The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

A RIGOROUS DESIGN METHODOLOGY FOR REPROGRAMMABLE LOGIC CONTROLLERS

Marian ADAMSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *M.Adamski@iie.pz.zgora.pl*

Abstract *In the paper it is shown how to implement parallel (concurrent) controllers in Field Programmable Logic (FPL). The main goal of the proposed design methodology is to preserve the direct, self-evident correspondence among a control interpreted Petri net and its possible hardware implementations. The symbolic specification of Petri net is considered in terms of the local state changes, which are distinguished by means of separated transitions, with their input and output places. Decision Rules are given as a set of Gentzen logic sequents (formal behavioural assertions). The initial specification, which is given in the form of symbolic logic expressions, may be verified, and then transformed into an intermediate format, which is accepted by industrial, VHDL- based, CAD tools. The logic (Boolean) expressions, which are suitable for direct mapping into FPGA or CPLD, can be also derived.*

Keywords*. Petri Nets, Logic Controllers, Hardware Description Languages, Field Programmable Logic, Sequential Function Chart*

1. INTRODUCTION

The well-structured formal specification, which is represented in the human-readable language, has a direct impact on the validation, formal verification and implementation of digital microsystems in Field Programmable Logic (FPL). The declarative, logic-based specification of Petri net can increase the efficiency of the Concurrent (Parallel) Controller design [1,4,8,14].

The model of *concurrent state machine* [4] is considered here inside the concept of *sequent automaton* and *parallel automaton* developed by Zakrevskij [16]. In the presented view, a control automaton, with distinguished, discrete and composite states, is also treated as a *dynamic inference system*, based on Gentzen logic [1]. The symbolic sequents-axioms may include some elements, taken from temporal logic [1,14]. The state space graph of controller is considered as a description of a discrete transition system [10]. Statements about the functionality of the designed system (behavioural axioms) are represented by means of *sequents-assertions*, forming the *rule-based decision system*. Complex sequents are formally

transformed into the set of equivalent *sequent-clauses* (*simple sequents* in [16]), which are very similar to the *production rules* [14].

At the beginning of design process, we use the control-oriented Petri net-based initial specifications of dedicated, reactive discrete-event systems (or Sequential Function Chart [3]). After analysis of some behavioural and structural properties of Petri net, a discrete-event model is related with a knowledge-based, textual, descriptive form of representation. The syntactic and semantic compatibility between Petri net descriptions and symbolic conditional assertions are kept as close, as possible. The formally transformed decision rules are directly mapped into VHDL statements. The Logic Controller is implemented in Field Programmable Logic, as a FPGA based reprogrammable unit. The automatic synthesis with VHDL oriented tools [2,3,10,15], as well as formal verification techniques and their efficiency [6,7,11,13,14,16], are out of the scope of the paper. The paper presents only the outline of formal methodology for Application Specific Logic Controllers (ASLC) design. The previous work on that subject has been recently summarised in papers [2,3,4].

2. DISCRETE EVENT CONTROL SYSTEM

As an example we have selected the simplified version of Logic Controller behaviour taken from papers [2, 3]. The chemical reactor V3 is fed with two kinds of liquids from measuring vessels V1 and V2. After the reaction between the liquids is completed, the reactor V3 is discharged. When the reactor V3 is empty, the process starts again from its initial state. To ensure complete reaction stirrer M agitates the process liquid in the reactor. Figure 1 shows a block diagram of the controlled system.

