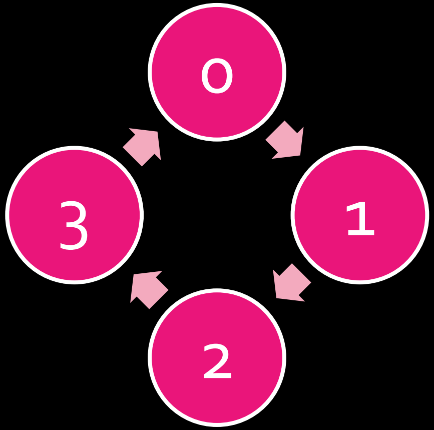
WEEK 4 REVIEW

Sequential Circuits

- How can same input lead to different outputs?
- Sequential logic depends on the sequence of inputs, not just current input
- They have some internal state

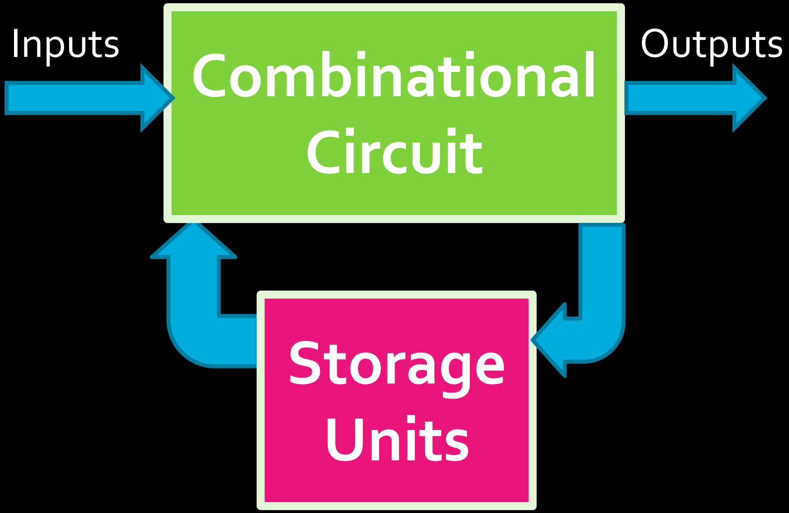


$$\begin{tikzcd}
 & 0 \arrow[dr] & \\
3 \arrow[ur] & & 1 \arrow[dl] \\
 & 2 \arrow[ul] &
\end{tikzcd}$$

Creating sequential circuits

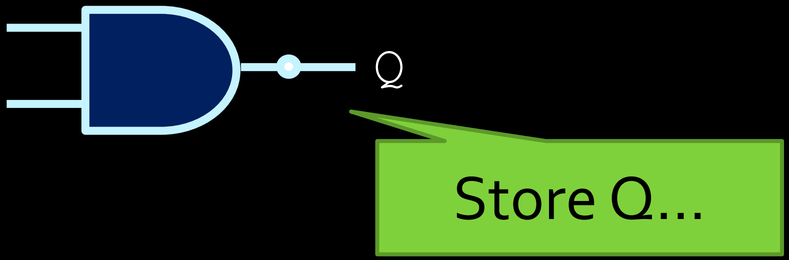
Key idea:
split the sequential circuit to two parts.

- A bunch of flip-flops to store state.
  - This week
- A combinatorial circuit to handle the logic to manage state and output.
  - More next week.

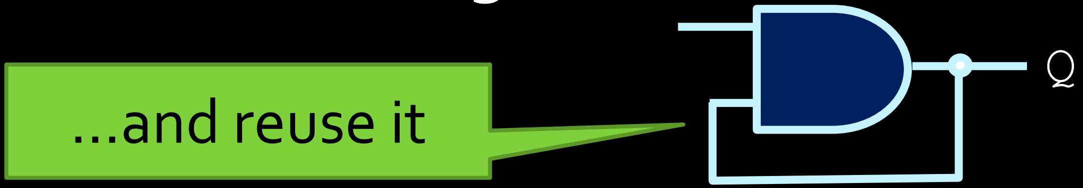


Feedback for Storing State

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

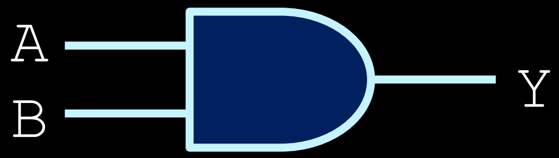


- Does the following work?

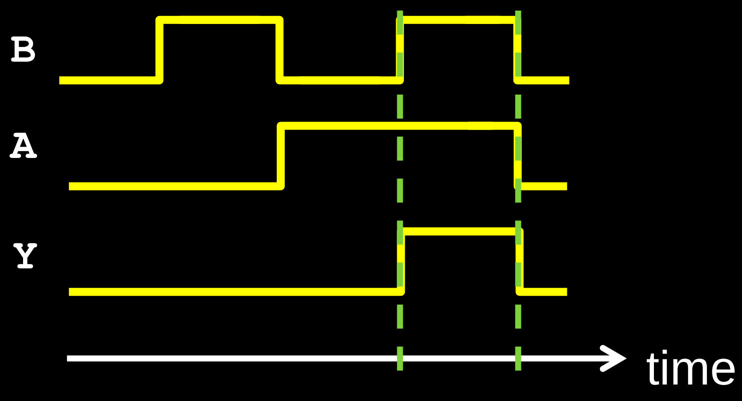


- How do we reason about this circuit?

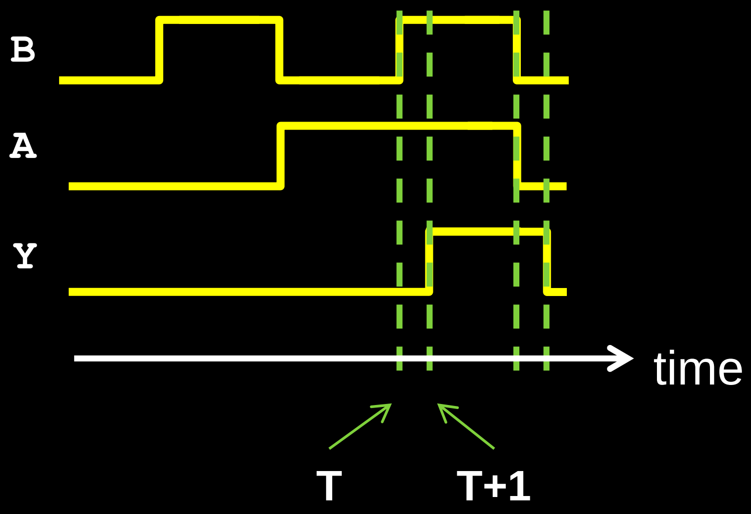
There is a Propagation Delay



Ideal

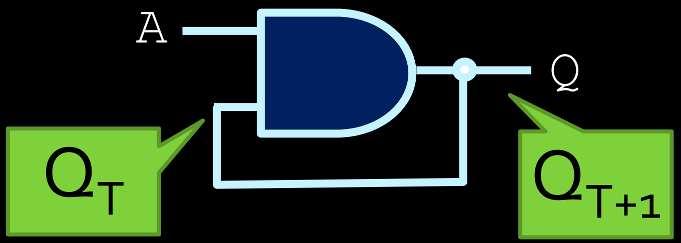


Considering delays



Feedback Circuit Example (AND)

- Let's analyze it

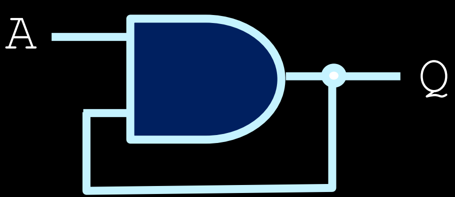


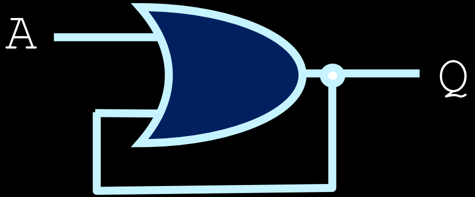
Q_T and Q_{T+1} represent the values of Q at a time T, and a point in time immediately after (T+1)

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

Feedback Circuit Examples

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





Stuck at 0

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

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

Stuck as 1

Feedback behaviour

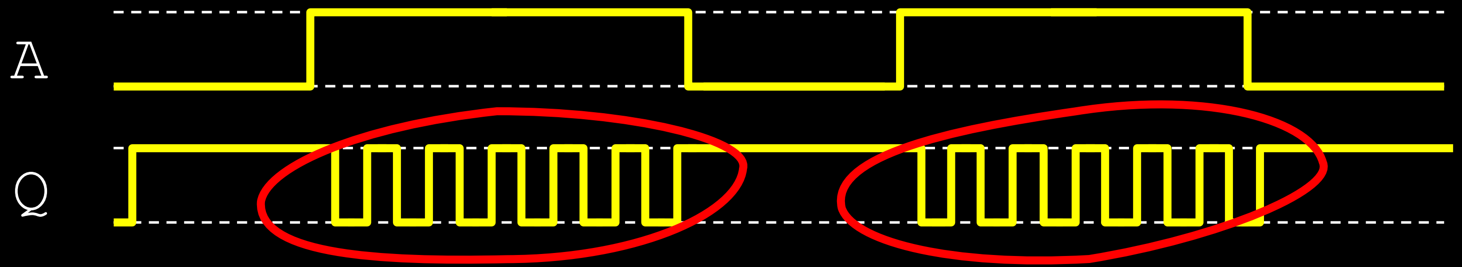
- NAND behaviour

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

- NOR behaviour

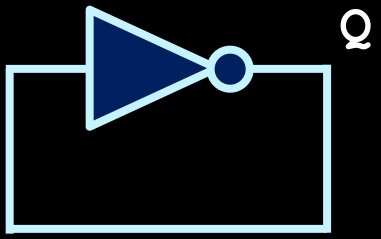
A	Q_T	Q_{T+1}
0	0	1
0	1	0
1	0	0
1	1	0

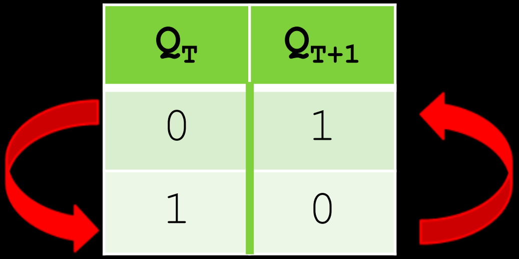
- Output Q_{T+1} can be changed, based on A.
- They are unstable – oscillations.



Question #1

- Is this circuit stable or does it oscillate?

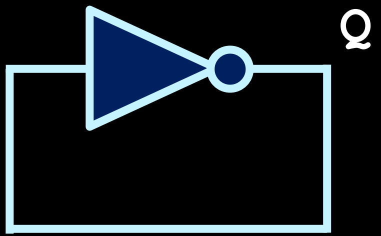




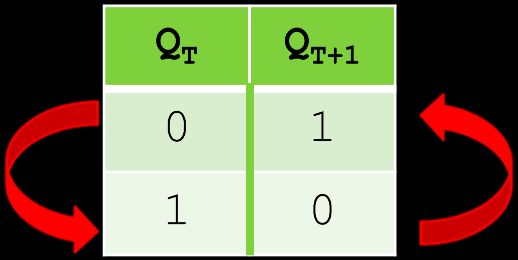
Q_T	Q_{T+1}
0	1
1	0

Question #2

- Is this circuit stable or does it oscillate?



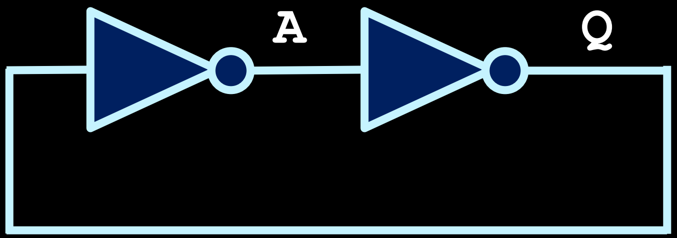
- It oscillates!



Q_T	Q_{T+1}
0	1
1	0

Question #2

- Is this circuit stable or does it oscillate?

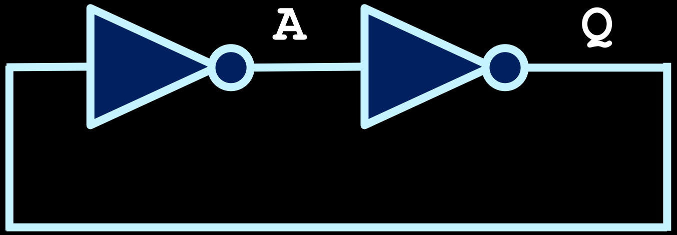


Q_T	A_T
0	1
1	0

A_T	Q_{T+1}
1	0
0	1

Question #2

- Is this circuit stable or does it oscillate?

