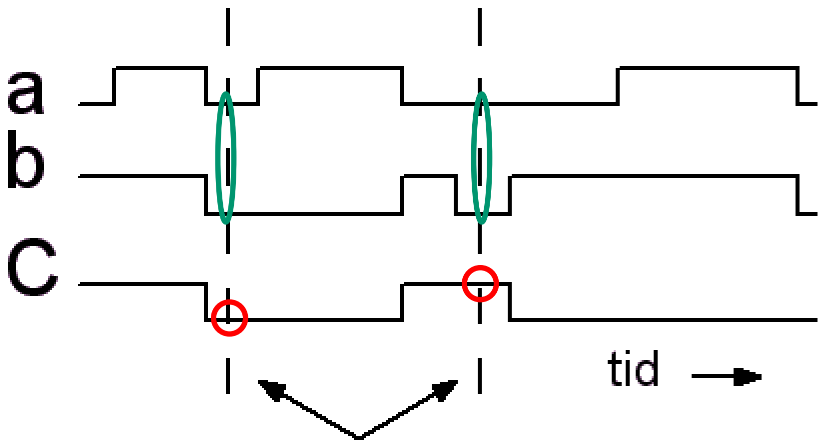
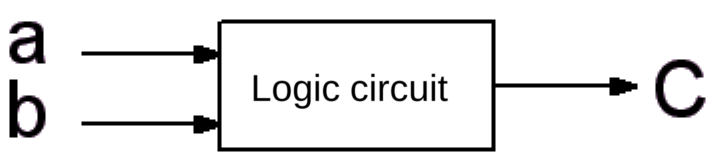
Sequential circuits



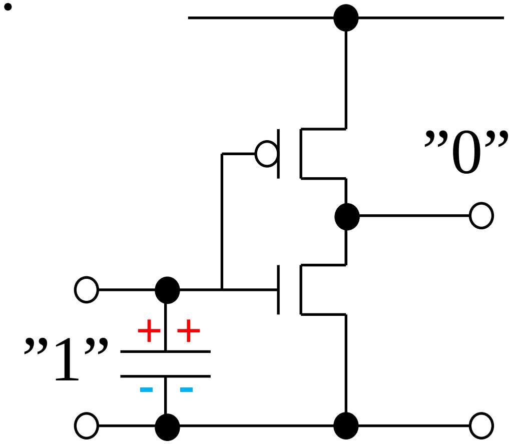
Same input a, b can produce different output C.



If the same input may produce different output signal, 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 can hardware remember?

• To remember something, then we must somehow store the information.

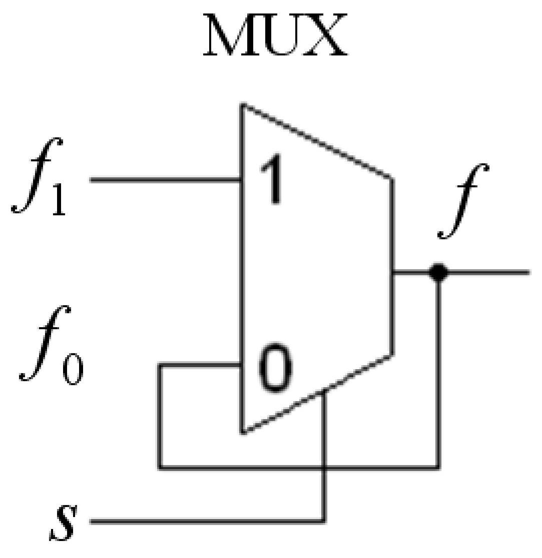


• One way is to store information is in the form of a charge on a Capacitance (DRAM).

There are other possibilities ...

"Latching"

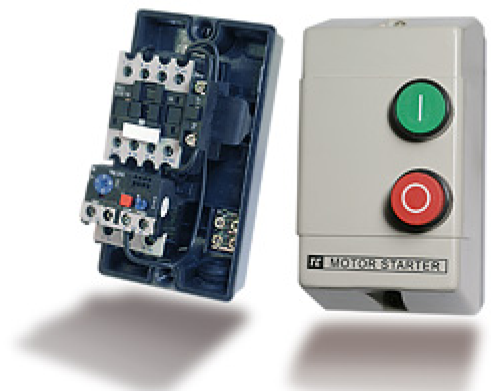
s	f_1	f_0	f
0	–	f_0	f_0
1	f_1	–	f_1

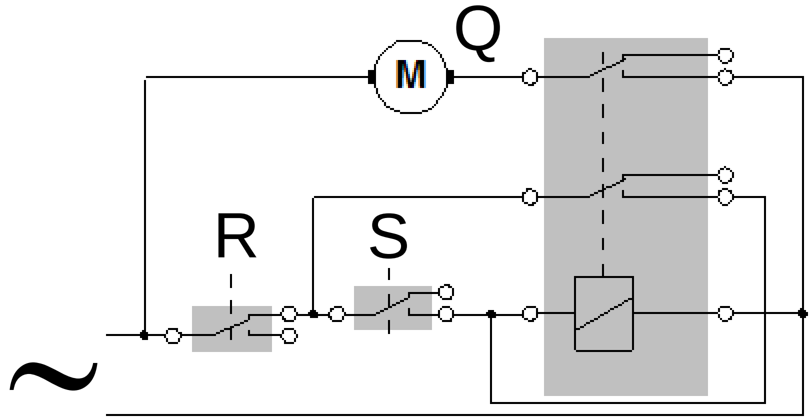


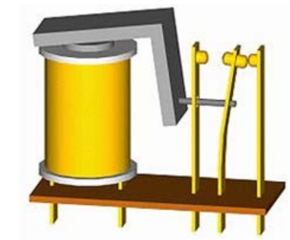
If s = 1 the output f follows the input f_1. When s becomes s = 0 the circuit ”latches” to the value f had in the moment ***before*** the transition s = 0.

$$s = follow/\overline{latch}$$

(Motor Protection)





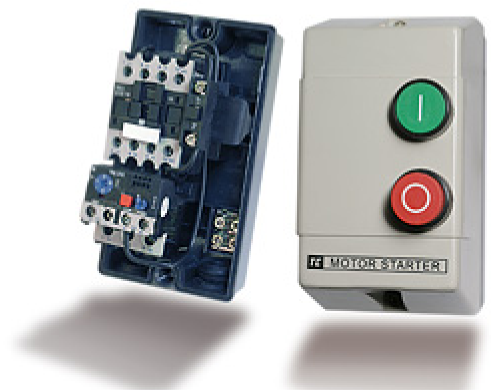


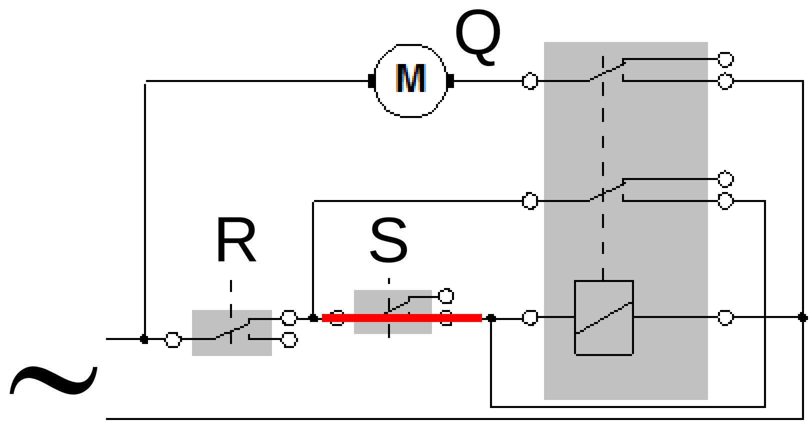
Relay

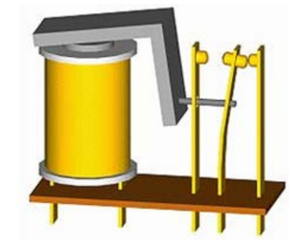
A **Motor protection circuit braker** is a relay with a latching contact.

- One need only press once for the engine to start.
- Will there be a power failure, so do not the engine start suddenly by itself when the power comes back - a good safety feature.
- The lights light up immediately, however - it is also good.

(Motor Protection)







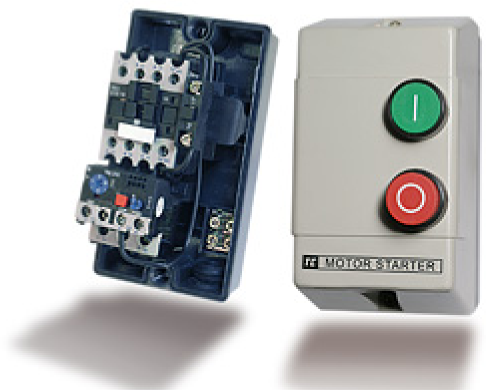
Relä

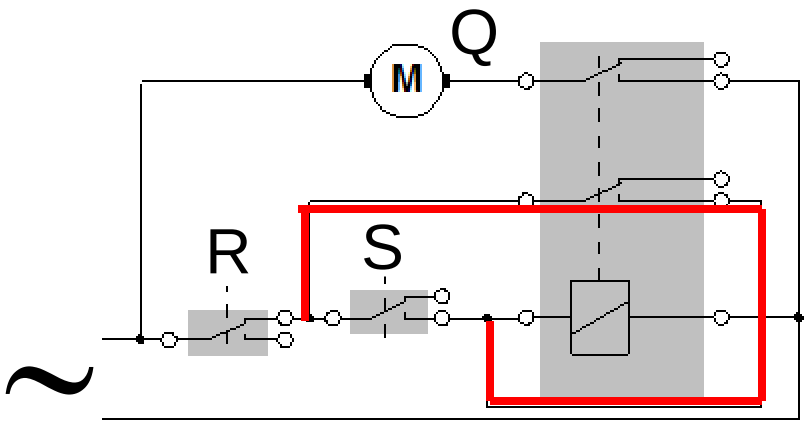
A **Motor protection circuit braker** is a relay with a latching contact.

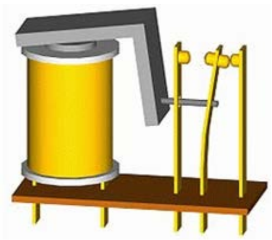
- One need only press once for the engine to start.
- Will there be a power failure, so do not the engine start suddenly by itself when the power comes back - a good safety feature.
- The lights light up immediately, however - it is also good.

William Sandqvist william@kth.se

(Motor Protection)







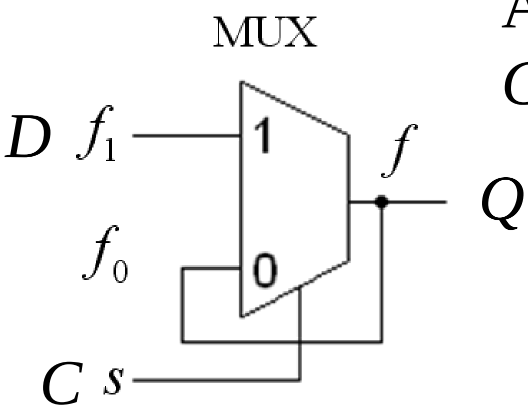
Relä

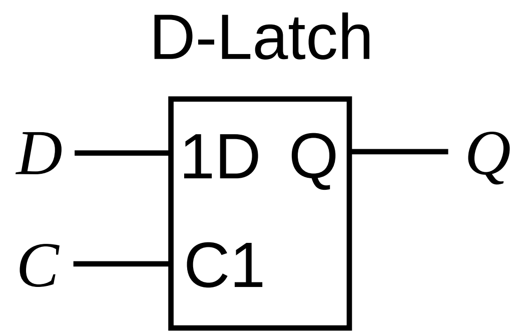
A **Motor protection circuit braker** is a relay with a latching contact.

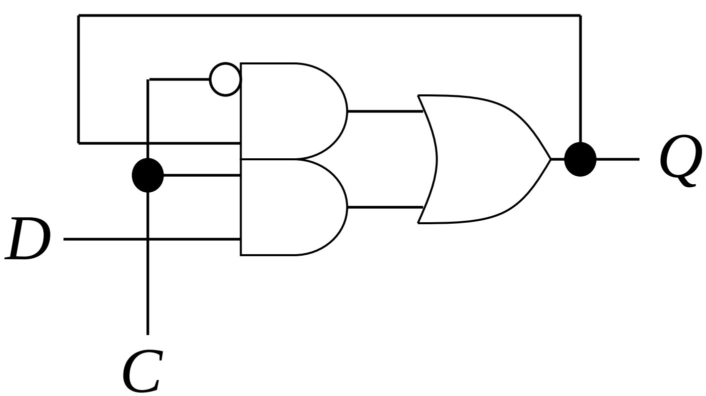
- One need only press once for the engine to start.
- Will there be a power failure, so do not the engine start suddenly by itself when the power comes back - a good safety feature.
- The lights light up immediately, however - it is also good.

D-Latch

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

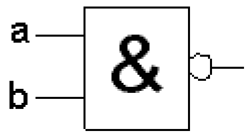
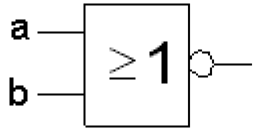






C $follow/\overline{latch}$	D	Q
0	–	M latch
1	D	D follow

NOR and NAND ”locking input signal”

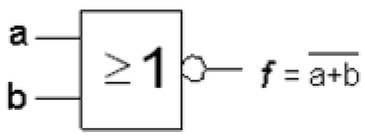
Name	Logic function - Gate
NAND	 $f = \overline{a \cdot b}$ a b | f 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR	 $f = \overline{a + b}$ a b | f 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0

Rule …

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

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

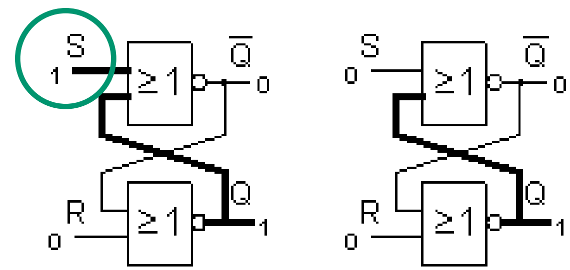
SR-latch with NOR-gates



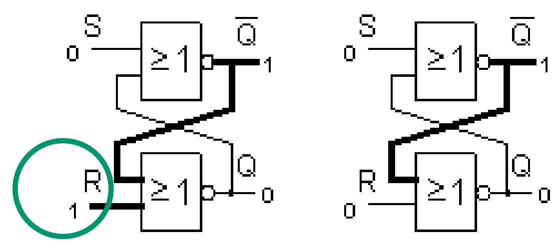
a	b	f
0	0	1
0	1	0
1	0	0
1	1	0

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".

