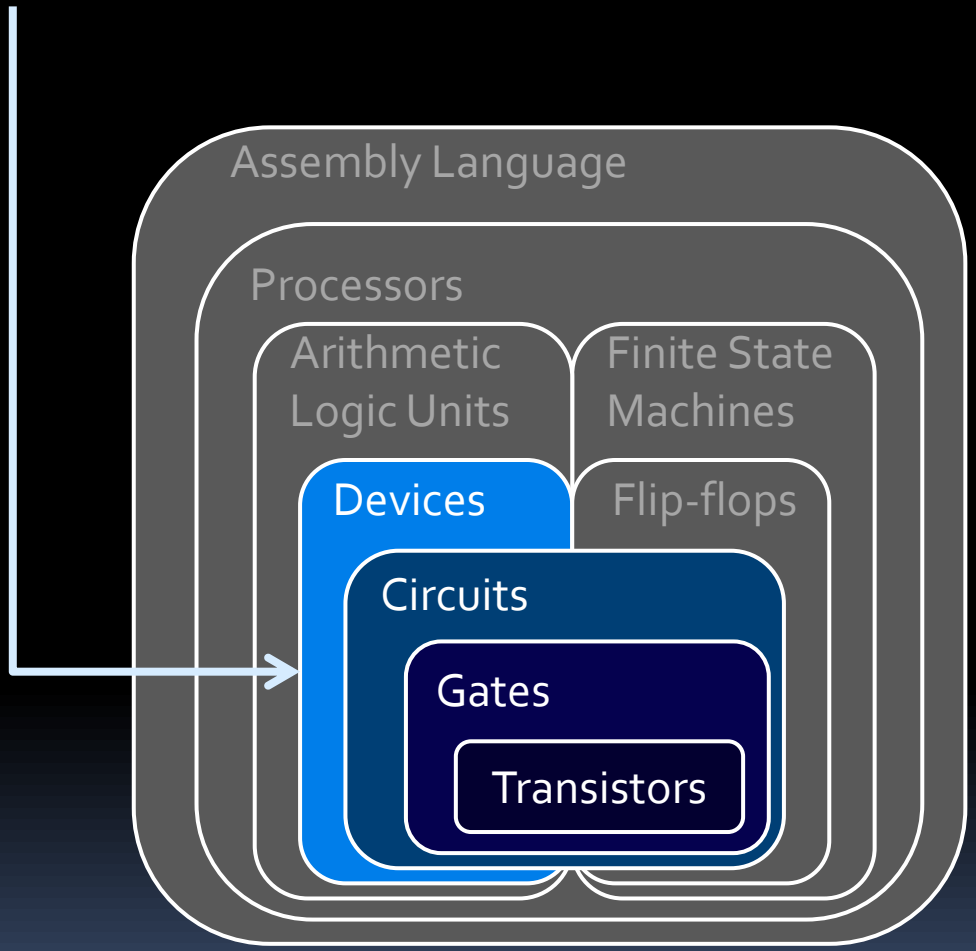
Week 4: Sequential Circuits

Last Week


Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

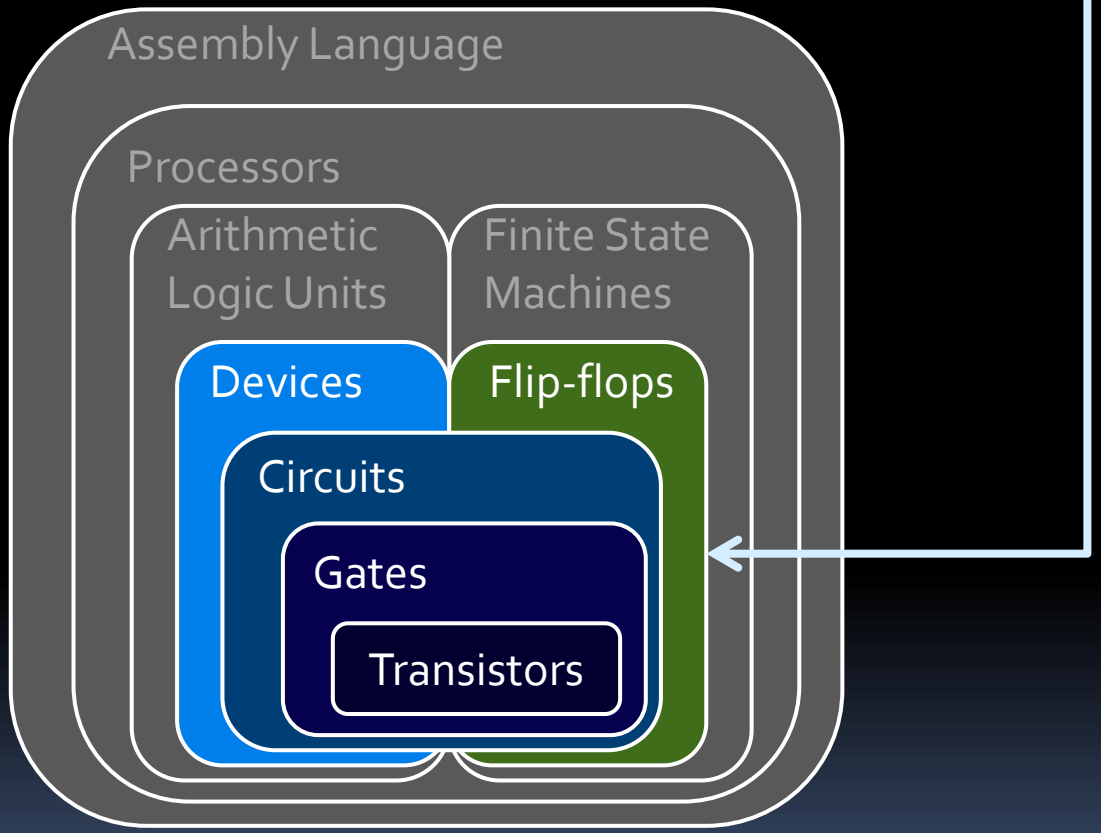
Build A Counter?

- Can you build a device that counts number of key presses?
  - After key is pressed once, hex display should show 1
  - After key is pressed again, show 2
  - Third time: show 3
  - Then reset to 0

Something else to consider…

- Computer specs use terms like "16 GB of RAM" and "5GHz processors".
  - What do these terms mean?
    - RAM = Random Access Memory
    - 16GB = 16 billion `chars` (bytes)
    - 5 GHz = 5 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?

This Week

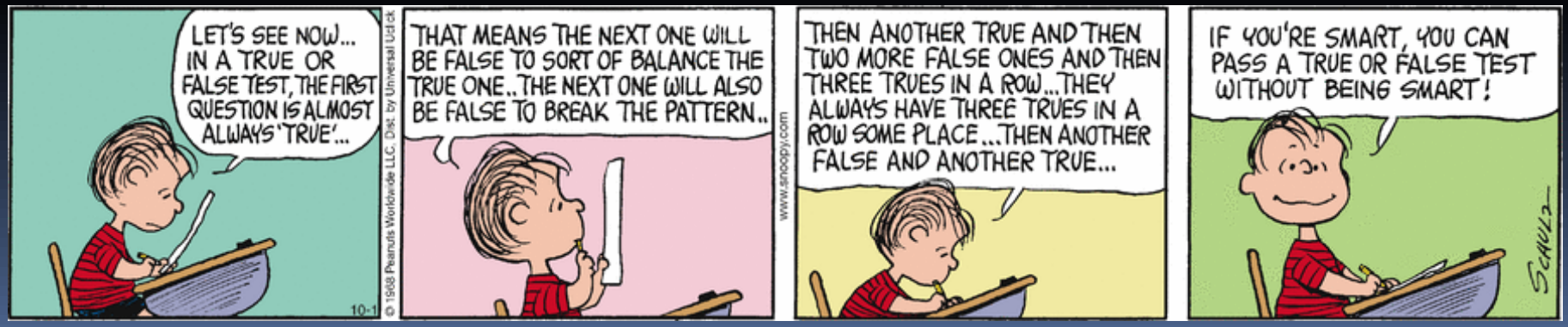

Assembly Language
Processors
Arithmetic Logic Units
Finite State Machines
Devices
Flip-flops
Circuits
Gates
Transistors

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.

