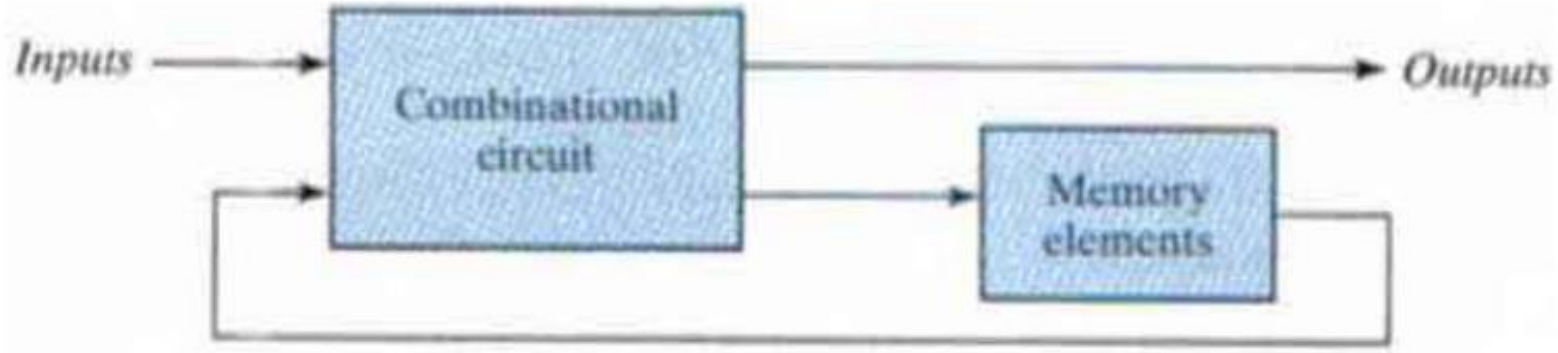
EE1201: Basic Electrical Engineering

Combinational Circuits

Combinational Circuits

- Logic circuits of two types: combinational & sequential
- For combinational circuits, outputs depend on the present values of the inputs
- Sequential circuits have memory elements, outputs depend upon present & past values of the inputs



Design of Combinational Circuits

- The design starts with the verbal outline of the problem & culminates in logic circuit diagram

1. Problem statement
2. Determination of the no. of input & output variables
3. Assignment of letter symbols to variables
4. Derivation of the truth table that defines the required relationships between the inputs & the outputs
5. Simplification of the Boolean functions
6. Logic diagram

- Proper interpretation of the verbal statement
- Algebraic manipulation or map method to simplify the output functions
- Variety of simplified expressions maybe available
- Some constraints like availability of the types of gates, minimum no. of gates, minimum no. of inputs to a gate, propagation time of the circuit etc. may dictate certain things
- Logic diagrams useful for visualizing the gate implementation of the expressions

Adders

- Simplest & most basic arithmetic operation, addition of two binary digits
- Four possible operations:
  1. 0 + 0 = 0
  2. 0 + 1 = 1
  3. 1 + 0 = 1
  4. 1 + 1 = 10
- Two outputs, sum & carry
- The carry obtained from the previous stage needs to be added to the next pair of significant bits
- Addition of two bits: half adder
- Addition of three bits: full adder

- Half adder
- Two inputs (x & y) & two outputs (S for sum & C for carry)
- Truth table

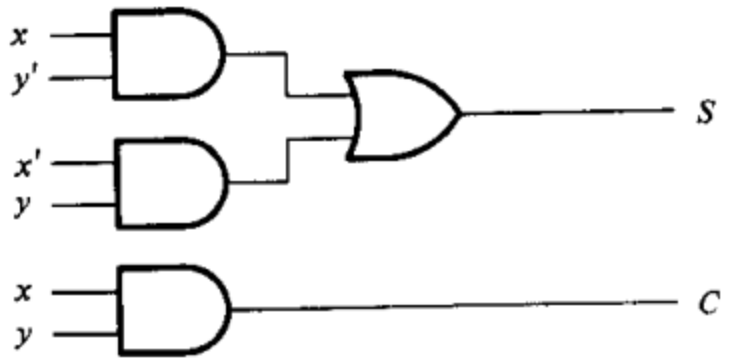
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- Simplified expressions

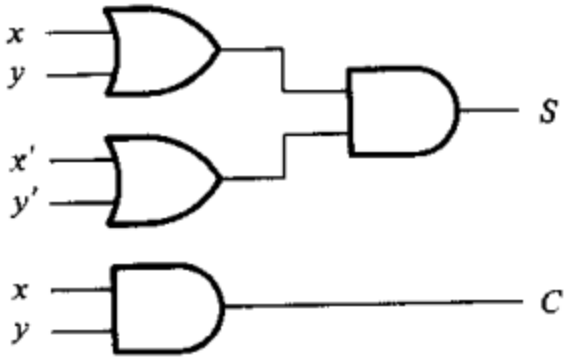
$$S = x'y + xy' = x \oplus y$$

$$C = xy$$

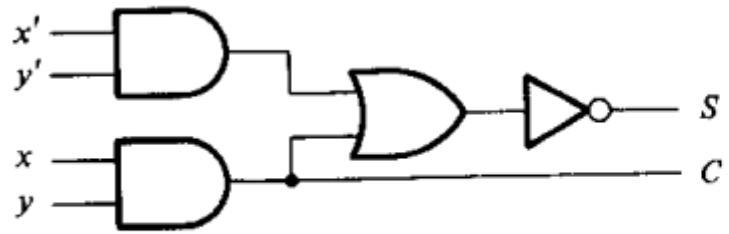
- Logic diagram, various implementation



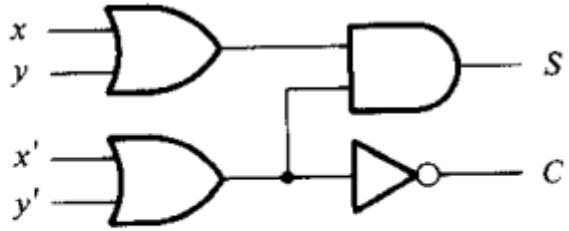
(a) $S = xy' + x'y$
$C = xy$



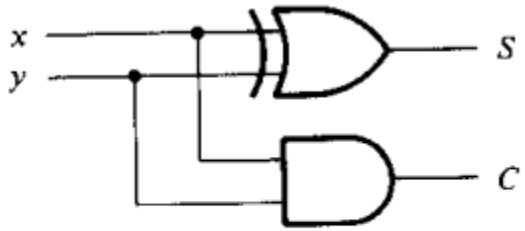
(b) $S = (x + y)(x' + y')$
$C = xy$



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



(d) $S = (x + y)(x' + y')$
$C = (x' + y')'$

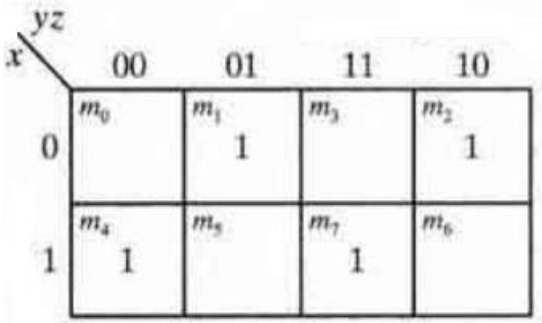


(e) $S = x \oplus y$
$C = xy$

- Full adder
- Three inputs (x, y, z) & two outputs (S for sum & C for carry)
- Truth table

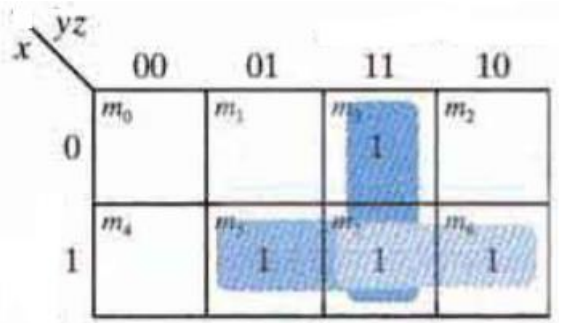
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- Simplified expressions



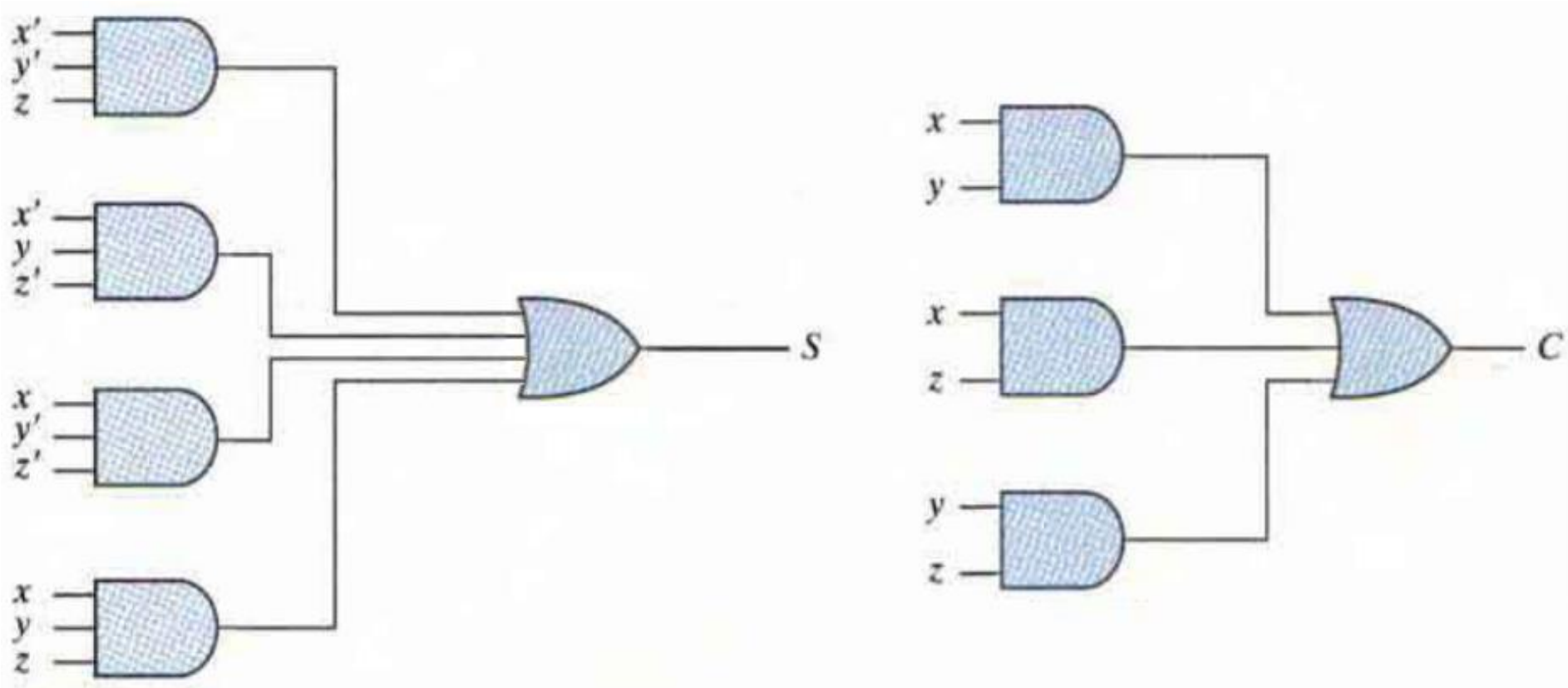
$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$= x \oplus y \oplus z$$

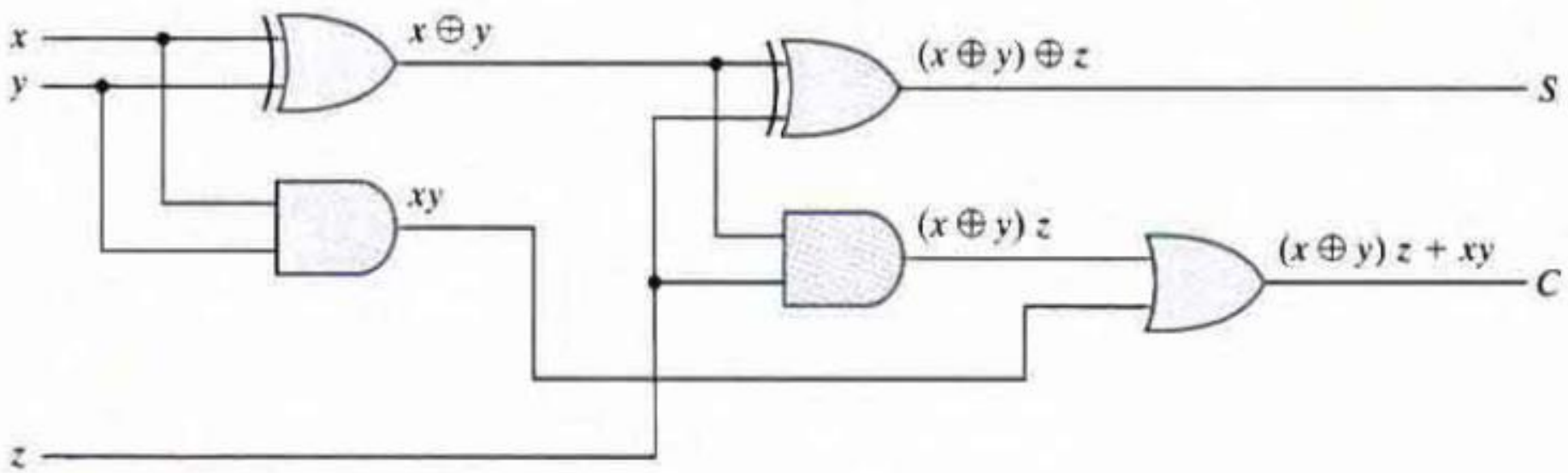


$$C = xy + yz + zx$$

- Logic diagram



- The carry output C which equals xy + yz + zx can as well be expressed as xy + z(x⊕y)
- Full adder with two half adders & an OR gate



Subtractors

- Subtraction maybe accomplished using complements, conversion to addition
- With logic circuits in a direct manner
- Subtrahend bit subtracted from the corresponding minuend bit
- If minuend bit smaller than the subtrahend bit, 1 borrowed from next significant position
- Two outputs, borrow & difference
- Half subtractor & full subtractor

