LECTURE NOTES

ON

DIGITAL LOGIC DESIGN

II B.Tech I semester (JNTUH-R15)

INFORMATION TECHNOLOGY

CMR TECHNICAL CAMPUS

Kandlakoya, Medchal Road, Hyderabad 501401

UNIT 1

NUMBER SYSTEMS & CODES

- Philosophy of number systems
- Complement representation of negative numbers
- Binary arithmetic
- Binary codes
- Error detecting & error correcting codes   Hamming codes

HISTORY OF THE NUMERAL SYSTEMS:

A **numeral system** (or **system of numeration**) is a linguistic system and mathematical notation for representing numbers of a given set by symbols in a consistent manner. For example, It allows the numeral "11" to be interpreted as the binary numeral for *three*, the decimal numeral for *eleven*, or other numbers in different bases. Ideally, a numeral system will:

- Represent a useful set of numbers (e.g. all whole numbers, integers, or real numbers)
- Give every number represented a unique representation (or at least a standard representation)

Reflect the algebraic and arithmetic structure of the numbers.

For example, the usual decimal representation of whole numbers gives every whole number a unique representation as a finite sequence of digits, with the operations of arithmetic (addition, subtraction, multiplication and division) being present as the standard algorithms of arithmetic. However, when decimal representation is used for the rational or real numbers, the representation is no longer unique: many rational numbers have two numerals, a standard one that terminates, such as 2.31, and another that recurs, such as 2.309999999... . Numerals which terminate have no non-zero digits after a given position. For example,numerals like **2.31 and 2.310 are taken** to be the same, except in the experimental sciences, where greater precision is denoted by the trailing zero.

The most commonly used system of numerals is known as Hindu-Arabic numerals.Great Indian mathematicians Aryabhatta of Kusumapura (5th Century) developed the place value notation. Brahmagupta (6th Century) introduced the symbol zero.

BINARY

The ancient Indian writer Pingala developed advanced mathematical concepts for describing prosody, and in doing so presented the first known description of a binary numeral system.A full set of 8 trigrams and 64 hexagrams, analogous to the 3-bit and 6-bit binary numerals, were known to the ancient Chinese in the classic text *I Ching*. An arrangement of the hexagrams of the *I Ching*, ordered according to the values of the corresponding binary numbers (from 0 to 63), and a method for generating thesame, was developed by the Chinese scholar and philosopher Shao Yong in the 11th century.

In 1854, British mathematician George Boole published a landmark paper detailing an algebraic system of logic that would become known as Boolean algebra. His logical calculus was to become instrumental in the design of digital electronic circuitry. In 1937, Claude Shannon produced his master's thesis at MIT that implemented Boolean algebra and binary arithmetic using electronic relays and switches for the first time in history. Entitled *A SymbolicAnalysis of Relay and Switching Circuits*, Shannon's thesis essentially founded practical digital circuit design.

Binary codes

Binary codes are codes which are represented in binary system with modification from the original ones.

- Weighted Binary codes
- Non Weighted Codes

Weighted binary codes are those which obey the positional weighting principles, each position of the number represents a specific weight. The binary counting sequence is an example.

Decimal	BCD 8421	Excess-3	84-2-1	2421	5211	Bi-Quinary 5043210			5	0	4	3	2	1	0
0	0000	0011	0000	0000	0000	0100001		0		X					X
1	0001	0100	0111	0001	0001	0100010		1		X				X	
2	0010	0101	0110	0010	0011	0100100		2		X			X		
3	0011	0110	0101	0011	0101	0101000		3		X		X			
4	0100	0111	0100	0100	0111	0110000		4		X	X				
5	0101	1000	1011	1011	1000	1000001		5	X						X
6	0110	1001	1010	1100	1010	1000010		6	X					X	
7	0111	1010	1001	1101	1100	1000100		7	X				X		
8	1000	1011	1000	1110	1110	1001000		8	X			X			
9	1001	1111	1111	1111	1111	1010000		9	X		X				

Reflective Code

A code is said to be reflective when code for 9 is complement for the code for 0, and so is for 8 and 1 codes, 7 and 2, 6 and 3, 5 and 4. Codes 2421, 5211, and excess-3 are reflective, whereas the 8421 code is not.

Sequential Codes

A code is said to be sequential when two subsequent codes, seen as numbers in binary representation, differ by one. This greatly aids mathematical manipulation of data. The 8421 and Excess-3 codes are sequential, whereas the 2421 and 5211 codes are not.

Non weighted codes

Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value. Ex: Excess-3 code

Excess-3 Code

Excess-3 is a non weighted code used to express decimal numbers. The code derives its name from the fact that each binary code is the corresponding 8421 code plus 0011(3).

Gray Code

The gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next. The Gray code is non-weighted code, as the position of bit does not contain any weight. The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unitdistance code. In digital Gray code has got a special place.

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

Binary to Gray Conversion

- Gray Code MSB is binary code MSB.
- Gray Code MSB-1 is the XOR of binary code MSB and MSB-1.
- MSB-2 bit of gray code is XOR of MSB-1 and MSB-2 bit of binary code.
- MSB-N bit of gray code is XOR of MSB-N-1 and MSB-N bit of binary code.

Error detection codes

1) Parity bits

A **parity bit** is a bit that is added to a group of source bits to ensure that the number of set bits (i.e., bits with value 1) in the outcome is even or odd. It is a very simple scheme that can be used to detect single or any other odd number (i.e., three, five, etc.) of errors in the output. An even number of flipped bits will make the parity bit appear correct even though the data is erroneous.

2)Checksums

A **checksum** of a message is a modular arithmetic sum of message code words of a fixed word length (e.g.,byte values). The sum may be negated by means of a one's-complement prior to transmission to detect errorsresulting in all-zero messages.Checksum schemes include parity bits, check digits, and longitudinal redundancy checks. Some checksum schemes, such as the Luhn algorithm and the Verhoeff algorithm, are specifically designed to detect errorscommonly introduced by humans in writing down or remembering identification numbers.

3)Cyclic redundancy checks (CRCs)

A **cyclic redundancy check (CRC)** is a single-burst-error-detecting cyclic code and non-secure hash function designed to detect accidental changes to digital data in computer networks. It is characterized by specification of a so-called *generator polynomial*, which is used as the divisor in a polynomial long division over a finite field, taking the input data as the dividend, and where the remainder becomes the result.Cyclic codes have favorable properties in that they are well suited for detecting burst errors. CRCs are particularly easy to implement in hardware, and are therefore commonly used in digital networks and storage devices such as hard disk drives.Even parity is a special case of a cyclic redundancy check, where the single-bit CRC is generated by the divisor $x+1$.

NUMBER BASE CONVERSIONS

Any number in one base system can be converted into another base system

Types

1) decimal to any base
2) Any base to decimal
3) Any base to Any base

Decimal number: $123.45 = 1 \cdot 10^2 + 2 \cdot 10^1 + 3 \cdot 10^0 + 4 \cdot 10^{-1} + 5 \cdot 10^{-2}$

Base b number: $N = a_{q-1}b^{q-1} + \cdots + a_0 b^0 + \cdots + a_{-p}b^{-p}$

$b > 1, \quad 0 <= a_i <= b-1$

Integer part: $a_{q-1}a_{q-2} \cdots a_0$

Fractional part: $a_{-1}a_{-2} \cdots a_{-p}$

Most significant digit: a_{q-1}

Least significant digit: a_{-p}

Binary number (b=2): $1101.01 = 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2}$

Representing number N in base b: $(N)_b$

Complement of digit a: $a' = (b-1)-a$

Decimal system: 9's complement of 3 = 9-3 = 6

Binary system: 1's complement of 1 = 1-1 = 0

Fractional number:

$$(N)_{b_1} = a_{-1}b_2^{-1} + a_{-2}b_2^{-2} + \cdots + a_{-p}b_2^{-p}$$

$$b_2 \cdot (N)_{b_1} = a_{-1} + a_{-2}b_2^{-1} + \cdots + a_{-p}b_2^{-p+1}$$

Example: Convert $(0.3125)_{10}$ to base 8

$0.3125 \cdot 8 = 2.5000$ hence $a_{-1} = 2$

$0.5000 \cdot 8 = 4.0000$ hence $a_{-2} = 4$

Thus, $(0.3125)_{10} = (0.24)_8$

Decimal to Binary

Example: Convert $(432.354)_{10}$ to binary

Q_i	r_i
216	$0 = a_0$
108	$0 = a_1$
54	$0 = a_2$
27	$0 = a_3$
13	$1 = a_4$
6	$1 = a_5$
3	$0 = a_6$
1	$1 = a_7$
	$1 = a_8$

$0.354 \cdot 2 = 0.708$ hence $a_{-1} = 0$

$0.708 \cdot 2 = 1.416$ hence $a_{-2} = 1$

$0.416 \cdot 2 = 0.832$ hence $a_{-3} = 0$

$0.832 \cdot 2 = 1.664$ hence $a_{-4} = 1$

$0.664 \cdot 2 = 1.328$ hence $a_{-5} = 1$

$0.328 \cdot 2 = 0.656$ hence $a_{-6} = 0$

$a_{-7} = 1$

etc.

