Assignment 2 Part 2

Multiplexer and Demultiplexer

A multiplexer is a circuit that accept many input but give only one output. A demultiplexer function exactly in the reverse of a multiplexer, that is a demultiplexer accepts only one input and gives many outputs. Generally multiplexer and demultiplexer are used together, because of the communication systems are bi directional.

Mutliplexer:

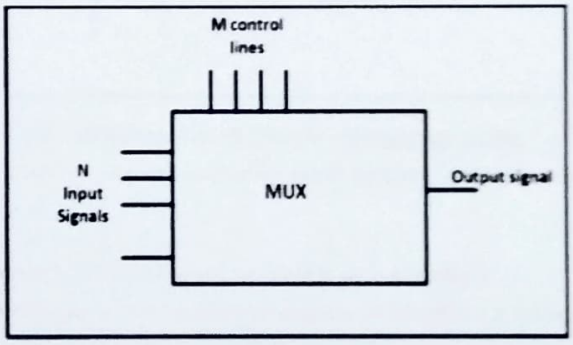
Multiplexer means many into one. A multiplexer is a circuit used to select and route any one of the several input signals to a signal output. An simple example of an non electronic circuit of a multiplexer is a single pole multiposition switch.

Multiposition switches are widely used in many electronics circuits. However circuits that operate at high speed require the multiplexer to be automatically selected. A mechanical switch cannot perform this task satisfactorily. Therefore, multiplexer used to perform high speed switching are constructed of electronic components.

Multiplexer handle two type of data that is analog and digital. For analog application, multiplexer are built of relays and transistor switches. For digital application, they are built from standard logic gates.

The multiplexer used for digital applications, also called digital multiplexer, is a circuit with many input but only one output. By applying control signals, we can steer any input to the output. Few types of multiplexer are 2-to-1, 4-to-1, 8-to-1, 16-to-1 multiplexer.

Following figure shows the general idea of a multiplexer with n input signal, m control signals and one output signal.

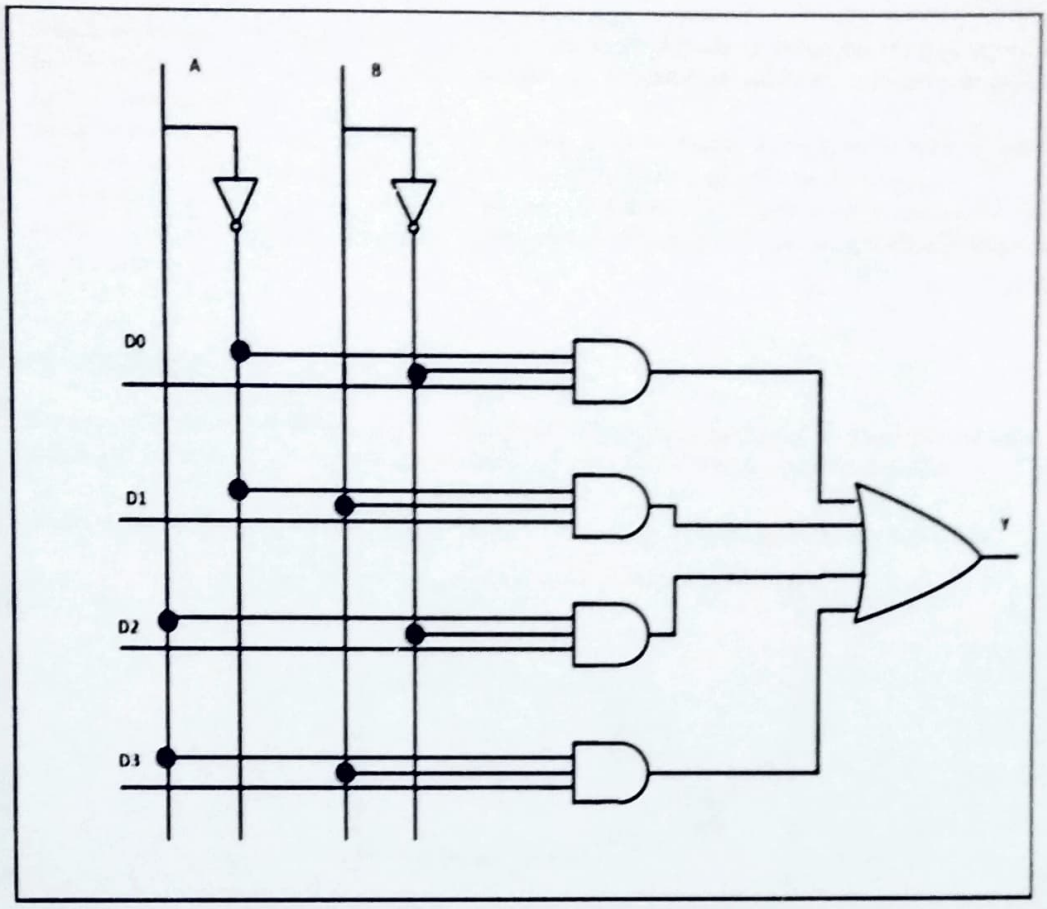


Multiplexer Pin Diagram

Understanding 4-to-1 Multiplexer:

The 4-to-1 multiplexer has 4 input bit, 2 control bits, and 1 output bit. The four input bits are D0,D1,D2 and D3. only one of this is transmitted to the output y. The output depends on the value of AB which is the control input. The control input determines which of the input data bit is transmitted to the output.

For instance, as shown in fig. when AB = 00, the upper AND gate is enabled while all other AND gates are disabled. Therefore, data bit D0 is transmitted to the output, giving Y = Do.



4 to 1 Multiplexer Circuit Diagram – ElectronicsHub.Org

If the control input is changed to AB =11, all gates are disabled except the bottom AND gate. In this case, D3 is transmitted to the output and Y = D3.

- An example of 4-to-1 multiplexer is IC 74153 in which the output is same as the input.
- Another example of 4-to-1 multiplexer is 45352 in which the output is the compliment of the input.
- Example of 16-to-1 line multiplexer is IC74150.

Applications of Multiplexer:

Multiplexer are used in various fields where multiple data need to be transmitted using a single line. Following are some of the applications of multiplexers -

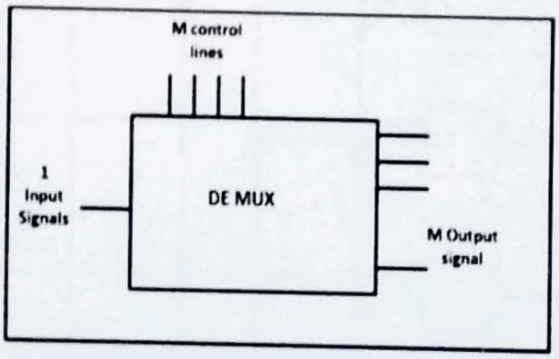
1. **Communication system** – Communication system is a set of system that enable communication like transmission system, relay and tributary station, and communication network. The efficiency of communication system can be increased considerably using multiplexer. Multiplexer allow the process of transmitting different type of data such as audio, video at the same time using a single transmission line.

