

Transients in combinational circuits

- 1 Delay of signals from inputs to outputs
- 2 Hazardous states in combinational circuits
- 3 Finding hazard in maps
- 4 Finding hazard from an expression
- 5 Hazard with changes of more than one variable
- 6 Dynamic hazard
- 7 False pulse removal
 - 7.1 Hazards elimination by redundant circuits
 - 7.2 Removing a false impulse by a filter
 - 7.3 Elimination of a false impulse by time division of activities

Transients in combinational circuits

We consider a combinational circuit with m inputs (x_1, \dots, x_m) and p outputs (y_1, \dots, y_p). When analyzing transients, two cases need to be distinguished:

- When the input state changes, the monitored output changes from $0 \rightarrow 1$ or $1 \rightarrow 0$. The extreme values of the signal **delay** at the monitored output against the input are investigated.
- When changing the input state, the monitored outputs should stay stable, but often short output pulses are generated instead. The situation in the circuit in which these **false pulses** may appear is called **hazard states** or shortly "hazards".

For a brief notation, we introduce the following symbolism:

- \tilde{a} variable a can have a value of 0 or 1 without affecting the output
- $a \updownarrow$ variable a varies alternating between 0 and 1
- $a \uparrow$ variable a changes from 0 to 1
- $a \downarrow$ variable a changes from 1 to 0

For example, notation $a\tilde{b}\tilde{c}d$ means a combination of values $a = 1, b = 0, c$ arbitrary, $d = 1$. Notation $\bar{a}b\bar{d}, c\updownarrow$ means that $a = 0, b = 1, d = 0, c$ oscillates between 0 and 1.

1 Delay of signals from inputs to outputs

The delay of the whole combinational circuit is given by the time from the change of the input state to the stabilization of the output state, i.e. **all** output signals. The delay of the circuit is a consequence of the delay of individual logic devices (parameter t_{pd}) and also of the connections - we will not deal with the influence of connections here. While the **maximum value** of t_{pd} for all types of logic devices can be found in the company datasheets, the minimum value is not given. Based on practical experience, about a third of the maximum value can be expected. **The actual** value of the delay is therefore **unknown** and depends on the selection of components, supply voltage, temperature, length of the connections, and load of the individual devices. Measurements on a real circuit are important only for verification of calculations, but a guaranteed value cannot be determined on this basis. On the contrary, the analysis of the circuit in terms of extreme values of the delay gives more reliable data.

To determine the delay from the selected input to the selected output, it is necessary to determine the **length of the path** from input x_i to output y_j , i.e. the number and type of logic devices in the path. With known extreme values t_{pd} of individual members in the path, the total delay can then be calculated as the sum of the individual delays along the entire path. In a very precise calculation, it is necessary to take into account the fact that for most devices $t_{pHL} \neq t_{pLH}$ and that it will therefore depend on the **direction** of the change in the input x_i ($0 \rightarrow 1$ or $1 \rightarrow 0$). In many circuits, the paths from different inputs to different outputs are of **different lengths**. The delay then depends on which input is changing. Sometimes we encounter a case where there is more than one path from the input to the output and the choice of the path through which the signal propagates may depend on some **other** inputs. In each case, however, the longest path and thus the **maximum** delay can be determined.

As an example, we examine the circuit delay in Figure 1. After the application of DeMorgan's rules, its function can be described as:

$$y = \overline{\overline{bcd} \cdot (b + ad)} = bcd + \overline{a} \overline{b} + \overline{b} \overline{d}$$

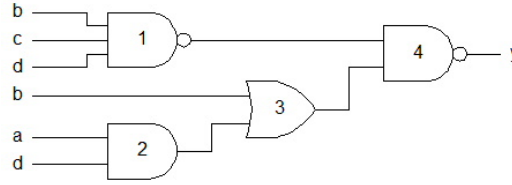


Fig. 1: Circuit with different delay times

Apparently there are three different lengths of paths: 1-4, 3-4, 2-3-4. Individual gates can have different delays and also $t_{pHL} \neq t_{pLH}$. We see that even with such a simple circuit, the delay can vary widely. It can also be shown that the delay from one input to one output (here there is only one) can have **different values**. Assume the state at the input abc and $d \uparrow$. Since $b = 1$, at the output of gate 3 will be state 1 (AND) and the output signal from the path 1-4. When entering abc and $d \uparrow$, at the output of gate 1 will be state 1 and the signal d will pass through 2-3-4, therefore longer.

