

Experiment - 7

Aim: To design

1. Shift Register
2. Bidirectional register with parallel load

Software Used: ModelSim

Theory: A register is a device which is used to store multiple bits of data. Shift register is a group of register which can move data within registers and inout of register by applying clock pulses. Shift register are of 4 types:-

1. Serial In Serial Out Register: The shift register which allows serial input & produces a serial output is known as serial in serial out register. Since there is only one output. The data leaves the shift register one bit at a time in serial pattern.

2. Serial In Parallel Out Register: The shift register which allows serial input & produces a parallel output is known as serial in parallel out shift register. The output of the first flip flop is connected to input of next flip flop & so on. All these flip flops are synchronous with each other since same clock is applied to each flip flop.

3. Parallel In Serial Out Register:

The shift register which allows parallel input & produces serial output is known as parallel in serial out register. Serial output is directly connected to all flip flops. The output of previous flip flop & parallel data input are connected to MUX & output is connected to next flip flop.

4. Parallel In Parallel Out Register:

The shift register which allows parallel input & also produces parallel output is known as parallel output register. In this is known as parallel in parallel out register. In this type of register, there is no interconnection b/w individual flip flops since no serial shifting of data is required. Data is given as input separately for each flip flop and in same way output is collected individually from each flip flop.

Bidirectional Shift Register with parallel load:

Bidirectional shift register are registers which are capable of shifting the data either right or left depending on the mode selected. If the mode selected is high (1), the data is shifted towards right direction & if mode selected is low (0), the data will be shifted towards left direction. The circuit consist of four D flip flops, the input data is connected at 2 ends of circuit depending on the mode selected only one and gate is in the active state

CODE:

SISO STRUCTURAL

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

entity D_FlipFlop is
port( din: in std_logic;
clk: in std_logic;
q: out std_logic);
end D_FlipFlop;

architecture bin of D_FlipFlop is
begin
process(clk,din)
begin
if(clk='1' and clk'EVENT) then
q <= din;
end if;
end process;
end bin;

library ieee;
use ieee.std_logic_1164.all;

entity siso is
port(clk,Sin:in std_logic;
Sout:out std_logic);
end siso;
architecture structural of siso is
component D_FlipFlop is
port( din: in std_logic;
clk: in std_logic;
q: out std_logic);
end component;
signal t1,t2,t3:std_logic;
begin
ff1:D_FlipFlop port map(Sin,clk,t1);
ff2:D_FlipFlop port map(t1,clk,t2);
ff3:D_FlipFlop port map(t2,clk,t3);
ff4:D_FlipFlop port map(t3,clk,Sout);
end structural;
```

SISO BEHAVOURAL

```

library ieee;
use ieee.std_logic_1164.all;

entity siso is
port(SI:in std_logic;
CLK:in std_logic;
SO:out std_logic);
end siso;

architecture behavioural of siso is
signal temp:std_logic_vector(3 downto 0);
begin
process (CLK)
begin
if(CLK'event and CLK='1') then
for i in 0 to 2 loop
temp(i+1)<=temp(i);
end loop;
temp(0)<=SI;
end if;
end process;
SO<=temp(3);
end behavioural;

```

SIFO BEHAVOURAL

```

library ieee;
use ieee.std_logic_1164.all;

entity sipo is
port(
CLK: in std_logic;
SI: in std_logic;
Q: inout std_logic_vector(3 downto 0) );
end sipo;

```

architecture behavioural of sipo is

```

begin
process (clk)
begin
if (CLK'event and CLK='1') then
Q(3 downto 1) <= Q(2 downto 0);

```

```
Q(0) <= SI;  
end if;  
end process;  
end behavioural;
```

SIFO STRUCTURAL

```
library ieee;  
use ieee.std_logic_1164.all;  
  
entity D_FlipFlop is  
port( din: in std_logic;  
clk: in std_logic;  
q: out std_logic);  
end D_FlipFlop;  
  
architecture bin of D_FlipFlop is  
begin  
process(clk,din)  
begin  
if(clk='1' and clk'EVENT) then  
q <= din;  
end if;  
end process;  
end bin;  
  
library ieee;  
use ieee.std_logic_1164.all;  
  
