Integrated Circuits Design by FPGA

م.م. أحمد مؤيد عبدالحسين

جامعة الفرات الأوسط التقنية / الكلية التقنية الهندسية / نجف

Lecture 9

Finite State Machines (FSM)

Traffic Light Controller (TLC)

Objectives of this Lecture

- To implement the example of **Traffic Light Controller (TLC),** using **FSM** method by FPGA technique.

- To understand the **FSM** method.

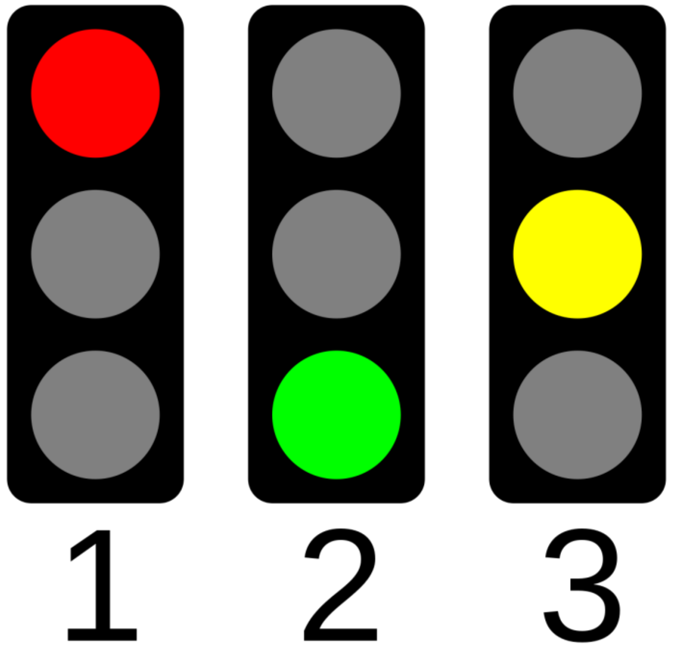
Contents of this Lecture

- **FSM** introduction. (Review)

- **TLC** example

FSM introduction

- **FSM** can be very helpful in the design of certain types of systems, particularly those whose tasks form a well-defined sequence (digital controllers, for example).



FSM introduction

- Figure 8.1 shows the block diagram of a single-phase state machine. As indicated in the figure, the lower section contains the sequential logic (flip-flops), while the upper section contains the combinational logic (parallel VHDL).

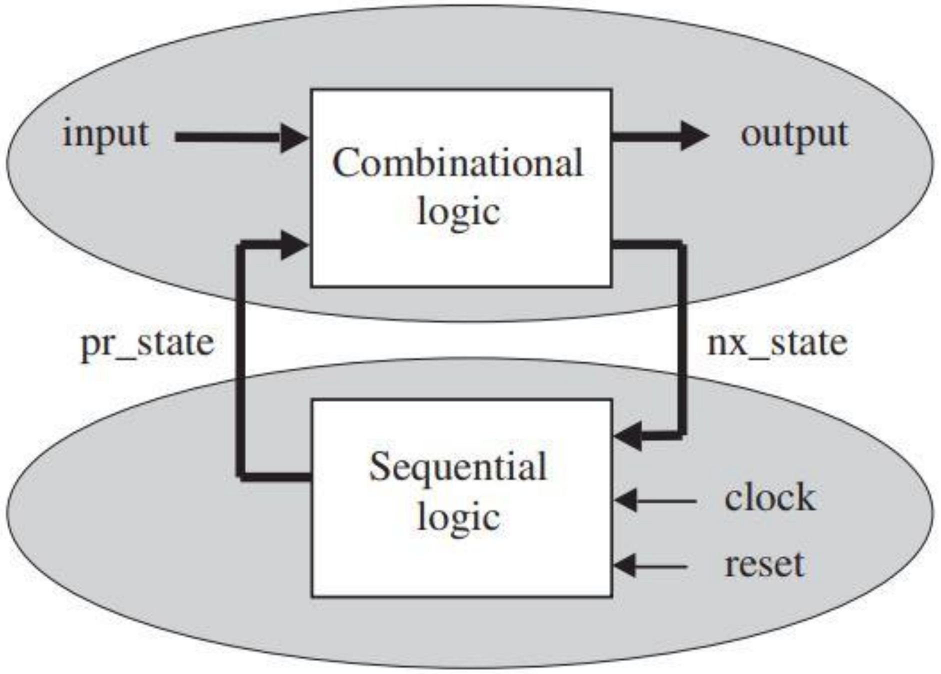


Figure 8.1
Mealy (Moore) state machine diagram.

