LOGIC CIRCUITS

TRINH VAN LOAN – Hanoi University of Science and Technology

1

1

REFERENCES

- Introduction to Logic Design, 2nd Edition, Alan B. Marcovitz, McGraw-Hill, 2005
- Complete Digital Design A Comprehensive Guide to Digital Electronics and Computer System Architecture, Mark Balch, McGraw-Hill, 2003
- Electronic - Digital Design Fundamentals, 2nd Edition, Kenneth J. Breeding, Prentice Hall, 1992
- Fundamentals of Digital Logic with VHDL, Stephen Brown, Zvonko Vranesic, McGraw-Hill, 2005
- Verilog HDL - A Guide to Digital Design and Synthesis. 2nd Edition, Samir Palnitkar, Prentice Hall, 2007

2

2

CONTENTS

3

3

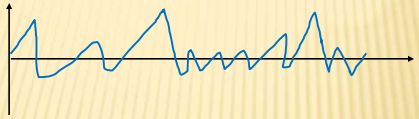
1.NUMBER SYSTEMS

- Digital systems: all of the signals are represented by discrete values (computers, calculators, most electronic systems)
- Digital systems usually operate with two-valued signals (0 and 1).
- Two-valued systems are more reliable
- The design of digital systems is referred to as logic design

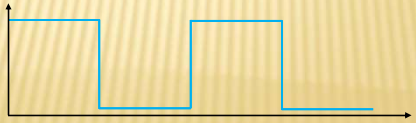
4

4

- Signals (Electric Signal - Voltage):
  - Analog Signal: x(t) x: function, t: variable (time)
    - x, t: continuous -> any value



  - Digital Signal xd(n): xd: function, n: variable (time)
    - xd, n: discrete -> certain value



5

5

A BRIEF REVIEW OF NUMBER SYSTEMS

- Integers are normally written using a positional number system, each digit represents the coefficient in a power series

$$N = a_{n-1}b^{n-1} + a_{n-2}b^{n-2} + \ldots + a_2b^2 + a_1b^1 + a_0$$

- n: number of digit, b: base (or radix), a_i: coefficients

$$0 \le a_i < b$$

- Decimal: $b = 10, a: 0 \ldots 9$
- Binary: $b = 2, a: 0,1$
- Octal: $b = 8$
- Hexadecimal: $b = 16, a: 0 \ldots 15$, the digits above 9: the first six letters of the alphabet (upper case) A, B, C, D, E, F

6

6

CONVERSION BETWEEN BASES

- Base *b* to base 10 conversion

$$N_{(b)} = a_n a_{n-1} a_{n-2} \ldots a_1 a_0$$

$$N_{(10)} = a_n.b^n + a_{n-1}.b^{n-1} + \ldots + a_1.b^1 + a_0.b^0$$

- $(110101)_2 = 1.2^5 + 1.2^4 + 0.2^3 + 1.2^2 + 0.2^1 + 1.2^0$
  $= (32)_{10} + (16)_{10} + (4)_{10} + (1)_{10} = (53)_{10}$
- $(73)_8 = 7.8^1 + 3.8^0 = (56)_{10} + (3)_{10} = (59)_{10}$
- $(32AF)_{16} = 3.16^3 + 2.16^2 + A.161 + F.160$
  $= (12288)_{10} + (512)_{10} + (160)_{10} + (15)_{10}$
  $= (12975)_{10}$

7

7

CONVERSION BETWEEN BASES

- Binary, Octal, Hexadecimal conversion
  - From binary to octal:
    - Group 3-bit groups from right to left.
    - Replace each group by octal digit with the same value.

$$(110111010011)_2 = \underbrace{110}_{6}\ \underbrace{111}_{7}\ \underbrace{010}_{2}\ \underbrace{011}_{3}$$
$$= (6723)_8$$

8

8

CONVERSION BETWEEN BASES

- From binary to hexadecimal:
  - Group 4-bit groups from right to left.
  - Replace each group by hexadecimal digit with the same value.

$$(1110111000101100 1)_2 = \underbrace{0001}_{1}\ \underbrace{1101}_{D}\ \underbrace{1100}_{C}\ \underbrace{0101}_{5}\ \underbrace{1001}_{9} = (1DC59)_{16}$$

9

9

CONVERSION BETWEEN BASES

- From base 2^n to binary:
  - Each digit is assigned a group of *n* bits
  - The equivalent binary number is obtained by concatenating these groups.

$$(4736)_8 = \underbrace{4}_{100}\ \underbrace{7}_{111}\ \underbrace{3}_{011}\ \underbrace{6}_{110}$$

$$(12A7F)_{16} = \underbrace{1}_{0001}\ \underbrace{2}_{0010}\ \underbrace{A}_{1010}\ \underbrace{7}_{0111}\ \underbrace{F}_{1111}$$

10

10

CONVERSION BETWEEN BASES

- From base k to base j:
  - If k and j are power of 2, to make an intermediate conversion to the binary equivalent:
    base 8 → binary→ hexadecimal
  - If not, using decimal as an intermediate conversion:
    base 5 → decimal→ binary

11

11

BINARY ADDITION

- To compute the sum of two binary numbers, say

  0 1 1 0 6
  0 1 1 1 +7

  as we do in decimal, we add one digit at a time, producing a sum and a carry to the next bit. We have and addition table for binary

  0+0=0
  0+1=1
  1+0=1
  1+1=10

12

12

BINARY ADDITION

- One-bit adder

a	b	c_{in}	c_{out}	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

13

13

SIGNED NUMBERS

- Positive integers referred to as unsigned numbers
- Computers must deal with signed numbers (both positive and negative numbers)
- To use the first bit of a number as a sign indicator (0 for positive , 1 for negative) and the remaining bits for magnitude
- In a 4-bit system

  $+5 \rightarrow 0101$ $-5 \rightarrow 1101$ $-3 \rightarrow 1011$
- With 3 bits for magnitude, the range of numbers available would be from -7 to +7

14

14

SIGNED NUMBERS

- This representation has both a positive (0000) and negative (1000) zero
- The major problem with signed-magitude is the complexity of arithmetic

+5	–5	+5	–5	–3	+3
+3	–3	–3	+3	+5	–5
+8	–8	+2	–2	+2	–2

- When the signs of the two operands are the same, we just add the magnitudes and retain the sign
- For other examples, we must determine which is the larger magnitude then subtract the smaller from the larger and finally attach the sign of the larger magnitude

15

15

SIGNED NUMBERS-TWO'S COMPLEMENT

- Signed binary numbers are nearly always stored in two's complement format
- The leading bit is still the sign bit (0 for positive)
- Positive numbers (and zero) are stored in normal binary
- The negative number, $-a$, is stored as the binary equivalent of $2^n - a$ in an n-bit system.
- In a 4-bit system, – 3 is stored as the binary for 16–3=13, 1101
- The largest number can be stored: $2^{n-1} - 1$
- The most negative number can be stored: -2^{n-1}

16

16

SIGNED NUMBERS-TWO'S COMPLEMENT

- Three- step approach to find the storage format for negative number in two's complement

1. Find the binary equivalent of the magnitude
2. Complement each bit (change 0's to 1's, 1's to 0's)
3. Add 1

–5		–1		–0	
1.	5: 0101	1.	1: 0001	1.	0: 0000
2.	1010	2.	1110	2.	1111
3.	1	3.	1	3.	1
	–5: 1011		–1: 1111		0000

There is no negative zero. In two's complement addition, the carry out of the most significant bit is ignored

17

17

SIGNED NUMBERS-TWO'S COMPLEMENT

- To find the magnitude of a negative number stored in two's complement format

1. Bit by bit complement (change 0's to 1's, 1's to 0's)
2. Add 1

–5		–1	
	1011		1111
1.	0100	1.	0000
2.	1	2.	1
	5: 0101		1: 0001

18

18

SIGNED NUMBERS-TWO'S COMPLEMENT

- Addition and Subtraction Using Two's Complement

To add any two numbers, no matter what the sign of each is, just to do binary addition on their representation

–5	1011	–5	1011	–5	1011
+7	0111	+5	0101	+3	0011
+2	(1)0010	0	(1) 0000	–2	(0) 1110

19

19

SIGNED NUMBERS-TWO'S COMPLEMENT

- Addition and Subtraction Using Two's Complement

Subtraction is accomplished by first taking the two's complement of the second operand, and then adding.

a – *b* is computed as *a* + (-*b*)

Consider the computation of 7 - 5

5: 0101		7:	0111
1010		–5:	+1011
+ 1		2	(1) 0010
–5: 1011			

20

20

BINARY-CODED DECIMAL REPRESENTATION

- In BCD, we use four bits (a nible) to represent each of the decimal digit 0 through 9.

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

21

21

BINARY-CODED DECIMAL REPRESENTATION

- Decimal-to-BCD conversion: each decimal digit is represented by its corresponding BCD nibble

  Conversion $(3729)_{10}$ to BCD

$$(3729)_{10} = \underbrace{3}_{0011}\ \underbrace{7}_{0111}\ \underbrace{2}_{0010}\ \underbrace{9}_{1001}$$

- BCD-to-Decimal conversion: to partition the binary pattern into groups of four bits each

  Conversion $(1001001101010001)_{BCD}$ to decimal

$$\underbrace{1001}_{9}\ \underbrace{0011}_{3}\ \underbrace{0101}_{5}\ \underbrace{0001}_{1}$$

22

22

2. SWITCHING ALGEBRA AND LOGIC CIRCUITS

- Each gate is defined by an algebraic expression specifying the output in terms of the input
- Algebra allows us to simplify this expression
- We can use Boolean algebra as a tool to analyze and design digital circuits
- Switching algebra is binary, all variables and functions take on one of two values, 0 and 1

23

23

DEFINITION OF SWITCHING ALGEBRA

- We first define the three operators of switching algebra:
- OR (written as +)

  a+b (a OR b) is 1 if and only if a=1 or b=1 or both
- AND (written as . or simply two variables catenated)

  a.b=ab (a AND b) is 1 if only if a=1 and b=1
- NOT (written —)

  $\overline{a}$ (NOT a) is 1 if and only if a=0
- Operator hierarchy: We always perform the NOT operator first, followed by AND and then OR, in the absence of parentheses. If there are parentheses, the expressions within them are evaluated first

24

24

DEFINITION OF SWITCHING ALGEBRA

- Truth table: a listing of all the possible input combinations and the value of each of the outputs for each of these input combinations
- Truth tables for the three operators

a	b	a+b
0	0	0
0	1	1
1	0	1
1	1	1

a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

a	$\overline{a}$
0	1
1	0

25

25

BASIC PROPERTIES OF SWITCHING ALGEBRA

Commutative	$a + b = b + a$	$ab = ba$
Associative	$a + (b + c) = (a + b) + c$	$a(bc) = (ab)c$
Identity	$a + 0 = a$	$a.1 = a$
Null	$a + 1 = 1$	$a.0 = 0$
Complement	$a + \overline{a} = 1$	$a.\overline{a} = 0$
Idempotency	$a + a = a$	$a.a = a$
Involution	$\overline{\overline{a}} = a$	
Distributive	$a(b + c) = ab + ac$	$a + bc = (a + b)(a + c)$
Adjacency	$ab + a\overline{b} = a$	$(a + b)(a + \overline{b}) = a$
Simplification	$a + \overline{a}b = a + b$	$a(\overline{a} + b) = ab$
DeMorgan's theorem	$\overline{(a + b)} = \overline{a}\,\overline{b}$	$\overline{ab} = \overline{a} + \overline{b}$

26

26

MANIPULATION OF ALGEBRAIC FUNCTIONS

- Literal: the apperance of a variable or its complement.
- Product term: one or more literals connected by AND operators

  The expression $a\overline{b} + b\overline{c}d + \overline{a}d + \overline{e}$ *contains 8 literals and 4 product terms*
- Standard product term *or* minterm: a product term that includes each variable of the problem either uncomplemented or complemented

  For a function of 4 variables, $w, x, y,$ *and* z*:*

  $\overline{w}xy\overline{z}$ *and* $wxyz$ *are minterms*, but $w\overline{y}z$ is not

27

27

MANIPULATION OF ALGEBRAIC FUNCTIONS

- Sum of products (SOP) expression: one or more product terms connected by OR operators

  $wx\overline{y}z + \overline{w}xy\overline{z} + w\overline{x}yz + wxyz$ (4 product terms)

  $x + \overline{w}y + wx\overline{y}z$ (3 product terms)

  $w\overline{y}$ (1 product term)

  z (1 product term)
- Canonical sum or sum of standard product terms: a sum of products expression where all of the terms are standard product terms

  $wx\overline{y}z + \overline{w}xy\overline{z} + w\overline{x}yz + wxyz$

28

28

MANIPULATION OF ALGEBRAIC FUNCTIONS

- Minimum sum of products expression: one of those SOP expressions for function that has the fewest number of product terms with the fewest number of literals

$xyz + xy\overline{z} + x\overline{y}\overline{z} + x\overline{y}z + \overline{x}yz$ (1) 5 terms, 15 literals

$\overline{x}y + x\overline{y} + xyz$ (2) 3 terms, 7 literals

$\overline{x}y + x\overline{y} + xz$ (3) 3 terms, 6 literals

$\overline{x}y + x\overline{y} + yz$ (4) 3 terms, 6 literals

Expressions (3) and (4) are minima

29

29

MANIPULATION OF ALGEBRAIC FUNCTIONS

- Sum term: one or more literals connected by OR operators
- Standard sum term or maxterm: a sum term that includes each variables of the problem either uncomplemented or complemented

  For a function of 4 variables $w, x, y,$ and z, the terms $w + x + y + z$ and $\overline{w} + x + y + \overline{z}$ are standard sum terms but $w + \overline{y} + z$ is not
- Product of sum (POS) expression: one or more sum terms connected by AND operators

$(\overline{w} + x)(w + y)$, $w(x + y)$ 2 terms

$w + x$, x 1 term

30

30

MANIPULATION OF ALGEBRAIC FUNCTIONS

- Canonical product or product of standard sum terms: a product of sum expression where all of the terms are standard sum term.

$$(\overline{w} + x + y + \overline{z})(w + \overline{x} + \overline{y} + \overline{z})$$

- Minimum is defined the same way for both POS and SOP. The expression with the fewest number of terms and the fewest number of literals.