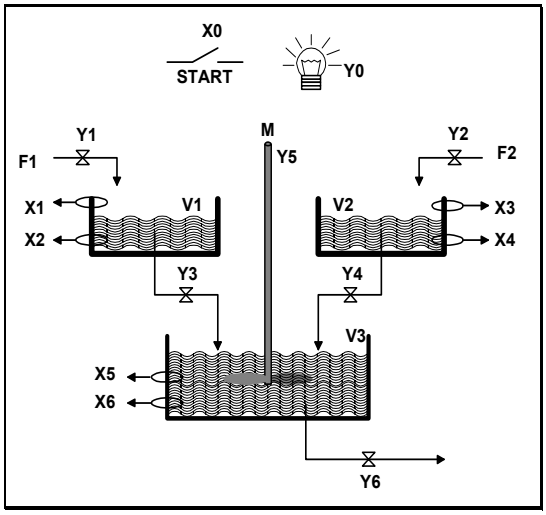

Fig. 1. The controlled part of discrete systems

3. BASIC THEORETICAL MODEL

Designing the discrete controller as a digital system involves a Petri net-based behavioural specification, taking into account the properties of the controlled objects (Figure 1). *Control Interpreted Petri Net* [6] has been shown to be a powerful tool to specify and model the behaviour of parallel (concurrent) controllers. The specification is given in terms of the local state changes. An event driven system can be abstracted as a concurrent state machine (CSM), in which several local states (represented as places in Petri net) may change, when event occurs (transition in Petri net fires). The marking (distribution of tokens among places) of a Petri net can be regarded as the current global state of the modelled system. From the present

global internal state (collection of simultaneously holding local states), the concurrent state machine goes to the next distributed internal global state, to generate the necessary combinational and registered output signals {y}. In such a way, an explicit local change of the marking, during the occurrence of transition, corresponds to an implicit global state change. The colours, which are attached to places [15], distinguish the intended sequential processes

The synchronous Petri nets [10,12] are introduced to model binary systems, which are synchronised by a global clock. Input signals of controller are associated with transitions as a Boolean guard. The most common static Moore type output signals are linked with places. Some static Mealy type output signals are related both with places and input signals. A part of Mealy type outputs frequently coincidence with the firing of transition. That makes it possible to label net transitions with some particular dynamic signal names.

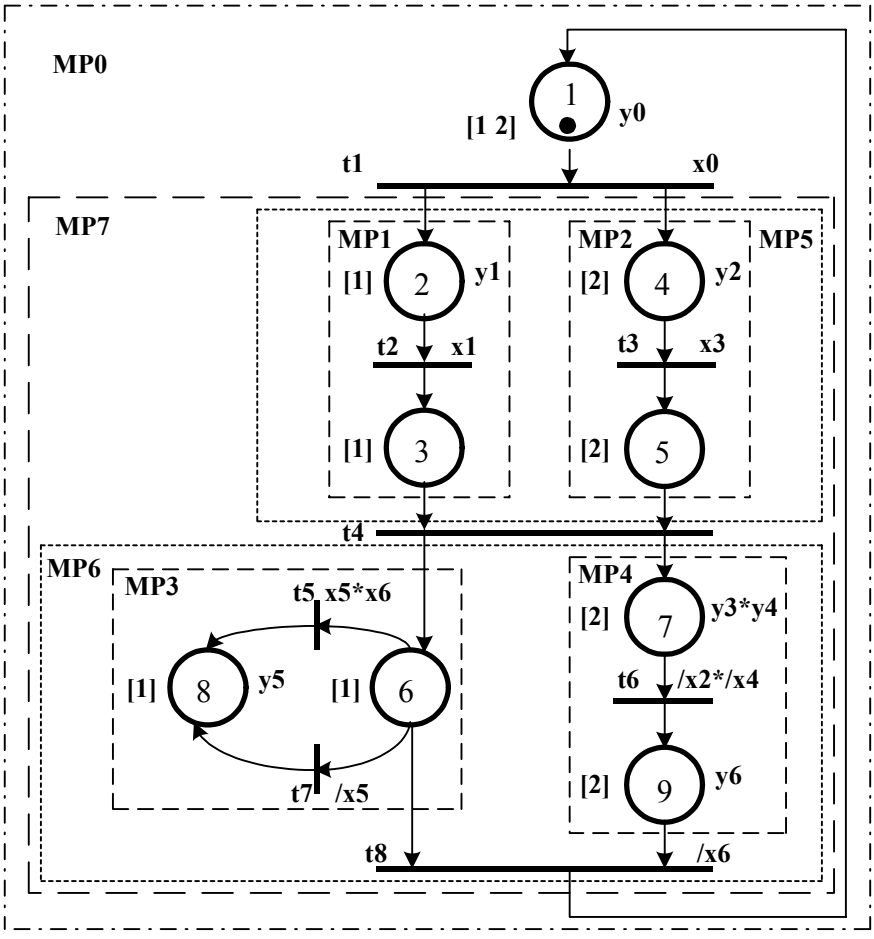

Fig. 2. Modular behavioural specification by means of Petri nets

To obtain the economical implementation and easy maintenance, the Petri net may be directly mapped into the Boolean equations without explicit enumeration of all possible global states and all possible global state changes. Since the specification is given only in terms of the local state changes (local transitions), the structured local state assignment (place encoding) is used [1,16].