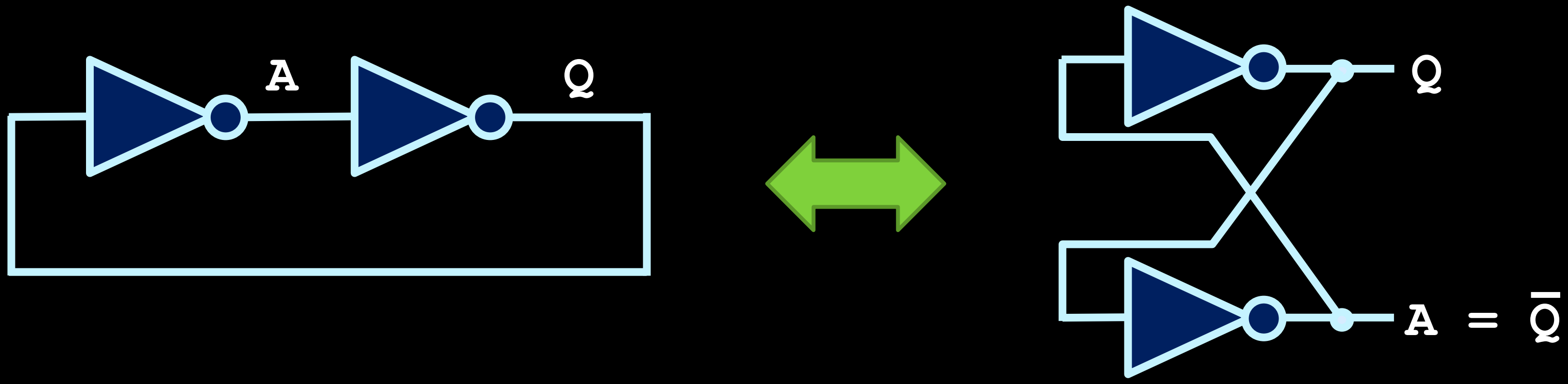


- It is stable!
  - Q=0 remains 0
  - Q=1 remains 1

Q_T	A_T
0	1
1	0

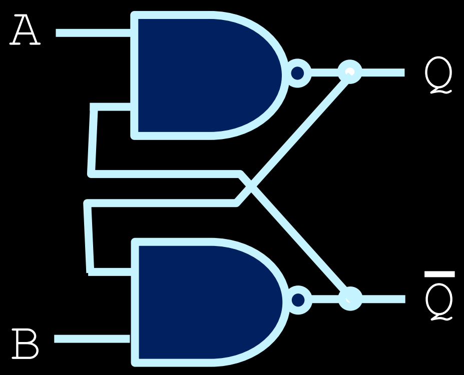
A_T	Q_{T+1}
1	0
0	1

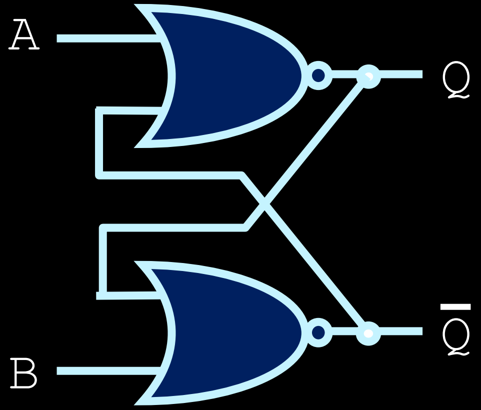
Rearrange the circuit…



Latches

- We overcome oscillations by combining multiple NAND or NOR gates.

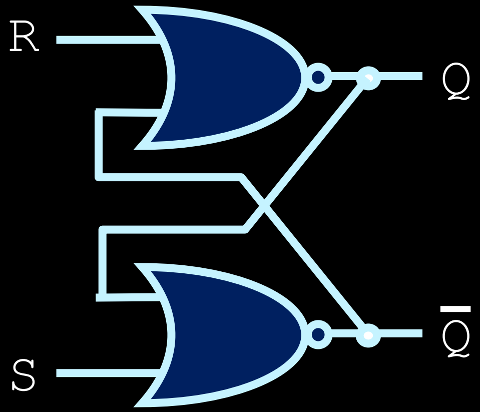




- These circuits are called latches.

Question #3

- Complete the truth table
  - Don’t-care inputs allowed.
- And name the 4 possible S,R input combinations



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

← Hold

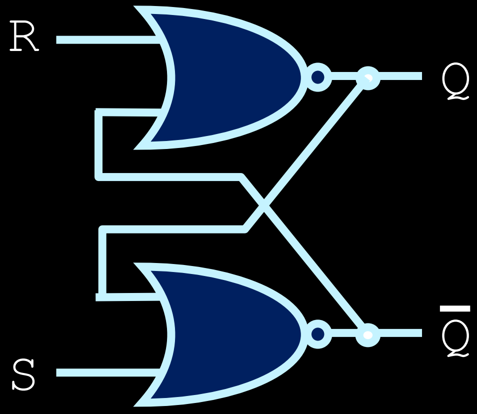
← Reset

← Set

← Forbidden

Question #3

- Complete the truth table
  - Don't-care inputs allowed.
- And name the 4 possible S,R input combinations

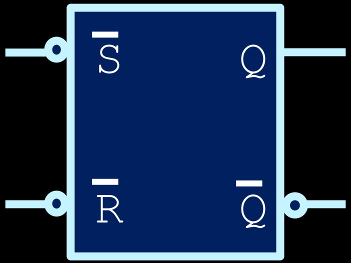


S	R	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$	
0	0	0	1	0	1	← Hold
0	0	1	0	1	0	
0	1	X	X	0	1	← Reset
1	0	X	X	1	0	← Set
1	1	X	X	0	0	← Forbidden

Summary: S'R' and SR latches

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

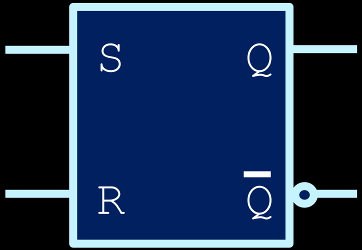
$\overline{S}$	$\overline{R}$	Q_T	$\overline{Q}_T$	Q_{T+1}	$\overline{Q}_{T+1}$	
0	0	X	X	1	1	Forbidden state
0	1	X	X	1	0	Set Q to 1
1	0	X	X	0	1	Reset Q to 0
1	1	0	1	0	1	Maintain Q
1	1	1	0	1	0	



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

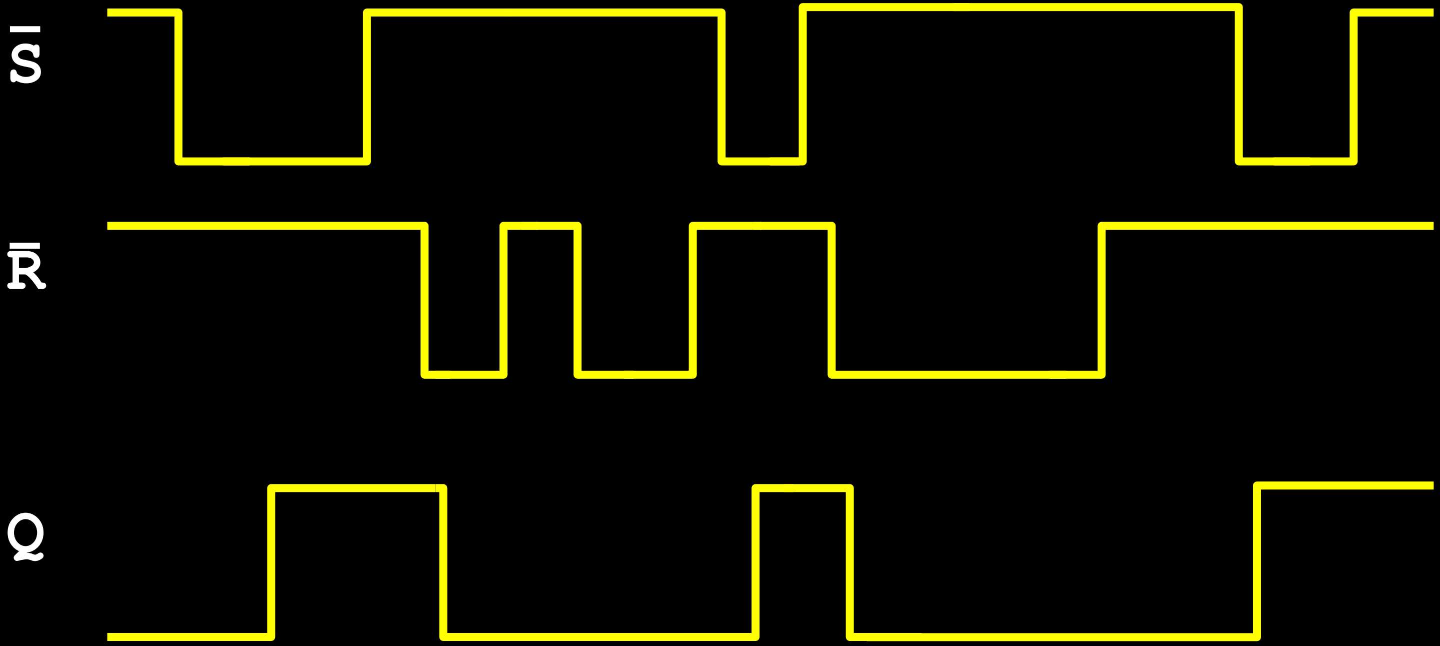
SR-latch

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



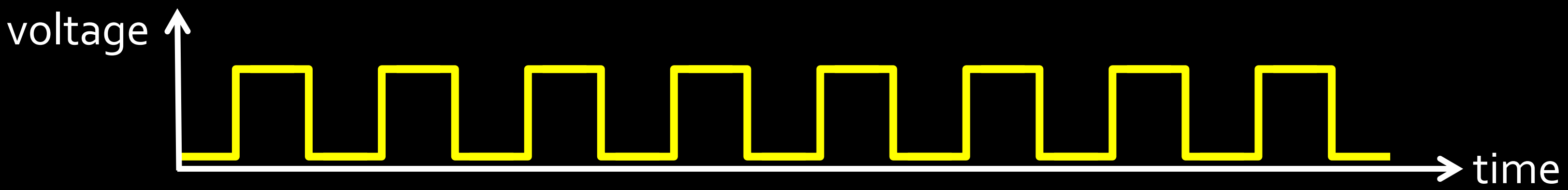
Question #4

- Given the input waveforms, sketch the output Q of an $\overline{S}\overline{R}$ latch. Assume Q was zero initially.

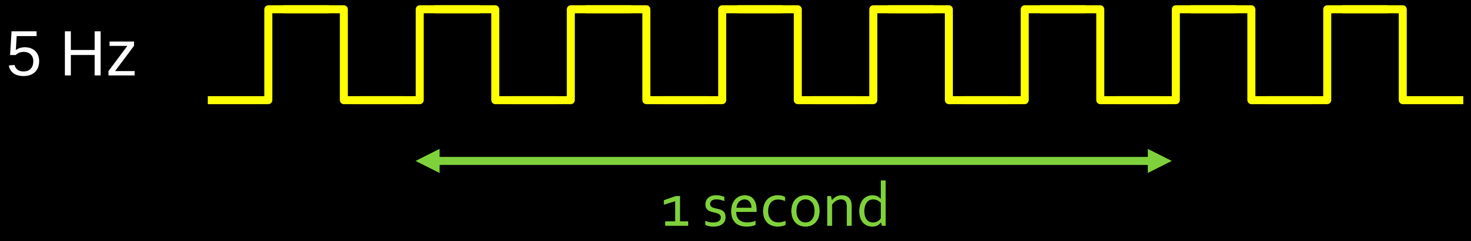


Clocks

- A periodic signal that gives timing for our circuit



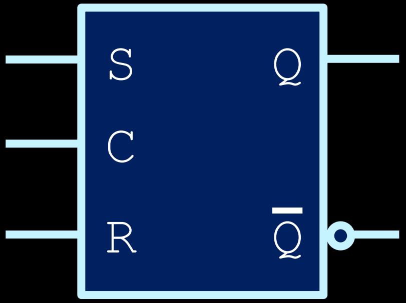
- Frequency = the number of pulses occur per second.



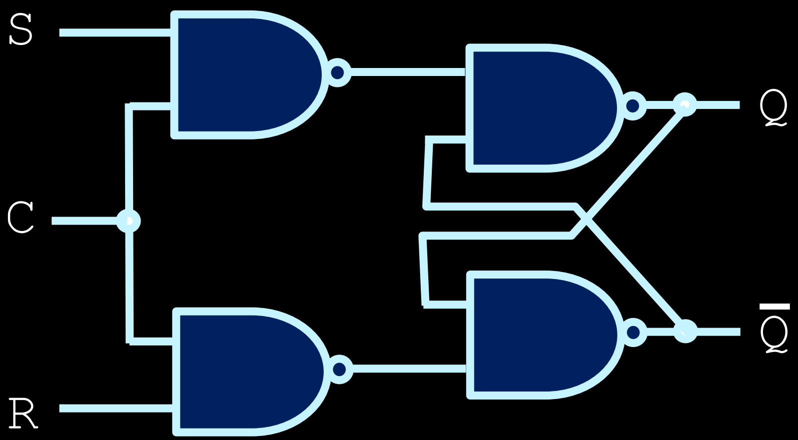
Clocked/Gated latches

- Add a new input C that acts like a “gate” over the inputs:

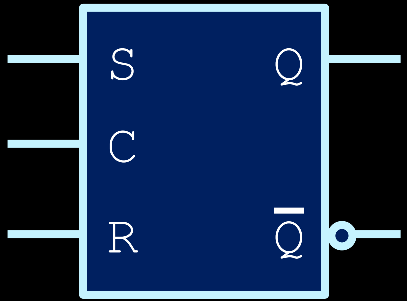
- If `C=0` the latch ignores input
  - Maintains state.
- If `C=1` the latch is “active”
  - Responds to input



