VLSI Lab Report Assignment 6

BCSE 4th Year 2nd Semester

Name: ***Priti Shaw***
Roll No. : ***001710501076***
Batch: ***A3***

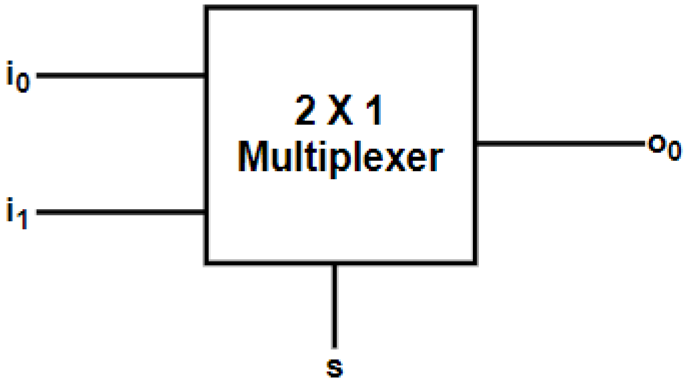
1. 2 X 1 Multiplexer

Description

Implement a 2X1 multiplexer using gate level modelling.

2 X 1 Multiplexer takes two inputs and using one select line, it selects any one of the two input lines and gives it as output. A bit binary number formed by the select lines determine the index of the input line which will be selected.

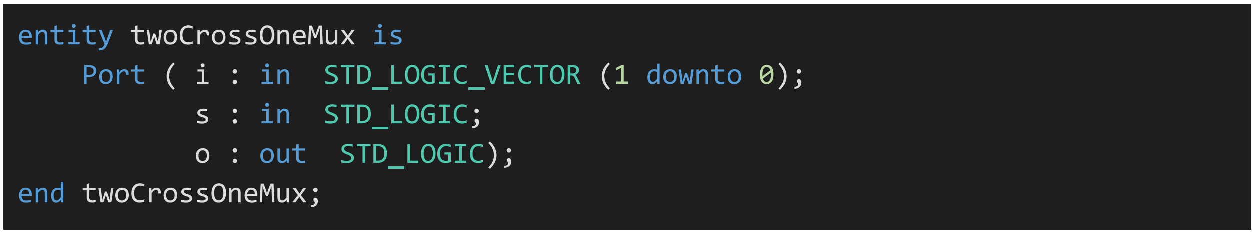
Block Diagram



Truth Table

s	o_0
0	i_0
1	i_1

Entity



```
entity twoCrossOneMux is
    Port ( i : in  STD_LOGIC_VECTOR (1 downto 0);
           s : in  STD_LOGIC;
           o : out  STD_LOGIC);
end twoCrossOneMux;
```

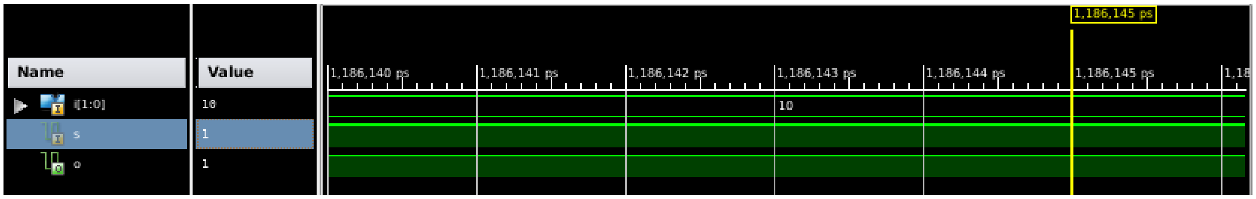
Architecture

```
architecture Behavioral of twoCrossOneMux is

begin
p1:process(i,s)
      variable oo:std_logic;
      begin
            proc: twoCrossOneMuxProcedure(i(1 downto 0),s,oo);
            o<=oo;
      end process;

end Behavioral;
```

Procedure of 2X1 Multiplexer in Package

```
procedure twoCrossOneMuxProcedure(i: in std_logic_vector;s:in std_logic;o:
out std_logic) is
       variable ii: std_logic_vector(1 downto 0);
       begin
                  ii(0):= not(s) and i(0);
                  ii(1):= s and i(1);
                  o:= ii(0) or ii(1);
       end procedure;
```

