Digital circuits and logical design

数字电路与逻辑设计

6 Functions of Combinational Logic

第六章　组合逻辑电路的功能模块

6-1 Basic Adders

6.1.1 the Half-Adder

A half-adder adds two bits and produces a sum and a carry output.

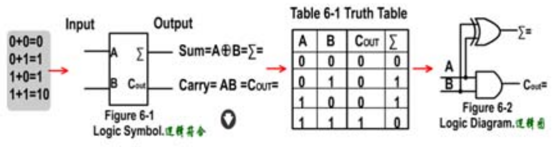


Table 6-1 Truth Table

A	B	COUT	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The operations are performed by a logic circuit called a half-adder.

The half-adder accepts two binary digits on its inputs and produces two binary digits on its outputs, a sum bit and a carry bit.

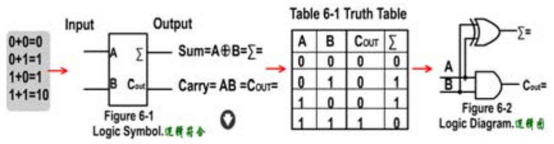


Table 6-1 Truth Table

A	B	COUT	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Notice that the output carry (Cout) is a 1 only when both A and B are 1s; therefore, Cout can be expressed as the AND of the input variables.

$$C_{OUT} = AB$$

Now observe that the sum output(∑) is a 1 only if the input variables, A and B, are not equal. The sum can therefore be expressed as the exclusive-OR of the input variables.

$$\Sigma = A\overline{B} + \overline{A}B = A \oplus B$$

6.1.2 the Full-Adder

The basic difference between a full-adder and a half-adder is that the full-adder accept an input carry. A logic symbol for a full-adder is shown in Figure 6-3, and the truth table in Table 6-2 shows the operation of a full-adder.

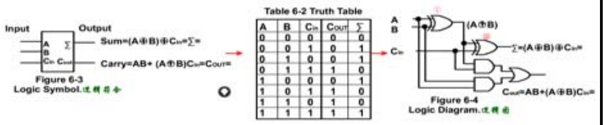
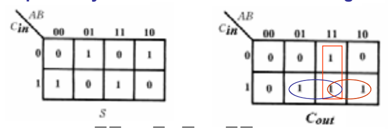


Table 6-2 Truth Table

A	B	Cin	COUT	Σ
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

map directly from a truth table to a Karnaugh map.



	Cin \ AB	00	01	11	10
S	0	0	1	0	1
S	1	1	0	1	0
Cout	0	0	0	1	0
Cout	1	0	1	1	1

$$\Sigma = S = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + ABC$$
$$= (\overline{A}\,\overline{B} + AB)C + (\overline{A}B + A\overline{B})\overline{C}$$
$$= \overline{(A\oplus B)}C + (A\oplus B)\overline{C} = (A\oplus B)\oplus C$$
$$= (A\oplus B)\oplus C$$

$$C_{OUT} = AB + AC_{in} + BC_{in}$$
$$= AB + A\overline{B}C_{in} + \overline{A}BC_{in}$$
$$= AB + (A\overline{B} + \overline{A}B)C_{in}$$
$$= AB + (A\oplus B)C_{in}$$

INPUT			OUTPUT	
Cin	A	B	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

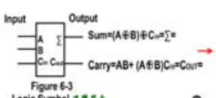
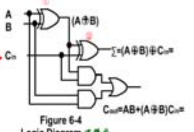

Input
Output
A
B
C_{in}
$\sum$
C_{out}
Sum=$(A\oplus B)\oplus C_{in}=\sum$=
Carry=$AB+(A\oplus B)C_{in}=C_{OUT}$=
Figure 6-3
Logic Symbol.

Table 6-2 Truth Table

A	B	C_{in}	C_{OUT}	$\sum$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1


A
B
C_{in}
$(A\oplus B)$
$\sum=(A\oplus B)\oplus C_{in}$=
$C_{out}=AB+(A\oplus B)C_{in}$=
Figure 6-4
Logic Diagram.

Notice in Figure 6-4 there are two half-adders, connected as shown in the black diagram of Figure (a), with their output carries ORed. The logic symbol shown in Figure (b) will normally be used to represent the full-adder.

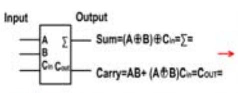

Input
Output
A
B
C_{in}
$\sum$
C_{out}
Sum=$(A\oplus B)\oplus C_{in}=\sum$=
Carry=$AB+(A\oplus B)C_{in}=C_{OUT}$=
Figure (b) Full-adder logic symbol

Table 6-2 Truth Table

A	B	C_{in}	C_{OUT}	$\sum$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

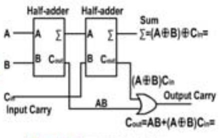


Figure (a) Arrangement of two half-adders to form a full-adder.

Example

For each of the three full-adders in Figure 6-6, determine the outputs for the inputs shown.

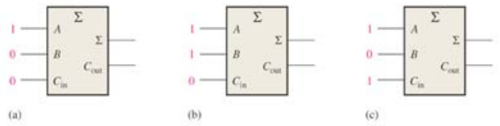


Figure 6-6

Solution

(a) The input bits are A=1,B=0, and Cin=0. 1+0+0=1 with no carry. Therefore, ∑=1 and Cout=0.

(b) The input bits are A=1,B=1, and Cin=0. 1+1+0=0 with a carry of 1. Therefore, ∑=0 and Cout=1.

(c) The input bits are A=1,B=0, and Cin=1. 1+0+1=0 with a carry of 1. Therefore, ∑=0 and Cout=1.

6-2 Parallel Binary Adders

A single full-adder is capable of adding two 1-bit numbers and an input carry. To add binary numbers with more than one bit, additional full-adders must be used. When one binary number is added to another, each column generates a sum bit and a 1 or 0 carry bit to the next column to the left, as illustrated here with 2-bit numbers.

In this case, the carry bit from second column becomes a sum bit.
此例中，从第二栏来的进位成为总和位。

Carry bit from right column
从右边来的进位

```
  1
  1 1
+ 0 1
1 0 0
```

To add two binary numbers, a full-adder is required for each bit in the numbers. So for 2-bit numbers, two adders are needed; for four-bit numbers, four adders are used; and so on.

The carry output of each adder is connected to the carry input of the next higher-order adder, as shown in Figure 6-7 for a 2-bit adder. Notice that either a half-adder can be used for the least significant position or the carry input of a full-adder can be made 0 (grounded) because there is no carry input to the least significant bit position.

Figure 6-7 Block diagram of a basic 2-bit paralel adder using two full-adder.
图6-7 基本二位并行加法器的框图。

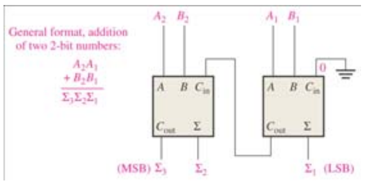


Figure 6-7

In Figure 6-7 the least significant bits(LSB) of the two numbers are represented by A1 and B1. The next higher-order bits are represented by A2 and B2. The three sum bits are ∑1, ∑2 and ∑3. Notice that the output carry from the left-most full-adder becomes the most significant bit(MSB) in the sum,∑3.

Example

Determine the sum generated by the 3-bit parallel adder in Figure 6-8 and shown the intermediate carries when the binary numbers 101 and 011 are being added.

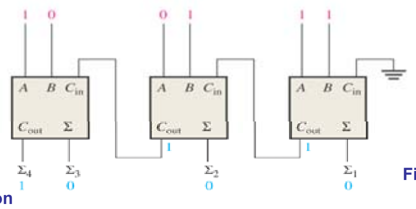


Figure 6-8

Solution

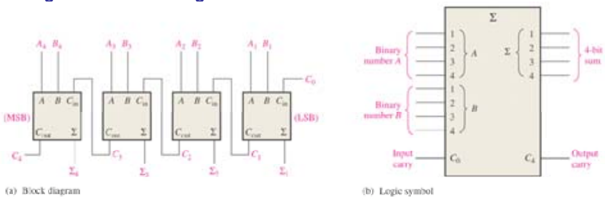
The LSBs of the two numbers are added in the right-most full-adder. The sum bits and the intermediate carries are indicated in blue in Figure 6-8.

What are the sum outputs when 111 and 101 are added by the 3-bit parallel adder?

Four-Bit Parallel Adder 四位并行加法器

A group of four bits is called a nibble.

A basic 4-bit parallel adder is implemented with four full-adder stages as shown in Figure 6-9.



In keeping with most manufactures' data sheets, the input labeled Co is the input carry to the least significant bit adder; C4 in the case of four bits, is the output carry of the most significant bit adder; and ∑1 (LSB) through ∑4 (MSB) are the sum outputs. The logic symbol is shown in Figure 6-9(b).

Example

Use the 4-bit parallel adder truth table to find the sum and output carry for the addition of the following two 4-bit numbers if the input carry(Cn-1) is 0:

A4A3A2A1=1100 and B4B3B2B1=1100

Solution

For n=1: A1=0, B1=0, and Cn-1=0.
From the 1st row of the table,
∑=0 and C1=0

For n=2: A2=0, B2=0, and Cn-1=0.
From the 1st row of the table,
∑=0 and C2=0

For n=3: A3=1, B3=1, and Cn-1=0.
From the 4th row of the table,
∑=0 and C3=1

For n=4: A4=1, B4=1, and Cn-1=1.
From the last row of the table,
∑=1 and C4=1

the table for each stage of a 4-bit parallel adder

C_{n-1}	A_n	B_n	∑	C_n
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

C4 becomes the output carry; the sum of 1100 and 1100 is 11000.

6.3 RIPPLE CARRY VERSUS LOOK-AHEAD CARRY ADDERS

In terms of the method used to handle carries in a parallel adder, there are two types:

the ripple carry adder and the carry look-ahead adder.

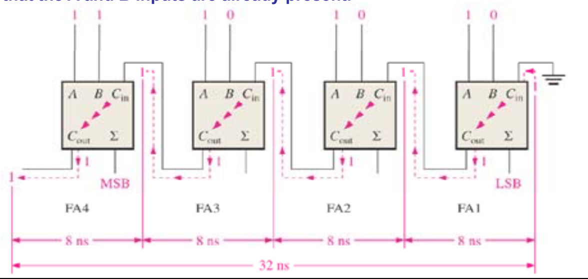
6.3.1 The Ripple Carry Adders 行波进位加法器（串行进位加法器）

A ripple carry adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder). The sum and the output carry of any stage can not be produced until the input carry occurs; this causes a time delay in the addition process.

The carry propagation delay for each full-adder is the time from the application of the input carry until the output carry occurs, assuming that the A and B inputs are already present.

A ripple carry adder is one in which the carry output of each full-adder is connected to the carry input of the next higher-order stage (a stage is one full-adder). The sum and the output carry of any stage can not be produced until the input carry occurs; this causes a time delay in the addition process.

The carry propagation delay for each full-adder is the time from the application of the input carry until the output carry occurs, assuming that the A and B inputs are already present.



6.3.2 The Look-Ahead Carry Adders 超前进位加法器

A method of speeding up the addition process by eliminating this ripple carry delay is called look-ahead carry addition. The look-ahead carry adder anticipates the output carry of each stage, and based on the input bits of each stage, produces the output carry by either carry generation or carry propagation.

Carry generation 进位生成

Carry generation occurs when an output carry is produced (generated) internally by the full-adder. A carry is generated only when both input bits are 1s . The generated carry, Cg, is expressed as the AND function of the two input bits. Aand B.

Cg=AB

Carry propagation 进位传送

Carry propagation occurs when the input carry is rippled to become the input carry. An input carry may be propagated by the full-adder when either or both of the input bits are 1s. The propagated carry, Cp, is expressed as the OR function of the input bits.

Cp=A+B

Introduction: To eliminating the time delay of the carry propagation.
Method: The input carry $(CI)_i$ can be expressed by $A_{i-1}\ A_{i-2}\ldots A_0$ and $B_{i-1}\ B_{i-2}\ldots B_0$ exclusively.