Thus, $(432.354)_{10} = (110110000.0101101...)_2$

Octal To Binary

Example: Convert $(123.4)_8$ to binary

$(123.4)_8 = (001\ 010\ 011.100)_2$

Example: Convert $(1010110.0101)_2$ to octal

$(1010110.0101)_2 = (001\ 010\ 110.010\ 100)_2 = (126.24)_8$

Error Detection and Correction Codes

- No communication channel or storage device is completely error-free
- As the number of bits per area or the transmission rate increases, more errors occur. • Impossible to detect or correct 100% of the errors

Hamming Codes

1. One of the most effective codes for error-recovery
2. Used in situations where random errors are likely to occur
3. Error detection and correction increases in proportion to the number of parity bits (error-checking bits) added to the end of the information bits

   code word = information bits + parity bits

   Hamming distance: the number of bit positions in which two code words

   differ. 10001001 10110001

   * * *

   Minimum Hamming distance or D(min) : determines its error detecting and correcting capability.

4. Hamming codes can always detect D(min) – 1 errors, but can only correct half of those errors.

EX.

Data Bits	Parity Bit	Code Word
00	0	000
01	1	011
10	1	101
11	0	110

000* 100
001 101*
010 110*
011* 111

5.Single parity bit can only detect error, not correct it 6.Error-correcting codes require more than a single parity bit

EX. 0 0 0 0 0

0 1 0 1 1 1 0 1

1 0

1 1 1 0 1

Minimum Hamming distance = 3

Can detect up to 2 errors and correct 1 error

Cyclic Redundancy Check

1. Let the information byte F = 1001011
2. The sender and receiver agree on an arbitrary binary pattern P. Let P = 1011.
3. Shift F to the left by 1 less than the number of bits in P. Now, F = 1001011000.
4. Let F be the dividend and P be the divisor. Perform “modulo 2 division”.
5. After performing the division, we ignore the quotient. We got 100 for the remainder, which becomes the actual CRC checksum.
6. Add the remainder to F, giving the message M: 1001011 + 100 = 1001011100 = M

M is decoded and checked by the message receiver using the reverse process.

```
            1010100
1011 | 1001011100
          1011
          001001
            1001
            0010
            001011
              1011
              0000            ← Remainder
```

BOOLEAN ALGEBRA AND SWITCHING FUNCTIONS

- **Fundamental postulates of Boolean algebra**
- **Basic theorems and properties**
- **Switching functions**
- **Canonical and Standard forms**
- **Algebraic simplification digital logic gates, properties of XOR gates**
- **Universal gates**
- **Multilevel NAND/NOR realizations**

Boolean Algebra: Boolean algebra, like any other deductive mathematical system, may be defined with aset of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is anycollection of objects having a common property. If **S** is a set and ***x*** and ***y*** are certain objects, then **x** Î **S**denotes that ***x*** is a member of the set **S**, and ***y*** Ï**S** denotes that ***y*** is not an element of **S**. A set with adenumerable number of elements is specified by braces: **A** = {1,2,3,4}, *i.e.* the elements of set **A** are thenumbers 1, 2, 3, and 4. A *binary operator* defined on a set S of elements is a rule that assigns to each pair ofelements from S a unique element from S._ Example: In *a*b=c*, we say that * is a binary operator if it specifies a rule for finding *c* from the pair (*a*,*b*)and also if *a*, *b*, *c* Î S.

CLOSURE: The Boolean system is *closed* with respect to a binary operator if for every pair of Boolean values,it produces a Boolean result. For example, logical AND is closed in the Boolean system because it accepts only Boolean operands and produces only Boolean results.

_ A set *S* is closed with respect to a binary operator if, for every pair of elements of *S*, the binary operator specifies a rule for obtaining a unique element of *S*.

_ For example, the set of natural numbers N = {1, 2, 3, 4, … 9} is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any a, b Î N we obtain a unique c Î N by the operation $a + b = c$.

ASSOCIATIVE LAW:

A binary operator * on a set S is said to be associative whenever $(x * y) * z = x * (y * z)$ for all x, y, z Î S, forall Boolean values x, y and z.

COMMUTATIVE LAW:

A binary operator * on a set S is said to be commutative whenever $x * y = y * x$ for all $x, y, z \in S$

IDENTITY ELEMENT:

A set S is said to have an identity element with respect to a binary operation * on S if there exists an element $e \in S$ with the property $e * x = x * e = x$ for every $x \in S$

BASIC IDENTITIES OF BOOLEAN ALGEBRA

- *Postulate 1(Definition)*: A Boolean algebra is a closed algebraic system containing a set K of two or more elements and the two operators · and + which refer to logical AND and logical OR •$x + 0 = x$
- $x \cdot 0 = 0$
- $x + 1 = 1$
- $x \cdot 1 = 1$
- $x + x = x$
- $x \cdot x = x$
- $x + x' = x$
- $x \cdot x' = 0$
- $x + y = y + x$
- $xy = yx$
- $x + (y + z) = (x + y) + z$
- $x (yz) = (xy) z$
- $x (y + z) = xy + xz$
- $x + yz = (x + y)(x + z)$
- $(x + y)' = x' y'$
- $(xy)' = x' + y'$
- $(x')' = x$

DeMorgan's Theorem

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

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

Generalized DeMorgan's Theorem

(a) $(a + b + \ldots z)' = a'b' \ldots z'$

(b) $(a.b \ldots z)' = a' + b' + \ldots z,,$

LOGIC GATES

Formal logic: In formal logic, a statement (proposition) is a declarative sentence that is either true(1) or false (0). It is easier to communicate with computers using formal logic.

- Boolean variable: Takes only two values – either true (1) or false (0). They are used as basic units of formal logic.
- Boolean algebra: Deals with binary variables and logic operations operating on those variables.
- Logic diagram: Composed of graphic symbols for logic gates. A simple circuit sketch that represents inputs and outputs of Boolean functions.

Name	Graphic symbol	Algebraic function	Truth table	
Inverter	A → x	x = A'	A | x 0 | 1 1 | 0	
AND	A, B → x	x = AB	A B | x 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1	True if both are true.
OR	A, B → x	x = A + B	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1	True if either one is true.

- Other common gates include:

Name	Graphic symbol	Algebraic function	Truth table	
Exclusive-OR (XOR)	A, B → x	x = A ⊕ B = A'B + AB'	A B | x 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0	Parity check: True if only one is true.
NAND	A, B → x	x = (AB)'	A B | x 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0	Inversion of AND.
NOR	A, B → x	x = A + B	A B | x 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0	Inversion of OR.

UNIT 2

MINIMIZATION AND DESIGN OF COMBINATIONAL CIRCUITS

INTRODUCTION

Minimization of switching functions is to obtain logic circuits with least circuit complexity. This goal is very difficult since how a minimal function relates to the implementation technology is important.

For example, If we are building a logic circuit that uses discrete logic made of small scale Integration ICs(SSIs) like 7400 series, in which basic building block are constructed and are available for use. The goal of minimization would be to reduce the number of ICs and not the logic gates. For example, If we require two 6 and gates and 5 Or gates,we would require 2 AND ICs(each has 4 AND gates) and one OR IC. (4 gates). On the other hand if the same logic could be implemented with only 10 nand gates, we require only 3 ICs. Similarly when we design logic on Programmable device, we may implement the design with certain number of gates and remaining gates may not be used. Whatever may be the criteria of minimization we would be guided by the following:

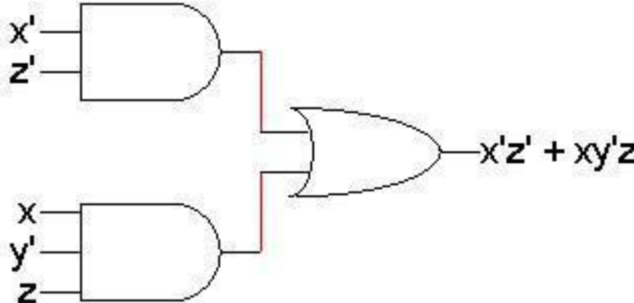
- Boolean algebra helps us simplify expressions and circuits
- Karnaugh Map: A graphical technique for simplifying a Boolean expression into either form: o minimal sum of products (MSP)

  o minimal product of sums (MPS)
- Goal of the simplification.

  o There are a minimal number of

  product/sum terms o Each term has

  a minimal number of literals
- Circuit-wise, this leads to a *minimal* two-levelimplementation



- A two-variable function has four possible minterms. We can re-arrange these minterms into a Karnaugh map

x	y	minterm
0	0	x'y'
0	1	x'y
1	0	xy'
1	1	xy

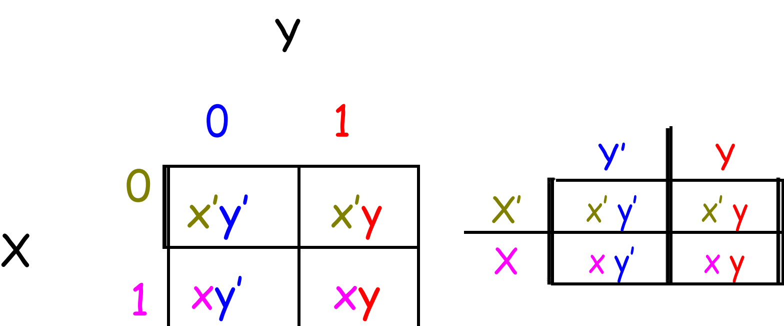
0

X

1

	0	1
	x'y'	x'y
	xy'	xy

- Now we can easily see which minterms contain common literals
  - Minterms on the left and right sides contain y' and y respectively
  - Minterms in the top and bottom rows contain x' and x respectively

