OF THE
DEPARTMENT OF
INFORMATION ENGINEERING

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Digital Systems

Sequential Logic - Exercises

Marta Bagatin, marta.bagatin@unipd.it

Degree Course in Information Engineering

Academic Year 2023-2024

Exercise 4.6

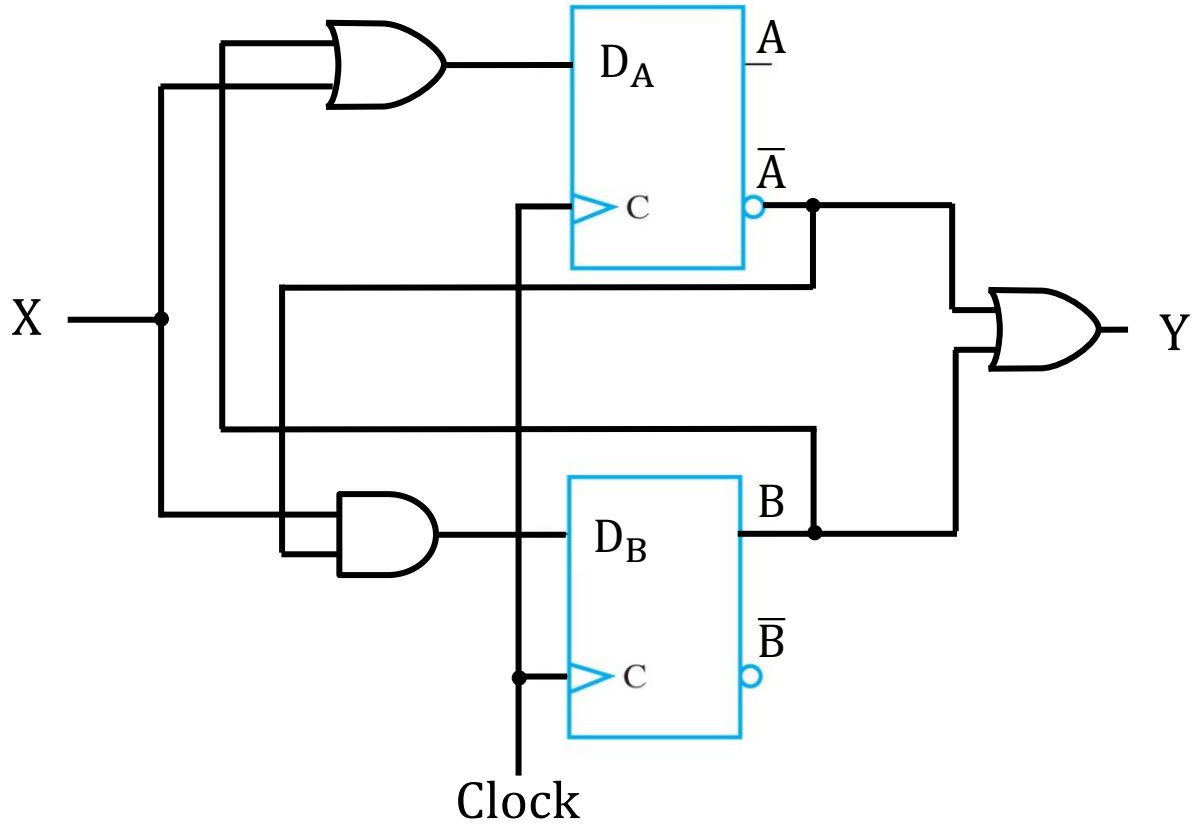
- Example of **sequential circuit analysis**, starting from the flip-flop input equations and output equation
- A sequential circuit with two D flip-flops A and B, input X, and output Y is specified by the following equations:
  - $Y = \overline{A} + B$
  - $D_A = X + B$
  - $D_B = X \cdot \overline{A}$

a) Draw the logic diagram of the circuit
b) Derive the state table
c) Is this a Mealy or a Moore machine?
d) Derive the state diagram

Exercise 4.6

a) Draw the **logic diagram** of the circuit

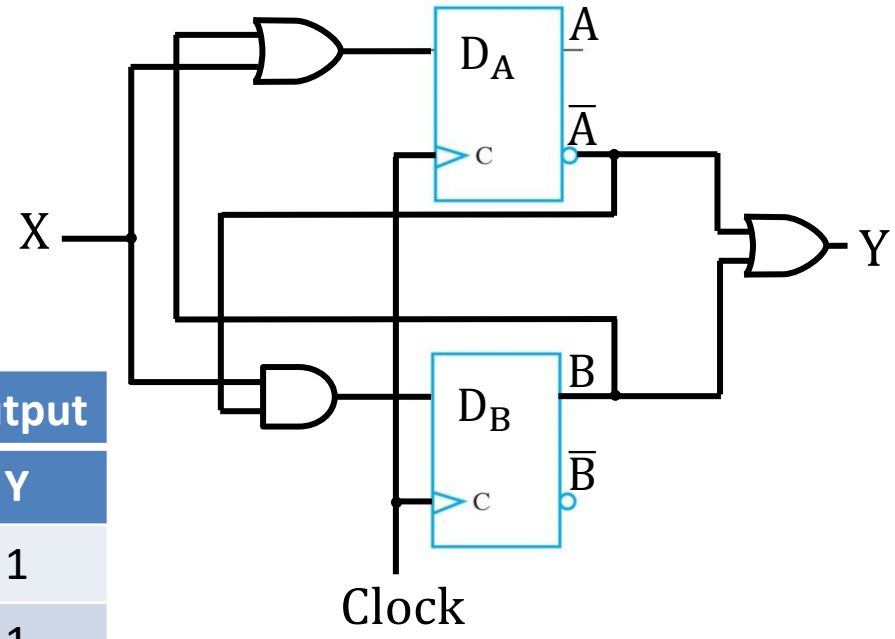
- $Y = \overline{A} + B$
- $D_A = X + B$
- $D_B = X \cdot \overline{A}$



Exercise 4.6

b) Derive the **state table**

- $Y = \overline{A} + B$
- $D_A = X + B$
- $D_B = X \cdot \overline{A}$



Present state		Input	Next state		Output
A	B	X	D_A	D_B	Y
0	0	0	0	0	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Exercise 4.6

c) Is this a Mealy or Moore machine?

It's a **Moore machine**, because the output depends ONLY on the present state and not on the input

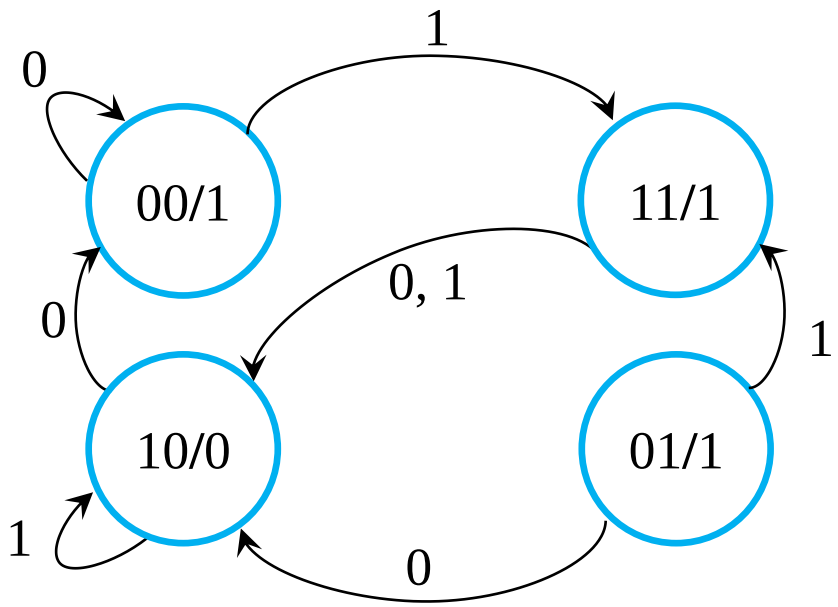
- $Y = \overline{A} + B$
- $D_A = X + B$
- $D_B = X \cdot \overline{A}$

Present state		Input	Next state		Output
A	B	X	D_A	D_B	Y
0	0	0	0	0	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Exercise 4.6

d) Find the **state diagram**

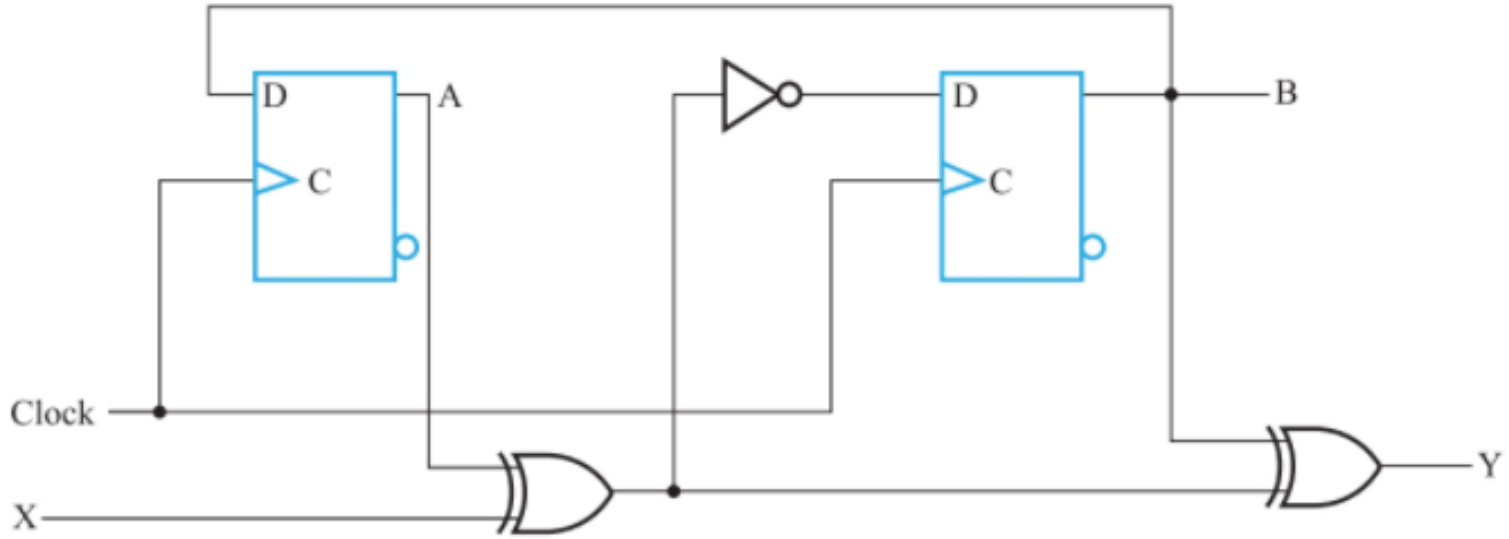
- $Y = \overline{A} + B$
- $D_A = X + B$
- $D_B = X \cdot \overline{A}$



Present state		Input	Next state		Output
A	B	X	D_A	D_B	Y
0	0	0	0	0	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	0	1