2. **Telephone network –** In telephone network, multiple audio signals are integrated on a single line for transmission with the help of multiplexers. In this way, multiple audio signals can be isolated and eventually, the desire audio signals reach the intended recipients.
3. **Computer memory** - Multiplexers are used to implement huge amount of memory into the computer, at the same time reduces the number of copper lines required to connect the memory to other parts of the computer circuit.
4. **Transmission from the computer system of a satellite** – Multiplexer can be used for the transmission of data signals from the computer system of a satellite or spacecraft to the ground system using the GPS (Global Positioning System) satellites.

Demultiplexer:

Demultiplexer means one to many. A demultiplexer is a circuit with one input and many output. By applying control signal, we can steer any input to the output. Few types of demultiplexer are 1-to 2, 1-to-4, 1-to-8 and 1-to 16 demultiplexer.

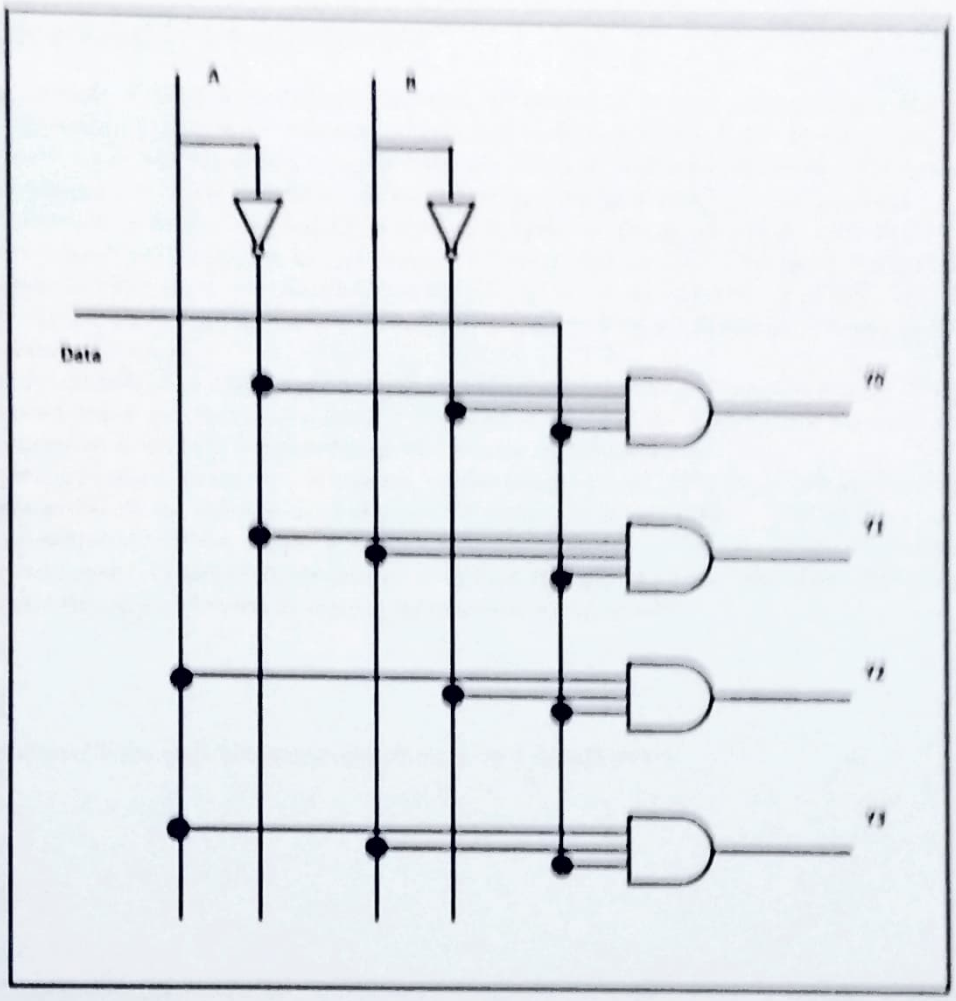
Following figure illustrate the general idea of a demultiplexer with 1 input signal, m control signals, and n output signals.



Demultiplexer Pin Diagram

Understanding 1- to-4 Demultiplexer:

The 1-to-4 demultiplexer has 1 input bit, 2 control bit, and 4 output bits. An example of 1-to-4 demultiplexer is IC 74155. The 1-to-4 demultiplexer is shown in figure below-



1 to 4 Dempultiplexer Circuit Diagram – ElectronicsHub.Org

The input bit is labelled as Data D. This data bit is transmitted to the data bit of the output lines. This depends on the value of AB, the control input.

When AB = 01, the upper second AND gate is enabled while other AND gates are disabled. Therefore, only data bit D is transmitted to the output, giving Y1 = Data.

If D is low, Y1 is low. IF D is high,Y1 is high. The value of Y1 depends upon the value of D. All other outputs are in low state.

If the control input is changed to AB = 10, all the gates are disabled except the third AND gate from the top. Then, D is transmitted only to the Y2 output, and Y2 = Data.

Example of 1-to-16 demultiplexer is IC 74154 it has 1 input bit, 4 control bits and 16 output bit.

Applications of Demultiplexer:

1. Demultiplexer is used to connect a single source to multiple destinations. The main application area of demultiplexer is communication system where multiplexer are used. Most of the communication system are bidirectional i.e. they function in both ways (transmitting and receiving signals). Hence, for most of the applications, the multiplexer and demultiplexer work in sync. Demultiplexer are also used for reconstruction of parallel data and ALU circuits.
2. **Communication System** - Communication system use multiplexer to carry multiple data like audio, video and other form of data using a single line for transmission. This process make the transmission easier. The demultiplexer receive the output signals of the multiplexer and converts them back to the original form of the data at the receiving end. The multiplexer and demultiplexer work together to carry out the process of transmission and reception of data in communication system.
3. **ALU (Arithmetic Logic Unit)** – In an ALU circuit, the output of ALU can be stored in multiple registers or storage units with the help of demultiplexer. The output of ALU is fed as the data input to the demultiplexer. Each output of demultiplexer is connected to multiple register which can be stored in the registers.
4. **Serial to parallel converter** - A serial to parallel converter is used for reconstructing parallel data from incoming serial data stream. In this technique, serial data from the incoming serial data stream is given as data input to the demultiplexer at the regular intervals. A counter is attach to the control input of the demultiplexer. This counter directs the data signal to the output of the demultiplexer where these data signals are stored. When all data signals have been stored, the output of the demultiplexer can be retrieved and read out in parallel.

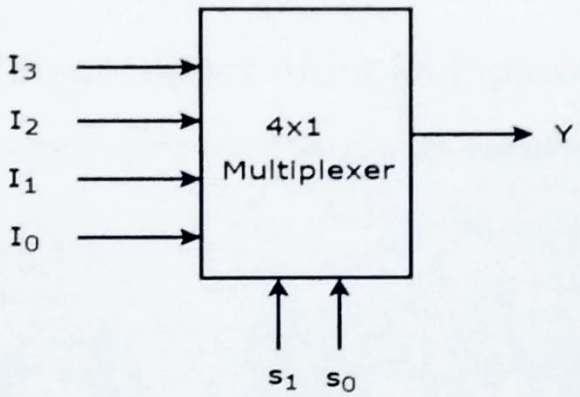
Source: http://www.electronicshub.org/multiplexer-and-demultiplexer/

Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output 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 will select only one data input. Multiplexer is also called as **Mux**.

4x1 Multiplexer

4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y. The **block diagram** of 4x1 Multiplexer is shown in the following figure.



