Disclaimer

The slides presented here are a combination of the CS251 course notes from previous terms, the work of Xiao-Bo Li, and material from the required textbook "Computer Organization and Design, ARM Edition," by David A. Patterson and John L. Hennessy. It is being used here with explicit permission from the authors.

CS251 course policy requires students to delete all course files after the term. Therefore, please do not post these slides to any website or share them.

CS251 - Computer Organization and Design

Digital Logic Design - Sequential Logic

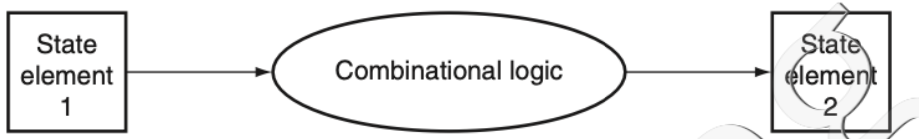
Instructor: Zille Huma Kamal

University of Waterloo

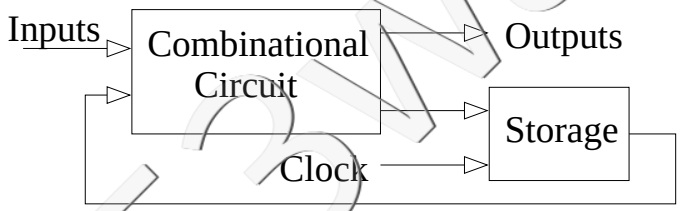
Spring 2023

Clocks and Sequential Circuits

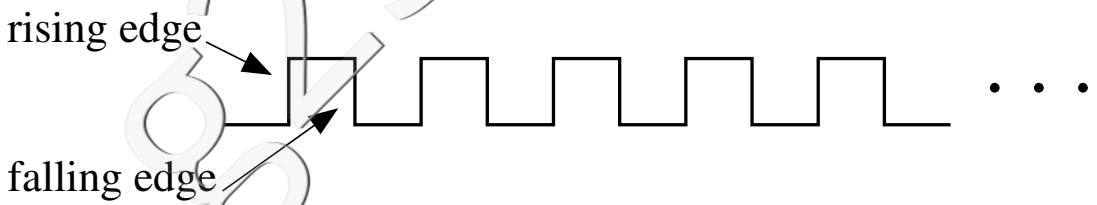
- A sequential circuit has a storage (state) element



- Synchronous: has a clock and storage (memory) changes only at discrete points in time



Clock pulse:

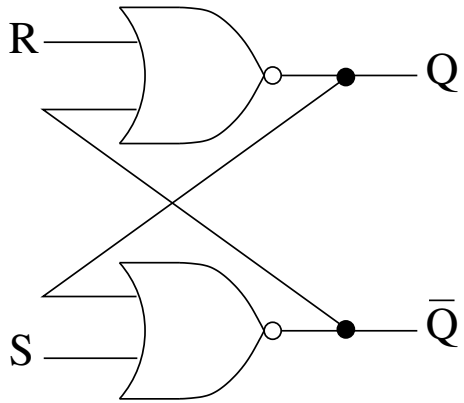


Easier to analyze, tends to be more stable

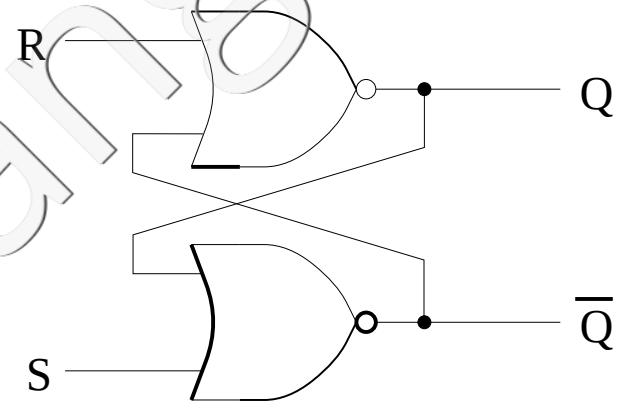
- Asynchronous: no clock, potentially faster and less power-hungry, but harder to design and analyze

SR Latch with NOR gates

- SR Latch with NOR gates



(bad circuit drawing style)



(good circuit drawing style)

Think About It

Can you derive a truth table for outputs Q and $\bar{Q}$?

Functional Description of SR Latch

- Truth table for SR latch

S	R	Q	$\bar{Q}$	
0	0	Q	$\bar{Q}$	Latch state (no change)
0	1	0	1	Reset state
1	0	1	0	Set state
1	1	?	?	Undefined

- Advantages:
  - Can "remember" value
  - Natural "reset" and "set" signals (SR=01 is "reset" to 0, SR=10 is "set" to 1)
- Disadvantages:
  - SR=11 input has to be avoided
  - No notion of a clock or change at discrete points in time yet

Improve the SR Latch

Idea: Use AND gates with clock. When clock is off (signal is 0), the AND gate outputs 0. The other input to this AND gate is called D, which is the "data" we want to remember. This circuit is called the **D-Latch**.

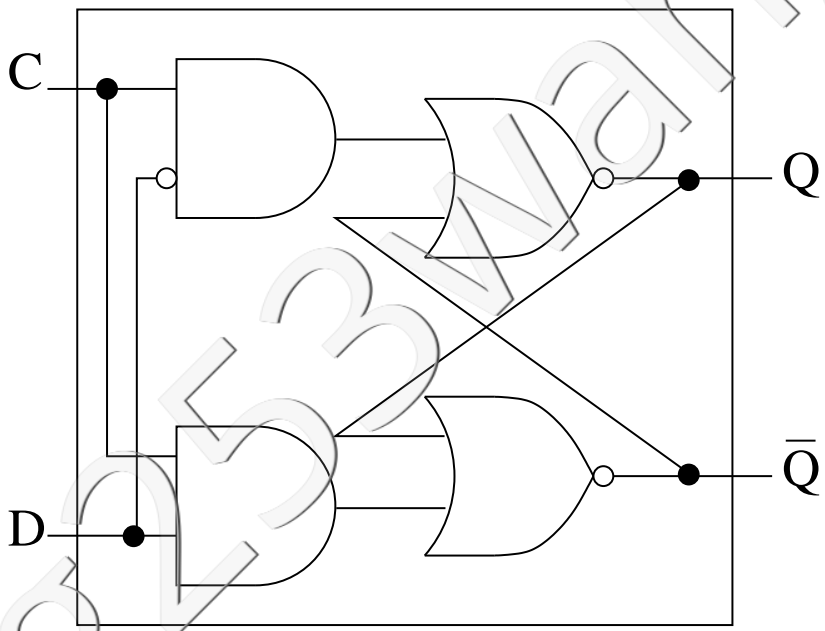
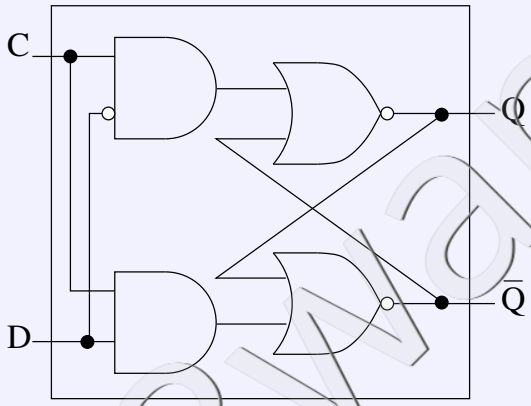


Figure: D Latch

The D Latch

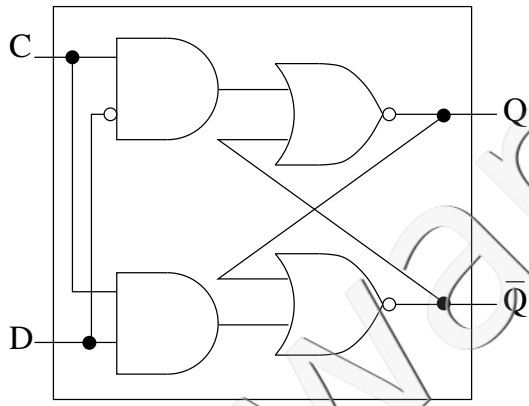
Think About It



Complete the truth table for the D latch:

C	D	Next state of Q
0	0	
0	1	
1	0	
1	1	

Solution: The D Latch Truth Table

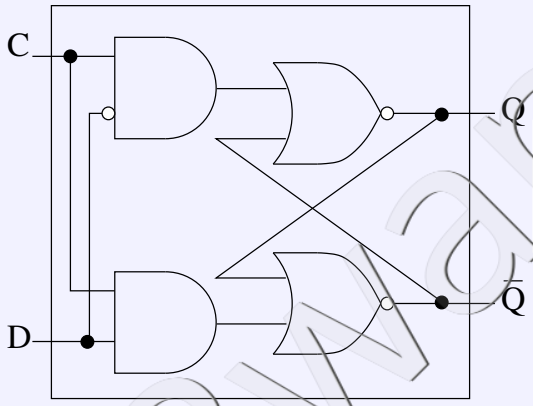


Complete the truth table for the D latch:

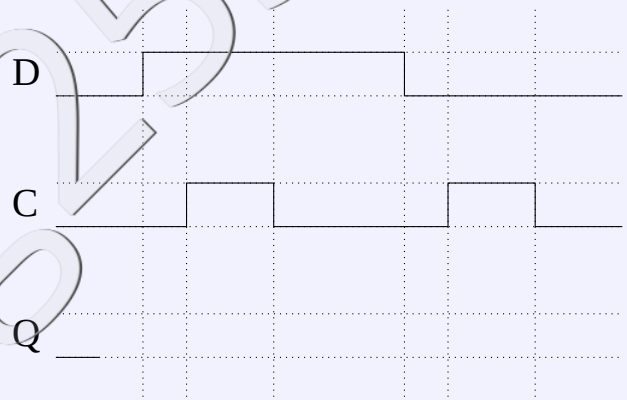
C	D	Next state of Q
0	X	No change
1	0	$Q = 0$ (Reset)
1	1	$Q = 1$ (Set)

The D Latch Signal

Think About It



Trace the output signal Q, given the input C and D.



Solution: D Latch Signal

While $C = 1$, then $Q = D$. While $C = 0$, Q is whatever it was before.

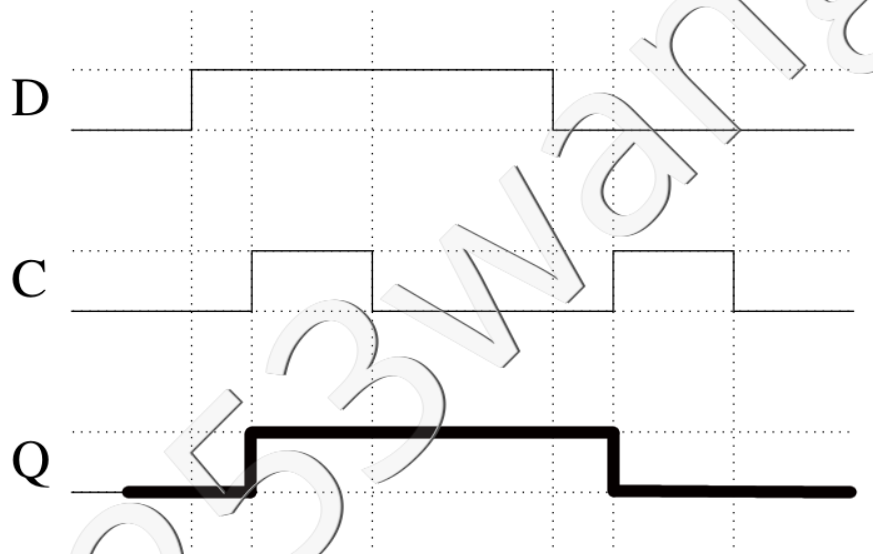
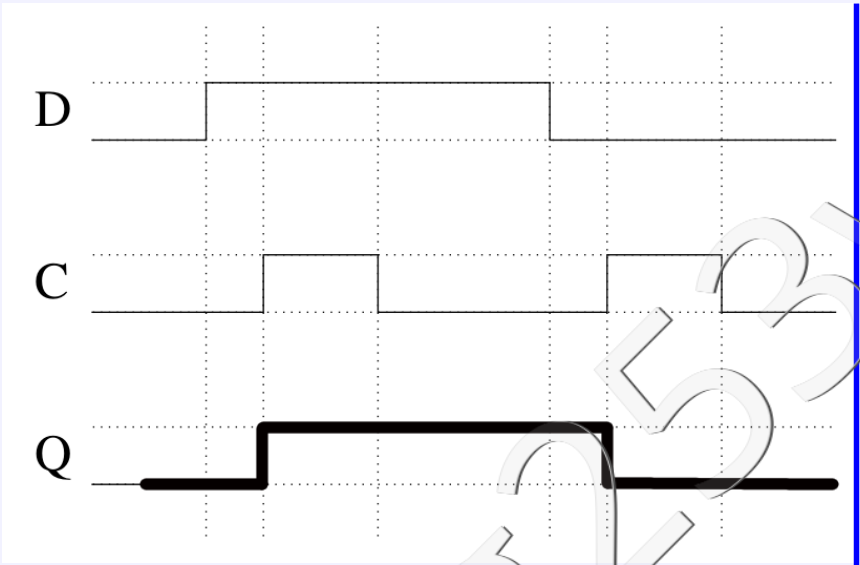


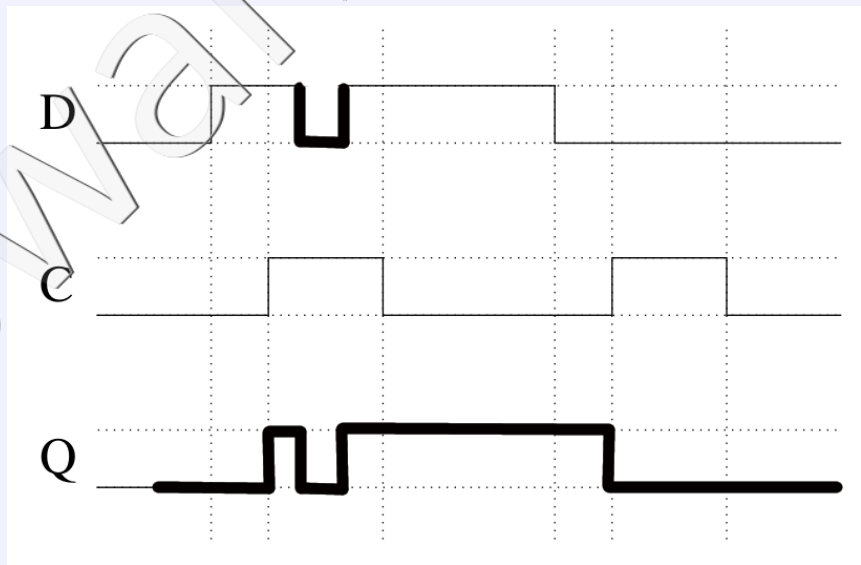
Figure: D latch signal.

D Latch Problem

Think About It

While $C = 1$, D can change Q.

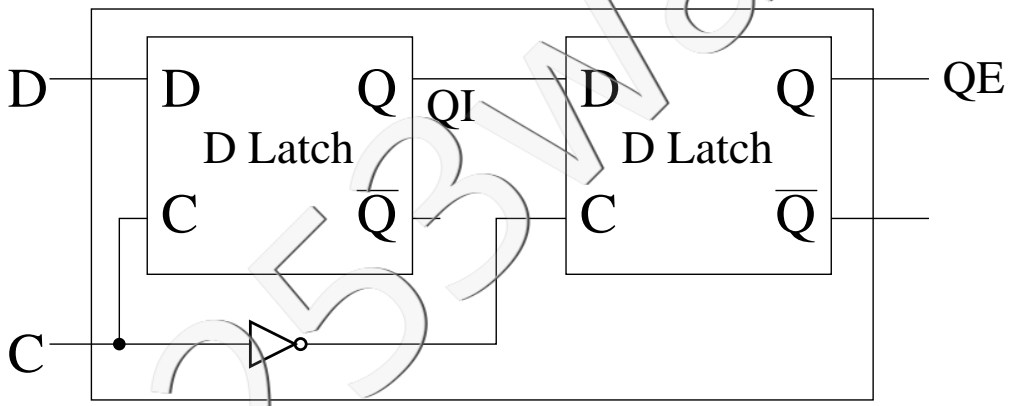




Is this what we want? **No**.

The D Flip-Flop

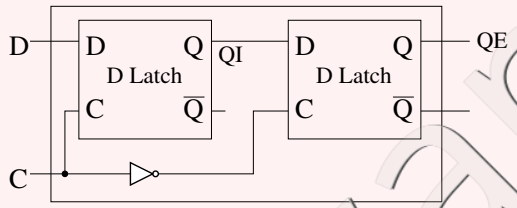
- We want state to be affected only at discrete points in time; a master-slave design achieves this.