Petri nets provide a unified method for the design of discrete-event systems from a hierarchical system description (Figure 2) to possibly hierarchical physical realizations. The hierarchically structured Petri net consists of subnets, which, except possibly the Base Net are well-formed blocks. The concurrency relation between subnets is depicted by means of colours, which are attached to the explicitly to the places, and implicitly to the transitions and

arcs as well as to the tokens [15]. The set of subnets is partially ordered. The coloured hierarchy relation tree (Figure 3) graphically represents the hierarchy and concurrency relations among subnets. The Base Net *MP0* is on the root of the tree. It contains the double-macroplaces *MP1-MP7*, which stand for the hierarchically structured subnets at the lower level of hierarchy. Each double macroplace (called shortly *double*) corresponds to a compound operation, which is itself a discrete sub-process described by the doubled block. The colours *[1]* and *[2]* are used for distinguishing some particular intended sequential processes, and continuously controlling the place invariants (P-subnets) and hierarchy tree during the composition or reduction of the net. The Petri net (Figure 3) is hierarchically encoded by means of state variables *Qi*, *i= 1,2,3,4*. The symbols *Qi* or */Qi,* attached to the particular path, which is directed from the root to the leave, form the unique encoding term for the considered macroplace or place.

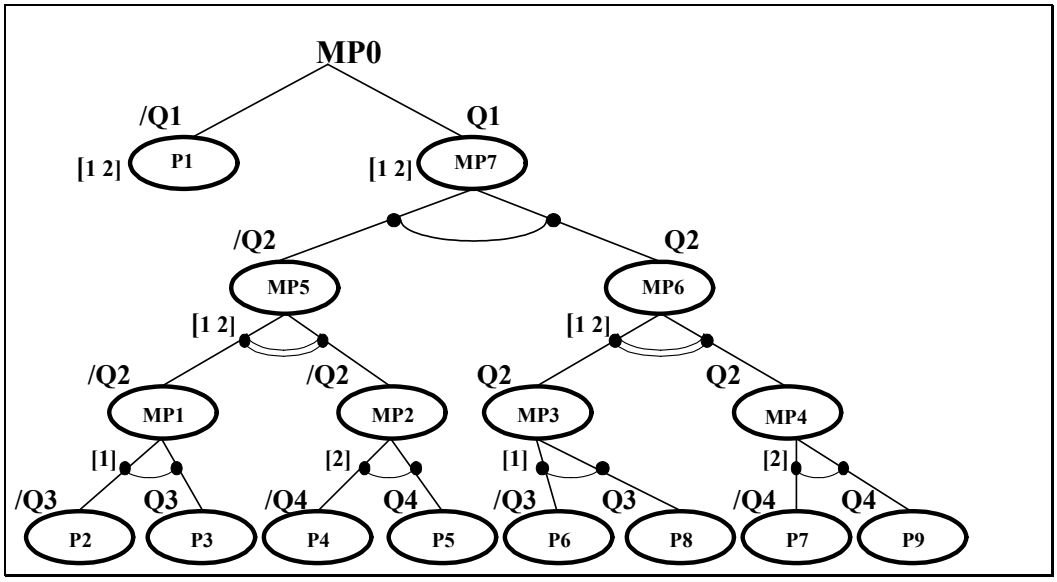

Fig. 3. Hierarchy tree

4. GENTZEN SEQUENT LOGIC

Petri nets can be viewed as a formal model for logic rule-based specification (interpretation structure). Transition rules are usually treated as production rules ('if-then' non-procedural statements). The principal design language used to specify the Logic Controller behaviour in extended nested *If-Then-Else* form in our design environment is Gentzen Sequent Logic [9].

While formulae may be regarded as a formal representation of compound propositions, sequents in our approach represent asserted statements. Sequents may formally describe all general forms of conditional assertions (rules). The Gentzen formal system naturally simulates and records human-like reasoning. The synthesis, based on Gentzen calculus, is treated as a formal symbolic transformation of the initial set of sequents (specification) into another equivalent set of sequents (implementation) [1,2]. The rules of inference are directly based on Gentzen Logic or they are previously proven, so the implementations are correct by construction.

