Journal of Engineering Science *Vol. XXIX, no. 2 (2022), pp. 91 - 107*
Fascicle *Electronics and Computer Science* *ISSN 2587-3474*
Topic *Computers and Information Technology* *eISSN 2587-3482*

https://doi.org/10.52326/jes.utm.2022.29(2).09
UDC 004.8/.9:576

COMPUTER SYSTEMS SYNTHESIS INSPIRED FROM BIOLOGIC CELLS STRUCTURES

Silvia Munteanu*, ORCID: 0000-0003-0749-8457,
Viorica Sudacevschi, ORCID: 0000-0003-0125-3491,
Victor Ababii, ORCID: 0000-0002-0769-8144

Technical University of Moldova, 168 Ștefan cel Mare Blvd., Chișinău, Republic of Molodova
*Corresponding author: Silvia Munteanu, *silvia.munteanu@calc.utm.md*

Received: 02. 20. 2022
Accepted: 03. 25. 2022

Abstract. This paper deals with a synthesis method of membrane computer systems inspired from the biological cells structure. The functional concept of the computer system is based on the internal structure, chemical transformations and the way of interaction between living cells. Thus, the computer system is composed of a set of autonomous, homogeneous or heterogeneous computing cells, which communicate with each other in synchronous or asynchronous mode. The algorithmic complexity of the solved problem depends on the knowledge gained, the rules of data processing and the computer system topology. It is proposed the sequence of operations for the synthesis of membrane computer systems with implementation in reconfigurable FPGA architectures which includes: analysis of behavioural models of living organisms and their interpretation in artificial intelligence models; structure and functional description of the computing cell; cell modelling and performance evaluation using Petri nets; Van diagram of the membrane computer system topology; formal description of the membrane computer system topology by applying the JSON formatting language; validation, functional modelling and performance evaluation of the membrane computer system topology using Petri nets; Hardware Description Language (HDL) code of the cell from which the principle electrical diagram and the time diagrams are obtained; implementation of the computer system in the FPGA circuit.

Keywords: *computational topology, cognitive properties, evolutionary computing, FPGA, HDL, knowledge, membrane computing, nature-inspired computing, natural computing, P-Systems, Petri nets, swarm intelligence, JSON.*

Rezumat. În lucrare se propune o metodă de sinteză a sistemelor de calcul membranar inspirate din structura celulelor biologice. Conceptul funcțional al sistemului de calcul este bazat pe structura internă, transformările chimice și modul de interacțiune dintre celulele vii. Astfel, sistemul de calcul este compus dintr-o mulțime de celule de calcul autonome, omogene sau eterogene, care comunică între ele în regim sincron sau asincron. Complexitatea algoritmică a problemei soluționate depinde de cunoștințele acumulate, regulile de procesare a datelor și topologia sistemului de calcul. Este propusă secvența de operații pentru sinteza sistemelor de calcul membranar cu implementare în arhitecturi reconfigurabile FPGA care include: analiza modelelor comportamentale ale organismelor vii și modele de interpretare a acestora în inteligența artificială; dezvoltarea structurii și descrierea funcțională a celulei de calcul; modelarea și evaluarea performanțelor celulei prin Rețele Petri; sinteza diagramei Van a topologiei sistemului de calcul membranar; descrierea formală a topologiei sistemului de calcul membranar prin aplicarea limbajului de formatare JSON; validarea, modelarea funcțională și evaluarea performanțelor topologiei sistemului de calcul membranar prin Rețele Petri; elaborarea codului de descriere hardware a celulei în baza căreia sunt obținute schema electrică de principiu și diagramele de timp; implementarea sistemului de calcul în circuitul FPGA.

Cuvinte cheie: *topologii de calcul, proprietăți cognitive, calcul evolutiv, FPGA, HDL, cunoștințe, calcul membranar, calcul inspirat din natură, calcul natural, P-Systems, rețele Petri, inteligență de grup, JSON.*

Introduction

The theory of evolution is the most important theory in biology. Evolution can add an extra dimension to a lot of aspects of natural history, give new sense to living facts and social change. Evolution means change, change in the form and behaviour of organisms between generations. The forms of organisms, at all levels from DNA sequences to macroscopic morphology and social behaviour, can change from those of ancestors during evolution. Adaptation is another crucial concept of the theory of evolution and refers to life design and those properties of living being that allow them to survive and reproduce in nature. Natural selection, also an important concept of evolution, means that some individuals in the population tend to contribute with more descendants to the next generation than others, being better adapted to the conditions of the environment in which they live [1, 2].

In 1998, academician Gheorghe Păun laid the foundations of a new field in computer science called membrane computing. Membrane computing is a branch of natural computing, being defined as a field of computer science with two major objectives: to be inspired by nature for the benefit of computer science - data bases, data operations, computer architectures, in the most general way - and vice versa, to provide new tools to those who study nature by other means, such as biologists or physicists. Natural calculus is a field that contains genetic algorithms, evolutionary calculus, with ideas from the genome area, neural algorithms. Computing inspired by the area of DNA - DNA Computing, also proposes a new hardware, the DNA molecule. Membrane computing aims to teach from the example of the cell and create cell-like systems by the help of mathematics and computer science [3].

Nature has served as a source of inspiration for many areas of the exact sciences, including computer science, artificial intelligence and related fields [4, 5]. Because most modern technological systems that have been and are being developed are extremely complex, distributed and interconnected, they depend on efficient communication, require high flexibility, adaptability and the ability to meet quality and performance requirements.

Carrying out a more detailed analysis of natural systems, we can see that they are usually characterized by a very high level of complexity. This complexity indicates that the behaviour of natural systems may be unpredictable and inaccurate, but at the same time living organisms and the ecosystems in which they are living have a substantial degree of survival. Examples of such resistant systems can be colonies of social organisms, nervous systems, etc. The survivability of the system is determined by a number of parameters: a large number of objects in each system that can be replaced by others; free but flexible interconnections between objects; differences between objects in the system that allow flexible responses to a changing environment; the complex environment in which all components interact and produce various responses / actions [4]. Natural organisms have demonstrated by their existence the ability to cope with exceptional situations and to adapt to the environment through the ability to learn throughout life and by evolving over several generations [5].

It follows that natural systems have several properties that form the basis for many nature-inspired applications, namely dynamics, flexibility, robustness, self-organization, simplicity of basic objects and decentralization. Another important aspect of living organisms is that they always live in conditions of extreme competition. Therefore, they can be subject to actions that may partially or completely destroy the population, but in most cases they can be fully recovered. The ability to recover is determined by the presence of the immune system of individuals or interactions within and between populations. Resource limitation is one of the selection criteria that can ultimately lead to more efficient self-organization and recovery [1-3].

Underlying the definition of natural computation is the concept of P-System which is a computational model based on processes inspired by the behaviour and structure of biological cells. P-System is an abstraction model of computational processes by applying the interaction mode of chemicals and the cell membrane. This idea was first proposed by Gheorghe Păun in 1998 [6], which were later developed as a new branch of theoretical informatics [7, 8].

In the process of membrane computer systems (P-Systems) developing were designed some components that are functionally present in biological cells. The activity of a P-System can only be analysed in an environment that is the provider of input data and respectively the consumer of the results of their processing. Membranes are the main components of a P-Systems structure. A computational membrane is an autonomous functional logical unit that includes a set of objects, a set of rules, and / or a set of other membranes. The outer membrane that interacts with the environment is also called the "container membrane". A membrane may dissolve itself and in this case its contents migrate to the membrane to which it belongs or divide while retaining all or part of its properties [7, 8].