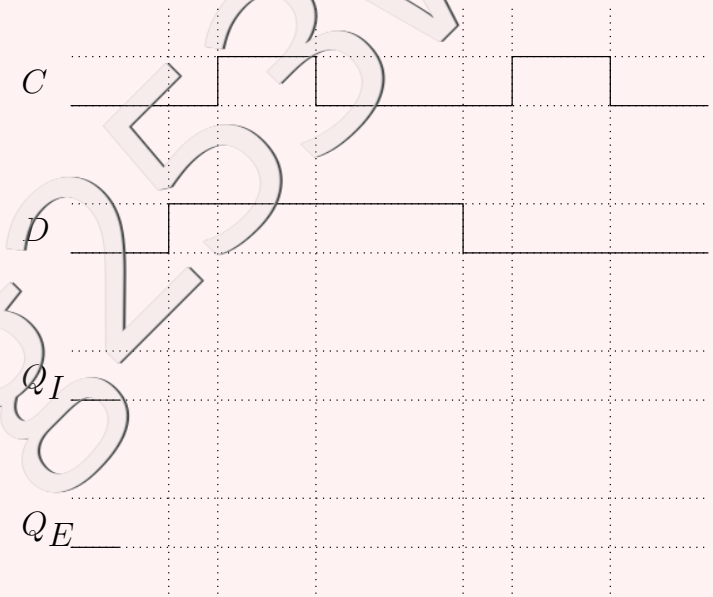
Example: D Flip-Flop

Try this

For the circuit illsutrated:



Draw the resulting traces for Q_I and Q_E, given the trace for inputs C and D.



Solution: D Flip-Flop

- While $C = 1$, $\overline{C} = 0$, Q_I can change with D, but Q_E ignores Q_I.
- While $C = 0$, $\overline{C} = 1$, Q_I remembers its previous state and sends that to Q_E.
  - Q_I cannot change, therefore, Q_E also cannot change.

Therefore, Q_E **only** changes when C changes from $C = 1$ to $C = 0$.

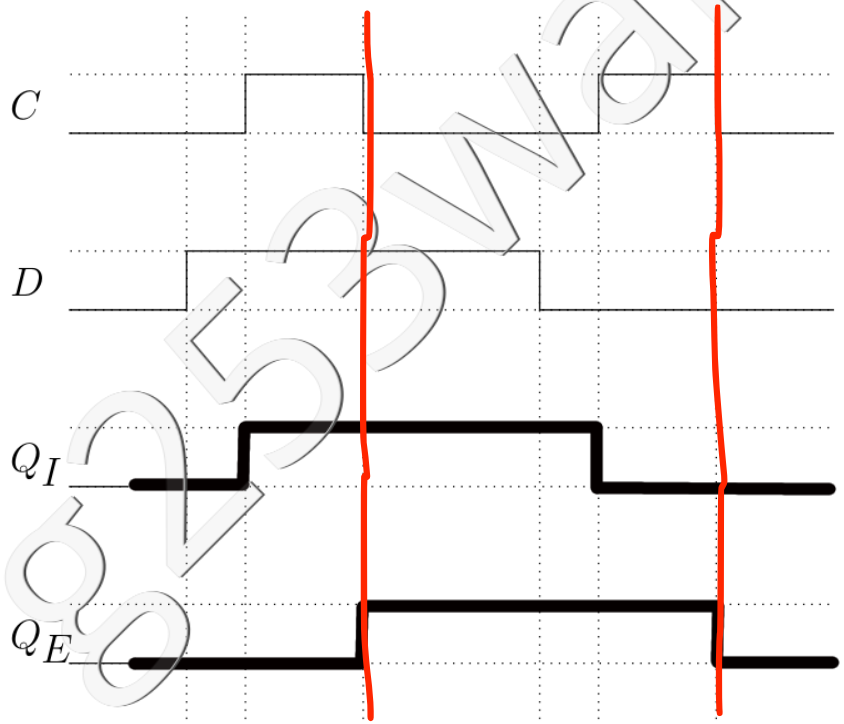
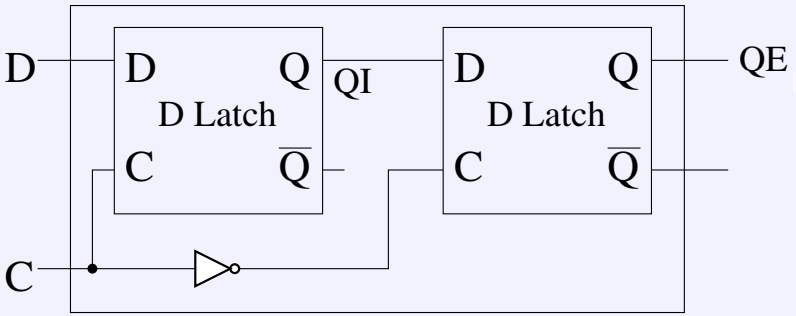


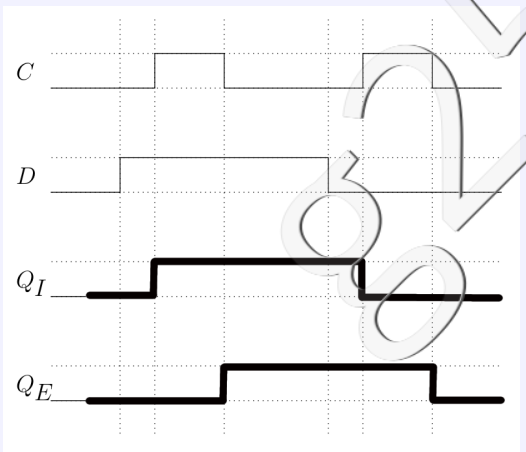
Figure: D flip-flop signal.

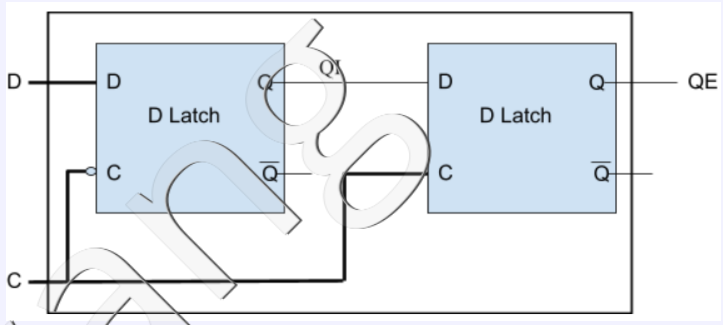
Edge-triggered D Flip-Flop

Think About It

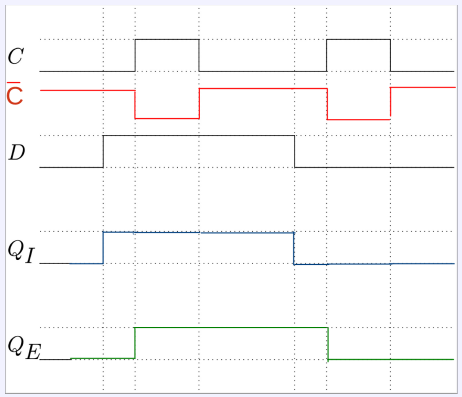


This D flip-flop **only** passes the value of D to Q_E on the **falling-edge** of the **C** (clock).





When inverted clock is input to the D flip-flop, it **only** passes the value of D to Q_E on the **rising-edge** of the **C** (clock).



D Flip-Flop Transistor Count

One D flip-flop uses two D latches and one NOT gate.

- One D latch has the following transistors:
  - NOR gate: 4 transistors (2)
  - AND gate: 6 transistors (2)
  - NOT gate: 2 transistors (1)
- Total transistors in a D Latch: $(4+6) \times 2 + 2 = 22$.

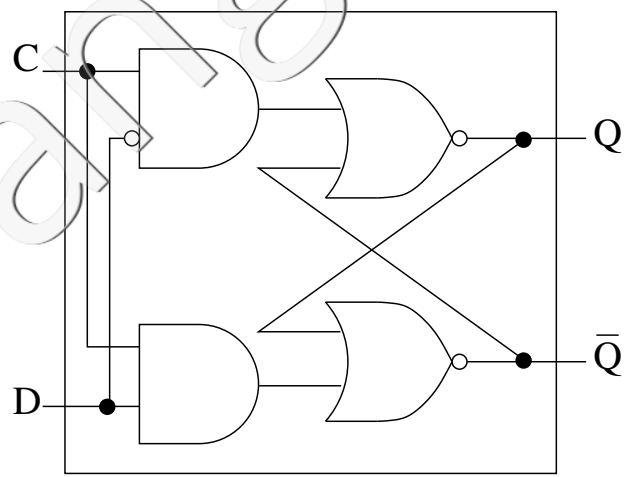


Figure: D latch.

Therefore, in a D flip-flop, total transistors: $2 \times 22 + 2 = 46$

Textbook Readings

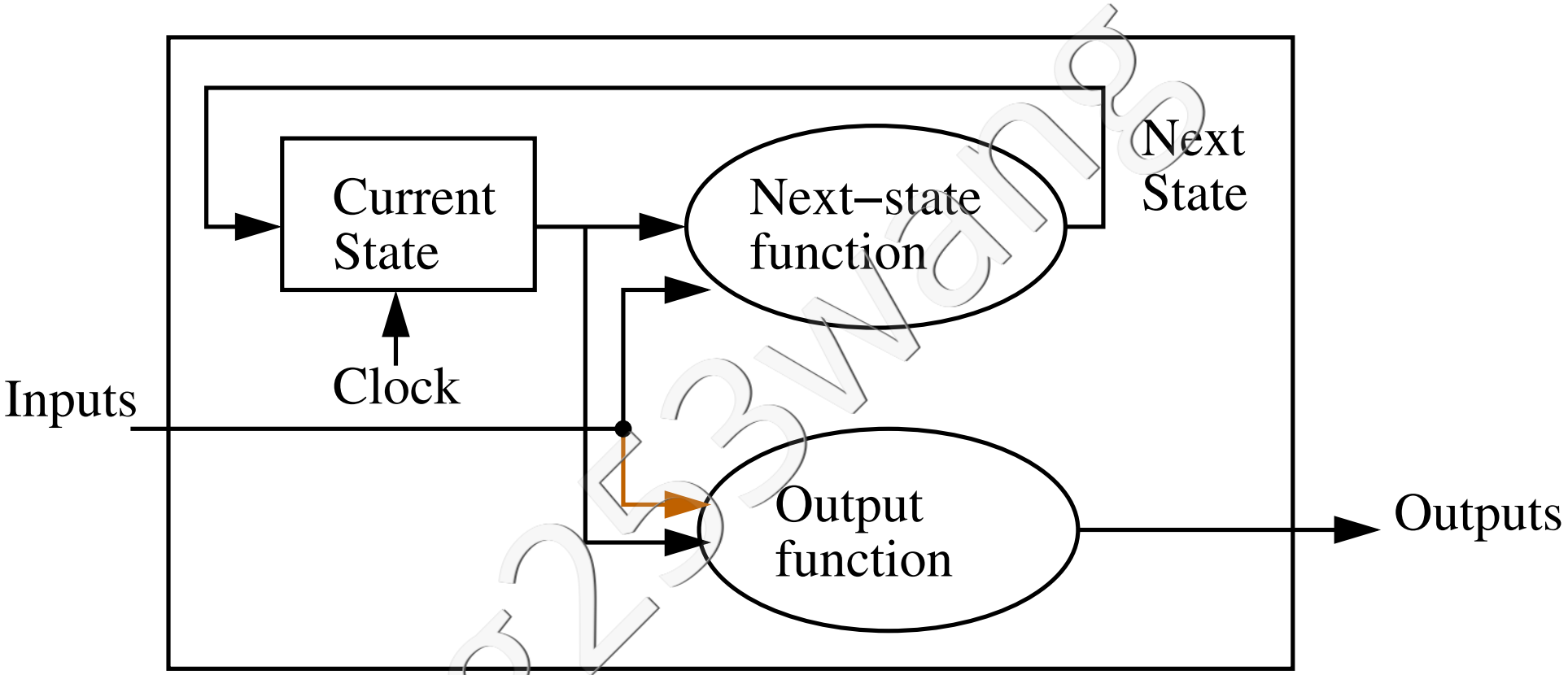
- Section 4.2
- Appendix A.7
- Appendix A.8

Analyzing Sequential Circuits

To design a sophisticated sequential circuit:

- generate functional descriptions
  - use finite state machines to help

Designing Using Finite-State Machines



High-level circuit implementation of finite-state machine

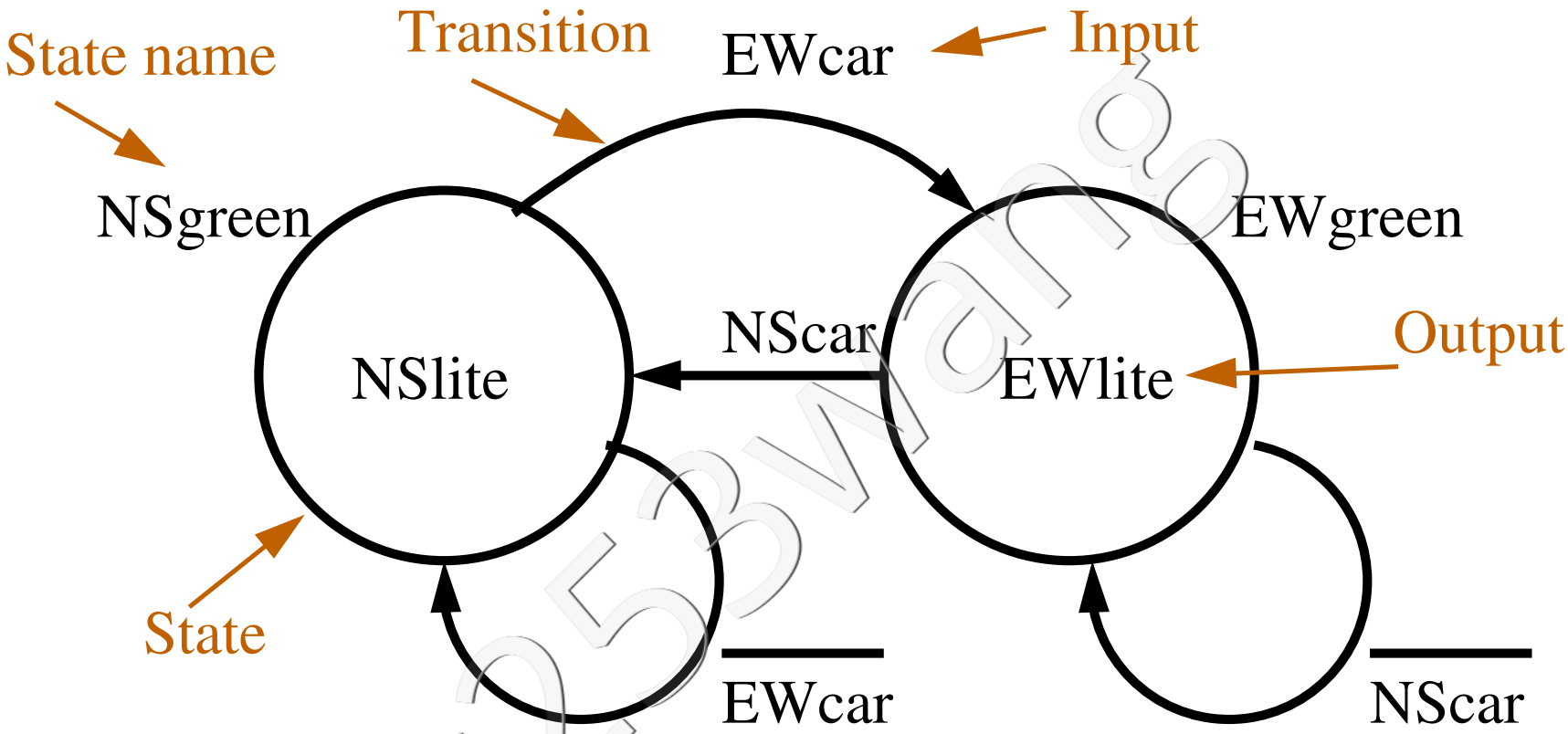
Variations on Finite-State Machines

- Moore machine: output depends only on state (what we use)
- Mealy machine: output can depend on inputs
- Moore machine may be faster, Mealy machine may be smaller
- Conceptually, computation is infinite (input streams have no beginning or end)
- In practice, need to worry about power-up and power-down (as with all our state devices)
- Different in language-recognition context (e.g. CS 241)
  - Input is single character at a time, not set of bits
  - Because strings have finite length, computation is finite (start state, final states)
  - Mealy machines used (outputs on transition arcs)

Example: Simple Traffic Light Controller

- Output signals: NSlight, EWlight
- Input signals: NScar, EWcar
- State names: NSgreen, EWgreen (no yellow for now)
- Functionality: want light to change only if car is waiting at red light

Graphical Representation of Traffic Light Controller



- Names of states outside ovals
- Output in given state inside oval
- Transition arc labelled with Boolean formula of inputs

Next State Table

We want a Boolean expression for next state S'.

S	$EWcar$	$NScar$	S'
0	0	X	0
0	1	X	1
1	X	0	1
1	X	1	0

Check when $S' = 1$:

$$S' = \overline{S} \cdot EWcar + S \cdot \overline{NScar}$$

Output Table

The outputs have the following Boolean expressions.

- In state *NSgreen*, this is $S = 0$, $NSlite = 1$.

$$NSlite = \overline{S}$$

- In state *EWgreen*, this is $S = 1$, $EWlite = 1$.

$$EWlite = S$$

The outputs are a function of the states only, this is a Moore Machine.

S	$NSlite$	$EWlite$
0	1	0
1	0	1

State Circuit

The state is 0 or 1, so use one flip-flop to remember this.

- The input D to the flip-flop is the expression for S'.
- The flip-flop has $Q = S$ and $\bar{Q} = \bar{S}$.
  - Connect these outputs to the inputs in the expression for S'.
  - Also use these S and $\bar{S}$ from the flip-flop as inputs to the circuit for the outputs *NSlite* and *EWlite* in the Output Table.

Every clock cycle, S' is calculated by the circuit, and S and S' are updated on the clock edge. Once S and S' updates, the outputs, *NSlite* and *EWlite* updates.

Electronic Implementation of Finite-State Controller

Try this

Can you use a PLA to build the traffic light controller?

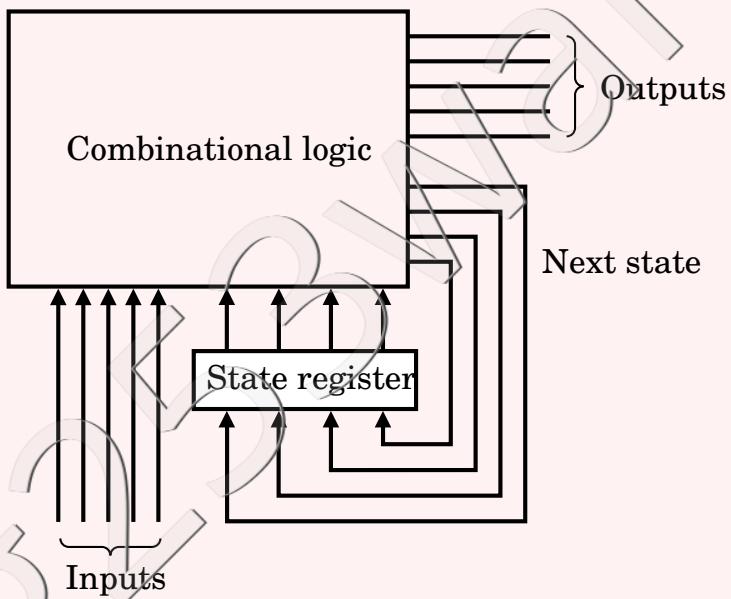
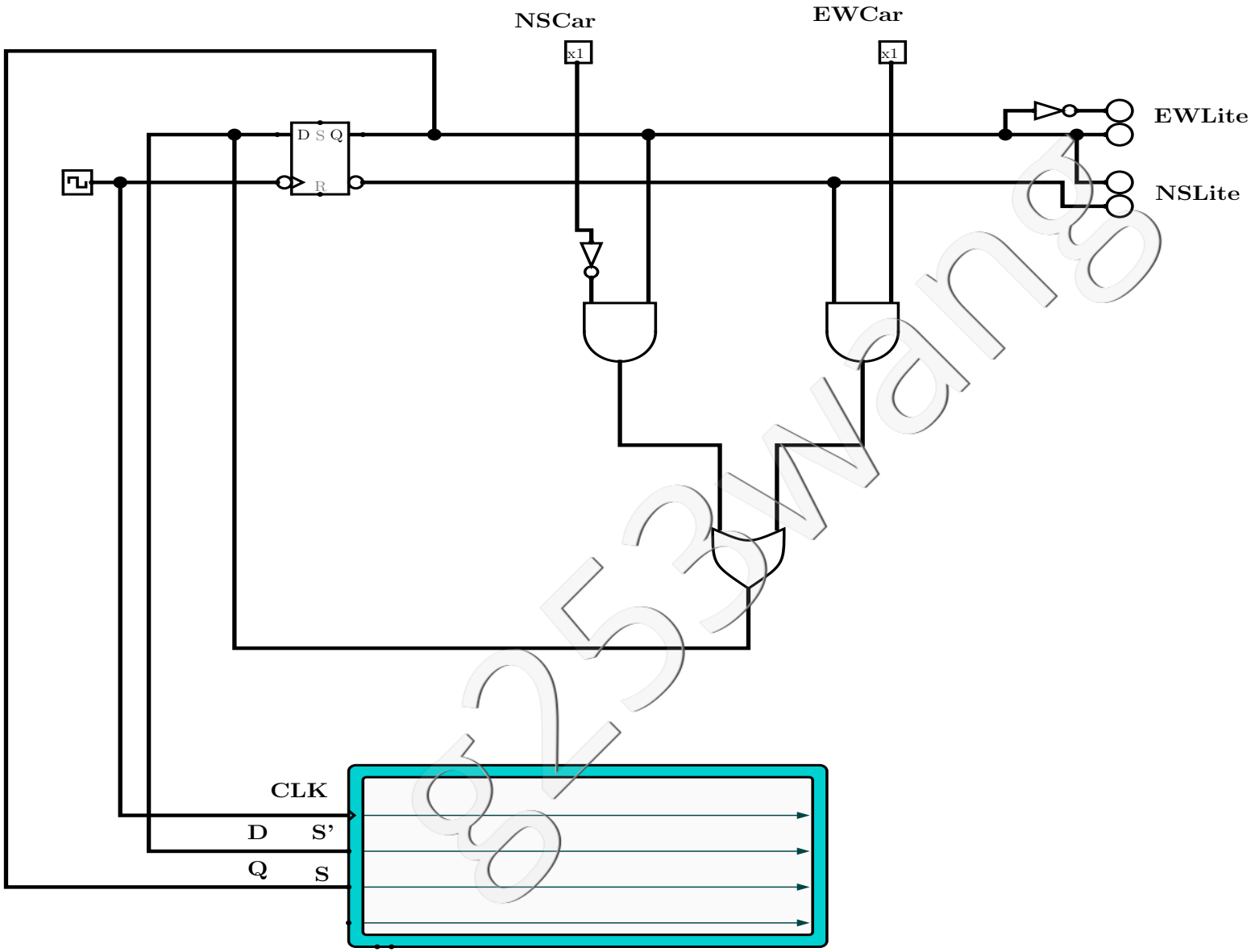


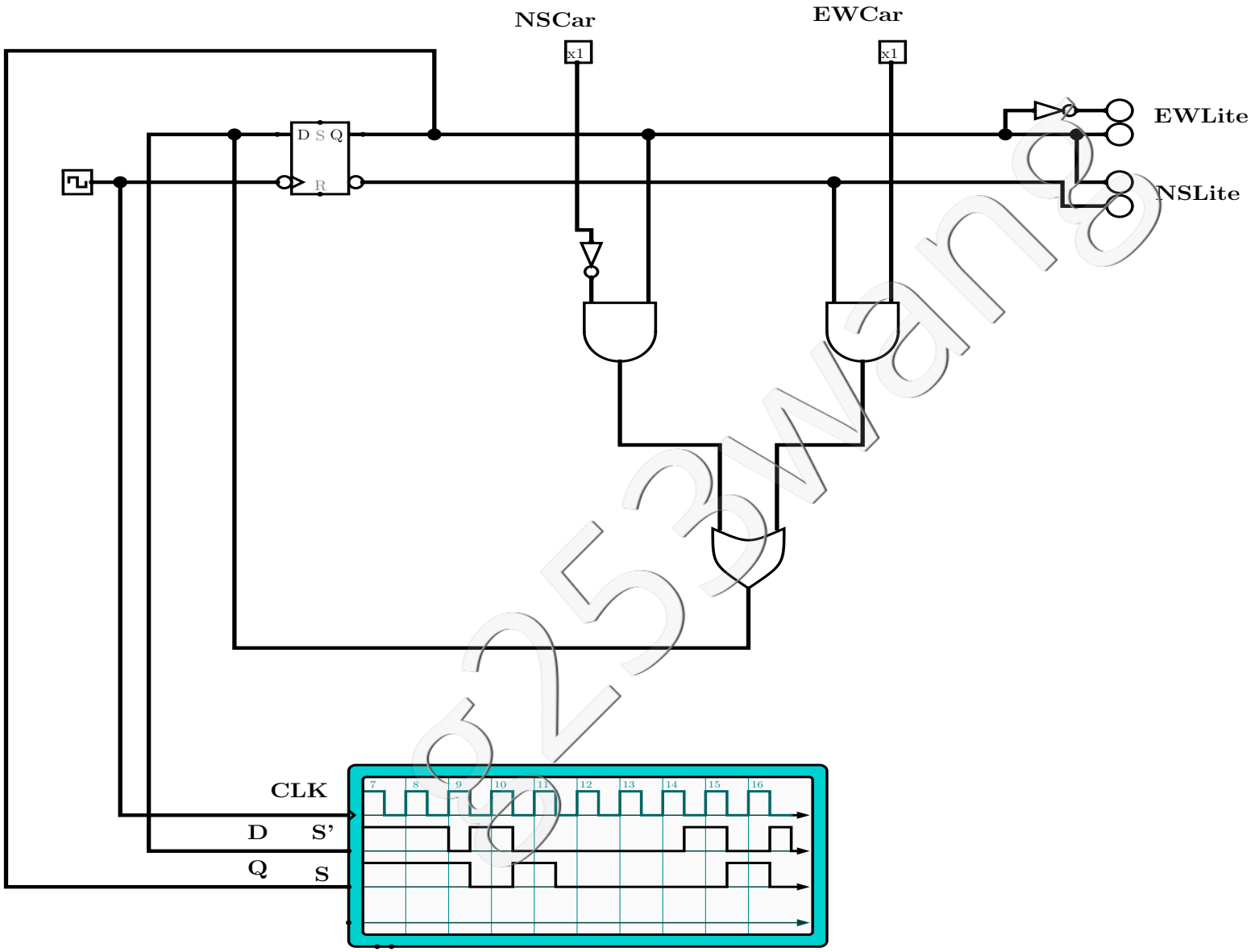
Figure A.10.3 from the text

Solution: Simple Traffic Controller



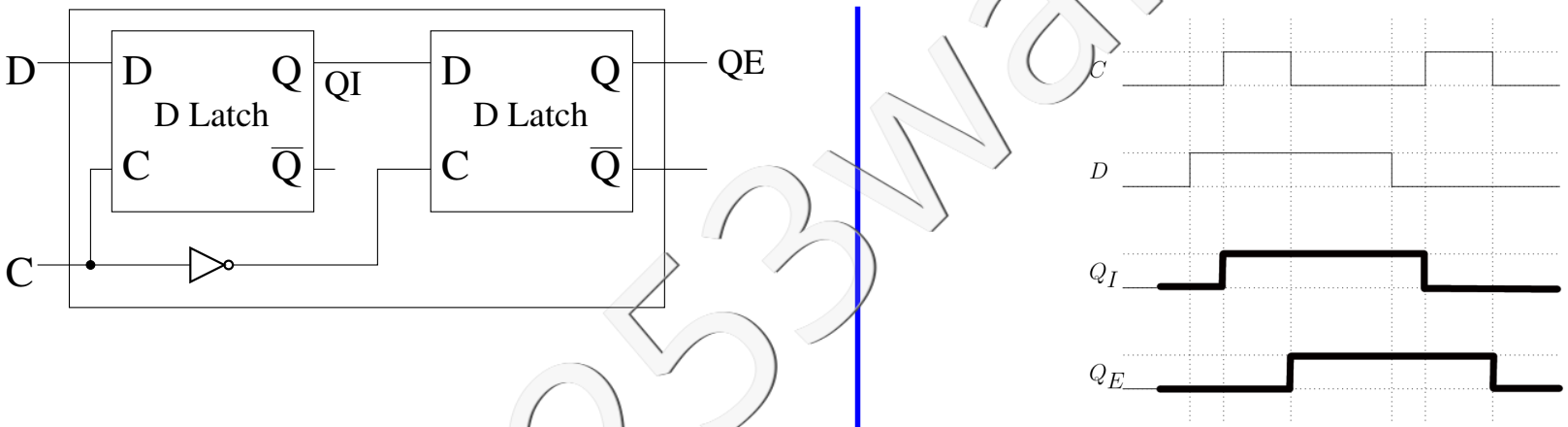
- combinational logic is usually implemented via a PLA or ROM (more on this later).

Solution: Simple Traffic Controller



Recall: D Flip-Flop

Unlike Latches, flip-flops are **not** transparent. Flip-flops only change on the clock edge (e.g. falling edge), not the entire time $C = 1$.

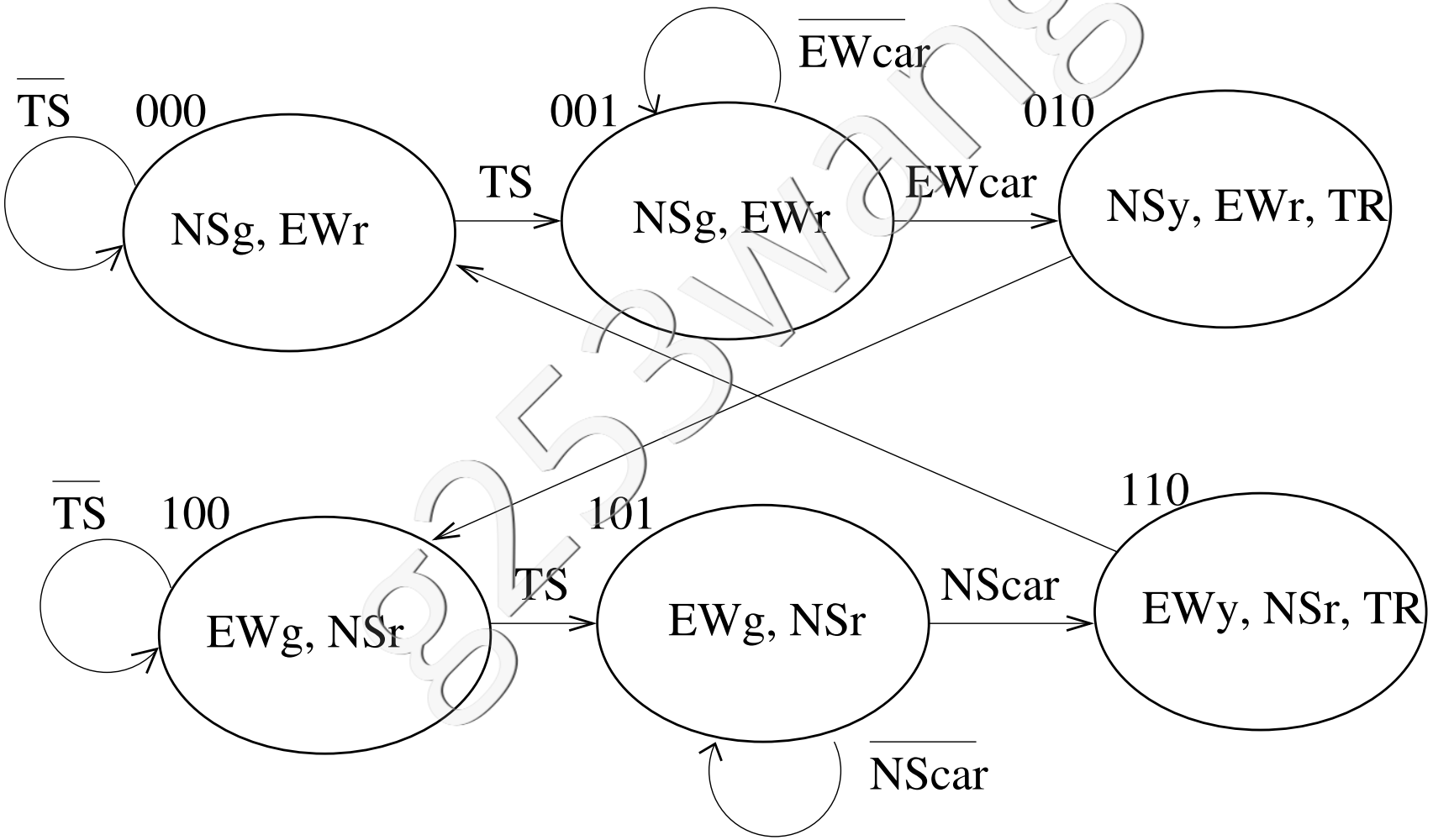


Extending the Traffic-Light Controller

- Behaviour of system
  - Stay green in one direction (red in the other direction) until car arrives or **32** seconds elapse, whichever happens last
  - Green turns to yellow for 4 seconds; red in other direction stays
  - Yellow turns to red, red in other direction turns to green
- Assume 0.25Hz clock $\Rightarrow \frac{1}{0.25} = 4$ seconds.
- Extended traffic controller needs:
  - A 4-second yellow light
  - A 28-second timer
    - ★ Timer input: TimerReset (TR)
    - ★ Timer output: TimerSignal (TS)

State Diagram of Extended Traffic Light Controller

- Inputs: NScar, EWcar, TS
- Outputs: NSg, NSy, NSr, EWg, EWy, EWr, TR



Next-State Table for Extended Controller

Think About It

Can you complete a truth table for the extended traffic light controller?

current state	inputs			next state	current state	inputs			next state
$S_2 S_1 S_0$	NS-car	EW-car	TS	$S_2' S_1' S_0'$	$S_2 S_1 S_0$	NS-car	EW-car	TS	$S_2' S_1' S_0'$
0 0 0	X	X	0	0 0 0	1 0 0	X	X	0	1 0 0
0 0 0	X	X	1	0 0 1	1 0 0	X	X	1	1 0 1
0 0 1	X	0	X	0 0 1	1 0 1	0	X	X	1 0 1
0 0 1	X	1	X	0 1 0	1 0 1	1	X	X	1 1 0
0 1 0	X	X	X	1 0 0	1 1 0	X	X	X	0 0 0
0 1 1	X	X	X	X X X	1 1 1	X	X	X	X X X

- unused states
- symmetric entries

Output Table For Extended Traffic Light Controller

- Output table looks like truth table
  Inputs are State, Outputs are Outputs
- Traffic light outputs: NSg, NSy, NSr, EWg, EWy, EWr, TR
- If output listed in State, then 1 in output table
  If output not listed in State, then 0 in output table

Try this

S_2	S_1	S_0	NSg	NSy	NSr	EWg	EWy	EWr	TR
0	0	0							
0	0	1							
0	1	0							
0	1	1							
1	0	0							
1	0	1							
1	1	0							
1	1	1							

Solution: Output Table

The complete output table is below.

S_2	S_1	S_0	NSg	NSy	NSr	EWg	EWy	EWr	TR
0	0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	0	1	0
0	1	0	0	1	0	0	0	1	1
0	1	1	X	X	X	X	X	X	X
1	0	0	0	0	1	1	0	0	0
1	0	1	0	0	1	1	0	0	0
1	1	0	0	0	1	0	1	0	1
1	1	1	X	X	X	X	X	X	X

Next-State/Output Logic For Extended Controller

Current state $= S_2S_1S_0$, next state $= S_2'S_1'S_0'$

$S_0' = \bar{S}_1\bar{S}_0 \cdot TS + \bar{S}_2\bar{S}_1S_0 \cdot EW\bar{c}ar + S_2\bar{S}_1S_0 \cdot N\bar{S}car$

$S_1' = \bar{S}_2\bar{S}_1S_0 \cdot EWcar + S_2\bar{S}_1S_0 \cdot NScar$

$S_2' = \bar{S}_2S_1\bar{S}_0 + S_2\bar{S}_1$

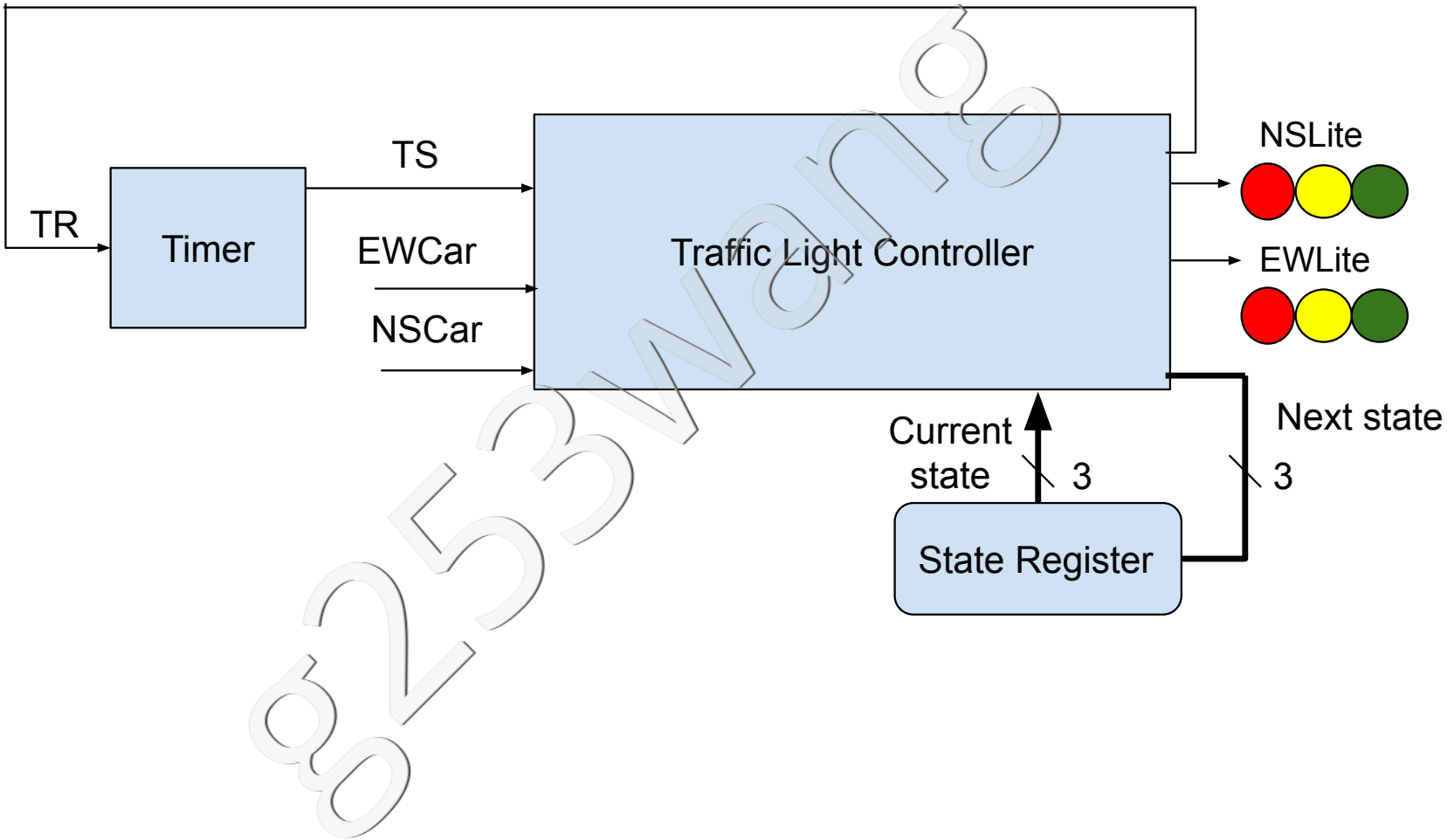
$NSg = \bar{S}_2\bar{S}_1$, $EWg = S_2\bar{S}_1$

$NSy = \bar{S}_2S_1\bar{S}_0$, $EWy = S_2S_1\bar{S}_0$

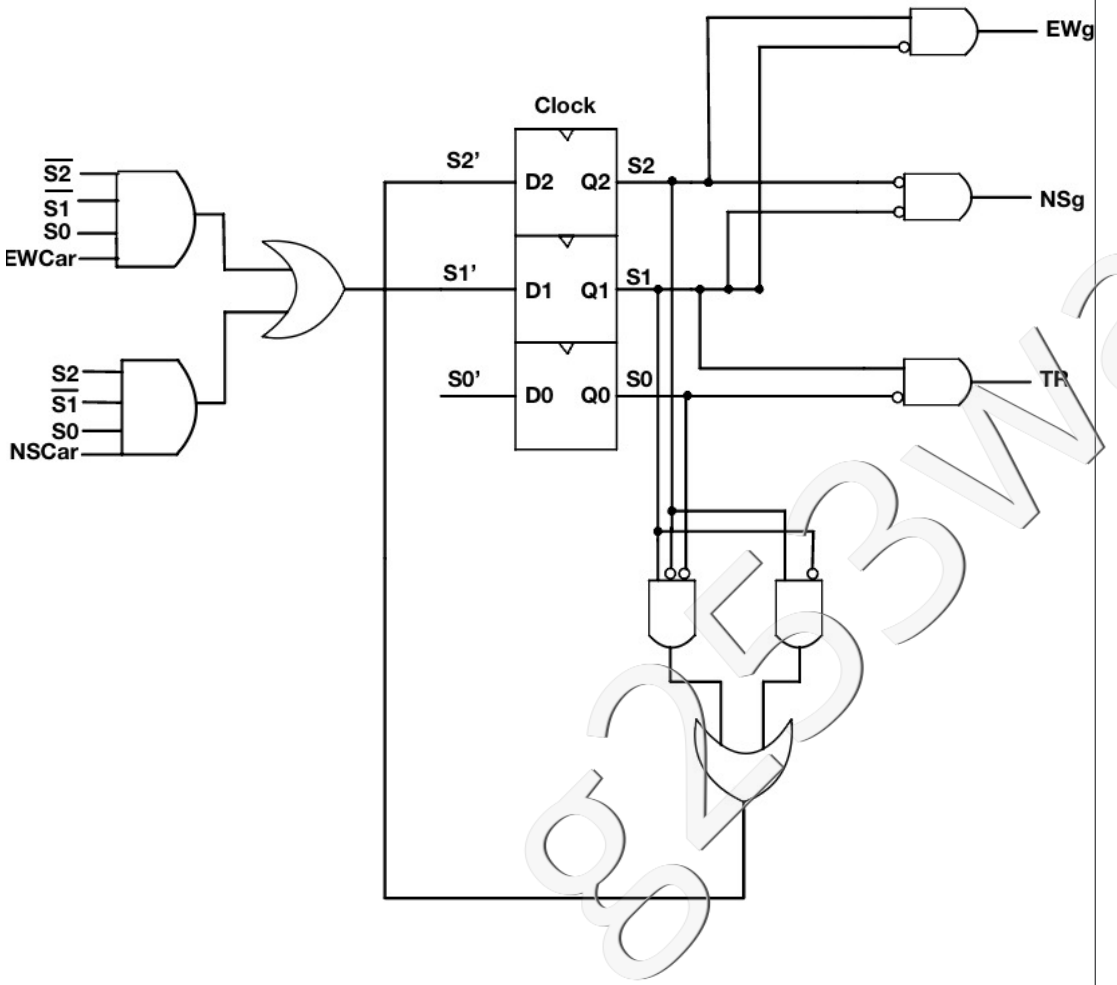
$NSr = S_2$, $EWr = \bar{S}_2$

$TR = S_1\bar{S}_0$

Extended Traffic Light Controller



Extended Traffic Light Controller[1]



[1]Image by Sherlock Yang

Adding Pedestrian Transitions

We can add pedestrians in the NS and EW directions:

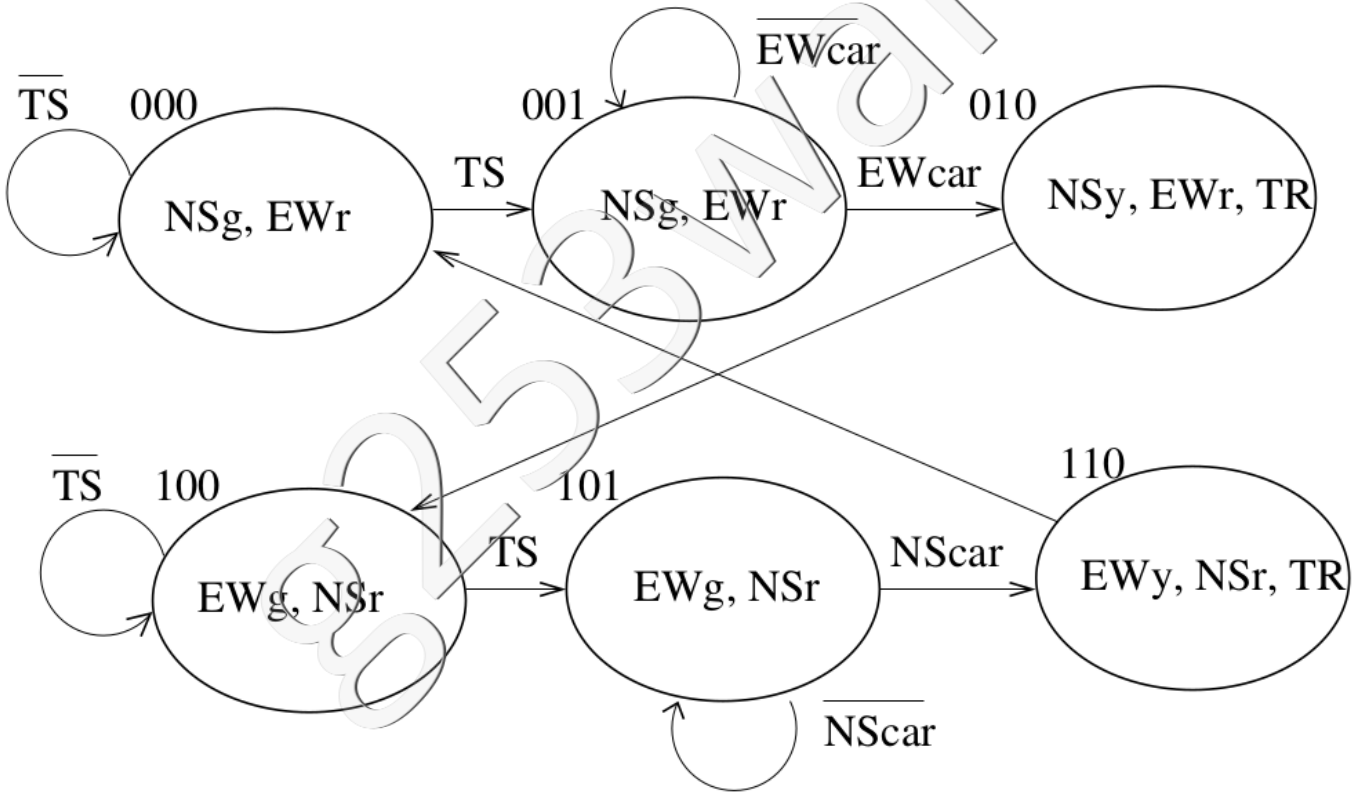
- *NSped*,
- *EWped*.

There is no pedestrian light, just the same traffic light for cars and pedestrians.

Adding Pedestrian Transitions

Consider the state 001 with outputs (NSg, EWr)

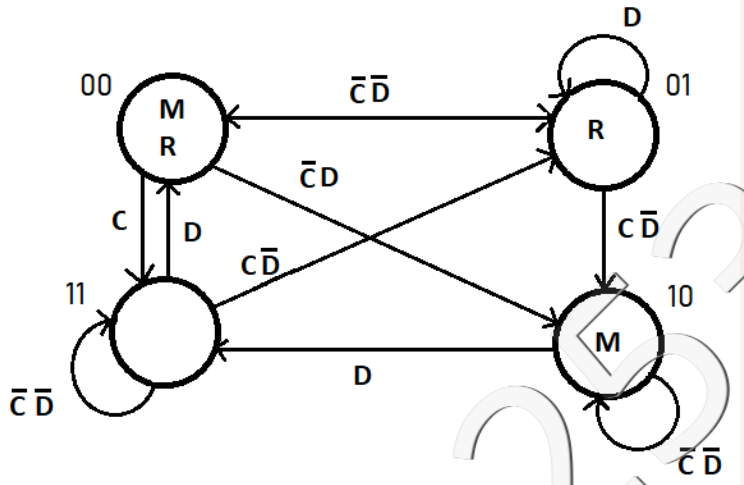
- Leave this state if either a car or pedestrian is waiting : $EWcar + EWped$.
- Stay in this state otherwise: $\overline{EWcar + EWped} = \overline{EWcar} \cdot \overline{EWped}$.



Example: Finite State Machines

Try this

Given the FSM:



Complete the table on a particular sequence of input bits for C and D.

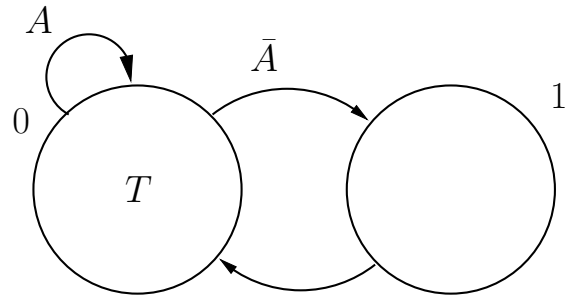
Current State	00	01				
R	1					
M	1					
C	0	1	0	0	1	1
D	0	0	0	1	1	1
Next State	01					

Solution: Trace a Finite State Machine

Current State	00	01	10	10	11	00
R	1	1	0	0	0	1
M	1	0	1	1	0	1
C	0	1	0	0	1	1
D	0	0	0	1	1	1
Next State	01	10	10	11	00	11

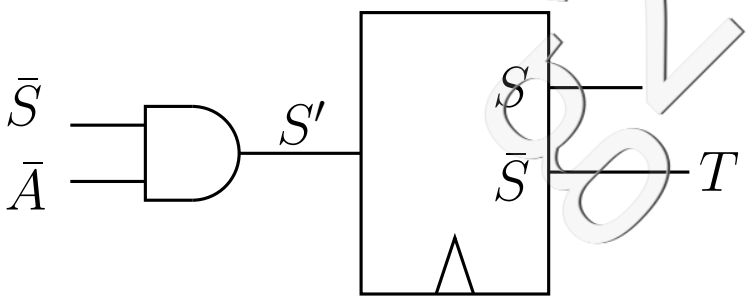
Important Reminder

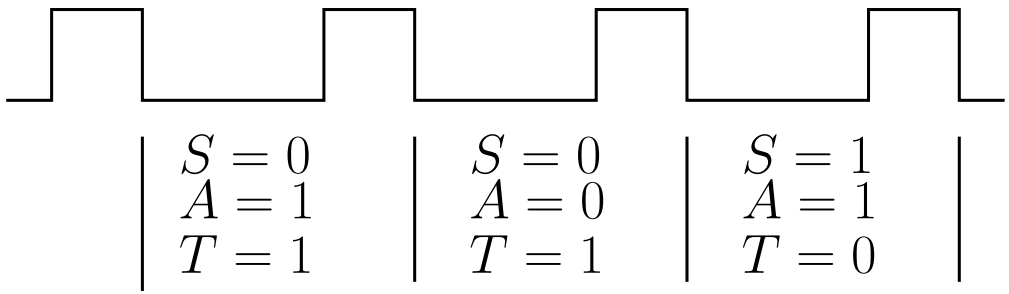
Moore machines, output is based on *current state*



Next State		
S	A	S'
0	0	1
0	1	0
1	X	0

Output	
S	T
0	1
1	0





Textbook Readings

- Appendix A.10