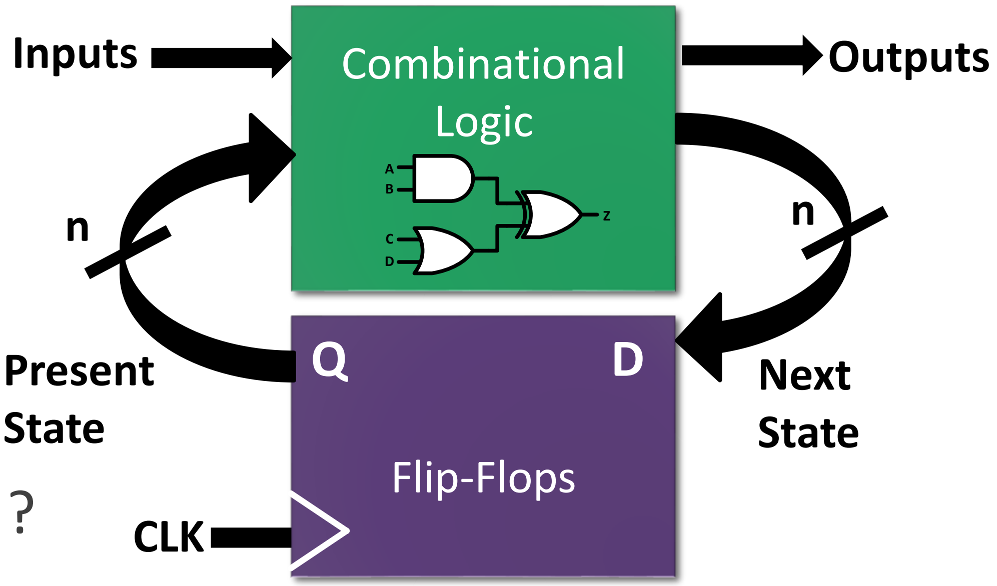
FSM 1 : Intro to Finite State Machines

What is a FSM?

- FSM : Finite State Machine
- Synchronous Machine with “states” of operation
- At each *active clock edge*, combinational logic computes **outputs** and **next state,** as a function of **inputs and present state**

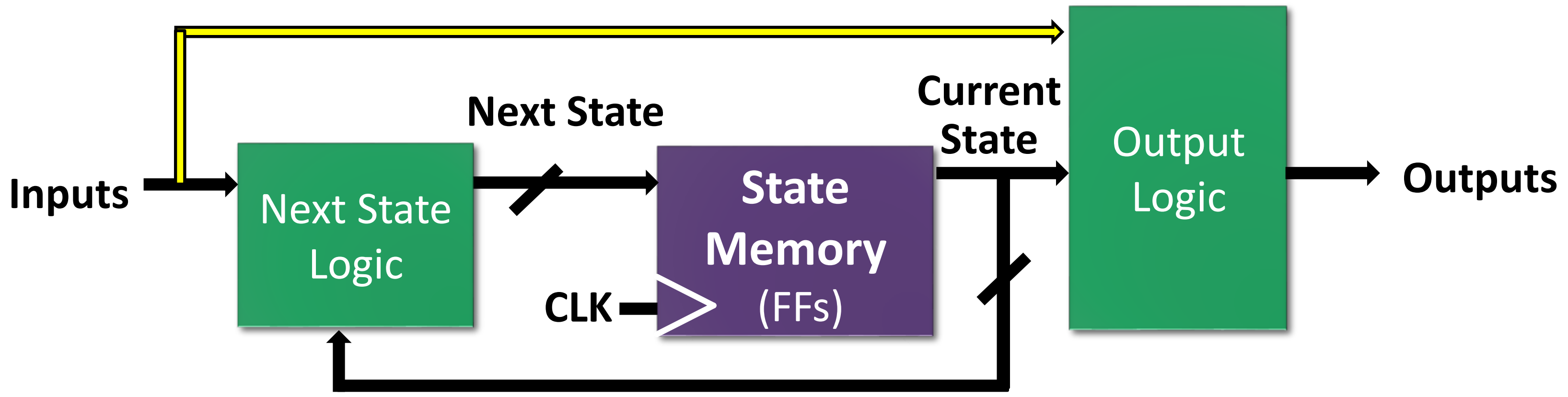


- Examples ?

