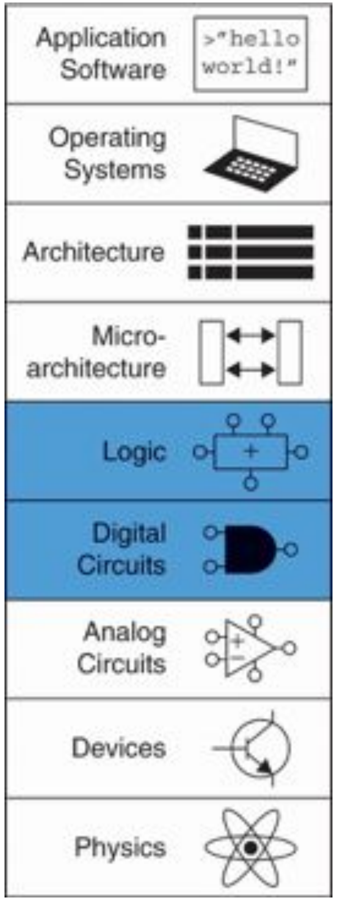
COSC175 (Systems I): Computer Organization & Design

Professor Lillian Pentecost
Fall 2024

Warm-Up September 24

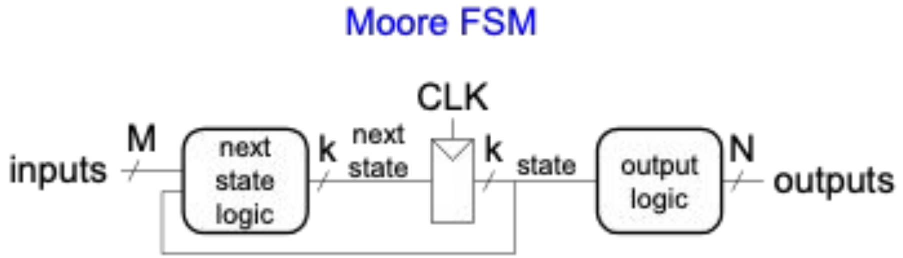
- Where we were
  - Introducing Synchronous Sequential Logic & FSMs
- Where we are going
  - Detailed FSM Example (continued)
  - Testing our designs in simulation
- Logistics, Reminders
  - TA help 7-9PM on Sundays, Tuesdays, Thursdays in C107
    - Vivado Help Session & Tutorial tonight!!
  - LP Office Hours at 2:30-4PM Today
  - Weekly Exercises Due Friday 5PM
  - Pre-Lab for tomorrow (FSM for a different traffic light, read DDCA 4.9)
- Textbook Tags: 3.4, 4.4, 4.6, 4.9

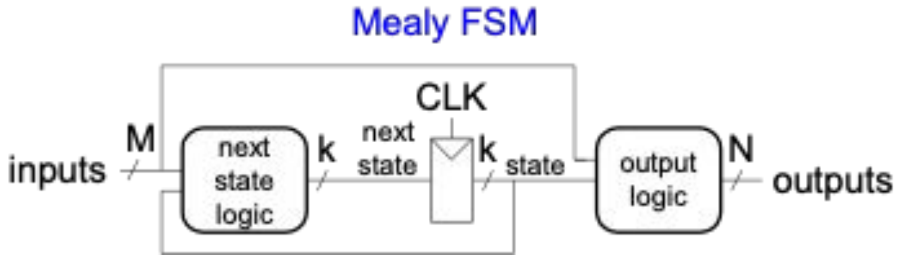


Finite State Machines (FSMs)

- ***State register*** to store *current state*
  - The register is updated to a *next state* at the clock edge
- ***Combinational Logic*** to compute the next state and outputs based on the current state and inputs

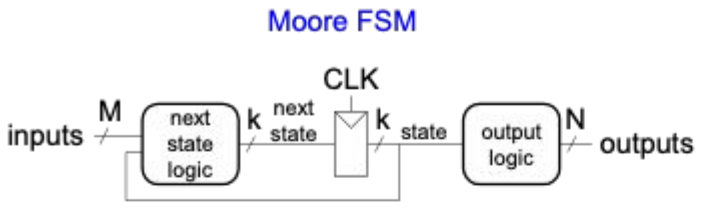
More broadly though: A FSM is an expressive and meaningful way to describe the behavior of a system, and framing questions and system behavior

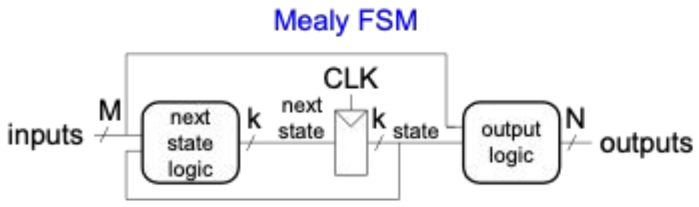




FSM Design Procedure!

1. Identify inputs and outputs
2. Sketch ***state transition diagram***
3. Write state transition table and output table
    a. Moore FSM: write separate tables
    b. Mealy FSM: write combined state transition and output table
4. Select state encodings
5. Rewrite state transition table and output table with state encodings
6. Write Boolean equations for next state and output logic
7. Sketch the circuit schematic



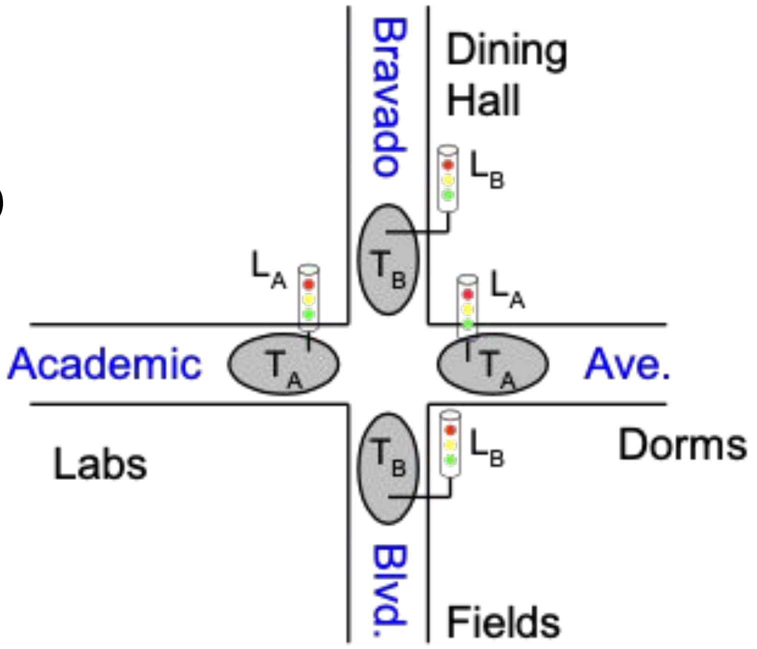


So squishy! Let's try an example.