2 Hazardous states in combinational circuits

Consider a situation where we apply signals from the previous circuits to the inputs of the AND or OR logic, one of which changes from 0 to 1 and the other vice versa, but the changes of both signals are **not absolutely simultaneous**. A pulse of the length given by the delay between the two signals is generated at the output. For different types of gates, the situation is summarized in Fig. 2.

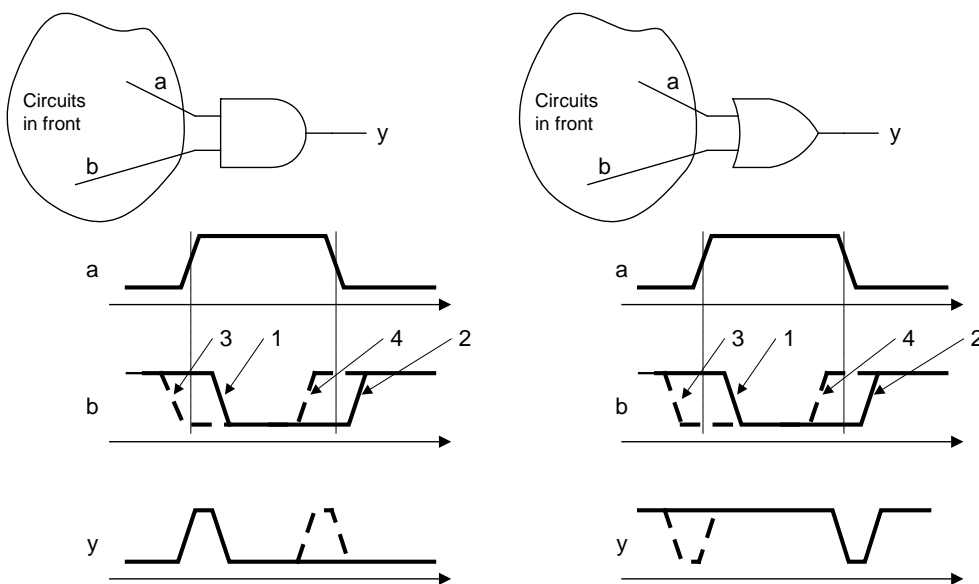


Fig. 2 : Generation of a false pulse in the AND and OR gate

In the left figure with the AND gate, with the signal b shifted after the signal a (waveform with falling edge 1 and leading edge 2), $a = b = 1$ for a moment, and thus $y = 1$. This will provide a short pulse at y marked by solid line. When the signal b is in advance of the signal a (falling edge 3 and a leading edge 4), the same impulse at y would appear later - see the dashed line y . The delay of the signals could be other - if the signal b had a falling edge 3 and a leading edge 2, the impulse to y would never occur. However, if the signal b had a falling edge 1 and a leading edge 4, even 2 pulses will be generated at y (full and dashed). The mutual delays of the signals therefore determine **whether** a false impulse occurs, **when** it occurs, or whether it **does not** occur at all. It is largely a matter of chance and that is why this phenomenon is called **hazard**. A similar situation occurs with the OR gate as shown on the right, only the false impulse is from 1 to 0. For the NAND gate and the NOR gate, the y -waveforms are inverted.

A false pulse can cause the same operation in other circuits as an intentionally generated pulse - which is, of course, a mistake. It will only be a question of whether the following circuits will register such a short false pulse. These can be circuits with a long time constant (e.g. relay), where a short pulse does not matter. In general, however, it is a **defect** and must be taken into account.

