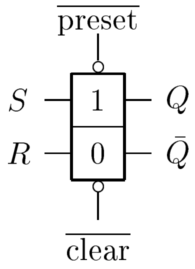
7. Memory elements

A combinatorial circuit consisting of logical functions (gates) without feedback loops, i.e. connections from outputs to previous inputs, cannot remember what happened before: the outputs depend only on the actual values of the inputs. In many cases we want a circuit to remember, for example if we want to make a counter that goes from state 3 to state 4 if we supply an input pulse. In such cases we need memory elements, which are normally called **flip-flops**.

A flip-flop is a *bistable* element: it has only two states called *on* (or 1) and *off* (or 0). Depending on the inputs, the flip-flop will remain in its old state, or it will change: $0 \rightarrow 1$ or $1 \rightarrow 0$. There are no other possibilities, although in many cases additional inputs like *preset* and *clear* are provided in order to initialise some or all flip-flops in a circuit. We will treat the most common flip-flop types below.

7.1 SR flip-flop

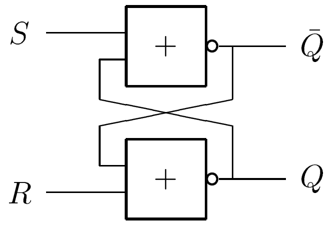
The SR or **Set-Reset** flip-flop has two inputs S and R and two outputs Q and $\bar{Q}$, sometimes also a $\overline{\text{preset}}$ and a $\overline{\text{clear}}$. A $\overline{\text{clear}}$ is a so-called *active-low* input: a 0 implies that the flip-flop goes to state 0, independent of the signal values on the S and R inputs. A 0 on the $\overline{\text{preset}}$ implies that the flip-flop goes to state 1. Below left is the symbol, below right the function table.



S	R	Q_{new}	
0	0	Q_{old}	stable
0	1	0	reset
1	0	1	set
1	1		illegal

In the table we can see that the SR combination 00 does not lead to a change: $Q_{\text{new}} = Q_{\text{old}}$. The combination 01 will always reset the flip-flop (if it was off it remains off, if it was on it will go to off) with the output $Q = 0$ and the output $\bar{Q} = 1$. The combination 10 will always set the flip-flop ($Q = 1$ and $\bar{Q} = 0$). Finally, the combination 11 is illegal. Why this is we will see by means of realisations.

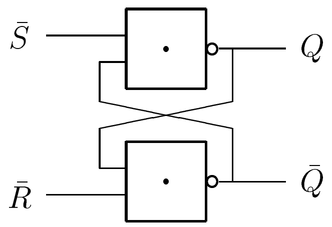
SR flip-flop with NORs



S	R	Q_{new}	$\bar{Q}_{\text{new}}$	
0	0	Q_{old}	$\bar{Q}_{\text{old}}$	stable
0	1	0	1	reset
1	0	1	0	set
1	1	0	0	illegal

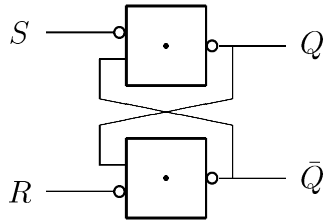
Now we can see what happens with the SR combination 11: each NOR will have a 0 output and therefore $\bar{Q}$ will not be the inverse of Q. You should familiarise yourself with this circuit, assuming values for Q and $\bar{Q}$ as well as S and R, following the connections.

SR flip-flop with NANDs



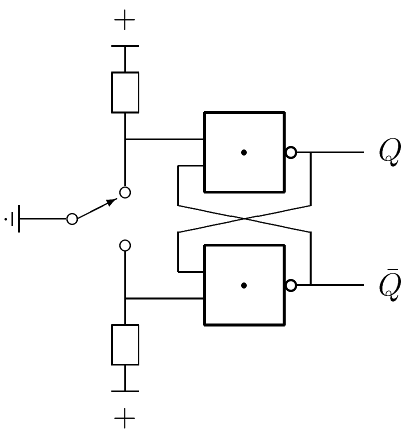
$\bar{S}$	$\bar{R}$	Q_{new}	$\bar{Q}_{\text{new}}$	
0	0	1	1	illegal
0	1	1	0	set
1	0	0	1	reset
1	1	Q_{old}	$\bar{Q}_{\text{old}}$	stable

