Sequential Circuits

Something to consider…

- Computer specs use terms e
  "8 GB of RAM" and :
  processors".
  - What do these terms mean?
    - RAM = Random Access Memory; 8GB = 8 billion `bytes`
    - 2.2 GHz = 2.2 billion clock pulses per second.
  - But what does this mean in circuitry?
    - How do you use circuits to store values?
    - What is the purpose of a clock signal?

Something else to consider…

- How does the "Tickle Me Elmo" work?

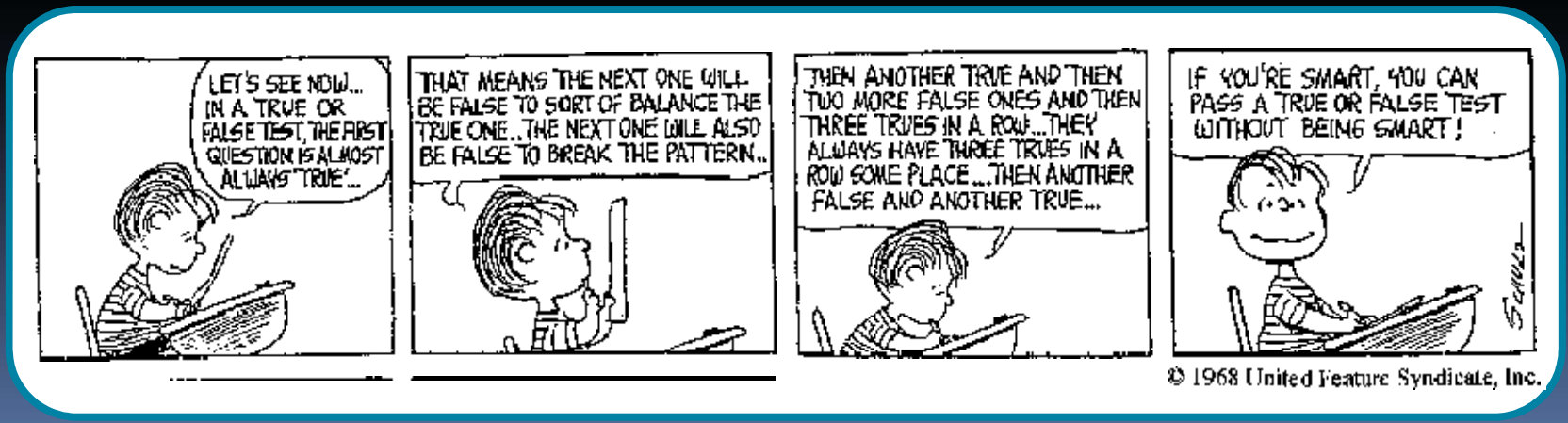
Two kinds of circuits

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

therefore there is the concept of state in sequential circuits

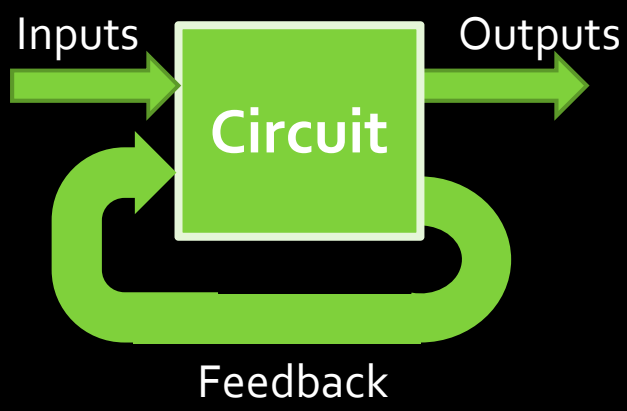
Sequential circuits

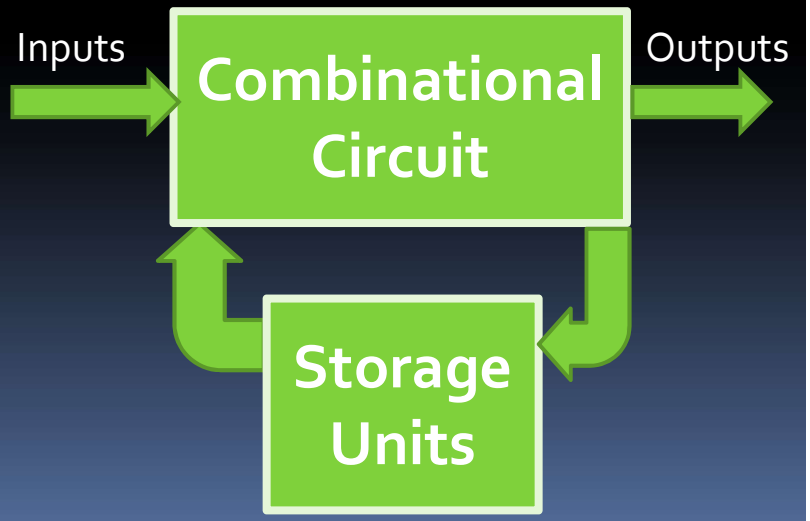
- This creates circuits whose internal state can change over time, where the same input values can result in different outputs.
- Why would we need circuits like this?
  - Memory values
  - Reacting to changing inputs



Creating sequential circuits

- Essentially, sequential circuits are a result of having feedback in the circuit.
  - How is this accomplished?
  - What is the result of having the output of a component or circuit be connected to its input?





Gate Delay

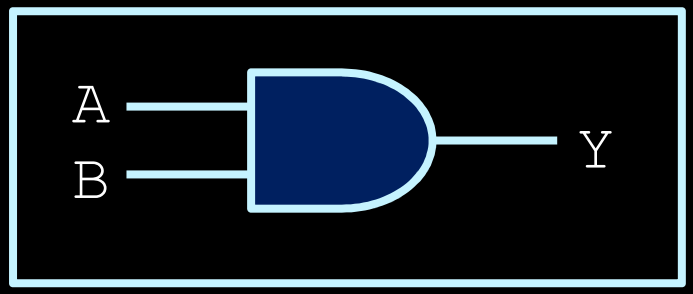
- Even in combinational circuits, outputs don't change instantaneously.

- Gate Delay or Propagation Delay:
  - "The length of time it takes for an input change to result in the corresponding output change."

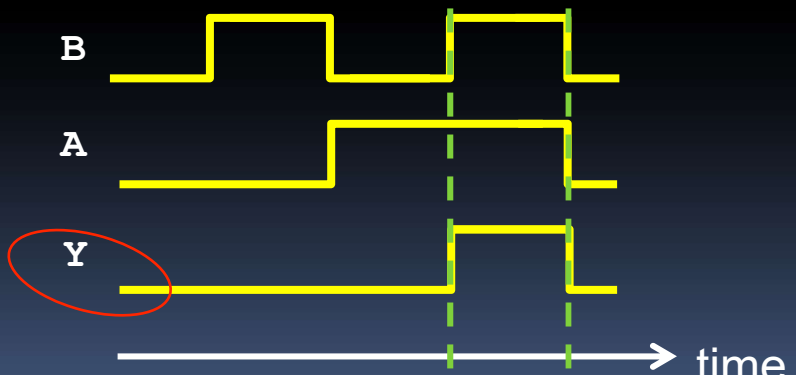
every gate adds delay to circuit; and they are different for different operations

the propagation delay, or gate delay, is the length of time which starts when the input to a logic gate becomes stable and valid to change, to the time that the output of that logic gate is stable and valid to change

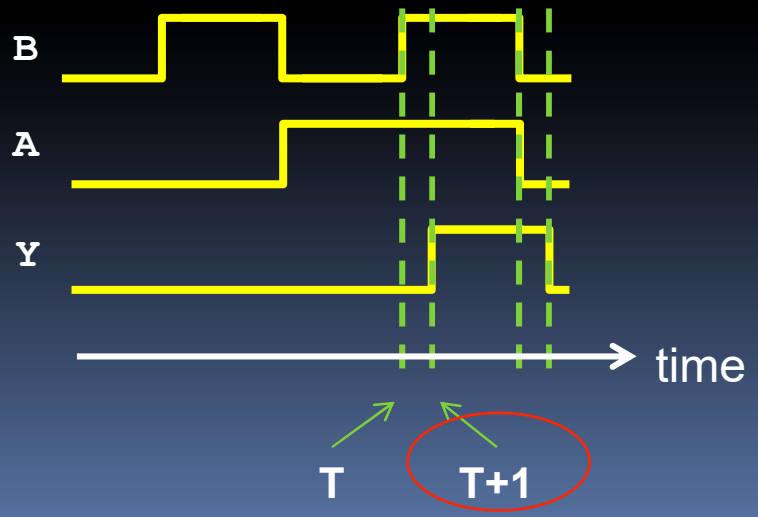
Gate delay example



Ideal

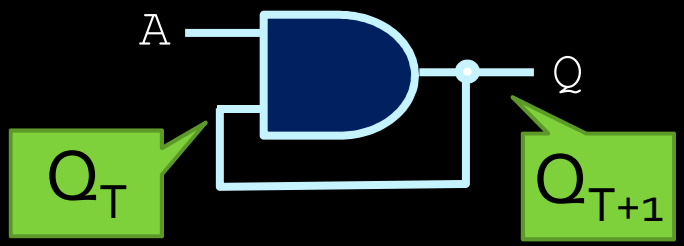


Considering delays



Feedback Circuit Example (AND)

- Some gates don't have useful results when outputs are fed back on inputs.



because anything AND 0 —> 0

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

If A=0, Q_{T+1} becomes 0 no matter what Q_T was.

