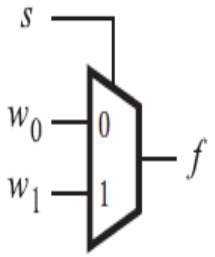
# Module-3

# Combinational Circuit Building Blocks

- In this chapter you will learn about:
  - Commonly used combinational subcircuits
  - Multiplexers, which can be used for selection of signals and for implementation of general logic functions
  - Circuits used for encoding, decoding, and code-conversion purposes
  - Arithmetic comparison circuits

# Multiplexers

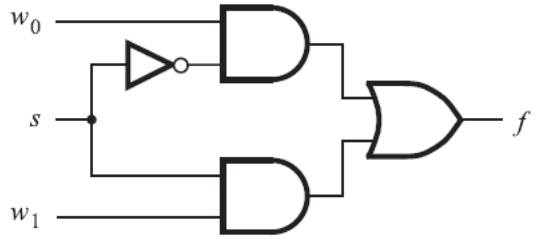
- Multiplexers – Data Selectors.
- A multiplexer circuit has a number of data inputs, one or more select inputs, and one output.
- It passes the signal value on one of the data inputs to the output.
- The data input is selected by the values of the select inputs.
- Since there are ‘n’ selection lines, there will be $2^n$ possible combinations of zeros and ones.
- Each combination will select only one data input.
- Multiplexer is also called as **Mux.**




(a) Graphical symbol

| $s$ | $f$ |
|---|---|
| 0 | $w_0$ |
| 1 | $w_1$ |

(b) Truth table




(c) Sum-of-products circuit

**Figure 1:** A 2-to-1 multiplexer.

- A larger multiplexer with four data inputs, *w*0, . . . ,*w*3, and two select inputs, *s*1 and *s*0.
- As shown in the truth table in part *(b)* of the figure, the two-bit number represented by *s*1, *s*0 selects one of the data inputs as the output of the multiplexer.

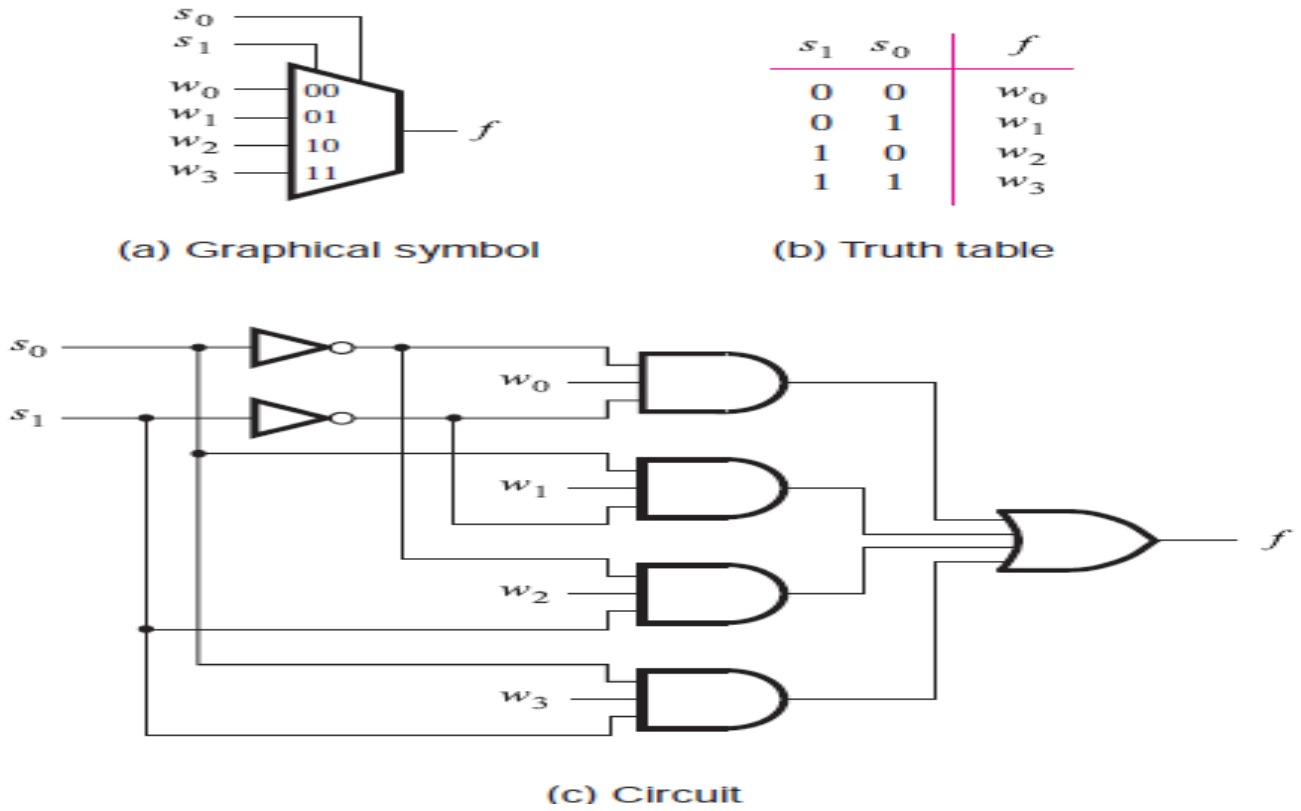



**Figure 2:** A 4-to-1 multiplexer.

- It is possible to build larger multiplexers using the same approach.
- Usually, the number of data inputs, *n*, is an integer power of two.
- A multiplexer that has *n* data inputs,*w*0, . . . ,*wn*−1, requires $\log_2 n$ select inputs.
- Larger multiplexers can also be constructed from smaller multiplexers.
- For example, the 4-to-1 multiplexer can be built using three 2-to-1 multiplexers as illustrated in Figure 3.
- Figure 4 shows how a 16-to-1 multiplexer is constructed with five 4-to-1 multiplexers.

