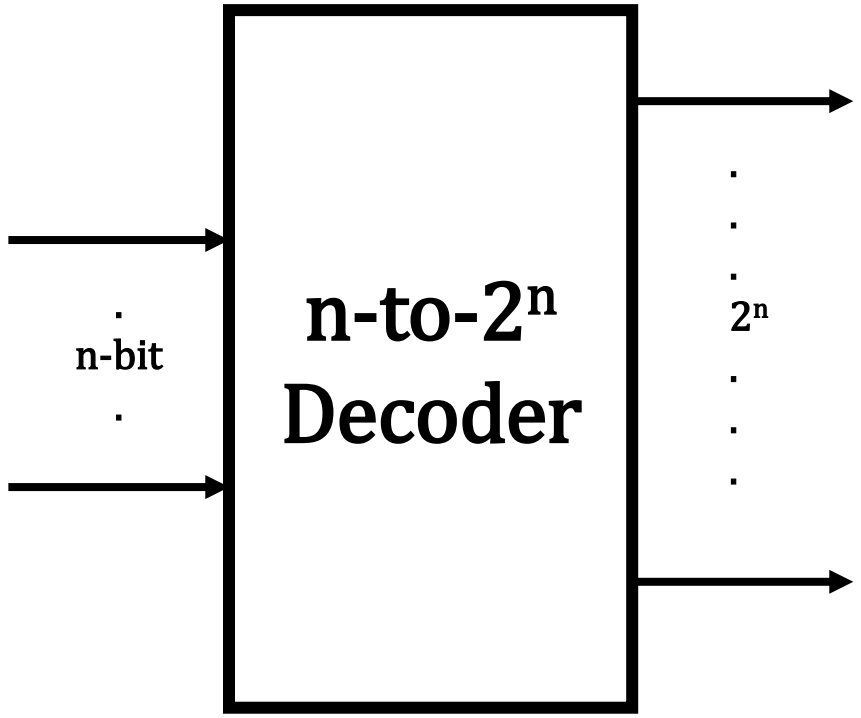
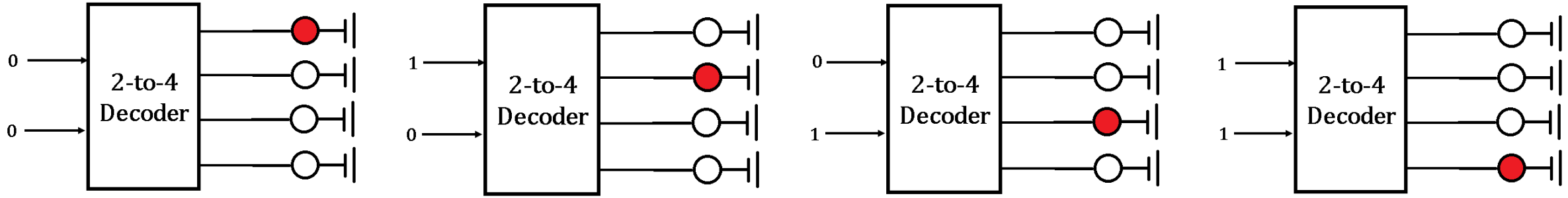
Lecture -6
Combination Circuits-2

Prepared By: Asif Mahfuz

Decoder

- A n-to-2^n decoder takes an n-bit input and produces 2^n outputs.
- The n-bit inputs represent a binary number that determines which of the 2^n outputs is **uniquely true**.
- As the name suggest the job of the decoder is to decode.
- For instance, if you ask the guard of a building, which floor does Mr. Y lives? The guard gives a specific answer based on your question.
- A decoder is commonly used in address decoding.





Decoder Design

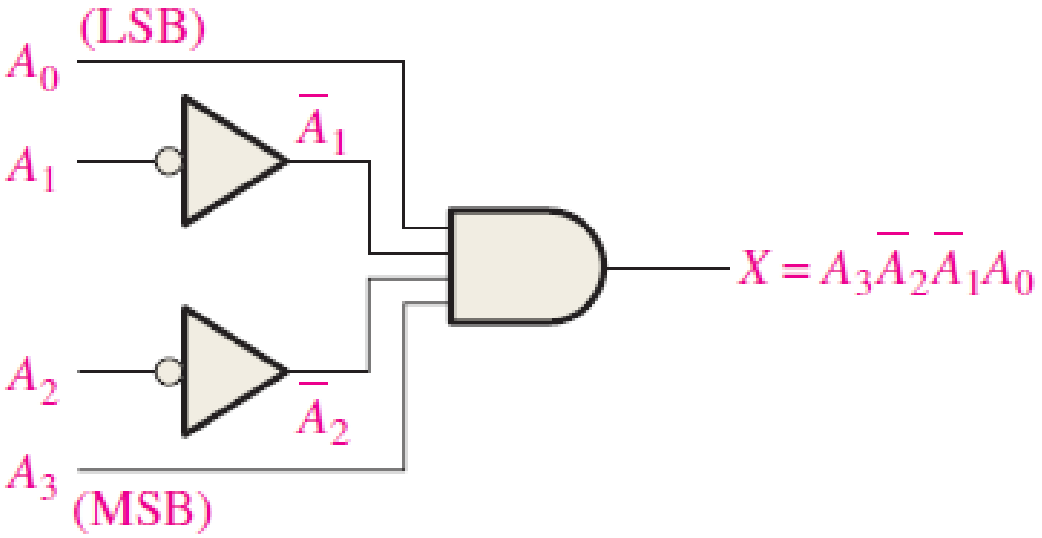
- We have now a superficial idea of how a decoder works.
- A 2-to-4 decoder has two inputs namely, **S0** and **S1** and 4 outputs namely **Q0**, **Q1**, **Q2** and **Q3**.
- The 2-to-4 decoder operates according to the following truth table.
- So how is this decoding done?
- Implementing the min-term for a specific combination is the answer.

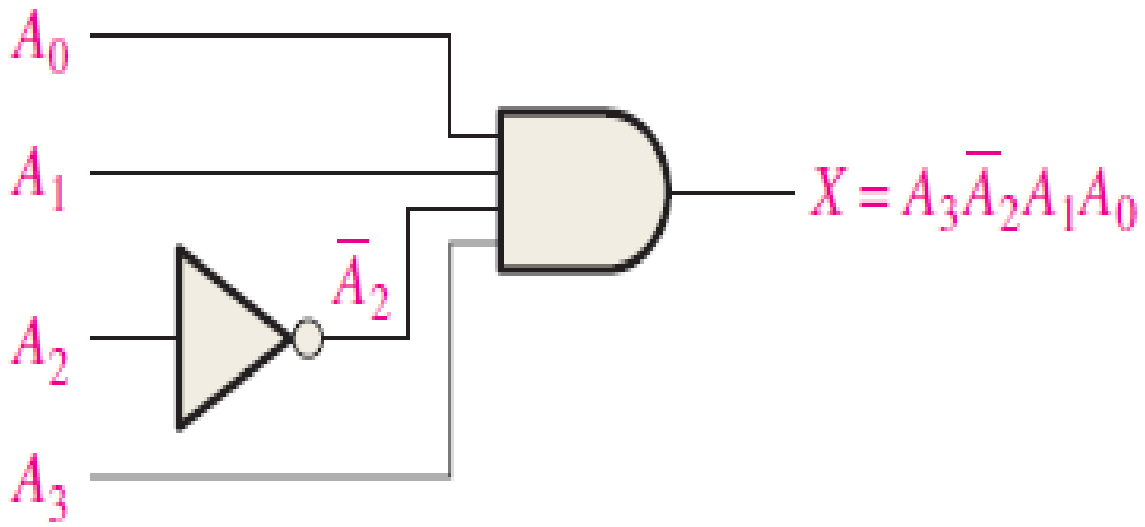
S1	S0	Q0	Q1	Q2	Q3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

S1	S0	Q0	Q1	Q2	Q3
0	0	$\overline{S1}\,\overline{S0}$	0	0	0
0	1	0	$\overline{S1}\,S0$	0	0
1	0	0	0	$S1\,\overline{S0}$	0
1	1	0	0	0	$S1\,S0$

Decoding Operation

- So how do we implement the decoding function.
- Say we want to decode a combination **1001**.
- The min-term of this combination is $\mathbf{A_3\overline{A_2}\,\overline{A_1}A_0}$.
- Now we implement this min-term.
- Now say we want to decode **1011**.
- We need to implement the min-term $\mathbf{A_3\overline{A_2}A_1A_0}$.

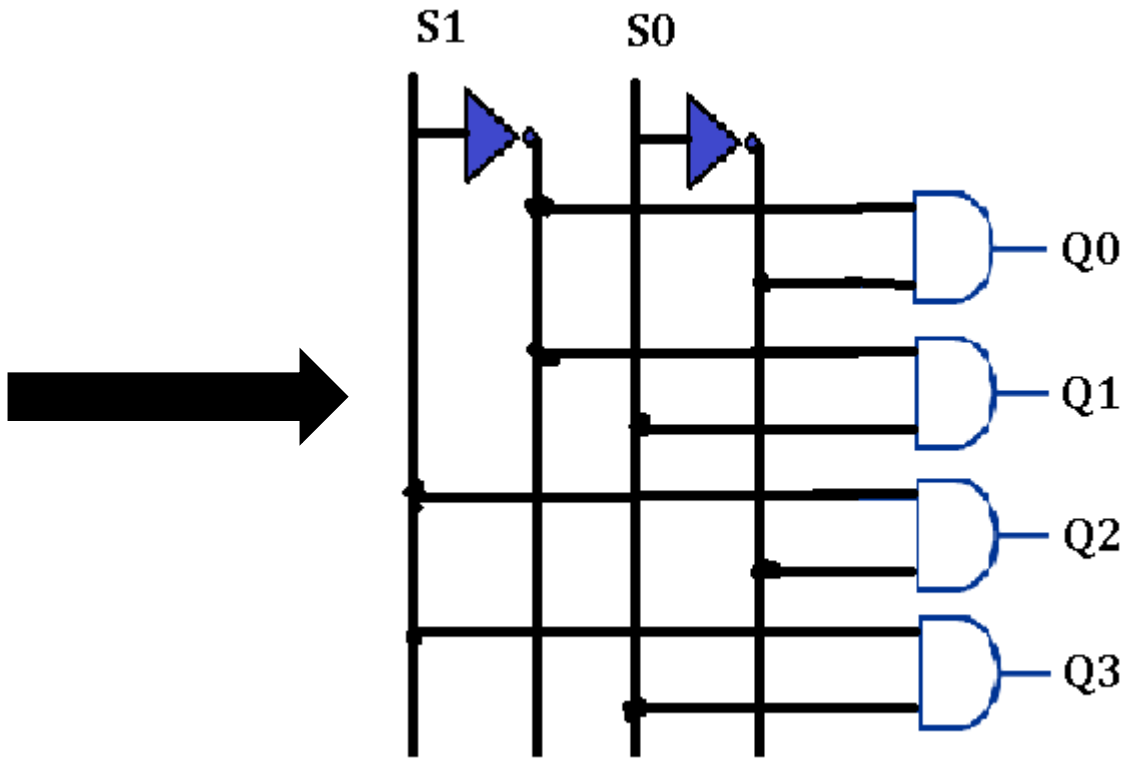




Logical Circuit of 2-to-4 Decoder

- So now that we know how to decode a combination, we can draw the logical circuit of the a 2-to-4 Decoder.
- The truth table of the decoder is as follows.
- From the truth table we can draw the logical circuit.
- As each output is implemented from a min-term, an **Active HIGH** decoder is also called a min term generator.