FSM introduction

- The combinational (upper) section has two inputs, being one **pr_state** (present state) and the other the external input proper. It has also two outputs, **nx_state** (next state) and the external output proper.

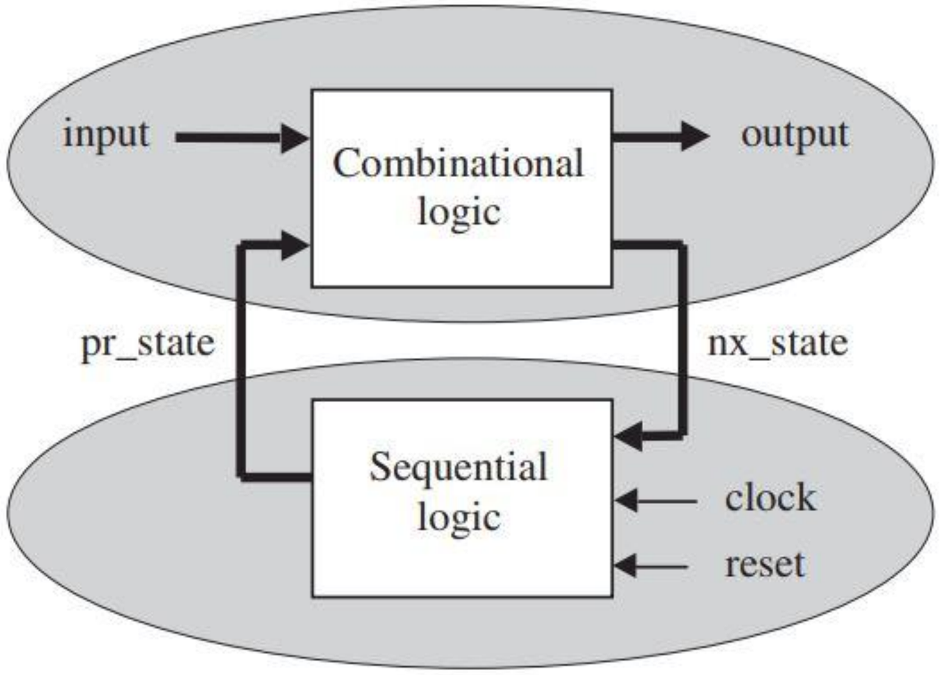


Figure 8.1
Mealy (Moore) state machine diagram.

FSM introduction

- The sequential (lower) section has three inputs (clock, reset, and nx_state), and one output (pr_state). Since all flip-flops are in this part of the system, clock and reset must be connected to it.