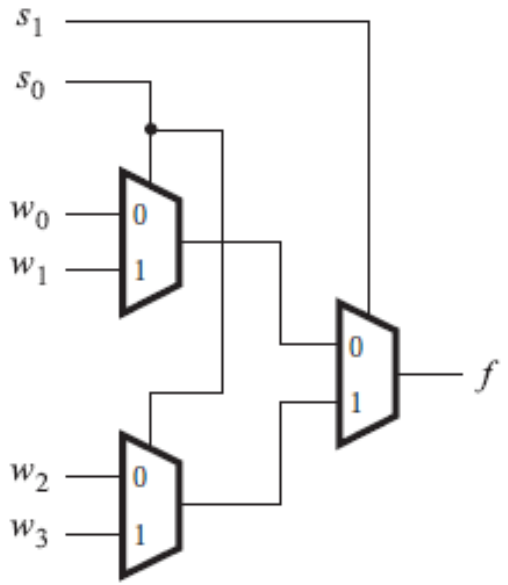



**Figure 3:** Using 2-to-1 multiplexers to build a 4-to-1multiplexer.

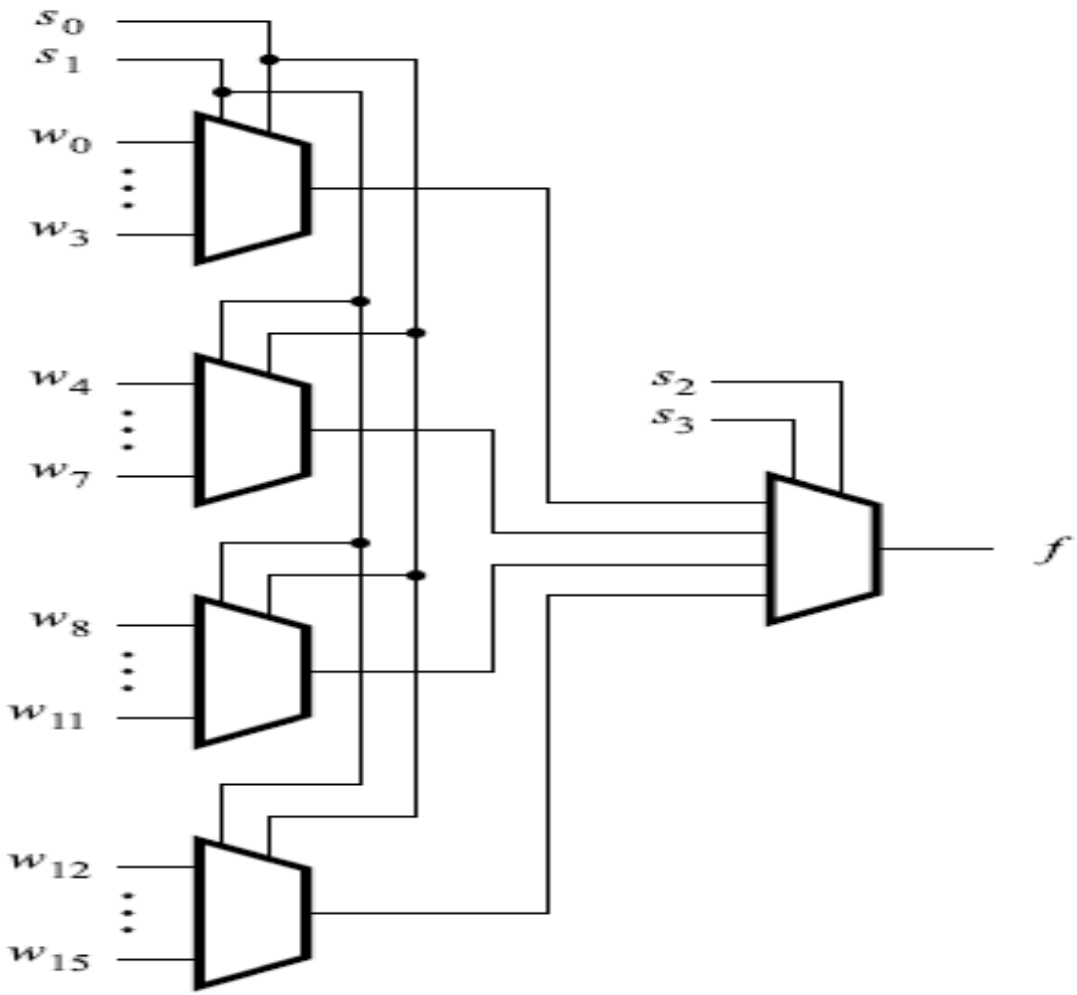



**Figure 4:** A 16-to-1 multiplexer.

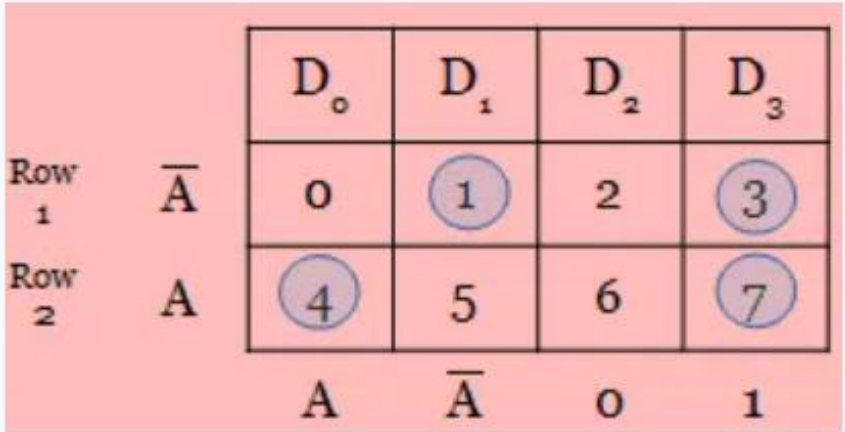
**Type #2**

If the boolean expression has n+1 variables, we take only n variables as the selection line of the multiplexer. The remaining single variable is used as the inputs of the mux. In this way, any boolean expression having n variables can be implemented with $2^{n-1}$ to 1 multiplexer.

- How to solve this method?

- From the given set of n+1 input variables, the n least significant variables are used as selection line inputs.
- The $2^n$ inputs for $2^n$ : 1 multiplexer are derived by using an implementation table.
- The implementation table has all the inputs($D_0$, $D_1$, $D_2$, $D_3$,…) for the multiplexer, under which, all the minterms are listed in two rows.
- The first row consists of all minters where A is complemented and the second row has the remaining minterms where A is in uncomplemented form.
- The minterms in the given boolean expression alone are circled.

Look at the example of implementation table here,



| | | $D_0$ | $D_1$ | $D_2$ | $D_3$ |
|---|---|---|---|---|---|
| Row 1 | $\overline{A}$ | 0 | 1 | 2 | 3 |
| Row 2 | A | 4 | 5 | 6 | 7 |
| | | A | $\overline{A}$ | 0 | 1 |

Each column is analyzed separately as follows:

- If both the minterms in the two rows are circled, 1 is applied to the corresponding multiplexer input.*(check 4th column in the above table)*
- If both the minterms in the two rows are not circled, 0 is applied to the corresponding multiplexer input.*(check 3rd column in the above table)*
- If the minterm in the first row alone is circled, $\overline{A}$ is applied to the corresponding multiplexer input.*(check 2nd column in the above table)*
- If the minterm in the second row alone is circled, $A$ is applied to the corresponding multiplexer input.*(check 1st column in the above table)*

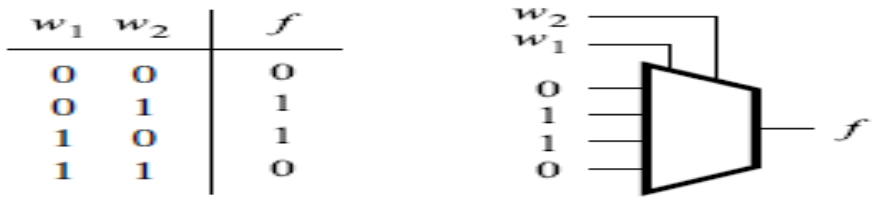
# Synthesis of Logic Functions Using Multiplexers

- Consider an Example, the truth table defines the function $f = w1 \oplus w2$.
- This function can be implemented by a 4-to-1 multiplexer in which the values of $f$ in each row of the truth table are connected as constants to the multiplexer data inputs.
- The multiplexer select inputs are driven by $w1$ and $w2$.
- Thus, for each valuation of $w1$, $w2$, the output $f$ is equal to the function value in the corresponding row of the truth table.

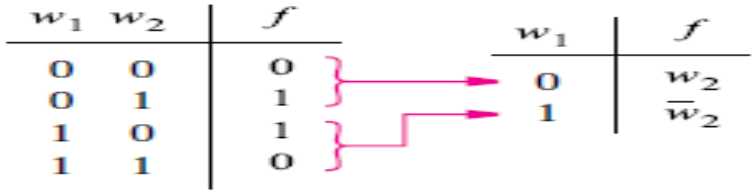
- A better implementation can be derived by manipulating the truth table as indicated.
- $f$ to be implemented by a single 2-to-1 multiplexer.
- One of the input signals, $w1$ is chosen as the select input of the 2-to-1 multiplexer.

