IE1204 Digital Design

L8: Memory Elements: Latches and Flip-Flops. Counter

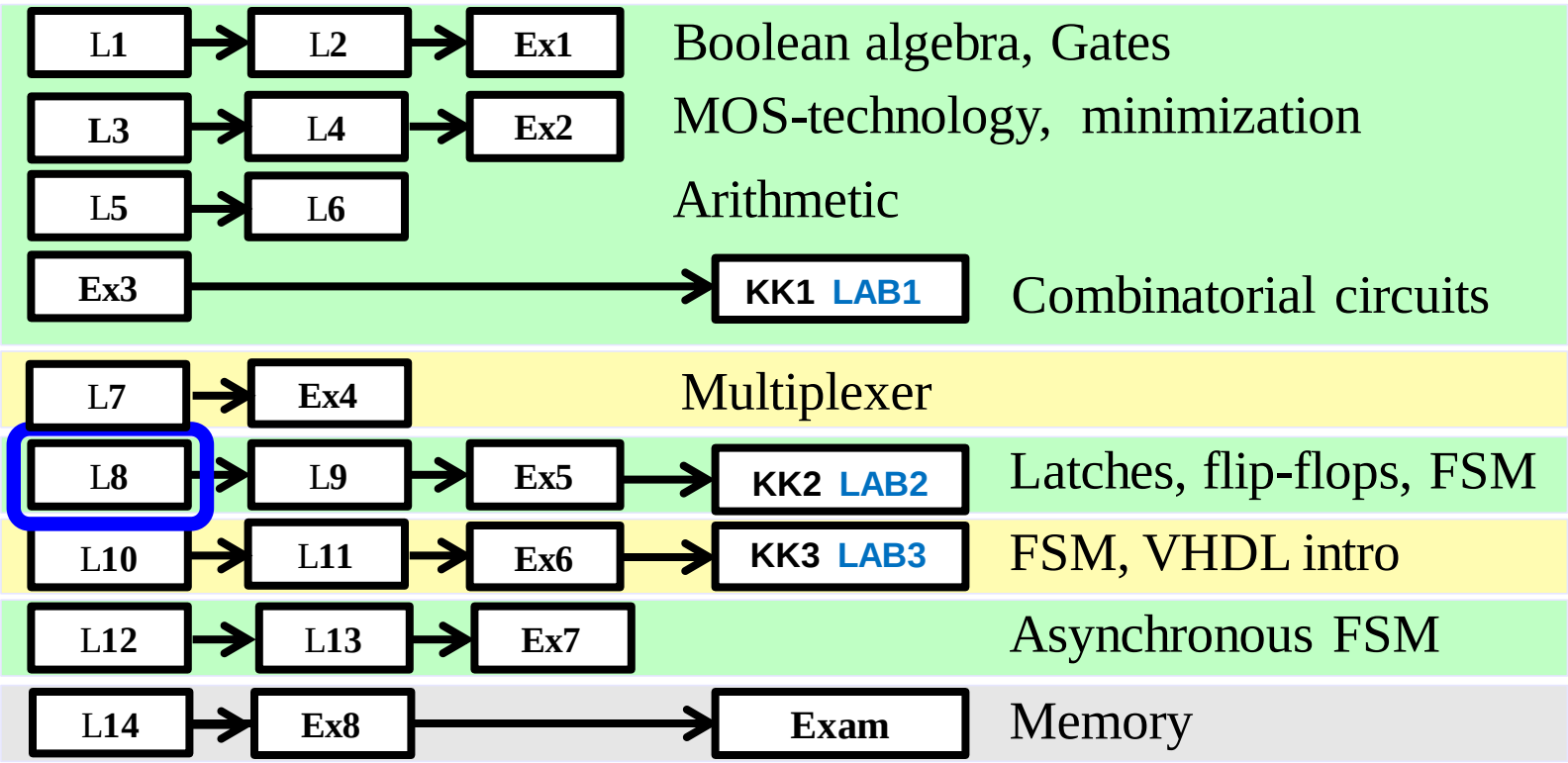
KTH Informations- och kommunikationsteknik

Masoumeh (Azin) Ebrahimi

KTH/ICT

mebr@kth.se

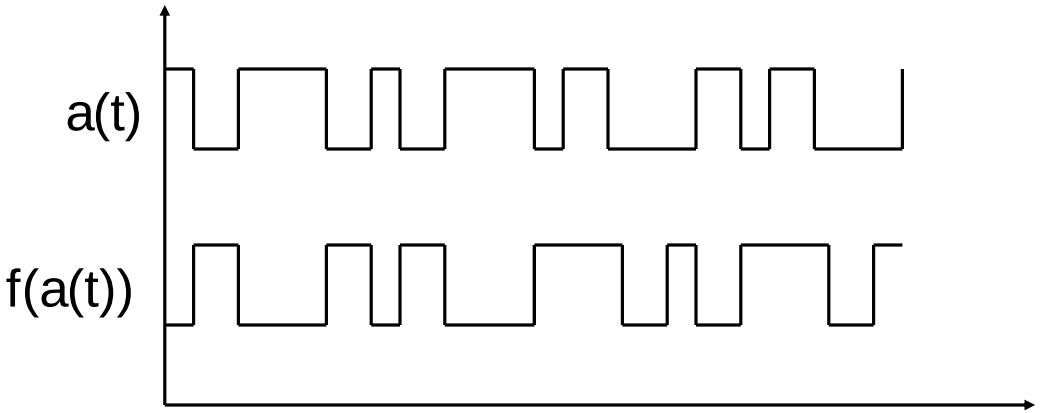
IE1204 Digital Design



This lecture

- BV pp. 383-418, 469-471

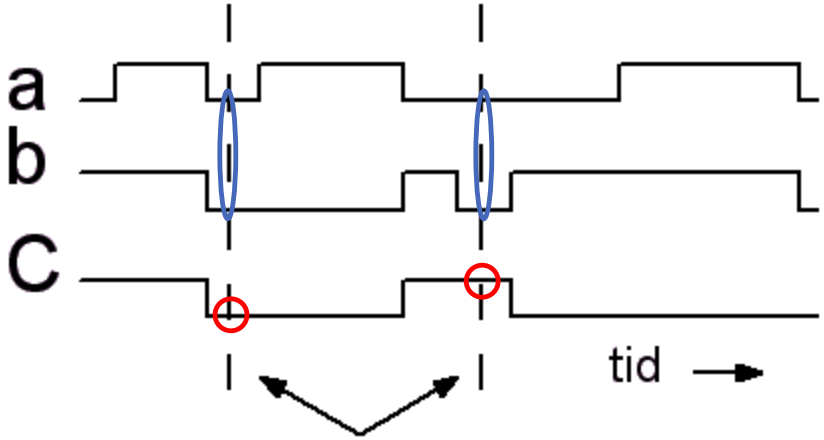
Sequential circuits



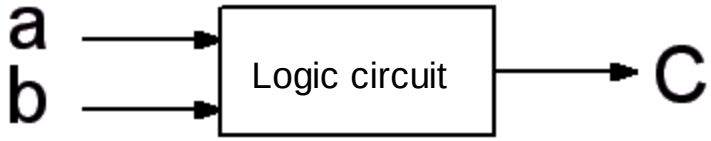
A sequential system has a built-in memory - the output depends therefore BOTH on the current and previous value(s) of the input signal

Lecture 8 - Lecture 13

Sequential circuits



Same input a, b can produce different output C.



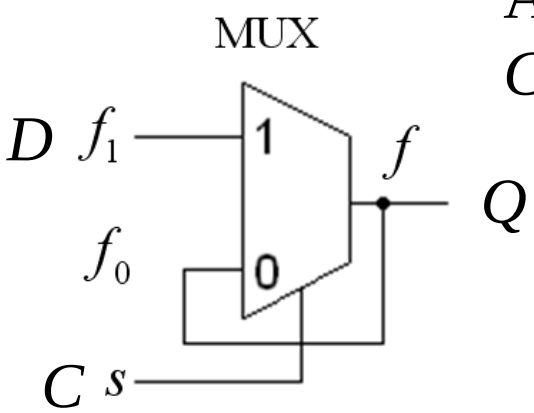
If the same inputs produce different output values, we have a sequential logic circuit. It must then have an internal **memory** that allows the output to be affected by both the current and previous inputs!

How do we get the hardware to remember something?

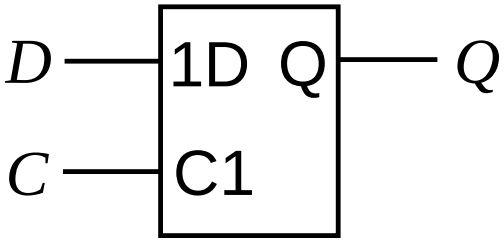
- To remember something, we have to somehow retain the information
- One way is to store information in the form of a charge on a capacitance (DRAM)
- Another way is to let the information "run around in a circle and bit its own tail"

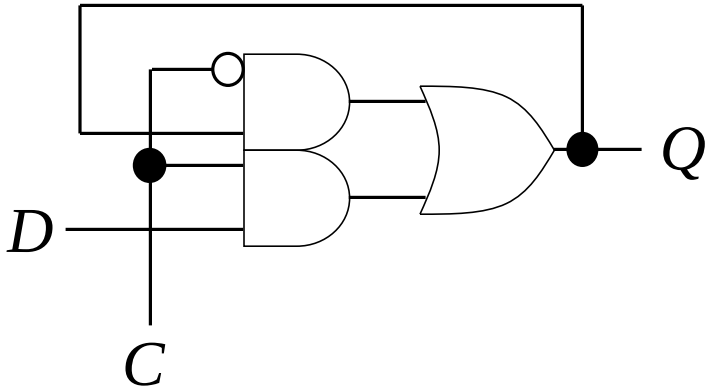
D-Latch

A D-latch is a MUX with feedback. When $C = 0$ the value is latched.



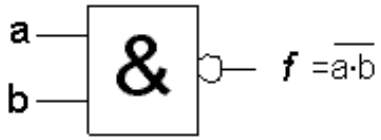
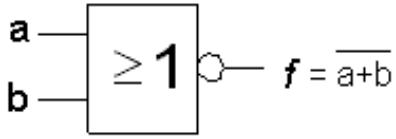
D-Latch





C follow / $\overline{latch}$	D	Q
0	–	M latch
1	D	D follow

NOR and NAND ”locking input signal”

Name	Logic function - Gate
NAND	
NOR	

a	b	f
0	0	1
0	1	1
1	0	1
1	1	0

a	b	f
0	0	1
0	1	0
1	0	0
1	1	0

Rule …

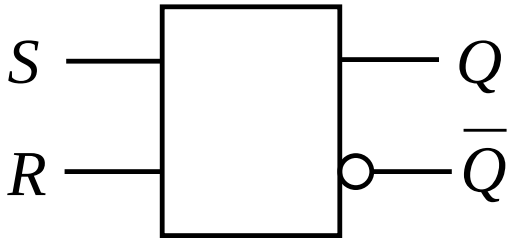
NAND. If any input is "0", the output is "1" regardless of the value of the other input!

NOR. If any input is "1", the output "0" regardless of the value of the other input!

Set-Reset (SR) Latch

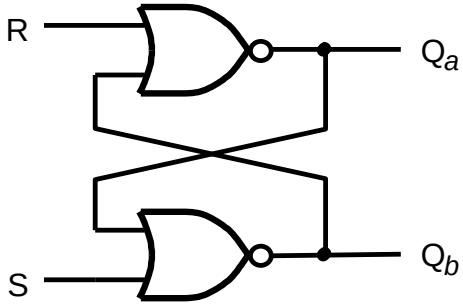
- It has two inputs and two outputs
  - Inputs: Set (S) and Reset (R)
  - Outputs: Q and $\overline{Q}$ which should be always complement of each other
- It can be constructed from two cross-coupled NOR gates or two cross-coupled NAND gates.
- It has three modes of operation
  - No change: Output does not change
  - Reset: Output (Q) reset to 0
  - Set: Output (Q) set to 1

SR-Latch



SR-latch (NOR)

- SR-latch can be implemented with NOR gates
- SET and RESET inputs are active high
- SET and RESET should not be active at the same time!



(A) Circuit

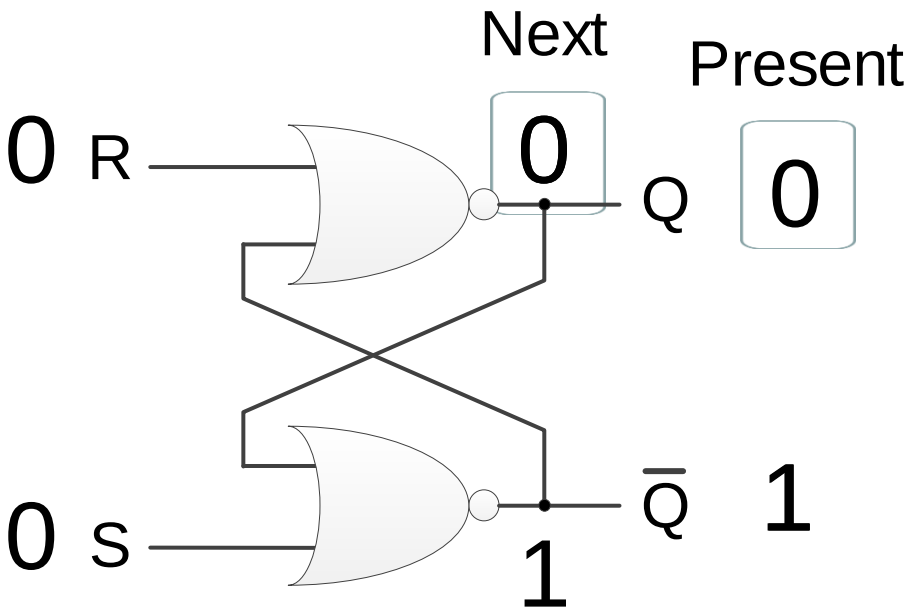
S	R	Q_a	Q_b	
0	0	0/1	1/0	(No change)
0	1	0	1	
1	0	1	0	
1	1	0	0	

Prohibited input combination (causes oscillation)

(B) Characteristic Table

SR-Latch: no change (S=0,R=0)

For a NOR gate "1" is a "locking" input



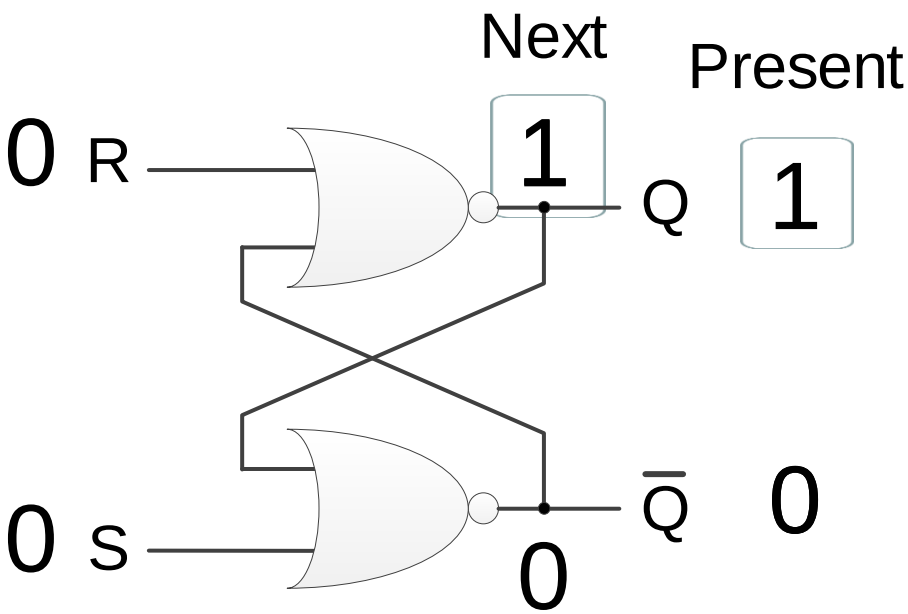
Inputs		Present		Next	
S	R	Q	$\overline{Q}$	Q	$\overline{Q}$
0	0	0	1	0	1

