EE 102: INTRODUCTION TO COMPUTER ENGINEERING

Lecture 3: Combination Logic

4/5/2010

Avinash Kodi, kodi@ohio.edu

Administration

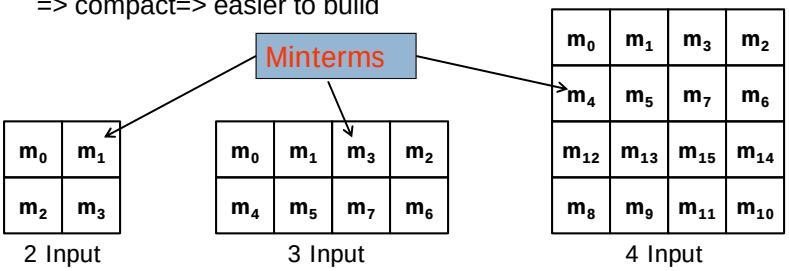
2

- Hw 1 due Wednesday 4/7
- Quiz 1 on Wednesday 4/7 (includes today's lectures)
- Hw 2 will be posted later today
- Lab 1 next week – more information on course webpage

Karnaugh Maps - Definitions

3

- Specific layouts of truth tables to simplify the logic expressions in Canonical forms to their minimum form
  - Canonical forms: SoP (minterms) or PoS (maxterms)
    - Each min/max term contains all variables => not compact
  - The minimum form: contains the fewest possible terms (operations) and variables (literals)
  - Can be implemented with fewer logic gates than canonical forms => compact=> easier to build

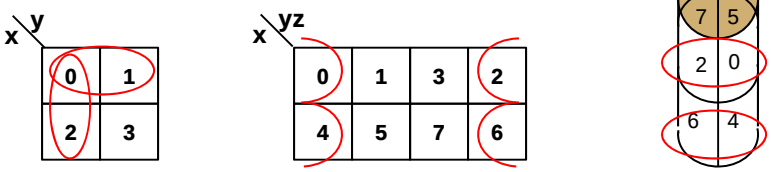


3 Steps in K-maps: (1) Mapping, (2) Simplification, (3) Reading

Adjacency

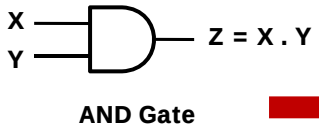
4

- Defined by a single variable change in min/max terms
- In K-Maps, each cell is adjacent to the cells that are immediately next to it on any of its 4 sides
  - Diagonal neighbors are not adjacent
  - Perimeter cells in any column or row are adjacent (wrap-around adjacency)
- Useful for reduction: $m_i + m_{i+1}$ eliminates one variable
  - AB + A'B = B
  - (X+Y)(X+Y') = X



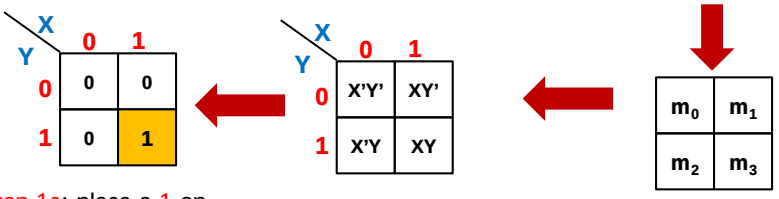
Step 1: Mapping a SoP (2 Variables)

5



Truth Table

X	Y	Z = X . Y	Index	Binary
0	0	0	m_0	00
0	1	0	m_1	01
1	0	0	m_2	10
1	1	1	m_3	11



Step 1c: place a 1 on the K-map in the cell having the same index

Step 1b: determine the indices of the minterms (or use the variable key on the upper left)

2 Input

Step 1a: ensure that each cell of K-map differs by one-bit

Step 2: SoP Simplification

6

- Objective: to obtain an expression with the least number of terms and literals

- 3 steps in the process of SoP simplification:
  - 2a: grouping the 1s in blocks of 2^n adjacent elements:
    - groups of 1,2,4,8,etc.
    - prefer larger groups
    - groups themselves do not have to adjacent
    - each 1 to be included at least in one of the groups
    - groups overlaps (or sharing 1s) allowed
  - 2b: determine the group product using adjacency
    - for each group of 2^n, n variables are eliminated
  - 2c: summing the resulting product terms.

