"The First Industrial Revolution used water and steam power to mechanize production. The Second used electric power to create mass production. The Third used electronics and information technology to automate production."

- Klaus Schwab

1

Combinational Circuits

1.1 Introduction

A combinational circuit consists of logic gates whose outputs at any time are determined directly from the present combination without regard to previous inputs. It consists of input variables, logic gates, and output variables. The logic gates signals from the inputs and generate signals to the outputs. The n input binary variables come from an external source; the m output variables go to an external destination. For n input variables, there 2^n possible combinations of binary input variables. For each possible input combination, there is one and only one possible output combination. A combinational circuit can be described by m boolean functions, one for each output variable. Each output function is expressed in terms of the n input variables.

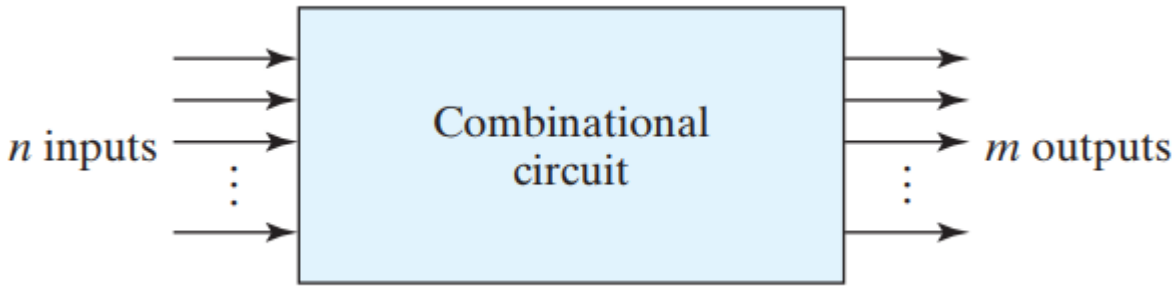


Figure 1.1: Block digram of combinational circuit

1.2 Design Procedure

1. The problem is stated.
2. The number of available input variable and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationships between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

1.3 Half Adder

A combinational circuit that performs the addition of two bits is called a half-adder. It consists of two binary input and two binary outputs. The input variable designate the augend and addend bits; the output variables produce the sum and the carry. We arbitrarily assign symbols X and Y to the two inputs an *S(for sum)* and *C(for carry* to the outputs.

The simplified Boolean functions for the two outputs can be directly obtained from the truth table as:

$$S = \bar{x}y + x\bar{y}$$

$$= x \oplus y$$

$$C = xy$$

Table 1.1: Half-adder truth table

Inputs		Outputs	
X	Y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The half-adder is implemented with exclusive-OR and AND gate as shown in figure 1.2

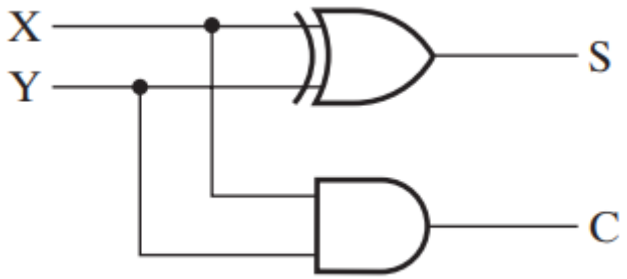


Figure 1.2: Half-adder logical digram

1.4 Full Adder

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of three inputs and two outputs. Two of the inputs variables, denoted by *x* and *y* represent the two significant bits to be added. The third input, *z* represents the carry from the previous lower significant position. The two output variables are designated by symbol *S* for sum and *C* for carry.

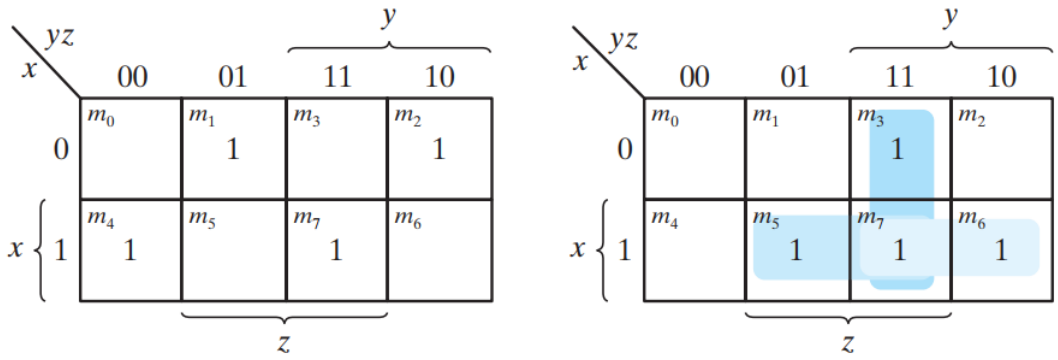
Implementing full-adder using two half-adder and one OR gate

1.5 Binary Parallel Adder

- A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output

Table 1.2: Full adder truth table

Inputs			Outputs	
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



(a) $S = x'y'z + x'yz' + xy'z' + xyz$ (b) $C = xy + xz + yz$

Figure 1.3: Kmaps for full adder and simplified Boolean expression for its outputs

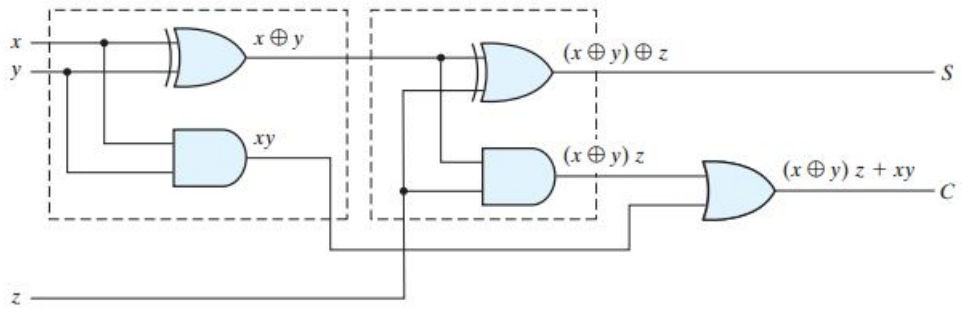


Figure 1.4: Implementation of full adder with two half adder and an OR gate.

carry from each full adder connected to the input carry of the next full adder in the chain.

- Addition of *n*-bit numbers requires a chain of *n* full adders or a chain of one-half adder

and *n-1* full adder. In the former case, the input carry to the least significant position is fixed at 0.

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}

Figure 1.5: Addition of two 4-bit numbers example.

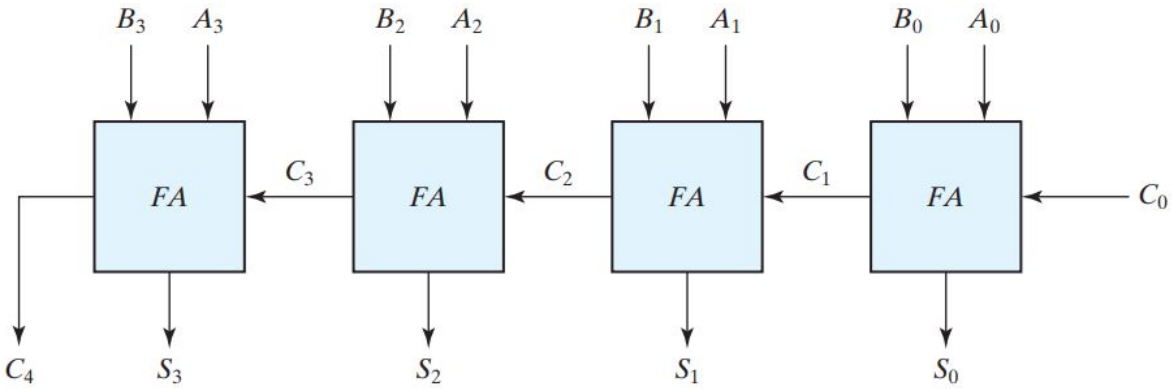


Figure 1.6: A 4-bit binary parallel adder.

- Figure 1.6 shows the interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder. The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit.

- The carries are connected in a chain through the full adders. The input carry to the adder is C_0, and it ripples through the full adders to the output carry C_4. The S outputs generate the required sum bits.

- The signal from the input carry, C_i to the output carry, C_{i+1}, propagate through and AND gate and OR gate, which constitute two gate level. If there are four full-adders in the parallel adder, the output carry C_4 would have $2 \times 4 = 8$ gate levels.

- An n-bit adder requires n full adders,with each output carry connected to the input carry of the next higher order full adder. For an n-bit parallel adder, there are $2n$ gate levels for the carry to propagate through.

1.6 Carry Propagation

- In binary parallel adder, each bit of the sum output depends on the value of the input carry, the value of S_i at any given stage in the adder will be in its steady-state final value only after the input carry to that stage has been propagated. In this regard, consider output S_3 in figure . Inputs A_3 and B_3 are available as soon as input signals are applied to the adder. However, input carry C_3 does not settle to its final value until C_2 is available from the previous stage. Similarly, C_2 has to wait for C_1 and so on down to C_0. Thus, only after the carry propagates and ripples through all stages will the last output S_3 and carry C_4 settle to their final correct value.

- In any combinational circuit, the signal must propagate through the gates before the correct output sum is available in the output terminals. The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit.

1.7 Carry-Lookahead Adder

- It is a fast adder, the speeds up the addition of two n-bit numbers, that overcomes the limitation of binary parallel adder.

- To reduce the delay caused by the effect of carry propagation through the ripple-carry adder, we can attempt to evaluate quickly for each stage whether the carry-in from the previous stage will have a value 0 or 1. If a correct evaluation can be made in a relatively short time, then the performance of the complete adder will be improved.

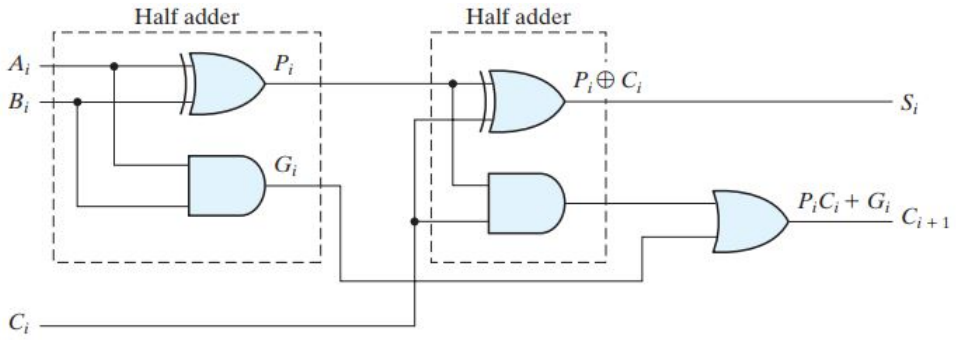


Figure 1.7: Full adder circuit with P and G.

- Full adder circuit is redrawn as shown in figure 1.7 with new binary variables intro-

duced P and G such that

$$P_i = A_i \oplus B_i$$
$$G_i = A_i B_i$$

The output sum and carry can respectively expresses as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_i C_i$$

- G_i is called a carry generate, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i.
P_i is called a carry propagate, because it determines whether a carry into stage i will propagate into stage $i+1$.

- Now we write the Boolean expression for the carry outputs for each stage and substitute the value for each C_i from the previous equations:
For stage 0 (i=0)

$$C_0 = input_carry$$
$$P_0 = A_0 \oplus B_0$$
$$G_i = A_i B_i$$
$$S_0 = P_0 \oplus C_0$$

For stage 1 (i=1)

$$C_1 = G_0 + P_0 C_0$$
$$P_1 = A_1 \oplus B_1$$
$$S_1 = P_1 \oplus C_1$$
$$G_1 = A_1 B_i$$

For stage 2 (i=2)

$$\begin{aligned} C_2 &= G_1 + P_1C_1 \\ &= G_1 + P_1(G_0 + P_0C_0) \\ &= G_1 + P_1G_0 + P1P_0C_0 \\ P_2 &= A_2 \oplus B_2 \\ S_2 &= P_2 \oplus C_2 \\ G_2 &= A_2B_2 \end{aligned}$$

For stage 3 (i=3)

$$\begin{aligned} C_3 &= G_2 + P_2C_2 \\ &= G_2 + P_2(G_1 + P_1G_0 + P1P_0C_0) \\ &= G_1 + P_2G_1 + P_2P_1G_0 + P_2P1P_0C_0 \\ P_3 &= A_3 \oplus B_3 \\ S_3 &= P_3 \oplus C_3 \\ G_3 &= A_3B_3 \end{aligned}$$

In general, carry propagation can be written as:

$$C_{i+1} = G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + ...P_iP_{i-1}...p_2P_1G_0 + P_iP_{i-1}...P_1P_0C_0$$

- In figure 1.9 all output carries are generate after a delay through two levels of gates. Thus, outputs S_1 through S_3 have equal propagation delay times. The two-level circuit for the output carry C_4 is not shown. This circuit can easily be derived by the equation-substitution method.

1.8 Decimal Adder

- Computers or calculators that perform arithmetic operations directly in the decimal number system represent decimal numbers in binary-coded form.

- An adder for such a computer must employ arithmetic circuits that accept coded decimal numbers and present results in the accepted code.

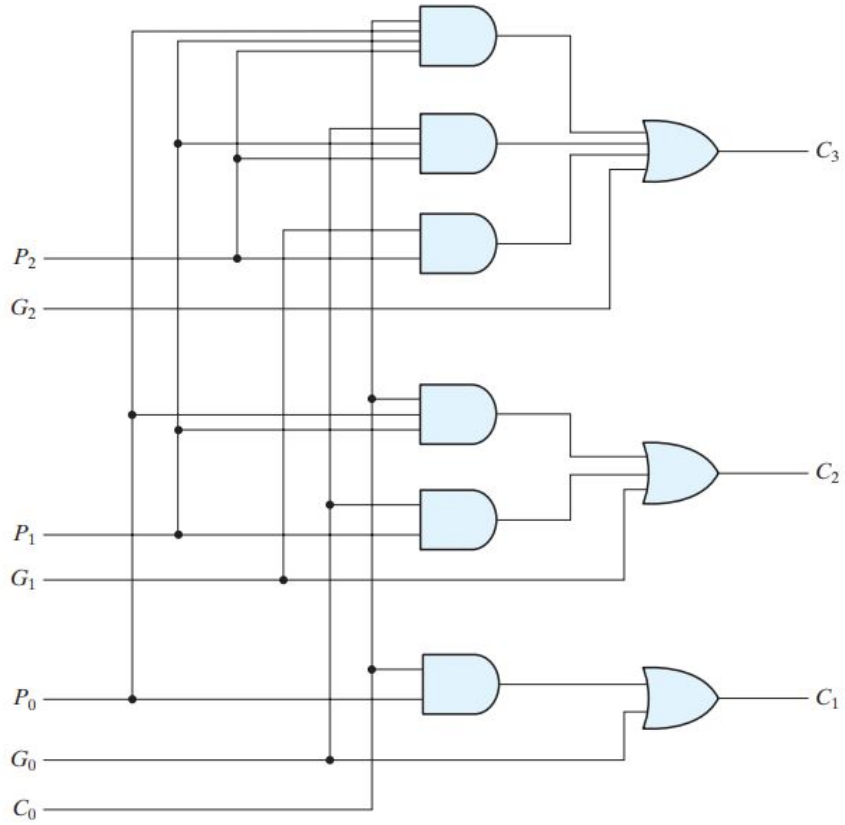


Figure 1.8: Carry-look ahead generator.

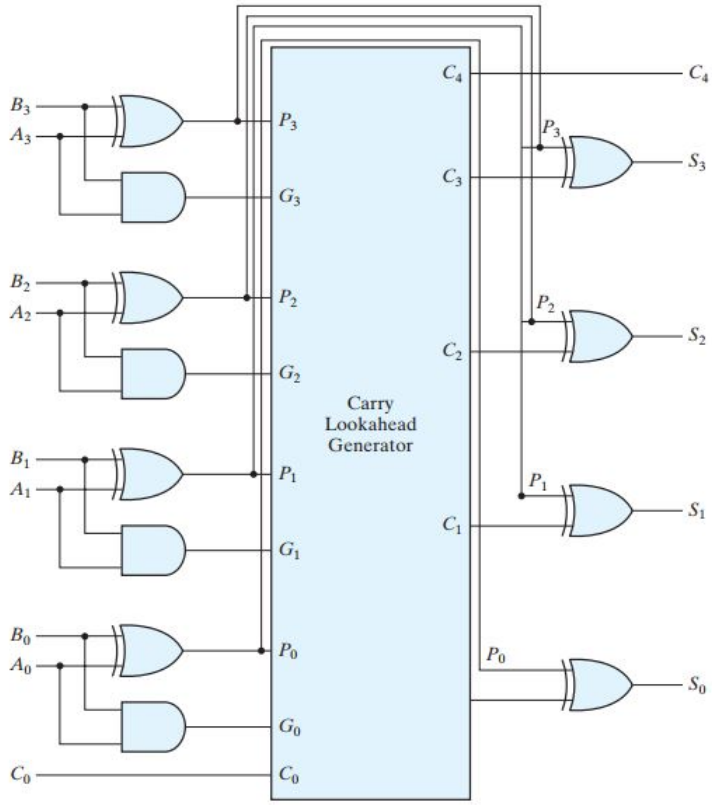


Figure 1.9: Four bit adder with carry lookahead.

- A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input carry and output carry.

- Example: BCD adder

1.9 BCD Adder

- BCD adder is a combinational circuit that adds up two decimal numbers when they are encoded with binary-coded decimal (BCD) form.

- Adding two decimal digits to a 4-bit binary adder, the output sum cannot be greater than 9+9+1=19.

- Applying two BCD digits to a 4-bit binary adder, the adder will form the sum in binary ranging from 0 to 19, shown in figure 1.10. These binary numbers are listed in Table below and are labelled by symbols K, Z_8, Z_4, Z_2, Z_1. K is the carry, and the subscripts under the letter Z represents the weights 8,4,2, and 1 that can be assigned to the four bits in the BCD code.

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

Figure 1.10: BCD adder deviation.

Analysis looking at table

When (binary sum) $\leq$ 1001

• Corresponding BCD number is identical, and therefore no conversion is needed.

When (binary sum) > 1001

• Non-valid BCD representation is obtained.

• The addition of binary 6 (0110) to the binary sum converts to the correct BCD representation and also produces and output carry as required.

Correction is Needed when:

• The binary sum has an output carry $K = 1$.

• The other six combinations form 1010 to 1111 that have $Z_8 = 1$. To distinguish them from binary 1000 and 1001, which also have a 1 in position Z_8,we specify that either Z_4 or Z_2 must have a 1. The condition for a correction and an output carry can be expressed by the Boolean function.

$$C = K + Z_8Z_4 + Z_8Z_2$$

• When C=1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.

