Fault-tolerant Software Solutions in Microcontroller Based Systems

György Györök, Bertalan Beszédes
* Óbuda University/ Alba Regia Technical Faculty, Székesfehérvár, Hungary
e-mail: {gyorok.gyorgy, bertalan.beszedes}@amk.uni-obuda.hu

***Abstract*—In this article, the focus is on the different kind of software solutions, how can a microcontroller increase the fault-tolerance level of the embedded system. At the beginning, it will be shown, the theoretical base connection between fault, error and failure, the possible causes of faults, fault tolerant solutions and fault tolerant software solutions. In the realization part, it will be shown, how the error-free running time and error detecting features can be increase the robustness of the system.**

I. INTRODUCTION

Fault-masking architectures can be classified into mainly hardware or software categories. Of course, neither can stand alone. It is containing mainly the type of the solution, which is in its name. The duplication of frequently failing units (typically, power supply unit [1]) is the most common way to realize a hardware redundancy [2]. In software solutions, there are the multiple execution, the multiple measurements [3] and the majority voters as the most common major categories.

From the foregoing, it seems, depending to what kind of redundancy had been used, it has significantly impact to system performance, required power, weight, price and reliability. It is important to review the various methods to assess – the perspective of – the possibilities how to increase the reliability.

But first, let us declare the exact meanings the three basic concepts, fault, error and failure. These concepts are connected through causes and effects. The fault causes an error, which is causes failure.

A. Fault, error and failure

The fault is the errors proven or suspected cause. In case of a hardware component it could be short circuit, connection cut or parameter changes listed here, while in software case, it could be unexpected input combination, staying in an endless loop, make an addressing mistake, to mentioning only a few. An example, during manufacturing an AND gate, and the surface of the semiconductor had been polluted by a micro-sized dust particle, the AND gate's input may stay in high logic level.

The error – caused by the fault – is already appears the internal state of the device. [4] For example, if the AND gate's input gets a high logic level input voltage, the input signal is the same as the stacked leg's signal. If the input signal changing – gets a low logic level – the change is no longer transmitted through the input drive, and the AND gate's output will not change. That will cause an error in the system.

A failure occurs, when the error gets out of the system's output. The gate's output did not enforce in the logical function, so it affected for the system's output signals. Thereby, the error gets out to the outside world and become a failure. [5].

The three mentioned type of errors, appears in three different level (Fig. 1.). The fault is inherently physical, the error is modifying the internal state of the system, it's informational nature, and failure essentially affect to the outside world.

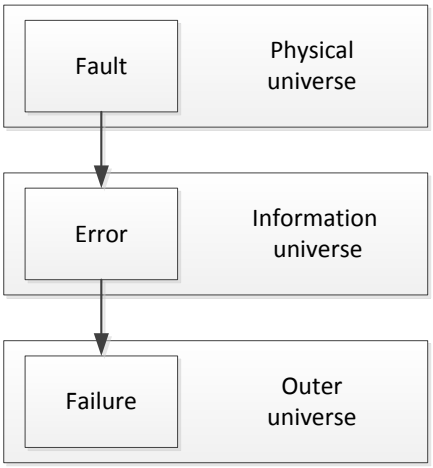

Figure 1. Three-universe

B. Causes of faults

The developing process of a device is starting with the specification phase. If this is not successful, that will cause a failure conception.

The formation of errors can be traced back to several things, like external interference or the consequences of the mistakes made during the design of the system or a component. The fault will not come to the surface, even during the examination of the final conformity test of the finished product. The error only turns out during the installation, operation and appears as a failure.

The topic of specification mistakes includes for example not correct timing, conversion, leveling, etc. between hardware or software modules.

The implementation mistakes may occurs when specification is implemented into practice. Improper or wrong component selection, not correct planning decisions or mistakes in coding may be the root cause.

The component imperfection is the most common source of fault. None of the components – neither from the same type – are matching perfectly, because their

parameters will may vary slightly. This can be easily remedied with conscious design or with component selection, however, the problems related to random component failures are much more significant. Typical causes are, for example, in case of microelectronic components, – within the case – the rupture of the bonds; metal corrosion; in case of electronic PCB's, some manufacturing imperfections or changes in the operating conditions – operating in extreme conditions. The failure type of component imperfection is also including the failure due to aging of the parts.

