Assignment 2 Part 3

Decoders and Encoders

Lesson Objectives

In this lesson, we will learn about

- Decoders
- Expansion of decoders
- Combinational circuit implementation with decoders
- Some examples of decoders
- Encoders
- Major limitations of encoders
- Priority encoders
- Some examples of ecnoders

Decoders

As its name indicates, a decoder is a circuit component that decodes an input code. Given a binary code of n-bits, a decoder will tell which code is this out of the 2^n possible codes (See Figure 1(a)).

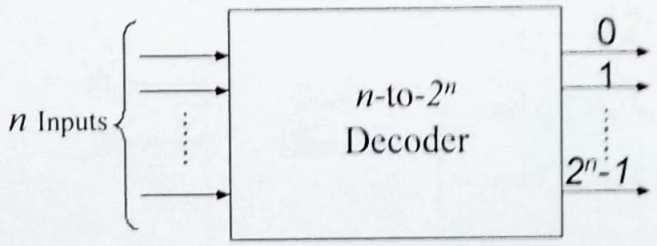


Figure 1(a): A typical decoder

Thus, a decoder has n-inputs and 2^n outputs. Each of the 2^n outputs corresponds to one of the possible 2^n input combinations.

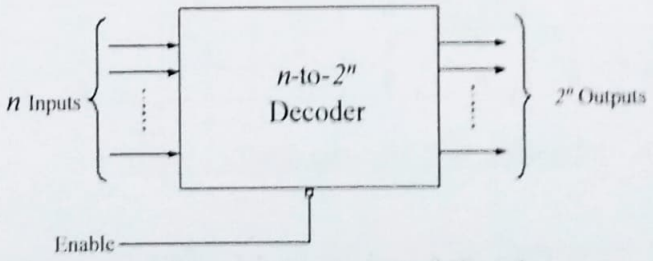


Figure 1(b): A typical decoder

Figure 1(b) shows the block diagram of a typical decoder, which has n input lines, and m output lines, where m is equal to 2^n. The decoder is called n-to-m decoder. Apart from this, there is also a single line connected to the decoder called enable line. The operations of the enable line will be discussed in the flowing text.

- In general, output i equals 1 if and only if the input binary code has a value of i.
- Thus, each output line equals 1 at only one input combination but is equal to 0 at all other combinations.
- In other words, each decoder output corresponds to a minterm of the n input variables.
- Thus, the decoder generates all of the 2^n minterms of n input variables.

Example: 2-to-4 decoders

Let us discuss the operation and combinational circuit design of a decoder by taking the specific example of a 2-to-4 decoder. It contains two inputs denoted by A_1 and A_0 and four outputs denoted by D_0, D_1, D_2, and D_3 as shown in figure 2. Also note that A_1 is the MSB while A_0 is the LSB.

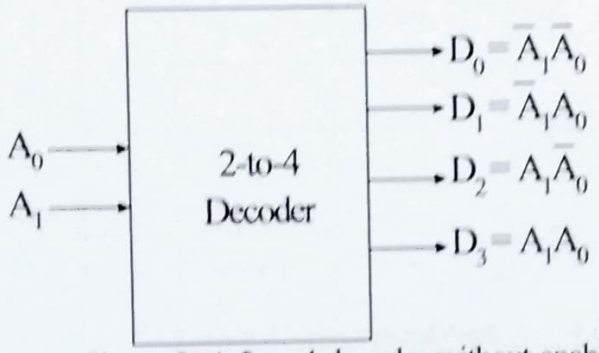


Figure 2: A 2-to-4 decoder without enable

Decimal #	Input		Output			
	A_1	A_0	D_0	D_1	D_2	D_3
0	0	0	1	0	0	0
1	0	1	0	1	0	0
2	1	0	0	0	1	0
3	1	1	0	0	0	1

Table 1: Truth table for 2-to-4 decoder

As we see in the truth table (table 1), for each input combination, one output line is activated, that is, the output line corresponding to the input combination becomes 1, while other lines remain inactive. For example, an input of 00 at the input will activate line D_0. 01 at the input will activate line D_1, and so on.

- Notice that, each output of the decoder is actually a minterm resulting from a certain combination of the inputs, that is
  - $D_0 = \overline{A_1}\,\overline{A_0}$, (minterm m_0) which corresponds to input 00
  - $D_1 = \overline{A_1}\,A_0$, (minterm m_1) which corresponds to input 01
  - $D_2 = A_1\,\overline{A_0}$, (minterm m_2) which corresponds to input 10
  - $D_3 = A_1\,A_0$, (minterm m_3) which corresponds to input 11

- This is depicted in Figures 2 where we see that each input combination will inovke the corresponding output, where each output is minterm corresponding to the input combination.