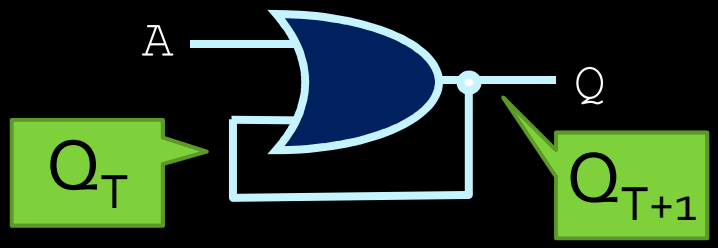
What happens next for later values of A?

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

Q_{T+1} gets stuck at 0 and cannot change ☹

Feedback Circuit Example (OR)

- Some gates don't have useful results when outputs are fed back on inputs.



idea is that changing A does not affect Q at all

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

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

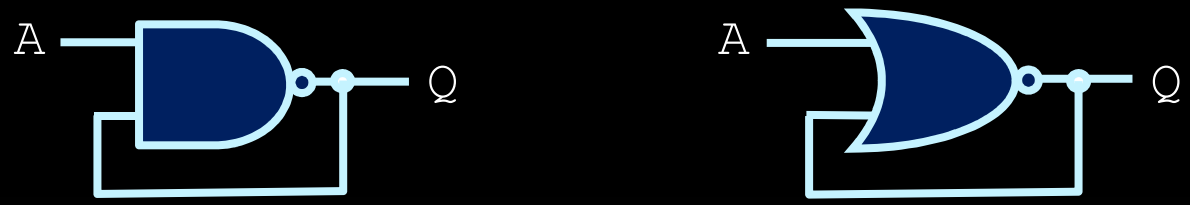
If A=1, Q_{T+1} becomes 1 no matter what Q_T was.

What happens next for later values of A?

Q_{T+1} gets stuck at 1. Not very useful ☹

Feedback Examples (NAND, NOR)

- NAND, NOR gates w/ feedback have more interesting characteristics, which lend themselves to storage devices.

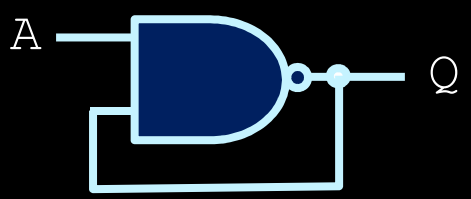


- What makes NAND and NOR feedback circuits different?
  - Unlike the AND and OR gate circuits (which get stuck), the output Q_{T+1} can be changed, based on A.

Based on A !!!

Feedback Example (NAND)

- Let's assume we set A=0
  - Then, output Q will go to 1.
  - If we leave A unchanged we can store 1 indefinitely!
- If we set A=1, Q's value can change, but there's a catch!



Q can change depending on A, there is no deadend

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

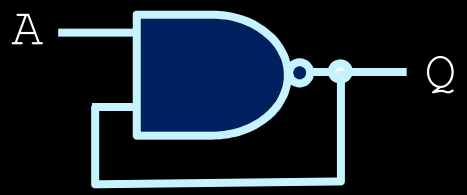
What happens in these last two scenarios?

Unsteady state! Can't store 0 long!

A=0 —> always 1
A=1 —> unstable

loops over for the last 2 loops

NAND waveform behaviour



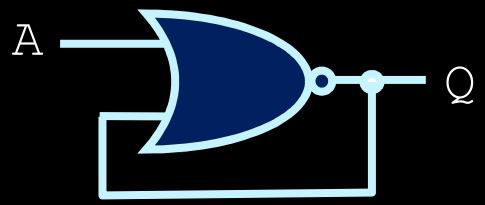
A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

Gate delay. Output does not change instantaneously

We want to avoid this. We should be able to store high and low values for as long as we want, and change those values as needed.

Feedback Example (NOR)

- Let's assume we set A=1
- Then, output Q will go to 0.
- If we leave A unchanged we can store 0 indefinitely!
- If we flip A, we can change Q, but there's a catch here too!



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

A=1 —> always 0
A=0 —> unstable

Feedback behaviour

- NAND behaviour

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

- NOR behaviour

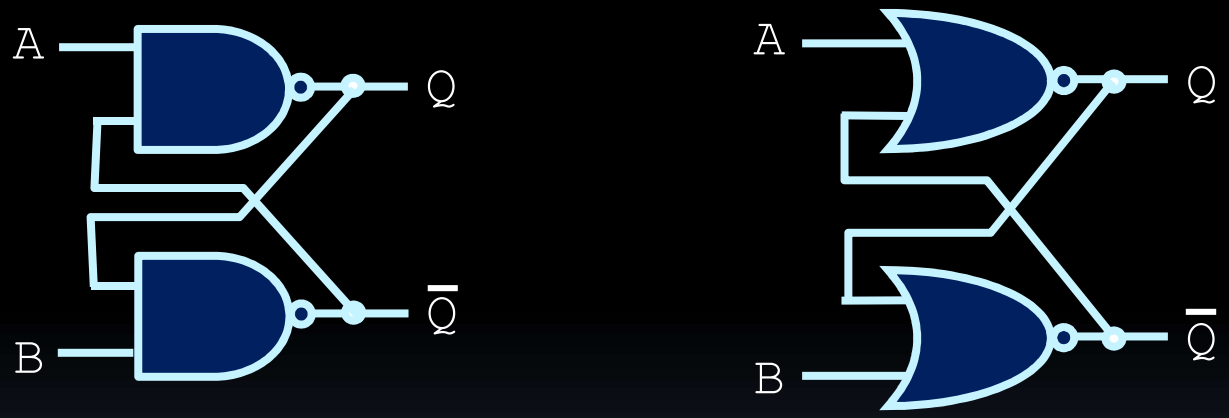
A	Q_T	Q_{T+1}
0	0	1
0	1	0
1	0	0
1	1	0

- Output Q_{T+1} can be changed, based on A.
- However, gates like these that feed back on themselves could enter an unsteady state.

outputs not holding a steady value

Latches

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



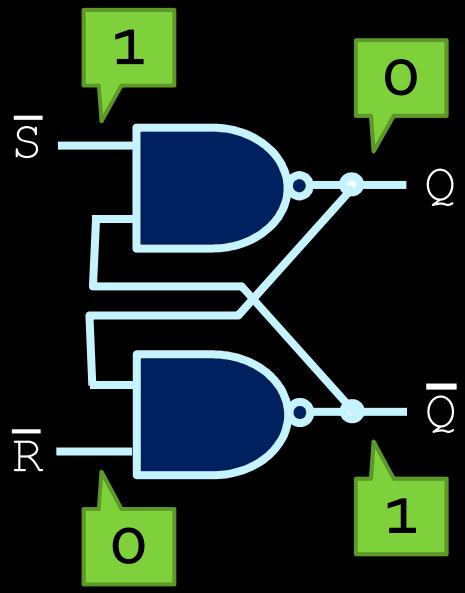
- These circuits are called latches.

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

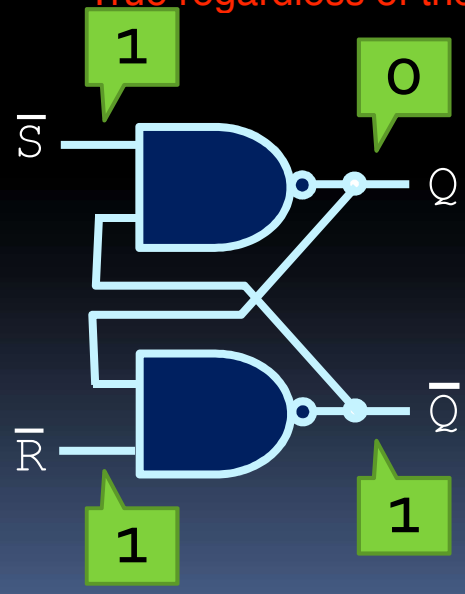
S = 1; R = 0

~S and ~R are active low signals

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



NAND -> 1 if one input is 0
True regardless of the other input

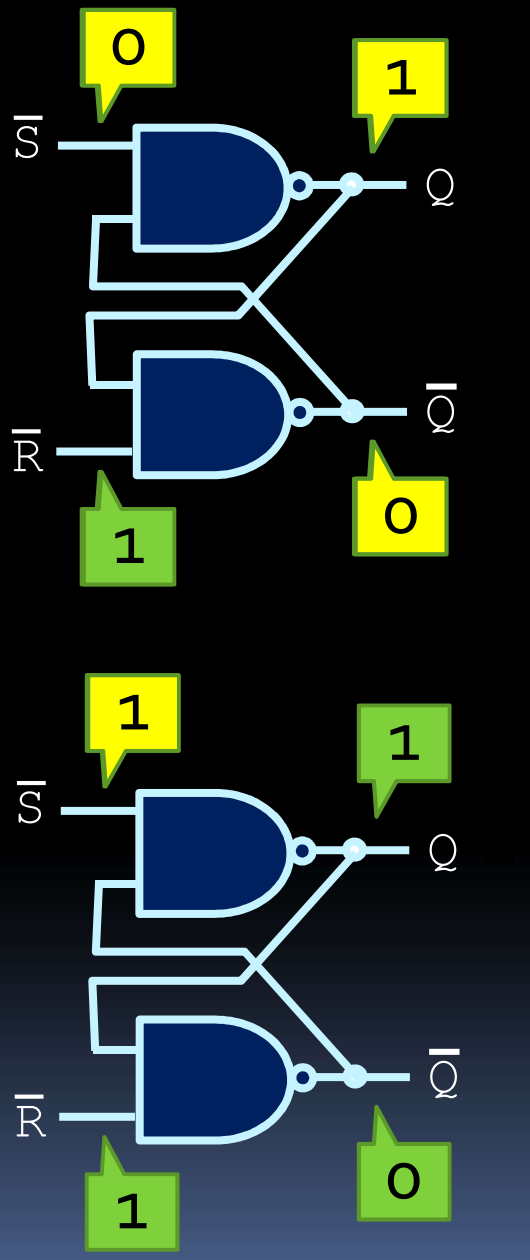


R has not effects on state.

