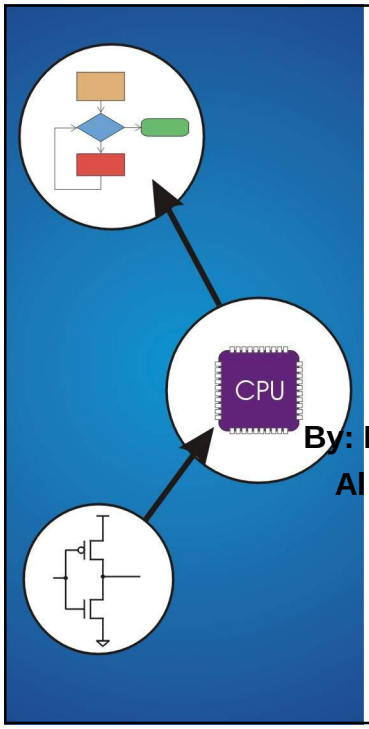


Digital Logic design 901220

By: Dr. Wa'el Al Qassas
Al Albayt university

Text Book : Digital Logic & Computer Design/Moris Mano

Course part

- **Theoretical Lectures: 3 hours weekly**
- **Practical LAB: 1 lab (2 hours) weekly**

Grading policy:

- **First Theoretical Exam: 15 points**
- **Second Theoretical Exam: 15 points**
- **Final Theoretical Exam : 40 points**
- **Mid Practical Exam: 10 points**
- **Final Practical Exam: 10 points**

Web sites.

Www.geocities.com/wael_it2003

www.aabu.edu.jo/it/~wael

http://web2.aabu.edu.jo:8080

http://web2.aabu.edu.jo:8080/tool/course_file/901220_lectures.pdf

www.aabu.edu.jo/~wael

Syllabus

- **Syllabus review & Introduction :1 hour**
- **Simple logic Circuits and manufacturing technology :1 hours**
- **Truth table and symbolic representation :1 hour**
- **Fundamental properties for Boolean algebra :1 hours**
- **Implementing Circuits form Truth table , practice : 2 hours**
- **XOR gate, Demorgan's Law : 1 hour**
- **Logical expression simplification using Fundamental properties, Demorgan , Practice : 1 hours.**
- **Karnaugh map (3 input, 4 input), SOP,POS, practice: 3 hour**
- **Tabulation method : 2 hours.**
- **Numbering systems, Binary numbers, Hexadecimal,…, real number implementation,: 3 hours**
- **First exam., Question solving & evaluation : 1 hour.**

Transistor: Building Block of Computers

Microprocessors contain millions of transistors

- Intel Pentium II: **7 million**
- Compaq Alpha 21264: **15 million**
- Intel Pentium III: **28 million**

Logically, each transistor acts as a switch

Combined to implement logic functions

- **AND, OR, NOT**

Combined to build higher-level structures

- **Adder, multiplexer, decoder, register, …**

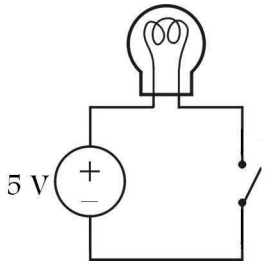
Simple Switch Circuit

Switch open:

- **No current through circuit**
- **Light is off**
- **V_{out} is + 5V**

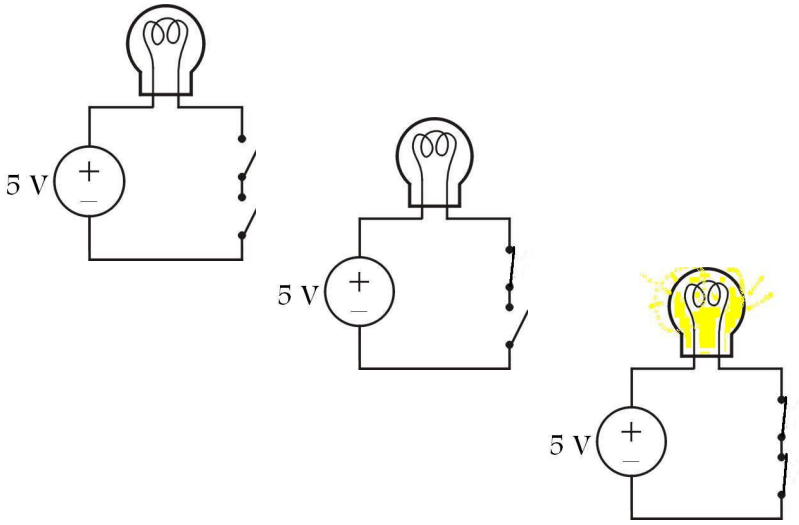
Switch closed:

- **Short circuit across switch**
- **Current flows**
- **Light is on**
- **V_{out} is 0V**



Switch-based circuits can easily represent two states: on/off, open/closed, voltage/no voltage.

LOGICAL AND:



7

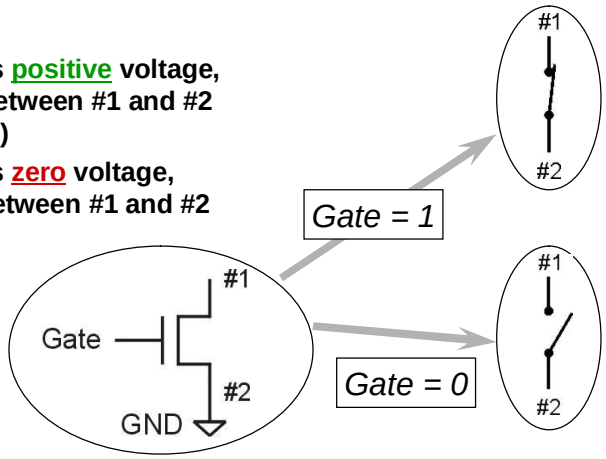
N-type MOS Transistor

MOS = Metal Oxide Semiconductor

- **two types: N-type and P-type**

N-type

- **when Gate has positive voltage, short circuit between #1 and #2 (switch closed)**
- **when Gate has zero voltage, open circuit between #1 and #2 (switch open)**



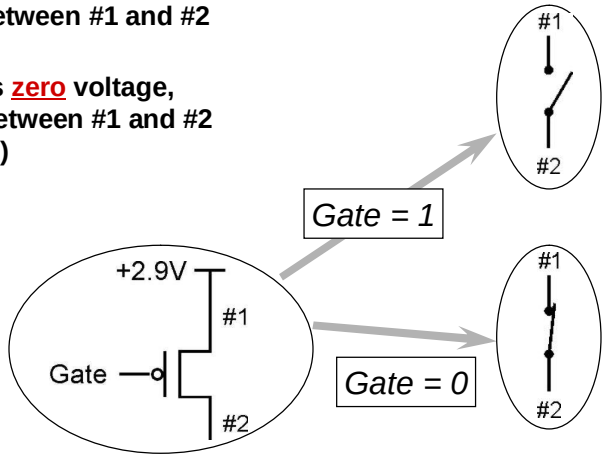
Terminal #2 must be connected to GND (0V).

8

P-type MOS Transistor

P-type is *complementary* to N-type

- **when Gate has positive voltage, open circuit between #1 and #2 (switch open)**
- **when Gate has zero voltage, short circuit between #1 and #2 (switch closed)**



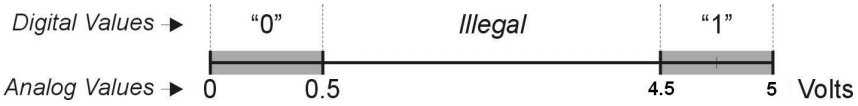
Terminal #1 must be connected to +2.9V.

Logic Gates

Use switch behavior of MOS transistors to implement logical functions: AND, OR, NOT.

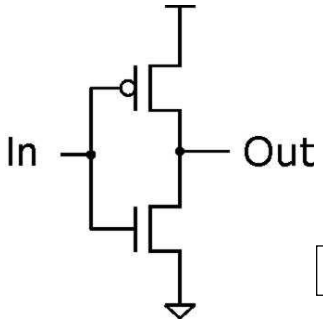
Digital symbols:

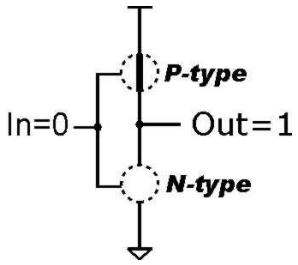
- **recall that we assign a range of analog voltages to each digital (logic) symbol**



- **assignment of voltage ranges depends on electrical properties of transistors being used**
  - **Ø typical values for "1": +5V, +3.3V, +2.9V**
  - **Ø from now on we'll use +5V**

Inverter (NOT Gate)

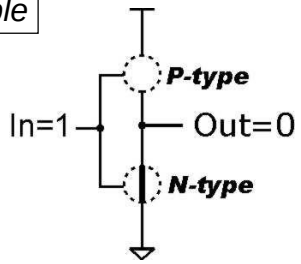




Truth table

In	Out
0 V	2.9 V
2.9 V	0 V

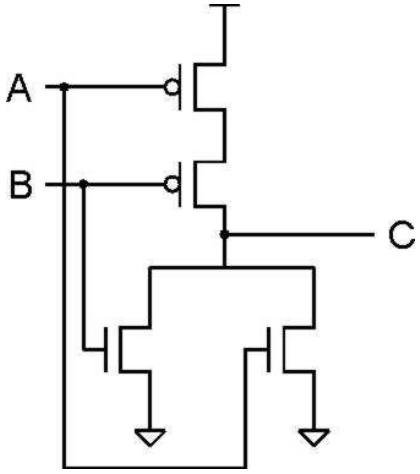
In	Out
0	1
1	0

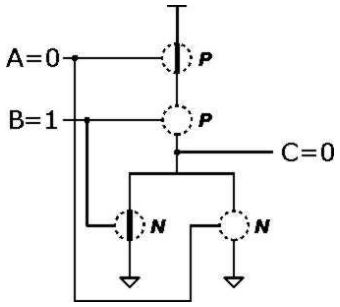


11

AABU/وائل قصاص

NOR Gate





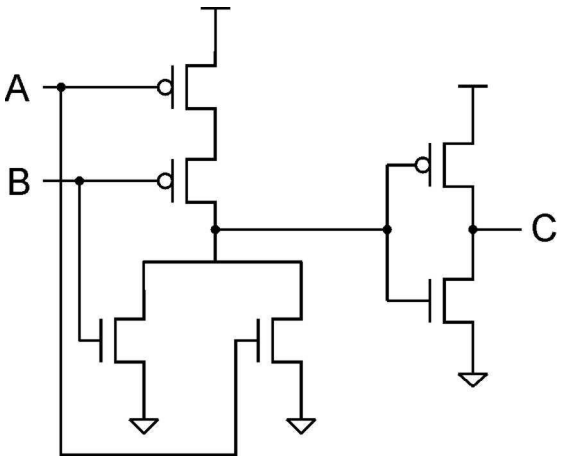
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

Note: Serial structure on top, parallel on bottom.

12

AABU/وائل قصاص

OR Gate

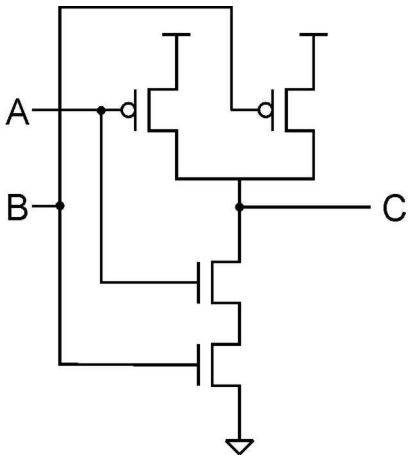


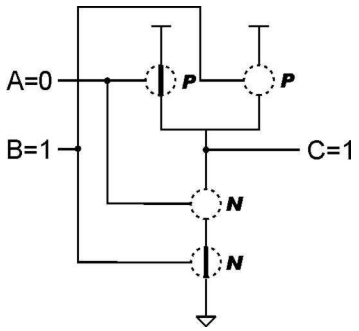
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

Add inverter to NOR.

13

NAND Gate (AND-NOT)



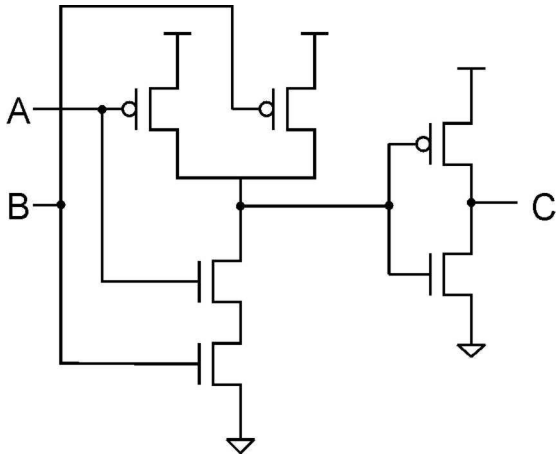


A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Note: Parallel structure on top, serial on bottom.

14

AND Gate



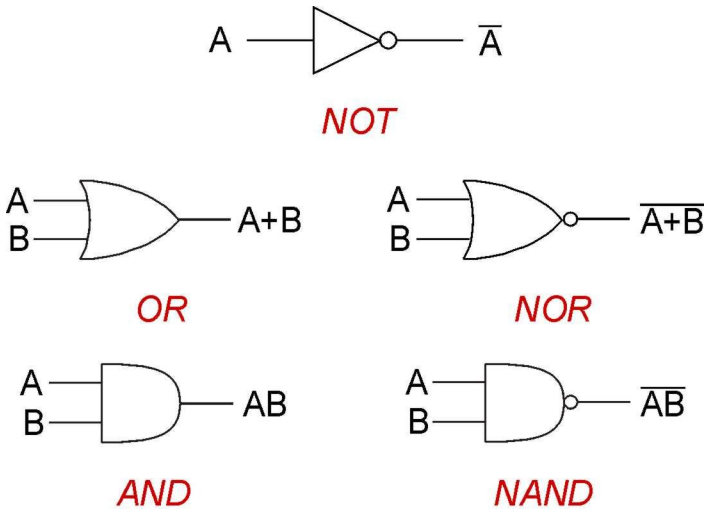
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Add inverter to NAND.

15

AABU/وائل قصاص

Basic Logic Gates



16

AABU/وائل قصاص

Fundamental Properties of boolean algebra:

Commutative:

- X + Y = Y + X
- X . Y = Y . X

Associative:

- (X + Y) + Z = X + (Y + Z)
- (X . Y) . Z = X . (Y . Z)

Identity:

- X + 0 = X
- X . 1 = X

Complement:

- X +(X') = 1
- X . (X') = 0

Distributive:

- X . (Y + Z) = (X . Y) + (X . Z)
- X + (Y . Z) = (X + Y) . (X + Z)

Absorption

- X + X.Y = X

17

X.X = X

X+X = X

X + 1= 1

X . 0 = 0

18

X.(Y+Z)=(X.Y)+(X.Z)

X Y Z	Y+Z	X.(Y+Z)	X.Y	X.Z	(X.Y)+(X.Z)
0 0 0	0	0	0	0	0
0 0 1	1	0	0	0	0
0 1 0	1	0	0	0	0
0 1 1	1	0	0	0	0
1 0 0	0	0	0	0	0
1 0 1	1	1	0	1	1
1 1 0	1	1	1	0	1
1 1 1	1	1	1	1	1

19

X+(Y.Z)=(X+Y).(X+Z)

X Y Z	Y.Z	X+(Y.Z)	X+Y	X+Z	(X+Y).(X+Z)
0 0 0	0	0	0	0	0
0 0 1	0	0	0	1	0
0 1 0	0	0	1	0	0
0 1 1	1	1	1	1	1
1 0 0	0	1	1	1	1
1 0 1	0	1	1	1	1
1 1 0	0	1	1	1	1
1 1 1	1	1	1	1	1

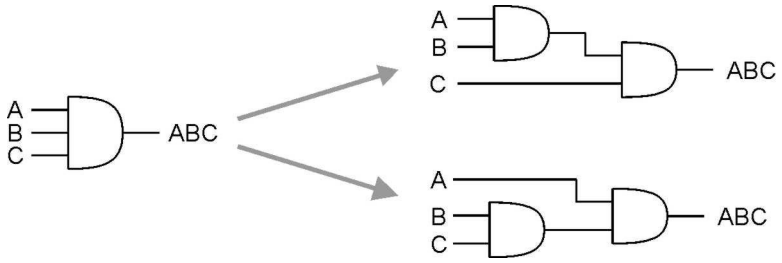
20

More than 2 Inputs?