Exercise 4.11

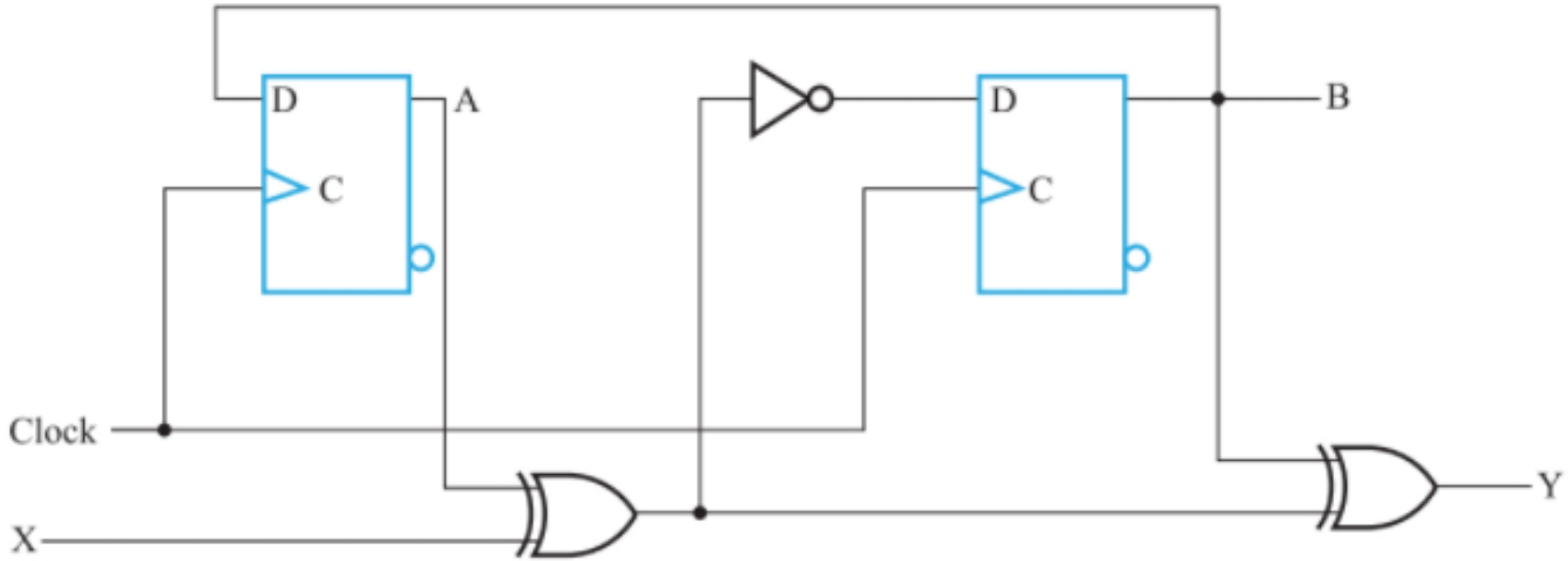
- Example of **sequential circuit analysis**, starting from the logic diagram
- A sequential circuit consists of two D flip-flops (with asynchronous reset), one input X, and one output Y. Reset brings the system to initial state «00». The logic diagram is shown in the figure:



a) Derive the state update and output equations
b) Find the state table
c) Derive the state diagram

Exercise 4.11

a) Derive the state update and output equations



State update equations: $D_A = A\,(t+1) = B$

$D_B = B\,(t+1) = \overline{X \oplus A}$

Output equation: $Y = B \oplus (X \oplus A)$

Exercise 4.11

b) Derive the state table

State update equations: $D_A = A\,(t+1) = B$

$D_B = B\,(t+1) = \overline{X \oplus A}$

Output equation: $Y = B \oplus (X \oplus A)$

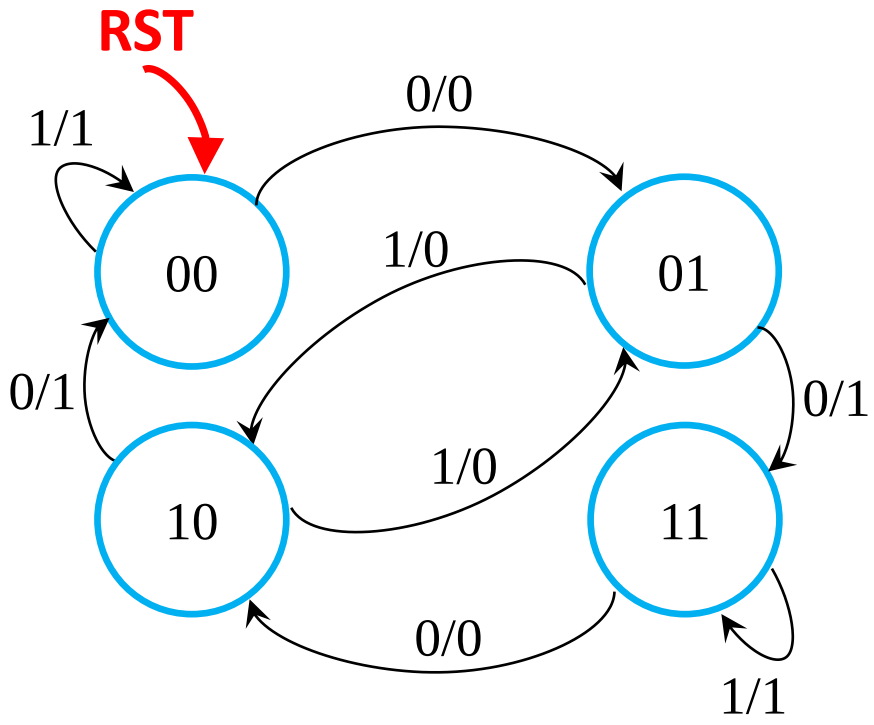
Present state		Input	Next state		Output
A	B	X	D_A	D_B	Y
0	0	0	0	1	0
0	0	1	0	0	1
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

Exercise 4.11

c) Derive the state diagram

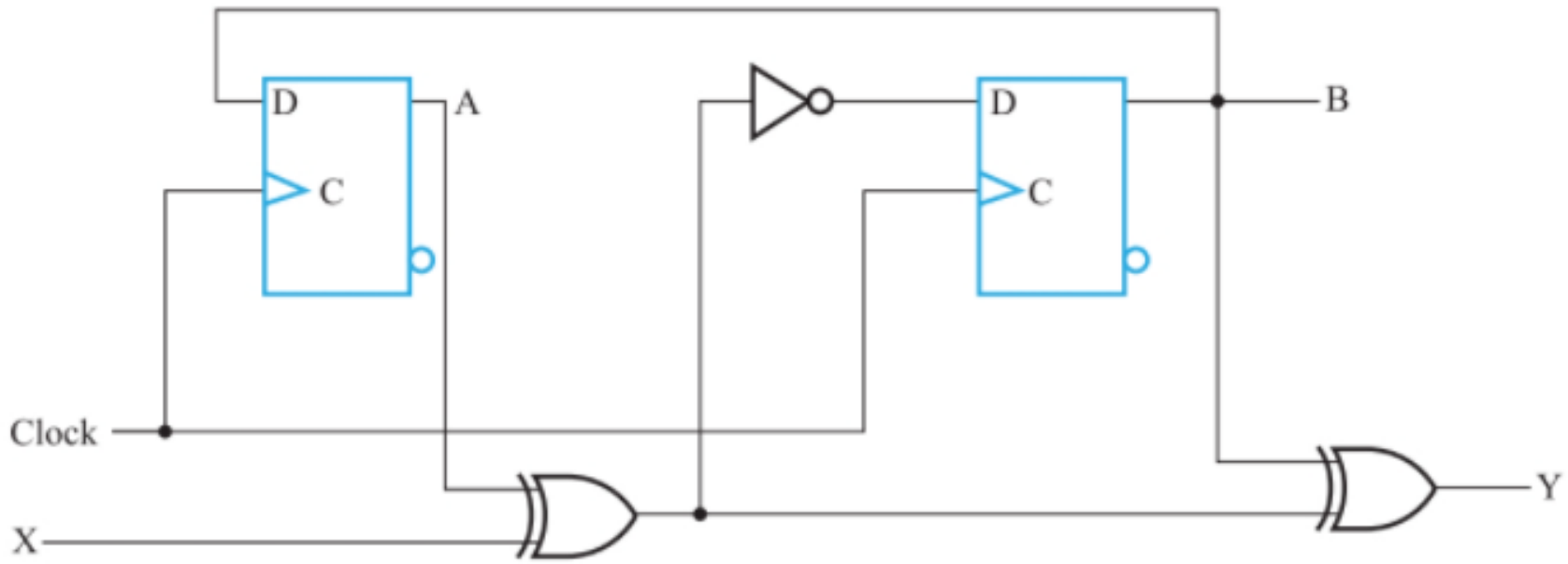
Present state		Input	Next state		Output
A	B	X	D_A	D_B	Y
0	0	0	0	1	0
0	0	1	0	0	1
0	1	0	1	1	1
0	1	1	1	0	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	0
1	1	1	1	1	1

Mealy Model (the output depends on both the present state and the input)



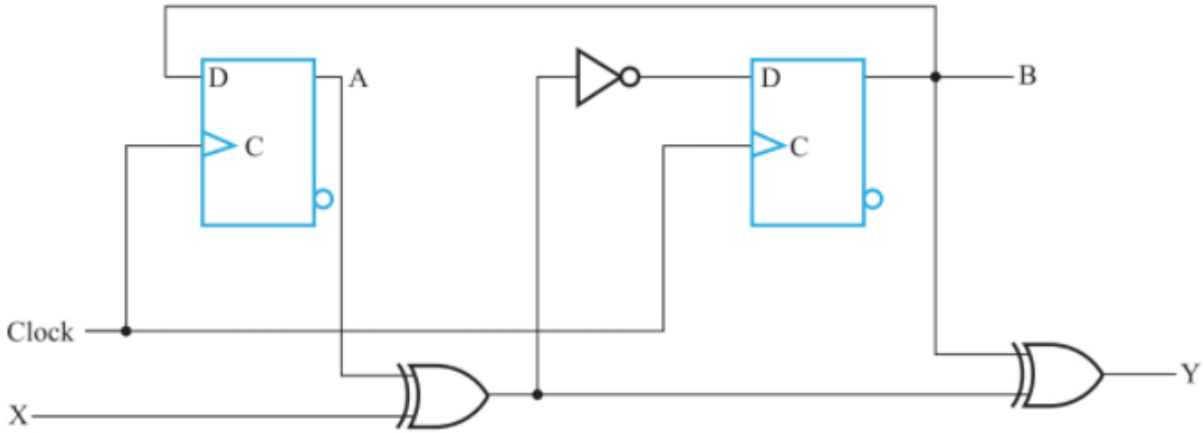
Exercise 4.40

- Write a **structural VHDL description** of the circuit in exercize 4.11:



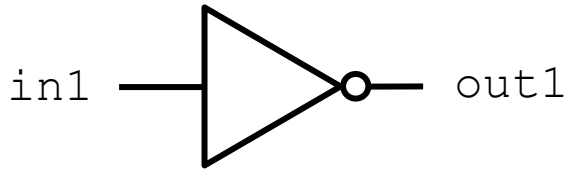
Exercise 4.40

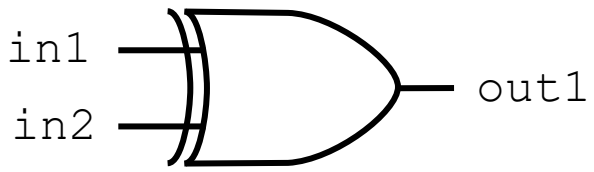
```
library IEEE;
use IEEE.std_logic_1164.all;

entity prob_4_40 is
   port (clk, rst, X: in std_logic;
                   Y: out std_logic);
end prob_4_40;

architecture impl_struct of prob_4_40 is

   signal A, B, XxorA, XxorA_n: std_logic;

   -- direct instantiation of the components
begin
   INV: entity work.inv(impl) port map (XxorA, XxorA_n);
   DFFA: entity work.dff(impl) port map (rst, clk, B, A);
   DFFB: entity work.dff(impl) port map (rst, clk, XxorA_n, B);
   XOR1: entity work.xor2(impl) port map (A, X, XxorA);
   XOR2: entity work.xor2(impl) port map (B, XxorA, Y);

end impl_struct;
```



Exercise 4.40

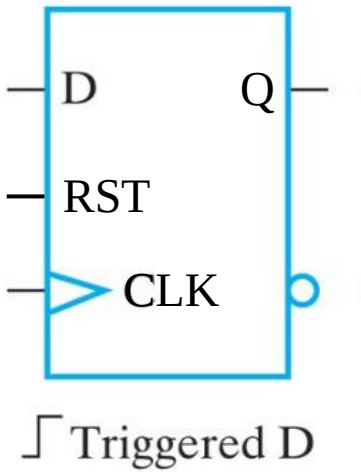
```
-- description of inverter
library ieee;
use ieee.std_logic_1164.all;
entity inverter is
   port (in1 : in std_logic;
         out1 : out std_logic);
end inverter;
architecture impl of inverter is
begin
   out1 <= not in1;
end impl;
-- description of XOR gate
library ieee;
use ieee.std_logic_1164.all;
entity xor2 is
   port (in1, in2 : in std_logic;
               out1 : out std_logic);
end xor2;
architecture impl of xor2 is
begin
   out1 <= in1 xor in2;
end impl;
```





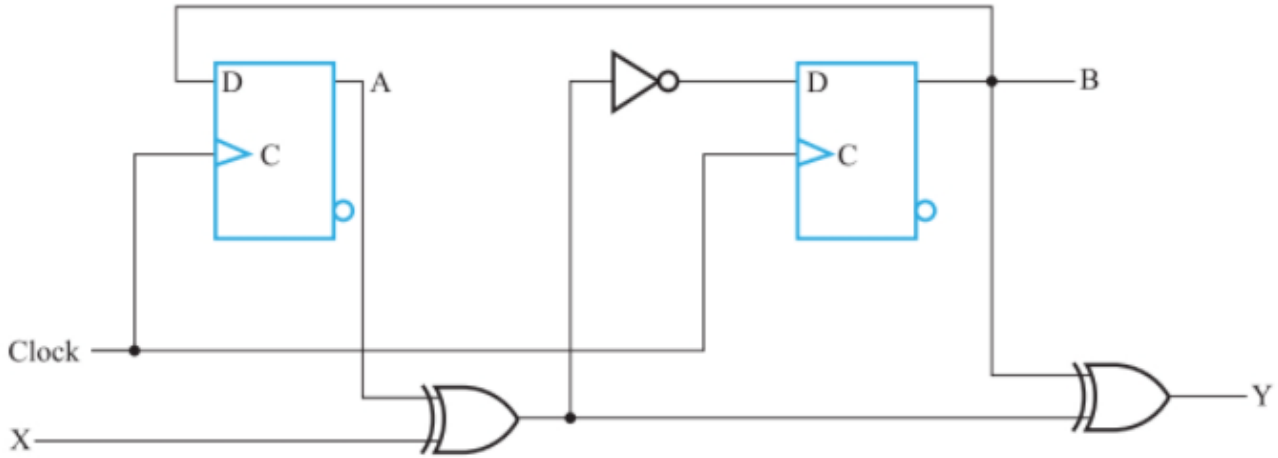
Exercise 4.40

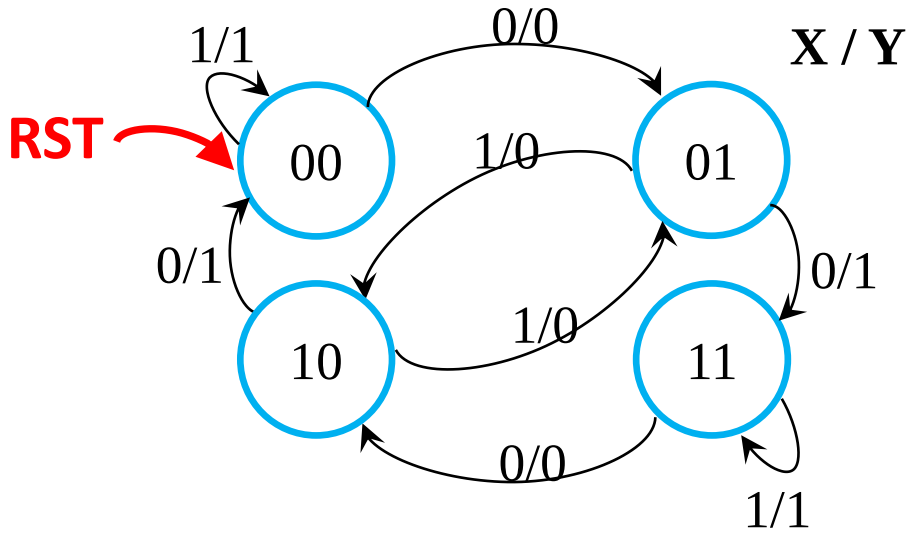
```
-- description of D flip-flop with asynchronous reset

library ieee;
use ieee.std_logic_1164.all;
entity dff is
   port ( rst, clk, d: in std_logic;
                    q: out std_logic);
end dff;

architecture impl of dff is
begin
   process (rst, clk)
   begin
      if rst = '1' then
         q <= '0';
      elsif (clk'event and clk = '1') then
         q <= d;
      end if;
   end process;
end impl;
```



Exercise 4.41

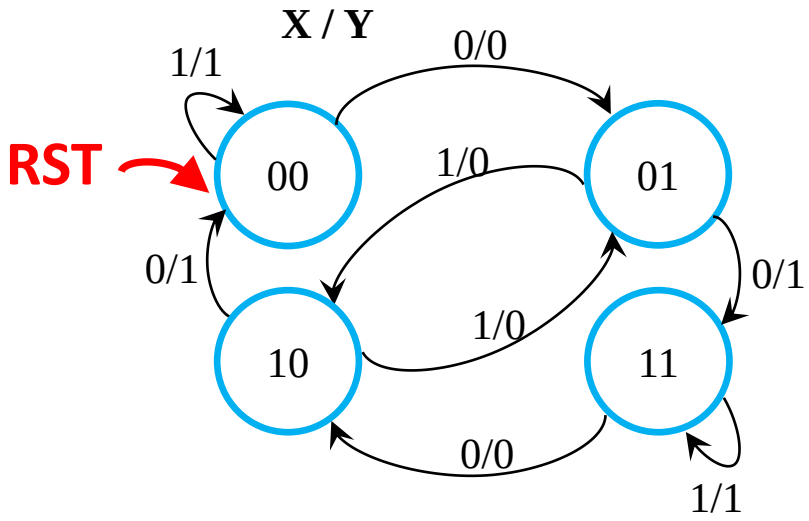
- Provide a **behavioral VHDL** description of the circuit in exercize 4.11, using a process to describe the state diagram. Suppose Reset brings the system to 00 state:



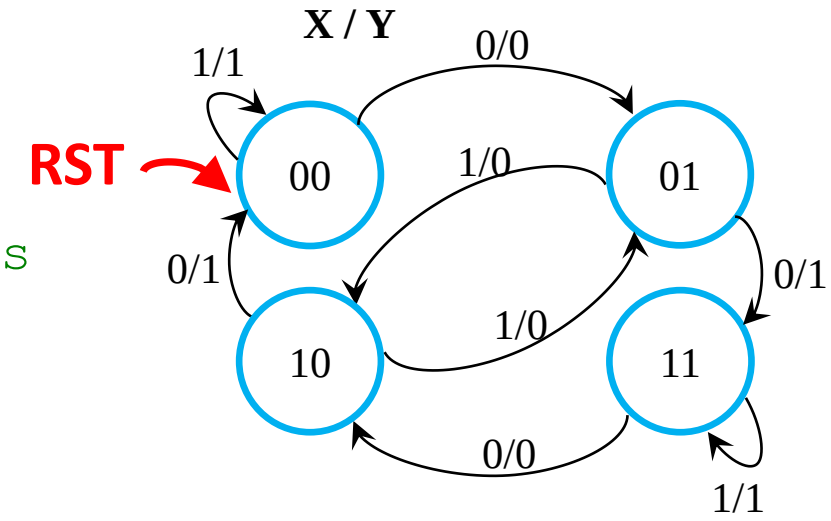


Exercise 4.41

```
library IEEE;
use IEEE.std_logic_1164.all;

entity prob_4_41 is
   port ( clk, rst, X: in std_logic;
                    Y: out std_logic);
end prob_4_41;

architecture impl_beh of prob_4_41 is

   type state_type is (S0, S1, S2, S3);
   signal state, next_state: state_type;

begin
```

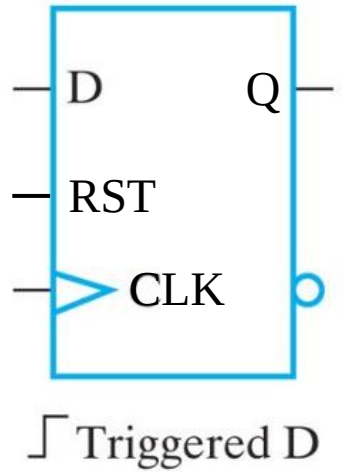


Exercise 4.41




```
-- 1st process: updates status and implements
asynchronous reset

   state_register: process (clk, rst)
   begin
      if rst then
         state <= S0;
      elsif (clk'event and clk = '1') then
         state <= next_state;
      end if;
   end process;
```

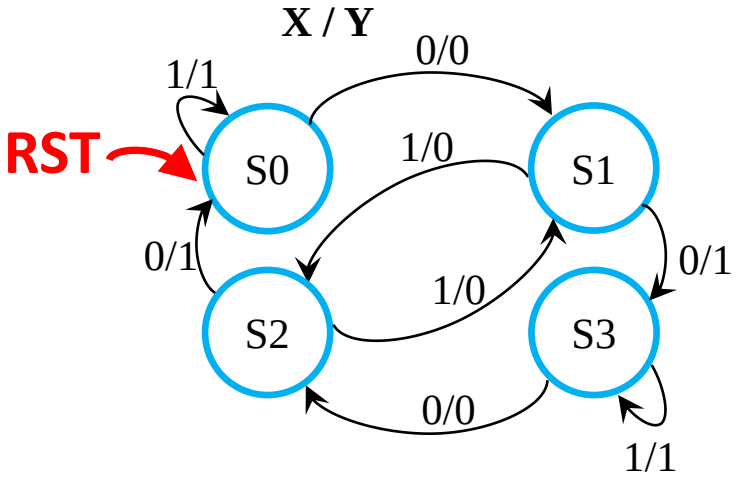
Process 1: Implements the state register with positive edge triggered D-FF



