**Iwona GROBELNA**

University of Zielona Gora, Institute of Computer Eng. And Electronics

# Formal verification of embedded logic controller specification with computer deduction in temporal logic

***Abstract***. *The article presents a novel approach to formal verification of logic controller specification. Model checking technique is used to verify some behavioral properties. The approach proposes to use a rule-based logical model presented at RTL-level. Proposed logical model is suitable both for formal verification (model checking in the NuSMV tool) and for logical synthesis (using hardware description language VHDL). As the result, logic controller program (its implementation) will be valid according to its primary specification.*

***Streszczenie.*** *Artykuł przedstawia nowatorskie podejście do formalnej weryfikacji specyfikacji sterownika logicznego. Zaproponowany został regułowy model logiczny, który jest dogodny zarówno do formalnej weryfikacji (weryfikacja modelowa w narzędziu NuSMV), jak również do syntezy logicznej (z użyciem języku opisu sprzętu VHDL). Program sterownika logicznego (jego implementacja) będzie zatem poprawny względem początkowej specyfikacji. (**Formalna weryfikacja specyfikacji wbudowanych sterowników logicznych z wykorzystaniem wnioskowania komputerowego w logice temporalnej**).*

**Keywords**: formal verification, embedded logic controllers, temporal logic.
**Słowa kluczowe**: formalna weryfikacja, wbudowane sterowniki logiczne, logika temporalna.

## 1. Introduction

Embedded logic controllers specification is the first step in development process. Possible errors in this phase [1] may influence oncoming phases or even the whole venture. Usually, it generates enormous costs to remove errors which were detected too late. Dependable embedded logic controllers have additional requirements which requires beside high quality also reliability, availability, safety and secureness. Even a tiny error in design phase may change the total system behavior and may have tragic effects.

One of commonly used formal techniques by logic controllers specification are Petri Nets (PN) [2,3], and especially Control Interpreted Petri Nets (CIPN) [3,4]. They are well suited for modeling of hardware behavior including i.e. concurrency or resources sharing. There are many approaches to analyze CIPNs and check their boundness or liveness. There are however few approaches which allow to formally verify Petri Nets against some defined requirements, and none of them addresses directly CIPNs focused on RTL-level and their behavioral specification. Model checking technique with temporal logic offers the possibility to check behavior of designed logic controller.

Model checking [5] is one of formal verification methods among others like e.g. theorem proving [6,7] and is currently used in the industry in hardware [8] and software production. System model is compared with defined properties and an answer whether they are satisfied or not is given. In case of detected errors, appropriate counterexamples are generated which allow to localize error source.

The article is structured as follows. Section 2 describes related work in the area. Section 3 presents novel approach to formal verification of embedded logic controller specification with computer deduction in temporal logic. Embedded logic controller specification is formally written as an Control Interpreted Petri Net. The approach proposes to use a rule-based logical model presented at RTL-level (subsection 3.1) suitable both for formal verification (subsection 3.2) and for logical synthesis (subsection 3.3). Section 4 concludes the paper.

## 2. Related work

There have already been some approaches to verify Petri Nets. However, none of them addressed Control Interpreted Petri Nets focused on RTL-level and checked against behavioral properties. There have also been some approaches to verify UML diagrams, like in [9,10]. Programs for PLC controllers [11] have also been verified, prepared as the ST textual language [12], the Grafcet language (sequential part) [13] or the SFC graphical and hierarchical language [14].

In [15] Timed Petri Nets with clocks assigned to transitions or places are taken into account. Timed Petri Nets are also considered in [16] (nets with additional priorities). In [17] Petri Nets with time stamps for embedded systems are considered. Tokens hold there a value and a time stamp, what makes the nets different from classical Petri Nets. Furthermore, each transition is interpreted as a separate process, which changes marking of places.

In [18] Synchronous and Interpreted Petri Nets are considered. Authors verify them using PROMELA language and the SPIN model checker. In the first approach they do not allow for two or more guards of transitions (input signals of logic controller) to be simultaneously true, what makes difficult to analyze system behavior. In the second approach they also focus more on structural properties than on the behavioral. Priorities for transitions are introduced artificially, what causes deformation of system functionality. It is also hardly acceptable, that only one input signal can be active at one moment.