Q=1

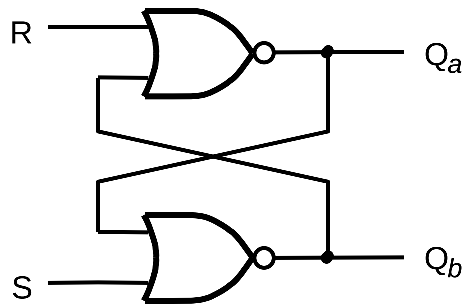


Q=0



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



(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	

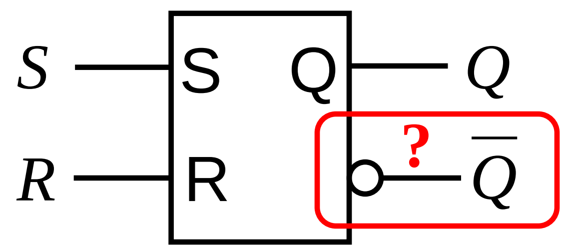
(b) Truth table

Forbidden input
S=R=1

$$Q_a \neq \overline{Q}_b$$

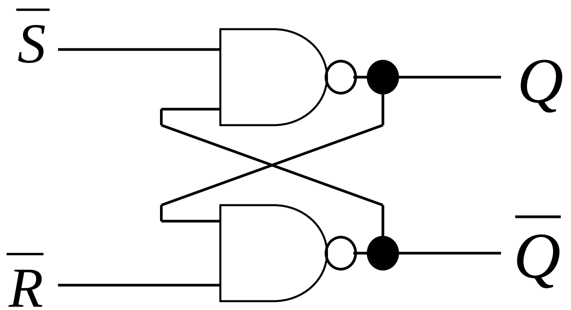
As long as one avoids the input signal S = R = 1 (= forbidden input combination), the outputs Q_a and Q_b will be each other's inverses. One can then use the symbol to the right.

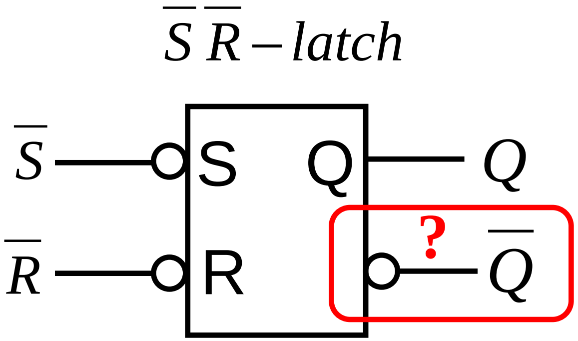
SR-Latch



If one takes signals from latches, thus inverses are always available!

$\overline{S}\overline{R}$-latch with NAND-gates



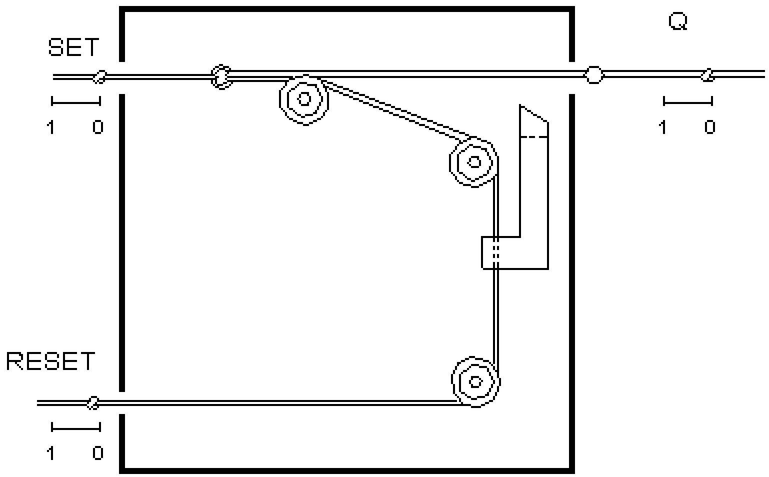


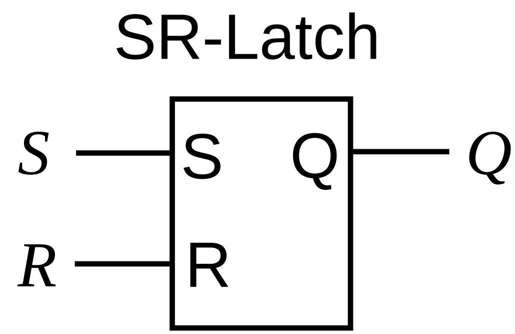
$\overline{S}$	$\overline{R}$	Q	$\overline{Q}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1	M	M

For NAND gates "0" is a latching input signal that forces the output to "1".

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

SR-Latch

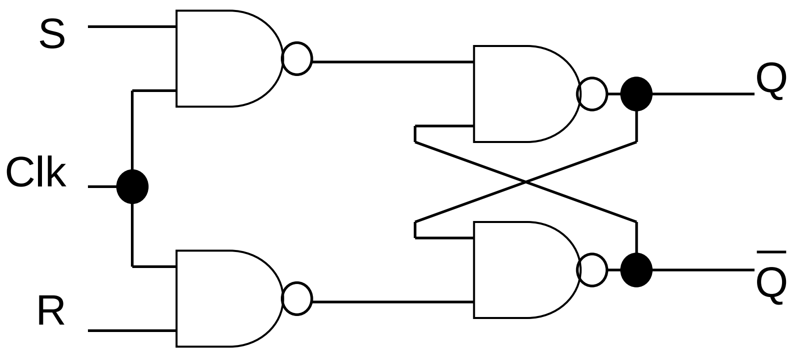




To the left we have an SR-latch with ropes - April 1-joke from Scientific American! Again there can be seen that you should not pull the SET and RESET ropes simultaneously!

(Gated SR-Latch)

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 $S = R = 1$ could be tolerated.



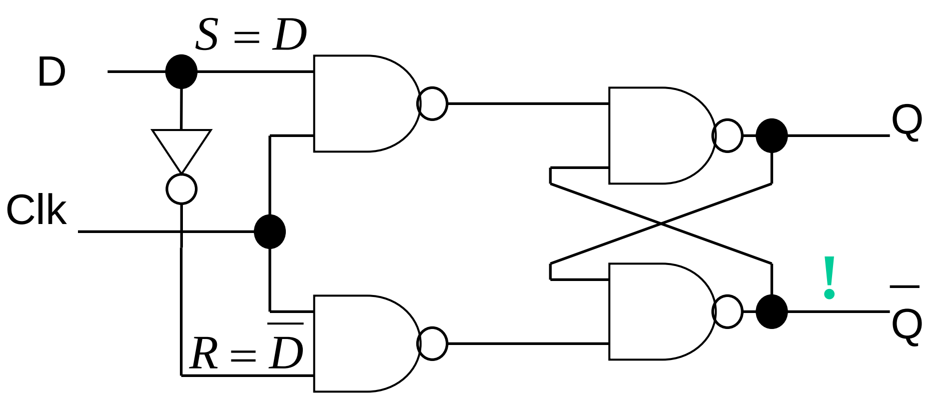
Clk	S	R	Q	$\overline{Q}$
1	0	0	M	M
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1
0	-	-	M	M

Forbidden combination

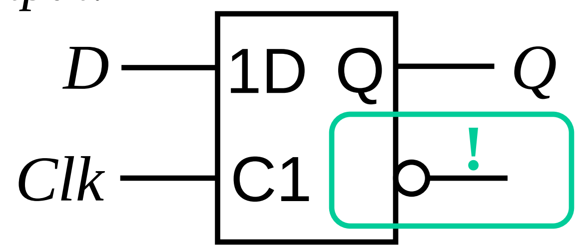
D-latch

A still better solution to the problem of the "forbidden" state is the D-latch. With an ***inverter*** one ensures that the *S* and *R* simply always has different values!

The latch output follows the *D* input when *Clk* = 1 to 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*.

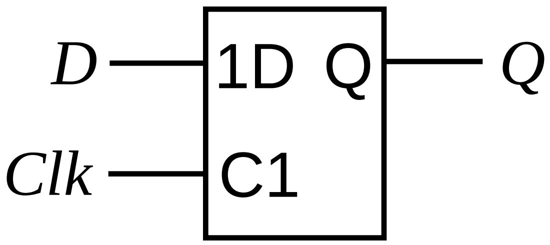


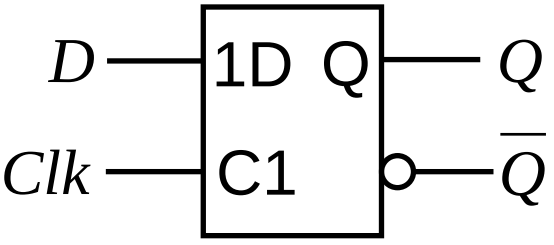
$$Clk = follow / \overline{latch}$$



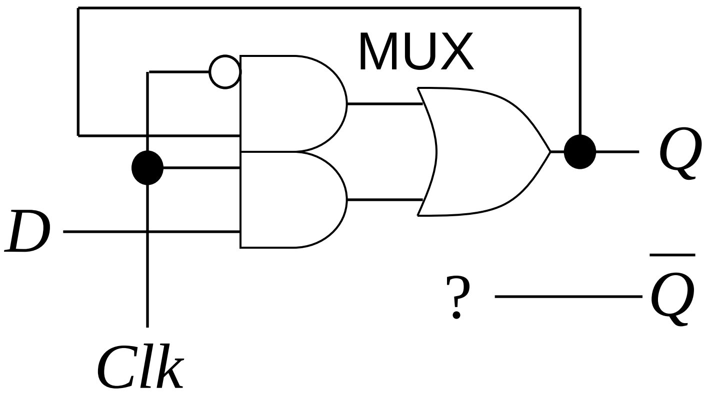
Clk	D	Q	$\overline{Q}$
1	0	0	1
1	1	1	0
0	-	M	M

Two different D-latches

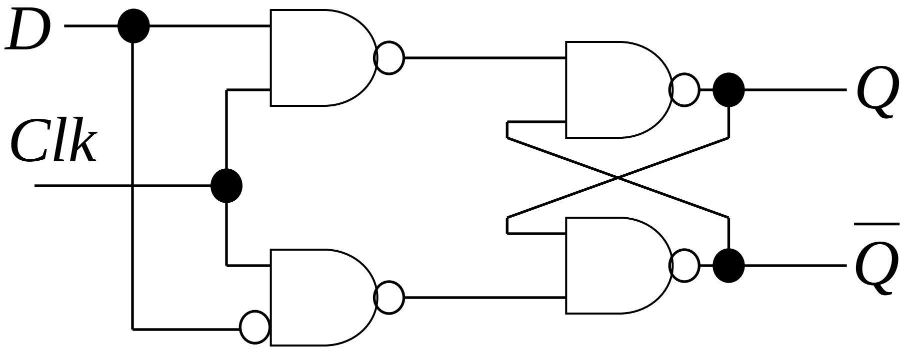




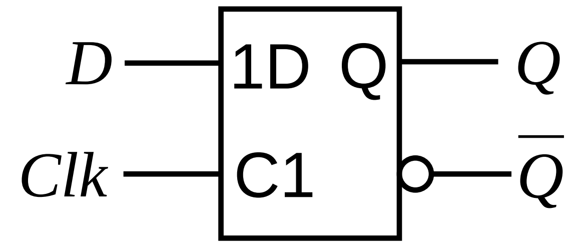
Long feedback (~4T)

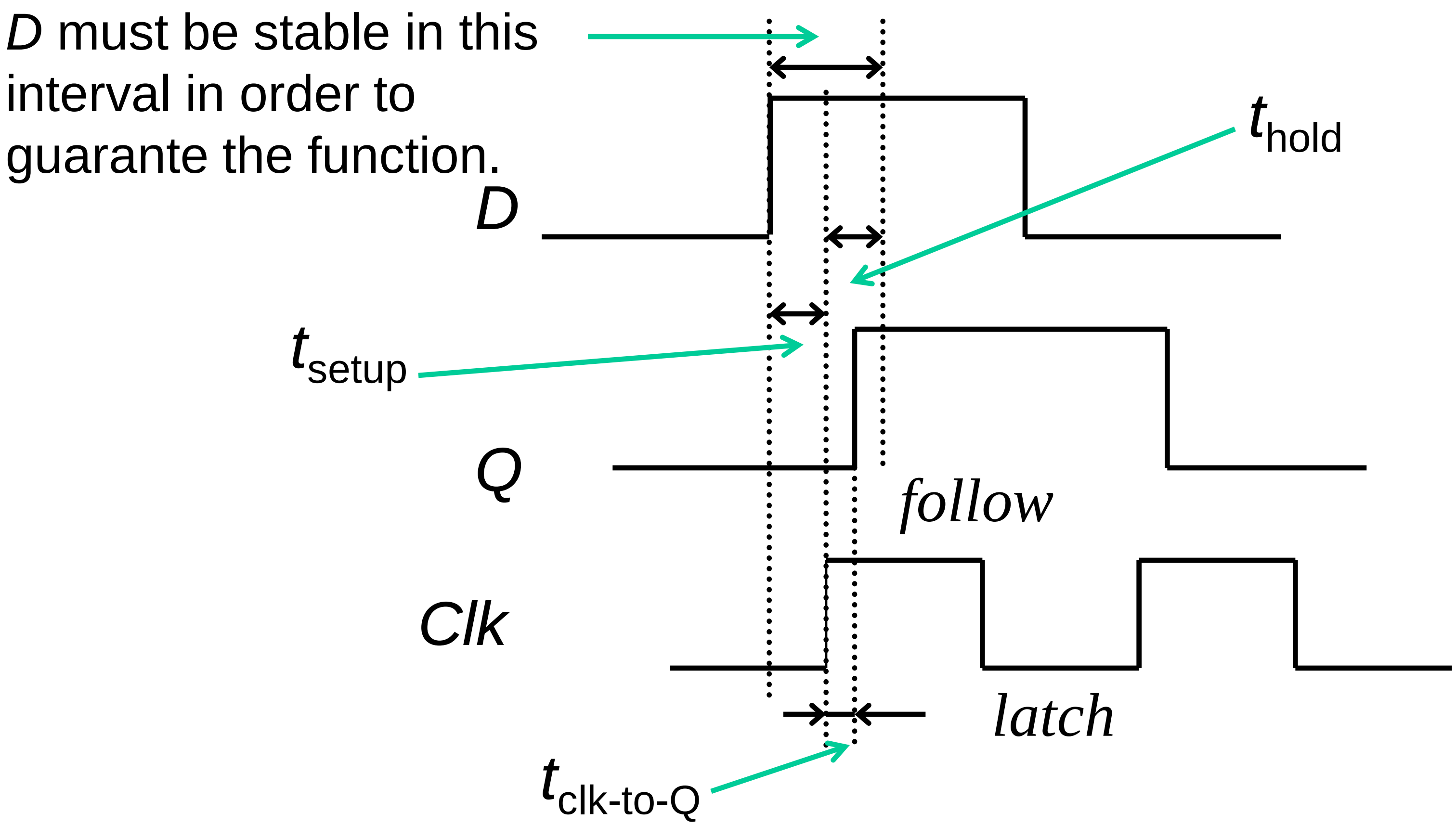


Short feedback (~1T)



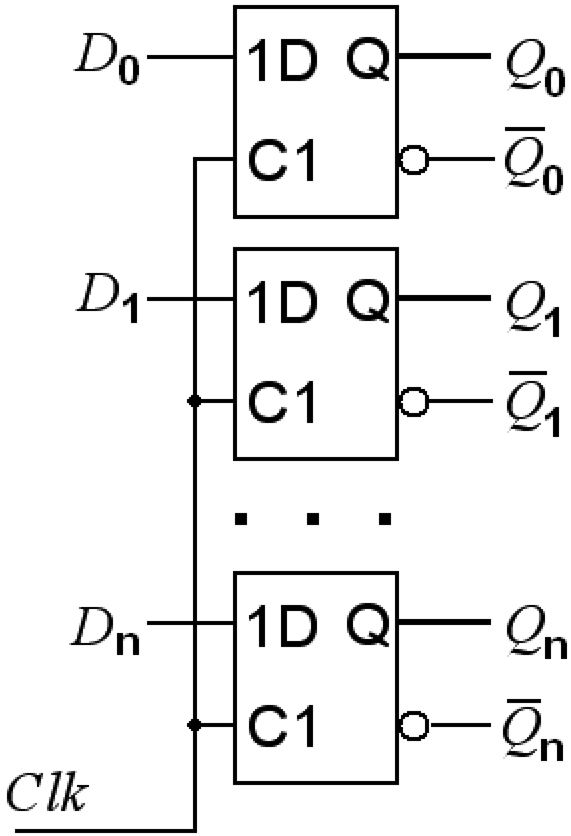
Setup- & Hold-time





William Sandqvist william@kth.se

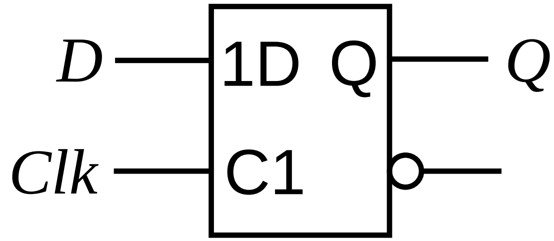
Register – inverted signals



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 in the calculation examples usually assumes that inverted signals are available.