The functional logic of membrane computer systems is determined by a set of rules. A rule is validated for a lot of objects or input conditions that are applied, is consumed and produce a lot of objects or output conditions. In order to exclude internal competition, a rule may take precedence over other rules (dominant rules), so that priority computational models can be developed with priority in which less dominant rules (with low quality parameters) will be applied only in critical or exceptional conditions [9-11].

The algorithmic complexity of a P-System depends on the set of rules defined for each membrane and their topological structure. The main objective of applying membrane computation models is the optimal distribution of computational tasks in order to obtain a parallel computing process with maximum efficiency. Membrane computer systems with both asynchronous and synchronous processing can be defined [13, 14]. A topology of a P-System can be defined in the form of Ven diagram, tree or formal description [12].

Membrane computing models offer the possibility to formally and structurally describe computing and control systems of varying complexity, for example specialized logic structures and processors, reconfigurable computer systems, complex computing architectures, which include functional elements with hierarchical interaction, parallel and concurrent computing architectures and network topologies for distributed computer systems.

Membrane computing cells being considered systems with artificial intelligence and cognitive properties implement as rules for data processing both mathematical and logical models, as well as models based on neural networks, Fuzzy logic and evolutionary computation.

The paper [15] presents a study of the interaction between membrane computation and Fuzzy logic theory. Theoretical research is focused on the processing of uncertainties in P-Systems, the fusion, representation and reasoning of unclear knowledge. Applications of Fuzzy-membrane computing focus on the representation of Fuzzy knowledge and the identification of its uncertainties.

Another example is described in paper [16] which examines the use of membrane computing models to control a group of mobile robots. The topics addressed in the paper are oriented towards the complexity of the robot and the tasks performed, the development of a genetic algorithm that uses a high level of abstraction, the design of swarm algorithms using a top-down approach with real-time operation.

P-Systems computing architectures represent a multitude of homogeneous or heterogeneous units for data processing. In the paper [17] it was shown that the synchronous interaction between them plays a decisive role, especially for real-time systems. The paper investigates synchronous membrane computer systems with communication and division rules, using the strategy of evolving parallelism together with the synchronization between uncooperative rewriting rules. As mentioned in the paper [18], in order to achieve maximum parallelism, a strategy for evolving the topology of the membrane computer system is used by applying synchronization between the applied rules. The most efficient are the membrane computer systems that operate in standby or event mode. The efficiency of the synchronization process is demonstrated by simulating the basic operations (addition, subtraction, multiplication, and division).

An essential factor in the evolution of membrane computer systems is their cognitive abilities. The paper [19] explored two methods of applying associative memory for knowledge storage. In the first method, associative memory is considered a device for memorizing the process of objects evolution, in which they are constantly updated to monitor the evolution and circumstances in which they were produced. This model allows the study of the evolution of the membrane. In the second method, associative memory is considered as a device for storing previously consumed objects. This model allows describing the dynamics of the object allocation process.

The field of use of membrane computer systems and models is very wide. The scientific papers for the last 20 years specify the advantages of applying membrane computation in such areas as image processing [20], biology, ecology, robotics or engineering [21, 22], decision-making systems [23, 24], collaborative systems and swarm intelligence [24, 25], reliable diagnostic systems [26], etc.

In this paper, the authors propose a new method for membrane computation models implementation in reconfigurable hardware architectures. In order to achieve the established objectives, were elaborated the algorithm for the synthesis of hardware membrane computer systems, the functional scheme of the data processing cell based on membrane computing models, the Petri net model for performance evaluation, the functional model for membrane computing structures with complex topology, the model for the formal description of membrane computing structures with complex topology, the HDL code for cell implementation in hardware architecture. The validation of the obtained results is performed based on the diagrams generated as a result of the modelling and simulations of membrane computing cells implemented into FPGA circuits.

1. Algorithm for synthesis of membrane computer systems implemented in hardware architectures

The basis of proposed algorithm (Figure 1) for synthesis of the membrane computer systems implemented in hardware architectures is the multitude of knowledge gained from various fields of natural sciences (biology, genetics, natural intelligence, etc.) and exact sciences (physics, mathematics, artificial intelligence, engineering and computer science).

The synthesis of membrane computer systems implemented in hardware architectures includes the following steps:

Nature Inspired Computing - includes the analysis and mathematical and behavioural description of Swarm Computing models and the association of physiological and chemical processes performed in living cells with models and the formal description of computational processes for the implementation of membrane architecture in hardware architecture;

Artificial Intelligence - includes providing mathematical and logical models based on neural networks and Fuzzy logic to implement the knowledge and rules of data processing performed by the cell. At this stage, optimization models based on genetic algorithms can be applied, which will ensure the optimization of knowledge and data processing rules;

Cells Computing - based on the accumulated information and specifications for the membrane computing system, the synthesis of the cell structure scheme is performed. As a result of this stage, the cell architecture is obtained at the level of functional elements and data flows;

Cell: Petri Net Modelling - modelling and performance evaluation of the membrane computing cell through Petri nets [27, 28, 29]. At this stage, parallel and concurrent processes, resource access conflicts, and the efficiency of the rules synchronization model are identified. As a result of the performance analysis, graphs are generated that will highlight the strengths and weaknesses of the cell architecture;

Membrane Computing (P-Systems) - based on the mathematical model or formal description is developed the topology of the membrane computer system in the form of a Van diagram which specifies the set of rules made by each membrane and the relationships between them;

XML (JSON) - based on the Van diagram, a formal description of the topology of the membrane computer system is presented applying the XML (JSON) formatting language [30]. The result of this operation is a text file that describes the system topology, the inputs and outputs for each cell, and the identifiers of the cell hardware description files;

P-Systems: Petri Net Modelling - at this stage the performance modelling and evaluation of the membrane computer system through Petri nets is performed [31, 32, 33]. The purpose of these models is to optimize the synchronization process between the system cells and their rules. As a result of modelling, time graphs and diagrams are generated;

P-Systems: HDL - cell implementation (P-Systems) in hardware architecture based on HDL [34]. As a result of code compilation in the Intel Quatrus Prime 19.1 Design Software

[35] design tool, the wiring diagram, the files for the configuration of the FPGA circuit and the time diagrams of the membrane computer system are obtained;

