Heaven's Light is Our Guide

Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology, Bangladesh

Course Code: CSE 2203
Course Title: Digital Techniques

Date : 10.10.2017
Session : VI
Topic : Combinational Logic Circuits

Faculty : Dr. Boshir Ahmed
Professor
Department of CSE, RUET
E mail: boshir78@gmail.com

Circuit Simplification Methods

- Boolean algebra: greatly depends on inspiration and experience.
- Karnaugh map: systematic, step-by-step approach.

Algebraic Simplification

- Use the Boolean algebra theorems introduced in Lecture 4 to help simplify the expression for a logic circuit.
- Based on experience, often becomes a trial-and-error process.
- No easy way to tell whether a simplified expression is in its simplest form.

Two Essential Steps

- The original expression is put into the sum-of-products form by repeated application of DeMorgan's theorem and multiplication of terms.
- The product terms are checked for common factors, and factoring is performed whenever possible.

Examples 1-4

Original	Simplified
ABC+AB'(A'C')'	A(B'+C)
ABC+ABC'+AB'C	A(B+C)
A'C(A'BD)'+A'BC'D'+AB'C	B'C+A'D'(B+C)
(A'+B)(A+B+D)D'	BD'

Examples 5-6

- (A'+B)(A+B'): equivalent form A'B'+AB
- AB'C+A'BD+C'D': cannot be simplified further.

Analysis procedure

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

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.

Analysis procedure

3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

8

Example

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

$F_1 = T_3 + T_2$

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

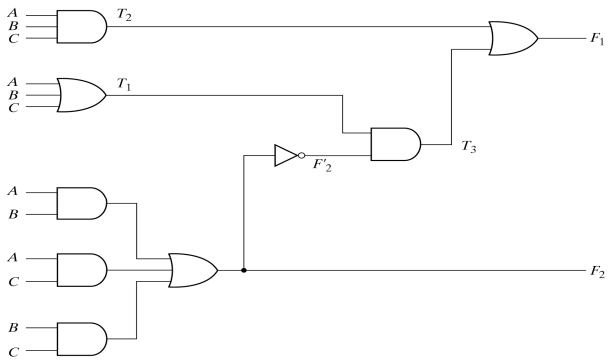


Fig. 4-2 Logic Diagram for Analysis Example

9

Derive truth table from logic diagram

- We can derive the truth table in Table 4-1 by using the circuit of Fig.4-2.

Table 4-1
Truth Table for the Logic Diagram of Fig. 4-2

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

Design procedure

1. Table4-2 is a Code-Conversion example, first, we can list the relation of the BCD and Excess-3 codes in the truth table.

Table 4-2
Truth Table for Code-Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

11

Designing Combinational Logic Circuits

1. Set up the truth table.
2. Write the AND term for each case where the output is a 1.
3. Write the sum-of-products expression for the output.
4. Simplify the output expression.
5. Implement the circuit for the final expression.

Example 7

- Design a logic circuit that is to produce a HIGH output when the voltage (represented by a four-bit binary number ABCD) is greater than 6V.

Example 8

- Four chairs A, B, C, and D are placed in a row. Each chair may be occupied (“1“) or empty (“0“). A Boolean function F is (“1“) if and only if there are two or more adjacent chairs that are empty.

  a) Give the truth table defining the Boolean F.
  b) Express F as a minterm expansion (standard sum of products).
  c) Express F as a maxterm expansion (standard product of sums).
  d) Simplify the minterm expansion of F.

Karnaugh Map Method

- A graphical device to simplify a logic expression.
- Generally work on examples with up to 4 input variables.
- From truth table to logic expression to K map.

Complete Simplification Process

- Step 1: Construct the K map and places 1s in those squares corresponding to the 1s in the truth table. Places 0s in the other squares.
- Step 2: Examine the map for adjacent 1s and loop those 1s which are not adjacent to any other 1s. (isolated 1s)
- Step 3: Look for those 1s which are adjacent to only one other 1. Loop any pair containing such a 1.
- Step 4: Loop any octet even when it contains some 1s that have already been looped.