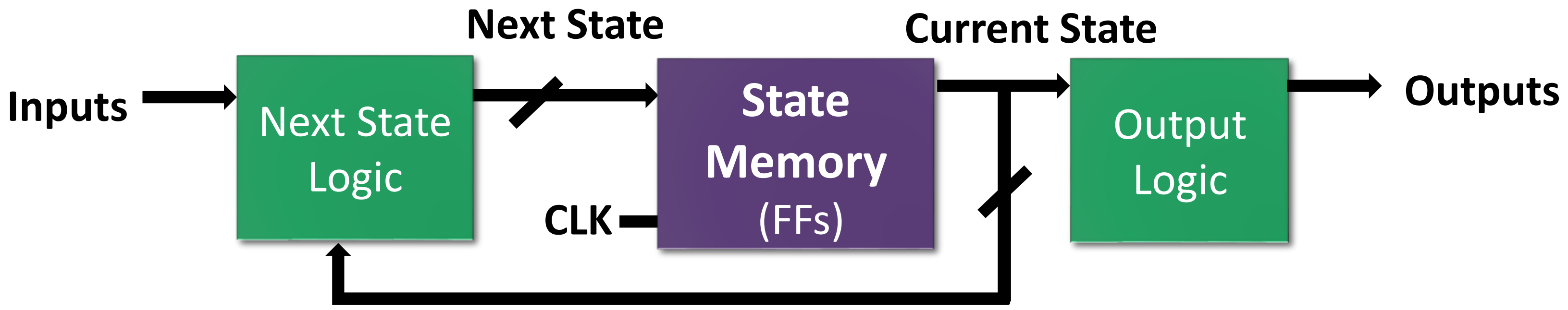
Mealy and Moore

Moore vs Mealy FSMs : Different Output Generation

- **Mealy machine:** ***output is a function of a present state & inputs*.**



- **Moore machine:** ***output is a function of a present state only*.**



Structure

State Memory

- set of n FFs store current state of machine; up to 2^n states.
- FFs can be J-K or D, but D FFs simpler (1 input vs. 2 inputs for J-K FFs)

Next State Logic

- combinational circuit which decides the next state of the machine based on current state and inputs:

  Next state = f (inputs, current state)

Output logic

- combinational circuit which creates the output

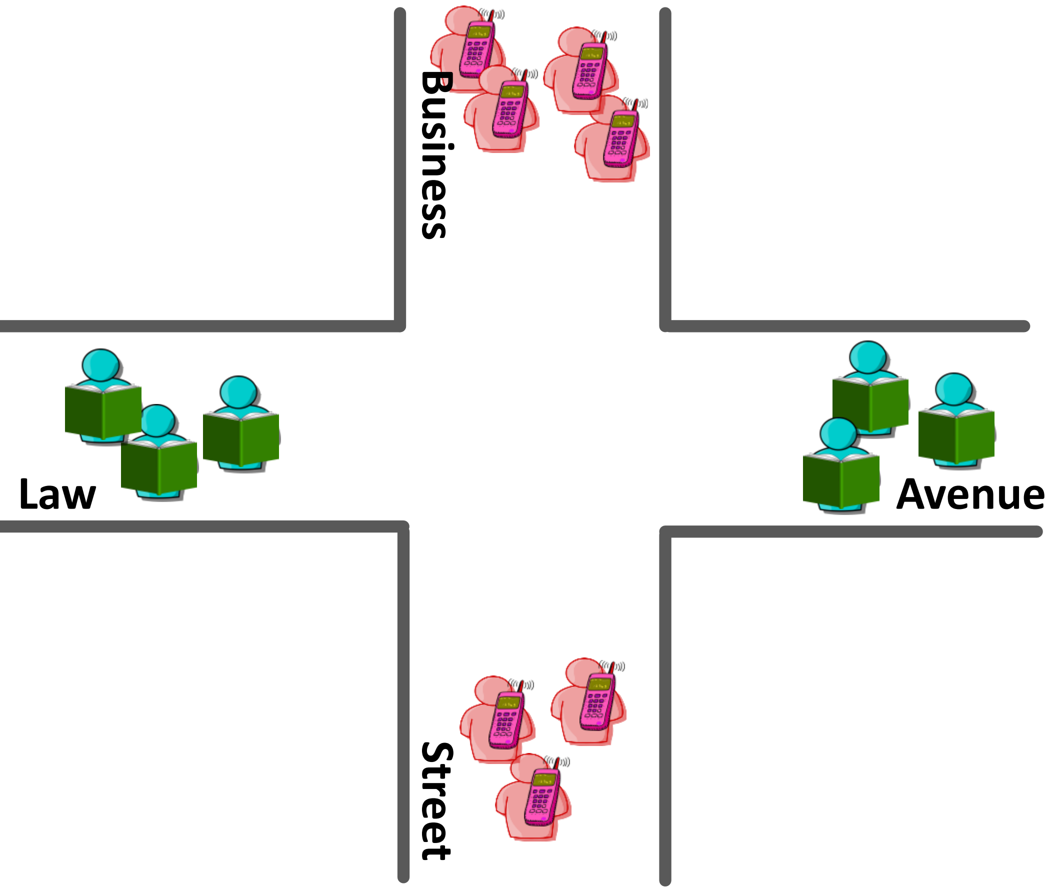
  Moore : outputs depend on current state

  outputs = g (current state)

  Mealy : outputs depend on the current state & inputs

  outputs = g (inputs, current state)

