CSC258 Winter 2016
Lecture 4

Learning

- Labs
  - You're ready to start preparing for the lab when the handout is posted, no need to wait until Monday's lecture.
  - You will be kept (reasonably) busy with all the pre-lab and in-lab works throughout the term. Need to be persistent to keep on track.
  - Labs are very important learning components for this course. The knowledge involved in the labs will be the part you learn the best in this course.
  - In exams, if certain questions require deeper understanding than others; they are more likely to be related to what you did in the labs.
  - If you have trouble preparing for labs, come to office hours.

Learning

Quizzes

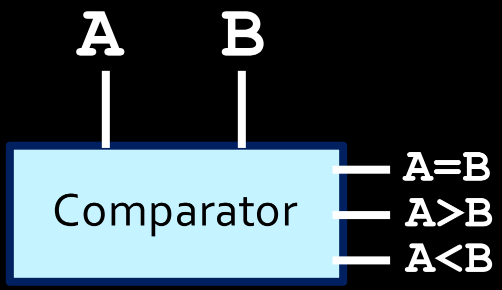
- It also serves as a “weekly reality check” of whether you have really understood last week’s content.
- It may be for bonus marks, but it’s content is not “extra”.
- Solutions (with explanations) are included when slides are posted. Make sure you know the right answers; if have question, ask on Piazza or come to office hours.
- Cheating in Quiz?!
  - Caught once, no bonus for the term.

Comparators (leftover from last week)

Comparators

- A circuit that takes in two input vectors, and determines if the first is greater than, less than or equal to the second.
- How does one make that in a circuit?

Basic Comparators

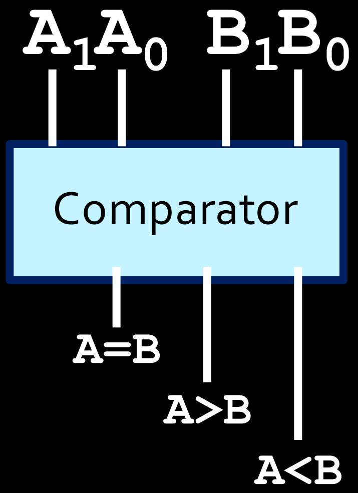


- Consider two binary numbers A and B, where A and B are one bit long.
- The circuits for this would be:
  - A==B: $A \cdot B + \overline{A} \cdot \overline{B}$
  - A>B: $A \cdot \overline{B}$
  - A<B: $\overline{A} \cdot B$

A	B
0	0
0	1
1	0
1	1

Basic Comparators

- What if A and B are two bits long?
- The terms for this circuit for have to expand to reflect the second signal.
- For example:

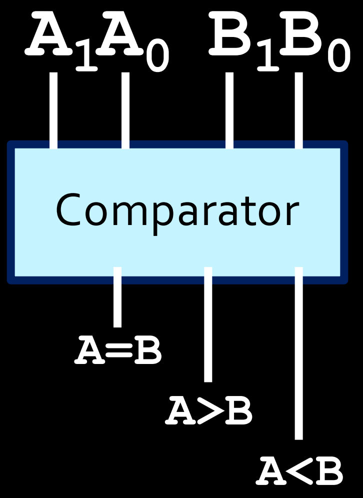


  - A==B: $\boxed{(A_1 \cdot B_1 + \overline{A_1} \cdot \overline{B_1}) \cdot (A_0 \cdot B_0 + \overline{A_0} \cdot \overline{B_0})}$

Make sure that the values of bit 1 are the same

Make sure that the values of bit 0 are the same

Basic Comparators



- What about checking if A is greater or less than B?
  - A>B: $\boxed{A_1 \cdot \overline{B_1}} + \boxed{(A_1 \cdot B_1 + \overline{A_1} \cdot \overline{B_1})} \cdot \boxed{(A_0 \cdot \overline{B_0})}$
  - A<B: $\boxed{\overline{A_1} \cdot B_1} + \boxed{(A_1 \cdot B_1 + \overline{A_1} \cdot \overline{B_1})} \cdot \boxed{(\overline{A_0} \cdot B_0)}$

Check if first bit satisfies condition

If not, check that the first bits are equal…

…and then do the 1-bit comparison

A > B if and only if A1 > B1 or (A1 = B1 and A0 > B0)

Comparing large numbers

- The circuit complexity of comparators increases quickly as the input size increases.
- For comparing large number, it may make more sense to just use a subtractor.
  - Subtract and then check the sign bit.

New Topic:
Sequential Circuit

Something we can't do yet

- How does the Tickle Me Elmo work?

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

Same input, different outputs.

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…

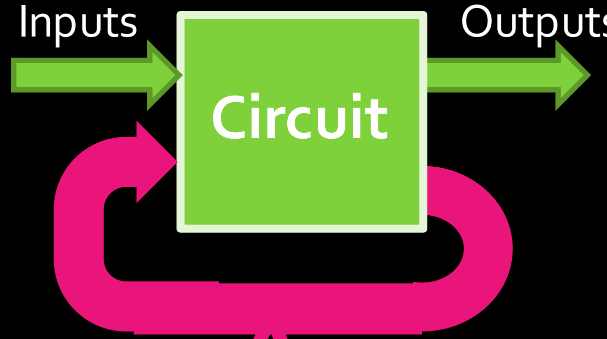
- Combination circuits does NOT have memory
  - Everything is based on current inputu value
- Sequential Circuits have **memory**.
  - They **remember** something from the past (the previous state of the circuit), and its output depends on that
  - It is why computers have memory (e.g., RAM)

Sequential circuits

- This creates circuits whose internal state can change over time, where the same input values can result in different outputs.

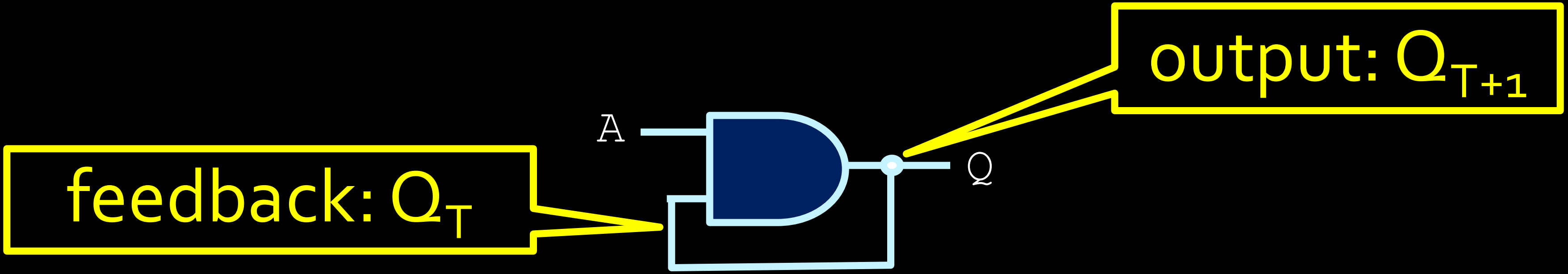
- Why would we need circuits like this?
  - Memory values
    - To remember stuff
  - Reacting to changing inputs
    - "Output X = 1 when input A changes"

How can a circuit have memory?



The magic: feedback!

Truth table of a sequential circuit

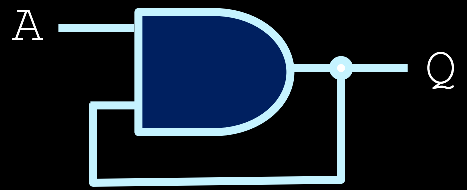


A	Q_T	$Q_{T+1} = A \cdot Q_T$
0	0	0
0	1	0
1	0	0
1	1	1

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

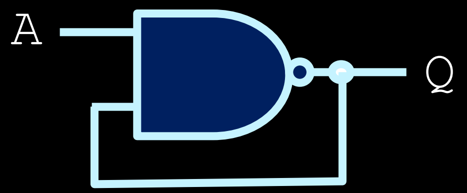


A	Q_T	$Q_{T+1} = A \cdot Q_T$
0	0	0
0	1	0
1	0	0
1	1	1

(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



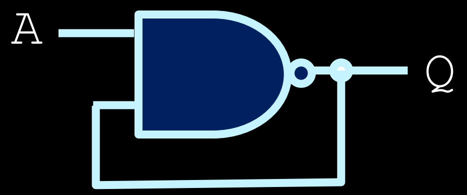
A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

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

Oscillates between each other

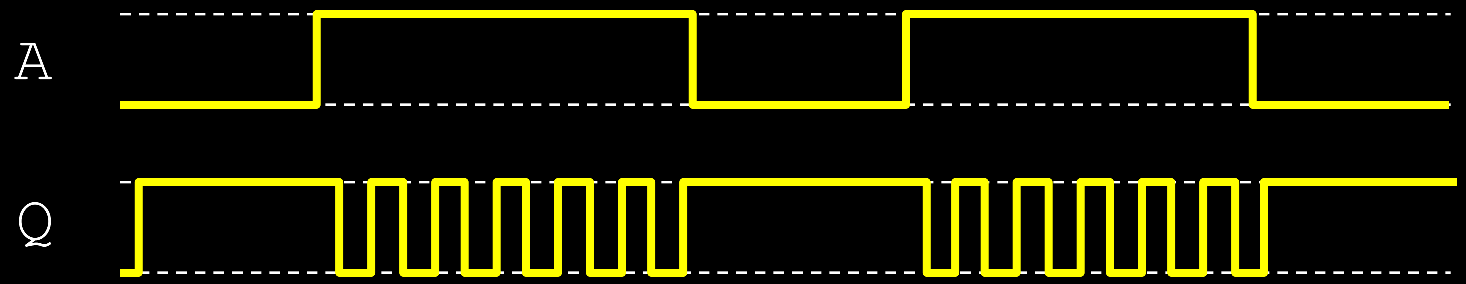
Output Q_{T+1} can be changed by changing A

NAND waveform behaviour



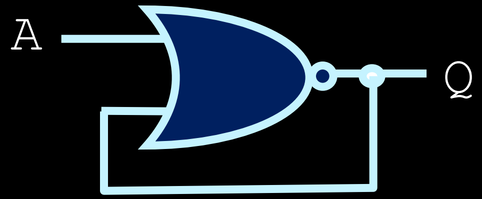
A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

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

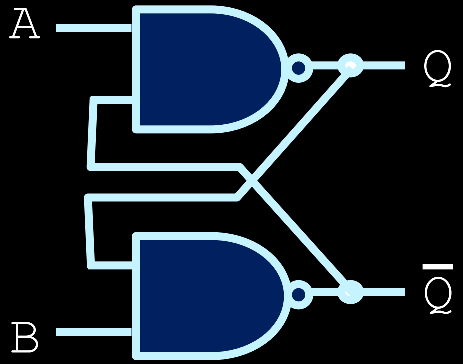
- What makes NAND and NOR feedback circuits different?
  - Unlike the AND and OR gate circuits (which get stuck), the output Q_{T+1} can be changed, based on A.
- However, gates like these that feed back on themselves could enter an unsteady state.

