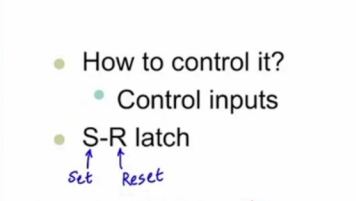
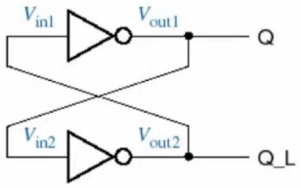
Lecture 8.2 Latches

Back to the Bistable....

- How to control it?
  - Control inputs
- S-R latch
  - Set, Reset





Characteristic Table

S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

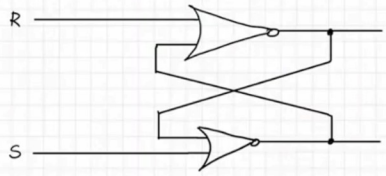
We are Looking at characteristic Tables for sequential elements

← We are Looking at Last Q which is more about the characteristic of the sequential element

From Last class

SR Latch Analysis — Set - Reset Latch

NOR Version — NOR gates in cross coupled Configuration.



Since we are working with NOR gates, let's rewrite NOR Truth Table.

A	B	$\overline{A+B}$
0	0	1
0	1	0
1	0	0
1	1	0

Important property from NOR TT,

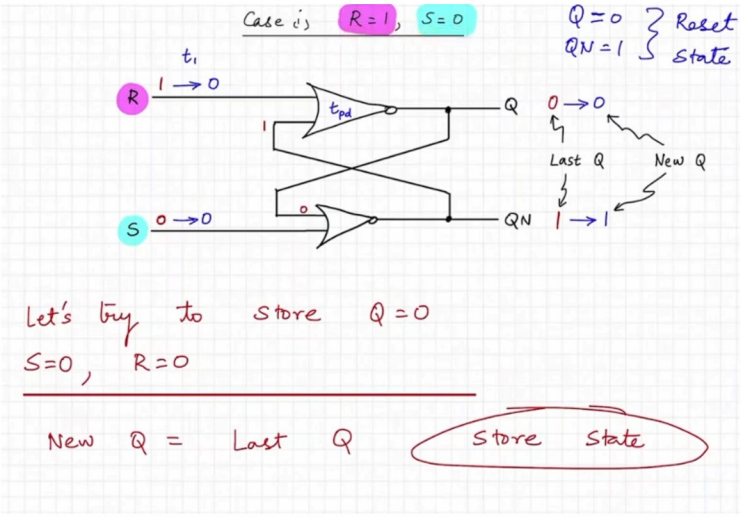
If any input is high then, output is low, irrespective of other input.

From the NOR TT, we see that if one of the inputs is '1', we know for sure that the output is '0'

→ we don't have to wait for the other input to come in.

Because of propagation delay in the Latch, if one input is '0', we have to wait for the other input to be '0' for the output to be a '1'

2



$Q = 0$ at $t_0 + t_{pd}$

$QN = 1$ at $t_0 + t_{pd} + t_{pd}$

NOR delay

We made a claim that if we make the inputs both zero, we can store the Q STATE.

$QN = 1$ from the previous reset state

→ this will keep $Q = 0$ in the next state

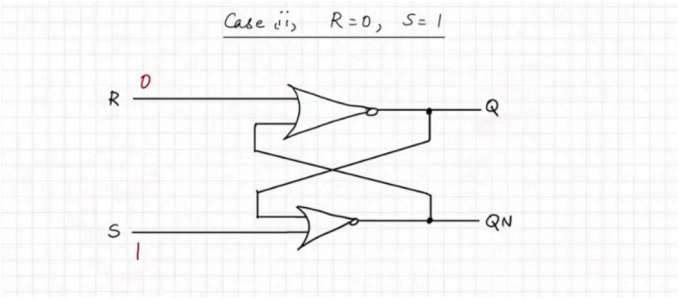
The New Q is like the store state and it's stable

Terminology

Now	Earlier
Q	Last Q
Q_{new}	Q_{prev}
Q^+	Q

Right now, we use

Case ii



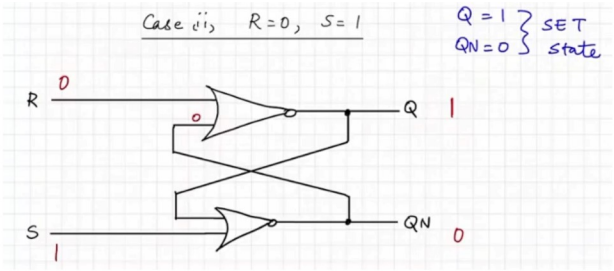
If we know S=1, we already know what QN will be.

$Q_N = 0$

Since $Q_N = 0$ and $R = 0$

→ $Q = 1$ Is this a stable state? Yes

Since we set the output to this, Let's call it the SET STATE



We set the output to '1', now it's time to store it.

We store by making both R and S equal to '0'

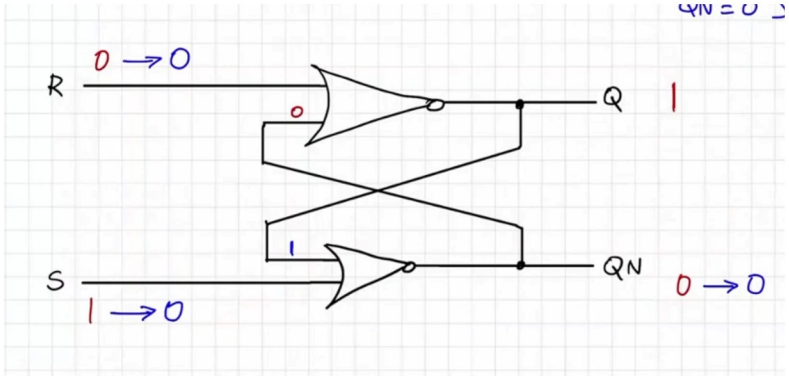
We we say 'Store it' that really means we want

