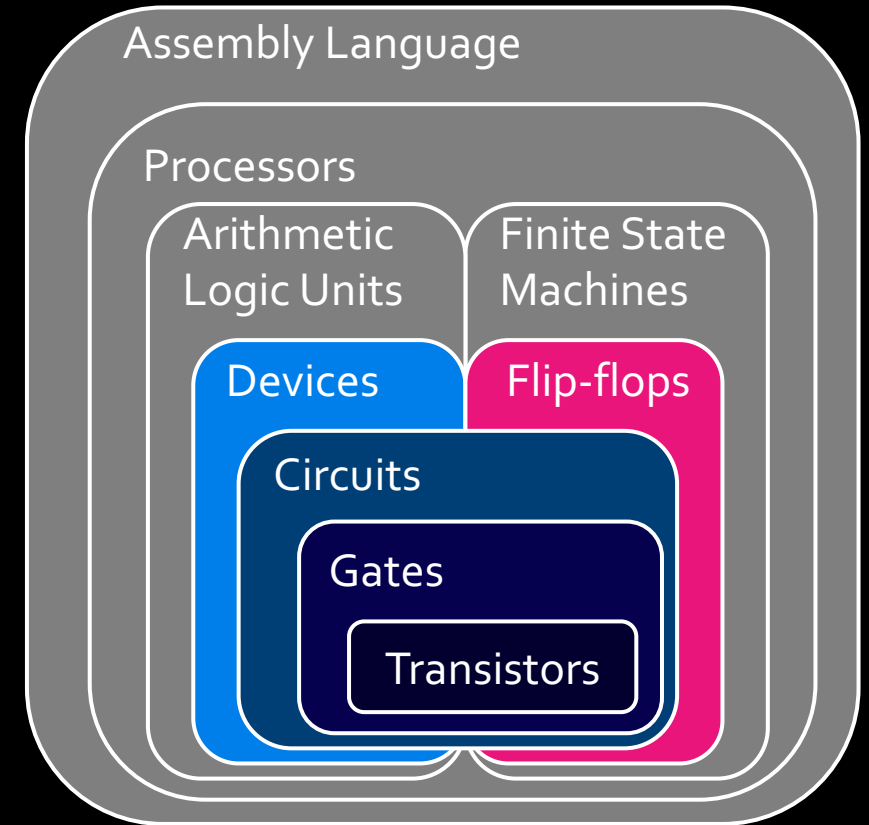


CSC258 Week 4

New Topic: Sequential Circuit



Memory

Why do circuits have memory?



Something we can't do yet

- How does the Tickle Me Elmo work?

<https://www.youtube.com/watch?v=zG62dirxRgc>

- Same input, different outputs.
- The circuit somehow remembers how many pushes have been made.

Two kinds of circuits

- So far, we've dealt with **combinational circuits**:
 - Circuits where the output values are entirely dependent and predictable from the input values.
- Another class of circuits: **sequential circuits**
 - Circuits that also depend on both the inputs and the **previous state** of the circuit.

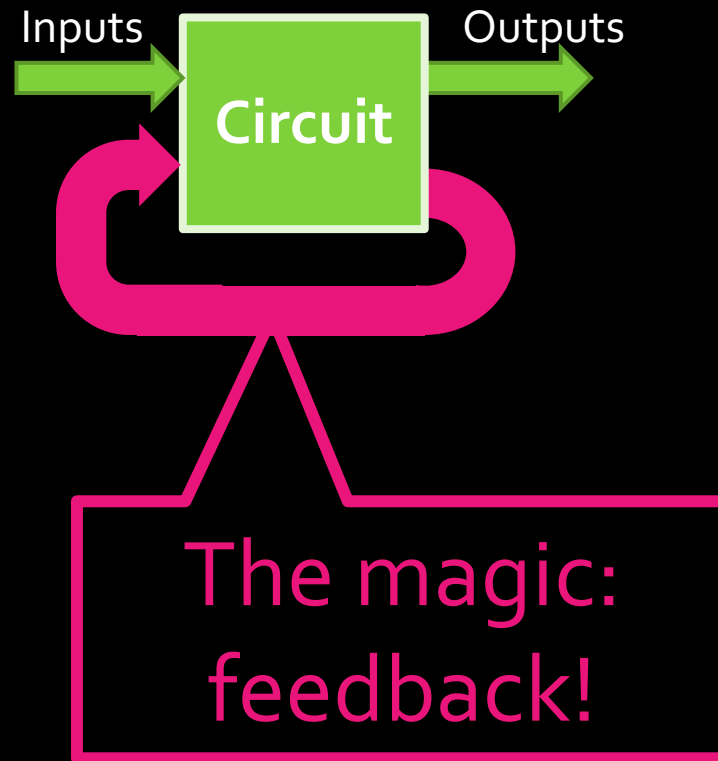
In other words...

- Combination circuits does NOT have memory
 - Everything is based on current input value
- Sequential Circuits have **memory**.
 - They **remember** something from the past (the previous state of the circuit), and its output depends on that
 - It is why computers have memory (e.g., RAM)

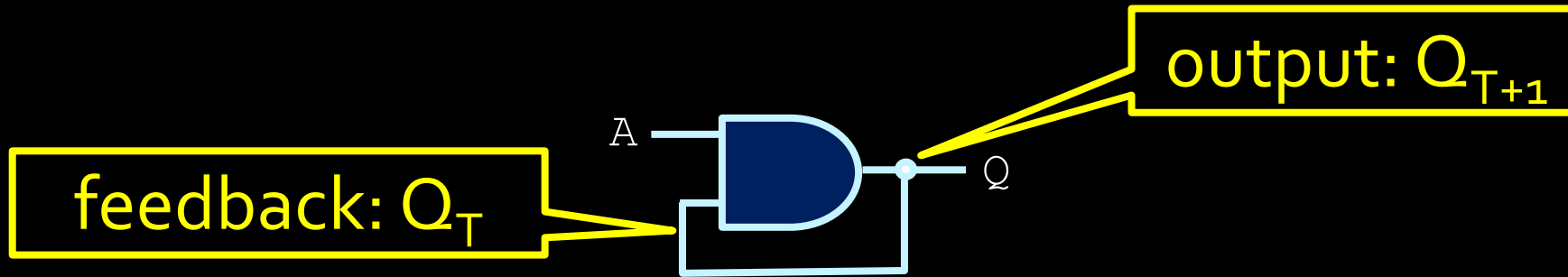
Sequential circuits

- This creates circuits whose **internal state** can change over time, where the same input values can result in different outputs.
- Why would we need circuits like this?
 - Memory values
 - To remember stuff
 - Reacting to changing inputs
 - “Output $X = 1$ when input A changes”

How can a circuit have **memory**?



Truth table of a sequential circuit

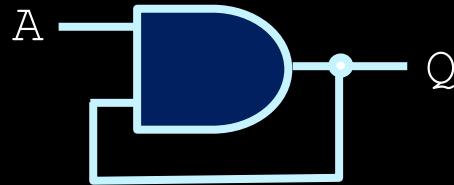


A	Q_T	$Q_{T+1} = A \cdot Q_T$
0	0	0
0	1	0
1	0	0
1	1	1

In these truth tables, Q_T and Q_{T+1} represent the values of Q at a time T, and a point in time immediately after (T+1)

The output not only depends on the input A, but also depends on the previous state of the circuit.

AND with feedback



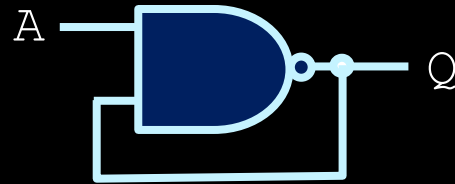
A	Q_T	$Q_{T+1} = A \cdot Q_T$
0	0	0
0	1	0
1	0	0
1	1	1

(0, 1, 0) is a transient state since it will become (0, 0, 0) immediately

Once the output is 0, it will be stuck at 0, no matter how you change A.

It has a memory that cannot be changed.

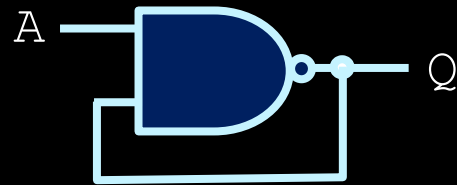
NAND with feedback, more interesting



A	Q_T	Q_{T+1}	
0	0	1	Transient (-> (0, 1, 1))
0	1	1	
1	0	1	Oscillates between each other
1	1	0	

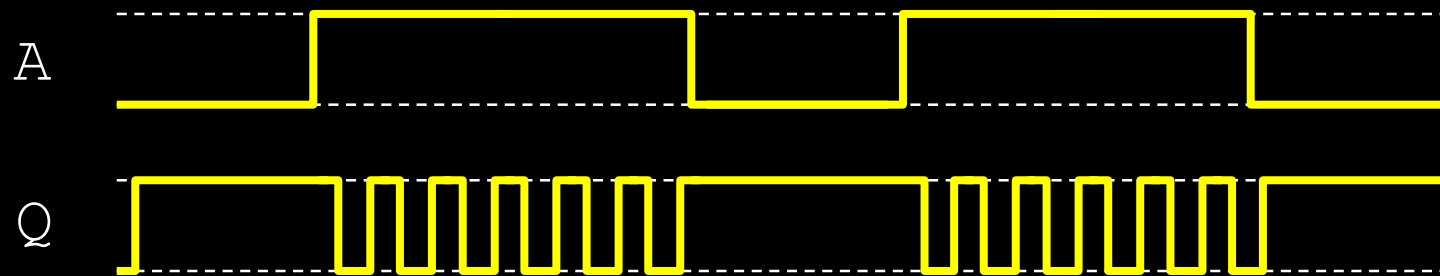
Output Q_{T+1} can be changed by changing A

NAND waveform behaviour



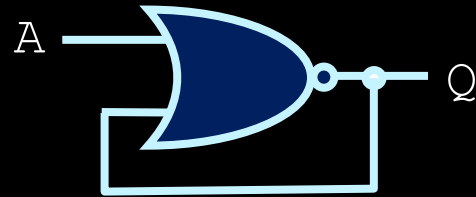
A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

Oscillates



Input A can control output Q, but the behavior of Q is not very stable.

