Reliability and diagnostics of digital systems

Control circuits and algorithms of the continuous diagnostics

Lecture notes for course PA175/12-2

Lesson 12 - content

12 Diagnostic of the digital systems

Origination of the failures is characteristic feature of the electronic systems. Therefore, testing of ones is frequency start task for determination state of the digital system hardware. There are group of possibilities how check state of the digital system. List of check method is next:

- program testing,
- microprogram testing,
- hardware test circuits.

Program test is possible separate into:

- developmental testing,
- manufacturing testing,
- user testing.

From point of start time can define:

- prophylactic testing,
- start testing,
- through (nonstop) testing.

For provision of maximum probability failure-free state is optimal implement through test components into digital systems. Through tests are running continuously - against a background of the operation system and user tasks. For its function the through testing needs hardware check circuits. In next chapter is discussed hardware support of the through testing - checking circuits.

12.1 Checking circuits

There are group of method used for non-stop testing of the digital system hardware:

- redundancy:
    - information redundancy,
    - hardware redundancy,
- prediction,
- checking of the time sequences.

For explanation of principle will use electric scheme and shot comment.

12.1.1 Information redundancy

Numbers and data transmitted, processed and stored in individual modules, design of digital systems may be all sorts of influences amended. Improved or less complete protection against these adverse effects of providing information to enable redundancy and to detect adverse changes in advanced implementations as possible to repair. In subsequent chapters, the question of security code, namely the issue of detection and correction of faults and errors in the transmitted data discussed in more detail.

12.1.1.1 Theory of information redundancy operation

Information required adding add redundancy data bits, bearing on the control bits. This means that the input data into the system is installed generator control bits are added to information bits. The whole system is transmitted (including the entries in the memory system) and at designated sites to check its accuracy. Control is performed so that the transmitted data bits to generate control bits and

these are compared with devolved control bits or the rules of correct shape-secure code, see the following chart.

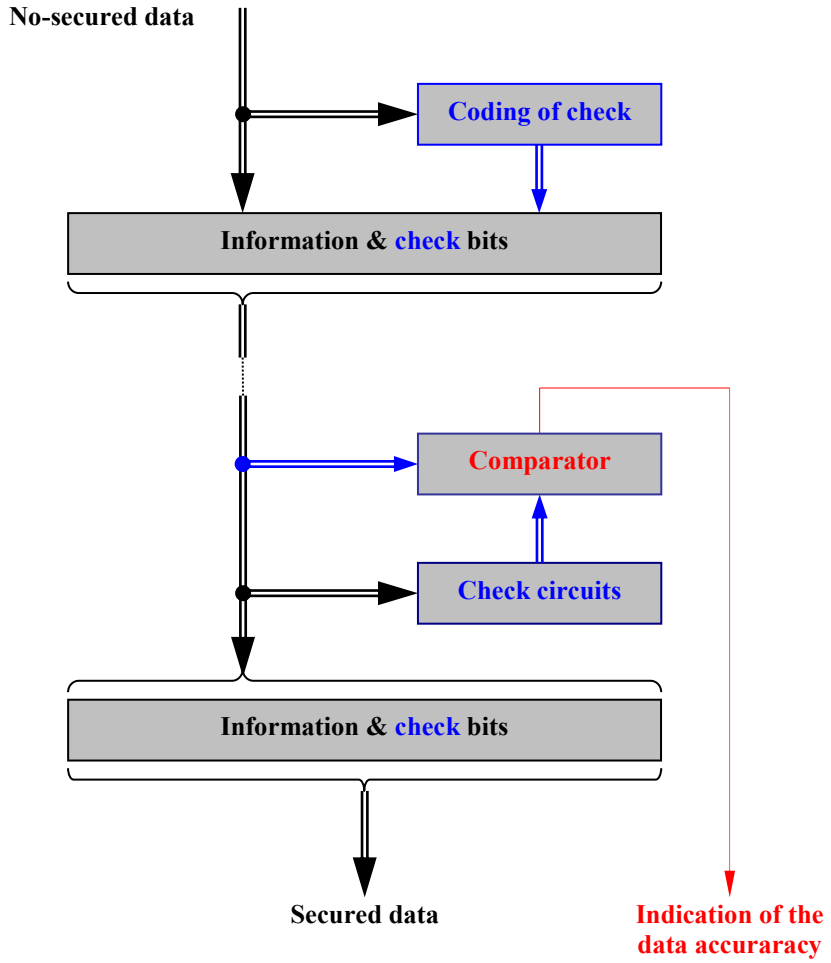


Figure 12.1: Working principle of data protection information redundancy

Indications violation of the correctness of data is a logic level (electrical signal), which indicates its level of integrity of the data at the controls. This signal is used most often interrupt system that runs sub-treatment of this indication. The specific response depends on the spot verification of data in the system development phase of the order during which the indication of the accuracy of the data breach and the configuration of the operating system.

Kinds of information redundancy for system security against failures are show below:

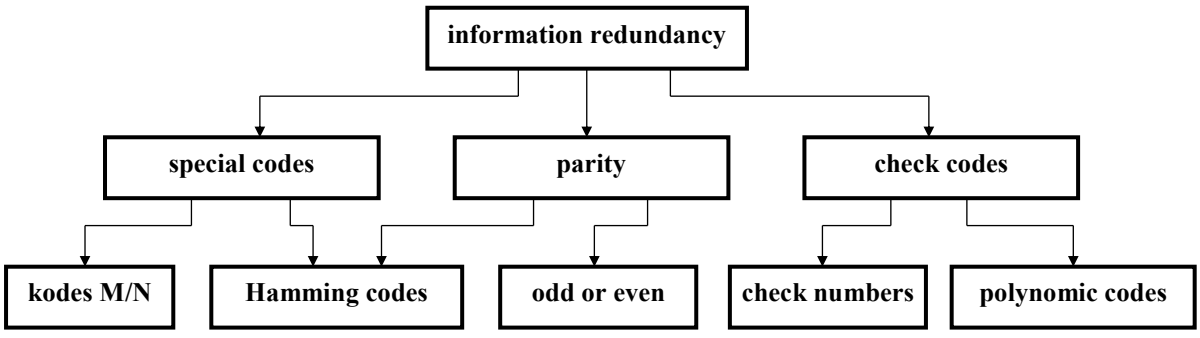


Figure 12.2: Classification of information redundancy methods for detection and repair information damage

file

Note:

It should consider the fact that the disorder may be in the control circuit and indication of the accuracy of the data breach is issued false or not licensed at all.

12.1.1.2 Parity information security

This principle uses the redundancy of information often within a single byte. Redundancy bit informs the system whether a selected number of bits preserve the original number. If the controlled words invert two bits, the parity does not indicate adverse condition. It is thus obvious that parity data protection applications are very limited.

Parity is used both for creating security bits of information transmitted and stored in digital systems, as well as for checking secure information. Parity is an additional bit, which complements the information bits even or odd number of ones. Most often generates parity for byte and odd parity is used when a binary zero in the parity bit is 1 and it is clear that the data source is working (if it had been disconnected, they will have all the bits including the parity value **0**), see next picture.

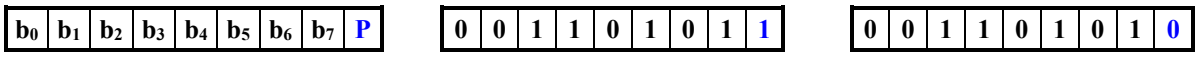


Figure 12.3: Principle and example of parity security implementation
a) principle of redundancy security,
b) example of odd parity security,
c) example of event parity security.

The above diagram represents a linear parity word (here byte), the change is detected by only one bit. This means that a simple inversion of two bits of parity already detect.

This gap partly eliminates another independent parity bit words - every independent parity bit increases the distance of the so-called code value of 1. Independence of the parity bit means that it is generated over another set of data bits than any previous parity bits. A characteristic example is the so-called **cross-parity** - see next tables:

even parity	8	4	2	1	P_1
	0	**1**	**1**	**1**	**1**
	0	**1**	**1**	**0**	**0**
	1	**0**	**0**	**0**	**1**
	0	**1**	**0**	**0**	**1**
P_2	**1**	**1**	**0**	**1**	**1**

odd parity	8	4	2	1	P_1
	0	**1**	**1**	**1**	**0**
	0	**1**	**1**	**0**	**1**
	1	**0**	**0**	**0**	**0**
	0	**1**	**0**	**0**	**0**
P_2	**0**	**0**	**1**	**0**	**0**

Table 12.1: Principle of cross-parity mechanism

even parity	8	4	2	1	P_1
	0	**1**	**1**	**1**	**1**
	0	**1**	**1**	**0**	**0**
	1	**0**	**1**	**0**	**1**
	0	**1**	**0**	**0**	**1**
P_2	**1**	**1**	**0**	**1**	**1**

odd parity	8	4	2	1	P_1
	0	**1**	**1**	**1**	**0**
	0	**1**	**1**	**0**	**1**
	1	**0**	**1**	**0**	**0**
	0	**1**	**0**	**0**	**0**
P_2	**0**	**0**	**1**	**0**	**0**

Table 12.2: Indication of failure by cross-parity mechanism

Note:

Cross-parity allows correcting an error in the four tetrad inversion at the intersection of fields with the wrongly generated parity.

Parity bit can be generated serially or in parallel. Role serial parity generator successfully performs a synchronous flip-flop type **T**, whose data input data are introduced in serial form. Direct output generates even number parity and odd parity inversion output.

Serial parity checks this, the input generator will take both the meaning and the redundant bits and check whether the unit was controlled by some odd digits of **1** (odd parity) or even digits of **1** (even parity).

The following figure shows the scheme of the parity bit generating for serial transmitted byte (eight-bit word).

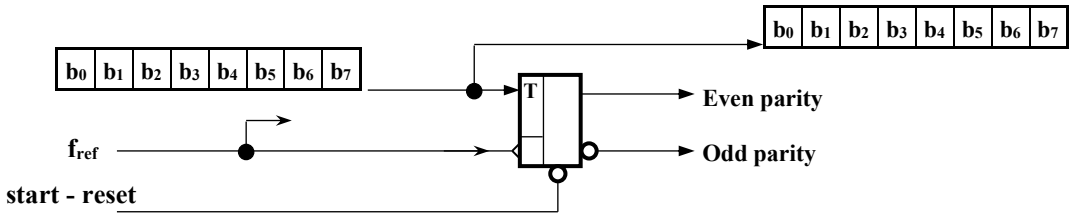


Figure 12.4: Principle of design and example of parity serial generator

The construction of parallel parity generator and also to build a control circuit parity can be used the **XOR** circuits. These circuits are often available with direct and inverse output (so-called **M2** circuit) and allow the generator and the control circuit to draw in the so-called *two-wire logic* (it will be the discussion in the chapter on peripheral redundancy).

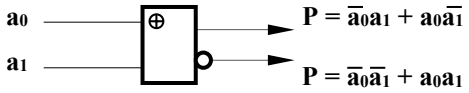


Figure 12.5: Schematic symbol of the **M2** circuit - standardized sum modulo 2

Because the logical **XOR** function does not fully functional system, it not be used for systematic process their applications. Was made a number methods and rules for transcription of Boolean functions in form witch are applicable for **XOR** circuits (Zegalkin algebra).

Standardized circuit **M2** has a direct and inverse output, and therefore can be used to generate an even and odd parity bit. In the absence of **XOR** circuits can be applied more frequently offered **AND-OR-INVERT** ones. These circuits can easily build a universal two-bit parity generator that also generates two types of parity in the form of so-called two-wire logic. Scheme of this is displayed in the following figure.