Every other time?



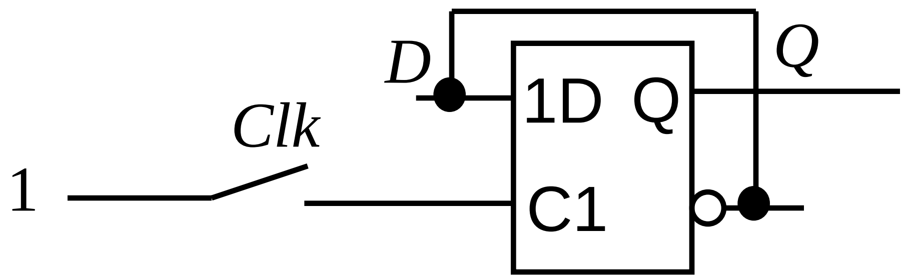
How do you construct a **sequential circuit** that will toggle its output 1/0 at every clockpulse, *Clk* ?

- The circuit needs to remember it's previous value Q
- And change this to $Q = D = \overline{Q}$.

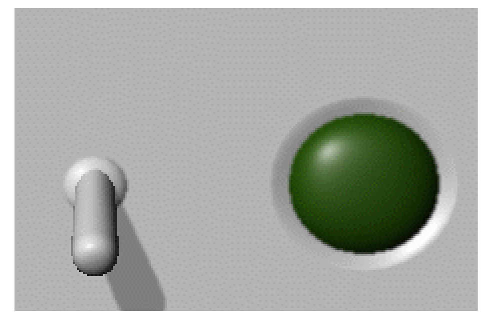
The latch has both "memory" and an inverted output - could it be used?

Not possible with a simple latch…

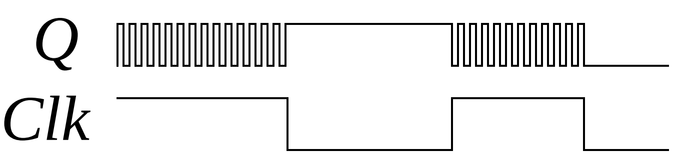
$$Clk = follow/\overline{latch}$$



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

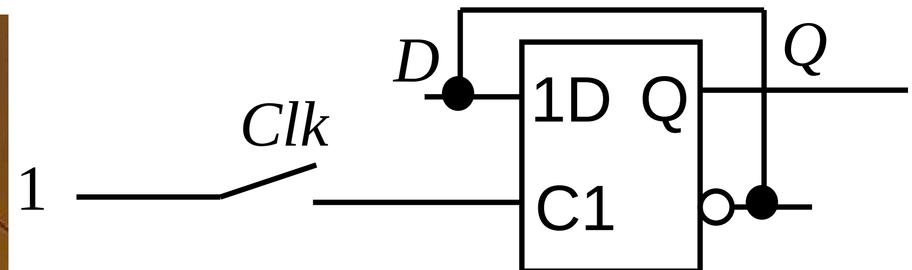


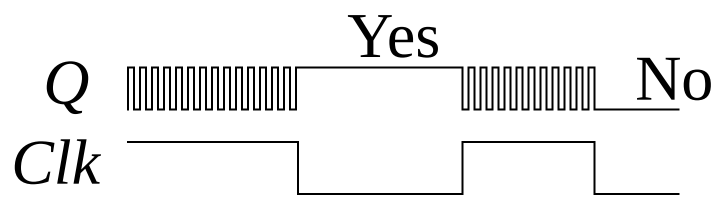
- When *Clk* = 1 the output follows the input – therefore the output changes 1/0 as quickly as possible! The circuit becomes an oscillator!



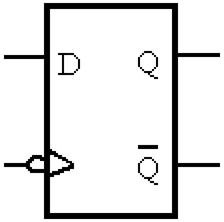
- Later when *Clk* = 0 the output retains its value 1/0 after what it happened to be. (= Random Number Generator?)

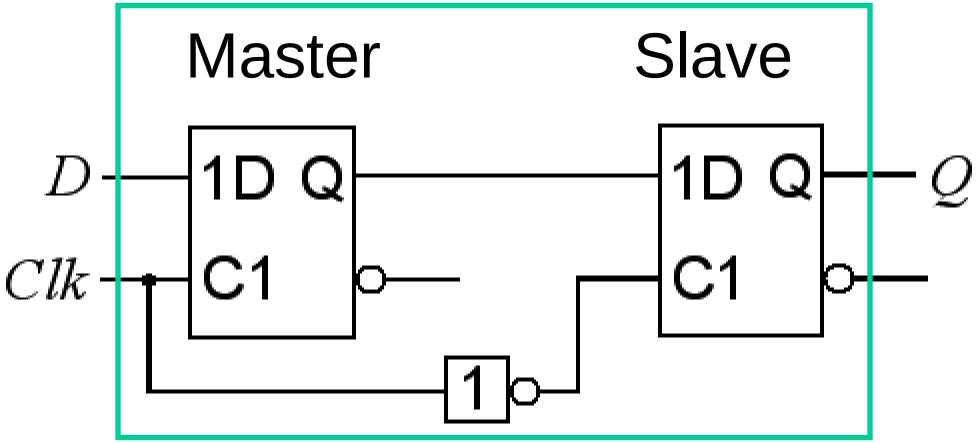
Voting Help in parliament?





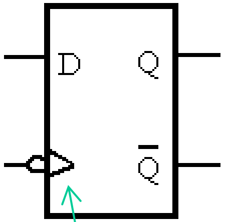
Clocked flip-flops
Master-Slave flip-flop



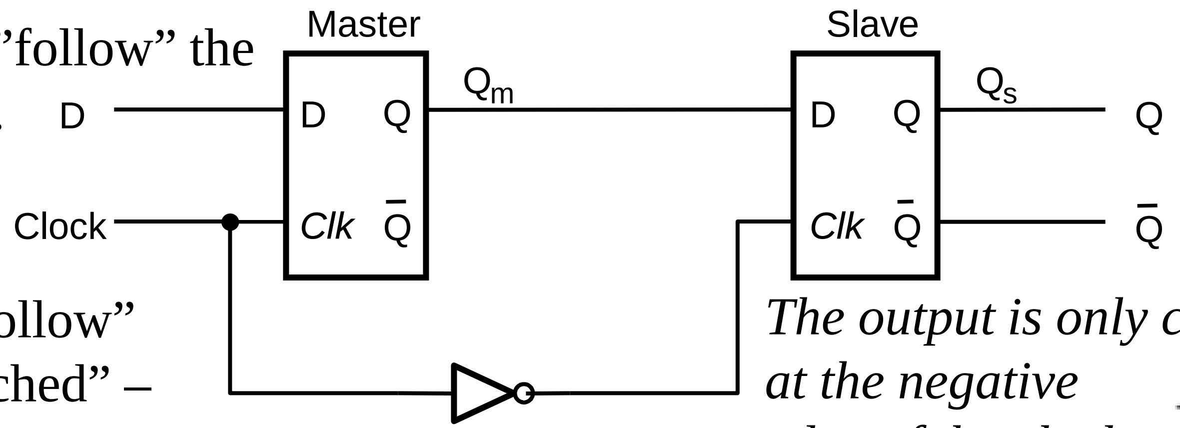


The problem is that the **simple latch** is open to change right up until it will unlock its value.
The solution is the **clocked flip-flop** consisting of several latches. One latch receives new data (Master) while another latch retaines the old data (Slave).

Timing diagram Master-Slave

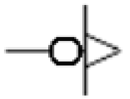


When **Master** do ”follow” the **Slave** is ”latched”.

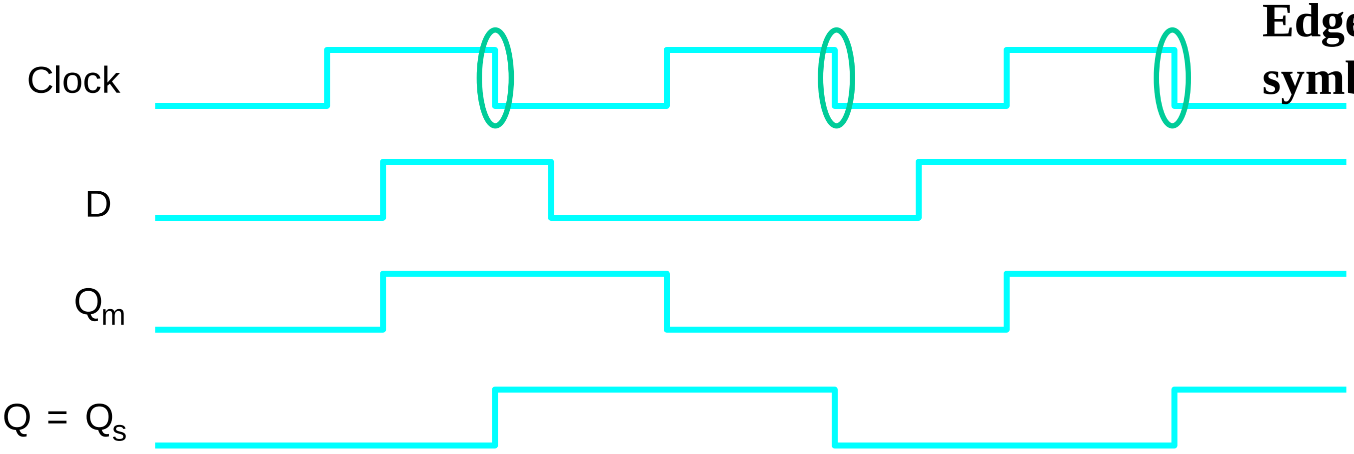


When **Slave** do ”follow” the **Master** is ”latched” – but then there is nothing to follow.

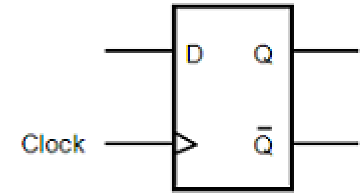
The output is only changed at the negative edge of the clock



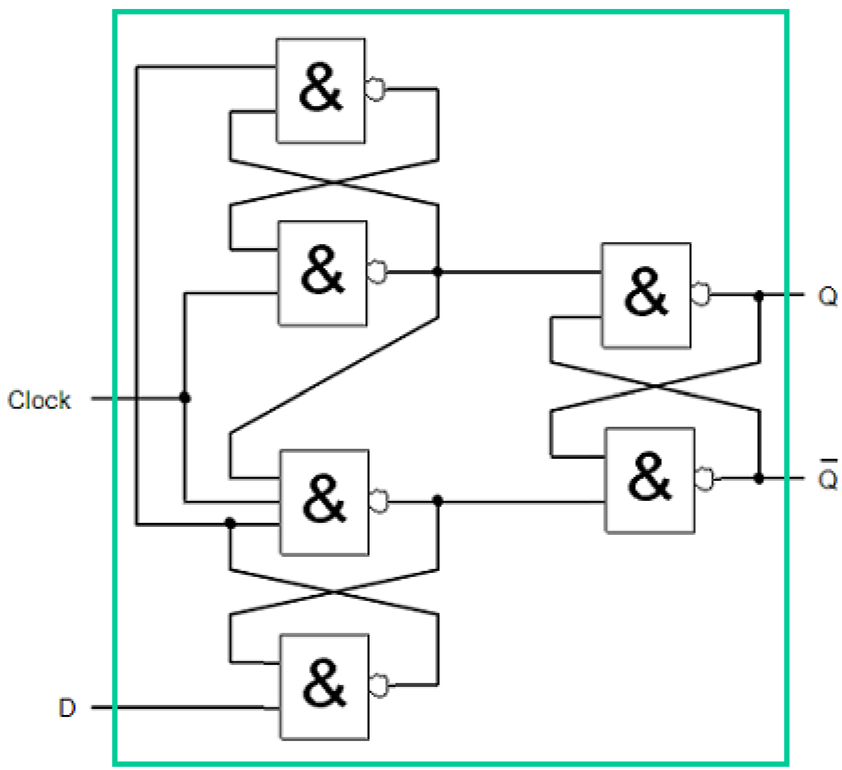
Edgetriggering symbol

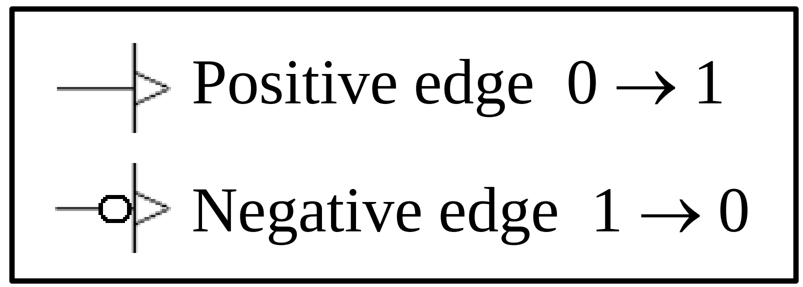


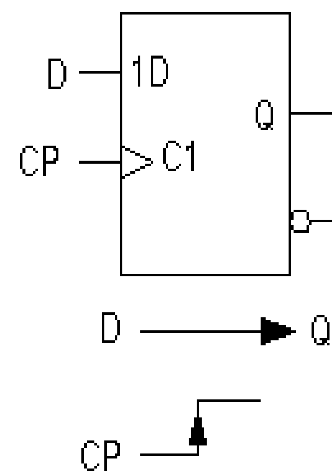
Edgetriggered D-flipflop



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 → 1.





