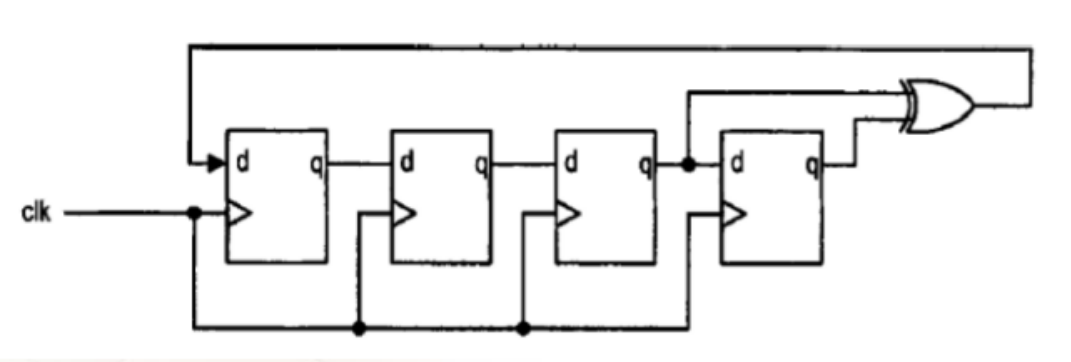
# CEG3155: DGD 8

**Question 1**

Let the propagation delay of an xor cell be 4 ns, and the setup time and clock-to-q delay of the register be 2 and 3 ns respectively. Determine the maximum clock frequency of a 4-bit LFSR.

***Solution***




Critical path delay:

$$t_c = 4 + 2 + 3 = 9\,ns$$

Maximum allowable frequency

$$f_c = \frac{1}{t_c} = 111.11\,MHz$$

**Question 2 (Chu 8.11)**

Consider the block diagram of the decade counter in Figure 8.13. Let $T_{cq}$ and $T_{setup}$ of the DFF be 1 and 0.5 ns, and the propagation delays of the incrementor, comparator and multiplexer be 5, 3 and 0.75 ns respectively. Assume that no further optimization will be performed during synthesis. Determine the maximal clock rate.

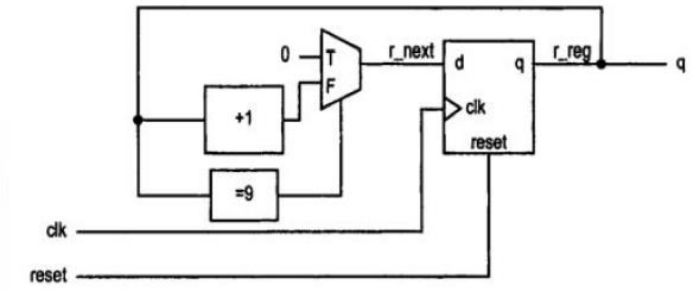



**Figure 8.13** Conceptual diagram of a decade counter.

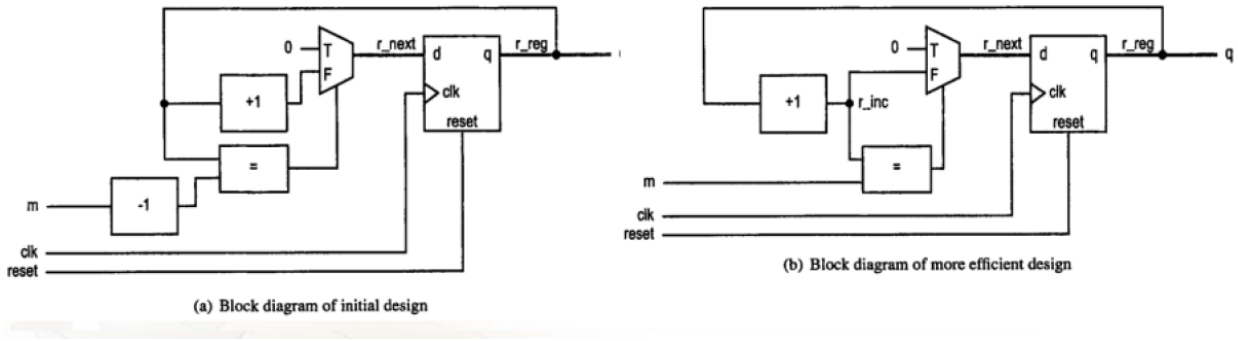
***Solution***

The critical path of the next-state logic consists of the incrementor and multiplexer (the comparator run in parallel with the incrementor and thus has no effect on the critical path).

