



# Sequential Circuits

# Something to consider...

- Computer specs use terms like “8 GB of RAM” and “2.2GHz 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?

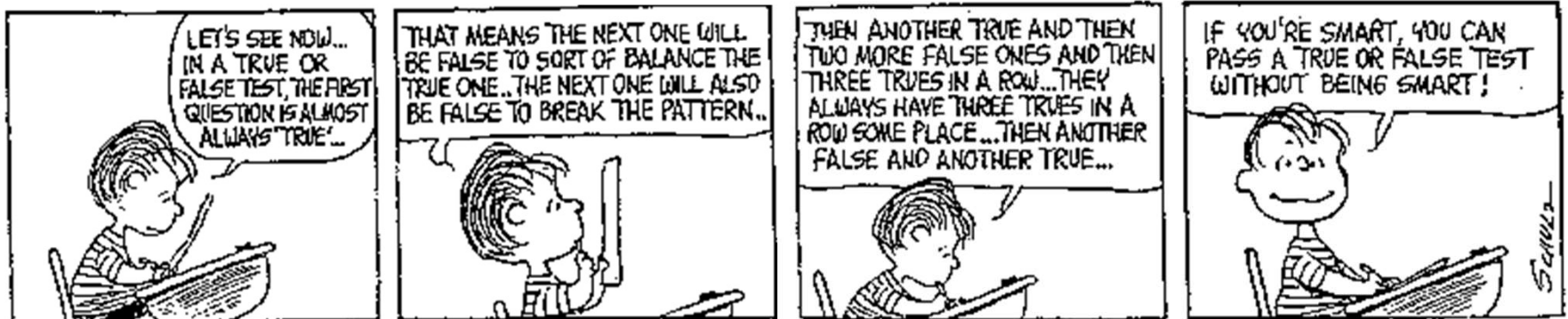


# Two kinds of circuits

- 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.

# Sequential circuits

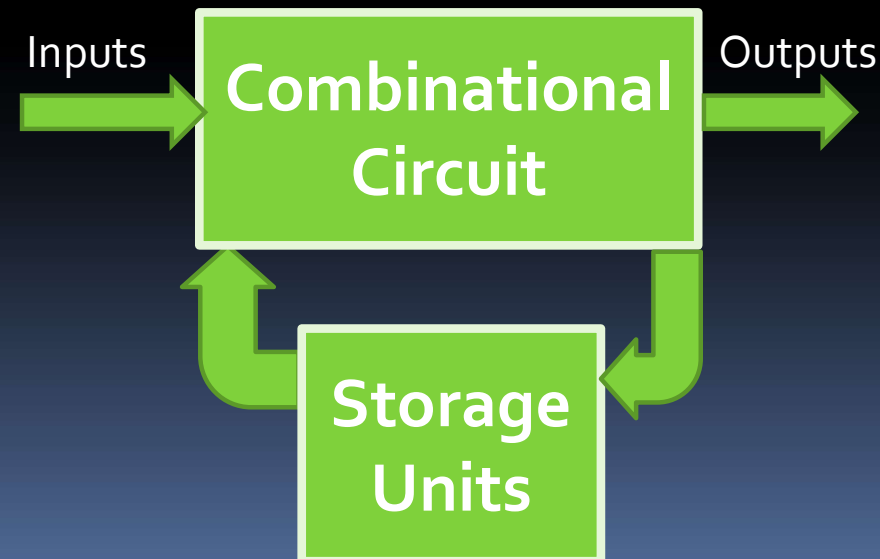
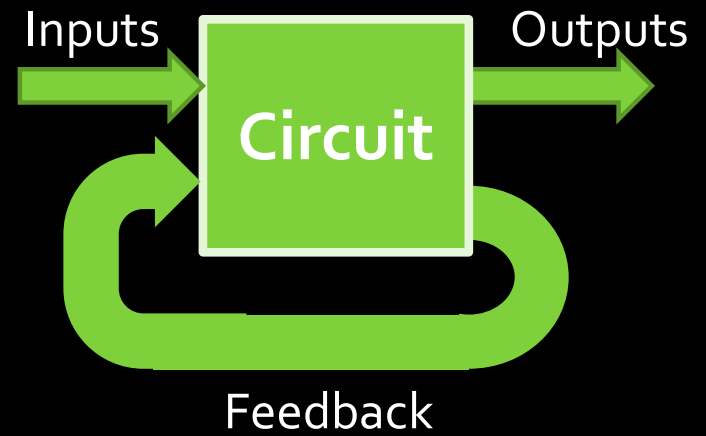
- This creates circuits whose internal state can change over time, where the same input values can result in different outputs.
- Why would we need circuits like this?
  - Memory values
  - Reacting to changing inputs



© 1968 United Feature Syndicate, Inc.

# Creating sequential circuits

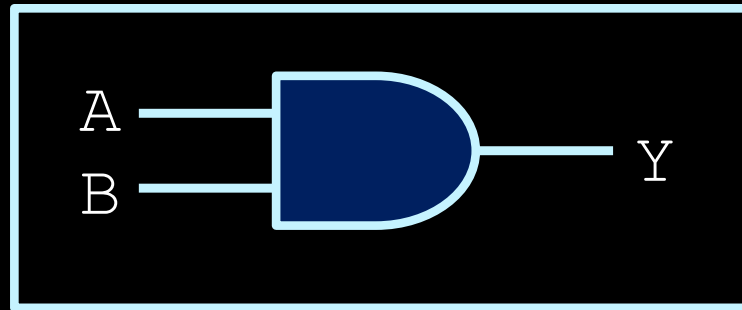
- Essentially, sequential circuits are a result of having feedback in the circuit.
  - How is this accomplished?
  - What is the result of having the output of a component or circuit be connected to its input?



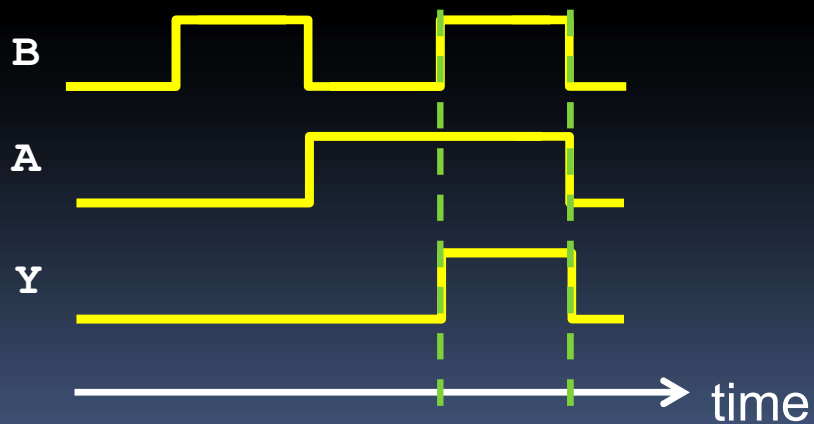
# Gate Delay

- 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."

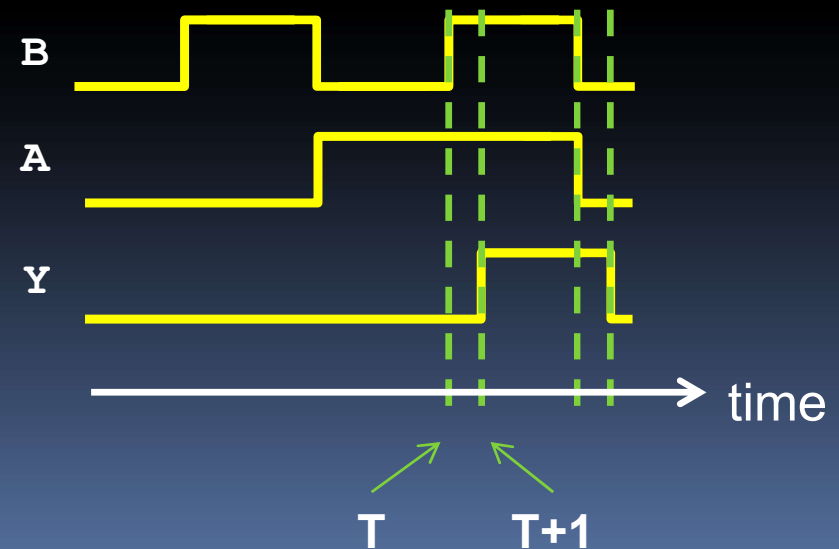
# Gate delay example



Ideal



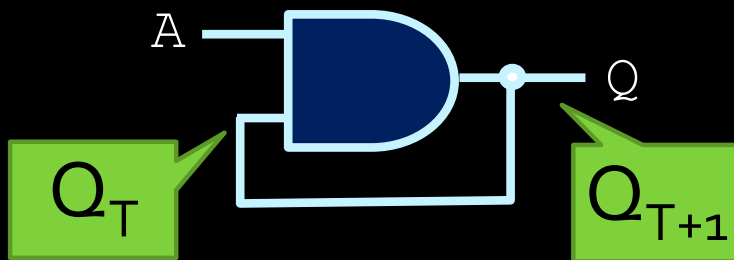
Considering delays





# Feedback Circuit Example (AND)

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



$Q_T$  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         |

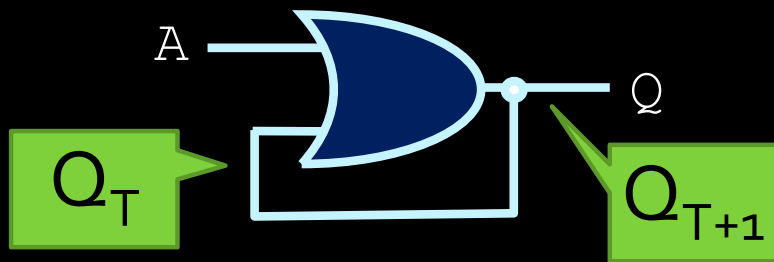
If  $A=0$ ,  $Q_{T+1}$  becomes 0 no matter what  $Q_T$  was.

What happens next for later values of  $A$ ?

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

# Feedback Circuit Example (OR)

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

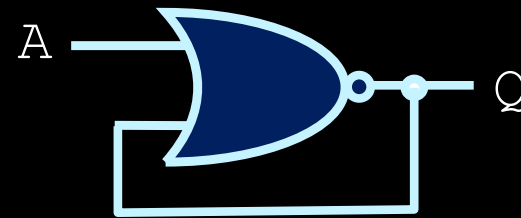
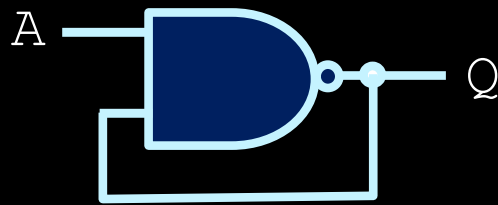
If  $A=1$ ,  $Q_{T+1}$  becomes 1 no matter what  $Q_T$  was.

What happens next for later values of  $A$ ?

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

# Feedback Examples (NAND, NOR)

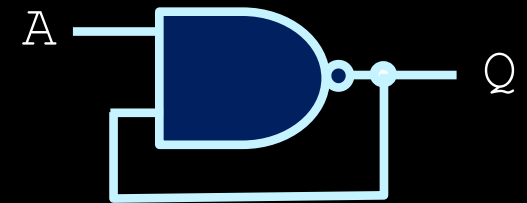
- NAND, NOR gates w/ feedback have more interesting characteristics, which lend themselves to storage devices.



- 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!
- If we set  $A=1$ ,  $Q$ 's value can change, but there's a catch!

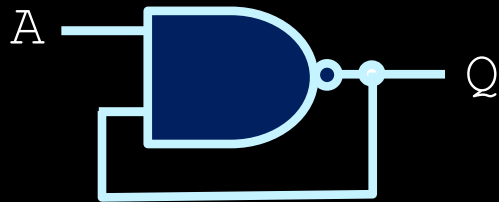


What happens in these last two scenarios?