Next State = Present State

Holding a logic value of 0

SR-Latch: no change (S=0,R=0)

For a NOR gate "1" is a "locking" input



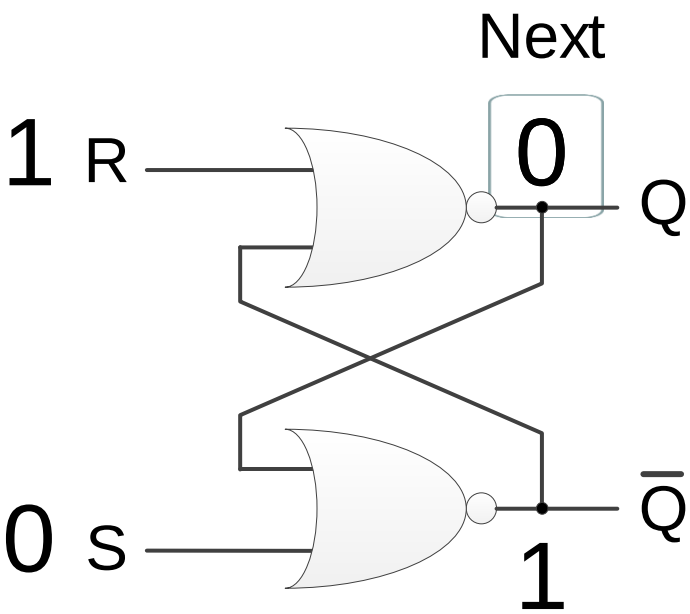
Next State = Present State

	Inputs		Present		Next	
	S	R	Q	$\overline{Q}$	Q	$\overline{Q}$
Hold	0	0	0	1	0	1
	0	0	1	0	1	0

Holding a logic value of 1

SR-Latch: Reset (S=0,R=1)

For a NOR gate "1" is a "locking" input

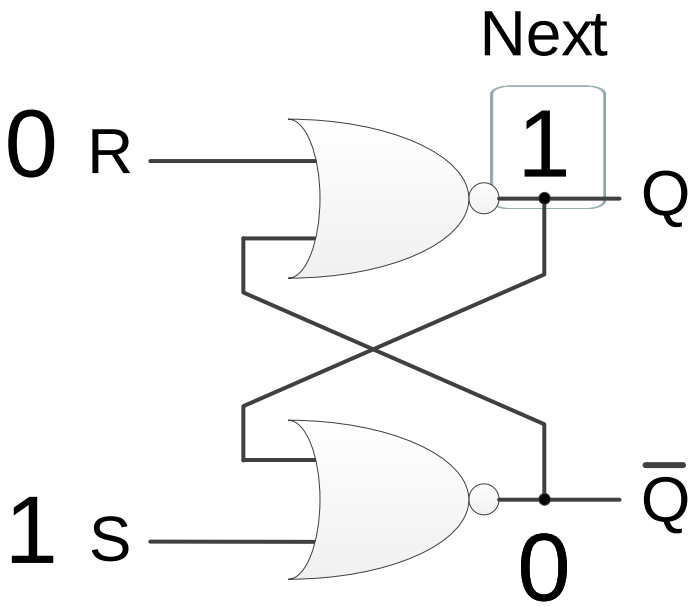


	Inputs		Present		Next	
	S	R	Q	$\overline{Q}$	Q	$\overline{Q}$
Hold	0	0	0	1	0	1
	0	0	1	0	1	0
Reset	0	1	0	1	0	1
	0	1	1	0	0	1

Reset the SR-Latch output to 0 regardless of the present value

SR-Latch: Set (S=1,R=0)

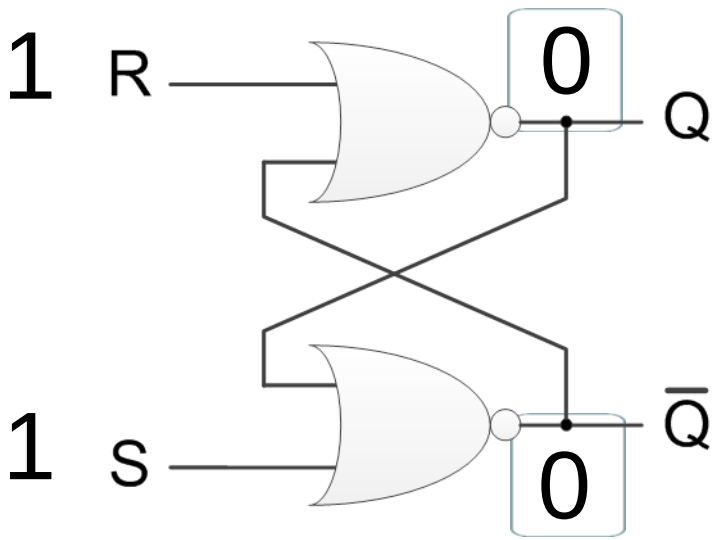
For a NOR gate "1" is a "locking" input



	Inputs		Present		Next	
	S	R	Q	$\overline{Q}$	Q	$\overline{Q}$
Hold	0	0	0	1	0	1
	0	0	1	0	1	0
Reset	0	1	0	1	0	1
	0	1	1	0	0	1
Set	1	0	0	1	1	0
	1	0	1	0	1	0

Set the SR-Latch output to 1 regardless of the present value

SR-Latch: Improper Operation



Q and $\overline{Q}$ should be complementory of each other

	Inputs		Present		Next	
	S	R	Q	$\overline{Q}$	Q	$\overline{Q}$
Hold	0	0	0	1	0	1
	0	0	1	0	1	0
Reset	0	1	0	1	0	1
	0	1	1	0	0	1
Set	1	0	0	1	1	0
	1	0	1	0	1	0
Forbiden	1	1	-	-	-	-

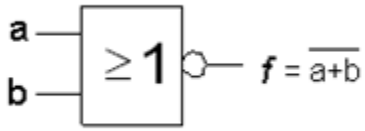
Characteristic Table

	Inputs		Present		Next	
	S	R	Q	$\overline{Q}$	Q	$\overline{Q}$
Hold	0	0	0	1	0	1
	0	0	1	0	1	0
Reset	0	1	0	1	0	1
	0	1	1	0	0	1
Set	1	0	0	1	1	0
	1	0	1	0	1	0
Forbiden	1	1	-	-	-	-

S	R	Q_a	Q_b	
0	0	0/1	1/0	(no change)
0	1	0	1	(Reset)
1	0	1	0	(Set)
1	1	0	0	(Forbidden)

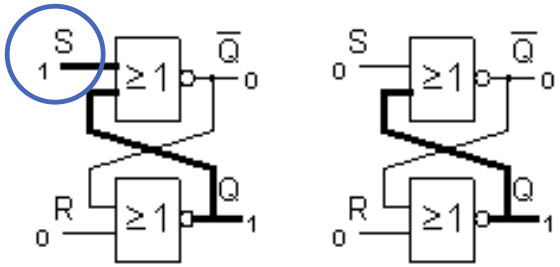
Prohibited input combination (causes oscillation)

SR-latch (NOR)

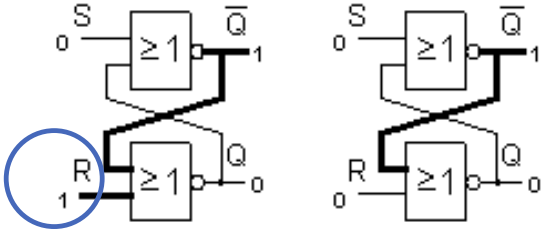


a	b	f
0	0	1
0	1	0
1	0	0
1	1	0

Q=1



Q=0

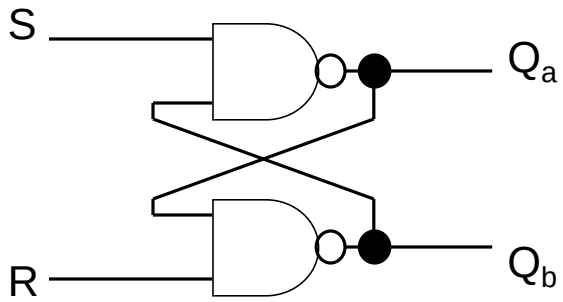


For a NOR gate "1" is a "locking" input - if any input is "1" it does not matter what input value any other input has - the output will then always "0".

It is therefore enough with a short pulse "1" on S for the circuit to keep Q = 1. A short pulse "1" on R then gives Q = 0.

SR-latch (NAND)

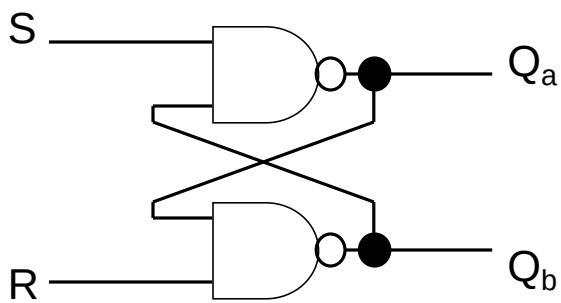
- SR-latch can also be implemented with NAND gates
- SET and RESET inputs are active low



NAND. If any input is "0", the output is "1" regardless of the value of the other input!

SR-latch (NAND)

- SR-latch can also be implemented with NAND gates
- SET and RESET inputs are active low

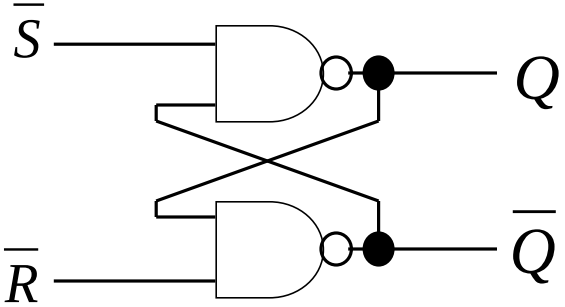


S	R	Q_a	Q_b	
0	0	1	1	Prohibited input combination
0	1	1	0	
1	0	0	1	
1	1	0/1	1/0	(No change)

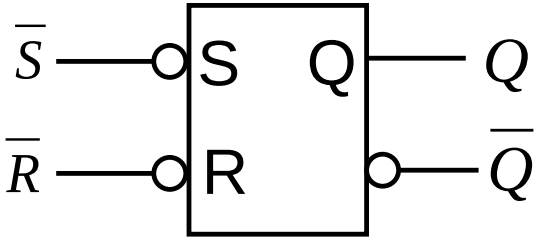
A Latch with NAND gates have active low SET and RESET inputs. They may not be "0" both at the same time.

$\overline{S}\overline{R}$-latch (NAND)

In fact the input signals of the SR-latch with NAND-gates are active low and thus we show them as $\overline{S}$ and $\overline{R}$ and call this latch as $\overline{S}\overline{R}$-latch. They inputs may not be "0" both at the same time.



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

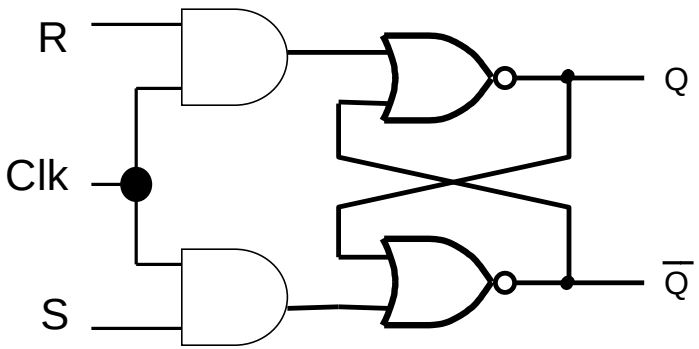


$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	M	M

Active low in this example means when the set signal S=0 then the Q is set to 1 and when the reset signal R=0, the the Q is reset to 0.

Gated SR-Latch (NOR)

- To ensure that the state can only be changed at certain points in time, a special *Clock* signal is used

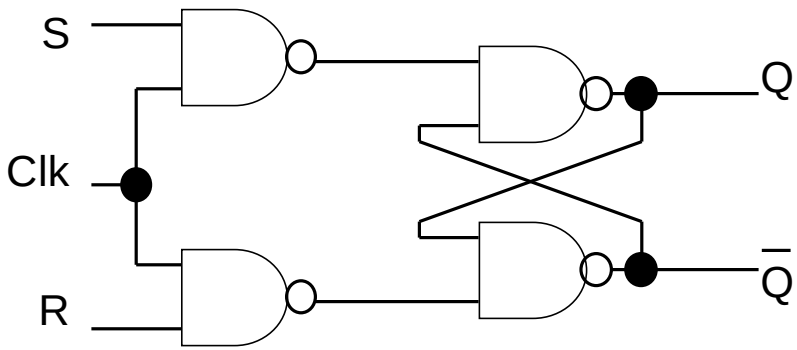


Clk	S	R	Q(t+1)
0	-	-	Q(t) (no change)
1	0	0	Q(t) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

Forbidden combination

Gated SR-Latch (NAND)

With two additional gates and a clock signal *Clk* you can control when the latch will get affected by the inputs *S* and *R*. When *Clk* = 0 there is no influence, then even $\boxed{S = R = 1}$ could be tolerated.



Clk	S	R	Q(t+1)
0	-	-	Q(t) (no change)
1	0	0	Q(t) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

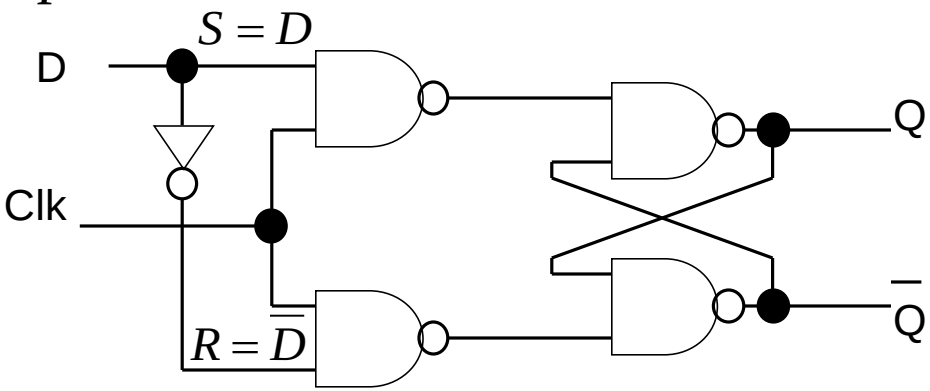
Forbidden combination

Gated D-Latch

A better solution to the problem of the "forbidden" state is the D-latch.

The latch output follows the *D* input when *Clk* = 1 and lock the value when *Clk* = 0.

This latch circuit has the same function as the MUX circuit with feedback. The difference is that this circuit has *faster feedback*. Moreover, we also have access to an *inverted output*.