Step 3: Reading a K-Map

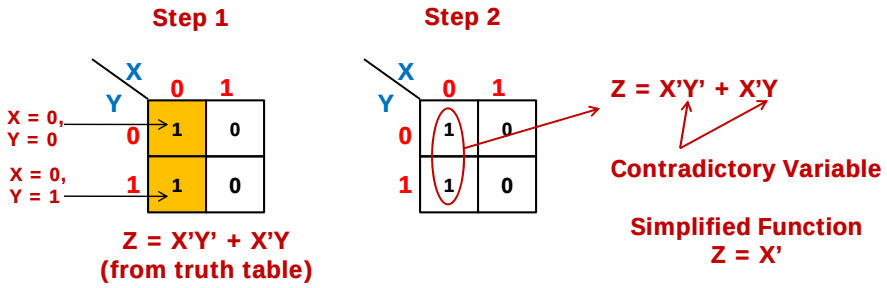
7

- Example

3a: remember to maximize groups

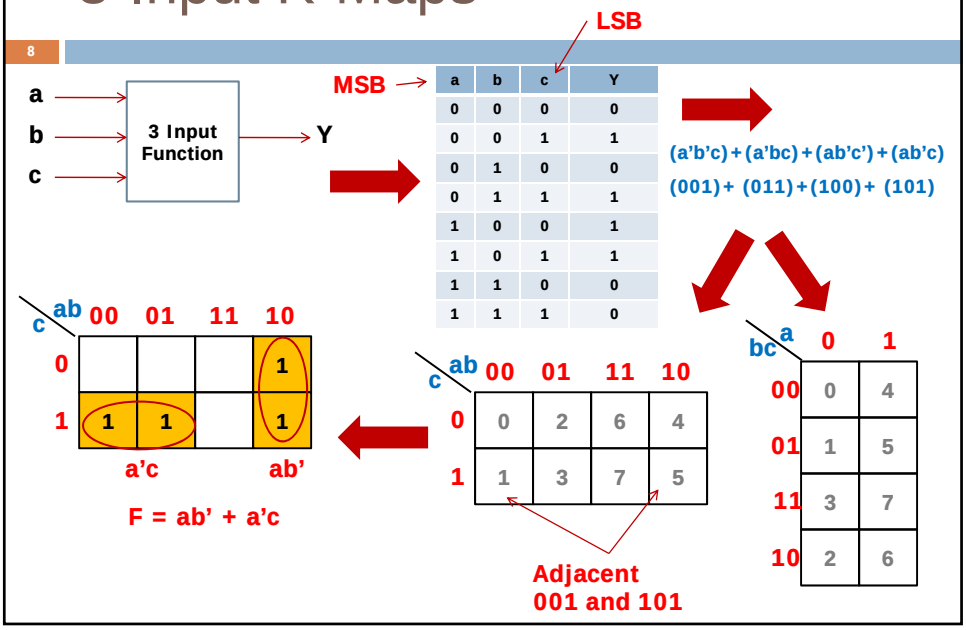
3b: decide carefully which input variable eliminated in each group

X	Y	Z	Index	Binary
0	0	1	m_0	00
0	1	1	m_1	01
1	0	0	m_2	10
1	1	0	m_3	11



3-Input K-Maps

8

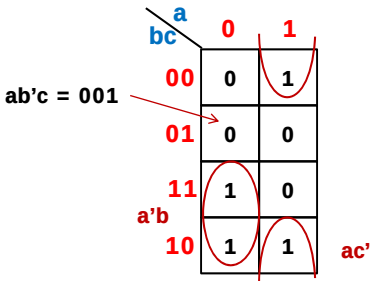


a	b	c	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Example

9

a	b	c	Y
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

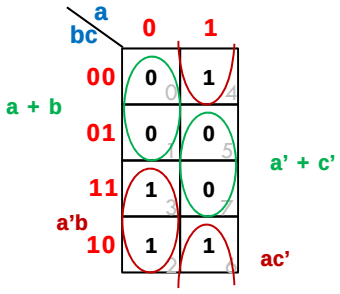


Y (a,b,c) = a'b + ac'

K-Maps: PoS Form

10

- In case of PoS => group and reduce zeros (0)
  - Corresponding to maxterms
  - Should give the same function as SoP



Y (a,b,c) = a'b + ac' **SoP**

Y (A, B, C) = (a + b)(a' + c') **PoS**

Y = aa' + ac' + a'b + bc'
= ac' + a'b **(Consensus)**

4-Variable K-Maps

11

- Each term composed of 4 variables
- Similar to and expansion of 3-variable tables

MSB

LSB

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

LSB

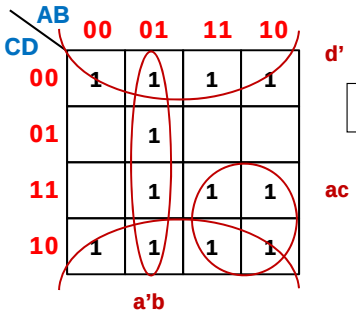
MSB

YZ / WX	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

Example

12

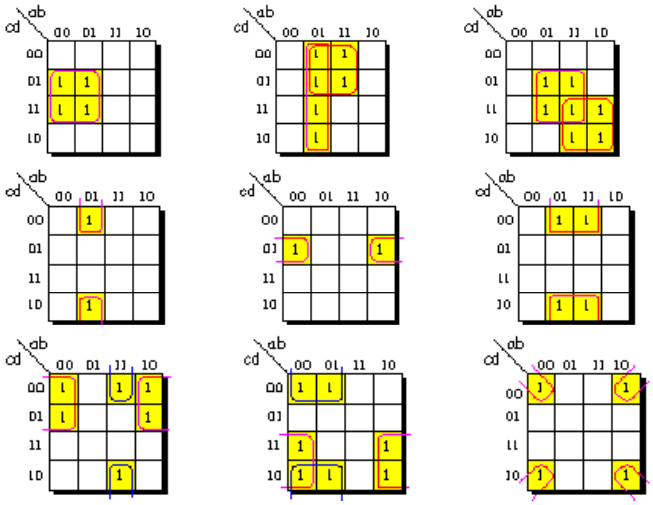
$F(a,b,c,d) = \Sigma(0,2,4,5,6,7,8,10,11,12,14,15)$



F(a,b,c,d) = a'b + ac + d'

Examples of Combining

13

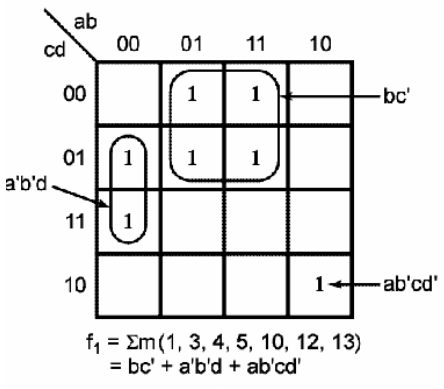


More Examples

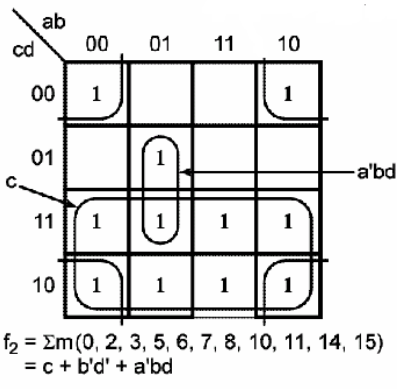
14

F(a,b,c,d) =S(1,3,4,5,19,12,13)

F(a,b,c,d) =S(0,2,3,5,6,7,8,10,11,14,15)



