

## Chapter 2

# FORMAL LOGIC DESIGN OF REPROGRAMMABLE CONTROLLERS

Marian Adamski

*University of Zielona Góra, Institute of Computer Engineering and Electronics,  
ul. Podgorna 50, 65-246 Zielona Góra, Poland; e-mail: M.Adamski@iie.uz.zgora.pl*

**Abstract:** The goal of the paper is to present a formal, rigorous approach to the design of logic controllers, which are implemented as independent control units or as central control parts inside modern reconfigurable microsystems. A discrete model of a dedicated digital system is derived from the control interpreted Petri net behavioral specification and considered as a modular concurrent state machine. After hierarchical and distributed local state encoding, an equivalent symbolic description of a sequential system is reflected in field programmable logic by means of commercial CAD tools. The desired behavior of the designed reprogrammable logic controller can be validated by simulation in a VHDL environment.

**Key words:** Petri nets; logic controllers; hardware description languages (HDL); field programmable logic.

## 1. INTRODUCTION

The paper covers some effective techniques for computer-based synthesis of reprogrammable logic controllers (RLCs), which start from the given interpreted Petri net based behavioral specification. It is shown how to implement parallel (concurrent) controllers<sup>1,4,8,14</sup> in field programmable logic (FPL). The symbolic specification of the Petri net is considered in terms of its local state changes, which are represented graphically by means of labeled transitions, together with their input and output places. Such simple subnets of control interpreted Petri nets are described in the form of decision rules – logic assertions in propositional logic, written in the Gentzen sequent style<sup>1,2,12</sup>.

Formal expressions (sequents), which describe both the structure of the net and the intended behavior of a discrete system, may be verified formally in

the context of mathematical logic and Petri net theory. For professional validation by simulation and effective synthesis, they are automatically transformed into intermediate VHDL programs, which are accepted by industrial CAD tools.

The main goal of the proposed design style is to continuously preserve the direct, self-evident correspondence between modular interpreted Petri nets, symbolic specification, and all considered hierarchically structured implementations of modeled digital systems, implemented in configurable or reconfigurable logic arrays.

The paper presents an extended outline of the proposed design methodology, which was previously presented in *DESDes'01* Conference Proceedings<sup>3</sup>. The modular approach to specification and synthesis of concurrent controllers is applied, and a direct hierarchical mapping of Petri nets into FPL is demonstrated. The author assumes that the reader has a basic knowledge of Petri nets<sup>5,9,10,13,14</sup>. The early basic ideas related with concurrent controller design are reported in the chapter in Ref. 1. The author's previous work on reprogrammable logic controllers has been summarized in various papers<sup>2,4,6,8</sup>. Several important aspects of Petri net mapping into hardware are covered in books<sup>3,13,14</sup>. The implementation of Petri net based controllers from VHDL descriptions can be found in Refs. 2, 6, and 13. Some arguments of using Petri nets instead of linked sequential state machines are pointed in Ref. 9.

## 2. CONCURRENT STATE MACHINE

In the traditional *sequential finite state machine* (SFSM) model, the logic controller changes its *global internal states*, which are usually recognized by their mnemonic names. The set of all the possible internal states is finite and fixed. Only one *current state* is able to hold (be active), and only one *next state* can be chosen during a particular *global state change*. The behavioral specification of the modeled sequential logic controller is frequently given as a state graph (diagram) and may be easily transformed into state machine–Petri net (SM-PN), in which only one current *marked place*, representing the active state, contains a *token*. In that case, the state change of controller is always represented by means of a *transfer transition*, with only one input and only one output place. The traditional single SFSM based models are useful only for the description of simple tasks, which are manually coordinated as linked state machines with a lot of effort<sup>9</sup>. The equivalent SFSM model of highly concurrent system is complicated and difficult to obtain, because of the state space explosion.