Timing Diagram



Name	Value
i[1:0]	10
s	1
o	1

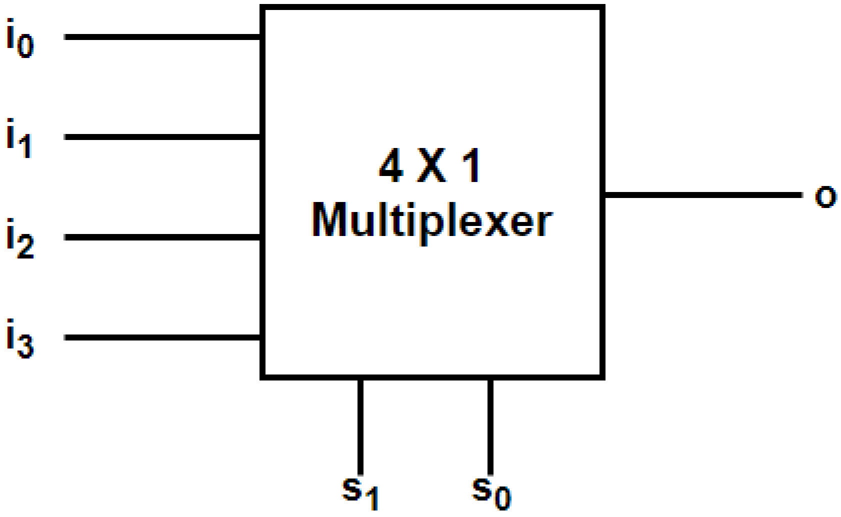
2. 4 X 1 Multiplexer

Description

Implement a 4X1 multiplexer using gate level modelling.

4 X 1 Multiplexer takes four inputs and using two select lines, it selects any one of the four input lines and gives it as output. The two bit binary number formed by the select lines determine the index of the input line which will be selected.

Block Diagram



Truth Table

s_1	s_0	o_0
0	0	i_0
0	1	i_1
1	0	i_2
1	1	i_3

Entity

```
entity fourCrossOneMux is
    Port ( i : in  STD_LOGIC_VECTOR (3 downto 0);
           s : in  STD_LOGIC_VECTOR (1 downto 0);
           o : out  STD_LOGIC);
end fourCrossOneMux;
```

Architecture

```
architecture Behavioral of fourCrossOneMux is

begin
      p1:process(i,s)
      variable oo:std_logic;
      begin
            proc: fourCrossOneMuxProcedure(i(3 downto 0),s(1 downto 0),oo);
            o<=oo;
      end process;
end Behavioral;
```

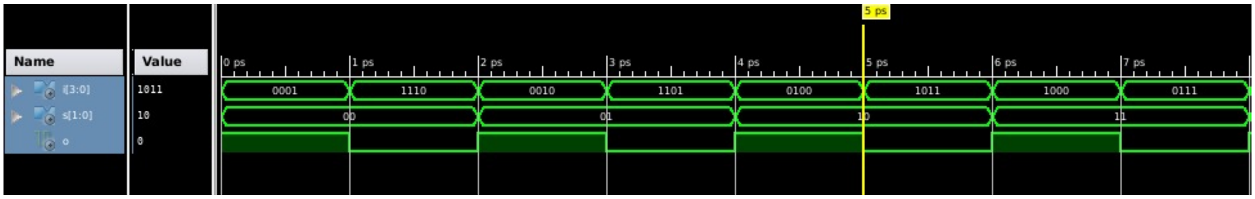
Procedure of 4 X 1 Multiplexer in Package

```
procedure fourCrossOneMuxProcedure(i: in std_logic_vector;s:in
std_logic_vector;o: out std_logic) is
       variable ii: std_logic_vector(3 downto 0);
       begin
                  ii(0):= not(s(1)) and not(s(0)) and i(0);
                  ii(1):= not(s(1)) and s(0) and i(1);
                  ii(2):= s(1) and not(s(0)) and i(2);
                  ii(3):= s(1) and s(0) and i(3);
                  o:= ii(0) or ii(1) or ii(2) or ii(3);
       end procedure;
```

TestBench

```
stim_proc: process
      variable k:integer;
      variable binary:std_logic_vector(1 downto 0);
   begin
                  for k in 0 to 3 loop
                        prock: decimalToBinaryProcedure(k,2,binary(1 downto
0));
                        s(1 downto 0)<=binary(1 downto 0);

                        i(3 downto 0)<=(others=>'0');
                        i(k)<='1';
                        wait for 1 ps;

                        i(3 downto 0)<=(others=>'1');
                        i(k)<='0';
                        wait for 1 ps;
                  end loop;
   end process;
END;
```