Complete Simplification Process

- Step 5: Loop any quad that contains one or more 1s that have not already been looped, making sure to use the minimum number of loops.
- Step 6: Loop any pairs necessary to include any 1s have not already been looped, making sure to use the minimum number of loops.
- Step 7: Form the ORed sum of all the terms generated by each loop.

Filling K Map from Output Expression

- What to do when the desired output is presented as a Boolean expression instead of a truth table?
- Step 1: Convert the expression into SOP form.
- Step 2: For each product term in the SOP expression, place a 1 in each K-map square whose label contains the same combination of input values. Place a 0 in other squares.
- Example 9: y=C'(A'B'D'+D)+AB'C+D'

Don't-Care Conditions

- Some logic circuits can be designed so that there are certain input conditions for which there are no specified output levels.
- A circuit designer is free to make the output for any don't care condition either a 0 or a 1 in order to produce the simplest output expression.

Karnaugh map

2. For each symbol of the Excess-3 code, we use 1's to draw the map for simplifying Boolean function.

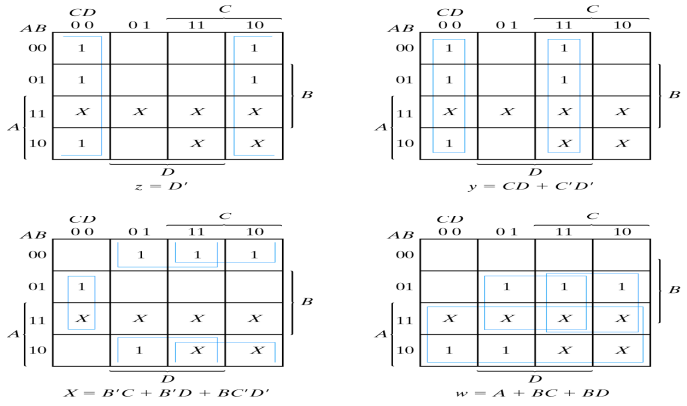


Fig. 4-3 Maps for BCD to Excess-3 Code Converter

20

Circuit implementation

z = D'; y = CD + C'D' = CD + (C + D)'

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

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

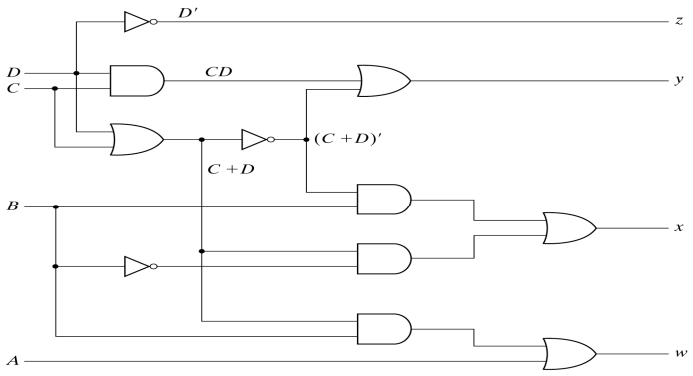


Fig. 4-4 Logic Diagram for BCD to Excess-3 Code Converter

21

Binary Adder-Subtractor

- A combinational circuit that performs the addition of two bits is called a half adder.
- The truth table for the half adder is listed below:

Table 4-3
Half Adder

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

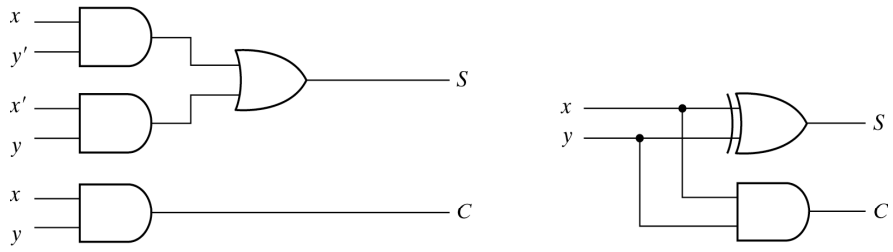
S: Sum
C: Carry

S = x'y + xy'

C = xy

22

Implementation of Half-Adder



(a) $S = xy' + x'y$
$C = xy$

(b) $S = x \oplus y$
$C = xy$

Fig. 4-5 Implementation of Half-Adder

23

Full-Adder

- One that performs the addition of three bits(two significant bits and a previous carry) is a full adder.