1.10 Half Subtractor

- A half subtractor is a combinational circuit that subtracts two bits and produce their difference. It also has an output to specify if a 1 has been borrowed.

- It has two inputs and two outputs, one for difference and other for borrow.

- Let us designate the minuend bit by x and subtrahend bit by y, and for difference be D and for borrow be B.

Boolean functions for two outputs

$$D = \bar{x}y + x\bar{y}$$

$$= A \oplus B$$

$$B = \bar{x}y$$

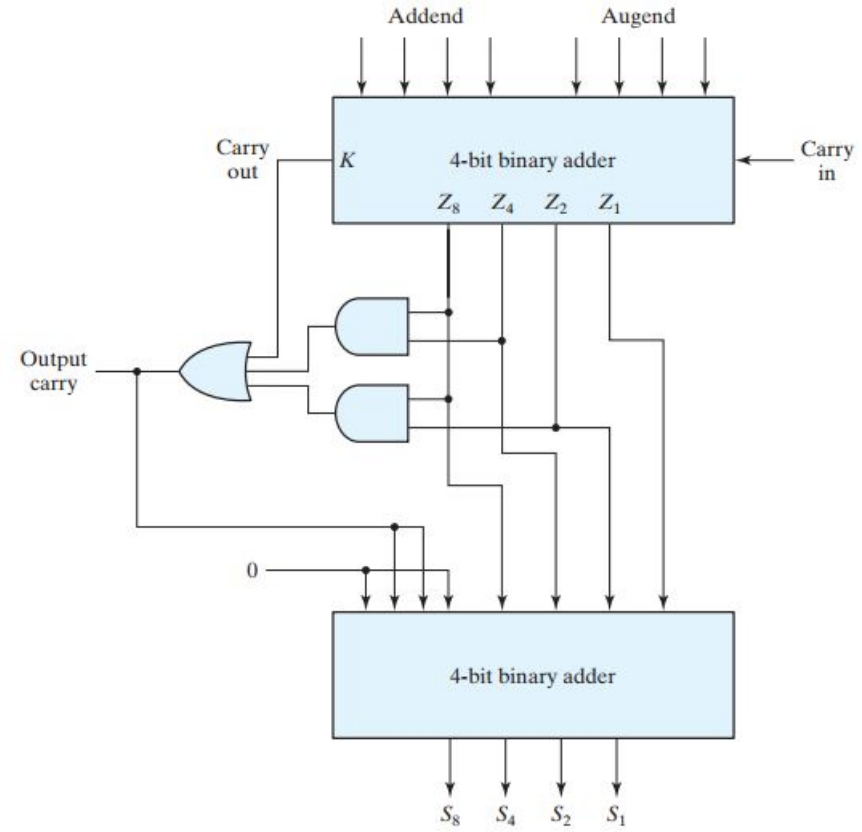


Figure 1.11: Block diagram of BCD adder.

Table 1.3: Half-subtractor truth table

Inputs		Outputs	
x	y	B	D
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

1.11 Full Subtractor

- A full-subtractor is a combinational circuit that performs a subtraction between two bits, taking into account that a 1 may have been borrowed by a lower significant stage.

- It has three inputs and two outputs.

- The three inputs, *x*, *y*, and *z*, denote the minuend, subtrahend, and previous borrow, respectively. The two outputs, *D* and *B*, represent the difference and output borrow, respectively.

Table 1.4: Full subtractor truth table

Inputs			Outputs	
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

The simplified sum of products output functions derived from k-map are:

$$D = \bar{x}\bar{y}z + \bar{x}y\bar{z} + x\bar{y}\bar{z} + xyz$$

$$B = \bar{x}y + \bar{x}z + yz$$

1.12 Controlled Inverter

- Figure 1.12 shows a controlled inverter. When *INVERT* is low, it transmit the 8-bit input to the output; when *INVERT* is high, it transmit the 1's complement.

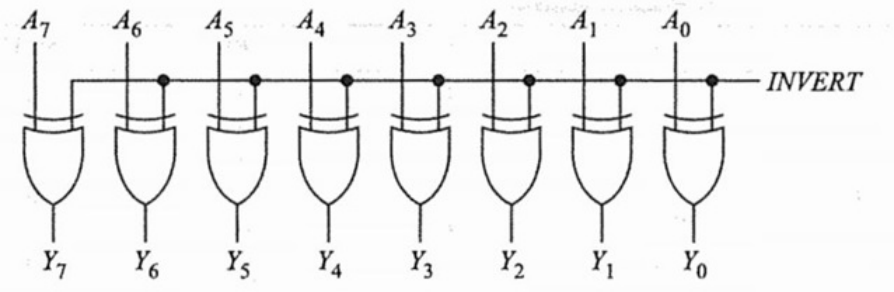


Figure 1.12: A 8-bits controlled inverter.

For example: if the input is $A_7...A_0$ = 01101110, then

- A low logic on INVERT produces $Y = A$ i.e. $Y_7...Y_0$ = 01101110

- A High logic on INVERT produces $Y = \bar{A}$ i.e. $Y_7...Y_0$ = 10010001.

1.13 Parallel Binary Addder-Subtractor

- The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder.

- A four-bit adder–subtractor circuit is shown figure 1.13.

- The mode input M controls the operation.

- When $M = 0$, the circuit is an adder,
  - ◇ Performs $A + B$
- When $M = 1$, the circuit becomes a subtractor.
  - ◇ Performs $A + \bar{B} + 1$ i.e A plus 2's complement of B(=A-B).

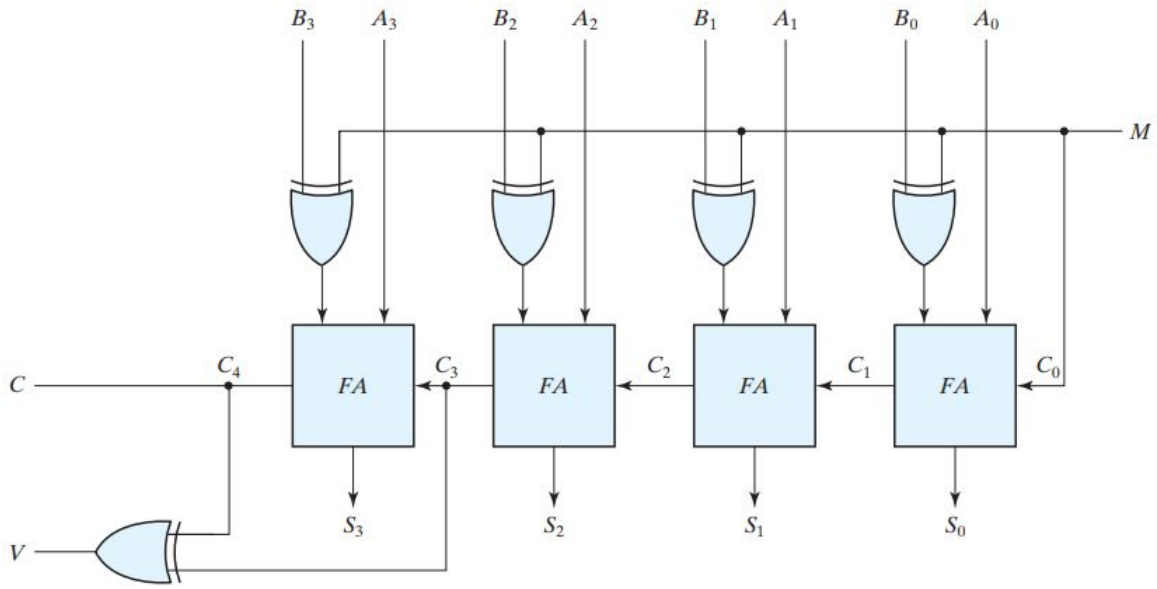


Figure 1.13: A 4-bit parallel binary adder-subtractor.

1.13.0.1 Overflow

- When two numbers with n digits each are added and the sum is a number occupying $n + 14$ digits, we say that an *overflow* occurred. This is true for binary or decimal numbers, signed or unsigned.

- Overflow is a problem in digital computers because the number of bits that hold the number is finite and a result that contains n + 1 bits cannot be accommodated by an n-bit word.

- With addition of two unsigned numbers, overflow is detected whenever is the end carry $C = 1$.

- With addition of two signed numbers, overflow is detected whenever is the end carry $V = 1$. V equals to exclusive-OR of the carry into the sign bit position and the carry out of the sign bit position.

- It is to be noted that, overflow can occur *only if* two both numbers to be added are positive or both are negative; if they have different sign, overflow does not occur.

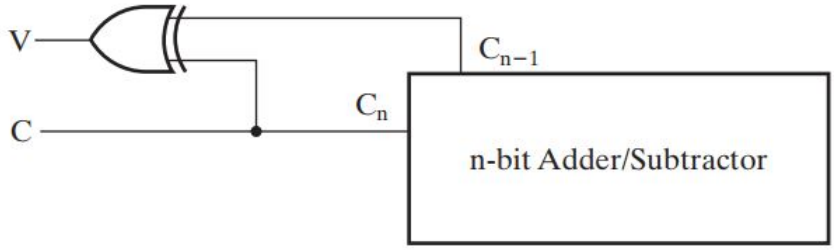


Figure 1.14: Overflow detection logic for addition and subtraction.

1.14 Decoder

- A binary code of n bits is capable of representing up to 2^n distinct elements of the coded information.

- A *decoderis* a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n-bit decoded information has unused or don't care combinations, the decoder output will have less than 2^n outputs.

- A *decoder* detects the presence of a specified combination of bits (code) on its inputs and indicates the presence of that code by a specified output level.

- The decoder presented here are called n-to-m line decoders where $m \leq 2^n$. Their purpose is to generate the 2^n (or less) minterms of n input variables. The name decoder is also used in conjunction with some code converters such as a BCD-to-seven-segment decoder.

1.14.0.1 The Basic Binary Decoder

Suppose you need to determine when a binary 1001 occurs on the inputs of a digital circuit. An AND gate can be used as the basic decoding element because it produces a HIGH output only when all of its inputs are HIGH. Therefore, you must make sure that all of the inputs to the AND gate are HIGH when the binary number 1001 occurs; this can be done by inverting the two middle bits (the 0s), as shown in figure 1.15.

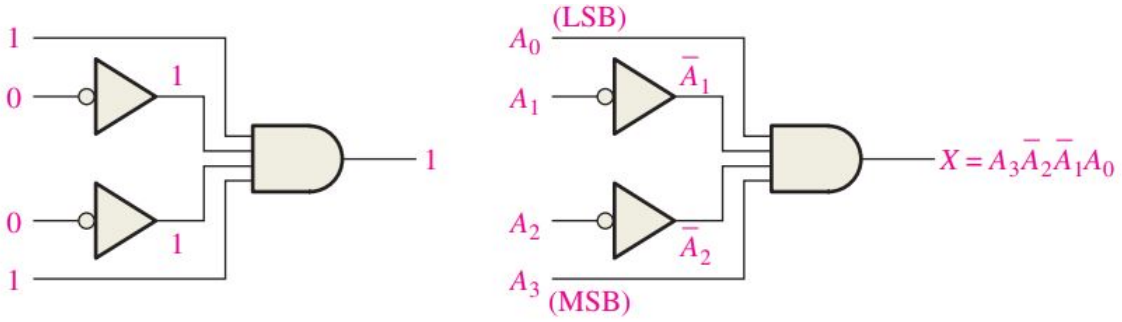


Figure 1.15: Decoding logic for the binary code 1001 with an active-HIGH output.

1.14.0.2 2-to-4 Line Decoder (Active High Design Without Enable Input)

- Let the two inputs be A and B, output lines be D_0, D_1, D_3 and D_4 and E be enable input.

Inputs		Outputs			
A	B	D_0	D_1	D_2	D_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

Table 1.5: Truth table of 2-to-4 line decoder.

Decoding Functions:

$$D_0 = \bar{A}\bar{B}$$

$$D_1 = \bar{A}B$$

$$D_2 = A\bar{B}$$

$$D_3 = AB$$

1.14.0.3 2-to-4 line Decoder (Active LOW Design With Enable Input)

- It consist of 2 input lines, A and B, one enable input, E, and 4 output lines, D_0, D_1, D_3 and D_4.

- The design is such that when E=1, all outputs are in high states irrespective of inputs; when E=0, the circuit is enables and accordingly the output asserted LOW as per input applied to input lines, A and B.

- NAND gates are used for this design.

Enable	Inputs		Outputs			
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

Table 1.6: Truth table 2-to-4 line decoder with enable input.

Decoding Functions:

$$D_0 = \overline{E\bar{A}\bar{B}}$$

$$D_1 = \overline{E\bar{A}B}$$

$$D_2 = \overline{EA\bar{B}}$$

$$D_3 = \overline{EAB}$$

1.14.0.4 3-to-8 Line Decoder

- It consists of three input lines and eight output lines.

- The three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables.

- A particular application of this decoder is binary-to-octal conversion. The input variables represent a binary number, and the outputs represent the eight digits of a number

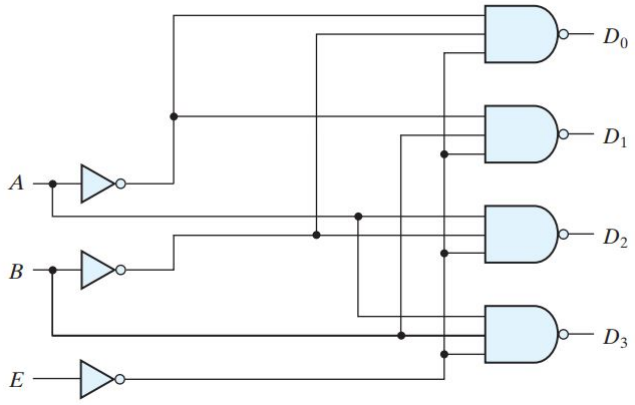


Figure 1.16: A 2-to-4 line decoder with enable input (Active LOW Design).

in the octal number system. However, a three-to-eight-line decoder can be used for decoding any three-bit code to provide eight outputs, one for each element of the code.

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

Figure 1.17: Truth table of 3-to-8 decoder.

1.14.0.5 4-to-16 Line Decoder

1.14.0.6 BCD-to-Decimal Decoder

Design Method 1 (Active HIGH Design with Don't Care):

- The BCD-to-decimal decoder converts each BCD code (8421 code) into one of ten possible decimal digit indications. - It is frequently referred as *1-of-10 decoder* because only 1 of 10 output lines is high.

- The decoder have four inputs to accept the coded digit and ten outputs, one for each decimal digit. This will give a *4-line-to-10-line BCD-to-Decimal decoder*.

- There are six don't care conditions here, and they must be taken into consideration when we simplify each of the output functions.

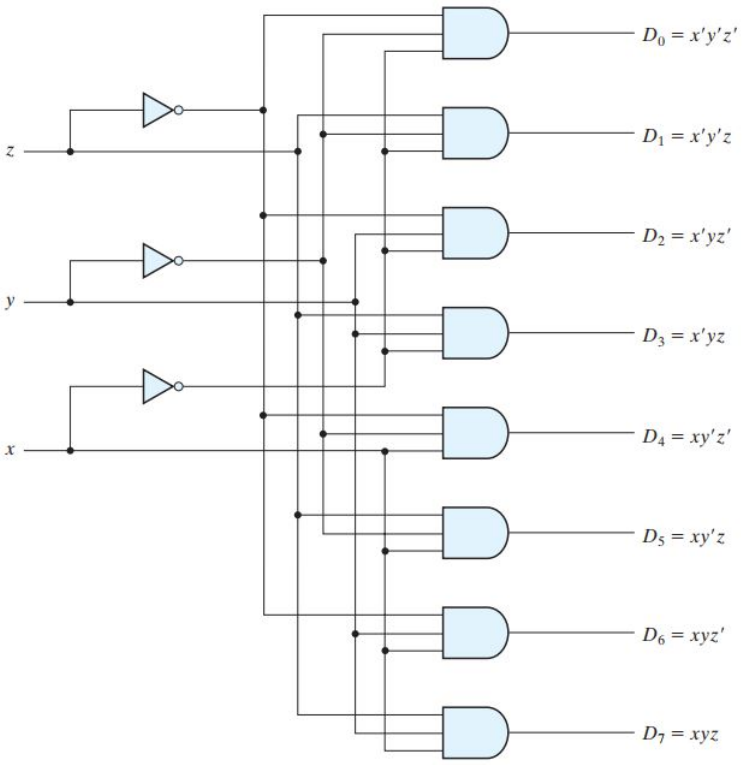


Figure 1.18: 3-to-8 line decoder.

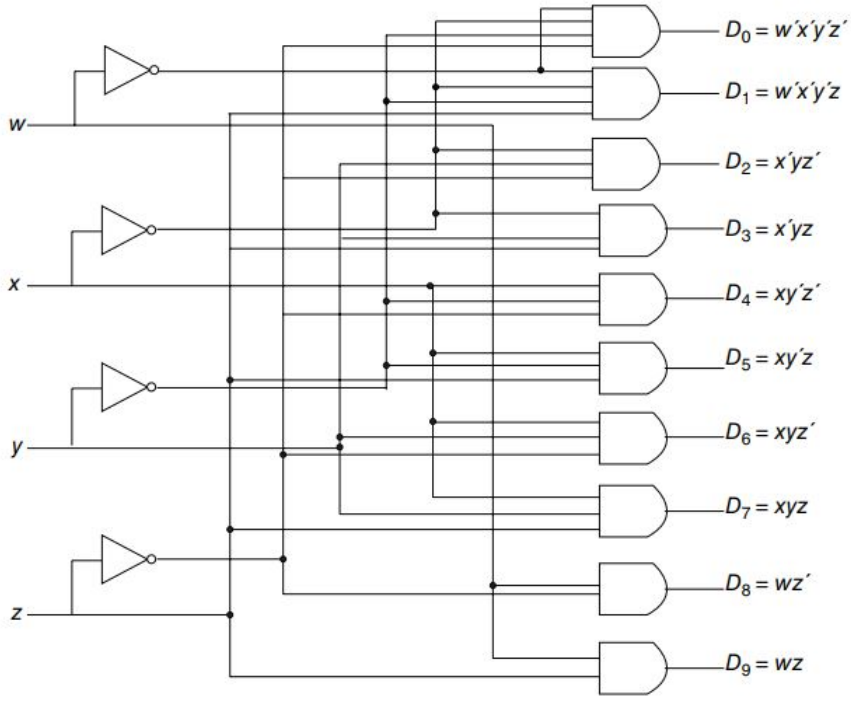


Figure 1.19: BCD-to-Decimal Decoder with Active-HIGH output.

