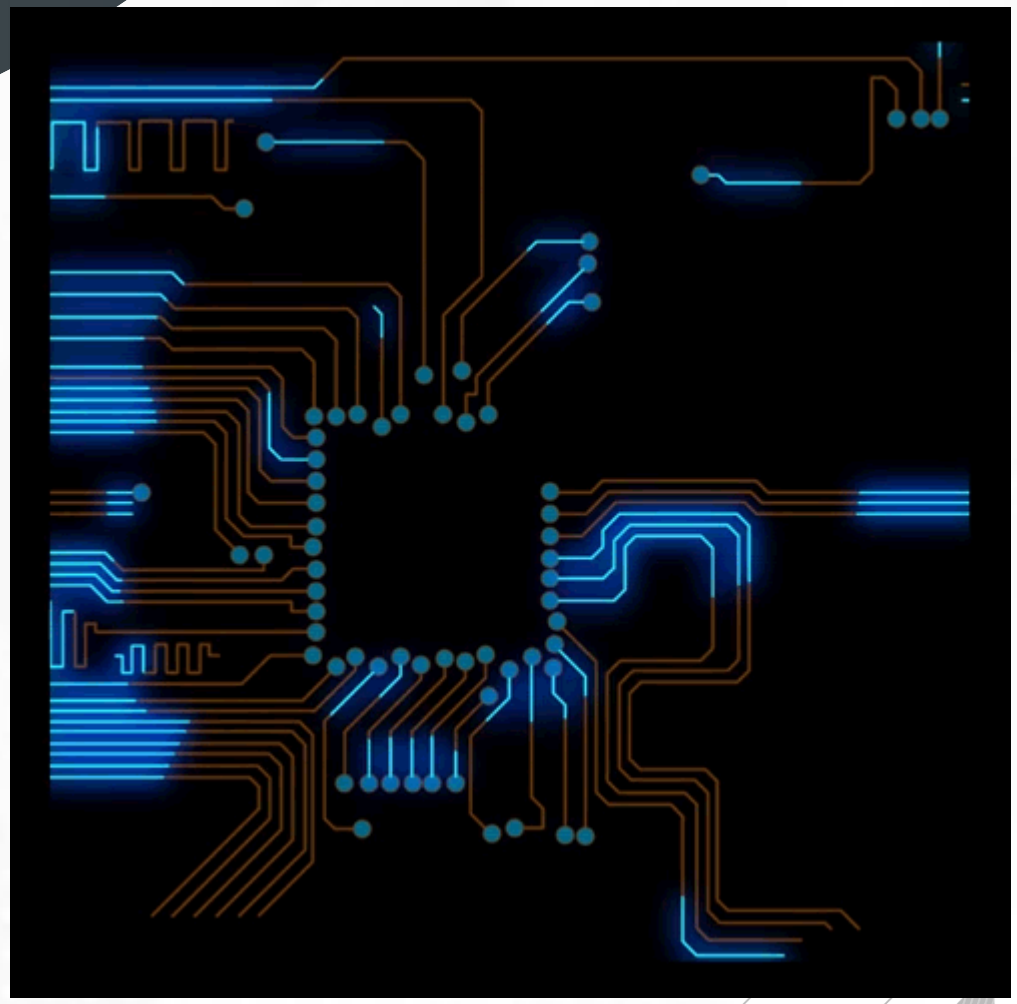
Digital Electronics (DE)
Code: 01EC0102 3rd sem

Unit-1
Number Systems and Codes



Outline

1. Analogue Versus Digital
2. Introduction to Number Systems and its Conversions
3. Floating-Point Numbers
4. Various binary code

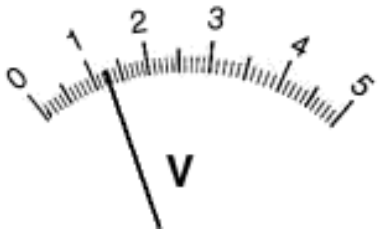
Marwadi University

UNIT -1

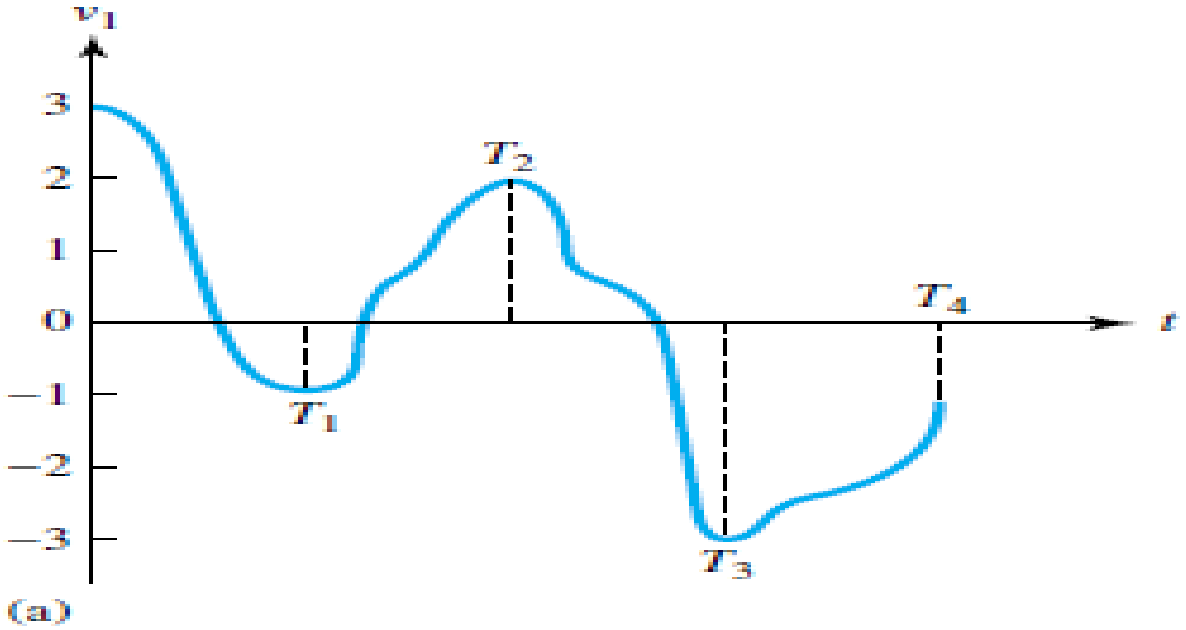
Signal

- Signal: Physical quantity which contains some information.
- It is a function of one or more independent variables.
- Types: Analog and Digital

Types of signal: Aanalog and Digital signal



- An ***analog signal*** is an electric signal whose value **varies** in analogy with a physical quantity such as **temperature, force, or acceleration and any natural signal.**



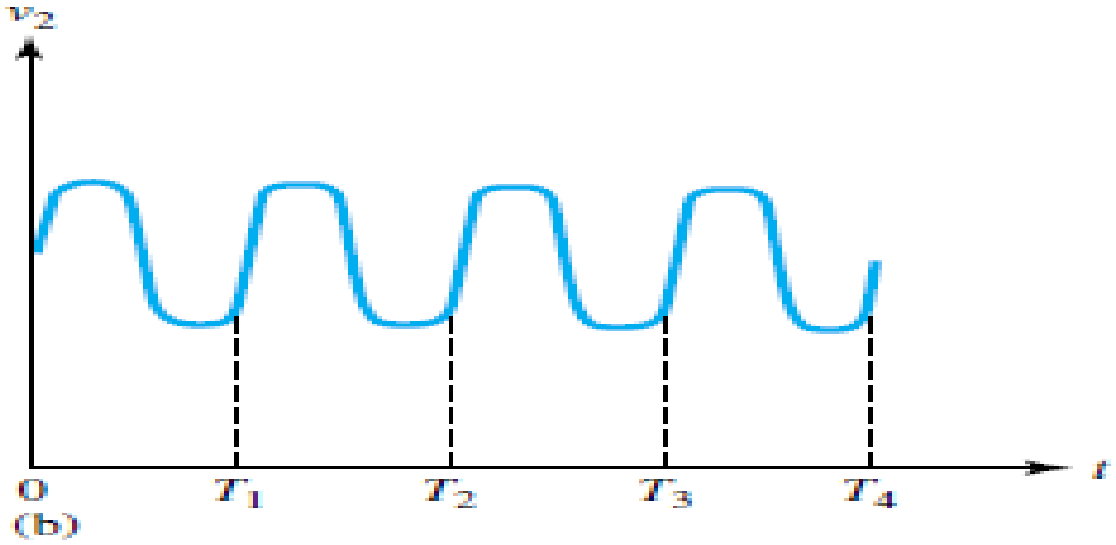
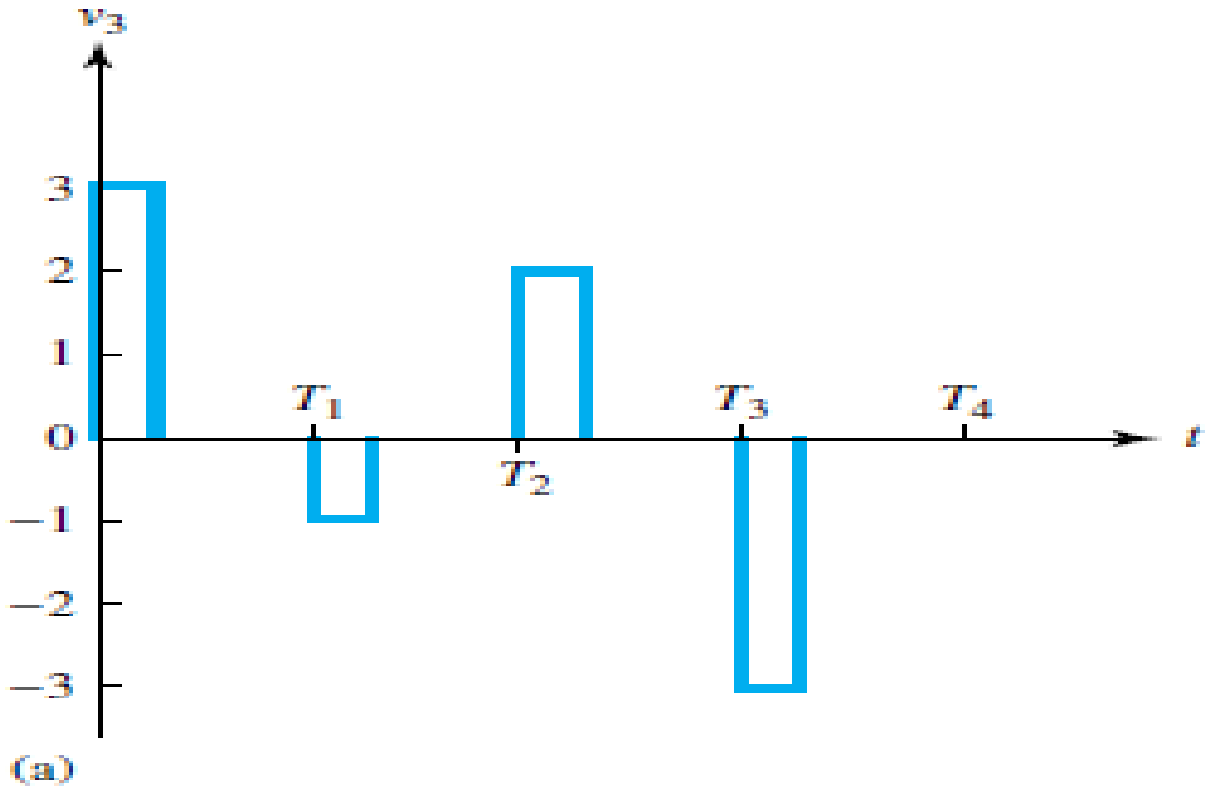


Figure · Continuous signals.

Types of signal: Aanalog and Digital signal

- A **digital signal** can only have a finite number of discrete amplitudes at any given time.
- Digital devices works only digital signal not analog signal.
- Greater accuracy



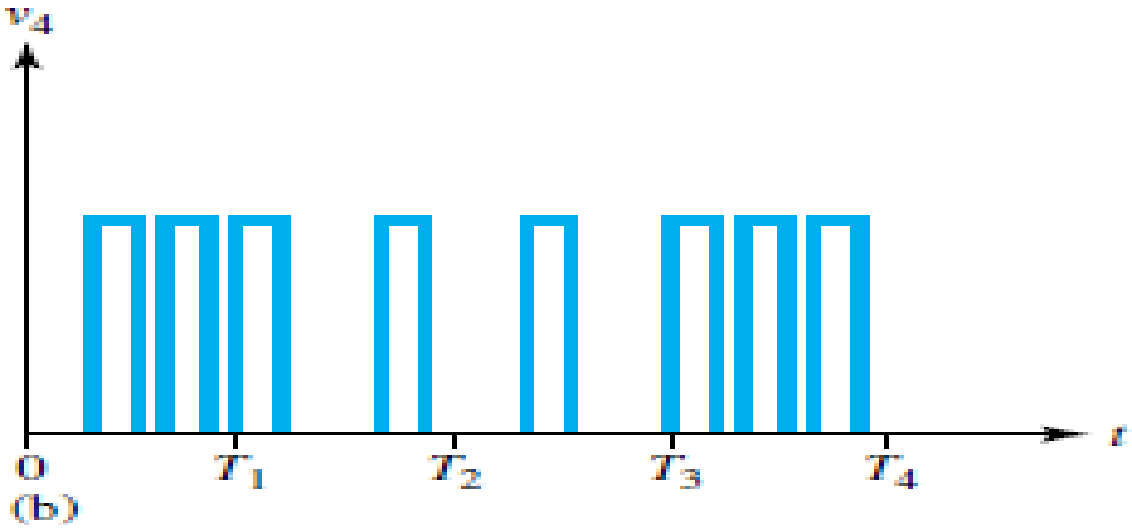


Figure Discrete signals.

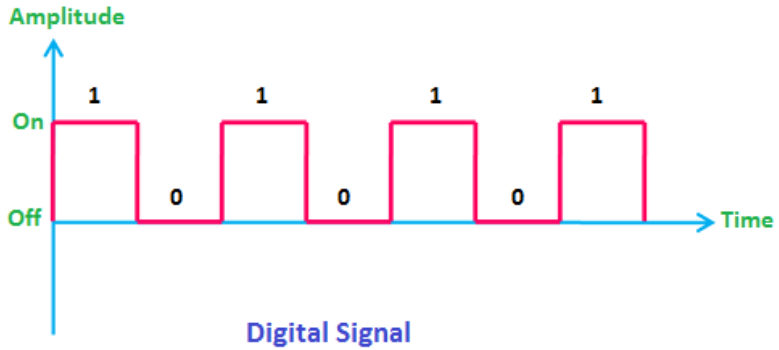
Types of signal: Aanalog and Digital signal

- The most common digital signals are ***binary signals***.

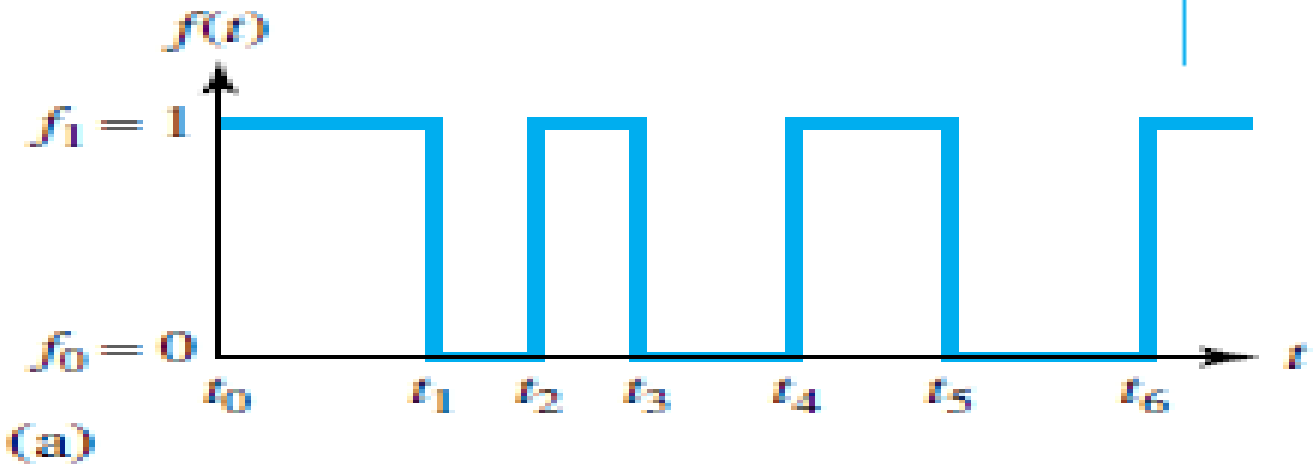
- A binary signal is a signal that can take only one of two discrete values and is therefore characterized by transitions **between two states**.

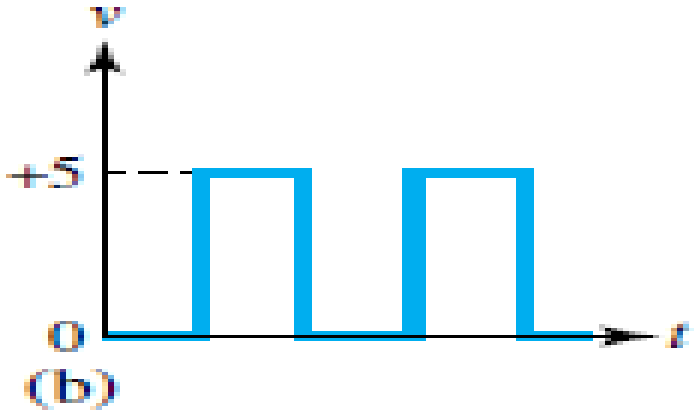
Marwadi University

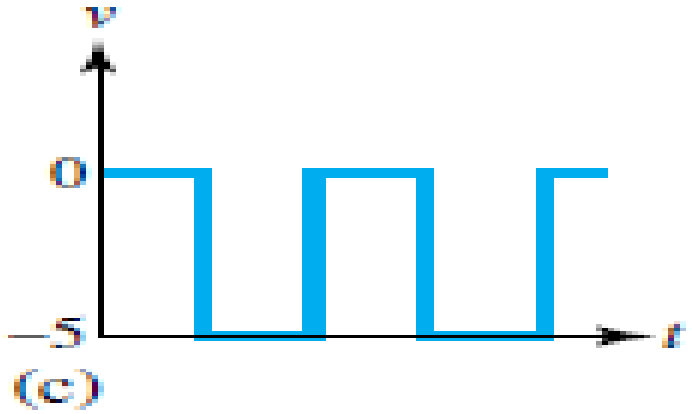
Types of signal: Digital signal



Digital Signal







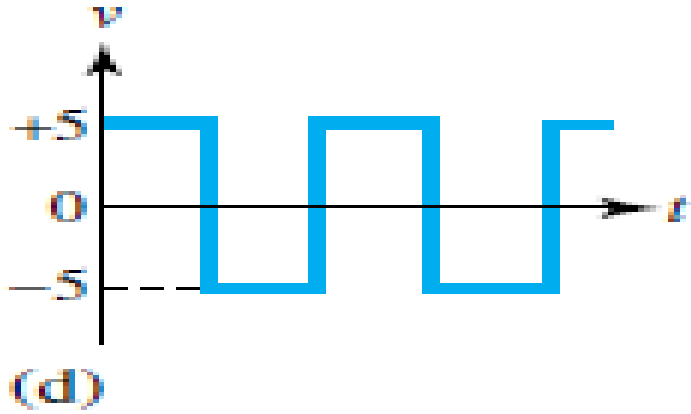


Figure Typical binary signals.

Difference between Analog and Digital signal

Comparison of Analog and Digital Signal

S.N.	Analog Signal	Digital Signal
1	Analog signal has a infinite values.	Digital signal has a finite number of the values.
2	Analog signal has a continuous nature.	Digital signal has a discrete nature.
3	Analog signal is generate by transducers and signal generators.	Digital signal is generate by A to D converter.
4	Example of analog signal: sine wave, triangular waves.	Example of digital signal: binary signal.

Example of Analog and Digital Devices

1) **Voltmeter** → analog or digital (digital voltmeter, or DVM)
2) **Speedometer of a car** → analog device,

3) **Odometer** (which records kilometres) → digital
4) **Toggle switch** → digital,

5) **Dimmer switch** → analog

Number System

Section - 1

Different number system with different number bases which plays very important role in today's computer world.

Image Source: https://learnabout-electronics.org/Digital/dig10.php

Marwadi University

Common Number Systems

System	Base/ Radix	Symbols	Used by Humans?	Used in Computers and digital circuit?
Decimal	10	0, 1, … 9	Yes	No
Binary	2	0, 1	No	Yes
Octal	8	0, 1, … 7	No	No
Hexa-decimal	16	0, 1, … 9, A, B, … F	No	No

- The base **(radix)** of the number system is the total number of digits in the system.
- Octal and Hexa decimal : Used to represent long binary pattern into compact form.

Marwadi University

DECIMAL	HEXADEC	OCTAL	BINARY
0	0	000	00000000
1	1	001	00000001
2	2	002	00000010
3	3	003	00000011
4	4	004	00000100
5	5	005	00000101
6	6	006	00000110
7	7	007	00000111
8	8	010	00001000
9	9	011	00001001
10	A	012	00001010
11	B	013	00001011
12	C	014	00001100
13	D	015	00001101
14	E	016	00001110
15	F	017	00001111
16	10	020	00010000
17	11	021	00010001
18	12	022	00010010
19	13	023	00010011
20	14	024	00010100
21	15	025	00010101
22	16	026	00010110
23	17	027	00010111
24	18	030	00011000
25	19	031	00011001

3 places

8 places

Some definition:

- Decimal→ Decimal point
- Binary→ Binary point
- Bit(**B**inary Dig**it**s→ 0 or 1
- Nibble→ Group of 4 bits→ 0010
- Byte→ Group of 8 bits→ 10101100
- word→ 16 bits

Count from 0 to 8 in radix 6 system.

Count from 0 to 8 in radix 3 system.

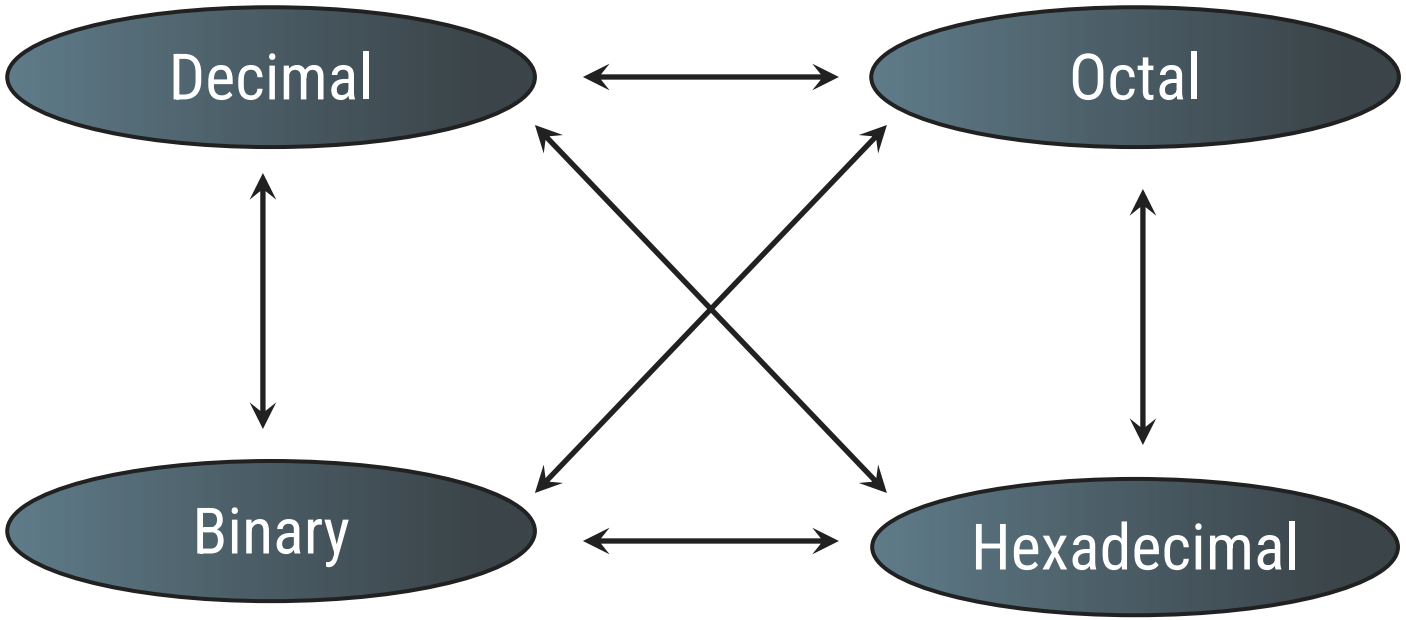
In decimal there are 0 to 25 numbers and in Hexa decimal there are 0 to 19 but in hexa between 9 and 10 there will be alphabets A to F.

Marwadi University

Conversion among Bases

- Possibilities

There are total 12 possibilities among convesion bases



- Example

$$25_{10} = 11001_2 = 31_8 = 19_{16}$$

Base

Marwadi University

1) Decimal to Binary

- Technique
  - Divide by two, keep track of the remainder
  - The remainders read from bottom to top give the equivalent binary integer number.

- Example - 1

$125_{10} = ?_2$

2	125	1
2	62	0
2	31	1
2	15	1
2	7	1
2	3	1
2	1	1
	0	

62 x 2 = 124 + 1 =125

$125_{10} = 1111101_2$

- Example - 2

$0.6875_{10} = ?2$

	integer		*fraction*
0.6875 x 2 = 1.3750	1	+	0.3750
0.3750 x 2 = 0.7500	0	+	0.7500
0.7500 x 2 = 1.5000	1	+	0.5000
0.5000 x 2 = 1.0000	1	+	0.0000

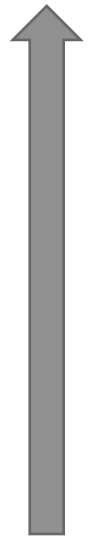
$0.6875_{10} = 0.1011_2$

Marwadi University

1) Decimal to Binary

❖The decimal number $(75)_{10}$ is converted into its binary equivalent:

	Quotient	Remainder	
75 ÷ 2 =	37	1	LSB
37 ÷ 2 =	18	1	
18 ÷ 2 =	9	0	
9 ÷ 2 =	4	1	
4 ÷ 2 =	2	0	
2 ÷ 2 =	1	0	
1 ÷ 2 =	0	1	MSB
	Stop		



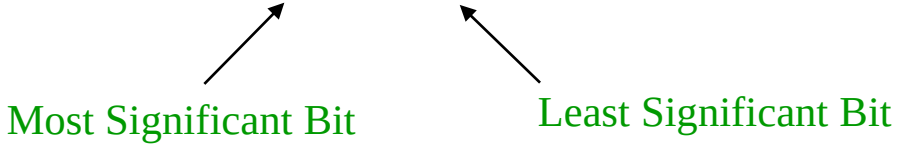
MSB : Most Significant Bit
LSB: Least Significant Bit

Thus, the binary number for $(75)_{10}$ is given by:

$(1001011)_2$

MSB LSB

- The left most bit in binary is called the Most Significant Bit (MSB)
- The rightmost bit in binary is called the Least Significant Bit (LSB)
- 11110

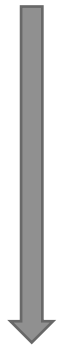


Marwadi University

1)Decimal to Binary

❖Conversion of fractional decimal number (0.4375)10 into its binary equivalent

$$
\begin{aligned}
0.4375 \times 2 &= 0.8750 \\
0.8750 \times 2 &= 1.7500 \\
0.7500 \times 2 &= 1.5000 \\
0.5000 \times 2 &= 1.0000 \\
&\text{Stop}
\end{aligned}
$$



$0.252_{10} = ?_2$

Thus, $(0.4375)_{10} = (.0111)_2$.

Accuracy in Binary Number Conversion

Example

- Convert $(0.252)_{10}$ to binary with an error less than 1%.

Solution

- Absolute value of allowable error is found by calculating 1% of the number

$$E_{allow} = 0.01\ X\ 0.252 = 0.00252_{10}$$

- Maximum error due to truncation is set to be less than allowable error by solving from $E_{10} = 2^{-n}$. This equation is written as

$$2^{-n} < 0.00252$$

- Inverting both sides of the inequality

$$2^{n} > 397$$

Accuracy in Binary Number Conversion

- Taking log of both sides and solving for n

$$n \log 2 = \log 397$$

$$n = \frac{\log 397}{\log 2} = 8.63 \approx 9 \ (next\ largest\ integer)$$

- This indicates that the use of 9 bits in the binary number will guarantee an error less than 1%.
- So the conversion is carried out to 9 places which results in

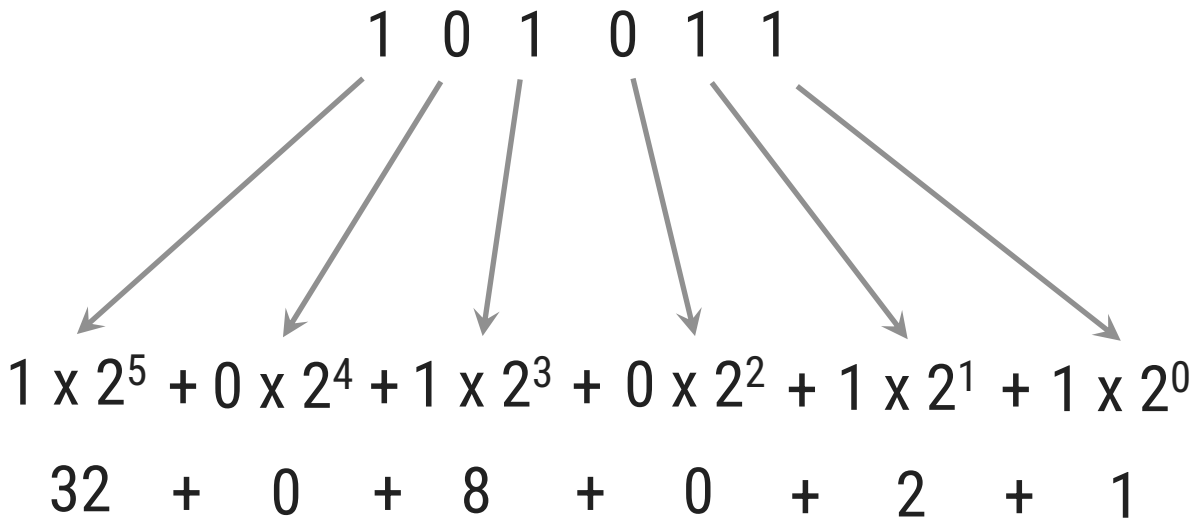
$$0.252_{10} = 0.010000001_2$$

2) Binary to Decimal

- Technique
  - Multiply each bit by 2^n, where *n* is the "weight" of the bit
  - The weight is the position of the bit, starting from 0 on the right(**Weighted form**). Finally, Add the results.

- Example - 1

  $101011_2 = ?_{10}$

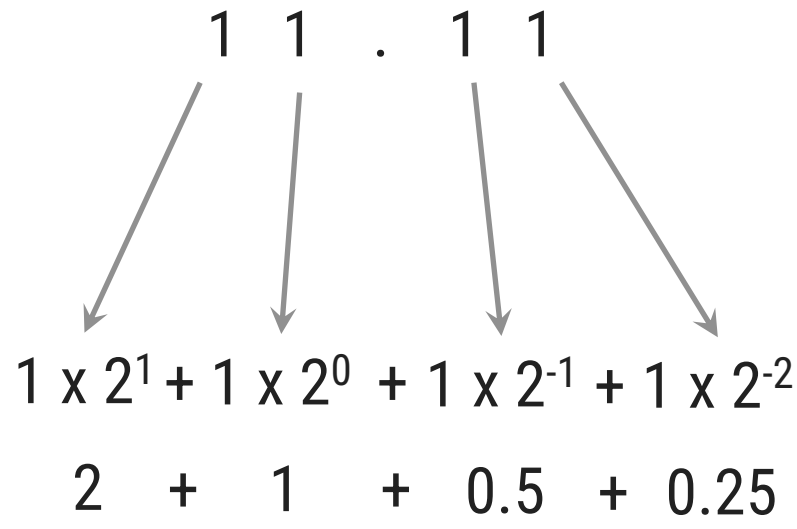


$101011_2 = 43_{10}$

- Example - 2

  $11.11_2 = ?_{10}$

In this time of system assume number system like integer and the decimal represents the addition of two numbers



$11.11_2 = 3.75_{10}$

Marwadi University

3)Decimal to Octal

- Technique
  - Divide by **eight**, keep track of the remainder
  - The remainders read from **bottom to top** give the equivalent octal integer number.

USE like euclid division lemma.

- Example - 1

$125_{10} = ?_8$

8	125	5
8	15	7
8	1	1
	0	

$125_{10} = 175_8$

- Example - 2

$0.6875_{10} = ?_8$

	integer		*fraction*
0.6875 x 8 = 5.5000	5	+	0.5000
0.5000 x 8 = 4.0000	4	+	0.0000

USE DUAL FACTOR .

$0.6875_{10} = 0.54_8$

Marwadi University

4) Octal to Decimal

- Technique
  - Multiply each digit by 8^n, where *n* is the "weight" of the digit
  - The weight is the position of the digit, starting from 0 on the right. Finally, Add the results.

- Example - 1

$724_8 = ?_{10}$

7 2 4

$7 \times 8^2 + 2 \times 8^1 + 4 \times 8^0$

$448 + 16 + 4$

$724_8 = 468_{10}$

- Example - 2

$43.25_8 = ?_{10}$

4 3 . 2 5

$4 \times 8^1 + 3 \times 8^0 + 2 \times 8^{-1} + 5 \times 8^{-2}$

$32 + 3 + 0.25 + 0.0781$

In this type the conversion is about octal to decimal so we have to first multiply by 8 and further we hav eto reduce power and then we have to addition.

$43.25_8 = 35.3281_{10}$

Marwadi University

5) Decimal to Hexa-Decimal

- Technique
  - Divide by sixteen, keep track of the remainder
  - The remainders read from bottom to top give the equivalent hexadecimal integer number.

- Example - 1

$1234_{10} = ?_{16}$

16	1234	2
16	77	13=D
16	4	4
	0	

$1234_{10} = 4D2_{16}$

- Example - 2

$0.03125_{10} = ?_{16}$

	integer		*fraction*
0.03125 x 16 = 0.5000	0	+	0.5000
0.5000 x 16 = 8.0000	8	+	0.0000

in this the conversion is about decimal to hexadecimal so we have to use dual factor also

$0.03125_{10} = 0.08_{16}$

Marwadi University

6) Hexa-Decimal to Decimal

- Technique
  - Multiply each digit by 16^n, where *n* is the "weight" of the digit
  - The weight is the position of the digit, starting from 0 on the right. Finally, Add the results.

- Example - 1

$ABC_{16} = ?_{10}$

A B C

$A \times 16^2 + B \times 16^1 + C \times 16^0$

$10 \times 16^2 + 11 \times 16^1 + 12 \times 16^0$

$2560 + 176 + 12$

$ABC_{16} = 2748_{10}$

- Example - 2

$43.25_{16} = ?_{10}$

4 3 . 2 5

in this we have to do do hexadecimal to decimal so we have to multiply with 16 and do further process which we have done in previous one

$\times 16^1 + 3 \times 16^0 + 2 \times 16^{-1} + 5 \times 16^{-2}$

$64 + 3 + 0.125 + 0.0195$

$43.25_{16} = 67.1445_{10}$

Marwadi University

6) Hexa-Decimal to Decimal

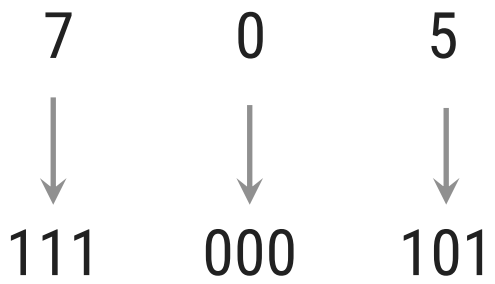
$$
\begin{aligned}
356_{16} &= 3 \times 16^2 + 5 \times 16^1 + 6 \times 16^0 \\
&= 768 + 80 + 6 \\
&= 854_{10}
\end{aligned}
$$

$$
\begin{aligned}
2AF_{16} &= 2 \times 16^2 + 10 \times 16^1 + 15 \times 16^0 \\
&= 512 + 160 + 15 \\
&= 687_{10}
\end{aligned}
$$

7) Octal to Binary

- Technique
  - Convert each octal digit to a 3-bit equivalent binary representation
- Example

$705_8 = ?_2$



$705_8 = 111000101_2$

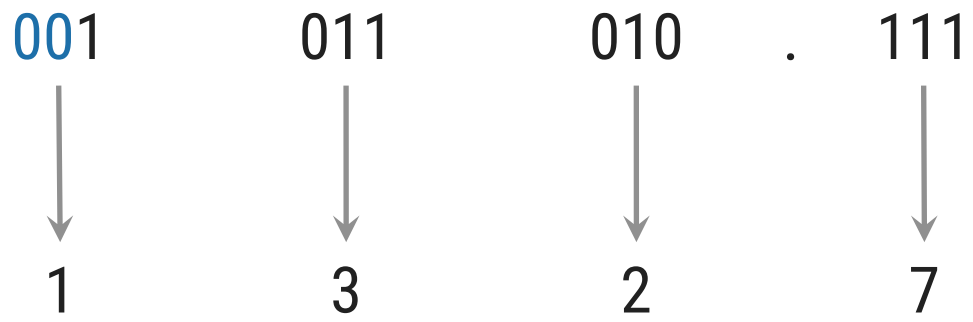
In this we have to use binary conversion table

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

8) Binary to Octal

- Technique
  - From given fractional point, group bits in threes to right and group bits in threes to left
  - If, left with less than 3 bits at the end then stuff 0s to make it group of three
  - Convert to octal digits
- Example

$1011010.111_2 = ?_8$



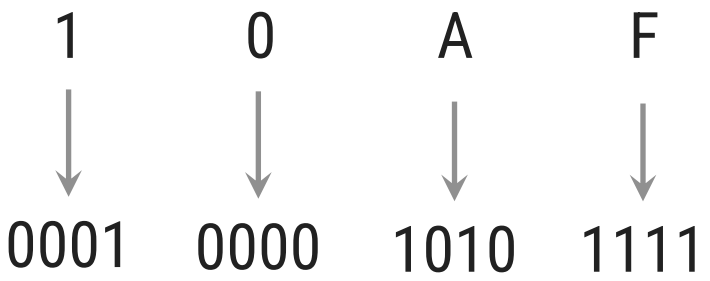
$1011010.111_2 = 132.7_8$

Marwadi University

9) Hexa-Decimal to Binary

- Technique
  - Convert each hexadecimal digit to a 4-bit equivalent binary representation
- Example

$10AF_{16} = ?_2$



$10AF_{16} = 10000101011111_2$

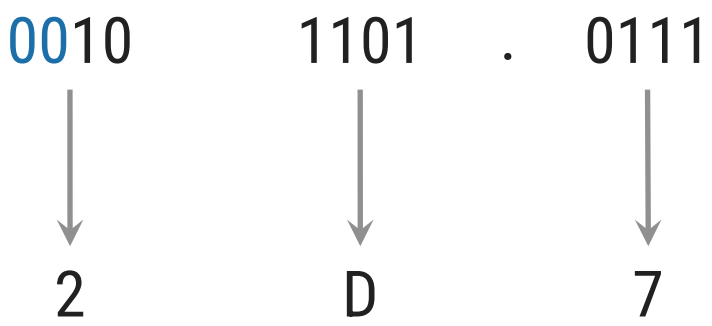
Hexa-Decimal	Binary	Hexa-Decimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	A	1010
3	0011	B	1011
4	0100	C	1100
5	0101	D	1101
6	0110	E	1110
7	0111	F	1111

same procedure we have to use which we have used in octal binary

10)Binary to Hexa-Decimal

- Technique
  - From given fractional point, group bits in fours to right and group bits in fours to left
  - If, left with less than 4 bits at the end then stuff 0s to make it group of four
  - Convert to hexadecimal digits
- Example

$101101.0111_2 = ?_{16}$



$1011010111_2 = 2D.7_{16}$

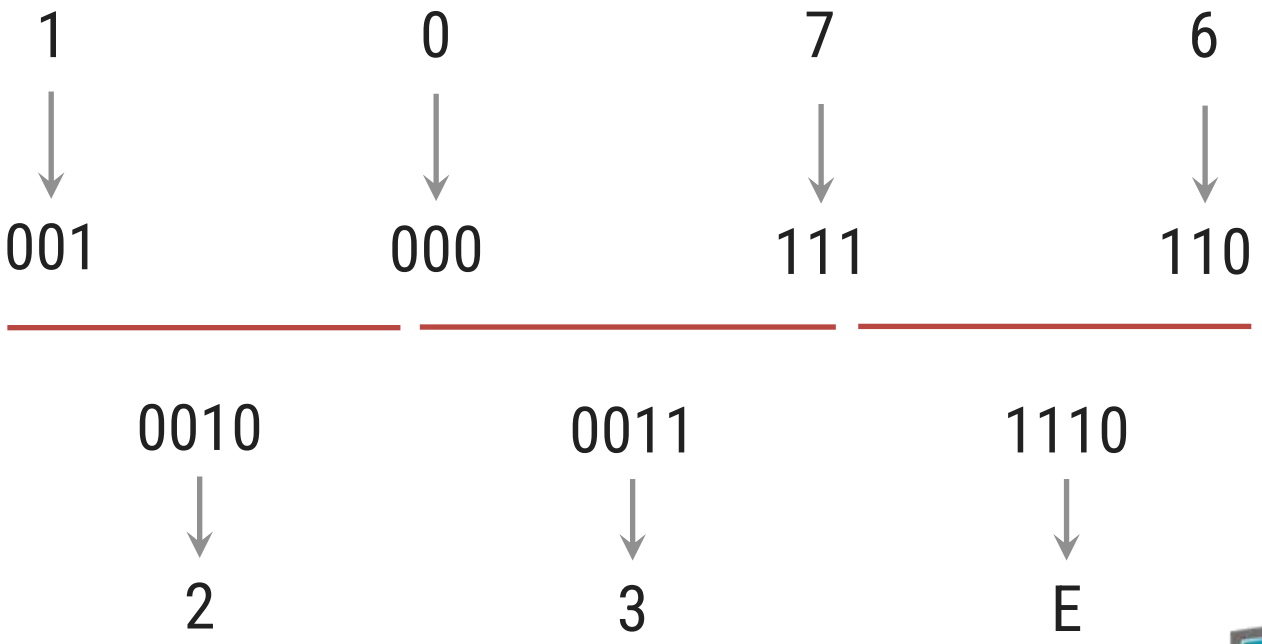
same procedure we have to use which we have used in octal binary

Marwadi University

11)Octal to Hexa-Decimal

- Technique
- Direct conversion method is not available for it.
  - Convert Octal to Binary
  - these bits are grouped in four bits.
  - Convert Binary to Hexa-Decimal
- Example

$1076_8 = ?_{16}$



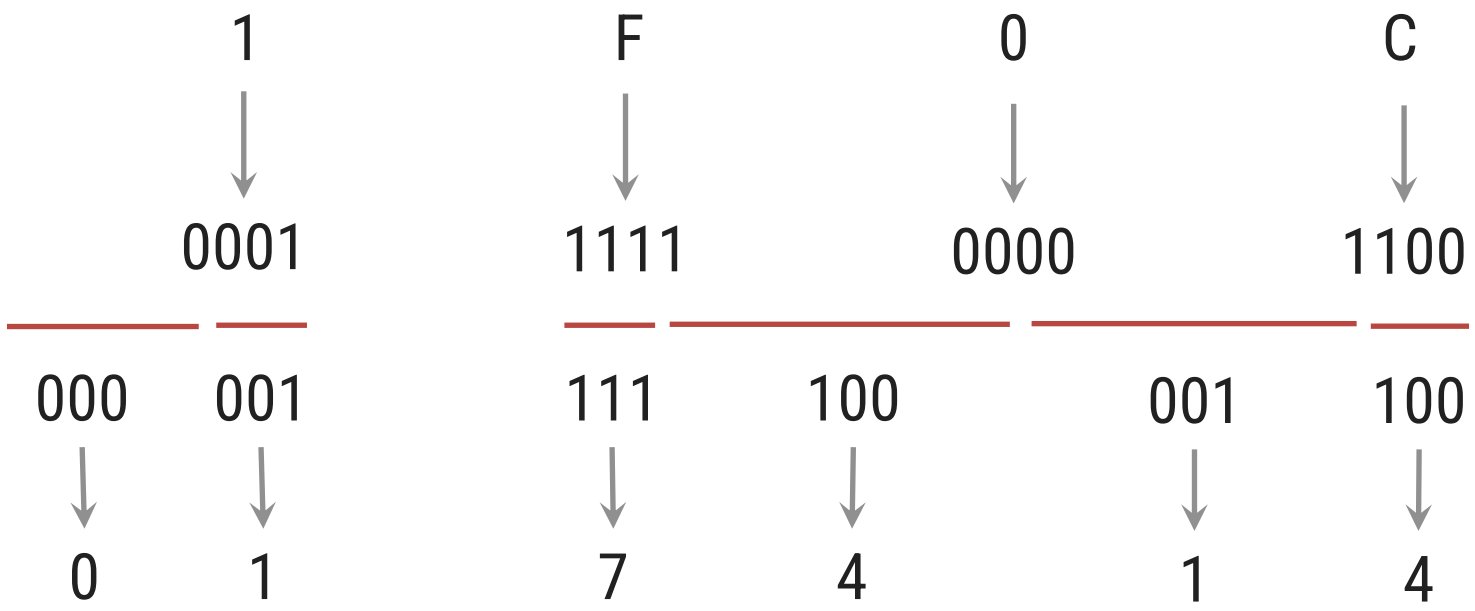
$1076_8 = 23E_{16}$

Marwadi University

12)Hexa-Decimal to Octal

- Technique
  - Convert Hexa-Decimal to Binary
  - these bits are grouped in three bits.
  - Convert Binary to Octal
- Example

  $1F0C_{16} = ?_8$



$1F0C_{16} = 17414_8$

Marwadi University

Exercise – Convert ...

Decimal	Binary	Octal	Hexa-decimal
33			
	1110101		
		703	
			1AF

Exe

Answer

Decimal	Binary	Octal	Hexa-decimal
33	100001	41	21
117	1110101	165	75
451	111000011	703	1C3
431	110101111	657	1AF

Marwadi University

Exercise – Convert ...

Decimal	Binary	Octal	Hexa-decimal
29.8			
	101.1101		
		3.07	
			C.82

Don’t use a calculator!

Marwadi University

Exe

Answer

Decimal	Binary	Octal	Hexa-decimal
29.8	11101.110011…	35.63…	1D.CC…
5.8125	101.1101	5.64	5.D
3.109375	11.000111	3.07	3.1C
12.5078125	1100.10000010	14.404	C.82

Marwadi University

Floating Point Representation

- Scientific notation for Decimal number system to represent very large or small number is ,
  1) 976,000,000,000,000 = 9.76 * 10^(+14)
  2) 0.0000000000000976 = 9.76 * 10^(-14).

- **Decimal point** is put on a convenient location and use the **exponent of 10 to keep track of that decimal point**

- This allows a range of very large and very small numbers to be represented with only a few digits.

• **Normalized Floating point** number: if most significant digit of the number is non-zero. i.e. **350 is normalized but 00035 is not normalized**

Floating Point Representation

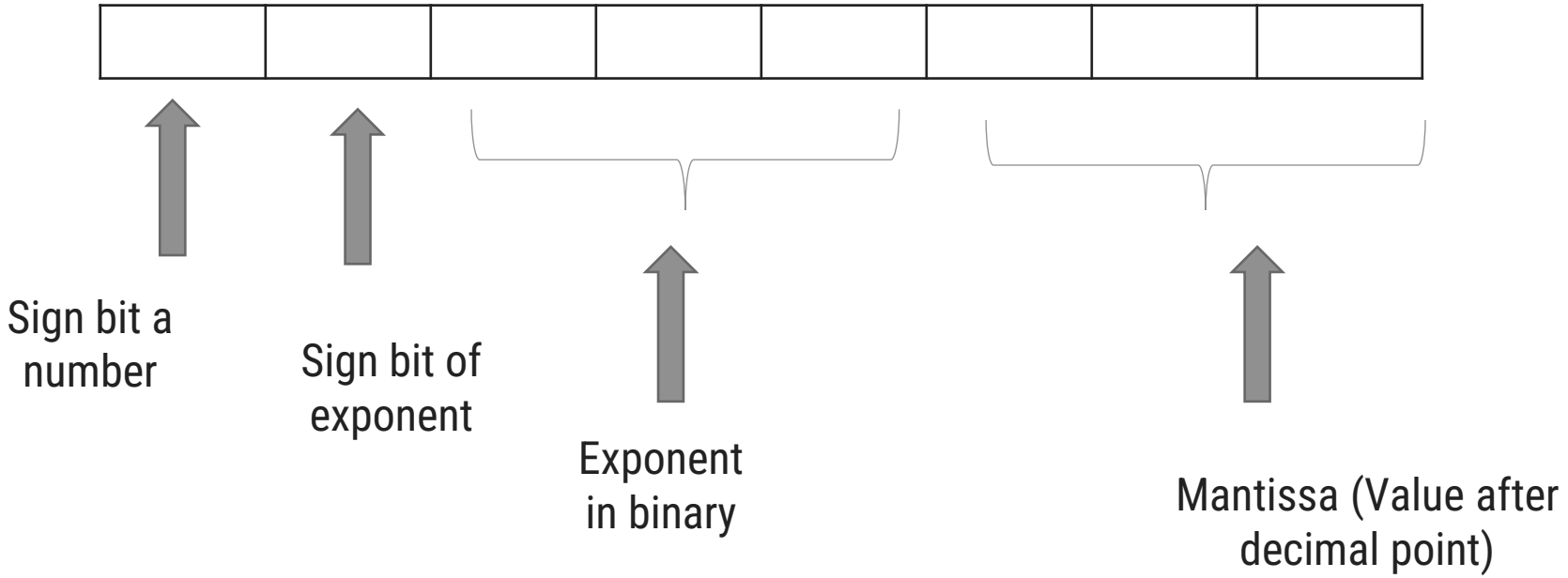
- **Scientific notation** of **binary number is Floating point representation.**

- A floating-point (FP) number :

$$\pm m^{*} b^{e}$$

1. **m = mantissa - represents the fraction part of the number**,
2. **e** = **exponent, :- the position of the decimal(binary) point**
3. **b** = base (radix) of the exponent.

Floating point Representation

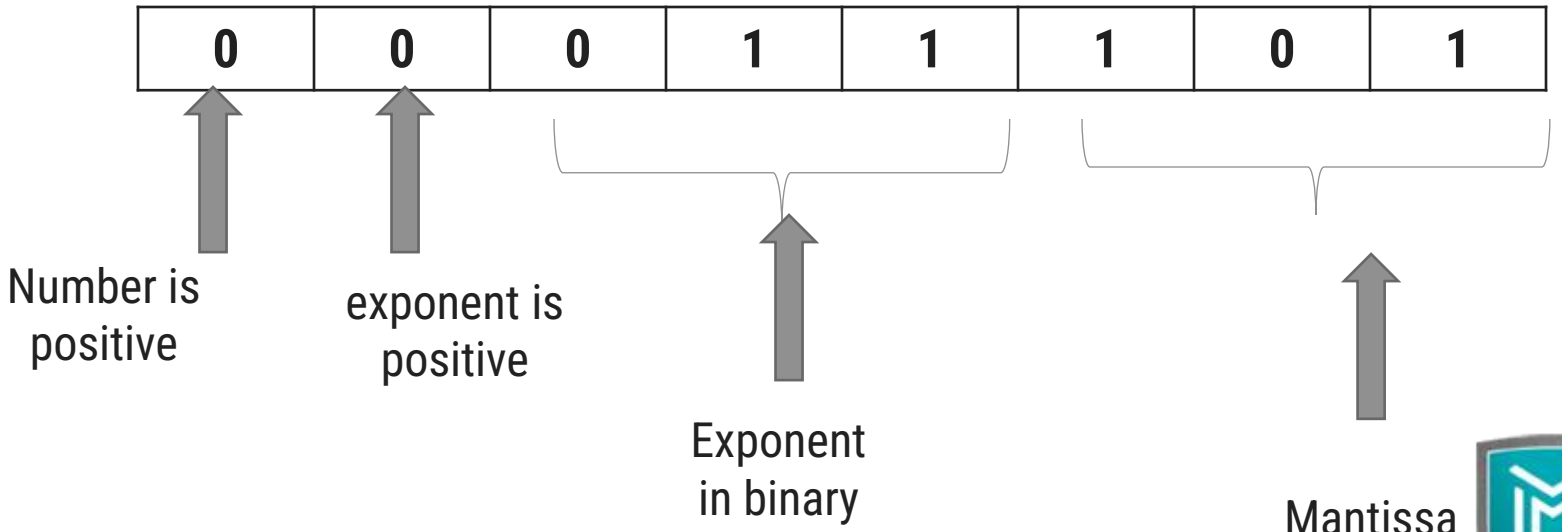


Representation of number in floating point method

- $(13.8)_{10} = (1101.11001\ldots)_2$

$= (1.10111001..) * 2^3$

$= (1.1011) * 2^{(011)_2}$



Example -2

- $(-13.9)_{10} = -(1101 . 11100\ldots)_2$

  $= -(1.10111100) * 2^3$

- $= -(1.10111100) * 2^{(011)_2}$

- $= -(1.1011) * 2^{(011)_2}$

1	0	0	1	1	1	0	1

Floating Point Representation

- Two main Standard:
    1) ANSI(**A**merican **N**ational **S**tandard **I**nstitute)
    2) IEEE(**I**nstitute of **E**lectrical and **E**lectronics **E**ngineers)

32 bit floating point number in ANSI(American National Standard Institute)

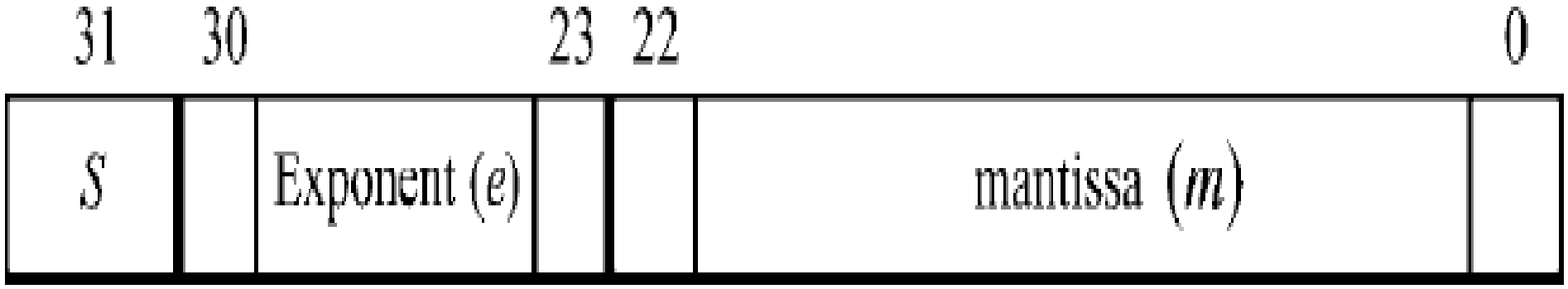


Figure Representation of a floating-point number

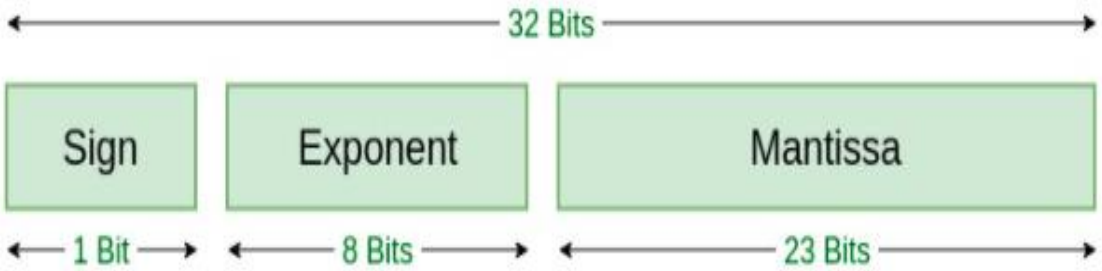
Floating Point Representation

- The most commonly used format for representing floating-point numbers is the IEEE-754 standard.
- Established in 1985 by the Institute of Electrical and Electronics Engineers (IEEE).