- Truth table is redrawn to indicate the value of *f* for each value of *w*1. When *w*1 = 0, *f* has the same value as input *w*2, and when *w*1 = 1, *f* has the value of *w*2.

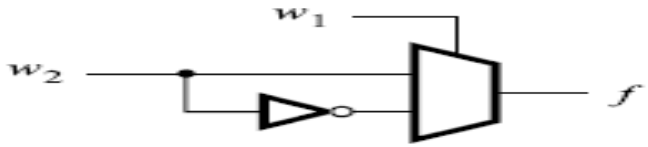
- This procedure can be applied to synthesize a circuit that implements any logic function.




(a) Implementation using a 4-to-1 multiplexer




(b) Modified truth table

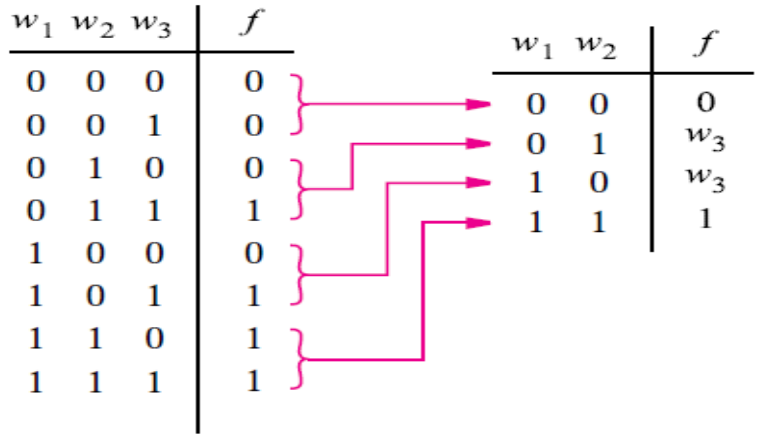



(c) Circuit

**Figure 5**: Synthesis of a logic function using mutiplexers.

## ❑ Example Problems:

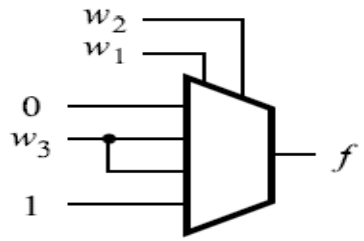
1. Figure below gives the truth table for the three-input majority function, and it shows how the truth table can be modified to implement the function using a 4-to-1 multiplexer. Any two of the three inputs may be chosen as the multiplexer select inputs.



| $w_1$ | $w_2$ | $w_3$ | $f$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

| $w_1$ | $w_2$ | $f$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | $w_3$ |
| 1 | 0 | $w_3$ |
| 1 | 1 | 1 |

(a) Modified truth table




(b) Circuit

**Figure 6:** Implementation of the three-input majority function using a 4-to-1 multiplexer.

# Multiplexer Synthesis Using Shannon's Expansion

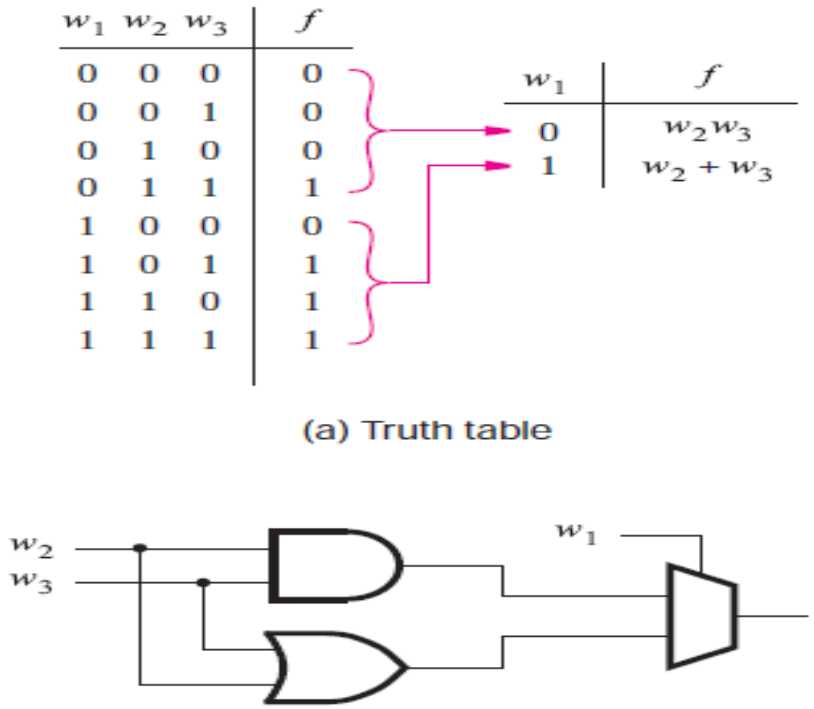
- The inputs to the multiplexers are the constants 0 and 1, or some variable or its complement.
- Besides using such simple inputs, it is possible to connect more complex circuits as inputs to a multiplexer, allowing functions to be synthesized using a combination of multiplexers and other logic gates.
- The truth table can be modified as shown on the right. If $w1 = 0$, then $f = w2w3$, and if $w1 = 1$, then $f = w2 + w3$.
- This implementation can be derived using algebraic manipulation as follows.
- The function in Figure 6.11*a* is expressed in sum-of-products form as

$$f = \overline{w}_1 w_2 w_3 + w_1 \overline{w}_2 w_3 + w_1 w_2 \overline{w}_3 + w_1 w_2 w_3$$

It can be manipulated into,

$$\begin{aligned} f &= \overline{w}_1 (w_2 w_3) + w_1 (\overline{w}_2 w_3 + w_2 \overline{w}_3 + w_2 w_3) \\ &= \overline{w}_1 (w_2 w_3) + w_1 (w_2 + w_3) \end{aligned}$$

- which corresponds to the circuit in Figure 6.11*b*.
- Multiplexer implementations of logic functions require that a given function be decomposed in terms of the variables that are used as the select inputs. This can be accomplished by means of a theorem proposed by Claude Shannon [1].




(a) Truth table

(b) Circuit

**Figure 6.11** The three-input majority function implemented using a 2-to-1 multiplexer.

# Shannon's Expansion Theorem

Any Boolean function $f(w_1, \ldots, w_n)$ can be written in the form

$$f(w_1, w_2, \ldots, w_n) = \overline{w}_1 \cdot f(0, w_2, \ldots, w_n) + w_1 \cdot f(1, w_2, \ldots, w_n)$$

This expansion can be done in terms of any of the $n$ variables. We will leave the proof of the theorem as an exercise for the reader (see problem 6.9).

To illustrate its use, we can apply the theorem to the three-input majority function, which can be written as

$$f(w_1, w_2, w_3) = w_1 w_2 + w_1 w_3 + w_2 w_3$$

Expanding this function in terms of $w_1$ gives

$$f = \overline{w}_1 (w_2 w_3) + w_1 (w_2 + w_3)$$

which is the expression that we derived above.

For the three-input XOR function, we have

$$\begin{aligned} f &= w_1 \oplus w_2 \oplus w_3 \\ &= \overline{w}_1 \cdot (w_2 \oplus w_3) + w_1 \cdot \overline{(w_2 \oplus w_3)} \end{aligned}$$

which gives the circuit in Figure 6.9*b*.

