ENGN2219/COMP6719
Computer Systems & Organization

Convener: Shoaib Akram

shoaib.akram@anu.edu.au

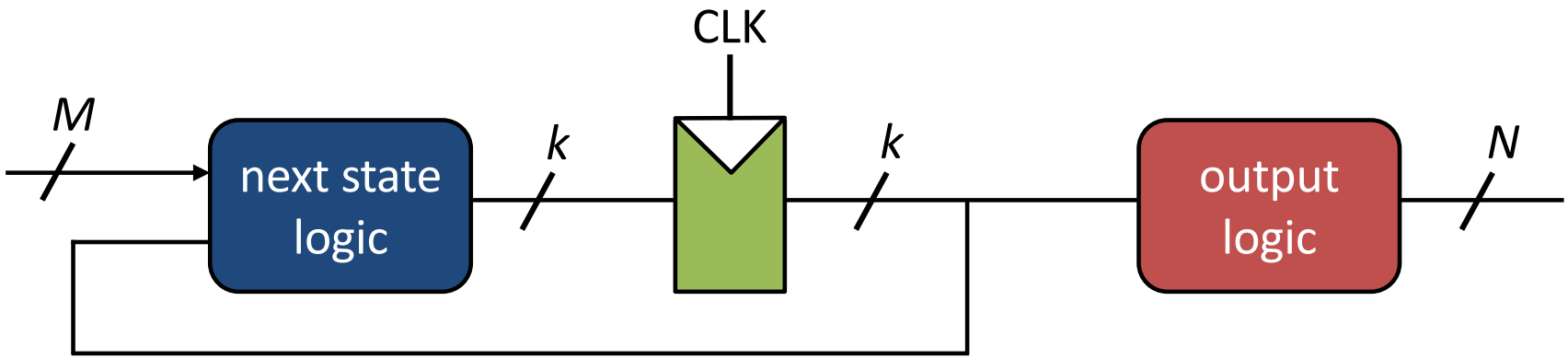
Australian
National
University

Two Sync. Sequential Circuits

- Two widely used synchronous sequential circuits
  - Finite state machine (FSM)
  - Pipelines
- FSMs can be used to solve many real-world problems
  - Traffic light controller, elevator controller, ...
- Pipelining helps reduce the clock period of synchronous sequential circuits (via parallelism)
- These circuits gives us greater insight into how synchronous sequential circuits behave

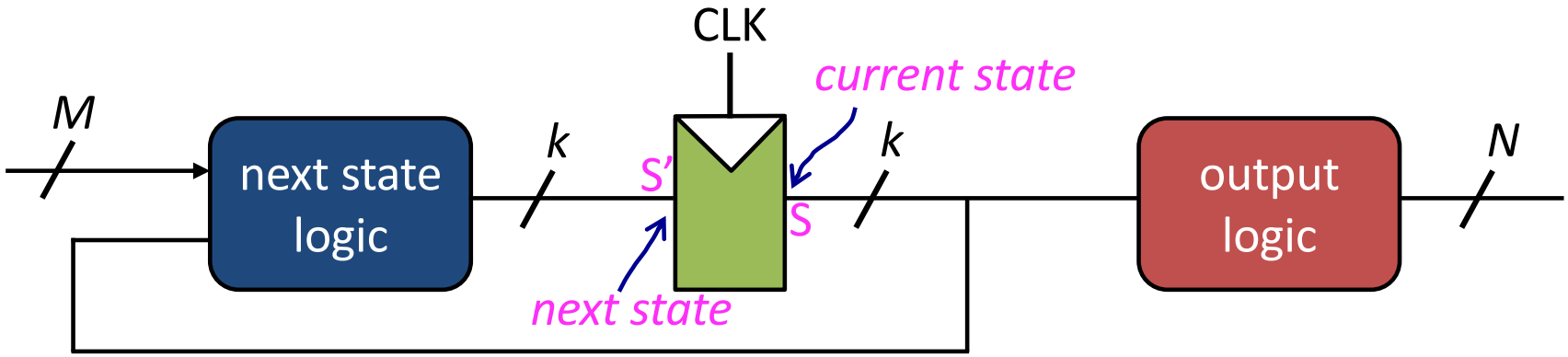
Finite State Machine (FSM)

