ELEC 457

ASSIGNMENT-6

Sait Berk VARIMLI

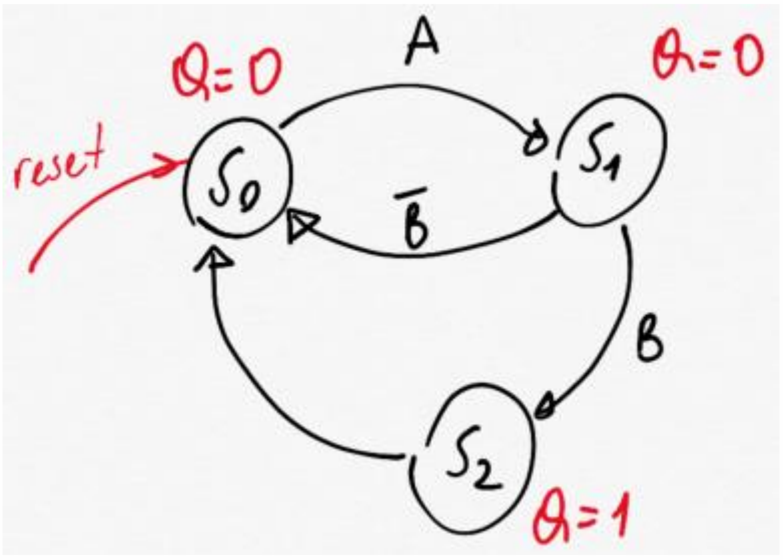
141024054

This assignment took 7.5 hours for finish.

(First question 30min. , second question 2 hours, third question 5 hours)

Problem 1

Design the state machine given below.



Code:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;entity state is

port(

        clk, rst : in std_logic;

        a : in std_logic_vector(1 downto 0); -- a[0] for A a[1] for B

        moore : out std_logic

    );

end state;



architecture Behavioral of state is

type state_type is (s0, s1, s2);

signal state, state_next : state_type := s0;

begin

    process(clk) is

    begin

        if rising_edge(clk) then
        if rst = '1' then

            state <= s0;

        else

            state <= state_next;

        end if;

    end if;

end process;

process(state, a) is

begin

    case state is

        when s0 =>

            if a = "10" then

                    state_next <= s1;

            elsif a = "10" then

                state_next <= s2;


            else

                state_next <= s0;

            end if;

        when s1 =>

            if a = "01" then

                state_next <= s2;

            elsif a = "10" then

                state_next <= s0;

            else

                state_next <= s1;

            end if;

        when s2 =>
```

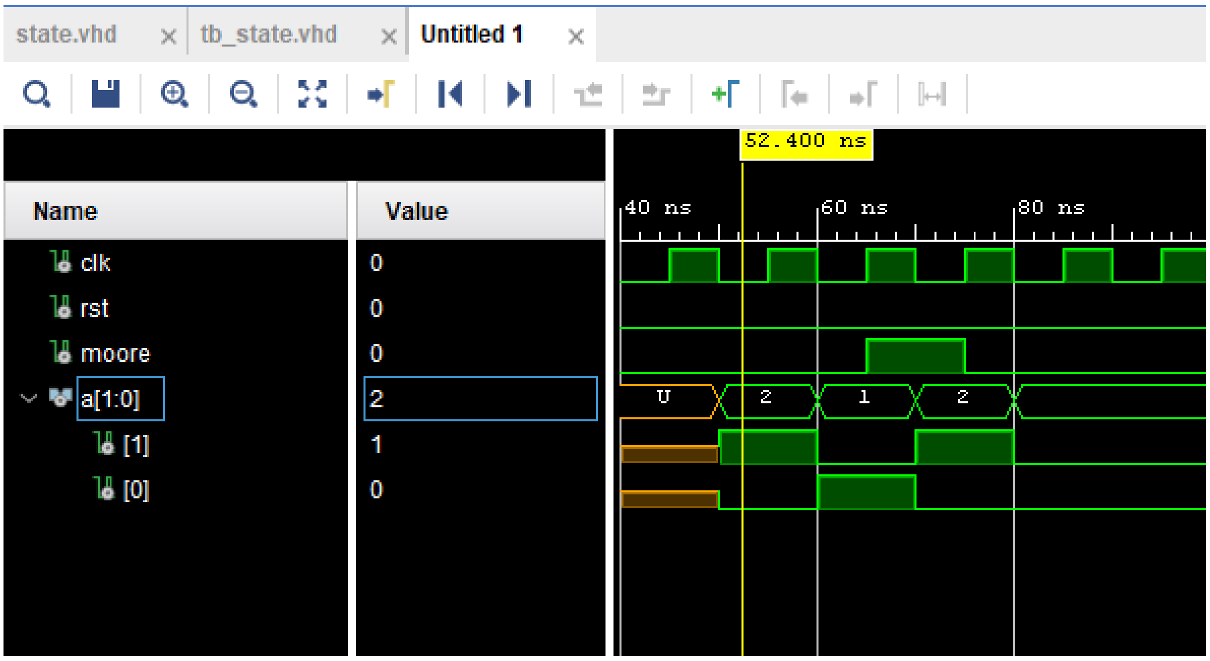
```
                    state_next <= s0;