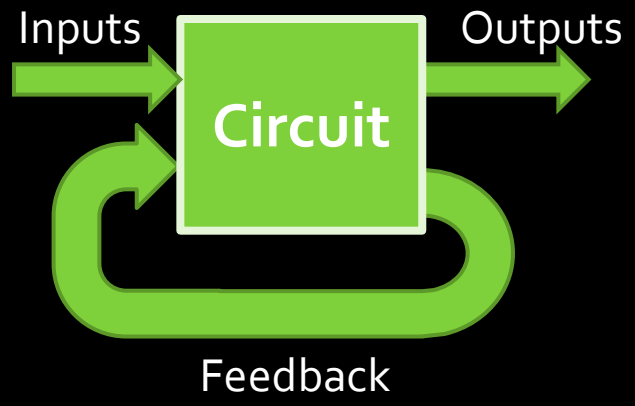
Sequential circuits

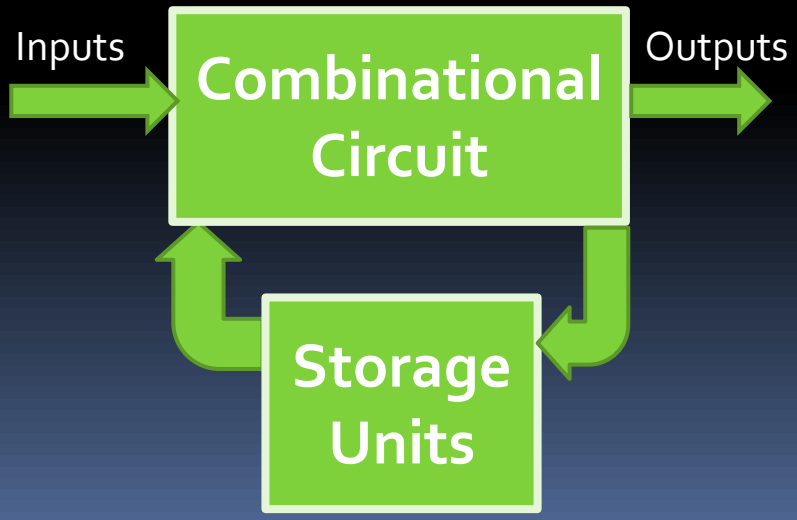
- This creates circuits whose internal state can change over time, where the same input values can result in different outputs.
- Why would we need circuits like this?
  - Memory values
  - Reacting to changing inputs



Creating sequential circuits

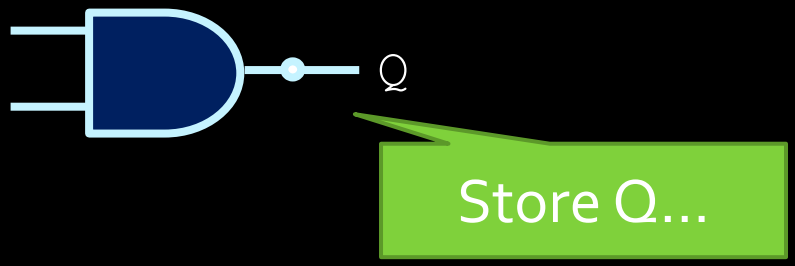
- Essentially, sequential circuits have feedback in the circuit.
  - How is this accomplished?
  - What is the result of having the output of a component or circuit be connected to its input?





Storing and Reusing Values

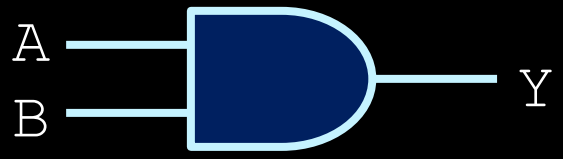
- I want to store the output of an AND gate and reuse it as input.

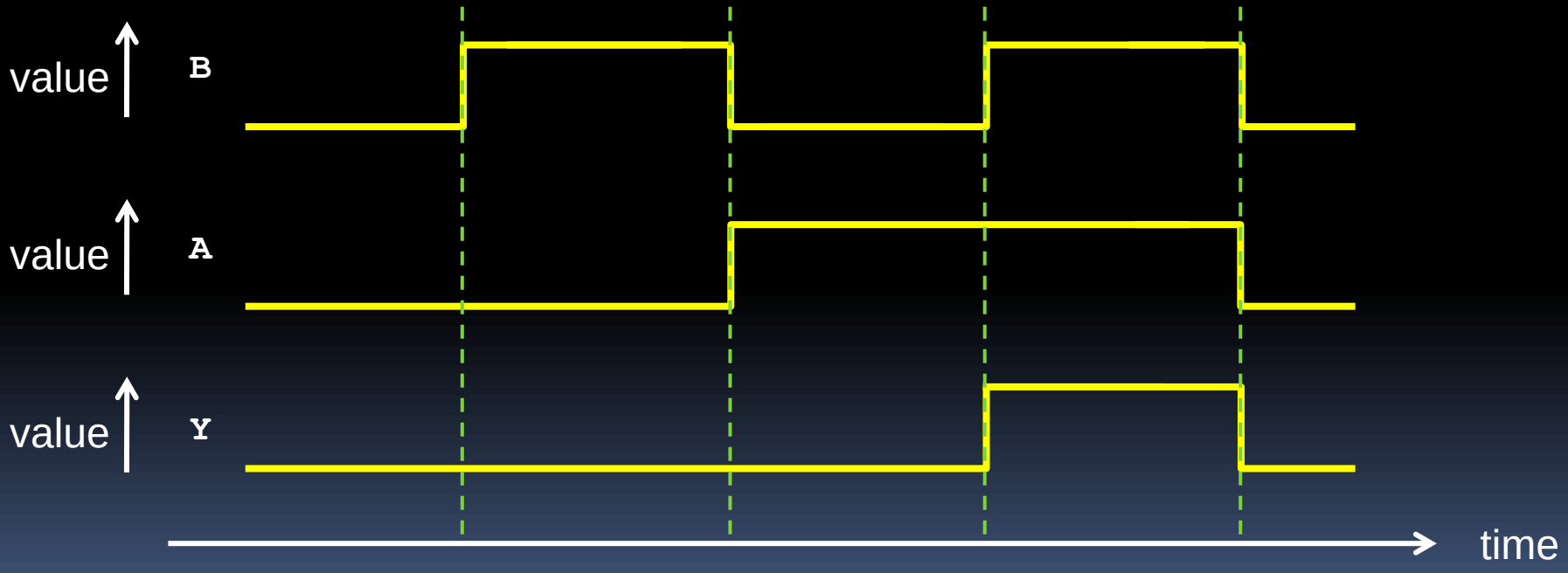


- Does the following work?

- How do we reason about this circuit?

Waveform Diagrams





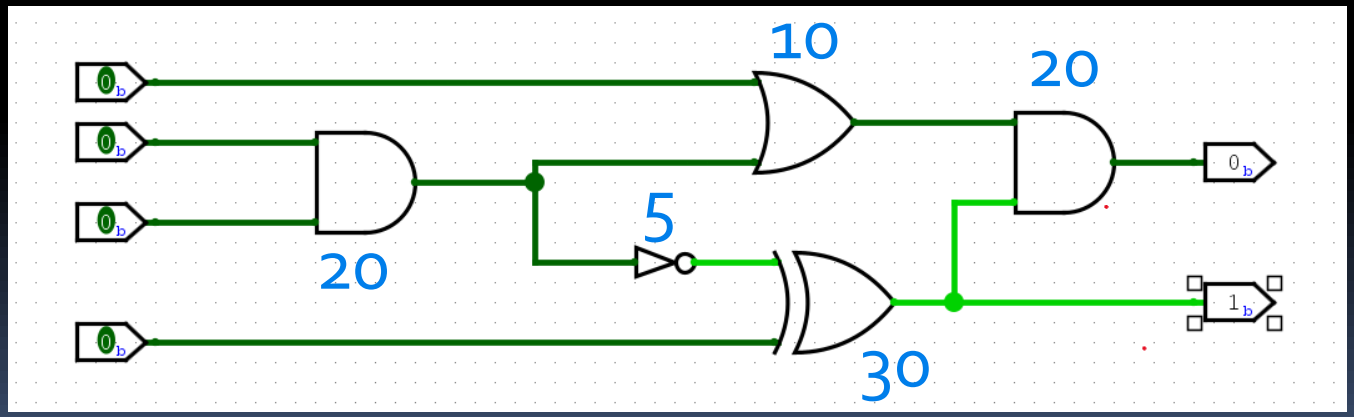
Propagation Delay

- Outputs don't change instantaneously.
  - Electrons have to move, transistors open/close…
  - Even in combinatorial circuits!