Strong emphasis should be placed on the control of external interferences. The foreseeing planning can give the ability to control these unpredictable effects. It can be classified into external interferences for example, the electromagnetic interference, radiation, the mistake of the operator, the result of a physical injury or environmental extremity (vibration, temperature, dust, humidity, …), etc.

C. Nature of faults

If it could understand better the root causes of faults, then it could be developed improved procedures to prevent their formation. But until this, it need to be intervened after the appearance of an error, to maintain the operation of the system. [6]

As a first step, it is needed to know the types of faults:

- Source of faults:
  - Specification mistakes
  - Implementation mistakes
  - External interference
  - Component imperfection
- Type of faults:
  - Software
  - Hardware
    - Analog
    - Digital
- Duration of fault:
  - Permanent
  - Sporadic
  - Transient
- Expense of fault:
  - Local
  - Global
- Value of the fault:
  - Determined
  - Not determined

Very many fault combinations can be imagined based on the upper – broadly – classification. It is not expected to give a proper global solution for the problems. As a result, many theories have been advanced for the treatment of certain causes of errors.

The discussion of these is beyond the scope of the article, but it will be appreciated that, it needs to correct the relevant faults. After the exploration of the fault possibilities, it need to analyze the probability of the fault, and its consequences, and the resources spent for the troubleshooting.

It is practical – if the conditions allowing it – to choose the minimal hardware-intensive solutions, and prefer the software solutions, especially in embedded systems.

D. Propagation of faults

Fault tolerance, fault avoidance, fault prevention and fault masking is also a constructive technic, which can increase the reliability of the devices. Fig. 2. shows the application areas of the mentioned techniques.

Fault avoidance, fault prevention is the first defense line to prevent the formation of faults. Several techniques can be used, to increase the quality of the used components and the applied technologies. The method is characterized in that the disorder can be treated even before the formation. For example: plan criticism by an outside expert, using design practices to increasing reliability, component selection, oversizing, testing, shielding, and other methods to improve quality.

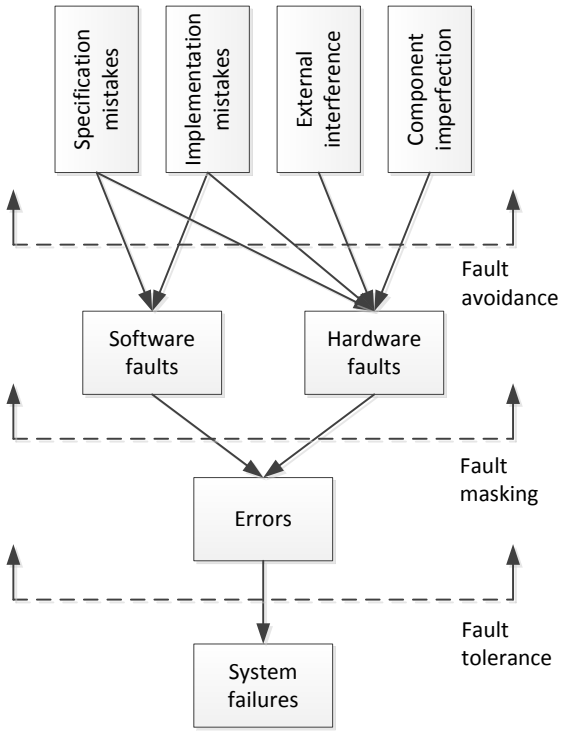

Figure 2. Preventing the spread of errors

The fault masking techniques are about to protect the system from the evolution of faults become an error. [7] When the fault has already occurred, fault masking is try to eliminate the fault's effect from the system – it does not let the fault to step out from the physical universe. A typical solution for a fault-masking system, is based on majority voting, where several autonomous decisions are made, and the result is given by the majority of voters. In this example, if one of the participants generates incorrect outcome, it becomes filter out, for this, it just need to be compared with the results of the other participants. Thus, the fault does not cause an error in the results.

The purpose of fault tolerance is to avoid faults, if the fault evolved an error. In this case fault masking and/or reconfiguration can be used. It can detect the error, then find out the source of it and the defective item will be

removed from the system and might set into operation a new one. [8]

II. FAULT-TOLERANT SOLUTIONS

At the beginning of the fault tolerance history, the usage of redundancy is always limited to physical hardware solutions. The most common solution to realize fault tolerance was to multiply the physical parts of the device, but nowadays we have more sophisticated solutions [9]. A redundant system – compared to simple system – have added information, resources or time.