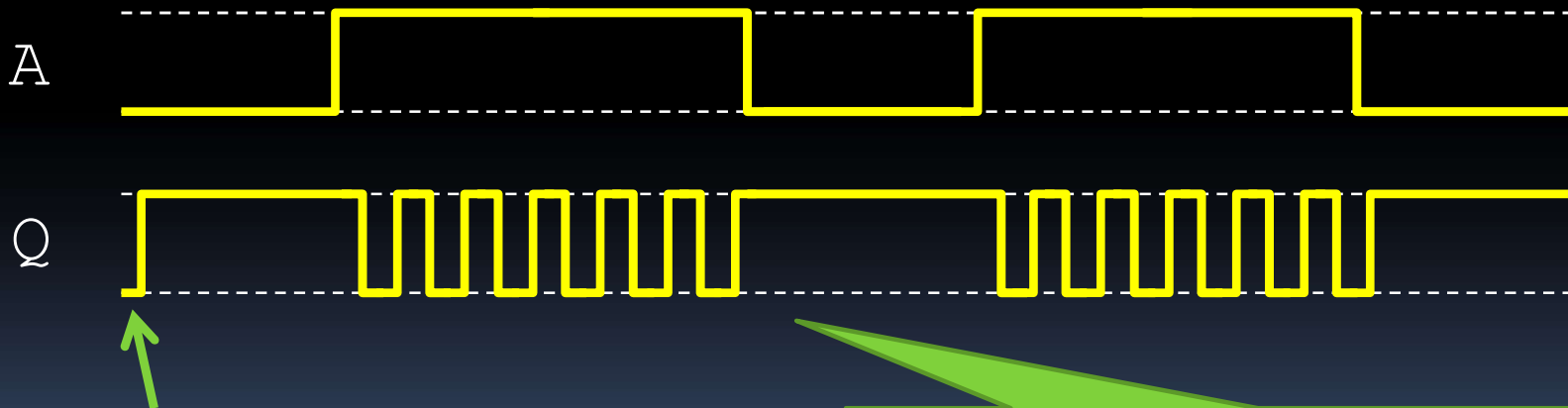
| A | $Q_T$ | $Q_{T+1}$ |
|---|-------|-----------|
| 0 | 0     | 1         |
| 0 | 1     | 1         |
| 1 | 0     | 1         |
| 1 | 1     | 0         |

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         |

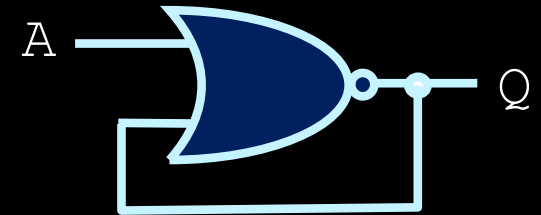


Gate delay. Output does not change instantaneously

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         |

# Feedback behaviour

- NAND behaviour

| A | $Q_T$ | $Q_{T+1}$ |
|---|-------|-----------|
| 0 | 0     | 1         |
| 0 | 1     | 1         |
| 1 | 0     | 1         |
| 1 | 1     | 0         |

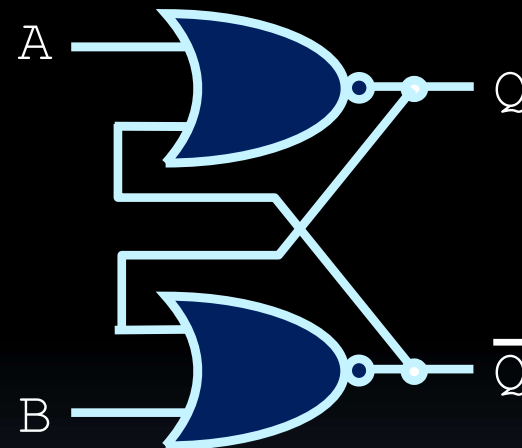
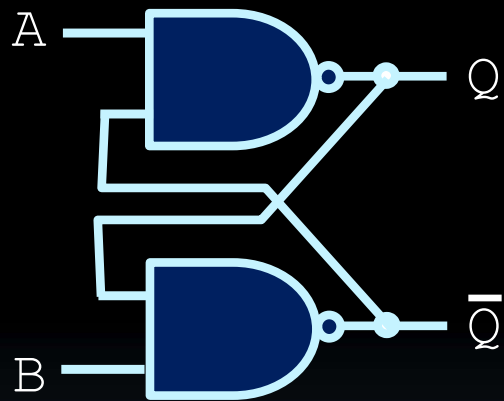
- NOR behaviour

| 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**.

# Latches

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

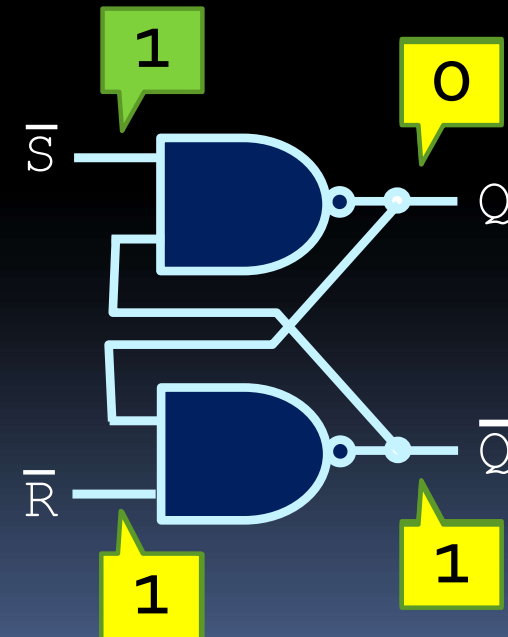
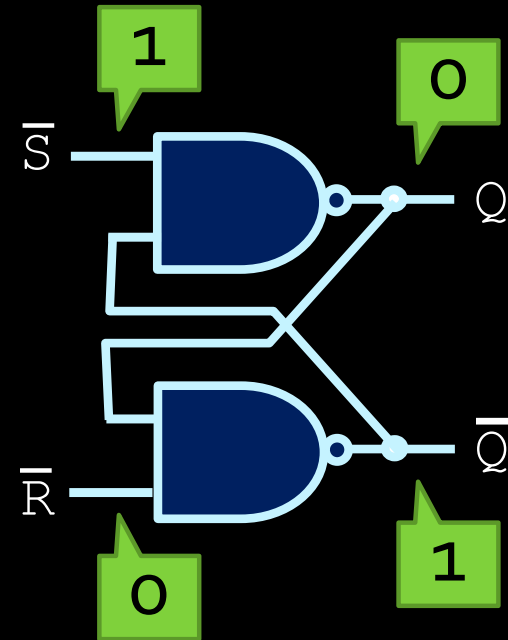


- These circuits are called **latches**.



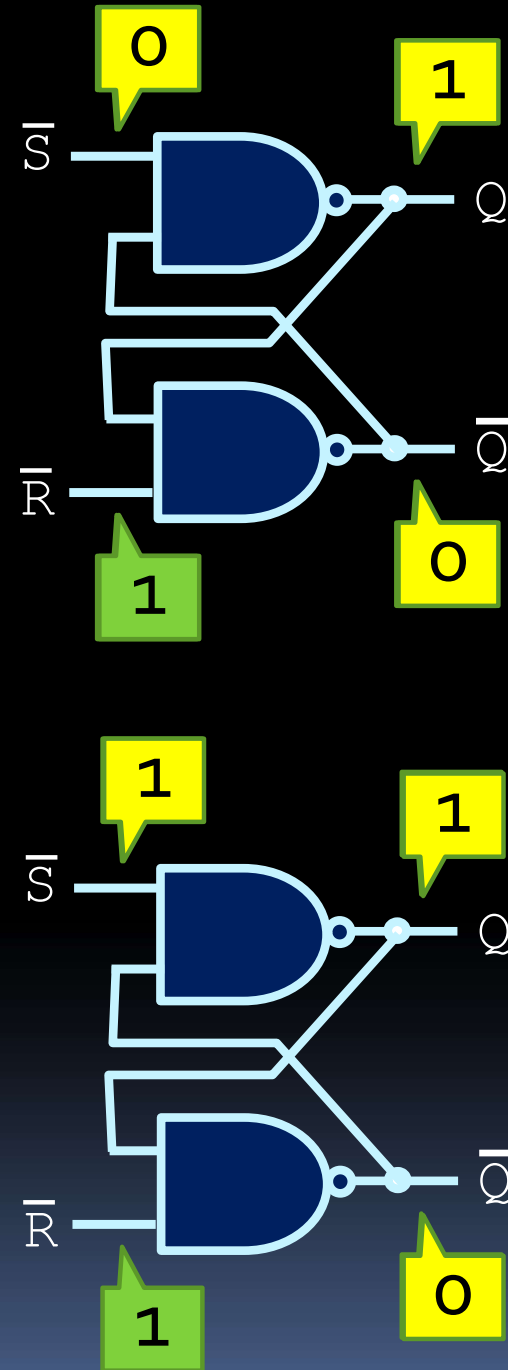
# $\overline{S}\overline{R}$ latch

- 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.

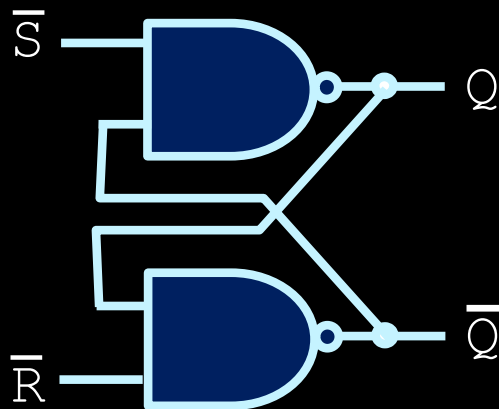


# $\overline{S}\overline{R}$ latch

- (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!



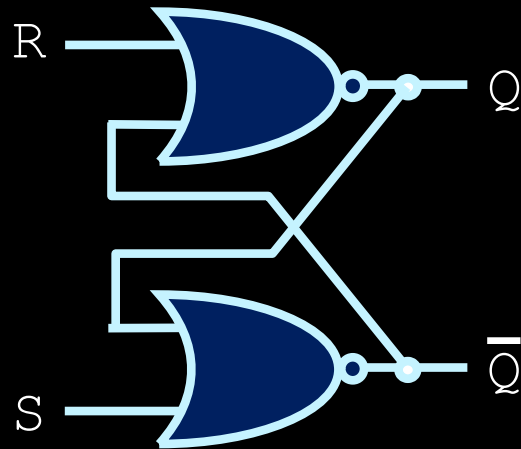
# $\overline{S}\overline{R}$ latch



| $\overline{S}$ | $\overline{R}$ | $Q_T$ | $\overline{Q}_T$ | $Q_{T+1}$ | $\overline{Q}_{T+1}$ |
|----------------|----------------|-------|------------------|-----------|----------------------|
| 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 unstable behaviour!
  - Output depends on which input changes first.

# SR latch

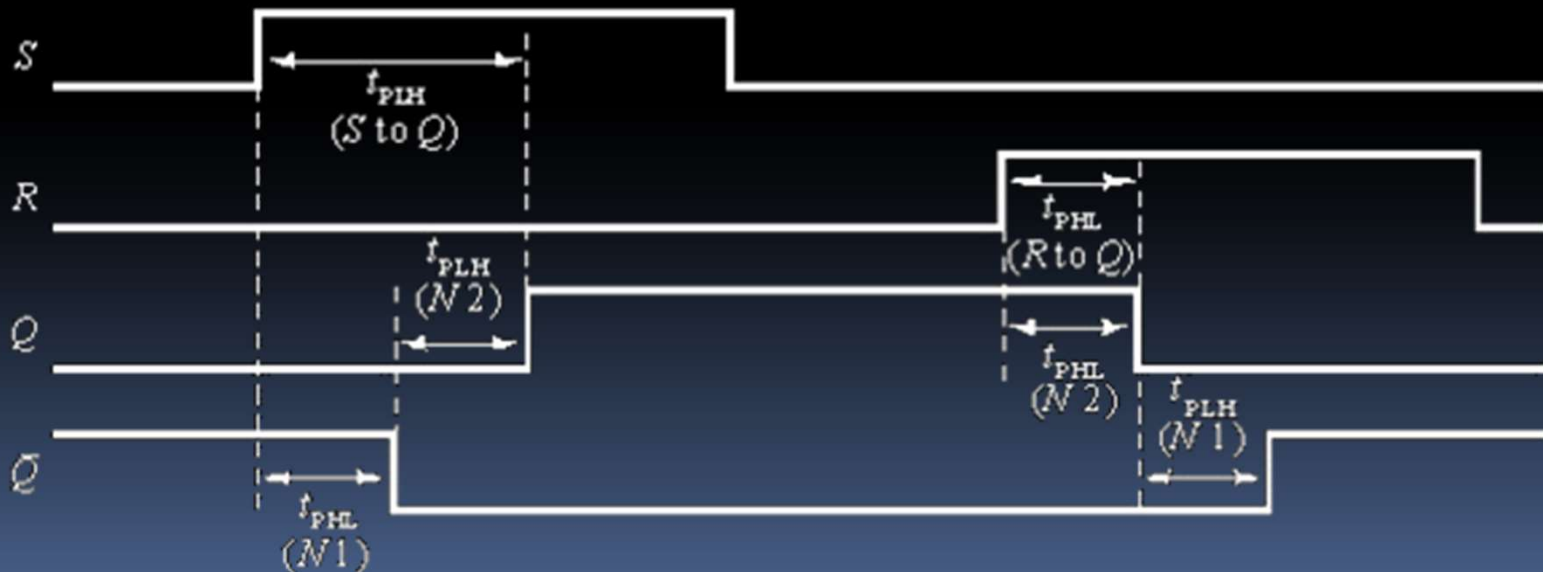
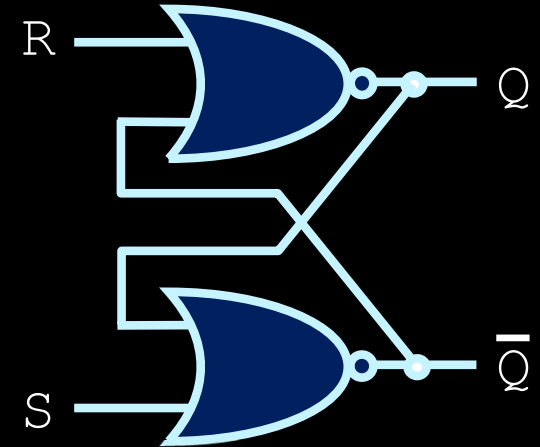


| S | R | $Q_T$ | $\bar{Q}_T$ | $Q_{T+1}$ | $\bar{Q}_{T+1}$ |
|---|---|-------|-------------|-----------|-----------------|
| 0 | 0 | 0     | 1           | 0         | 1               |
| 0 | 0 | 1     | 0           | 1         | 0               |
| 0 | 1 | X     | X           | 0         | 1       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

- In this case, S and R are “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.