$f_1 = \Sigma m(1, 3, 4, 5, 10, 12, 13)$
$= bc' + a'b'd + ab'cd'$



$f_2 = \Sigma m(0, 2, 3, 5, 6, 7, 8, 10, 11, 14, 15)$
$= c + b'd' + a'bd$

Essential and Non-Essential Prime Implicants

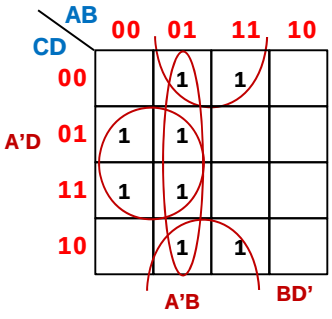
15

- A product term is an implicant of a function, if the function has the value of 1 for all minterms
- If the removal of any literal from an implicant P results in a product term that is not an implicant of the function, then P is a prime implicant
- If the minterm of a function is included in only one prime implicant, that prime implicant is essential

Essential and Non-Essential Prime Implicants

16

$F(a,b,c,d) = \Sigma(1,3,4,5,6,7,12,14)$



Prime Implicants = A'D, A'B and BD'

Essential Prime Implicants = A'D + BD'

Non-essential Prime Implicant = A'B

Essential and Non-Essential Prime Implicants

17

$F(a,b,c,d) = \Sigma(0, 5, 10, 11, 12, 13, 15)$

Combinational Circuits

18

- System of interconnected logic gates whose outputs at any time are determined by combining the values of the applied inputs only
  - Performs am operation specified by a set of Boolean expressions
  - Consists of input variables, output variables, logic gates and interconnections

  - Examples: Encoders, Decoders, Multiplexers, Demultiplexers, Adders, Comparators

Design Procedure

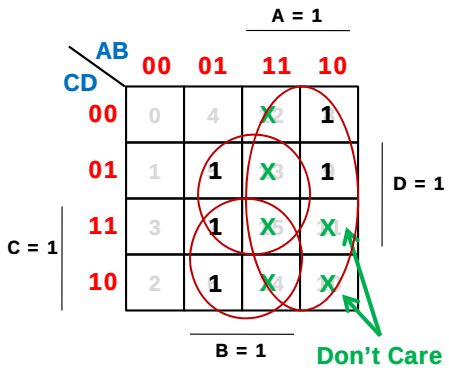
19

- The design starts from the specification of the problem and culminates in a logic diagram or set of Boolean equations. The procedure involves the following steps:
  1. Using the specifications of the circuit, determine the input and output variables.
  2. Derive the truth table that defines the required relationship between inputs and outputs.
  3. Break the circuit into small single-output blocks.
  4. Obtain the simplified Boolean functions for each output in terms of input variables
  5. Draw the logic diagram.
  6. Verify the design (logic simulation).

Example 1: Code Converter (1/2)

20

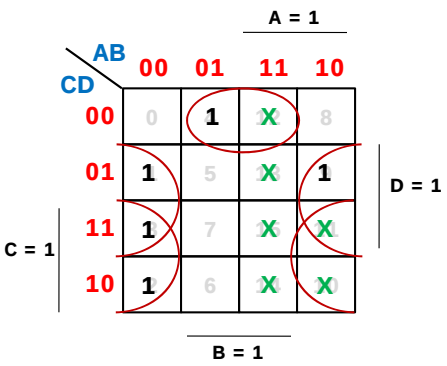
	Input BCD	Output Excess-3
Decimal	A B C D	W X Y Z
0	0 0 0 0	0 0 1 1
1	0 0 0 1	0 1 0 0
2	0 0 1 0	0 1 0 1
3	0 0 1 1	0 1 1 0
4	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 0 0
6	0 1 1 0	1 0 0 1
7	0 1 1 1	1 0 1 0
8	1 0 0 0	1 0 1 1
9	1 0 0 1	1 1 0 0



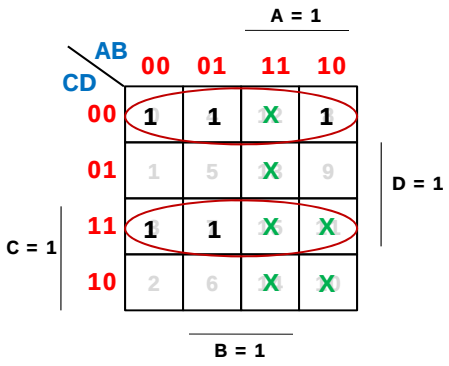
$W = A + BC + BD$

Example 1: Code Converter (2/2)

21



X = BC'D' + B'D + B'C



Y = C'D' + CD

Z = D'

Example Code Converter: Circuit Design

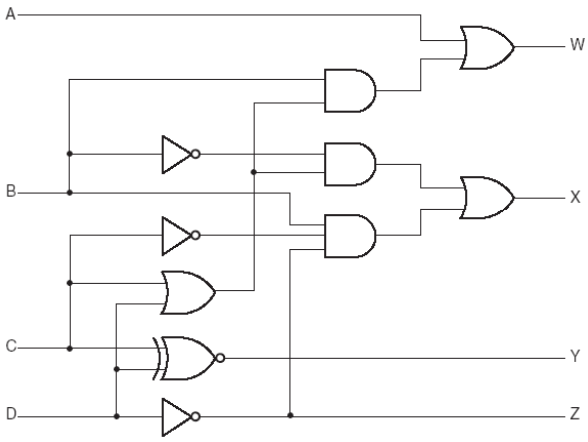
22

W = A + BC + BD = A + B(C + D)

X = BC'D' + B'D + B'C = BC'D' + B'(C + D)

Y = C'D' + CD

Z = D'



Example 2

23

Truth Table for BCD–to–Seven-Segment Decoder

BCD Input				Seven-Segment Decoder						
A	B	C	D	a	b	c	d	e	f	g
0	0	0	0	1	1	1	1	1	1	0
0	0	0	1	0	1	1	0	0	0	0
0	0	1	0	1	1	0	1	1	0	1
0	0	1	1	1	1	1	1	0	0	1
0	1	0	0	0	1	1	0	0	1	1
0	1	0	1	1	0	1	1	0	1	1
0	1	1	0	1	0	1	1	1	1	1
0	1	1	1	1	1	1	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1
All other inputs				0	0	0	0	0	0	0