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

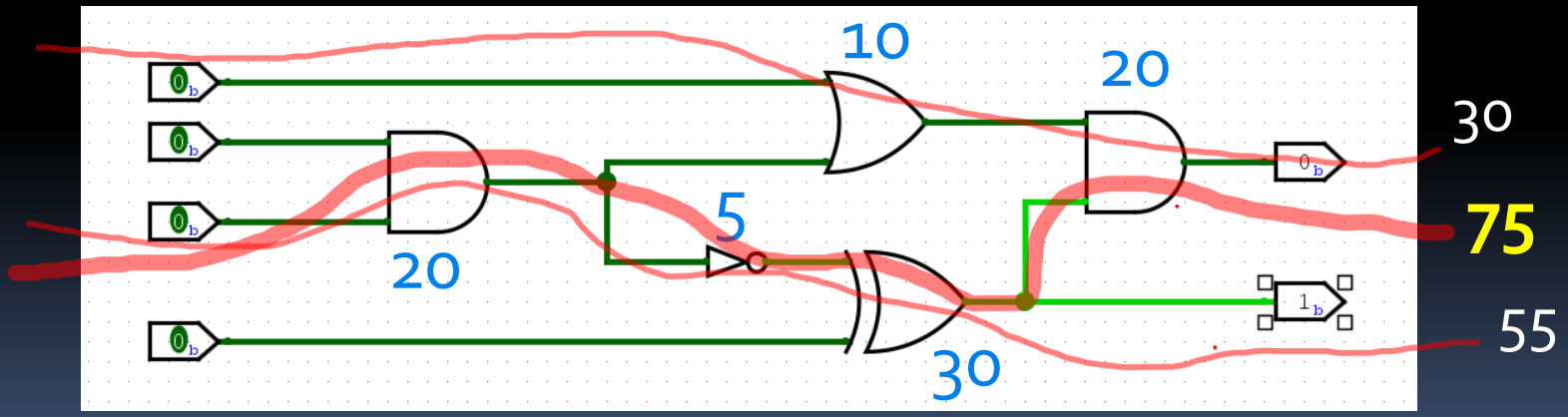
Max propagation delay

- This is the longest possible delay for a circuit.
- Slowest path from any input to any output.
  - In terms of total propagation delay, not path length or number of components on it.

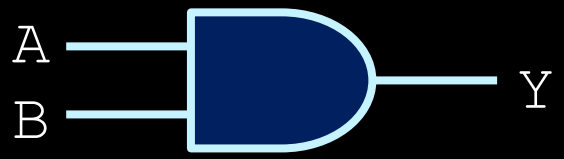


Max propagation delay

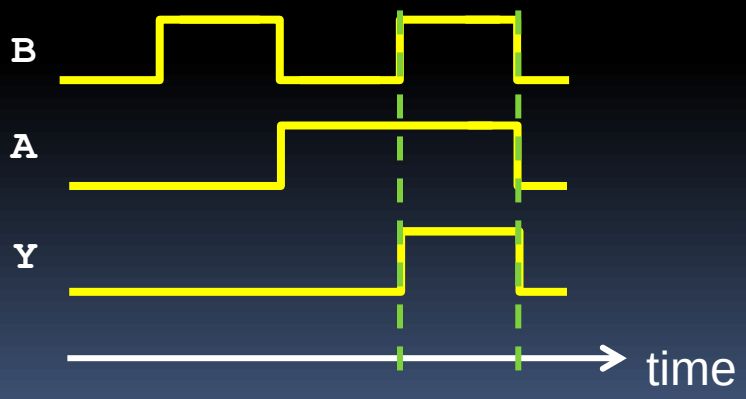
- This is the longest possible delay for a circuit.
- Slowest path from any input to any input.
  - In terms of total propagation delay, not path length or number of components on it.



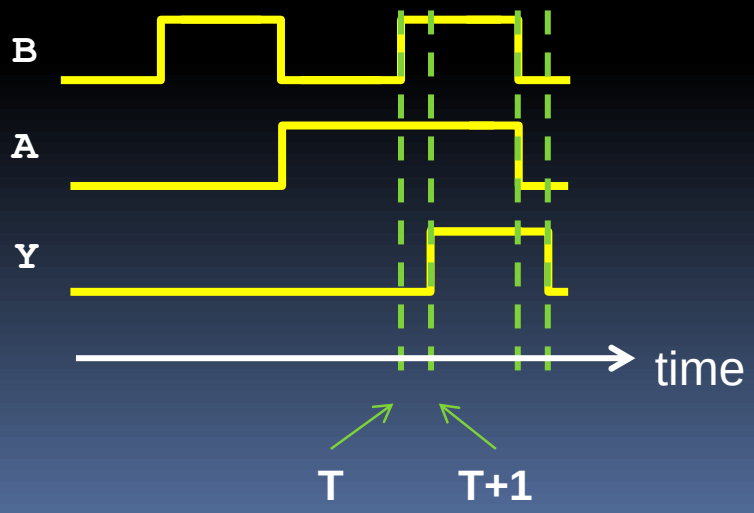
Gate Delays



Ideal

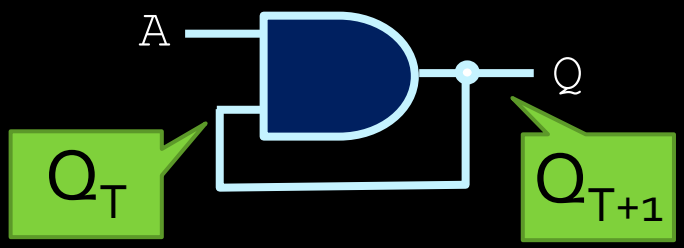


Considering delays



Feedback Circuit Example (AND)

- Let's analyze it

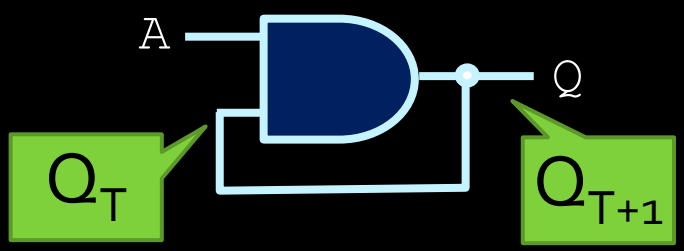


Q_T and Q_{T+1} represent the values of Q at a time T, and a point in time immediately after (T+1)

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

Feedback Circuit Example (AND)

- Let's analyze it



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

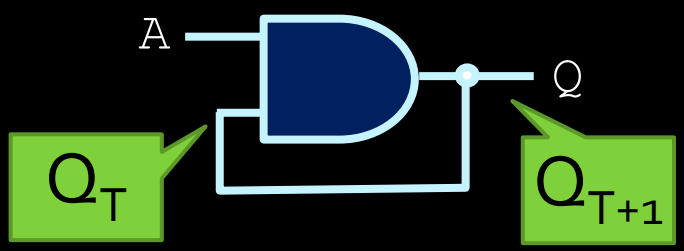
A	Q_T	Q_{T+1}
0	0	0
0	1	0
1	0	0
1	1	1

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

What happens next for later values of A?

Feedback Circuit Example (AND)

- Let's analyze it



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

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.

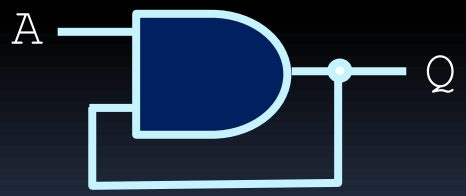
What happens next for later values of A?