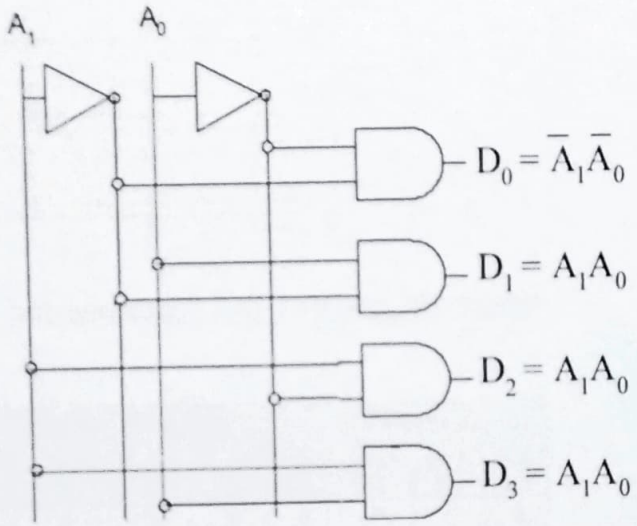


Figure 3: Implementation 2-to-4 decoder

The circuit is implemented with AND gates, as shown in figure 3. In this circuit we see that the logic equation for D_2 is $A_1 \overline{A_0}$. D_0 is $\overline{A_1}\,\overline{A_0}$, and so on. These are in fact the minterms being implemented. Thus, each output of the decoder generates a minterm corresponding to the input combination.

The "enable" input in decoders

Generally, decoders have the "enable" input .The enable input perroms no logical operation, but is only responsible for making the decoder ACTIVE or INACTIVE.

- If the enable "E"
  - is zero, then all outputs are zero regardless of the input values.
  - is one, then the decoder performs its normal operation.

For example, consider the 2-to-4 decoder with the enable input (Figure 4). The enable input is only responsible for making the decoder active or inactive. If Enable E is zero, then all outputs of the decoder will be zeros, regardless of the values of A_1 and A_0. However, if E is 1, then the decoder will perform its normal operation, as is shown in the

truth table (table 2). In this table we see that as long as E is zero, the outputs D_0 to D_3 will remain zero, no matter whatever value you provide at the inputs A_1 A_0, depicted by two don't cares. When E becomes 1, then we see the same behavior as we saw in the case of 2-to-4 decoder discussed earlier.

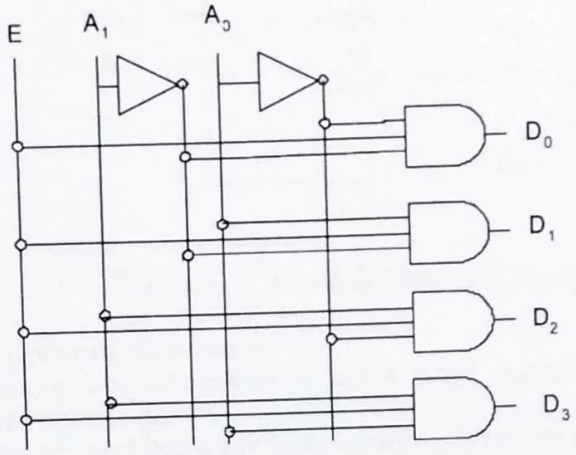


Figure 4: Implementation 2-to-4 decoder with enable

Decimal value	Enable	Inputs		Outputs			
	E	A_1	A_0	D_0	D_1	D_2	D_3
	0	X	X	0	0	0	0
0	1	0	0	1	0	0	0
1	1	0	1	0	1	0	0
2	1	1	0	0	0	1	0
3	1	1	1	0	0	0	1

Table 2: Truth table of 2-to-4 decoder with enable

Example: 3-to-8 decoders

In a three to eight decoder, there are three inputs and eight outputs, as shown in figure 5. A_0 is the least significant variable, while A_2 is the most significant variable.

The three inputs are decoded into eight outputs. That is, binary values at the input form a combination, and based on this combination, the corresponding output line is activated.