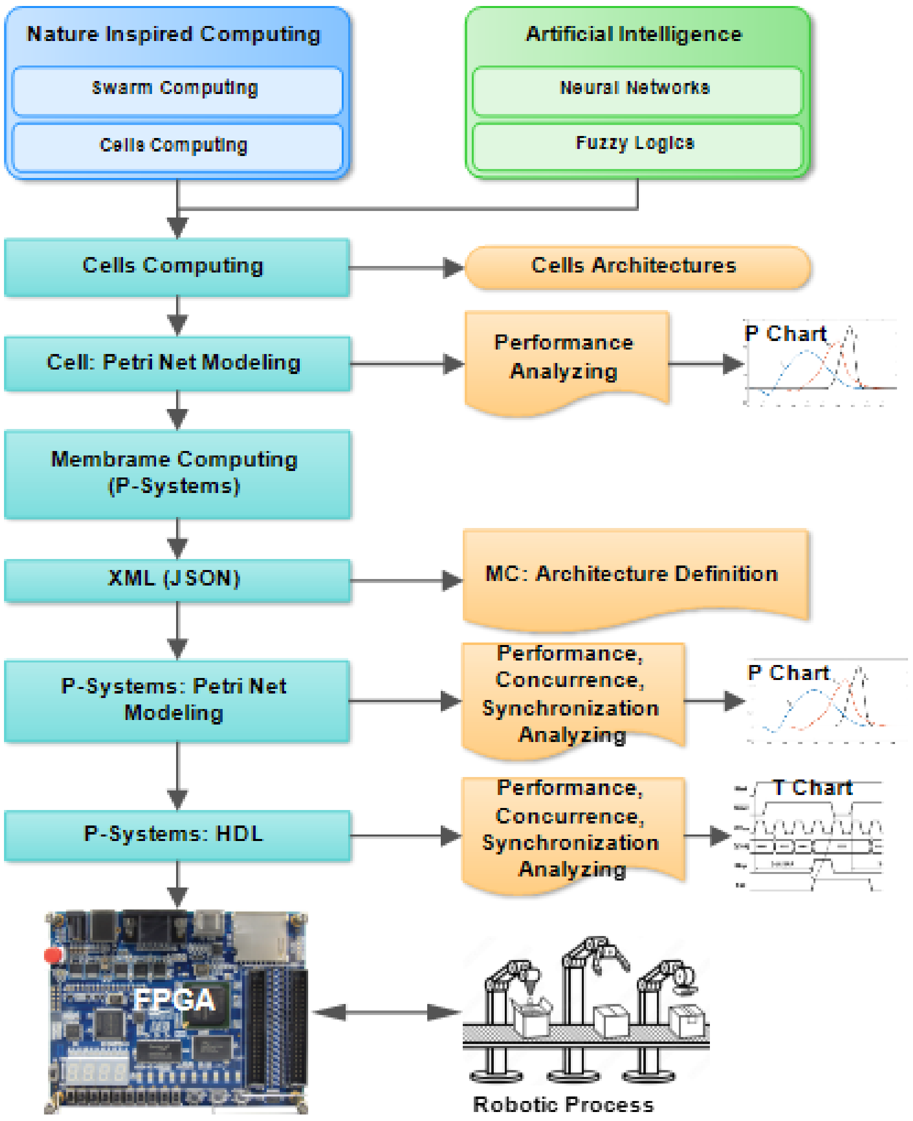

Figure 1. Algorithm for synthesis of membrane computer systems implemented in hardware architectures.

FPGA - loading the configuration code into the FPGA circuit [36] which is then used to control a robotic process which was used to formulate the specifications of the synthesis process of membrane computer systems.

2. Synthesis and modelling of computing cells

Computing cells represent a set of smart and autonomous elements for data processing. The data processing algorithm is determined by the set of rules obtained as a result of adapting the knowledge gained during the evolution of the cell. The interaction between the cells and the topological structure determines the algorithmic complexity of the membrane computing system. The synthesizing process of the computing cell structure requires the analysis of the chemical and physiological processes that take place in living cells and the tasks that are performed by it. A membrane computing system forms a collective intelligence (Swarm Intelligence) and can be composed of homogeneous or heterogeneous computing cells. In the paper the synthesis of a general cell structure that can serve as a model for the synthesis of cells oriented towards certain specific fields of application is proposed.

The computing cell (Figure 2) consists of the following components:

Environment - the cell activity environment that delivers input data for the cell activity and reacts to the actions applied as a result of data processing;

$X(t)$ - input data or activity environment state;

Filtering (I) - the input data filtering mechanism that selects only certain data and allows them to enter into the cell to be processed;

Fuzzification – Fuzzy operations for input data that generate the set of events for processing;

Associative Memory (I) - associative memory for storing the list of input events;

$X[T]$ - the list of events waiting to be processed;

Knowledge - the set of knowledge that forms the rules of data processing;

$Op[T]$ - the set of elementary operations defined by the set of rules for data processing (events);

ALU - the block for performing arithmetic and logic operations according to the set, defined by the applied rule;

$Y[T]$ - list of events (decisions and actions) obtained as a result of data processing;

Associative Memory (O) - associative memory for storing the list of output events. After processing the input and output events they are removed from the lists;

Filtering (O) - the output data filtering mechanism that selects events to be moved at the output of the computing cell;

De-Fuzzification - turns the list of output events into action data or communication with the environment;

$Y(t)$ - action or communication data with the environment;

Synchronization - the synchronization block of the operations performed by the computing cell.

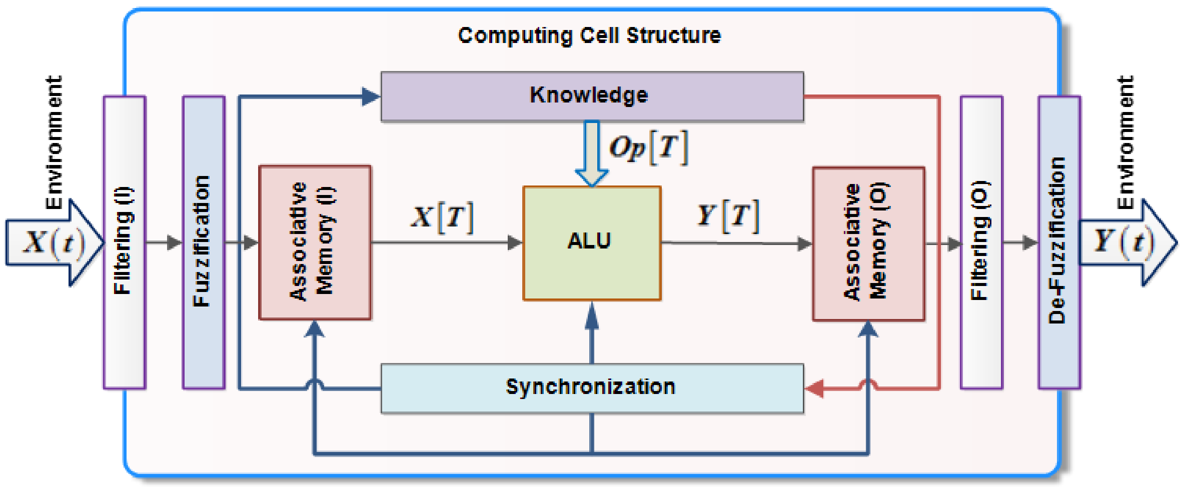

Figure 2. Computing Cell diagram.

Functional validation and performance evaluation of computing cells was performed based on the Petri net model shown in Figure 3. For this purpose, the following functions performed by the model cell were identified:

1. The cell receives from the activity *Environment* two state values $X(t)=\{x_1(t),x_2(t)\}$ modelled by timed transitions $\{t_1,t_2\}$;

2. The events $X_1[T]|x_1(t)\geq 4$ and $X_2[T]|x_2(t)\geq 6$ are identified by the *Fuzzification* which are modelled by transitions $\{t_3,t_4\}$ and arcs $\{p_1,t_3\}$ with weights 4 and 6, respectively;

3. *Associative Memory (I)* stores two values of events $X[T]=\{X_1[T],X_2[T]\}$ and is modelled by places $\{p_3,p_4\}$;

4. Places p_{12} and p_{13} monitors the waiting time to serve the events $X_1[T]|Time\ Delay(1)$ and $X_2[T]|Time\ Delay(2)$, being synchronized by timed transitions $\{t_{13},t_{14}\}$;

5. The *ALU* block is modelled by place p_5 and synchronized by $\{t_5,p_{8,}t_7\}$ to serve the events $X_1[T]$, and respectively synchronized by $\{t_6,p_{9,}t_8\}$ to serve the events $X_2[T]$, where t_7 and t_8 are timed transitions;

6. The places p_{14} - *Event Processing (1)* and p_{15} - *Event Processing (2)* monitor the served events $X_1[T]$ and $X_2[T]$;

7. The places $\{p_6,p_{7,}\}$ model the *Associative Memory (O)* in which two values of the output events are stored $Y[T]=\{Y_1[T],Y_2[T]\}$;

8. The *De-Fuzzification* block is modelled by combinations $\{p_{10,}t_{11}\}$ and $\{p_{11,}t_{12}\}$ which transforms the output events $Y_1[T]$ and $Y_2[T]$ into pulse sequences $y_1(t)|n=8$ and $y_2(t)|n=10$, where t_{11} and t_{12} there are timed transitions.