# SR latch timing diagram

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



# More on instability

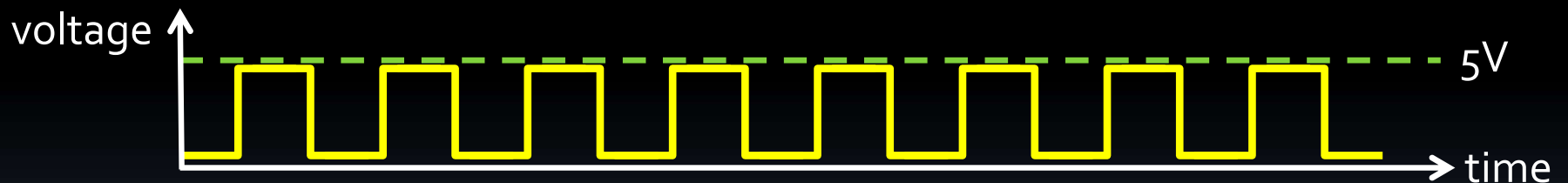
- Unstable behaviour occurs when a  $\overline{S}\overline{R}$  latch's inputs go from 00 to 11, or a SR latch's inputs go from 11 to 00.
  - The signals don't change simultaneously, so the outcome depends on which signal changes first.
- Because of the unstable behaviour, 00 is considered a **forbidden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

# Introducing the Clock

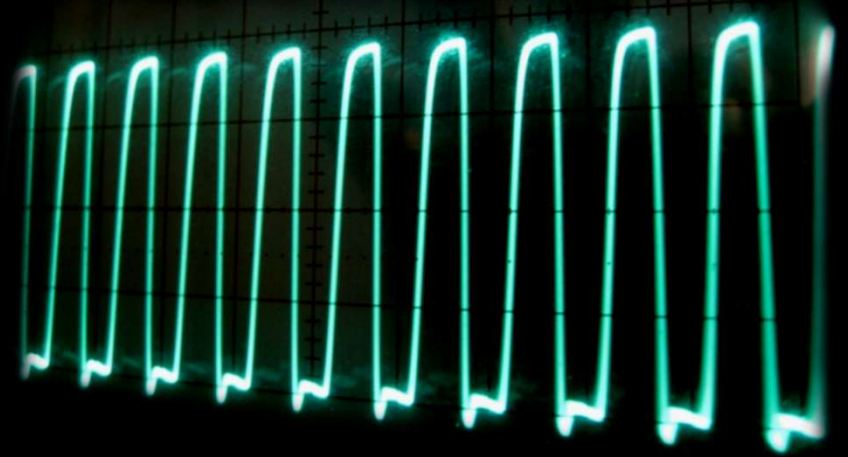
- Now we have circuit units that can store high or low values. How can we read from them?
  - For instance, when do we know when the output is ready to be sampled?
  - If the output is high, how can we tell the difference between a single high value and two high values in a row?
- Need some sort of timing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



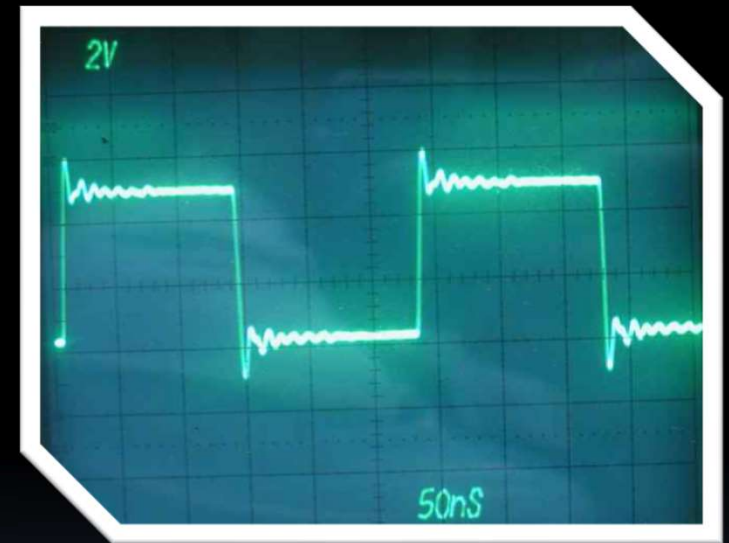
- But looks more like:



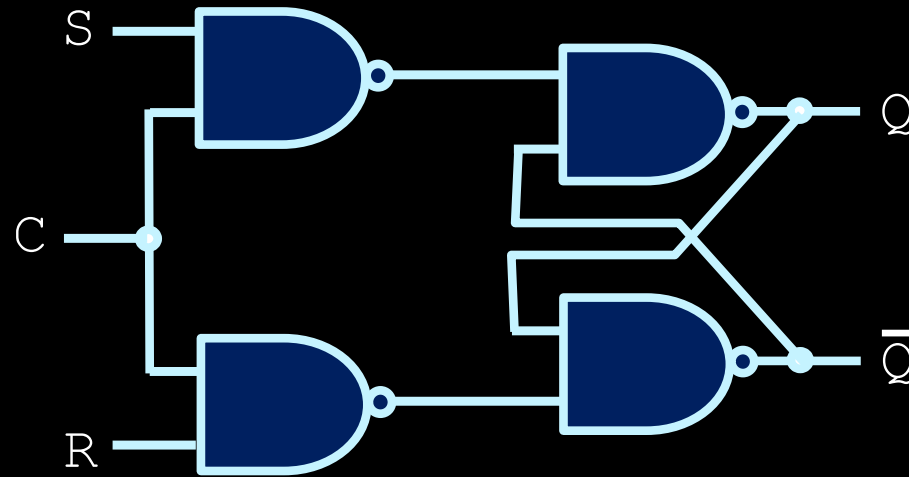


# Signal restrictions

- 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).



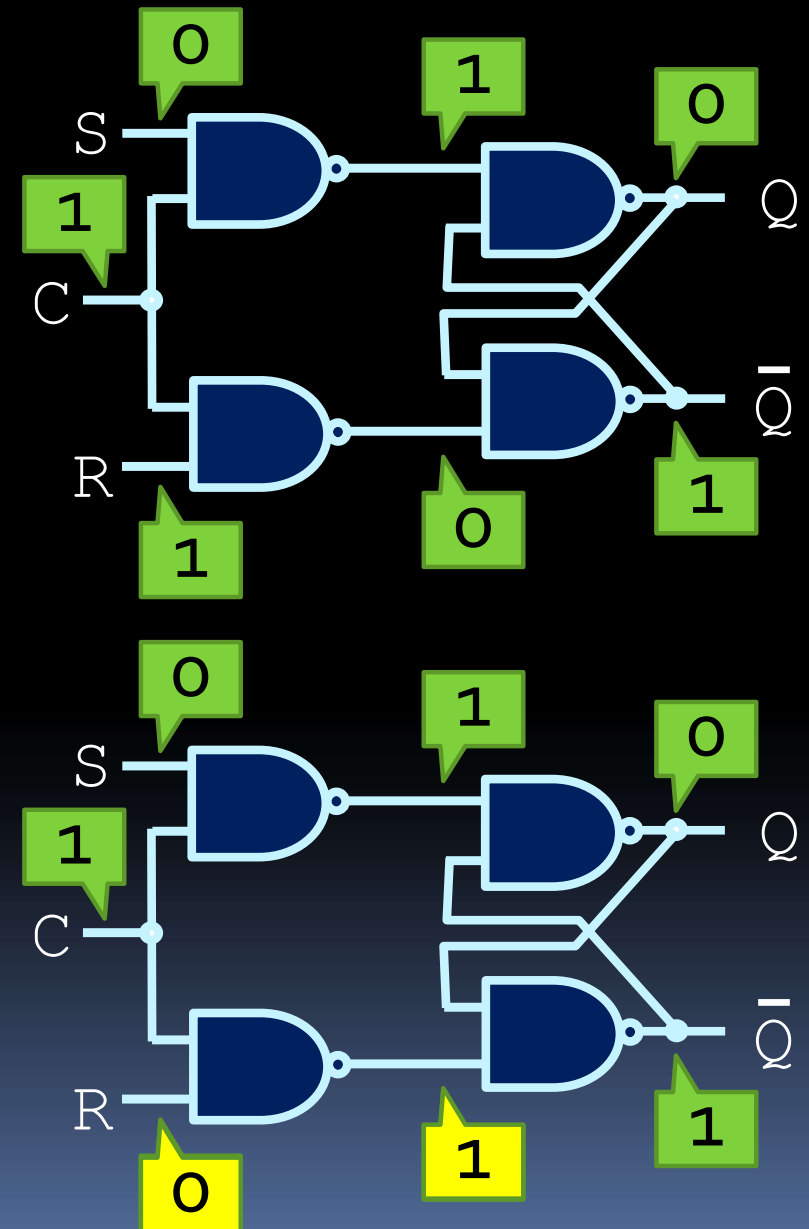
# Clocked SR latch



- Adding another layer of NAND gates to the  $\bar{S}\bar{R}$  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)

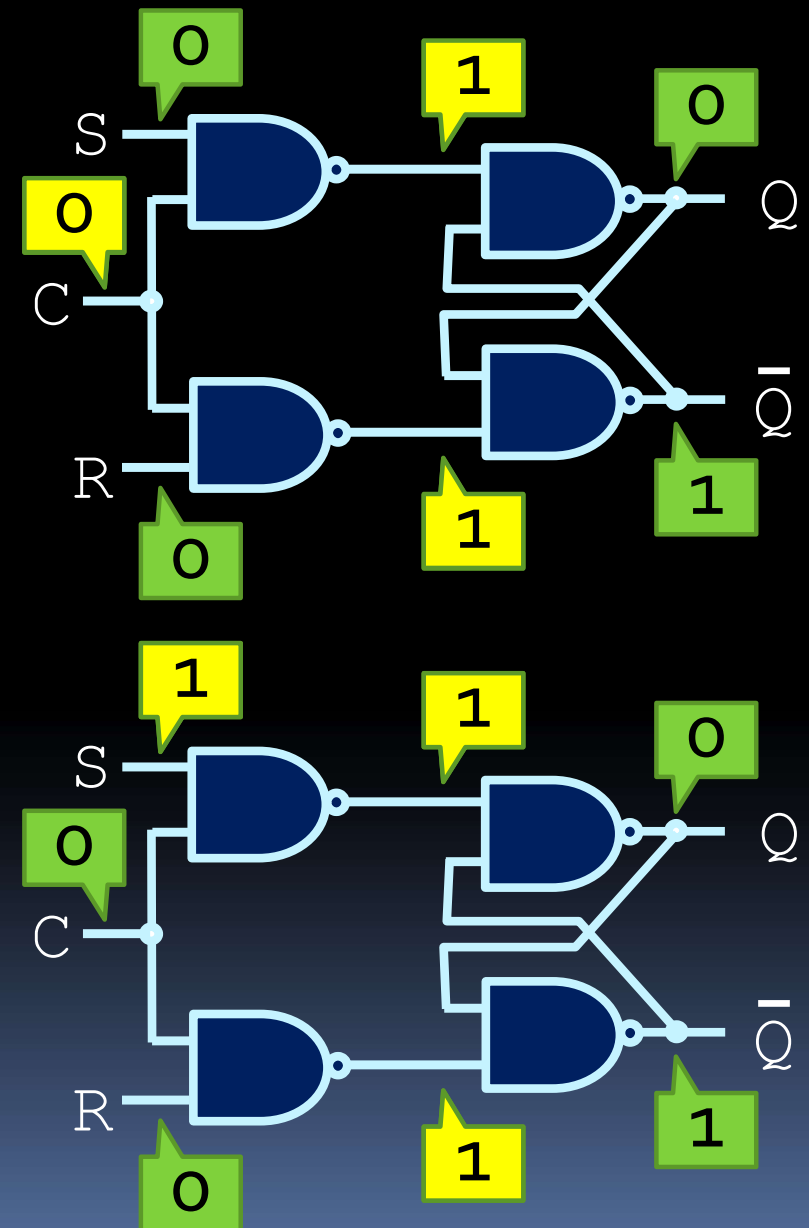
# Clocked SR latch behaviour

- 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.

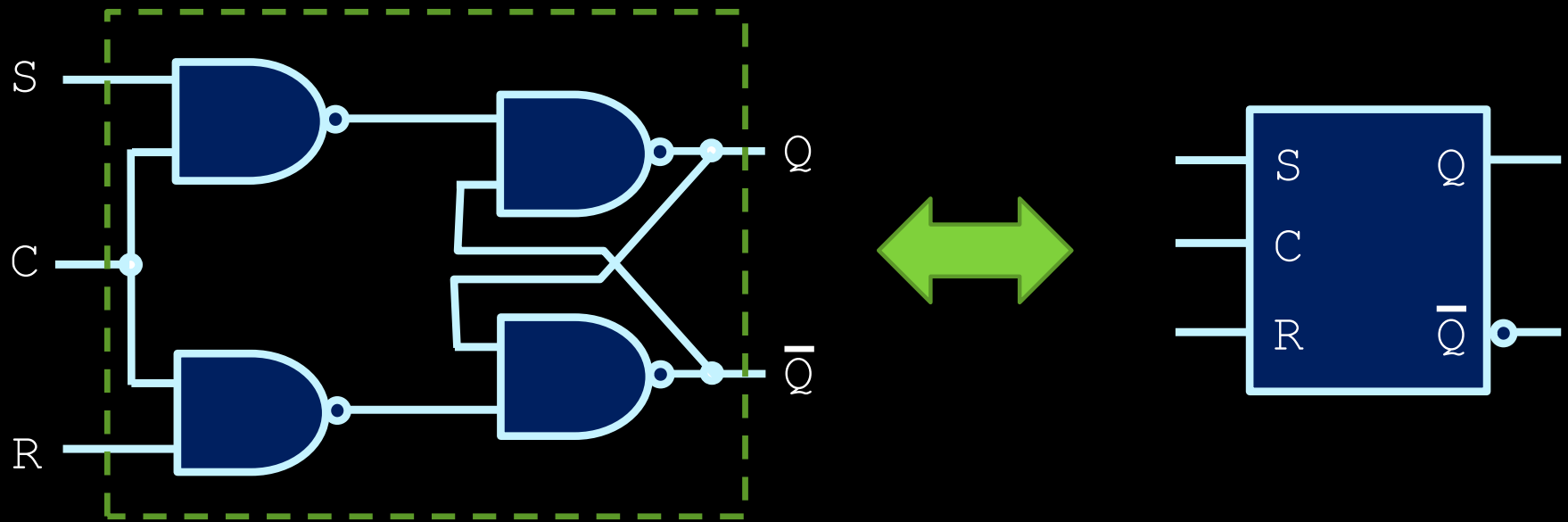


# Clocked SR latch behaviour

- 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.