AND/OR can take any number of inputs.

- **AND = 1 if all inputs are 1.**
- **OR = 1 if any input is 1.**
- **Similar for NAND/NOR.**

Can implement with multiple two-input gates, or with single CMOS circuit.



21

Practice

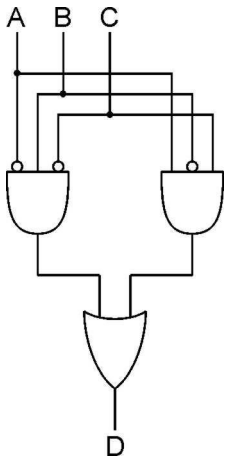
Implement a 3-input NOR gate with CMOS.

22

Logical Completeness

Can implement ANY truth table with AND, OR, NOT.

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



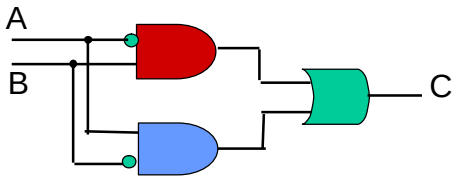
1. AND combinations that yield a "1" in the truth table.

2. OR the results of the AND gates.

Practice

Implement the following truth table.

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



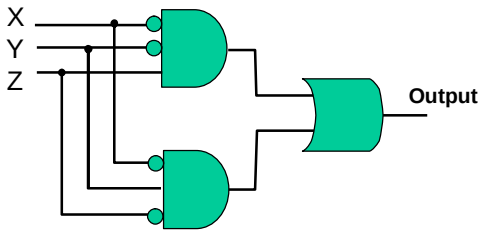
Another example:

We want to build a circuit that has 3 binary inputs.

This CKT is On if the inputs are X'Y'Z or X'YZ' .

Build the Truth table, then Draw the Ckt

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

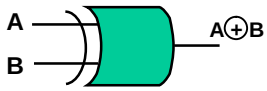


25

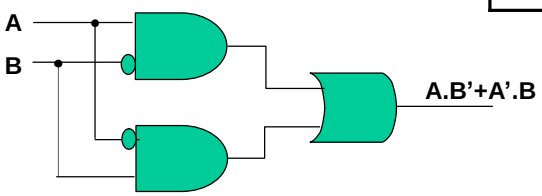
XOR gate:

A XOR B= A.B'+A'.B

A⊕B



A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	0

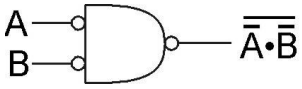


26

DeMorgan's Law

Converting AND to OR (with some help from NOT)

Consider the following gate:



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

Same as A+B!

To convert AND to OR (or vice versa), invert inputs and output.

27

AABU/وائل قصاص

DeMorgan Law:

ØA+B= (A' . B')'

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

OR Truth table

ØA.B=(A'+B')'

A	B	A'	B'	A'+B'	(A'+B')'
0	0	1	1	1	0
0	1	1	0	1	0
1	0	0	1	1	0
1	1	0	0	0	1

AND Truth table

28

AABU/وائل قصاص

Build the following logical expression using AND, Not Gates only:

F=X.Y+Z'

=((X.Y)'.Z'')'

=((X.Y)'.Z)'

Another example:

F= XYZ+Y'Z+XZ'

= ((XYZ)'.(Y'Z)'.(XZ')')'

29

Simplify the following boolean expression

F= (A'BC⊕C + A + D)'

= (A'BCC' + (A'BC)'C +A+D)'

= (0 + (A'BC)'C +A+D)'

= (0 + (A+B'+C')C+A+D)'

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

= (AC+B'C+ 0 + A +D)'

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

= (A + B'C + D)'

= A' (B'C)' D'

= A' D'(B+C')

= A'BD'+A'C'D'

30

Simplification using boolean algebra:

We have the following truth table for a logical circuit and we want to implement this using the minimum number of gates:

SOP

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

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

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

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

= A'C+AB'+AC'(B+B') ;we can reuse AB'C'

=A'C+AB'+AC'

31

Another example:

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

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

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

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

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

=A'C+AC+AB

= C(A+A')+AB

= C+AB

32

Karnaugh Maps

Karnaugh map : is a representation for the truth table in a graphical way, which makes the simplification of any boolean function easier.

For 3 input boolean function the Karnaugh map will be as follows :

A \ BC	00	01	11	10
0	A'B'C' 000	A'B'C 001	A'BC 011	A'BC' 010
1	AB'C' 100	AB'C 101	ABC 111	ABC' 110

As we can see, each cell represent one raw of the truth table

Next step is to fill the map using the truth table output.

Simplification using Karnaugh:

Let us resolve the previous examples using karnaugh map:

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

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

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

Resolve the same example in another way:

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

F=A'C+AB'+AC'

35

Another example

	00	01	11	10
0	0	1	1	0
1	1	1	1	1

36

Convert from logical expression to minterms

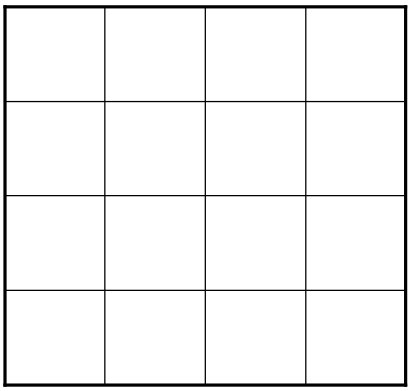
What are the minterms that represent the following expression assuming that we have 3 inputs A,B, and C

F=A+A'C

F=

Karnaugh map for 4 input boolean function

AB \ CD	00	01	11	10
00	0 0000	1 0001	3 0011	2 0010
01	4 0100	5 0101	7 0111	6 0110
11	12 1100	13 1101	15 1111	14 1110
10	8 1000	9 1001	11 1011	10 1010



Assume we have the boolean function F with 4 inputs:
F(A,B,C,D)= ∑ (0,1 ,4, 6, 8,11,13, 15)

- **Make the truth table for this function**
- **Write the boolean equation for this function (Before simplification)**
- **Simplify this function using karnaugh map technique**
- **Draw the simplified equation.**

	00	01	11	10
00	1	1		
01	1			1
11		1	1	
10	1		1	

Minterms & Maxterms

To understand the relation between Minterms and Maxterms let is see the following example:

F=Σ(1,3,5,6,7)

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

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

We know that F''=F

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

F=(A'B'C')' .(A'BC')' . (AB'C')'

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

F=Π(0,2,4)

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

41

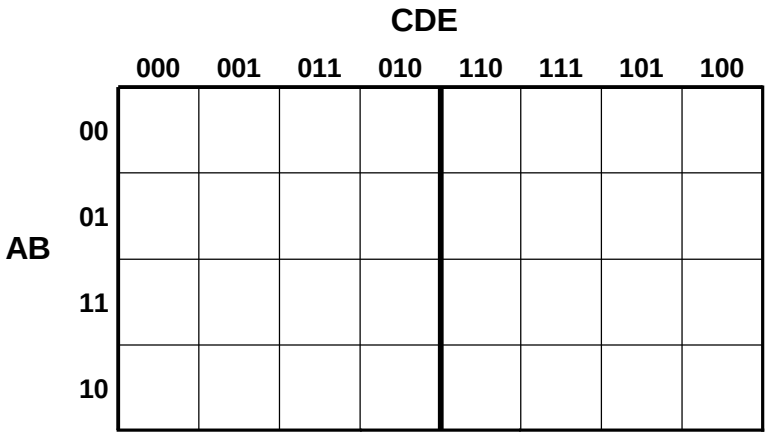
Minterms & Maxterms

For a 3 binary variables

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

42

5 variables Karnaugh map



43

Tabulation method

- **Last lecture we noticed the difficulty of simplifying 5 variable equations using karnaugh Map.**
- **Another method is used to simplify boolean equations.**
- **Tabulation method (Quine-McCluskey), or prime implicants :-**
  - **Its suitable for machine computation**
  - **<u>Uses two phases:</u>**
    i. **Finding Prime implicants**
    ii. **Selecting Prime implicants**

44

Determining Prime implicants

Example 1 : F(w,x,y,z)=Σ(0,1,2,8,10,11,14,15)

First we list the minterms in groups depending on the number of ones in each minterm

0 0000
~~~~~~~~~~
1 0001
2 0010
8 1000
~~~~~~~~~
10 1010
~~~~~~~~~
11 1011
14 1110
~~~~~~~~~
15 1111

Then look for any two minterms that differ on one variable only , between any adjacent groups

0 0000	**0,1**	**000-**	**0,2,8,10**	**-0-0**
1 0001	**0,2**	**00-0**	**0,8,2,10**	**-0-0**
2 0010	**0,8**	**-000**	**10,11,14,15**	**1-1-**
8 1000	**2,10**	**-010**	**10,14,11,15**	**1-1-**
10 1010	**8,10**	**10-0**		
11 1011	**10,11**	**101-**		
14 1110	**10,14**	**1-10**		
15 1111	**11,15**	**1-11**		
	14,15	**111-**		

The prime implicants are : **w'x'y' , x'z' , wy**

Next step is to select needed prime implicants.

	0	1	2	8	10	11	14	15
W'X'Y'	V	V						
X'Z'	V		V	V	V			
WY					V	V	V	V

Example 2

Simplify the following SOP using tabulation method.
F(W,X,Y,Z)=Σ(1,4,6,7,8,9,10,11,15)

Remember : first Find Prime implicants
then select the needed prime implicants.

Determining the prime implicants

1	0001	1,9	-001	8,9,10,11	10- -
4	0100	4,6	01-0	8,10,9,11	10- -
8	1000	8,9	100-	~~~~~~~~~~~~	
~~		8,10	10-0		
6	0110	~~~~~~~~			
9	1001	6,7	011-		
10	1010	9,11	10-1		
~~		10,11	101-		
7	0111	~~~~~~~~~			
11	1011	7,15	-111		
~~		11,15	1-11		
15	1111				

X'Y'Z, W'XZ', W'XY, XYZ, WYZ, WX'

- **Selection for the needed prime implicants**

	1	4	6	7	8	9	10	11	15
X'Y'Z	V					V			
W'XZ'		V	V						
W'XY			V	V					
XYZ				V					V
WYZ								V	V
WX'					V	V	V	V	

IC's characteristics (From Ch 1):

- **Fan-out: the number of standard loads that the output of a gate can drive without impairing its normal operation (20 to 50 gates)**
- **Power dissipation : the supplied power required to operate the gate (in mW)**
- **Propagation delay: The average transition delay time for a signal to propagate from input to output when the binary signals change in value (in ns)**

Binary numbers (Chapter 1)

- **Computers uses Binary system.**
- **We need to represent different numbering types use the Binary system.**
- **Remember the conversion between Binary & Decimal, this was used to represent positive integer numbers only.**
- **But , what about negative numbers.**
- **Three different methods were used to represent negative integer numbers:**
  - **Sign magnitude**
  - **Ones Complement**
  - **Twos complement**

Binary numbers

Negative numbers (Sign magnitude)

This technique uses additional binary digit to represent the sign, 0 to represent positive, 1 to represent negative.

- **Ex.:**

5 = 0101

-5 = 1101

Also

5 = 00000101

-5= 10000101

www.microcode.com **Circuit maker**

Negative numbers (1's Complement)

- 1's comp is another method used to represent negative numbers.
- In this method we invert every bit in the positive number in order to represent its negative.
- Ex.:

  9 = 01001

  -9= 10110

- Again remember that we use additional digit to represent the sign
- We may represent 9 as 00001001 ; zeros on left have no value

-9 in 1's comp will be 11110110

Negative numbers (2's complement)

- This is the method which is used in most computers to represent integer numbers nowadays.
- Remember always to represent a positive number using any of the previous methods is the same, all what is needed is a 0 on the left to show that the number is positive
- To represent a negative number in 2's Comp , first we find the 1's Comp, then add 1 to the result
- Ex:

How we represent -9 in 2's comp

1- 9 in binary= 01001

2- invert = 10110

3 add 1 = 10111; -9 in 2's Comp.

-22

22 = 010110

In 1's= 101001

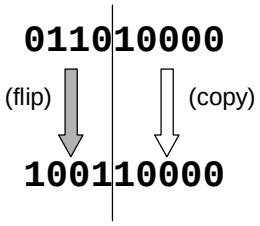
In 2's= 101010 -22 in 2's comp

Two's Complement Shortcut

To take the two's complement of a number:

- **copy bits from right to left until (and including) the first "1"**
- **flip remaining bits to the left**

```
 011010000
 100101111   (1's comp)
+        1
 100110000
```



Operations: Arithmetic and Logical

Recall:
a data type includes *representation* and *operations*.

We now have a good representation for signed integers, so let's look at some arithmetic operations:

- **Addition**
- **Subtraction**
- **Sign Extension**

We'll also look at overflow conditions for addition.

Multiplication, division, etc., can be built from these basic operations.

Logical operations are also useful:

- **AND**
- **OR**
- **NOT**

Addition

As we've discussed, 2's comp. addition is just binary addition.

- **assume all integers have the same number of bits**
- **ignore carry out**
- **for now, assume that sum fits in n-bit 2's comp. representation**

	01101000	(104)		**11110110**	(-10)
+	**11110000**	(-16)	+		(-9)
	01011000	(88)			(-19)

Assuming 8-bit 2's complement numbers.

Subtraction

Negate subtrahend (2nd no.) and add.

- **assume all integers have the same number of bits**
- **ignore carry out**
- **for now, assume that difference fits in n-bit 2's comp. representation**

01101000	(104)	**11110110**	(-10)
- 00010000	(16)	**-**	(-9)
01101000	(104)	**11110110**	(-10)
+ 11110000	(-16)	**+**	(9)
01011000	(88)		(-1)

Assuming 8-bit 2's complement numbers.

61

Sign Extension

To add two numbers, we must represent them with the same number of bits.

If we just pad with zeroes on the left:

4-bit		**8-bit**	
0100	(4)	**00000100**	(still 4)
1100	(-4)	**00001100**	(12, not -4)

Instead, replicate the MS bit -- the sign bit:

4-bit		**8-bit**	
0100	(4)	**00000100**	(still 4)
1100	(-4)	**11111100**	(still -4)

62

Overflow

If operands are too big, then sum cannot be represented as an *n*-bit 2's comp number.

01000	(8)	11000	(-8)
+ 01001	(9)	+ 10111	(-9)
10001	(-15)	01111	(+15)

We have overflow if:

- **signs of both operands are the same, and**
- **sign of sum is different.**

Another test -- easy for hardware:

- **carry into MS bit does not equal carry out**

Build a Ckt that discovers the overflow

Inputs Sa: Sign for the first number
Sb: Sign for the second number
Ss: Sign for the answer number

Sa Sb Ss	Overflow
0 0 0	
0 0 1	
0 1 0	
0 1 1	
1 0 0	
1 0 1	
1 1 0	
1 1 1	

Real numbers (outside the text book)

- All what we discussed before was about integers.
- What about real numbers (ex.: 6.125)
- To represent real numbers we take first the integer part and convert is as we learned before, and then take the fraction part and convert it using the following algorithm
  - 1- multiply fraction by 2
  - 2- take the integer of the result
  - 3- Repeat 1 and 2 until the fraction is zero, or until u reach get enough digits.
- Ex. : 6.125

6= 110

0.125x2 = 0.25

0.25x2 = 0.5

0.5x2 = 1.0

6.125= 110.001

Another example:

Convert 9.2 to binary.

