ECSE 2610 Computer Components & Operations

Lecture 8.1

Latches

Some Remarks

In the Lectures from now on and to the future, we shift the conversation.

So far, we have Looked at Combinational Logic. which are things Moving from one side to the other. (Inputs to outputs)

Usually F (output) depends on the combinations of the present Inputs

→ So can we actually store data?
can we build Logic circuits that have the ability to change information?

↳ can we store this information?

2

Latches are your basic element that allows you to store 1 bit of information.

If you understand Latches well, you will understand everything that follows well,

- Flip-Flops
- Registers
- Counters
- Finite State Machines

} These guys are based on Latches

What have we done so far?

What We Have Covered

Basics

- **Number systems**: positional number notation, binary number, number system conversion, two's complement ...
- **Computer codes**: Gray code, parity code, and BCD code.
- **Digital circuits**: logic signal, MOS circuits, noise margin, fan-in/out
- **Boolean algebra**: basic Boolean operations, laws and theorems, DeMorgan's law, standard representations, logic minimization using K-map.

Combinational Circuits

Programmable Logic Device; Decoder; Encoder; Three-State Devices; Multiplexers; Parity Circuits and Comparators; Adders, Subtractors, Multipliers.

What's next? Sequential Logic Circuits

Combinational vs. Sequential Logic

Combinational logic circuits

Circuits whose outputs are a function of their current inputs only

Sequential logic circuits

Circuits whose outputs are a function of their current inputs AND stored information about previous inputs

Contain storage elements

Contain feedback connections

Output (S) depend on combination of present Inputs

→ present inputs and SEQUENCE of past Inputs

EX) A door code

- To gain access to a room, you need to Input a 4 digit code sequentially

- Circuit not only needs to respond to ~~current~~ most recent Inputs but also keep track of what has come before (Past Inputs as well)

A Thought About How to implement Memory in circuits

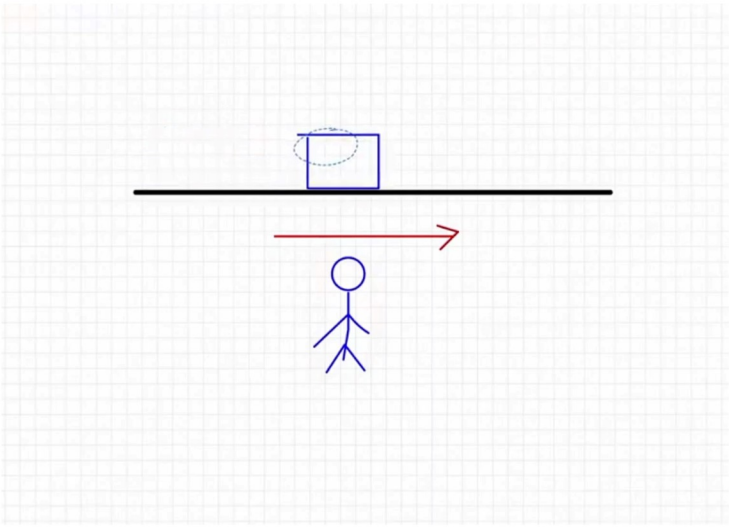
- It's the same way you are expected to memorize something

4

Methods

- Revise (look at the same information again and again)

Conveyor Belt



If this guy needs to look at this box again after a minute, what needs to happen?

The Box needs to be moved to the Left and after a minute, the box will again be in front of the human.

In circuits, things move from Left to Right

- when we change inputs, the outputs of the Logic gates change by those inputs

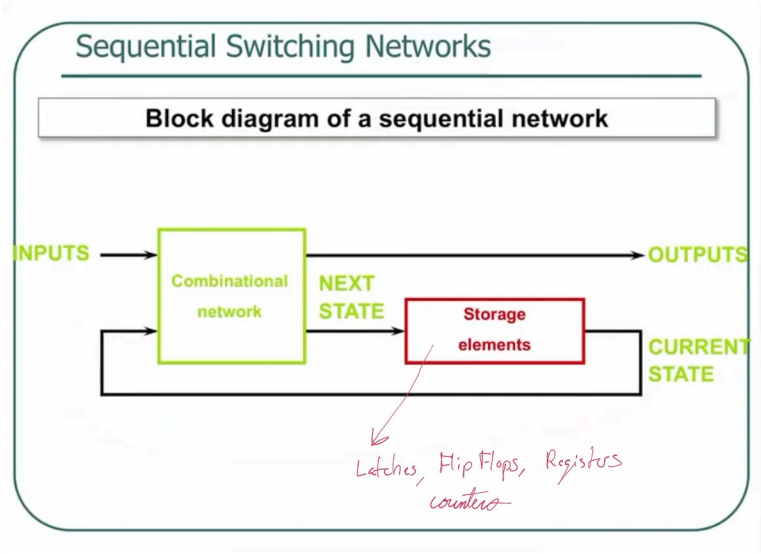
We can physically move the box to the Left but how is this principle implemented in circuits?