- Half subtractor
- Subtracts two bits (x-y) & produces two outputs difference (D) & borrow (B)
- Truth table

x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

- Simplified expressions

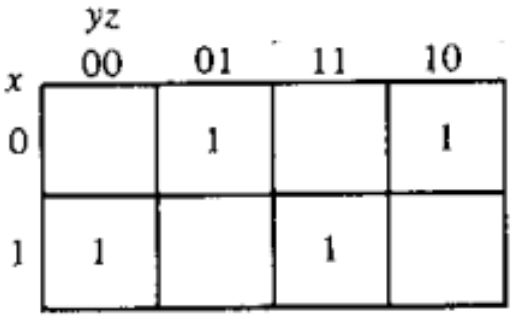
$$D = x'y + xy' = x \oplus y$$

$$B = x'y$$

- Full subtractor
- Three inputs (x, y, z) & two outputs (D for difference & B for borrow)
- Truth table

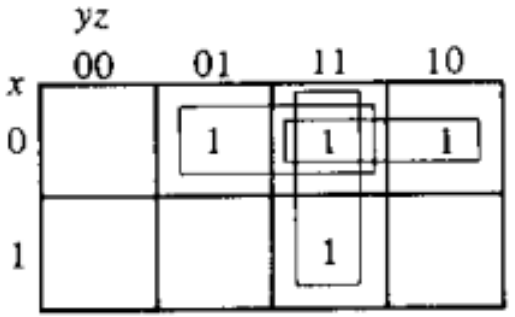
x	y	z	B	D
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

- Simplified expressions



$$D = x'y'z + x'yz' + xy'z' + xyz$$

$$= x \oplus y \oplus z$$



$$B = x'y + x'z + yz$$

Binary Adders

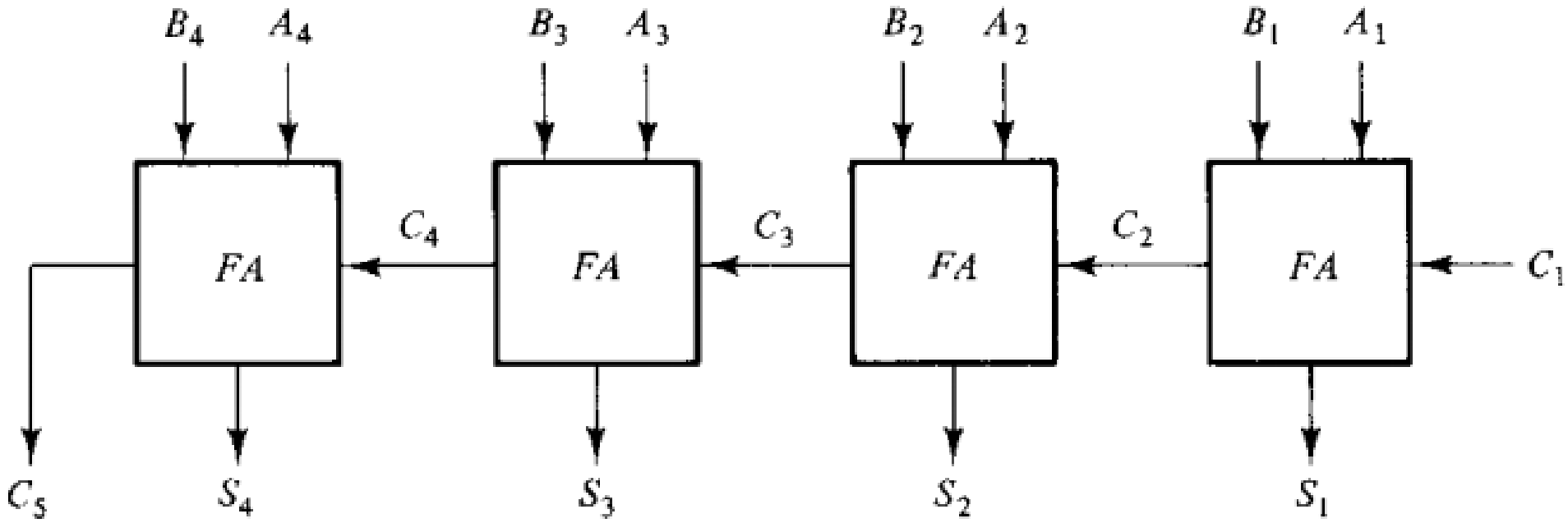
- Full adder forms the sum of two bits & a previous carry
- Two binary numbers of n bits can be added using the full adder

Subscript i	4	3	2	1	
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}

- Two ways to do, serially or parallely

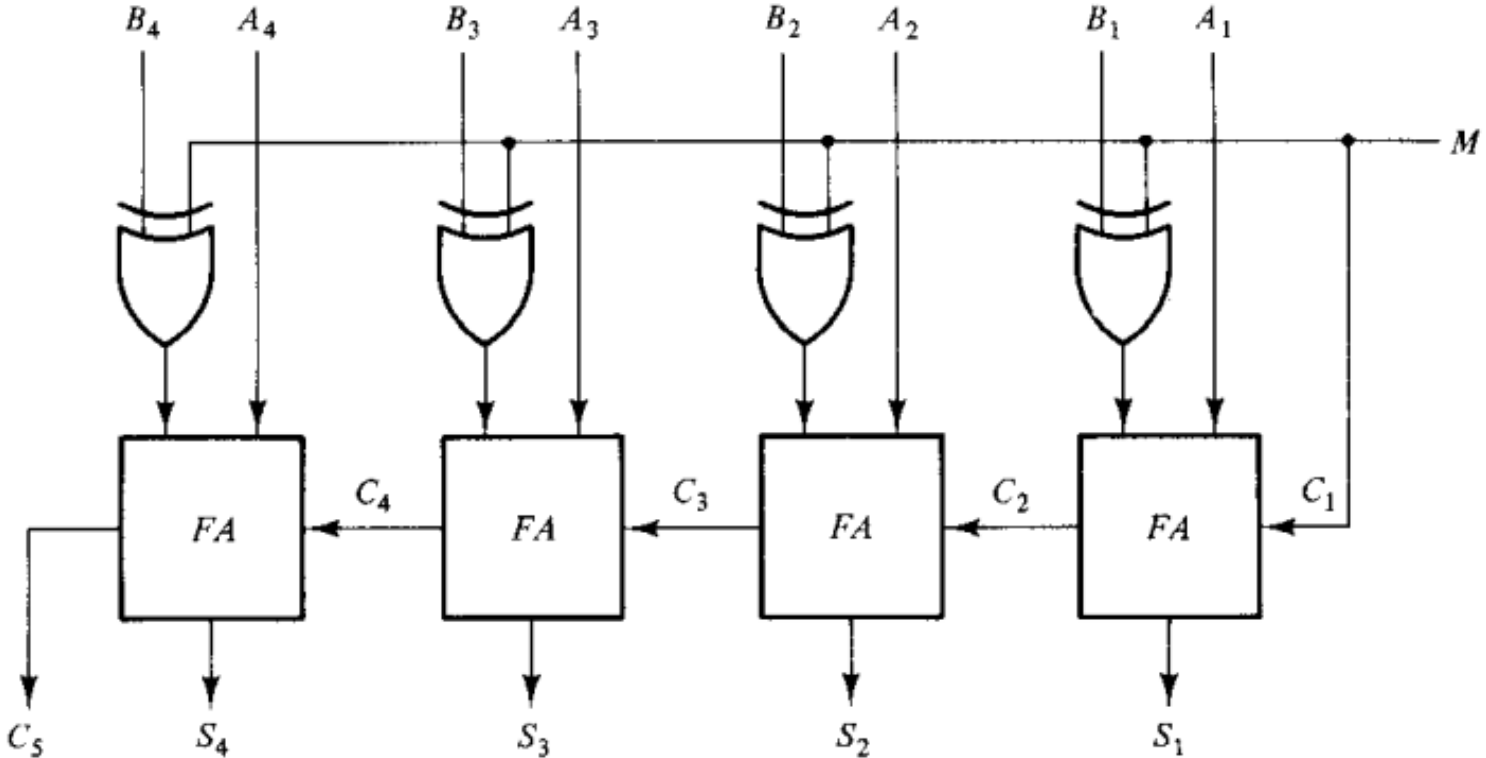
Binary Parallel Adder

- Full adders connected in a chain
- Output carry from previous full adder connected to the input carry of the next full adder



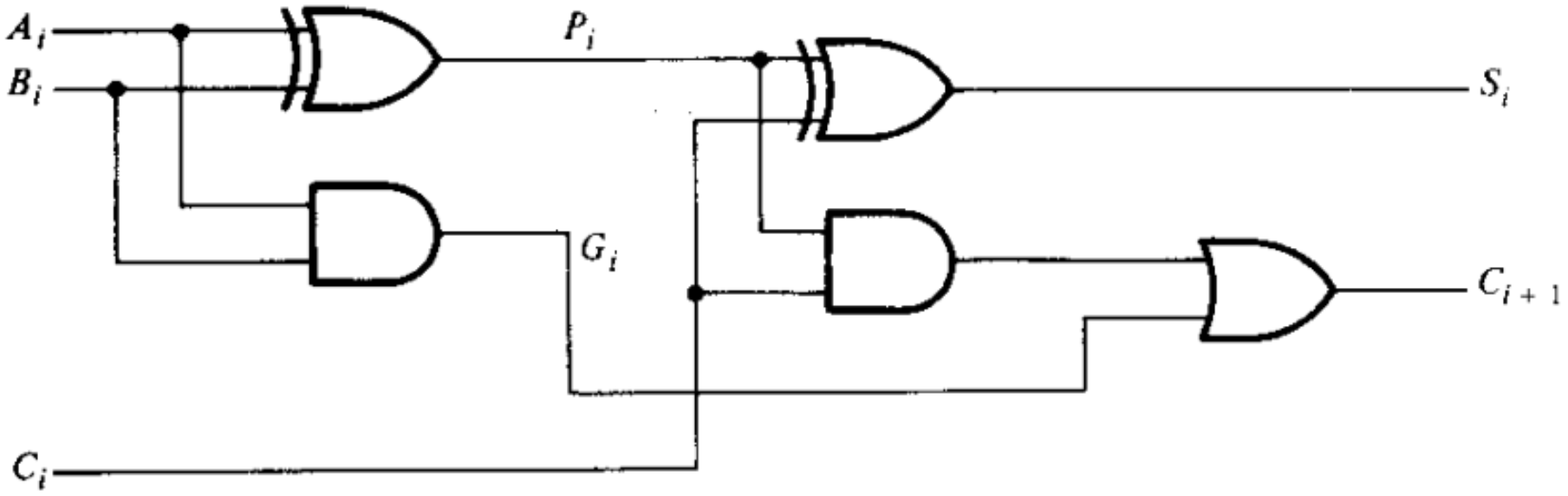
Binary Adder-Subtractor

- Subtraction using complements
- A – B = A + (2's complement of B)
- Parallel adder with inverters placed between B inputs & full adder inputs, with carry input 1
- Combine addition & subtraction into one circuit



Carry Propagation

- Parallel addition, all augend & addend bits available at the same time
- Finite time for signal propagation through the gates
- Carry propagation, longest delay time, limiting factor in determining the speed



$$P_i = A_i \oplus B_i \qquad S_i = P_i \oplus C_i$$

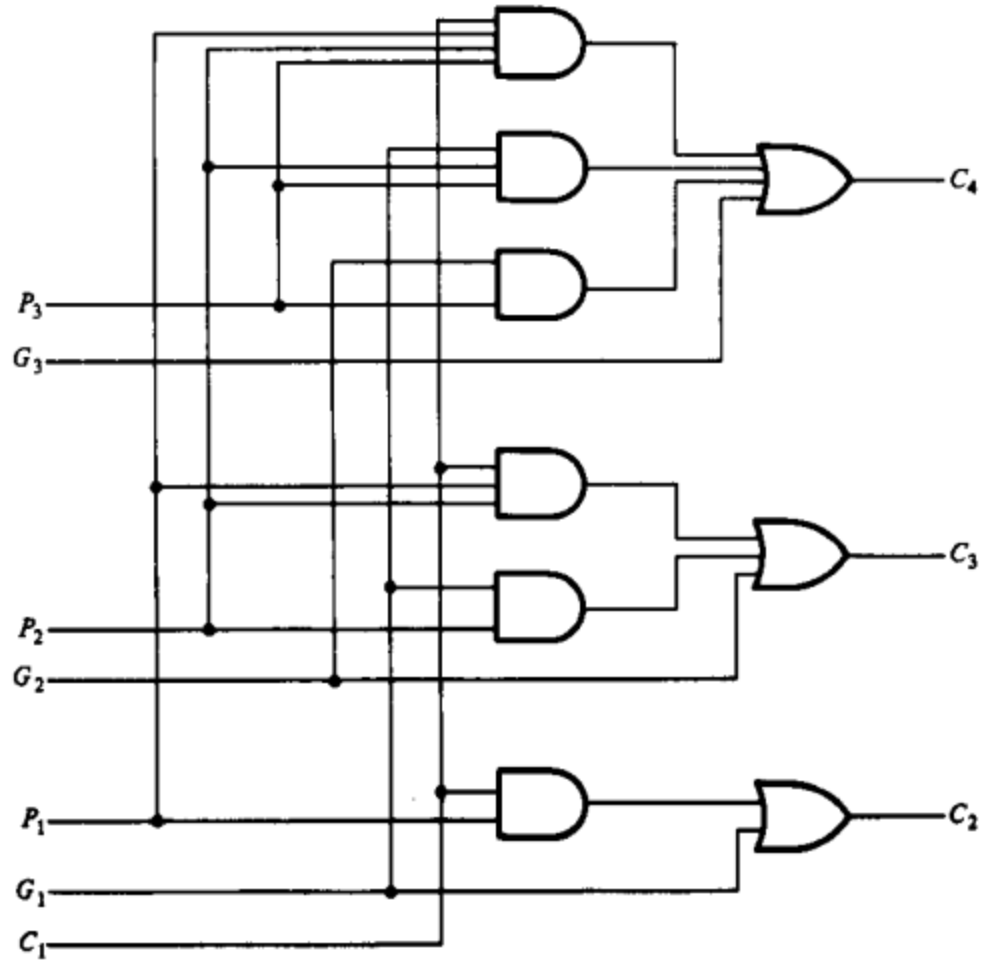
$$G_i = A_i B_i \qquad C_{i+1} = G_i + P_i C_i$$