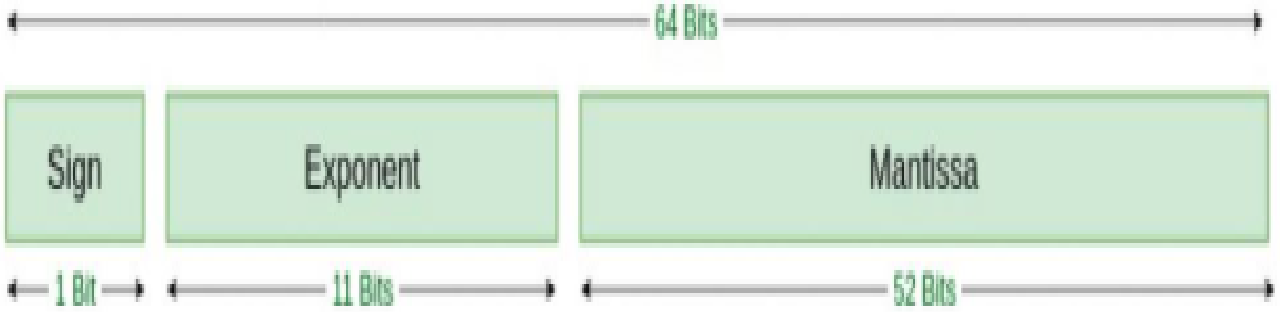
Characteristic parameters of IEEE-754 formats.

Precision	Sign (bits)	Exponent (bits)	Mantissa (bits)	Total length (bits)
Single	1	8	23	32
Single-extended	1	≥ 11	≥ 32	≥ 44
Double	1	11	52	64
Double-extended	1	≥ 15	≥ 64	≥ 80

Floating Point Representation- IEEE-754



Single Precision
IEEE 754 Floating-Point Standard



Double Precision
IEEE 754 Floating-Point Standard

Marwadi University

Binary Addition & Subtraction

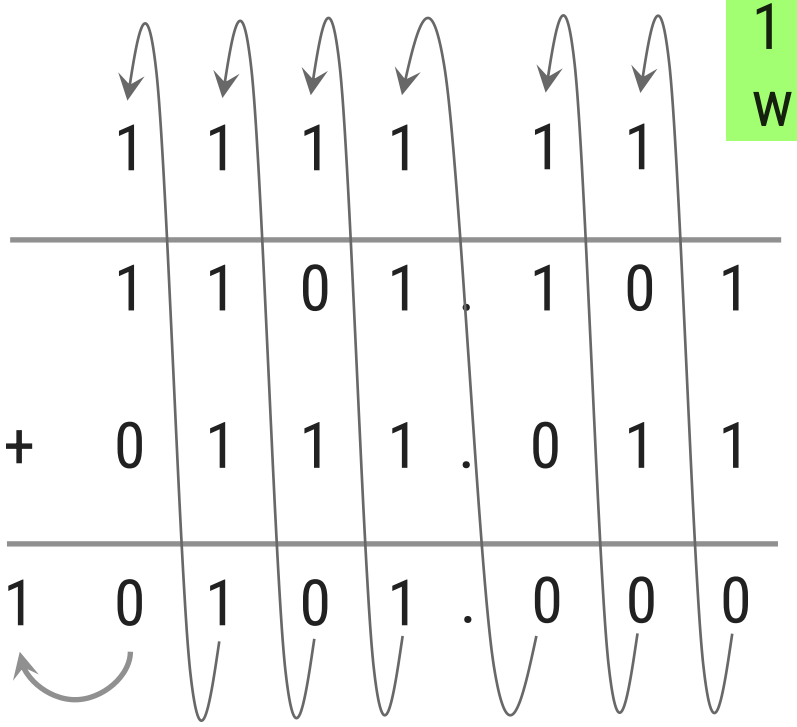
- Rules for binary addition

0 + 0 = 0

0 + 1 = 1

1 + 0 = 1

1 + 1 = 10 i.e. 0 with a carry of 1



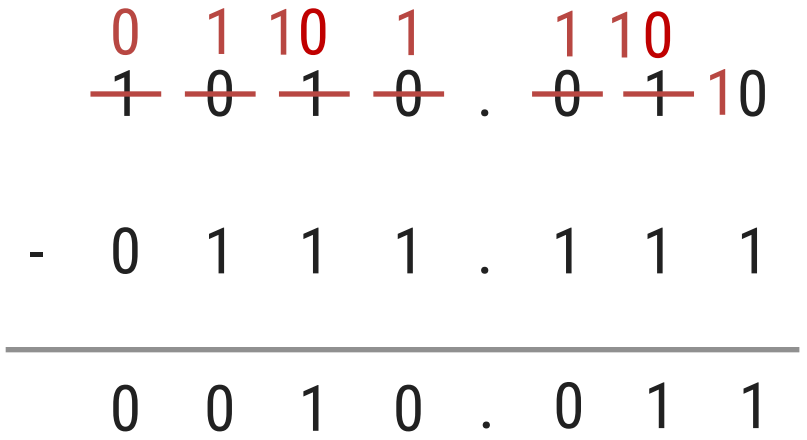
- Rules for binary subtraction

0 – 0 = 0

1 – 1 = 0

1 – 0 = 1

0 – 1 = 1, with a borrow 1



Binary Multiplication & Division

- Multiplication

```
          1 0 1 1 1
      x   1 0 0 1 1
  ---------------------
          1 0 1 1 1
        1 0 1 1 1
      0 0 0 0 0
    0 0 0 0 0
  1 0 1 1 1
---------------------
1 1 0 1 1 0 1 0 1
```

- Division

```
110 | 1 0 1 1 0 1 | 0 1 1 1 . 1
      0 0 0
    ---------
      1 0 1 1
        1 1 0
      -------
      1 0 1 0
        1 1 0
      -------
        1 0 0 1
          1 1 0
        -------
            1 1 0
            1 1 0
          -------
              0 0 0
```

Marwadi University

Signed Binary Numbers

- Two ways of representing signed numbers:
  - 1) Sign-magnitude form, 2) Complement form.
- Most of computers use complement form for negative number notation.
- 1’s complement and 2’s complement are two different methods in this type.

1’s Complement

- 1’s complement of a binary number is obtained by subtracting each digit of that binary number from 1.
- Example

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

(1’s complement of 1101)

```
  1 1 1 . 1 1
- 1 0 1 . 0 1
-------------
  0 1 0 . 1 0
```

(1’s complement of 101.01)

Shortcut: Invert the numbers from 0 to 1 and 1 to 0

Marwadi University

2's Complement

- 2's complement of a binary number is obtained by adding 1 to its 1's complement.
- Example

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

(2's complement of 1100)

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

(2's complement of 101.01)

Shortcut: Starting from right side, all bits are same till first 1 occurs and then invert rest of the bits

Marwadi University

Representation of negative number in 2's complement form

- Express -65.5 in 12 bit 2's complement form.

2	65	1
2	32	0
2	16	0
2	8	0
2	4	0
2	2	0
2	1	1
	0	

$0.5 \times 2 = 1.0$

So, result in 12-bit binary is as follows:

$65.5_{10} = 01000001.1000_2$

For negative number, we have to convert this into 2's complement form

$-65.5_{10} = 10111110.1000_2$

Accuracy in Binary Number Conversion

Example

- Convert $(0.252)_{10}$ to binary with an error less than 1%.

Solution

- Absolute value of allowable error is found by calculating 1% of the number

$$E_{allow} = 0.01\ X\ 0.252 = 0.00252_{10}$$

- Maximum error due to truncation is set to be less than allowable error by solving from $E_{10} = 2^{-n}$. This equation is written as

$$2^{-n} < 0.00252$$

- Inverting both sides of the inequality

$$2^{n} > 397$$

Accuracy in Binary Number Conversion

- Taking log of both sides and solving for n

$$n \log 2 = \log 397$$

$$n = \frac{\log 397}{\log 2} = 8.63 \approx 9 \ (next\ largest\ integer)$$

- This indicates that the use of 9 bits in the binary number will guarantee an error less than 1%.
- So the conversion is carried out to 9 places which results in

$$0.252_{10} = 0.010000001_2$$

9's Complement

- 9's complement of a decimal number is obtained by subtracting each digit of that decimal number from 9.
- Example

9 9 9 9	9 9 9 . 9 9
- 3 4 6 5	- 7 8 2 . 5 4
6 5 3 4	2 1 7 . 4 5
(9's complement of 3465)	(9's complement of 782.54)

10’s Complement

- 10’s complement of a decimal number is obtained by adding 1 to its 9’s complement.
- Example

	9	9	9	9			9	9	9	.	9	9
-	3	4	6	5		-	7	8	2	.	5	4
	6	5	3	4			2	1	7	.	4	5
+				1		+						1
	6	5	3	5			2	1	7	.	4	6

(10’s complement of 3465) (10’s complement of 782.54)

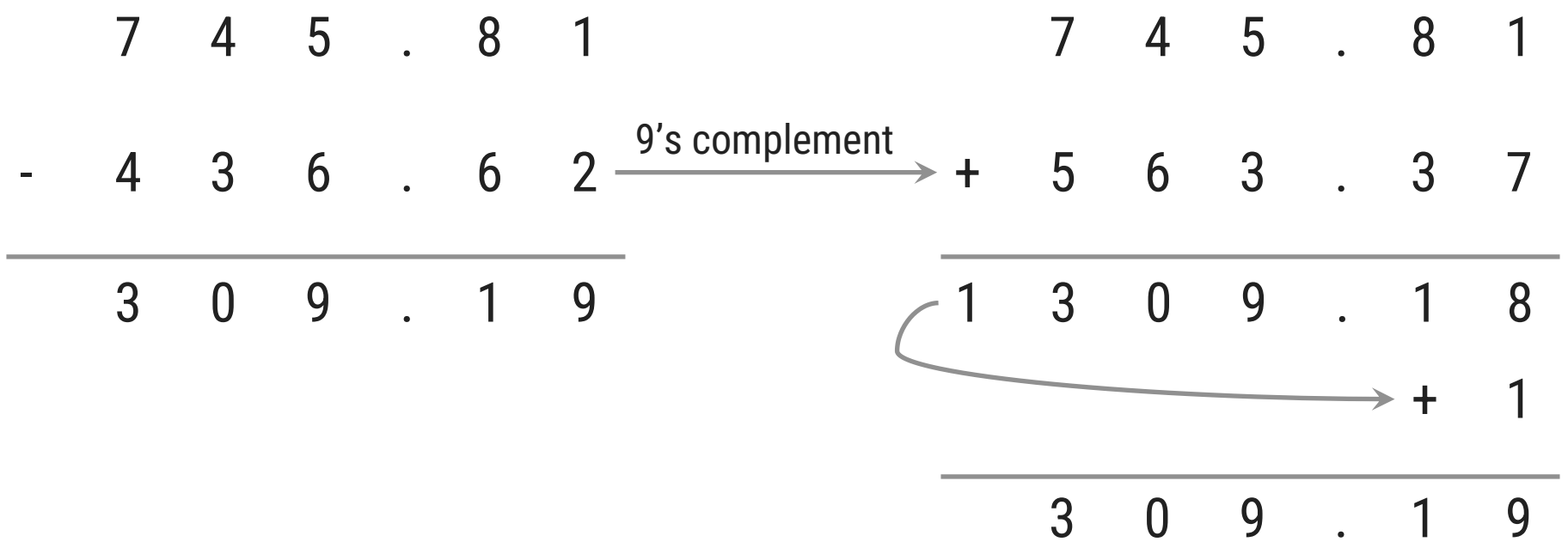
Subtraction using 9's complement & 10's complement

- Using 9's complement
  - Obtain 9's complement of subtrahend
  - Add the result to minuend and call it intermediate result
  - If carry is generated then answer is positive and add the carry to Least Significant Digit (LSD)
  - If there is no carry then answer is negative and take 9's complement of intermediate result and place negative sign to the result.

- Using 10's complement
  - Obtain 10's complement of subtrahend
  - Add the result to minuend
  - If carry is generated then answer is positive, ignore carry and result itself is answer
  - If there is no carry then answer is negative and take 10's complement of intermediate result and place negative sign to the result.

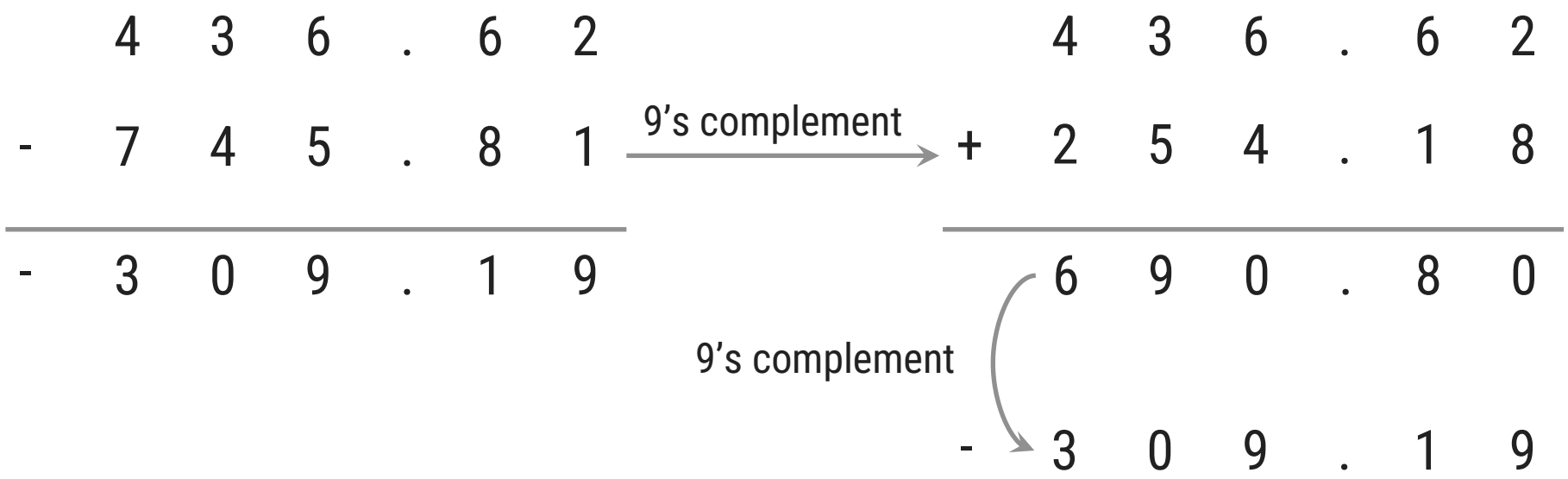
Subtraction using 9's complement (Examples)

- Example - 1
  745.81 – 436.62



Subtraction using 9's complement (Examples)

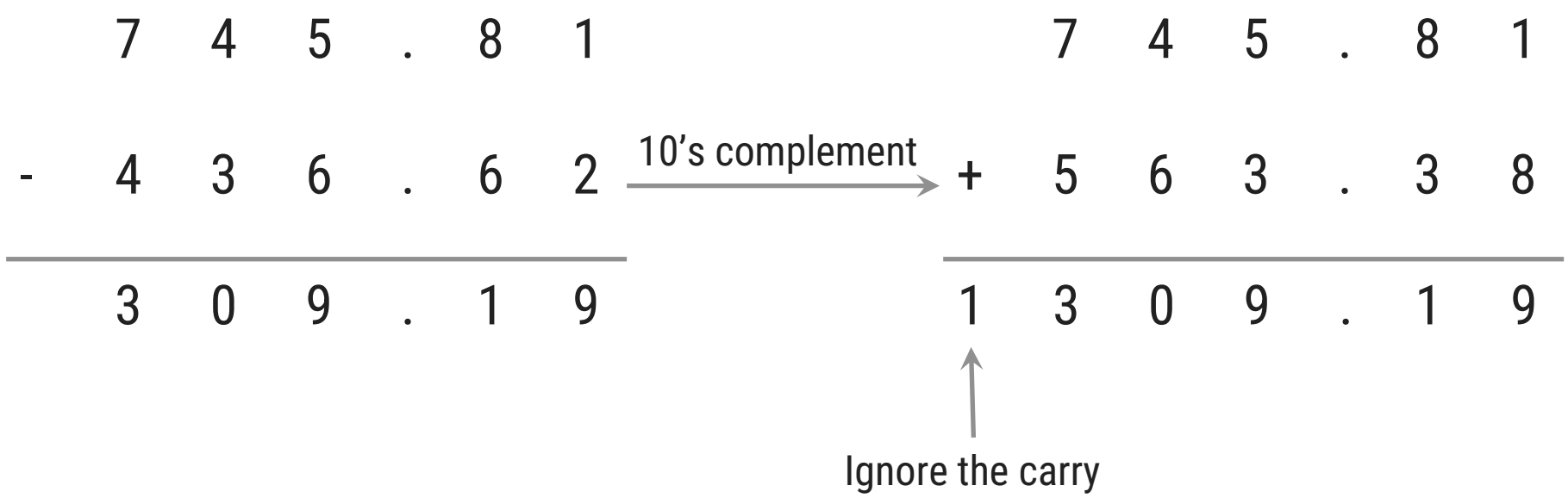
- Example - 2
  436.62 - 745.81



As carry is not generated, so take 9's complement of the intermediate result and add ' – ' sign to the result

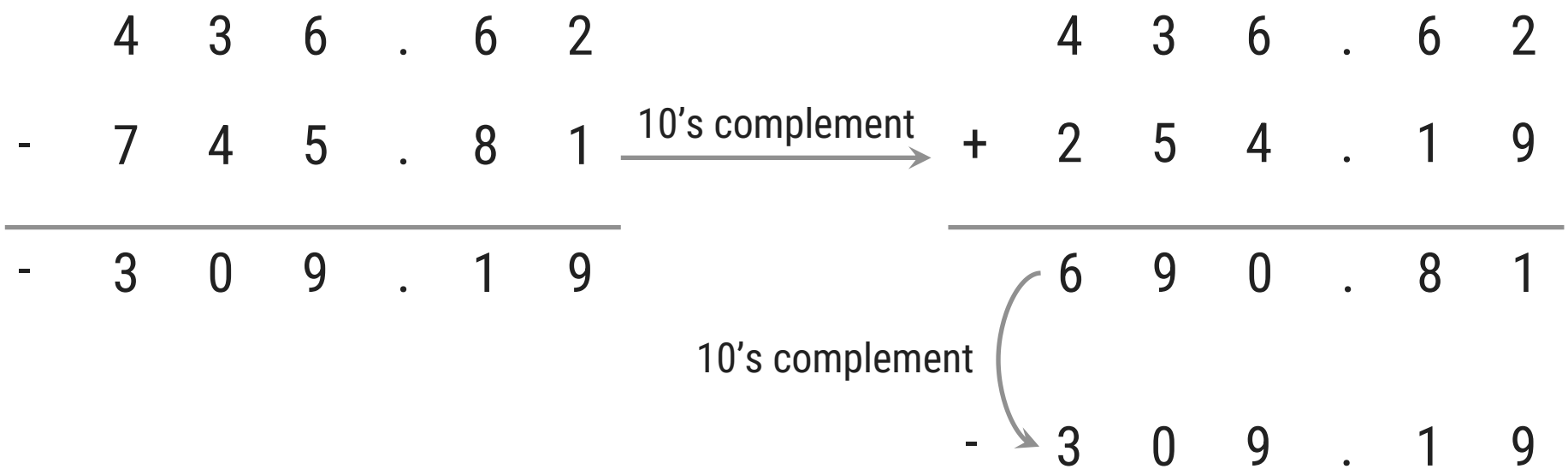
Subtraction using 10’s complement (Examples)

- Example - 1
  745.81 – 436.62



Subtraction using 10's complement (Examples)

- Example - 2
  436.62 - 745.81



As carry is not generated, so take 10's complement of the intermediate result and add ' – ' sign to the result

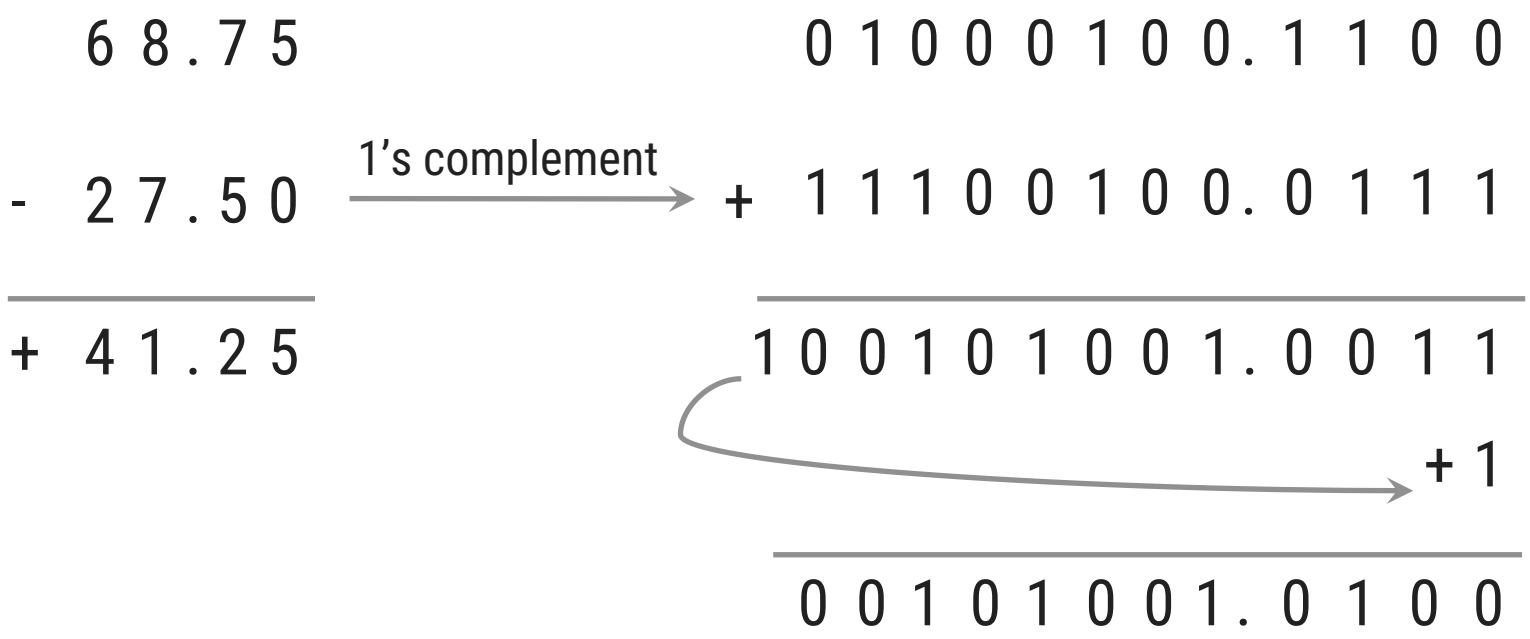
Subtraction using 1's complement & 2's complement

- Using 1's complement
  - Obtain 1's complement of subtrahend
  - Add the result to minuend and call it intermediate result
  - If carry is generated then answer is positive and add the carry to Least Significant Digit (LSD)
  - If there is no carry then answer is negative and take 1's complement of intermediate result and place negative sign to the result.

- Using 2's complement
  - Obtain 2's complement of subtrahend
  - Add the result to minuend
  - If carry is generated then answer is positive, ignore carry and result itself is answer
  - If there is no carry then answer is negative and take 2's complement of intermediate result and place negative sign to the result.

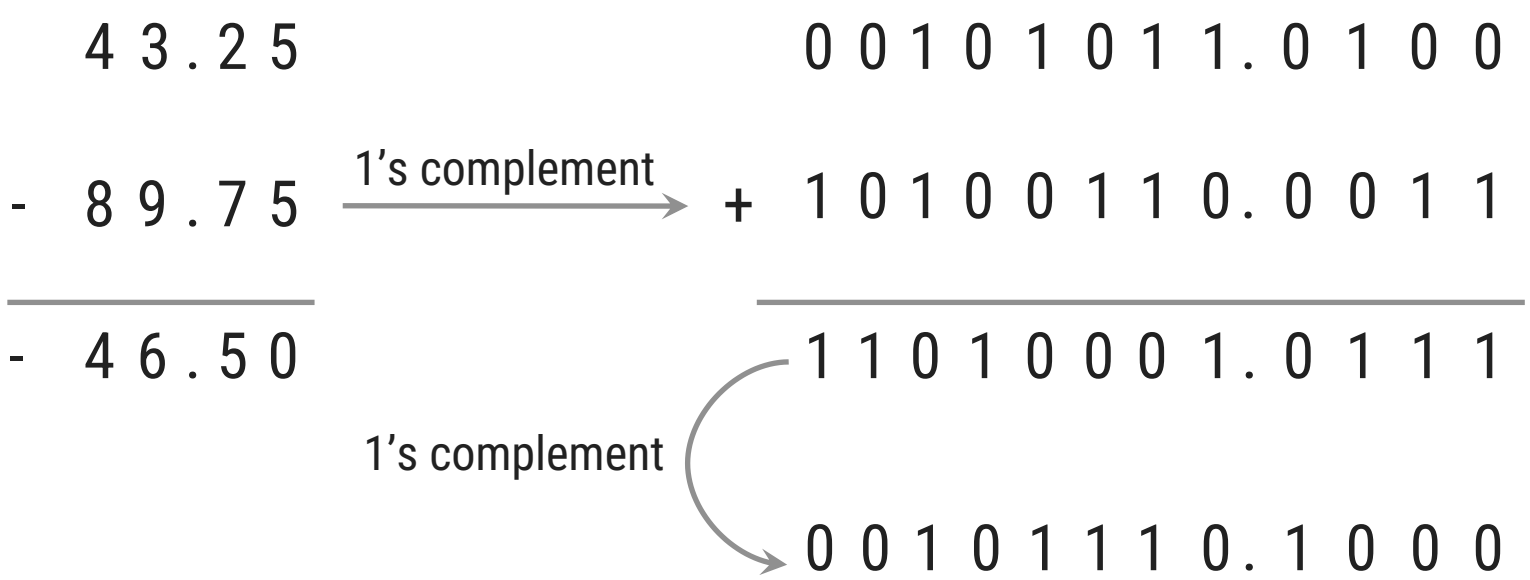
Subtraction using 1's complement (Examples)

- Example - 1
  68.75 – 27.50



Subtraction using 1's complement (Examples)

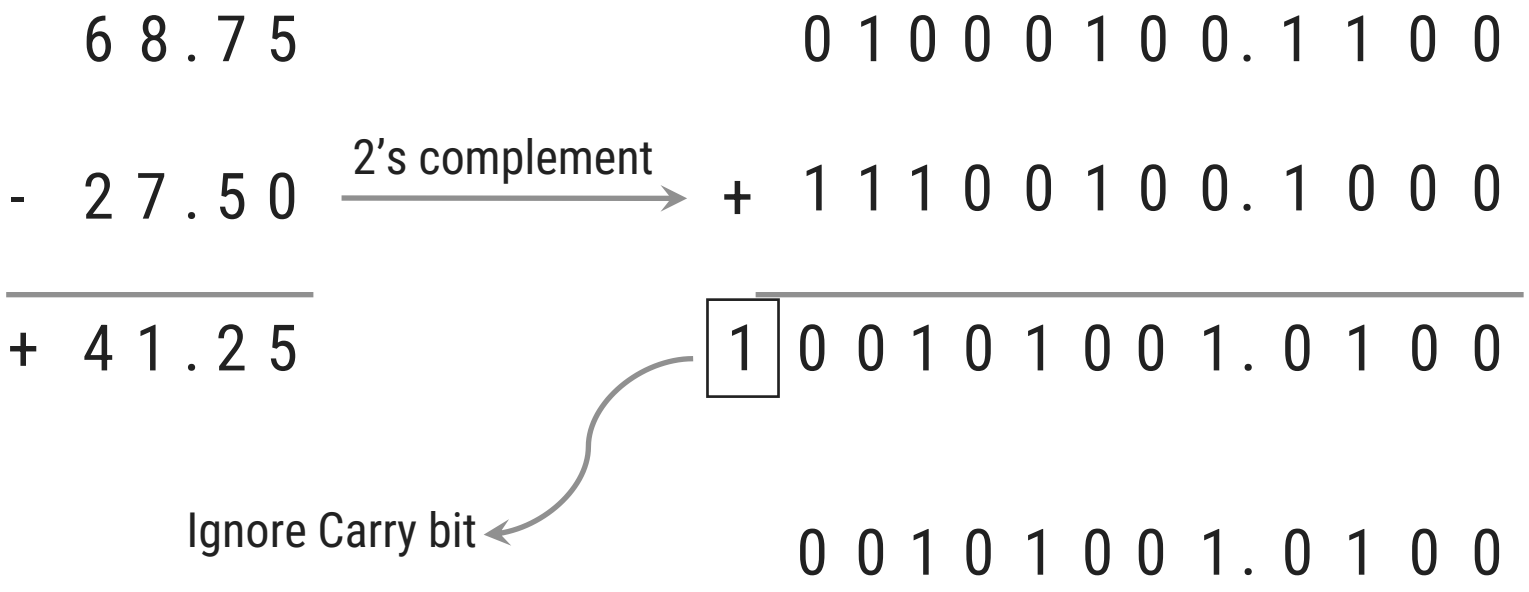
- Example - 2
  43.25 - 89.75



As carry is not generated, so take 1's complement of the intermediate result and add ' – ' sign to the result

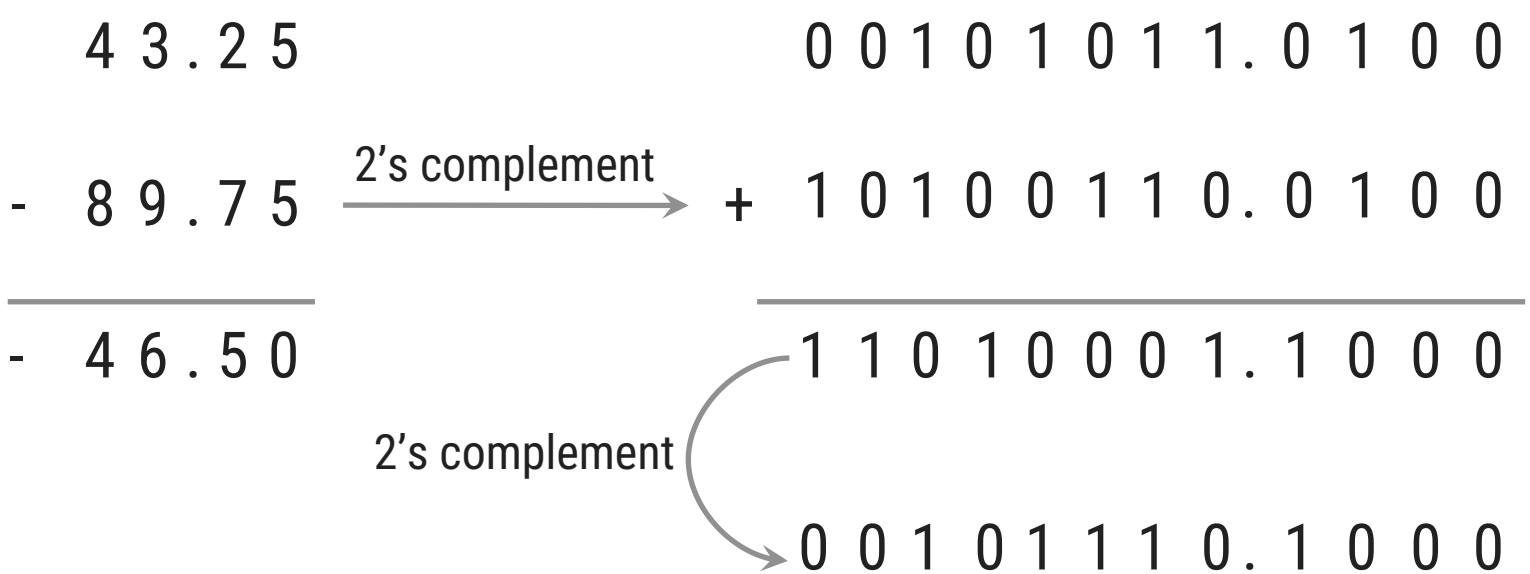
Subtraction using 2's complement (Examples)

- Example - 1
  68.75 – 27.50



Subtraction using 2's complement (Examples)

- Example - 2
  43.25 - 89.75



As carry is not generated, so take 2's complement of the intermediate result and add ' – ' sign to the result

Binary Codes

Section - 2

Marwadi University

DECIMAL	HEXADEC	OCTAL	BINARY
0	0	000	00000000
1	1	001	00000001
2	2	002	00000010
3	3	003	00000011
4	4	004	00000100
5	5	005	00000101
6	6	006	00000110
7	7	007	00000111
8	8	010	00001000
9	9	011	00001001
10	A	012	00001010
11	B	013	00001011
12	C	014	00001100
13	D	015	00001101
14	E	016	00001110
15	F	017	00001111
16	10	020	00010000
17	11	021	00010001
18	12	022	00010010
19	13	023	00010011
20	14	024	00010100
21	15	025	00010101
22	16	026	00010110
23	17	027	00010111
24	18	030	00011000
25	19	031	00011001

8421 BCD Code (Natural BCD Code)

- Each decimal digit, 0 through 9, is coded by 4-bit binary number
- 8, 4, 2 and 1 weights are attached to each bit
- BCD code is weighted code
- 1010, 1011, 1100, 1101, 1110 and 1111 are illegal codes
- Less efficient than pure binary
- Arithmetic operations are more complex than in pure binary
- Example

Decimal	1	4
BCD	0001	0100
Binary	1110	

Binary Codes

Decimal	Binary	BCD
0	0	0000
1	1	0001
2	10	0010
3	11	0011
4	100	0100
5	101	0101
6	110	0110
7	111	0111

Decimal	Binary	BCD
8	1000	1000
9	1001	1001
10	1010	0001 0000
11	1011	0001 0001
12	1100	0001 0010
13	1101	0001 0011
14	1110	0001 0100
15	1111	0001 0101

Marwadi University

BCD Addition

- Example - 1

	2	5
+	1	3
	3	8

					1	1	1	
	0	0	1	0	0	1	0	1
+	0	0	0	1	0	0	1	1
	0	0	1	1	1	0	0	0

No carry, no illegal code. So, this is the correct sum.

Rule: If there is an illegal code or carry is generated as a result of addition, then add 0110 to particular that 4 bits of result.

BCD Addition

- Example - 2

		1	111		
	679.6	0110	0111	1001	.0110
+	536.8	+ 0101	0011	0110	.1000
	1216.4	1011	1010	1111	.1110
		+0110	+0110	+0110	+.0110
		10001	10000	10101	1.0100
		+1	+1	+1	+1
		0001	0010	0001	0110 .0100

All are illegal codes

Add 0110 to each

Propagate carry

Corrected sum

BCD Subtraction

- Example - 1

	3	8		0	0	1	1	1	0	0	0
-	1	5	-	0	0	0	1	0	1	0	1
	2	3		0	0	1	0	0	0	1	1

No borrow. So, this is the correct difference.

Rule: If one 4-bit group needs to take borrow from neighbor, then subtract 0110 from the group which is receiving borrow.

Marwadi University

BCD Subtraction

- Example - 2

	206.7	0010	0000	0110	.0111	
-	147.8	- 0001	0100	0111	.1000	
	58.9	0000	1011	1110	.1111	Borrows are present
			-0110	-0110	-.0110	Subtract 0110
			0101	1000	.1001	Corrected difference

Excess Three (XS-3) Code

- Excess Three Code = 8421 BCD + 0011(3)
- XS-3 code is non-weighted BCD code
- Also known as self complementing code
- 0000, 0001, 0010, 1101, 1110 and 1111 are illegal codes
- Example

Decimal	1	4
BCD	0001	0100
XS-3	0100	0111

XS-3 Addition

- Example

	247.6	0101	0111	1010	.1001
+	359.4	+ 0110	1000	1100	.0111
	607.0	1011	1111	10110	1.0000
				+1 +1	
		1011	10000	0111	.0000
				+1	
		1100	0000	0111	.0000
		- 0011	+0011	+0011	+.0011
		1001	0011	1010	.0011

Carry generated

Propagate carry

Corrected Sum in XS-3

Rule: Add 0011 to group which generated carry and Subtract 0011 to group which do not generated carry

Marwadi University

XS-3 Subtraction

- Example

	57.6	1000	1010	.1001
-	27.8	- 0101	1010	.1011
	29.8	0010	1111	.1110
		+0011	-0011	-.0011
		0101	1100	.1011

Rule: Subtract 0011 to group which generated borrow and Add 0011 to group which do not generated borrow

Marwadi University

Gray Code

- Only one bit changes between each pair of successive code words (Unit distance code).
- Gray code is a reflected code.
- Gray codes are designed recursively using following rules:
  - 1-bit Gray code has two code words, 0 and 1.
  - The first 2^n code words of an (n+1)-bit Gray code equal the code words of n-bit gray code, written in order with a leading 0 appended.
  - The last 2^n code words of an (n+1)-bit Gray code equal the code words of n-bit gray code, but written in reverse order with a leading 1 appended.

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

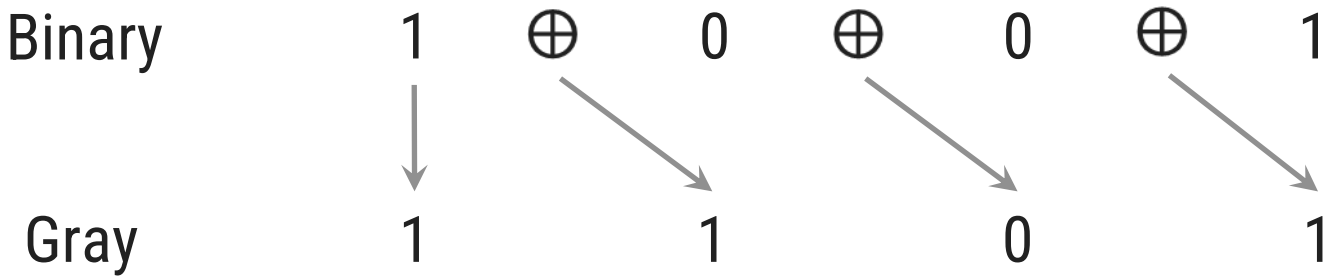
Marwadi University

Binary to Gray and Gray to Binary Conversion

- Conversion of n-bit Binary number (B) to Gray Code (G) is as follows:

$G_n = B_n$	$G_{n-1} = B_n \oplus B_{n-1}$	$G_{n-2} = B_{n-1} \oplus B_{n-2} \ldots$	$G_1 = B_2 \oplus B_1$

- Example: Convert $(1001)_2$ to Gray Code.

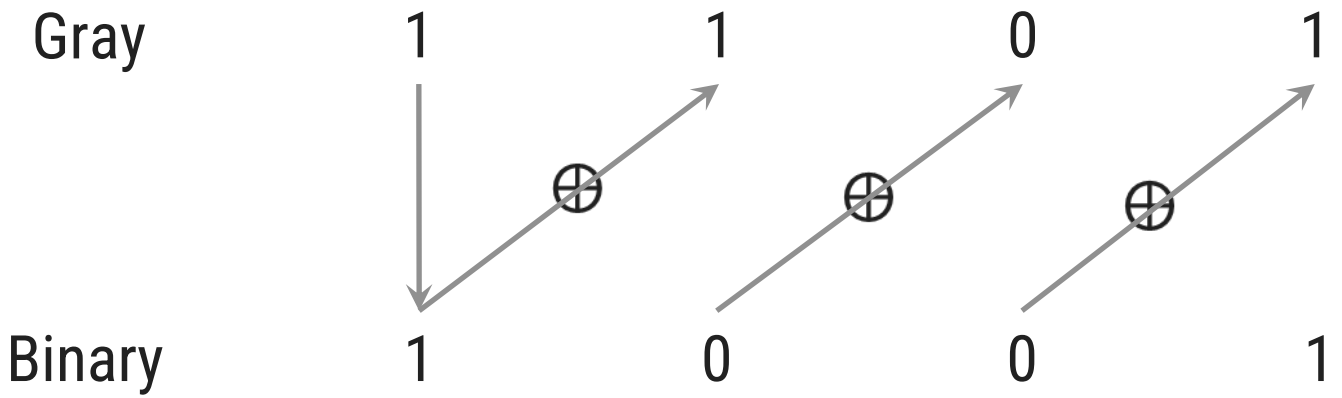


Gray to Binary Conversion

- Conversion of n-bit Gray Code (G) to Binary Number (B) is as follows:

$B_n = G_n$	$B_{n-1} = B_n \oplus G_{n-1}$	$B_{n-2} = B_{n-1} \oplus G_{n-2} \ldots$	$B_1 = B_2 \oplus G_1$

- Example: Convert Gray code 1101 to Binary.



Error-Detecting Codes

- Noise can alter or distort the data in transmission.
- The 1s may get changed to 0s and 0s to 1s.
- Because digital systems must be accurate to the digit, errors can pose a serious problem.
- Single bit error should be detect & correct by different schemes.
- Parity, Check Sums and Block Parity are few examples of error detecting code.

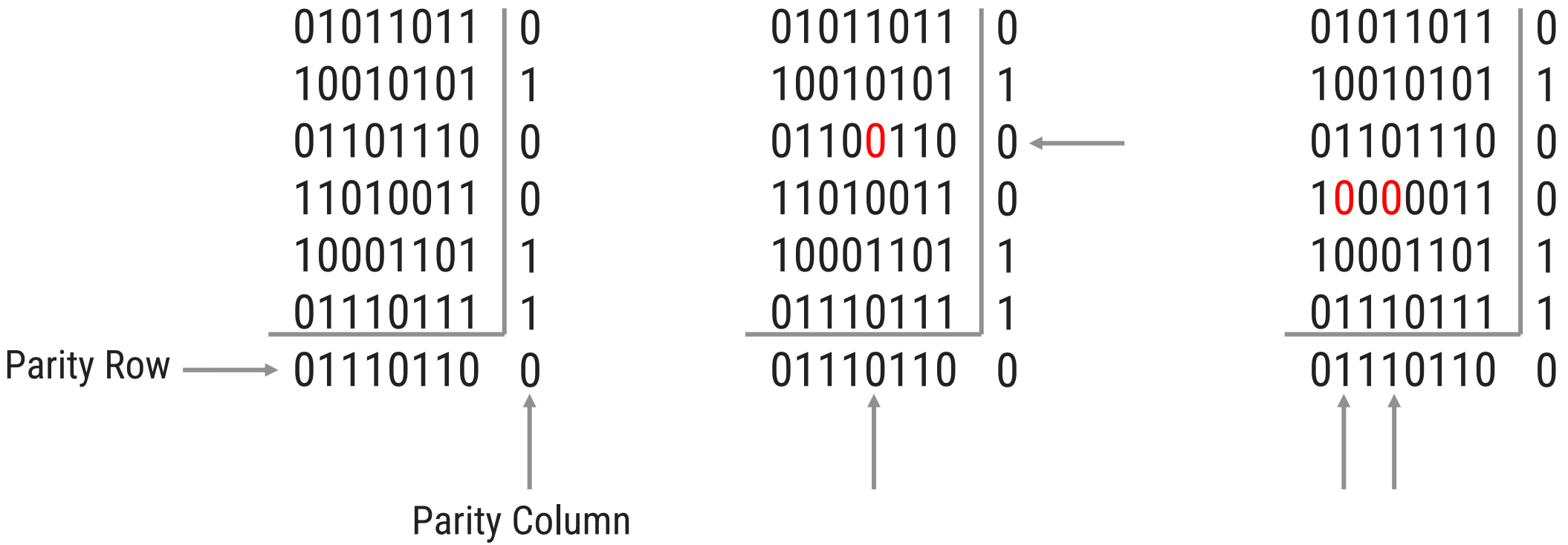
Parity

- Parity bit is the simplest technique.
- There are two types of parity – Odd parity and Even parity.
- For odd parity, the parity is set to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an odd number.
- For even parity, the parity is set to a 0 or a 1 at the transmitter such that the total number of 1 bits in the word including the parity bit is an even number.
- For example, 0110 binary number has “1” as Odd parity and “0” as Even parity.
- Detect a single-bit error but can not detect two or more errors within the same word.
- In any practical system, there is always a finite probability of the occurrence of single error.
- E.g. In an even-parity scheme, code 10111001 is erroneous because number of 1s is odd(5), while code 11110110 is error free because number of 1s is even(6).

Check Sums

- Simple parity can not detect two errors within the same word.
- Added to the sum of the previously transmitted words
- At the transmission, the check sum up to that time is sent to the receiver.
- The receiver can check its sum with the transmitted sum.
- If the two sums are the same, then no errors were detected at the receiver end.
- If there is an error, the receiving location can ask for retransmission of the entire data.
- This type of transmission is used in teleprocessing system.

Block Parity



Error Correcting Code

- 7-bit Hamming Code is widely used error correcting code, containing 4 bits of data and 3 bits of even parity.
- Pattern: $P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7$
- Group - 1: $P_1 D_3 D_5 D_7$
- Group - 2: $P_2 D_3 D_6 D_7$
- Group - 3: $P_4 D_5 D_6 D_7$
- Example: Data = 1101

  $P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7 = P_1\ P_2\ 1\ P_4\ 1\ 0\ 1$

  $P_1\ D_3\ D_5\ D_7 = 1\ 1\ 1$

  $P_2\ D_3\ D_6\ D_7 = 1\ 0\ 1$

  $P_4\ D_5\ D_6\ D_7 = 1\ 0\ 1$
- 7-bit Hamming Code is 1 0 1 0 1 0 1

- How to detect error?
- Example: Received data = 1001001

  $P_1\ P_2\ D_3\ P_4\ D_5\ D_6\ D_7 = 1\ 0\ 0\ 1\ 0\ 0\ 1$

  $P_1\ D_3\ D_5\ D_7 = 1\ 0\ 0\ 1$ (No Error)

  $P_2\ D_3\ D_6\ D_7 = 0\ 0\ 0\ 1$ (Error)

  $P_4\ D_5\ D_6\ D_7 = 1\ 0\ 0\ 1$ (No Error)
- The error word is $0\ 1\ 0 = 2_{10}$.
- Complement the 2nd bit (from left).
- Correct code is 1 1 0 1 0 0 1

Marwadi University

UNIT - 3

Boolean Algebra

Section - 3

Marwadi University

Boolean Algebra Laws

- AND laws
  1. $A \cdot 0 = 0\ (Null\ Law)$
  2. $A \cdot 1 = A\ (Identity\ Law)$
  3. $A \cdot A = A$
  4. $A \cdot \bar{A} = 0$

- OR laws
  1. $A + 0 = A\ (Null\ Law)$
  2. $A + 1 = 1\ (Identity\ Law)$
  3. $A + A = A$
  4. $A + \bar{A} = 1$

- Commutative laws
  1. $A + B = B + A$
  2. $A \cdot B = B \cdot A$

- Associative laws
  1. $(A + B) + C = A + (B + C)$
  2. $(A \cdot B)C = A(B \cdot C)$

Marwadi University

Boolean Algebra Laws

- Distributive laws

1. $A(B + C) = AB + AC$
2. $A + BC = (A + B)(A + C)$

- Redundant Literal Rule

1. $A + \bar{A}B = A + B$
2. $A(\bar{A} + B) = AB$

- Idempotent laws

1. $A \cdot A = A$
2. $A + A = A$

- Absorption laws

1. $A + AB = A$
2. $A(A + B) = A$

- De Morgan's Theorem

1. $\overline{A + B} = \bar{A}\bar{B}$
2. $\overline{AB} = \bar{A} + \bar{B}$

Break the line change the sign

Proof of $\overline{A+B+C} = \bar{A}\,\bar{B}\,\bar{C}$

L.H.S. R.H.S.

A	B	C	A+B+C	$\overline{A+B+C}$	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A}\,\bar{B}\,\bar{C}$
0	0	0	0	1	1	1	1	1
0	0	1	1	0	1	1	0	0
0	1	0	1	0	1	0	1	0
0	1	1	1	0	1	0	0	0
1	0	0	1	0	0	1	1	0
1	0	1	1	0	0	1	0	0
1	1	0	1	0	0	0	1	0
1	1	1	1	0	0	0	0	0

From truth table, it is clearly visible that L.H.S. = R.H.S. Hence, **the complement of a sum of variables is equal to the product of their individual complements.**

Proof of $\overline{ABC} = \bar{A} + \bar{B} + \bar{C}$

L.H.S. R.H.S.

A	B	C	A B C	$\overline{A\,B\,C}$	$\bar{A}$	$\bar{B}$	$\bar{C}$	$\bar{A} + \bar{B} + \bar{C}$
0	0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0	1
0	1	0	0	1	1	0	1	1
0	1	1	0	1	1	0	0	1
1	0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0	1
1	1	0	0	1	0	0	1	1
1	1	1	1	0	0	0	0	0

From truth table, it is clearly visible that L.H.S. = R.H.S. Hence, **the complement of a product of variables is equal to the sum of their individual complements.**

Reducing Boolean Expression (Example – 1)

- Reduce the expression $f = A + B[AC + (B + \bar{C})D]$

$$f = A + B[AC + (B + \bar{C})D]$$

$$f = A + B[AC + BD + \bar{C}D] \quad \text{(Distributive law)}$$

$$f = A + BAC + BBD + B\bar{C}D \quad \text{(Distributive law)}$$

$$f = A + ABC + BD + B\bar{C}D \quad \text{(A.A = A)}$$

$$f = A(1 + BC) + BD(1 + \bar{C})$$

$$f = A + BD \quad \text{(1 + A = 1)}$$

Reducing Boolean Expression (Example – 2)

- Reduce the expression $f = A[B + \bar{C}(\overline{AB + A\bar{C}})]$

$$f = A[B + \bar{C}(\overline{AB + A\bar{C}})]$$
$$f = A[B + \bar{C}(\overline{AB}\,\overline{A\bar{C}})] \quad \text{(De-Morgan's law)}$$
$$f = A[B + \bar{C}(\bar{A} + \bar{B})(\bar{A} + C)] \quad \text{(De-Morgan's law)}$$
$$f = A[B + \bar{C}(\bar{A}\bar{A} + \bar{A}C + \bar{B}\bar{A} + \bar{B}C)] \quad \text{(Distributive law)}$$
$$f = A[B + \bar{C}\bar{A} + \bar{C}\bar{A}C + \bar{C}\bar{B}\bar{A} + \bar{C}\bar{B}C] \quad \text{(Distributive law)}$$
$$f = A[B + \bar{C}\bar{A} + 0 + \bar{C}\bar{B}\bar{A} + 0] \quad \text{(A.A' = 0)}$$
$$f = AB + A\bar{C}\bar{A} + A\bar{C}\bar{B}\bar{A} \quad \text{(Distributive law)}$$
$$f = AB + 0 + 0 \quad \text{(A.A' = 0)}$$
$$f = AB$$

Logic Gates

Section - 4

Marwadi University

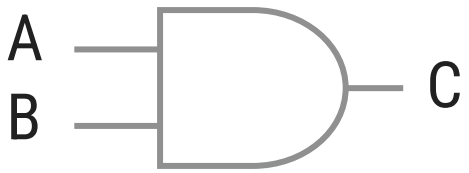
Logic Gates

- Most basic logical unit of the digital system is gate circuit.
- Types of gate circuits are as follows
  1. AND Gate
  2. OR Gate
  3. NOT Gate (Inverter)
  4. NOR Gate
  5. NAND Gate
  6. XOR Gate
  7. XNOR Gate

1. AND Gate

- AND Gate has an output which is normally at logic level “0” and only goes “HIGH” to a logic level “1” when ALL of its inputs are at logic level “1”

2-input AND Gate



Truth Table

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

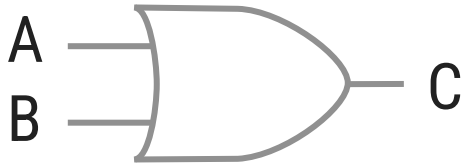
Logic Notation

$$C = A \cdot B$$

2. OR Gate

- OR Gate or Inclusive-OR gate has an output which is normally at logic level “0” and only goes “HIGH” to a logic level “1” when one or more of its inputs are at logic level “1”.