In the modular Petri net approach, a *concurrent finite state machine* (CFSM) simultaneously holds several *local states*, and several local state changes can

occur independently and concurrently. The *global states* of the controller, included into the equivalent SFSM model, can be eventually deduced as maximal subsets of the local states, which simultaneously hold (*configurations*). They correspond to all different maximal sets of marked places, which are obtained during the complete execution of the net. They are usually presented in compact form as vertices in Petri net reachability graph<sup>5,10,13</sup>. It should be stressed that the explicitly obtained behaviorally equivalent *transition systems* are usually complex, both for maintenance and effective synthesis. The methodology proposed in the paper makes it possible to obtain a correctly encoded and implemented transition system directly from a Petri net, without knowing its global state set.

The novel tactic presented in this paper is based on a hierarchical decomposition of Petri nets into self-contained and structurally ordered modular subsets, which can be easily identified and recognized by their common parts of the internal state code. The total codes of the related modular Petri net subnets, which are represented graphically as *macroplaces*, can be obtained by means of a simple hierarchical superposition (merging) of appropriate codes of individual places. On the other hand, the code of a particular place includes specific parts, which precisely define all hierarchically ordered macroplaces, which contain the considered place inside. In such a way any separated part of a behavioral specification can be immediately recognized on the proper level of abstraction and easily found in the regular cell structure (logic array). It can be efficiently modified, rejected, or replaced during the validation or redesign of the digital circuit.

Boolean expressions called *predicate labels* or *guards* depict the external conditions for transitions, so they can be enabled. One of enabled transition occurs (it *fires*). Every immediate *combinational Moore type output signal*  $y$  is linked with some *sequentially related places*, and it is activated when one of these places holds a token. Immediate *combinational Mealy type output signals* are also related with proper subsets of sequentially related places, but they also depend on relevant (valid) input signals or internal signals. The Mealy type output is active if the place holds a token and the correlated logic conditional expression is true.

The implemented Petri net should be determined (without conflicts), safe, reversible, and without deadlocks<sup>5,7</sup>. For several practical reasons the synchronous hardware implementations of Petri nets<sup>4,6,7</sup> are preferred. They can be realized as dedicated digital circuits, with an internal state register and eventual output registers, which are usually synchronized by a common clock. It is considered here that all enabled concurrent transitions can fire independently in any order, but nearly immediately.

In the example under consideration (Fig. 2-1), Petri net places  $P = \{p1-p9\}$  stand for the local states  $\{P1-P9\}$  of the implemented logic controller. The

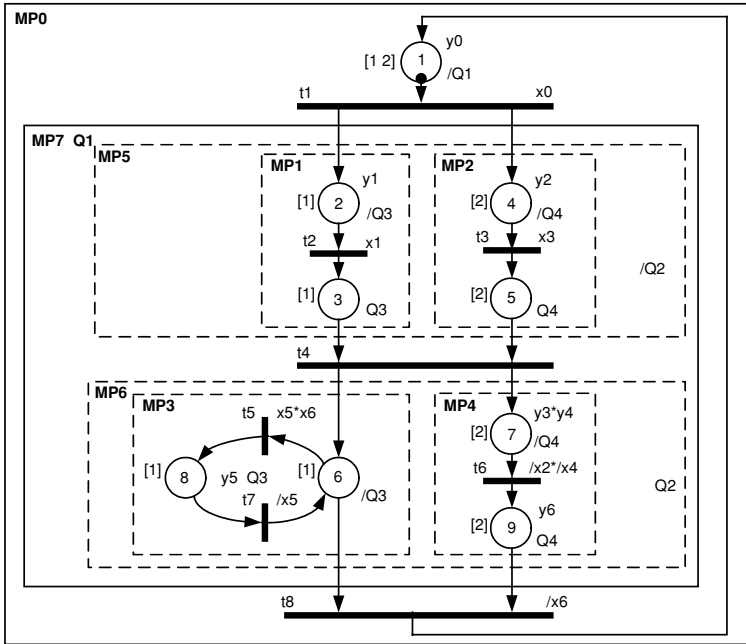


Figure 2-1. Modular, hierarchical and colored control interpreted Petri net.

Petri net transitions  $T = \{t_1-t_8\}$  symbolize all the possible *local state changes*  $\{T_1-T_9\}$ . The Petri net places are hierarchically grouped as nested modular macroplaces **MP0-MP7**. The Petri net describes a controller with inputs  $x_0-x_6$  and outputs  $y_0-y_6$ . The controller contains an internal state register with flip-flops **Q1-Q4**. The state variables structurally encode places and macroplaces to be implemented in hardware.