The following types of redundancies are available:

- Hardware redundancy is when extra hardware is added to the system, it is typically used for fault detection and for fault tolerance. For example, multiplication of modules.
- Software redundancy means that, the added extra software modules, giving the possibilities to detect the faults – if possible, fix them –, next to the default functions of the original software. For example, timeout monitoring assigned to waiting's.
- Information redundancy is the extra information what is used to fault detection - if possible, fault correction – which would not be necessary for the default functions of the device. For example, using error correcting bits.
- Time redundancy is the extra time what is used to fault detection and fault tolerance features. For example, running identical calculations multiple times and checking the consistency of the outcome.

Whichever type of redundancy is had been used, the costs will rise. When choosing hardware redundancy extra parts are required, also the power needs, the size of the device, and the cost of the development will increase parallel [10]. If we using software redundancy, it also has some effect to hardware redundancy, because we need stronger processor and bigger memory capacity, more time in developing phase, and so on. [11]

III. SOFTWARE REDUNDANCY

Reliability can be increased by increasing the number of the software segments. It need to be compare – with the proper algorithm – these redundant software segments, and calculate/chose the right result as a local output. This result will be one of the input parameter of the next software module. [12]

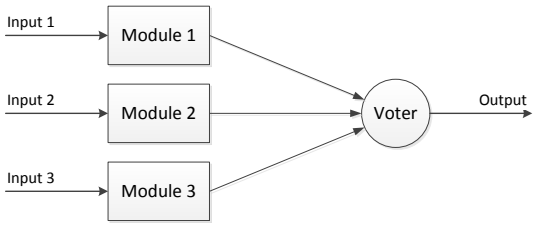

Figure 3. Triple modular redundancy

A. Majority voters

Majority voters need to have at least three different inputs. If two of the inputs are working properly, the voter will give a correct result. So, the method can tolerate only one malfunction. If more than one of the inputs gets a false signal, the output of the majority voter will be incorrect (see at Fig. 3.).

In software, there could run three different software method in parallel. The results of these software methods need to be compared by the majority voter. [13]

Majority voters are simple modules both is software and hardware realization. Therefore, it has a fairly high reliability compared with the other system modules. But, if the voter gets out of order – single point of failure – the whole systems operation becomes impossible. A solution could be, if the voters are tripled, as showed in Fig. 4. By converting functions as a sequence of sequential steps, and by incorporating voters between each level, the reliability of the system can be increased significantly [14]. This way it is possible to stop the error near to the appearance of the error, so it will not spread out to the other parts of the system. Thanks to this, the system can tolerate multiple errors if, they are appearing in different levels.

B. Software solutions

The advantage of the software solution – compared to the hardware solution – is the flexibility, less parts demand (against with a 32-bit long hardware voter), resulting in lower consumption and cost. Although, the algorithm requires only a small computing capacity, but it is slower than the dedicated hardware, and in addition, in

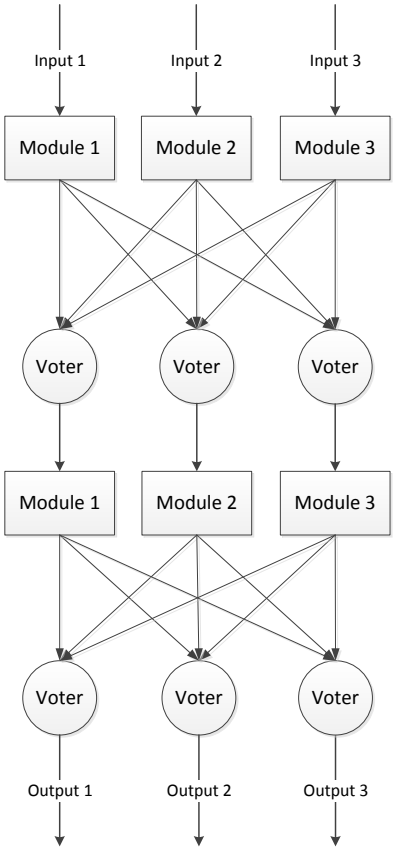

Figure 4. Sequential triple modular redundancy

the case of independent systems, synchronization problems need to be solved. [15]

C. Redundant measurements

However, many times (such as outputs of sensors), the values are correct, but they are not exactly the same, they differ within their accuracy. In a software solution makes it easy to produce the correct output. [16] A delta deviation is allowed for the measured values. If the measured values are within the delta range (relative to each other), it does not count as an error. But, in the case of multiple parallel voters, it is necessary to ensure that each of the voting outputs is exactly the same (bits are exactly the same).