Design Method 2 (Active LOW Design):

1.14.0.7 Decoder as a multiplexer

Decoder with enable

- A decoder with enable input can function as a demultiplexer - a circuit that receives information from a single line and directs it to one of 2^n possible output lines.
- The selection of a specific output is controlled by the bit combination of n selection lines.
- The decoder of figure can function as a one-to-four-line demultiplexer when E is taken as a data input line and A and B are taken as the selection inputs.
- The single input variable E has a path to all four outputs, but the input information is directed to only one of the output lines, as specified by the binary combination of the two selection lines A and B.
- This feature can be verified from the truth table of the circuit. For example, if the selection lines AB = 10, output D_2 will be the same as the input value E, while all other outputs are maintained at 1.
- Because decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a *decoder–demultiplexer*.

1.14.0.8 Decoder Based Combinational Circuit

A decoder provides the 2^n minterms of n input variables. Since any Boolean function can be expressed as a sum of minterms, one can use a decoder to generate the minterms and combine them with an external OR gate to form a sum-of-minterms implementation. In this way, any combinational circuit with n inputs and m outputs can be implemented with an n˘to˘2^n-line decoder and m OR gates.

Example: Implementing Full Adder using Decoder and OR Gates

- If X, Y and Z are inputs of full adder and C ans S are its two outputs, then we have

$$S(X,Y,Z) = \sum m(1,2,4,7)$$
$$C(X,Y,Z) = \sum m(3,5,6,7)$$

The decoder method can be used to implement any combinational circuit. However,

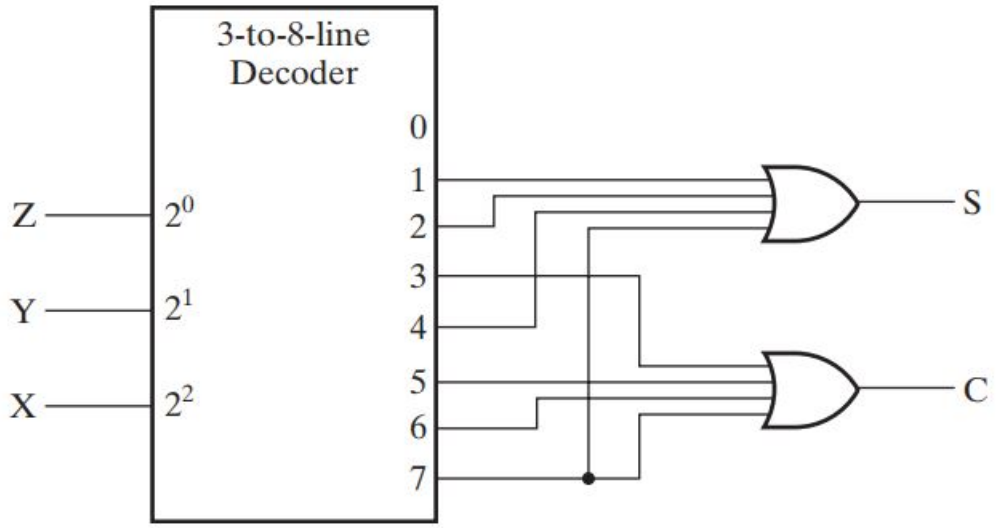


Figure 1.20: Implementing Full Adder using a Decoder.

this implementation must be compared with other possible implementations to determine the best solution. The decoder method may provide the best solution, particularly if the combinational circuit has many outputs based on the same inputs and each output function is expressed with a small number of minterms.

Example: Show how using a 3-to-8 line decoder and multi-input OR gates following Boolean expressions can be realized simultaneously.

$$F_1(A,B,C) = \sum m(0,4,6)$$
$$F_2(A,B,C) = \sum m(0,5)$$
$$F_3(A,B,C) = \sum m(1,2,3,7)$$

Soution: Since at the decoder output we get all the minterms, we use them as shown in figure 1.21) to get the required Boolean functions.

1.14.0.9 Seven Segment Decoders

Seven-Segment Indicator

- Figure 1.22 shows a *seven-segment indicator*, i.e. seven LEDs labelled a through g. By forward-biasing different LEDs, we can display the digits 0 through 9. For instance,

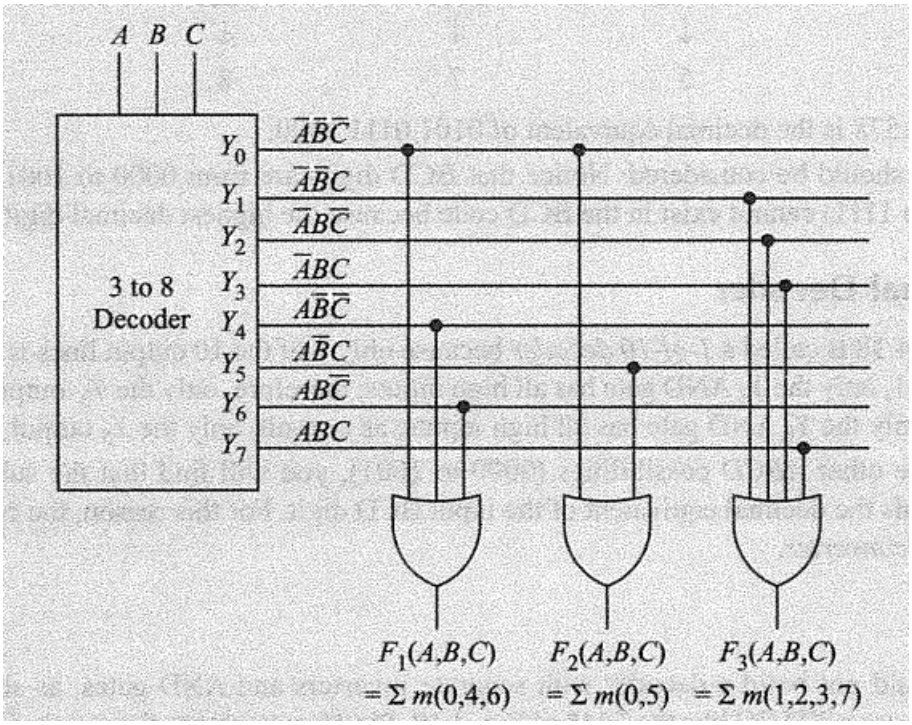


Figure 1.21: Implementing given Boolean function using a Decoder and OR gates.

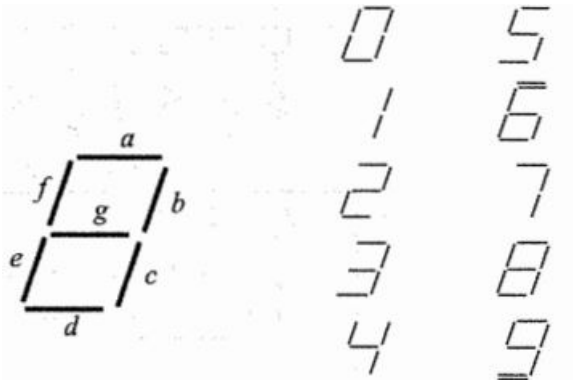


Figure 1.22: Seven-segment indicator.

to display a 0, we need to light up segments *a,b,c,d*, and *f*. To light up a 5, we need segments *a,c,d,f*, and *g*.

- Seven-segment indicators may be the common-anode type where all nodes are connected together or the common-cathode type where all cathodes are connected together. (Figure 1.23) With the common anode type, we have to connect a current limiting resistor between each LED and ground.

- A seven-segment decoder-driver is an IC decoder that can be used to drive a seven-

segment indicator. There are two types of decoder-drivers, corresponding to the common-anode and common-cathode indicators. Each decoder-driver has 4 input pins (the BCD input) and 7 output pins (the a through g segments).

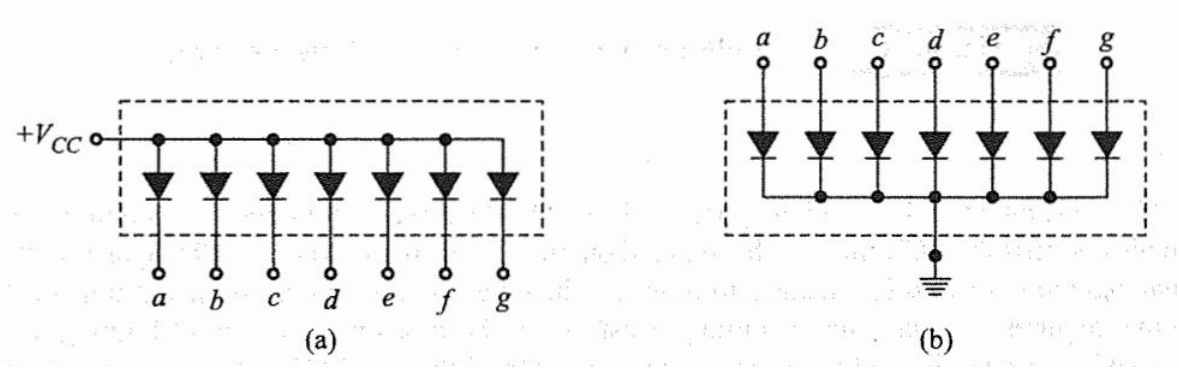


Figure 1.23: (a) Common-anode type (b) Common-cathode type.

- Figure 1.24 a shows 7446 IC is driving a common-anode indicator.For instance, if the BCD input is 0111, the internal logic (not shown) of the 7446 will force LEDs a, b, and c to conduct. As a result, digit 7 will appear on the seven-segment indicator.
- Figure 1.24 b shows 7448 IC driving a common-cathode indicator.Again,internal logic converts the BCD input to the required output. For example, when a BCD input of 0100 is used, the internal logic forces LEDs b, c,f, and g to conduct. The seven-segment indicator then displays a 4. Unlike the 7446 that requires external current-limiting resistors, the 7448 has its own current-limiting resistors on the chip.

1.14.0.10 BCD-TO-Seven Segment Decoder

- The BCD-to-7-segment decoder (such as 7448 IC) accepts the BCD code on its inputs and provides outputs to drive 7-segment display devices to produce a decimal readout. The seven outputs of the decoder (a,b,c,d,e,f,g) select the corresponding the segments in the display.

- This table (Figure 1.26) indicates the segments which are to be driven high to obtain certain decimal digit at the output of the seven segment display. - However, it is to be noted that in the case of common anode type, the only change will be to interchange ones and zeros on the table. This means that from the truth table so obtained one can get to know where low has to be driven so as to obtain the required digit at the output.

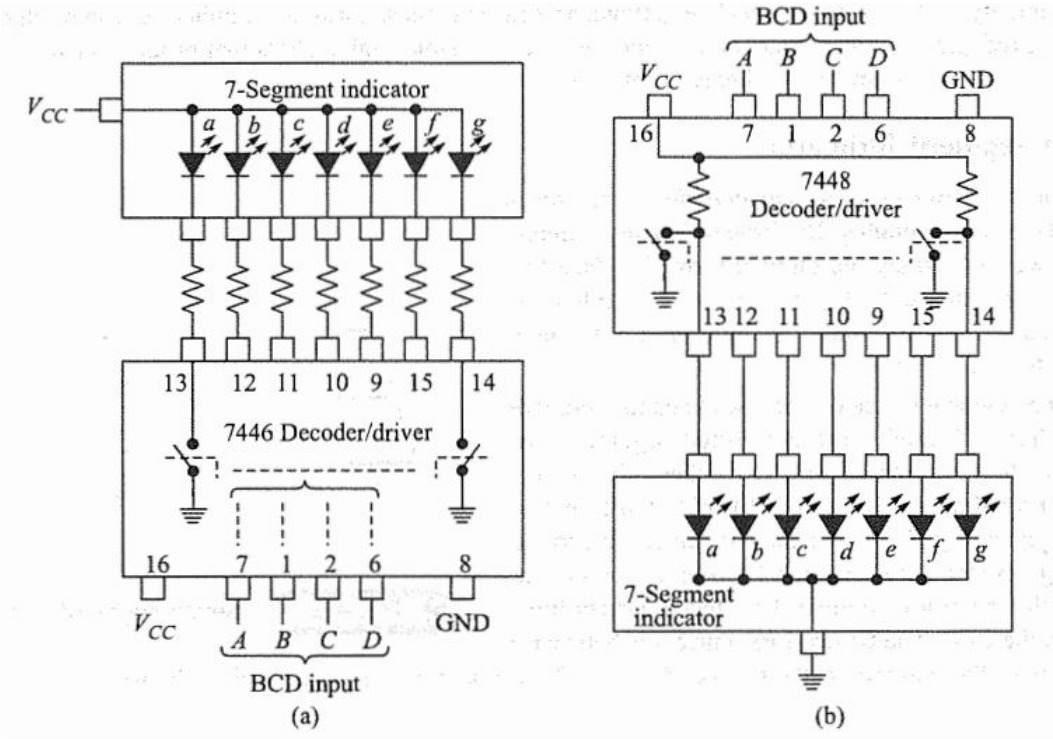


Figure 1.24: Seven-segment indicator.

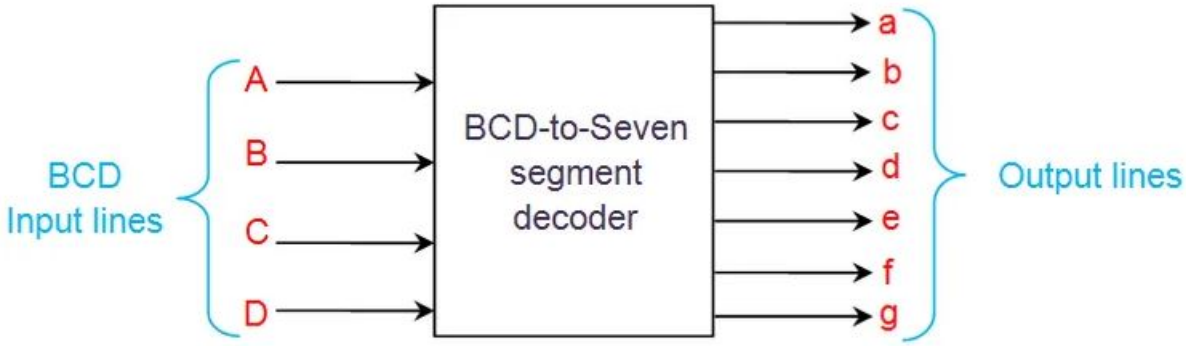


Figure 1.25: BCD-To-Seven-segment decoder.

From truth table:

$$a = \sum m(0,2,3,5,6,7,8,9) + don'tcare(10,11,12,13,14,15,16)$$

$$b = \sum m(0,1,2,3,4,7,8,9) + don'tcare(10,11,12,13,14,15,16)$$

$$c = \sum m(0,1,3,4,5,6,7,8,9) + don'tcare(10,11,12,13,14,15,16)$$

$$d = \sum m(0,2,3,5,6,8) + don'tcare(10,11,12,13,14,15,16)$$

$$e = \sum m(0,2,6,8) + don'tcare(10,11,12,13,14,15,16)$$

$$f = \sum m(0,4,5,6,8,9) + don'tcare(10,11,12,13,14,15,16)$$

$$g = \sum m(2,3,4,5,6,8,9) + don'tcare(10,11,12,13,14,15,16)$$

Decimal Digit	Input lines				Output lines							Display pattern
	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	

Figure 1.26: Truth table of seven-segment decoder for common cathode type.

1.14.0.11 Realizing of Higher Order Decoder from Lower Order Decoder

1.15 Encoders

- An encoder is a combinational logic circuit that essentially performs a “reverse” decoder function.

- An encoder accepts an active level on one of its inputs representing a digit, such as a decimal or octal digit, and converts it to a coded output, such as BCD or binary.

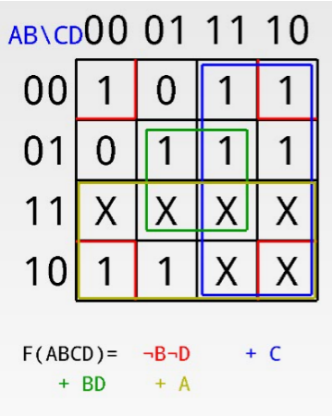
- Encoders can also be devised to encode various symbols and alphabetic characters.

- The process of converting from familiar symbols or numbers to a coded format is called ***encoding***.

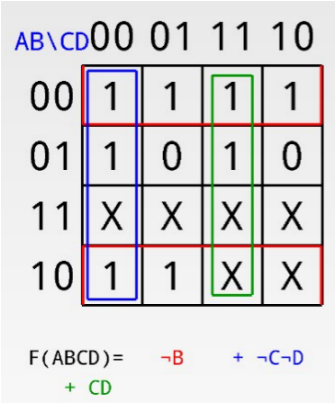
- An encoder has 2^n (or fewer) input lines and n output lines. The output lines generate the binary code for the 2^n input variables. An encoder should have only one input high at a time.

- The encoder can be implemented with OR gates whose inputs are determined directly from the truth table.

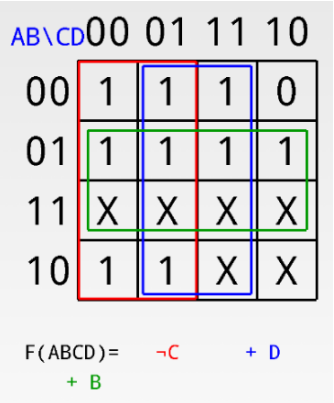
- Example: octal-to-binary encoder, decimal to BCD encoder



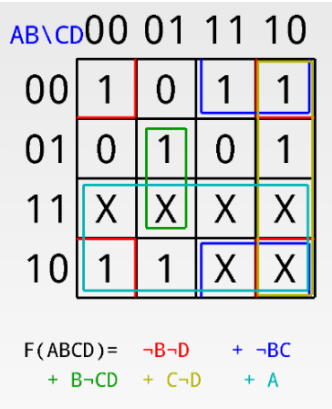
(a) K-map for segment a



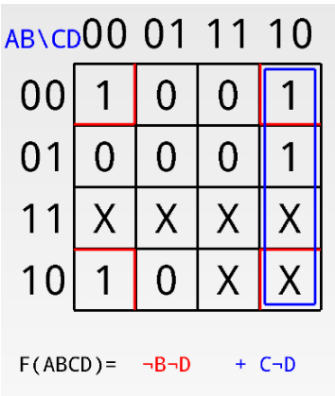
(b) K-map for segment b



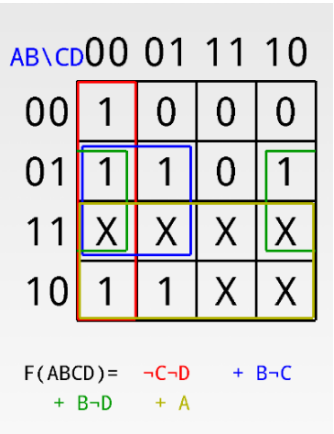
(c) K-map for segment c



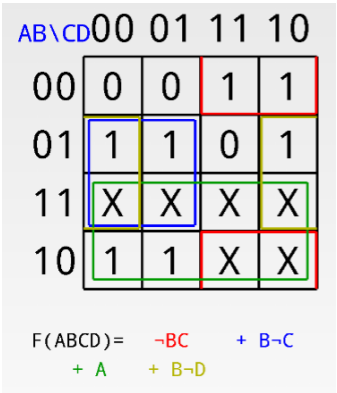
(d) K-map for segment d



(e) K-map for segment e



(f) K-map for segment f



(g) K-map for segment g

Figure 1.27: K-maps for seven segment decoder

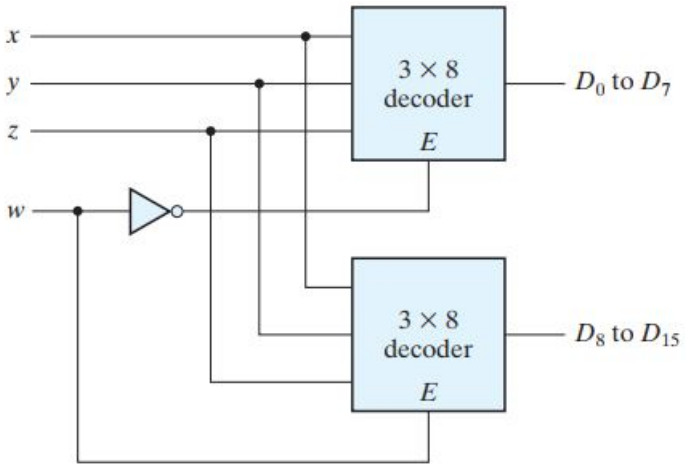


Figure 1.28: 4×16 decoder constructed with two 3×8 decoders.