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

AND with feedback

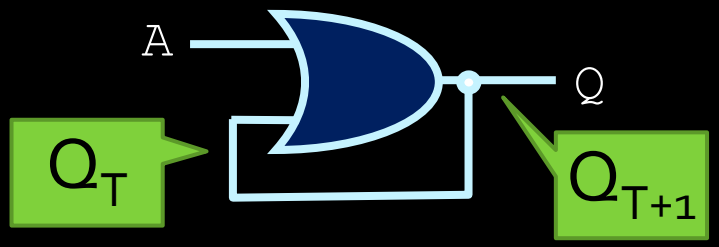
- Some gates don't have useful results when outputs are fed back into inputs.
- Once Q becomes 0, it stays stuck as 0.

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



OR with feedback

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



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)

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

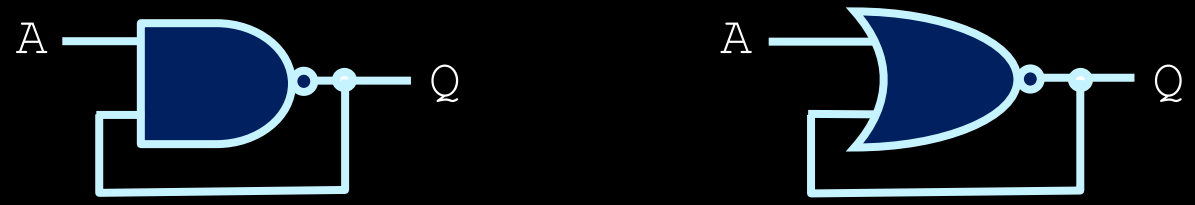
If A=1, Q_{T+1} becomes 1 no matter what Q_T was.

What happens next for later values of A?

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

Feedback Examples (NAND, NOR)

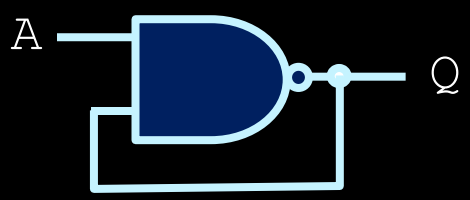
- NAND, NOR gates w/ feedback have more interesting characteristics, which lend themselves to storage devices.



- 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.

Feedback Example (NAND)

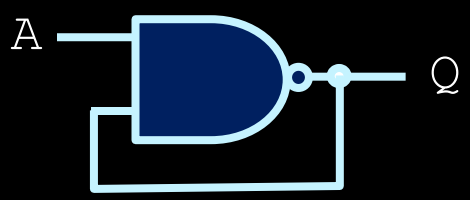
- Let's assume we set A=0
  - Then, output Q will go to 1.
  - If we leave A unchanged we can store 1 indefinitely!



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

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!

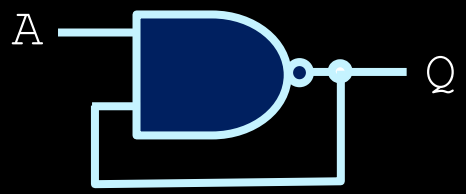


A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

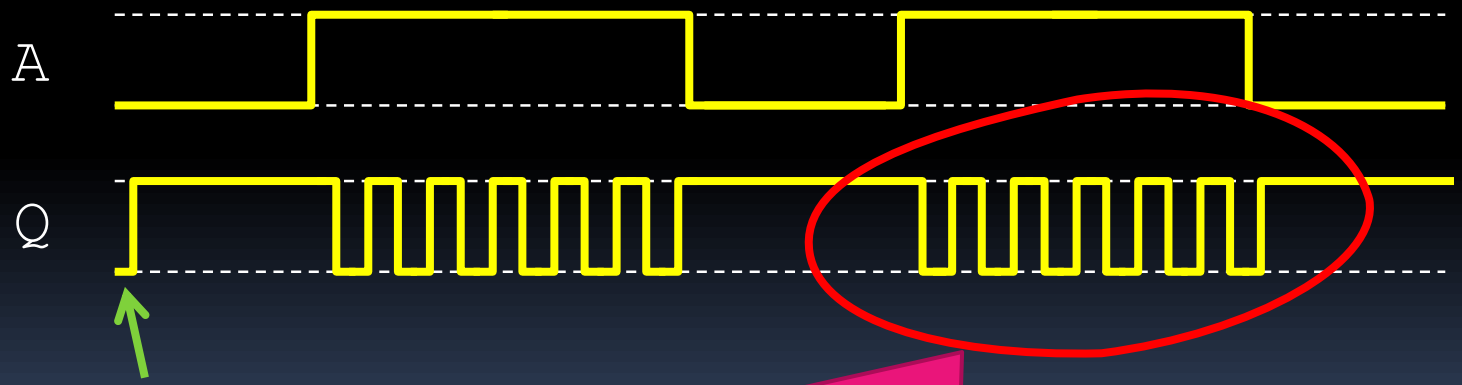
What happens in these last two scenarios?

Unsteady state! Can't store 0 long!

NAND waveform behaviour



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

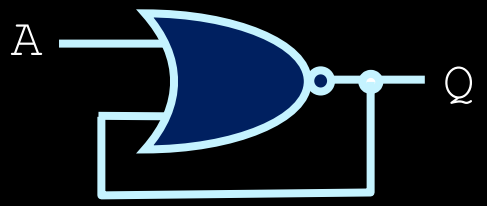


Gate delay. Output does not change instantaneously

This is known as **oscillation**.
The circuit oscillates between high and low.

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

Feedback behaviour

- NAND behaviour

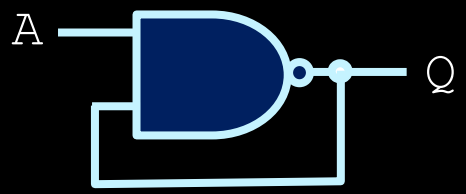
A	Q_T	Q_{T+1}
0	0	1
0	1	1
1	0	1
1	1	0

- 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 enter an unstable state.
  - The output is not stable – it oscillates.

NAND waveform behaviour



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

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.

Unstable Storage

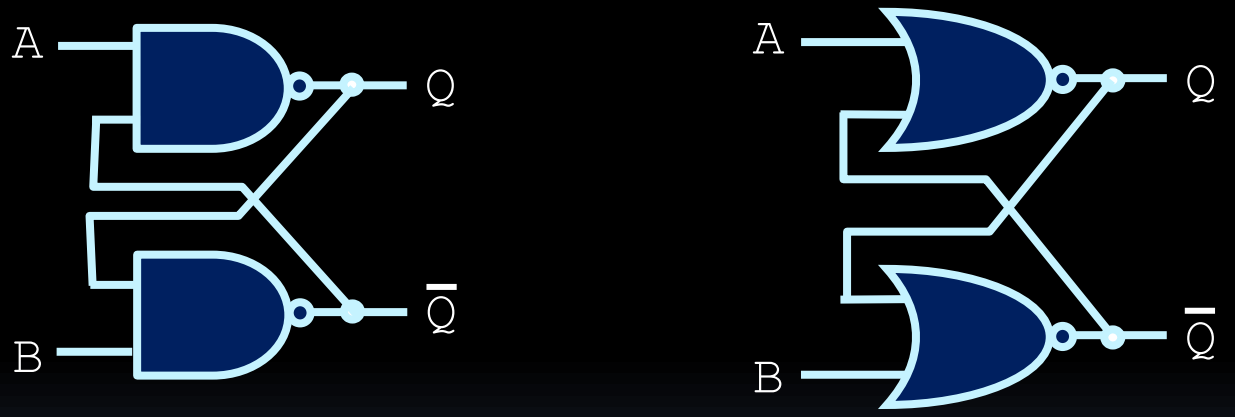
- We almost have storage.
  - But we can't have this unstable feedback.
- Need to be able to set high or reset to low whenever we want, and maintain these.

The Quest for Storage

- We need a feedback circuit with stable behaviour:
  - We should be able to set to `0` or to `1`
  - Circuit should maintain state for long-term.
  - No instability.

