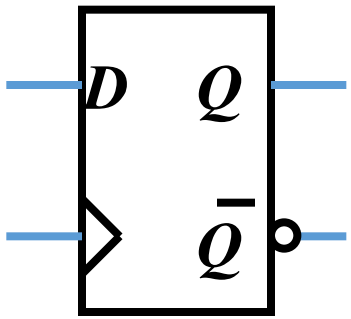


Q (t+1) = D
When Clock is enabled

Q(t+1) = Q(t)
When Clock is disabled

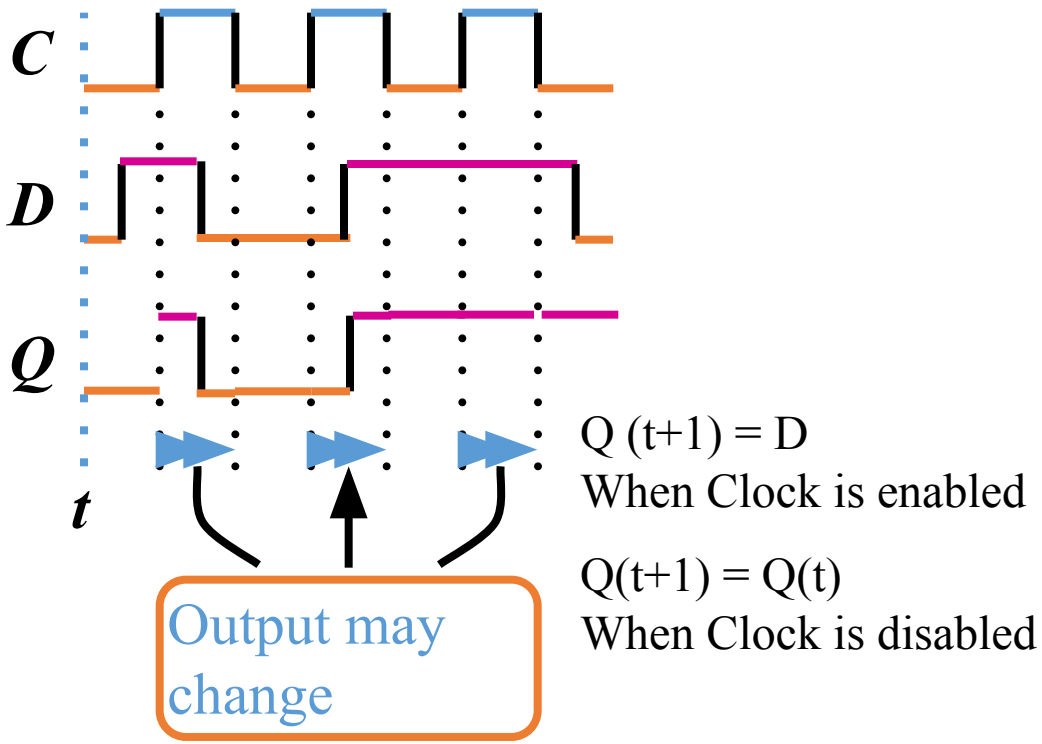
Controlled Latches (Task 1)

❖ *D* Latch (D = Data)



C D	Q	
0 x	Q_0	No change
1 0	0	Reset
1 1	1	Set

Timing Diagram

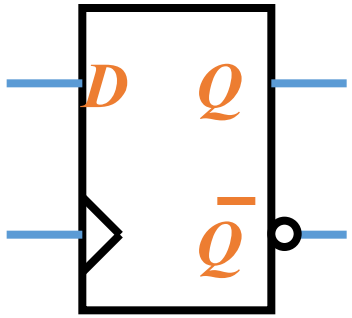


Q (t+1) = D
When Clock is enabled

Q(t+1) = Q(t)
When Clock is disabled

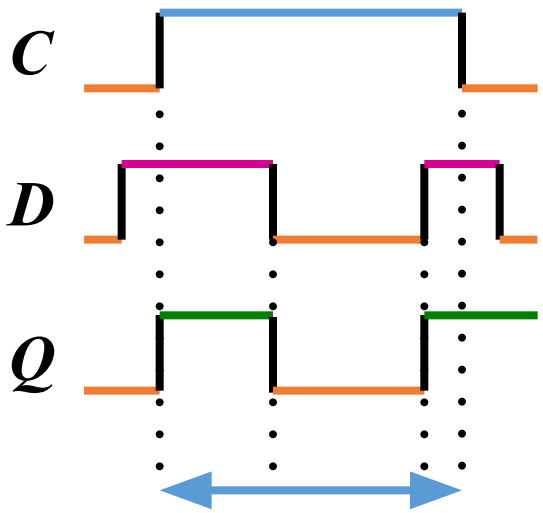
Controlled Latches

- D Latch (D = Data)



C D	Q	
0 x	Q_0	No change
1 0	0	Reset
1 1	1	Set

Timing Diagram



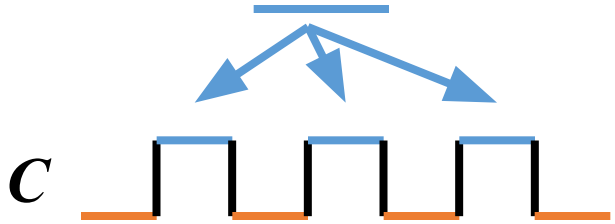
Output may change

Q (t+1) = D
When Clock is enabled

Q(t+1) = Q(t)
When Clock is disabled

Flip-Flops

- Controlled latches are level-triggered



Q (t+1) = D
When Clock is enabled

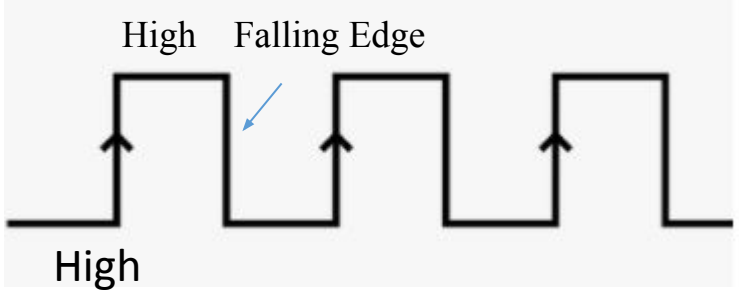
Q(t+1) = Q(t)
When Clock is disabled

- Flip-Flops are edge-triggered

Positive Edge

Negative Edge

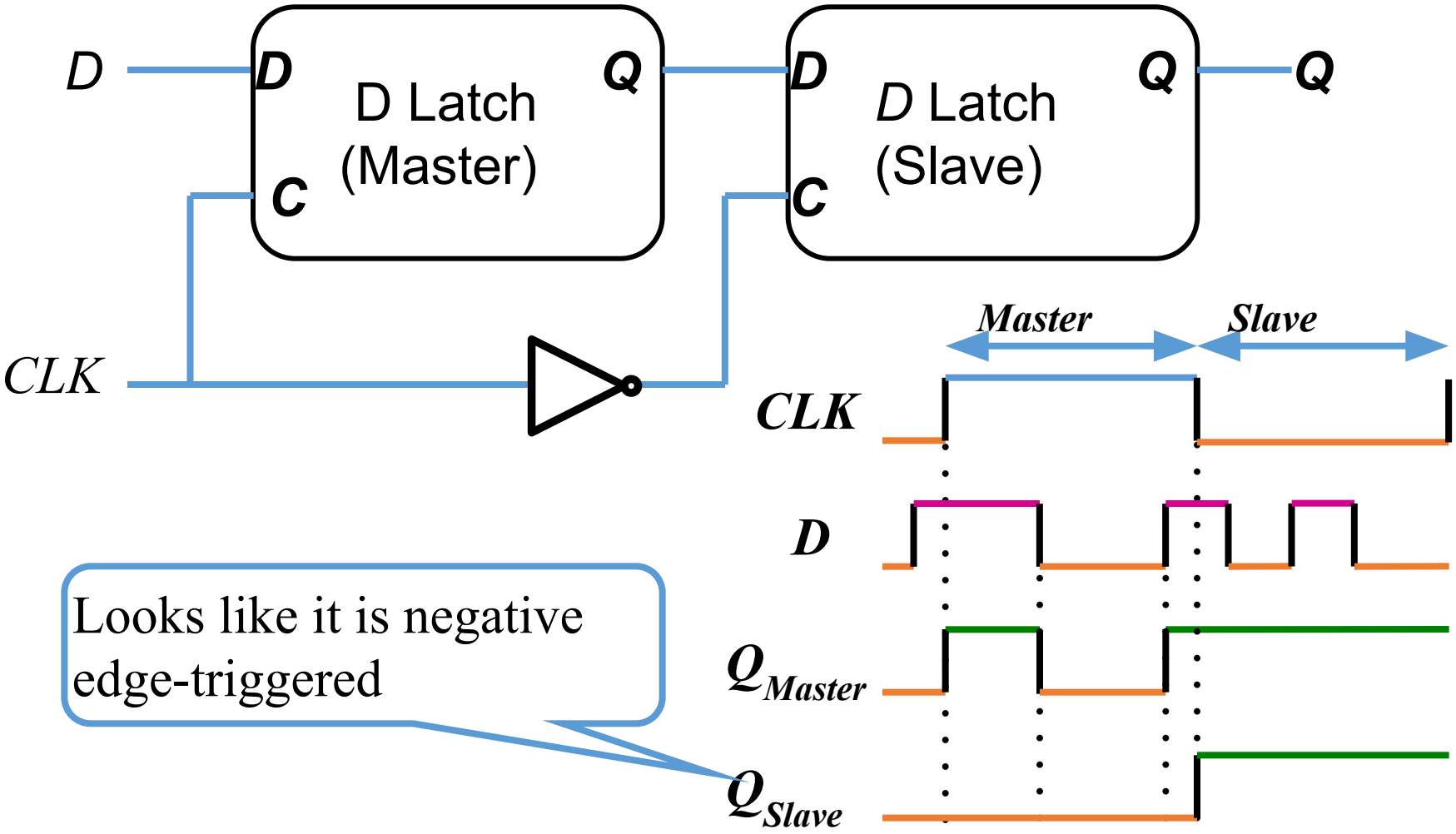
S	R	Q	Q’
0	0	NO change	
0	1	0	
1	0	1	
1	1	Not Used	