- A *k-bit register* to store one of 2^k (finite) states
- The *next state logic* computes the next state
  - Next state depends on the current state and inputs
  - FSM advances to the next state on each clock edge
  - *Remark: We use S for current state and S' for the next state*
- The *output logic* computes the output based on the current state (Moore machine) and current state and inputs (Mealy machine)



Finite State Machine (FSM)

- In one cycle (say *N*), next state logic produces the next state (S') based on the current state (S) and the M inputs
- In the next cycle (*N+1),* the state register updates the current state (S) of the system to be equal to S'
- *Warning: S' (or S prime) has two possible meanings. We use the prime notation for NOT or Invert earlier, and now we will use it for next state (convention). The context will make it clear what we mean*

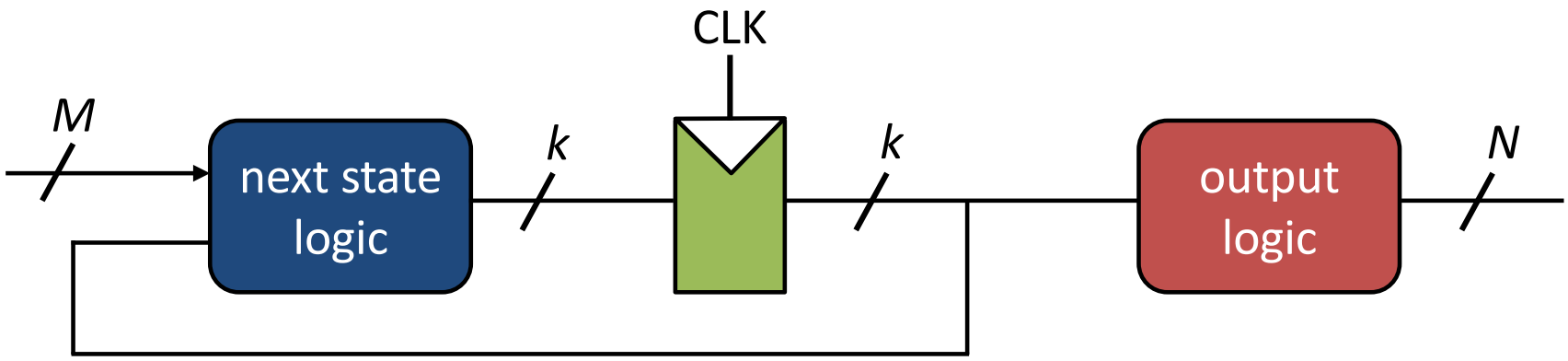


Slide added after the lecture

Implementing FSMs

Goal: Take the initial specification in English and build the FSM circuit using logic gates

- Use the FSM circuit template shown below
- Derive next state logic and output logic



Implementing FSMs

Step # 1: State transition diagram

- Formalize the specification and remove ambiguity

Step # 2: Derive the next state logic

- Binary encoding for states
- State transition (truth) table
- Minimized Boolean equations for next state logic

Step # 3: Derive the output logic

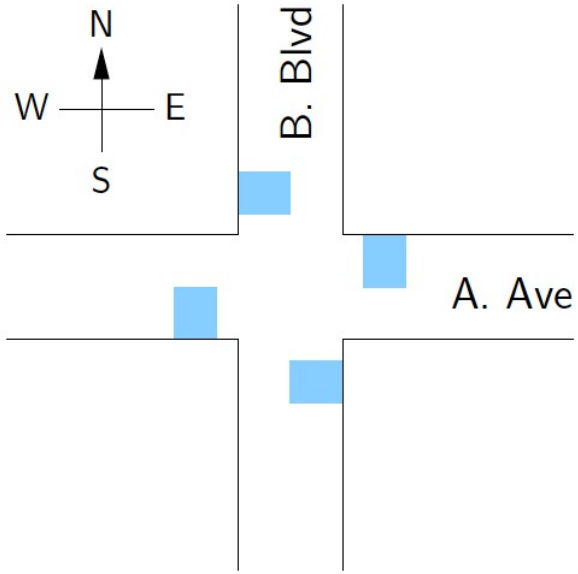
- Binary encoding for outputs
- Output table & Boolean equations