        end case;
    end process;
    process(state) is
    begin
        case state is
            when s0 | s1 =>
                moore <= '0';
            when s2 =>
                moore <= '1';
        end case;
    end process;
end Behavioral;
```

Testbench Code:

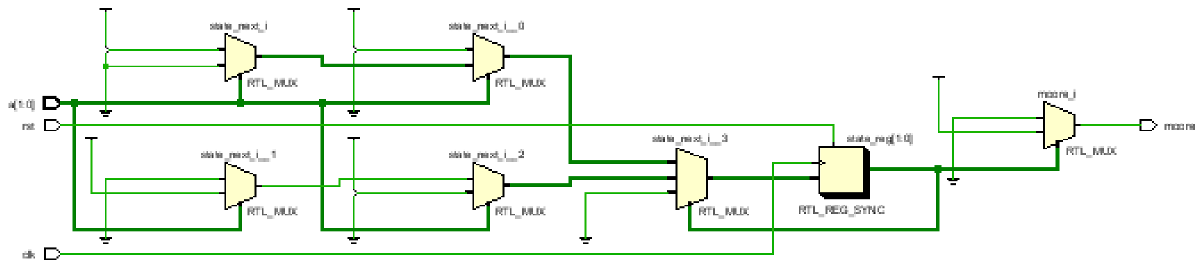
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_state is
end tb_state;

architecture Behavioral of tb_state is
component state is
    port(
        clk, rst : in std_logic;
        a : in std_logic_vector(1 downto 0);
        moore : out std_logic
    );
end component;
```

```
signal clk,rst,moore : std_logic;

signal a : std_logic_vector(1 downto 0);

begin


h0 : state port map(

clk => clk, rst => rst, a=>a, moore=>moore);

clock_process :process

begin

rst <='0';

clk <='0';

wait for 5 ns;

clk <= '1';

wait for 5 ns;

end process;


process

begin

wait for 50 ns;

a <= "10";

wait for 10 ns;

a <= "01";

wait for 10 ns;

a <= "10";

wait for 10 ns;

a <= "00";

wait;

end process;

end Behavioral;
```

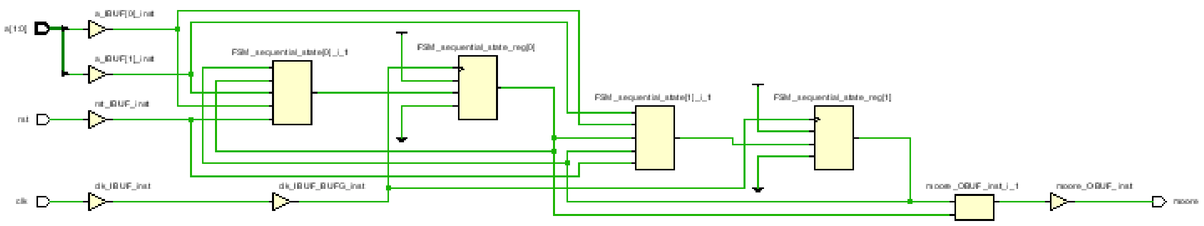
Simulation:



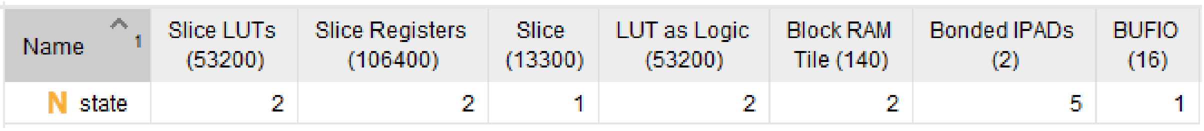
Elaborated Schematic:



Implementation Schematic:



Utizilation Report:



Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	Block RAM Tile (140)	Bonded IPADs (2)	BUFIO (16)
N state	2	2	1	2	2	5	1

Problem 2

Design a circuit that outputs a given sequence of bits.