Petri Nets verification has also been proposed by G. Frey. In [19] a complete tool for logic controllers development with usage of Petri Nets is presented. Authors focused on Signal Interpreted Petri Nets (SIPN), which are similar to Control Interpreted Petri Nets (CIPN). The end result of logic controller development is here a program for PLC. Proposed approach does not take into account reconfigurable logic controllers based on FPGA circuits.

## 3. Novel approach to formal verification of embedded logic controller specification

The novel approach to model checking of Control Interpreted Petri Nets focuses on rule-based logical model generation basing on a CIPN. Logical model with temporal logic formulas is presented at RTL-level in such a way that it is easy to synthesize as reconfigurable logic controller or PLC as well as to formally verify for behavioral properties (see Fig. 1). Logical model is therefore transformed into model description in the NuSMV model checker [20] input language with behavioral assertions. Model checker tool verifies the model and checks whether some defined behavior properties are fulfilled. On the other hand, logical model is transformed into hardware description language (i.e. VHDL) with structural assertions. Logical model is

hence used for synthesis purposes as well as for model checking. It is a format in which the behavior of logic controller is specified. It contains logical formulas describing changes in the designed system.

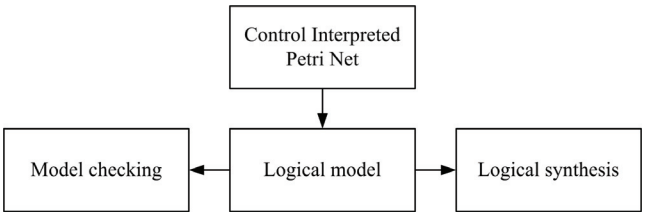


Fig.1. Logical model

### 3.1 Logical model

Logical model includes variables definition with their initial values. The following elements of Control Interpreted Petri Net are interpreted as model variables:

a) places (*P*)
   Each place of Control Interpreted Petri Net is assigned a variable of *Boolean* type. It takes the *TRUE* value if the place is marked. It is possible that multiple places have the *TRUE* value assigned, as there may be several concurrent processes defined.
b) input signals (*X*)
   Each input signal of logic controller is assigned a variable of *Boolean* type. It takes the *TRUE* value if the signal is active and the *FALSE* value if it is inactive. It is possible that multiple input signals have the *TRUE* value assigned, as there may be several input signals active at the same time.
c) output signals (*Y*)
   Each output signal of logic controller is assigned a variable of *Boolean* type. It takes the *TRUE* value if the signal is active and the *FALSE* value if it is inactive. It is possible that multiple output signals have the *TRUE* value assigned, as there may be several output signals active at the same time.

Beside variables, logical model specifies also some rules (transitions). The rules define how model variables change over time. Each rule (transition) is described separately and contains firing condition (with active places and input signals) and marking changing of Petri Nets places (transition input places become inactive, while transition output places become active):

```
transition_name:
firing_conditions -> X marking_changing;
```

Output signals are assigned to the corresponding places:

*for each p ϵ P assign (if defined) output signals y ϵ Y*

The assignment includes active place and active output signals:

```
active_place -> active_output_signals;
```

*active_output_signals = active_sig_1 & active_sig_2 & ... ;*

Logical model also defines possible changing of input signals. However, the definition is only used by model checking process (model description preparation). In the HDL (*Hardware Description Language*) file input signals are not concerned as they are inputs to the logic controller. Input signals are supposed to change randomly, but only in expected situations. Allowing completely random value changing would improve the real world simulation. However, in the model checking process in NuSMV we would then face the so called state space explosion problem, as generated state space could reach enormous size even by a simple logic controller with couple input signals. Input signals in logical model are assigned to places, by which active marking they can change value:

```
active_place -> input_signals;
```

*input_signals = signal_1 & signal_2 & ... ;*

where *signal_1* takes any of values:
*TRUE* (`signal_1`) or *FALSE* (`!signal_1`).