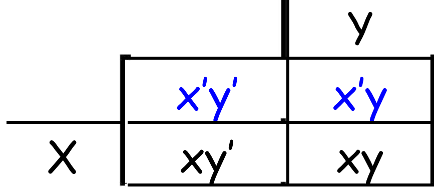


k-map Simplification

- Imagine a two-variable sum of minterms

$x'y' + x'y$

- Both of these minterms appear in the top row of a Karnaugh map, which means that they both contain the literal x'



- What happens if you simplify this expression using Boolean algebra?
- $x'y' + x'y = x'(y' + y)$ [Distributive]

- = x‘‘ 1 [y + y‘‘ = 1]
- = x‘‘ [x 1 = x]

A Three-Variable Karnaugh Map

- For a three-variable expression with inputs x, y, z, the arrangement of minterms is more tricky:

X \ YZ	00	01	11	10
0	x'y'z'	x'y'z	x'yz	x'yz'
1	xy'z'	xy'z	xyz	xyz'

X \ YZ	00	01	11	10
0	m_0	m_1	m_3	m_2
1	m_4	m_5	m_7	m_6

- Another way to label the K-map (use whichever you like):

			Y	Y
	x'y'z'	x'y'z	x'yz	x'yz'
X	xy'z'	xy'z	xyz	xyz'
		Z	Z	

			Y	Y
	m_0	m_1	m_3	m_2
X	m_4	m_5	m_7	m_6
		Z	Z	

- With this ordering, any group of 2, 4 or 8 adjacent squares on the map contains common literals that can be factored out

			Y	Y
	x'y'z'	x'y'z	x'yz	x'yz'
X	xy'z'	xy'z	xyz	xyz'
		Z	Z	

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

- "Adjacency" includes wrapping around the left and right sides:

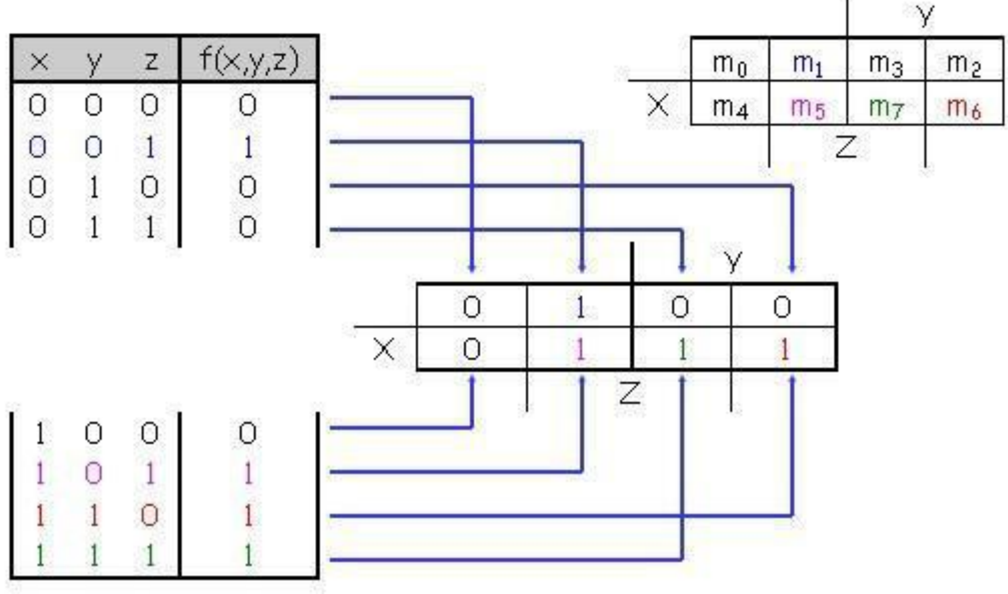
			Y	Y
	x'y'z'	x'y'z	x'yz	x'yz'
X	xy'z'	xy'z	xyz	xyz'
		Z	Z	

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

- We'll use this property of adjacent squares to do our simplifications.

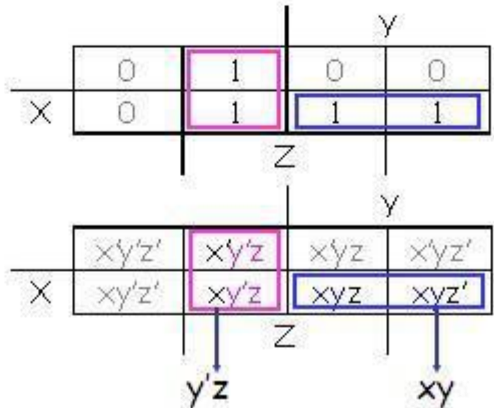
K-maps From Truth Tables

- We can fill in the K-map directly from a truth table
  - The output in row *i* of the table goes into square m_i of the K-map
  - Remember that the rightmost columns of the K-map are "switched"



Reading the MSP from the K-map

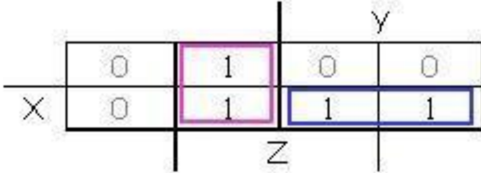
- You can find the minimal SoP expression
  - Each rectangle corresponds to one product term
  - The product is determined by finding the common literals in that rectangle



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

Grouping the Minterms Together

- The most difficult step is grouping together all the 1s in the K-map
  - Make rectangles around groups of one, two, four or eight 1s
  - All of the 1s in the map should be included in at least one rectangle
  - Do *not* include any of the 0s
  - Each group corresponds to one product term



K-map Simplification of SoP Expressions

- Let's consider simplifying $f(x,y,z) = xy + y'z + xz$
- You should convert the expression into a sum of minterms form,
  - The easiest way to do this is to make a truth table for the function, and then read off the minterms
  - You can either write out the literals or use the minterm shorthand
- Here is the truth table and sum of minterms for our example:

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$f(x,y,z) = x'y'z + xy'z + xyz' + xyz$$
$$= m_1 + m_5 + m_6 + m_7$$

Unsimplifying Expressions

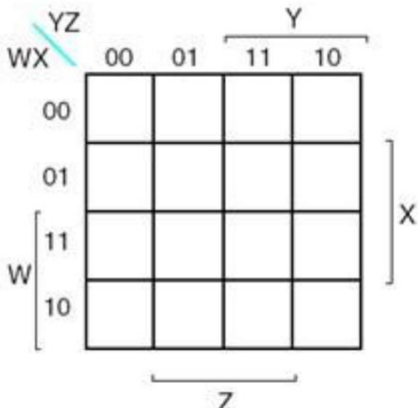
- You can also convert the expression to a sum of minterms with Boolean algebra
  - Apply the distributive law in reverse to add in missing variables.
  - Very few people actually do this, but it's occasionally useful.

$$
\begin{aligned}
xy + y'z + xz &= (xy \bullet 1) + (y'z \bullet 1) + (xz \bullet 1) \\
&= (xy \bullet (z' + z)) + (y'z \bullet (x' + x)) + (xz \bullet (y' + y)) \\
&= (xyz' + xyz) + (x'y'z + xy'z) + (xy'z + xyz) \\
&= xyz' + xyz + x'y'z + xy'z \\
&= m_1 + m_5 + m_6 + m_7
\end{aligned}
$$

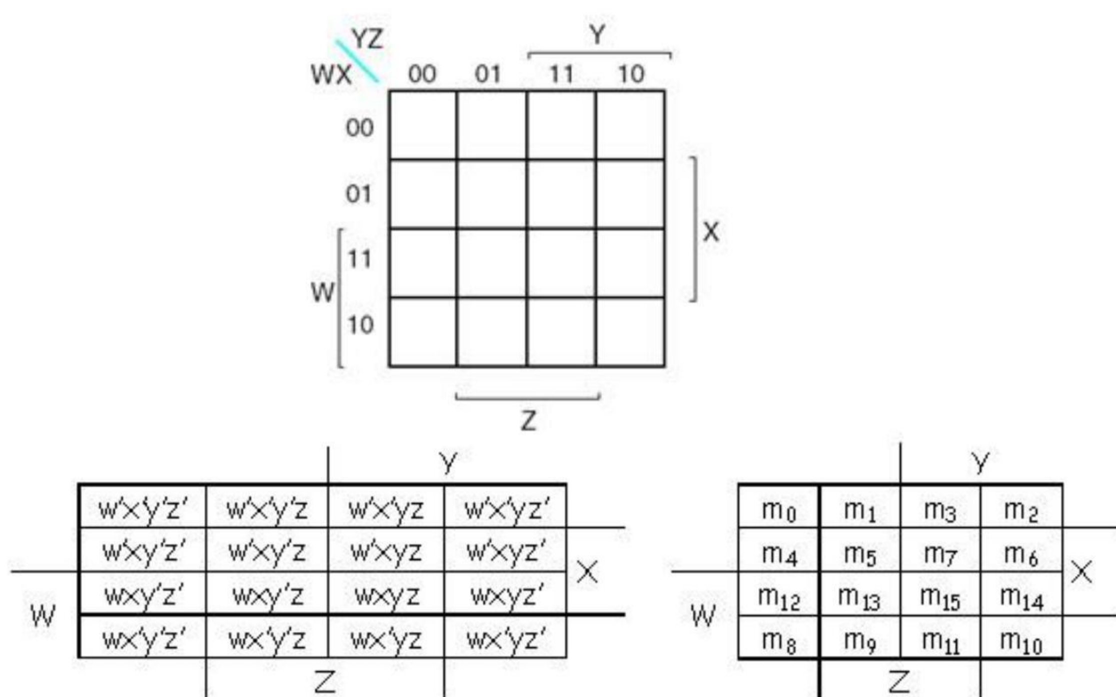
- In both cases, we're actually "unsimplifying" our example expression
  - The resulting expression is larger than the original one!
  - But having all the individual minterms makes it easy to combine them together with the K-map