NOR with feedback



A	Q_T	Q_{T+1}
0	0	1
0	1	0
1	0	0
1	1	0

Oscillates between each other

Transient (-> (1, 0, 0))

Output Q_{T+1} can be changed by changing A

Feedback behaviour

- NAND behaviour

A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

- NOR behaviour

A	Q_T	Q_{T+1}
0	0	1
0	1	0
1	0	0
1	1	0

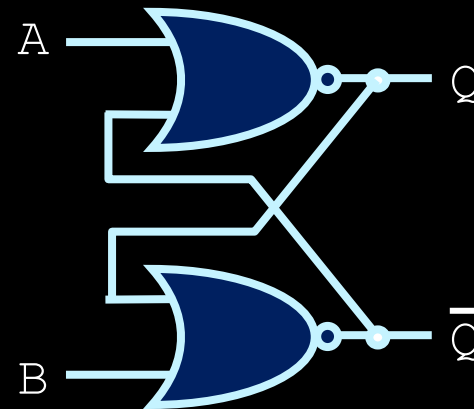
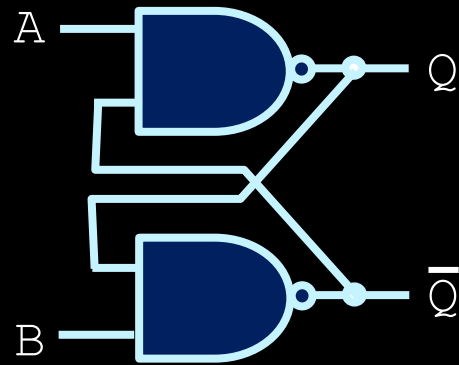
- What makes NAND and NOR feedback circuits different?
 - Unlike the AND and OR gate circuits (which get stuck), the output Q_{T+1} can be changed, based on A .
- However, gates like these that feed back on themselves could enter an unsteady state.

Latch

A feedback circuit with (sort-of) stable behaviour

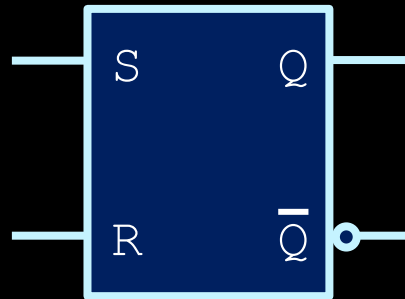
Latches

- If multiple gates of these types are combined, you can get more steady behaviour.



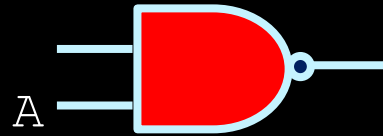
- These circuits are called **latches**.

S'R' Latch



Warm up

This "locking" situation is typically a good starting point when analyzing the behaviour of a sequential circuit.

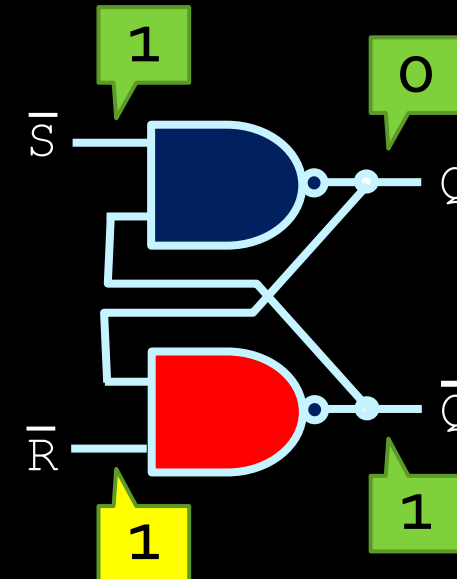
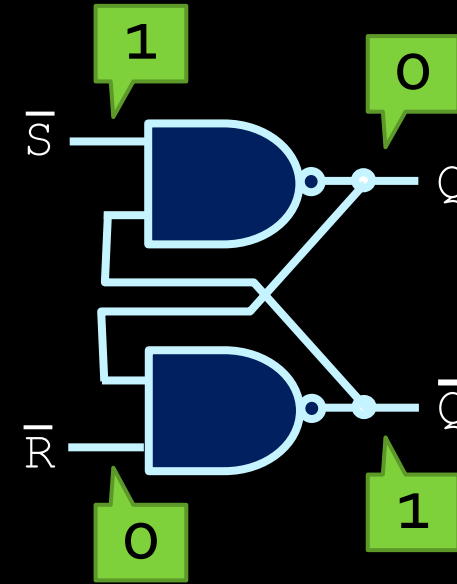


Knowing $A = 0$, what can you say about the output of the NAND gate?

- Must be 1, regardless of the other input
- i.e., a zero input **"locks"** the NAND gate

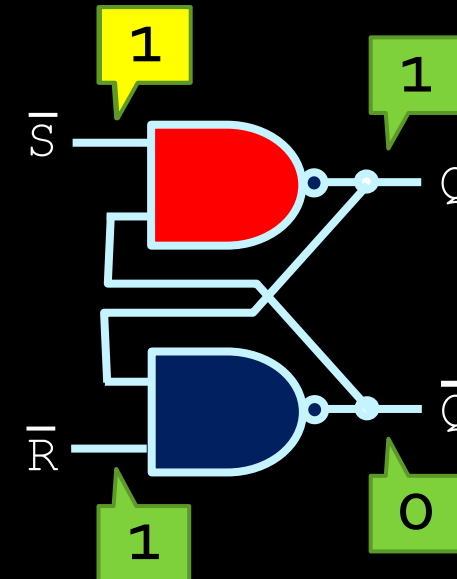
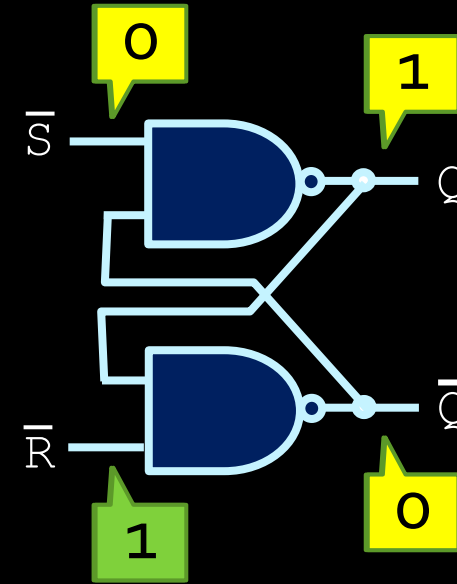
S'R' latch: Case 1

- Let's see what happens when the input values are changed...
 - Assume that S' and R' are set to 1 and 0 to start.
 - The R' input sets the output Q' to 1, which sets the output Q to 0.
 - Setting R' to 1 keeps the output value Q' at 1, which maintains both output values.

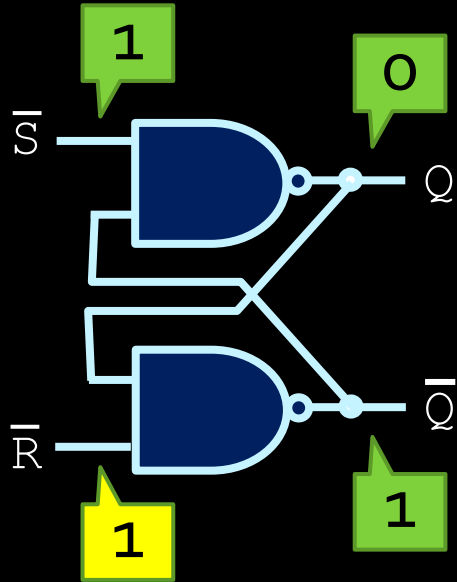


S'R' latch: Case 2

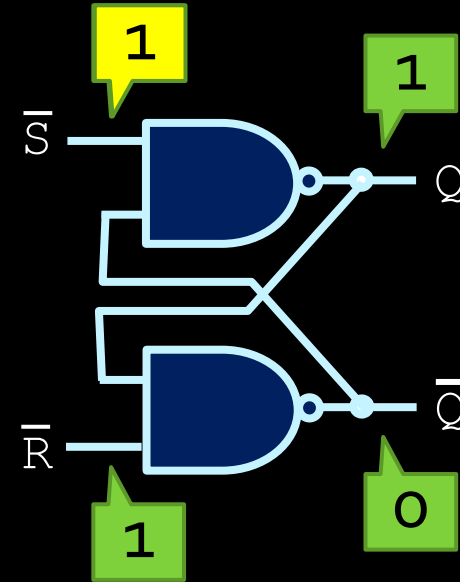
- (continuing from previous)
 - S' and R' start with values of 1, when S' is set to 0.
 - This sets output Q to 1, which sets the output Q' to 0.
 - Setting S' back to 1 keeps the output value Q at 0, which maintains both output values.



Whaaaaaf?!