Traffic Problem…

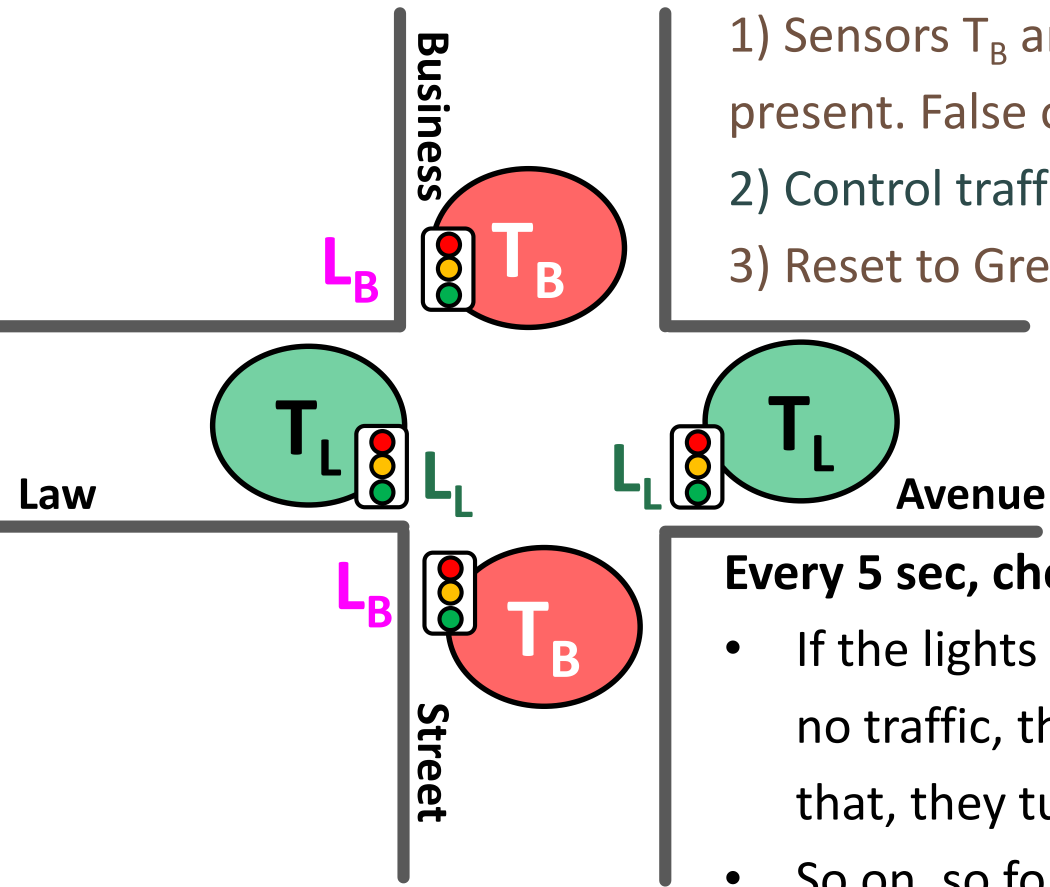


At a busy intersection on campus…

Students from the Law faculty are burying their heads in their books and are not looking where they are going.

Students from the Business faculty are occupied on their phones and aren't looking at where they're going either….

Traffic Problem…



Design a FSM to control the traffic lights!

1) Sensors T_B and T_L is TRUE when students are present. False otherwise.

2) Control traffic lights L_L, L_B to be green, yellow, red.

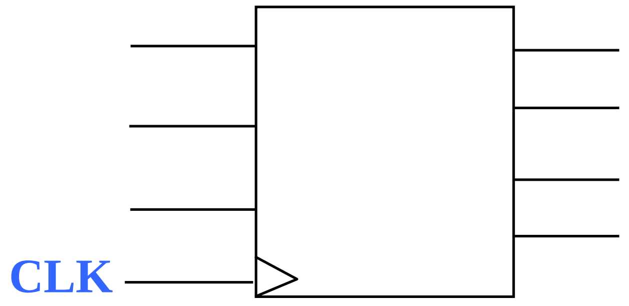
3) Reset to Green on Law Ave and Red on Business St.