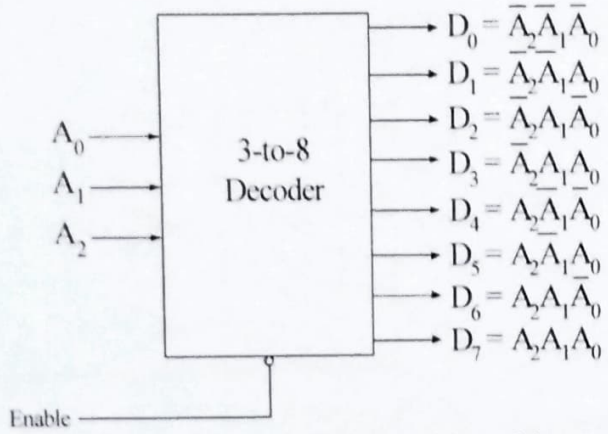


Figure 5: A 3-to-8 decoder with enable

Each output represents one minterm .

- For example, for input combination $A_2A_1A_0 = 001$, output line D_1 equals 1 while all other output lines equal 0's
- It should be noted that at any given instance of time, one and only one output line can be activated. It is also obvious from the fact that only one combination is possible at the input at a time, so the corresponding output line is activated.

Dec. Code	Inputs			Outputs							
	A_2	A_1	A_0	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
2	0	1	0	0	0	1	0	0	0	0	0
3	0	1	1	0	0	0	1	0	0	0	0
4	1	0	0	0	0	0	0	1	0	0	0
5	1	0	1	0	0	0	0	0	1	0	0
6	1	1	0	0	0	0	0	0	0	1	0
7	1	1	1	0	0	0	0	0	0	0	1

Table 3: Truth table of 3-to-8 decoder

Since each input combination represents one minterm, the truth table (table 3) contains eight output functions, from D_0 to D_7 seven, where each function represents one and only one minterm. Thus function D_0 is $A_2' A_1' A_0'$. Similarly function D_7 is $A_2A_1A_0$. The corresponding circuit is given in Figure 6. In this figure, the three inverters provide complement of the inputs, and each one of the AND gates generates one of the minterms. It is also possible to add an Enable input to this decoder.

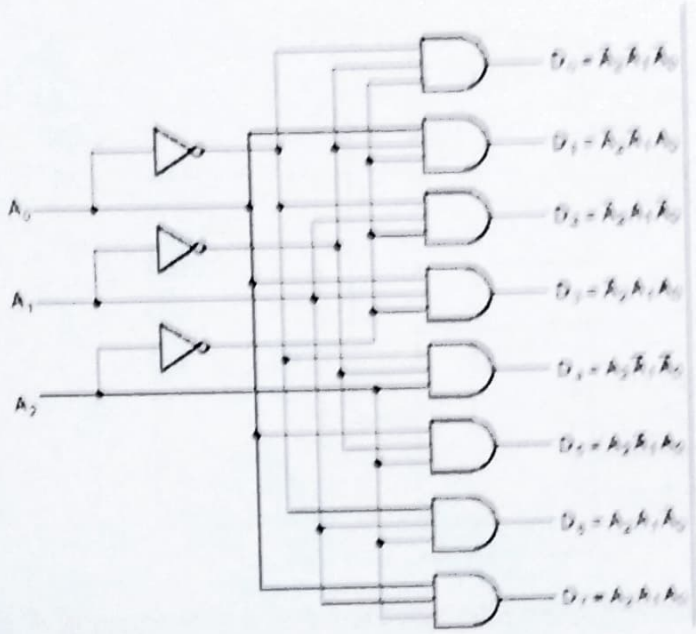


Figure 6: Implementation of a 3-to-8 decoder without enable

Decoder Expansion

- It is possible to build larger decoders using two or more smaller ones.
- For example, a 6-to-64 decoder can be designed with four 4-to-16 decoders and one 2-to-4 line decoder.

Example: Construct a 3-to-8 decoder using two 2-to-4 decoders with enable inputs.

Figure 7 shows how decoders with enable inputs can be connected to form a larger decoder. Two 2-to-4 line decoders are combined to build a 3-to-8 line decoder.

- The two least signifncat bits (i.e. A_1 and A_0) are connected to both decoders
- Most signifcant bit (A_2) is connected to the enable input of one decoder.
- The complement of most significant bit ($\overline{A_2}$) is connected to the enable of the other decoder.
- When $A_2 = 0$, upper decoder is enabled, while the lower is disabled. Thus, the outputs of the upper decoder correspond to minterms D_0 through D_3
- When $A_2 = 1$, upper decoder is disabled, while the lower is enabled. Thus, the outputs of the lower decoder correspond to minterms D_4 through D_7.