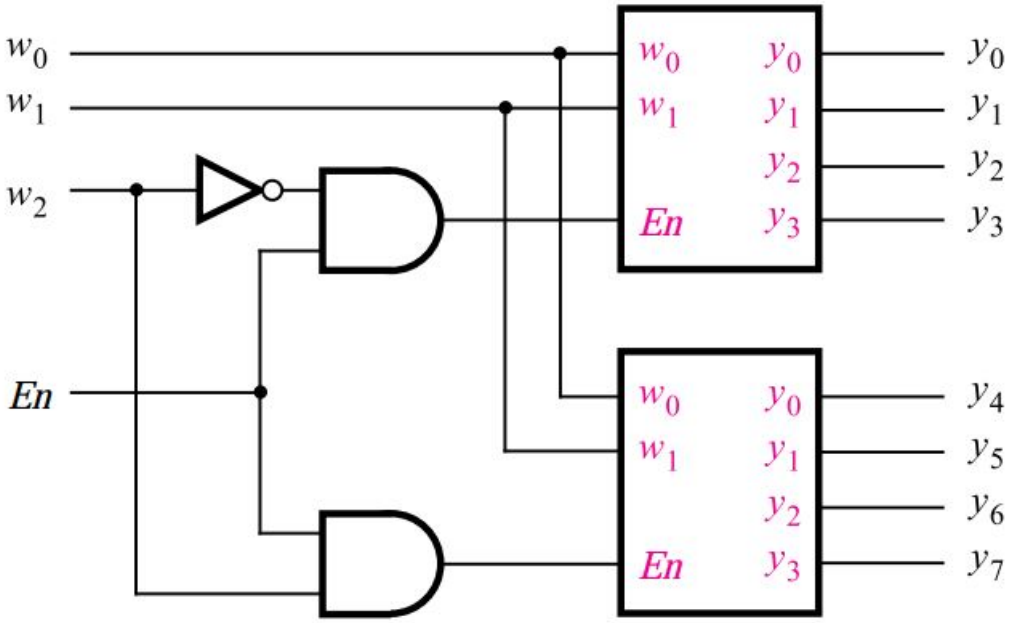


Figure 1.29: 3×8 decoder constructed with two 2×4 decoders.

1.15.0.1 4:2 Encoder

- The 4-to-2 line encoder consists of 4 input lines an 2 output lines.

1.15.0.2 Octal-To-Binary Encoder

- It consists of eight input lines and 3 output lines.

- At a given time, only one input line is high, whose value is then encoded into 3-bits.

- Let $D_0, D_1, D_2, D_3, D_4, D_5, D_6$, and D_7 be inputs binary variables and x,y and z be output binary variables.

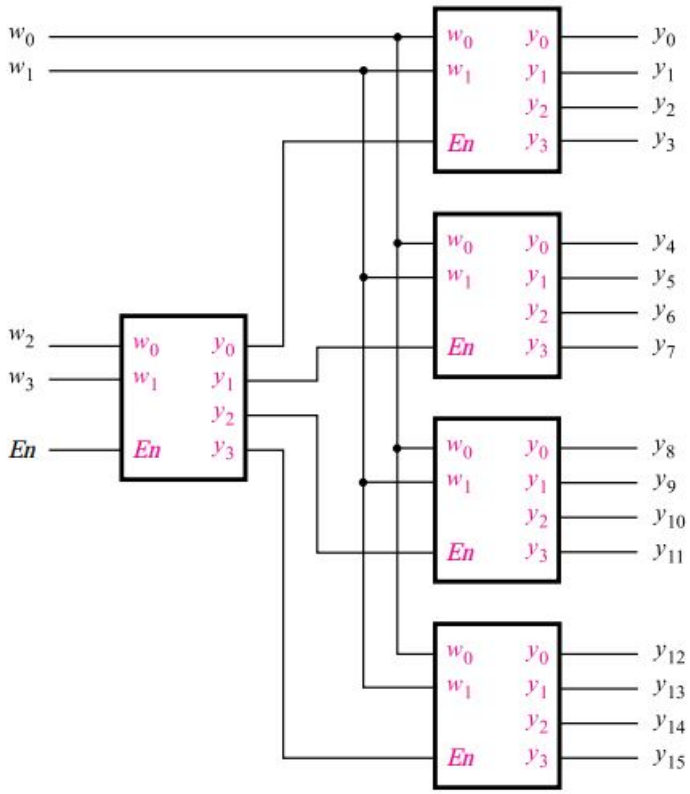


Figure 1.30: A 4-to-16 line multiplexer using 2-to-4 line decoders.

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

Figure 1.31: Truth table for octal-to-binary encoder.

From truth table, the output Boolean functions are:

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

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

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

which can be implemented with three 4-input OR gates.

Drawback of Normal Encoder

1. There is ambiguity, an output with all 0's is generated when all the inputs are 0; but this output is the same as when D_0 is equal to 1. For example, if D_3 and D_6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. The output 111 does not represent either binary 3 or binary 6. To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded.
2. If more than one input is active HIGH, then the encoder produces an output, which may not be correct code. The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

1.15.0.3 4-to-2 Line Priority Encoder

- A priority encoder is an encoder circuit that includes the priority function.
- The operation of the priority encoder is such that if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.
- The truth table of a four-input priority encoder is given in talbe 1.7.
- In addition to the two outputs x and y, the circuit has a third output designated by V; this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.
- The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions.
- Note that whereas X 's in output columns represent don't-care conditions, the 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, the truth table uses an X to represent either 1 or 0. For example, X 100 represents the two minterms 0100 and 1100.

Boolean Functions:

$$x = D_2 + D_3$$

$$y = D_3 + D_1\bar{D}_2$$

$$V = D_0 + D_1 + D_3 + D_4$$

Inputs				Outputs		
D_4	D_3	D_1	D_0	x	y	V
0	0	0	0	X	X	0
0	0	0	1	0	0	1
0	0	1	X	0	1	1
0	1	X	X	1	0	1
1	X	X	X	1	1	1

Table 1.7: Condensed truth table for priority encoder.

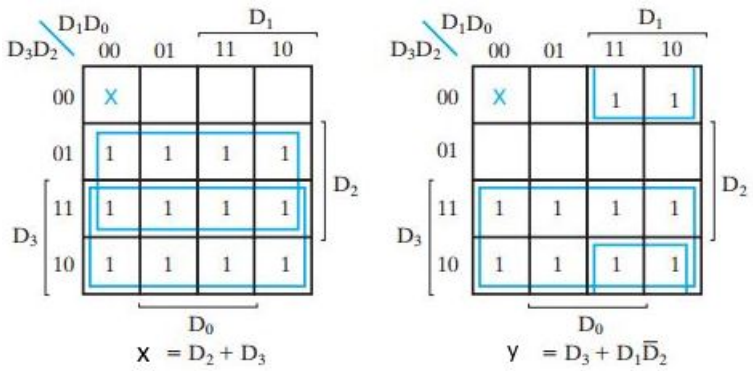


Figure 1.32: Kmap for x and y.

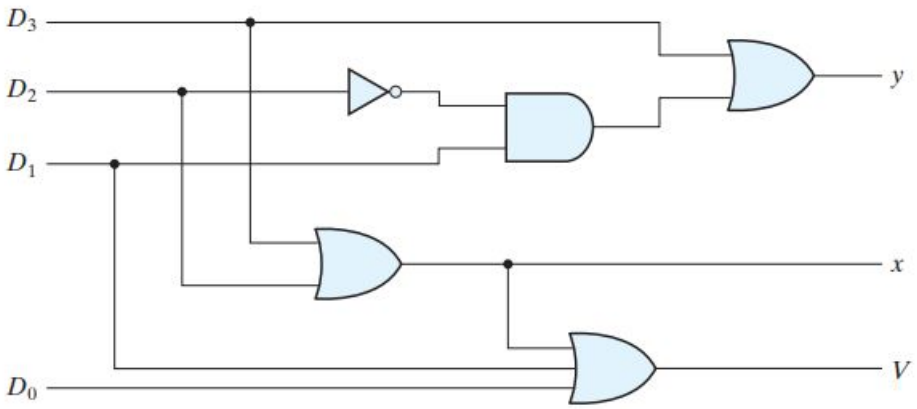


Figure 1.33: Octal-to-binary encoder.

1.15.0.4 Octal-to-Binary Priority Encoder

See; `https://www.electronics-tutorials.ws/combination/comb_4.html`

1.15.0.5 Decimal-to-BCD Encoder

- This type of encoder has ten inputs - one for each decimal digit- and four outputs corresponding to the BCD code.

1.16 Multiplexers

- A multiplexer circuit has a number of data inputs, one or more select inputs, and one output.

- A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs the information to a single output line. The selection of a particular input line is controlled by a set of input variables, called *selection inputs*.

-Normally, there are 2^n input lines and n selection inputs whose bit combinations determine which input is selected.

- It is also known as '*Data Selector*'. In short, it is called, *MUX*.

1.16.0.1 2-to-1 MUX

- It has 2 input lines, one selection line and one output line. Let I_0 and I_1 be binary input variables, S be selection line and Y be output binary variable.

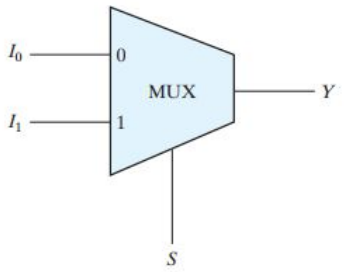


Figure 1.34: Block Diagram of 2:1 MUX

Inputs			Output
S	I_0	I_1	**Y**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Table 1.8: Truth table for 2:1 MUX.

Boolean Expression:$Y = \bar{S}I_0 + SI_1$

1.16.0.2 4-to-1 Line Multiplexer

- The output, Y, depends on four inputs, say I_0, I_1, I_2 and I_4 and two selection lines, say S_0 and S_1.

The data output is equal to D_0 only if $S_1 = 0$ and $S_0 = 0$: Y = $I_0\bar{S_1}\bar{S_0}$.

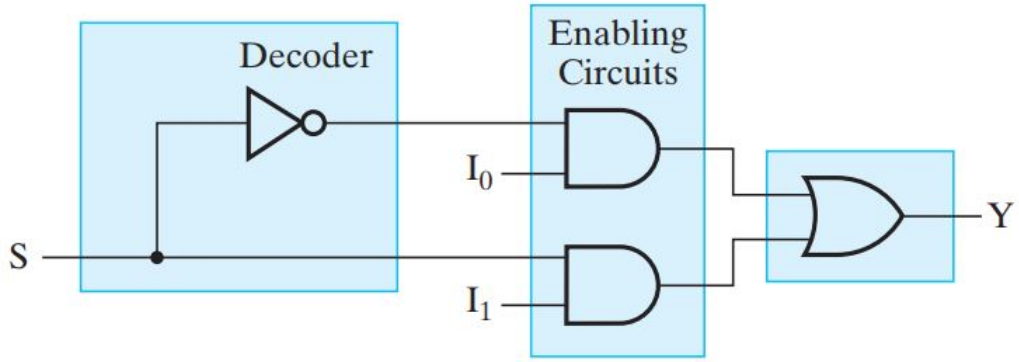


Figure 1.35: 2-to-1 line MUX.

Selection Lines		Output
S_1	S_0	**Y**
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Table 1.9: Function table for 4:1 MUX

The data output is equal to I_1 only if $S_1 = 0$ and $S_0 = 1$: $Y = I_1\bar{S_1}S_0$.

The data output is equal to I_2 only if $S_1 = 1$ and $S_0 = 0$: $Y = I_0S_1\bar{S_0}$.

The data output is equal to I_3 only if $S_1 = 1$ and $S_0 = 1$: $Y = I_0S_1S_0$.

When these terms are ORed, the total expression for the data output is

$$Y = I_0\bar{S_1}\bar{S_0} + I_1\bar{S_1}S_0 + I_0S_1\bar{S_0} + I_0S_1S_0$$

If this equation is implemented directly, two inverters, four 3-input AND gates, and a 4-input OR gate are required, giving a gate-input cost of 18 (Figure 1.36). A different implementation (Figure 1.37) can be obtained by factoring the AND terms to give

$$Y = I_0(\bar{S_1}\bar{S_0}) + I_1(\bar{S_1}S_0) + I_0(S_1\bar{S_0}) + I_0(S_1S_0)$$

1.16.0.3 64-to-1 Line Multiplexer

A multiplexer is to be designed for n = 6. This will require a 6–to–64-line decoder and a 64 $\times$ 2 AND-OR gate. The resulting structure is shown in Figure 3-26

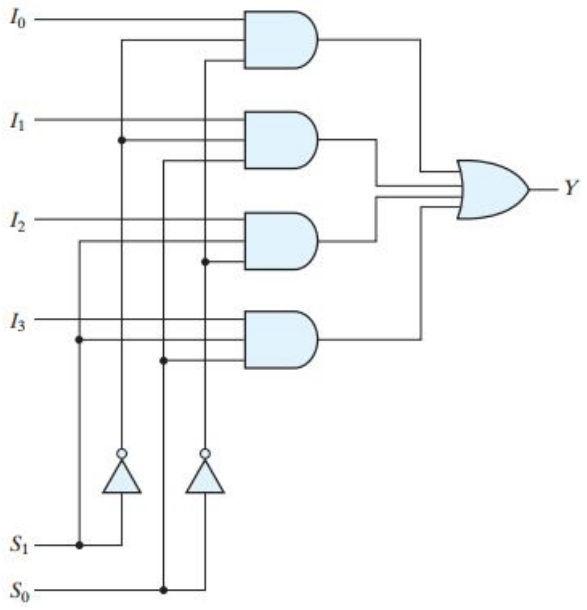


Figure 1.36: 4-to-1 line MUX.

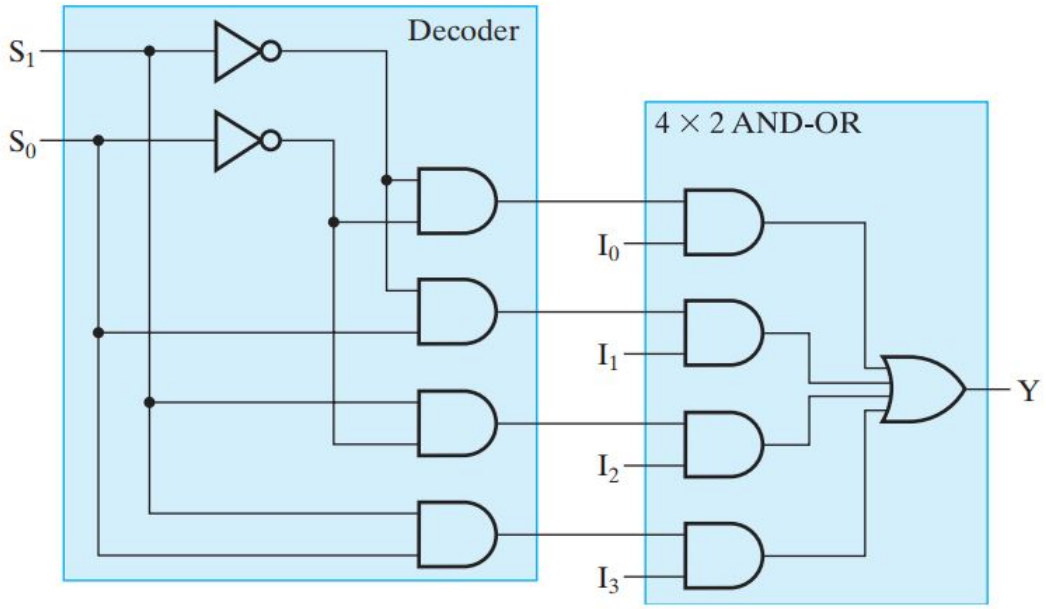


Figure 1.37: 4-to-1 line MUX (Using decoder circuit and AND-OR gate).

1.16.0.4 Quadruple 2-to-1 Line Multiplexer (Nibble Multiplexer)

- Sometimes we want to select one of two input nibbles. In this case, we can use a nibble multiplexer like the one shown in 1.39.

- The two 4-bit inputs are $A(=A_3A_2A_1A_0)$ and $B(=B_3B_2B_1B_0)$.

- The control signal labelled select (s) determines which input nibble is transmitted to the output.