Step # 4: Turn the Boolean equations into logic gate implementation

- Next state logic & output logic

Traffic Light Controller

- Let's build a traffic light controller for a busy intersection
- Traffic sensors are built into the road
- Each sensor indicates if a street is empty or there are vehicles nearby

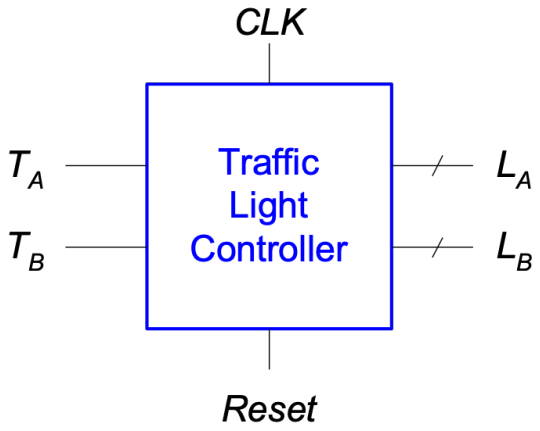


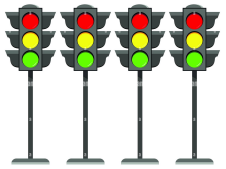
T_A = (eastbound traffic on A) OR (westbound traffic on A)

T_B = (northbound traffic on B) OR (southbound traffic on B)

Traffic Light Controller Problem

- Inputs T_A and T_B
  - Returns TRUE if there are cars on the road
  - Returns FALSE if the road is empty
- Outputs $L_{A1:0}$ and $L_{B1:0}$
  - Each set of lights receive 2-bit digital inputs from the traffic light controller specifying whether it should be: RED, YELLOW, GREEN





Output	Encoding
GREEN	00
YELLOW	01
RED	10

Traffic Light Controller Problem

- Clock with period of 5 seconds (frequency = 0.2 Hz)
  - On each rising edge, the lights may change based on traffic sensors
- A Reset input to put the controller in a known state