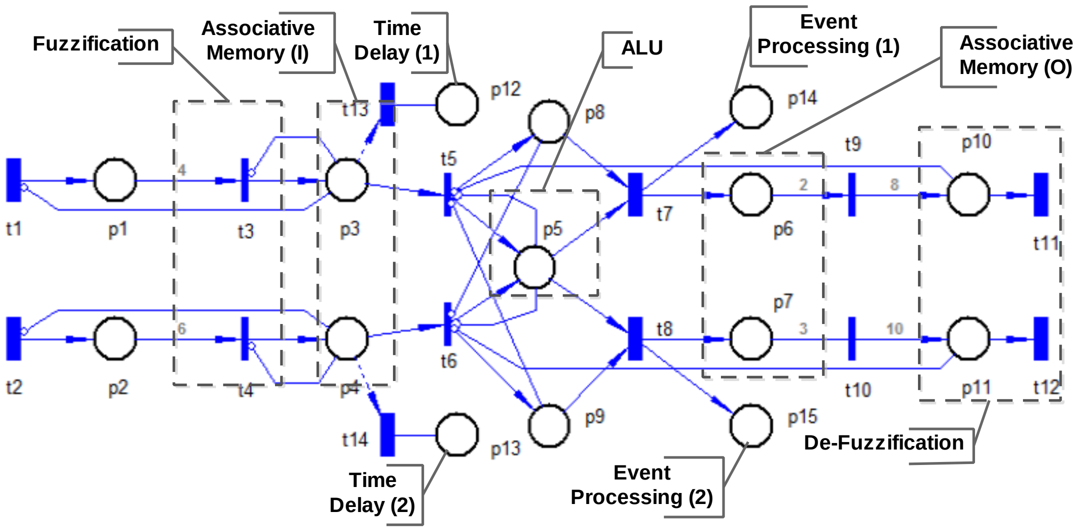

Figure 3. Petri Net model for Computing Cell validation and performance evaluation.

The VPNP application, developed at the Department of Computer Science and Systems Engineering by the university professor Emilian Guțuleac [37], was used for the functional validation and evaluation of the computing cell performance.

3. Synthesis and modelling of the membrane computing (P-Systems) architecture

As mentioned above, the synthesis of the membrane computing system topology (P-Systems) depends on the specific of the solved problem and its algorithmic complexity. The purpose of the synthesis is to obtain a computing architecture with maximum parallelism and minimal concurrency in data processing and resources accessing.

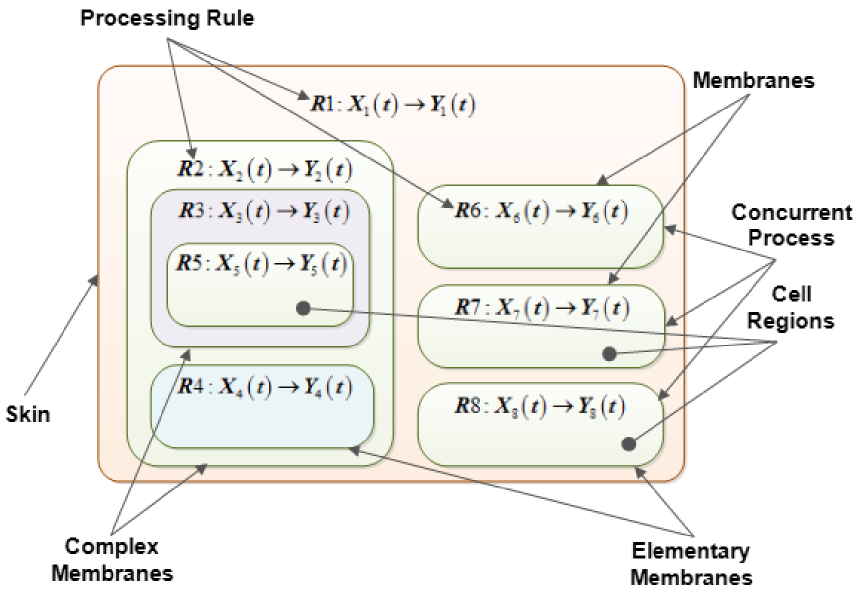
Processing Rule
$R1: X_1(t) \rightarrow Y_1(t)$
Membranes
$R2: X_2(t) \rightarrow Y_2(t)$
$R3: X_3(t) \rightarrow Y_3(t)$
$R5: X_5(t) \rightarrow Y_5(t)$
$R6: X_6(t) \rightarrow Y_6(t)$
Concurrent Process
$R7: X_7(t) \rightarrow Y_7(t)$
Cell Regions
$R4: X_4(t) \rightarrow Y_4(t)$
$R8: X_8(t) \rightarrow Y_8(t)$
Skin
Complex Membranes
Elementary Membranes

Figure 4. Membrane computing (P-Systems) architecture.

Figure 4 shows the synthesis result of a calculation membrane topology using Van diagram format that includes the following specifications:

1. The set of rules for data processing $R = \{R_1, R_2, ..., R_8\}$ that also determine the name of the calculation cell;
2. The set of elementary membranes $\{R_4, R_5, R_6, R_7, R_8\}$;
3. The set of complex membranes $\{R_1, R_2, R_3\}$;
4. Concurrent / parallel processes $\{R_2, R_6, R_7, R_8\}$ and $\{R_3, R_4\}$;
5. Sequential processes $\left\{R_1 : \left\{R_2 : \left\{R_3 : \{R_5\}, R_4\right\}, R_6, R_7, R_8\right\}\right\}$;

Skin external membrane that communicates with the business environment.

The Petri net model for validation and performance evaluation of the membrane computing architecture is presented in Figure 5.

The Petri net model includes the following elements:

1. The calculation cell R_1 modelled by the timed transition t_1 (generator of external events) and place p_1 (memory for the accumulation of events generated by the activity environment);

2. The set of computing cells $R_2 : \{t_2, p_2, t_6, p_6\}$, $R_6 : \{t_3, p_3, t_7, p_7\}$, $R_7 : \{t_4, p_4, t_8, p_8\}$ and $R_8 : \{t_5, p_5, t_9, p_9\}$, which operate in parallel / concurrently, where: $\{t_2, t_3, t_4, t_5\}$ are transitions

for validating event processing, $\{p_2, p_3, p_4, p_5\}$ are places for events modelling accepted for processing by that cell, $\{t_6, t_7, t_8, t_9\}$ are timed transitions that model the data processing time required for each calculation cell, $\{p_6, p_7, p_8, p_9\}$ are places that model the end of the data processing performed by the respective cell;