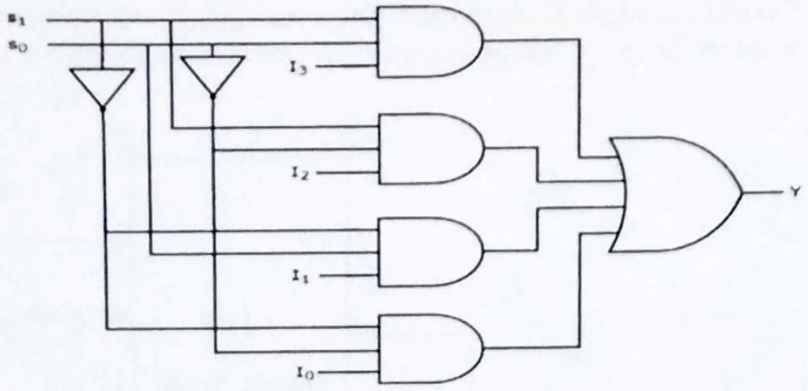
One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

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

From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y=S_1'S_0'I_0+S_1'S_0I_1+S_1S_0'I_2+S_1S_0I_3 \quad Y=S1'S0'I0+S1'S0I1+S1S0'I2+S1S0I3$$

We can implement this Boolean function using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 8x1 Multiplexer and 16x1 multiplexer by following the same procedure.

Implementation of Higher-order Multiplexers.

Now, let us implement the following two higher-order Multiplexers using lower-order Multiplexers.

- 8x1 Multiplexer
- 16x1 Multiplexer

8x1 Multiplexer

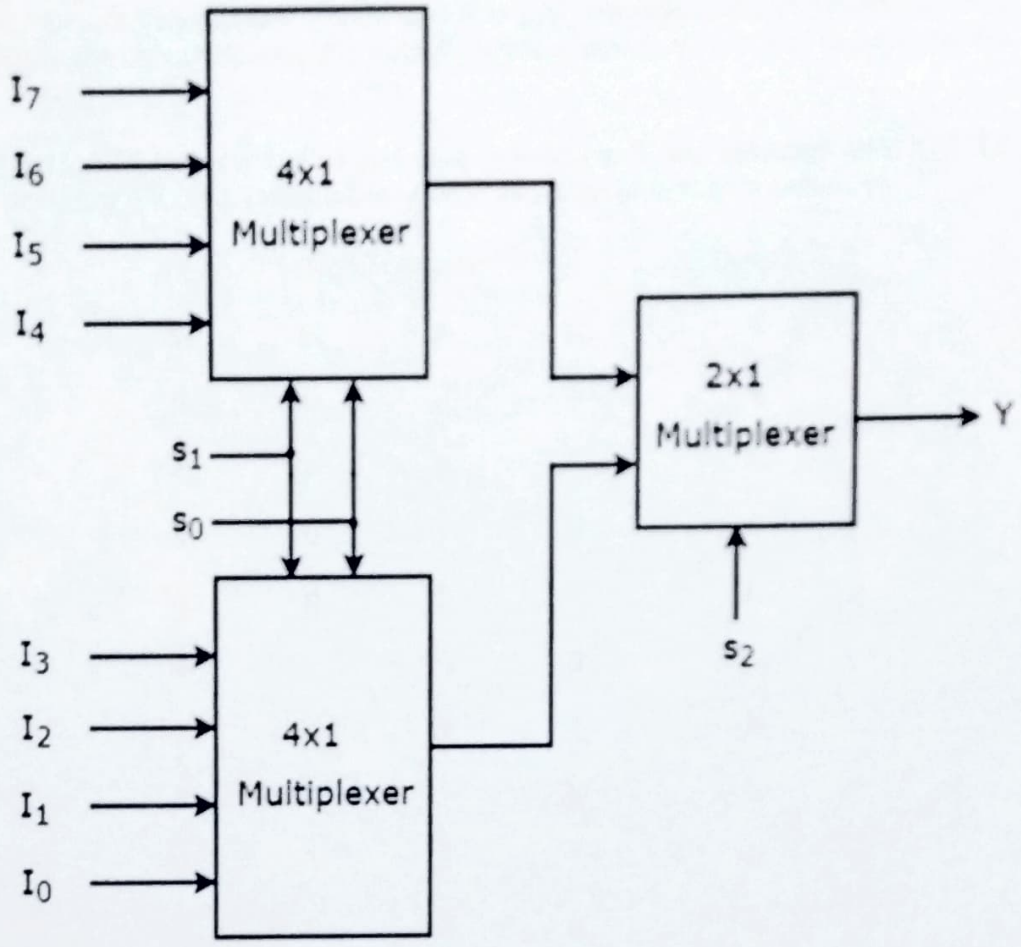
In this section, let us implement 8x1 Multiplexer using 4x1 Multiplexers and 2x1 Multiplexer. We know that 4x1 Multiplexer has 4 data inputs, 2 selection lines and one output. Whereas, 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output.

So, we require two **4x1 Multiplexers** in first stage in order to get the 8 data inputs. Since, each 4x1 Multiplexer produces one output, we require a **2x1 Multiplexer** in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 8x1 Multiplexer has eight data inputs I_7 to I_0, three selection lines s_2, s_1 & s0 and one output Y. The **Truth table** of 8x1 Multiplexer is shown below.

Selection Inputs			Output
S_2	S_1	S_0	Y
0	0	0	I_0
0	0	1	I_1
0	1	0	I_2
0	1	1	I_3
1	0	0	I_4
1	0	1	I_5
1	1	0	I_6
1	1	1	I_7

We can implement 8x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 8x1 Multiplexer is shown in the following figure.



The same **selection lines, s_1 & s_0** are applied to both 4x1 Multiplexers. The data inputs of upper 4x1 Multiplexer are I_7 to I_4 and the data inputs of lower 4x1 Multiplexer are I_3 to I_0. Therefore, each 4x1 Multiplexer produces an output based on the values of selection lines, s_1 & s_0.

The outputs of first stage 4x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_2** is applied to 2x1 Multiplexer.

- If s_2 is zero, then the output of 2x1 Multiplexer will be one of the 4 inputs I_3 to I_0 based on the values of selection lines s_1 & s_0.
- If s_2 is one, then the output of 2x1 Multiplexer will be one of the 4 inputs I_7 to I_4 based on the values of selection lines s_1 & s_0.

Therefore, the overall combination of two 4x1 Multiplexers and one 2x1 Multiplexer performs as one 8x1 Multiplexer.