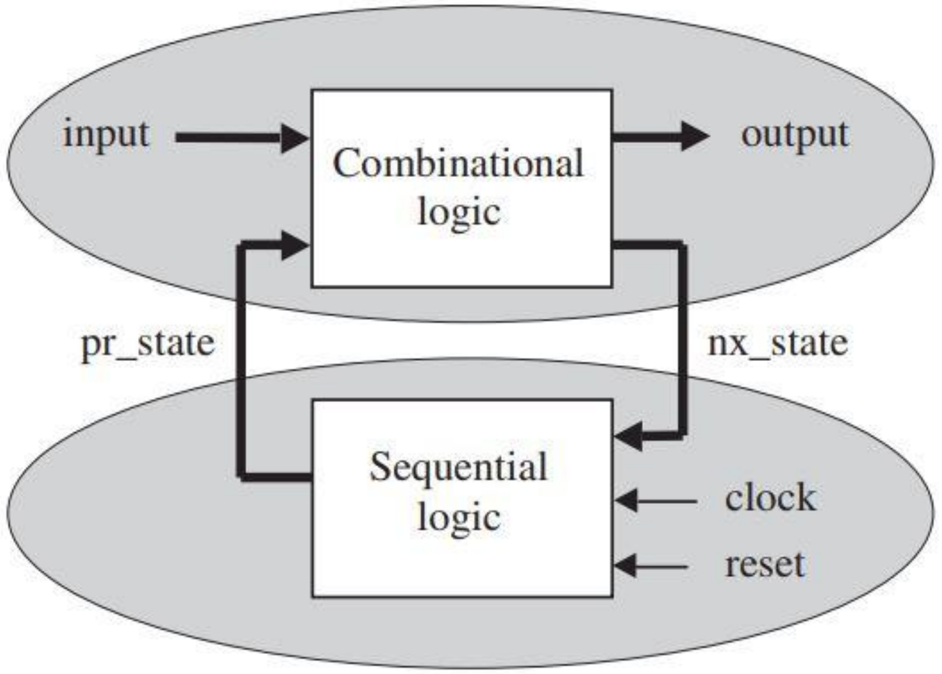


Figure 8.1
Mealy (Moore) state machine diagram.

FSM introduction

- From a VHDL perspective, it is clear that the lower part, being sequential, will require a PROCESS, while the upper part, being combinational, will not.
- However, it is also possible to use PROCESS inside the upper part to implement combinational (parallel) design.

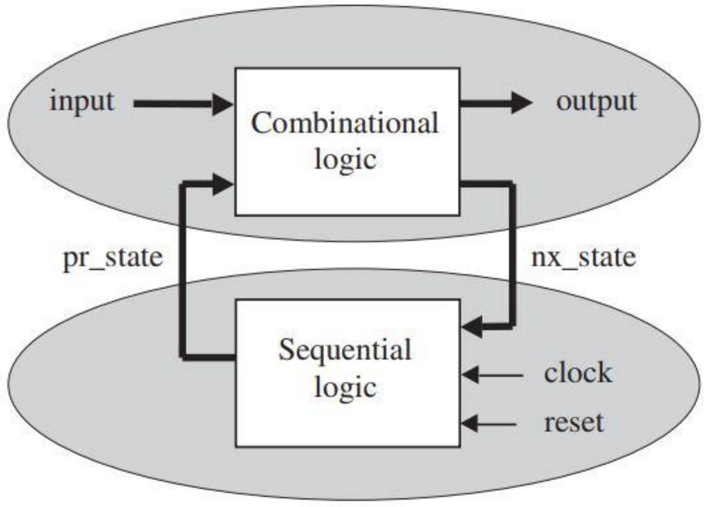


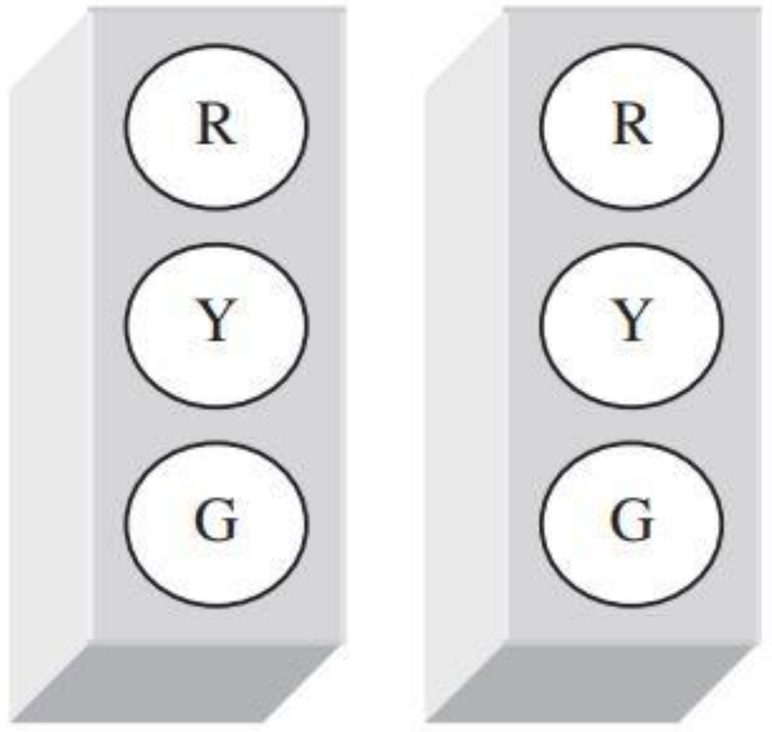
Figure 8.1
Mealy (Moore) state machine diagram.