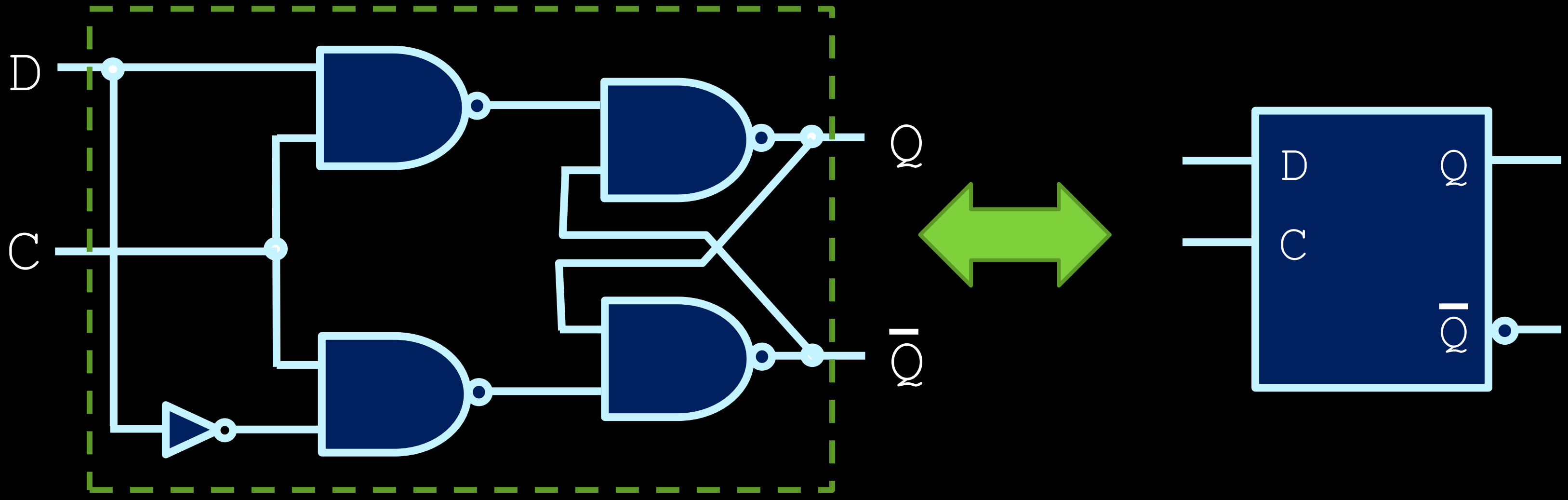
Clocked SR latch behaviour



C	S	R	Q_{T+1}	Result
0	X	X	Q_T	hold
1	0	0	Q_T	hold
1	0	1	0	Reset
1	1	0	1	Set
~~1~~	~~1~~	~~1~~	?	undefined



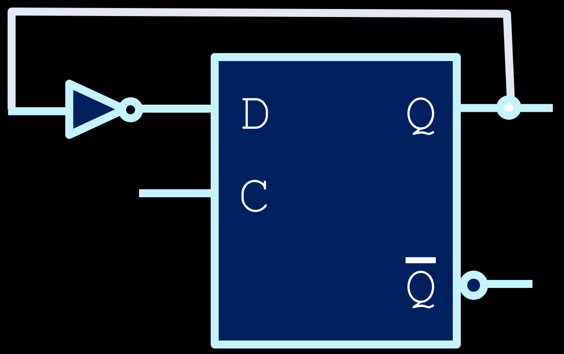
D latch

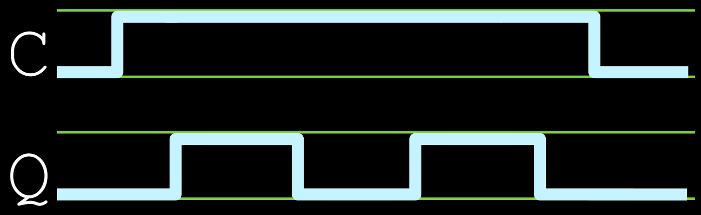


- This design is good!
  - Easy to store a bit: just set D to what you want to store.
  - Can maintain state as long as C is low
  - No weird forbidden inputs.

D latch is transparent

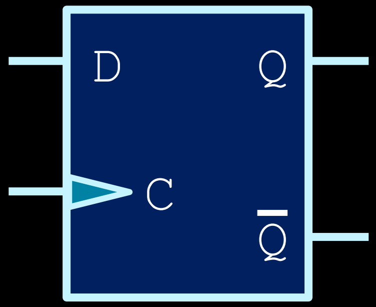
- Any changes to its inputs are visible to the output when control signal (Clock) is 1.
  - Output keeps toggling back and forth.
- We want output to change exactly once per cycle.



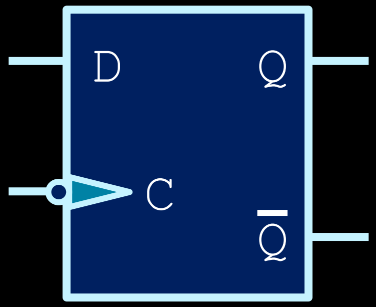


Flip-flops

- Positive edge:
  triggered on rising edge of the clock

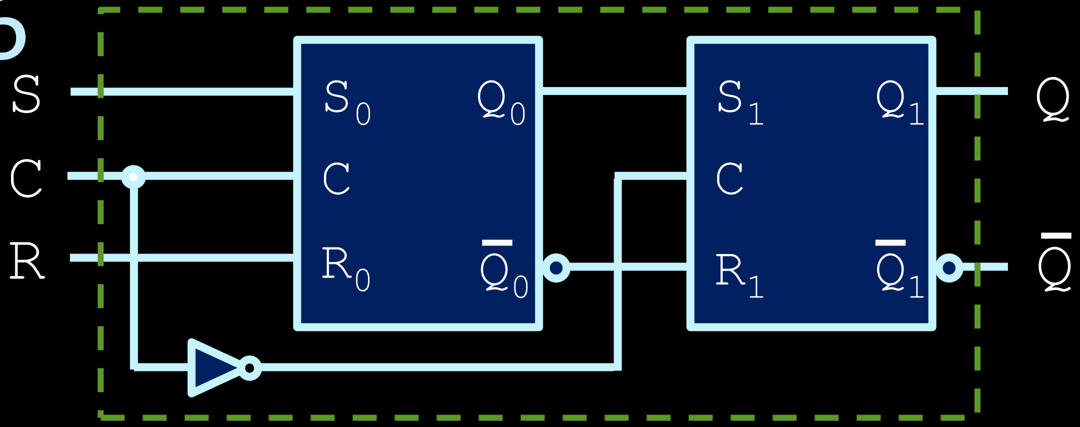


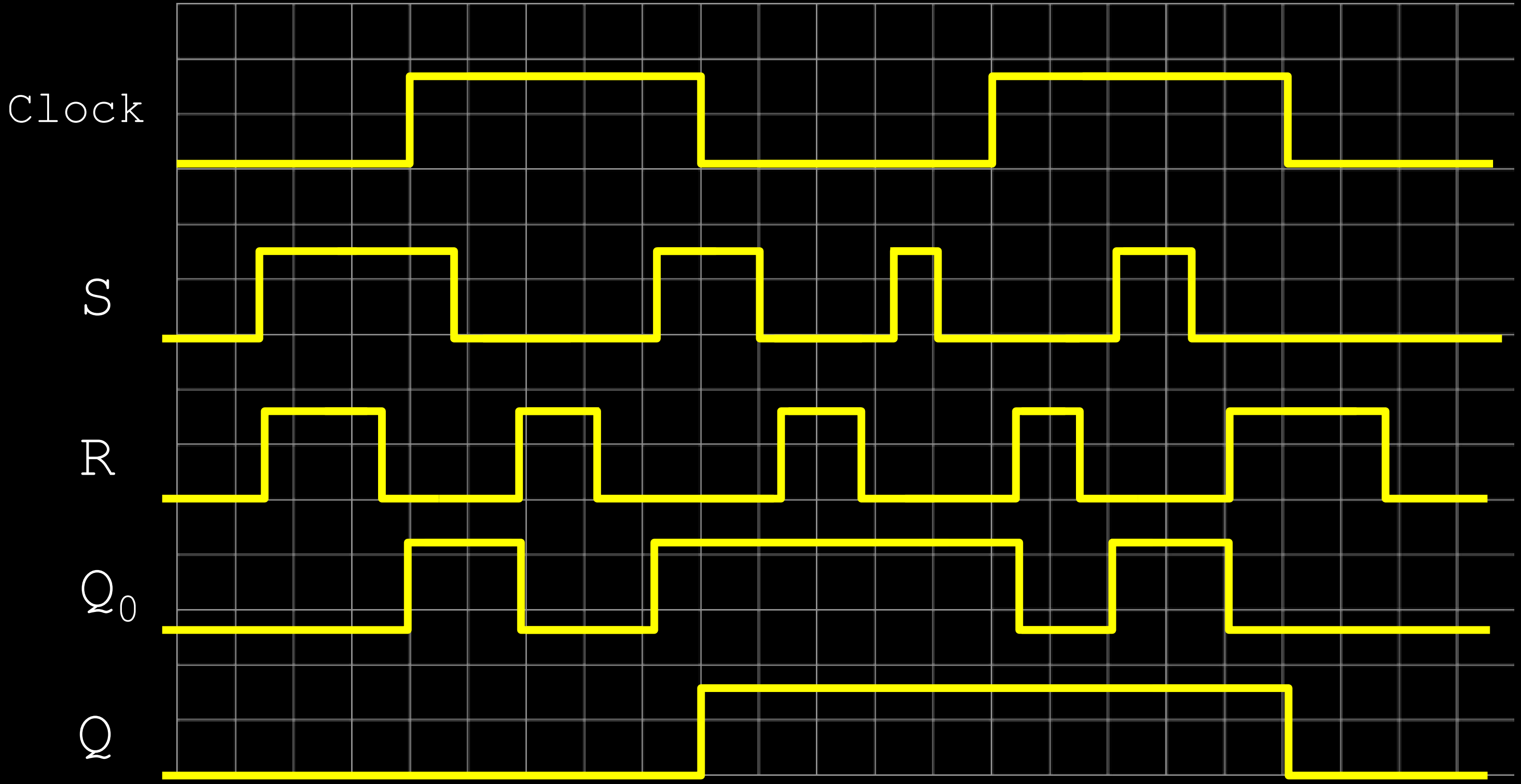
- Negative edge:
  triggered on falling edge of the clock



Question #6

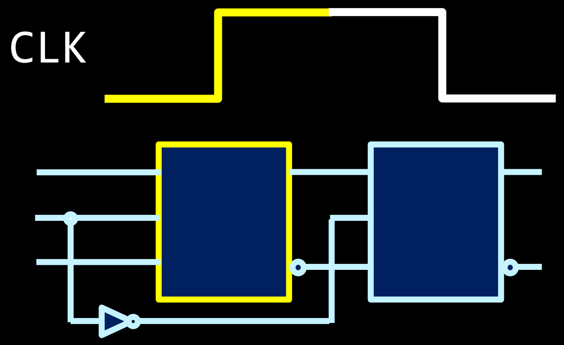
- Assuming all inputs start low, complete the timing diagram

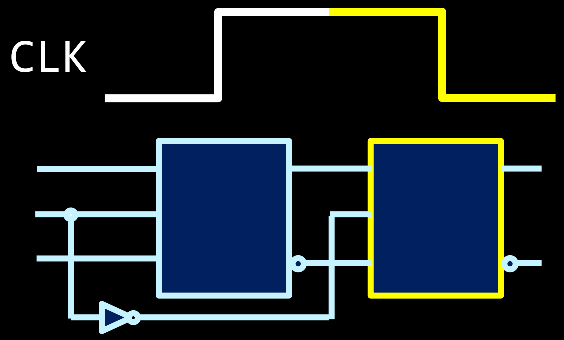




Flip-flops

- For input to propagate to output, it takes each of the latches to be active once.
- First latch changes on "flip".
- Output can only change upon "flop", which is basically the falling edge of the clock signal
- At most one change per clock cycle