2-input OR Gate



Truth Table

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

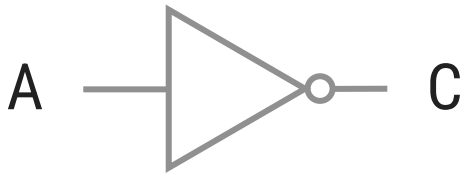
Logic Notation

$$C = A + B$$

3. NOT (Inverter) Gate

- NOT gate has an output which is always opposite to input level.

Inverter Gate



Truth Table

A	C
0	1
1	0

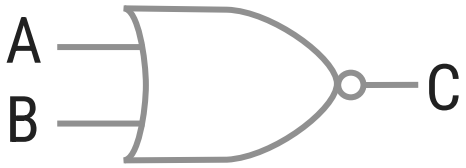
Logic Notation

$$C = \bar{A} \; or \; C = A'$$

4. NOR Gate

- NOR Gate is an OR gate followed by an inverter.
- NOR Gate has an output which is normally at logic level “1” and only goes “LOW” to a logic level “0” when one or more of its inputs are at logic level “1”.

2-input NOR Gate



Truth Table

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

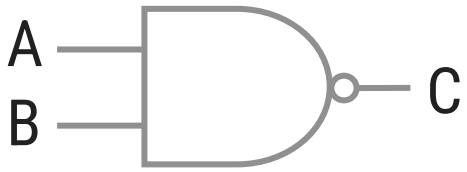
Logic Notation

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

5. NAND Gate

- NAND Gate is an AND gate followed by an inverter.
- NAND Gate has an output which is normally at logic level “1” and only goes “LOW” to a logic level “0” when ALL inputs are at logic level “1”.

2-input NAND Gate



Truth Table

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

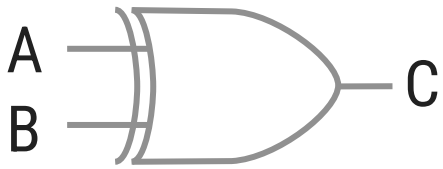
Logic Notation

$$C = (A \cdot B)'$$

6. Exclusive-OR (X-OR) Gate

- X-OR gate that has 1 state when one and only one of its two inputs assumes a logic 1 state and has 0 state when all of its input are same.
- Also known as **anti-coincidence gate** or **inequality detector**.

2-input XOR Gate



Truth Table

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

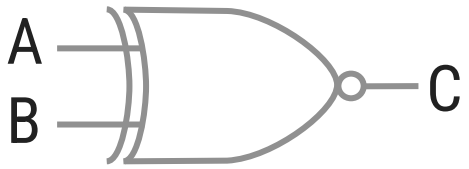
Logic Notation

$$C = A \oplus B$$

7. Exclusive-NOR (X-NOR) Gate

- X-NOR gate that has 1 state when all of its input are same and has 0 state when one of its input has 0 state and other input is 1 state.
- Also known as **coincidence gate** or **equality detector**.

2-input XNOR Gate



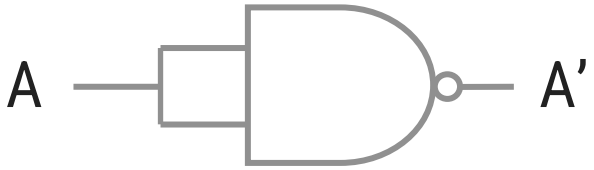
Truth Table

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

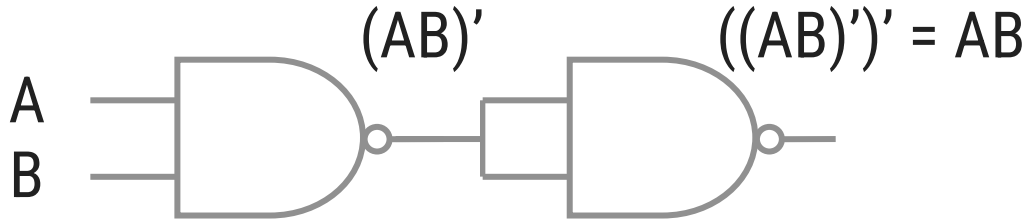
Logic Notation

$$C = A \odot B$$

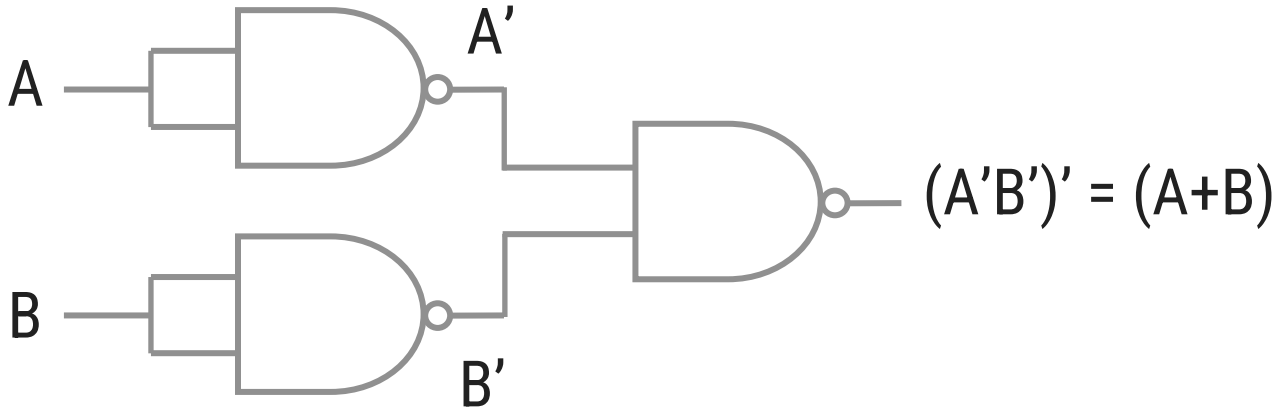
NAND as Universal Gate



NOT using NAND

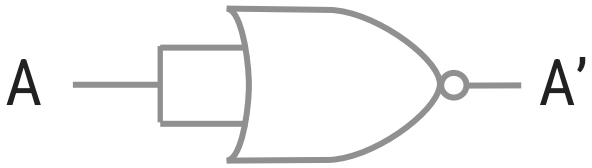


AND using NAND

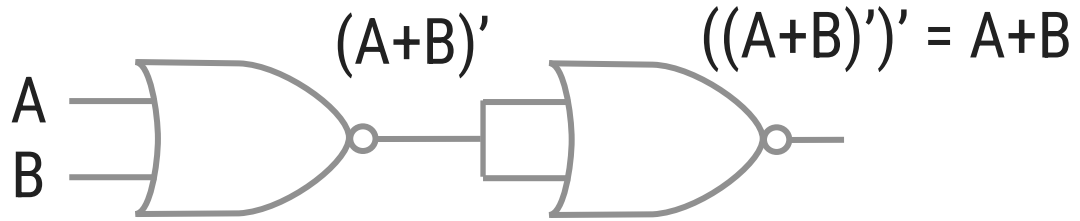


OR using NAND

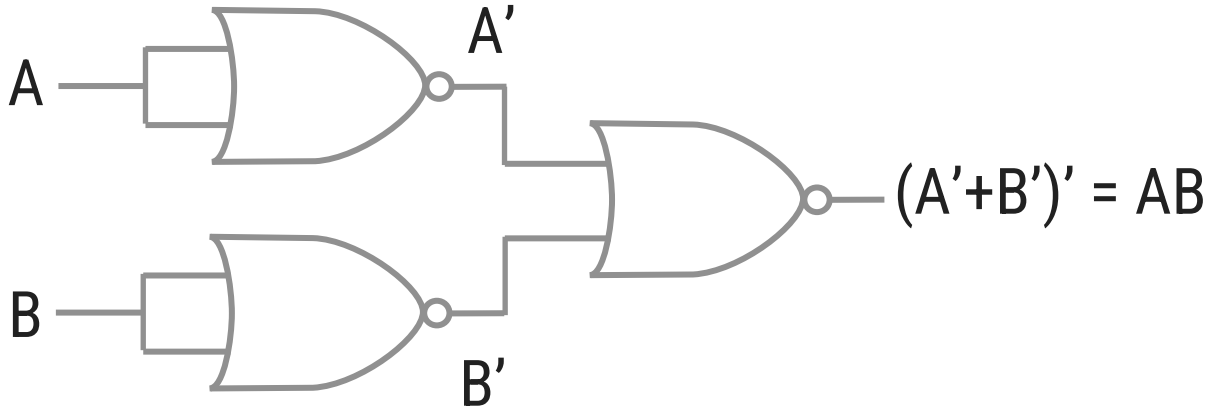
NOR as Universal Gate



NOT using NOR



OR using NOR



AND using NOR

UNIT - 4

Combinational Digital Circuits

BY .JAYDIP SIYARA

Outline

- Standard representation for logic functions
  - SOP, POS
- Simplification using K-Map
  - 2-variable, 3-variable, 4-variable, VEM
- Q-M Method (Tabulation Method)
- Adders, Subtractors
  - Half adder, Full adder, Half Subtractor, Full Subtractor
- Combinational Circuits
  - Converters, Comparators, Generators
- Multiplexer/De-multiplexer, Encoder/Decoder

Standard representation for logic functions

Section - 1

Boolean functions & representation

1. Sum-of-products (SOP) form (Disjunctive Normal Form)

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

2. Products-of-sums (POS) form (Conjunctive Normal Form)

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

Boolean functions & representation

3. Standard sum-of-products form (Minterm)
   - Contains all the variables of the function either in complemented or uncomplemented form.

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

Value of Minterm = 1

   - Variable appears in uncomplemented form if it has a value of 1 in the combination and appears in complemented form if it has a value of 0 in the combination
   - Sum of minterms whose value is equal to 1 is the standard sum of products form of the function.
   - Minterms are denoted as m_0, m_1, m_2, …,
   - For 3 variable function, $m_0=\bar{A}\bar{B}\bar{C}$, $m_1=\bar{A}\bar{B}C$, $m_2=\bar{A}B\bar{C}$, $m_3=\bar{A}BC$, $m_4=A\bar{B}\bar{C}$, $m_5=A\bar{B}C$, $m_6=AB\bar{C}$, and $m_7=ABC$

Boolean functions & representation

- Canonical SOP form is shown by the sum of minterms for which the function equals 1.

$$f(A,B,C) = m_1 + m_2 + m_3 + m_5$$

Or

$$f(A,B,C) = \sum_m (1,2,3,5)$$

4. Standard product-of-sum form (Maxterm)
   - Derived by considering the combinations of which f = 0
   - Each term is a sum of all the variables
   - Variable appears in uncomplemented form if it has a value of 0 in the combination and appears in complemented form if it has a value of 1 in the combination

Boolean functions & representation

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

Value of Maxterm = 0

- A sum term which contains each of the n variables in either complemented or uncomplemented form is called a maxterm.
- Maxterms are denoted as M_0, M_1, M_2, …,

$$f(A,B,C) = M_0M_1M_6M_7$$

Or

$$f(A,B,C) = \prod_M(0,1,6,7)$$

Simplification using K-Map

Section - 2

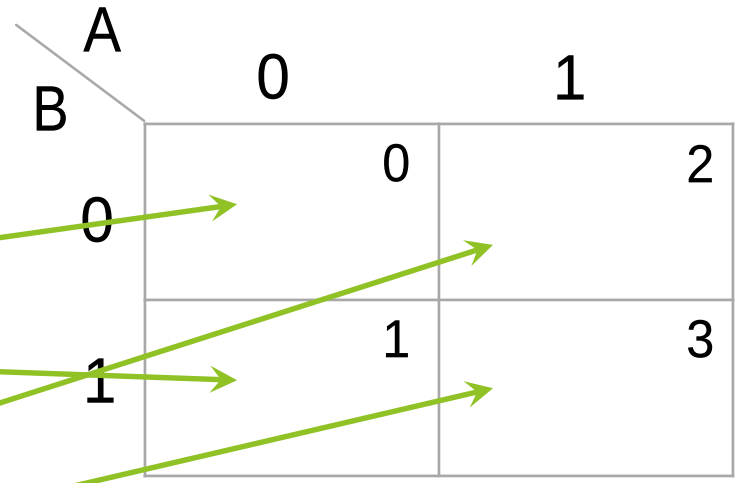
Introduction to K-Maps

- Simplification of Boolean functions leads to simpler (and usually faster) digital circuits.
- Simplifying Boolean functions using identities is time-consuming and error-prone.
- This special section presents an easy, systematic method for reducing Boolean expressions.
- A K-Map is a matrix consisting of rows and columns that represent the output values of a Boolean function.
- The output values placed in each cell are derived from the minterms of a Boolean function.
- A minterm is a product term that contains all of the function's variables exactly once, either complemented or not complemented.

2 - Variable K-Map

- The two variables A and B have four possible combinations that can be represented by the map as follows

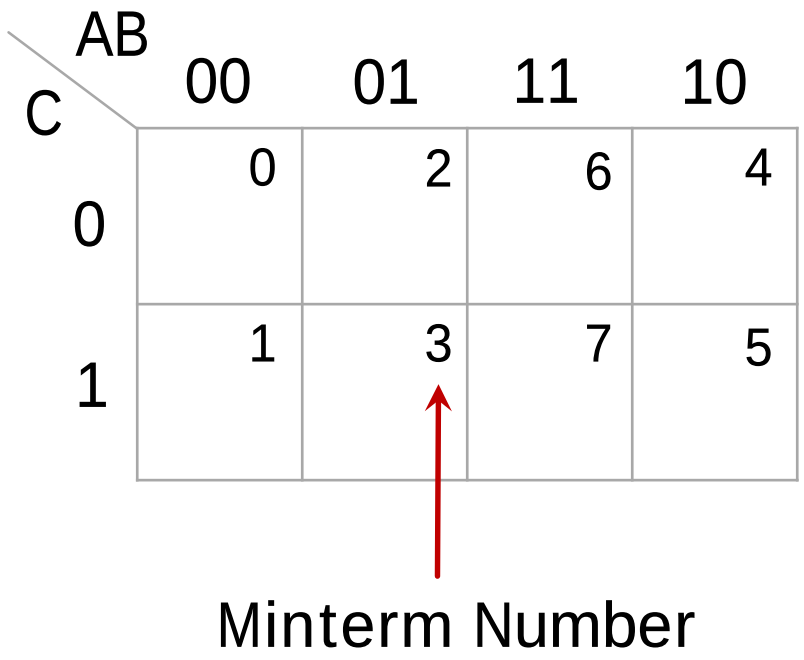
A	B	Minterm
0	0	$m_0 = A'B'$
0	1	$m_1 = A'B$
1	0	$m_2 = AB'$
1	1	$m_3 = AB$



3 - Variable K-Map

- The three variables A, B and C have eight possible combinations that can be represented by the map as follows

A	B	C	Minterm
0	0	0	m_0 = A'B'C'
0	0	1	m_1 = A'B'C
0	1	0	m_2 = A'BC'
0	1	1	m_3 = A'BC
1	0	0	m_4 = AB'C'
1	0	1	m_5 = AB'C
1	1	0	m_6 = ABC'
1	1	1	m_7 = ABC



4 - Variable K-Map

- The four variables A, B, C and D have sixteen possible combinations that can be represented by the map as follows

A	B	C	D	Minterm
0	0	0	0	m_0 = A'B'C'D'
0	0	0	1	m_1 = A'B'C'D
0	0	1	0	m_2 = A'B'CD'
0	0	1	1	m_3 = A'B'CD
0	1	0	0	m_4 = A'BC'D'
0	1	0	1	m_5 = A'BC'D
0	1	1	0	m_6 = A'BCD'
0	1	1	1	m_7 = A'BCD

A	B	C	D	Minterm
1	0	0	0	m_8 = AB'C'D'
1	0	0	1	m_9 = AB'C'D
1	0	1	0	m_{10} = AB'CD'
1	0	1	1	m_{11} = AB'CD
1	1	0	0	m_{12} = ABC'D'
1	1	0	1	m_{13} = ABC'D
1	1	1	0	m_{14} = ABCD'
1	1	1	1	m_{15} = ABCD

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

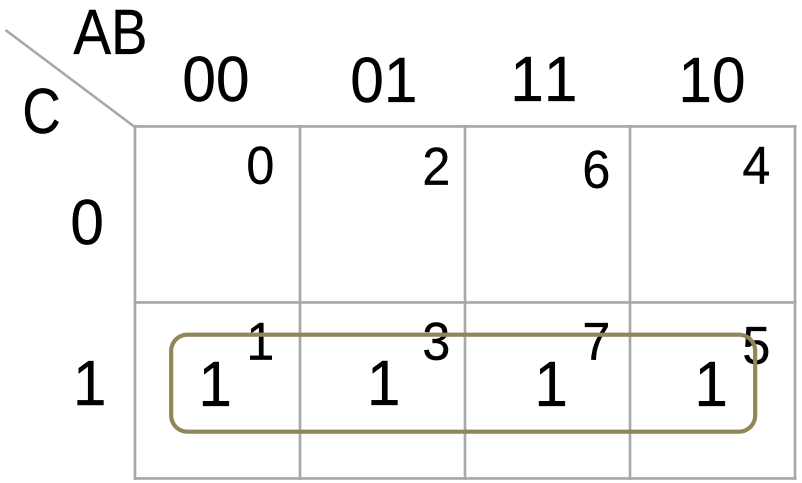
Reduce Boolean Expression

- Consider the function:

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

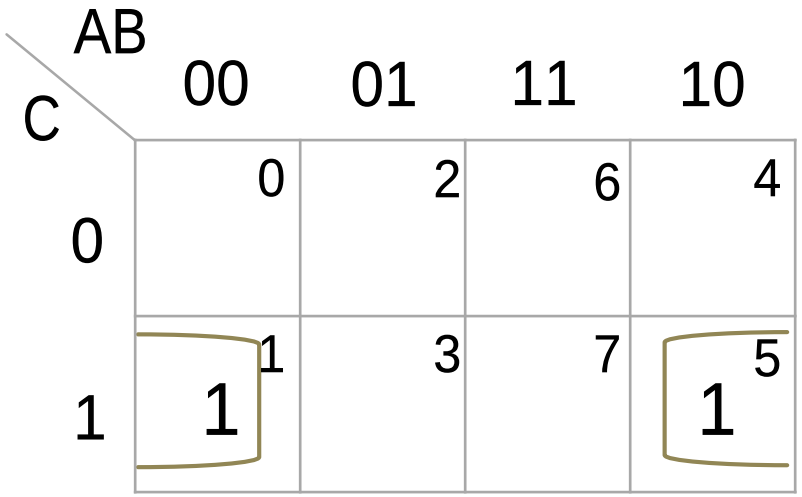
- Its K-Map is given below.
- What is the largest group of 1's that is a power of 2?
- This grouping tells us that changes in the variables *A* and *B* have no influence upon the value of the function: They are irrelevant.

$$f(A,B,C) = C$$



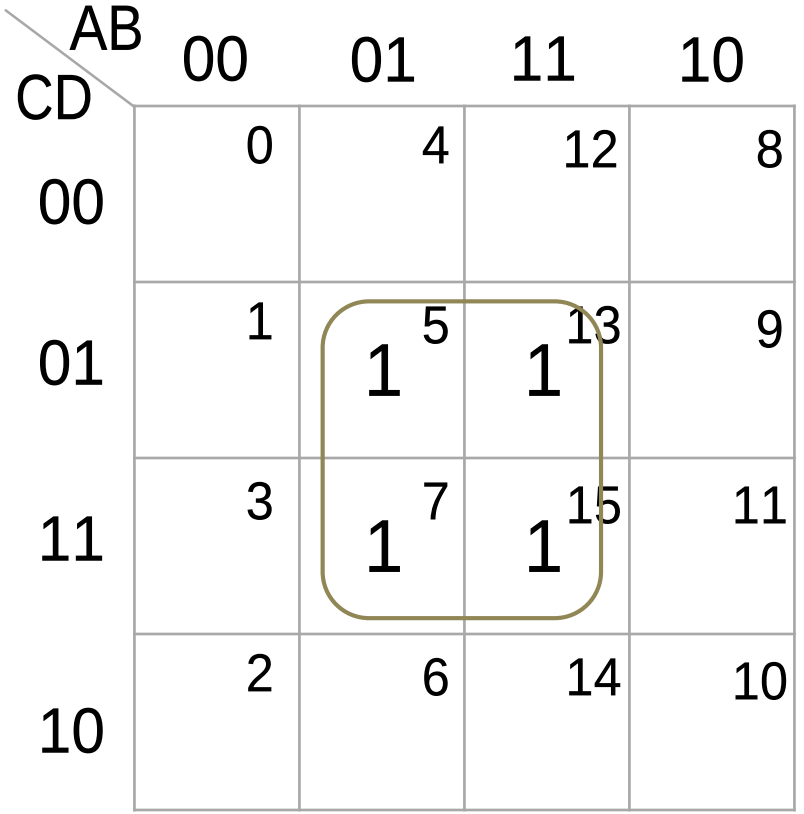
Examples (SOP)

$$f = AB'C + A'B'C$$



$$f = B'C$$

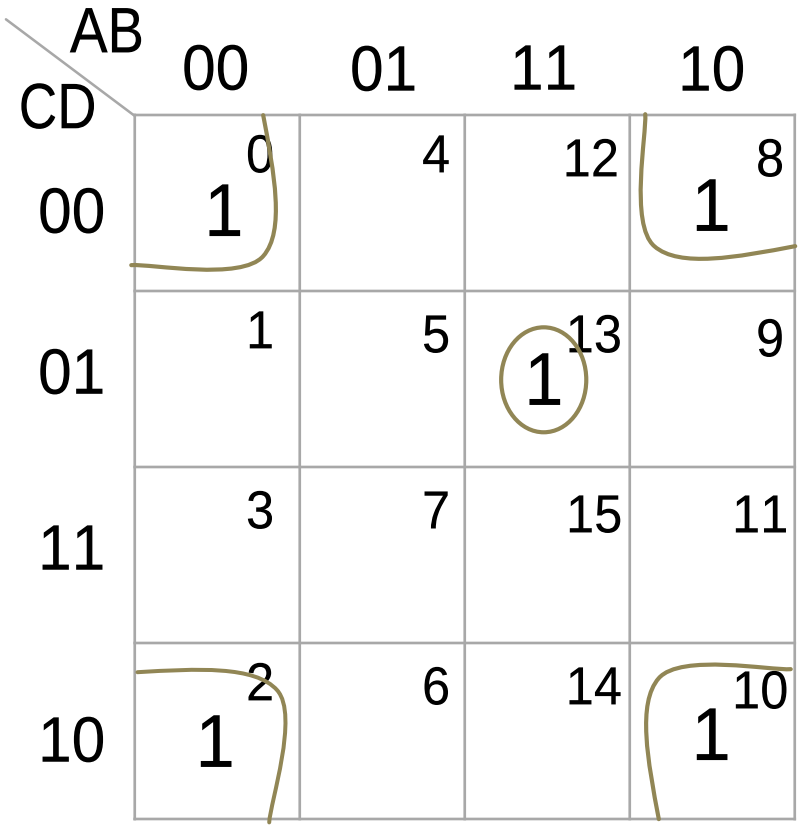
$$f = ABC'D + ABCD + A'BC'D + A'BCD$$



$$f = BD$$

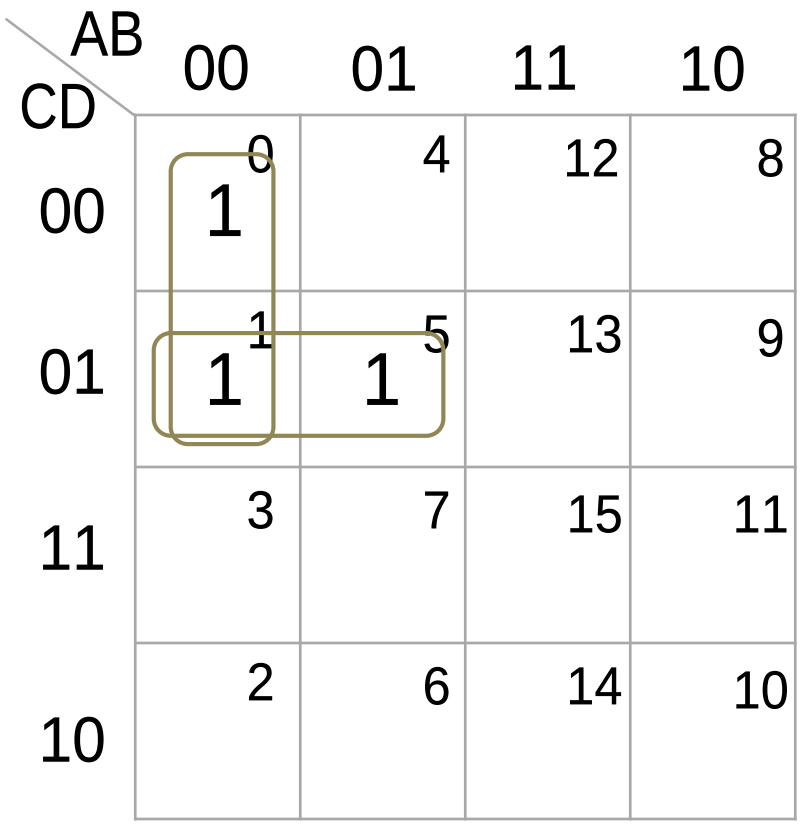
Examples (SOP)

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



$f = ABC'D + B'D'$

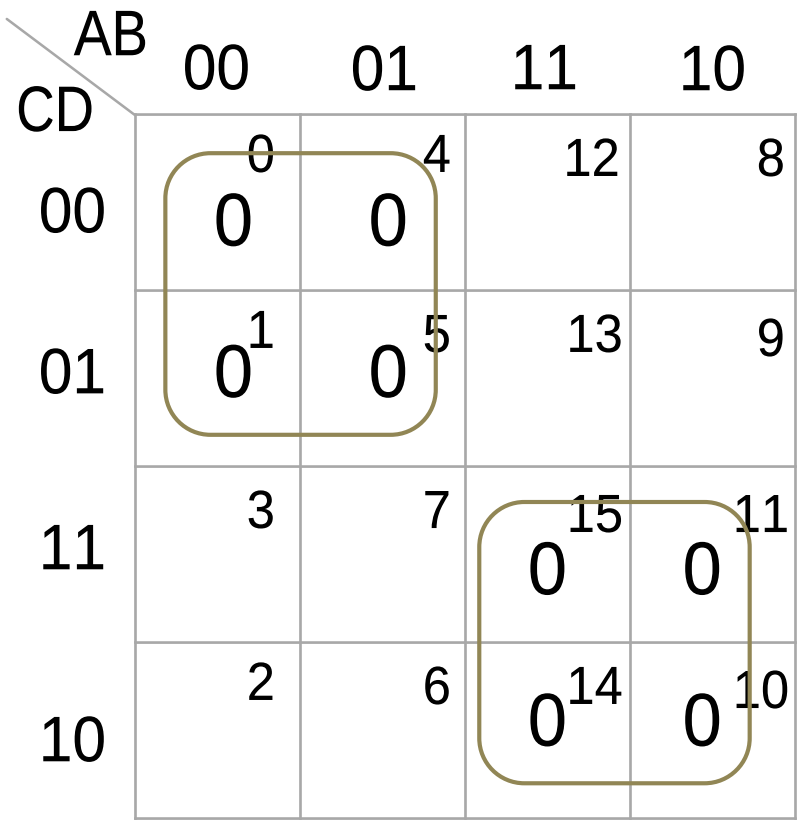
$f = A'BC'D + A'B'C'D + A'B'C'D'$



$f = A'B'C' + A'C'D$

Example (POS)

$$f(A,B,C,D) = \prod_M (0,1,4,5,10,11,14,15)$$

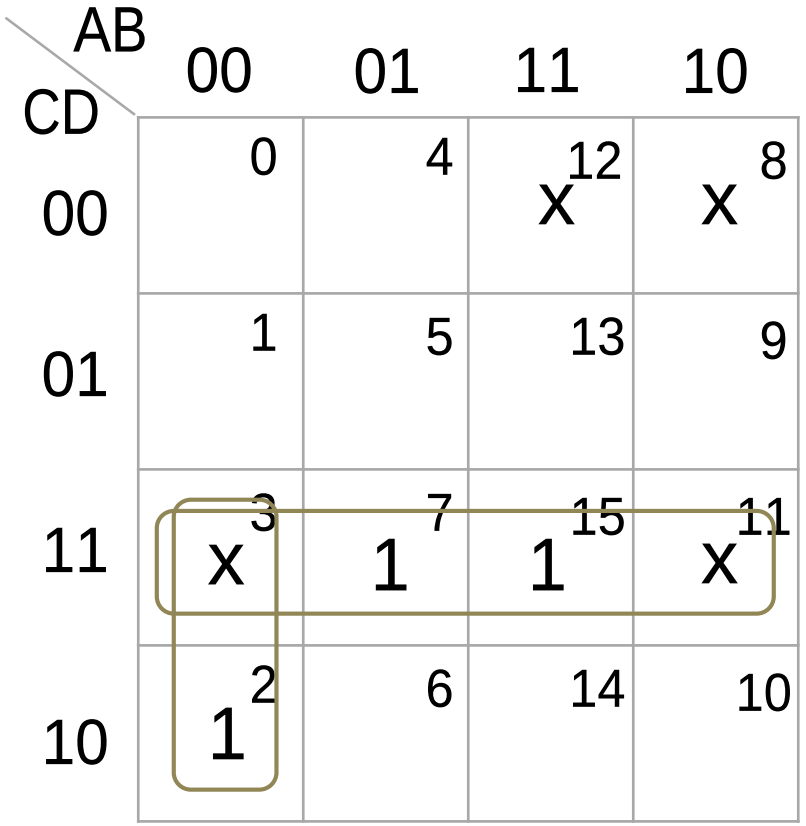


$$f = (A + C)(A' + C')$$

Don’t care conditions

- Suppose we are given a problem of implementing a circuit to generate a logical 1 when a 2, 7, or 15 appears on a four-variable input.
- A logical 0 should be generated when 0, 1, 4, 5, 6, 9, 10, 13 or 14 appears.
- The input conditions for the numbers 3, 8, 11 and 12 never occur in the system. This means we don’t care whether inputs generate logical 1 or logical 0.
- Don’t care combinations are denoted by ‘x’ in K-Map which can be used for the making groups.
- The above example can be represented as

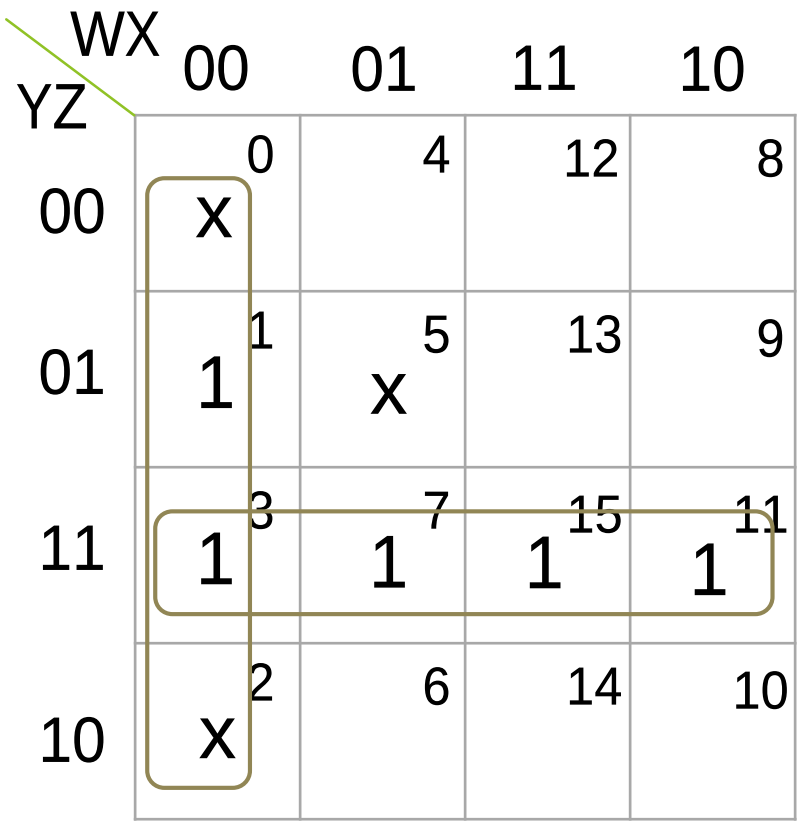
$f(A,B,C,D) = \sum_m (2,7,15) + d(3,8,11,12$



$$f = CD + A'B'C$$

Example (Don’t care)

$$F(W,X,Y,Z) = \sum_m (1,3,7,11,15) + d(0,2,5)$$



$$F = W'X' + YZ$$

Variable-Entered Map (VEM)

- Variable-entered map can be used to plot an n-variable problem on n – 1 variable map
- Possible to reduce the map dimension by two or three in some cases
- Advantage of using VEM occurs in design problems involving multiplexers
- Map-entered variable for 3 variable function
- Consider the function

$$X = A'B'C' + ABC' + AB'C' + ABC$$

- Considering value of X to be function of map location with the variable C and plotting 2 variable map.

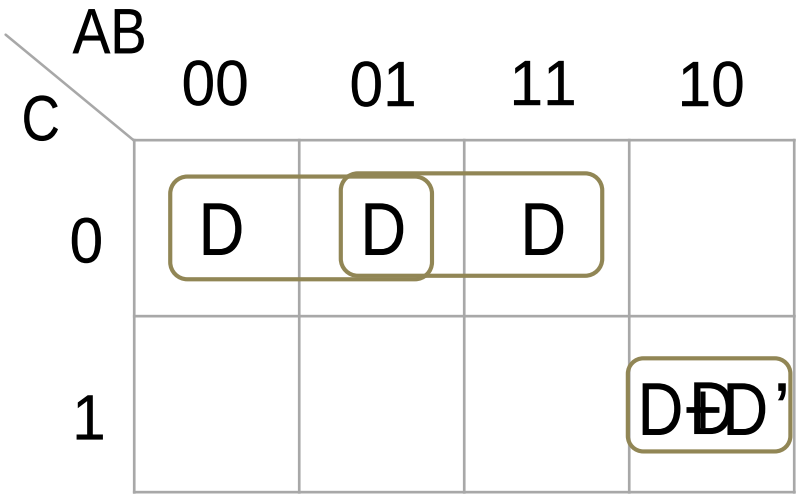
B \ A	0	1
0	X=1 if C'=1	X=1 if C'=1
1	X=0	X=1 if C'=1 or if C = 1

B \ A	0	1
0	C'	C'
1	0	C + C'

Reducing Expressions with VEM (Example)

- Choose groups of *similar terms* to cover all nonzero terms appearing in the map except “don’t care” terms.
- Rest complete process is similar to K-Map

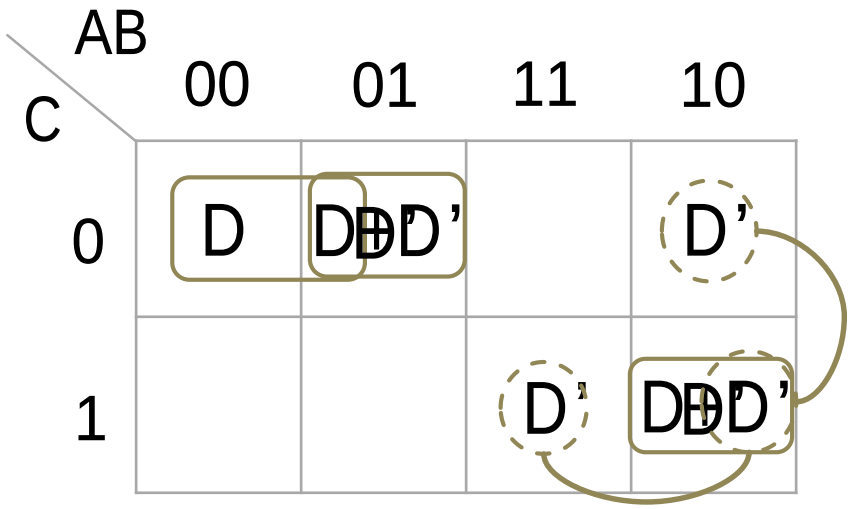
$$f = AB'CD + A'BC'D + AB'CD' + ABC'D + A'B'C'D$$



$$f = A'C'D + BC'D + AB'C$$

Reducing Expressions with VEM (Example)

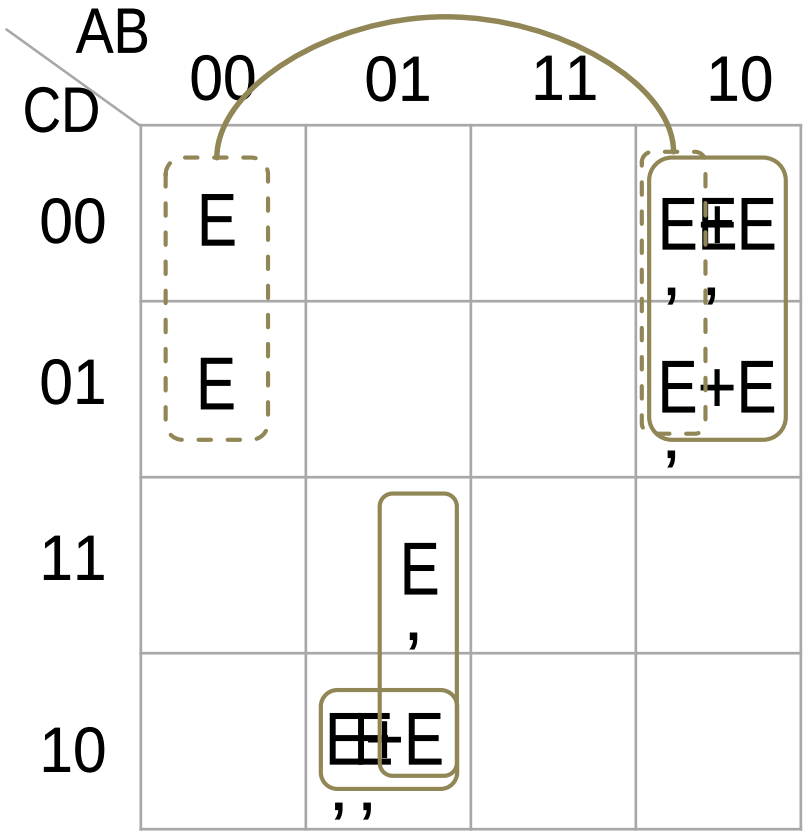
$$f = A'B'C'D + A'BC'D' + A'BC'D + AB'C'D' + AB'CD' + AB'CD + ABCD'$$



$$f = A'C'D \ + A'BC' + ACD' + AB'D' + AB'C$$

Reducing Expressions with VEM (Example)

$$f = A'B'C'D'E + A'B'C'DE + A'BCD'E' + A'BCD'E + AB'C'D'E' + AB'C'D'E + AB'C'D + A'BCDE'$$



$$f = B'C'E + AB'C' + A'BCD' + A'BCE'$$

Q-M Method (Tabulation Method)

Section - 3

Tabulation Method (Quine-McCluskey Method)

- Fundamental idea for tabulation method is combining theorem.

$$PA + PA' = P$$

- The above theorem is applied repeatedly to all adjacent pairs of terms which results in prime implicants.
- Consider the minimization of the expression

$\sum_m(0,1,4,5)$ = A'B'C' + A'B'C + AB'C' + AB'C

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

= A'B' + AB'

= B'(A + A')

= B'

Procedure for minimization using Tabulation Method

1. List all the minterms.
2. Arrange all minterms in groups of the same number of 1s in their binary representation in column 1. Start with the least number of 1s group and continue with groups of increasing number of 1s.
3. Compare each term of the lowest index group with every term in the succeeding group. Whenever possible, combine the two terms being compared by means of the combining theorem. Two terms from adjacent groups are combinable, if their binary representations differ by just a single digit in the same position; the combined terms consist of the original fixed representation with the differing one replaced by a dash (-). Place a check mark (√) next to every term, which has been combined with at least one term and write the combined terms in column 2. Repeat this by comparing each term in a group of index i with every term in the group of index i + 1, until all possible applications of the combining theorem have been exhausted.

Procedure for minimization using Tabulation Method

4. Compare the terms generated in step 3 in the same fashion; combine two terms which differ by only a single 1 and whose dashes are in the same position to generate a new term. Two terms with dashes in different positions cannot be combined. Write the new terms in column 3 and put a check mark next to each term which has been combined in column 2. Continue the process with terms in columns 3, 4 etc. until no further combinations are possible. The remaining *unchecked terms* constitute the *set of prime implicants* of the expression.
5. List all the prime implicants and draw the *prime implicant chart*. (The don't cares if any should not appear in the prime implicant chart).
6. Obtain the *essential prime implicants* and write the minimal expression.

Tabulation Method (Example)

$$f = \sum_m (0,1,6,7,8,9,13,14,15)$$

Step – 1: List all minterms

Minterms	Binary Designation
0	0 0 0 0
1	0 0 0 1
6	0 1 1 0
7	0 1 1 1
8	1 0 0 0
9	1 0 0 1
13	1 1 0 1
14	1 1 1 0
15	1 1 1 1

Step – 2: Arrange all minterms in groups of same numbe

Column 1		
Index	**Minterms**	**Binary Designation**
Index 0	0	0 0 0 0
Index 1	1 8	0 0 0 1 1 0 0 0
Index 2	6 9	0 1 1 0 1 0 0 1
Index 3	7 13 14	0 1 1 1 1 1 0 1 1 1 1 0
Index 4	15	1 1 1 1

Tabulation Method (Example) Cont..

Step – 3: Compare each term of the lowest index group with every term in the succeeding group till no change

Column 1		
Index	**Minterms**	**Binary Designation**
Index 0	0	0 0 0 0 √
Index 1	1 8	0 0 0 1 √ 1 0 0 0 √
Index 2	6 9	0 1 1 0 √ 1 0 0 1 √
Index 3	7 13 14	0 1 1 1 √ 1 1 0 1 √ 1 1 1 0 √
Index 4	15	1 1 1 1 √

Column 2	
Pairs	**A B C D**

Tabulation Method (Example) Cont..

Step – 4: Compare the terms generated in step 3 in the same fashion until no further combinations are possible

Pairs	A B C D	
0, 1	0 0 0 _	√
0, 8	_ 0 0 0	√
1, 9	_ 0 0 1	√
8, 9	1 0 0 _	√
6, 7	0 1 1 _	√
6, 14	_ 1 1 0	√
9,13	1 _ 0 1	S
7, 15	_ 1 1 1	√
13, 15	1 1 _ 1	R
14, 15	1 1 1 _	√

Quads	A B C D
(	Q
( )	P

Tabulation Method (Example) Cont..

Step – 5: List all prime implicants and draw prime implicants chart

Prime Implicants : P(BC), Q(B'C'), R(ABD), S(AC'D)

PIs									
			^	^				^	^
	x	x			x	x			
									x
S (9,13)						x	x		

Step – 6: Obtain essential prime implicants and minimal expression

Essential Prime Implicants : , Q(B'C')

P(BC)

Minimal expression : P + Q + R = BC + B'C' + ABD

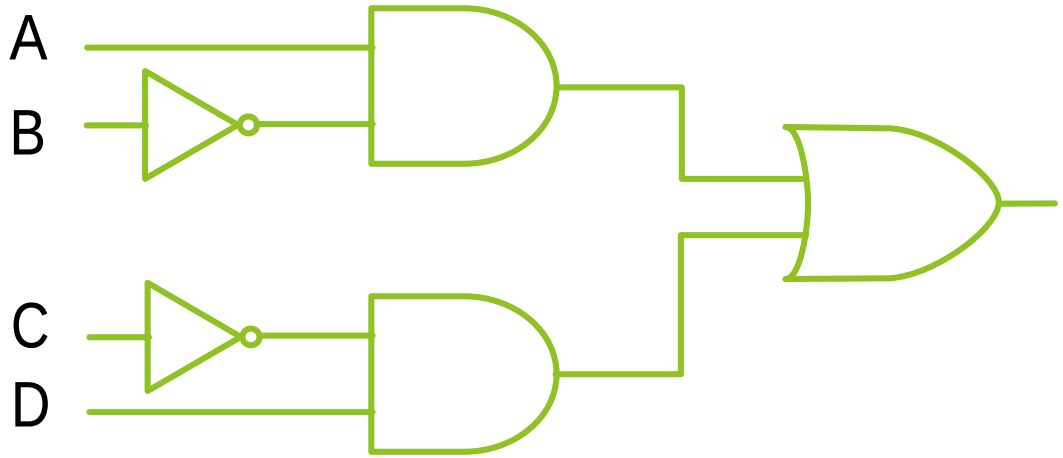
OR

P + Q + S = BC + B'C' + AC'D

As minterm 13 is covered by R and S

Realizing Logic Function with Gates

- Implement AB’ + C’D using AND, OR & Invert Gates

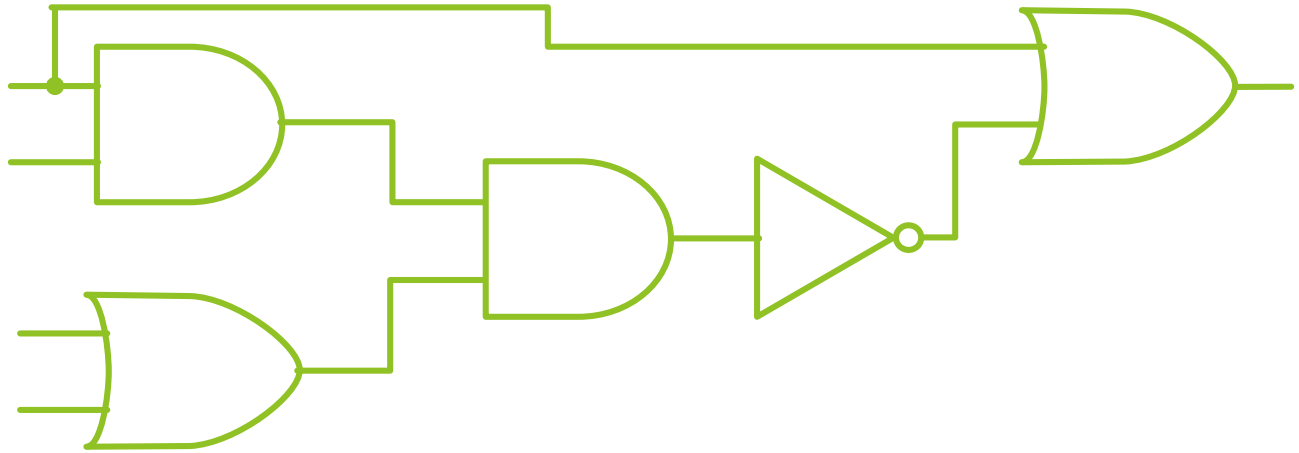


Converting AND/OR/Invert Logic to NAND/NOR Logic

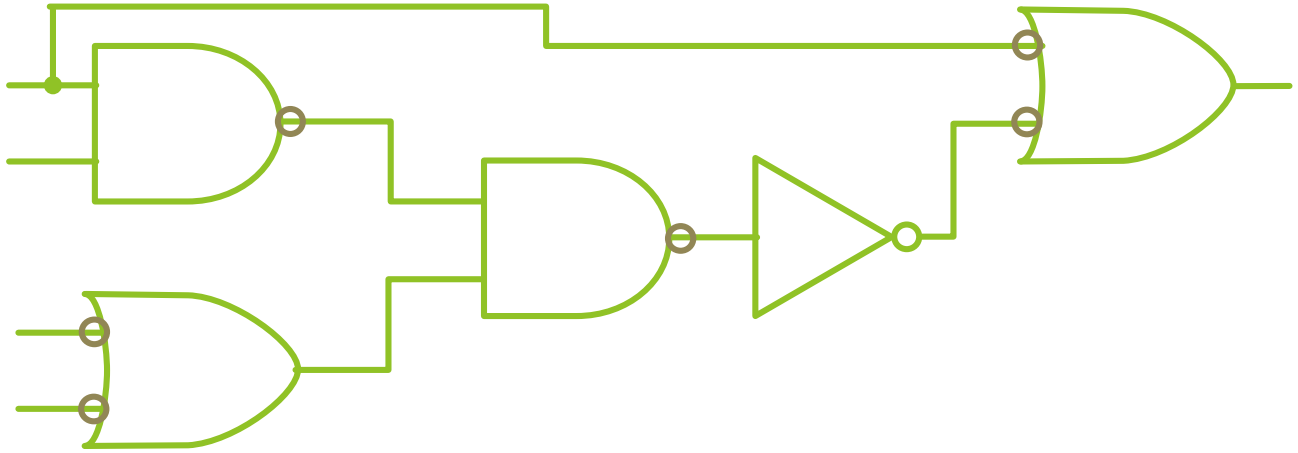
1. Draw the circuit in AOI logic.
2. If NAND hardware is chosen, add a circle at the output of each AND gate and at the inputs to all the OR gates.
3. If NOR hardware is chosen, add a circle at the output of each OR gate and at the inputs to all the AND gates.
4. Add or subtract an inverter on each line that received a circle in steps 2 or 3 so that the polarity of signals on those lines remains unchanged from that of the original diagram.
5. Replace bubbled OR by NAND and bubbled AND by NOR.
6. Eliminate double inversions.

Converting AND/OR/Invert Logic to NAND/NOR Logic (Example)

- Implement the following AOI logic using NAND logic

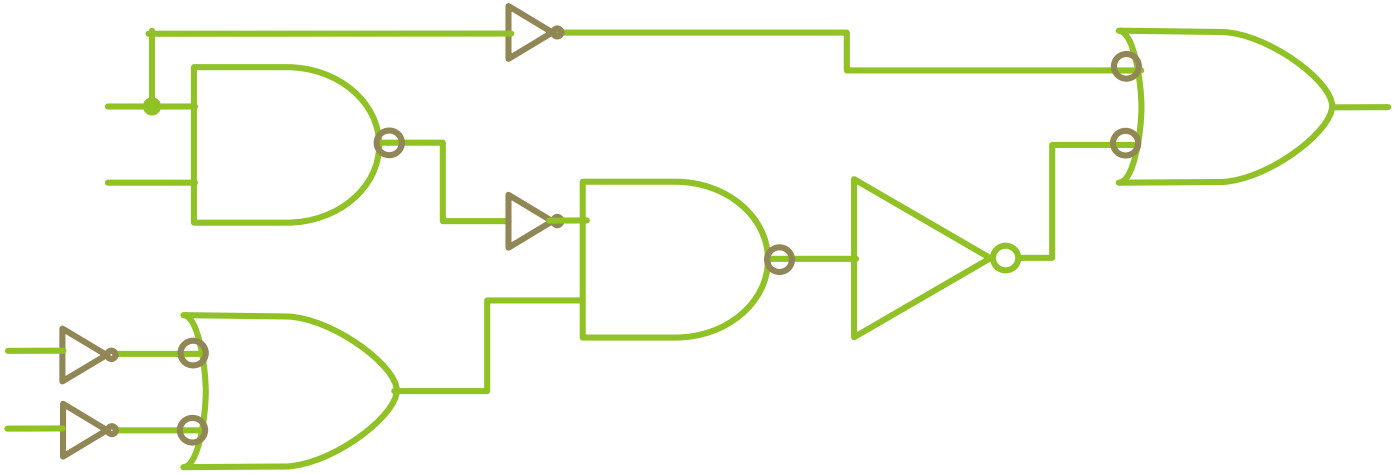


Put a circle at the output of each AND gate and at the inputs to all OR gates



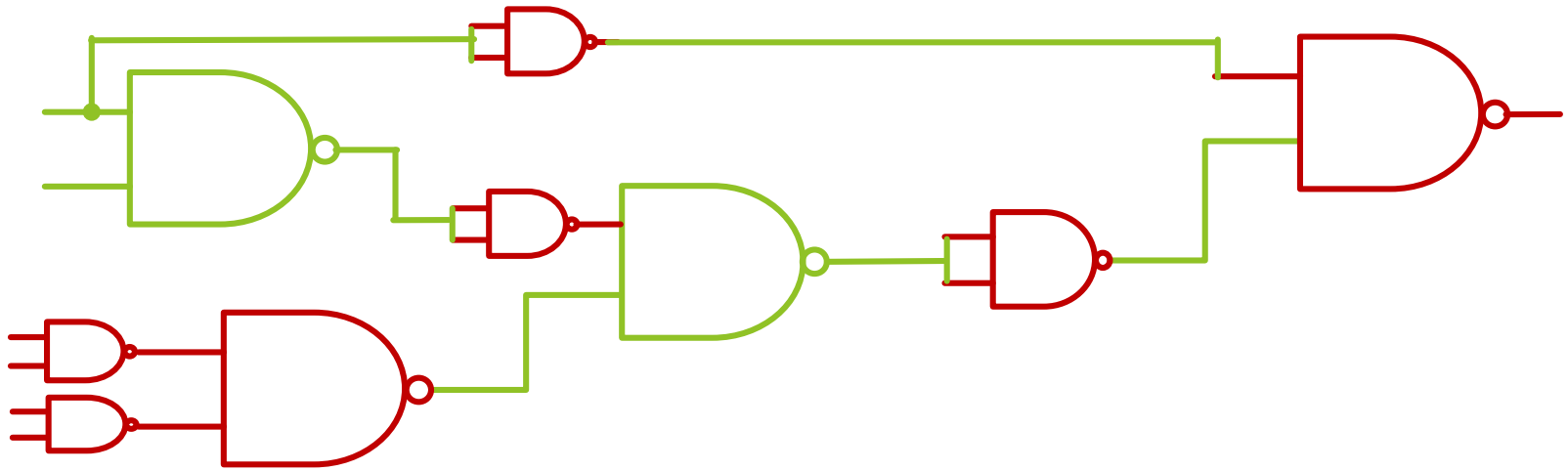
Converting AND/OR/Invert Logic to NAND/NOR Logic (Example)

Add an inverter to each of the lines that received only one circle at input so that polarity remains unchanged.