- Traffic sensors:
  - TA, TB (1 when there are cars present)
- Lights:
  - LA, LB can be green (go), yellow (slow), or red (stop)

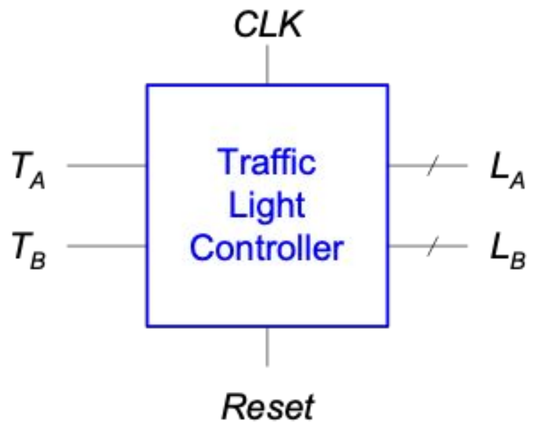
We would like to control the traffic light so that:

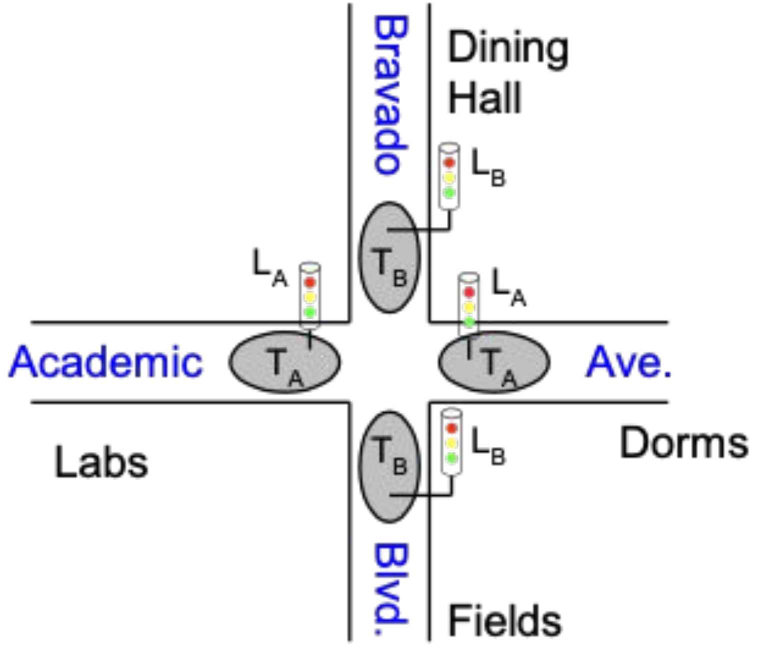
- The two lights do not cause accidents (can't both be green!)
- The appropriate light stays green until no traffic is present



So squishy! Let's try an example.

Here is our basic system:

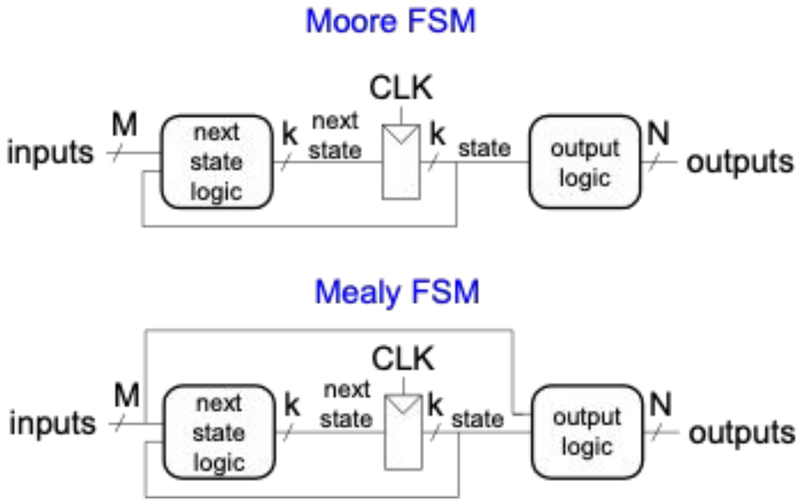




… now we need to define the ***<u>state transitions</u>*** that should occur under each circumstance

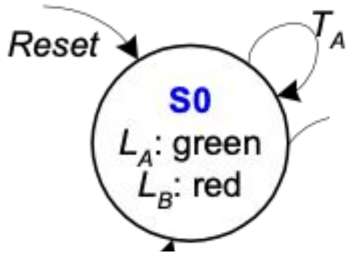
FSM State Transition Diagrams

- We will start with a ***Moore***-style FSM, so we will separately determine the next state based on current state, the output based on the state
- Each possible ***state*** is represented by a circle
- ***Transitions*** between states are drawn as arcs



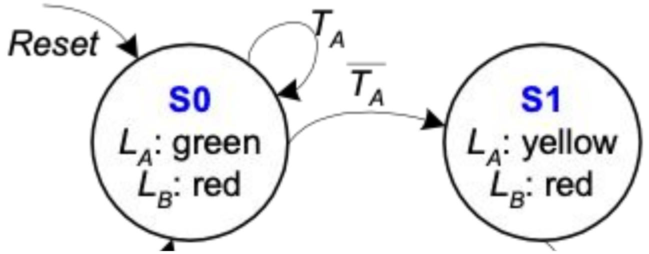
FSM State Transition Diagrams

- Each possible ***state*** is represented by a circle
- ***Transition***s between states are drawn as arcs



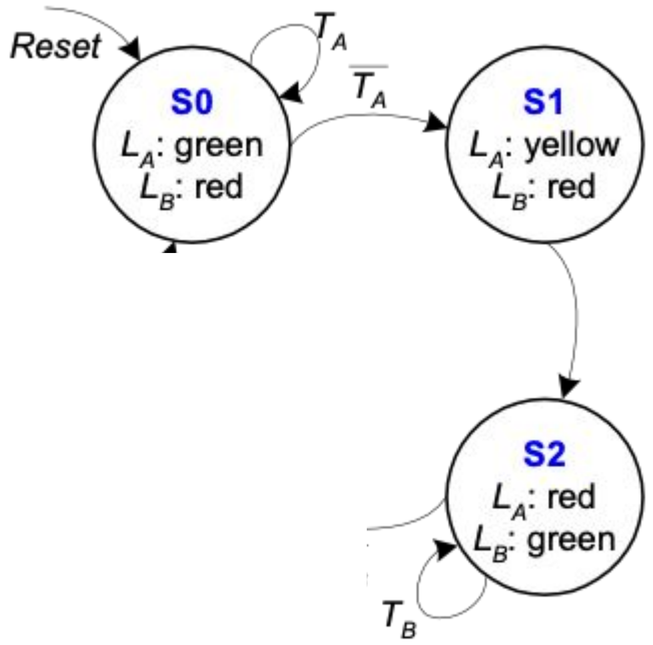
FSM State Transition Diagrams

- Each possible ***state*** is represented by a circle
- ***Transition***s between states are drawn as arcs



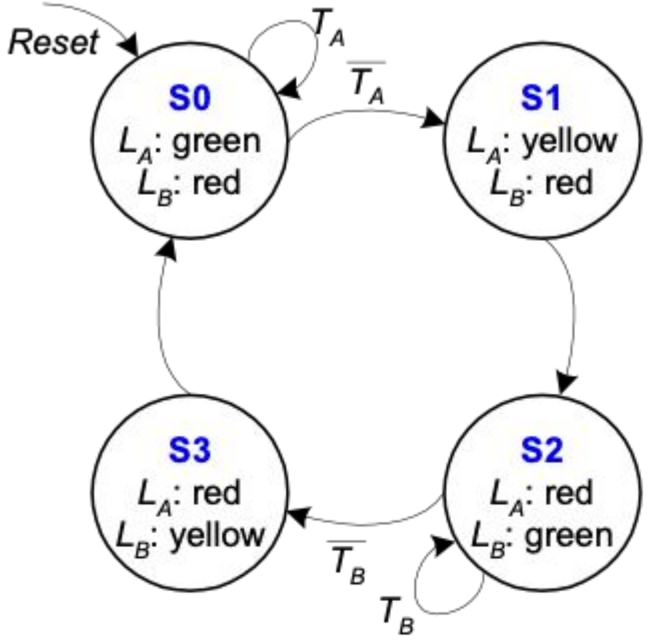
FSM State Transition Diagrams

- Each possible ***state*** is represented by a circle
- ***Transitions*** between states are drawn as arcs