High Falling Edge

Flip-Flops

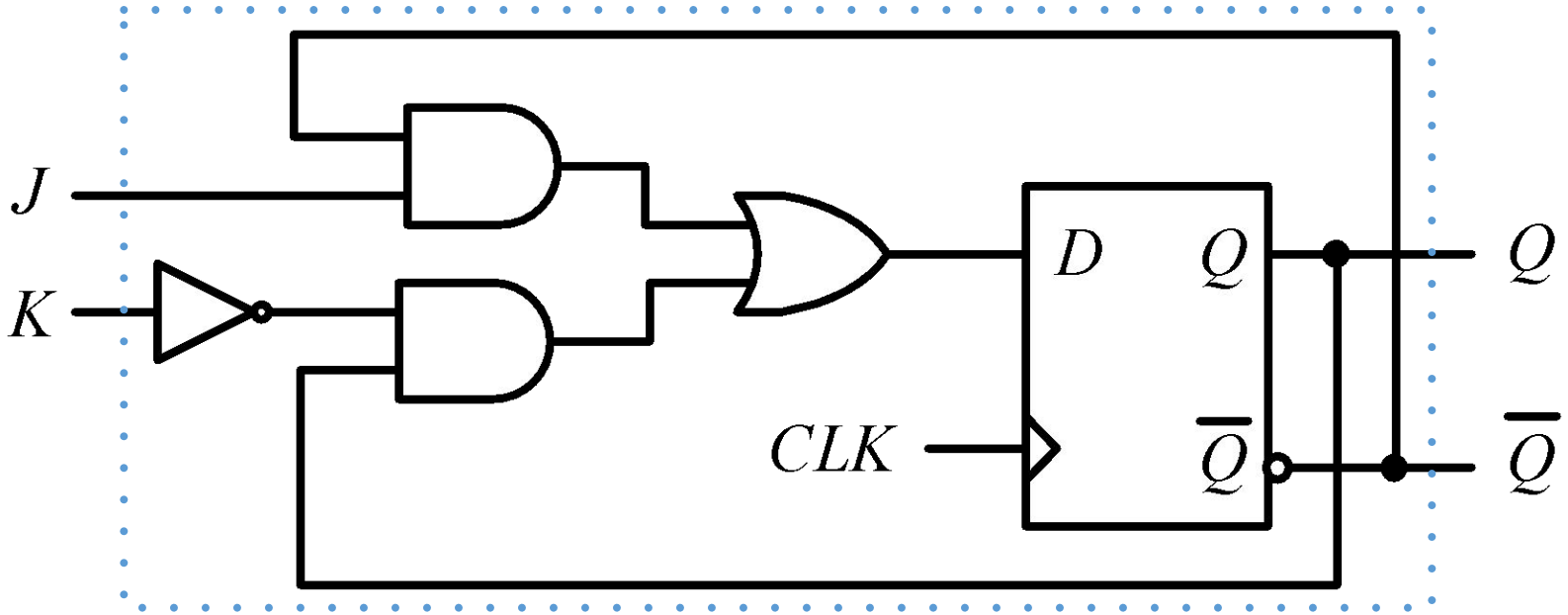
- Master-Slave *D* Flip-Flop



Looks like it is negative edge-triggered

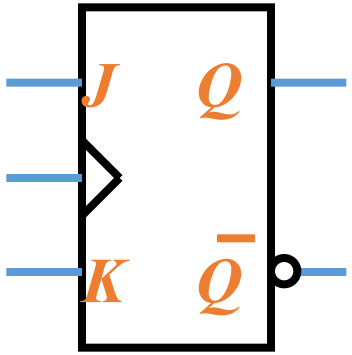
Flip-Flops

- *JK* Flip-Flop



$$D = JQ' + K'Q$$

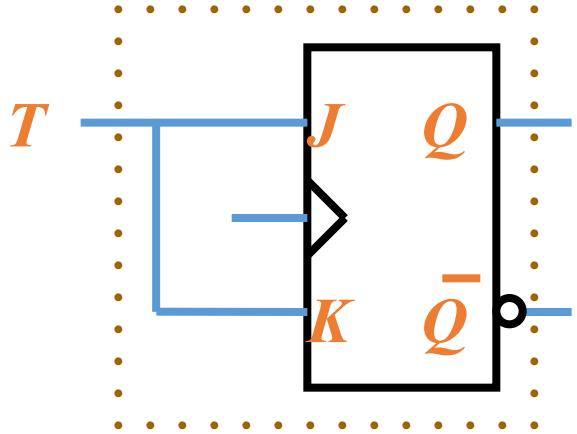
$$Q(t+1) = D = JQ' + K'Q$$

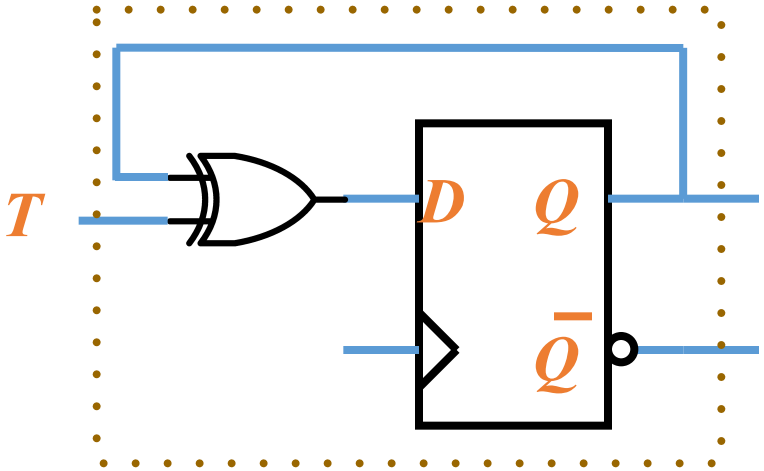


J	K	$Q(t+1)$
0	0	$Q(t)$
0	1	0
1	0	1
1	1	$Q'(t)$

Flip-Flops

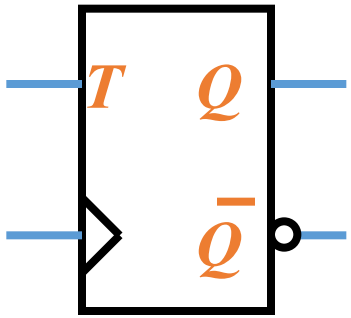
- T Flip-Flop





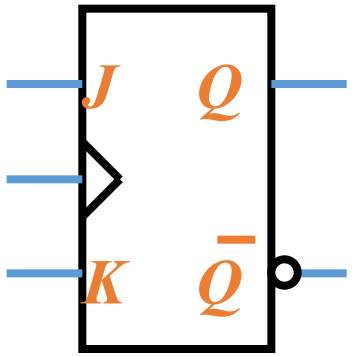
$D = JQ' + K'Q$

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



Flip-Flop Characteristic Equations

- Analysis / Derivation

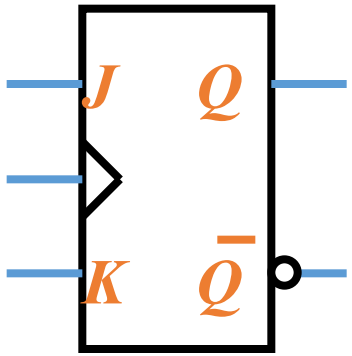


J	K	$Q(t)$	$Q(t+1)$
0	0	0	0
0	0	1	1
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

No change

Flip-Flop Characteristic Equations

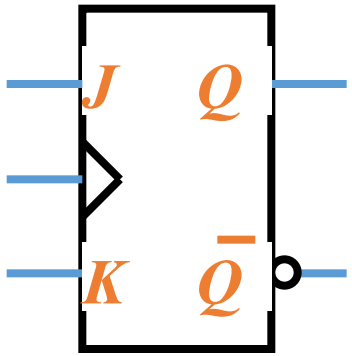
❖ Analysis / Derivation



J	K	$Q(t)$	$Q(t+1)$	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Flip-Flop Characteristic Equations

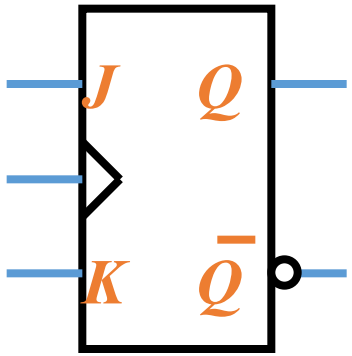
❖ Analysis / Derivation



J	K	$Q(t)$	$Q(t+1)$	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0		
1	1	1		

Flip-Flop Characteristic Equations

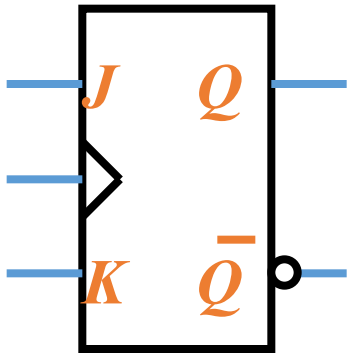
❖ Analysis / Derivation



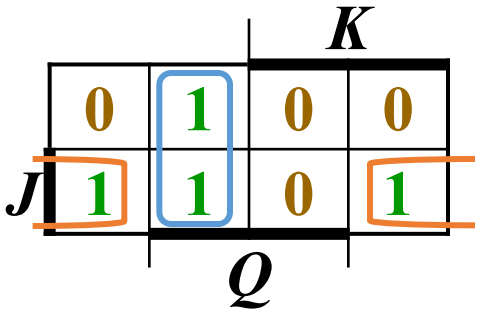
J	K	$Q(t)$	$Q(t+1)$	
0	0	0	0	No change
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

Flip-Flop Characteristic Equations

❖ Analysis / Derivation

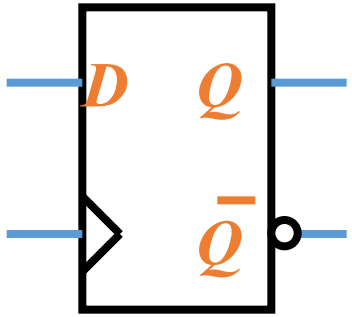


J	K	$Q(t)$	$Q(t+1)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

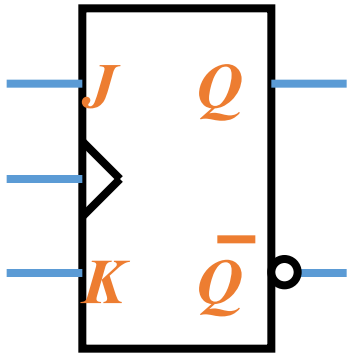


$$Q(t+1) = JQ' + K'Q$$

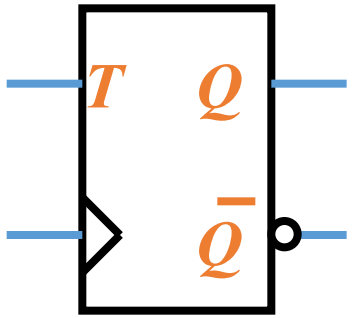
Flip-Flop Characteristic Tables



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

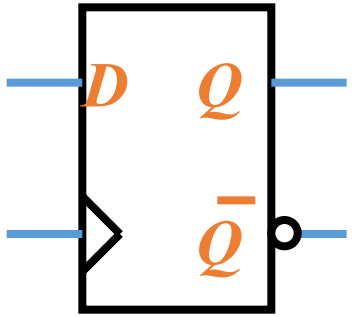


J	K	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Toggle



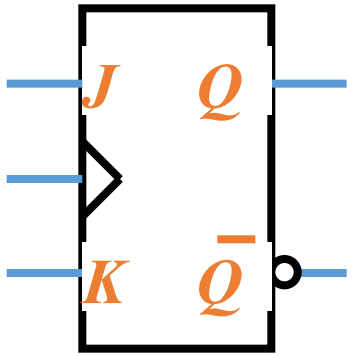
T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Toggle