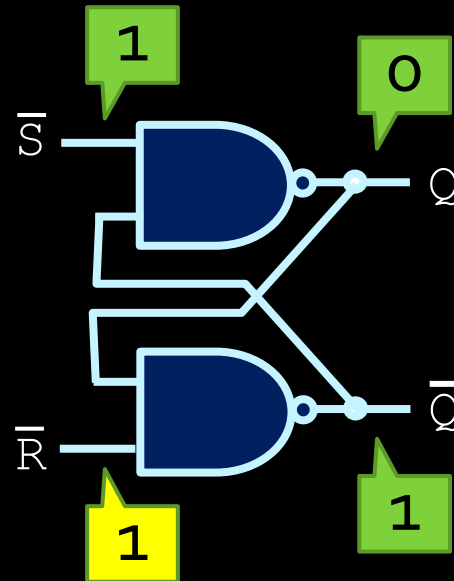
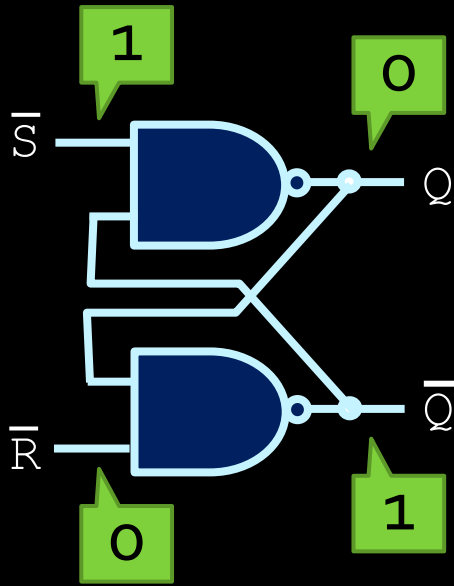


Case 1



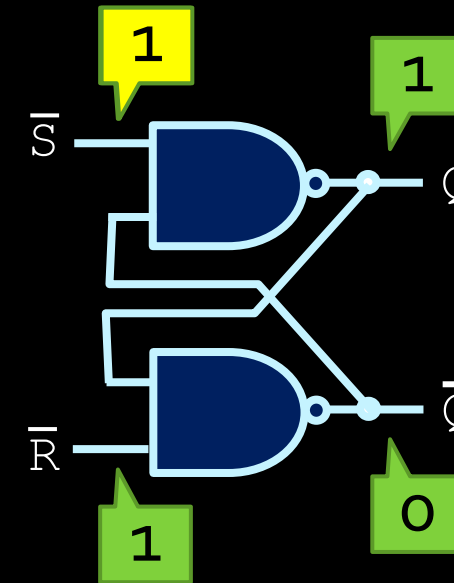
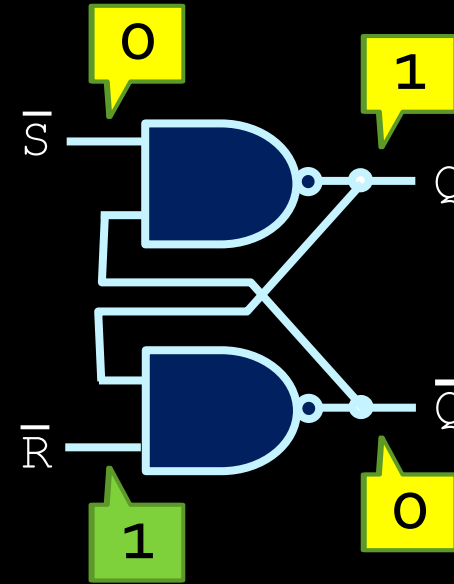
Case 2

Same input, different outputs

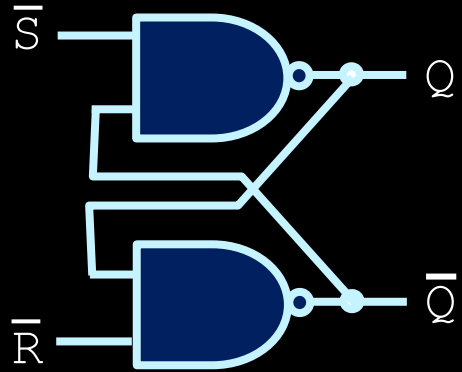


Output of inputs 11 :

maintain the previous output!



S'R' latch



\bar{S}	\bar{R}	Q_T	\bar{Q}_T	Q_{T+1}	\bar{Q}_{T+1}
0	0	X	X	1	1
0	1	X	X	1	0
1	0	X	X	0	1
1	1	0	1	0	1
1	1	1	0	1	0

- S and R are called “set” and “reset” respectively.
- When $S' = 0$, $R' = 1$, Q is 1
- When $S' = 1$, $R' = 0$, Q is 0
- When $S'R' = 11$, same as previous state (01 or 10)
- How about going from 00 to 11
 - Depends on whether it changes from 00 to 01 to 11, or from 00 to 10 to 11 (race condition)
 - **unstable behaviour**

More on instability

- Unstable behaviour occurs when a $S'R'$ latch goes from 00 to 11, or a SR latch goes from 11 to 00.
 - The signals don't change simultaneously, so the outcome depends on which signal changes first.
- Because of the unstable behaviour, 00 is considered a **forbidden state** in NAND-based $S'R'$ latches, and 11 is considered a **forbidden state** in NOR-based SR latches.

More on instability

\bar{S}	\bar{R}	Q_T	\bar{Q}_T	Q_{T+1}	\bar{Q}_{T+1}
0	0	X	X	1	1
0	1	X	X	1	0
1	0	X	X	0	1
1	1	0	1	0	1
1	1	1	0	1	0

Forbidden state

$\bar{S}\bar{R}$ -latch

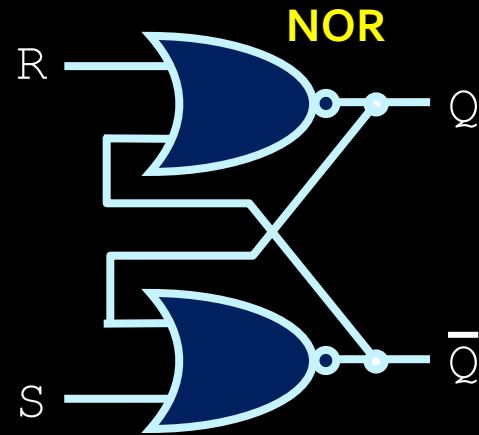
“set” and “reset”
cannot be both true!

S	R	Q_T	\bar{Q}_T	Q_{T+1}	\bar{Q}_{T+1}
0	0	0	1	0	1
0	0	1	0	1	0
0	1	X	X	0	1
1	0	X	X	1	0
1	1	X	X	0	0

SR-latch

Forbidden state

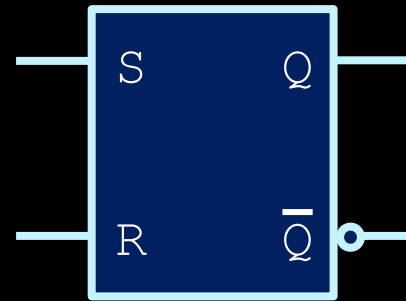
SR latch, with NOR gates



S	R	Q_T	\bar{Q}_T	Q_{T+1}	\bar{Q}_{T+1}
0	0	0	1	0	1
0	0	1	0	1	0
0	1	X	X	0	1
1	0	X	X	1	0
1	1	X	X	0	0

- In this case, S and R are simply “set” and “reset”.
- In this case, the circuit “remembers” previous output when going from 10 or 01 to 00.
- As with $\bar{S}\bar{R}$ latch, unstable behaviour is possible, but this time when inputs go from 11 to 00.

Summary of S'R' / SR Latch behaviour



S: "set"; R: "reset"

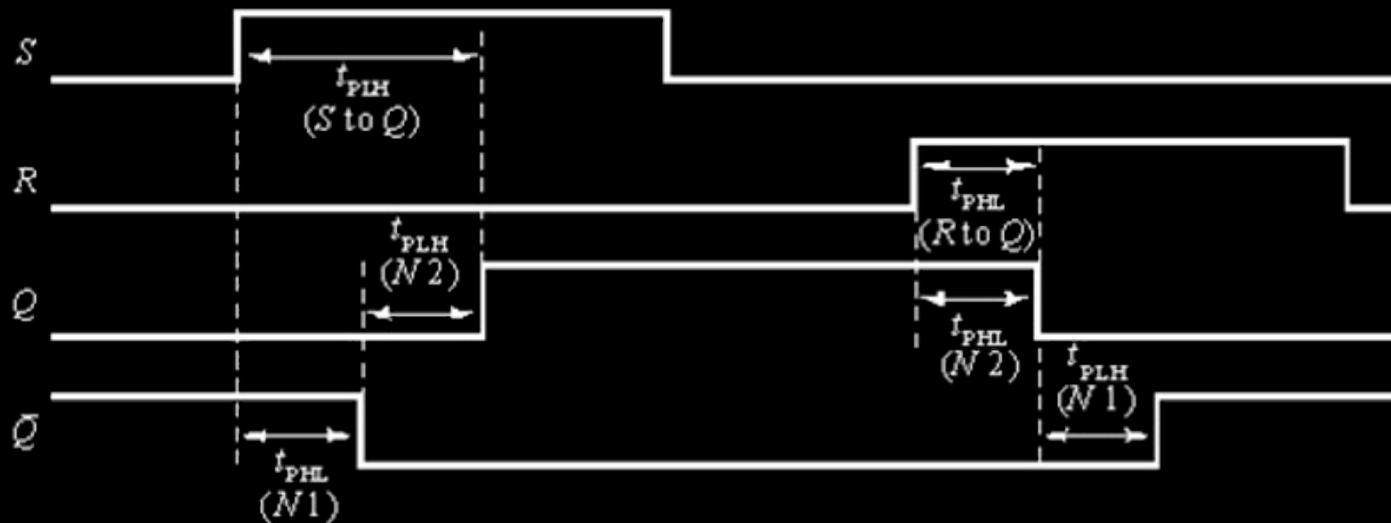
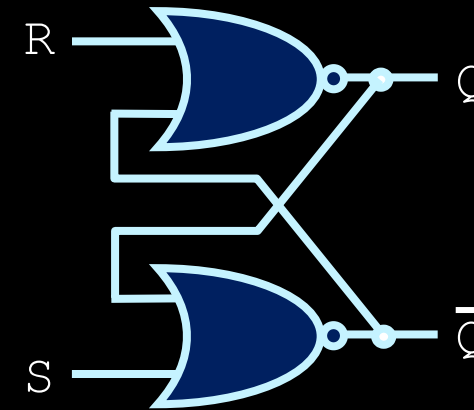
$S = 1, R = 0: Q = 1$

$S = 0, R = 1: Q = 0$

$S = 0, R = 0: \text{"keep"}$

Be aware of delays

- Important to note that, in reality, the output signals don't change instantaneously, but with a certain delay.