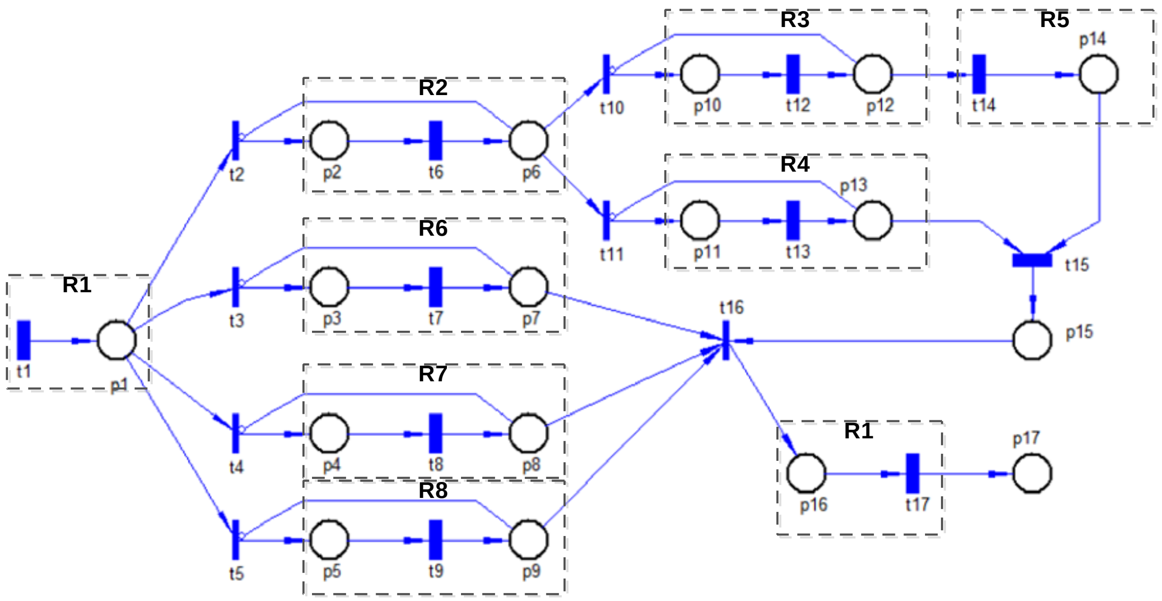

Figure 5. Modelling of membrane computing (P-Systems) architecture.

3. The set of computing cells $R_3:\{t_{10}, p_{10}, t_{12}, p_{12}\}$ and $R_4:\{t_{11}, p_{11}, t_{13}, p_{13}\}$, where: $\{t_{10}, t_{11}\}$ there are transitions that model the distribution of events to those cells, $\{p_{10}, p_{11}\}$ there are places that indicate that the cell is processing the data, $\{t_{12}, t_{13}\}$ there are timed transitions that model the data processing time by that cell, $\{p_{12}, p_{13}\}$ are places that model the end of data processing by that cell;

4. The cell $R_5:\{t_{14}, p_{14}\}$ where t_{14} is a timed transition that models the data processing time and p_{14} is the place that models the end of the cell data processing;

5. Timed transition t_{15} and place p_{15} model the synchronization process between cells $\{R_3, R_4, R_5\}$;

6. The cell $R_1:\{t_{16}, p_{16}, t_{17}\}$ model the action of the membrane computing architecture on the activity environment, where the place p_{17} indicates the number of processed events.

In order to achieve the circuit of the membrane computing system presented in the form of a Van diagram, it is proposed to use a formal description model in JSON format [30]. The purpose of these transformations is to gradually move from the Van diagram presentation of the membrane computing system to the HDL hardware description code. Figure 6 shows the result of this operation for the example analysed in Figure 4.

Description of the JSON code for membrane computing system formatting:

1. *"Membrane_Computing"* - the name of the main object that integrates the architecture of the membrane computing system in a logical structure;
2. *"Cell_Name"* - the name of the cell, obtained from the Van diagram that describes the topology of the membrane computing system;

```
{ // JSON model for Membrane Computer systems
// Synthesis Inspired from Biologic Cells Structures
"Membrane_Computing" : { // Cell_R1 definition
        "Cell_Name" : "Cell_R_1",
                "Cell_Type" : "Cell_Type_1",
        "Input" : {"Input_1" : "IValue_1"},
        "Output" : {"Output_1" : "OValue_1"},
        "Knowledge" : {
                "Data" : ["KData_1"],
                "Methods" : "e:/Cells_Device/R_1.inc"
        },
        "Cell_2" : { // Cell R2 definition
        "Cell_Name" : "Cell_R_2",
        "Cell_Type" : "Cell_Type_2",
        "Input" : {"Input_1" : "IValue_1"},
        "Output" : {"Output_1" : "OValue_1"},
        "Knowledge" : {
                "Data" : ["KData_2"],
                "Methods" : "e:/Cells_Device/R_2.inc"
        },
        "Cell_3" : { // Cell R3 definition
        "Cell_Name" : "Cell_R_3",
        "Cell_Type" : "Cell_Type_3",
        "Input" : {"Input_1" : "IValue_1"},
        "Output" : {"Output_1" : "OValue_1"},
        "Knowledge" : {
                "Data" : ["KData_3"],
                "Methods" : "e:/Cells_Device/R_3.inc"
                },
        "Cell_5" : { // Cell R5 definition
        "Cell_Name" : "Cell_R_5",
        "Cell_Type" : "Cell_Type_5",
        "Input" : {"Input_1" : "IValue_1"},
        "Output" : {"Output_1" : "OValue_1"},
        "Knowledge" : {
                "Data" : ["KData_5"],
                "Methods" : "e:/Cells_Device/R_5.inc"
                }}},
"Cell_4" : { // Cell R4 definition
        "Cell_Name" : "Cell_R_4",
        "Cell_Type" : "Cell_Type_4",
        "Input" : {"Input_1" : "IValue_1"},
        "Output" : {"Output_1" : "OValue_1"},
        "Knowledge" : {
                "Data" : ["KData_4"],
                "Methods" : "e:/Cells_Device/R_4.inc"
                }}},
        "Cell_6" : { // Cell R6 definition
        "Cell_Name" : "Cell_R_6",
        "Cell_Type" : "Cell_Type_6",
        "Input" : {"Input_1" : "IValue_1"},
        "Output" : {"Output_1" : "OValue_1"},
        "Knowledge" : {
                "Data" : ["KData_6"],
                "Methods" : "e:/Cells_Device/R_6.inc"
        }},
        "Cell_7" : { // Cell R6 definition
        "Cell_Name" : "Cell_R_7",
        "Cell_Type" : "Cell_Type_7",
        "Input" : {"Input_1" : "IValue_1"},
        "Output" : {"Output_1" : "OValue_1"},
        "Knowledge" : {
                "Data" : ["KData_7"],
                "Methods" : "e:/Cells_Device/R_7.inc"
        }},
        "Cell_8" : { // Cell R6 definition
        "Cell_Name" : "Cell_R_8",
        "Cell_Type" : "Cell_Type_8",
        "Input" : {"Input_1" : "IValue_1"},
        "Output" : {"Output_1" : "OValue_1"},
        "Knowledge" : {
                "Data" : ["KData_8"],
                "Methods" : "e:/Cells_Device/R_8.inc"
        }}}}
```

Figure 6. JSON format for membrane computing architecture description.

3. *"Cell_Type"* - the type of cell that represents a specification of it (ordinary, complex, etc.);
4. *"Input"* – specification of the input variables for the respective cell;
5. *"Output"* - specification of the output variables for the respective cell;
6. *"Knowledge"* - the specifications of the object for the description of the knowledge that will present the rules of data processing;
7. *"Date"* - the list of knowledge set for the initial state of the cell;
8. *"Methods"* - the name of the file and the path to it for implementing the rules of data processing.

4. HDL synthesis of membrane computing architecture

As an example for the synthesis of a membrane computing architecture using HDL code, it is proposed to design a computing cell with input variable $X\ (t)$ and output variable $Y(t)$, where:

$$X\ (t) = \left\{ X_1[T] \in \{0000,...,0111\}, X_2[T] \in \{1000,...,1111\} \right\},$$

$\forall X_1[T] : Op[T] \rightarrow Y_1[T]$ and $\forall X_2[T] : Op[T] \rightarrow Y_2[T]$.