In Shannon's expansion the term $f(0, w_2, \ldots, w_n)$ is called the *cofactor* of $f$ with respect to $\overline{w}_1$; it is denoted in shorthand notation as $f_{\overline{w}_1}$. Similarly, the term $f(1, w_2, \ldots, w_n)$ is called the cofactor of $f$ with respect to $w_1$, written $f_{w_1}$. Hence we can write

$$f = \overline{w}_1 f_{\overline{w}_1} + w_1 f_{w_1}$$

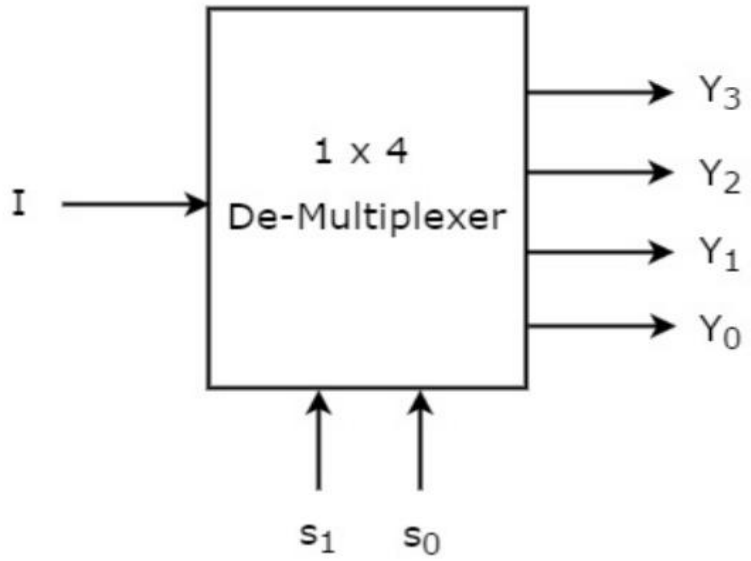
In general, if the expansion is done with respect to variable $w_i$, then $f_{w_i}$ denotes $f(w_1, \ldots, w_{i-1}, 1, w_{i+1}, \ldots, w_n)$ and

$$f(w_1, \ldots, w_n) = \overline{w}_i f_{\overline{w}_i} + w_i f_{w_i}$$

The complexity of the logic expression may vary, depending on which variable, $w_i$, is used, as illustrated in Example 6.5.

# De-Multiplexer

- **De-Multiplexer** is a combinational circuit that performs the reverse operation of Multiplexer.
- It has single input, 'n' selection lines and maximum of $2^n$ outputs.
- The input will be connected to one of these outputs based on the values of selection lines.
- Since there are 'n' selection lines, there will be $2^n$ possible combinations of zeros and ones.
- So, each combination can select only one output.
- De-Multiplexer is also called as **De-Mux**.
- 1x4 De-Multiplexer
- 1x4 De-Multiplexer has one input I,
- Two selection lines, $s_1$ & $s_0$ and four outputs $Y_3$, $Y_2$, $Y_1$ &$Y_0$.



- The single input 'I' will be connected to one of the four outputs, $Y_3$ to $Y_0$ based on the values of selection lines $s_1$ & s0.

| Selection Inputs | | Outputs | | | |
|---|---|---|---|---|---|
| $S_1$ | $S_0$ | $Y_3$ | $Y_2$ | $Y_1$ | $Y_0$ |
| 0 | 0 | 0 | 0 | 0 | I |
| 0 | 1 | 0 | 0 | I | 0 |
| 1 | 0 | 0 | I | 0 | 0 |
| 1 | 1 | I | 0 | 0 | 0 |

$$Y_3 = s_1 s_0 I$$

$$Y_1 = s_1' s_0 I$$

$$Y_2 = s_1 s_0' I$$

$$Y_0 = s_1' s_0' I$$

# Decoders

- Decoder circuits are used to decode encoded information.
- A binary decoder, depicted in Figure 6.15, is a logic circuit with *n* inputs and 2*n* outputs.
- Only one output is asserted at a time, and each output corresponds to one valuation of the inputs.
- The decoder also has an enable input, *En*, that is used to disable the outputs; if *En* = 0, then none of the
- decoder outputs is asserted. If *En* = 1, the valuation of $w_n{-}1 \cdot \cdot \cdot w1w0$ determines which of the outputs is asserted.

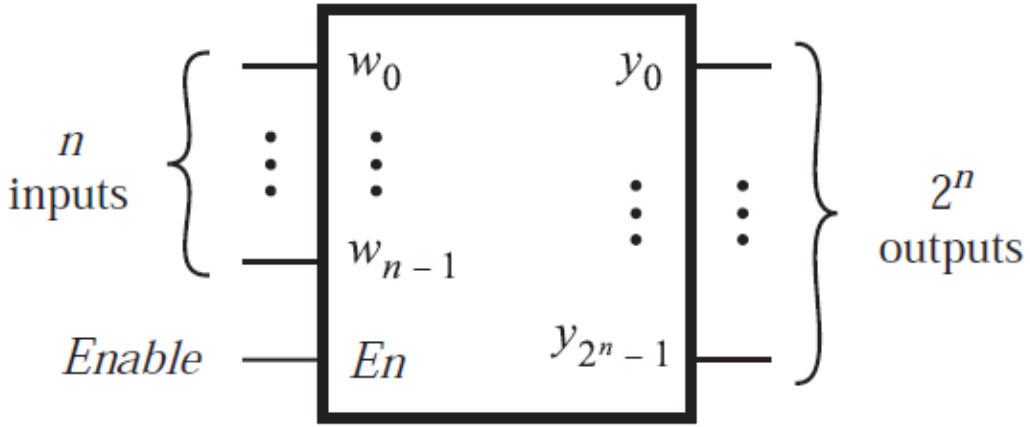
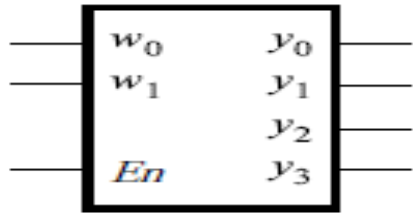



**Figure 6.15** An $n$-to-$2^n$ binary decoder.

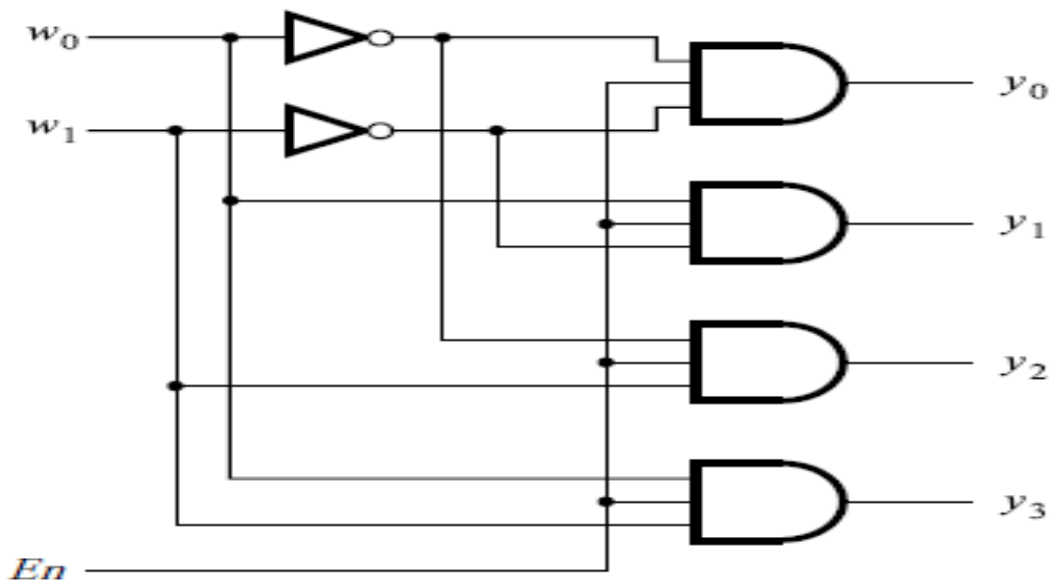
- An *n*-bit binary code in which exactly one of the bits is set to 1 at a time is referred to as *one-hot encoded*, meaning that the single bit that is set to 1 is deemed to be "hot." The outputs of a binary decoder are one-hot encoded.
- A 2-to-4 decoder is given in **Figure 6.16.** The two data inputs are $w1$ and $w0$.
- They represent a two-bit number that causes the decoder to assert one of the outputs $y0, \ldots, y3$.
- Although a decoder can be designed to have either active-high or active-low outputs, in **Figure 6.16** active-high outputs are assumed. Setting the inputs $w1w0$ to 00, 01, 10, or 11 causes the output $y0$, $y1$, $y2$, or $y3$ to be set to 1, respectively.
- A graphical symbol for the decoder is given in **part *(b)*** of the figure, and a logic circuit is shown in **part *(c)*.**