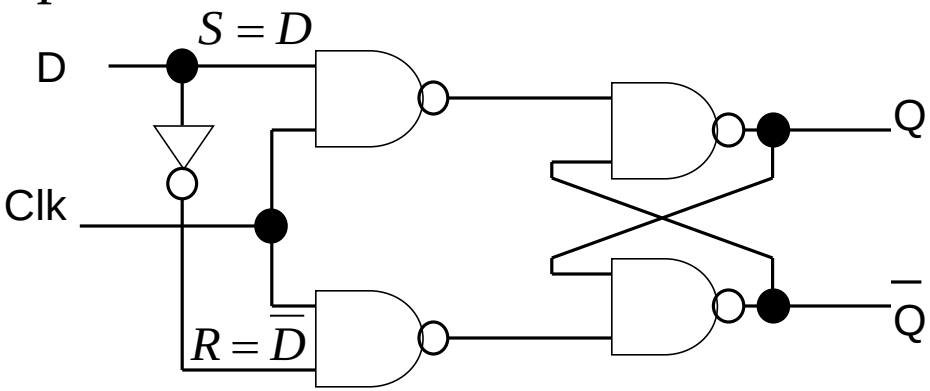


Gated D-Latch

A better solution to the problem of the "forbidden" state is the D-latch.

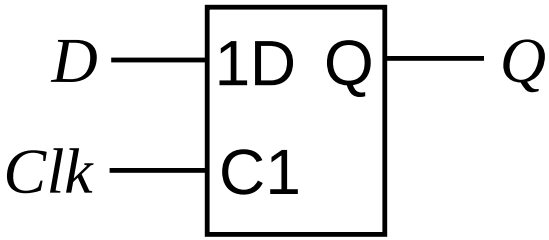
The latch output follows the *D* input when *Clk* = 1 and lock the value when *Clk* = 0.

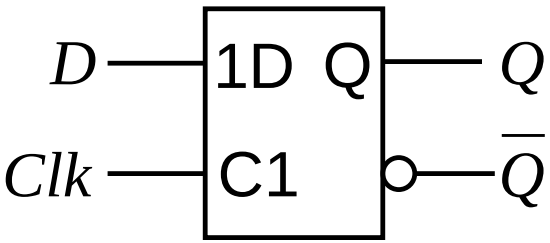
This latch circuit has the same function as the MUX circuit with feedback. The difference is that this circuit has *faster feedback*. Moreover, we also have access to an *inverted output*.



Clk	D	Q(t+1)	
0	x	Q(t)	(lock)
1	0	0	(follow)
1	1	1	(follow)

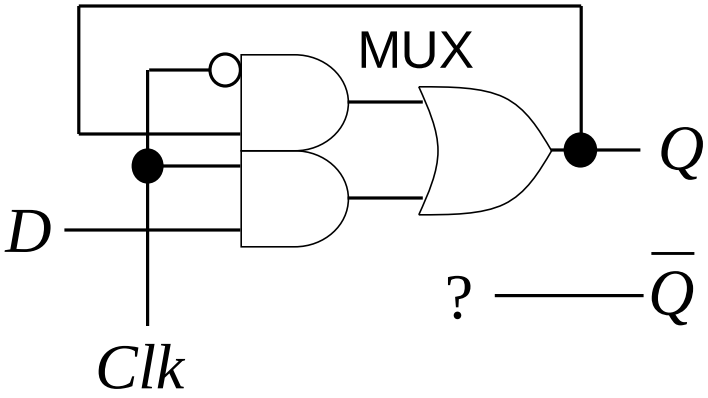
Two different D-latches

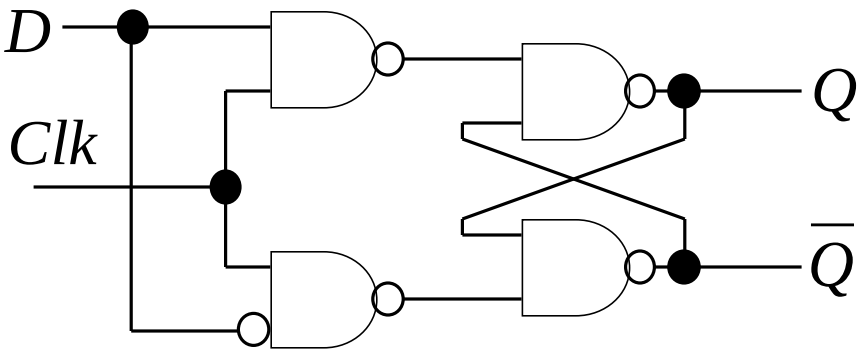




Long feedback (~4T)

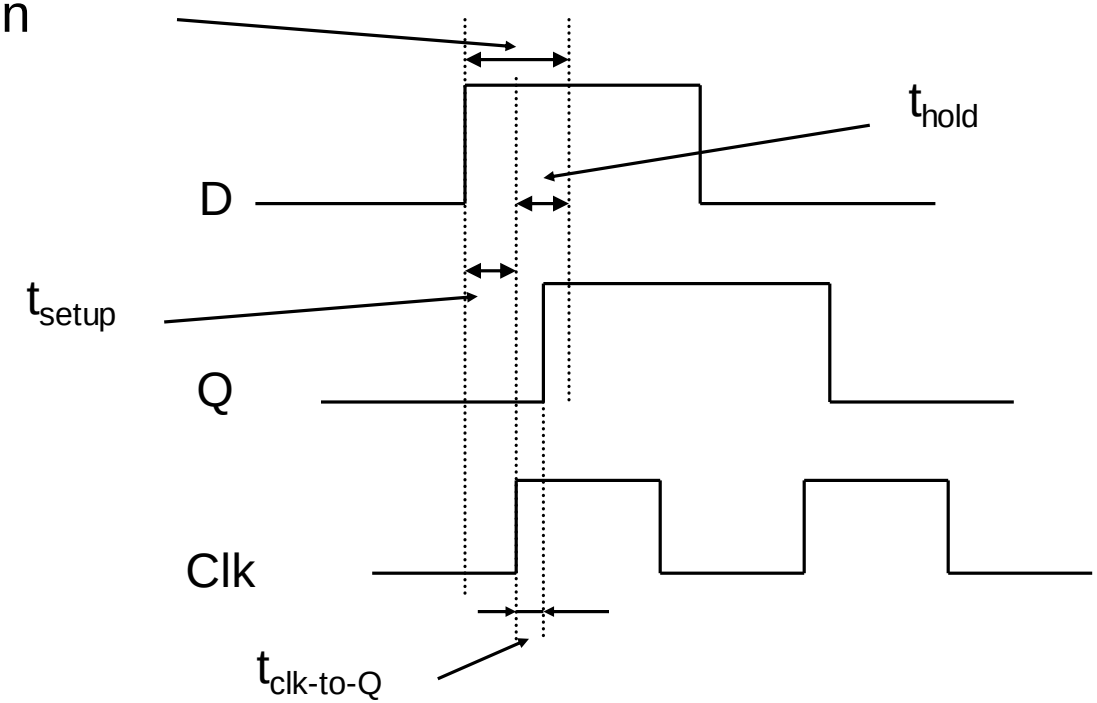
Short feedback (~1T)





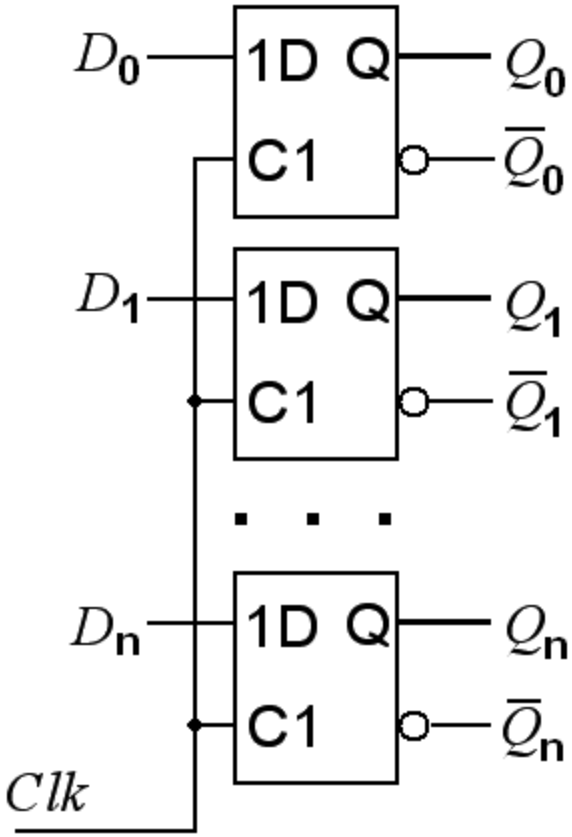
Setup & Hold Time

D must be stable within this area to ensure correct operation



Setup time: the minimum time that the D signal must be stable prior to the (positive/negative) edge of the clk signal.
Hold time: the minimum time that the D signal must remain stable after the (positive/negative) edge of the clk signal.

Register – inverted signals



A common way to design digital circuits is that the signal is taken via registers (= a set of latches or flip-flops) to the combinatorial network inputs. D-latches "automatically" provides inverted signals at their outputs.

That's why we usually assume that inverted signals are available.

How do we create a sequence?

- How we can construct a sequential circuit that toggle its output at every clock pulse, Clk ?

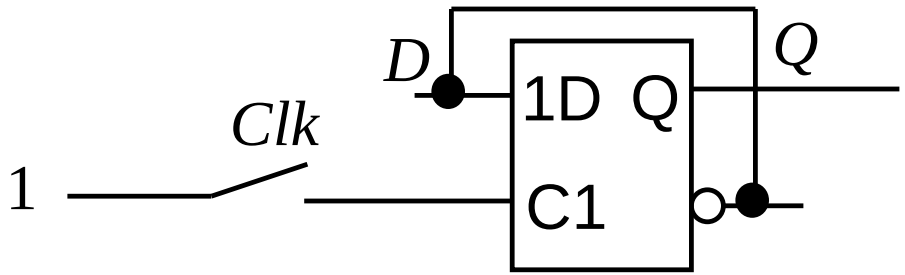
Ex: 0,1,0,1, ...

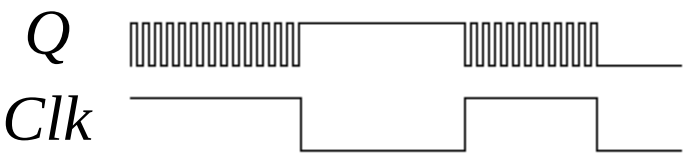
the next value = NOT (present value)

- We need to process (invert) the current value and then remember it until the next value is calculated

Not possible with a simple latch…

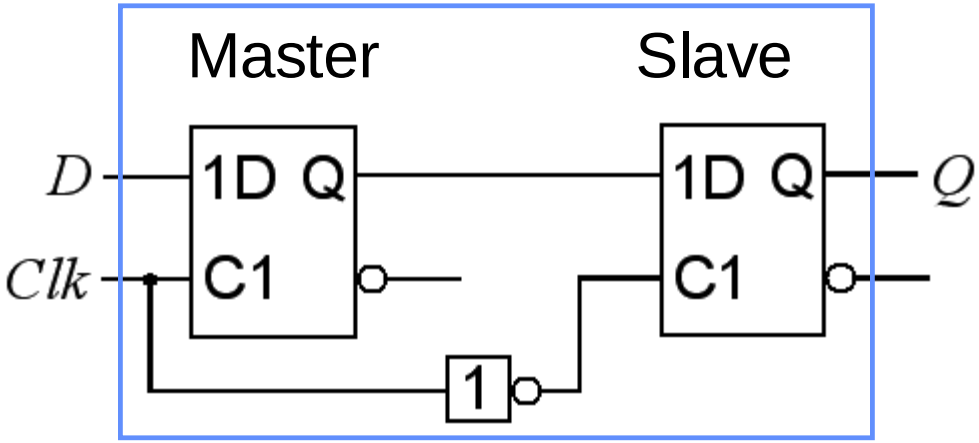
$$Clk = follow / \overline{latch} \qquad D = \overline{Q} \quad Q = D$$





- When *Clk* = 1 the output follows the input – therefore the output changes 1/0 as quickly as possible!
  The circuit becomes an oscillator!
- When *Clk* = 0 the output retains its value 1/0 after whatever it happened to be.

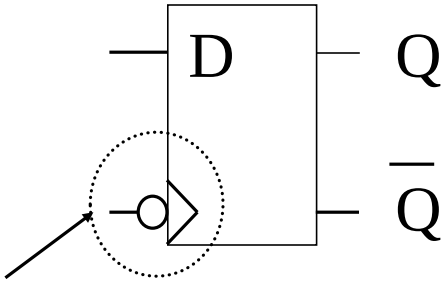
Master-slave D-flip flop



Clk	D	Q(t+1)
↓	0	0
↓	1	1

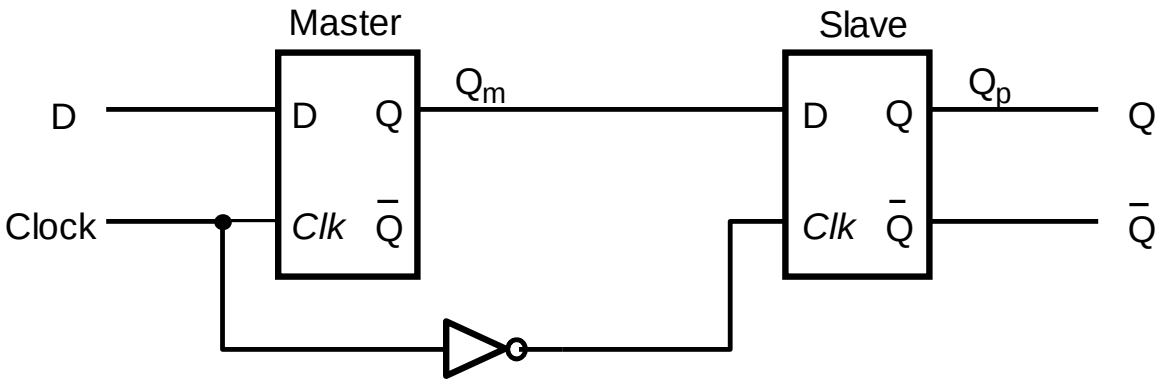
Negative edge triggered D-flip flop

The solution is the **clocked flip-flop** consisting of several latches. One latch receives new data (Master) while another latch retains the old data (Slave).



