PAK vol. 58, nr 6/2012

Iwona GROBELNA, Michał GROBELNY
UNIWERSYTET ZIELONOGÓRSKI,
Podgórna 50, 65-246 Zielona Góra

Inhibitor and enabling arcs in logic controller design

Mgr inż. Iwona GROBELNA

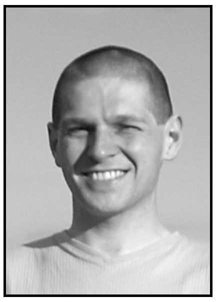
In 2007 she graduated Computer Science at the Faculty of Electrical Engineering, Computer Science and Telecommunications at University of Zielona Góra and Fachhochschule Giessen-Friedberg (Germany). Since March 2008 employed as an assistant at the Institute of Computer Science and Electronics, University of Zielona Góra. Her research interests include verification methods of embedded systems specification. Member of the Polish Information Processing Society.

e-mail: I.Grobelna@iie.uz.zgora.pl

Mgr inż. Michał GROBELNY

In 2007 he graduated Computer Science at the Faculty of Electrical Engineering, Computer Science and Telecommunications of University of Zielona Góra and Fachhochschule Giessen-Friedberg (Germany). Since October 2011 employed as an assistant in the Department of Media and Information Technology at University of Zielona Góra. His research interests include specification methods of embedded control systems. Member of the Polish Information Processing Society.

e-mail: M.Grobelny@weit.uz.zgora.pl



Abstract

The paper presents a novel approach to rule-based logic controller specification and its verification. The proposed abstract model is suited for formal verification (using model checking technique) as well as for logic synthesis (using hardware description language VHDL). Special focus is put on Interpreted Petri Nets with inhibitor and enabling arcs, their realization in rule-based model and, additionally, their interpretation in another logic controller specification technique – UML Activity Diagrams (version 2.x).

Keywords: logic controller specification, formal verification, Petri Nets inhibitor and enabling arcs, UML Activity Diagrams.

Łuki zakazujące i zezwalające w projektowaniu sterowników logicznych

Streszczenie

Artykuł przedstawia nowatorskie podejście do regułowej specyfikacji sterownika logicznego, wraz z jej weryfikacją (walidacją). Proponowany abstrakcyjny model logiczny jest dogodny zarówno do formalnej weryfikacji modelowej, jak również do syntezy logicznej (język opisu sprzętu VHDL). Szczególną uwagę poświęcono łukom zakazującym i zezwalającym interpretowanych sieci Petriego. Po krótkim wprowadzeniu do omawianej tematyki (rozdział 2), przedstawiono przykład interpretowanej sieci Petriego z łukami zakazującymi i zezwalającymi (rys. 1). Podano sposób ich realizacji w abstrakcyjnym modelu logicznym (rozdział 3, schemat kompletnego proponowanego systemu na rys. 2 oraz przykład regułowego modelu sterownika logicznego na rys. 3). Zaproponowano interpretację łuków zakazujących i zezwalających sieci Petriego w innej postaci specyfikacji zachowania sterownika logicznego (rozdział 4) – diagramach aktywności języka UML (w wersji 2.x). Ze względu na bezstanowość diagramów aktywności, nie jest możliwe bezpośrednie odwzorowanie rozpatrywanych łuków. W artykule zaproponowano dwa rozwiązania – opierające się na wprowadzeniu dodatkowego sygnału (rys. 4a) oraz alternatywne – bazujące na etykietowaniu przepływów (rys. 4b). Przedstawiono sposób formalnej weryfikacji tak przygotowanej specyfikacji regułowej oraz jej syntezy logicznej (rozdział 5). Publikacja kończy się podsumowaniem oraz wnioskami (rozdział 6).

Słowa kluczowe: specyfikacja sterownika logicznego, formalna weryfikacja, łuki zakazujące i zezwalające sieci Petriego, diagramy aktywności języka UML.

1. Introduction

The process of logic controllers design [1] usually starts with specification. In this phase system properties and main functionality goals are specified. It is important that the specification is verified (validated) [2] before implementation, as it allows to early discovering possible errors. Subtle errors in the specification may influence oncoming phases or even the whole venture, especially by dependable embedded logic controllers where requirements address beside high quality also reliability, availability, safety and secureness.

Behavior (functionality of designed system) may be formalized using various techniques [3], one of them are Petri Nets [4, 5] or UML Activity Diagrams [6].

Model checking technique [7] can be used to verify model description and check whether it satisfies some defined requirements.

The novel approach proposes to use a rule-based logical model presented at RTL-level, suitable both for formal verification (model checking) and for logical synthesis.