| $En$ | $w_1$ | $w_0$ | $y_0$ | $y_1$ | $y_2$ | $y_3$ |
|---|---|---|---|---|---|---|
| 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 |
| 0 | x | x | 0 | 0 | 0 | 0 |

(a) Truth table




(b) Graphical symbol




(c) Logic circuit

**Figure 6.16** A 2-to-4 decoder.

- Larger decoders can be built using the sum-of-products structure in **Figure 6.16*c***, or else they can be constructed from smaller decoders.
- **Figure 6.17** shows how a 3-to-8 decoder is built with two 2-to-4 decoders.
- The *w*2 input drives the enable inputs of the two decoders.
- The top decoder is enabled if *w*2 = 0, and the bottom decoder is enabled if *w*2 = 1.
- This concept can be applied for decoders of any size.

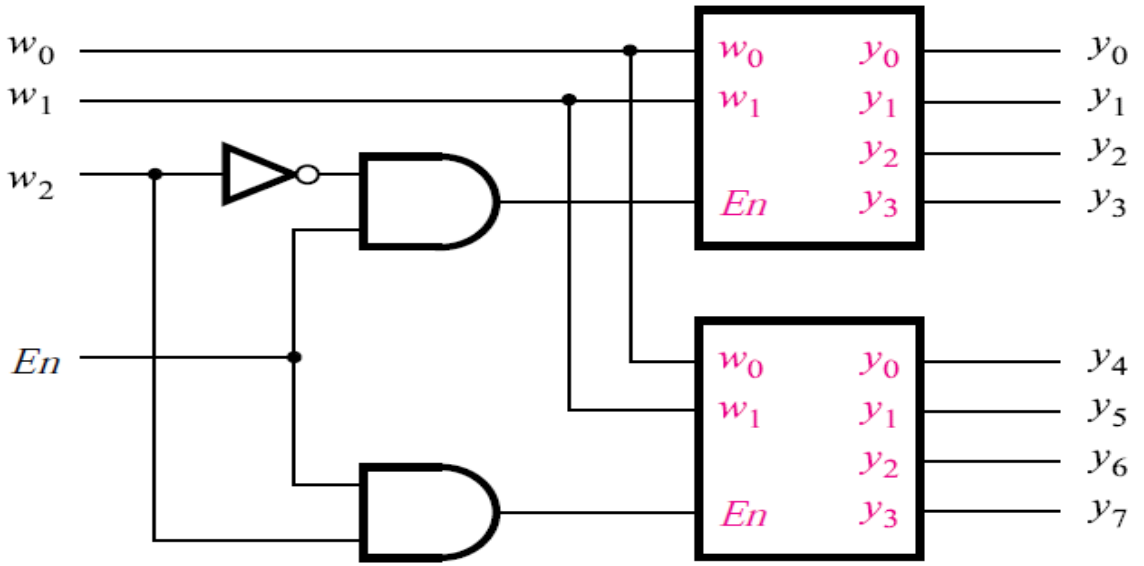



**Figure 6.17** A 3-to-8 decoder using two 2-to-4 decoders.

- **Figure 6.18** shows how five 2-to-4 decoders can be used to construct a 4-to-16 decoder.
- Because of its treelike structure, this type of circuit is often referred to as a *decoder tree*.