Timing Diagram



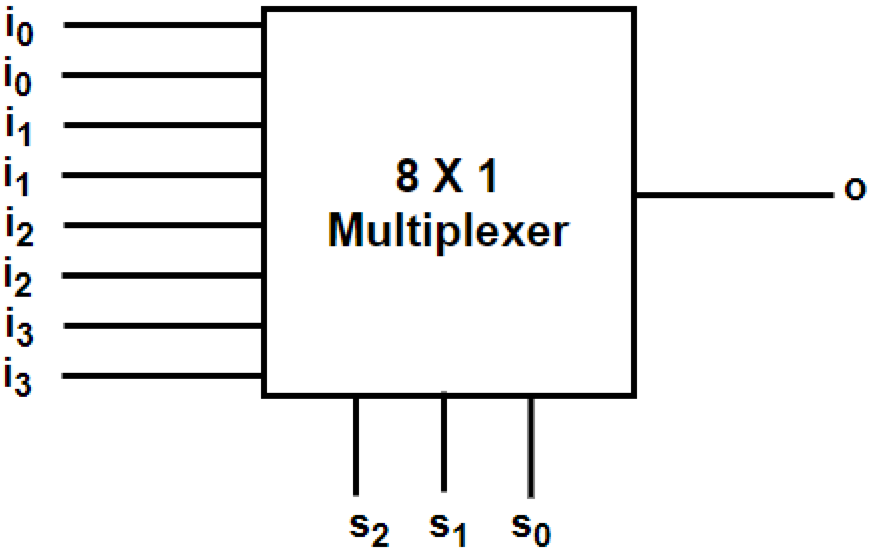
3. 8 X 1 Multiplexer using 4 X1 and 2 X 1 multiplexer

Description

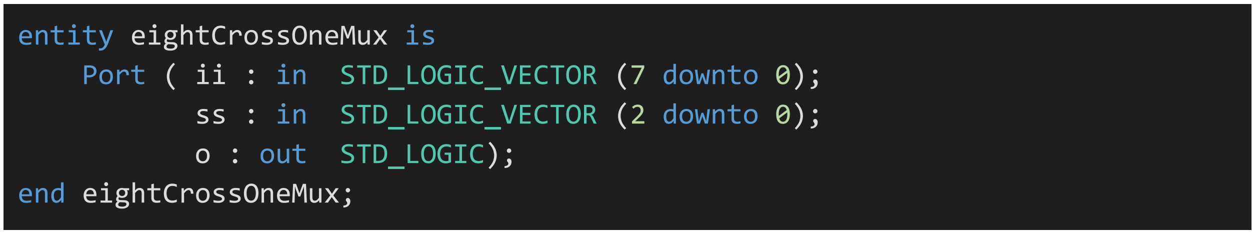
Implement a 8X1 multiplexer using 4X1 multiplexer and 2X1 multiplexer.

8 X 1 Multiplexer takes eight inputs and using three select lines, it selects any one of the eight input lines and gives it as output. The three bit binary number formed by the three select lines determine the index of the input line which will be selected.

Block Diagram

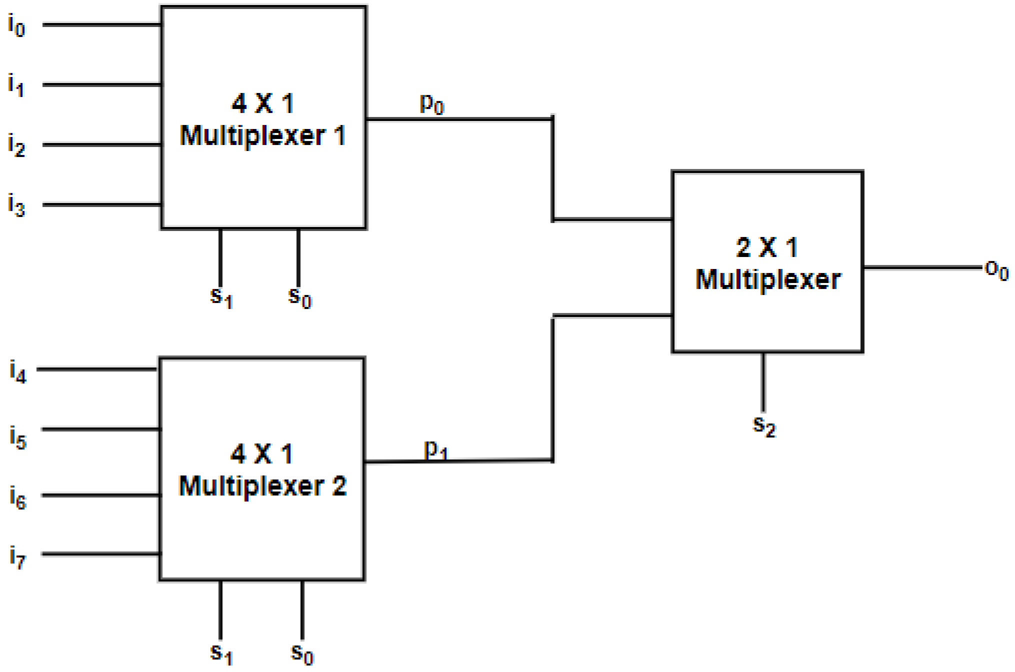


Entity



```
entity eightCrossOneMux is
    Port ( ii : in  STD_LOGIC_VECTOR (7 downto 0);
           ss : in  STD_LOGIC_VECTOR (2 downto 0);
           o : out  STD_LOGIC);
end eightCrossOneMux;
```