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 place of the Petri net is assigned only to the particular part of the register block (only to selected variables from internal state register **Q1-Q4**). The beginning of local state changes is influenced by the edge of the clock signal, giving always, as a superposition of excitations, the predicted final global state in the state register. The high-active input values are denoted as  $x_i$ , and low-active input values as  $/x_i$ .

The net could be SM-colored during the specification process, demonstrating the paths of recognized intended sequential processes (state machines subnets). These colors evidently help the designer to intuitively and formally validate the consistency of all sequential processes in the developed discrete state model<sup>4</sup>. The colored subnets usually replicate Petri net place invariants. The invariants of the top-level subnets can be hierarchically determined by invariants of its

subnets. If a given net or subnet has not been previously colored by a designer during specification, it is possible to perform the coloring procedure by means of analysis of configurations. Any two concurrent places or macroplaces, which are marked simultaneously, cannot share the same color. It means that coloring of the net can be obtained by coloring the *concurrency graph*, applying the well-known methods taken from the graph theory<sup>1,2</sup>. For some classes of Petri nets, the concurrency relation can be found without the derivation of all the global state space<sup>4,7,13</sup>. The colors  $([1], [2])$ , which paint the places in Fig. 2-1, separate two independent sequences of local state changes. They are easy to find as closed chains of transitions, in which selected input and output places are painted by means of identical colors.

The equivalent interpreted SM Petri net model, derived from equivalent transition system description (interpreted Petri net reachability graph of the logic controller), is given in Fig. 2-2.

The distribution of Petri net tokens among places, before the firing of any transition, can be regarded as the identification of the *current global state M*. Marking  $M$  after the firing of any enabled transition is treated as the *next global*

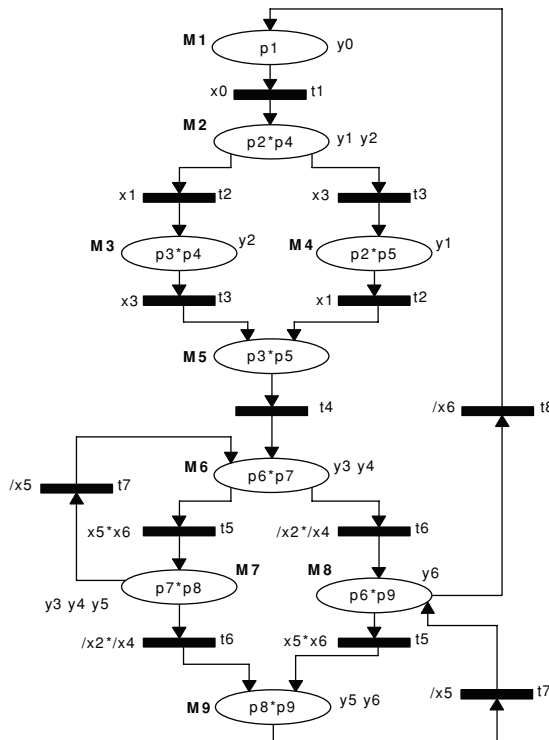


Figure 2-2. Global states of control interpreted Petri net. Transition system modeled as equivalent state machine Petri net.

*state @M*. From the present global internal state *M*, the modeled controller goes to the next internal global state @M, generating the desired combinational immediate output signals *y* and registered @*y* output signals.

There are 9 places describing global states M1–M9 and 13 transitions between 13 pairs of global states. Such an implicit formal structure really exists in hardware, although its internal structure could be unknown, because eventually deduced global state codes are immediately read from the state register as a consistent superposition of local state codes. Since a Moore-type output should be stable during the entire clock period, it can also be produced as a *registered Moore-type output @y*. The registered Moore-type output signals should be predicted before the local state changes.