Feedback

If you have Feedback you are looking at that some information that was there earlier again after some time.

Memory is being implemented via feedback

Sequential Switching Networks



sequential → just means it has memory

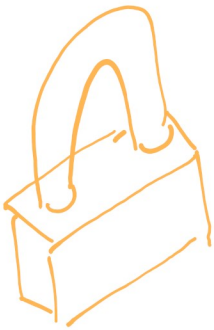
- Memory can have different forms. And can be implemented using a variety of storage elements.

Flip Flops are made from Latches

Registers are made from Flip Flops

FlipFlops are used to make counters

So what's happening in the diagram?
— we are taking some of the outputs of this combinational network
— called NEXT STATE



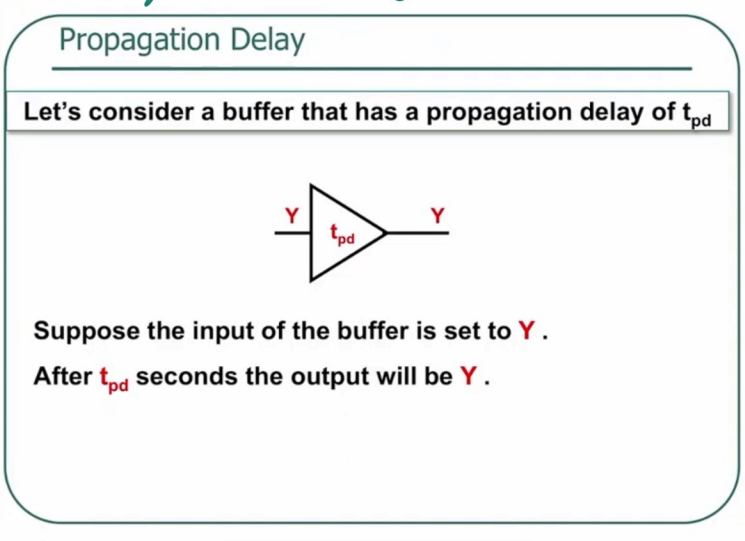
Locked
↑
This is a STATE

unLocked
↑
This is another STATE

This is how to think of STATES

So Current State is fed back into

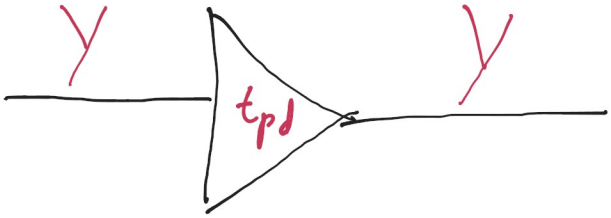
Propagation Delay



When you put things on a conveyor belt, there's a delay before the box comes in front of you.

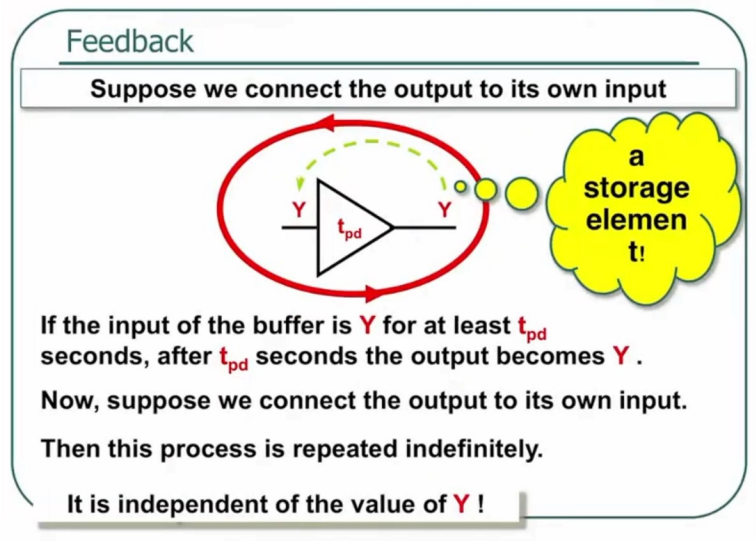
In our context, that delay is the propagation delay

This is a buffer with propagation delay t_{pd}



We understand delay, but now we want to use that delay to help us store information

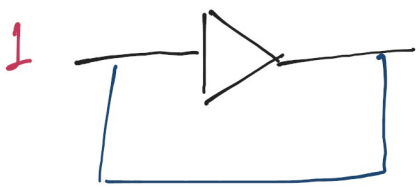
The way a storage element is built is by doing a Feedback



So what happens when we connect the output back to its own input.

Example

Let's say that somehow we have a way to make the Input into a 1



8

So After tpd seconds, The output becomes a 1

& now Y = 1, and this will hold the input at 1

This will persist in a circular loop

So as long as we don't make the Input 0, this 1 will be stored in that buffer

→ This is the Feed back we will leverage to implement latches

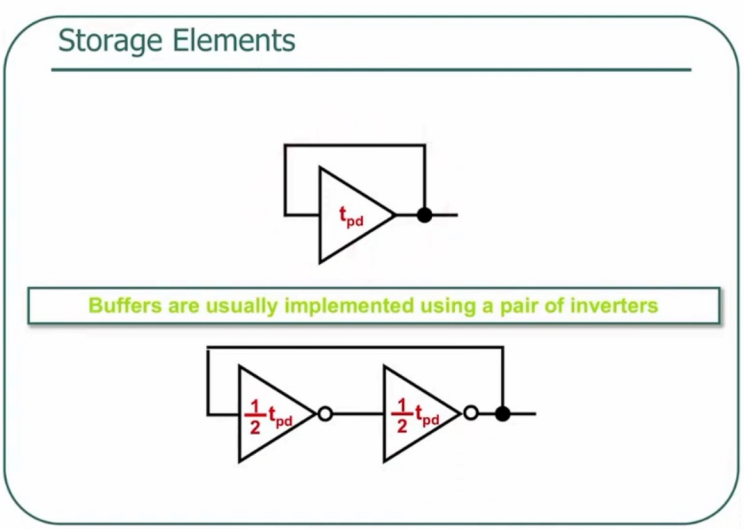
We physically want to store a '1' or a '0'

STORE 1-bit

① Make a '0' and Store it

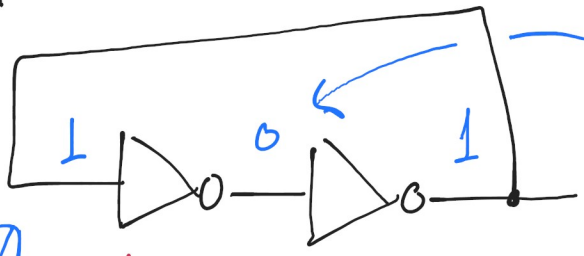
② Make a '1' and Store it

Storage Elements



2 NOT Gates are used to make a buffer

If



$\frac{1}{2}t_{pd}$ $\frac{1}{2}t_{pd}$

Half of propagation delay

If we somehow make this a 1, This is a 0 after $\frac{1}{2}t_{pd}$ seconds. The output is a 1 after $\frac{1}{2}t_{pd}$ as well

→ This is called a STABLE STATE

because it will continue to be in this state

$1 \rightarrow 0 \rightarrow 1$

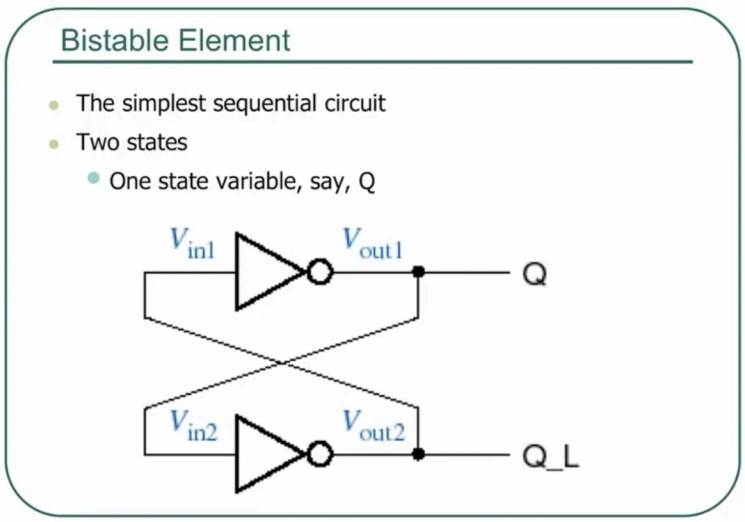
until I go in and change something

10

If I connect 2 NOT Gates back to back, I've got 2 Stable states

→ Bi STable configuration
— named so because we have 2 stable states

Connecting things in Feedback allows us to get some stability and allows us to store



This is not very useful because we aren't able to change the inputs very easily

This still stores information

We have 2 NOT Gates Configured in a Cross-Coupled configuration

The output of the first NOT gate is connected to The Input of the second NOT Gate

The output of the Second Not gate is connected to the first NOT Gate

Q
Q_L
} are used to indicate that the output of this bistable element is available in its true form and its complemented form

Example

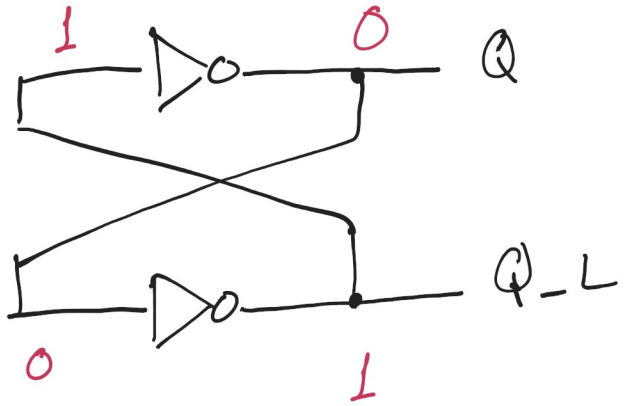
Suppose we find a way to make V_{in1} a 1
– so what would be V_{out1} (Q)?

→ a 0

→ This 0 is the input V_{in2}

→ $V_{out2} = 1$ ⟶ $Q = 0$, $Q\_L = 1$

Is this going to be stable or not?

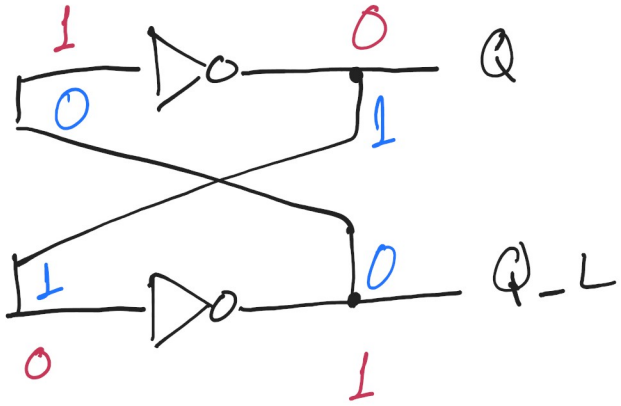


This will be stable as long as we don't change anything

12

So what does an Unstable situation look like?

It will be a situation where Q and Q_L keep changing



The red is one stable state

The blue is the other stable state

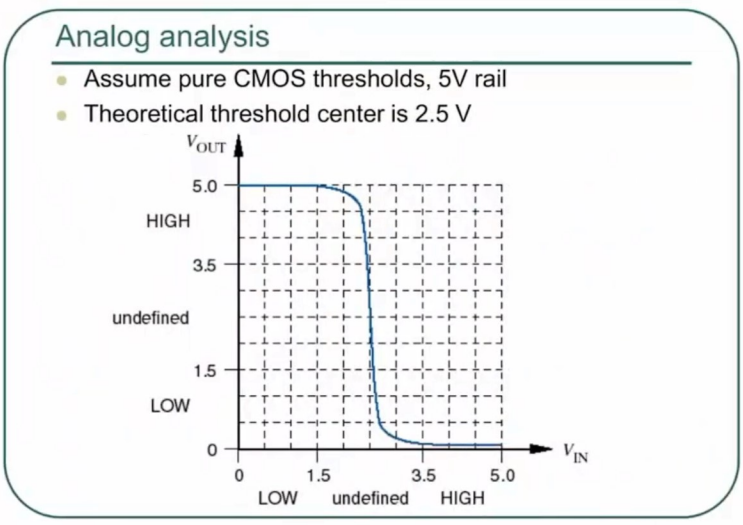
we have 2 stable states — when Q = 0; when Q = 1

Note: Q is used to indicate a transistor, but it also is commonly used to indicate the output of a sequential element

↳ Latch, FF, counter, register

Q is used to indicate the output of those elements

Discuss more about cross-coupled configuration



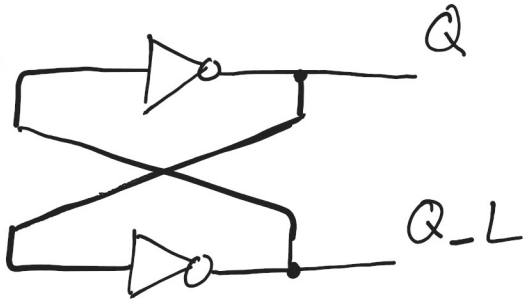
We have a problem.
We have 2 stable states but we also have a meta stable state.

Meta stability is stability on knife's edge
— It's stable but under strict conditions

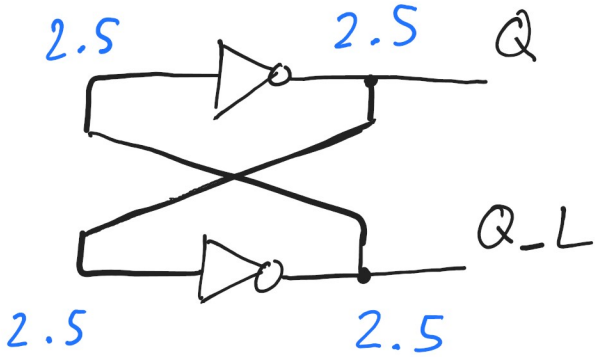
The Graph is a CMOS characteristic of a NOT gate
— When Input is Low, output is high
— when Input is High, output is Low

What if the input is right in the center? (2.5 V)
what would be the output of a NOT Gate?

what if you applied 2.5 V Here? →



What would be the output Q?
What would be the output Q_L?



Everything would be 2.5V as stated in the CMOS characteristic

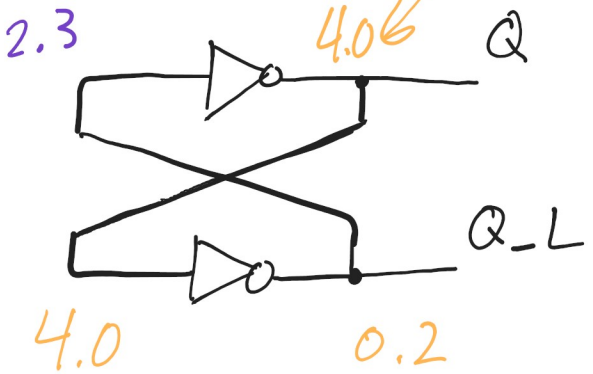
However, it would only be stable at that point

Nothing is exactly 2.5 V

what if you have a very small amount of Noise in your system that changes — 2.5 → 2.4 or 2.3

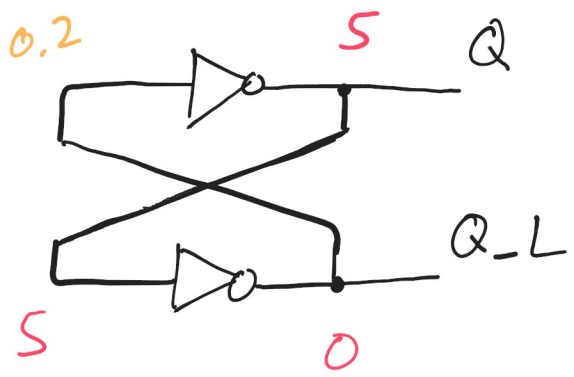
what would happen then?

What if the noise causes the input to change to 2.3?



When you look at the CMOS characteristic, Q will increase

Now 0.2 will feed back into the 2.3



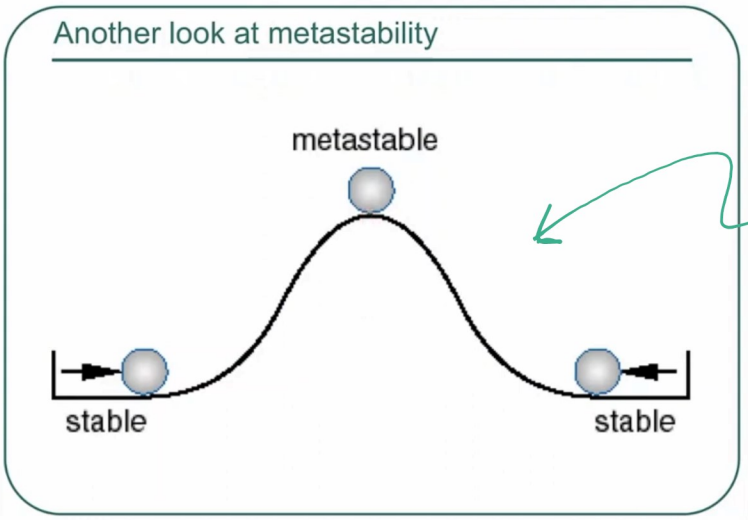
So NoW 0.2 is at our initial stage, what will be Q now?

Q = 5V

Now our 0.2V becomes 0V

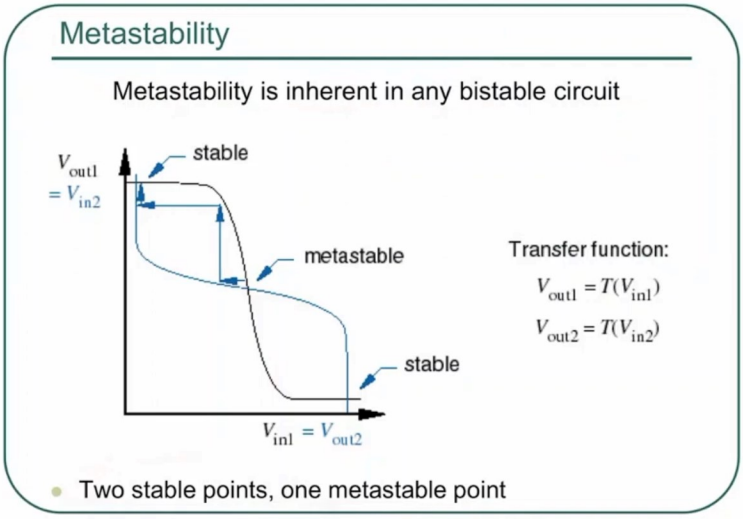
→ We went from the meta stable state to one of the stable states

It will remain at that stable state



Metastable is stable

The roll down the hill (either side) is unstable



16

Why do we care about metastability?

→ We have to worry about Timing constraints in sequential circuits

Why all the harping on metastability?

- All real systems are subject to it
  - Problems are caused by "asynchronous inputs" that do not meet flip-flop setup and hold times.
  - Details in Chapter-7 flip-flop descriptions and in Section 8.9.
- Especially severe in high-speed systems
  - since clock periods are so short, "metastability resolution time" can be longer than one clock period.
- Many digital designers, products, and companies have been burned by this phenomenon.

This is similar to setup and hold times that we talked about

→ When can an input change before or after another thing changes in a circuit that will cause metastability?

You don't want metastability because that will make your output become unstable in a realistic scenario.

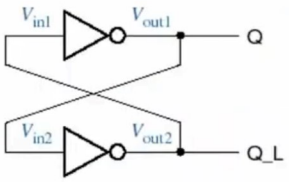
Metastability and instability will happen in our sequential ~~circuits~~ elements due to the user breaking certain hold times or setup time requirements.

→ These need to be followed when using Latches and FlipFlops

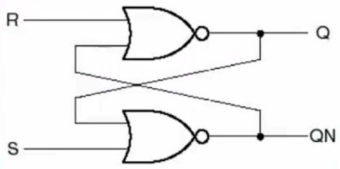
Here is the NOR version of the S-R Latch

Back to the Bistable....

- How to control it?
  - Control inputs
- S-R latch



Cross-coupled NOR gates



Characteristic Table

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

What does S and R stand for?

S = set

R = Reset

latch = latch on a door

(To make sure nothing goes in or out)

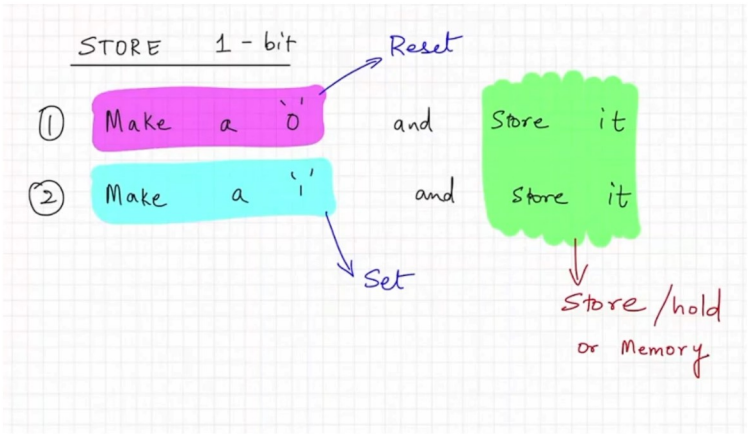
We were using a cross-coupled NOT gate configuration.

So why do we need to go for a Cross-coupled NOR gate configuration?

We have this configuration to gain control over the inputs.

Now we have a way to influence the inputs. Why would we want to do that?