Flip-Flop Characteristic Equations



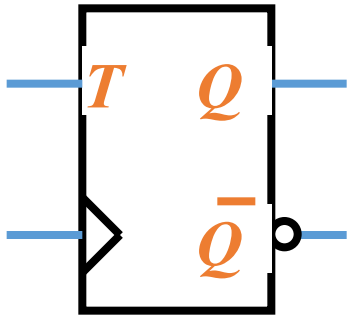
D	$Q(t+1)$
0	0
1	1

$$Q(t+1) = D$$



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

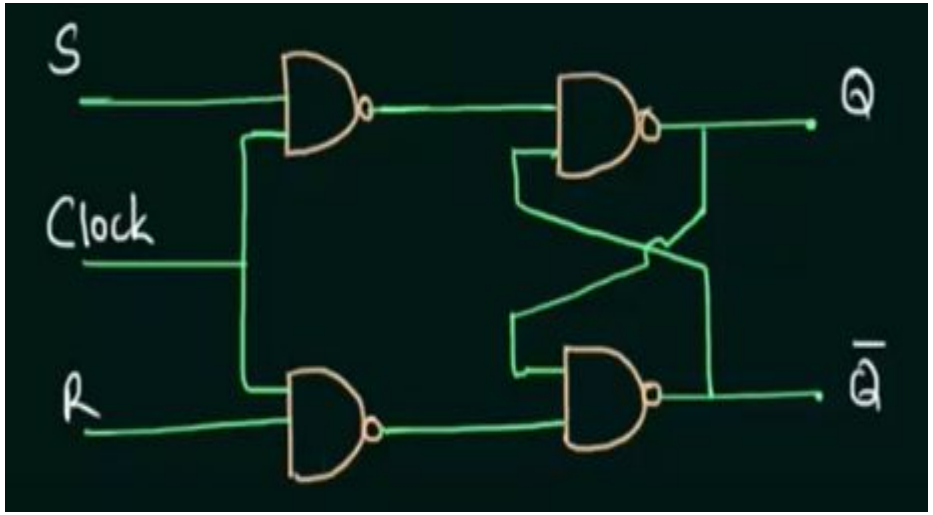
$$Q(t+1) = JQ' + K'Q$$

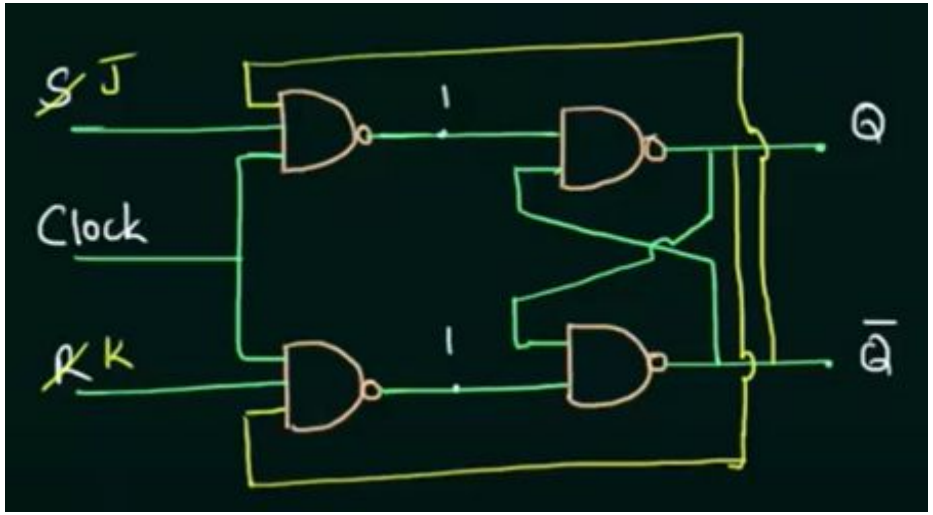


T	$Q(t+1)$
0	$Q(t)$
1	$Q'(t)$

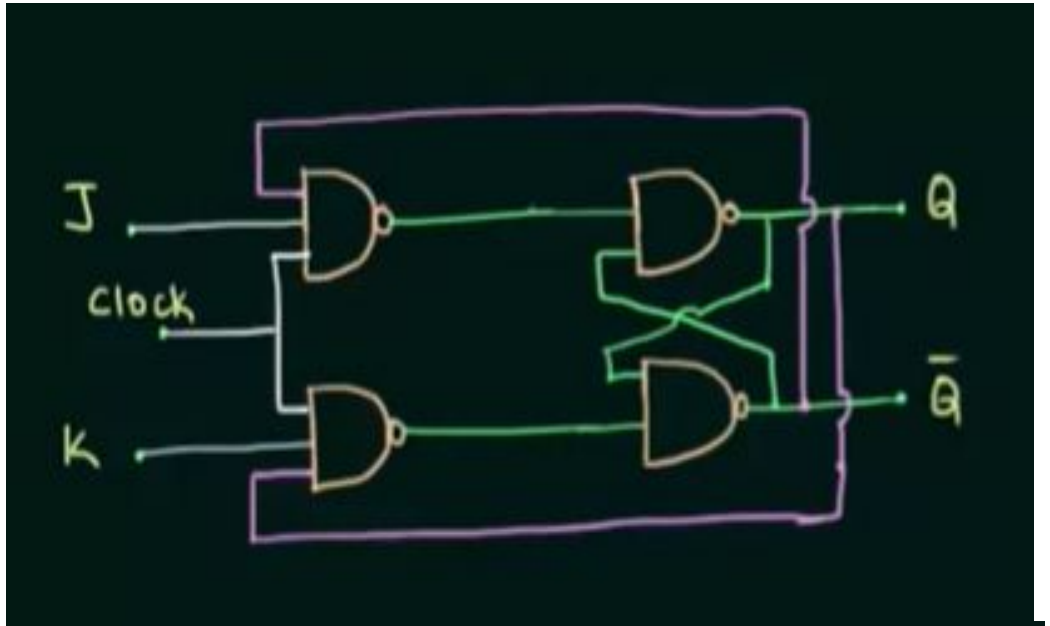
$$Q(t+1) = T \oplus Q$$

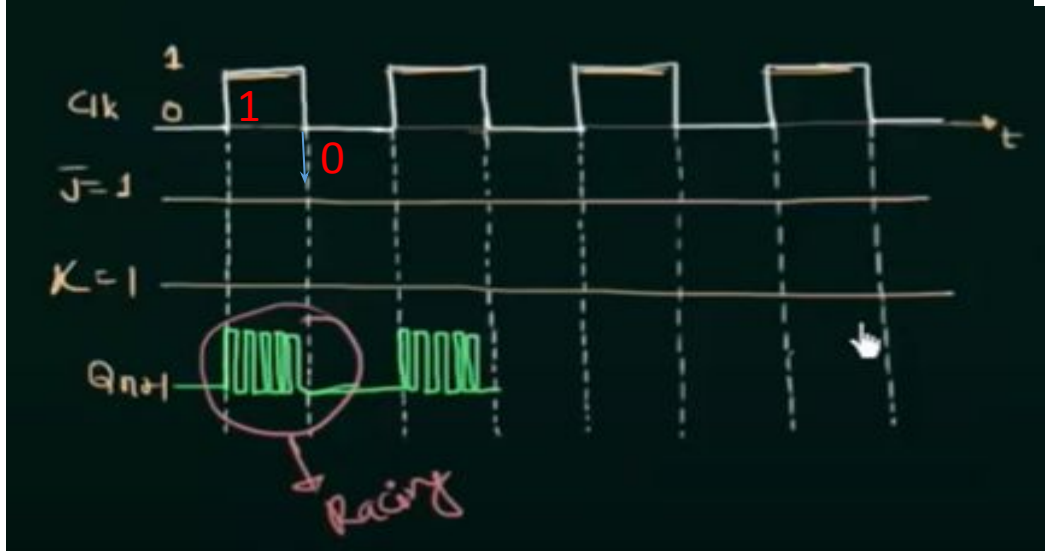
SR and JK flip flop





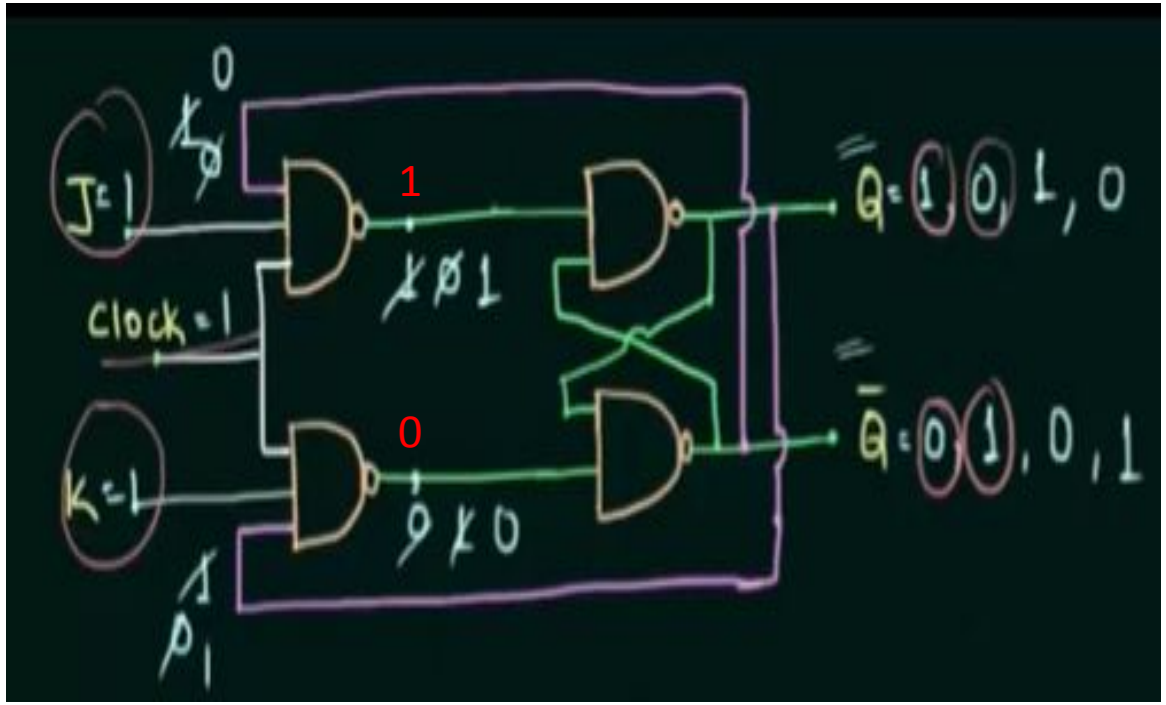
Race Around Condition of JK flip flop



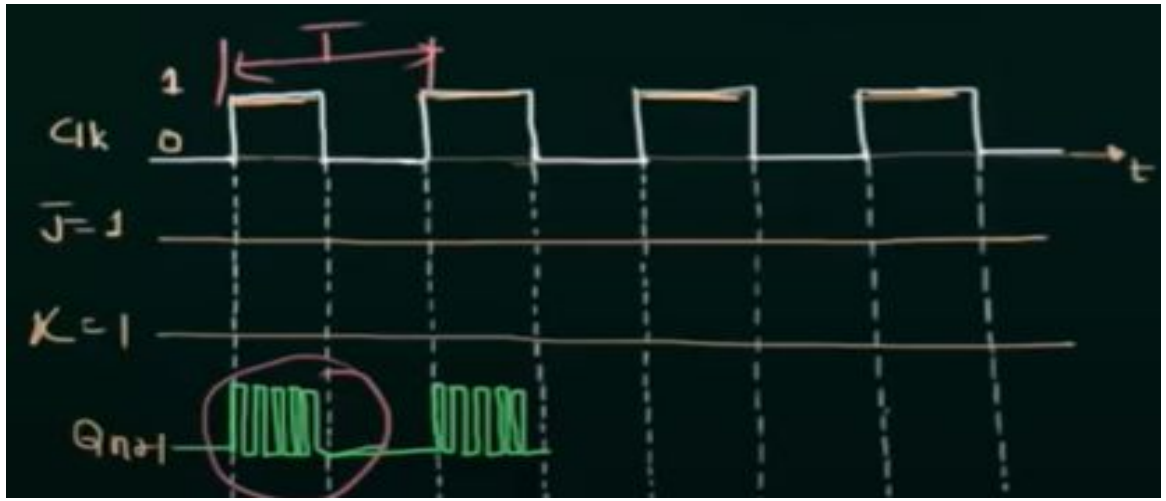


Clock	J	K	Q_{n+1}	$\overline{Q_{n+1}}$
0	X	X	Q_n	$\overline{Q_n}$
1	0	0	Q_n	$\overline{Q_n}$
1	0	1	0	1
1	1	0	1	0
1	1	1		

Race Around Condition of JK flip flop

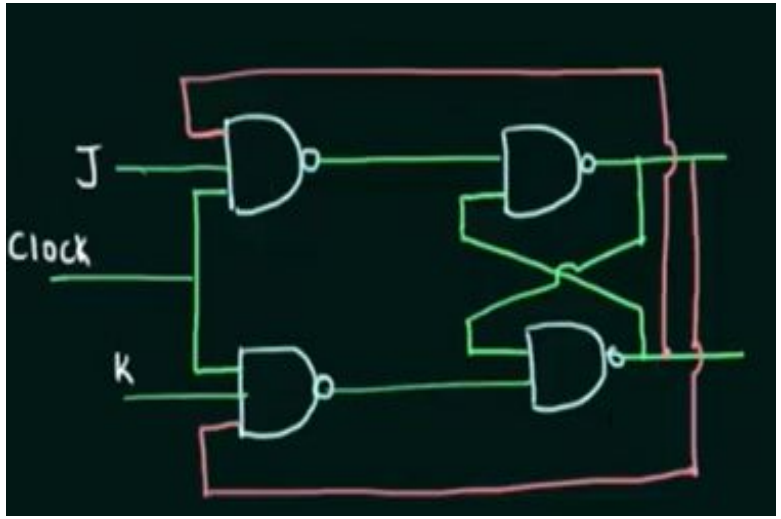


Clock	J	K	Q_{n+1}	$\bar{Q}_{n+1}$
0	X	X	Q_n	$\bar{Q}_{n-1}$
1	0	0	Q_n	$\bar{Q}_{n-1}$
1	0	1	0	1
1	1	0	1	0
1	1	1	?	?

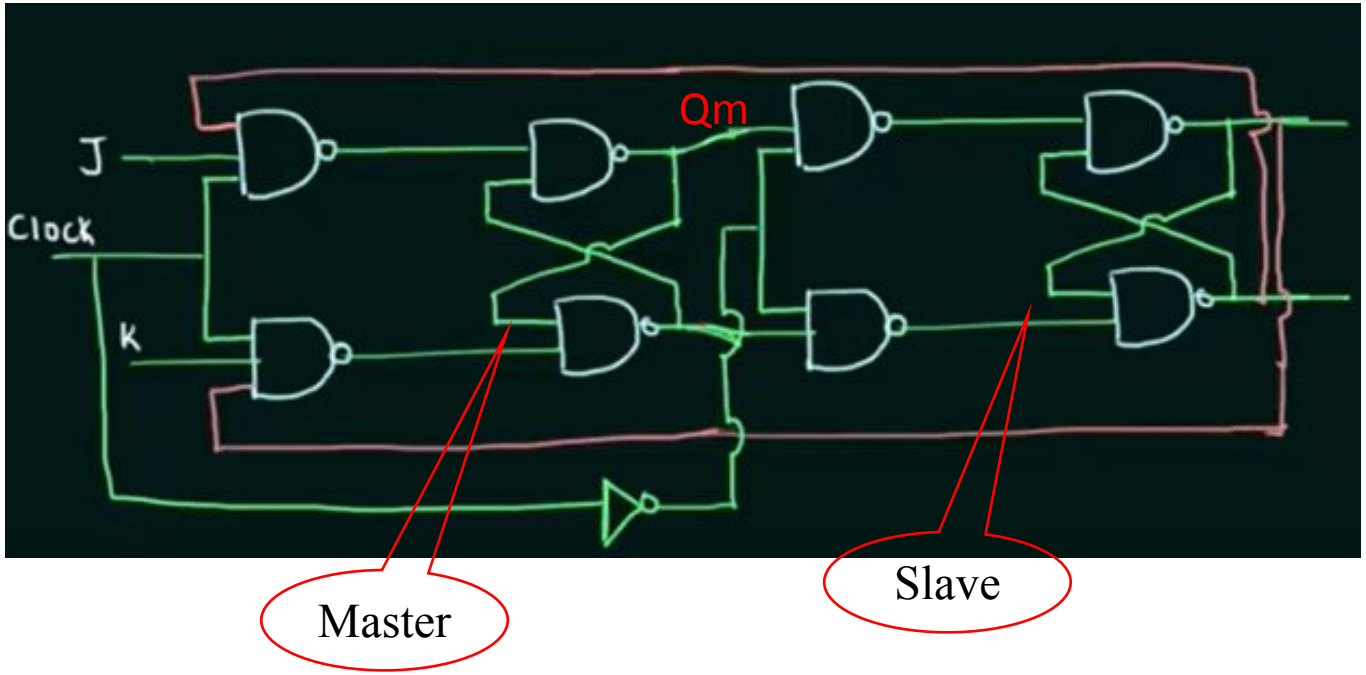


- Conditions to overcome Racing :
- T/2<Propagation delay of FF x
- Edge Triggering
- Master-Slave