16x1 Multiplexer

In this section, let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and

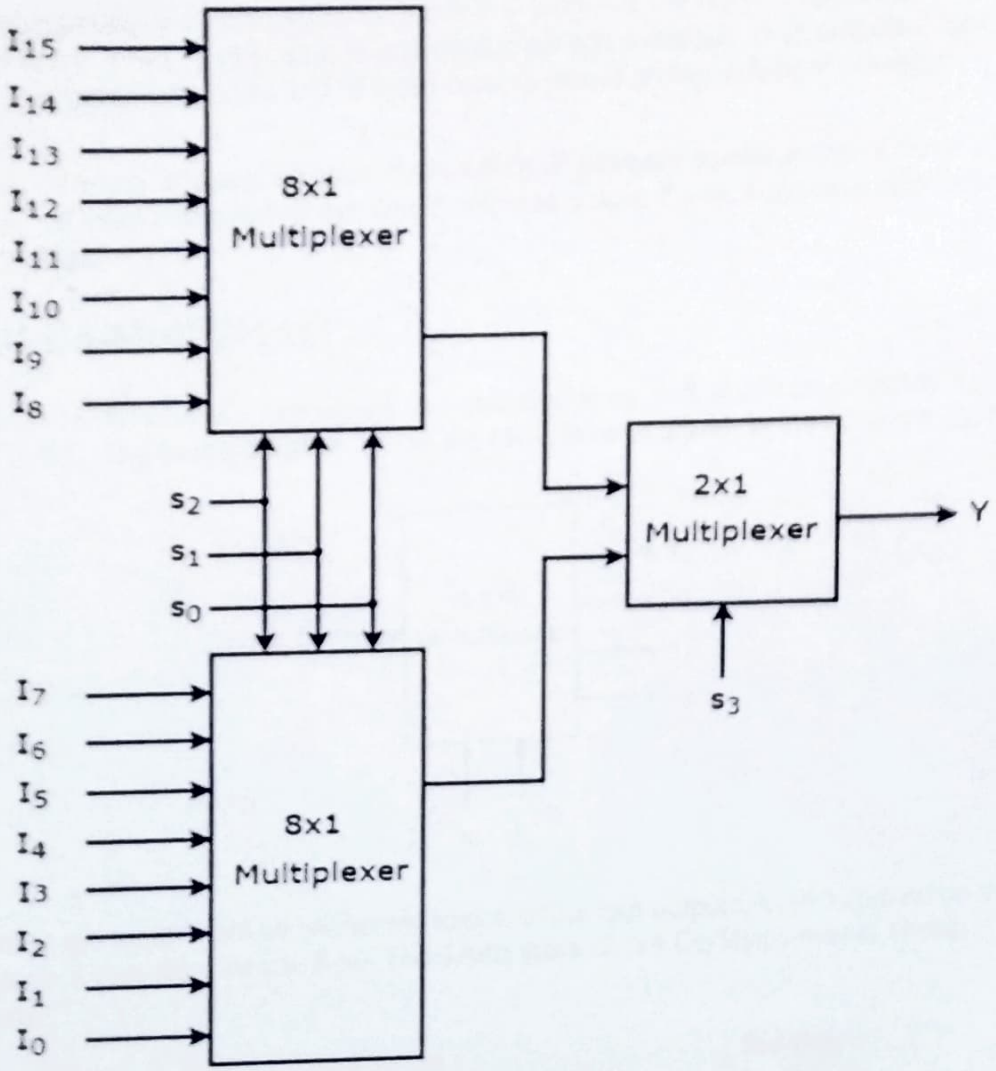
one output. Whereas, 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output.

So, we require two **8x1 Multiplexers** in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.

Let the 16x1 Multiplexer has sixteen data inputs I_{15} to I_0, four selection lines s_3 to s_0 and one output Y. The **Truth table** of 16x1 Multiplexer is shown below.

Selection Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4
0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7
1	0	0	0	I_8
1	0	0	1	I_9
1	0	1	0	I_{10}
1	0	1	1	I_{11}
1	1	0	0	I_{12}
1	1	0	1	I_{13}
1	1	1	0	I_{14}
1	1	1	1	I_{15}

We can implement 16x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 16x1 Multiplexer is shown in the following figure.



The **same selection lines, s_2, s_1 & s_0** are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are I_{15} to I_8 and the data inputs of lower 8x1 Multiplexer are I_7 to I_0. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, s_2, s_1 & s_0.

The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_3** is applied to 2x1 Multiplexer.

- If s_3 is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs Is_7 to I_0 based on the values of selection lines s_2, s_1 & s_0.
- If s_3 is one, then the output of 2x1 Multiplexer will be one of the 8 inputs I_{15} to I_8 based on the values of selection lines s_2, s_1 & s_0.

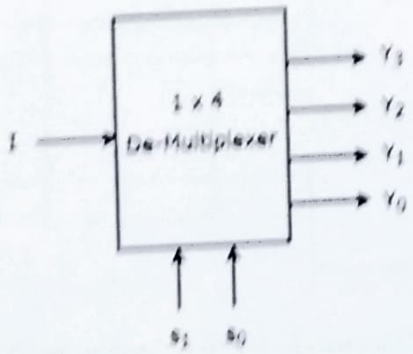
Therefore, the overall combination of two 8x1 Multiplexers and one 2x1 Multiplexer performs as one 16x1 Multiplexer.

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 **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



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. The **Truth table** of 1x4 De-Multiplexer is shown below.

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

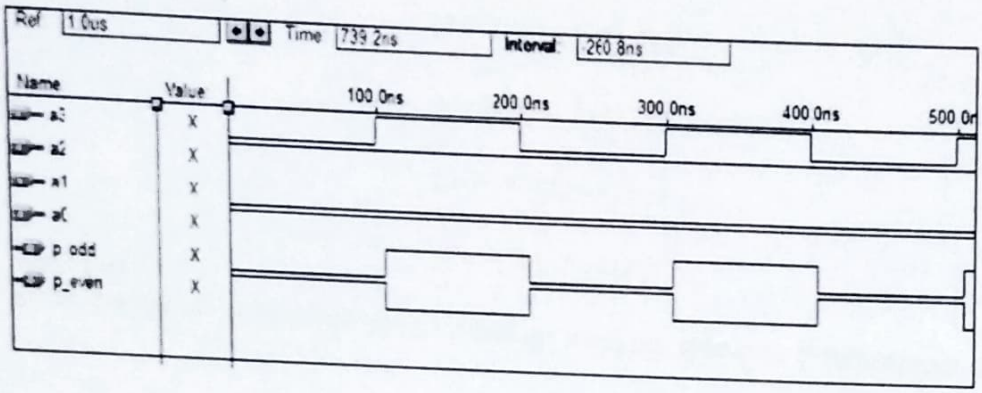
From the above Truth table, we can directly write the **Boolean functions** for each output as