The addition for two numbers will produce the carry output $(CO)_i$

$$(CO)_i = A_iB_i + (A_i + B_i)(CI)_i$$

Suppose we define the generation function as $A_iB_i=G_i$ and the carry transfer function as $(A_i+B_i)=P_i$ then we get

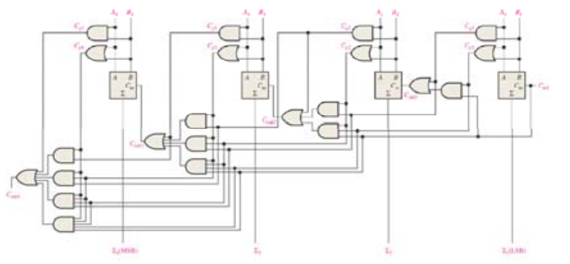
$$\begin{aligned}(CO)_i &= G_i + P_i(CI)_i\\ &= G_i + P_i[G_{i-1} + P_{i-1}(CI)_{i-1}]\\ &= G_i + P_iG_{i-1} + P_iP_{i-1}[G_{i-2} + P_{i-2}(CI)_{i-2}]\\ &\ \ \vdots\\ &= G_i + P_iG_{i-1} + P_iP_{i-1}G_{i-2} + \cdots + P_iP_{i-1}\cdots P_1G_0 + P_iP_{i-1}\cdots P_0C_0 \qquad (1)\end{aligned}$$

Recurrent method
递推方法

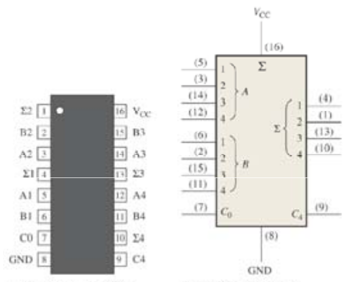
$$\Sigma_i = A_i \oplus B_i \oplus (CI)_i \qquad (2)$$

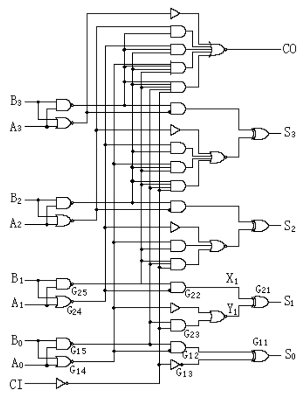
The Cout equations are implemented with logic gates and connected to the full-adders to create a 4-bit look-ahead carry adder.

The Cout equations are implemented with logic gates and connected to the full-adders to create a 4-bit look-ahead carry adder.



The 74LS283 4-bit adder is a look-head carry adder.


(a) Pin diagram of 74LS283 (b) 74LS283 logic symbol



$$X_1 = \overline{(A_1 B_1)}\,(A_1 + B_1) = A_1 \oplus B_1$$

$$Y_1 = \overline{\overline{A_0 + B_0} + \overline{(CI)_0}\;\overline{A_0 B_0}}$$

$$= A_0 B_0 + (A_0 + B_0)(CI)_0$$

$$= G_0 + P_0 (CI)_0 = (CO)_0 = (CI)_1$$

$$S_1 = X_1 \oplus Y_1$$

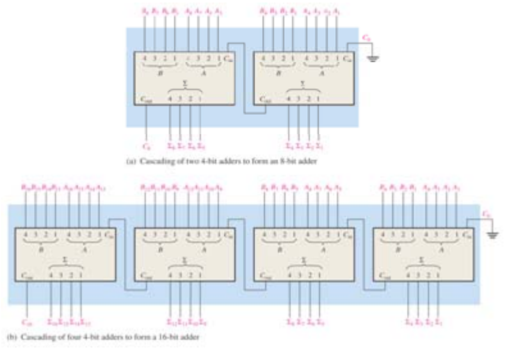
$$= A_1 \oplus B_1 \oplus (CI)_1$$

Symbol	Parameter	Limits			Unit
		Min	Typ	Max	
t_{PLH} t_{PHL}	Propagation delay, C_0 input to any Σ output		16 15	24 24	ns
t_{PLH} t_{PHL}	Propagation delay, any A or B input to Σ outputs		15 15	24 24	ns
t_{PLH} t_{PHL}	Propagation delay, C_0 input to C_4 output		11 11	17 22	ns
t_{PLH} t_{PHL}	Propagation delay, any A or B input to C_4 output		11 12	17 17	ns

6.3.3 Combination Look-ahead and Ripple Carry Adders

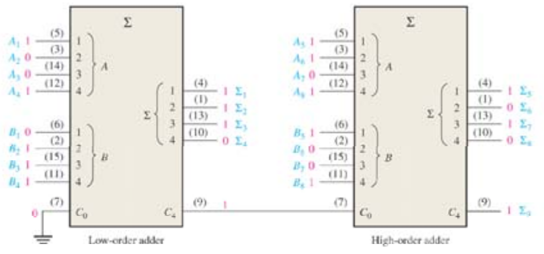
Adder Expansion

Adders can be Expanded to handle more bits by cascading.


(a) Cascading of two 4-bit adders to form an 8-bit adder

(b) Cascading of four 4-bit adders to form a 16-bit adder

Combination Look-ahead and Ripple Carry Adders



Two 74LS283 adders connected as an 8-bit parallel adder (pin numbers are in parentheses).

Example

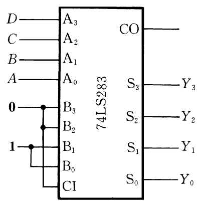
Design a cuitcuit that can change 8421BCD code into excess-three code.（设计一个电路，将8421 BCD码转成余3码。）

Solution

We let 4-bit BCD code to be DCBA , then DCBA and 0011 as the inputs, Finally we get the output $Y_3Y_2Y_1Y_0$.

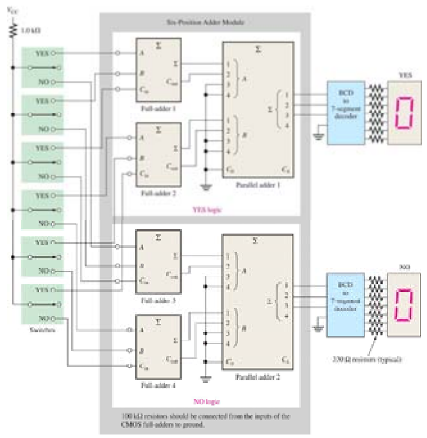
Hence $Y_3Y_2Y_1Y_0$=DCBA+0011

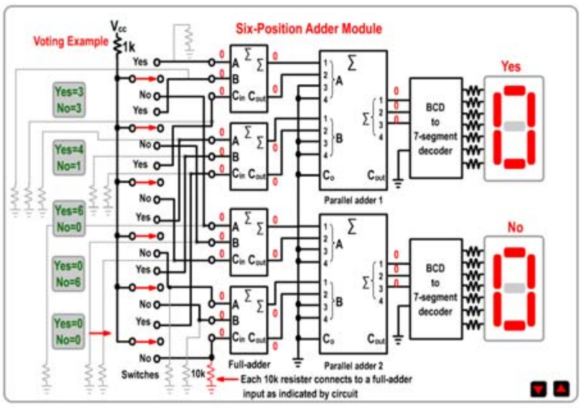
The logic circuit will be as follows:



6.2.4 An Application

An example of full-adder and parallel adder application is a simple voting system that can be used to simultaneously provide the number of "yes" votes and the number of "no" votes.


Six-Position Adder Module
Full-adder 1
Full-adder 2
Full-adder 3
Full-adder 4
Parallel adder 1
Parallel adder 2
BCD to 7-segment decoder
YES
NO
YES logic
NO logic
Switches
220 Ω resistors (typical)
100 kΩ resistors should be connected from the inputs of the CMOS full-adders to ground.

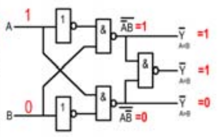


6-4 Comparators

The basic function of a comparator is to compare the magnitude of two binary quantities to determine the relationship of those quantities. In its simplest form, a comparator circuit determines whether two numbers are equal.

1-Bit Magnitude Comparator

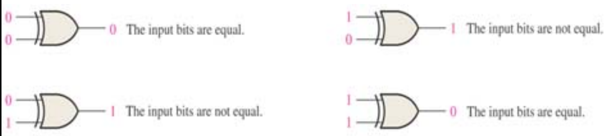
A>B	A=1	B=0	$A\overline{B}=1$
A<B	A=0	B=1	$\overline{A}B=1$
A=B	A=0	B=0	$A\oplus B=1$
A=B	A=1	B=1	



6.4.1 Equality

The exclusive-OR gate can be used as a basic comparator because its output is a 1 if the two inputs bits are not equal and a 0 if the input bits are equal. Figure 6-19 shows the exclusive-OR gate as a basic comparator.

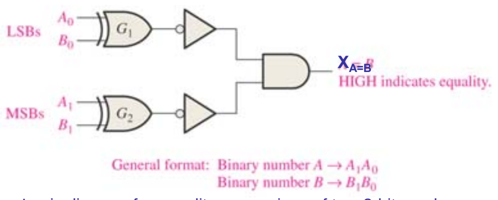
the exclusive-OR gate as a basic comparator:


The input bits are equal.
The input bits are not equal.
The input bits are not equal.
The input bits are equal.

Basic comparator operation.

In order to compare binary numbers containing two bits each, an additional exclusive-OR gate is necessary.

$$
\begin{aligned}
X_{A=B} &= \overline{A}_0\overline{B}_0\overline{A}_1\overline{B}_1 + \overline{A}_0\overline{B}_0A_1B_1 + A_0B_0\overline{A}_1\overline{B}_1 + A_0B_0A_1B_1 \\
&= \overline{A}_0\overline{B}_0(\overline{A}_1\overline{B}_1 + A_1B_1) + A_0B_0(\overline{A}_1\overline{B}_1 + A_1B_1) \\
&= (\overline{A}_0\overline{B}_0 + A_0B_0)(\overline{A}_1\overline{B}_1 + A_1B_1) \\
&= (\overline{\overline{A}_0B_0 + A_0\overline{B}_0})(\overline{\overline{A}_1B_1 + A_1\overline{B}_1}) \\
&= \overline{A_0 \oplus B_0}\,\overline{A_1 \oplus B_1}
\end{aligned}
$$


LSBs
A_0
B_0
G_1
MSBs
A_1
B_1
G_2
$X_{A=B}$
HIGH indicates equality.
General format: Binary number $A \rightarrow A_1A_0$
Binary number $B \rightarrow B_1B_0$

Logic diagram for equality comparison of two 2-bit numbers.

Example

Apply each of the following sets of binary numbers to the comparator inputs in Figure 6-21, and determine the output by following the logic levels through the circuit.

(a) 10 and 10 (b)11 and 10

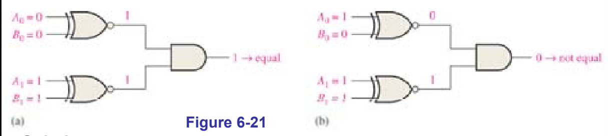


Figure 6-21

Solution

(a) The output is 1 for input 10 and 10, as shown in Figure 6-21(a).

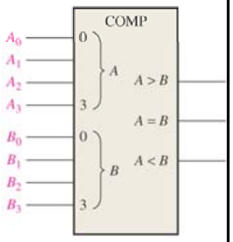
(b) The output is 0 for input 11 and 10, as shown in Figure 6-21(b).

As you know from Chapter3, the basic comparator can be expanded to any number of bits. The AND gate sets the condition that all corresponding bits of the two numbers must be equal if the two numbers themselves are equal.

6.4.2 Inequality 不相等

In addition to the equality output, many IC comparators provide additional outputs that indicate which of the two binary numbers being compared is the larger. That is, there is an output that indicates when number A is greater than number B (A>B) and an output that indicates when number A is less than number B(A<B), as shown in the logic symbol for a 4-bit comparator in the following Figure.

following				OUTPUT
A3 B3	A2 B2	A1 B1	A0 B0	A>B A<B A=B
A3>B3	x x	x x	x x	1 0 0
A3<B3	x x	x x	x x	0 1 0
A3=B3	A2>B2	x x	x x	1 0 0
A3=B3	A2<B2	x x	x x	0 1 0
A3=B3	A2=B2	A1>B1	x x	1 0 0
A3=B3	A2=B2	A1<B1	x x	0 1 0
A3=B3	A2=B2	A1=B1	A0>B0	1 0 0
A3=B3	A2=B2	A1=B1	A0<B0	0 1 0
A3=B3	A2=B2	A1=B1	A0=B0	0 0 1

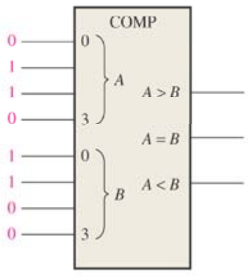


Example

Determine the A=B, A>B, A<B outputs for the input numbers shown on the comparator in the following Figur.

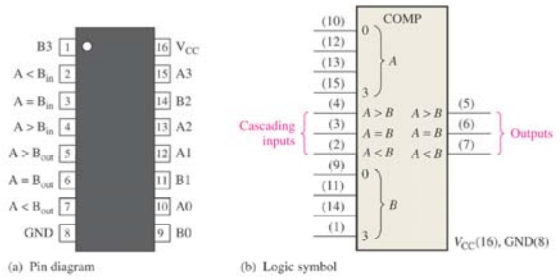
Solution

The number on the A inputs is 0110 and the number on the B input is 0011. Then A>B output is HIGH and the other outputs are LOW.