Table 4-4
Full Adder

x	y	z	C	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

24

Simplified Expressions

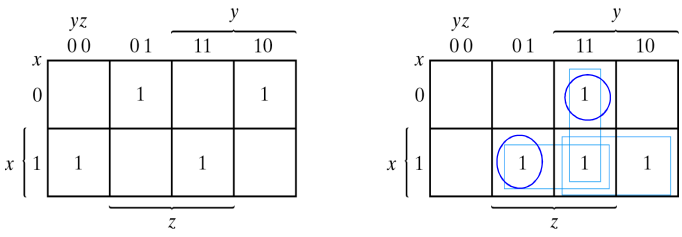


Fig. 4-6 Maps for Full Adder

S = x'y'z + x'yz' + xy'z' + xyz

C = xy + xz + yz

Full adder implemented in SOP

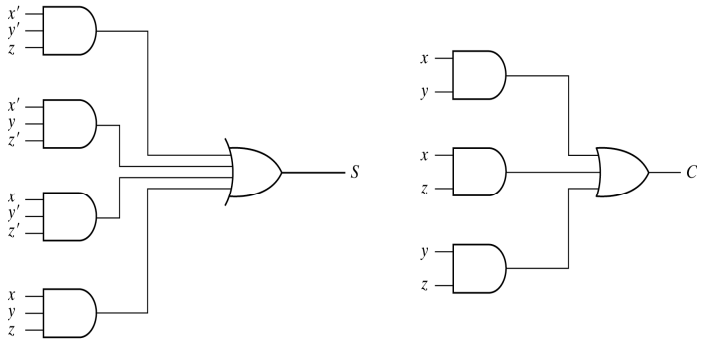


Fig. 4-7 Implementation of Full Adder in Sum of Products

Another implementation

- Full-adder can also implemented with two half adders and one OR gate (Carry Look-Ahead adder).

$S = z \oplus (x \oplus y)$
$= z'(xy' + x'y) + z(xy' + x'y)'$
$= xy'z' + x'yz' + xyz + x'y'z$
$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$

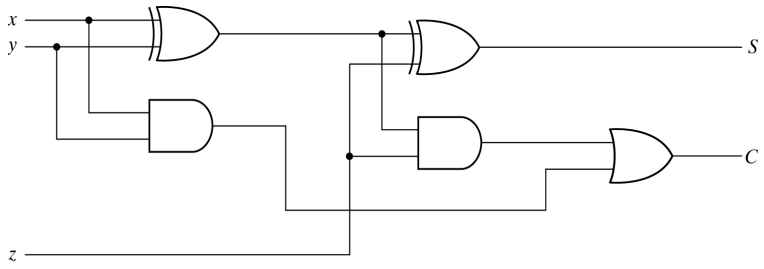


Fig. 4-8 Implementation of Full Adder with Two Half Adders and an OR Gate

27

Binary adder

- This is also called Ripple Carry Adder ,because of the construction with full adders are connected in cascade.

Subscript i:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

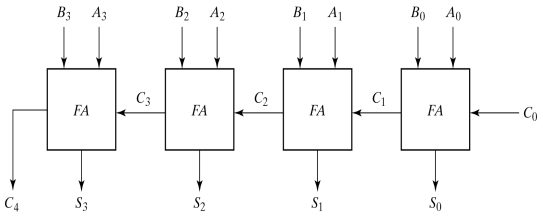


Fig. 4-9 4-Bit Adder

Carry Propagation

- Fig.4-9 causes a unstable factor on carry bit, and produces a longest propagation delay.
- The signal from C_i to the output carry C_{i+1}, propagates through an AND and OR gates, so, for an n-bit RCA, there are 2n gate levels for the carry to propagate from input to output.