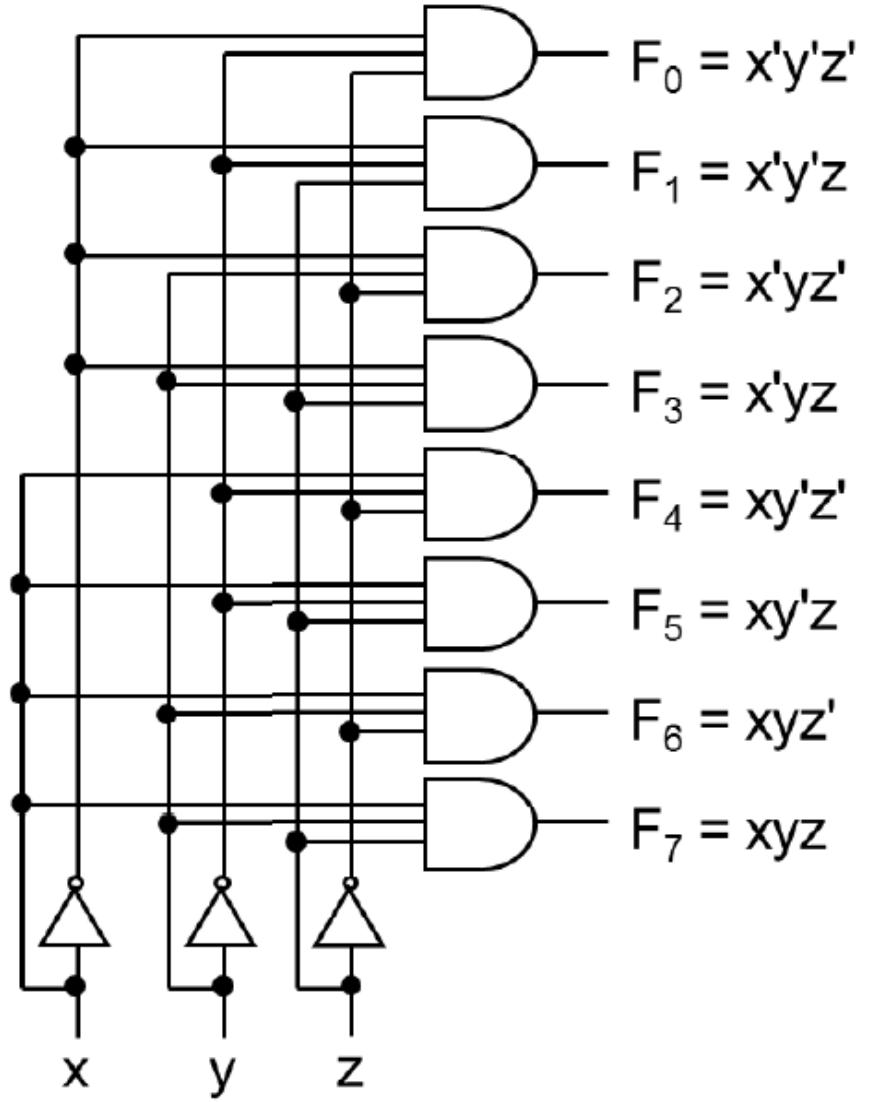
S1	S0	Q0	Q1	Q2	Q3
0	0	$\overline{S1}\,\overline{S0}$	0	0	0
0	1	0	$\overline{S1}\,S0$	0	0
1	0	0	0	$S1\,\overline{S0}$	0
1	1	0	0	0	$S1\,S0$



Design of a 3-to-8 Decoder

- We can design a 3-to-8 decoder following the same procedure.

X	Y	Z	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



Variation of Decoder

- Till now we have only seen the active-HIGH decoders.
- That is, when a combination is given in the input the corresponding output is HIGH and the other outputs are LOW.
- A very common variation of decoders is the active-LOW decoders.
- This means when a combination is given in the input, the corresponding output is LOW and the other outputs are HIGH.
- The same procedure can be followed to design an active-LOW decoder, with the only difference of using a NAND gate instead of the AND gate for decoding.
- Thus active-LOW decoders are also called Max-Term generators.

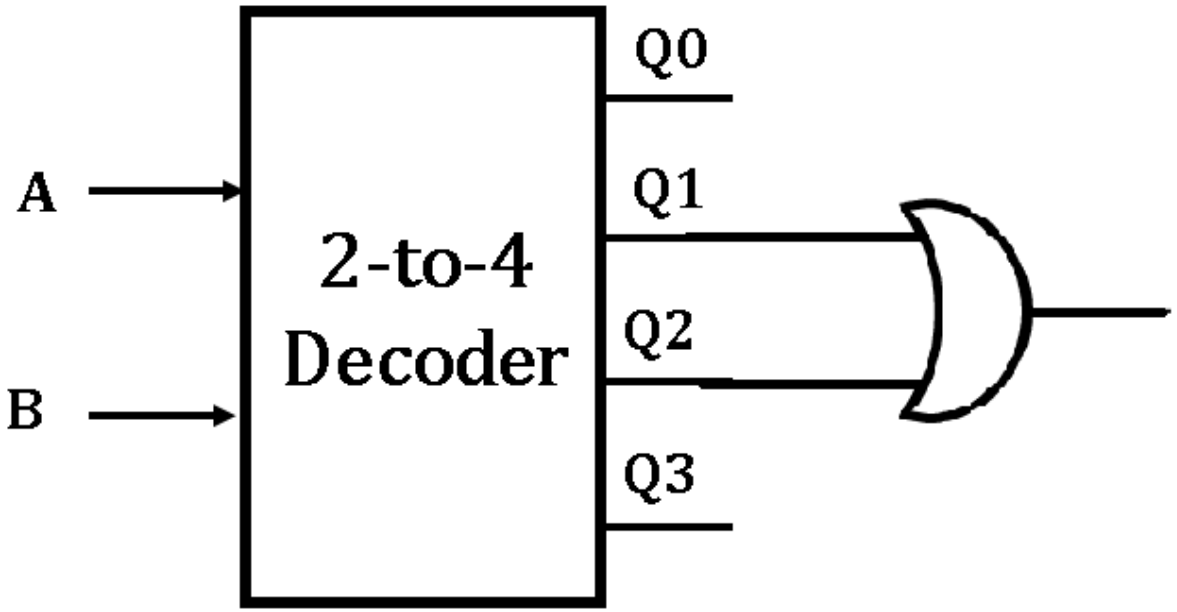
S1	S0	Q0	Q1	Q2	Q3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

S1	S0	Q0	Q1	Q2	Q3
0	0	(S1+S0)	1	1	1
0	1	1	$(S1+\overline{S0})$	1	1
1	0	1	1	$(\overline{S1}+S0)$	1
1	1	1	1	1	$(\overline{S1}+\overline{S0})$

Application of Decoders

- As we have already studied, one of the application of decoders is addressing or selecting memory locations.
- Another very important use of decoders is to implement functions.
- As active-HIGH decoders are min-term generators and active-LOW decoders are max-term generators the SOP form and POS form can be easily implemented with decoders.
- The procedure is very simple, if we have the truth-table of the function or can somehow determine the truth-table, function implementation will be very easy.

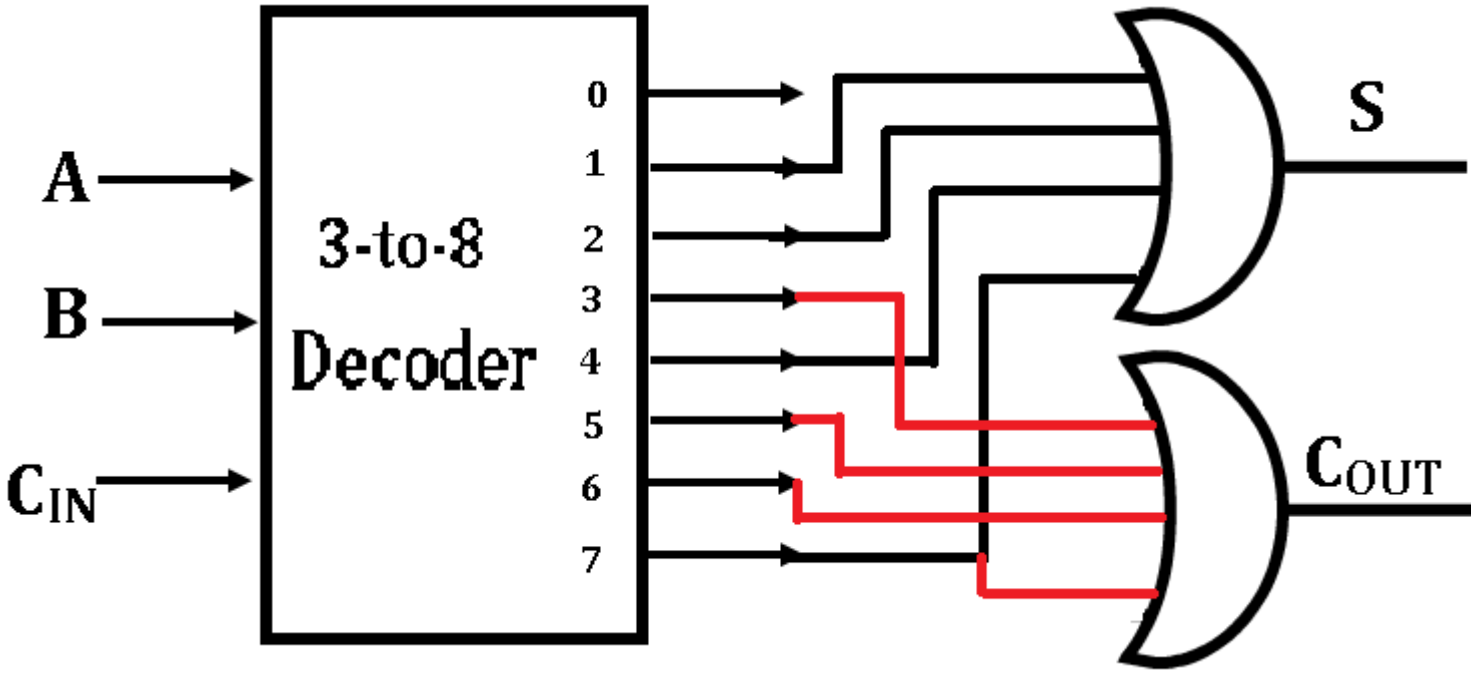
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



Application of Decoders

- A full-adder can be very easily implemented with the help of a decoder.
- The truth-table of a full adder is given below:
- The full adder can be implemented as shown in the diagram.