9=1001

0.2x2 = 0.4

0.4x2 = 0.8

0.8x2 = 1.6 ; take the fraction only

0.6x2 = 1.2

0.2x2 = 0.4 ; let us stop here, 5 digits after the point

The binary equivalent will be: 1001.00110

Convert from binary

Again let us take the previous results

$2^2\ 2^1\ 2^0\ 2^{-1}\ 2^{-2}\ 2^{-3}$

1 1 0 . 0 0 1

= 4+2+.125=6.125

Ex. 2:

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

1 0 0 1. 0 0 1 1 0

= 8+1+ .125+.0625 = 9.1875

Why not 9.2 !??

Floating numbers

Nowadays numbers with decimal point are represended in the computer using IEEE 754 float number format.

This format has to subtypes :

- **Float: 32 bit number can represent up to 10^{35} .**
- **Double: 64 bit number can represent up to 10^{350} with more number of digits after the point.**

We will not cover this topic here.

It will be covered in Computer architecture course.

Hexadecimal

- As we noticed to read a long binary number is confusing
- So another numbering system was invented (base 16)
- We know base 10 ,base 2, and now base 16
- Numbers in (base 16) are : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
- There is a direct relation between base 2 and base 16, 2^4=16 , so the conversion from binary to hexadecimal is quite easy.
- Each 4 binary digits are converted to one hexadecimal digit.

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

Examples on Hexadecimal

01010001=51h, 0x51,$51, 51_{16}

1110010 =$72

10101110=$AE

From Hex to decimal

$ff= 1111 1111 =$255_{10}$

Binary Codes

Binary coding is different from binary conversion

BCD code : Binary coded decimal, is used to represent each decimal digit to a 4 bit binary number,

Ex: 13 = 0001 0011 in BCD

Remember 13 = 1101 in binary

Ex: 49 = 0100 1001 in BCD

Remember that 49 = 0011 0001 in binary.

Excess-3 code

In Excess-3 0 = 0011, 1=0100, 2=0101 ,… , 9=1100

This means that we add 3 to the number then convert it to binary

We mainly use this to avoid having zeros in transmission lines.

Other coding methods (See Page 17).

Error detection :

Many techniques are used in order to detect if an error has occurred in the data transmitted or stored, one of these is the parity check.

The idea of the parity check is to add an extra bit to the binary number, the value of this bit depends on the number of ones in the binary number.

The parity bit is generated on transmitting end, and checked at receiving end

A parity check involves appending a bit that makes the total number of binary 1 digits in a character or word , either odd (for *odd parity*) or even (for *even parity*).
Examples:
Even parity 0000 0, 0001 1, 1111 0

Odd parity:0000 1, 0001 0, 1111 1

73

Alphanumeric Codes

To represent numbers and letters we need some coding method.

First we need to know the number of symbols (letters, numbers, or other symbols)

ASCII: American standard code for information Interchange, is one of the commonly used codings

It uses 7 bits , it can represent 127 different symbols

Ex: A = 100 0001,
a = 110 0001
1 = 011 0001
space = 010 0000
WAEL = 101 0111 100 0001 100 0101 100 1100

74

EBCDIC: Extended BCD Interchange, is an 8 bit coding

Ex: A= 1100 0001

1= 1111 0001

space= 0100 0000

Summary

MOS transistors are used as switches to implement logic functions.

- **N-type: connect to GND, turn on (with 1) to pull down to 0**
- **P-type: connect to +2.9V, turn on (with 0) to pull up to 1**

Basic gates: NOT, NOR, NAND

- **Logic functions are usually expressed with AND, OR, and NOT**

Properties of logic gates

- **Completeness**
  - **Ø can implement any truth table with AND, OR, NOT**
- **DeMorgan's Law**
  - **Ø convert AND to OR by inverting inputs and output**

77

First exam , Tus. 31-10-2006 from 2:00 to 2:50

78

Building Functions from Logic Gates

We've already seen how to implement truth tables using AND, OR, and NOT -- an example of *combinational logic*.

Combinational Logic Circuit

- **output depends only on the current inputs**
- **stateless**

Sequential Logic Circuit

- **output depends on the sequence of inputs (previous inputs and present inputs)**
- **stores information (state) from past inputs**

Chapter 4.
Combinational Circuits

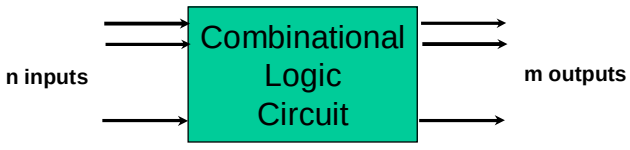
A combinational circuit consists of :

1- Input variables.

2- Logic gates

3- Output variables

Logic gates accepts signals (Binary signals) from inputs and generate signals to the outputs.



Design Procedure

To design a combinational circuit, the following steps are used:

1. The problems in stated

2. The number of inputs and outputs are determined

3. Assigning letter symbols to each input and output

4. Building the Truth table, which defines the relationship between inputs and outputs, (and the don't care conditions).

5. Simplifying the truth table.

6. Drawing the logic diagram.

Adders

Let us implement what we learned in the previous slide and build a combinational circuit that adds two binary numbers.

1. State the problem: Build a circuit that can add two binary numbers (HALF ADDER)

0+0= 0

0+1=1

1+0=1

1+1=10

2. Number of inputs is two , Number of outputs derived is also two.

3. Let us name the inputs as X, Y, and the outputs as S for Sum, and C for Carry_out.

4. Truth table

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

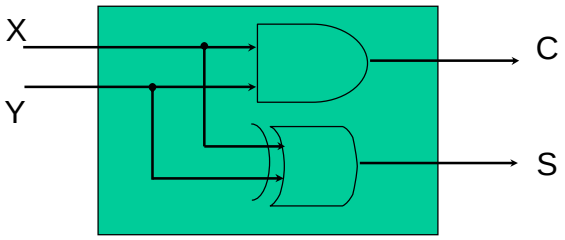
5. Simplification:

C = x.y (No more simplification)

S = x.y' + x'y

= x⊕y

6. Draw logic diagram



Full ADDER: Build a circuit that can Add three binary numbers.

Subtractors

Half Subtractor

Full Subtractor

X- Y – Z	BD
0 – 0 – 0 =	0 0
0 – 0 - 1 =	1 1
0 – 1 – 1 =	1 0
1 - 1 - 0 =	0 0
1 -1 – 1 =	1 1

Code Conversion

We studied before different coding techniques for numbers or alphanumarics, such as BCD, Excess_3 ,…

Let us build a combinational circuit that can convert from BCD to Excess_3.

1. **BCD code and Excess_3 code are used to represent a decimal digit in binary, each code represnts the decimal digit as 4 bits.**
2. **Let us name the BCD input variables as A,B,C,D , and the output Excess_3 variables as w,x,y,z.**

Truth table

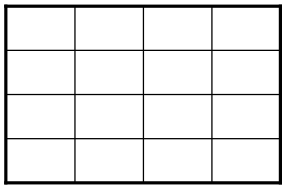
Decimal	BCD ABCD	Ex_3 wxyz
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

We also have don't care conditions in this example, 1010, 1011,1100,1101, 1110,1111 are not accepted codes in BCD, which so can be considers as don't care.

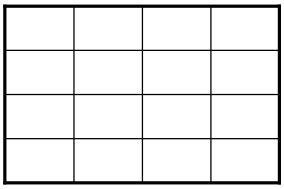
d(A,B,C,D)=Σ(10,11,12,13,14,15) for all outputs.

89

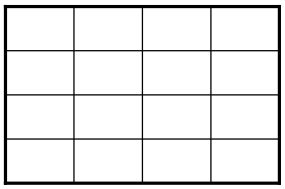
Simplify the function for each output.



w



x



y

z

90

Ex-3 to BCD

4.6 Analysis procedure

Analysis of combinational circuit to find the boolean function from the logic diagram.

1. **Label the output of all gates , that are connected to input variables**
2. **Label other gates, and find the function of the these gates**
3. **Repeat the process until you reach the output**

Example

Derivation of truth table

To derive the truth table for a given logical diagram

- **First label the output of each gate**
- **Write the function of each label, using the labels of the previous gates**
- **Then after putting all possible input combinations , start to find the output of each level going from input side.**
- **Repeat the for the next levels until you reach the output.**

Q 4.14

BCD to 7 segment display.

1. **Build the truth table for each segment**
2. **Minimize the function of each segment, (use the don't care conditions)**

MSI and LSI

MSI : Medium Scale Integrated circuit, < 50 gate in one IC

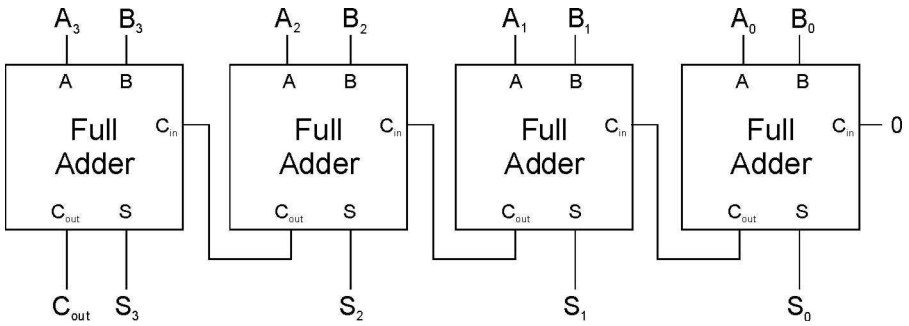
LSI : Large Scale Integrated circuit > 50 gate in one IC

VLSI : Very Large Scale Integrated circuit.

Ex.

- **4 bit full adder**
- **BCD to 7 segment decoder**

Four-bit Adder



Examples using MSI circuits

- **Using 4 bit full adder as a BCD to Ex-3 converter**
- **Using the 4 bit full adder as an Ex_3 to BCD converter**
- **Using 4 bit full adder as a 4 bit subtractor**
- **Using 4 bit full adder as an adder and a subtractor**

Building a BCD ADDER

The BCD adder should have 9 inputs, and 5 outputs.

Trying to build a truth table for this is not a good idea

(number of possible states for inputs is 2^9 =512.

We also will have many don't care states.

The other solution is to use 4bit full adders with a small combinational circuit

Things to take into consideration

1. **If the sum of the 2 BCD numbers is <= 9 , no problem in the result**
2. **If the sum was greater than 9, it will not be in the BCD format, we need to correct the result**

99

0101 0101

0100 0111

1001=9 1100 = 12 ,

but we write 12 in BCD as 0001 0010

Another case (worst one)

1001

1001

10010 =18

Again , in BCD 18 is 0001 1000

100

The solution:

If the resulting sum is >9 we need to correct the sum by adding 6= 0110

We have two cases

1) **The answer between 1010 and 1111 , with carry =0**
2) **The carry out is 1 , which means that the sum is > 15**

4 bit Magnitude comparator

We need to build a combinational circuit that compares two numbers ,

The circuit will have 3 outputs

A>B

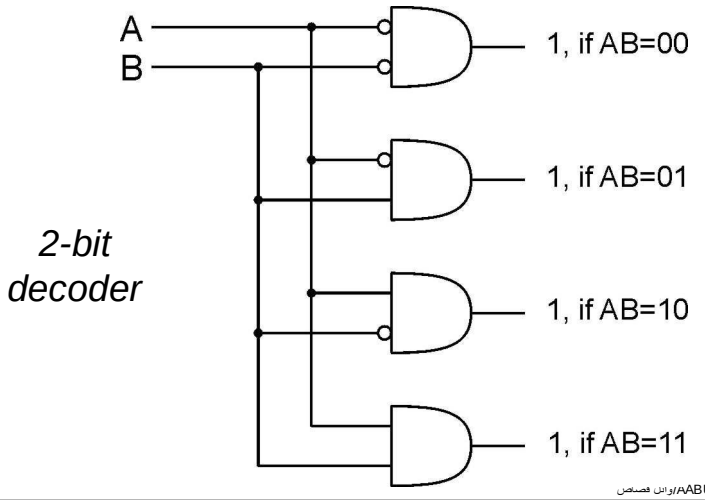
A<B

A=B

Decoder

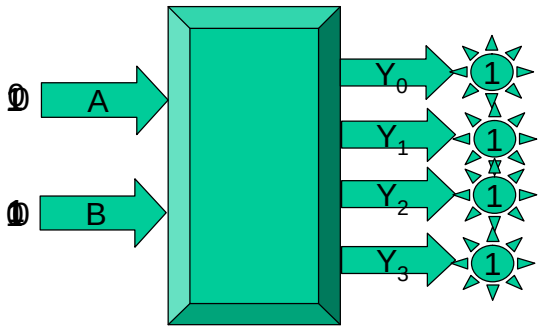
***n* inputs, 2^n outputs**

- **exactly one output is 1 for each possible input pattern**



Decoder

***n* inputs, 2^n outputs**

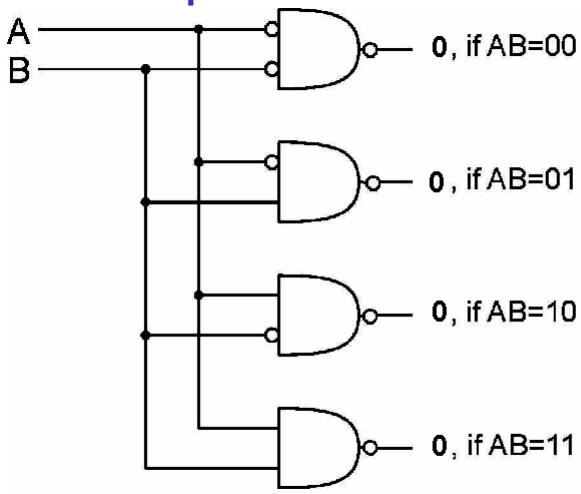


3x8 Decoder

Implement combinational circuits using a decoder , OR gates

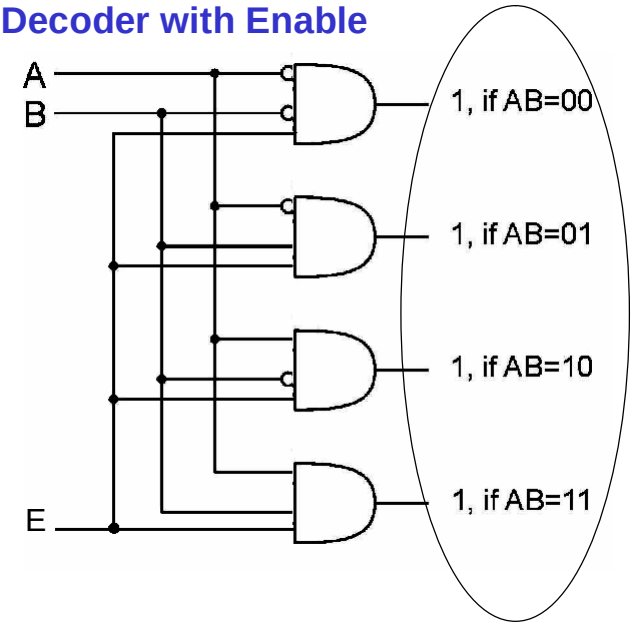
Ex. Full adder, using 3x8 decoder

Inverted output decoder



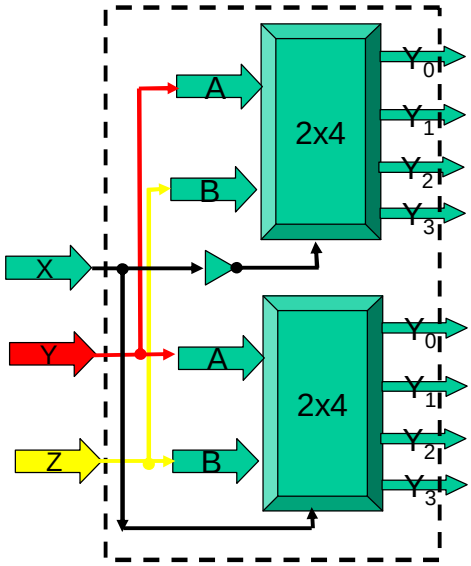
Why we build a decoder using NAND gates?