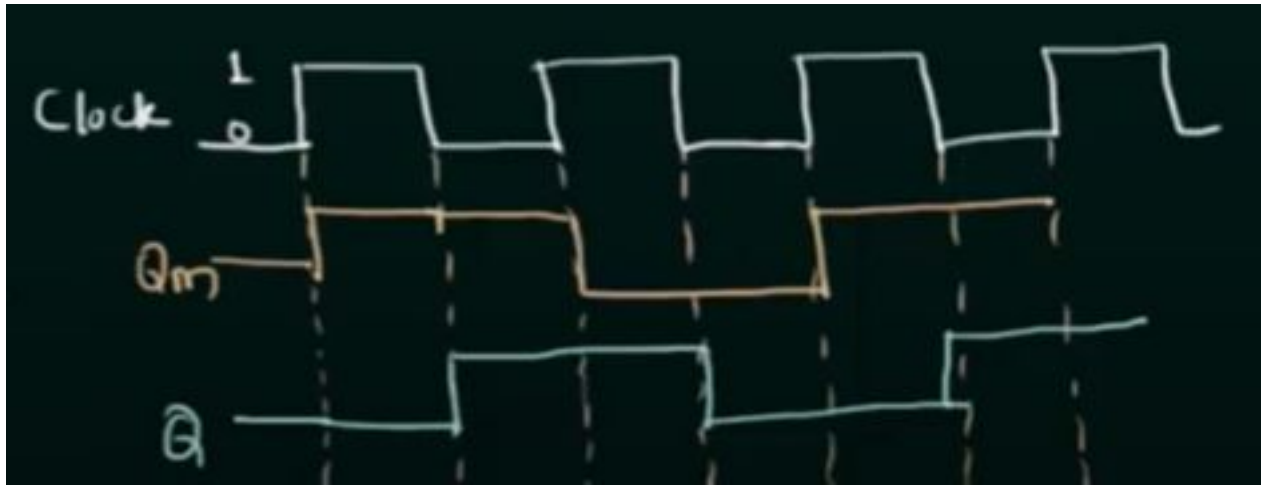
JK flip flop



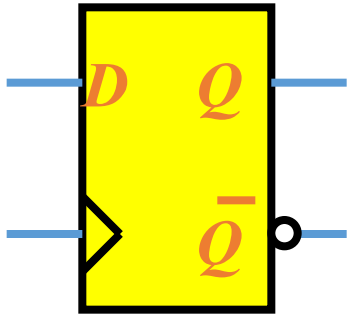
Master Slave Operation of JK flip flop



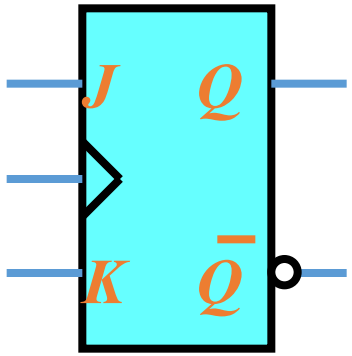
- T/2<Propagation Delay of FF
- Edge Triggered
- Master-Slave (M.S. operation is same as negative edge triggering])



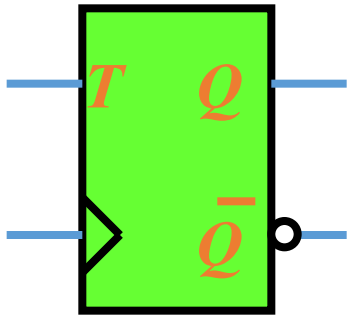
Flip-Flop Characteristic Tables



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

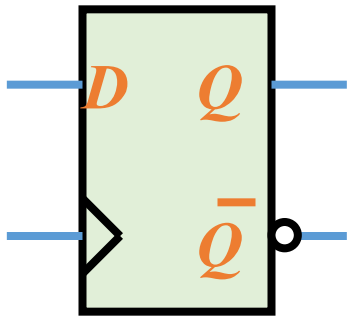


J	K	$Q(t+1)$	
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$Q'(t)$	Toggle



T	$Q(t+1)$	
0	$Q(t)$	No change
1	$Q'(t)$	Toggle

Flip-Flop Characteristic Equations

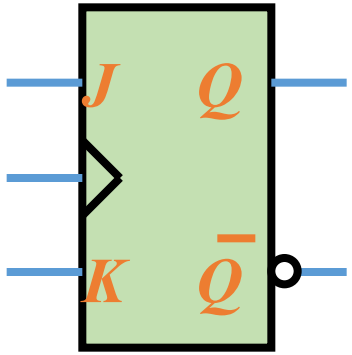


D	Q	*Q*(*t*+1)
0	0	0
0	1	0
1	0	1
1	1	1

$Q(t+1) = D$

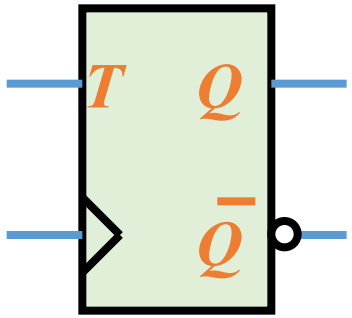
D \ Q	0	1
0	0	0
1	1	1

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



J \ KQ	00	01	11	10
0	0	1	0	0
1	1	1	0	1

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



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

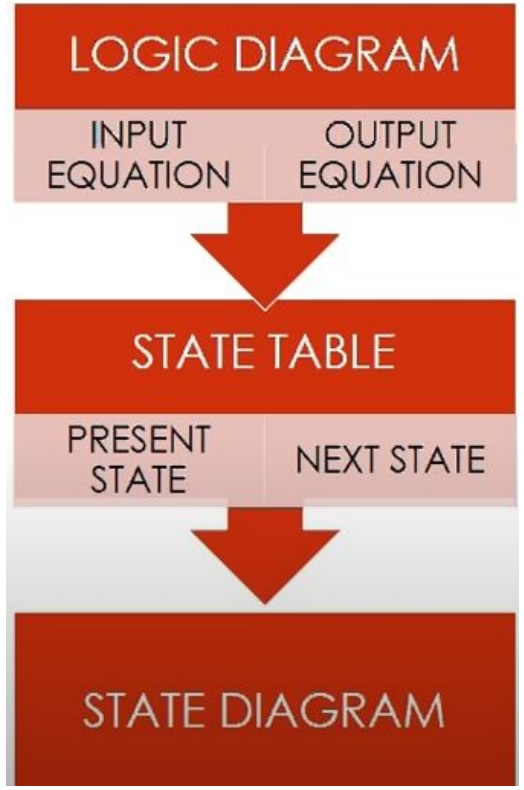
$Q(t+1) = TQ'+T'Q = T \oplus Q$

T \ Q	0	1
0	0	1
1	1	0

Clocked Sequential Circuits

1) Analyzing a clocked sequential circuit
2) Designing a sequential circuit to address a problem

Clocked Sequential Circuits



- ***STEP 1:*** Write down the equations for the outputs and the flip-flop inputs.
- ***STEP 2:*** Using these equations, derive a state table which describes the next state.
- ***STEP 3:*** Obtain a state diagram from the state table.

NOTE:

- It is the state table and/or state diagram that specifies the behavior of the circuit.
- The flip-flop input equations are sometimes called the excitation equations.
- The state table is sometimes called a transition table.

Clocked Sequential Circuits

1) Analyzing a clocked sequential circuit
2) Designing a sequential circuit to address a problem

Analyzing a Clocked Sequential Circuit

- A circuit will be given and we must understand the boundary between combinational stage and sequential stage
- Then identify the following specifications
  - No. flip flops and what types
  - How many external input
  - How many external output
  - No. of state variable (n)
  - No of states, 2^n

Analyzing a Clocked Sequential Circuit

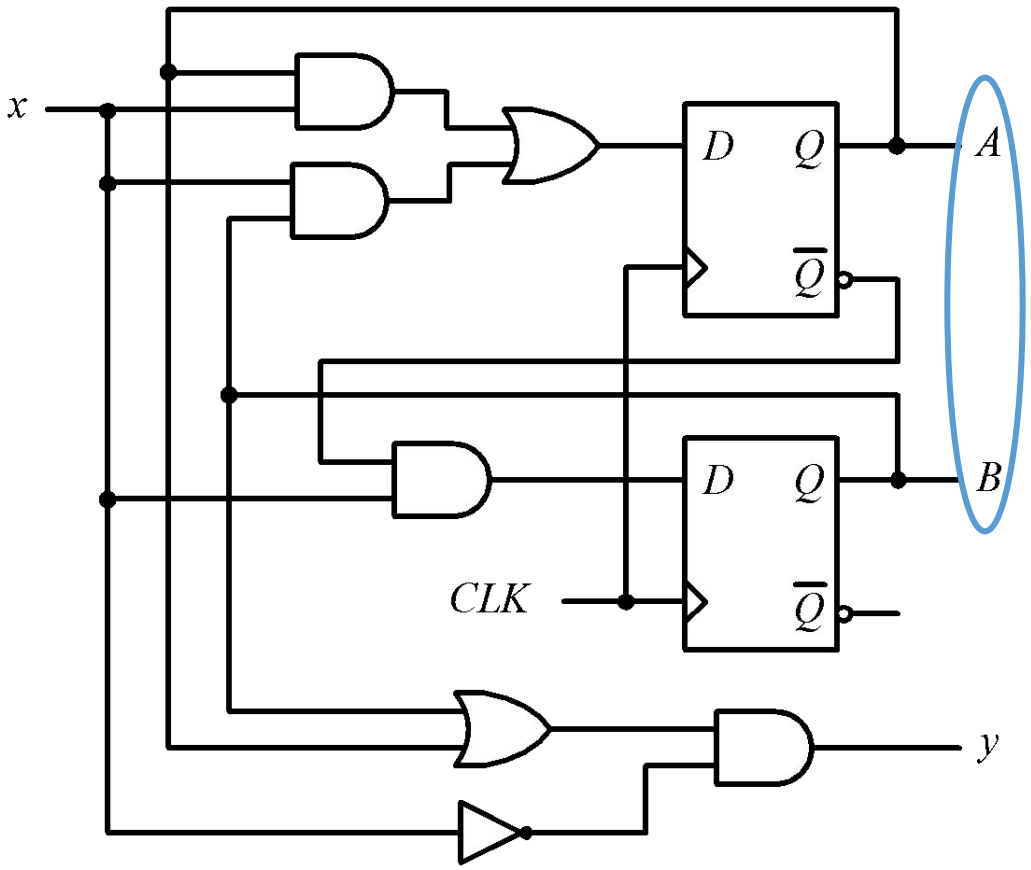
- Write down the equation of Flip Flop Input, External Output [3]
- Write down the equation for flip-flop output (known as state equation). This can be written by combining [3] with the flip flop's driving equation.
- Draw the state table
- Draw the state diagram

Analysis of Clocked Sequential Circuits

- The State
  - State = Values of all Flip-Flops

Example

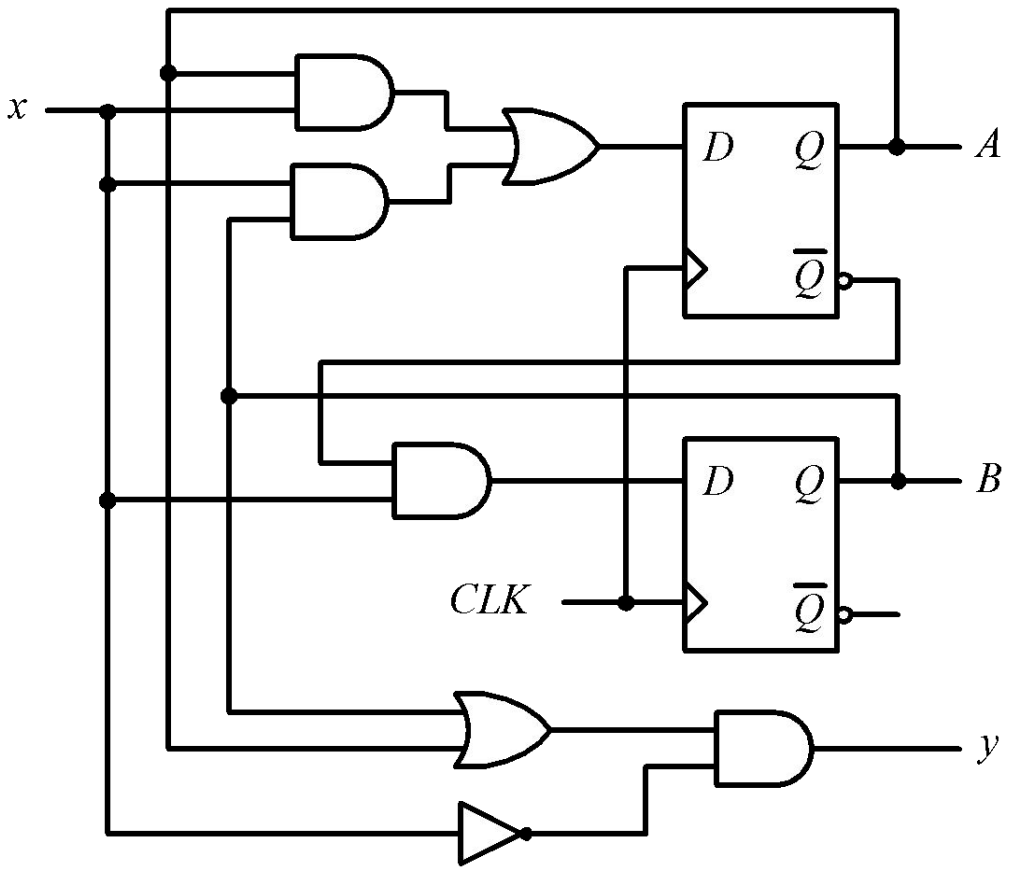
$A\ B = 0\ 0$



Analysis of Clocked Sequential Circuits

- State Equations

$$A(t+1) = D_A$$
$$= A(t)\,x(t) + B(t)\,x(t)$$
$$= A\,x + B\,x$$
$$B(t+1) = D_B$$
$$= A'(t)\,x(t)$$
$$= A'\,x$$
$$y(t) = [A(t) + B(t)]\,x'(t)$$
$$= (A + B)\,x'$$