Converting AND/OR/Invert Logic to NAND/NOR Logic (Example)

Replace bubbled OR gates and NOT gates by NAND gates.



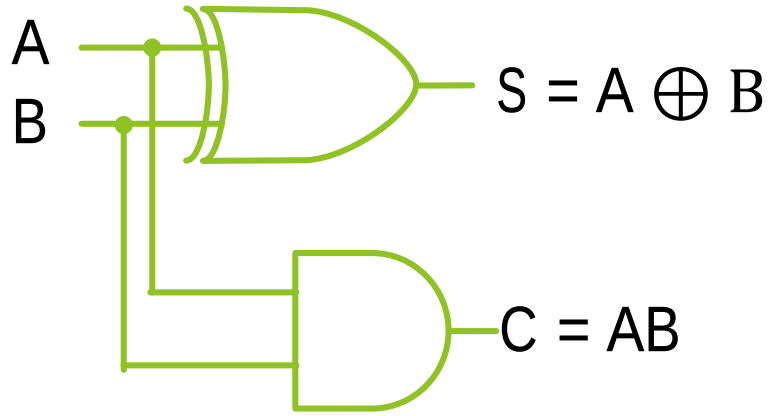
Adders, Subtractors

Section - 4

Half Adder

- A combinational circuit which adds two one-bit binary numbers is called a half-adder.

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



- The sum column resembles like an output of the XOR gate.
- The carry column resembles like an output of the AND gate.
- Limitations:
  - In multi-digit addition we have to add two bits along with the carry of previous digit addition. Such addition requires addition of 3 bits. This is not possible in half-adders.

Full Adder

- The full-adder adds the bits A and B and the carry from the previous column called carry-in C_{in} and outputs the sum bit S and the carry bit called carry-out C_{out}.

Inputs			Outputs	
A	**B**	**C_{in}**	**S**	**C_{out}**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

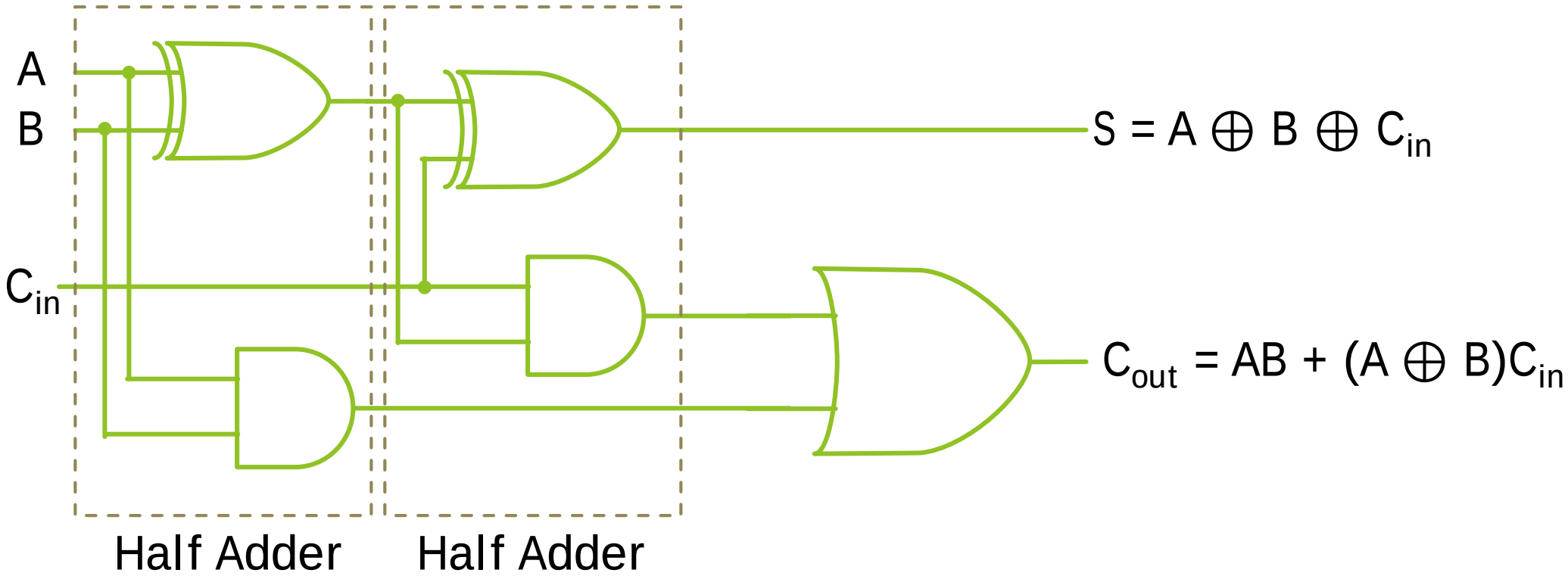
$$S = A'B'C_{in} + A'BC_{in}' + AB'C_{in}' + ABC_{in}$$
$$= (AB' + A'B)C_{in}' + (AB + A'B')C_{in}$$
$$= (A \oplus B)C_{in}' + (A \oplus B)'C_{in}$$
$$= A \oplus B \oplus C_{in}$$

$$C_{out} = A'BC_{in} + AB'C_{in} + ABC_{in}' + ABC_{in}$$
$$= AB + (A \oplus B)C_{in}$$

Full Adder – Logical Diagram

$S = A \oplus B \oplus C_{in}$

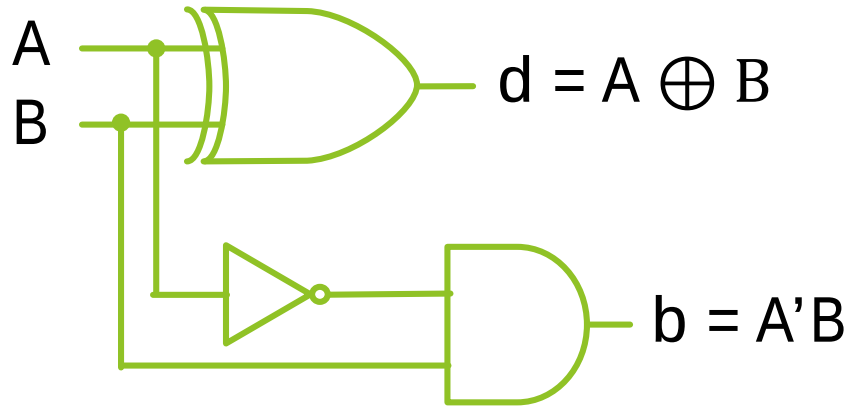
$C_{out} = AB + (A \oplus B)C_{in}$



Half Subtractor

- Subtracts one bit from the other and produces the difference.
- Other output is to specify if 1 is borrowed.

Inputs		Outputs	
A	**B**	**d**	**b**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0



- Limitation:
  - Half subtractor can be used only for LSB subtraction.

Full Subtractor

- Full subtractor is a combinational circuit with 3 inputs (A, B, b_i)
- Subtraction = A – B – b_i

Inputs			Outputs	
A	**B**	**b_i**	**d**	**b**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

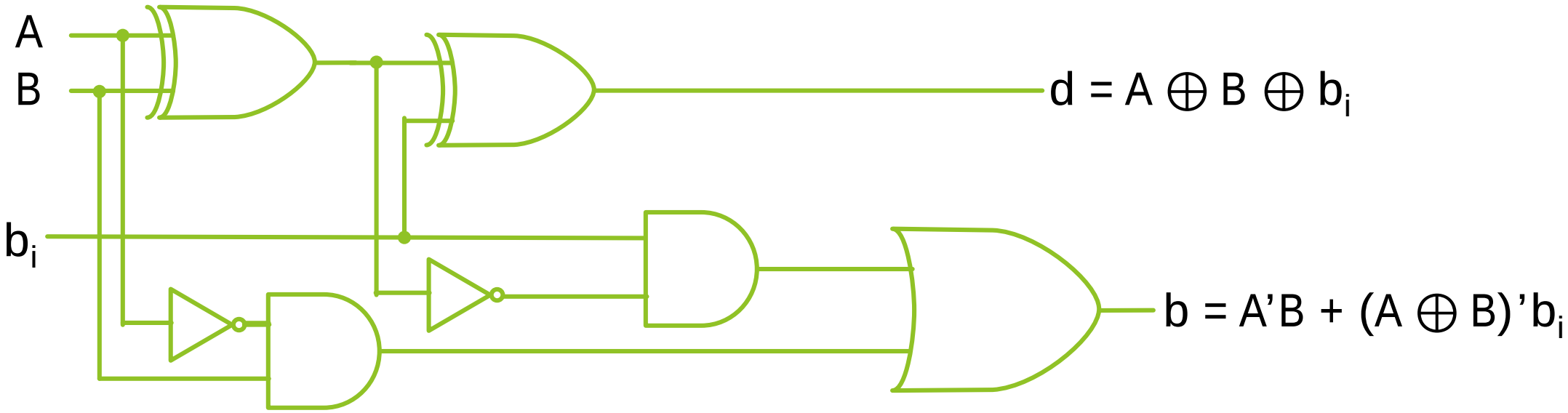
$$d = A'B'b_i + A'Bb_i' + AB'b_i' + ABb_i$$
$$= (AB' + A'B)b_i' + (AB + A'B')b_i$$
$$= (A \oplus B)b_i' + (A \oplus B)'b_i$$
$$= A \oplus B \oplus b_i$$

$$b = A'B'b_i + A'Bb_i' + A'Bb_i + ABb_i$$
$$= A'B(b_i + b_i') + (AB + A'B')b_i$$
$$= A'B + (A \oplus B)'b_i$$

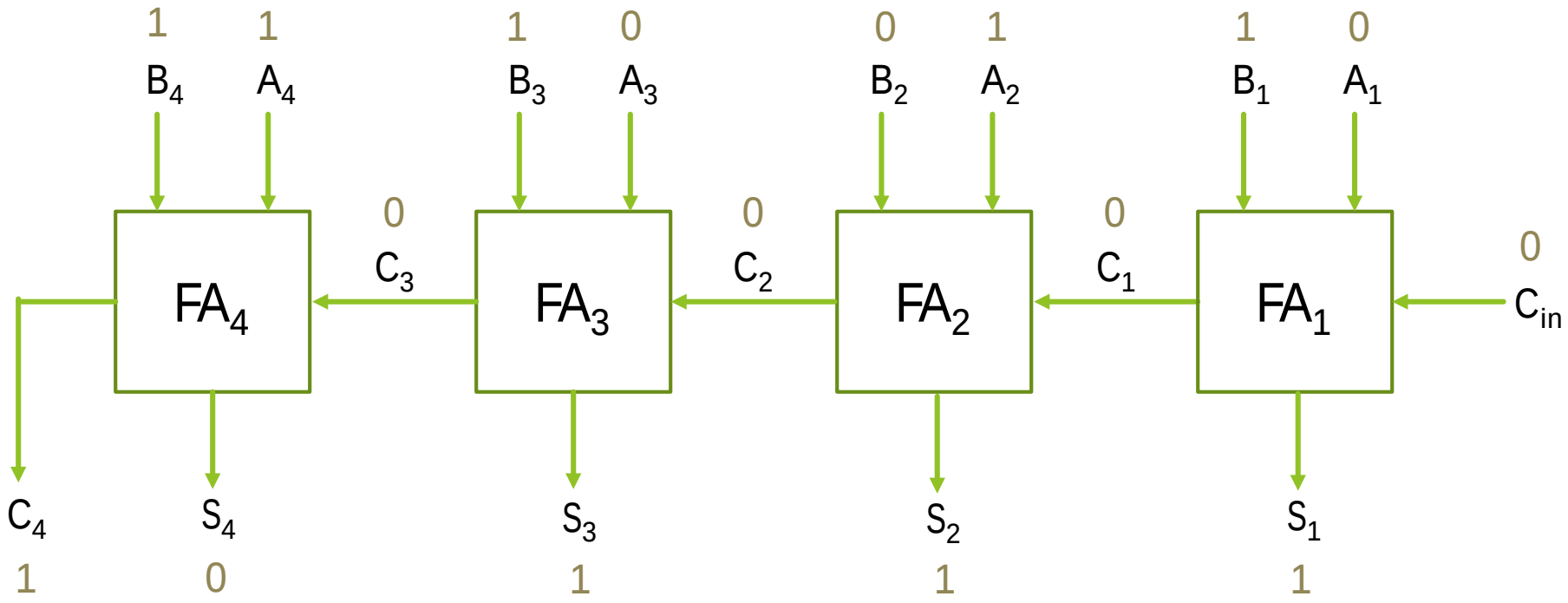
Full Subtractor – Logical Diagram

$d = A \oplus B \oplus b_i$

$b = A'B + (A \oplus B)'b_i$



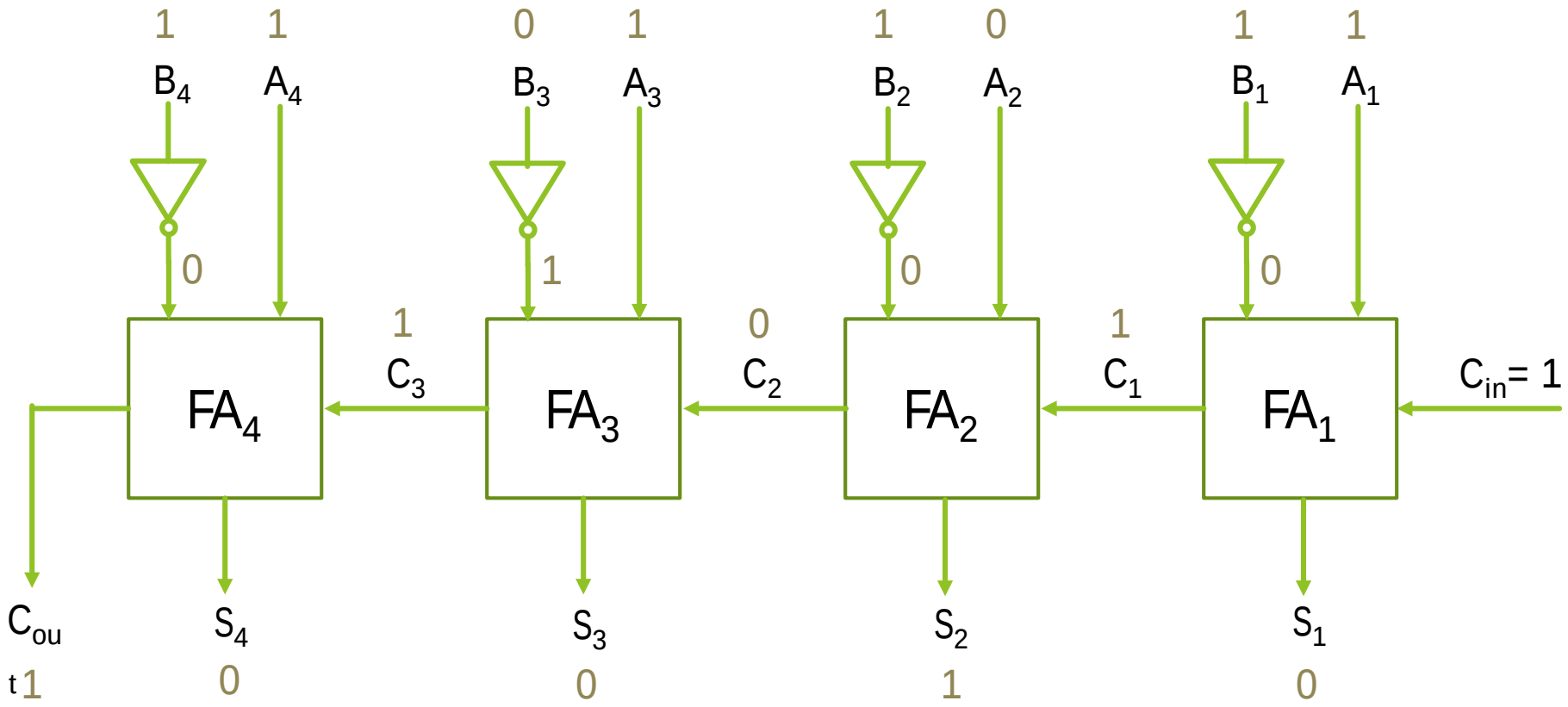
Binary Parallel Adder



Example: A = 1010, B = 1101

$A_4\ A_3\ A_2\ A_1 = 1010$
$B_4\ B_3\ B_2\ B_1 = 1101$

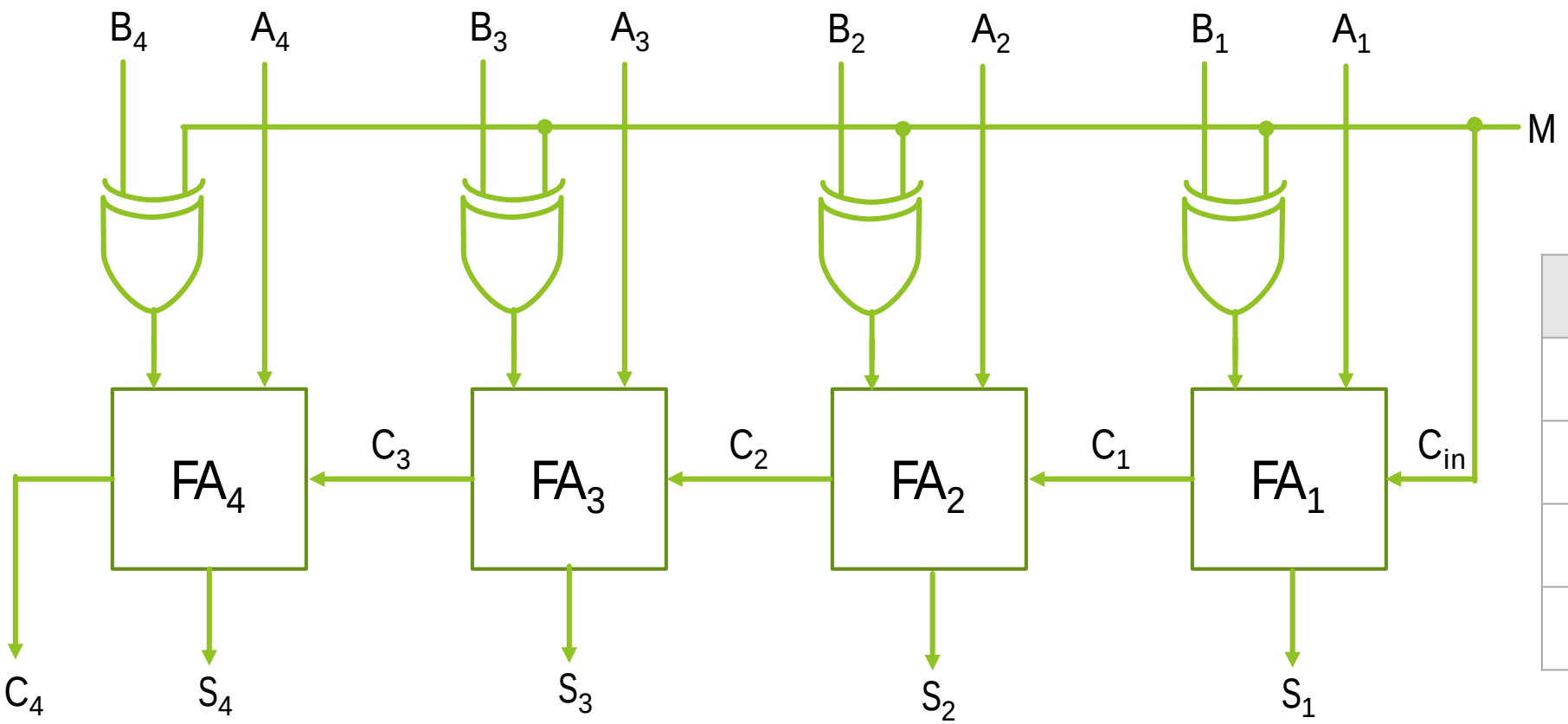
Binary Parallel Subtractor



Example: A = 1101, B = 1011

$A_4\ A_3\ A_2\ A_1$ = 1101
$B_4\ B_3\ B_2\ B_1$ = 1011

Binary Adder-Subtractor



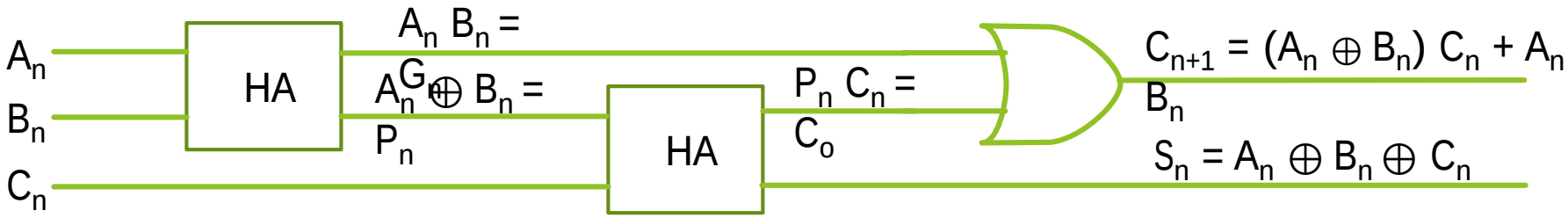
M	B_i	O/P
0	0	0
0	1	1
1	0	1
1	1	0

When M = 0, Circuit is an adder.

When M = 1, Circuit becomes a subtractor.

Look Ahead Carry Adder

- Speeds up the process by eliminating the ripple carry delay.
- The method of speeding up the addition process is based on: carry generate and carry propagate functions.
- Consider one full adder stage; say the nth stage of a parallel adder shown in below Figure.



Look Ahead Carry Adder

- $S_n = P_n \oplus C_n$ where $P_n = A_n \oplus B_n$
- $C_{n+1} = G_n + P_n C_n$ where $G_n = A_n B_n$
- Possible to express the output carry of a higher significant stage in terms of applied input variables A, B and carry-in to the LSB adder.
- Based on these, the expressions for the carry-outs of various full adders are as follows:

$C_2 = G_1 + P_1 C_1$

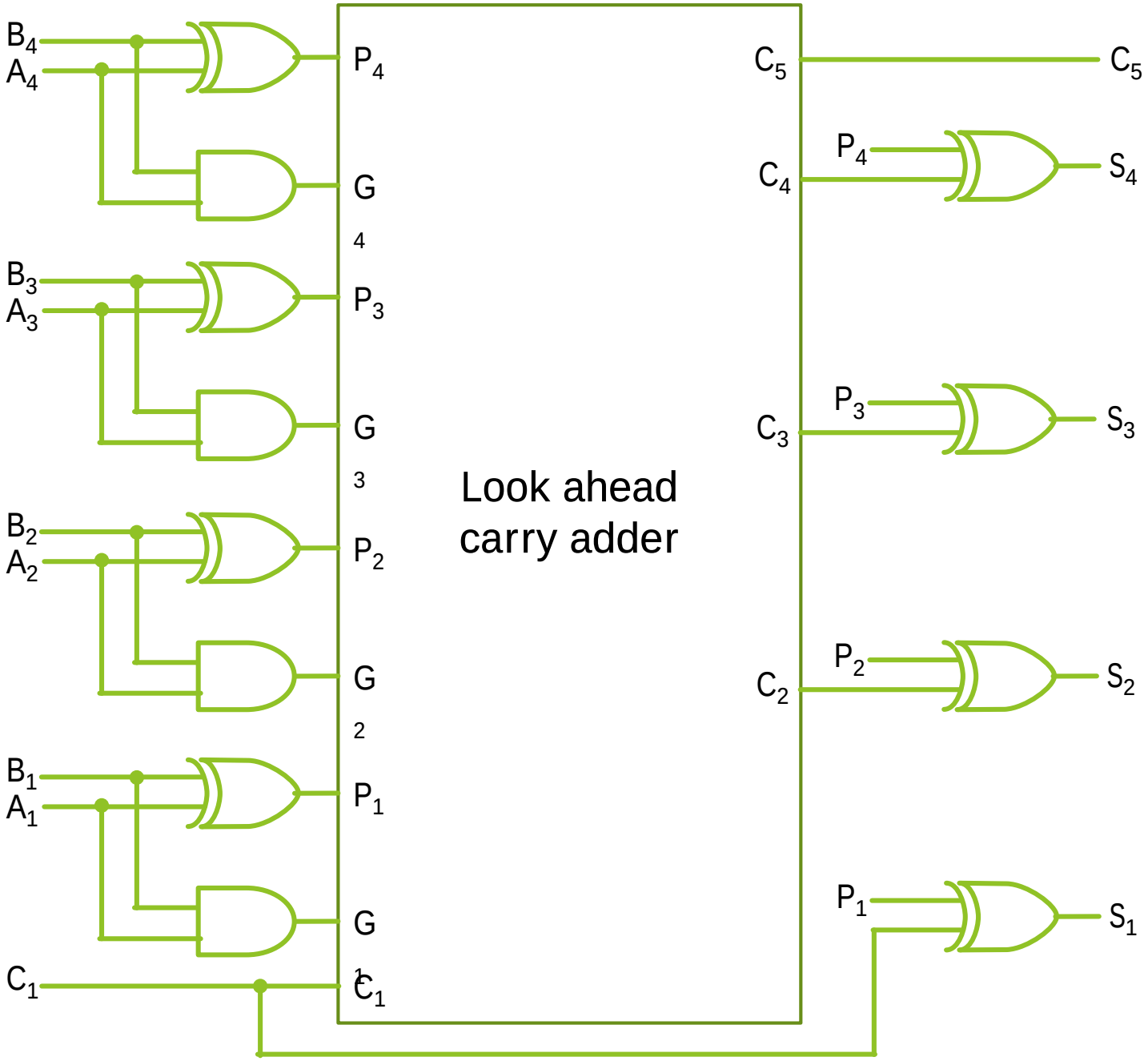
$C_3 = G_2 + P_2 C_2$

$= G_2 + P_2 (G_1 + P_1 C_1)$

$= G_2 + P_2 G_1 + P_2 P_1 C_1$

$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$

$C_5 = G_4 + P_4 C_4 = G_4 + P_4 G_3 + P_4 P_3 G_2 + P_4 P_3 P_2 G_1 + P_4 P_3 P_2 P_1 C_1$


B4
A4
P4
G4
B3
A3
P3
G3
B2
A2
P2
G2
B1
A1
P1
G1
C1
Look ahead carry adder
C5
C5
P4
C4
S4
P3
C3
S3
P2
C2
S2
P1
S1

Combinational Circuits

Section - 5

Binary to Gray Code Converter

$$G_4 = \sum m(8,9,10,11,12,13,14,15)$$

$$G_3 = \sum m(4,5,6,7,8,9,10,11)$$

$$G_2 = \sum m(2,3,4,5,10,11,12,13)$$

$$G_1 = \sum m(1,2,5,6,9,10,13,14)$$

4-bit Binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Binary to Gray Code Converter

- K-Map for G_4, G_3, G_2, G_1 function and their minimization are as follows:

$$G_4 = \sum m(8,9,10,11,12,13,14,15)$$

B_2B_1 \ B_4B_3	00	01	11	10
00	0	4	12 1	8 1
01	1	5	13 1	9 1
11	3	7	15 1	11 1
10	2	6	14 1	10 1

$G_4 = B_4$

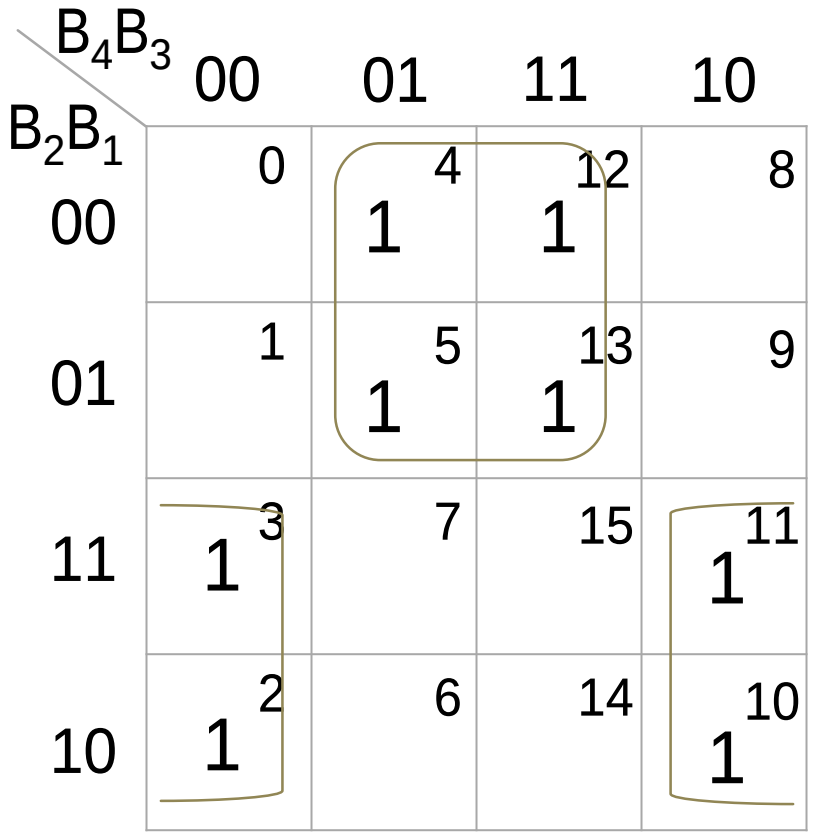
$$G_3 = \sum m(4,5,6,7,8,9,10,11)$$

B_2B_1 \ B_4B_3	00	01	11	10
00	0	4 1	12	8 1
01	1	5 1	13	9 1
11	3	7 1	15	11 1
10	2	6 1	14	10 1

$G_3 = B_4'B_3 + B_4B_3' = B_4 \oplus B_3$

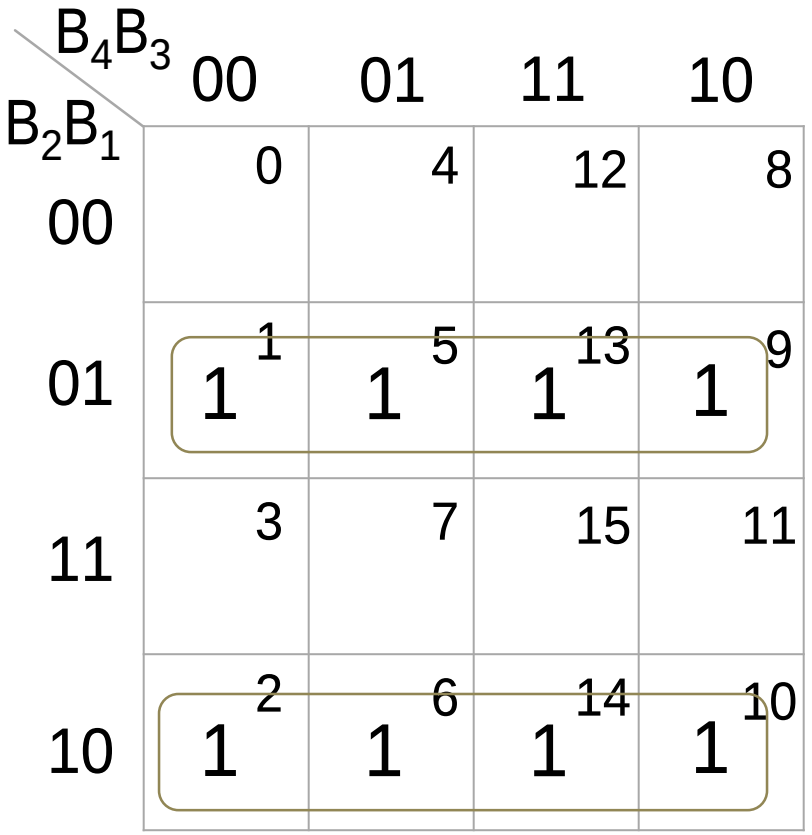
Binary to Gray Code Converter

$$G_2 = \sum m(2,3,4,5,10,11,12,13)$$



$$G_2 = B_3'B_2 + B_3B_2' = B_3 \oplus B_2$$

$$G_1 = \sum m(1,2,5,6,9,10,13,14)$$



$$G_1 = B_2'B_1 + B_2B_1' = B_2 \oplus B_1$$

Binary to Gray Code Converter

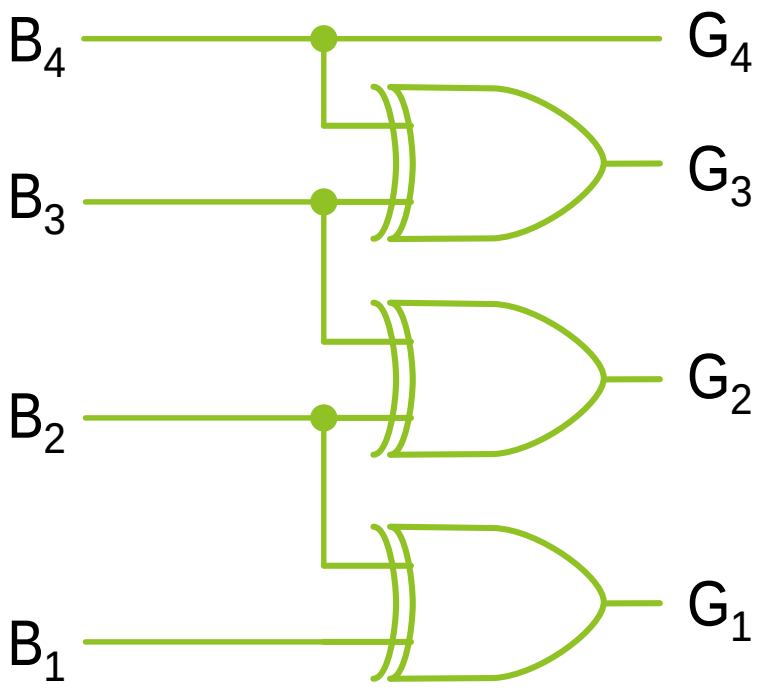
- Logic diagram for binary to Gray code converter is as follows:

$$G_4 = B_4$$

$$G_3 = B_4'B_3 + B_4B_3' = B_4 \oplus B_3$$

$$G_2 = B_3'B_2 + B_3B_2' = B_3 \oplus B_2$$

$$G_1 = B_2'B_1 + B_2B_1' = B_2 \oplus B_1$$



BCD to Excess-3 Code Converter

8421 BCD				XS - 3			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
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

$$X_4 = \sum m(5,6,7,8,9) + d(10,11,12,13,14,15)$$

$$X_3 = \sum m(1,2,3,4,9) + d(10,11,12,13,14,15)$$

$$X_2 = \sum m(0,3,4,7,8) + d(10,11,12,13,14,15)$$

$$X_1 = \sum m(0,2,4,6,8) + d(10,11,12,13,14,15)$$

BCD to Excess-3 Code Converter

- K-Map for X_4, X_3, X_2, X_1 function and their minimization are as follows:

$$X_4 = \sum m(5,6,7,8,9) + d(10,11,12,13,14,15)$$

B_2B_1 \ B_4B_3	00	01	11	10
00	0	4	12 x	8 1
01	1	5 1	13 x	9 1
11	3	7 1	15 x	11 x
10	2	6 1	14 x	10 x

$X_4 = B_4 + B_3B_2 + B_3B_1$

$$X_3 = \sum m(1,2,3,4,9) + d(10,11,12,13,14,15)$$

B_2B_1 \ B_4B_3	00	01	11	10
00	0	4 1	12 x	8
01	1 1	5	13 x	9 1
11	3 1	7	15 x	11 x
10	2 1	6	14 x	10 x

$X_3 = B_3B_2'B_1' + B_3' B_1 + B_3' B_2$

BCD to Excess-3 Code Converter

$$X_2 = \sum m(0,3,4,7,8) + d(10,11,12,13,14,15)$$

B_2B_1 \ B_4B_3	00	01	11	10
00	1 (0)	1 (4)	X (12)	1 (8)
01	(1)	(5)	X (13)	(9)
11	1 (3)	1 (7)	X (15)	X (11)
10	(2)	(6)	X (14)	X (10)

$X_2 = B_2'B_1' + B_2B_1$

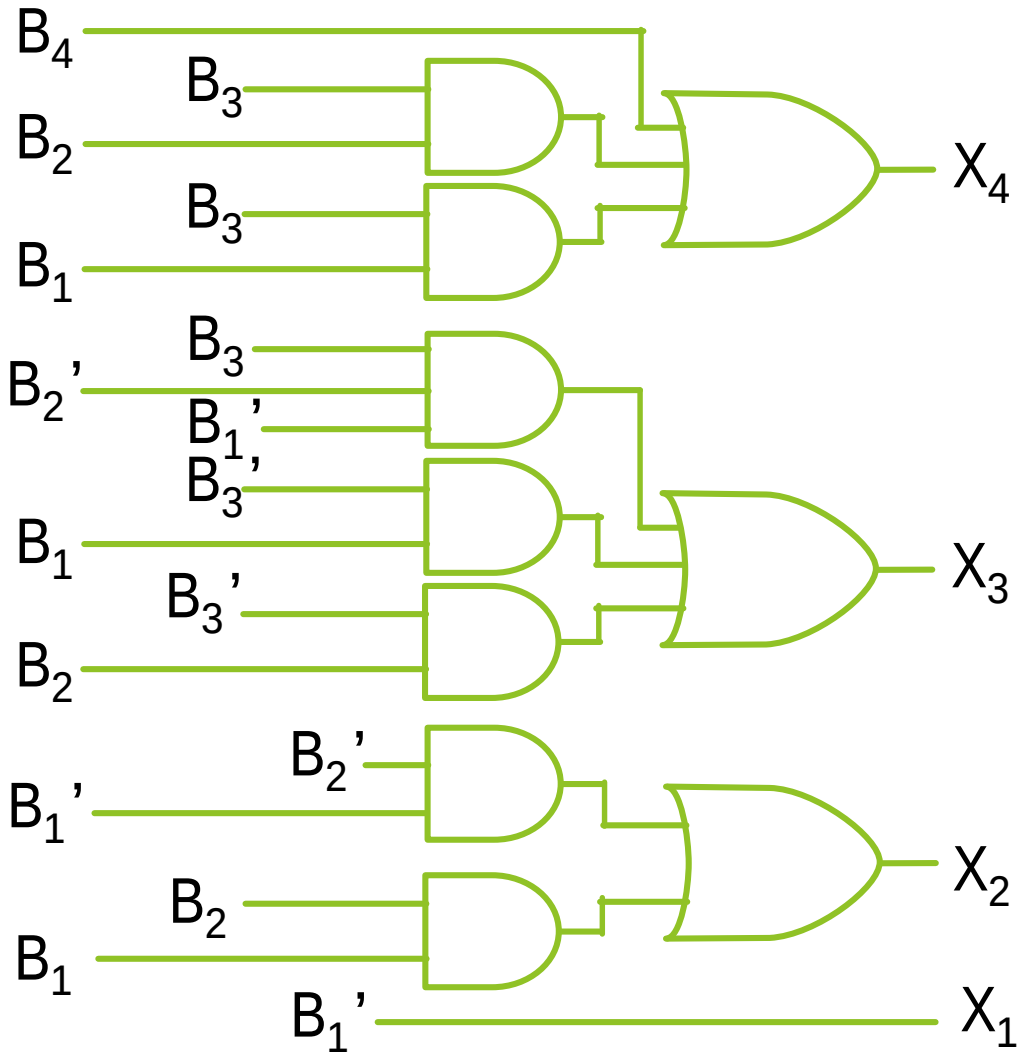
$$X_1 = \sum m(0,2,4,6,8) + d(10,11,12,13,14,15)$$

B_2B_1 \ B_4B_3	00	01	11	10
00	1 (0)	1 (4)	X (12)	1 (8)
01	(1)	(5)	X (13)	(9)
11	(3)	(7)	X (15)	X (11)
10	1 (2)	1 (6)	X (14)	X (10)

$X_1 = B_1'$

BCD to Excess-3 Code Converter

- Logic diagram for a BCD to Excess-3 is as follows



Comparators

- A comparator is a logic circuit used to compare the magnitudes of two binary numbers.
- Comparator circuit provides 3 outputs
  1. A = B
  2. A > B
  3. A < B
- X-NOR gate is a basic comparator (the output is 1 if and only if the input bits coincide).
- 2 binary numbers are equal if and only if all their corresponding bits coincide.
- E.g. $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are equal if and only if $A_3=B_3$, $A_2=B_2$, $A_1=B_1$ and $A_0=B_0$

$$\text{Equality} = (A_3 \odot B_3)\ (A_2 \odot B_2)\ (A_1 \odot B_1)\ (A_0 \odot B_0)$$

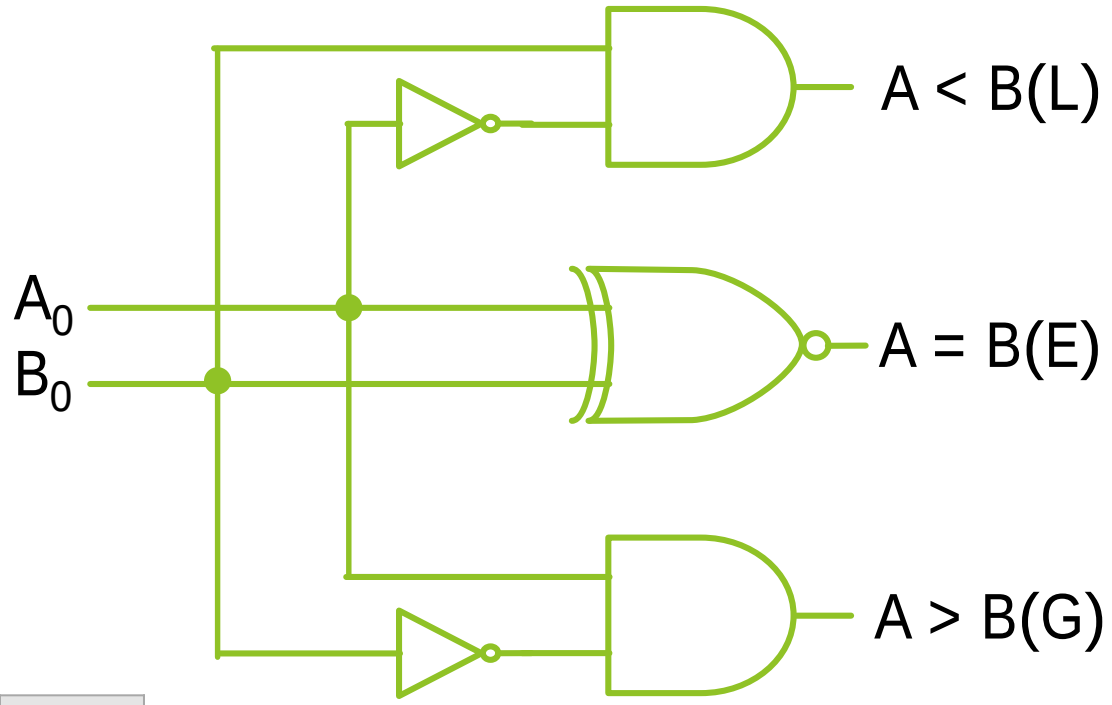
1-bit Magnitude Comparator

- Let the 1-bit numbers be $A = A_0$ and $B = B_0$
- If $A_0 = 1$ and $B_0 = 0$ then $A > B$

  $A > B : G = A_0B_0'$
- If $A_0 = 0$ and $B_0 = 1$ then $A < B$

  $A < B : L = A_0'B_0$
- If $A_0 = 1$ and $B_0 = 1$ (coincides) then $A = B$

  $A = B : E = A_0 \odot B_0$



A_0	B_0	L	E	G
0	0			
0	1			
1	0			
1	1			

2-bit Magnitude Comparator

- Let the two 2-bit numbers be $A = A_1A_0$ and $B = B_1B_0$

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$.

$$A > B : G = A_1B_1' + (A_1 \odot B_1)\, A_0B_0'$$

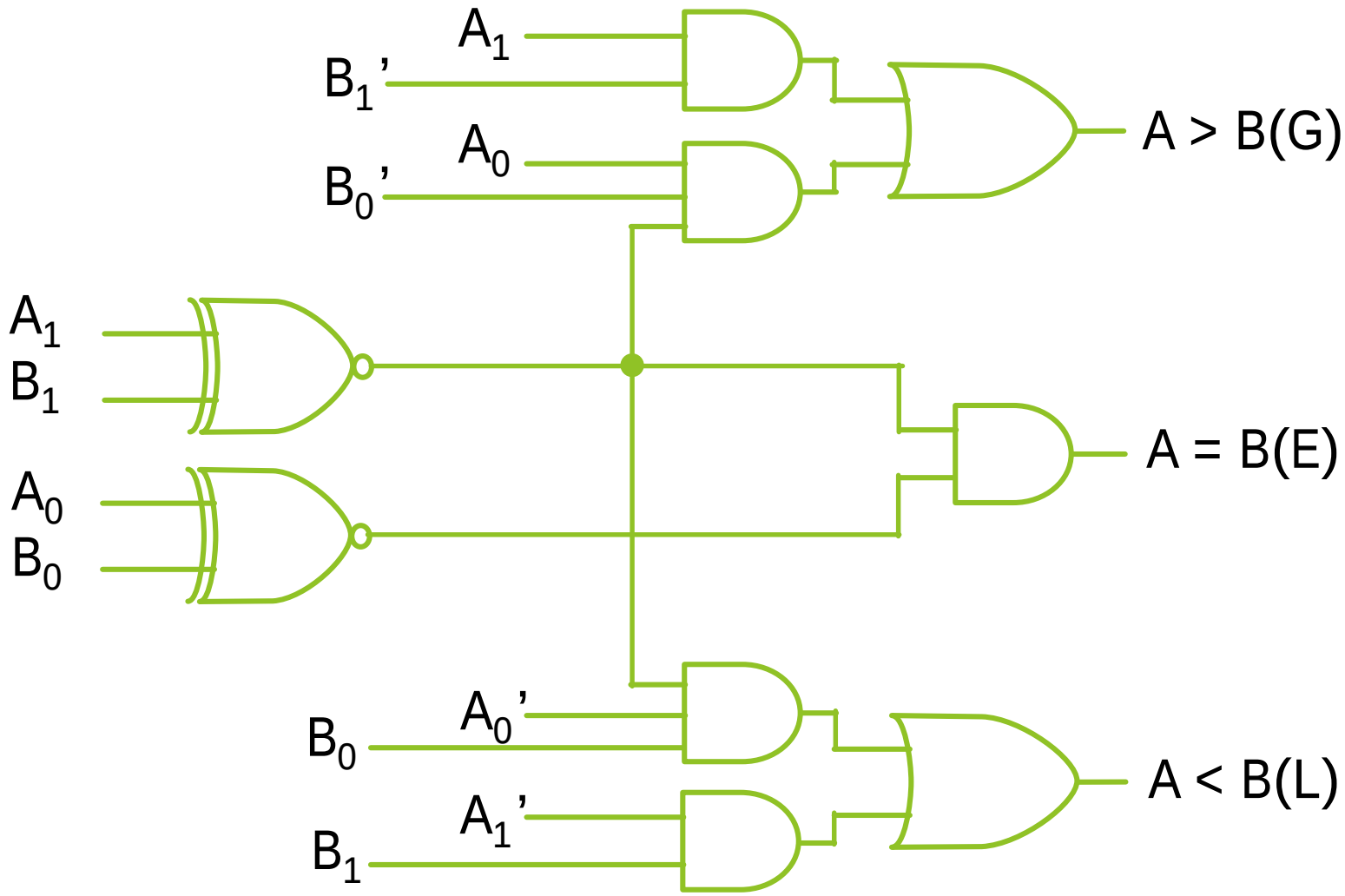
1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A < B$.

$$A < B : L = A_1'B_1 + (A_1 \odot B_1)\, A_0'B_0$$

1. If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$.

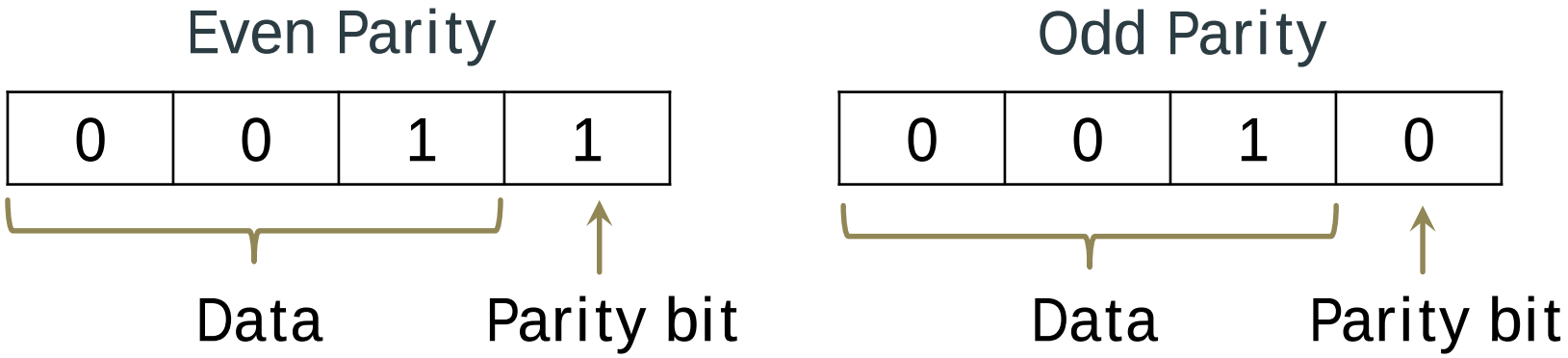
$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$

2-bit Magnitude Comparator



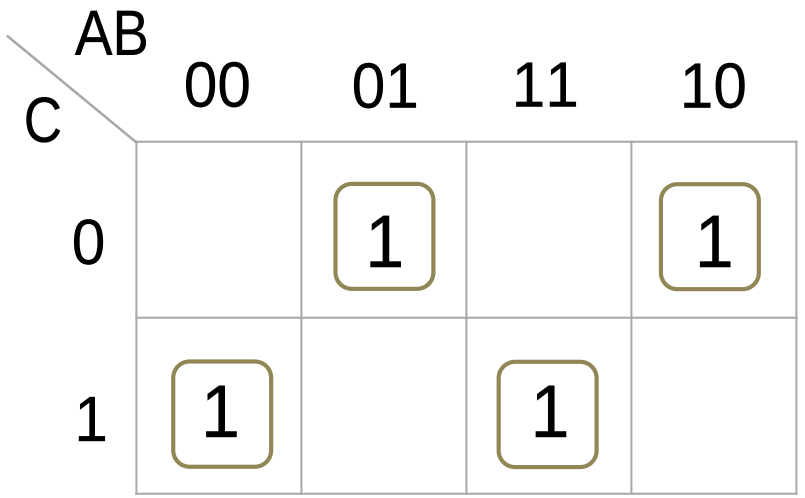
Parity Generator

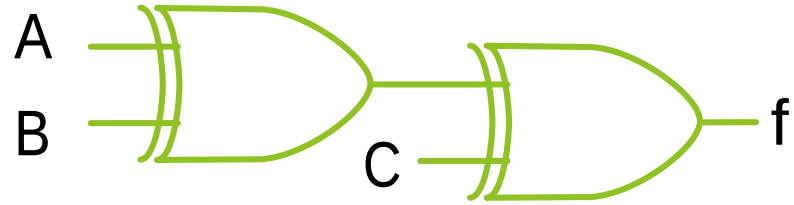
- Binary data, when transmitted and processed is susceptible to noise that can alter its 1s to 0s and 0s to 1s.
- To detect such errors, an additional bit called parity bit is added to the data bits and the word containing the data bits and the parity bit is transmitted.
- At the receiving end the number of 1s in the word received are counted and the error, if any, is detected.