- Send LSB first.
- Sequence should start with 4 `1`s and ends wth 4 `0`s.
- Make the sequence 8-bits long.
- The default output should stay at `0`.
- Example: For a sequence of `10111011`, the output should look like `1111110111010000`.
- When the transmission ends, generate a transmission done signal.
- The data is passed into the circuit using an 8-bit port.

An example entity is given below:

```
entity sequencer is
generic (
    D : integer := 8 -- data bits
);
port (
    clk, rst : in std_logic;
    i_start : in std_logic; -- start transmission
    i_data : in std_logic_vector(D-1 downto 0); -- send this data
    o_tx : out std_logic; -- transmission out
    o_tx_done : out std_logic -- transmission done
);
end sequencer;
```

Code:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity sequencer is

generic (

    D : integer := 8 -- data bits

);

port (

    clk, rst : in std_logic;

    i_start : in std_logic; -- start transmission

    i_data : in std_logic_vector(D-1 downto 0); -- send this data

    o_data : out std_logic_vector(D+7 downto 0) := "0000000000001111";

    o_tx : out std_logic; -- transmission out

    o_tx_done : out std_logic -- transmission done

);

end sequencer;


architecture Behavioral of sequencer is
```

```
signal t_done : std_logic;

signal i_data_reverse : std_logic_vector(D-1 downto 0);

type state_type is (s0, s1, s2);

signal state, state_next : state_type := s0;


begin


 process(clk) is

    begin

        if rising_edge(clk) then

            if rst = '1' then

                state <= s0;

            else

                state <= state_next;

            end if;

        end if;

    end process;


    process(state, i_data) is

    begin

        case state is

            when s0 =>

                if i_start <= '1' then

                state_next <= s1;

                end if;

            when s1 =>

                i_data_reverse(7) <= i_data(0);

                i_data_reverse(6) <= i_data(1);

                i_data_reverse(5) <= i_data(2);

                i_data_reverse(4) <= i_data(3);

                i_data_reverse(3) <= i_data(4);

                i_data_reverse(2) <= i_data(5);

                i_data_reverse(1) <= i_data(6);

                i_data_reverse(0) <= i_data(7);


                o_data(11 downto 4) <= i_data_reverse;

                state_next <= s2;

            when s2 =>
```

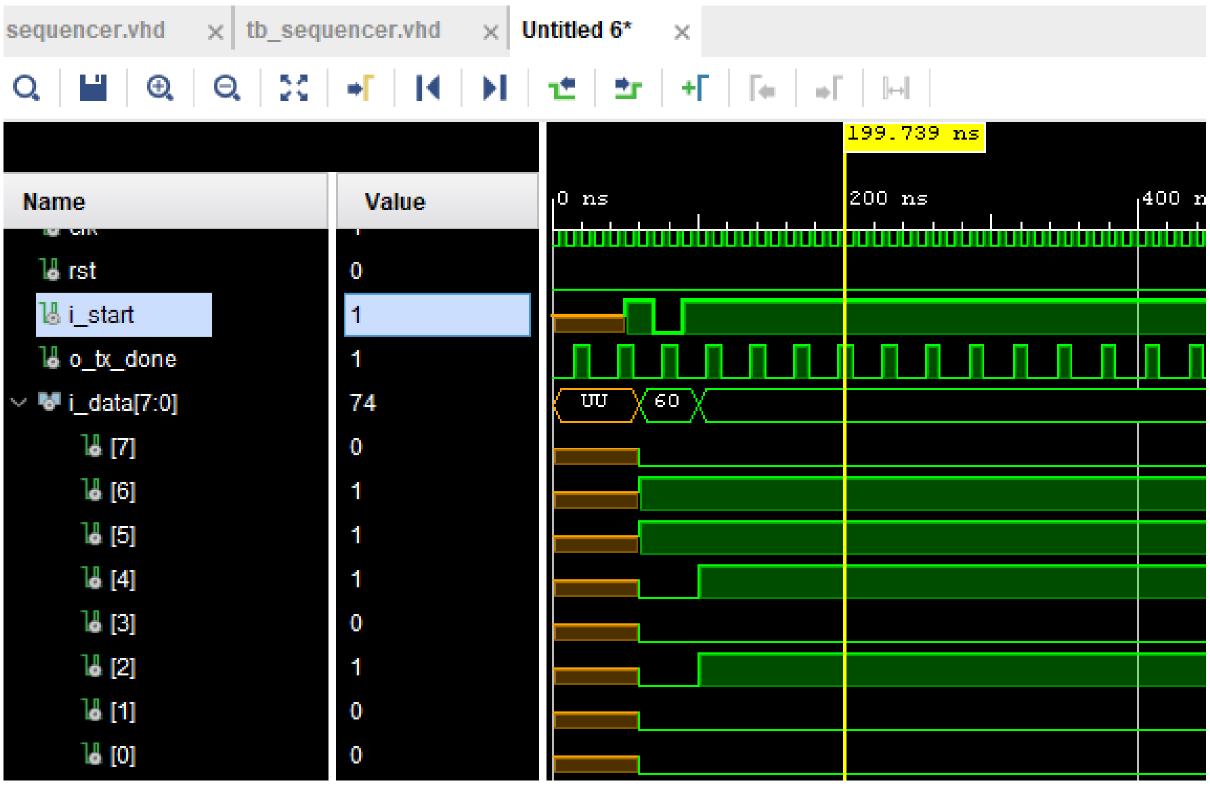
```
                    state_next <= s0;


