CSC34300
Lecture 05: Simple FSM Implementation

Instructor: Zheng Peng

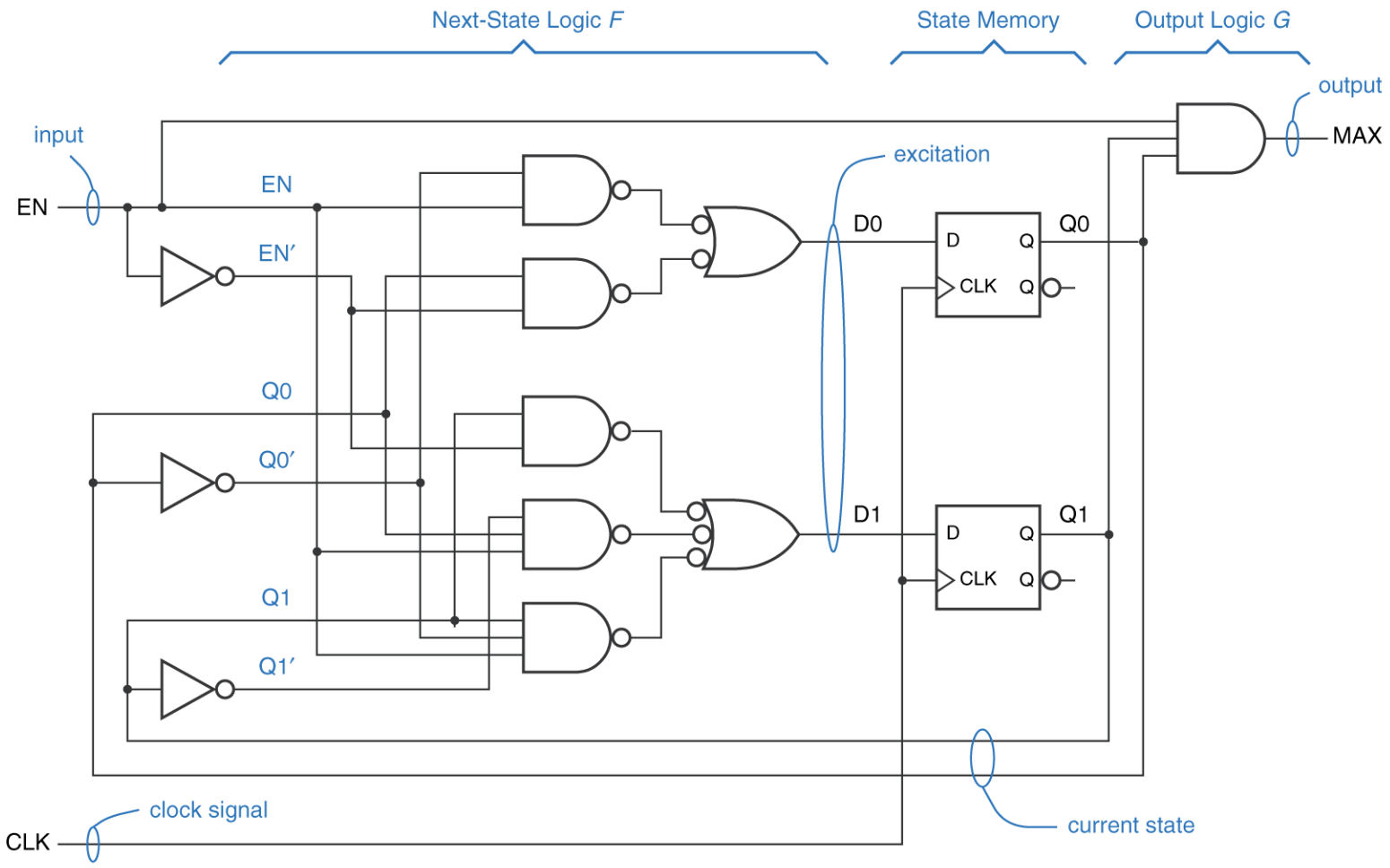
Computer Science Department

City College of New York

Introduction

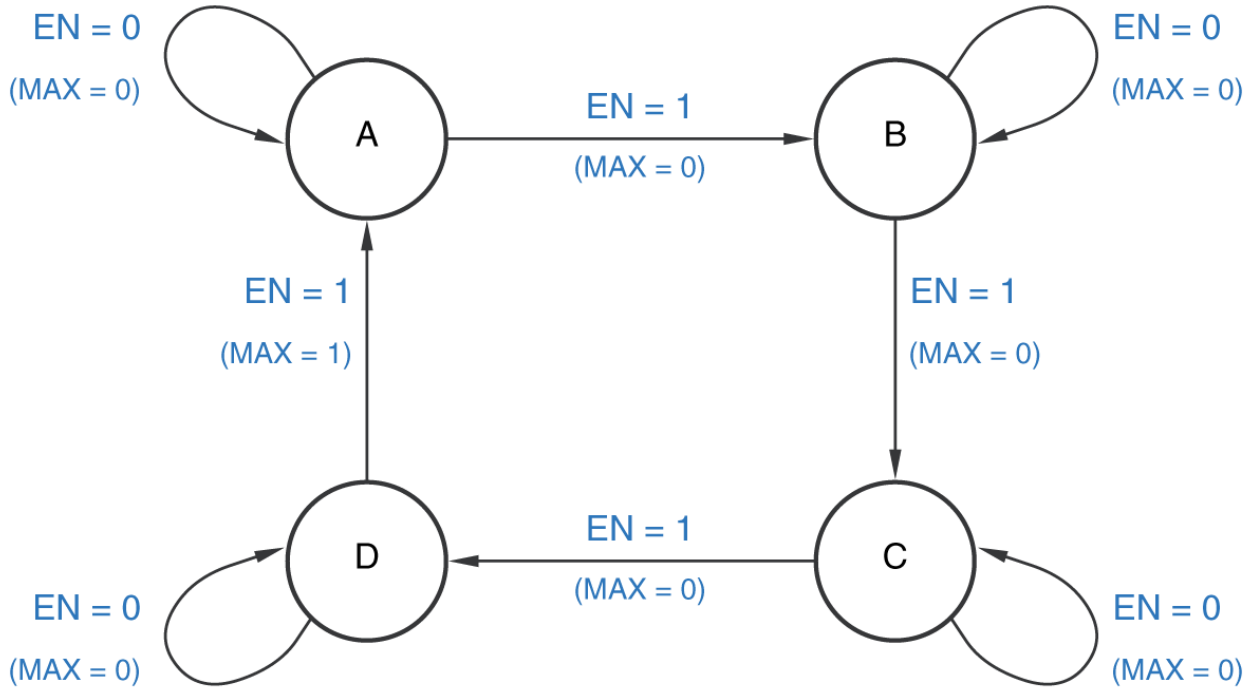
- FSM: Finite State Machine
  - The term “finite” refers to the fact that the number of states the circuit can display is finite
  - Sequential Circuit