Remember How we store 1 bit.



we have to make it a '1' or a '0' first

The Green stores The previous Q

How could analysing 2 NOR gates be complicated?

Well, it is complicated.

We must analyze the S-R latch to derive the characteristic Table. (Next State Table)

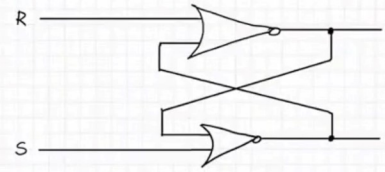
We are no longer dealing with Truth Tables.

- Truth Tables work for Combinational Logic
- Sequential Logic uses characteristic Tables

SR Latch Analysis

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,

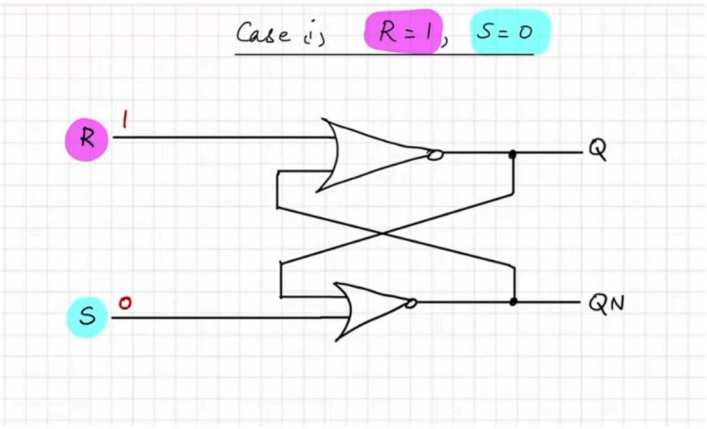
If any input is high then, output is low, irrespective of other input.

We know the truth Table of a NOR gate

- If one Input is '0', I have to wait for the other input to come in. Only when we know the other ~~Input~~ output is zero do we know if the output will be 1

If we know one of the Inputs is '1', I don't have to wait for the other Input to come in.

Case 1



So what is Q?

Right away we know $Q = 0$ because $R = 1$

(so one of the inputs to the NOR gate was 1, we already know the output)

Next Question, when did output Q change to '0'?

Let's say R changed to '1' at t_0

→ so does Q change to '0' at t_0 or $t_0 + t_{pd}$

So if we know that $Q = 0$ and $S = 0$, we know that Q_N is going to be $Q_N = 1$

Is this going to be a stable state?

$Q_N = 1$ and $R = 1$ will keep $Q = 0$

As long as $R = 1, S = 0$ Q will be reset to '0'

$Q = 0$, $Q_N = 1$ } Reset state

Now Let's try to store it

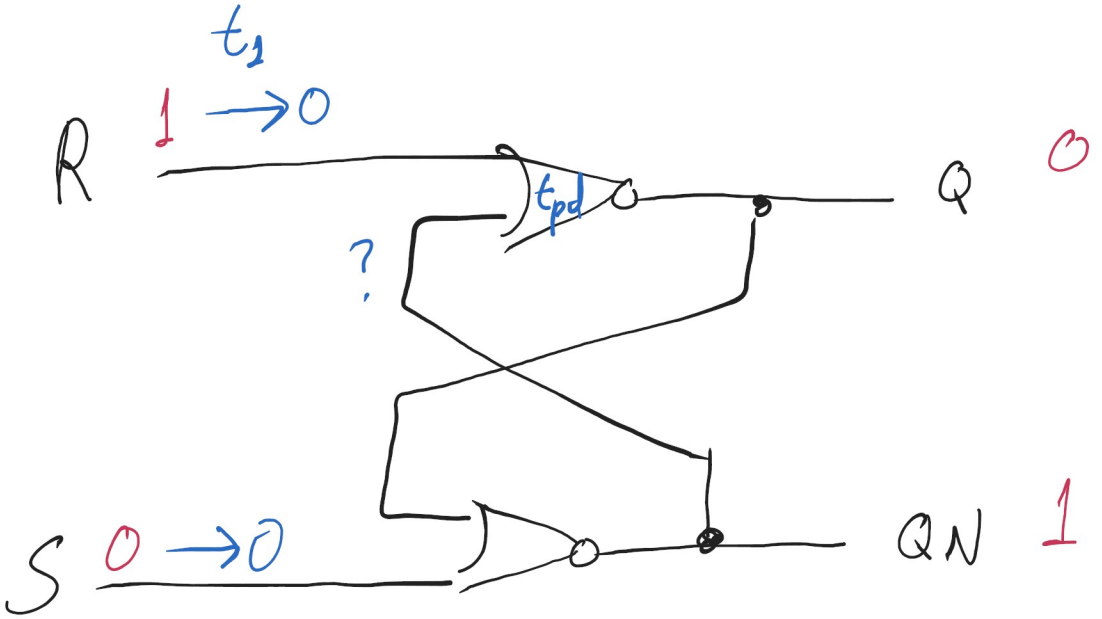
Let's try to store $Q=0$

We made a '0' so Let's try to store it

The way to store things is by making both Inputs '0'