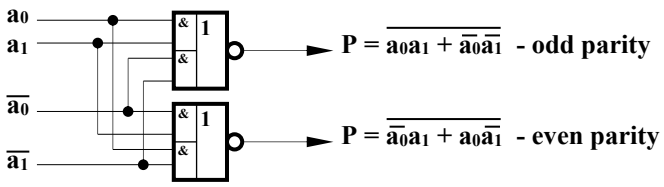


Figure 12.6: Elementary cell of the parity generator based on **AND-OR-INVERT** circuit

For further interpretation, we will use for this cell pattern as shown - see Figure 12.7

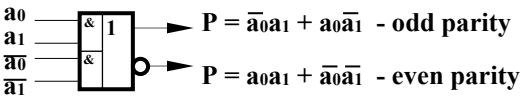


Figure 12.7: Schematic symbol of the parity generator elementary cell

If we want to establish multi-bit parity circuit, offering to apply the method of substitution and the parity generator has the shape of the tree (pyramid). The principle of substitution is the use of the method outlined in the following Table 12.3

b_0	b_1	L_1
0	0	1
0	1	0
1	0	0
1	1	1

L_1	L_2	L_P
0	0	1
0	1	0
1	0	0
1	1	1

L_P	L_Q	L_α	**generation of odd parity**
0	0	1	enter even number of 1
0	1	0	enter odd number of 1
1	0	0	enter odd number of 1
1	1	1	enter even number of 1

Table 12.3: To explain of the substitution principle used for generating the parity bit

The corresponding eight-bit odd parity generator (compiled on the basis of the application of substitution method) is shown in Figure 12.8

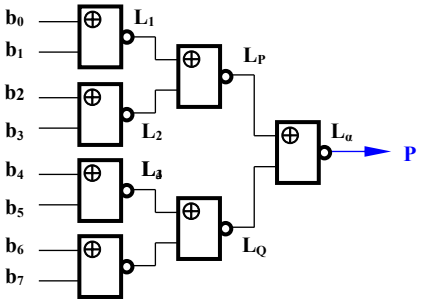


Figure 12.8: Principle and example of parallel parity generator

Parity is often generated within a byte. The choice of odd or even parity is depending on the specific characteristics of the controlled circuit. Very common is the protection of data in memory odd parity. If the memory is disconnected or the read address is outside the memory capacity, then read byte parity bit has zero value and the error is indicated. Indication of the violation of parity is carrier of this information (other than information about the read error information).

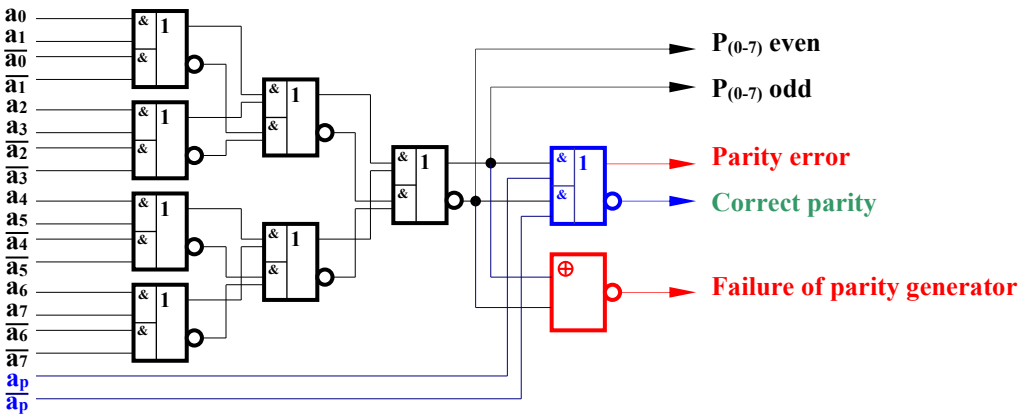


Figure 12.9: The complete schema of the parity generator with checking circuits

file

12.1.1.3 N/M code information security

For these codes applies rule that every valid character is shown as a group of **N** bits with value **log.1** from **M** bits. It diminishes the capacity of the code, although it did in comparison with the classical binary code, but increases the distance code, which allows you to detect single or multiple errors of the same type (**log.0** or **log.1**). It is this feature codes **N/M** is very advantageous, because the current electronic systems malfunctions arise situations where accruing **log.0** incorrect or defective l**og.1**.

A number of conceivable combinations expressed by **N/M** code can be calculated as the combination of **N** elements from set **M** elements with repeating, see 12.1:

$$\mathbf{D = \frac{N!}{M!*(N-M)!}} \qquad \mathbf{12.1}$$

If this code is used is to attempt to assign to numbers a combination, when the numbers assigned to each combination of near certain weighted code (see the following sample code 2/5 code very analogous to **01247 weighted code**).

Decimal form	**2/5**					**2/7**						
	0	**1**	**2**	**4**	**7**	**5**	**0**	**4**	**3**	**2**	**1**	**0**
0	0	0	0	**1**	**1**	0	**1**	0	0	0	0	**1**
1	**1**	**1**	0	0	0	0	**1**	0	0	0	**1**	0
2	1	0	**1**	0	0	0	**1**	0	0	**1**	0	0
3	0	**1**	**1**	0	0	0	**1**	0	**1**	0	0	0
4	**1**	0	0	**1**	0	0	**1**	**1**	0	0	0	0
5	0	**1**	0	**1**	0	**1**	0	0	0	0	0	**1**
6	0	0	**1**	**1**	0	**1**	0	0	0	0	**1**	0
7	**1**	0	0	0	**1**	**1**	0	0	0	**1**	0	0
8	0	**1**	0	0	**1**	**1**	0	0	**1**	0	0	0
9	0	0	**1**	0	**1**	**1**	0	**1**	0	0	0	0

Table 12.4: Examples of the **2/5** and **2/7** codes

Check the correctness of the code can be done by combination circuit or by a simple analog summing circuit. Karnaugh map for the establishment of the combinations circuit for checking code **2/5** is given in the following table.

4 7 \ **0 1 2**			**3**			**1**		**2**
		5		**6**	**4**			
0								
7	**8**		**9**					

Table 12.5: Karnaugh map for the control circuit of the **2/5** code (for convenience, valid fields are represented by values of the numbers)

The table shows that the logical formula can be substantially minimized and, therefore, that the realization of a control circuit to be somewhat cumbersome.

The second option is an analogue adder, which counted the input voltage level. Analog comparators detect on the adders output permitted level of the total voltage sum of two **log.1** and three **log.0**.

Although it is possible to replace comparators by transistors, realization of the resistance in a structure of the integrated circuit is more than debatable (although here depends on the ratio of resistors and

not on their absolute value). Analog circuit is not an appropriate subject for the implementation of the current structure of digital circuits, and therefore this possibility given the full picture of the possible solutions.

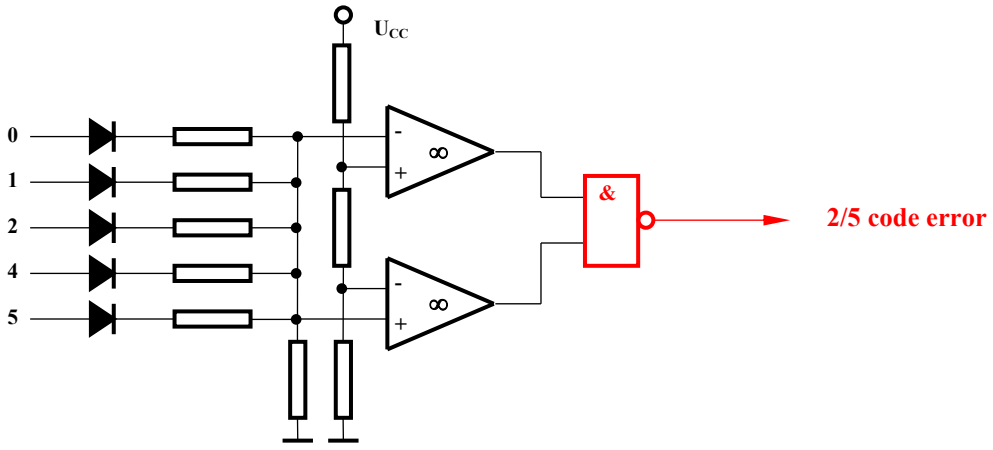


Figure 12.10: Block diagram of the **2/5** code analog control circuit

12.1.1.4 Hamming codes

This is a special group of linear binary **(n, k)** codes with code distance **3** or **4**, which enable indication of the double-failures and simple-failures correction. Adding a few bits of meaningful bits into protected words increases the code distance of the resulting code. Hamming code can describe the control matrix, whose columns are all the nonzero words of length **n - k = r** which no one repeat.

12.1.1.4.1 Hamming code generation algorithm

The unsecured **m** bits a_i of the word **A** with a code distance **1** is join on **k** other bits p_1, p_2, ... p_k of the control word **P**, which consists from unique, non-repeat subsets bits a_i. Secured word **B** formed by bits b_1, b_2, ... b_{m+k} we get by the appropriate grouping of bits a_i and p_i. Above is secured word **B** is possible form a number **S** consisting from bits s_1, s_2, ... s_k (*error of syndrome*), whose numerical value directly indicate the order of bit errors in the secure word **B**.

Note:

Syndrome gives the position of all the wrong bits secured word and of the parity bits of course.

Syndrome reflects the position of bit error with a faulty value, and therefore value **0** corresponds to flawless code and the values from **1** to **m + k** corresponds to locate bad words. Because it is necessary to cover a total of **m + k +1** option, the number of bits of security to meet the inequality 12.2:

$$2^k \geq m+k+1 \qquad \textbf{12.2}$$

To ensure four-bit word is **k= 3**. Number of control bits we get as the solution equation 12.2, a quantification of the following relationship:

$2^k \geq 4+k+1$ which is fulfilled for $k \geq 3$

The procedure of generating control bits can be described as follows:

- position of equal power of the resulting code used for the unique parity bits p_i, where **i** takes values (**1, 2, 4, 8, 16, 32, ...**)
- other positions are allocated bits unsecured words a_i, where **i** takes values (**3, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15, 17, ...**)

- each parity bit is calculated from a unique combination of bits of the insecure word. Location of the parity bit clearly specifies which sequence of bits is used for generating the parity bit.

Parity bit $\mathbf{p_1}$ (first position) belongs to all the odd bits from the information bit $\mathbf{a_1}$ of the secure word, so the 3rd bit of the secured word. Parity bit $\mathbf{p_2}$ (second position) is the security bit of each odd pair bits from the secured word starting with **2nd** and **3rd** bits of the secured word. Parity bit $\mathbf{p_3}$ (fourth position) provides each odd four bits of the secured word starting with bits **4**, **5**, **6** and **7** of the secured words. For additional parity bits are proceeding by analogy.