- Carry look ahead adder

$$C_2 = G_1 + P_1 C_1$$

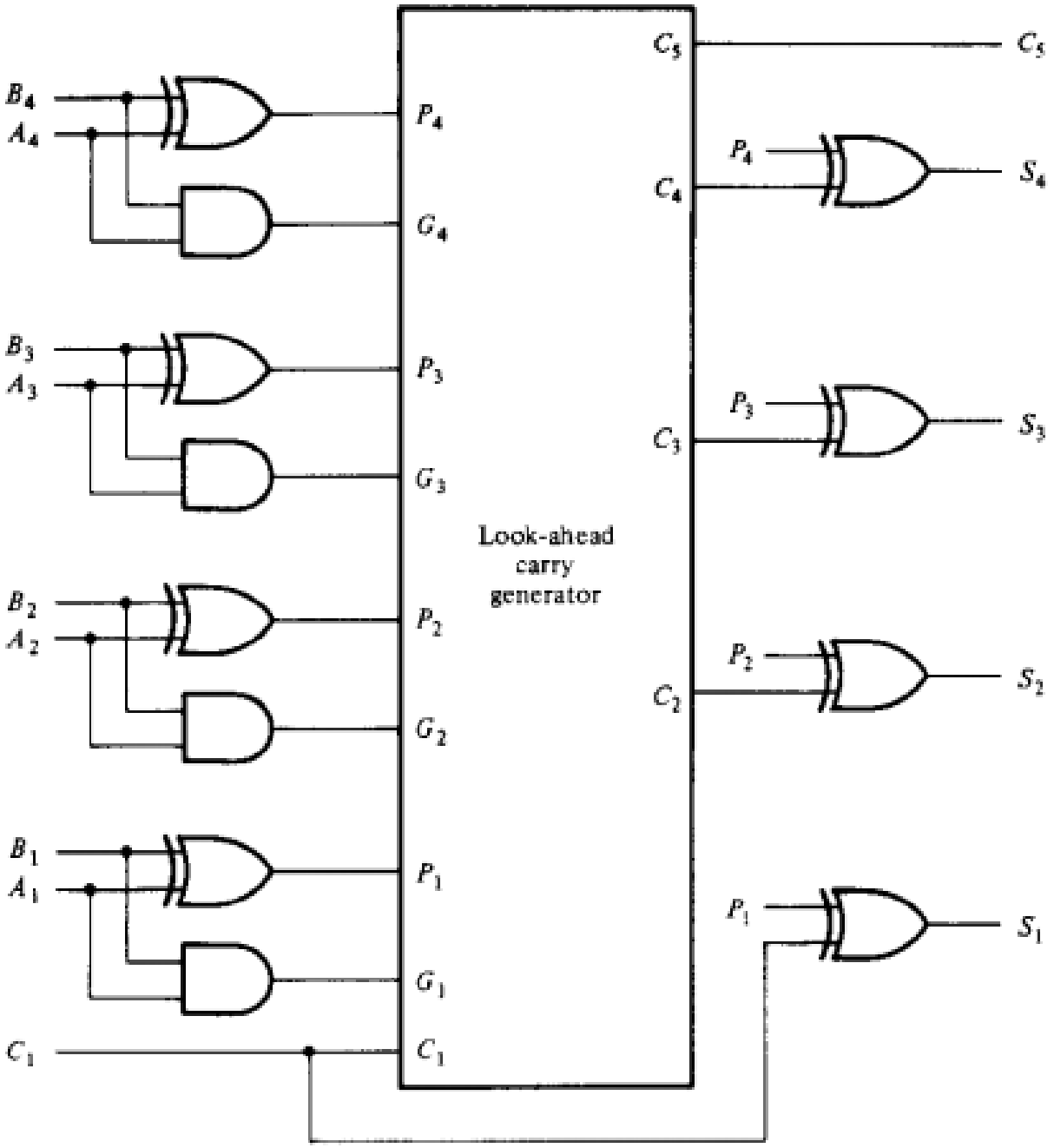
$$C_3 = G_2 + P_2 C_2 = G_2 + P_2 \left(G_1 + P_1 C_1 \right) = G_2 + P_2 G_1 + P_2 P_1 C_1$$

$$C_4 = G_3 + P_3 C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 C_1$$



- Sum of product form, one level of AND gate followed by one level of OR gate
- Increase complexity to improve speed

- 4 bit carry look ahead adder



BCD Adder

- Arithmetic addition in decimal system
- Addition of decimal digits in BCD with a possible carry from previous stage
- 9 inputs, truth table with 2^9=512 entries
- Use of 4 bit binary adders
- Maximum sum 9+9+1=19 & in binary form
- Output required in BCD form, proper conversion from binary to BCD output

- BCD adder table

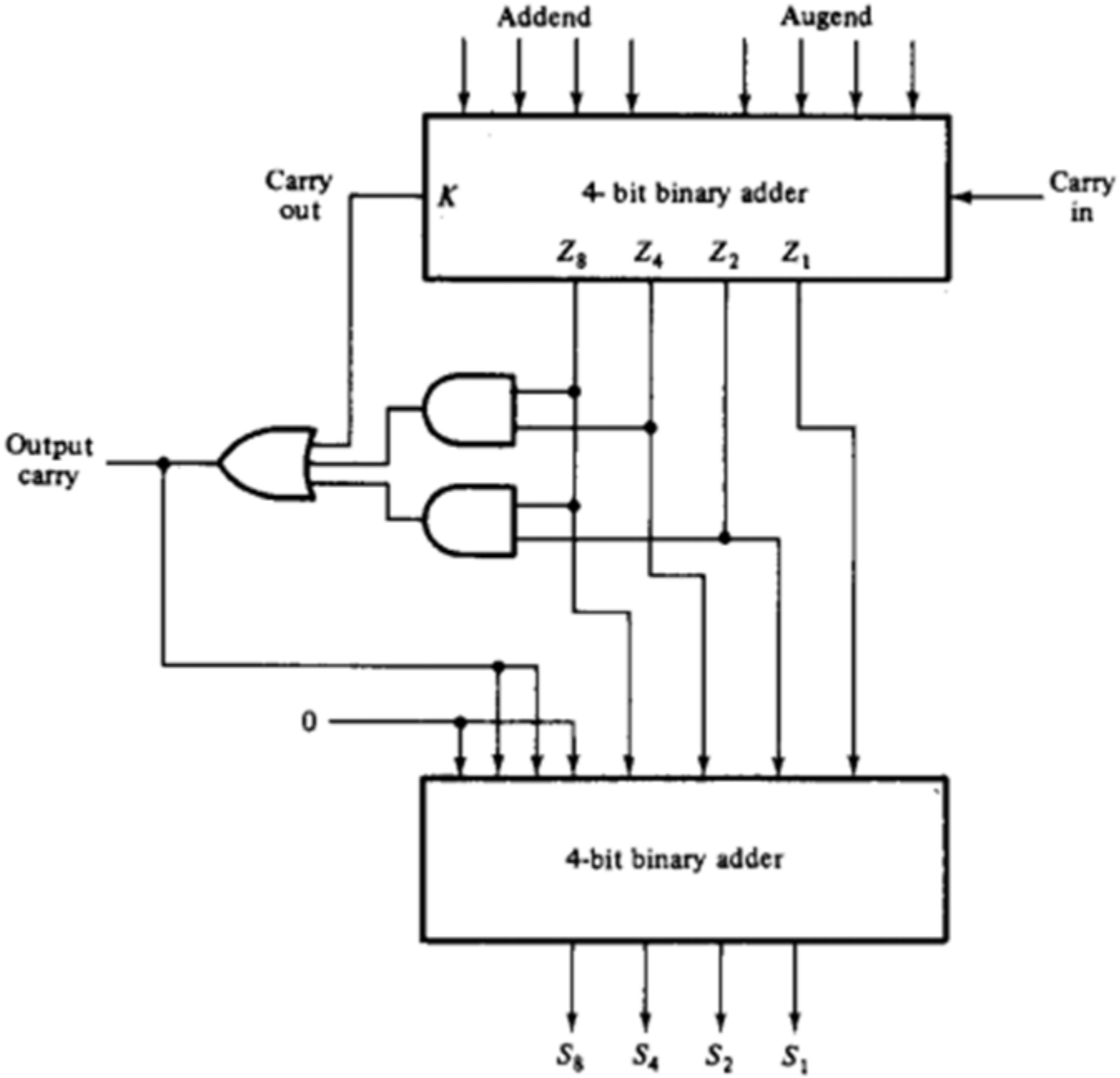
Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	C	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

- If binary sum less than or equal to 1001 no conversion needed
- If it is greater than 1001, addition of binary 6 (0110) converts it to correct BCD representation
- Condition for correction & output carry

$$C = K + Z_8 Z_4 + Z_8 Z_2$$

- Extra 4-bit binary adder to add 0110
- Decimal parallel adder with n decimal digits will need n such BCD adders with output carry of one stage connected to the input carry of the next higher order stage

- BCD adder block diagram



Magnitude Comparators

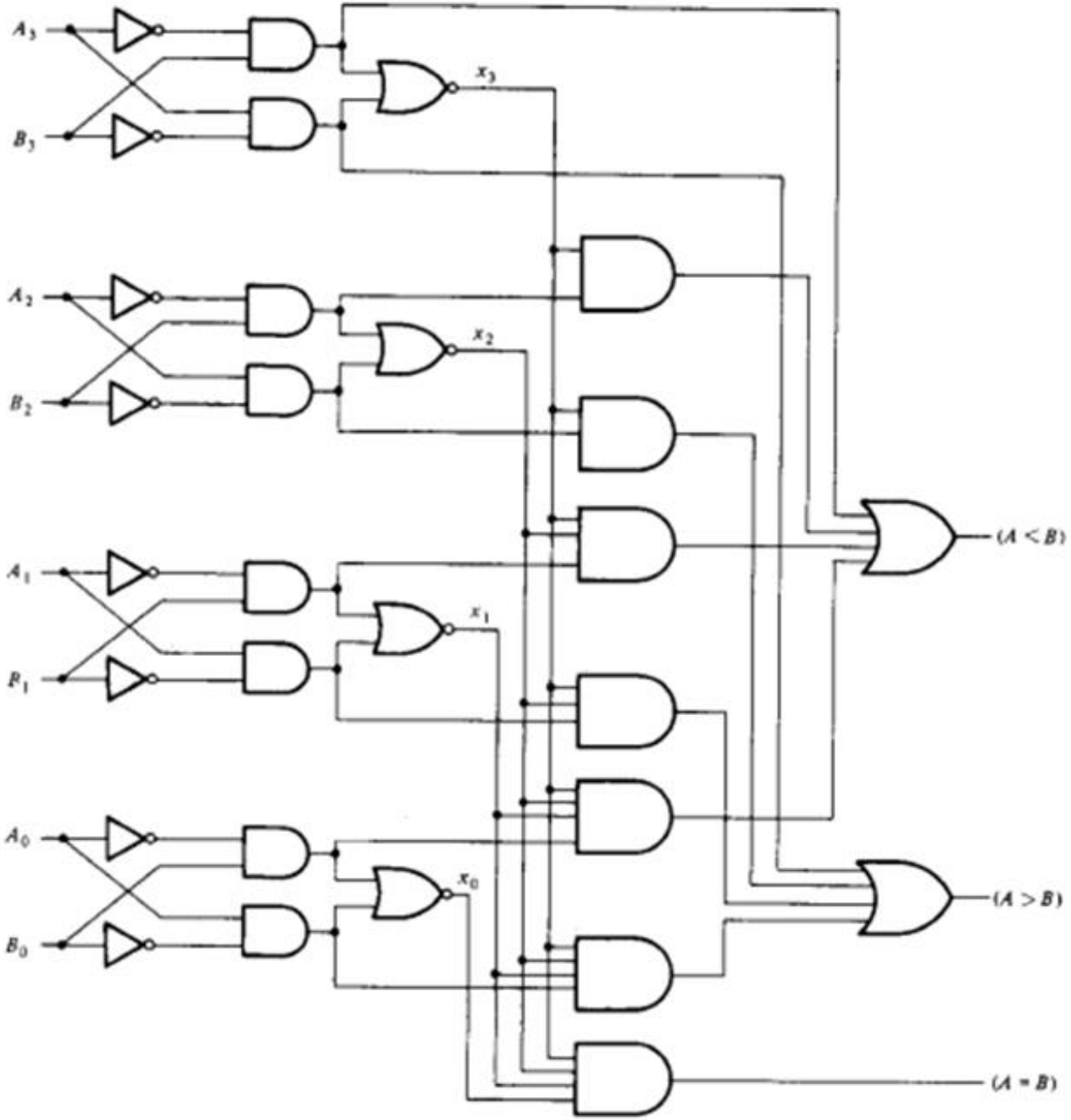
- Compare two numbers to determine if one number is greater than, less than, or equal to the other number
- Two n bit numbers with 2^{2n} entries- truth table, well defined inherent regularity- algorithmic approach
- Two numbers $A=A_3A_2A_1A_0$ & $B=B_3B_2B_1B_0$
- XNOR (equivalence) gate, if inputs are same output is 1, otherwise output is 0

$$x_i = A_iB_i + A_i'B_i' \; ; \; (A = B) = x_3x_2x_1x_0$$

$$(A > B) = A_3B_3' + x_3A_2B_2' + x_3x_2A_1B_1' + +x_3x_2x_1A_0B_0'$$

$$(A < B) = A_3'B_3 + x_3A_2'B_2 + x_3x_2A_1'B_1 + x_3x_2x_1A_0'B_0$$

- Magnitude comparator circuit



Some code converters

- Use of different codes by different systems
- Output of one system as input to the other
- Code converters for making the systems compatible
- Input lines supply the bit combination of elements specified by one code
- Output lines generate the bit combination in another code
- BCD to excess-3 conversion
- Binary to gray and vice versa