In a real circuit, we can never rule out mutual delays in the signals, simply because their paths are of different lengths. False pulse will even appear, even if both signals are changed simultaneously (which is only a hypothetical assumption). It is the consequence of the logic gate electrical properties when processing the input signals of a trapezoidal shape (which is the normal case).

In real circuits, therefore, most logical relations are valid only in a steady state. During the **transition**, we must modify them. In the figure 2, the signal b changed (approximately) in antiphase with the signal a , thus being its inversion. We would expect that

$$a \cdot \bar{a} = 0, \quad a + \bar{a} = 1,$$

however, this only applies at steady state. A false impulse was generated during the transition. Thus, the rules of Boolean algebra **are not applicable** in transients. This also applies to the operation of **combining** (minterms) according to the relation $\bar{x}\bar{y} + xy = x$. This is valid only in a steady state, because $\bar{x}\bar{y} + xy = x \cdot (y + \bar{y})$ and we already know that the sum $x + y$ is not 1, but it indicates hazard. The risk of hazard arises in any circuit where at the inputs of one of its elementary logic gates at least two signals change in the opposite direction approximately **simultaneously**.

An additional input of the gate may affect the occurrence of hazard at its output. If the additional input guarantees a **constant** state at the output regardless of other inputs, then there is no hazard. Consider an AND gate with three inputs. The value 0 is on the first of them, and 0 and 1 on the other two alternate in opposite phases. This, of course, would create a false impulse at the output. But if there is a constant value of 0 on the first input during that time, there will always be 0 on the output and no false impulse will occur - it is blocked. The same is true for the OR member, when the value 1 is on the **blocking** input - it alone guarantees the value 1 on the output.

Of course, blocking signals cannot block gates permanently - this would disable the whole circuit. Blocking is only necessary as long as the hazards could occur. It must therefore be **part of the design** of the logic function.

Fig. 3 shows the circuit implementing the function $y = \bar{b}\bar{c} + ac + \bar{a}b$. In the left half of the figure, bc , $a \uparrow$ is fed to the input of the circuit. On two inputs of gate 4 (OR) signals vary in mutually opposite directions and the third is constantly 0, which does not block hazard. The output will be a false pulse to 0. In the right half of the picture is a circuit with the same function, but slightly modified. With the same state at the inputs, there is a permanent state 1 at one input of the gate 4, which is sufficient to permanently force 1 at the output regardless of changes in the signals from the gates 2 and 3.