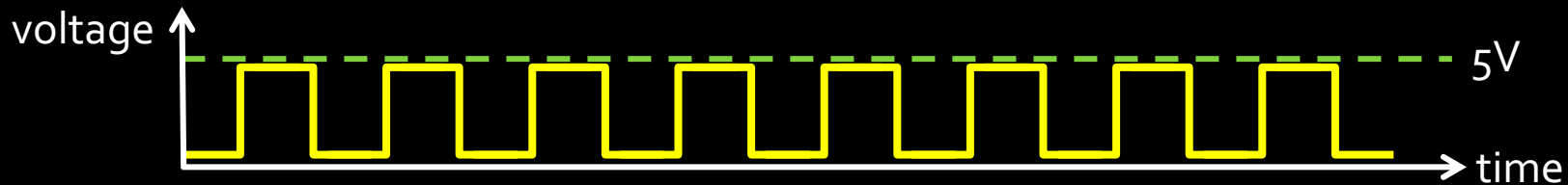
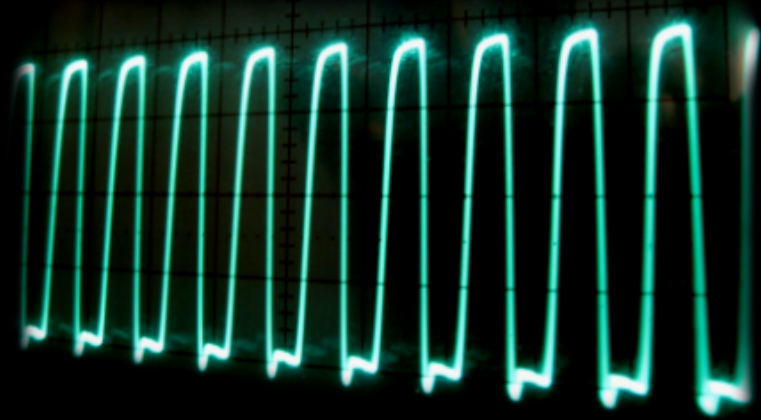
Reading from latches

- Now we have circuit units that can **store** high or low values.
- How do we distinguish
 - “5 highs in a row”
 - “10 highs in a row”
- We want to **sample** the signal with certain frequency.
- Need to use some sort of timing signal, to let the circuit know when the output may be sampled.

→ **clock signals.**

Clock signals

- “Clocks” are a regular pulse signal, where the high value indicates that the output of the latch may be sampled.
- Usually drawn as:

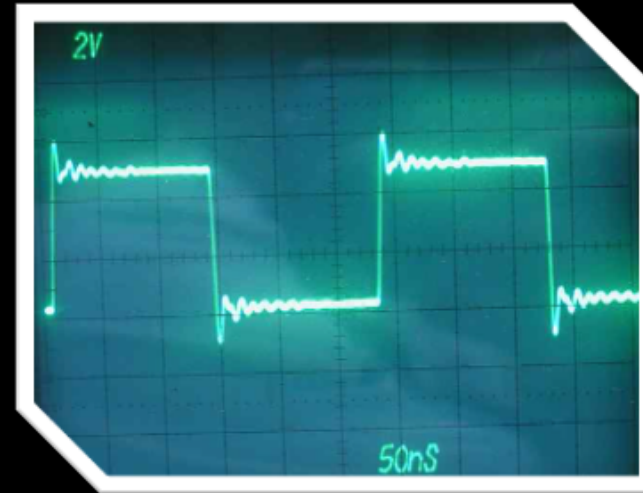


- But looks more like (frequency is usually high):



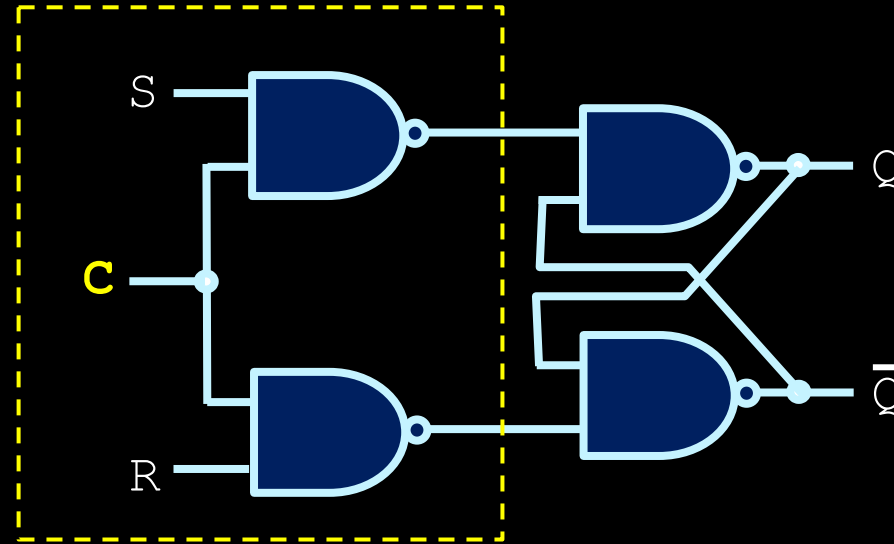
Signal restrictions

- What's the limit to how fast the latch circuit can be sampled?
- Determined by:
 - latency time of transistors
 - Setup and hold time
 - setup time for clock signal
 - Jitter
 - Gibbs phenomenon
- **Frequency** = how many pulses occur per second, measured in Hertz (or Hz).
- What Intel and AMD try to increase every year.



Clocked SR Latch

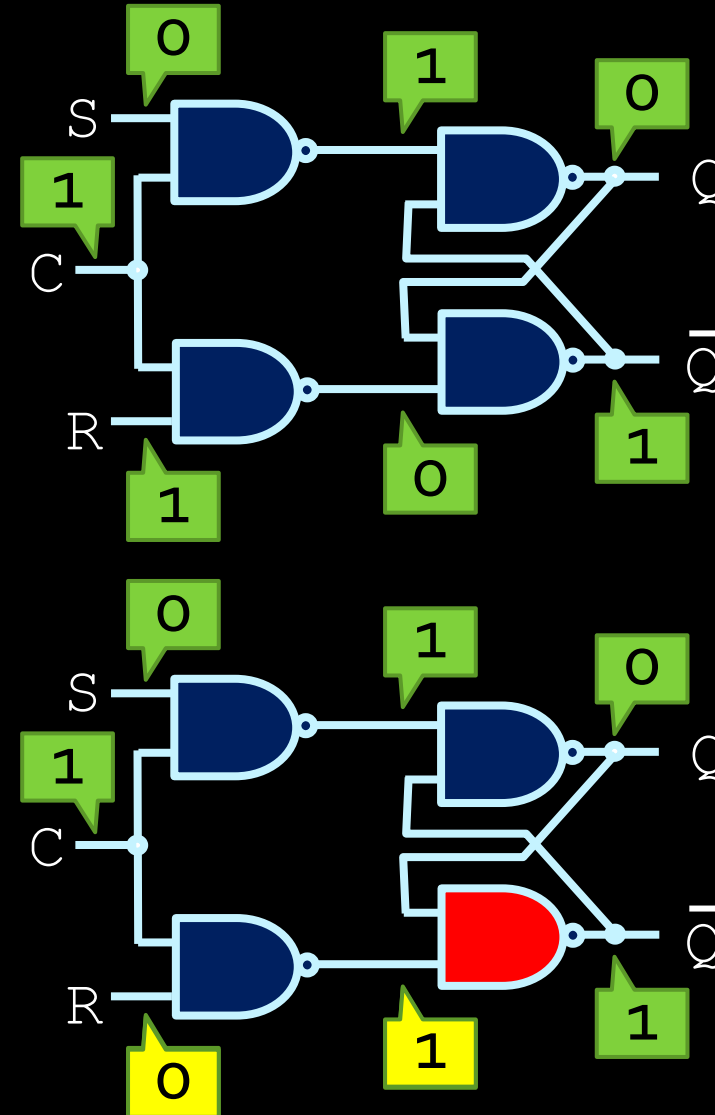
Clocked SR latch



- By adding another layer of NAND gates to the $\bar{S}\bar{R}$ latch, we end up with a **clocked SR latch** circuit.
- The clock is often connected to a pulse signal that alternates regularly between 0 and 1.