The following code presents some snapshots from a sample logical model based on a Control Interpreted Petri Net (part of it in Fig. 2):

```
VARIABLES
  places: p1, p2, ...
  inputs: x0, x1, ...
  outputs: y0, y1, ...
INITIALLY
  p1; !p2; ...
  !x0; !x1; ...
  y0; !y1; ...
TRANSITIONS
  t1: p1 & x0 -> X (!p1 & p2 & p4);
  t2: p2 & x1 -> X (!p2 & p3);
  ...
OUTPUTS
  p1 -> y0;
  p2 -> y1;
  ...
INPUTS
  p1 -> !x0 | x0;
  p2 -> !x1 | x1;
  ...
```

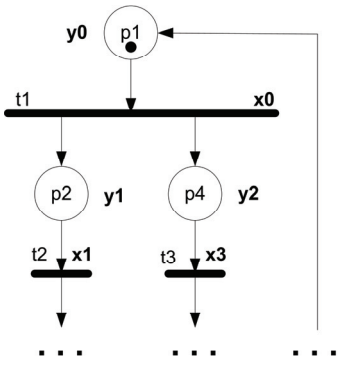


Fig.2. Part of a sample Control Interpreted Petri net

### 3.2 Formal verification

Logical model derived from Control Interpreted Petri Net is transformed into format of the NuSMV model checker according to the following rules:

a) Each place *p ϵ P* is a variable of *Boolean* type
   For each variable *p ϵ places* a separate variable `p:boolean` is defined
b) Each input signal *x ϵ X* is a variable of *Boolean* type
   For each variable *x ϵ inputs* a separate variable `x:boolean` is defined
c) Each output signal *y ϵ Y* is a variable of *Boolean* type
   For each variable *y ϵ outputs* a separate variable `y:boolean` is defined
d) Defined variables take predefined initial values:
   `init(var) := TRUE` or `init(var) := FALSE`
e) Each place changes its marking according to the defined rules; conditions of changes between places (token flow) occur in pairs – in the previous place and in the next place
f) Each output signal changes its value according to the predefined rules

g) Each input signal changes randomly, but can take the expected values connected with Petri net places or change adequately to the situation

Changing of variables values are considered as separate segments in the NuSMV:

```
next(variable) := case
   deactivation_condition : FALSE;
   activation_condition   : TRUE;
   TRUE                   : variable;
esac;
```

Properties to be checked are defined using either LTL or CTL logic [21-24]. Properties [25] describe safety requirements (*something bad will never happen*), as well as liveness requirements (*something good will eventually happen*). Safety and liveness requirements are the most frequently specified requirements to be verified. The requirements list should include as much desired properties as possible, as only they will be checked.

Structural properties can be checked on the Petri net level and does not require using model checking technique. Properties like liveness or deadlock-freedom can be verified by some tools dedicated to Petri nets, like i.e. WoPeD [26] or PIPE2 [27].

Properties are defined according to the schema: `LTLSPEC LTL_expresion` or `CTLSPEC CTL_expresion`

A sample liveness property is `LTLSPEC G (y4 & x4 -> F (y5 & y6))` what means, that always when the *y4* output signal is active and the *x4* input signal becomes active, finally both output signals *y5* and *y6* will be active.

A sample safety property is `LTLSPEC G !(y5 & y8)` what means, that it should never be the case, that both output signals *y5* and *y8* are active at the same time.

The following code presents some snapshots from model description in the NuSMV tool:

```
VAR
   p1 : boolean;
   p2 : boolean;
   ...
   x0 : boolean;
   x1 : boolean;
   ...
   y0 : boolean;
   y1 : boolean;
   ...
ASSIGN
   init(p1) := TRUE;
   init(p2) := FALSE;
   ...
   init(x0) := FALSE;
   init(x1) := FALSE;
   ...
   init(y0) := TRUE;
   init(y1) := FALSE;
   ...
   next(p1) := case
      p1 & x0        : FALSE;
      p6 & p9 & !x6  : TRUE;
   TRUE              : p1;
   esac;
   ...
   next(y1) := case
      p2   : TRUE;
      TRUE : FALSE;
   esac;
   ...
   next(x0) := case
      p1   : {FALSE, TRUE};
      TRUE : FALSE;
   esac;
   ...
```