Circuit Diagram



Truth Table

s_2	s_1	s_0	o_0
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

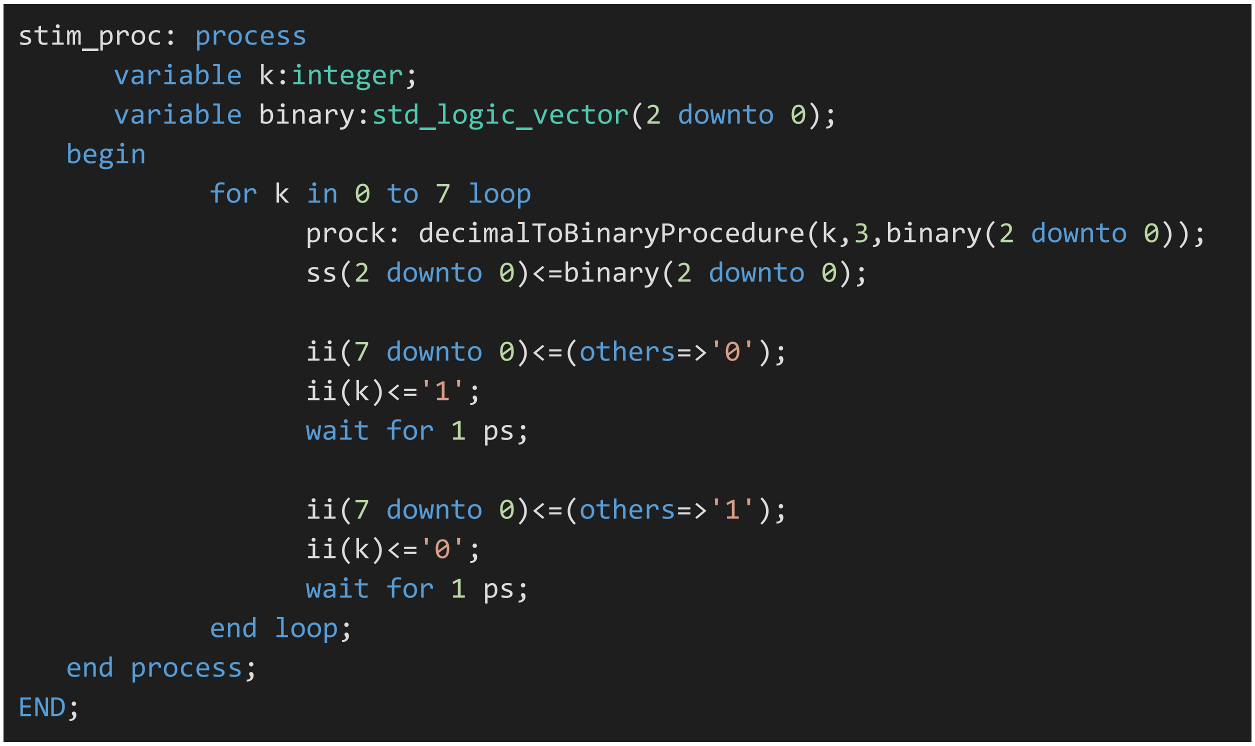
Architecture

```
architecture Behavioral of eightCrossOneMux is

begin
p1:process(ii,ss)
variable oo:std_logic;
begin
      procc:eightCrossOneMuxProcedure(ii(7 downto 0),ss(2 downto 0),oo);
      o<=oo;
end process;

end Behavioral;
```