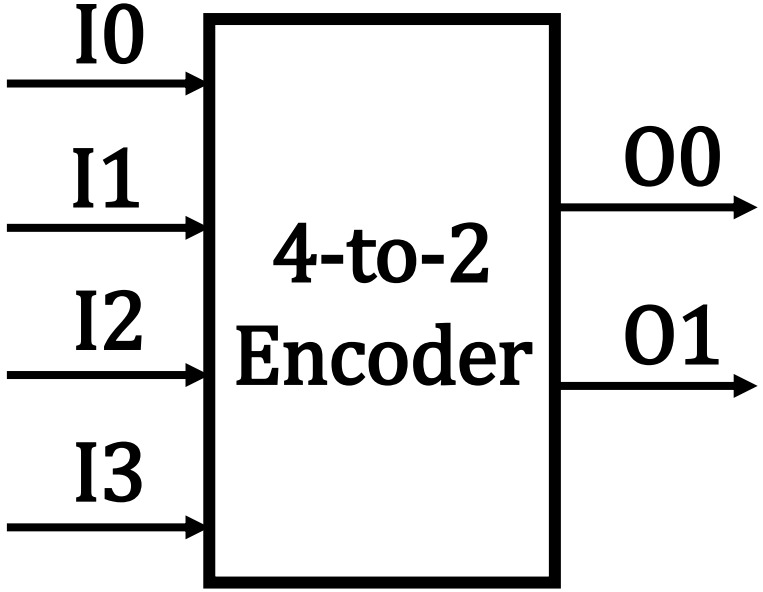
A	B	C_{IN}	C_{out}	$\sum$
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

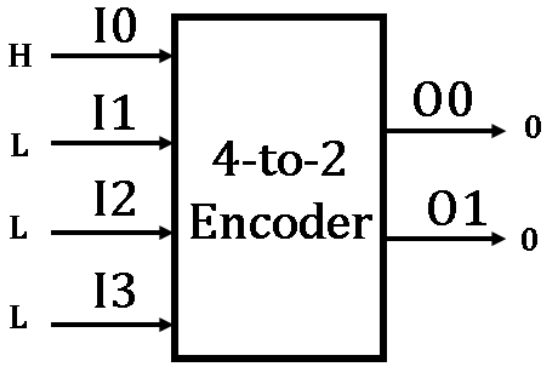


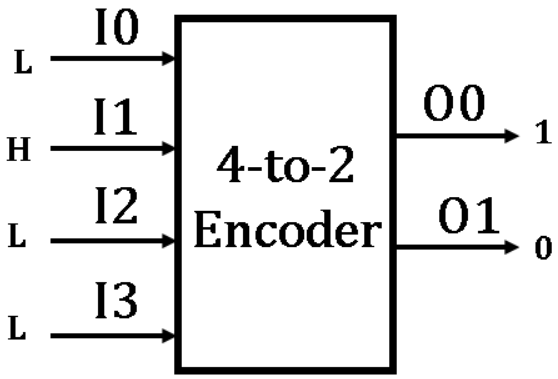
- Implement the function $F = \sum(1,3,4,7)$ using a 3-to-8 decoder.

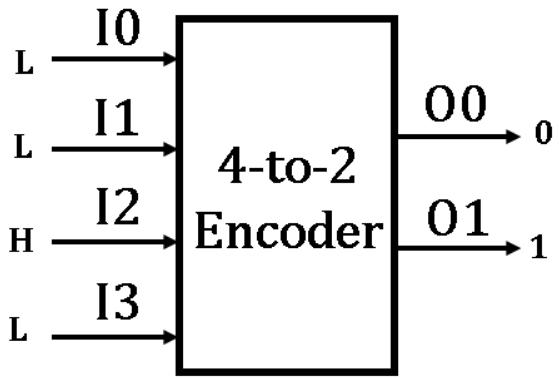
Encoders

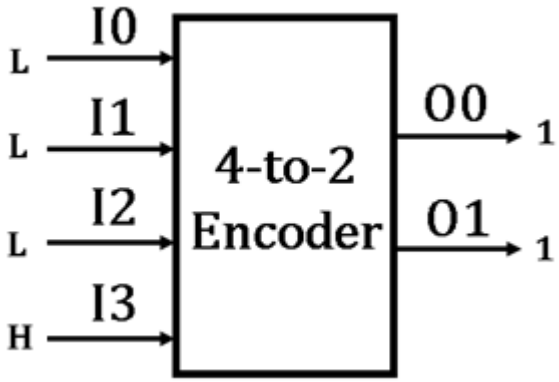
- An Encoder is a combinational logic circuit that performs a “reverse” decoder functions.
- As the name suggests its function is to encode an input signal.
- An encoder accepts an active level on one of its input and outputs the coded output for the input.
- Generally an encoder has 2^n inputs and **n**-outputs.
- There can be a variety of encoders, but we will only keep ourselves to binary encoders.











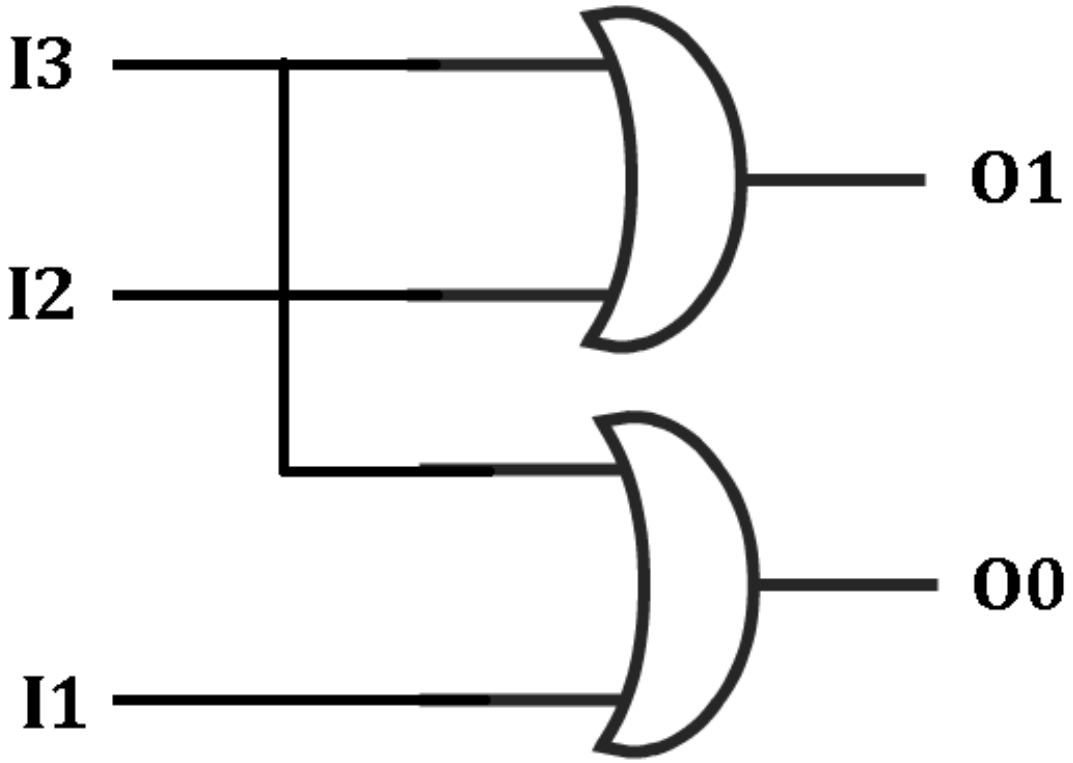
4-to-2 Encoder Design

- We now have a superficial idea of how the encoder works.
- So now we can form the truth table and finally design the encoder.

I3	I2	I1	I0	O1	O0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

- Since we have only these 4-combinations in the ideal case, the Boolean expressions are:

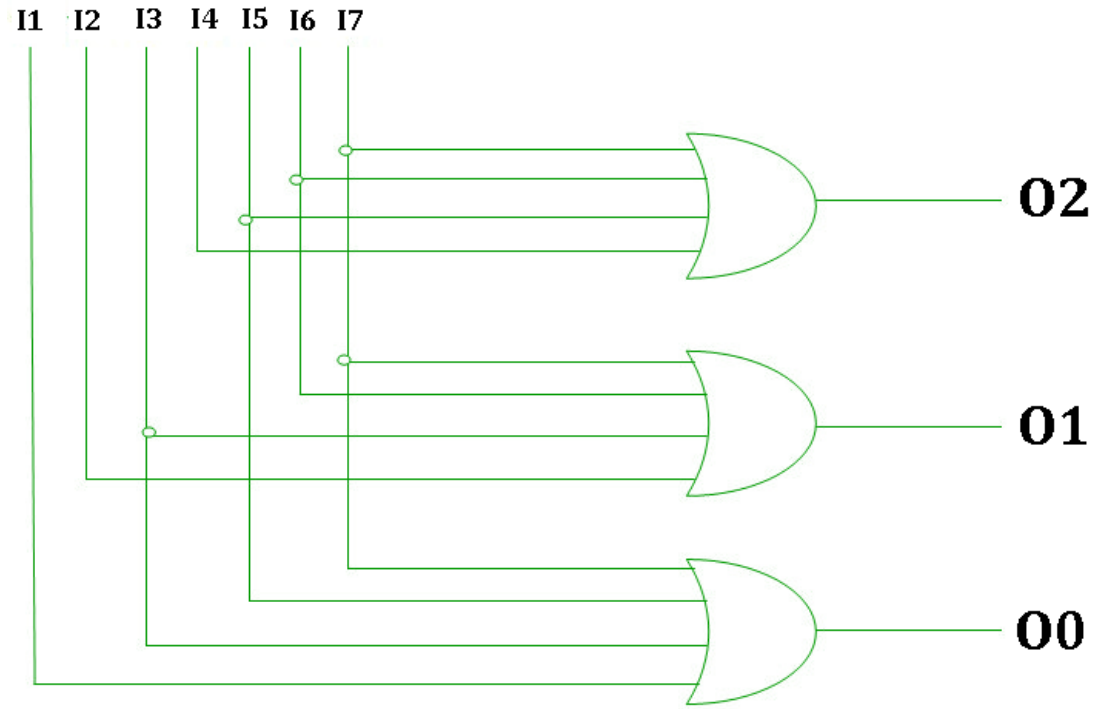
$$O0 = I1 + I3$$
$$O1 = I2 + I3$$



8-to-3 Encoder Design

- An 8-to-3 encoder can be designed in the same way.