Exercise 4.41

- 2nd process: computes next state as a combinational function of input and present state

```
next_state_comb: process (X, state)
begin
   case state is
      when S0 =>
         if X = '1' then
            next_state <= S0;
         else
            next_state <= S1;
         end if;

      when S1 =>
         if X = '1' then
            next_state <= S2;
         else
            next_state <= S3;
         end if;
```

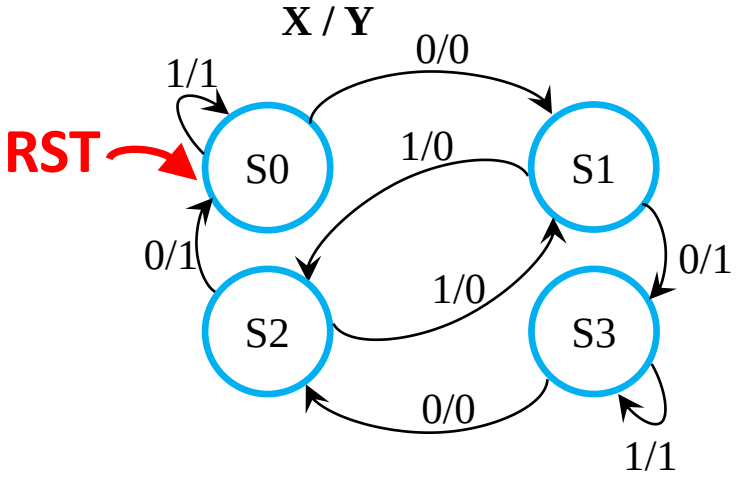


Process 2: Implements the combinational logic calculating the next state as a function of present state and input

Exercise 4.41

- 2nd process: continued

```
        when S2 =>
            if X = '1' then
                next_state <= S1;
            else
                next_state <= S0;
            end if;

        when S3 =>
            if X = '1' then
                next_state <= S3;
            else
                next_state <= S2;
            end if;
    end case;
end process;
```

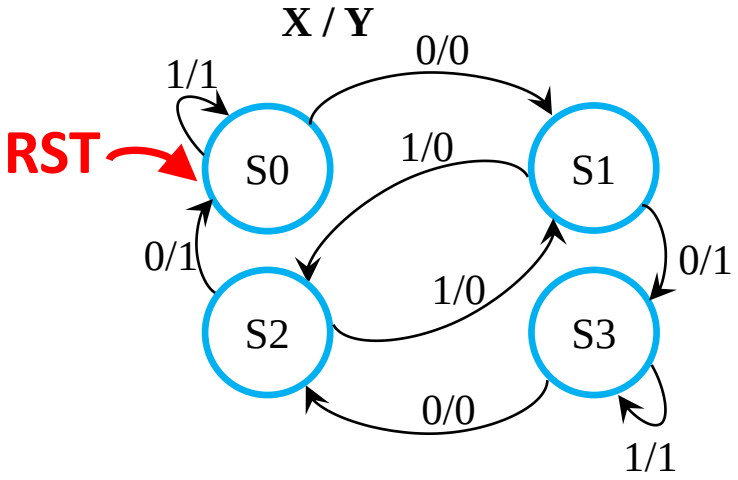


Process 2: Implements the combinational logic calculating the next state as a function of present state and input

Exercise 4.41

- 3rd process: calculates the output as a combinational function of input and present state

```
output_comb: process (X, state)
begin
case state is
   when S0 =>
      if X = '1' then
         Y <= '1';
      else
         Y <= '0';
      end if;

   when S1 =>
      if X = '0' then
         Y <= '1';
      else
         Y <= '0';
      end if;
```

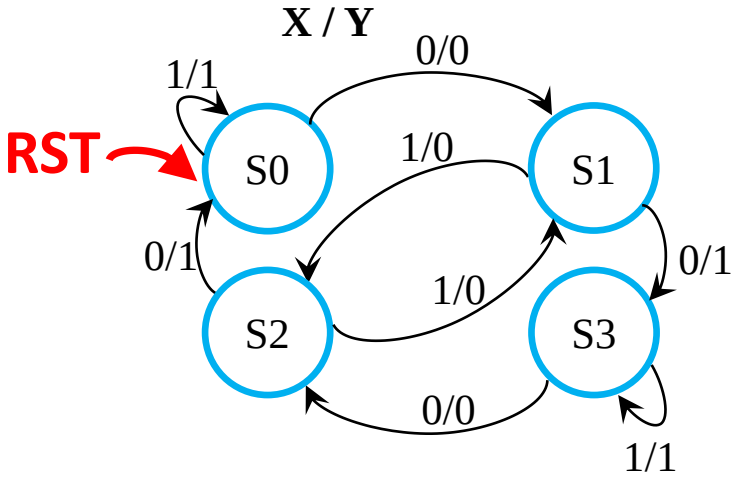


Process 3: Implements the combinational logic calculating the output as a function of input and present state

Exercise 4.41

- 3rd process: continued

```
        when S2 =>
            if X = '0' then
                Y <= '1';
            else
                Y <= '0';
            end if;

        when S3 =>
            if X = '1' then
                Y <= '1';
            else
                Y <= '0';
            end if;
    end case;
    end process;

end impl_beh;
```



Exercise 4.26

- Example of **state diagram derivation, starting from the circuit specifications**
- A serial sequence detector must be able to detect the presence of three consecutive 1s on an incoming data line. When input X has three consecutive 1s, then output Z = 1; in all the other cases, Z = 0. Once Z = 1, it remains '1' until there is a 0 at the input. If X = 0, the circuit is reset.

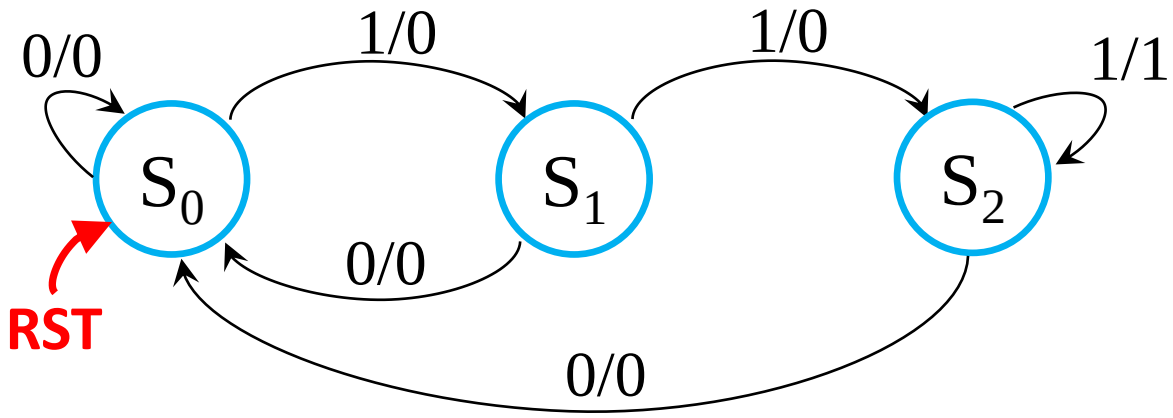
a) Find the state diagram of the circuit

b) Is this a Mealy or Moore sequential machine?

Exercise 4.26

- Input X has three consecutive 1s => output Z = 1
- Otherwise => output Z = 0
- After Z = 1, the system remains in that state until at least one 0 arrives at the input
- If X = 0, a reset takes place

a) Find the **circuit state diagram**

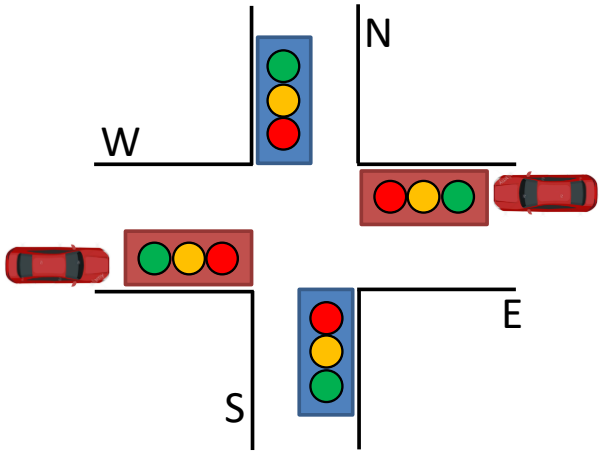


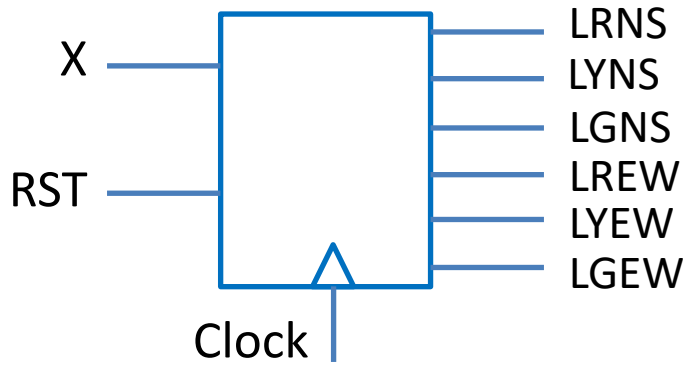
S_0: no '1' recognized at the input
S_1: one '1' recognized at the input
S_2: two '1' recognized at the input

b) It is a **Mealy machine**, as the output depends on both the present state and the input

Traffic Light Controller

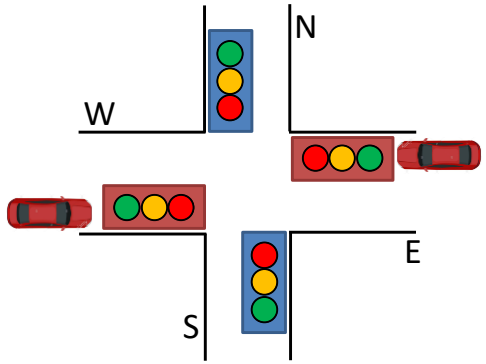
- Example of **synthesis of sequential circuit**
- At a road junction at the intersection of a NS road with an EW road, there are 4 traffic lights. **Design a sequential circuit for traffic light control (in total, 6 lights to be controlled)**, with an input X indicating the presence of a car waiting in the EW direction and 6 outputs indicating whether the corresponding traffic lights should be switched on (red NS, yellow NS, green NS, red EW, yellow EW, green EW) with the following specifications:
  - Reset (RST): leads to green light in NS direction and red light in EW direction
  - If a car in EW direction is detected (X = 1), the system enters a loop which turns the light green in EW direction and then returns to green in NS direction
  - Red light must be preceded by yellow light
  - Light in NS direction can be green only if light is red in EW direction, and vice versa