31

31

SHANNON EXPANSION

- Boole's expansion theorem, often referred to as the Shannon expansion or decomposition, is the identity:

$$f(X_1, X_2, \ldots, X_n) = X_1 \cdot f(1, X_2, \ldots, X_n) + X_1' \cdot f(0, X_2, \ldots, X_n)$$

- Dual Form: $f(X_1, X_2, \ldots, X_n) = (X_1 + f(0, X_2, \ldots, X_n)) \cdot (X_1' + f(1, X_2, \ldots, X_n))$
- Repeated application for each argument leads to the Sum of Products (SoP) canonical form of the Boolean function *f*. For example for *n*=2 that would be:

$$\begin{aligned} f(X_1, X_2) &= X_1 \cdot f(1, X_2) + X_1' \cdot f(0, X_2) \\ &= X_1X_2 \cdot f(1,1) + X_1X_2' \cdot f(1,0) + X_1'X_2 \cdot f(0,1) + X_1'X_2' \cdot f(0,0) \end{aligned}$$

- Likewise, application of the dual form leads to the Product of Sums (PoS) canonical form (using the distributivity law of + over .

$$\begin{aligned} f(X_1, X_2) &= (X_1 + f(0, X_2)) \cdot (X_1' + f(1, X_2)) \\ &= (X_1 + X_2 + f(0,0)) \cdot (X_1 + X_2' + f(0,1)) \cdot (X_1' + X_2 + f(1,0)) \cdot (X_1' + X_2' + f(1,1)) \end{aligned}$$

32

32

SHANNON EXPANSION

- Boole's expansion theorem, often referred to as the Shannon expansion or decomposition, is the identity:

$$f(X_1, X_2, \dots, X_n) = X_1 . f(1, X_2, \dots, X_n) + \overline{X_1} . f(0, X_2, \dots, X_n)$$

- Dual Form:

$$f(X_1, X_2, \dots, X_n) = \left(X_1 + f(0, X_2, \dots, X_n)\right) . \left(\overline{X_1} + f(1, X_2, \dots, X_n)\right)$$

33

33

SHANNON EXPANSION

$$f(X_1, X_2) = X_1 . f(1, X_2) + \overline{X_1} . f(0, X_2) =$$
$$X_1 X_2 . f(1,1) + X_1 \overline{X_2} . f(1,0) + \overline{X_1} X_2 . f(0,1) + \overline{X_1}\ \overline{X_2} . f(0,0)$$

$$f(X_1, X_2) = \left(X_1 + f(0, X_2)\right) . \left(\overline{X_1} + f(1, X_2)\right) =$$
$$(X_1 + X_2 + f(0,0)) . \left(X_1 + \overline{X_2} + f(0,1)\right) . \left(\overline{X_1} + X_2 + f(1,0)\right) . \left(\overline{X_1} + \overline{X_2} + f(1,1)\right)$$

34

34

FROM THE TRUTH TABLE TO ALGEBRAIC EXPRESSIONS

- Each row of the truth table corresponds to a product term. A sum of products expression is formed by ORing those product terms corresponding to rows of the truth table for which the function is 1

a	b	f
0	0	0
0	1	1
1	0	1
1	1	0

$$f(a,b) = \overline{a}b + a\overline{b}$$
$$= (a+b)(\overline{a} + \overline{b})$$
$$= m_1 + m_2$$
$$= \sum m(1,2)$$

$$\overline{f}(a,b) = \overline{a}\overline{b} + ab$$
$$= m_0 + m_3$$
$$= \sum m(0,3)$$

35

35

a	b	c	f(a,b,c)
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

$$f(a,b,c) = \overline{a}\,\overline{b}\,\overline{c} \dots = \sum m(0,2,3,6)$$

$$f(a,b,c) = (a + b + \overline{c})(\quad)(\quad)(\quad)$$
$$= \prod M(1,4,5,7)$$

36

36

DON'T CARE CONDITIONS

- In some systems, the value of the output is specified for only some of the input conditions (*incompletely specified function*)
- For the remaining input combinations, it does not matter what the output is, that is, we don't care.
- In a truth table, don't cares are indicated by an X (some of literature uses *d*)

a	b	f
0	0	1
0	1	1
1	0	1
1	1	x

a	b	f_1	f_2
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	1

$$f(a,b) = \sum m(0,1,2) + \sum d(3)$$

37

37

SIMPLIFICATION OF ALGEBRAIC EXPRESSIONS

- The following properties are the ones most likely to reduce the number of terms or literals:

$$ab + a\overline{b} = a \qquad (a+b)(a+\overline{b}) = a$$
$$a + \overline{a}b = a + b \qquad a(\overline{a}+b) = ab$$
$$a + ab = a \qquad a(a+b) = a$$

- Some examples

$$wx + wxy + \overline{w}yz + \overline{w}\overline{y}z + \overline{w}xy\overline{z} =$$
$$(wx + wxy) + (\overline{w}yz + \overline{w}\overline{y}z) + \overline{w}xy\overline{z} =$$
$$wx + \overline{w}(z + xy\overline{z}) = wx$$
$$= \overline{w}z$$
$$= \overline{w}z$$

38

38

SIMPLIFICATION OF ALGEBRAIC EXPRESSIONS

$$(x+y)(x+y+\overline{z}) + \overline{y} = (x+y) + \overline{y} = 1$$

- t_1 and t_2 represent product terms

$$at_1 + \overline{a}t_2 + t_1t_2 = at_1 + \overline{a}t_2 + t_1t_2(a+\overline{a})$$
$$= at_1 + \overline{a}t_2$$

- Properties always appear in dual pairs. To obtain the dual of a property, interchange OR and AND, and the constants 0 and 1.

$$(a+t_1)(\overline{a}+t_2)(t_1+t_2) = (a+t_1)(\overline{a}+t_2)$$

- The principle of duality states that if an equation is always valid in Boolean algebra, its dual is also valid.

39

39

THE KARNAUGH MAP

- The algebraic methods allow us, in theory, to simplify any function. However, there is no formal method, the approach is totally heuristic, depending heavily on experience.
- An approach is easier to implement, Karnaugh map (K-map). This is a graphical approach to finding suitable product terms for use in sum of product expressions

40

40

THE KARNAUGH MAP

- The K-map consists of one square for each possible minterm in a function. A two-variable map has 4 squares, a three-variable map has 8 squares, and a four-variable map has 16 squares

A \ B	0	1
0	0	1
1	2	3

A \ BC	00	01	11	10
0	0	1	3	2
1	4	5	7	6

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

41

41

THE KARNAUGH MAP

- An **implicant** of a function is a product term that can be used in a SOP expression for that function. An implicant is a rectangle of 1,2,4,8,...(any power of 2)1's. That rectangle may not include any 0's. All minterms are implicants. An implicant covers certain minterms

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

Minterms	*Group of 2*	*Group of 4*
$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}CD$	CD
$\bar{A}\bar{B}C\bar{D}$	BCD	
$\bar{A}B\bar{C}D$	ACD	
$AB\bar{C}\bar{D}$	$B\bar{C}D$	
$AB\bar{C}D$	$AB\bar{C}$	
$ABCD$	ABD	
$A\bar{B}CD$		

42

42

A	B	F(A,B)
0	0	0
0	1	1
1	0	1
1	1	1

B \ A	0	1
0	0	1
1	1	1

43

43

THE KARNAUGH MAP

- A prime implicant is an implicant that is not fully contained in any one other implicant. Each of the product terms obtained by grouping of 1s in the K-map is called a prime implicant. It is a candidate to be a term in the minimized function

$$\bar{A}\bar{B}\bar{C}\bar{D},\ A\bar{B}\bar{C},\ ABD,\ CD$$

- An essential prime implicant is a prime implicant that includes at least one 1 that is not included in any other prime implicant.

$\bar{A}\bar{B}\bar{C}\bar{D},\ A\bar{B}\bar{C},\ CD$: *essential prime implicants*

The term essential is derived from the idea that we must use that PI in any minimum sum of products expression

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

44

44

THE KARNAUGH MAP

- Minimum SOP expressions using K-map

1. Find all essential PI. Circle them on the map and mark the minterm(s) that make them essential with an asterisk (*). It is usually quickest to start with the most isolated 1's, that is, those that have the fewest adjacent squares with 1's in them.
2. Find enough other PI to cover the function. Do this using two criteria:

   a) Choose a PI that covers as many new 1's (that is , those not already covered by a chosen PI)

   b) Avoid leaving isolated uncovered 1's
3. If we group 2^n adjacent 1s,we can eliminate *n* literals.

45

45

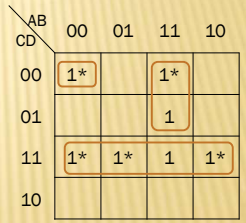
THE KARNAUGH MAP

- **Examples**

m_0 has no adjacent 1's, therefore $\overline{A}\overline{B}\overline{C}\overline{D}$ is a PI and an EPI

m_{12} has only one adjacent 1 and thus $A\overline{B}\overline{C}$ is essential

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



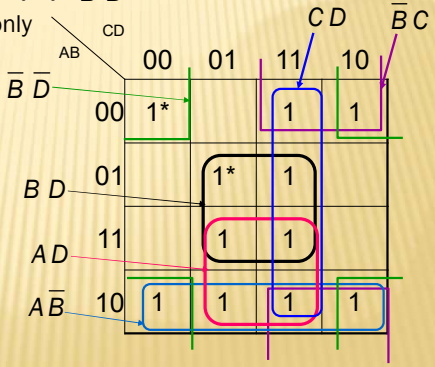
46

46

THE KARNAUGH MAP

- Minterm m_0 is covered only by $\overline{B}\,\overline{D}$
- Minterm m_5 is covered only by BD
- There are 4 simplified forms

$$F = \overline{B}\overline{D} + BD + \underbrace{AD}_{Or\ A\overline{B}} + \underbrace{CD}_{Or\ \overline{B}C}$$



47

47

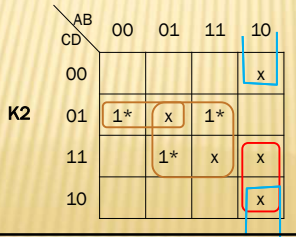
THE KARNAUGH MAP – DON'T CARE

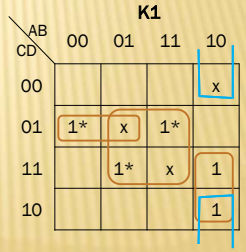
- From the point of view of finding prime implicants, X's (don't care) are treated as 1's

$$F = BD + \overline{A}\overline{C}D \quad (1)$$

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

$$F = BD + \overline{A}\overline{C}D + A\overline{B}C + A\overline{B}\overline{D} \quad (3)$$





48

48

THE KARNAUGH MAP – DON'T CARE

From the point of view of finding prime implicants, X's (don't care) are treated as 1's

- Don't cares (X's) do not make a essential prime implicant
  - There are two esssential prime implicants

$\overline{B}D$ and $\overline{A}CD$

  - The group of four don't cares $\overline{A}\overline{B}$ is a prime implicant but it is not essential (since it does not cover any 1's not covered by some other prime implicant)

$F = \overline{B}D + \overline{A}CD$

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

49

49

THE KARNAUGH MAP

- Minimum POS expressions using K-map
- **Example** $F(A,B,C,D) = \prod M(0,1,4,5,10,11,14,15)$

$F(A,B,C,D) = \sum m(2,3,6,7,8,9,12,13)$

- SOP $F = A\overline{C} + \overline{A}C$
- POS = SOP

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

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

50

50

FIVE-VARIABLE MAP

	A=0				A=1			
DE \ BC	00	01	11	10	10	11	01	00
00	0	4	12	8	24	28	20	16
01	1	5	13	9	25	29	21	17
11	3	7	15	11	27	31	23	19
10	2	6	14	10	26	30	22	18

51

51

FIVE-VARIABLE MAP

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

52

52

QUINE-McCLUSKEY METHOD

- The Quine-McCluskey method is computerized and effective for a larger number of variables than K map
- This method uses the following steps:

1. First, the minterms (and don't cares) of the function are classified into groups so that each term in a group contains the same number of 1s in the binary representation of the term
2. Then the groups formed in step 1 are arranged in the increasing order of number of 1s. Let the number of groups be *n*
3. Each minterm in the group *i* (*i*=0 to *n*-1) is compared with those in group (*i*+1); if the two terms are adjacent, a combined term is formed. The variable thus eliminated is represented as "–"in the combined term

53

53

QUINE-McCLUSKEY METHOD

4. The matching operation of step 3 is repeated on the combined terms until no more combinations can be done. Each combined term in the final list is a PI
5. A PI chart is then constructed, in which there is one column for each minterm (don't cares are not listed) and one row for each PI. An X in a row-column intersection indicates that the PI corresponding to the row covers the minterm corresponding to the column
6. Then all the essential PIs (i.e., the PIs that cover at least one minterm not covered by any other PI) are located
7. A minimum number of PIs from the remaining ones are selected to cover those minterms not covered by the essential PIs
8. The set of PIs thus selected forms the minimum function

54

54

QUINE-McCLUSKEY METHOD

- **Example 1** $F(A,B,C,D,E,F) = \sum m(0,2,6,7,8,10,12,14,15,41)$
- Steps 1, 2:

Each minterm and don't care is expanded into binary form, and groups of terms with the same number of 1s are formed. The groups are arranged in the order of increasing number of 1s. The don't cares are treated as equivalent to minterms until the selection of PIs in step 5

0	000000	Group 0: terms with no 1s
2	000010	Group 1: terms with one 1
8	001000	
6	000110	Group 2: terms with two 1s
10	001010	
12	001100	
7	000111	Group 3: terms with three 1s
14	001110	
41	101001	
15	001111	Group 4: terms with four 1s

55

55

QUINE-McCLUSKEY METHOD

- Steps 3,4:

Each minterm in the group *i* (*i*=0 to *n*-1) is compared with those in group (*i*+1)