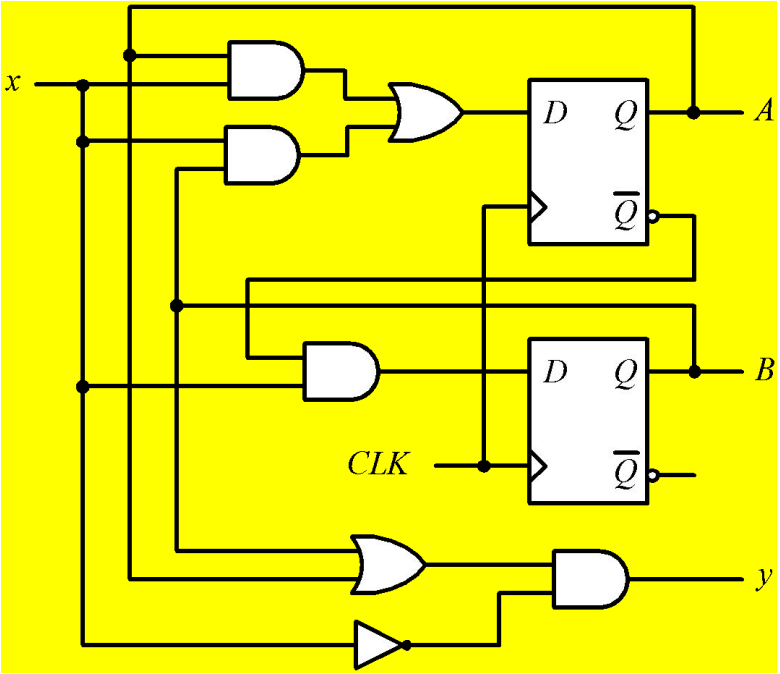


Analysis of Clocked Sequential Circuits

- State Table (Transition Table)

Present State		Next State				Output	
		$x = 0$		$x = 1$		$x = 0$	$x = 1$
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0

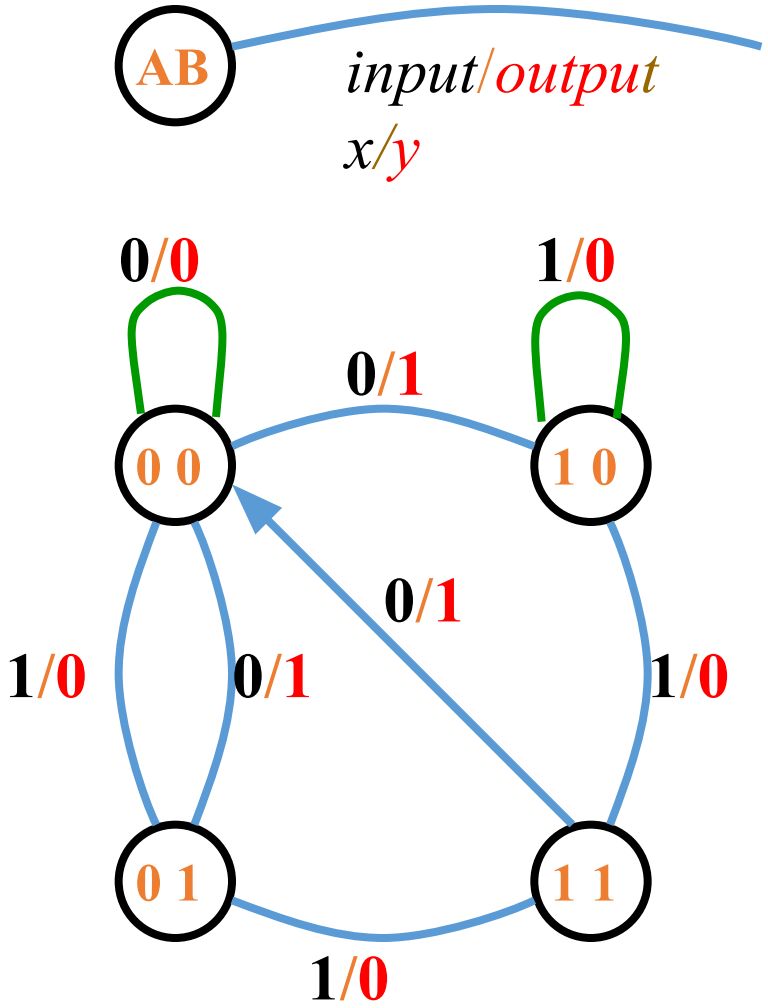
t $\quad$ $t+1$ $\quad$ t



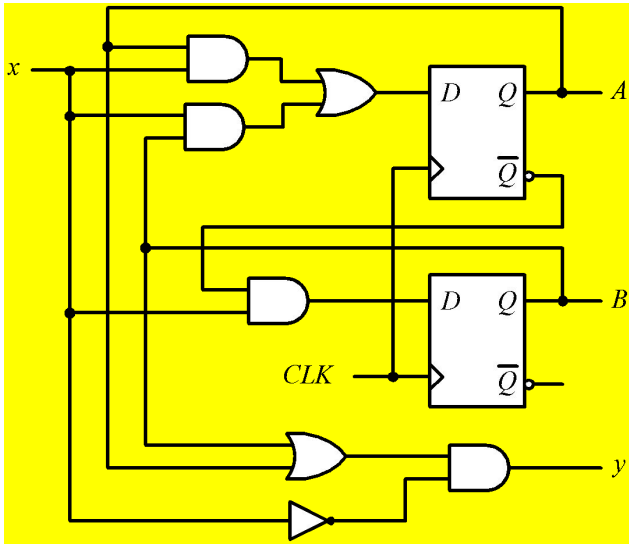
$$A(t+1) = A\,x + B\,x$$
$$B(t+1) = A'\,x$$
$$y(t) = (A + B)\,x'$$

Analysis of Clocked Sequential Circuits

- State Diagram



Present State	Next State				Output	
	$x = 0$		$x = 1$		$x = 0$	$x = 1$
A B	A	B	A	B	y	y
0 0	0	0	0	1	0	0
0 1	0	0	1	1	1	0
1 0	0	0	1	0	1	0
1 1	0	0	1	0	1	0

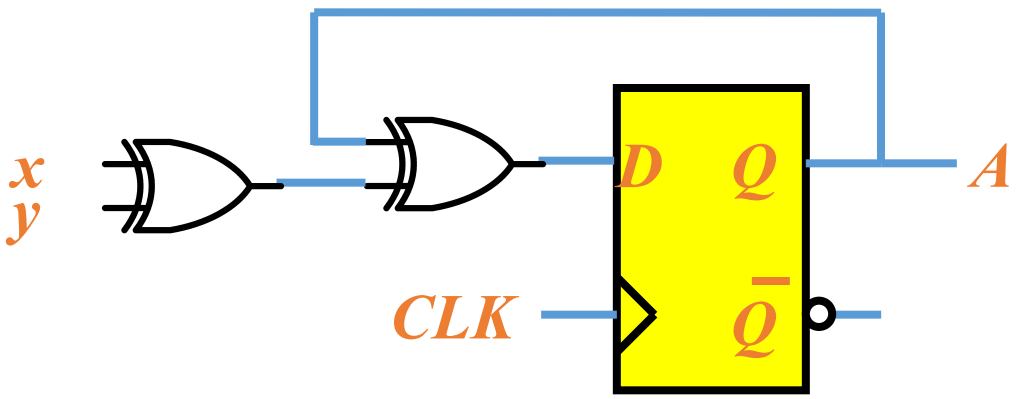


Analysis of Clocked Sequential Circuits

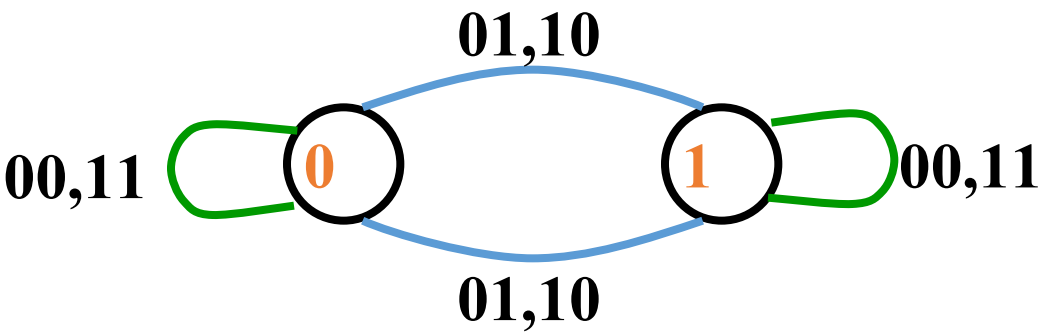
- *D* Flip-Flops

Example:

Present State	Input		Next State
A	*x*	*y*	*A*+
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



$$A(t+1) = D_A = A \oplus x \oplus y$$



❑ State Table and State Diagram

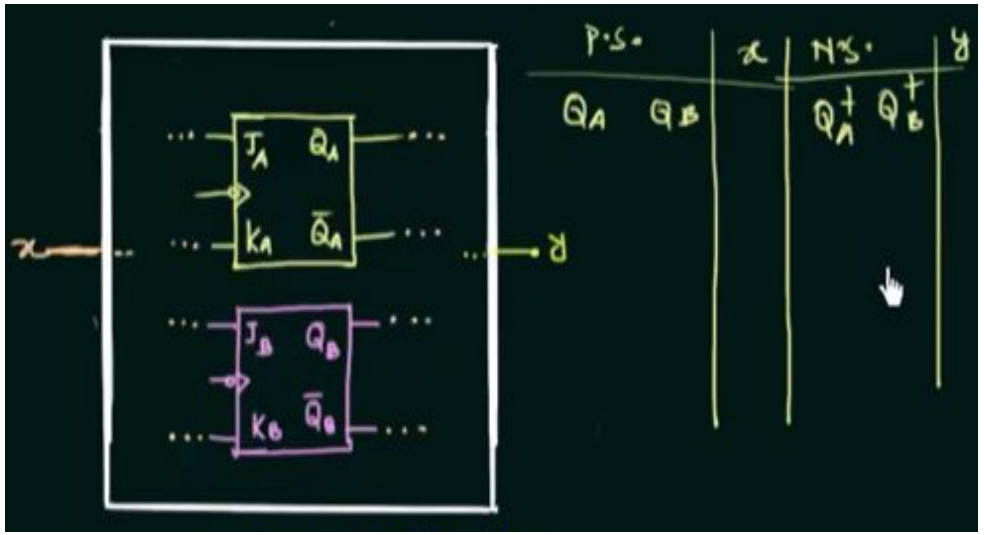
❖ State Diagram

- The state diagram is the pictorial representation of the behavior of sequential circuits. It clearly shows the transition of states from the present state to the next state and output for a corresponding input.

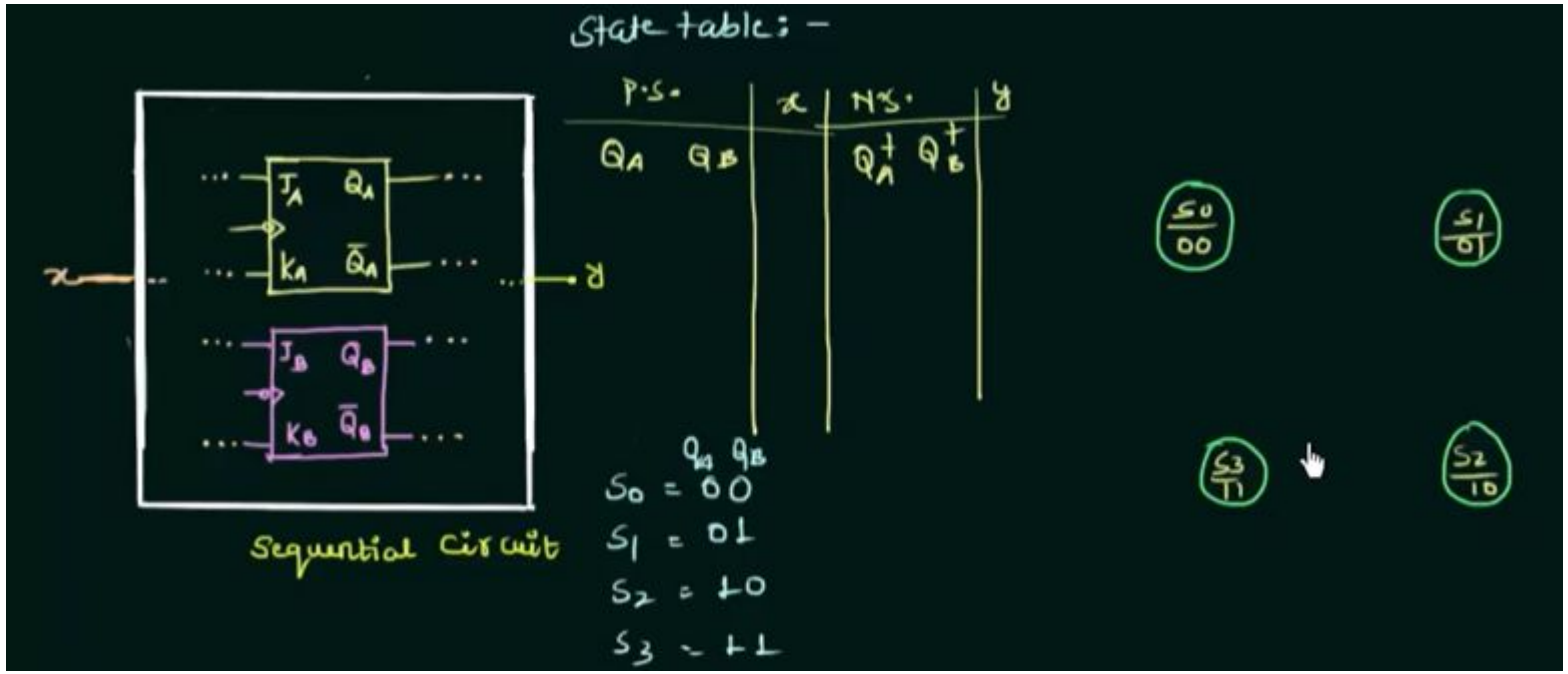
❖ State table

- The information contained in the state diagram is transformed into a table called as state table or state synthesis table. Although the state diagram describes the behavior of the sequential circuit, in order to implement it in the circuit, it has to be transformed into the tabular form.