$$t_{logic} = t_{inc} + t_{mux} = 5 + 0.75 = 5.75\ ns$$
$$t_{clk} = t_{logic} + t_{cq} + t_{setup} = 5.75 + 1 + 0.5 = 7.25\ ns$$
$$f_{clk} = \frac{1}{t_{clk}} = 138\ MHz$$

**Question 3 (Chu 8.12)**

Consider the two block diagrams of the programmable mod-m counter in Figure 8.14. Assume that no further optimization will be performed during synthesis. Use the timing information in Problem 8.1 1 to determine the maximal clock rates of the two configurations.




***Solution***

a)

$$t_{logic} = t_{inc} + t_{mux} = 5 + 0.75 = 5.75\ ns$$
$$t_{clk} = t_{logic} + t_{cq} + t_{setup} = 5.75 + 1 + 0.5 = 7.25\ ns$$
$$f_{clk} = \frac{1}{t_{clk}} = 138\ MHz$$

b)

$$t_{logic} = t_{inc} + t_{comp} + t_{mux} = 5 + 3 + 0.75 = 8.75\ ns$$
$$t_{clk} = t_{logic} + t_{cq} + t_{setup} = 8.75 + 1 + 0.5 = 10.25\ ns$$
$$f_{clk} = \frac{1}{t_{clk}} = 97\ MHz$$

**Question 4**

Odd/Even Parity Detector

Consider a logic circuit that counts the number of ones in a bit serial input stream. If the circuit asserts its output when the input contains an odd number of ones, it is called odd parity. If it asserts its output when the input contains an even number of ones, it is called even parity. Use FSM to design an odd parity detector circuit.

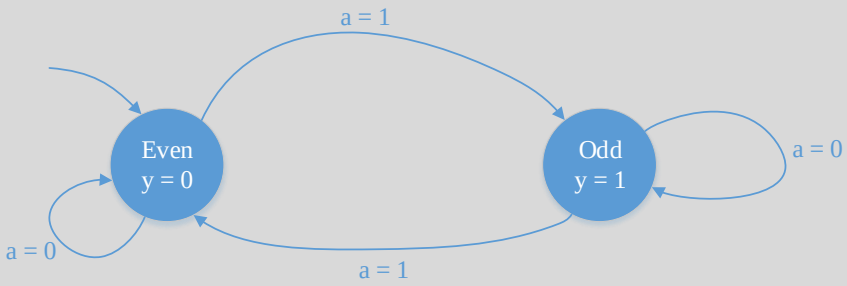
***Solution***

FSM Design Process:

1. Draw the state diagram.
2. Translate the state diagram into a state transition table.
3. Write the logic expressions for next-state and output logic using the state transition table.
4. Implement the circuit using logic gates.

1. Draw the state diagram

The circuit can have two states: Either even or odd number of ones have been seen since reset.




2. State transition table

| Symbolic Table | Present state | Input (a) | Next state | Output (y) |
|---|---|---|---|---|
| | Even | 0 | Even | 0 |
| | Even | 1 | Odd | 0 |
| | Odd | 0 | Odd | 1 |
| | Odd | 1 | Even | 1 |