Procedure of 8 X 1 Multiplexer in Package

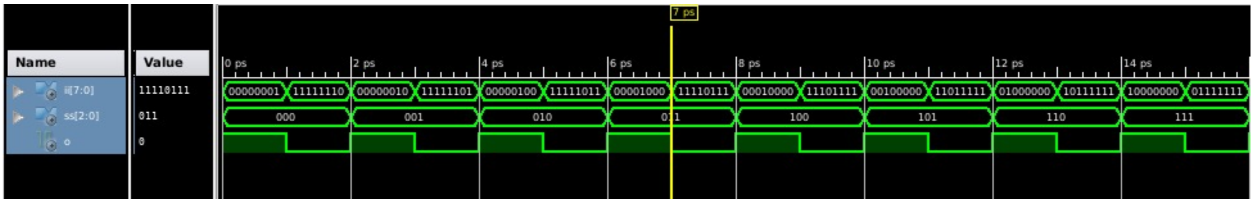
```
procedure eightCrossOneMuxProcedure(i: in std_logic_vector;s:in
std_logic_vector;o: out std_logic) is
       variable p:std_logic_vector(1 downto 0);
       variable ii:std_logic_vector(3 downto 0);
       variable ss:std_logic_vector(1 downto 0);
       variable k:integer;
       begin
             for k in 0 to 1 loop
                  ii(3 downto 0):= i(((4*k)+3) downto (4*k));
                  ss(1 downto 0):= s(1 downto 0);
                  prock:fourCrossOneMuxProcedure(ii(3 downto 0),ss(1 downto
0),p(k));
             end loop;
             procc:twoCrossOneMuxProcedure(p(1 downto 0),s(2),o);
       end procedure;
```

TestBench



```
stim_proc: process
      variable k:integer;
      variable binary:std_logic_vector(2 downto 0);
   begin
            for k in 0 to 7 loop
                  prock: decimalToBinaryProcedure(k,3,binary(2 downto 0));
                  ss(2 downto 0)<=binary(2 downto 0);

                  ii(7 downto 0)<=(others=>'0');
                  ii(k)<='1';
                  wait for 1 ps;

                  ii(7 downto 0)<=(others=>'1');
                  ii(k)<='0';
                  wait for 1 ps;
            end loop;
   end process;
END;
```

Timing Diagram



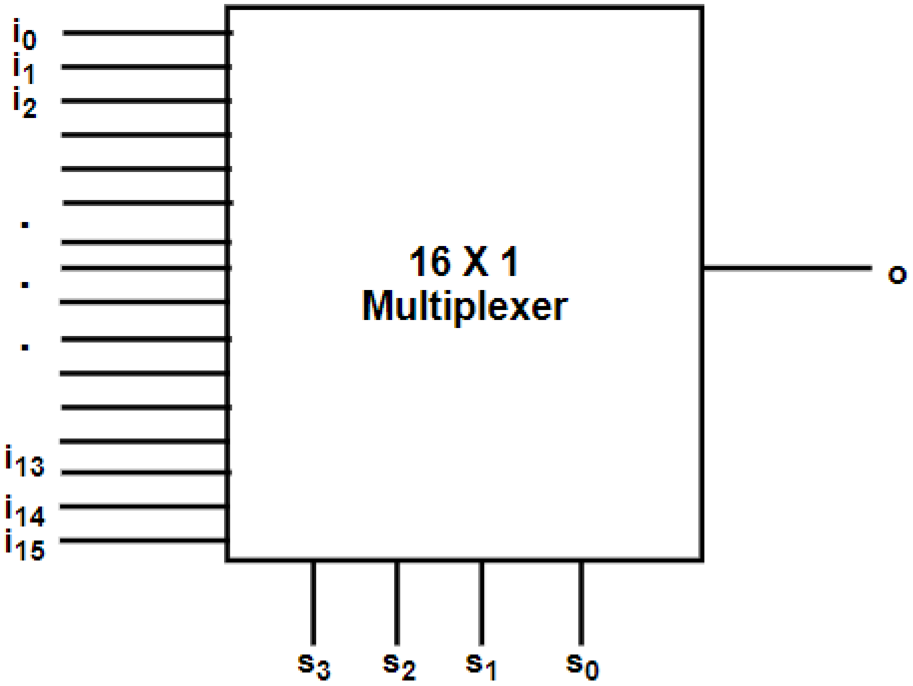
4. 16 X 1 Multiplexer using 8 X 1 and 2 X 1 multiplexer

Description

Implement a 16X1 multiplexer using 8X1 multiplexer and 2X1 multiplexer.

16X1 Multiplexer takes sixteen inputs and using four select lines, it selects any one of the sixteen input lines and gives it as output. The four bit binary number formed by the four select lines determine the index of the input line which will be selected.