### 3.3 Synthesis

Logical model can be easy synthesized as reconfigurable logic controller or PLC. Logical model derived from Control Interpreted Petri Net is transformed into VHDL language according to the following rules:

a) Each place $p \in P$ is an internal signal of *std_logic* type
   For each variable $p \in places$ a separate internal signal is defined: `signal p : std_logic`
b) Each input signal $x \in X$ is an input port of *std_logic* type
   For each variable $x \in inputs$ a separate input port is defined: `x : in STD_LOGIC`
c) Each output signal $y \in Y$ is an output port of *std_logic* type
   For each variable $y \in outputs$ a separate output port is defined: `y : out STD_LOGIC`
d) Each defined internal signal (Petri net place) takes an initial value, set by clock rising edge and active reset signal
e) Each place changes its marking according to defined rules:
   - fired transition changes marking of its input and output places
   - not fired transition makes its inputs places to maintain tokens
f) Each output signal changes its value according to active places; output signals are active by active marking of corresponding places
g) Input signals are not considered, as they are inputs to the logic controller.

The following code presents some snapshots from synthesizable code in VHDL:

```
Entity Logic_contr is
  port (Clk, Reset : in STD_LOGIC;
  x0, x1, ... : in STD_LOGIC;
  y0, y1, ... : out STD_LOGIC );
end Logic_contr;
architecture arch of Logic_contr is
  signal p1, p2, ... : std_logic;
  ...
  if p1='1' and x0='1' then
    p1 <= '0';
    p2 <= '1';
    p4 <= '1';
  end if;
  if p1='1' and x0='0' then
    p1 <= '1';
  end if;
  ...
  y0 <= p1;
  y1 <= p2;
  ...
```

Presented model in VHDL is oriented on places and transitions [28]. Prepared model can be simulated (using i.e. *Active HDL* environment) and synthesized (using i.e. *XILINX Plan Ahead* environment).

## 4. Conclusions

Proposed novel approach to verification of embedded logic controller specification allows to detect some subtle errors on an early stage of system development. Rule-based representation of Control Interpreted Petri Nets in temporal logic is presented on RTL-level and is easy to formally verify using model checking technique and to synthesize using hardware description languages into reconfigurable logic controller or PLC. It is also possible to

use another forms of system specification as basis for formal verification and synthesis, i.e. UML 2.x Activity Diagrams which can be transformed into CIPNs [29].

Presented approach was tested on several examples of industrial logic controllers specifications by means of Control Interpreted Petri Nets. As a support for testing, a tool has been developed, which allows automatic transformation from logical model into model description in the NuSMV format. Transformation into VHDL code is still under development.

Results of the work include the assurance that verified behavioral specification in temporal logic will be an abstract program of matrix reconfigurable logic controller. So, logic controller program (its implementation) will be valid according to its primary specification.

## ACKNOWLEDGMENTS

*The author is a scholar within Sub-measure 8.2.2 Regional Innovation Strategies, Measure 8.2 Transfer of knowledge, Priority VIII Regional human resources for the economy Human Capital Operational Programme co-financed by European Social Fund and state budget*