As a result of the application of the broadcasting operations, the transformations $Y_1[T] \rightarrow Y_1(t)$ and $Y_2[T] \rightarrow Y_2(t)$ will take place, where $Y_1(t)$ and $Y_2(t)$ are the sequence of synchronization signals with a number proportional to the respective variables $X_1[T]$ and $X_2[T]$. The HDL description code of the computing cells is shown in Figure 7.

```
SUBDESIGN Celula
(
          clk, load, ena, clr, fuzzy[3..0]    : INPUT;
          yt1, yt2, q_yt1[3..0], q_yt2[3..0]            : OUTPUT;
)
VARIABLE
          xt1, xt2                        :           NODE;
          z_yt1, z_yt2            :                   NODE;
          count_yt1[3..0]         :                   DFF;
          count_yt2[3..0]         :                   DFF;
BEGIN
                    TABLE               -- Fuzificztion
                              fuzzy[3..0]                 =>        xt1,                xt2;
                                        B"0XXX"           =>        1,                  0;
                                        B"1XXX"           =>        0,                  1;
                    END TABLE;
                              count_yt1[].clk = clk;
                              count_yt1[].clrn = !clr;
                              count_yt2[].clk = clk;
                              count_yt2[].clrn = !clr;
          -- Function 1
                              IF        (xt1 & load) THEN
                                                  count_yt1[].d = fuzzy[];
                              ELSIF ((count_yt1[3].q # count_yt1[2].q # count_yt1[1].q # count_yt1[0].q) & ena) THEN
                                        count_yt1[].d = count_yt1[].q - 1;
                              ELSE
                                        count_yt1[].d = count_yt1[].q;
                              END IF;
                              yt1 = count_yt1[0].q;
                              q_yt1[] = count_yt1[].q;
          -- Function 2
                              IF        (xt2 & load) THEN
                                        count_yt2[].d = fuzzy[];
                              ELSIF ((count_yt2[3].q # count_yt2[2].q # count_yt2[1].q # count_yt2[0].q) & ena) THEN
                                        count_yt2[].d = count_yt2[].q - 1;
                              ELSE
                                        count_yt2[].d = count_yt2[].q;
                              END IF;
                              yt2 = count_yt2[0].q;
                              q_yt2[] = count_yt2[].q;
END;
```

Figure 7. The HDL code of Cell.

Figure 8 shows the logic diagram of the cell obtained as a result of compiling the HDL code using the Intel Quatrus Prime 19.1 development environment.

Figure 9 shows the graphical symbol of the cell with the input and output signals of the cell obtained as a result of HDL code compilation in the Intel Quatrus Prime 19.1 development environment.

Specification of the input signals (Figure 8 and 9):

1. *clk* - clock signal synchronizes the data processing;
2. *load* - loading data into the associative input memory of the cell;
3. *ena* - validation of data processing;
4. *clr* - deleting data from associative memory;
5. *fuzzy [3..0]* - the binary code of the input data.

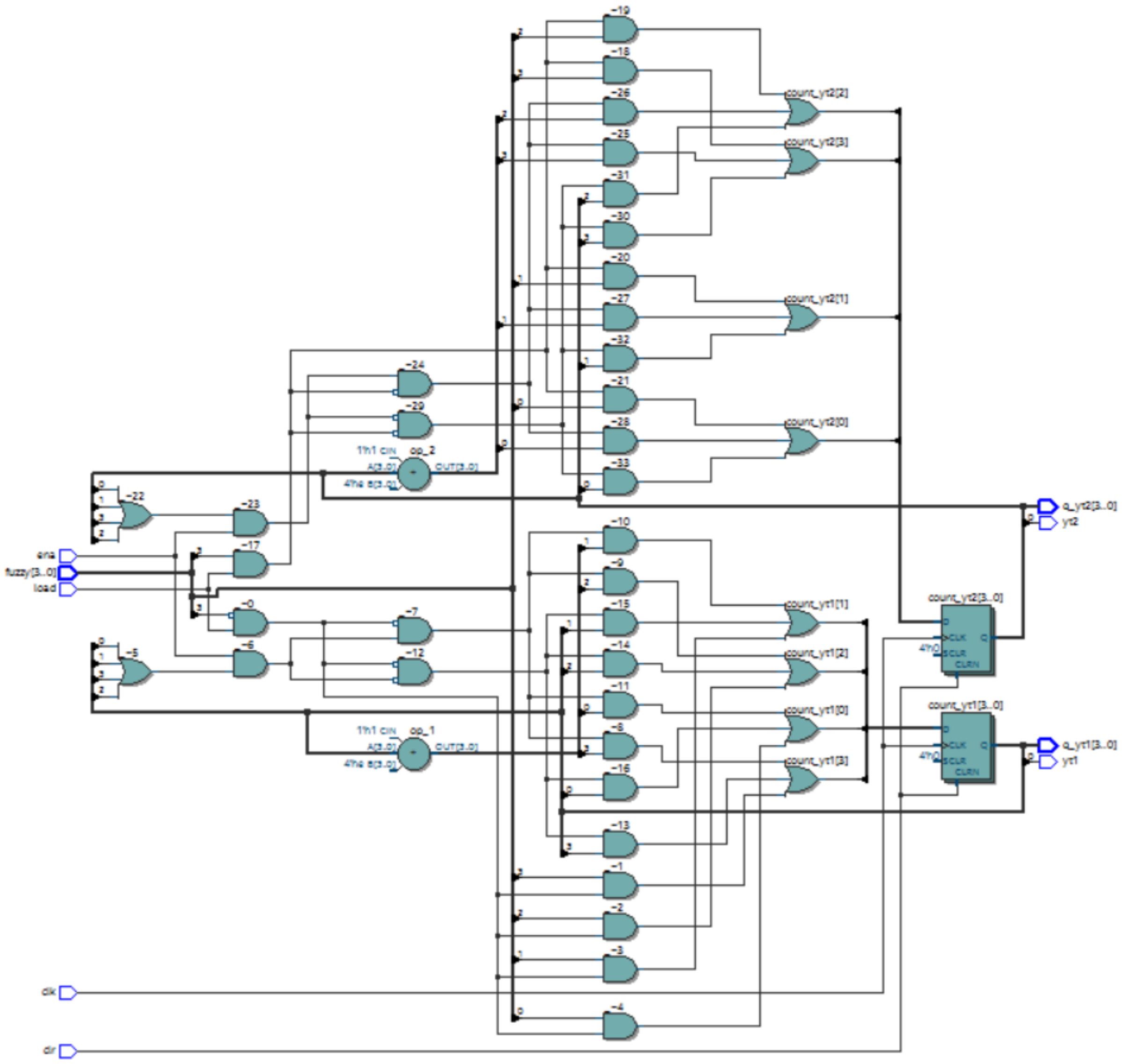

Figure 8. The logic circuit obtained from HDL code.

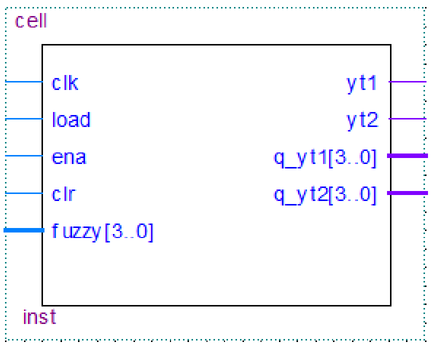

Figure 9. The Cell symbol.

Specification of the output signals (Figure 8 and 9):
1. *yt1* - pulse sequence;
2. *yt2* - pulse sequence;
3. q_*yt1 [3..0]* - the binary code from the associative output memory;
4. *q_yt2 [3..0]* - the binary code from the associative output memory.