    - Output depends not only on current input but also on past input values
    - Store information between operations
    - Outputs from the system are usually taken as new inputs (usually with delay), or “feedbacks”

A State Machine with Two Positive-edge-triggered D Flip-Flop



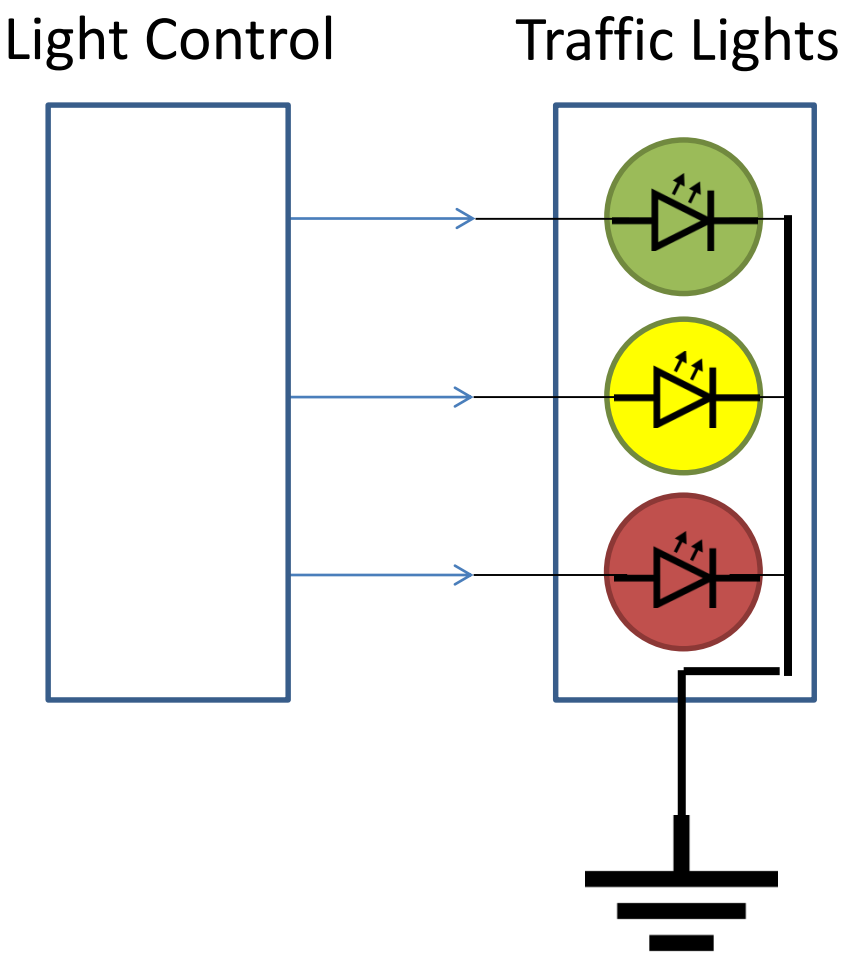
A State Machine with Two Positive-edge-triggered D Flip-Flop