Because a NAND always follows an input with level 0, we have to assume the *active-low* inputs $\bar{S}$ and $\bar{R}$. You should compare this circuit carefully with the NOR circuit! We can obtain the same functionality and table with NANDs if we introduce inverted inputs:



S	R	Q_{new}	$\bar{Q}_{\text{new}}$	
0	0	Q_{old}	$\bar{Q}_{\text{old}}$	stable
0	1	0	1	reset
1	0	1	0	set
1	1	1	1	illegal

This type of flip-flop is also called a *latch,* and can be used to *debounce* a switch: a switch between 0 and e.g. +5V always shows noise and this can cause problems in digital circuits. A solution is to apply a latch which provides clean Q and $\bar{Q}$ signals:

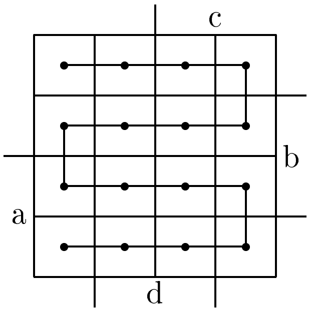


7.2 Application SR flip-flop: Gray code counter

Suppose we have a pulsed signal, p = 0 1 0 1 0 1 etc, periodic or not, and that we want to make a counter which counts 0 1 2 3 ... 15 0 1 2 ... etc. But instead of counting in binary code, we want to count in the Gray or mirror code. Counting from 0 to 15 means 16 steps, so we need 4 flip-flops ($2^4 = 16$). Call the flip-flops A, B, C and D. They have outputs a, b, c, and d, as well as the inverted ones $\bar{a}$ etc. So in total we have 5 signals p, a, b, c, and d, and the question is: what functions are required for the S and R inputs of the flip-flops to count in the Gray code? From now on we will sometimes write FF instead of flip-flop. Let us first recall the Gray code:

step	a	b	c	d	p
0	0	0	0	0	0
1	0	0	0	1	1
2	0	0	1	1	0
3	0	0	1	0	1
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	0
7	0	1	0	0	1
8	1	1	0	0	0
9	1	1	0	1	1
10	1	1	1	1	0
11	1	1	1	0	1
12	1	0	1	0	0
13	1	0	1	1	1
14	1	0	0	1	0
15	1	0	0	0	1

Note: the Gray code is often used in position measurements because only 1 bit changes (when the binary code goes from 0111 to 1000 all 4 bits change). Contrary to the binary code, the Gray code follows a continuous path in the Karnaugh diagram:

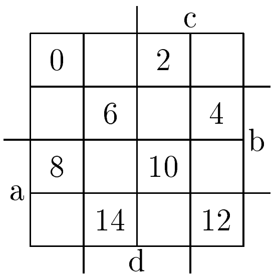


There are many codes which change only 1 bit at a time; just select a continuous path in the Karnaugh diagram!

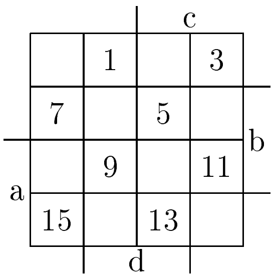
In step (or state) 0 the FF's for state 1 must be set and must remain. We must keep in mind that gates and FF's have a certain delay between inputs and outputs. It takes some time for a circuit to react, so there will be brief unstable states before the stable ones will be reached. This is shown in the table below, which is a copy of the top of the table above. Note that S means stable and U means unstable.

state	a	b	c	d	p	
0	0	0	0	0	0	S
	0	0	0	0	1	U
1	0	0	0	1	1	S
	0	0	0	1	0	U
2	0	0	1	1	0	S
	0	0	1	1	1	U
3	0	0	1	0	1	S
	0	0	1	0	0	U
4	0	1	1	0	0	S