Complex sentences are built up from propositions by application of the propositional connectives: *not (/)*, *and (*)*, *or (+)*, *if-then (->)*, *if-and-only-if (<->)* and *if-then-else (-> |*).

If F, *G*, and *H* are sentences then expression *F->G|H* (*if F then G else H*) is called *conditional*. The implication *F->G (if F then G)* is a special case of the conditional. We call *F*

the antecedent and *G* the consequent of implication. In our particular application (rule-based description) *F* is usually a conjunction of simple propositions, *G* and *H* are possibly nested conditionals, or *G* and *H* are conjunctions of simple propositions. The most frequently used form of simple decision rule is an *asserted implication* |- *F->G*, in which both *F* and *G* are elementary conjunctions. According to Gentzen the simple decision rule can be represented also as an equivalent sequent *F*|- *G*.

The complex conditional assertion (Complex Decision Rule) is described as a sequent:

|- *F->G | H;*

It encloses possibly nested components *G* and *H*. From the Gentzen logic point of view it is a shorter form of a complex sequent |-*(F->G)*(/F->H).*

Each Gentzen *inference figure* (*inference rule)* applied to the sequents (transition rules) preserves the meaning of the specification, but changes the number of the sequents and a form, in which sequents are expressed. If a specification is correct, transformation by means of natural inference guarantees correctness by construction.

As a simple example of formal transformation we consider the declarative assertion:

|- *(F- >G)* (/F->H);*

It can be transformed into an equivalent set of two simpler sequents (assertions) by splitting the right side

|- (F- >G) (/F->H);*

———————————— (Rule |- *)

|- (F->G); |- (/F->H);

The final assertions (simple Decision Rules) are as follows:

F |-G;

/F |-H;

5. PETRI NET SPECIFICATION IN SEQUENT LANGUAGE

The Logic Controller is considered as an abstract reasoning system (rule based system) implemented in reconfigurable hardware. The mapping between inputs, outputs and local internal states of the system is described in a formal manner by means of logic rules (represented as sequents) with some temporal operators, especially with operator 'next' @ [1,11,14]. The correctness preserving synthesis, based on Gentzen calculus, is treated as a formal transformation of the initial set of compound rules (*Specification*) into another set of compound rules (*Implementation)*.

As a basic form of Petri net specification in rule based format the *transition-oriented declarative specification* is presented. It describes all possible active events in concurrent state machine, when local states are changed, and the guard (Boolean label) associated to transition is true.

```
T1: P1 * X0            |-    @P2 *@P4;
T2: P2 * X1            |-    @P3;
T3: P4 * X3            |-    @P5;
T4: P3 * P5            |-    @P6 * @P7;
T5: P6 * X5*X6         |-    @P8;
T6: P7 * /X2*/X4       |-    @P9;
T7: P8 * /X5           |-    @P6;
T8: P6 *P9 * /X6       |-    @P1;
```

The static (level) Moore type outputs depend directly on places:

```
P1 |- Y0;   P2 |- Y1;   P4 |- Y2;   P7 |- Y3 * Y4;
P8 |- Y5;   P9 |- Y6;
```

The total discrete state space (9 global states), which could be also possibly given in hierarchical manner, should be always consistent with all local state changes:

|-**P1***/P2*/P3*/P4*/P5*/P6*/P7*/P8*/P9, /P1***P2***/P3***P4***/P5*/P6*/P7*/P8*/P9,…,
/P1*/P2*/P3*/P4*/P5*/P6***P7****P8***/P9, /P1*/P2*/P3*/P4*/P5*/P6*/P7***P8****P9**

The presented form of description is very closed to well-known production rules, whose are a principal forms of Petri net description in LOGICIAN [1], CONPAR [8,10], PARIS [12], and PeNCAD [3,15,].

The dynamic (pulse or registered) output signal can be included directly to the decision rule, when it changes its value together with the occurrence of transition. On the other hand, all changes of the place making could be also explicitly included into the sequent, for example:

```
T1: P1 * X0  |-   @P2 @P4*/@P1*@/Y0*@Y1*@Y2;
```

In some cases, like implementations with D flip-flops in FPGA, the declarative, *place oriented specification* is taken into account:

```
P1:   P1    |- X0 -> @P2 * P4        | @P1;
P2:   P2    |- X1 -> @P3             | @P2;
P3:   P3    |- P5 -> @P6             | @P3;
P4:   P4    |- X3 -> @P5             | @P4;
P5:   P5    |- P3 -> @P7             | @P5;
P6:   P6    |- P9 * /X6 -> @P1       | (X5*X6 -> @P8 | @P6);
P7:   P7    |- /X2*/X4 -> @P9        | @P7;
P8:   P8    |- /X5 -> @P6            | @P8;
P9:   P9    |- P6*/X6->@P1           | @P9;
```