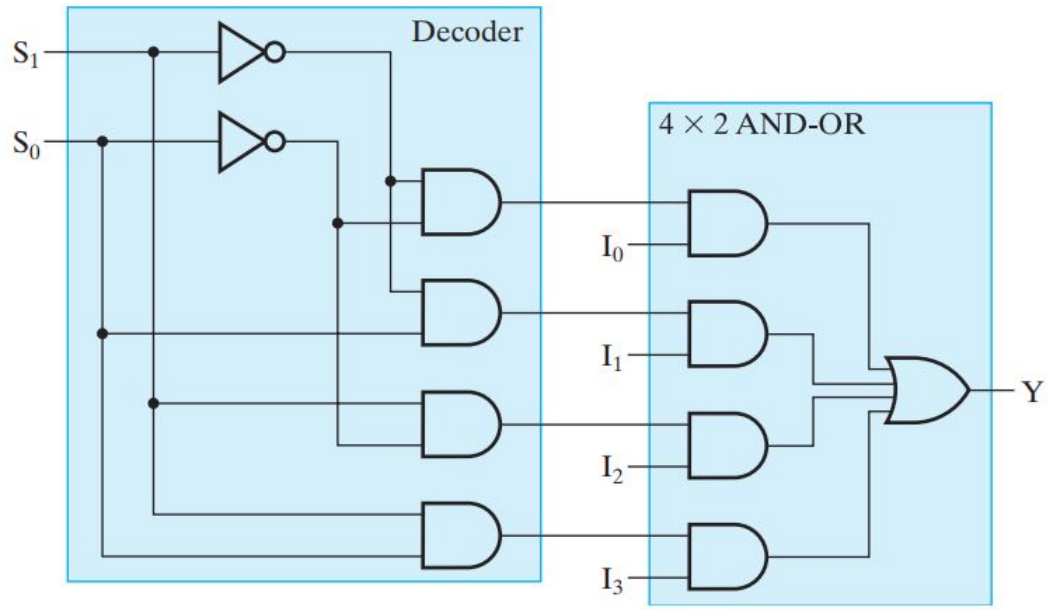
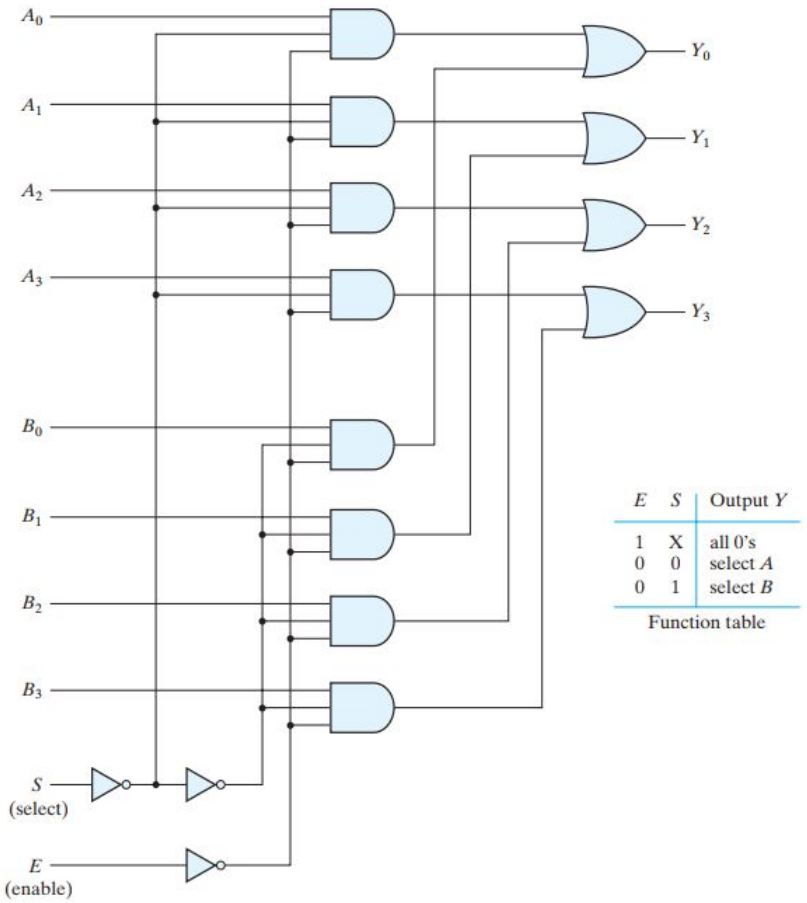


Figure 1.38: 64-to-1 line MUX (Using decoder circuit and AND-OR gate).

- Enable (E) signal is used to enable the circuit. When E=1, all the outputs equal to 0. When E=0, input is selected, determined by value of select.s



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

Function table

Figure 1.39: Quadruple 2-to-1 line multiplexer with enable input.

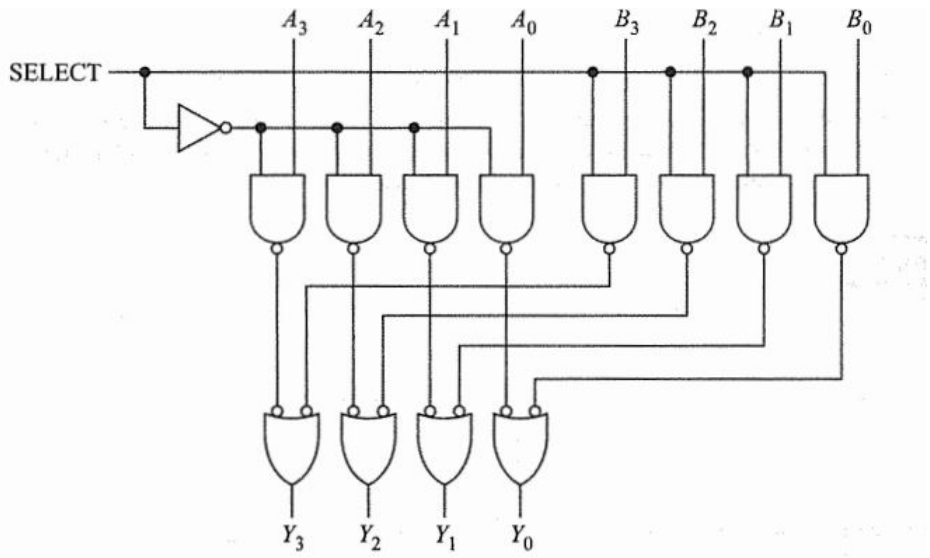


Figure 1.40: Nibble multiplexer (Without enable).

1.16.0.5 A Quad 4-to-1 Line Multiplexer

- A quad 4–to–1-line multiplexer, which has two selection inputs and each information input replaced by a vector of four inputs, is to be designed.
- Since the information inputs are a vector, the output Y also becomes a four-element vector.
- The implementation for this multiplexer requires a 2–to–4-line decoder, and four 4 $\times$ 2 AND-OR gates. The resulting structure is shown in figure 1.41.

1.16.0.6 Multiplexer Based Combinational Circuits

Implementing Boolean Functions with Multiplexer

Question: (a) Realize $Y = \bar{A}B + \bar{B}\bar{C} + ABC$ using an 8-to-1 line multiplexer. (b) Can it be realized with 4-to-1 multiplexer?

Solution

(a) First we express Y as a function of minterms of three variables.Thus

$$Y = \bar{A}B + \bar{B}\bar{C} + ABC$$

$$Y = \bar{A}B(C + \bar{C}) + \bar{B}\bar{C}(A + \bar{A}) + ABC$$

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$$

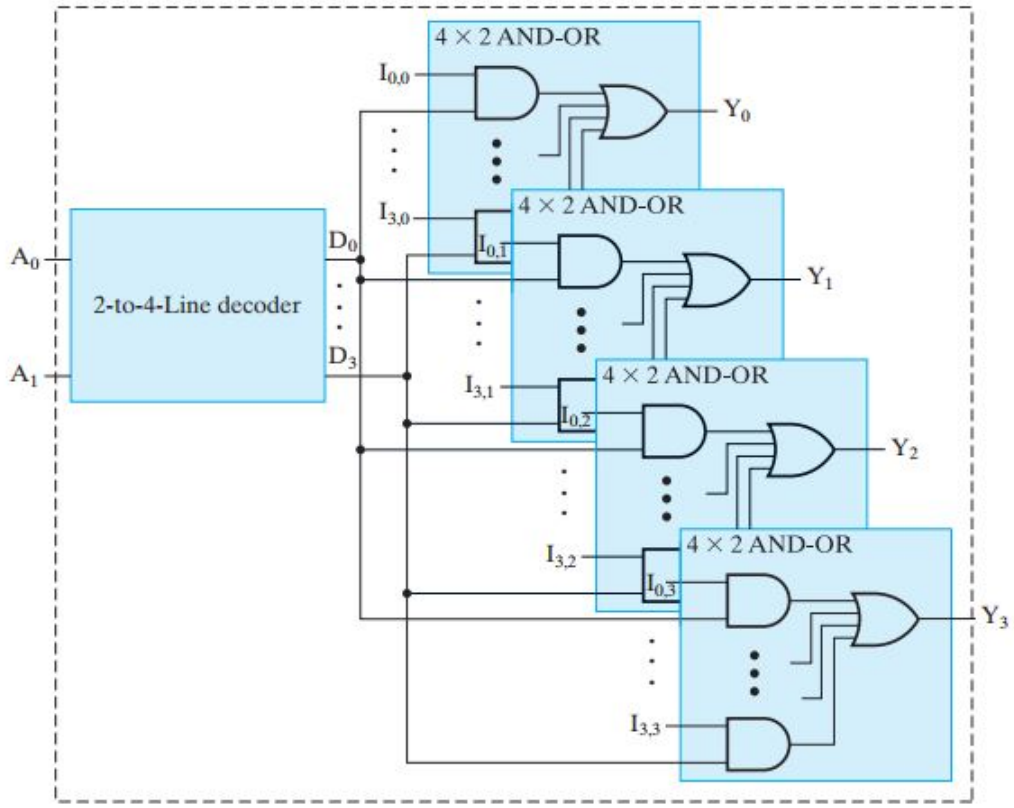


Figure 1.41: A quad 4-to-1 line multiplexer.

Comparing this with equation of 8-to-1 multiplex, we find by substituting $I_0 = I_2 = I_3 = I_4 = I_7 = 1$ and $I_1 = I_5 = I_6 = 0$, we get given logical relation.

(b). Let variables A and B be used .as selector in 4 to 1 multiplexer .and C fed as input The 4-to-1 multiplexer generates 4 minterms for different combinations of AB. We rewrite given logic equation in such a way that all these terms are present in the equation.

$$Y = \bar{A}B + \bar{B}\bar{C} + ABC$$

$$Y = \bar{A}B + \bar{B}\bar{C}(A + \bar{A}) + ABC$$

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B.1 + A\bar{B}\bar{C} + AB.C$$

Comparing above with equation of a 4-to-1 line multiplexer, we see $I_0 = \bar{C}$,$I_1 = 1$, $I_2 = \bar{C}$ and $I_3 = C$ generate the given logic function.

General Procedure for implementing any Boolean function of n variables

- Use a multiplexer that has $n-1$ selection lines and 2^{n-1} data inputs.

- Connect first $n-1$ variables of the functions to the selection inputs of the multiplexer.

- Use the remaining single variable of the function for the data inputs; the data input can be 0, or 1, or normal value, or its complement.

Example: Implementing $F(x,y,z) = \sum(1,2,6,7)$ using suitable multiplexer.

Solution:

- We implement the given function using 4:1 line multiplexer.
- x and y variables are applied to the selection lines **in order**; x in connected to S_1 and y to S_0.
- The values for the data input lines are determined from the truth table shown in figure 1.42 a. The multiplexer implementation is shown in figure 1.42 b.

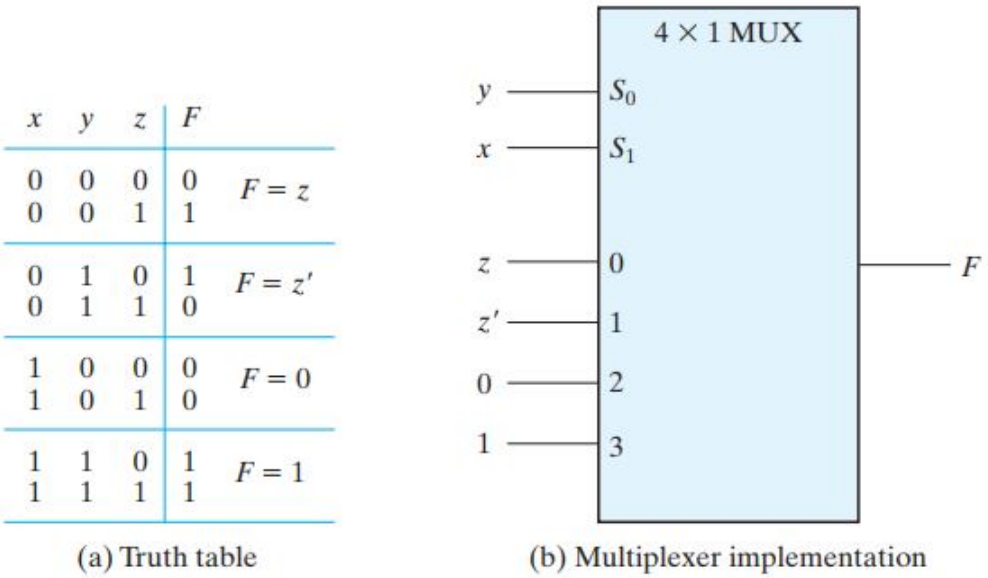


Figure 1.42: Implementing a Boolean function $F(x,y,z) = \sum(1,2,6,7)$ using 4:1 Mux .

- When $xy = 00$, output F is equal to z because $F = 0$ when $z = 0$ and $F = 1$ when $z = 1$. This requires that variable z be applied to data input 0. The operation of the multiplexer is such that when $xy = 00$,data input 0 has a path to the output, and that makes F equal to z.
- In a similar fashion, we can determine the required input to data lines 1, 2, and 3 from the value of F when xy = 01, 10, and 11, respectively.

Example: Implement $F(A,B,C,D) = \sum m(1,3,4,11,12,13,14,15)$.

Solution:

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	

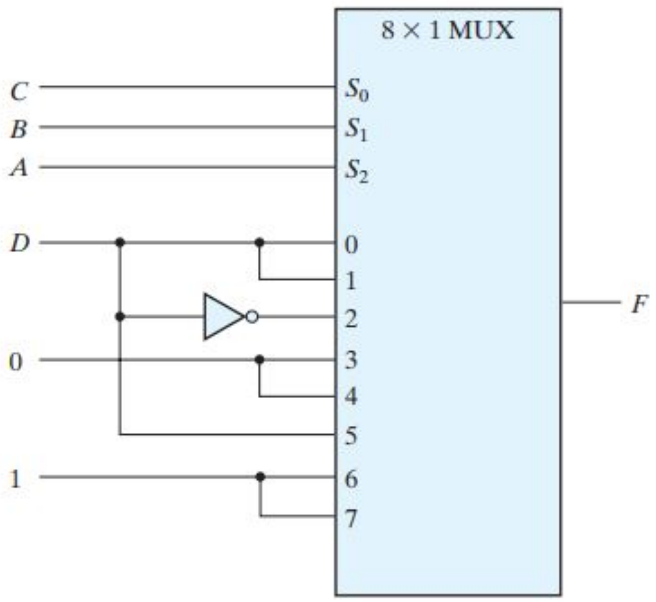


Figure 1.43: Implementing a Boolean function $F(x,y,z) = \sum(1,3,4,11,12,13,14,15)$ using 8:1 line Mux .

Alternate Method

Method 1: Choosing leftmost bit as input

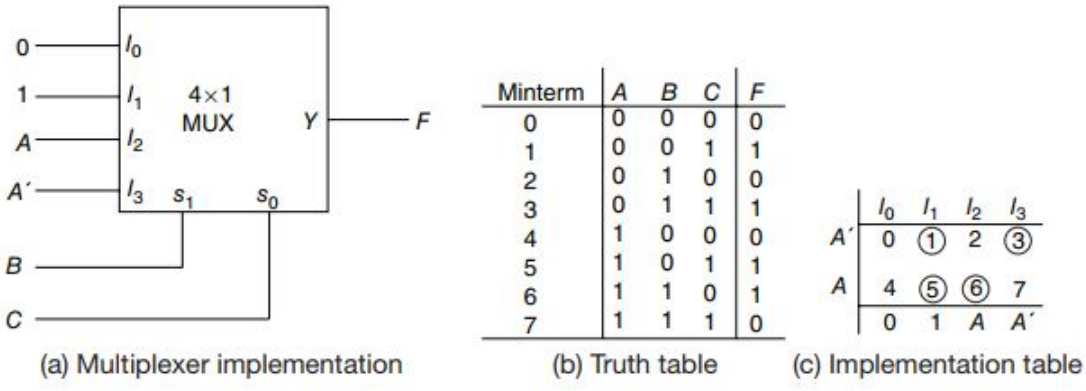
Say the Boolean function is made of three variables, A,B, and C, A being MSB and C being LSB. Then follow the following procedure: - List the inputs of the multiplexer and under them list all the minterms in two rows. - The first row lists all those minterms where A is complemented, and the second row all the minterms with A uncomplemented, as shown in figure 1.44. - Circle all the minterms of the function and inspect each column separately.

(i) If the two minterms in a column are not circled, apply 0 to the corresponding multiplexer input.

(ii) If the two minterms are circled, apply 1 to the corresponding multiplexer input.

(iii) If the bottom minterm is circled and the top is not circled, apply A to the corresponding multiplexer input.

(iv) If the top minterm is circled and the bottom is not circled, apply $\bar{A}$ to the corresponding multiplexer input.

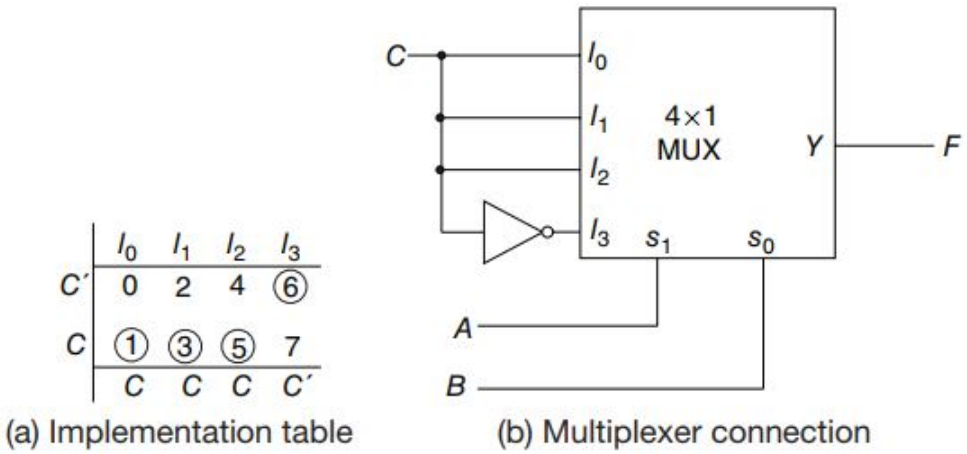


Minterm	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

	I_0	I_1	I_2	I_3
A´	0	①	2	③
A	4	⑤	⑥	7
	0	1	A	A´

Figure 1.44: Implementing a Boolean function $F(A,B,C) = \sum(1,3,5,6)$ with a multiplexer .

Method 2: Choosing right most bit as input



	I_0	I_1	I_2	I_3
C´	0	2	4	⑥
C	①	③	⑤	7
	C	C	C	C´

Figure 1.45: Alternate implementation of a Boolean function $F(A,B,C) = \sum(1,3,5,6)$ with a multiplexer .

Assignment:

Implement 3-input XOR using (a) two 2:1 Mux (b) 4:1 multiplexer

1.16.0.7 Multiplexer Using Three-State Gates

- A multiplexer can be constructed with three-state gates digital circuits that exhibit three states.

- Two of the states are signals equivalent to logic 1 and logic 0 as in a conventional gate. The third state is a *high-impedance state*.