## BIBLIOGRAPHY

[1] Grobelna I., Grobelny M., Gaps in design and tests of dependable embedded systems, *Metody Informatyki Stosowanej* (2010) nr 2, 45 - 51.
[2] Adamski M., Karatkevich A., Wegrzyn M. (ed.), Design of embedded control systems, Springer 2005 (USA).
[3] David R., Alla H., Discrete, Continuous, and Hybrid Petri Nets, Springer-Verlag Berlin Heidelberg (2010).
[4] Adamski M., A rigorous design methodology for reprogrammable logic controllers, *The International Workshop on Discrete-Event System Design, DESDes'01* (2001) Poland.
[5] Emerson E. A., The beginning of model checking: a personal perspective, *Lecture Notes in Computer Science, 25 Years of Model Checking: History, Achievements, Perspectives* (2008), 27 - 45.
[6] Clarke E.M., Wing J.M. et al, Formal methods: state of the art and future directions*, ACM Computing Surveys* Vol. 28, (1996), No. 4.
[7] Kern C., Greenstreet M.R., Formal verification in hardware design: a survey, *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, Vol. 4, (1999), Issue 2, 123 - 193.
[8] Fix L., Fifteen years of formal property verification in Intel, *Lecture Notes in Computer Science (LNCS)*, Vol. 5000/2008, O. Grumberg, H. Veith (Eds.), (2008), 139 - 144.
[9] Niewiadomski A., Penczek W., Szreter M., Towards checking parametric reachability for UML state machines, *Proc of. 7th International Andrei Ershov Memorial Conference Perspectives of System Informatics*, (2009), 229 - 240.
[10] Lam V.S.W., Padget J., Symbolic Model Checking of UML Statechart Diagrams with an Integrated Approach, *Proceedings of the 11th IEEE International Conference and Wrokshop on the Engineering of Computer-Based Systems (ECBS'04)*, (2004).
[11] Broel-Plater B., Uklady wykorzystujace sterowniki PLC. Projektowanie algorytmów sterowania, Wydawnictwo Naukowe PWN SA (2008).
[12] Gourcuff V., De Smet O., Faure J.-M., Efficient representation for formal verification of PLC programs, *Discrete Event Systems, 8th International Workshop*, (2006), 182 - 187.
[13] Lampérière-Couffin S., Lesage J.J., Formal Verification of the Sequential Part of PLC Programs, *5th Workshop on Discrete Event Systems (WODES 2000)*, Ghent, (2000), 247 - 254.
[14] Bauer N., Engell S., Huuck R., Lohmann S., Lukoschus B., Remelhe M., Stursberg O., Verification of PLC Programs given as Sequential Function Charts, *Proceedings of SoftSpez Final Report,* (2004), 517 - 540.
[15] Penczek W., Półrola A., Advances in verification of Time Petri Nets and timed automata, Springer (2006).
[16] Berthomieu B., Peres F., Vernadat F., Model checking bounded prioritized time Petri nets, *ATVA'07 Proceedings of the 5th international conference on Automated technology for verification and analysis*, (2007).
[17] Cortés L.A., Eles P., Peng Z., Formal coverification of embedded systems using model checking, *Proceedings of the 26th EUROMICRO Conference (EUROMICRO'00)*, IEEE (2000).
[18] Ribeiro O.R., Fernades J.M., Translating synchronous Petri Nets into PROMELA for verifying behavioural properties, IEEE (2007).
[19] Frey G., Wagner F., A Toolbox for the Development of Logic Controllers using Petri Nets, *Proceedings of the 8th International Workshop on Discrete Event Systems (WODES 2006)*, 473 - 474.
[20] Cavada R. et al., NuSMV 2.5 user manual, available at *http://nusmv.fbk.eu/*
[21] Ben-Ari M., Logika matematyczna w informatyce. Klasyka informatyki. Wydawnictwa Naukowo-Techniczne, Warszawa (2005).
[22] Huth M., Ryan M., Logic in Computer Science. Modelling and Reasoning about Systems, Cambridge University Press (2004).
[23] Lamport L., Sometime is sometimes not never, On the Temporal Logic of Programs, *Proceedings of the Seventh ACM Symposium on Principles of Programming Languages, ACM SIGACT-SIGPLAN* (1980), 174 - 185.
[24] Rice M.V., Vardi M.Y., Branching vs. Linear Time: Final Showdown, *Proceedings of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001,* LNCS Volume 2031, Springer-Verlag (2001), 1 - 22.
[25] Kropf T., Introduction to Formal Hardware Verification, Springer-Verlag Berlin Heidelberg New York, (1999).
[26] WoPeD homepage: *www.woped.org*.
[27] PIPE homepage: *http://pipe2.sourceforge.net/*.
[28] Węgrzyn M., Modelowanie sieci Petriego w jezyku VHDL, *Przeglad Elektrotechniczny*, (2010), nr 1, 212 - 216.
[29] Grobelna I., Grobelny M., Adamski M., Petri Nets and activity diagrams in logic controller specification - transformation and verification, *Mixed Design of Integrated Circuits and Systems - MIXDES 2010*, (2010) 607 - 612.

***Authors**: mgr. inż. Iwona Grobelna, University of Zielona Gora, ul. Podgorna 50, 65-246 Zielona Gora, Poland E-mail: i.grobelna@iie.uz.zgora.pl*