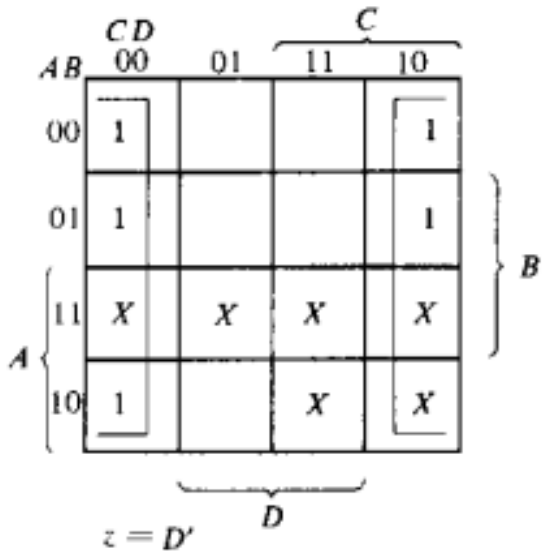
BCD to excess-3 conversion

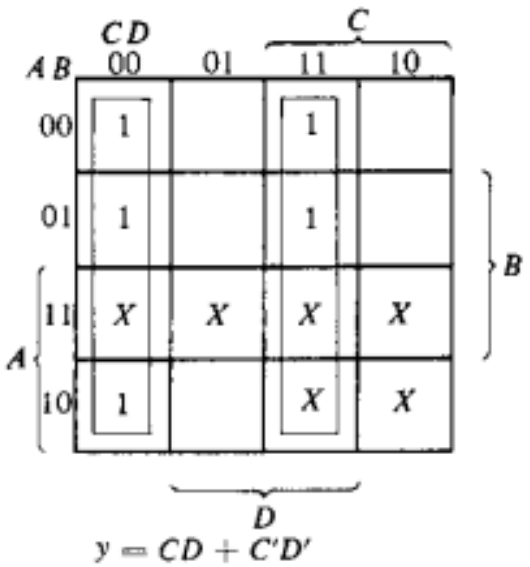
- 4 bits for representation of decimal digits, 4 inputs A, B, C, D & 4 outputs w, x, y, z

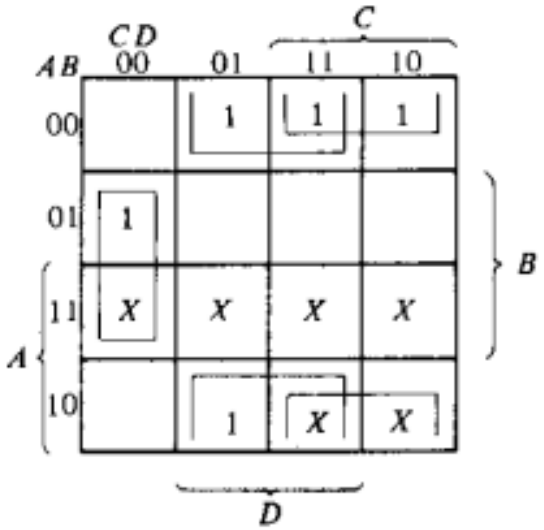
Input BCD				Output Excess-3 Code			
A	*B*	*C*	*D*	*w*	*x*	*y*	*z*
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

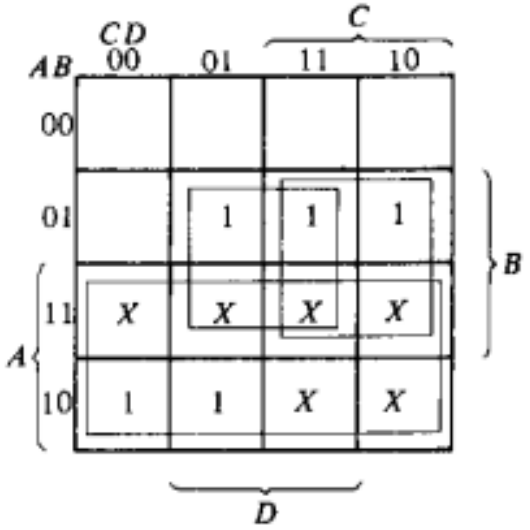
M Mano Digital Design

- 16 possible combinations, 6 are unused, treated as don’t care while simplifying









M Mano Digital Design

- Logic diagram

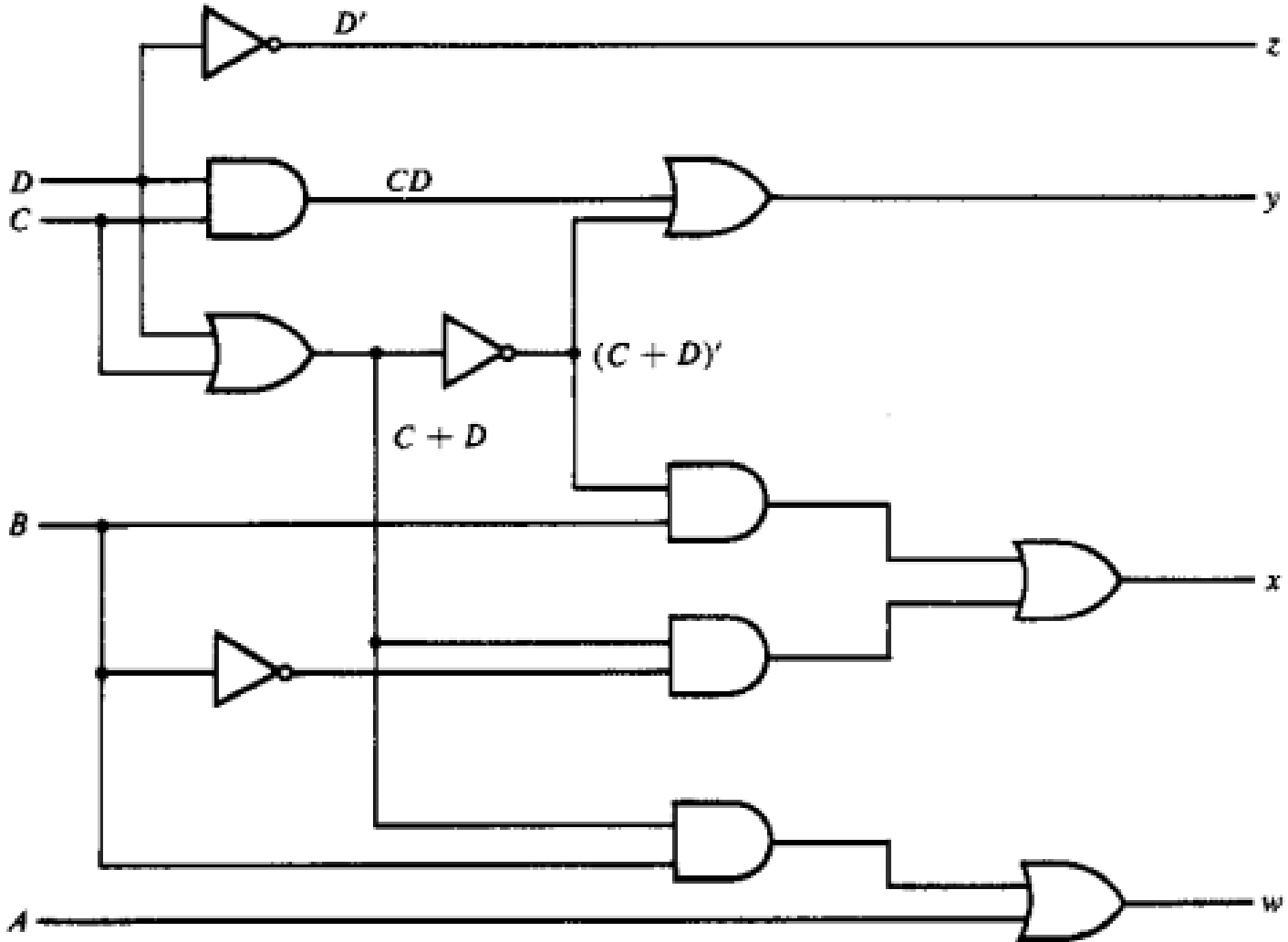
$$z = D'$$

$$y = CD + C'D' = CD + (C + D)'$$

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

$$= B'(C + D) + B(C + D)'$$

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

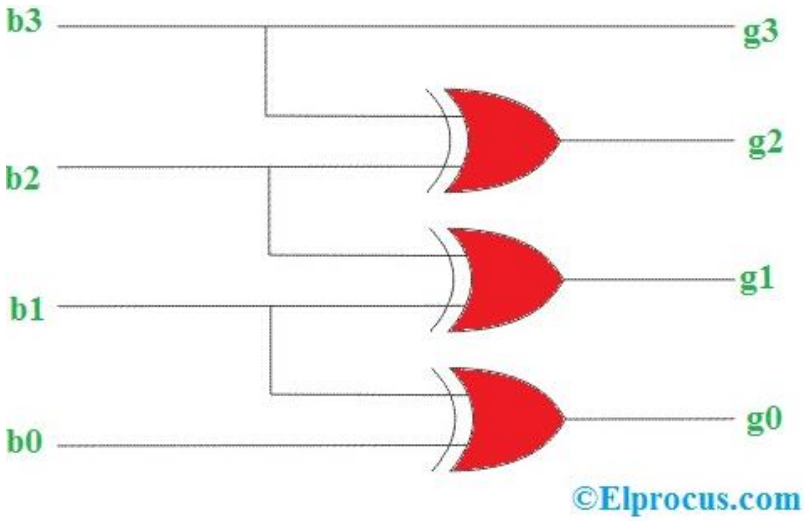


Binary to gray code conversion

- A binary code $b_3b_2b_1b_0$ needs to be converted to the gray code $g_3g_2g_1g_0$
- Use of XOR gates

b_3	b_2	b_1	b_0
↓	↓	↓	↓
g_3	g_2	g_1	g_0
b_3	$b_3 \oplus b_2$	$b_2 \oplus b_1$	$b_1 \oplus b_0$

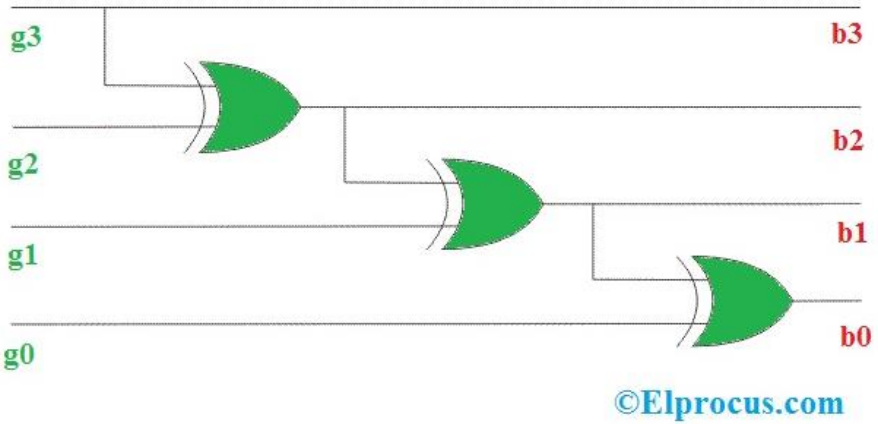
1	1	0	1
↓	↓	↓	↓
1	0	1	1
1	$1 \oplus 1$	$1 \oplus 0$	$0 \oplus 1$



Gray to binary code conversion

- A gray code $g_3g_2g_1g_0$ needs to be converted to the binary code $b_3b_2b_1b_0$
- Use of XOR gates

g_3	g_2	g_1	g_0
↓	↓	↓	↓
b_3	b_2	b_1	b_0
g_3	$b_3 \oplus g_2$	$b_2 \oplus g_1$	$b_1 \oplus g_0$
1	0	1	1
↓	↓	↓	↓
1	1	0	1
1	$1 \oplus 0$	$1 \oplus 1$	$0 \oplus 1$



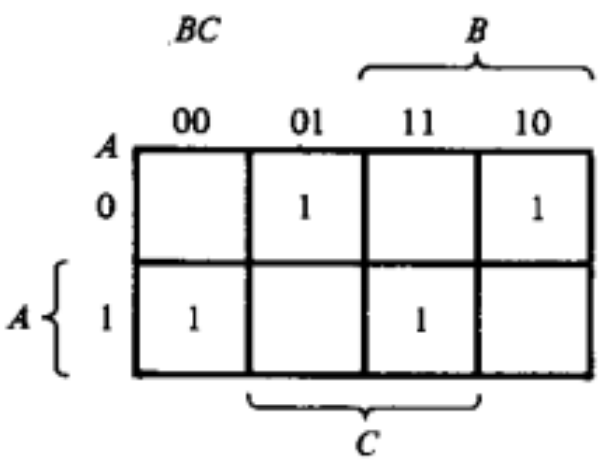
Error detection codes

- Parity generation & checking
- XOR gates useful, it is equal to 1 if odd number of variables are equal to 1, odd function
- For 3 variables

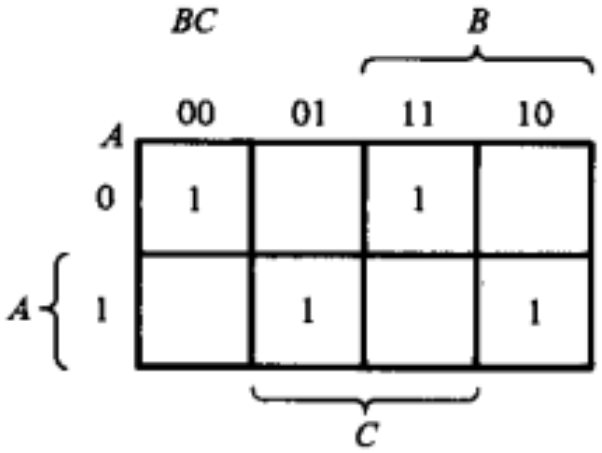
$$A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C$$