### 3. LOGIC CONTROLLER AS ABSTRACT REASONING SYSTEM IMPLEMENTED IN DIGITAL HARDWARE

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 FPL. The declarative, logic-based specification of the Petri net can increase the efficiency of the concurrent (parallel) controller design. The proposed model of a concurrent state machine can be considered within a framework of the concept of sequent parallel automaton, developed by Zakrevskij<sup>3,14</sup>. Here, a control automaton, with discrete elementary states and composite super-states, is treated as a dynamic inference system, based on Gentzen sequent logic<sup>1,2,12</sup>.

After analysis of some behavioral and structural properties of the Petri net<sup>5,7,10,13,14</sup>, 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 symbolic sequents-axioms may include elements, taken from temporal logic, especially operator “next” @<sup>11</sup>. Statements about the discrete behavior of the designed system (behavioral axioms) are represented by means of sequents-assertions, forming the rule-base of the decision system, implemented in reconfigurable hardware. Eventual complex sequents are formally, step by step, transformed into the set of the equivalent sequent-clauses, which are very similar to elementary sequents<sup>1,3,14</sup>. The simple *decision rules*, which are transformed into reprogrammable hardware, can be automatically mapped into equivalent VHDL statements on RTL level<sup>2,6,13</sup>. The next steps of design are performed by means of professional CAD tools.

The implicit or explicit interpreted reachability graph of the net is considered here only as a conceptual supplement: compact description of an equivalent discrete transition system (Fig. 2-2), as well as Kripke interpretation structure<sup>11</sup> for symbolic logic conditionals.

#### 4. STRUCTURED LOCAL STATE ENCODING

The simplest technique for Petri net place encoding is to use one-to-one mapping of places onto flip-flops in the style of one-hot state assignment. In that case, a name of the place becomes also a name of the related flip-flop. The flip-flop  $Q_i$  is set to 1 if and only if the particular place  $p_i$  holds the token. In such a case it is a popular mistake to think that other state variables  $Q_j, \dots, Q_k$  have simultaneously “*don't care values*.” It is evidently seen from the reachability graph (Fig. 2-2) that the places *from the same P-invariant*, which are sequentially related with the considered place  $p_i$ , do not hold tokens, and consequently all flip-flops used for their encoding have to be set to logical 0. On the other hand, the places from the same configuration but with different colors, which are concurrently related with the selected place  $p_i$ , have strictly defined, but not necessarily explicitly known, markings. The only way of avoiding such misunderstanding of the *concurrent one-hot encoding* is the assumption that the considered place, marked by token, can be *recognized* by its own, private flip-flop, which is set to logical 1, and that signals from other flip-flops from the state register are always fixed but usually unknown.

In general encoding, places are recognized by their particular coding conjunctions<sup>1</sup>, which are formed from state variables, properly chosen from the fixed set of flip-flop names  $\{Q_1, Q_2, \dots, Q_k\}$ . The registered output variables  $\{Y\}$  can be eventually merged with the state variable set  $\{Q\}$  and economically applied to the Petri net place encoding as local state variables. For simplicity, the encoded and implemented place  $p_i$  is treated as a complex signal  $P_i$ .

The local states, which are simultaneously active, must have nonorthogonal codes. It means that the Boolean expression formed as a conjunction of *coding terms* for such concurrent places is always satisfied (always different from logical 0). The configuration of concurrent places gives as superposition of coding conjunctions a unique code of the considered global state.

The local states, which are not concurrent, consequently belong to at least one common sequential process (Figs. 2-2, 2-3). Their symbols are not included in the same vertex of the reachability graph, so they may have orthogonal codes.

The code of a particular place or macroplace is represented by means of a vector composed of  $\{0, 1, \dots, *\}$ , or it is given textually as a related Boolean term. The symbols of the values for logic signals 0, 1, and “don't care” have the

usual meanings. The symbol \* in the vector denotes “explicitly don’t know” value (0 or 1, but no “don’t care”). In expressions, the symbol / denotes the operator of logic negation, and the symbol \* represents the operator of logic conjunction. An example<sup>3</sup> of a heuristic hierarchical local state assignment [ $Q1$ ,  $Q2$ ,  $Q3$ ,  $Q4$ ] is as follows:

$P1[1, 2]$	=	0	-	-	-	$QP1 =$	$/Q1$
$P2[1]$	=	1	0	0	*	$QP2 =$	$Q1 * /Q2 * /Q3$
$P3[1]$	=	1	0	1	*	$QP3 =$	$Q1 * /Q2 * Q3$
$P4[2]$	=	1	0	*	0	$QP4 =$	$Q1 * /Q2 * /Q4$
$P5[2]$	=	1	0	*	1	$QP5 =$	$Q1 * /Q2 * Q4$
$P6[1]$	=	1	1	0	*	$QP6 =$	$Q1 * Q2 * /Q3$
$P7[2]$	=	1	1	*	0	$QP7 =$	$Q1 * Q2 * /Q4$
$P8[1]$	=	1	1	1	*	$QP8 =$	$Q1 * Q2 * Q3$
$P9[2]$	=	1	1	*	1	$QP9 =$	$Q1 * Q2 * Q4$

The global state encoding is correct if all vertices of the reachability graph have different codes. The total code of the global state (a vertex of the reachability graph) can be obtained by merging the codes of the simultaneously marked places. Taking as an example some global states (vertices of the reachability graph; Fig. 2-2), we obtain

$$QM3 = QP3 * QP4 = Q1 * /Q2 * Q3 * /Q4;$$

$$QM4 = QP2 * QP5 = Q1 * /Q2 * /Q3 * Q4.$$

## 5. MODULAR PETRI NET

Modular and hierarchical Petri nets can provide a unified style for the design of logic controllers, from an initial behavioral system description to the possibly different hierarchical physical realizations. The concurrency relation between subnets can be partially seen from the distribution of colors. Colors<sup>5</sup> are attached explicitly to the places and macroplaces, and implicitly to the transitions, arcs, and tokens<sup>2,3,4</sup>. Before the mapping into hardware, the Petri net is hierarchically encoded. The Petri net from Fig. 2-2 can be successfully reduced to one compound multiactive macroplace  $MP0$ . The colored hierarchy tree in Fig. 2-3 graphically represents both the hierarchy and partial concurrency relations between subnets (modules). It contains a single ordinary monoactive place  $P1[1,2]$ , coded as  $QP1 = /Q1$ , and a multiactive double-macroplace  $MP7[1,2]$ , coded as  $QMP7 = Q1$ , which stands for other hierarchically nested subnets, from lower levels of abstraction containing places  $p1$ – $p9$ . The macroplace  $MP7[1,2]$  is built of the sequentially related macroplaces  $MP5[1,2]$  and  $MP6[1,2]$ , which are coded respectively as  $Q1 * /Q2$  and  $Q1 * Q2$ .



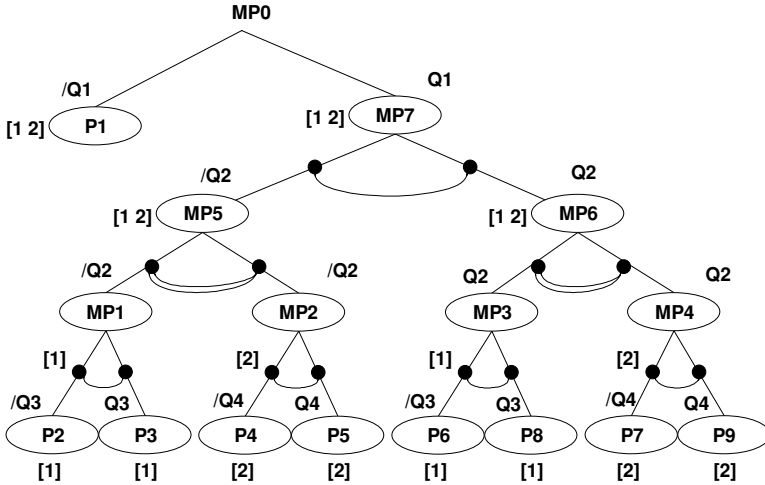


Figure 2-3. Hierarchy tree.