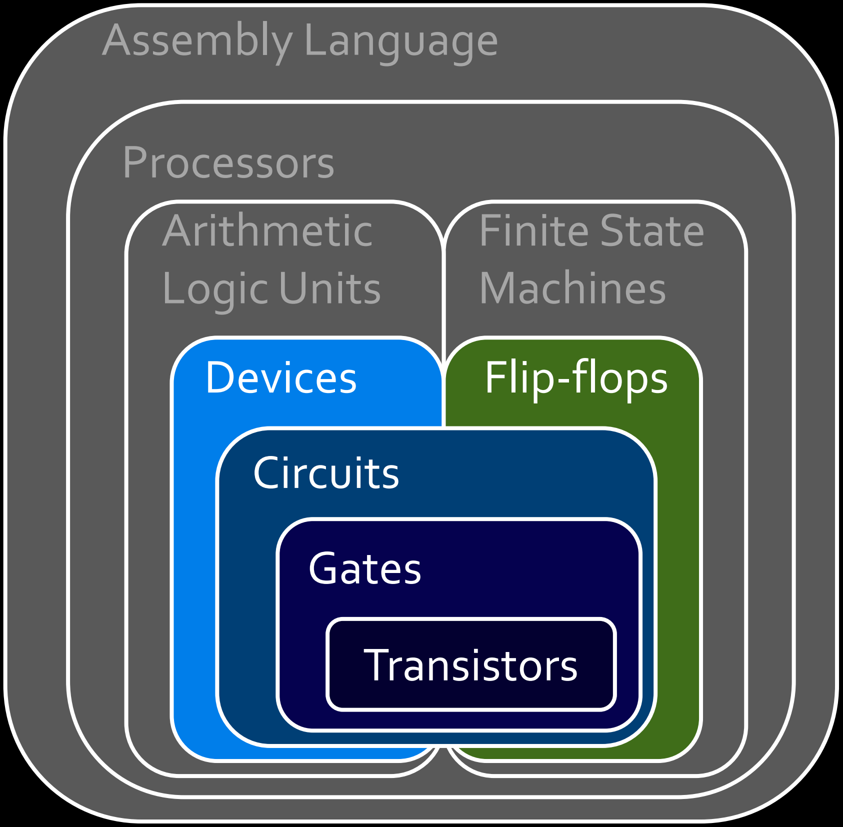


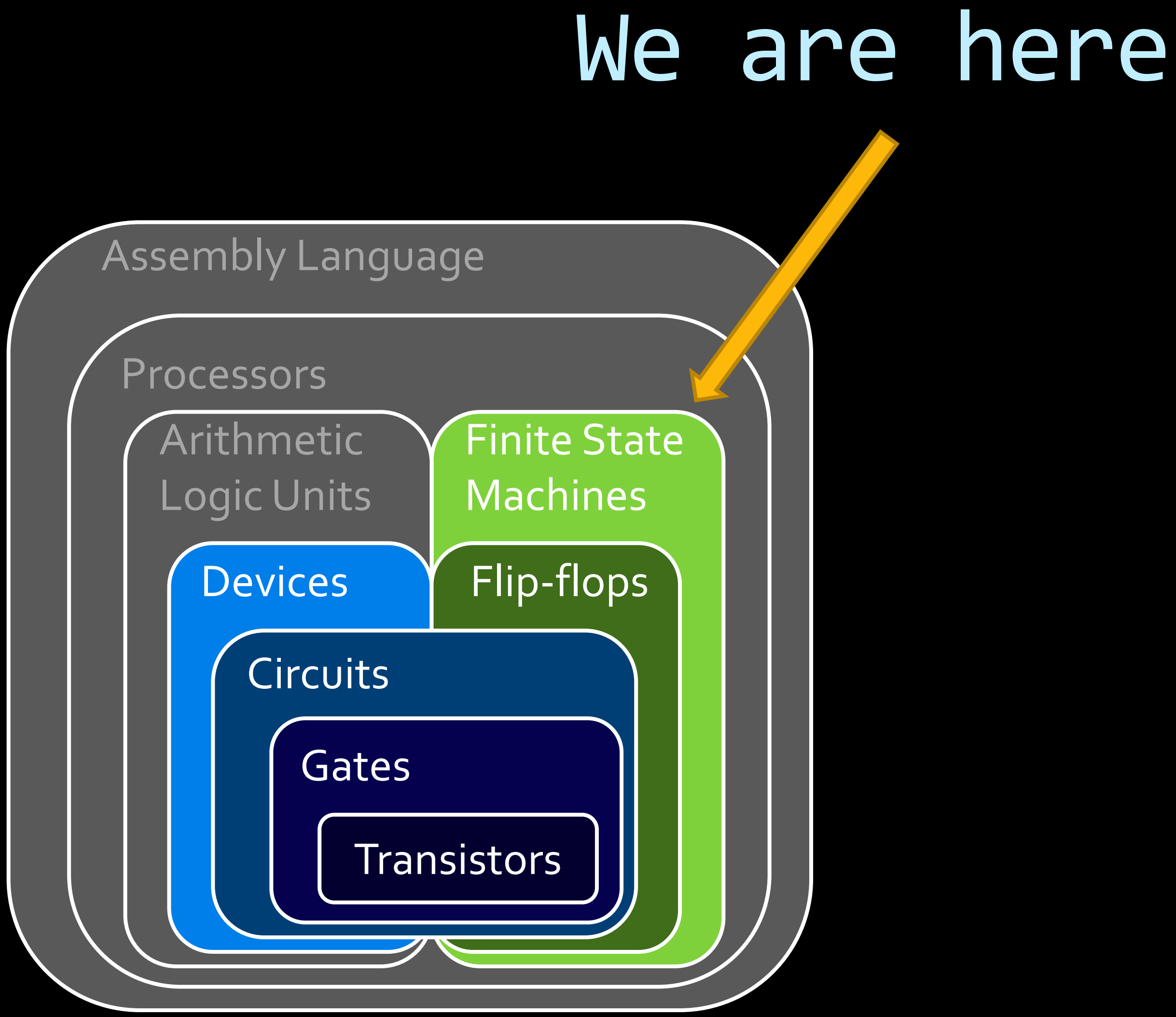


Flip-flops

- We have:
  - D flip-flops (most common type!)
  - SR flip-flops
  - T flip-flops (for “toggle”)
  - JK flip-flops

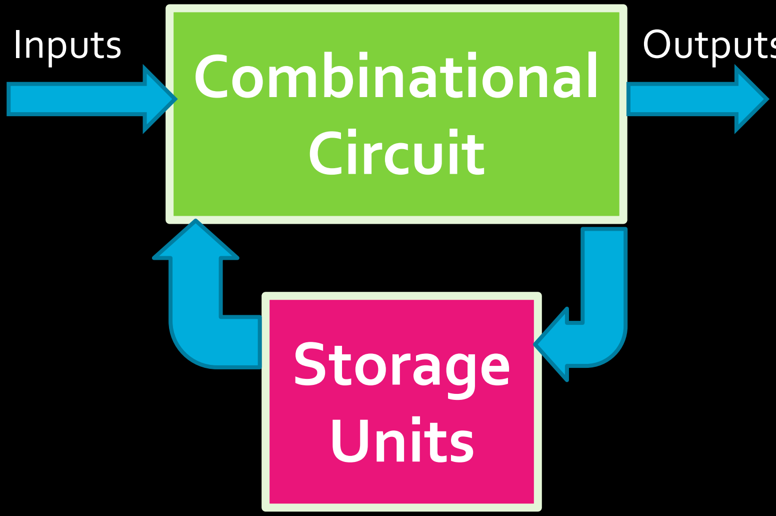
Week 5: Sequential Circuit Design Part A: registers


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


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

Circuits using flip-flops

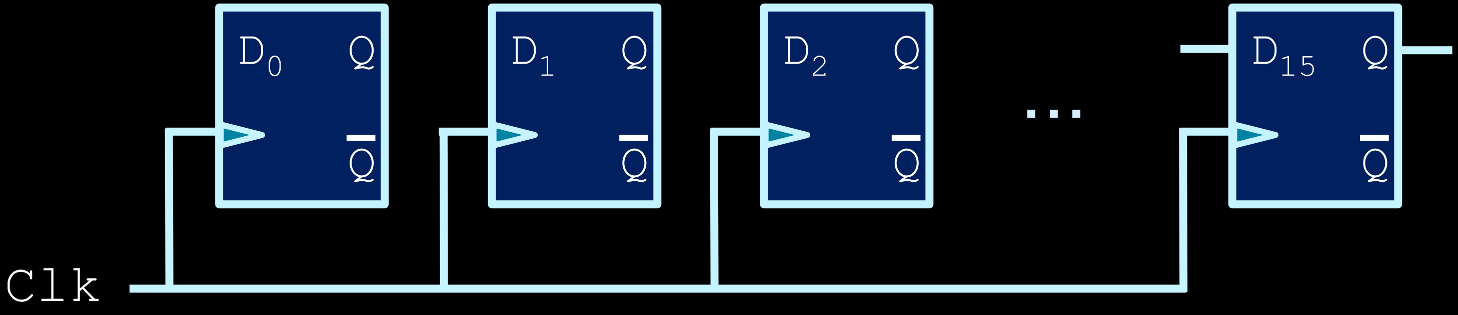
- Now that we know about flip-flops and what they do, how do we use them in circuit design?
- What's the benefit in using flip-flops in a circuit at all?



Example #1: Registers

Registers

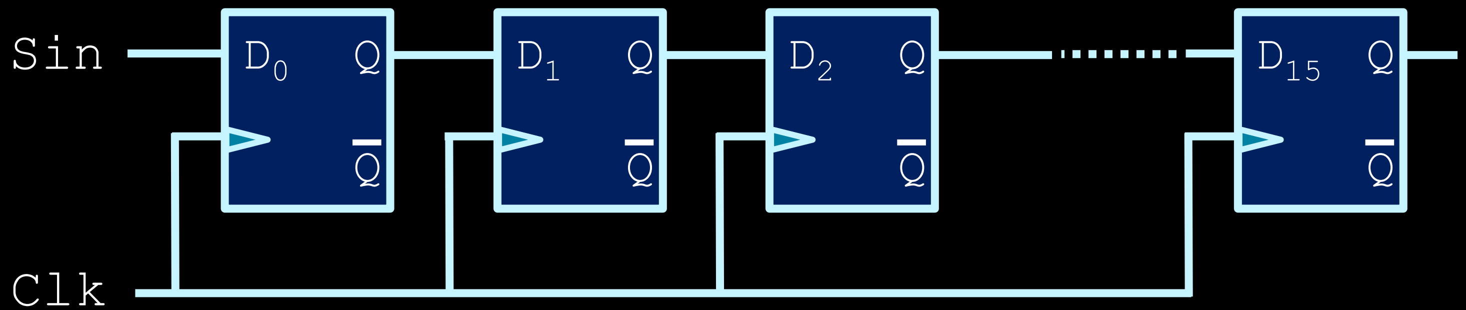
- An n-bit register: a bank of n flip-flops that share a common clock.
- Registers store a multi-bit value.



- All bits written at the same time.
- Key building block of sequential systems and CPUs.

Shift Registers

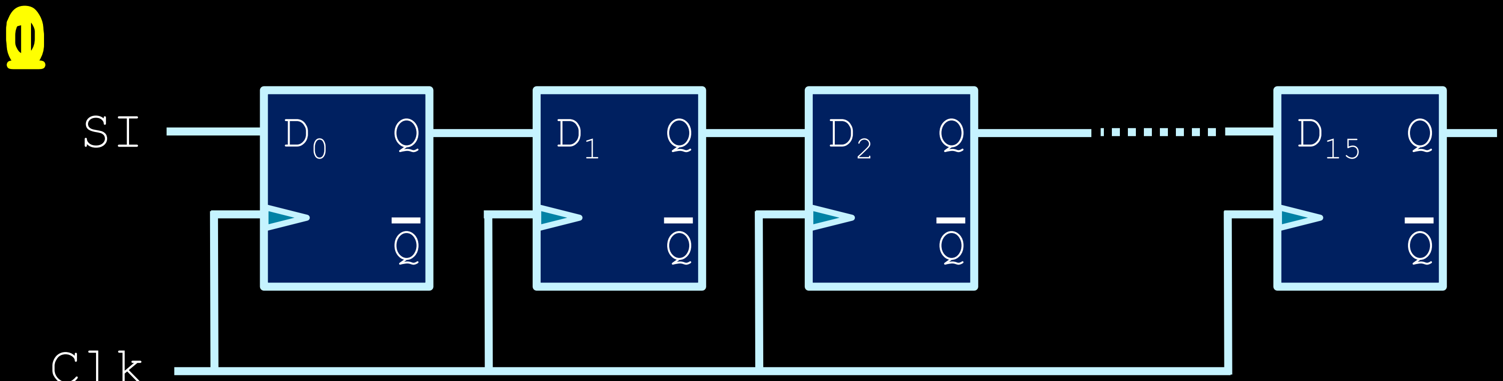
- A series of D flip-flops where output of flip-flop i is connected to input of i+1



- To set the value of an n-bit shift register, shift data into it one bit at a time, over n clock cycles.

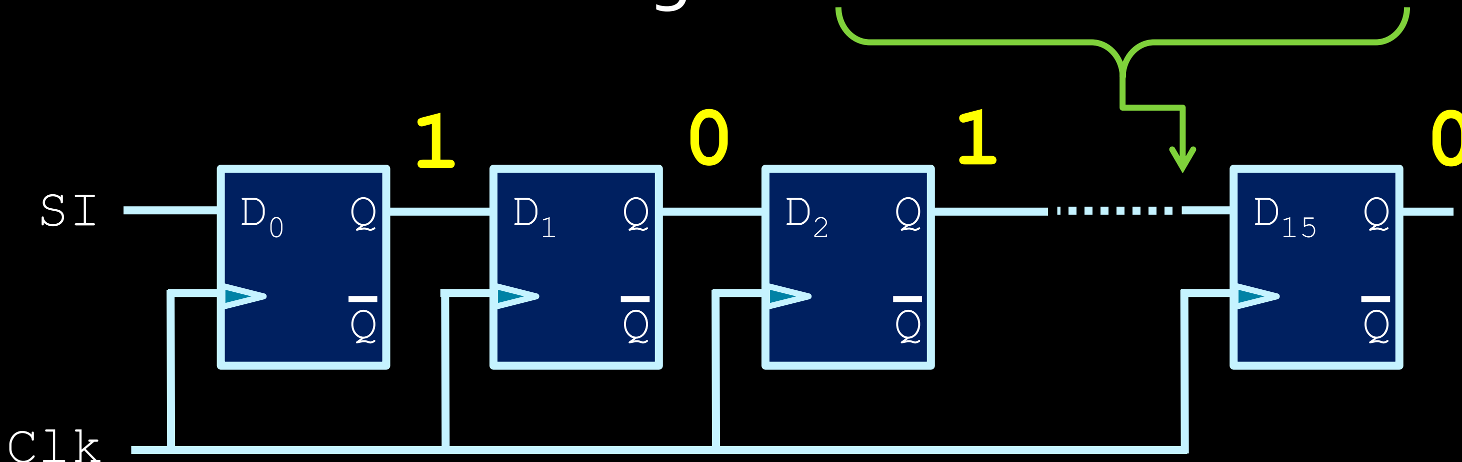
Shift registers

- Illustration: shifting in 0101010101010101



Shift registers

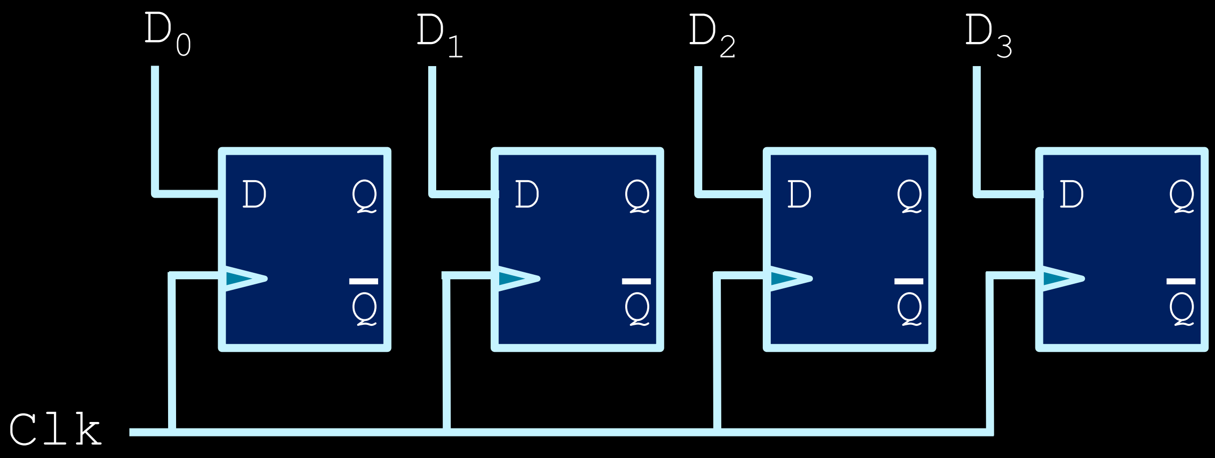
- Illustration: shifting in 0101010101010101



- After 16 clock cycles....