In this kind of specification, if the next value of the temporal variable, for example *@P1*, cannot be proved in the current marking (global state) as *true,* it is considered that it takes the value *false.*

When control outputs are immediate (combinational), they may be associated with the appropriate places and later transformed into registered ones. It should be noted that in many cases registered outputs could be used not only as names of places, but also directly as *codes* of the associated places

The sequents with transition symbols *{T1, T2, … , T8}*, after mapping the Petri net into VHDL statements according to M. Bolton's style, give economical implementations in FPGA [8]:

```
P1 * X0 |- T1
P2 * X1 |- T2
........
T1+P2*/T2 |- @P2
```

6. PETRI NET MODELLING AND SYNTHESIS WITH VHDL

The direct mapping of a Petri net into Field Programmable Logic (FPL) is based on a self-evident correspondence between a place and a clearly defined bit-subset of a state register. The places of the Petri net are assigned to the particular flip-flops in the Register Block. VHDL supports conditional-statement constructs, which can be used to describe Petri net. The proper local state assignment (encoding) makes it possible to map a given Interpreted Petri net directly into FPGA or CPLD without its transformation into an equivalent global State Machine. The simplest technique for Petri net place encoding is to use one-to-one mapping of places onto flip-flops in the style of a one-hot state assignment. In that case, a name of the place becomes also a name of the related flip-flop. The flip-flop is set into 1 if and only if the

particular place holds the token. Some of the recent developments involving modelling and analysis such constructs in VHDL were reported, for example in [2,3,8,10,15].

In general, places after encoding are distinguished by conjunctions, which are formed from state variables from the set *{Q1, Q2, ... , Qk}*. The local states, which are active simultaneously, have non-orthogonal codes. They are represented by places holding the tokens concurrently and belonging to the same vertex from the implicitly or explicitly given reachability graph of Petri net. The local states, which belong to the different, but sometimes overlapping sequential processes (P-invariants, SM-components) have orthogonal codes. One particular method of place encoding is based on hierarchical decomposition of the net. The example of an efficient heuristic hierarchical local state assignment `[Q1, Q2, Q3, Q4]` is as follows:

```
P1  = 0 - - -                    P1 = /Q1
MP7 = 1 * * *                    MP7= Q1
MP5 = 1 0 * *                    MP5= Q1*/Q2
MP6 = 1 1 * *                    MP6= Q1*Q2
MP1 = 1 0 * *                    MP1= Q1*/Q2
MP2 = 1 0 * *                    MP2= Q1*/Q2
MP3 = 1 1 * *                    MP3= Q1*Q2
MP4 = 1 1 * *                    MP4= Q1*Q2

P2  = 1 0 0 *                    P2= Q1*/Q2*/Q3
P3  = 1 0 1 *                    P3= Q1*/Q2*Q3
P4  = 1 0 * 0                    P4= Q1*/Q2*/Q4
P5  = 1 0 * 1                    P5= Q1*/Q2*Q4
P6  = 1 1 0 *                    P6= Q1*Q2*/Q3
P7  = 1 1 * 0                    P7= Q1*Q2*/Q4
P8  = 1 1 1 *                    P8= Q1*Q2*Q3
P9  = 1 1 * 1                    P9= Q1*Q2*Q4
```

The global state encoding is correct if all vertices of the reachability graph have different codes. The total code of the reachability graph vertex would be obtained by merging the codes of the simultaneously marked places. The code of the particular place or macroplace is represented by means of the vector composed from {0, 1, - , *}, or it is given as a related Boolean term. The symbols 0, 1, - ('don't care') have the usual meanings, but the symbol * in vector denotes 'explicitly don't know' (0 or 1, but not 'don't care'). For practical applications (CAD tools) it is usually sufficient to manipulate with Boolean expressions (product terms) [1,2,3,15].