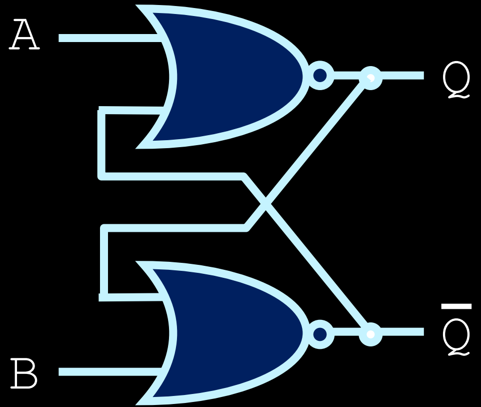
Latch

A feedback circuit with (sort-of) stable behaviour

Latches

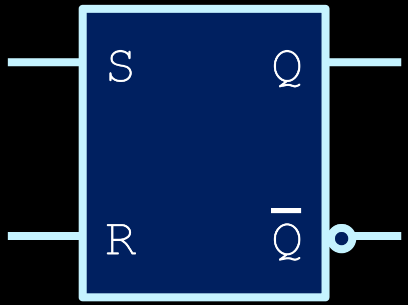
- If multiple gates of these types are combined, you can get more steady behaviour.



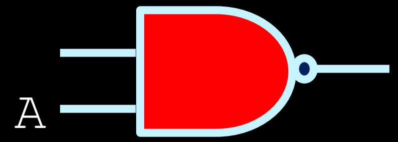


- These circuits are called latches.

S’R’ Latch



Warm up

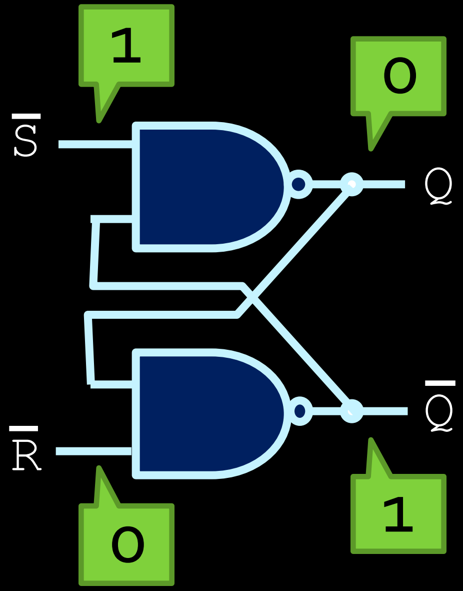


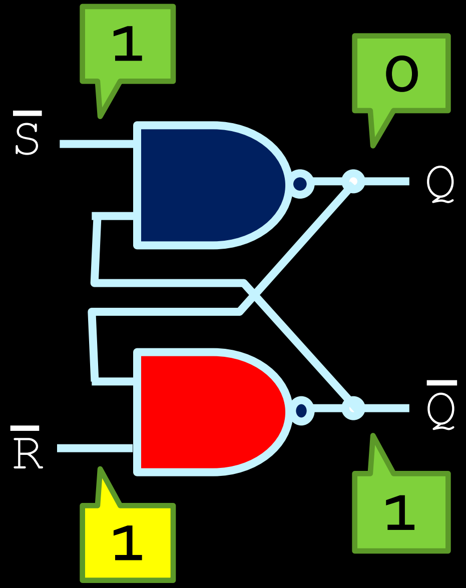
Knowing A = 0, what can you say about the output of the NAND gate?

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

S'R' latch: Case 1

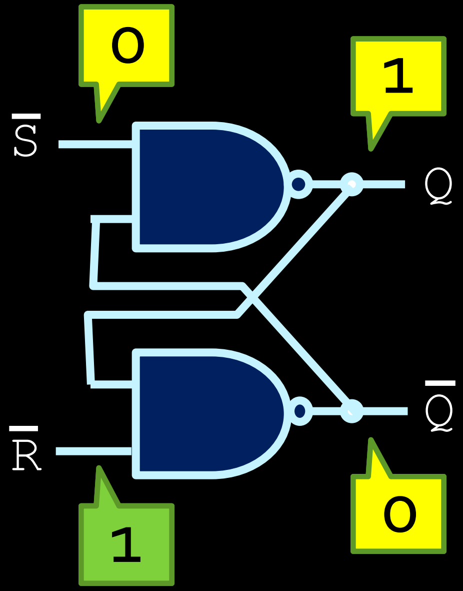
- Let's see what happens when the input values are changed…
  - Assume that `S'` and `R'` are set to `1` and `0` to start.
  - The `R'` input sets the output `Q'` to `1`, which sets the output `Q` to `0`.
  - Setting `R'` to `1` keeps the output value `Q'` at `1`, which maintains both output values.

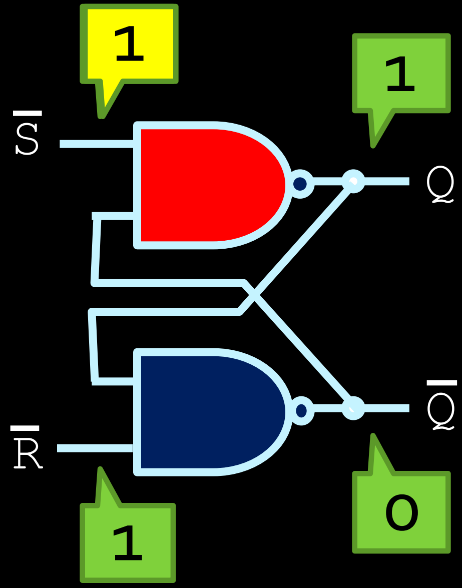


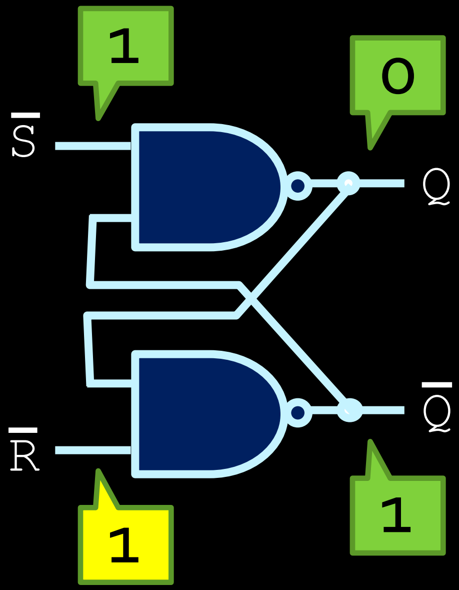


S'R' latch: Case 2

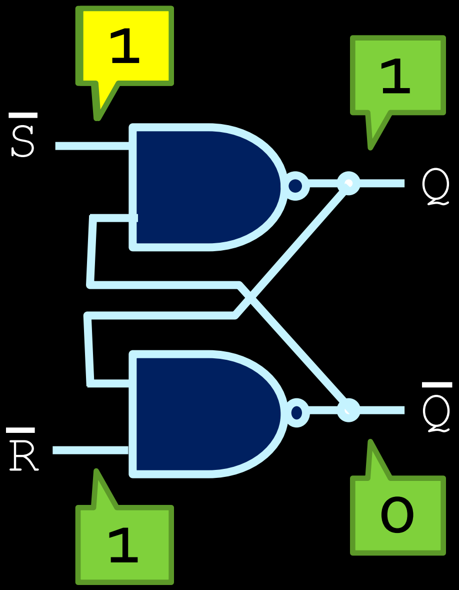
- (continuing from previous)
  - `S'` and `R'` start with values of `1`, when `S'` is set to `0`.
  - This sets output `Q` to `1`, which sets the output `Q'` to `0`.
  - Setting `S'` back to `1` keeps the output value `Q` at `0`, which maintains both output values.