I7	I6	I5	I4	I3	I2	I1	I0	O2	O1	O0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

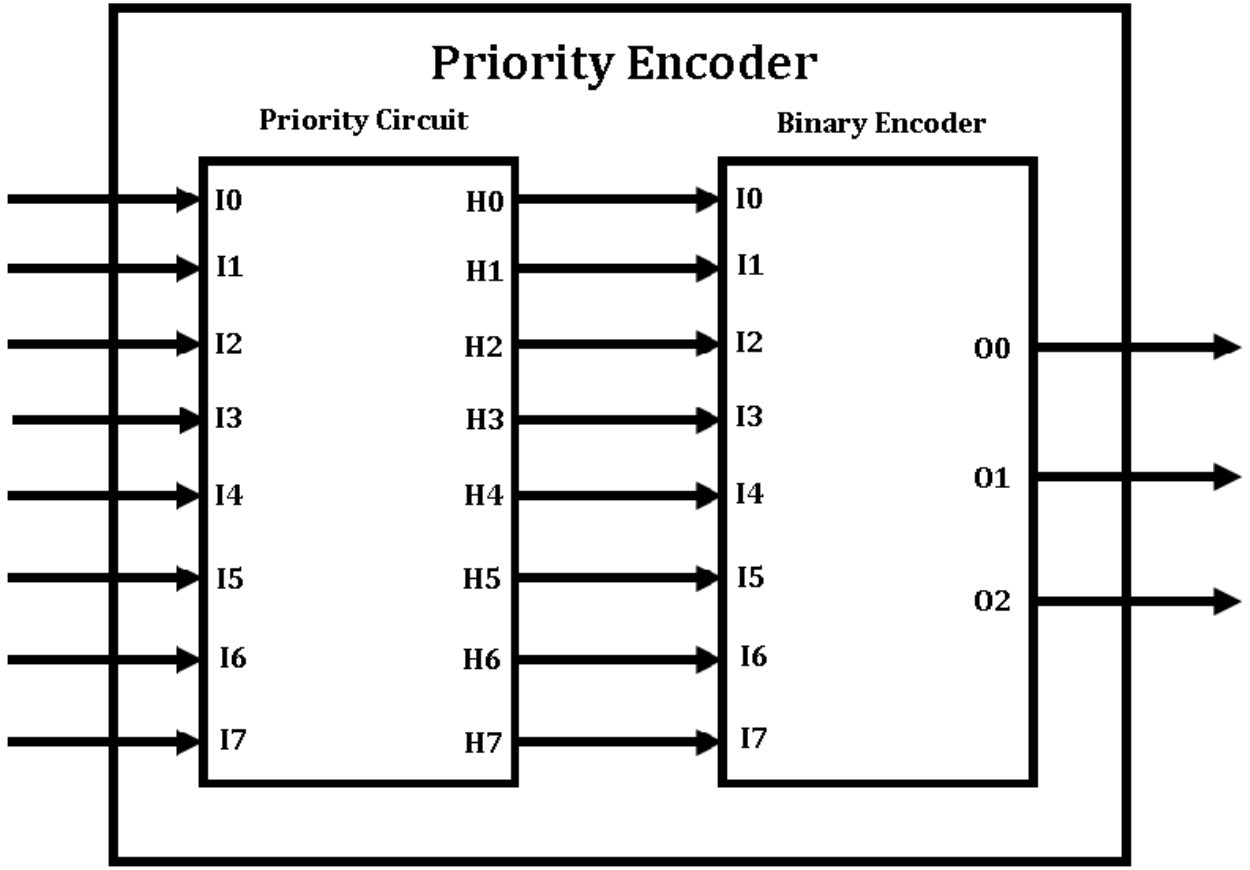


- Boolean expression:

$$O0 = I1 + I3 + I5 + I7$$
$$O1 = I2 + I3 + I6 + I7$$
$$O2 = I4 + I5 + I6 + I7$$

Priority Encoders

- We have designed the encoders for an ideal scenario, where only 1 input is active at time.
- However, in real world scenario, this will always not be the case.
- Hence, we need some modifications to be made.
- The solution to this problem is to define priority amongst the inputs.
- The priority circuit will detect the input with highest priority and output that signal only to the binary encoder.
- The encoder will then encode the received input.
- So lets us learn how to design a priority circuit.



Designing the Priority Circuit

- We now know the purpose of the priority circuit.
- So how to design the priority circuit.
- First, we need to know the priority of the inputs. So for our designing example let the priority be, I0>I1>I2>I3>I4>I5>I6>I7.
- Now, we need to have the truth table of the priority circuit.

I7	I6	I5	I4	I3	I2	I1	I0	H7	H6	H5	H4	H3	H2	H1	H0
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1
X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0
X	X	X	X	X	1	0	0	0	0	0	0	0	1	0	0
X	X	X	X	1	0	0	0	0	0	0	0	1	0	0	0
X	X	X	1	0	0	0	0	0	0	0	1	0	0	0	0
X	X	1	0	0	0	0	0	0	0	1	0	0	0	0	0
X	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Designing the Priority Circuit

I7	I6	I5	I4	I3	I2	I1	I0	H7	H6	H5	H4	H3	H2	H1	H0
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1
X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0
X	X	X	X	X	1	0	0	0	0	0	0	0	1	0	0
X	X	X	X	1	0	0	0	0	0	0	0	1	0	0	0
X	X	X	1	0	0	0	0	0	0	0	1	0	0	0	0
X	X	1	0	0	0	0	0	0	0	1	0	0	0	0	0
X	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

Boolean Expression for the priority circuit:

- $H0 = I0$
- $H1 = I1\overline{I0}$
- $H2 = I2\ \overline{I1}\ \overline{I0}$
- $H3 = I3\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H4 = I4\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H5 = I5\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H6 = I6\ \overline{I5}\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H7 = I7\overline{I6}\ \overline{I5}\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$

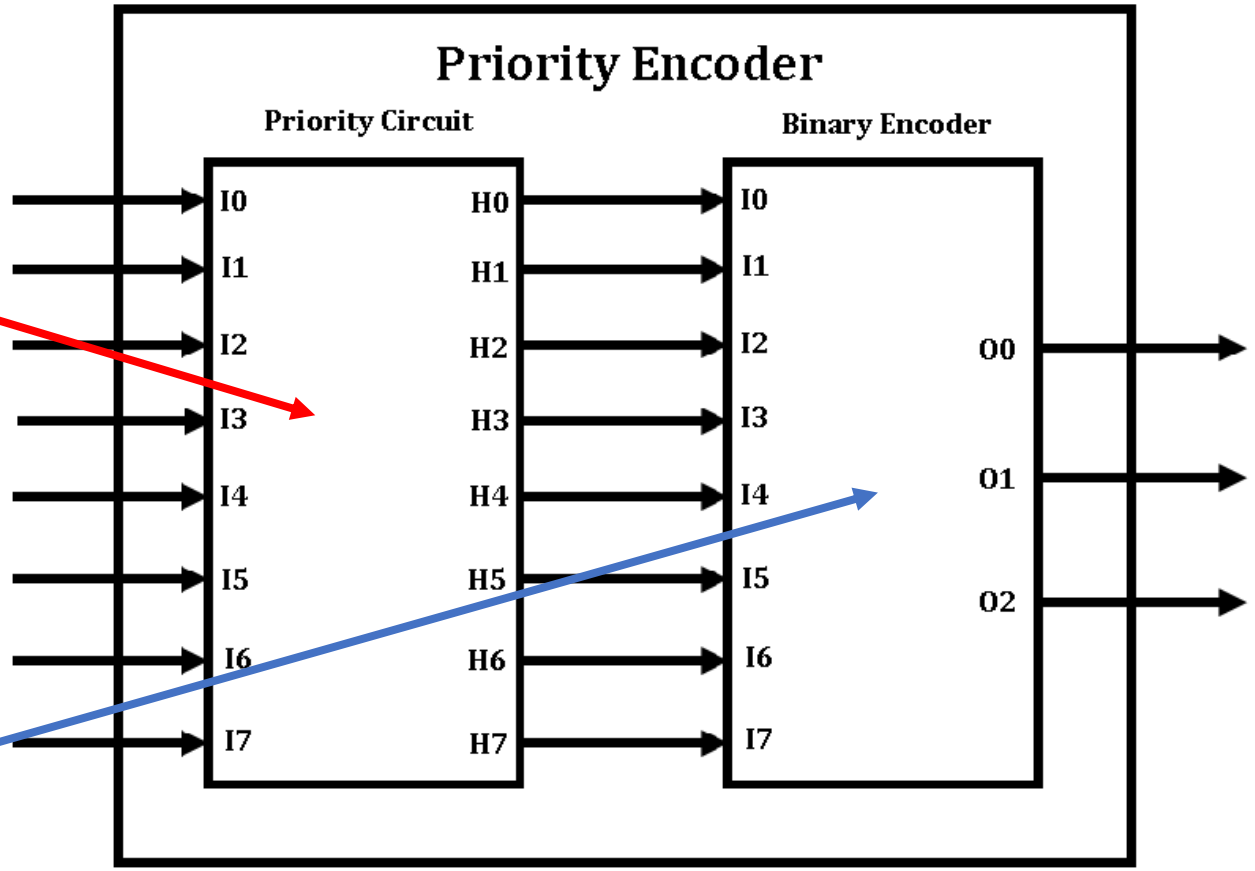
Now that we have assured the ideal scenario, we can just use the same logic for the encoder design.

