COE 405, Term 062

Design & Modeling of Digital Systems

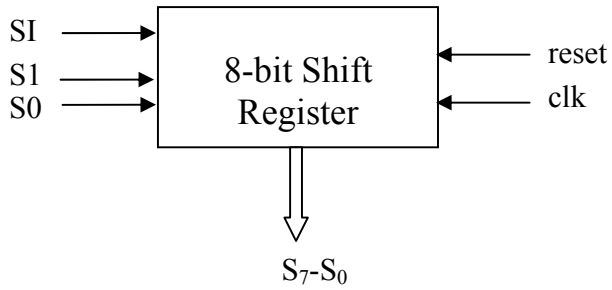
HW# 2 Solution

Due date: Monday, March 26, 2007

Q.1. It is required to design an **8-bit shift register**. The shift register should be able to shift or rotate either left or right based on the following table:

S1 S0	Operation
00	Shift left
01	Shift right
10	Rotate left
11	Rotate right

The reset is a synchronous reset and the shift register is rising-edge triggered. Assume that the shifted bit is based on the input SI. The interface description of the 8-bit shifter is shown below.



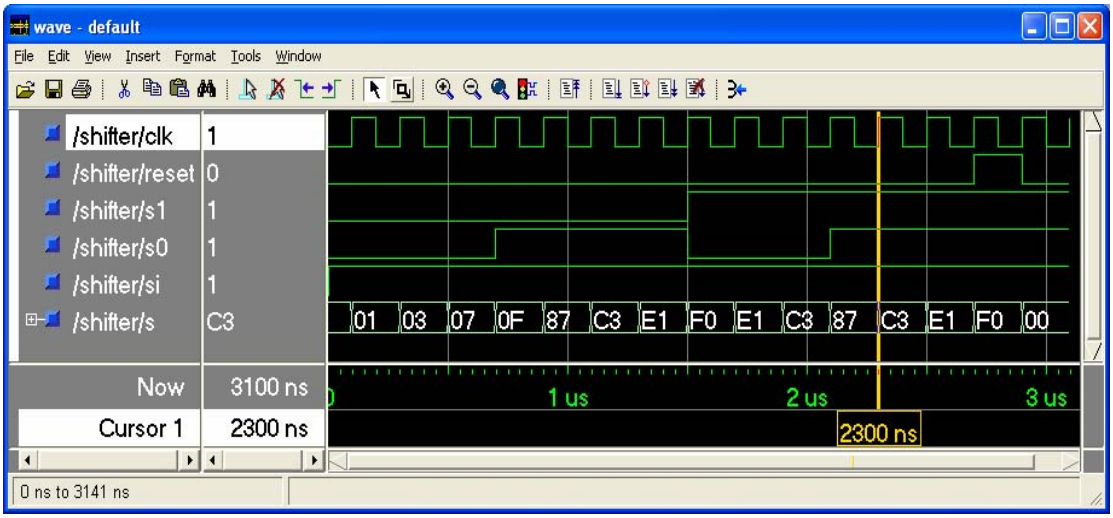
(i) Describe an Entity **Shifter8** for the 8-bit shift register.

```
Entity Shifter is
Generic (N: Integer :=8);
Port (clk, reset, S1, S0, SI : IN Bit;
   S : Buffer Bit_Vector(N-1 Downto 0));
End;
```

(ii) Model a behavioral Architecture **Behave** for this 8-bit shift register. Verify its correctness by simulation.

```
Architecture behave of shifter is
begin
   process(clk)
```

```
    begin
        if (clk = '1' and clk'event) then
            if(reset='1') then s <= (N-1 Downto 0 =>'0');

            elsif(s0='0' and s1='0')  then
                s <=s(N-2 downto 0)&si;
            elsif(s0='1' and s1='0')   then
                s <=si&s(N-1 downto 1);
            elsif(s0='0' and s1='1')  then
                s <=s(N-2 downto 0 )&s(N-1);
            elsif(s0='1' and s1='1')  then
                s <=s(0)&s(N-1 downto 1);
            end if;
        end if;
    end process;
end behave;
```