(a) Segment designation

(b) Numeric designation for display

$$a = \overline{A}C + \overline{A}BD + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$
$$b = \overline{A}\,\overline{B} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}CD + A\overline{B}\,\overline{C}$$
$$c = \overline{A}B + \overline{A}D + \overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$
$$d = \overline{A}C\overline{D} + \overline{A}B\overline{C} + \overline{B}\,\overline{C}D + A\overline{B}\,\overline{C} + \overline{A}B\overline{C}D$$
$$e = \overline{A}C\overline{D} + \overline{B}\,\overline{C}D$$
$$f = \overline{A}B\overline{C} + \overline{A}\,\overline{C}\,\overline{D} + \overline{A}B\overline{D} + A\overline{B}\,\overline{C}$$
$$g = \overline{A}C\overline{D} + \overline{A}B\overline{C} + \overline{A}BC + A\overline{B}\,\overline{C}$$

7 output functions

Positive and Negative Representations

Code	Signed Magnitude	One's Complement	Two's Complement
000	+0	+0	+0
001	+1	+1	+1
010	+2	+2	+2
011	+3	+3	+3
100	-0	-3	-4
101	-1	-2	-3
110	-2	-1	-2
111	-3	-0	-1

- Issues: balance, number of zeros, ease of operation
- Which one is best? Why?

1's Complement

25

- 1's complement: Convert all 0 to 1 and 1 to 0 (invert all bits)

Number	1's Complement
+7	00111
-7	11000

$(7)_{10} = 1*2^2+1*2^1+1*2^0 = 0\ 0\ 1\ 1\ 1$

1's Complement 1 1 0 0 0

Not a sign bit

1's Complement

26

- Representing negative numbers for n = 32 bits
  - $(b31\times -2^{31}) + (b30 \times 2^{30}) + (b29 \times 2^{29}) + \ldots + (b1 \times 2^1) + (b0 \times 2^0) + 1$
- Example: if n = 4 bits, decimal value of 1000
  - $(1\times -2^3) + (0 \times 2^2) + (0 \times 2^1) + (0 \times 2^0) + 1 = -7$
  - Easier method is to convert into positive number and then determine the decimal equivalent

2's Complement

27

- Invert all the bits and ADD 1

Original Number	1's Complement	2's Complement
+7	00000111	00000111
-7	11111000	11111001

1's 1 1 1 1 1 0 0 0

\+ 1

2's 1 1 1 1 1 0 0 1

Other method

Start at the right and complement all bits to the left of the first 1.

0000011 1 → (2's) 1111100 1

2's Complement

- Representing positive and negative numbers for n = 32 bits
  - $(b31 \times -2^{31}) + (b30 \times 2^{30}) + (b29 \times 2^{29}) + \ldots + (b1 \times 2^{1}) + (b0 \times 2^{0})$
- Example: if n = 4 bits, decimal value of 1000
  - $(1 \times -2^{3}) + (0 \times 2^{2}) + (0 \times 2^{1}) + (0 \times 2^{0}) = -8$

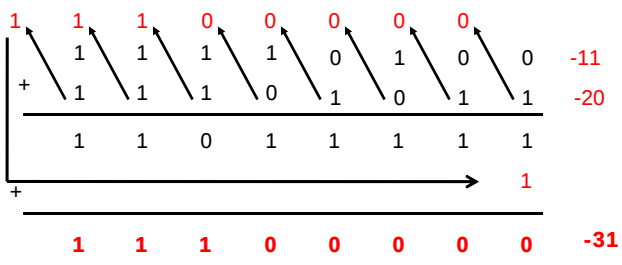
Number Representation for 4 Bits

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

1's Complement Operations

30

- Example: if n = 8, add -11 and -20
  - +11 = 00001011 and +20 = 00010100
  - Bit-by-bit complement
  - -11 = 11110100 and -20 = 11101011

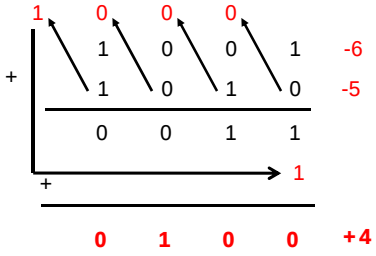


Must add the carry-out 1 to the result to get the correct answer

More 1's Complement Operations

31

- Addition of two negative numbers, n= 4 add -5 and -6

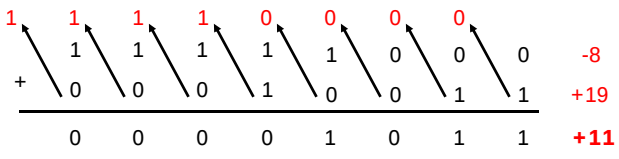


Wrong answer because of carry addition (OVERFLOW)

2's Complement Operations

32

- Example: Add -8 and +19 in 2's Complement for n = 8 bits
  - +8 = 00001000, 2's Complement = 11111000
  - +19 = 00010011

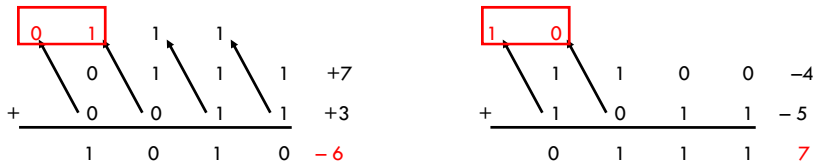


Discard the carry-out 1 in 2's complement

2's Complement subtraction can be handled by a summation i.e. A – B = A + B' + 1

Overflow

- Overflow: the result is too large to represent in 4 bits
- Overflow occurs when
  - adding two positives yields a negative
  - or, adding two negatives gives a positive
  - or, subtract a negative from a positive gives a negative
  - or, subtract a positive from a negative gives a positive
- On your own: Prove you can detect overflow by:

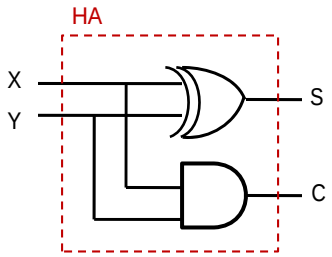


Adders

- Half-Adder

Inputs Outputs

X	Y	C	S
0	0	0	0
1	0	0	1
0	1	0	1
1	1	1	0



C = XY
S = XY' + X'Y
= X ⊕ Y

Full Adders

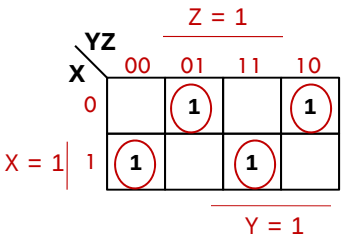
35

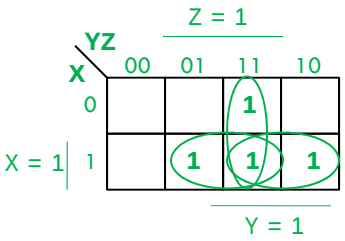
Inputs			Outputs	
X	Y	Z (Carry-In)	C (Carry-Out)	S (Sum)
0	0	0	**0**	**0**
0	0	1	**0**	**1**
0	1	0	**0**	**1**
0	1	1	**1**	**0**
1	0	0	**0**	**1**
1	0	1	**1**	**0**
1	1	0	**1**	**0**
1	1	1	**1**	**1**

Carry-in bit (Z) takes into account any carry information coming from earlier stages of a multi-bit addition

Full Adders

36





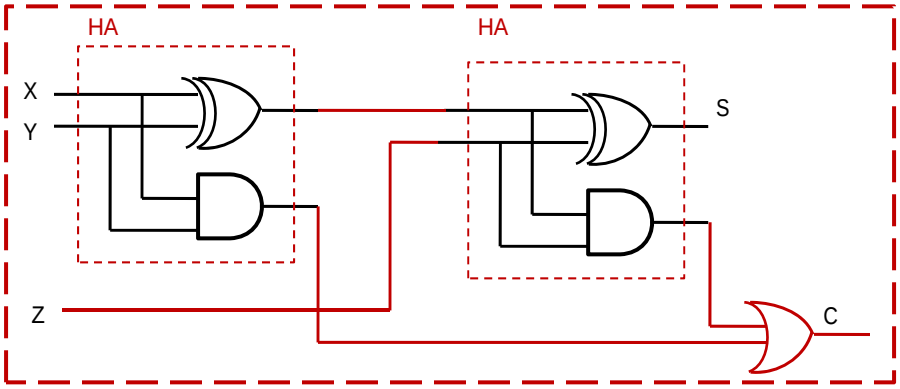
$S = XYZ + X'YZ' + XY'Z' + X'Y'Z$
$= X \oplus Y \oplus Z$

$C = XY + YZ + XZ$
$= XY + Z (X + Y)$
(equivalent)
$= XY + Z(XY' + X'Y)$
$= XY + Z (X \oplus Y)$

Full Adders

37

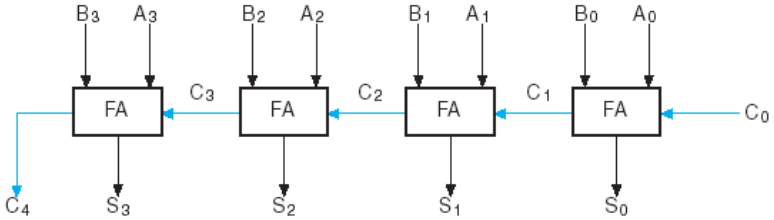
$S = X \oplus Y \oplus Z$ $C = XY + Z(X \oplus Y)$

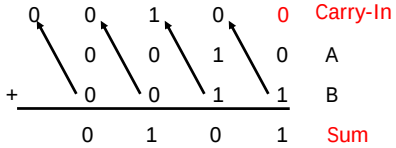


Using the common XOR terms, a single full adder can be realized using two half adders and an additional OR gate. Outputs are the sum (S) and carry-out (C)

Multi-Bit Full Adder (Ripple Carry Adder)

38

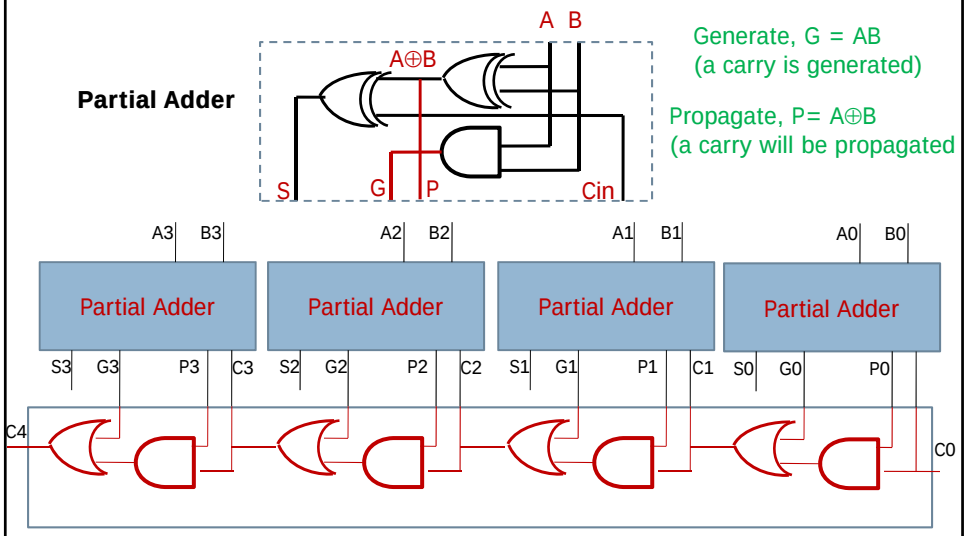




- Ripple adder produces the arithmetic sum of two binary numbers using no memory elements:
  - Slow: note that one must wait until all carries propagate to next stage and C_{out} emerges