$Q = \text{Last } Q$

$Q^+ = Q$

$Q_{new} = Q_{prev}$

So set R=0, S=0

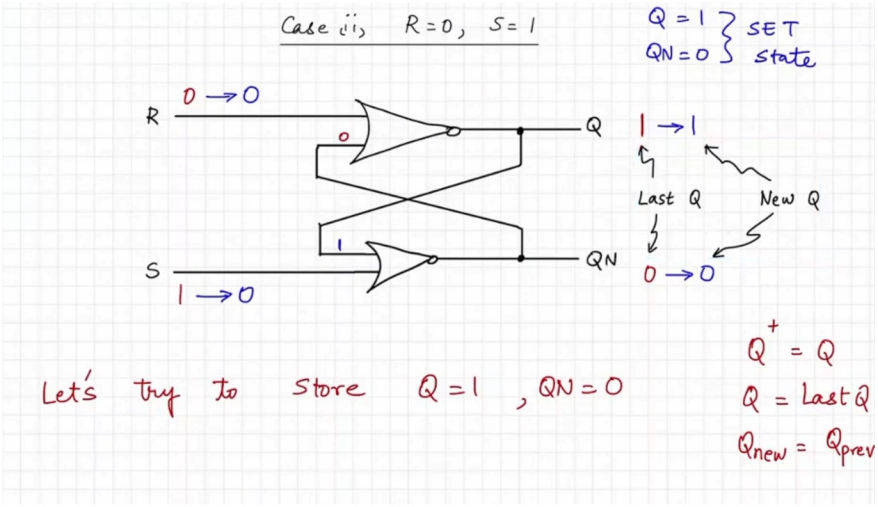


The previous Q (Q=1) Maintains QN=0

So now if QN=0 and R=0

→ Q = 1 still

4 So Our New Q is



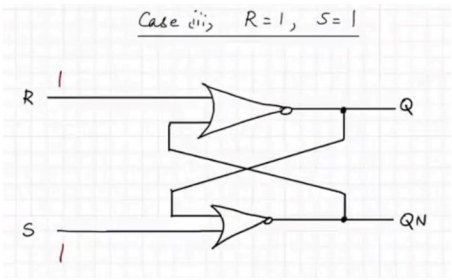
By making S=0
R=0

We have
Q = Last Q
QN = Last QN
} STORE STATE

Reset then Store
Set then Store

Confusion State / head scratcher State

Case iii



In this case Both Q and QN must be '0'

Is this stable? Yes

But the obvious problem now is that QN is not a complemented form of Q

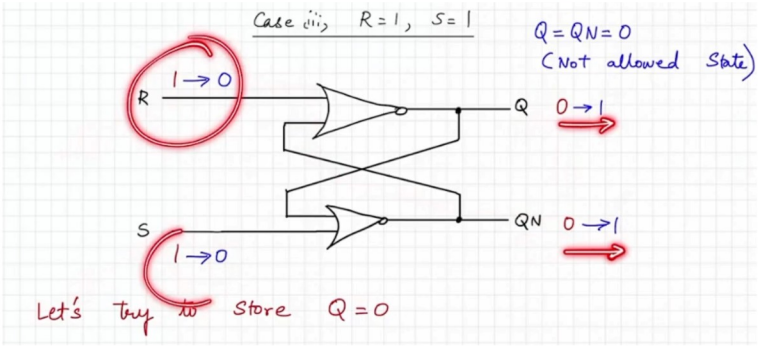
Q = QN = 0

(NOT allowed state)

Let's try to store Q=0

How do we store this?

Make S and R go to '0' at the same time



both Q and QN change to '1' at the same time.

→ Propagation delay of NOR gates is the same.

If $R=0$, $S=0$ at time t_1, when does $Q=1, QN=1$?

$t_1 + t_{pd}$ for both

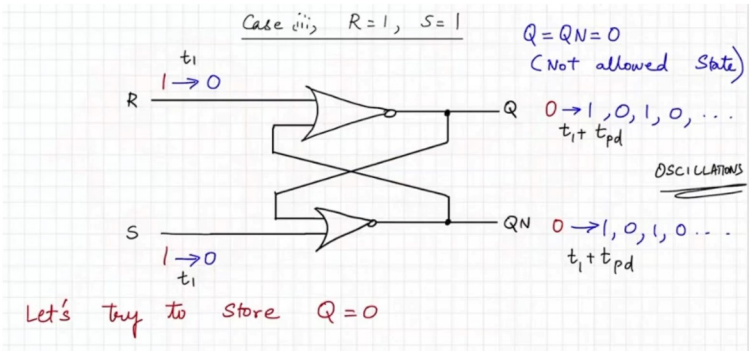
But Then at $t_1 + t_{pd} + t_{pd}$, $Q=0$, $QN=0$

$t_1 + t_{pd} + t_{pd} + t_{pd}$, $Q=1$, $QN=1$

Q, QN will Oscillate

as long as $S=0$, $R=0$ and the propagations delay are exactly the same

$$\text{Frequency} = \frac{1}{t_{pd}}$$



What if propagation delays are not the same?

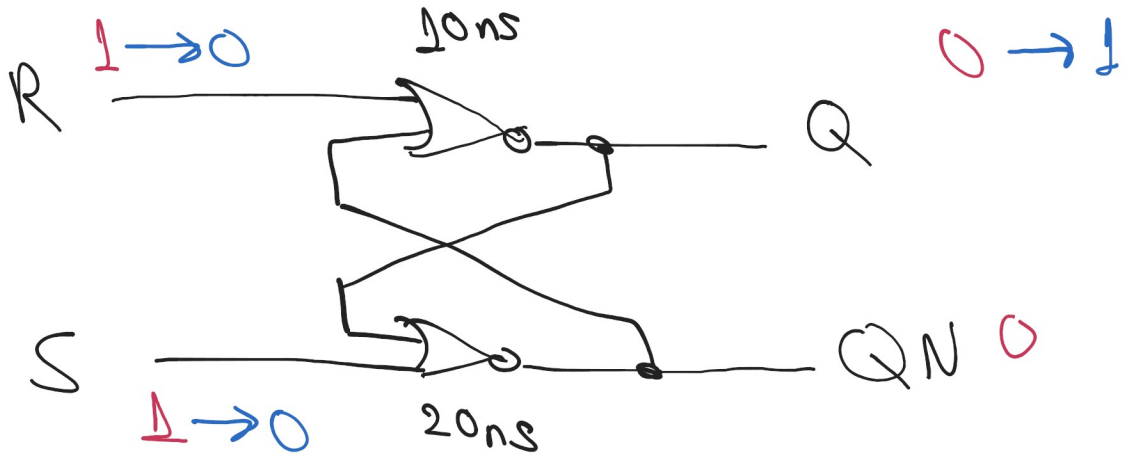
1.) You won't get oscillations

2.) You will get a stable state

MeTastability

Stable only under certain conditions

IF the propagation delays are different

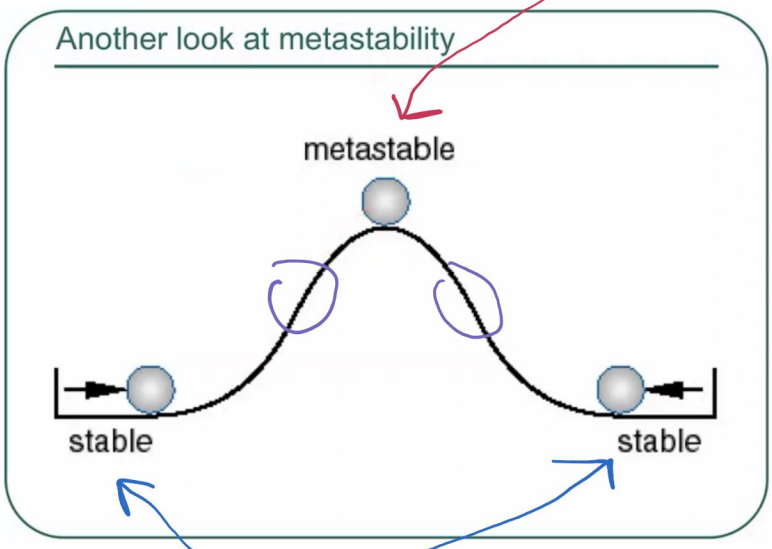


Q becomes a '1' first. QN may change to a '1' but Lag behind. Eventually the Latch will settle in one of the stable states.