What are the comparator outputs when $A_3A_2A_1A_0$=1001 and $B_3B_2B_1B_0$=1010?

The 74HC85 4-bit Magnitude Comparator (74HC85 4位数字大小比较器)



Pin diagram and logic symbol for the 74HC85 4-bit magnitude comparator

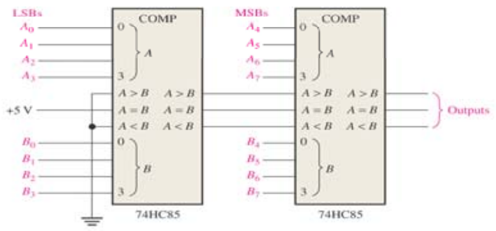
INPUT					OUTPUT
A3 B3	A2 B2	A1 B1	A0 B0	$A>B_{in}$ $A<B_{in}$ $A=B_{in}$	A>B A<B A=B
A3>B3	x x	x x	x x	x x x	1 0 0
A3<B3	x x	x x	x x	x x x	0 1 0
A3=B3	A2>B2	x x	x x	x x x	1 0 0
A3=B3	A2<B2	x x	x x	x x x	0 1 0
A3=B3	A2=B2	A1>B1	x x	x x x	1 0 0
A3=B3	A2=B2	A1<B1	x x	x x x	0 1 0
A3=B3	A2=B2	A1=B1	A0>B0	x x x	1 0 0
A3=B3	A2=B2	A1=B1	A0<B0	x x x	0 1 0
A3=B3	A2=B2	A1=B1	A0=B0	1 0 0	1 0 0
A3=B3	A2=B2	A1=B1	A0=B0	0 1 0	0 1 0
A3=B3	A2=B2	A1=B1	A0=B0	0 0 1	0 0 1

Example

Use 74HC85 comparators to compare the magnitudes of two 8-bit numbers. show the comparators with proper interconnections.

Solution

Two 74HC85s are required to compare two 8-bit numbers. They are connected as shown in the following Figure, in a cascaded arrangement.



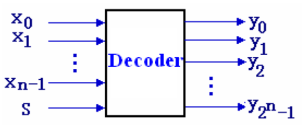
An 8-bit magnitude comparator using two 74HC85s.

Related Problem: Expand the circuit 6-25 to a 16-bit comparator.

6-5 Decoders 译码器

The basic function of a decoder is to detect the presence of a specified combination of bits (code) on its inputs and to indicate the presence of that code by a specified output level.

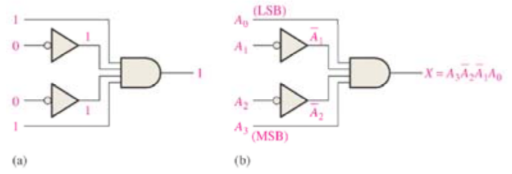
In its general form, a decoder has n input lines to handle n bits and from one to 2^n output lines to indicate the presence of one or more n-bit combination.



In this section, several decoders are introduced. The basic principles can be extended to other type of decoders.

6.5.1 The basic Binary Decoder 基本二进制译码器

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 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 the following Figure.



Decoding logic for the binary code 1001 with an active-HIGH output

Example

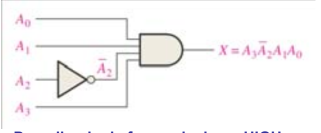
Determine the logic required to decode the binary number 1011 by producing a HIGH level on the output.

Solution

The decoding function can be formed by complementing only the variables that appear as 0 in the desired binary number, as follows:

$$X = A_3\overline{A}_2A_1A_0$$

The decoding logic is shown in the following Figure.



Decoding logic for producing a HIGH output when 1011 is on the inputs.

Related Problem: Develop the logic required to detect the binary code 10010 and produce an active-LOW output.

6.5.2 The 3-Bit Decoder 3位译码器

In order to decode all possible combinations of three bits, eight decoding gates are required(2^3=8). This type of decoder is commonly called a 3-line-to-8-line decoder because there are three inputs and eight outputs or a 1-of-8 decoder because for any given code on the inputs, one of the eight outputs is activated.

Decoding functions and truth table for the 74ls138 3-line- to-8- line (1 of 8) decoder with active LOW outputs

INPUT					OUTPUT							
S_1	$\overline{S}_2+\overline{S}_3$	A_2	A_1	A_0	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$
0	×	×	×	×	1	1	1	1	1	1	1	1
×	1	×	×	×	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0

When $S_1 = 1, \overline{S}_2 = 0, \overline{S}_3 = 0$

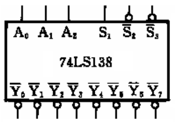
$$\overline{Y}_0 = \overline{\overline{A}_2\overline{A}_1\overline{A}_0} = \overline{m_0}$$
$$\overline{Y}_1 = \overline{\overline{A}_2\overline{A}_1A_0} = \overline{m_1}$$
$$\overline{Y}_2 = \overline{\overline{A}_2A_1\overline{A}_0} = \overline{m_2}$$
$$\overline{Y}_3 = \overline{\overline{A}_2A_1A_0} = \overline{m_3}$$
$$\overline{Y}_4 = \overline{A_2\overline{A}_1\overline{A}_0} = \overline{m_4}$$
$$\overline{Y}_5 = \overline{A_2\overline{A}_1A_0} = \overline{m_5}$$
$$\overline{Y}_6 = \overline{A_2A_1\overline{A}_0} = \overline{m_6}$$
$$\overline{Y}_7 = \overline{A_2A_1A_0} = \overline{m_7}$$

Decoding functions and truth table for the 74ls138 3-line- to-8- line (1 of 8) decoder with active LOW outputs

INPUT					OUTPUT							
S_1	$\overline{S}_2+\overline{S}_3$	A_2	A_1	A_0	$\overline{Y}_0$	$\overline{Y}_1$	$\overline{Y}_2$	$\overline{Y}_3$	$\overline{Y}_4$	$\overline{Y}_5$	$\overline{Y}_6$	$\overline{Y}_7$
0	×	×	×	×	1	1	1	1	1	1	1	1
×	1	×	×	×	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0

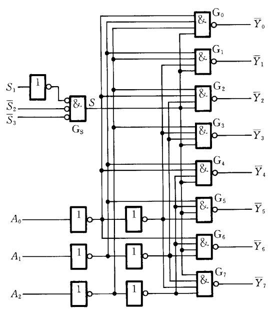
When $S_1 = 1, \overline{S}_2 = 0, \overline{S}_3 = 0$:

$$\overline{Y}_0 = \overline{\overline{A}_2\overline{A}_1\overline{A}_0} = \overline{m_0} \qquad \overline{Y}_4 = \overline{A_2\overline{A}_1\overline{A}_0} = \overline{m_4}$$
$$\overline{Y}_1 = \overline{\overline{A}_2\overline{A}_1A_0} = \overline{m_1} \qquad \overline{Y}_5 = \overline{A_2\overline{A}_1A_0} = \overline{m_5}$$
$$\overline{Y}_2 = \overline{\overline{A}_2A_1\overline{A}_0} = \overline{m_2} \qquad \overline{Y}_6 = \overline{A_2A_1\overline{A}_0} = \overline{m_6}$$
$$\overline{Y}_3 = \overline{\overline{A}_2A_1A_0} = \overline{m_3} \qquad \overline{Y}_7 = \overline{A_2A_1A_0} = \overline{m_7}$$

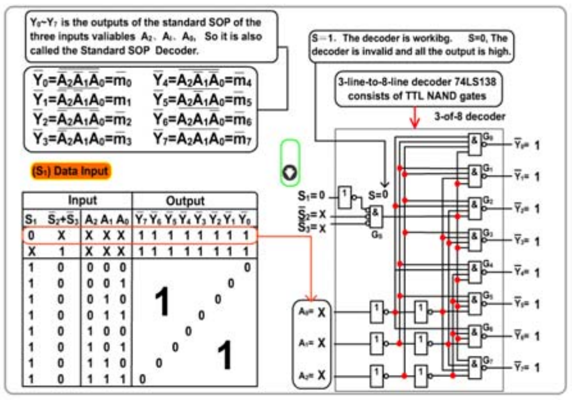


A_0 A_1 A_2 S_1 $\overline{S}_2$ $\overline{S}_3$
74LS138
$\overline{Y}_0$ $\overline{Y}_1$ $\overline{Y}_2$ $\overline{Y}_3$ $\overline{Y}_4$ $\overline{Y}_5$ $\overline{Y}_6$ $\overline{Y}_7$

Logic symbol for the 74LS138



Logic diagram for the 74LS138



Y0~Y7 is the outputs of the standard SOP of the three inputs valiables A2、A1、A0, So it is also called the Standard SOP Decoder.
S=1. The decoder is workibg. S=0, The decoder is invalid and all the output is high.
3-line-to-8-line decoder 74LS138 consists of TTL NAND gates
3-of-8 decoder
(S1) Data Input
S1=0
S=0
G0
A0= X
A1= X
A2= X

$\overline{Y}_0=\overline{\overline{A}_2\overline{A}_1\overline{A}_0}=\overline{m}_0$ $\overline{Y}_4=\overline{A_2\overline{A}_1\overline{A}_0}=\overline{m}_4$

$\overline{Y}_1=\overline{\overline{A}_2\overline{A}_1A_0}=\overline{m}_1$ $\overline{Y}_5=\overline{A_2\overline{A}_1A_0}=\overline{m}_5$

$\overline{Y}_2=\overline{\overline{A}_2A_1\overline{A}_0}=\overline{m}_2$ $\overline{Y}_6=\overline{A_2A_1\overline{A}_0}=\overline{m}_6$

$\overline{Y}_3=\overline{\overline{A}_2A_1A_0}=\overline{m}_3$ $\overline{Y}_7=\overline{A_2A_1A_0}=\overline{m}_7$

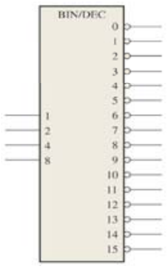
Input					Output							
S_1	$\overline{S}_2+\overline{S}_3$	A_2	A_1	A_0	$\overline{Y}_7$	$\overline{Y}_6$	$\overline{Y}_5$	$\overline{Y}_4$	$\overline{Y}_3$	$\overline{Y}_2$	$\overline{Y}_1$	$\overline{Y}_0$
0	X	X	X	X	1	1	1	1	1	1	1	1
X	1	X	X	X	1	1	1	1	1	1	1	1
1	0	0	0	0								0
1	0	0	0	1							0	
1	0	0	1	0		1				0		
1	0	0	1	1					0			
1	0	1	0	0				0				
1	0	1	0	1			0				1	
1	0	1	1	0		0						
1	0	1	1	1	0							

$\overline{S}_2=X$ $\overline{S}_3=X$ $\overline{Y}_0= 1$ $\overline{Y}_1= 1$ $\overline{Y}_2= 1$ $\overline{Y}_3= 1$ $\overline{Y}_4= 1$ $\overline{Y}_5= 1$ $\overline{Y}_6= 1$ $\overline{Y}_7= 1$

6.5.3 The 4-Bit Decoder 4位译码器

In order to decode all possible combinations of four bits, sixteen decoding gates are required(2^4=16). This type of decoder is commonly called a 4-line-to-16-line decoder because there are four inputs and sixteen outputs or a 1-of-16 decoder because for any given code on the inputs, one of the sixteen outputs is activated.

Decoding functions and truth table for a 4- line- to-16- line (1 of 16) decoder with active LOW outputs. (page 198)

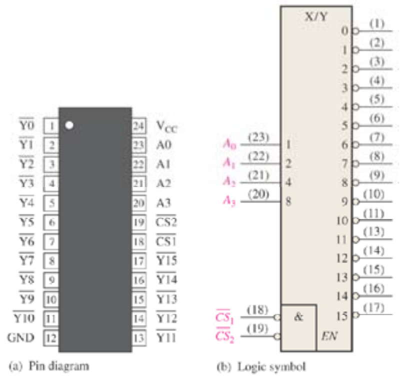


Logic symbol for a 4-line-to-16-line (1-of-16) decoder.

Decoding functions and truth table for a 4- line- to-16- line (1 of 16) decoder with active LOW outputs. (page 214)

DECIMAL DIGIT	INPUT $A_3A_2A_1A_0$	OUTPUTS 0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0 0 0 0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0 0 0 1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	0 0 1 0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1
3	0 0 1 1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1
4	0 1 0 0	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1
5	0 1 0 1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1
6	0 1 1 0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1
7	0 1 1 1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1
8	1 0 0 0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1
9	1 0 0 1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
10	1 0 1 0	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1
11	1 0 1 1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1
12	1 1 0 0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1
13	1 1 0 1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1
14	1 1 1 0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1
15	1 1 1 1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

THE 74HC154 1-OF-16 DECODER



Y0 Y1 Y2 Y3 Y4 Y5 Y6 Y7 Y8 Y9 Y10 GND
VCC A0 A1 A2 A3 CS2 CS1 Y15 Y14 Y13 Y12 Y11
X/Y
A0 A1 A2 A3
CS1 CS2
EN
(a) Pin diagram
(b) Logic symbol

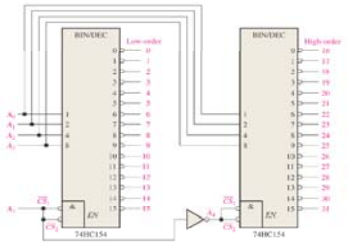
Pin diagram and logic symbol for the 74HC154 1-of-16 decoder.