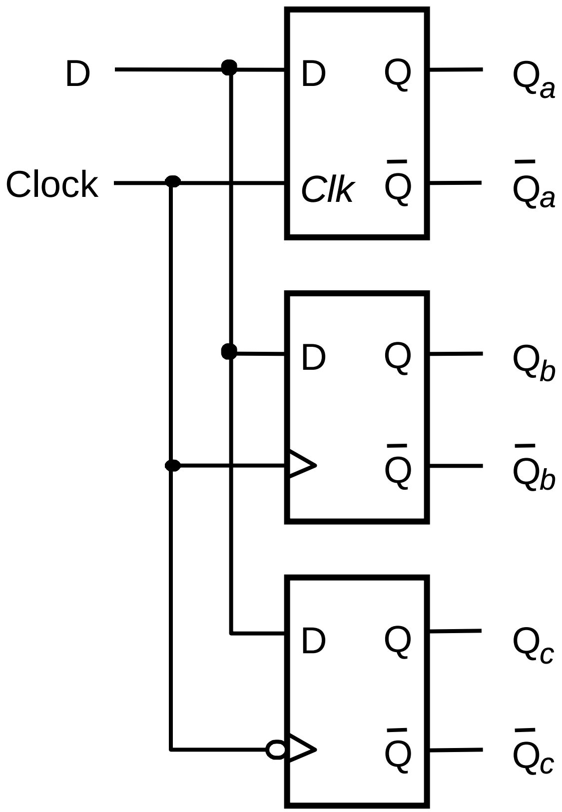


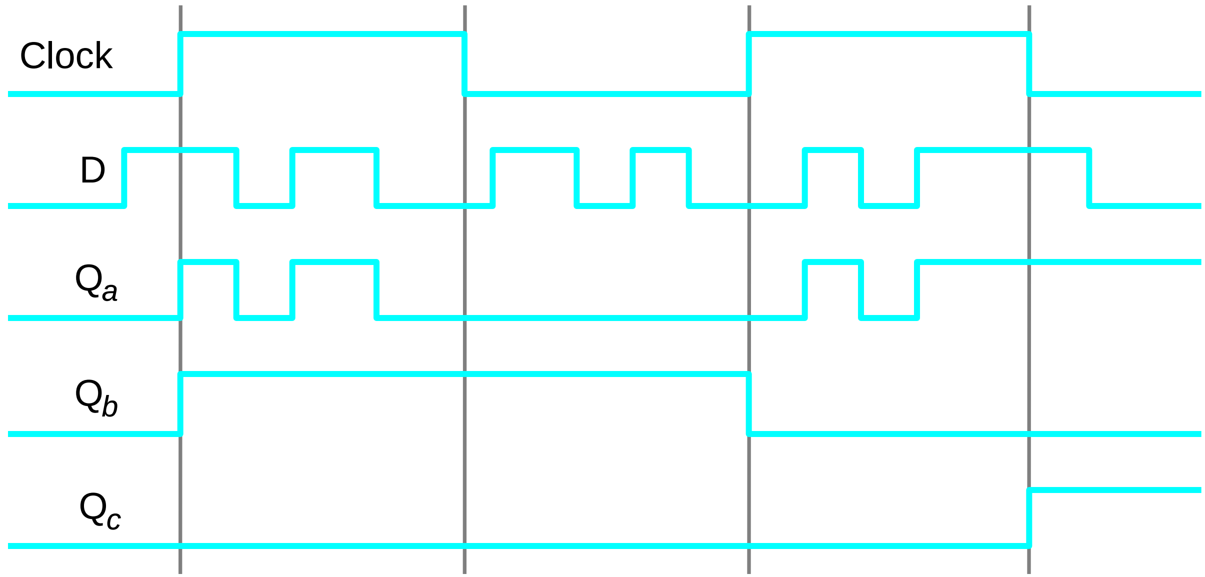
Latch or Flipflop?

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

b) Positive edge triggered flipflop

c) Negative edge triggered flipflop

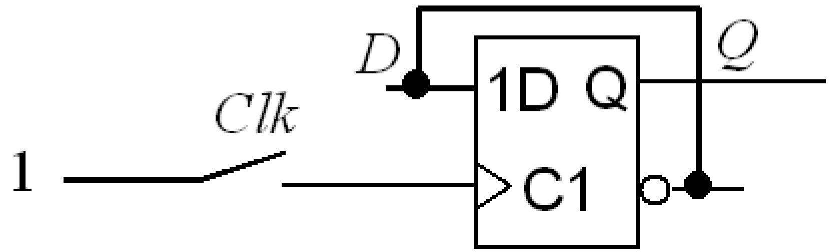


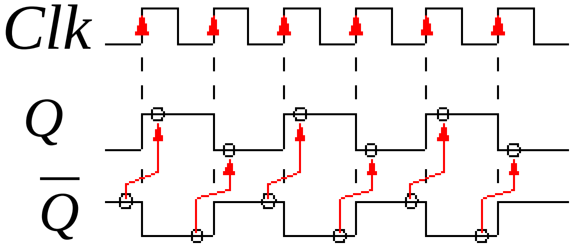


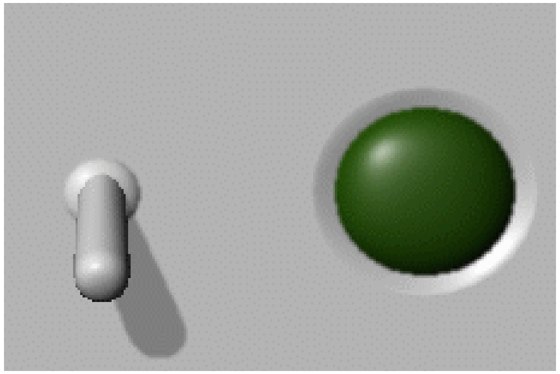
William Sandqvist william@kth.se

Every other time?

$$D = \overline{Q}$$





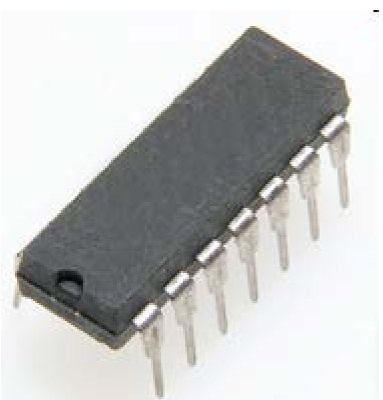


Now the "every other time“ circuit works just as planned!

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

Every second time with Impulse relay On-Off-On-Off …

Impulse relay
Cost: 300:-

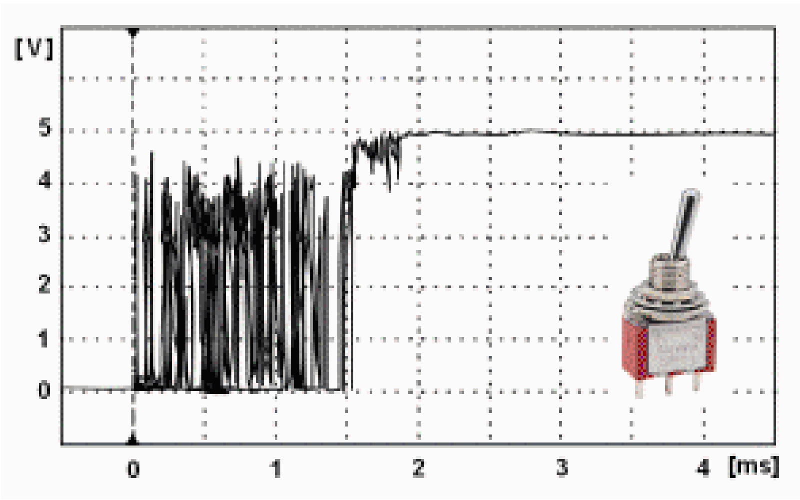


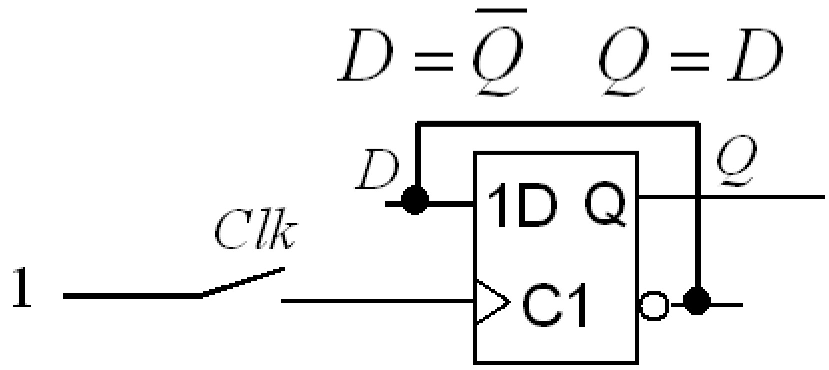
7474 (2st D-flipflop)
Cost: 5:- each

William Sandqvist william@kth.se

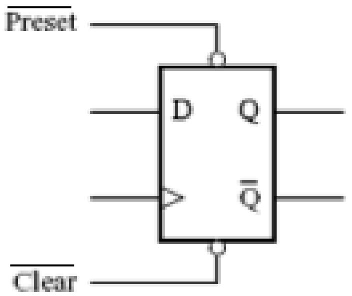
(Contact Bounces)

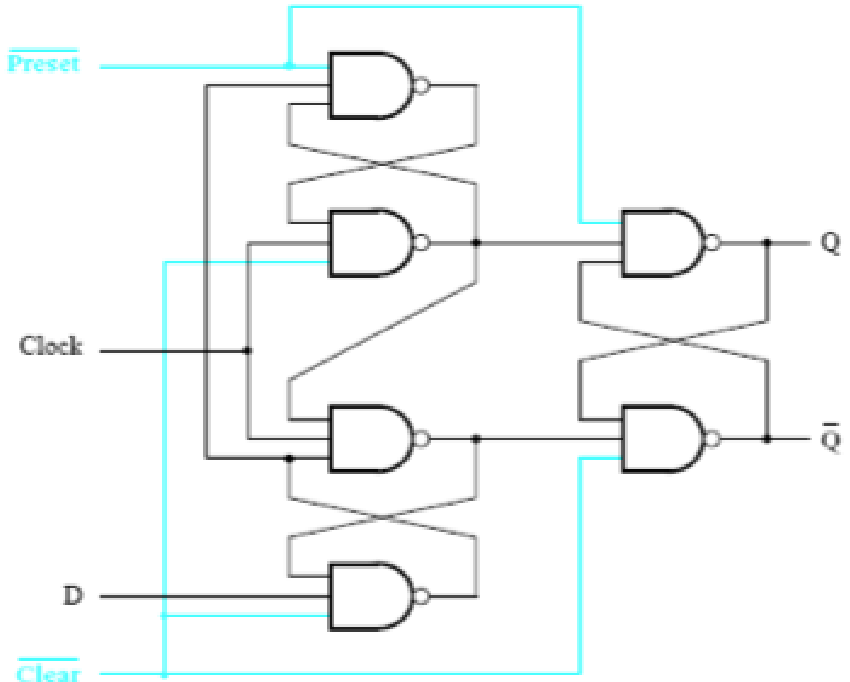
There may be another threat to the "every other time" circuit, and it is that mechanical contacts bounces! You can try at the lab ...





Clear and Preset



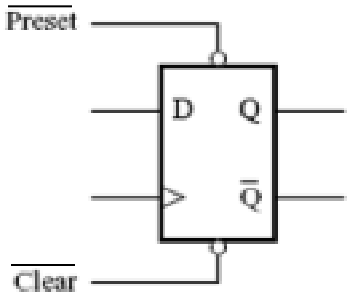


D flip-flop contains three latches. **Preset** and **Clear** signals go directly to the latches and can "lock" these independent of the clock pulse. **Preset** and **Clear** are active low.

Preset = 0 forces Q = 1, while Clear = 0 forces Q = 0.
Preset = Clear = 1 allow the flipflop to perform as intended.

Reset-button

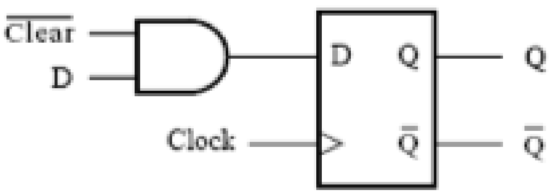
Most digital systems needs to be started in a known state. This may mean that some flip-flops should be "1" while others will be "0". A reset function may need to be connected to either the **Preset** or **Clear** input on the flip-flops.



Preset and **Clear** are asynchronous inputs - the flipflop changes state instantly regardless of the clock pulse.

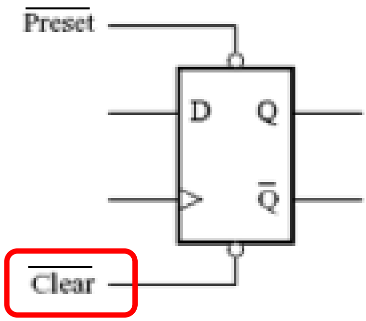
Synchronous Reset

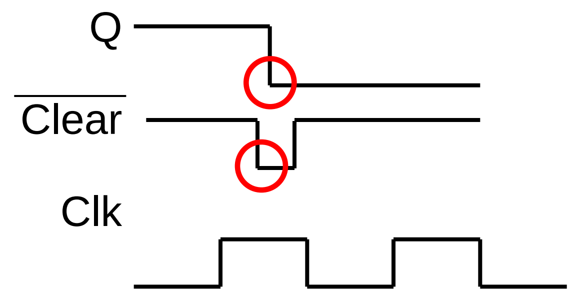
If the flip-flop lacks the Preset and Clear inputs, the reset is implemented with additional logic. Synchronous reset causes the flip-flop to reset to 0 at the next clock edge.



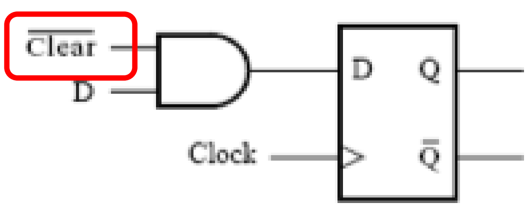
Asynchronous/Synchronous Reset

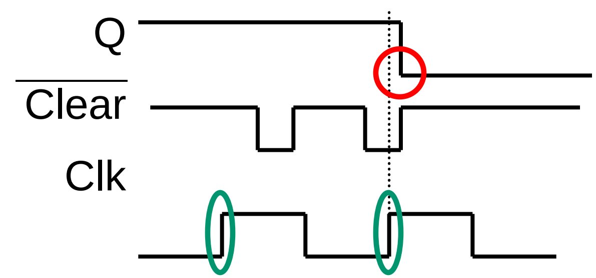
Asynchronous reset





Synchronous reset

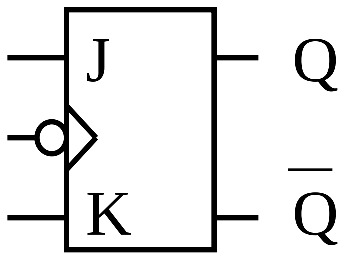




Other common types of flip-flops

JK-flip-flop

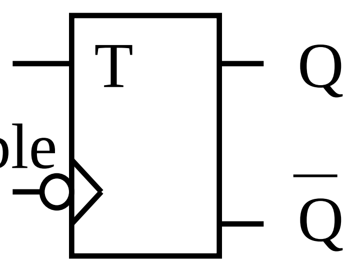
(JK flip-flop is an SR flip-flop with "toggle" instead of the forbidden state)



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)

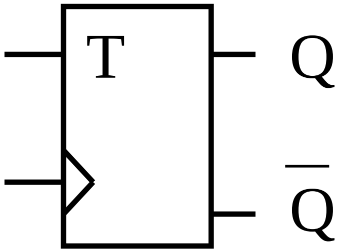
(T-flip-flop is particularly suitable for ”counters”)