Implementing FSMs

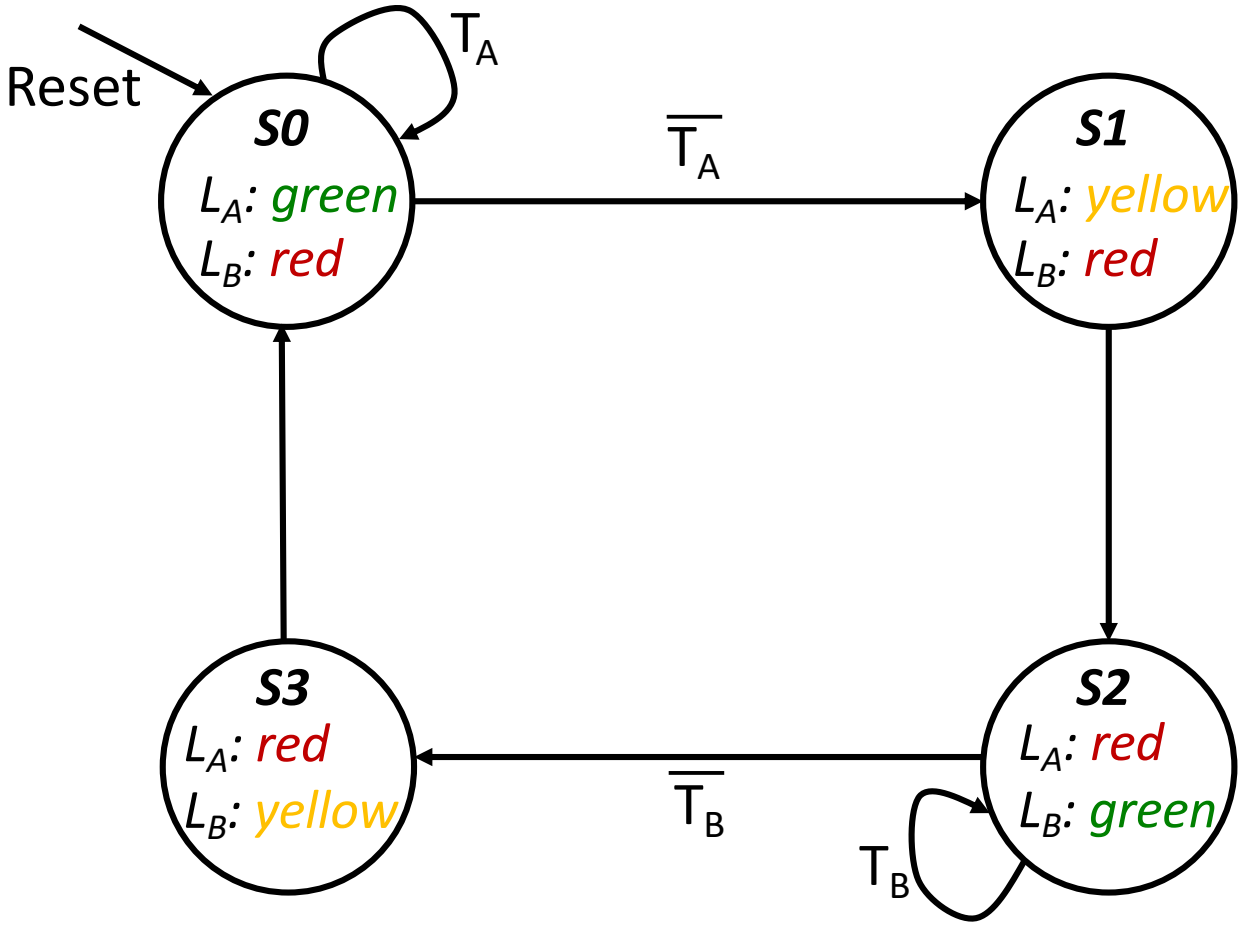
Step # 1: State transition diagram

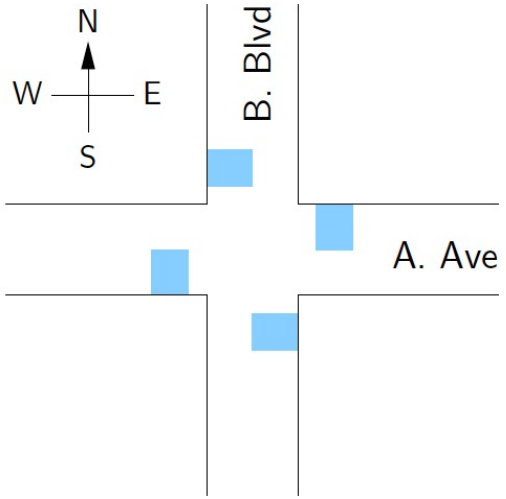
- Formalize the specification and remove ambiguity

State Transition Diagram

- A state transition diagram graphically depicts
  - States with circles
  - Transitions (rising edge) with arcs
  - How does inputs affect the transitions?
  - How are outputs related to the current state?