- $O0 = I1 + I3 + I5 + I7$
- $O1 = I2 + I3 + I6 + I7$
- $O2 = I4 + I5 + I6 + I7$

Designing the Priority Circuit

Boolean Expression for the priority circuit:

- $H0 = I0$
- $H1 = I1\overline{I0}$
- $H2 = I2\ \overline{I1}\ \overline{I0}$
- $H3 = I3\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H4 = I4\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H5 = I5\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H6 = I6\ \overline{I5}\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$
- $H7 = I7\overline{I6}\ \overline{I5}\ \overline{I4}\ \overline{I3}\ \overline{I2}\ \overline{I1}\ \overline{I0}$



Boolean Expression for the priority circuit:

- $O0 = I1 + I3 + I5 + I7$
- $O1 = I2 + I3 + I6 + I7$
- $O2 = I4 + I5 + I6 + I7$

Priority Encoders with Irregular Sequence

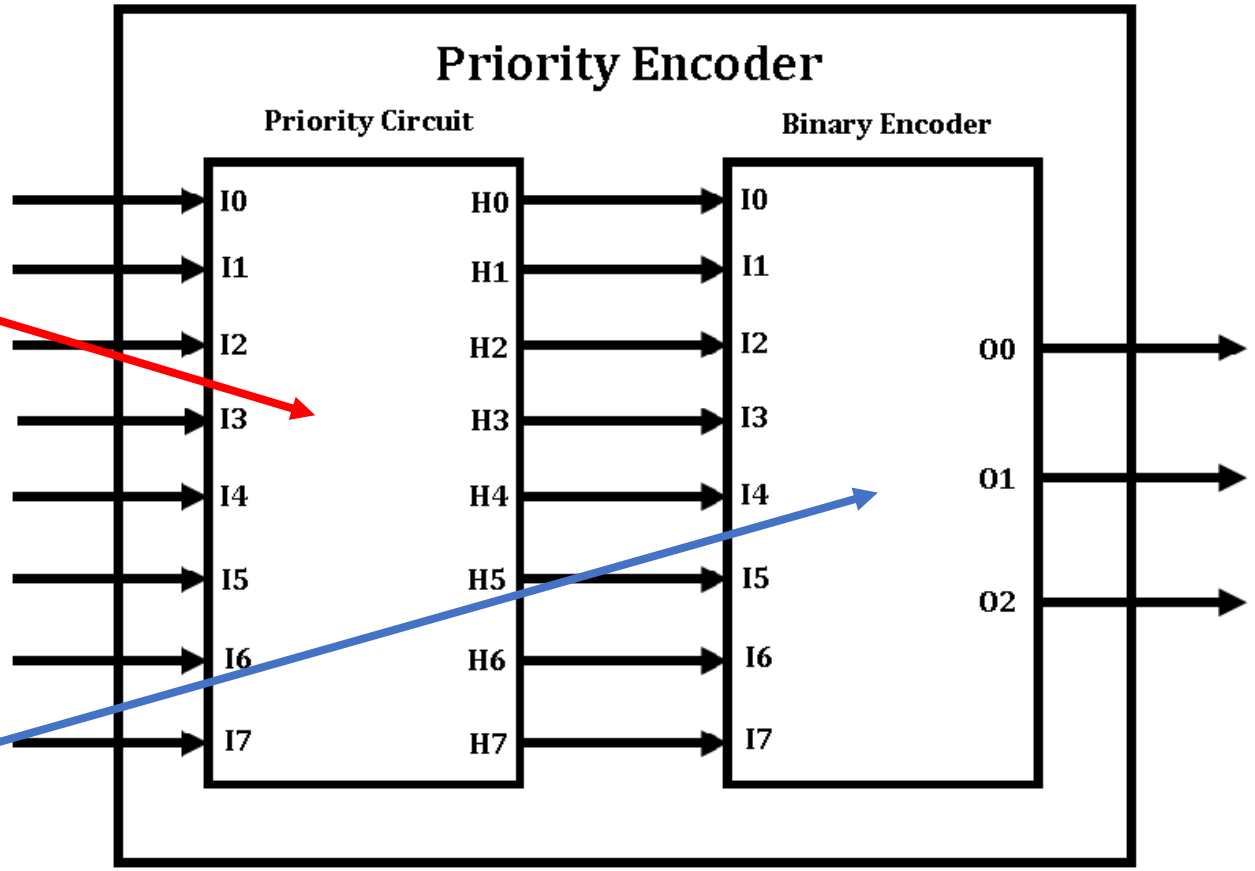
- Priority sequence of an encoder might not always be with a regular priority sequence such as ascending priority or descending priority.
- The priority for sequence might be irregular.
- An example can be: **I0>I1>I5>I6>I7>I3>I2>I4.**
- Designing procedure for this will be the same.
- So first we need the truth-table.

I7	I6	I5	I4	I3	I2	I1	I0	H7	H6	H5	H4	H3	H2	H1	H0
X	X	X	X	X	X	X	1	0	0	0	0	0	0	0	1
X	X	X	X	X	X	1	0	0	0	0	0	0	0	1	0
0	0	0	X	0	1	0	0	0	0	0	0	0	1	0	0
0	0	0	X	1	X	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
X	X	1	X	X	X	0	0	0	0	1	0	0	0	0	0
X	1	0	X	X	X	0	0	0	1	0	0	0	0	0	0
1	0	0	X	X	X	0	0	1	0	0	0	0	0	0	0

Priority Encoders with Irregular Sequence

Boolean Expression for the priority circuit:

- $H0 = I0$
- $H1 = I1\overline{I0}$
- $H2 = I2\ \overline{I5}\ \overline{I6}\ \overline{I7}\ \overline{I3}\ \overline{I1}\ \overline{I0}$
- $H3 = I3\ \overline{I7}\ \overline{I6}\ \overline{I5}\ \overline{I1}\ \overline{I0}$
- $H4 = I4\ \overline{I2}\ \overline{I3}\ \overline{I7}\ \overline{I6}\ \overline{I5}\ \overline{I1}\ \overline{I0}$
- $H5 = I5\ \overline{I1}\ \overline{I0}$
- $H6 = I6\ \overline{I5}\ \overline{I1}\ \overline{I0}$
- $H7 = I7\overline{I6}\ \overline{I5}\ \overline{I1}\ \overline{I0}$

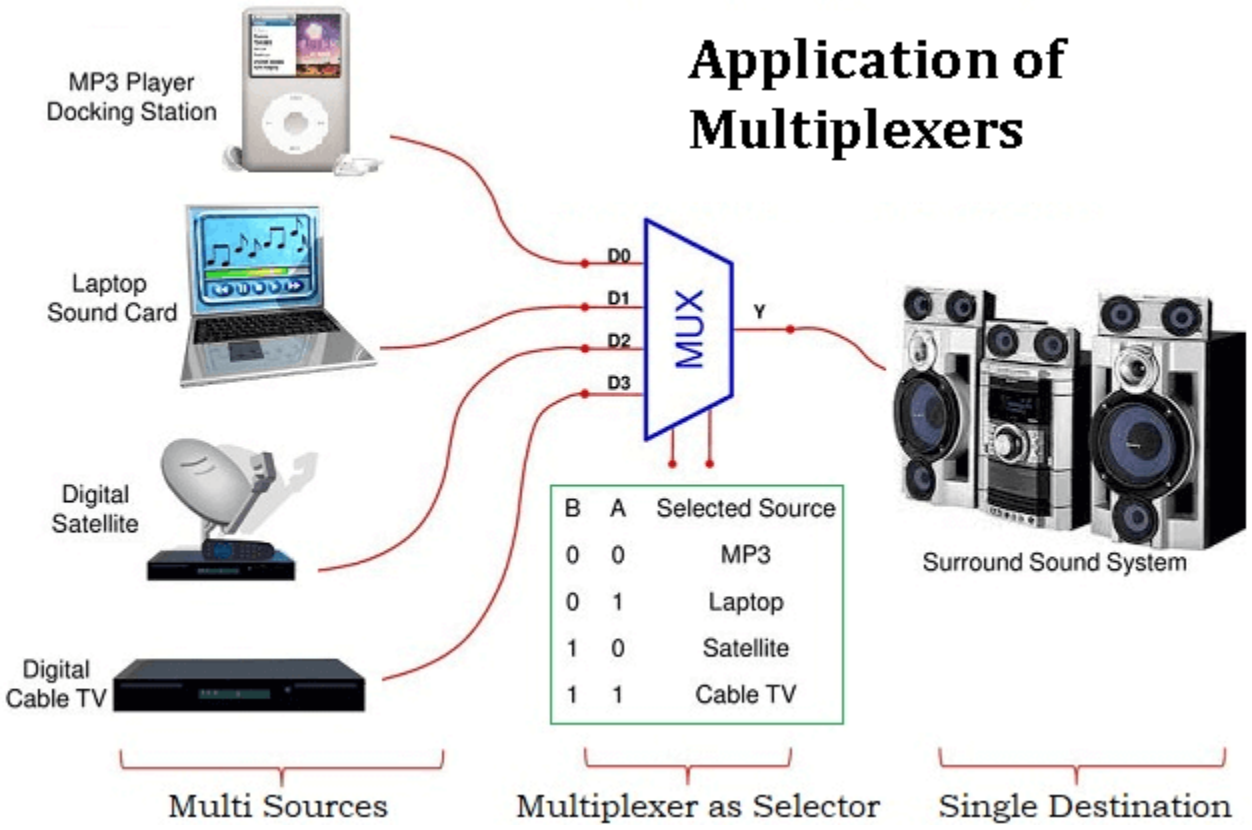


Boolean Expression for the priority circuit:

- $O0 = I1 + I3 + I5 + I7$
- $O1 = I2 + I3 + I6 + I7$
- $O2 = I4 + I5 + I6 + I7$

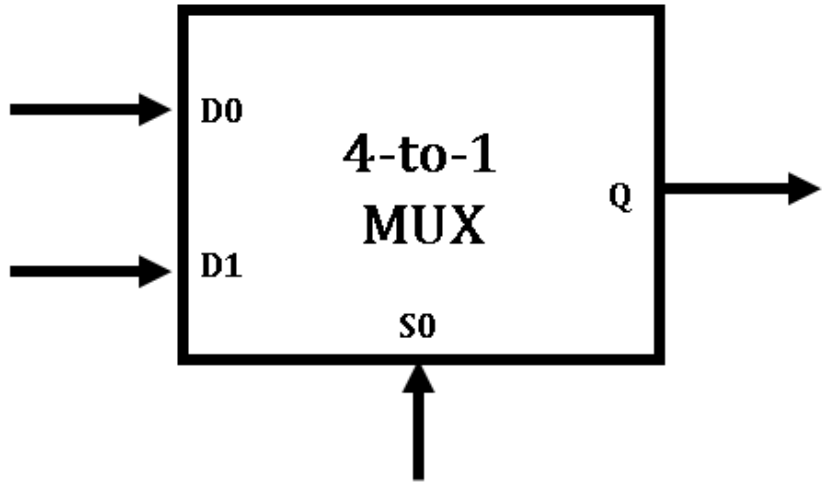
Multiplexers (MUX)

- In old days, multiple devices could be connected to a single device with the help of a switch.
- For example, multiple sources of input could be used as an input to a single home theater.
- The device that acts as switch is known as the multiplexer.
- What we see in the picture is a 4-to-1MUX.
- A 4-to-1 MUX has 2 selector pins.
- Therefore a 2^n-to-1 MUX will have n selector pins.
- As for our example we can see:
  - When selector pins are **00** the MP3 will be selected.
  - When the selector pins are **01** the laptop will be selected.
  - When the selector pin are 10 the digital satellite is selected and so on.