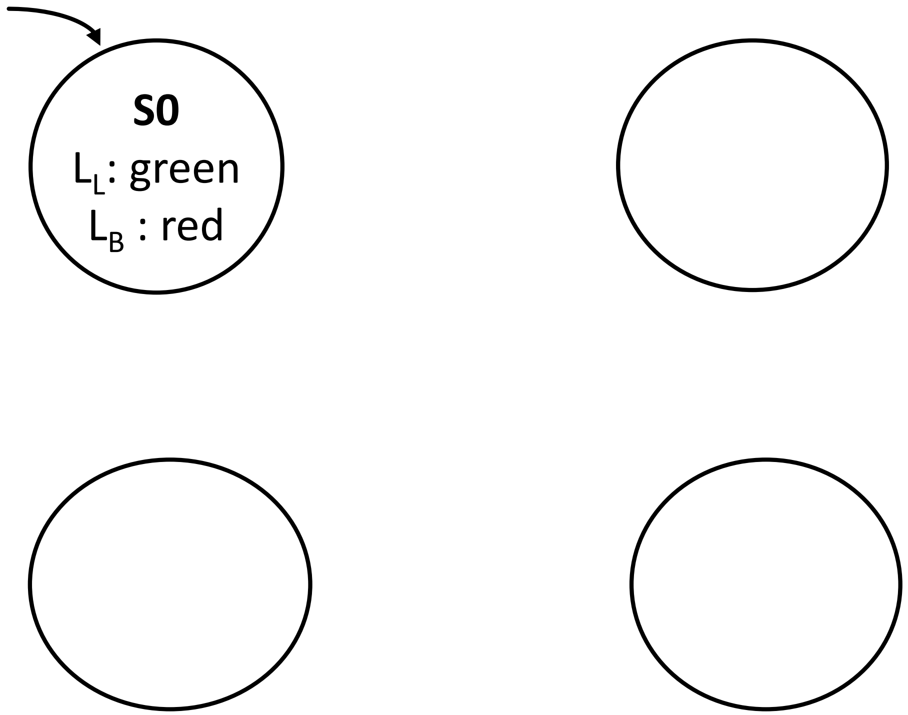
Every 5 sec, check the traffic and decide what to do!

- If the lights on Business St. are green and there's no traffic, the lights turn yellow for 5 secs. After that, they turn red and Law Ave. lights turn green.
- So on, so forth.
- If there is traffic, lights do not change.

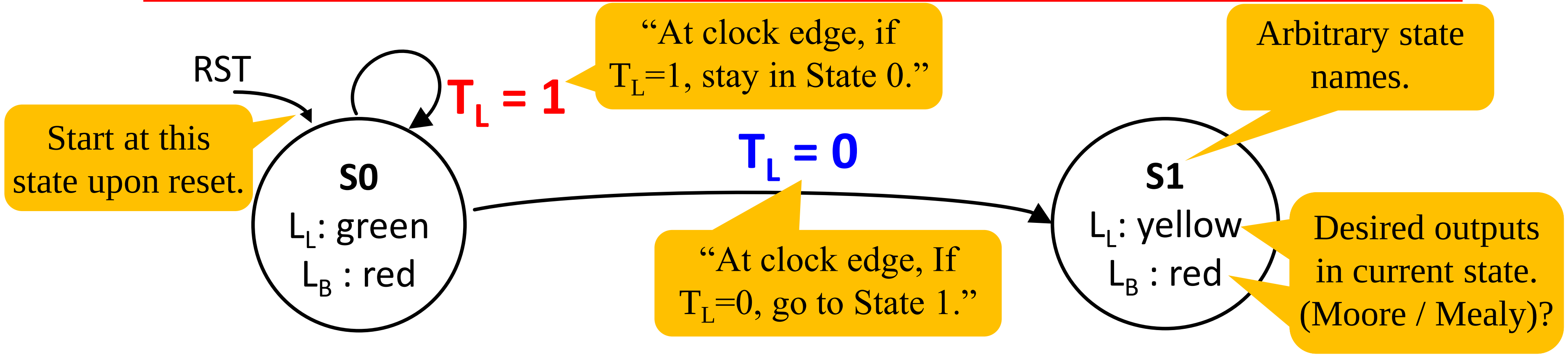
Step 1 : State Transition Diagram

- Overall Block Diagram of System
- Design ***State Transition Diagram*** to represent FSM





State Transition Diagrams


RST
Start at this state upon reset.
S0
L_L: green
L_B : red
$T_L = 1$
"At clock edge, if T_L=1, stay in State 0."
$T_L = 0$
"At clock edge, If T_L=0, go to State 1."
S1
L_L: yellow
L_B : red
Arbitrary state names.
Desired outputs in current state. (Moore / Mealy)?

- Circles represent ***states***.
  - * Each state specifies values for all outputs (Moore)
- Arcs represents ***transitions*** between states.
  - * Labels ➔ input that triggers the transition.
  - * Transitions take place on the active edge of the clock.
- Arc from outer space indicates initial state upon reset
- Within each state, for any combination of input values, there's exactly one applicable arc.

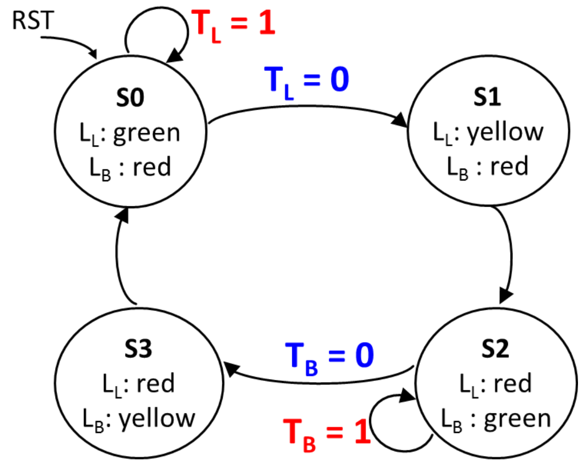
Step 2 : Next State Table

1. From STD, find the number of states
2. Number of bits / FFs required = _?_ to go through these states
3. State Assignment