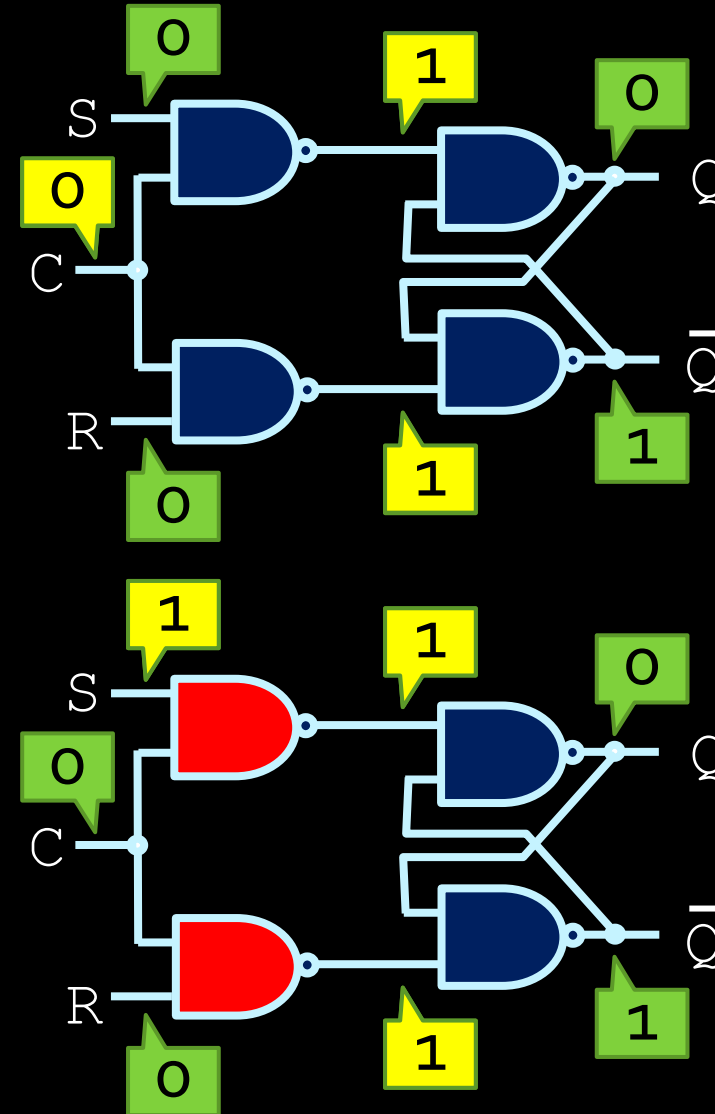
Clocked SR latch behaviour

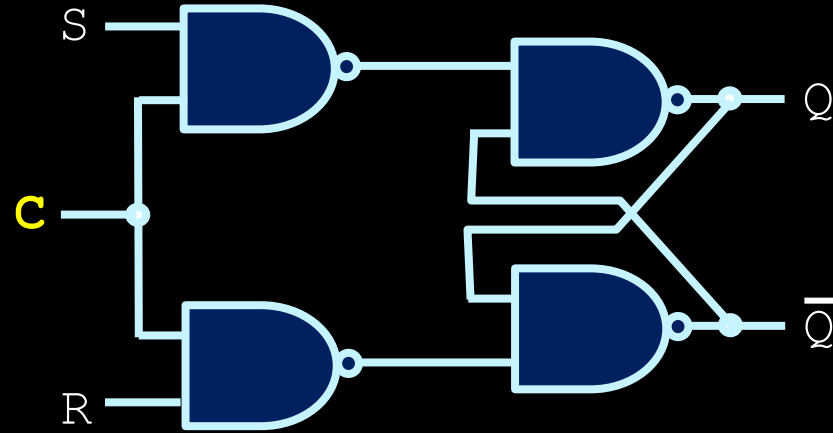
- Same behaviour as SR latch, but with timing:
 - Start off with $S=0$ and $R=1$, like earlier example.
 - If **clock is high**, the first NAND gates invert those values, which get inverted again in the output.
 - Setting both inputs to 0 maintains the output values.



Clocked SR latch behaviour

- Continued from previous:
 - Now set the **clock low**.
 - Even if the inputs change, the low clock input prevents the change from reaching the second stage of NAND gates.
 - Result: the clock needs to be high in order for the inputs to have any effect.

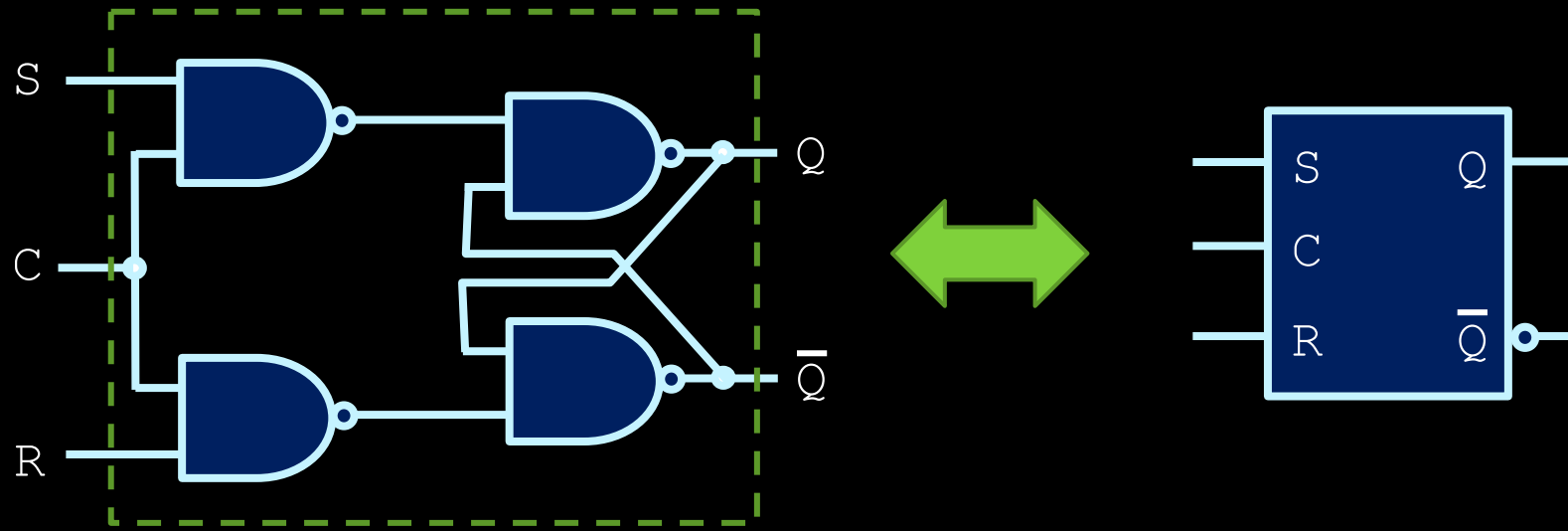




When clock is high, behave like a SR latch.

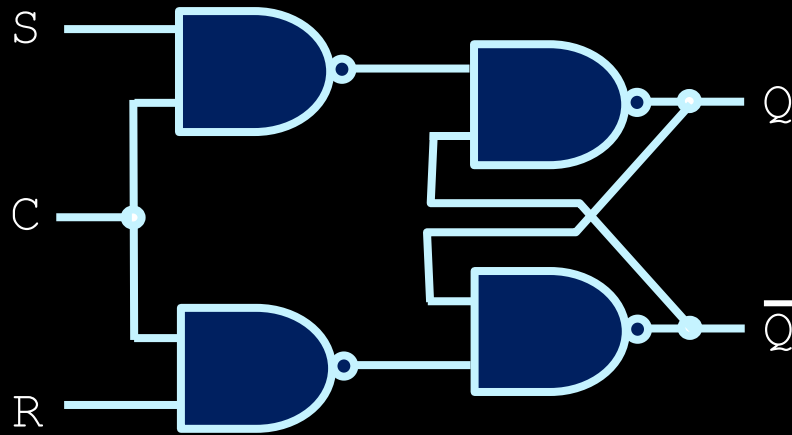
When clock is low, S and R are blocked and there is no way to change the output.

Clocked SR latch



- This is the typical symbol for a clocked SR latch.
- This only allows the S and R signals to affect the circuit when the clock input (C) is high.
- Note: the small NOT circle after the \bar{Q} output is simply the notation to use to denote the inverted output value. It's not an extra NOT gate.

Clocked SR latch behaviour



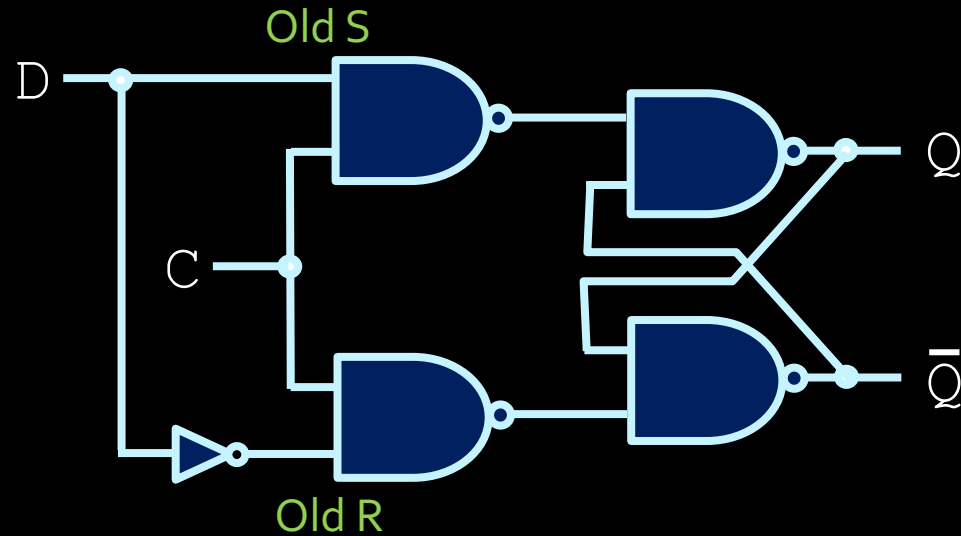
Q_T	S	R	Q_{T+1}	Result
0	0	0	0	no change
0	0	1	0	reset
0	1	0	1	set
0	1	1	?	???
1	0	0	1	no change
1	0	1	0	reset
1	1	0	1	set
1	1	1	?	???

- Assuming the clock is 1, we still have a problem when S and R are both 1, since it is the **forbidden state**.
 - Better design: prevent S and R from both going high.