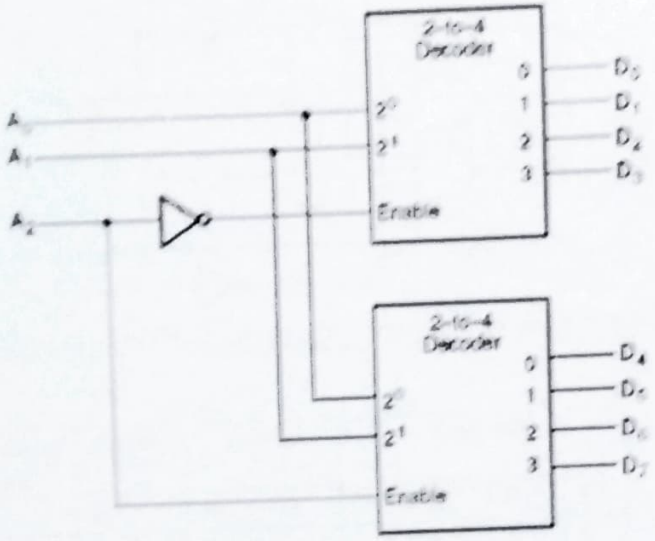


Figure 7: Implementing a 3-to-8 decoder with two 2-to-4 decoders

Decoder design with NAND gates

- Some decoders are constructed with NAND rather than AND gates.
- In this case, all decoder outputs will be 1's except the one corresponding to the input code which will be 0.

Decimal #	Input		Output			
	A_1	A_0	D_0'	D_1'	D_2'	D_3'
0	0	0	0	1	1	1
1	0	1	1	0	1	1
2	1	1	1	1	0	1
3	1	1	1	1	1	0

$$\overline{D}_0 = \overline{\overline{A}_1\overline{A}_0} \qquad \overline{D}_1 = \overline{\overline{A}_1 A_0}$$

$$\overline{D}_2 = \overline{A_1\overline{A}_0} \qquad \overline{D}_3 = \overline{A_1 A_0}$$

Table 4: Truth table of 2-to-4 decoder with NAND gates

This decoder can be constructed without enable, similar to what we have seen in the design of decoder with AND gates, without enable. The truth table and corresponding minterms are given in table 4. Notice that the minters are in the complemented form.

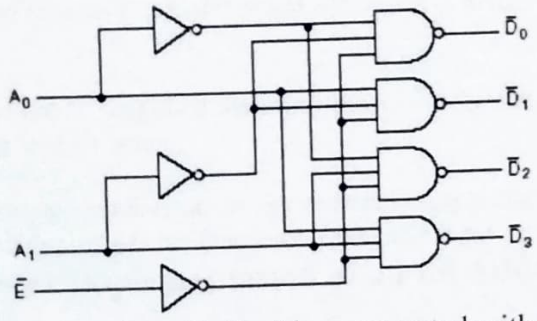


Figure 8: A 2-to-4 decoder with Enable constructed with NAND gates.

Decimal value	Enable	Inputs		Outputs			
	E'	A_1	A_0	D_0'	D_1'	D_2'	D_3'
	1	X	X	1	1	1	1
0	0	0	0	0	1	1	1
1	0	0	1	1	0	1	1
2	0	1	1	1	1	0	1
3	0	1	1	1	1	1	0

$$\overline{D}_0 = \overline{E\overline{A}_1\overline{A}_0} \qquad \overline{D}_1 = \overline{E\overline{A}_1 A_0}$$
$$\overline{D}_2 = \overline{EA_1\overline{A}_0} \qquad \overline{D}_3 = \overline{EA_1 A_0}$$

Table 5: Truth table of 2-to-4 decoder with Enable using NAND gates

A 2-to-4 line decoder with an enable input constructed with NAND gates is shown in figure 8. The circuit operates with complemented outputs and enable input E' is also complemented to match the outputs of the NAND gate decoder. The decoder is enabled when E' is equal to zero. As indicated by the truth table, only one output can be equal to zero at any given time, all other outputs being equal to one. The output with the value of zero represents the minterm selected by inputs A_1 and A_0. The circuit is disabled when E' is equal to one, regardless of the values of the other two inputs. When the circuit is disabled, none of the outputs are equal to zero, and none of the minterms are selected. The corresponding logic equations are also given in table 5.