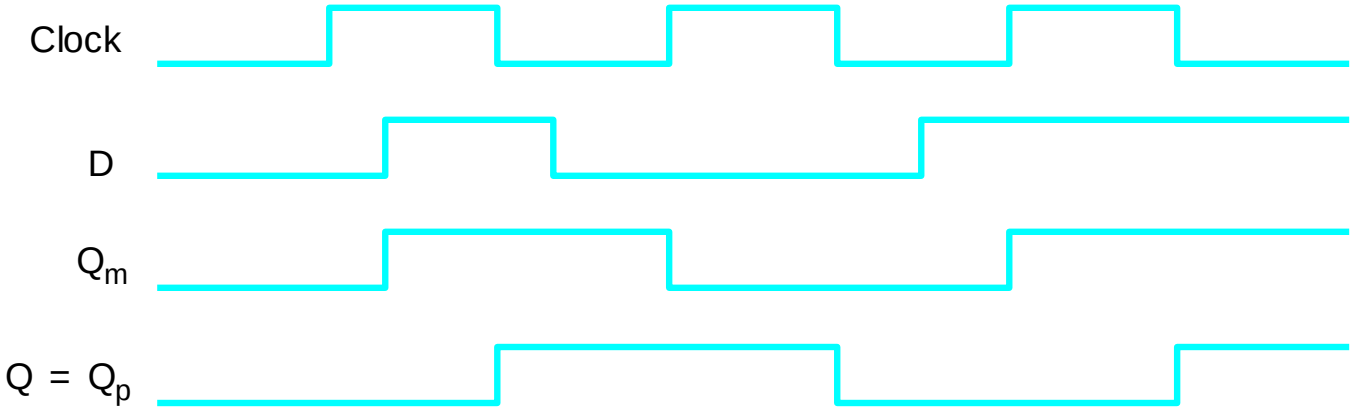
Inversion ring at CLK indicates a negative edge. Triangle indicates edge sensitivity

Timing Chart Master-slave

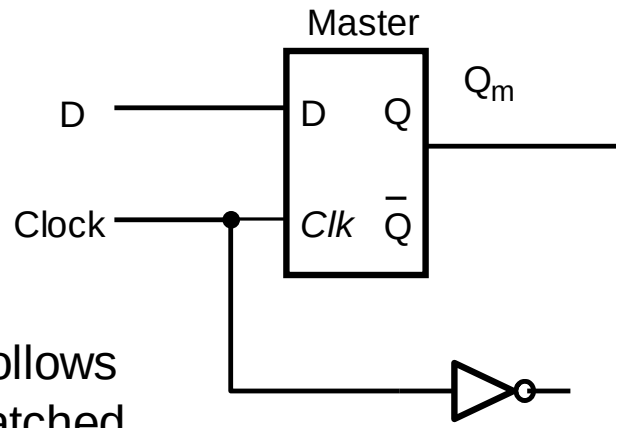


Clock=1, Master follows
Clock=0, Master latched

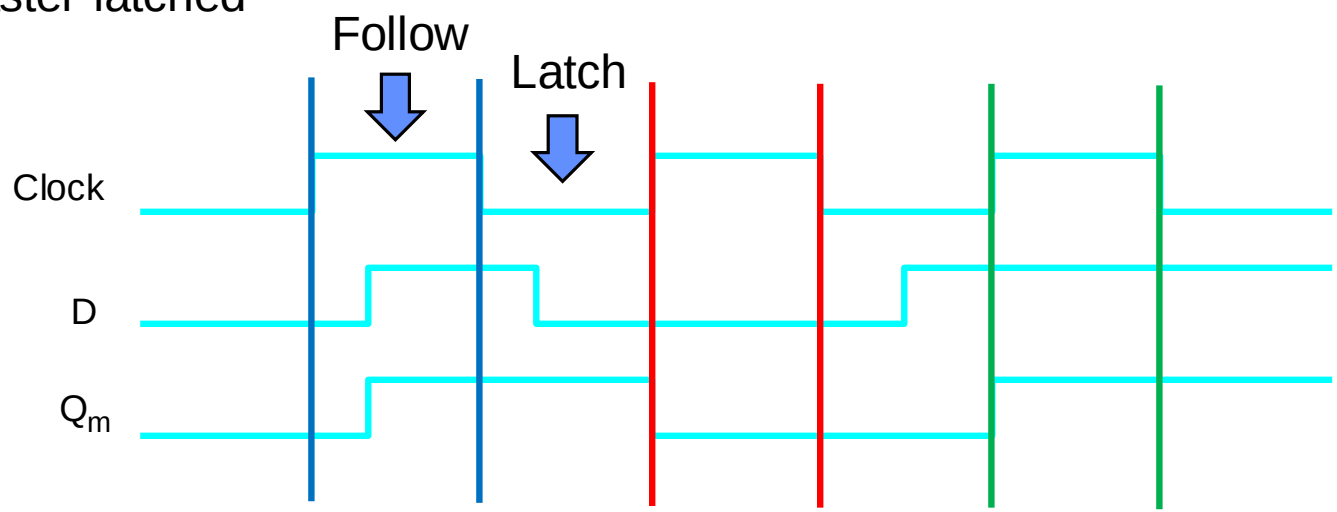
Clock=1, Slave latched
Clock=0, Slave follows



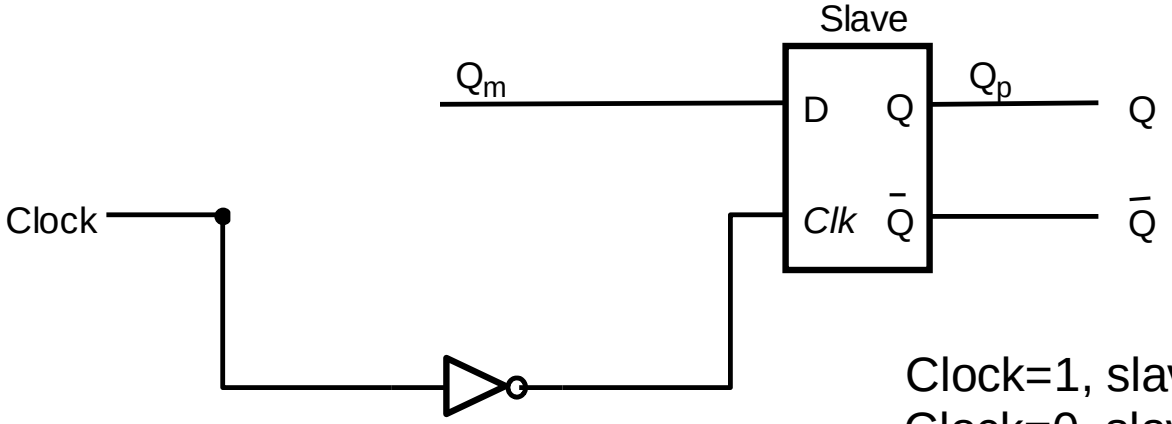
Timing Chart Master-slave



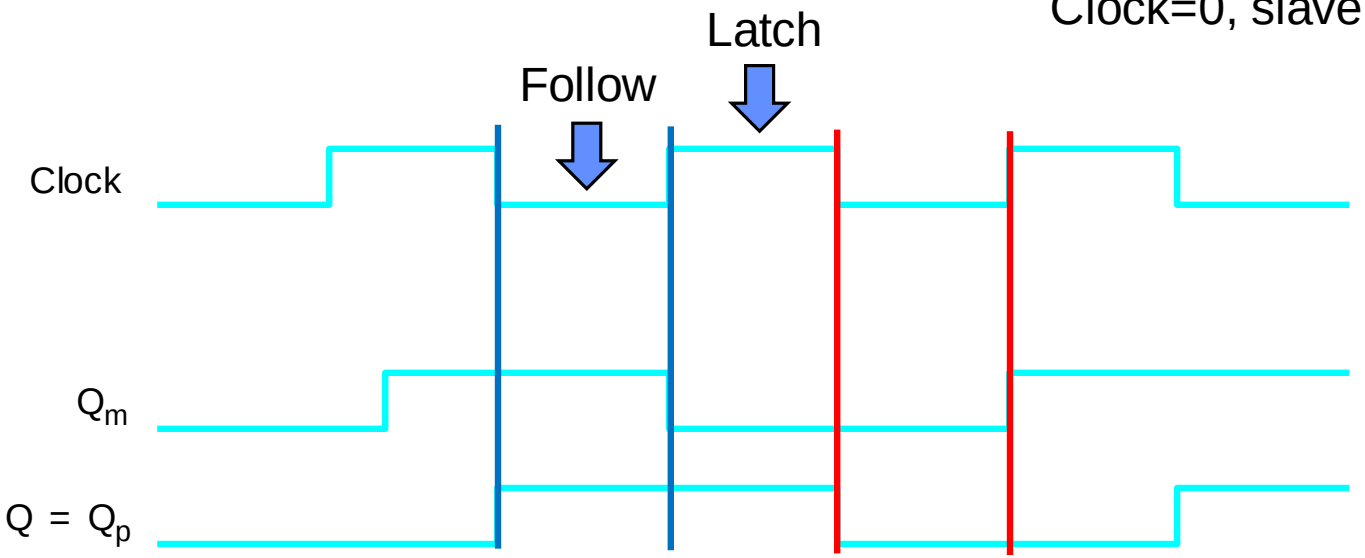
Clock=1, master follows
Clock=0, master latched



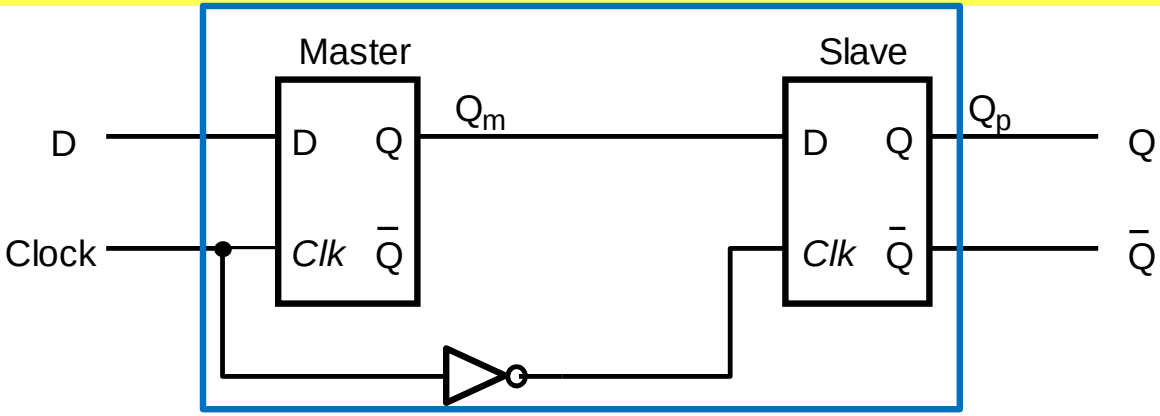
Timing Chart Master-slave



Clock=1, slave latched
Clock=0, slave follows

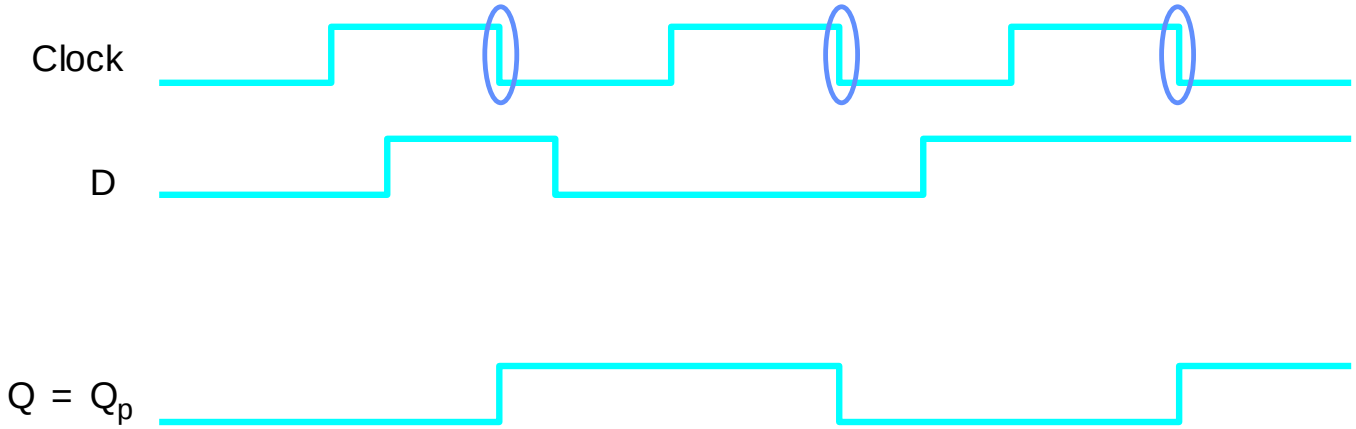


Timing Chart Master-slave



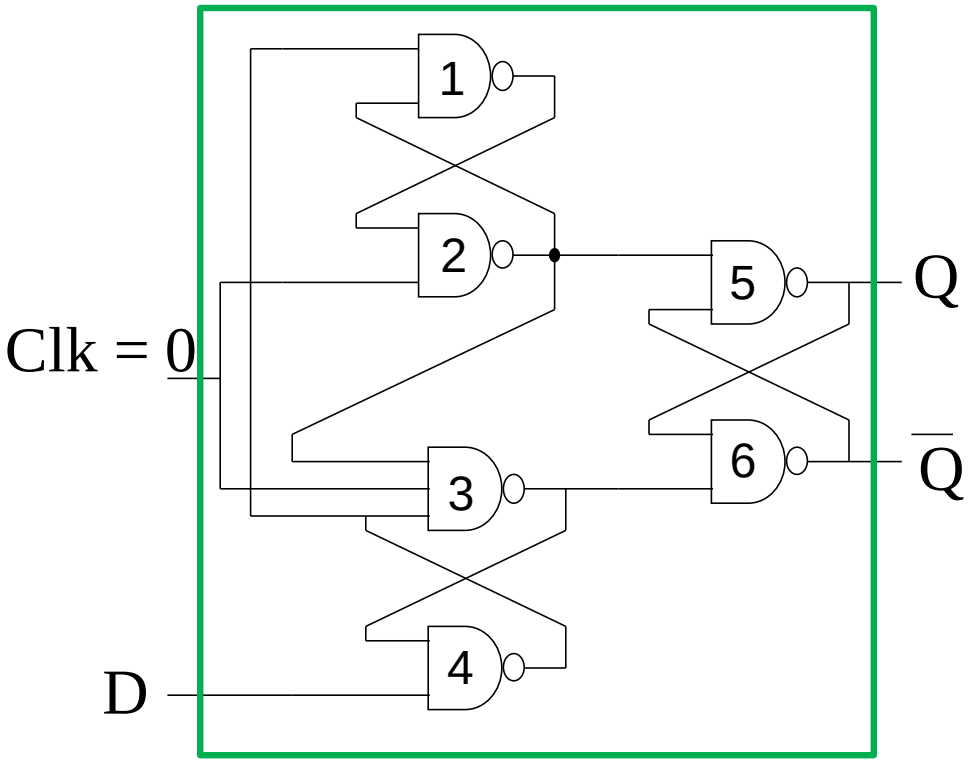
Clock=1, master follows
Clock=0, master latched

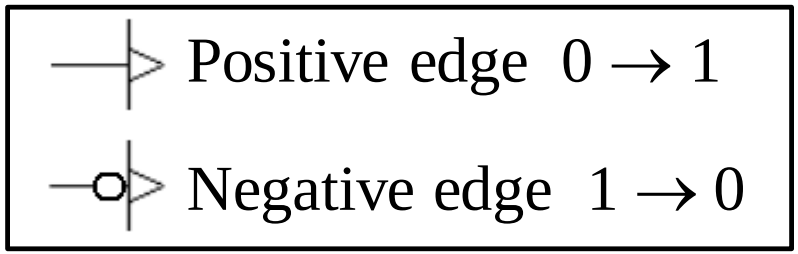
Clock=1, slave latched
Clock=0, slave follows

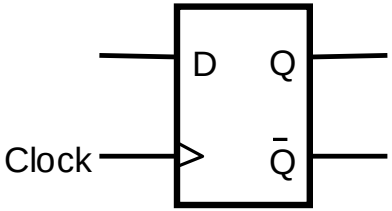


Positive edge-triggered D flip-flop

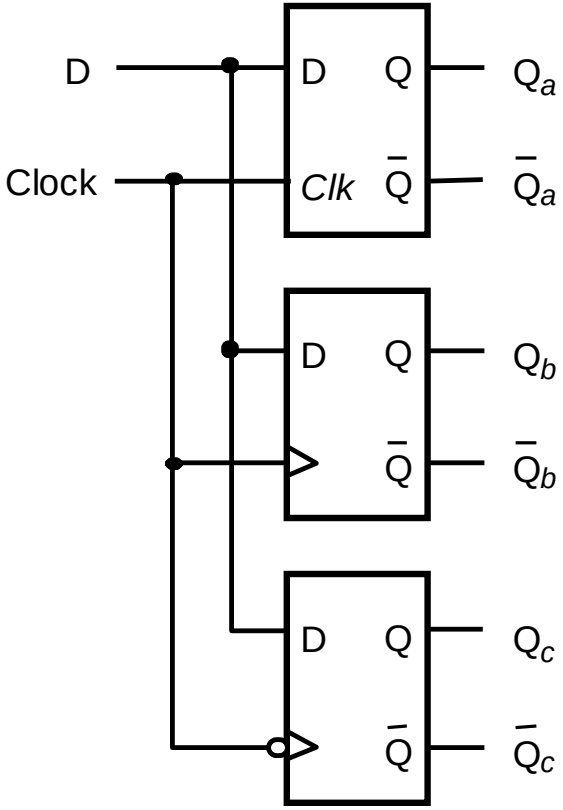
Another edge-triggered flip-flop consists of three latches. The data value is "copied" to the output just when the clock signal goes from $0 \rightarrow 1$.