Issues with Ripple Adder

39



Carry Look-Ahead (CLA) Adder

40

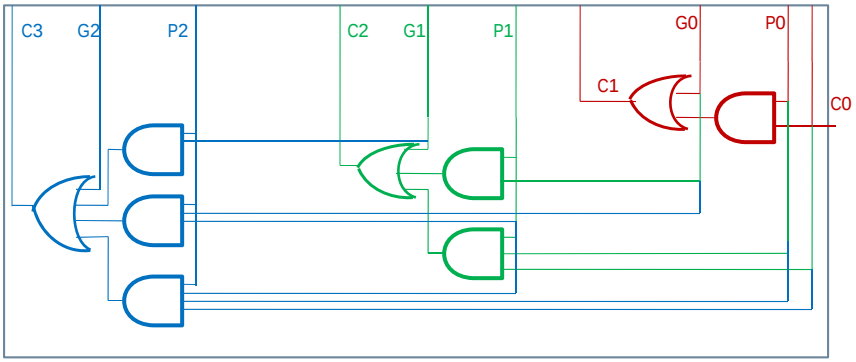
- $c_{i+1} = c_i (b_i \oplus a_i) + (a_i . b_i)$
- $c_1 = (a_0.b_0) + (a_0 \oplus b_0).c_0$
- $c_2 = (a_1 . b_1) + (a_1 \oplus b_1) . c_1$
  $= (a_1 . b_1) + (a_1 \oplus b_1) ((a_0 . b_0) + (a_0 \oplus b_0) . c_0)$
- Repeated use of $(a_i . b_i)$ and $(a_i \oplus b_i)$ in the formula
- Generate (g_i) and Propagate (p_i)
- $g_i = a_i . b_i$ [a carry is always *generated*]
- $p_i = a_i + b_i$ [if $(i-1)^{th}$ bit has a carry, it will be *propagated* to $(i+1)^{th}$ bit]
- $c_{i+1} = g_i + p_i . c_i$

The carry terms are faster because they are evaluated by the knowledge of the p_i and g_i terms from preceding stages:
More expensive to build

Carry Look Ahead

41

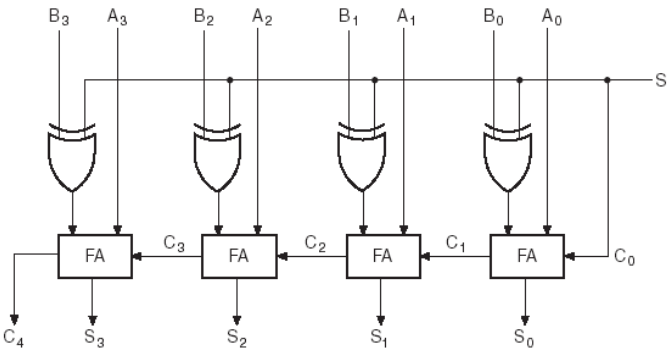
- c1 = g0 + p0.c0
- c2 = g1 + p1.c1 = g1 + p1.(g0 + p0.c0) = g1 + p1.g0 + p1.p0.c0
- c3 = g2 + p2.c2 = g2 + p2.(g1 + p1.g0 + p1.p0.c0)
  = g2 + p2.g1 + p2.p1.g0 + p2.p1.p0.c0



Binary Adder-Subtractors

42

- Design a 4-bit binary adder and subtractor using a Full Adders depending on the select logic,
  - S = 0 implies addition and S = 1 implies subtraction
  - Hint: A – B = A + B' + 1



S=0 Adder
S=1 Subractor

For S=0 $B_i \oplus 0 = B_i$
Cin=0 => A+B

For S=1 $B_i \oplus 1 = B_i'$
Cin=1 =>A-B=A+B'+1
2's complement

Multiplexers

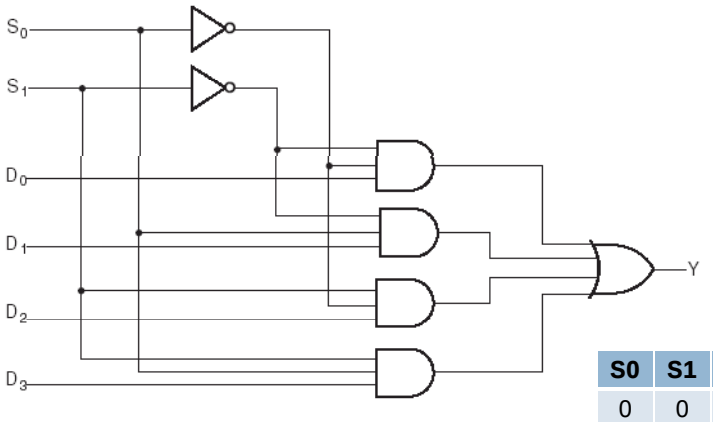
43

- A combinational circuit that selects binary information from one of many input lines and directs the information to a single output line
  - Select input variables to decide the output
  - Example: 4-to-1 Mux (Multiplexer)
    - 4 inputs (D0, D1, D2, D3)
    - 1 output Y
    - Mux Selector Logic (S0, S1)