State	S_1S_0
S0	00
S1	01
S2	10
S3	11

4. Next State Table

Current State	Inputs		Next State
S	T_L	T_B	S+
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3	X	X	S0



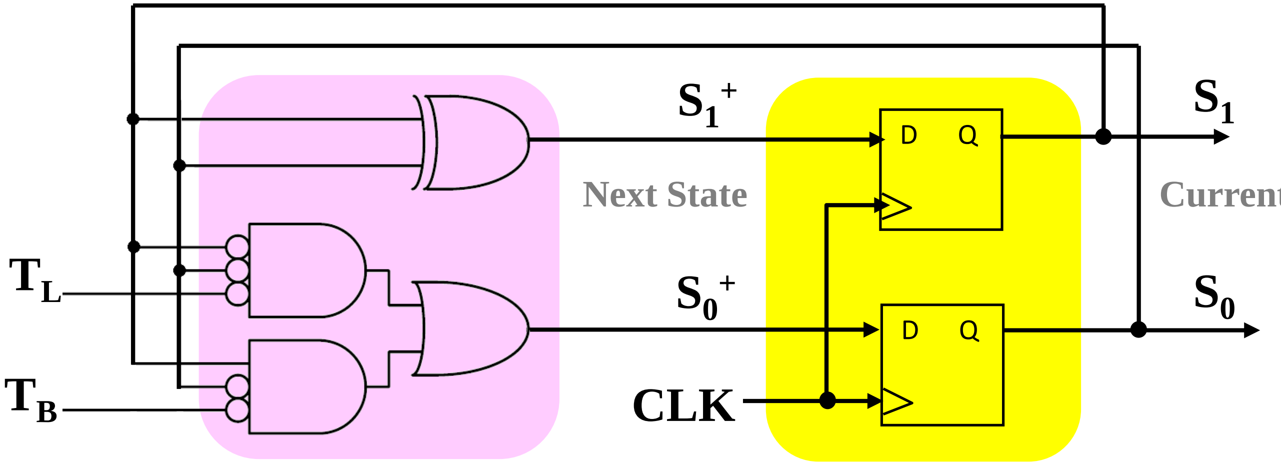
Step 2 : Next State Table

5. State Generator Circuit

Current State		Inputs		Next State	
S_1	S_0	T_L	T_B	S_1+	S_0+
0	0	**0**	X	0	1
0	0	**1**	X	0	0
0	1	X	X	1	0
1	0	X	**0**	1	1
1	0	X	**1**	1	0
1	1	X	X	0	0

$$S_1^+ = \overline{S_1} S_0 + S_1 \overline{S_0}\,\overline{T_B} + S_1 \overline{S_0} T_B$$
$$= S_1 \oplus S_0$$

$$S_0^+ = \overline{S_1}\,\overline{S_0}\,\overline{T_L} + S_1 \overline{S_0}\,\overline{T_B}$$