State Transition Diagram





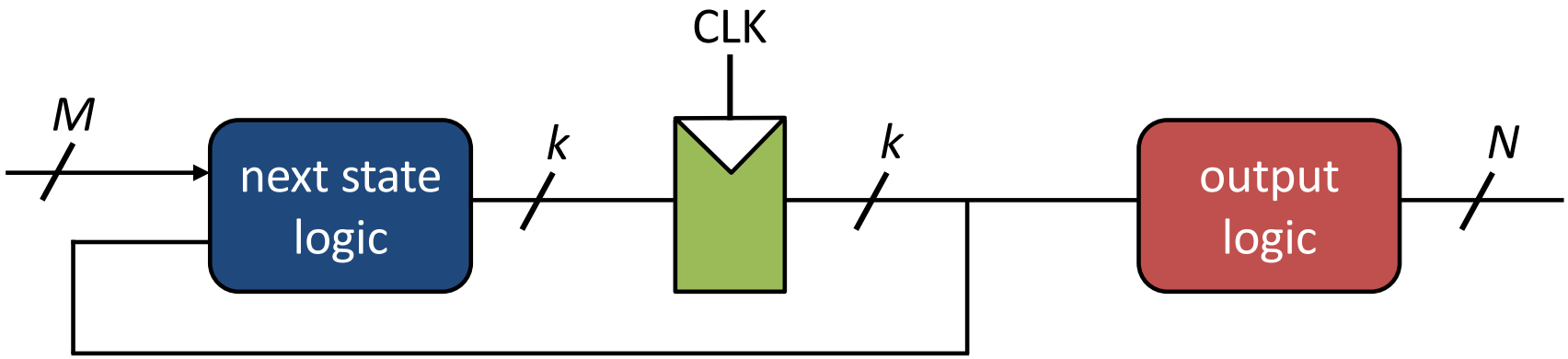
Implementing FSMs

Step # 1: State transition diagram ✓

- Formalize the specification and remove ambiguity

Step # 2: Derive the next state logic

- Binary encoding for states
- State transition (truth) table
- Minimized Boolean equations for next state logic

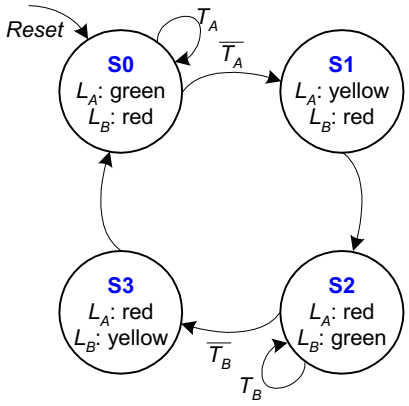


State Transition Table

- To move a step closer from specification to circuit design, a table listing the next states for all possible combinations of current state and input signals is helpful
  - Use X for don't cares to help keep the size of the table manageable

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	X	X	S0



Encoding States

- There is a problem. Circuits do not understand S0, S1, ...
- We need to store 0's and 1's in our state register
- Therefore, we need an encoding scheme for our states

State	Encoding
S0	00
S1	01
S2	10
S3	11

Note: S0 – S3 for states and S_0 and S_1 for bits in the state register

New State Transition Table

- We can substitute our state encodings into the state transition table
- The new version of the state transition table completely specifies the next state as a combinational logic function of the current state and input variables

State Transition Table with Binary Encoding

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

Whiteboard: S_1' Derivation

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

State Transition Table with Binary Encoding

State	Encoding
S0	00
S1	01
S2	10
S3	11

$$S'_1 = S_1 \oplus S_0$$
$$S'_0 = \overline{S_1}\,\overline{S_0}\,\overline{T_A} + S_1\overline{S_0}\,\overline{T_B}$$

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

Implementing FSMs

Step # 1: State transition diagram ✓

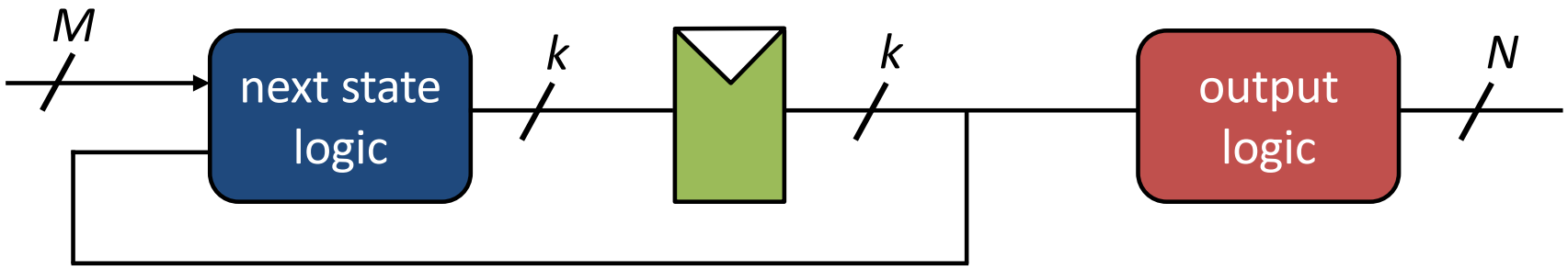
- Formalize the specification and remove ambiguity