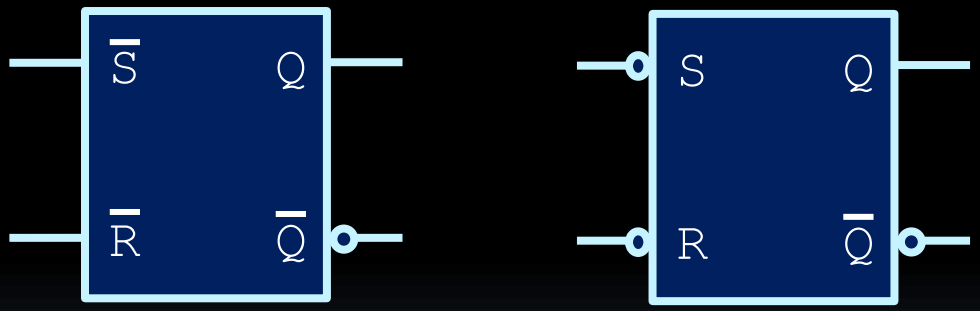
Latches

- Combine multiple NAND or NOR gates to get stable behaviour.

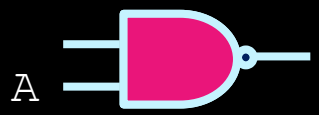


- These circuits are called latches.

- The $\bar{S}\bar{R}$ Latch
  - …or S′R′ or ~S~R and so on



Quick Reminder

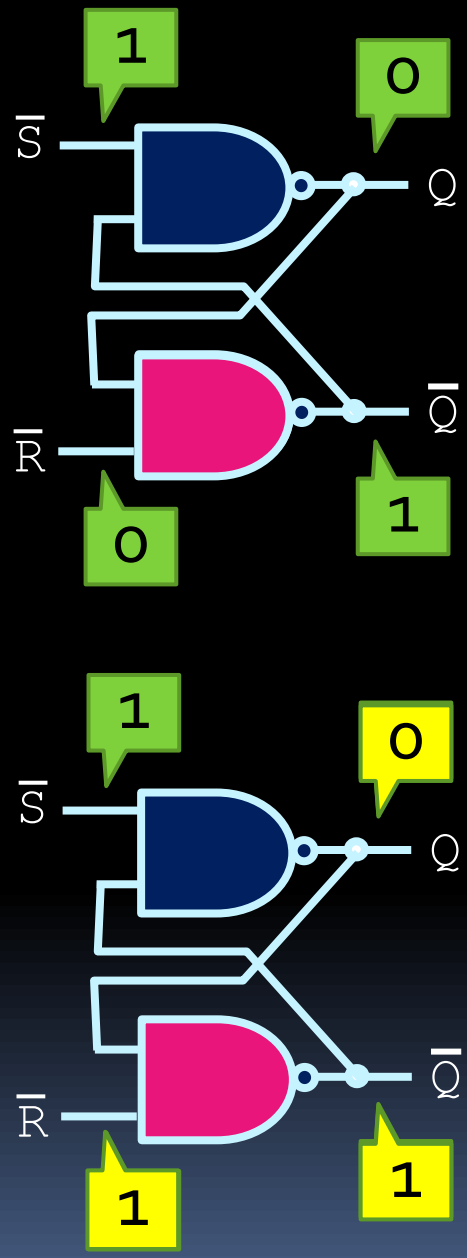


Knowing A = 0, what can you say about the output of this NAND gate?

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

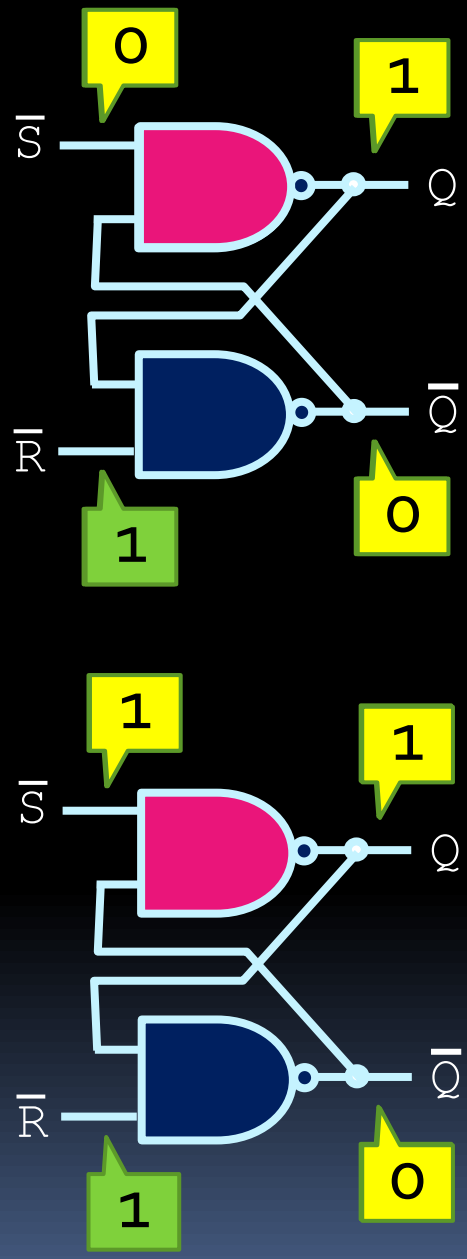
$\overline{S}\overline{R}$ latch: case 1

- 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.



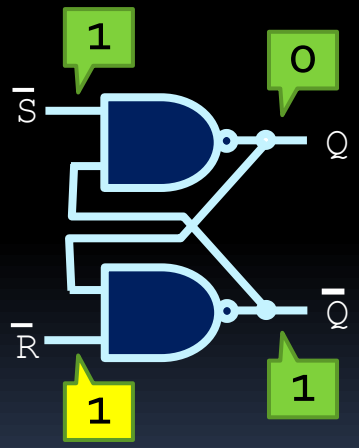
$\overline{S}\overline{R}$ latch: case 2

- (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.

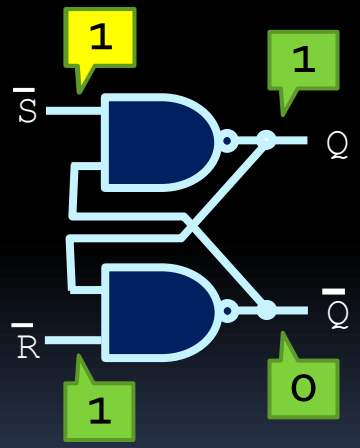


Conclusion

- The input 11 maintains the previous output state!
- Same input, different outputs.
- It's the sequence of inputs that matters.

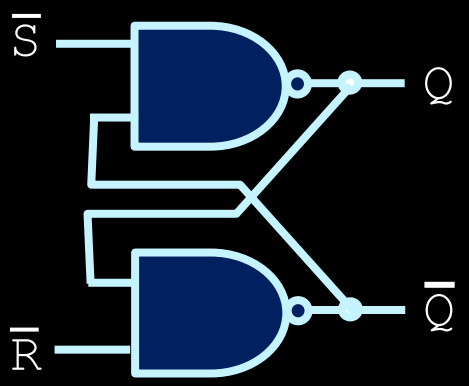


Case 1



Case 2

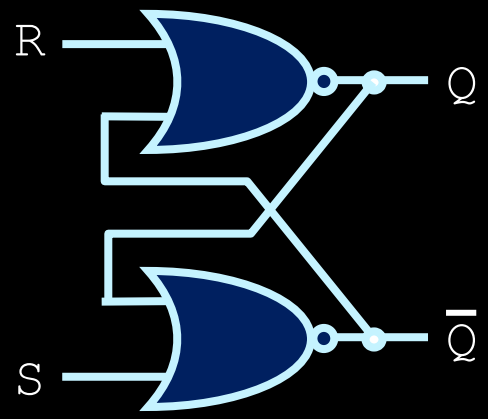
$\overline{S}\overline{R}$ latch



$\overline{S}$	$\overline{R}$	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.
- Note how the circuit "remembers" its signal when going from 10 or 01 to 11.
- Going from 00 to 11 produces a race condition:
  - We don't know what the output will be. It can be 0 or 1, depending on which input changes first. This causes unstable behaviour.