If delta tolerance is selected according to the powers of two, then this method can be used for both hardware and software voters if the LSB bits are omitted from the comparison – with masking or shifting right the measured values. [17]

Mean value voter gives a different solution for the above-mentioned problem. It is providing the best result for multiple – even with significantly different – inputs. [18] As shown in Figure 5., the voter is selecting the middle value. As long as, two signs out of three are correct, the voter always choose the correct signal. The principle can be applied to any voter with an odd number of inputs.

In some cases, it makes sense to determine the output value as a function of the input values. For example, if not the middle value had been chosen – like in the mean value voter – but calculates the currently expected output signal based on the values of the inputs recorded at previous times. [19]

IV. Realization

If we use only one actuator, we cannot duplicate the voters. Therefore, if possible and the nature of the process permits, several interveners and a sufficient number of voters should be used.

In our solution three digital temperature sensors output values are used as the input signals of the system, three voters had been applied, and three fan used as actuators. It is shown in Fig. 6 and Fig. 7. Fig. 8. shows the flowchart of the demonstration project.

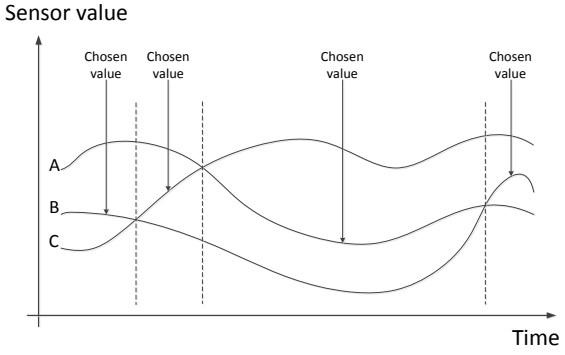

Figure 5. Technological voter

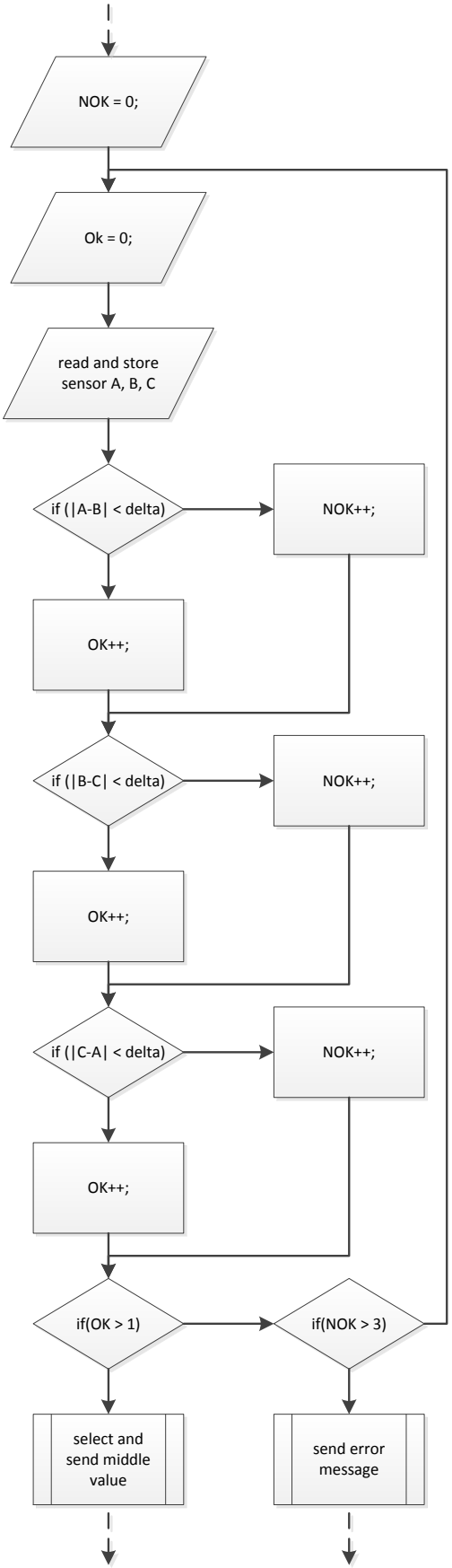

Figure 8. Flowchart of multiple majority voters

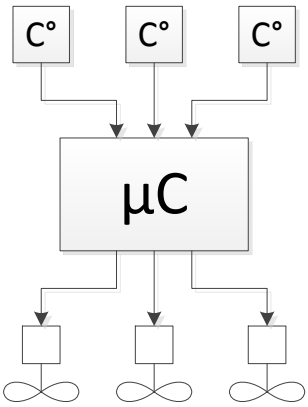

Figure 6. Operating plan of the multiple majority voter

Another solution – to show a robust solution – is if the voting circuit had been left, and the interconnected system of several interveners are used, which also performs the voting task, in addition to the process control. This is called as a technology voter (Fig. 9). The voting takes place by serial parallel coupling of six FET's. Technology voting does not only have the advantages of increasing reliability due to the lack of a voting circuit, but it can also be used to replicate the interveners and to deal with errors in them.