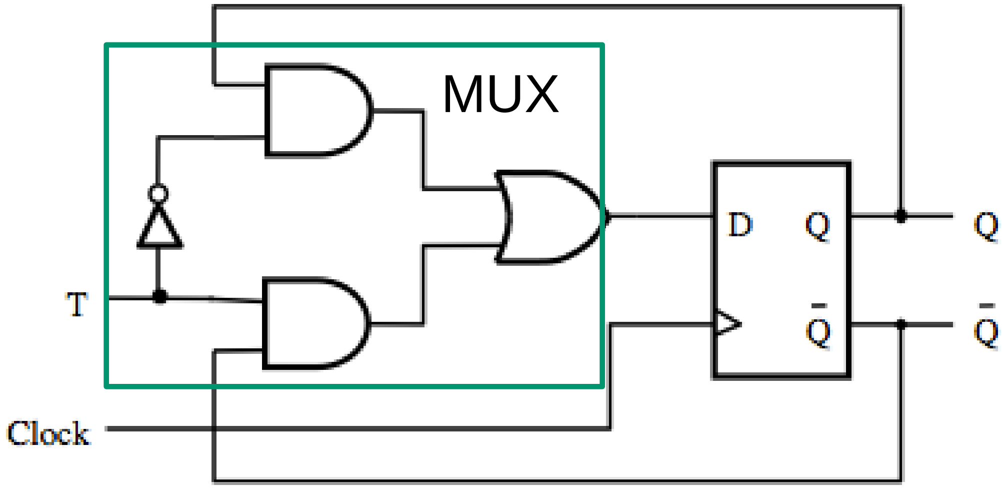
Clk	T	Q	$\overline{Q}$
↓	0	M	M
↓	1	Toggle	Toggle

William Sandqvist william@kth.se

Make a T-flip-flop out of a D-flip-flop



$$D = Q \quad hold$$



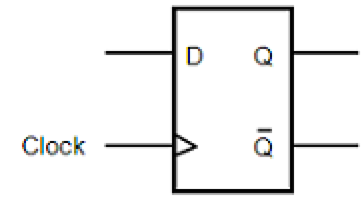
$$D = \overline{Q} \quad toggle$$

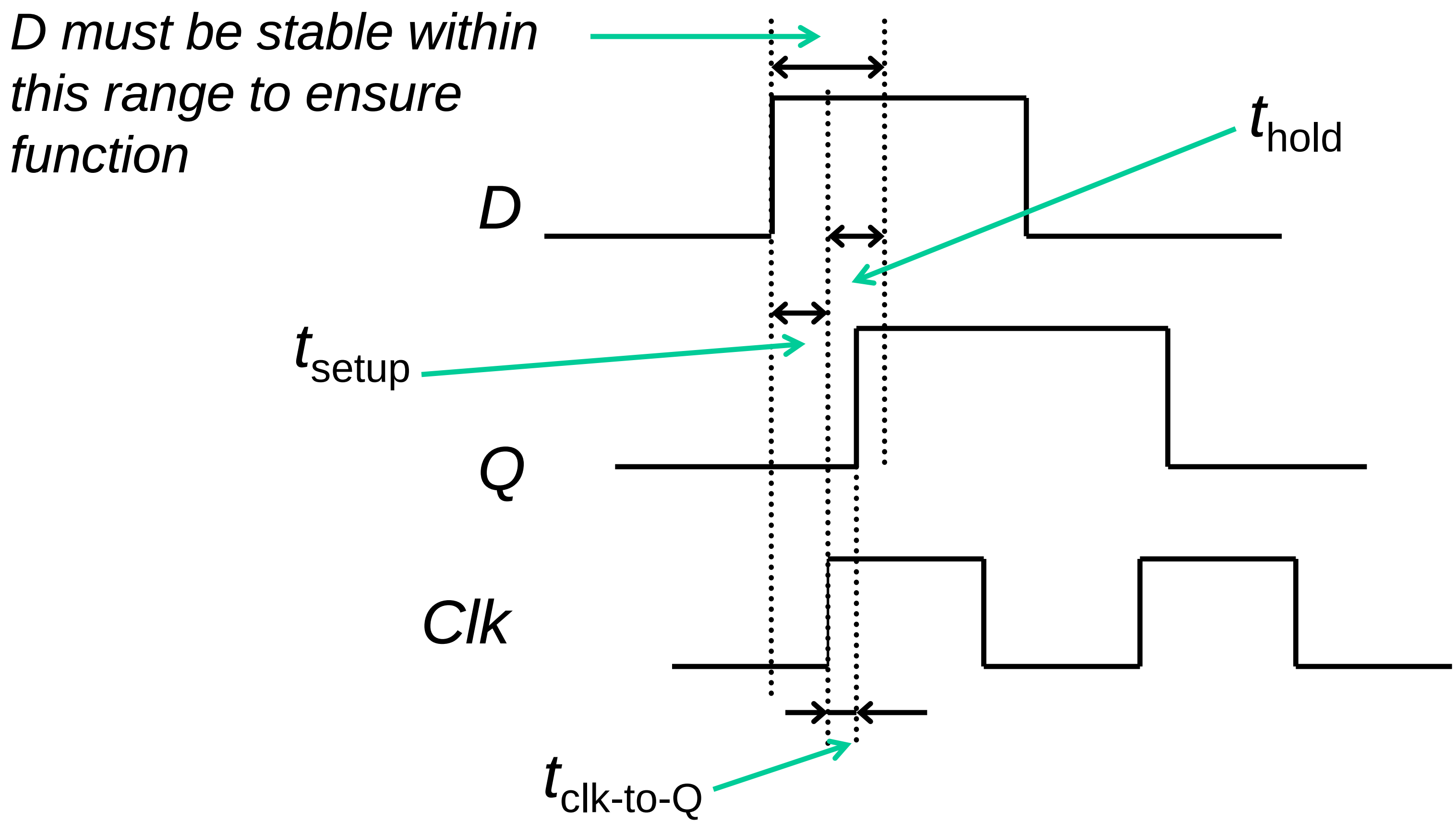
Timing analysis

It is possible to determine the maximum frequency in a sequential circuit by having information about

- Gate delays t_{logic}
- Setup-time t_{su} for the flip-flop
- Hold-time t_{h} for the flip-flop
- Clock-to-output t_{cQ} time

Setup- & Hold-time





William Sandqvist william@kth.se

What is the maximum frequency?

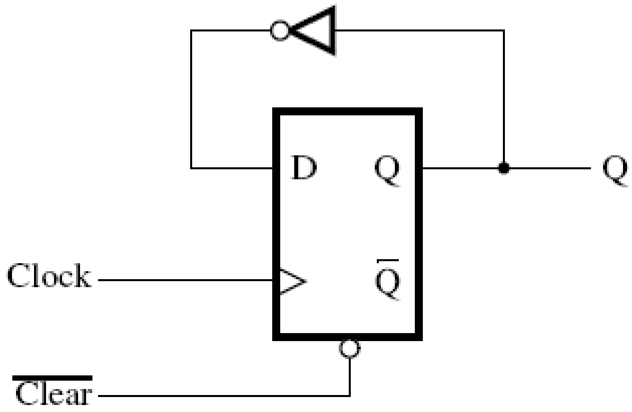
- Gatedelays

  $t_{logic} = t_{NOT} = 1.1$ ns
- Setup-time

  $t_{su} = 0.6$ ns
- Hold-time

  $t_h = 0.4$ ns
- Clock-to-output

  $t_{cQ} = 1.0$ ns

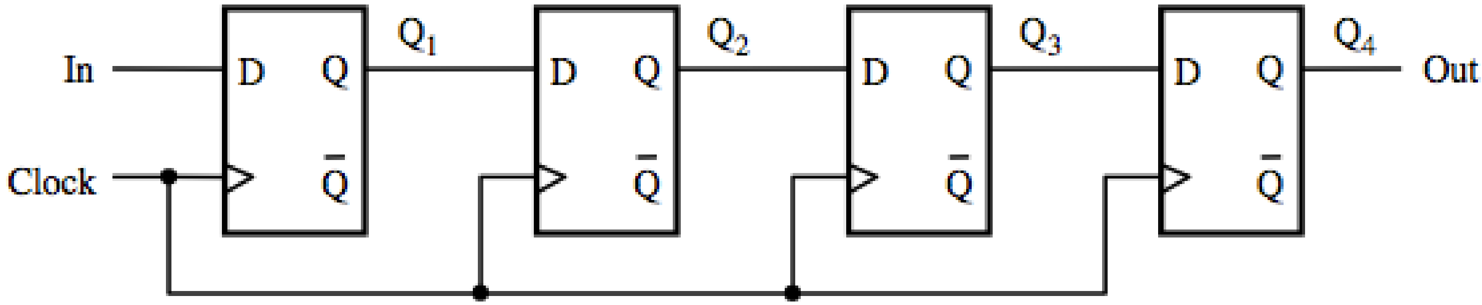


0.6 $\qquad$ $0.4 < 1.0$ $\qquad$ 1.1

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

$f = 1/T = 370$ MHz

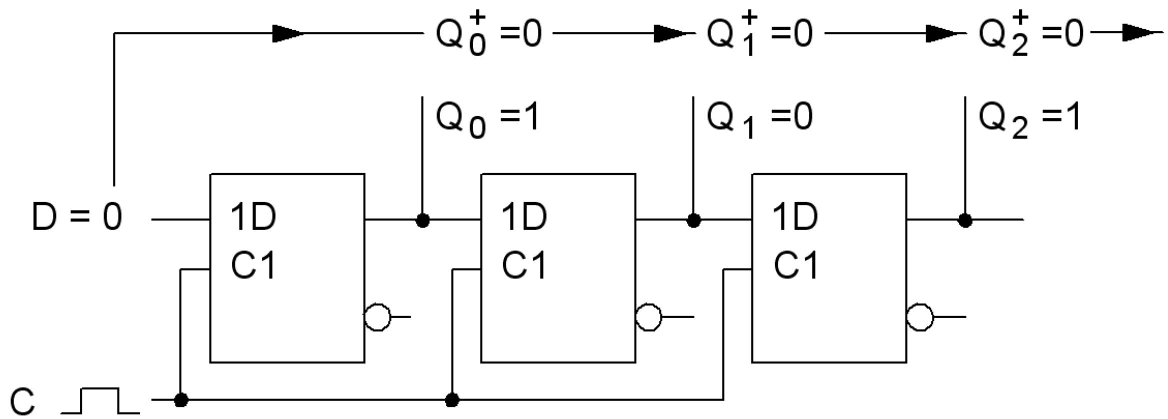
Shiftregister



- A **shiftregister** contains several flip-flops
  For each clock cycle a value will be shifted from left to right
- Many designs use shift registers and the values Q_4, ..., Q_1 as input values to other Components

Would not work with latches …

You can not build a shift register with latches.



When C = 1 ***follow*** the data will “run" through all latches ...

William Sandqvist william@kth.se

Common types of shift registers

- Parallel-In/Parallel-Out (**PIPO**)
- Parallel-In/Serial-Out (**PISO**)
- Serial-In/Parallel-Out (**SIPO**)
- Serial-In/Serial-Out (**SISO**)

- Uses
  - Queues, eg. First-In/First-Out (**FIFO**)
  - Pattern recognizers

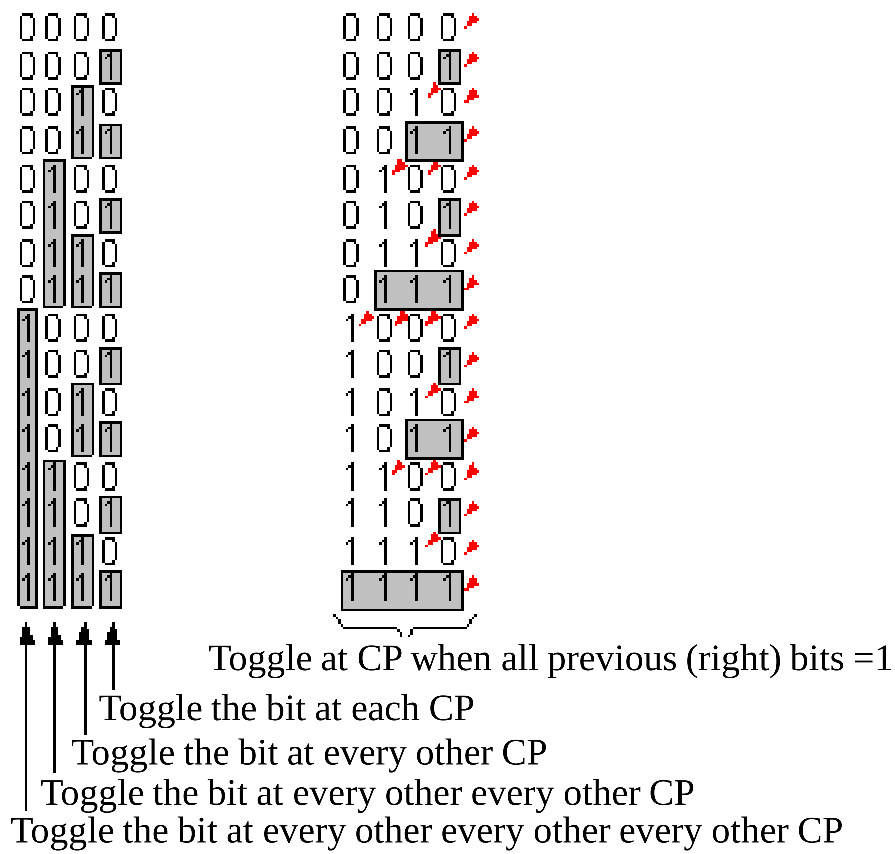
Counters

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 again. The number of states is the counter's module.

The counter does not need to have any inputs except the clock pulses (which then can then be viewed as the input signal). Such sequential circuits are called autonomous.