- The corresponding state diagram



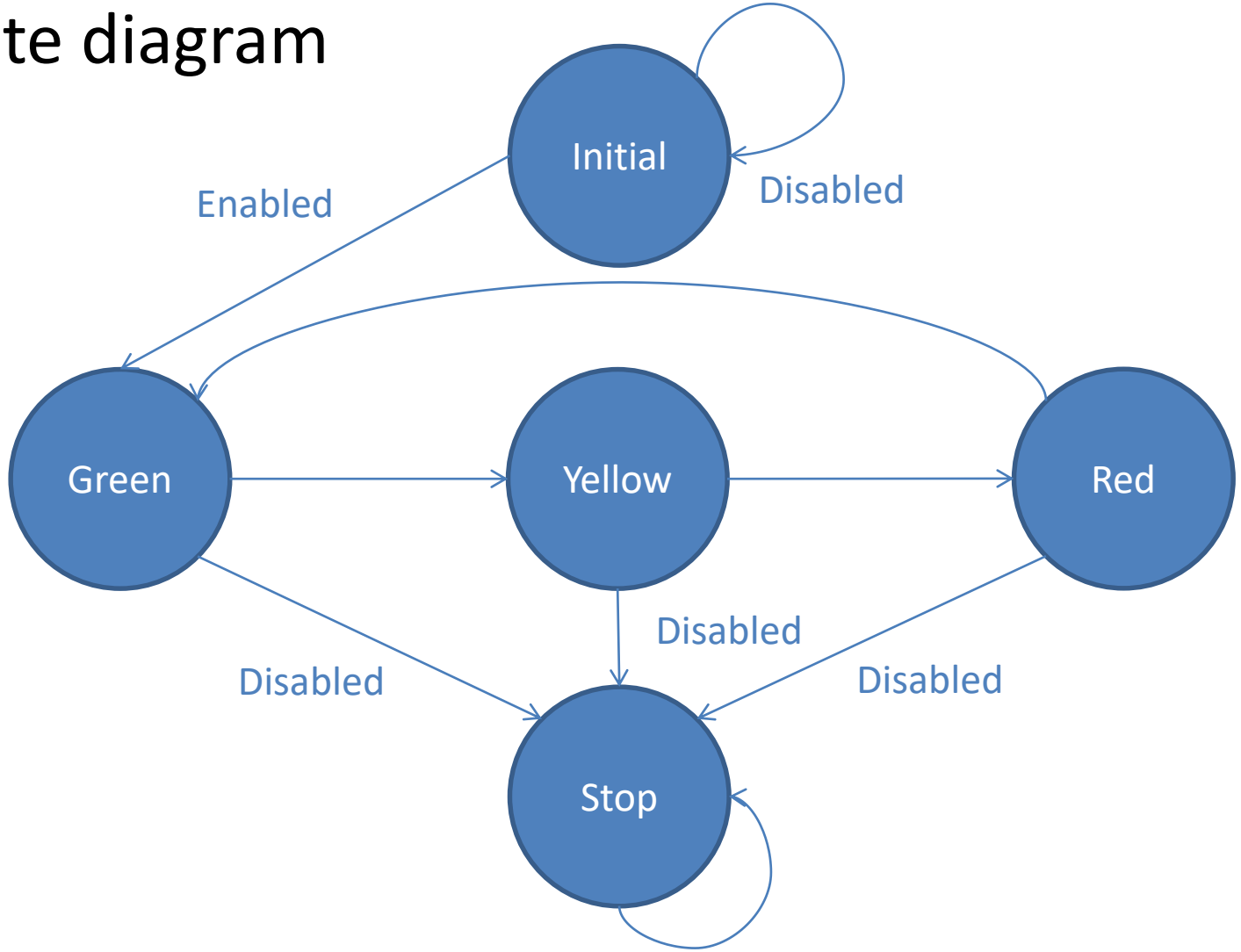
	EN	
S	***0***	***1***
A	A, 0	B, 0
B	B, 0	C, 0
C	C, 0	D, 0
D	D, 0	A, 1
	S∗, MAX	

VHDL FSM Implementation Example: Traffic Lights Control



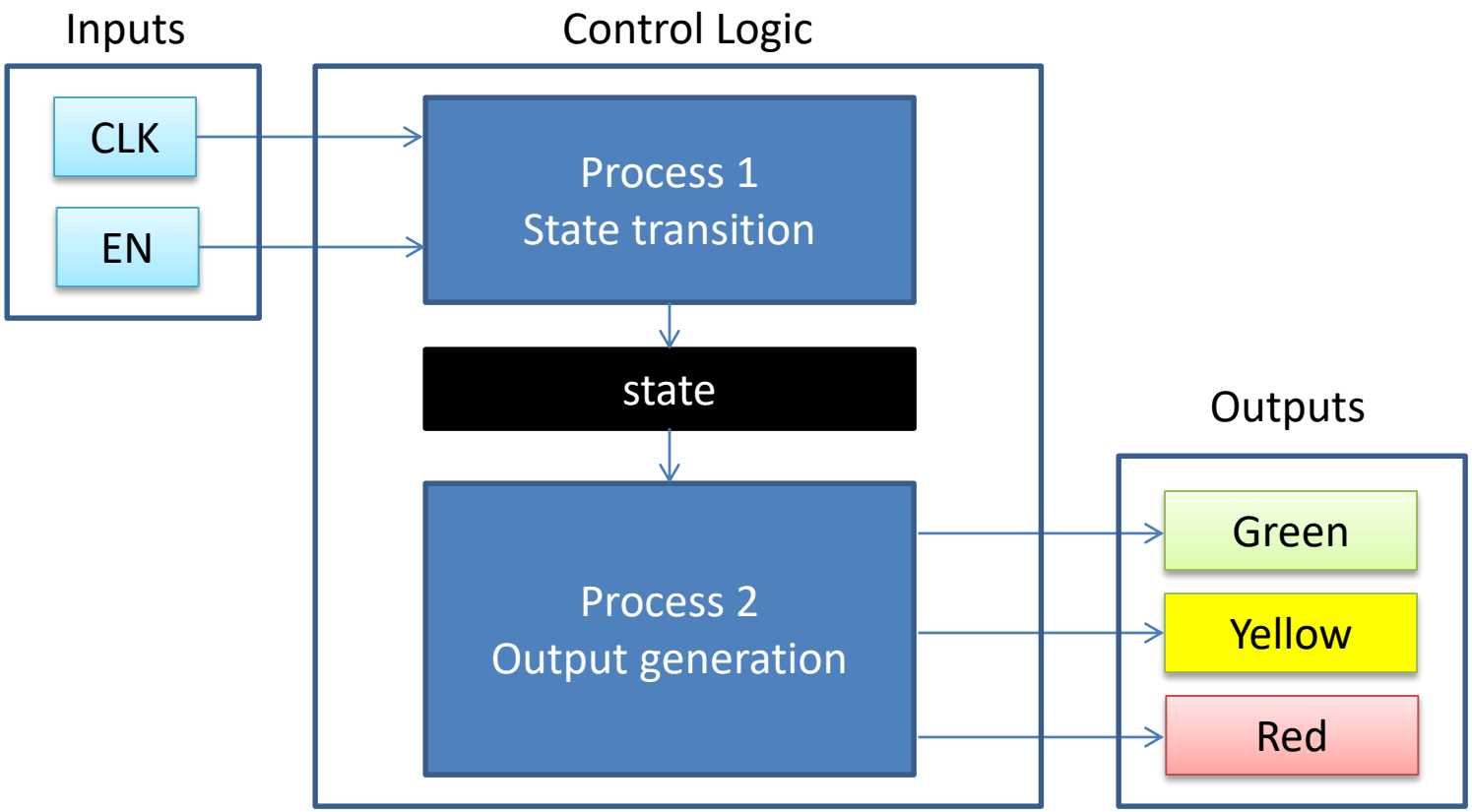
VHDL FSM Implementation Example: Traffic Lights Control