Designing a Multiplexers (MUX)

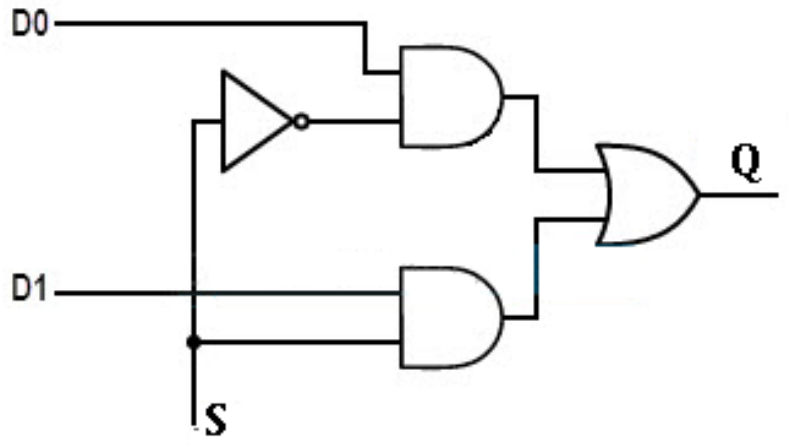
- So how is a MUX designed?
- MUX is a digital combinational circuit.
- Thus we need to first find its truth table.
- Let us for example take a 2-to-1 MUX.
- A 2-to-1 MUX will be 2 inputs to select from using a 1 selector pin.



S	D1	D0	Q
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Boolean expression for a MUX:

- $Q = \bar{S}\,\overline{D1}D0 + \bar{S}D1D0 + SD1\overline{D0} + SD1D0$
- $Q = \bar{S}D0 + SD1$



Designing a Multiplexers (MUX)

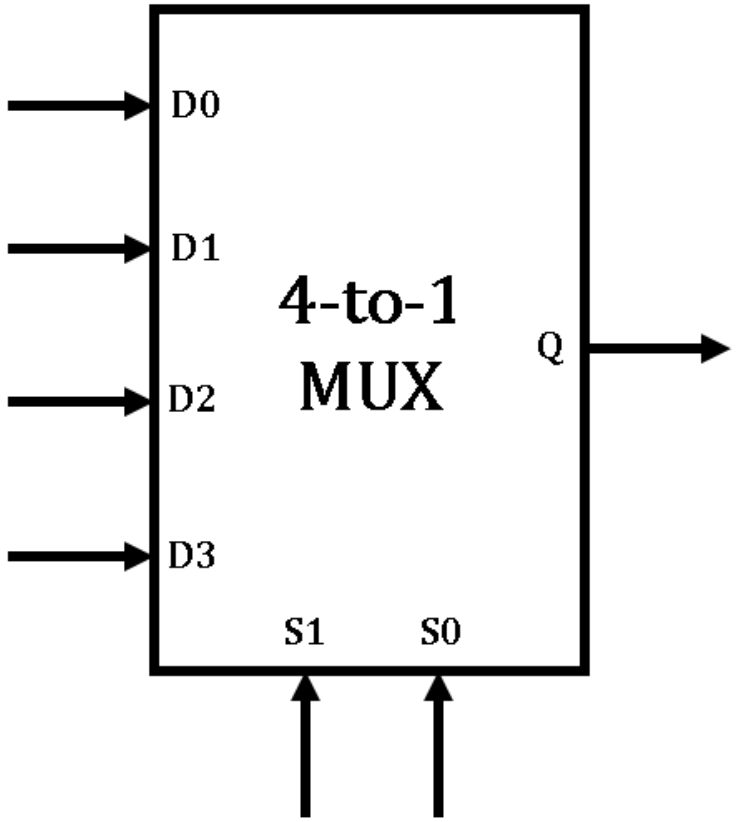
- Let us now design a 4-to-1 MUX.
- Since there are 6 inputs in total the truth table will become very big with 64 combinations.
- We can have the truth table, then find the min terms as our previous example, but this will surely become very tedious.
- So we can probably find a trick if we look at Boolean expression of our last design

$$\mathbf{Q = \bar{S}D0 + SD1}$$

- Seeing this equation we can find a shortcut to write the equation for 4-to-1 MUX as:

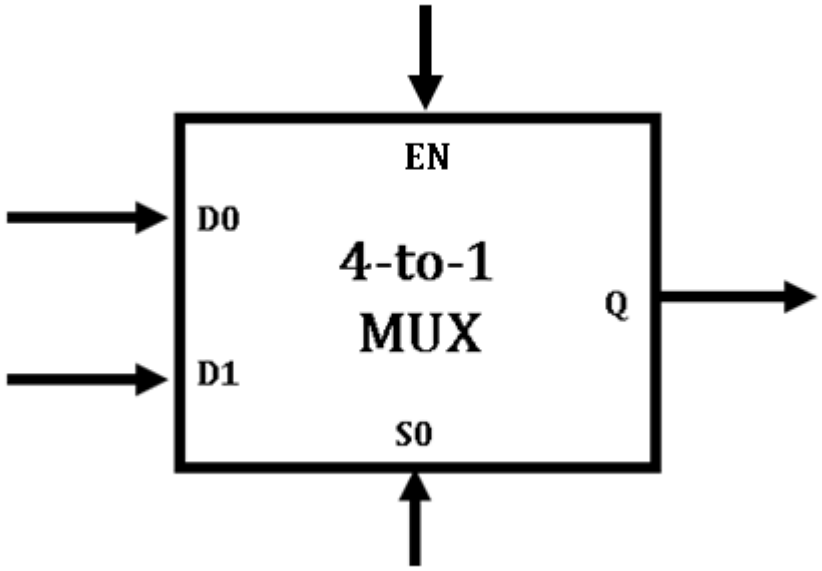
$$\mathbf{Q = \overline{S1S0}D0 + \overline{S1}S0D1 + S1\overline{S0}D2 + S1S2D3}$$

- After finding the Boolean expression we can draw the logical circuit of the 4-to-1 MUX



Enable Input

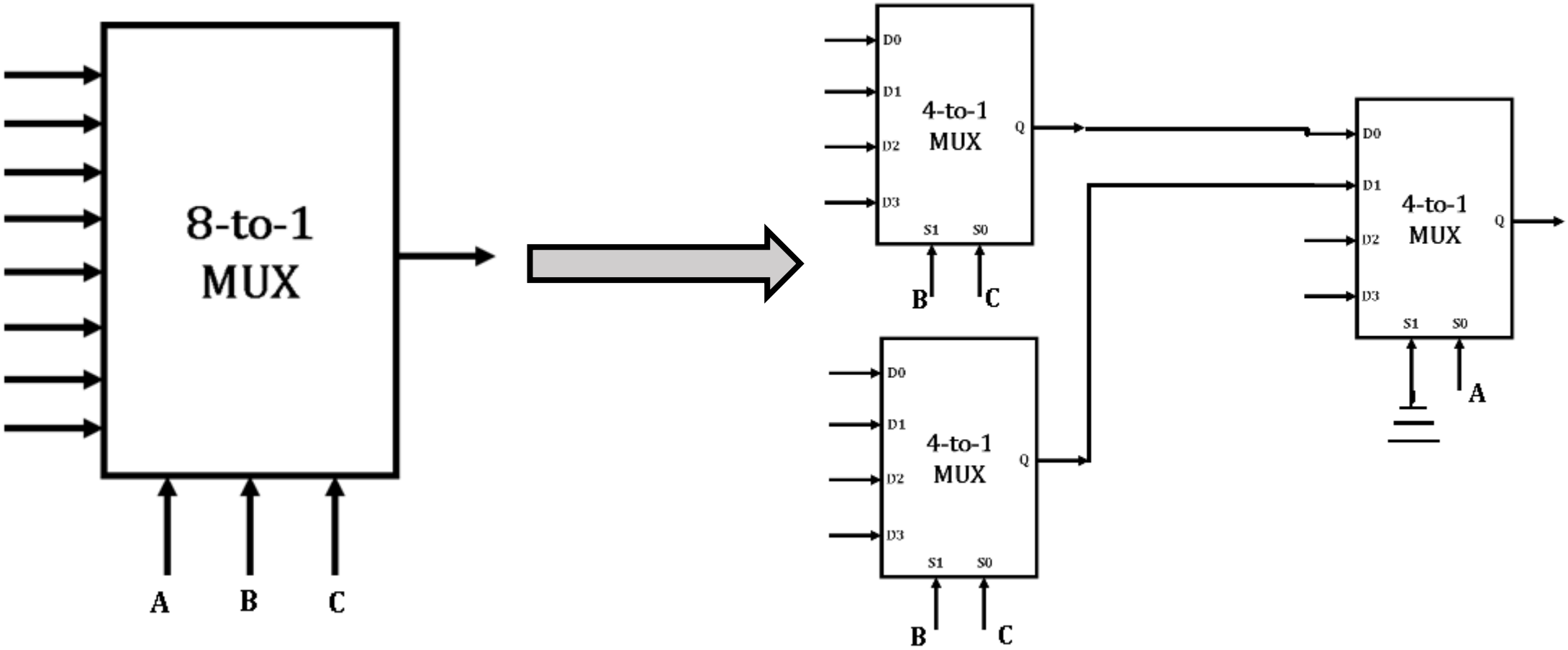
- Many devices have an extra input known as the “enable” input.
- This is used to activate or deactivate a device.
- Enables are especially useful in combining smaller MUXs to make larger MUXs.
- So how does the truth table vary with inclusion go enable.
- When enable is 0, the output is 0 irrespective of all other inputs.
- And when enable is 1, the MUX operates as specified earlier.