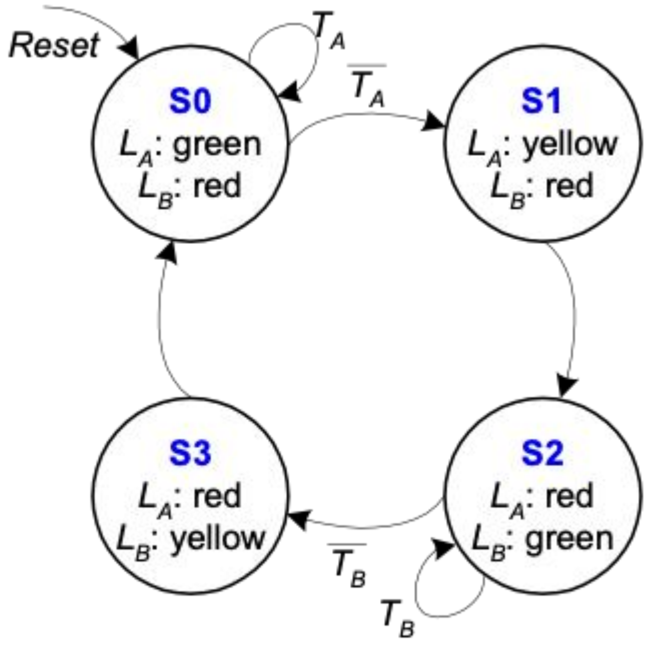


FSM State Transition Diagrams

- Each possible ***state*** is represented by a circle
- ***Transition***s between states are drawn as arcs

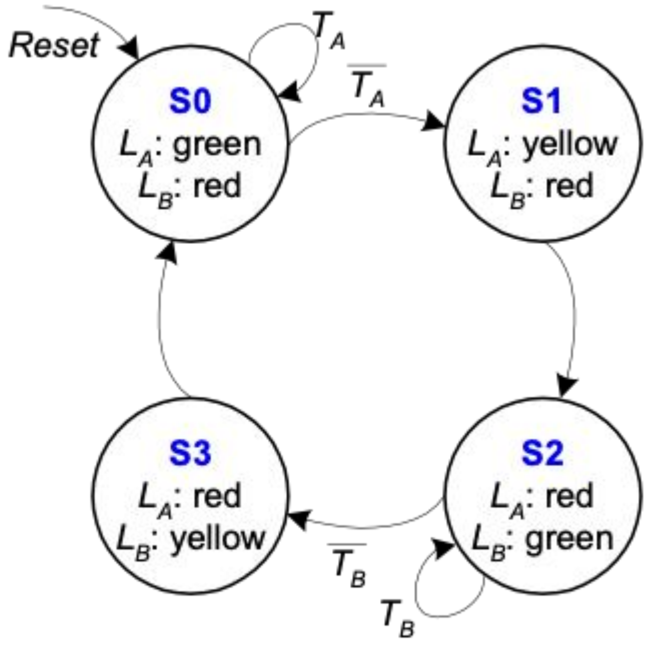


Translate behavior to a ***State Transition Table***



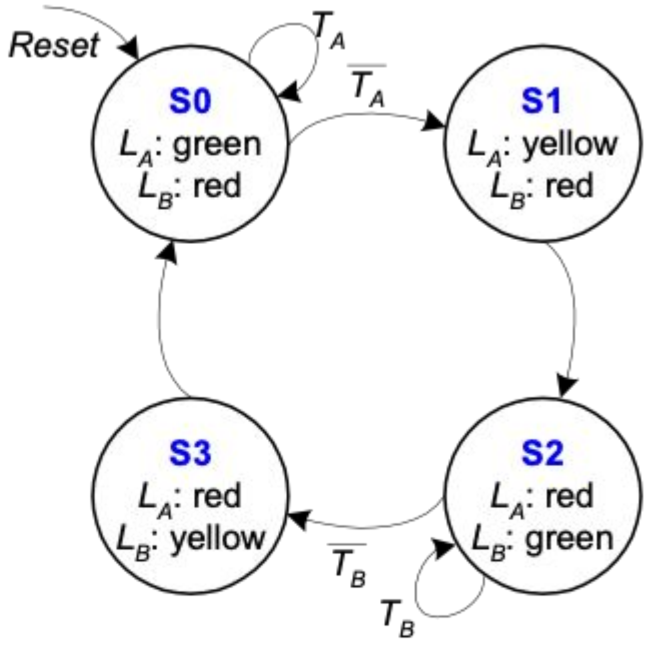
Current State	Inputs		Next State
S	T_A	T_B	***S'***
S0			
S0			
S1			
S2			
S2			
S3			

Translate behavior to a ***State Transition Table***



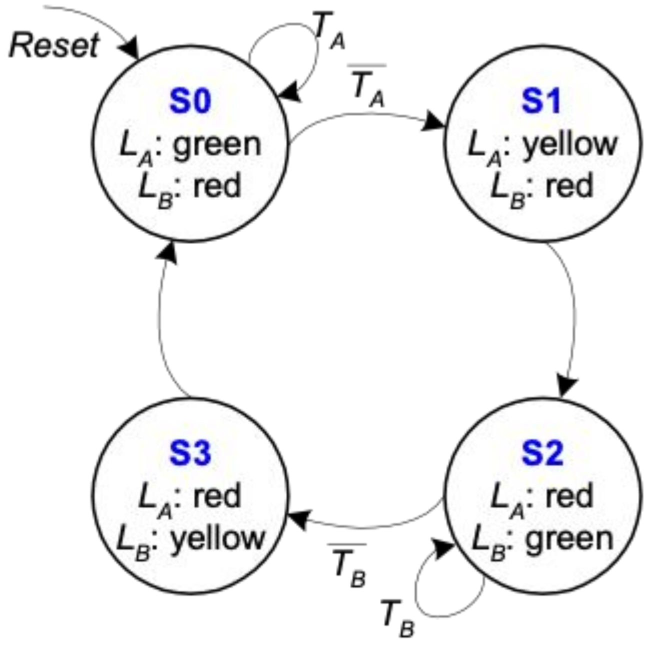
Current State	Inputs		Next State
S	T_A	T_B	S'
S0	0	X	S1
S0			
S1			
S2			
S2			
S3			

Translate behavior to a ***State Transition Table***



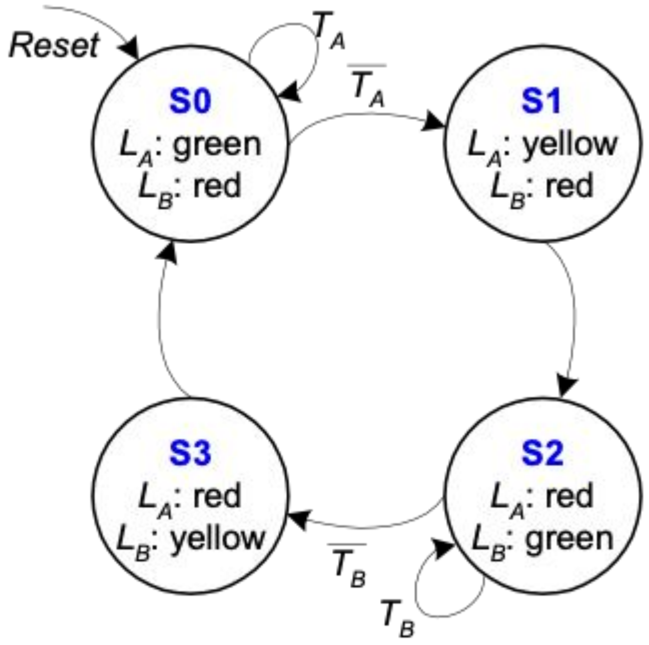
Current State	Inputs		Next State
S	T_A	T_B	***S'***
S0	0	X	S1
S0	1	X	S0
S1			
S2			
S2			
S3			