The values of the variables *fuzzy [3..0]*, *q_yt1 [3..0]* and *q_yt2 [3..0]* are used to interconnect the cells according to the JSON file format.

The result of the functional validation of the computing cell, in the form of time diagram, is shown in Figure 10.

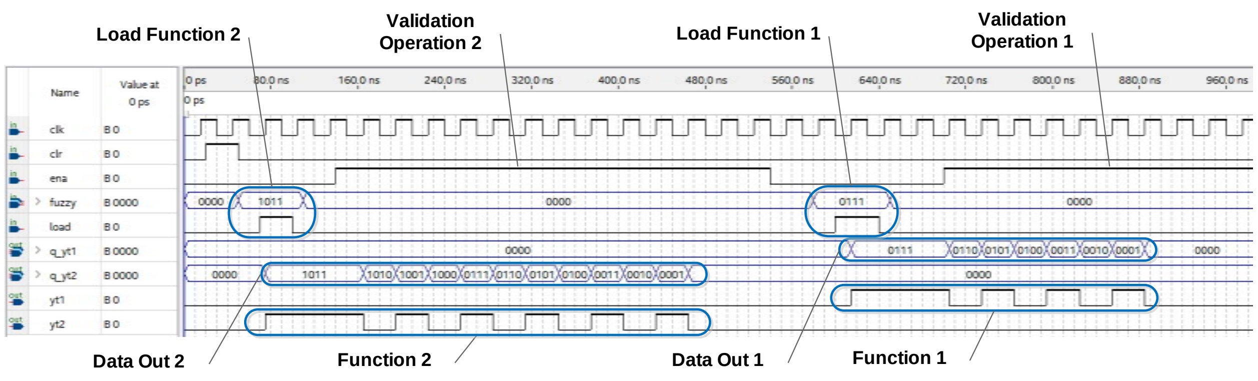

Figure 10. The Time diagram.

Specification of the signals grouped by function:

1. *Load Function 2* - uploading data for the function according to the condition $X(t) = X_2[T] \in \{1000,...,1111\}$;
2. *Data Out 2* - binary code output data $Y_2[T]$;
3. *Function 2* - pulse sequence $Y_2(t)$;
4. *Load Function 1* - uploading data for function according to the condition $X\ (t) = X_1[T] \in \{0000,...,0111\}$;
5. *Data Out 1* - binary code output data $Y_1[T]$;
6. *Function 1* - pulse sequence $Y_2(t)$.

5. Functional testing of membrane computing cells implemented in HDL models

Functional testing of membrane computing cells implemented in HDL models was performed based on the Altera DE0 Board kit [38].

This development kit is equipped with an Altera Cyclone III 3C16 FPGA circuit that provides the user (designer) with 15,408 LEs and 346 Input / Output pins, enough to test the example proposed in the paper.

Figure 11 shows the functional structure of the stand for testing membrane computing models that includes: a personal computer with *Intel Quartus Prime* development environment installed and the USB cable for connecting the Altera *DE0 Board* kit.

The development kit highlights the areas used in the process of functional testing of membrane computing models: *Binary Switches x 10* - used to enter the input code (variable values $X[T]$); *Binary LEDs x 10* - for displaying in binary code the result of processing the data performed by the cell (variable values $Y[T]$); *4x7-Segment Display* - for displaying the result of data processing in decimal or hexadecimal code; *Extension Headers* - for connecting external input devices and assistive or action devices.

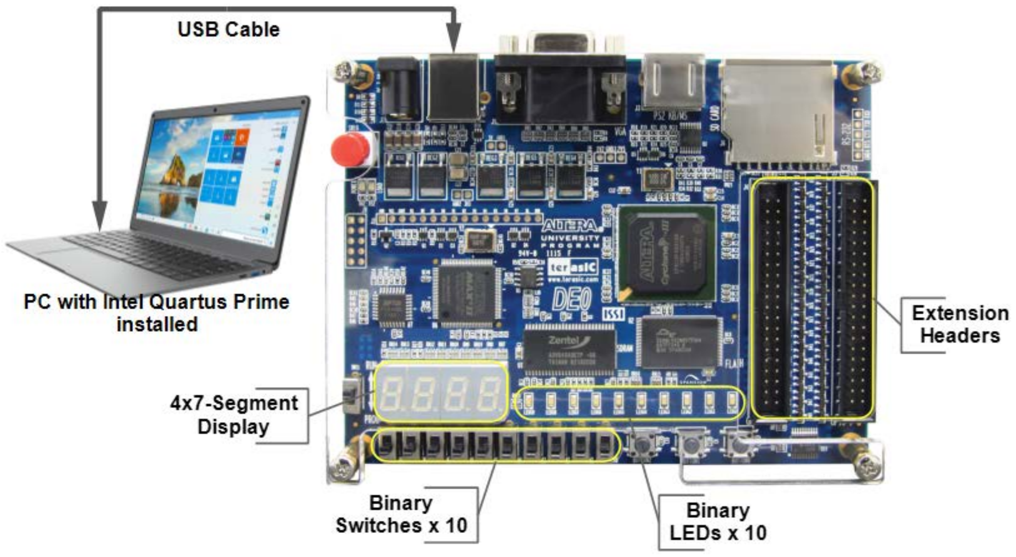

Figure 11. Functional structure of stand for testing of membrane computing models.

Acknowledgments: The results obtained in this paper are part of the research conducted under the PhD topic *"Distributed computational structures based on intelligent algorithms with cognitive properties"*.

Conclusion

The paper describes an application proposed by the authors for the design and implementation of membrane computing systems based on FPGA reconfigurable devices. The synthesis algorithm of membrane computing systems includes a sequence of activities involving knowledge in the fields of biology, nature-inspired computing, artificial intelligence, system modelling, formatting languages, hardware description languages, and systems design based on FPGA reconfigurable devices. The operating model of the computer system is inspired by the functioning and interaction of living cells.

The synthesis process includes the following steps: development of the cell structure diagram for data processing; modelling and evaluating the performance of computing cells and the topology of the membrane computing system through Petri nets; elaboration of the Van diagram for the membrane computing system; formal description of the membrane computation model based on the JSON formatting language; development of the hardware description code of the cells for data processing and implementation of the cells in reconfigurable FPGA computing architectures.

For the future, the development of membrane computing systems for the control of a mobile robot and a robotic arm is planned. The objectives of the research are aimed to highlight the possibilities of performing operations in parallel, the use of methods of synchronization and control of concurrency inside the cells and at the level of topology of the membrane computing system.

References

1. Ridley M. *Evolution.* Third edition, Blackwell, 2004, 751p. ISBN: 1-4051-0345-0.
2. Futuiama D.J. *Evolution.* Third edition, Sinauser Associates, 2013, 655p.
3. Păun G., Rozenberg G., Salomaa A. *DNA Computing: New Computing Paradigm.* Texts in Theoretical Computer Science (An EATCS Series). Springer, Berlin, Heidelberg. 1998, 409p. ISBN: 978-3-540-64196-4.
4. Siddique N., Adeli H. Nature Inspired Computing: An Overview and Some Future Directions. *Cognitive Computing*, 2015, 7: 706-714. DOI: 10.1007/s12559-015-9370-8.