SR latch

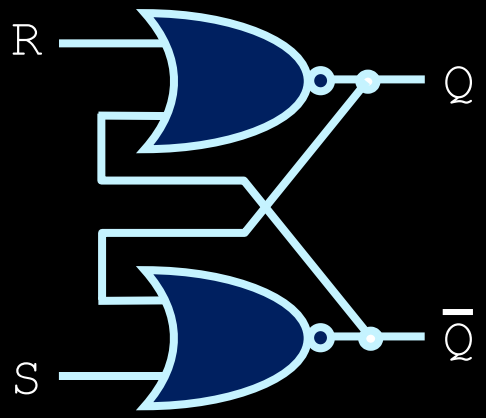


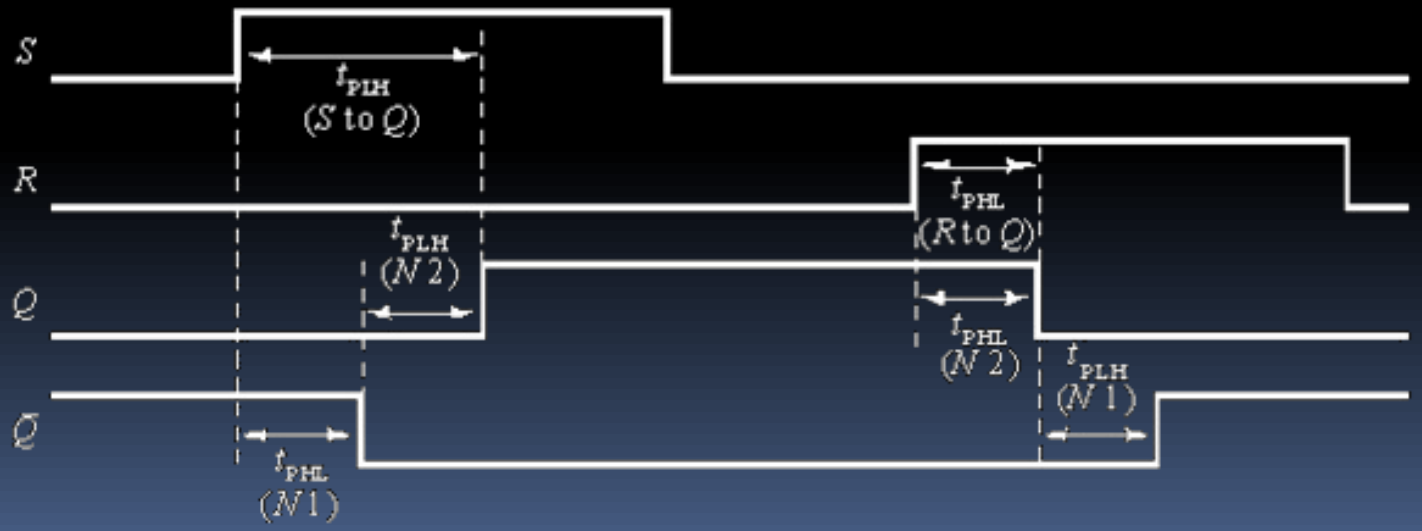
S	R	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$
0	0	0	1	0	1
0	0	1	0	1	0
0	1	X	X	0	1
1	0	X	X	1	0
1	1	X	X	0	0

- 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.

SR latch timing diagram

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





Summary: S'R' and SR latches

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

$\overline{S}$	$\overline{R}$	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$	
0	0	X	X	1	1	Forbidden state
0	1	X	X	1	0	Set Q to 1
1	0	X	X	0	1	Reset Q to 0
1	1	0	1	0	1	Maintain Q
1	1	1	0	1	0	

SR-latch

S	R	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$	
0	0	0	1	0	1	Maintain Q
0	0	1	0	1	0	
0	1	X	X	0	1	Reset Q to 0
1	0	X	X	1	0	Set Q to 1
1	1	X	X	0	0	Forbidden state

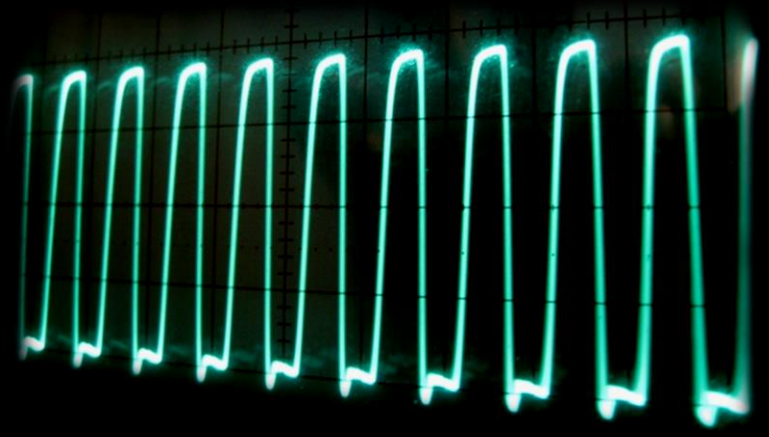
"set" and "reset" cannot be both true!

Reading from latches

- We now have circuits that can store values.
- But how do we distinguish:
  - "5 highs in a row" vs "10 highs in a row"
  - "001100" vs "010"
- Impose some frequency for reading the signal?
  - But when can we sample (read) the signal?
- Need some sort of timing signal, to let the circuit know when the output may be sampled.

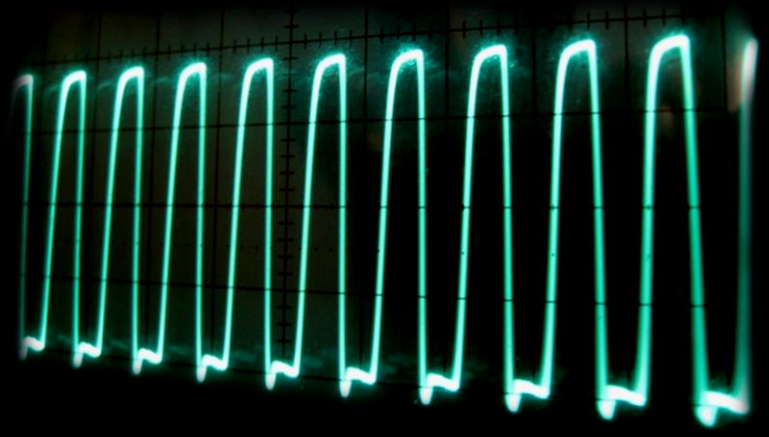
  → clock signals.

Why A Clock

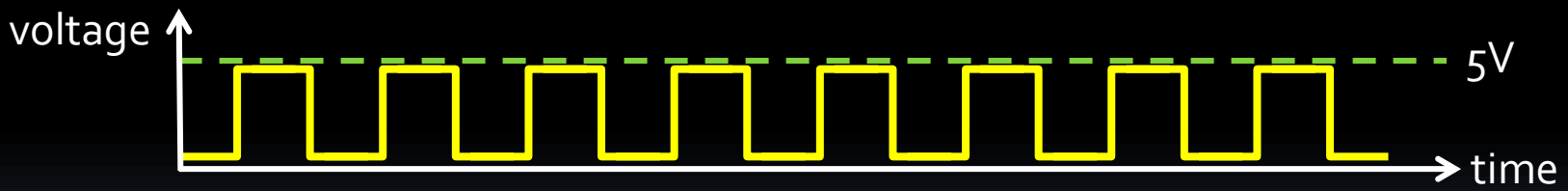


- How do we distinguish between:
  - "high 5 times in a row"
  - "high 10 times in a row"
- What do "5 times" or "10 times" mean?
- Need a way to tell when a signal (input or output) is "ready" to be sampled