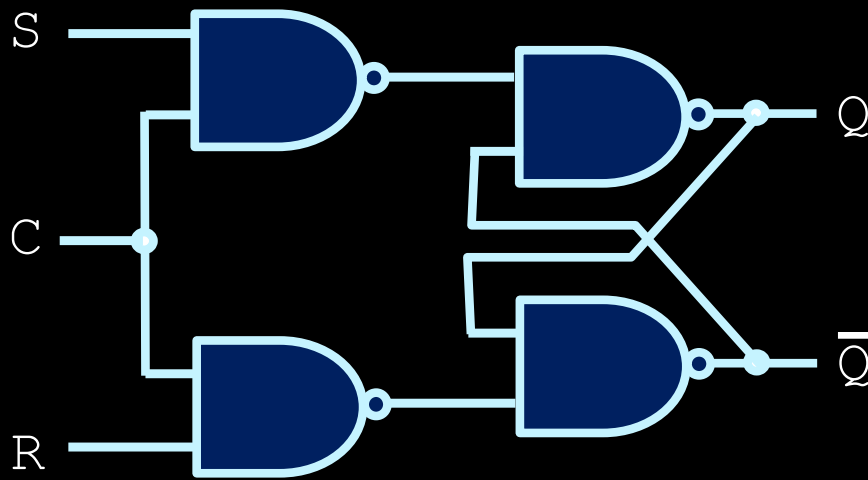


# Clocked SR latch



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

# Clocked SR latch behaviour

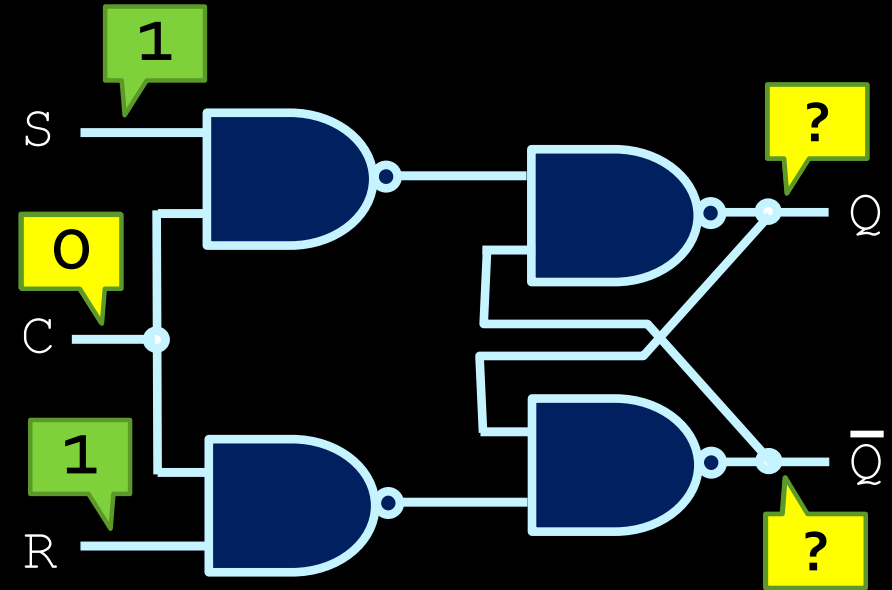


| C | S | R | $Q_{T+1}$ | Result    |
|---|---|---|-----------|-----------|
| 0 | X | X | $Q_T$     | no change |
| 1 | 0 | 0 | $Q_T$     | no change |
| 1 | 0 | 1 | 0         | reset     |
| 1 | 1 | 0 | 1         | set       |
| 1 | 1 | 1 | ?         | Undefined |

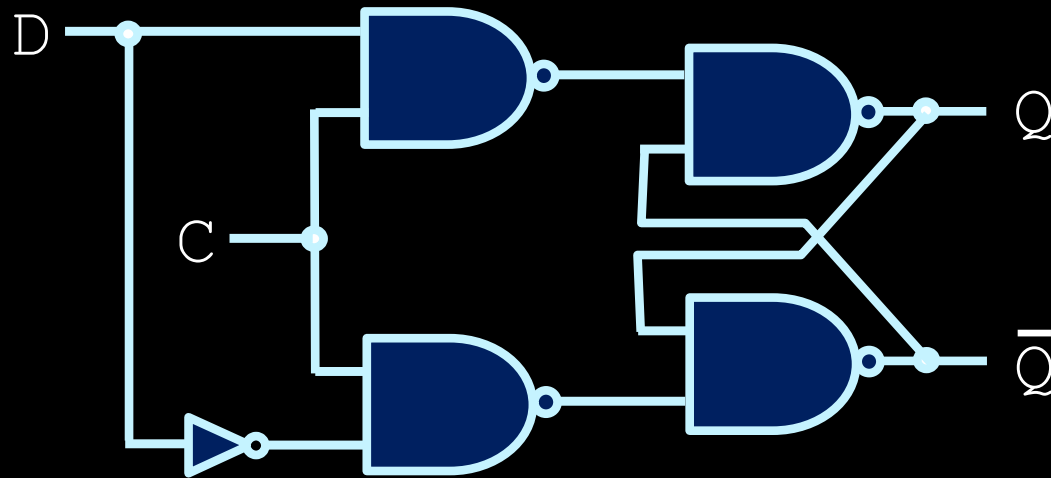
- Assuming the clock is 1, we still have a problem when S and R are both 1, since the state of Q is indeterminate.
  - This set of inputs is considered a **forbidden state**.

# Forbidden state

- Why is it “forbidden” to have  $S$  and  $R$  high at the same time?
- If the clock is high as well, then the  $Q$  and  $\bar{Q}$  outputs will also be high.
  - When the clock goes from 1 to 0, the output values depend on which middle wire changes first.
- One solution: Prevent  $S$  and  $R$  from having high values at the same time.



# D latch (or gated D-latch)

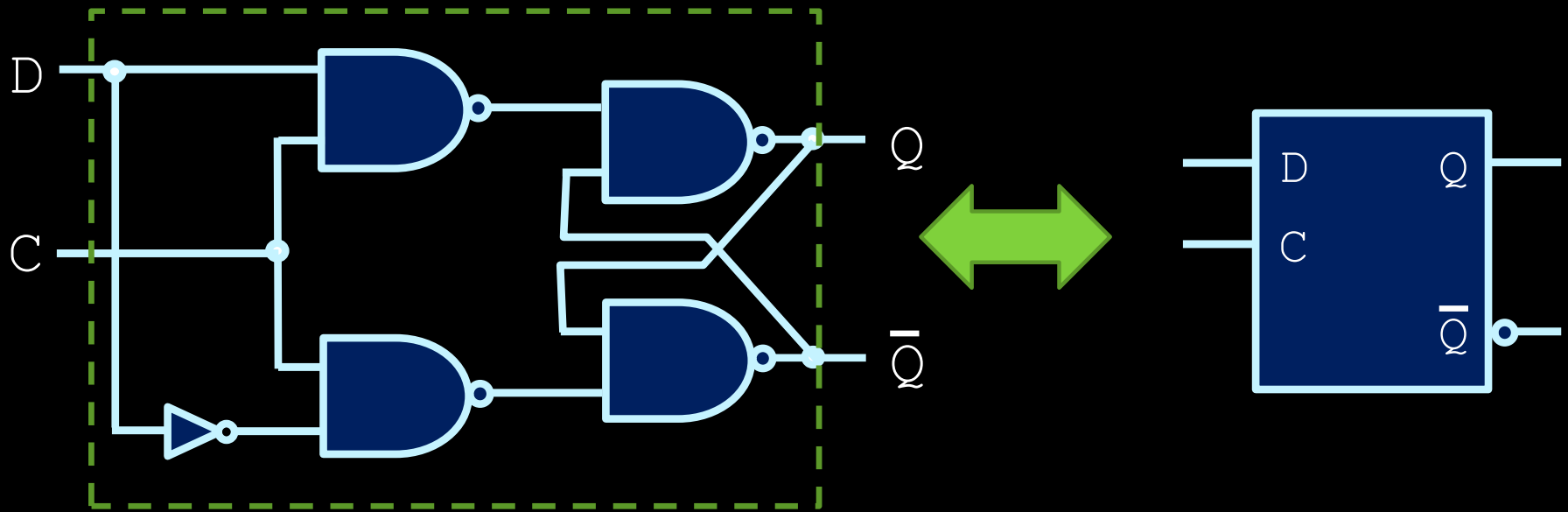


| $Q_T$ | $D$ | $Q_{T+1}$ |
|-------|-----|-----------|
| 0     | 0   | 0         |
| 0     | 1   | 1         |
| 1     | 0   | 0         |
| 1     | 1   | 1         |

- 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 whenever  $C$  is high.



# D latch



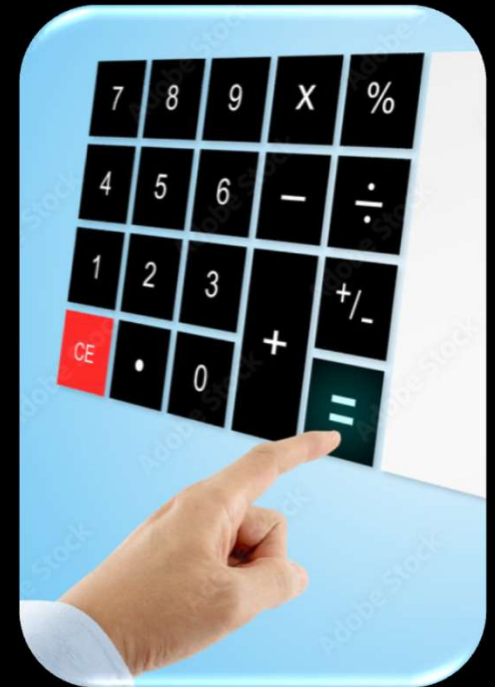
- This design is good, but still has problems.
  - i.e. **timing issues**.

# Latches vs Flip-Flops



# Calculator Behaviour

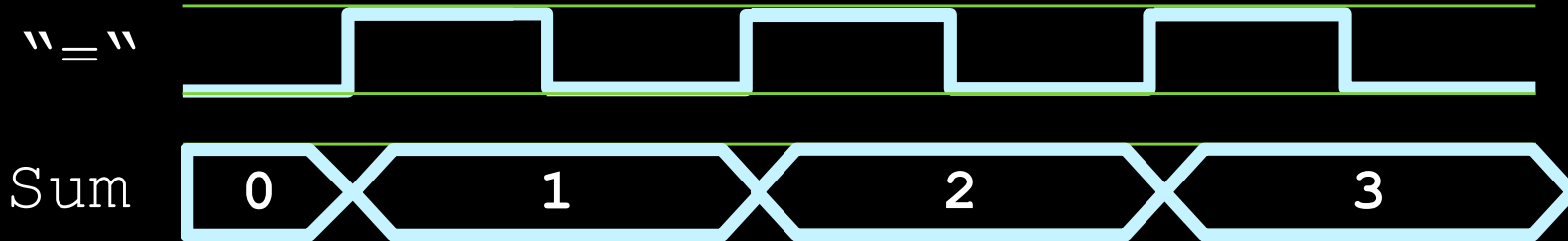
- If you enter "1+1" on a calculator, what happens each time you press the "=" button?
- Expectations:



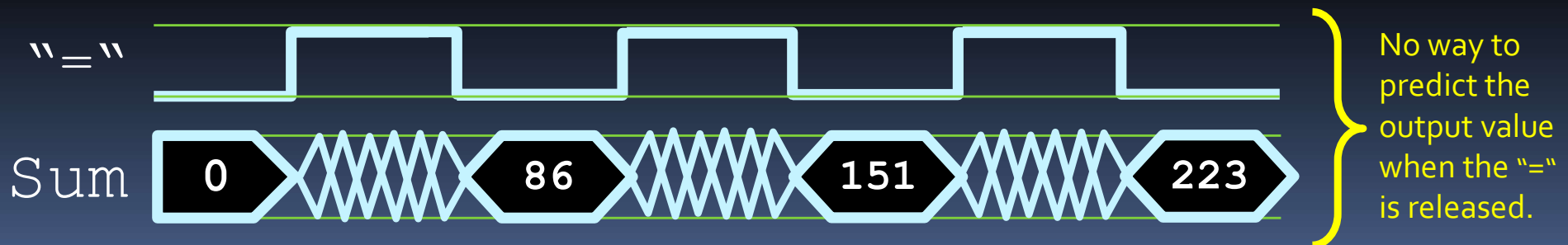
- Latches can't do this!

# Calculator Behaviour (cont'd)

- Latches allow their output to change **as long as the clock signal is high**. So instead of this:



- ...what you would see when you press the "=" button would be this:

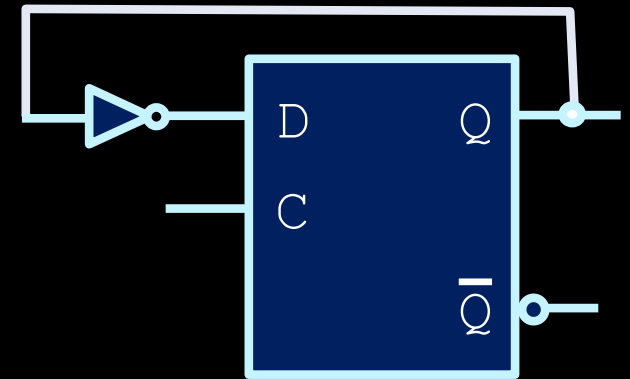


# Latches are transparent

- **Transparent** means:
  - Any changes to the inputs of a D latch are visible to the output when control signal (`clock`) is 1.
- **What this means:**
  - “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.”