FSM introduction

- One important aspect related to the **FSM** approach is that, though any sequential circuit can in principle be modeled as a state machine, this is not always advantageous. The reason is that the code might become longer, more complex, and more error prone than in a conventional approach. This is often the case with simple registered circuits, like counters.

- The **FSM** approach is advisable in systems whose tasks constitute a well - structured list so all states <u>can be easily enumerated</u>. That is, in a typical state machine implementation, we will encounter, at the beginning of the ARCHITECTURE, a user-defined enumerated data type, containing <u>a list of all possible system states.</u>

TLC Example



State	Operation Mode		
	REGULAR	TEST	STANDBY
	Time	Time	Time
RG	timeRG (30s)	timeTEST (1s)	---
RY	timeRY (5s)	timeTEST (1s)	---
GR	timeGR (45s)	timeTEST (1s)	---
YR	timeYR (5s)	timeTEST (1s)	---
YY	---	---	Indefinite

TLC Example

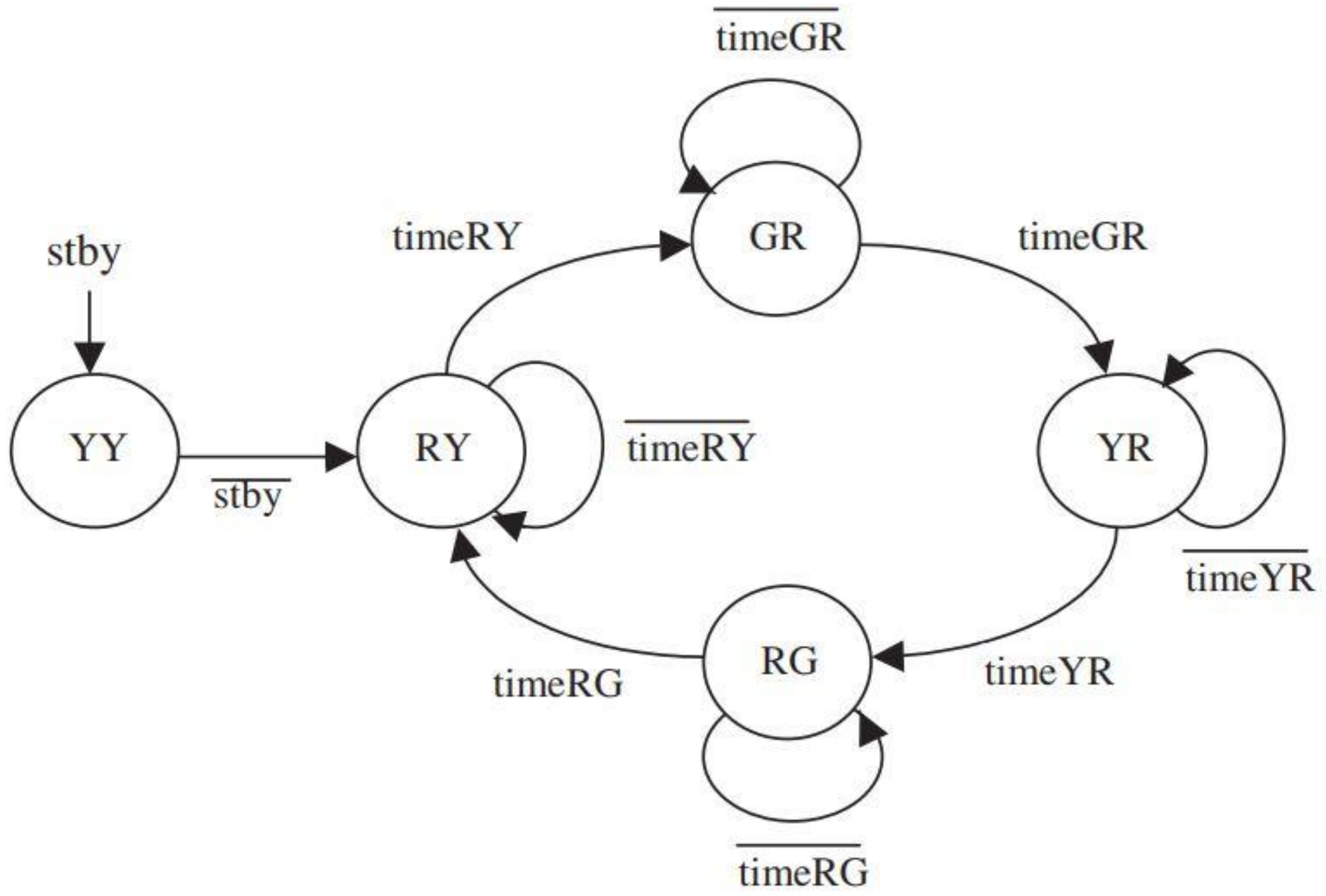
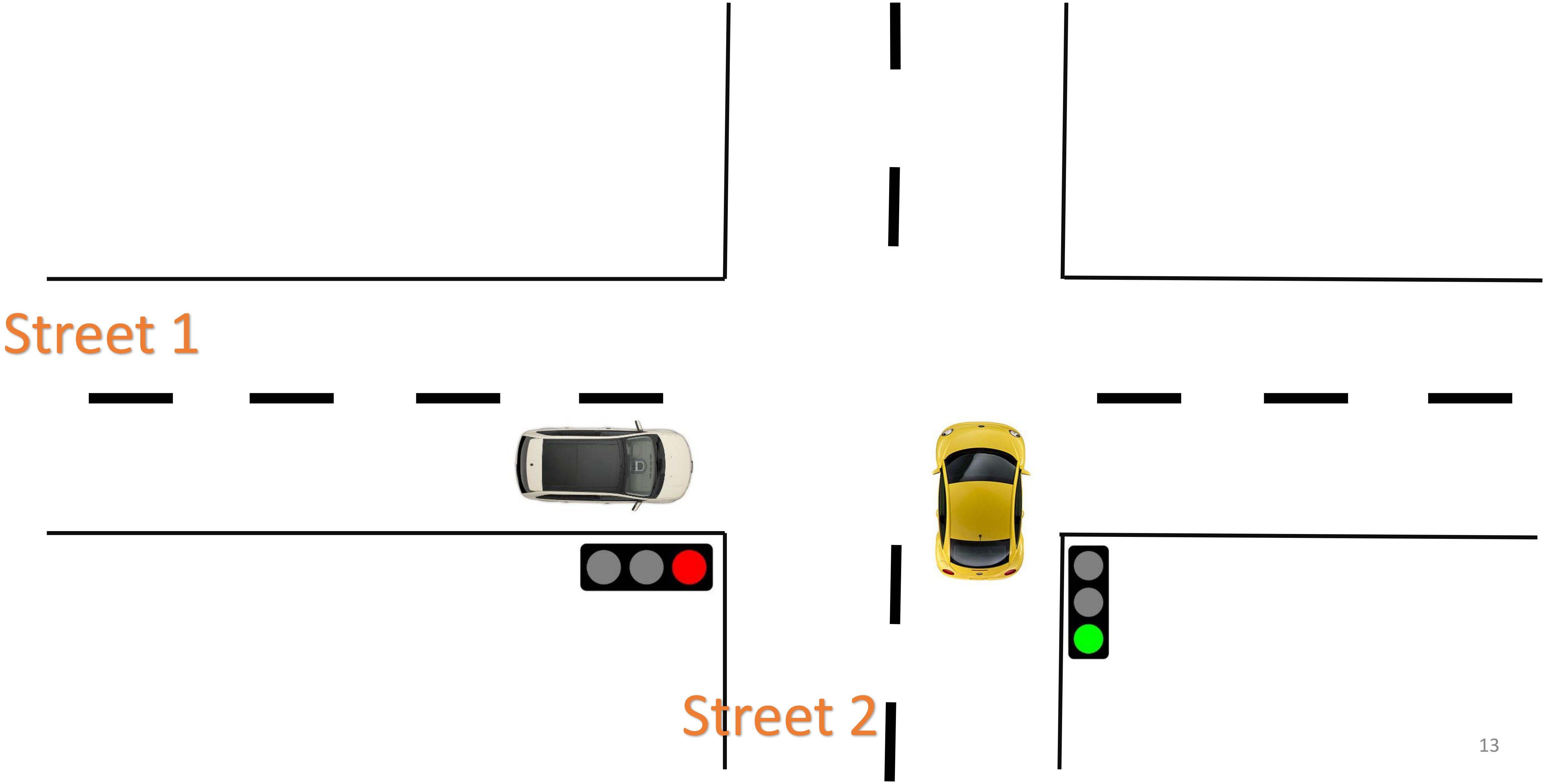
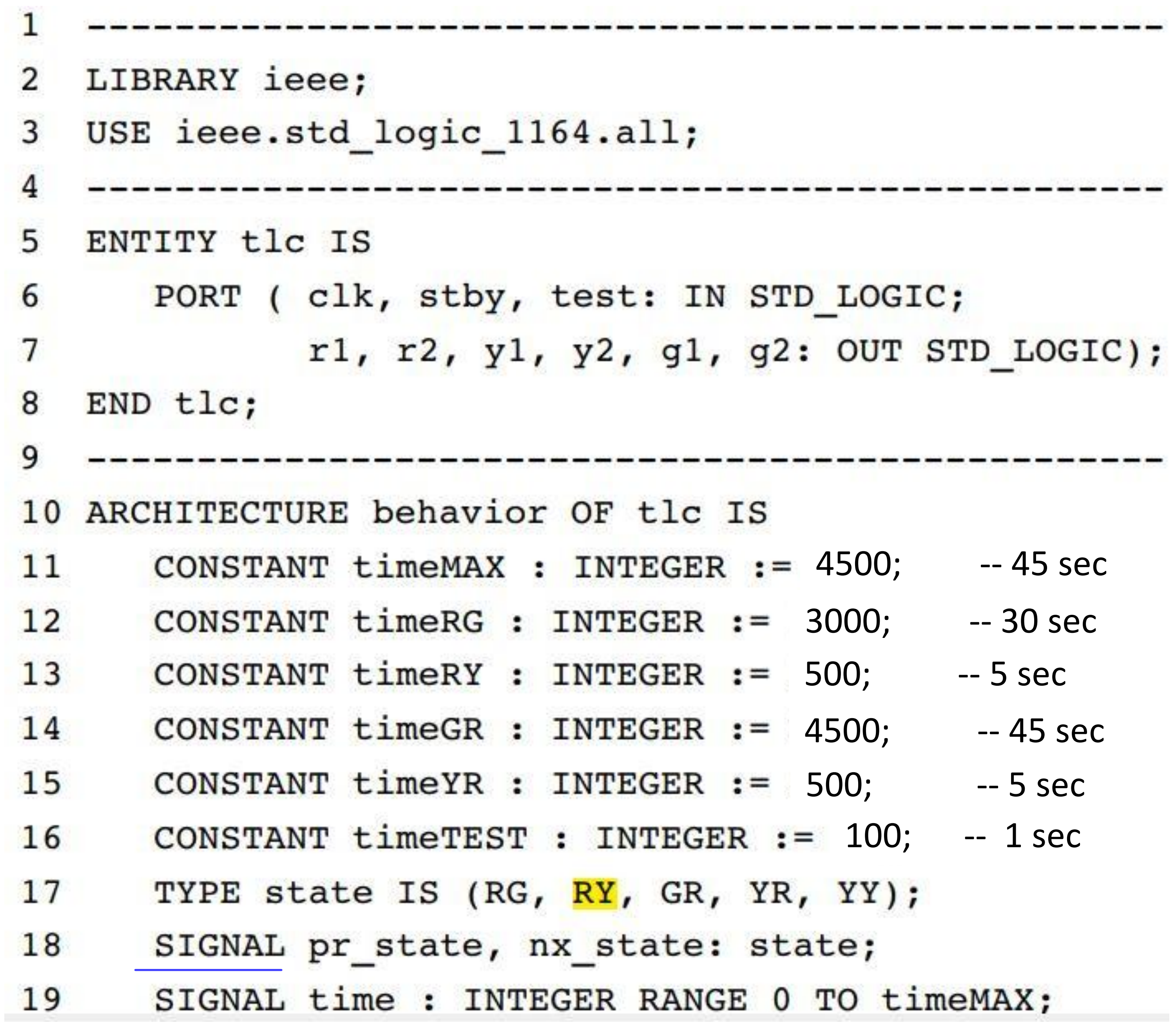