Translate behavior to a ***State Transition Table***



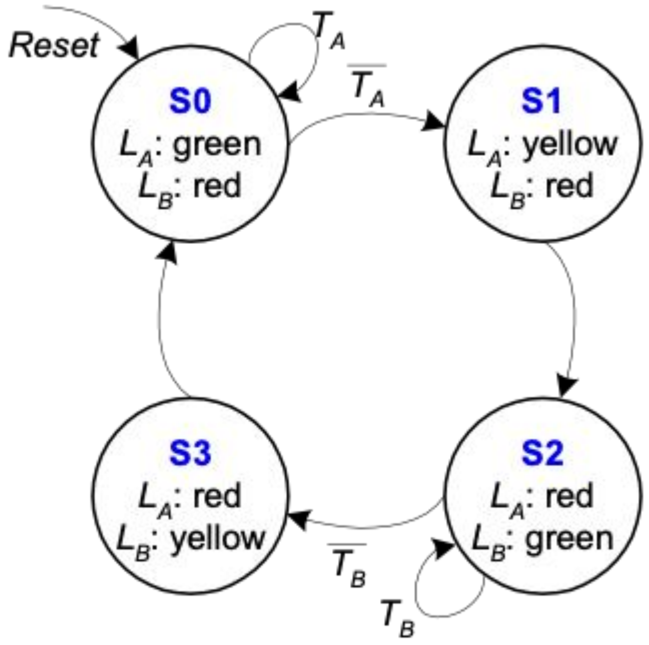
Current State	Inputs		Next State
S	***T_A***	***T_B***	***S'***
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2			
S2			
S3			

Translate behavior to a ***State Transition Table***



Current State	Inputs		Next State
S	***T_A***	***T_B***	***S'***
S0	0	X	S1
S0	1	X	S0
S1	X	X	S2
S2	X	0	S3
S2	X	1	S2
S3			

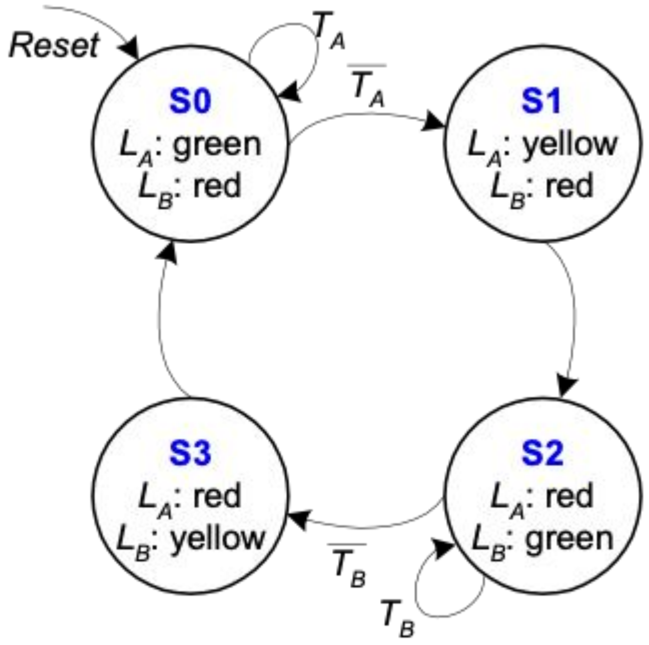
Translate behavior to a ***State Transition Table***



Current State	Inputs		Next State
S	***T***		

What about the ***output*** of the system?

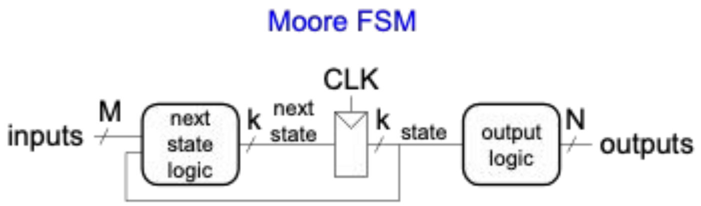
Output	Encoding
green	00
yellow	01
red	10

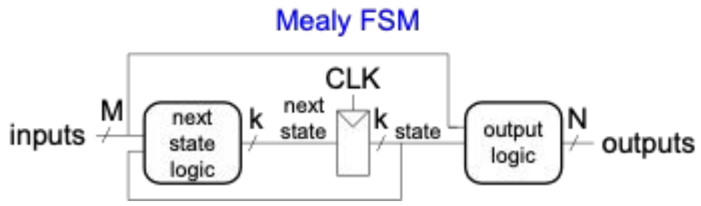


Current State	Outputs			
	L_{A1}	L_{A0}	L_{B1}	L_{B0}
S0	0	0	1	0
S1	0	1	1	0
S2	1	0	0	0
S3	1	0	0	1

FSM Design Procedure!

1. Identify inputs and outputs
2. Sketch ***state transition diagram***
3. Write state transition table and output table
    a. Moore FSM: write separate tables
    b. Mealy FSM: write combined state transition and output table
4. **Select state encodings**
5. Rewrite state transition table and output table with state encodings
6. Write Boolean equations for next state and output logic
7. Sketch the circuit schematic





Encoded State, Resulting TTs
We know this part!!
Translate to equations & schematic

State	Encoding
S0	00
S1	01
S2	10
S3	11

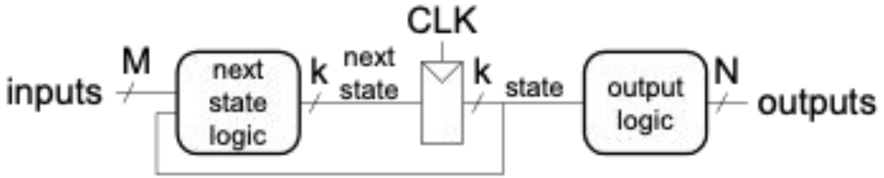
Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	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

Encoded State, Resulting TTs
We know this part!!
Translate to equations & schematic

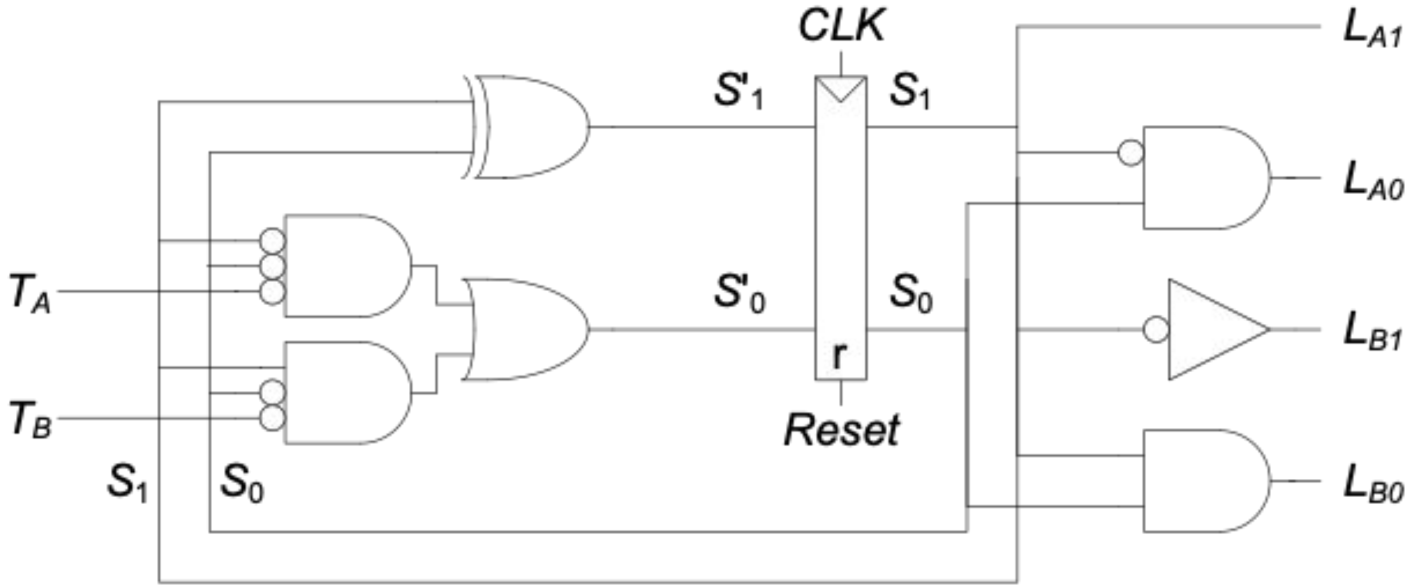
Current State		Inputs		Next State	
S_1	S_0	T_A	T_B	S'_1	S'_0
0	0	0	X	0	1
0	0	1	X	0	0
0	1	X	X	1	0
1	0	X	0	1	1
1	0	X	1	1	0
1	1	X	X	0	0