3-bit parity generator using even parity bit

Inputs			Outputs parity bit (f)
A	B	C	
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





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

$$f = A'(B'C + BC') + A(BC + B'C')$$

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

$$f = A \oplus B \oplus C$$

Multiplexer/De-multiplexer, Encoder/Decoder

Section - 6

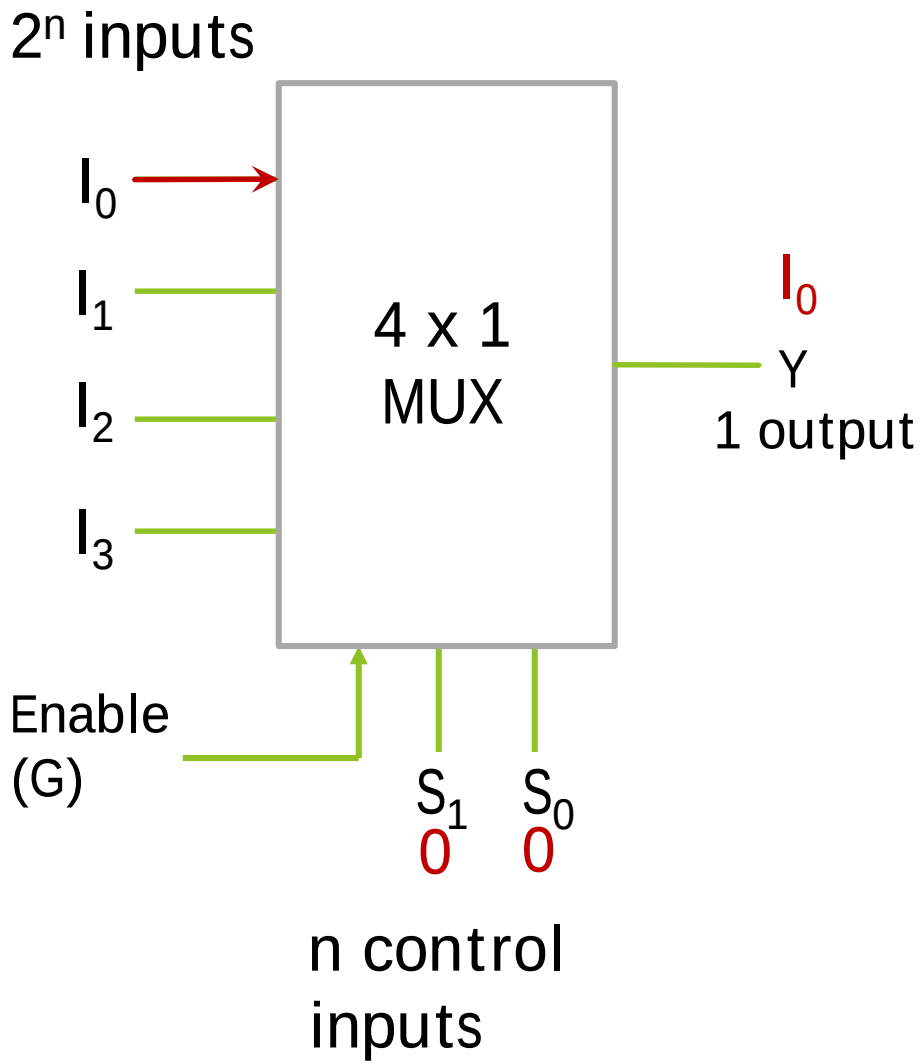
Multiplexer

- A multiplexer(MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.
- Consider an integer ‘m’, which is constrained by the following relation:

  $m = 2^n$, where m and n are both integers.
- A m-to-1 Multiplexer has
  - m Inputs: $I_0, I_1, I_2, \ldots\ldots\ldots\ldots\ldots I_{(m-1)}$
  - One Output: Y
  - n Control inputs: $S_0, S_1, S_2, \ldots\ldots S_{(n-1)}$
  - One (or more) Enable input(s)

Such that Y may be equal to one of the inputs, depending upon the control inputs.

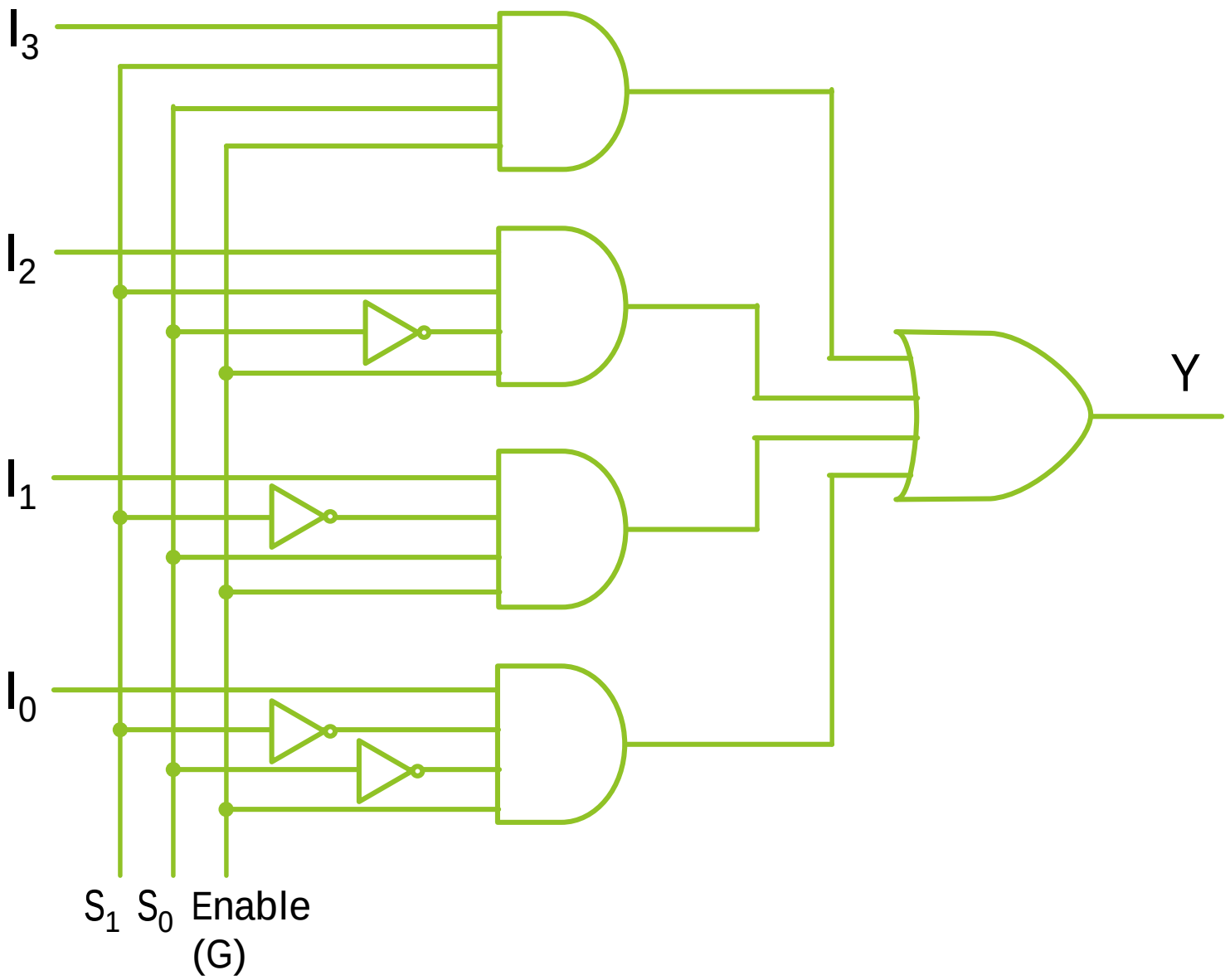
4-to-1 Multiplexer



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

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

4 x 1 MUX Actual Circuit

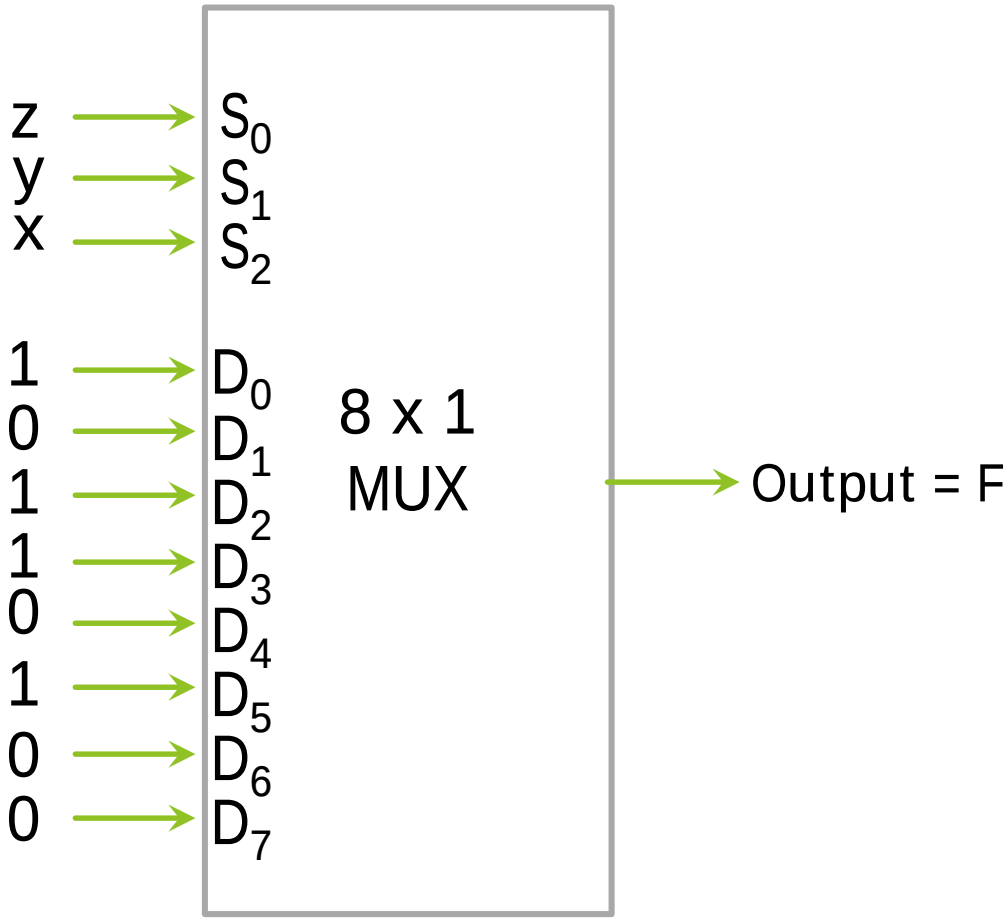


Logic function generator using Multiplexer

- Implement the following function using 8 to 1 MUX

$$F(x,y,z) = \sum_m(0,2,3,5)$$

S_2	S_1	S_0	F
x	y	z	
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



Logic function generator using Multiplexer (Method)

- Multiplexer with n-data select inputs can implement any function of n + 1 variables.
- The first n variables of the function as the select inputs and to use the least significant input variable and its complement to drive some of the data inputs.
- If the single variable is denoted by D, each data output of the multiplexer will be D, D', 1, or 0.
- Suppose, we wish to implement a 4-variable logic function using a multiplexer with three data select inputs.
- Let the input variables be A, B, C, and D; D is the LSB.
- A truth table for the function F(A, B, C, D) is constructed with ABC has the same value twice once with D = 0 and again with D = 1.

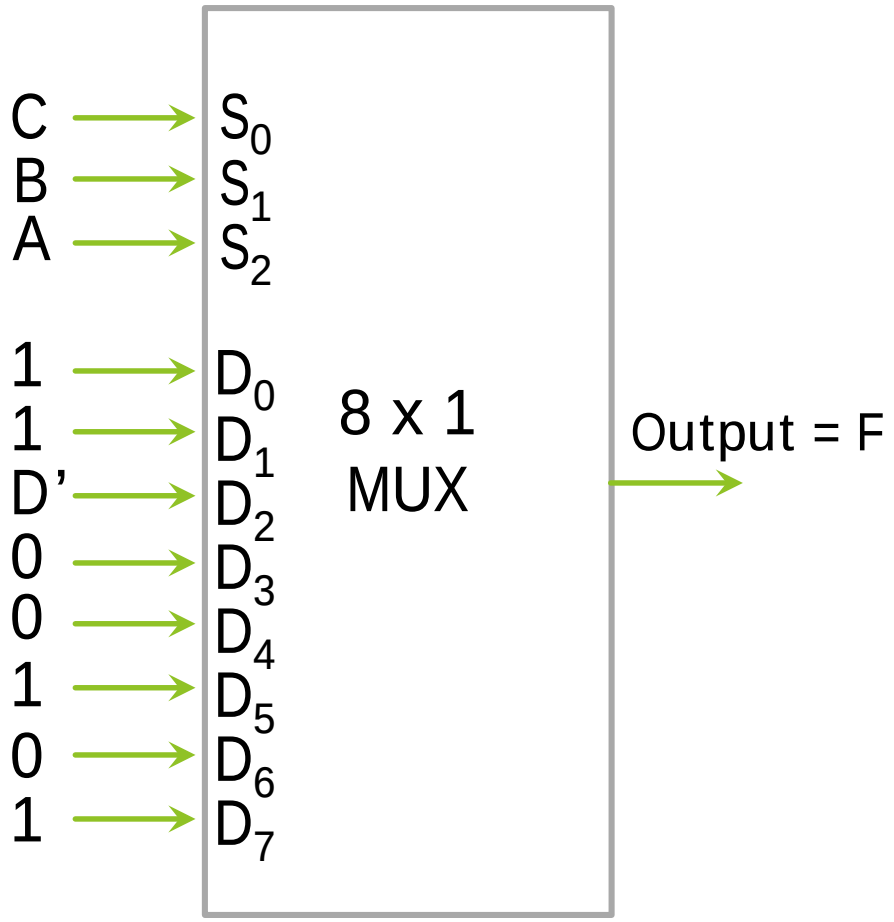
Logic function generator using Multiplexer (Method)

- The following rules are used to determine the connections that should be made to the data inputs of the multiplexer.
  1. If F = 0 both times when the same combination of ABC occurs, connect logic 0 to the data input selected by that combination.
  2. If F = 1 both times when the same combination of ABC occurs, connect logic 1 to the data input selected by that combination.
  3. If F is different for the two occurrences of a combination of ABC, and if F = D in each case, connect D to the data input selected by that combination.
  4. If F is different for the two occurrences of a combination of ABC, and if F = D’ in each case, connect D’ to the data input selected by that combination.

Implement the following function using 8 to 1 MUX

$F = \sum_m(0,1,2,3,4,10,11,14,15)$

S_2	S_1	S_0	D	F	
A	B	C			
0	0	0	0	1	
0	0	0	1	1	
0	0	1	0	1	
0	0	1	1	1	
0	1	0	0	1	
0	1	0	1	0	
0	1	1	0	0	
0	1	1	1	0	
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	1	
1	0	1	1	1	
1	1	0	0	0	
1	1	0	1	0	
1	1	1	0	1	
1	1	1	1	1	



Demultiplexer

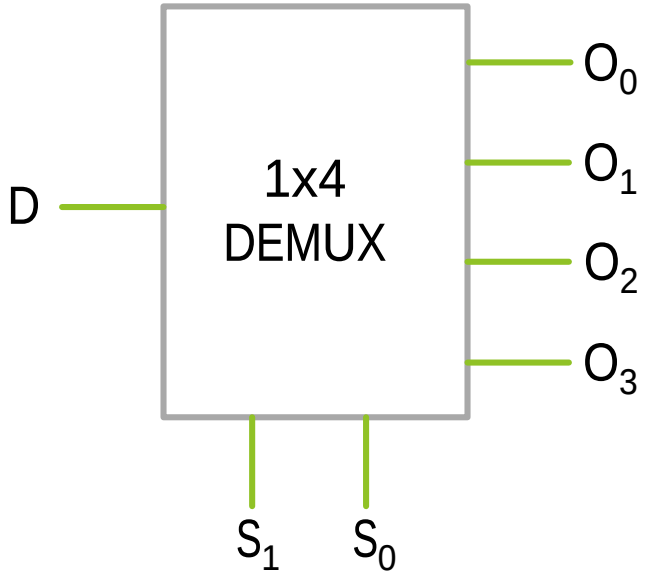
- A demultiplexer(DEMUX) is a device that allows digital information from one source to be routed onto a multiple lines for transmission over different destinations.
- Consider an integer ‘m’, which is constrained by the following relation:

  $m = 2^n$, where m and n are both integers.

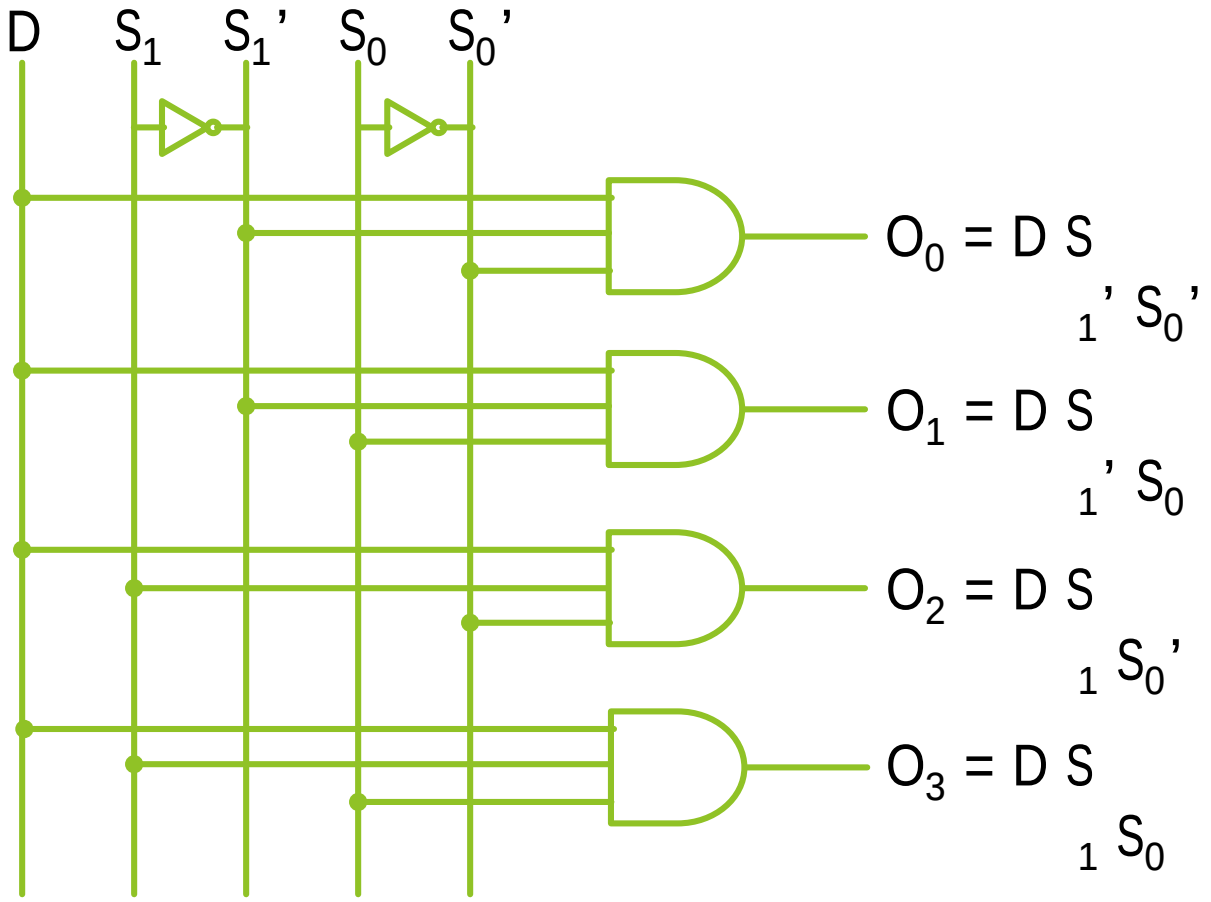
- A 1-to-m Demultiplexer has
  - One Input: D
  - m Outputs: O_0, O_1, O_2, ... $O_{(m-1)}$
  - n Control inputs: S_0, S_1, S_2, ... $S_{(n-1)}$
  - One (or more) Enable input(s)

  Such that D may be transfer to one of the outputs, depending upon the control inputs.

1-to-4 Demultiplexer

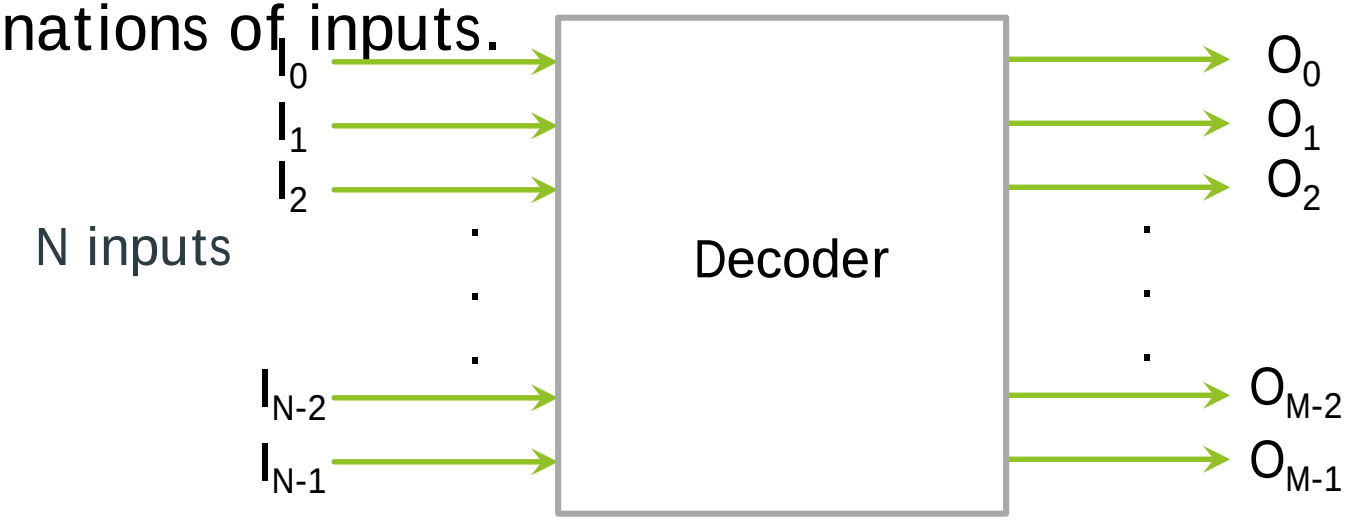


Select code		Outputs			
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0



Decoder

- A decoder is a logic circuit that accepts a set of inputs which represents a binary number and activates the only output that corresponds to the input number.
- In other words, a decoder circuit looks at its inputs, determines which binary number is present there, and activates the specific output which corresponds to that number; all other outputs remain inactive.
- In its general form, a decoder has N input lines to handle N bits and M output lines such that only one output line is activated for each one of the possible combinations of inputs.

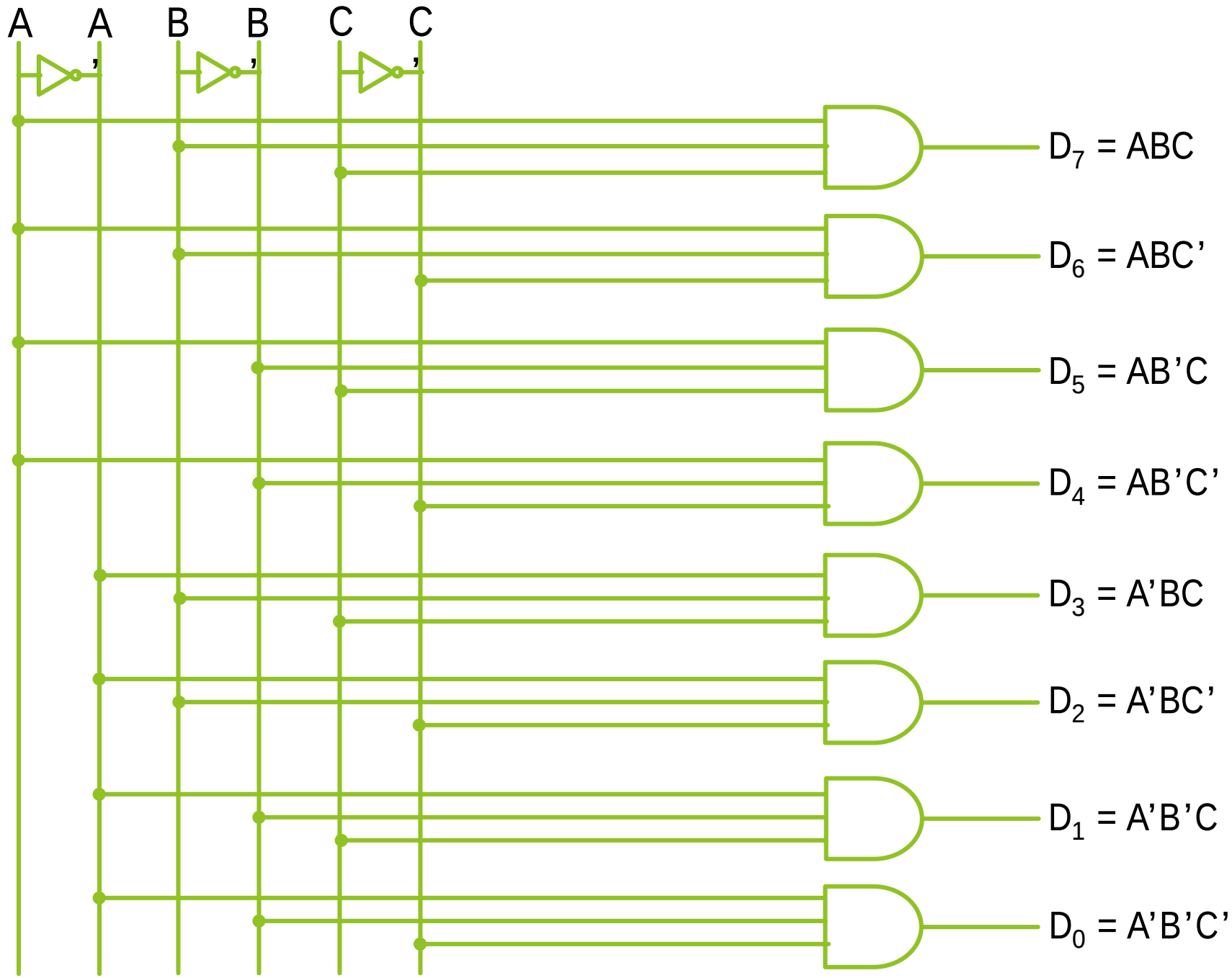


$M = 2^N$

3-Line to 8-Line Decoder

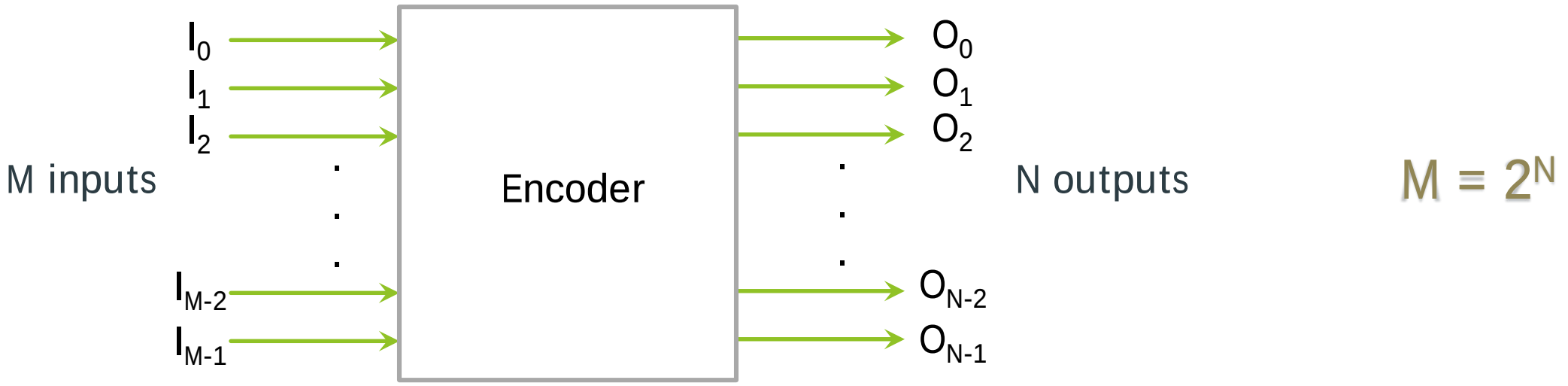
Inputs			Outputs							
A	B	C	D_0 A'B'C'	D_1 A'B'C	D_2 A'BC'	D_3 A'BC	D_4 AB'C'	D_5 AB'C	D_6 ABC'	D_7 ABC
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

- 3 to 8 line decoder can be implemented using AND gates to achieve active-HIGH output.
- For active-LOW outputs, NAND gates are used.


A
A'
B
B'
C
C'
D7 = ABC
D6 = ABC'
D5 = AB'C
D4 = AB'C'
D3 = A'BC
D2 = A'BC'
D1 = A'B'C
D0 = A'B'C'

Encoder

- Device to convert familiar numbers or symbols into coded format.
- It has a number of input lines, only one of which is activated at a given time, and produces an N-bit output code depending on which input is activated.
- Figure shows the block diagram of an encoder with M inputs and N outputs.



Priority Encoder

- A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously HIGH.
- The most common priority system is based on the relative magnitudes of the inputs; whichever decimal input is the largest, is the one that is encoded.
- For example, if both decimal 3 and decimal 4 are activated simultaneously, then a priority encoder would encode decimal 4.

Inputs				Outputs		
D_0	D_1	D_2	D_3	A	B	V
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

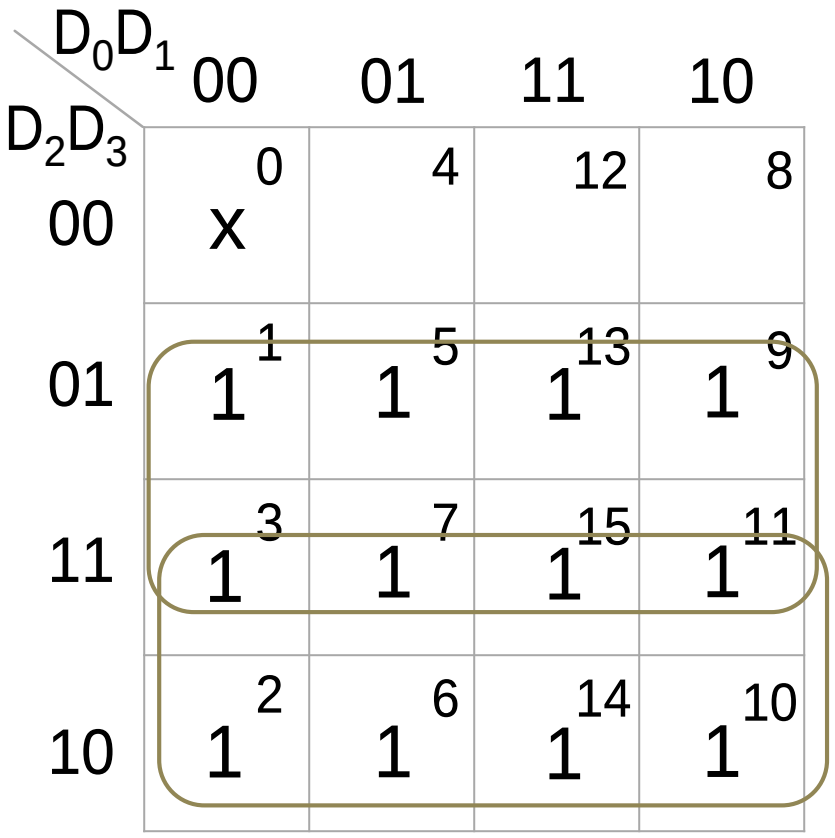
$$A = \sum_m (1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$$

$$B = \sum_m (1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$$

$$V = \sum_m (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$

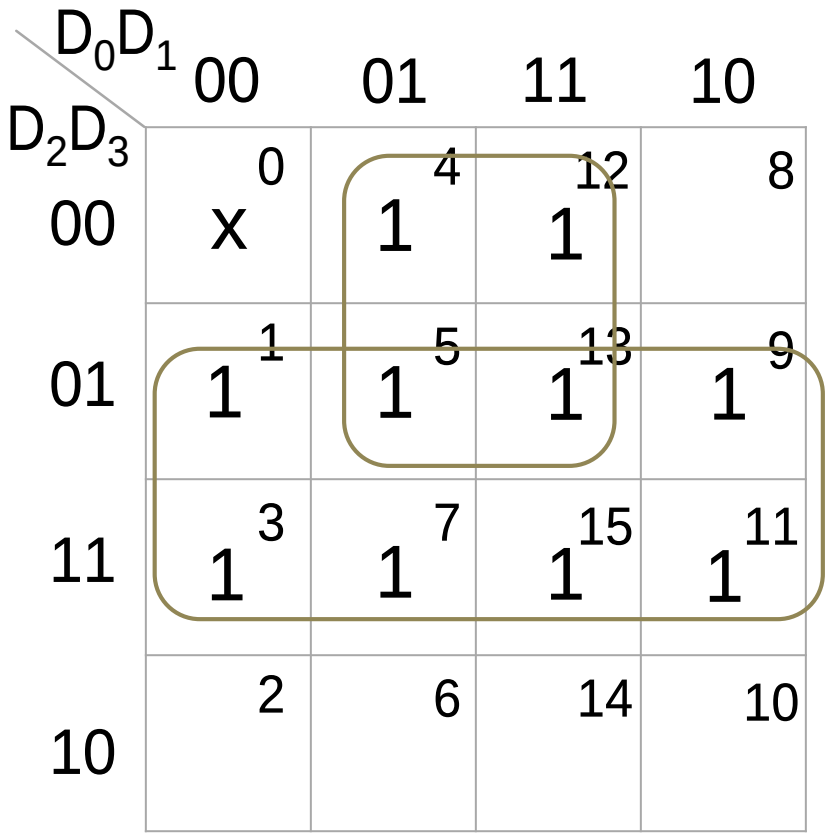
Priority Encoder

$$A = \sum_m (1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$$



A = D_3 + D_2

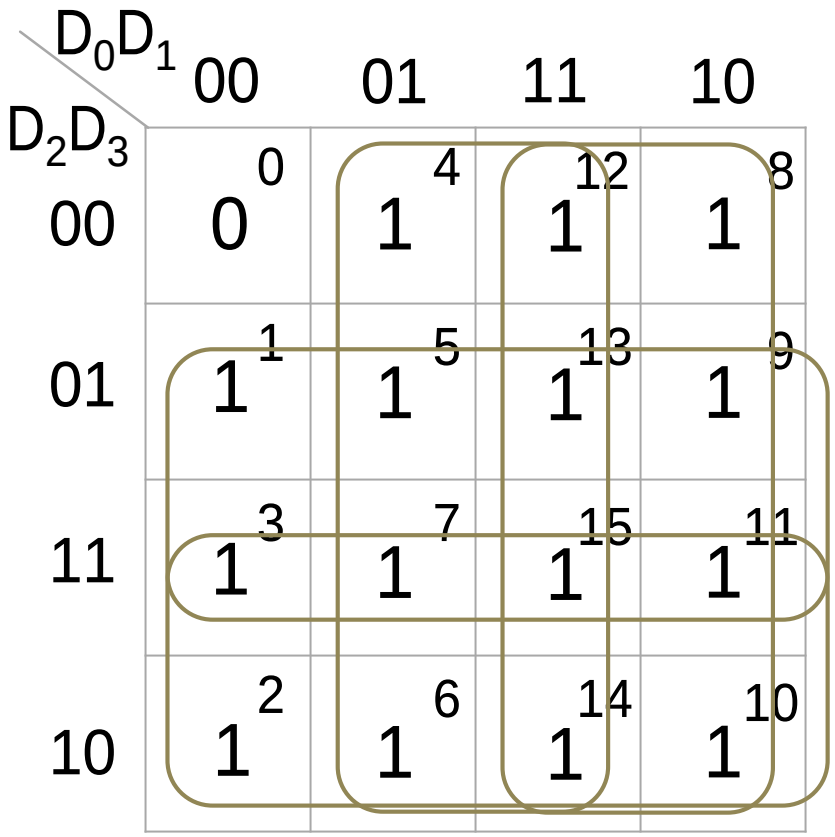
$$B = \sum_m (1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$$



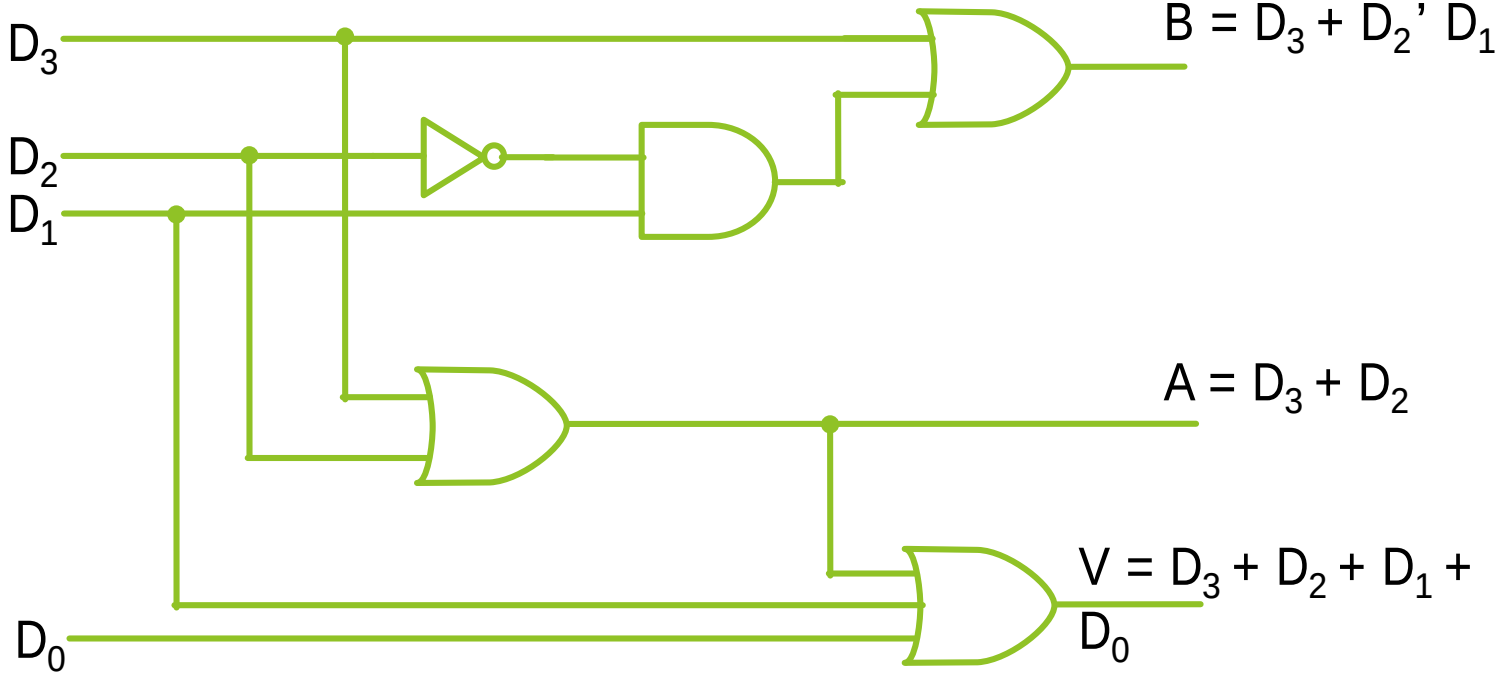
B = D_3 + D_2’ D_1

Priority Encoder

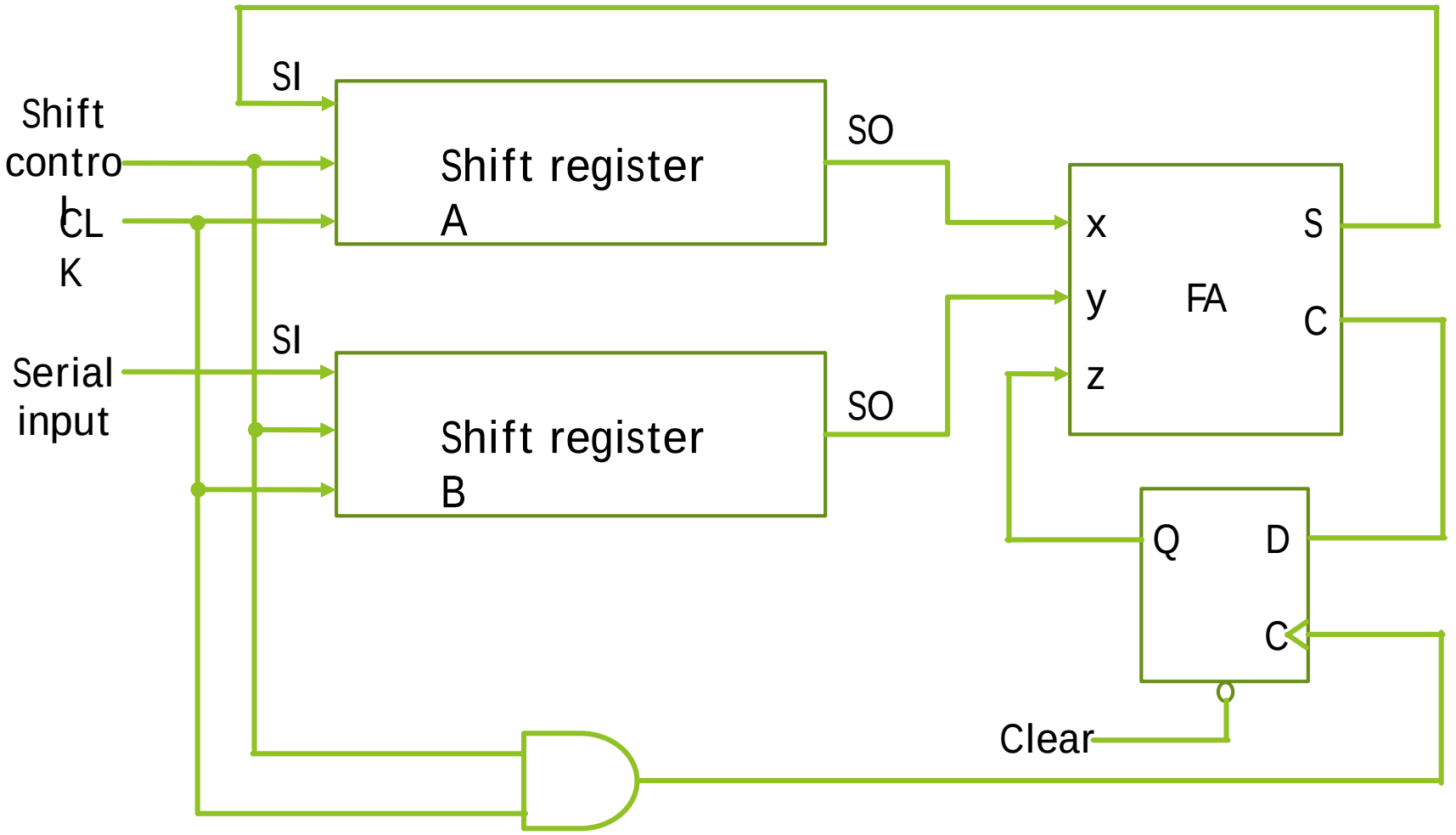
$$V = \sum_m (1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15)$$



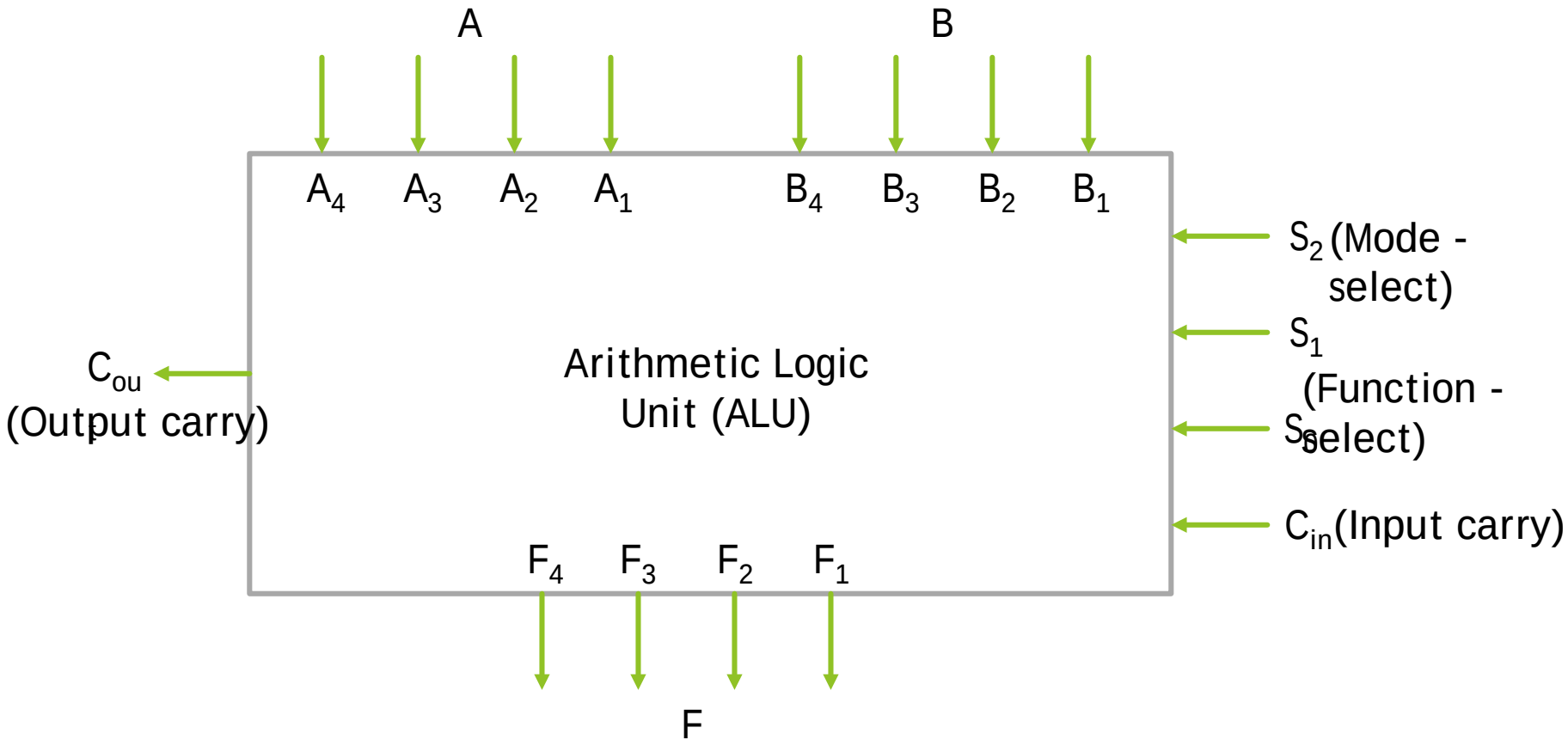
$V = D_3 + D_2 + D_1 + D_0$



Serial Adder



Arithmetic Logic Unit (ALU)



Arithmetic Logic Unit (ALU)

Selection				Output	Function
S_2	S_1	S_0	C_{in}		
0	0	0	0	F = A	Transfer A
0	0	0	1	F = A + 1	Increment A
0	0	1	0	F = A + B	Addition
0	0	1	1	F = A + B + 1	Add with carry
0	1	0	0	F = A – B – 1	Subtract with borrow
0	1	0	1	F = A – B	Subtraction
0	1	1	0	F = A – 1	Decrement A
0	1	1	1	F = A	Transfer A
1	0	0	X	F = A + B	OR
1	0	1	X	F = A $\oplus$ B	XOR
1	1	0	X	F = A . B	AND
1	1	1	X	F = A’	Complement A

Sequential logic circuit

Digital Electronics (DE)

Information and Communication Technology

Prepared by Prof.jaydip siyara

Outline

- Ripple/Asynchronous Counters
  - Ripple up-down, Modulo or Modulus counter
- Synchronous Counters
  - Up counter, Down counter using different FFs
- Decade and BCD counter
- Presettable Counters
- Decoding a Counter
- Designing Counters with Arbitrary Sequences
- Registers
  - Buffer, Shift, Bidirectional, Universal
  - applications of shift registers,
    - serial to parallel converter, parallel to serial converter
    - ring counter, sequence generator,

UNIT - 5

Registers

Section - 2

Registers

- As a flip-flop (FF) can store only one bit of data, a 0 or a 1, it is referred to as a single-bit register.
- A register is a group of FFs used to store binary data.
- The storage capacity of a register is the number of bits (1s and 0s) of digital data it **can retain.**
- Loading a register means setting or resetting the individual FFs, i.e. inputting data into the register so that their states correspond to the bits of data to be stored.
- To store **n bit n D/SR/JK FF** are connected together.
- Loading may be serial or parallel.
- In serial loading, data is transferred into the register in serial form i.e. one bit at a time.
- In parallel loading, the data is transferred into the register in parallel form meaning that all the FFs are triggered into their new states at the same time.
- Types of registers are:
  1. Shift register,
  2. Bidirectional shift register,
  3. Universal shift register

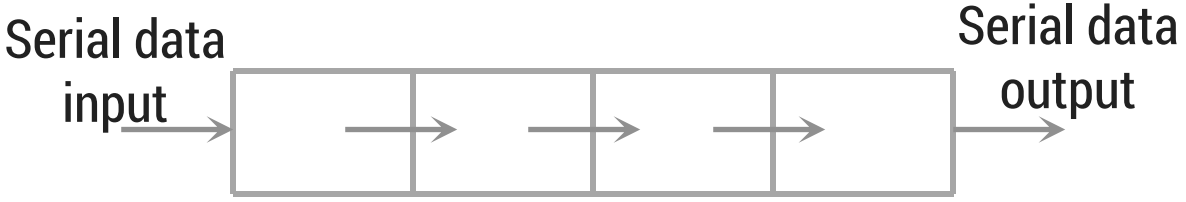
Shift Register

- A number of FFs connected together such that data may be shifted into and shifted out of them is called a shift register.
- Data may be shifted into or out of the register either in serial form or in parallel form.
- So, there are four basic types of shift registers:
  1. serial-in, serial-out (SISO)
  2. serial-in, parallel-out (SIPO)
  3. parallel-in, serial-out(PISO)
  4. parallel-in, parallel-out(PIPO) (Buffer register)

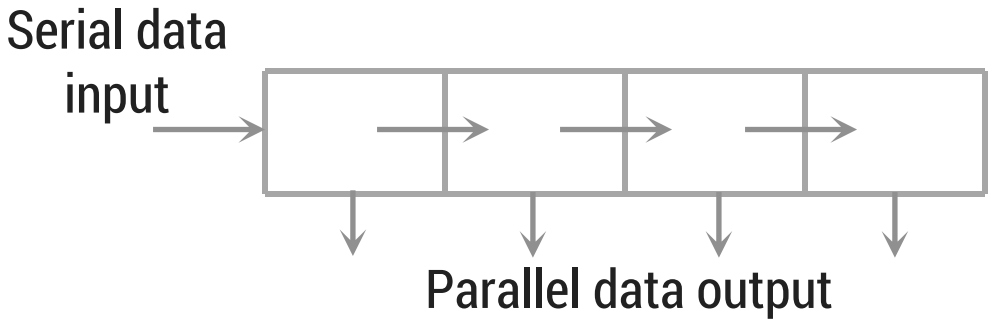
Find-out IC name

- Data may be rotated left or right. Data may be shifted from left to right or right to left at will, i.e. in a bidirectional way.
- Also, data may be shifted in serially (in either way) or in parallel and shifted out serially (in either way) or in parallel.

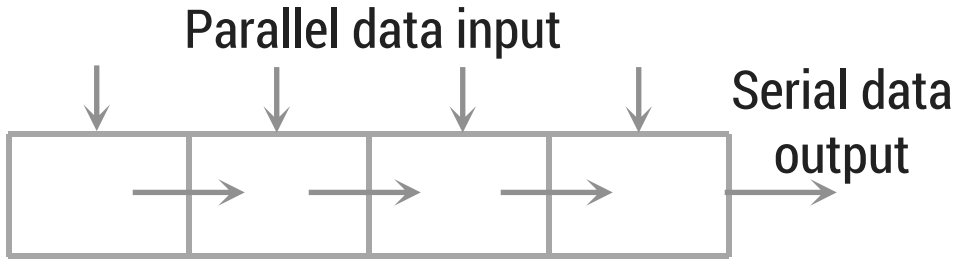
Data transmission in shift register



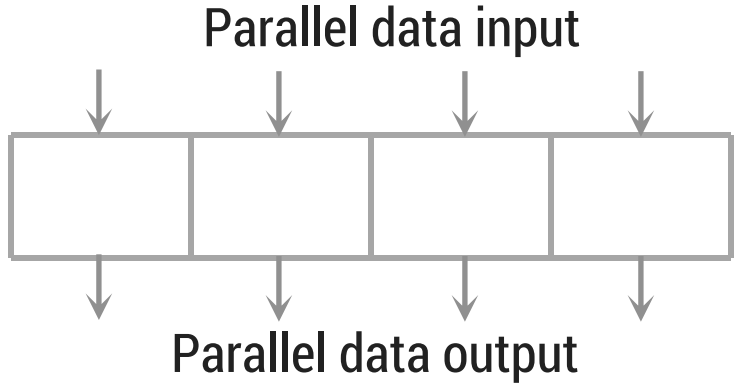
Serial-in, serial-out shift-right, shift register



Serial-in, parallel-out, shift register



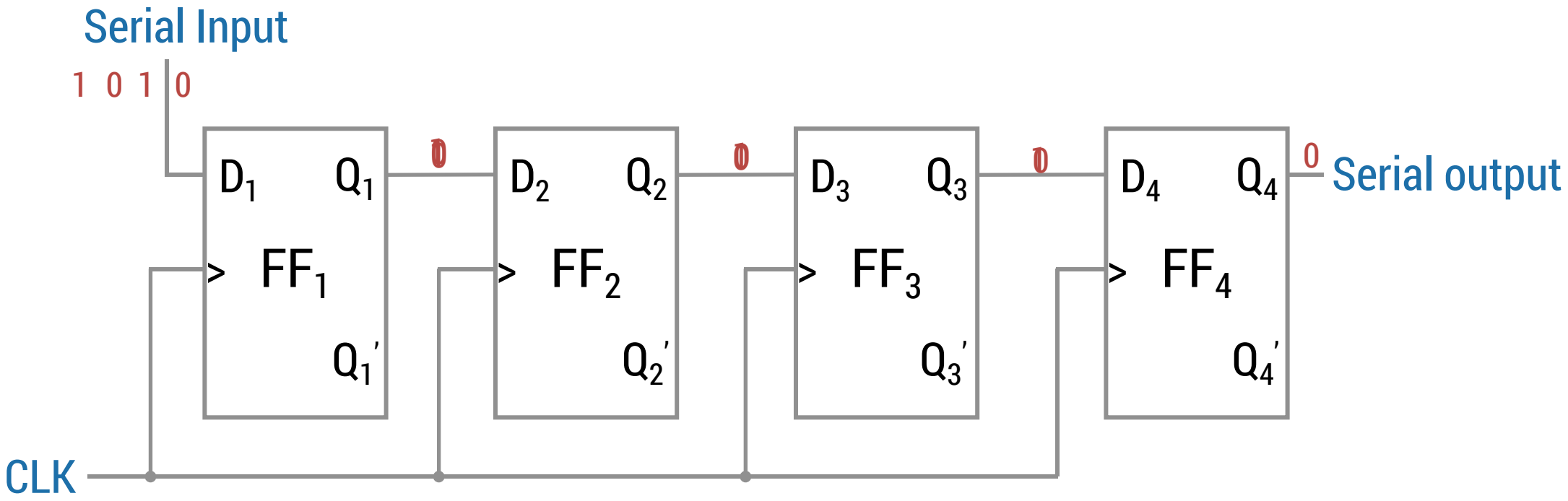
Parallel-in, serial-out, shift register



Parallel-in, parallel-out, shift register

Serial-in, Serial-out, Shift-right, Shift register

To store 4bit 4 D Flip-flops are connected.