29

Carry Propagation

- Because the propagation delay will affect the output signals on different time, so the signals are given enough time to get the precise and stable outputs.
- The most widely used technique employs the principle of carry look-ahead to improve the speed of the algorithm.

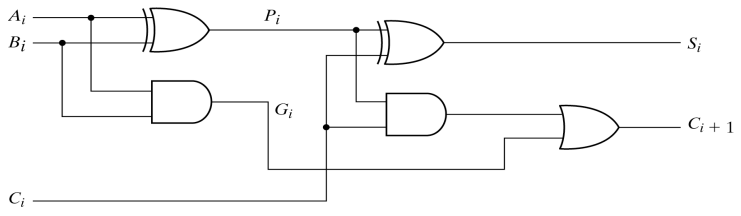


Fig. 4-10 Full Adder with P and G Shown

30

Boolean functions

$P_i = A_i \oplus B_i$ steady state value

$G_i = A_iB_i$ steady state value

Output sum and carry

$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_iC_i$

G_i : carry generate P_i : carry propagate

C_0 = input carry

$C_1 = G_0 + P_0C_0$

$C_2 = G_1 + P_1C_1 = G_1 + P_1G_0 + P_1P_0C_0$

$C_3 = G_2 + P_2C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_0$

- C_3 does not have to wait for C_2 and C_1 to propagate.

31

Logic diagram of carry look-ahead generator

- C_3 is propagated at the same time as C_2 and C_1.