entity sipo is  
port(clk,Sin:in std_logic;  
Sout: inout std_logic_vector(3 downto 0));  
end sipo;  
architecture structural of sipo is  
component D_FlipFlop is  
port( din: in std_logic;  
clk: in std_logic;  
q: out std_logic);  
end component;  
begin  
ff1:D_FlipFlop port map(Sin,clk,Sout(3));  
ff2:D_FlipFlop port map(Sout(3),clk,Sout(2));  
ff3:D_FlipFlop port map(Sout(2),clk,Sout(1));  
ff4:D_FlipFlop port map(Sout(1),clk,Sout(0));  
end structural;
```

PIPO BEHAVOURAL

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

entity pipo is
port(
clk : in std_logic;
D: in std_logic_vector(3 downto 0);
Q: out std_logic_vector(3 downto 0)
);
end pipo;

architecture behavioural of pipo is
begin
process (clk)
begin
if (clk'event and clk='1') then
Q <= D;
end if;
end process;
end behavioural;
```

PIPO STRUCTURAL

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

entity D_FlipFlop is
port( din: in std_logic;
clk: in std_logic;
q: out std_logic);
end D_FlipFlop;

architecture bin of D_FlipFlop is
begin
process(clk,din)
begin
if(clk='1' and clk'EVENT) then
q <= din;
```

```

end if;
end process;
end bin;

library ieee;
use ieee.std_logic_1164.all;

entity pipo is
port(clk:in std_logic;
Sin:in std_logic_vector(3 downto 0);
Sout: inout std_logic_vector(3 downto 0));
end pipo;

architecture structural of pipo is

component D_FlipFlop is
port( din: in std_logic;
clk: in std_logic;
q: out std_logic);
end component;

```

```

begin
ff1:D_FlipFlop port map(Sin(3),clk,Sout(3));
ff2:D_FlipFlop port map(Sin(2),clk,Sout(2));
ff3:D_FlipFlop port map(Sin(1),clk,Sout(1));
ff4:D_FlipFlop port map(Sin(0),clk,Sout(0));
end structural;

```

PISO STRUCTURAL

```

library ieee;
use ieee.std_logic_1164.all;

entity D_FlipFlop is
port( din: in std_logic;
clk: in std_logic;
q: out std_logic);
end D_FlipFlop;

architecture bin of D_FlipFlop is
begin
process(clk,din)
begin
if(clk='1' and clk'EVENT) then
q <= din;
end if;

```

```
end process;  
end bin;
```

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

```
entity and_gate is  
port (a:in std_logic;  
b:in std_logic;  
c:out std_logic);  
end and_gate;
```

```
architecture bin of and_gate is  
begin  
c<=a and b;  
end bin;
```

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

```
entity or_gate is  
port (a:in std_logic;  
b:in std_logic;  
c:out std_logic);  
end or_gate;
```

```
architecture bin of or_gate is  
begin  
c<=a or b;  
end bin;
```

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

```
entity not_gate is  
port (a:in std_logic;  
b:out std_logic);
```

```
end not_gate;

architecture bin of not_gate is
begin
b<=not a;
end bin;
```

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

entity piso is
port(clk:in std_logic;
shift:in std_logic;
Sin:in std_logic_vector(3 downto 0);
Sout: inout std_logic);
end piso;
```

architecture structural of piso is

```
component D_FlipFlop is
port( din: in std_logic;
clk: in std_logic;
q: out std_logic);
end component;
```

```
component and_gate is
port( a: in std_logic;
b: in std_logic;
c: out std_logic);
end component;
```

```
component or_gate is
port( a: in std_logic;
b: in std_logic;
c: out std_logic);
end component;
```

```
component not_gate is
port( a: in std_logic;
b: out std_logic);
end component;
```

```
signal out1,out2,out3,shiftnot,or1,or2,or3,and1,and2,and3,and4,and5,and6:std_logic;
begin
```

```

ff0:not_gate port map(shift,shiftnot);
ff1:D_FlipFlop port map(Sin(3),clk,out1);
ff2:and_gate port map(out1,shift,AND1);
ff3:and_gate port map(shiftnot,Sin(2),AND2);
ff4:or_gate port map(AND1,AND2,OR1);
ff5:D_FlipFlop port map(OR1,clk,out2);
ff6:and_gate port map(out2,shift,AND3);
ff7:and_gate port map(shiftnot,Sin(1),AND4);
ff8:or_gate port map(AND3,AND4,OR2);
ff9:D_FlipFlop port map(OR2,clk,out3);
ff10:and_gate port map(out3,shift,AND5);
ff11:and_gate port map(shiftnot,Sin(0),AND6);
ff12:or_gate port map(AND5,AND6,OR3);
ff13:D_FlipFlop port map(OR3,clk,Sout);
end structural;