$$Y_3=s_1s_0/Y3=s1s0I$$

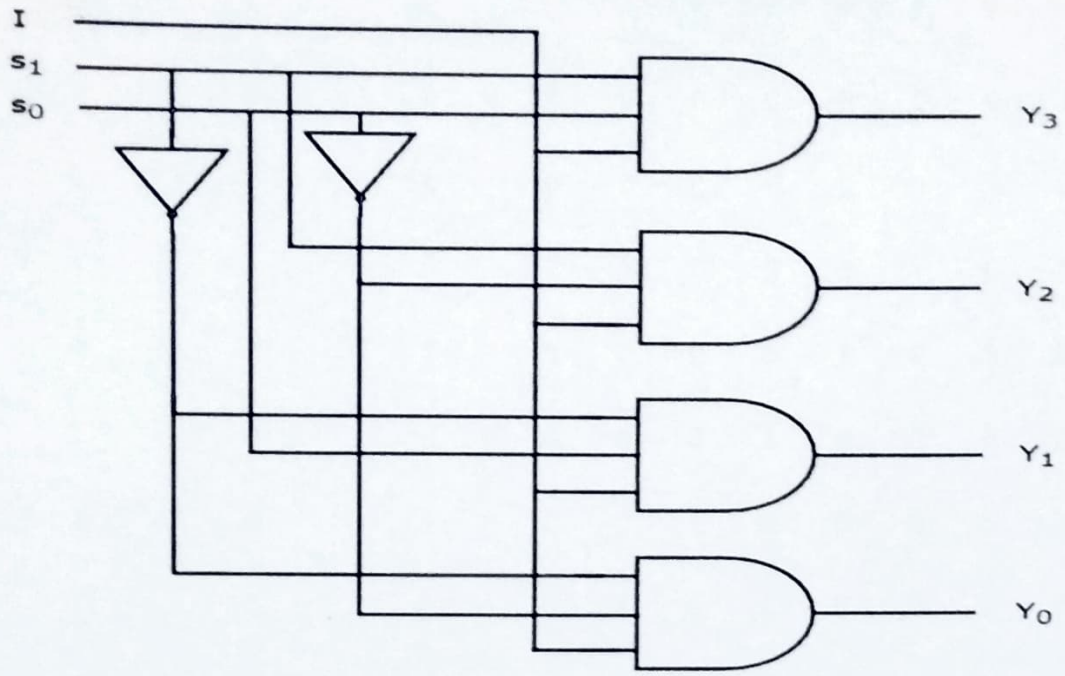
$$Y_2=s_1s_0'/Y2=s1s0'I$$

$$Y_1=s_1's_0/Y1=s1's0I$$

$$Y_0=s_1's_0'/Y0=s1's0'I$$


Ref
1.0us
Time
739.2ns
Interval
260.8ns
Name
Value
100.0ns
200.0ns
300.0ns
400.0ns
500.0n
a3
a2
a1
a0
p_odd
p_even
X
X
X
X
X
X

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.



We can easily understand the operation of the above circuit. Similarly, you can implement 1x8 De-Multiplexer and 1x16 De-Multiplexer by following the same procedure.

Implementation of Higher-order De-Multiplexers

Now, let us implement the following two higher-order De-Multiplexers using lower-order De-Multiplexers.

- 1x8 De-Multiplexer
- 1x16 De-Multiplexer

1x8 De-Multiplexer

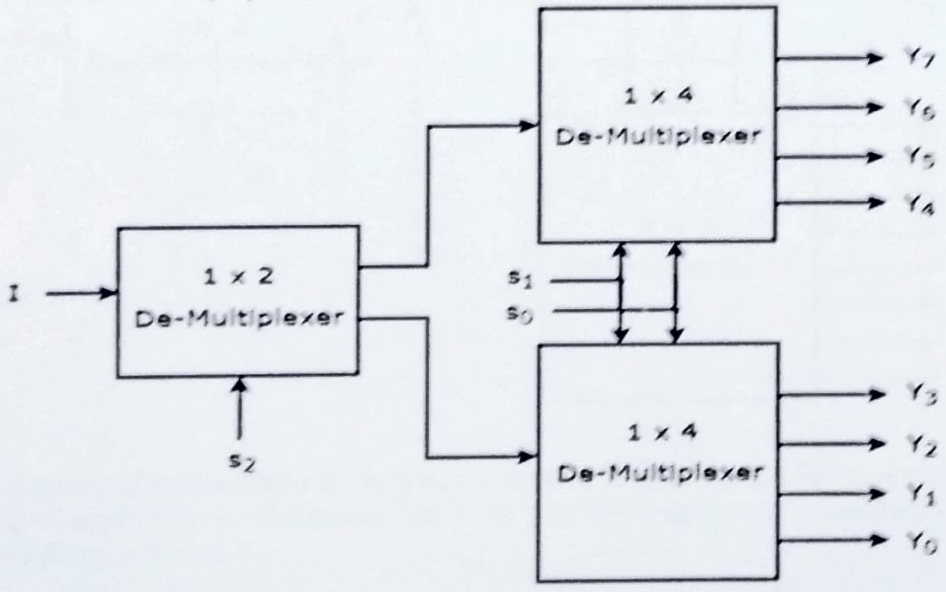
In this section, let us implement 1x8 De-Multiplexer using 1x4 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x4 De-Multiplexer has single input, two selection lines and four outputs. Whereas, 1x8 De-Multiplexer has single input, three selection lines and eight outputs.

So, we require two **1x4 De-Multiplexers** in second stage in order to get the final eight outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x8 De-Multiplexer.

Let the 1x8 De-Multiplexer has one input I, three selection lines s_2, s_1 & s_0 and outputs Y_7 to Y_0. The **Truth table** of 1x8 De-Multiplexer is shown below.

Selection Inputs			Outputs							
s_2	s_1	s_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	I
0	0	1	0	0	0	0	0	0	I	0
0	1	0	0	0	0	0	0	I	0	0
0	1	1	0	0	0	0	I	0	0	0
1	0	0	0	0	0	I	0	0	0	0
1	0	1	0	0	I	0	0	0	0	0
1	1	0	0	I	0	0	0	0	0	0
1	1	1	I	0	0	0	0	0	0	0

We can implement 1x8 De-Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 1x8 De-Multiplexer is shown in the following figure.



The common **selection lines, s_1 & s_0** are applied to both 1x4 De-Multiplexers. The outputs of upper 1x4 De-Multiplexer are Y_7 to Y_4 and the outputs of lower 1x4 De-Multiplexer are Y_3 to Y_0.

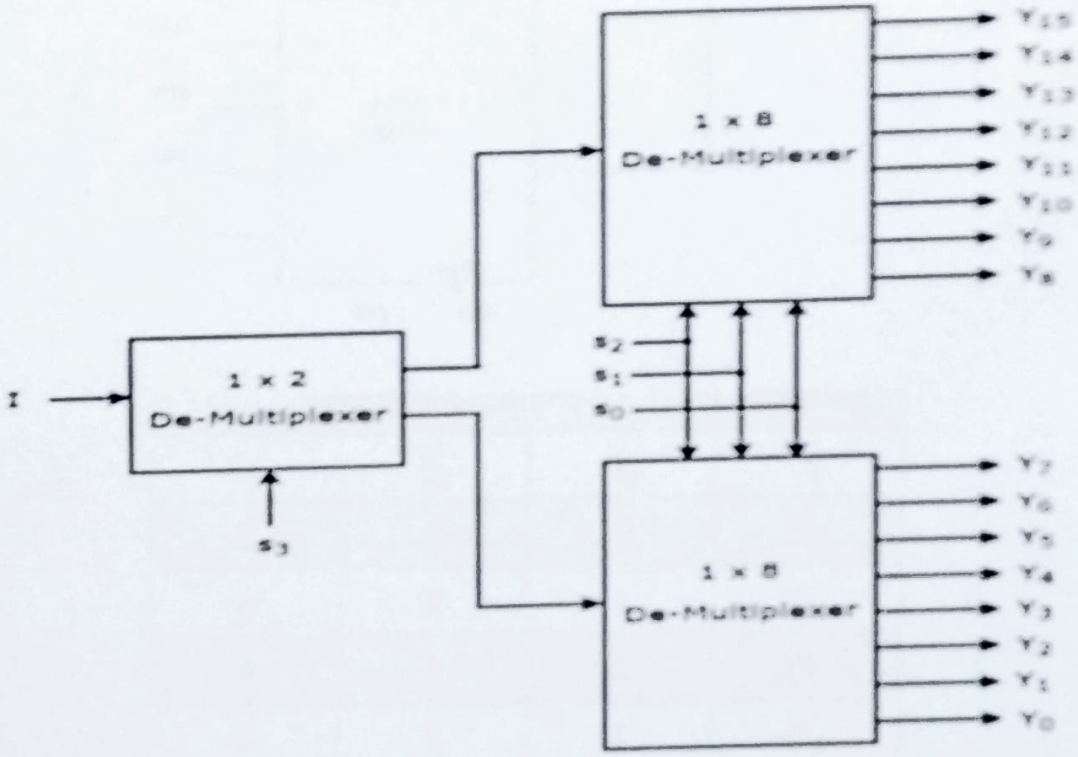
The other **selection line, s_2** is applied to 1x2 De-Multiplexer. If s_2 is zero, then one of the four outputs of lower 1x4 De-Multiplexer will be equal to input, I based on the values of selection lines s_1 & s_0. Similarly, if s_2 is one, then one of the four outputs of upper 1x4 DeMultiplexer will be equal to input, I based on the values of selection lines s_1 & s_0.