The previous interpretation that four-bit binary code is necessary to ensure a minimum of **3** bits, the resulting word has the following format:

$$\mathbf{p_1^{2^0}, p_2^{2^1}, a_1^{2^2}, p_3^{2^3}, a_2^{2^4}, a_3^{2^5}, a_4^{2^6}}$$

The individual bits of security code $\mathbf{p_i}$ (generated as even parity bits) are generated by creating the following equations:

$$\mathbf{p_1 = a_1 \oplus a_2 \oplus a_4}$$
$$\mathbf{p_2 = a_1 \oplus a_3 \oplus a_4}$$
$$\mathbf{p_3 = a_2 \oplus a_3 \oplus a_4}$$

Is clear thus satisfy the following equation:

$\mathbf{p_1 \oplus a_1 \oplus a_2 \oplus a_4 = 0}$ - left site of the equation is the syndrome bit s_3 (positon $\mathbf{b_1}$, $\mathbf{b_3}$, $\mathbf{b_5}$, $\mathbf{b_7}$ of secured word)

$\mathbf{p_2 \oplus a_1 \oplus a_3 \oplus a_4 = 0}$ - left site of the equation is the syndrome bit s_2 (positon $\mathbf{b_2}$, $\mathbf{b_3}$, $\mathbf{b_6}$, $\mathbf{b_7}$ of secured word)

$\mathbf{p_3 \oplus a_2 \oplus a_3 \oplus a_4 = 0}$ - left site of the equation is the syndrome bit s_1 (position $\mathbf{b_4}$, $\mathbf{b_5}$, $\mathbf{b_6}$, $\mathbf{b_7}$ of secured word)

From the generating equations is clear that every bit of the security code is generated from another set of data bits. Each syndrome bit in generating equations of the syndrome bits is repeating in the binary expression of their position in the secured word. If one of the secure bits inverts the words, equation syndrome bits take the values that correspond to the position of the changed bit (in binary).

Generating individual bits of Hamming code and its properties demonstrates the following example.

Example:

*Build single-fault correcting Hamming code for the **BCD 8421** and check its properties.*

Solution:

*For **BCD** code is **m = 4.** To meet the expression 1.14 must be **k ≥ 3.** If we choose the shortest code, the Hamming code **BCD** words will have a length of 7 bits, and its shape is as follows:*

$$b_1,\ b_2,\ b_3,\ b_4,\ b_5,\ b_6,\ b_7$$

Whereas the importance and weight of the individual bits in a safe word is as follows:

$$p_1^{2^0}, p_2^{2^1}, a_1^{2^2}, p_3^{2^3}, a_2^{2^4}, a_3^{2^5}, a_4^{2^6}$$

Syndrome individual bits are generated as follows (see the rules in the preceding text):

- *bit s_1 is generated from bits b_4, b_5, b_6, b_7*
- *bit s_2 is created from bits b_2, b_3, b_6, b_7*
- *bit s_3 is the result of the evaluation of bits b_1, b_3, b_5, b_7.*

failure position	syndrome		
	s_3	s_2	s_1
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

*Each additional parity bit is generated from another set of bits of the original (source) words. Because syndrome bits are generated from each different (unique) bit sets of the source words, each error accompanied by another syndrome - see next table. Syndrome **S**, quantified bits (s_1 s_2 s_3), expresses the position of erroneous bit in the secured word expressed in binary code. The syndrome bits are thus generated in the previous table. In each column is highlighted by gray color fields involved in generating syndrome bits. Yellow highlighted fields are positions (additional, parity) bits of security.*

*It is evident that the position expressed syndrome erroneous bits in the secured word. It is necessary to set up your error bit s_1 in positions **1**, **3**, **5** and 7, s_2 bit to set up your error in positions **2**, **3**, **6** and 7 and bit error s_3 positions **4**, **5**, **6** and 7. Syndrome consists from the parity bits values of each group position errors (see previous table).*

*Now the task is to assign position of the Hamming code bits and the BCD code bits into secured word. In essence, additional parity bits can be placed in any secure word position. However, to uniquely identify the error position, the weight of the syndrome bit must be uniquely related to the position of the parity bit of the error position group. Because of the easy location of the wrong bit, it is preferable to place the additional parity bits at positions with the weight corresponding to the weight of these bits in the syndrome. In general, the parity bits are at positions **1**, **2**, **4**, **8**, ...2^{k-1}.*

*Security bits located in positions **1**, **2** and **4**. Bit p_1 is located at bit position **4**, bit p_2 at bit position **2** and bit p_3 in position **1** (see following table with blue highlighted syndrome bit value).*

*Example of generating Hamming code word by entering the number **4** for example is shown in the table below.*

position	1	2	3	4	5	6	7
	p_1	p_2	m_1	p_3	m_2	m_3	m_4
BCD = 4			0		1	0	0
$p_{1\,(3,5,7)}$ =1	1		0		1	0	0
$p_{2\,(3,6,7)}$ =0		0	0		1	0	0
$p_{3\,(5,6,7)}$ =1			0	1	1	0	0
secured word	1	0	0	1	1	0	0

To illustrate the indications and the possibility of correcting an error is in the next table shown generation of the syndrome for both words - with inverted 7th bit secured word, and then for the same word with intact 7th bit. Syndrome error code for a broken word is equal to the value of 7 - it shows an error in the 7th secure bit words.

failed word	1	0	0	1	1	0	1
$s_{1\,(1,3,5,7)}$ =1	1	0	0	1	1	0	1
$s_{2\,(2,3,6,7)}$ =1	1	0	0	1	1	0	1
$s_{3\,(4,5,6,7)}$ =1	1	0	0	1	1	0	1

non failed word	1	0	0	1	1	0	0
$s_{1\,(1,3,5,7)}$ =0	1	0	0	1	1	0	0
$s_{2\,(2,3,6,7)}$ =0	1	0	0	1	1	0	0

file

$s_{3\,(4,5,6,7)}=0$	1	0	0	1	1	0	0

*Syndrome error code intact word is equal to the value of **0** - it reports on the accuracy of secure code words.*

decimal number	1 p_1	2 p_2	3 m_1	4 p_3	5 m_2	6 m_3	7 m_4
0	0	0	0	0	0	0	0
1	1	1	0	1	0	0	1
2	0	1	0	1	0	1	0
3	1	0	0	0	0	1	1
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	1
6	1	1	0	0	1	1	0
7	0	0	0	1	1	1	1
8	1	1	1	0	0	0	0
9	0	0	1	1	0	0	1

Table 12.6: List of **BCD 8421** code digits secured by the Hamming code.

The example demonstrates that generate secure bits, indications of the single-fault and its correction by Hamming code is the combination logic circuit resolved. Complete list of the **BCD 842**1 digits secured by the Hamming code is in Table 12.6.

The amount of generated code is the code distance 3rd. Every bit of the **BCD** digits in the code is checked at least two additional parity bits.

Note:

Each independent additional parity bit of the Hamming code increases the code distance of the resulting code of value 1.

The whole process of generating Hamming code can be precisely expressed using a generator matrix $\mathbf{G_H}$. For four-bit code constructs it so that it progressively encodes sequences $\mathbf{1000_1}$, $\mathbf{0100_2}$, $\mathbf{0010_3}$ and $\mathbf{0001_4}$ (so that the rows are linearly independent and formed the basis of space).

$$\mathbf{G_H}=\begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & p_{1_1} & p_{2_1} & 1 & p_{3_1} & 0 & 0 & 0 \\ 2 & p_{1_2} & p_{2_2} & 0 & p_{3_2} & 1 & 0 & 0 \\ 3 & p_{1_3} & p_{2_3} & 0 & p_{3_3} & 0 & 1 & 0 \\ 4 & p_{1_4} & p_{2_4} & 0 & p_{3_4} & 0 & 0 & 1 \end{pmatrix}=\begin{pmatrix} & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 4 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Similarly, it is possible to derive the control matrix Hamming code called $\mathbf{H_H}$. Bits $\mathbf{b_3, b_5, b_6, b_7}$ contain bits $\mathbf{a_i}$ of information and even parity bits $\mathbf{b_1, b_2, b_4}$ are unique sets of data bits $\mathbf{a_i}$. Quantifying rates of all the unique sets getting syndrome $\mathbf{s_1, s_2, s_4}$ secured word, see the following relations:

$$\begin{aligned} \mathbf{s_1} &= \mathbf{b_4}\oplus\mathbf{b_5}\oplus\mathbf{b_6}\oplus\mathbf{b_7}=\mathbf{0} \\ \mathbf{s_2} &= \mathbf{b_2}\oplus\mathbf{b_3}\oplus\mathbf{b_6}\oplus\mathbf{b_7}=\mathbf{0}\Rightarrow \mathbf{H_H}= \\ \mathbf{s_3} &= \mathbf{b_1}\oplus\mathbf{b_3}\oplus\mathbf{b_5}\oplus\mathbf{b_7}=\mathbf{0} \end{aligned} \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

12.1.1.4.2 Enhanced Hamming code

To increase the effectiveness of Hamming code was created so-called *extension of the binary Hamming code*, which is based on adding additional symbol $\mathbf{p_0}$ to the beginning of each coded word.

This additional bit is used for control of the word parity. Bit $\mathbf{p_0}$ is chosen so that all bits $\mathbf{b_i}$ secured word is formed as the even parity, see below:

$$\mathbf{p_0 \oplus b_1 \oplus b_2 \oplus b_3 \oplus b_4 \oplus b_5 \oplus b_6 \oplus b_7 = 0}$$

The extended code allows correct the single-error and detects the double-error in the word. Generator matrix $\mathbf{G'_H}$ of the extended Hamming code construct as is the case otherwise of the basic Hamming code so that gradually encoding the sequences $\mathbf{1000_1}$, $\mathbf{0100_2}$, $\mathbf{0010_3}$, $\mathbf{0001_4}$.

$$\mathbf{G'_H} = \begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & p_{0_1} & p_{1_1} & p_{2_1} & 1 & p_{3_1} & 0 & 0 & 0 \\ 2 & p_{0_2} & p_{1_2} & p_{2_2} & 0 & p_{3_2} & 1 & 0 & 0 \\ 3 & p_{0_3} & p_{1_3} & p_{2_3} & 0 & p_{3_3} & 0 & 1 & 0 \\ 4 & p_{0_4} & p_{1_4} & p_{2_4} & 0 & p_{3_4} & 0 & 0 & 1 \end{array} = \begin{array}{c|cccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 3 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}$$

12.1.1.4.3 Decoding and check of the reading words by Hamming code

First after receipting of the secured word **B** is quantifying syndrome bits **S**. Is possible use the matrix operation $\mathbf{S = H_H * B.}$ The procedure is demonstrated by the following example.

Example:

Determine syndrome of the received word $\boldsymbol{B = 1010111}$ *secured by Hamming code using the control matrix* $\boldsymbol{H_H}$*:*

$$\mathbf{S = H_H * B} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$$

Syndrome $\boldsymbol{S}$ *is equal to the number* $\boldsymbol{6}$*, which means that during a write or read data has appeared an error sixth bit of the secured word* $\boldsymbol{B}$*. Flawless word has the form* $\boldsymbol{B = 101,010,101}$ *- compared with the adopted by the inverted bit* $\boldsymbol{b_6}$*. Checking of this word is given below:*

$$\mathbf{S = H_H * B} = \begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix} * \begin{pmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

12.1.1.5 Check number

Control numbers are a simple means of security information. They are an appropriate means for checking of the information block. May take the form of a checksum or add a checksum to the pre-

selected values.

Checksum is different from the arithmetic sum transfers to neglect higher order, is actually the lowest order of the arithmetic sum, or add it to the selected value.

Simple checksum error can be clearly detected. In addition, the checksum can also detect multiple faults and clusters of errors. Can prove in theory there are clusters of errors that the checksum cannot be detected, because the individual errors offset each other. The probability of such errors in the real transmission channel is negligible, because in these conditions occurs, the increased sensitivity to noise **log.1** or **log.0** and not complementary sensitivity (this phenomenon can use intruders of the communication channels, but there are other factors than the subject of our research).