- State diagram



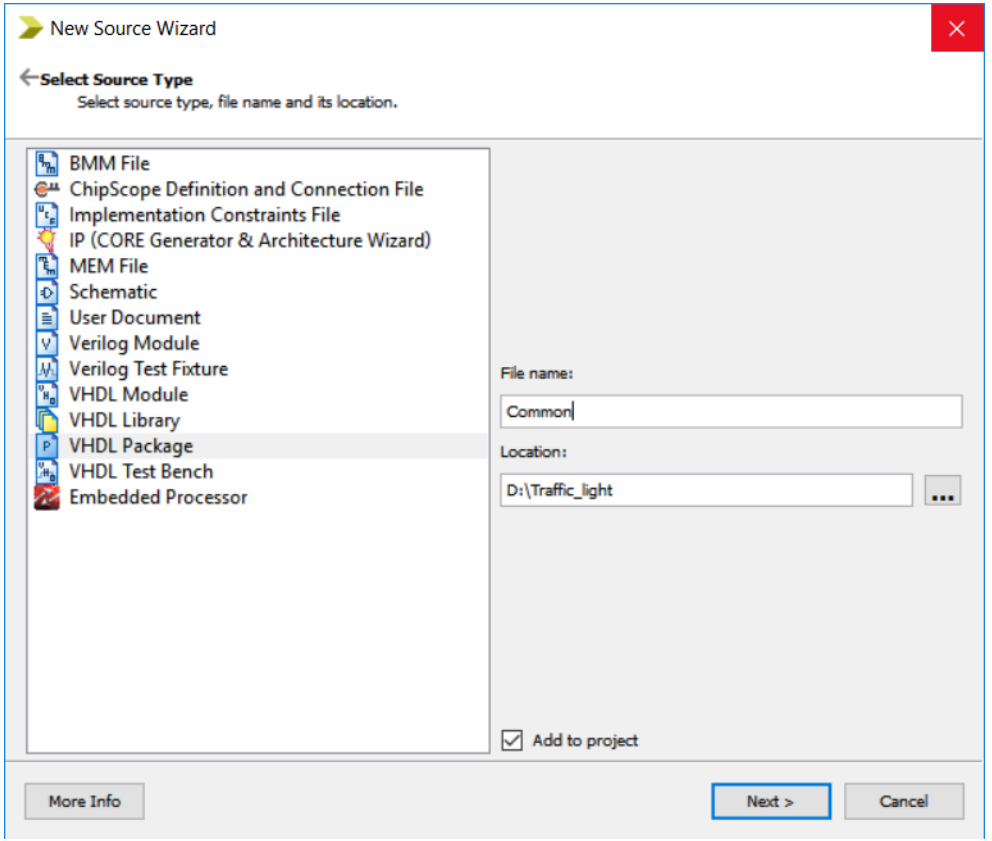
VHDL FSM Implementation Example: Traffic Lights Control

- Module design



VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation
  - Create a Package for user defined types, constants and functions



VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation
  - Create a Package for user defined types, constants and functions
  - traffic_light_state
    - S0: Initial
    - S1: Green
    - S2: Yellow
    - S3: Red
    - S4: Stop

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package Common is
type traffic_light_state is (S0, S1, S2, S3, S4);
end Common;

package body Common is
end Common;
```

VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation
  - Create a Package for user defined types, constants and functions
  - Use the package

```
use work.Common.all;
```

  - Entity declaration

```
entity LightControl is
    Port (  I_EN     : in  STD_LOGIC;
            I_CLK    : in  STD_LOGIC;
            O_GREEN  : out STD_LOGIC;
            O_YELLOW : out STD_LOGIC;
            O_RED    : out STD_LOGIC);
end LightControl;
```

VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation
  - Architecture definition