```

PISO BEHAVIORIAL

```

library ieee;
use ieee.std_logic_1164.all;
entity piso is
port(PI: in std_logic_vector(3 downto 0);
CLK: in std_logic;
SHFT: in std_logic;
SO: out std_logic);
end piso;

```

architecture behav of piso is

```

begin
process (CLK,SHFT)
variable tmp: std_logic_vector(3 downto 0);
begin
if (CLK='1' and CLK'event) then
if (SHFT='1') then
tmp:= PI;
else
tmp(0):= tmp(1);
tmp(1):= tmp(2);
tmp(2):= tmp(3);
tmp(3):= '0';
end if;
end if;
SO<= tmp(0);
end process;
end behav;

```

Bidirectional Structural:

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

entity D_FlipFlop is
port( din: in std_logic;
clk: in std_logic;
q: out std_logic);
end D_FlipFlop;
```

```
architecture bin of D_FlipFlop is
begin
process(clk,din)
begin
if(clk='1' and clk'EVENT) then
q <= din;
end if;
end process;
end bin;
```

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

```
entity and_gate is
port (a:in std_logic;
b:in std_logic;
c:out std_logic);
end and_gate;
```

```
architecture bin of and_gate is
begin
c<=a and b;
end bin;
```

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

```
entity or_gate is
port (a:in std_logic;
b:in std_logic;
c:out std_logic);
end or_gate;
```

```
architecture bin of or_gate is
begin
c<=a or b;
end bin;
```

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

entity not_gate is
port (a:in std_logic;
b:out std_logic);
end not_gate;

architecture bin of not_gate is
begin
b<=not a;
end bin;
```

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

```
entity bidirectional is
port(clk:in std_logic;
shift:in std_logic;
SIn:in std_logic;
SoutL: inout std_logic;
SoutR: inout std_logic);
end bidirectional;
```

```
architecture structural of bidirectional is
```

```
component D_FlipFlop is
port( din: in std_logic;
clk: in std_logic;
q: out std_logic);
end component;
```

```
component and_gate is
port( a: in std_logic;
b: in std_logic;
c: out std_logic);
end component;
```

```
component or_gate is
port( a: in std_logic;
b: in std_logic;
c: out std_logic);
end component;
```

```
component not_gate is
port( a: in std_logic;
```

```

b: out std_logic);
end component;

signal
out1,out2,out3,shiftnot,or1,or2,or3,or4,and1,and2,and3,and4,and5,and6,and7,and8:std_logic;
begin
ff0:not_gate port map(shift,shiftnot);
ff1:and_gate port map(Sin,shift,and1);
ff2:and_gate port map(shiftnot,out2,and2);
ff3:or_gate port map(and1,and2,or1);
ff4:D_FlipFlop port map(or1,clk,out1);
SoutR<=out1;
ff5:and_gate port map(out1,shift,and3);
ff6:and_gate port map(shiftnot,out3,AND4);
ff7:or_gate port map(and3,AND4,or2);
ff8:D_FlipFlop port map(or2,clk,out2);
ff9:and_gate port map(out2,shift,AND5);
ff10:and_gate port map(shiftnot,SoutL,AND6);
ff11:or_gate port map(and5,AND6,or3);
ff12:D_FlipFlop port map(or3,clk,out3);
ff13:and_gate port map(out3,shift,AND7);
ff14:and_gate port map(shiftnot,Sin,AND8);
ff15:or_gate port map(and7,AND8,or4);
ff16:D_FlipFlop port map(or4,clk,SoutL);
end structural;

```

BIDIRECTIONAL BEHAVIORAL:

```

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity BID_shiftreg is
port(IP: in std_logic_vector(3 downto 0);
CLK: in std_logic;
SHFT: in std_logic;
SO: out std_logic);
end BID_shiftreg;

```

architecture Behavioral of BID_shiftreg is

```

function divide (a; b) return is
variable a1 : (a'length-1 downto 0):=a;