✓ 0	000000	✓(0,2)	0000-0	(0,2,8,10)	00-0-0
✓ 2	000010	✓(0,8)	00-000	(2,6,10,14)	00--10
✓ 8	001000	✓(2,6)	000-10	(8,12,10,14)	001--0
✓ 6	000110	✓(2,10)	00-010	(6,7,14,15)	00-11-
✓ 10	001010	✓(8,10)	0010-0		
✓ 12	001100	✓(8,12)	001-00		
✓ 7	000111	✓(6,7)	00011-		
✓ 14	001110	✓(6,14)	00-110		
41	101001	✓(10,14)	001-10		
✓ 15	001111	✓(12,14)	0011-0		
		✓(7,15)	00-111		
		✓(14,15)	00111-		

56

56

QUINE-McCLUSKEY METHOD

- Steps 5,6: Draw a PI chart and select EPI

	✓ 0	✓ 2	✓ 6	✓ 7	✓ 8	✓ 10	✓ 12	✓ 14	✓ 15	✓ 41
*PI_1(41)										⊗
*PI_2(0,2,8,10)	⊗	x			x	x				
PI_3(2,6,10,14)		x	x			x		x		
*PI_4(8,10,12,14)					x	x	⊗	x		
*PI_5(6,7,14,15)			x	⊗				x	⊗	

57

57

QUINE-McCLUSKEY METHOD

- Steps 7. 8: The set of PIs selected forms the minimum function

$$F(A,B,C,D,E,F) = PI_1 + PI_2 + PI_4 + PI_5$$
$$= 101001 + 00\text{-}0\text{-}0 + 001\text{--}0 + 00\text{-}11\text{-}$$
$$= \bar{A}\bar{B}C\bar{D}\bar{E}F + \bar{A}\bar{B}\bar{D}\bar{F} + \bar{A}\bar{B}C\bar{F} + \bar{A}\bar{B}DE$$

58

58

QUINE-McCLUSKEY METHOD

- Example 2

$$F(A,B,C,D) = \bar{A}\bar{B}C\bar{D} + B\bar{C}D + ACD + \bar{A}\bar{B}C\bar{D} + \bar{A}BCD$$

Don't care: $\bar{A}D$

59

59

QUINE-McCLUSKEY METHOD

- Steps 1,2,3,4:

✓0	0000	✓(0,1)	000-	(0,1,2,3)	00--
✓ 1	0001	✓(0,2)	00-0	(1,5,3,7)	0--1
✓ 2	0010	✓(1,3)	00-1	(3,7,11,15)	--11
✓ 3	0011	✓(1,5)	0-01	(5,7,13,15)	-1-1
✓ 5	0101	✓(2,3)	001-		
✓ 7	0111	✓(3,7)	0-11		
✓ 11	1011	✓(3,11)	-011		
✓ 13	1101	✓(5,7)	01-1		
✓ 15	1111	✓(5,13)	-101		
		✓(7,15)	-111		
		✓(11,15)	1-11		
		✓(13,15)	11-1		

60

60

QUINE-McCLUSKEY METHOD

- Steps 5,6,7,8:

	✓0	✓2	✓ 11	✓ 13	✓ 15
*PI_1 (0,1,2,3)	⊗	⊗			
PI_2 (1,3,5,7)					
*PI_3 (3,7,11,15)			⊗		x
*PI_4 (5,7,13,15)				⊗	x

$$F(A,B,C,D) = PI_1 + PI_3 + PI_4 = 00--+--11+-1-1$$
$$= \overline{A}\overline{B} + CD + BD$$

61

61

QUINE-McCLUSKEY METHOD

- Example 3

$$F(A,B,C,D) = \sum m(0,2,4,5,6,9,10) + \sum d(7,11,12,13,14,15)$$

62

62

QUINE-McCLUSKEY METHOD

- Steps 1 and 2

0	0000	Group 0: terms with no 1s
2	0010	Group 1: terms with one 1
4	0100	
5	0101	Group 2: terms with two 1s
6	0110	
9	1001	
10	1010	
12	1100	
7	0111	Group 3: terms with three 1s
11	1011	
13	1101	
14	1110	
15	1111	Group 4: terms with four 1s

63

63

QUINE-McCLUSKEY METHOD

- Step 3

✓	0	0000	✓(0,2)	00-0	Obtained by matching groups 0 and 1
✓	2	0010	✓(0,4)	0-00	
✓	4	0100	✓(2-6)	0-10	Obtained by matching groups 1 and 2
✓	5	0101	✓(2-10)	-010	
✓	6	0110	✓(4-5)	010-	
✓	9	1001	✓(4-6)	01-0	
✓	10	1010	✓(4-12)	-100	
✓	12	1100	✓(5-7)	01-1	Obtained by matching groups 2 and 3
✓	7	0111	✓(5-13)	-101	
✓	11	1011	✓(6-7)	011-	
✓	13	1101	✓(6-14)	-110	
✓	14	1110	✓(9-11)	10-1	
✓	15	1111	✓(9-13)	1-01	
			✓(10-11)	101-	
			✓(10-14)	1-10	
			✓(12-13)	110-	
			✓(12-14)	11-0	
			✓(7-15)	-111	Obtained by matching groups 3 and 4
			✓(11-15)	1-11	
			✓(13-15)	11-1	
			✓(14-15)	111-	

64

64

QUINE-McCLUSKEY METHOD

- Step 4

(0,2,4,6)	0--0	Same as (0,4,2,6)
(2,6,10,14)	--10	Same as (2,10,6,4)
✓(4,5,6,7)	01--	Same as (4,6,5,7)
✓(4,5,12,13)	-10-	Same as (4,12,5,13)
✓(4,6,12,14)	-1-0	Same as (4,12,6,14)
✓(5,7,13,15)	-1-1	Same as (5,13,7,15)
✓(6,7,14,15)	-11-	Same as (6,14,7,15)
(9,11,13,15)	1--1	Same as 9,13,11,15)
(10,11,14,15)	1-1-	Same as (10,14,11,15)
✓(12,13,14,15)	11--	Same as (12,14,13,15)

65

65

QUINE-McCLUSKEY METHOD

- **Step 4**: To repeat the comparison between the three group obtained to derive additional combinations. No combinations are possible between the first two groups. (2,6,10,14) in the 2nd group cannot be combined with any term in the 3rd group. Two term in the 3rd group do not combine with any term.

(0,2,4,6)	0--0		
(2,6,10,14)	--10	(4,5,12,13,6,7,14,15)	-1--
✓ (4,5,6,7)	01--	(4,6,12,14,5,7,13,15)	-1--
✓(4,5,12,13)	-10-	(4,5,6,7,12,13,14,15)	-1--
✓(4,6,12,14)	-1-0		
✓(5,7,13,15)	-1-1		
✓(6,7,14,15)	-11-		
(9,11,13,15)	1--1		
(10,11,14,15)	1-1-		
✓ (12,13,14,15)	11--		

66

66

QUINE-McCLUSKEY METHOD

- Step 4:

PI_1(0,2,4,6)	0--0
PI_2(2,6,10,14)	--10
PI_3(4,5,6,7,12,13,14,15)	-1--
PI_4(9,11,13,15)	1--1
PI_5(10,11,14,15)	1-1-

- No other combinations are possible. Hence, the five terms above are PI

67

67

QUINE-McCLUSKEY METHOD

- **Step 5 and 6:** To draw a PI chart in which one row corresponding to each PI and one column corresponding to each term. Don't cares are ignored

	✓ 0	✓ 2	✓ 4	✓ 5	✓ 6	✓ 9	✓ 10
*PI_1: 0--0	⊗	x	x		x		
PI_2: --10		x			x		x
*PI_3: -1--			x	⊗	x		
*PI_4: 1--1						⊗	
PI_5: 1-1-							x

- PI_1, PI_3,PI_4 are essential

68

68

QUINE-McCLUSKEY METHOD

- **Step 7:** Once PI_1, PI_3 and PI_4 are selected, all the minterms except 10 are covered. To covered the minterm 10 we can select either PI_2 or PI_5, since they contribute the same number of literals
- **Step 8**: The reduced function is

$$F(A,B,C,D) = PI_1 + PI_3 + PI_4 + PI_2 \text{ or } PI_5$$
$$= 0--0 \; + \; -1-- \; + \; 1--1 \; + \; --10 \text{ OR } 1-1-$$
$$= \overline{AD} \; + \; B \; + \; AD + C\overline{D} \text{ or } AC$$

69

69

DIGITAL LOGIC IMPLEMENTATION

- Logic circuits that perform logical operations such as AND, OR, and NOT are called gates
- A gate is a block of hardware that produces a logic 0 or a logic 1 output signal in response to binary signals applied to its inputs
- Eight functions are implemented as standard logic gates: AND, OR, NOT (complement), transfer, NAND, NOR, XOR and equivalence (XNOR)

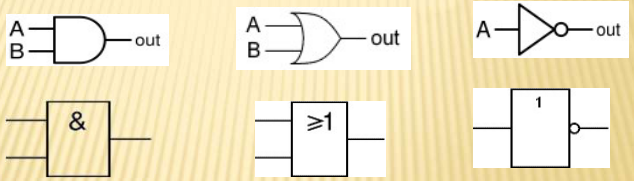
70

70

DIGITAL LOGIC IMPLEMENTATION

- Graphic symbols of the eight standard gates

AND OR NOT



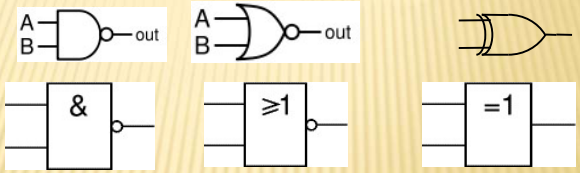
71

71

DIGITAL LOGIC IMPLEMENTATION

- Graphic symbols of the eight standard gates

NAND NOR XOR



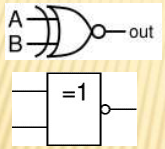
72

72

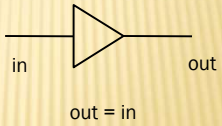
DIGITAL LOGIC IMPLEMENTATION

- Graphic symbols of the eight standard gates

XNOR



Buffer (Transfer)

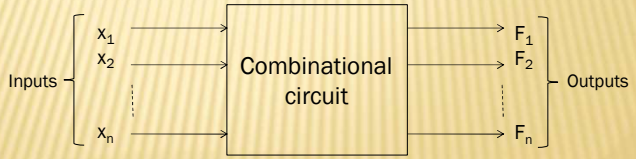


73

73

3. COMBINATIONAL LOGIC CIRCUITS

- Digital systems are classified either as combinational systems or sequential systems
- In a combinational system, the output signals depend only on the current input signals

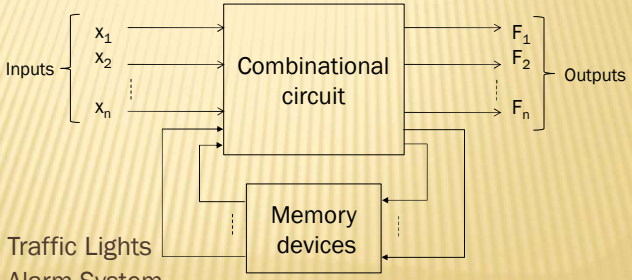


74

74

COMBINATIONAL LOGIC CIRCUITS

- In a sequential system, the output signals depend on the current input signals and the previous signals



- 1) Traffic Lights
- 2) Alarm System

75

75

COMBINATIONAL LOGIC CIRCUITS

- **Design Procedure**
- The design of combinational circuits starts with a word description of the problem and ends with a circuit logic diagram. In general, the design procedure involves the following steps:

1. State the problem
2. Assign letter symbols to the input variables and the output functions
3. Derive the truth tables that define the relationship between the inputs and the outputs
4. Obtain Boolean expressions for each output
5. Simplify (minimize) the Boolean expressions
6. Draw the circuit logic diagram

76

76

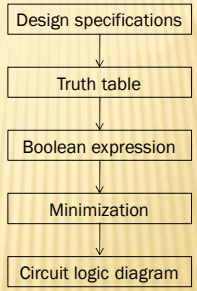
- TWO PROBLEMS for Digital Systems:
- - Design (Synthesis): Function -> Logic Circuit ?
- - Analysis: Logic Circuit -> Function ?