Four-variable K-maps – f(W,X,Y,Z)

- We can do four-variable expressions too!
  - The minterms in the third and fourth columns, *and* in the third and fourth rows, are switched around.
  - Again, this ensures that adjacent squares have common literals



- Grouping minterms is similar to the three-variable case, but:
  - You can have rectangular groups of 1, 2, 4, 8 or 16 minterms
  - You can wrap around *all four* sides



w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'
w'xy'z'	w'xy'z	w'xyz	w'xyz'
wxy'z'	wxy'z	wxyz	wxyz'
wx'y'z'	wx'y'z	wx'yz	wx'yz'

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

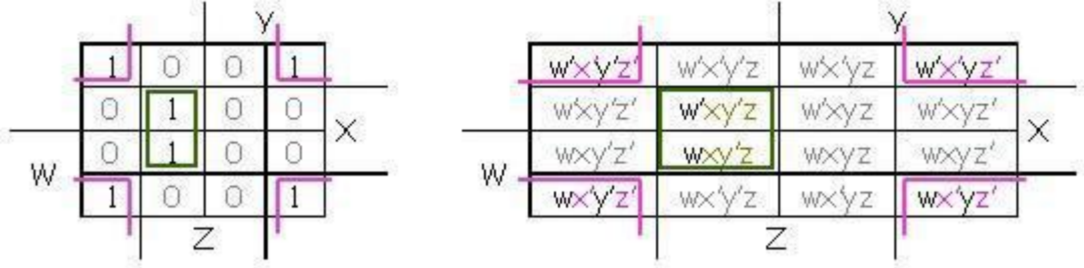
Simplify $m_0+m_2+m_5+m_8+m_{10}+m_{13}$

- The expression is already a sum of minterms, so here's the K-map:

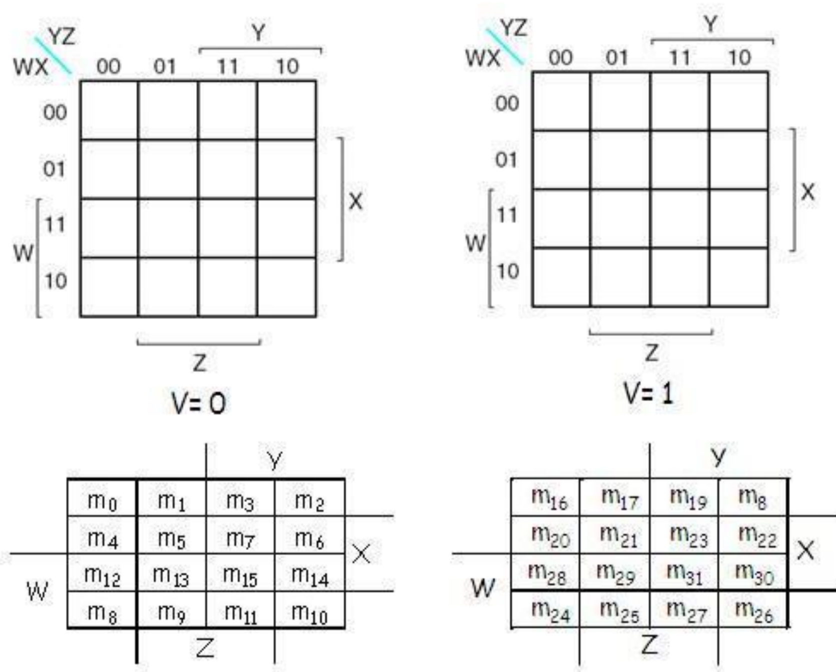
1	0	0	1
0	1	0	0
0	1	0	0
1	0	0	1

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

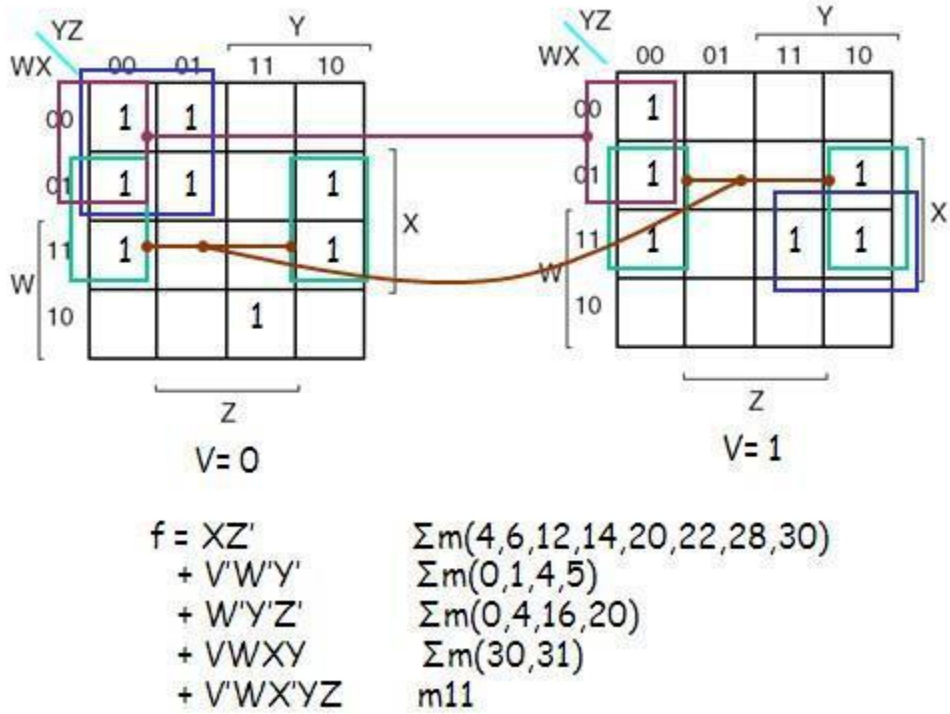
- We can make the following groups, resulting in the MSP x'z' + xy'z



Five-variable K-maps – f(V,W,X,Y,Z)



Simplify f(V,W,X,Y,Z)=Σm(0,1,4,5,6,11,12,14,16,20,22,28,30,31)

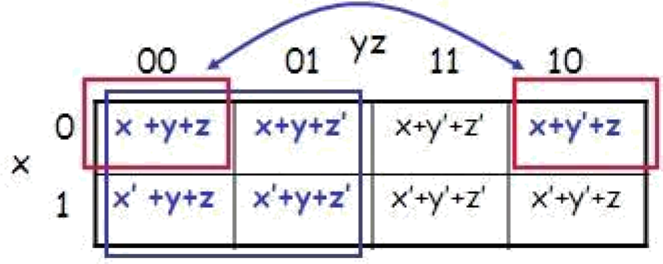


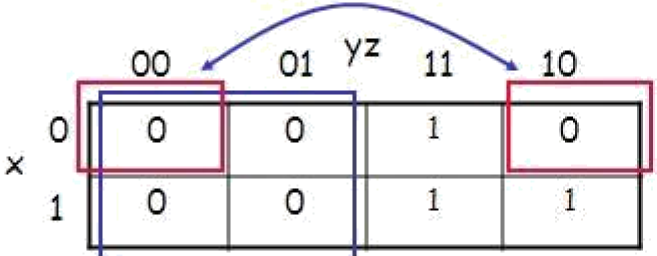
PoS Optimization

- Maxterms are grouped to find minimal PoS expression

x \ yz	00	01	11	10
0	x+y+z	x+y+z'	x+y'+z'	x+y'+z
1	x'+y+z	x'+y+z'	x'+y'+z'	x'+y'+z

- $F(W,X,Y,Z) = \prod M(0,1,2,4,5)$



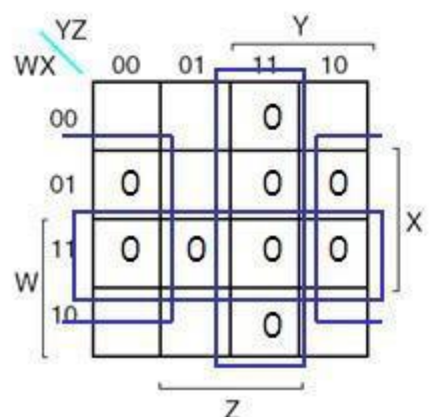


$F(W,X,Y,Z) = Y \cdot (X + Z)$

PoS Optimization from SoP

$F(W,X,Y,Z) = \Sigma m(0,1,2,5,8,9,10)$

$= \prod M(3,4,6,7,11,12,13,14,15)$



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

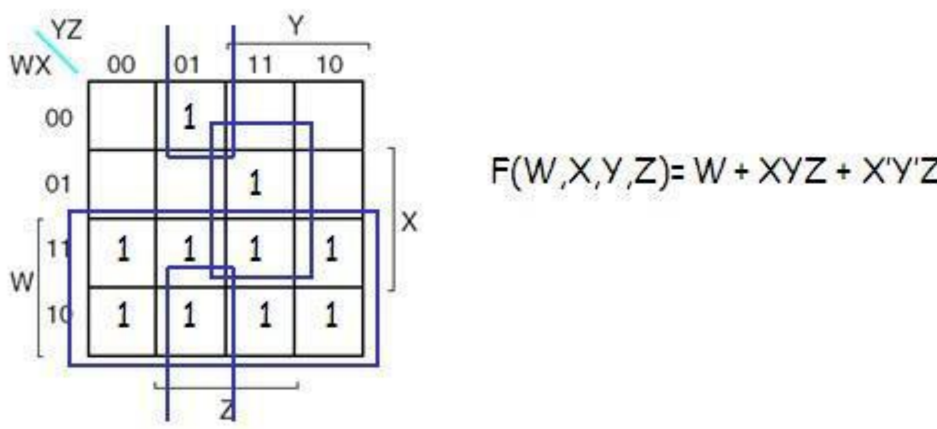
Or,

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

Which one is the minimal one?