Block Diagram



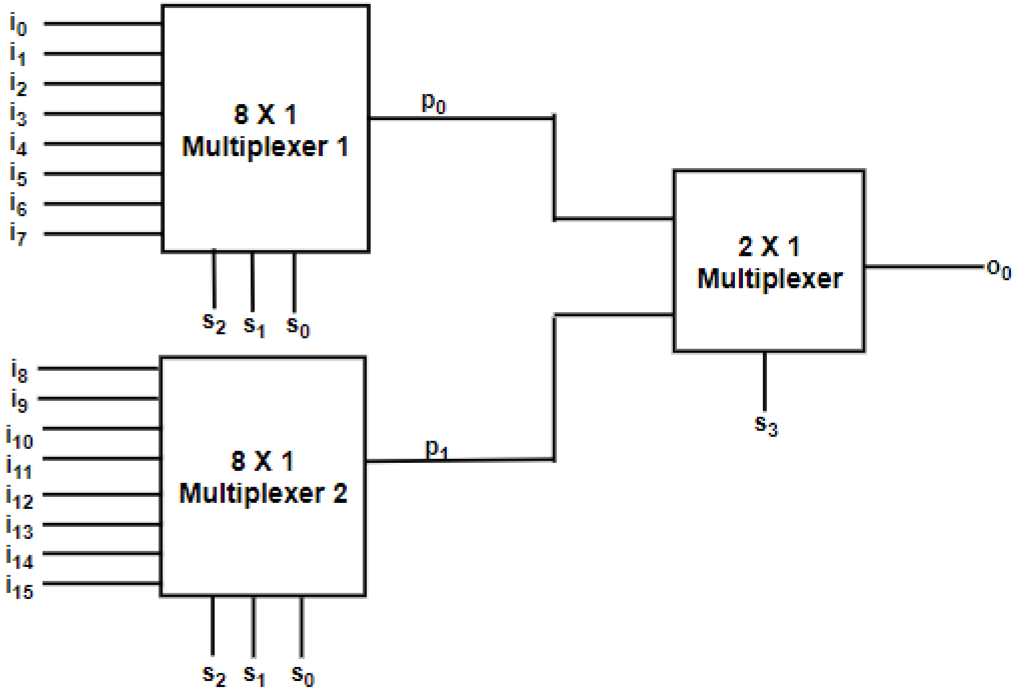
Entity

```
entity sixteenCrossOneMux82 is
    Port ( i : in  STD_LOGIC_VECTOR (15 downto 0);
           s : in  STD_LOGIC_VECTOR (3 downto 0);
           o : out  STD_LOGIC);
end sixteenCrossOneMux82;
```

Truth Table

s_3	s_2	s_1	s_0	o_0
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}

Circuit Diagram



Architecture

```
architecture Behavioral of sixteenCrossOneMux82 is

begin
process(i, s)
            variable oo: STD_LOGIC;
            begin
                  sixteenCrossOneMuxUsing82Procedure(i,s,oo);
                  o<=oo;
            end process;
end Behavioral;
```

Procedure of 16 X 1 Multiplexer in Package

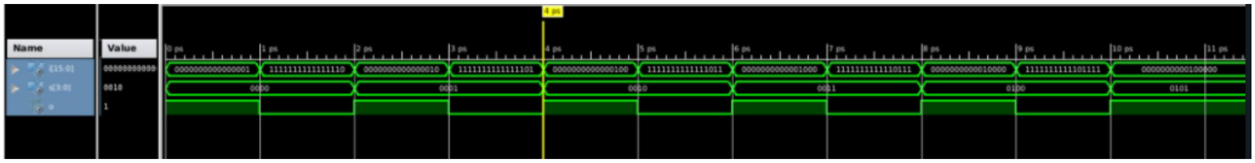
```
procedure sixteenCrossOneMuxUsing82Procedure(i: in std_logic_vector;s:in
std_logic_vector;o: out std_logic) is
       variable ss:std_logic_vector(2 downto 0);
       variable p:std_logic_vector(1 downto 0);
       variable a:std_logic_vector(7 downto 0);
       variable k:integer;
       begin
             for k in 0 to 1 loop
```

```
                        a:=i(8*k+7 downto 8*k);
                        ss:=s(2 downto 0);
                        eightCrossOneMuxProcedure(a,ss,p(k));
                  end loop;
                  twoCrossOneMuxProcedure(p,s(3),o);
       end procedure;
```

TestBench

```
stim_proc: process
      variable k:integer;
      variable binary:std_logic_vector(3 downto 0);
   begin
            for k in 0 to 15 loop
                  prock: decimalToBinaryProcedure(k,4,binary(3 downto 0));
                  s(3 downto 0)<=binary(3 downto 0);

                  i(15 downto 0)<=(others=>'0');
                  i(k)<='1';
                  wait for 1 ps;

                  i(15 downto 0)<=(others=>'1');
                  i(k)<='0';
                  wait for 1 ps;
            end loop;
   end process;
```