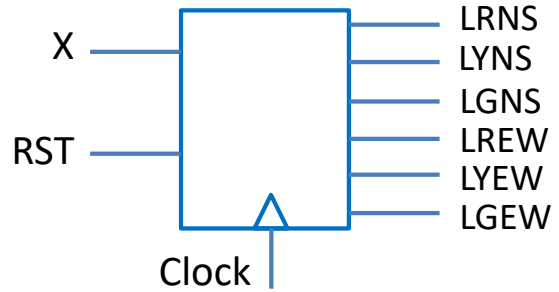


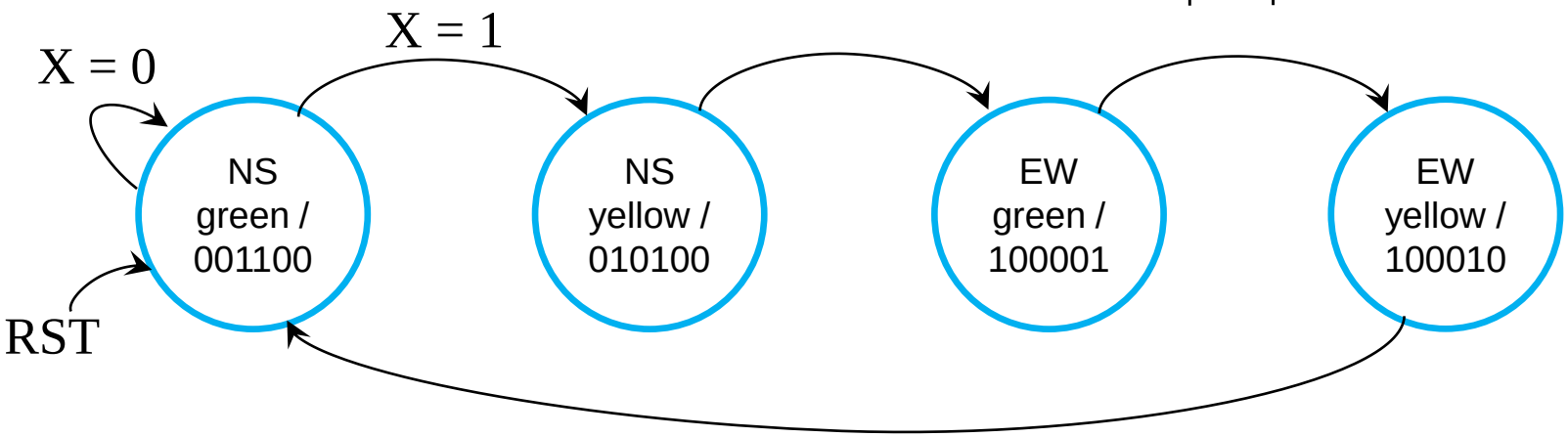


Traffic Light Controller: State Diagram

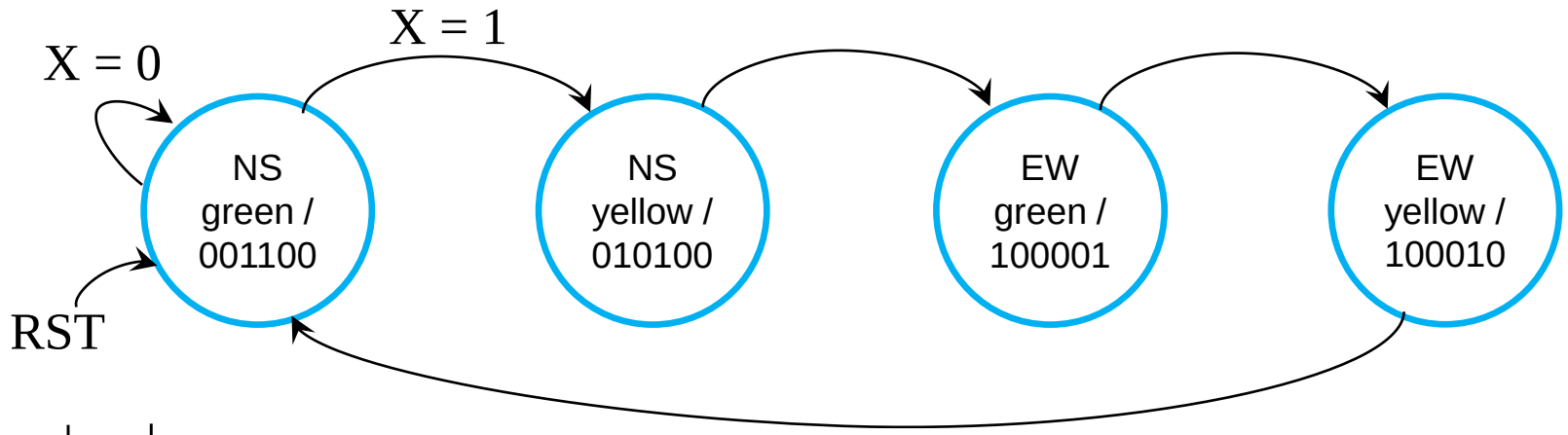
- Reset (RST): leads to green light in NS direction and red light in EW direction
- If a car in EW direction is detected (X = 1), the system enters a loop which turns the light green in EW direction and then returns to green in NS direction
- Red light must be preceded by yellow light
- Light in NS direction can be green only if light is red in EW direction, and vice versa

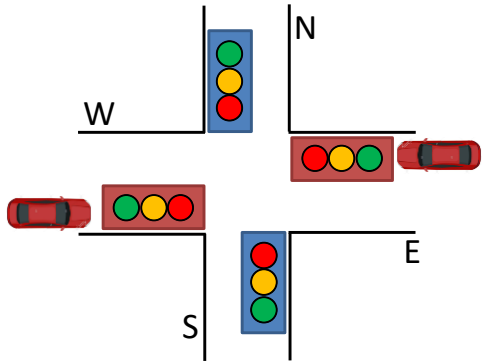


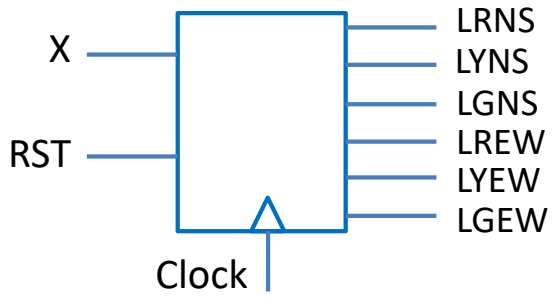




Traffic Light Controller: State Table



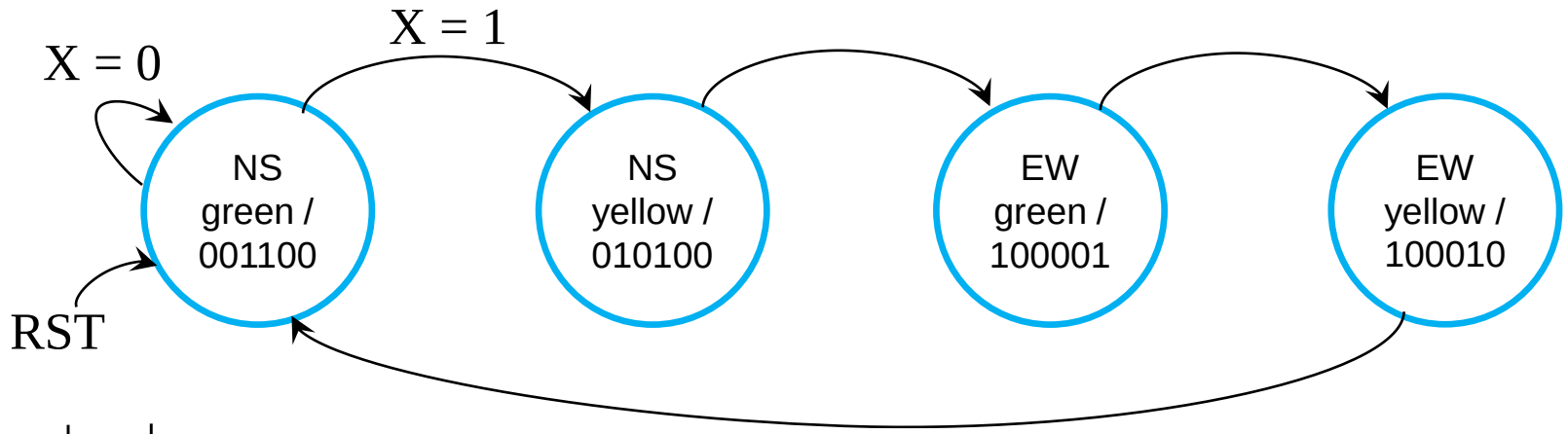


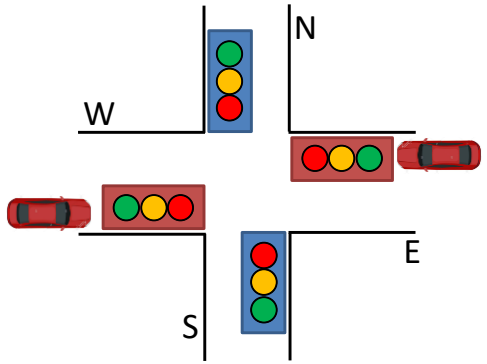


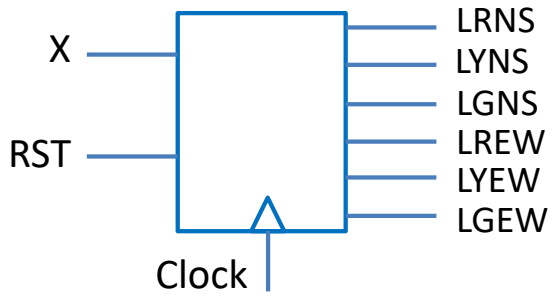
Present state	Next state		Output					
	X = 0	X = 1	LRNS	LYNS	LGNS	LREW	LYEW	LGEW
GNS	GNS	YNS	0	0	1	1	0	0
YNS	GEW	GEW	0	1	0	1	0	0
GEW	YEW	YEW	1	0	0	0	0	1
YEW	GNS	GNS	1	0	0	0	1	0