Example:

*Build a checksum for twelve **BCD code** number and check its properties.*

1	**2**	**4**	**5**	**8**	**6**	**0**	**0**	**0**	**0**	**0**	**0**	**6**
1	**2**	**4**	**5**	**8**	**6**	**0**	**0**	**0**	**0**	**0**	**0**	**4**

1+2+4+5+8+0 ..... +0=26 → check sum = **6**, complement to 10 = **4**

If we do not only error detect, but also require the error correction must be the check digit combine with other security mechanisms to enable error correction to make, see the following example.

Example:

Display the corrective mechanism for a number secured by checksum.

1	**2**	**4**	**5**	**X**	**6**	**0**	**0**	**0**	**0**	**0**	**0**	**6**
1	**2**	**4**	**5**	**X**	**6**	**0**	**0**	**0**	**0**	**0**	**0**	**4**

1+2+4+5+6+0 ..... +0=18 → complement to 10 = **2**, lost number = $(2+6)_{\text{without carry}} = 8$
1+2+4+5+6+0 ..... +0+4=22 → value of lost number is complement to 10 = **8**

*Please enter a numeric check on the correctness of the code and then we can uniquely reconstruct the wrong number. If as the check digit is used a complement to **10**, then summing all valid digits beyond the disturbed digit we get value to the number **10** represents impaired digit (see the second line). If as the check digit is use direct value, the reconstruction of broken digits is more complicated (see first line).*

12.1.1.6 Polynomial codes

Polynomial codes use again the principle of complementarity of control codes to the original information. Creation of additional bits is based on algebraic principles. Selecting generating polynomial properties of the resulting code can be adapted to the nature of errors occurring in the transmission channel or in write and control data on magnetic or optical storage media.

To ensure appropriate polynomial block code we see the data as a polynomial with **k** members, where individual bits represent a block of coefficients and are marked with symbols in the range of a_{k-1}, ……., a_1, a_0. The value of the polynomial is expressed by relation:

$$A(x) = a_{k-1} * x^{k-1} + \ldots\ldots + a_1 * x^1 + a_0 * x^0$$

where value **(k-1)** is a polynomial degree.

Security bits are the remainder of the integer division modulo 2 [*from a theoretical point of view is*

file

the division in finite field $GF_{(2)}$ *of polynomials over the integers*] generating polynomial **P(x)** according to the formula **12.3**, where **Q(x)** is the integer quotient and **R(x)** the integer remainder from integer division. **Generating polynomial P(x)** is the polynomial of degree **r**, which must be less than the value of **k** at the same time greater than the number **0**, i.e. the coefficient a_0 must be **1**.

$$\frac{x^r * A(x)}{P(x)} = Q(x) \oplus \frac{R(x)}{P(x)} \qquad \textbf{12.3}$$

Division modulo 2 by the generating polynomial expressed by relationship 12.3 can be modified, see link 12.4:

$$x^r * A(x) = Q(x) * P(x) \oplus R(x)$$

$$x^r * A(x) \oplus R(x) = Q(x) * P(x) \qquad \textbf{12.4}$$

Note:

Operation in the second row can be made on the basis of equivalence summation and subtraction modulo 2

The left side of equation 12.4 represents a block of data **T(x)** secured by polynomial code. The right side of the equation indicates that it is fully divisible by generating polynomial **P(x)**. This feature allows you to check the integrity of transmitted or read information. On the receiving side is the secured information divided modulo 2 generating polynomial and if the rest is equal to **0**, the information is flawless. The procedure of generating security bits:

- polynomial **A(x)** multiplying by the value x^r, where **r** is the degree of generating polynomial. Practically, this means the polynomial **A(x)** connect **r** right-hand zeros,
- resulting polynomial divided modulo 2 with polynomial **P(x)**
- prepare a secure polynomial **T(x)** - the remainder **R(x)** to connect to the polynomial **A(x)**.

Example:

Demonstration of the properties sum and the product ***modulo 2***

$$(x^2 + x) + (x + 1) = x^2 + 2x + 1 \equiv x^2 + 1$$

$$(x^2 + x) * (x + 1) = x^3 + 2x^2 + x \equiv x^3 + x$$

Likewise the multiplication, where the x^2 *takes after sum* ***modulo 2*** *in the binary scale the value* ***0***

$$\frac{x^3 + x^2 + x}{x + 1} = x^2 + 1 - \frac{1}{x + 1}$$

Division modulo 2 can be modified into a $(x^3 + x^2 + x) = (x^2 + 1) * (x + 1) - 1$. *The rest of the division in this case is value* ***1****.*

The interpretation for quotient in that division is $x^2 + x + 1$ *are input data* ***111****,* $x + 1$ *is the generating polynomial and the rest* ***1*** *is security polynomial. Security level shall be* ***1****. Digit* ***0*** *is added into secured input data and thus obtain a secure polynomial* $x^3 + x^2 + x$.

The procedure will demonstrate the following example.

Example:

Deduce the secure bits for polynomial ***A(x)*** *b with the use of a generating polynomial* ***P(x)****.*

$A(x) = \underline{1010001101} = x^9 + x^7 + x^3 + x^2 + 1$

$P(x) = x^5 + x^4 + x^2 + 1$

$A(x) * x^5 = x^{14} + x^{12} + x^8 + x^7 + x^5 = \underline{1010001101}00000$

$A(x) * x^5/P(x) = x^{14} + x^{12} + x^8 + x^7 + x^5/x^5 + x^4 + x^2 + 1 = 10100011010000 0/110101$

A(x) - no secured data										checking bits					Quotient – Q(x)
1	0	1	0	0	0	1	1	0	1	0	0	0	0	0	
1	0	1	0	0	0										
1	1	0	1	0	1										
0	1	1	1	0	1	1									1
	1	1	0	1	0	1									
	0	0	1	1	1	0	1								1 1
		1	1	0	1	0	1								
		1													1 1 0
			1	1	1	0	1	0							
			1	1	0	1	0	1							
			0	0	1	1	1	1	1						1 1 0 1
				1	1	0	1	0	1						
				1											1 1 0 1 0
					1	1	1	1	1	0					
					1	1	0	1	0	1					
					0	0	1	0	1	1	0				1 1 0 1 0 1
						1	1	0	1	0	1				
						1									1 1 0 1 0 1 0
							1	0	1	1	0	0			
							1	1	0	1	0	1			
							0	1	1	0	0	1	0		1 1 0 1 0 1 0 1
								1	1	0	1	0	1		
								1							1 1 0 1 0 1 0 1 1
									1	1	0	1	0	1	
									1						1 1 0 1 0 1 0 1 1 0
									$R_{(x)}$	0	1	1	1	0	

Result:

$A(x) = x^9 + x^8 + x^6 + x^4 + x^2 + x^1 = 1101010110$, $R(x) = x^3 + x^2 + x^1 = 01110$

Secured polynomial $T(x) = 1010001101 01110$

12.1.1.1.1 Failure detection by the polynomial code

Let the combination of bad bits is a polynomial **E(x)**. Instead the transmitting secure polynomial **T(x)** is in fact received message **T(x)** ⊕ **E(x)**. If this polynomial is flawed fully divisible generating polynomial **P(x)**, then such mistake could not be detected.

If you know the character of possible errors, you can find such a generating polynomial **P(x)** that the probability detection of the selected character error was maximizes.

The following rules apply:

- generating polynomial (at least first-order) provides detects all single-errors
- generating polynomial of the second-order provides detection of all single and double-errors
- when generating polynomial contains a factor **(x +1)**, then provides detection of errors with odd multiplicity. These include polynomial shape $\mathbf{x^c + 1}$ - one is always broken down into a product of the members $\mathbf{(x + 1) * (x^{c-1} + ... + 1)}$.
- Generating polynomial of degree r detects all clusters of errors of length **r-1**.

Note:

*Cluster of errors is a group of **b** bits error polynomial, of which at least the first and last bit is incorrect.*

If the error polynomial is represented by a sequence **0000010100110000**, then the cluster contains

errors of length **b = 7** and the error polynomial can be divided into factors:

$$E(x) = x^{i} * E_{1}(x)$$

In this example, is as follows:

$$x^{10} + x^{8} + x^{5} + x^{4} = x^{4} * (x^{6} + x^{4} + x^{1} + 1)$$

It is clear that the shape factor x^{i} cannot be divisible by generating polynomial, which of course applies to the polynomial x^{4} from the above example.

Example of data transfer without fault:

*The polynomial **T(x) = 1010001101**01110 is the result of the transmission of the polynomial **T(x)** – see previous example – through the transmission line. The integrity check of the received data block **V(x)** is done by dividing the modulo 2 with the same generating polynomial **P(x)**. The value of the integer residue (syndrome) **S(x)** after division is an indicator of the accuracy of the content.*

$$T(x) = 101000110101110 = x^{14} + x^{12} + x^{8} + x^{7} + x^{5} + x^{3} + x^{2} + x$$

$$P(x) = x^{5} + x^{4} + x^{2} + 1$$

	T(x)															Quotient – Q(x)
	1	0	1	0	0	0	1	1	0	1	0	1	1	1	0	
	1	0	1	0	0	0										
	1	1	0	1	0	1										
	0	1	1	1	0	1	1									1
		1	1	0	1	0	1									
		0	0	1	1	1	0	1								1 1
			1	1	0	1	0	1								
			1													1 1 0
				1	1	1	0	1	0							
				1	1	0	1	0	1							
				0	0	1	1	1	1	1						1 1 0 1
					1	1	0	1	0	1						
					1											1 1 0 1 0
						1	1	1	1	1	0					
						1	1	0	1	0	1					
						0	0	1	0	1	1	1				1 1 0 1 0 1
							1	1	0	1	0	1				
							1									1 1 0 1 0 1 0
								1	0	1	1	1	1			
								1	1	0	1	0	1			
								0	1	1	0	1	0	1		1 1 0 1 0 1 0 1
									1	1	0	1	0	1		
									0	0	0	0	0	0	0	1 1 0 1 0 1 0 1 1
										1	1	0	1	0	1	
										1						1 1 0 1 0 1 0 1 1 0
$S_{(x)}$											0	0	0	0	0	

*The integer residue (syndrome) of modulo 2 division is **S(x) = 00000***

Result:

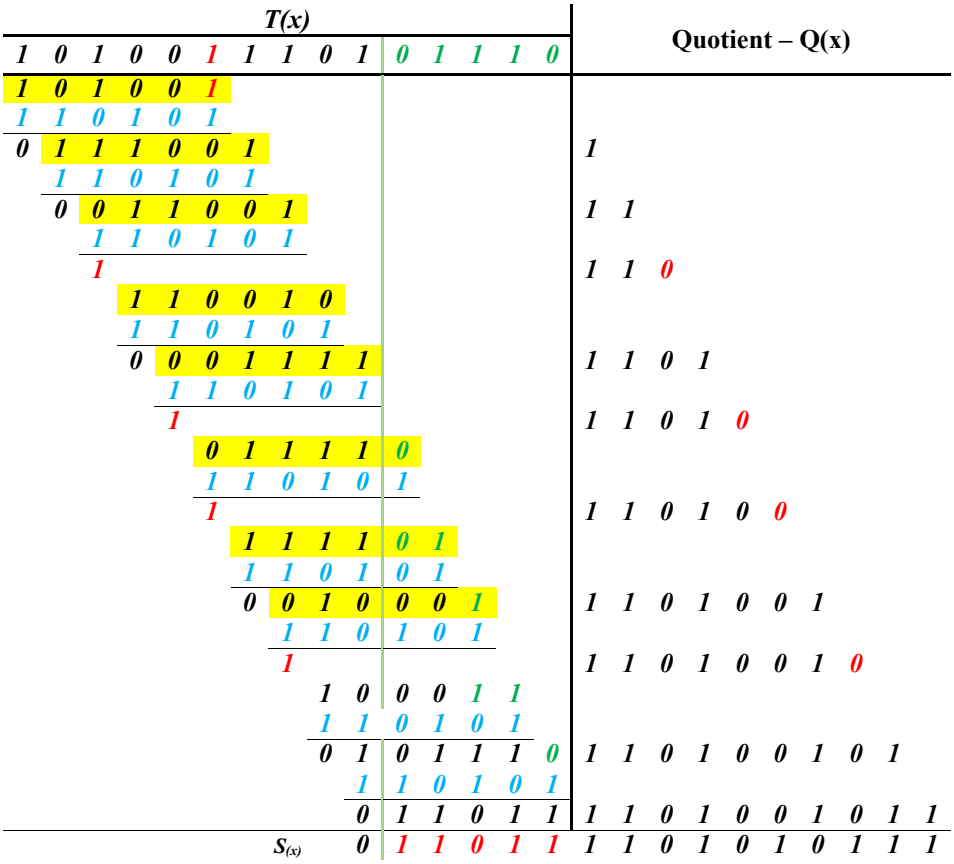
*The remainder (syndrome) of **S(x)** of the polynomial **T(x)** is zero. The polynomial **T(x)** is very likely to match the polynomial **V(x)**. Polynomial **V(x)** was unlikely to be intact.*

Example of data transfer with fault:

The polynomial ***T(x) = 10100111010111*** *is the result of the transmission of the polynomial* ***T(x)*** *– see previous example – through the transmission line. The integrity check of the received data block* ***V(x)*** *is done by dividing the modulo 2 with the same generating polynomial* ***P(x)****. The value of the integer residue (syndrome)* ***S(x)*** *after division is an indicator of the accuracy of the content.*

$$T(x) = 10100111010111 = x^{14} + x^{12} + x^9 + x^8 + x^7 + x^5 + x^3 + x^2 + x$$

$$P(x) = x^5 + x^4 + x^2 + 1$$



The integer residue (syndrome) of modulo 2 division is ***S(x) = 11011***

Result:

The remainder (syndrome) of ***S(x)*** *of the polynomial* ***T(x)*** *is not zero, the content of the transmitted secure polynomial* ***V(x)*** *is violated.*

12.1.1.7 Cyclic redundancy checking - CRC

Cyclic redundancy checks, known as **CRC** (*Cyclic redundancy check in*) is used for protecting data recorded on magnetic and optical media. **CRC** is version of polynomial codes. The difference lies in the use of special polynomial generating types. This is a special case of the hashing function.

CRC is calculated before the registration of data and is stored along with data. After reading the data is again **CRC** independently derived and compared with the stored **CRC**. If the value of generated **CRC** is different, is marked as erroneous reading.

Note:

It some cases it is possible to correct errors using ***CRC****.*

12.1.2 Hardware redundancy

The working principle of the hardware redundancy is similar information redundancy. As pointed out previously, the principle activities of redundancy are to allow the creation of additional symptoms or fail to recognize the correct operation of the reference functional block (not only the electronic equipment). Additional symptoms may be generated by different mechanisms resulting from the nature of the functions carried out in controlled facilities.

For implantation redundancy circuit there are two principles - redundancy (duplication) functional blocks (circuits) and the prediction of important properties of the outcome. Both principles are very often used to control the activities of the basic functional blocks as the digital systems as a whole, as well as structures within the integrated circuits and microprocessors.

12.1.2.1 Doubling of functional blocks with comparing of the outputs

Doubling of functional blocks is based on inspection results or input variables implemented independent of each circuit. Checking the results of two independent realization of a functional block is very dynamic, but because of the circuit performance is used only for the most important functional parts of digital systems.

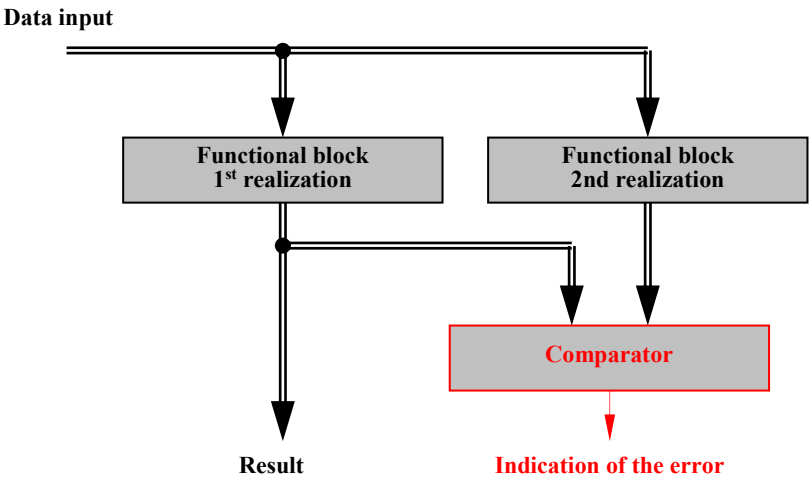


Figure 12.11: Check the correct block operation by the method of comparing the output values

12.1.1.2 Implementation of the inverse function and compare on inputs

Application of duplication method is associated with a significant increase of the technical equipment size. When the realization of the inverse function is much simpler is effective to apply the duplication of functional blocks with control on inputs. This approach is seemingly illogical, but for some applications it is economically and operationally feasible than the standard duplication. Principal block scheme see Figure 12.12.

12.1.2.2 Two-wired logic

This is again a doubling of technical equipment. There are realized direct and inverse functions and the outputs are checked whether both circuits generate mutually inverse values. If the values of both outputs are inverted each other, controlled part works without fault. Checking of the correctness can be realized by nonequivalence circuit that controls the mutual inversion of each functions output.

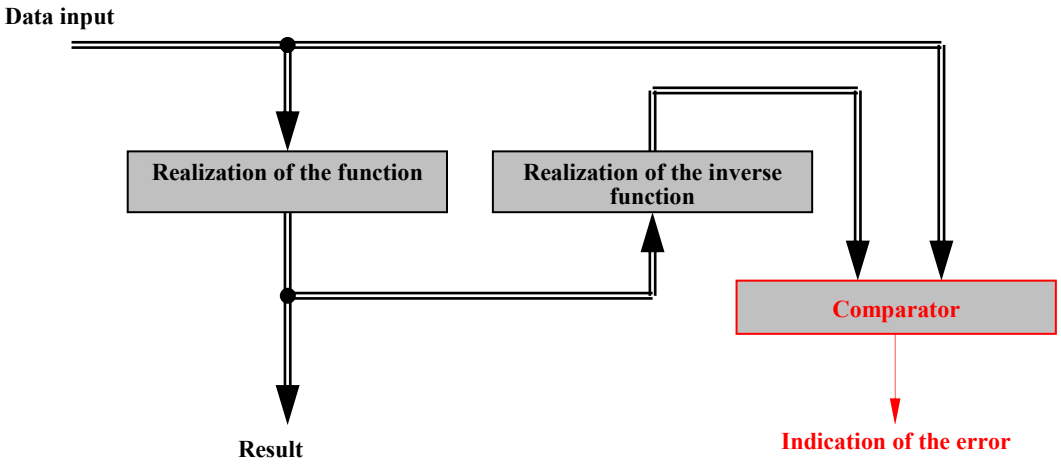


Figure 12.12: Check the correct block operation by the method of comparing the input values

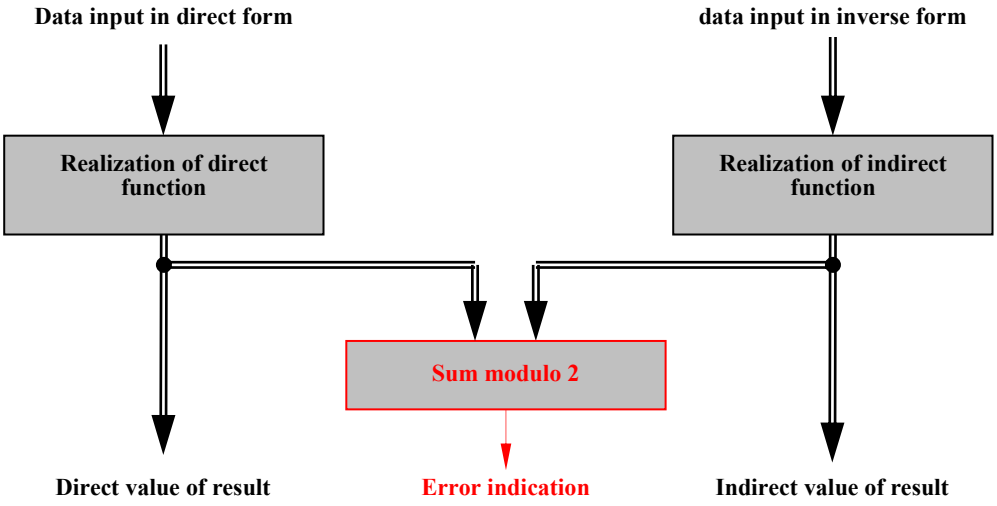


Figure 12.13: Check the correct function by two-wired logic

This design has interesting behaviors. Some logical function can be realized by crossing of the wires. List of basic logical function follow in next picture.

file

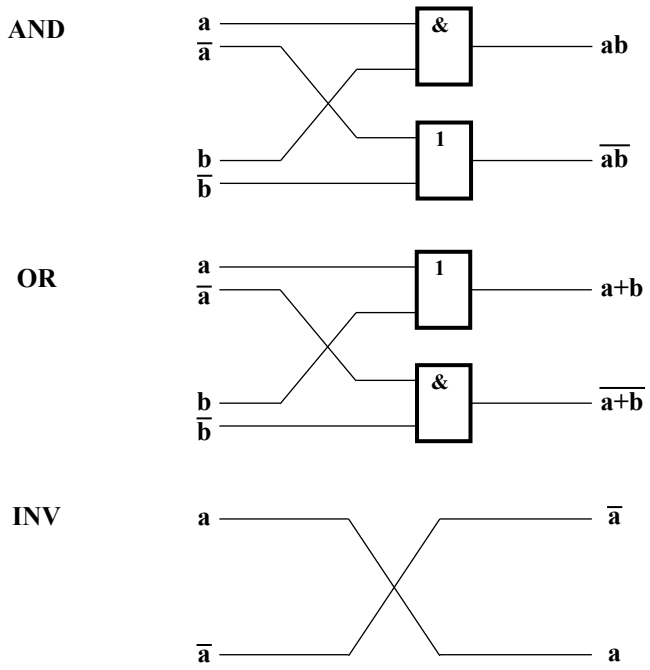


Figure 12.14: Examples of basic logic modules of two-wire logic

12.1.2.3 Four-wired logic (fourfold logic)

This principle of circuit redundancy provides possibility of single-failure self-correction inside in a four-wired logic (hence the name is deriving of this solution verifying the correct functioning by circuit redundancy). This option is given by an extreme size increase of technical equipment. In this solution, each logic signal carried four times and in another layer of logic circuits enters in the same order of replacements. It can be graphically described on the example of bus drivers. There are two basic solutions - two layers of **AND** and **OR** circuits driver with two layers of **NOR** circuits. These two layers represent a fourfold logic gate.