Example

A certain application requires that a 5-bit number is decoded. Use 74HC154 decoders to implement the logic. The binary number is represented by the format $A_4A_3A_2A_1A_0$.

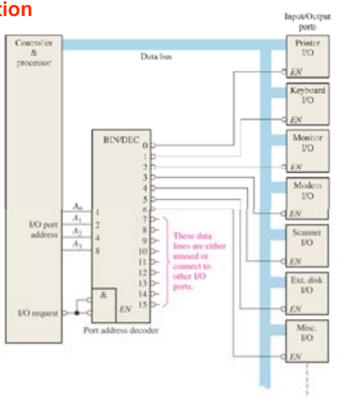
Solution

Since the 74HC154 can handle only four bits, two decoders must be used to decode five bits.



A 5-bit decoder using 74HC154s.

An Application



A simplified computer I/O port system with a port address decoder with only four address lines shown.

Example

Combinational Logic Implementation

A decoder provides the 2^n minterms of n input variables that can be used to form logic expression.

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

full adder truth table

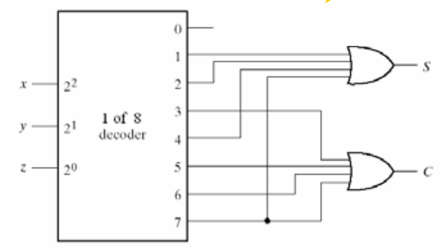
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

$$\Rightarrow \begin{cases} S(x,y,z) = \sum(1,2,4,7) \\ C(x,y,z) = \sum(3,5,6,7) \end{cases}$$

$$S(x,y,z) = \sum(1,2,4,7)$$

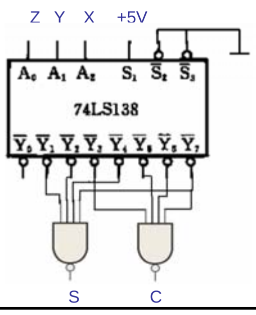
$$C(x,y,z) = \sum(3,5,6,7)$$

Implementation of a full adder with a decoder



$$S(x,y,z) = \sum(1,2,4,7)$$

$$C(x,y,z) = \sum(3,5,6,7)$$



Z Y X +5V
A_0 A_1 A_2 S_1 $\bar S_2$ $\bar S_3$
74LS138
$\bar Y_0$ $\bar Y_1$ $\bar Y_2$ $\bar Y_3$ $\bar Y_4$ $\bar Y_5$ $\bar Y_6$ $\bar Y_7$
S
C

6.5.4 The BCD-to-Decimal Decoder

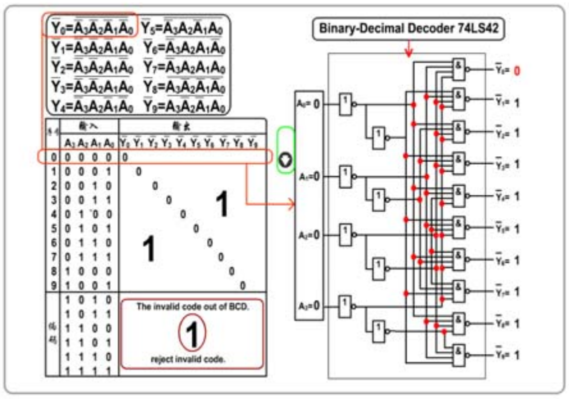
The BCD-decimal decoder converts each BCD (8421 code) into one of ten possible decimal digit indications. It is frequently referred as a 4-line-to-10-line decoder or a 1-of-10 decoder.

BCD-十进制译码器把每个BCD (8421 码) 转换成十进制数的一个输出。常常被称为4线-10线译码器或1-10译码器。

The method of implementation is the same as for the 1-of-16 decoder previously discussed, except that ten decoding gates are required because the BCD codes represents only the ten decimal digits 0 through 9.

BCD decoding functions.

Decimal Digit	BCD Code A_3	A_2	A_1	A_0	Decoding Function
0	0	0	0	0	$\bar A_3\bar A_2\bar A_1\bar A_0$
1	0	0	0	1	$\bar A_3\bar A_2\bar A_1 A_0$
2	0	0	1	0	$\bar A_3\bar A_2 A_1\bar A_0$
3	0	0	1	1	$\bar A_3\bar A_2 A_1 A_0$
4	0	1	0	0	$\bar A_3 A_2\bar A_1\bar A_0$
5	0	1	0	1	$\bar A_3 A_2\bar A_1 A_0$
6	0	1	1	0	$\bar A_3 A_2 A_1\bar A_0$
7	0	1	1	1	$\bar A_3 A_2 A_1 A_0$
8	1	0	0	0	$A_3\bar A_2\bar A_1\bar A_0$
9	1	0	0	1	$A_3\bar A_2\bar A_1 A_0$



Example

The 74HC42 is an integrated circuit BCD-to- decimal decoder. The logic symbol is shown below (Figure 1). If the input wave forms in Figure 2(a) are applied to the inputs of the 74HC42, show the output waveform.

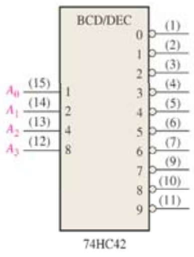


Figure 1 The 74HC42 BCD-to-decimal decoder

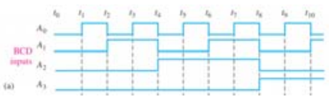


Figure 2

Solution

The output waveforms are shown in Figure 2(b). As you can see, the inputs are sequenced through the BCD for digits 0 to 9. The output waveform in the timing diagram indicate that sequence.

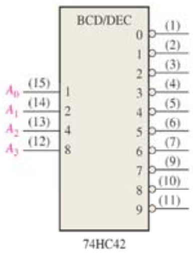


Figure 6–32 The 74HC42 BCD-to-decimal decoder

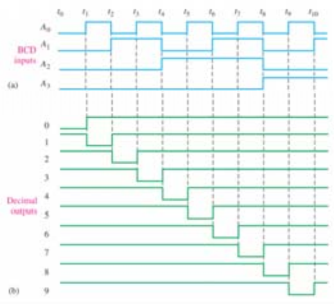
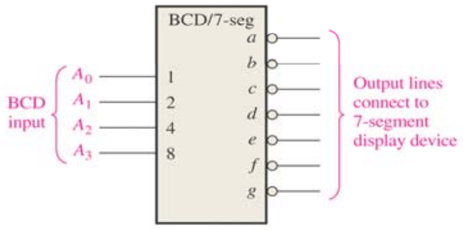


Figure 2

6.5.5 The BCD-to-7-Segment Decoder BCD-7段显示译码器

The BCD-to-7-Segment decoder accepts the BCD code on its inputs and provides outputs to drive 7-segment display devices to produce a decimal readout. The logic diagram for a basic 7-segment decoder is shown in the followinng Figure.



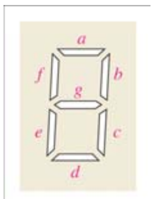
Logic symbol for a BCD-to-7-segment decoder/driver with active-LOW outputs

review

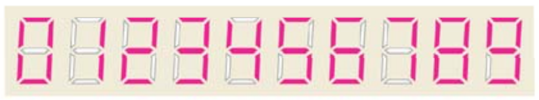
4-11 DIGITAL SYSTEM APPLICATION

Digital watches and other electronic equipment often display BCD-encoded decimal digits on seven-segment displays.（七段数码显示管）

4.11.1 The 7-segment Display

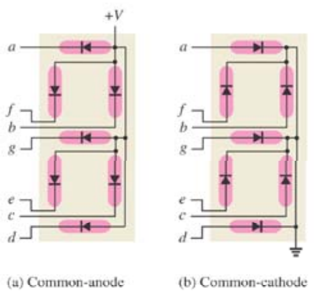


Seven-segment display format showing arrangement of segments.



Display of decimal digits with a 7-segment device

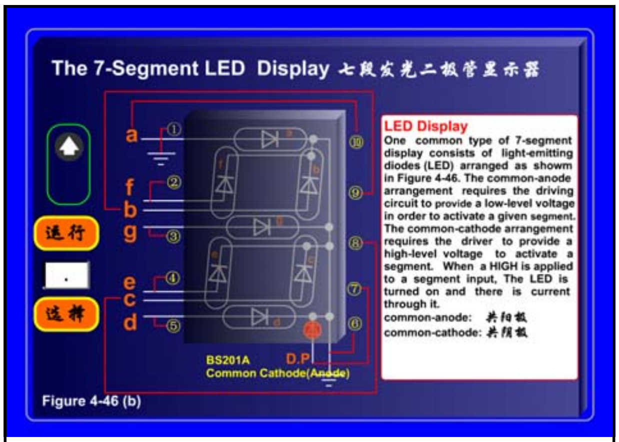
LED Displays



(a) Common-anode (b) Common-cathode

共阳极 共阴极

Arrangements of 7-segment LED displays.



The 7-Segment LED Display 七段发光二极管显示器

LED Display

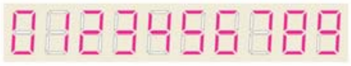
One common type of 7-segment display consists of light-emitting diodes (LED) arranged as showm in Figure 4-46. The common-anode arrangement requires the driving circuit to provide a low-level voltage in order to activate a given segment. The common-cathode arrangement requires the driver to provide a high-level voltage to activate a segment. When a HIGH is applied to a segment input, The LED is turned on and there is current through it.

common-anode: 共阳极

common-cathode: 共阴极

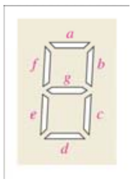
Figure 4-46 (b)

4.11.2 Segment Logic



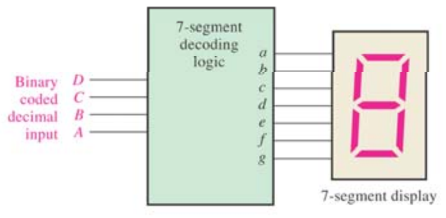
Active segments for each decimal digit

DIGIT	SEGMENTS ACTIVATED
0	a, b, c, d, e, f
1	b, c
2	a, b, d, e, g
3	a, b, c, d, g
4	b, c, f, g
5	a, c, d, f, g
6	a, c, d, e, f, g
7	a, b, c
8	a, b, c, d, e, f, g
9	a, b, c, d, f, g



4.11.3 Truth Table for the Segment Logic

The segment decoding logic requires four binary coded decimal (BCD) inputs and seven outputs, one for each segment in the display.



Block diagram of 7-segment logic and display.

DIGIT	SEGMENTS ACTIVATED
0	a, b, c, d, e, f
1	b, c
2	a, b, d, e, g
3	a, b, c, d, g
4	b, c, f, g
5	a, c, d, f, g
6	a, c, d, e, f, g
7	a, b, c
8	a, b, c, d, e, f, g
9	a, b, c, d, f, g

Truth table for 7-segment logic

DECIMAL DIGIT	INPUT				SEGMENT OUTPUTS						
	D	C	B	A	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
10	1	0	1	0	X	X	X	X	X	X	X
11	1	0	1	1	X	X	X	X	X	X	X
12	1	1	0	0	X	X	X	X	X	X	X
13	1	1	0	1	X	X	X	X	X	X	X
14	1	1	1	0	X	X	X	X	X	X	X
15	1	1	1	1	X	X	X	X	X	X	X

Boolean Expression for the segment logic

From the truth table, a standard SOP or POS expression can be written for each segment.

For example, the standard SOP expression for segment a is

$$a = \overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}B\overline{A} + \overline{D}\,\overline{C}BA + \overline{D}C\overline{B}A + \overline{D}CB\overline{A}$$
$$+ \overline{D}CBA + D\overline{C}\,\overline{B}\,\overline{A} + D\overline{C}\,\overline{B}A$$

and the standard SOP expression foe segment e is

$$e = \overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}B\overline{A} + \overline{D}CB\overline{A} + D\overline{C}\,\overline{B}\,\overline{A}$$

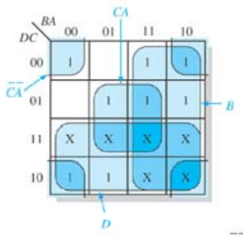
Expression for the other segments can be similarly developed.