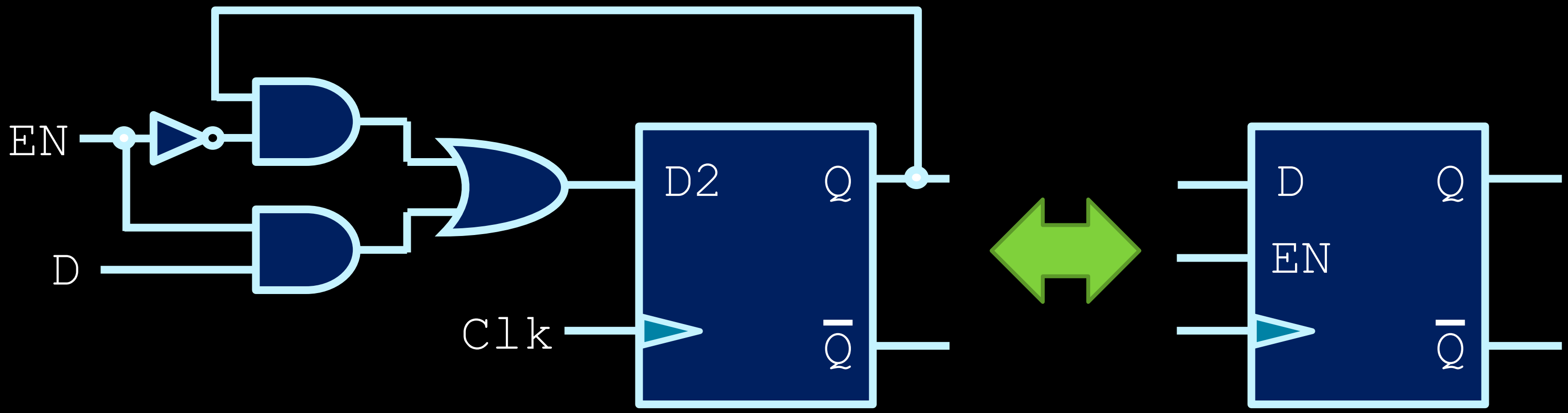
Load registers

- 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.



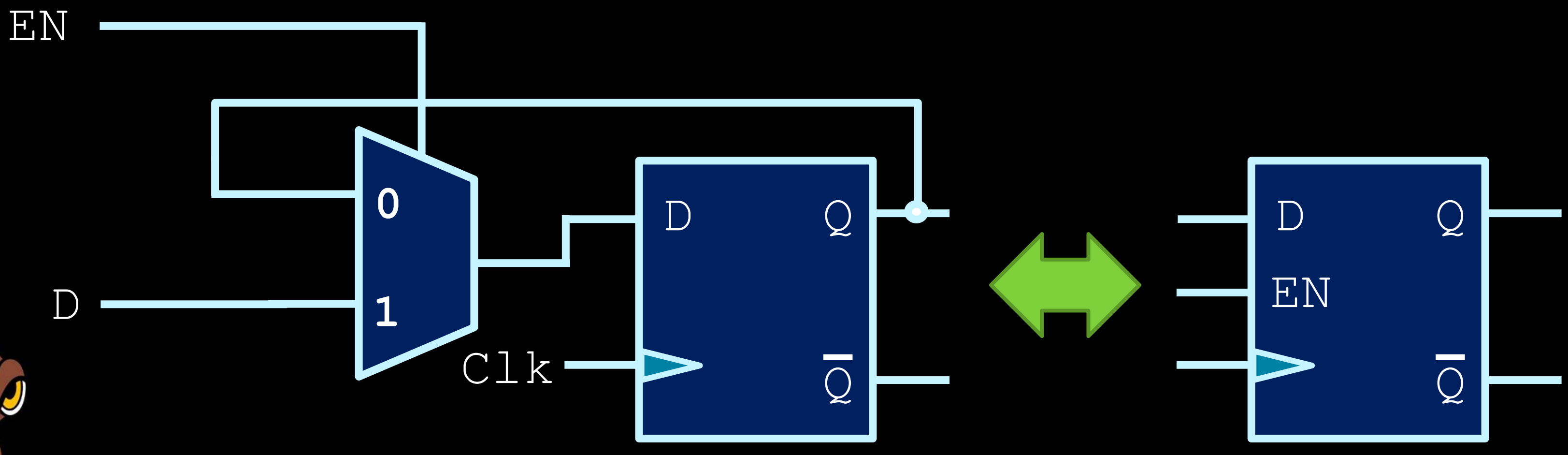
Load registers

- D flip-flop with enable: controls when the flip-flop is allowed to load D :
  - When EN = 1, D2 is whatever D is → load D
  - When EN = 0, D2 is whatever Q is → maintain Q

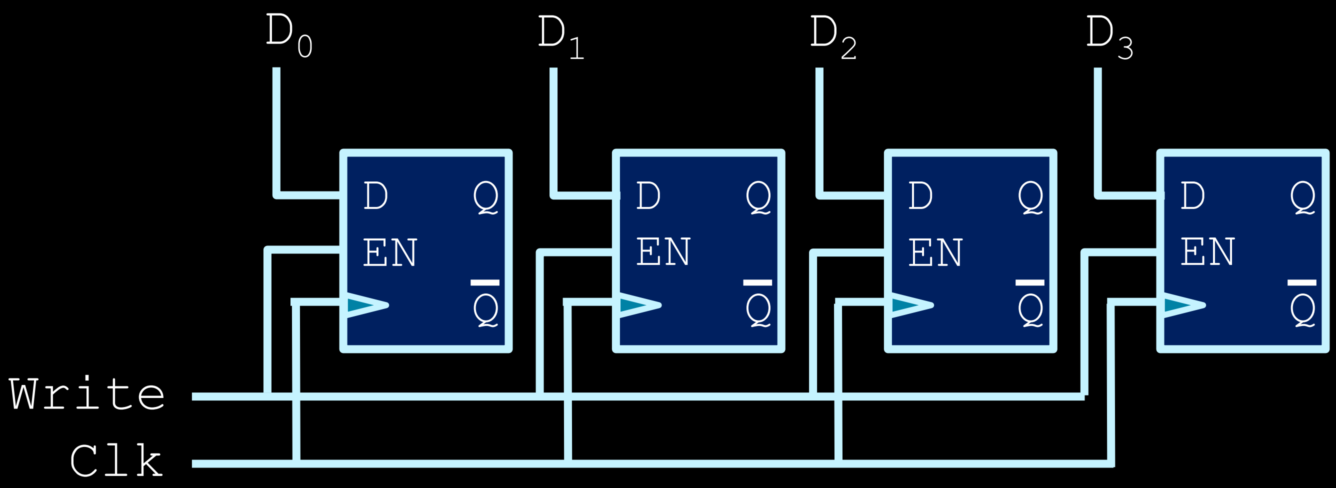


Load registers

- D flip-flop with enable: controls when the flip-flop is allowed to load D :
  - When EN = 1, D2 is whatever D is → load D
  - When EN = 0, D2 is whatever Q is → maintain Q



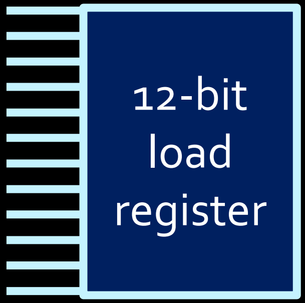
Load registers

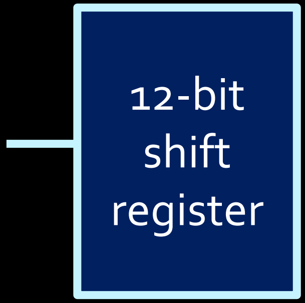


- A register implemented with these new D flip-flops will maintain the old value in the register until overwritten by setting EN high.

Shift vs Load

- Shift registers are serial – load bits one at a time
- Load registers are parallel – load all bits immediately.
- Load registers (a.k.a "registers") seem easier to work with... so why?
- Sometimes one wire is better than many.
  - Lack of space on circuit for many wires.
  - Clock skew on long wires, fast clocks.
  - Physics.

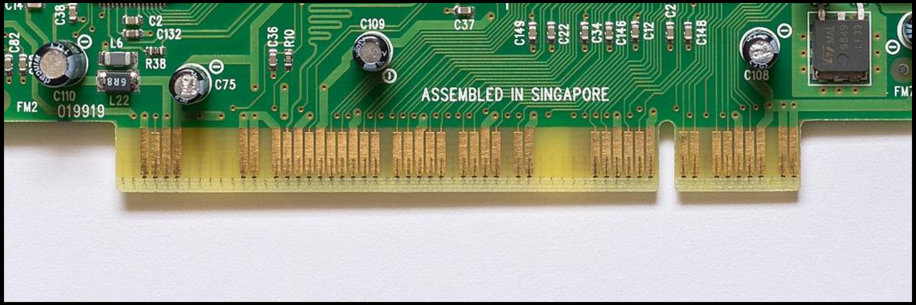




Serial vs Parallel

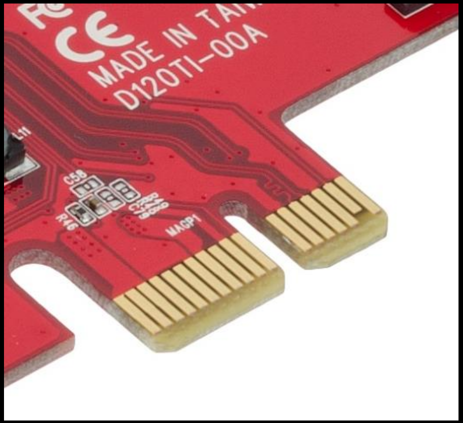
PCI

- Parallel
- Many pins



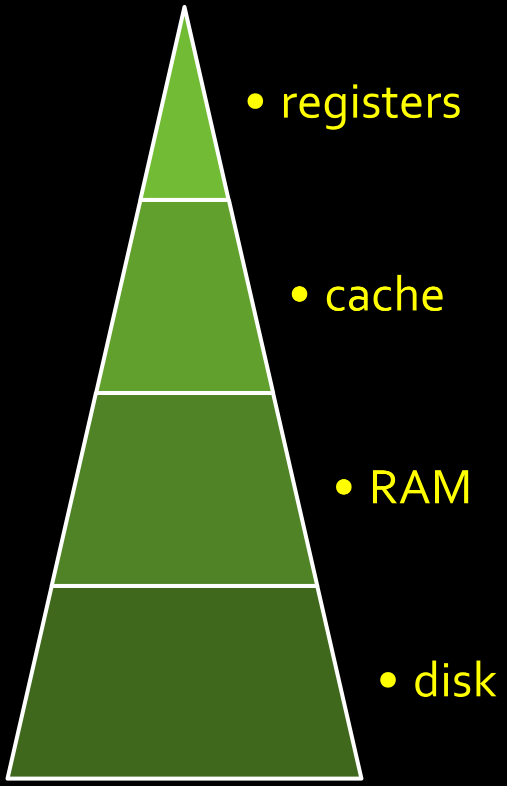
PCIe

- Serial
- Fewer pins… yet faster



Memory Hierarchy

- In computer architecture, registers are the first level in the memory hierarchy.
- The CPU's most local, fastest storage
  - 30+ of them on-chip.
- They are the memory units that the CPU interacts with directly for computation.
  - Anything else is too far, and is mediated by registers.
- Higher levels: cache, RAM, disk, etc.

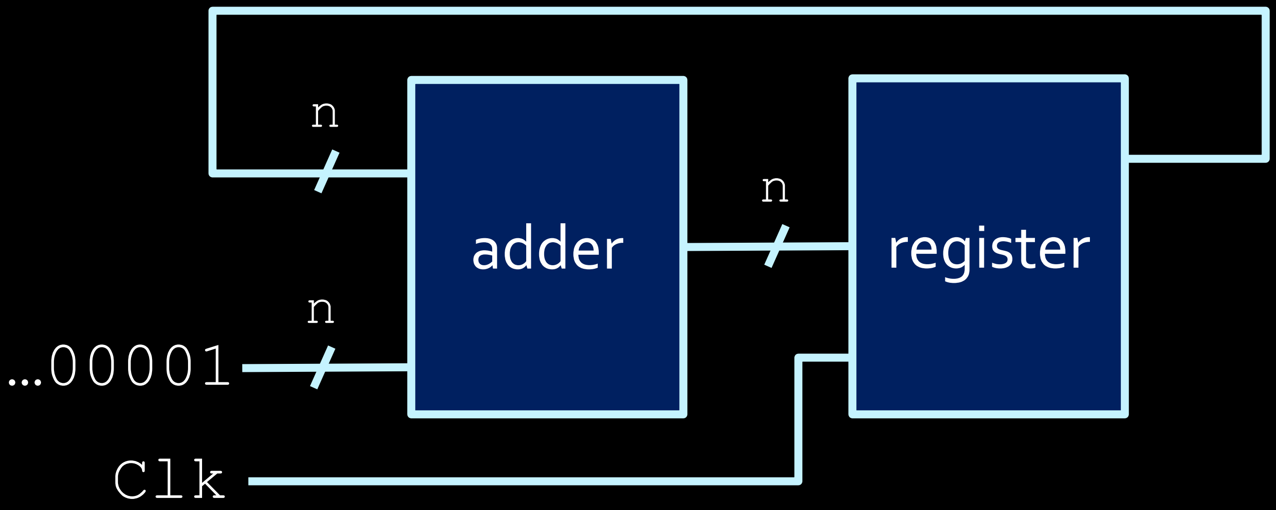


- What else can we build?