Step # 2: Derive the next state logic ✓

- Binary encoding for states
- State transition (truth) table
- Minimized Boolean equations for next state logic

Step # 3: Derive the output logic

- Binary encoding for outputs
- Output table & Boolean equations



Output Encoding & Table

In our Moore FSM, the output only depends on the current state

Output	Encoding
GREEN	00
YELLOW	01
RED	10

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

Output Encoding & Table

In our Moore FSM, the output only depends on the current state

Output	Encoding
GREEN	00
YELLOW	01
RED	10

$$L_{A1} = S_1$$
$$L_{A0} = \overline{S_1} S_0$$
$$L_{B1} = \overline{S_1}$$
$$L_{B0} = S_1 S_0$$

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

Implementing FSMs

Step # 1: State transition diagram ✓

- Formalize the specification and remove ambiguity

Step # 2: Derive the next state logic ✓

- Binary encoding for states
- State transition (truth) table
- Minimized Boolean equations for next state logic

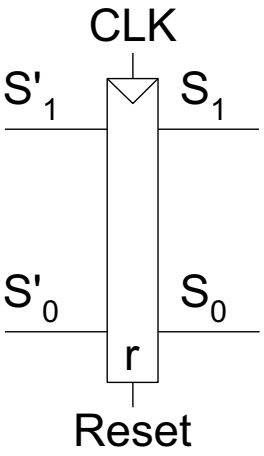
Step # 3: Derive the output logic ✓

- Binary encoding for outputs
- Output table & Boolean equations

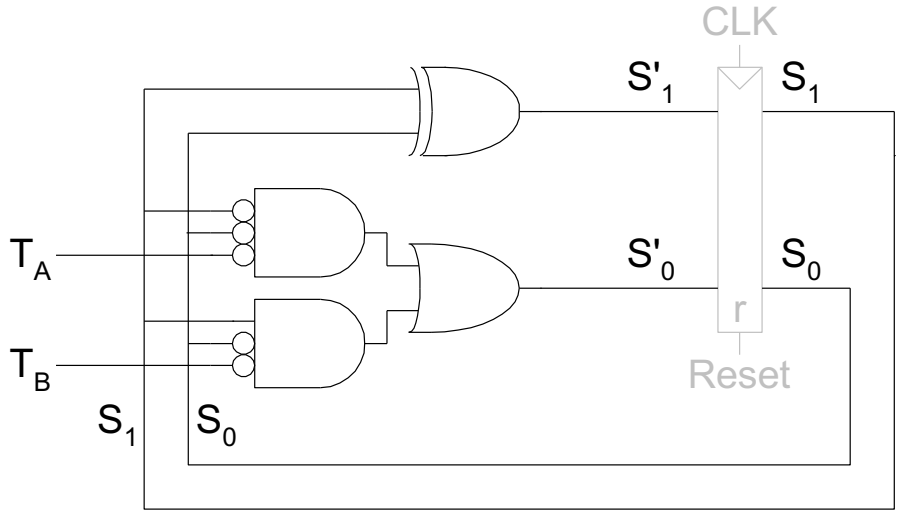
Step # 4: Turn the Boolean equations into logic gate implementation