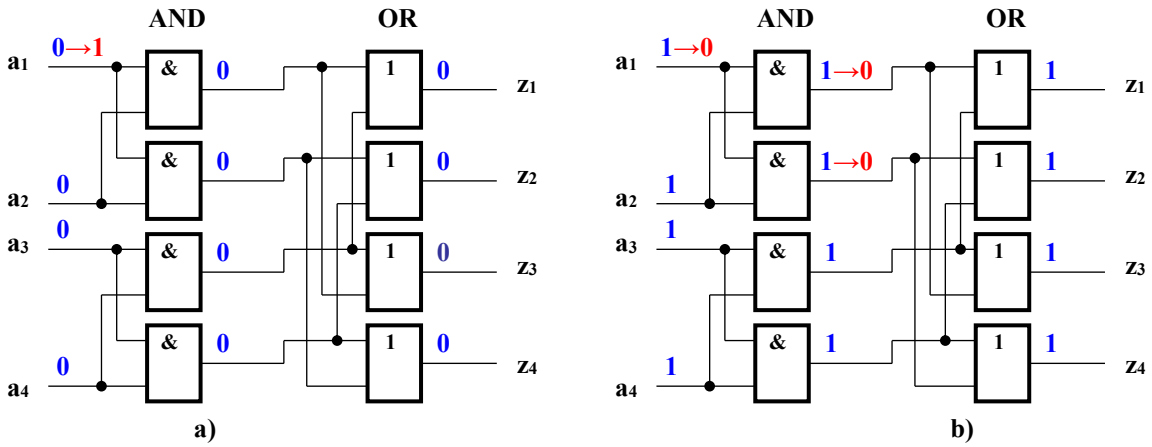


Figure 12.15: Suppression of speech disorders in bus drivers fourfold logic element **AND-OR** type:
a) for t_1 error type
b) for t_0 error type

From the diagram it is clear that the failure t_1 is already filtered out in the first layer of logic, but the failure of the type t_0 in the second layer. In this case, the quadruple logic can correct one failure in a

file

single logical element.

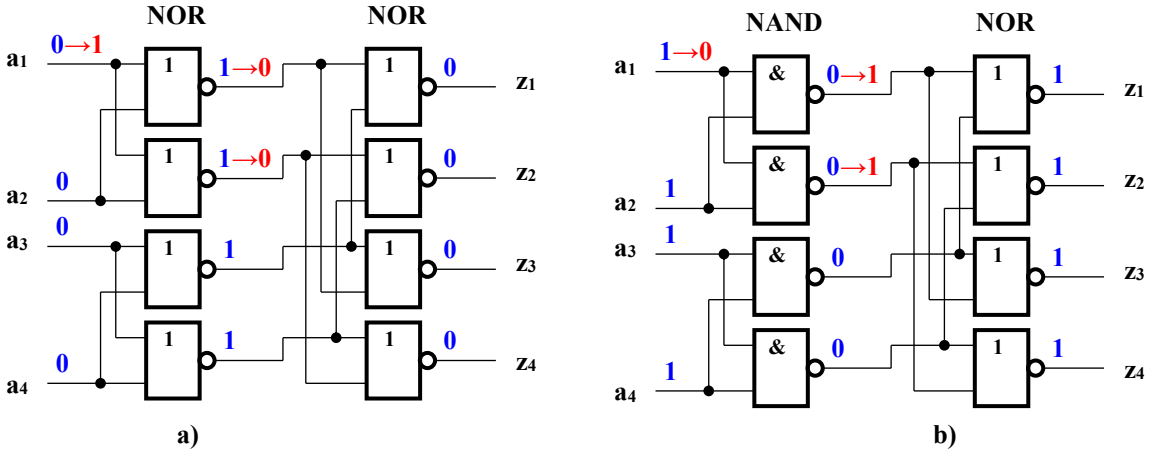


Figure 12.16: Suppression of speech disorders in bus drivers fourfold logic element **NOR-NOR** and **NAND-NOR** types:
a) for $\mathbf{t_1}$ error type
b) for $\mathbf{t_0}$ error type

The fourfold logic **NOR-NOR** and **NAND-NOR** types disorders filters out in reverse order - disorders of the type $\mathbf{t_1}$ are filtered in a second layer of logic and fault type $\mathbf{t_0}$ in the first layer of the logical element.

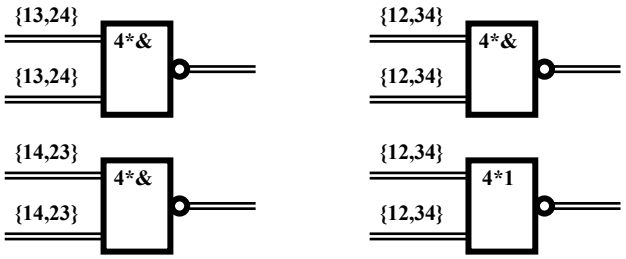


Figure 12.17: Schematic mark of the certain fourfold logic gates

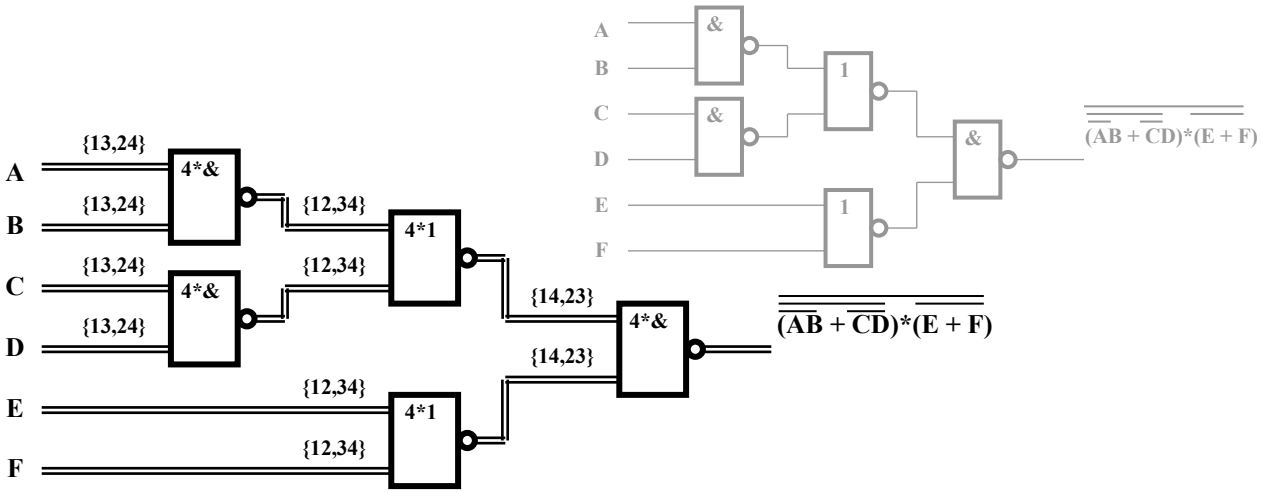


Figure 12.18: Example of the logic circuit realized by the fourfold logic

Similar manner each gate can be realized. Schematic symbol must also include information on the

file

order of confusion within the implementation logic elements. It is important to enable the compilation of logical network elements with homothetic commutations of all logic signals within each layer of the network.

12.1.2.4 Checking the correct operation of the decoders

In this case, actually is a control code. A typical decoder converts the binary number of length **n** bits code **1/N**, where **N** is $\mathbf{2^n}$. To checking of the decoder error-free operation can be used the odd parity check circuit or combinational circuit with the function similar to the **2/N** code check circuit -see chapter "Data Coding" of the Design Digital Systems course.

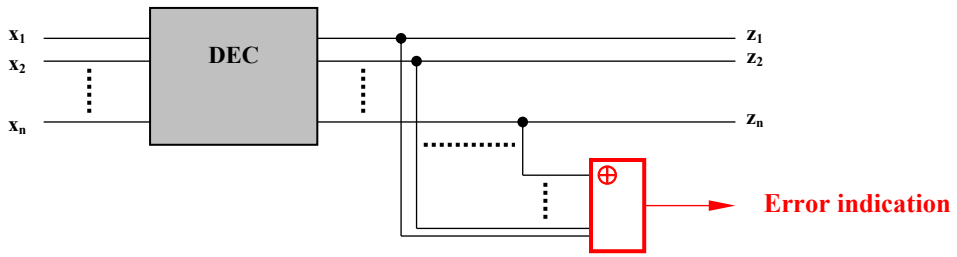


Figure 12.19: The principle of the address decoder valid operation checking by the odd parity circuit

More rigorous checking of the decoder operation is to check the correctness of its output functions. The control circuit is bulky and its dimension is comparable to a simple duplication decoder circuit - see the following example.

Example:

Design checking circuit witch check failure-free operation of the four-bit decoder.

Solution:

Karnaugh map failure-free operation of the address decoder is shown at Table 12.7

$x_2 x_3$ \ $x_0 x_1$	00	01	11	10
00	0	1	0	1
01	1	0	0	0
11	0	0	0	0
10	1	0	0	0

Table 12.7: Karnaugh map of the four-bit decoder valid operation

Logical equation correct functioning four-bit decoder is as follows:

$$\overline{E} = x_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4$$

*Relationship to check of decoder has more **log.0** than **log.1** which implies that the control circuit thought **log.0** could be more efficient - see the following relationship.*

$$E = x_1x_2 + x_3x_4 + x_1x_3 + x_1x_4 + x_2x_3 + x_2x_4 + \overline{x}_1\overline{x}_2\overline{x}_3\overline{x}_4$$

The shape of the two equations suggests that it is not. The equation for the correct operation has four implicants with four variables, while equation malfunction has seven implicants of witch only one is composed of four variables and the other two variables in the direct form.

To test the four-bit decoder is advantageous in terms first equation:

$$\overline{E} = x_1\overline{x}_2\overline{x}_3\overline{x}_4 + \overline{x}_1x_2\overline{x}_3\overline{x}_4 + \overline{x}_1\overline{x}_2x_3\overline{x}_4 + x_1\overline{x}_2\overline{x}_3\overline{x}_4$$

file

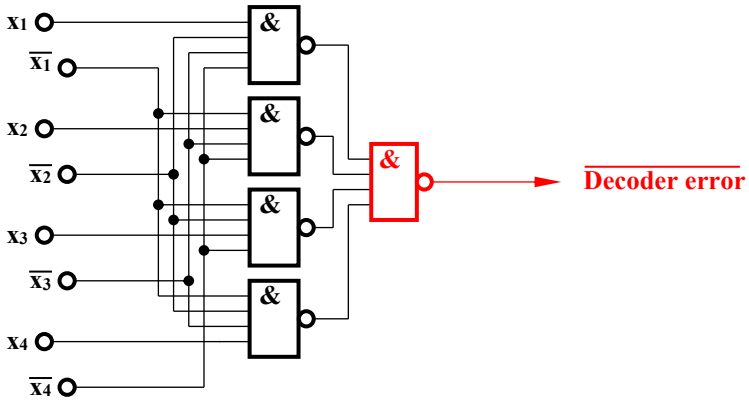


Figure 12.20: Diagram of control circuit for checking four-bit decoder output code

12.1.2.5 Control of register correct operation

In some applications it is necessary to ensure control of fixation important symptom (often realized as a short pulse) in the registry and passed for further processing. Valid values of symptoms is usually limited to frequency stabilize the output signals in the aftermath of the transitional phenomena. These impulses must be safely secured to fix the register.