```
architecture Behavioral of LightControl is
   signal state : traffic_light_state := S0;
begin
   process(I_CLK)
   begin
      if rising_edge(I_CLK) then
         if I_EN = '1' then
            if state = S0 then state <= S1;
            elsif state = S1 then state <= S2;
            elsif state = S2 then state <= S3;
            elsif state = S3 then state <= S1;
            end if;
         else
            if state = S0 then state <= S0;
            else state <= S4;
            end if;
         end if;
      end if;
   end process;

   process(state)
   begin
      if state = S1 then
         O_GREEN <= '1'; O_YELLOW <= '0'; O_RED <= '0';
      elsif state = S2 then
         O_GREEN <= '0'; O_YELLOW <= '1'; O_RED <= '0';
      elsif state = S3 then
         O_GREEN <= '0'; O_YELLOW <= '0'; O_RED <= '1';
      else
         O_GREEN <= '0'; O_YELLOW <= '0'; O_RED <= '0';
      end if;
   end process;
end Behavioral;
```

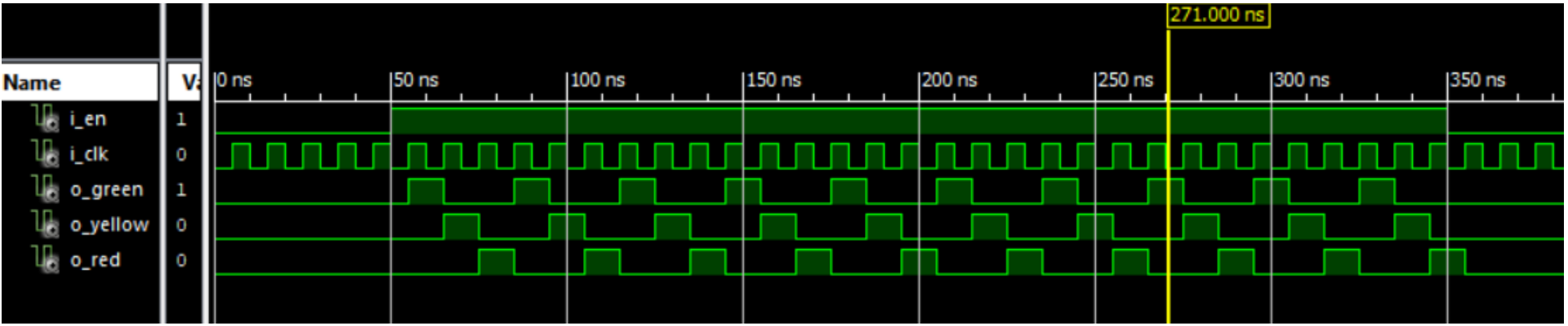
VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation: Testbench program

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

use work.Common.all;

ENTITY LightControl_test IS
END LightControl_test;

ARCHITECTURE behavior OF LightControl_test IS
   COMPONENT LightControl
   PORT(
         I_EN : IN  std_logic;
         I_CLK : IN  std_logic;
         O_GREEN : OUT  std_logic;
         O_YELLOW : OUT  std_logic;
         O_RED : OUT  std_logic
        );
   END COMPONENT;
   signal I_EN : std_logic := '0';
   signal I_CLK : std_logic := '0';
   --Outputs
   signal O_GREEN : std_logic;
   signal O_YELLOW : std_logic;
   signal O_RED : std_logic;
   -- Clock period definitions
   constant I_CLK_period : time := 10 ns;
BEGIN
   -- Instantiate the Unit Under Test (UUT)
   uut: LightControl PORT MAP (
          I_EN => I_EN,
          I_CLK => I_CLK,
          O_GREEN => O_GREEN,
          O_YELLOW => O_YELLOW,
          O_RED => O_RED
        );
   -- Clock process definitions
   I_CLK_process :process
   begin
      I_CLK <= '0';
      wait for I_CLK_period/2;
      I_CLK <= '1';
      wait for I_CLK_period/2;
   end process;
   -- Stimulus process
   stim_proc: process
   begin
      -- hold init state for 50 ns.
      wait for 50 ns;
      -- insert stimulus here
      I_EN <= '1'; wait for 300 ns;
      I_EN <= '0'; wait;
   end process;
END;
```