Moore FSM

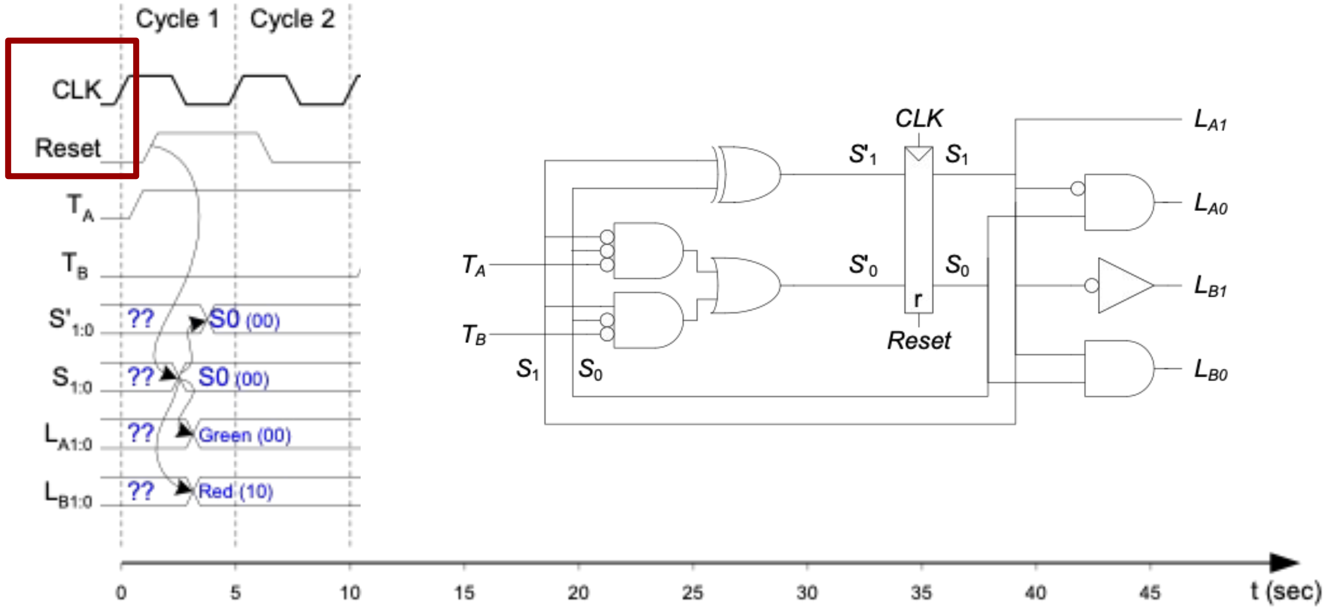


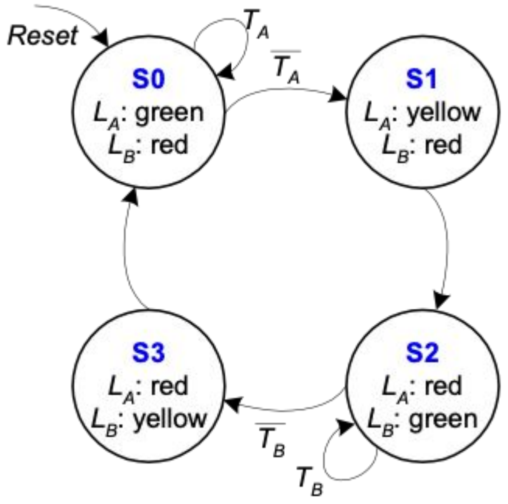
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

Encoded State, Resulting TTs
We know this part!!
Translate to equations & schematic

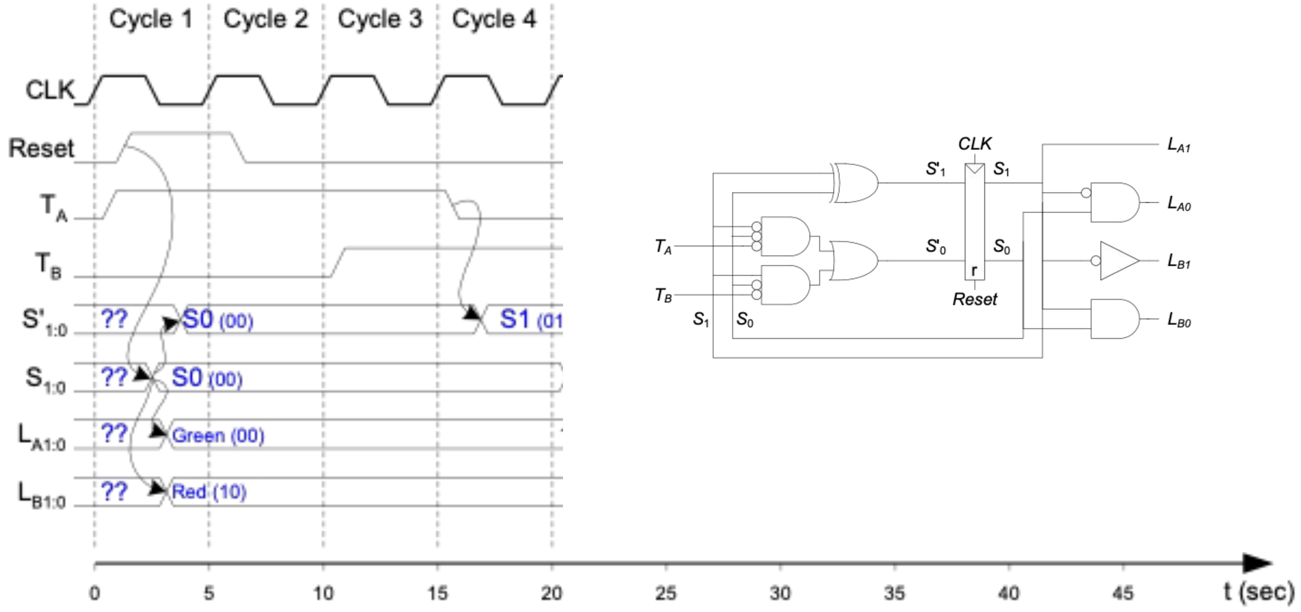


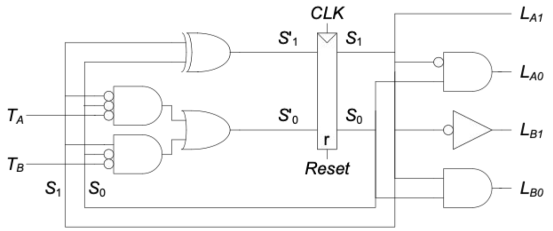
Let’s understand the *timing* for an example input