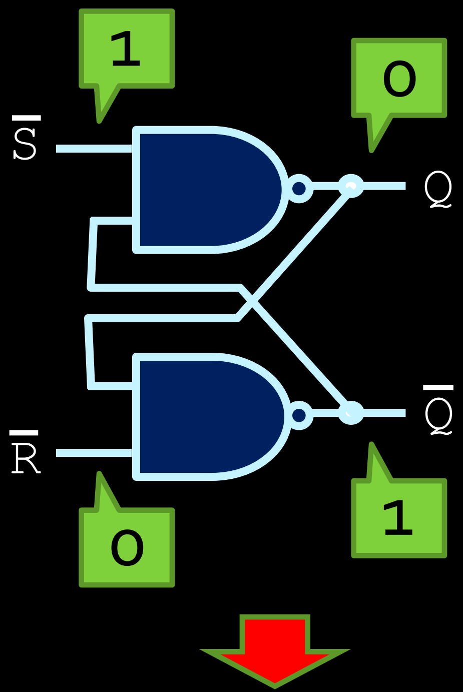


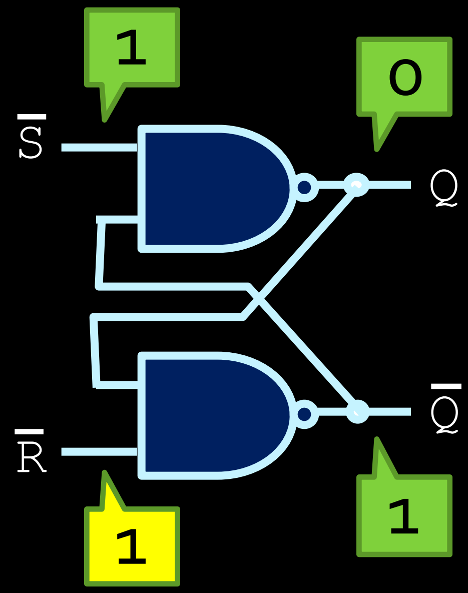
Case 1



Case 2

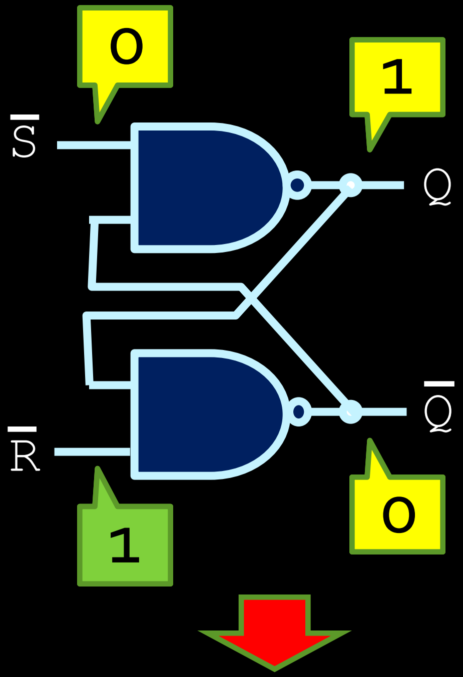
Same input, different outputs

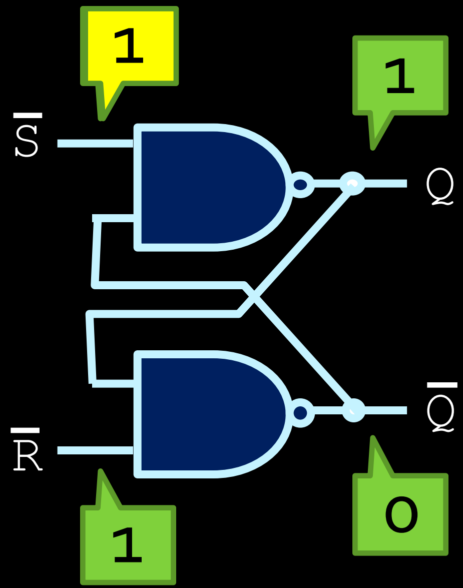




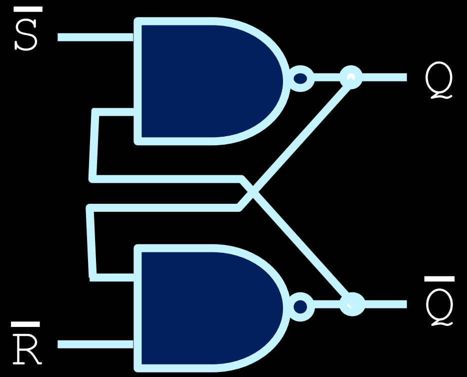
Output of inputs 11 :

maintain the previous output!





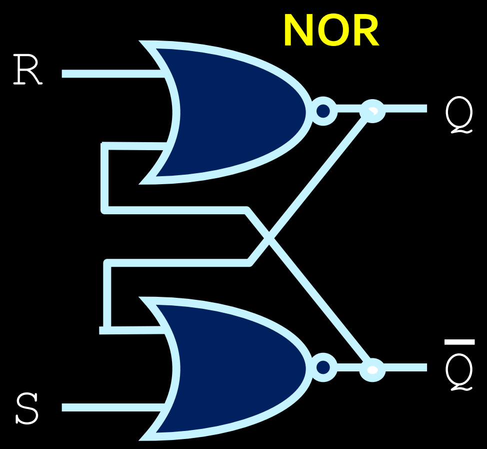
S'R' latch



$\overline{S}$	$\overline{R}$	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$
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

- $\overline{S}$ and $\overline{R}$ 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 oo to 11
  - Depends on whether it changes from 00 to 01 to 11, or from 00 to 10 to 11
  - unstable behaviour

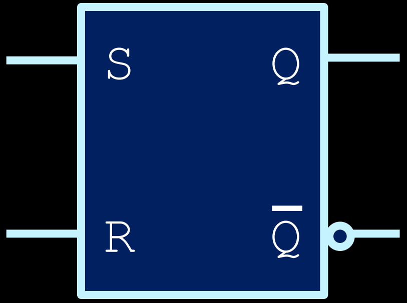
SR latch, with NOR gates



S	R	Q_T	Q_T	Q_{T+1}	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 "set" and "reset".
- In this case, the circuit "remembers" previous output when going from `10` or `01` to `00`.
- As with $\overline{\mathtt{S}}\overline{\mathtt{R}}$ latch, unstable behaviour is possible, but this time when inputs go from `11` to `00`.

Summary of S’R’ / SR Latch behaviour

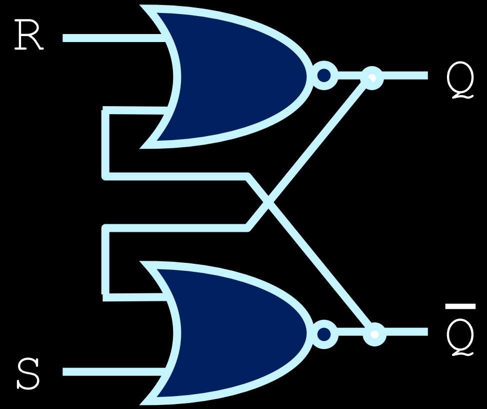


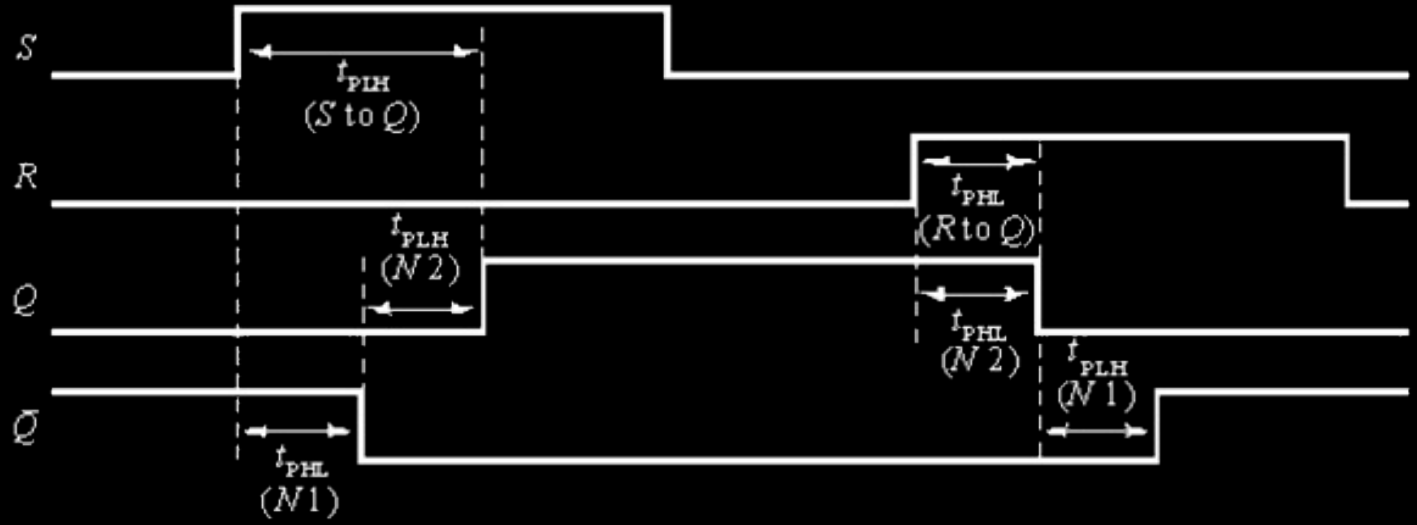
S: “set”; R: “reset”

S = 1, R = 0: Q = 1
S = 0, R = 1: Q = 0
S = 0, R = 0: “keep”

Be aware of delays

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





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

$\overline{S}$	$\overline{R}$	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$
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

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

"set" and "reset" cannot be both 1

S	R	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$
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

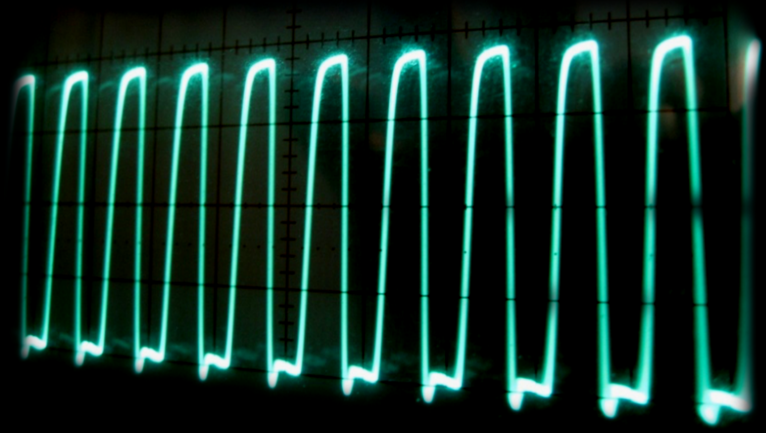
Forbidden state

Reading from latches

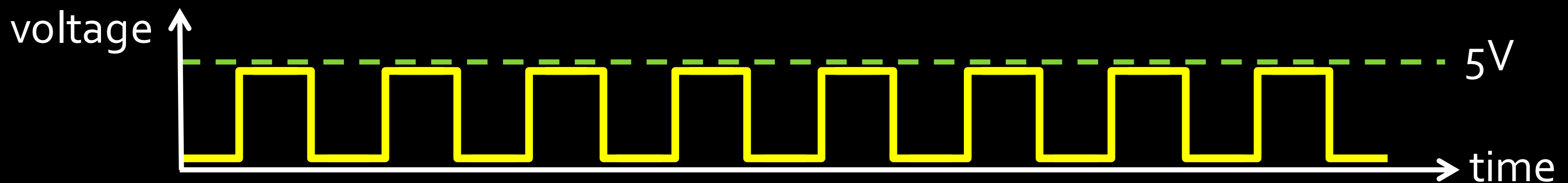
- Now we have circuit units that can store high or low values.
- How do we distinguish
  - "5 highs in a row"
  - "10 highs in a row"
- We want to **sample** the signal with certain frequency.
- Need to use some sort of timing signal, to let the circuit know when the output may be sampled.

→ clock signals.