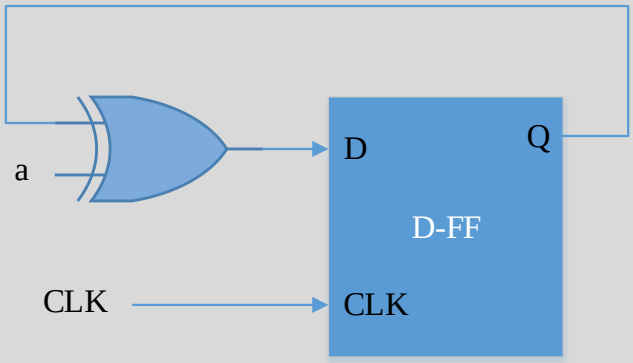
| Encoded Table | Present state (PS) | Input (a) | Next state (NS) | Output (y) |
|---|---|---|---|---|
| | 0 | 0 | 0 | 0 |
| | 0 | 1 | 1 | 0 |
| | 1 | 0 | 1 | 1 |
| | 1 | 1 | 0 | 1 |

3. Logic expressions

$$NS = a \oplus PS$$

$$y = PS$$

4. Implementation

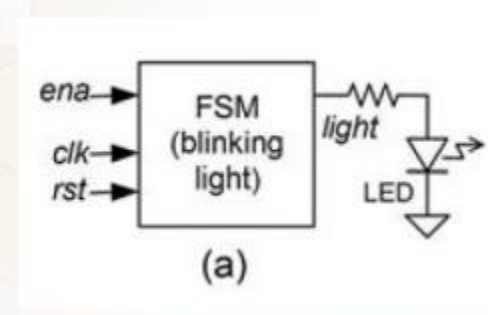



**Question 5**

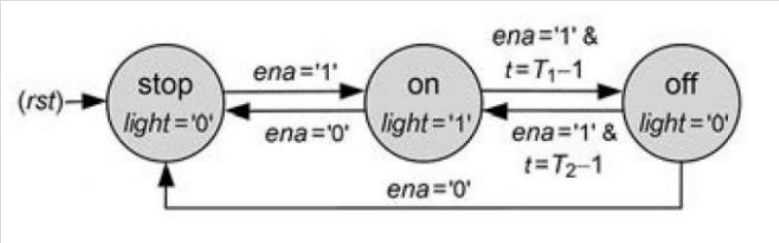
Blinking light

This problem concerns a circuit that must turn a light on and off, remaining on during T1 clock periods and off during T2 clock periods. An important desired feature is that when the circuit is enabled it must start from a state with the light on (to prevent the user from thinking that the circuit is not working when large transition times are involved).

a) Design the FSM.
b) Write a VHDL to describe the operation of the circuit.




***Solution***



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

entity blink is
    port(
        clk, rst, ena: in std_logic;
        light: in std_logic_vector(1 downto 0)
    );

    constant T1: integer:= 10;
    constant T2: integer:= 10;
end blink;

architecture blink_arch of blink is
    type state is (stop, onState, offState);
    signal prState, nxState: state;
    signal t, t_next: std_logic_vector(7 downto 0);
begin

    -- FSM state register
    process(clk, rst)
    begin
        if rst = '1' then
            prState <= stop;
        elsif rising_edge(clk) then
            prState <= nxState;
        end if;
    end process;

    -- FSM next state and output logic
    process(all)
    begin
        case prState is
            when stop =>
                light <= '0';

                if ena = '1' then
                    nxState <= onState;
                else
                    nxState <= stop;
                end if;

                t_next <= (others => '0');

            when onState =>
                light <= '1';

                if ena = '0' then
                    nxState <= stop;
                    t_next <= (others => '0');
                elsif (ena = '1' and t = T1-1) then
                    nxState <= offState;
                    t_next <= (others => '0');
                else
                    nxState <= onState;
                    t_next <= t+1;
                end if;

            when offState =>
                light <= '0';

                if ena = '0' then
                    nxState <= stop;
                    t_next <= (others => '0');
                elsif (ena = '1' and t = T2-1) then
                    nxState <= onState;
                    t_next <= (others => '0');
                else
                    nxState <= offState;
                    t_next <= t+1;
                end if;
        end case;
    end process;
```

```
75
76      -- t counter (for T1 and T2 delays)
77      process(clk, rst)
78      begin
79          if rst='1' then
80              t <= (others => '0');
81          else
82              t <= t_next;
83          end if;
84      end process;
85
86  end blink_arch;
87
88
```