We see that each abcd code has a stable and an unstable state; the last is caused by a transition in p, but the circuit did not have the time to react and to go directly to the stable new state. We can now make diagrams which show the stable and unstable states for p=0 and p=1. First the stable states:



p=0



p=1

and now the same for the unstable states:

			c	c	
		2		4	
	8		6		b
a		10		12	b
a	0		14		
		d	d		

p=0

			c	c	
	1		3		
		7		5	b
a	9		11		b
a		15		13	
		d	d		

p=1

We can now combine the stable and unstable states for p=0 (left diagram below) and then fill the value that a FF should have in the table of the Gray code, for example FF D (right diagram):

			c	c	
	0	2	2	4	
	8	6	6	4	b
a	8	10	10	12	b
a	0	14	14	12	
		d	d		

p=0

			c	c	
	0	1	1	0	
	0	1	1	0	b
a	0	1	1	0	b
a	0	1	1	0	
		d	d		

FF D for p=0

In the same way can we make the diagrams for p=1 and FF D:

			c	c	
	1	1	3	3	
	7	7	5	5	b
a	9	9	11	11	b
a	15	15	13	13	
		d	d		

p=1

			c	c	
	1	1	0	0	
	0	0	1	1	b
a	1	1	0	0	b
a	0	0	1	1	
		d	d		

FF D for p=1

On the basis of the left diagrams (**p=0** and **p=1**) we can also fill the diagrams for the other FF's, for example FF C:

			c	c	
	0	1	1	1	
	0	0	0	1	b
a	0	1	1	1	b
a	0	0	0	1	
		d	d		

			c	c	
	0	0	1	1	
	0	0	1	1	b
a	0	0	1	1	b
a	0	0	1	1	
		d	d		

FF C for p=0 **FF C for p=1**

Until now we have seen how to make the Karnaugh diagrams for p=0 and 1 for the FF's. But now we should take look at how to generate the set and reset functions for the FF's. Let's take the diagram for p=0 and FF D. The top part is shown below left. Now we are going to compare (see the arrows) the values of D according to the position in the diagram (on top) and the actual values in the diagram. This comparison allows us to make a new diagram, one for generating the reset for FF D and p=0 which is shown below the diagram with the arrows:

0	1	1	0
0	1	1	0
0	1	1	0

	d		
×	0	0	×
×	0	0	×

For reset D and p=0:

- If d=0 (on top) and d should be 0 (in diagram), the circuit may generate a reset, but it is not required because d is already 0. Conclusion: we can fill a don't care.
- If d=1 (on top) and d should be 1 (in diagram), the circuit may certainly NOT generate a reset. We need to fill a 0.

In the same way we can make a Karnaugh diagram for reset D and p=1:

0	1	1	0
1	1	0	0
0	0	1	1

	d		
0	0	1	×
×	1	0	0

For reset D and p=1:

- If d=0 and should be 1: not a reset.
- If d=1 and should be 1: not a reset.
- If d=1 and should be 0: a reset!
- If d=0 and should be 0: a dont't care.

As a result, we obtain the following two Karnaugh diagrams:

			c		
	×	0	0	×	
	×	0	0	×	b
a	×	0	0	×	b
a	×	0	0	×	
		d	d		

Reset D for p=0

			c		
	0	0	1	×	
	×	1	0	0	b
a	0	0	1	×	b
a	×	1	0	0	
		d	d		

Reset D for p=1

We see that the reset FF D for p=0 is zero because we can replace all don't cares by 0's. The reset FF D for p=1 equals $(\bar{b}c\bar{a} + b\bar{c}\bar{a} + bca + \bar{b}\bar{c}a) \cdot p = (a \oplus b \oplus c) \cdot p$.

We can do the same for the set of FF D. We then get the following two Karnaugh diagrams:

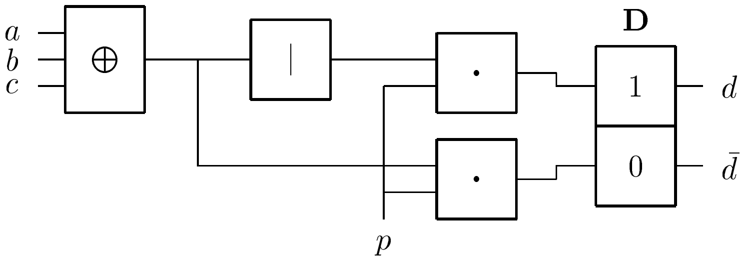
			c	c	
	0	×	×	0	
	0	×	×	0	b
a	0	×	×	0	b
a	0	×	×	0	
		d	d		