Combinational circuit implementation using decoder

- As known, a decoder provides the 2^n minterms of n input variables
- Since any boolean functions can be expressed as a sum of minterms, one can use a decoder to implement any function of n variables.
- In this case, the decoder is used to generate the 2^n minterms and an additional OR gate is used to generate the sum of the required minterms.
- In this way, any combinational circuit with n inputs and m outputs can be

implemented using an n-to-2^n decoder in addition to m OR gates.

- Remember, that
  - The function need not be simplified since the decoder implements a function using the minterms, not product terms.
  - Any number of output functions can be implemented using a single decoder, provided that all those outputs are functions of the same input variables.

Example: Decoder Implementation of a Full Adder

Let us look at the truth table (table 6) for the given problem. We have two outputs, called S, which stands for sum, and C, which stands for carry. Both sum and carry are functions of X, Y, and Z.

Decimal value	Input			Output	
	X	Y	Z	S	C
0	0	0	0	0	0
1	0	0	1	1	0
2	0	1	0	1	0
3	0	1	1	0	1
4	1	0	0	1	0
5	1	0	1	0	1
6	1	1	0	0	1
7	1	1	1	1	1

Table 6: Truth table of the Full Adder

- The output functions S & C can be expressed in sum-of-minterms forms as follows:
  - $S(X,Y,Z) = \Sigma m(1,2,4,7)$
  - $C(X,Y,Z) = \Sigma m(3,5,6,7)$

Looking at the truth table and the functions in sum of minterms form, we observe that there are three inputs, X, Y, and Z that correspond to eight minterms. This implies that a 3-to-8 decoder is needed to implement this function. This implementation is given in Figure 9, where the sum S is implemented by taking minterms 1, 2, 4, and 7 and the OR gates forms the logical sum of minterm for S. Similarly, carry C is implemented by taking logical sum of minterms 3, 5, 6, and 7 from the same decoder.

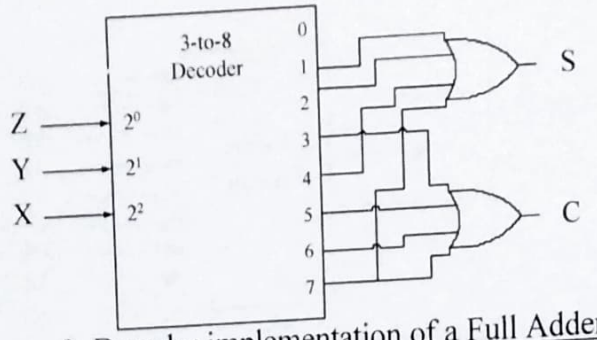


Figure 9: Decoder implementation of a Full Adder

Encoders

- An encoder performs the inverse operation of a decoder, as shown in Figure 10.
- It has 2^n inputs, and n output lines.
- Only one input can be logic 1 at any given time (active input). All other inputs must be 0's.
- Output lines generate the binary code corresponding to the active input.

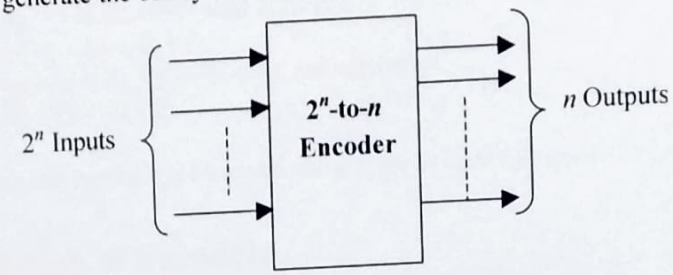


Figure 10: A typical Encoder

Example: Octal-to-binary encoder

We will use 8-to-3 encoder (Figure 11) for this problem, since we have eight inputs, one for each of the octal digits, and three outputs that generate the corresponding binary number. Thus, in the truth table, we see eight input variables on the left side of the vertical lines, and three variables on the right side of the vertical line (table 7).