```

```

variable b1 : (b'length-1 downto 0):=b;
variable p1 : (b'length downto 0):= (others => '0');
variable i : integer:=0;

begin
for i in 0 to b'length-1 loop
p1(b'length-1 downto 1) := p1(b'length-2 downto 0);
p1(0) := a1(a'length-1);
a1(a'length-1 downto 1) := a1(a'length-2 downto 0);
p1 := p1-b1;
if(p1(b'length-1) ='1') then
a1(0) :='0';
p1 := p1+b1;
else
a1(0) :='1';
end if;
end loop;
return a1;

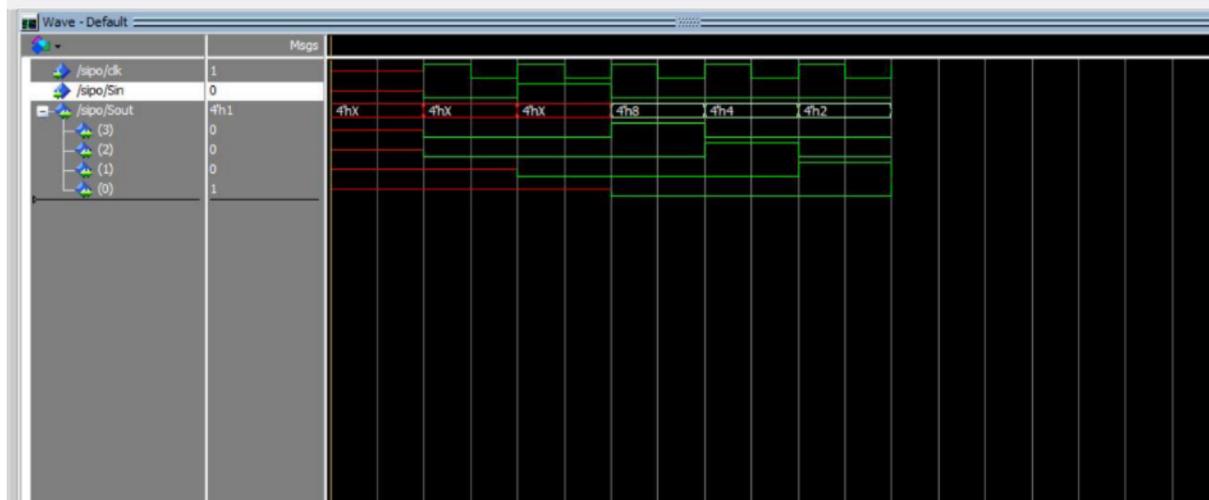
end divide;

begin
process (CLK,SHFT)
variable tmp: std_logic_vector(3 downto 0);
two:="10";
begin
if (CLK='1' and CLK'event) then
if (SHFT='1') then
tmp:=IP * "10";
else
tmp:=divide(IP,two);
end if;
end if;
SO<=tmp(0);
end process;
end Behavioral;

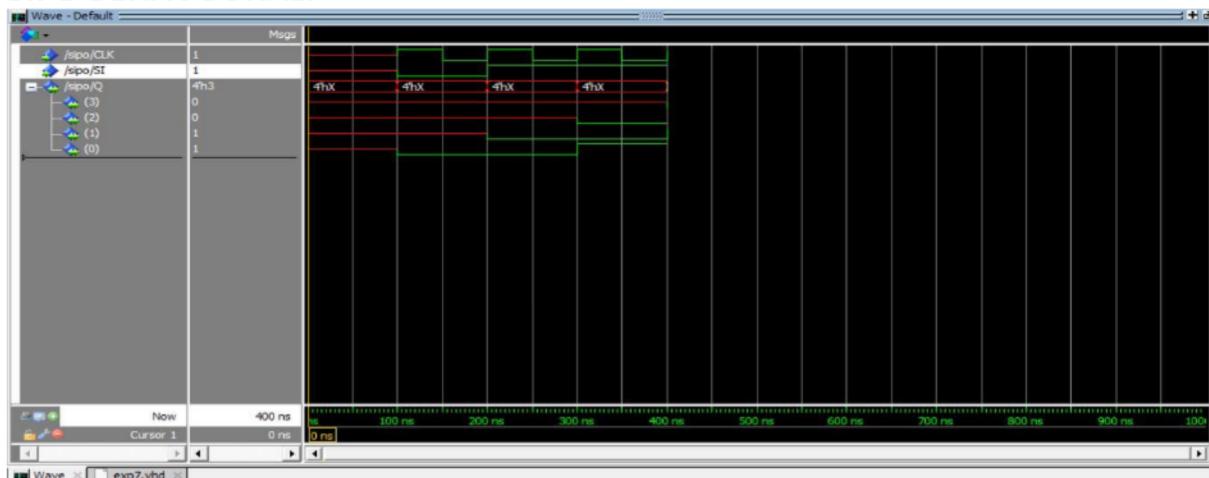
```

OUTPUT:

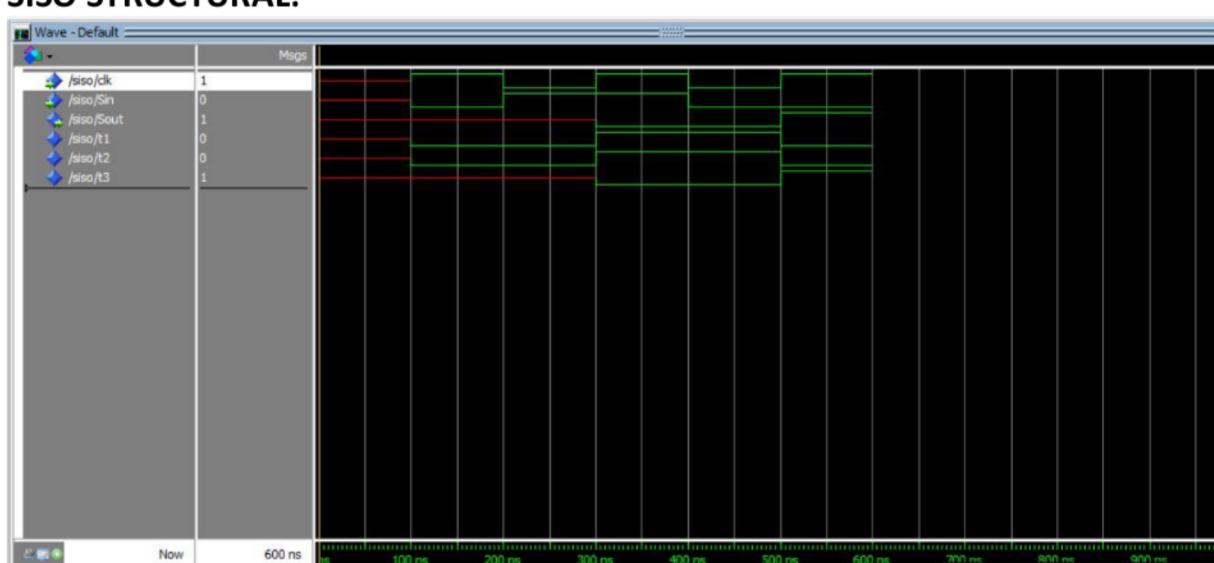
SIFO STRUCTURAL:



SIFO BEHAVOURAL:



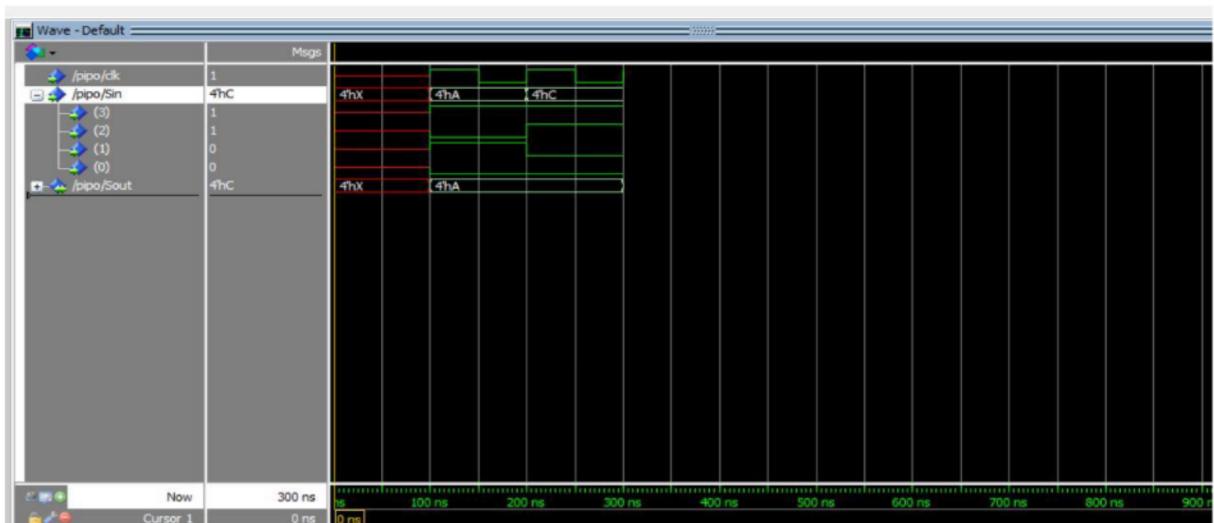
SISO STRUCTURAL:



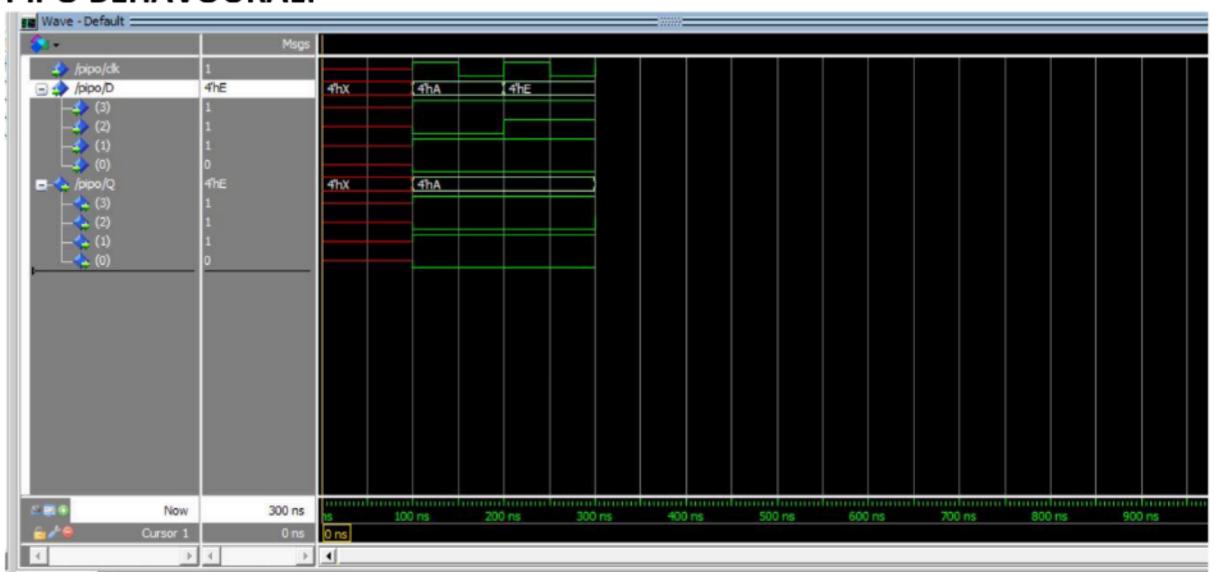
SISO STRUCTURAL:



PIPO STRUCTURAL:



PIPO BEHAVOURAL:



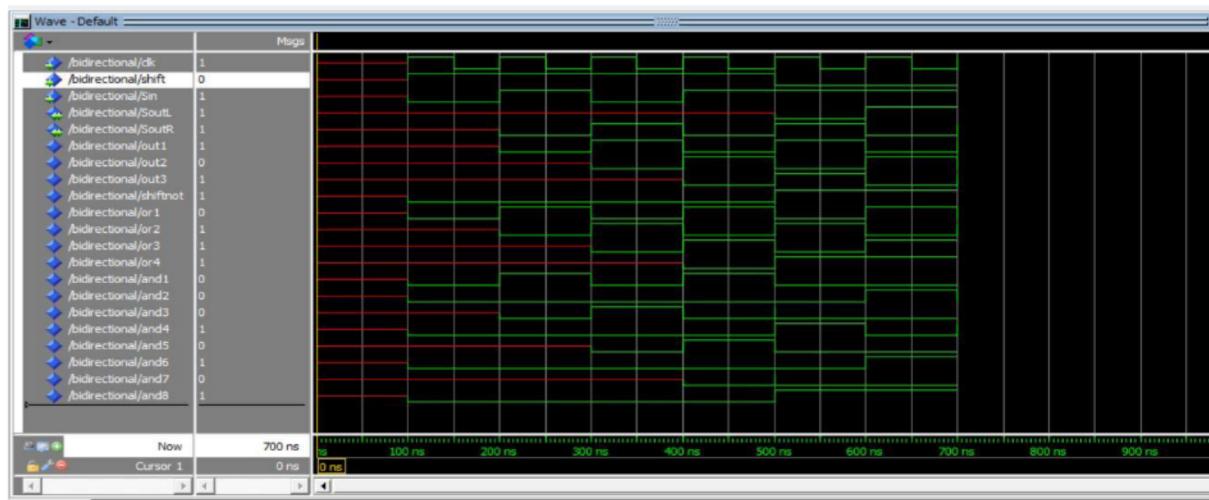
PISO STRUCTURAL:



PISO BEHAVIORAL



BIDIRECTIONAL STRUCTURAL



BIDIRECTIONAL BEHAVIORAL:

