§ 3.1 INTRODUCTION

Sequential logic depends on current and prior input values.

It has memory.

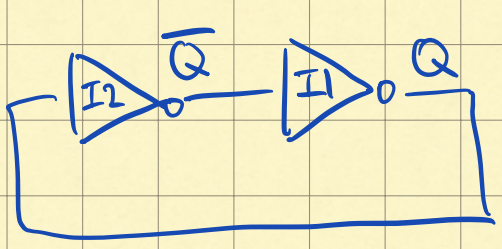
Remember prior inputs or distill into a smaller amount of info.
state

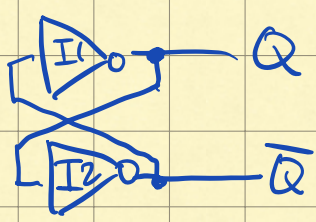
The state of a digital circuit is a set of bits, called state variables.

3.2 LATCHES AND FLIP-FLOPS

The fundamental building block of a memory is a bistable element,

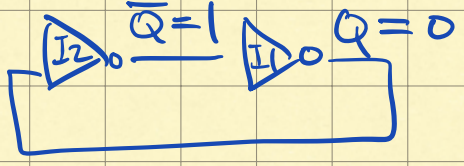
element w/ two stable states.





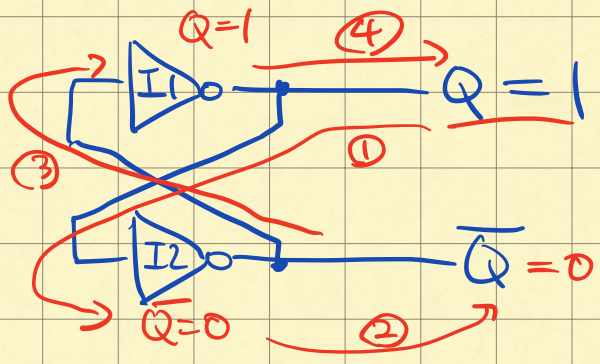
The inverters are cross-coupled.

- Case I: $Q = 0$



This is consistent with the original $Q=0$, so the case is said to be stable.

- Case II: $Q = 1$



Stable

Because the cross-coupled inverters have two stable states, when $Q=0$ and $Q=1$, the circuit is said to be bistable.

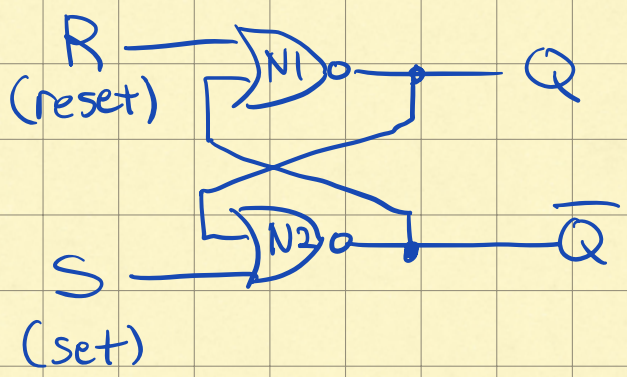
If circuit has a third state, approximately half-way between 0 and 1,

→ metastable

An element with N stable states conveys $\log_2 N$ bits of information.
In this case, 1-bit.

(The cross-coupled inverters can store, but the user can't control.
latches and flip-flops can.)

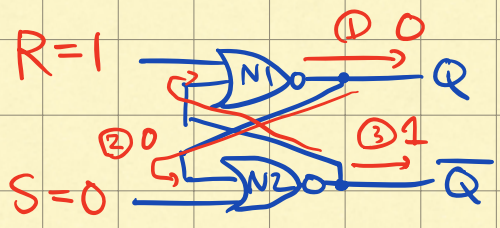
3.2.1 SR Latch



w/ two NOR gates.

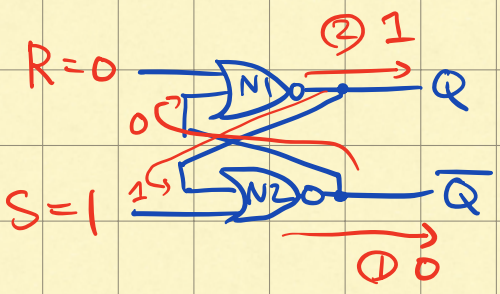
Consider the four possible combinations of R and S:

① R=1, S=0



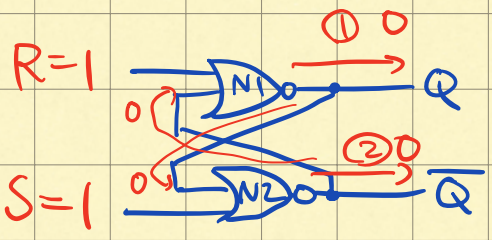
Q=0
$\bar{Q}$=1
Stable

③ R=0, S=1



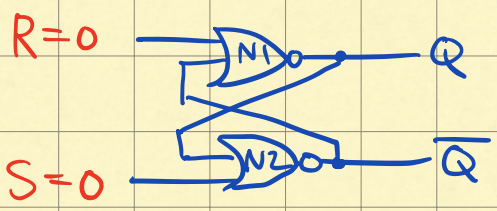
Q=1
$\overline{Q}$=0 stable

③ R=1, S=1



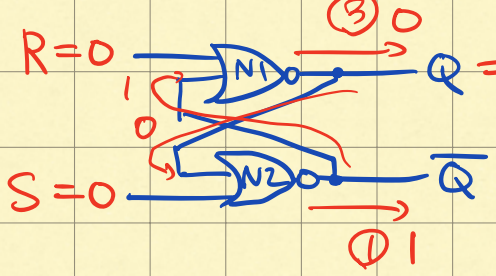
Q=0
$\overline{Q}$=0 stable.

④ R=0, S=0



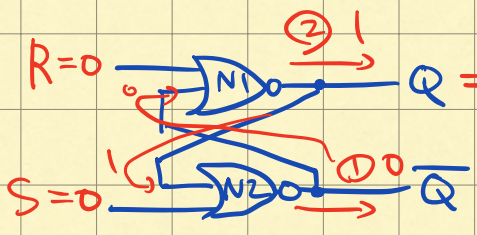
→ ?

④a : Q=0 (Q_{prev})



Q=0
$\overline{Q}$=1 stable.

④b : Q=1 (Q_{prev})



Q=1
$\overline{Q}$=0 stable

When $R=S=0$, it holds Q_{prev} and $\overline{Q_{prev}}$.
$\Rightarrow$ It has memory.

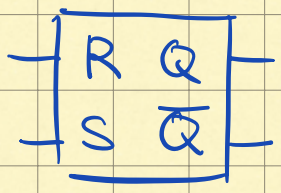
Truth table of the SR Latch.

S	R	Q	$\overline{Q}$
0	0	Q_{prev}	$\overline{Q_{prev}}$
0	1	0	1
1	0	1	0
1	1	0	0

To set a bit, ($S=1$.)
$\rightarrow Q=1$
To reset a bit, ($R=1$)
$\rightarrow Q=0$

($S=R=1$ does not make sense.)

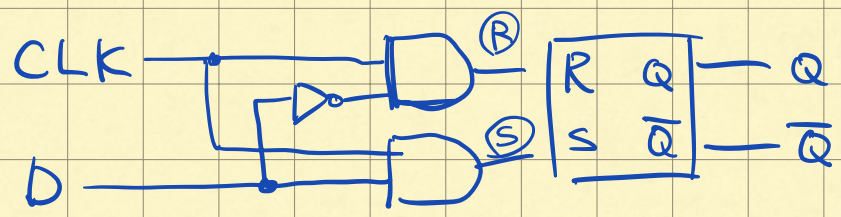
Symbol.



Like the cross-coupled inverters, SR Latch is a bistable element w/ one bit of state stored in Q.
But it can be controlled through the S and R inputs.

3.2.2 D Latch.

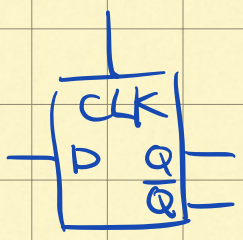
D: what should be
CLK: when should be



If CLK=0, R=S=0 → Q_{prev}.
If CLK=1, D=S, $\bar{D}$=R → set or reset.

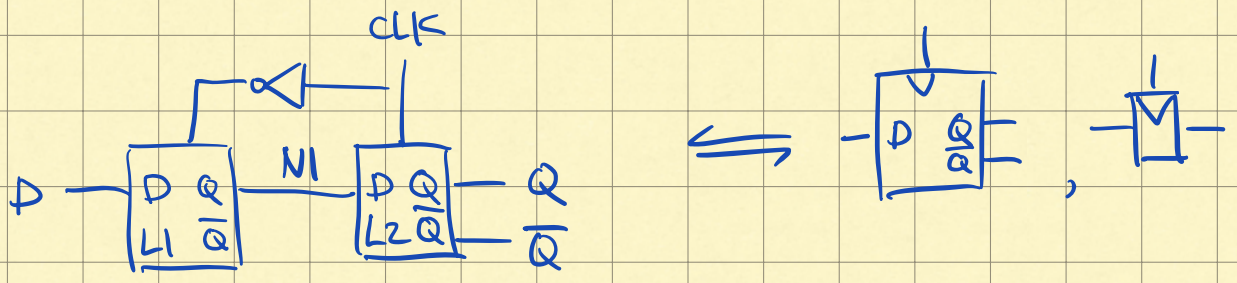
It avoids the case of S=R=1.

Also called, transparant latch or level-sensitive latch.



3.2.3 D Flip-Flop

A D Flip-Flop copies D to Q on the rising edge of the clock and remembers its state at all other times.



CLK=0 : L1 is transparent and L2 is opaque.

$D \rightarrow N1$

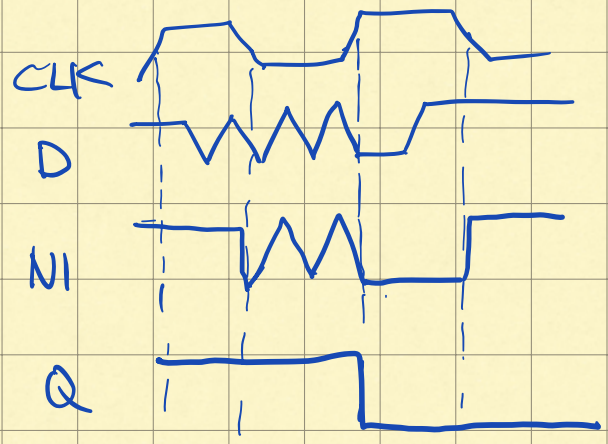
CLK=1 : L1 is opaque and L2 is transparant.

$N1 \rightarrow Q$

$\overline{N1} \rightarrow \overline{Q}$

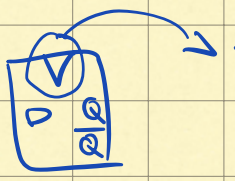
(D, N1)

Hence the value Immediately before the rising edge is copied to Q immediately after the rising edge (0->1)



D flip-flop $\Leftrightarrow$ edge-triggered flip-flop

positive-edge-triggered flip-flop.



indicates edge-triggered.

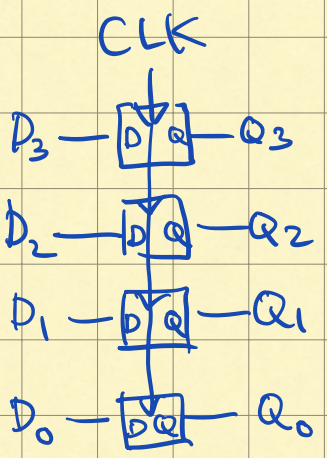
Bistable element w/ a clock input $\Leftrightarrow$ flip-flop.

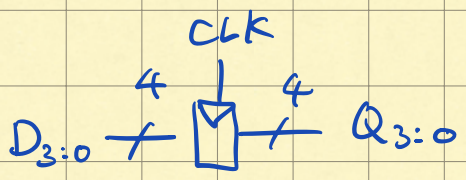
Bistable element w/o a clock input $\Leftrightarrow$ latch.

3.4.2 Register

N-bit register has N flip-flops that share a common CLK input.

key building block of most sequential circuit.

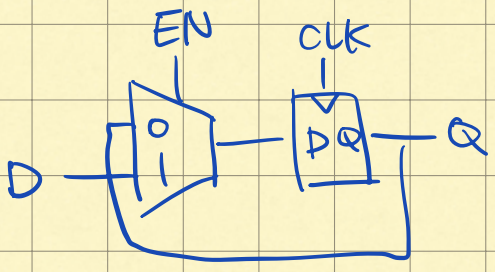




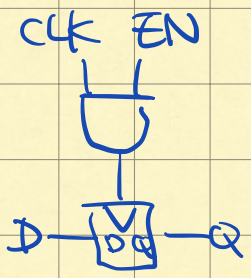
3.2.5 Enabled Flip-Flop.

When we want to pass data at some of the clock edge.

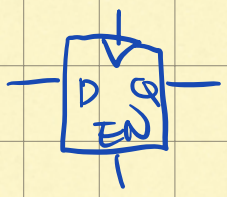
(MUX)



(Clock gated)



Symbol:



EN must not change when CLK=1. (due to the glitch)

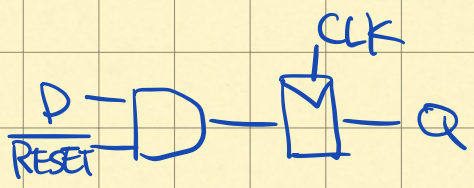
can cause timing errors.

3.2.6 Resettable Flip-Flop.

adds RESET function.

synchronously resettable flip-flop.

(RESETs @ CLK edge).



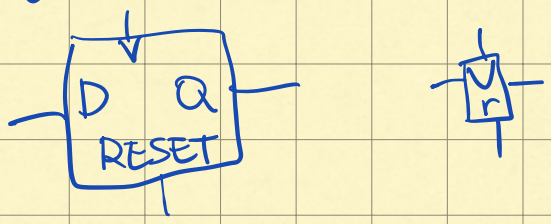
RESETs the Q when $\overline{RESET} = 0$.

→ active low signal ($\overline{RESET}$)

asynchronously resettable flip-flop.

→ RESETs anytime.

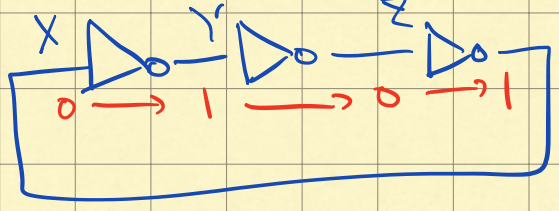
resettable flip-flops w/ active high reset symbol:



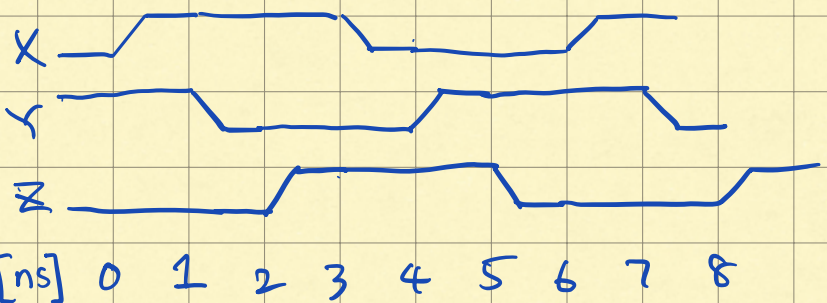
§3.3 SYNCHRONOUS LOGIC DESIGN

3.3.1 Some Problematic Circuits

Ex 3.3



Assume X=0 → 1.

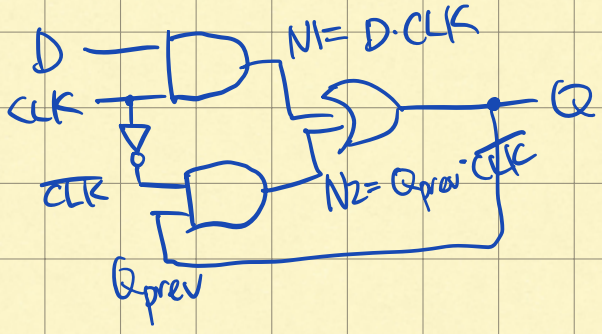


periodically oscillates with T = 6ns.

ring oscillator (difficult to accurately calculate

Example 3.4

D latch?



It seems fine but, it has race condition.

Suppose that CLK=D=1.

→ Q=1.

Now, CLK falls. Q should be 1.
However, suppose delay through the inverter is rather long compared to the AND and OR gate.

Then, before the change of $\overline{CLK}$,
Q becomes 0.
Even if $\overline{CLK}$ =0 → 1, Q_{prev} is 0.
So Q stuck at 0.

Slight different delays make circuit operates differently. These malfunctions are extremely difficult to track.

3.3.2 Synchronous Sequential Circuits

Sequential circuits with cyclic paths can have undesirable races or unstable behavior.

To avoid this, designer break the cyclic paths by inserting registers.

⇒ The collection of combinational logic & registers.

I|O

A synchronous sequential circuit has a clock input.
- rising edges are seq. of times (transition occurs)

FUNC

next state & possible output (curr & inputs)

TIMING

upper bound: t_{pcq} (propagation CLK to Q)

lower bound: t_{ccq} (contamination CLK to Q)

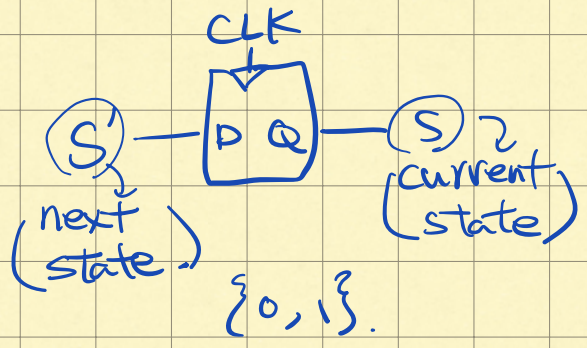
t_{setup}

t_{hold}

The rules of Synchronous sequential circuit composition

- Every circuit element is either a register or a combinational circuit.
- At least one circuit element is a register
- All registers receive the same clock signal.
- Every cyclic path contains at least one register

For example, flip-flop.
(simplest synchronous sequential circuit)



Common types of S.S. circuit:
finite state machines & pipelines.

3.3.3 Synchronous and Asynchronous Circuits

Asynchronous design in theory is more general than synchronous design.

However, synchronous circuits are more easier to design.

Asynchronous circuits are necessary when communicating with different systems (diff. CLK) and real world.

3.4 FINITE STATE MACHINES

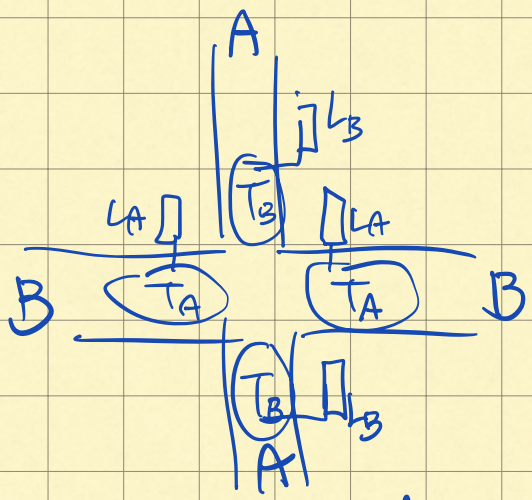
A circuit w/ k registers can be in one of a finite number (2^k) of unique states.

An FSM has M inputs, N outputs, and k-bit of state. (CLK & RESET too)

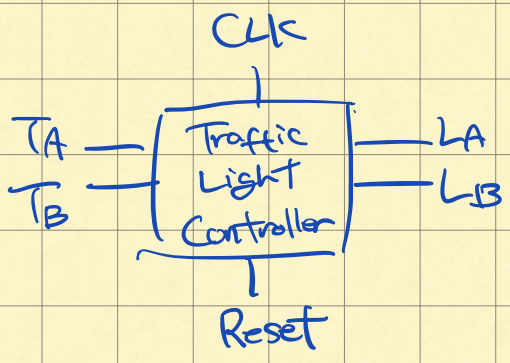
Consists of two blocks of CL, next state logic and output logic, and a register that stores the state.

3.4.1 FSM Design Example

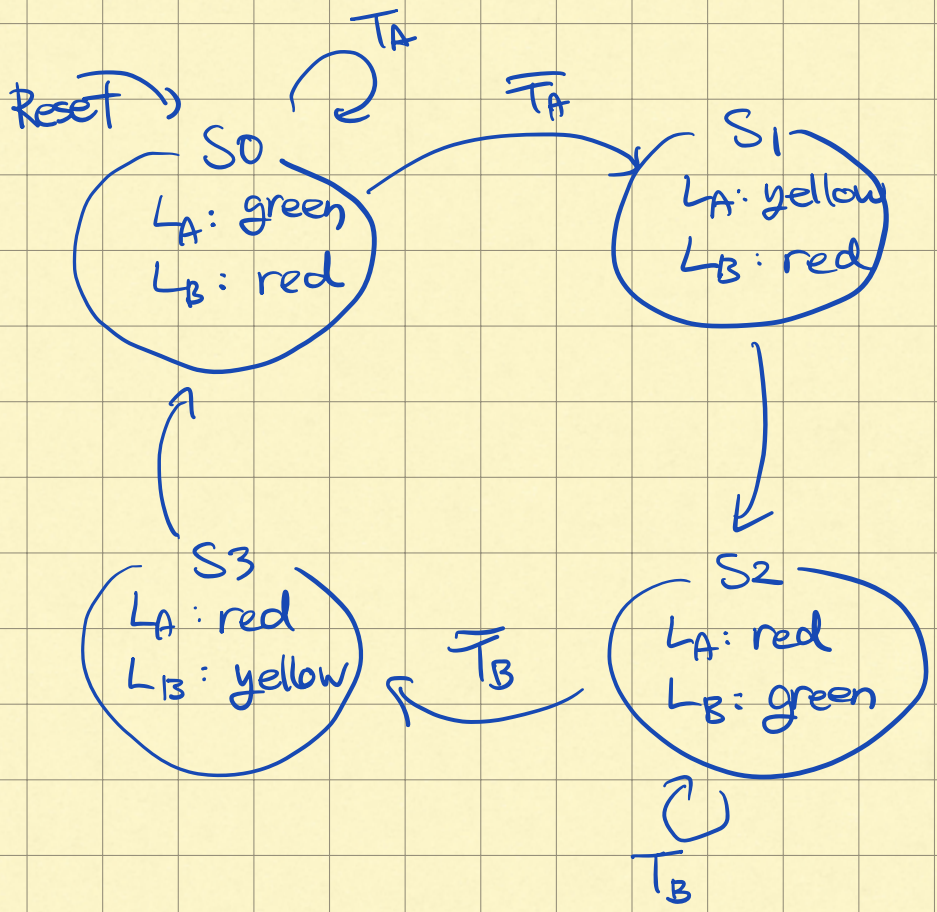
traffic sensors T_A, T_B.
traffic lights L_A, L_B.



(5-sec period clock.)



State Transition Diagram.



Every 5-sec, controller examines the traffic pattern.

○ : states

⤵ : transition @ rising edge.

State Transition Table

Current State S	Inputs T_A	Inputs T_B	Next State 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

State should encoded into $S_{1:0}$ (2^2 states)

Output should encoded into $L_{1:0}$ (2^2-1 states)

State	$S_{1:0}$
S0	00
S1	01
S2	10
S3	11

Output	$L_{1:0}$
green	00
yellow	01
red	10

Encoded State Transition Table

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

From this table, we can derive next state function in terms of current state & inputs.

$$S_1' = \bar{S}_1 S_0 + S_1 \bar{S}_0 \bar{T}_B + S_1 \bar{S}_0 T_B$$

$$S_0' = \bar{S}_1 \bar{S}_0 \bar{T}_A + S_1 \bar{S}_0 \bar{T}_B$$

By inspection,

$$S_1' = \bar{S}_1 S_0 + S_1 \bar{S}_0 = S_0 \oplus S_1$$

$$S_0' = \bar{S}_1 \bar{S}_0 \bar{T}_A + S_1 \bar{S}_0 \bar{T}_B$$

Output table

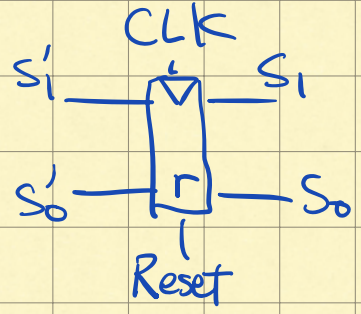
Current State		Outputs			
S_1	S_0	L_{A1}	L_{A0}	L_{B1}	L_{B0}
0	0	0	0	1	0
0	1	0	1	1	0
1	0	1	0	0	0
1	1	1	0	0	1

$L_{A1} = S_1$ $\quad L_{B1} = \overline{S_1}$

$L_{A0} = \overline{S_1} S_0$ $\quad L_{B0} = S_1 S_0$

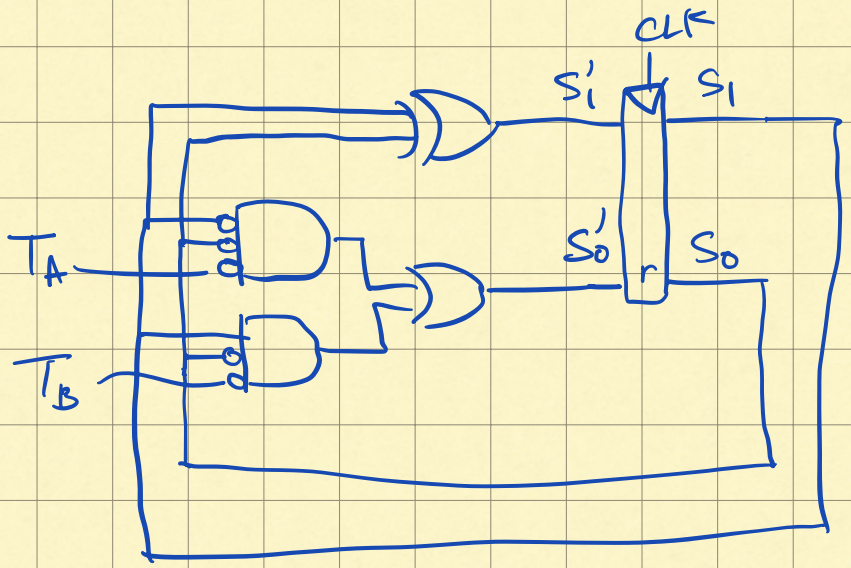
Moore FSM.

① 2-bit state register.

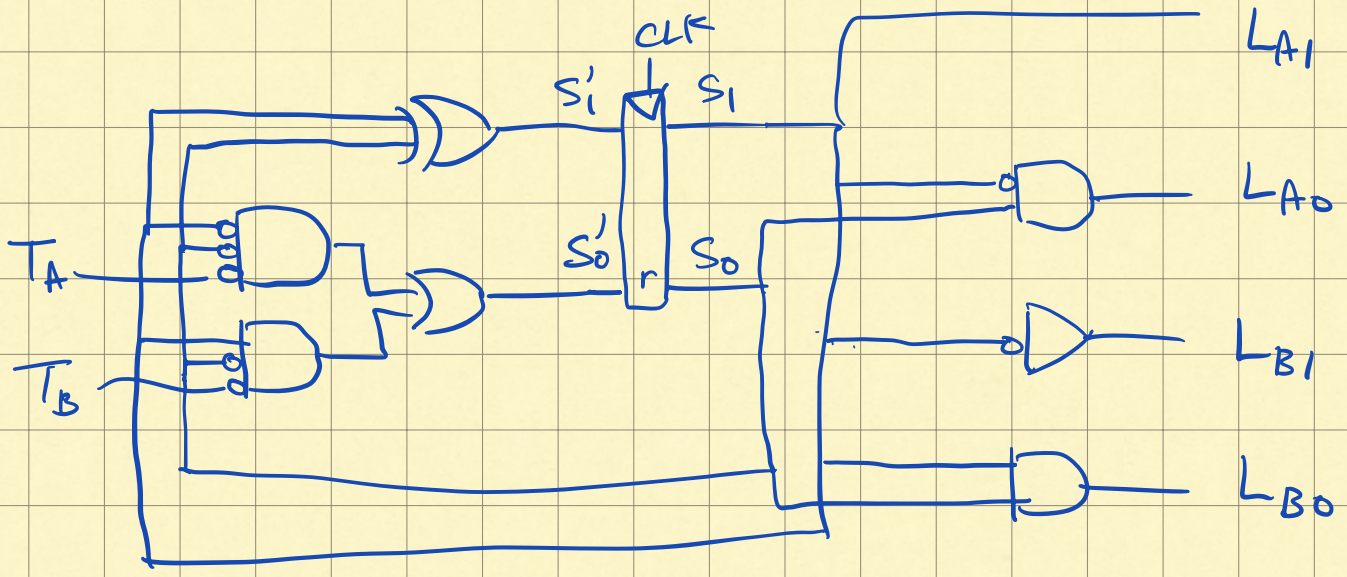


② Next state Logic

$$S_1' = S_0 \oplus S_1$$
$$S_0' = \overline{S_1} \overline{S_0} T_A + S_1 \overline{S_0} T_B$$



③ Output logic.



$L_{A1} = S_1$ $\quad L_{B1} = \overline{S_1}$

$L_{A0} = \overline{S_1} S_0$ $\quad L_{B0} = S_1 S_0$

asynchronous reset, set $S_{1:0} = 00$

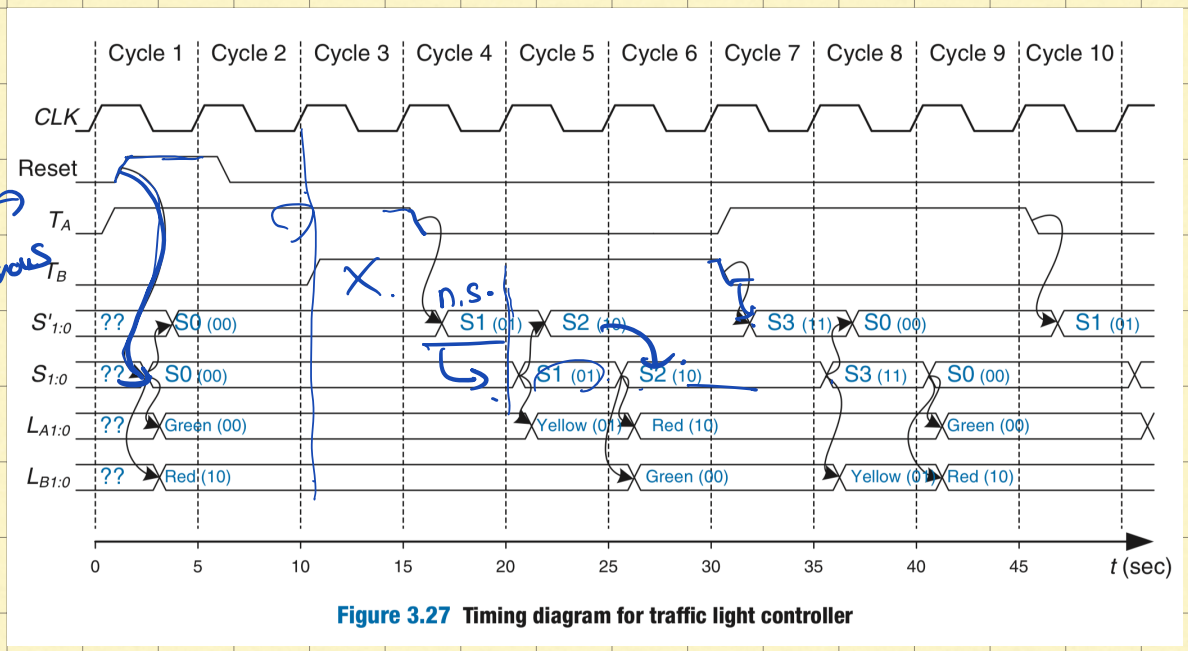


Figure 3.27 Timing diagram for traffic light controller

3.4.2 State Encodings

In the previous example, we choosed arbitrary encodings such as green is 00, yellow is 01, and red is 10.

But different choice can be made. Which encoding produces the most efficient circuit?

Binary encoding VS One-hot encoding.

① Binary encoding.

A system w/ k states needs only $\log_2 k$ bits of state.
(e.g. 4 state → $\log_2 2 = 2$-bits.)

② One-hot encoding.

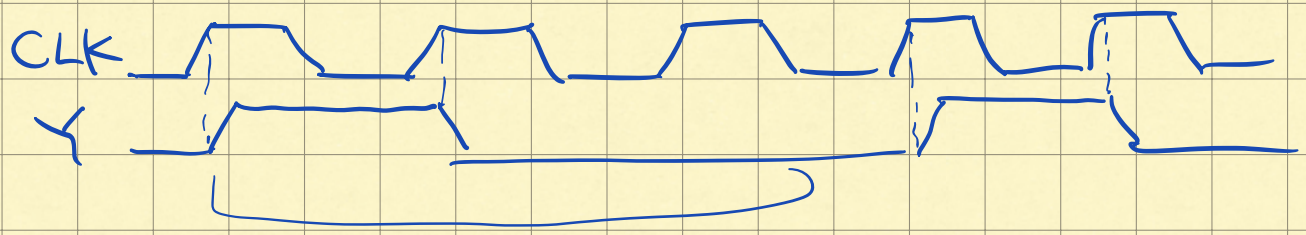
e.g. 3 states → 001, 010, and 100.

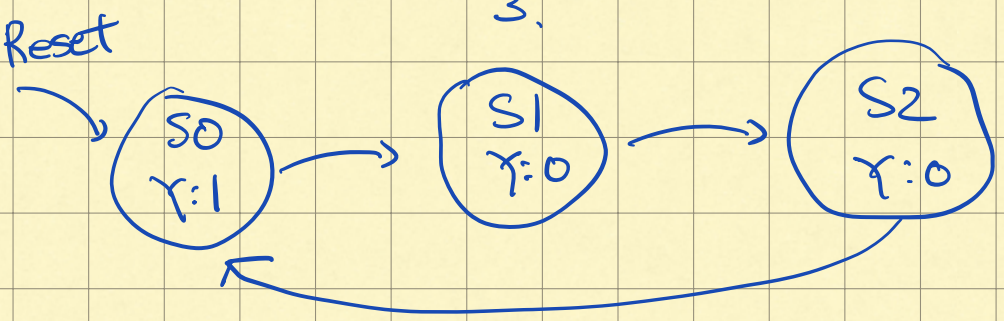
Each bit of states is stored in a flip-flop
→ requires more ff.
However, the next state & output logic is simpler.

Example 3.6 FSM STATE ENCODING

A divide-by-N counter.

divide-by-3 counter:





Sol.

State transition table

Curr. State	Next State
S0	S1
S1	S2
S2	S0

Output table

Curr. State	Output
S0	1
S1	0
S2	0

State	One-Hot Encoding S_2	S_1	S_0	Binary Encoding S_1	S_0
S0	0	0	1	0	0
S1	0	1	0	0	1
S2	1	0	0	1	0

① Binary Encoding

State transition table.

Curr. State S_1	S_0	Next State S_1'	S_0'
0	0	0	1
0	1	1	0
1	0	0	0

Output table

State S_1	S_0	Output Y
0	0	1
0	1	0
1	0	0

$$\Rightarrow \begin{aligned} S_1' &= \overline{S_1} S_0 \\ S_0' &= \overline{S_1}\,\overline{S_0} \end{aligned} \qquad Y = \overline{S_1}\,\overline{S_0}$$

② One-hot Encoding

State transition table.

Curr. State S_2	S_1	S_0	Next State S_2'	S_1'	S_0'
0	0	1	0	1	0
0	1	0	1	0	0
1	0	0	0	0	1

Output table

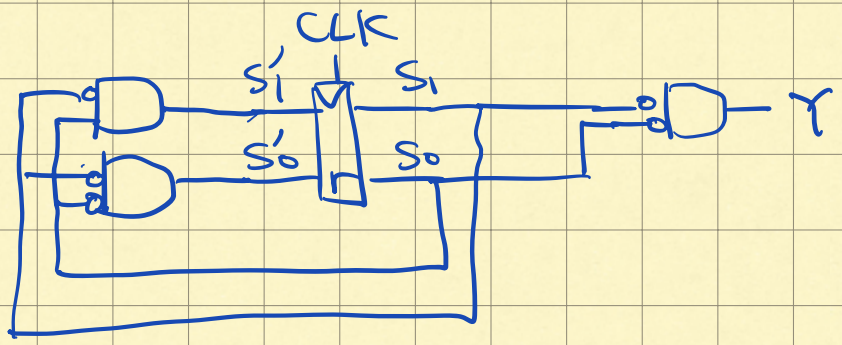
State S_2	S_1	S_0	Output Y
0	0	1	1
0	1	0	0
1	0	0	0

$$\Rightarrow \begin{aligned} S_2' &= \overline{S_2} S_1 \overline{S_0} = S_1 \\ S_1' &= \overline{S_2}\,\overline{S_1} S_0 = S_0 \\ \overline{S_0'} &= S_2 \overline{S_1}\,\overline{S_0} = S_2 \end{aligned} \qquad \begin{aligned} Y &= \overline{S_2}\,\overline{S_1} S_0 \\ &= S_0 \end{aligned}$$

Schematics

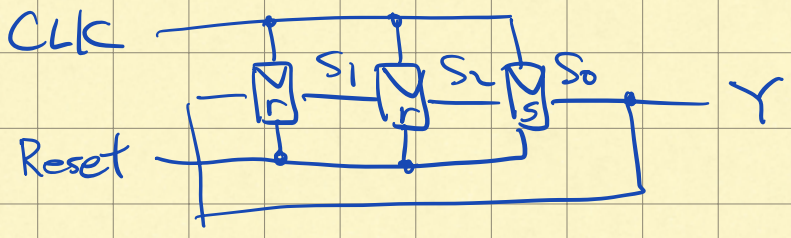
① Binary Encoding.

$S_1' = \bar{S_1} S_0$
$S_0' = \bar{S_1} \bar{S_0}$
$Y = \bar{S_1} \bar{S_0}$



③ One-hot Encoding

$S_2' = S_1$
$S_1' = S_0$
$S_0' = S_2$
$Y = S_0$

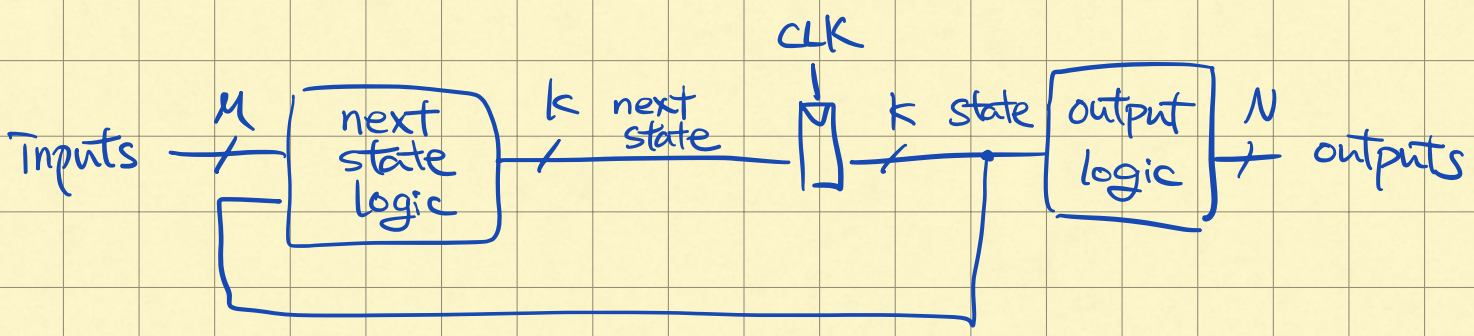


NOTE:  represents a flip-flop or a register. (consists of multiple FF.)

3.4.3 Moore and Mealy Machines

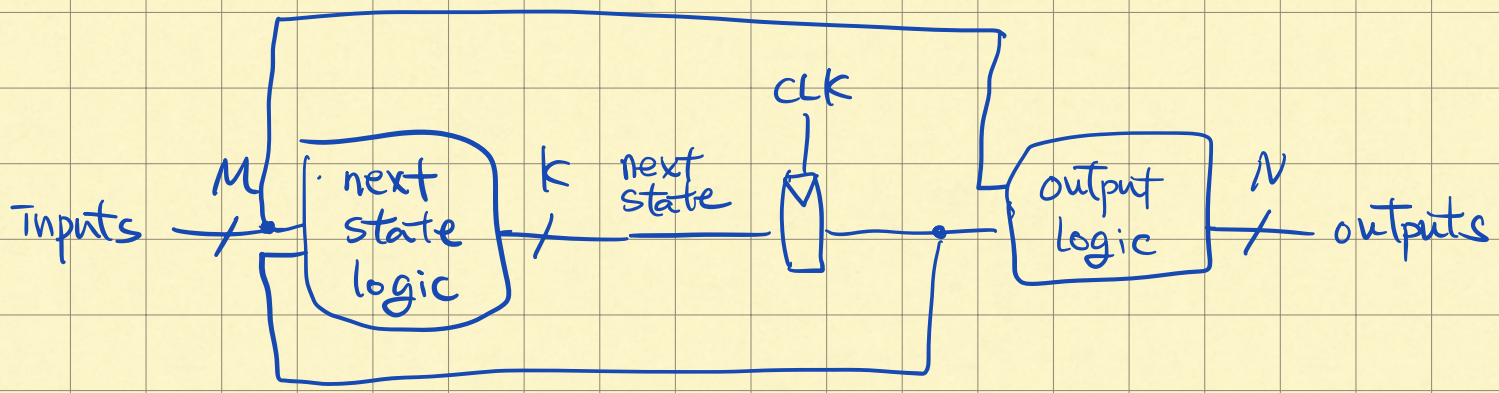
Moore machine's output depends on current state.

→ outputs are in circles (in diagram)

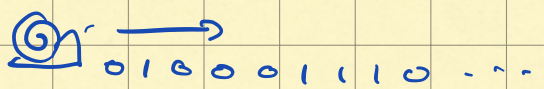


Mealy machine's output depends on input & current state.

→ outputs are on arcs. (in diagram)

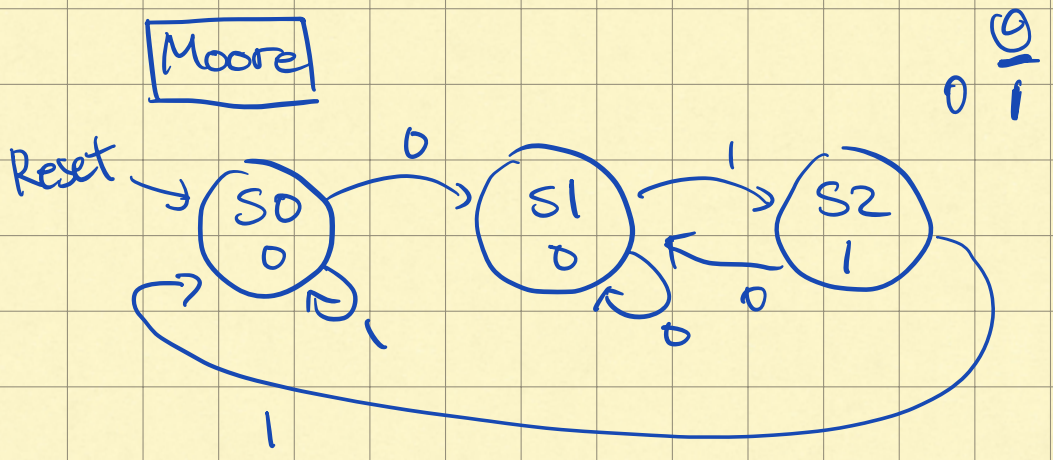


Examples



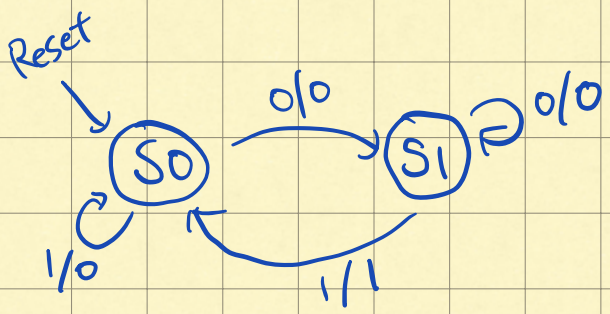
The output is TRUE when the last seq of bits are 01.

Moore



Mealy

A/Y (Input/output)



(After) 1.

(After) 0.

State transition table for Moore

Current State S	Input A	Next State S'
S0	0	S1
S0	1	S0
S1	0	S1
S1	1	S2

S_2	0	S_1
S_2	1	S_0

Output Table for Moore

Current State S	Output
S_0	0
S_1	0
S_2	1

After encoding $S0 = 00$, $S1 = 01$, $S2 = 10$

Curr. State S		Input	Next State S'	
S_1	S_0	A	S_1'	S_0'
0	0	0	0	1
0	0	1	0	0
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0

Curr. State S		Output
S_1	S_0	Y
0	0	0
0	1	0
1	0	1

$S_1' = S_0 A$ $\quad$ $S_0' = \overline{A}$ $\quad$ $Y = S_1$ $\quad$ (state 11_2 doesn't exist.)

Mealy combined table. (∵ output depends on curr. state & inputs).

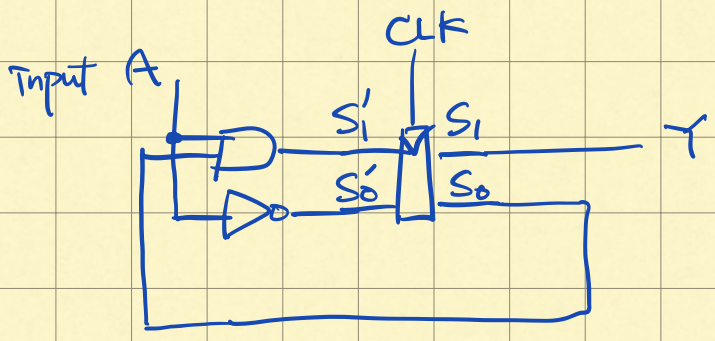
Curr. State S_0	Input A	Next State S_0'	Output Y
0	0	1	0
0	1	0	0
1	0	1	0
1	1	0	1

$\Rightarrow \quad S_0' = \overline{A}, \qquad Y = S_0 A$

Schematics

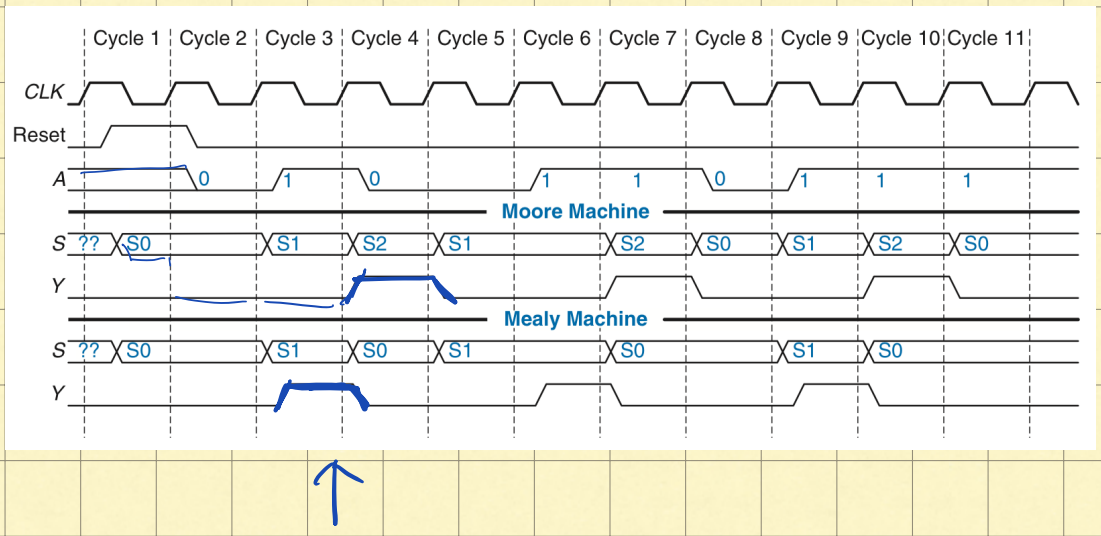
Moore: $S_1' = S_0 A \qquad Y = S_1$

$S_0' = \overline{A}$

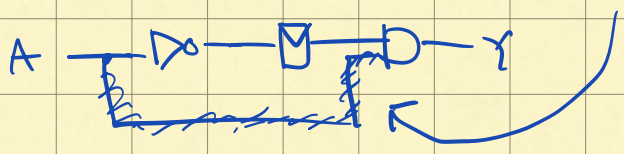


Mealy: $S_0' = \overline{A} \qquad Y = S_0 \cdot A$

Timing Diagram



Mealy's output rises a cycle sooner because it responds to the input rather than waiting for the state change.



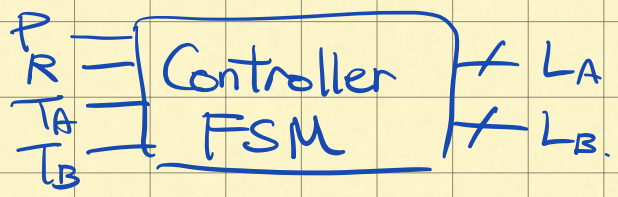
If the output were delayed by the FF, it'll match the Moore output.

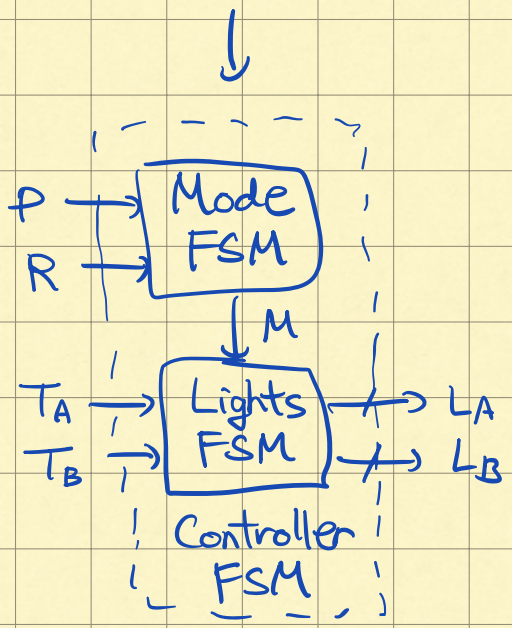
3.4.4 Factoring State Machines.

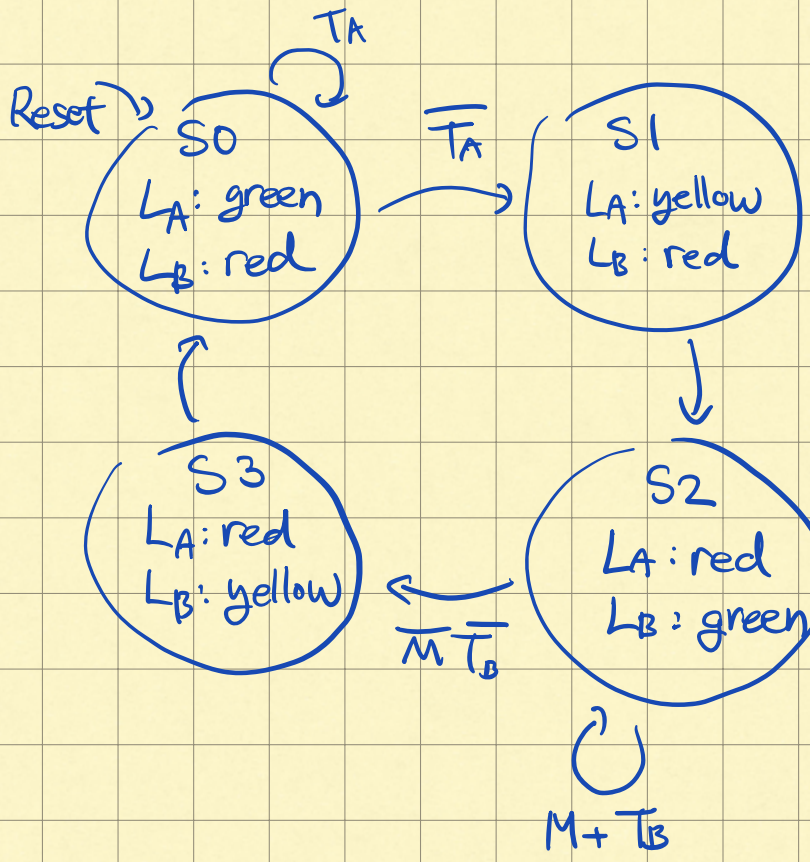
complex state machine ——factoring——> multiple simple state machines.

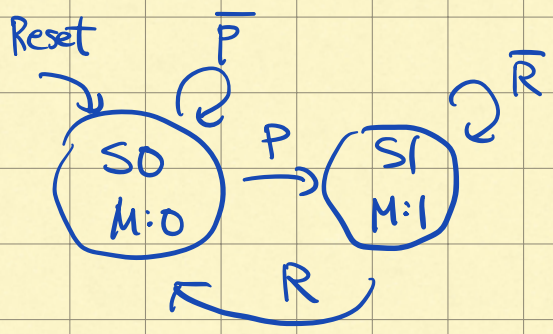
Example 3.8 UNFACTORED & FACTORED STATE MACHINES.

Add Parade Mode. new input: P, R.


P
R
T_A
T_B
Controller FSM
L_A
L_B


P
R
Mode FSM
M
T_A
T_B
Lights FSM
L_A
L_B
Controller FSM


Reset
S0
L_A: green
L_B: red
T_A
$\overline{T_A}$
S1
L_A: yellow
L_B: red
S2
L_A: red
L_B: green
$M + T_B$
S3
L_A: red
L_B: yellow
$\overline{M}\,\overline{T_B}$


Reset
S0
M:0
$\overline{P}$
P
S1
M:1
$\overline{R}$
R

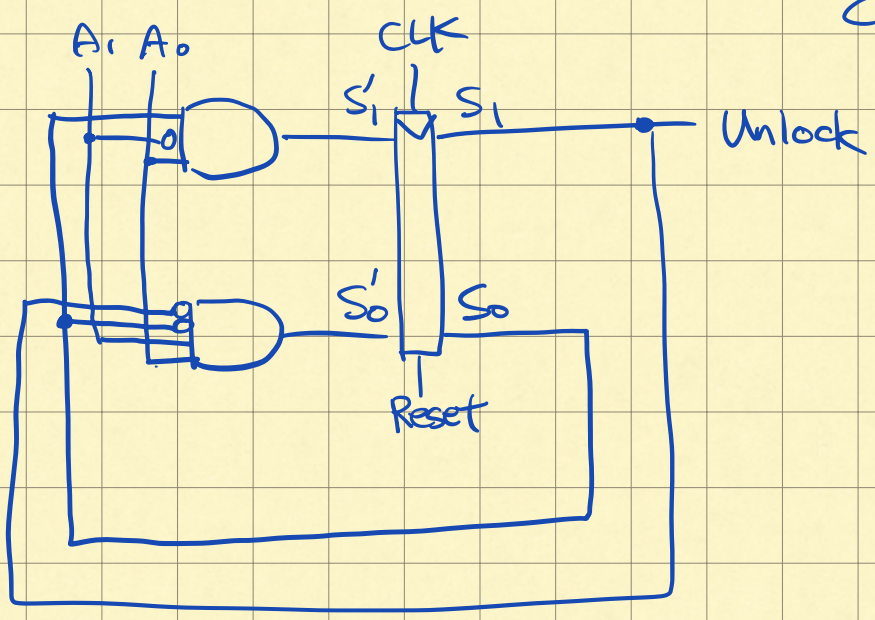
3.4.5 Deriving an FSM from a Schematic

Schematic → State Transition Diagram.

When? { Incompletely documented project
or
reverse engineering.

1. Examine circuit, stating Inputs, outputs, and state bits.
2. Write next state and output equations.
3. Create next state and output tables.
4. Reduce the next state table to eliminate unreachable states.
5. Assign each valid state bit combination a name.
6. Rewrite next state and output tables w/ state names.
7. Draw state transition diagram.
8. State in words what the FSM does.

Example 3.9 DERIVING AN FSM FROM ITS CIRCUIT



① Inputs: $A_{1:0}$
Outputs: Unlock
State bits: (Labeled)

(Also, it is a Moore machine.)

② $S_1' = S_0 \overline{A_1} A_0$ Unlock = S_1
$S_0' = \overline{S_1}\,\overline{S_0} A_1 A_0$

③

Current State		Input		Next State	
S_1	S_0	A_1	A_0	S_1'	S_0'
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1

⋮

Reduced:

S_1	S_0	A_1	A_0	S_1'	S_0'
0	0	0	0	0	0
0	0	0	1	0	0
0	0	1	0	0	0
0	0	1	1	0	1
0	1	0	0	0	0
0	1	0	1	1	0
0	1	1	0	0	0
0	1	1	1	0	0
1	0	x	x	0	0

↓

State Transition Table.

3.4.6 FSM Review

Procedure to design an FSM.

① Identify the inputs and outputs

② Sketch a state transition diagram.

1. For a Moore machine:
   - Write a state transition table.
   - Write an output table.

2. For a Mealy machine: (A/Y)
- Write a combined state transition and output table.

③ Select state encodings – your selection affects the hardware design

④ Write Boolean equations for the next state and output logic.

⑤ Sketch the circuit schematic.

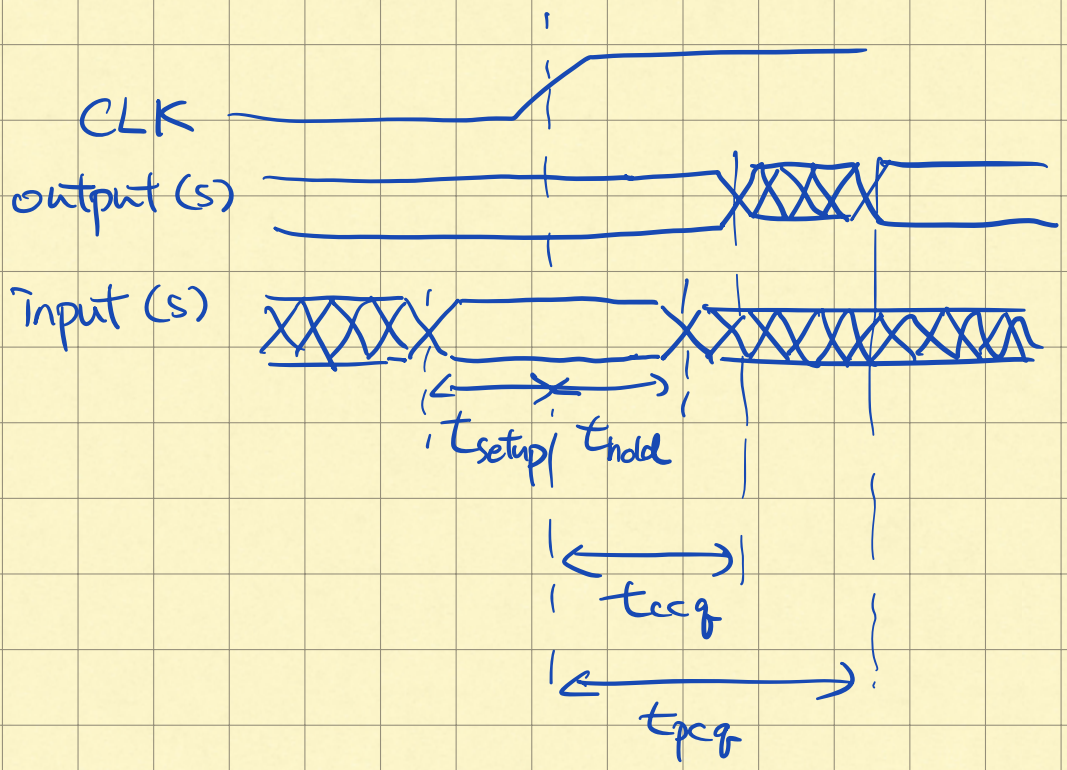
§3.5 TIMING OF SEQUENTIAL LOGIC

3.5.1 The Dynamic Discipline

Timing Specification

When the clock rises,
the output starts to change.
after clock-to-Q contamination delay t_{ccq}

must be settle to the final value
within the clock-to-Q propagation delay t_{pcq}



$t_{setup} + t_{hold}$ = aperture time

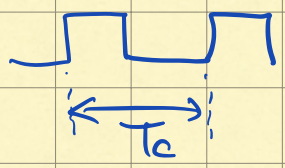
dynamic discipline:

The inputs of a synchronous sequential circuit must be stable during the setup and hold aperture time around the clock edge.

⇒ So that we can treat input as a discrete signal.

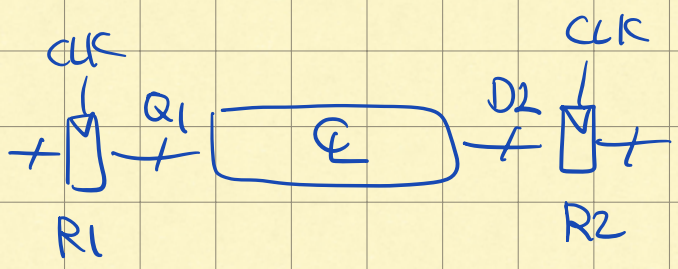
3.5.2 System Timing

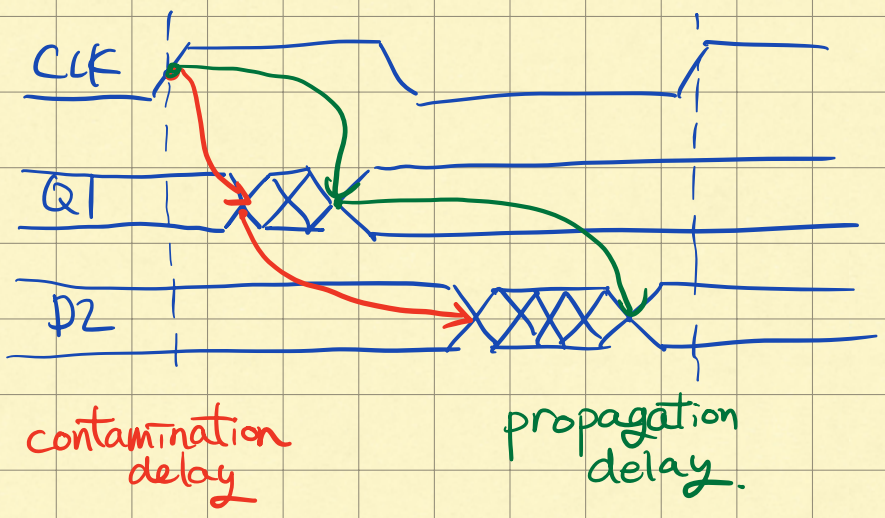
- The clock period or cycle time, T_c, is



- The clock frequency $f_c = 1/T_c$ [Hz]

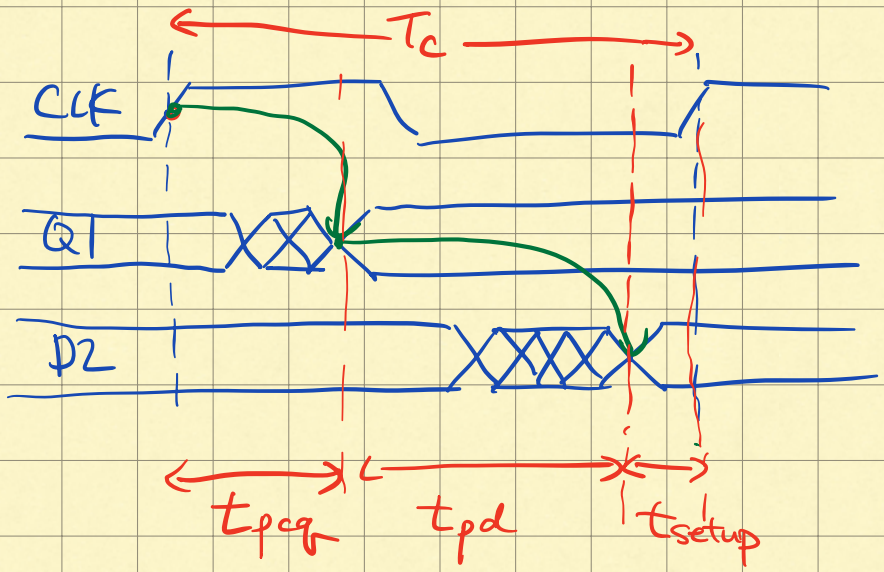
Increasing the clock frequency increases the work that a digital system can accomplish per unit time.





Setup Time Constraint

To satisfy the setup time of R2, D2 must settle before the setup time.



Minimum clock period:

$$T_c \geq t_{pcq} + t_{pd} + t_{setup}$$

t_{pcq} and t_{setup} are specified by the manufacturer.
Thus, t_{pd} is a variable.

Maximum propagation delay through the C
or setup time constraint
max-delay constraint

$$t_{pd} \leq T_c - (t_{pcq} + t_{setup})$$

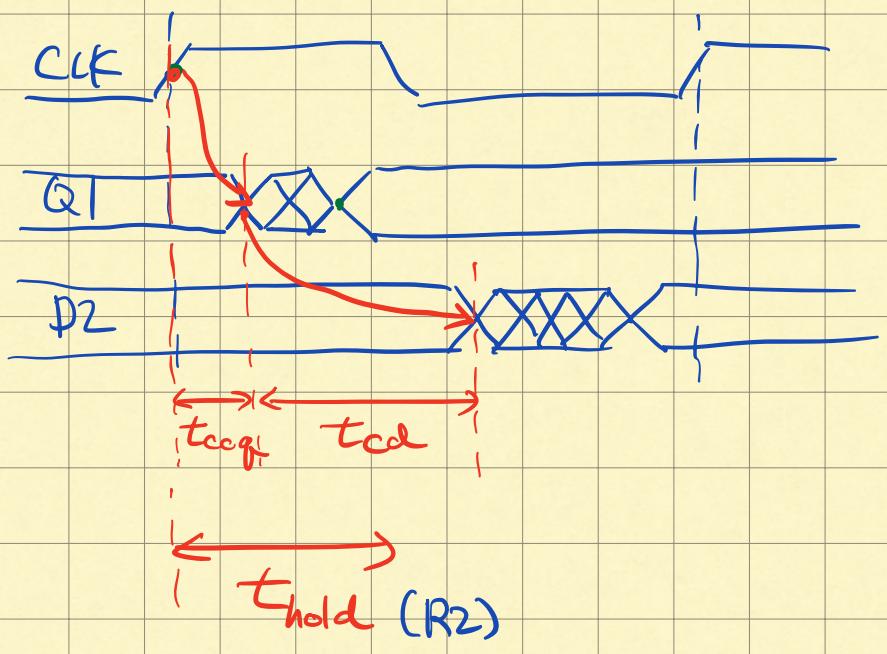
$t_{pcq} + t_{setup}$ is called sequencing overhead.

⇒ T_c would be the available computation time, but the sequencing overhead cuts the available time.

(If the t_{pd} is too great, D2 may not be settled into valid state.
⇒ malfunction.

Hence, increasing T_c or reducing t_{pd} can solve this.)

Hold Time Constraint



D2 must not be changed within the t_{hold}.

$$\Rightarrow \quad t_{hold} \leq t_{ccq} + t_{cd}$$

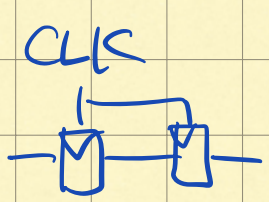
Again, t_{hold} and t_{ccq} are not variable.

Hold time constraint or min-delay constraint.

$$t_{cd} \geq t_{hold} - t_{ccq}$$

(limits the minimum delay through C.)

Back-to-Back FF.



→ There is no C, thus reliable register:

$$t_{ccq} \geq t_{hold}$$

Often, registers are designed with $t_{hold} = 0$.

(Unlike setup time violation, which can be fixed by increasing T_c, hold time violation needs to be fixed by redesigning ICs and manufacturing them.
This must be taken seriously.)

Putting It All Together

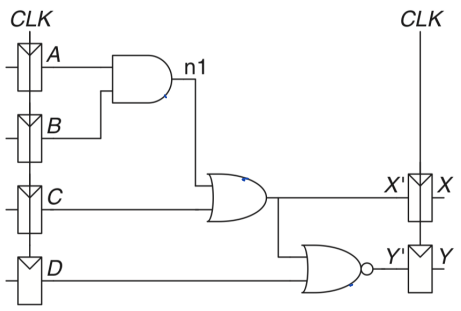
Sequential circuits have setup and hold time that dictate the maximum and minimum delays of the CL between FFs.

Modern FFs are usually designed so that the minimum delay through the CL can be zero — that is, FFs can be placed back-to-back.

The maximum delay constraint limits the number of consecutive gates on the critical path of a high-speed circuit.

Example 3.10 TIMING ANALYSIS

Ben Bitdiddle designed the circuit in Figure 3.42. According to the data sheets for the components he is using, flip-flops have a clock-to-Q contamination delay of 30 ps and a propagation delay of 80 ps. They have a setup time of 50 ps and a hold time of 60 ps. Each logic gate has a propagation delay of 40 ps



and a contamination delay of 25 ps. Help Ben determine the maximum clock frequency and whether any hold time violations could occur. This process is called *timing analysis*.

My Sol.

max-delay.

$t_{pcd} = 80\,ps.$

$t_{setup} = 50\,ps.$

$T_s \leq t_{pcd} + t_{pd} + t_{setup}$

$T_s \leq 130\,ps + t_{pd}.$ → 120ps

min-delay.

$t_{ccd} = 30\,ps, \quad t_{hold} = 60\,ps.$

$t_{hold} \leq t_{ccd} + t_{cd}$

$60\,ps \leq 30\,ps + t_{cd}$

$30\,ps \leq t_{cd}$ → 75ps

$T_{s,min} = 250\,ps$

not violating ~~hold-time~~ violation.

WRONG

Sol.

short path → 1 gates

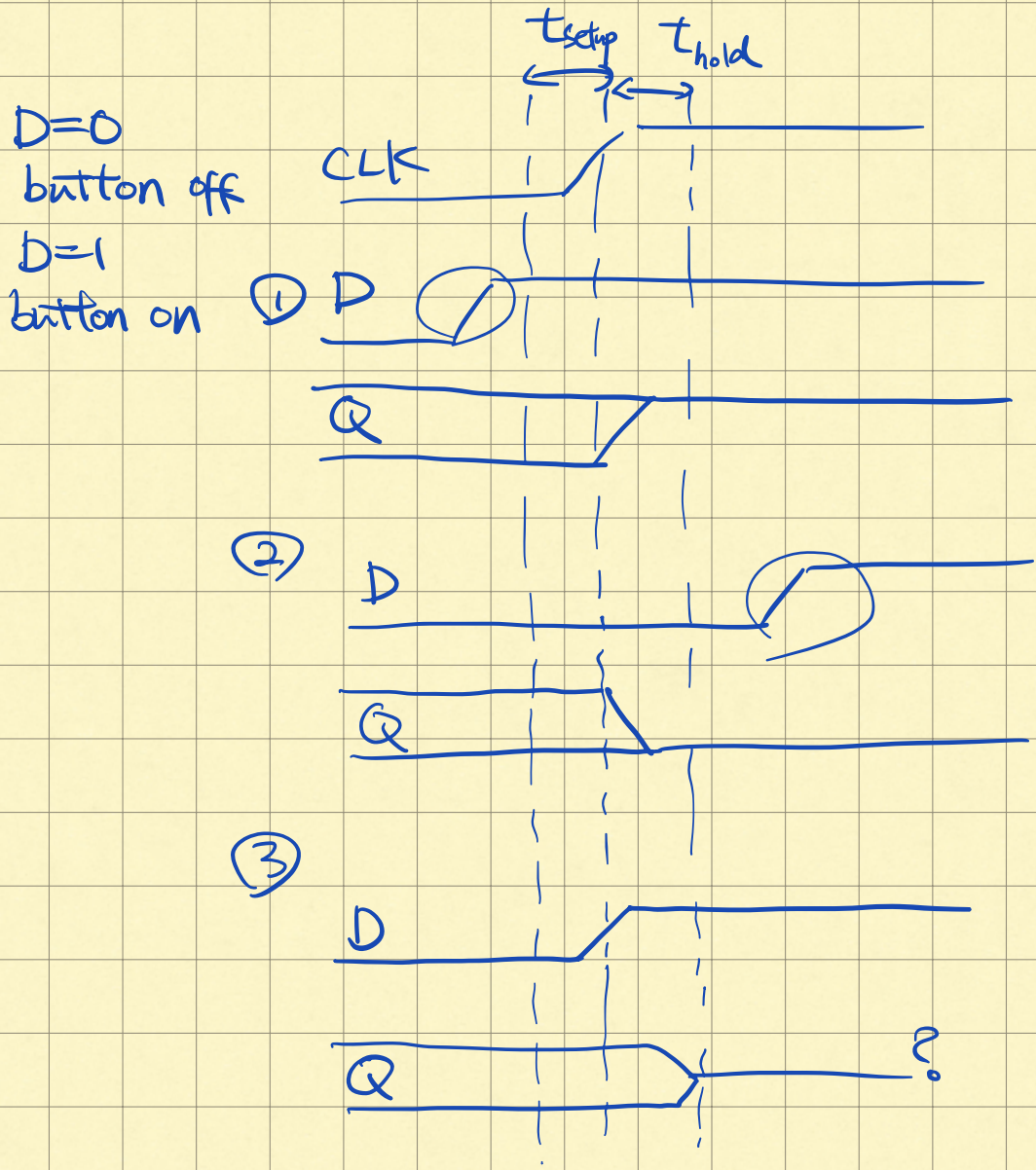
$\Rightarrow t_{hold} \leq t_{ccd} + t_{cd}$

$60\,ps \leq 30\,ps + 25\,ps.$

$\Rightarrow$ Violates

3.5.4 Metastability

e.g. Inputs from external world.



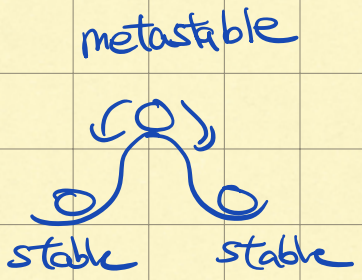
"Violates the dynamic discipline"

Metastable State

In this case, the output momentarily take on a voltage 0 and V_{DD}.

Eventually, the FF will resolve the output to a stable state either 0 or 1.

However, required <u>resolution time</u> is unbounded.



Every bistable device has a metastable state between the two stable states.

Resolution Time

If the input changes outside the aperture,

$$t_{res} = t_{pcq}$$

But the input changes within the aperture, t_{res} can be substantially longer.

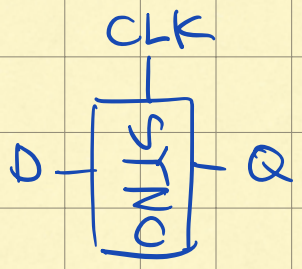
The probability that the t_{res}, exceeds some arbitrary time, t, <u>decreases exponentially</u> with t:

$$P(t_{res} > t) = \frac{T_0}{T_c} e^{-\frac{t}{\tau}}$$

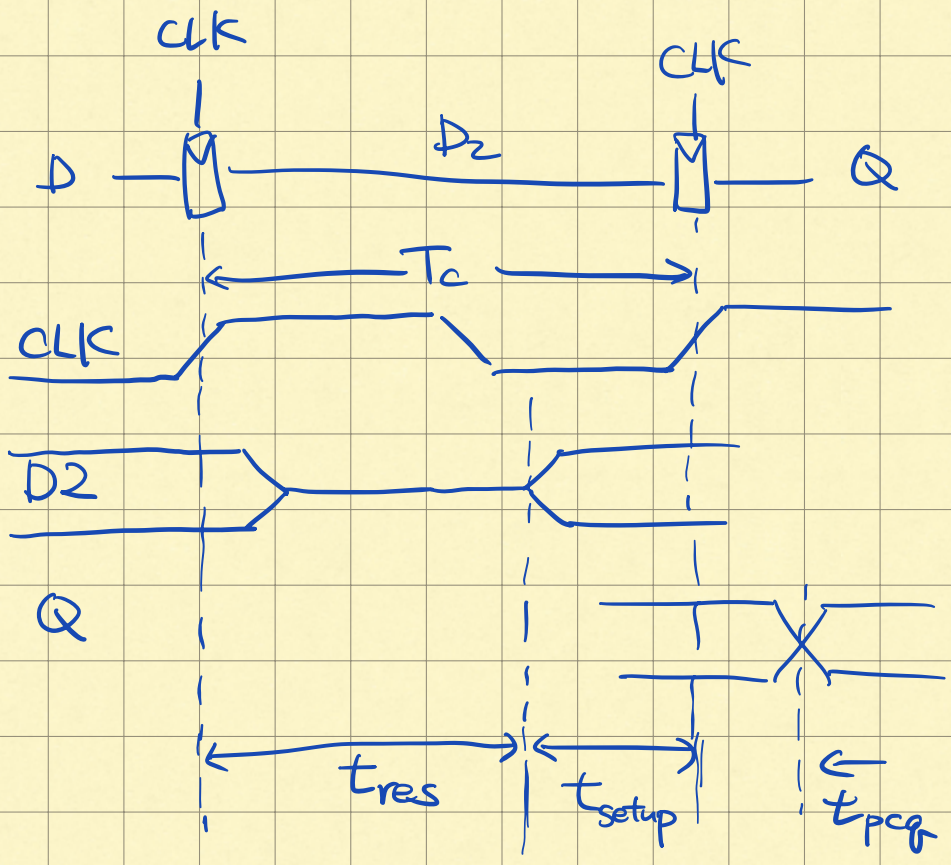
(T_0 and τ are charateristic of FF.)

3.5.5 Synchronizers

Asynchronous Inputs are Inevitable.
All asynchronous Inputs should be passed through synchronizers.



If D is stable within aperture, Q = D
If not, Q is either HIGH or LOW.
(not metastable.)



Synchronizer fails → If Q is metastable.
This may happen if

$$t_{res} > T_c - t_{setup}$$

⇒ The probability of failure for a single input change at a random time

$$P(\text{failure}) = \frac{T_0}{T_c} e^{-\frac{T_c - t_{setup}}{\tau}}$$

If D changes N times per sec,

$$P(\text{failure})/\text{sec} = N \frac{T_0}{T_c} e^{-\frac{T_c - t_{setup}}{\tau}}$$

System reliability is measured in mean time between failures. (MTBF)

→ average amount of time between failures of the system.

$$MTBF = \frac{1}{P(\text{failure})/\text{sec}} = \frac{T_c e^{\frac{T_c - t_{setup}}{\tau}}}{N T_0}$$

MTBF exponentially grow as a synchronizer waits for longer time. ($T_c \uparrow$).

(For most system, a synchronizer waits a one cycle.)

§3.6 PARALLELISM

The speed of a system is characterized by the latency and throughput of information moving through it.

token : a group of inputs that are processed to produce a group of outputs.

"token" conjures up the notion that it moves around the circuits.

latency : the time required for one token to pass through the system from start to end.

throughput: number of tokens that can be produced per unit time.

The throughput can be improved by processing several tokens at the same time. → parallelism

① Spatial parallelism:
multiple copies of the hardware are provided so that multiple tasks can be done at the same time.

② Temporal parallelism:

A task is broken into stages, like an assembly line.

Multiple tasks can be spread across the stages.
→ pipelining

Example 3.16 COOKIE PARALLELISM.

Spatial Parallelism: A helps B. A has their own tray and oven,

Temporal Parallelism: B gets another tray. So he can roll cookies while previous one is being baked.

Consider a task w/ latency L.
In a system w/ no parallelism, the throughput is 1/L.

In a spatially parallel system w/ N copies, the throughput is N/L.

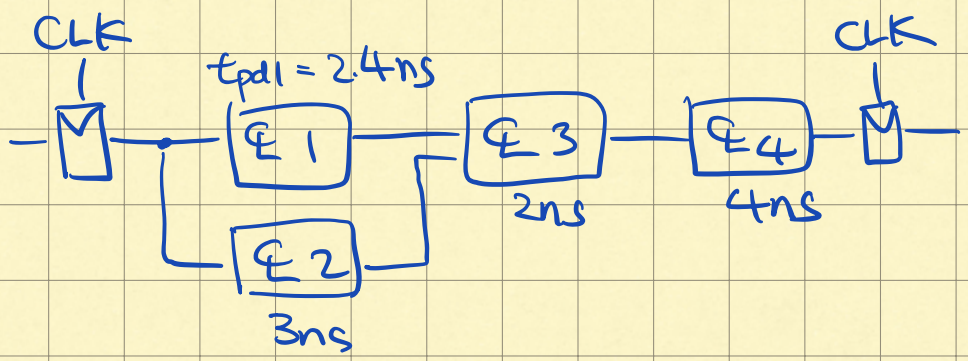
In a temporally parallel system, the task is Ideally broken into N steps, or stages, of equal length.
→ the throughput is N/L.

finding N steps of equal length is often Impratical.
L_1 : longest step's latency.
the pipelined throughput is $1/L_1$

↓

Divide stages w/ registers!

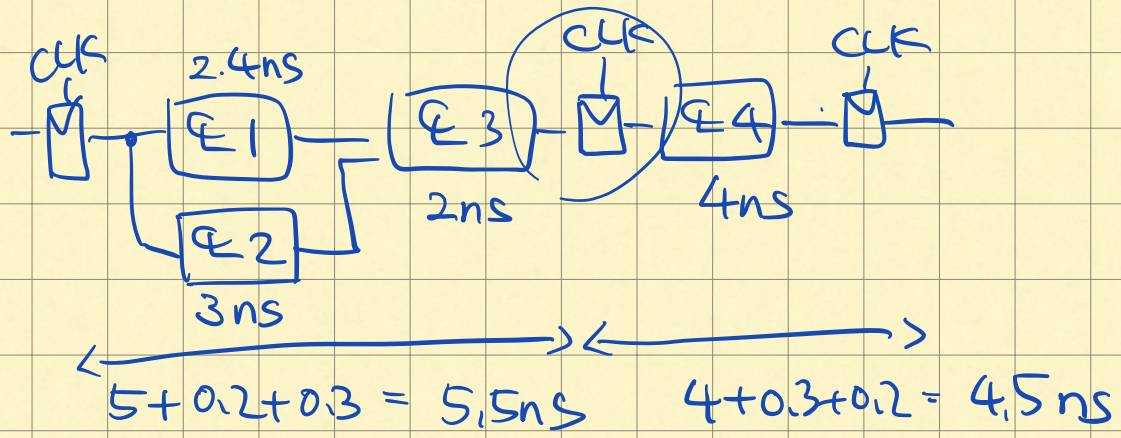
Example of no pipelining system



reg. has $t_{pcq} = 0.3ns$
$t_{setup} = 0.2ns$.

Then, $T_c = 0.3+0.2+3+2+4 = \underline{9.5ns}$ = latency
throughput = $\frac{1}{9.5ns} = 105MHz$.

Partitioned into a two-stage pipeline



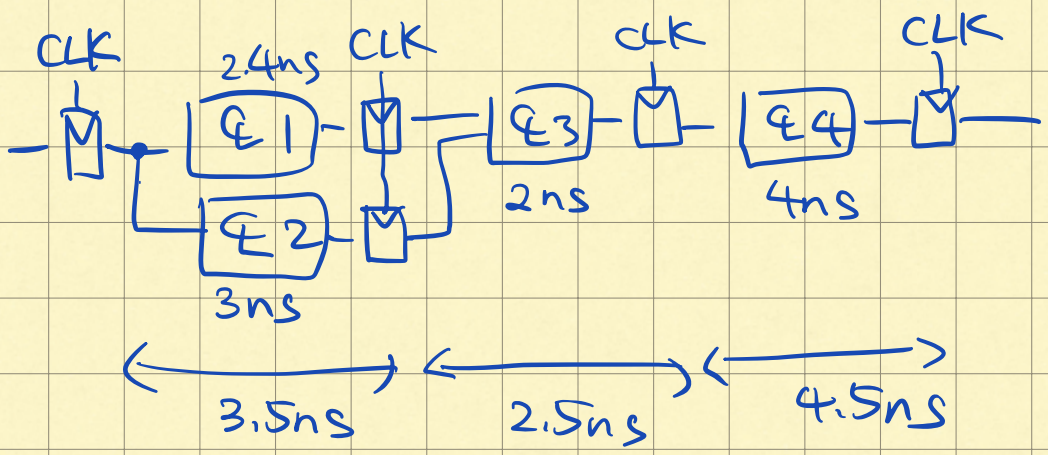
Hence, $T_c = 5.5ns$
latency is two cycles, 11ns

throughput $\frac{1}{5.5ns} = 182MHz$

=> latency ↑, throughput ↑ almost x2.

(Ideally latency does not increase & throughput must be doubled.
If they are exactly divided into halves and there is no sequencing overhead.)

Three-Stage Pipeline



The latency : three cycles = 13.5ns
The throughput : $\frac{1}{4.5ns}$ = 222MHz.

They do not apply to all situations.

dependencies

If a current task is dependent on the result of the prior task, the task cannot start.