1x16 De-Multiplexer

In this section, let us implement 1x16 De-Multiplexer using 1x8 De-Multiplexers and 1x2 De-Multiplexer. We know that 1x8 De-Multiplexer has single input, three selection lines and eight outputs. Whereas, 1x16 De-Multiplexer has single input, four selection lines and sixteen outputs.

So, we require two **1x8 De-Multiplexers** in second stage in order to get the final sixteen outputs. Since, the number of inputs in second stage is two, we require **1x2 DeMultiplexer** in first stage so that the outputs of first stage will be the inputs of second stage. Input of this 1x2 De-Multiplexer will be the overall input of 1x16 De-Multiplexer.

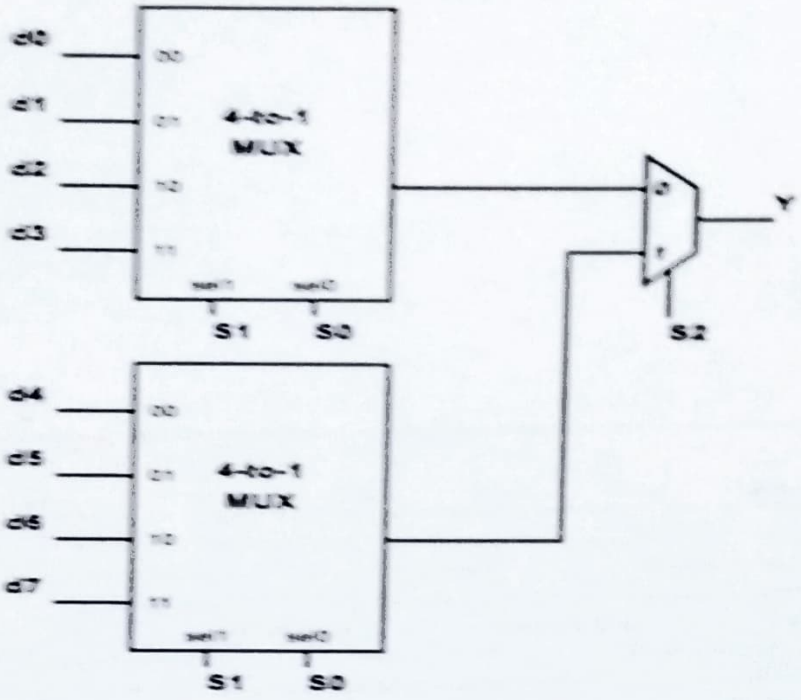
Let the 1x16 De-Multiplexer has one input I, four selection lines s_3, s_2, s_1 & s_0 and outputs Y_{15} to Y_0. The **block diagram** of 1x16 De-Multiplexer using lower order Multiplexers is shown in the following figure.



The common **selection lines s_2, s_1 & s_0** are applied to both 1x8 De-Multiplexers. The outputs of upper 1x8 De-Multiplexer are Y_{15} to Y_8 and the outputs of lower 1x8 DeMultiplexer are Y_7 to Y_0.

The other **selection line, s_3** is applied to 1x2 De-Multiplexer. If s_3 is zero, then one of the eight outputs of lower 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0. Similarly, if s3 is one, then one of the 8 outputs of upper 1x8 De-Multiplexer will be equal to input, I based on the values of selection lines s_2, s_1 & s_0.

Multiplexer

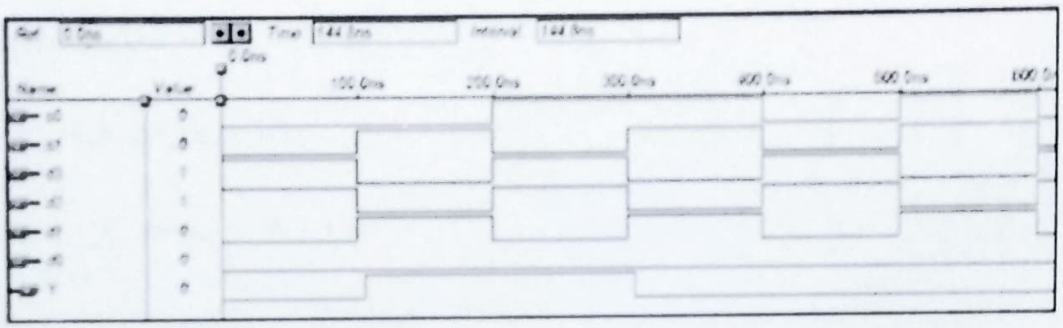


Select Data Inputs			Output
S_2	S_1	S_0	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

VHDL Code for a Multiplexer

```
Library ieee;
use ieee.std_logic_1164.all;

entity mux is
   port(S1,S0,D0,D1,D2,D3:in bit; Y:out bit);
end mux;

architecture data of mux is
begin
   Y<= (not S0 and not S1 and D0) or
      (S0 and not S1 and D1) or
      (not S0 and S1 and D2) or
      (S0 and S1 and D3);
end data;
```

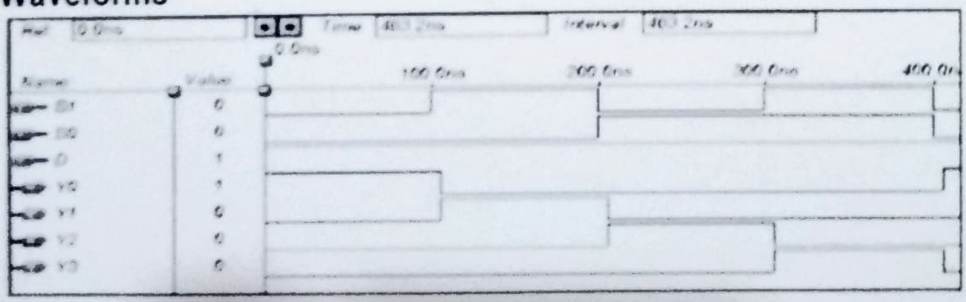
Waveforms



VHDL Code for a Demultiplexer

```
Library ieee;
use ieee.std_logic_1164.all;

entity demux is
   port(S1,S0,D:in bit; Y0,Y1,Y2,Y3:out bit);
end demux;

architecture data of demux is
begin
   Y0<=  ((Not S0) and (Not S1) and D);
   Y1<=  ((Not S0) and S1 and D);
   Y2<=  (S0 and (Not S1) and D);
   Y3<=  (S0 and S1 and D);
end data;
```