Decoder with Enable



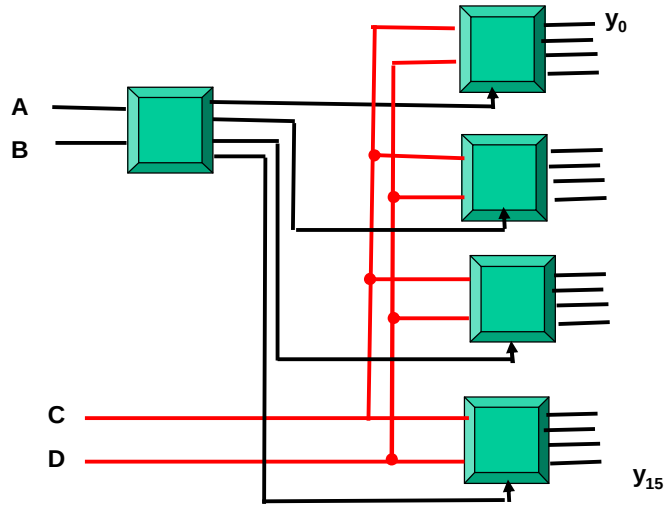
If E=1 , else all outputs are zeros

Building an 3x8 decoder using 2x4 decoders with enable



3x8

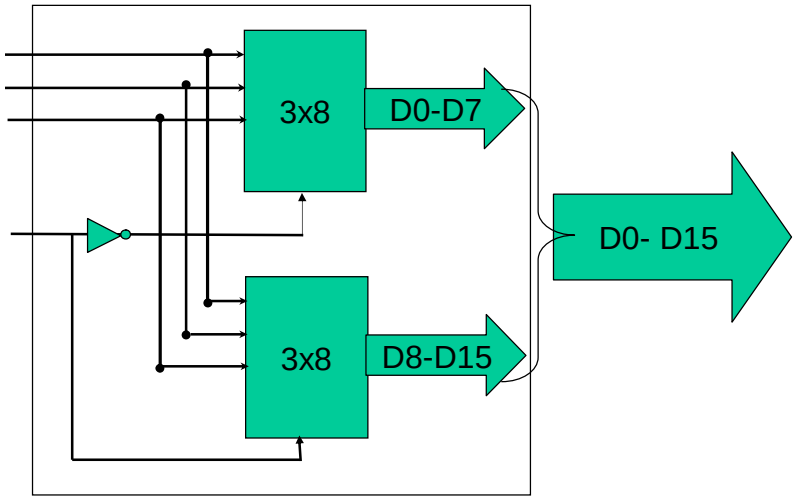
4x16 Decoder using 2x4 decoders



109

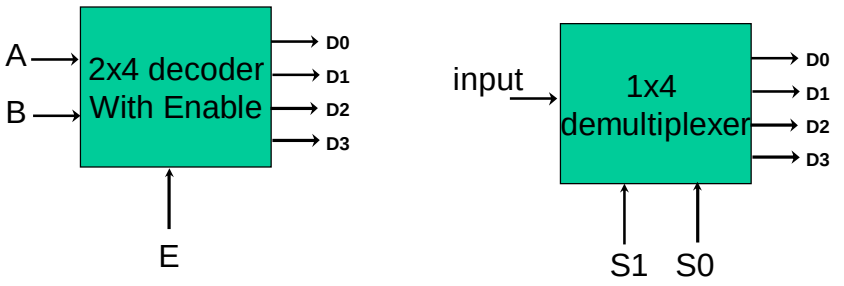
Build larger decoders

How can we build a 4x16 decoder using 3x8 decoders



110

Demultiplexer



A demultiplexer is a combinational circuit that sends the input to the output selected by selection lines.

A demultiplexer can be seen as a decoder with enable.

111

Encoder

An Encoder is a digital function that produces a reverse operation for that of a decoder.

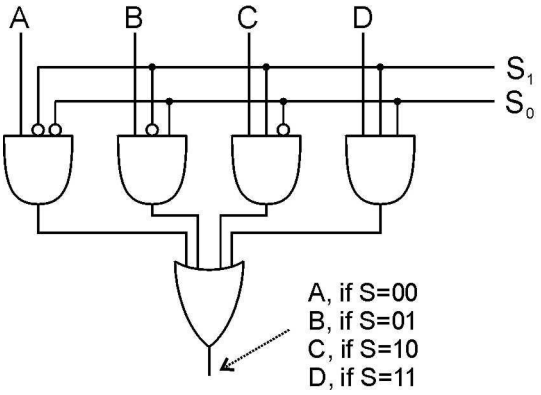
2^n x n encoder , has 2^n inputs and n outputs

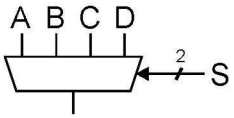
112

Multiplexer (MUX)

***n*-bit selector and 2^n inputs, one output**

- **output equals one of the inputs, depending on selector**

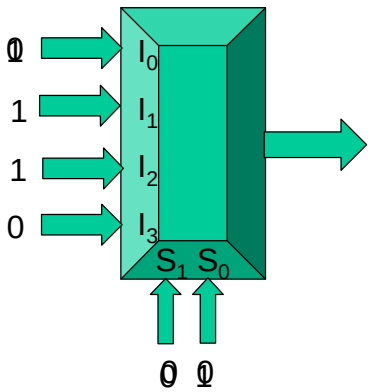




4-to-1 MUX

Multiplexer (MUX)

***n*-bit selector and 2^n inputs, one output**

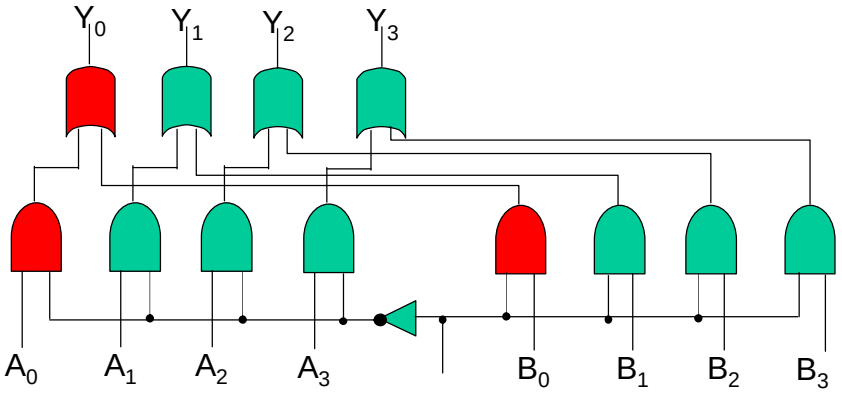


115

Quadruple 2x1 Mux

If we have 2 binary numbers A & B, each is 4 bit and we need to select one of the two numbers.

We need 4 (2x1) MUXs, or a Quad 2x1 mux with one selection line .



116

MUX to implement boolean functions.

We saw before how to implement a boolean function using decoders.

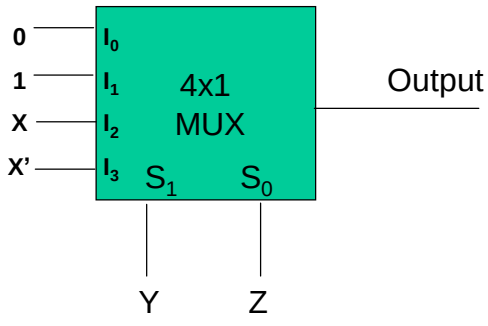
We can also implement this using a mux

1. If we have a function with n variables we need a MUX with n-1 selection lines (2^{n-1} MUX)

2. Find the implementation table, then draw the circuit

Ex. for F(x,y,z)=Σ(1,3,5,6)

	I_0	I_1	I_2	I_3
X'	0	(1)	2	(3)
X	4	(5)	(6)	7
	0	1	X	X'



Implement the boolean function F(w,x,y,z)=Σ(0,1,3,4,8,9,15) using Multiplexer

ROM

Types:

ROM

PROM

EPROM

EEPROM

HOW ROM works

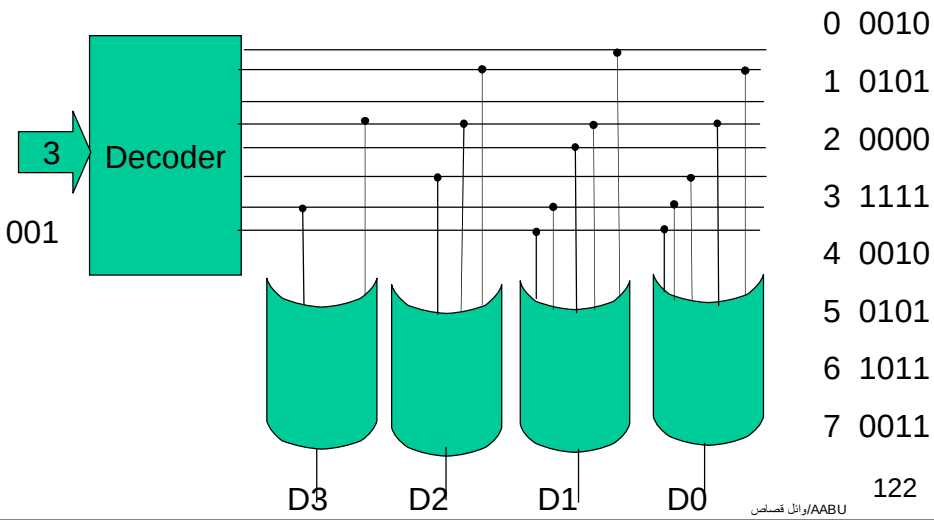
It consists of a decoder and OR gates.

In general Memory has address and Data lines.

We call a ROM with n address lines and m data lines 2^nxm memory

This memory has 2^n words, each word is m bits

8 x 4 bit Memory



From the previous discussion its clear that each output of the ROM represents a SOP

Ex.

$D_0 = \Sigma(1,3,5,6,7)$

$D_1 = \Sigma(0,3,4,6,7)$

$D_2 =$

$D_3 =$

Another example

Assume we want to build the Following Ckt that has two inputs and three outputs

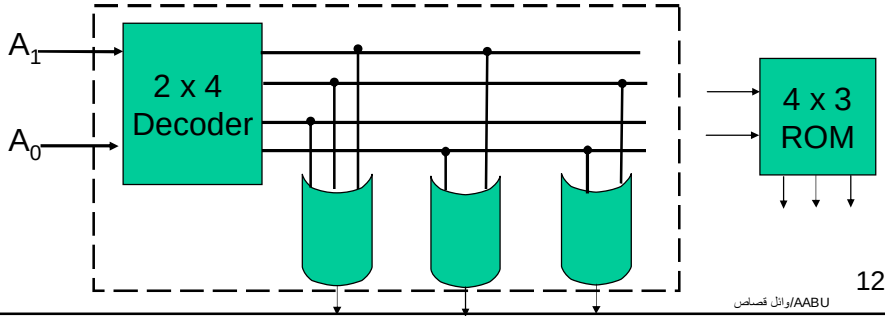
$F_0(A_1,A_0) = \Sigma(1,3)$

$F_1(A_1,A_0) = \Sigma(0,3)$

$F_2(A_1,A_0) = \Sigma(0,1,2)$

We can build this Ckt using a 4x3 ROM = 2^2 x 3 bit ROM

This ROM is built from a decoder, OR gates



Q. What is the Data stored in this ROM?

BCD to Ex.3 Converter

What is the size of the ROM needed?

What will be the data stored?

BCD to 7 segment Converter

What is the size of the ROM needed?

What will be the data stored?

Combinational vs. Sequential

Combinational Circuit

- always gives the same output for a given set of inputs
  - Ø ex: adder always generates sum and carry, regardless of previous inputs

Sequential Circuit

- stores information
- output depends on stored information (state) plus input
  - Ø so a given input might produce different outputs, depending on the stored information
- *example:* ticket counter
  - Ø advances when you push the button
  - Ø output depends on previous state
- useful for building "memory" elements and "state machines"

Sequential Logic

Synchronous sequential circuits:

A system whose behavior can be defined form the knowledge of its signals at discrete instants of time.

Asynchronous sequential circuits: a system whose behavior depends on the order which its input signals changes at any instance of time

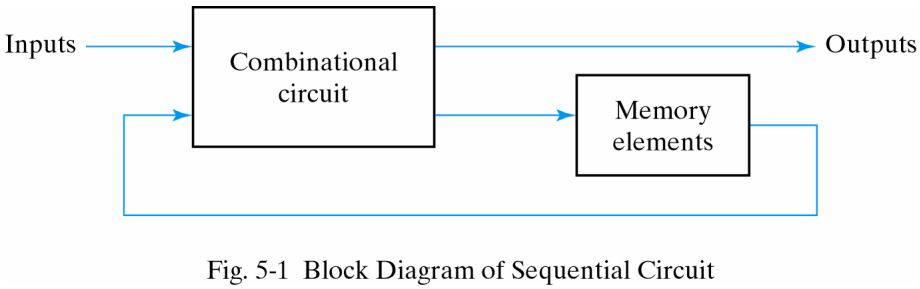
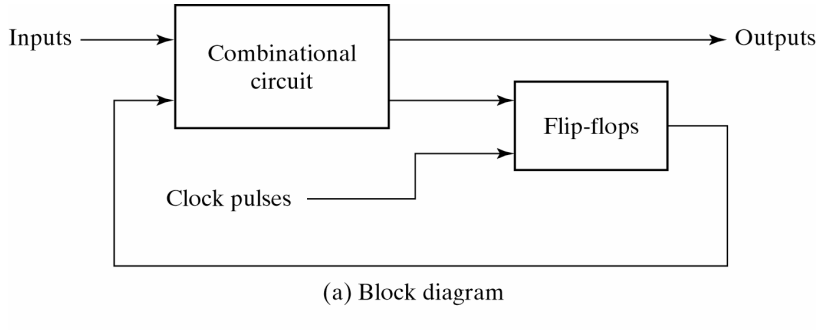
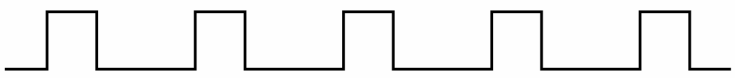


Fig. 5-1 Block Diagram of Sequential Circuit



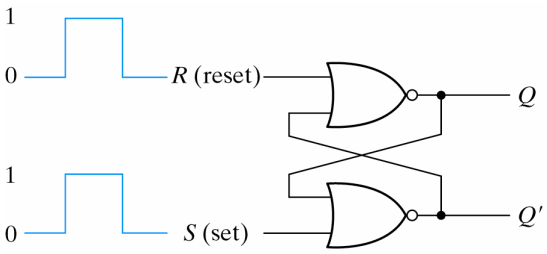
(a) Block diagram



(b) Timing diagram of clock pulses

Fig. 5-2 Synchronous Clocked Sequential Circuit

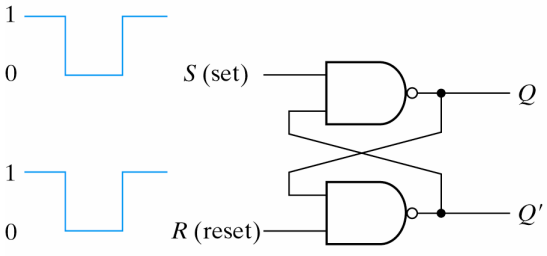
Flip Flops



S	R	Q	Q'	
1	0	1	0	
0	0	1	0	(after $S = 1, R = 0$)
0	1	0	1	
0	0	0	1	(after $S = 0, R = 1$)
1	1	0	0	

(a) Logic diagram (b) Function table

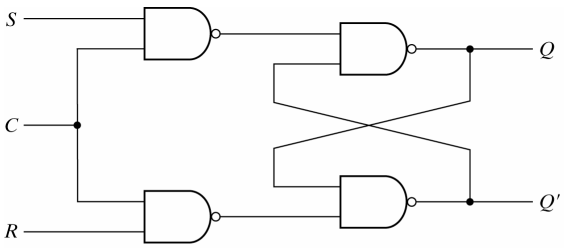
Fig. 5-3 *SR* Latch with NOR Gates



S	R	Q	Q'	
1	0	0	1	
1	1	0	1	(after $S = 1, R = 0$)
0	1	1	0	
1	1	1	0	(after $S = 0, R = 1$)
0	0	1	1	

(a) Logic diagram (b) Function table

Fig. 5-4 *SR* Latch with NAND Gates

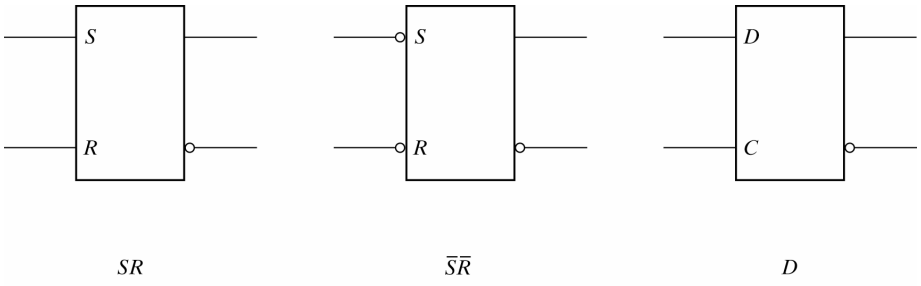


(a) Logic diagram

C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	$Q = 0$; Reset state
1	1	0	$Q = 1$; set state
1	1	1	Indeterminate

(b) Function table

Fig. 5-5 SR Latch with Control Input



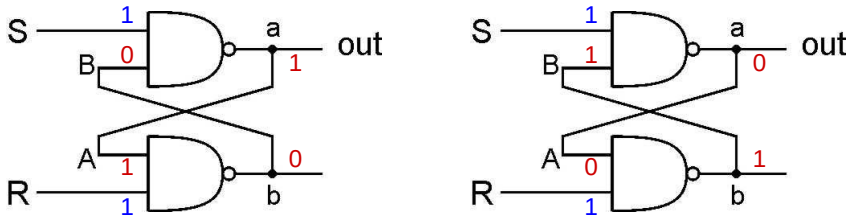
SR $\overline{S}\overline{R}$ D

Fig. 5-7 Graphic Symbols for Latches

R-S Latch: Simple Storage Element

R is used to "reset" or "clear" the element – set it to zero.

S is used to "set" the element – set it to one.



If both R and S are one, out could be <u>either</u> zero or one.

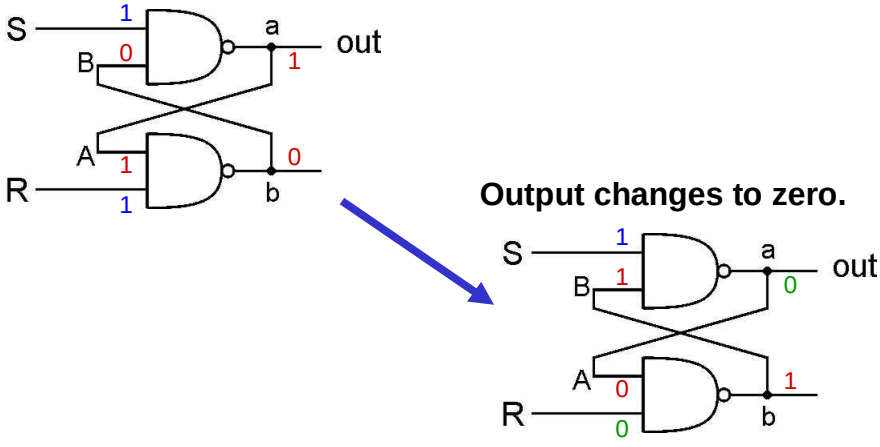
- **"quiescent" state -- holds its previous value**
- **note: if a is 1, b is 0, and vice versa**

135

AABU/وائل قصاص

Clearing the R-S latch

Suppose we start with output = 1, then change R to zero.



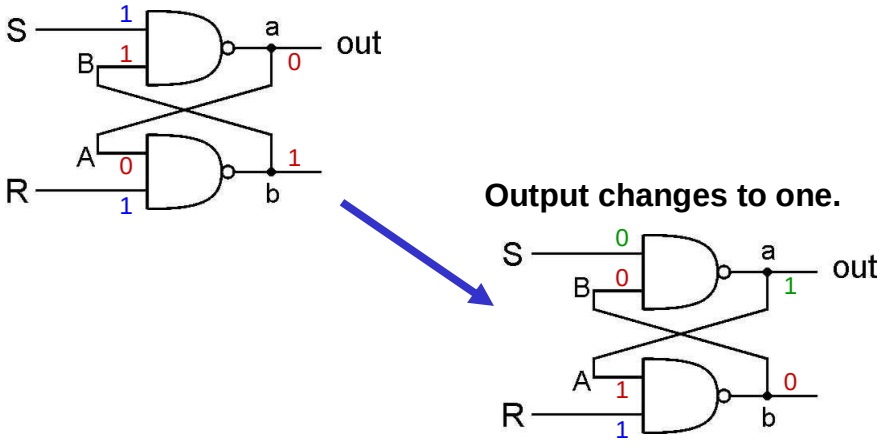
Then set R=1 to "store" value in quiescent state.

136

AABU/وائل قصاص

Setting the R-S Latch

Suppose we start with output = 0, then change S to zero.



Then set S=1 to "store" value in quiescent state.

R-S Latch Summary built by NAND

R = S = 1

- **hold current value in latch**

S = 0, R=1

- **set value to 1**

R = 0, S = 1

- **set value to 0**

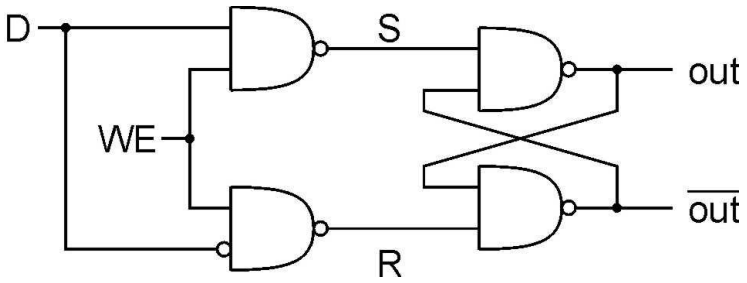
R = S = 0

- **both outputs equal one**
- **final state determined by electrical properties of gates**
- *Don't do it!*

Gated D-Latch

Two inputs: D (data) and WE (write enable)

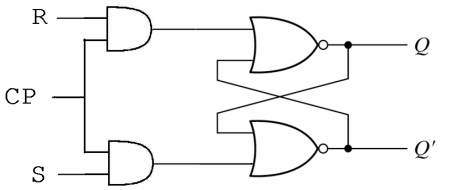
- **when WE = 1, latch is set to value of D**
  - **ØS = NOT(D), R = D**
- **when WE = 0, latch holds previous value**
  - **ØS = R = 1**



139

Characteristic equations

Remember that the output depends on the inputs and the current state.



- **First find the characteristic table**
- **Then derive the characteristic equation for the RS flip flop.**

SR

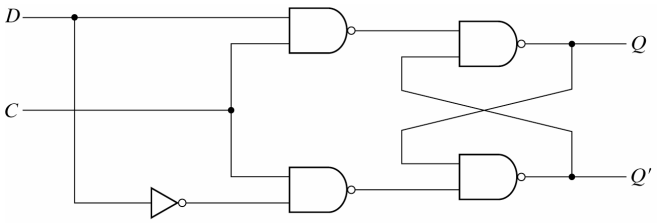
Q				
	0	0	X	1
	1	0	X	1

Q(t+1)=S+R'Q

Q_t	S	R	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

140

Characteristic equation for D flip flop



(a) Logic diagram

C	D	Next state of Q
0	X	No change
1	0	Q = 0; Reset state
1	1	Q = 1; Set state

(b) Function table

Fig. 5-6 D Latch

Qt	D	Q_{t+1}
0	0	0
0	1	1
1	0	0
1	1	1

0	1
0	1

Q(t+1)=D

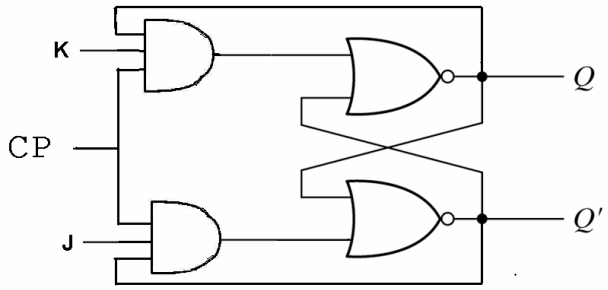
141

J K flip flop

It's a refinement of RS flip flop , that defined the indeterminate states in RS.

In RS flip flop the state 11 is not allowed ,

In JK flip flop the state 11 makes the flip flop changes (switches) its output.



142

S3S2S1S0

1010
1011
1100
1101
1110
1111

S1S0

S3S2				
	0	**0**	**0**	**0**
	0	**0**	**0**	**0**
	1	**1**	**1**	**1**
	0	**0**	**1**	**1**

S>=9 : Cout+S3S2+S3S1

Problem in JK flip flop:

- **When J=1 ,K=1 and the clock is 1 , $Q_{t+1}=Q'_t$**
- **Assume Q=1, it will flip to 0 then to 1 then to 0 and so on as long as the clock is 1.**
- **To avoid that the clock pulse (duration) must be less than the propagation delay of the Flip flop.**
- **But this is not a solution.**
- **The solution is to build a Master slave or edge triggered construction.**

Characteristic table and Equation

Q_t	J	K	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Q \ JK	0	0	1	1
	1	0	0	1

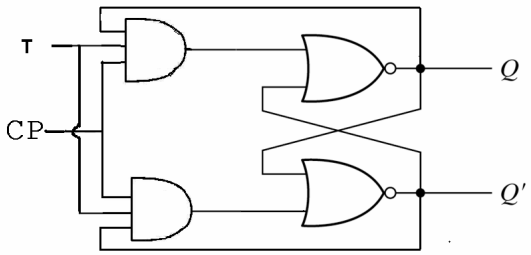
$$Q(t+1)=JQ'+K'Q$$

145

T flip flop

It's a single input version of the JK Flip flop

T flip Flop has the same Problem of JK when T=1

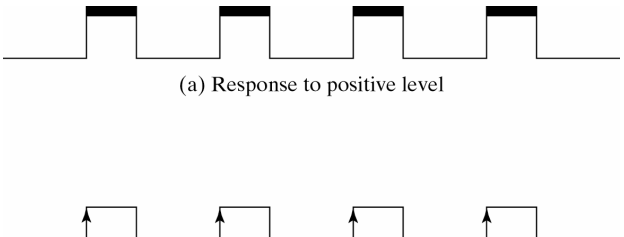


Qt	T	Q_{t+1}
0	0	0
0	1	1
1	0	1
1	1	0

0	1
1	0

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

146



(a) Response to positive level

(b) Positive-edge response

(c) Negative-edge response

Fig. 5-8 Clock Response in Latch and Flip-Flop

147

Master Slave D flip flop

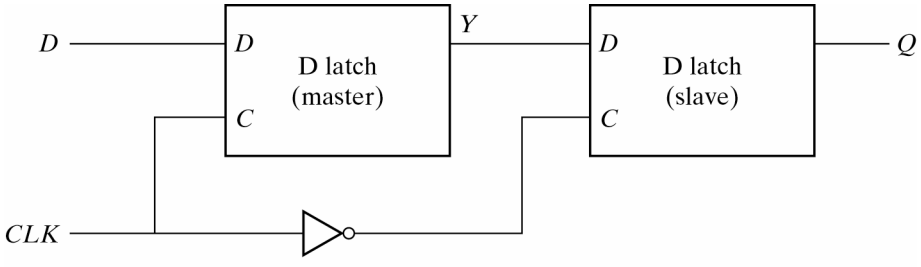
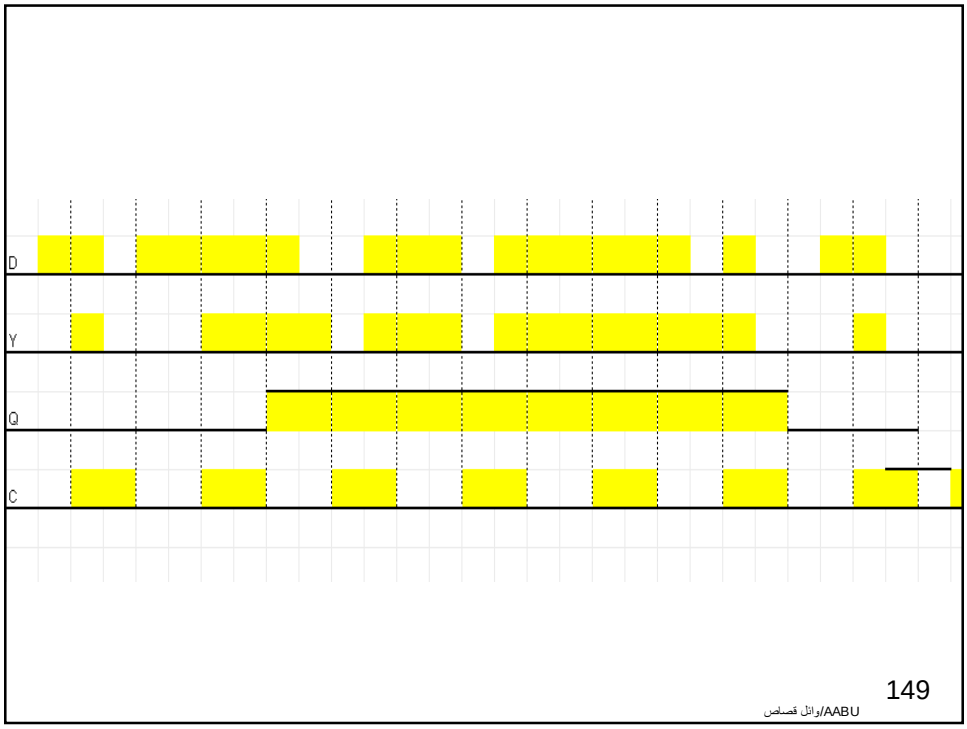


Fig. 5-9 Master-Slave *D* Flip-Flop

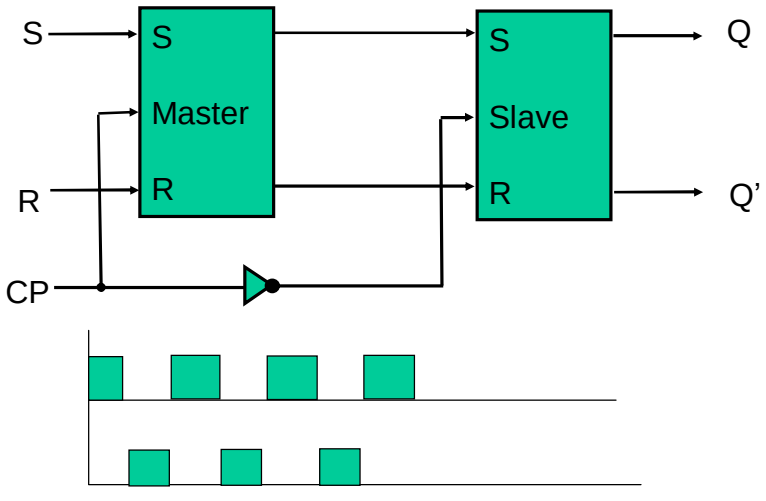
148



149

AABU/وائل قصاص

Master Slave RS flip flop



150

AABU/وائل قصاص

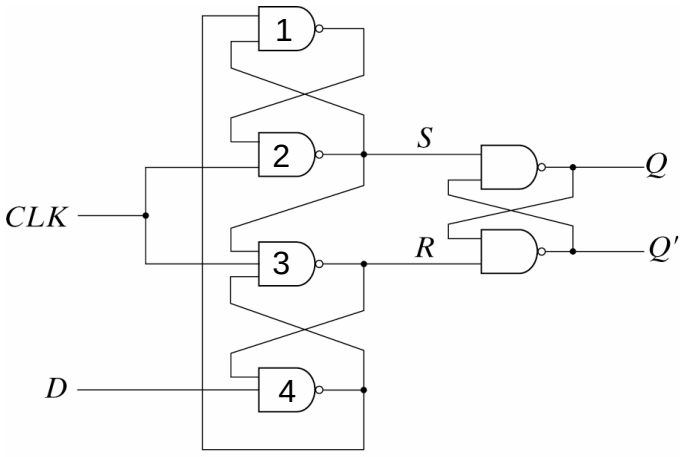


Fig. 5-10 *D*-Type Positive-Edge-Triggered Flip-Flop

151

When CLK = 0 , gates 2,3 are not working,gate 2=1.
gate3=1 è R=1,S=1
è No change in the output.
If D=0; Gate 4=1, Gate1 =0.
If D=1; Gate 4=0, Gate1 =1.

152

IF D=0 è Gate1=0,Gate4=1,Gate2=1,Gate3=1;

and CLK goes to 1

Gate 4=1, Gate 3=0, Gate1=0;Gate 2=1;

After the CLK being 1, if D changed to 1, this will not affect on Gate 4 , nor any other gates.

153

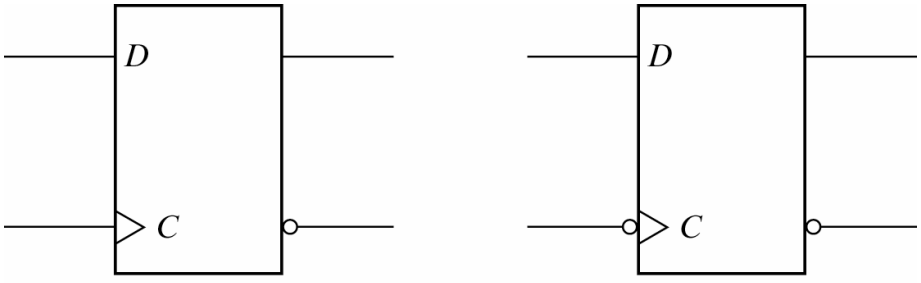
IF D=1 è Gate1=1,Gate4=0,Gate2=1,Gate3=1;

and CLK goes to 1

Gate 4=0, Gate 3=1, Gate1=1;Gate 2=0;

After the CLK being 1, if D changed to 0, Gate 4 =1 , other gates will not affect by this change.

154

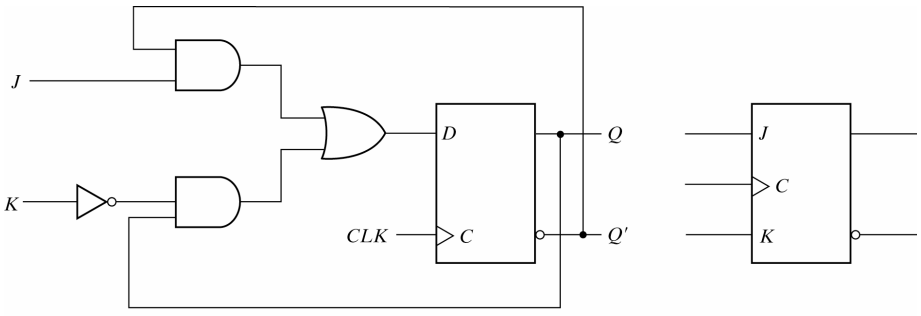


(a) Positive-edge (a) Negative-edge

Fig. 5-11 Graphic Symbol for Edge-Triggered *D* Flip-Flop

155

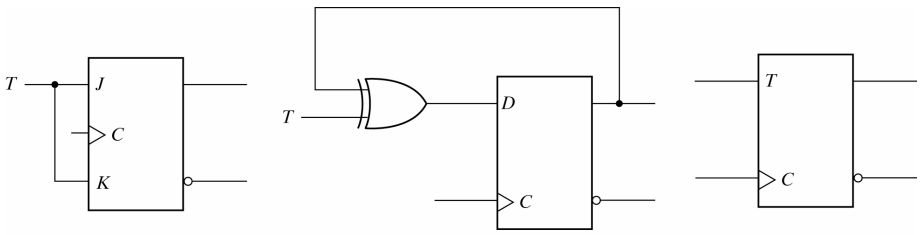
JK flip Flop from D flip flop



(a) Circuit diagram (b) Graphic symbol

Fig. 5-12 *JK* Flip-Flop

156



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

Fig. 5-13 T Flip-Flop

157

Analysis of sequential circuit

The behavior of a seq. ckt is determined form

a) **Inputs**
b) **Outputs**
c) **States of its flip flop**

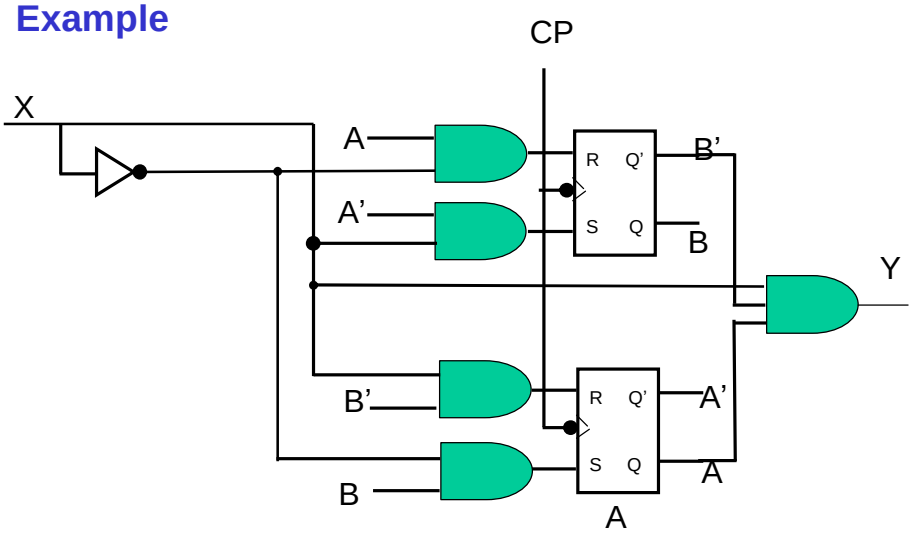
Both outputs and next state are function of input and present state.

From a seq. ckt diagram , we will find the

a) **state table**
b) **State diagram**

158

Example



159

State Table

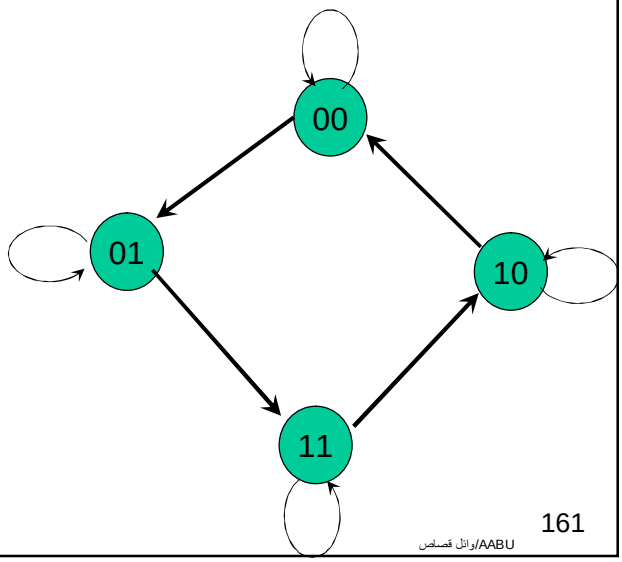
Present State	Next state		Output	
	X=0	X=1	X=0	X=1
AB	AB	AB	Y	Y
00	00	01	0	0
01	11	01	0	0
10	10	00	0	1
11	10	11	0	0

160

State diagram

The state table can be represented graphically using the state diagram.

Transition from a state to state is shown as arrow labeled with two values (Input/output)



161

State equation

State equation (or application equation) is an expression that shows the relation for the next state of each flip flop as a function of the present state and the inputs,

Method 1: using the characteristic equation of the flip flop

A(t+1) =S+R'Q

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

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

= X'B+X'.A+A.B

B(t+1) = S+R'Q

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

= X.A'+X.B+A'.B

162

State equation

Method 2:

From the state table.

A(t+1) = (A'B+AB'+AB).X' + ABX

=A'BX'+AB'X'+ABX' +ABX

=BX'+AX'+AB

6.7 Design procedure

1. **Build the state diagram**
2. **Build the state table**
3. **Assign binary values for each state**
4. **Determine the number of flip flops needed and assign a symbol for each flip flop**
5. **Choose the type of flip flop to be used (we will use JK)**
6. **From the state table derive the excitation and output tables**
7. **Simplify the flip flop functions**
8. **Draw the logic diagram**

The following formulas for JK flip flop inputs will help us

Q_t Q_{t+1}
0 to 0 J=0, K=X (don't care)
0 to 1 J=1, K=X
1 to 0 J=X, K=1
1 to 1 J=X, K=0

Excitation tables for Flip Flops

Q_t	Q_{t+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

RS

Q_t	Q_{t+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

JK

Q_t	Q_{t+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

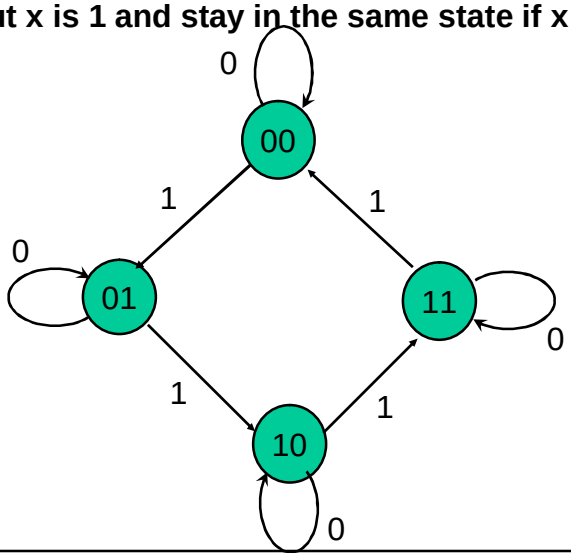
D

Q_t	Q_{t+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

T

Example 1

Design a circuit that works as a counter from 0 to 3 if the input x is 1 and stay in the same state if x is 0



Example 0

Design a counter that counts from 3 down to 0 one step on each clock

State table

	X=0	X=1
AB	AB	AB
00	00	01
01	01	10
10	10	11
11	11	00

We have 4 states so we need 2 flip flops

Excitation table

	Next state	flip flop inputs			
A B X	A B	JA	KA	JB	KB
0 0 0	0 0	0	X	0	X
0 0 1	0 1	0	X	1	X
0 1 0	0 1	0	X	X	0
0 1 1	1 0	1	X	X	1
1 0 0	1 0	X	0	0	X
1 0 1	1 1	X	0	1	X
1 1 0	1 1	X	0	X	0
1 1 1	0 0	X	1	X	1

Now we need to simplify the equation of each flip flop input

0	0	1	0
x	x	x	X

JA=Bx

x	x	x	X
0	0	1	0

KA=Bx

0	1	x	X
0	1	x	x

JB=x

x	x	1	0
x	x	1	0

KB=x

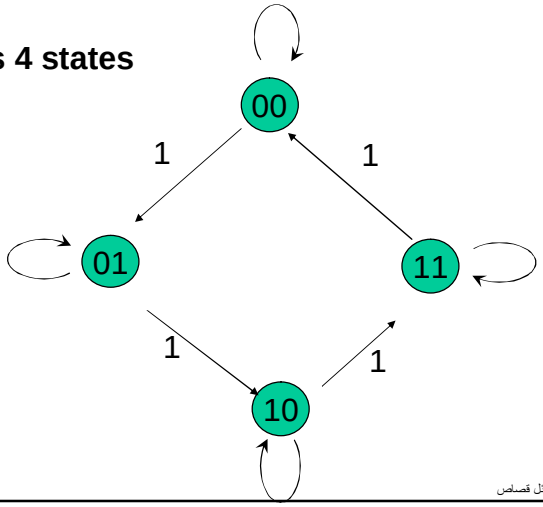
Example 2

Design a circuit that works as a down counter from 3 down to 0 if the input x is 1 and stay in the same state if x is 0

Example 3

Design a sequential Ckt that generate the following sequence 000, 001, 010,100 , using JK flip flops

This circuit has 4 states



173

Example 4

Repeat example 1 using RS flip flop

1) How RS Flip flop works

0 to 0 S=0, R=X (don't care)

0 to 1 S=1, R=0

1 to 0 S=0, R=1

1 to 1 S=X, R=0

2) Build the Excitation table, then simplify the equations that represents R and S for each flip flop.

174

Excitation table

	Next state	flip flop inputs			
A B X	A B	SA	RA	SB	RB
0 0 0	0 0	0	X	0	X
0 0 1	0 1	0	X	1	0
0 1 0	0 1	0	X	X	0
0 1 1	1 0	1	0	0	1
1 0 0	1 0	X	0	0	X
1 0 1	1 1	X	0	1	0
1 1 0	1 1	X	0	X	0
1 1 1	0 0	0	1	0	1

Now we need to simplify the equation of each flip flop input

175

0	0	1	0
X	X	0	X

SA=

X	X	0	X
0	0	1	0

RA=

0	1	0	X
0	1	0	X

SB=

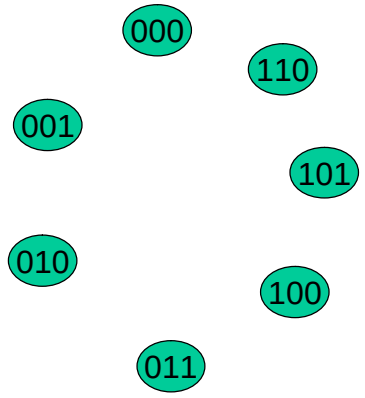
x	0	1	0
x	0	1	0

RB=

176

Example 5

Design an up_down counter that counts from 0 to 6, depending on the input value, if x=0 it counts down, if 1 it counts up



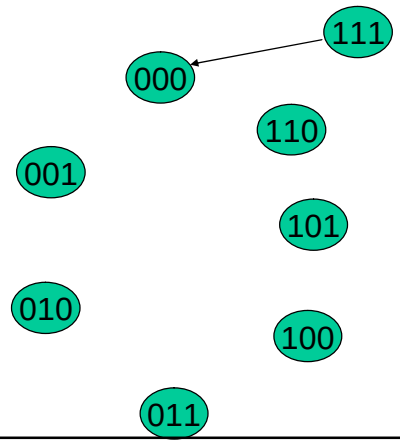
177

We have a small problem here.

What if the counter started with 111 ?

We need to move it to one of the valid states,

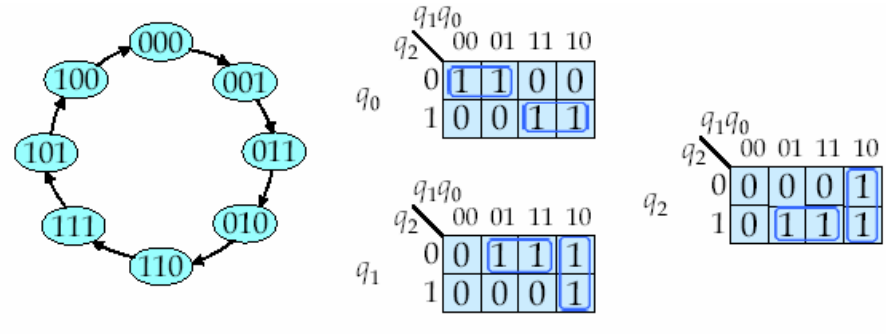
To 000 for example



178

Non-Standard Counters

- Counters are sometimes defined that count in an order other than standard numerical order.
- The state machine below is for a *gray code* counter in which one bit changes at a time.