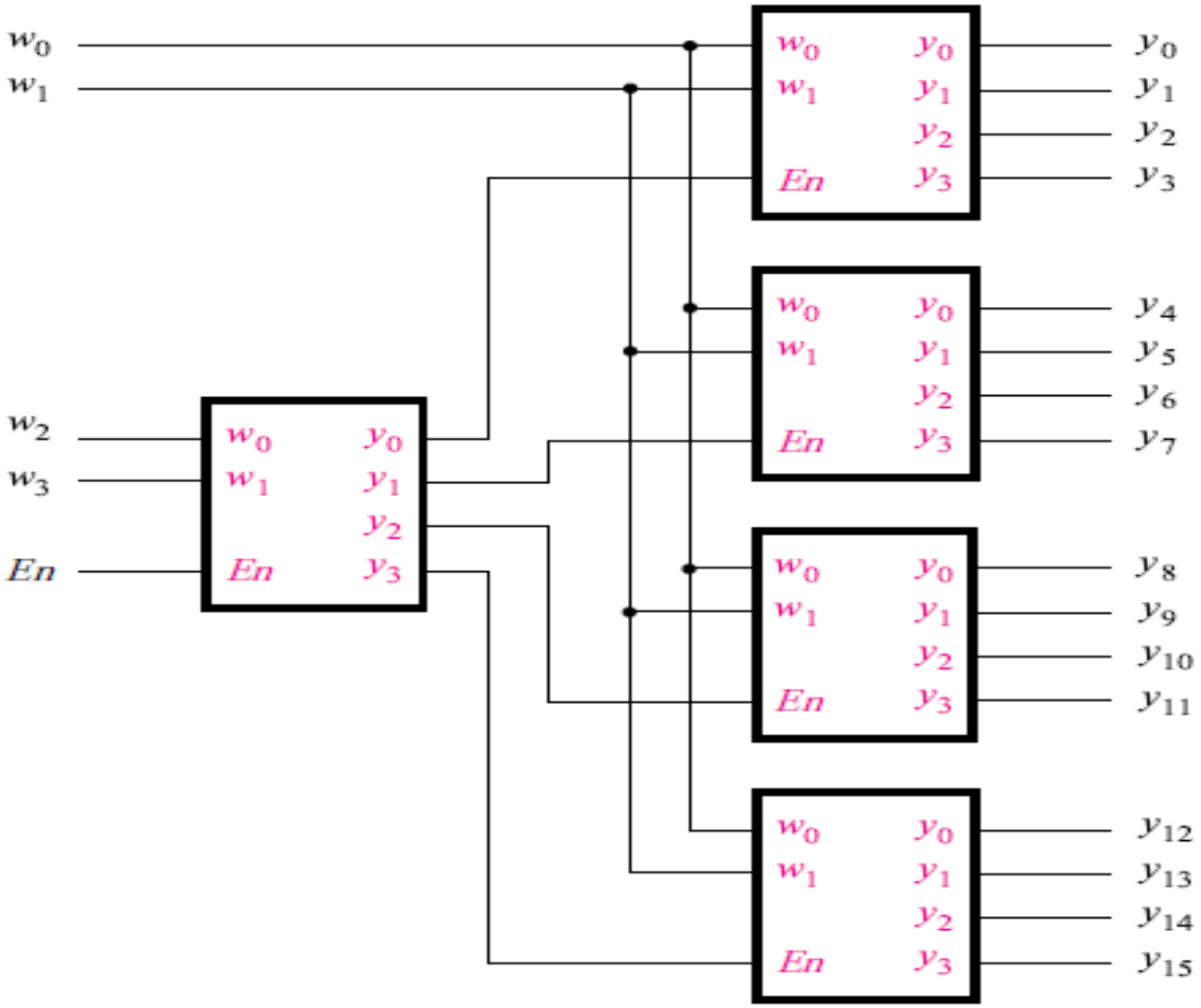



**Figure 6.18** A 4-to-16 decoder built using a decoder tree.

- Decoders are useful for many practical purposes.
- The sum-of- products implementation of the 4-to-1 multiplexer, which requires AND gates to distinguish the four different valuations of the select inputs $s1$ and $s0$.
- Since a decoder evaluates the values on its inputs, it can be used to build a multiplexer as illustrated in **Figure 6.19.**
- The enable input of the decoder is not needed in this case, and it is set to 1.
- The four outputs of the decoder represent the four valuations of the select inputs.

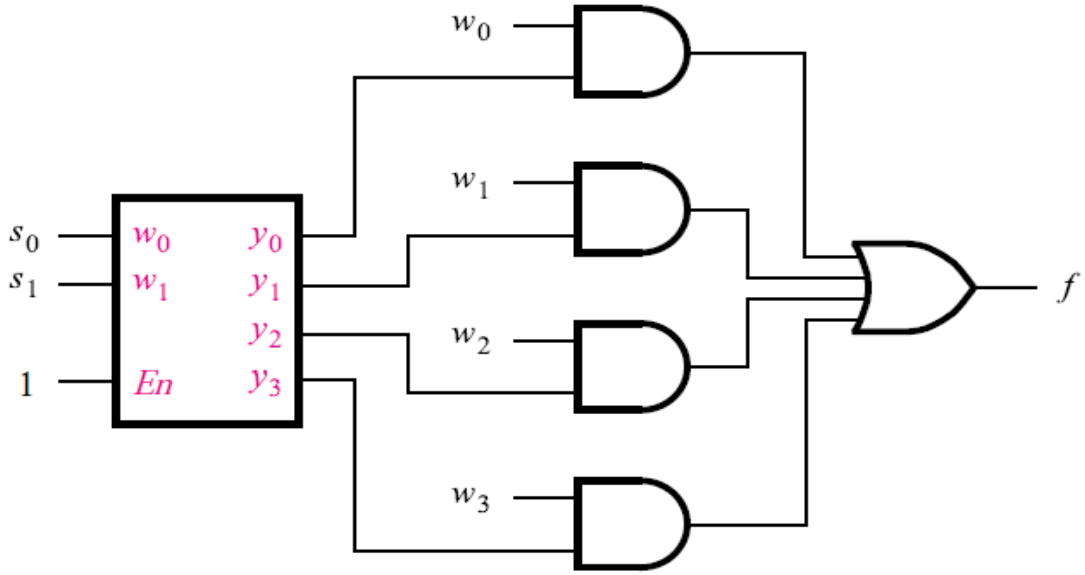



**Figure 6.19** A 4-to-1 multiplexer built using a decoder.

# Encoders

- An encoder performs the opposite function of a decoder.
- It encodes given information into a more compact form.

## ❑ Binary Encoders

- A binary encoder encodes information from 2n inputs into an n-bit code, as indicated in **Figure 6.22**.

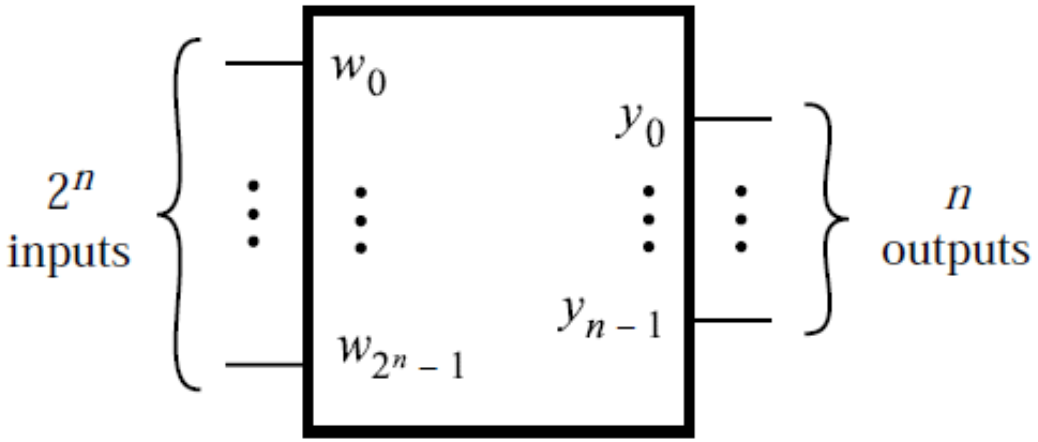
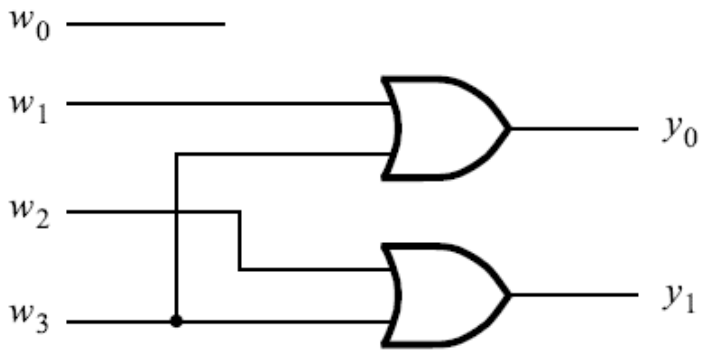



**Figure 6.22** A $2^n$-to-$n$ binary encoder.

- Exactly one of the input signals should have a value of 1, and the outputs present the binary number that identifies which input is equal to 1.
- The truth table for a 4-to-2 encoder is provided in **Figure 6.23*a*.**
- Observe that the output *y*0 is 1 when either input *w*1 or *w*3 is 1, and output *y*1 is 1 when input *w*2 or *w*3 is 1.
- Hence these outputs can be generated by the circuit in Figure 6.23*b*.
- **Note:** We assume that the inputs are one-hot encoded. All input patterns that have multiple inputs set to 1 are not shown in the truth table, and they are treated as don't-care conditions.

| $w_3$ | $w_2$ | $w_1$ | $w_0$ | $y_1$ | $y_0$ |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 |

(a) Truth table




(b) Circuit

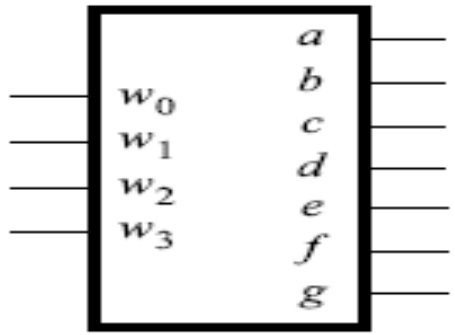
**Figure 6.23** A 4-to-2 binary encoder.