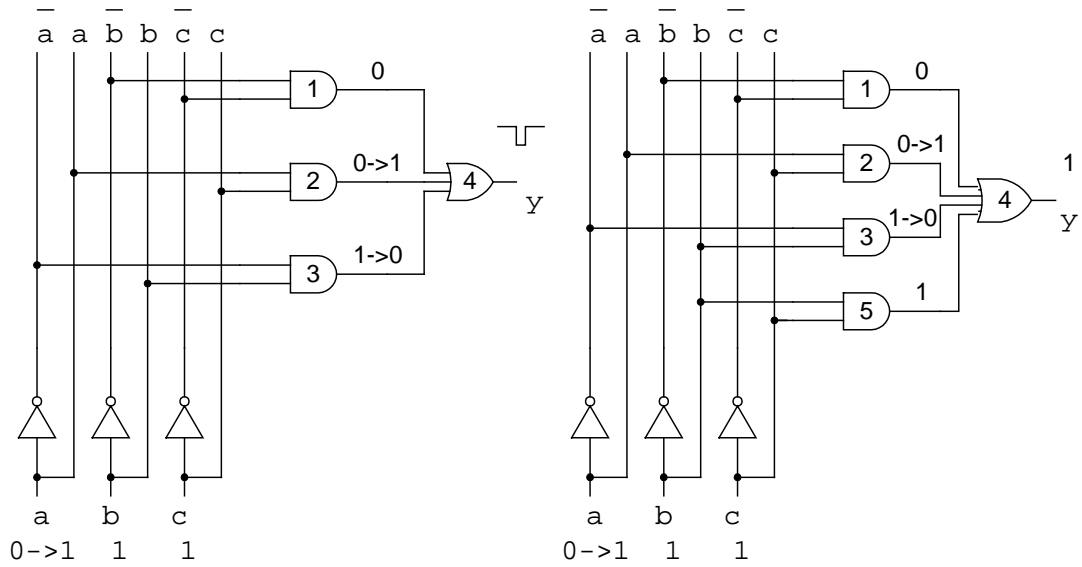


Fig. 3: Blocking hazard by signal with value 1

Implicants of the function y are $\bar{b}\bar{c}$, ac , $\bar{a}b$, created by the gates 1, 2, 3. During a change of state $\bar{a}bc$ to abc on the input is the value $y = 1$ provided initially by the implicant $\bar{a}b$ and then ac . There is thus a "jump" between two implicants, in which the other remaining implicants (here only $\bar{b}\bar{c}$) do not cover this jump (i.e. they do not give a value 1). In the right figure is an additional implicant bc , corresponding to the gate 5; it gives a permanent 1 when $b = 1$, $c = 1$, regardless of the changes of a . This masks the changes in the outputs of gates 2 and 3.

This suggests one of the methods for eliminating hazards. All jumps between implicants must be covered by additional, i.e. redundant implicants. This is the principle of the **redundant circuit** method. The question is how to create a redundant implicant. The **consensus** rule (exclusion of the third) can be used for this:

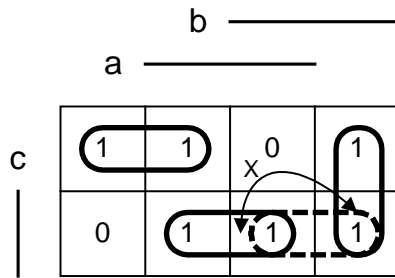
$$y = x.A + \bar{x}.B + A.B = x.A + \bar{x}.B \quad ,$$

where x , y are variables and A , B are arbitrary expressions. The $A.B$ term is redundant. For our case, we added bc to the implicants $ac + \bar{a}b$ and applied the rule in the opposite direction. The consensus is very well visible in the map - see below. The jumps between implicants that led to hazard are now covered by the redundant implicant. However, this only applies to

jumps caused by changes in the variable a . The effect of changes in other variables should still be determined and, if necessary, other redundant implicants should be introduced.

3 Finding hazard in maps

The coverage of the implicant function and the **jumps** between them can be seen very clearly in the maps. Fig. 4 shows the function $y = \bar{b}\bar{c} + ac + \bar{a}b$ from the Fig. 3 left. The loops in the map correspond to the implicants of the function.



Figt. 4: Jumps between implicants in the map

Arrow X shows the jumps between implicants, which evokes hazard. On the map, the jumps happen between two loops - ac and $\bar{a}b$. The dashed line indicates an extra loop bc . The jumps than happen inside this loop.

The map clearly shows the jumps between the implicants (loops) and the possibilities of covering these jumps with redundant implicants (loops). Jumps between the points $\bar{a}\bar{b}\bar{c} \leftrightarrow \bar{a}b\bar{c}$ can be covered with the redundant loop $\bar{a}\bar{c}$, jumps $a\bar{b}\bar{c} \leftrightarrow a\bar{b}c$ can be covered with the redundant loop $a\bar{b}$. However, not all jumps can be covered. E.g. it is not possible between the points $a\bar{b}\bar{c}$ and abc . In general, you **cannot** loop through points that are **not adjacent**.

4 Finding hazard from an expression

Hazards can also be found by analyzing the logic expression. Again, it is assumed that **only one** variable changes. It is necessary to find out whether or not a false output pulse is generated when some variable changes, and unfavorable (constant) values of other variables. To do this, the function is **decomposed** according to a variable that changes (e.g. x). This decomposition is always possible.

$$y = x \cdot A + \bar{x} \cdot B + C$$

Terms A , B , C denote separate variables or logical expressions. Any of the remaining variables may occur in them. At their constant values, the terms A or B or C then take the values 0 or 1. The presence of hazard is indicated by the expression $x + \bar{x}$, as already shown in the previous text. In order for the original expression to be converted to this form, it must be valid:

$$A = 1, B = 1, C = 0$$

This creates a set of **three equations**. If it has a solution, a hazard **arises** when the variable x changes. If there is no solution, hazard **does not arise**. The solution of the conditions yields a combination of values of other variables at which a hazard is possible. There is often more than one solution. The whole procedure is repeated for **all variables**.