Week 5, part B: Counters

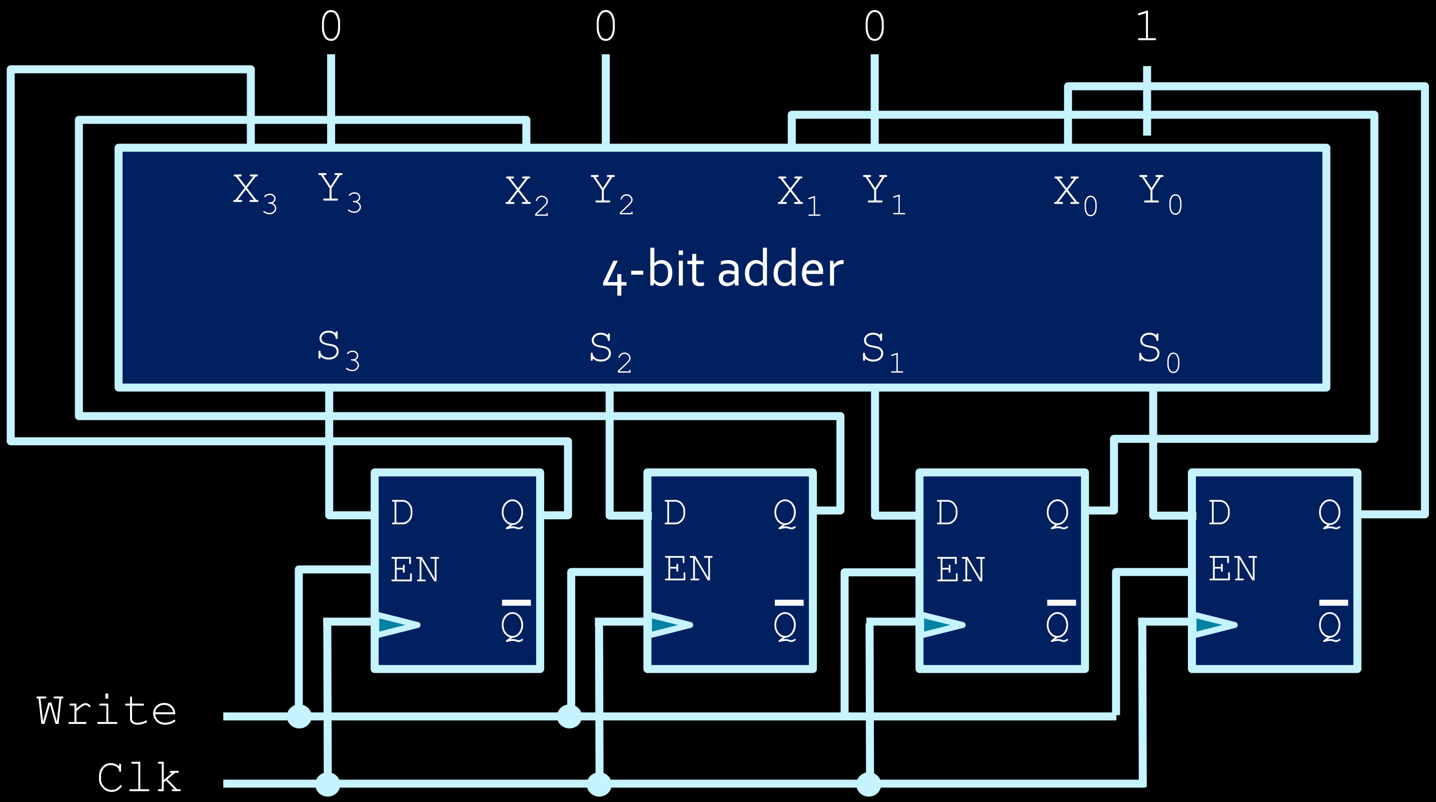
Idea for A Counter

- Load register + Ripple Carry Adder



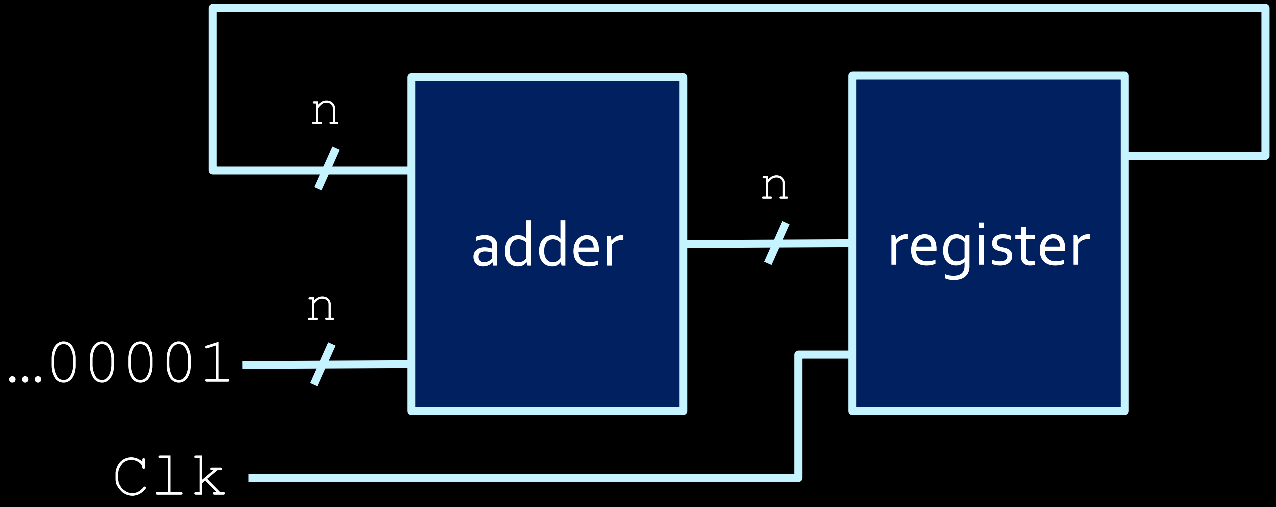
Idea for A Counter

- Example: 4-bit counter



Idea for A Counter

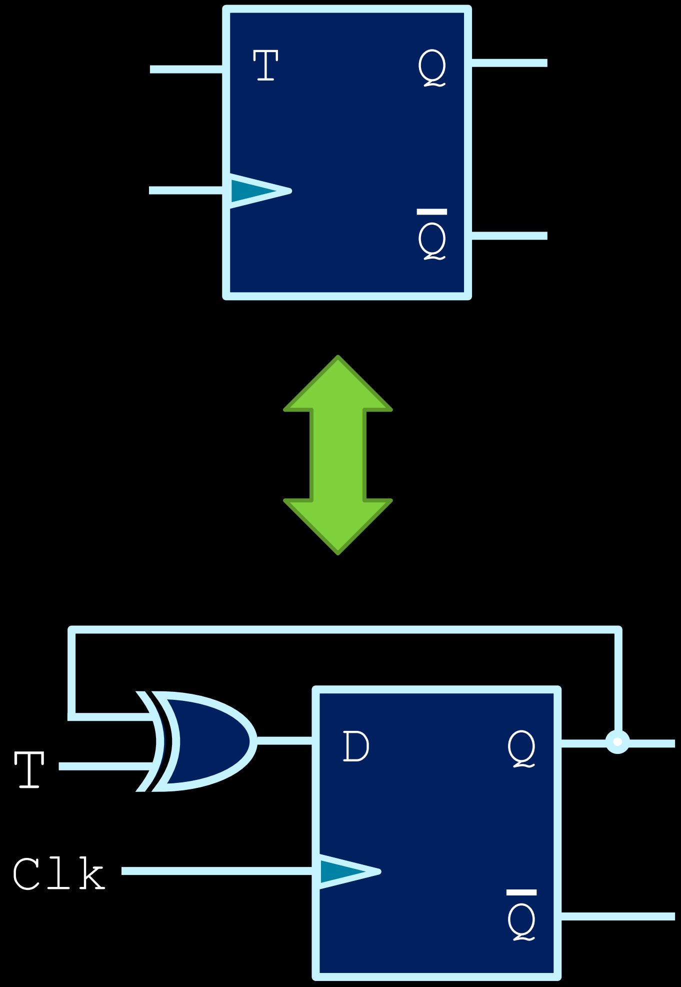
- Will this work?



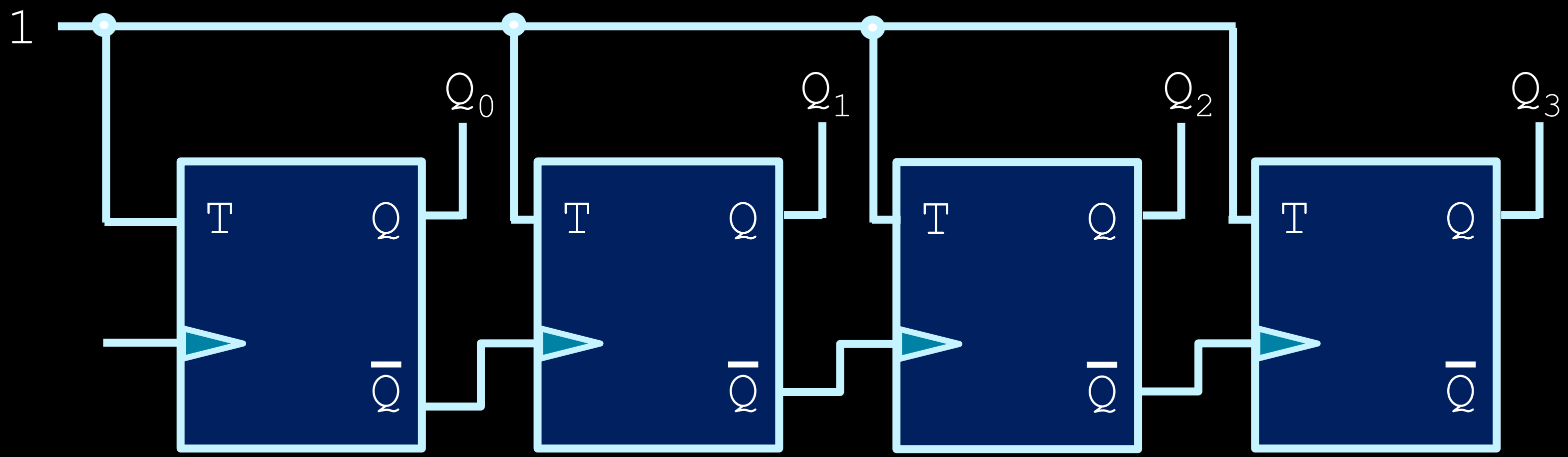
- Yes, but lots of gates needed for the adder.

A better idea

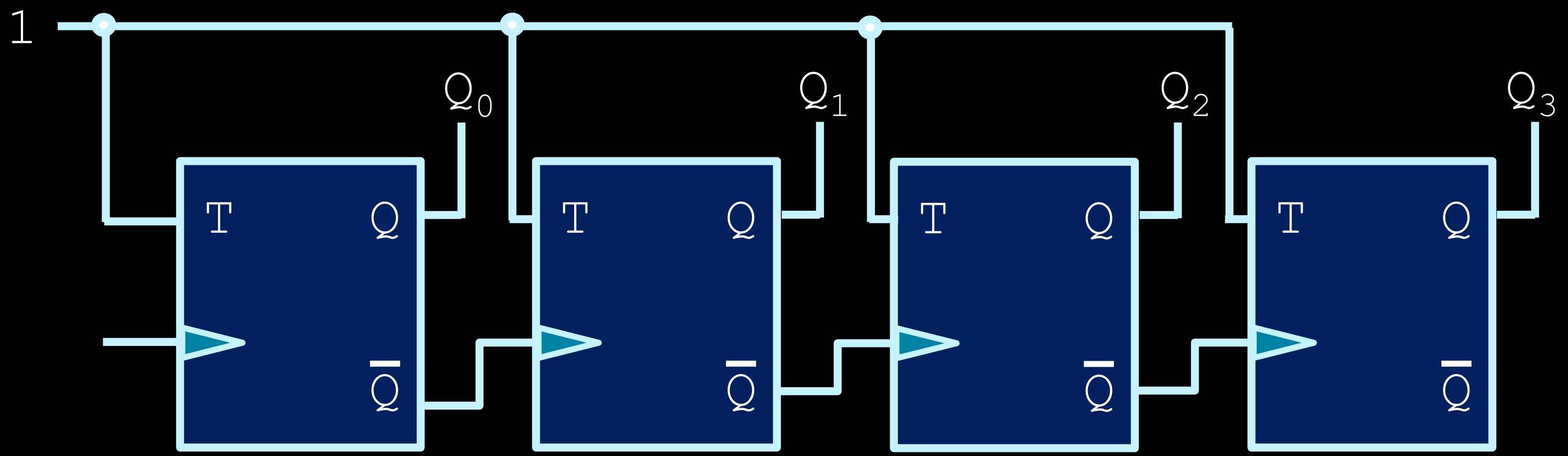
- Consider the T flip-flop:
  - Output is inverted when input T is high.
- What happens when a series of T flip-flops are connected together in sequence?
- More interesting: Connect the *output* of one flip-flop to the clock input of the next!