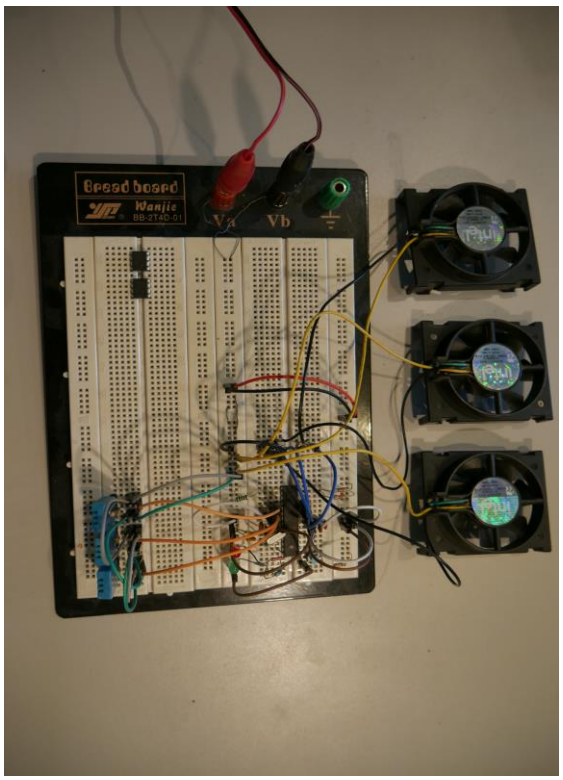

Figure 7. Realization of the multiple majority voter

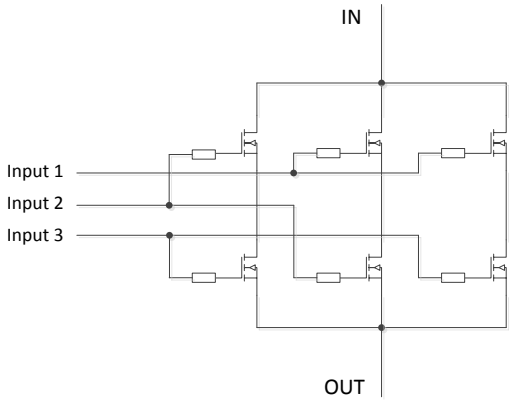

Figure 9. Technological voter

V. CONCLUSION

In this paper, is showed the theoretical base connection between fault, error and failure, the possible causes of faults, fault tolerant solutions and fault tolerant software solutions. In a realization, it had been shown, a redundant software block based system, with a reliable measurement algorithm. By the mentioned solutions, the error-free running time and error detecting features can be increased, as showed in the implementation of the system. We believe that the presented methods can be used in several applications.

REFERENCES

[1] Gy. Györök. A-class amplifier with FPAA as a predictive supply voltage control. In: 9th International Symposium of Hungarian Researcherson Computational Intelligence and Informatics (CINTI2008). 2008. 361–368. p.

[2] György Györök, Bertalan Beszedes. Fault tolerant power supply systems In: Orosz Gábor Tamás 11th International Symposium on Applied Informatics and Related Areas (AIS 2016). Székesfehérvár, Magyarország, Budapest: Óbudai Egyetem, 2016. pp. 68-73. (ISBN:978-615-5460-92-0)

[3] Györök György, Beszédes Bertalan. Duplicated Control Unit Based Embedded Fault-masking Systems. In: Szakál Anikó IEEE 15th International Symposium on Intelligent Systems and Informatics : (SISY 2017) Óbudai Egyetem. Szabadka, Szerbia. 2017. pp. 1-6. (ISBN:978-1-5386-3855-2)

[4] Bray W. Johnson. Design and Analysis of Fault-Tolerant Didital Systems. 1989. Addison-Wesley Publishing

[5] Gy. Györök. Embedded hybrid controller with programmable analog circuit. In: Intelligent Engineering Systems (INES), 2010 14th International Conference on. IEEE, 2010.