Karnaugh Map Minimization of the Segment Logic

Karnaugh map minimization of the segment-a logic expression.

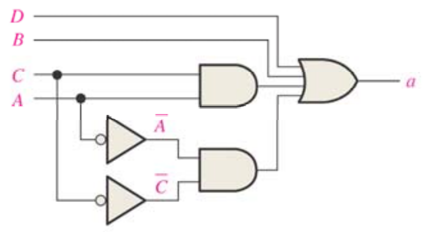
Standard SOP expression:

$\overline{D}\,\overline{C}\,\overline{B}\,\overline{A} + \overline{D}\,\overline{C}B\overline{A} + \overline{D}\,\overline{C}BA + \overline{D}C\overline{B}A + \overline{D}CB\overline{A} + \overline{D}CBA + D\overline{C}\,\overline{B}\,\overline{A} + D\overline{C}\,\overline{B}A$



Minimum SOP expression: $D + B + CA + \overline{C}\,\overline{A}$

$$a = D + B + CA + \overline{C}\,\overline{A}$$



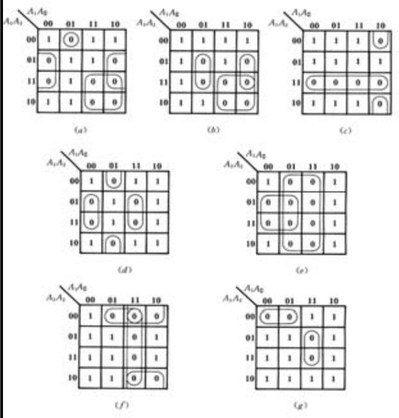
The minimum logic implementation for segment a of the 7-segment display.

Step1. The 1s mapped directly from the truth table.

Step2. All of the "don't care" (x) are placed on the map.

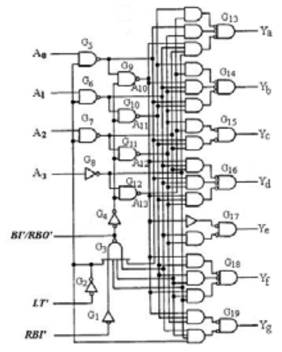
Step3. The 1s are grouped as shown. "Don't cares" and overlapping of cells are utilized to form the largest groups possible.

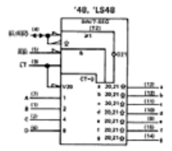
Step4. Write the minimum product term for each group and sum the terms to form the minimum SOP expression.



$$\begin{cases} Y_a = \overline{\overline{A}_3\overline{A}_2\overline{A}_1A_0 + A_3A_1 + A_2\overline{A}_0} \\ Y_b = \overline{A_3A_1 + A_2A_1\overline{A}_0 + A_2\overline{A}_1A_0} \\ Y_c = \overline{A_3A_2 + \overline{A}_2A_1\overline{A}_0} \\ Y_d = \overline{A_2A_1A_0 + A_2\overline{A}_1\overline{A}_0 + \overline{A}_2\overline{A}_1A_0} \\ Y_e = \overline{A_2\overline{A}_1 + A_0} \\ Y_f = \overline{\overline{A}_3\overline{A}_2A_0 + \overline{A}_2A_1 + A_1A_0} \\ Y_g = \overline{\overline{A}_3\overline{A}_2\overline{A}_1 + A_2A_1A_0} \end{cases}$$

$$\begin{cases} Y_a = \overline{\overline{A}_3\overline{A}_2\overline{A}_1A_0 + A_3A_1 + A_2\overline{A}_0} \\ Y_b = \overline{A_3A_1 + A_2A_1\overline{A}_0 + A_2\overline{A}_1A_0} \\ Y_c = \overline{A_3A_2 + \overline{A}_2A_1\overline{A}_0} \\ Y_d = \overline{A_2A_1A_0 + A_2\overline{A}_1\overline{A}_0 + \overline{A}_2\overline{A}_1A_0} \\ Y_e = \overline{A_2\overline{A}_1 + A_0} \\ Y_f = \overline{\overline{A}_3\overline{A}_2A_0 + \overline{A}_2A_1 + A_1A_0} \\ Y_g = \overline{\overline{A}_3\overline{A}_2\overline{A}_1 + A_2A_1A_0} \end{cases}$$

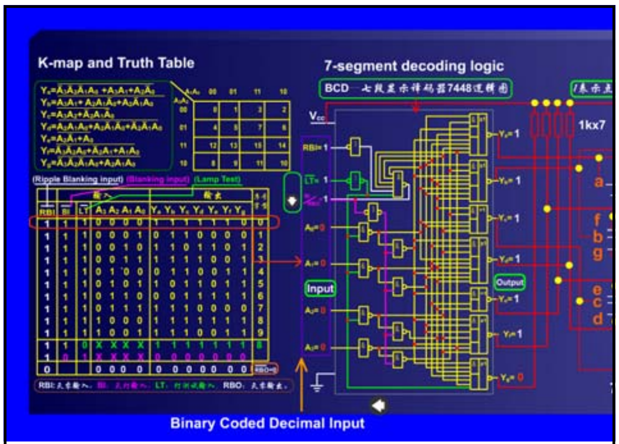




(b) traditional logic symbol

(a) logic diagrams

The 74x48 BCD-to-seven-segment decoder



K-map and Truth Table

7-segment decoding logic

BCD—七段显示译码器7448逻辑图

(Ripple Blanking input) (Blanking input) (Lamp Test)

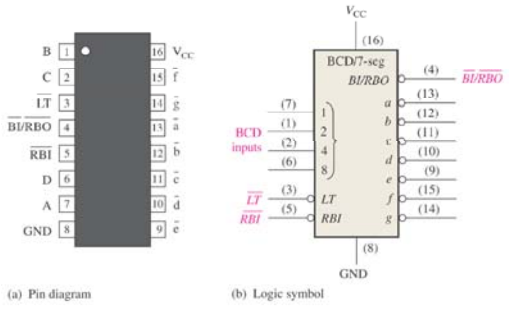
RBI	BI	LT	A_3	A_2	A_1	A_0	Y_a	Y_b	Y_c	Y_d	Y_e	Y_f	Y_g	字形
1	1	1	0	0	0	0	1	1	1	1	1	1	0	0
1	1	1	0	0	0	1	0	1	1	0	0	0	0	1
1	1	1	0	0	1	0	1	1	0	1	1	0	1	2
1	1	1	0	0	1	1	1	1	1	1	0	0	1	3
1	1	1	0	1	0	0	0	1	1	0	0	1	1	4
1	1	1	0	1	0	1	1	0	1	1	0	1	1	5
1	1	1	0	1	1	0	0	0	1	1	1	1	1	6
1	1	1	0	1	1	1	1	1	1	0	0	0	0	7
1	1	1	1	0	0	0	1	1	1	1	1	1	1	8
1	1	1	1	0	0	1	1	1	1	0	0	1	1	9
1	1	0	X	X	X	X	1	1	1	1	1	1	1	8
1	0	1	X	X	X	X	0	0	0	0	0	0	0	
0			0	0	0	0	0	0	0	0	0	0	0	RBO=0

RBI: 灭零输入, BI: 灭灯输入, LT: 灯测试输入, RBO: 灭零输出。

Binary Coded Decimal Input

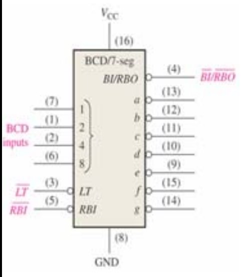
The 74LS47 BCD-to-7-Segment Decoder/Driver

The outputs can drive a common-anode 7-segment display directly.



Pin diagram and logic symbol for the 74LS47 BCD-to-7-segment decoder/driver.

The 74LS47 is an example of an MSI device that decodes a BCD input and drives a 7-segment display. In addition to its decoding and segment drive capability, the 74LS47 has several additional features as indicated by the $\overline{LT}$, $\overline{RBI}$, $\overline{BI}/\overline{RBO}$ functions in the logic symbol of the following Figure.

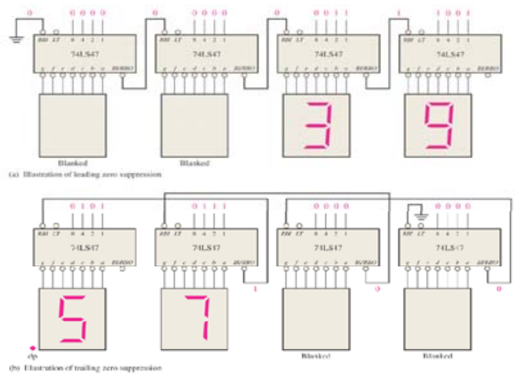


$\overline{LT}$: lamp test		灯测试
$\overline{RBI}$: ripple blanking input		灭零输入
$\overline{BI}$: blanking input		灭灯输入
$\overline{RBO}$: ripple blanking output		灭零输出

Lamp Test Whe n a LOW is applied to the $\overline{LT}$ input and $\overline{BI}/\overline{RBO}$ is HIGH, all of the 7 segments in the display are turned on. Lamp test is used to verify that no segments are burned out.

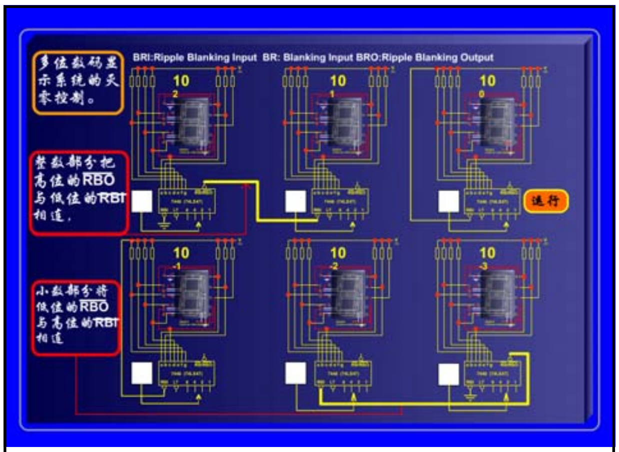
当LT加上低电平，BI/RBO 加上高电平，这时所有的7段都被点亮。灯测试用来测试是否有显示段烧坏。

Zero Suppression Zero suppression is a feature used for multidigit displays to blank out unnecessary zeros. For example, in a 3-digit display the number 3 may be displayed as 003. If the zeros are not blanked out.



74LS47
Blanked
(a) Illustration of leading zero suppression
(b) Illustration of trailing zero suppression

Examples of zero suppression using the 74LS47 BCD to 7-segment decoder/driver.



BRI:Ripple Blanking Input BR: Blanking Input BRO:Ripple Blanking Output
10
运行

多位数码显示系统的灭零控制。

整数部分把高位的$\overline{RBO}$与低位的$\overline{RBI}$相连。

小数部分将低位的$\overline{RBO}$与高位的$\overline{RBI}$相连

6-6 ENCODERS 编码器

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 symbols or numbers to coded format is called encoding.

6.6.1 The Decimal-to-BCD Encoder

This type of encoder has ten inputs — one for each digit — and four outputs corresponding to the BCD code, as shown in Figure 6-37. This is a basic 10-line-to-4-line encoder.

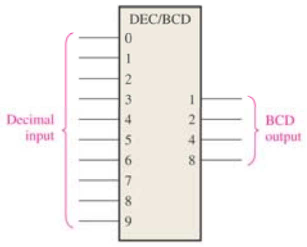


Figure 6–37 Logic symbol for a decimal-to-BCD encoder.

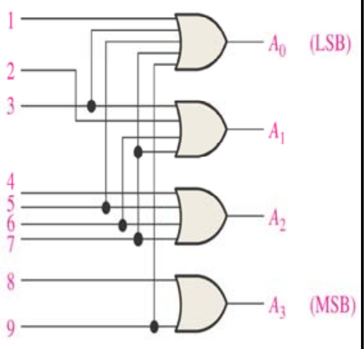
DECIMAL DIGIT	BCD CODE A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

A_3=8+9

A_2=4+5+6+7

A_1=2+3+6+7

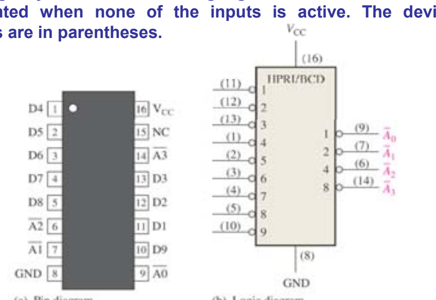
A_0=1+3+5+7+9



Basic logic diagram of a decimal-to-BCD encoder. A 0-digit input is not needed because the BCD outputs are all LOW when there are no HIGH inputs.