The conditions $A = 1, B = 1, C = 0$ must all apply simultaneously. Therefore, it is possible to write $A \cdot B \cdot \bar{C} = 1$. This product is implemented and possibly simplified like any other logic expression. If it has a value of 1 for some combination of values of variables that are found in it, this combination is the sought **solution** of a set of equations.

Example - an expression that would indicate hazard:

$$z = \bar{b}\bar{d} + \bar{b}c + \bar{a}b$$

We are looking for the possibility of hazard at $b\uparrow$. We will decompose z according to this variable:

$$z = b(\bar{a}) + \bar{b}(c + \bar{d}) + 0$$

Obviously $A = \bar{a}, B = (c + \bar{d}), C = 0$. Therefore

$$\bar{a} \cdot (c + \bar{d}) \cdot 1 = 1, \text{ or } \bar{a}c + \bar{a}\bar{d} = 1$$

It has two solutions: $a = 0, c = 1$ or $a = 0, d = 0$

Thus, a false impulse at $\bar{a}c\bar{d}, b\uparrow$ (d arbitrary) or at $\bar{a}\bar{c}\bar{d}, b\uparrow$ **will be** generated in the circuit.

Example - hazard at $a\uparrow$ will not exist:

$$y = acd + bc + \bar{b}d + \bar{a}d$$

The same procedure leads to the condition

$$(cd)(d)(\overline{bc + \bar{b}d}) = 1, \quad \text{or} \quad cd \cdot \bar{b}\bar{d} + cd \cdot b\bar{c} + cd \cdot \bar{c}\bar{d} = 1$$

The expression on the left is equal to zero for any combination of values of the variables b, c, d and therefore the system has no solution. Hazard at $a\uparrow$ **does not threaten**.

5 Hazard with changes of more than one variable

If more input variables of the combinational circuit change approximately simultaneously, and the exact order of their changes is not known, then short-term **intermediate false states** between the old and new steady-state can appear at the output.

For instance, the two variables a and b of the function $y(a, b)$ can change in the following order between the states 01 and 10:

01 \rightarrow 11 \rightarrow 10 (a changed before b)
 01 \rightarrow 00 \rightarrow 10 (b changed before a)

Thus, all possible input states could occur. If the output state was the same value in both the old and the new input state, then at least one intermediate output state must exist different from the old and new one - otherwise the output would not depend on the input, which is a contradiction. A short-term false impulse can therefore be expected. In the case of a change of more than one input variable, no redundant circuits help. In general, for circuits with a change of more than one input variable, a false impulse should **always** be expected. It is therefore necessary to use other methods than redundancy to remove it (see next text) .

Changing the state of a **group** of signals so that intermediate states occur is called **signal races**. Concurrency are very important in the design of digital systems.

6 Dynamic hazard

The hazard described so far, manifesting itself as a false impulse instead of a constant value, is called **static**. Besides him, there is also a **dynamic** hazard. This is manifested as false bouncing on the rising or the falling edge of the signal instead of its simple change. This phenomenon occurs in the circuit where two signals join - one with a single change and the other with a false pulse - see Fig. 5.