[6] K. Lamár, J. Neszveda. Average probability of failure of aperiodically operated devices. In: Acta Polytechnica Hungarica, 10.(8.). 2013. 153–167. p.

[7] Gy Györök, T Orosz, M Makó, T Treiber To Achieve Circuit Robustness by Co-operation of FPAA and Embedded Microcontroller In: Szakál Anikó (szerk.) IEEE 8th International Symposium on Applied Computational Intelligence and Informatics: SACI 2013. Konferencia helye, ideje: Timisoara, Románia, 2013.05.23-2013.05.25. (IEEE) New York: IEEE, 2013. pp. 315-320. (ISBN:978-1-4673-6397-6)

[8] Gy. Györök. The FPAA realization of analog robust electronic circuit. In: Computational Cybernetics, 2009.

ICCC 2009. IEEE International Conference on. IEEE, 2009.

[9] Gy Györök Embedded hybrid controller with programmable analog circuit In: Szakál A (szerk.) 14th International Conference on Intelligent Engineering Systems: Proceedings. Konferencia helye, ideje: Las Palmas, Spanyolország, 2010.05.05-2010.05.07. Budapest: IEEE Hungary Section, 2010. pp. 1-4. (ISBN:978-1-4244-7651-0)

[10] György Györök, Bertalan Beszédes. Artificial Education Process Environment for Embedded Systems In: Orosz Gábor Tamás (szerk.) 9th International Symposium on Applied Informatics and Related Areas - AIS2014. Konferencia helye, ideje: Székesfehérvár, Magyarország, 2014.11.12 Székesfehérvár: Óbudai Egyetem, 2014. pp. 37-42. (ISBN:978-615-5460-21-0)

[11] J. Kopják J. Kovács. Compering event-driven program models used in embedded systems. In: Automotive-Entwicklungen und Technologien. 2011. 90-95. p.

[12] J. Kopják. Dynamic analysis of distributed control network based on event driven software gates. In: IEEE 11th International Symposium on Intelligent Systems and Informatics. Subotica. Serbia. 2013. ISBN: 978-1-4673-4751-8. 293–297. p.

[13] J. Kopják, J. Kovács. Implementation of event driven software gates for combinational logic networks. In: IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics. Subotica, Serbia. 2012. ISBN: 978-1-4673-4751-8. 299–304 p.

[14] Gy Györök, M Seebauer, T Orosz, M Makó, A Selmeci Multiprocessor Application in Embedded Control System In: Szakál A (szerk.) 2012 IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics, SISY 2012, Subotica, 2012, September, 20-22. Konferencia helye, ideje: Subotica, Szerbia, 2012.09.20-2012.09.22. Piscataway: IEEE, 2012. pp. 305-309. (ISBN:978-1-4673-4751-8)

[15] Gy. Györök, M. Makó. Configuration of EEG input-unit by electric circuit evolution. In: 9th International Conference on Intelligent Engineering Systems (INES2005), 2005. 1–7. p.

[16] Gy. Györök, M. Makó, J. Lakner. Combinatorics at electronic circuit realization in FPAA. In: Acta Polytechnica Hungarica, Journal of Applied Sciences, 2009. 6(1). 151–160. p.

[17] Gy. Györök. The function-controlled input for the IN CIRCUIT equipment. In: 8th Intelligent, Engineering Systems Conference(INES2004), 2006. 443–446. p.

[18] Gy. Györök. Self configuration analog circuit by FPAA. In: 4th Slovakien – Hungarien Joint Symposium on Applied Machine Intelligence (SAMI2006), 2006. 34–37. p.

[19] Gy. Györök, L. Vokorokos, L. Hluchý. Crossbar network for automatic analog circuit synthesis. In: 12th International Symposium on Applied Machine Intelligence and Informatics (SAMI 2014). IEEE Computational Intelligence Society. Szerk.: J. Fodor. Budapest. 2014. ISBN:978-1-4799-3441-6, 263–267. p.