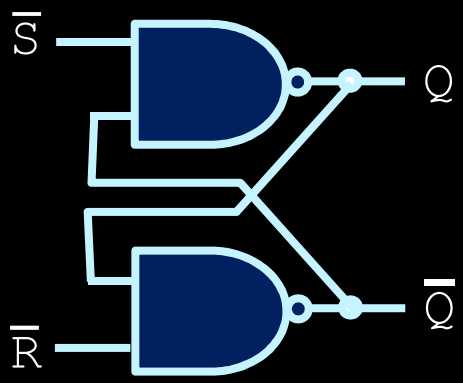
$\overline{S}\overline{R}$ latch

S = 0; R = 1

- (continuing from previous)
  - $\overline{S}$ and $\overline{R}$ start with values of 1, when $\overline{S}$ is set to 0.
  - This sets output Q to 1, which sets the output $\overline{Q}$ to 0.
  - Setting $\overline{S}$ back to 1 keeps the output value $\overline{Q}$ at 0, which maintains both output values.
- Note: inputs of 11 maintain the previous output state!



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



in summary if S and R are both 1 or 0 —> state persists (also, Q and ~Q are complements)

$\overline{S}$	$\overline{R}$	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

unstable

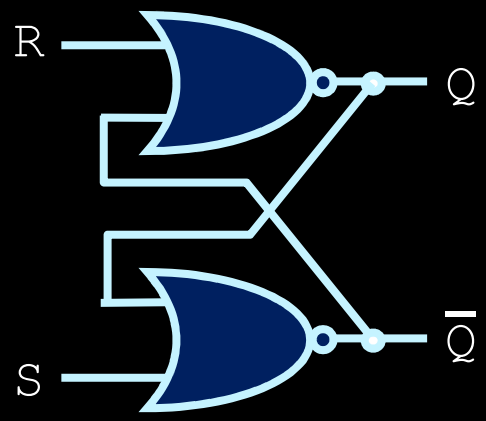
cant turn set and reset on same time

- $\overline{S}$ and $\overline{R}$ are called "set" and "reset" respectively.
- Note how the circuit "remembers" its signal when going from 10 or 01 to 11.
- Going from 00 to 11 produces unstable behaviour!
  - Depending on which input changes first.

When S=1 and R=1; Q and \bar{Q} are identical, alternating between 0 and 1 ===> unstable

1. when S and R are low; output Q and ~Q in constant state
2. if set S high and R is still low; output Q high, and stays high even if S return to low
3. if set R high; and S is still low; output Q low, and stays low even if R returns to lowWhen

SR latch



ouput constant

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

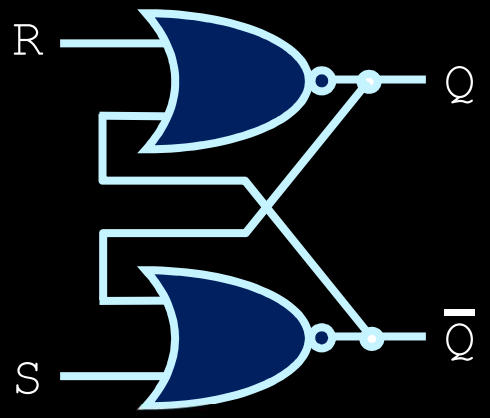
R = S = 1is a restricted combination, because it breaks logical equation with Q = ~Q

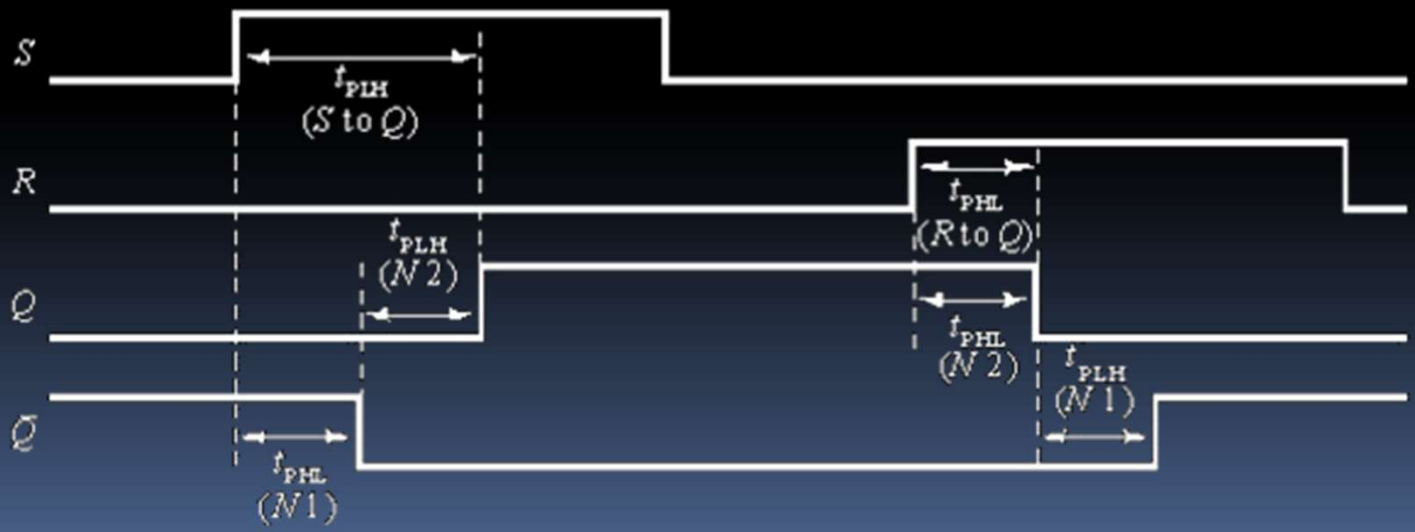
- In this case, S and R are "set" and "reset".
- In this case, the circuit "remembers" previous output when going from 10 or 01 to 00.
- As with $\overline{S}\overline{R}$ latch, unstable behaviour is possible, but this time when inputs go from 11 to 00.

because NOR feedback gurantees
R=S=1 —> Q = 0 regardless of other input

SR latch timing diagram

- Important to note that the output signals don't change instantaneously.





More on instability

unstable at 00 or 11 and unpredictable when going from 00 to 11 or 11 to 00

- Unstable behaviour occurs when a $\overline{S}\overline{R}$ latch's inputs go from `00` to `11`, or a SR latch's inputs go 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 $\overline{S}\overline{R}$ latches, and `11` is considered a forbidden state in NOR-based SR latches.

i.e. set and reset both set to high

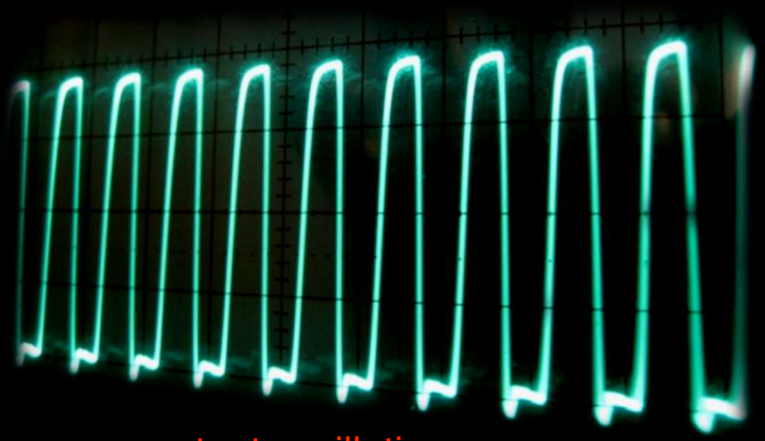
mostly because Q = ~Q is violated in these cases

Introducing the Clock

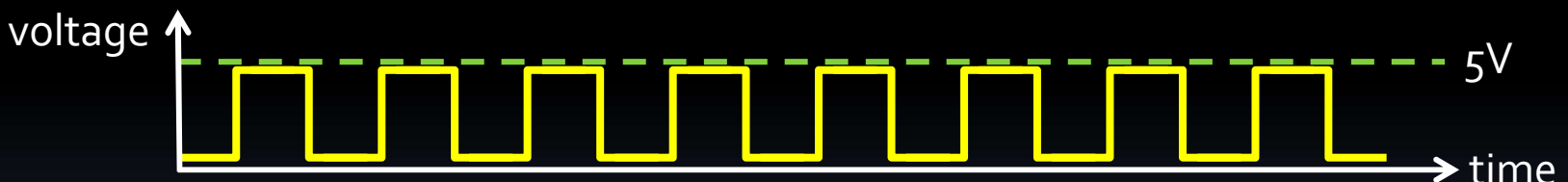
- Now we have circuit units that can store high or low values. How can we read from them?
  - For instance, when do we know when the output is ready to be sampled?
  - If the output is high, how can we tell the difference between a single high value and two high values in a row?
- Need some sort of timing signal, to let the circuit know when the output may be sampled.