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

$$= \sum(1,2,4,7)$$



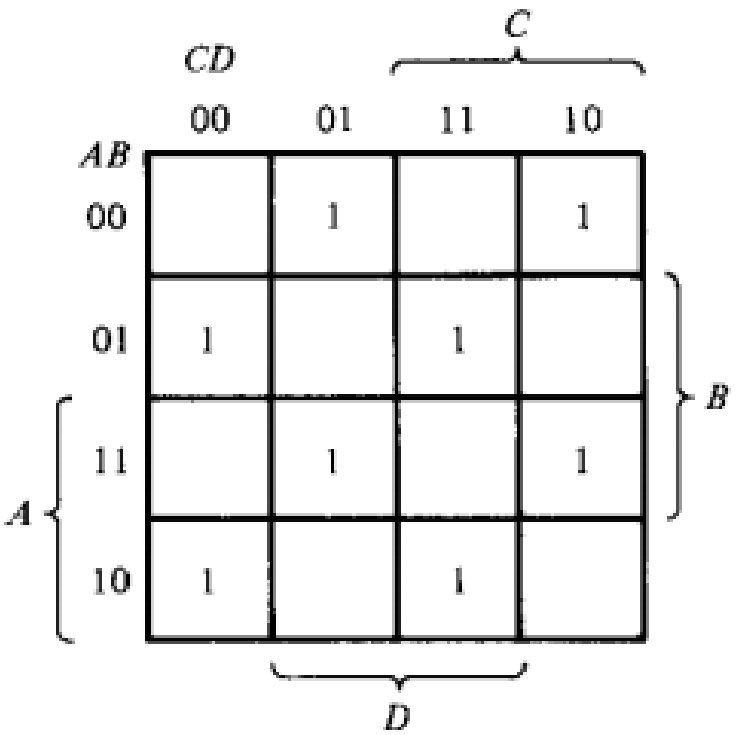
Odd function A⊕B⊕C



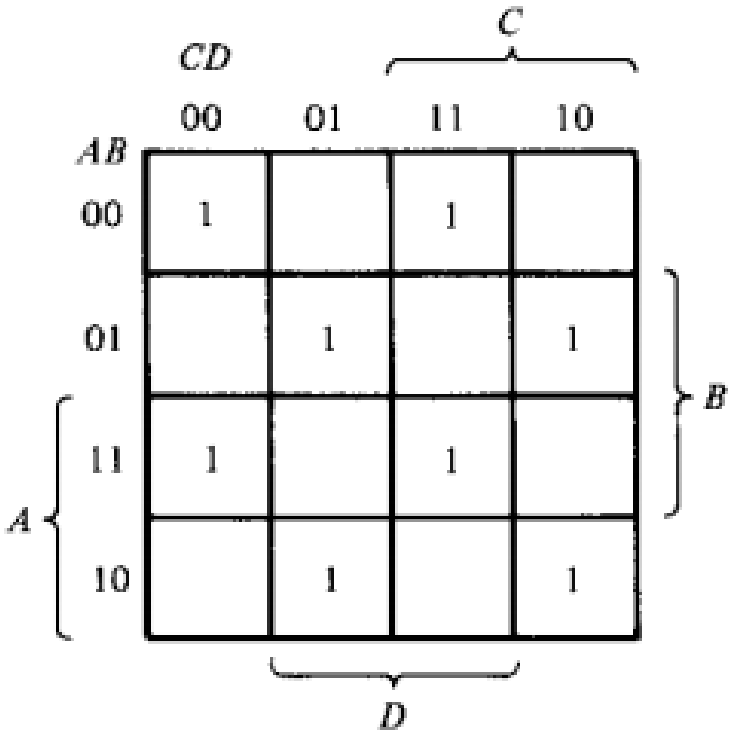
Even function (A⊕B⊕C)'

- For 4 variables

$$A \oplus B \oplus C \oplus D = \sum(1,2,4,7,8,11,13,14)$$



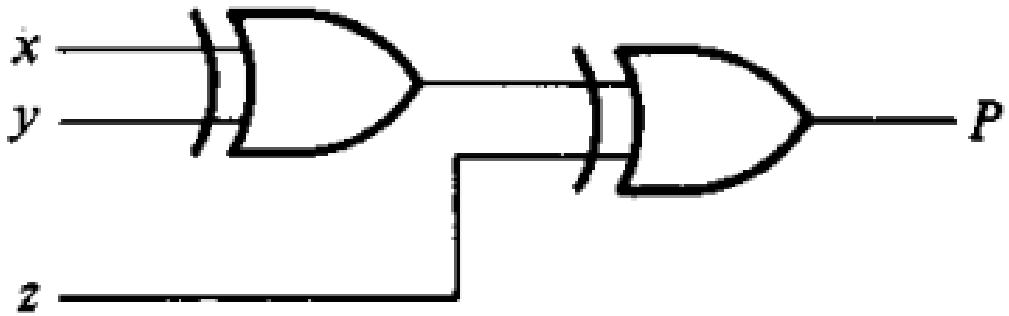
(a) Odd function
$F = A \oplus B \oplus C \oplus D$



(b) Even function
$F = (A \oplus B \oplus C \oplus D)'$

- Even parity generator for 3 bit message

Three-Bit Message			Parity Bit
x	*y*	*z*	*P*
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

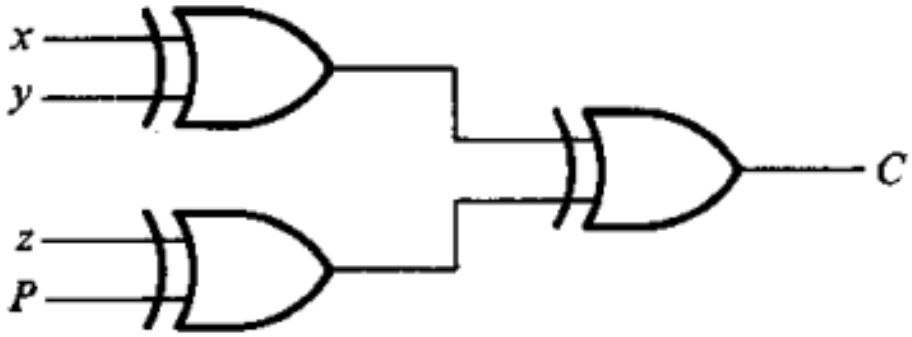


- P will be 1 if message has odd no. of 1's

$$P = x \oplus y \oplus z$$

- Checking parity of the message

Four Bits Received				Parity Error Check
x	y	z	P	C
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

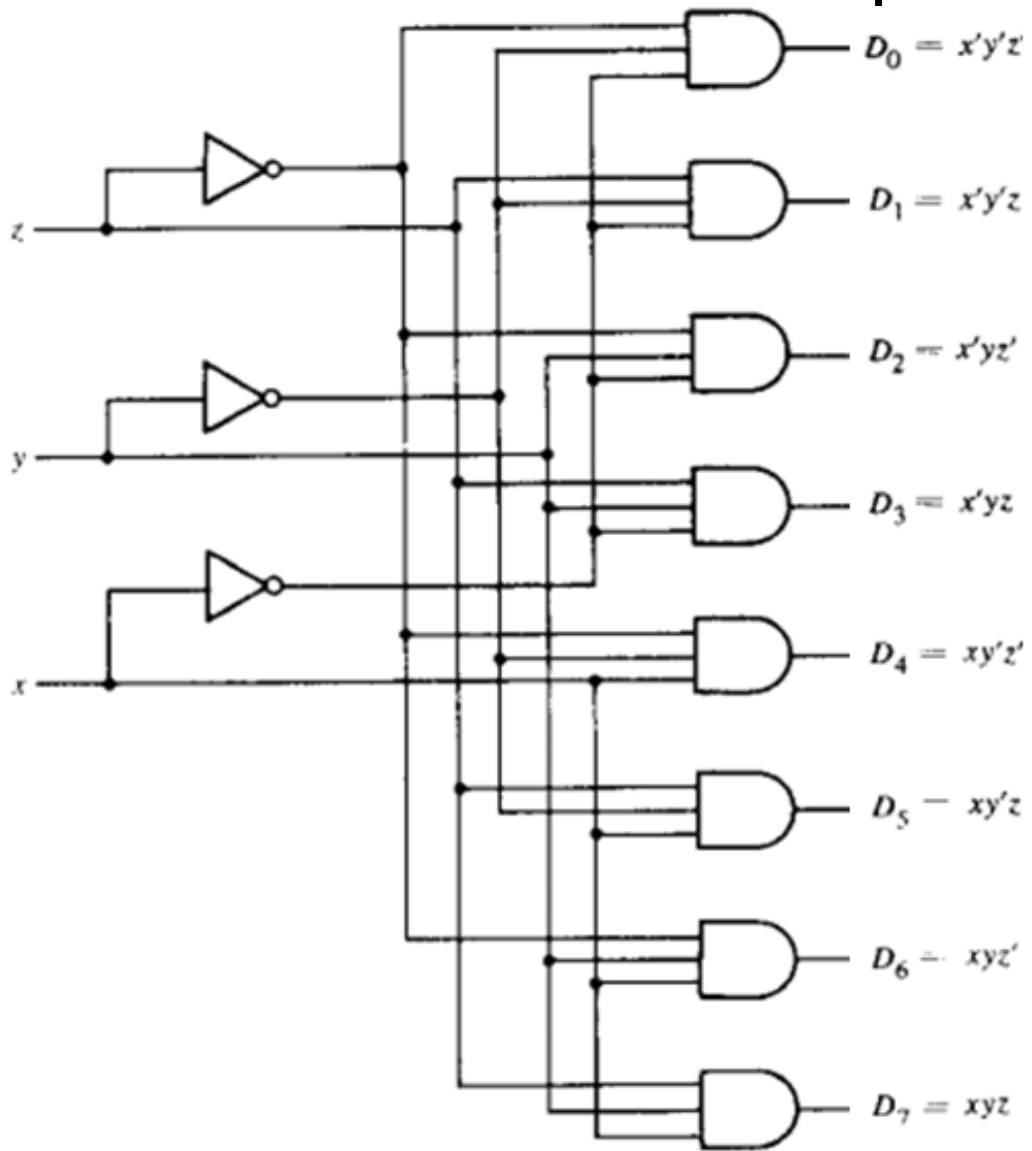


- Message should have even no. of 1's, odd no. of 1's will indicate error

$$C = x \oplus y \oplus z \oplus P$$

Decoder

- Conversion of binary information from n input lines to a maximum of 2^n unique output lines



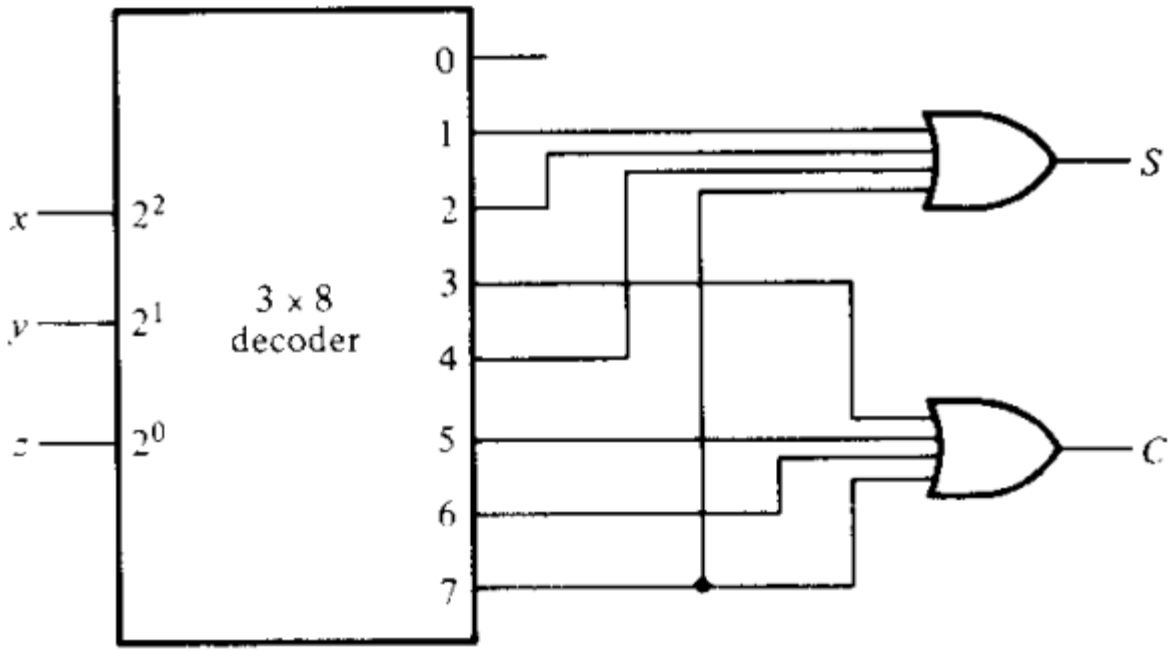
- Generation of 2^n minterms of n input variables
- 3 to 8 or 3×8 decoder

- Truth table

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

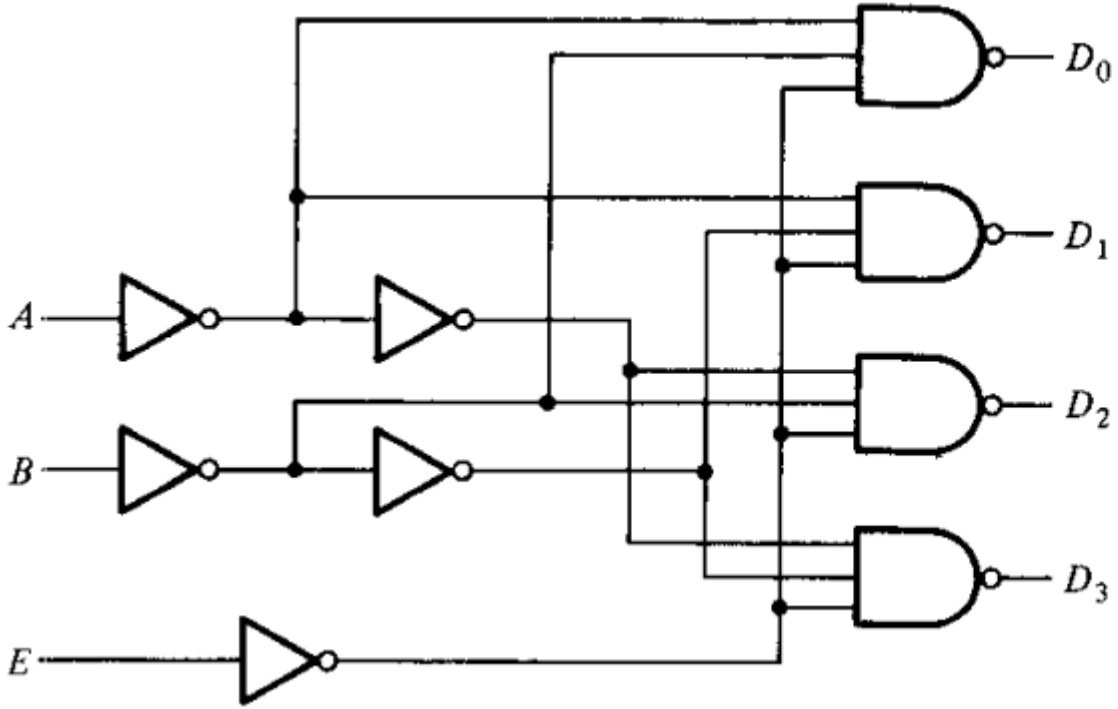
- Outputs mutually exclusive i.e. one output variable equal to 1 at a time, minterm equivalent of the binary number
- Decode 3 bit code to provide 8 outputs, binary to octal conversion