→ Either the SET STATE or RESET STATE

because one output is being computed earlier Than the other output.

(The output will be stored for an additional 10 ns)

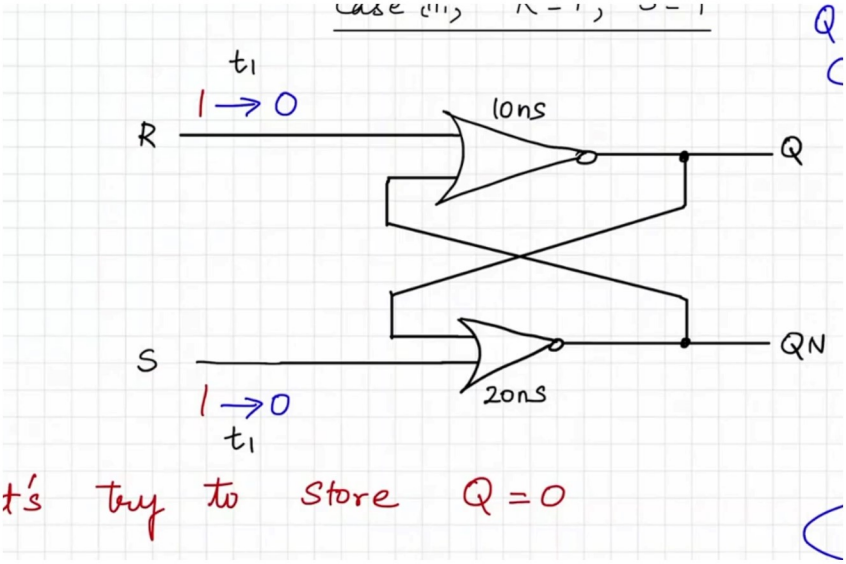


We are here when
$R = 1$ $Q = 0$
$S = 1$ $QN = 0$

We are unstable when
$R = 0$
$S = 0$

We are Here when one of the NOR gates have propagation delay

Practically we can't set $R = 0$ and $S = 0$ at the same time!



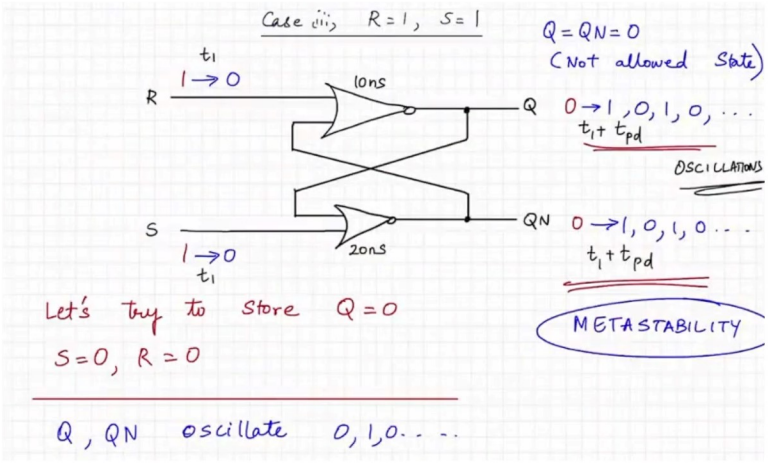
If you press/dupress switches, you can't do it at the same time.

We are talking about nano seconds

Unless it's automatic and synced, It's not possible.

Because of practical constraints, you can't implement Metastability. You couldn't monitor it with LEDs.

8



A Thought, if we never tried to store this, we wouldn't have a problem

If we don't set S=0, R=0 Then we can't store the SET STATE or RESET STATE

If we have the condition $S \neq R$
We LOSE
- STORE
- NOT ALLOWED

We need the STORE STATE

→ Find different way to STORE

If you are in the Meta stable STate, can you come out of It?

If you are in the unstable state (oscillations), can you come out of it?

You would have to go into SET or RESET state. (One of the stable states)

1.) Just set R≠S, then you go to a stable state

Let's write Characteristic Tables (NEXT State Tables) for S-R Latch

State	S	R	Q^+
STORE	0	0	Q
RESET	0	1	0
SET	1	0	1
NOT ALLOWED	1	1	X

Store state stores last Q

This is the case independent of previous output

← It will actually be a '0' but we don't care because it's not allowed

Q can be either a 1 or 0, so Let's write a Table that captures this detail

The old Q will be seen as an Input

10

	S	R	Q	Q^+
STORE	0	0	0	0
	0	0	1	1
RESET	0	1	0	0
	0	1	1	0
SET	1	0	0	1
	1	0	1	1
NOT Allowed	1	1	0	X
	1	1	1	X

We are capturing both possibilities of Q in every one of those 4 STATES

It doesn't matter what you were wearing earlier, I Reset you

It doesn't matter what you were earlier, I set you

This is why we can call this the NEXT STATE Table

Characteristic Table

S	R	Q	QN
0	0	last Q	last QN
0	1	0	1
1	0	1	0
1	1	0	0

S-R Latch Characteristic Equation

S	R	Q	Q^+	
0	0	0	0	HOLD
0	0	1	1	
0	1	0	0	RESET
0	1	1	0	
1	0	0	1	SET
1	0	1	1	
1	1	0	X	NOT
1	1	1	X	ALLOWED

K-map for Q^+

SR / Q	00	01	11	10
0	0	0	X	1
1	1	0	X	1

Simplify !!!

Characteristic equation

$Q^+ = S + \overline{R}\,Q$

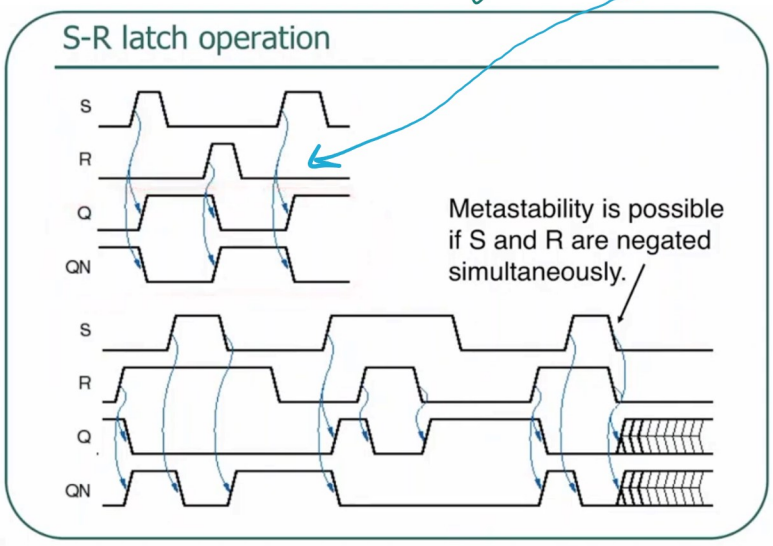
From the characteristic equation, It's clear to see why set and reset work like they do.

