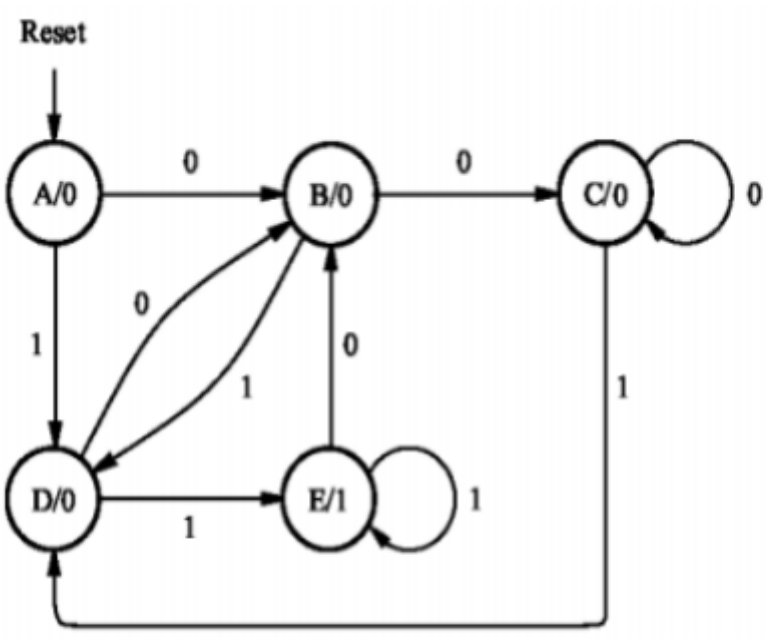
Tutorial 8

Name: Nour-aldin Ibrahim Ahmed Elbadawy
ID: 2018-13394

Q1)

Consider the state diagram of a sequential circuit is given as shown:



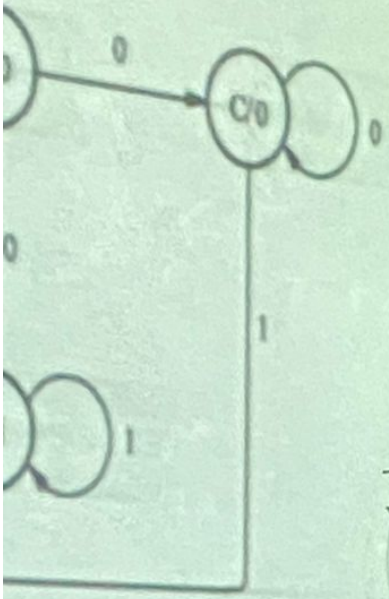
(a) Minimize the number of states with the aid of state table.

(b) Write the VHDL code to describe the minimized FSM.

(c) Implement the minimized FSM diagram using T- flip flops.

(d) Write a VHDL code to descripe the circuit implemented in part (c).

quential circuit is given as shown:



State	next state x=0	next state x=1	o/p
A	B	D	0
B	B	D	0
D	B	E	0
E	B	E	1

present State	next state x=0	next state x=1	o/p
A	B	D	0
B	C	D	0
C	B	D	0
D	B	E	0
E	B	E	1

f states with the aid of state table.

o describe the minimized FSM.

ed FSM diagram using T- flip flops.

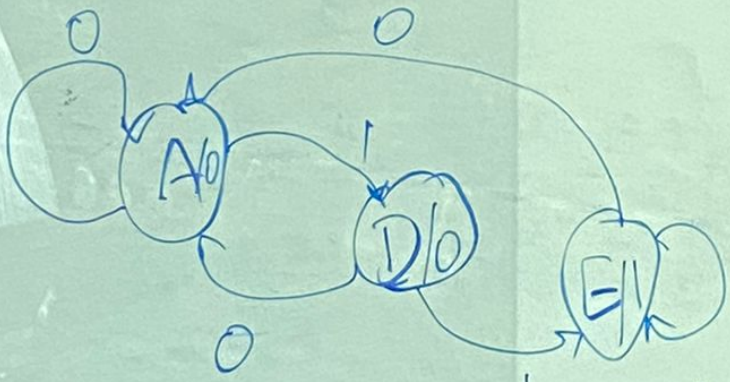
describe the circuit implemented in part (c).