  → clock signals.

Clock signals



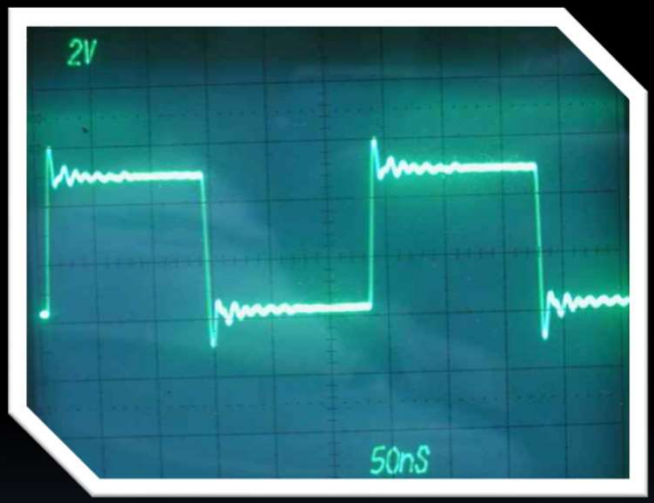
- "Clocks" are a regular pulse signal, where the high value indicates when to update the output of the latch.
- Usually drawn as:



- But looks more like:

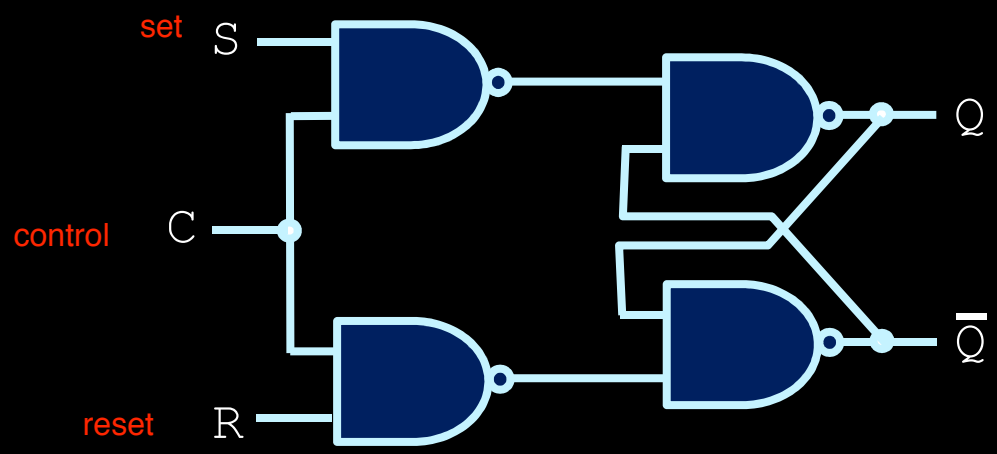
Signal restrictions

- 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).



Intuitively does the same thing as SR latch with NOR gate ; Implemented with 2NOR + 2AND gate which is equivalent to 4NAND

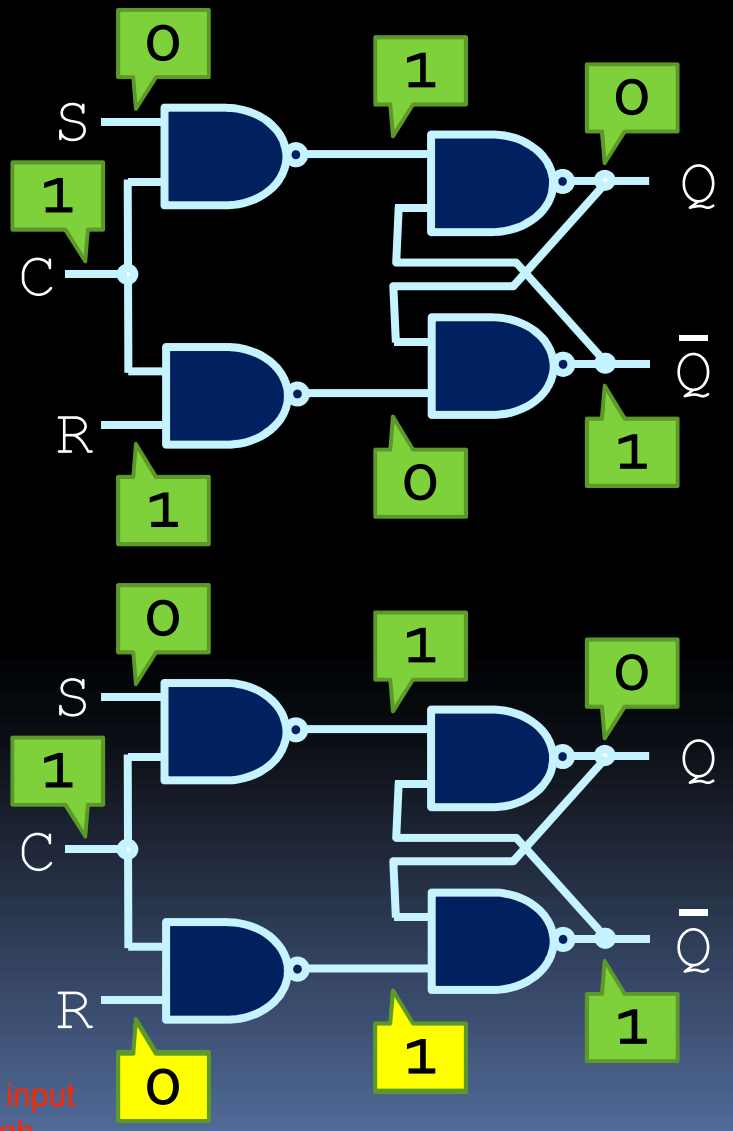
Clocked SR latch



- Adding another layer of NAND gates to the $\overline{S}\overline{R}$ latch gives us a clocked SR latch or gated SR latch)
  - Basically, a latch with a control input signal C.
- The input C is often connected to a pulse signal that alternates regularly between 0 and 1 (clock)

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.



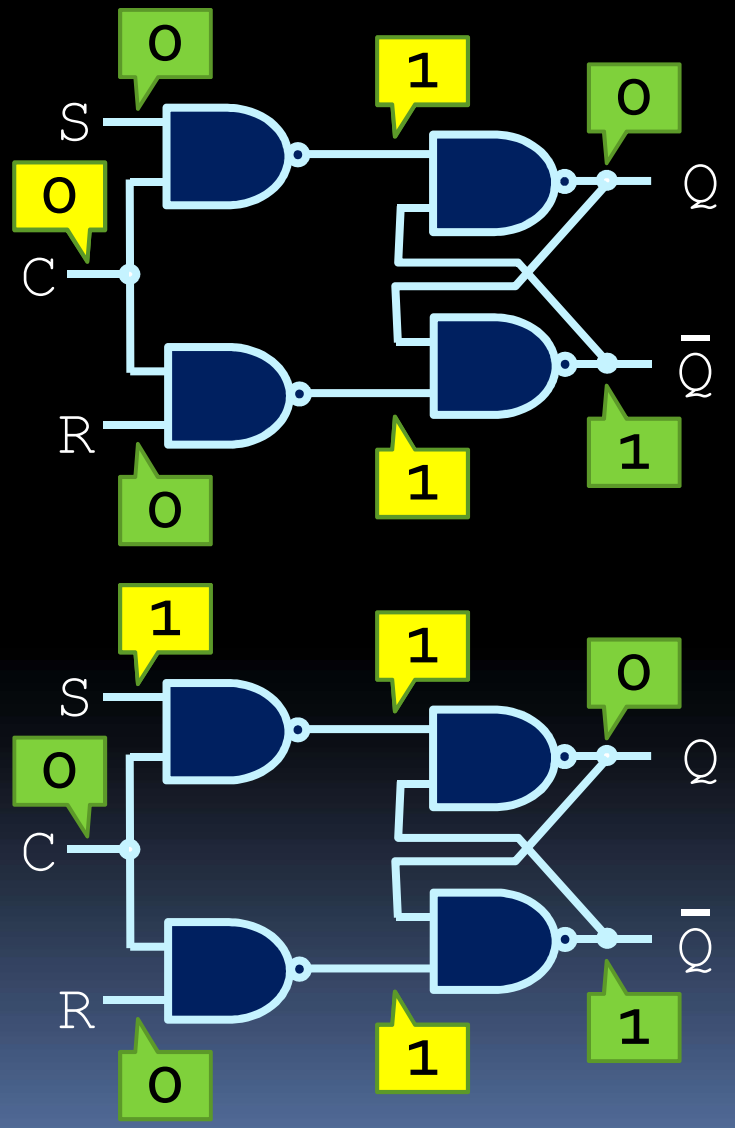
NAND: output high only if input are not both high
1. If one input is high: then NAND inverts the other input
2. if one input is low: then NAND always outputs high