SoP Optimization from PoS

$$F(W,X,Y,Z) = \prod M(0,2,3,4,5,6)$$
$$= \Sigma m(1,7,8,9,10,11,12,13,14,15)$$



Don't care

- You don't always need all 2^n input combinations in an n-variable function
  - If you can guarantee that certain input combinations never occur
  - If some outputs aren't used in the rest of the circuit
- We mark don't-care outputs in truth tables and K-maps with Xs.

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	X
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	1

- Within a K-map, each X can be considered as either 0 or 1. You should pick the interpretation that allows for the most simplification.

- Find a MSP for

$$f(w,x,y,z) = \Sigma m(0,2,4,5,8,14,15), d(w,x,y,z) = \Sigma m(7,10,13)$$

This notation means that input combinations wxyz = 0111, 1010 and 1101 (corresponding to minterms m_7, m_{10} and m_{13}) are unused.

1	0	0	1	
1	1	×	0	x
0	×	1	1	x
1	0	0	×	

(y spans the 3rd and 4th columns; z spans the 2nd and 3rd columns; w spans the 3rd and 4th rows)

K-map Summary

- K-maps are an alternative to algebra for simplifying expressions
  - The result is a MSP/MPS, which leads to a minimal two-level circuit
  - It‟s easy to handle don‟t-care conditions
  - K-maps are really only good for manual simplification of small expressions...
  - Things to keep in mind:
  - Remember the correct order of minterms/maxterms on the K-map
  - When grouping, you can wrap around all sides of the K-map, and your groups can overlap
  - Make as few rectangles as possible, but make each of them as large as possible. This leads to fewer, but simpler, product terms
  - There may be more than one valid solution

Combinational Logic

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of input variables, logic gates, and output variables.

Analysis procedure

To obtain the output Boolean functions from a logic diagram, proceed as follows:

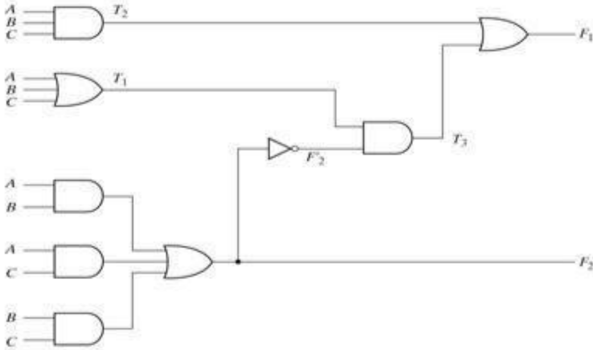
1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

Example

$F_2 = AB + AC + BC$; $T_1 = A + B + C$; $T_2 = ABC$; $T_3 = F_2'T_1$;

$F_1 = T_3 + T_2$

$F_1 = T_3 + T_2 = F_2'T_1 + ABC = A'BC' + A'B'C + AB'C' + ABC$



Derive truth table from logic diagram

We can derive the truth table by using the above circuit

A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

UNIT 4

- Logic circuits that can store information
- Flip-flops, which store a single bit
- Registers, which store multiple bits
- Shift registers, which shift the contents of a register
- Counters of various types

The Basic Latch

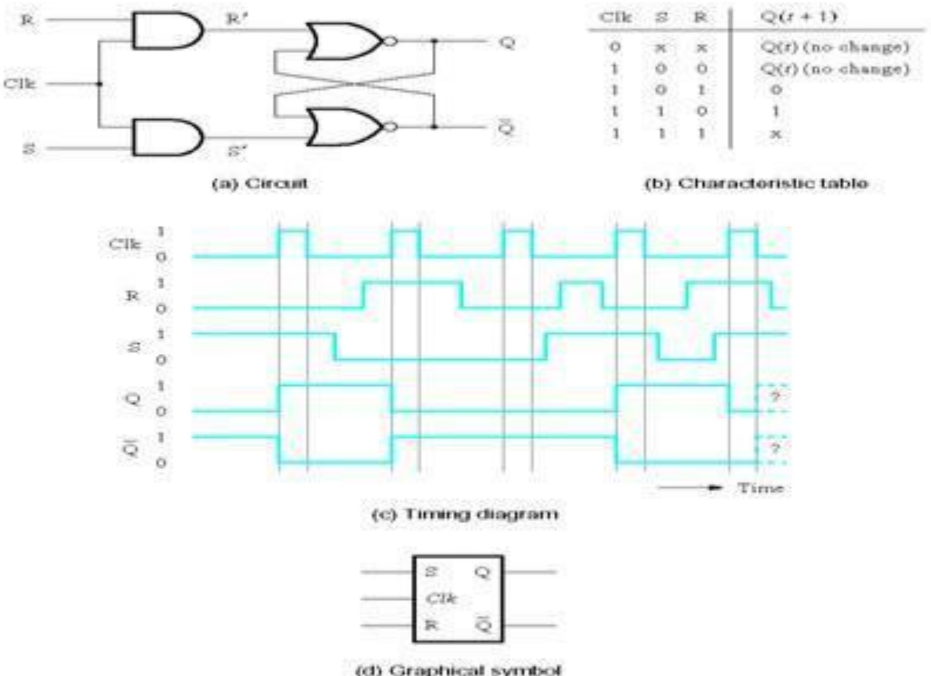
- **Basic latch**is a feedback connection of two NOR gates or two NAND gates
- It can store one bit of information

It can be set to 1 using the *S* input and reset to 0 using the *R* input

The Gated Latch

- **Gated latch** is a basic latch that includes input gating and a control signal
- The latch retains its existing state when the control input is equal to 0
- Its state may be changed when the control signal is equal to 1. In our discussion we referred to the control input as the clock
- We consider two types of gated latches:
  - **Gated SR latch** uses the *S* and *R* inputs to set the latch to 1 or reset it to 0, respectively.
  - **Gated D latch** uses the *D* input to force the latch into a state that has the same logic value as the *D* input.

Gated S/R Latch



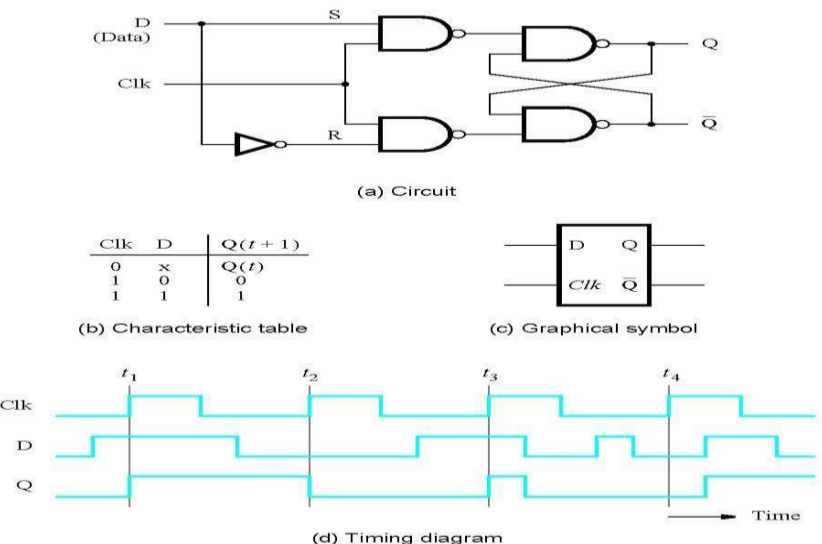
(a) Circuit

(b) Characteristic table

(c) Timing diagram

(d) Graphical symbol

Gated D Latch



(a) Circuit

(b) Characteristic table

(c) Graphical symbol

(d) Timing diagram

Setup and Hold Times

- Setup Time t_{su}

  The minimum time that the input signal must be stable prior to the edge of the clock signal.

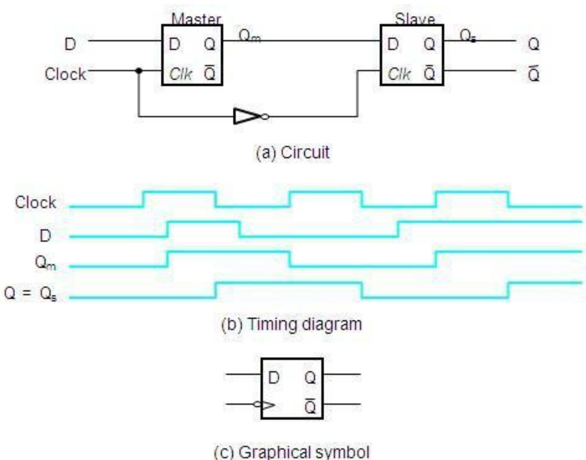
- Hold Time t_h

The minimum time that the input signal must be stable after the edge of the clock signal.