The output depends on the previous input. And that previous input depended on the Q before that.

This is why we call this SR Latch a sequential element.

Q^t and even Q depend on the past sequence of Q

S-R Latch Timing diagram



Causality arrows are used to indicate Q, QN are changing depending on the changes made in the S, R signals.

(Input controls)

So we assume $Q = 0$, $QN = 1$, Note, at the start we are in the STORE STATE

So technically, we don't actually know Q, QN

And there isn't a good way to know. But we can do things to Inputs S, R and then we know Q, QN

When S goes to 1, we are in the Set State

→ Therefore we know $Q = 1$

12

In the set state, your output is independent of the previous output because we set $Q = 1$

and $Q_N = 0$

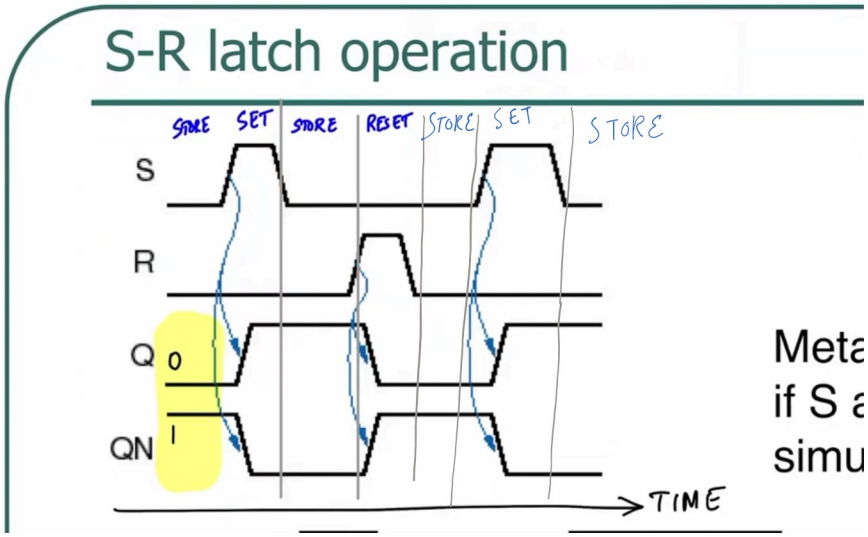
if $Q = 1$ before, it is still 1

if $Q_N = 0$ before, it is still 0

We return to the STORE STATE and then we go to reset state

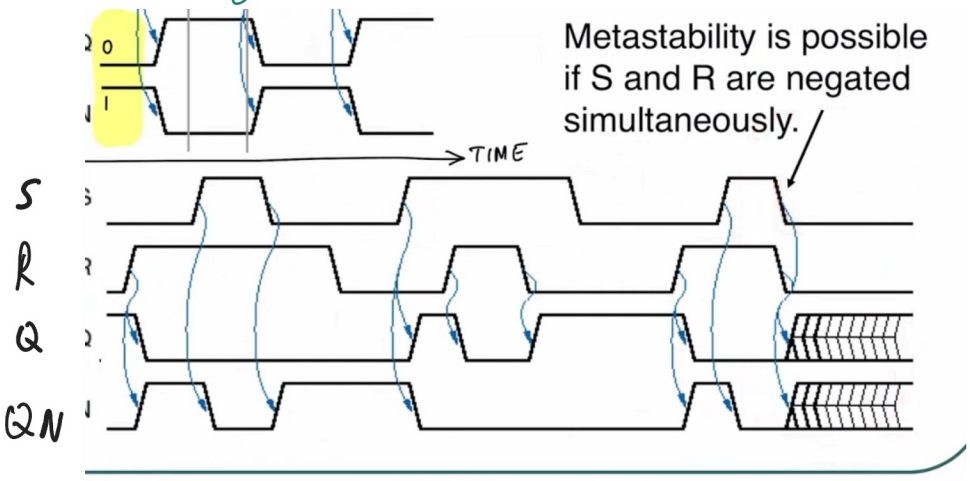
Reset state $\begin{cases} Q = 0 \\ Q_N = 1 \end{cases}$

Then we go to Store state



This is easy because we only have 3 states

Sometimes we have To worry about Metastability

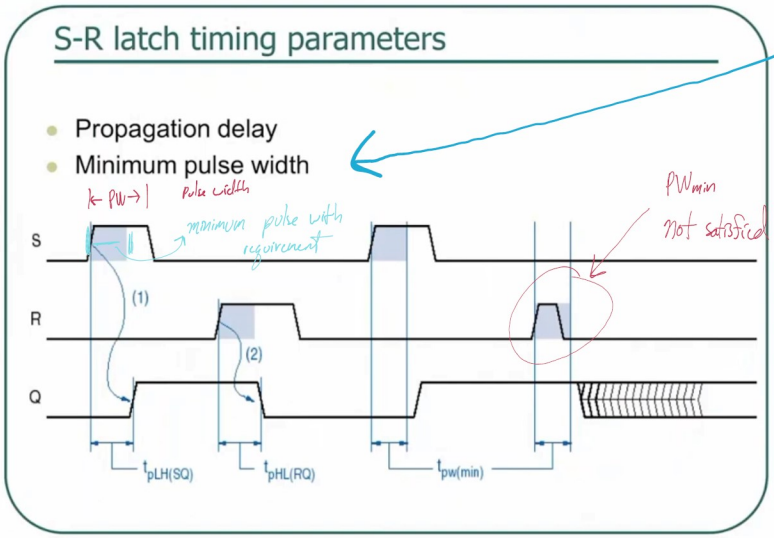


When we go from
S = 1 ⇒ S = 0
R = 1 ⇒ R = 0
simultaneously

So we see when S = 1 Q = 0
R = 1 / QN = 0

Then when S = 0 Q oscillate
R = 0 / QN

S-R Latch timing parameters



2 parameters with timing constraints

Break these constraints, and we have meta stability

14

If you do not follow the minimum pulse width requirement for an S-R Latch. (The minimum time the S input should be a '1') (or the R input)

↓

You'll end up in Metastability

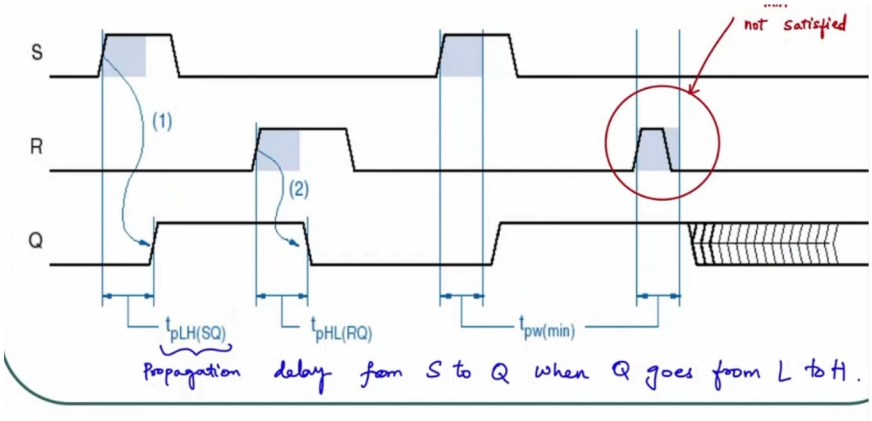
Propagation delay, we have different propagation delay