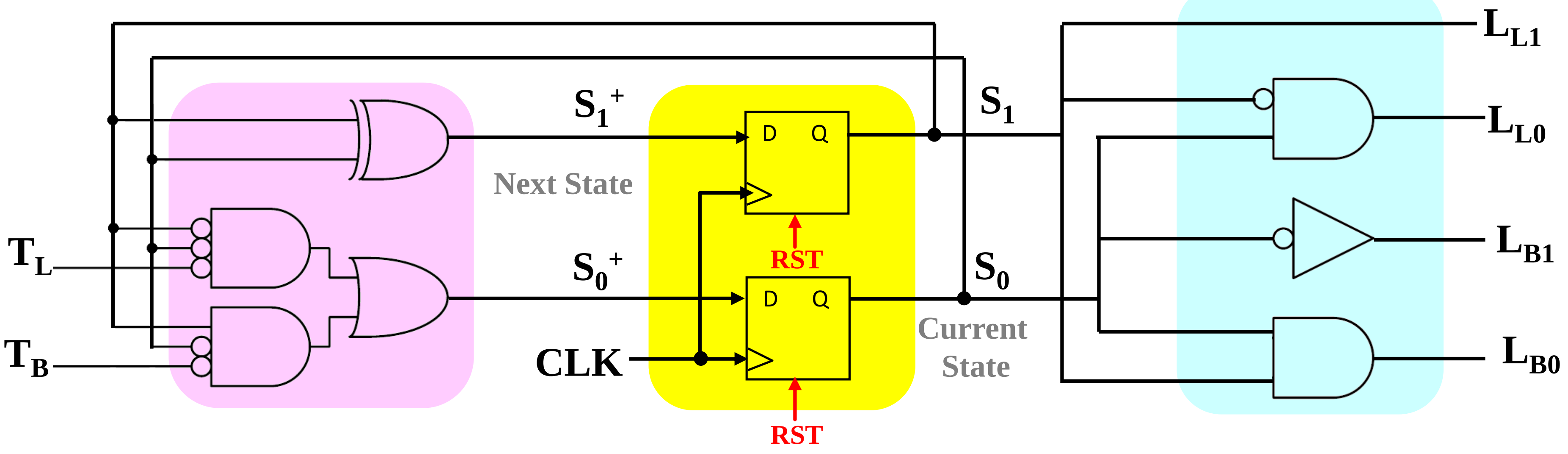
Step 3 : Output Logic

1. Output Truth Table

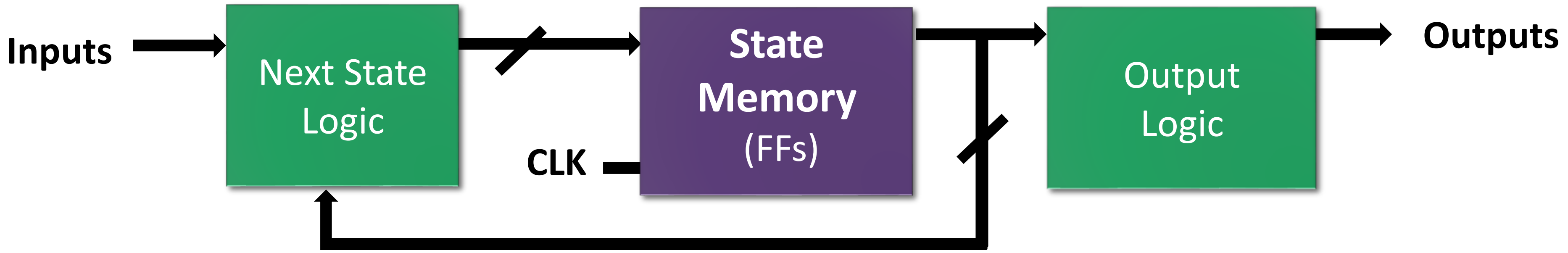
Output	L_1L_0
Green	00
Yellow	01
Red	10

Current State		Outputs			
S_1	S_0	L_{L1}	L_{L0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

$$L_{L1} = S_1 \qquad L_{L0} = \overline{S_1}S_0$$
$$L_{B1} = \overline{S_1} \qquad L_{B0} = S_1S_0$$



Verilog~! – Code Structure



Combinational Circuit

`always @ (*)`

Sequential Circuit

`always @ (posedge clk)`

Combinational Circuit

`assign y =`

FSM in Verilog

```
module fsm(input clk, … , output … … );
reg __ state, nextstate;

parameter S0 = 2’b00;
parameter S1 = 2’b01;
```

parameter is used to define constants within a module, improving code readability.

```
always (*)
    case (state)
        S0  : nextstate = S1;
        S1  : nextstate = S0;
    endcase
```

Next State Combinational Logic : **(*)** code is triggered whenever any input changes ➔ combinational logic. **case** represents next state table.

```
always @ (posedge clk, posedge reset)
    if (reset)  state <= S0;
    else        state <= nextstate;
```

Sequential Logic : Use **<=** to infer flip-flops. *Is this Sync or Async reset?*

```
assign y = ( state == S0 );

endmodule
```

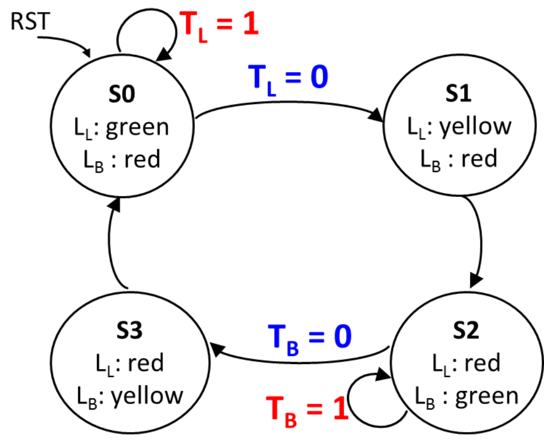
Output Logic : Use `assign` to infer combinational logic.

Equality Comparison :
`a == b` evaluates to 1 if a equals b.