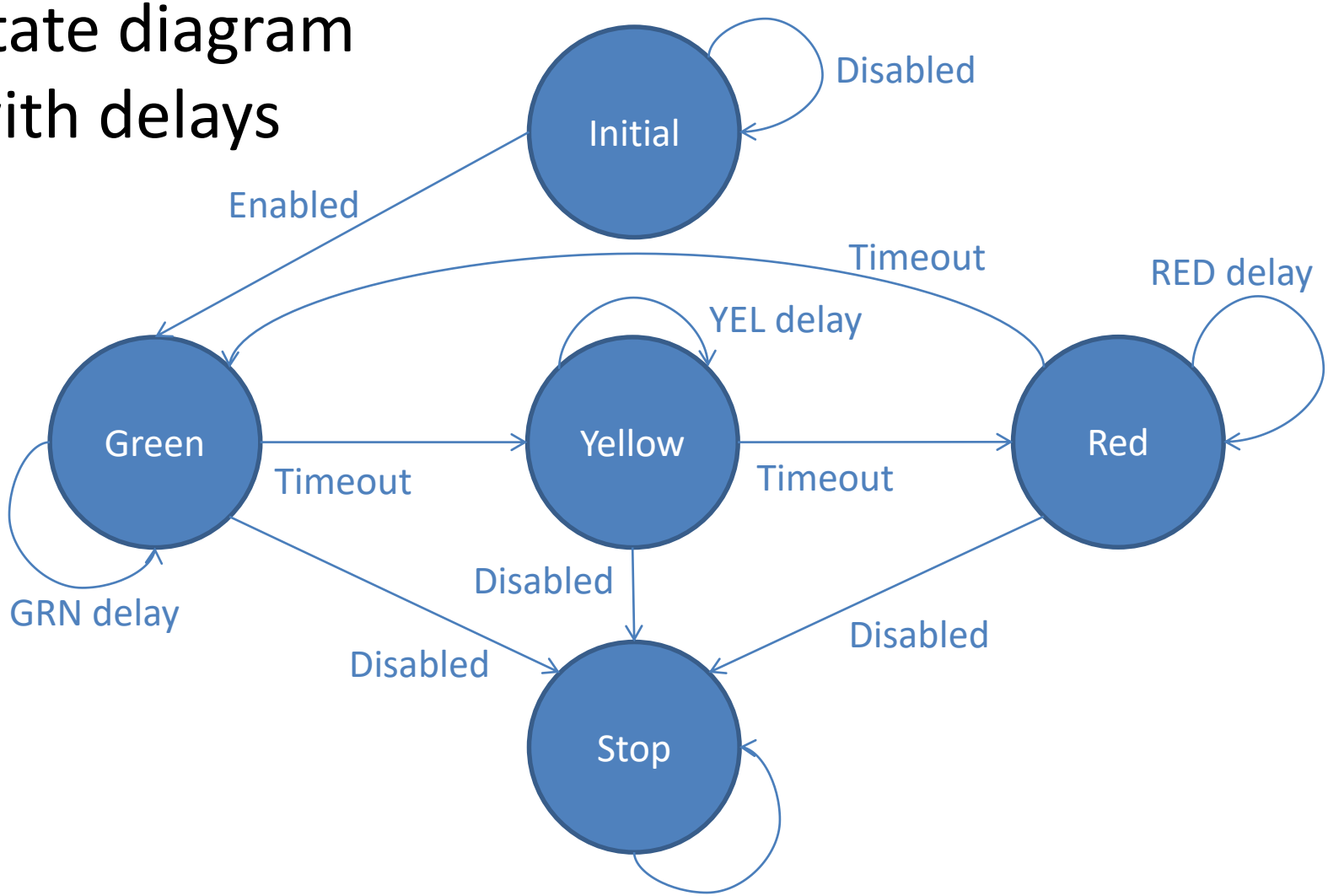
VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation:
  - Simulation and waveforms



VHDL FSM Implementation Example: Traffic Lights Control

- State diagram with delays



VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation (with delay)
  - Create a Package for user defined types, constants and functions
  - traffic_light_state
    - S0: Initial
    - S1: Green
    - S2: Yellow
    - S3: Red
    - S4: Stop

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

package Common is
type traffic_light_state is (S0, S1, S2, S3, S4);
constant GRN_WAIT_CYCLES : integer := 5;
constant YEL_WAIT_CYCLES : integer := 2;
constant RED_WAIT_CYCLES : integer := 5;
end Common;

package body Common is
end Common;
```

VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation (with delay)
  - Architecture definition (page 1)

```
15  architecture Behavioral of LightControlDelay is
16     signal state : traffic_light_state := S0;
17     signal counter_grn : integer := GRN_WAIT_CYCLES;
18     signal counter_yel : integer := YEL_WAIT_CYCLES;
19     signal counter_red : integer := RED_WAIT_CYCLES;
20  begin
21     process(I_CLK)
22     begin
23        if rising_edge(I_CLK) then
24           if I_EN = '1' then
25              if state = S0 then
26                 state <= S1;
27              elsif state = S1 then
28                 if counter_grn > 0 then
29                    counter_grn <= counter_grn-1;
30                 else
31                    state <= S2;
32                    counter_grn <= GRN_WAIT_CYCLES;
33                 end if;
34              elsif state = S2 then
35                 if counter_yel > 0 then
36                    counter_yel <= counter_yel-1;
37                 else
38                    state <= S3;
39                    counter_yel <= YEL_WAIT_CYCLES;
40                 end if;
41              elsif state = S3 then
42                 if counter_red > 0 then
43                    counter_red <= counter_red-1;
44                 else
45                    state <= S1;
46                    counter_red <= RED_WAIT_CYCLES;
47                 end if;
48              end if;
49           else
50              if state = S0 then state <= S0;
51              else state <= S4;
52              end if;
53           end if;
54        end if;
55     end process;
```

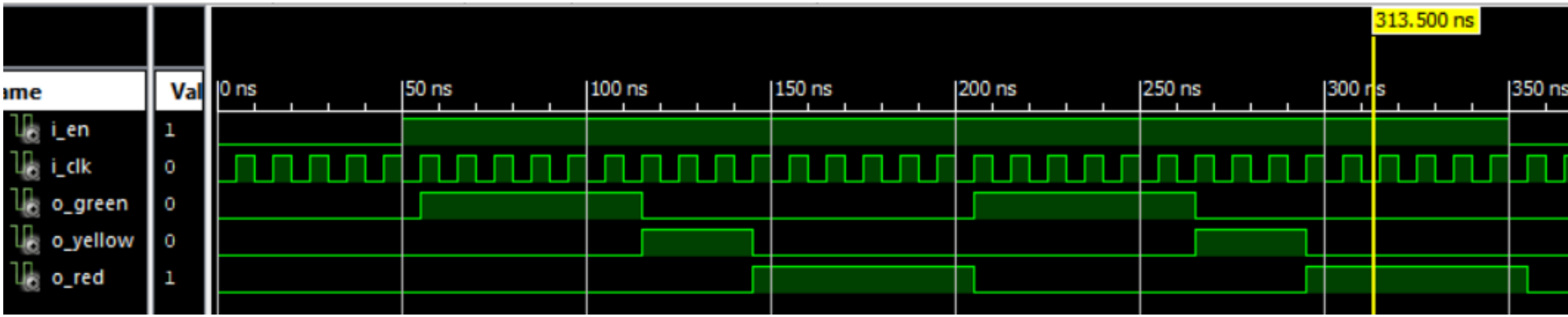
VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation (with delay)
  - Architecture definition (page 2)