77

77

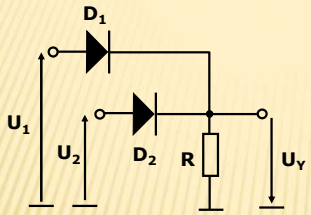
COMBINATIONAL LOGIC CIRCUITS

- Design flow diagram



78

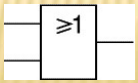
78



U_1, U_2 = 0 or E v

$U_1 \Leftrightarrow A$, $U_2 \Leftrightarrow B$, $U_Y \Leftrightarrow F(A,B)$

0v⇔0, Ev⇔1



U_1	U_2	U_Y
0	0	0
0	E	E
E	0	E
E	E	E

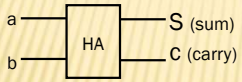
A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

79

79

COMBINATIONAL LOGIC CIRCUITS

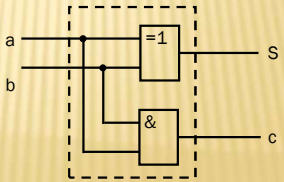
- Arithmetic Circuits
  - + Adders
    - × Half-adder (HA)



a	b	S	c
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$S = a \oplus b$

$c = ab$

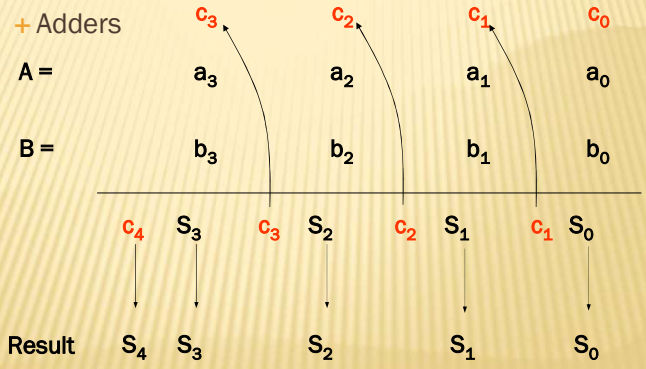


80

80

COMBINATIONAL LOGIC CIRCUITS

- Adders

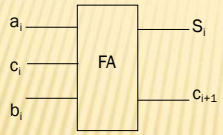


81

81

COMBINATIONAL LOGIC CIRCUITS

- Adders
  - Full-adder (FA)



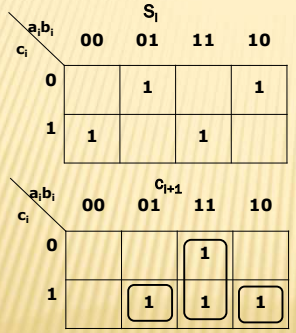
a_i	b_i	c_i	S_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

82

82

COMBINATIONAL LOGIC CIRCUITS

- Full-adder (FA)



$S_i = a_i \oplus b_i \oplus c_i$

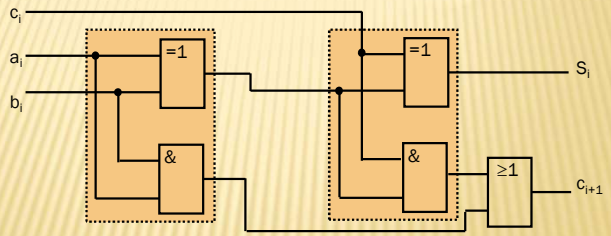
$c_{i+1} = a_i\, b_i + c_i\, (a_i \oplus b_i)$

83

83

COMBINATIONAL LOGIC CIRCUITS

- Full-adder (FA)



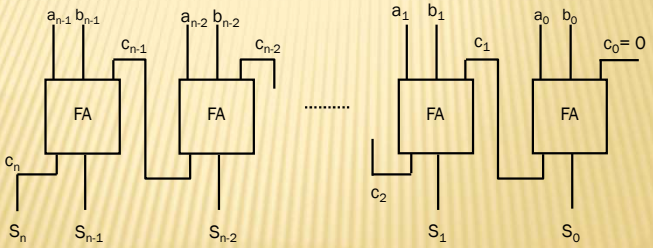
84

84

COMBINATIONAL LOGIC CIRCUITS

- n-bit parallel adder

  $A = a_{n-1}a_{n-2}...a_1a_0$, $B = b_{n-1}b_{n-2}...b_1b_0$



85

85

COMBINATIONAL LOGIC CIRCUITS

- Look-Ahead Carry Adders

  For n-bit parallel adder, the carry signal must propagate through the n full-adder modules. The values at the output terminals will not be correct unless the signals are given enough time to propagate through the circuit.

  The carry propagation time in the parallel adder is a limit factor on the speed with which two numbers can be added.

86

86

COMBINATIONAL LOGIC CIRCUITS

- Look-Ahead Carry Adders

  To speed up the addition process, a look-ahead carry adder, resulting in carry propagation time independent of n is used widely.

  The basic idea is that all the carries can be generated simultaneously rather than successively.

$$c_{i+1} = a_ib_i + c_i(a_i \oplus b_i) = G_i + c_iP_i$$
$$P_i = (a_i \oplus b_i)$$
$$G_i = a_ib_i$$

87

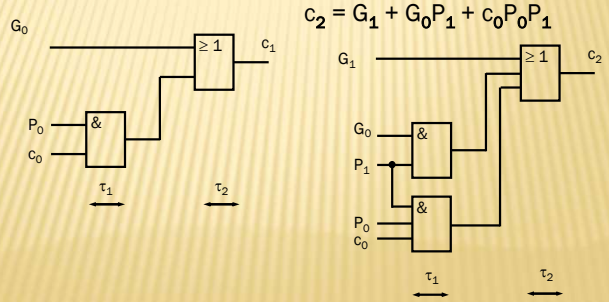
87

COMBINATIONAL LOGIC CIRCUITS

- Look-Ahead Carry Adders

$c_1 = G_0 + c_0P_0$ $\quad c_2 = G_1 + c_1P_1 = G_1+(G_0 + c_0P_0)P_1$

$c_2 = G_1 + G_0P_1 + c_0P_0P_1$

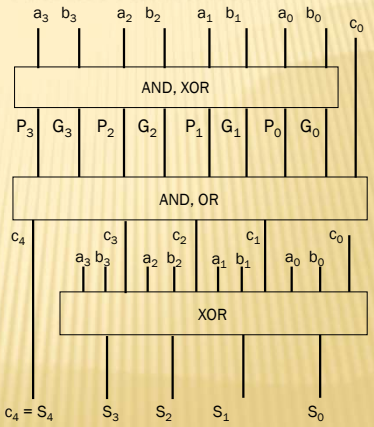


88

88

COMBINATIONAL LOGIC CIRCUITS

- Four-bit Look-Ahead Carry Adder

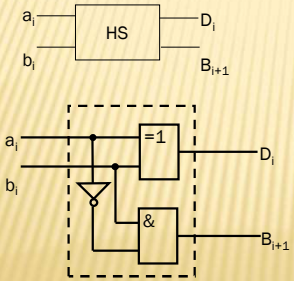


89

89

COMBINATIONAL LOGIC CIRCUITS

- Subtractor
  - Half-Subtractor (HS)



a_i	b_i	D_i	B_{i+1}
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

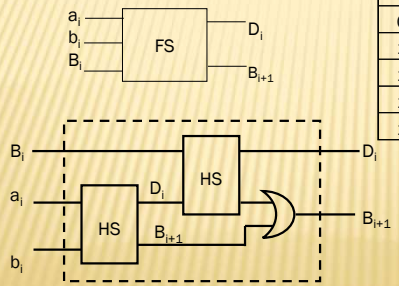
$$D_i = a_i \oplus b_i$$
$$B_{i+1} = \overline{a_i}\, b_i$$

90

90

COMBINATIONAL LOGIC CIRCUITS

- Subtractor
  - Full-Subtractor (FS)



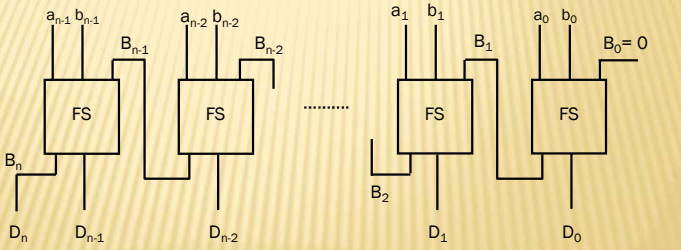
a_i	b_i	B_i	D_i	B_{i+1}
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

91

91

COMBINATIONAL LOGIC CIRCUITS

- n-bit parallel subtractor



92

92

COMBINATIONAL LOGIC CIRCUITS

- **Multiplier**

  Multiplication can be carried out in two steps: (1) Compute the partial products (2) sum the partial products

				a_3	a_2	a_1	a_0	multiplicand
				b_3	b_2	b_1	b_0	multiplier
				a_3b_0	a_2b_0	a_1b_0	a_0b_0	
			a_3b_1	a_2b_1	a_1b_1	a_0b_1		← left-shift
		a_3b_2	a_2b_2	a_1b_2	a_0b_2			
	a_3b_3	a_2b_3	a_1b_3	a_0b_3				
p_7	p_6	p_5	p_4	p_3	p_2	p_1	p_0	product

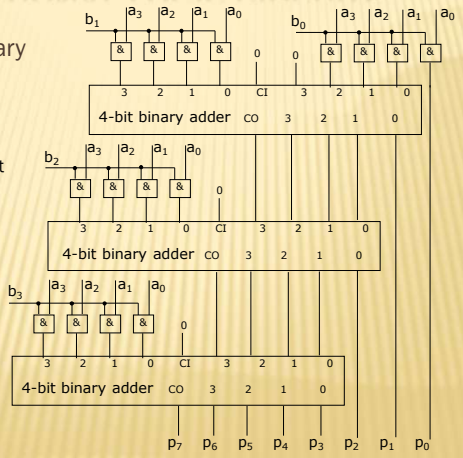
93

93

COMBINATIONAL LOGIC CIRCUITS

- Four-bit binary multiplier

CI: Carry Input
CO: Carry Output



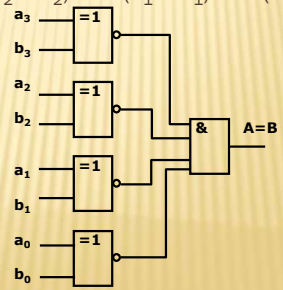
94

94

COMBINATIONAL LOGIC CIRCUITS

- Comparator: An n-bit comparator is a circuit that compares the magnitude of two n-bit binary numbers A and B
- Simple comparator:

  A=B if ($a_3 = b_3$) and ($a_2 = b_2$) and ($a_1 = b_1$) and ($a_0 = b_0$)



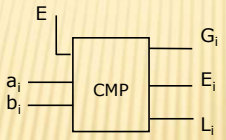
95

95

COMBINATIONAL LOGIC CIRCUITS

- Parallel comparator

  The circuit has three output functions designated by L, E, and G, corresponding to A<B, A=B, and A>B. In other words, L=1 if only if A<B, E=1 if only if A=B, and G=1 if only if A>B



E=1: Enable
E=0: Gi=Ei=Li=0

96

96

COMBINATIONAL LOGIC CIRCUITS

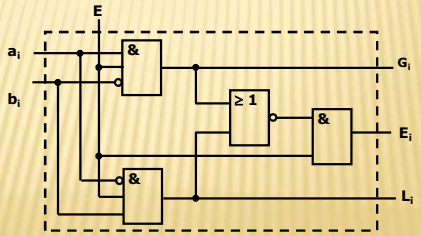
E	a_i	b_i	E_i $a_i=b_i$	G_i $a_i>b_i$	L_i $a_i<b_i$
0	0	0	0	0	0
0	0	1	0	0	0
0	1	0	0	0	0
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	1
1	1	0	0	1	0
1	1	1	1	0	0

97

97

COMBINATIONAL LOGIC CIRCUITS

$$G_i = E(a_i\bar{b}_i)$$
$$L_i = E(\bar{a}_i b_i)$$
$$E_i = E(\overline{a_i \oplus b_i}) = Ea_ib_i + E\bar{a}_i\bar{b}_i = E.\bar{G}_i.\bar{L}_i = E(\overline{G_i + L_i})$$

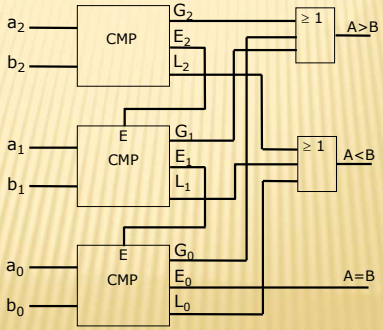


98

98

COMBINATIONAL LOGIC CIRCUITS

- Three-bit parallel comparator



99

99

COMBINATIONAL LOGIC CIRCUITS

- Decoders

  A decoder is a device that, when activated, selects one of several output lines, based on a coded input signal.

  Most commonly, the input is an n-bit binary number, and there are 2^n output lines.
- Two-input (four-output) decoder
  - An active high decoder

a	b	0	1	2	3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

An active low decoder

a	b	0	1	2	3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

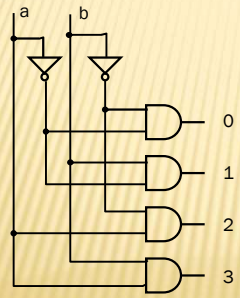
100

100

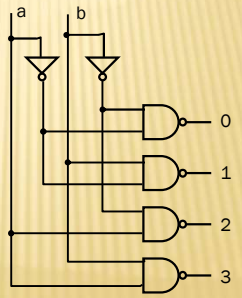
COMBINATIONAL LOGIC CIRCUITS

- Decoders

An active high decoder



An active low decoder



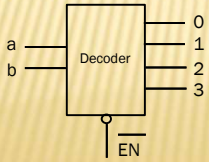
101

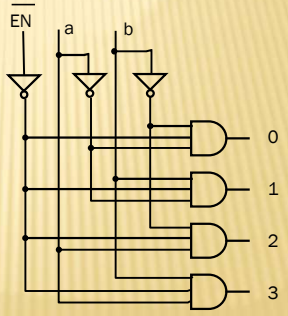
101

COMBINATIONAL LOGIC CIRCUITS

- Decoder with enable

$\overline{EN}$	a	b	0	1	2	3
1	x	x	0	0	0	0
0	0	0	1	0	0	0
0	0	1	0	1	0	0
0	1	0	0	0	1	0
0	1	1	0	0	0	1





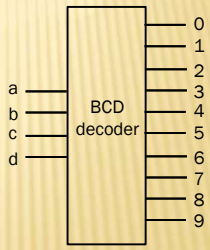
102

102

COMBINATIONAL LOGIC CIRCUITS

- BCD Decoder

a	b	c	d	0	1	...	9
0	0	0	0	1	0	...	0
0	0	0	1	0	1	...	0
0	0	1	0	0	0	...	0
0	0	1	1	0	0	...	0
0	1	0	0	0	0	...	0
0	1	0	1	0	0	...	0
0	1	1	0	0	0	...	0
0	1	1	1	0	0	...	0
1	0	0	0	0	0	...	0
1	0	0	1	0	0	...	1



103

103

COMBINATIONAL LOGIC CIRCUITS

- **Encoders**

  An encoder is a circuit that performs the function of a decoder in reverse. It has *m* inputs and *n* outputs, where $m \leq 2^n$. The outputs generate the binary codes for the *m* input variables. Only one input should be 1 at a time.

- **Example**

  An octal-to-binary (8x3) encoder has 8 inputs, one for each of the eight octal digits, and 3 outputs that generate the corresponding binary numbers.