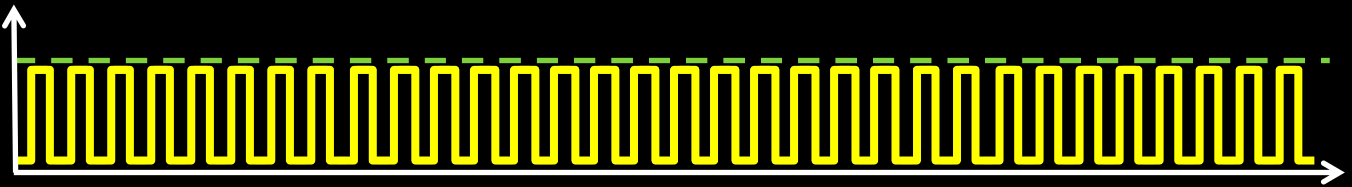
Clock signals



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

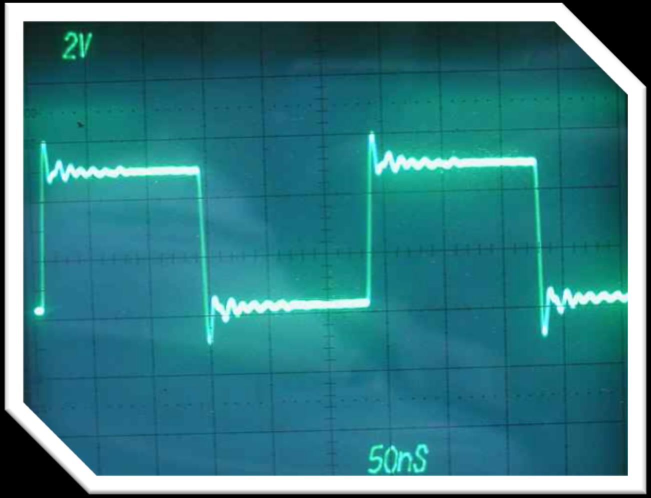


- But looks more like (frequency is usually high):



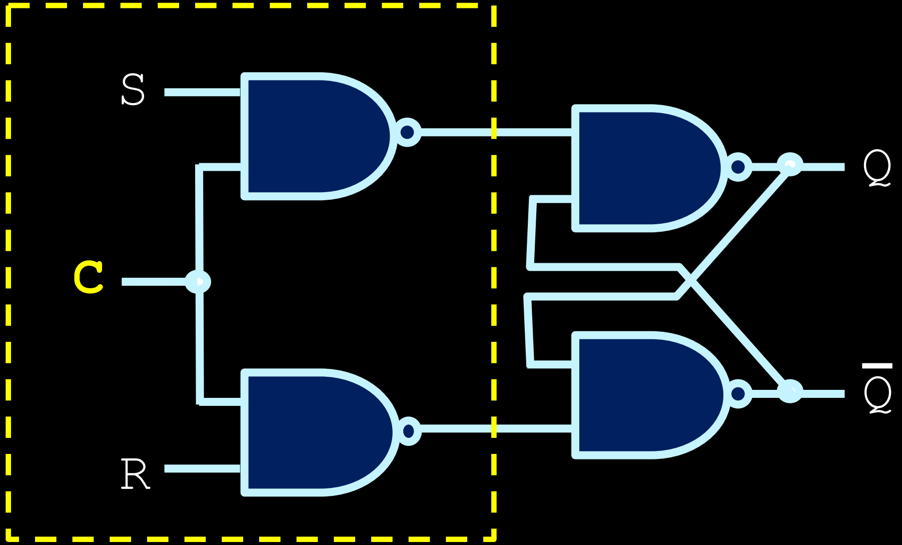
Signal restrictions

- 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

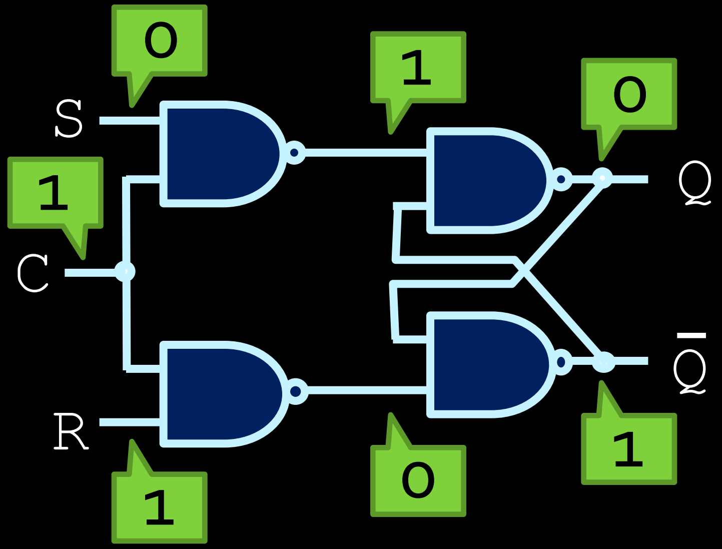
Clocked SR latch

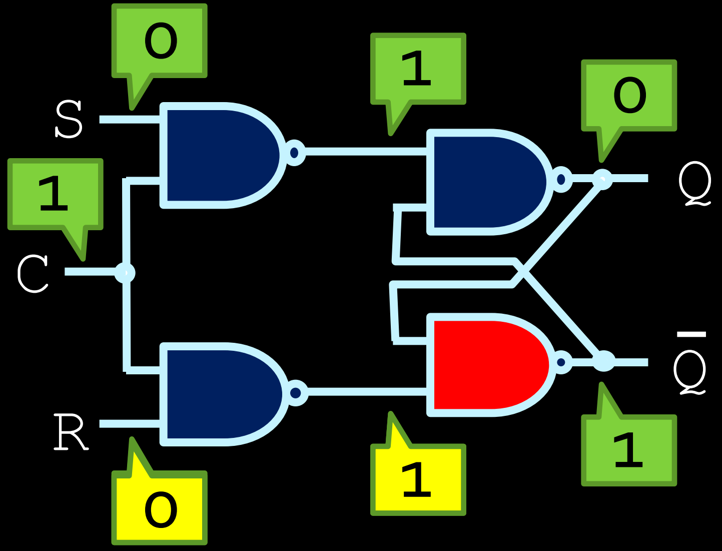


- By adding another layer of NAND gates to the $\overline{S}\overline{R}$ latch, we end up with a clocked SR latch circuit.
- The clock is often connected to a pulse signal that alternates regularly between 0 and 1.

Clocked SR latch behaviour

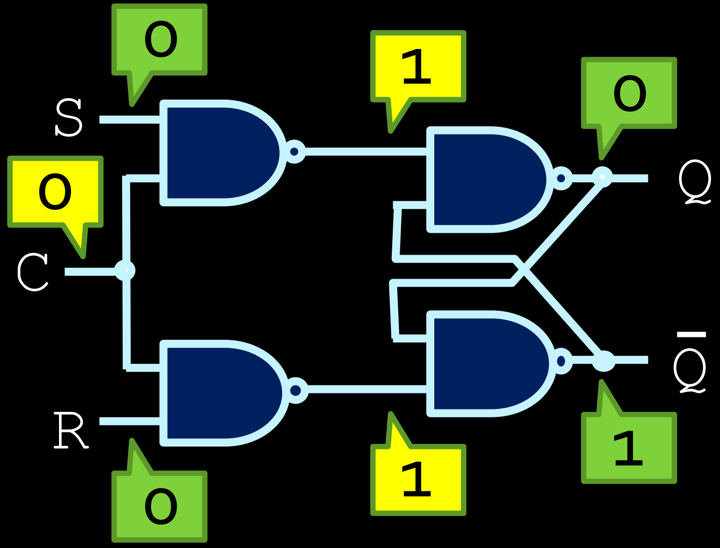
- Same behaviour as SR latch, but with timing:
  - Start off with S=0 and R=1, like earlier example.
  - If clock is high, the first NAND gates invert those values, which get inverted again in the output.
  - Setting both inputs to 0 maintains the output values.

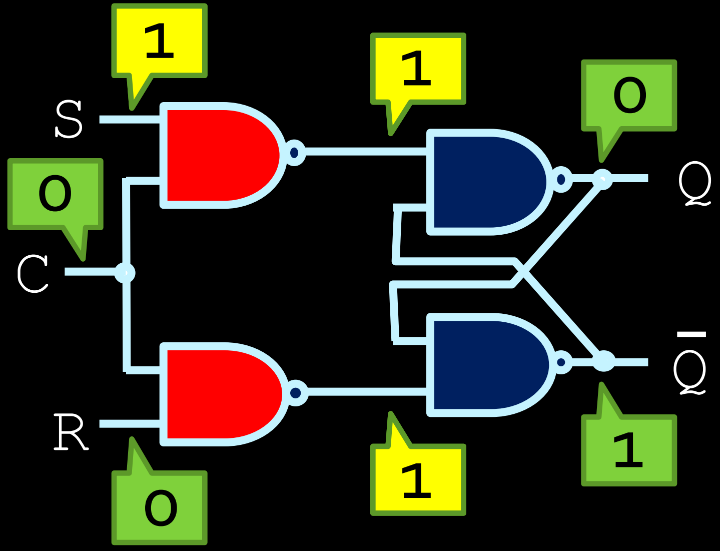


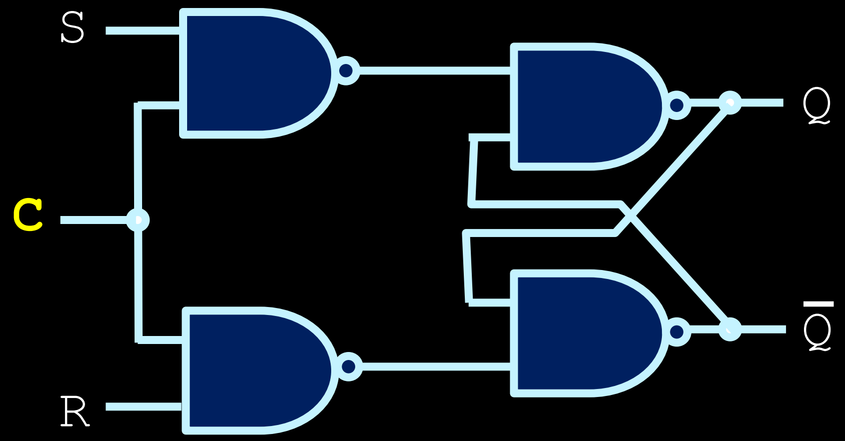


Clocked SR latch behaviour

- 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.