=> Moore model: output depends only on the state

Traffic Light Controller: State Assignment

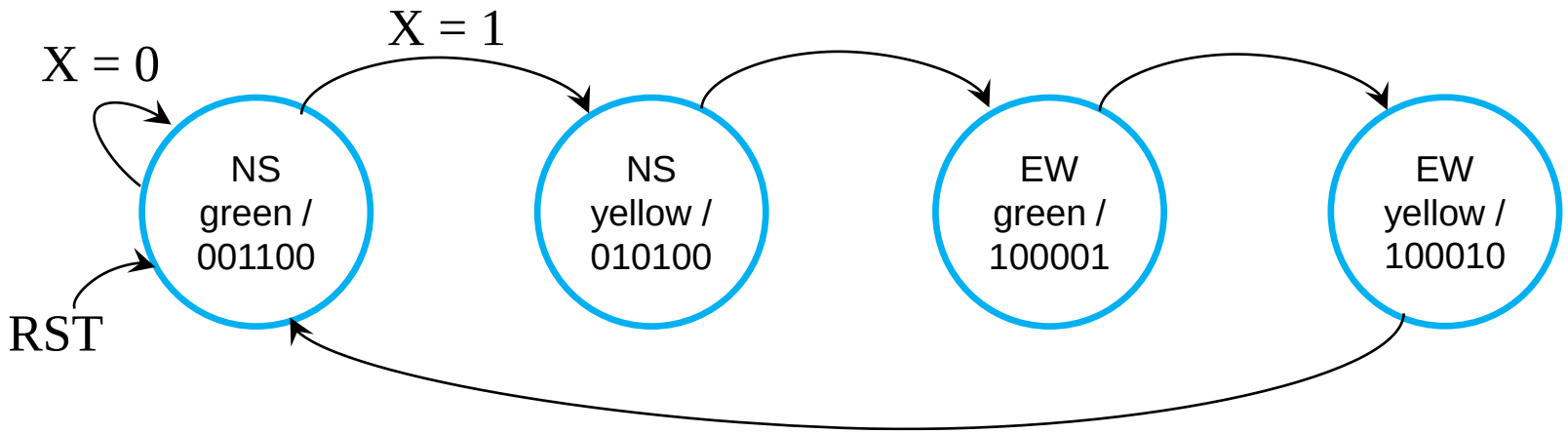






State	Gray encoding A B
GNS	0 0
YNS	0 1
GEW	1 1
YEW	1 0

Traffic Light Controller: Encoded State Table

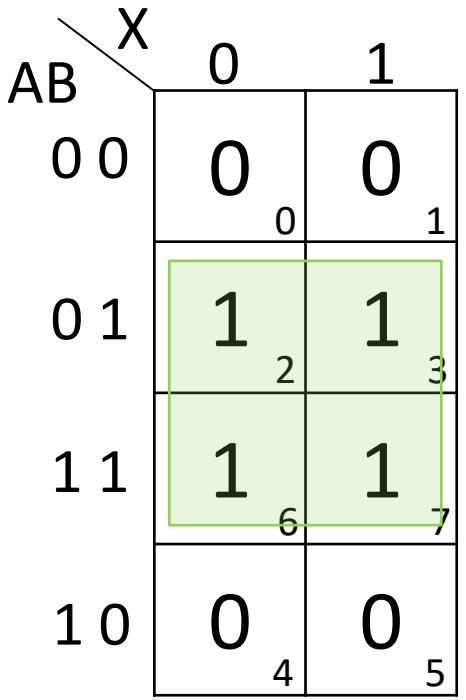


State	Gray encoding A B
GNS	0 0
YNS	0 1
GEW	1 1
YEW	1 0

Present state A B	Next state D_A D_B		Output LRNS LYNS LGNS LREW LYEW LGEW
	X = 0	X = 1	
0 0	0 0	0 1	0 0 1 1 0 0
0 1	1 1	1 1	0 1 0 1 0 0
1 1	1 0	1 0	1 0 0 0 0 1
1 0	0 0	0 0	1 0 0 0 1 0

Traffic Light Controller: Next State Equations

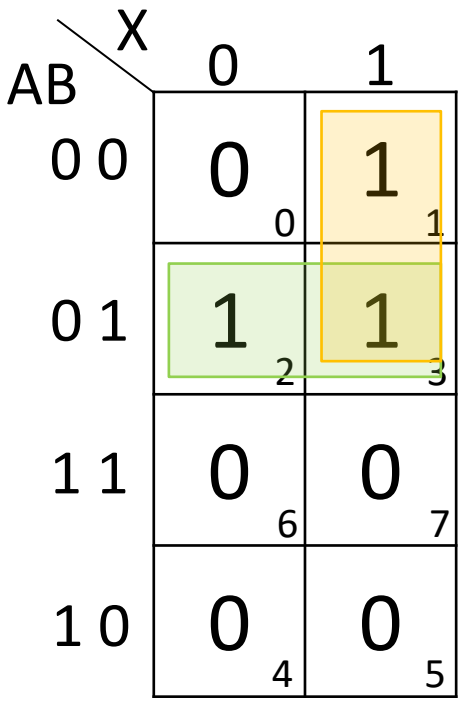
Present state A B	Next state D_A D_B		Output
	X = 0	X = 1	
0 0	0 0	0 1	001 100
0 1	1 1	1 1	010 100
1 1	1 0	1 0	100 001
1 0	0 0	0 0	100 010



$$D_A = A(t+1) = B$$

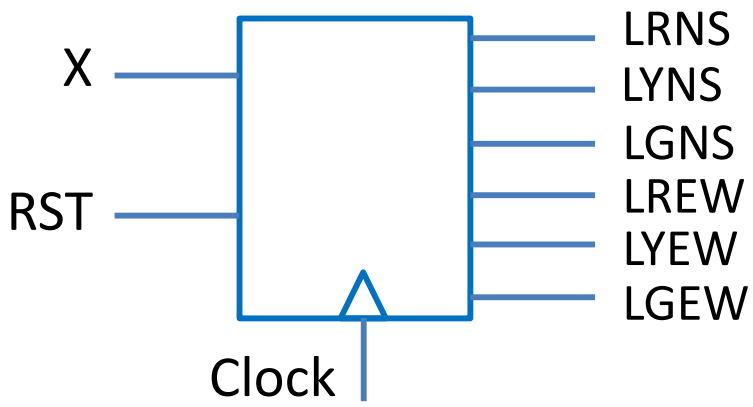
Traffic Light Controller: Next State Equations

Present state A B	Next state D_A D_B		Output
	X = 0	X = 1	
0 0	0 0	0 1	001 100
0 1	1 1	1 1	010 100
1 1	1 0	1 0	100 001
1 0	0 0	0 0	100 010



$$D_B = B(t+1) = \overline{A}B + \overline{A}X = \overline{A}(B + X)$$

Traffic Light Controller: Output Equations



Present state A B	Next state D_A D_B		Output
	X = 0	X = 1	
0 0	0 0	0 1	001 100
0 1	1 1	1 1	010 100
1 1	1 0	1 0	100 001
1 0	0 0	0 0	100 010

$$\mathrm{LRNS} = \mathrm{A}$$

$$\mathrm{LYNS} = \overline{\mathrm{A}} \cdot \mathrm{B}$$

$$\mathrm{LGNS} = \overline{\mathrm{A}} \cdot \overline{\mathrm{B}}$$

$$\mathrm{LREW} = \overline{\mathrm{A}}$$

$$\mathrm{LYEW} = \mathrm{A} \cdot \overline{\mathrm{B}}$$

$$\mathrm{LGEW} = \mathrm{A} \cdot \mathrm{B}$$

Traffic Light Controller: Final Circuit

State update equations:

$D_A = B$

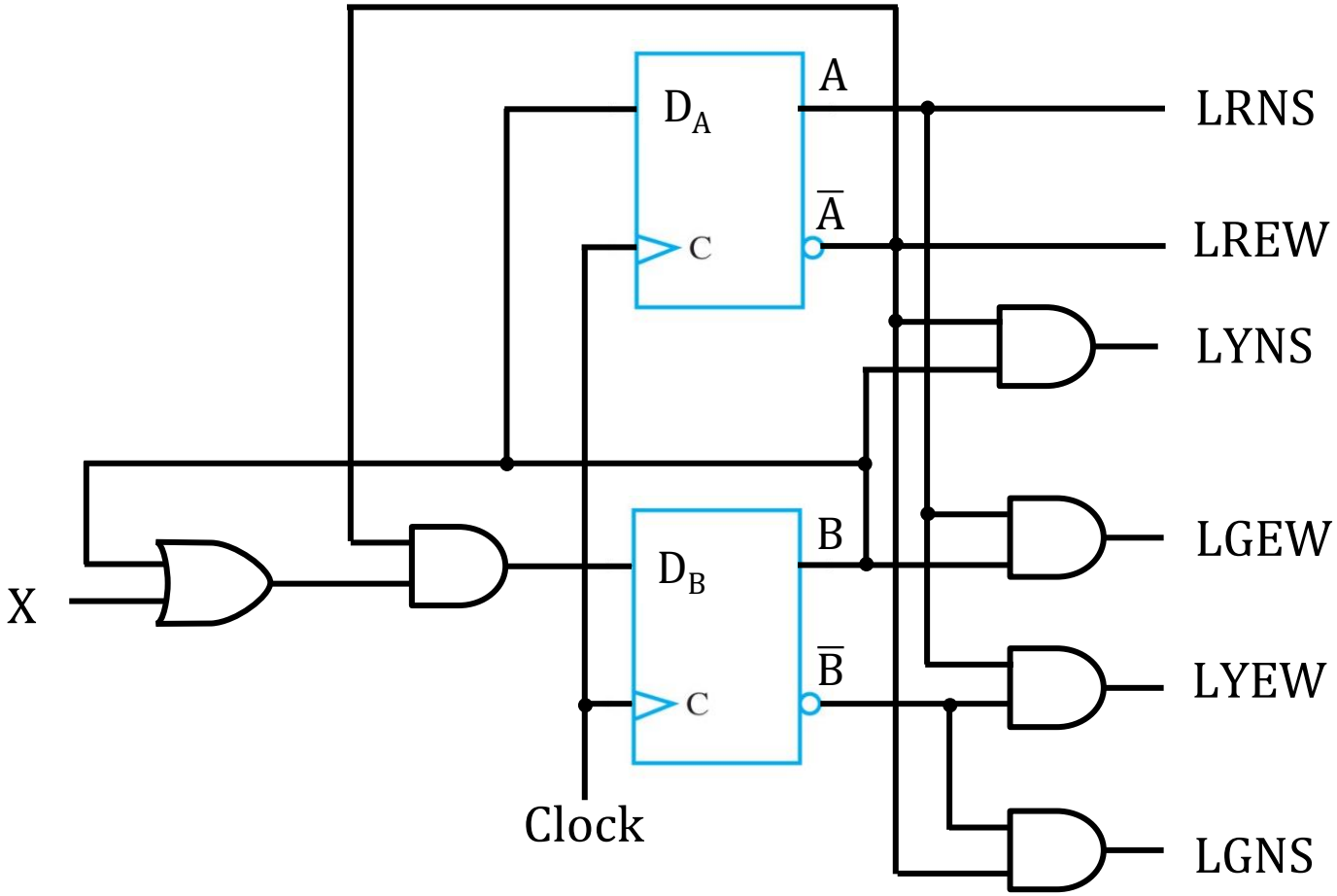
$D_B = \overline{A}B + \overline{A}X = \overline{A}(B + X)$

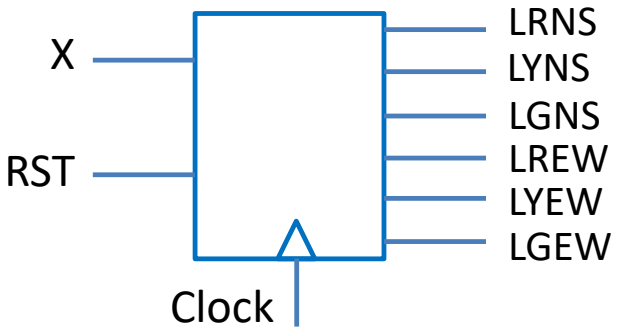
Output equations:

$LRNS = A$ $\quad LREW = \overline{A}$

$LYNS = \overline{A} \cdot B$ $\quad LYEW = A \cdot \overline{B}$

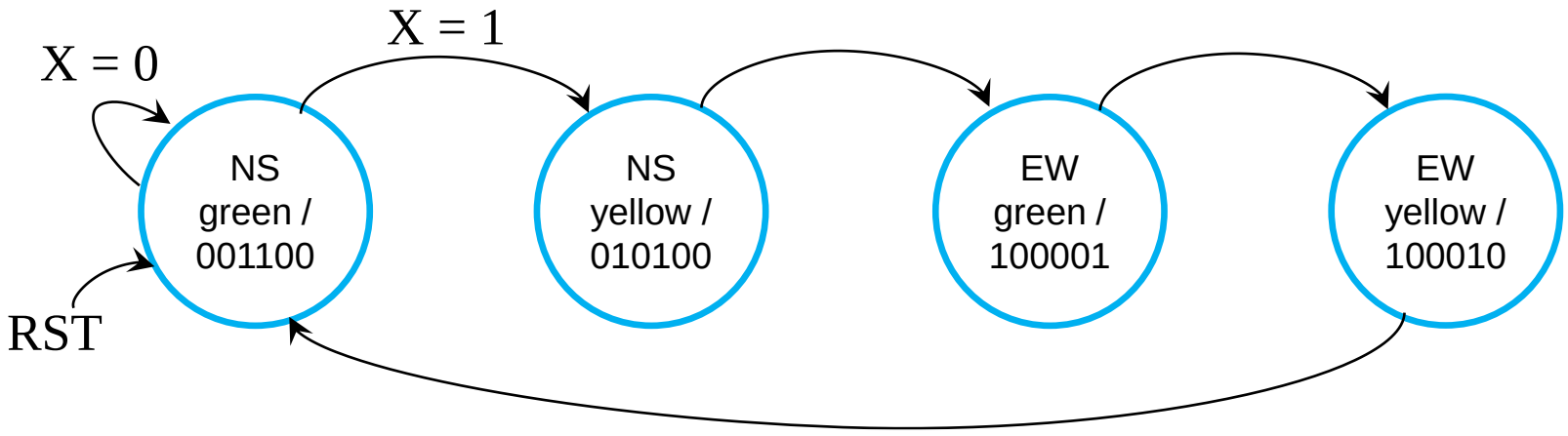
$LGNS = \overline{A} \cdot \overline{B}$ $\quad LGEW = A \cdot B$

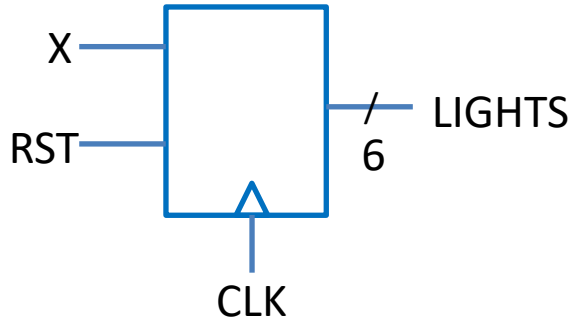




Traffic Light Controller: VHDL

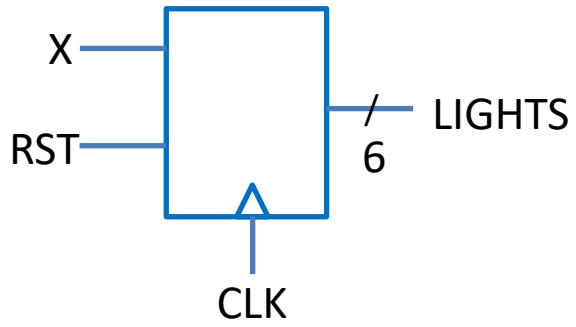
- Write a **behavioral VHDL description** for the traffic light controller sequential circuit with the following characteristics:
  - Input X, Outputs LRNS, LYNS, LGNS, LREW, LYEW, LGEW
  - Reset: leads to green light in NS direction and red light in EW direction
  - If a car in EW direction is detected (X = 1): make the light green in EW direction and then return to green in NS direction
  - Red light must have a yellow light before
  - Light in NS direction ca be green only if light is red in EW direction, and vice versa





Traffic Light Controller: Behavioral VHDL (1/4)

```
library IEEE;
use IEEE.std_logic_1164.all;

entity trafficLight is
   port ( clk, rst, X : in std_logic;
                lights : out std_logic_vector (5 downto 0));
--(LRNS, LYNS, LGNS, LREW, LYEW, LGEW)
end trafficLight;

architecture impl_beh of trafficLight is

-- constants for state encoding
   subtype state_type is std_logic_vector (1 downto 0);
   constant GNS: state_type : = "00";
   constant YNS: state_type : = "01";
   constant GEW: state_type : = "11";
   constant YEW: state_type : = "10";
   signal state, next_state: state_type;
begin
```

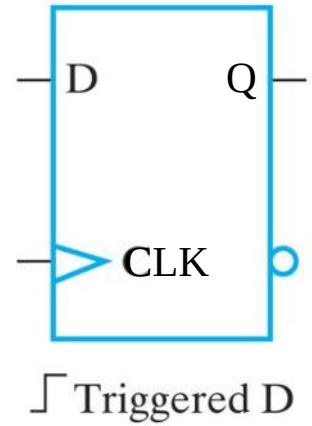


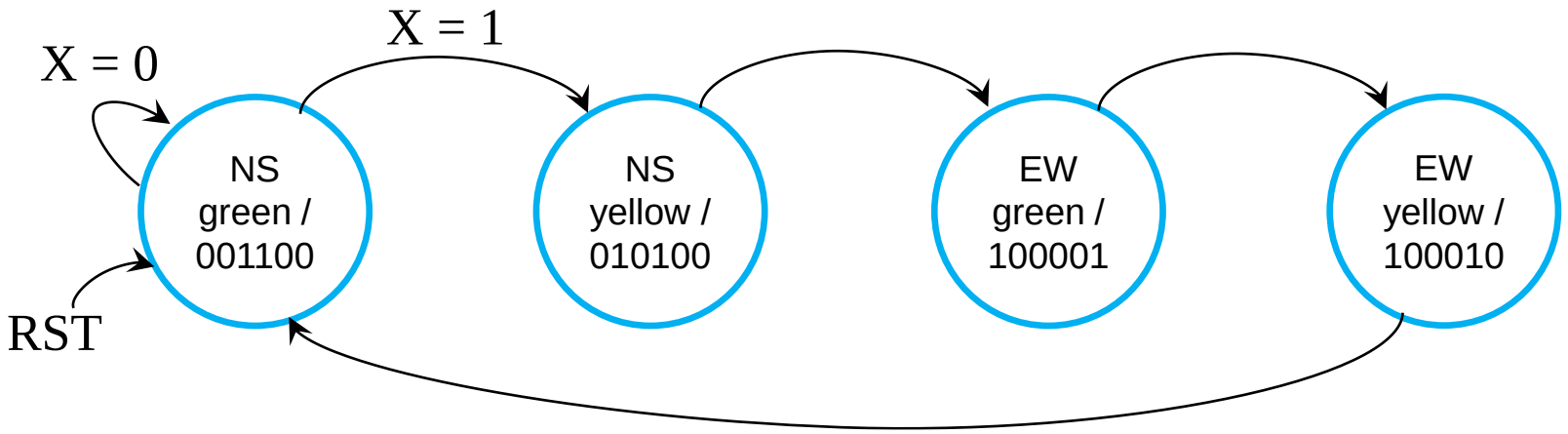
Using constants is convenient: if you want to change the encoding, just change here (the rest of the code remains unchanged)

Traffic Light Controller: Behavioral VHDL (2/4)

```
-- process 1: updates state and implements reset

state_register: process (clk, rst)
begin
   if (rst = '1') then
      state <= GNS;
   elsif (clk'event and clk = '1') then
      state <= next_state;
   end if;
end process;
```

Process 1: Implements the state register with PET D-FF and implements RESET

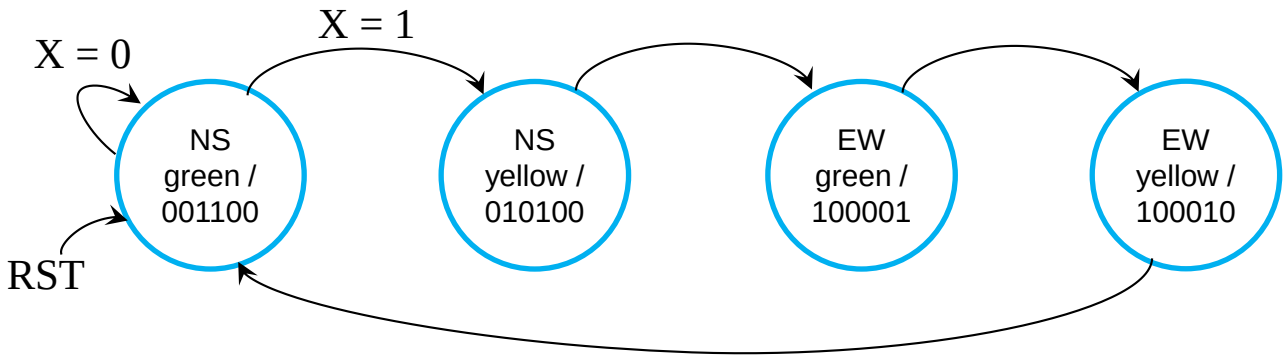




Traffic Light Controller: Behavioral VHDL (3/4)

```
-- process 2: calculates next state as a function of input and present state
 next_state_comb: process (X, state)
 begin
    case state is

       when GNS =>
          if X='0' then
             next_state <= GNS;
          else
             next_state <= YNS;
          end if;

       when YNS => next_state <= GEW;

       when GEW => next_state <= YEW;

       when YEW => next_state <= GNS;

       when others => null;
    end case;
 end process;
```

Process 2: Calculates the next state



All possible choices must be enumerated, state is a std_logic_vector type!

Traffic Light Controller: Behavioral VHDL (4/4)

```
-- process 3: Implements the output as a function of the state
-- LRNS, LYNS, LGNS, LREW, LYEW, LGEW

output_comb: process (state)
begin
   case state is

      when GNS => lights <= "001100";

      when YNS => lights <= "010100";

      when GEW => lights <= "100001";

      when YEW => lights <= "100010";

      when others => lights <= "XXXXXX";

   end case;
end process;
end impl_beh;
```

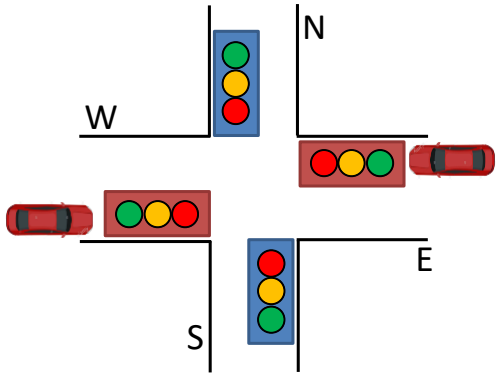
Process 3: Calculates the output

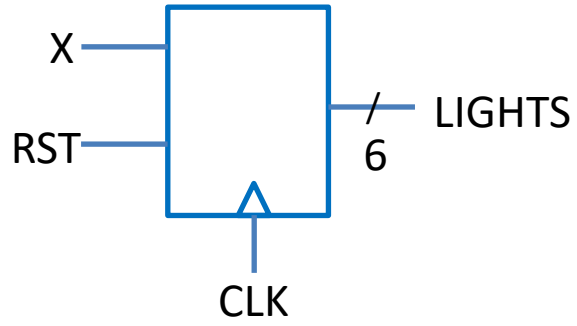
State	Output
GNS	001 100
YNS	010 100
GEW	100 001
YEW	100 010