Counters



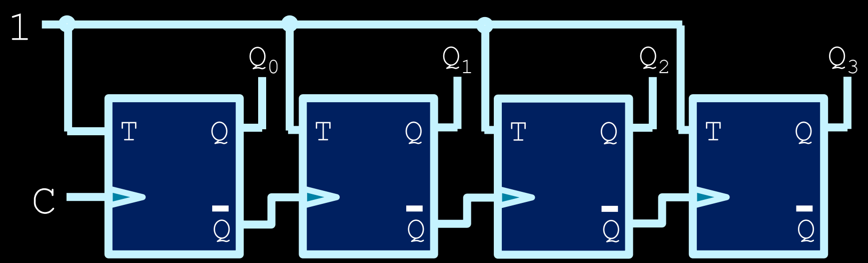
Counters

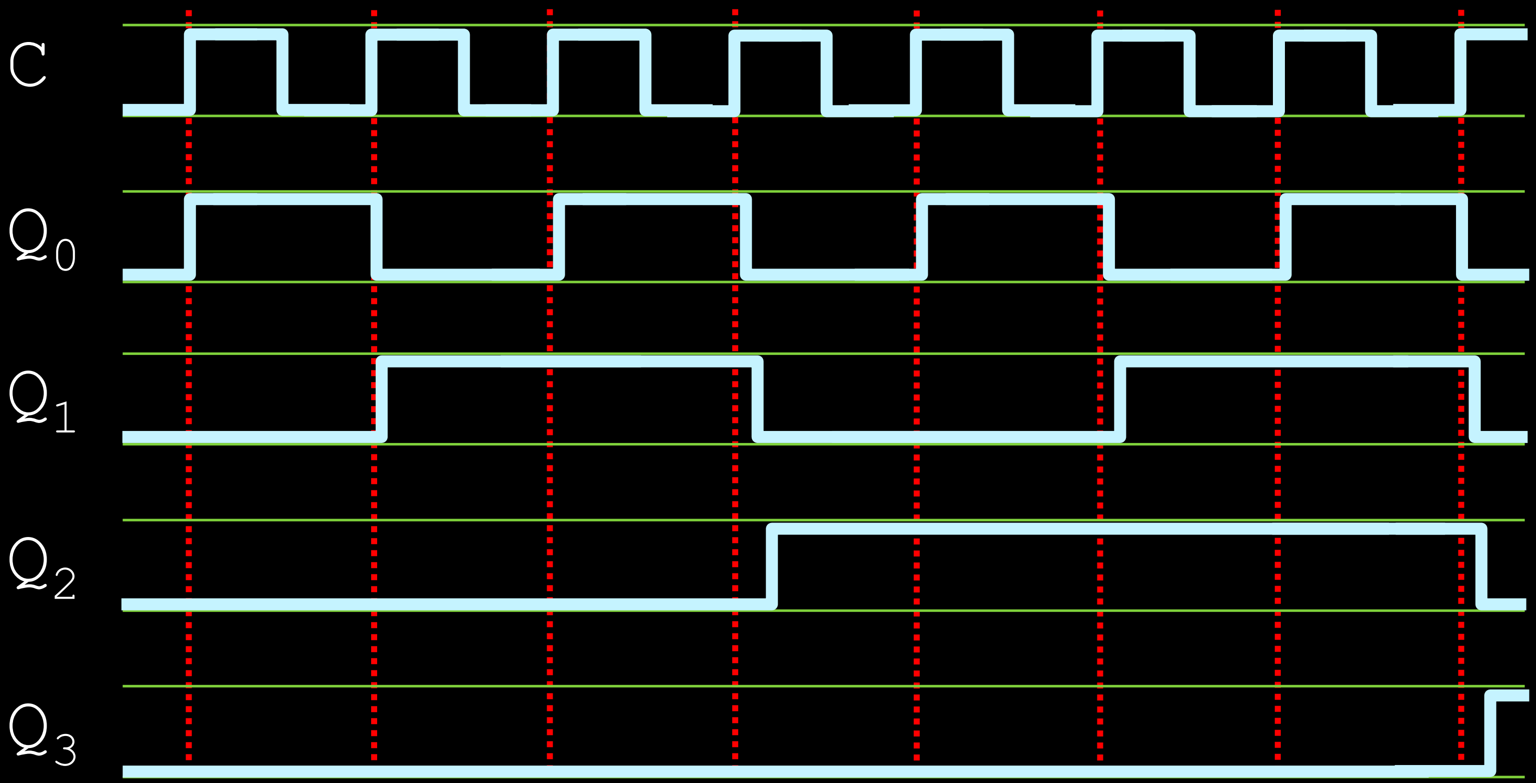


- This is a 4-bit ripple counter.
- Let's see how it works

Counters

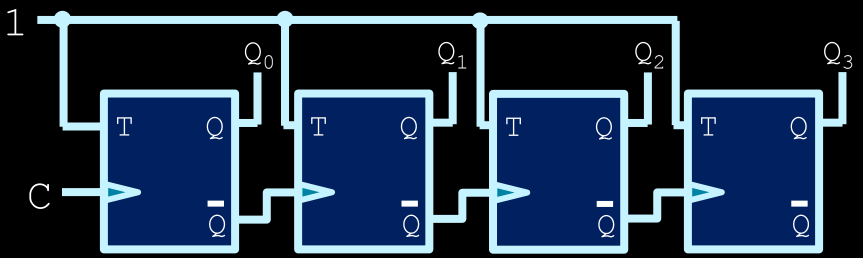
- Timing diagram

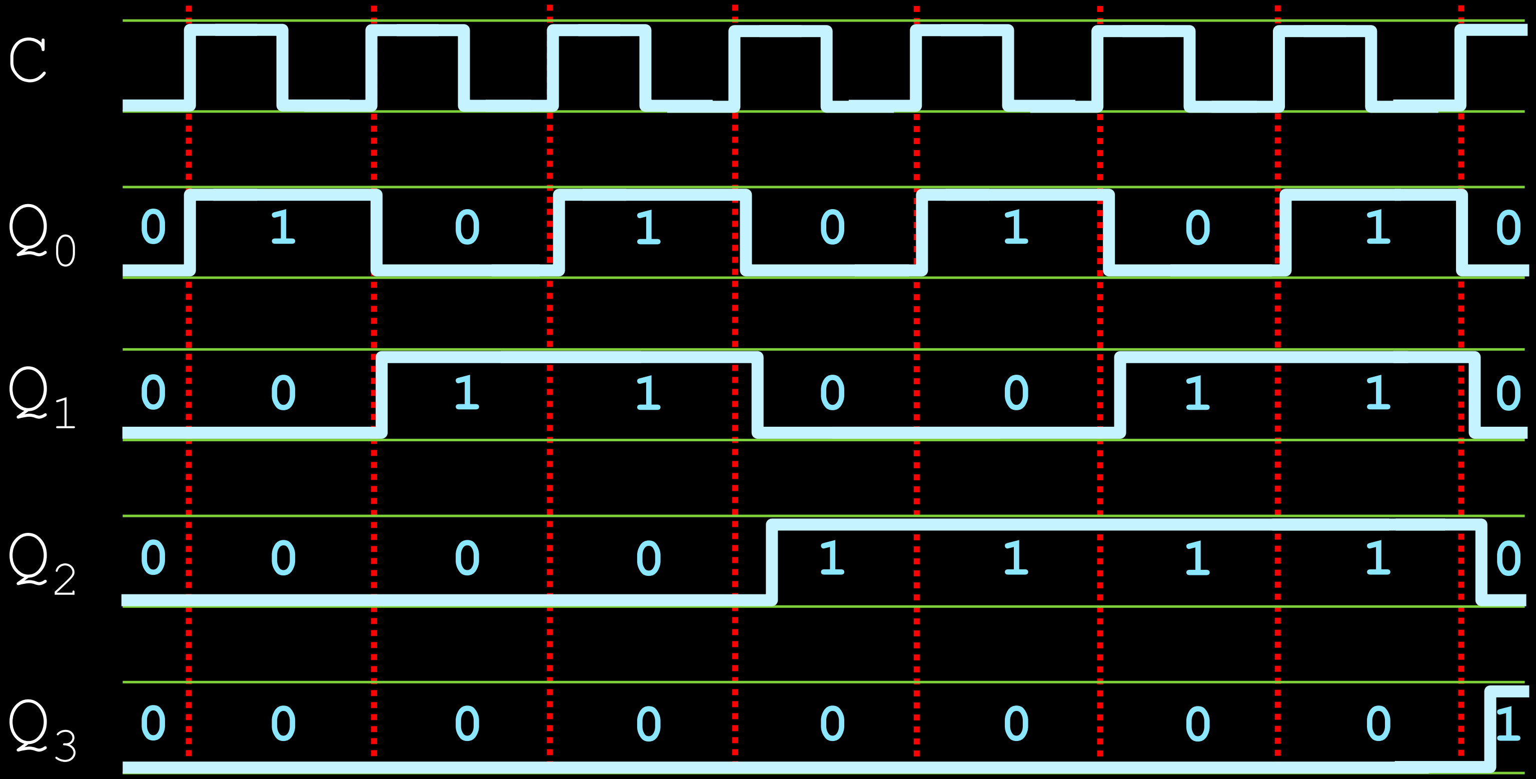




Counters

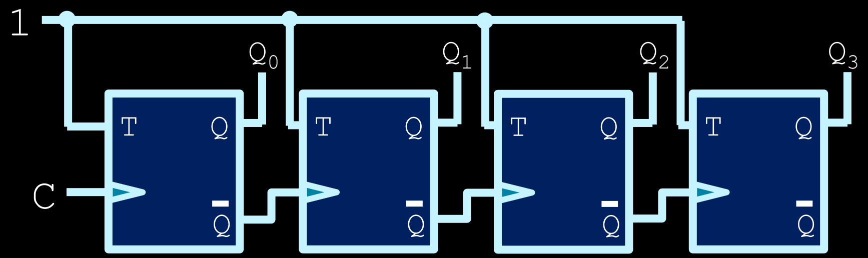
- Timing diagram





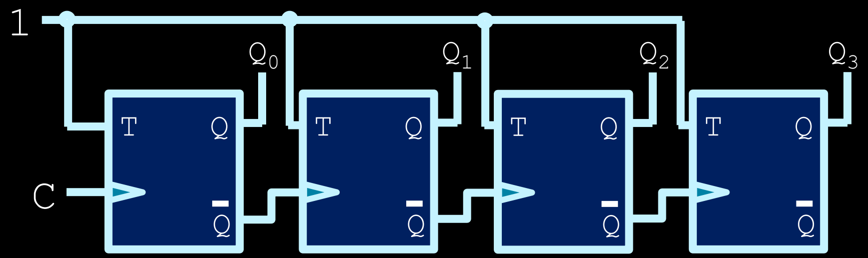
Counters

- Timing diagram

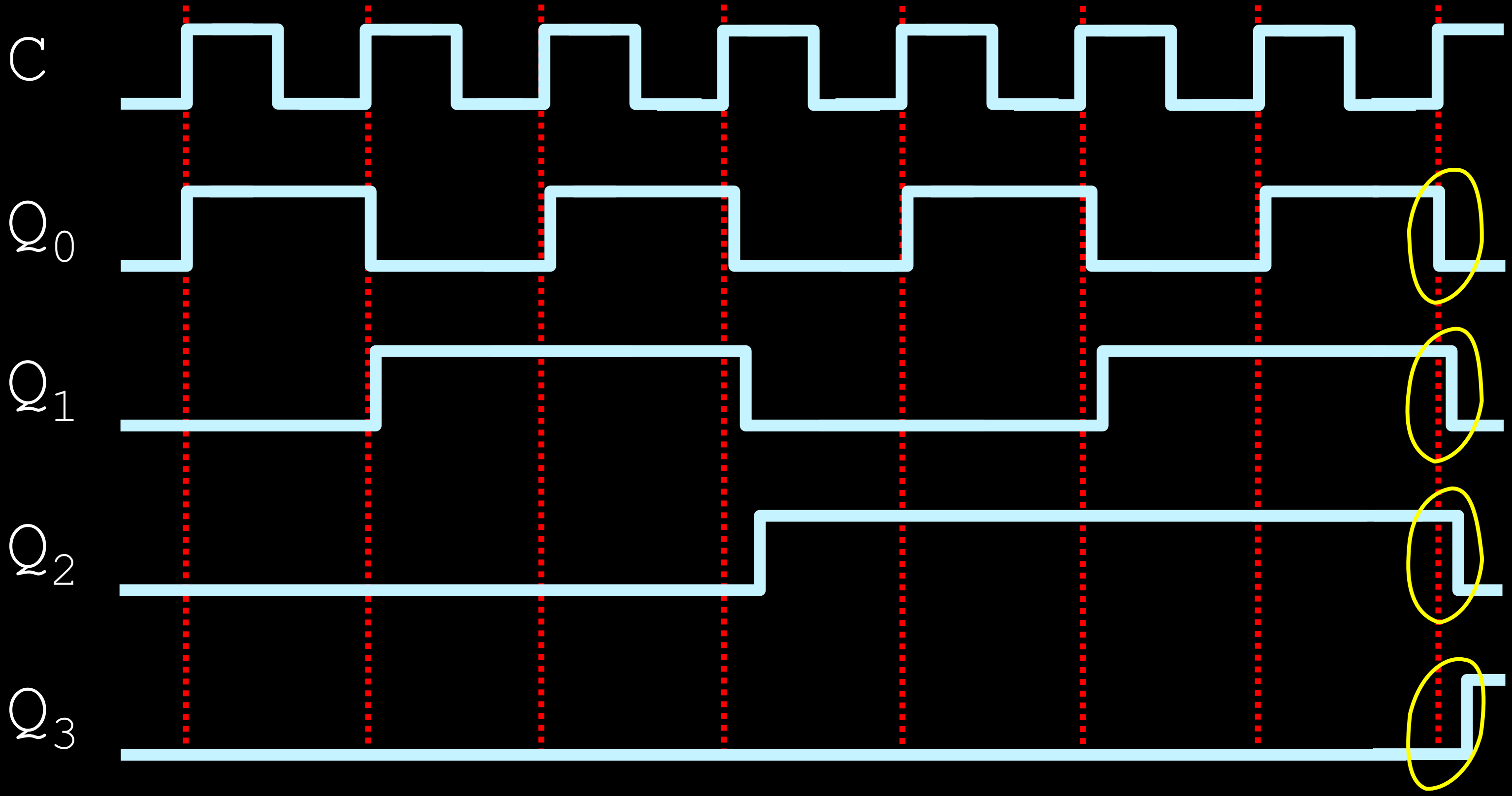


Q =	0	1	2	3	4	5	6	7	8
Q_0	0	1	0	1	0	1	0	1	0
Q_1	0	0	1	1	0	0	1	1	0
Q_2	0	0	0	0	1	1	1	1	0
Q_3	0	0	0	0	0	0	0	0	1

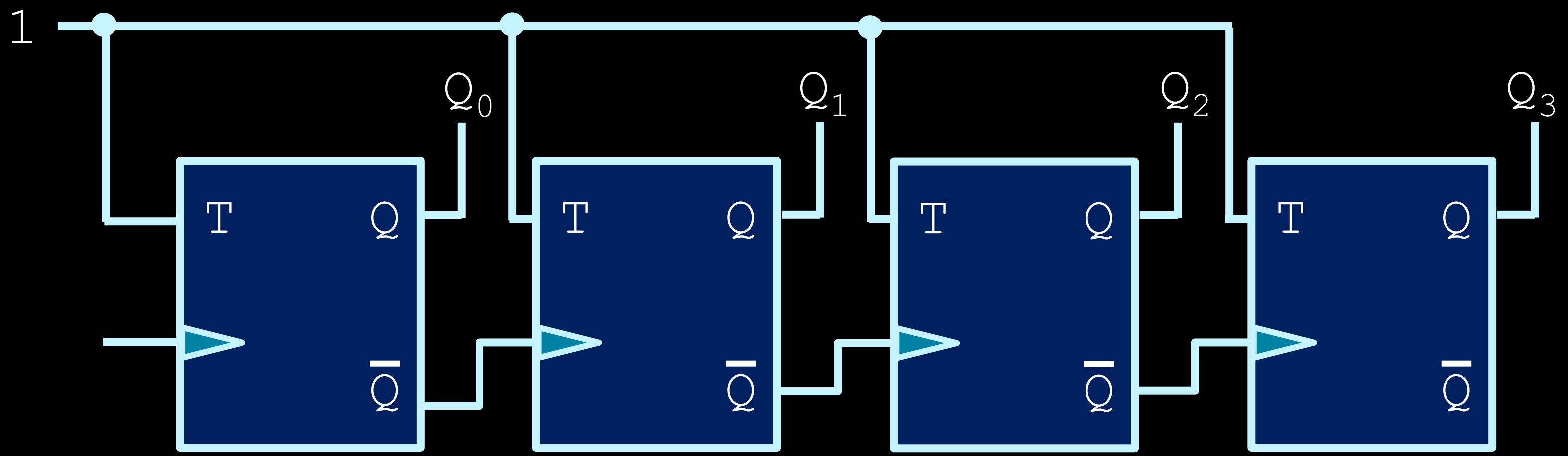
Counters



- Timing diagram
  - Propagation delay increases for later Qs

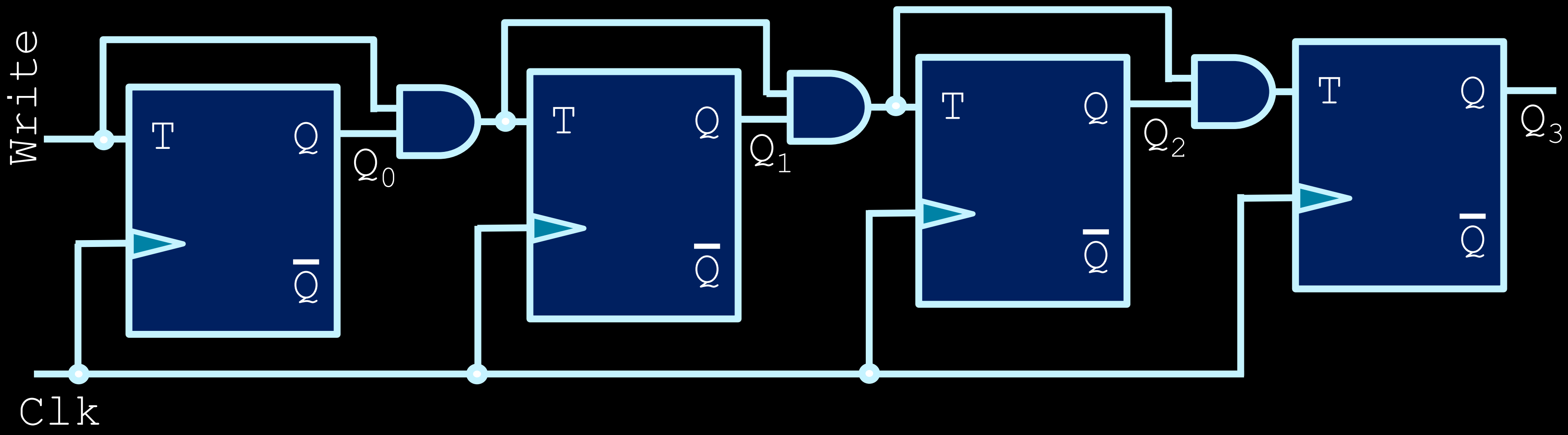


Counters



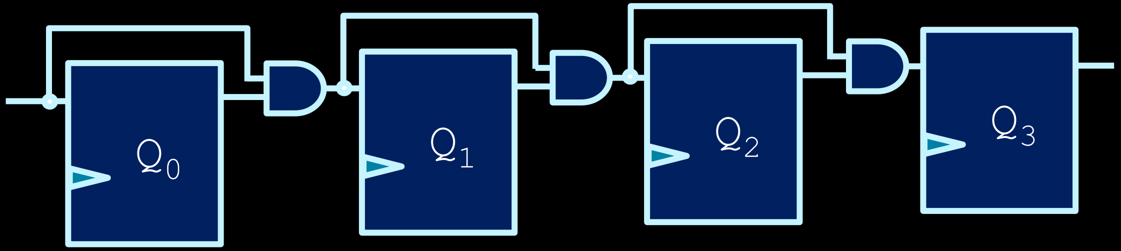
- The ripple counter, is an example of an asynchronous circuit:
  outputs do not all change with the same clock signal.
- Timing isn't quite synchronized with the rising clock pulse → hard to know when output is ready.
  - Cheap to implement, but unreliable for timing.