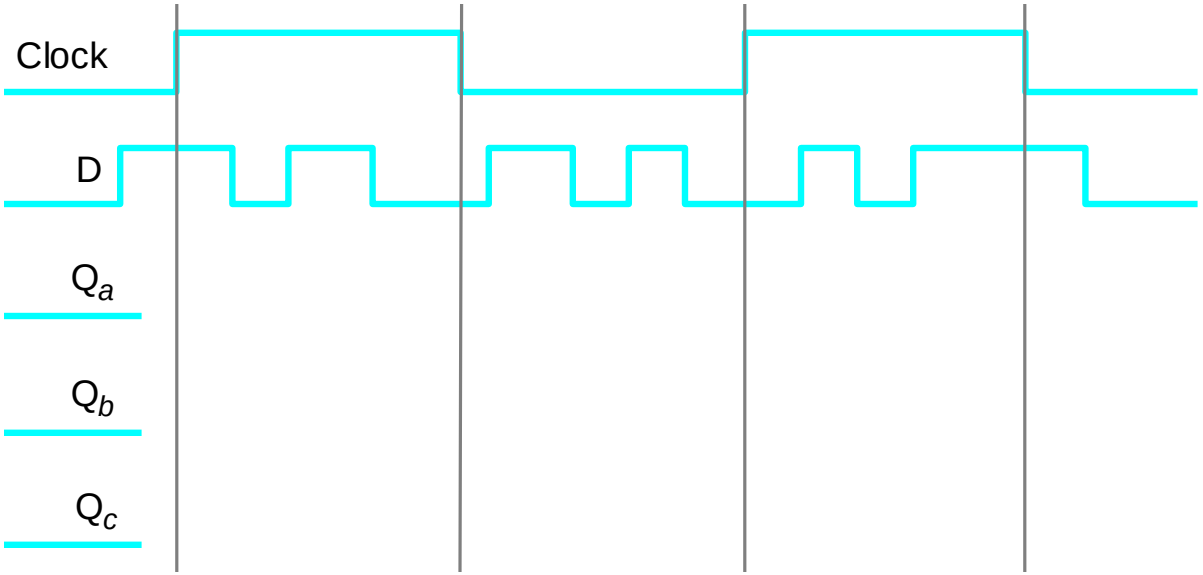
Latch vs. Flipflop



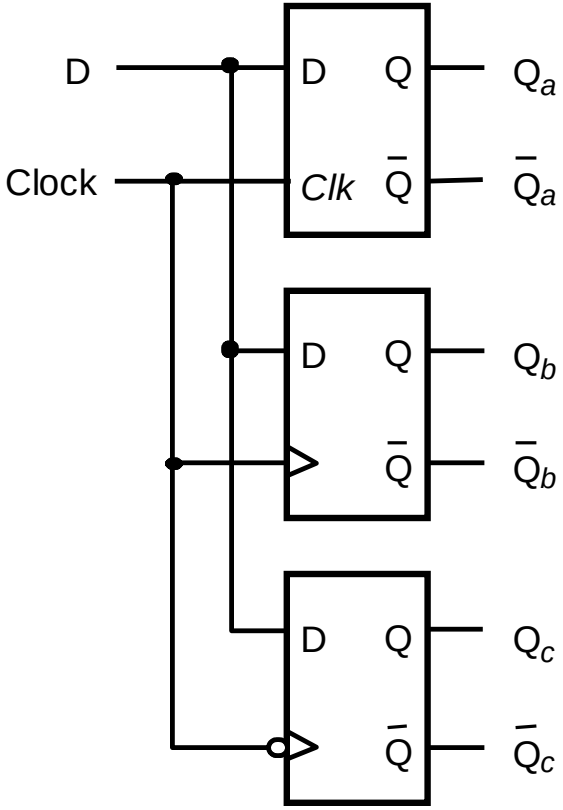
a) Latch – follow/$\overline{\text{latch}}$

b) Positive edge triggered flipflop

c) Negative edge triggered flipflop



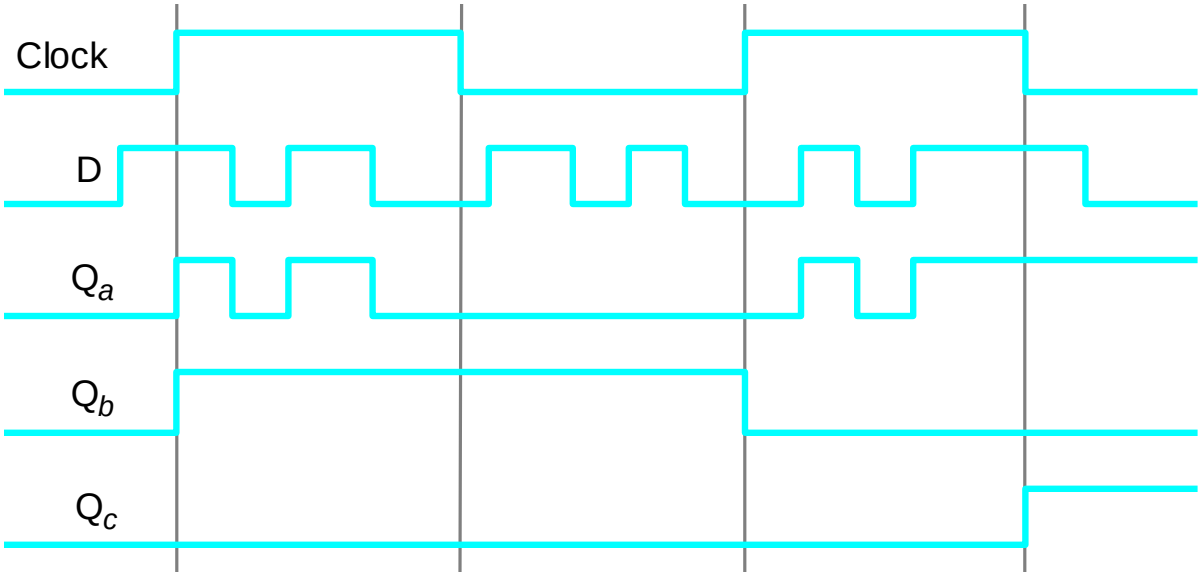
Latch vs. Flipflop



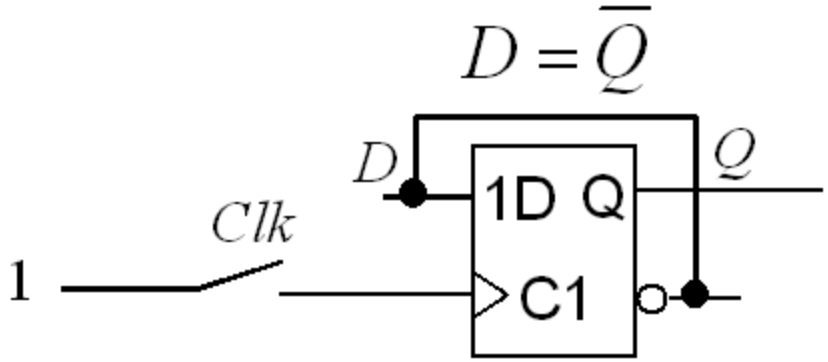
a) Latch – follow/$\overline{\text{latch}}$

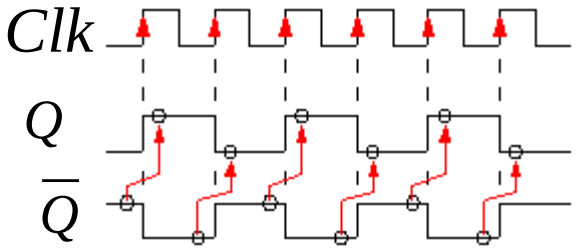
b) Positive edge triggered flipflop

c) Negative edge triggered flipflop



Every other time?



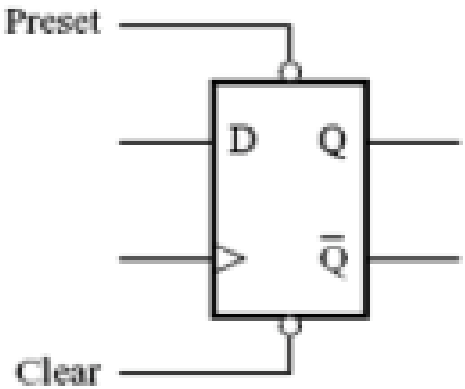


Now the "every other time“ circuit works just as planned!

In general, for sequential circuits, edge-triggered flip-flops are employed as the memory elements!

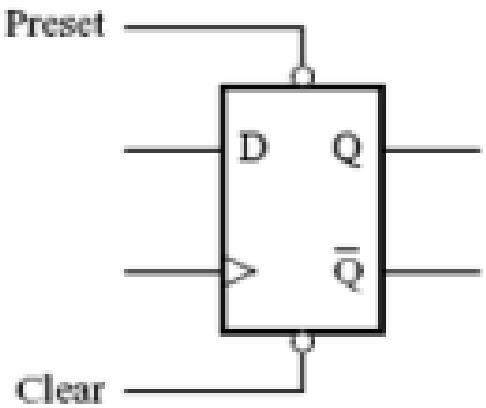
Flip-Flops with Clear and Preset inputs

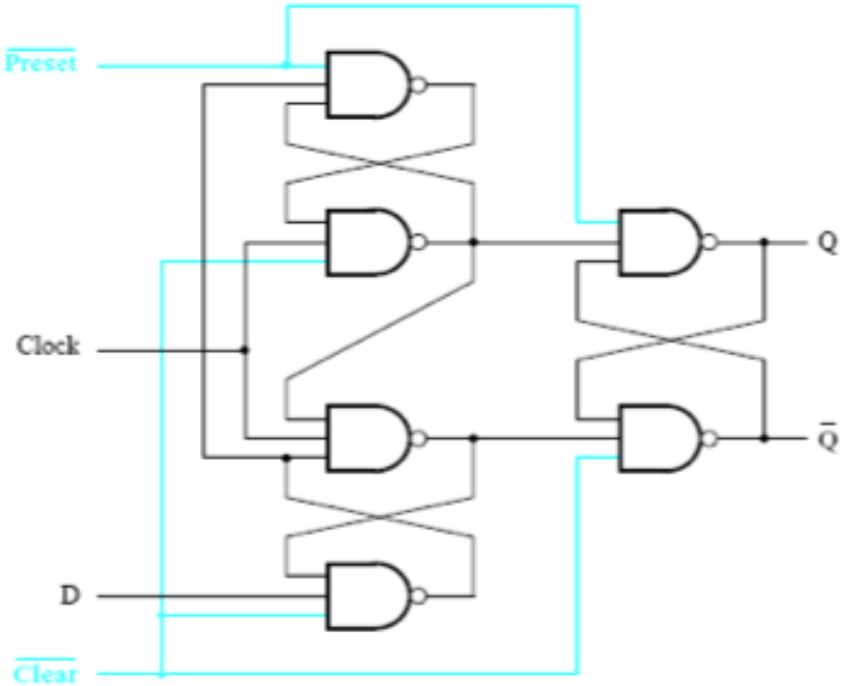
- It is important for the design of sequential circuits to be able to set flip-flops to predetermined values.
- This may mean that some flip-flops should be "1" while others will be "0".
  - Preset: Sets the flip-flop to 1
  - Clear: Sets the flip-flop to 0



Asynchronous reset

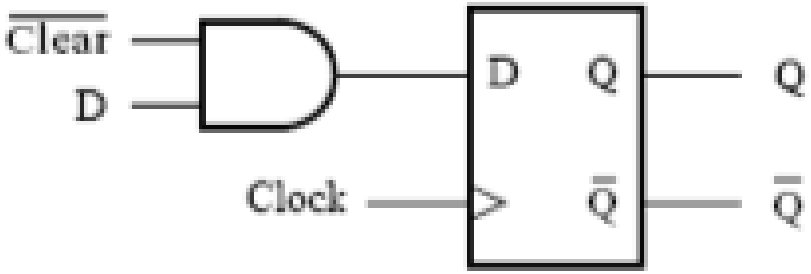
- An asynchronous reset (clear) means that the flip-flop will change its state to 0 immediately after the rest is active





Synchronous reset

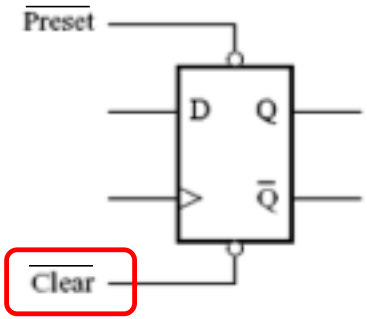
- A synchronous reset causes the flip-flop to take state 0 at the next clock edge
- Synchronous reset is implemented with an additional logic

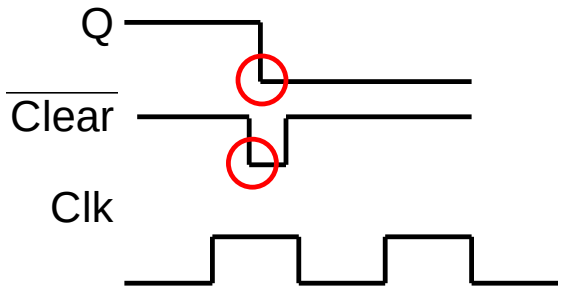


Clk	D	Q(t+1)
↑	0	0
↑	1	1

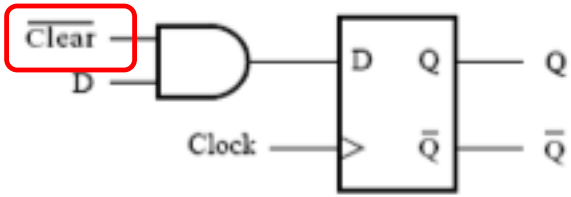
Asynchronous/Synchronous Reset

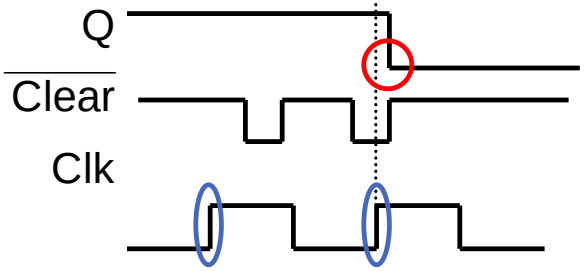
Asynchronous reset





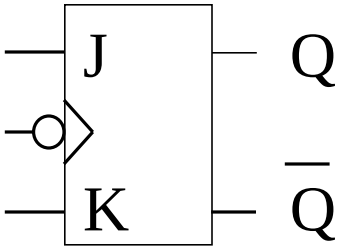
Synchronous reset





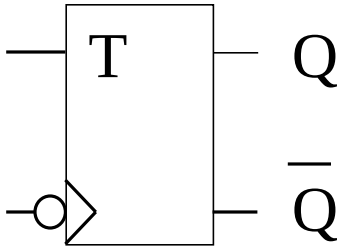
Other common types of flip-flops

JK flip-flop (by Jack Kilby - Nobel Prize 2000)