S0	S1	Y
0	0	D0
1	0	D1
0	1	D2
1	1	D3

4-to-1 Multiplexer

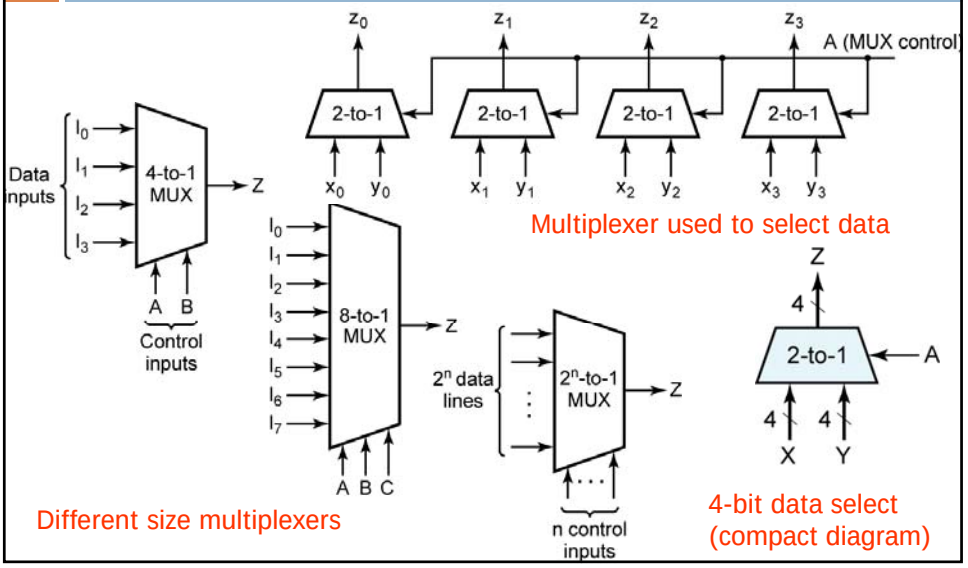
44



S0	S1	Y
0	0	D0
1	0	D1
0	1	D2
1	1	D3

Multiplexer Example - 1

45



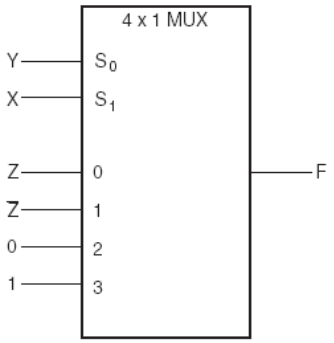
Multiplexer Example - 3

46

- Multiplexers maybe used to manipulate/design other functions
  - Especially useful to re-use/expand large blocks

Select Conditions

X	Y	Z	F	
0	0	0	0	F = Z
0	0	1	1	
0	1	0	1	$F = \overline{Z}$
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	



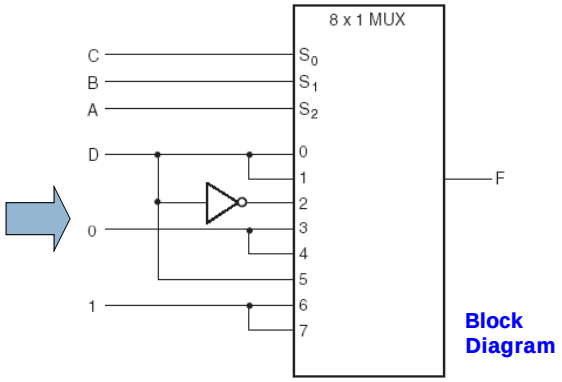
Block Diagram

Multiplexer Example - 4

47

$$F(A,B,C,D) = \Sigma m(1,3,4,11,12,13,14,15)$$

A	B	C	D	F	
0	0	0	0	0	F = D
0	0	0	1	1	
0	0	1	0	0	F = D
0	0	1	1	1	
0	1	0	0	1	F = $\overline{D}$
0	1	0	1	0	
0	1	1	0	0	F = 0
0	1	1	1	0	
1	0	0	0	0	F = 0
1	0	0	1	0	
1	0	1	0	0	F = D
1	0	1	1	1	
1	1	0	0	1	F = 1
1	1	0	1	1	
1	1	1	0	1	F = 1
1	1	1	1	1	

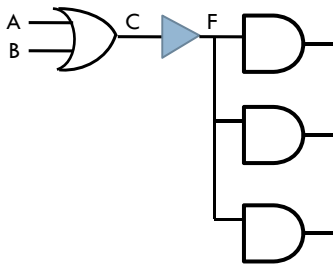


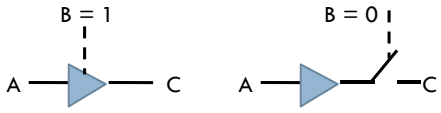
Block Diagram

Three-State Buffers

48

- A gate output can be connected to limited number of input devices
- A buffer can increase the driving capability of a gate output





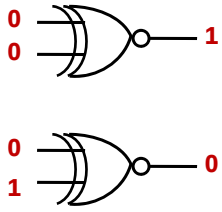
B	A	C
0	0	Z
0	1	Z
1	0	0
1	1	1

Comparators

49

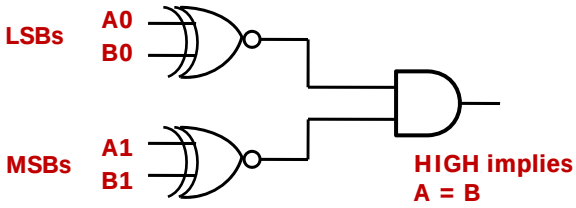
- Compares the magnitude of two quantities to determine the relationship between them
- In its simplest form, a comparator circuit determines whether two numbers are equal

Basic comparator operations



Example

Logic diagram for comparison of two 2-bit binary numbers



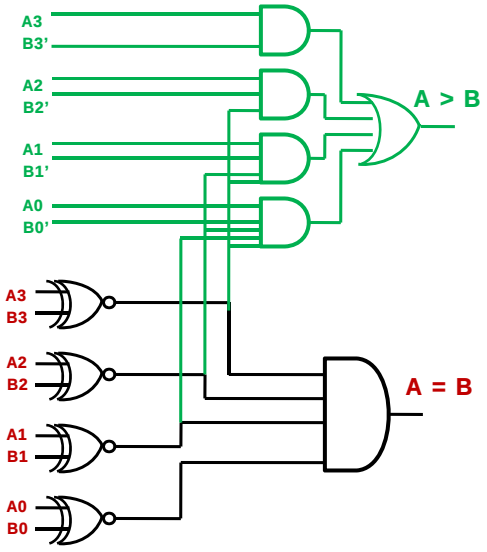
Design Problem: Comparators

50

Compare two four bit numbers and indicate when A=B and when A>B.