Synchronous Counter



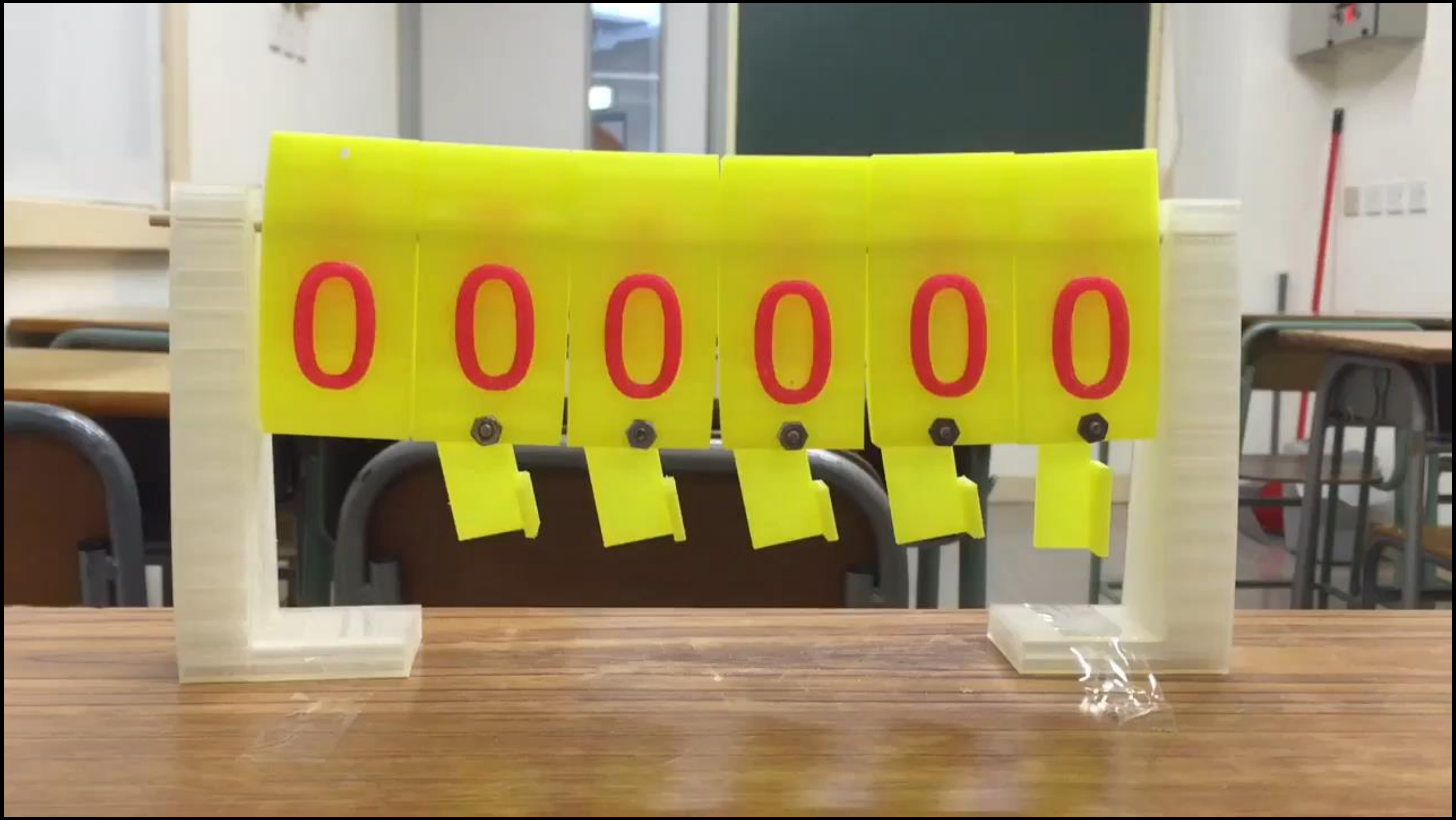
- This is a synchronous counter, with a slight delay.
- Each AND gate combine outputs of all previous flip-flops
- Each flip-flop only changes when all previous flip flops are set

How it works



- The first FF toggles every clock cycle.
- Every other FF "looks to the left"
  - That's what the AND gates do!
- If all FFs to the left are 1 on clock edge, toggle.
  - Like a carry bit generated from adder!
- Otherwise, do not toggle.
- Result:
  - 1 1 1 0 → 0 0 0 1 → 1 0 0 1 → 0 1 0 1
  - Read from right to left: 0111 → 1000 → 1001

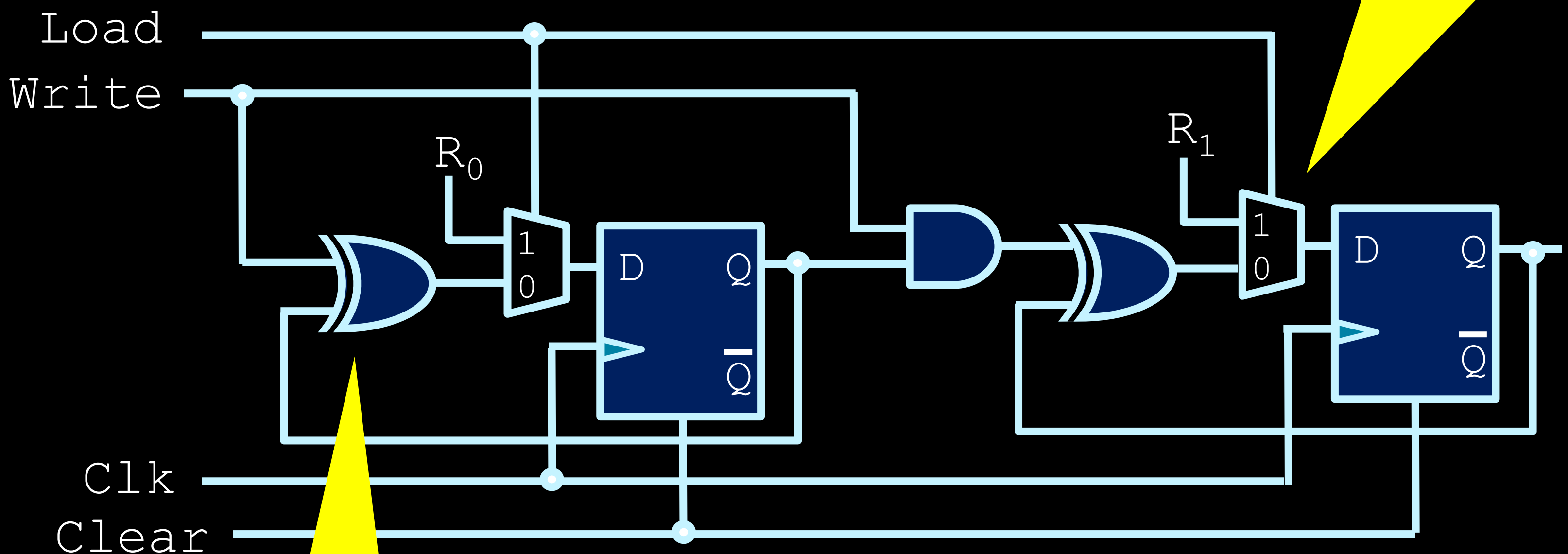
How it works? Like this



https://www.youtube.com/watch?v=_okGZhUE5Do

Counters with Load

When load is high, read from $\mathbf{R_i}$ instead of toggling



only advance when **`write`** is high (like **`T`** in T flip-flop)

Counter with parallel load, write, and clear (reset) inputs.

- Useful for countdowns and more.

What About Custom Designs?

- Registers and counters are simple.
- What about more sophisticate designs?