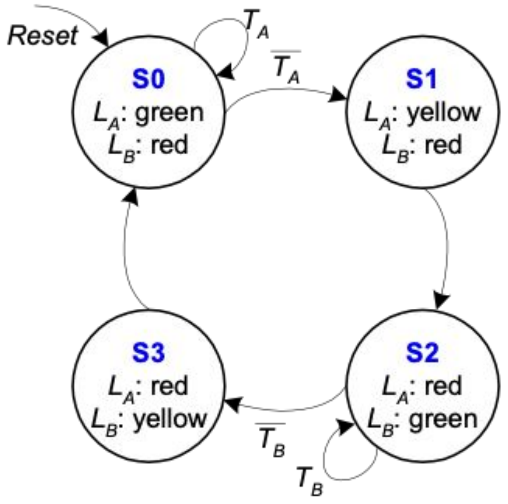




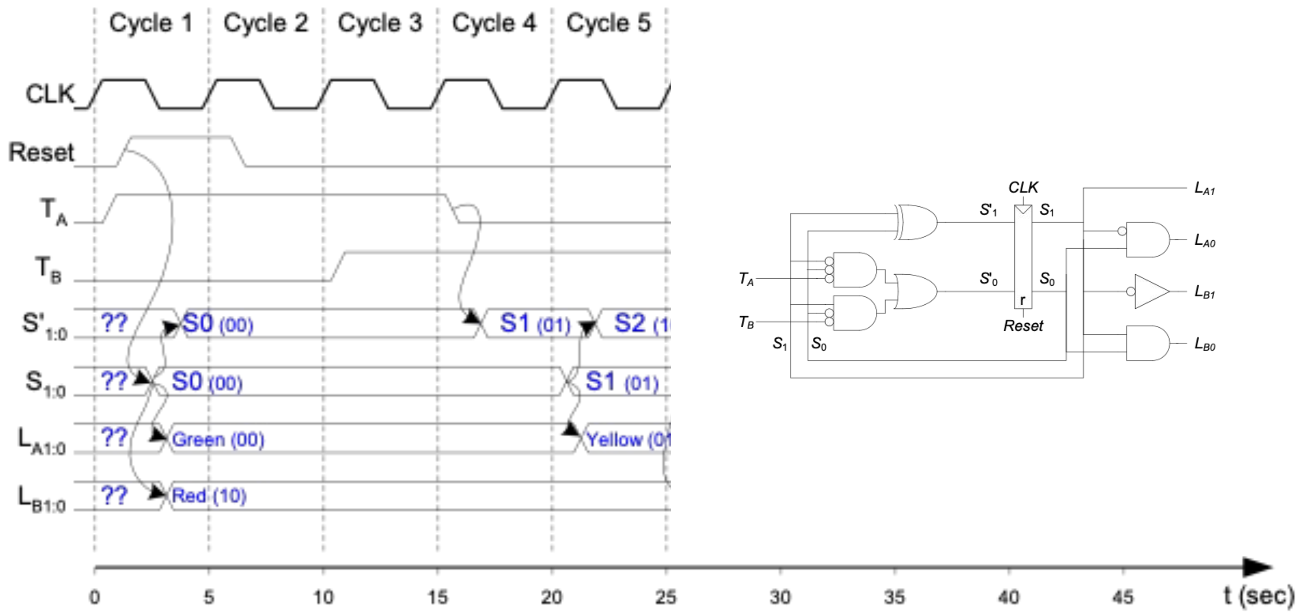
Let's understand the *timing* for an example input

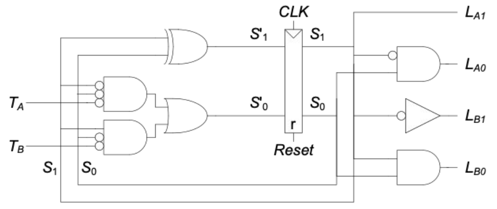


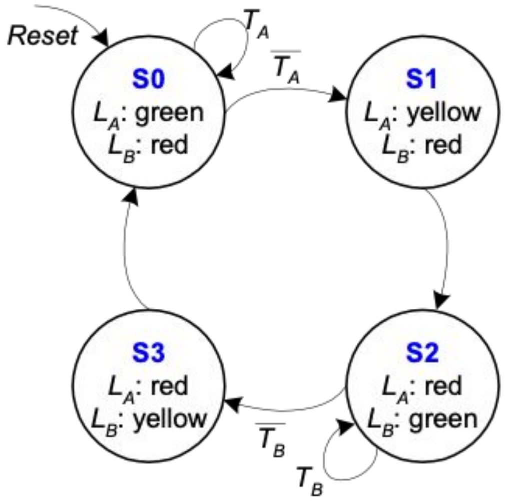




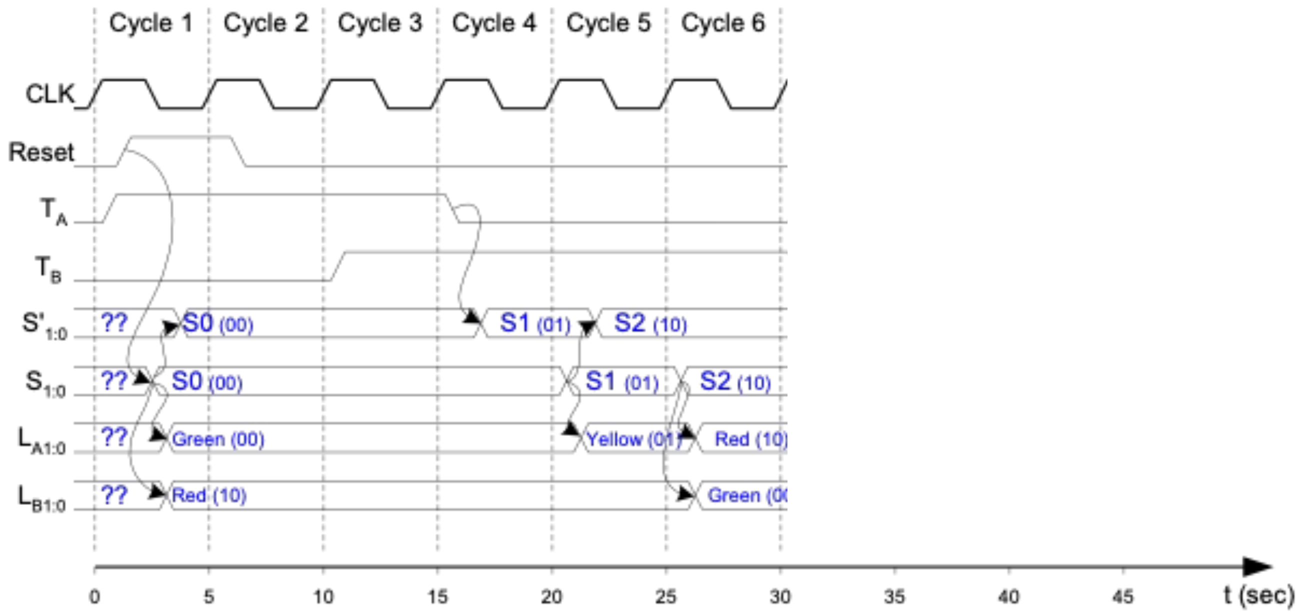
Let’s understand the *timing* for an example input