- Implementing combinational circuits
- Full adder with a decoder & two OR gates
- S = $\Sigma(1, 2, 4, 7)$ & C = $\Sigma(3, 5, 6, 7)$



- Instead of F, sum the minterms of F' & use NOR gate to get back F

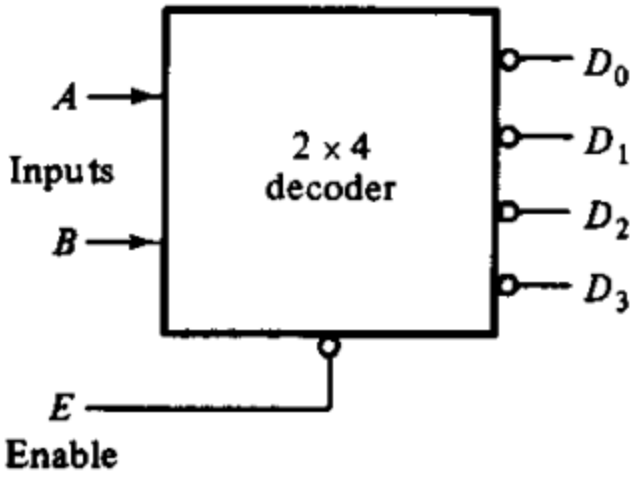
- NAND gates instead of AND gates to generate decoder minterms in their complemented form
- Enable input to control the circuit operation

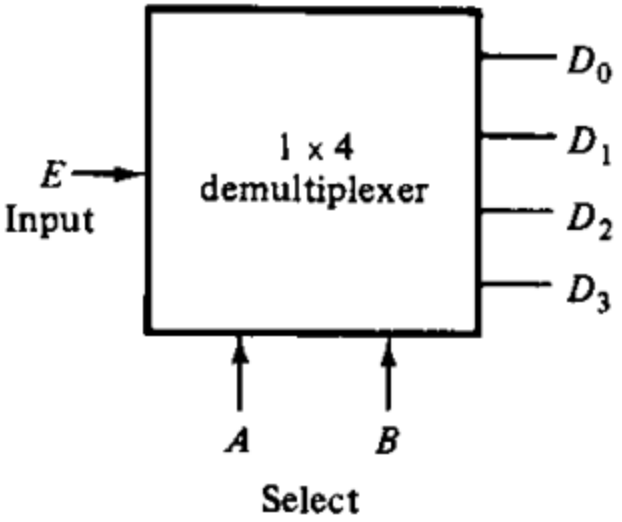


E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

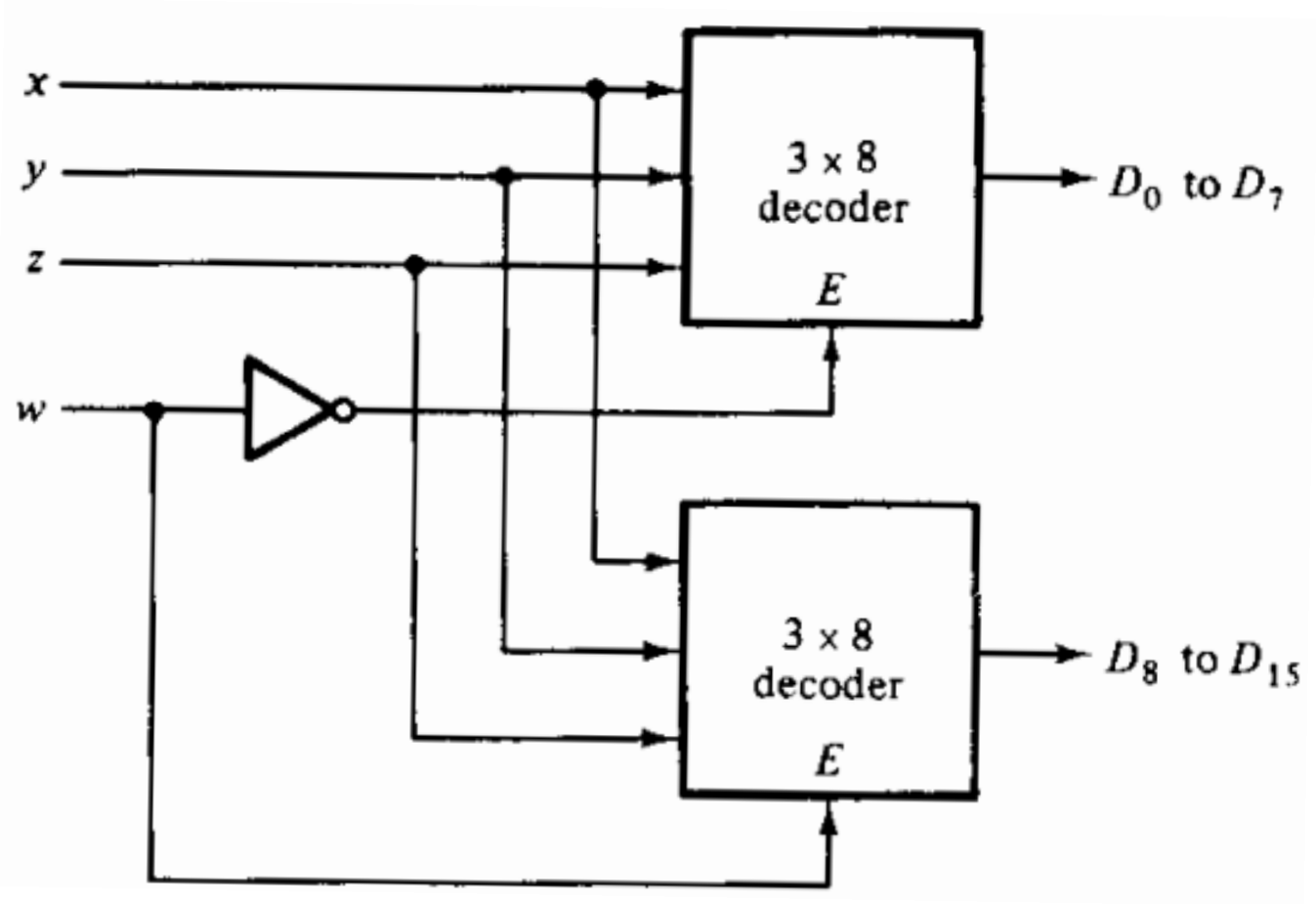
- If E = 1, all outputs will be 0 irrespective of the value of A & B
- E = 0 for normal decoder operation

- Decoder with enable pin can be used as demultiplexer (DeMUX)
- DeMUX receives information on one input lines & transfers it to one of the 2^n possible output lines
- For DeMUX, E taken as input & A, B taken as selection lines





- Construction of higher order decoders using lower order decoders
- 4×16 decoder using 3×8 decoders



Encoder

- Inverse operation of decoder
- 2^n input lines & n output lines

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

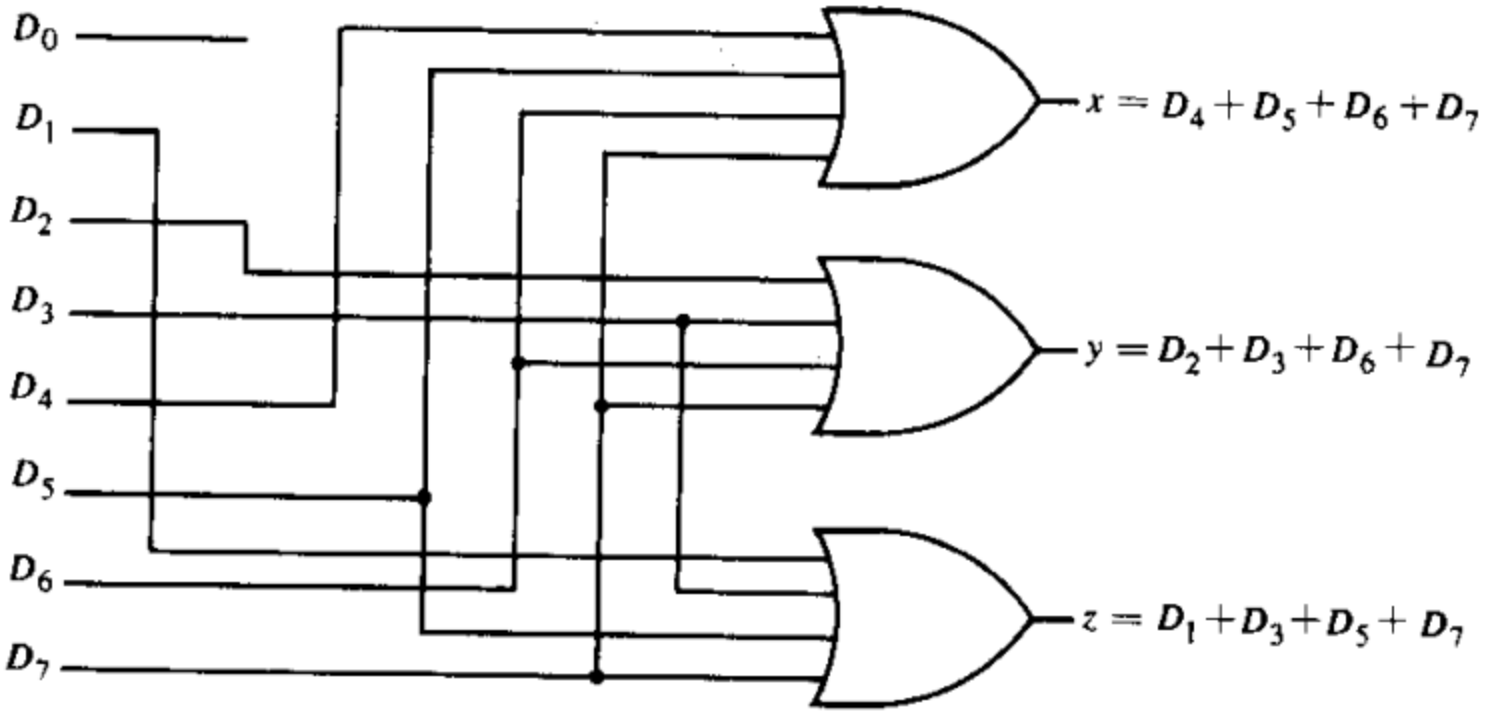
- Implementation with OR gates

$$x = D_4 + D_5 + D_6 + D_7$$

$$y = D_2 + D_3 + D_6 + D_7$$

$$z = D_1 + D_3 + D_5 + D_7$$

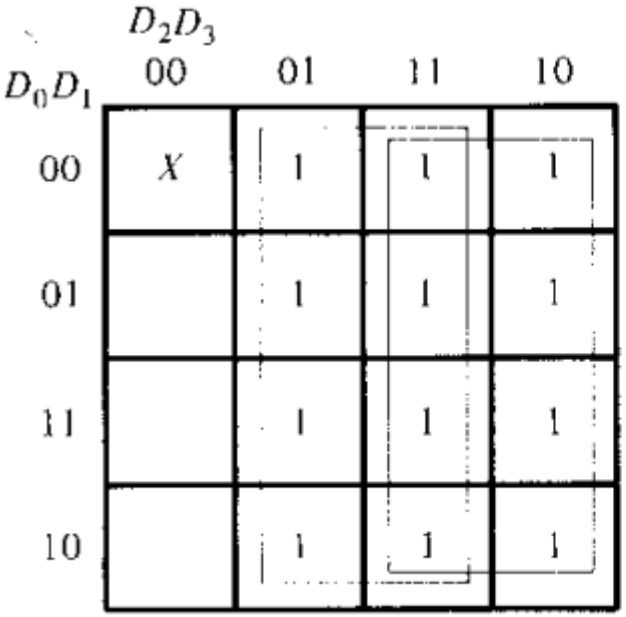
- Encoder construction



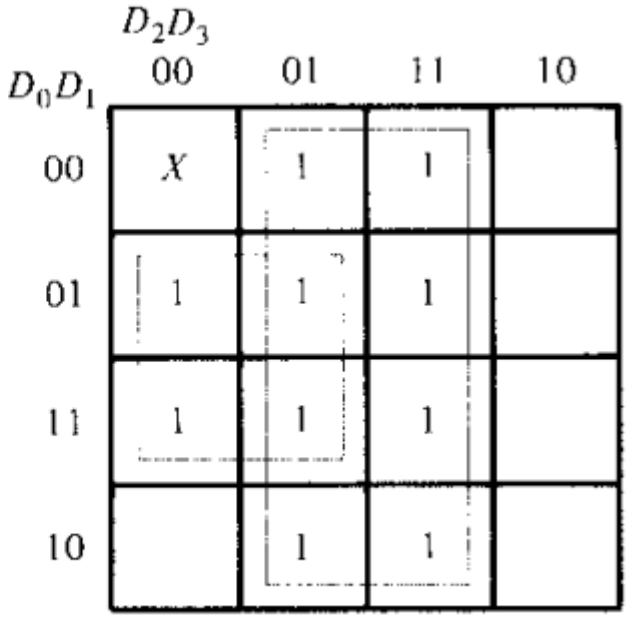
- Only one input can be active at a given time
- Establish priority to avoid ambiguities
- All outputs 0 if D_0=0 or all inputs 0, additional output to specify this condition