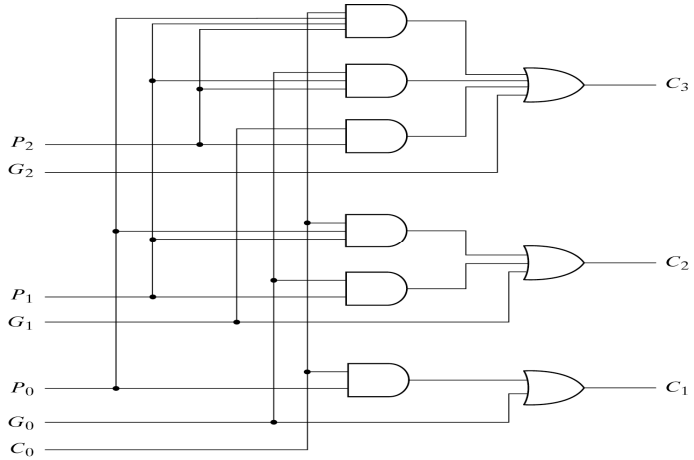


Fig. 4-11 Logic Diagram of Carry Lookahead Generator

32

4-bit adder with carry lookahead

- Delay time of n-bit CLAA = XOR + (AND + OR) + XOR

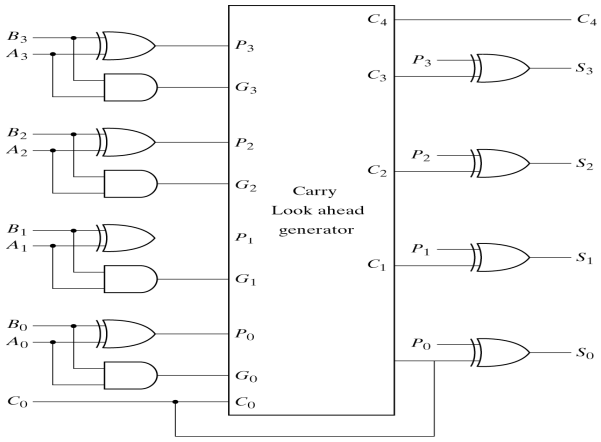


Fig. 4-12 4-Bit Adder with Carry Lookahead

33

Binary subtractor

M = 1→subtractor ; M = 0→adder

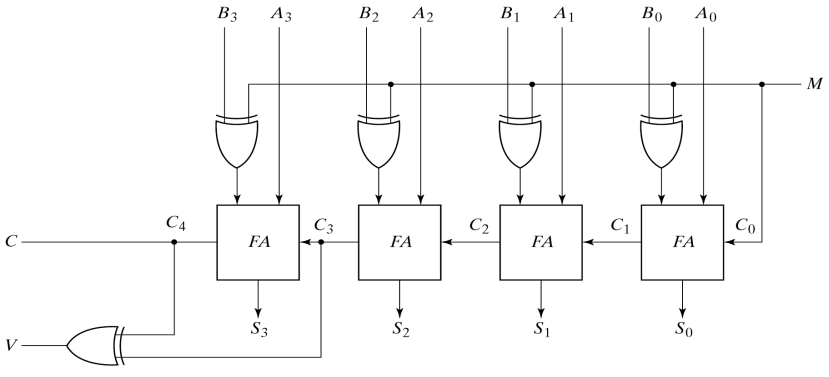


Fig. 4-13 4-Bit Adder Subtractor

34

Rules of BCD adder

- When the binary sum is greater than 1001, we obtain a non-valid BCD representation.
- The addition of binary 6(0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.
- To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8, we specify further that either Z_4 or Z_2 must have a 1.

$$C = K + Z_8Z_4 + Z_8Z_2$$

35

Implementation of BCD adder

- A decimal parallel adder that adds n decimal digits needs n BCD adder stages.
- The output carry from one stage must be connected to the input carry of the next higher-order stage.

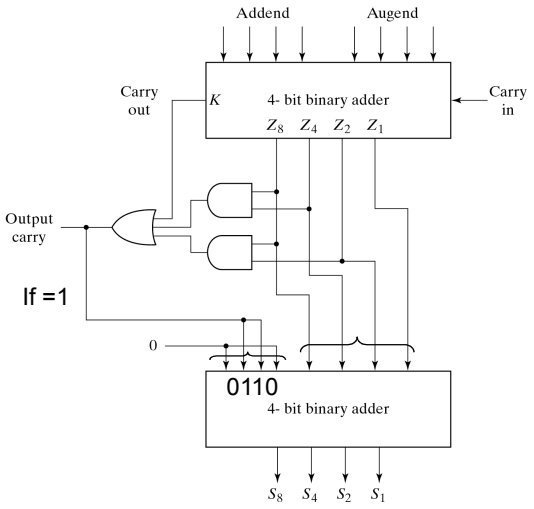


Fig. 4-14 Block Diagram of a BCD Adder

Binary multiplier

- Usually there are more bits in the partial products and it is necessary to use full adders to produce the sum of the partial products.

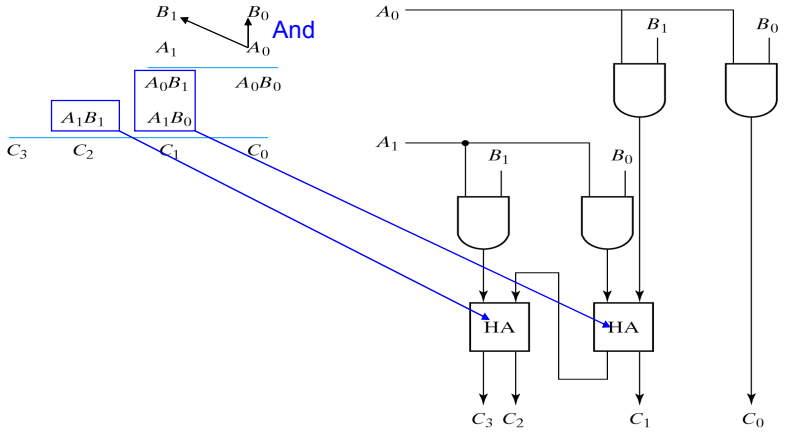


Fig. 4-15 2-Bit by 2-Bit Binary Multiplier

37

4-bit by 3-bit binary multiplier

- For J multiplier bits and K multiplicand bits we need (J X K) AND gates and (J – 1) K-bit adders to produce a product of J+K bits.
- K=4 and J=3, we need 12 AND gates and two 4-bit adders.