The propagation delay could be different going from Low to High and High to Low

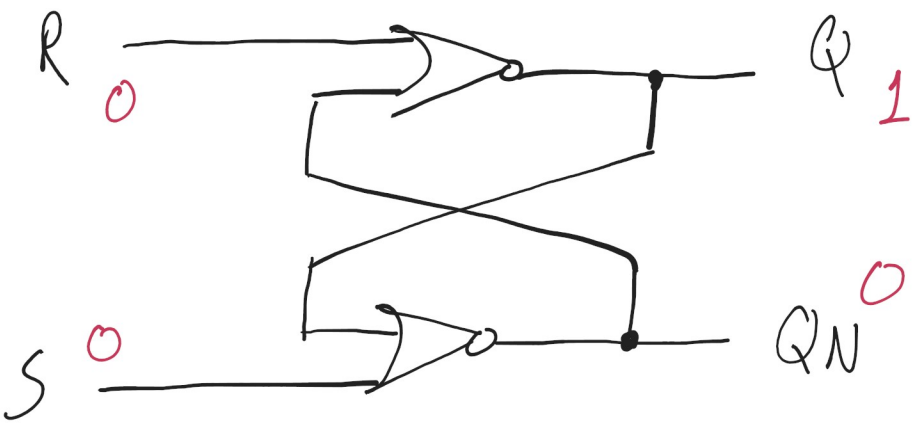
$t_{pLH(SQ)}$ $t_{pHL(RQ)}$

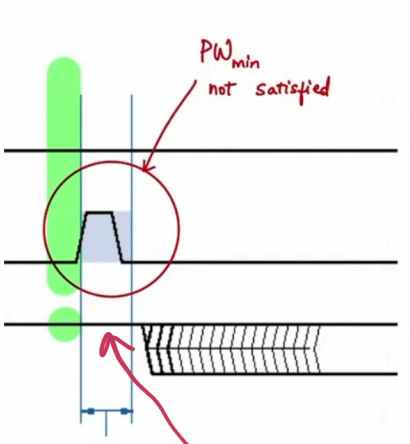
When you make S=1, the time delay it takes for Q=1

Time delay it takes to make Q=0 due to R



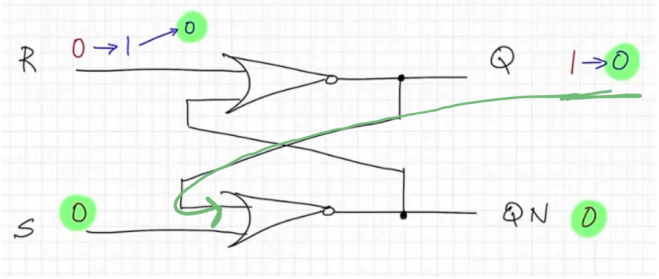
What happens when the pulse width is not satisfied?





If we try to recreate this pulse width problem,

We make R=1 for a very short amount of time



Before Q=0 goes into the Lower NOR gate, R changes back into a '0'.

Because I change it very quickly, My state is

R=0 Q=0
S=0 QN=0

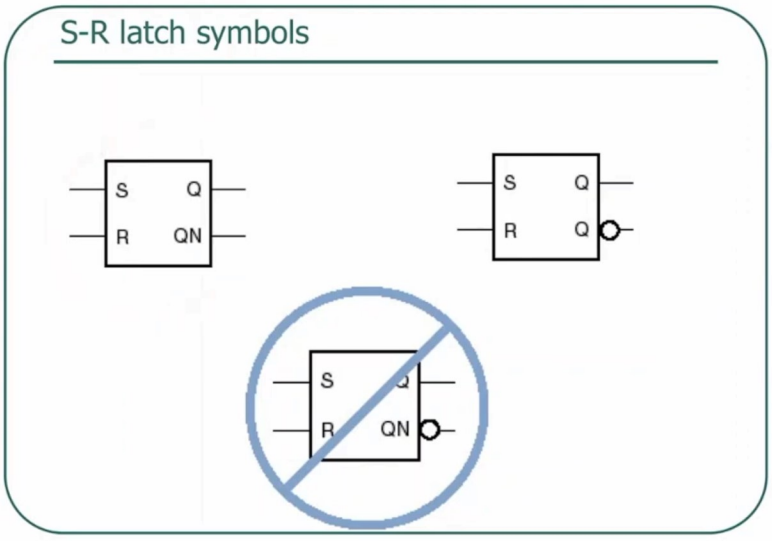
} This is the problem state from before.

We have kinda recreated that NOT allowed state which leads to Metastability

16

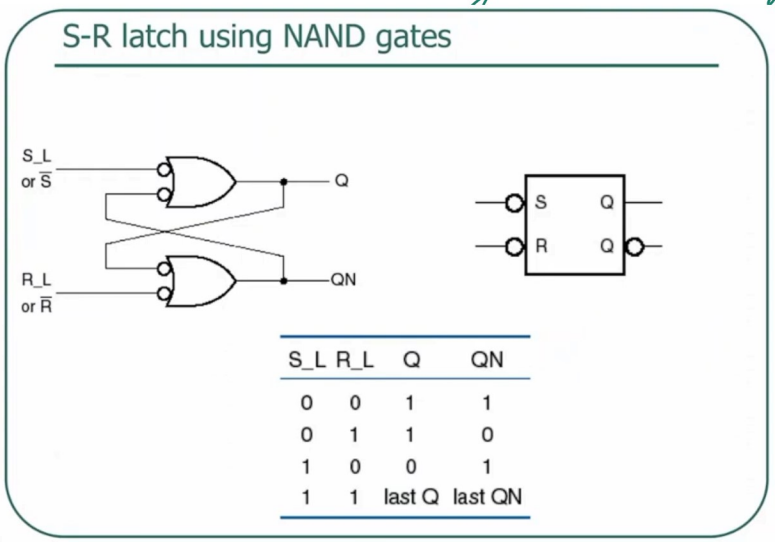
The change in R was not allowed to be reflected at both outputs. It was only reflected at one of the outputs

S-R Latch Symbols



Pretty straight forward

S-R Latch using NAND gates

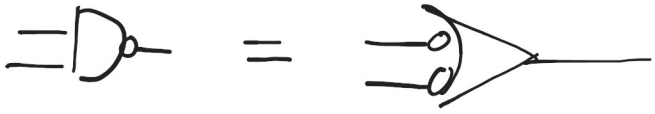


S_L	R_L	Q	QN
0	0	1	1
0	1	1	0
1	0	0	1
1	1	last Q	last QN

2 NAND Gates connected in a cross coupled configuration

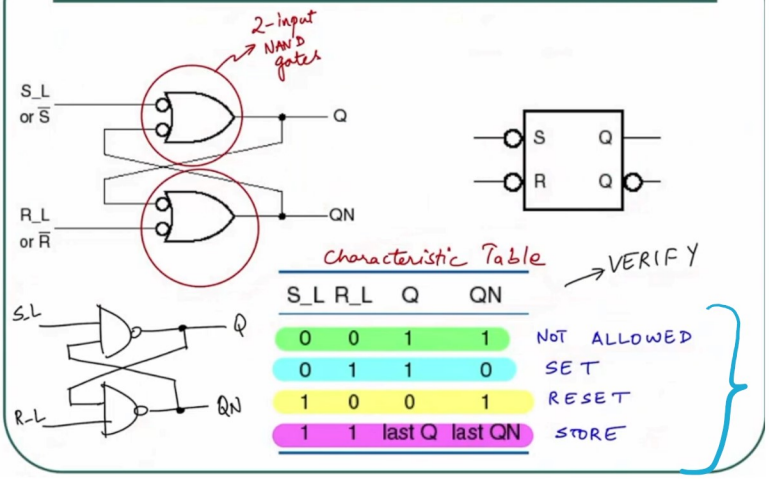
You can analyse this circuit and realize this characteristic Table

NOTE



S-R latch using NAND gates

Cross-coupled NAND-gates configuration



Characteristic Table → VERIFY

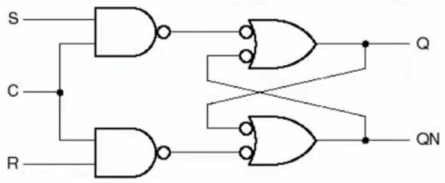
S_L	R_L	Q	QN	
0	0	1	1	NOT ALLOWED
0	1	1	0	SET
1	0	0	1	RESET
1	1	last Q	last QN	STORE

We can see the STATES from the characteristic Table.

To verify the characteristic Table, start How?

- Start from the SET or RESET STATE and try to store it
- We'll use NAND Gate Truth Table in our analysis

S-R latch with enable



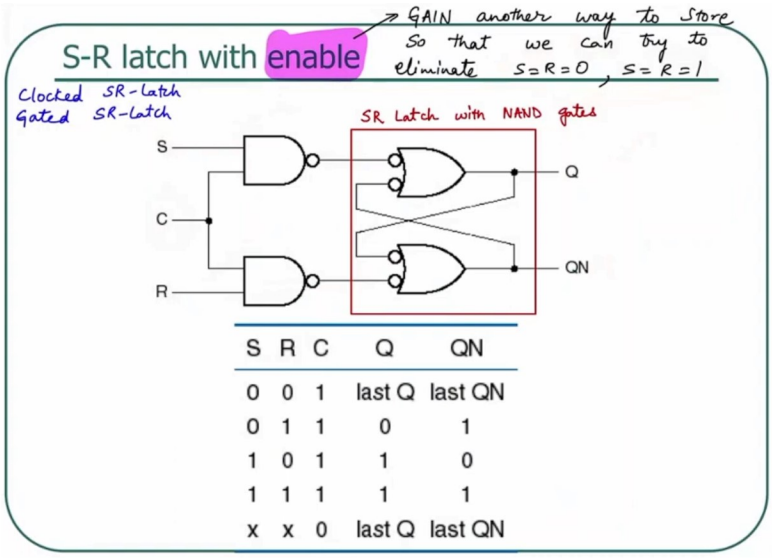
S	R	C	Q	QN
0	0	1	last Q	last QN
0	1	1	0	1
1	0	1	1	0
1	1	1	1	1
x	x	0	last Q	last QN

We had issues with the NOT ALLOWED STATE and SToring the NOT ALLOWED STATE

S=1, R=1 } Trying to store this state gives us problems

The enable addresses that issue

18 This is also called a clocked S-R Latch



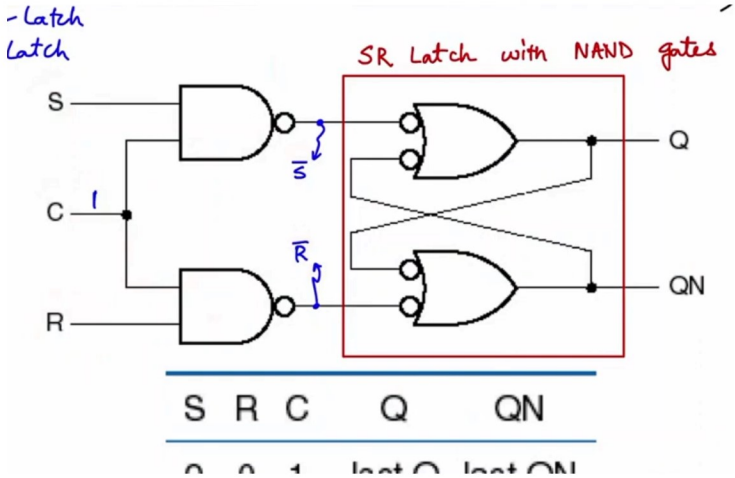
S	R	C	Q	QN
0	0	1	last Q	last QN
0	1	1	0	1
1	0	1	1	0
1	1	1	1	1
x	x	0	last Q	last QN

If we eliminate S=R=1 we can avoid meta stability and oscillation.

But if we eliminate S=R=0, we lose the store state

So we use the C input as another way to store things

Let C=1, what are the signals after the NAND gates?

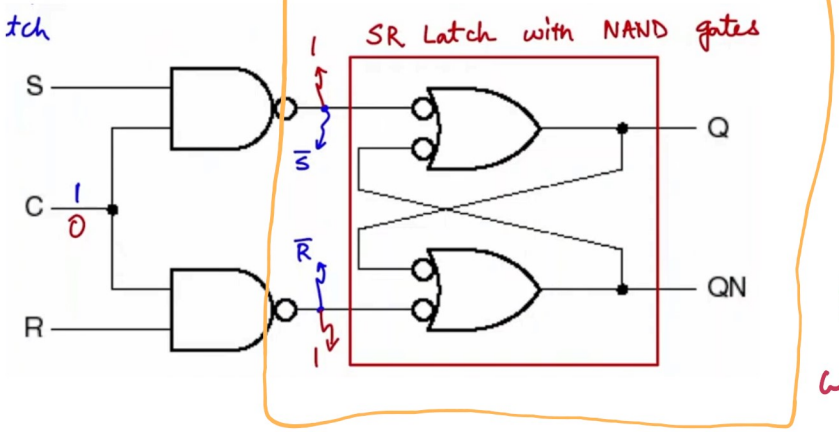


S	R	C	Q	QN

We have $\overline{S}$ and $\overline{R}$

with the enable, we have recreated the S-R Latch (with true S and true R)

When C is enabled, we have an S-R Latch with AH S and AH R. So what happens in the circuit when C=0



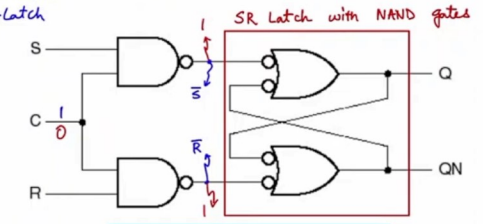
When C=0

we get an ordinary Cross-coupled NAND gate configuration with inputs S_L=1, R_L=1 which is the STORE STATE.