Paste Copy Format Painter Clipboard

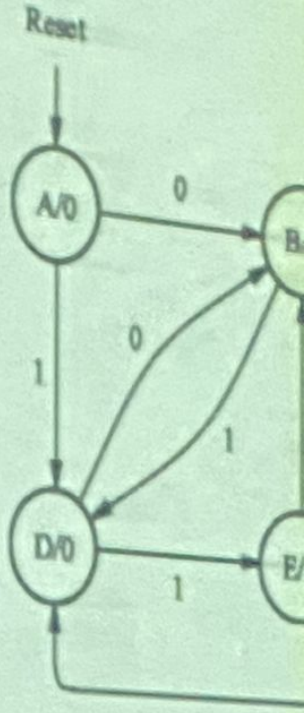
Times New Ro 18 Font

Question 1:

Consider the state diagram of a s



State	next state X=0	X=1	O/P
A	A	D	0
D	A	E	0
E	A	E	1



(a) Minimize the number

(b) Write the VHDL code

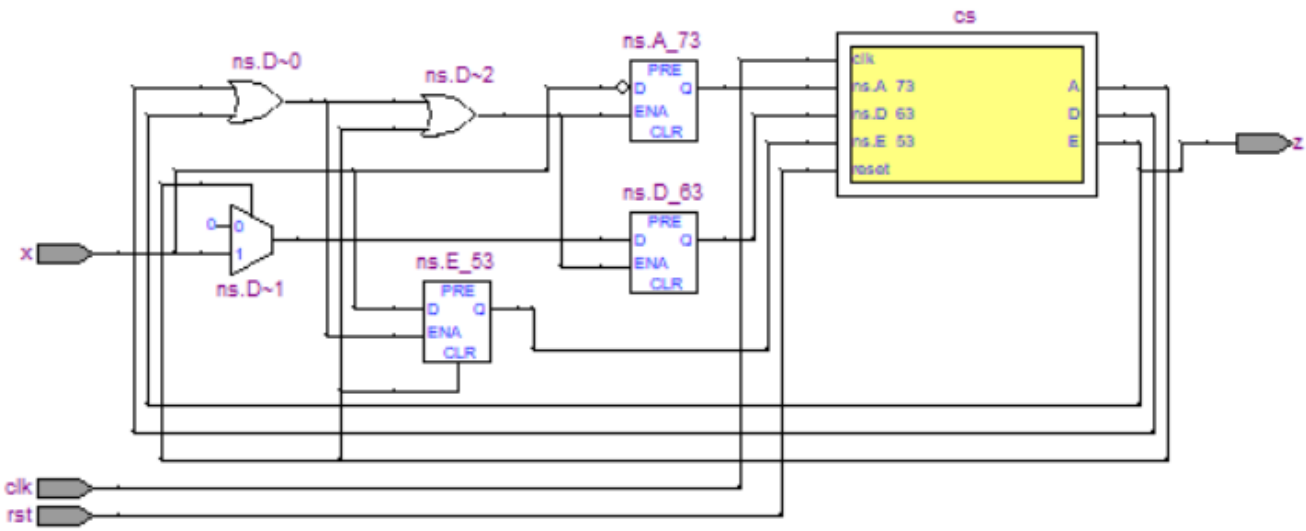
(c) Implement the minimi

(d) Write a VHDL code to

Page 1 of 8 47 words English (United Kingdom)

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

entity sheet5 is
        port(
                        x,clk,rst:in std_logic;
                        z: out std_logic
                );
end entity;



architecture behave of sheet5 is

        type states is (A,D,E);
        signal cs,ns : states;  --Current state, Next state

begin

        process (clk,rst)
        begin

                if rst='1' then cs <= A;

                elsif rising_edge (clk) then cs <= ns;

                end if;

        end process;



        process( clk, x )
        begin

                if cs = A then
                        if x='0' then ns <= A;

                        else ns <= D;

                        end if;

                elsif cs = D then
```

```
                        if x= '0' then ns <= A;

                        else ns <= E;

                        end if;

                elsif cs = E then
                        if x= '0' then ns <= A;

                        else ns <= E;

                        end if;

                end if;

        end process;

        z <= '1' when cs = E else '0';