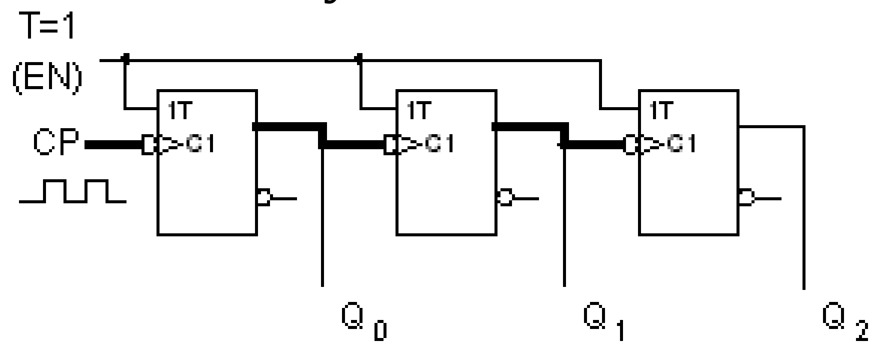
Binary Code counting properties

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



Toggle ”every other” …

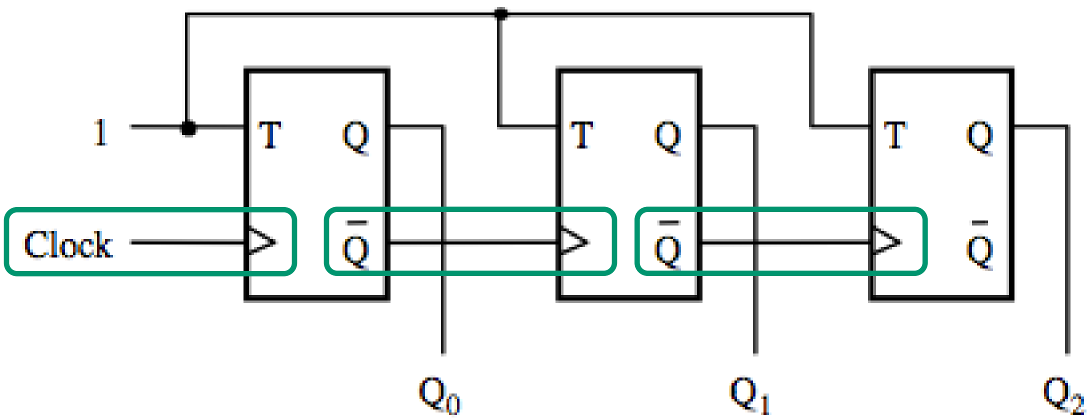
MODULO-8 Asynchronous counter



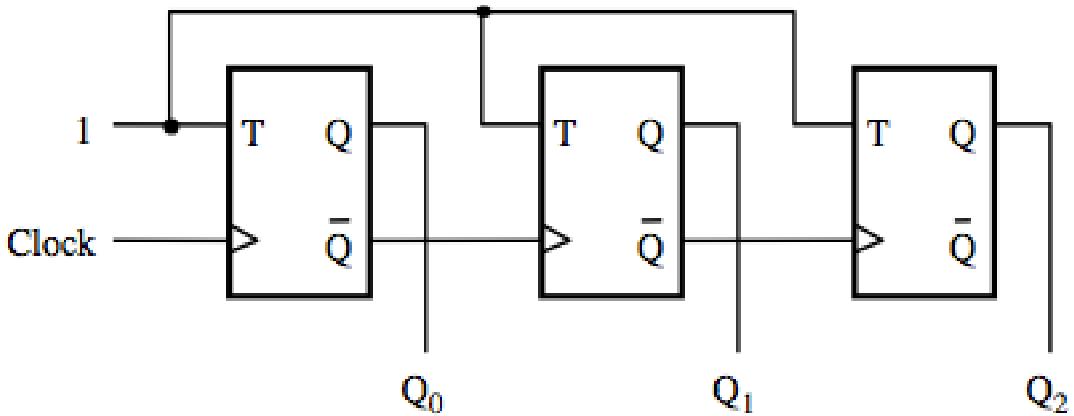
every other, every other every other, every other every other every other

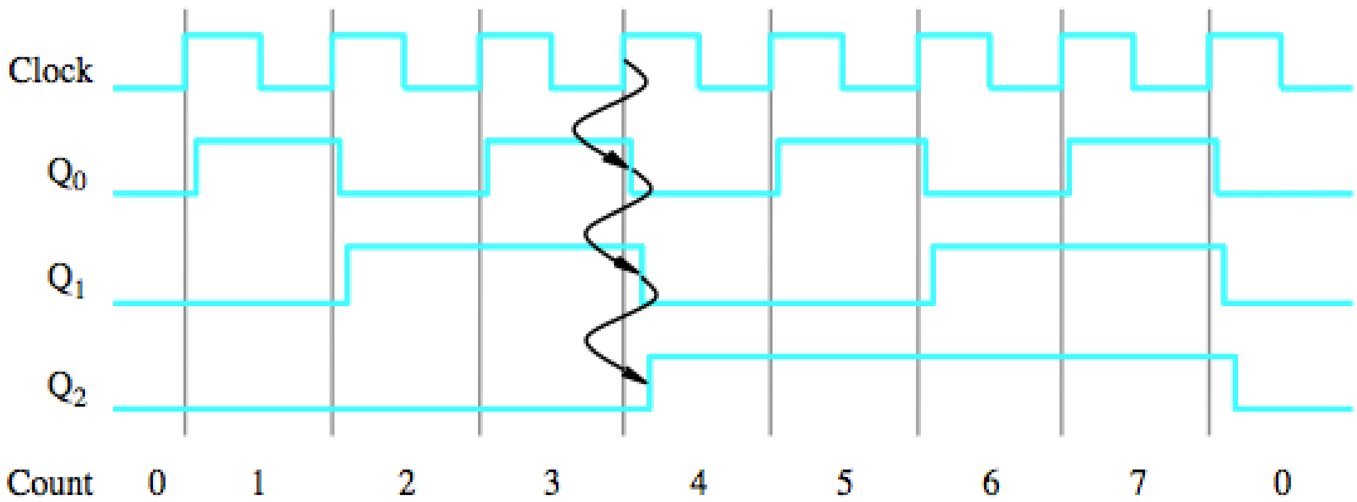
The counter is built of T-flip-flops, they all have T = 1 and "toggles" at clock pulses. The first flip-flop Q_0 "toggles" at each clockpulse. The next flip-flop Q_1 is clocked by the first flip-flop. It will only toggle for each other clockpulse. The third flip-flop Q_2 will toggle for each other each other clockpulse.
According to the binary table, the counter will be counting in binary code. ($Q_2Q_1Q_0$: 000 001 010 011 100 101 110 111 000 ...).

How is this counter counting?

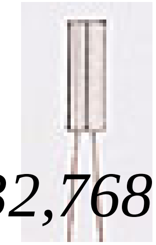


Asynchronous counter



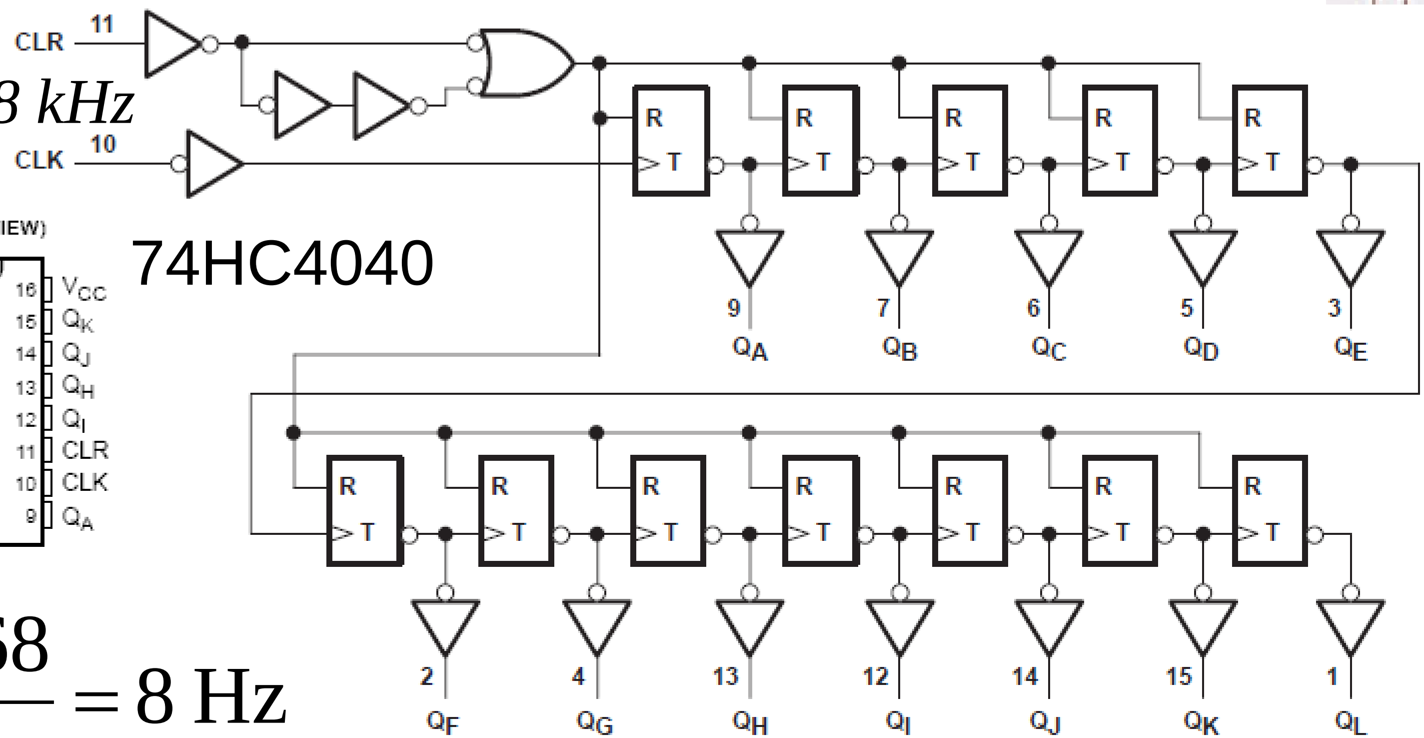


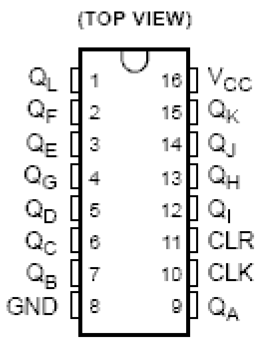
A counter circuit



32,768 kHz

32,768 kHz





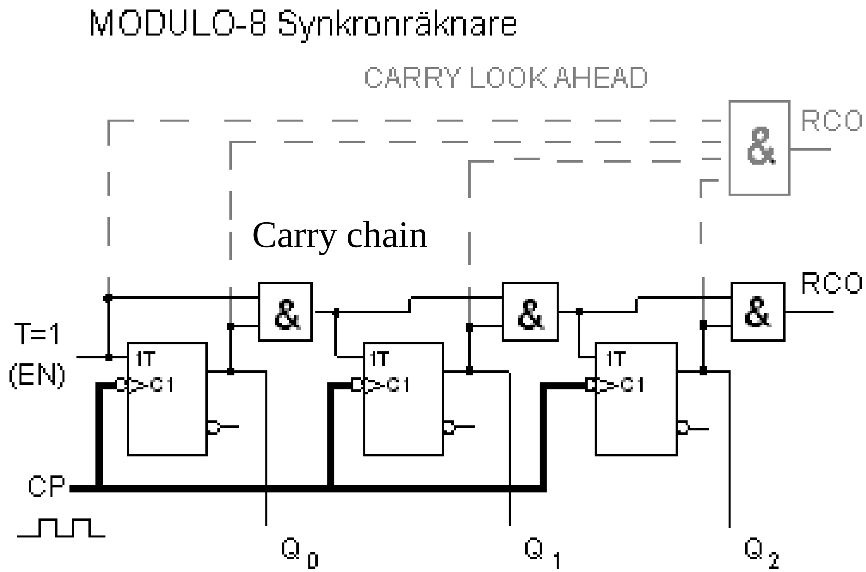
74HC4040

$$\frac{32768}{2^{12}} = 8 \text{ Hz}$$

8 Hz

How to get one second you have to figure out yourself ...

Toggle if all previous are 1…



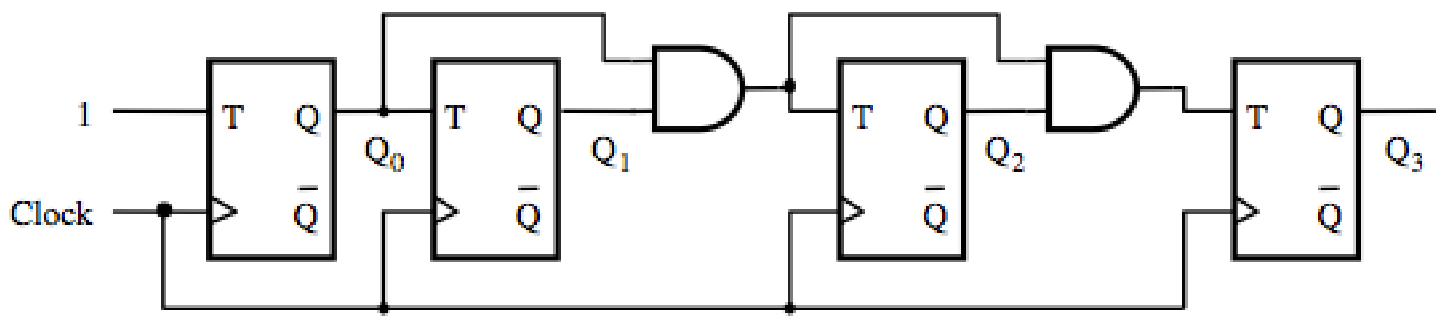
A faster counter can be designed with parallel gates for the carry – carry look ahead.

If you want to expand the counter it is done with a flip-flop and an AND gate per bit.

The clock pulses go directly to all the flip-flops and therefore they change state at the same time. What flip-flop to turn on or not is controlled by the T-inputs. The first flip-flop has T = 1, and it toggles on every clock pulse. The rule is that a flip-flop should toggle if all previous flip-flops stands at "1". This condition is obtained from the AND gates in the so-called Carry chain and it is these gates that control the T-inputs.

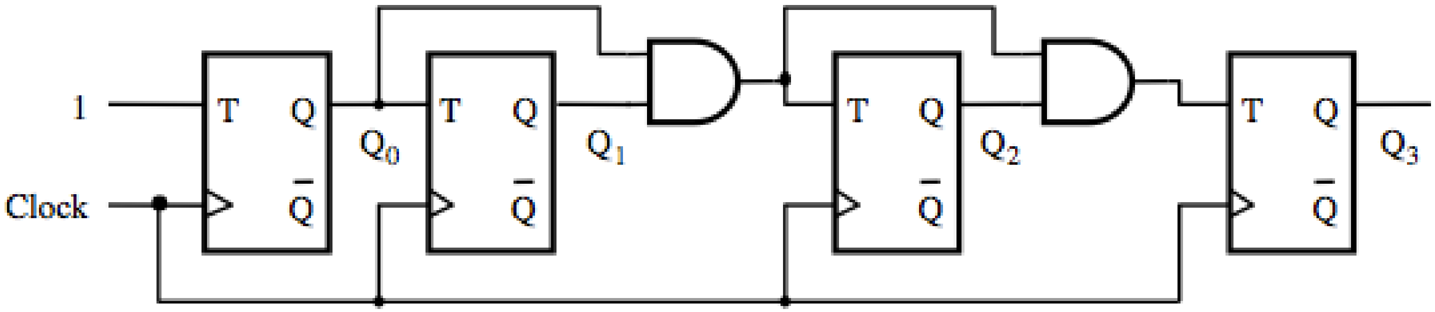
Synchronous counter

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

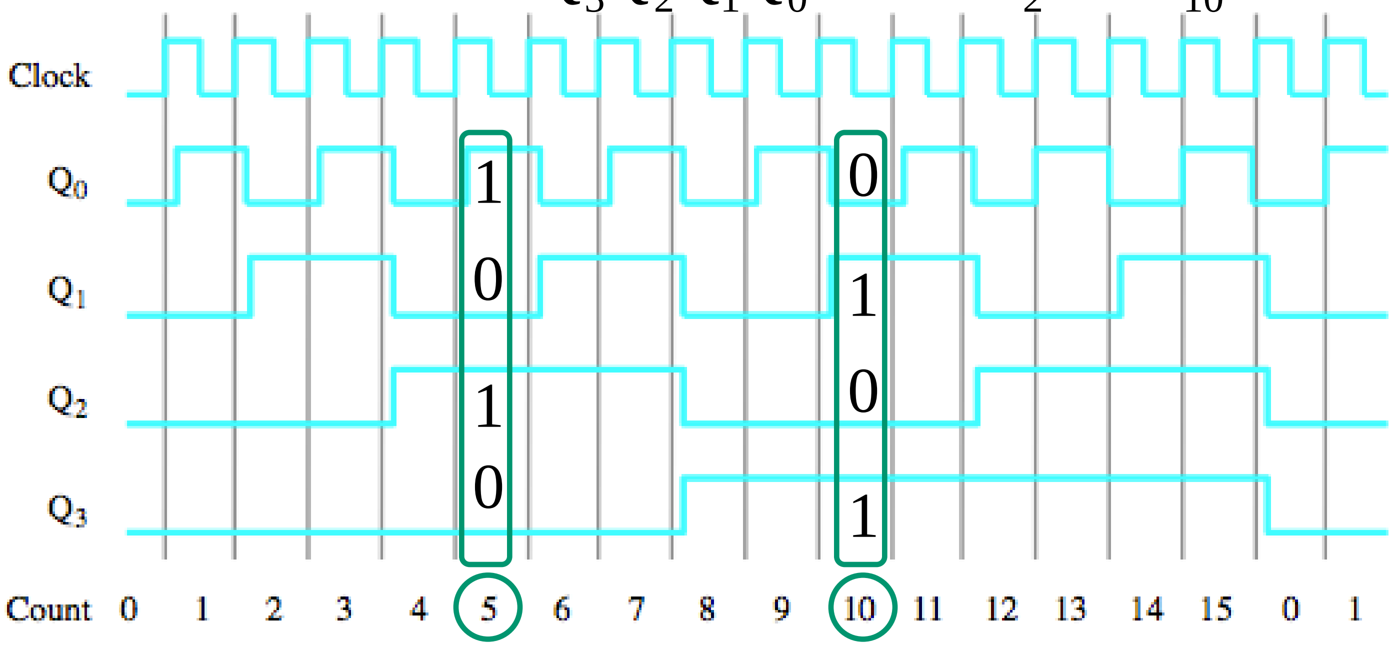


How does this counter count?