```
56
57      process(state)
58      begin
59          if state = S1 then
60              O_GREEN <= '1'; O_YELLOW <= '0'; O_RED <= '0';
61          elsif state = S2 then
62              O_GREEN <= '0'; O_YELLOW <= '1'; O_RED <= '0';
63          elsif state = S3 then
64              O_GREEN <= '0'; O_YELLOW <= '0'; O_RED <= '1';
65          else
66              O_GREEN <= '0'; O_YELLOW <= '0'; O_RED <= '0';
67          end if;
68      end process;
69  end Behavioral;
```

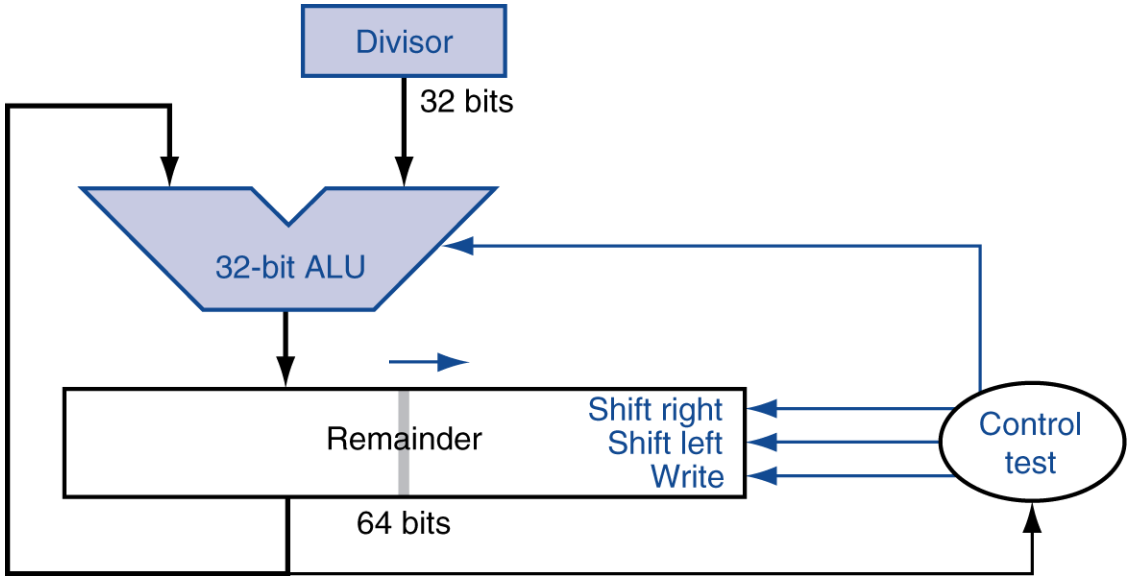
VHDL FSM Implementation Example: Traffic Lights Control

- VHDL implementation (with delay)
  - Simulation and waveforms



Homework

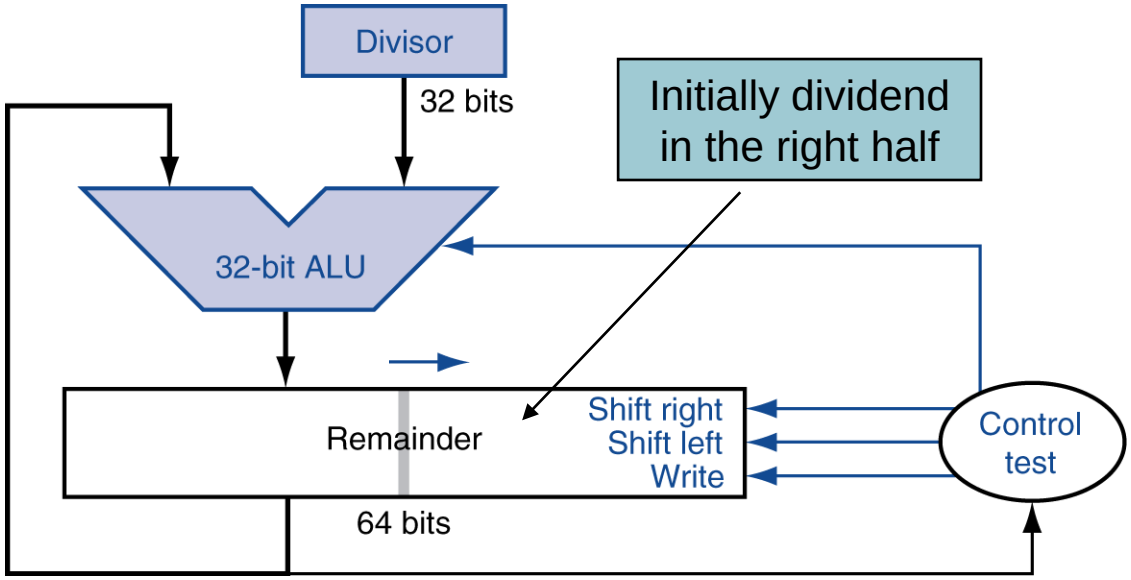
- Optimized Divider



  - N-bit division involves N steps
  - Each step, the circuit is in one of the following states
    - Initial state