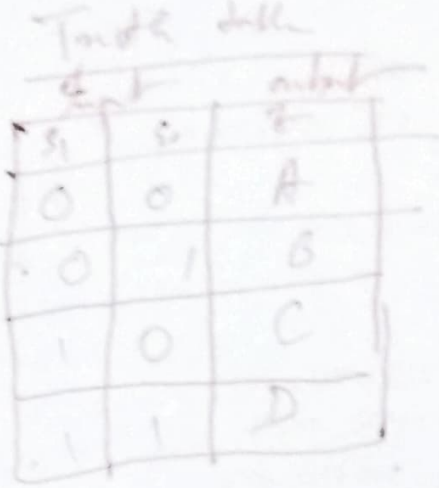
Waveforms

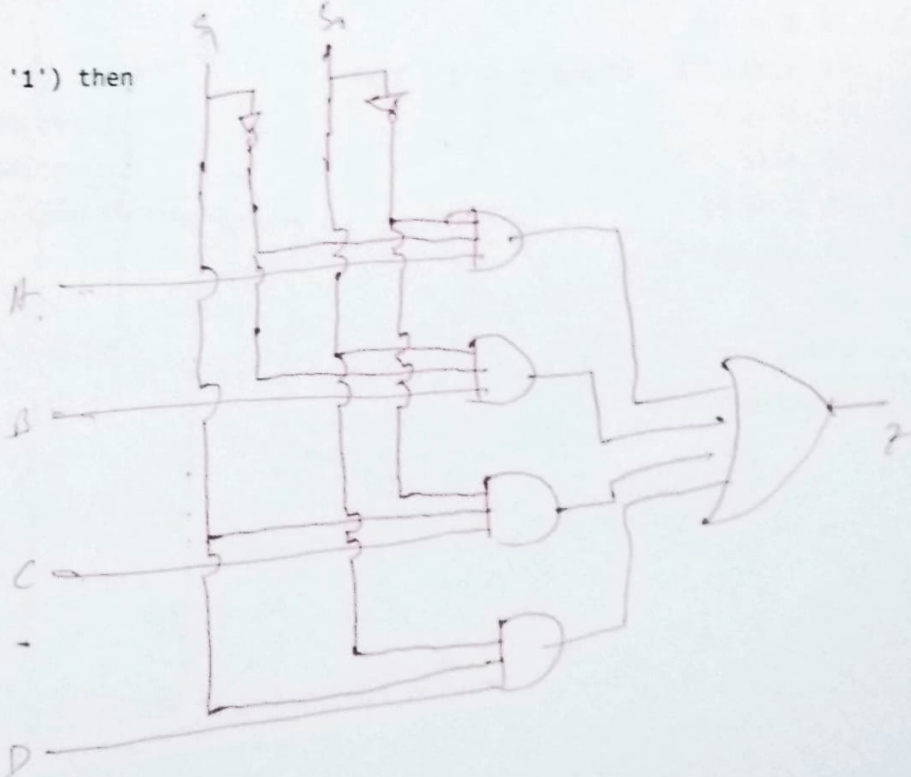


VIEW SOURCE

print

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity mux_4to1 is
5  port(
6
7       A,B,C,D : in STD_LOGIC;
8       S0,S1: in STD_LOGIC;
9       Z: out STD_LOGIC
10   );
11 end mux_4to1;
12
13 architecture bhv of mux_4to1 is
14 begin
15 process (A,B,C,D,S0,S1) is
16 begin
17   if (S0 ='0' and S1 = '0') then
18        Z <= A;
19   elsif (S0 ='1' and S1 = '0') then
20        Z <= B;
21   elsif (S0 ='0' and S1 = '1') then
22        Z <= C;
23   else
24        Z <= D;
25   end if;
26
27 end process;
28 end bhv;
```





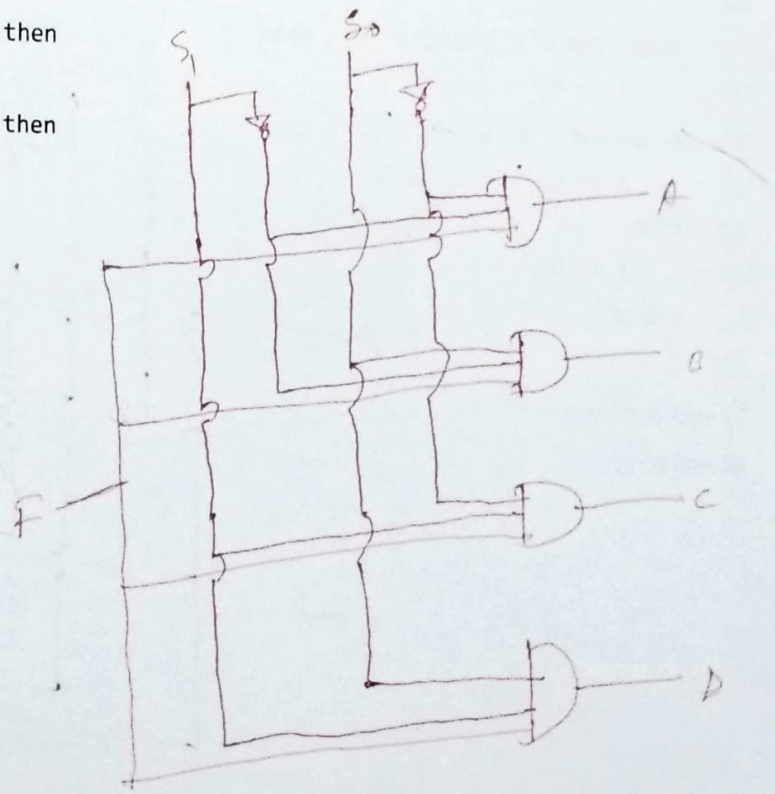
view source

print?

```
1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3
4 entity demux_1to4 is
5  port(
6
7  F : in STD_LOGIC;
8  S0,S1: in STD_LOGIC;
9  A,B,C,D: out STD_LOGIC
10  );
11 end demux_1to4;
12
13 architecture bhv of demux_1to4 is
14 begin
15 process (F,S0,S1) is
16 begin
17  if (S0 ='0' and S1 = '0') then
18  A <= F;
19  elsif (S0 ='1' and S1 = '0') then
20  B <= F;
21  elsif (S0 ='0' and S1 = '1') then
22  C <= F;
23  else
24  D <= F;
25  end if;
26
27 end process;
28 end bhv;
```

Truth table

Input	selection line		Output			
F	S_1	S_0	D	C	B	A
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0
0	X	X	0	0	0	0



VIEW SOURCE

mux4_1

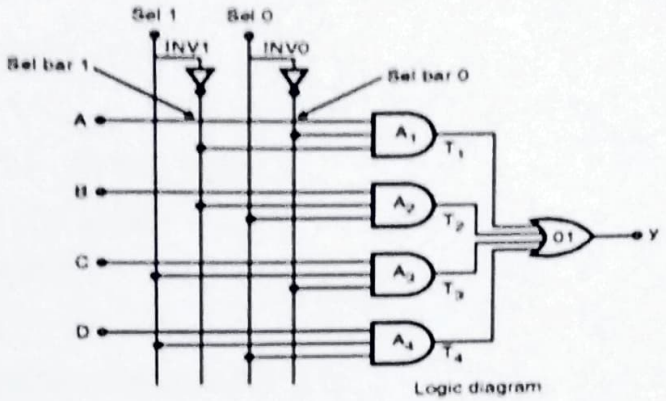
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux4_1 is
port(

A_0,C_0 : in STD_LOGIC;
S0,S1: in STD_LOGIC;
Z: out STD_LOGIC
);
end mux4_1;

architecture Behavioral of mux4_1 is
component mux2_1
port( A_0 : in STD_LOGIC;
S: in STD_LOGIC;
Z: out STD_LOGIC);
end component;
signal temp1, temp2: std_logic;

begin
m1: mux2_1 port map(A_0,S0,temp1);
m2: mux2_1 port map(C_0,S0,temp2);
m3: mux2_1 port map(temp1,temp2,S1,Z);

end Behavioral;
```

view source

print?