  It is assumed that external conditions prevent any input other than those that represent the eight octal digits (2^8-8=248 input combinations cannot happen)

104

104

COMBINATIONAL LOGIC CIRCUITS

- An octal-to-binary (8x3) encoder

D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x_2	x_1	x_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

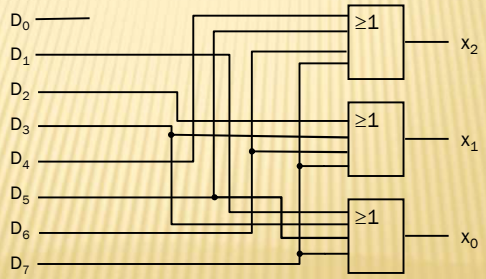
105

105

COMBINATIONAL LOGIC CIRCUITS

- An octal-to-binary (8x3) encoder

$x_2=D_4+D_5+D_6+D_7$ $x_1=D_2+D_3+D_6+D_7$ $x_0=D_1+D_3+D_5+D_7$



106

106

COMBINATIONAL LOGIC CIRCUITS

- Priority encoders

If more than one input can occur at the same time, then some priority must be established. The priorities are normally arranged in descending (or ascending) order with the highest priority given to the largest (smallest) input number.

107

107

COMBINATIONAL LOGIC CIRCUITS

- Priority encoders

The truth table for an eight-input priority encoder

A_0	A_1	A_2	A_3	A_4	A_5	A_6	A_7	Z_2	Z_1	Z_0	NR
0	0	0	0	0	0	0	0	x	x	x	1
x	x	x	x	x	x	x	1	1	1	1	0
x	x	x	x	x	x	1	0	1	1	0	0
x	x	x	x	x	1	0	0	1	0	1	0
x	x	x	x	1	0	0	0	1	0	0	0
x	x	x	1	0	0	0	0	0	1	1	0
x	x	1	0	0	0	0	0	0	1	0	0
x	1	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0

108

108

COMBINATIONAL LOGIC CIRCUITS

- Priority encoders

  The output NR indicates that there are no requests and we don't care what the other outputs are.

  If device 7 has an active signal (that is, a 1), then the output is the binary for 7, regardless of what the other inputs are. Only when A_7=0, any other input will be recognized.

  The equations describing this device are

$$NR = \overline{A}_0\overline{A}_1\overline{A}_2\overline{A}_3\overline{A}_4\overline{A}_5\overline{A}_6\overline{A}_7$$

$$Z_2 = A_4 + A_5 + A_6 + A_7$$

$$Z_1 = A_6 + A_7 + (A_2 + A_3)\overline{A}_4\overline{A}_5$$

$$Z_0 = A_7 + A_5\overline{A}_6 + A_3\overline{A}_4\overline{A}_6 + A_1\overline{A}_2\overline{A}_4\overline{A}_6$$

109

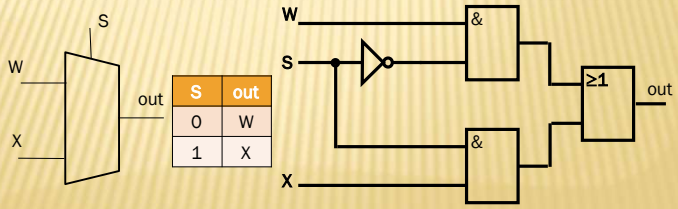
109

COMBINATIONAL LOGIC CIRCUITS

- Multiplexers

  A multiplexer is basically a switch that passes one of its data input through to the output.

  Two-way multiplexer



S	out
0	W
1	X

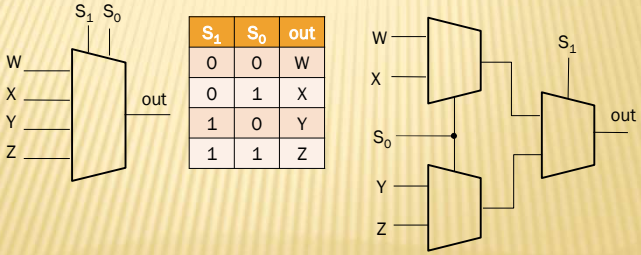
110

110

COMBINATIONAL LOGIC CIRCUITS

- Multiplexers

  Four-way multiplexer



S_1	S_0	out
0	0	W
0	1	X
1	0	Y
1	1	Z

111

111

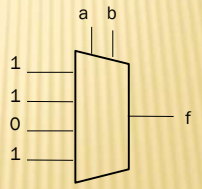
COMBINATIONAL LOGIC CIRCUITS

- Multiplexers

  Multiplexers can be used to implement logic funtions.

  Example Implement the function $f(a,b)=\Sigma m(0,1,3)$

a	b	f
0	0	1
0	1	1
1	0	0
1	1	1



112

112

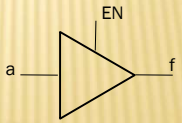
COMBINATIONAL LOGIC CIRCUITS

- Three-state (tristate) gates

  In a three-state gate, there is an enable input. If that input is active (it could be active high or active low) the gate behaves as usual. If it is inactive, the output behaves as if the output is not connected (as an open circuit)

  Three-state buffer with an active high

EN	a	f
0	0	Z
0	1	Z
1	0	0
1	1	1

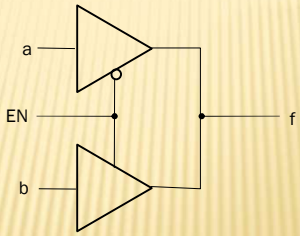


113

113

COMBINATIONAL LOGIC CIRCUITS

- A multiplexer using three-state gates

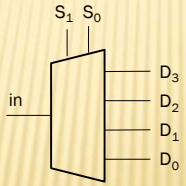


114

114

COMBINATIONAL LOGIC CIRCUITS

- **Demultiplexer**. A demultiplexer routes data from a single source to one of several outputs, it performs the opposite function of a multiplexer.
- **Example** 1x4 demultiplexer

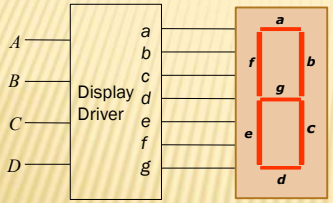


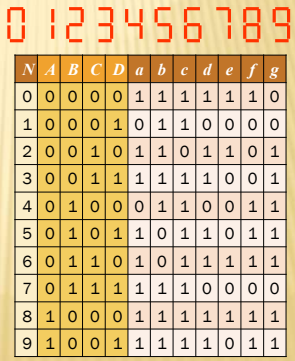
115

115

COMBINATIONAL LOGIC CIRCUITS

- Larger examples
  - Seven-Segment

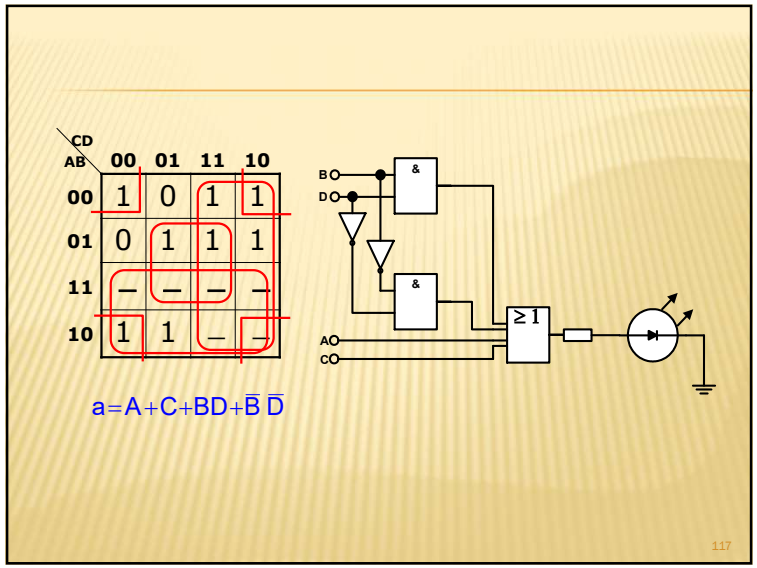




N	A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

116

116



117

117

COMBINATIONAL LOGIC CIRCUITS

- Larger examples
  - An error coding system

When data is transmitted (or stored), errors occur. Hamming developed a technique for coding data (by adding extra digits) so that a single error (that is, an error in 1 bit)can be corrected.

To detect an error in a set of bits, a *check bit* is created so that the total number of 1's in the word, including the check bit, is even. That bit is referred to as a *parity bit*.

If one error is made, either a 1 will become a 0, or a 0 will become a 1, making the total number of 1's odd.

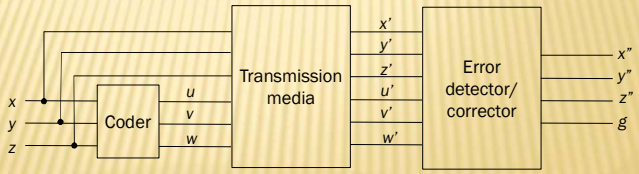
118

118

COMBINATIONAL LOGIC CIRCUITS

- Larger examples
  - An error coding system

Consider three data bits and three check bits, which provide for the correction of all single errors and detection of some double errors.



119

119

COMBINATIONAL LOGIC CIRCUITS

- The first check bit, u, checks x and y.

  v checks x and z, w checks y and z

  $u = x \oplus y$ $v = x \oplus z$ $w = y \oplus z$

Transmitted words

Data			Check		
x	y	z	u	v	w
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	1	0	1
0	1	1	1	1	0
1	0	o	1	1	0
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	0	0	0

120

120

COMBINATIONAL LOGIC CIRCUITS

Computing the Exclusive-OR of each check bit with the bits that formed it, we get

$t_1 = x' \oplus y' \oplus u'$ $t_2 = x' \oplus z' \oplus v'$ $t_3 = y' \oplus z' \oplus w'$

That test word indicates which bit is in error (if a single error was made)

t_1	t_2	t_3	Error
0	0	0	none
0	0	1	w'
0	1	0	v'
0	1	1	z'
1	0	o	u'
1	0	1	y'
1	1	0	x'
1	1	1	multiple

121

121

COMBINATIONAL LOGIC CIRCUITS

- xyz=000, uvw=000, $x'y'z'$=001, $u'v'w'$=000
  $t_1 = x' \oplus y' \oplus u' = 0$ $t_2 = x' \oplus z' \oplus v' = 1$
  $t_3 = y' \oplus z' \oplus w' = 1$
- xyz=000, uvw=000, $x'y'z'$=000, $u'v'w'$=001
  $t_1 = x' \oplus y' \oplus u' = 0$ $t_2 = x' \oplus z' \oplus v' = 0$
  $t_3 = y' \oplus z' \oplus w' = 1$
- xyz=000, uvw=000, $x'y'z'$=100, $u'v'w'$=000
  $t_1 = x' \oplus y' \oplus u' = 1$ $t_2 = x' \oplus z' \oplus v' = 1$
  $t_3 = y' \oplus z' \oplus w' = 0$
- xyz=000, uvw=000, $x'y'z'$=110, $u'v'w'$=000
  $t_1 = x' \oplus y' \oplus u' = 1$ $t_2 = x' \oplus z' \oplus v' = 1$
  $t_3 = y' \oplus z' \oplus w' = 1$

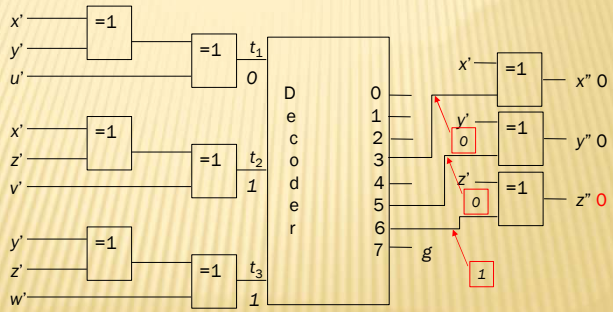
t_3	t_2	t_1	Error
0	0	0	none
0	0	1	u'
0	1	0	v'
0	1	1	x'
1	0	0	w'
1	0	1	y'
1	1	0	z'
1	1	1	multiple

122

122

COMBINATIONAL LOGIC CIRCUITS

Error decoder xyz=000, uvw=000, $x'y'z'$=001, $u'v'w'$=000

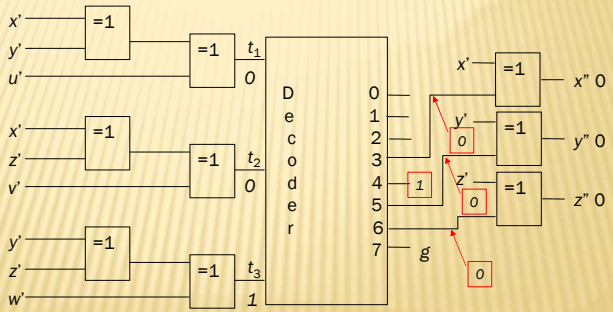


123

123

COMBINATIONAL LOGIC CIRCUITS

Error decoder xyz=000, uvw=000, $x'y'z'$=000, $u'v'w'$=001

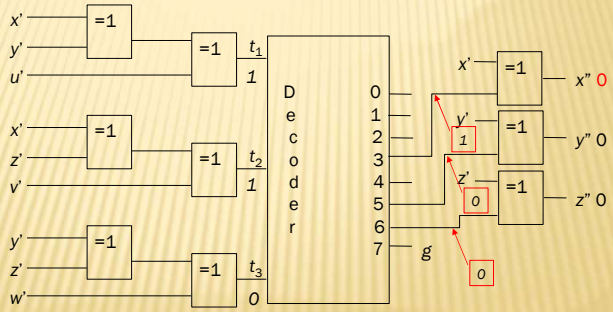


124

124

COMBINATIONAL LOGIC CIRCUITS

Error decoder *xyz=000, uvw=000, x'y'z'=100, u'v'w'=110*

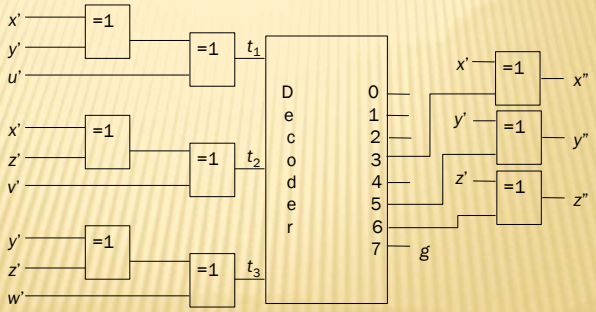


125

125

COMBINATIONAL LOGIC CIRCUITS

Error decoder

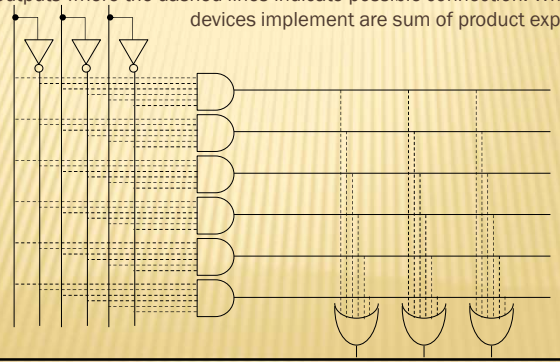


126

126

GATE ARRAYS-ROMs, PLAs AND PALs

- Gate arrays are one approach to the rapid implementation of fairly complex system. The basic concept is illustrated for a system with 3 inputs and 3 outputs where the dashed lines indicate possible connection. What these devices implement are sum of product expression