Synchronous counter

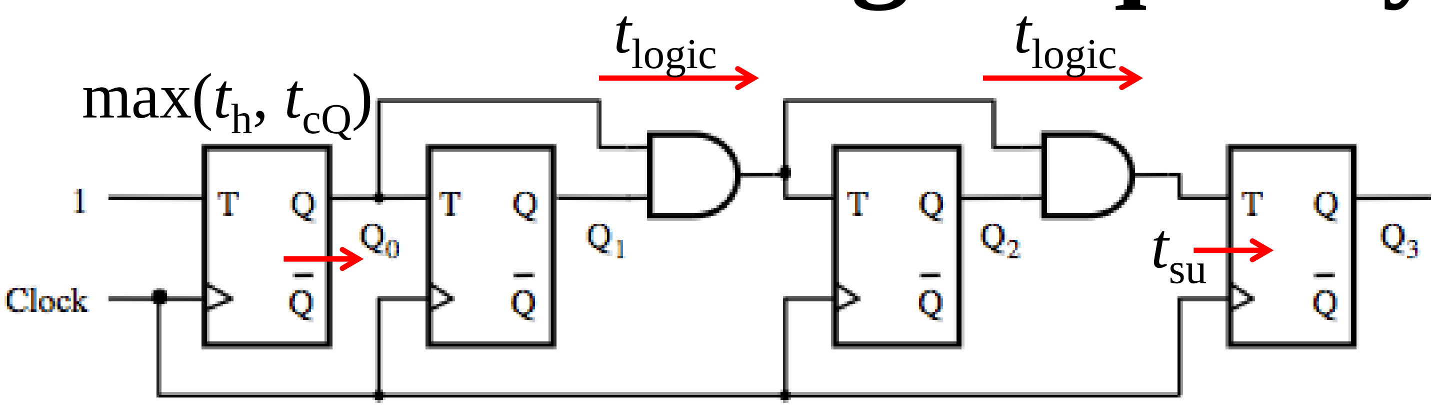


$$Q_3Q_2Q_1Q_0 = 1010_2 = 10_{10}$$



William Sandqvist william@kth.se

Maximum counting frequency?

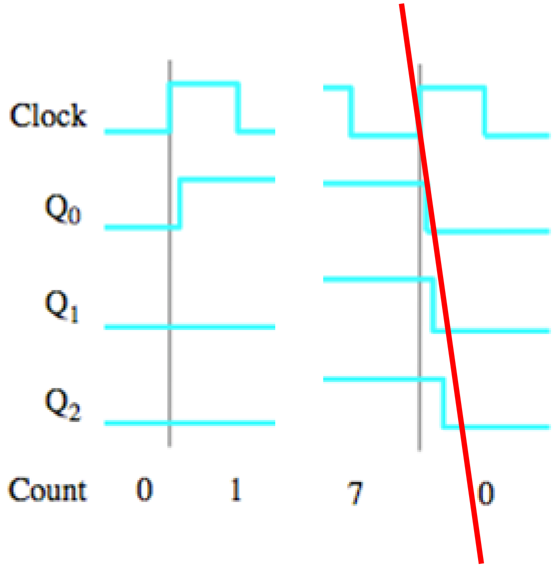


The critical path determines the maximum frequency!
This is the longest combinational path from Q_0 through the two AND gates to the input of flip-flop that calculates Q_3

t_{logic} is thus equivalent to the delay of two AND gates.

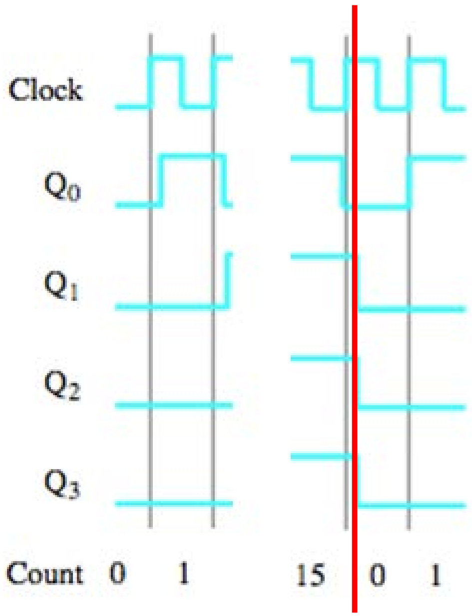
Asynchronous or Synchronous counter

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-flop and latches

Programable logic has embedded flip-flops.

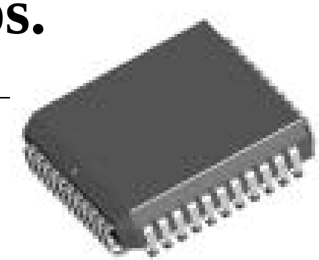
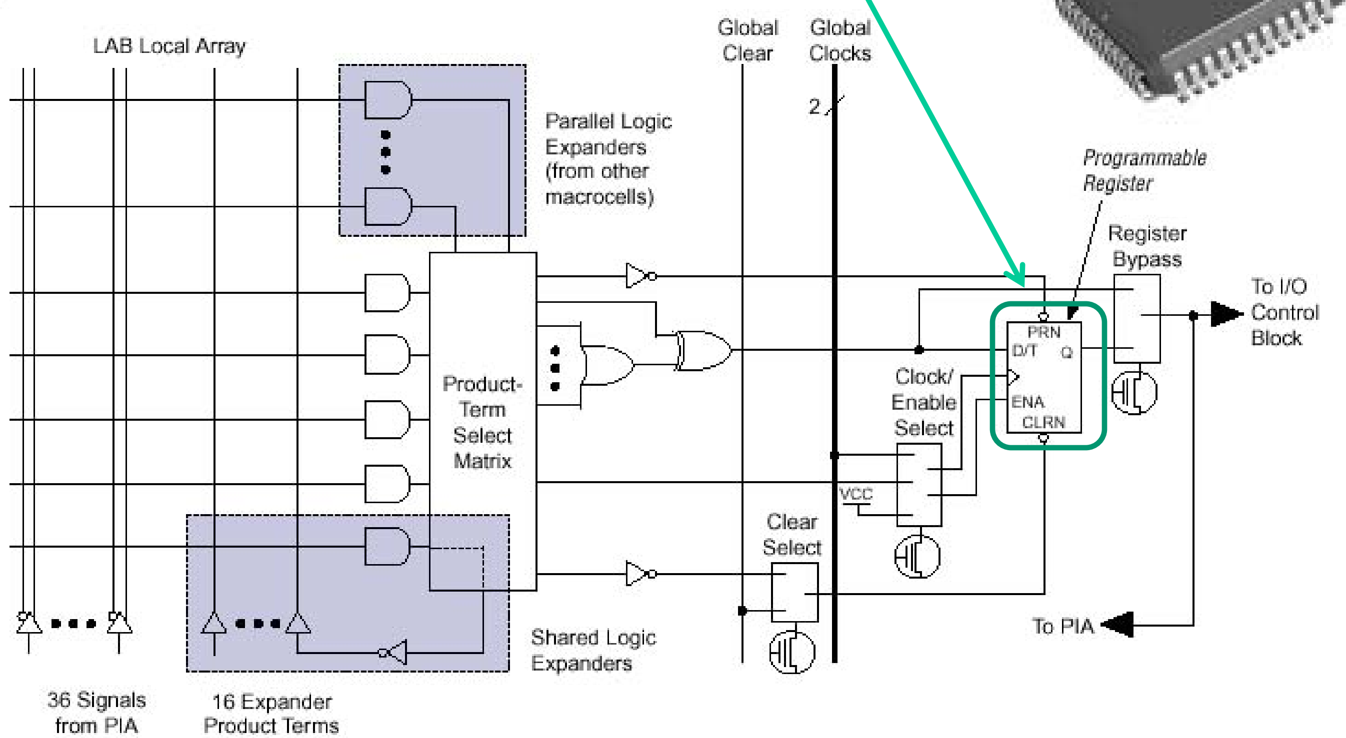


Figure 2. MAX 3000A Macrocell



VHDL for flip-flops and latches

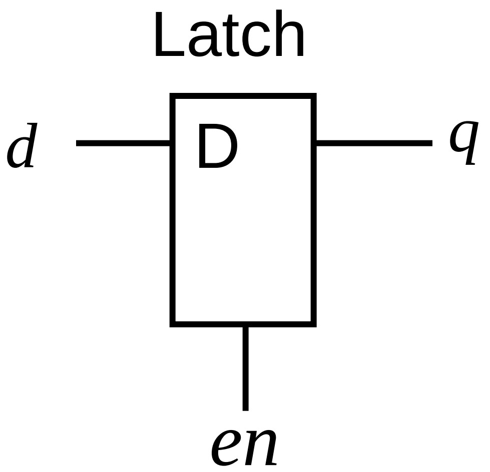
Programmable logic has embedded flip-flops.
How to write VHDL code that "tells" the compiler that you want to use them?

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? → (q <= d;)



Enable	D	Q
0	-	M
1	D	D

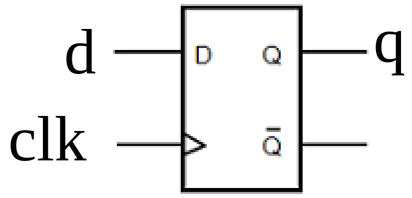
Latch as a process

```
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



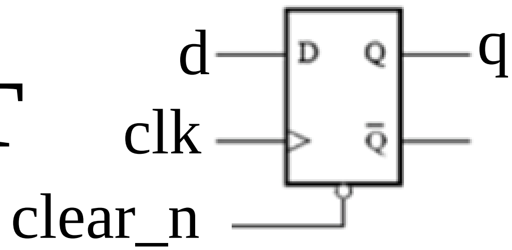

```
PROCESS(clk)
    BEGIN
       IF rising_edge(clk) THEN
          q <= d;
       END IF;
END PROCESS;
```

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.

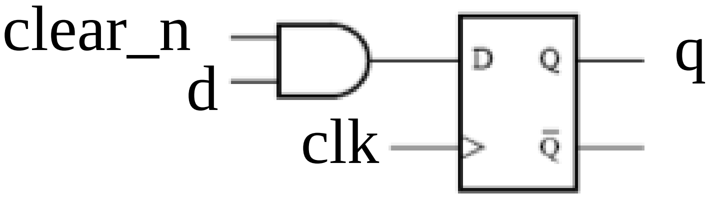
With asynchronous RESET



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 PROCESS;
```

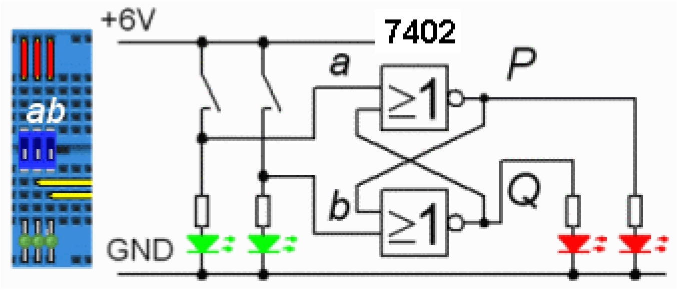
Counters and other sequential circuits

What does this "counter"?

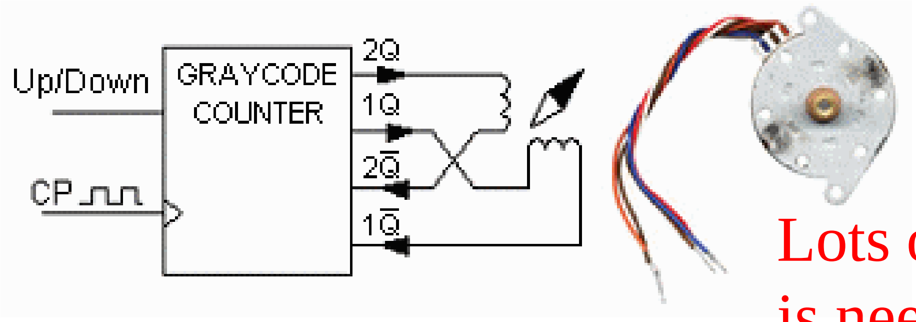
```
bcd:
PROCESS(clk)
    BEGIN
       IF rising_edge(clk) THEN
          IF (count = 9) THEN
             count <= 0;
          ELSE
             count <= count+1;
          END IF;
       END IF;
END PROCESS;
```

LAB Sequence circuits

Latches …

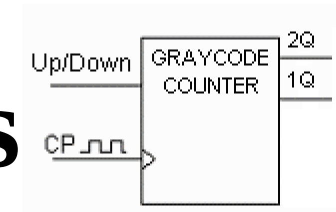


Gray code counter as stepper motor controller …



Lots of preparation time is needed before Lab!