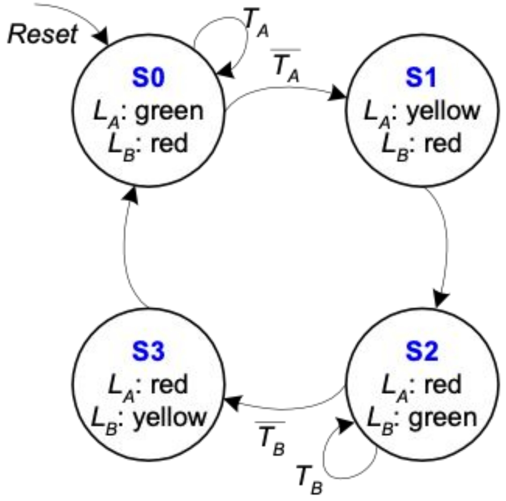




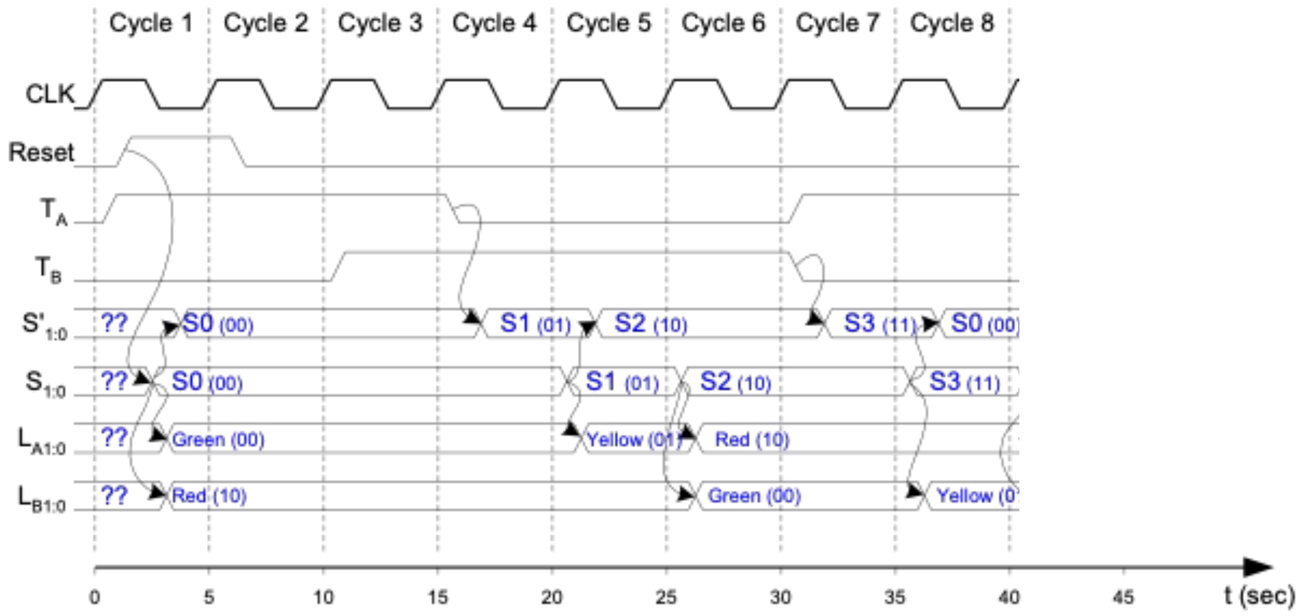


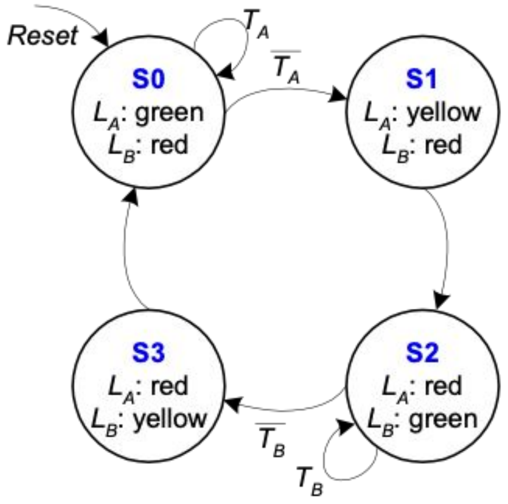
Let's understand the ***timing*** for an example input



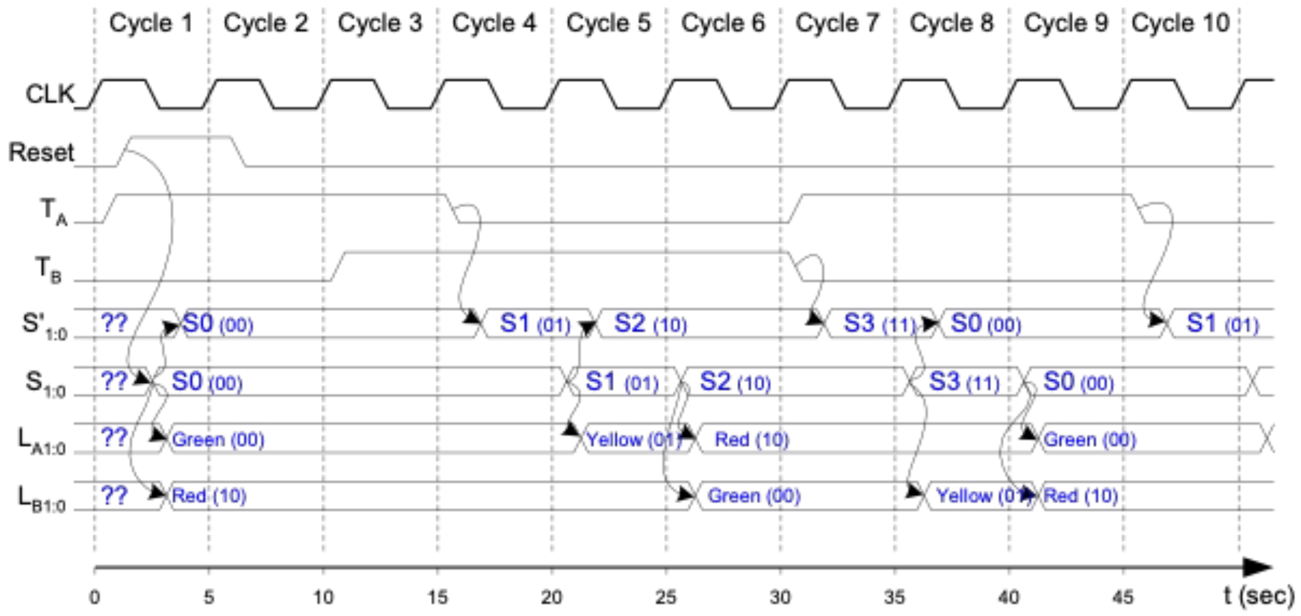


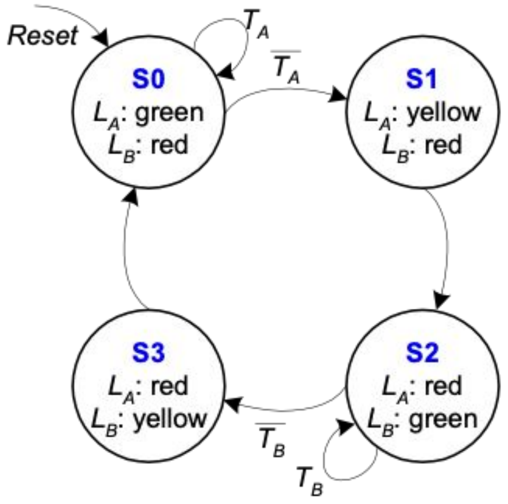
Let’s understand the *timing* for an example input





Let's understand the *timing* for an example input





Testing designs in simulation with ***testbenches***

- A ***testbench*** is an HDL module crafted to test another module
  - Contains statements to apply particular input values, observe / check corresponding output
    - These statements are called ***test vectors***
  - NOT synthesizable!!!
  - Labeled by time stamps or wait times
  - The best testbenches are ***self-checking*** – use `assert` and `error` statements