stby
YY
$\overline{\text{stby}}$
RY
$\overline{\text{timeRY}}$
timeRY
GR
$\overline{\text{timeGR}}$
timeGR
YR
$\overline{\text{timeYR}}$
timeYR
RG
$\overline{\text{timeRG}}$
timeRG

Figure 8.10
Specifications and states diagram (regular mode) for example 8.5.

TLC Example



TLC Example

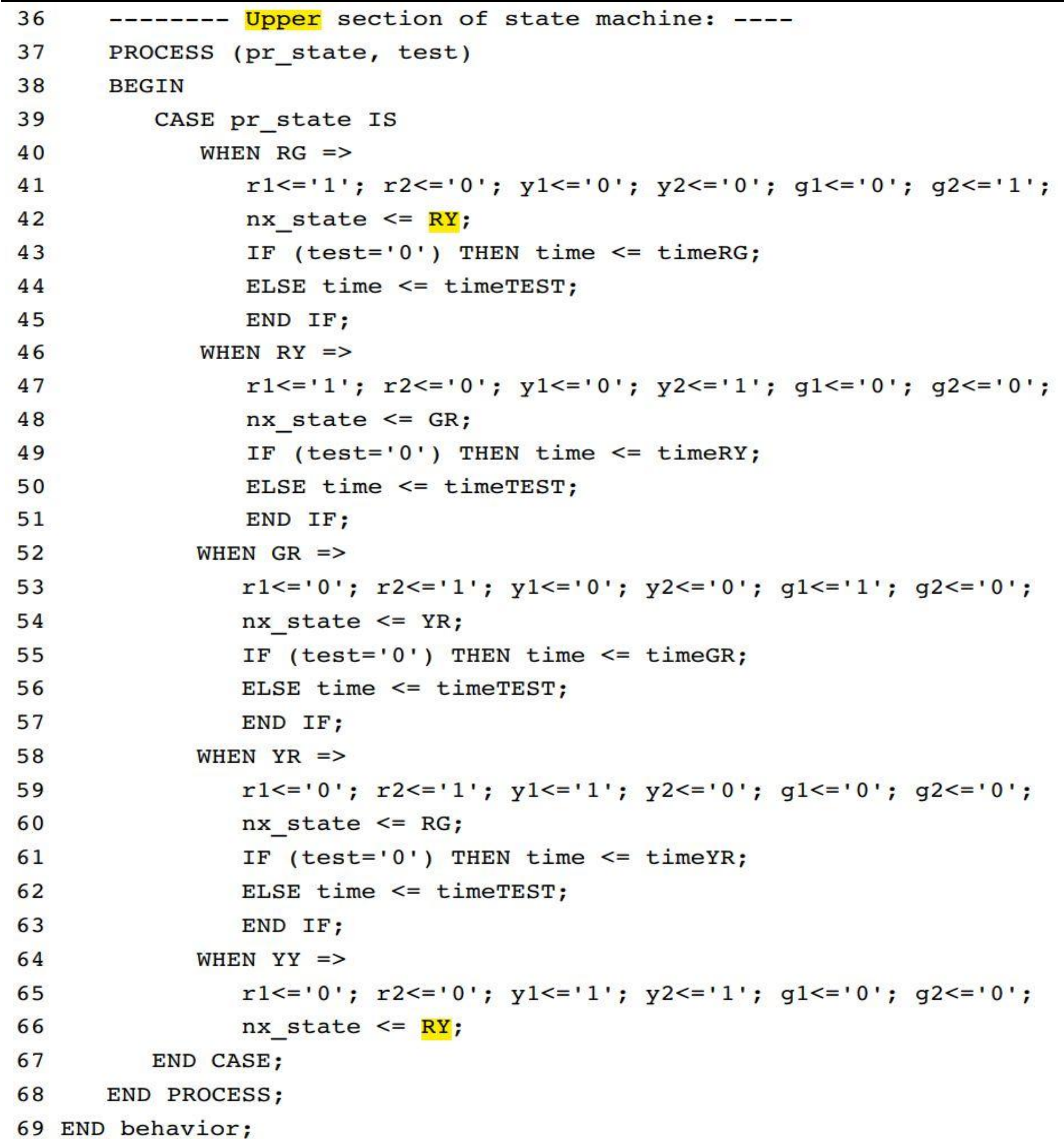


```
----------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
----------------------------------------
ENTITY tlc IS
   PORT ( clk, stby, test: IN STD_LOGIC;
          r1, r2, y1, y2, g1, g2: OUT STD_LOGIC);
END tlc;
----------------------------------------
ARCHITECTURE behavior OF tlc IS
   CONSTANT timeMAX : INTEGER := 4500;     -- 45 sec
   CONSTANT timeRG : INTEGER := 3000;      -- 30 sec
   CONSTANT timeRY : INTEGER := 500;       -- 5 sec
   CONSTANT timeGR : INTEGER := 4500;      -- 45 sec
   CONSTANT timeYR : INTEGER := 500;       -- 5 sec
   CONSTANT timeTEST : INTEGER := 100;     -- 1 sec
   TYPE state IS (RG, RY, GR, YR, YY);
   SIGNAL pr_state, nx_state: state;
   SIGNAL time : INTEGER RANGE 0 TO timeMAX;
```