Checking the entry in the registry uses the principle of similar two-wire logic. Register is to control the registration flags and control the correct functioning of the registry is divided into several sectors of length **n** bits. Each sector is completed by the redundant bit prescribers that the sector does not have a bit, so that each sector of the secured register has **n +1** bits. The result is a situation where in each sector of the secured register is set always at least one bit. If even one bit is not set, it is a failure of logic input register or malfunction of the register. Diagram of the secured register is mentioned in the following figure.

A typical example of a secured register is errors register of the digital systems equipped with recovery block from error. The following example shows the securing procedure of the parity errors registration into the fault register.

Example:

To the error register are written indications of the structural modules control circuit. Valid values of the output signals of the control circuits are time limited to stabilizing the output signals of functional blocks in the aftermath of the transitional phenomena. These impulses must fix into error register. Later is from the contents of the error register shaped the interrupt requests signal from the control circuits and also the so-called code of failure. Formation of fault code is complicated by fact that fault indication is often indicated from several check circuits and is necessary to found the primary indication of the failure.

*If we consider the content control of the 32-bits register **RX** (this may be the input register of the adder) is obvious, that consists of **4** Bytes, each of which is protected by an odd parity bit. However, when an error changes one Byte, the parity circuit sets the failure signal of the consonant Byte and also resets the accuracy signal of the consonant Byte (see Picture 12.9). At the same time is generated the parity error signal of the register **RX**, which is the summation of Byte failure signals and the accuracy signal of the register **RX** contents (the summation of the accuracy signal of all Bytes content). Signals Byte accuracy parity and the signal accuracy of the register **RX** content are fed to the input logic of the error register. Here are generated adjustment functions Bytes error in the relevant error register bits, which are next functions:*

file

- $\overline{error\ RX_i} * error\ RX$
- $\overline{error\ RX_i} * \overline{error\ RX}$
- $error\ RX_i * \overline{error\ RX}$
- $\overline{error\ RX_i} * \overline{error\ RX}$ *(function of the register **RX** content correctness)*
- *indication of the write error into error register.*

Diagram of the error register input circuits and its security is shown in the picture – see Figure 12.21.

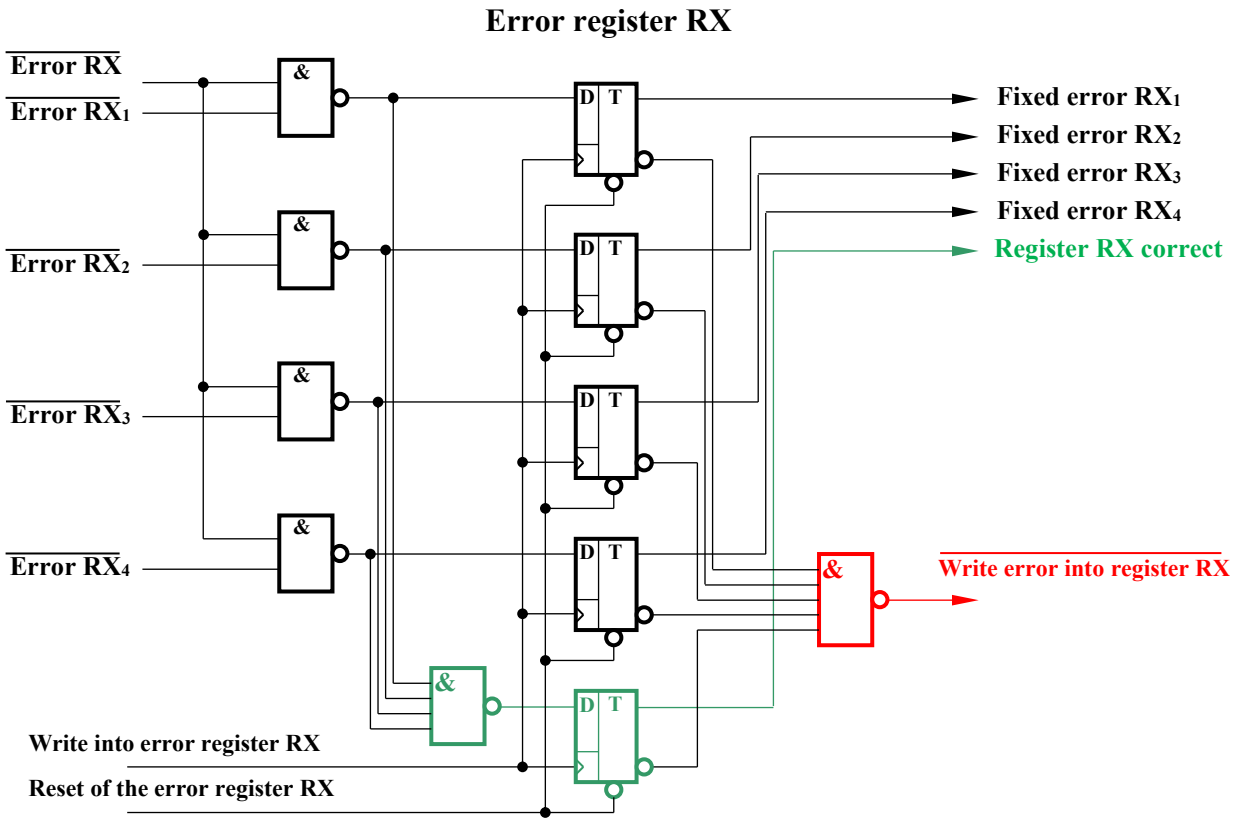


Figure 12.21: Funkcion principle of the checking operation write into the register

Note:

In both diagrams of this chapter is highlighted in green forming part of the information absence in the register and blue circuit forming the register write error.

12.1.3 Prediction

This is another continuous diagnostics method of the digital system operability. This method uses a prediction of characteristic symptom of result on the basis of some properties of input variables. It is possible to predict the outcome of parity or parity inversion results. With this procedure can be found at the testing of the binary counter, or testing of the arithmetic circuits.

12.1.3.1 Checking of the binary counters operation

For the binary counters are characteristic, that the parity of the result is inverted when the carry or loan ends on an even rank of the counter. This feature applies to both binary counters - forward and back. This feature is demonstrated in the following table for odd parity.

From Table 12.9 can be derived logical equations for predicting the inversion parity resulting plus or

minus of value 1.

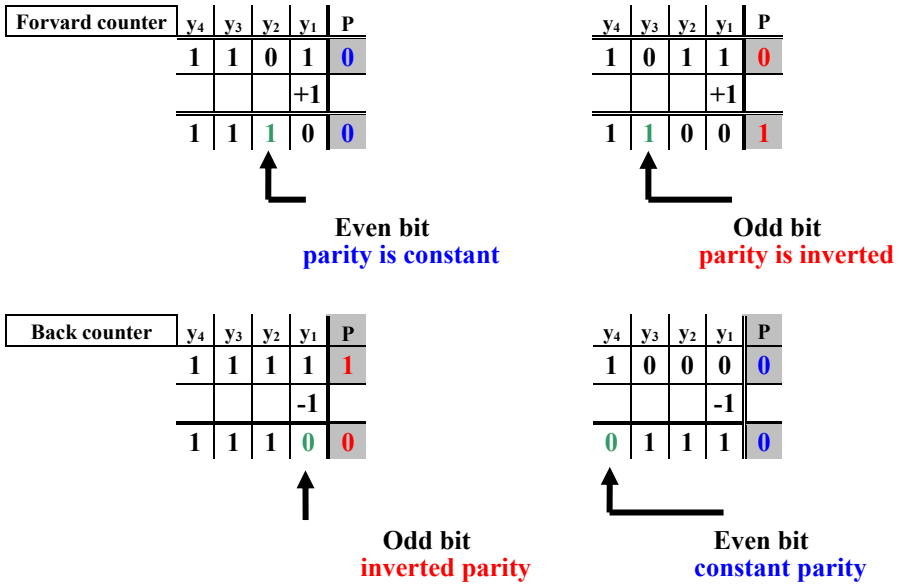


Table 12.8: Principle of the parity inversion prediction

y_4	y_3	y_2	y_1	P	Parity inversion - forward counter	Parity inversion - back counter
0	0	0	0	1	*	
0	0	0	1	0		*
0	0	1	0	0	*	
0	0	1	1	1	*	*
0	1	0	0	0	*	*
0	1	0	1	1		*
0	1	1	0	1	*	
0	1	1	1	0		*
1	0	0	0	0	*	
1	0	0	1	1		*
1	0	1	0	1	*	
1	0	1	1	0	*	*
1	1	0	0	1	*	*
1	1	0	1	0		*
1	1	1	0	0	*	
1	1	1	1	1		*

Table 12.9: The logic derivation of equations for predicting parity inversion in dependence on carry or loan bits

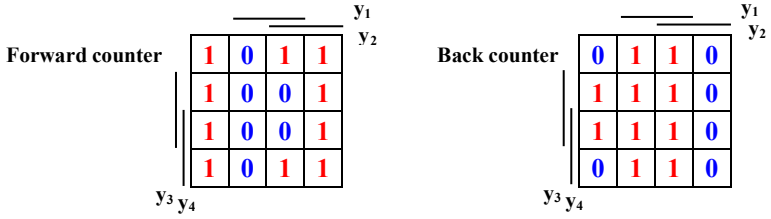


Table 12.10: Karnaugh maps of the functions for result parity inversion prediction

file

For binary forward counter is valid the following equation of the prediction:

- result parity is inverted $\overline{y_1} + y_2 \overline{y_3}$
- result parity is constant $y_1 \overline{y_2} + y_1 y_3$

For binary back counter is valid the following equation of the prediction:

- result parity is inverted $y_1 + \overline{y_2} y_3$
- result parity is constant $\overline{y_1} y_2 + \overline{y_1}\ \overline{y_3}$

Check the correct function of the binary counter is based on comparing the predicted parity of content in the current step corresponds with parity of result in the next step. Predicted parity is derived from parity of counter content in the current step and the value of prediction function. If the predictive value of the function becomes **log.1**, the parity of the counter content in the next step will invert in compare with parity of the counter content in the current state. If the predictive value of the function becomes **log.0**, the parity of the counters content in the next step will be the same as parity of the counter content in the current state. Predicted parity of the counter content is stored in the flip-flop and in the next step is compared with the actual parity of the counters content. Diagram of control circuit working on this principle is shown in the picture below.