D latch

prevent S and R from both going high

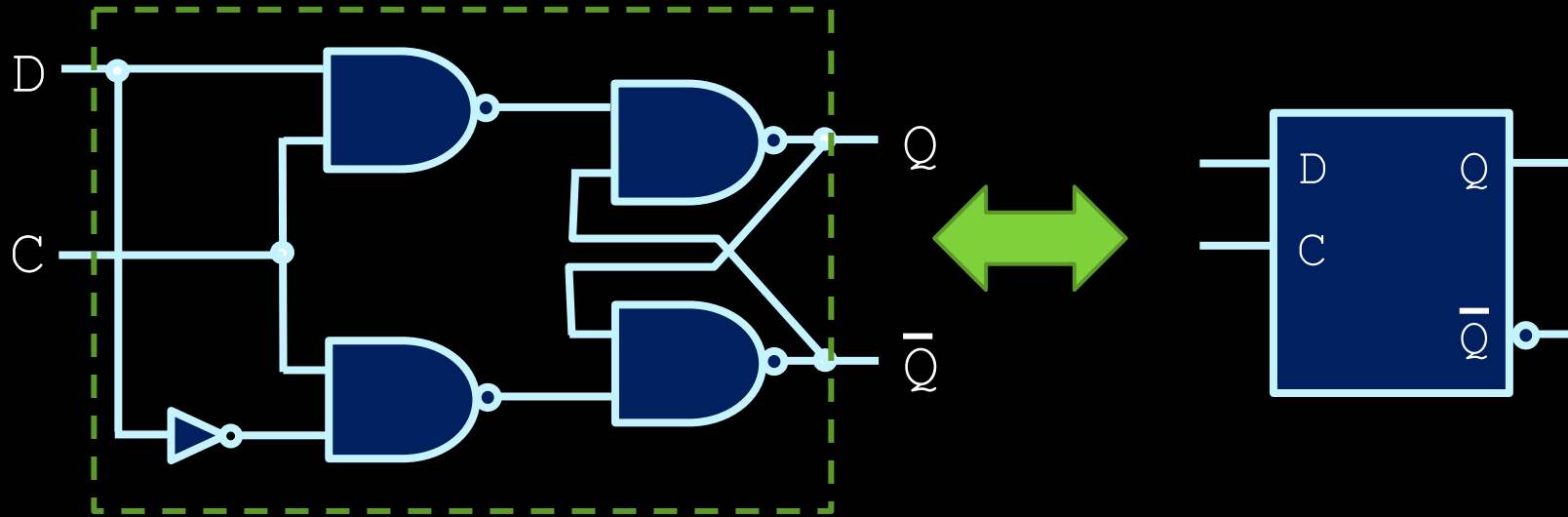
D latch



Q_T	D	Q_{T+1}
0	0	0
0	1	1
1	0	0
1	1	1

- By making the inputs to R and S dependent on a single signal D, you avoid the indeterminate state problem.
- The value of D now sets output Q low or high.

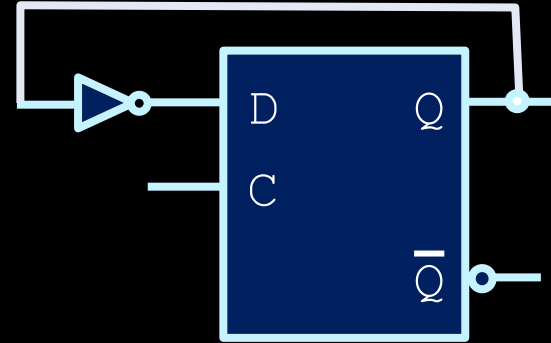
D latch



- This design is good, but still has problems.
 - i.e. **timing issues**.

Latch timing issues

- Consider the circuit on the right:
- When the clock signal is high, the output looks like the waveform below:

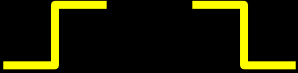


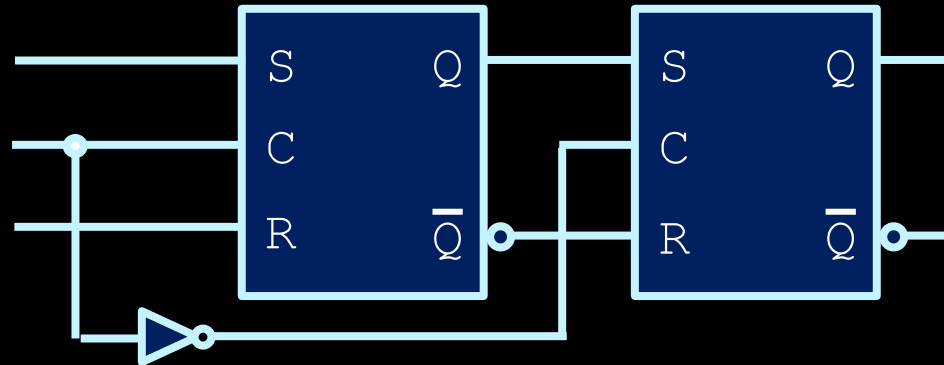
- Output keeps toggling back and forth.



Would be nice if Q changes only **once** within **one clock cycle**

Latch timing issues

- Preferable behaviour:
 - Have output change only once when the clock pulse changes. 
 - Solution: create disconnect between circuit output and circuit input, to prevent unwanted feedback and changes to output.



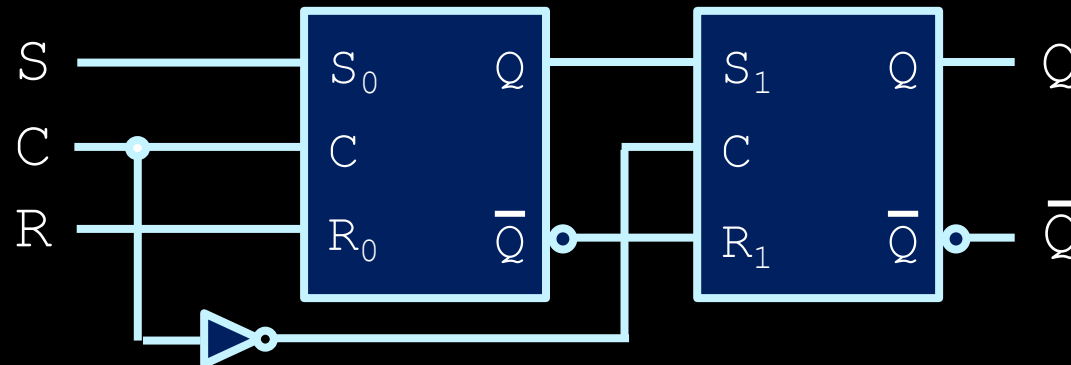
Only one latch is active at one time, so in order for a change to propagate from input to output, you need to wait at least for one turn (a clock cycle) of both being active.

Flip Flops



SR master-slave flip-flop

- A **flip-flop** is a latched circuit whose output is triggered with the **rising edge** or **falling edge** of a clock pulse.
- Example: The SR master-slave flip-flop



Demo: (Virtual) Human flesh flip-flop




1. Need two volunteers, A and B
2. Clock signal: "flip" and "flop"
3. Person A open eyes when hearing "flip", and close eyes when hearing "flop"
4. Person B does the opposite
5. Person A gestures the number (Latch 1 output) upon seeing it from Larry's input.
6. When Person B sees the number gestured by A, shout it out. (final output)

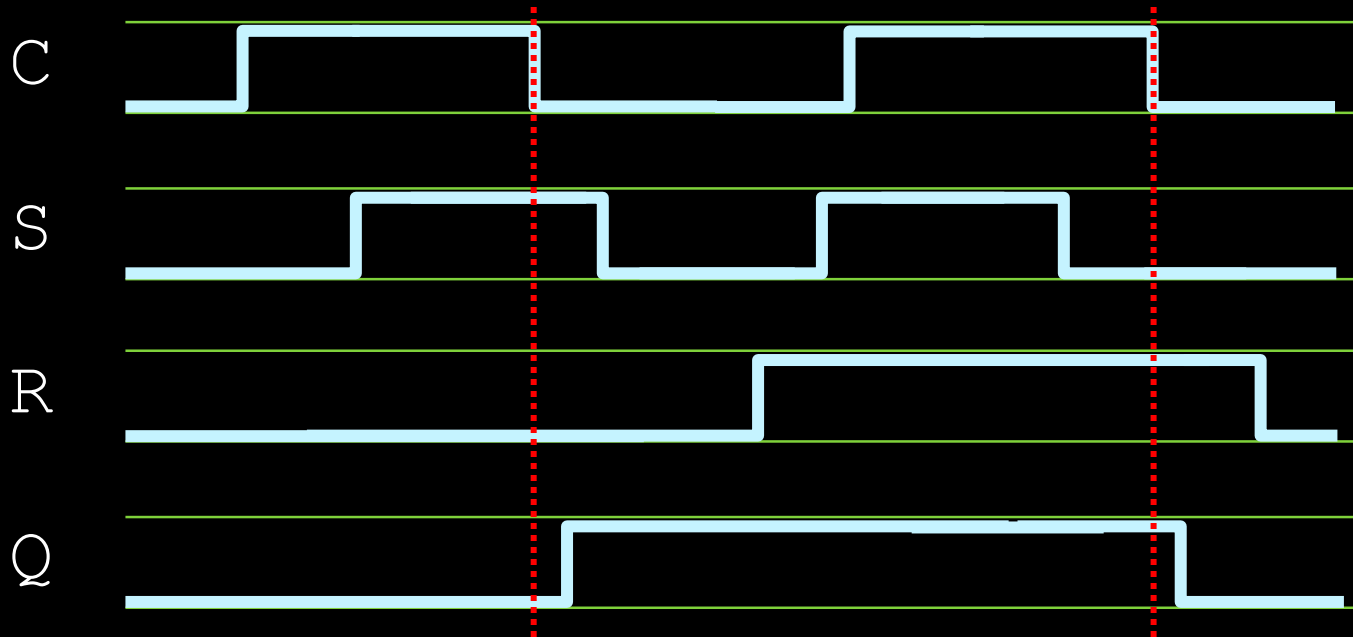
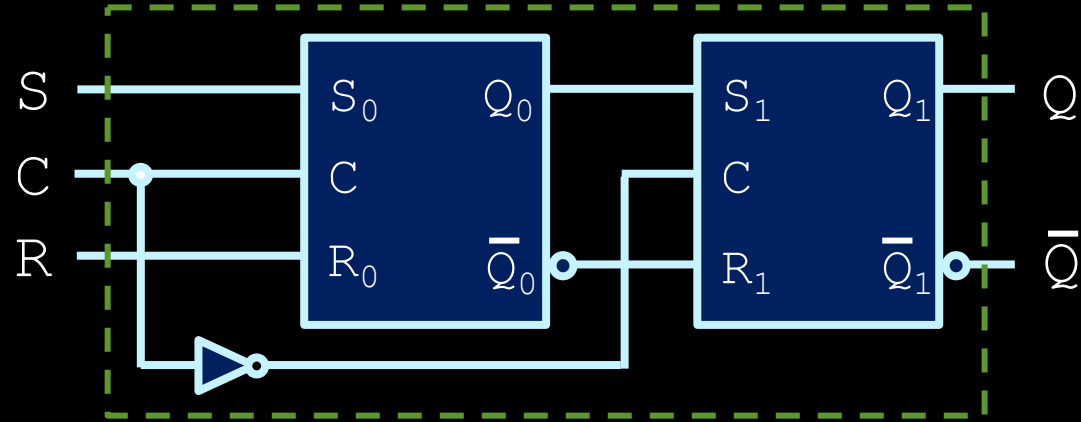
Demo: Human flesh flip-flop