$S=0$ $R=0$

for both of these conditions, the claim is that $Q=0$ will be stored.



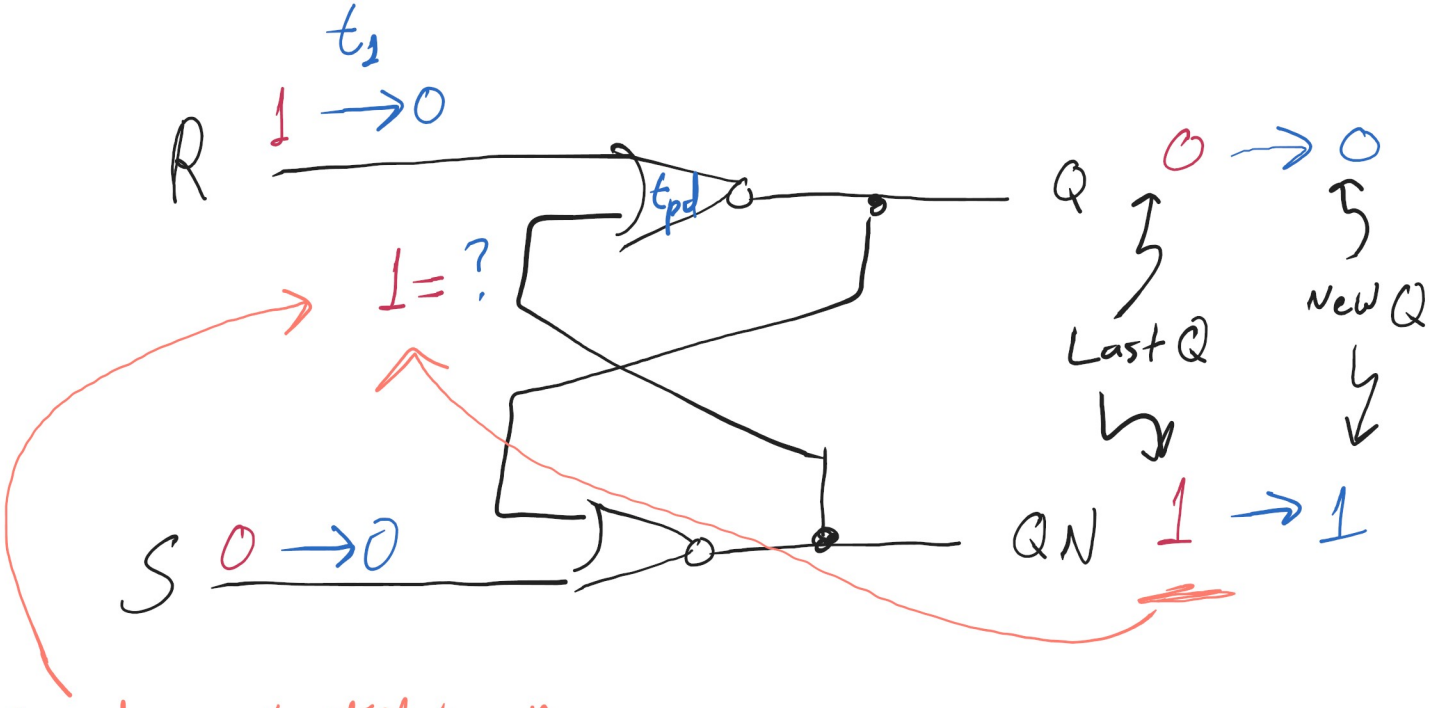
At t_1, R becomes 0. What is at the '?' point?

QN is still a '1', it hasn't changed yet. So $Q=0$ for now. (Q still equals 0)

At t_1, QN still equals 1

The idea is that it will take time for QN to change. QN does not change at t_1

→ Propagation delay helps us store things



Therefore Q = 0 (because there's a 1 at the NOR gate)

Last Q = New Q Hence, stored STORE STATE

Propagation delay is why this is able to work