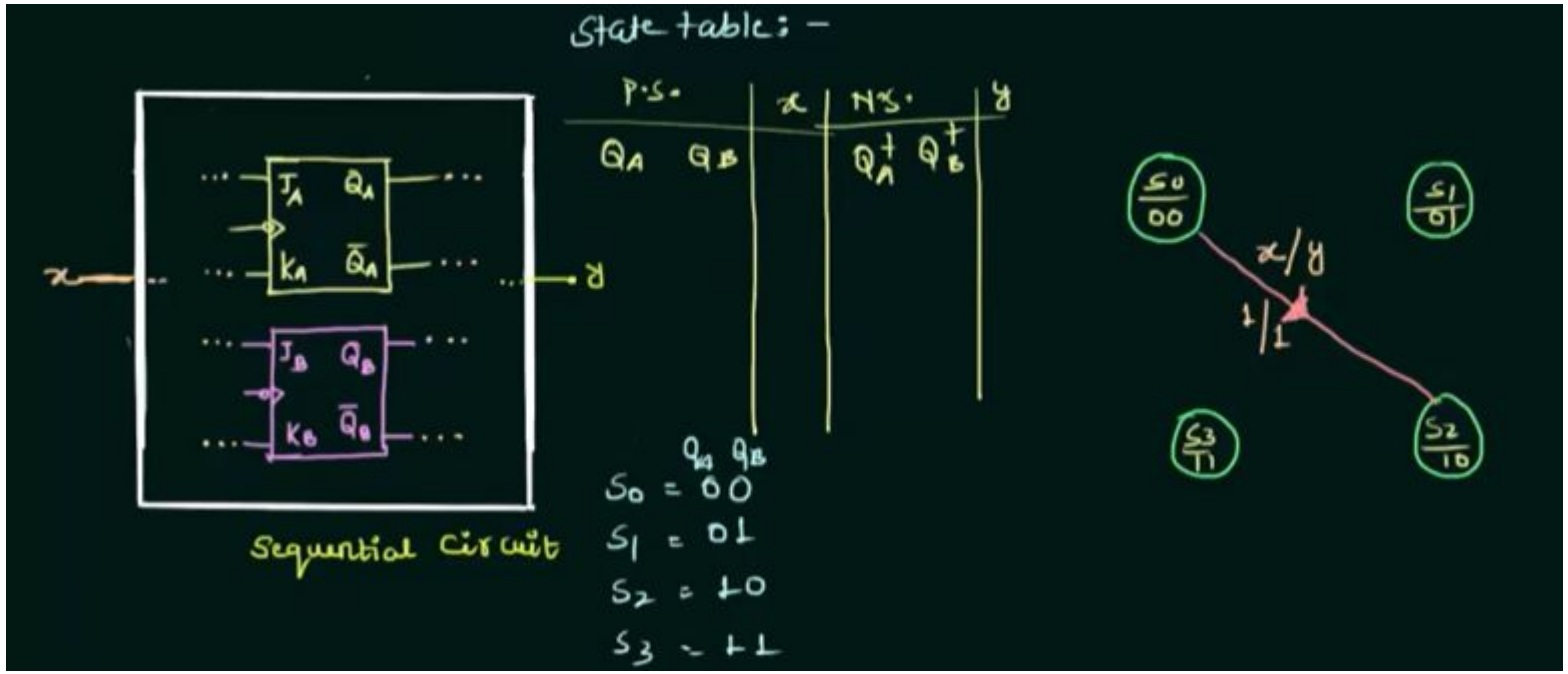
State Table



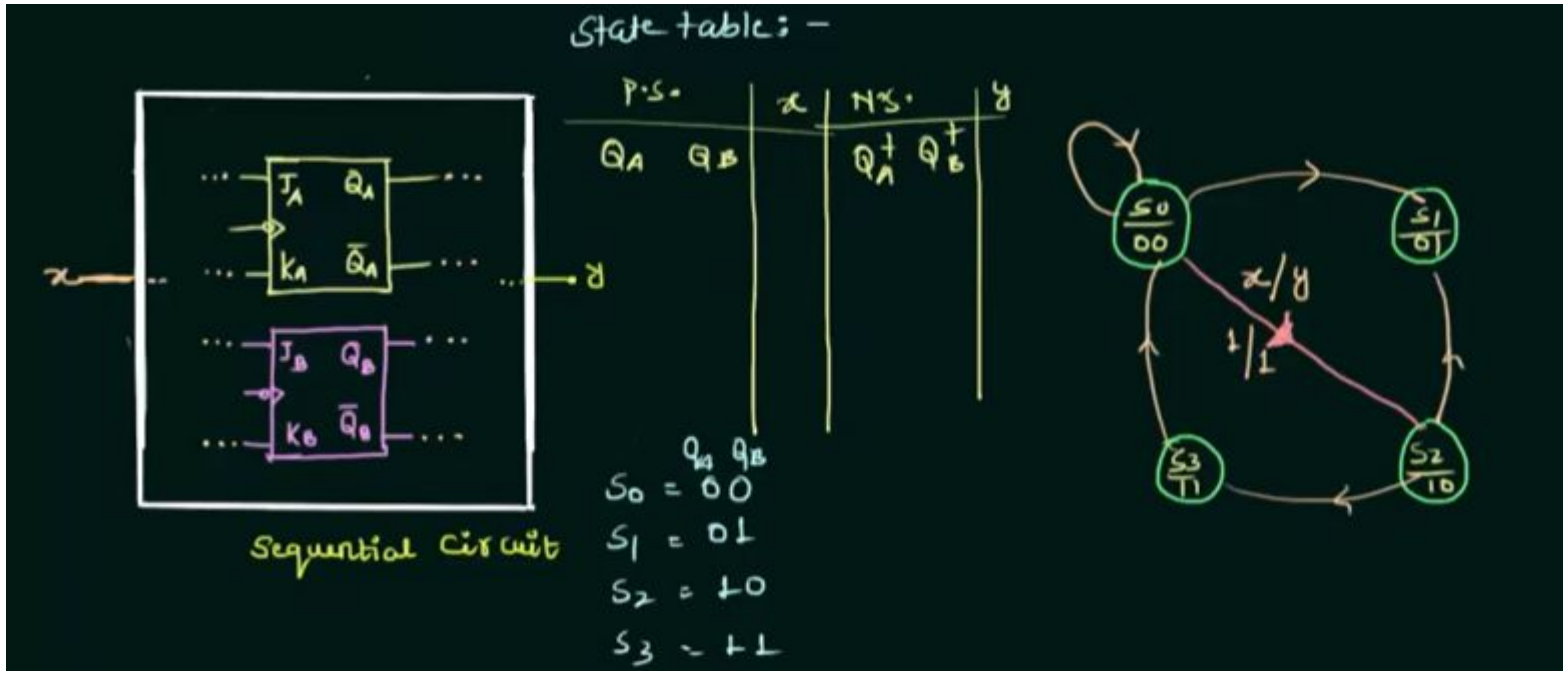
State Table and State Diagram



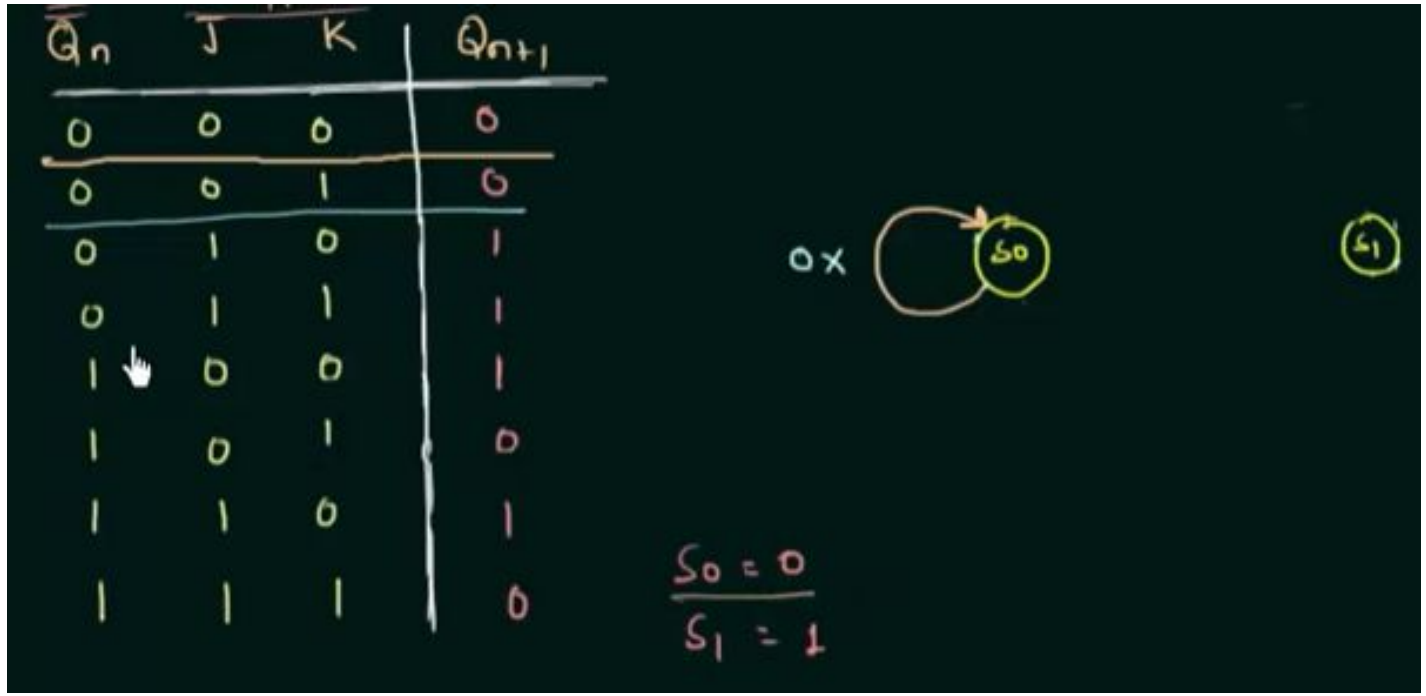
❑ State Table and State Diagram



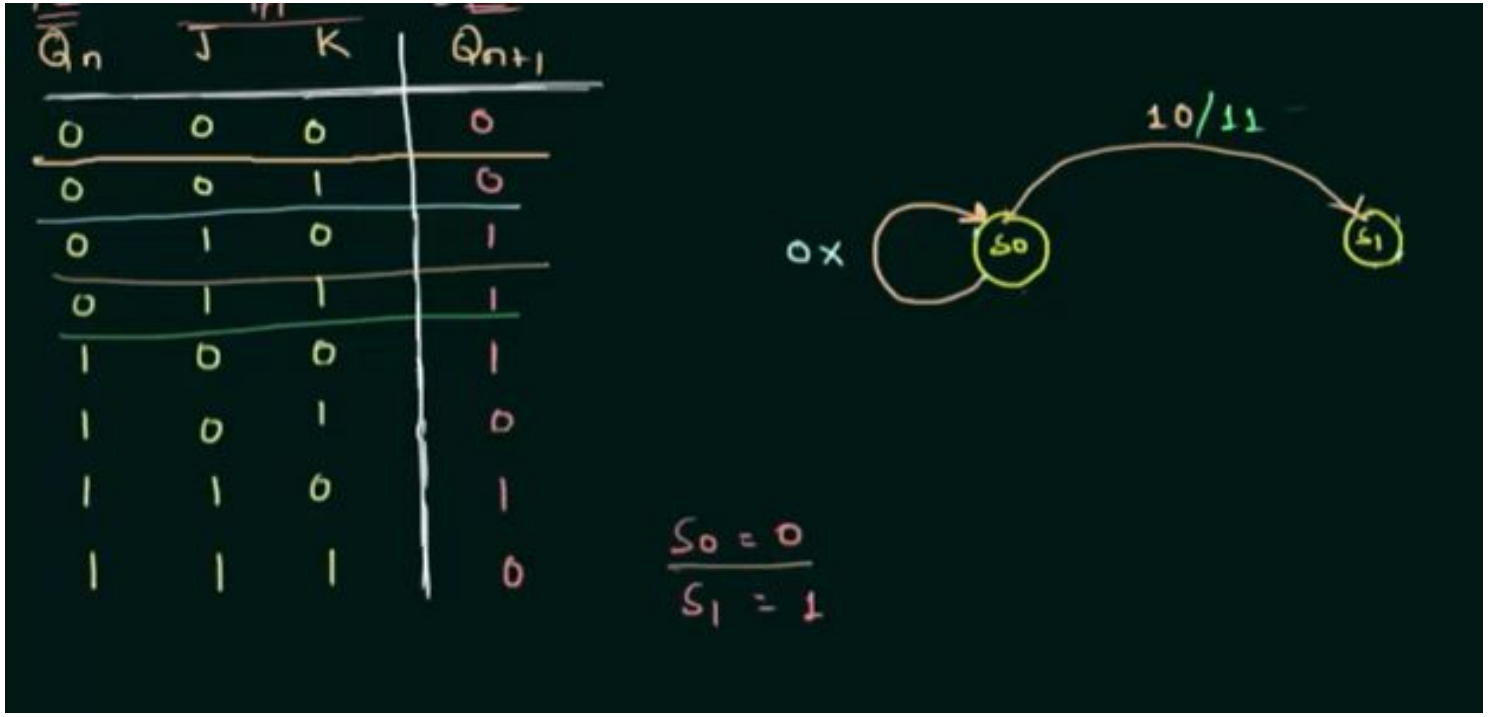
State Table and State Diagram



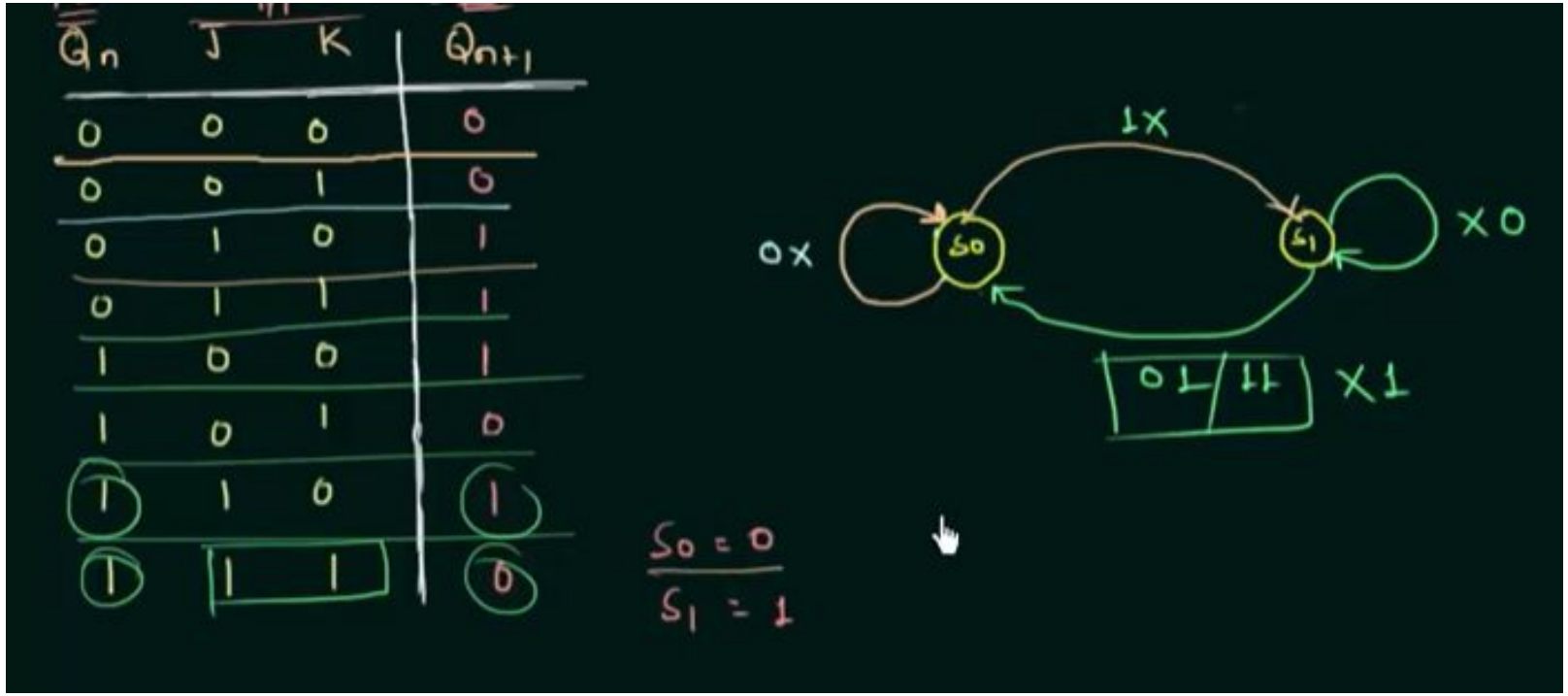
State Table and State Diagram



❑ State Table and State Diagram



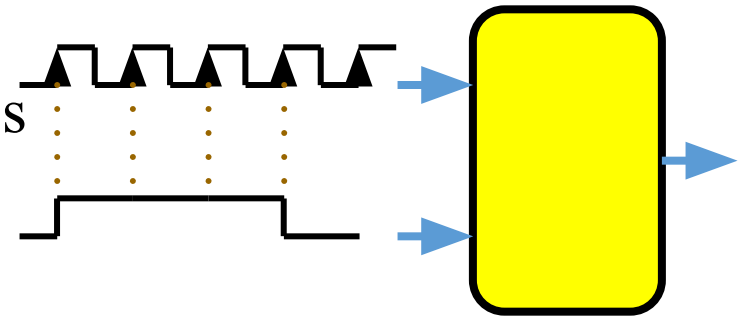
❑ State Table and State Diagram



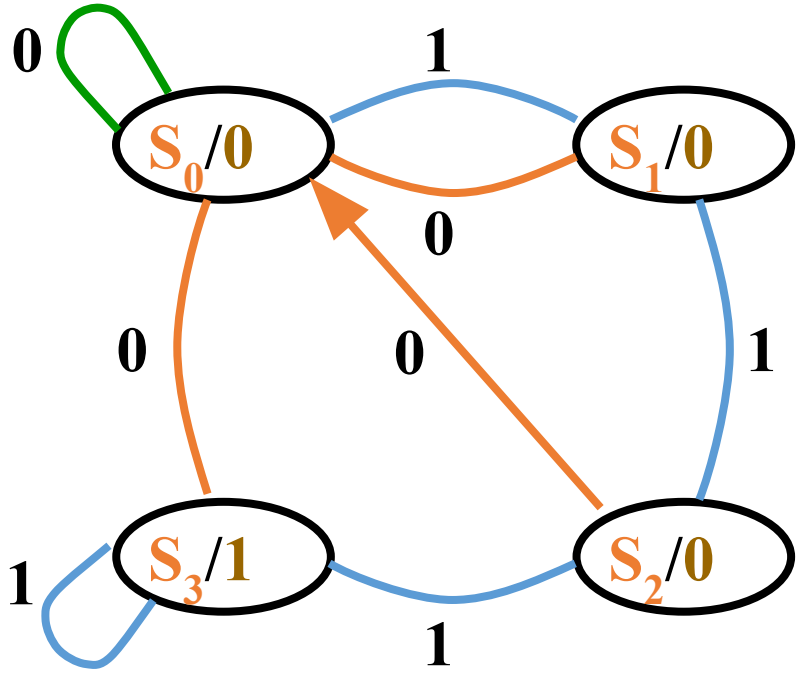
Example: Design a sequence detector to detect three or more consecutive 1's in a string of bits coming through an input line.

- Example:

Detect 3 or more consecutive 1's



Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

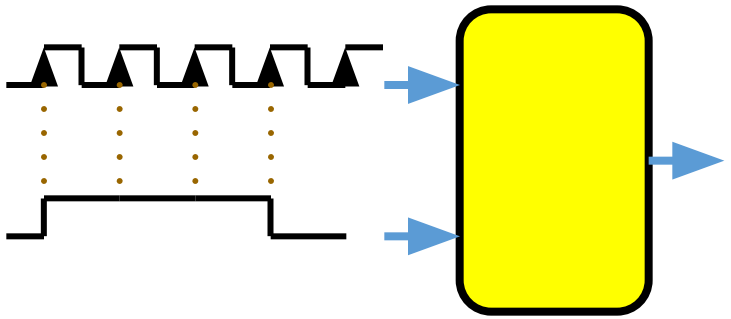


Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
S_0		0	S_0		0
S_0		1	S_1		0
S_1		0	S_0		0
S_1		1	S_2		0
S_2		0	S_0		0
S_2		1	S_3		0
S_3		0	S_0		1