- Next state logic & output logic

State Machine Circuit

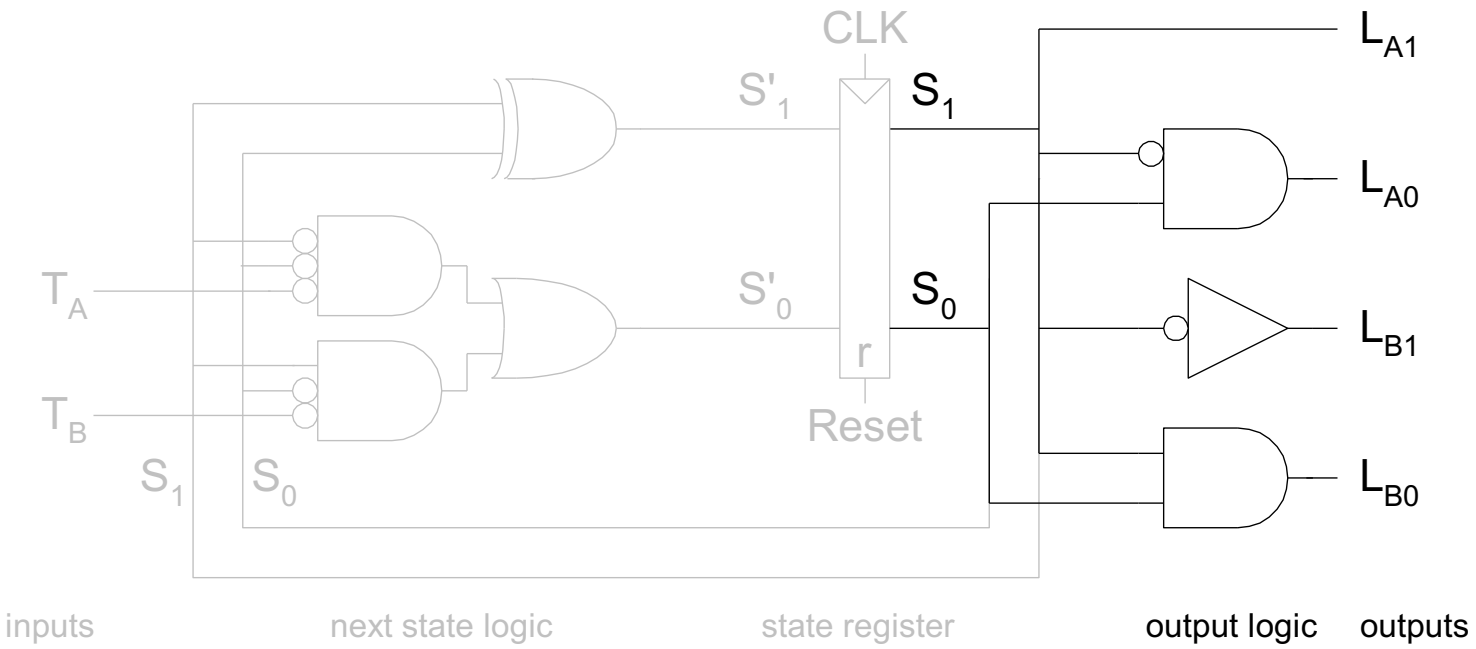


state register



inputs next state logic state register

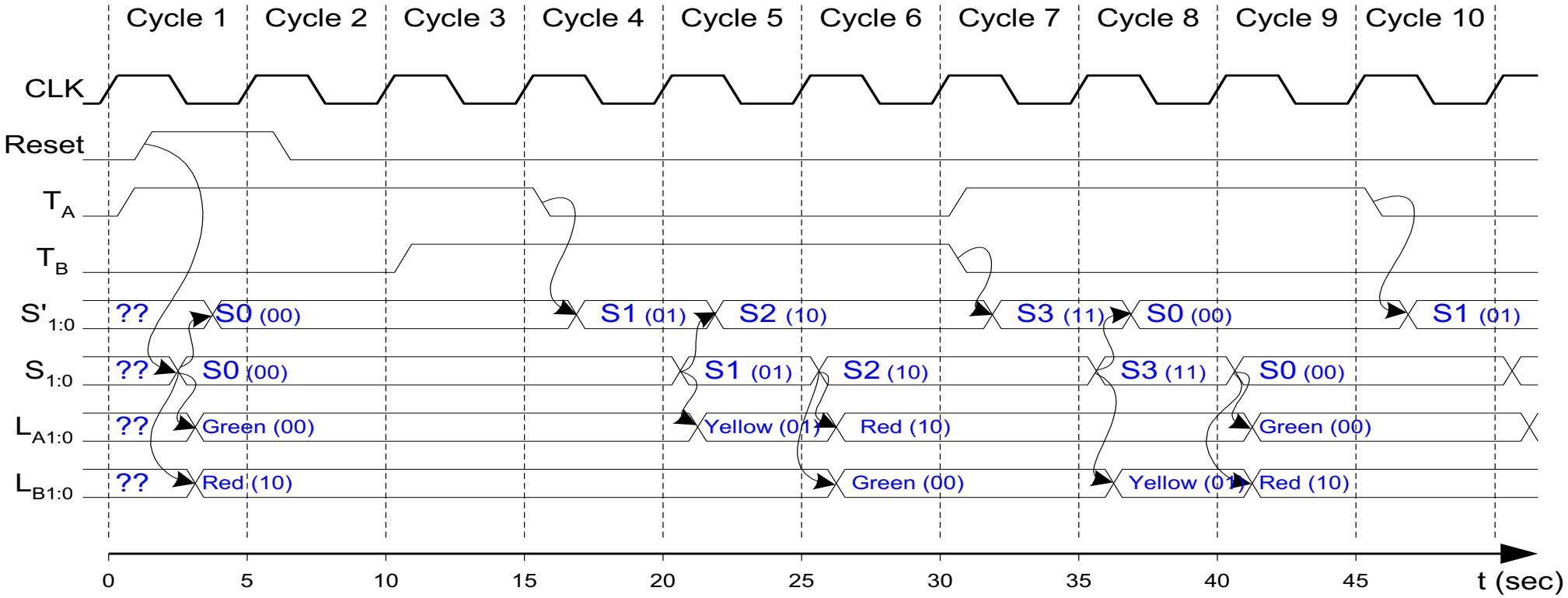
State Machine Circuit



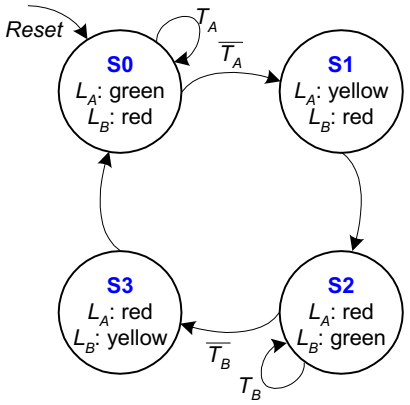
Note

Moore FSM: The outputs depend on the current state alone (e.g., traffic light controller)

Mealy FSM: The outputs depend on the current state and the inputs (simple extension to Moore FSM)