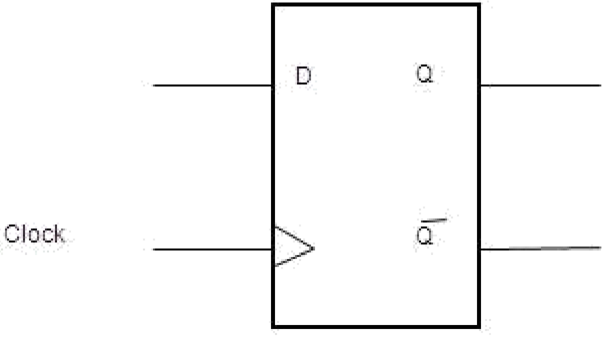
Flip-Flops

- A **flip-flop** is a storage element based on the gated latch principle
- It can have its output state changed only on the edge of the controlling clock signal
- We consider two types:
- **Edge-triggered flip-flop** is affected only by the input values present when the active edge of the clock occurs
- **Master-slave flip-flop** is built with two gated latches
- The master stage is active during half of the clock cycle, and the slave stage is active during the other half.
- The output value of the flip-flop changes on the edge of the clock that activates the transfer into the slave stage.

Master-Slave D Flip-Flop

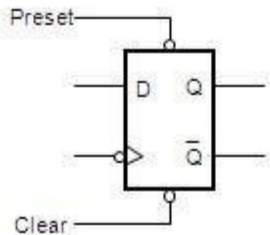


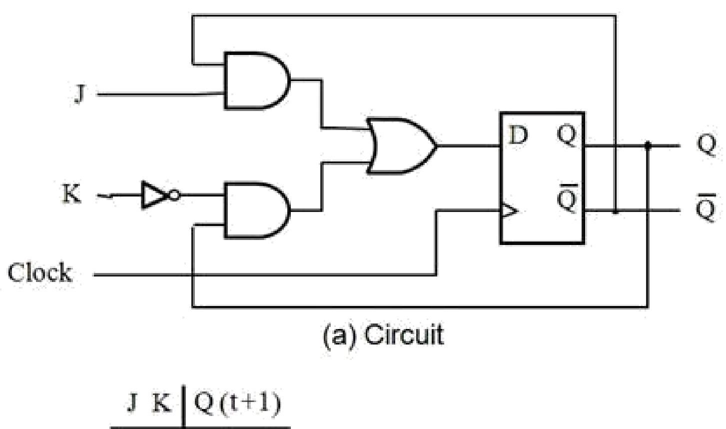
A Positive-Edge-Triggered D Flip-Flop



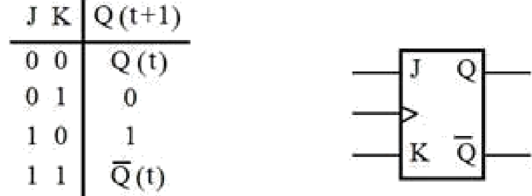
Graphical symbol

Master-Slave D Flip-Flop with *Clear* and *Preset*





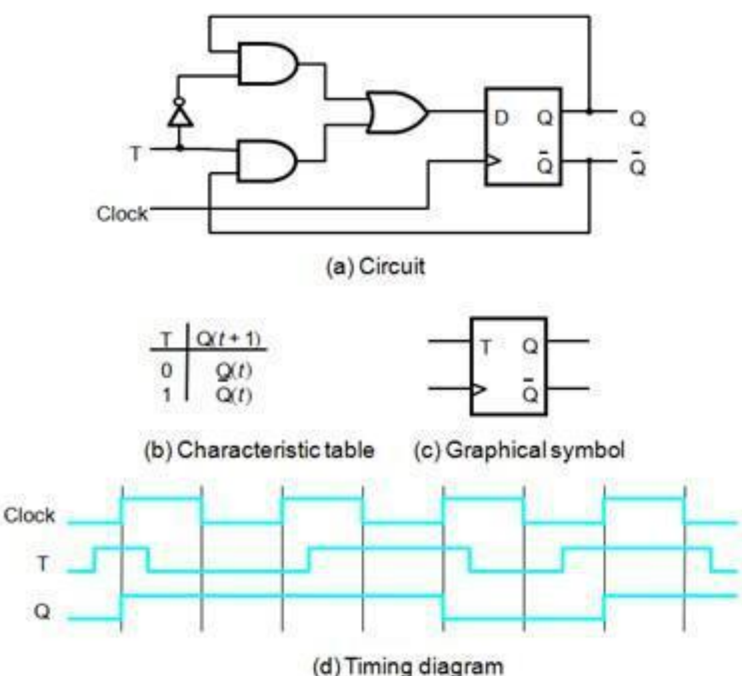
(a) Circuit



J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

(b) Characteristic table (c) Graphical symbol

T Flip-Flop



(a) Circuit

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

(b) Characteristic table (c) Graphical symbol

(d) Timing diagram

Excitation Tables

Previous State -> Present State	D
0 -> 0	0
0 -> 1	1
1 -> 0	0
1 -> 1	1

Previous State -> Present State	J	K
0 -> 0	0	X
0 -> 1	1	X
1 -> 0	X	1
1 -> 1	X	0

Previous State -> Present State	S	R
0 -> 0	0	X
0 -> 1	1	0
1 -> 0	0	1
1 -> 1	X	0

Previous State -> Present State	T
0 -> 0	0
0 -> 1	1
1 -> 0	1
1 -> 1	0

Conversions of flip-flops

Example: Use JK-FF to realize D-FF

1) Start transition table for D-FF

2) Create K-maps to express J and K as functions of inputs (D, Q)

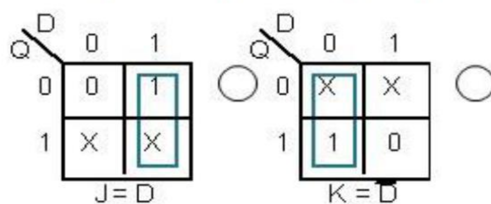
3) Fill in K-maps with appropriate values for J and K
to cause the same state transition as in the D-FF transition table

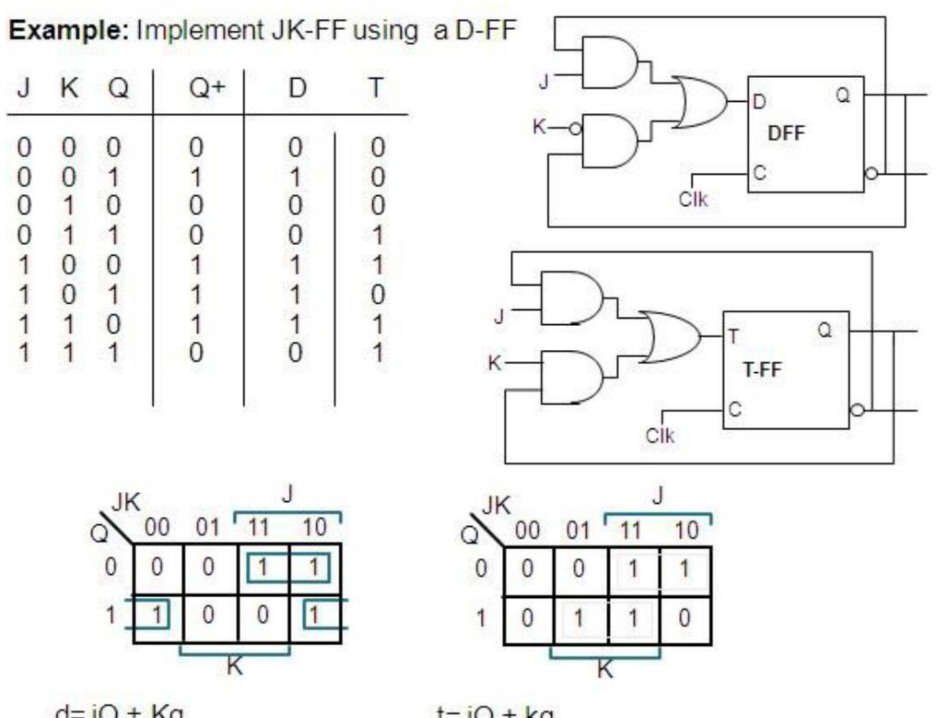
D	Q	Q^+	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

State-Table

Q	Q^+	R	S	J	K	T	D
0	0	X	0	0	X	0	0
0	1	0	1	1	X	1	1
1	0	1	0	X	1	1	0
1	1	0	X	X	0	0	1

e.g.
when D=Q=0, then $Q^+=0$
the same transition Q-->Q^+
is realize with J=0, K=X




Example: Implement JK-FF using a D-FF

J	K	Q	Q+	D	T
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

Sequential Circuit Design

- Steps in the design process for sequential circuits
- State Diagrams and State Tables Examples

- Steps in Design of a Sequential Circuit

- 1. Specification – A description of the sequential circuit. Should include a detailing of the inputs, the outputs, and the operation. Possibly assumes that you have knowledge of digital system basics.
- 2. Formulation: Generate a state diagram and/or a state table from the statement of the problem.
- 3. State Assignment: From a state table assign binary codes to the states.
- 4. Flip-flop Input Equation Generation: Select the type of flip-flop for the circuit and generate the needed input for the required state transitions
- 5. Output Equation Generation: Derive output logic equations for generation of the output from the inputs and current state.
- 6. Optimization: Optimize the input and output equations. Today, CAD systems are typically used for this in real systems.
- 7. Technology Mapping: Generate a logic diagram of the circuit using ANDs, ORs, Inverters, and F/Fs.
- 8. Verification: Use a HDL to verify the design.