Example:

```
module sillyfunction(
    input  logic a, b, c,
    output logic y);
  assign y = ~b & ~c | a & ~b;
endmodule
```

Testing designs in simulation with ***testbenches***

```
module sillyfunction(
     input  logic a, b, c,
     output logic y);
  assign y = ~b & ~c | a & ~b;
endmodule
```

```
module testbench1();
  logic a, b, c;
  logic y;
  sillyfunction dut(a, b, c, y);
  initial begin
    a = 0; b = 0; c = 0; #10;
    c = 1; #10;
    b = 1; c = 0; #10;
    c = 1; #10;
    a = 1; b = 0; c = 0; #10;
    c = 1; #10;
    b = 1; c = 0; #10;
    c = 1; #10;
  end
endmodule
```

Self-checking version

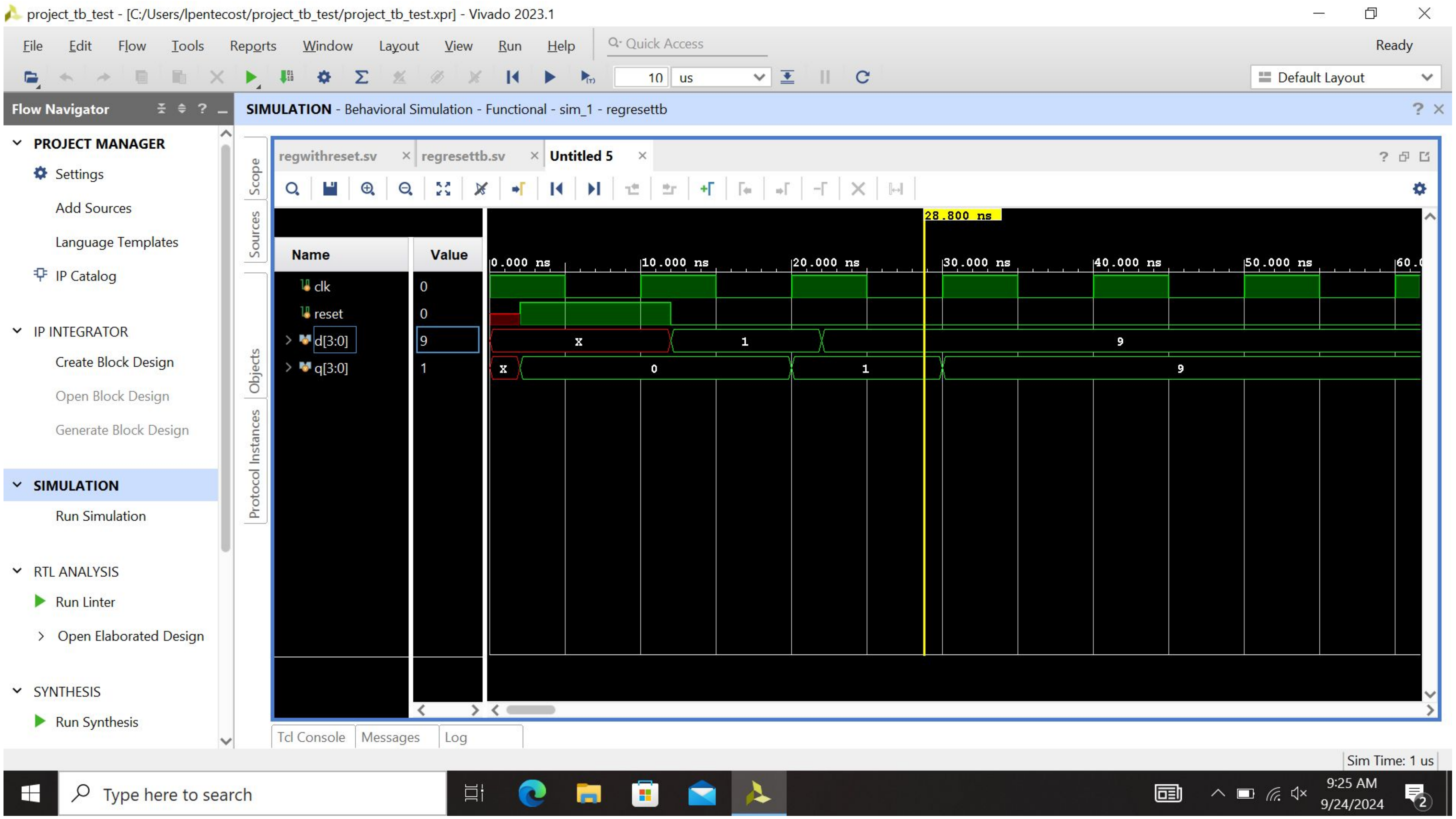
```
module sillyfunction(
     input  logic a, b, c,
     output logic y);
  assign y = ~b & ~c | a & ~b;
endmodule
```

```
module testbench2();
  logic  a, b, c;
  logic y;
  sillyfunction dut(a, b, c, y);
initial begin
    a = 0; b = 0; c = 0; #10;
    if (y !== 1) $display("000 failed.");
    c = 1; #10;
    if (y !== 0) $display("001 failed.");
    b = 1; c = 0; #10;
    if (y !== 0) $display("010 failed.");
    c = 1; #10;
    if (y !== 0) $display("011 failed.");
    a = 1; b = 0; c = 0; #10;
    if (y !== 1) $display("100 failed.");
    c = 1; #10;
```

For a synchronous sequential example

```
module regwithreset(
      input logic clk,
      input logic reset,
      input logic [3:0] d,
     output logic [3:0] q);

   // asynchronous reset
   always_ff @(posedge clk, posedge reset)
     if (reset) q <= 4’b0;
     else       q <= d;

endmodule
```

```
module regresettb();
  logic clk, reset;
  logic [3:0] d;
  logic [3:0] q;
  regwithreset dut(clk, reset, d, q);

  // generate clock
  always
    begin
      clk = 1; #5; clk = 0; #5;
    end

  // then send timed + checked tests
  initial
    begin
      reset = 1; #12; reset = 0;
      d = 1; #10;
      assert ( q === d ) else $error(“failed”);
    end
endmodule
```



project_tb_test - [C:/Users/lpentecost/project_tb_test/project_tb_test.xpr] - Vivado 2023.1

SIMULATION - Behavioral Simulation - Functional - sim_1 - regresettb

Name	Value
clk	0
reset	0
d[3:0]	9
q[3:0]	1

Notes for testing synchronous sequential logic!

Now, you will also connect a clock signal to your circuit:

```
    set_property PACKAGE_PIN W5 [get_ports clk]
    set_property IOSTANDARD LVCMOS33 [get_ports clk]
    create_clock -period 10 [get_ports clk]
```

Use your Basys 3 Reference Manual to understand constraints!

Deploying to FPGA is still our gold-standard for testing, but simulation + waveform viewing helps debug timing, speed up your design/test cycle.

Wrap-Up September 24

SCAN ME

- Coming up next!
  - Test your timed designs in ***simulation*** before deploying to the FPGA
  - Get comfortable with the ***design cycle*** for more complex efforts
  - This week, controllers and more tools/methods, next week ***arithmetic units***
- Logistics, Reminders
  - Evening help sessions 7-9PM on Sundays, Tuesdays, Thursdays in C107
  - Weekly Exercises Due Friday 5PM
  - Stay up to date with readings!
  - Pre-Lab for tomorrow (FSM for a different traffic light, read DDCA 4.9)
  - Lab 0 Feedback will be posted via Moodle, reach out to **me** with questions
- FEEDBACK
  - https://forms.gle/5Aafcm3iJthX78jx6