Set D for p=0

			c	c	
	1	×	0	0	
	0	0	×	1	b
a	1	×	0	0	b
a	0	0	×	1	
		d	d		

Set D for p=1

Again, the set D for p=0 is zero because we can replace all don't cares by 0's. The set D for p=1 equals $(\bar{b}\bar{a}\bar{c} + b\bar{a}c + ba\bar{c} + \bar{b}ac) \cdot p = (a \oplus b \oplus c)' \cdot p$. Summarising, we see that the output of FF D does not play any role in it's set and reset inputs, that there is only a change when p=1 and that $D_{\text{reset}} = (a \oplus b \oplus c) \cdot p$ whereas $D_{\text{set}} = (a \oplus b \oplus c)' \cdot p$. We get the following circuit for FF D:



Now that we are ready with FF D, we can do the same for FF C. We then get the following four Karnaugh diagrams:

			c	c	
	×	0	0	0	
	×	×	1	0	b
a	×	0	0	0	b
a	×	×	1	0	
		d	d		

C$_{\text{reset}}$; p=0

			c	c	
	×	×	0	0	
	×	×	0	0	b
a	×	×	0	0	b
a	×	×	0	0	
		d	d		

C$_{\text{reset}}$; p=1

			c	c	
	0	1	×	×	
	0	0	0	×	b
a	0	1	×	×	b
a	0	0	0	×	
		d	d		

C$_{\text{set}}$; p=0

			c	c	
	0	0	×	×	
	0	0	×	×	b
a	0	0	×	×	b
a	0	0	×	×	
		d	d		

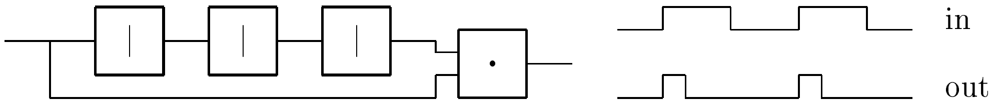
C$_{\text{set}}$; p=1

This results in the simplyfied functions $C_{\text{reset}} = (b\bar{a}d + a\bar{b}d) \cdot \bar{p} = d\bar{p}(a \oplus b)$ and $C_{\text{set}} = (\bar{b}d\bar{a} + bda) \cdot \bar{p} = d\bar{p}(a \oplus b)'$.

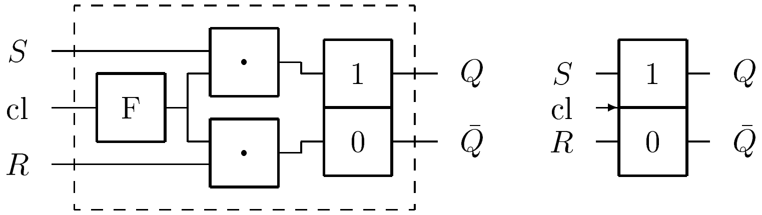
If you have verified the above examples, you can now do the analysis for flip-flops B and A. You should get $B_{\text{set}} = \bar{p}\bar{a}c\bar{d}$, $B_{\text{reset}} = \bar{p}ac\bar{d}$, $A_{\text{set}} = \bar{p}\bar{b}\bar{c}\bar{d}$ and $A_{\text{reset}} = \bar{p}\bar{b}\bar{c}\bar{d}$. You should also be able to draw the circuits.