Mealy and Moore

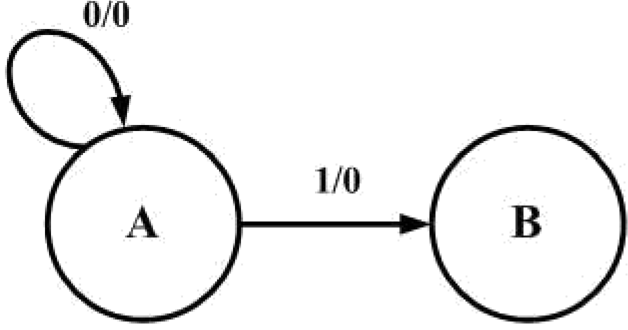
- Sequential machines are typically classified as either a Mealy machine or a Moore machine implementation.
- Moore machine: The outputs of the circuit depend only upon the current state of the circuit.
- Mealy machine: The outputs of the circuit depend upon both the current state of the circuit and the inputs.

An example to go through the steps

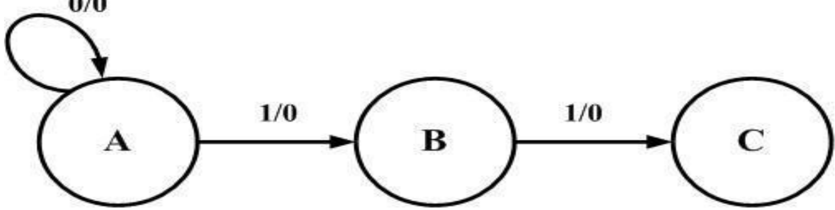
The specification: The circuit will have one input, X, and one output, Z. The output Z will be 0 except when the input sequence 1101 are the last 4 inputs received on X. In that case it will be a 1

Generation of a state diagram

- Create states and meaning for them.
  - State A – the last input was a 0 and previous inputs unknown. Can also be the reset state.
  - State B – the last input was a 1 and the previous input was a 0. The start of a new sequence possibly.
- Capture this in a state diagram

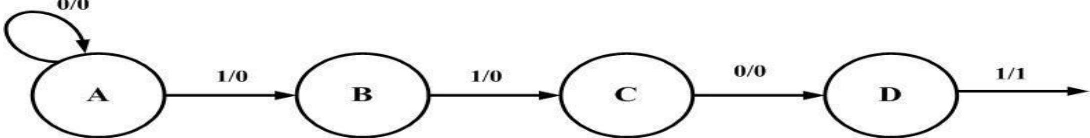


- Capture this in a state diagram oCircles represent the states oLines and arcs represent the transition between state.
  - The notation Input/Output on the line or arc specifies the input that causes this transition and the output for this change of state.
  - Add a state C – Have detected the input sequence 11 which is the start of the sequence

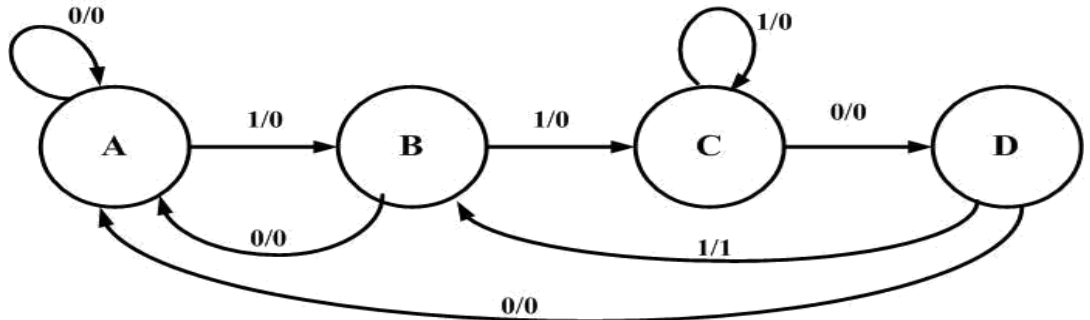


- Add a state D oState D – have detected the 3rd input in the start of a sequence, a 0, now having 110.

  From State D, if the next input is a 1 the sequence has been detected and a 1 is output.



- The previous diagram was incomplete.
- In each state the next input could be a 0 or a 1. This must be included



- The state table
- This can be done directly from the state diagram

Prresent State	Next State X =0	Next State X=1	Output X=0	Output X=1
A	A	B	0	0
B	A	C	0	0
C	D	C	0	0
D	A	B	0	1

- Now need to do a state assignment

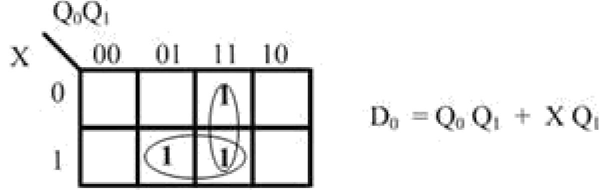
Select a state assignment

- Will select a gray encoding
- For this state A will be encoded 00, state B 01, state C 11 and state D 10

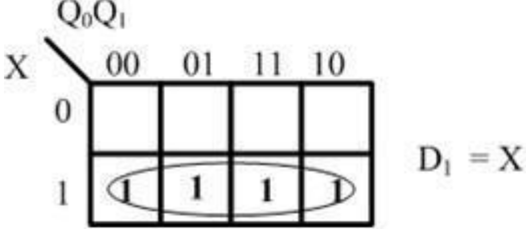
Prresent State	Next State X =0	Next State X=1	Output X=0	Output X=1
00	00	01	0	0
01	00	11	0	0
11	10	11	0	0
10	00	01	0	1

Flip-flop input equations

- Generate the equations for the flip-flop inputs
- Generate the D_0 equation



- Generate the D_1 equation



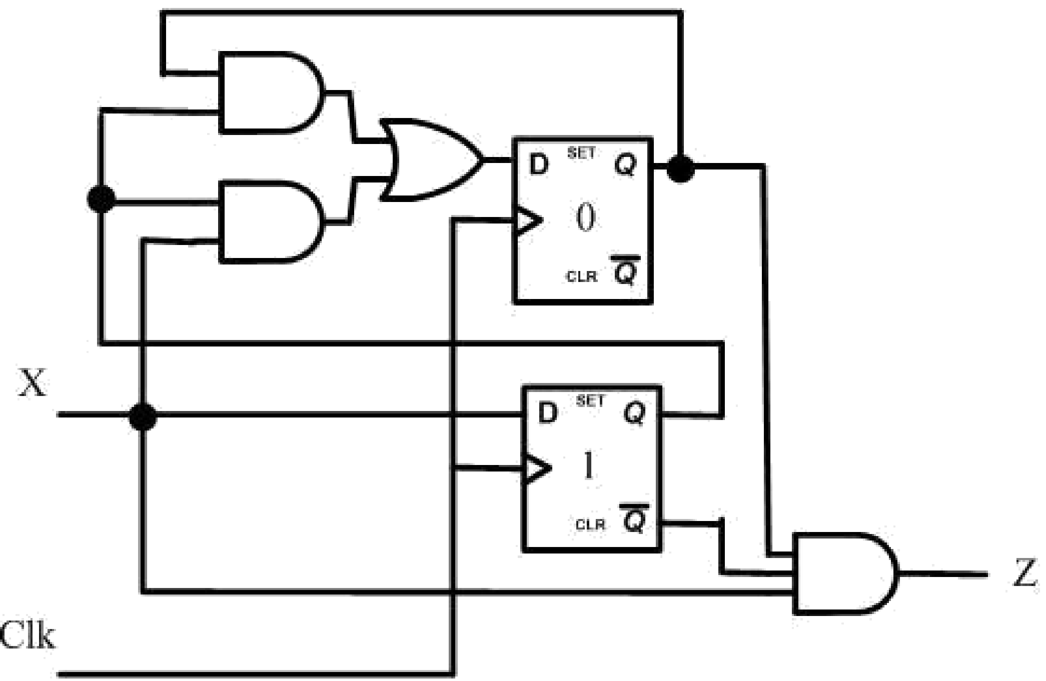
The output equation

- The next step is to generate the equation for the output Z and what is needed to generate it.
- Create a K-map from the truth table.