end behave;
```



Current	Next (x=0)	Next (x=1)	Output
A	A	D	0
D	A	E	0
E	A	E	1

Current	I/P	Next	FF I/P	O/P
Q_1Q_2	X	$Q_1^+Q_2^+$	T_1T_2	Z
00	0	00	00	0
00	1	01	01	0
01	0	00	01	0
01	1	10	11	0
10	0	00	10	1
10	1	10	00	1

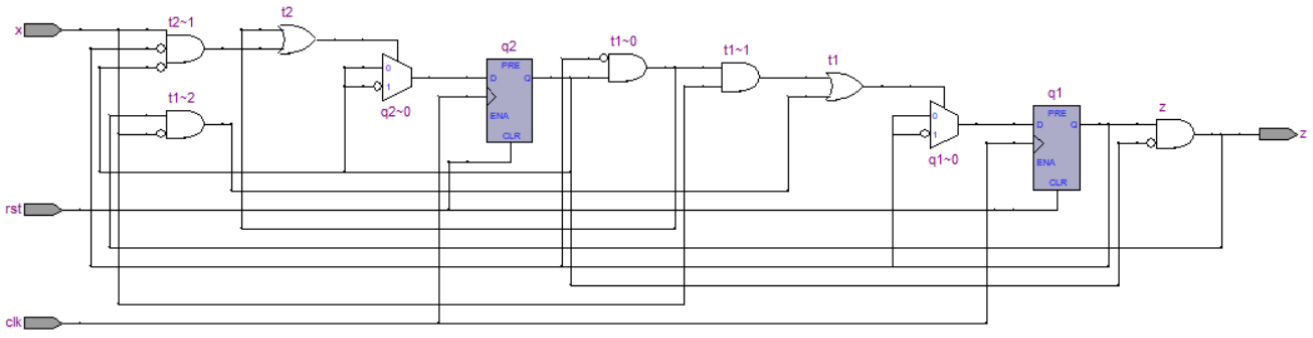
$$Z = Q_1\overline{Q_2}$$

$$T_1 = \overline{Q_1}Q_2X + Q_1\overline{Q_2}\,\overline{X}$$

$$T_2 = \overline{Q_1}\,\overline{Q_2}X + \overline{Q_1}Q_2$$

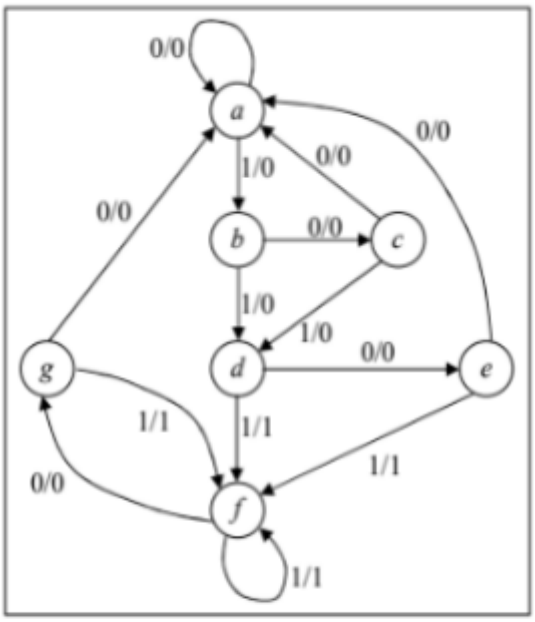
```
library ieee;
use ieee.std_logic_1164.all;

entity sheet5 is
        port(
                        x,clk,rst:in std_logic;
                        z: out std_logic
                );
```

```
end entity;

architecture behave of sheet5 is
        signal q1,q2,t1,t2: std_logic;
begin
        z <= q1 and not(q2);
        t1 <= (not(q1) and q2 and x) or (q1 and not(q2) and not(x));
        t2 <= (not(q1) and not(q2) and x) or (not(q1) and q2);