Clk	J	K	Q	$\overline{Q}$
↓	0	0	M	M
↓	0	1	0	1
↓	1	0	1	0
↓	1	1	Toggle	Toggle

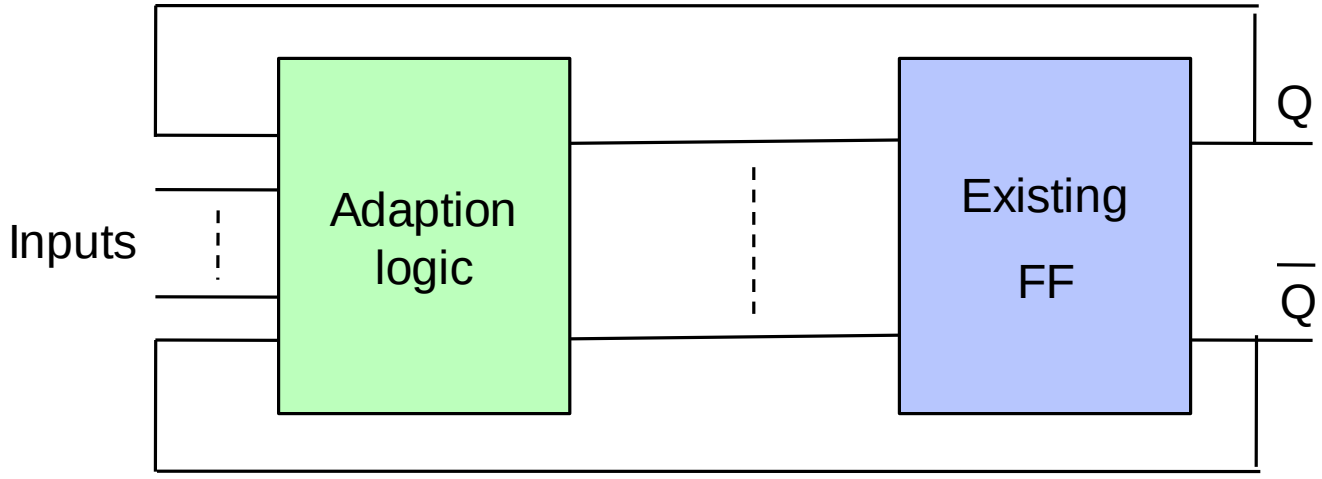
T-flip-flop (T = Toggle)



Clk	T	Q	$\overline{Q}$
↓	0	M	M
↓	1	Toggle	Toggle

Construction of new flip-flops

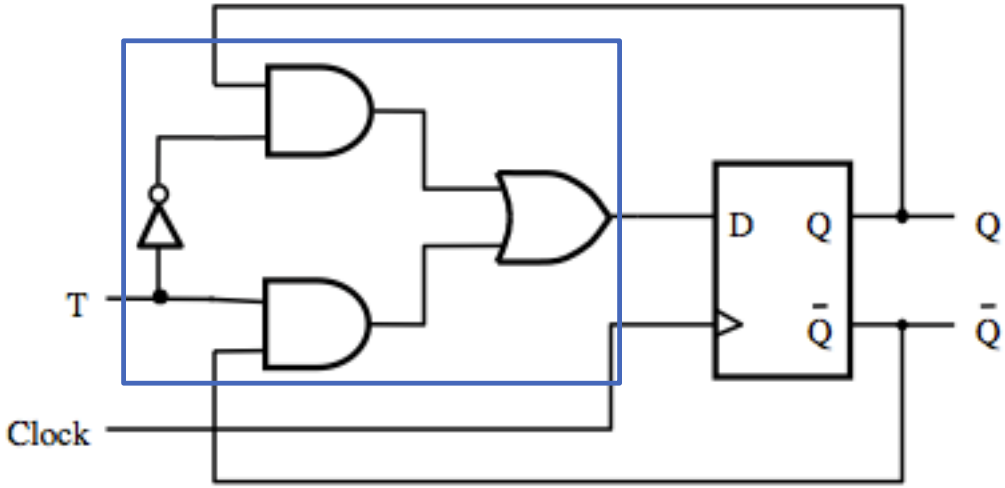
- One can construct new flip-flops based on the existing types



Construction of the T flip-flop with D flip-flop

- One can construct the new flip-flops based on an existing type

$$D = Q \quad hold$$



$$D = \overline{Q} \quad toggle$$

Clk	T	Q(t+1)
↑	0	Q(t)
↑	1	$\overline{Q(t)}$

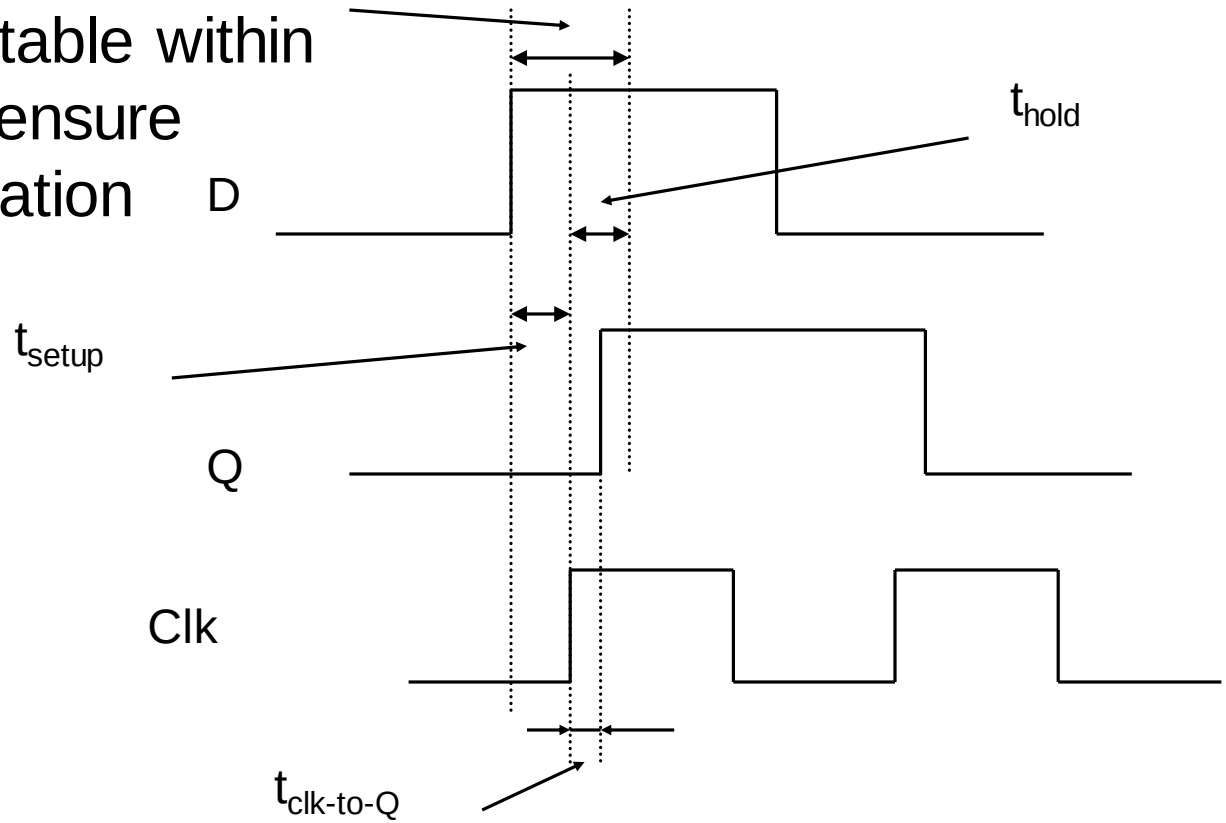
Toggles at each positive edge of clock

Timing Analysis

- It is possible to determine the maximum frequency in a sequential circuit by having information about
  - Gate delays t_{logic}
  - Setup time t_{su} of flip-flops
  - Hold time t_h of flip-flops
  - Clock-to-output t_{cQ} time

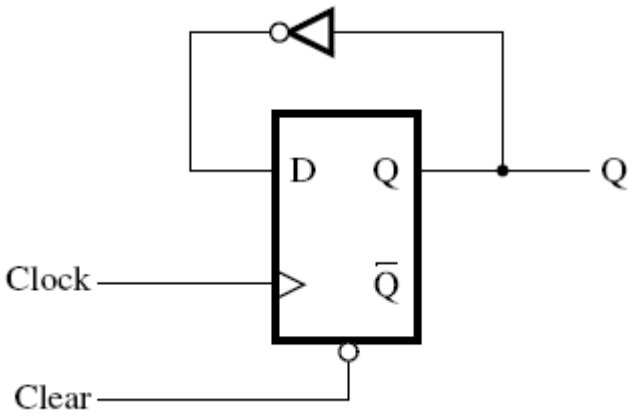
Setup & Hold Time

D must be stable within this area to ensure correct operation



What is the maximum frequency?

- Gate delays
  - $t_{logic} = t_{NOT} = 1.1$ ns
- Setup time
  - $t_{su} = 0.6$ ns
- Hold time
  - $t_h = 0.4$ ns
- Clock-to-output
  - $t_{cQ} = 1.0$ ns



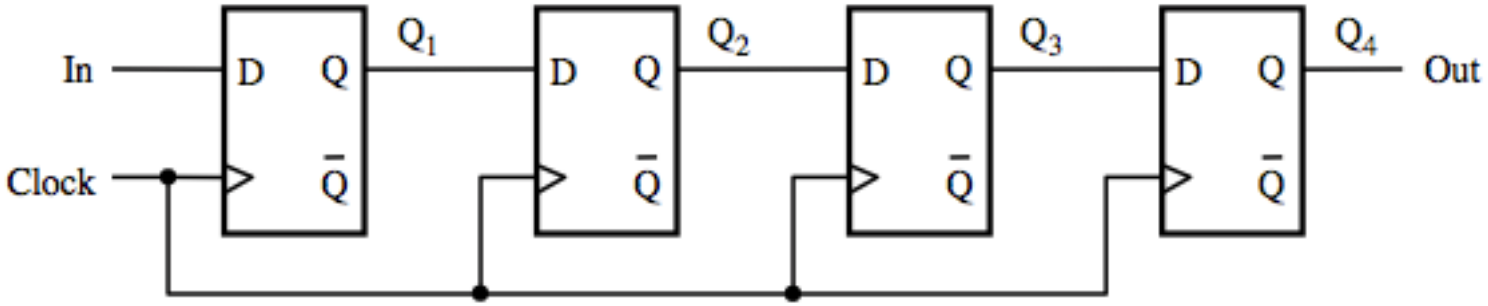
0.6 0.4 < 1.0 1.1

$$T = t_{su} + \max(t_h, t_{cQ}) + t_{logic} = 2.7 \text{ ns}$$

$$F = 1/T = 370 \text{ MHz}$$

Shift Register

- A shift register contains several flip-flops
- For each clock cycle, we shift all values from left to right
- Many designs use shift registers and values Q_4, ..., Q_1 as input to other components

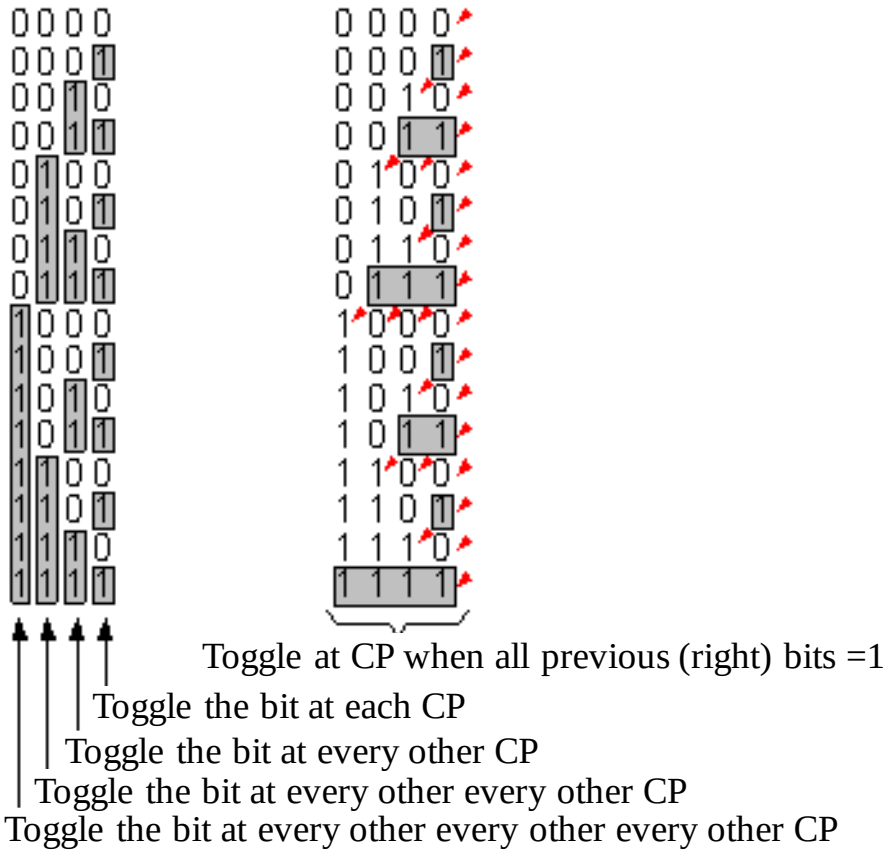


Counters

- A counter is a special type of sequential circuit that records the number of incoming clock pulses.
- Registration is usually done in the binary code.
- After a certain number of pulses the counter reaches its final state and then it starts from the beginning.
- The counter does not need to have any inputs except the clock pulses (which then can then be viewed as the input signal).

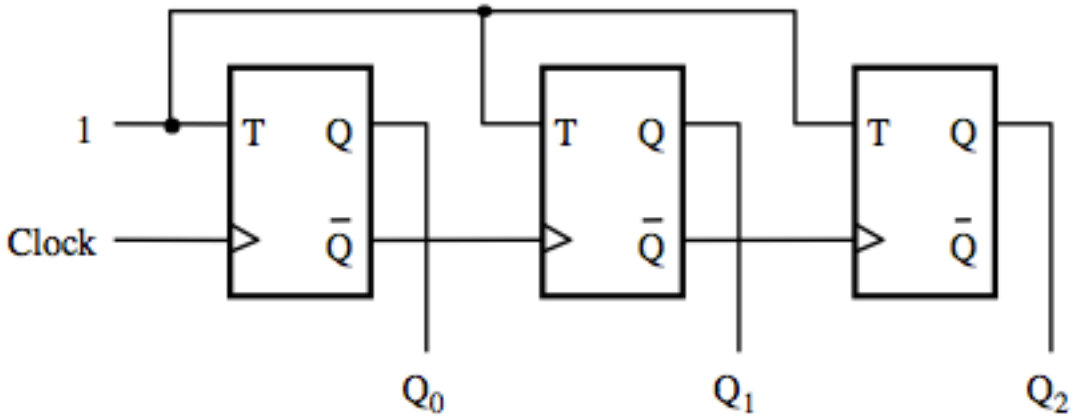
Binary Code counting properties

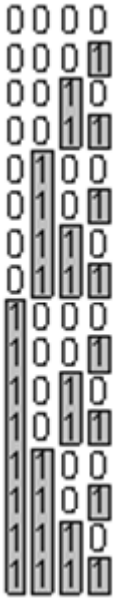
There are two different "rules" for constructing the binary code from the less significant bits.
Example with binary code 0 ... 15.