# Another transparency example

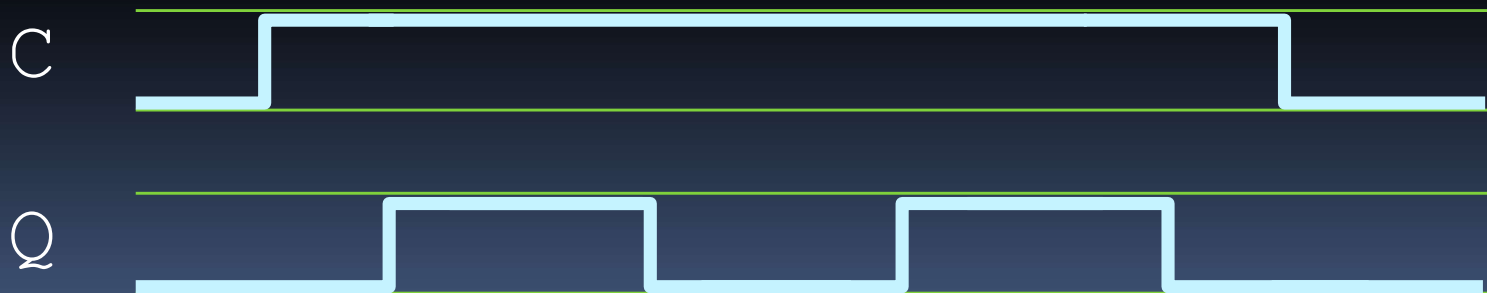
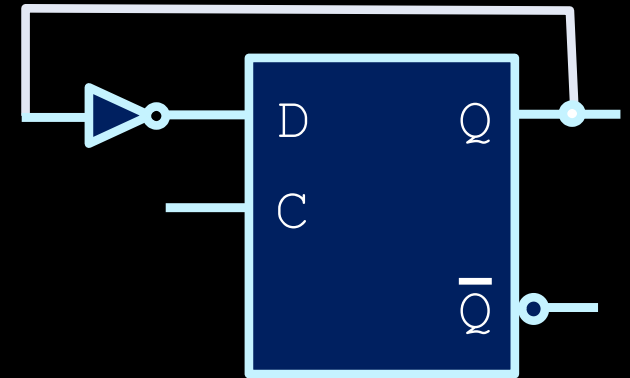
- Consider the circuit here:
  - You might expect the output to flip whenever the clock goes high.
- However, when the clock signal is high, **the output toggles back and forth from 0 and 1** until the clock goes low again.



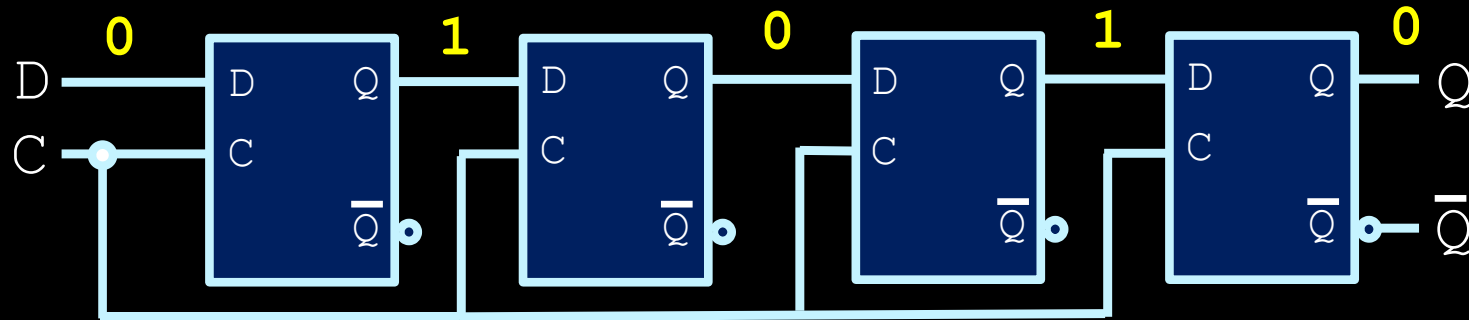
...what happens next?

# Another transparency example

- Consider the circuit here:
  - ▣ You might expect the output to flip whenever the clock goes high.
- However, when the clock signal is high, **the output toggles back and forth from 0 and 1** until the clock goes low again.



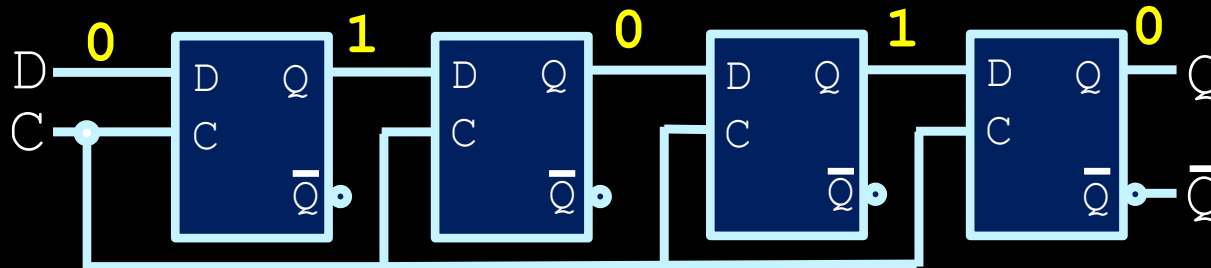
# Final transparency example



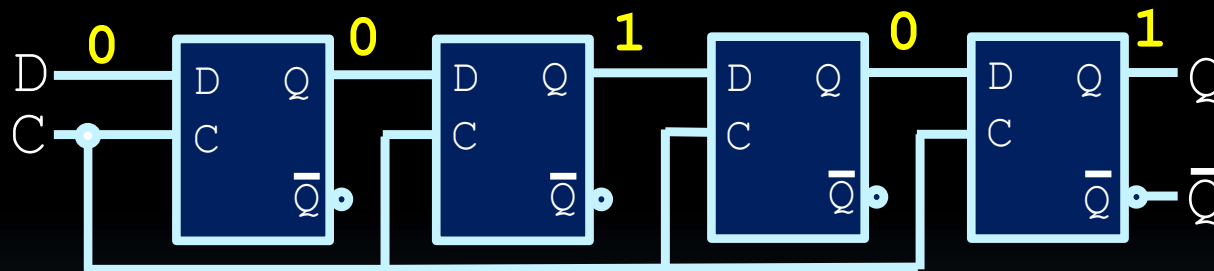
- If the wires have the values 01010 when the clock goes from 0 to 1, what do you expect the wire values to change to?
  - But what do they actually change to?



# Value- vs Edge-triggered



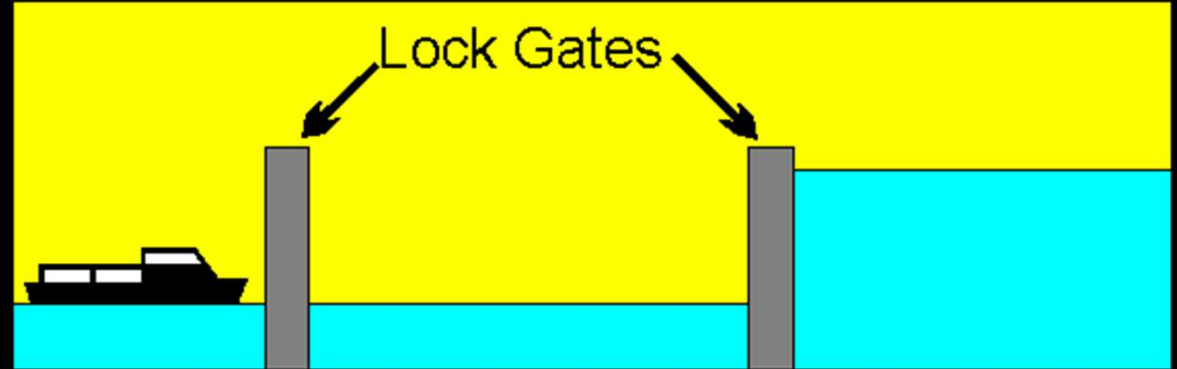
- Instead of having all these latches flooded by the 0 value, we'd rather have the clock shifting the latch values over by one:



- We'd also imagine this happening at **the moment the clock changes**, not for **the duration of the high clock signal**.
  - **Edge-triggered** vs **value-triggered**.
- How can we make this happen?

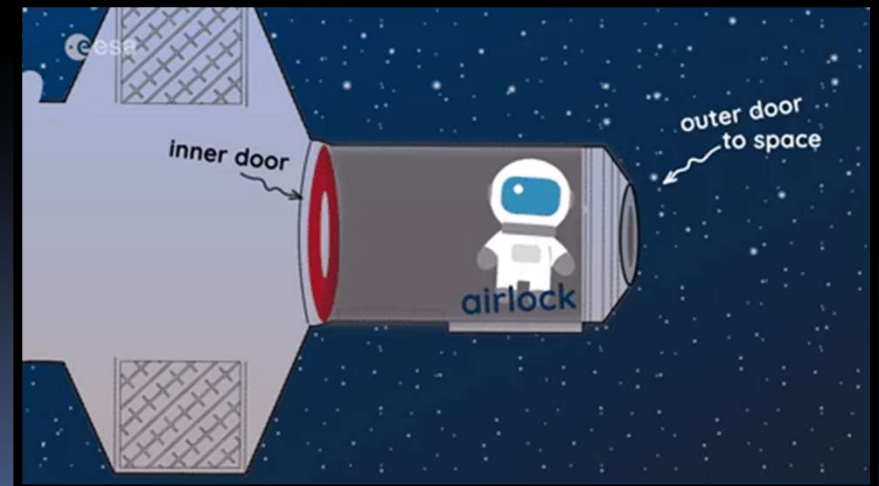
# Canal (or airlock) model

- Picture how boats travel through a canal in stages, one stage at a time:



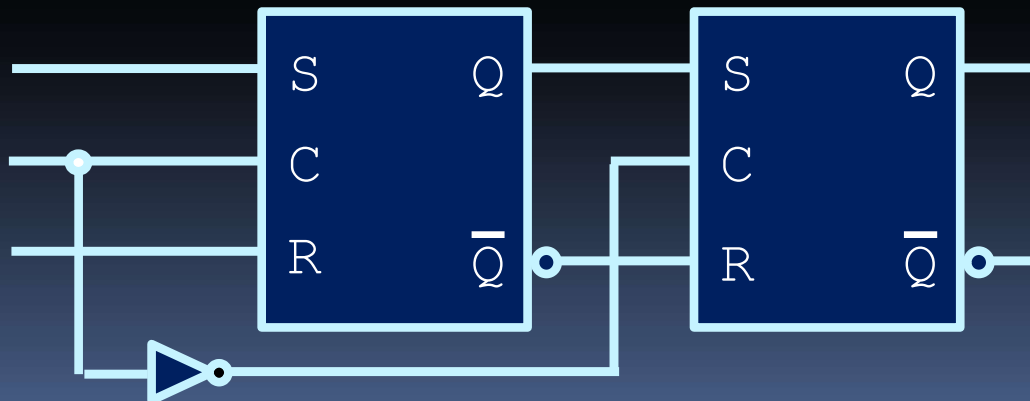
1. Boats enter the middle section (aka the lock),
2. Entry gate is closed,
3. Water level is raised,
4. Exit gate is opened,
5. Boat leaves the canal.

- Similar idea to airlocks on a space station.



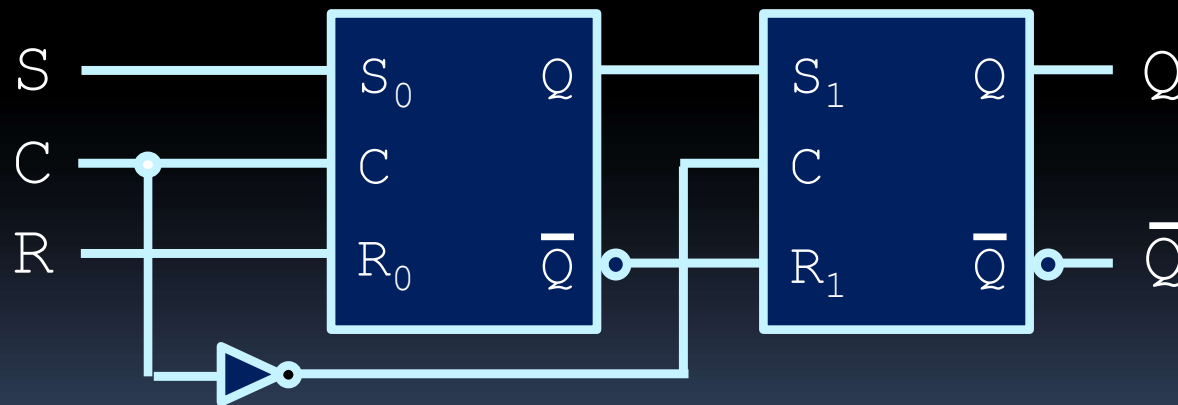
# Latch timing issues

- How does this work with latches?
  - We want the output to change only once when the clock changes, **at the moment it changes**.
  - Solution: Create a middle section that is set to the input value when clock is high, and then sends that value to the output when the clock is low.
  - Prevents unwanted feedback and changes to output.