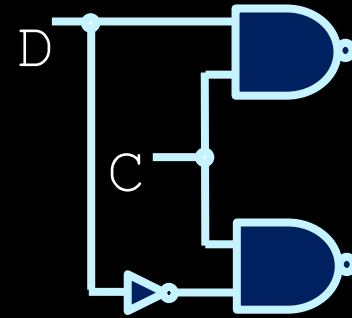
Summary:

1. For input to propagate to output, it takes each of the latches to be active once.
2. Output can only change upon “flop”, which is basically the falling edge of the clock signal 
3. At most one change per clock cycle

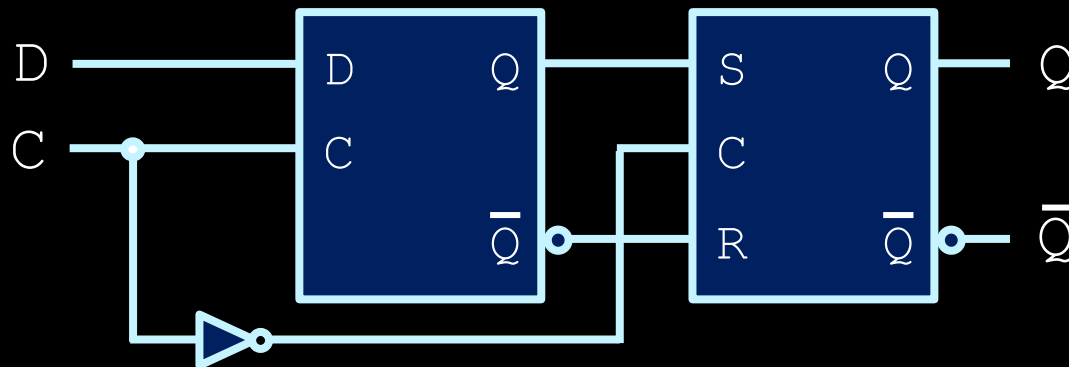
Change of Q triggered by falling clock edges



Edge-triggered D flip-flop

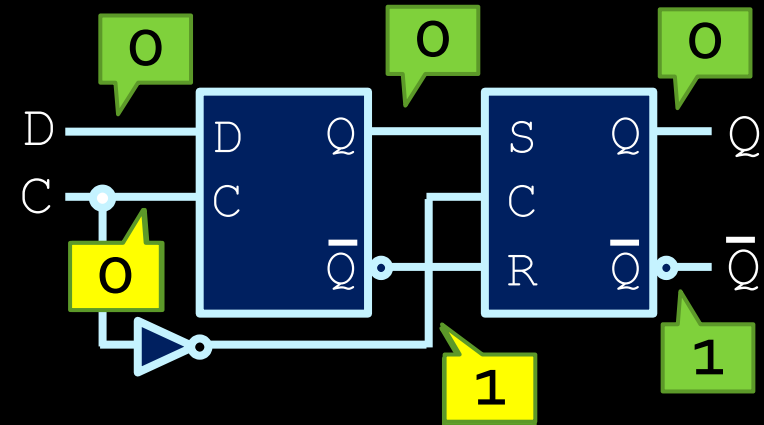
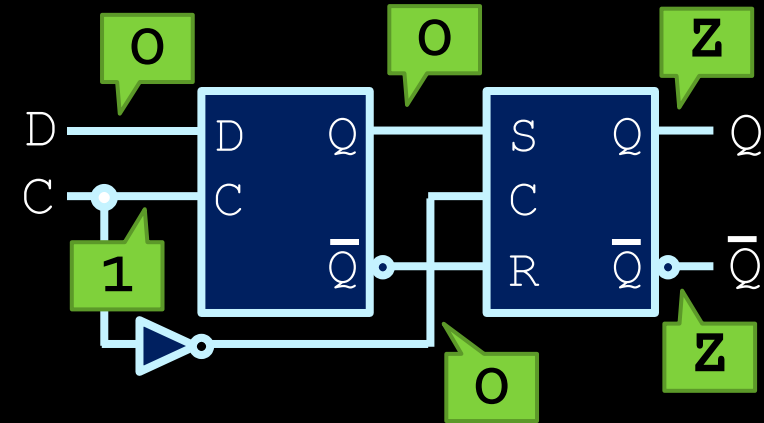


- SR flip-flops still have issues of unstable behavior (**forbidden state**)
- Solution: **D flip-flop**
 - Connect D latch to the input of a SR latch.
 - **Negative-edge triggered** flip-flop (like the SR)



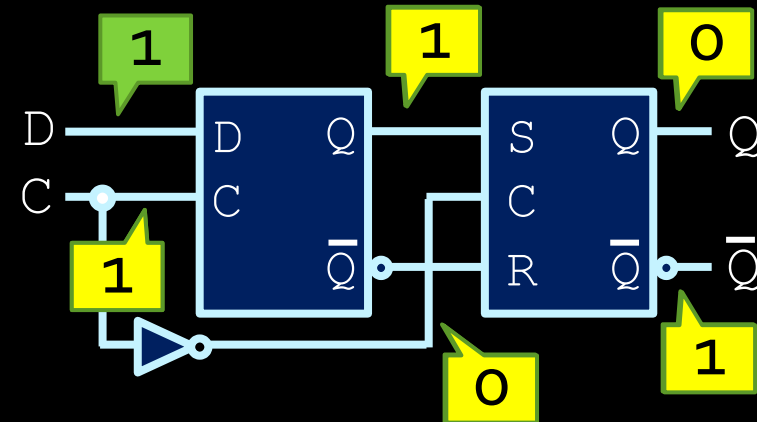
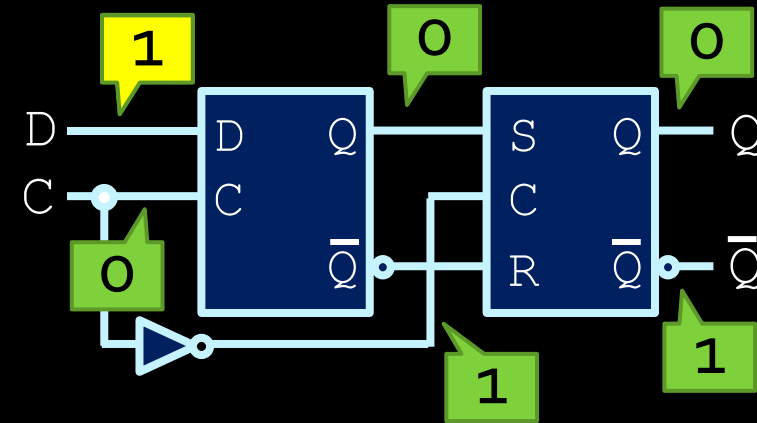
Flip-flop behaviour

- Observe the behaviour:
 - If the clock signal is high, the input to the first flip-flop is sent out to the second.
 - The second flip-flop doesn't do anything until the clock signal goes down again.
 - When it clock goes from high to low, the first flip-flop stops transmitting a signal, and the second one starts.