S_3		1	S_3		1

Design of Clocked Sequential Circuits

- *Example*:

Detect 3 or more consecutive 1's



Present State		Input	Next State		Output
A	*B*	*x*	*A*	*B*	*y*
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0
0	1	1	1	0	0
1	0	0	0	0	0
1	0	1	1	1	0
1	1	0	0	0	1
1	1	1	1	1	1

$$A(t+1) = D_A(A, B, x) = \sum(3, 5, 7)$$

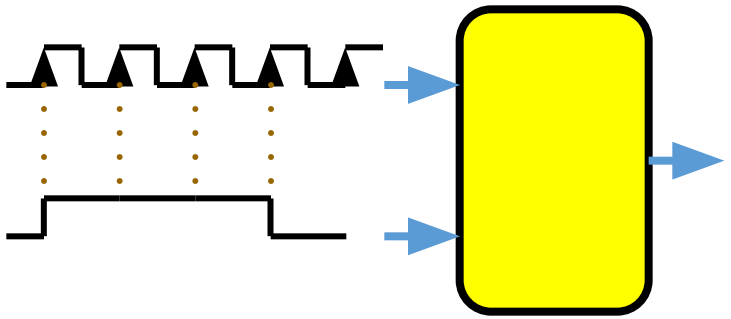
$$B(t+1) = D_B(A, B, x) = \sum(1, 5, 7)$$

$$y(A, B, x) = \sum(6, 7)$$

Design of Clocked Sequential Circuits with *D* F.F.

- Example:

Detect 3 or more consecutive 1's



Synthesis using *D* Flip-Flops

$$D_A(A, B, x) = \sum(3, 5, 7)$$
$$= A\,x + B\,x$$
$$D_B(A, B, x) = \sum(1, 5, 7)$$
$$= A\,x + B'\,x$$
$$y(A, B, x) = \sum(6, 7)$$
$$= A\,B$$

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

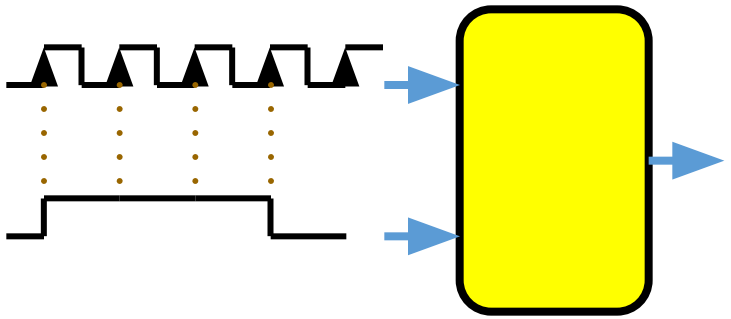
A \ x	B	
	0	1	0	0
0	1	1	0

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

Design of Clocked Sequential Circuits with *D* F.F.