7.3 Clocked SR flip-flop

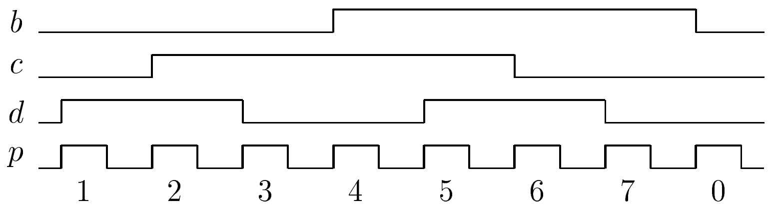
A circuit with Set-Reset flip-flops generally requires more components than are strictly required and the design of a counter is straightforward but takes a lot of work. In realising a counter, we may not be interested in the **levels** of the input signal, but only in the **positive flanks**. So we could combine a SR flip-flop with some circuit that generates a very brief pulse on the positive flank, having a duration that is just enough for the flip-flop to react. One simple circuit is:



The three invertors act as just a single one, but the total delay is three times that of a single invertor. This, together with the AND gate, will give a positive pulse at a positive flank of the input signal. If we connect this circuit F to a SR flip-flop (below left) we obtain a **clocked SR flip-flop**, the symbol of which is shown to the right:



The "cl" input is called the **clock input**. In order to illustrate the use of a clocked SR FF, we will design the Gray code counter but only for counting from 0 to 7 (8 pulses). Hence, we get the following timing diagram and count sequence in the Karnaugh diagram:



		d	d	
0	1	2	3	
7	6	5	4	b
		c	c	

The circuit needs to react on positive flanks, so we don't have to make different tables for p=0 and p=1. Following the count sequence in the karnaugh diagram (above right) and looking at the values of the flip-flops, we can directly fill Karnaugh diagrams for the set and reset inputs, for example for FF D:

0→1→1→0

		d	d	
1	×	0	0	
0	0	×	1	b
		c	c	

D set

0→1→1→0

		d	d	
0	0	1	×	
×	1	0	0	b
		c	c	

D reset

Explanation: the count sequence goes 01234567 012... and the value that FF D should have is 01100110 011... Hence, in the first position (0) D is 0. In the second position D must be 1, so we need to generate a set in the first position (set = 1, reset = 0). In the second position (1) D is 1 and in the third position (2) as well: so the circuit could generate a set but not necessarily. We can enter in the second position set = don't care, reset = 0. In the third position (2) D=1, but in the fourth (3) D=0: we need to generate a reset (set = 0, reset = 1). Doing this for the entire count cycle, we obtain the above two diagrams from which we get the functions

$$D_{\text{set}} = (\bar{c}\bar{b} + cb)p = (b \oplus c)'p$$

and

$$D_{\text{reset}} = (c\bar{b} + \bar{c}b)p = (b \oplus c)p$$

Exactly the same way, but looking at the values that the other flip-flops should have, we obtain for FF C

	d	d		
0	1	×	×	
0	0	0	×	b
		c	c	

$$C_{\text{set}} = d\bar{b}p$$

	d	d		
×	0	0	0	
×	×	1	0	b
		c	c	

$$C_{\text{reset}} = bdp$$

and for FF B

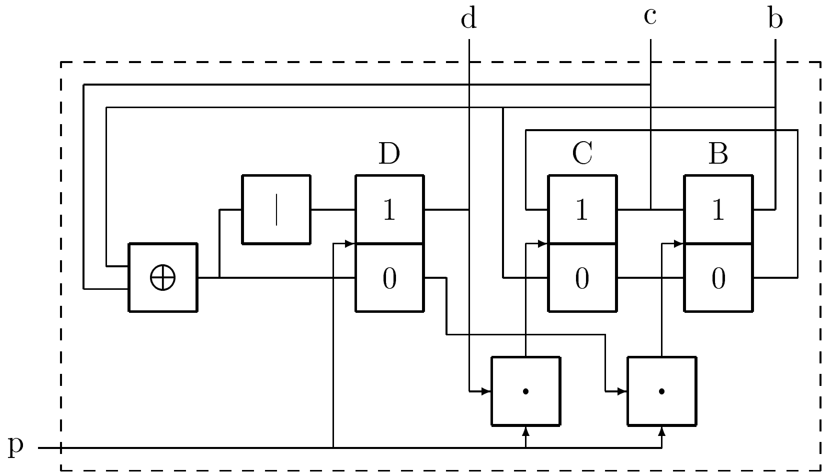
	d	d		
0	0	0	1	
0	×	×	×	b
		c	c	

$$B_{\text{set}} = \bar{d}cp$$

	d	d		
×	×	×	0	
1	0	0	0	b
		c	c	

$$B_{\text{reset}} = \bar{c}\bar{d}p$$

The resulting circuit is much simpler than in the case of a normal SR flip-flop, because it requires less gates:



Another example: a binary counter

Gray counters are not very common, but binary counters are. The design is exactly the same as in the previous example. Let us design a binary counter mod 8 with clocked pulses. Let us call the FF's 1, 2, and 4, which corresponds to the weight of the bits in the code. We can actually follow two design strategies: we can make a table and compare the bits, or we can follow the count sequence in the Karnaugh diagram and look at the bit values. Both are shown below:

before					after		
4	2	1			4	2	1
0	0	0	0	1	0	0	1
0	0	1	1	2	0	1	0
0	1	0	2	3	0	1	1
0	1	1	3	4	1	0	0
1	0	0	4	5	1	0	1
1	0	1	5	6	1	1	0
1	1	0	6	7	1	1	1
1	1	1	7	0	0	0	0

0	1	1	0	
	1	1		
0	1	3	2	
4	5	7	6	4
		2	2	

The table shows the 421 bit codes for the count sequence 0,1,2,...7, with the bit codes before the positive flank of the count pulse as well as after. This allows to compare the bits directly and filling the Karnaugh diagrams for the set and reset inputs. In the Karnaugh diagram with the count sequence we always must look at the bit values corresponding with the position. As an example, we'll take a look at the "1" flip-flop. The code starts with state 0 and goes to state 1, so bit "1" goes from 0 to 1. In this case we need a set, so we fill a 1 in the set diagram and a 0 in the reset diagram (shown below). Then the count goes from 1 to 2 and bit "1" goes from 1 to 0. We need a reset (set 0, reset 1). Etc. The complete Karnaugh diagrams, the functions, as well as the circuit are shown below:

	1	1		
1	0	0	1	
1	0	0	1	4
		2	2	

$S_1 = \bar{1}p$

	1	1		
0	1	1	0	
0	1	1	0	4
		2	2	

$R_1 = 1p$

	1	1		
0	1	0	×	
0	1	0	×	4
		2	2	

$S_2 = \bar{2}1p = \bar{2}R_1$

	1	1		
×	0	1	0	
×	0	1	0	4
		2	2	

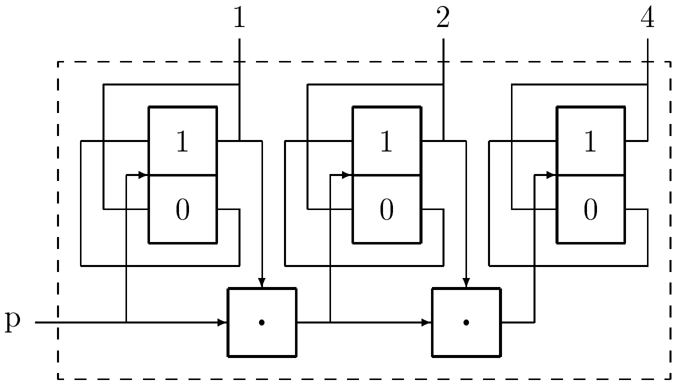
$R_2 = 21p = 2R_1$

	1	1		
0	0	1	0	
×	×	0	×	4
		2	2	

$S_4 = \bar{4}21p = \bar{4}R_2$

	1	1		
×	×	0	×	
0	0	1	0	4
		2	2	

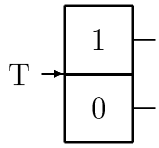
$R_4 = 421p = 4R_2$



Note: The use of R_1 in S_2 and of R_2 in S_4 is a simplification which is not strictly required. The advantage is that less elements are required. The disadvantage is a *ripple-through* of the counting flank (from FF 1 to FF 2 to FF 4); it takes some time to reach a stable state. You can draw the circuit with the direct combinations of the input signals. On a positive flank of p, all FF's can change at the same time (synchronous counter). This type of counter, which counts on a positive flank, is also called a *coherent* counter.