THE 74HC147 DECIMAL-to-BCD PRIORITY ENCODER

The 74HC147 is a priority encoder with active-LOW inputs (0) for decimal digits 1 through 9 and active-LOW BCD outputs as indicated in the logic symbol in the following Figure 6-39. A BCD zero output is represented when none of the inputs is active. The device pin numbers are in parentheses.



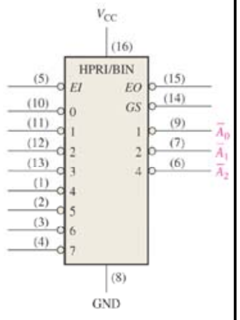
(a) Pin diagram (b) Logic diagram

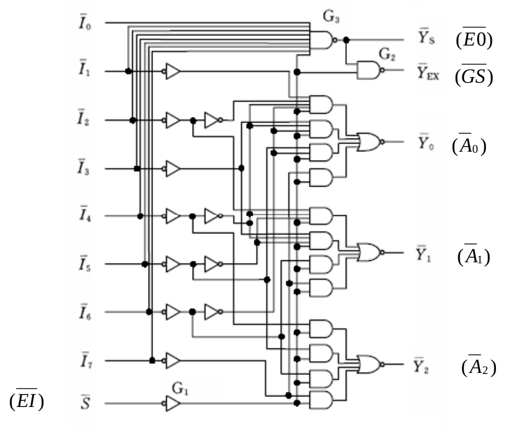
INPUTS									OUTPUTS			
$\bar{I}_1$	$\bar{I}_2$	$\bar{I}_3$	$\bar{I}_4$	$\bar{I}_5$	$\bar{I}_6$	$\bar{I}_7$	$\bar{I}_8$	$\bar{I}_9$	$\bar{Y}_3$	$\bar{Y}_2$	$\bar{Y}_1$	$\bar{Y}_0$
1	1	1	1	1	1	1	1	1	1	1	1	1
×	×	×	×	×	×	×	×	0	0	1	1	0
×	×	×	×	×	×	×	0	1	0	1	1	1
×	×	×	×	×	×	0	1	1	1	0	0	0
×	×	×	×	×	0	1	1	1	1	0	0	1
×	×	×	×	0	1	1	1	1	1	0	1	0
×	×	×	0	1	1	1	1	1	1	0	1	1
×	×	0	1	1	1	1	1	1	1	1	0	0
×	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

THE 74LS148 8-LINE-TO-3-LINE ENCODER

The 74LS148 is a priority encoder that has eight active-LOW inputs and three active-LOW binary outputs, as shown in the following Figure.

This device can be used for converting octal inputs (recall that octal digits are 0 through 7) to a 3-bit binary code. To enable the device, the EI (enable input) must be LOW. It also has the EO (enable output) and GS output for expansion purposes. The EO is LOW when the EI is LOW and none of the inputs (0 through 7) is active. GS is LOW when EI is LOW and any of the inputs is active.




$\bar{I}_0$
$\bar{I}_1$
$\bar{I}_2$
$\bar{I}_3$
$\bar{I}_4$
$\bar{I}_5$
$\bar{I}_6$
$\bar{I}_7$
$(\overline{EI})$ $\bar{S}$
G1
G2
G3
$\bar{Y}_S$ $(\overline{E0})$
$\bar{Y}_{EX}$ $(\overline{GS})$
$\bar{Y}_0$ $(\bar{A}_0)$
$\bar{Y}_1$ $(\bar{A}_1)$
$\bar{Y}_2$ $(\bar{A}_2)$

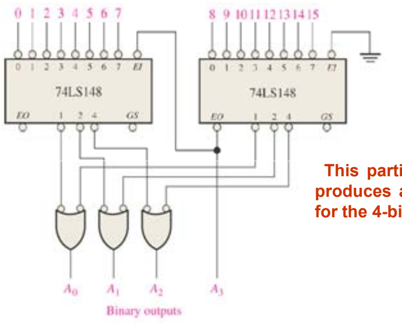
INPUTS									OUTPUTS				
$\overline{EI}$	$\bar{I}_0$	$\bar{I}_1$	$\bar{I}_2$	$\bar{I}_3$	$\bar{I}_4$	$\bar{I}_5$	$\bar{I}_6$	$\bar{I}_7$	$\bar{A}_2$	$\bar{A}_1$	$\bar{A}_0$	$\overline{EO}$	$\overline{GS}$
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	0	1
0	×	×	×	×	×	×	×	0	0	0	0	1	0
0	×	×	×	×	×	×	0	1	0	0	1	1	0
0	×	×	×	×	×	0	1	1	0	1	0	1	0
0	×	×	×	×	0	1	1	1	0	1	1	1	0
0	×	×	×	0	1	1	1	1	1	0	0	1	0
0	×	×	0	1	1	1	1	1	1	0	1	1	0
0	×	0	1	1	1	1	1	1	1	1	0	1	0
0	0	1	1	1	1	1	1	1	1	1	1	1	0

$$\overline{EO} = \overline{\bar{I}_0\bar{I}_1\bar{I}_2\bar{I}_3\bar{I}_4\bar{I}_5\bar{I}_6\bar{I}_7(EI)}$$

$$\overline{GS} = \overline{\overline{\bar{I}_0\bar{I}_1\bar{I}_2\bar{I}_3\bar{I}_4\bar{I}_5\bar{I}_6\bar{I}_7(EI)}\cdot(EI)}$$
$$= \overline{(I_0+I_1+I_2+I_3+I_4+I_5+I_6+I_7)\cdot(EI)}$$

Encoder Expansion

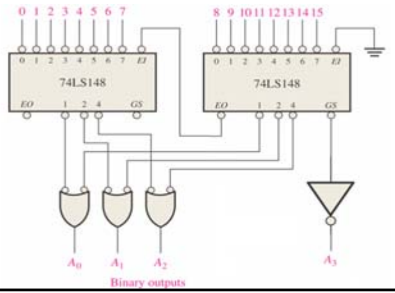
Encoders can be Expanded to handle more bits by cascading.



This particular configuration produces active-High outputs for the 4-bit binary number.

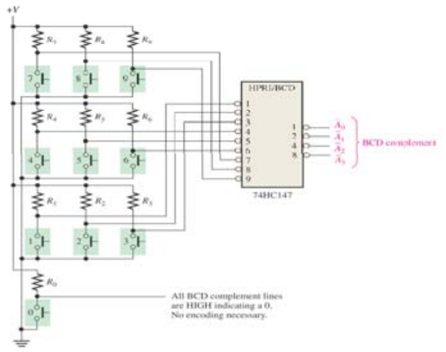
A 16-line-to-4 line encoder using 74LS148s and external logic

The 74LS148 can be expanded to a 16-line-to-4-line encoder by connecting the EO of the higher-order encoder to the EI of the lower-order encoder and NAND the corresponding binary outputs as shown in the following Figure. The GS links to an inverter as the fourth and most significant bit. This particular configuration produces active-High outputs for the 4-bit binary number.



6.6.2 An Application

keyboard encoder



+V
HPRI/BCD
74HC147
BCD complement
All BCD complement lines are HIGH indicating a 0. No encoding necessary.

A simplified keyboard encoder

6-7 CODE CONVERTERS

In this section, we will examine some method of using combinational logic circuit to convert from one code to another.

6.7.1 BCD-to-Binary Conversion

One method of BCD-to-Binary code conversion uses adder circuits. The basic conversion process is as follows:

1. The value, or weight, of each bit in the BCD number is represented by a binary number.
2. All of the binary representations of the weights of bits that are 1s in the BCD number are added.
3. The result of this addition is the binary equivalent of the BCD number.

A more concise statement of this operation is

The binary number representing the weights of the BCD bits are summed to produce the total binary number.

Binary representation of BCD bit weights

		(MSB)	Binary Representation					(LSB)
BCD Bit	BCD Weight	64	32	16	8	4	2	1
A_0	1	0	0	0	0	0	0	1
A_1	2	0	0	0	0	0	1	0
A_2	4	0	0	0	0	1	0	0
A_3	8	0	0	0	1	0	0	0
B_0	10	0	0	0	1	0	1	0
B_1	20	0	0	1	0	1	0	0
B_2	40	0	1	0	1	0	0	0
B_3	80	1	0	1	0	0	0	0

EXAMPLE

the decimal number 87 can be expressed in BCD as

$\underline{1000}\ \underline{0111}$
8 7

	Tens Digit				Units Digit			
Weight:	80	40	20	10	8	4	2	1
Bit designation:	B_3	B_2	B_1	B_0	A_3	A_2	A_1	A_0

```
80  40  20  10   8   4   2   1

1   0   0   0    0   1   1   1
                                0000 0001    1
                                0000 0010    2
                                0000 0100    4
                              + 0101 0000   80
                                0101 0111  ← Binary number
                                             for decimal 87
```

Example

Convert the BCD numbers 00100111(decimal 27) and 10011000 (decimal 98) to binary.

Solution

Write the binary representations of the weights of all 1s appearing in the numbers, and then add them together.

```
80  40  20  10   8   4   2   1

 0   0   1   0   0   1   1   1
                             └→ 0000 0001    1
                         └────→ 0000 0010    2
                     └────────→ 0000 0100    4
         └────────────────────→+0001 0100   20
                                ─────────
                                0001 1011  ← Binary number
                                             for decimal 27
```

```
80  40  20  10   8   4   2   1

 1   0   0   1   1   0   0   0
                 └────────→ 000 1000    8
             └────────────→ 000 1010   10
 └────────────────────────→+101 0000   80
                            ────────
                            110 0010  ← Binary number
                                        for decimal 98
```

6.7.2 Binary-to-Gray and Gray-to-Binary Conversion

The basic process for Gray-binary conversions was covered in Chapter 2. Exclusive-OR gates can be used for these conversions.

Figure (a) shows a 4-bit binary-to-Gray code converter, and Figure (b) illustrates a 4-bit Gray-to-binary converter.

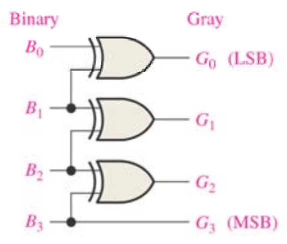


Figure (a) Four-bit binary-to-Gray conversion logic

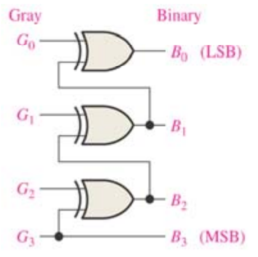


Figure (b) Four-bit Gray-to-binary conversion logic.

Four-bit binary-to-Gray conversion logic

INPUT B_3 B_2 B_1 B_0	OUTPUT G_3 G_2 G_1 G_0
0 0 0 0	0 0 0 0
0 0 0 1	0 0 0 1
0 0 1 0	0 0 1 1
0 0 1 1	0 0 1 0
0 1 0 0	0 1 1 0
0 1 0 1	0 1 1 1
0 1 1 0	0 1 0 1
0 1 1 1	0 1 0 0
1 0 0 0	1 1 0 0
1 0 0 1	1 1 0 1
1 0 1 0	1 1 1 1
1 0 1 1	1 1 1 0
1 1 0 0	1 0 1 0
1 1 0 1	1 0 1 1
1 1 1 0	1 0 0 1
1 1 1 1	1 0 0 0

B_1B_0 \ B_3B_2	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$$G_0 = \overline{B}_1 B_0 + B_1 \overline{B}_0$$
$$= B_1 \oplus B_0$$

B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$$G_1 = \overline{B}_2 B_1 + B_2 \overline{B}_1$$
$$= B_2 \oplus B_1$$

B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$$G_2 = \overline{B}_3 B_2 + B_3 \overline{B}_2$$
$$= B_3 \oplus B_2$$

B_1B_0 \ B_3B_2	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

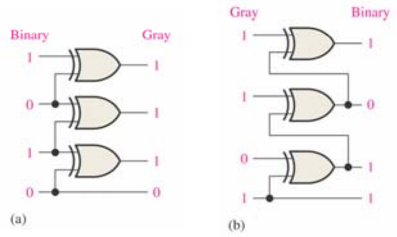
$$G_3 = B_3$$

Example

(a) Convert the binary number 0101 to Gray code with exclusive-OR gates.
(b) Convert the Gray code 1011 to binary with exclusive-OR gates.

Solution

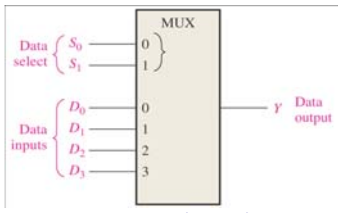
(a) 0101_2 is 0111 Gray. See Figure (a).
(b) 1011 Gray is 1101_2. See Figure (b).



6-8 Multiplexers (Data Selectors)

A multiplexer (MUX) is a device that allows digital information from several sources to be routed onto a single line for transmission over that line to a common destination.