Clock Signals



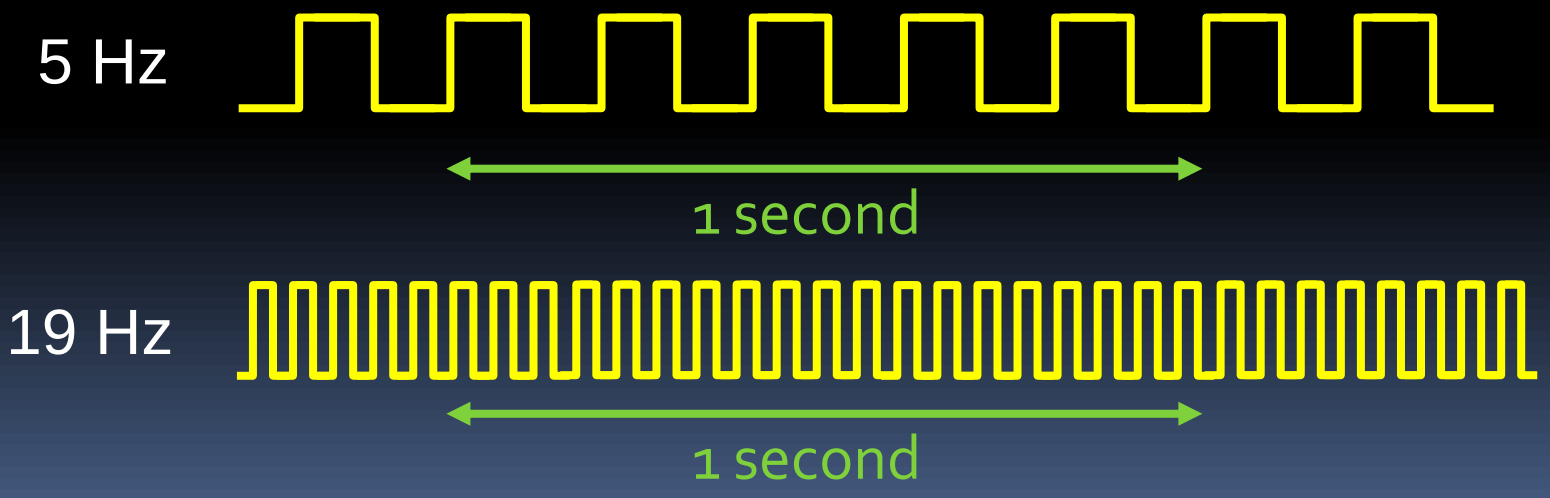
- Clock: a regular signal, where high value indicates that the output of the latch may be sampled.
- Usually drawn as:



- But looks more like:

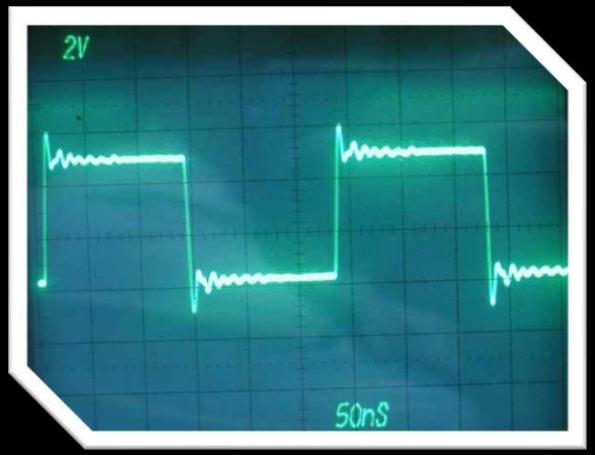
Clock Frequency

- Frequency = the number of pulses occur per second.
  - measured in Hertz (Hz).
- Higher frequency → can do more every second.



Signal Restrictions

- What limits the clock frequency?
  - Circuit complexity
    - Max propagation delay!
  - Physical limits
    - Latency of transistors
  - Manufacturing variations
  - Physical clock limits
    - Gibbs phenomenon
    - Jitter

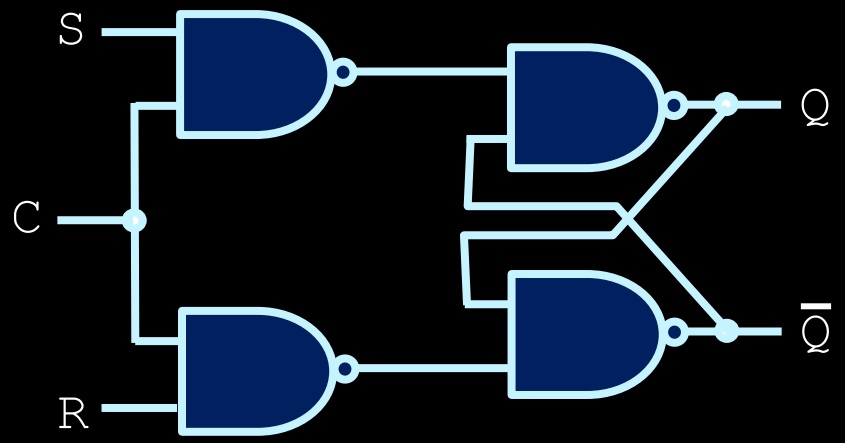


Beyond the scope of CSCB58

Signal Restrictions

- CPU makers try to increase it every year.
- 40 years ago: 1 Mhz
  - Apple II – MOS 6502
  - C64 – MOS 6510
- Today: 3-5 GHz
- x 5000 increase in 40 years!