It should be mentioned that parallel macroplaces may obtain exactly the same top-level codes. The macroplace  $MP5[1,2]$  consists of two parallel macroplaces  $MP1[1]$  and  $MP2[2]$ , which are not recognized by different conjunctions inside  $MP5$ . The macroplace  $MP6[1,2]$  appears as an abstraction of two other parallel macroplaces  $MP3[1]$  and  $MP4[2]$ . The macroplaces  $MP1[1]$ ,  $MP2[2]$ ,  $MP3[1]$ , and  $MP4[2]$  are directly extracted from the initial Petri net as elementary sequential subnets.

The concurrency relation between subnets, which belong to the same macroplace, can be expressed graphically by means of additional double or single lines (Fig. 2-3). The code of the macroplace or place on a lower level of hierarchy is obtained by means of superposition of codes, previously given to all macroplaces to which the considered vertex belongs hierarchically. Taken as an example, the code of place  $p_2$  is described as a product term  $QP2 = Q1*/Q2*/Q3$ .

## 6. PETRI NET MAPPING INTO LOGIC EXPRESSIONS

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 the operator “next”  $@^{1,2}$ . As an example of a basic

style of the textual controller description, the *transition-oriented declarative specification* is presented.

The declarative style of description is close to well-known production rules, which are principal forms of Petri net specification in LOGICIAN<sup>1</sup>, CONPAR<sup>6</sup>, PARIS<sup>4,8</sup>, and PeNCAD<sup>2,3</sup>. It should be noted that here the names of transitions T1, T2, . . . , T8 serve only as decision rule labels, keeping the easy correspondence between Petri net transitions and their textual logic descriptions. The symbol  $\mid$ — denotes “yield,” the symbol  $*$  stands for the logic conjunction operator, and  $/$  stands for negation.

```

T1[1,2] : P1[1,2] * X0 | -@P2[1] * @P4[2];
T2[1]   : P2[1] * X1 | -@P3[1];
T3[2]   : P4[2] * X3 | -@P5[2];
T4[1,2] : P3[1] * P5[1] | -@P6 * @P7;
T5[1]   : P6[1] * X5*X6 | -@P8[1];
T6[2]   : P7[2] * /X2*/X4 | -@P9[2];
T7[1]   : P8[1] * /X5 | -@P6[1];
T8[1,2] : P6[1] * P9[2] * /X6 | -@P1[1,2].

```

The immediate combinational Moore-type output signals Y0–Y6, depend directly only on appropriate place markings:

```

P1[1,2] | -Y0; P2[1] | -Y1; P4[2] | -Y2; P7[2] | -Y3*Y4;
P8[1] | -Y5; P9[2] | -Y6.

```

Instead of combinational, intermediate Moore-type outputs Y0–Y6, the values of next registered outputs @Y0–@Y6 could be predicted in advance and included directly in the initial rule-based specification. The transition-oriented specification of the controller, after the substitution of encoding terms and next values of changing outputs, would look as follows:

```

T1: /Q1*X0 | - @Q1*@Q2*@Q3*@Q4*/@Y0*@Y1*@Y2;
T2: Q1*/Q2*/Q3*X1 | - @Q1*@Q2*@Q3*@Y1;
( . . . )
T7: Q1*Q2*Q3*/X5 | -@Q1*@Q2*@Q3*@Y5;
T8: Q1*Q2*/Q3*Q4*/X6 | -@Q1*@Y6* @Y0.

```

In FPGA realizations of concurrent state machines with D flip-flops, it is worth introducing and directly implementing the intermediate binary signals  $\{T1, T2, \dots\}$  for detecting in advance the enabled transitions, which fire together with the next active edge of the clock. This way of design is especially suitable when relatively small FPGA macrocells might be easily reconfigured. In such Martin Bolton’s style, the subset of simultaneously *activated transitions* keeps the logic signal 1 on its appropriate *transition status lines*. Simultaneously, the complementary subset of *blocked transitions* is recognized by logic signal 0 on

its transition status lines. The great advantage of using transition status lines is the self-evident possibility of reducing the complexity of the next state and the output combinational logic. The registered output signals together with the next local state codes may be generated in very simple combinational structures, sharing together several common AND terms.