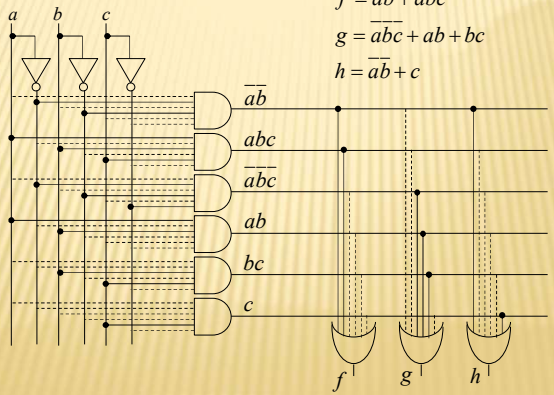
127

127

GATE ARRAYS-ROMs, PLAs AND PALs

- The circuit shows the implement of

$$f = \overline{\overline{ab}} + abc$$
$$g = \overline{\overline{abc}} + ab + bc$$
$$h = \overline{\overline{ab}} + c$$



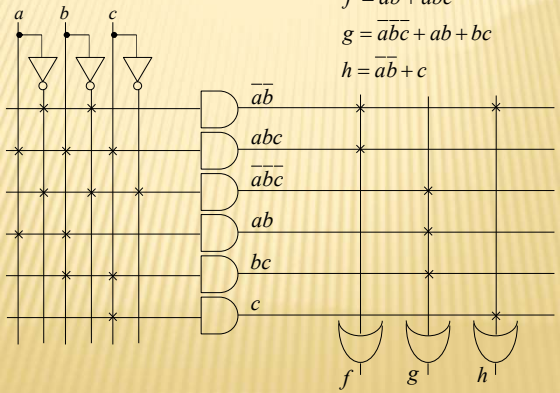
128

128

GATE ARRAYS-ROMs, PLAs AND PALs

- The circuit shows the implement of

$$f = \bar{a}\bar{b} + abc$$
$$g = \bar{a}\bar{b}\bar{c} + ab + bc$$
$$h = \bar{a}\bar{b} + c$$

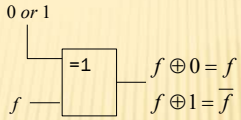


129

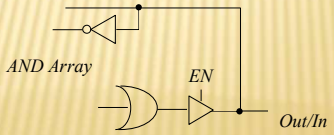
129

GATE ARRAYS-ROMs, PLAs AND PALs

- A programmable output circuit



- Three-state output



130

130

COMBINATIONAL CIRCUITS

- Design problem

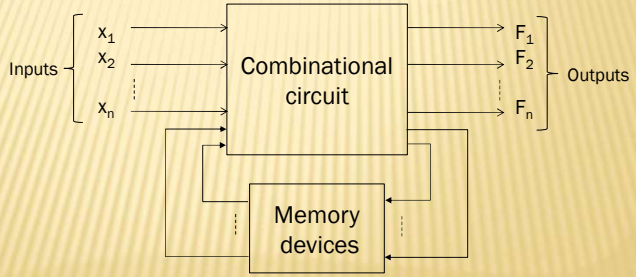
1) Compare two 2-bit numbers A = (A1, A0) and B = (B1, B0), produce three signals G, L, and E so that G is 1 only when A > B, L is 1 only when A < B, and E is 1 only when A = B
2) Pass or fail:There are three components in a course: homework (H), lab(L), and exam (E). You pass the course (P) only if you pass two or more components

131

131

4. SEQUENTIAL SYSTEMS

- In a sequential system, the output signals depend on the current input signals and the previous signals

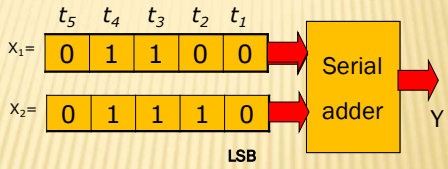


132

132

SEQUENTIAL SYSTEMS

- **Example** A serial binary adder with two inputs X_1, X_2 and output Y.



	t_5	t_4	t_3	t_2	t_1
X_1 =	0	1	1	0	0
X_2 =	0	1	1	1	0
Y =	1	1	0	1	0

133

133

SEQUENTIAL SYSTEMS

- A sequential system is defined by the quintuple $M = (X,S, Y, F_s, F_Y)$, where

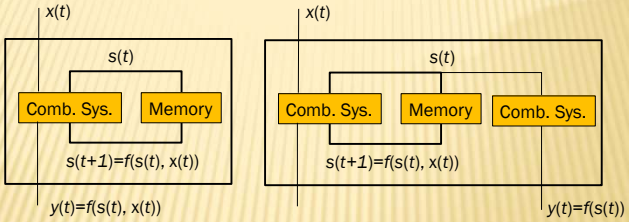
1. X - finite non-empty set of input symbols $x_1, \ldots, x_m$
2. S - finite non-empty set of states $s_1, \ldots, s_n$,
3. Y - finite non-empty set of output symbols $y_1, \ldots, y_k$,
4. F_s - state function,
5. F_Y - output function.

134

134

SEQUENTIAL SYSTEMS

- There are two basic sequential systems: Mealy and Moore



- A sequential circuit in which the outputs depend on both the circuit inputs and the present state is called a Mealy circuit
- When the outputs depend only on the present state, the circuit is called a Moore circuit

135

135

SEQUENTIAL SYSTEMS

- **Example** A Mealy sequential system for sequential binary adder

The input symbols $X=\{x_1x_2\} = \{00,01,10,11\}$
The output symbols $Y= \{0,1\}$
The set of states $S= \{s_0,s_1\}$, s_0 – no carry, s_1 – carry
The state function F_S
$F_s(s_0, 11) = s_1$,
$F_s(s_0, 00) = F_s(s_0, 01) = F_s(s_0, 10) = s_0$,
$F_s(s_1, 01) = F_s(s_1, 10) = F_s(s_1, 11) = s_1$,
$F_s(s_1, 00) = s_0$.
The output functions
$Fy(s_0, 00) = Fy(s_0, 11) = 0$, $Fy(s_0, 01) = Fy(s_0, 10) = 1$,
$Fy(s1, 00) = Fy(s_1, 11) = 1$, $Fy(s_1, 01) = Fy(s_1, 10) = 0$.

136

136

SEQUENTIAL SYSTEMS

- **Example** A Moore sequential system for sequential binary adder

  The input symbols $X=\{x_1x_2\} = \{00,01,10,11\}$

  The output symbols $Y= \{0,1\}$

  The set of states $S= \{s_{00},s_{01}, s_{10},s_{11}\}$

  The state s_{00} denotes the combination when there is no carry and sum is 0. Similarly, the state s_{01} denotes that there is no carry, but the sum is 1, etc.

  The state function F_S

  $Fs(s_{00}, 00) = Fs(s_{01}, 00) = s_{00}$,

  $Fs(s_{00}, 11) = Fs(s_{01}, 11) = s_{10}$,...

  The output functions

  $Fy(s_{00}) = Fy(s_{10}) = 0$,

  $Fy(s_{01}) = Fy(s_{11}) = 1$.

137

137

CLASSIFICATION OF SEQUENTIAL SYSTEMS

- Synchronous versus Asynchronous

  The timing of the signals in the circuit determine two types of sequential circuits.

  In a **synchronous** circuits, the state can change only at discrete instants of time. The circuit uses a timing device , called a **clock generator**, that produces trains of periodic or aperiodic **clock pulses.** The clock pulses are input to the memory devices so that they can change state *only* in response to the arrival of a pulse and only once for each pulse occurrence.

  The behavior of an **asynchronous** sequential circuit depends *only* on the order input change and can be affected at any instant of time.

138

138

REPRESENTATION OF SEQUENTIAL CIRCUITS

- State Table

  The state table presents in a tabular form the operation of a sequential circuit

  Example Mealy state table for sequential binary adder

PS	NS, Output			
	x_1x_2			
	00	01	11	10
S_0	S_0,0	S_0,1	S_1,0	S_0,1
S_1	S_0,1	S_1,0	S_1,1	S_1,0

139

139

REPRESENTATION OF SEQUENTIAL CIRCUITS

- State Table

  Example Moore state table for sequential binary adder

PS	NS				Output
	x_1x_2				
	00	01	11	10	
S_{00}	S_{00}	S_{01}	S_{10}	S_{01}	0
S_{01}	S_{00}	S_{01}	S_{10}	S_{01}	1
S_{10}	S_{01}	S_{10}	S_{11}	S_{10}	0
S_{11}	S_{01}	S_{10}	S_{11}	S_{10}	1

140

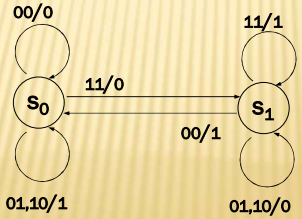
140

REPRESENTATION OF SEQUENTIAL CIRCUITS

- State Diagram

  The state diagram depicts graphically the same information contained in the state table

  Example The Mealy state diagram for sequential binary adder



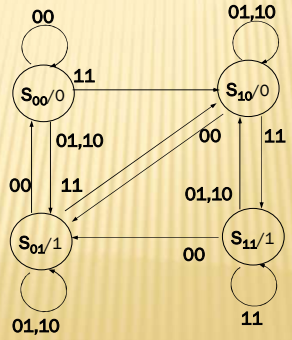
141

141

REPRESENTATION OF SEQUENTIAL CIRCUITS

- State Diagram

  Example The Moore state diagram for sequential binary adder



142

142

MEMORY DEVICES

- The memory part in sequential circuits is mostly implemented with bistable devices.
- A bistable device has two stable states. It can remain in either one of them indefinitely until directed by an input signal to change state. Usually, the device has two complementary outputs designated by Q and $\overline{Q}$, The two stable states are Q=1 ($\overline{Q}$=0) and Q=0 ($\overline{Q}$=1)
- Two types of bistable devices: latches and flip-flops

143

143

MEMORY DEVICES

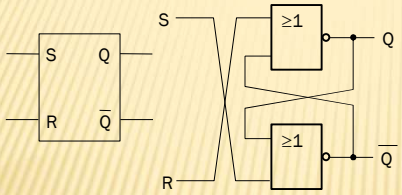
- A latch changes state when the input values change. The new output value is delayed only by the propagation time delays of the gate between the input and the outputs of the latch. This property is called the *transparency* property
- Flip-flops do not have the transparency property.
- A flip-flop has a control (triggering) input, called a clock, and can change state only in response to a transition of a clock pulse at this input.

144

144

MEMORY DEVICES

- SR latch



q \ SR	00	01	11	10
0	0	0	–	1
1	1	0	–	1

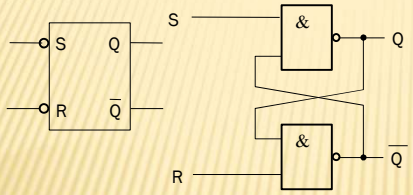
- The latch requires a logic 1 at the inputs to change its state, it is called an active-HIGH latch.
- Characteristic equation $Q = S + \overline{R}q$

145

145

MEMORY DEVICES

- Active-LOW SR latch



q \ SR	00	01	11	10
0	–	1	0	0
1	–	1	1	0

- Characteristic equation

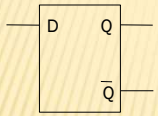
$$Q = \overline{S} + Rq$$

146

146

MEMORY DEVICES

- D latch



q \ D	0	1
0	0	1
1	0	1

- Characteristic equation

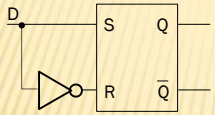
$$Q = D$$

147

147

MEMORY DEVICES

- Implementation using SR latch



q \ D	0	1
0	0	1
1	0	1

q \ D	0 SR	1 SR
0	0–	10
1	01	–0

- Excitation equation

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

148

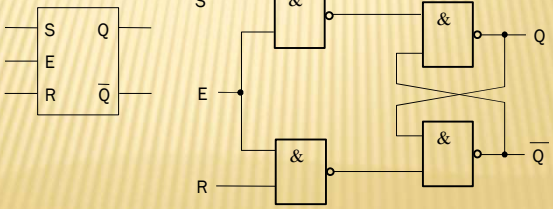
148

MEMORY DEVICES

- Gated Latches

  In many applications, data must be entered into the latch only when some control signal becomes active.

  For SR latch, the enable (E) input controls the operation of the circuit so that the latch will not change state as long as E=0. When E=1, the latch operates normally.