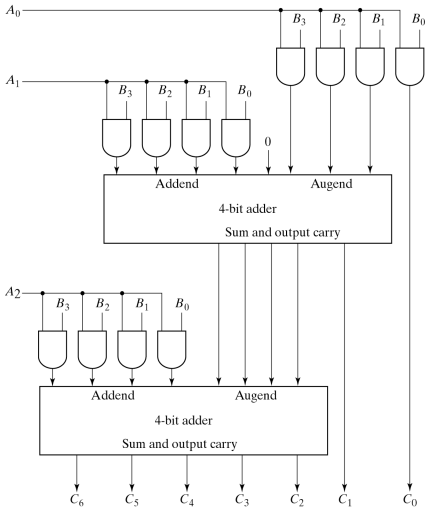


Fig. 4-16 4-Bit by 3-Bit Binary Multiplier

Decoders

- The decoder is called n-to-m-line decoder, where $m \leq 2^n$.
- the decoder is also used in conjunction with other code converters such as a BCD-to-seven_segment decoder.
- 3-to-8 line decoder: For each possible input combination, there are seven outputs that are equal to 0 and only one that is equal to 1.

39

Implementation and truth table

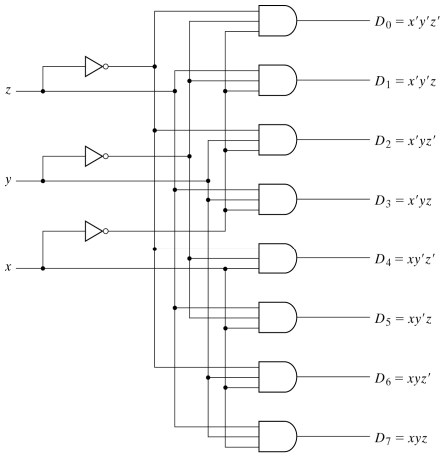


Fig. 4-18 3-to-8-Line Decoder

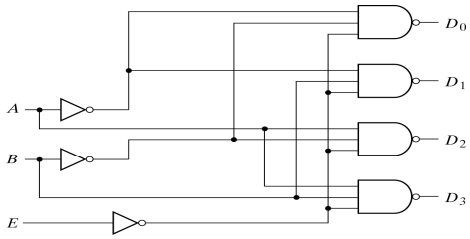
Table 4-6
Truth Table of a 3-to-8-Line Decoder

Inputs			Outputs							
x	y	z	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

40

Decoder with enable input

- Some decoders are constructed with NAND gates, it becomes more economical to generate the decoder minterms in their complemented form.
- As indicated by the truth table , only one output can be equal to 0 at any given time, all other outputs are equal to 1.



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

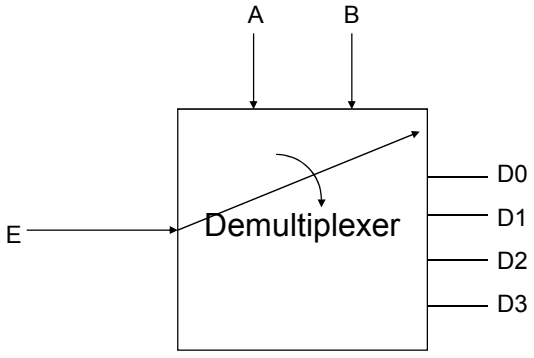
(b) Truth table

Fig. 4-19 2-to-4-Line Decoder with Enable Input

41

Demultiplexer

- A decoder with an enable input is referred to as a decoder/demultiplexer.
- The truth table of demultiplexer is the same with decoder.