Timing Diagram



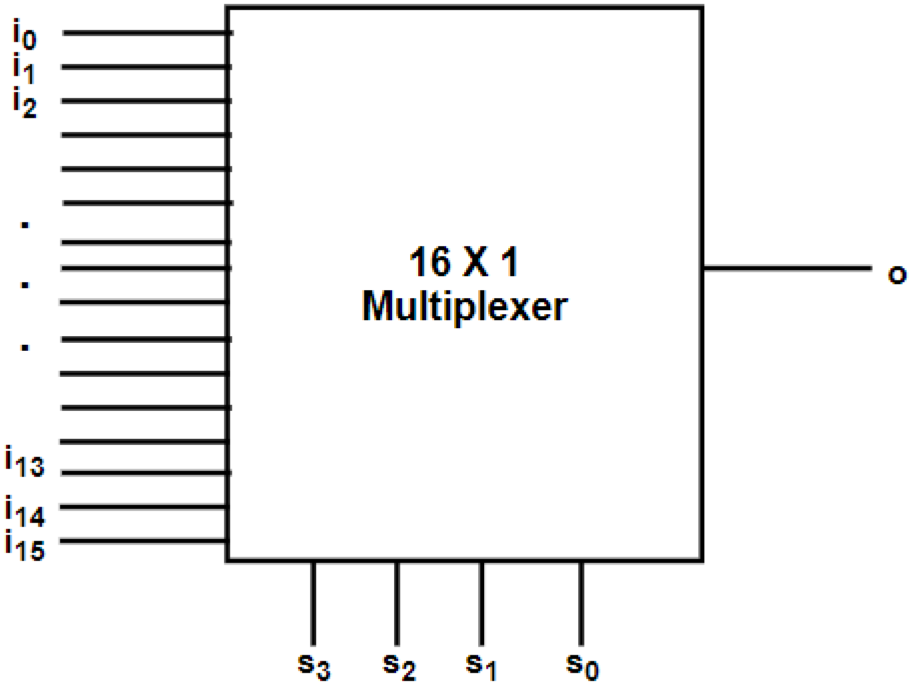
5. 16 X 1 Multiplexer using 4 X 1 multiplexer

Description

Implement a 16X1 multiplexer using a 4X1 multiplexer only.

16X1 Multiplexer takes sixteen inputs and using four select lines, it selects any one of the sixteen input lines and gives it as output. The four bit binary number formed by the four select lines determine the index of the input line which will be selected.

Block Diagram



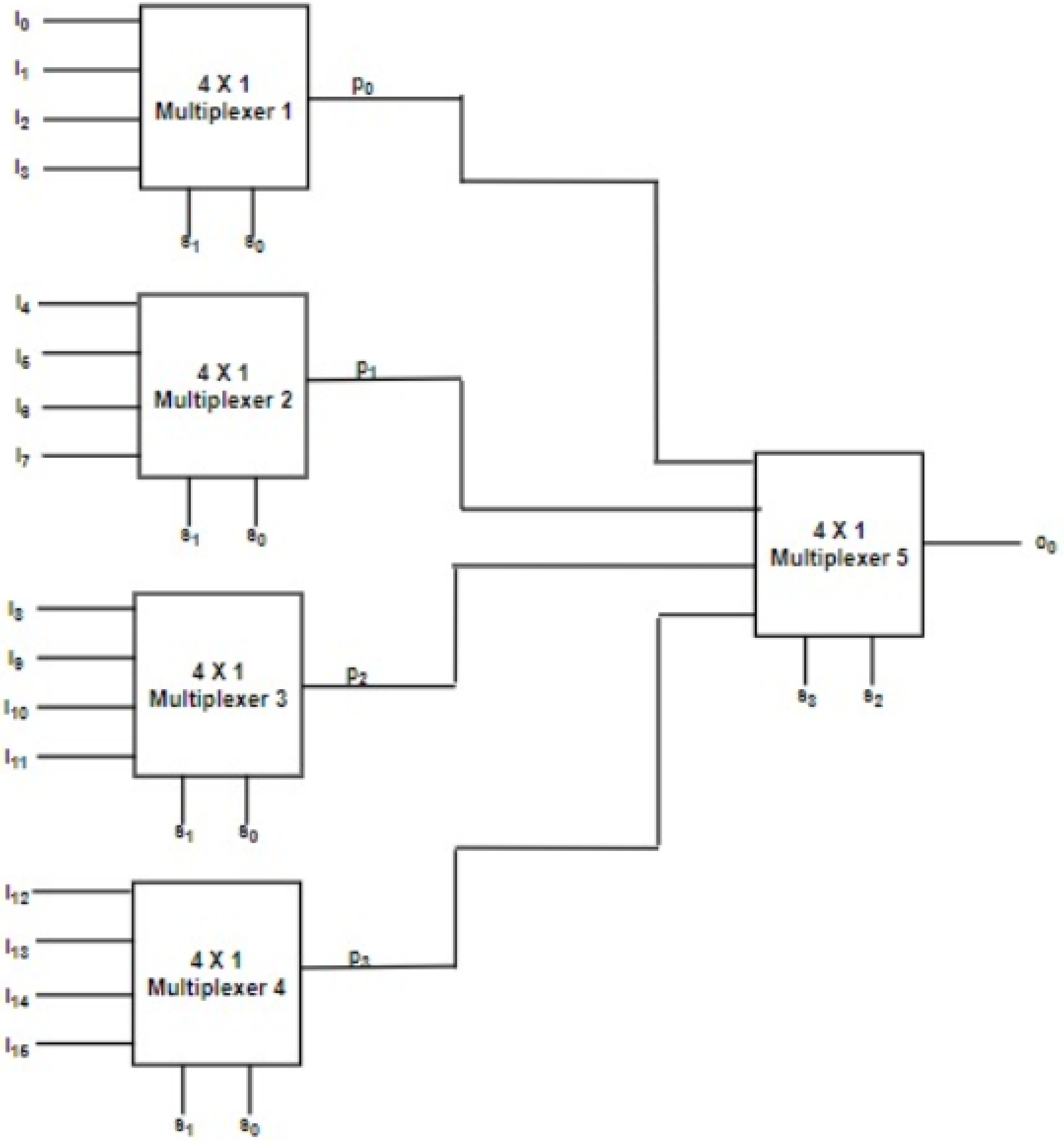
Entity

```
entity sixteenCrossOneMux is
    Port ( i : in  STD_LOGIC_VECTOR (15 downto 0);
                    s : in  STD_LOGIC_VECTOR(3 downto 0);
           o : out STD_LOGIC);
end sixteenCrossOneMux;
```