7.4 T flip-flop

In the last example, the binary counter, we have seen that the FF 1 counts on every input pulse, i.e. its switching is given by $p \cdot \bar{Q}$. In fact all three FF's are acting like this, which can be seen from the feedback loops around each FF. There exist flip-flops which do this automatically, the so-called T or **trigger** FF's. Each time the T FF sees a positive flank on its input, it will change from 1 to 0 or from 0 to 1.



For a binary counter, we must again indicate in Karnaugh diagrams when must be switched. There are **no** don't cares!

	1	1		
1	1	1	1	
1	1	1	1	4
		2	2	

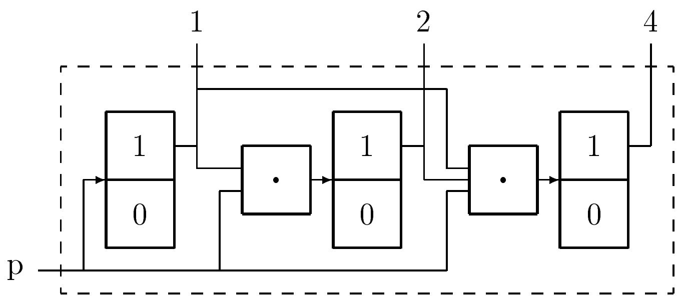
$T_1 = p$

	1	1		
0	1	1	0	
0	1	1	0	4
		2	2	

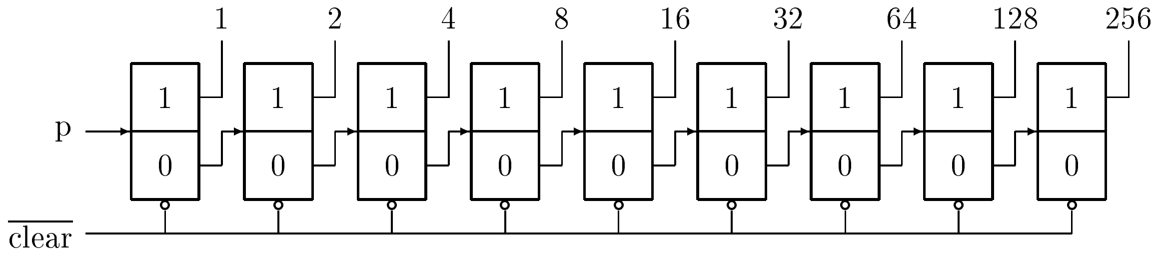
$T_2 = 1p$

	1	1		
0	0	1	0	
0	0	1	0	4
		2	2	

$T_4 = 21p = 2T_2$



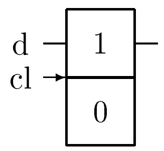
This is a synchronous counter, because all FF's can change at about the same time (but not quite; do you know how to change the above circuit such that all FF's can change at *exactly* the same time?). If a ripple-through is allowed, for example in non time-critical applications, it is even easier to use T flip-flops for making an asynchronous counter:



This is a mod 512 counter or an *up* counter: after a 0 pulse on the $\overline{\text{clear}}$ input, all FF's are 0. At each positive flank on the p input, the counter increments by one. If the delay of one FF is 10 ns, it may take 90 ns to reach a stable state.
Question 1: Do you know how to change this circuit to get a *down* or decremental counter?
Question 2: Do you know how to make an up-down counter which has one extra input, being 1 for up and 0 for down?

7.5 D flip-flop

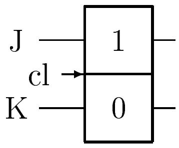
The **D flip-flop** is often used in registers or latches to store some bits, bytes or words temporarily. Its function is simple: it stores the level at the d input when it gets a positive flank at the cl or clock input. In other words: it copies the d input to the Q output. Making a counter with D FF's is also simple: just fill a Karnaugh diagram with the next level of each bit. Check this for a mod 8 binary counter.



7.6 JK flip-flop

The **JK flip-flop** is the most versatile one: it combines the functions of the SR and T flip-flops. The function table and symbol are show to the right. The inputs J and K can assume any value as long as the p input does not get a positive flank: the FF stays in its old state, $Q_{n+1} = Q_n$.