Clocked SR latch behaviour

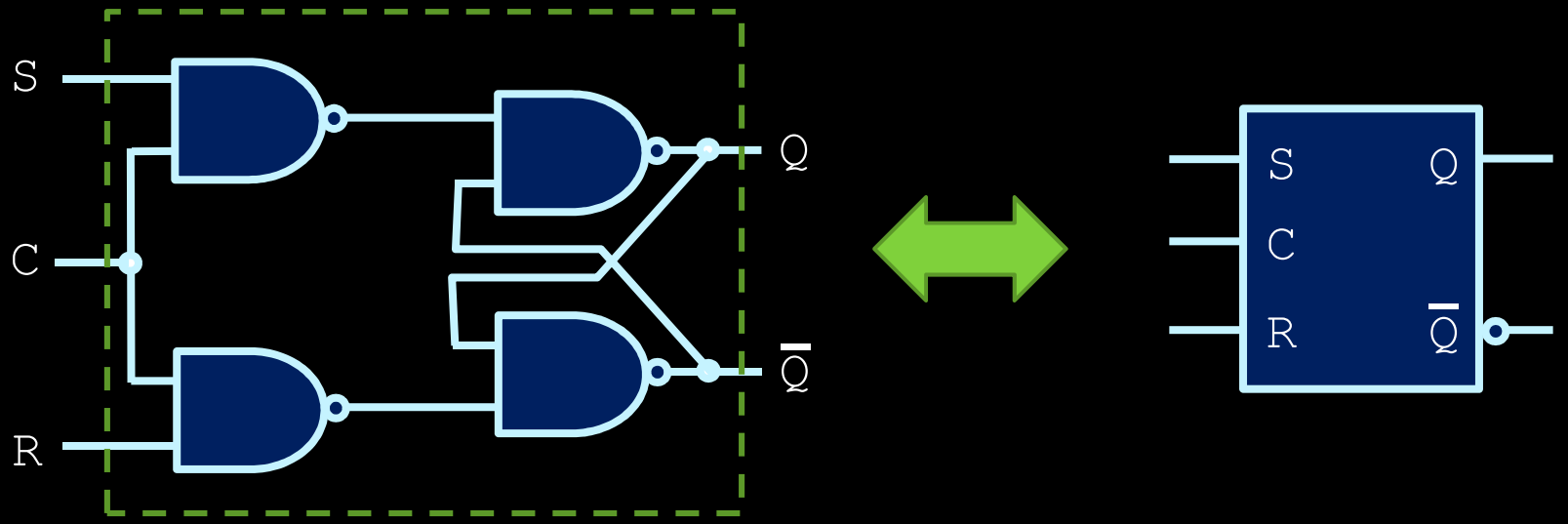
- Continued from previous:
  - Now set the clock low.
  - Even if the inputs change, the low clock input prevents the change from reaching the second stage of NAND gates.
  - Result: the clock needs to be high in order for the inputs to have any effect.

since C=0 the middle wire always 1, Q not changed for varying value of S and R

no effect on middle layer and output

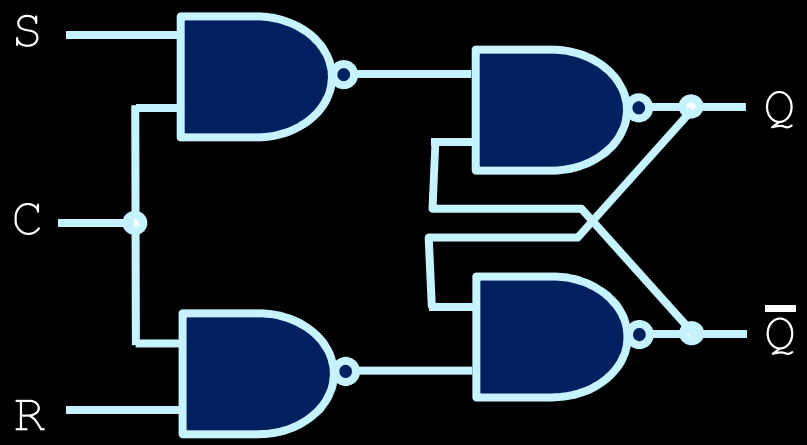


Clocked SR latch



- This is the typical symbol for a clocked SR latch.
- This only allows the S and R signals to affect the circuit when the control 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

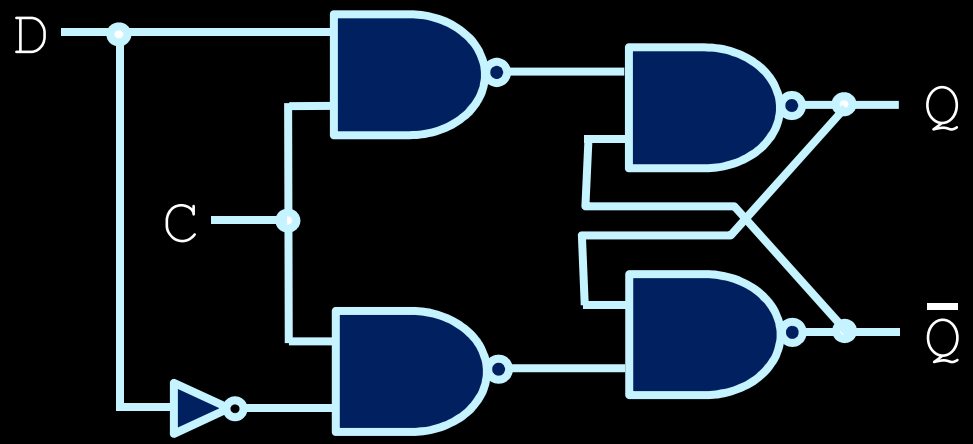


C	S	R	Q_{T+1}	Result
0	X	X	Q_T	no change
1	0	0	Q_T	no change
1	0	1	0	reset
1	1	0	1	set
1	1	1	?	Undefined

default behaviour for C=1

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

D latch (or gated D-latch)

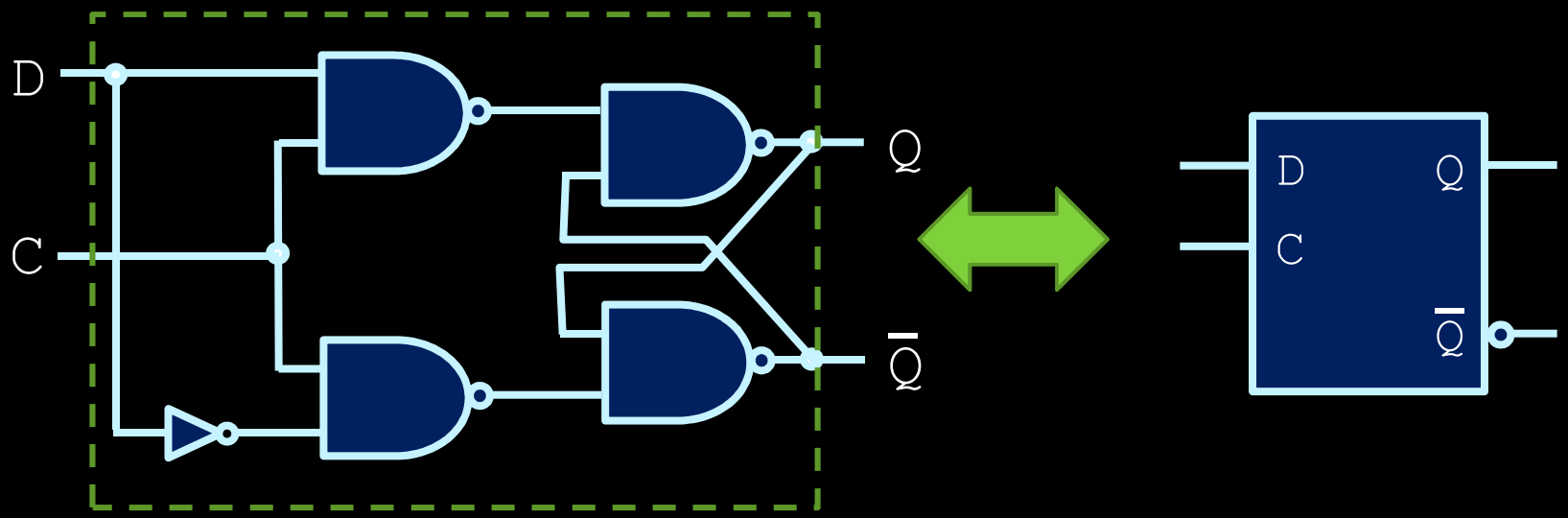


Q_T	D	Q_{T+1}
0	0	0
0	1	1
1	0	0
1	1	1

Note that C=0 acts as the state where S=R=0;
The output Q is same as the previous state

- 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 whenever C is high.

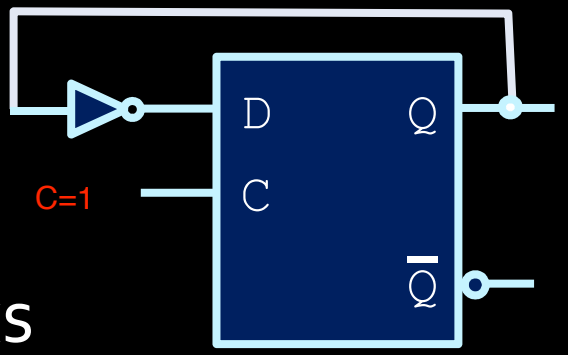
D latch

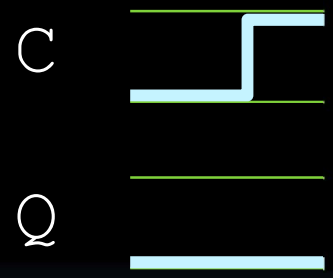


- This design is good, but still has problems.
  - i.e. timing issues.

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.