        process(rst,clk)
        begin
                if(rst='1') then
                        q1 <= '0';
                        q2 <= '0';
                elsif(rising_edge(clk)) then
                        if(t1='1') then
                                q1 <= not(q1);
                        end if;
                        if(t2='1') then
                                q2 <= not(q2);
                        end if;
                end if;
        end process;
end architecture;
```



Q2)

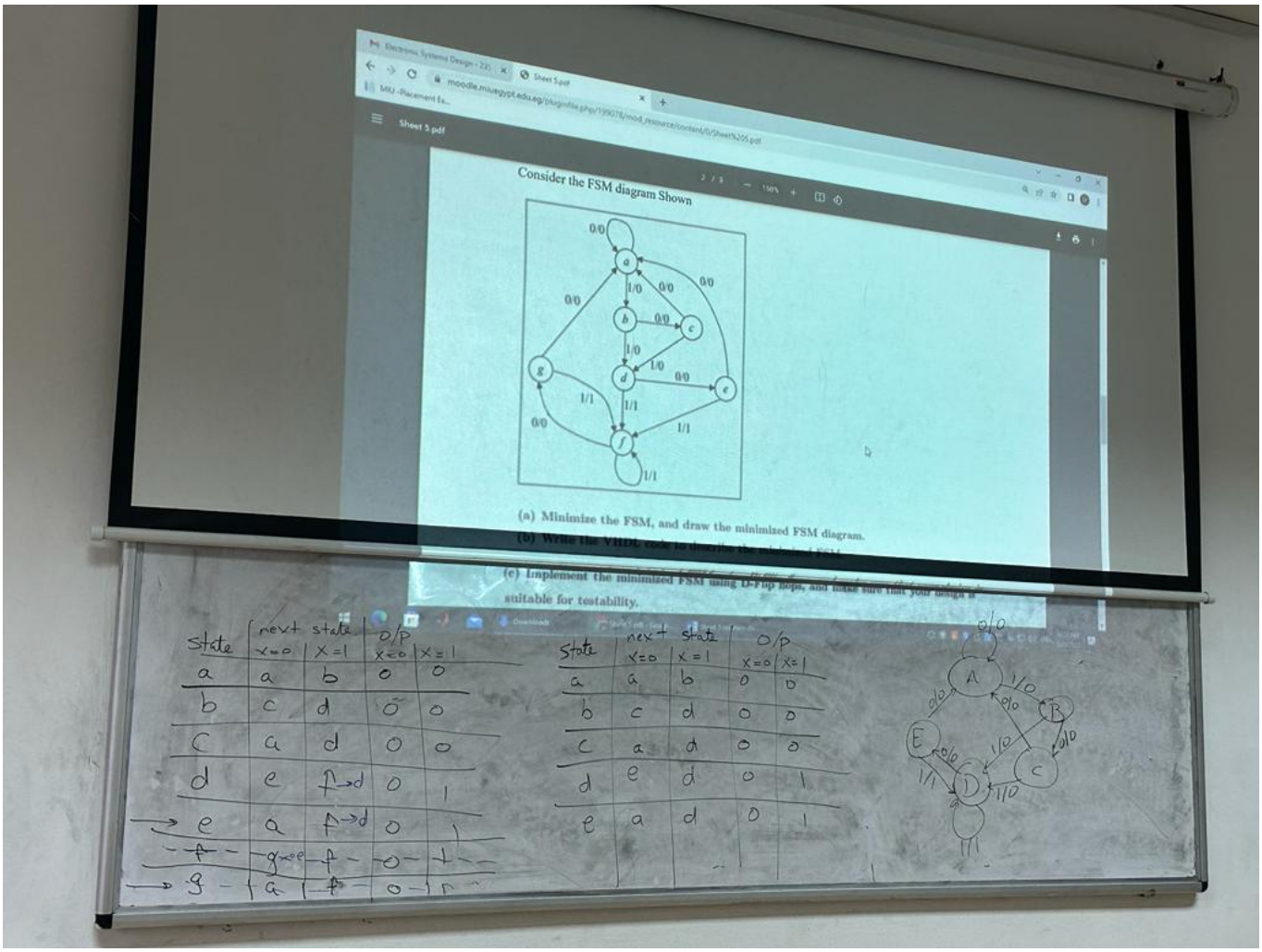
Consider the FSM diagram Shown



(a) Minimize the FSM, and draw the minimized FSM diagram.

(b) Write the VHDL code to describe the minimized FSM.

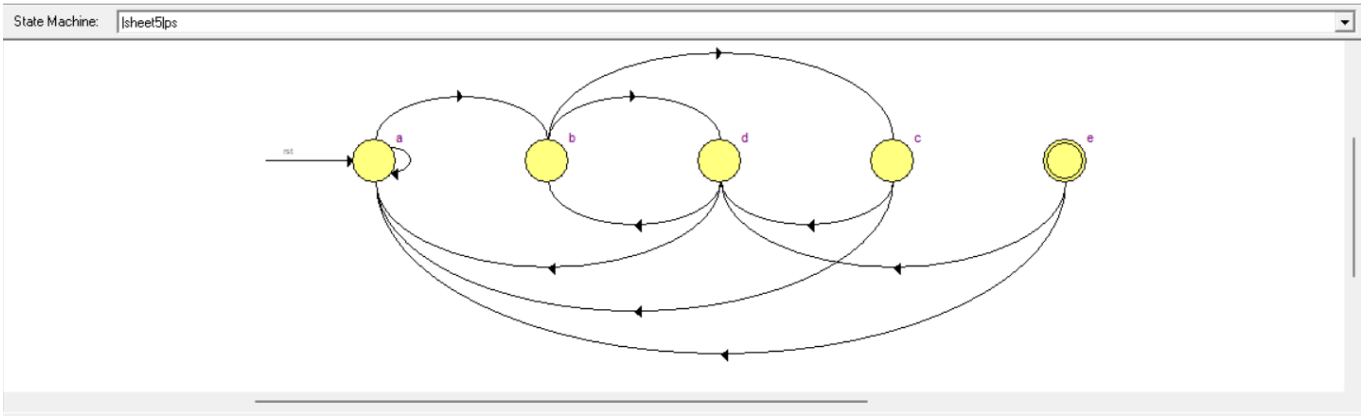
(c) Implement the minimized FSM using D-Flip flops, and make sure that your design is suitable for testability.




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

entity sheet5 is
	port(
		rst,clk,x1 : in std_logic;
		z : out std_logic
	);
end entity;

architecture behave of sheet5 is
	type states is (a,b,c,d,e);
	signal ps,ns: states;
begin
	process(rst,clk)
	begin
		if rst='1' then ps <= a;
		elsif rising_edge(clk) then ps <= ns;
```

```
        end if;
    end process;

    process(clk,x1,ps)
    begin
        case ps is
            when a =>
                if x1='0' then
                    ns <= a;
                    z <= '0';
                else ns <= b;
                    z <= '0';
                end if;
            when b =>
                if x1='0' then
                    ns <= c;
                    z <= '0';
                else ns <= d;
                    z <= '0';
                end if;
            when c =>
                if x1='0' then
                    ns <= a;
                    z <= '0';
                else ns <= d;
                    z <= '0';
                end if;
            when d =>
                if x1='0' then
                    ns <= a;
                    z <= '0';
                else ns <= b;
                    z <= '0';
                end if;
            when e =>
                if x1='0' then
                    ns <= a;
                    z <= '0';
                else ns <= d;
                    z <= '1';
                end if;
        end case;
```

```
        end process;
end behave;
```