7. CONCLUSIONS

Formal logic language, which is complementary with Petri nets, is suitable in specifying system level designs of logic controllers, implemented in FPL. Simulating of Petri net model and its hardware implementation can be simplified by translating of rule-based description to VHDL. The simulation results, at circuit level and algorithmic level, can be compared immediately. To simulate the pair consisting of the controller and discrete object under control, the test bench must include, in addition to the Reprogrammable Controller description, a second VHDL program, which model the controlled subsystem behaviour. The next design step concentrates on the automatic synthesis of Reprogrammable logic Controllers from their VHDL description. The paper presents the hierarchical Petri net approach for synthesis, in which the modular net is mapped into the Field Programmable logic as structured, but a flat netlist. The hierarchy levels are conserved and related with some particular local state variable subsets, and clearly distinguished by the encoding vectors (encoding terms). A concise, understandable specification can be easily locally modified. The

experimental Petri net to VHDL translator has been implemented on the top of standard VHDL design tools, like ALDEC Active-HDL.

REFERENCES

[1] M. Adamski, "Parallel Controller Implementation using Standard PLD Software". In: *FPGAs*, W.R. Moore, W. Luk (Ed.), Abingdon EE&CS Books, Abingdon, England, 1991, Chapter 5.5, pp.296-304.

[2] M. Adamski M., J. L. Monteiro, "From Interpreted Petri Net Specification to Reprogrammable Logic Controller Design", *Intern. Symposium on Industrial Electronics ISIE'2000*, 4-8 Dec. 2000, Puebla (Mexico), Vol. 1, pp.13-19.

[3] M. Adamski, "SFC, Petri Nets and Application Specific Logic Controllers". *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybern.*, San Diego, USA, 1998, pp.728-733.

[4] M. Adamski, A.D.Zakrevskij, "Rule-Based Specification of Reactive Logical Control Devices", *Proceedings of the Polish-German Symposium on Science Research Education*, SRE'2000, Zielona Gora, 28-29.Sept.2000, Vol.1, pp. 199-204.

[5] K. Bilinski, M. Adamski, J.M. Saul, E. L. Dagless, "Petri net based algorithms for parallel controller synthesis", *IEE Proceedings-E, Computers and Digital Techniques,* Vol. 141, No. 6, Nov. 1994, pp. 405-412.

[6] R. David, H. Alla, *Petri Nets & Grafcet. Tools for modelling discrete event systems*, Prentice Hall, New York, 1992.

[7] W. Fengler, A. Wendt, M. Adamski, J.L. Monteiro, "Petri Net based Program Design and Implementation for Controller Systems", *1996 IFAC Triennial World Congress*, San Francisco, CA, USA, Vol. J, Identification II, Discrete Event Systems, pp.425-429.

[8] J. M. Fernandes, M. Adamski, A.J. Proença, "VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controllers", *IEE Proc.-E, Computer and Digital Techniques*, Vol.144, No.2, 1997, pp.127-137.

[9] M. E. Szabo (Ed.), *The collected papers of Gerhard Gentzen*, North Holland Publishing Company, Amsterdam, 1969.

[10] A. Yakovlev, L. Gomes, L. Lavagno (Ed.), *Hardware Design and Petri Nets*, Kluwer Academic Publishers, Boston, 2000.

[11] M. Heiner, "Petri Net Based System Analysis without State Explosion", *Proc. High Performance Computing'98*, Boston, April 1998, SCS Int., San Diego, 1988, pp.394-403

[12] T. Kozlowski, E. L. Dagless, J.M. Saul, M. Adamski, J. Szajna "Parallel controller synthesis using Petri nets", *IEE Proc.-E, Computers and Digital Techniques*, Vol. 142, No. 4, 1995, pp. 263-271.

[13] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol.77, No. 4, April 1989, pp. 541-580.

[14] J.S. Sagoo, D.J. Holding, "A comparison of temporal Petri net based techniques in the specification and design of hard real-time systems", *Microprocessing and Microprogramming*, Vol. 32, No. 1-5, 1991, pp. 111 - 118.

[15] M. Wegrzyn, P. Wolanski, M. Adamski, J.L. Monteiro, "Field Programmable Device as a Logic Controller", *Proc. of the 2nd Conf. on Automatic Control - Control'96*, Oporto, Portugal, 1996, Vol. 2, pp.715-720.

[16] A.D. Zakrevskij, "Parallel algorithms for logical control". Inst. of Engineering Cybern. of NAS of Belarus, Minsk, 1999 (book in Russian).