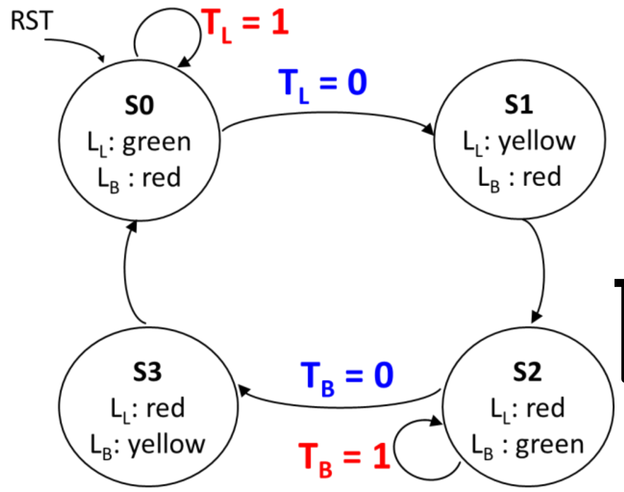
FSM Traffic Controller in Verilog

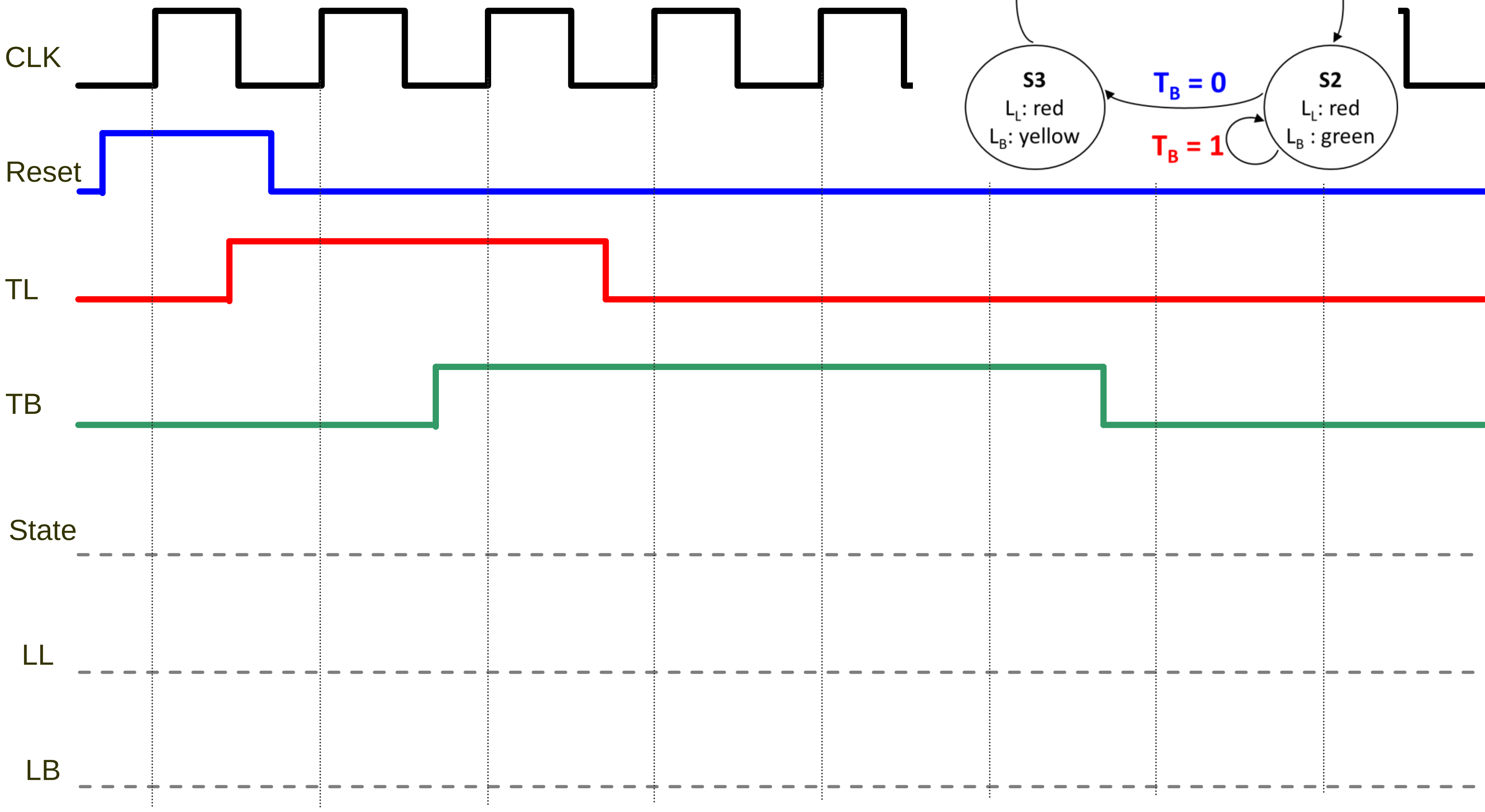
```
module traffic (input clk, reset,
TL, TB, output [1:0] LL, LB);

reg [1:0] state, nextstate;

parameter S0 = 2'b00, S1 = 2'b01,
S2 = 2'b10, S3 = 2'b11;

parameter green = 2'b00,
yellow= 2'b01, red= 2'b10;

always @ (*) begin
  case (state)
    S0: nextstate = TL ? S0 : S1;
    S1: nextstate = S2;
    S2: nextstate = TB ? S2 : S3;
    S3: nextstate = S0;
  endcase
end
```

```
always@(posedge clk, posedge reset)
begin
    if (reset) state <= S0;
    state <= nextstate;
end

//What's the diff between these
//2 ways of coding output logic ?
assign LL
= {state[1], ~state[1] & state[0]};

assign LB =
(state== S0 || state== S1) ? red :
( (state == S2) ? green : yellow );

endmodule
```