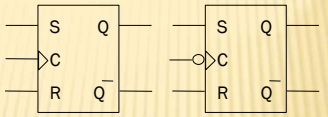


149

149

MEMORY DEVICES

- SR Flip-Flop



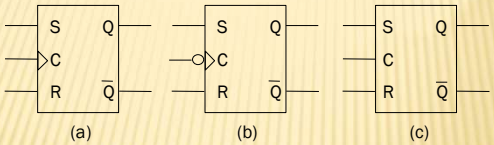
- SR flip-flop is edge-trigerred meaning the flip-flop is sensitive to its S and R input signals either at the positive edge (a) or at the negative edge (b) of the clock pulse

150

150

MEMORY DEVICES

- SR Flip-Flop



- SR flip-flop is edge-trigerred meaning the flip-flop is sensitive to its S and R input signals either at the positive edge (a) or at the negative edge (b) of the clock pulse
- Pulse-triggered (c)(or master-slave)

151

151

MEMORY DEVICES

- SR Flip-Flop

State Table

q \ SR	00	01	11	10
0	0	0	–	1
1	1	0	–	1

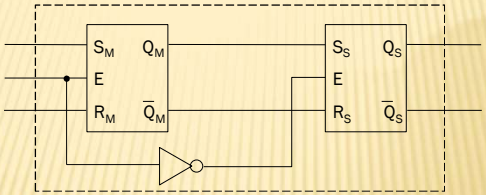
Excitation Table

q	Q	S	R
0	0	0	–
0	1	1	0
1	0	0	1
1	1	–	0

152

152

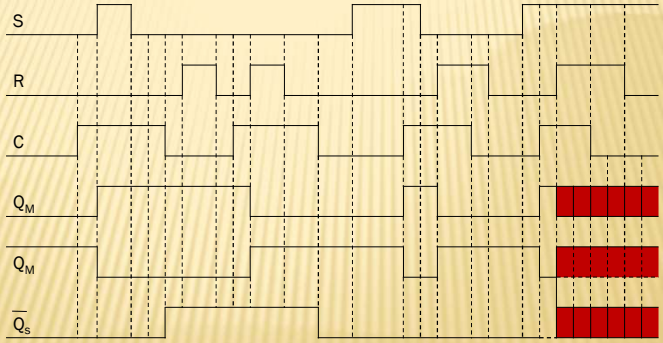
MEMORY DEVICES



153

153

MEMORY DEVICES

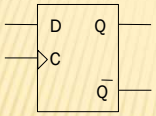


154

154

MEMORY DEVICES

- D Flip-Flop



State Table

q \ D	0	1
0	0	1
1	0	1

Excitation Table

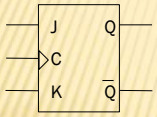
q	Q	D
0	0	0
0	1	1
1	0	0
1	1	1

155

155

MEMORY DEVICES

- JK Flip-Flop



State Table

q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

Excitation Table

q	Q	J	K
0	0	0	–
0	1	1	–
1	0	–	1
1	1	–	0

- Characteristic equation

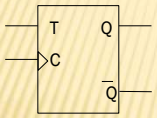
$$Q = J\overline{q} + \overline{K}q$$

156

156

MEMORY DEVICES

- T Flip-Flop



State Table

q \ T	0	1
0	0	1
1	1	0

Excitation Table

q	Q	T
0	0	0
0	1	1
1	0	1
1	1	0

- Characteristic equation

$$Q = T\bar{q} + \bar{T}q$$

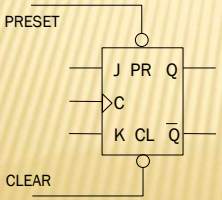
157

157

MEMORY DEVICES

- **Asynchronous inputs**

  Most integrated circuit flip-flops have one or more asynchronous inputs that operate independently of the synchronous inputs and the clock input. The asynchronous inputs can override all other inputs in order to place the flip-flop in one state or the other.



PRESET	CLEAR	Flip-flop Output
0	0	Indeterminate
0	1	Set (Q=1)
1	0	Reset (Q=0)
1	1	Unaffected

158

158

SEQUENTIAL CIRCUITS

- Registers

  Registers are digital circuits commonly used to store binary information. They can be classified into two types:

  - Parallel registers used solely for storing binary information
  - Shift registers used to store information also to process data

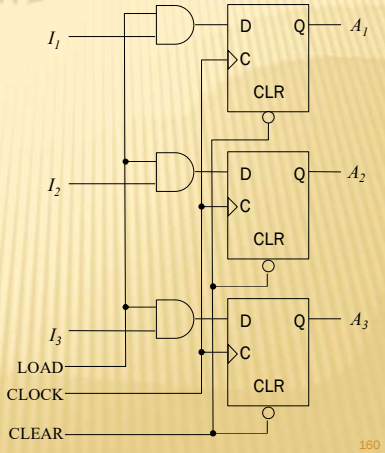
159

159

SEQUENTIAL CIRCUITS

- Parallel Registers

3-bit register, the LOAD input controls the operation of the register.
LOAD=0, the AND gates are disables, D=0, the register cannot change state on any clock pulse.
LOAD=1, the AND gates are Enables, the input information can be transferred into the register on the next trigerring transition of clock
CLEAR=0, the content of register will be 0.



160

160

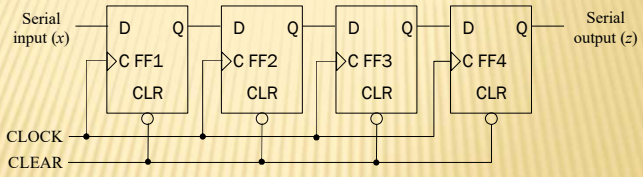
LOAD	CLOCK	CLEAR	I1	I2	I3	A1	A2	A3
1	0	0	1	1	0	?	?	?
1	0	1	1	1	0	?	?	?
0	1	1	1	1	0	?	?	?
1	2	1	1	1	0	?	?	?
0	3	1	1	1	0	?	?	?

161

161

SEQUENTIAL CIRCUITS

- Shift Registers



162

162

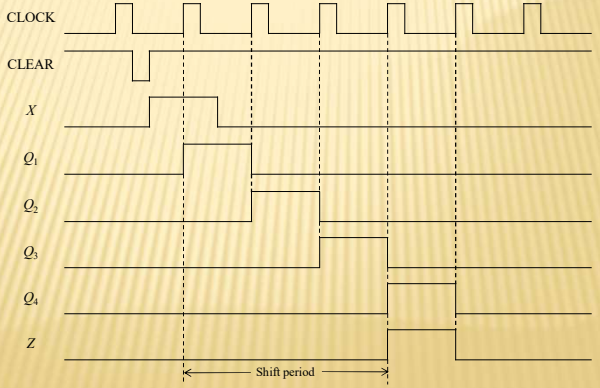
Row	x	CLEAR	CLOCK	Output FF1 FF2 FF3 FF4
1	1	0	0	0000
2	1	1	0	0000
3	1	1	1	1000
4	0	1	2	0100
5	0	1	3	0010
6	0	1	4	0001
7	0	1	5	0000

163

163

SEQUENTIAL CIRCUITS

- Timing diagram for the shift register



164

164

SEQUENTIAL CIRCUITS

- COUNTERS
- A counter is a sequential circuit that follows a prescribed sequence of distinct states under the control of input clock pulses. Counters are classified according to the way in which they are clocked as either *ripple counters (asynchronous counters)* or *synchronous counters*.
- The number of distinct states in the counting sequence is called the *modulus* of the counter. If the modulus is an integral power of 2, the counter is said to have a *natural binary modulus* (Ex. mod-4, mod-8, mod-16…)

165

165

SEQUENTIAL CIRCUITS

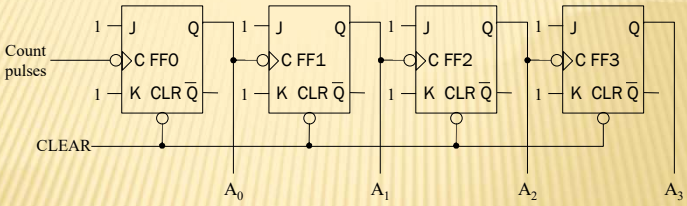
- Ripple counter
- A binary ripple counter consists of a cascade connection of flip-flops, each operating in a complementing mode.
- **Example** A 4-bit binary ripple counter implemented with negative edge-triggered JK flip-flops
- Since the J and K inputs of all the flip-flops are connected to 1, each flip-flop will toggle (complement; change state) on the transition from 1 to 0 at its clock input.
- The timing diagram does not consider the propagation time delays inherent in the operation of a ripple counter, but does explain how the counter works.

166

166

SEQUENTIAL CIRCUITS

- Ripple counter



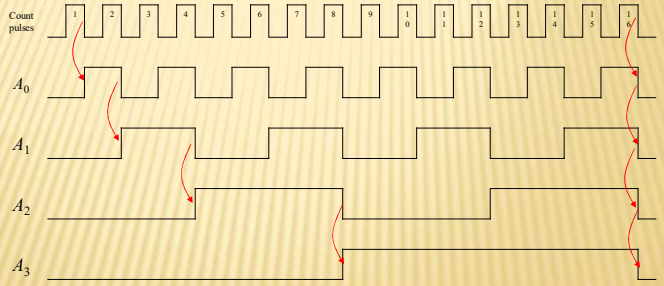
- The count pulses input is not connected directly to the clock input of each flip-flop, the ripple counter is also called an asynchronous counter
- This counter counts upward from zero and is an **up counter**. We have also a counter counting downward from some maximum count to zero, it is a **down counter**

167

167

SEQUENTIAL CIRCUITS

- Ripple counter



168

168

SEQUENTIAL CIRCUITS

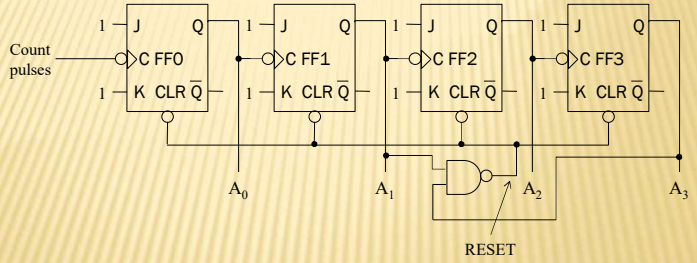
- Ripple Counters with Arbitrary Modulus
- The ripple counter examined is characterized by modulus number (16) that is integral power of 2, that means a natural binary modulus. For various applications, we may required to construct a counter whose modulus is not natural (for example 10)
- We can easily modify a binary ripple counter to produce any modulus by resetting it to 0 at the desired count. This technique is referred to as **premature resetting.**
- Consider the design of a mod-10 (*decimal*) ripple counter

169

169

SEQUENTIAL CIRCUITS

- Ripple Counter with Mod-10

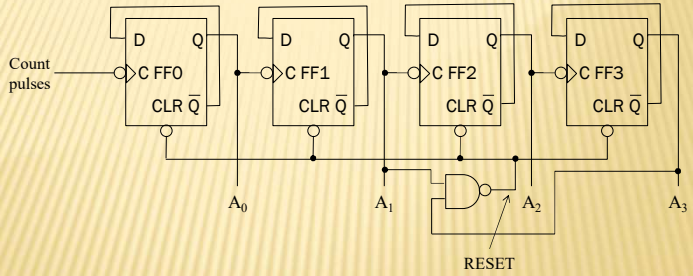


170

170

SEQUENTIAL CIRCUITS

- Ripple Counter with Mod-10



171

171

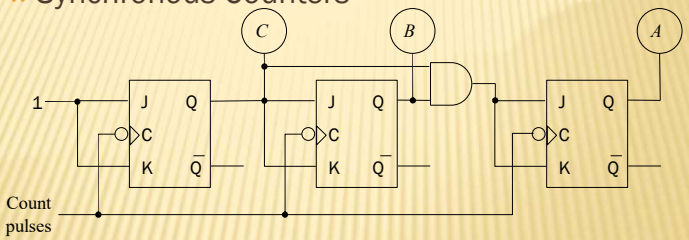
Q3	Q2	Q1	Q0
0	0	0	0
0	0	0	1
...	...	...	...
1	0	0	1
1	0	1	0
...	...	...	...
1	1	1	1

172

172

SEQUENTIAL CIRCUITS

- Synchronous Counters



173

173

DESIGN & ANALYSIS OF SYNCHRONOUS SEQUENTIAL CIRCUITS

- The design and analysis of any system are essentially carried out in opposite directions.
- The design of a sequential circuit is the process of deriving a circuit logic diagram from the specification of the circuit's required behavior. The circuit's behavior or the design specification are often expressed in words.
- The analysis of a sequential circuit begins with a circuit logic diagram and terminates when we obtain a state table (or a state diagram) describing the function of that circuit.

174

174

DESIGN & ANALYSIS OF SYNCHRONOUS SEQUENTIAL CIRCUITS

DESIGN

ANALYSIS

STEP 1. Deriving the state table (and state diagram) for the circuit from the problem statement

STEP 2. State assignment to create the state transition table.

STEP 3. Deriving the circuit excitation table and the output table

STEP 4. Deriving the excitation equations for each flip-flop and the circuit output equations

STEP 5. Drawing the circuit diagram

175

175

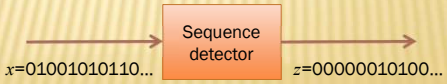
- S0 ->0 (1), S1 ->1 (0)
- S00 -> 00, S01 -> 01, S10 - > 10, S11 -> 11
- State Variables: 0, 1

176

176

EXAMPLES OF DESIGN

- **Example 1**. Design a synchronous circuit that will detect every occurrence of 0101 in a serially transmitted message of any length.
- Since the message is transmitted serially, only one input (x) is required. To detect the occurrence of the sequence 0101, we need a single output (z); let z=1 designate that the sequence has been detected. We allow the detection of overlapping sequences. To illustrate the operation of the required circuit, consider the following input/output sequence:

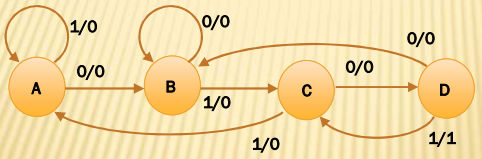


177

177

EXAMPLES OF DESIGN

- STEP1. Mealy state diagram and state table for the 0101 sequence detector.