S-R latch with enable

GAIN another way to store so that we can try to eliminate S=R=0, S=R=1

Clocked SR-Latch
Gated SR-Latch



S	R	C	Q	QN	
0	0	1	last Q	last QN	STORE
0	1	1	0	1	RESET
1	0	1	1	0	SET
1	1	1	1	1	NOT ALLOWED.
x	x	0	last Q	last QN	STORE

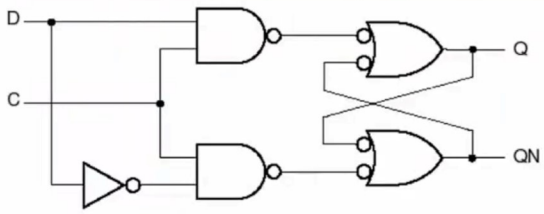
We can Think of C=0 as another STORE state

C = 1, S-R Latch

C = 0, STORE (State)

D Latch

D latch



C	D	Q	QN	
1	0	0	1	RESET
1	1	1	0	SET
0	x	last Q	last QN	STORE

We have eliminated the case where both Inputs can be the Same.

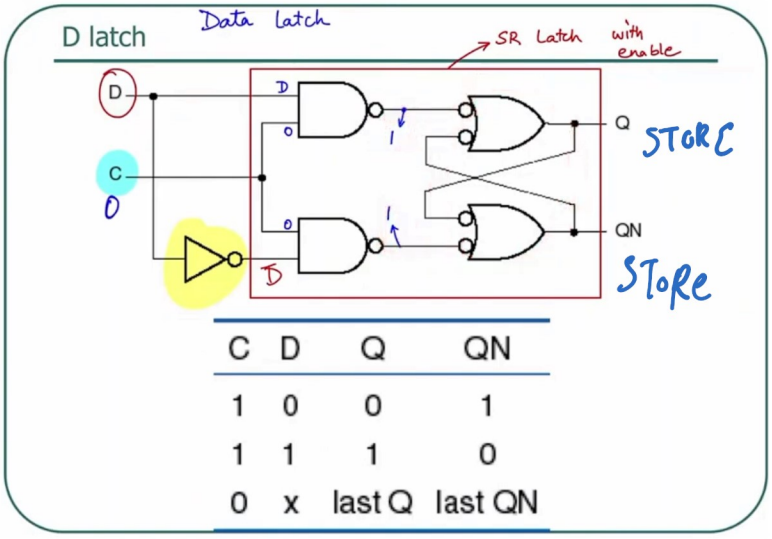
C input gives us the ability to Store

In a regular Latch, if S≠R, we Lose the STORE and NOT ALLOWED States. But input C has returned the STORE state.

What is D? DATA Latch

D Latches have a very significant place in the construction of registers.

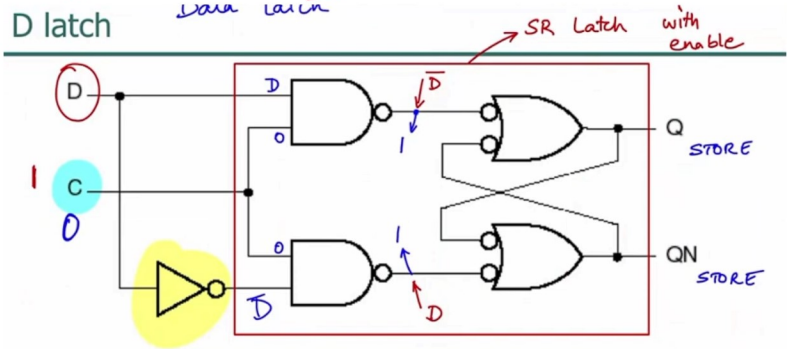
A Latch combined with another functionality will give a J-K Latch.



C	D	Q	QN
1	0	0	1
1	1	1	0
0	x	last Q	last QN

When we do circuit analysis, we clearly see that when C=0, its like setting a NAND Latch to the store state.

If C = 1 (Data Latch)

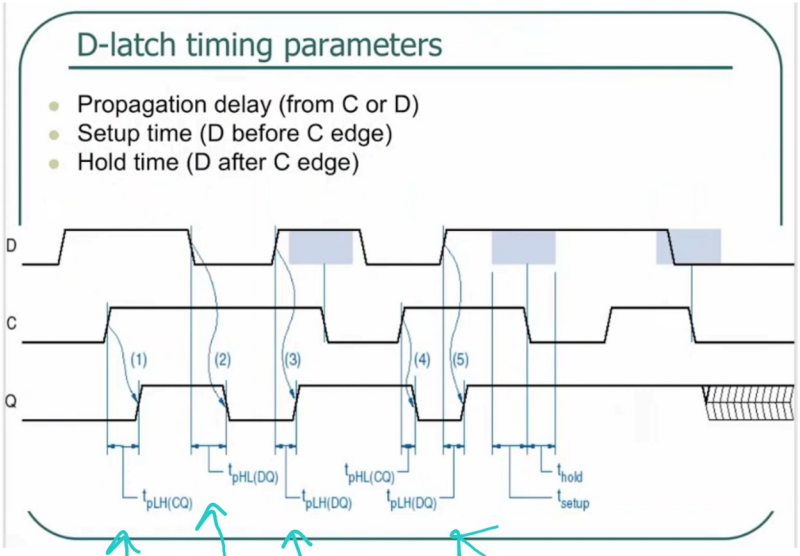


C	D	Q	QN
1	0	0	1
1	1	1	0
0	x	last Q	last QN

} SET/RESET

→ STORE

When C = 1, whatever goes in as data is reflected at the output.



When C changes or D change, output changes after certain propagation delay

as seen by causality arrow and seen by notation here

But we are adding 2 more timing constraints

SETUP time, You cannot change D input before or after there's a change in C (The edge of C)

Setup time (D before C edge)

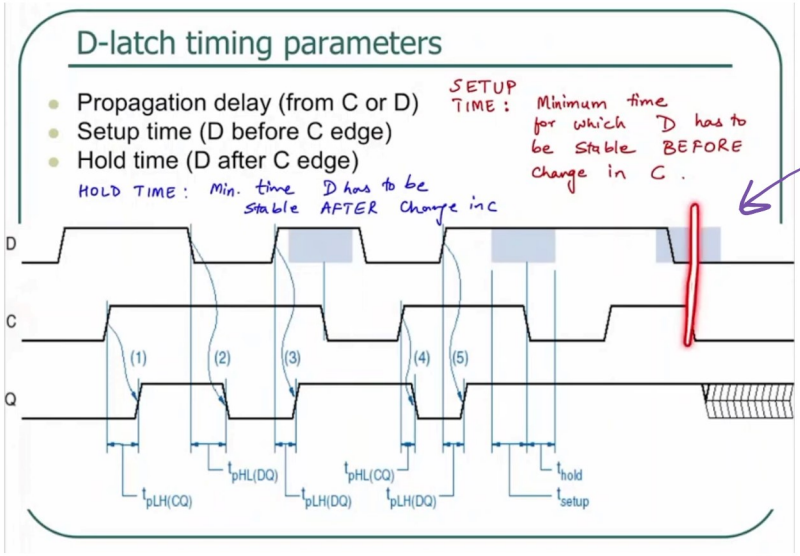
Hold time (D after C edge)

Breaking These conditions is similar to breaking minimum pulse width requirements for the S-R Latch

Violating these conditions results in Metastability

SETUP TIME: Minimum time for which D has to be stable BEFORE Change in C.