5. De CASTRO, L.N. Fundamentals of natural computing: an overview. *Phys Life Rev.* 2007; 4: 1–36. DOI: 10.1016/j.plrev.2006.10.002.
6. Păun Gh. *Computing with Membranes*. TUCS Report 208. Turku Centre for Computer Science. 1998. ISBN: 978-952-12-0303-9.
7. Păun Gh., Rozenberg G. A guide to membrane computing. *Theoretical Computer Science*. 287 (1): 73-100, 2002. DOI: 10.1016/S0304-3975(02)00136-6. ISSN: 0304-3975.
8. Păun Gh. Introduction to Membrane Computing. *Applications of Membrane Computing*. Springer Berlin Heidelberg. pp. 1-42. ISBN: 978-3-540-29937-0.
9. Alhazov A. *Communication in Membrane Systems with Symbol Objects*. Ph.D. Thesis. Tarragona, Spain: Universitat Rovira i Virgili, 2006, 218p.
10. Alhazov A. Small Abstract Computers. Habilitation Thesis. Chișinău (RM): Academy of Sciences of Moldova, 2013, 255p.
11. Aman B., Ciobanu G. Membrane Systems with Surface Objects. In: *Proceedings of the International Workshop on Computing with Biomolecules*, Wien, Austria, August 27th, 2008, pp. 17-28.
12. Giovitto J.-L., Michel O. *The Topological Structures of Membrane Computing*. LaMI technical report Nr. 70-2001. November 2001, France.
13. Dinneen M. J., Kim Y.-B. and Nicolescu R. Synchronization in P Modules. In: *Unconventional Computation, volume 6079 of Lecture Notes in Computer Science,* Springer-Verlag, Berlin Heidelberg, 2010, pp. 32–44.
14. Dinneen M. J., Kim Y.-B. and Nicolescu R. An Adaptive Algorithm for P System Synchronization. In: *Proceedings of the Twelfth International Workshop on Membrane Computing, (CMC11),* Fontainebleau/Paris, France, 2011, pp. 1–26.
15. Wang T., Zhang G., Perez-Jimenez M.J. Fuzzy Membrane Computing: Theory and Applications. *International Journal of Computers Communications & Control,* 2015, Vol. 10(6), pp. 144, DOI: 10.15837/ijccc.2015.6.2080.
16. A Generic Guide for Using Membrane Computing in Robot Control. *Advances in Computational Intelligence and Robots – Membrane Computing for Distributed Control of Robotic Swarms,* 2017, pp.86-96. DOI: 10.4018/978-1-5225-2280-5.ch005.
17. Aman B. On the Efficiency of Synchronized P-Systems. *Journal of Membrane Computing,* 2022, Springer, ISSN: 2523-8914, DOI: 10.1007/s41965-021-00091-1.
18. Aman B., Ciobanu G. Synchronization of Rules in Membrane Computing. *Journal of Membrane Computing*, 1, 233-240, 2019, ISSN: 2523-8914, DOI: 10.1007/s41965-019-00022-1.
19. Ciobanu G., Pinna G.M. Memory Associated with Membranes Systems. *Journal of Membrane Computing*, 3, 116-132, 2021, ISSN: 2523-8914, DOI: 10.1007/s41965-020-00066-8.
20. Diaz-Pernil D., Gutierrez-Maranjo M.A., Peng H. Membrane computing and Image Processing: a Short Survey. *Journal of Membrane Computing*, 1(1), pp. 58-73, 2019, ISSN: 2523-8914, DOI: 10.1007/s41965-018-00002-x.
21. Sanchez-Karhunen E., Valencia-Cabrera L. Modelling Complex Market Interaction using PDP Systems. *Journal of Membrane Computing*, 1(1), pp. 40-51, 2019, ISSN: 2523-8914, DOI: 10.1007/s41965-019-00008-z.
22. Ababii V., Sudacevschi V., Munteanu S., Borozan O., Nistiriuc A., Lasco V. IoT based on Membrane Computing Models. In: *Proceedings of the 13th International Conference on Electromechanically and Energy Systems (SIELMEN-2021),* 7-8 October, 2021, Chisinau, Republic of Moldova, pp. 010-014, ISBN: 978-1-6654-0078-7. DOI: 10.1109/SIELMEN53755.2021.9600341. (IEEE Catalog Number: CFP21L58-ART).
23. Sosik P. P-Systems Attacking Hard Problems Beyond NP: a survey. *Journal of Membrane Computing,* 1(3), pp. 198-208, 2019, ISSN: 2523-8914, DOI: 10.1007/s41965-019-00017-y.
24. Munteanu S., Sudacevschi V., Ababii V., Borozan O., Ababii C., Lasco V. Multi-Agent Decision Making System based on Membrane Computing. In: *The 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computer systems: Technology and Applications.* 22-25 September, 2021, Cracow, Poland, Vol. 2. pp. 851-854. ISBN: 978-1-6654-4210-7, DOI: 10.1109/IDAACS53288.2021.9660971.
25. Orellana-Martin D., Valencia-Cabrera L., Riscos-Nunez A., Perez-Jimenez MJ. Minimal Cooperation as a way to Achieve the Efficiency in Cell-Like Membrane Systems. *Journal of Membrane Computing,* 1(2), pp. 85-92, 2019, ISSN: 2523-8914, DOI: 10.1007/s41965-018-00004-9.
26. Zhou N., Wang A. Fault Diagnosis of Transmission Circuit Based on Triangular Interval Valued Fuzzy Spike Neural Network P-System. *Energy Reports,* 2022, Vol. 8, pp. 776-784. DOI: 10.1016/j.egyr.2021.11.086.
27. Desel J., Esparsa J. *Free Choice Petri Nets*. Cambridge University Press, 1995, 244 p. ISBN: 0-521-46519-2.
28. Guțuleac E. *Modelarea și Evaluarea Performanțelor Sistemelor de Calcul prin Rețele Petri*. UTM, 1999, 267p.

29. Diaz M. *Petri Nets: Fundamental Models, Verification and Applications*. Wiley, 2009, 613p. ISBN: 978-1-84821-079-0, DOI: 10.1002/9780470611647.
30. JSON. [online]. [accessed 17.11.2021]. Available: https://www.json.org/json-en.html.
31. Bernardini F., Gheorghe M., Romero-Compero F., Walkinshaw N. A Hybrid Approach to Modeling Biological Systems. *WMC8 2007, LNCS 4860*, Springer-Verlag, pp. 138-159.
32. Liu F., Heiner M., Yang M. Modeling and Analyzing Biological Systems using Colored Hierarchical Petri Nets Illustrated by C. Elegans Vulval Development. *Journal of Biological Systems,* Vol. 22, No. 3, 2014, pp. 463-493, DOI: 10.1142/S0218339014500181.
33. Liu F., Heiner M. Modeling Membrane Systems using Colored Stochastic Petri Nets. *Natural Computing,* 2013, Nr. 12, pp. 617-629, DOI: 10.1007/s11047-013-9367-8.
34. Harris S.L., Harris D. *Digital Design and Computer Architecture. RISC-V Edition.* ScienceDirect 2021, 564p., ISBN: 978-0-12-820064-3, DOI: 10.1016/C2019-0-00213-0.
35. Intel Quartus Prime Software Suite. [online]. [accessed 21.12.2021]. Available: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/overview.html.
36. Cofer R.C., Harding B.F. *Rapid System Prototyping with FPGAs. Accelerating the Design Process*. ScienceDirect 2006, 301p., ISBN: 978-0-7506-7866-7, DOI: 10.1016/B978-0-7506-7866-7.X5000-8.
37. Guţuleac E., Boşneaga C., Railean A. VPNP-Software tool for modeling and performance evaluation using generalized stochastic Petri nets, *In: Proceedings of the 6-th International Conference on DAS-2002*, 23-25 May 2002, Suceava, România, p. 243-248, ISBN: 973-98670-9-X.
38. Altera DE0 Board. [online]. [accessed 11.01.2022]. Available: https://www.terasic.com.tw/cgi-bin/page/archive.pl?No=364.