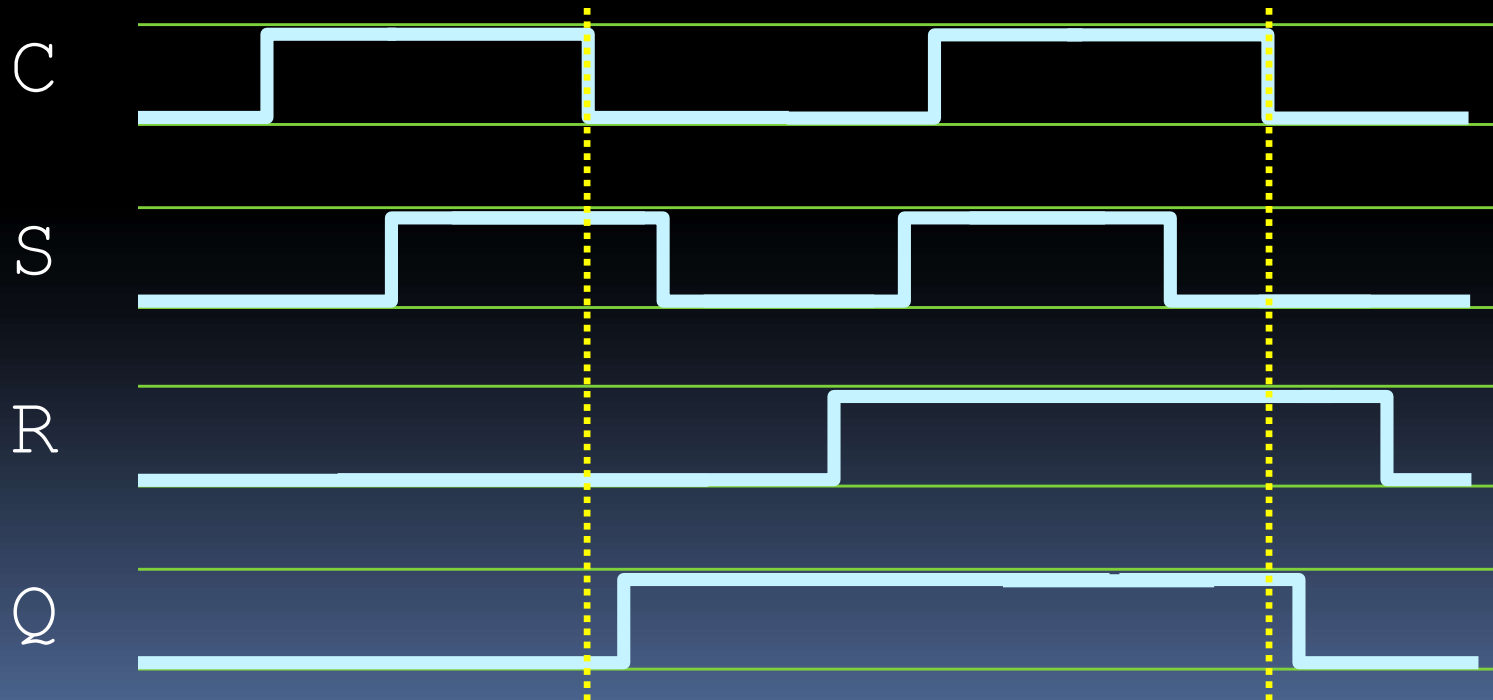
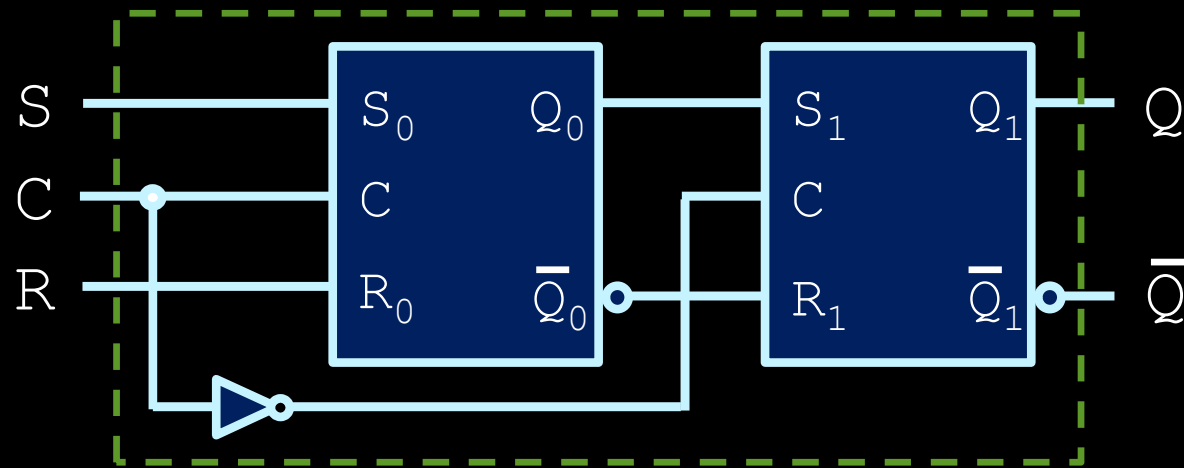


# SR master-slave flip-flop

- 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

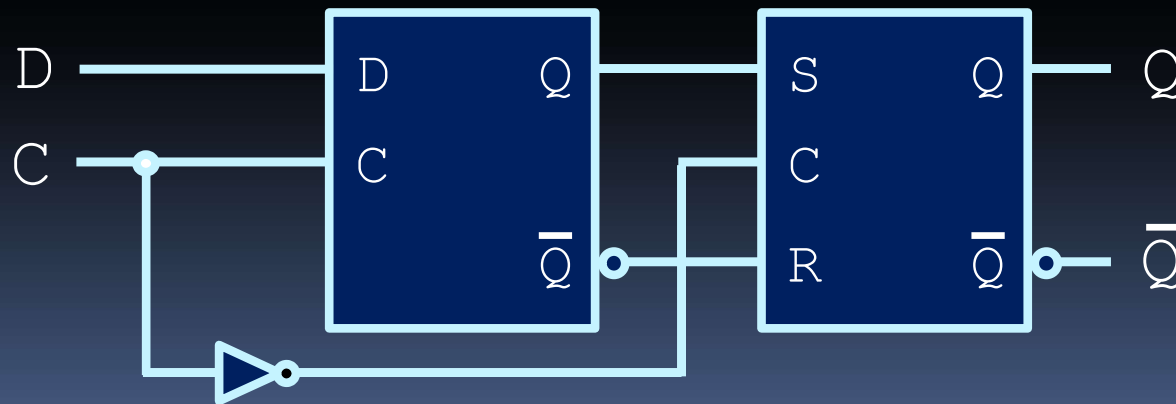


# SR master-slave flip-flop



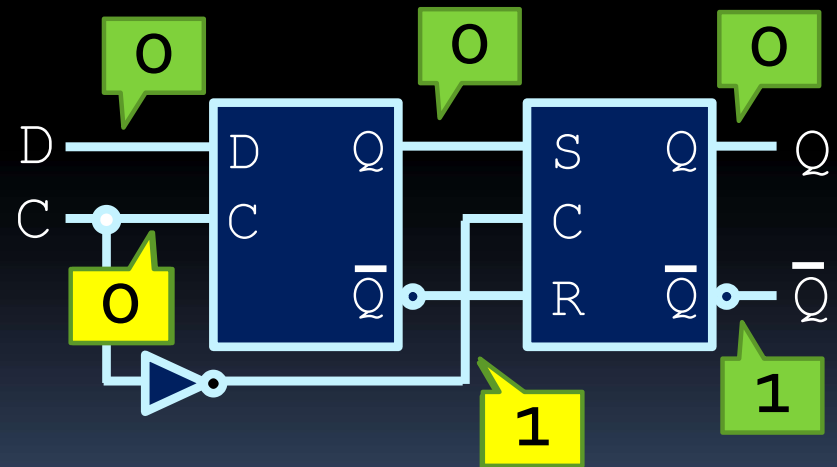
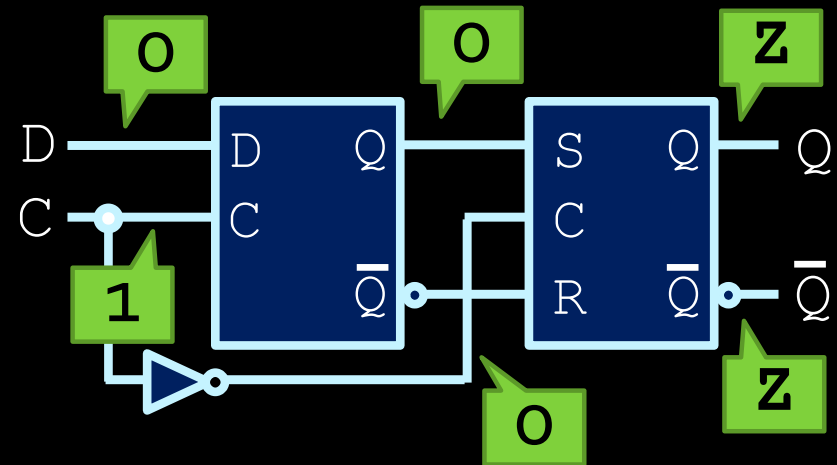
# Edge-triggered D flip-flop

- 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)



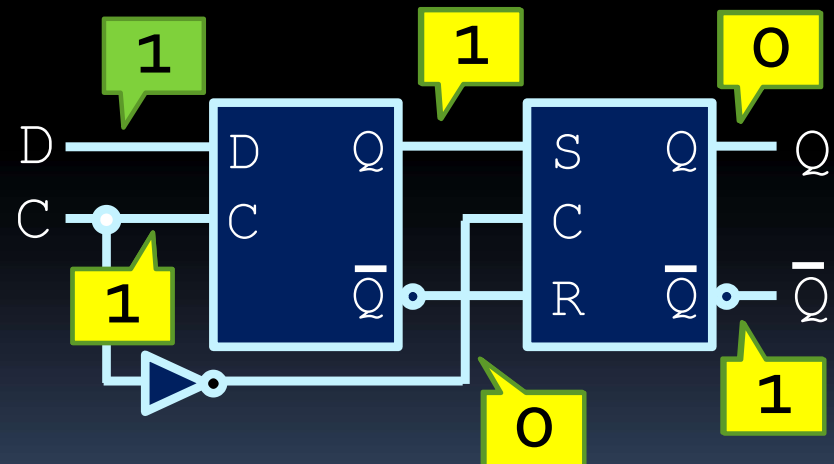
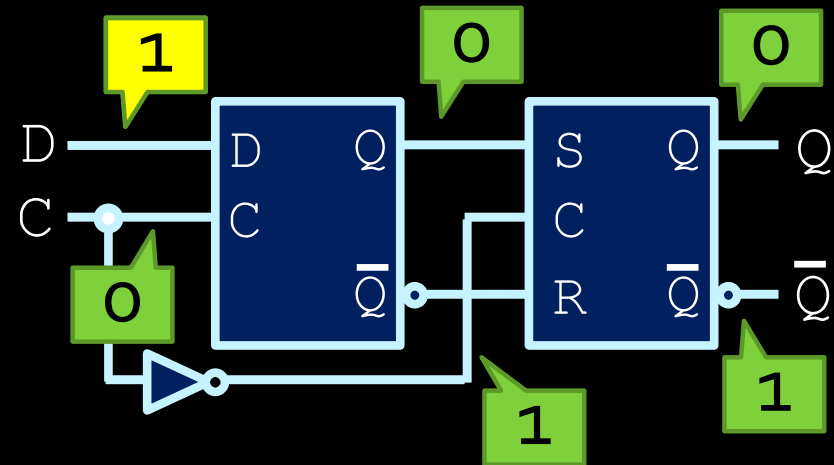
# Flip-flop behaviour

- 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 the clock goes from high to low, the first flip-flop stops transmitting a signal, and the second one starts.



# Flip-flop behaviour

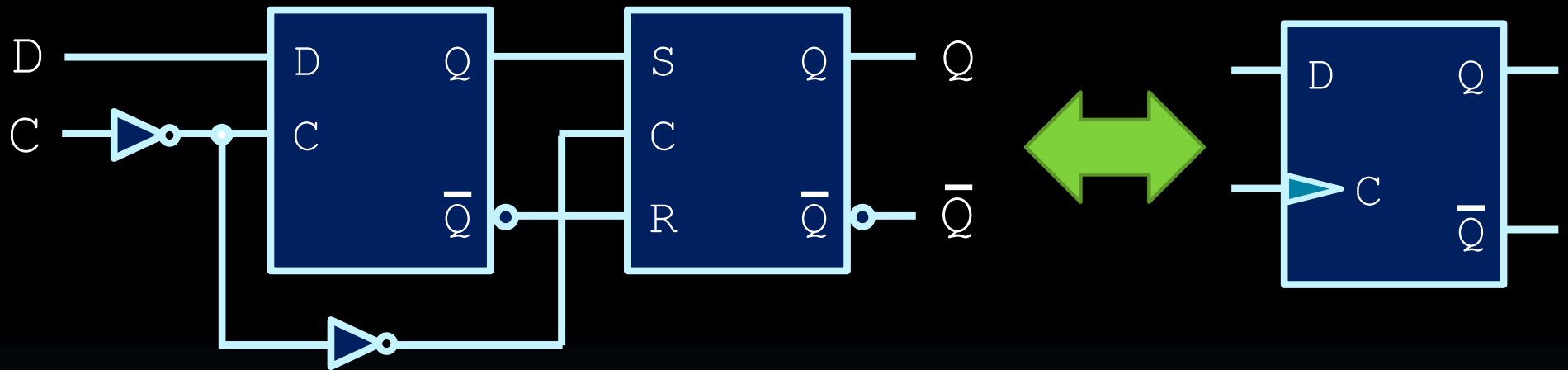
- Continued from previous:
  - If the input to D changes, the change isn't transmitted to the second flip-flop until the clock goes high again.
  - Once the clock goes high, the first flip-flop starts transmitting at the same time as the second flip-flop stops.