- 4 input priority encoder

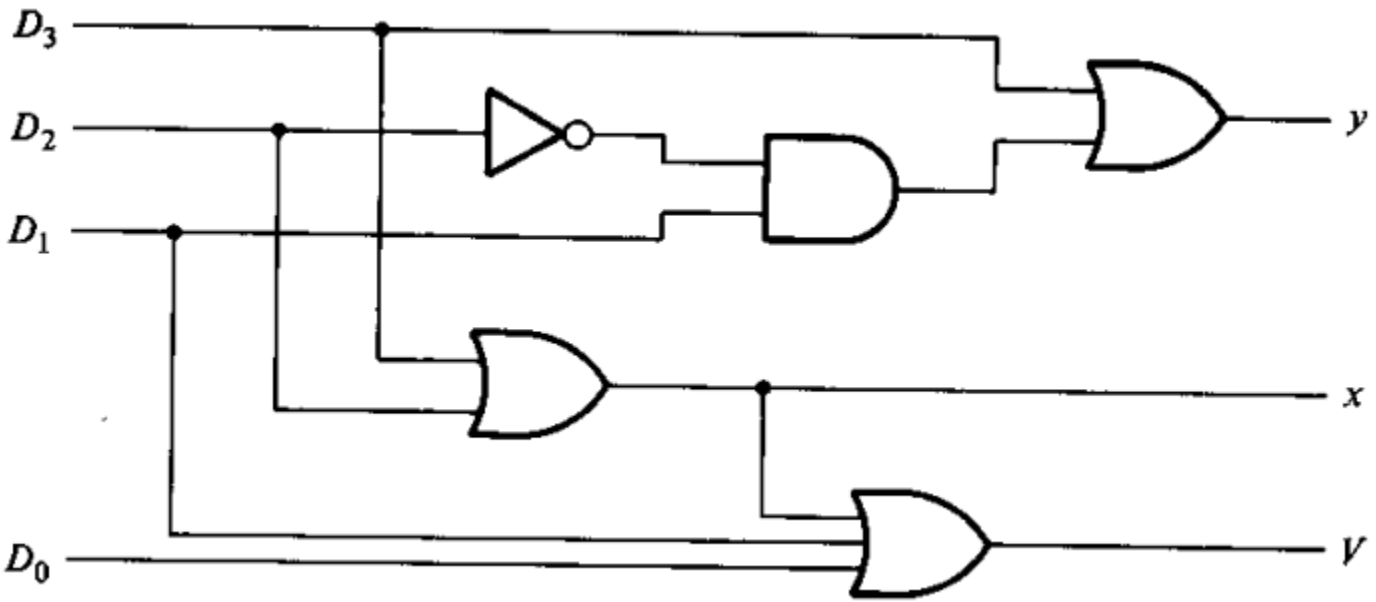
Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1



$x = D_2 + D_3$



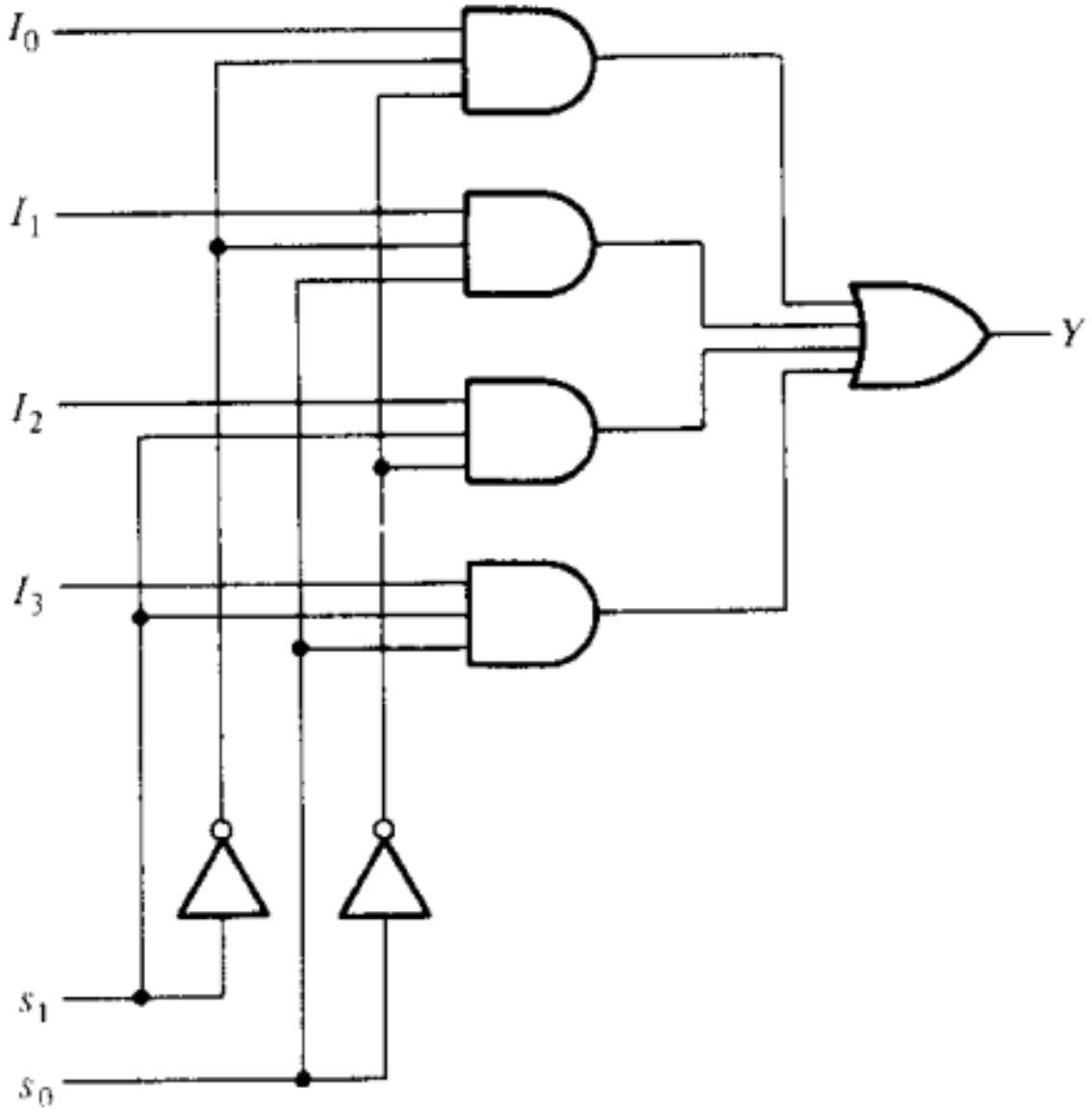
$y = D_3 + D_1 D_2'$



Multiplexers

- Multiplexing means transmitting a large number of information units over a smaller number of channels or lines
- Transfer of binary information from one of many input lines to a single output line
- Selection lines to select a particular input value
- 2^n input lines & n selection lines, $2^n \times 1$ MUX
- Also known as data selector as it selects one of many inputs & steers the binary information to the output

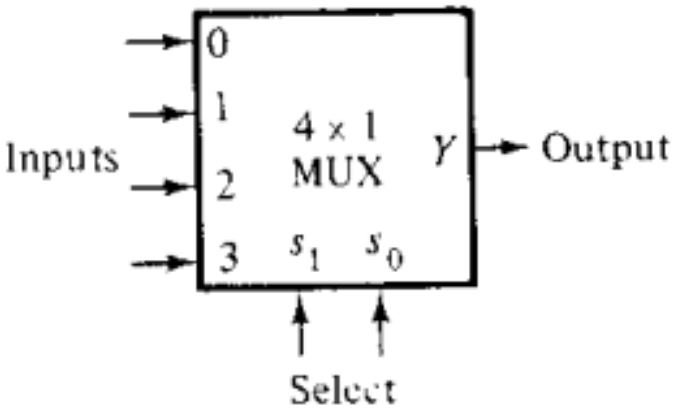
- 4 ×1 MUX



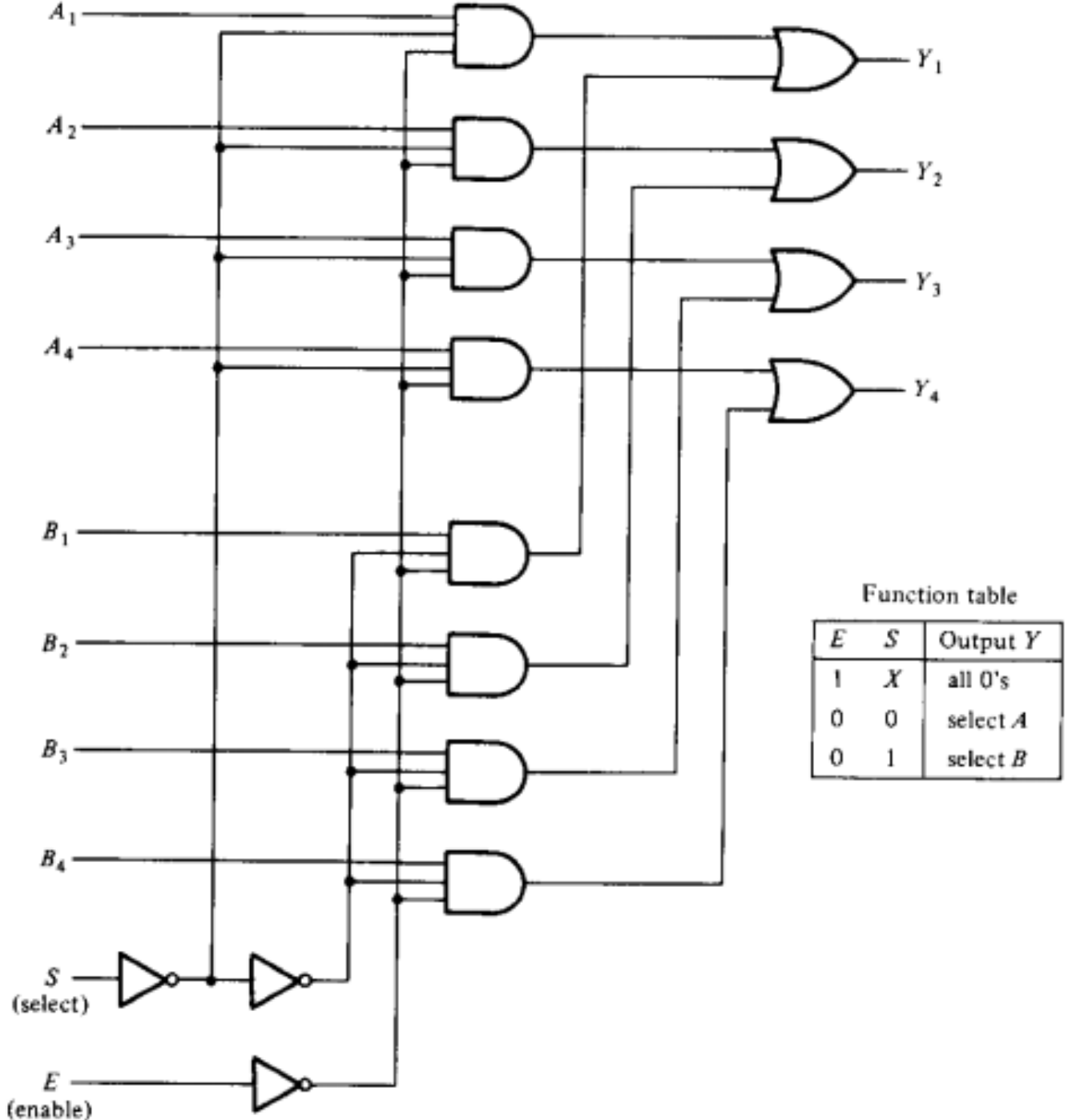
Logic diagram

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

Function table



Block diagram



Function table

E	S	Output Y
1	X	all 0's
0	0	select A
0	1	select B

Quadruple 2 to 1 line MUX & its function table

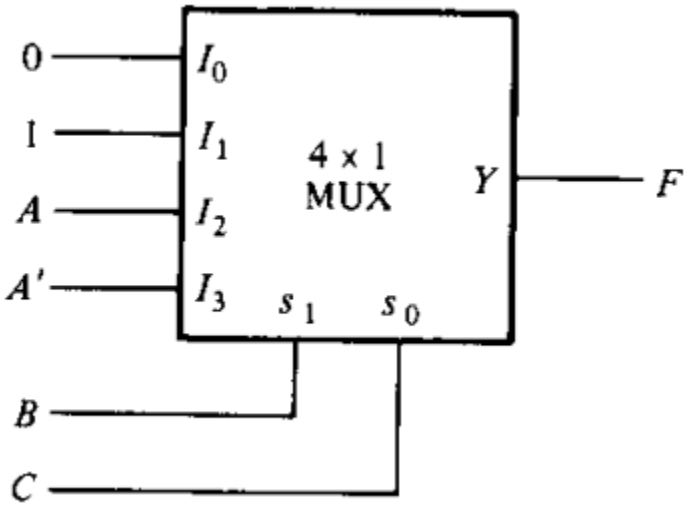
Boolean function implementation

- Kind of a decoder with OR gates
- Possible to implement Boolean functions with the help of MUX
- A Boolean function of n+1 variables can be implemented with a $2^n \times 1$ MUX
- n variables connected to the selection lines
- The remaining 1 variable (x) in appropriate form (x, x', 1, 0) connected to the input lines

- F(A, B, C) = $\Sigma(1, 3, 5, 6)$ with 4×1 MUX
- Variables B & C connected to the selection lines & A needs to be appropriately connected to the input lines

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

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



- Using implementation table:
  - Most significant bit (MSB) to input lines & remaining variables to selection lines
  - Order is (A, B, C, D, ...), A is MSB, connected to input lines, B to higher order selection line, C to next lower selection line & so on
  - List the MUX inputs & under them all the minterms in two rows, first row with minterms having A in complemented form & second row with minterms having A in direct form
  - Circle the minterms of the function & inspect each column. If no minterm selected, apply 0 to the corresponding input; if both minterms selected, apply 1; if only top is circled, apply A'; if only bottom is circled apply A