	Source State	Destination State	Condition
1	a	a	(x1)
2	a	b	(x1)
3	b	c	(x1)
4	b	d	(x1)
5	c	a	(x1)
6	c	d	(x1)
7	d	a	(x1)
8	d	b	(x1)
9	e	a	(x1)
10	e	d	(x1)

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

entity sheet5 is
        port(
                rst, clk, x1: in std_logic;
                z: out std_logic
        );
end entity;

architecture behave of sheet5 is
        type states is (a, b, c, d, e);
        signal ps, ns: states;

        signal d_a, d_b, d_c, d_d, d_e: std_logic;
        signal q_a, q_b, q_c, q_d, q_e: std_logic;
begin
        -- D flip-flop process for state a
        d_a <= '0' when x1 = '0' else '1';
        process(clk, rst)
        begin
                if rst = '1' then
                        q_a <= '0';
                elsif rising_edge(clk) then
```

```
                q_a <= d_a;
        end if;
end process;

-- D flip-flop process for state b
d_b <= '0' when x1 = '0' else '0';
process(clk, rst)
begin
        if rst = '1' then
                q_b <= '0';
        elsif rising_edge(clk) then
                q_b <= d_b;
        end if;
end process;

-- D flip-flop process for state c
d_c <= '0' when x1 = '0' else '1';
process(clk, rst)
begin
        if rst = '1' then
                q_c <= '0';
        elsif rising_edge(clk) then
                q_c <= d_c;
        end if;
end process;

-- D flip-flop process for state d
d_d <= '0' when x1 = '0' else '0';
process(clk, rst)
begin
        if rst = '1' then
                q_d <= '0';
        elsif rising_edge(clk) then
                q_d <= d_d;
        end if;
end process;

-- D flip-flop process for state e
d_e <= '1' when x1 = '0' else '0';
process(clk, rst)
begin
        if rst = '1' then
                q_e <= '0';
```

```
        elsif rising_edge(clk) then
                q_e <= d_e;
        end if;
end process;

-- State transition process
process(ps, q_a, q_b, q_c, q_d, q_e)
begin
        case ps is
                when a =>
                        ns <= b;
                        z <= '0';
                when b =>
                        if q_b = '1' then
                                ns <= c;
                                z <= '0';
                        else
                                ns <= d;
                                z <= '0';
                        end if;
                when c =>
                        if q_c = '1' then
                                ns <= d;
                                z <= '0';
                        else
                                ns <= a;
                                z <= '0';
                        end if;
                when d =>
                        if q_d = '1' then
                                ns <= b;
                                z <= '0';
                        else
                                ns <= a;
                                z <= '0';
                        end if;
                when e =>
                        if q_e = '1' then
                                ns <= d;
                                z <= '1';
                        else
                                ns <= a;
                                z <= '0';
```

```
                                end if;
                        end case;
                end process;

                -- Assign present state to output of corresponding flip-flop
                ps <= a when q_a = '1' else
                        b when q_b = '1' else
                        c when q_c = '1' else
                        d when q_d = '1' else
                        e;

end behave;
```