Truth Table

s_3	s_2	s_1	s_0	o_0
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}

Circuit Diagram



Architecture

```
architecture Behavioral of sixteenCrossOneMux is
begin
	process(i, s)
		variable oo: STD_LOGIC;
		begin
			sixteenCrossOneMuxProcedure(i,s,oo);
			o<=oo;
		end process;
end Behavioral;
```

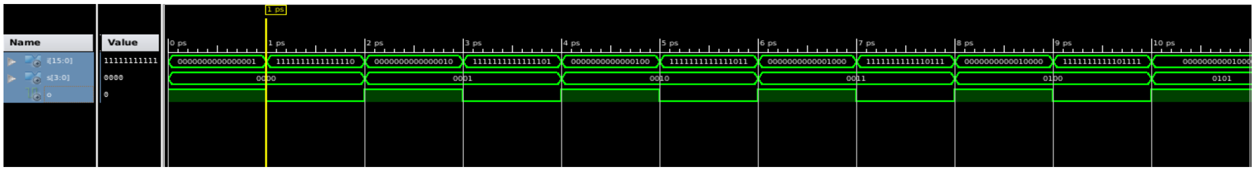
Procedure of 16 X 1 Multiplexer in Package

```
procedure sixteenCrossOneMuxProcedure(i: in std_logic_vector;s:in
std_logic_vector;o: out std_logic) is
       variable ss:std_logic_vector(1 downto 0);
       variable a,p:std_logic_vector(3 downto 0);
       begin
                  for k in 0 to 3 loop
                        a:=i(4*k+3 downto 4*k);
                        ss:=s(1 downto 0);
                        fourCrossOneMuxProcedure(a,ss,p(k));
                  end loop;
                  ss:=s(3 downto 2);
                  fourCrossOneMuxProcedure(p,ss,o);
       end procedure;
```

TestBench

```
stim_proc: process
      variable k:integer;
      variable binary:std_logic_vector(3 downto 0);
   begin
            for k in 0 to 15 loop
                  prock: decimalToBinaryProcedure(k,4,binary(3 downto 0));
                  s(3 downto 0)<=binary(3 downto 0);

                  i(15 downto 0)<=(others=>'0');
                  i(k)<='1';
                  wait for 1 ps;

                  i(15 downto 0)<=(others=>'1');
                  i(k)<='0';
                  wait for 1 ps;
            end loop;
   end process;
```

Timing Diagram


1 ps
Name
Value
i[15:0]
s[3:0]
o
11111111111
0000
0
0 ps
1 ps
2 ps
3 ps
4 ps
5 ps
6 ps
7 ps
8 ps
9 ps
10 ps
0000000000000001
1111111111111110
0000000000000010
1111111111111101
0000000000000100
1111111111111011
0000000000001000
1111111111110111
0000000000010000
1111111111101111
00000000001000
0000
0001
0010
0011
0100
0101