- Three state gate is most commonly performed on buffer gate.

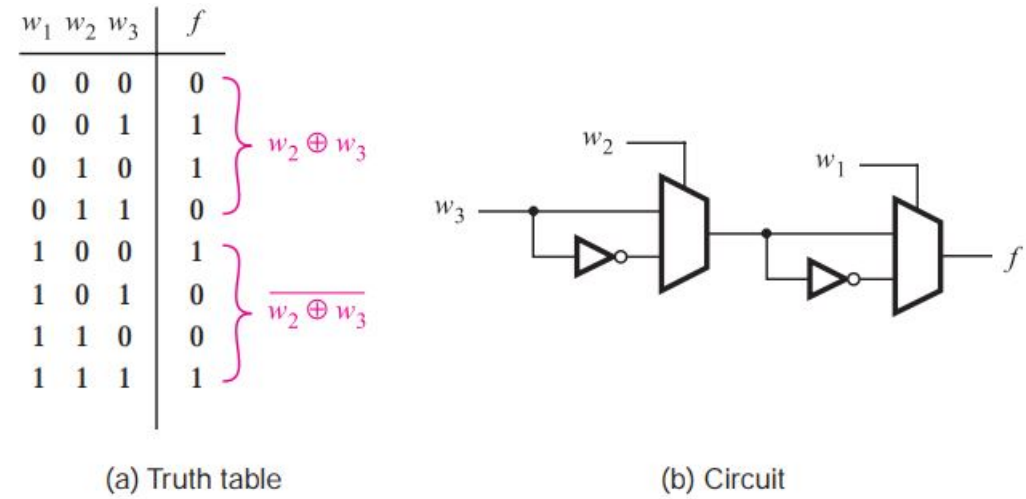


Figure 1.46: Implementing 3-input XOR using two 2:1 multiplexers .

Example:

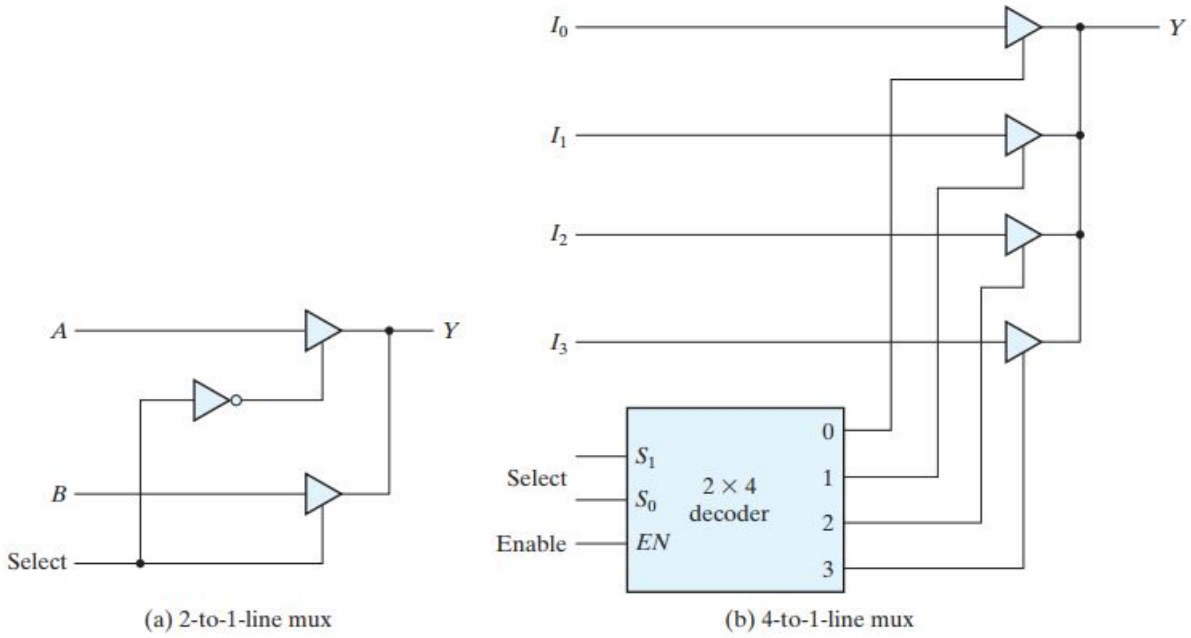
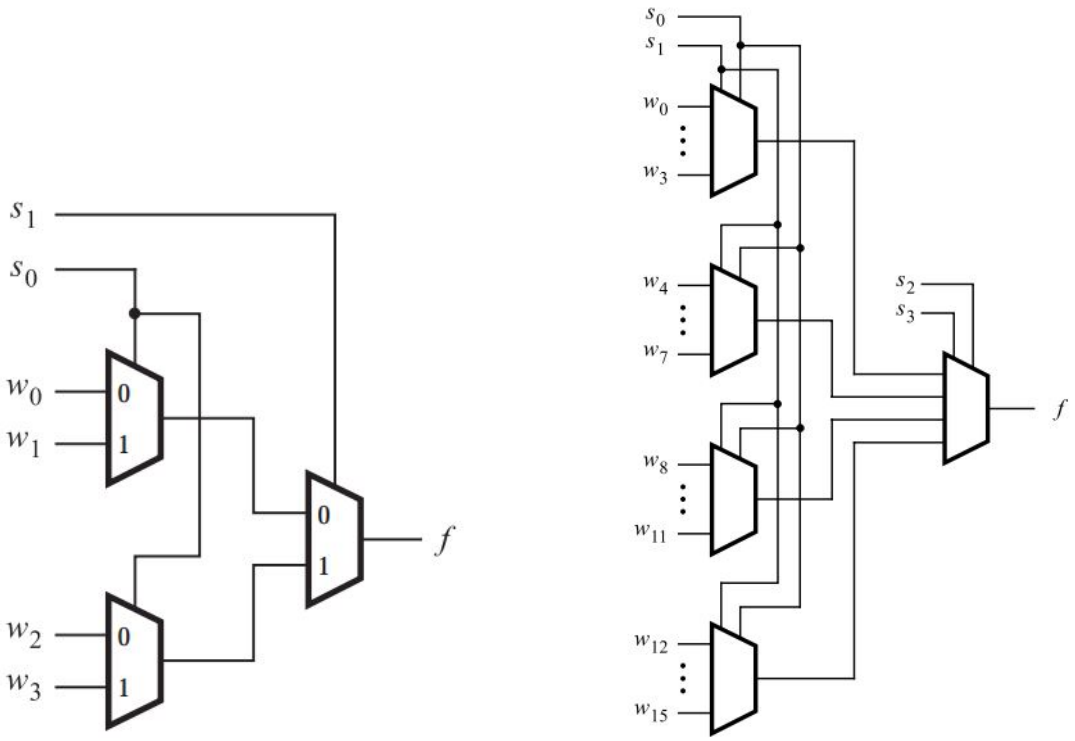


Figure 1.47: Multiplexers with three-state gates and decoder.

1.16.0.8 Realization of Higher Order Multiplexers Using Lower Orders

- Higher order multiplexer are implemented using lower orders multiplexer using mux-tree concept.

- See figure 1.48



(a) Using 2-to-1 multiplexers to build a 4-to-1 multiplexer

(b) Using 4-to-1 multiplexers to build a 16-to-1 multiplexer.

Figure 1.48: Higher order multiplexer using low order multiplexers.

1.17 Demultiplexers

- Demultiplex means one into many. A demultiplexer is a logic circuit with one input and many outputs.

- A demultiplexer receives information on a single line and transmit this information on one of 2^n possible output lines. The selection of a specific output line is controlled by the bit values of n selection lines.

- A demultiplexer (DEMUX) basically reverses the multiplexing function.

- It takes digital information from one line and distributes it to a given number of output lines. For this reason, the demultiplexer is also known as a *data distributor*.

- *A decoder with an enable input can function as a demultiplexer.* Referring to the figure 1.51 , the decoder function as demultiplexer if the E line is taken as a data input line and lines A and B are taken as the selection lines.

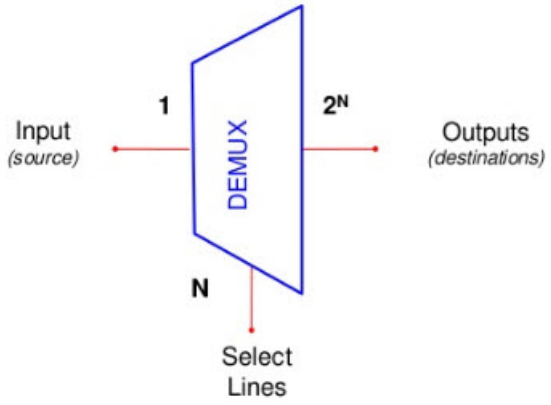


Figure 1.49: A block diagram of demultiplexer .

- A decoder and demultiplexer are obtained from the same circuit, a decoder with an enable input is referred to as a ***decoder/demultiplexer***. It is the enable input that makes the circuit a demultiplexer; the decoder itself can use AND, NAND, or NOR gates.

1.17.0.1 1-to-4 Line Demultiplexer

Design 1:

- It consists of one input line, 2 selection lines and 4 output lines.

- Let the selection lines be A and B, output lines be D_0, D_1, D_3 and D_4 and I be data input.

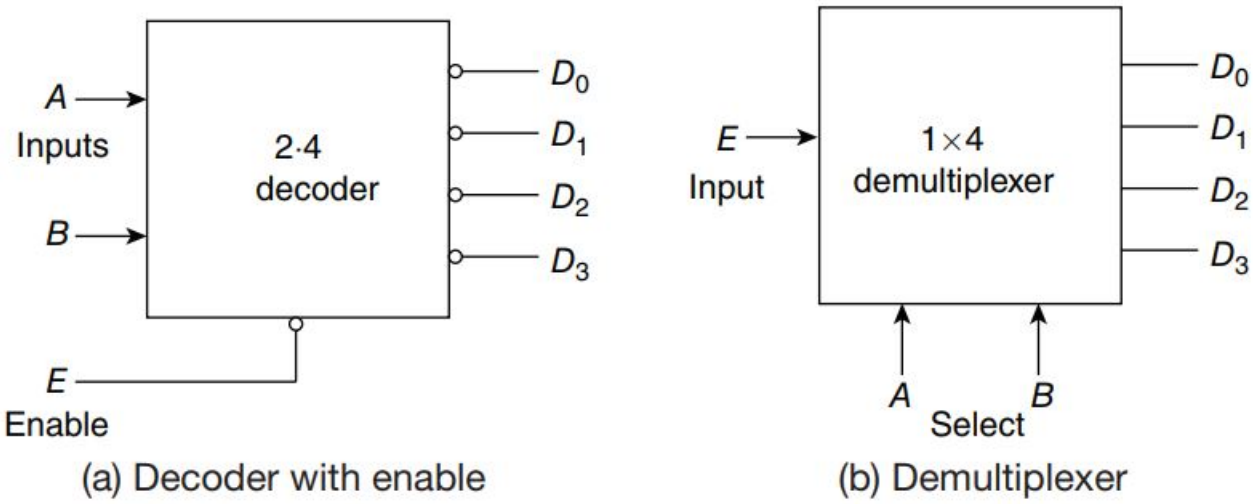


Figure 1.50: A 2-to-4 line decoder with enable acting as a demultiplexer .

Data Input	Selection Lines		Outputs			
I	A	B	D_0	D_1	D_2	D_3
1	0	0	1	0	0	0
1	0	1	0	1	0	0
1	1	0	0	0	1	0
1	1	1	0	0	0	1

Table 1.10: Truth table 1-to-4 line demultiplexer

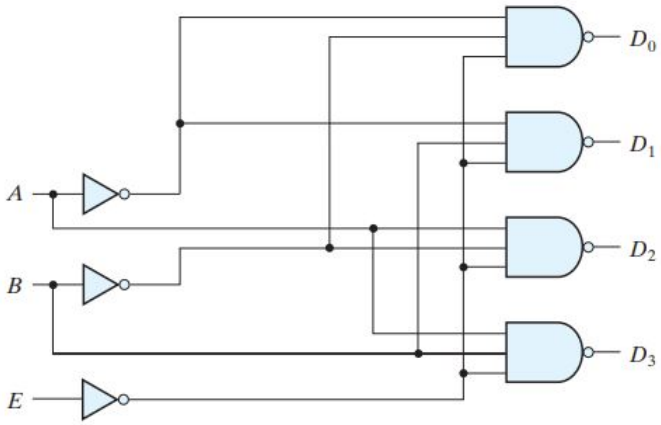
Boolean Expression:

$$D_0 = I\bar{A}\bar{B}$$

$$D_1 = I\bar{A}B$$

$$D_2 = IA\bar{B}$$

$$D_3 = IAB$$



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	1

Figure 1.51: Block diagrams.

1.18 Magnitude Comparator

- Magnitude comparator is a combination circuit that compares two n-bit binary numbers, say X and Y, and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether $X > Y$, $X = Y$, or $X < Y$.

- The logic equations for the outputs can be written as follows, where *G, L, E* stand for greater than, less than and equal to respectively.

- The circuit for comparing two n-bit numbers has 2^{2n} entries in the truth table and becomes too cumbersome, even with $n = 3$. So, we will use algorithmic procedure to design magnitude comparator, as it possess an inherent well-defined regularity.

1-bit magnitude comparator

Inputs		Outputs		
X	Y	$X > Y$	$X = Y$	$X < Y$
0	0	0	0	1
0	1	0	1	0
1	0	1	0	0
1	1	0	0	1

Table 1.11: Truth table for 1-bit magnitude comparator.

Boolean Expression:

$$(X > Y) : G = X\bar{Y}$$

$$(X < L) : L = \bar{X}Y$$

$$(X = Y) : E = \bar{X}\bar{Y} + XY = \overline{X\bar{Y} + \bar{X}Y} = \overline{G + L}$$

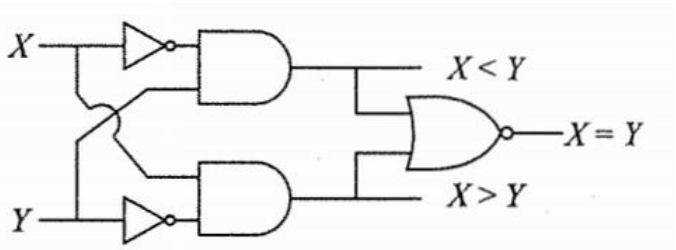


Figure 1.52: 1-bit magnitude comparator.

2-bit Magnitude Comparator

- We can form a 4-variable ($X : X_1X_0$ and $Y : Y_1Y_0$) truth table and get logic equations through any simplification technique. But this procedure will become very complex if we try to design a comparator for 3-bit numbers or more.

- - Here, we discuss a simple but generic procedure for 2-bit comparator design, which

can easily be extended to make any *n*-bit magnitude comparator.

- We shall use the truth table of 1-bit comparator that generates greater than, less than and equal terms.

Bit-wise greater than terms (G): $G_1 = X_1\bar{Y_1}$ $G_0 = X_0\bar{Y_0}$

Bit-wise less than terms (L): $L_1 = \bar{X_1}Y_1$ $G_0 = \bar{X_0}Y_0$

Bit-wise equality term(E): $\overline{(G_1 + L_1)}$ $E_0 = \overline{(G_0 + L_0)}$

From above definitions we can easily write 2-bit comparator outputs as follows:

$$(X = Y) = E_1.E_0$$
$$(X > Y) = G_1 + E_1.G_0$$
$$(X < Y) = L_1 + E_1.L_0$$

For 3-bit magnitude comparator

$$(X = Y) = E_2.E_1.E_0$$
$$(X > Y) = G_2 + E_2.G_1 + E_2.E_1.G_0$$
$$(X < Y) = L_3 + E_2.L_1 + E_2.E_1.L_0$$

Thus, for any two n-bit numbers, we can deduce the three outputs as follows:

$$(X = Y) = E_{n-1}.En_2...E_0$$
$$(X > Y) = G_{n-1} + E_{n-1}.G_{n-2} + ... + E_{n-1}.E_{n-2}....G_0$$
$$(X < Y) = L_{n-1} + L_{n-1}.L_{n-2} + ... + E_{n-1}.E_{n-2}....L_0$$

1.19 Binary Multiplier

- Multiplication of binary numbers is performed in the same way as multiplication of decimal numbers.

- The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit.

- Each such multiplication forms a partial product. Successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products.

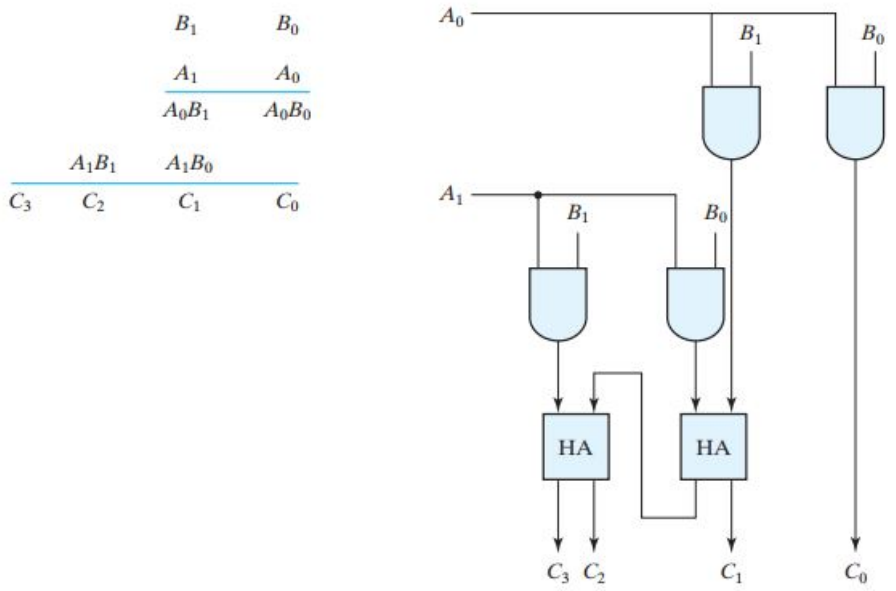


Figure 1.53: A 2-bit by 2-bit binary multiplier.

- Here, in figure 1.53, AND gates and two half-adder circuit are employed to add partial products.
- 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. For J multiplier bits and K multiplicand bits, we need $(J \times K)$ AND gates and $(J-1)$ K-bit adders to produce a product of $(J+K)$ bits.
- Note that the least significant bit of the product does not have to go through an adder, since it is formed by the output of the first AND gate.

4-bit by 3-bit multiplier

- Let us consider a 4 bit multiplicand represented by $B_3B_2B_1B_0$ and 3-bit the multiplier be represented by $A_2A_1A_0$. Since $K = 4$ and $J = 3$, we need 12 AND gates and two 4-bit adders to produce a product of seven bits. The logic diagram of the multiplier is shown in 1.54.