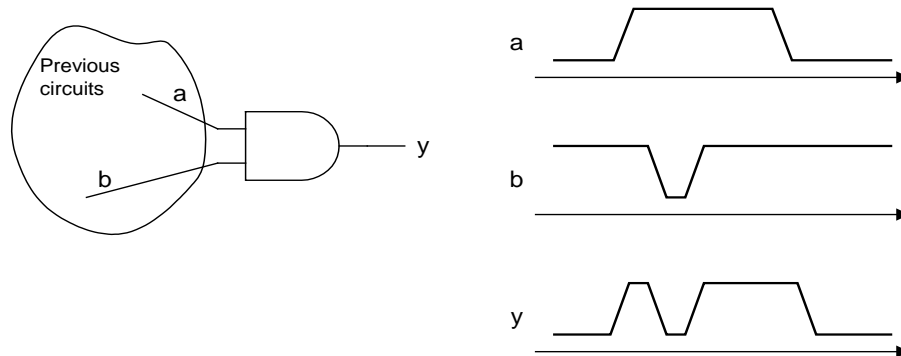


Fig. 5: Dynamic hazard

Dynamic hazard is similarly harmful as static hazard. It can lead to errors in evaluating the number of state changes (counting edges) of the signal. To remove it, it is sufficient to **remove** the **static** hazard - with constant b there would be no bouncing on y .

7 False pulses elimination

Several solutions can be used to eliminate the effect of false impulses. Each is suitable for different cases according to the function of the whole system.

7.1 Hazards elimination by redundant circuits

The modification of the circuits by the introduction of redundant components was explained in paragraph 2. The success of this method is only guaranteed if **only one variable** changes at a time. However, this is a very harsh condition and usually cannot be met. In a limited number of cases, codes with a change in one bit can be used - the best known is the **Gray code**.

The conversion between the 3-bit Gray and binary codes is shown in Tab. 1 below. The three rightmost columns illustrate the layout of the variables in the **Karnaugh** map. Indeed, this map was constructed using the Gray code. The extension to a larger number of variables is obvious.

b_2	b_1	b_0	g_2	g_1	g_0	g_2	g_1	g_0
0	0	0	0	0	0			
0	0	1	0	0	1			
0	1	0	0	1	1			
0	1	1	0	1	0			
1	0	0	1	1	0			
1	0	1	1	1	1			
1	1	0	1	0	1			
1	1	1	1	0	0			

Tab. 1: Relationship between binary and Gray code

The conversion relationships are as follows:

Binary → Gray

$$\begin{aligned} g_2 &= b_2 \\ g_1 &= b_2 \oplus b_1 \\ g_0 &= b_1 \oplus b_0 \end{aligned}$$

Gray → Binary

$$\begin{aligned} b_2 &= g_2 \\ b_1 &= g_2 \oplus g_1 \\ b_0 &= g_2 \oplus g_1 \oplus g_0 \end{aligned}$$

The extension to a larger number of variables is obvious. Operator \oplus denotes **exclusive OR** (XOR), expressed using the sum of products as follows:

$$y = a \oplus b = a.\bar{b} + \bar{a}.b$$

It should be noted that a change in only one bit of the Gray code occurs only if it goes through the table line-by-line from top to bottom, i.e. in the binary code **sequentially** from 000 to 111; otherwise a change in multiple bits may occur. E.g. with jumps in the binary code between 000 and 101, all bits in the Gray code change. There are not many cases where it is possible to pass through the input states sequentially, but they exist and are relevant.

The first case is sensors for **measuring position** or rotation **angle**. They are constructed on the basis of optics, magnetism, or contacts. In principle, the position sensor must travel through all positions one after the other - no jumps are possible principally. The paths for the individual bit sensors correspond to the lines in tab. 1 on the right, the position corresponds to the binary code on the left. The outputs from the position sensor can be introduced into the combination circuit, where redundancy can be easily applied and a hazard eliminated.

The second case is **counters** in Gray code (details are in another chapter). In normal operation, the counter gradually passes through all states, without jumps, in a Gray code. A combinational circuit with redundancy can then be connected directly to its output.

7.2 Removing a false impulse by a filter

This method is based on inserting the low-pass filter after the combinational circuit, so that the short pulses are **suppressed** - but not completely removed. Fig. 6 shows the principle.