    - Shift state
    - Calculation state
    - Remainder update state
    - Stop

Homework

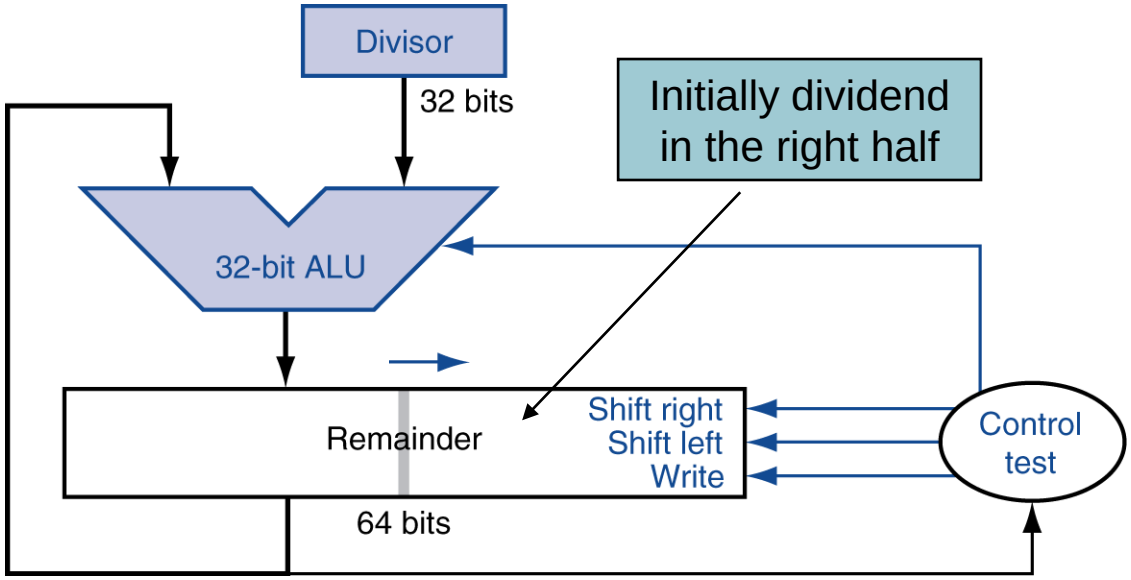
- Optimized Divider



  - The remainder register is double sized
    - Shift to left by 1 bit in each iteration
    - Only the higher half participates in subtraction
    - New quotient bit is shifted in from the right end
  - In the end, the higher half is the remainder and the lower half is the quotient

Homework

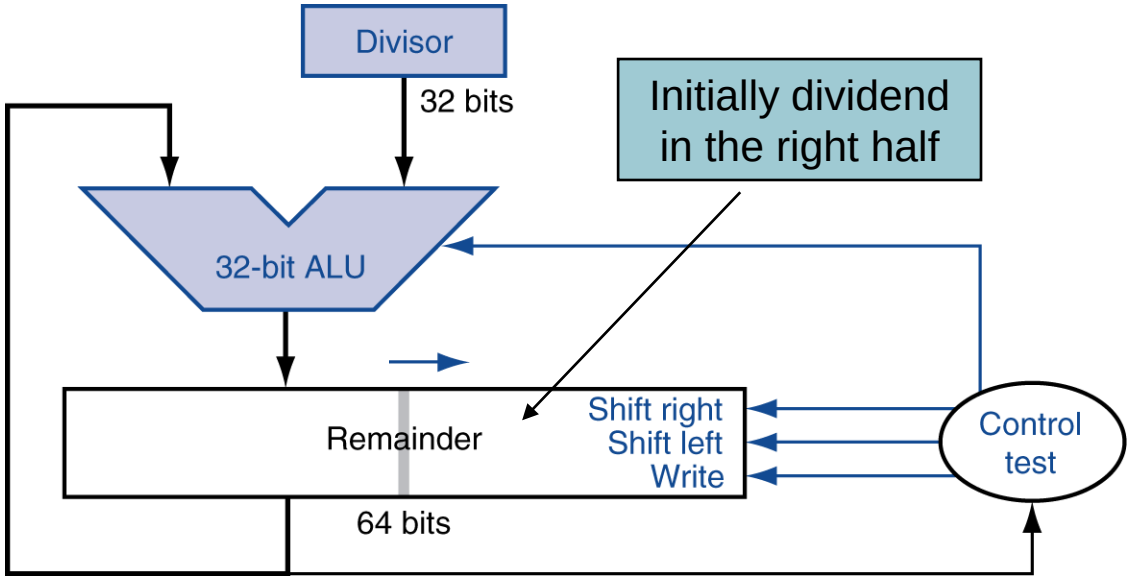
- Optimized Divider



  - The ALU
    - Performs subtractions when enabled
    - Can determine which operand is the bigger one
    - Controlled by the control logic

Homework

- Optimized Divider



  - The Control Unit
    - Handles state transition
    - Generate control signals
    - Start and stop the calculation steps