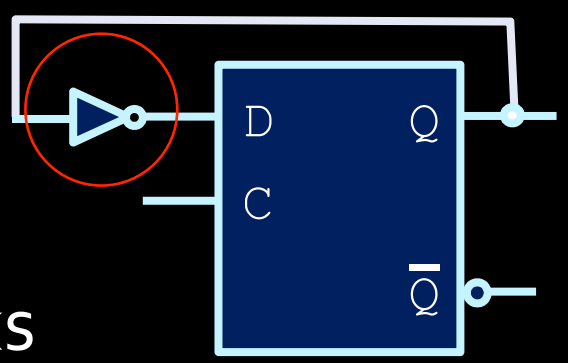




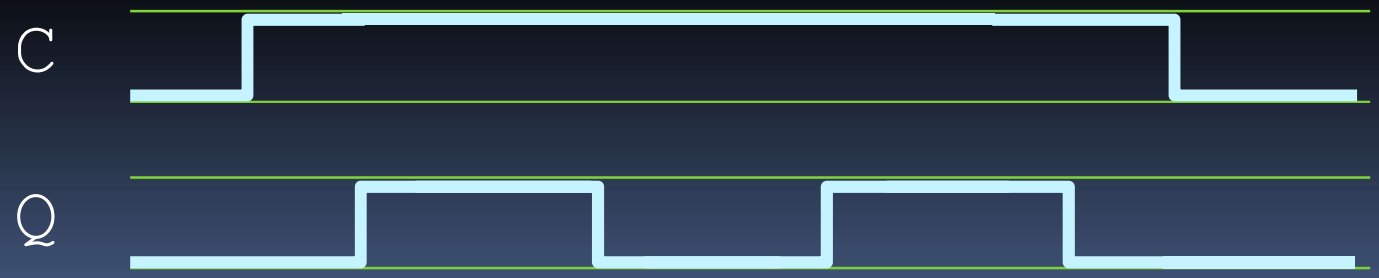
…what happens next?

Latch timing issues

because NOT gate inverts Q



- 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.



instability

D-Latch is transparent!

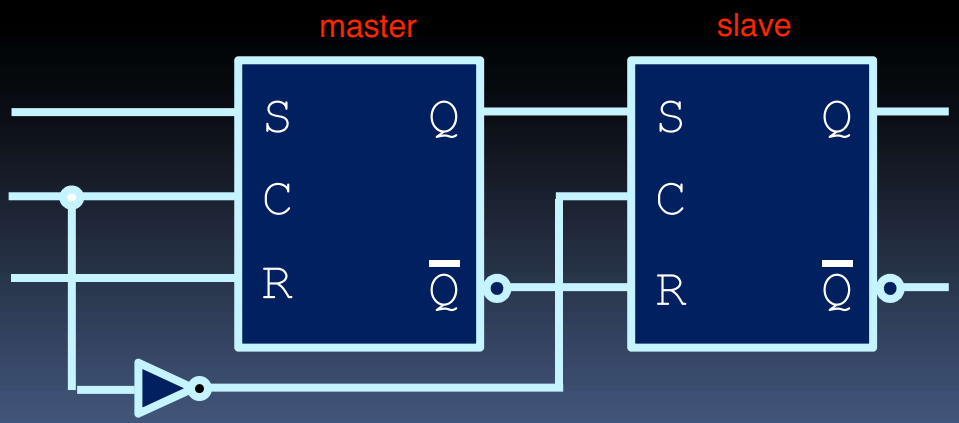
Transparent: input signal changes cause immediate changes in output.
+ i.e. D latch when C=1
Opaque: input signal changes after some trgger
+ i.e. D latch when C=0

- Transparent means that
  - Any changes to its inputs are visible to the output when control signal (Clock) is 1.

- Key Take-away: The "output of a latch **should not** be applied directly or through combinational logic to the input of the same or another latch when they all have the same control (clock) signal."

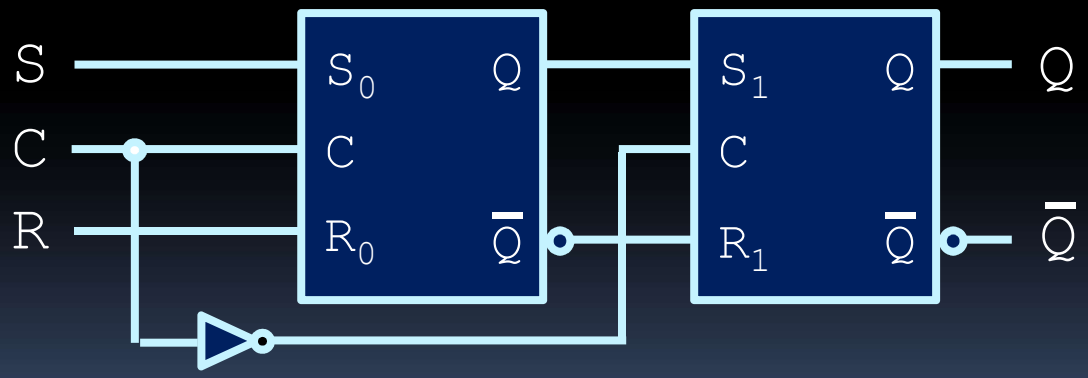
Latch timing issues

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



SR master-slave flip-flop

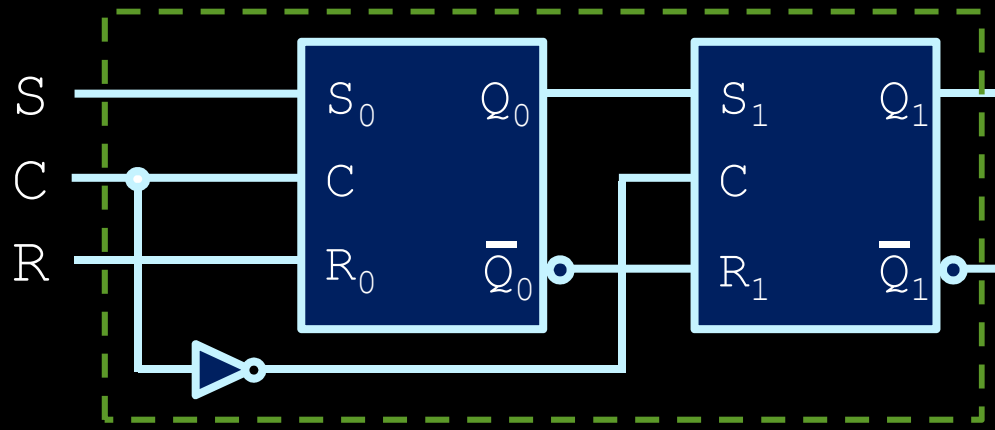
- A flip-flop is a latched circuit whose output is triggered with the rising edge or falling edge of a clock pulse.
- Example: The SR master-slave flip-flop

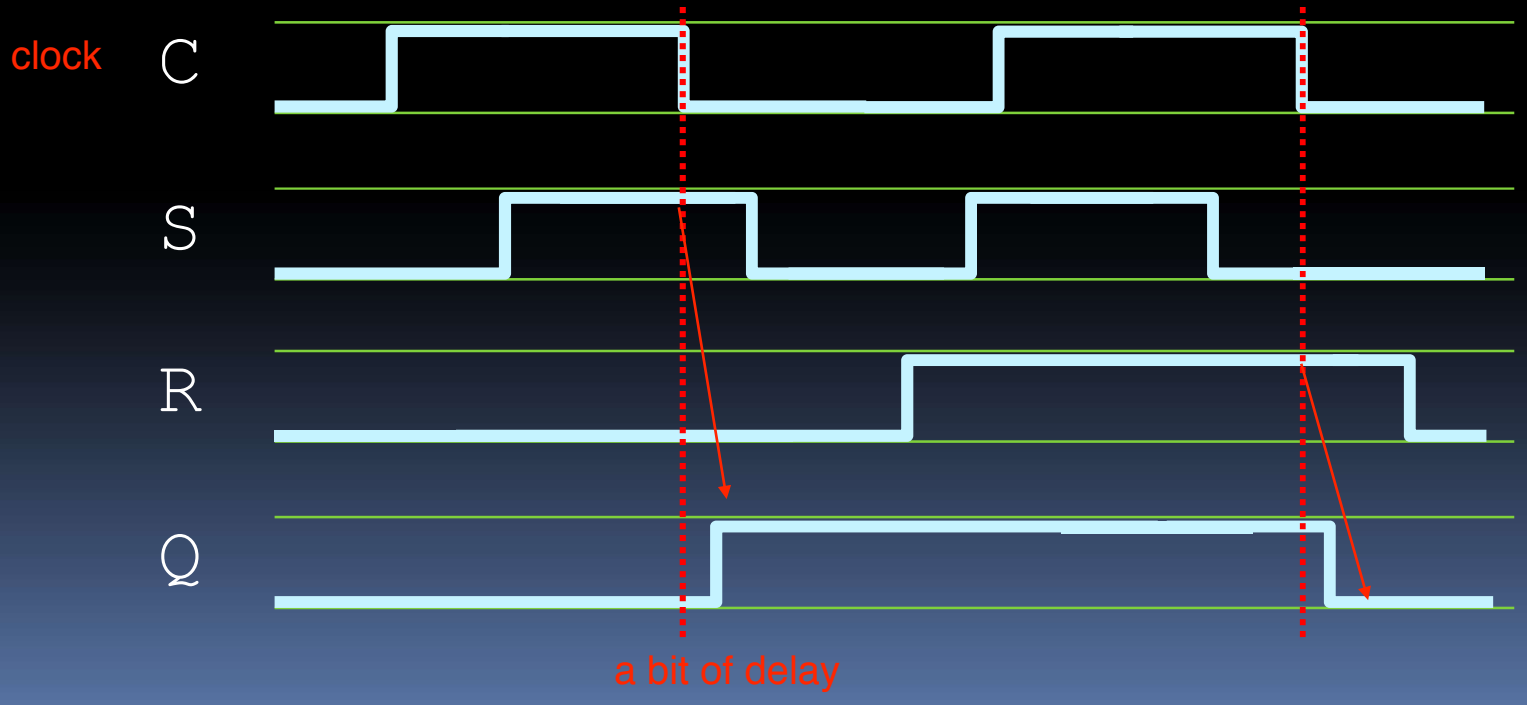


sync at same moment

Q in slave updates during falling edges and takes output from master at that instance