(iii) Model a structural Architecture **Struct** for this 8-bit shift register. Make your model generic using GENERATE statement to instantiate the required components such that your design can be extended into a shift register of any size by changing just a size parameter. Use configuration statement to configure the bounding of used components to entities. Verify the correctness of your model by simulation.

```
Entity DFF is
PORT(CLK, RESET, D: IN Bit; Q: OUT BIT);
END;
Architecture BV of DFF is
Begin
    Process(CLK)
    Begin
        If (CLK='1' and CLK'Event) Then
            If (Reset='1')Then
```

```
                    Q <= '0';
                Else
                    Q <= D;
                End if;
            End if;
    End Process;
End;
```

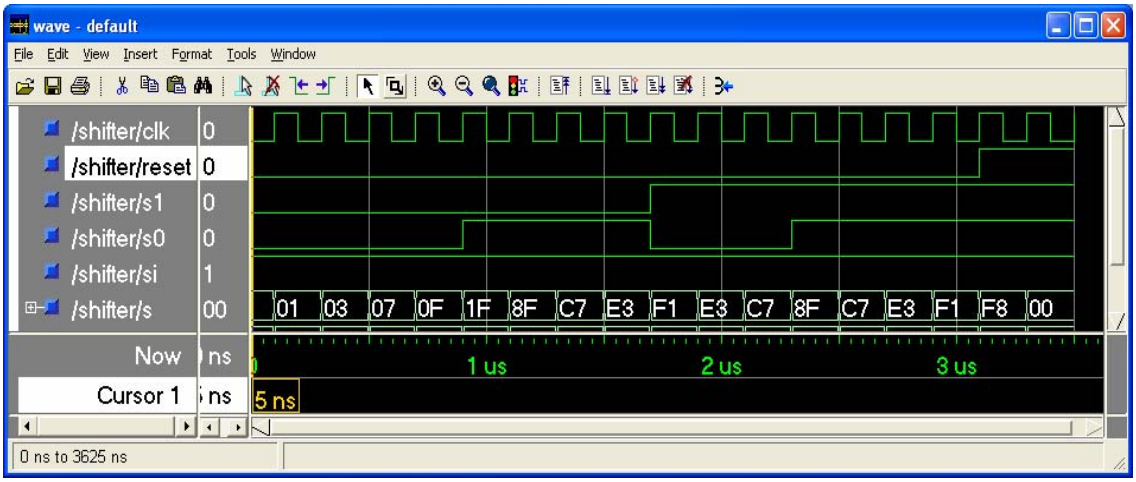
Entity Mux4x1 is
Port (I0, I1, I2, I3, S1, S0: IN Bit; O: OUT BIT);
End;

```
Architecture BV of Mux4x1 is
Begin
    Process(I0, I1, I2, I3, S1, S0)
    Variable T: Bit_Vector (1 Downto 0);
    Begin
            T := S1 & S0;
            Case T is
                When "00" => O <= I0;
                When "01" => O <= I1;
                When "10" => O <= I2;
                When "11" => O <= I3;
            End Case;
    End Process;
End;
```

Architecture ST of Shifter is

```
Component DFF
PORT(CLK, RESET, D: IN Bit; Q: OUT BIT);
END Component;

Component Mux4x1
Port (I0, I1, I2, I3, S1, S0: IN Bit; O: OUT BIT);
End Component;

For ALL: DFF Use Entity Work.DFF(BV);
For ALL: Mux4x1 Use Entity Work.Mux4x1(BV);

Signal MO, SO: Bit_Vector(N-1 Downto 0);
Begin

    MX0: Mux4x1 PORT MAP (SI, SO(1), SO(N-1), SO(1), S1, S0, MO(0));
    DF0: DFF PORT MAP (CLK, RESET, MO(0) , SO(0) );
GS: For I IN 1 To N-2 Generate
            MXI: Mux4x1 PORT MAP (SO(I-1), SO(I+1), SO(I-1), SO(I+1), S1, S0,
            MO(I));
            DFI: DFF PORT MAP (CLK, RESET, MO(I) , SO(I) );
```

```
    End Generate;
    MXL: Mux4x1 PORT MAP (SO(N-2), SI, SO(N-2), SO(0), S1, S0, MO(N-1));
    DFL: DFF PORT MAP (CLK, RESET, MO(N-1) , SO(N-1) );
    S <= SO;
End;
```