42

3-to-8 decoder with enable implement the 4-to-16 decoder

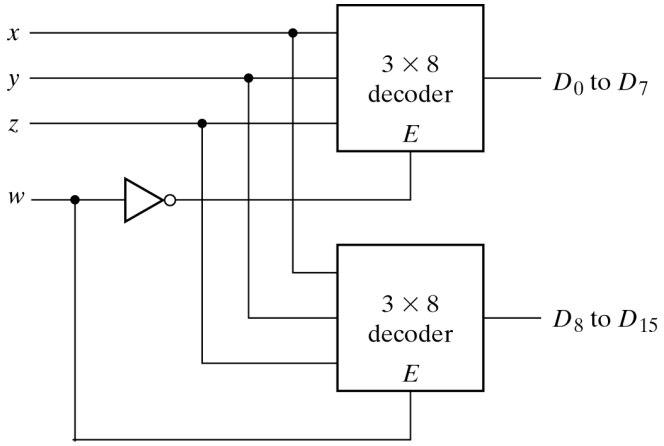


Fig. 4-20 4 × 16 Decoder Constructed with Two 3 × 8 Decoders

43

Implementation of a Full Adder with a Decoder

- From table 4-4, we obtain the functions for the combinational circuit in sum of minterms:

  S(x, y, z) = ∑(1, 2, 4, 7)

  C(x, y, z) = ∑(3, 5, 6, 7)

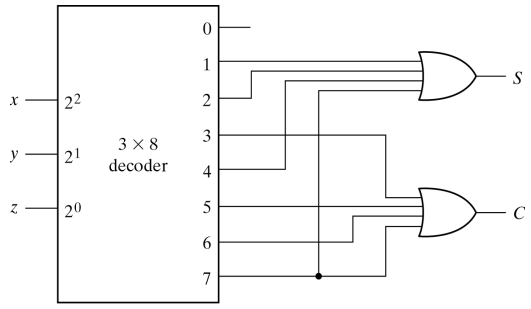


Fig. 4-21 Implementation of a Full Adder with a Decoder

44

Encoders

- An encoder is the inverse operation of a decoder.
- We can derive the Boolean functions by table 4-7

$$z = D_1 + D_3 + D_5 + D_7$$
$$y = D_2 + D_3 + D_6 + D_7$$
$$x = D_4 + D_5 + D_6 + D_7$$

Table 4-7
Truth Table of Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
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

Priority encoder

- If two inputs are active simultaneously, the output produces an undefined combination. We can establish an input priority to ensure that only one input is encoded.
- Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; the output is the same as when D_0 is equal to 1.
- The discrepancy tables on Table 4-7 and Table 4-8 can resolve aforesaid condition by providing one more output to indicate that at least one input is equal to 1.

46

Priority encoder

V=0→no valid inputs
V=1→valid inputs

X's in output columns represent don't-care conditions
X's in the input columns are useful for representing a truth table in condensed form.
Instead of listing all 16 minterms of four variables.

Table 4-8
Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

47

4-input priority encoder

- Implementation of table 4-8

$x = D_2 + D_3$
$y = D_3 + D_1D'_2$
$V = D_0 + D_1 + D_2 + D_3$

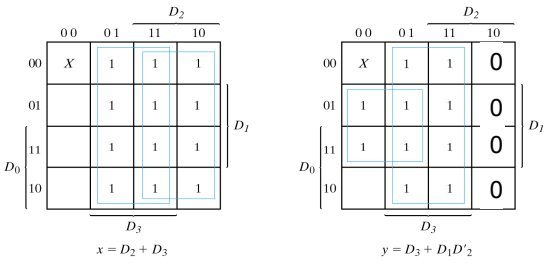


Fig. 4-22 Maps for a Priority Encoder

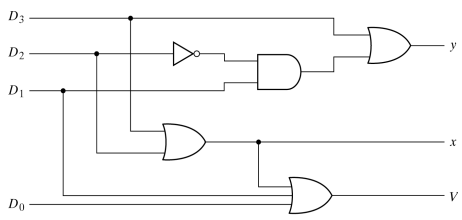


Fig. 4-23 4-Input Priority Encoder

48

Multiplexers

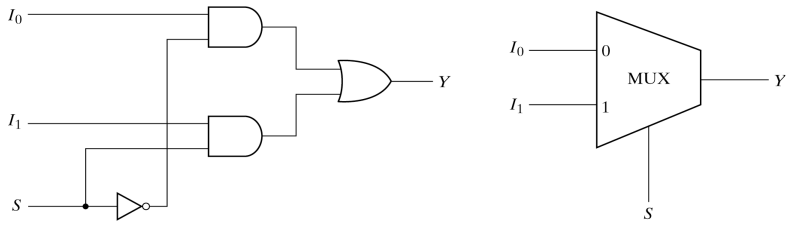
$S = 0, Y = I_0$

$S = 1, Y = I_1$

Truth Table→

S	Y
0	I_0
1	I_1

$Y = S'I_0 + SI_1$

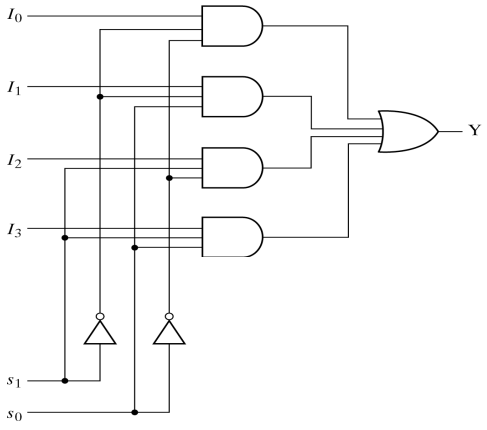


(a) Logic diagram (b) Block diagram

Fig. 4-24 2-to-1-Line Multiplexer

49

4-to-1 Line Multiplexer



(a) Logic diagram

s_1	s_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

Fig. 4-25 4-to-1-Line Multiplexer

50

Quadruple 2-to-1 Line Multiplexer

- Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic. Compare with Fig4-24.

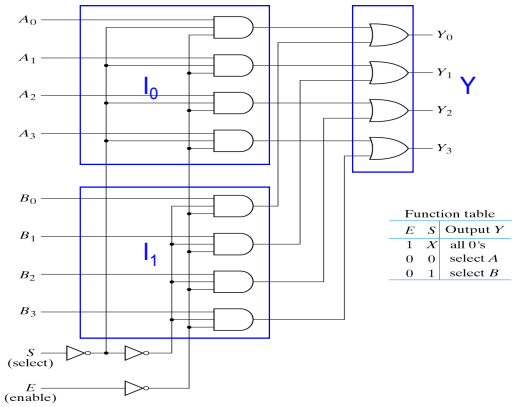


Fig. 4-26 Quadruple 2-to-1-Line Multiplexer

51

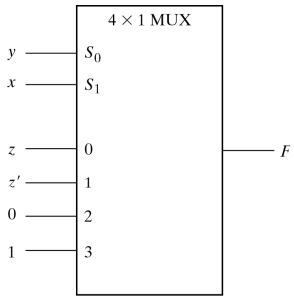
Boolean function implementation

- A more efficient method for implementing a Boolean function of n variables with a multiplexer that has n-1 selection inputs.

  F(x, y, z) = Σ(1,2,6,7)

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

Fig. 4-27 Implementing a Boolean Function with a Multiplexer

52

4-input function with a multiplexer

F(A, B, C, D) = Σ(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 = 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	

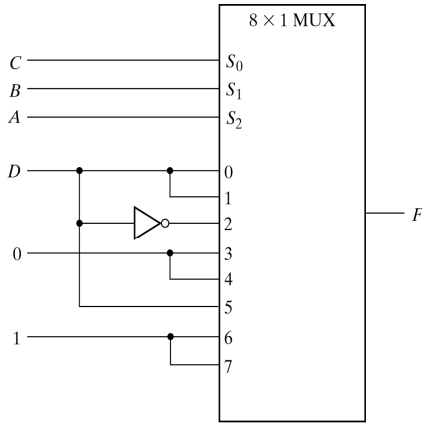


Fig. 4-28 Implementing a 4-Input Function with a Multiplexer

53

Thank You

Next topic