SR master-slave flip-flop



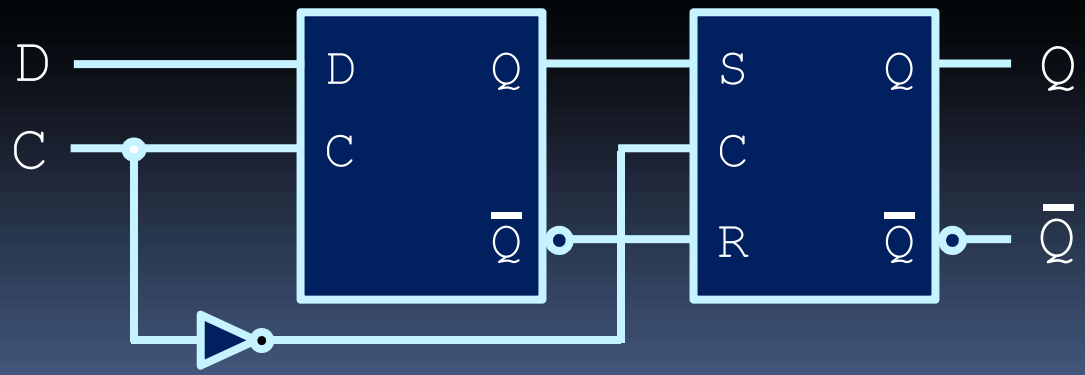


Edge-triggered D flip-flop

unstable when S=1 and R=1 (Since default state is S=0; R=0)

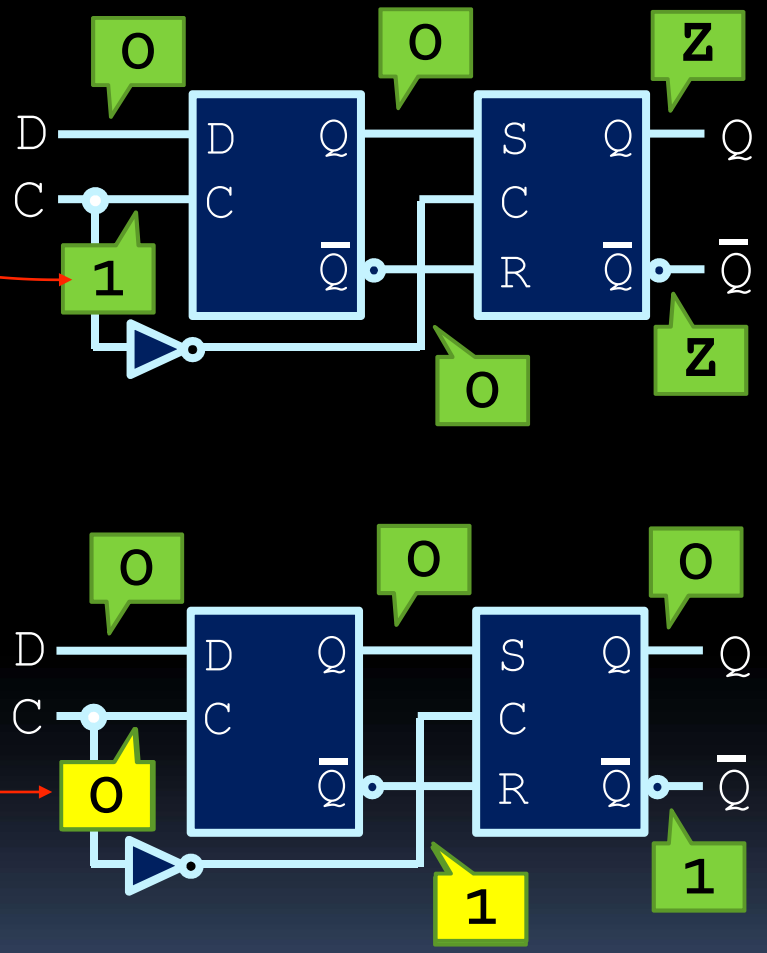
- SR flip-flops still have issues of unstable behaviour.
- Solution: D flip-flop
  - Connect D latch to the input of a SR latch.
  - Negative-edge triggered flip-flop (like the SR)

1. negative-edge: when output is low
2. negative-edges triggered: a circuit or component that changes its state only when an input signal becomes low.



Flip-flop behaviour

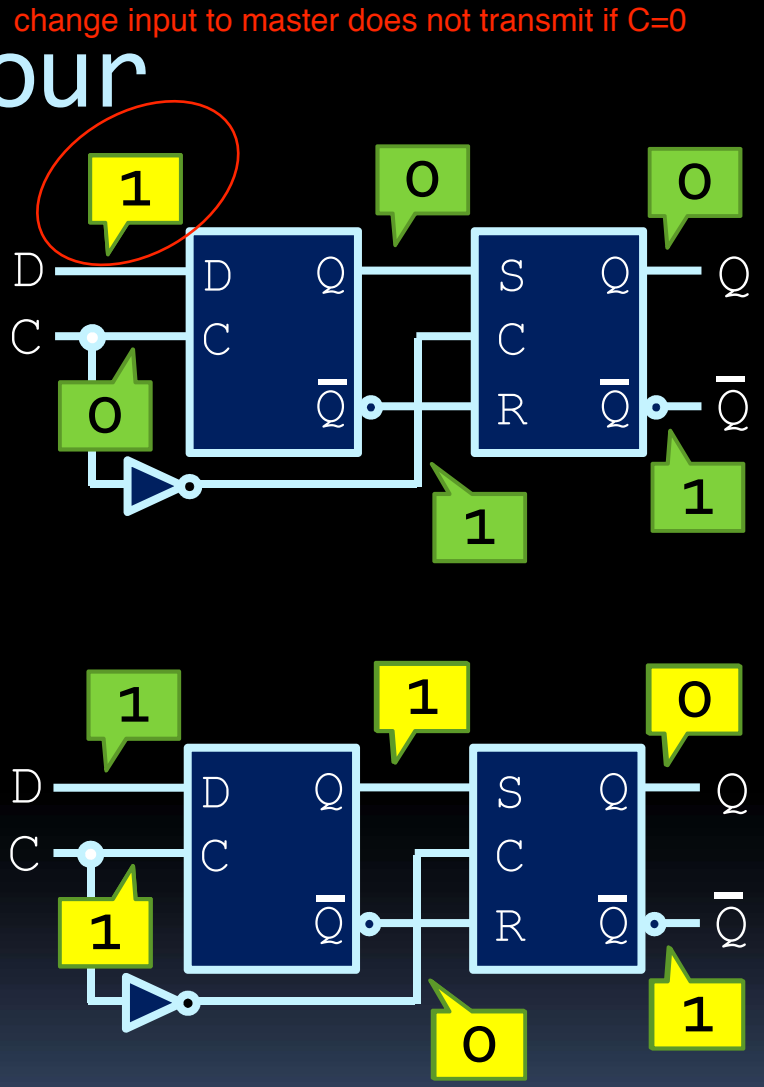
- Observe the behaviour:
  - If the clock signal is high, the input to the first flip-flop is sent out to the second.
  - The second flip-flop doesn't do anything until the clock signal goes down again.
  - When it clock goes from high to low, the first flip-flop stops transmitting a signal, and the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