Hold TIME: Minimum time D has to be stable AFTER change in C



Which time was violated Here?

We broke setup time

⇒ caused Metastability in output Q

When is Q able to change?

when C=1 and D changes

The only time when C can change is when C=1

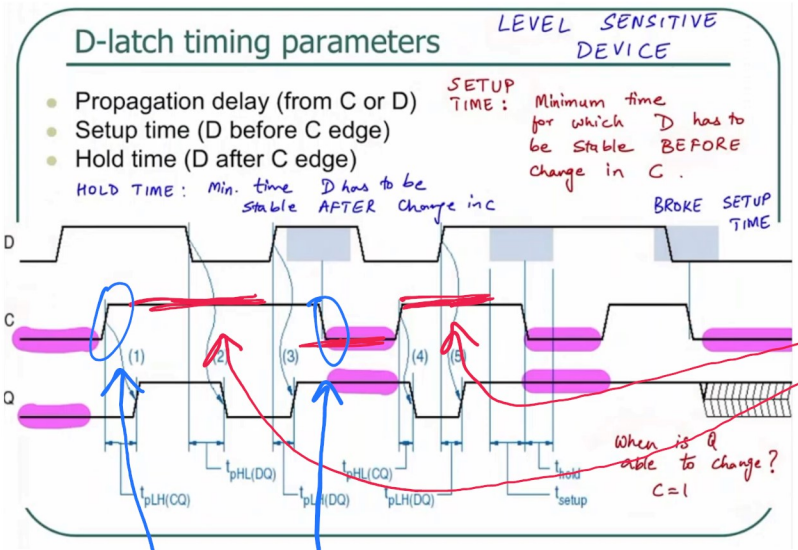
- cannot change when C=0
  - Even when there's a change in D
- cannot change when C goes from LOW to HIGH
- cannot change when C goes from HIGH to LOW

Because Q can only change when C=1

we call this circuit element a LEVEL SENSITIVE DEVICE

You want to change things? Adjust the LEVEL of the clock.

You Don't want to change things? Adjust the LEVEL of the clock

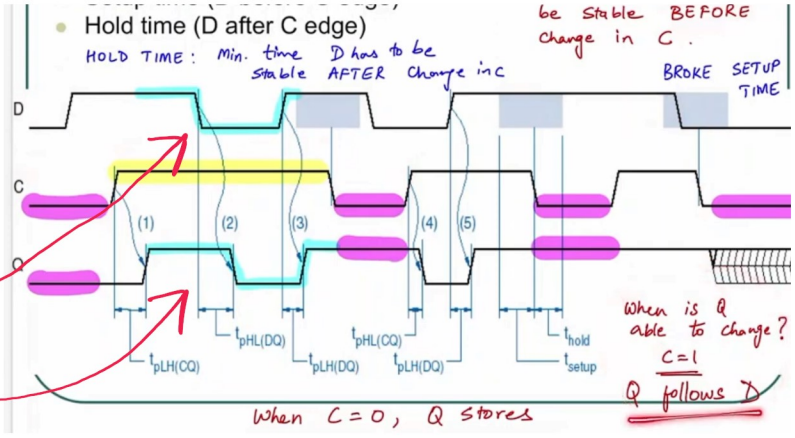


Level Sensitive, The operation of the device concerns the Levels

As opposed to the EDGES

When C=0, Q stores

When C=1, Q follows D



24 Why care about this? What's the motive?

We want to gain control over when things can change.

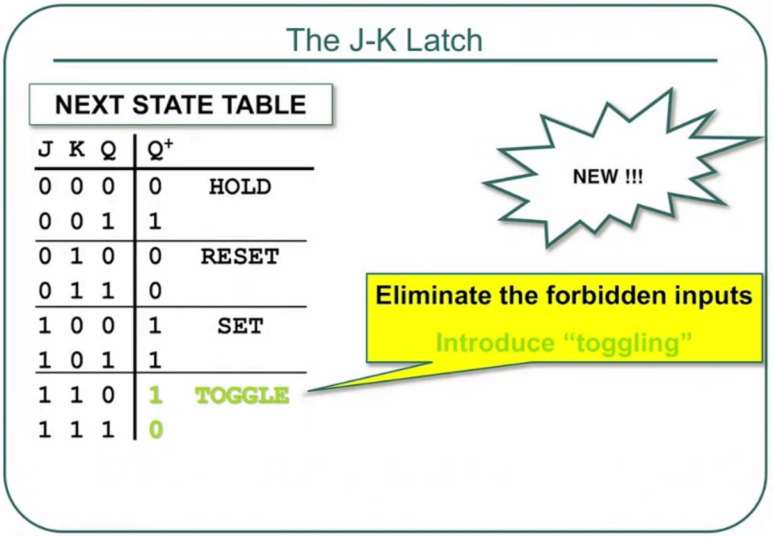
— Latches give 'Loose' control over Things because They are Level sensitive

— Flip flops give us more 'precise' control (control over when we can change things) because They are EDGE Sensitive

---

J-K Latch

Jump-Kill Latch



Jump: Jumping the output is like setting the output

Kill: Killing the output is like resetting the output

This has a similar functionality to The SR Latch but The NOT ALLOWED state is replaced with the TOGGLE state

This is a SR Latch with TOGGLE functionality

SR Latch with TOGGLE FUNCTIONALITY / The J-K Latch — Jump-Kill Latch

NEXT STATE TABLE

J	K	Q	Q^+	
0	0	0	0	HOLD
0	0	1	1	
0	1	0	0	RESET / Kill
0	1	1	0	
1	0	0	1	SET / Jump
1	0	1	1	
1	1	0	1	TOGGLE
1	1	1	0	

NEW !!!

Eliminate the forbidden inputs

Introduce "toggling"

Whatever it was before will TOGGLE (it will flip)

0 flips → 1

1 flips → 0

J-K Latch Characteristic Equation

J-K Latch Characteristic Equation

NEXT STATE TABLE

J	K	Q	Q^+	
0	0	0	0	HOLD
0	0	1	1	
0	1	0	0	RESET
0	1	1	0	
1	0	0	1	SET
1	0	1	1	
1	1	0	1	TOGGLE
1	1	1	0	

K-map for Q^+

Q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

Simplify !!!

Characteristic equation

$$Q^+ = J\overline{Q} + \overline{K}Q$$

So if we know J, K, Q, we use the characteristic equation to get Q^+