The simplified rule-based specification, especially planned for controllers with JK state and output registers, on the right side of sequents does not contain state coding signals, which keep their values stable, during the occurrence of transition<sup>1</sup>. Taking into account both the concept of transition status lines and introducing into specification only the changing registered Moore-type outputs signals, the specification may be rewritten as follows:

$$\begin{aligned} /Q1 * X0 \mid -T1; \\ \quad T1 \mid -@Q1 * @/Q2 * @/Q3 * @/Q4 * @Y0 * @Y1 * @Y2; \\ ( \dots ) \\ Q1 * Q2 * Q3 * /X5 \mid -T7; \\ \quad T7 \mid - @/Q3 * @/Y5; \\ Q1 * Q2 * /Q3 * Q4 * /X6 \mid -T8; \\ \quad T8 \mid - @/Q1 * @Y6 * @Y0. \end{aligned}$$

The translation of decision rules into VHDL is straightforward and can be performed as described in Refs. 2 and 6.

## 7. CONCLUSIONS

The paper presents the hierarchical Petri net approach to synthesis, in which the modular net is structurally mapped into field programmable logic. The hierarchy levels are preserved and related with some particular local state variable subsets. The proposed state encoding technique saves a number of macrocells and secures a direct mapping of Petri net into an FPL array. 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, such as ALDEC Active-HDL. VHDL syntax supports several conditional statements, which can be used to describe the topology and an interpretation of Petri nets.

## ACKNOWLEDGMENT

The research was supported by the Polish State Committee for Scientific Research (Komitet Badań Naukowych) grant 4T11C 006 24.

## REFERENCES

1. M. Adamski, Parallel controller implementation using standard PLD software. In: W.R. Moore, W. Luk (eds.), *FPGAs*. Abingdon EE&CS Books, Abingdon, England, pp. 296–304 (1991).
2. M. Adamski, SFC, Petri nets and application specific logic controllers. In: *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybern.*, San Diego, USA, pp. 728–733 (1998).
3. M. Adamski, M. Wegrzyn (eds.), *Discrete-Event System Design DESDes'01*, Technical University of Zielona Gora Press Zielona Góra, ISBN: 83-85911-62-6 (2001).
4. 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*, **141**, 405–412 (1994).
5. R. David, H. Alla, *Petri Nets & Grafcet. Tools for Modelling Discrete Event Systems*. Prentice Hall, New York (1992).
6. J.M. Fernandes, M. Adamski, A.J. Proença, VHDL generation from hierarchical Petri net specifications of parallel controllers. *IEE Proceedings-E, Computer and Digital Techniques*, **144**, 127–137 (1997).
7. M. Heiner, Petri Net based system analysis without state explosion. In: *Proceedings of High Performance Computing'98*, April 1998, SCS Int., San Diego, pp. 394–403 (1988).
8. T. Kozłowski, E.L. Dagless, J.M. Saul, M. Adamski, J. Szajna, Parallel controller synthesis using Petri nets. *IEE Proceedings-E, Computers and Digital Techniques*, **142**, 263–271 (1995).
9. N. Marranghello, W. de Oliveira, F. Damianini, Modeling a processor with a Petri net extension for digital systems. In: *Proceedings of Conference on Design Analysis and Simulation of Distributed Systems—DASD 2004*, Part of the ASTC, Washington, DC, USA (2004).
10. T. Murata, Petri Nets: Properties, analysis and applications. *Proceedings of the IEEE*, **77** (4), 541–580 (1989).
11. 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*, **32**, 111–118 (1991).
12. M.E. Szabo (Ed.), *The collected papers of Gerhard Gentzen*. North-Holland Publishing Company, Amsterdam (1969).
13. A. Yakovlev, L. Gomes, L. Lavagno (eds.), *Hardware Design and Petri Nets*. Kluwer Academic Publishers, Boston (2000).
14. A.D. Zakrevskij, *Parallel Algorithms for Logical Control*. Institute of Engineering Cybernetics of NAS of Belarus, Minsk (1999) (Book in Russian).



<http://www.springer.com/978-0-387-23630-8>

Design of Embedded Control Systems

Adamski, M.A.; Karatkevich, A.; Wegrzyn, M. (Eds.)

2005, XI, 267 p., Hardcover

ISBN: 978-0-387-23630-8