        end case;
    end process;


process(state) is
    begin
        case state is
            when s0 | s1 =>
                o_tx_done <= '0';
            when s2 =>
                o_tx_done <= '1';
        end case;
    end process;



end Behavioral;
```

Testbench Code:

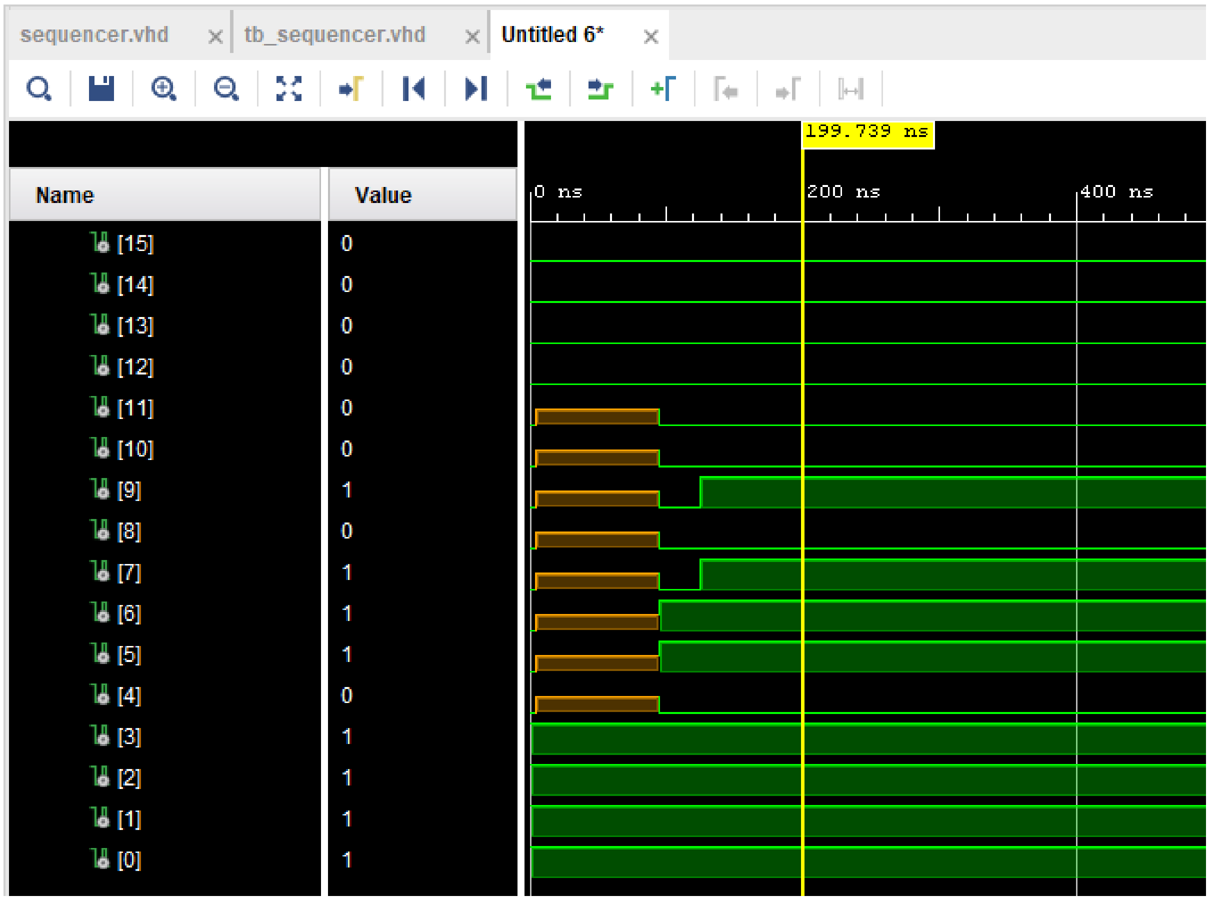
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity tb_sequencer is
end tb_sequencer;
architecture Behavioral of tb_sequencer is
component sequencer is
generic (
    D : integer := 8 -- data bits
);
port (
    clk, rst : in std_logic;
    i_start : in std_logic; -- start transmission
    i_data : in std_logic_vector(D-1 downto 0); -- send this data
    o_data : out std_logic_vector(D+7 downto 0) := "0000000000001111";
    o_tx : out std_logic; -- transmission out
    o_tx_done : out std_logic -- transmission done
);
end component;
```

```
signal clk,rst,i_start,o_tx_done : std_logic;

signal i_data : std_logic_vector(7 downto 0);

signal o_data : std_logic_vector(15 downto 0);


begin


h0: sequencer port map(

clk=>clk, rst=>rst, i_start=>i_start, o_tx_done=>o_tx_done, i_data=>i_data, o_data=>o_data);

clock_process :process

begin

rst <='0';

clk <='0';

wait for 5 ns;

clk <= '1';

wait for 5 ns;

end process;


process

begin

wait for 50 ns;

i_start <= '1';

wait for 10 ns;

i_data <= "01100000";

wait for 10 ns;

i_start <= '0';

wait for 10 ns;

i_data <= "01100000";

wait for 10 ns;

i_start <= '1';

wait for 10 ns;

i_data <= "01110100";

wait;


end process;

end Behavioral;
```