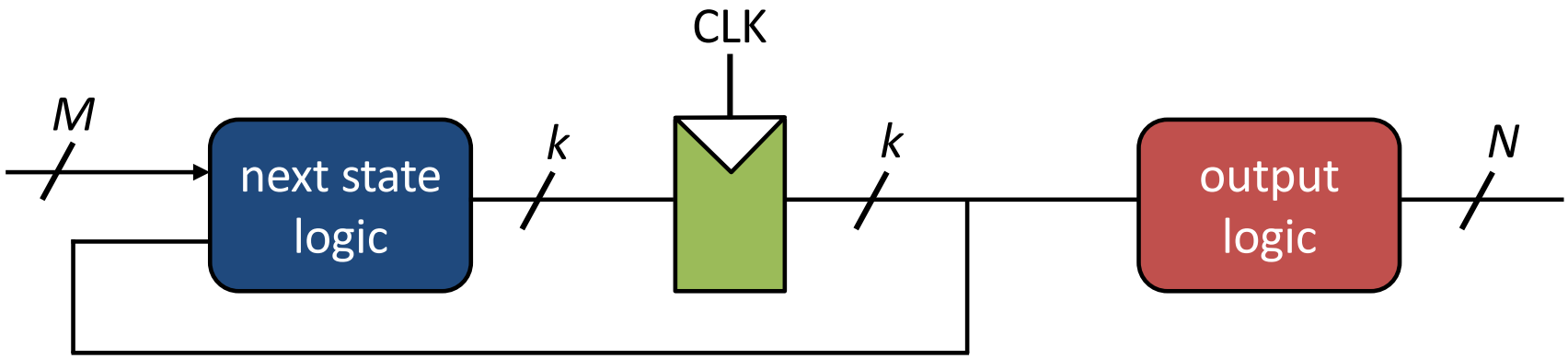


Timing Diagram



Quiz – 1

- Suppose the next state logic has a delay of 5.5 seconds. Will the traffic light controller work correctly?
- *What is the big lesson here regarding the minimum clock period (max. frequency) in synchronous sequential circuits?*



Quiz – 2

- What happens if we use a D latches to store the next state instead of D flipflops?
  - *Hint: The timing diagram can give insight*