Q.2. **(Problem 4.15)**

Given the following signal assignments, show all transactions placed on each signal. At each event, show transactions that are appended, overwritten, and expired. Show resulting waveforms on each signal.

```
Architecture dataflow of signals IS
    Type qit is ('0', '1', 'Z', 'X');
    Signal a, b, c: qit := '0';
Begin
    a <= NOT a after 10ns WHEN NOW <= 30 ns;
    b <= 'Z', a after 25 ns, '0' after 35 ns;
    c <= '1', a after 5 ns, b after 20 ns;

End dataflow;
```

Time	0	δ	2δ	5 ns	10 ns	10 + δ	15 ns
A	0	0	0	0	1	1	1
	('1' , 10 ns)	('1' , 10 ns)	('1' , 10 ns)	('1' , 5 ns)	('0' , 10 ns)	('0' , 10 ns)	('0' , 5 ns)
B	0	Z	Z	Z	Z	Z	Z
	('Z', δ) ('0' , 25 ns) ('0' , 35 ns)	('0' , 25 ns) ('0' , 35 ns)	('0' , 25 ns) ('0' , 35 ns)	('0' , 20 ns) ('0' , 30 ns)	~~('0' , 15 ns)~~ ~~('0' , 25 ns)~~ ('Z' , δ) ('1' , 25 ns) ('0' , 35 ns)	('1' , 25 ns) ('0' , 35 ns)	('1' , 20 ns) ('0' , 30 ns)
C	0	1	1	0	0	1	1
	('1' , δ) ('0' , 5 ns) ('0' , 20 ns)	~~('0' , 5 ns)~~ ~~('0' , 20 ns)~~ ('1' , δ) ('0' , 5 ns) ('Z' , 20 ns)	('0' , 5 ns) ('Z' , 20 ns)	('Z' , 15 ns)	~~('Z' , 10 ns)~~ ('1' , δ) ('1' , 5 ns) ('Z' , 20 ns)	('1' , 5 ns) ('Z' , 20 ns)	('Z' , 15 ns)

Time	20 ns	20 + δ	25 ns	30 ns	30 + δ	35 ns	40 ns
A	0	0	0	1	1	1	0
	('1' , 10 ns)	('1' , 10 ns)	('1' , 5 ns)	('0' , 10 ns)	('0' , 10 ns)	('0' , 5 ns)	-----
B	Z	Z	Z	Z	Z	Z	Z
	~~('1' , 15 ns)~~ ~~('0' , 25 ns)~~ ('Z', δ) ('0' , 25 ns) ('0' , 35 ns)	('0' , 25 ns) ('0' , 35 ns)	('0' , 20 ns) ('0' , 30 ns)	~~('0' , 15 ns)~~ ~~('0' , 25 ns)~~ ('Z', δ) ('1' , 25 ns) ('0' , 35 ns)	('1' , 25 ns) ('0' , 35 ns)	('1' , 20 ns) ('0' , 30 ns)	~~('1' , 15 ns)~~ ~~('0' , 25 ns)~~ ('Z', δ) ('0' , 25 ns) ('0' , 35 ns)
C	1	1	0	0	1	1	1
	~~('Z' , 10 ns)~~ ('1' , δ) ('0' , 5 ns) ('Z' , 20 ns)	('0' , 5 ns) ('Z' , 20 ns)	('Z' , 15 ns)	~~('Z' , 10 ns)~~ ('1' , δ) ('1' , 5 ns) ('Z' , 20 ns)	('1' , 5 ns) ('Z' , 20 ns)	('Z' , 15 ns)	~~('Z' , 10 ns)~~ ('1' , δ) ('0' , 5 ns) ('Z' , 20 ns)

Time	40 + δ	45 ns	60 ns	65 ns	65 + δ	70 ns	75 ns	85
A	0	0	0	0	0	0	0	0
	-----	-----	-----	-----	-----	-----	-----	-----
B	Z	Z	Z	0	0	0	0	0
	('0' , 25 ns) ('0' , 35 ns)	('0' , 20 ns) ('0' , 30 ns)	('0' , 5 ns) ('0' , 15 ns)	('0' , 10 ns)	('0' , 10 ns)	('0' , 5 ns)	-----	-----
C	1	0	Z	Z	1	0	0	0
	('0' , 5 ns) ('Z' , 20 ns)	('Z' , 15 ns)	-----	('1' , δ) ('0' , 5 ns) ('0' , 20 ns)	('0' , 5 ns) ('0' , 20 ns)	('0' , 15 ns)	('0' , 10 ns)	-----