3 bit binary counter

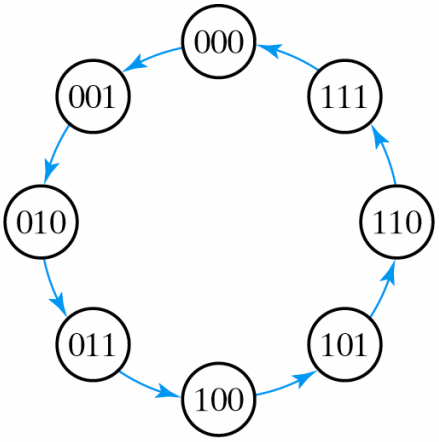


Fig. 5-29 State Diagram of 3-Bit Binary Counter

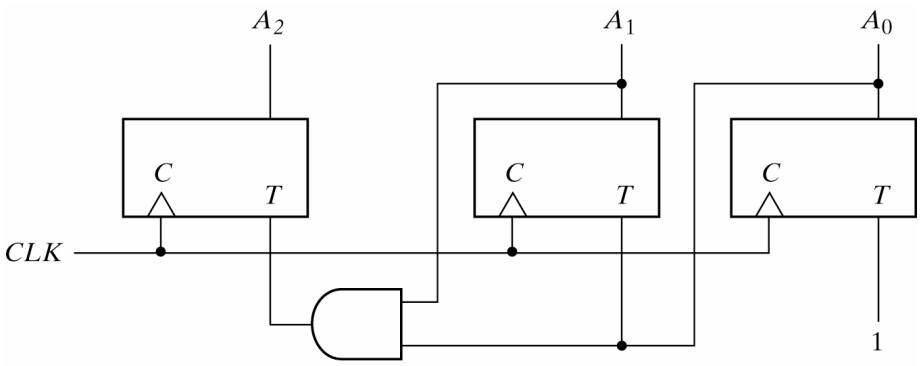


Fig. 5-31 Logic Diagram of 3-Bit Binary Counter

181

AABU/وائل قصاص

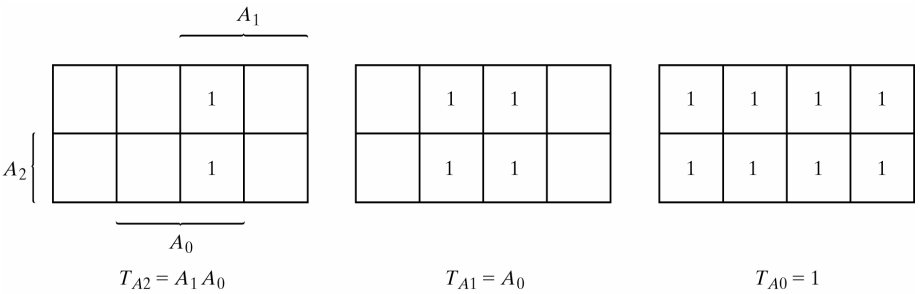


Fig. 5-30 Maps for 3-Bit Binary Counter

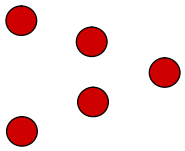
182

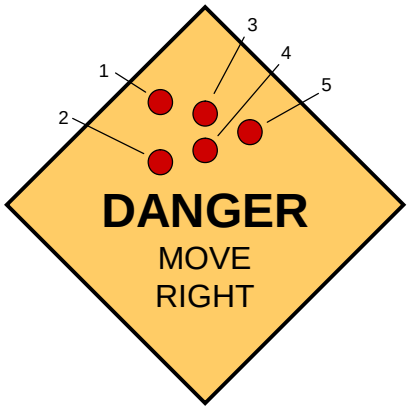
AABU/وائل قصاص

Complete Example

A blinking traffic sign

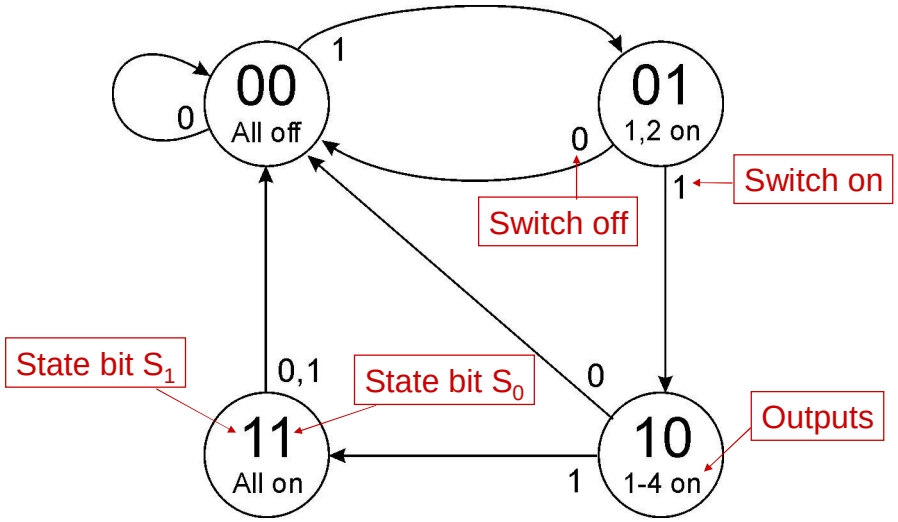
- **No lights on**
- **1 & 2 on**
- **1, 2, 3, & 4 on**
- **1, 2, 3, 4, & 5 on**
- **(repeat as long as switch is turned on)**
- **When the input is 0 , No lights on**





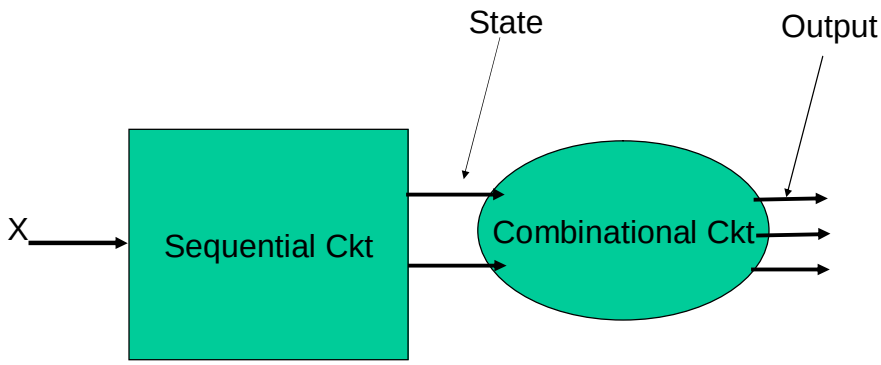
183

Traffic Sign State Diagram



Transition on each clock cycle.

184



185

Traffic Sign Truth Tables

Outputs
(depend only on state: S_1S_0)

Lights 1 and 2
Lights 3 and 4
Light 5

S_1	S_0	Z	Y	X
0	0	0	0	0
0	1	1	0	0
1	0	1	1	0
1	1	1	1	1

Next State: $S_1'S_0'$
(depend on state and input)

Switch

In	S_1	S_0	S_1'	S_0'
0	X	X	0	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	0

Whenever In=0, next state is 00.

186

2 input Sequential Ckts

Q. Build a counter that counts from 0 to 3, this counter has two inputs X,Y,

XY

00 Reset the counter (go to state 00)

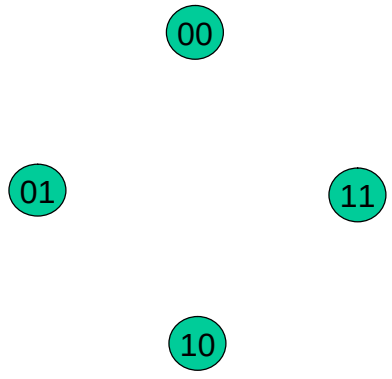
01 Count forward

10 Counts backward

11 No change

We have 4 states è we need two flip flops

We have 2 inputs è each state has 4 transitions



State table

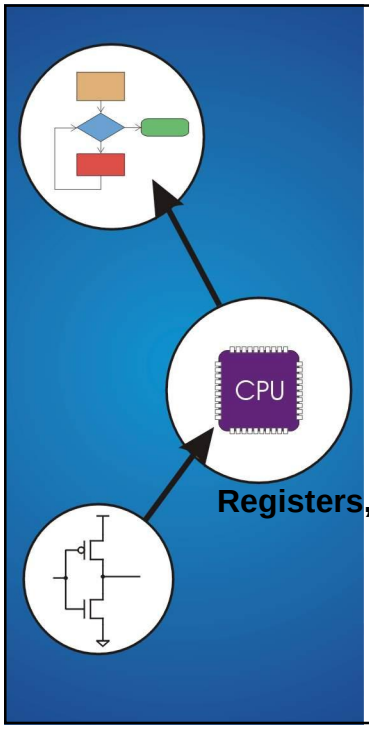
Present state A B	Next state XY=00 A B	XY=01 A B	XY=10 A B	XY=11 A B
0 0				
0 1				
1 0				
1 1				

189

Excitation table

XYAB	AB	TA	TB
0000			
0001			
0010			
0011			
0100			
0101			
0110			
0111			
1000			
1001			
1010			
1011			
1100			
1101			
1110			
1111			

190



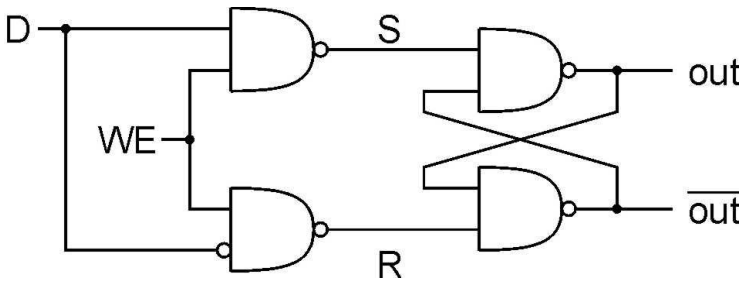
Chapter 7

Registers, Counters, and Memory units

Gated D-Latch

Two inputs: D (data) and WE (write enable)

- **when WE = 1, latch is set to value of D**
  - **S = NOT(D), R = D**
- **when WE = 0, latch holds previous value**
  - **S = R = 1**



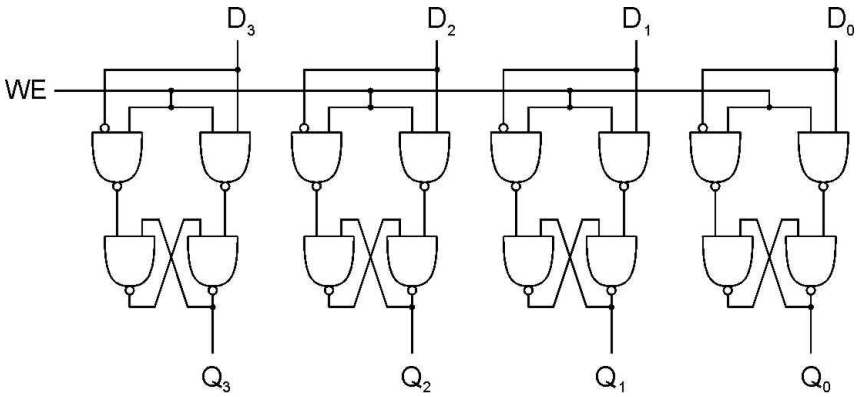
192

AABU/وائل قصاص

Register

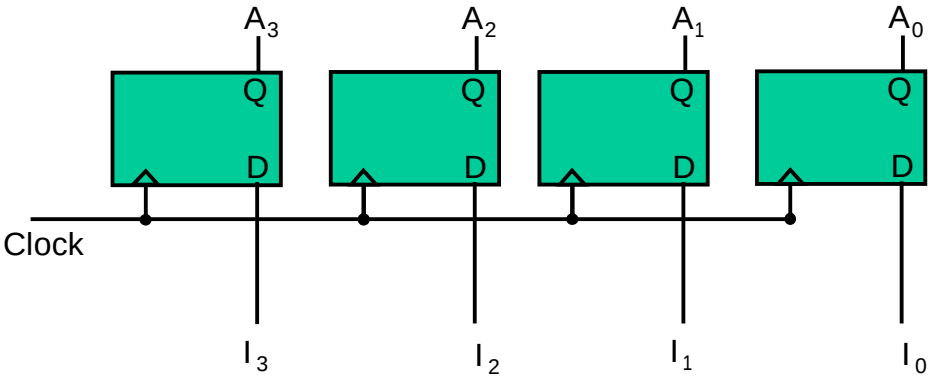
A register stores a multi-bit value.

- **We use a collection of D-latches, all controlled by a common WE.**
- **When WE=1, n-bit value D is written to register.**



193

4 bit register



194

Register with parallel load

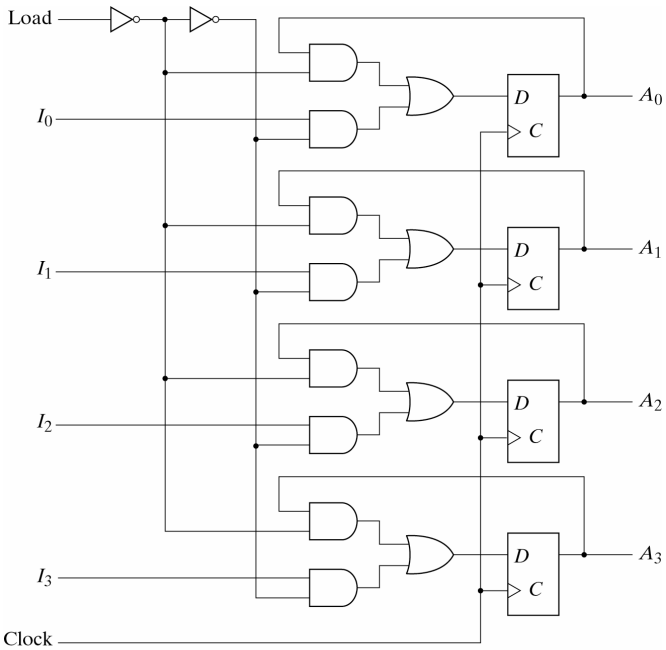


Fig. 6-2 4-Bit Register with Parallel Load

4 bit Shift right register

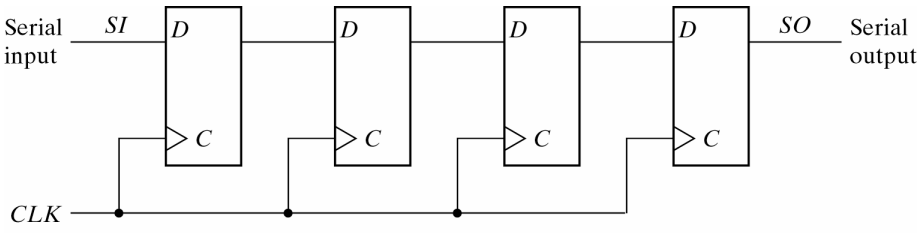
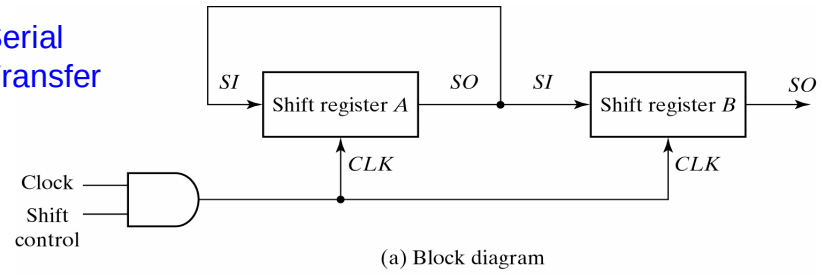
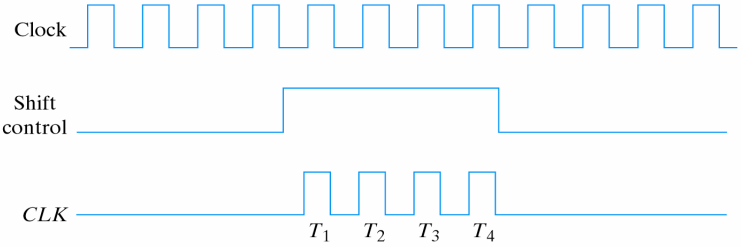