Data = 1010

In SISO n number of clock pulses are required to store n bit word.

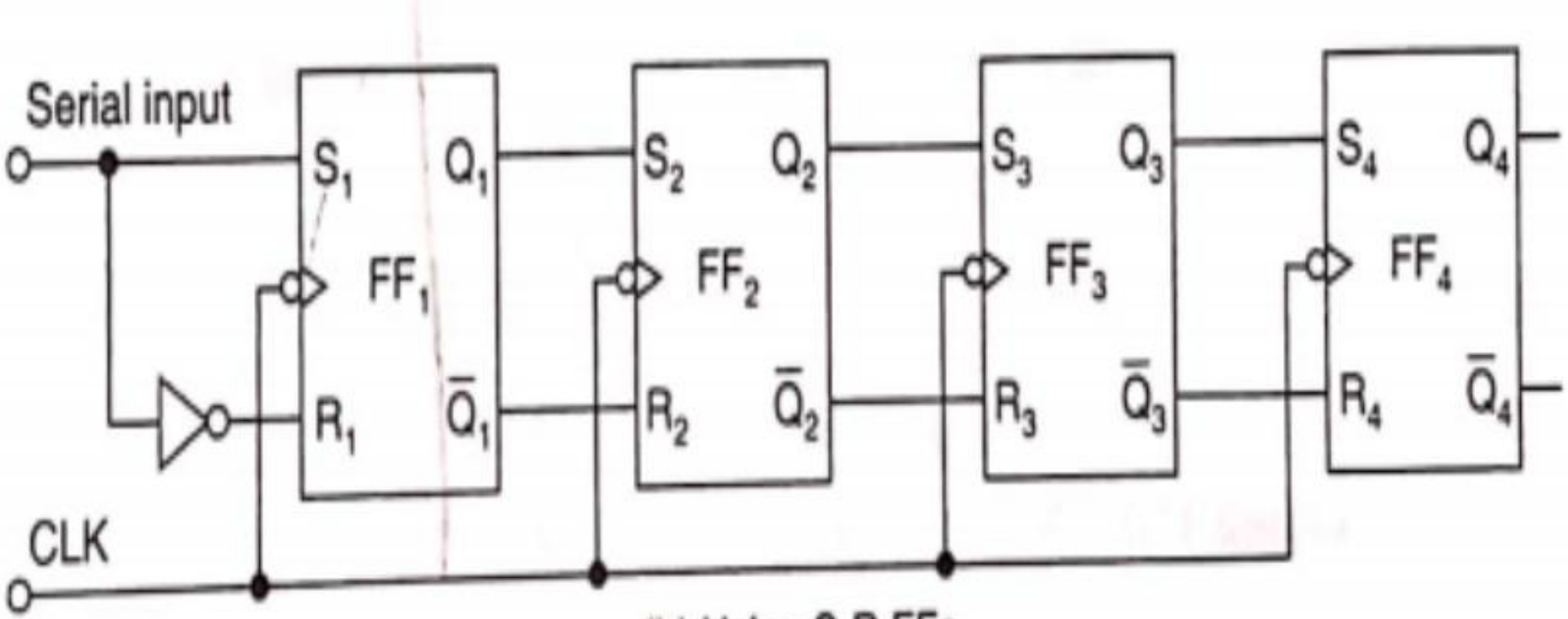
For description refer this site: http://studytronics.weebly.com/shift-registers.html

If data is 1011.Truth table is as below,

Timing pulse	Q_A	Q_B	Q_C	Q_D
Initial value	0	0	0	0
After 1st clock pulse	1	0	0	0
After 2nd clock pulse	1	1	0	0
After 3rd clock pulse	0	1	1	0
After 4th clock pulse	1	0	1	1

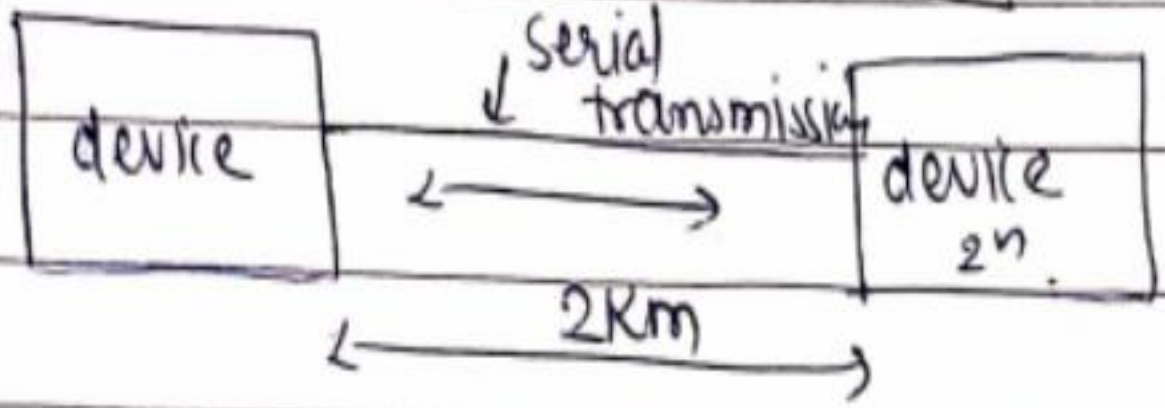
To the waveform see this you tube video: https://www.youtube.com/watch?v=zazz_LH1Sxo

SISO using SR FF



(b) Using S-R FFs

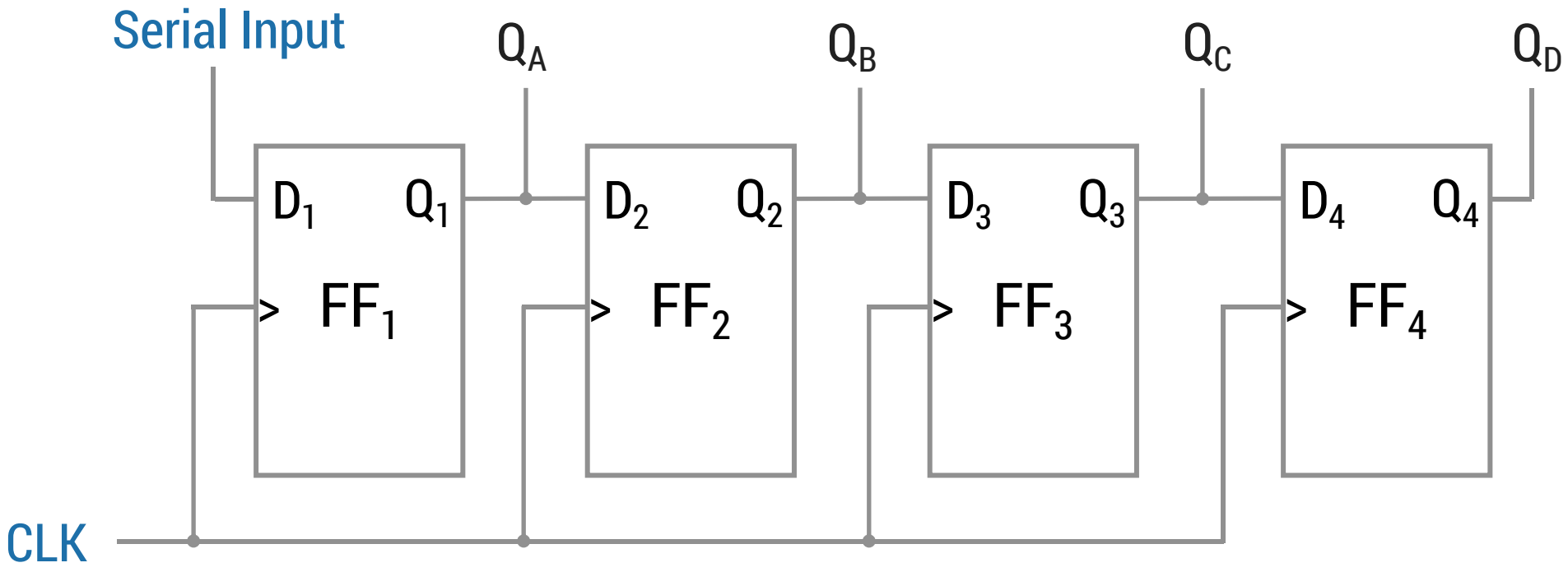
* Application of serial operation :-



a) 1 bit will be transmitted in 1 clock pulse. ⇒ Speed ↓

b) when distance is large only one conductor will be required.

Serial In Parallel Out using DFF(SIPO)

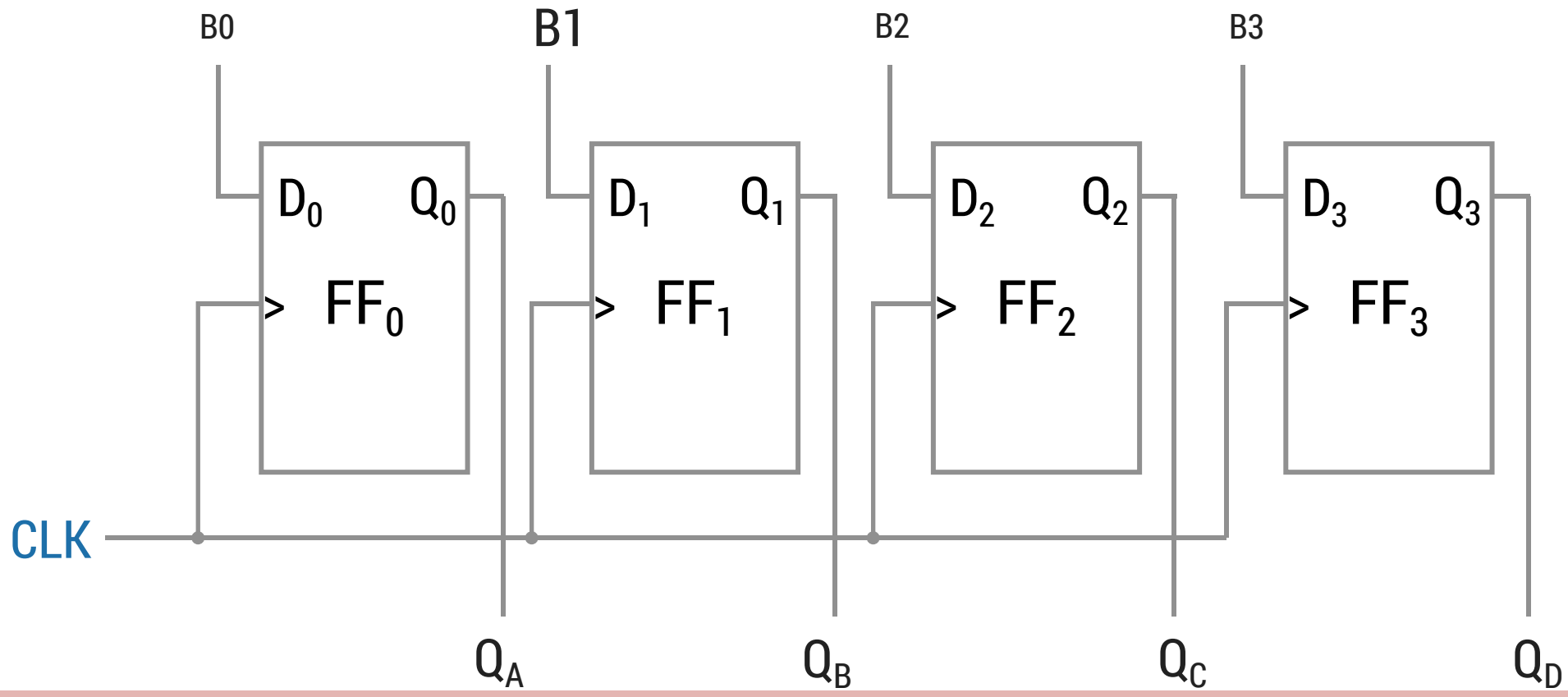


For description refer this site:
http://studytronics.weebly.com/shift-registers.html .

Serial In Parallel Out using DFF(SIPO)

1. In such types of operations, the data is entered serially and taken out in parallel fashion.
2. Data is loaded bit by bit. The outputs are disabled as long as the data is loading.
3. As soon as the data loading gets completed, all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output lines at the same time.
4. 4 clock cycles are required to load a four bit word. Hence the speed of operation of SIPO mode is same as that of SISO mode.

Parallel-in, Parallel-out, Shift register(PIPO)



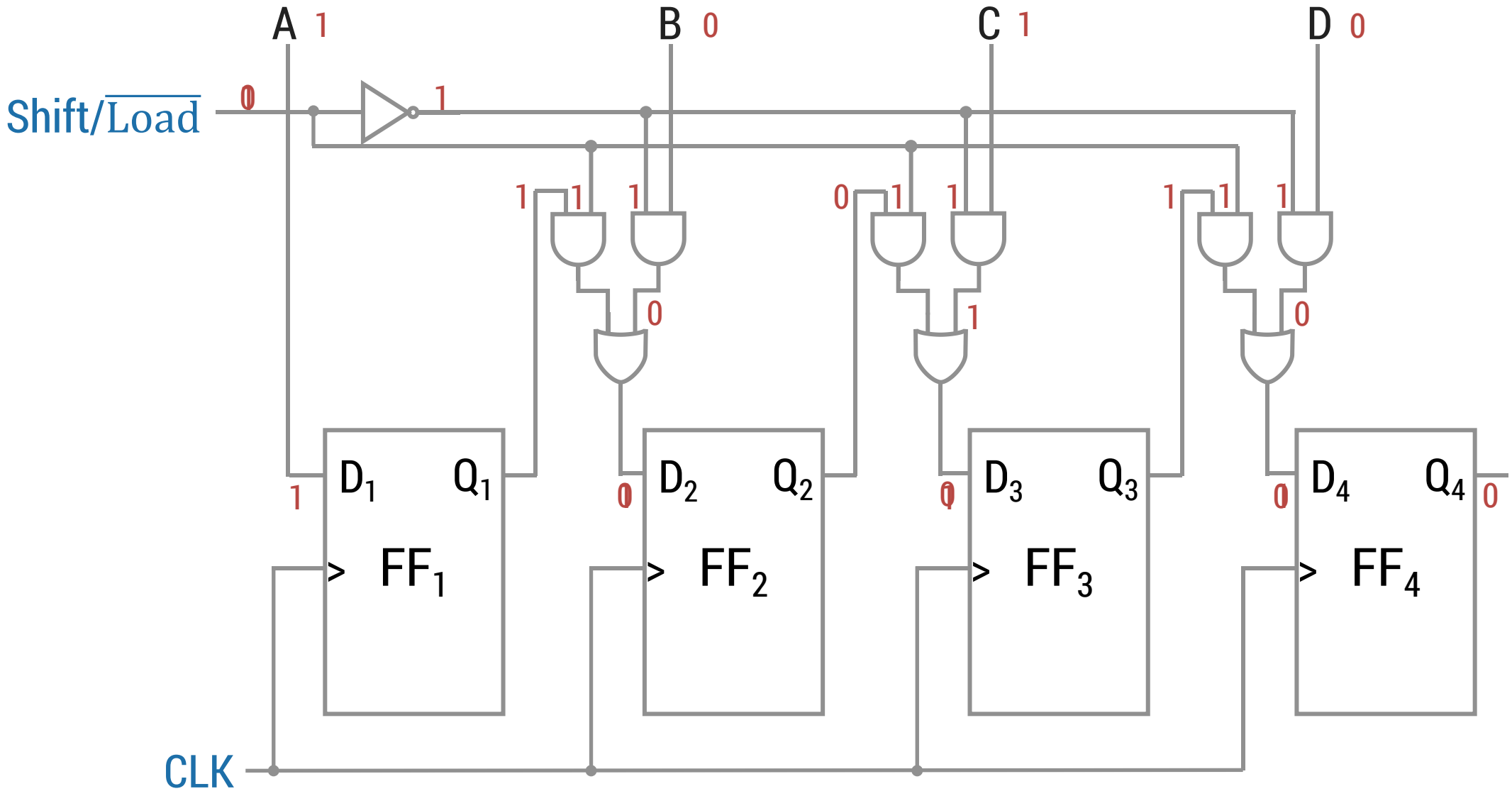
- It is called Buffer register.
- In one clock pulse all bits are stored simultaneously . Only one clock pulse is required.

For description refer this site: http://studytronics.weebly.com/shift-registers.html

Parallel-in, Parallel-out, Shift register(PIPO)

In this mode, the 4 bit binary input B0, B1, B2, B3is applied to the data inputs D0, D1, D2, D3respectively of the four flip-flops. As soon as a negative clock edge is applied, the input binary bits will be loaded into the flip-flops simultaneously. The loaded bits will appear simultaneously to the output side. Only clock pulse is essential to load all the bits.

Parallel-in, Serial-out, Shift register



Parallel-in, Serial-out, Shift register

- The input data will enter in parallel that means at a time to all flip flop and output will get serially.

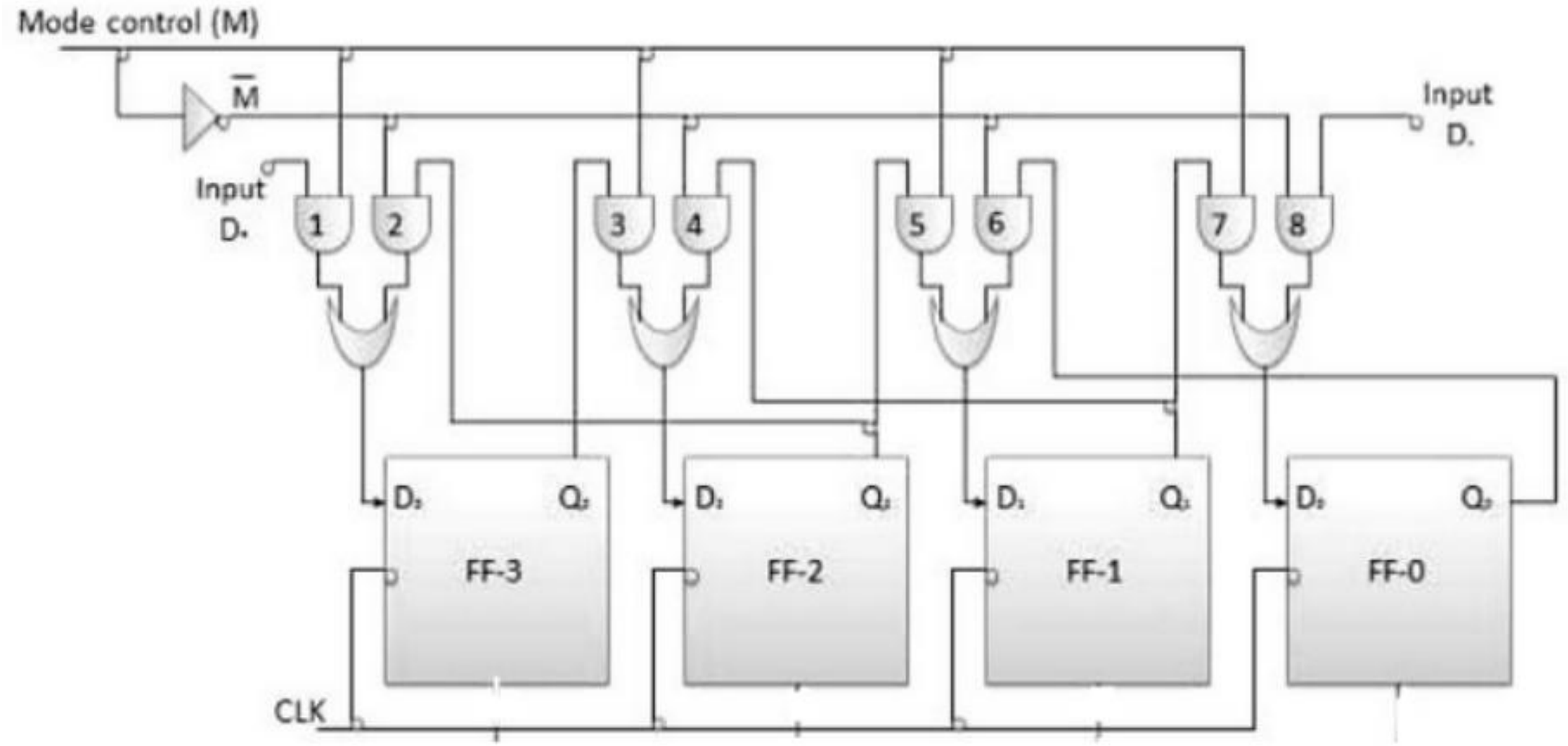
- Then all input are feed the inputs of different 4 number of flip flop. With single clock pulse all data are enter to all 4 **flip flops.**

- 4 bit parallel in serial out shift register, A, B, C, and D are the four parallel data input lines and SHIFT / LOAD (SH / LD) is a **control input** that allows the four bits of data at A, B, C, and D inputs to enter into the register in parallel or shift the data in serial.
- if SHIFT / LOAD = 0 then data will be stored and
  if SHIFT / LOAD = 1 then data will be shifted.
-

4 bit Bidirectional shift register

- **It is the registers which are capable of shifting the data either right or left depending on the mode selected.**
- If the **mode = 1**, the data will be shifted towards the right direction
- if the **mode = 0**, the data will be shifted towards the left direction.
- 4- D flip-flops which are connected.
- The input data is connected at two ends of the circuit and depending on the mode selected only one and gate is in the active state.

4 bit Bidirectional shift register



Application of Register

- Temporary data storage i.e. Microprocessor.
- Data transfer and data manipulation .i.e. multiplication and division by 2.
- Produce time delay to digital circuits.
- Used in communication lines where demultiplexing of a data line into several parallel line is required.
- Data Converter.
- Ring counter.
- Johnson or Twisted ring counter

https://learn.circuitverse.org/docs/seq-msi/registers.html#serial-in-serial-out

Counters

Introduction

- A counter – a group of flip-flops connected together to perform counting operations (flip-flops are used to construct counters)

- Two categories:
  1) Asynchronous counter
  2) Synchronous counter

- Classification of counters
  - UP counter
  - DOWN counter
  - UP/DOWN counter

https://learn.circuitverse.org/docs/seq-msi/counters.html

Introduction- Counters

- Primary purpose is to produce a **specified output pattern sequence**.
- The total number of **states** is called its **modulus.**
- *An n-bit counter that counts through all its natural states and does not skip any of the states has a modulus of 2^n.*

- If a counter has **m distinct states** then it is called a **mod-m counter.**
-

- n bit counter has n flip flops and it has 2^n distinct states with maximum count of $2^n - 1$.

Asynchronous Counters v/s Synchronous Counters

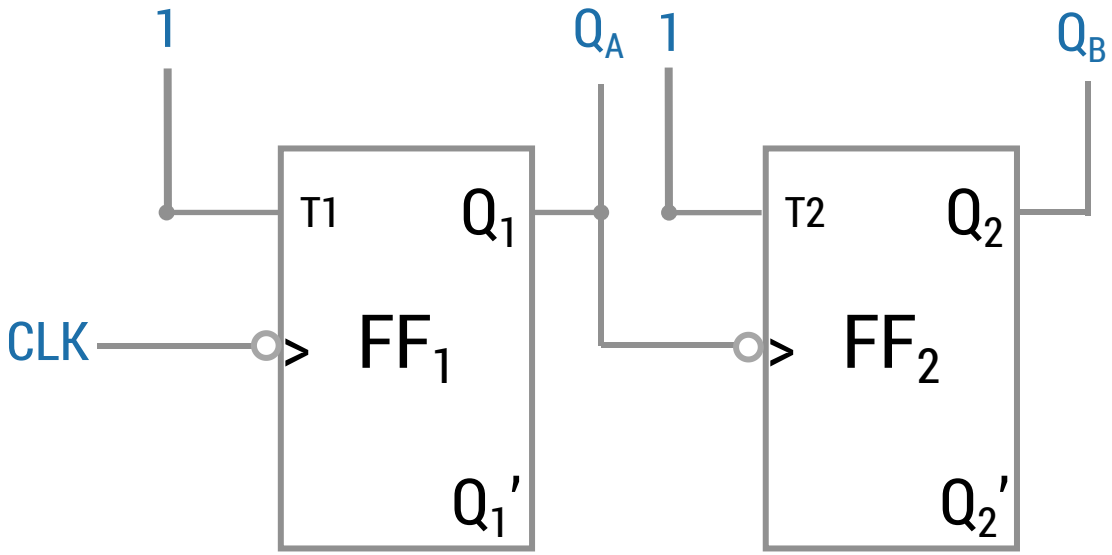
Asynchronous/Ripple/serial Counters	Synchronous/Ring/Johnson/parallel Counters
FFs are connected in such a way that the output of the first FF connected to the clock for the second FF, the output of the second the clock of the third and so on.	There is no connection between the output of first FF and clock input of next FF and so on.
All the FFs are not clocked simultaneously.	All the FFs are clocked simultaneously.
Design and implementation is very simple even for more number of states.	Design and implementation becomes tedious and complex as the number of states increases.
Main drawback of these counters is their low speed as the clock is propagated through a number of FFs before it reaches the last FF.	Since clock is applied to all the FFs simultaneously the total propagation delay is equal to the propagation delay of only one FF. Hence they are faster.
the delay is dependent of the size of the counter.	the delay is independent of the size of the counter.

1.Asynchronous Counters

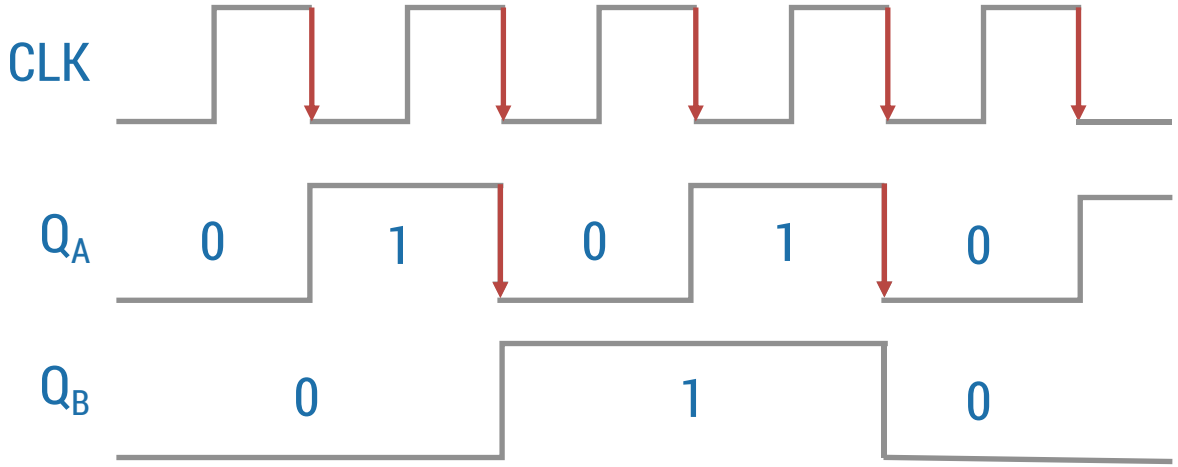
- Group of flip-flops, **count** a stream of **pulses applied** to the counter's Clock input.
- The output is a binary value whose value is equal to the number of pulses received at the CK input.

- Mod 4 counter →4 states(0,1,2,3) → 2 bit counter → 2 FF are required.
- Mod 8 counter → 8 states(0,1,2,3,4,5,6,7) → 3 bit counter → 3 FF are required.
- Mod 10 counter(BCD) → 10 states(0,1,2,3,4,5,6,7,8,9) → 4 bit counter → 4 FF are required.(https://www.youtube.com/watch?v=EVb1cn4QOGM)
- Mod N counter→ N states(0,1,2,3,4……..N-1) → n bit counter → n FF are required.

2-bit Ripple Up-Counter- T FF(MOD-4) Negative Edge-triggered



CLK	Present State		Next State	
	Q_B	Q_A	Q_B	Q_A



http://falstad.com/circuit/e-counter.html

2-bit Ripple Up-Counter- T FF(MOD-4) Negative Edge-triggered

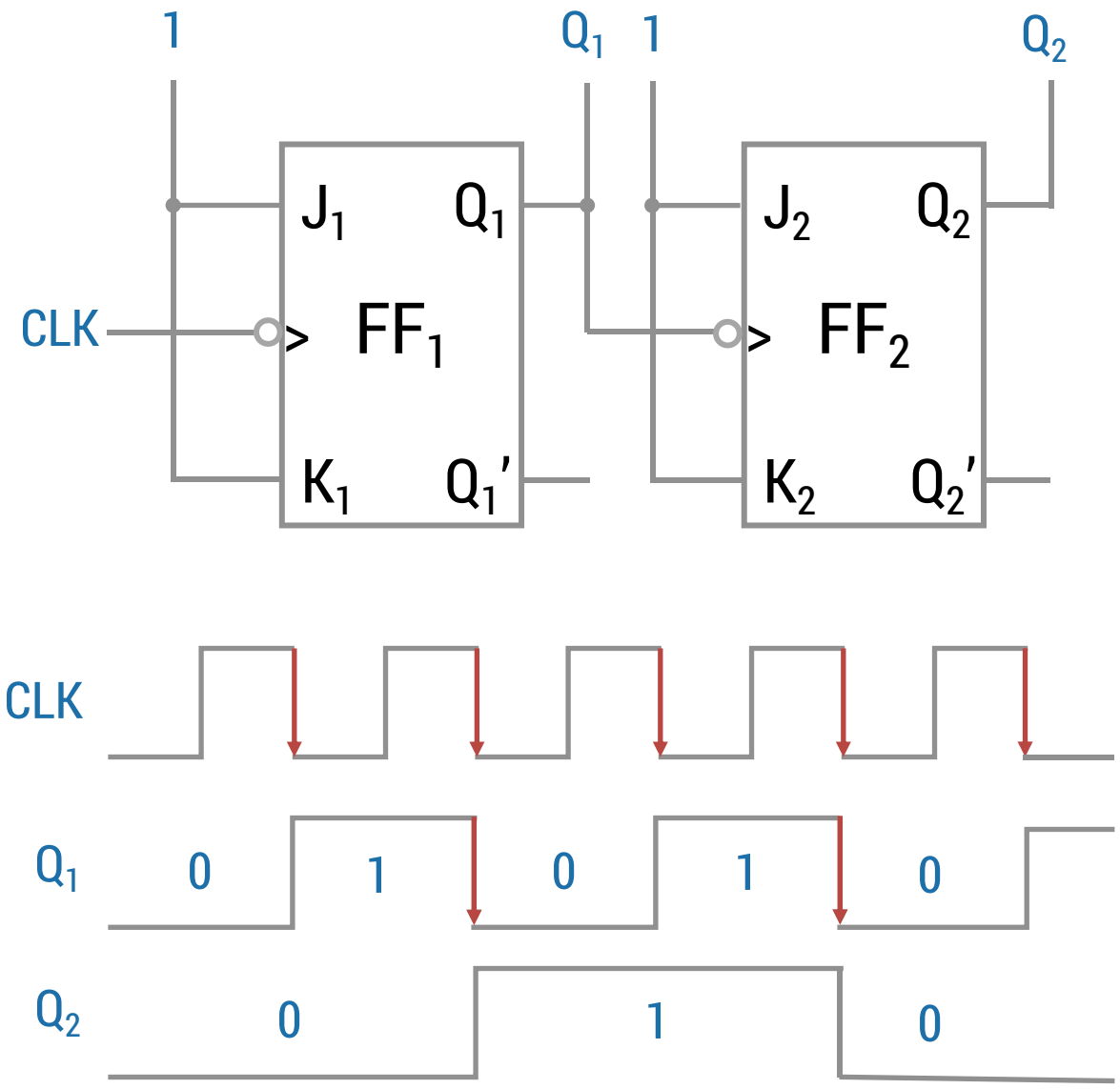
Operation

S.N.	Condition	Operation
1	**Initially let both the FFs be in the reset state**	Q_BQ_A = 00 initially
2	**After 1st negative clock edge**	As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will be equal to 1. Q_A is connected to clock input of FF-B. Since Q_A has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in Q_B because FF-B is a negative edge triggered FF. Q_BQ_A = 01 after the first clock pulse.
3	**After 2nd negative clock edge**	On the arrival of second negative clock edge, FF-A toggles again and Q_A = 0. The change in Q_A acts as a negative clock edge for FF-B. So it will also toggle, and Q_B will be 1. Q_BQ_A = 10 after the second clock pulse.
4	**After 3rd negative clock edge**	On the arrival of 3rd negative clock edge, FF-A toggles again and Q_A become 1 from 0. Since this is a positive going change, FF-B does not respond to it and remains inactive. So Q_B does not change and continues to be equal to 1. Q_BQ_A = 11 after the third clock pulse.

2-bit Ripple Up-Counter- T FF(MOD-4) Negative Edge-triggered

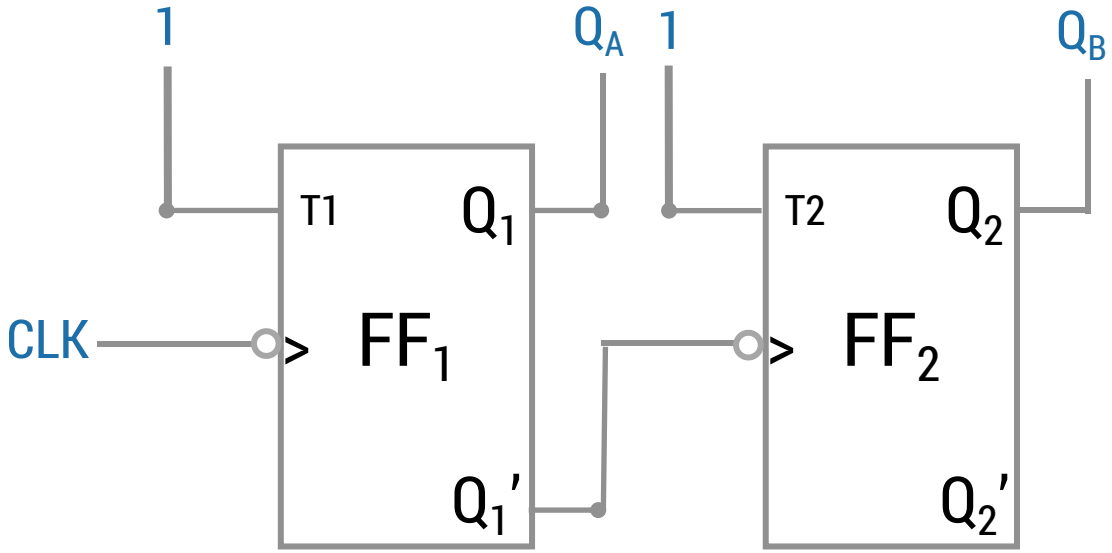
5	**After 4th negative clock edge**	On the arrival of 4th negative clock edge, FF-A toggles again and Q_A becomes 1 from 0. This negative change in Q_A acts as clock pulse for FF-B. Hence it toggles to change Q_B from 1 to 0. Q_BQ_A = 00 after the fourth clock pulse.

2-bit Ripple Up-Counter using Negative Edge-triggered Flip-Flop

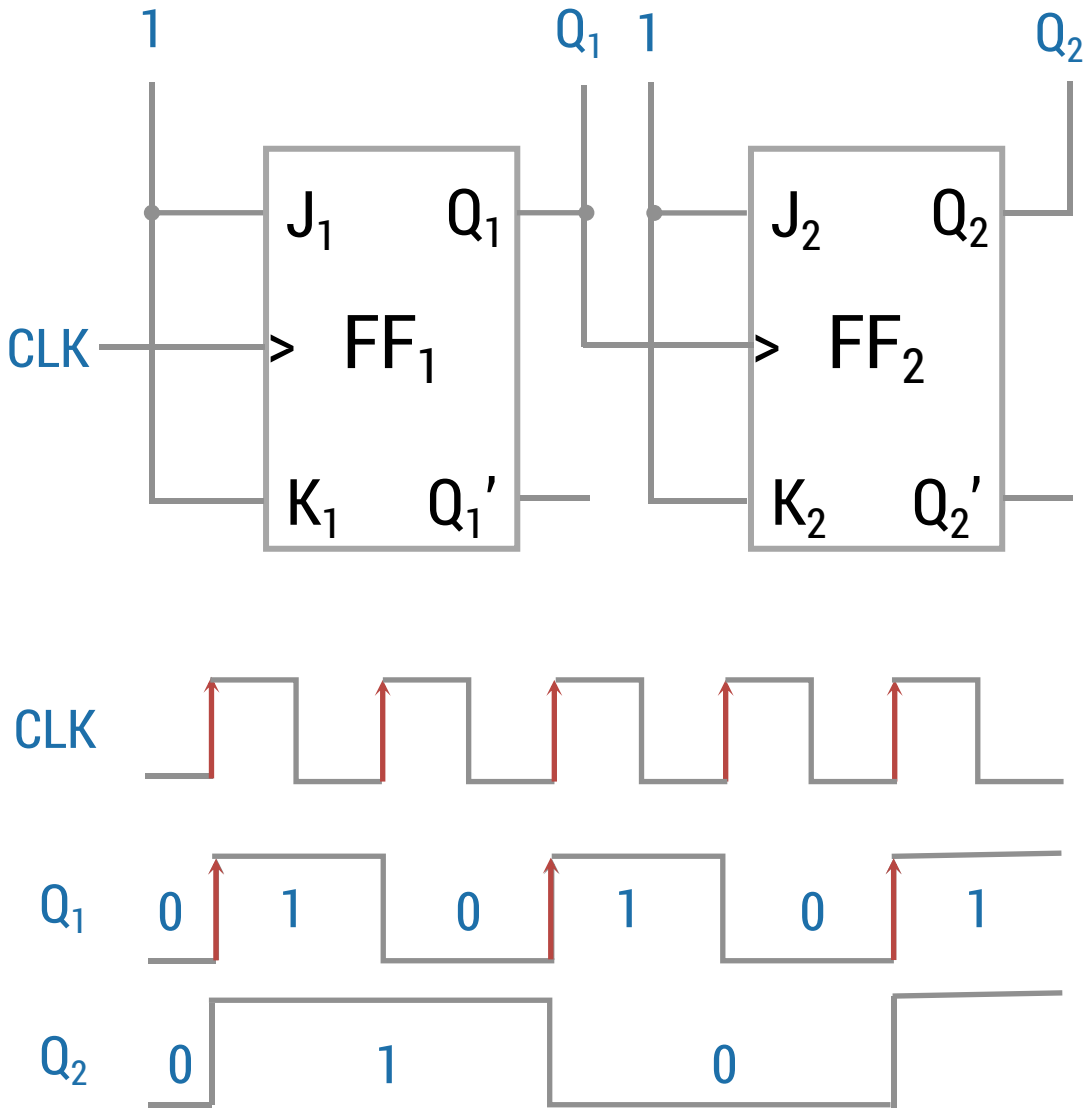


CLK	Present State		Next State	
	Q_2	Q_1	Q_2	Q_1

2-bit Ripple down-Counter- T FF(MOD-4) Negative Edge-triggered

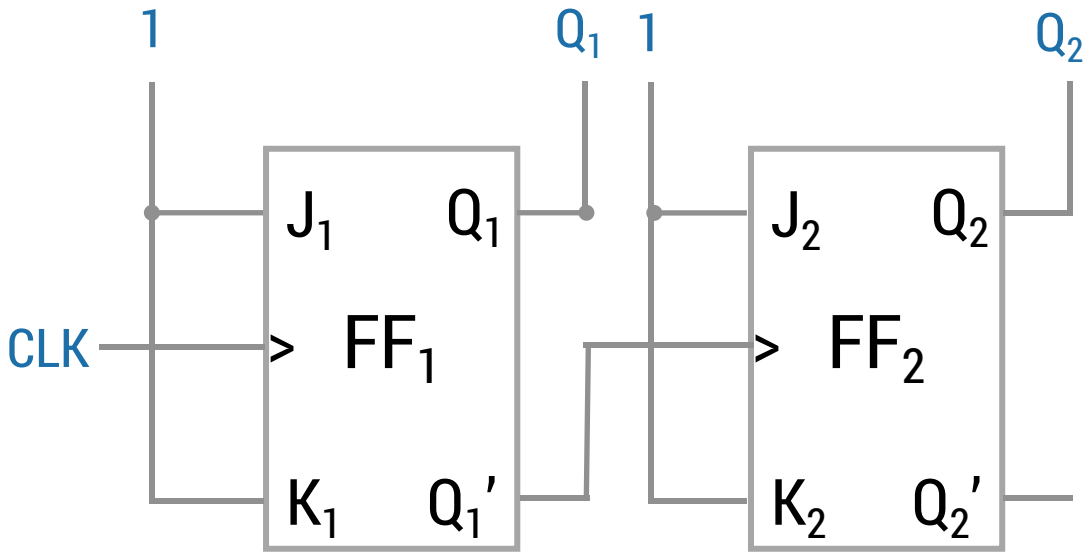


2-bit Ripple Down-Counter using Positive Edge-triggered Flip-Flop



CLK	Present State		Next State	
	Q_2	Q_1	Q_2	Q_1

2-bit Ripple up-Counter using Positive Edge-triggered Flip-Flop



Refer this video link to understand 3 bit and 4 bit negative edge up-counter.

1. https://www.youtube.com/watch?v=ucCtDhYFCJs
2. https://www.youtube.com/watch?v=s1DSZEaCX_g
3. https://www.youtube.com/watch?v=eEeBh8jfDjg

- Design 4-bit Ripple up -Counter using JK FF. Write its TT and Draw Timing diagram.
- Refer Anil K.Maini page no 431

- ripple counter type number 74293 IC.→ four-bit binary ripple counter

Homework

1. Design 3-bit Ripple up -Counter using T/JK FF. Write its TT and Draw Timing diagram.(http://www.prajval.in/edudetail/284/2413/%3Cp%3E%3Cstrong%3EDesign-3-bit-ripple-up-counter-using-negative-edge-triggered-JK-flip-flops-Also-draw-the-waveforms%3C-strong%3E%3C-p%3E-).

1. Design-3-bit-ripple-up-counter-using-negative-edge-triggered-JK-flip-flops-Also-draw-the-waveforms.
2. Design 3-bit Ripple down -Counter using T/JK FF. Write its TT and Draw Timing diagram.
3. Design 4-bit Ripple down -Counter using T/JK FF. Write its TT and Draw Timing diagram.

Up/Down counter

2 bit up/down ripple counter

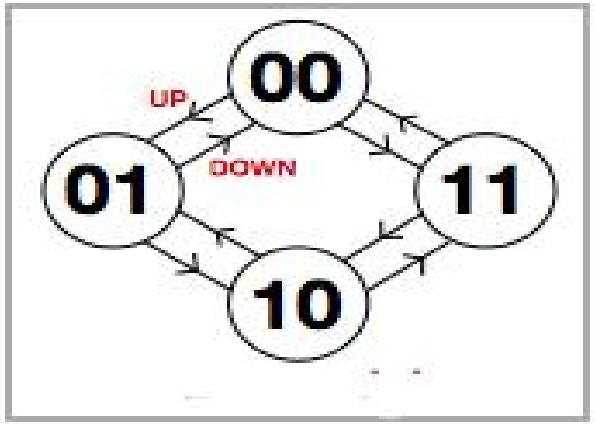
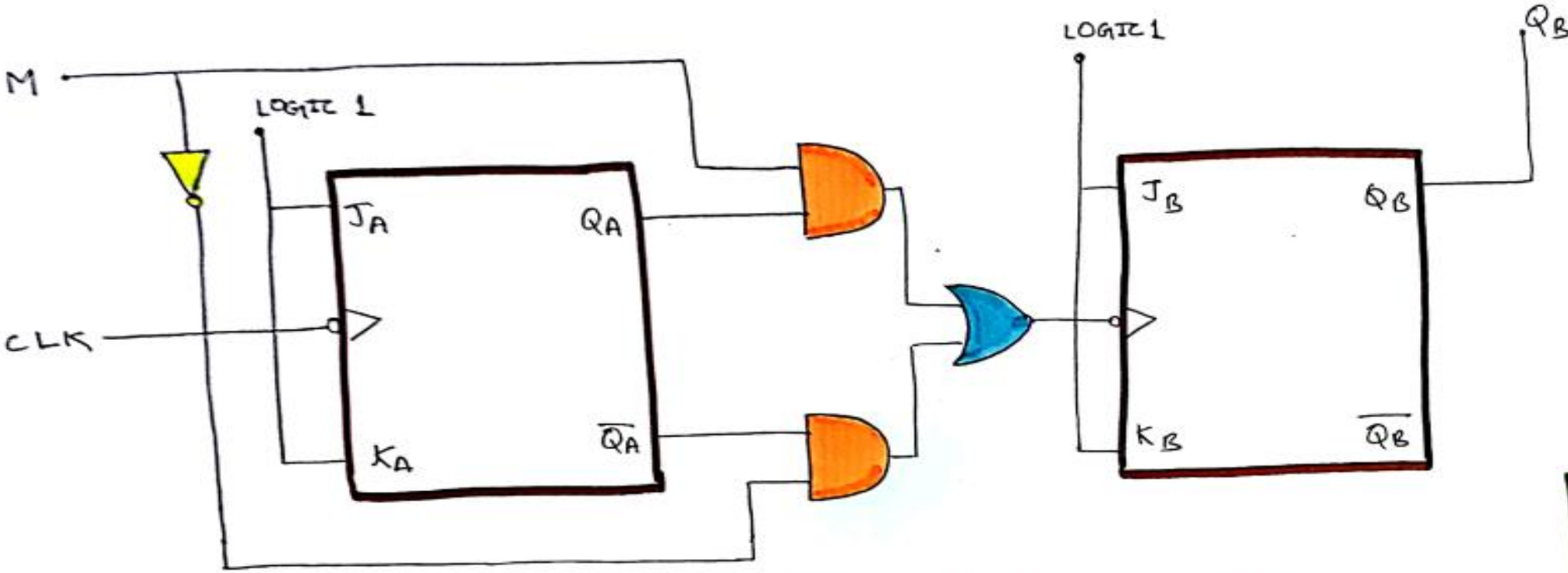


Illustration states 2-bit UP/DOWN counter.



2-bit ripple up/down counter

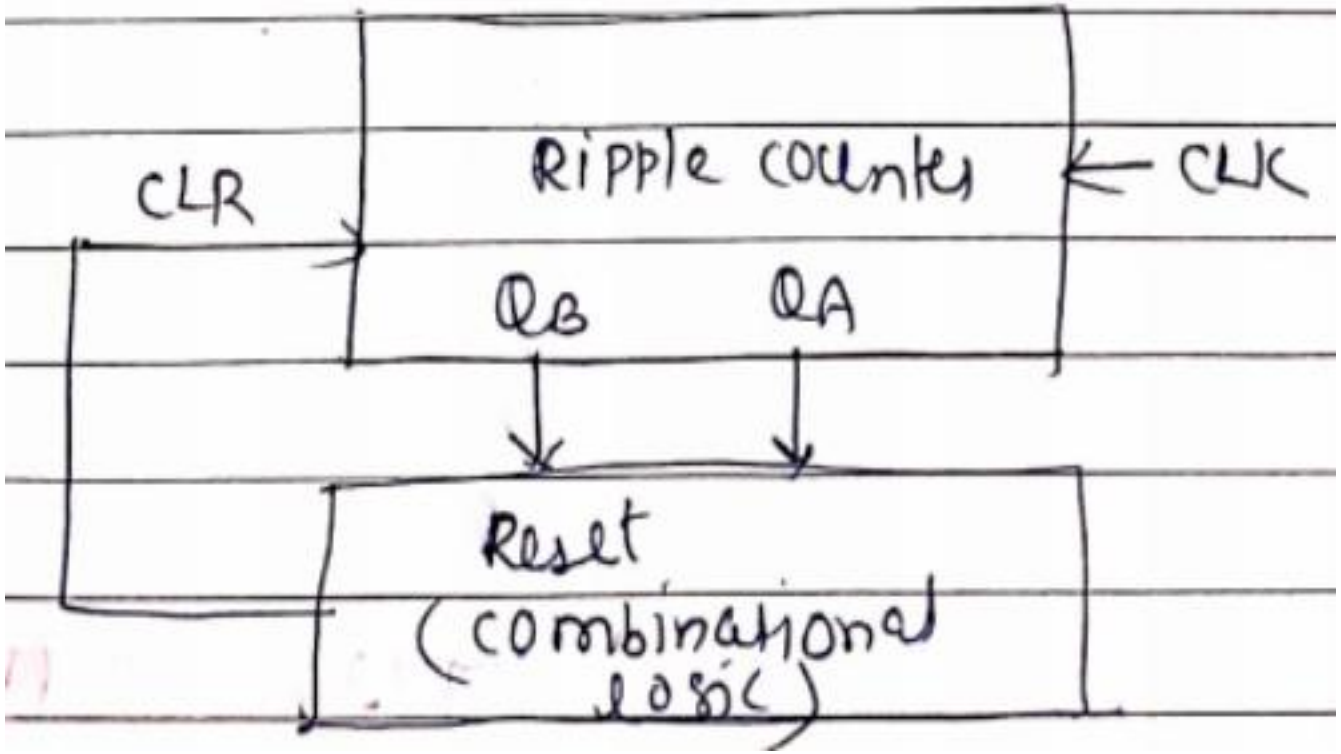
M=1 UP COUNTER

M=0 DOWN COUNTER

Source: https://player.uacdn.net/lesson-raw/CTESRQVKPN4ZGUOGTTJX/pdf/7718910058.pdf

Modulus Ripple Counter

- Mod 4 counter → 2 bit counter → 2 FF are required.
- Mod 8 counter → 3 bit counter → 3 FF are required.
- Mod N counter→ n bit counter → n FF are required.
- Modulus counter= Ripple Counter+ Reset Logic(Combinational logic)



https://www.youtube.com/watch?v=5mfN5KdjcQw

Asynchronous Counters

Design Steps:

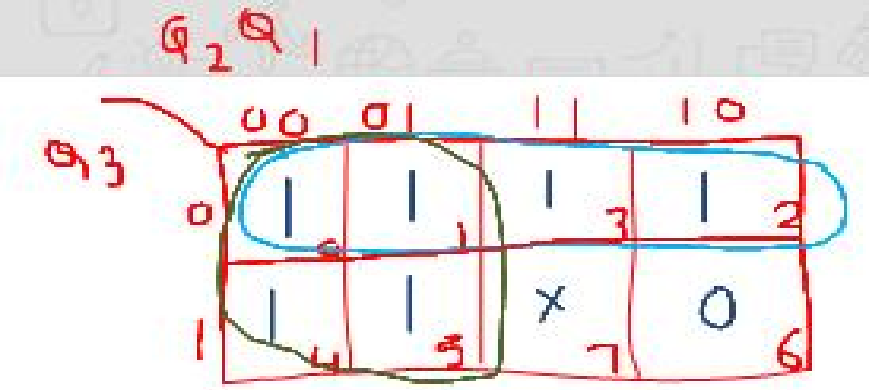
1. Determine number of flip-flop.
2. Choose the type of flip-flop.
3. Write the truth table of the counter.
4. Make the Kmap and derive the boolean equation.
5. Design counter.

Mod-6 Asynchronous/Ripple Counter

1. Mod -6 , 6 states → 0,1,2,3,4,5
2. Maximum no is 5 ,so 3 bits are required.
3. No. of FF=3
4. Types of flip-flop T.
5. Make a truth table.
6. Boolean equations.
7. Design final counter circuit.