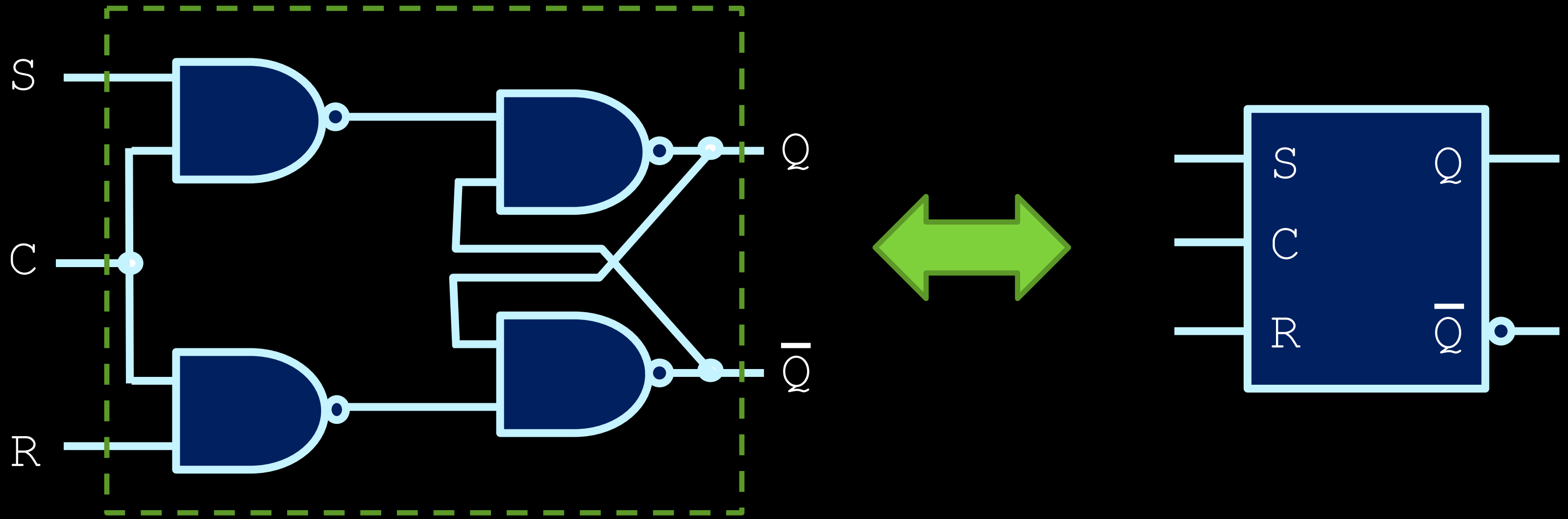




When clock is high, behave like a SR latch.

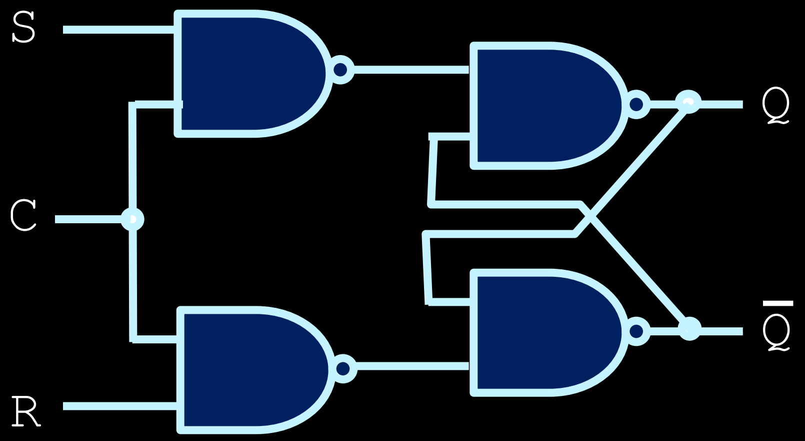
When clock is low, S and R are blocked and there is no way to change the output.

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 $\overline{Q}$ output is simply the notation to use to denote the inverted output value. It's not an extra NOT gate.

Clocked SR latch behaviour



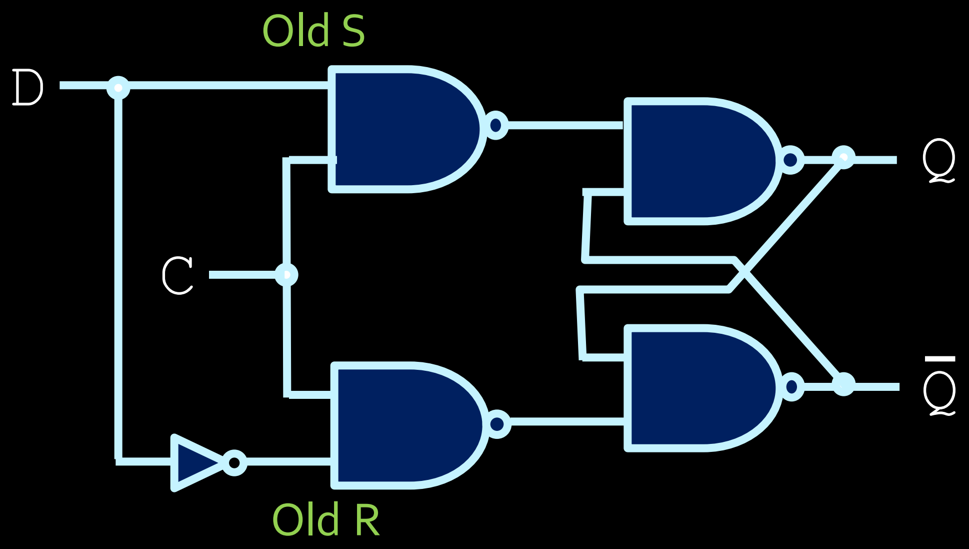
Q_T	S	R	Q_{T+1}	Result
0	0	0	0	no change
0	0	1	0	reset
0	1	0	1	set
~~0~~	~~1~~	~~1~~	~~?~~	~~???~~
1	0	0	1	no change
1	0	1	0	reset
1	1	0	1	set
~~1~~	~~1~~	~~1~~	~~?~~	~~???~~

- 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

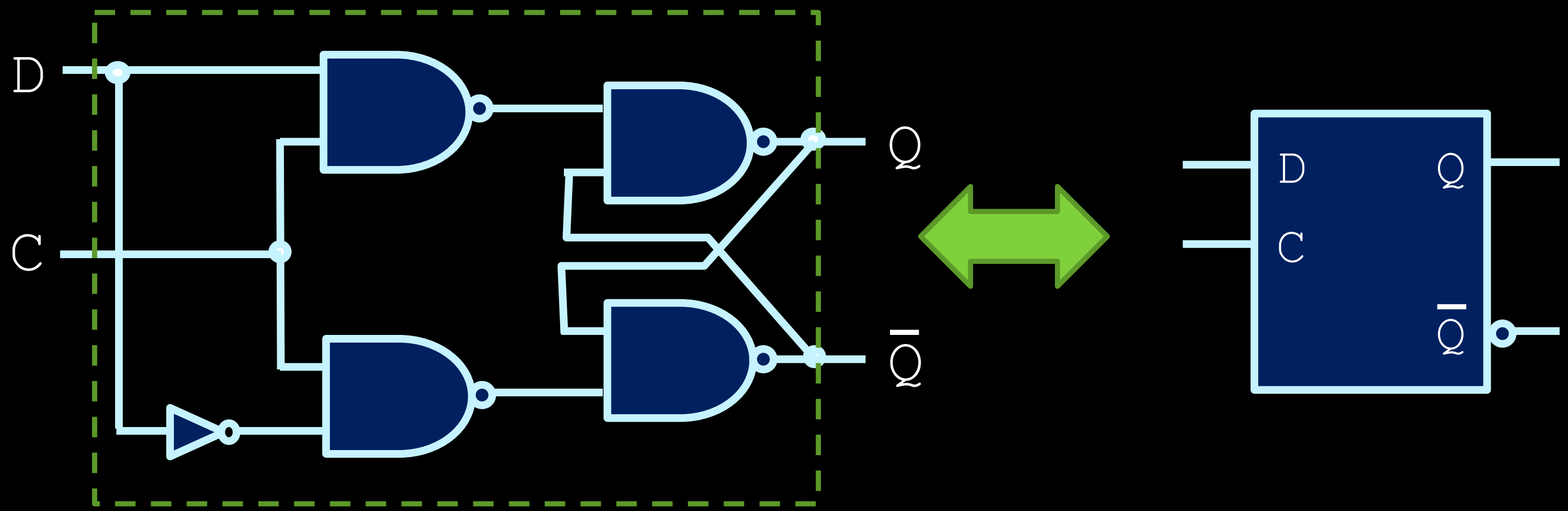
D latch



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.

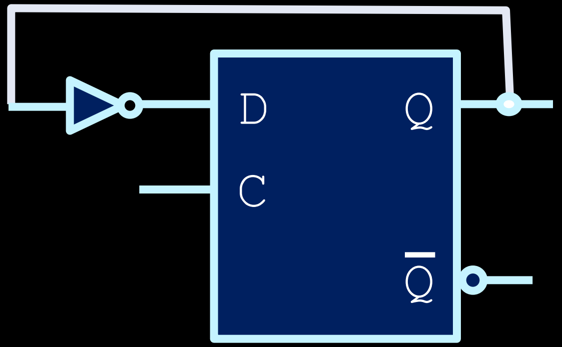
D latch

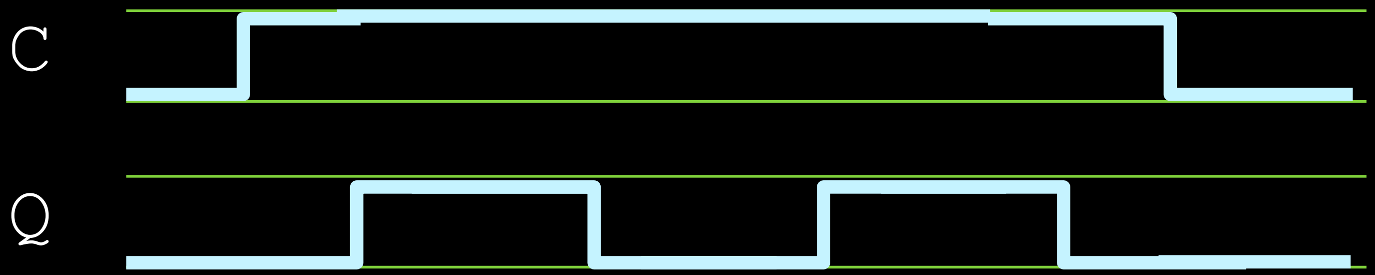


- This design is good, but still has problems.
  - i.e. timing issues.

Latch timing issues

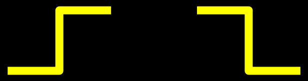
- Consider the circuit on the right:
- When the clock signal is high, the output looks like the waveform below:
  - Output keeps toggling back and forth.