Inputs								Outputs			Decimal Code
E7	E6	E5	E4	E3	E2	E1	E0	A2	A1	A0	
0	0	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	0	1	0	0	0	1	0	2
0	0	0	0	1	0	0	0	0	1	1	3
0	0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	0	0	0	0	1	0	1	5
0	1	0	0	0	0	0	0	1	1	0	6
1	0	0	0	0	0	0	0	1	1	1	7

Table 7: Truth table of Octal-to-binary encoder

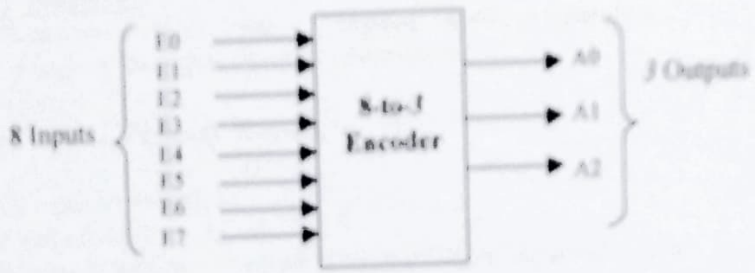


Figure 11: Octal-to-binary encoder

- Note that not all input combinations are valid.
- Valid combinations are those which have exactly one input equal to logic 1 while all other inputs are logic 0's.
- Since, the number of inputs = 8, K-maps cannot be used to derive the output Boolean expressions.
- The encoder implementation, however, can be directly derived from the truth table
  - Since $A_0 = 1$ if the input octal digit is 1 or 3 or 5 or 7, then we can write:
    $A_0 = I_1 + I_3 + I_5 + I_7$
  - Likewise, $A_1 = I_2 + I_3 + I_6 + I_7$, and similarly
  - $A_2 = I_4 + I_5 + I_6 + I_7$
- Thus, the encoder can be implemented using three 4-input OR gates.

Major Limitation of Encoders

- Exactly one input must be active at any given time.
- If the number of active inputs is less than one or more than one, the output will be incorrect.
- For example, if $I_3 = I_6 = 1$, the output of the encoder $A_2A_1A_0 = 111$, which implies incorrect output.

Two Problems to Resolve:

1. If two or more inputs are active at the same time, *what should the output be?*
2. An output of all 0's is generated in 2 cases:
   - when all inputs are 0
   - when I_0 is equal to 1.

   How can this ambiguity be resolved?

Solution To Problem 1:

- Use a *Priority Encoder* which produces the output corresponding to the input with higher priority.
- Inputs are assigned priorities according to their subscript value, e.g. higher subscript inputs are assigned higher priority.
- In the previous example, if $I_3 = I_6 = 1$, the output corresponding to I_6 will be produced ($A_2A_1A_0 = 110$) since I_6 has higher priority than I_3.

***Solution To Problem 2*:**

- Provide one more output signal V to indicate *validity* of input data.
- $V = 0$ if none of the inputs equals 1, otherwise it is 1

Example: 4-to-2 Priority Encoders

- Sixteen input combinations
- Three output variables A1, A0, and V
- V is needed to take care of situation when all inputs are equal to zero.

Inputs				Outputs			
E3	E2	E1	E0	A1	A0	V	
0	0	0	0	X	X	0	Invalid Input
0	0	0	1	0	0	1	
0	0	1	0	0	1	1	
0	0	1	1	0	1	1	
0	1	0	0	1	0	1	
0	1	0	1	1	0	1	
0	1	1	0	1	0	1	
0	1	1	1	1	0	1	
1	0	0	0	1	1	1	
1	0	0	1	1	1	1	
1	0	1	0	1	1	1	
1	0	1	1	1	1	1	
1	1	0	0	1	1	1	
1	1	0	1	1	1	1	
1	1	1	0	1	1	1	
1	1	1	1	1	1	1	

Table 8: Truth table of 4-to-2 Priority Encoder

In the truth table (table 8), we have sixteen input combinations. In the output, we have three variables. The variable V is needed to take care of the situation where all inputs are zero. In that case V is kept at zero, regardless of the values of A_1 and A_0. This combination is highlighted green. In all other cases, V is kept at 1, because at least one of the inputs is one.

When E_0 is 1, the output combination of A_1 and A_0 is 00. This combination is highlighted blue.

Then we have two combinations highlighted yellow. In both these combinations, A_1 and A_0 are 01. This is because in both these combinations E_1 is 1, regardless of the value of E_0, and since E_1 has higher subscript, the corresponding output value is 01.

This is followed by four input combinations in pink. In these four combinations, the output A_1A_0 is 10, since E_2 is 1 in all these combinations, and E_2 has the highest

precedence compared to E_0 and E_1. Although E_0 and E_1 are also having a value of one in this set of four combinations, but they do not have the priority.