Mod-6 Asynchronous/Ripple Counter

After pulses	State			Reset(R)
	Q_3	Q_2	Q_1	
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	**0**



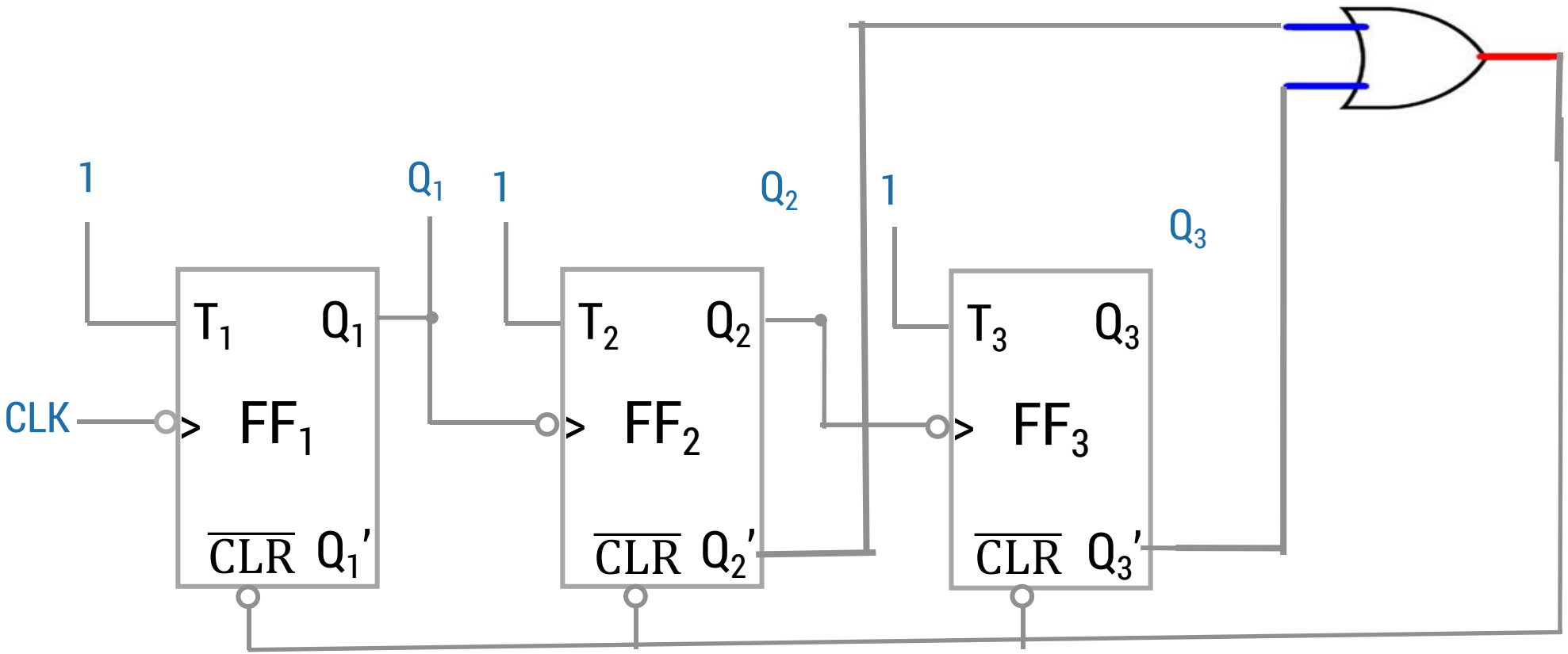
R = 1 for 000 to 101

R = 0 for 110

R = x for 111

$R = Q_3' + Q_2'$

Mod-6 Asynchronous Counter



Decade and BCD Counters

- that goes through 10 unique output combinations and then resets as the clock proceeds further.
- MOD-10 counter.
- 4 flip-flops.
- From 16 states any of the six states by using some kind of feedback is skipped.
- A decade counter does not necessarily count from 0000 to 1001.
- It could even count as 0000, 0001, 0010, 0101, 0110, 1001, 1010, 1100, 1101, 1111, 0000.
- https://upload.wikimedia.org/wikipedia/commons/3/33/DecadeCounter.jpg
- https://www.electronicsforu.com/technology-trends/learn-electronics/decade-counter-circuit-basics
- *BCD counter* is a special case of a decade counter.
- Counts from 0000 to 1001 and then resets.
- https://everycircuit.com/circuit/5043097612320768/decade-counter
- https://www.elprocus.com/bcd-counter-circuit-working/

Synchronous Counters

Section - 4

http://falstad.com/circuit/e-synccounter.html

Design of Synchronous Counters

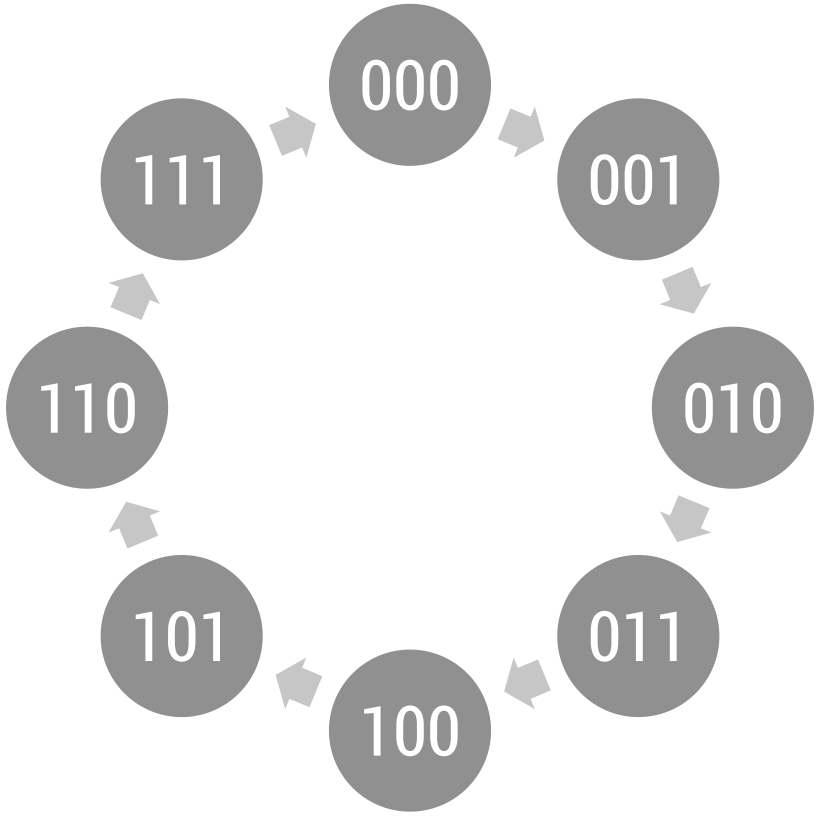
- **Step 1 - Number of flip-flops**: Based on the description of the problem, determine the required number n of the FFs - the smallest value of n is such that the number of states $N \leq 2^n$ and the desired counting sequence.
- **Step 2 - State diagram**: Draw the state diagram showing all the possible states.
- **Step 3 - Choice of flip-flops and excitation table:** Select the type of flip-flops to be used and write the excitation table.
- **Step 4 - Minimal expressions for excitations:** K-maps for the excitations of the flip-flops in terms of the present states and inputs.
- **Step 5 - Logic Diagram**: Draw the logic diagram based on the minimal expressions.

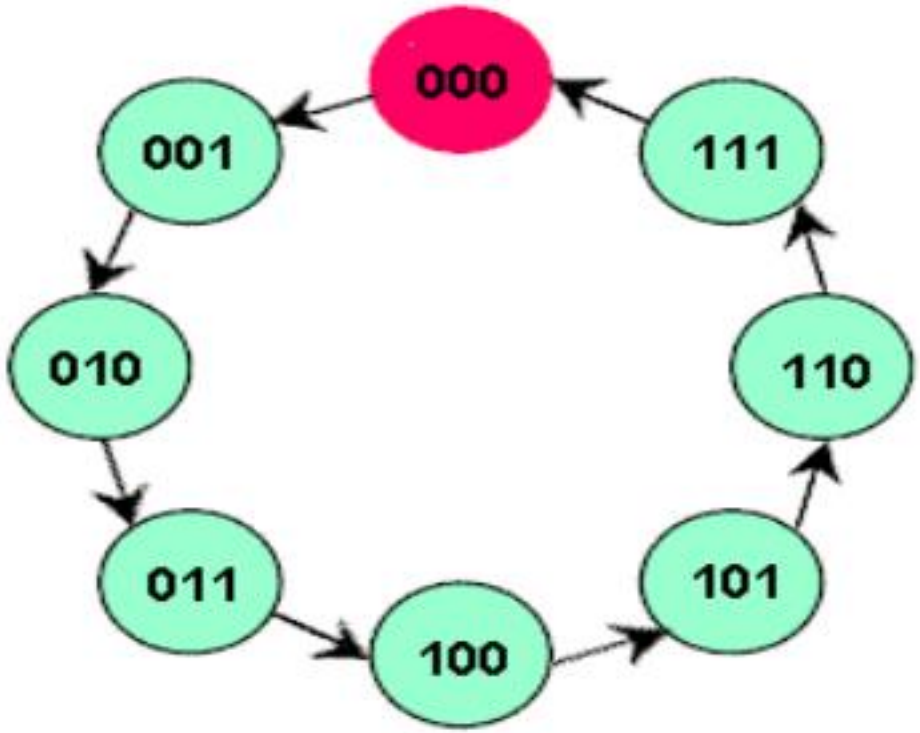
Design of Synchronous 3-bit Up Counters

- Step 1 - Number of flip-flops:

  A 3-bit up-counter requires 3 flip-flops. The counting sequence is 000, 001, 010, 011, 100, 101, 110, 111, 000 …

- Step 2 - Draw the state diagram:





0

Clk

Design of Synchronous 3-bit Up Counters

- Step 3 - Select the type of flip-flops and draw the excitation table:

  JK flip-flops are selected and the excitation table of a 3-bit up-counter using J-K flip-flops is drawn as shown below.

PS			NS			Required excitations					
Q_3	Q_2	Q_1	Q_3	Q_2	Q_1	J_3	K_3	J_2	K_2	J_1	K_1
0	0	0	0	0	1						
0	0	1	0	1	0						
0	1	0	0	1	1						
0	1	1	1	0	0						
1	0	0	1	0	1						
1	0	1	1	1	0						
1	1	0	1	1	1						
1	1	1	0	0	0						

J-K FF Excitation Table

PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0

Design of Synchronous 3-bit Up Counters

- Step 4 - Obtain the minimal expressions:

From excitation table, $J_1 = K_1 = 1$.

K – Maps for excitations J_3, K_3, J_2 and K_2 and their minimized form are as follows:

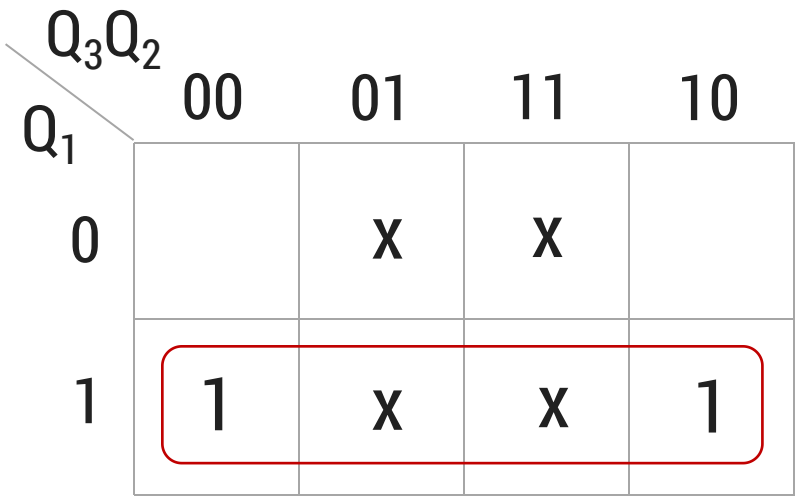
Q_1 \ Q_3Q_2	00	01	11	10
0			x	x
1		1	x	x

$J_3 = Q_2Q_1$

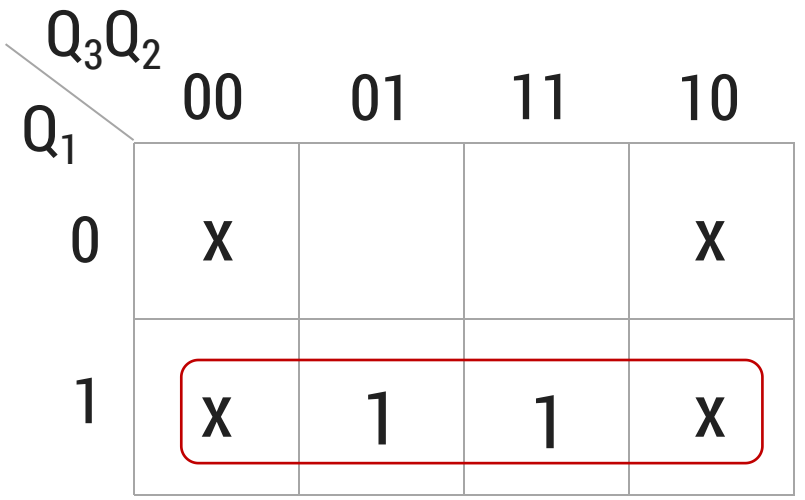
Q_1 \ Q_3Q_2	00	01	11	10
0	x	x		
1	x	x	1	

$K_3 = Q_2Q_1$

Design of Synchronous 3-bit Up Counters



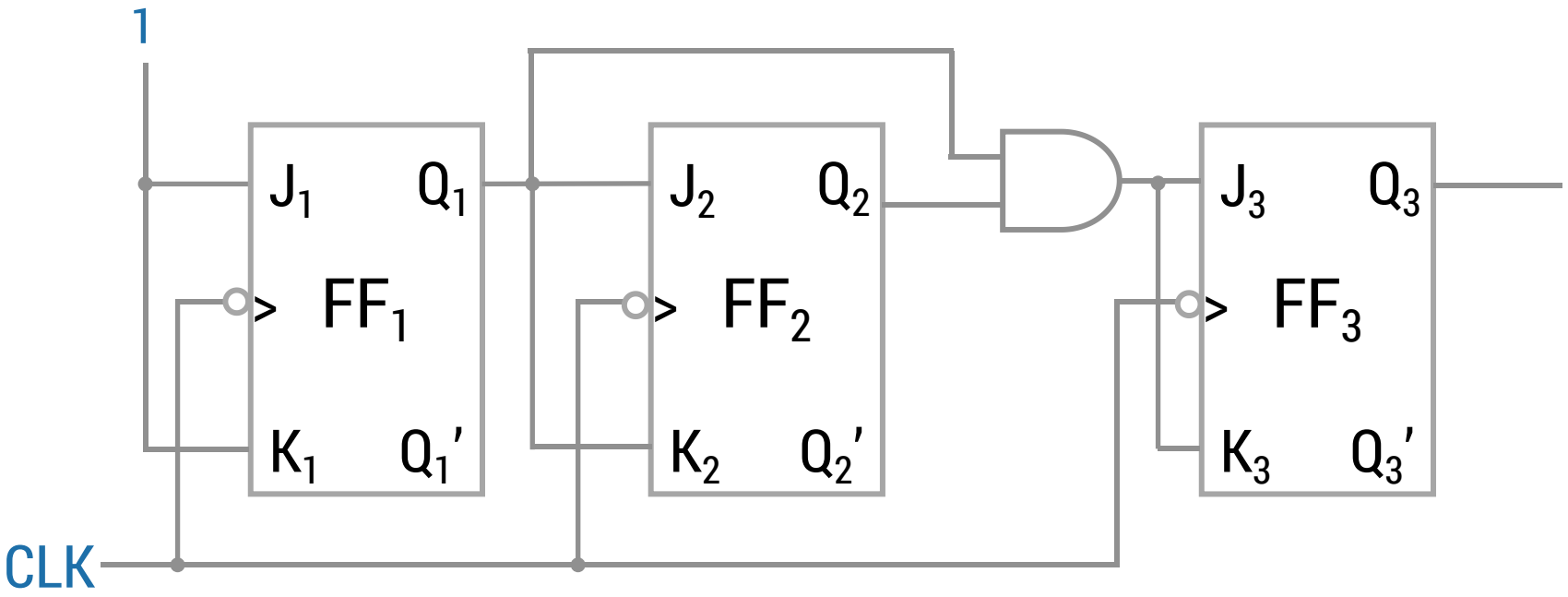
$J_2 = Q_1$



$K_2 = Q_1$

Design of Synchronous 3-bit Up Counters

- Step 5 - Draw the logic diagram:



Designing Counters with Arbitrary Sequences

ALL BELOW LINK WILL BE USEFUL TO UNDERSTAND THE TOPIC SO GO THROUGH ALL VIDEOS.

- https://www.youtube.com/watch?v=Zce6NlHuvfs
- https://www.youtube.com/watch?v=vx4PNd_Hl8U
- https://www.youtube.com/watch?v=ruxiO77HL9k

Decade and BCD Counters

- that goes through 10 unique output combinations and then resets as the clock proceeds further.
- MOD-10 counter.
- 4 flip-flops.
- From 16 states any of the six states by using some kind of feedback is skipped.
- A decade counter does not necessarily count from 0000 to 1001.
- It could even count as 0000, 0001, 0010, 0101, 0110, 1001, 1010, 1100, 1101, 1111, 0000.
- *BCD counter* is a special case of a decade counter.
- Counts from 0000 to 1001 and then resets.

State Table for BCD Counter

Present State				Next State				Output	Flip-Flop inputs			
Q8	Q4	Q2	Q1	Q8	Q4	Q2	Q1	Y	TQ8	TQ4	TQ2	TQ1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	0	0	0	1

- The flip flop input equations can be simplified by means of maps. The simplified functions are
  - $T_{Q1}=1$
  - $T_{Q2}=Q_8'Q_1$
  - $T_{Q4}=Q_2Q_1$
  - $T_{Q8}=Q_8Q_1+Q_4Q_2Q_1$
  - $y=Q_8Q_1$
- The circuit can be easily drawn with four T flip-flops, five AND gates, and one OR gate.

BCD COUNTER

- DRAW YOUR LOGIC DIAGRAM HERE FROM FOUND BOOLEN EQUATION FROM PREVIOUS SLIDE.

Ring Counter- Shift Register Counters

- **Ring counter is an application of Shift resister.**

- **Types of Ring Counter –**

  1. **Straight Ring Counter** 2. **Twisted Ring Counter**

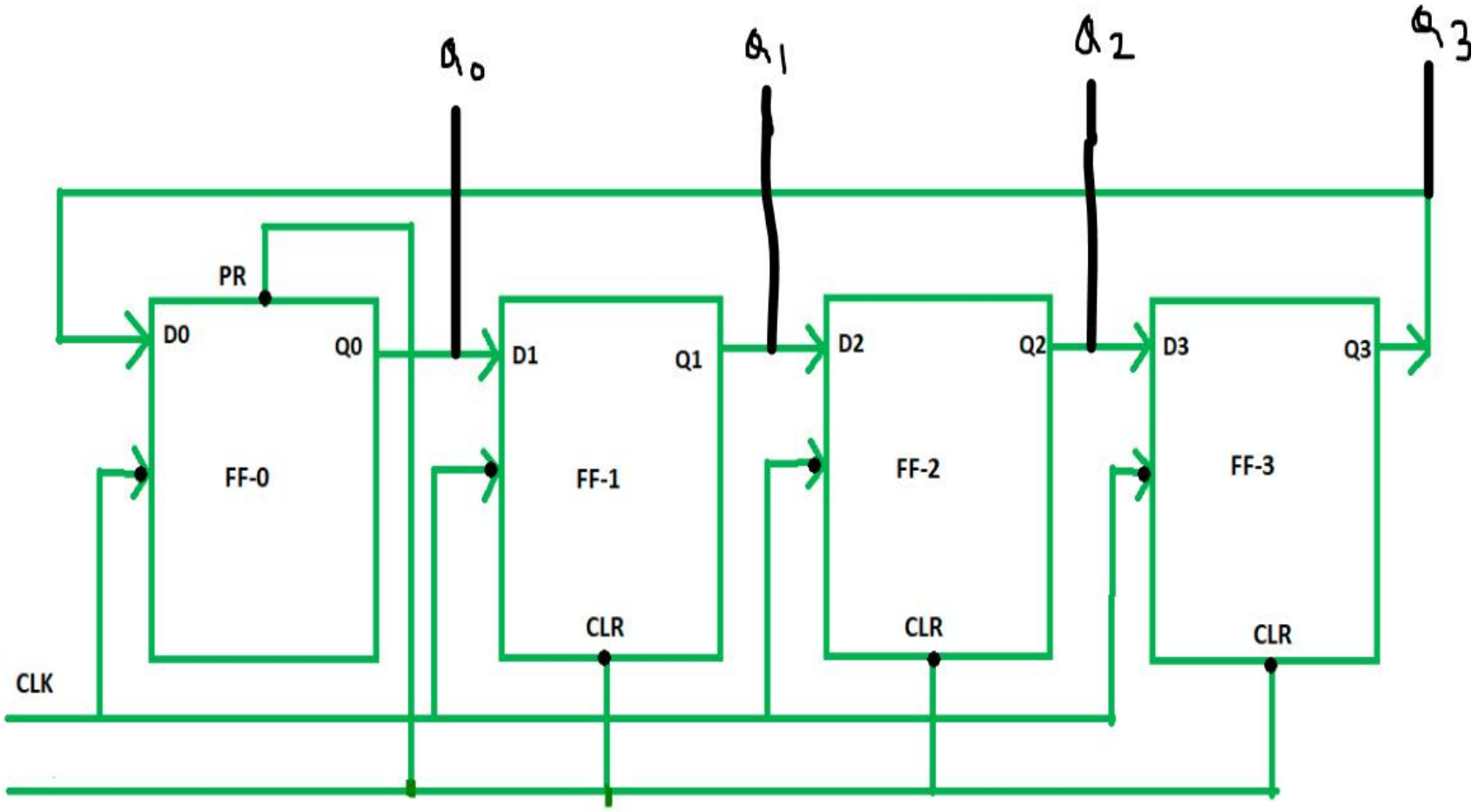
4-bit Ring counter with 4 D flip-flop.

- Straight Ring Counter: One hot Counter : Last flip-flop is connected to the input of the first flip-flop.

- Clock pulse (CLK) is applied to all the flip-flop simultaneously. Therefore, it is a Synchronous Counter.

- No of state = No of bits.

4-bit Ring counter with 4 D flip-flop.

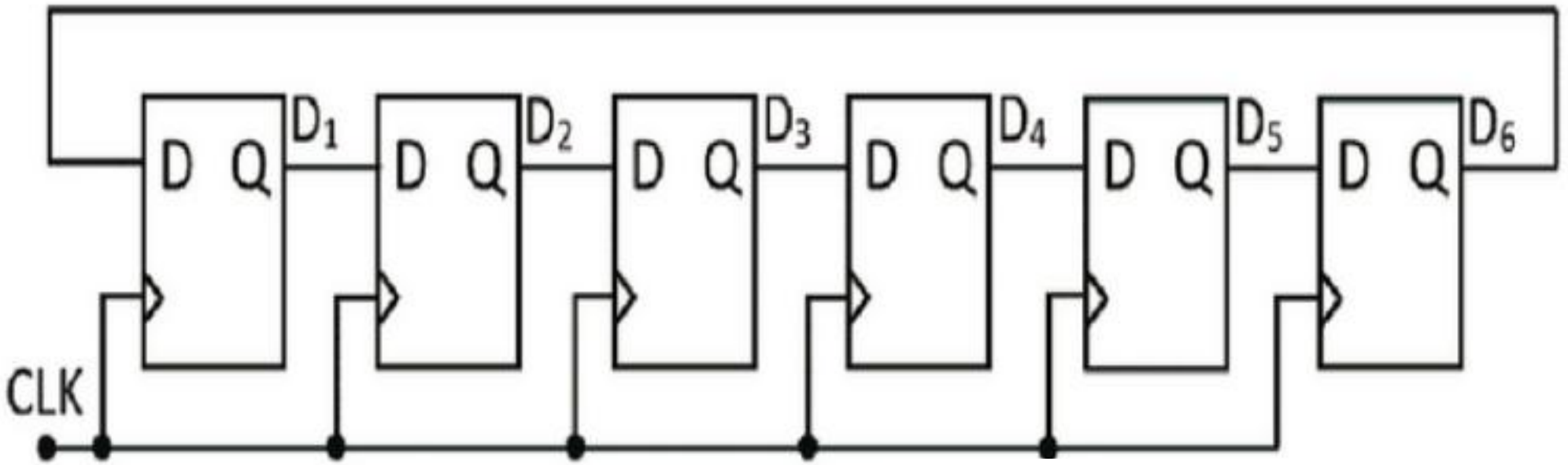


Ring Counter

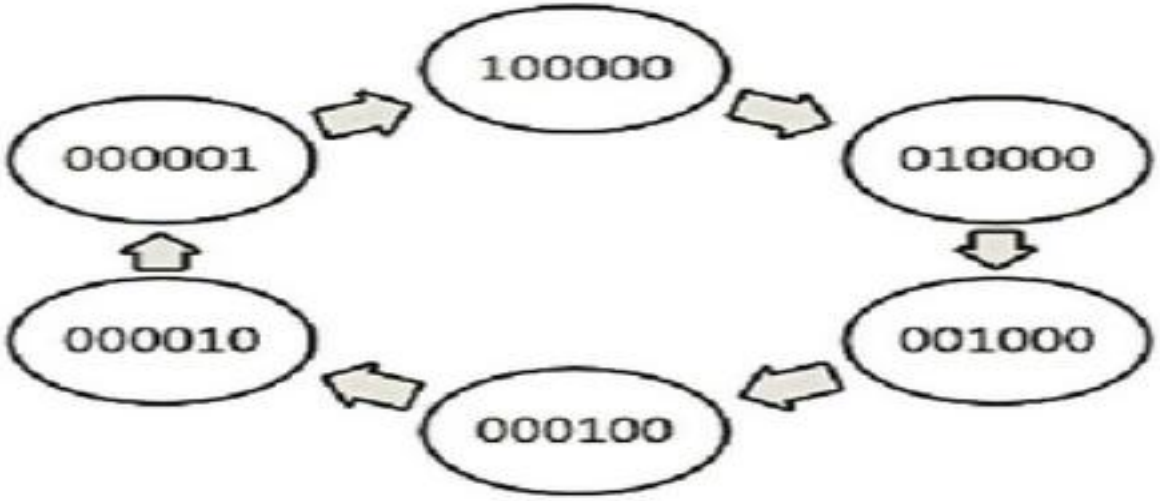
Straight ring counter				
State	Q0	Q1	Q2	Q3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1

To write about ring counter refer below link
https://www.geeksforgeeks.org/ring-counter-in-digital-logic/?ref=lbp

6-bit Ring counter with 6 D flip-flop.



CLK	D_1	D_2	D_3	D_4	D_5	D_6
1	1	0	0	0	0	0
2	0	1	0	0	0	0
3	0	0	1	0	0	0
4	0	0	0	1	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

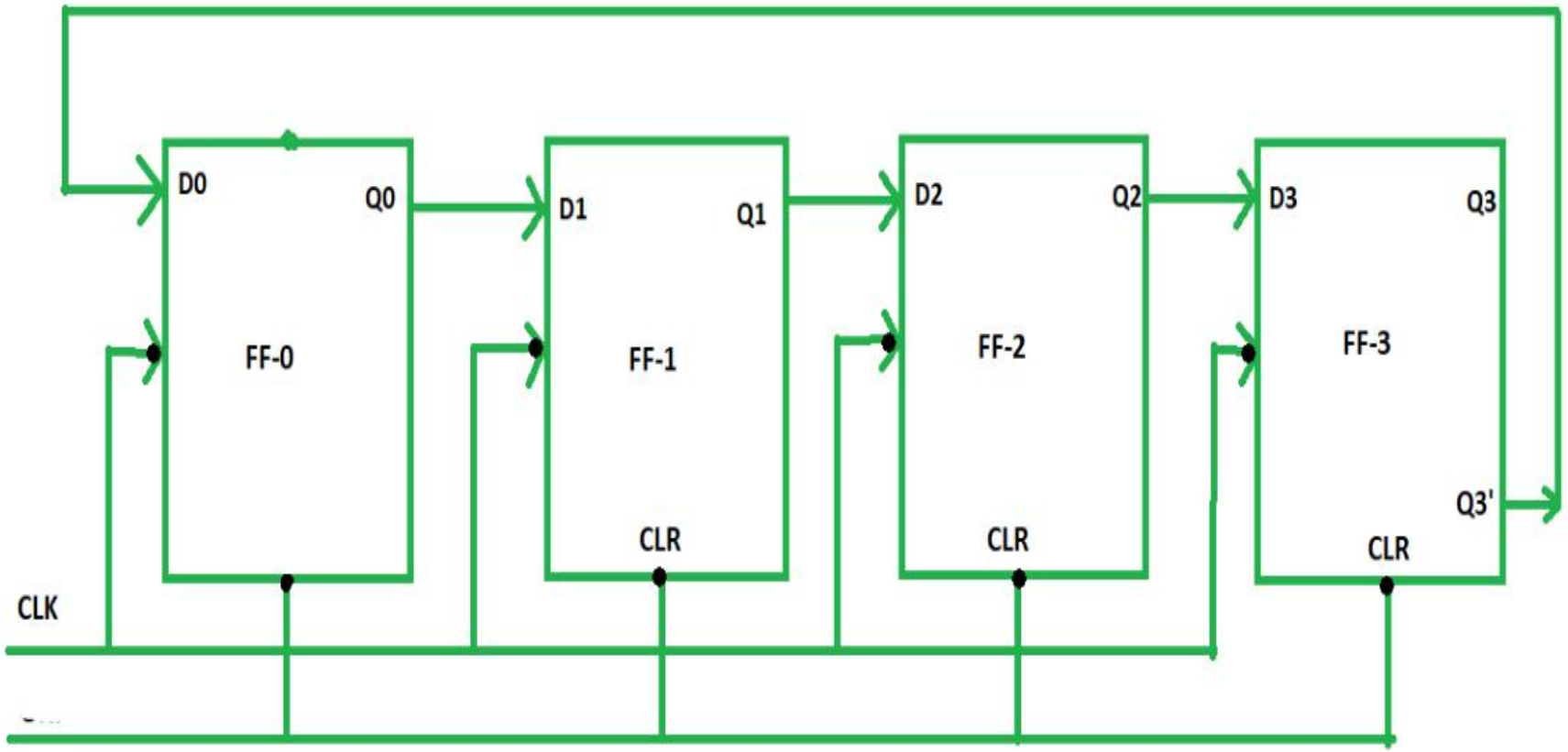


https://www.researchgate.net/figure/State-diagram-and-implementation-of-a-six-bit-ring-counter-with-D-Flip-flops_fig2_322106955

4 bit Twisted Ring Counter

- **Twisted Ring Counter : switch-tail ring counter, walking ring counter or Johnson counter.:**
- connects the **complement** of the output of the last FF to the input of the first FF.
- **No state = 2* No of bits.**

4 bit Twisted Ring Counter



Twisted Ring Counter

Johnson counter				
State	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
0	0	0	0	0

To write about advantage ,dis- advantage ,application and working of this counter refer below link
https://www.geeksforgeeks.org/n-bit-johnson-counter-in-digital-logic/?ref=lbp

Sequential logic circuit

Digital Electronics (DE)

Information and Communication Technology

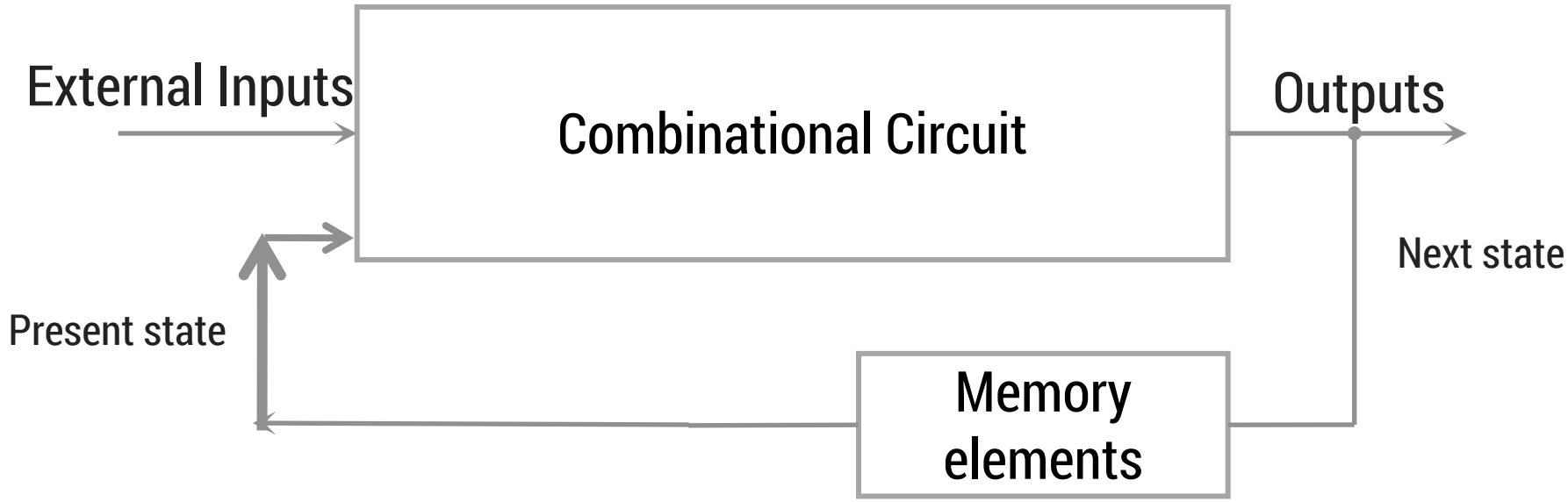
Prepared by Prof. jaydip siyara

Outline

- R-S Flip-Flop D Flip-flops
- Level-Triggered and Edge-Triggered Flip-Flops
- J-K Flip-Flop T Flip-flops
- Synchronous and Asynchronous Inputs
- Applications of FF
- Ripple/Asynchronous Counters
  - Ripple up-down, Modulo or Modulus counter
- Synchronous Counters
  - Up counter, Down counter using different FFs
- Decade and BCD counter
- Presettable Counters
- Decoding a Counter
- Designing Counters with Arbitrary Sequences
- Registers
  - Buffer, Shift, Bidirectional, Universal
  - applications of shift registers,
    - serial to parallel converter, parallel to serial converter
    - ring counter, sequence generator,
- IEEE/ANSI Symbology for Registers and Counters

Sequential Circuits

- Its output levels at any instant of time are dependent on the present inputs at that time and on the state of the circuit, i.e., on the prior input level conditions (i.e. on its past inputs)
- The past history is provided by feedback from the output back to the input.
- Combinational circuits + memory elements = Sequential Circuits.
- Eg. Counters, shift registers, etc.



Sequential Switching Circuits

- **Present state(State input)**: The data stored by memory element at given instant of time.

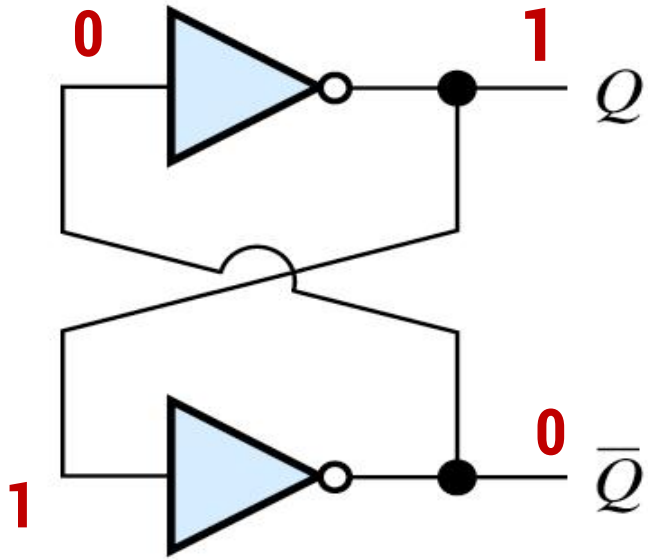
- **Next state:** Combinational circuit have two input 1) External input and 2) present state then new output will be stored in memory called Next state.
- Input: Signal from outside ckt.
- Output: Signals to the outside ckt.

Sequential Circuits v/s Combinational Circuits

Sequential Circuits	Combinational Circuits
The output variables at any instant of time are dependent on the **present input** variables and on the present state, i.e., on the **past history** of the system.	In combinational circuits, the output variables at any instant of time are **dependent only on the present input** variables.
Memory unit is required to store the past history of the input variables in sequential circuits.	Memory unit is **not required** in combinational circuits.
Slower than combinational circuits.	**Faster** because the delay between the input and the output is due to propagation delay of gates only.
Comparatively **harder to design.**	**Easy to design.**
Clock is required	**Clock not required.**
Example:FF,FSM,Registers,Counter	Example: Half- Full adder/Subtractor, Mux-demux Encoder/Decoder,Code converter
Block diagram	Block diagram

1-bit memory and the circuit properties of Bistable latch

- Electronic circuit that has **two stable states** and is capable to store one bit of memory.
- A latch bi-stable multivibrator.
- It can remain in either of the states indefinitely.
- Latch is used for **certain flip-flop** which are *non-clocked.*
- These 'latch on' to a 1 or a 0 immediately upon receiving the input pulse called SET or RESET.
- It has two stable states of complementary output values.
- As an example:

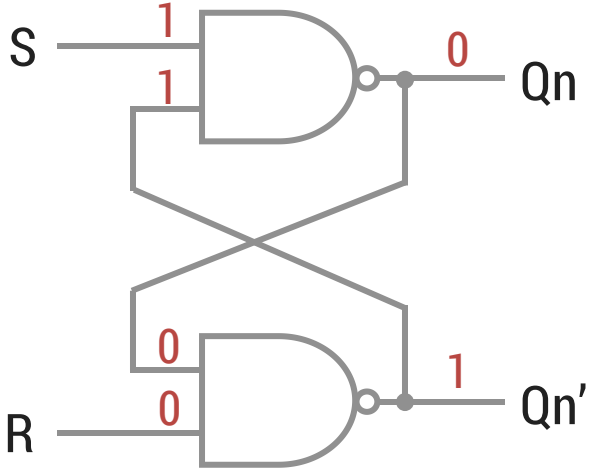


Bistable Latch/ 1 Bit Memory Circuit/S-R Latch

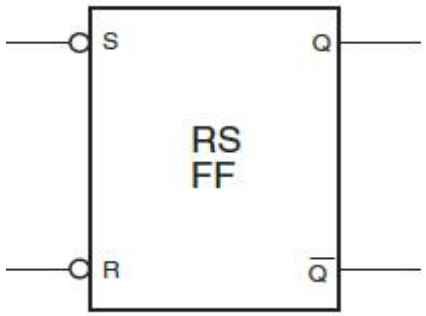
- The simplest type of flip-flop is called an S-R latch.
- It has two outputs labelled Q and Q' and two inputs labelled S and R.
- The state of the latch corresponds to the level of Q (HIGH or LOW, 1 or 0) and Q' is the complement of that state.
- It can be constructed using either **two cross-coupled NAND gates** or **two cross-coupled NOR gates.**
- Using two NOR gates, an active-HIGH S-R latch can be constructed and using two NAND gates an active-LOW S-R latch can be constructed(Cross coupled inverter).
- The name of the latch, S-R or **SET-RESET**, is derived from the names of its inputs.

S-R latch (Active Low) using NAND Gate

A	B	$\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0



Logic diagram

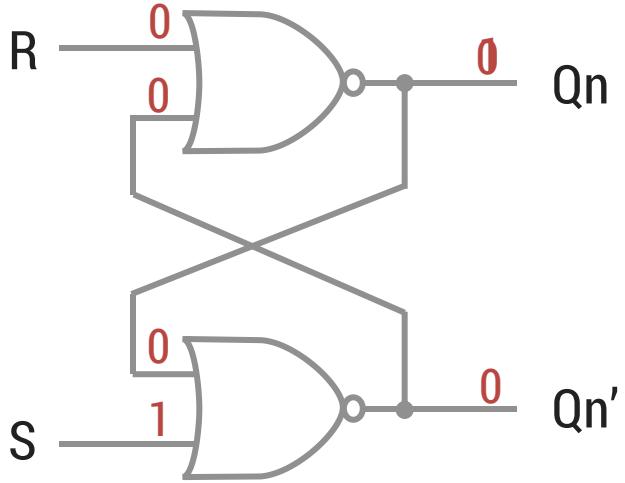


S	R	Q_n	Q_{n+1}	State

S-R latch (Active High) using NOR Gate

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

Logic Symbol(Block diagram)



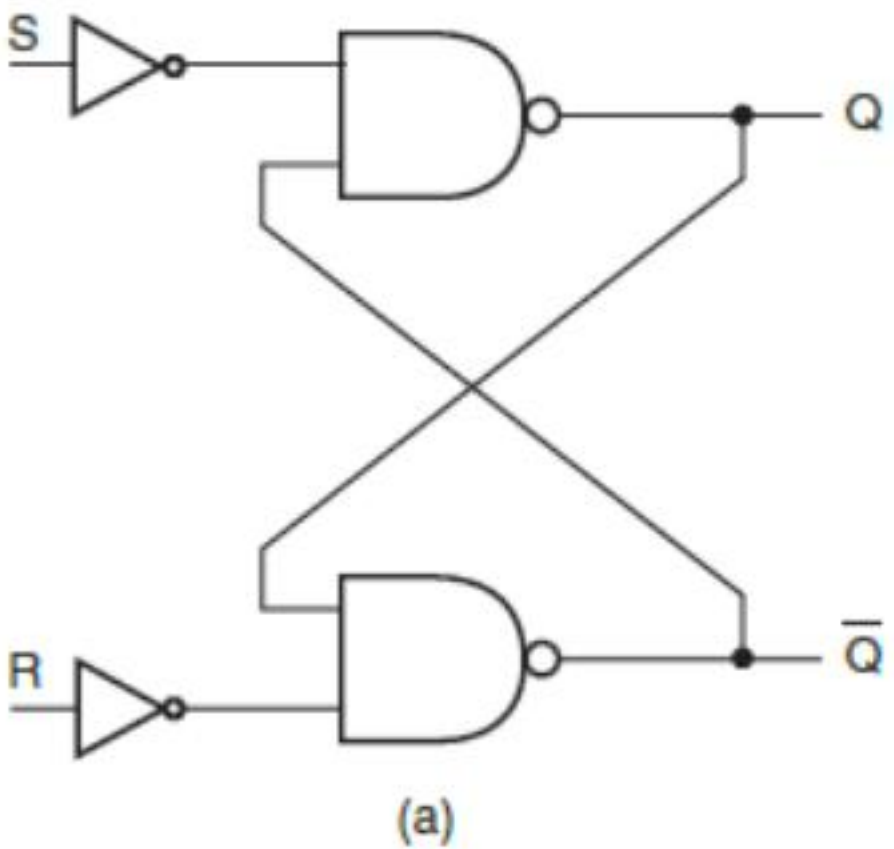
Logic diagram

S	R	$Q_{n(PS)}$	Q_{n+1NS}	State

Characteristic Table

Combination of input and present state and output is next state.

S-R latch (Active High) using NANDGate



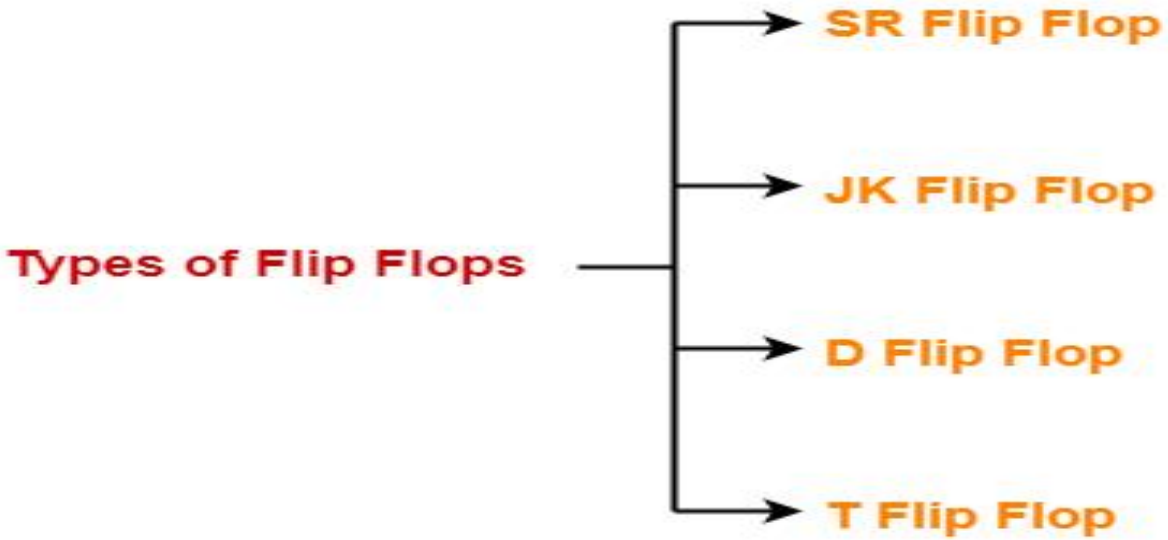
(a)

Operation Mode	S	R	Q_{n+1}
No change	0	0	Q_n
SET	1	0	1
RESET	0	1	0
Forbidden	1	1	—

Clocked latch = FF

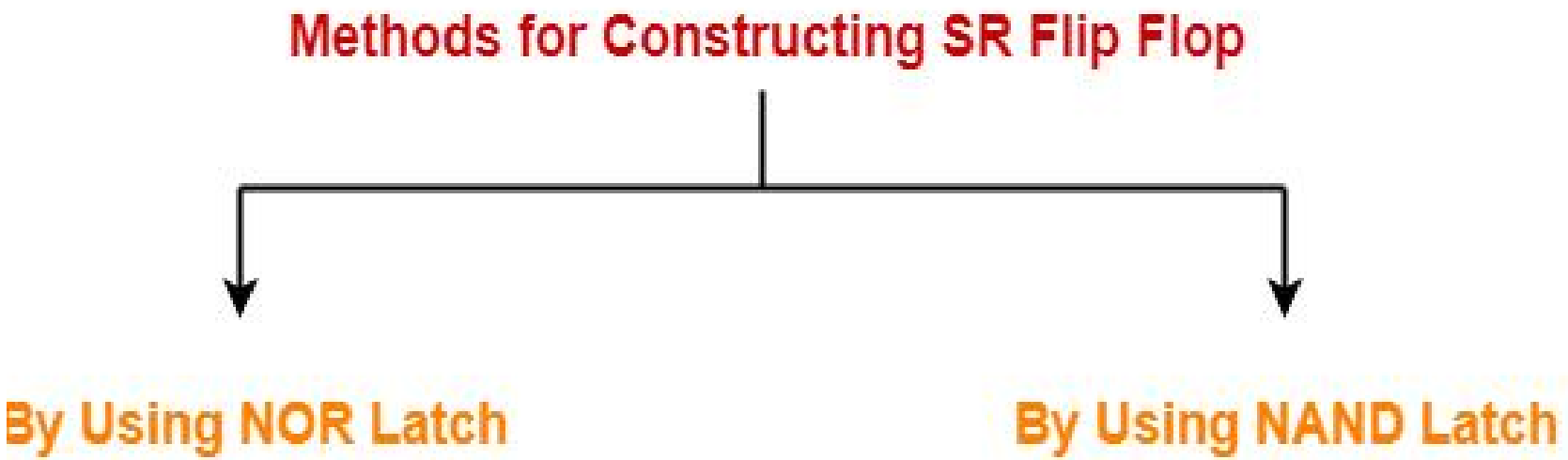
- Output of latch can change any time if input is changed.
- **It is not controllable.**
- To make it controllable, one more input is there called **Enable (CLOCK) signal**.
- A Flip Flop is a **memory element** that is capable of storing one bit of information.
- It is also called as **Bistable Multivibrator** since it has two stable states either 0 or 1.

There are following 4 basic types of flip flops-



Gated S-R Latch/ Clocked S-R Latch (S-R Flip-flop)

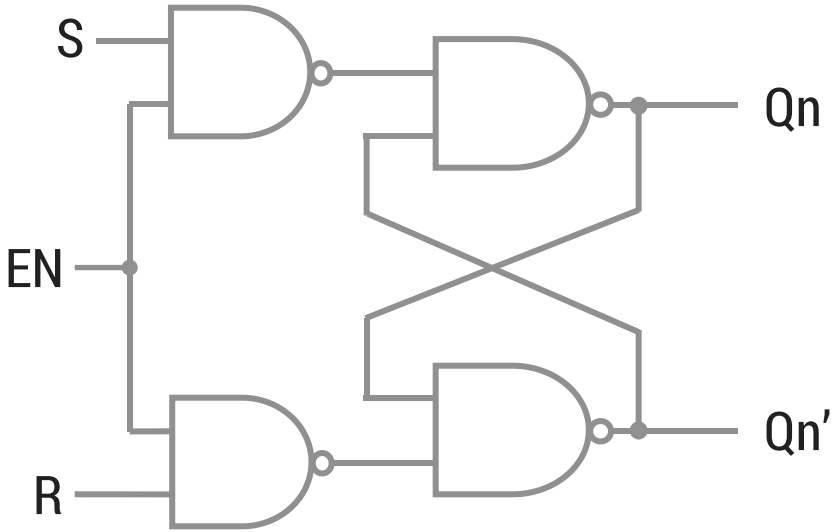
- SR flip flop is the simplest type of flip flops.
- It stands for **Set Reset flip flop.**
- It is a clocked flip flop.



Gated S-R Latch/ Clocked S-R Latch (S-R Flip-flop)

A	B	A·B
0	0	1
0	1	1
1	0	1
1	1	0

Logic Symbol



Logic diagram

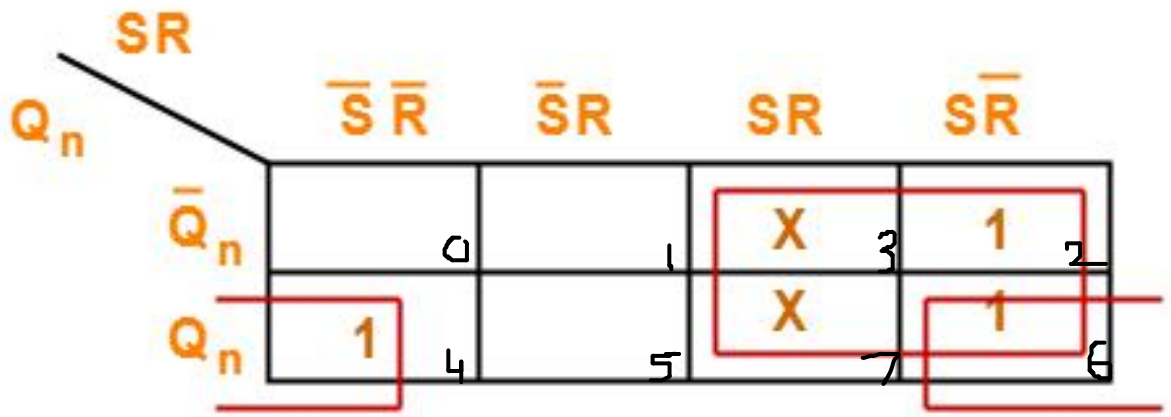
EN	S	R	Q_n	Q_{n+1}	State

Characteristic Table

Characteristic Equation-

Draw a k map using the above truth table-

Q_n	S	R	$Q(n+1)$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	UNKNOWN
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	UNKNOWN



K Map

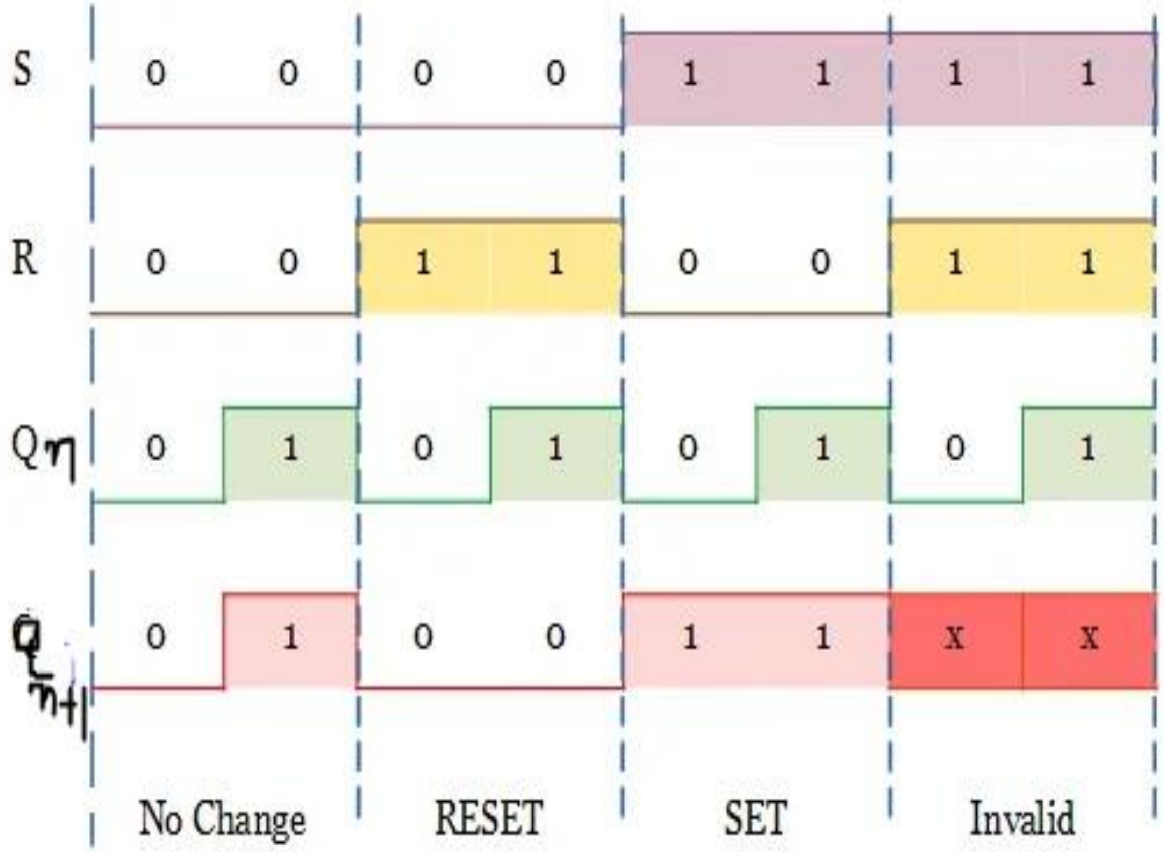
$$Q_{n+1} = S + Q_n R'$$

Switching or Timing diagram of SR FF

S-R Flip-Flop

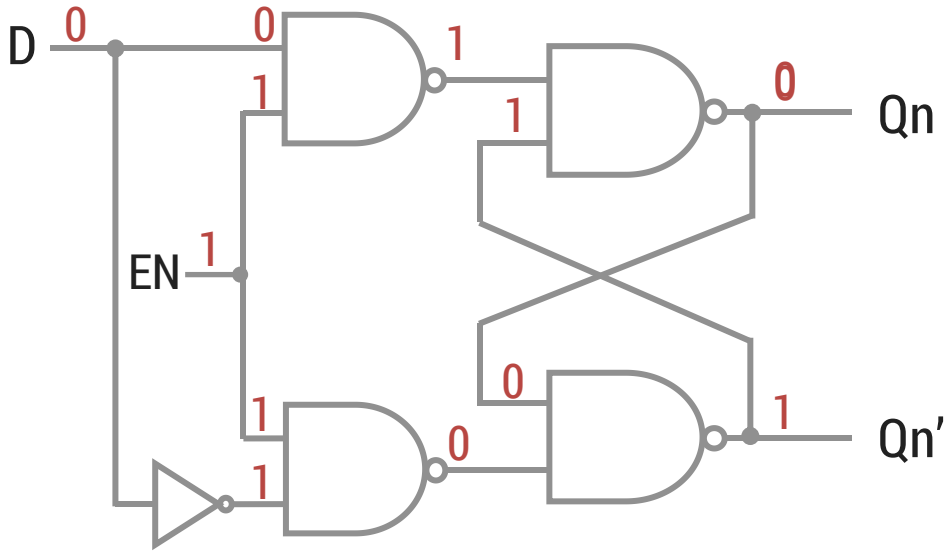
S	R	Q(n+1)
0	0	Q(n)
0	1	0
1	0	1
1	1	Indeterminate

The following figure shows the switching diagram of clocked SR flip flop.