# Encoder Uses

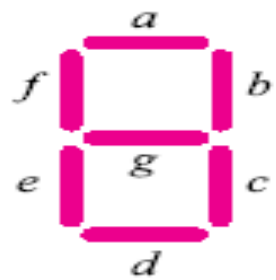
- Encoders are used to reduce the number of bits needed to represent given information.
- A practical use of encoders is for transmitting information in a digital system.
- Encoding the information allows the transmission link to be built using fewer wires.
- Encoding is also useful if information is to be stored for later use because fewer bits need to be stored.

# Code Converters

- The purpose of the decoder and encoder circuits is to convert from one type of input encoding to a different output encoding.
- For example, a 3-to-8 binary decoder converts from a binary number on the input to a one-hot encoding at the output. An 8-to-3 binary encoder performs the opposite conversion.
- There are many other possible types of code converters.
- One common example is a **BCD-to-7-segment decoder**, which converts **one binary-coded decimal (BCD) digit** into information **suitable for driving a digit-oriented display.**
- As illustrated in Figure 6.25*a*, the circuit converts the BCD digit into seven signals that are used to drive the segments in the display.
- Each segment is a small **light-emitting diode (LED),** which glows when driven by an electrical signal. The segments are labeled from *a* to *g* in the figure.
- The truth table for the BCD-to-7-segment decoder is given in **Figure 6.25*c*.**




(a) Code converter




(b) 7-segment display

| $w_3$ | $w_2$ | $w_1$ | $w_0$ | $a$ | $b$ | $c$ | $d$ | $e$ | $f$ | $g$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

(c) Truth table

**Figure 6.25** A BCD-to-7-segment display code converter.

- For each valuation of the inputs $w3, \ldots, w0$, the seven outputs are set to display the appropriate BCD digit.
- Note that the last 6 rows of a complete 16-row truth table are not shown.
- They represent don't-care conditions because they are not legal BCD codes and will never occur in a circuit that deals with BCD data.
- Finally, we should note that although the word ***decoder*** is traditionally used for this circuit, a more appropriate term is ***code converter*****.** The term ***decoder*** **is more appropriate for circuits that produce one-hot encoded outputs.**

# Arithmetic Comparison Circuits

- Another useful type of arithmetic circuit compares the relative sizes of two binary numbers.
- Such a circuit is called a ***comparator.***
- A magnitude digital Comparator is a combinational circuit that **compares two digital or binary numbers** in order to find out whether one binary number is equal, less than, or greater than the other binary number. We logically design a circuit for which we will have two inputs one for A and the other for B and have three output terminals, one for **A > B** condition, one for **A = B** condition, and one for **A < B** condition.

## ❑ 1-Bit Magnitude Comparator:

- A comparator used to compare two bits is called a single-bit comparator. It consists of two inputs each for two single-bit numbers and three outputs to generate less than, equal to, and greater than between two binary numbers.
- The truth table for a 1-bit comparator is given below:

| A | B | A<B | A=B | A>B |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |

- From the above truth table logical expressions for each output can be expressed as follows:

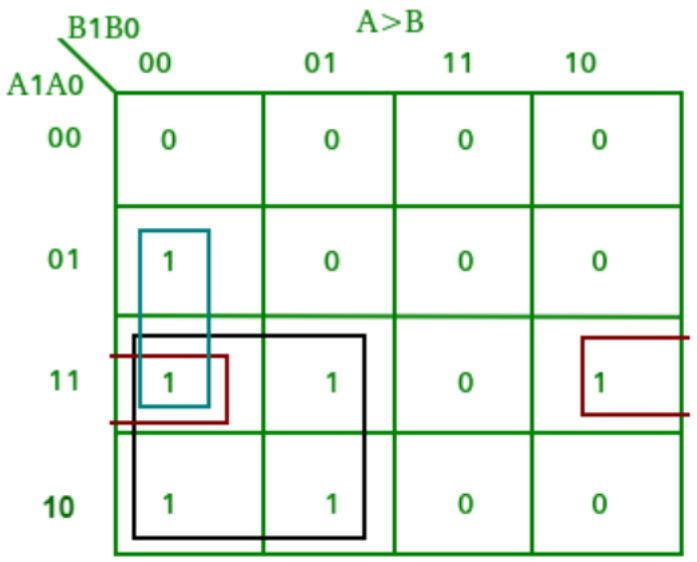
```
A>B: AB'
A<B: A'B
A=B: A'B' + AB
```

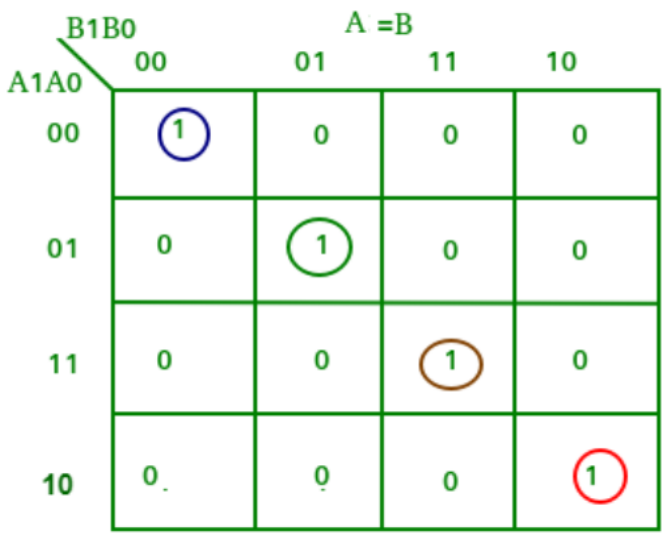
## ❑ 2-Bit Magnitude Comparator:

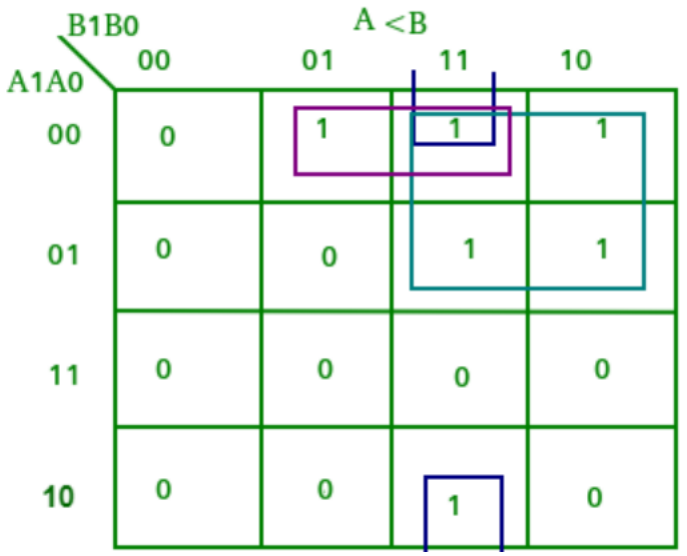
- A comparator used to compare two binary numbers each of two bits is called a 2-bit Magnitude comparator.
- It consists of four inputs and three outputs to generate less than, equal to, and greater than between two binary numbers.
- The truth table for a 2-bit comparator is given below:

| INPUT | | | | OUTPUT | | |
|---|---|---|---|---|---|---|
| A1 | A0 | B1 | B0 | A<B | A=B | A>B |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 |

- From the above truth table K-map for each output can be drawn as follows:



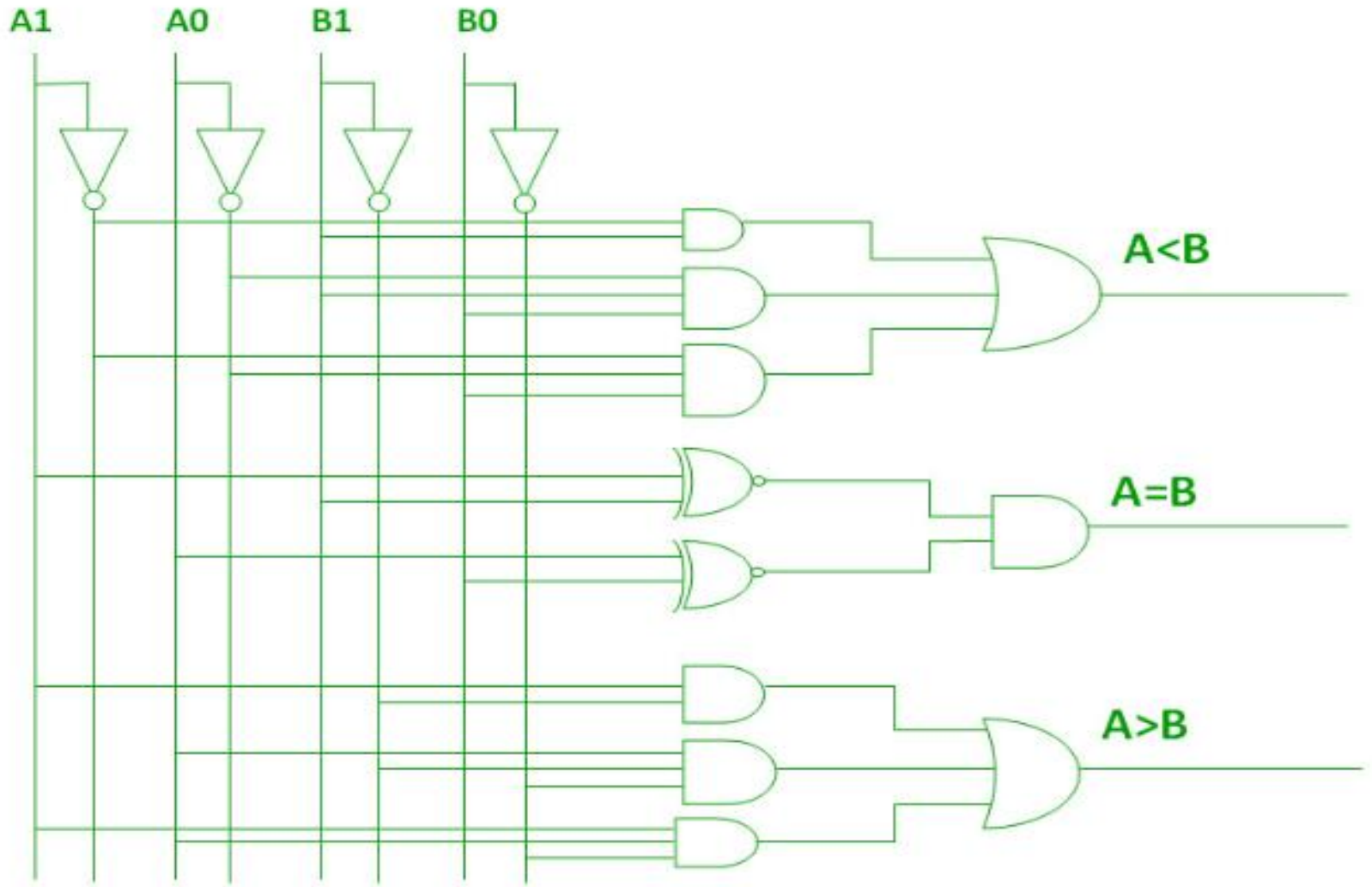




- From the above K-maps logical expressions for each output can be expressed as follows:

```
A>B:A1B1' + A0B1'B0' + A1A0B0'
A=B: A1'A0'B1'B0' + A1'A0B1'B0 + A1A0B1B0 + A1A0'B1B0'
   : A1'B1' (A0'B0' + A0B0) + A1B1 (A0B0 + A0'B0')
   : (A0B0 + A0'B0') (A1B1 + A1'B1')
   : (A0 Ex-Nor B0) (A1 Ex-Nor B1)
A<B:A1'B1 + A0'B1B0 + A1'A0'B0
```

- By using these Boolean expressions, we can implement a logic circuit for this comparator as given below:



## ❑ Applications of Comparators:

1.Comparators are used in central processing units (CPUs) and microcontrollers (MCUs).

2.These are used in control applications in which the binary numbers representing physical variables such as temperature, position, etc. are compared with a reference value.

3.Comparators are also used as process controllers and for Servo motor control.

4.Used in password verification and biometric applications.

# Priority Encoders

- Another useful class of encoders is based on the priority of input signals.
- In a *priority encoder* each input has a priority level associated with it.
- The encoder outputs indicate the active input that has the highest priority.
- When an input with a high priority is asserted, the other inputs with lower priority are ignored.
- The truth table for a 4-to-2 priority encoder is shown in **Figure 6.24.**
- It assumes that $w0$ has the lowest priority and $w3$ the highest.
- The outputs $y1$ and $y0$ represent the binary number that identifies the highest priority input set to 1.
- Since it is possible that none of the inputs is equal to 1, an output, $z$, is provided to indicate this condition.
- It is set to 1 when at least one of the inputs is equal to 1.
- It is set to 0 when all inputs are equal to 0. The outputs $y1$ and $y0$ are not meaningful in this case, and hence the first row of the truth table can be treated as a don't-care condition for $y1$ and $y0$.

| $w_3$ | $w_2$ | $w_1$ | $w_0$ | $y_1$ | $y_0$ | $z$ |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | d | d | 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 |

**Figure 6.24** Truth table for a 4-to-2 priority encoder.

- The behavior of the priority encoder is most easily understood by first considering the last row in the truth table.
- It specifies that if input $w3$ is 1, then the outputs are set to $y1y0 = 11$.
- Because $w3$ has the highest priority level, the values of inputs $w2$, $w1$, and $w0$ do not matter.
- To reflect the fact that their values are irrelevant, $w2$, $w1$, and $w0$ are denoted by the symbol x in the truth table.
- The second-last row in the truth table stipulates that if $w2 = 1$, then the outputs are set to $y1y0 = 10$, but only if $w3 = 0$.
- Similarly, input $w1$ causes the outputs to be set to $y1y0 = 01$ only if both $w3$ and $w2$ are 0.
- Input $w0$ produces the outputs $y1y0 = 00$ only if $w0$ is the only input that is asserted.
- However, a more convenient way to derive the circuit is to define a set of intermediate signals, $i0, \ldots, i3$, based on the observations above.
- Each signal, $ik$ , is equal to 1 only if the input with the same index, $wk$ , represents the highest-priority input that is set to 1.

The logic expressions for $i0, \ldots, i3$ are

$$i_0 = \overline{w}_3\overline{w}_2\overline{w}_1 w_0$$
$$i_1 = \overline{w}_3\overline{w}_2 w_1$$
$$i_2 = \overline{w}_3 w_2$$
$$i_3 = w_3$$

- Using the intermediate signals, the rest of the circuit for the priority encoder has the same structure as the binary encoder in **Figure 6.23,** namely

$$y_0 = i_1 + i_3$$
$$y_1 = i_2 + i_3$$

The output $z$ is given by

$$z = i_0 + i_1 + i_2 + i_3$$