En	S	D1	D0	Q
0	X	X	X	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

Modularity

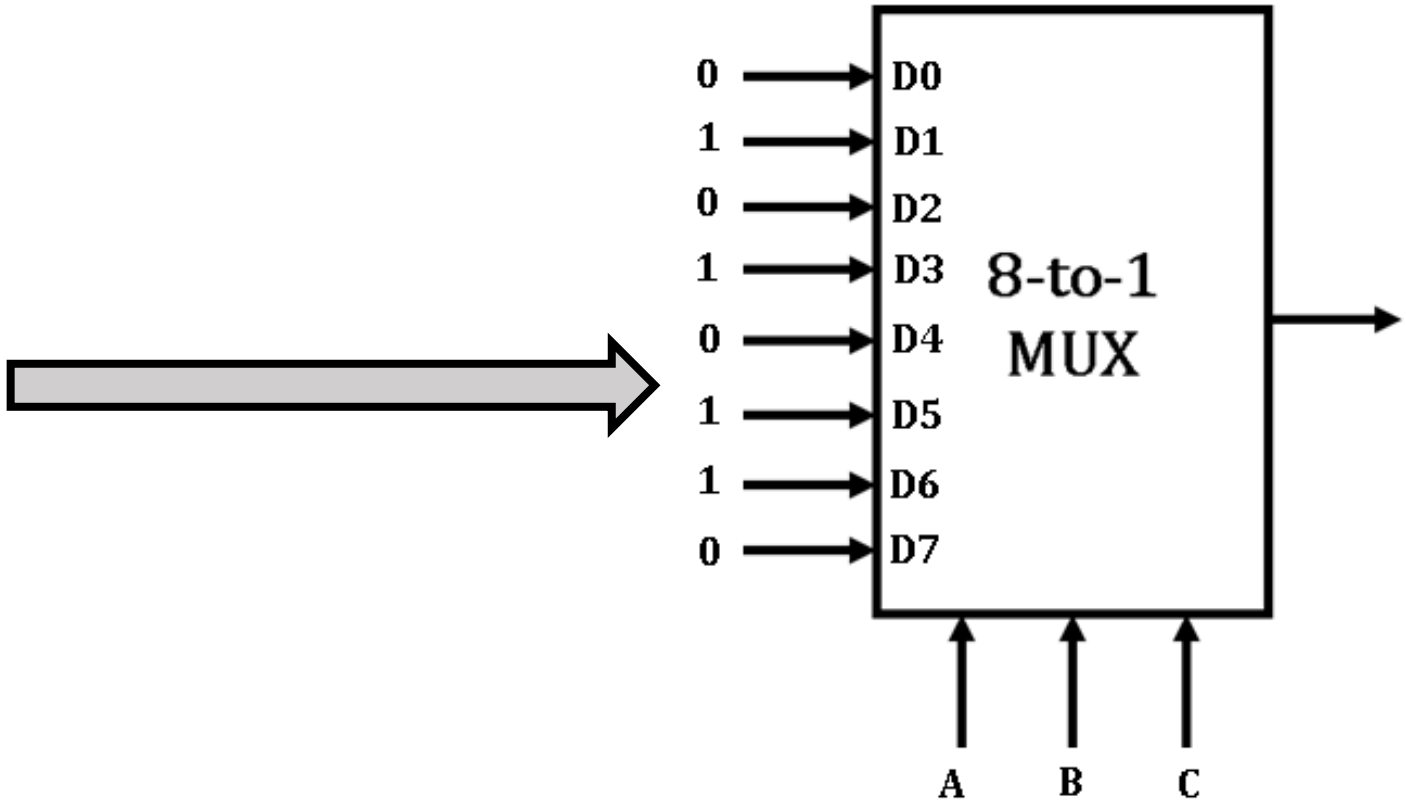
- It is not always possible to get a MUX with the exact number of required inputs.
- Say we need to connect 6 devices and we have only access to 4-to-1 MUXs.
- In such cases we can use smaller MUXs and combine them make larger MUXs.
- So as an example we will see how to use 4-to-1 MUX to make 8-to-1 MUX.



Implementing Functions with MUX

- MUX can also be used to implement Boolean functions.
- A n-variable function can be implemented using a 2^n-to-1 MUX (very easy).
- However, we can also implement a (n+1) variable function with 2^n-to-1 MUX (advanced).
- Let us see how we can implement the function $F(A, B, C) = \sum(1,3,5,6)$ using an 8-to-1 MUX.

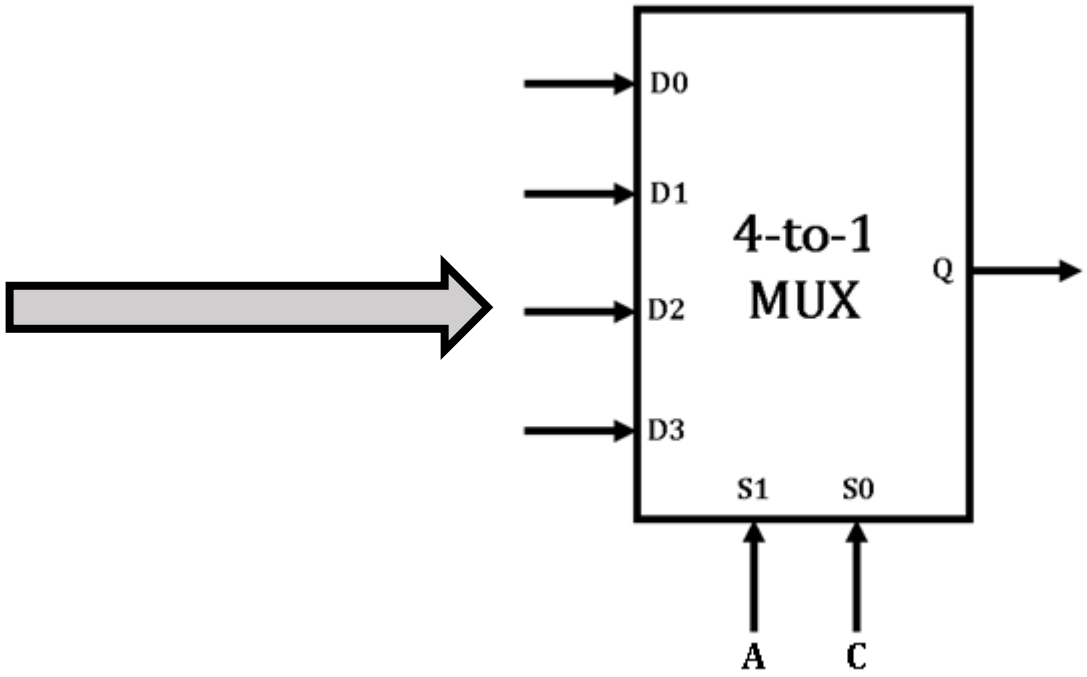
A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0



Implementing Functions with MUX

- Now let us implement the same function using a 4-to-1 MUX.
- First, we need to construct the truth table.
- Then we need to select the variables of the function we want to use as selector pin.
- Let us take A and C as selector pin.
- The remaining variable of B will be used in a different way.

A	B	C	F
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



Implementing Functions with MUX

- Now we will construct a special table for variable B from the truth-table.
- First, we need to find the min-terms for which B is 0. The min-terms are 0,1,4 and 5.
- Then, we need to find the min-terms for which B is 1. The min-terms are 2,3,6 and 7.
- Now list the input of the 4-to-1 MUX. The inputs are D0, D1, D2 and D3.
- Now create a table as shown

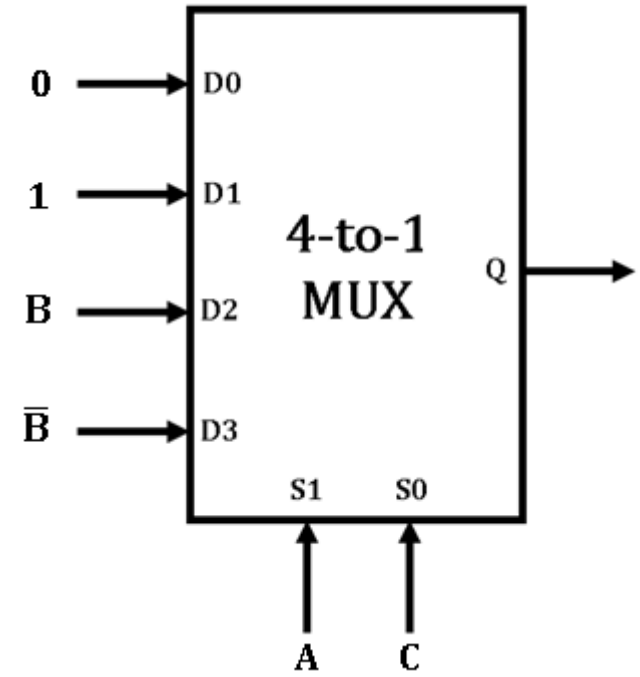
Minterms	A	B	C	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	0

	D0	D1	D2	D3
$\overline{B}$	0	1	4	5
B	2	3	6	7

Implementing Functions with MUX

- Now circle all the min-terms for which the function is 1.
- Now if two min-terms in a column is not circled, we apply a 0 for the input.
- Now if two min-terms in a column are circled, we apply 1 for the input.
- If there is one circle in a column, and the circle is the row of $\overline{B}$, we apply $\overline{B}$ for the input. And if the only circle is in the row of B, we apply B for the input.

	D0	D1	D2	D3
$\overline{B}$	0	(1)	4	(5)
B	2	(3)	(6)	7
	0	1	B	$\overline{B}$



Implementing Functions with MUX

- Implement the following function with an 8-to-1 MUX with B,C and D as selector pins.