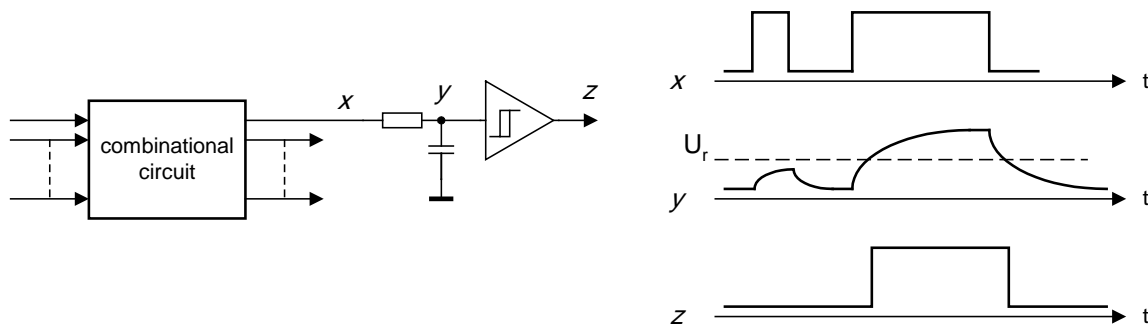


Fig. 6: Filter after the combinational circuit

The method does not eliminate the hazard inside the circuit, but a false impulse **at the output**. With a short pulse x , the voltage does not reach the reference level of the next device and the pulse is not "visible" at the output z . However, long pulses corresponding to the correct signal changes are delayed by the filter. The beveled edges of the signal should be formed by a device with a hysteresis characteristic. The filter **slows down** the operation of the entire system. The reliable function of the filter will depend on the length of the false pulse with respect to the time constant of the filter. For these reasons, this method is **not recommended**.

7.3 Elimination of a false impulse by time division of activities

The principle consists in the division of activity into two phases:

- **Preparatory**, when the output signals of the combinational circuit are generated.
- **Executive**, when these signals are used in subsequent circuits.

The executive phase begins at a time when transients are already guaranteed to **stabilize**. False impulses are not actually suppressed, but **ignored**. This method is **widely used**.

Registers are most often used as "follow-up" circuits. Data is written in the register by applying a write pulse W and remain stored until the next write pulse.

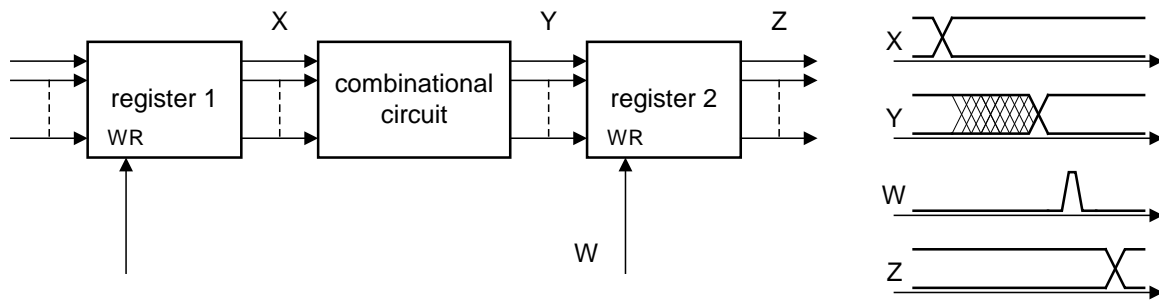


Fig. 7: Inserting a register at the output of the combinational circuit

The principle is shown in Fig. 7. The input signals X are changed by writing to the previous register 1. Then transient events occur at the Y outputs for some time (hatched). After their guaranteed fading, it is written by the impulse W to the following register 2. It will hold a stable signal and the danger of a false impulse is thus eliminated.

However, the structure of the whole digital system and the way of cooperation of its individual parts must be adapted to this. We come across the issue of **system synchronization**.

The question is the generation of writing pulses at the "right" moments, i.e. always **after changes** of states at the inputs of the combinational circuit. The easiest way is to extend synchronization in the previous stages. **Synchronization** pulses will be generated by the central control and timing block, and the combinational circuits inserted between the registers will provide **data processing** (data path). This is a completely **standard approach** to digital system design.