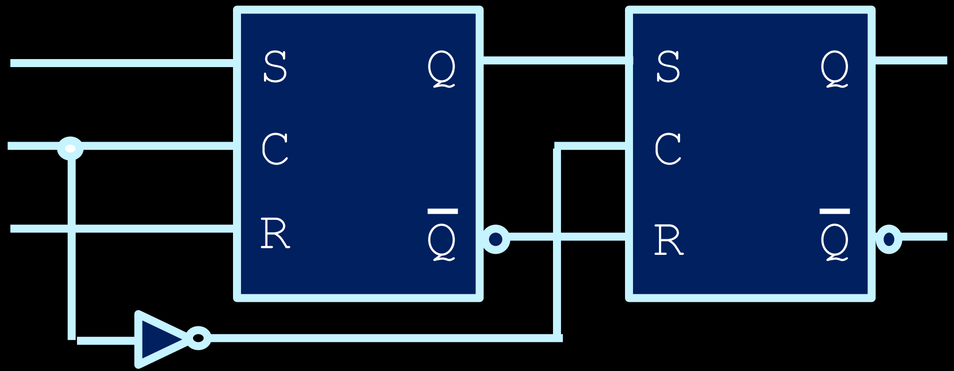




Would be nice if Q changes only once within one clock cycle

Latch timing issues

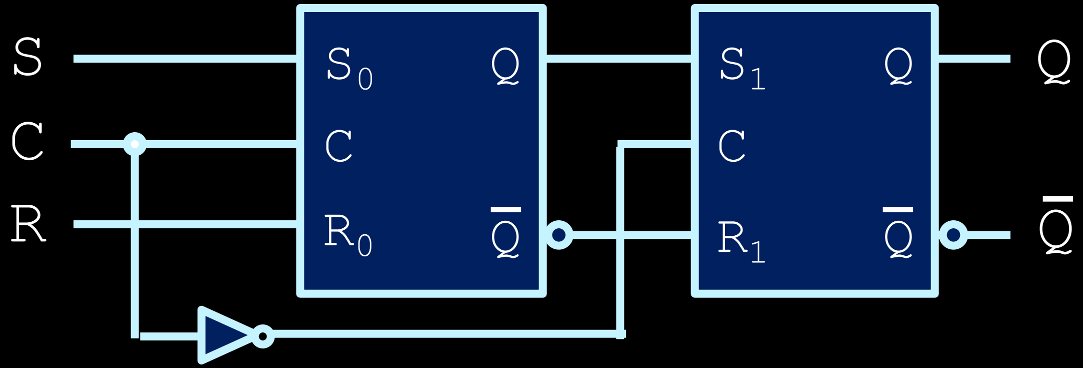
- Preferable behaviour:
  - Have output change only once when the clock pulse changes. 
  - Solution: create disconnect between circuit output and circuit input, to prevent unwanted feedback and changes to output.



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.

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



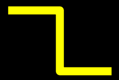
Demo: Human flesh flip-flop

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

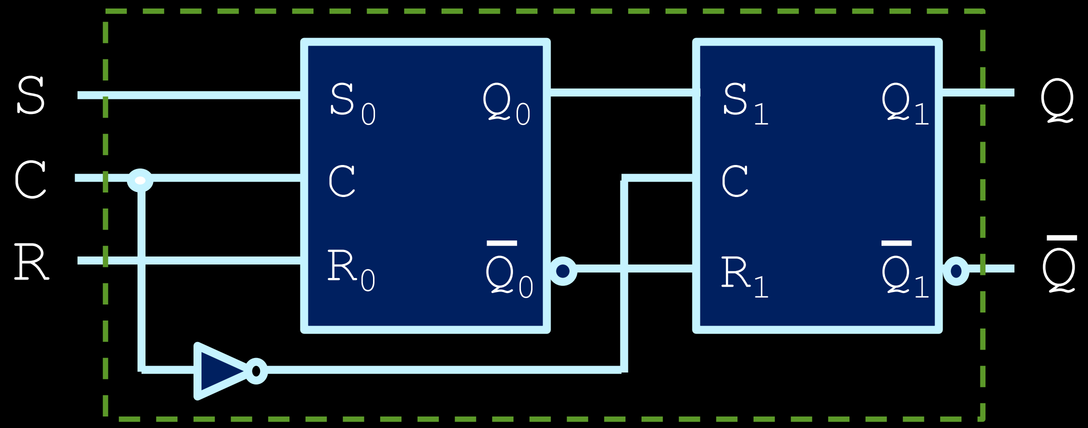
Summary:

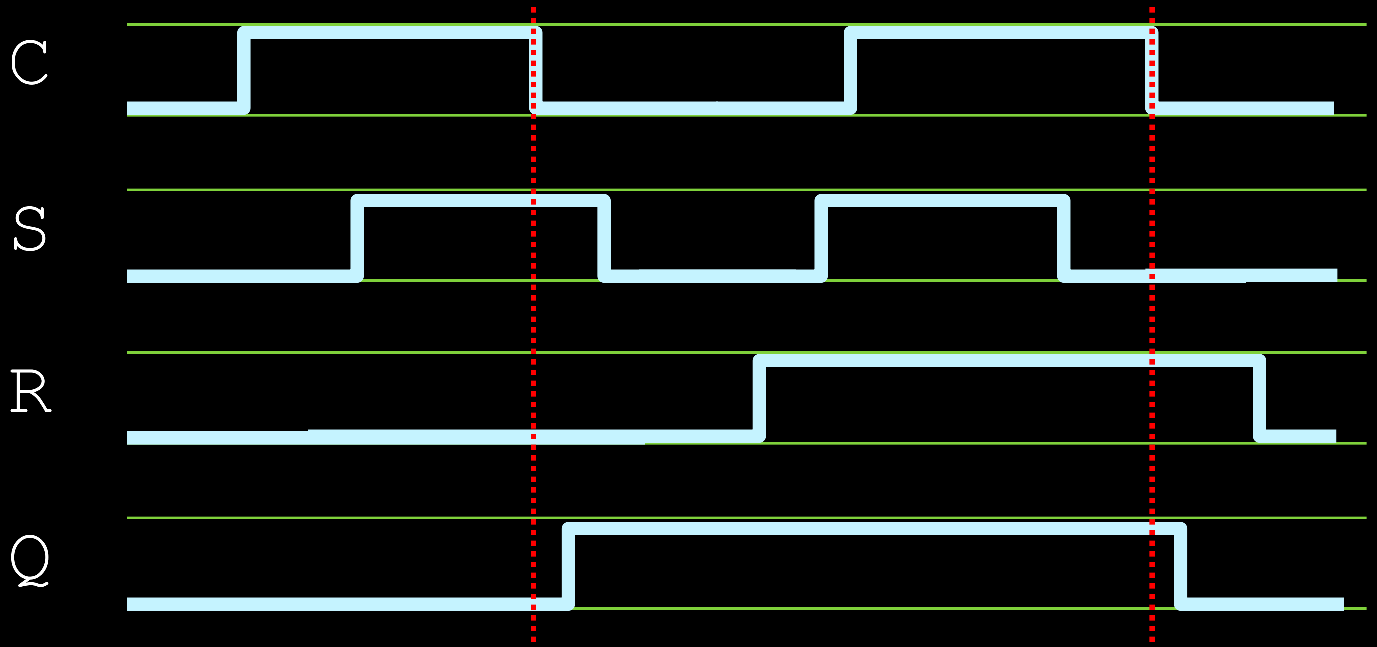
1. For input to propagate to output, it takes each of the latches to be active once.

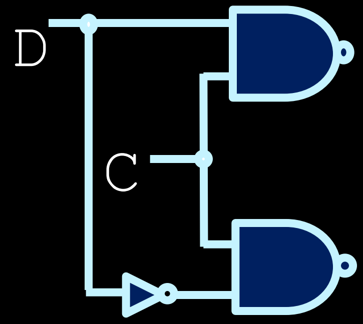
2. Output can only change upon "flop", which is basically the falling edge of the clock signal 

3. At most one change per clock cycle

Change of Q triggered by falling clock edges

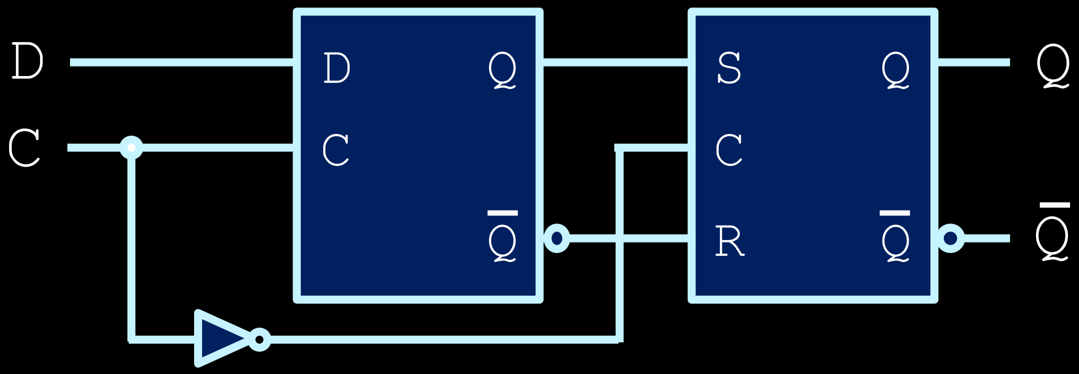






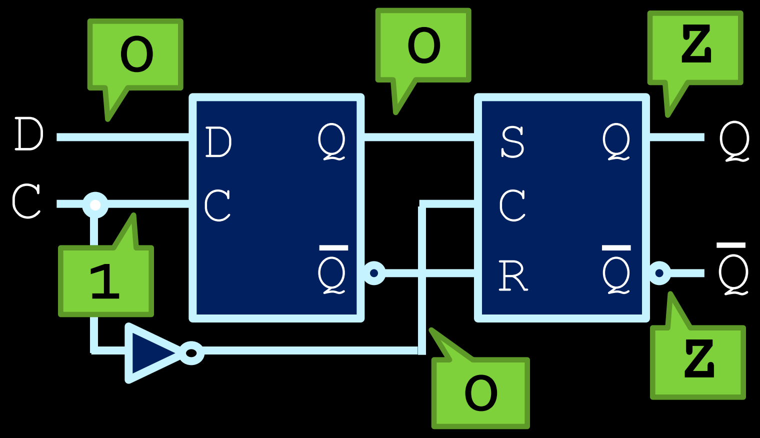
Edge-triggered D flip-flop

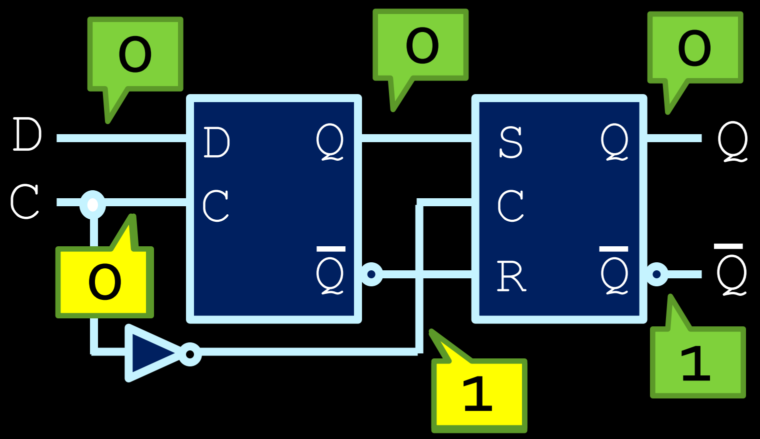
- SR flip-flops still have issues of unstable behavior (forbidden state)
- Solution: D flip-flop
  - Connect D latch to the input of a SR latch.
  - Negative-edge triggered flip-flop (like the SR)