PS	NS	
	x=0	x=1
A	B,0	A,0
B	B,0	C.0
C	D, 0	A,0
D	B,0	C,1

A waits for the 1st 0
B had the 1st 0, waits for the 1st 1
C had 01, waits for the 2nd 0, 011
D had 010, waits for the last 1

178

178

EXAMPLES OF DESIGN

- STEP 2. The transition table resulting from the chosen state assignment . Two flip-flops are required to implement the circuit.

PS	NS(Q_1Q_2)	
q_1q_2	x=0	x=1
A=00	01,0	00,0
B=01	01,0	10.0
C=10	11, 0	00,0
D=11	01,0	10,1

179

179

EXAMPLES OF DESIGN

- STEP 3. The excitation table using SR flip-flop

Transition Table

PS	NS(Q_1Q_2)	
q_1q_2	x=0	x=1
A=00	01,0	00,0
B=01	01,0	10,0
C=10	11, 0	00,0
D=11	01,0	10,1

Excitation Table

PS	NS(Q_1Q_2)		x=0		x=1	
q_1q_2	x=0	x=1	S_1R_1	S_2R_2	S_1R_1	S_2R_2
00	01	00	0 –	1 0	0 –	0 –
01	01	10	0 –	– 0	1 0	0 1
10	11	00	– 0	1 0	0 1	0 –
11	01	10	0 1	– 0	– 0	0 1

Excitation Table for SR flip-flop

q	Q	S	R
0	0	0	–
0	1	1	0
1	0	0	1
1	1	–	0

180

180

EXAMPLES OF DESIGN

- STEP 4. The excitation equations

q_1q_2 \ x	0	1
00	0	0
01	0	1
11	0	–
10	–	0

$$S_1 = xq_2$$
$$R_1 = x \oplus q_2$$
$$S_2 = \bar{x}$$
$$R_2 = x$$

The output equation

q_1q_2 \ x	0	1
00	0	0
01	0	0
11	0	1
10	0	0

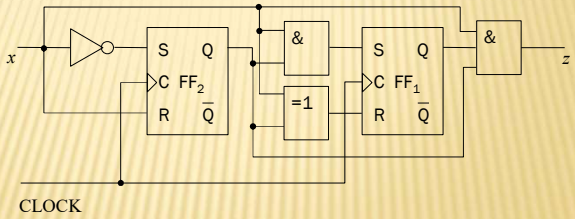
$$z = xq_1q_2$$

181

181

EXAMPLES OF DESIGN

- STEP 5. The circuit logic diagram

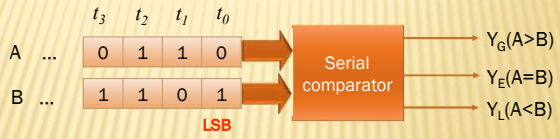


182

182

EXAMPLES OF DESIGN

- **Example 2**. Design a serial comparator for unsigned numbers *A* and *B* with any length. The comparator takes two bits a_i and b_i at a time. Assume the two bits are being fed in from LSB. Implement with *JK* flip-flops



A > B: G (Y_G = 1), A = B: E (Y_E = 1), A < B: L (Y_L = 1)

183

183

EXAMPLES OF DESIGN

- State table with Moore model

PS \ a_ib_i	00	01	11	10	Y_G	Y_E	Y_L
G	G	L	G	G	1	0	0
E	E	L	E	G	0	1	0
L	L	L	L	G	0	0	1

- Transition table E=00, G = 10, L=01

q_1q_2 \ a_ib_i	00	01	11	10	Y_G	Y_E	Y_L
00	00	01	00	10	0	1	0
01	01	01	01	10	0	0	1
11	--	--	--	--	-	-	-
10	10	01	10	10	1	0	0

184

184

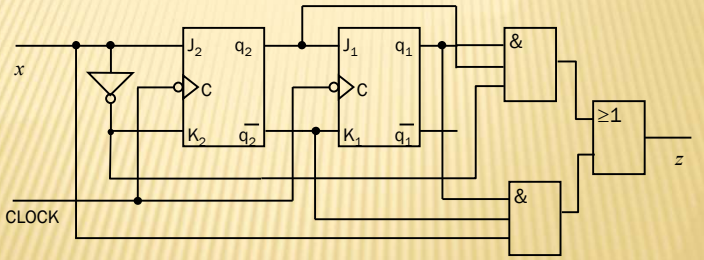
A	0 (MSB)	1	1	0 (LSB)
B	1(MSB)	1	0	0 (LSB)
	L	G	G	E

185

185

EXAMPLES OF ANALYSIS

- **Example 1.** Analyze the following logic diagram



186

186

EXAMPLES OF ANALYSIS

- The excitation equations and the output equations

$$J_1 = q_2,\ K_1 = \overline{q_2},\ J_2 = x,\ K_2 = \overline{x},\quad z = \overline{x}q_1q_2 + xq_1\overline{q}_2$$

- The excitation table

PS	x=0		x=1	
q_1q_2	J_1K_1	J_2K_2	J_1K_1	J_2K_2
00	0 1	0 1	0 1	1 0
01	1 0	0 1	1 0	1 0
11	1 0	0 1	1 0	1 0
10	0 1	0 1	0 1	1 0

187

187

EXAMPLES OF ANALYSIS

PS	x=0		x=1	
q_1q_2	J_1K_1	J_2K_2	J_1K_1	J_2K_2
00	0 1	0 1	0 1	1 0
01	1 0	0 1	1 0	1 0
11	1 0	0 1	1 0	1 0
10	0 1	0 1	0 1	1 0

JK / q	00	01	11	10
0	0	0	1	1
1	1	0	0	1

- The state table

PS	NS(Q_1Q_2)	
q_1q_2	x=0	x=1
00	00,0	01,0
01	10,0	11.0
11	10, 1	11,0
10	00,0	01,1

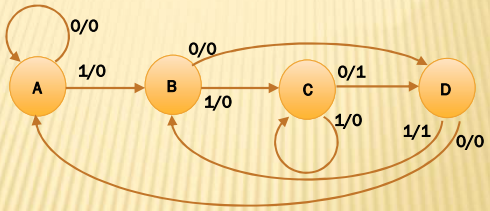
PS	NS	
	x=0	x=1
A	A,0	B,0
B	D,0	C.0
C	D,1	C,0
D	A,0	B,1

188

188

EXAMPLES OF ANALYSIS

- The state diagram

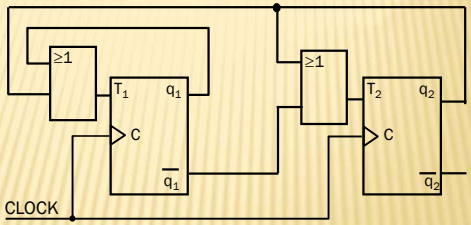


189

189

EXAMPLES OF ANALYSIS

- **Example 2**. Analyze the following logic diagram



190

190

EXAMPLES OF ANALYSIS

- Excitation equations

$$T_1 = q_1 + q_2$$
$$T_2 = \overline{q_1} + q_2$$

- Excitation table & Transition table

q_1q_2	T_1T_2
00	01
01	11
11	11
10	10

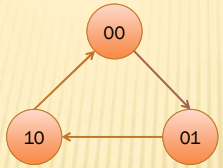
q_1q_2	Q_1Q_2
00	01
01	10
10	00
11	00

191

191

EXAMPLES OF ANALYSIS

- State diagram



192

192

SIMPLIFICATION OF SEQUENTIAL CIRCUIT

- Two states of a sequential system are *equivalent* if, starting in either state, any one input produces the same output and equivalent next state.
- If two states are equivalent, we can remove one of them and have a system with fewer states. Usually, a system with fewer states are less expensive to implement.

193

193

SIMPLIFICATION OF SEQUENTIAL CIRCUIT

- Occasionally, we can tell states are equivalent by just inspecting the state table

PS	NS	
	x=0	x=1
A	C,0	B,0
B	E,0	D,0
C	A,0	D,1
D	A,0	B,1
E	A,0	B,1

PS	NS	
	x=0	x=1
A	C,0	B,0
B	D,0	D,0
C	A,0	D,1
D	A,0	B,1

- For states D and E, the next state is the same (A) for x=0 and is also the same (B) for x=1. The outputs are the same for each state, for both x=0 and x=1. Thus, we can remove one of the states, for example, state E

194

194

A TABULAR METHOD FOR STATE REDUCTION

- A chart with one square for each possible pairing of states. The chart has one row for each state except the first and one column for each state except the last. We enter in each square
  - an X if those states cannot be equivalent because the outputs are different,
  - a √ if the states are equivalent (because they have the same output and go to the same state or go to each other for each input)
  - the conditions that must be met for those two states to be equivalent that is, which states must be equivalent to make these equivalent)

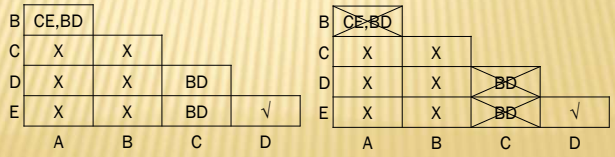
195

195

A TABULAR METHOD FOR STATE REDUCTION

Example 1. In order for states A and B to be equivalent. they must have the same output for both x=0 and x=1 and must go to equivalent state. Thus, C must be equivalent to E and B must be equivalent to D.
Since B cannot be equivalent to B and C cannot be equivalent to E (there are already x in BD and CE squares), so we cross out those squares, leaving only one check.

PS	NS	
	x=0	x=1
A	C,0	B,0
B	E,0	D,0
C	A,0	D,1
D	A,0	B,1
E	A,0	B,1



196

196

A TABULAR METHOD FOR STATE REDUCTION

- Example 2.

PS	NS		Z
	x=0	x=1	
A	B	D	1
B	D	F	1
C	D	A	0
D	D	E	0
E	B	C	1
F	C	D	0

B	BD,DF				
C	X	X			
D	X	X	AE		
E	CD	BD,CF	X	X	
F	X	X	CD,AD	CD,DE	X
	A	B	C	D	E

PS	NS		Z
	x=0	x=1	
A	B	C	1
B	C	F	1
C	C	A	0
F	C	C	0

B	~~BD,DF~~				
C	X	X			
D	X	X	AE		
E	CD	~~BD,CF~~	X	X	
F	X	X	~~CD,AD~~	~~CD,DE~~	X
	A	B	C	D	E

197

197

A TABULAR METHOD FOR STATE REDUCTION

- Example 3.

PS	NS		Z
	x=0	x=1	
A	B	D	1
B	D	F	1
C	D	A	1
D	D	E	0
E	B	C	0
F	C	D	0

B	BDF				
C	ABD	AF			
D	X	X	X		
E	X	X	X	BD,CE	
F	X	X	X	CDE	BCD
	A	B	C	D	E

- No states can be combined and the state table cannot be reduced

198

198

A TABULAR METHOD FOR STATE REDUCTION

- Example 4.

PS	NS		Z	
	x=0	x=1	x=0	x=1
A	B	D	0	0
B	E	D	1	0
C	B	C	0	0
D	F	A	0	0
E	A	B	1	1
F	E	C	1	0

B	X				
C	CD	X			
D	BF	X	BF,AC		
E	X	X	X	X	
F	X	CD	X	X	X
	A	B	C	D	E

- The state table is reduced with only 3 states A (A-C-D). B (B-F) and E

PS	NS		Z	
	x=0	x=1	x=0	x=1
A	B	A	0	0
B	E	A	1	0
E	A	B	1	1

199

199

PARTITIONS

- A partition on the states of a system is a grouping of the states of that system into one or more blocks. Each state must be in one and only one block. For a system with 4 states, *A*, *B*, *C* and *D*, the complete list of partitions is

P_0=(A)(B)(C)(D)	P_8=(AC)(BD)
P_1=(AB)(C)(D)	P_9=(AD)(BC)
P_2=(AC)(B)(D)	P_{10}=(ABC)(D)
P_3=(AD)(B)(C)	P_{11}=(ABD)(C)
P_4=(A)(BC)(D)	P_{12}=(ACD)(B)
P_5=(A)(BD)(C)	P_{13}=(A)(BCD)
P_6=(A)(B)(CD)	P_N=(ABCD)
P_7=(AB)(CD)	

200

200

PARTITIONS

- Three categories of partition will be of interest

1) Any partition with two blocks can be used to assign one of the state variables

q	q_1	q_2
A	0	0
B	0	1
C	1	0
D	1	1
	P_7	P_8

q	q_1	q_2
A	0	0
B	0	1
C	1	1
D	1	0
	P_7	P_9

q	q_1	q_2
A	0	0
B	1	1
C	0	1
D	1	0
	P_8	P_9

201

201

PARTITIONS

2) A useful class of partitions are those for which all of the states in each block have the same output for each of the inputs

q	q_1	q_2
A	0	0
B	0	1
C	1	0
D	1	1
	P_7	P_8

q	q_1	q_2
A	0	0
B	0	1
C	1	1
D	1	0
	P_7	P_9

q	q_1	q_2
A	0	0
B	1	1
C	0	1
D	1	0
	P_8	P_9

202

202

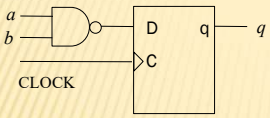
HARDWARE DESIGN LANGUAGES

- Most design of significant digital systems is done with computer aided tools. They allow the user to specify either the behavior of the system or the structure of the system (or a mixture of the two) using a notation similar to a programming language.
- The two most widely used systems are Verilog and VHDL. They have many similarities, but differ in detail.
- VHDL stands for VHSIC Hardware Description Language. VHSIC is itself an abbreviation for Very High Speed Integrated Circuits, an initiative funded by the United States Department of Defense in the 1980s that led to the creation of VHDL.

203

203

HARDWARE DESIGN LANGUAGES




```
1 ----------------------------------
2 ENTITY example IS
3 PORT ( a, b, clk: IN BIT;
4 q: OUT BIT);
5 END example;
6 ----------------------------------
7 ARCHITECTURE example OF example IS
8 SIGNAL temp : BIT;
9 BEGIN
10 temp <= a NAND b;
11 PROCESS (clk)
12 BEGIN
13 IF (clk'EVENT AND clk='1') THEN q<=temp;
14 END IF;
15 END PROCESS;
16 END example;
17 ----------------------------------
```

204

204