X \ Q_0Q_1	00	01	11	10
0				
1				1

$$Z = X\,Q_0\,\overline{Q_1}$$

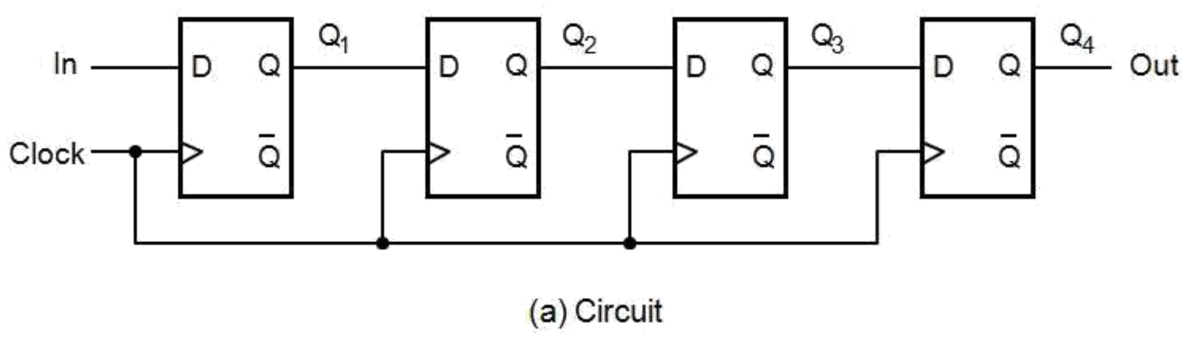
Now map to a circuit The circuit has 2 D type F/Fs



Registers and Counters

- An n-bit register is a cascade of n flip-flops and can store an n-bit binary data
- A counter can count occurrences of events and can generate timing intervals for control purposes

A Simple Shift Register

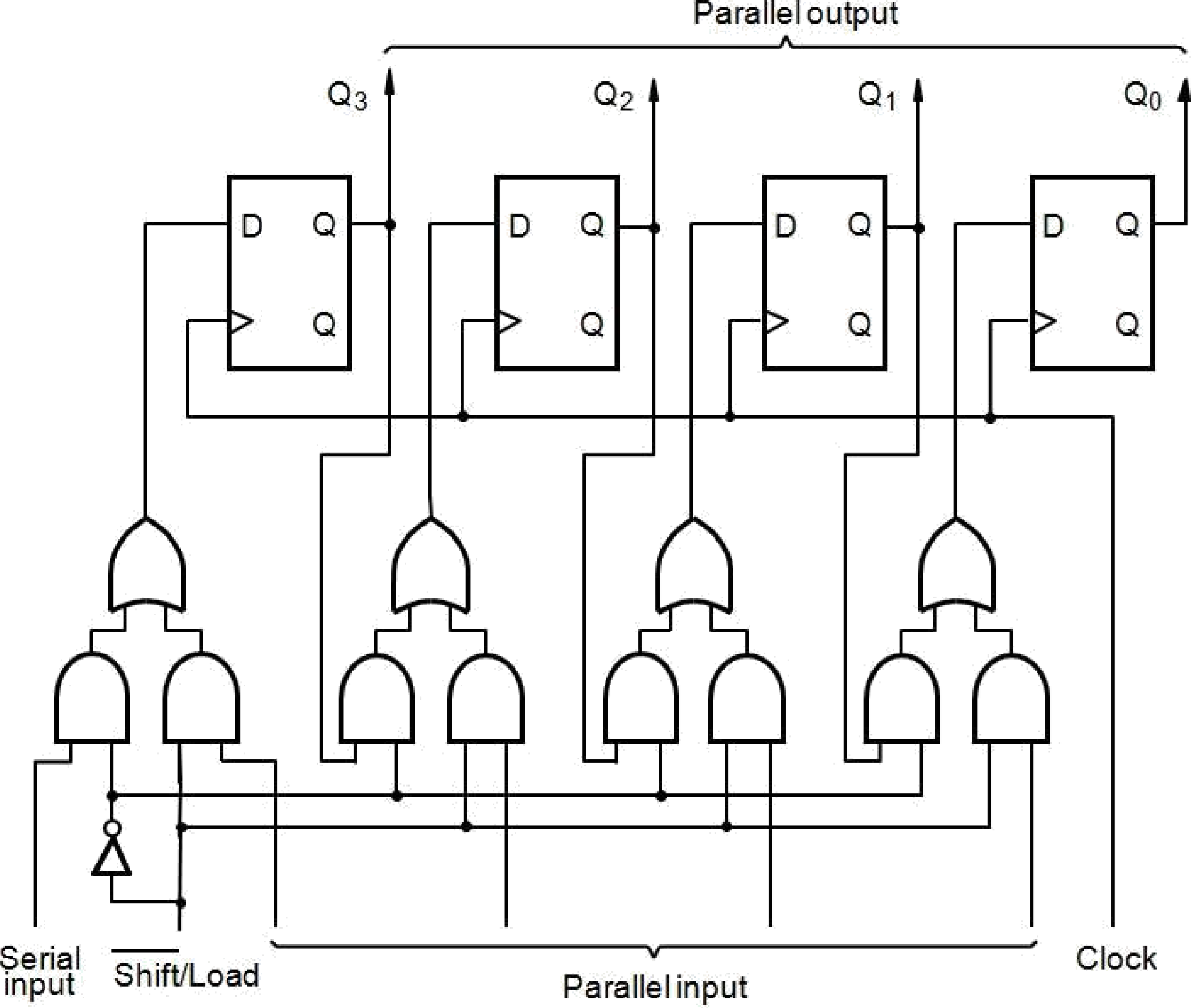


(a) Circuit

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) A sample sequence

Parallel-Access Shift Register



Counters

- Counters are a specific type of sequential circuit.
- Like registers, the state, or the flip-flop values themselves, serves as the “output.”
- The output value increases by one on each clock cycle.
- After the largest value, the output “wraps around” back to 0.
- Using two bits, we‟d get something like this:

Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Benefits of counters

- Counters can act as simple clocks to keep track of "time." •

  You may need to record how many times something has

  happened.

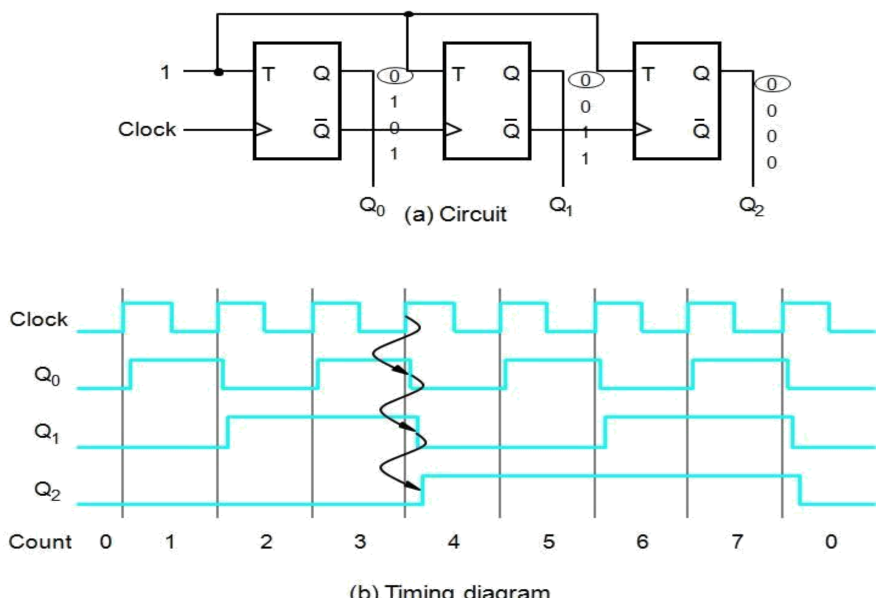
  - How many bits have been sent or received?
  - How many steps have been performed in some computation?

- All processors contain a program counter, or PC.

  - Programs consist of a list of instructions that are to be executed one after another (for the most part).
  - The PC keeps track of the instruction currently being executed.
  - The PC increments once on each clock cycle, and the next program instruction is then executed.

A Three-Bit Up-Counter

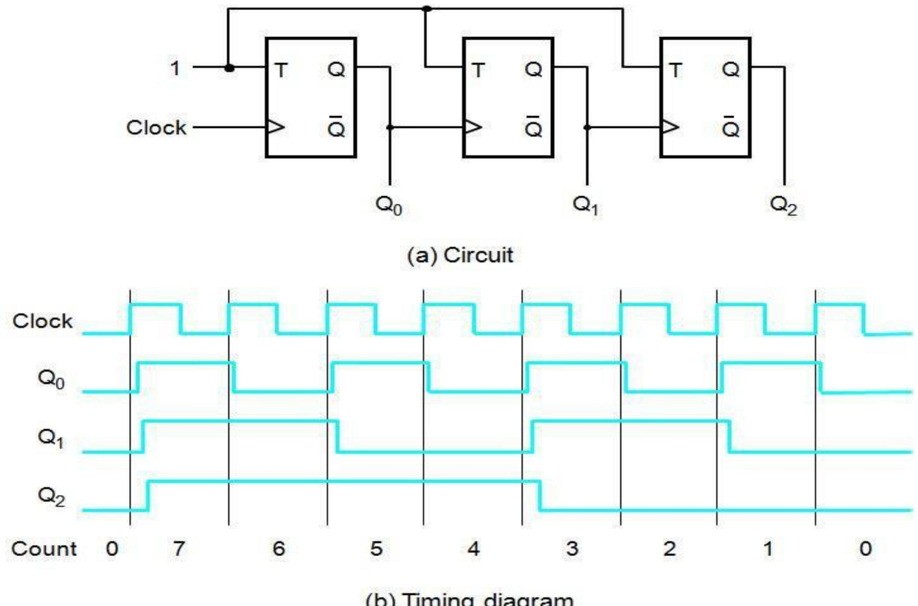
Q_1 is connected to clk, Q_2 and Q_3 are clocked by Q" of the preceding stage (hence called asynchronous or ripple counter



(a) Circuit

(b) Timing diagram

A Three-Bit Down-Counter



(a) Circuit

(b) Timing diagram