Finally we have the last eight input combinations, whose output is 11. This is because E3 is the highest priority input, and it is equal to 1. Though the other inputs with smaller subscripts, namely, E2, E1, and E0 are also having values of one in some combinations, but they do not have the priority.

The truth table can be rewritten in a more compact form using don't care conditions for inputs as shown below in table 9.

	Inputs				Outputs		
	E3	E2	E1	E0	A1	A0	V
1	0	0	0	0	X	X	0
2	0	0	0	1	0	0	1
3	0	0	1	X	0	1	1
4	0	1	X	X	1	0	1
5	1	X	X	X	1	1	1

Table 9: Truth table of 4-to-2 priority encoder (compact form)

- With 4 Input variables, the truth table must have 16 rows, with each row representing an input combination.
- With don't care input conditions, the number of rows can be reduced since rows with don't care inputs will actually represent more than one input combination.
- Thus, for example, row # 3 represents 2 combinations since it represents the input conditions $E_3E_2E_1E_0$=0010 and 0011.
- Likewise, row # 4 represents 4 combinations since it represents the input conditions $E_3E_2E_1E_0$=0100, 0101, 0110 and 0111.
- Similarly, row # 5 represents 8 combinations.
- Thus, the total number of input combinations represented by the 5-row truth table = 1+ 1+ 2+ 4 + 8= 16 input combinations.

Boolean Expressions for V, A_1 and A_0 and the circuit:

See next page:

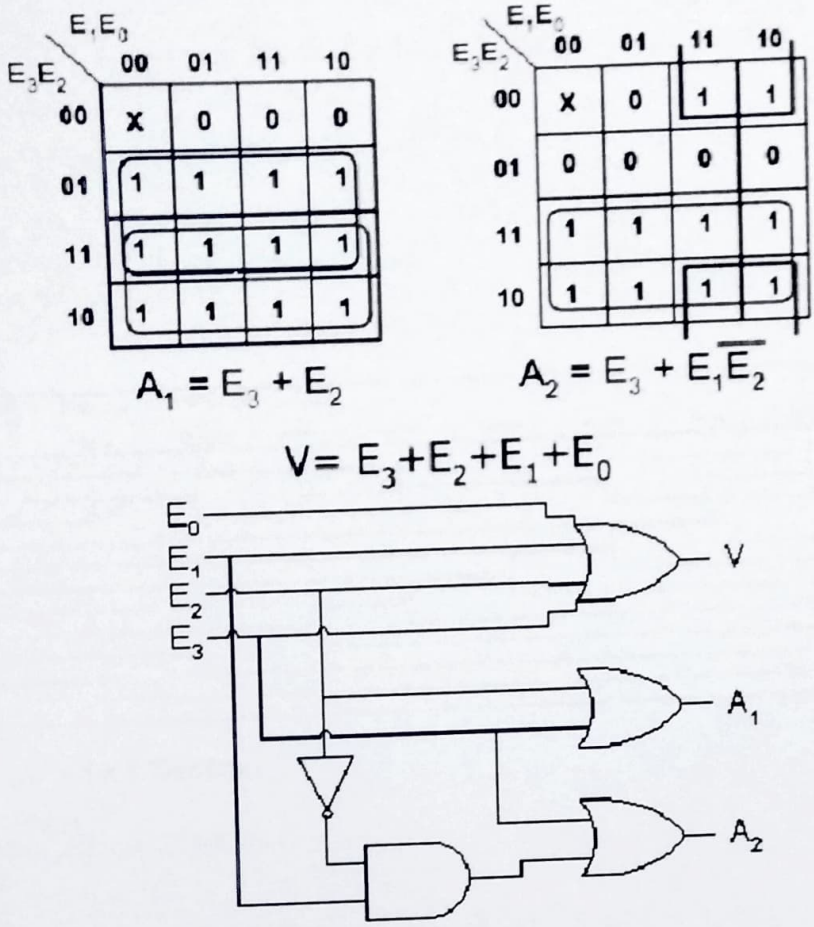


Figure 12: Equations and circuit for 4-to-2 priority encoder

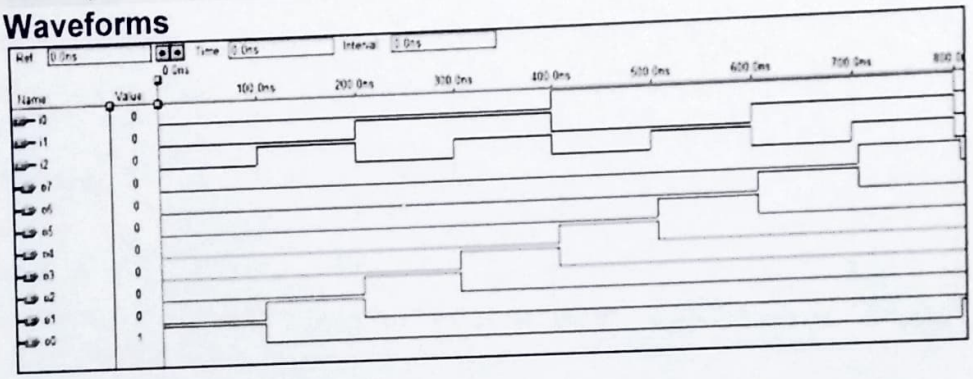
VHDL Code for a 8 x 3 Encoder

```
library ieee;
use ieee.std_logic_1164.all;

entity enc is
   port(i0,i1,i2,i3,i4,i5,i6,i7:in bit; o0,o1,o2: out bit);
end enc;

architecture vcgandhi of enc is
begin
   o0<=i4 or i5 or i6 or i7;
   o1<=i2 or i3 or i6 or i7;
   o2<=i1 or i3 or i5 or i7;
end vcgandhi;
```

Waveforms



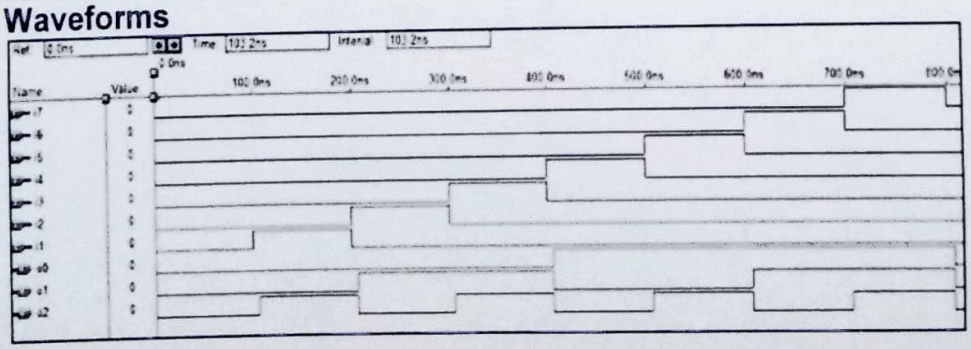
VHDL Code for a 3 x 8 Decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity dec is
   port(i0,i1,i2:in bit; o0,o1,o2,o3,o4,o5,o6,o7: out bit);
end dec;

architecture vcgandhi of dec is
begin
   o0<=(not i0) and (not i1) and (not i2);
   o1<=(not i0) and (not i1) and i2;
   o2<=(not i0) and i1 and (not i2);
   o3<=(not i0) and i1 and i2;
   o4<=i0 and (not i1) and (not i2);
   o5<=i0 and (not i1) and i2;
   o6<=i0 and i1 and (not i2);
   o7<=i0 and i1 and i2;
end vcgandhi;
```

Waveforms



VIEW SOURCE

PRINT

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity decoder is
5 port(
6 a : in STD_LOGIC_VECTOR(1 downto 0);
7 b : out STD_LOGIC_VECTOR(3 downto 0)
8 );
9 end decoder;
10 architecture bhv of decoder is
11 begin
12
13 process(a)
14 begin
15 case a is
16 when "00" => b <= "0001"; when "01" => b <= "0010"; when "10" => b <= "0100"; when "11"
   => b <= "1000";
17 end case;
18 end process;
19
20 end bhv;
```

Truth table

Input		Output			

View source

print?

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity decoder1 is
5  port(
6  a : in STD_LOGIC_VECTOR(1 downto 0);
7  b : out STD_LOGIC_VECTOR(3 downto 0)
8  );
9 end decoder1;
10
11 architecture bhv of decoder1 is
12 begin
13
14 process(a)
15 begin
16  if (a="00") then
17  b <= "0001";
18  elsif (a="01") then
19  b <= "0010";
20  elsif (a="10") then
21  b <= "0100";
22  else
23  b <= "1000";
24  end if;
25 end process;
26
27 end bhv;
```