Flip-flop behaviour

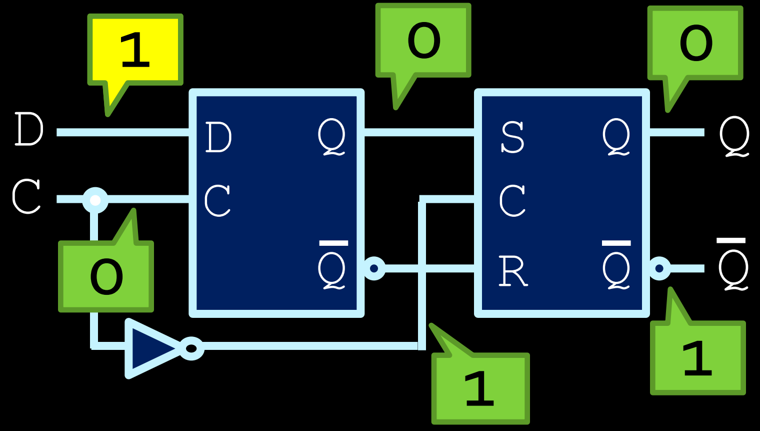
- Observe the behaviour:
  - If the clock signal is high, the input to the first flip-flop is sent out to the second.
  - The second flip-flop doesn't do anything until the clock signal goes down again.
  - When it clock goes from high to low, the first flip-flop stops transmitting a signal, and the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

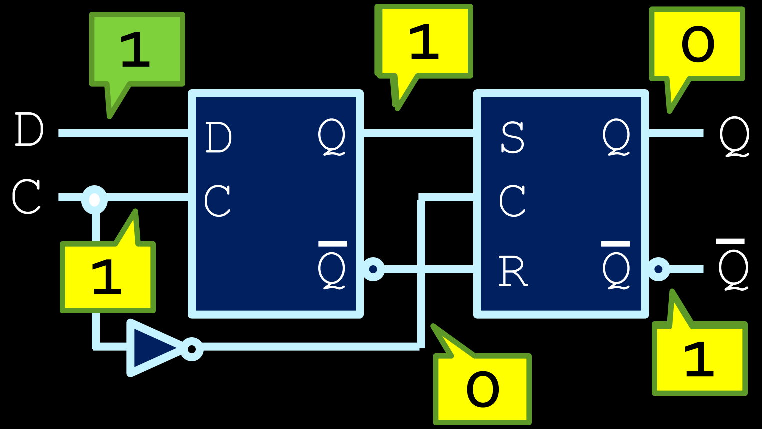




Flip-flop behaviour

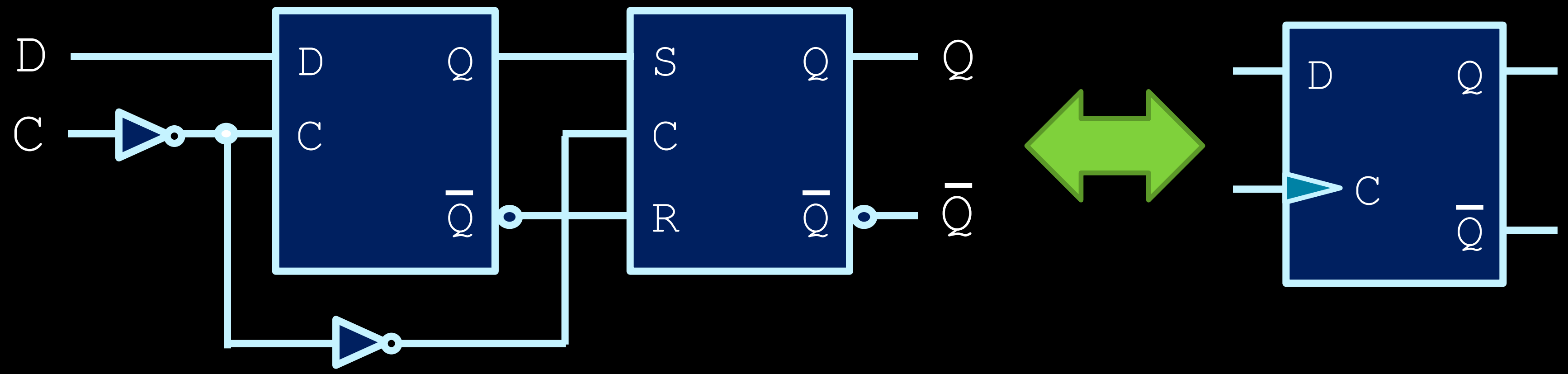
- Continued from previous:
  - If the input to D changes, the change isn't transmitted to the second flip-flop until the clock goes high again.
  - Once the clock goes high, the first flip-flop starts transmitting at the same time as the second flip-flop stops.





Edge-triggered flip-flop

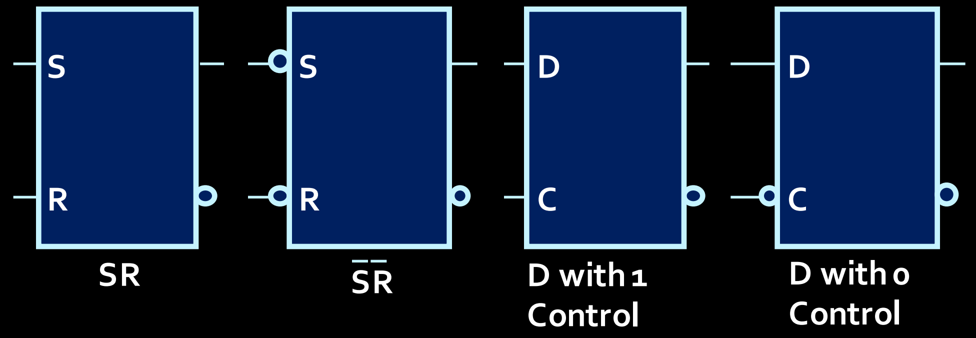
- Alternative: positive-edge triggered flip-flops



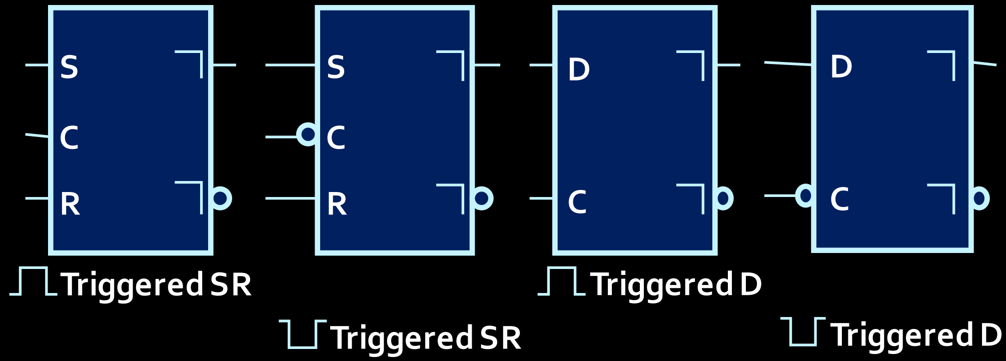
- These are the most commonly-used flip-flop circuits (and our choice for the course).

Notation

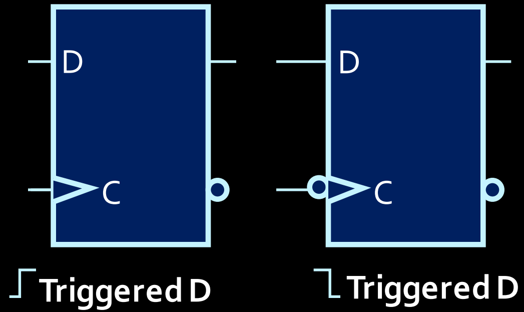
- Latches



- Master-slave flip-flops



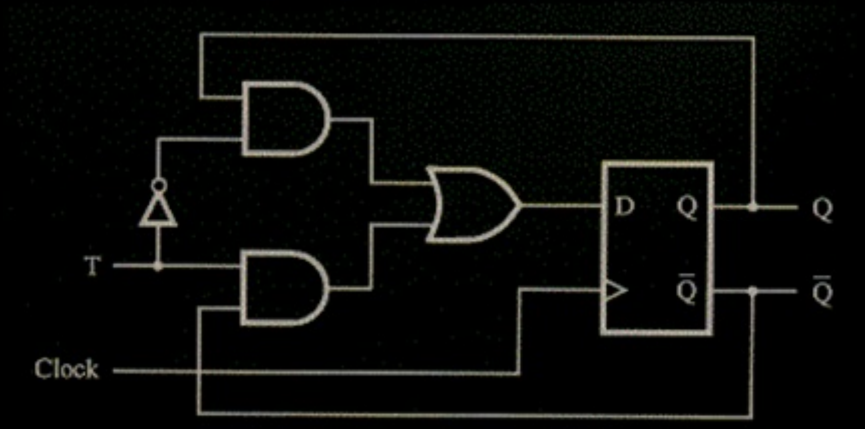
- Edge-triggered flip-flops

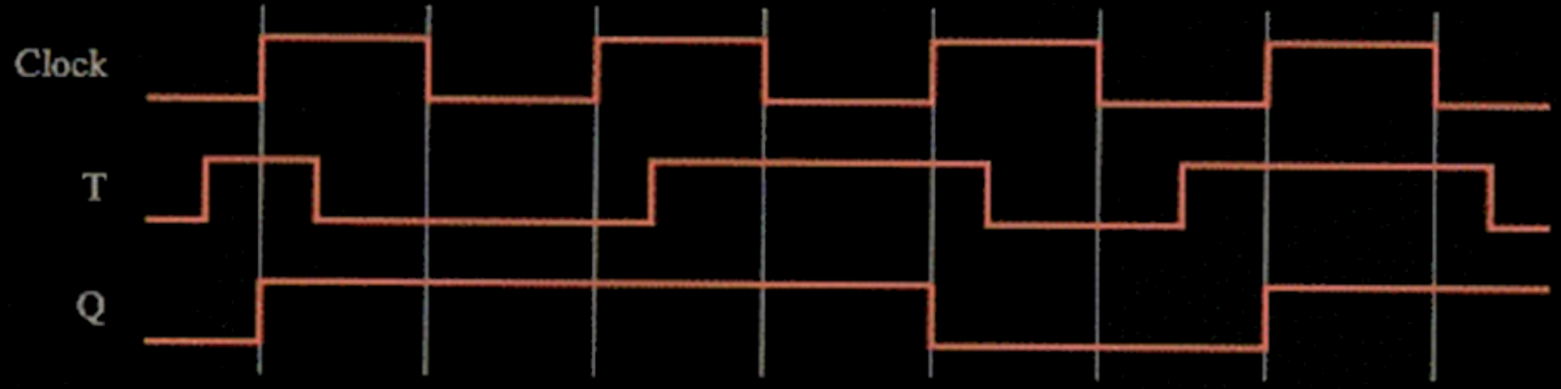


<u>Note:</u> While all these are possible, we mainly use edge-triggered D flip-flops in our designs.