The basic multiplexer has several data-input lines and a single output line. It also has data-select inputs, which permit digital data on any one of the inputs to be switched to the output line. Multiplexer are also known as data selectors.

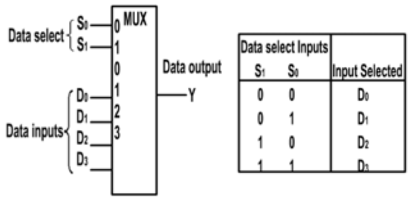


Logic symbol for a 1-of-4 data selector/multiplexer.

Notice that there are two data-select lines because with two select bits, any one of the four data-input lines can be selected.

A 2-bit code on the data-select(S) inputs will allow the data on the selected data input to pass through to the data output.

A 2-bit code on the data-select(S) inputs will allow the data on the selected data input to pass through to the data output.



Data select Inputs		Input Selected
S_1	S_0	
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

The data output is equal to the state of the selected data input. You can therefore, derive a logic expression for the output in terms of the data input and select inputs.

The data output is equal to D_0 only if S_1=0 and S_0=0: $Y=D_0\overline{S}_1\overline{S}_0$

The data output is equal to D_1 only if S_1=0 and S_0=0: $Y=D_1\overline{S}_1S_0$

The data output is equal to D_2 only if S_1=0 and S_0=0: $Y=D_2S_1\overline{S}_0$

The data output is equal to D_3 only if S_1=1 and S_0=1: $Y=D_3S_1S_0$

The data output is equal to D_0 only if S_1=0 and S_0=0: $Y=D_0\overline{S}_1\overline{S}_0$

The data output is equal to D_1 only if S_1=0 and S_0=0: $Y=D_1\overline{S}_1S_0$

The data output is equal to D_2 only if S_1=0 and S_0=0: $Y=D_2S_1\overline{S}_0$

The data output is equal to D_3 only if S_1=1 and S_0=1: $Y=D_3S_1S_0$

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

$$Y = D_0\overline{S}_1\overline{S}_0 + D_1\overline{S}_1S_0 + D_2S_1\overline{S}_0 + D_3S_1S_0$$

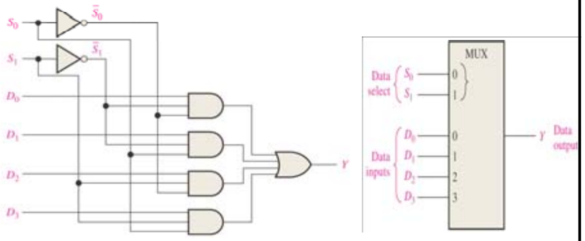


Figure 6-47 Logic diagram for a 4-input multiplexer.

Example

The data-input and data-select waveform in Figure 6-48(a) are applied to the multiplexer in Figure 6-47. Determine the output waveform in relation to the inputs.

Solution

The resulting output waveform is shown in Figure 6-48(b).

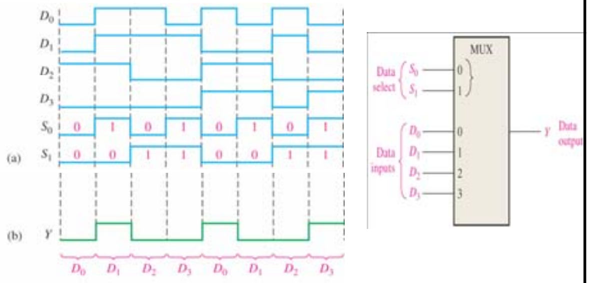


Figure 6-48

The 74HC157 QUAD 2-INPUT DATA SELECTOR / MULTIPLEXER

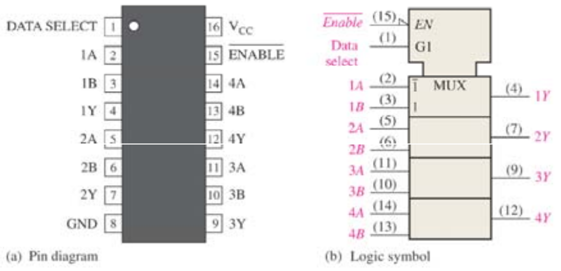


Figure 6–49 Pin diagram and logic symbol for the 74HC157 quadruple 2-input data selector/multiplexer.

The 74LS151 8-INPUT DATA SELECTOR / MULTIPLEXER

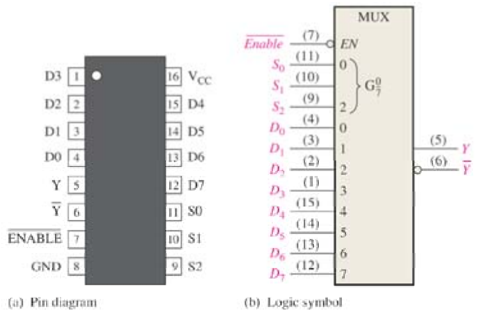


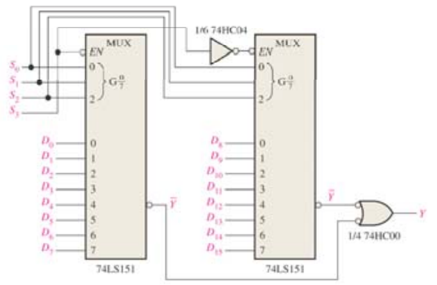
Figure 6–50 Pin diagram and logic symbol for the 74LS151 8-input data selector/multiplexer.

EXAMPLE

Use 74HC151s and any other logic necessary to multiplexer 16 data lines onto a single data-output line.

Solution

An implementation of this system is shown in Figure 6-50. Four bits are required to select one of 16 data inputs(2^4=16).



A 16-input multiplexer

Application Examples

A 7-Segment Display Multiplexer

Figure 6-52 shows a simplified method of multiplexing BCD numbers to a 7-segment display.

In this example, 2-digit numbers are displayed on the 7-segment readout by the use of a single BCD-TO-7-Segment decoder.

This basic method of display multiplexing can be extended to displays with any number of digits.

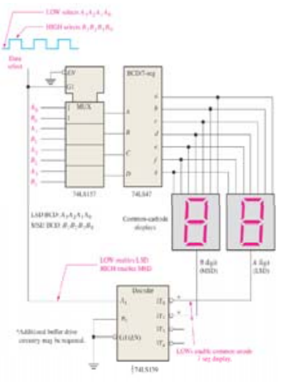
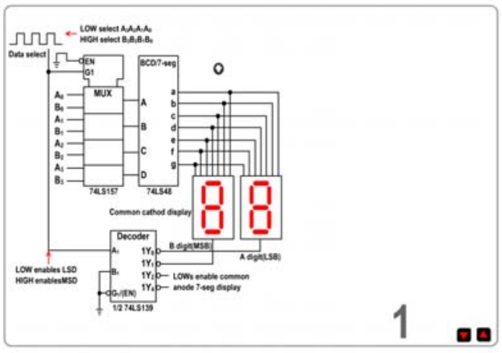


Figure 6–52 Simplified 7-segment display multiplexing logic.

The basic operation is as follows.



A Logic Function Generator

A useful application of the data selector/multiplexer is in the generation of combinational logic function in sum-of-products form. When used in this way, the device can replace discrete gates, can often greatly reduce the number of ICs, and can make design changes much easier.

To illustrate, a 74LS151 8-input data selector/multiplexer can be used to implement any specified 3-variable logic function if the variable combination are connected to the data-select inputs and each data input is set to the logic level required in the truth table for that function.

For example , if the function is a 1 when the variable combination is $\bar{A}_2A_1\bar{A}_0$, the 2 input (selected by 010) is connected to a HIGH. This HIGH is passed through to the output when this particular combination of variables occurs on the data-select lines.

EXAMPLE

Implement the logic function specified in Table 6-9 by using a 74LS151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

Solution

Notice from the truth table that Y is a 1 for the following input variable combinations: 001, 011, 101, and 110. For all other combinations, Y is 0.

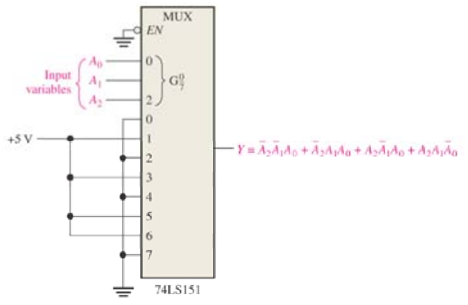
For this function to be implemented with the data selector, the data input selected by each of the above-mentioned combinations must be connected to a HIGH (5V). All the other data inputs must be connected to a LOW(ground), as shown in Figure 6-52.

$$Y=\bar{A}_2\bar{A}_1A_0+\bar{A}_2A_1A_0+A_2\bar{A}_1A_0+A_2A_1\bar{A}_0$$

Table 6-9

Inputs			Output
A_2	A_1	A_0	Y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

The implementation of this function with logic gates would require four 3-input AND gates, one 4-input OR gates, and three inverters unless the expression can be simplified.



MUX
EN
Input variables
A_0
A_1
A_2
$G\frac{0}{7}$
+5 V
$Y=\bar{A}_2\bar{A}_1A_0+\bar{A}_2A_1A_0+A_2\bar{A}_1A_0+A_2A_1\bar{A}_0$
74LS151

Figure 6–53 Data selector/multiplexer connected as a 3-variable logic function generator.

The 8-input data selector can be used as a logic function generator for three variable. Actually, this device can be used as a 4-variable logic function generator by the utilization of one of the bits(A_0) in conjunction with the data inputs.

A 4-variable truth table has sixteen combinations of input variables. When an 8-bit data selector is used, each input is selected twice: the first time when A_0 is 0 and the second time when A_0 is 1. With this in mind, the following rules can be applied (Y is the output, and A_0 is the least significant bit):

1. If Y=0 both times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, connect that data input to ground(0).

2. If Y=1 both times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, connect that data input +V (1).

3. If Y is different the two times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, and if $Y=A_0$, connect that data input to A_0.

4. If Y is different the two times a given data input is selected by a certain combination of the input variables, $A_3A_2A_1$, and if $Y=\overline{A}_0$, connect that data input to $\overline{A}_0$.

Example

Implement the logic function in Table 6-10 by using a 74LS151 8-input data selector/multiplexer. Compare this method with a discrete logic gate implementation.

Table 6-10

Decimal Digit	Inputs A_3	A_2	A_1	A_0	Output Y
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	1
6	0	1	1	0	1
7	0	1	1	1	1
8	1	0	0	0	1
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	0
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	0
15	1	1	1	1	1

Table 6-10

Decimal Digit	Inputs A_3	A_2	A_1	A_0	Output Y	Y
0	0	0	0	0	0	$\overline{A}_3\overline{A}_2\overline{A}_1\cdot A_0$
1	0	0	0	1	1	
2	0	0	1	0	1	$\overline{A}_3\overline{A}_2A_1\cdot\overline{A}_0$
3	0	0	1	1	0	
4	0	1	0	0	0	$\overline{A}_3A_2\overline{A}_1\cdot A_0$
5	0	1	0	1	1	
6	0	1	1	0	1	$\overline{A}_3A_2A_1\cdot 1$
7	0	1	1	1	1	
8	1	0	0	0	1	$A_3\overline{A}_2\overline{A}_1\cdot\overline{A}_0$
9	1	0	0	1	0	
10	1	0	1	0	1	$A_3\overline{A}_2A_1\cdot\overline{A}_0$
11	1	0	1	1	0	
12	1	1	0	0	1	$A_3A_2\overline{A}_1\cdot 1$
13	1	1	0	1	1	
14	1	1	1	0	0	$A_3A_2A_1\cdot A_0$
15	1	1	1	1	1	

$$\begin{aligned}Y=&\overline{A}_3\overline{A}_2\overline{A}_1A_0+\overline{A}_3\overline{A}_2A_1\overline{A}_0\\&+\overline{A}_3A_2\overline{A}_1A_0+\overline{A}_3A_2A_1\cdot 1\\&+A_3\overline{A}_2\overline{A}_1\overline{A}_0+A_3\overline{A}_2A_1\overline{A}_0\\&+A_3A_2\overline{A}_1\cdot 1+A_3A_2A_1A_0\end{aligned}$$

The expression of the logic function

$$\begin{aligned}Y=&\overline{A}_3\overline{A}_2\overline{A}_1A_0+\overline{A}_3\overline{A}_2A_1\overline{A}_0+\overline{A}_3A_2\overline{A}_1A_0+\overline{A}_3A_2A_1\cdot 1\\&+A_3\overline{A}_2\overline{A}_1\overline{A}_0+A_3\overline{A}_2A_1\overline{A}_0+A_3A_2\overline{A}_1\cdot 1+A_3A_2A_1A_0\end{aligned}\qquad\textbf{(1)}$$