Asynchronous counter

- One can realize a counter with flip-flops
- The example below shows an *asynchronous* counter
- Some clock inputs are coupled to the $\overline{Q}$ output of the previous flip-flop

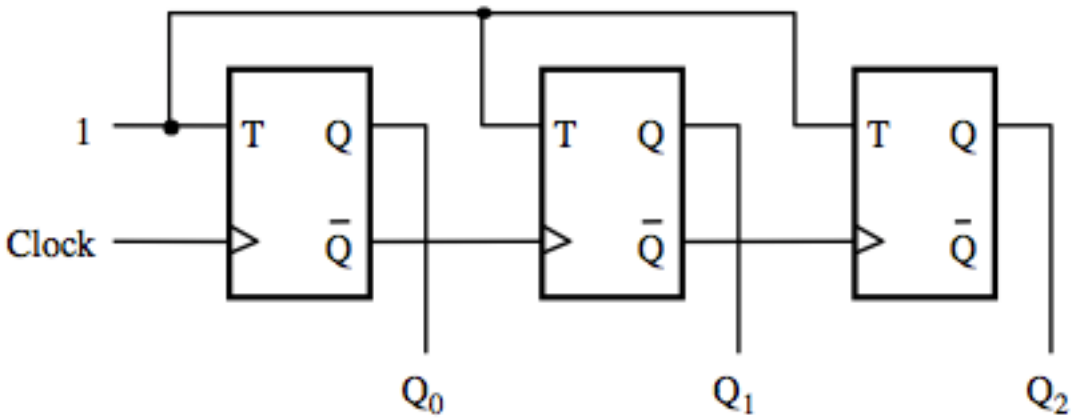


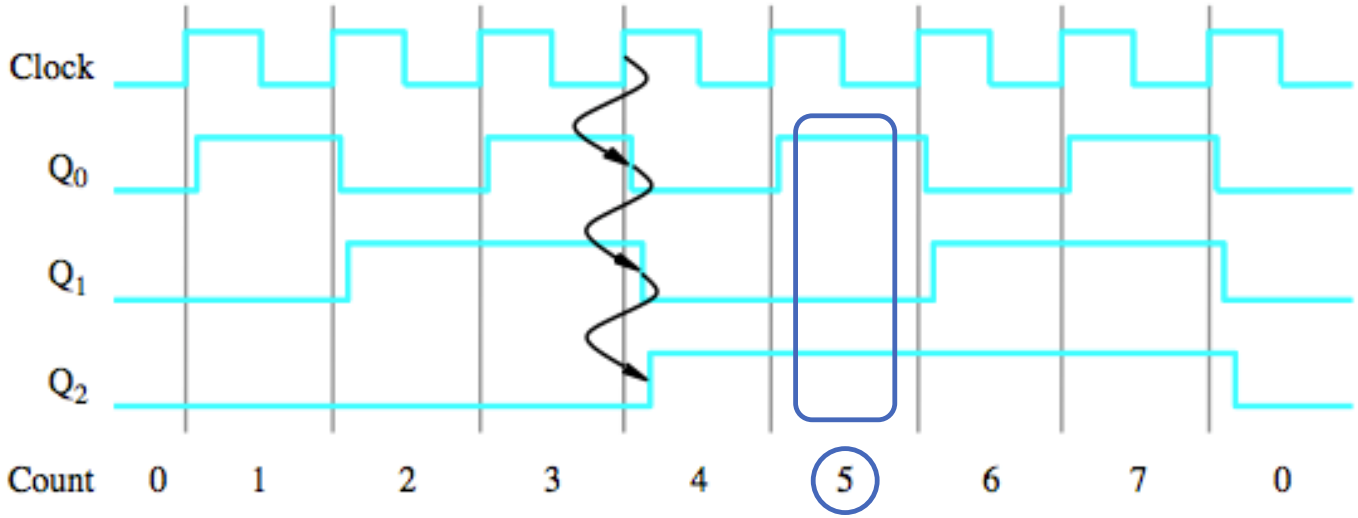


Asynchronous 3-bit counter

Clk	T	Q(t+1)
↑	0	Q(t)
↑	1	$\overline{Q(t)}$

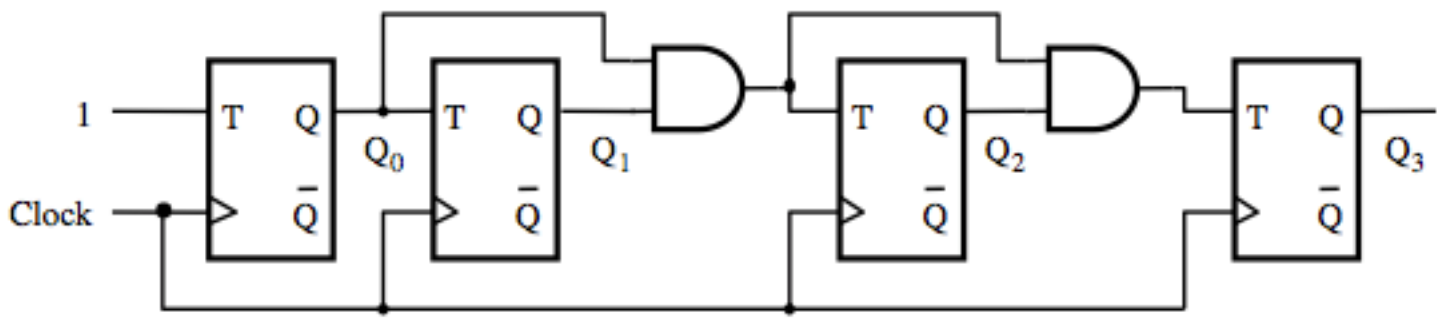
Toggles at each positive edge of clock





Synchronous counter

- In a *synchronous* counter clock inputs of flip-flops are connected to the same clock signal



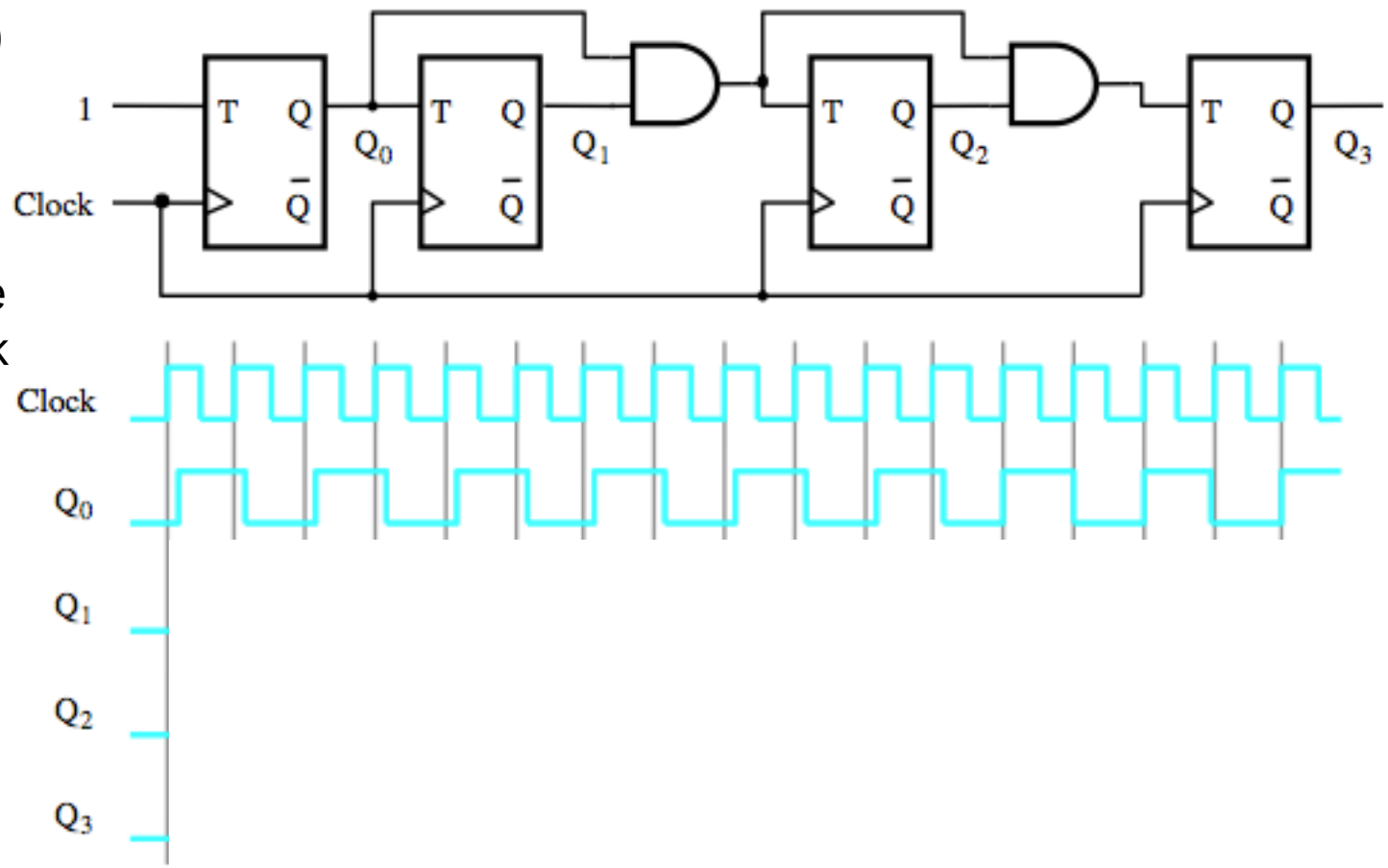
The first flip-flop has T = 1, and it toggles on every clock pulse. The other flip-flops toggle if all of their previous flip-flops stand at "1". This condition is obtained from the AND gates.