TLC Example

```
BEGIN
    -------- Lower section of state machine: ----
    PROCESS (clk, stby)
       VARIABLE count : INTEGER RANGE 0 TO timeMAX;
    BEGIN
       IF (stby='1') THEN
          pr_state <= YY;
          count := 0;
       ELSIF (clk'EVENT AND clk='1') THEN
          count := count + 1;
          IF (count = time) THEN
             pr_state <= nx_state;
             count := 0;
          END IF;
       END IF;
    END PROCESS;
```

TLC Example



```
    -------- Upper section of state machine: ----
    PROCESS (pr_state, test)
    BEGIN
       CASE pr_state IS
          WHEN RG =>
             r1<='1'; r2<='0'; y1<='0'; y2<='0'; g1<='0'; g2<='1';
             nx_state <= RY;
             IF (test='0') THEN time <= timeRG;
             ELSE time <= timeTEST;
             END IF;
          WHEN RY =>
             r1<='1'; r2<='0'; y1<='0'; y2<='1'; g1<='0'; g2<='0';
             nx_state <= GR;
             IF (test='0') THEN time <= timeRY;
             ELSE time <= timeTEST;
             END IF;
          WHEN GR =>
             r1<='0'; r2<='1'; y1<='0'; y2<='0'; g1<='1'; g2<='0';
             nx_state <= YR;
             IF (test='0') THEN time <= timeGR;
             ELSE time <= timeTEST;
             END IF;
          WHEN YR =>
             r1<='0'; r2<='1'; y1<='1'; y2<='0'; g1<='0'; g2<='0';
             nx_state <= RG;
             IF (test='0') THEN time <= timeYR;
             ELSE time <= timeTEST;
             END IF;
          WHEN YY =>
             r1<='0'; r2<='0'; y1<='1'; y2<='1'; g1<='0'; g2<='0';
             nx_state <= RY;
       END CASE;
    END PROCESS;
END behavior;
```

Assignments

- The assignments will be attached to your class room.

End of lecture 9

Any Questions ?