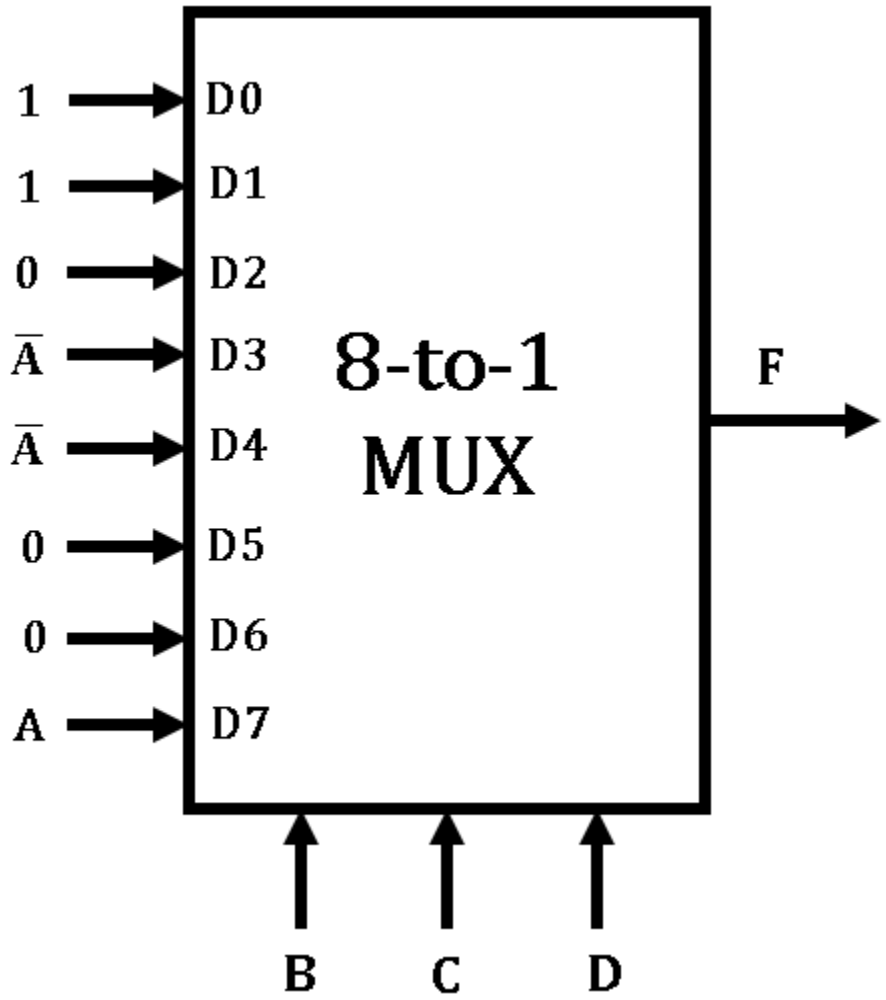
$$F(A, B, C, D) = \sum(0, 1, 3, 4, 8, 9, 15)$$

Minterms	A	B	C	D	F
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	0
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	0
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	1

	D0	D1	D2	D3	D4	D5	D6	D7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

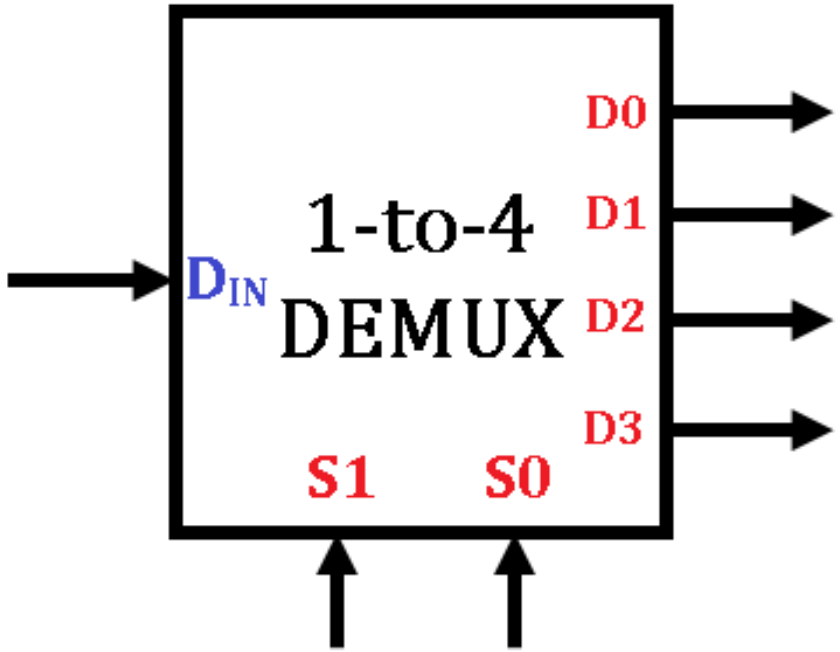
Implementing Functions with MUX

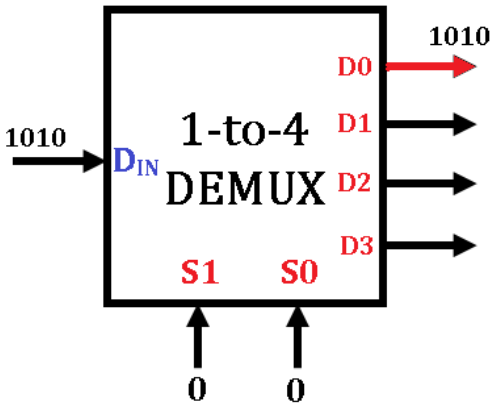
	D0	D1	D2	D3	D4	D5	D6	D7
$\bar{A}$	0	1	2	3	4	5	6	7
A	8	9	10	11	12	13	14	15
	1	1	0	$\bar{A}$	$\bar{A}$	0	0	A

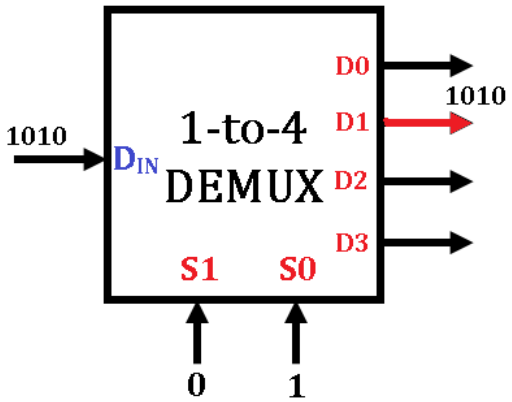


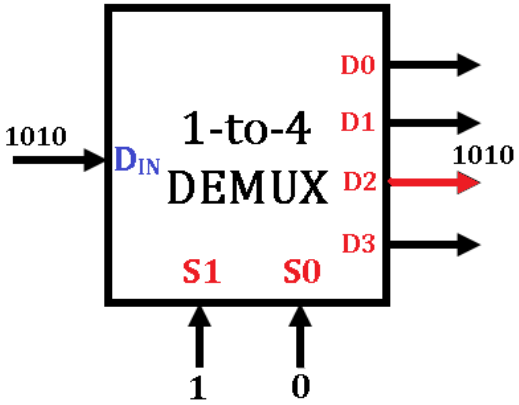
Demultiplexer (DEMUX)

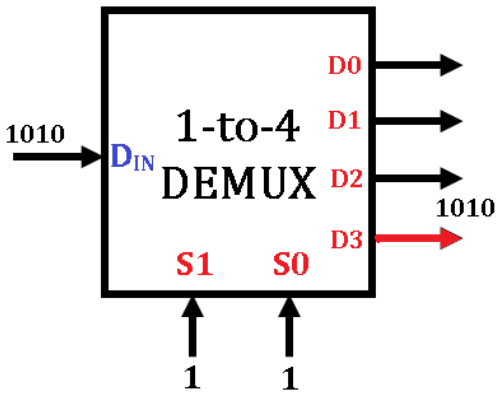
- A demultiplexer is a combinational circuit.
- It just reverses the operation of a multiplexer.
- It basically takes digital information from one line and distribute it to a given number of output lines.
- For this reason, the demultiplexer is also know as a data distributor.
- A 1-to-2^n DEMUX has n selector pins.
- The selector pin are used to select the output lines where the data is to be distributed.
- The figure shows a 1-to-4 demultiplexer.









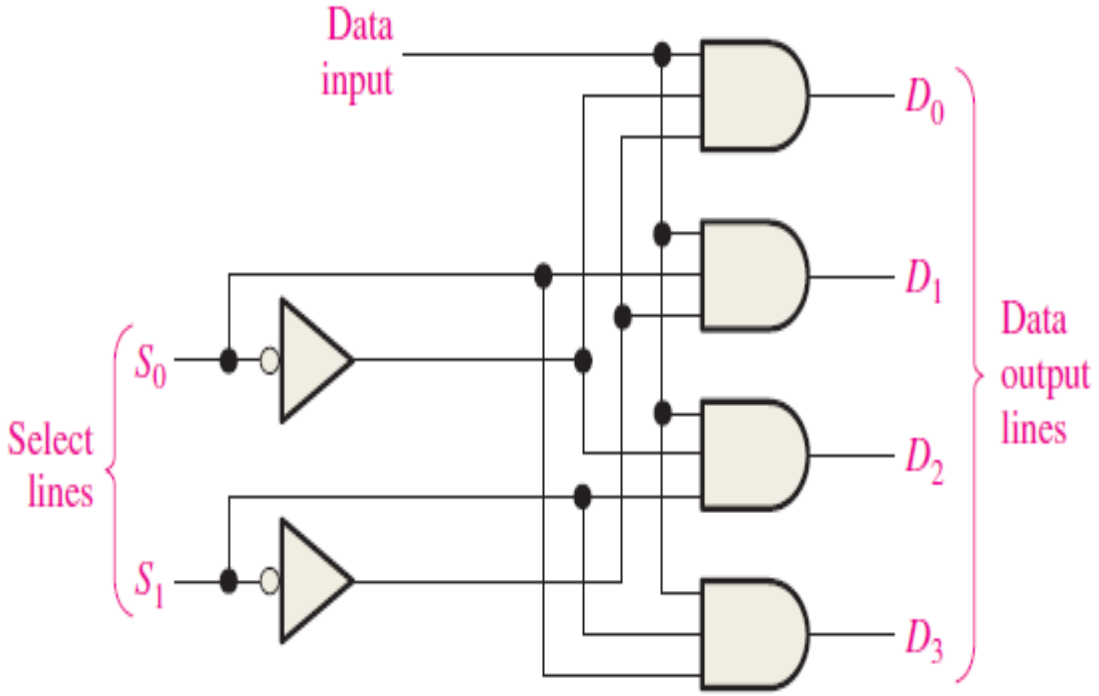


Demultiplexer Design

- How to design a 1-to-4 demultiplexer.
- First, according to our understanding of the operation we make the truth-table.
- The from the truth table we will form the Boolean expressions.
- And from the Boolean expression we will draw the logic circuit.

D_{IN}	S1	S0	D0	D1	D2	D3
D_{IN}	0	0	D_{IN}	0	0	0
D_{IN}	0	1	0	D_{IN}	0	0
D_{IN}	1	0	0	0	D_{IN}	0
D_{IN}	1	1	0	0	0	D_{IN}

- $D0 = \overline{S1}.\overline{S0}.D_{IN}$
- $D1 = \overline{S1}.S0.D_{IN}$
- $D2 = S1.\overline{S0}.D_{IN}$
- $D3 = S1.S2.D_{IN}$



References

1. Thomas L. Floyd, “Digital Fundamentals” 11th edition, Prentice Hall – Pearson Education.

Thank You