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux2_1 is
port(A,B : in STD_LOGIC;
S: in STD_LOGIC;
Z: out STD_LOGIC);
end mux2_1;

architecture Behavioral of mux2_1 is

begin

process (A,B,S) is
begin
if (S ='0') then
Z <= A;
else
Z <= B;
end if;
end process;

end Behavioral;
```

VHDL Code of 4:1 Mux using Different Modeling Styles :



Logic diagram

```
-- Behavioral Modeling of 4:1 mux
library ieee;
use ieee.std_logic_1164.all;
entity MUX4_1 is
port ( Sel : in std_logic_vector(1 downto 0);
A, B, C, D : in std_logic;
Y : out std_logic );
end MUX4_1;
architecture behavior of MUX4_1 is
begin
process (Sel, A, B, C, D)
begin
if (Sel = "00") then
Y<= A;
elsif (Sel = "01") then
Y<= B;
elsif (Sel = "10") then
Y<= C;
else
Y<= D;
end if;
end process;
end behavior;
```

```
-- Structural modeling of 4:1 mux
library ieee;
use ieee.std_logic_1164.all;
entity MUX4_1 is
port ( Sel0,Sel1 : in std_logic;
A, B, C, D : in std_logic;
Y : out std_logic );
end MUX4_1;
architecture structural of MUX4_1 is
component inv
port (pin : in std_logic;
pout :out std_logic);
end component;
component and3
port (a0,a1,a2: in std_logic;
aout:out std_logic);
end component;
component or4
port (r0,r1,r2,r3:in std_logic;
rout:out std_logic);
end component;
signal selbar0,selbar1,t1,t2,t3,t4: std_logic;
begin
INV0: inv port map (Sel0, selbar1);
INV1: inv port map (Sel1, selbar1);
A1: and3 port map (A, selbar0, selbar1, t1);
A2: and3 port map (B, Sel0, selbar1, t2);
A3: and3 port map (C, selbar0, Sel1, t2);
A4: and3 port map (D, Sel0, Sel1, t4);
O1: or4 port map (t1, t2, t3, t4, Y);
end structural;
```

```
-- Dataflow modeling of 4:1 mux

architecture dataflow of MUX4_1 is

signal selbar0,selbar1,t1,t2,t3,t4: std_logic;

begin

selbar0<=not sel0;

selbar1<=not sel1;

t1<=A and selbar0 and selbar1;

t2<=B and sel0 and selbar1;

t3<=C and selbar0 and sel1;

t4<=D and sel0 and sel1;

Y<= t1 or t2 or t3 or t4;

end dataflow;
```

Code for 16x1 mux

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity kanhe_16x1mux is
port(a:in std_logic_vector(15 downto 0);
 s: in std_logic_vector(3 downto 0);
Z:out std_logic);
End kanhe_16x1mux;

Architecture kanhe_16x1mux1 of kanhe_16x1mux is
signal z1,z2,z3,z4:std_logic;

component kanhe_4x1mux is
port(a,b,c,d,s0,s1:in std_logic;
Q:out std_logic);

End component;

Begin
M1: kanhe_4x1mux port map(a(0),a(1),a(2),a(3),s(0),s(1),z1);
m2: kanhe_4x1mux port map(a(4),a(5),a(6),a(7),s(0),s(1),z2);
m3: kanhe_4x1mux port map(a(8),a(9),a(10),a(11),s(0),s(1),z3);
m4: kanhe_4x1mux port map(a(12),a(13),a(14),a(15),s(0),s(1),z4);
m5: kanhe_4x1mux port map(z1,z2,z3,z4,s(2),s(3),z);

End kanhe_16x1mux1;
```

vhdl code for 16:1 mux using 8:1 :

```
library IEEE;

entity mux16using8 is
      port(d:in bit_vector(15 downto 0);
      s:in bit_vector(3 downto 0);
      e:in bit;
      y:out bit);
end mux16using8;

architecture Behavioral of mux16using8 is
component mux8 is
     port(d:in bit_vector(7 downto 0);
     e:in bit;
     s:in bit_vector(2 downto 0);
     y:out bit);
end component;
component mux2 is
     port ( d:in bit_vector(1 downto 0);
     e,s:in bit;
     y:out bit);
end component;
signal m:bit_vector(1 downto 0);
begin
     m1:mux8 port map(d(7 downto 0),e,s(2 downto 0),m(0));
     m2:mux8 port map(d(15 downto 8),e,s(2 downto 0),m(1));
     m3:mux2 port map(m,e,s(3),y);
end Behavioral;
```

Multiplexer using case when

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;

ENTITY MUX8_1 IS
PORT(A:IN STD_LOGIC_VECTOR( DOWNTO ));
B:IN STD_LOGIC_VECTOR( DOWNTO );
C:IN STD_LOGIC_VECTOR( DOWNTO );
D:IN STD_LOGIC_VECTOR( DOWNTO );
E:IN STD_LOGIC_VECTOR( DOWNTO );
F:IN STD_LOGIC_VECTOR( DOWNTO );
G:IN STD_LOGIC_VECTOR( DOWNTO );
SEL:IN STD_LOGIC_VECTOR( DOWNTO ); DOUT:OUT STD_LOGIC_VECTOR( DOWNTO ));
END MUX8_1;

ARCHITECTURE BEH123 OF MUX8_1 IS

BEGIN
PROCESS(A,B,C,D,E,F,G,SEL)
BEGIN
CASE SEL IS
WHEN"000"=>DOUT<=A;
WHEN"001"=>DOUT<=B;
WHEN"010"=>DOUT<=C;
WHEN"011"=>DOUT<=D;
WHEN"100"=>DOUT<=E;
WHEN"101"=>DOUT<=F;
WHEN"110"=>DOUT<=G;
WHEN OTHERS=>DOUT<=A;
END CASE;
END PROCESS;
END BEH123;
```

vhdl code for 16:1 mux using 4:1 :

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;

entity kanhe_4x1mux is
port(a,b,c,d : in std_logic;
S0,s1 : in std_logic;
q : out std_logic);
end kanhe_4x1mux;

Architecture kanhe_4x1mux1 of kanhe_4x1mux is
Begin
Process(a,b,c,d,s0,s1)
Begin

If s0 ='0' and s1 ='0' then q <= a;
Elsif s0 ='1' and s1 ='0' then q <= b;
elsif s0 ='0' and s1='1' then q <= c;
else q <=d;
end if;
End process;
End kanhe_4x1mux1;
```