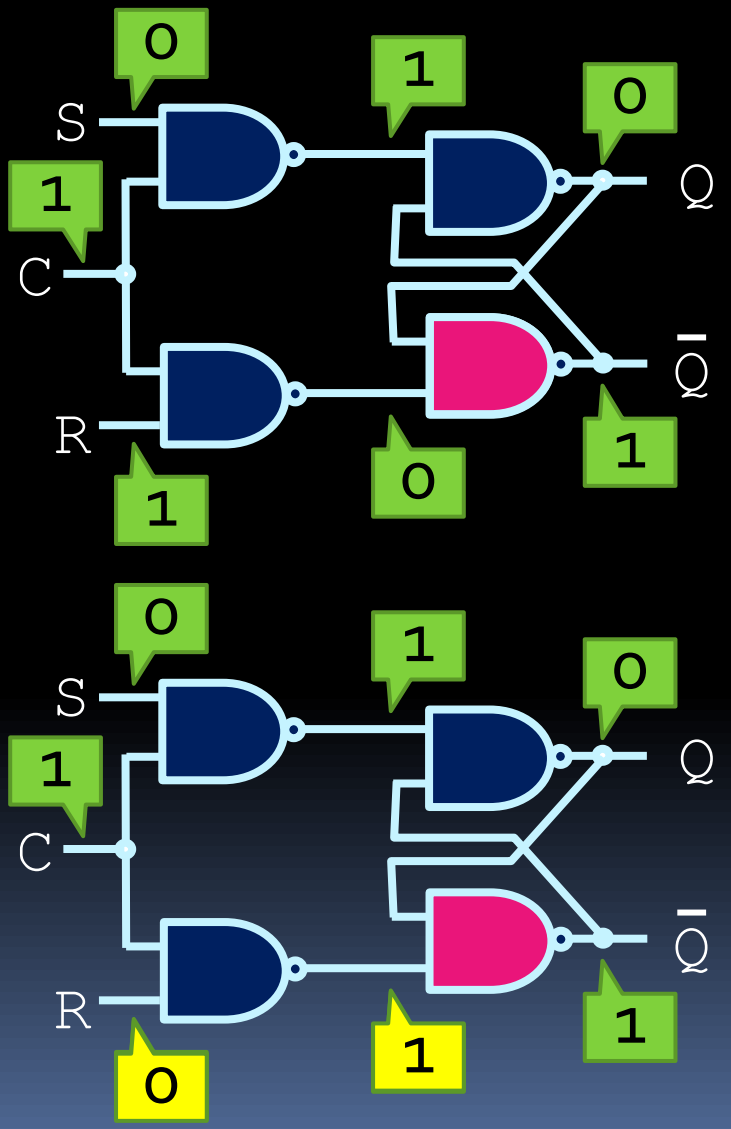
Clocked SR latch



- 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 clock signal

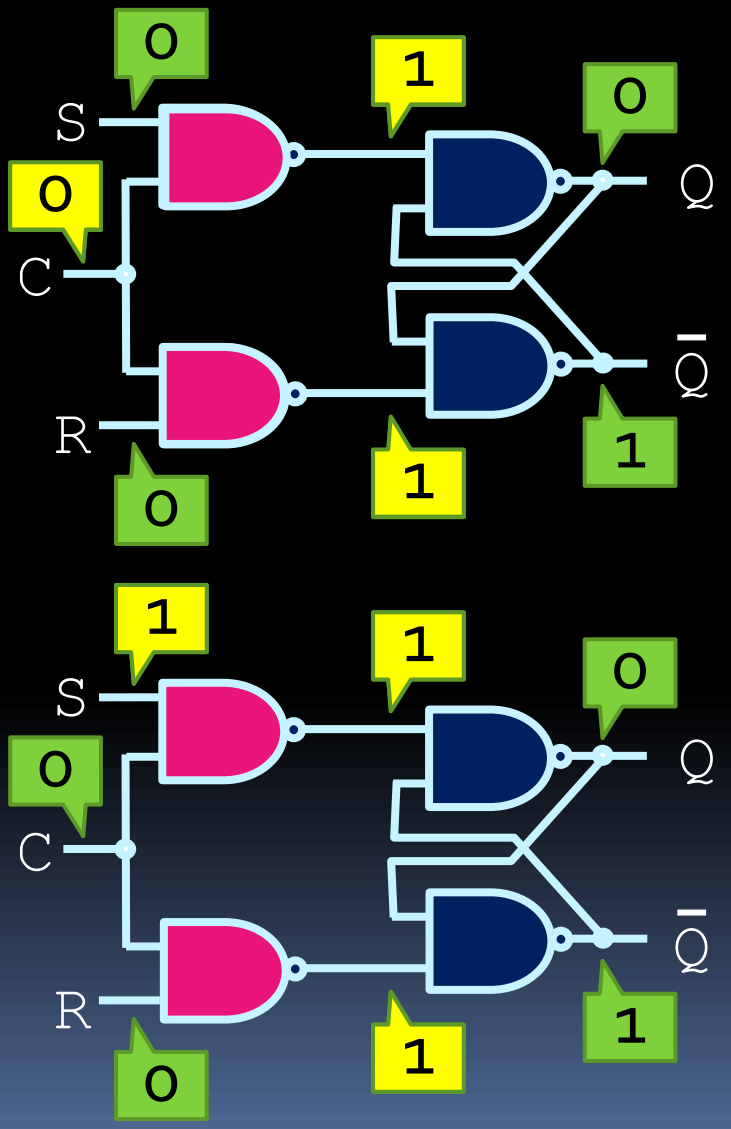
Clocked SR latch behaviour

- 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.

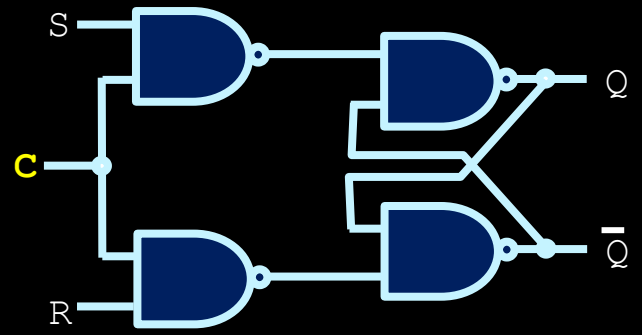


Clocked SR latch behaviour

- 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.



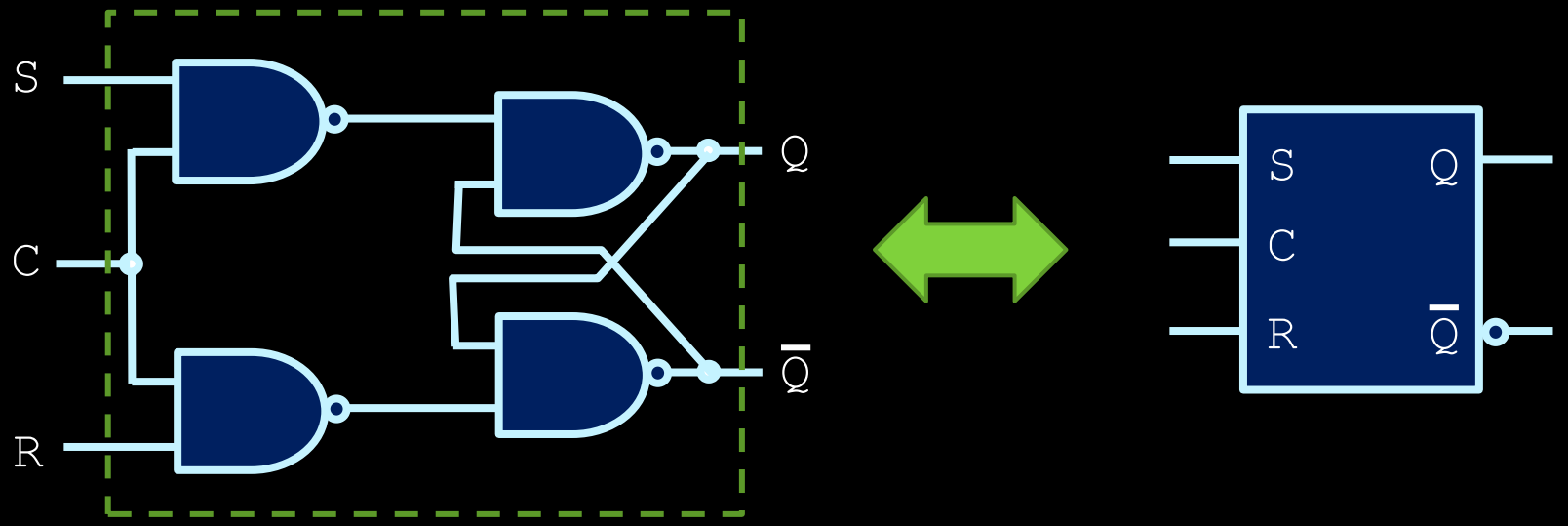
Clocked SR latch - summary



The S and R signals are only allowed to affect the circuit when the clock input (C) is high.

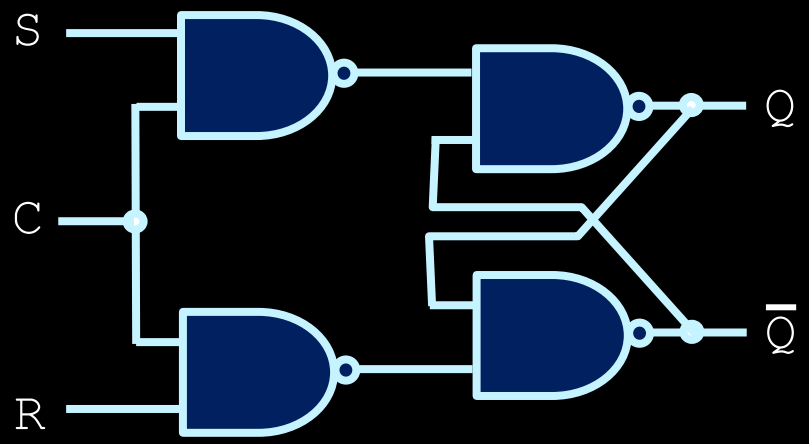
- When clock is high, behave like a SR latch.
- When clock is low, S and R are blocked and there is no way to change the output.

Clocked SR latch



- This is the typical symbol for a clocked SR latch.
- Note: the small NOT circle after the Q output is simply the notation to use to denote the inverted output value. It's not an extra NOT gate.

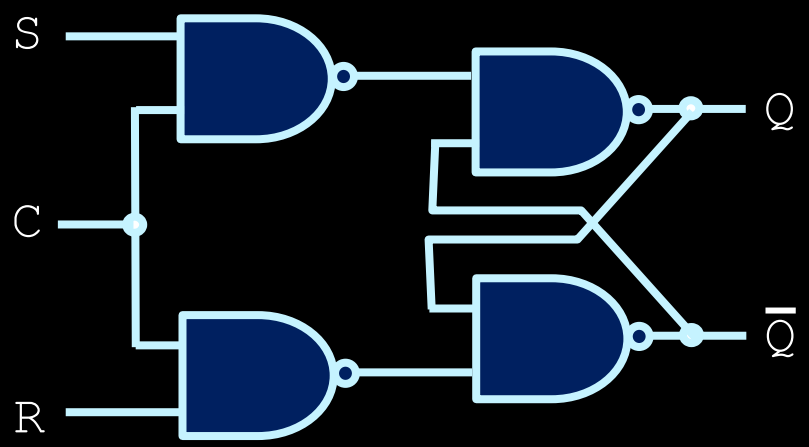
Clocked SR latch behaviour



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	?	???

- Wait!
- Where's the clock?
- There's a better way to look at this….

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~~

- Assuming the clock is 1, we still have a problem when S and R are both 1, since the state of Q is indeterminate.
- A better design would prevent S and R from both going high at the same time.