The expression of a 74LS151 8-to-1 MUX

$$\begin{aligned}Y=(&\overline{A}_3\overline{A}_2\overline{A}_1D_0+\overline{A}_3\overline{A}_2A_1D_1+\overline{A}_3A_2\overline{A}_1D_2+\overline{A}_3A_2A_1D_3\\&+A_3\overline{A}_2\overline{A}_1D_4+A_3\overline{A}_2A_1D_5+A_3A_2A_1D_6+A_3A_2A_1D_7)\cdot\overline{EN}\end{aligned}\qquad\textbf{(2)}$$

As *compared* (1) with (2) , we can consider A_3, A_2 and A_1 as A_3, A_2 and A_1. We can write

$$A_3=A_3,\quad A_2=A_2,\quad A_1=A_1,\quad \overline{EN}=0$$
$$D_0=A_0,\quad D_1=\overline{A}_0,\quad D_2=A_0,\quad D_3=1$$
$$D_4=\overline{A}_0,\quad D_5=\overline{A}_0,\quad D_6=1,\quad D_7=A_0$$

$$A_3=A_3,\quad A_2=A_2,\quad A_1=A_1,\quad \overline{EN}=0$$
$$D_0=A_0,\quad D_1=\overline{A}_0,\quad D_2=A_0,\ D_3=1,\ D_4=\overline{A}_0,\ D_5=\overline{A}_0,\ D_6=1,\ D_7=A_0$$

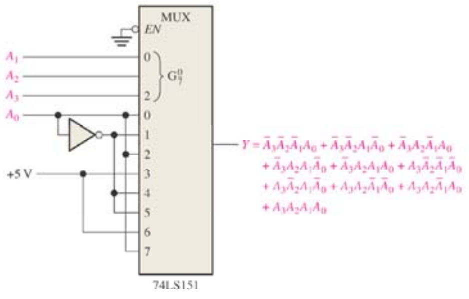

MUX
EN
A_1
A_2
A_3
A_0
0
2
G_7^0
0
1
2
3
4
5
6
7
+5 V
$Y=\overline{A}_3\overline{A}_2\overline{A}_1A_0+\overline{A}_3\overline{A}_2A_1\overline{A}_0+\overline{A}_3A_2\overline{A}_1A_0+\overline{A}_3A_2A_1\overline{A}_0+\overline{A}_3A_2A_1A_0+A_3\overline{A}_2\overline{A}_1\overline{A}_0+A_3\overline{A}_2A_1\overline{A}_0+A_3A_2\overline{A}_1\overline{A}_0+A_3A_2\overline{A}_1A_0+A_3A_2A_1A_0$
74LS151

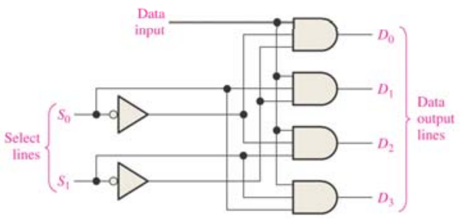
Figure 6–54 Data selector/multiplexer connected as a 4-variable logic function generator.

The implementation of this function with logic gates would require ten 4-input AND gates, one 10-input OR gates, and four inverters, although possible simplification would reduce this requirement.

6-9 Demultiplexers 多路分配器

A demultiplexer (DEMUX) basically reverses the multiplexing function. It takes data from one line and distributes them to a given number of output lines. For this reason, the demultiplexer is also known as a data distributor. As you will learn, decoders can also be used as demultiplexers.

Figure 6-55 shows a 1-line-to-4-line demultiplexer(DEMUX) cicuit.



$$D_0 = \overline{S}_1 \overline{S}_0 D_{in}$$

$$D_1 = \overline{S}_1 S_0 D_{in}$$

$$D_2 = S_1 \overline{S}_0 D_{in}$$

$$D_3 = S_1 S_0 D_{in}$$

Figure 6–55 A 1-line-to-4-line demultiplexer.

Example

The serial data-input waveform (Dat in) and data-select inputs (S_0 and S_1) are shown in Figure 6-56. Determine the data-output waveform on D_0 through D_3 for the demultiplexer in Figure 6-55.

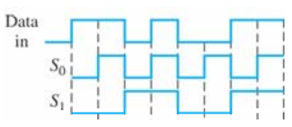


Figure 6–56

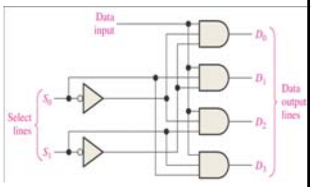
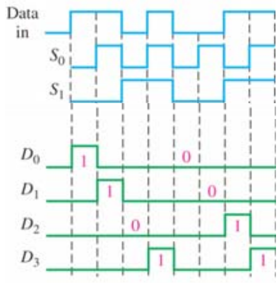
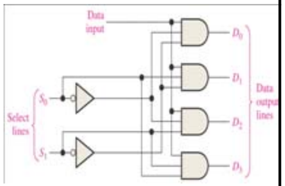


Figure 6–55

Solution

Notice that the select lines go through a binary sequence so that each successive input bit is routed to D_0, D_1, D_2, and D_3 in sequence, as shown by the output waveforms in Figure 6-56.





$$D_0 = \overline{S}_1 \overline{S}_0 D_{in} \quad D_1 = \overline{S}_1 S_0 D_{in}$$

$$D_2 = S_1 \overline{S}_0 D_{in} \quad D_3 = S_1 S_0 D_{in}$$

Figure 6-56

The 74HC154 decoder used as a demultiplexer

We have already discussed the 74HC154 in its application as a 4-line -to-16-line decoder (Section 6-4). This device and other decoders can also be used in demultiplexing application.

The logic symbol for this device when used as a demultiplexer is shown in Figure 6-57.

In demultiplexer applications, the input lines are used as the data-select lines. One of the chip select inputs is used as the data-input line, with the other chip select input held LOW to enable the internal negative-AND gate at the bottom of the diagram.

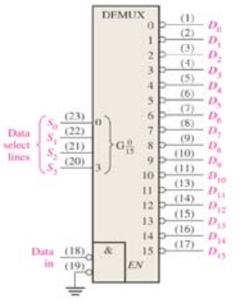


Figure 6–57 The 74HC154 decoder used as a demultiplexer.

6-10 Parity Generators/Checkers

Errors can occur as digital codes are being transferred from one point to another within a digital system or while codes are being transmitted from one system to another.

The errors take the form of undesired changes in the bits that make up the coded information; that is, a 1 can change to a 0, or a 0 to a 1, because of component malfunctions or electrical noise.

In most digital system, the probability that even a single bit error will occur is very small, and the likelihood that more than one will occur is even smaller. Nevertheless, when an error occurs undetected, it can cause serious problems in a digital system.

Review: The BCD code with parity bits

1.parity: 奇偶性 2.even: 偶数 3. odd : 奇数

Parity Method for Error Detection

Many system use a parity bit as a means for bit error detection. Any group of bits contain either an even or an odd number of 1s. A parity bit is attached to a group of bits to make the total number of 1s in a group always even or always odd. An even parity bit makes the total number of 1s even, and an odd parity bit makes the total odd.

Table 2-8
The BCD code with parity bits

Even Parity		Odd Parity	
P	BCD	P	BCD
0	0000	1	0000
1	0001	0	0001
1	0010	0	0010
0	0011	1	0011
1	0100	0	0100
0	0101	1	0101
0	0110	1	0110
1	0111	0	0111
1	1000	0	1000
0	1001	1	1001

1

6.10.1 Basic Parity Logic

A parity bit indicates if the number of 1s in a code is even or old for the purpose of error detection.

In order to check for or to generate the proper parity in a given code, a very basic principle can be used:

The sum (disregarding carries) of an even number of 1s is always 0, and the sum of an odd number of 1s is always 1.

Therefore, to determine if a given code has even parity or odd parity, all the bits in that code are summed.

the sum of two bits can be generated by an exclusive-OR gate, as shown in Figure 6-58(a); the sum of four bits can be formed by three exclusive-OR gates connected as shown in Figure 6-58 (b); and so on.

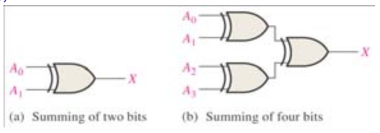
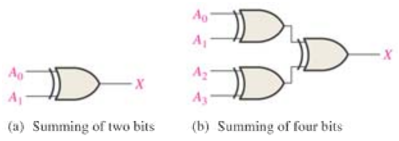


Figure 6–58

the sum of two bits can be generated by an exclusive-OR gate, as shown in Figure 6-58(a); the sum of four bits can be formed by three exclusive-OR gates connected as shown in Figure 6-58 (b); and so on.

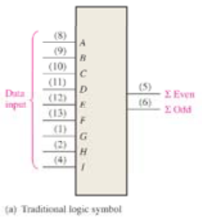


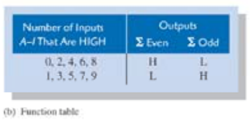
When the number of 1s on the inputs is even, the output X is 0 (LOW). When the number of 1s os odd, the output X is 1(High).

The 74LS280 9-bit Parity Generator/Checker

The logic symbol and function table for a 74LS280 are shown in Figure 6-59. This particular MSI device can be used to check for odd or even parity on a 9-bit code (eight data bits and one parity bit) or it can be used to generate a parity bit for a binary code with up to nine bits.

The inputs are A through I; when there is an even number of 1s on the inputs, the ∑ even output is High and the∑ odd output is LOW.





Number of Inputs A–I That Are HIGH	Outputs Σ Even	Outputs Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

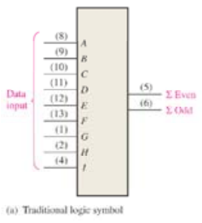
(b) Function table

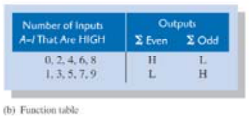
Figure 6–59 The 74LS280 9-bit parity generator/checker.

Parity Check

When this device is used as an even parity checker, the number of input bits should always be even; and when a parity error occurs, the ∑Even output goes LOW and the ∑Odd output goes HIGH.

When it is used as an odd parity checker, the number of input bits should always be odd; and when a parity error occurs, the ∑Odd output goes LOW and the ∑Even output goes HIGH.





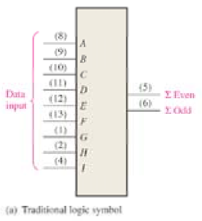
Number of Inputs A–I That Are HIGH	Outputs Σ Even	Outputs Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

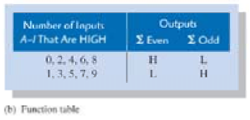
(b) Function table

Parity Generator

If this device is used as an even parity generator, the parity bit is taken at the ∑Odd output because this output is a 0 if there is an even number of input bits and it is a 1 if there is an odd number.

When used as an odd parity generator, the parity bit is taken at the ∑Even output because it is a 0 when the number of inputs bits is odd.





Number of Inputs A–I That Are HIGH	Outputs Σ Even	Outputs Σ Odd
0, 2, 4, 6, 8	H	L
1, 3, 5, 7, 9	L	H

(b) Function table

6.10.2 A Data Transmission System with Error Detection

A simplified data transmission system is shown in Figure 6-60 to illustrate an application of parity generators/checkers, as well as multiplexers and demultiplexers, and to illustrate the need for data storage in some applications.

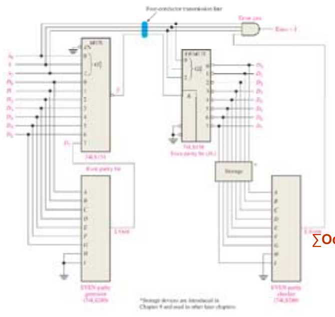
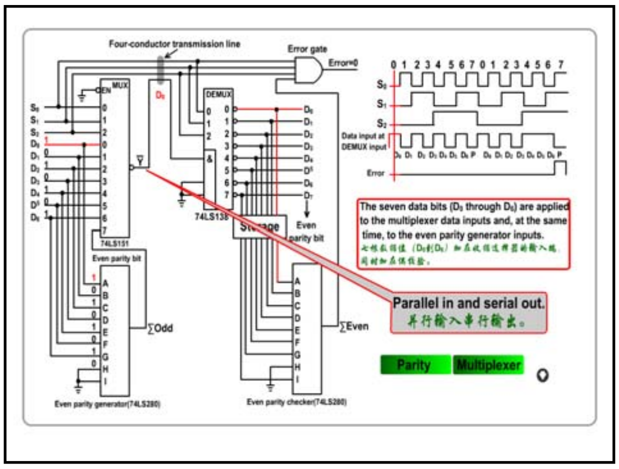


Figure 6-60



The seven data bits (D_0 through D_6) are applied to the multiplexer data inputs and, at the same time, to the even parity generator inputs.
七根数据位（D_0到D_6）加在数据选择器的输入端，同时加在偶校验器。

Parallel in and serial out.
并行输入串行输出。