Gated D-Latch or Clocked D-Latch (D Flip-flop)

Logic Symbol



Logic diagram

Characteristic Table

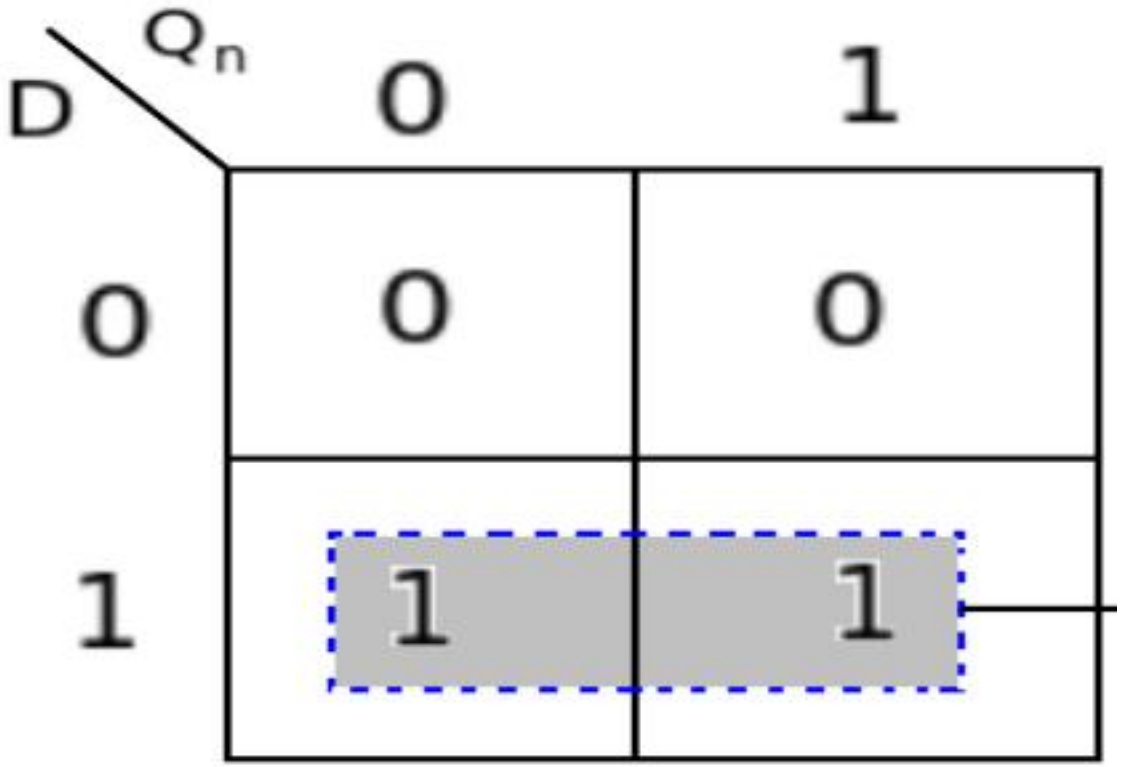
EN	D	Q_n	Q_{n+1}	State

- Here Output follows input , called Transparent latch .
- IC7475 Quad D- FF

Characteristic equation of D Flip-flop

D	Present state Q_n	Next state Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

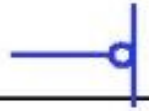
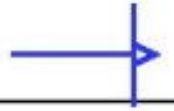
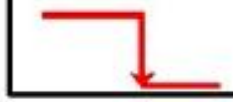
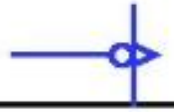
Truth table of D flip flop



Qn+1 = D

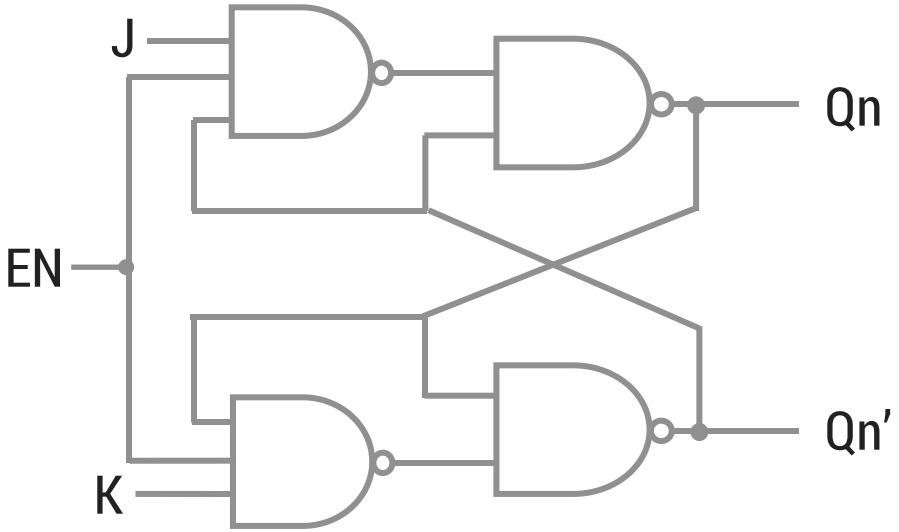
Level-Triggered and Edge-Triggered Flip-Flops

- *Level-triggered* : the output gives response to the inputs during the clock pulse **level** is HIGH or LOW.
- *Edge-triggered* : the output gives response to the inputs only on LOW-to-HIGH(positive edge triggered) or HIGH-to-LOW(negative edge triggered) transition of the clock signal.

Type		Symbol
	Positive Level	
	Negative Level	
	Positive Edge Triggered	
	Negative Edge Triggered	

J-K Flip-flop

Logic Symbol



Logic diagram

Characteristic Table

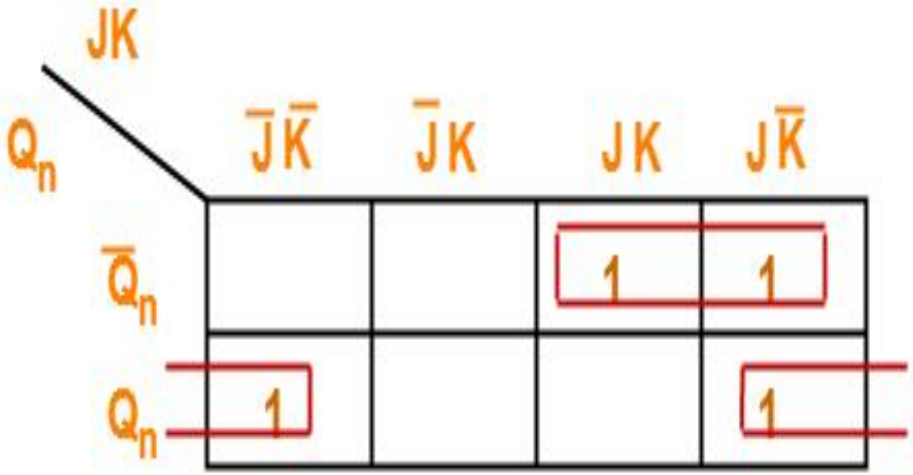
EN	J	K	Q_n	Q_{n+1}	State

Characteristic equation J-K Flip-flop

Draw a k map using the above truth table-

J	K	Present state Q_n	Next state Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Truth table of JK flip flop

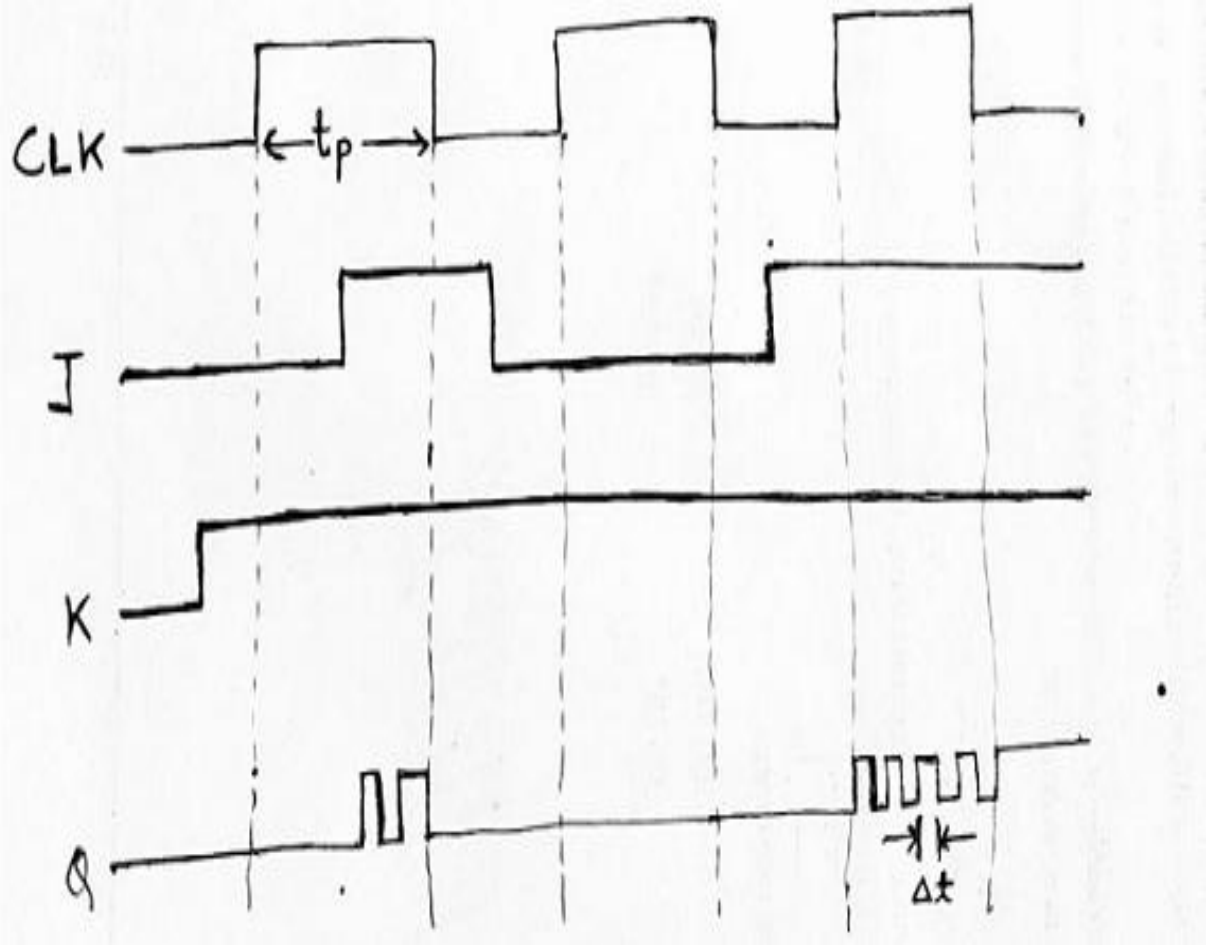


K Map

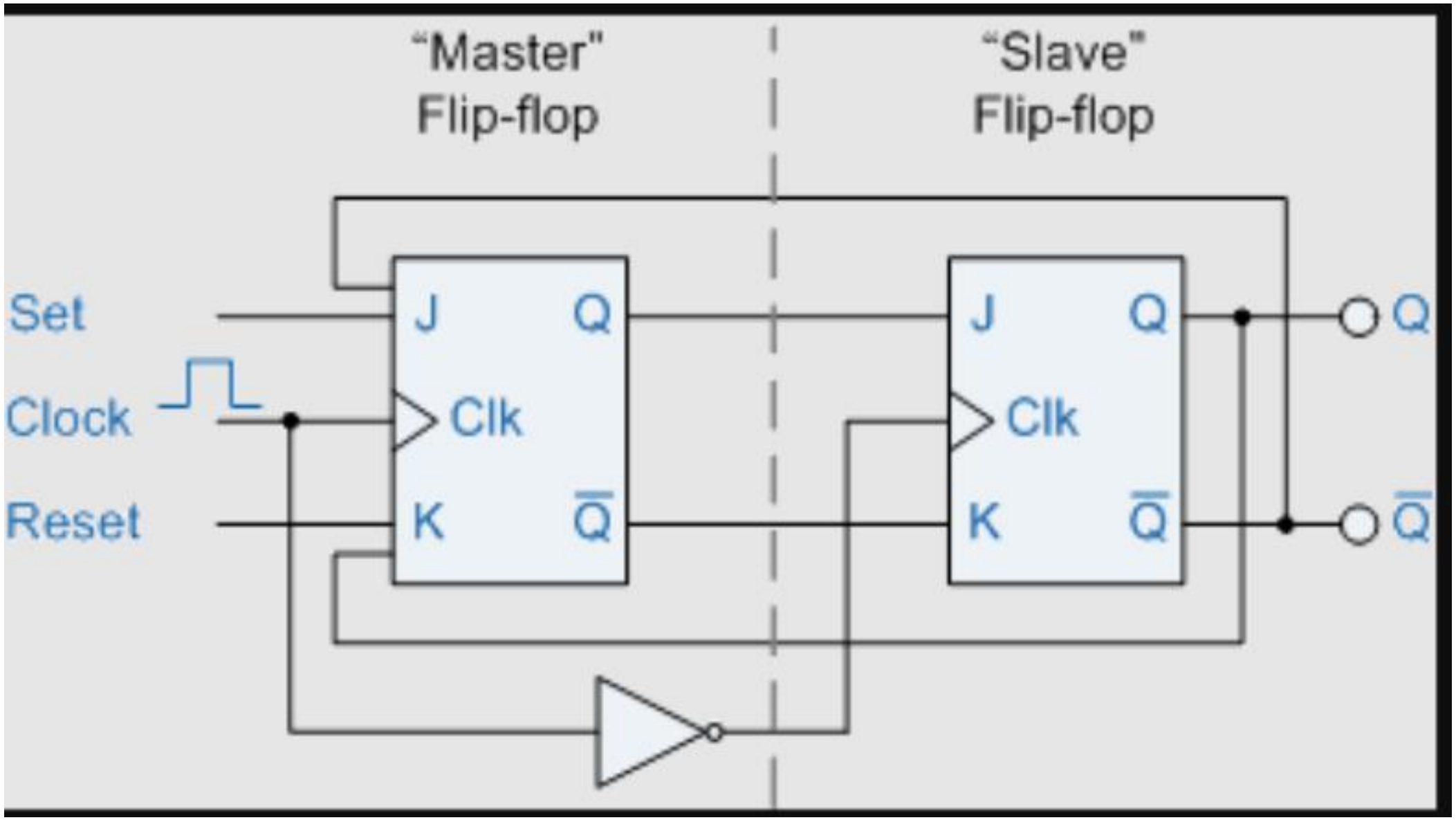
$$Q_{n+1} = Q'_n J + Q_n K'$$

Race around condition of J-K Flip-flop

- For J-K flip-flop, if J=K=1, and if **clk=1 for a long period of time**, then Q output will toggle as long as CLK is high, which makes the output of the flip-flop **unstable or uncertain**.
- This problem is called **race around** condition.
- This problem can be avoided by ensuring
  1. Edge triggering of clock
  2. clock input is at logic "1" only for a very short time.
  3. **Master Slave JK** flip flop.



Master-slave JK flip-flop



Master-slave JK flip-flop

- A combination of two JK flip-flops connected together in a series configuration.
- Out of these, one acts as the **“master”** and the other as a **“slave”**.
- The output from the master flip flop is connected to the two inputs of the slave flip flop whose output is fed back to inputs of the master flip flop.
- The inverter is connected to clock pulse in such a way that the inverted clock pulse is given to the slave flip-flop.
-

Master-Slave FF Working

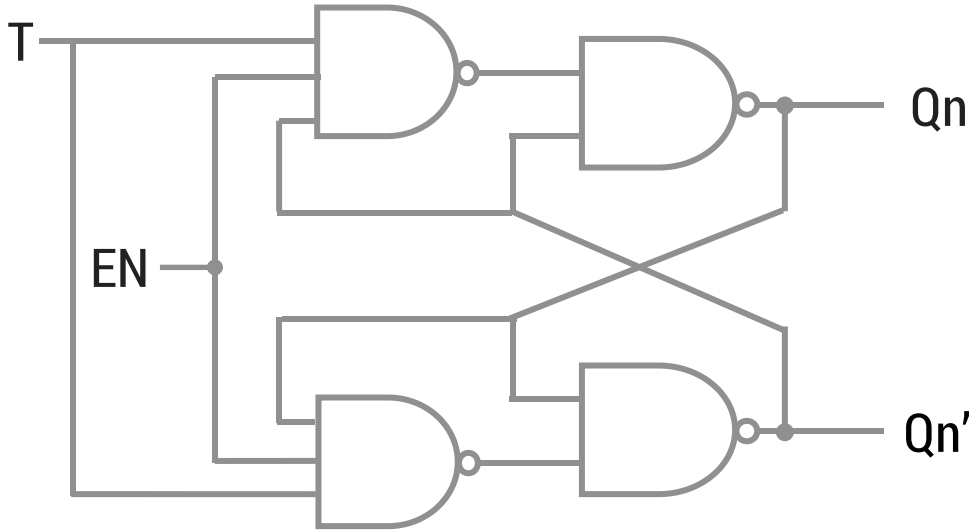
- When CLK = 1, then the **slave can be separated**; the inputs like J & K **affects on master only.**
- CLK = 0, then the data can be transmitted from the master to slave FF and the final **o/p can be obtained.**

- To write more about this FF refer below link,

  https://www.geeksforgeeks.org/master-slave-jk-flip-flop/?ref=lbp

T Flip-flop /Toggle FF

Logic Symbol



Logic diagram

Characteristic Table

EN	T	Q_n	Q_{n+1}	State

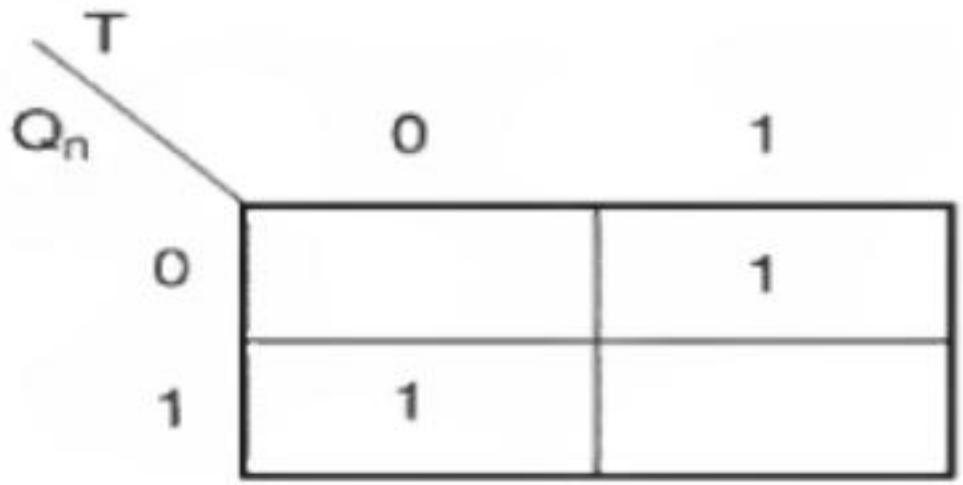
En= 1,
When T = 1 Output = Toggle
When T= 0 Output = remain unchanged.

Characteristic equation of T Flip-flop /Toggle FF

T	Present state Q_n	Next state Q_{n+1}
0	0	0
0	1	1
1	0	1
1	1	0

Truth table of T flip flop

K map for T Flip Flop:



$$Q_{n+1} = T.\overline{Q_n} + \overline{T}.Q_n$$

Excitation Tables

Based on Present State and Next State required input are found called Excitation Table.

S	R	Present state Q_n	Next state Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	x
1	1	1	x

Truth table of SR flip flop

PS	NS	Required inputs	
Q_n	Q_{n+1}	S	R
0	0		
0	1		
1	0		
1	1		

S-R FF Excitation Tables

Excitation Tables

J	K	Present state Q_n	Next state Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Truth table of JK flip flop

PS	NS	Required inputs	
Q_n	Q_{n+1}	J	K
0	0		
0	1		
1	0		
1	1		

J-K FF

Excitation Tables

PS	NS	Required inputs
Q_n	Q_{n+1}	D
0	0	
0	1	
1	0	
1	1	

D FF

PS	NS	Required inputs
Q_n	Q_{n+1}	T
0	0	
0	1	
1	0	
1	1	

T FF

FLIP FLOP CONVERSIONS

- SR to D
- SR to JK
- SR to T
- JK to T
- JK to D
- JK to SR
- D to T
- D to SR
- T to D

Conversion of Flipflop –conversion Steps

Draw the block diagram of the target flip flop from the given problem.

Write truth table for the target flip-flop.

Write excitation table for the available flip-flop.

Write Conversion Table for available flip-flop from Target Flip-flop's Present and Next state.

Using K-map solve equation and draw final diagram.

Conversion of Flipflop –Example 1

Convert SR FF to D FF.

- Here D FF is target FF and SR is given FF.

Apply steps.

Block diagram of D FF

Conversion of Flipflop –Example 1

D flip-flop input	Present State	Next State
D	Q t	Q $t+1$
0	0	0
0	1	0
1	0	1
1	1	1

Truth table of D FF

Present State	Next State	SR flip-flop inputs	
Q t	Q $t+1$	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Excitation table of SR FF

Conversion of Flipflop –Example 1

D flip-flop input	Present State	Next State	SR flip-flop inputs	
D	Q t	Q $t+1$	S	R
0	0	0	0	x
0	1	0	0	1
1	0	1	1	0
1	1	1	x	0

Conversion Table

Conversion of Flipflop –Example 1

Use Kamp.

K-Map for S

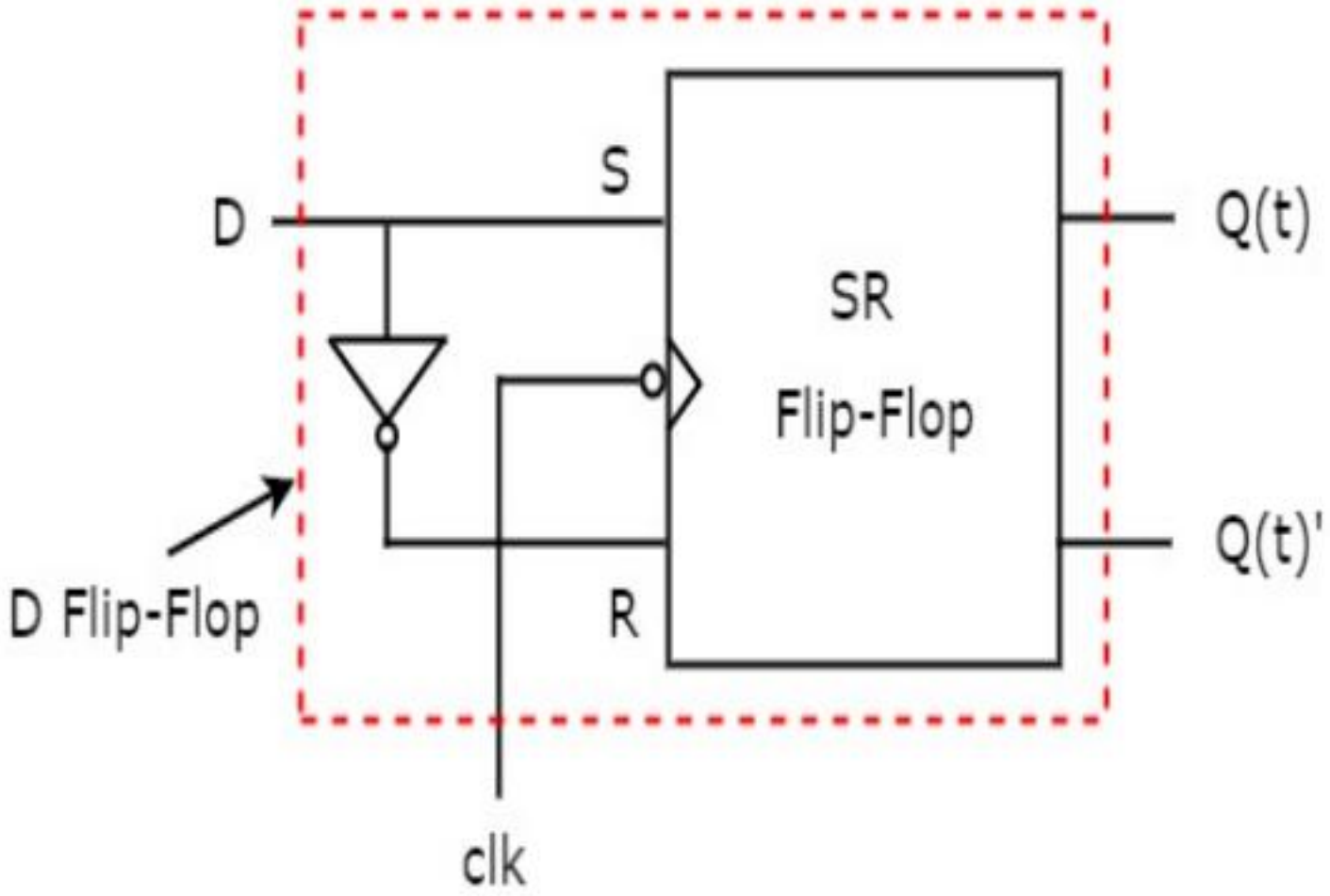
D \ Q(t)	0	1
0		
1	1	x

D

K-Map for R

D \ Q(t)	0	1
0	x	1
1		

D'



Conversion of Flipflop

For all example refer site as below link.

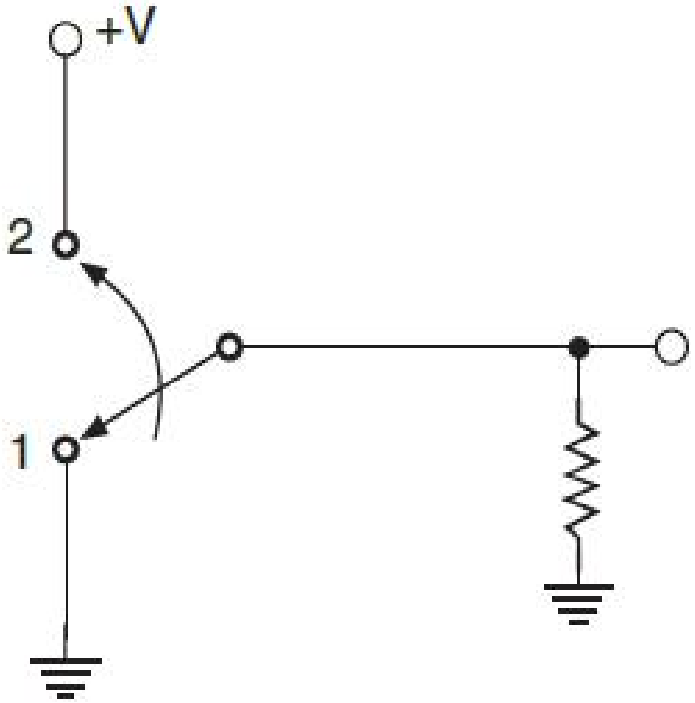
Digital Circuits - Conversion of Flip-Flops (tutorialspoint.com)

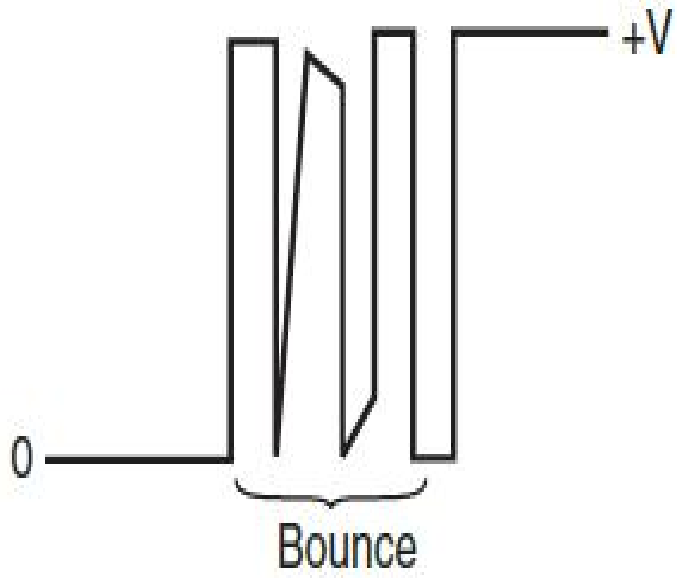
Synchronous and Asynchronous Inputs

- Synchronous inputs are those whose effect on the flip-flop output is synchronized with the clock input. R, S, J, K and D inputs are all synchronous inputs. Asynchronous inputs are those that operate independently of the synchronous inputs and the input clock signal.
- They force the flip-flop output to go to a predefined state irrespective of input.
- i.e. PRESET and CLEAR.
- are not active simultaneously.
- When it is desired that the flip-flop functions as per its synchronous inputs, the asynchronous inputs are kept in their inactive state.

Flip-Flop Applications

Switch De-bouncing

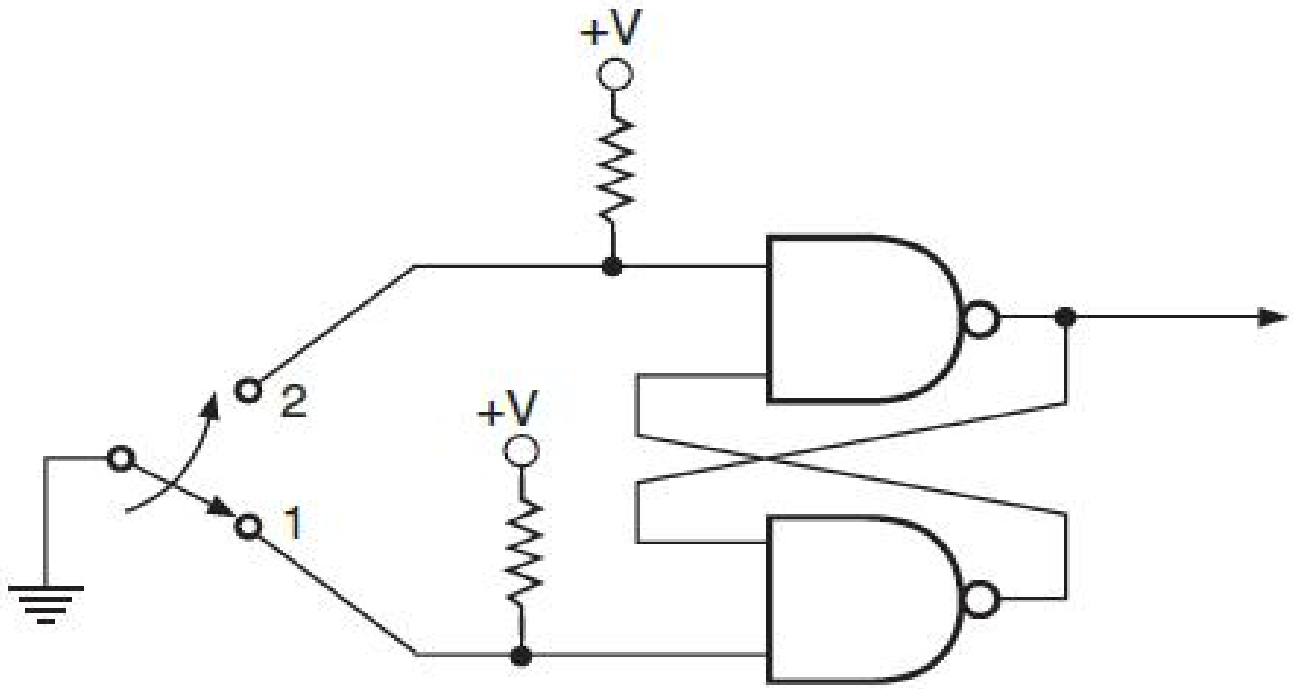


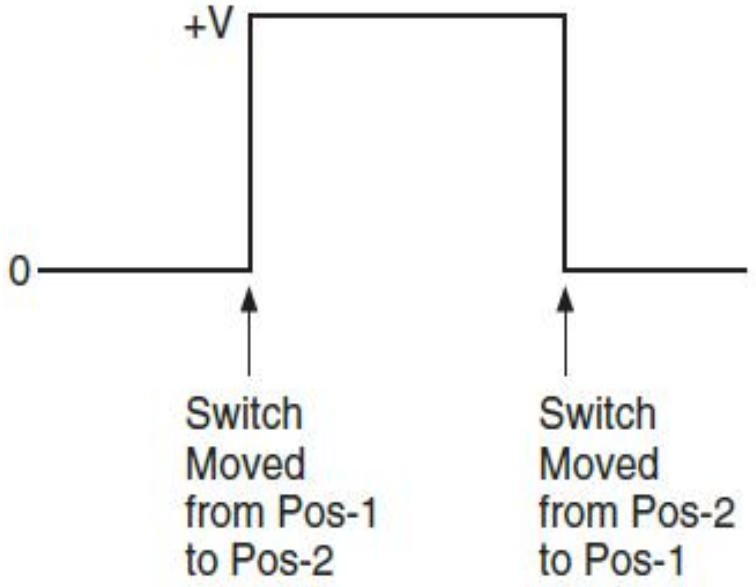


Switch bounce phenomenon→ **It is unacceptable for many digital circuit applications.**

A NAND or a NOR latch can solve this problem and provide a clean output transition.

Flip-Flop Applications → Switch De-bouncing





Switch de-bounce circuit.

Flip-Flop Applications → Switch De-bouncing

- When the switch is in position 1, the output is at a ‘0’ level. When it is moved to position 2, the output goes to a ‘1’ level.

6.15 PROGRAMMABLE DEVICES :

UNIT - 6

6.15.1 Application specific integrated circuit- ASIC:

Different types of standard ICs like decoder, encoder, code converter, multiplexer, de multiplexer, comparator, parity generator, checker etc. are manufactured by various manufacturers for the design and application. All these ICs are called the fixed function ICs because each type of IC is developed for certain definite function. For example, binary to Gray code converter IC can convert the binary code in to the Gray code only. When the digital circuit is not much complicated, the circuit can be designed by making use of some fixed function ICs.

But when the circuit is more complicated, large numbers (hundreds or thousands) of such fixed functions are to be used.

Disadvantages of fixed function ICs :

1. More space is required.
2. Power consumption increases.
3. Cost increases.
4. There is no security as the design can be copied.

To overcome the above problems, application specific integrated circuits (ASIC) are developed. In this the user (circuit designer) gives the design to the manufacturer and the manufacturer fabricates the IC as per the design.

Advantages of ASIC :

1. Space occupied is reduced.
2. Power consumption is reduced.
3. Cost can be reduced if large numbers of ICs are manufactured.
4. The design cannot be copied.
5. Overall size is reduced.

Disadvantages of ASIC :

1. Initial cost is high.
2. Cost also increases as the testing methods are to be developed.

6.15.2 Programmable logic devices-PLDs :

ASICs were developed to overcome the difficulties experienced in realizing the design of the complicated logic circuit. These difficulties can be overcome by making use of the programmable logic design (PLD) also.

Programmable logic device is a LSI chip, which contains numbers of logic gates, flip-flops, registers. These are interconnected and kept on the chip. The connections are made through the fusible links. User retains the links which are required and fuse the remaining links (this is called programming the PLD).

Advantages of PLD :

1. Space occupied is less.
2. Power consumption is less.
3. Reliability is more.
4. Development cost is less.
5. Design can be done quickly.
6. Operation is quick.
7. Amendments can be made in the circuit.
8. Security of design increases.
9. Circuit testing is simple.

The following devices are used as PLD.

1. Programmable read only memory (PROM)
2. Programmable array logic (PAL)
3. Programmable logic array (PLA)
4. Simple programmable logic devices (SPLD)
5. Complex programmable logic devices (CPLD)
6. Field programmable gate array (FPGA)

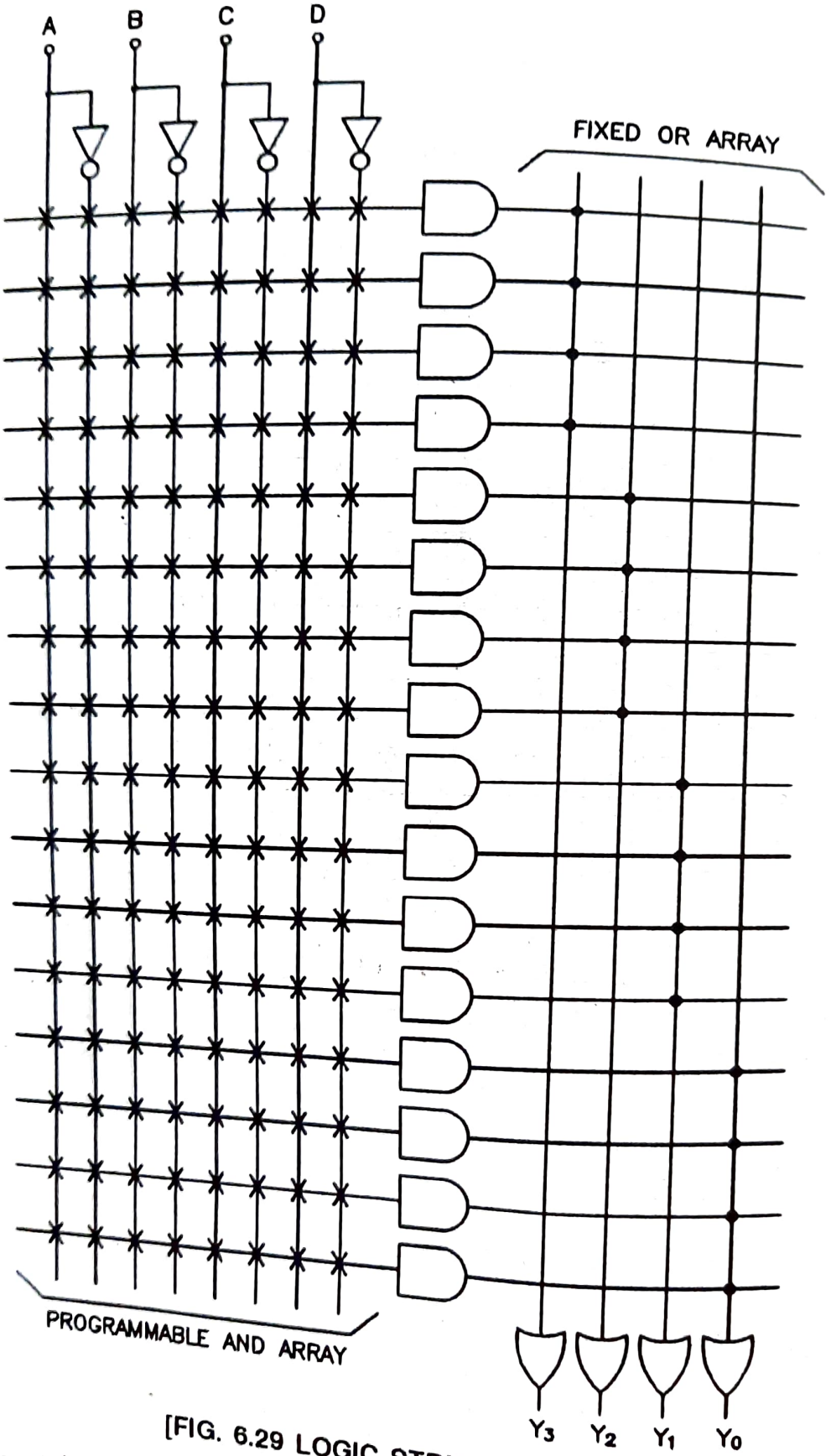
6.15.3 Use of ROM/PROM as PLD :

ROM is basically the memory device in which data is stored permanently. The data stored is not wiped off when the power supply fails. It is used to store fixed type of storage like tables etc.

ROM can also be used for the design of the combinational logic, but in this the users themselves cannot program the device. The user gives the design to the manufacturer and the manufacturer makes suitable arrangement of connection of diodes or gates in ROM IC.

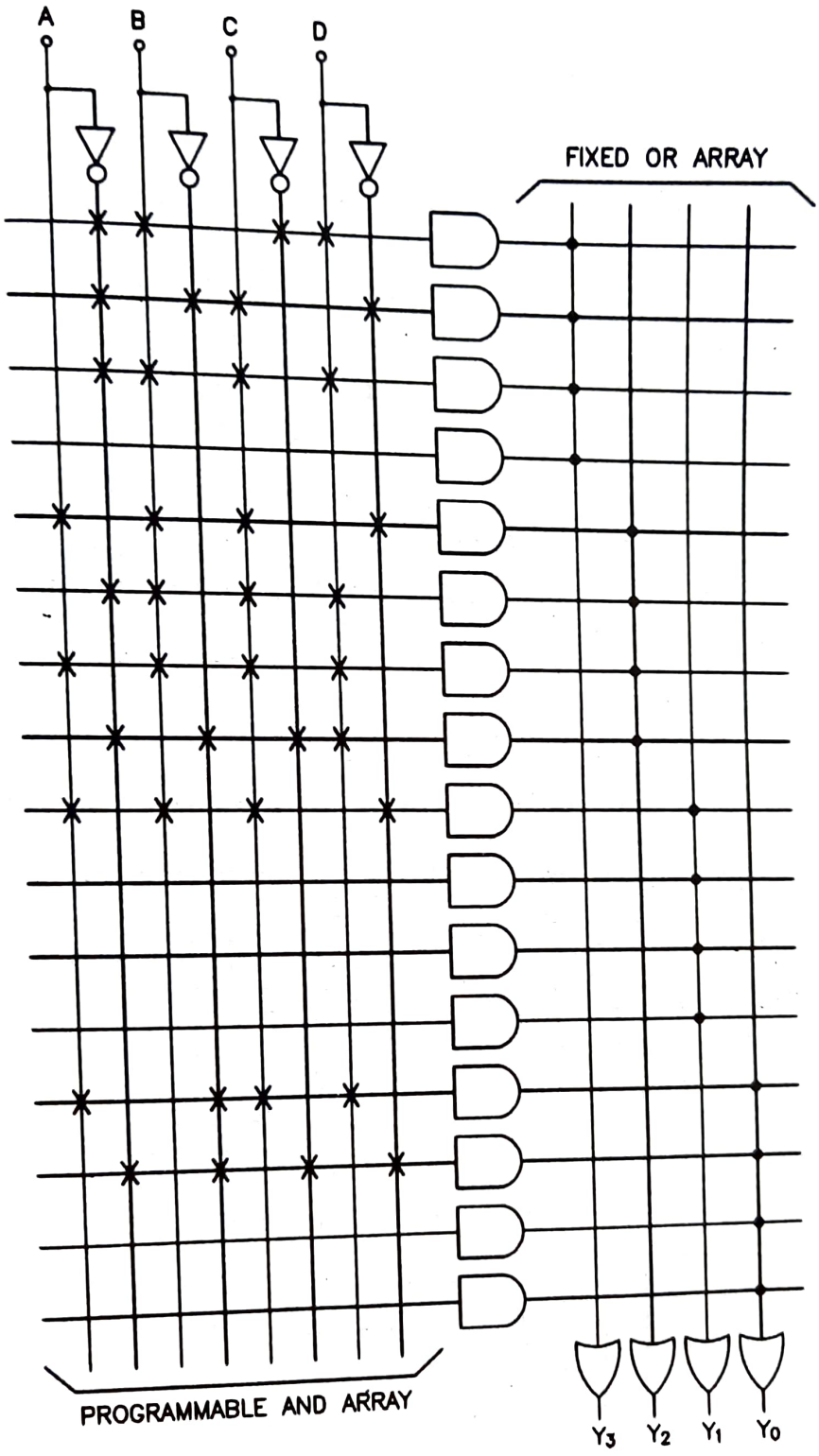
PROM means programmable read only memory. PROM is used to generate Boolean function

6.15.4 Programmable array logic-PAL :



[FIG. 6.29 LOGIC STRUCTURE OF PAL]

The arrangement in PAL is reverse than that of PROM. In this the AND gates on th input side are programmable while the OR gates on the output side are fixed. PAL can also b used to obtain the SOP function. Arrangement of PAL is shown in figure 6.29.



[FIG. 6.30 PROGRAMMING OF PAL]

In this there are four inputs A, B, C, D and four outputs Y_0 through Y_3. There are four inverters and 16 nos. of 8-input AND gates which are programmable. There are 16-input-four nos. of OR gates which cannot be programmed.

Let us suppose we require the following SOP functions.

$$Y_3 = \overline{A}B\overline{C}D + \overline{A}\overline{B}C\overline{D} + \overline{A}BCD$$

$$Y_2 = ABC\overline{D} + \overline{A}BCD + ABCD + \overline{A}\overline{B}\overline{C}D$$

$$Y_1 = ABC\overline{D}$$

$$Y_0 = A\overline{B}CD + \overline{A}\overline{B}\overline{C}\overline{D}$$

Let us consider Y_3.

The first min term is $\overline{A}B\overline{C}D$. So, we have to retain the link nos. second, third, sixth and seventh in the first row and all other links have to be fused.

The second min term is $\overline{A}\overline{B}C\overline{D}$. So, we have to retain the link nos. second, fourth, fifth and eighth in the second row and All other links have to be fused.

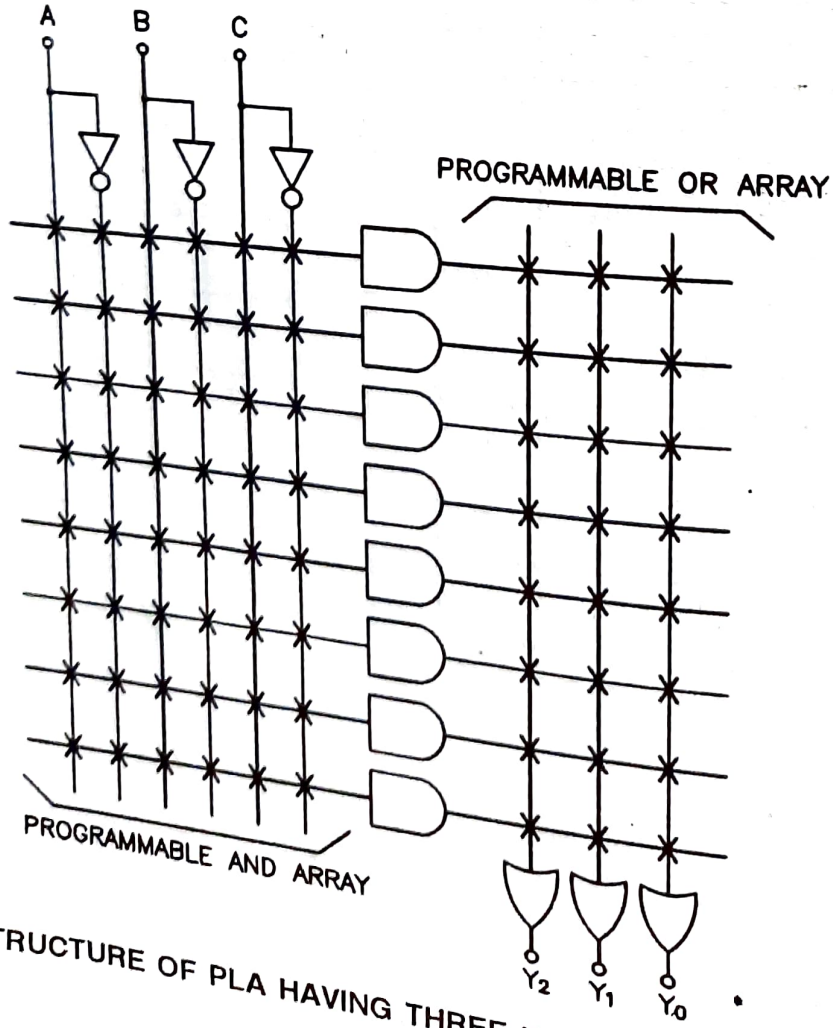
The third min term is $\overline{A}BCD$. So, we have to retain the link nos. second, third, fifth and seventh in the third row and fuse all other links.

There is no fourth min term. So, we shall have to fuse all the eight links in the fourth row.

Similarly we can program for Y_2, Y_1 and Y_0.

Programmed PAL is shown in figure 6.30.

6.15.5 PROGRAMMABLE LOGIC ARRAY – PLA :



[FIG. 6.31 STRUCTURE OF PLA HAVING THREE INPUTS AND THREE OUTPUTS]

The programmable logic array (PLA) includes the characteristics of both the PROM and PAL. In this the AND gates on the input side are programmable and OR gates on the output side are also programmable. Moreover, EX-OR gate is supplied for the controlled inversion. It is either masked programmable type or field programmable type like PLA. In figure 6.31, PLA structure of three inputs and three outputs is shown.

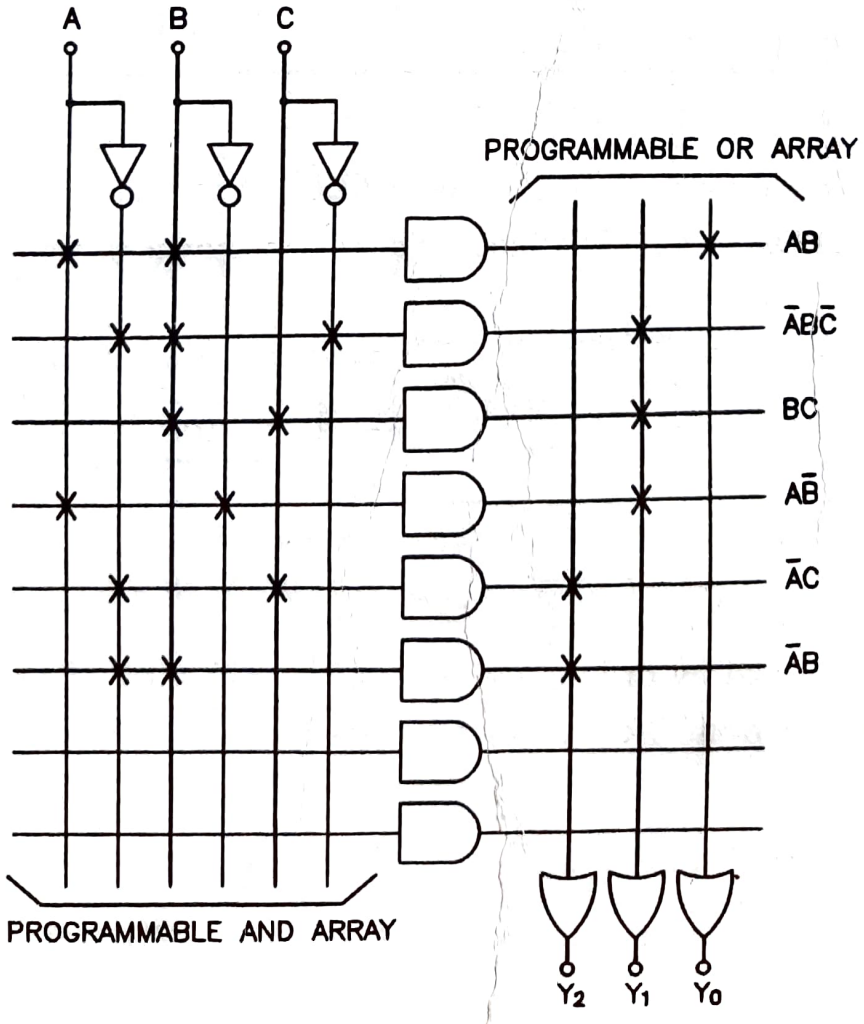
Let us suppose we have to obtain the following Boolean functions.

$$Y_0 = AB$$

$$Y_1 = \overline{A}B\overline{C} + BC + A\overline{B}$$

$$Y_2 = \overline{A}C + \overline{A}B$$

The programmed PLA programmed like PROM and PAL is shown in figure 6.32.



[Fig. 6.32 Programming of PLA]

6.15.6 Comparison of PROM, PAL and PLA :

PROM	PAL	PAL
1. AND array is fixed and OR array is programmable.	1. OR array is fixed and AND array is programmable.	1. Both the AND array and OR array are programmable.
2. Cost is less and easy to use.	2. Cheap and easy to use.	2. Costly and complex.
3. All min terms are decoded.	3. AND array is programmed for required min terms.	3. AND array can be programmed for necessary min terms.
4. Only the standard SOP form of Boolean function can be implemented.	4. Boolean function in SOP form can be implemented.	4. Boolean function in SOP form can be implemented.