Fig. 6-3 4-Bit Shift Register

Serial Transfer



(a) Block diagram



(b) Timing diagram

Fig. 6-4 Serial Transfer from Register A to register B

197

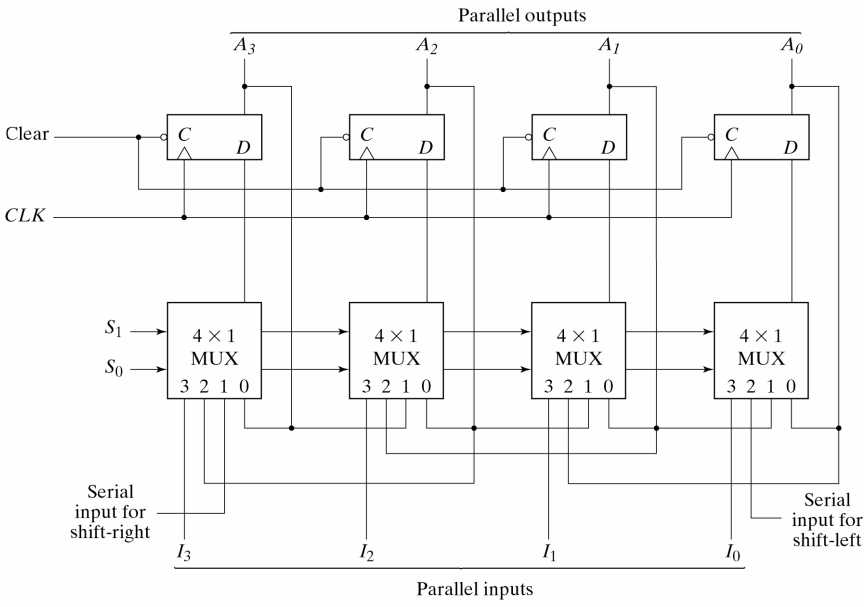


Fig. 6-7 4-Bit Universal Shift Register

198

Serial Adder

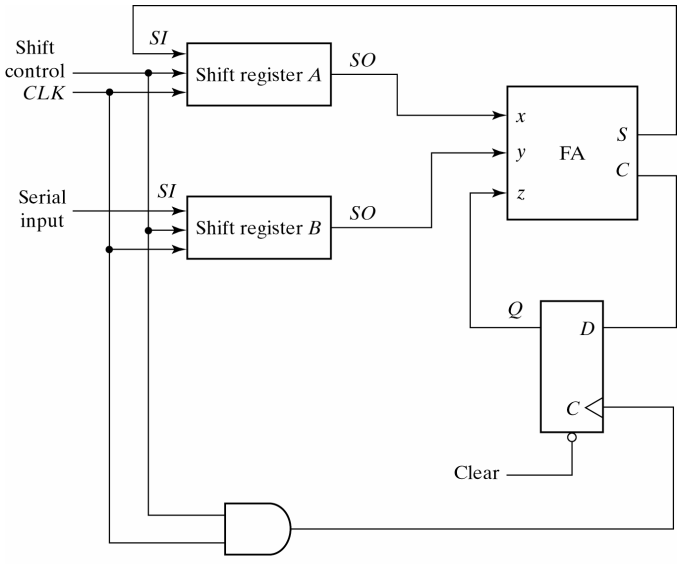


Fig. 6-5 Serial Adder

199

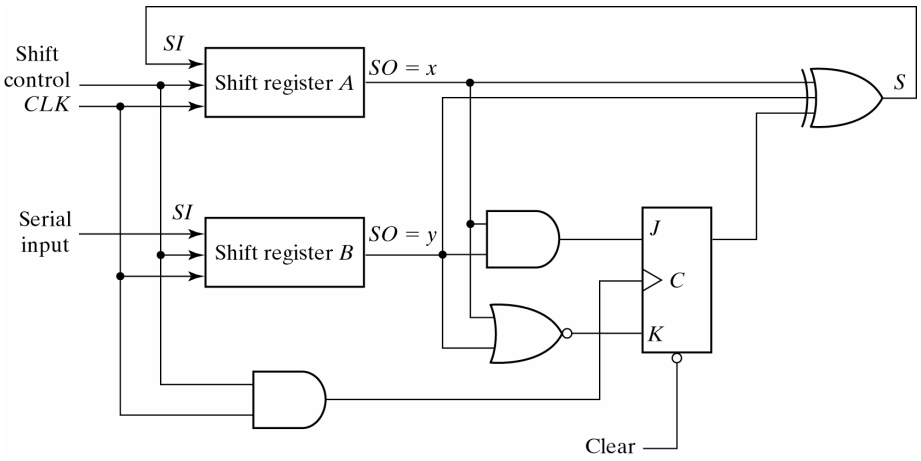


Fig. 6-6 Second form of Serial Adder

200

Ripple counter

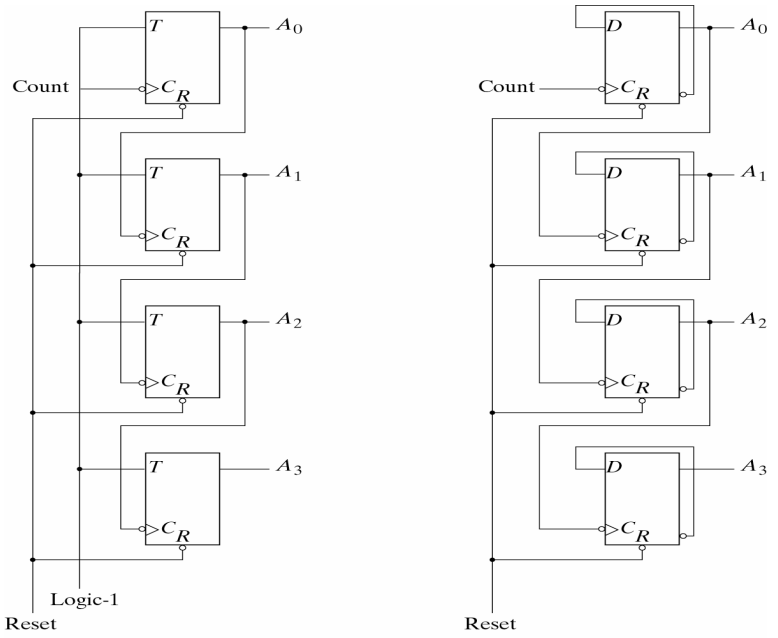
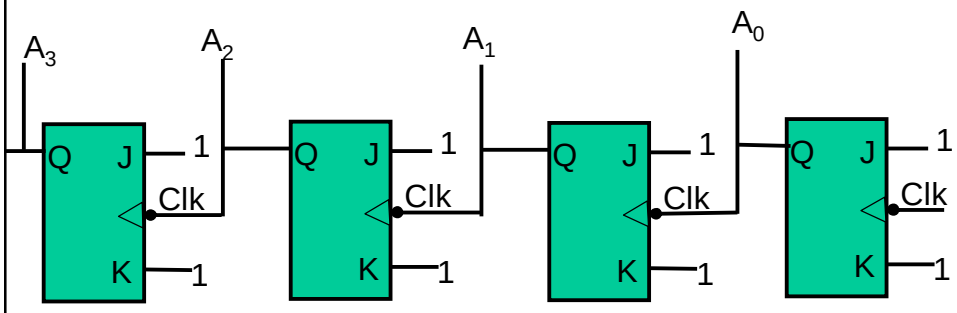


Fig. 6-8 4-Bit Binary Ripple Counter

4 bit ribble couter (JK)



202

AABU/وائل قصاص

Decimal counter

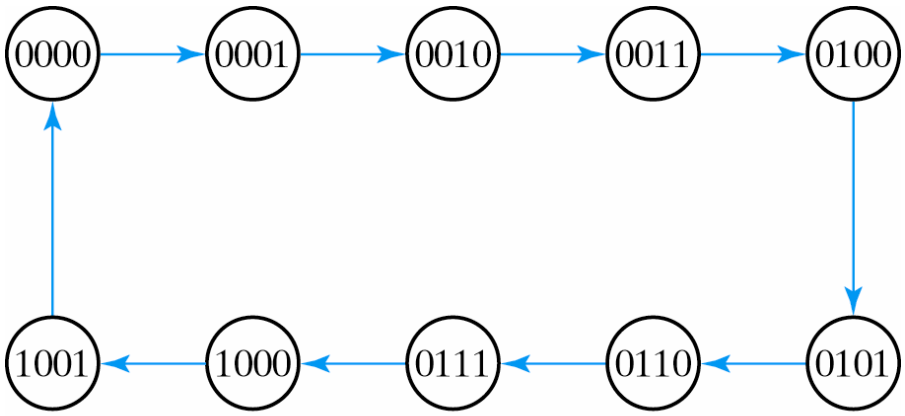


Fig. 6-9 State Diagram of a Decimal BCD-Counter

BCD Ripple counter

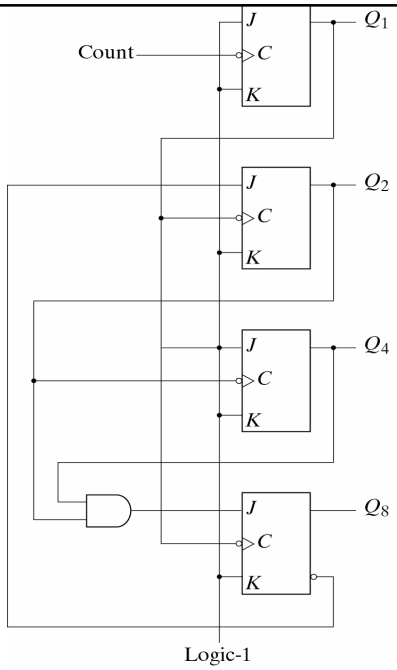
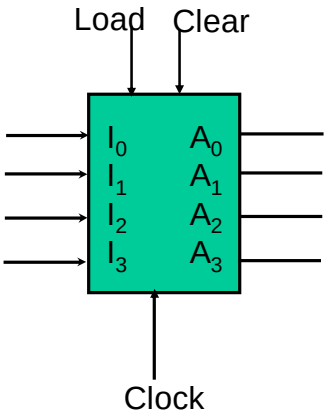


Fig. 6-10 BCD Ripple Counter

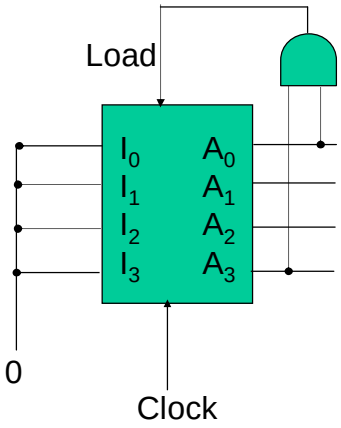
- **In BCD counter the first digit flips with the clock,**
- **The second digit flips depending on the first digit a clock if the number is less than 8, since J is connected to Q_8'**
- **When Q_8 becomes 1, J will be 0, this will clear Q_2.**
- **BUT this will take effect only after Q0 goes from 1 to 0.**
- **What about J8**

MSI 4 bit counter with Parallel load



Using an MSI Binary counter , Build a BCD counter

- **As you know, BCD counter goes to 0 after 9.**
- **All what we want to do is to load 0 if the counter value is 9**



207

Build a counter that counts from 0 to 6

208

Build a counter that counts from 5 to 13

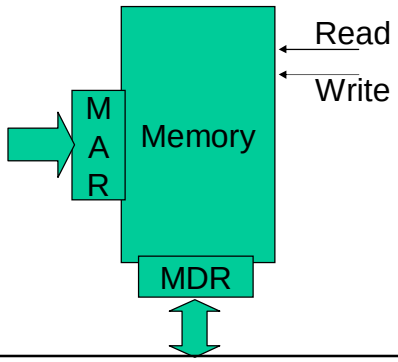
Johnson counter

Self read.

Required for the exam.

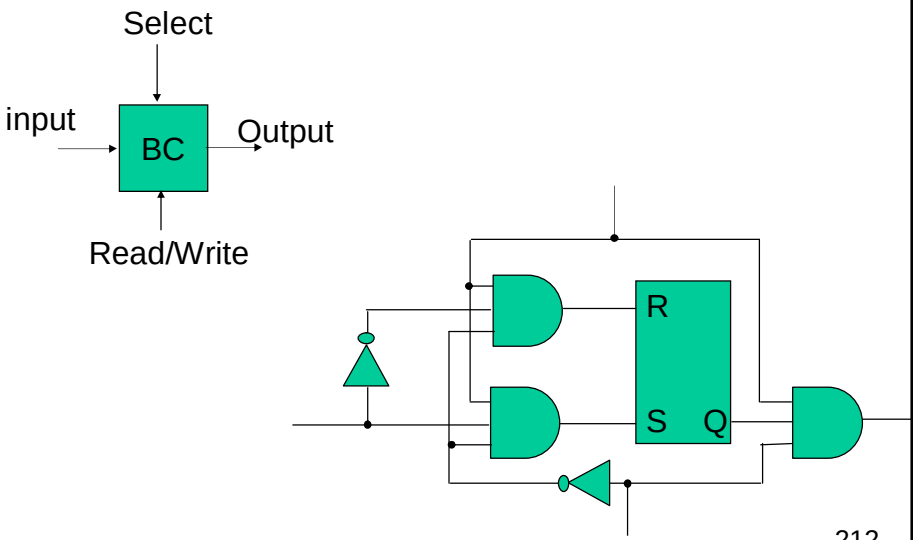
Memory Unit

- **A memory unit stores binary information in groups called *words. Each is n bits.***
- **Memory size is the number of locations (words) that a memory have.**
- **A memory word (which contains binary numbers) is used to represent an Instruction, Number, Character,…**
- **MAR**
- **MDR**



211

Binary Cell & RAM



212

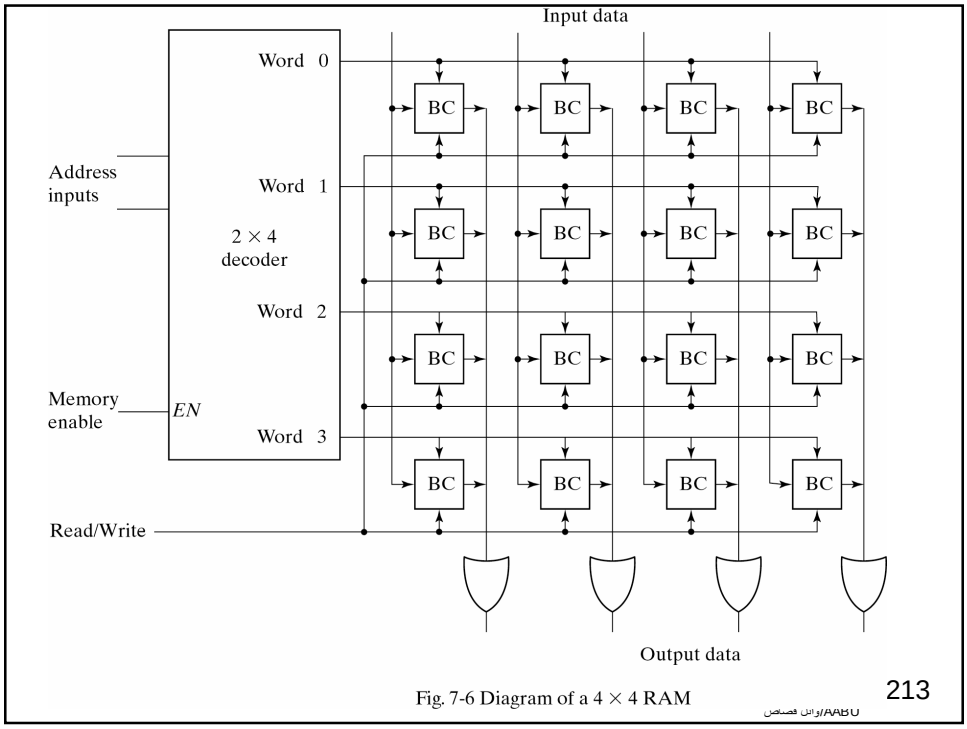


Fig. 7-6 Diagram of a 4 × 4 RAM

213