# Edge-triggered flip-flop

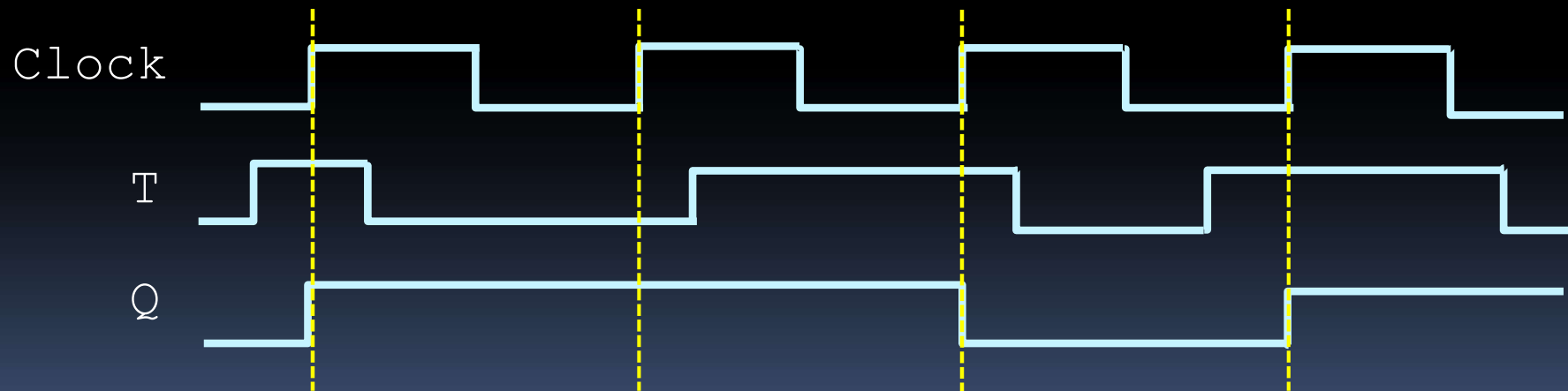
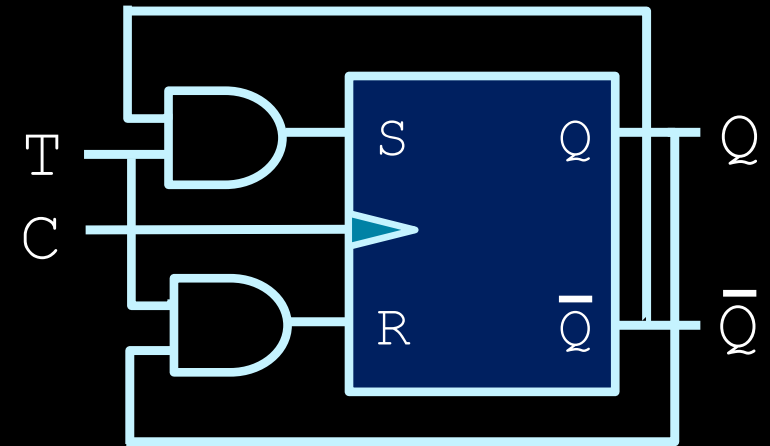
- Alternative: **positive-edge triggered** flip-flops



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

# Other Flip-Flops

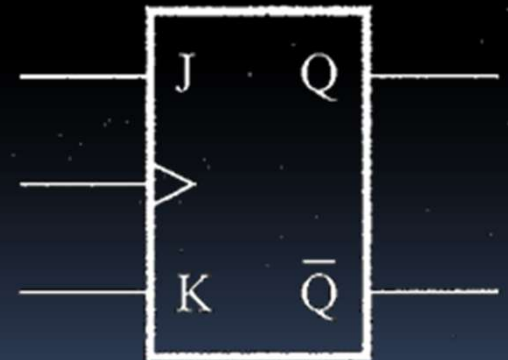
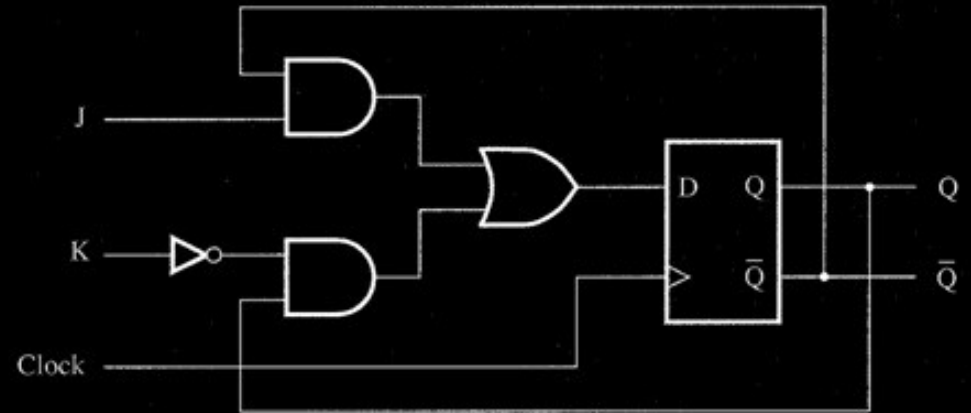
- The **T flip-flop**:
  - Similar to the D flip-flop, except the clock makes it **toggle** (flip between 0 and 1) whenever the T input is high.



# Other Flip-Flops

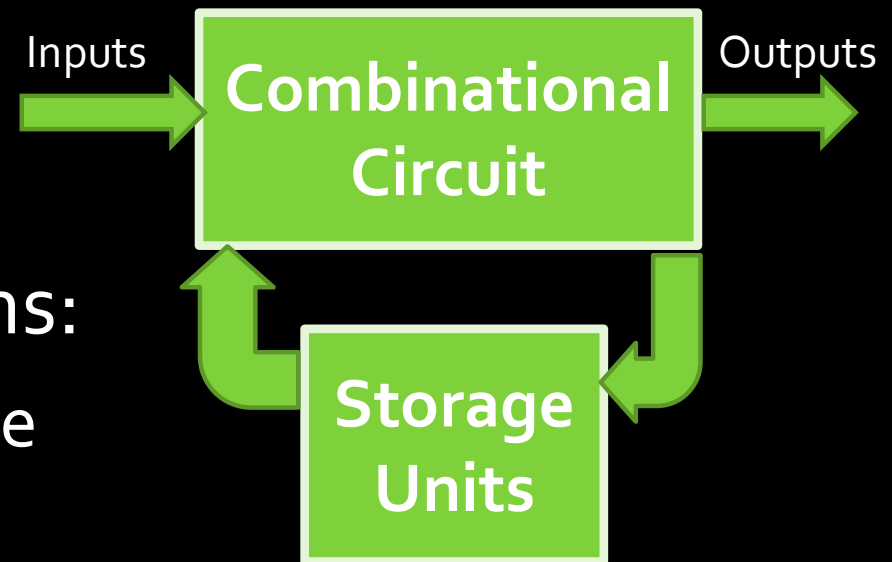
- The **JK Flip-Flop**:

- 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, **reset** 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 the next topic 😊

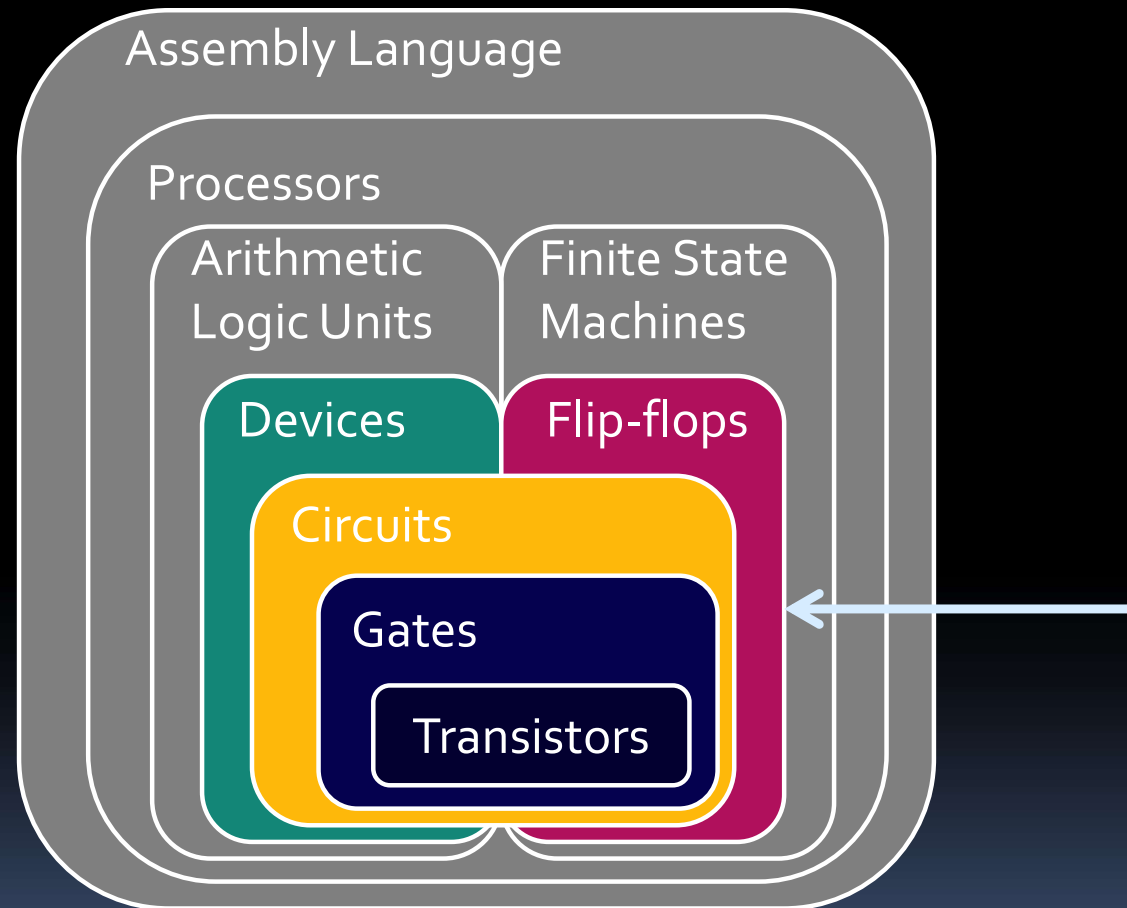




# Sequential Circuit Design

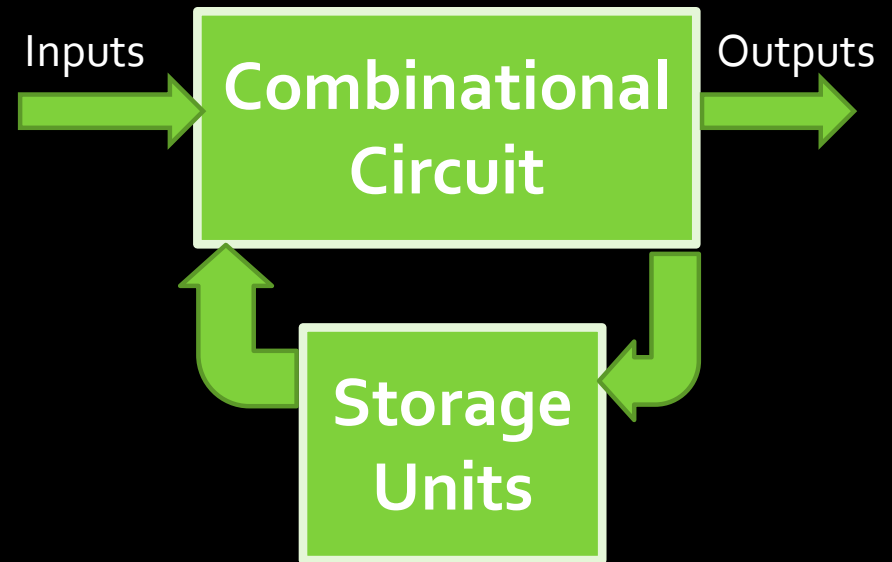


# We are here



# Circuits using flip-flops

- Now that we know about flip-flops and what they do, how do we use them in circuit design?
- What's the benefit in using flip-flops in a circuit at all?



# Timing Considerations

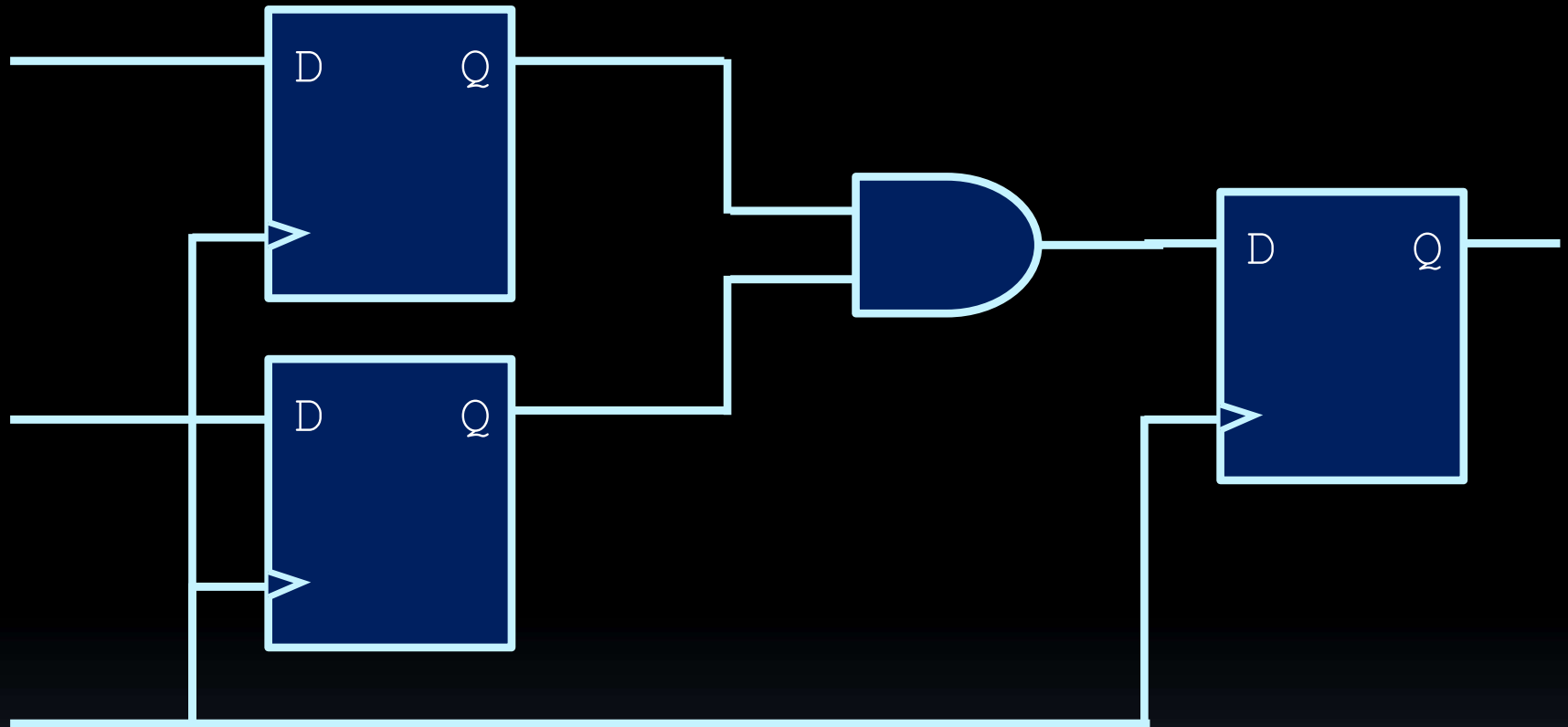




# Flip-Flop Timing

- Input should NOT be changing at the same time as the active edge of the clock.
- **Setup Time:** Input should be stable for some time before active clock edge
- **Hold Time:** Input should be stable for some time immediately after the active clock edge.