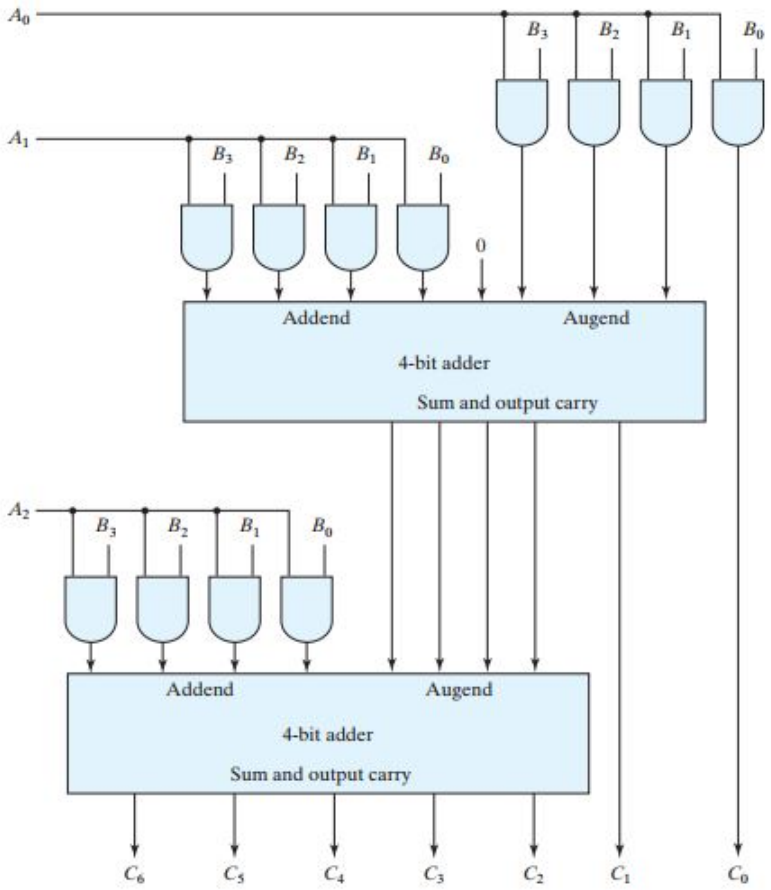


Figure 1.54: A 4-bit by 3-bit binary multiplier.

1.20 Arithmetic Logic Unit

- Arithmetic Logic unit, popularly called ALU is multifunctional device that can perform both arithmetic and logic functions.

- ALU is an integral part of central processing unit or CPU of a computer.

- It comes in various forms with wide range of functionality.

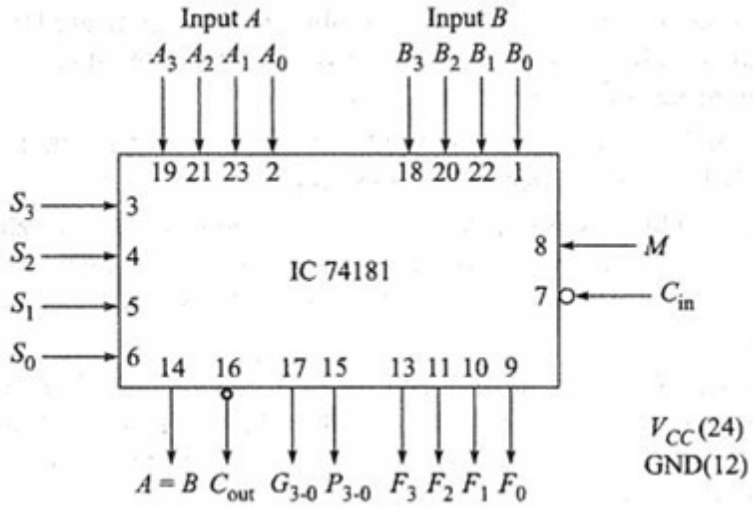
- It also comes with *PRESET* and *CLEAR* options.

- A mode selector input (M) decides whether ALU performs a logic operation or an arithmetic operation. In each mode different functions are chosen by appropriately activation a set of selection inputs.

1.21 Error Detection Codes

Error

- During data transmission, when the receiver does not receive the exact bits sent by the sender, then **error** is said to be occurred.



(a) Functional representation of ALU IC 74181.

S_3 S_2 S_1 S_0	$M = 1$ (Logic Function)	$M = 0$ (Arithmetic Function) $C_{in} = 1$ (For $C_{in} = 0$, add 1 to F)
0 0 0 0	$F = A'$	$F = A$
0 0 0 1	$F = (A+B)'$	$F = A + B$
0 0 1 0	$F = A'B$	$F = A + B'$
0 0 1 1	$F = 0$	$F =$ minus 1
0 1 0 0	$F = (AB)'$	$F = A$ plus (AB')
0 1 0 1	$F = B'$	$F = (A + B)$ plus (AB')
0 1 1 0	$F = A \oplus B$	$F = A$ minus B minus 1
0 1 1 1	$F = AB'$	$F = AB'$ minus 1
1 0 0 0	$F = A' + B$	$F = A$ plus (AB)
1 0 0 1	$F = (A \oplus B)'$	$F = A$ plus B
1 0 1 0	$F = B$	$F = (A + B')$ plus (AB)
1 0 1 1	$F = AB$	$F = AB$ minus 1
1 1 0 0	$F = 1$	$F = A$ plus A
1 1 0 1	$F = A + B'$	$F = (A + B)$ plus A
1 1 1 0	$F = A + B$	$F = (A + B')$ plus A
1 1 1 1	$F = A$	$F = A$ minus 1

(b)

(b) Its truth table.

Figure 1.55: Arithmetic Logic Unit.

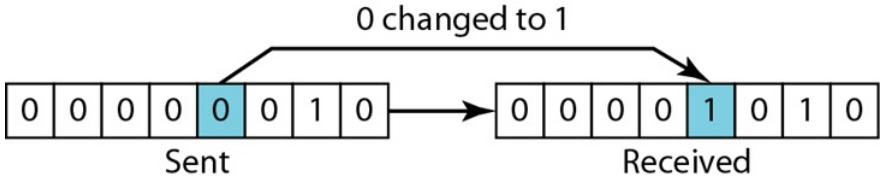
- Error is corruption of bit(s)during transmission because of noise, distortion or attenuation in the media.
- Due to error, some bits value in transmitted data change from 0 to 1 or 1 to 0.
- There are two types of error:

1. **Single Bit Error**

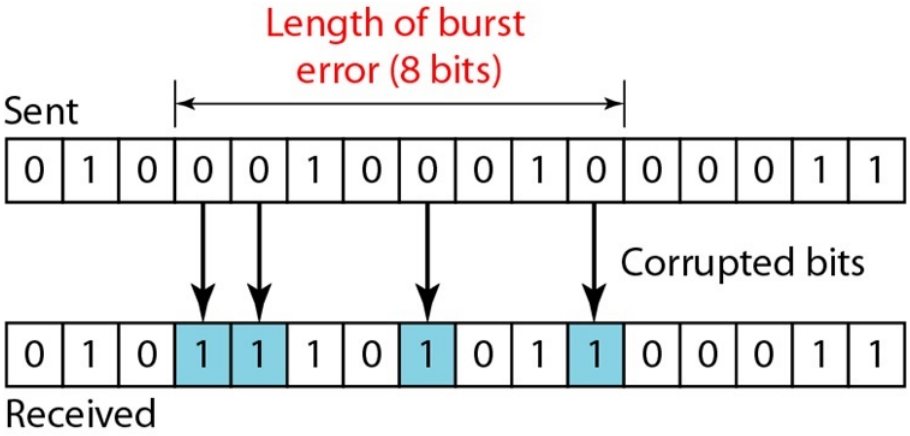
   If a single bit error, only 1 bit in the data has changed from 1 to 0 or 0 to 1. It is easy to detect and correct the single bit error.

2. **Burst Error**

   Burst error means that two or more bits in the data have changed form 0 to 1 or 1 to 0. It is easy to detect but much harder to correct the burst error when do occur.



(a) A single bit error



(b) A burst error.

Figure 1.56: Types of error

- Error control allows the receiver to inform the sender of any frames lost or duplicated or damaged in transmission and coordinates the retransmission of those frames by the sender.
- Error control is divided into two main categories:

1. **Error detection**

   - It allows a receiver to check whether received data has been corrupted during

transmission.

- If corrupted, it coordinates for retransmission.

- Example: Parity checking, CRC, checkum.

2. **Error Correction**

- It allows the receiver check error and to reconstruct the original information when it has been corrupted during transmission.

- Example: Hamming code

1.21.1 Parity Generator and Checker

- The most common error detection code used is the parity bit.

- A parity is an extra bit included with a binary message to make the total number of 1's either odd or even.

Table 1.12: Parity Bit Generation

Message	Parity	
xyz	*P(odd)*	*P(even)*
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

- From the table 1.12, for 3 bit message, we observe that the P(odd) bit is chosen to make the sum of all 1's (in all four bits) odd; while the P(even) bit is chosen to make the sum of all 1's even.

- During transfer of information from one location to another, the parity bit is handles as follows.

- At the sending side, the message (in this case three bits) is applied to the *parity generator*, where the required parity bit is generated.
- The message bit, including the parity bit, is transmitted to its destination.
- At the receiver end, all the incoming bit (in this case, four) are applied to a *parity checker* that checks the proper parity adopted (odd or even).
- An error is detected if the checked parity does not conform to the adopted parity.
- The parity method detects only any odd number of errors. An even number of errors in not detected.

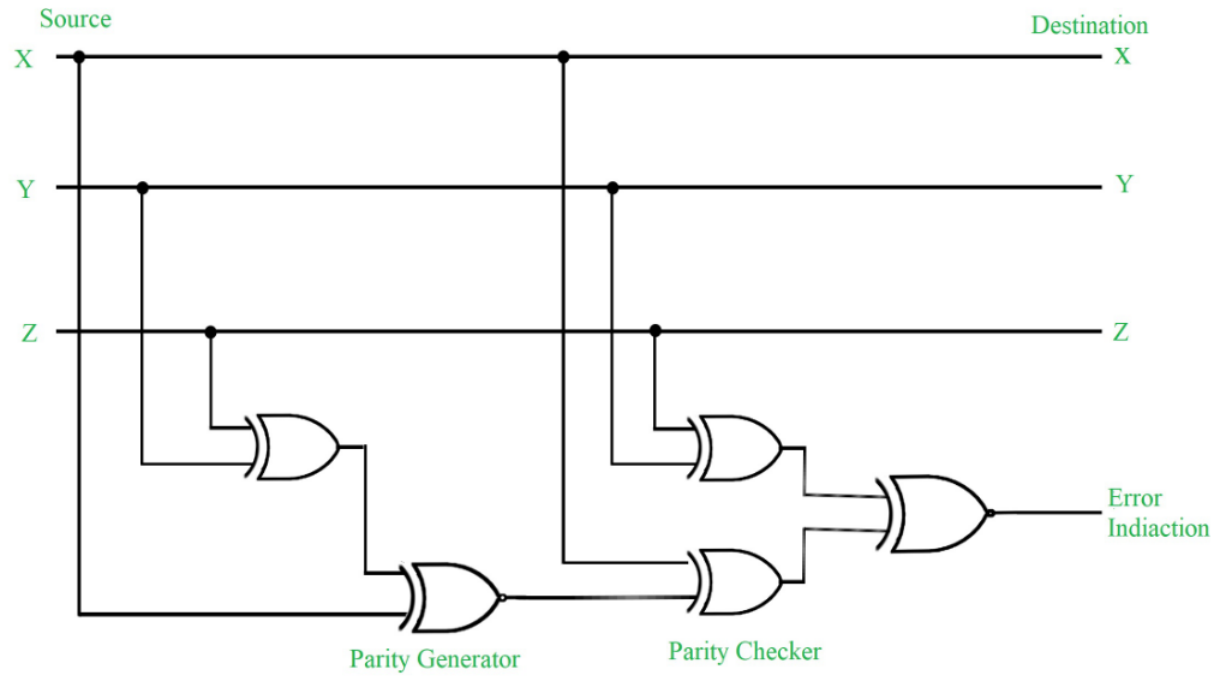


Figure 1.57: Error Detection with odd parity bit.

- Parity generator and checker are logic circuits constructed with exclusive-OR functions. Figure 1.57 is a logic circuit for 3 bit odd parity generator and checker. The output of parity checker is 1 when error occurs.

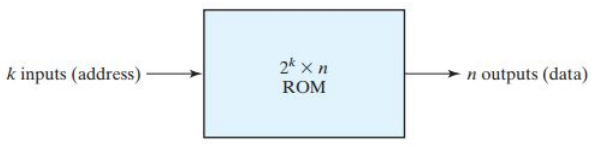


Figure 1.58: ROM block diagram.

1.22 Read-Only Memory

- A read-only memory (ROM) is an IC that can store thousands of binary numbers representing computer instructions and other fixed data.

- An example of fixed data is the unchanging information in a mathematical table.

- ROM uses "look-up" table to search needed data for computer system.

- Boolean function can be implemented by ROM.

1.22.1 Diode ROM

- Diode ROM uses the concept of *address* to store binary numbers.

- For instance, we want to store 0111 at address 0, 1000 at address 1, and so forth as shown in table 1.13. It is a 32-bit ROM organized as 8 words with 4 bits at each address (an 8 x 4 ROM).

Table 1.13: Diode ROM

Address	*Nibble*
0	0111
1	1000
2	1011
3	1100
4	0110
5	1001
6	0011
7	1110

- Referring to figure 1.59, when the switch is in position 0 (address 0), the upper row of diodes are conducting current (they act as closed switches). The output of the ROM is thus

$$Y_3Y_2Y_1Y_0 = 0111$$

- When the switch is moved to position 1, the second row is activated and

$$Y_3Y_2Y_1Y_0 = 1000$$

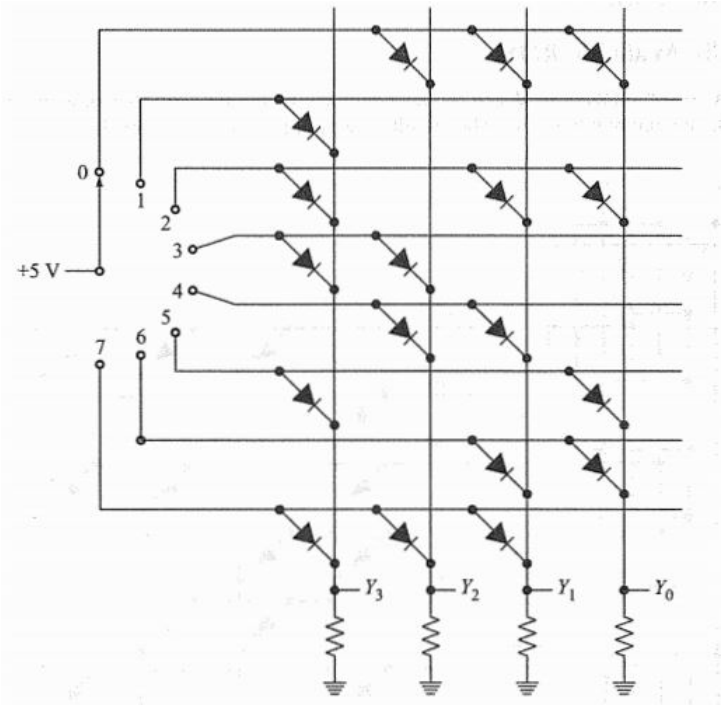


Figure 1.59: Diode ROM.

1.22.2 On-chip Decoding

- Rather than switch-select the address as shown in figure 1.59, a manufacture uses *on-chip decoding*.

- Figure 1.60 illustrates the idea behind it.

Some commercially available TTL ROMs are:

- 7488: 256 bits organized as 32x8
- 74187: 1024 bits organized as 256x4
- 74S370: 2048 bits organized as 512x4

1.22.3 Programmable ROMs

- The PROM is a combinational programmable logic device (PLD)—an integrated circuit with programmable gates divided into an AND array and an OR array to provide an AND–OR sum-of-product implementation.

- A programmable ROM (PROM) allows the user instead of the manufacturer to store the data. An instrument called a PROM programmer stores the words by "burning in".

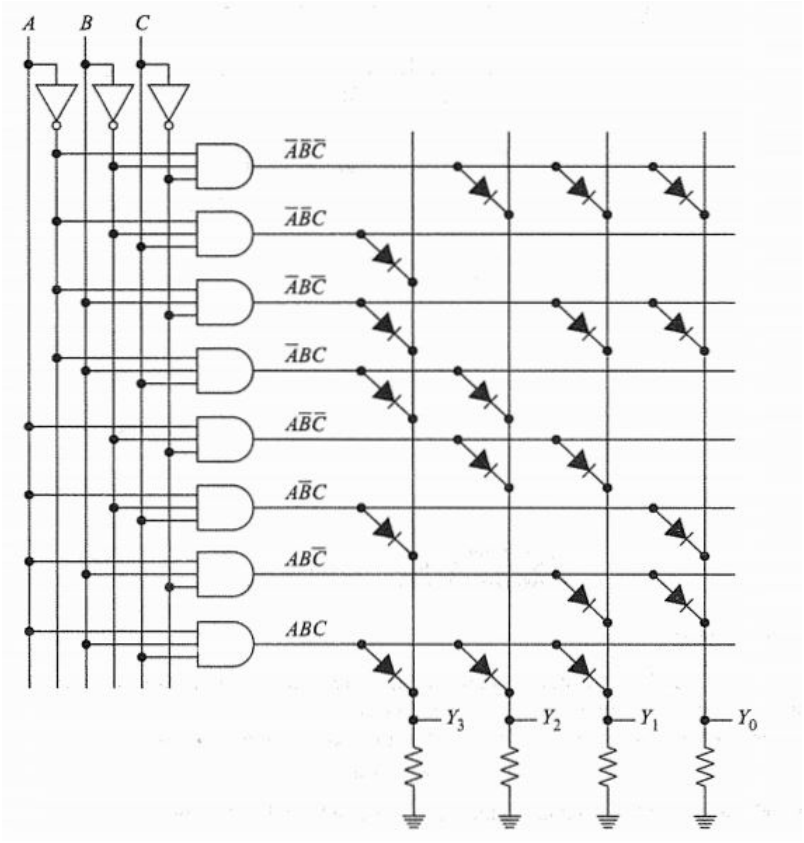


Figure 1.60: On-chip decoding.

- Originally, all diodes are connected at the cross points.

- For instance, in figure 1.59 there would be a total of 32 diodes (8 rows and 4 columns).

- Each of these diodes has a *fusible link* (a small fuse).

- The PROM programmer sends destructively high currents through all diodes to re removed. In this way, only the desired diodes remain connected after programming a PROM.

- Programming like this is permanent because the data cannot be erased after it has been burned in.

- Since PROMs are useful in many applications, manufacturers produce these chips in high volume. Furthermore, the PROM is a universal logic solution. Why? Because the AND gates generate all the fundamental products; the user can then OR these products as needed to generate any Boolean output. One disadvantage of PROMs is the limit on number of input variables; typically, PROMs have 8 inputs or less.