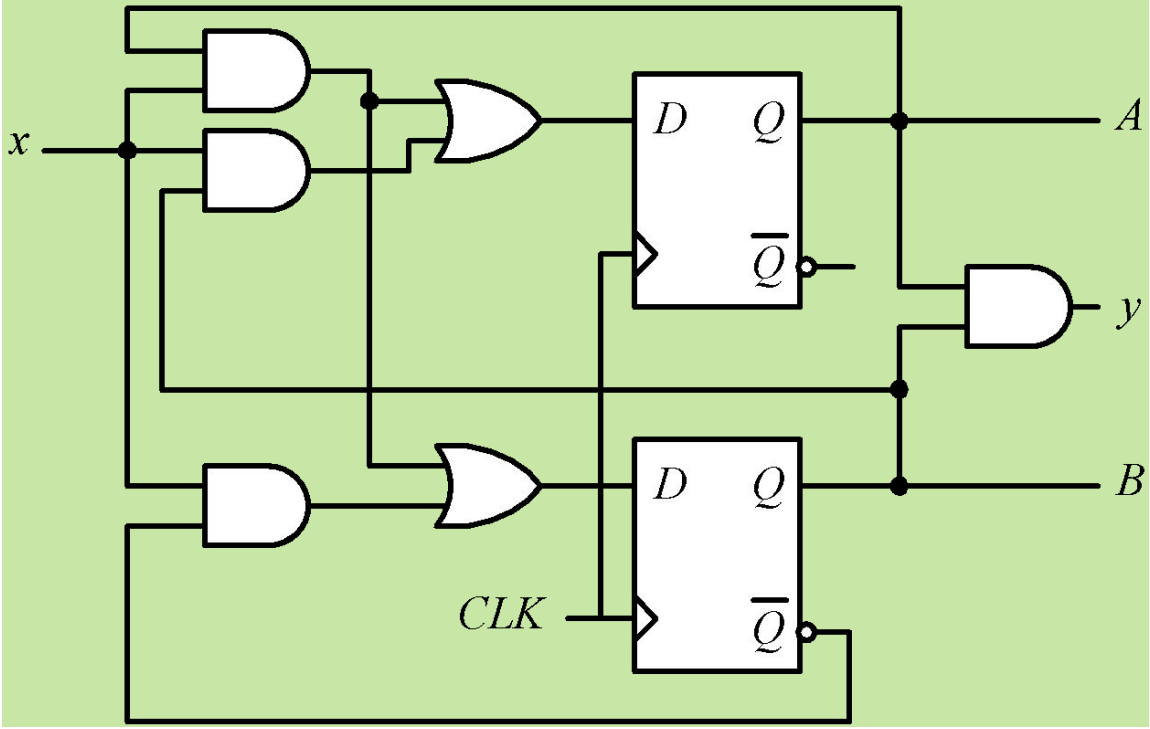
- Example:

Detect 3 or more consecutive 1's



Synthesis using *D* Flip-Flops

$$D_A = A\,x + B\,x$$
$$D_B = A\,x + B'\,x$$
$$y = A\,B$$



Flip-Flop Excitation Tables

Present State	Next State	F.F. Input
$Q(t)$	$Q(t+1)$	D
0	0	0
0	1	1
1	0	0
1	1	1

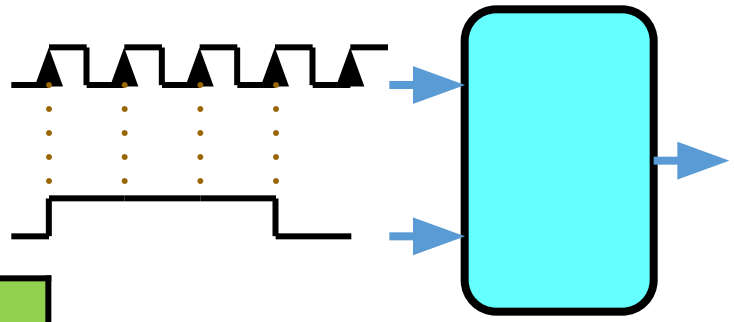
Present State	Next State	F.F. Input	
$Q(t)$	$Q(t+1)$	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

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

Design of Clocked Sequential Circuits with *JK* F.F.

- Example:

Detect 3 or more consecutive 1's



Present State		Input	Next State		Flip-Flop Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	x	0	x
0	0	1	0	1	0	x	1	x
0	1	0	0	0	0	x	x	1
0	1	1	1	0	1	x	x	1
1	0	0	0	0	x	1	0	x
1	0	1	1	1	x	0	1	x
1	1	0	0	0	x	1	x	1
1	1	1	1	1	x	0	x	0

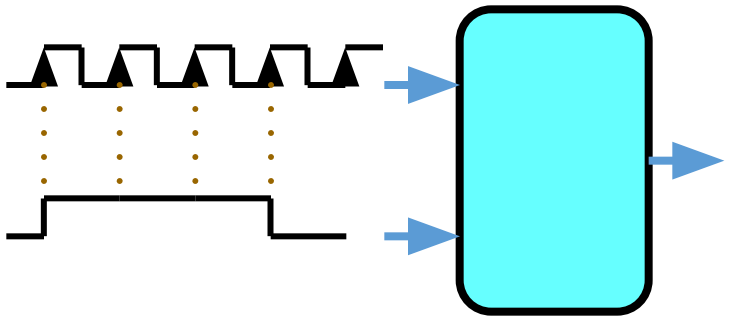
Synthesis using *JK* F.F.

$J_A(A, B, x) = \sum(3)$
$d_{JA}(A, B, x) = \sum(4,5,6,7)$
$K_A(A, B, x) = \sum(4, 6)$
$d_{KA}(A, B, x) = \sum(0,1,2,3)$
$J_B(A, B, x) = \sum(1, 5)$
$d_{JB}(A, B, x) = \sum(2,3,6,7)$
$K_B(A, B, x) = \sum(2, 3, 6)$
$d_{KB}(A, B, x) = \sum(0,1,4,5)$

Design of Clocked Sequential Circuits with *JK* F.F.

- Example:

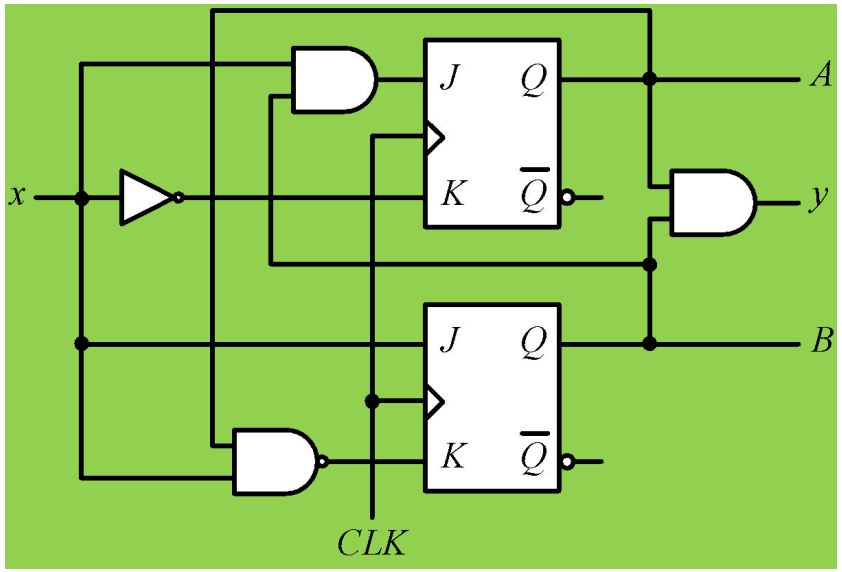
Detect 3 or more consecutive 1's

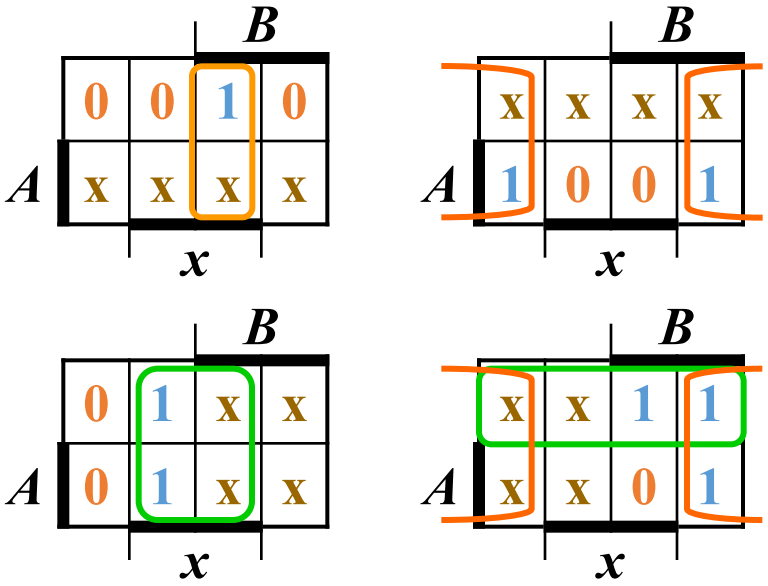


Synthesis using *JK* Flip-Flops

$J_A = B\,x \quad K_A = x'$

$J_B = x \quad K_B = A' + x'$



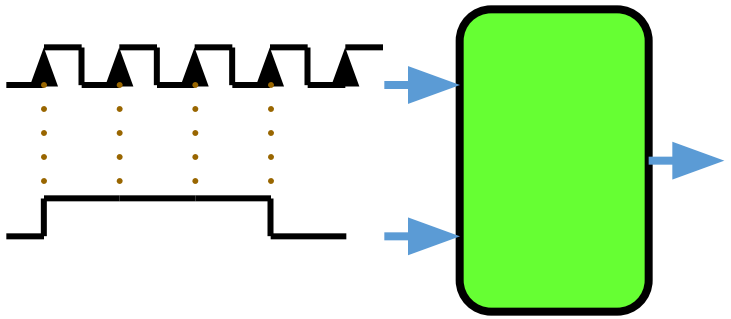


Design of Clocked Sequential Circuits with *T* F.F.

- Example:

Detect 3 or more consecutive 1's

Present State		Input	Next State		F.F. Input	
A	B	x	A	B	T_A	T_B
0	0	0	0	0	0	0
0	0	1	0	1	0	1
0	1	0	0	0	0	1
0	1	1	1	0	1	1
1	0	0	0	0	1	0
1	0	1	1	1	0	1
1	1	0	0	0	1	1
1	1	1	1	1	0	0



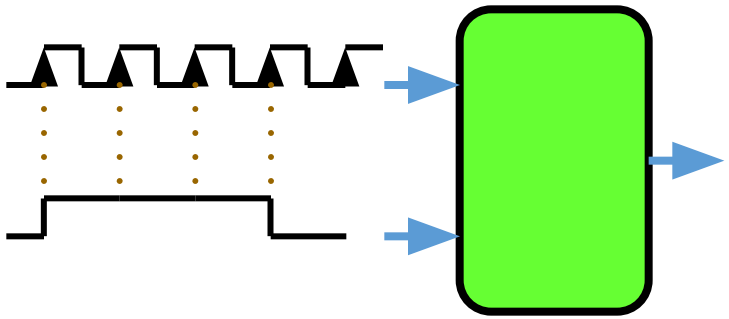
Synthesis using *T* Flip-Flops

$$T_A(A, B, x) = \sum(3, 4, 6)$$
$$T_B(A, B, x) = \sum(1, 2, 3, 5, 6)$$

Design of Clocked Sequential Circuits with *T* F.F.

- Example:

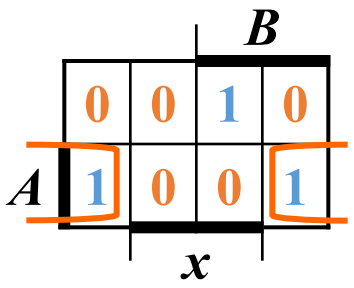
Detect 3 or more consecutive 1's

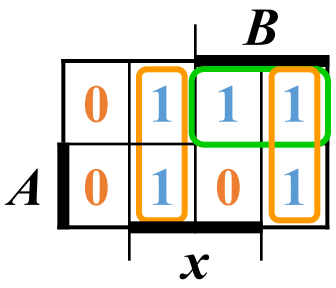


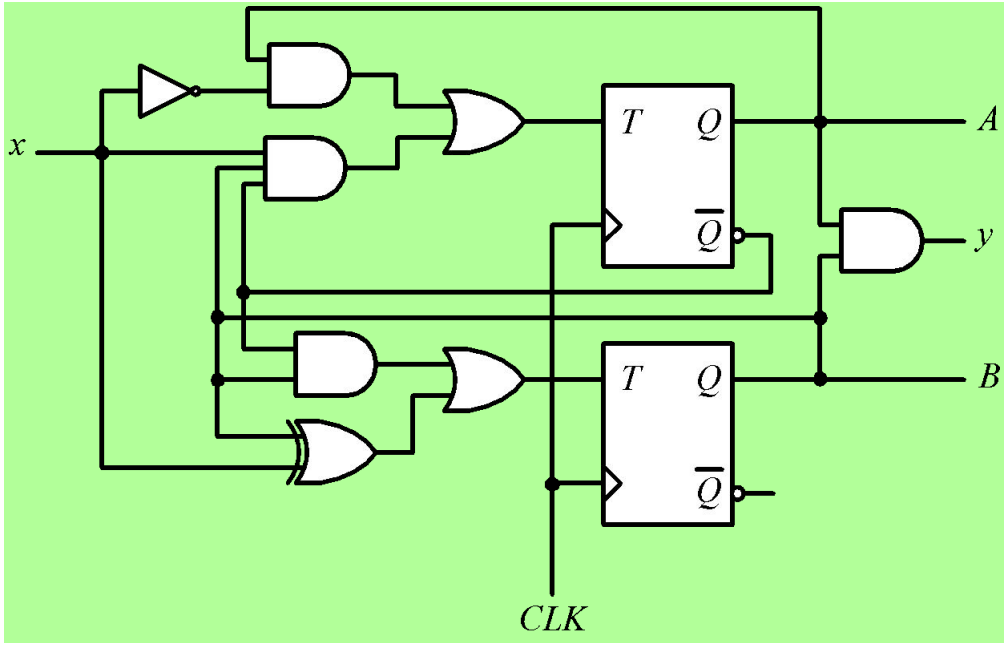
Synthesis using *T* Flip-Flops

$T_A = A\,x' + A'\,B\,x$

$T_B = A'\,B + B \oplus x$







Thanks