Timing Diagram

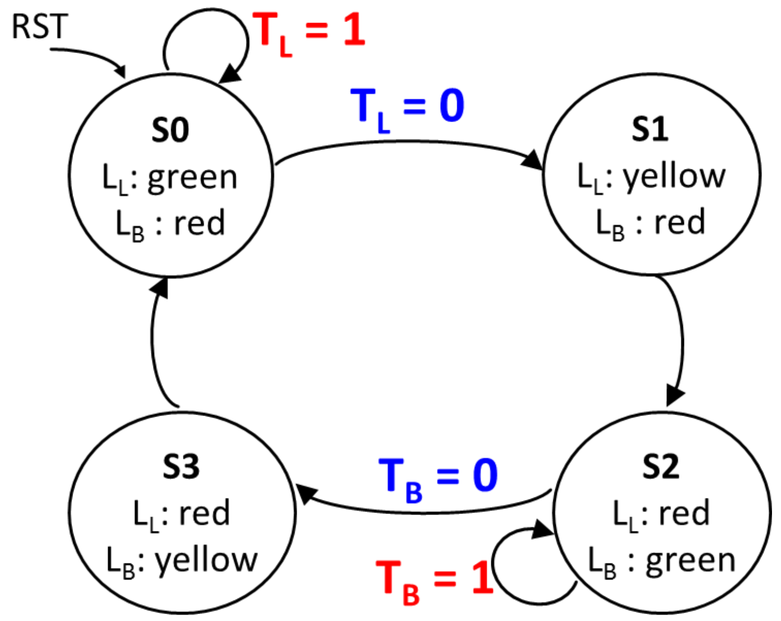




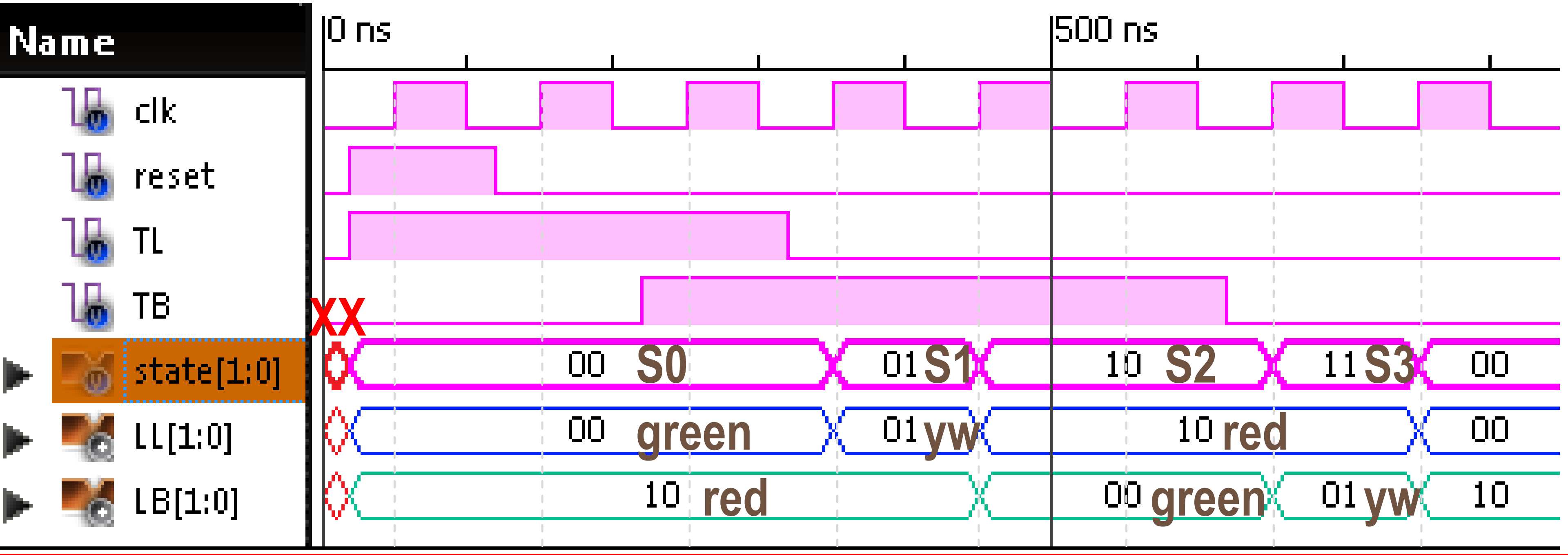
Traffic Controller

- Check the waveforms in the timing diagram below…

Are they correct?



RST
$T_L = 1$
$T_L = 0$
S0
L_L: green
L_B : red
S1
L_L: yellow
L_B : red
S2
L_L: red
L_B : green
S3
L_L: red
L_B: yellow
$T_B = 0$
$T_B = 1$



Name
0 ns
500 ns
clk
reset
TL
TB
state[1:0]
LL[1:0]
LB[1:0]
XX
00 S0
01 S1
10 S2
11 S3
00
00 green
01 yw
10 red
00
10 red
00 green
01 yw
10