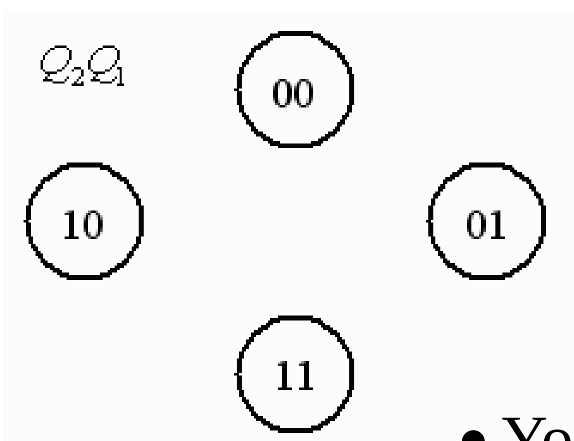
LAB Sequence circuits

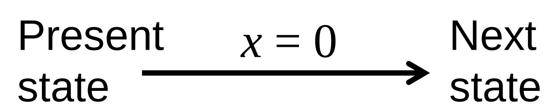


Control signal	Counter mode
x=1	Up: Q_2Q_1= 00, 01, 11, 10, 00, ...
x=0	Down: Q_2Q_1= 00, 10, 11, 01, 00, ...

$$Q_2^+Q_1^+ = f(x, Q_2, Q_1)$$

Next state $Q_2^+Q_1^+$ is a function of present state Q_2Q_1 and the input ***x***

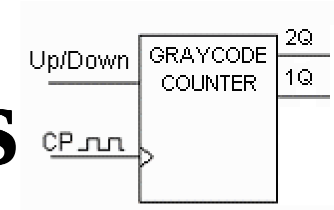




Arrows between states show conditions for transition

- You should draw a state chart

LAB Sequence circuits

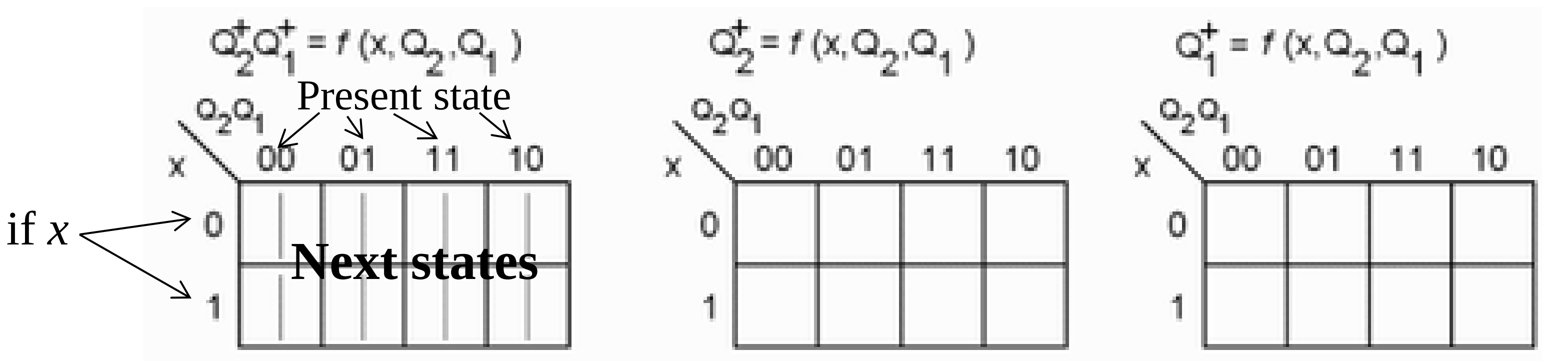


Control signal	Counter mode
x=1	Up: Q_2Q_1= 00, 01, 11, 10, 00, ...
x=0	Down: Q_2Q_1= 00, 10, 11, 01, 00, ...

$$Q_2^+Q_1^+ = f(x, Q_2, Q_1)$$

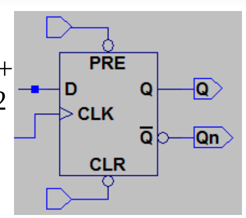
Left part of table

Right part of table

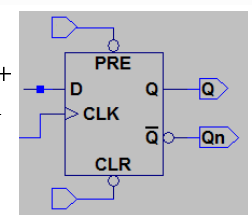


State table
Karnaugh map
style

Q_2^+

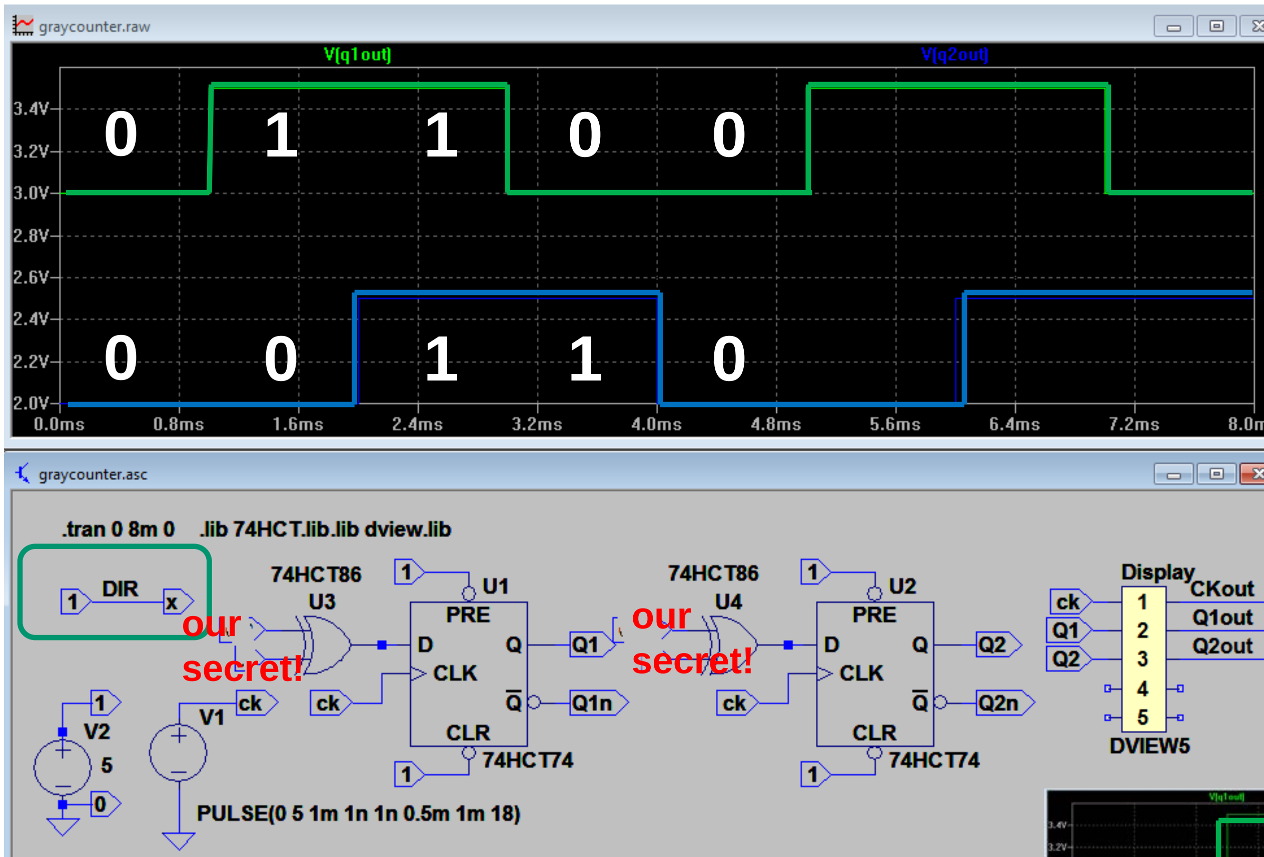


Q_1^+

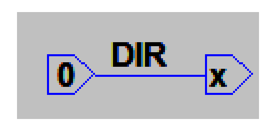


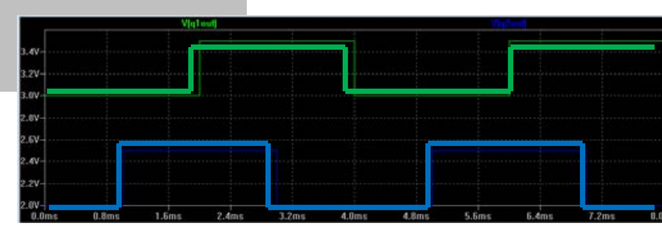
Gives us the logic functions for the flipflops!

Simulate Gray counter



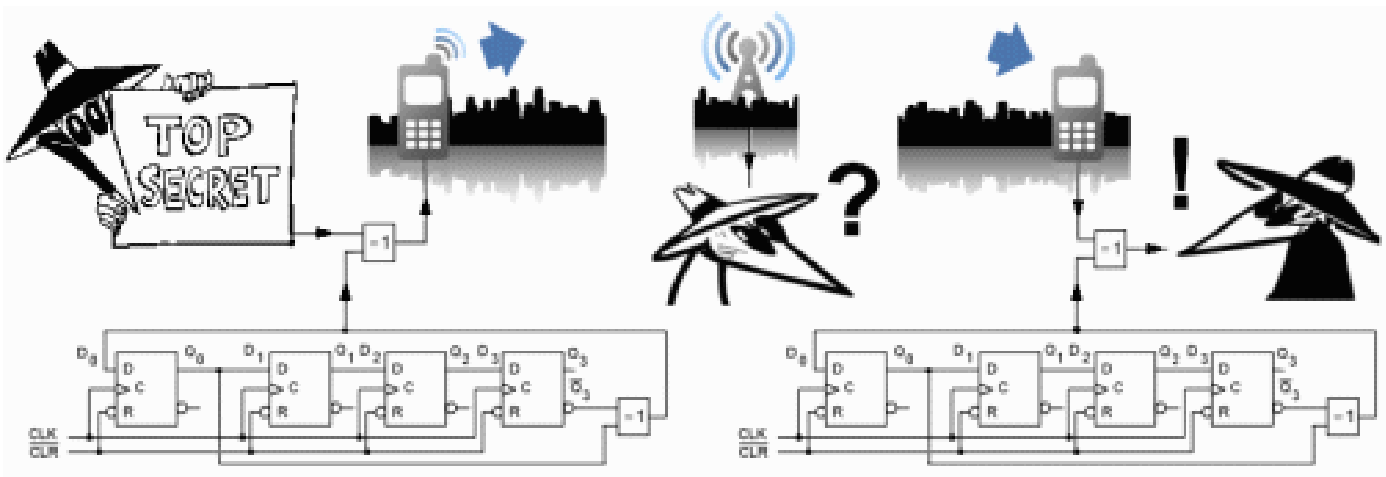
or





LAB Sequence circuits

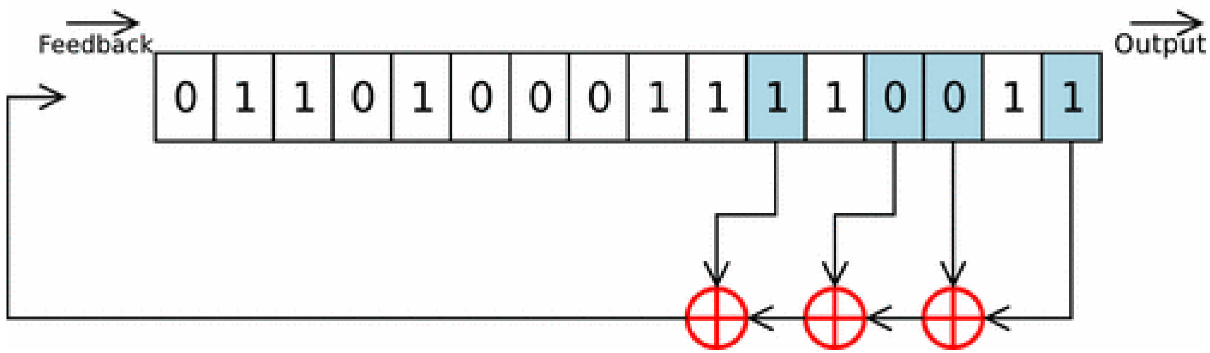
Shiftregister counter generates pseudo random numbers …



PRBS-sequencies (pseudo random numbers) are used to encrypt the data transfer in GSM-phones and for Bluetooth. Another use is to build "self-test-ability" in large digital chip.

William Sandqvist william@kth.se

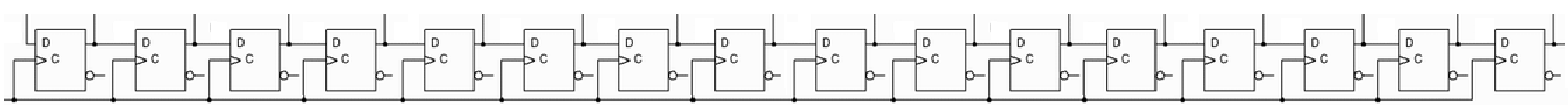
(Pseudo random numbers)



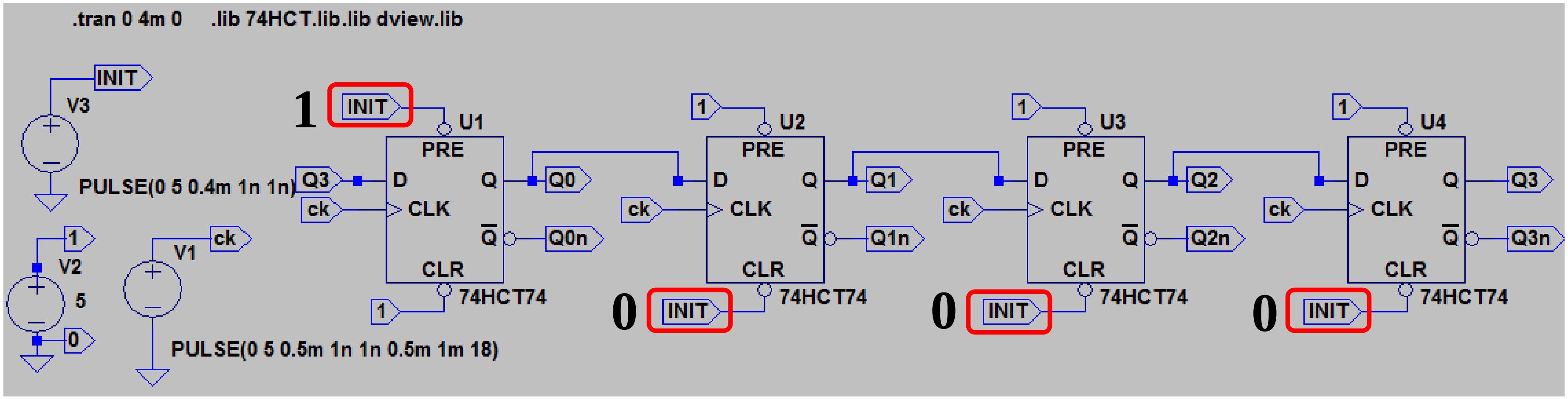
Example feedback shiftregister with 16 flipflops. Flipflop 0,2, 3, and 5 are fed to xor-gates.

This combination will give a *maximal* long sequence that will repete itself only after 65535 clock pulses.

If all flopflops are “0” the sequence will stop, this combination must be avoided!



Simulator flipflops are individually reset/presetable

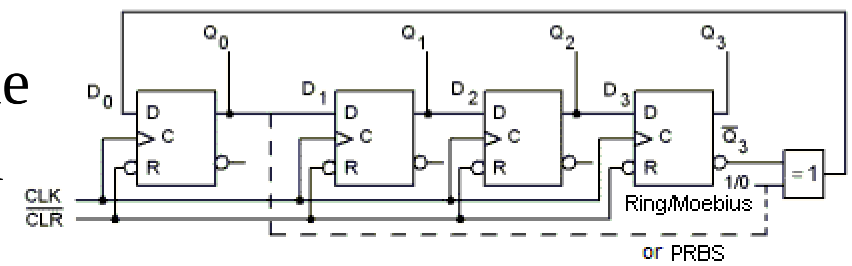


Real world flipflops deeply embedded inside chips are not (they need to be tested in another way)!

- This simulation will start with **1000**.

PRBS to test embedded circuits

All flipflops inside 75175 chip could only be reset at the same time. But no individual reset/preset is possible.



- Use **PRBS** to generate entry points for different ringcounter cycles!