# Maximum clock frequency



- Time period between two active clock edges cannot be shorter than **longest propagation delay** between any two flip-flops + **setup time** of the flip-flop

# Resetting inputs

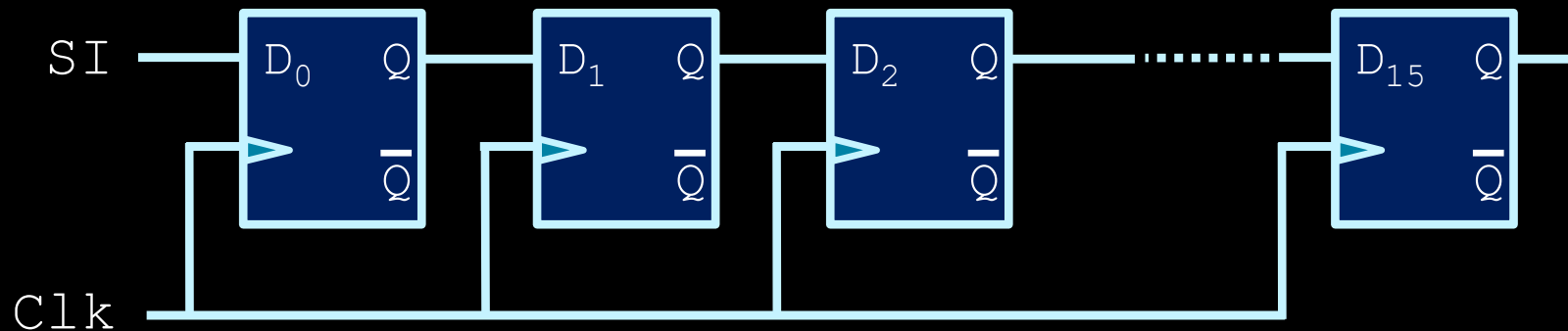
- Since flip-flops (FFs) have unknown state when they are first powered up, we need a convenient way to initialize them
- Reset signal resets the flip flop output to 0
  - This is unrelated to R input of SR latch
- Synchronous reset
  - The output is reset to 0 only **on the active edge of the clock**
- Asynchronous reset
  - The output is reset to 0 **immediately** (as soon as the asynchronous reset signal becomes active), **independent of the clock signal**

# Example #1: Registers



# Shift registers

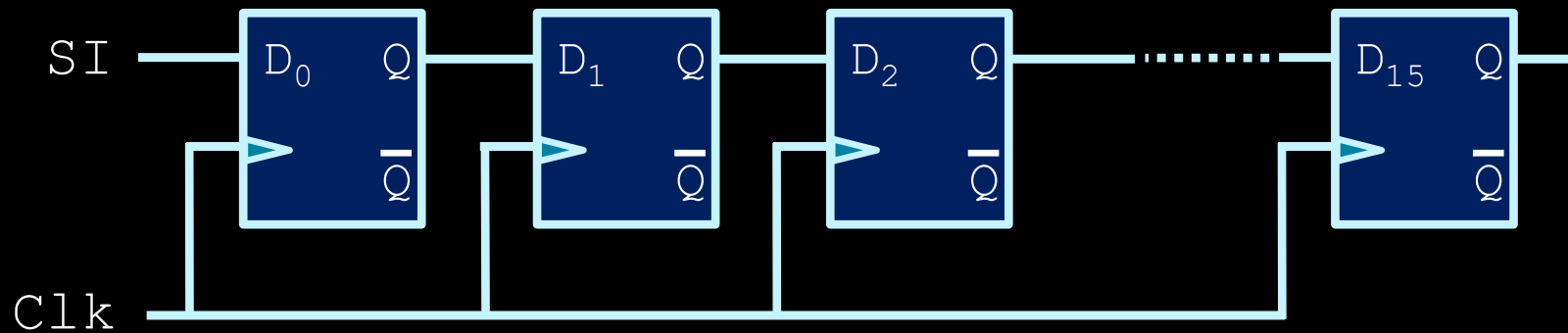
- A series of D flip-flops can store a multi-bit value (such as a 16-bit integer, for example).



- Data can be shifted into this register one bit at a time, over 16 clock cycles.
  - Known as a **shift register**.

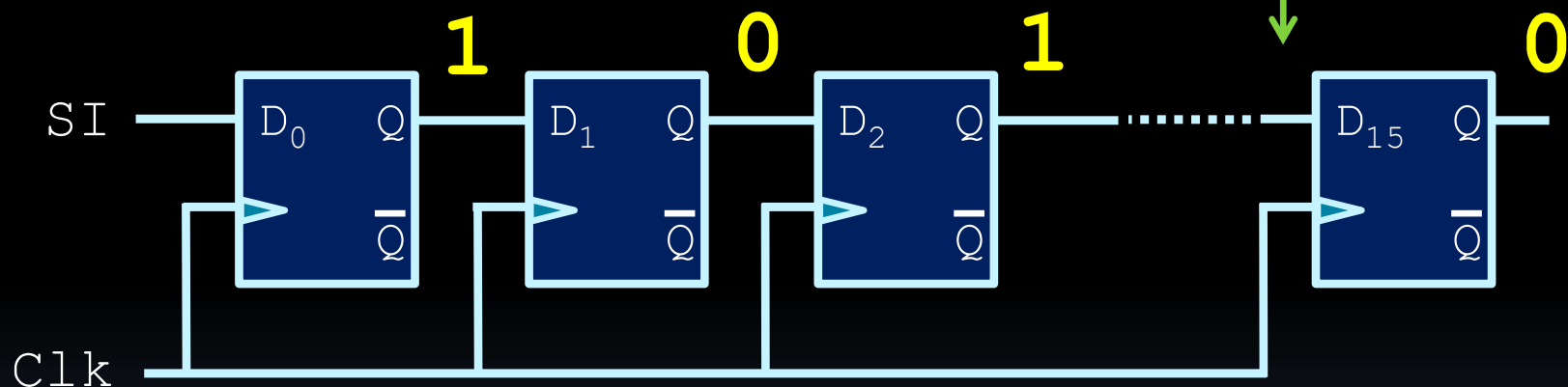
# Shift registers

- Illustration: shifting in 0101010101010101



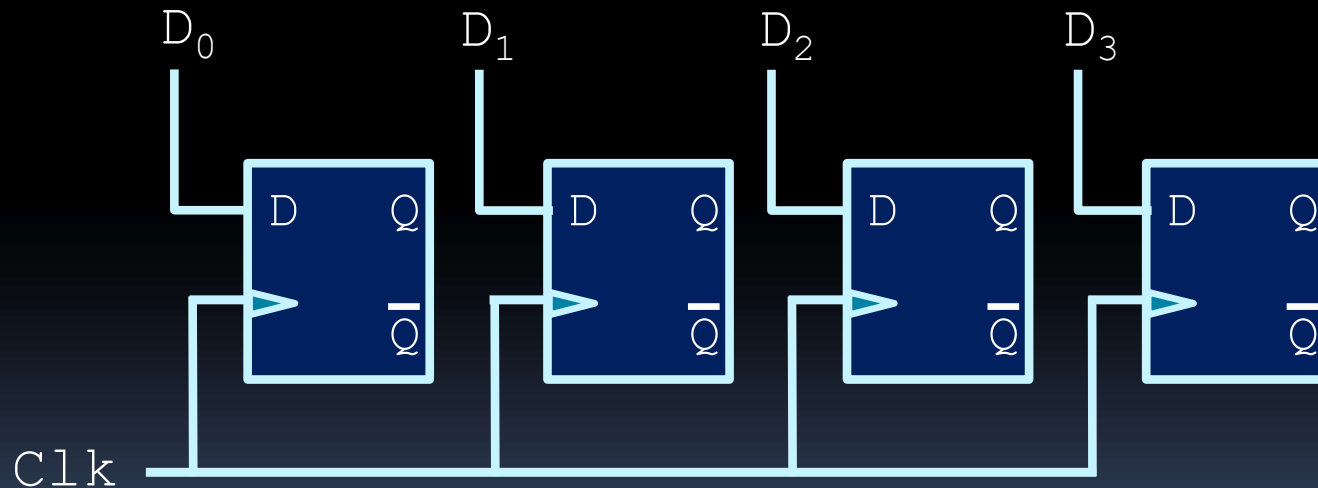
# Shift registers

- Illustration: shifting in 0101010101010101
  - ▣ After 16 clock cycles....



# Load registers

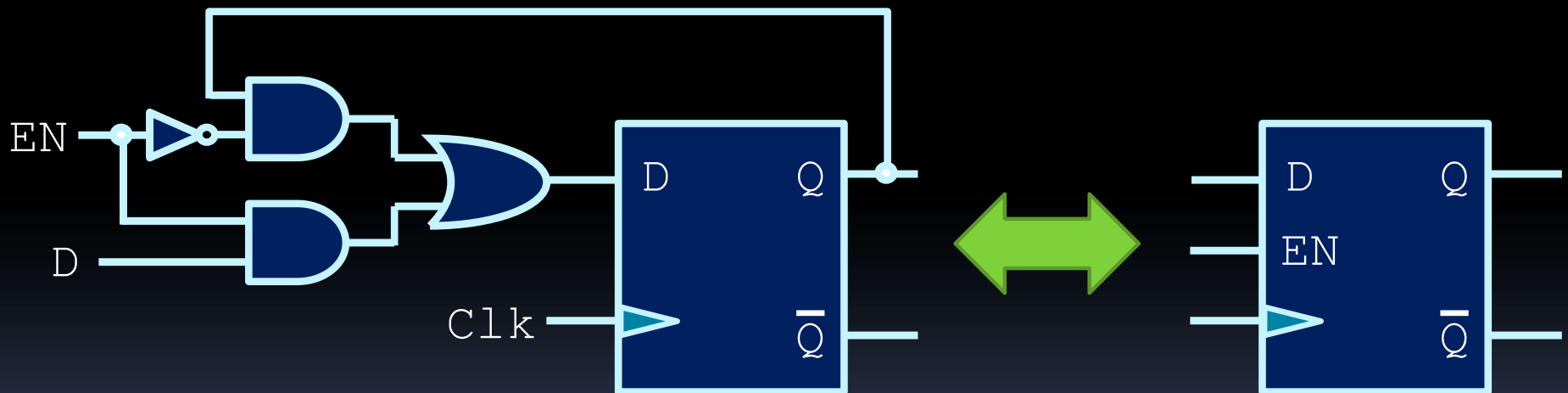
- One can also load a register's values all at once, by feeding signals into each flip-flop:
  - ▣ In this example: a 4-bit **load register**.



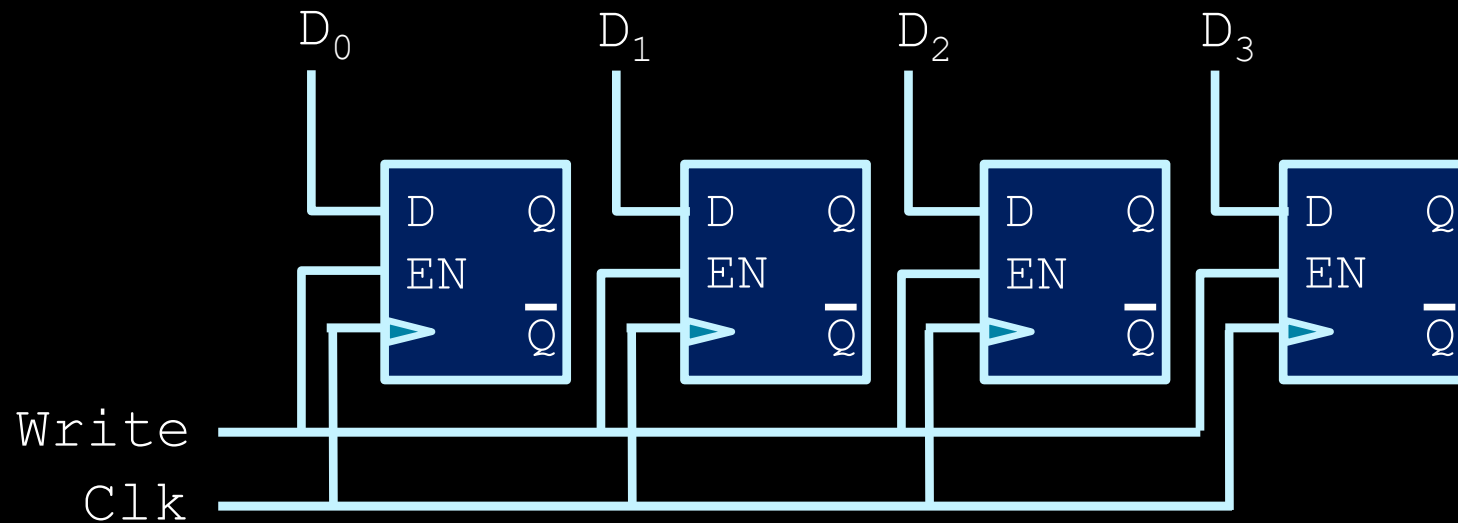


# Load registers

- To control when this register is allowed to load its values, we introduce the **D flip-flop with enable**:



# Load registers



- Implementing the register with these special D flip-flops will now maintain values in the register until overwritten by setting EN high.