To determine an inequality of numbers A and B, first examine the highest-order bit in each number. The following conditions are possible:

- If A3=1 and B3=0 number A is greater than B
- If A3=0 and B3=1 number A is less than number B
- If A3=B3 then examine the next lower bit position for an inequality



Decoders

51

- Combinational circuits that converts binary information from n coded inputs to a maximum of 2n unique outputs

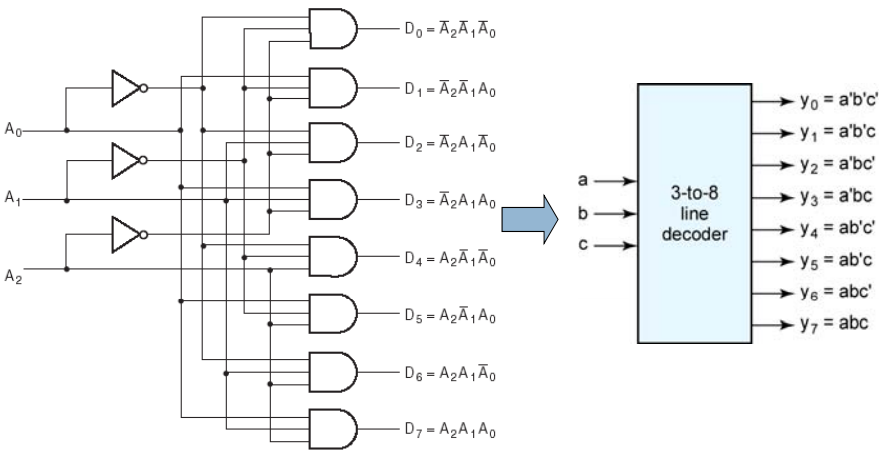
Inputs | Outputs

A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

$$D_0 = \overline{A_2}\,\overline{A_1}\,\overline{A_0}$$
$$D_1 = \overline{A_2}\,\overline{A_1}A_0$$
$$D_2 = \overline{A_2}A_1\overline{A_0}$$
$$D_3 = \overline{A_2}A_1A_0$$
$$D_4 = A_2\overline{A_1}\,\overline{A_0}$$
$$D_5 = A_2\overline{A_1}A_0$$
$$D_6 = A_2A_1\overline{A_0}$$
$$D_7 = A_2A_1A_0$$

Decoder (3-to-8)

52

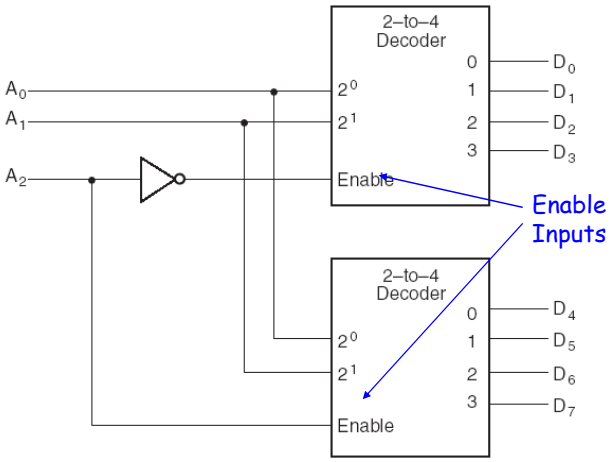


3-to-8-line decoder

Decoder Expansions

53

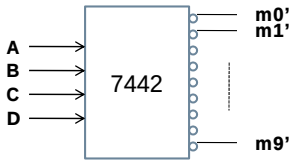
- 3-to-8-line decoder from 2-to-4-line decoders with ENABLE

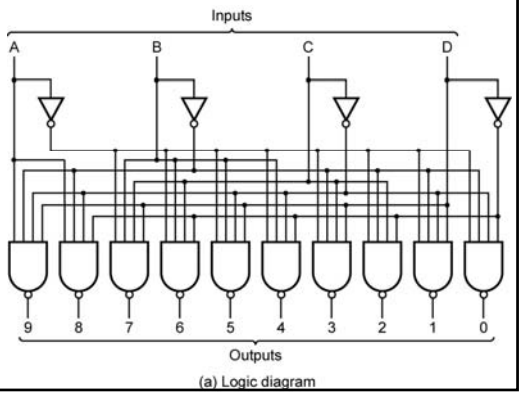


4-to-10 BCD to Decimal Decoder

54

Input BCD	Output Decimal
A B C D	0 1 2 3 4 5 6 7 8 9
0 0 0 0	0 1 1 1 1 1 1 1 1 1
0 0 0 1	1 0 1 1 1 1 1 1 1 1
0 0 1 0	1 1 0 1 1 1 1 1 1 1
0 0 1 1	1 1 1 0 1 1 1 1 1 1
0 1 0 0	1 1 1 1 0 1 1 1 1 1
0 1 0 1	1 1 1 1 1 0 1 1 1 1
0 1 1 0	1 1 1 1 1 1 0 1 1 1
0 1 1 1	1 1 1 1 1 1 1 0 1 1
1 0 0 0	1 1 1 1 1 1 1 1 0 1
1 0 0 1	1 1 1 1 1 1 1 1 1 0
1 0 1 0	1 1 1 1 1 1 1 1 1 1



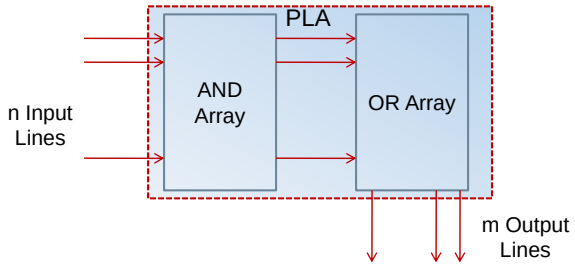


(a) Logic diagram

Programmable Logic Devices

55

- PLD can be programmed to provide a variety of different logic functions
- A PLA has *n* inputs and *m* outputs, can realize *m* functions of *n* variables using a AND array and OR array



Designing PLD

56

$F_0 = A'B' + AC'$
$F_1 = AC' + B$
$F_2 = A'B' + BC'$
$F_3 = B + AC$

Product Term	Inputs A B C	Outputs F_0 F_1 F_2 F_3
A'B'	0 0 --	1 0 1 0
AC'	1 -- 0	1 1 0 0
B	-- 1 --	0 1 0 1
BC'	-- 1 0	0 0 1 0
AC	1 -- 1	0 0 0 1