Other Flip-Flops

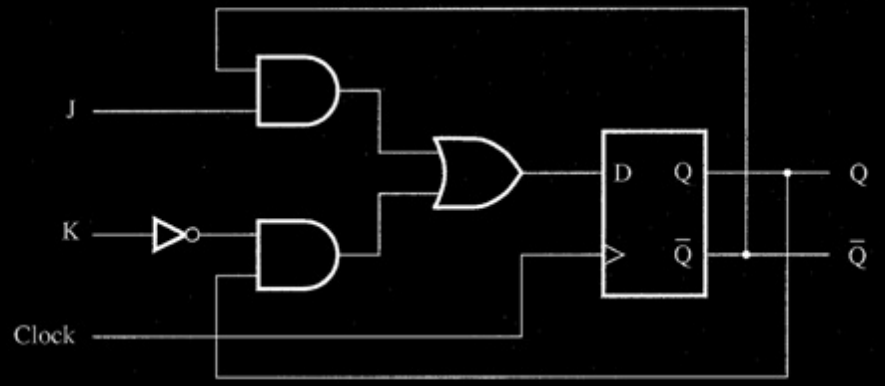
- The T flip-flop:
  - Like the D flip-flop, except that it toggles its value whenever the input to T is high.

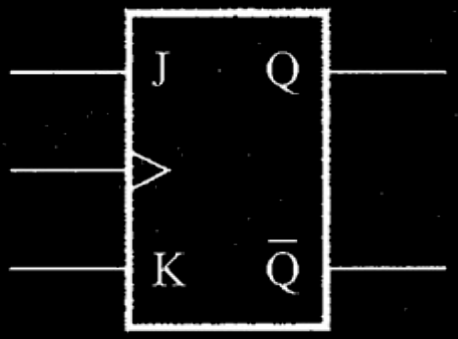




Other Flip-Flops

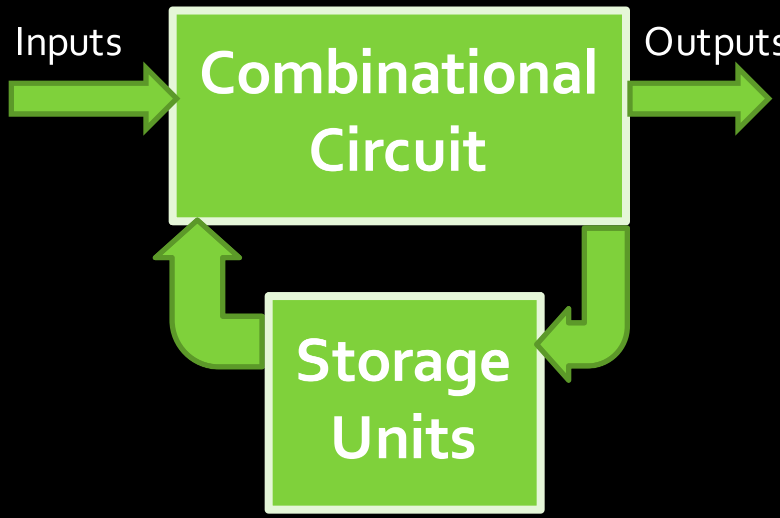
- The JK Flip-Flop:
  - Takes advantage of all combinations of two inputs (J & K) to produce four different behaviours:
    - if J and K are 0, maintain output.
    - if J is 0 and K is 1, set output to 0.
    - if J is 1 and K is 0, set output to 1.
    - if J and K are 1, toggle output value.





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 ☺

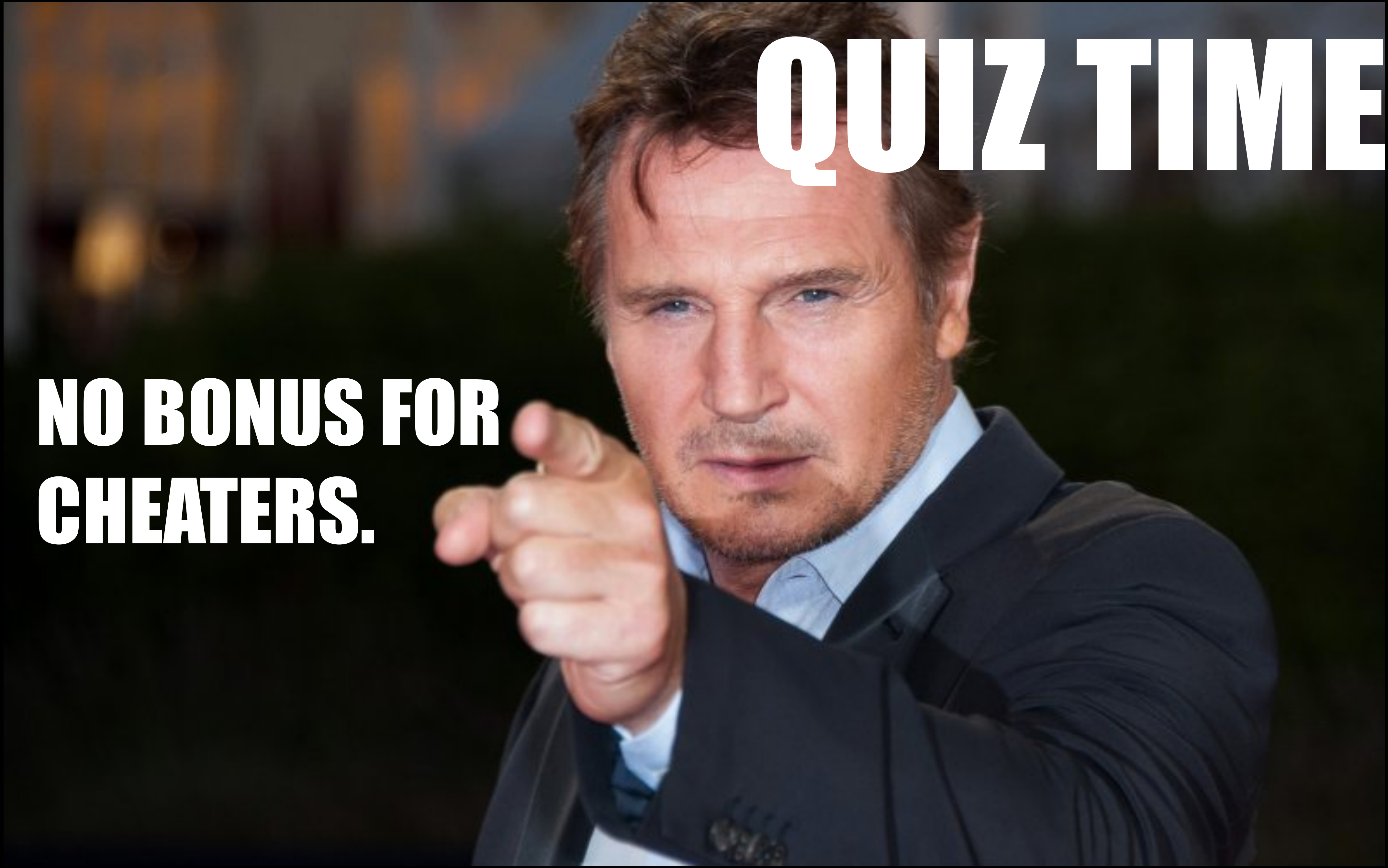


Today we learned

- Sequential circuits – circuits with memory
- Latch (SR, D)
- Flip-flop (SR, D)

Next week

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


QUIZ TIME
NO BONUS FOR CHEATERS.

Q1: Write -5 as signed 4-bit binary number.

Answer: 1011

5 is 0101, it’s 2’s-complement is 1011

Q2: Add two signed 4-bit integers 6 + 7, what result (also a 4-bit signed integer) will be produced?

A. 13 B. -2

C. -3 D. None of above

Answer: C

6 is 0110, 7 is 0111, adding them up we get 1101, which is -3 because 1101's 2's-complement is 0011 (3).

This situation is called an **overflow**, since the expected result 13 is exceeding the range of value that a 4-bit signed integer can represent (-8 ~ 7).

Try this C code

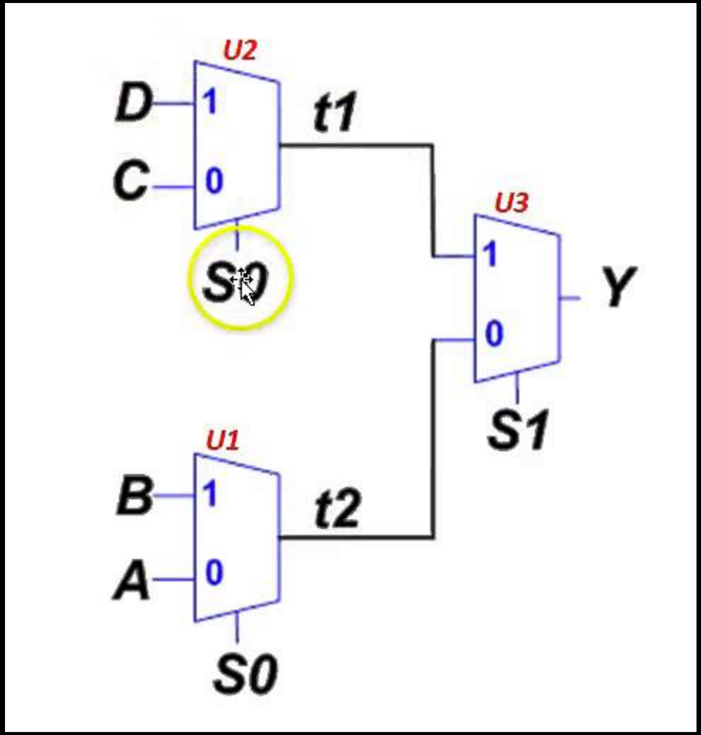
```c
#include <stdio.h>


int main()
{
    /* char is 8-bit integer */
    signed char a = 100;
    signed char b = 120;
    signed char s = a + b;
    printf("%d\n", s);
}
```

Q3: What is output Y when `S0 = 1` And `S1 = 0` ? Answer A, B, C or D.

Answer: B

S1 = 0 selects either A or B,
S0 = 1 selects B rather than A


U2
D
1
C
0
S0
t1
U3
1
0
Y
S1
U1
B
1
A
0
S0
t2