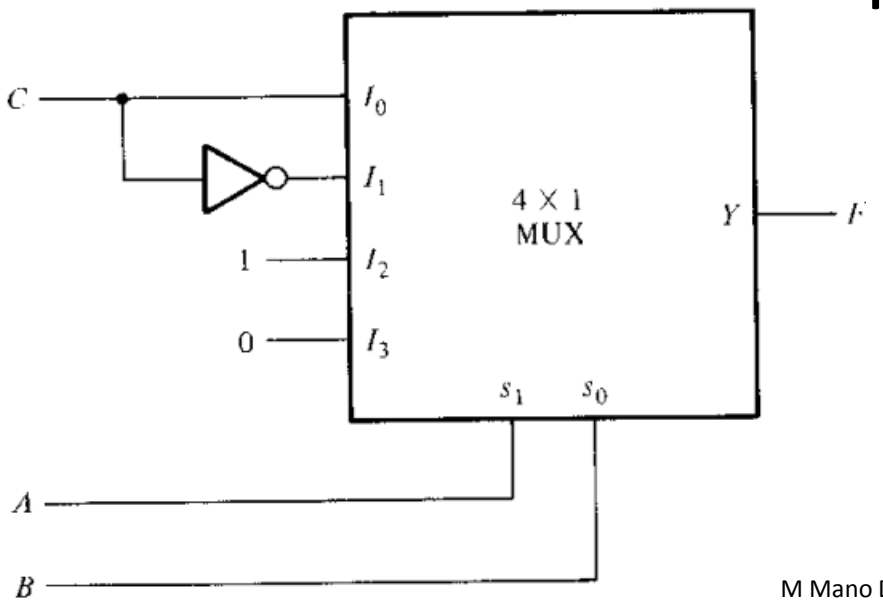
- F(A, B, C) = $\Sigma(1, 3, 5, 6)$ & F(A, B, C) = $\Sigma(1, 2, 4, 5)$

Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

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

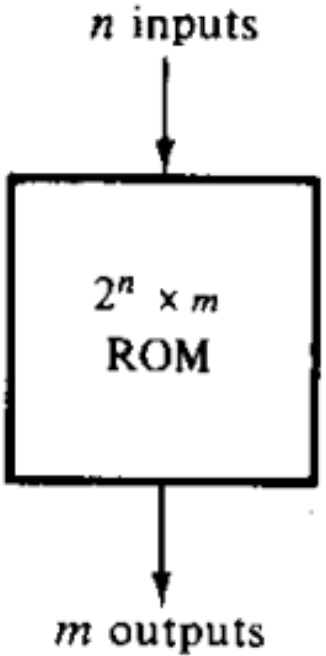
- Any other variable can also be chosen for the inputs

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



Read Only Memory (ROM)

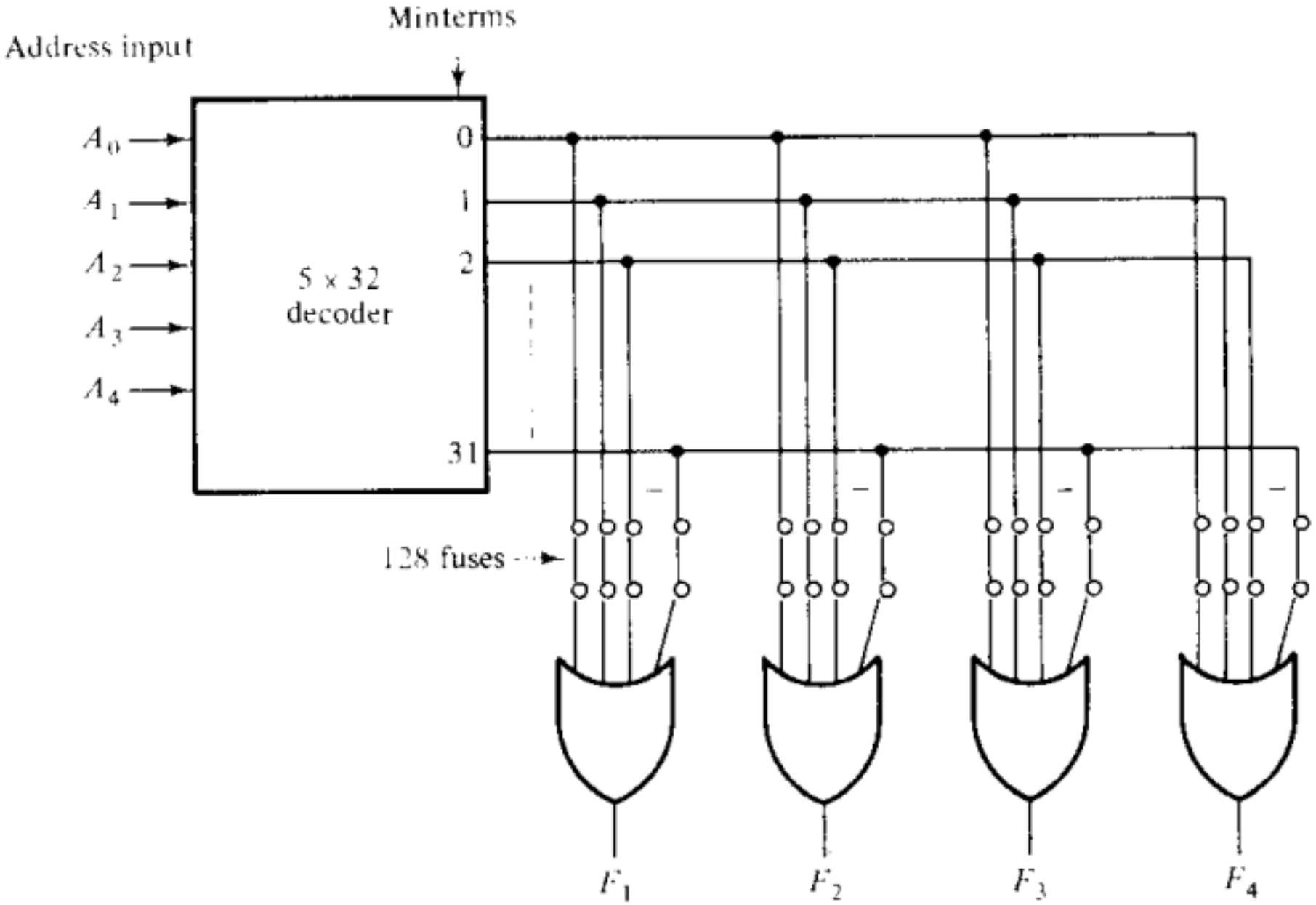
- Decoder generates minterms & OR gates can sum them for implementing a Boolean function
- ROM includes both the decoder & OR gates in a single IC package
- Can be used as a memory for storing binary information or can be used to implement Boolean functions



ROM block diagram

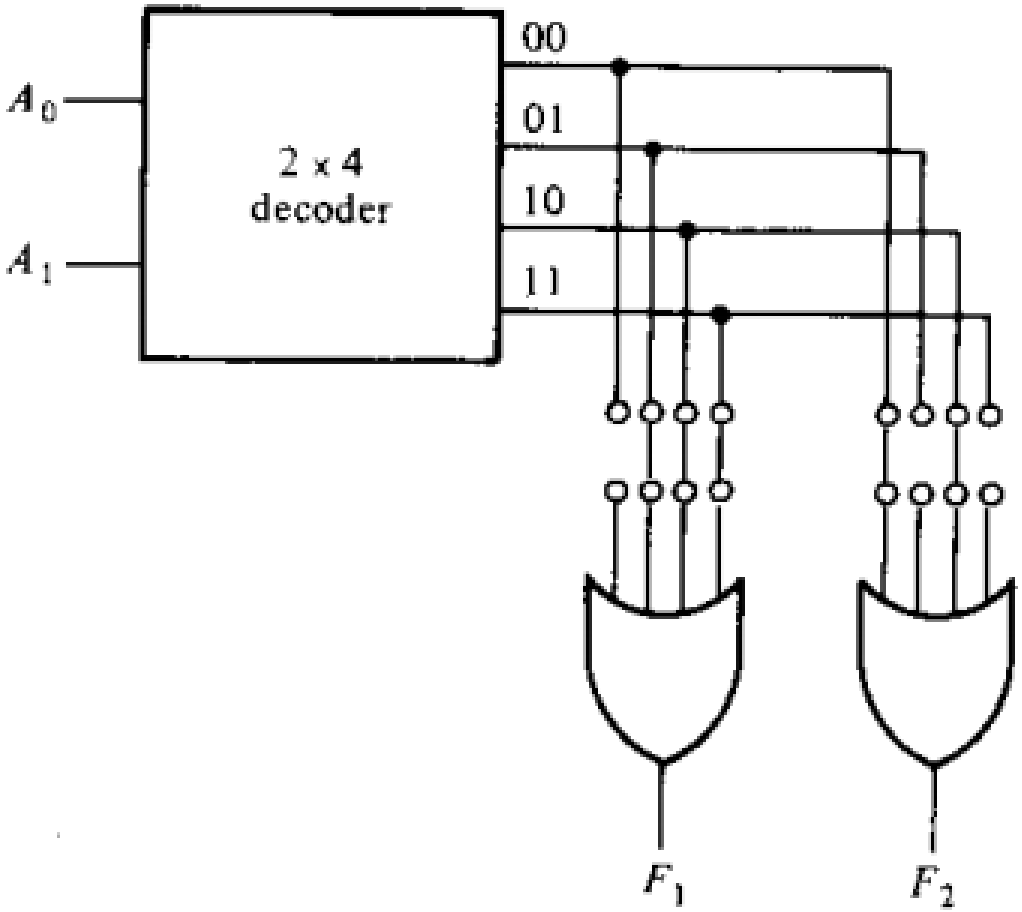
- n inputs, m outputs, size $2^n \times m$
- Bit combination of the input variables ‘address’
- Bit combination of the output lines ‘word’
- 32×4 ROM, 5 inputs & 4 outputs, 32 words each of 4 bits stored inside
- Selection of the word done by input lines
- Input 00000, word 0 will be selected; input 11111, word 31 will be selected & will appear at the output
- Decoder outputs connected by special fuses to the OR gate inputs
- Fuses are like switches & can be programmed
- Information stored in the condition of the fuse

- Internal construction of a 32×4 ROM



- Combinational logic with ROM, n input & m output circuit with $2^n \times m$ ROM

A_1	A_0	F_1	F_2
0	0	0	1
0	1	1	0
1	0	1	1
1	1	1	0

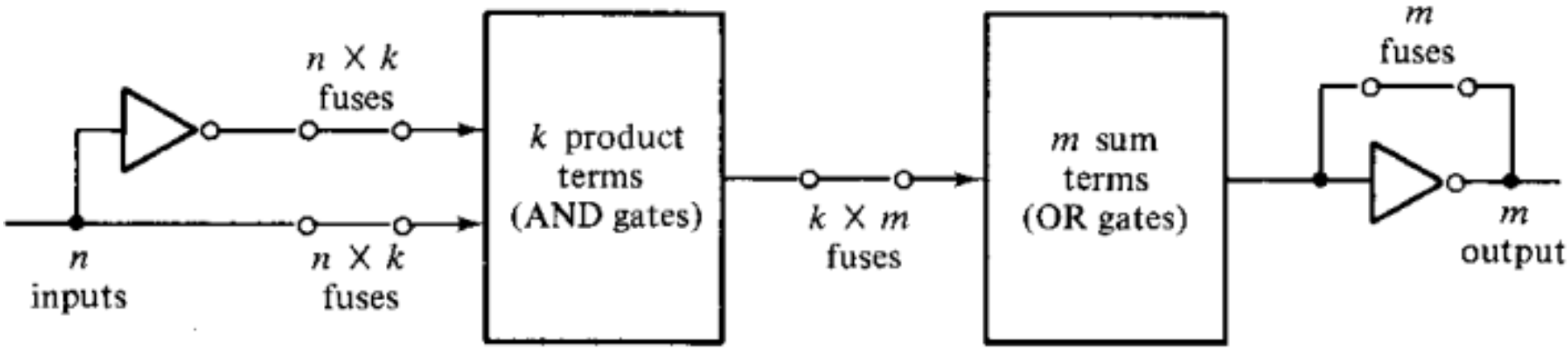


Types of ROM

- The programming in ROM can be done by the manufacturer (mask programming) or the user
- PROM
  - Programmable Read Only Memory
  - Fuses blown by application of current pulses
  - Commercially available programmers
- EPROM
  - Erasable Programmable Read Only Memory
  - Exposure to UV light for erasing & then reprogram
- EEPROM
  - Electrically Erasable Programmable Read Only Memory
  - Electrical signals instead of UV light for erasing & then reprogram

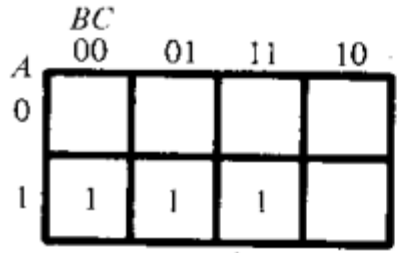
Programmable Logic Array (PLA)

- Similar to ROM in construction
- Instead of decoder, array of AND gates to have only the required minterms
- Combinational circuit with excessive don't care conditions
- n inputs, m outputs, k product terms & m sum terms

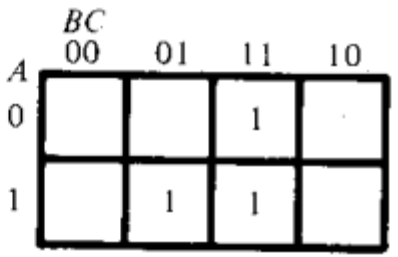


- Necessary to simplify the function to reduce the no. of gates required

A	B	C	F_1	F_2
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	0	0
1	1	1	1	1



$F_1 = AB' + AC$



$F_2 = AC + BC$

	Product term	Inputs A	B	C	Outputs F_1	F_2	
AB'	1	1	0	--	1	—	
AC	2	1	—	1	1	1	
BC	3	—	1	1	—	1	
					T	T	T/C