- Figure 1.61 shows a simplified PROM.

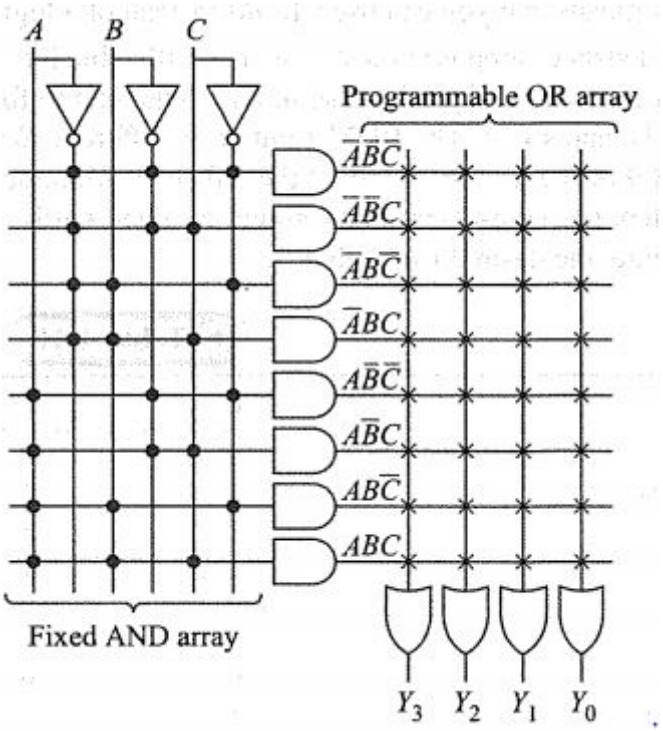


Figure 1.61: Streamlined drawing of PROM.

- In this simplified drawing,the solid black bullets indicate connections to the AND-gate inputs. Each bullet represents a *fixed* connection that cannot be changed.
- Furthermore, Each AND gate has 3 inputs, indicated by the bullet on its input line.
- Similarly, each OR gate has 8 inputs, as indicated by the x's on its input line, but each x is a fusible link that can be removed.

- The input side of PROM is a fixed AND array, that is the inputs to the AND gates are not programmable; on the other hand, the output side of the circuit is programmable because each connection at the input of each OR gate is a fusible link.
- A fixed AND array and a programmable OR array are characteristics of all PROMs.

Generating Boolean Function using PROM

- Generating a Boolean function at the output of a PROM is accomplished by fusing (melting) fusible link at the input of OR gates as shown in figure 1.62.

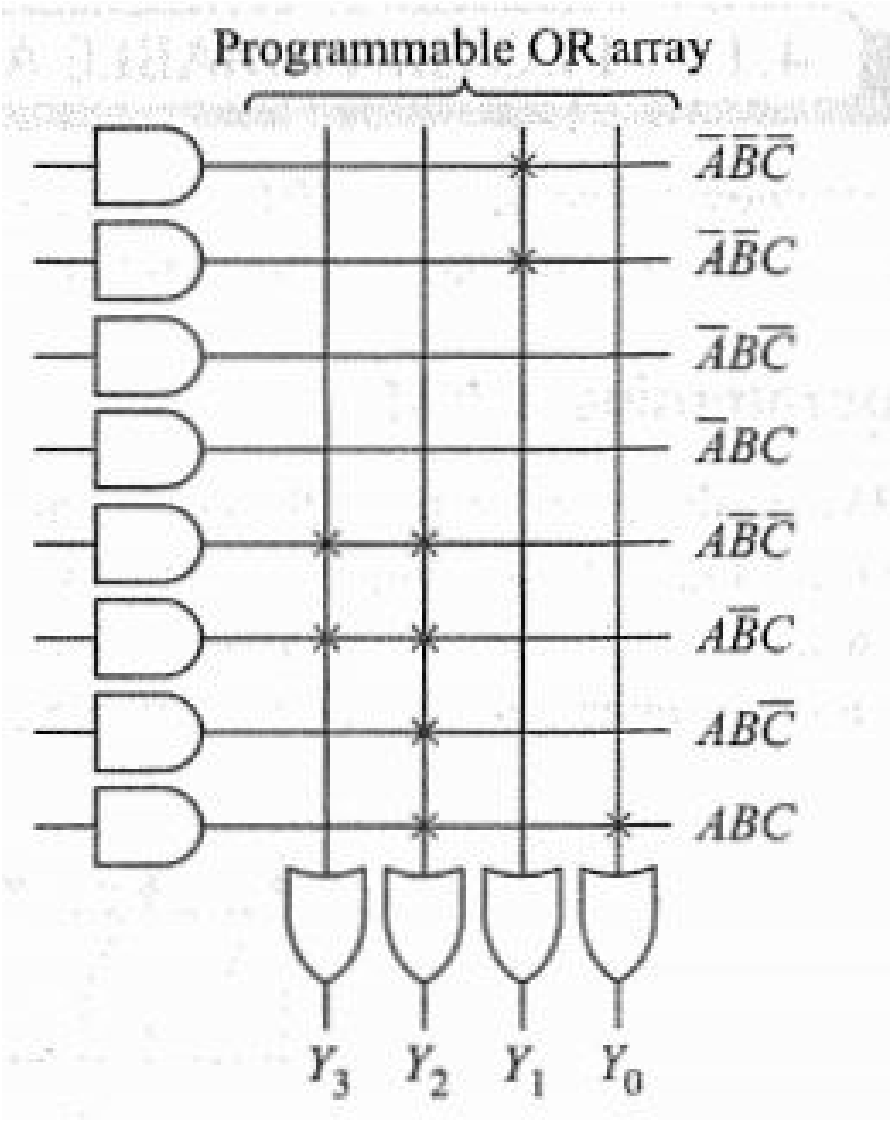


Figure 1.62: Boolean function from PROM.

- It generates:

$$Y_0 = ABC$$

$$Y_1 = \bar{A}\bar{C}$$

$$Y_2 = A$$

$$Y_3 = A\bar{B}$$

1.22.4 EPROM

- The *erasable PROM (EPROM)* uses metal-oxide-semiconductor field-effect transistors (MOSFETs).

- Data is stored with an EPROM programmer.
- Later, data can be erased with ultraviolet light.
- The light passes through a quartz window in the IC package.
- When it strikes the chip, the ultraviolet light releases all stored charges.
- The effect is to wipe out the stored contents.
- In other words, the EPROM is ultraviolet-light-erasable and electrically reprogrammable.

- The EPROM is useful in project development. With an ePROM, the designer can modify the contents until the stored data is pefect. When the design is finalized, the data can be burned into PROMs or sent to an IC manufacture who produces ROMs(large production runs).

1.22.5 Programmable Array Logic

- Programmable Array logic (PAL) has a programmable AND array and a fixed OR array.
- The AND gates are programmed to provide the product terms for the Boolean functions, which are logically summed in each OR gate.

- Figure 1.63 shows a PAL with 4 inputs and 4 outputs.
- The x's on the input side are fusible links, while the solid black bullets on the output side are fixed connection.
- With a PROM programmer, we can burn in the desired fundamental products which are the ORed by the fixed output connections.

Example: Suppose we want of generate the following Boolean functions:

$$Y_3 = \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD + ABC\bar{D}$$

$$Y_2 = \bar{A}BC\bar{D} + \bar{A}BCD + ABCD$$

$$Y_1 = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C}$$

$$Y_0 = ABCD$$

The solution to above implementation is given in figure 1.64.

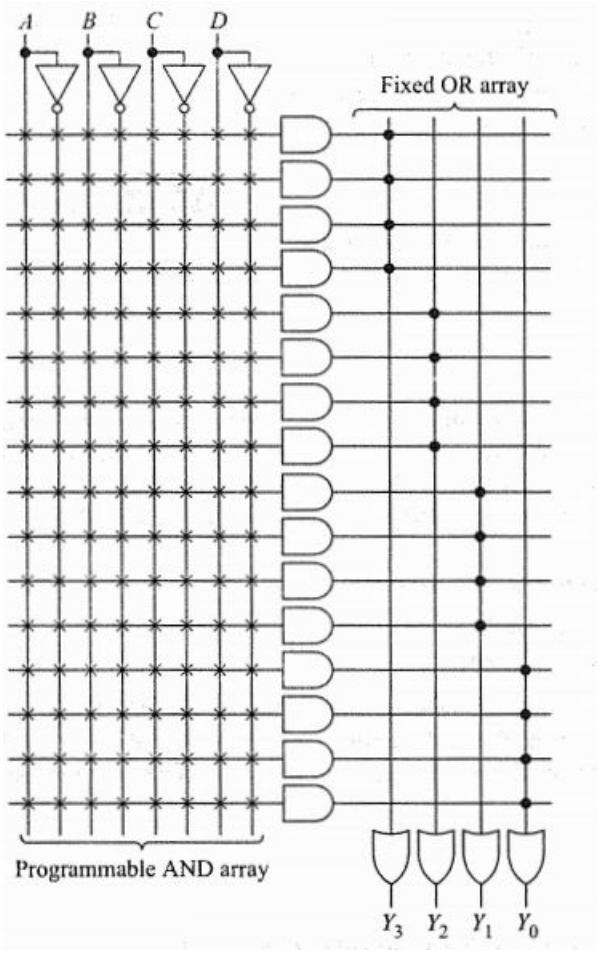


Figure 1.63: Structure of PAL with 4 input and 4 outputs.

Some TTL PALs are:

- 10H8: 10 input and 8 output AND-OR
- 16H2: 16 input and 2 output AND-OR
- 14L4: 4 input and 4 output AND-OR-INVERT

- Unlike PROMs, PALs are not a universal logic solution, because only some of the fundamental proucts can be generated an ORed at the final outputs.
- PALs have the advantage of 16 inputs compared to the typical limit of 8 inputs for PROMs.

1.22.6 Programmable Logic Array

- It the versatile PLD having both programmable AND array at input and programmable OR array at output.

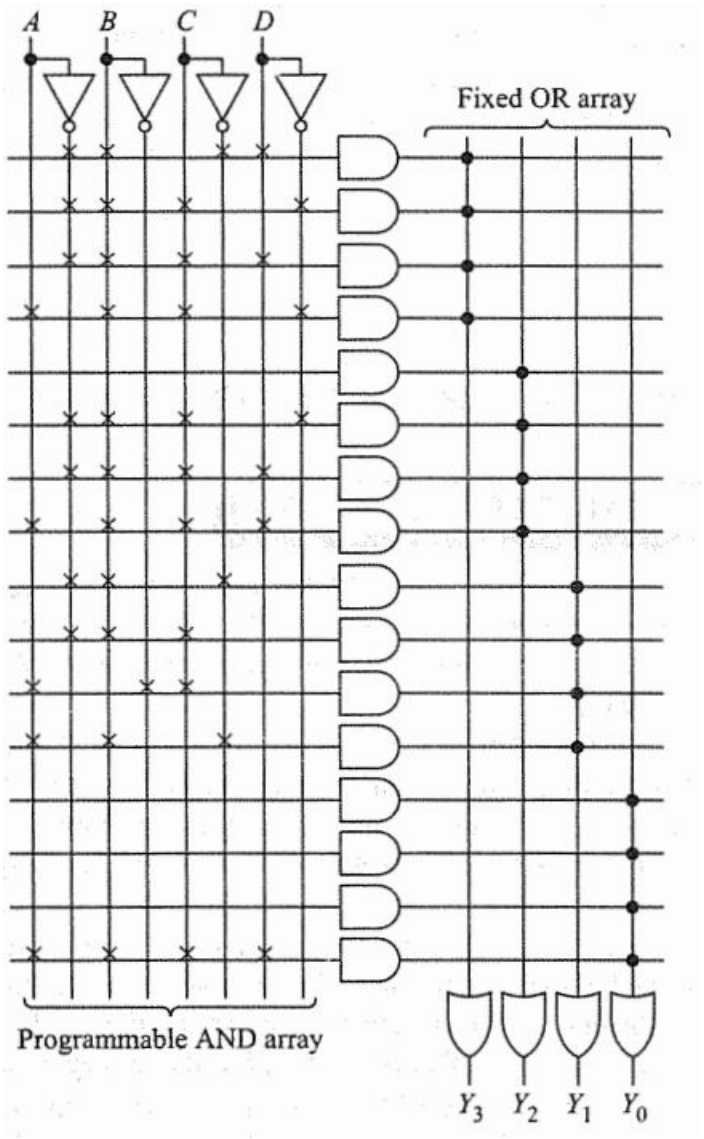


Figure 1.64: Example of PAL implementing Boolean functions.

- It is more complicated to utilize since the number of fusible links are doubled.
- Figure 1.65 shows a PLA having 3 input variables and 3 output variables.

Example: Using PLA to recognize 10 decimal digits represented by BCD and correctly drive a 7-segment display as shown in figure 1.66.

- For this, PLA must have 4 inputs, say A,B,C, and D, required to represent the 10 decimal numbers. And it must have 7 outputs (a,b,c,d,e,f,g), 1 output to drive of each of the 7 segments of the indicator.

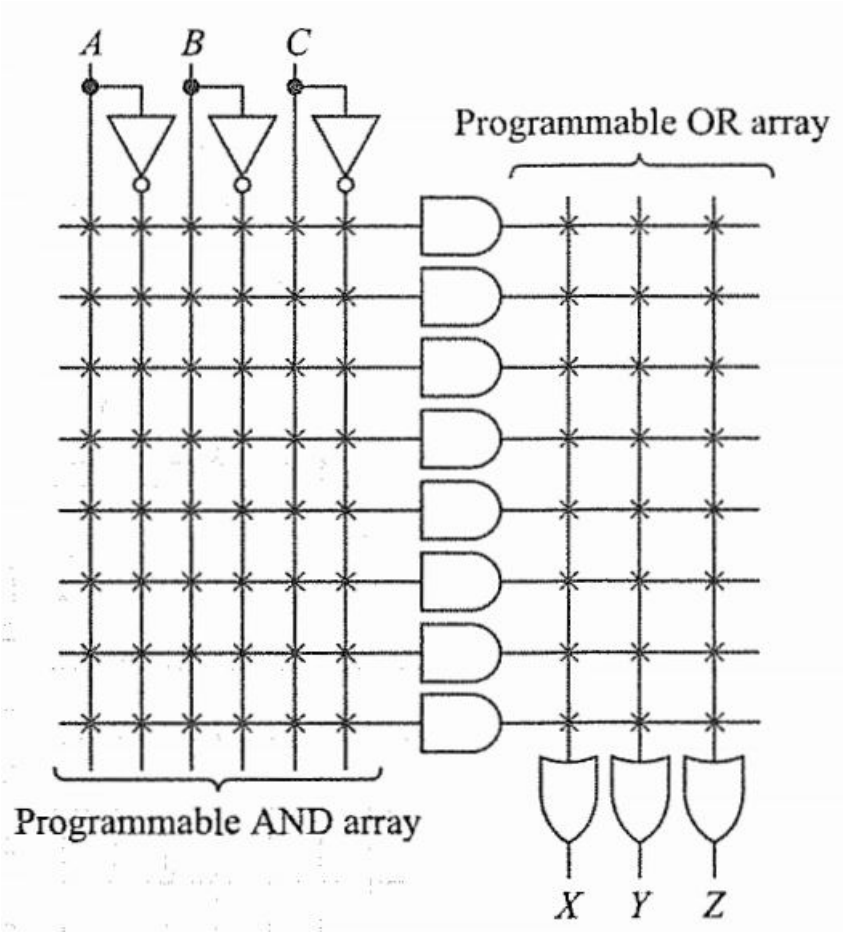


Figure 1.65: Structure of PLA for 3 inputs and 3 outputs.

1.22.7 Troubleshooting with a Logic Probe

- Logic probe is used for diagnosing faculty circuits.

- When you touch the probe tip on the output node of the logic gate, the device lights up for a hight state and goes dark for a low state.

- The probe is also useful for locating short circuits that occur in manufacturing. For example, during the stuffing and soldering of printed-circuit boards, an undesirable splash of solder may connect two adjacent traces (conducting lines). Known as a solder bridge, this kind of trouble can short circuit a node to the ground or to the supply voltage. The node is then stuck in a low or high state. The probe helps you to find short-circuited nodes because it stays in one state, no matter how the inputs are changing

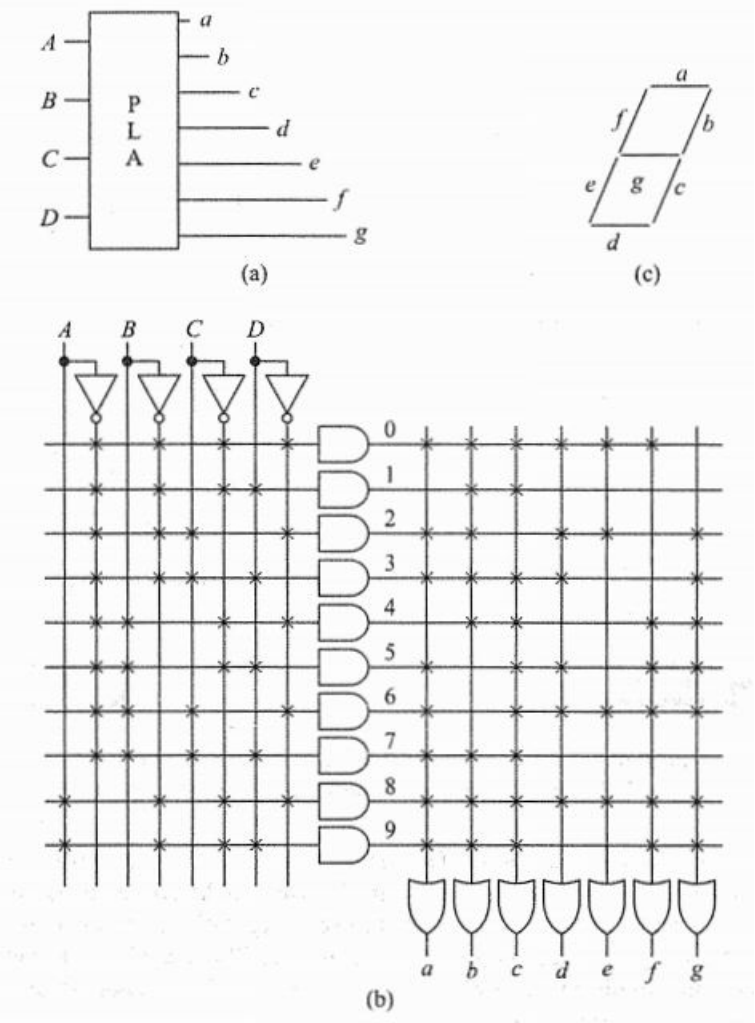


Figure 1.66: Structure of PLA for 3 inputs and 3 outputs.