Toggle at CP when all previous (right) bits =1

Synchronous counter

Clk	T	Q(t+1)
↑	0	Q(t)
↑	1	$\overline{Q(t)}$

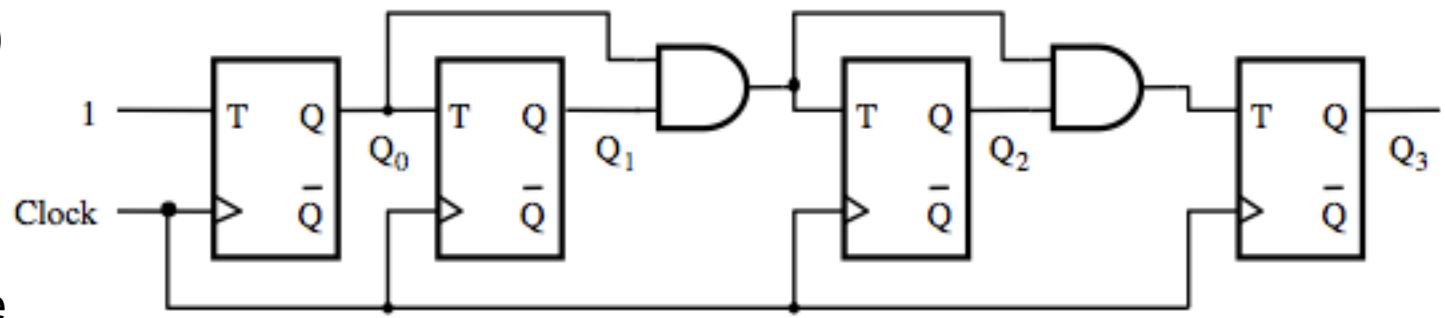
Toggles at each positive edge of clock

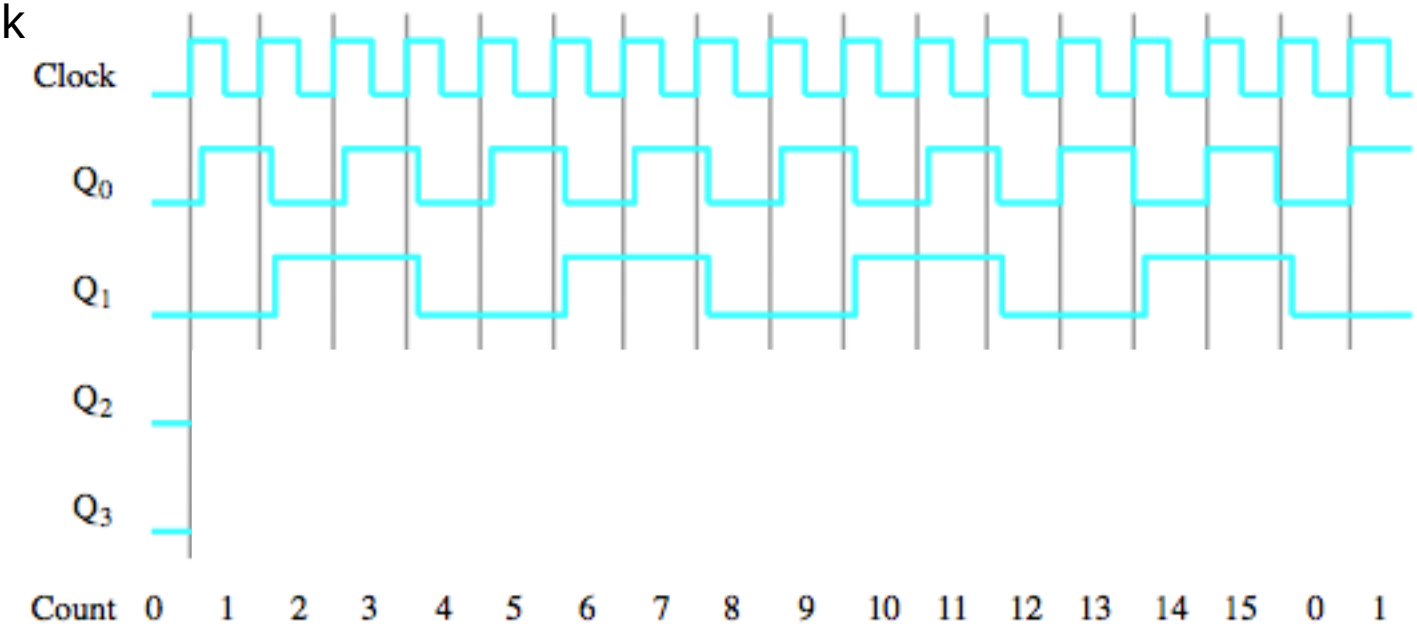


Synchronous counter

Clk	T	Q(t+1)
↑	0	Q(t)
↑	1	$\overline{Q(t)}$

Toggles at each positive edge of clock

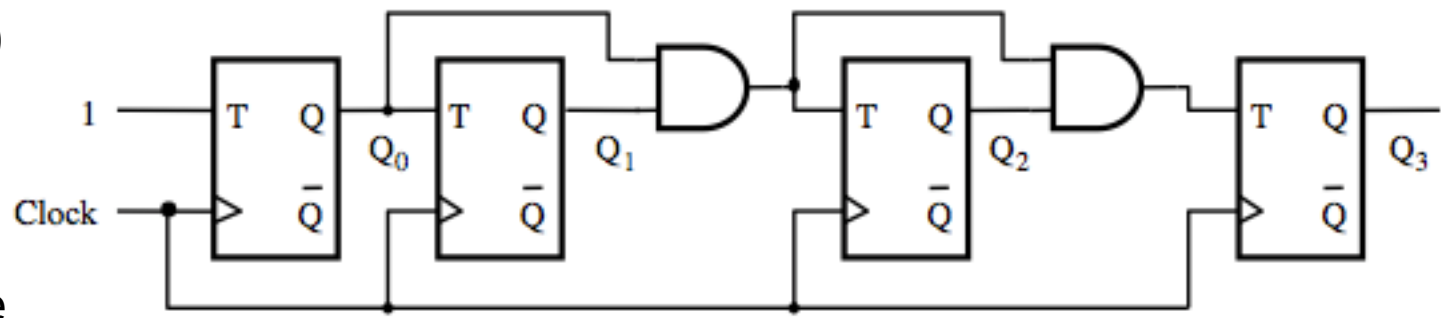


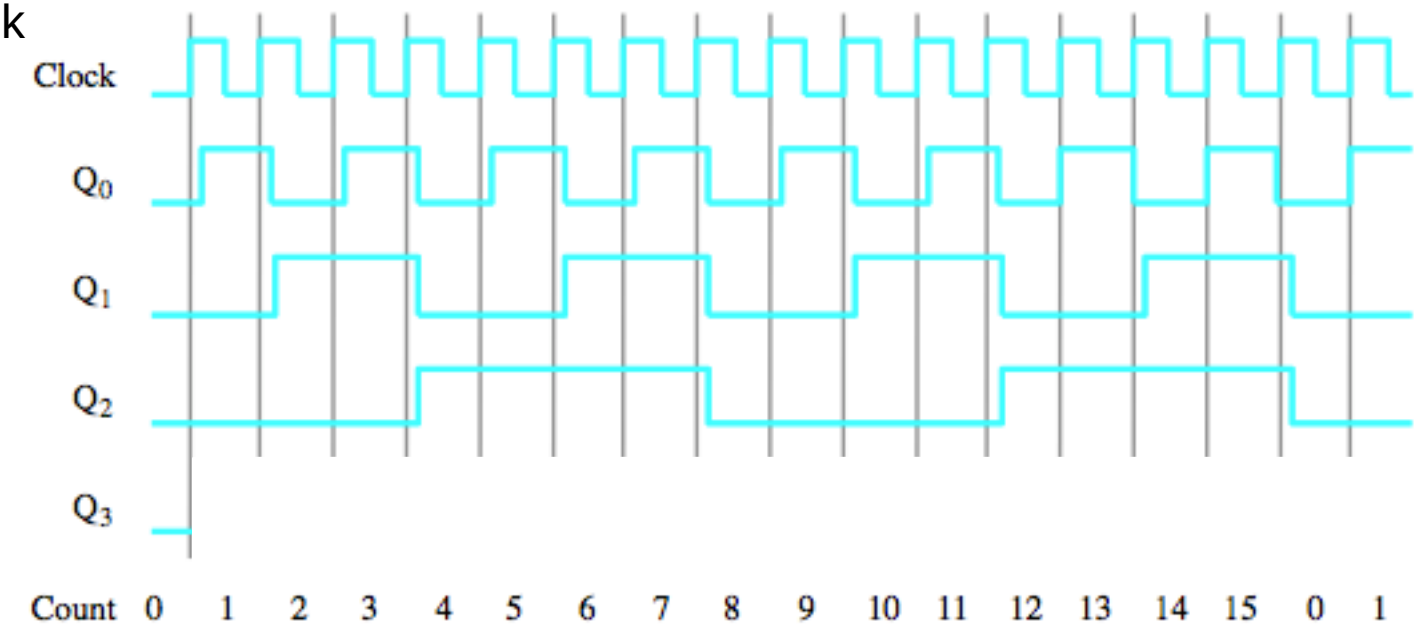


Synchronous counter

Clk	T	Q(t+1)
↑	0	Q(t)
↑	1	$\overline{Q(t)}$

Toggles at each positive edge of clock

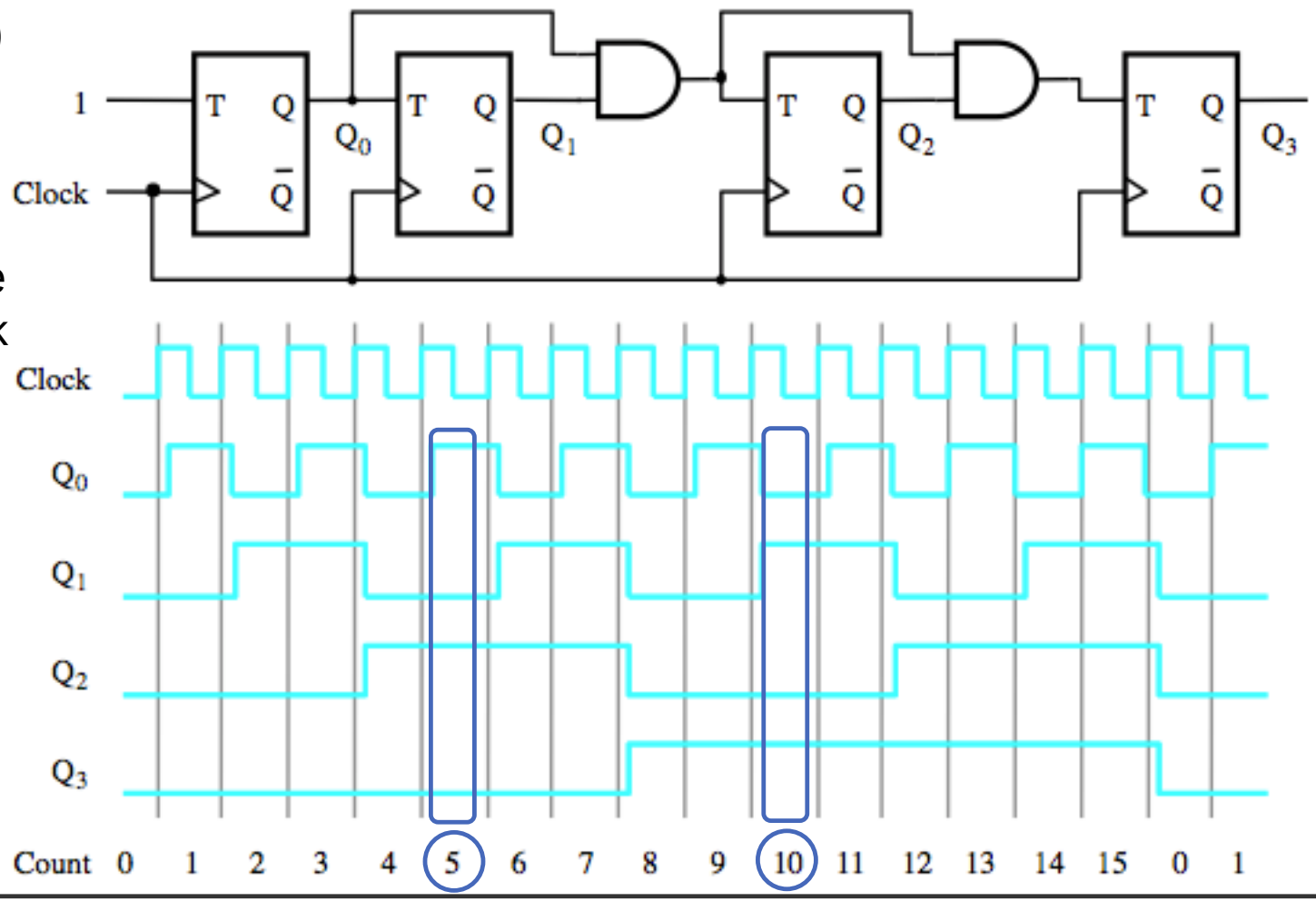




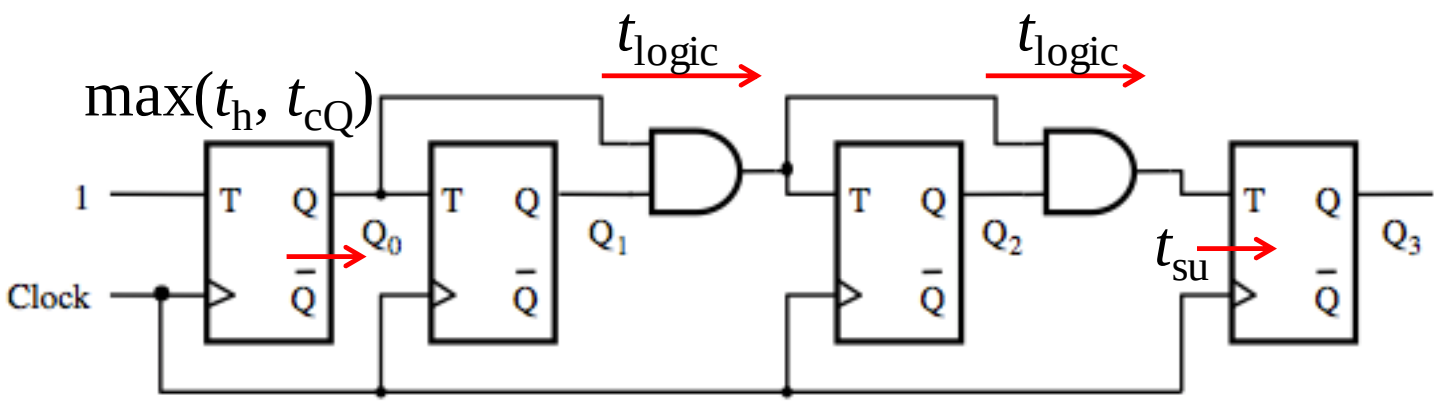
Synchronous counter

Clk	T	Q(t+1)
↑	0	Q(t)
↑	1	$\overline{Q(t)}$

Toggles at each positive edge of clock



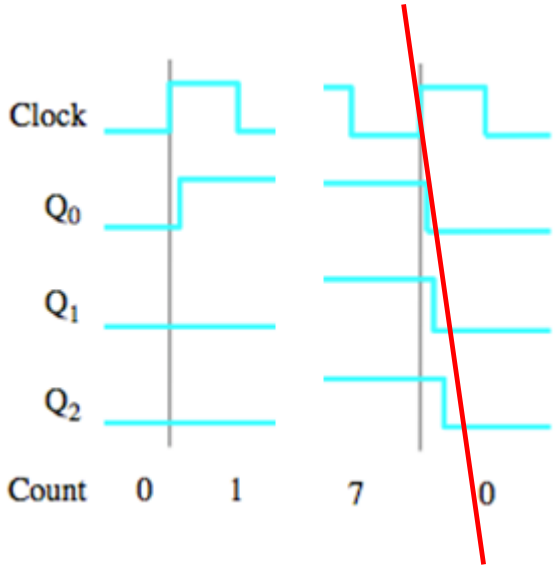
What is the maximum frequency?



- The critical path determines the maximum frequency!
- This is the longest combinational path from Q_0 through the two AND gates to the input of flip-flop that computes Q_3
  - t_{logic} thus is equivalent to the delay of two AND gates

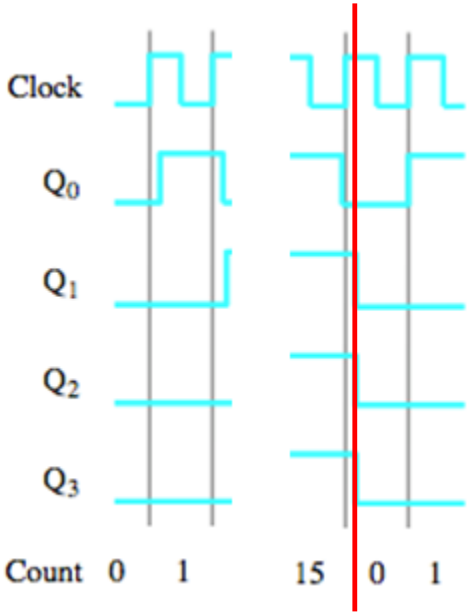
Asynchronous vs. Synchronous counter

Asynchronous counter



The output signals are delayed more and more with every step

Synchronous counter



The output signals have the same delay

VHDL for flip-flops and latches

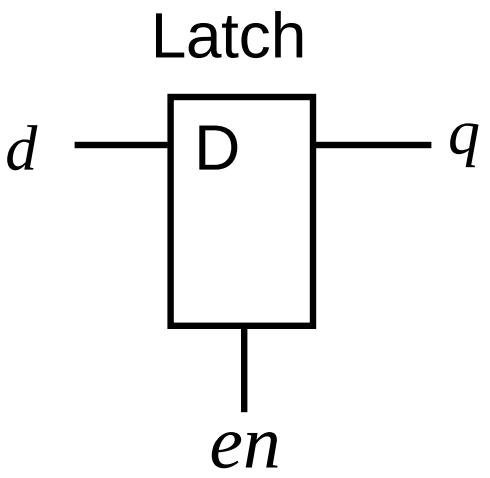
Programmable logic has embedded flip-flops. How to write VHDL code that "tells" the compiler that you want to use them?

A D-latch in VHDL

```
ENTITY D_Latch IS
    PORT(en : IN std_logic;
         d : IN std_logic;
         q : OUT std_logic);
END ENTITY D_Latch;

ARCHITECTURE RTL OF D_Latch IS
BEGIN
    PROCESS(en, d)
    BEGIN
        IF en = '1' THEN
            q <= d;
        END IF;
    END PROCESS;
END ARCHITECTURE RTL;
```

No else?



Enable	D	Q
0	-	M
1	D	D

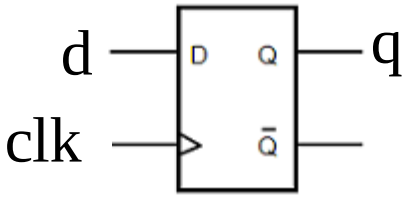
Latch as a process

```
PROCESS(en, d)
    BEGIN
        IF en = '1' THEN
            q <= d;
        END IF;
END PROCESS;
```

Latches are generally considered to be bad from the synthesis point of view because they are not always testable.

Therefore one avoids latches. (Programmable Logic has embedded flipflops with asynchronous Preset and Clear that you can use).

Flip-flop as a process



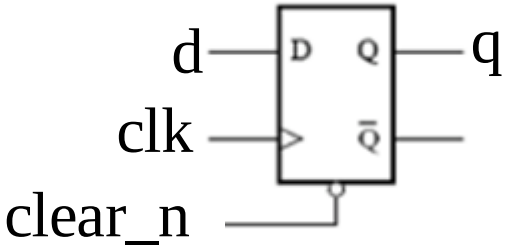

```
PROCESS(clk)
    BEGIN
        IF rising_edge(clk) THEN
            q <= d;
        END IF;
END PROCESS;
```

Only one edge is allowed per process

Instead of the function "`rising_edge(clk)`" you can write "`clk'event and clk=1`"

The compiler will "understand" that this is a flip-flop and using one of the built-in flip-flops to implement the process.

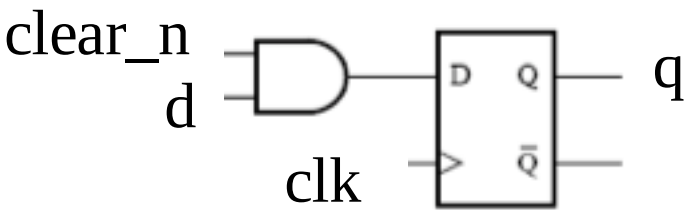
With asynchronous RESET



Clear independent of clk

```
PROCESS(clk, clear_n)
    BEGIN
        IF clear_n = '0' THEN
            q <= '0';
        ELSE IF rising_edge(clk) THEN
            q <= d;
        END IF;
END PROCESS;
```

With synchronous RESET




```
PROCESS(clk)
    BEGIN
        IF rising_edge(clk) THEN
          IF clear_n = '0' THEN
              q <= '0';
            ELSE
              q <= d;
           END IF;
    END IF;
END PROCESS;
```

Counters and other sequential circuits

What does this "counter"?

```
bcd:
PROCESS(clk)
    BEGIN
        IF rising_edge(clk) THEN
            IF (count = 9) THEN
                count <= 0;
            ELSE
                count <= count+1;
            END IF;
        END IF;
END PROCESS;
```

Summary

- Memory Elements
  - Latches
  - Flip-Flops
- Shift registers
- Counters
- Next lecture: BV pp. 485-507