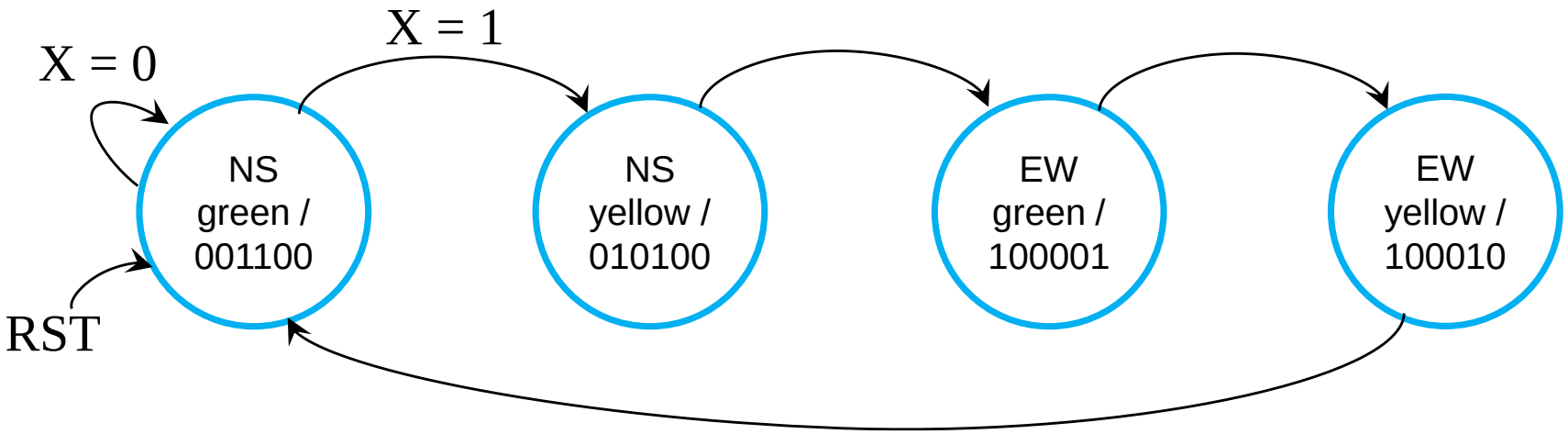
If 'state' does not assume any of the possible values, the output is set to 'undefined' value

Traffic Light Controller: Testbench

- Write a **testbench** of the traffic light controller, applying and simulating the circuit, simulating transitions through all the states

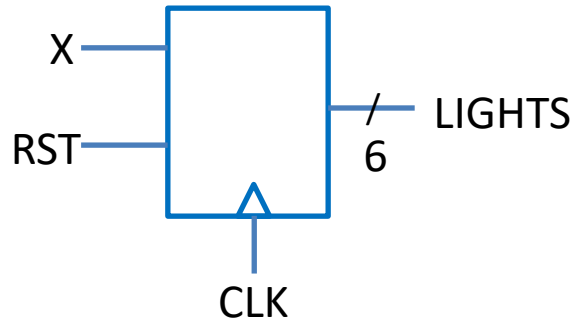






Traffic Light Controller: Testbench (1/3)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity traffic_test is
end traffic_test;

architecture test of traffic_test is

signal clock, reset, carEW: std_logic;
signal output: std_logic_vector (5 downto 0);

constant PERIOD: time : = 100 ns;

component trafficLight is
   port ( clk, rst, X : in std_logic;
                lights : out std_logic_vector (5 downto 0));
end component;

begin

DUT: trafficLight port map (clock, reset, carEW, output);
```



Traffic Light Controller: Testbench (2/3)

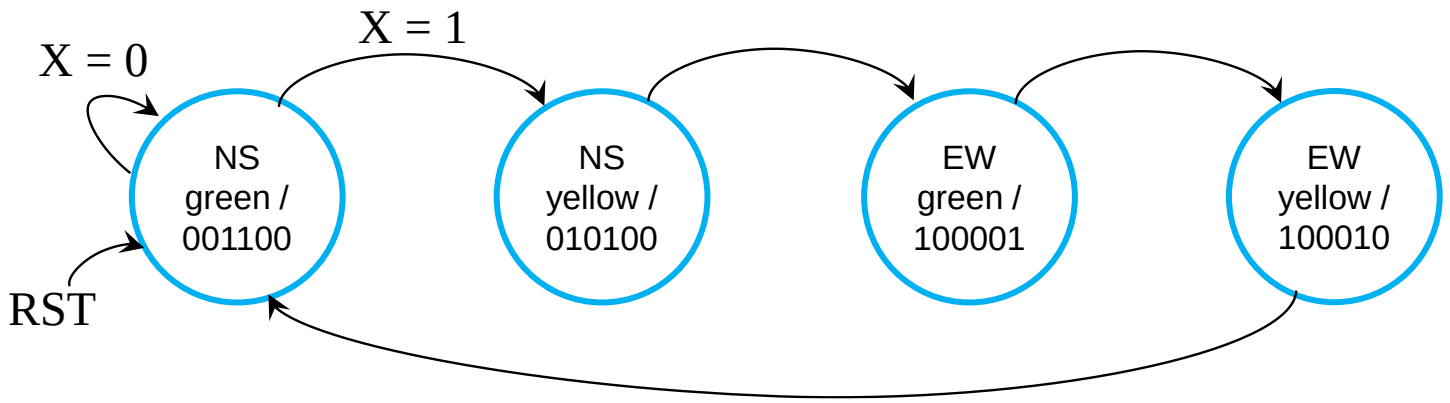
```
generate_clock: process
begin
    clock <= '1';
    wait for PERIOD/2;
    clock <= '0';
    wait for PERIOD/2;
end process;
```

Generates a clock with period PERIOD: the process runs continuously

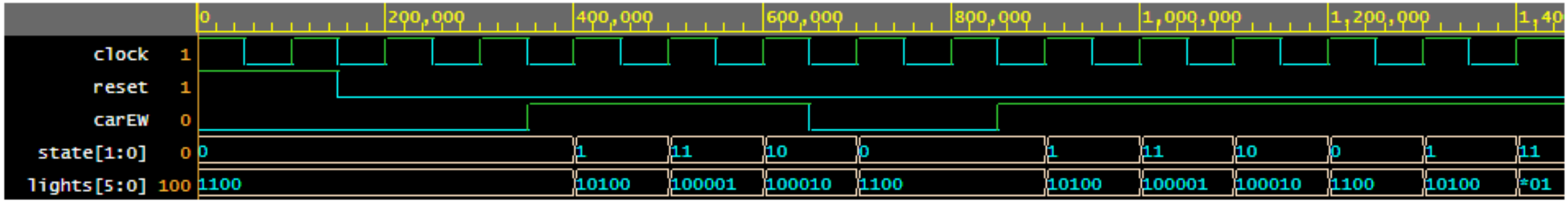
Traffic Light Controller: Testbench (3/3)

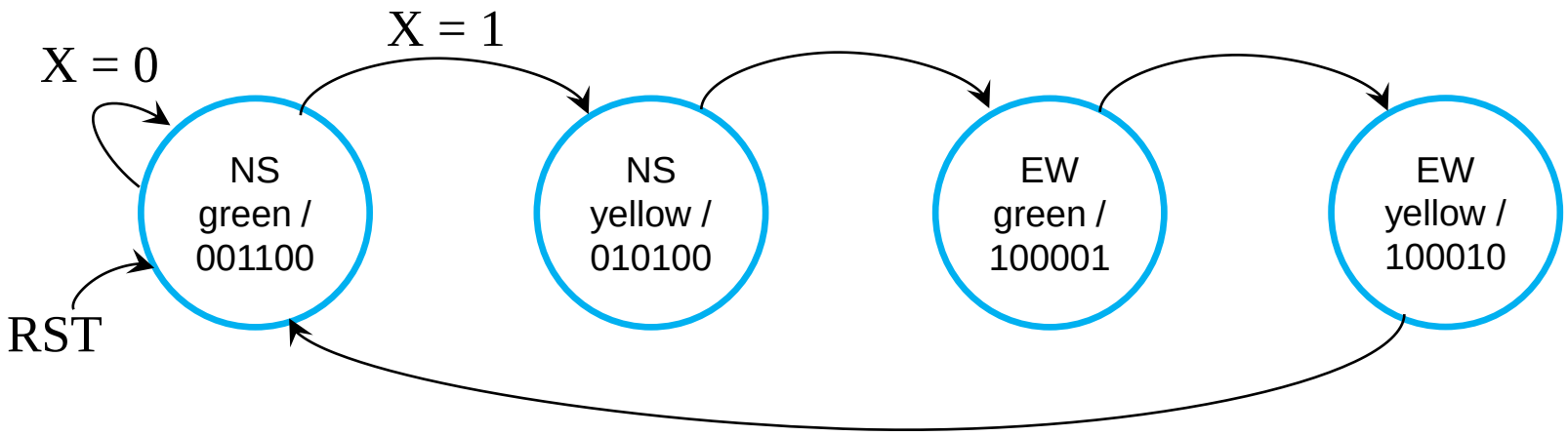
```
apply_inputs: process
begin
    reset <= '1';
    carEW <= '0';
    wait for 3*PERIOD/2;
    reset <= '0';
    wait for 2*PERIOD;
    carEW <= '1'; -- wait 2 clock cycles, then a car arrives
    wait for 3*PERIOD;
    carEW <= '0'; -- car leaves after 3 clock cycles
    wait for 2*PERIOD;
    carEW <= '1'; -- wait 2 clock cycles, then car comes and stays
    wait for 6*PERIOD;
    std.env.stop;
end process;

end test;
```

Applies inputs (at the falling edge of the clock)



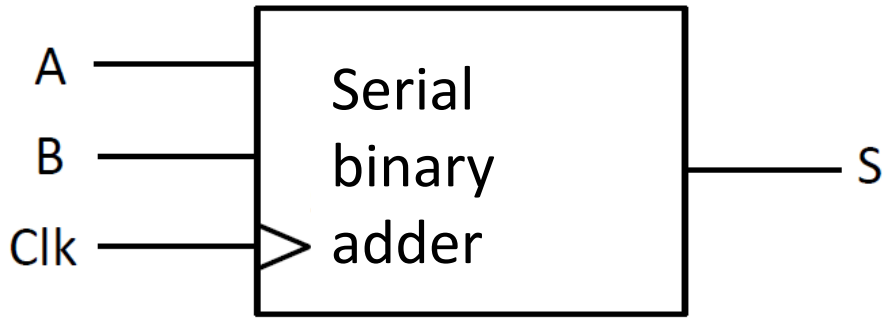
Traffic Light Controller: Simulation





Proposed Exercize

- Design a sequential circuit which performs the serial sum of two binary numbers. The two numbers to be added are provided one bit at a time (starting from the least significant bit) and the sum is performed one bit at a time, starting from the least significant bit. Use Positive-edge-triggered D-type flip flops. Highlight the following steps:
  - Find the state diagram
  - Is it a Mealy or a Moore machine?
  - Find the state and output table
  - Minimize the functions to calculate future state and output
  - Draw the circuit



Example:

0 1 1 0 1 + ← A
0 0 1 1 0 = ← B
1 0 0 1 1 ← S

Disclaimer

Figures from*Logic and Computer Design Fundamentals*, Fifth Edition, GE Mano |Kime| Martin