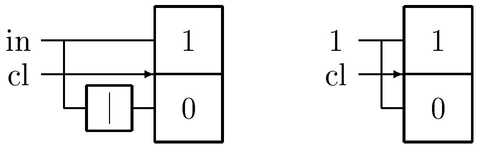
J	K	p	Q_{n+1}	
0	0	0	Q_n	stable
0	1	0	Q_n	stable
1	0	0	Q_n	stable
1	1	0	Q_n	stable
0	0	1	Q_n	stable
0	1	1	0	reset
1	0	1	1	set
1	1	1	$\bar{Q}_n$	T function



If we make a Karnaugh diagram for Q_{n+1},

Q \ JK	00	01	11	10
0	0	0	1	1
1	1	0	0	1

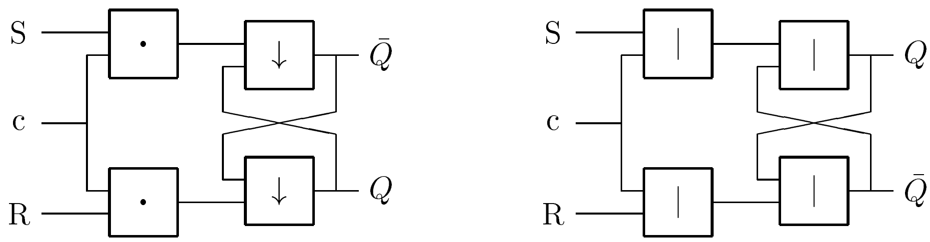
we can see that $Q_{n+1} = J \cdot \bar{Q}_n + \bar{K} \cdot Q_n$. A JK flip-flop can be used to make different functions, like a D or a T FF:



7.7 Other FF types

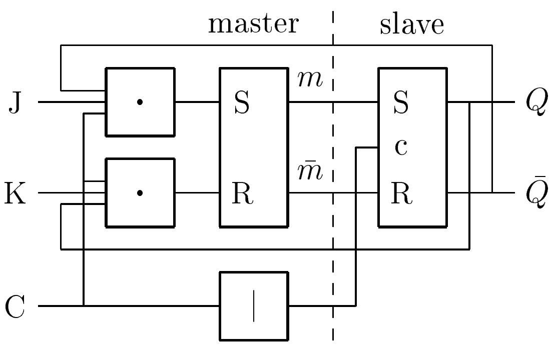
The flip-flops with a clock input which is sensitive to a positive (or negative) flank are often called *edge-triggered* FF's: the inputs, like S and R and D, must have a stable and valid value for a very brief time after the cl input goes high. There exist a few alternatives:

Gated latches. Examples:



These are the gated SR NOR latch (left) and the gated SR NAND latches. If the c input is 0, the S and R inputs are *disabled* and the outputs will remain stable. A 1 on the c input (or a brief positive pulse) *enables* the S and R inputs.

Pulse-triggered or master-slave FF's
Gated latches can be used in sequential circuits with feedback loops if the control signal c consists of a *very* brief pulse. If such a sequential circuit is driven by a periodic clock signal that controls all flip-flops, a method which is used very often, then problems may arise because delays and feedback loops may cause a sequence of transitions, even oscillations. For this reason **master-slave** flip-flops have been introduced. Example:



This is a master-slave JK flip-flop, which is a cascade of two gated SR latches such that the first latch (the master) drives the second (the slave). It works as follows:

• If C=1: the slave is disabled because of the invertor. The master is enabled, that is, the outputs m and $\bar{m}$ follow the inputs $J \cdot \bar{Q}$ and $K \cdot Q$. In other words, the master is *transparent.*

• If C makes a transition from 1 to 0, a negative flank: The master is disabled, that is, m and $\bar{m}$ will assume stable values depending on the previous values of $J \cdot \bar{Q}$ and $K \cdot Q$. Now the slave is enabled and will reflect the stable state of the master after a brief delay.

• If C=0: Now the slave is transparent, but the outputs cannot change (they are frozen) because the master provides it with stable input signals.

NOTE: All or most FF's (SR, JK, D, T) are available in three types: gated/basic, edge-triggered and master-slave. Edge-triggered are available for both positive (0 to 1) and negative (1 to 0) edges.