Simulation:



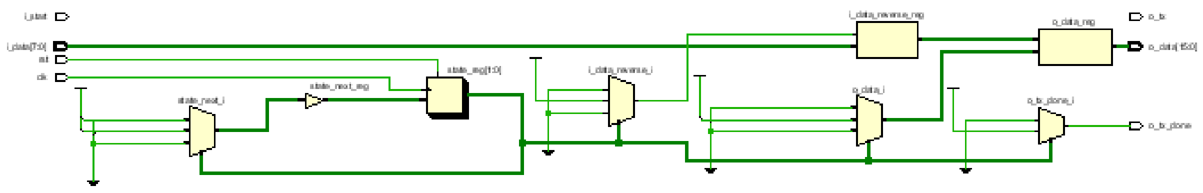
Name	Value
rst	0
i_start	1
o_tx_done	1
i_data[7:0]	74
[7]	0
[6]	1
[5]	1
[4]	1
[3]	0
[2]	1
[1]	0
[0]	0



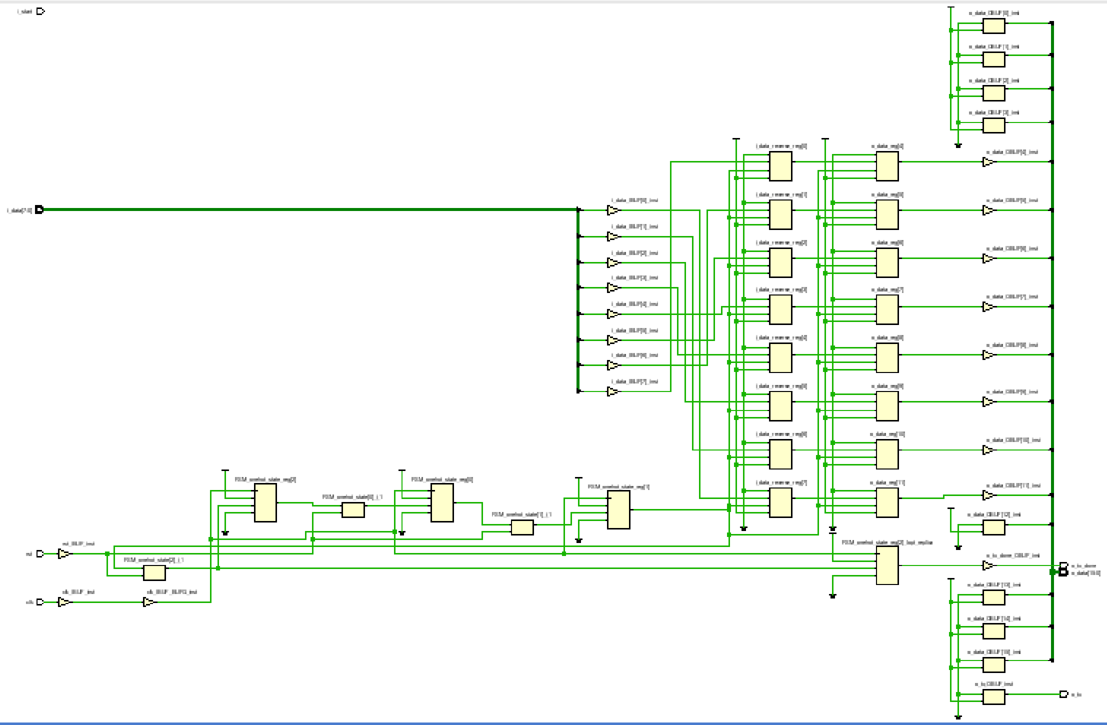
Name	Value
[15]	0
[14]	0
[13]	0
[12]	0
[11]	0
[10]	0
[9]	1
[8]	0
[7]	1
[6]	1
[5]	1
[4]	0
[3]	1
[2]	1
[1]	1
[0]	1

Input: 01110100 , Output: 0000001011101111

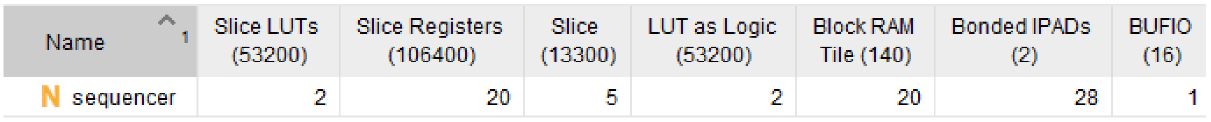
Elaborated Schematic:



Implementation Schematic:



Utizilation Report:



Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	Block RAM Tile (140)	Bonded IPADs (2)	BUFIO (16)
N sequencer	2	20	5	2	20	28	1

Problem 3

Based on the circuit that you designed in *sequencer* problem, implement a multi-cycle clone. Meaning that, each bit that you send should stay for a given number of clock cycles denoted below using the `M` generic. Use the timer circuit that you built before for this purpose.

```
entity serializer is
generic (
    D : integer := 8; -- data bits
    M : integer := 10 -- how many cycles the signals should be transmitted
);
port (
    clk, rst : in std_logic;
    i_start : in std_logic;
    i_data : in std_logic_vector(D-1 downto 0);
    o_tx : out std_logic;
    o_tx_done : out std_logic
);
end serializer;
```

Code:

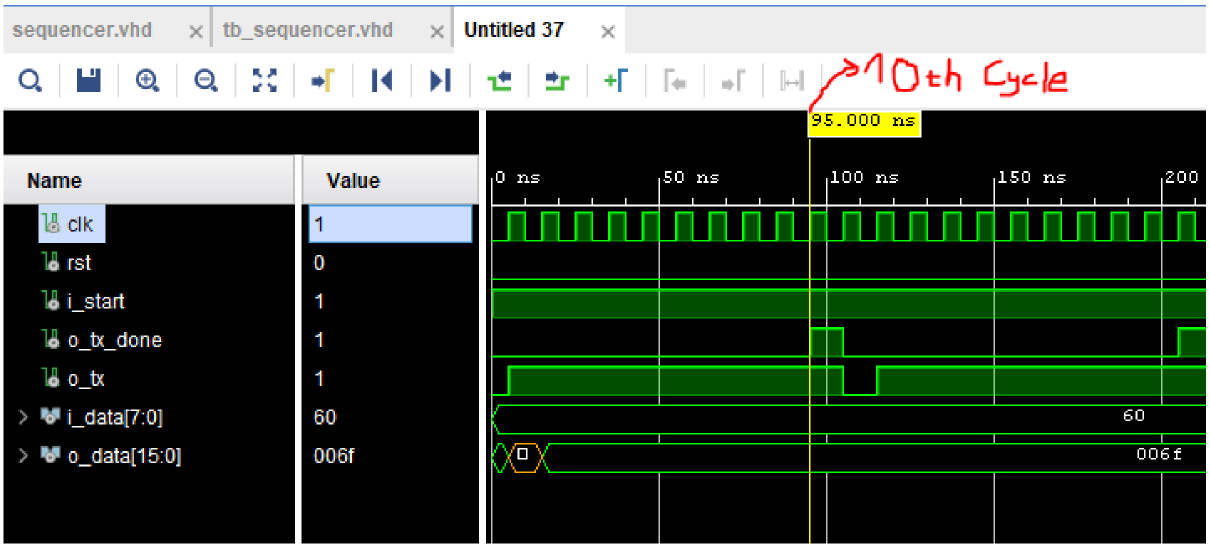
```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity sequencer is

generic (

    D : integer := 8;-- data bits

    M : integer := 10

);

port (

    clk, rst : in std_logic;

    i_start : in std_logic := '0'; -- start transmission

    i_data : in std_logic_vector(D-1 downto 0); -- send this data

    o_data : out std_logic_vector(D+7 downto 0) := "0000000000000000";

    o_tx : out std_logic; -- transmission out

    o_tx_done : out std_logic -- transmission done

);

end sequencer;


architecture Behavioral of sequencer is

signal t_done : std_logic;

signal i_data_reverse : std_logic_vector(D-1 downto 0);

type state_type is (s0, s1, s2);

signal state, state_next : state_type := s0;
signal cnt, cnt_next : integer range 0 to M := 0;

signal trgt, trgt_next : integer range 0 to M := 0;


begin


 process(clk) is

    begin

        if rising_edge(clk) then

            if rst = '1' then

                state <= s0;

            else

                state <= state_next;

                cnt <= cnt_next;

                trgt <= trgt_next;

            end if;

        end if;

    end process;


    process(state, i_data,cnt,trgt) is

    begin

    state_next <= state;

    cnt_next <= cnt;

    trgt_next <= trgt;

        case state is

            when s0 =>

                if i_start <= '1' then

                cnt_next <= 0;

                state_next <= s1;

                else

                state_next <= s0;

                end if;

            when s1 =>

                i_data_reverse(7) <= i_data(0);

                i_data_reverse(6) <= i_data(1);

                i_data_reverse(5) <= i_data(2);

                i_data_reverse(4) <= i_data(3);

               i_data_reverse(3) <= i_data(4);

                i_data_reverse(2) <= i_data(5);

                i_data_reverse(1) <= i_data(6);
                i_data_reverse(0) <= i_data(7);
                o_data(11 downto 4) <= i_data_reverse;
                o_data(15 downto 12) <= "0000";
                o_data(3 downto 0) <= "1111";
                if cnt = M-2 then
                state_next <= s2;
                else
                cnt_next <= cnt +1;
                end if;
            when s2 =>
            state_next <= s0;

        end case;
    end process;

process(state) is
    begin
        case state is
            when s0 =>
                o_tx_done <= '0';
                o_tx <= '0';
            when s1 =>
                o_tx_done <= '0';
                o_tx <= '1';
            when s2 =>
                o_tx_done <= '1';
                o_tx <= '1';
        end case;
    end process;



end Behavioral;
```

Testbench Code:

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

entity tb_sequencer is

end tb_sequencer;

architecture Behavioral of tb_sequencer is

component sequencer is

generic (

    D : integer := 8 -- data bits

);

port (

    clk, rst : in std_logic;

    i_start : in std_logic; -- start transmission

    i_data : in std_logic_vector(D-1 downto 0); -- send this data

    o_data : out std_logic_vector(D+7 downto 0) := "0000000000001111";

    o_tx : out std_logic; -- transmission out

    o_tx_done : out std_logic -- transmission done

);

end component;


signal clk,rst,i_start,o_tx_done,o_tx : std_logic;

signal i_data : std_logic_vector(7 downto 0);

signal o_data : std_logic_vector(15 downto 0);


begin


h0: sequencer port map(

o_tx => o_tx, clk=>clk, rst=>rst, i_start=>i_start, o_tx_done=>o_tx_done, i_data=>i_data,
o_data=>o_data);

clock_process :process

begin

rst <='0';

clk <='0';

wait for 5 ns;

clk <= '1';

wait for 5 ns;

end process;


process
```

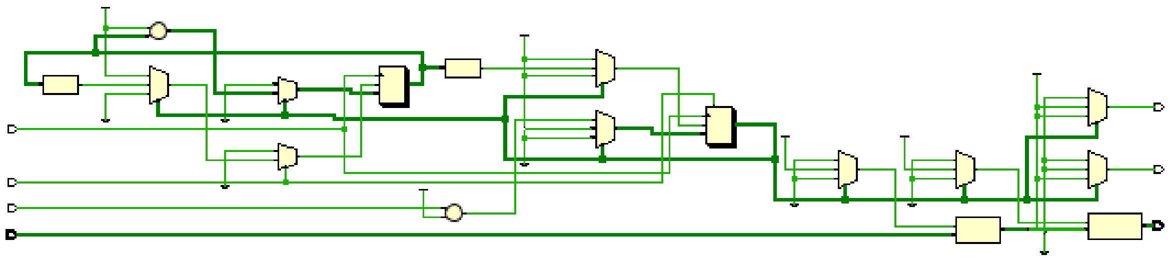
```
begin
i_start <= '1';
i_data <= "01100000";
wait;


end process;
end Behavioral;
```

Simulation:

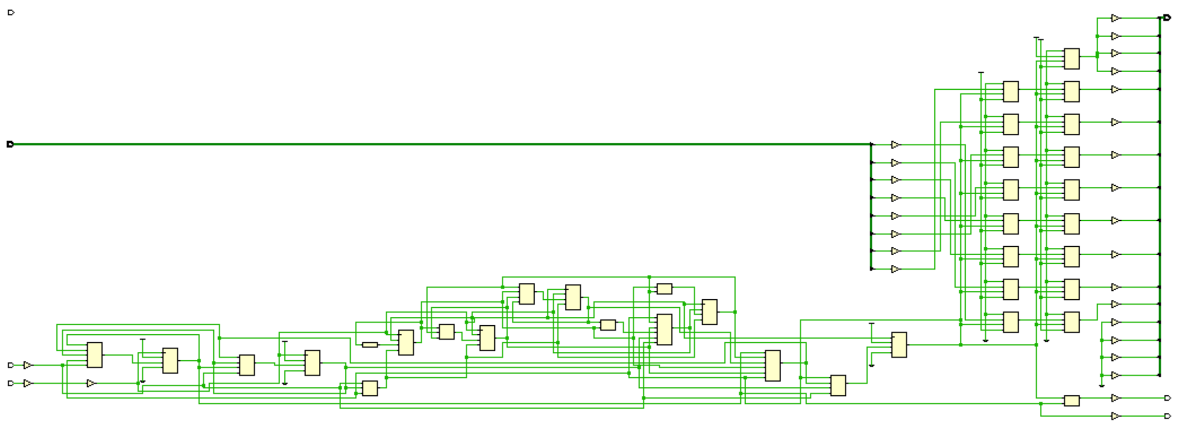


Name	Value
clk	1
rst	0
i_start	1
o_tx_done	1
o_tx	1
i_data[7:0]	60
o_data[15:0]	006f

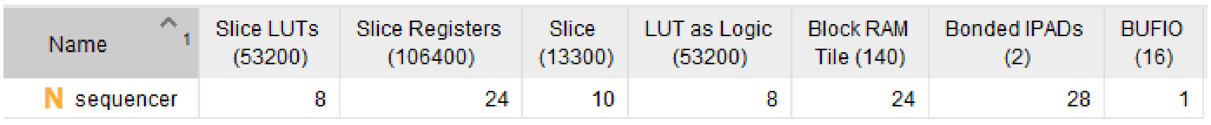
Elaborated Schematic:



Implementation Schematic:



Report Utilization



Name	Slice LUTs (53200)	Slice Registers (106400)	Slice (13300)	LUT as Logic (53200)	Block RAM Tile (140)	Bonded IPADs (2)	BUFIO (16)
sequencer	8	24	10	8	24	28	1