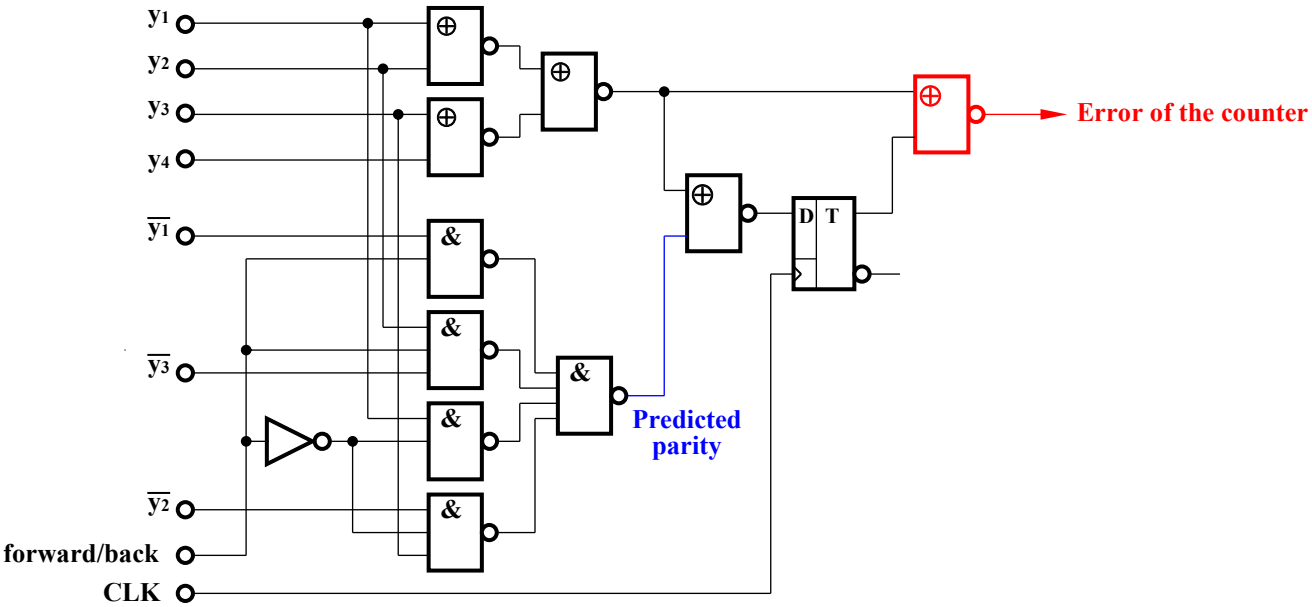


Figure 12.22: Diagram of bidirectional binary counter control circuit based on prediction of parity content in the following step

Note:

In addition, you can deduce the other rules - see below:

- *for forward counter - the parity bit inverts if odd bit is equal to **log.0** and all lower bits are **log.1**,*
- *for back counter - the parity bit inverts if even bit is equal to **log.1** and all lower bits are **log.0**.*

12.1.3.2 Adder activity checking

Prediction mechanism for checking the correct operation can be applied also in adder. The basic principle of control is the fact that if add two even or two odd numbers the result is an even number. If add even and odd number, we get an odd number.

Applying this principle of control is to ensure parity of both operands. The classic parallel adder works with words containing several bytes secured by parity. It causes partial complication in

predicting process.

Generated parity of result contains parity bits of result bytes. Prediction parity of result consists of prediction parity bits of input bytes. Checking of the valid operation is based on comparing of the prediction parity bits and generated parity bits of the result.

In the final in parity bit of byte prediction entering homothetic low significant bits of bytes and the carry bit of the sum of lower homothetic meaningful input bytes. Prediction of carry bits over a group of adder ranks - see chapter devoted to adder design (Design of the Digital System course).

The equation for generating of the predicted parity bit for one result byte of the adder is as follows:

$$p_i = a_{i0} \oplus b_{i0} \oplus carry_{i-1}$$

Scheme of the predictor is very simple and follows on next picture.

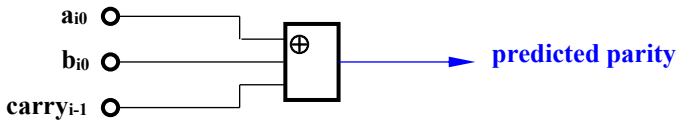


Figure 12.23: Diagram of adder parity predictor

Correct adder operation is indicated by matching of all bytes of result parity bits with all predicated parity bits of the bytes of result.

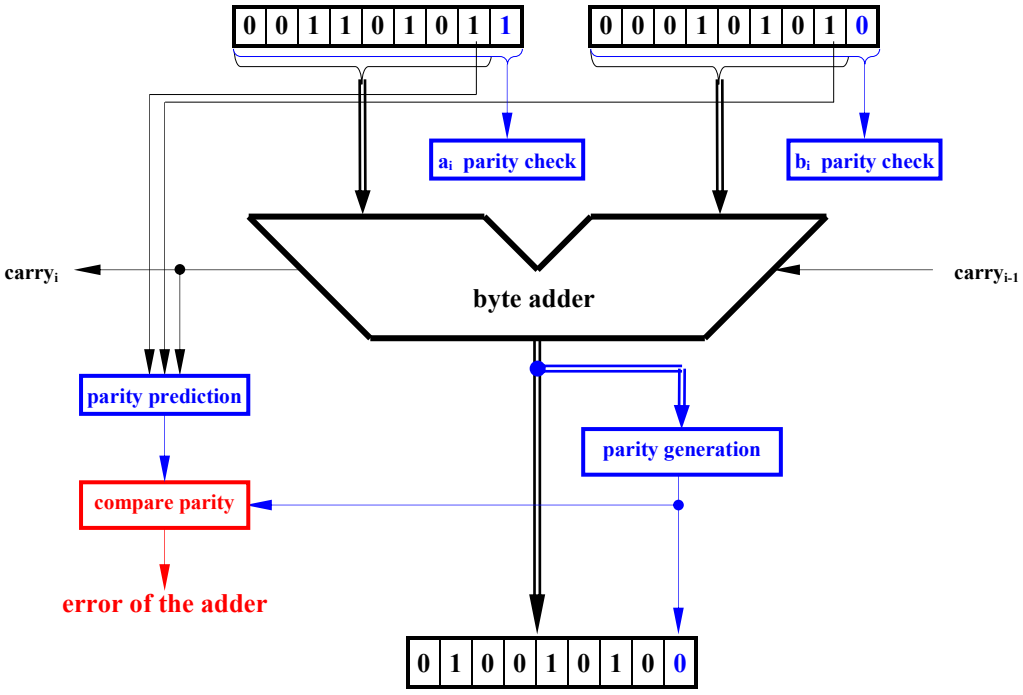


Figure 12.24: Scheme of the adder check circuit

12.1.4 Checking of the time sequences

This principle is based on monitoring of the requirement for two parameters. First is a defined period of maximum response from addressed subsystem. Second is the defined period of repeated operation

file

within a beforehand defined "time window". Among the frequently used methods include the so-called counter-TIME OUT and counter-WATCH DOG.

To implement the above functions is used universal counter, which is part of the auxiliary circuits most commonly manufactured microprocessors. These processors are usually equipped with multi-purpose counters are used for different purposes. The universal counter can be filled by optional initial value and you can select the frequency of synchronization pulses. Most universal counter is equipped with an indication of zero content, and this indication can be used to activate of an interrupt system.

12.1.4.1 Watch-dog mechanism

The universal counter can be used in WATCH-DOG function. Some microcontrollers are equipped with single-function counter WATCH-DOG. The principle of operation counters WATCH-DOG is based on measuring of the maximum permitted duration of selected activities. For an implementation of WATCH-DOG is a control of universal counter adjusted so that the initial content of the counter and its frequency synchronization is setup so that the time needed to reach zero is longer than the maximum period for re-repetition of selected activities. This time is chosen margin longer than duration of selected activity.

Content of the universal counter is by start indication of selected activities constantly put into default status. If the observed activity starts late or not at all content of the counter reaches zero and initializes an interrupt system that can restart the reported activity and possibly send an error report about the non-standard reference activity. Described above story is captured following time chart.

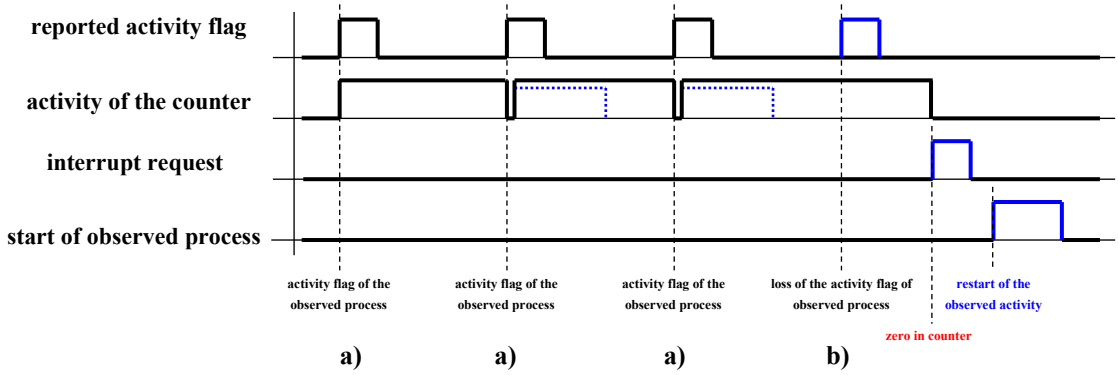


Figure 12.25: Time diagram of the **WATCH-DOG** operation
a) proper function of the monitored process
b) malfunction of the monitored process

12.1.4.2 Time-out mechanism

Feature of counter-OUT TIME is measuring the maximum time of an observed operation. The operation is running **START** command issuing by **MASTER** process of the operation (mostly program started in processor). Carrying out an operation then announces **SLAVE** operation (caused by **SLAVE** process) by signal **STOP**. Time between the **START** command and **STOP** signal from a properly functioning system, it is reasonably accurately estimated. If the process **SLAVE** does not emit signal **STOP** in a predetermined time, it is clear that the running process is not proceeding according to the assumptions and needs to be stopped by intervention of the **MASTER** process. The process of measuring the duration of **SLAVE** process can efficiently use one of the currently unused universal counters in timer mode.

For implementation of TIME-OUT function is the control of a universal counter adjusted so that the

file

initial content of the counter and its synchronization frequency is setup so that the time needed to reach zero is margin longer than the maximum length of a **SLAVE** process. Universal counter in the function TIME-OUT works so that the initial content is set by the **START** command and synchronization is pulled by **STOP** command. If the **STOP** command is late or not at all generated, the contents of the counters reaches zero and as a result of which is initialized interrupt system. The interrupt process stops waiting for end of **SLAVE** process and issues an error message about non-standard termination of the **SLAVE** process. Described above story is captured following time chart.

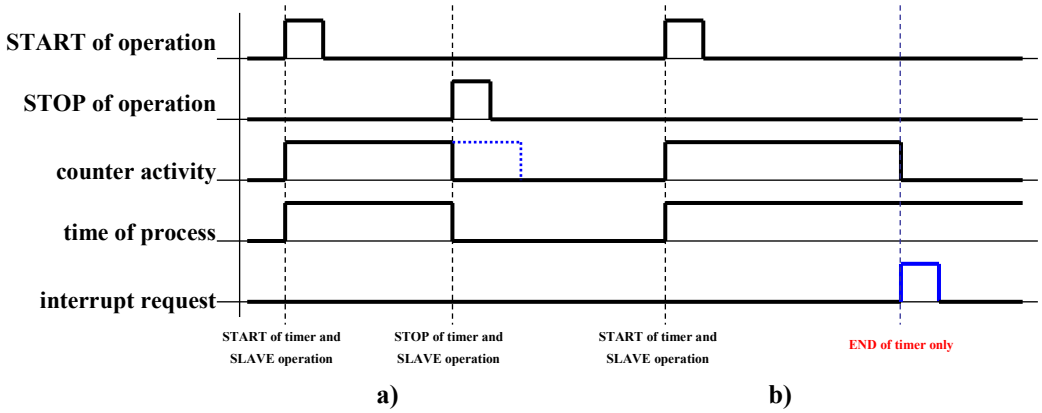


Figure 12.26: Time diagram of the **TIME-OUT** timer operation
a) correct function of the **SLAVE** process
b) malfunctions of the **SLAVE** process

Note:

Typical example of the counter TIME-OUT implementation is the driver of the parallel port printer of classic PC. ***START*** *signal of operation constitutes a signal* ***STROBE OUT*** *and signal to stop operation is a signal* ***ACK*** *of a printer. If the printer is turned off, or is unable to print a variety of reasons, omit to send the signal* ***ACK*** *and if interrupt system is activated is on the LCD screen given an error message (depending on your operating system, usually in the form of an information window).*

file

List of figures

List of tables

file