slave takes last output from master during falling edge when C = 1 -> 0

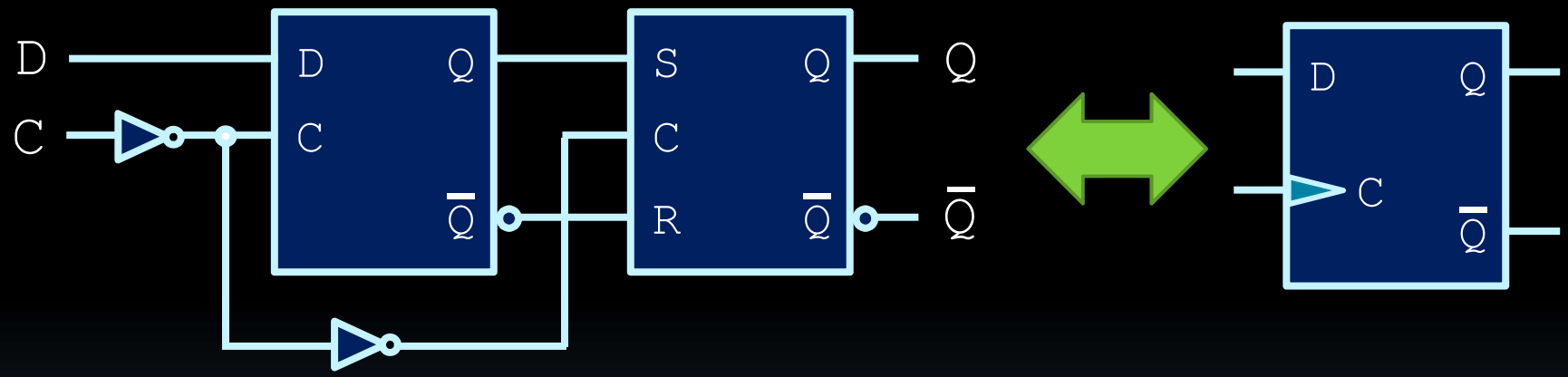
Flip-flop behaviour

- 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

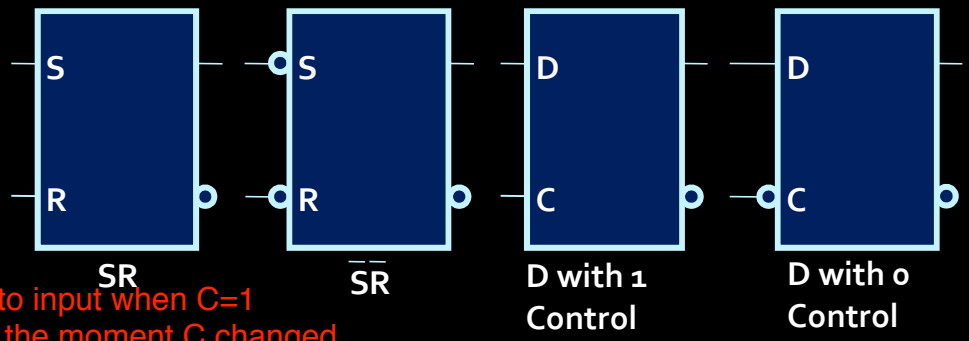


- These are the most commonly-used flip-flop circuits (and our choice for the course).

how memory stored

Notation

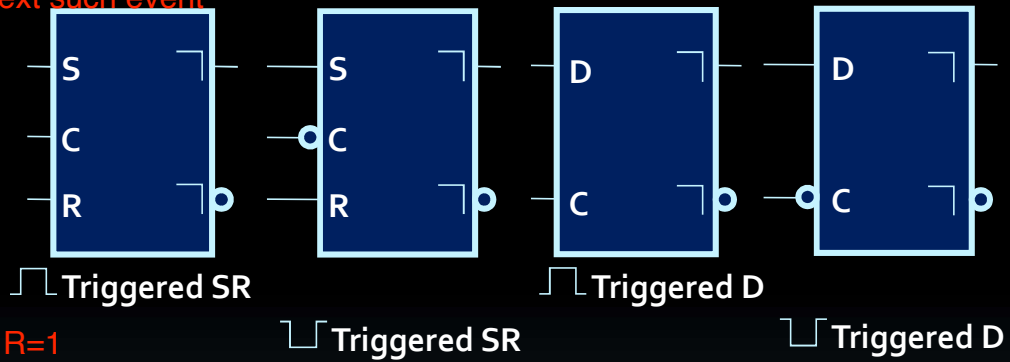
- Latches



Latch are transparent: output respond to input when C=1
Want output to respond to input during the moment C changed (0->1 or 1->0) and stay constant for next such event

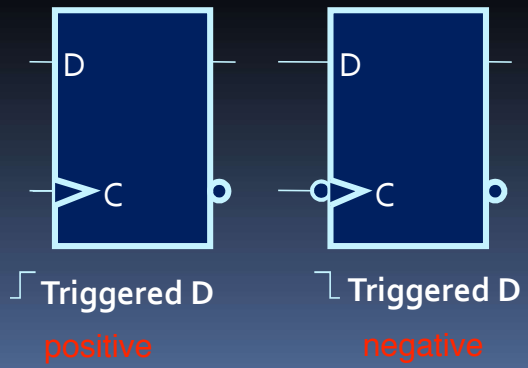
- Master-slave flip-flops

not transparent that is edge triggered



removes unstable state when S=1; R=1

- Edge-triggered flip-flops

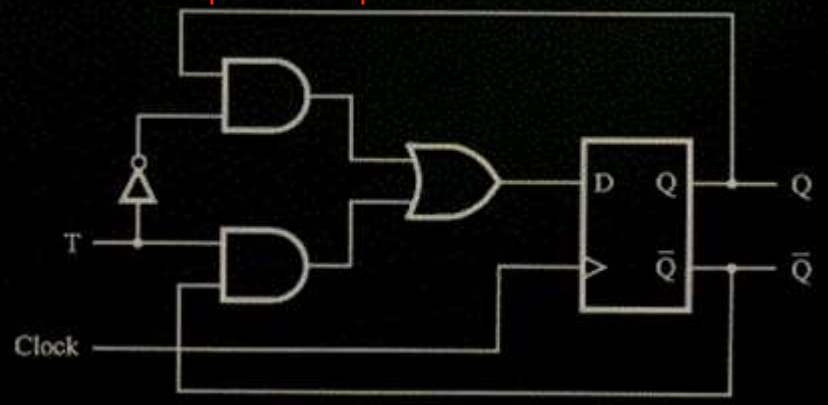


Note: While all these are possible, we mainly use edge-triggered D flip-flops in our designs.

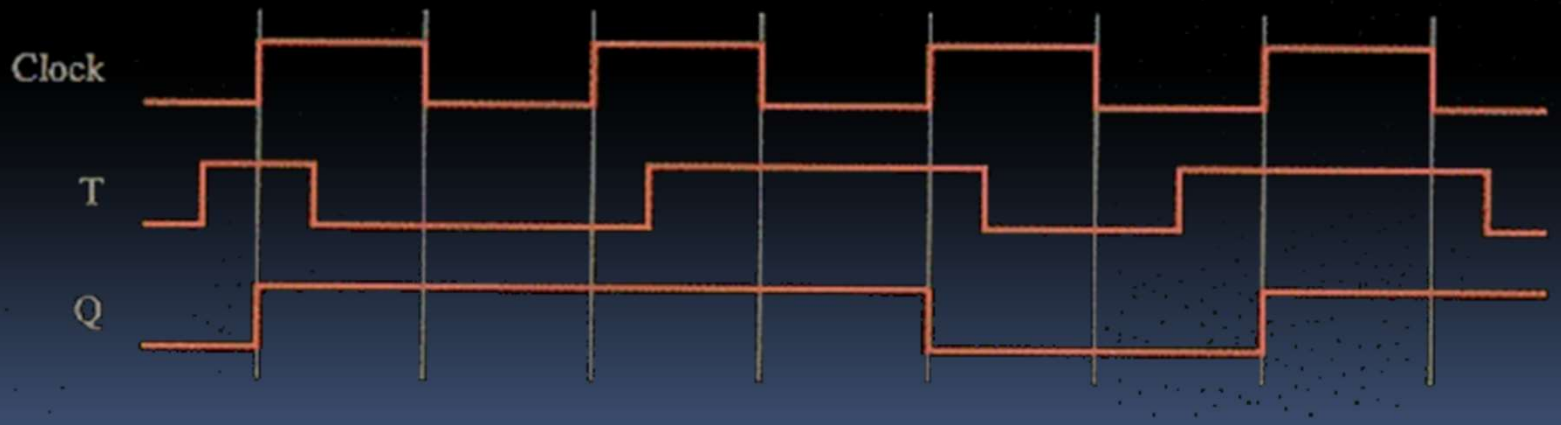
Other Flip-Flops

its like a mux2to1
1. T = 0 —> Q = Q_prev
2. T = 1 —> Q next = ~Q
here switch is T and Q and ~Q are the two possible inputs

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



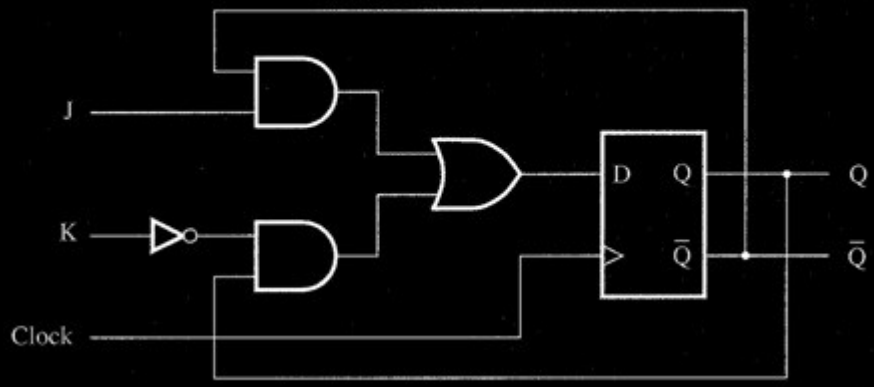
Q next = T & ~Q + ~T & Q

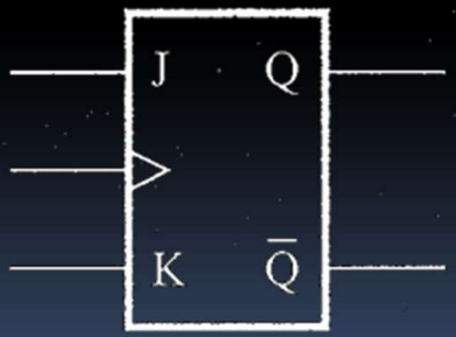


positive edge

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. R
    - if J is 1 and K is 0, set output to 1. S
    - 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 ☺