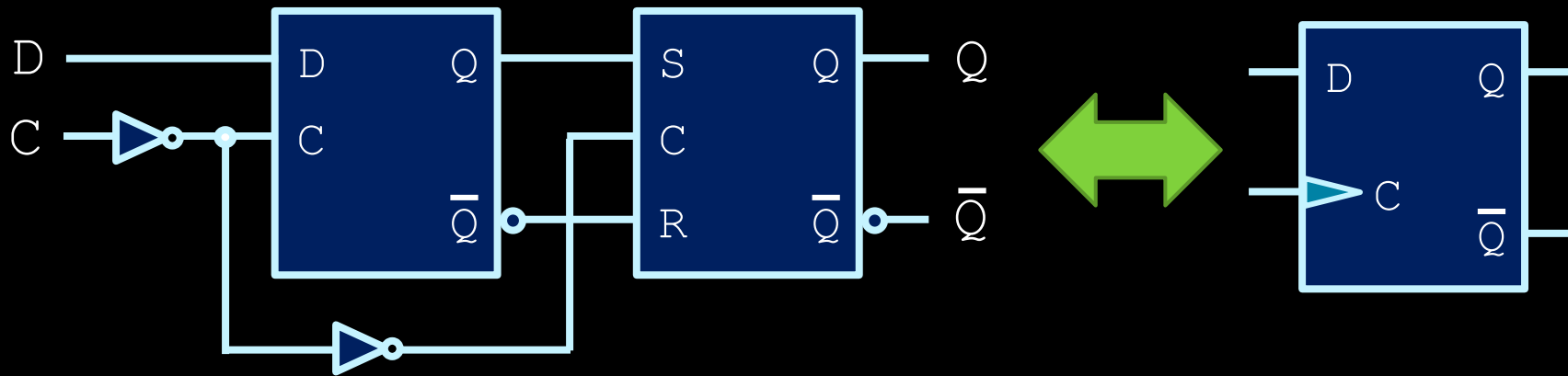
Flip-flop behaviour

- Continued from previous:
 - If the input to D changes, the change isn't transmitted to the second flip-flop until the clock goes high again.
 - Once the clock goes high, the first flip-flop starts transmitting at the same time as the second flip-flop stops.



Edge-triggered flip-flop

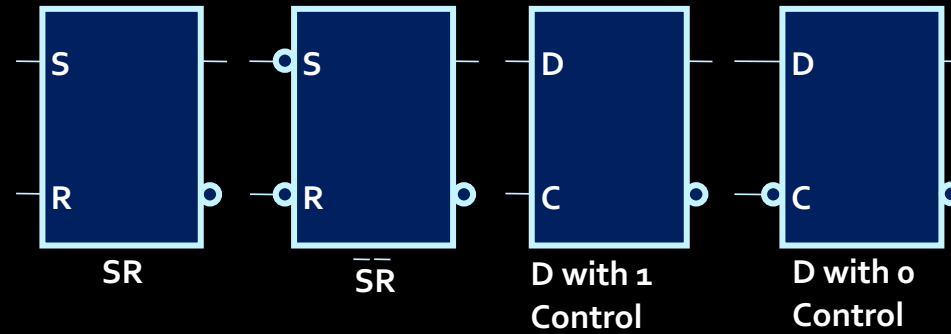
- Alternative: **positive-edge triggered** flip-flops



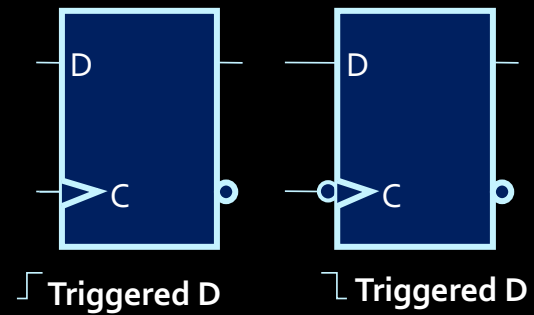
- These are the most commonly-used flip-flop circuits (and our choice for the course).

Notation

- Latches

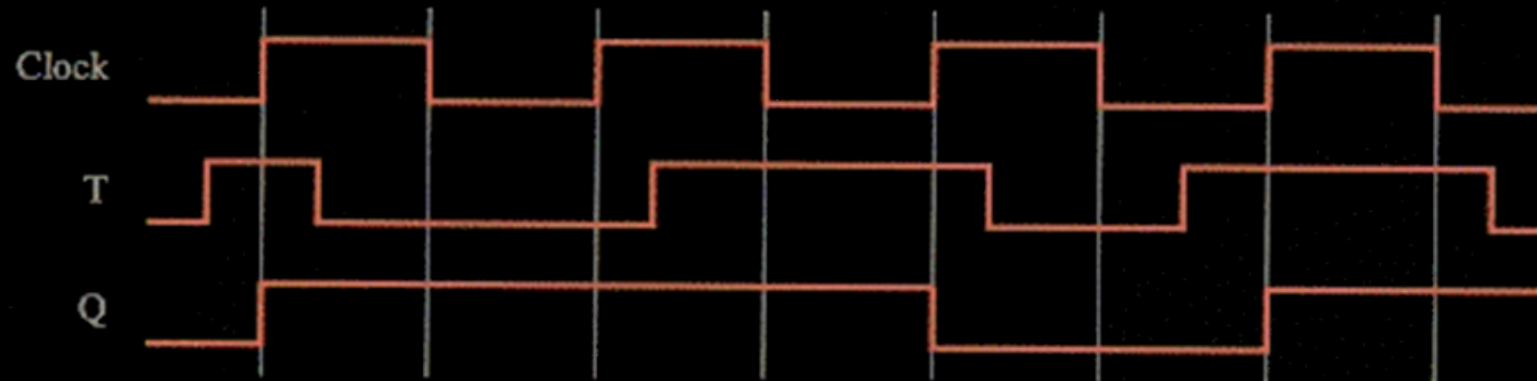
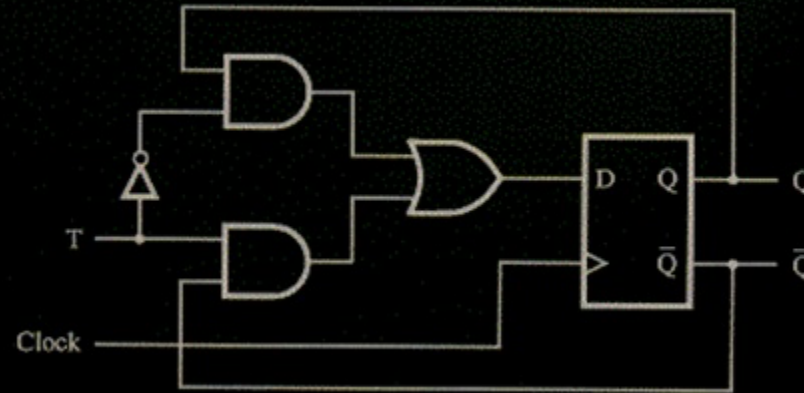


- Edge-triggered flip-flops



Other Flip-Flops

- The **T flip-flop**:
 - Like the D flip-flop, except that it toggles its value whenever the input to T is high.

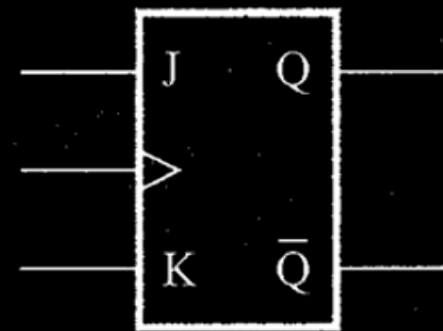
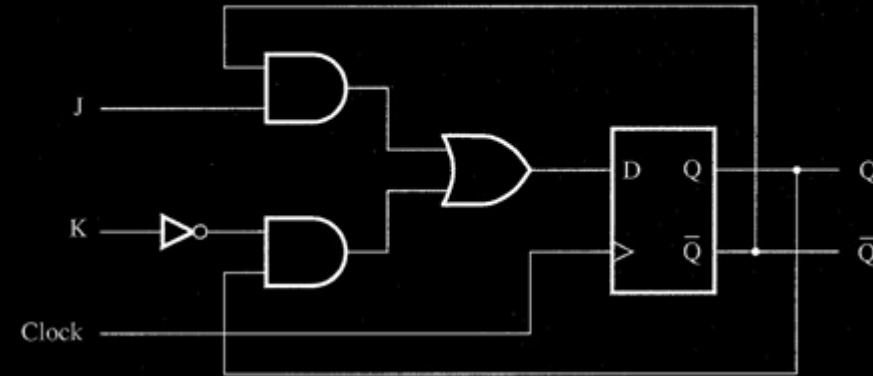


Other Flip-Flops

- The **JK Flip-Flop**:

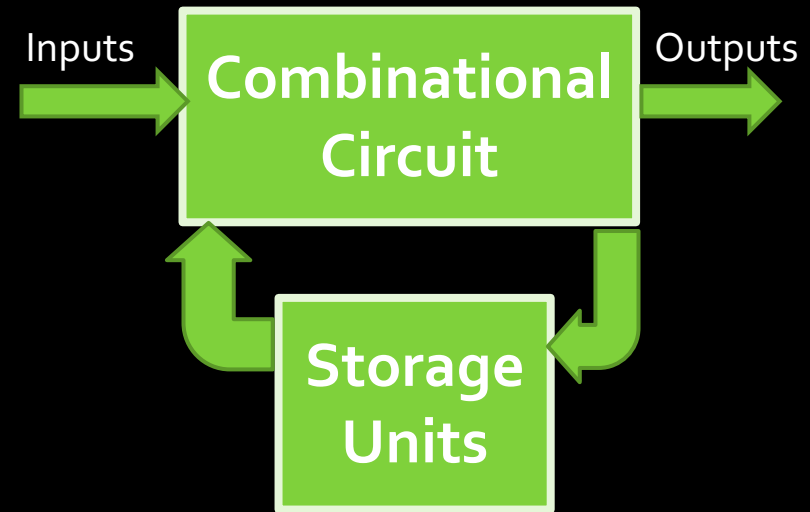
- Takes advantage of all combinations of two inputs (J & K) to produce four different behaviours:

- if J and K are 0, maintain output.
- if J is 0 and K is 1, set output to 0.
- if J is 1 and K is 0, set output to 1.
- if J and K are 1, toggle output value.



Sequential circuit design

- Similar to creating combinational circuits, with extra considerations:
 - The flip-flops now provide extra inputs to the circuit
 - Extra circuitry needs to be designed for the flip-flop inputs.
 - ...which is next week's lecture 😊



Today we learned

- Sequential circuits – circuits with memory
- Latch
- Flip-flop

Next week

- Registers, Counters
- Finite State Machines
- Sequential circuit design