The paper is structured as follows. Section 2 presents some basic information about specification of logic controllers, focusing especially on Petri Nets and UML Activity Diagrams. Section 3 illustrates the proposed design system for (reconfigurable) logic controllers and introduces a rule-based logical model. Section 4 proposes interpretation of inhibitor and enabling arcs in UML Activity Diagrams. Section 5 presents novel approach to formal verification of logic controller programs with computer deduction in temporal logic and describes synthesis method, in form of rapid prototyping. The paper ends with a short summary.

2. Specification of logic controller behavior

In this section some background on logic controller specification techniques is provided, focusing especially on Petri Nets, Control Interpreted Petri Nets (Signal Interpreted Petri Nets) and UML Activity Diagrams (in version 2.x).

Petri nets [4, 5] were firstly introduced as a general purpose mathematical model for describing relations between conditions and events. They are currently used in many industrial branches for planning and controlling of production flows, design and programming of microprocessor controllers, system software synthesis, etc. Graphic representation can be understood even by non-technical staff. It allows e.g. specifying such behaviors as parallelism and concurrency, choice, synchronization, memorizing, reading or resources sharing.

Formally, a Petri net can be defined as a 3-tuple $PN = (P, T, F)$, where:

1) P is for places

2) T is for transitions

3) F is for flows between elements of P and T or elements of T and P.

A transition can be fired only if each of its input places contains at least one token. Then from each of its input places one token is removed and added to each of its output places.

Control Interpreted Petri Nets [4] (as well as Signal Interpreted Petri Nets [8]) specify and model the behavior of concurrent logic controllers and take into account properties of controlled objects. Local states may change after the firing of transitions, if some events occur. Transition guards are associated with input signals of the controller. Places are associated with output signals of the controller. Global state of the logic controller is built of simultaneously holding local states.

Formally, a Control Interpreted Petri Net can be defined as a 6-tuple *CIPN = (PN, X, Y, ρ, λ, γ)*, where:

1) *PN* is an alive and safe Petri net

2) *X* is a set of input states

3) *Y* is a set of output states

4) $\rho: T \rightarrow 2^X$ is a function, that each transition assigns the subset of input states *X(T)*; 2^X states for the set of all possible subsets of *X*

5) $\lambda: M \rightarrow Y$ is a function of Moore outputs, that each marking *M* assigns the subset of output states *Y(M)*

6) $\gamma: (M \times X) \rightarrow Y$ is a function of Mealy outputs, that each marking *M* and input states *X* assigns the subset of output states *Y*.

Additionally, in Control Interpreted Petri Nets each place can include only one token (it is a safe net), input signals are assigned to transitions as guards, while output signals are assigned to places.

Beside basic elements of Petri nets (places, transitions, arcs), in Interpreted Petri Nets also inhibitor and enabling arcs are possible which enable or disable an indicated transition. Enabling arc does not change source place marking in case of firing an indicated transition. Enabling and inhibitor arcs may be used to prioritize transitions firings in concurrent processes.

So, in a sample net from Fig. 1 (example taken from [8]), the *t0* transition has assigned the *x0* input signal, what means that the *x0* input signal must be active and the *p0* place must contain token in order to fire the transition. Then, token is removed from place *p0* and added to place *p2*.

Additionally, in Signal Interpreted Petri Nets there are possible enabling arcs (from place *p3* to transition *t2*) and inhibitor arcs (from place *p0* to transition *t1*). By an enabling arc, place *p3* holds marking when transition *t2* is fired. By an inhibitor arc, so long as place *p0* holds marking, the *t1* transition cannot be fired, even if place *p1* maintains token.

The sample net has four places, four input signals and four transitions. Initial marking involves places *p0* and *p1*.

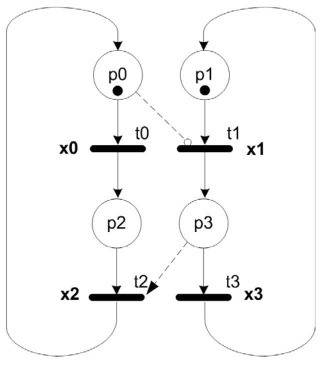


Fig. 1. A sample Signal Interpreted Petri Net [8]
Rys. 1. Przykładowa sieć Petriego typu SIPN [8]

3. System for logic controller design and rule-based logical model

Logic controller development process usually starts with a specification, further goes through verification and simulation, finally ending with implementation. Schema of the proposed system for design of logic controllers is shown in Fig. 2.

A logical model with temporal logic formulas is presented at RTL-level in such a way that it is easy to synthesize as a reconfigurable logic controller as well as to formally verify for behavioral properties. The logical model is therefore transformed into model description in the NuSMV model checker input language with behavioral assertions. A model checker tool verifies the model and checks whether some defined behavioral properties are fulfilled. On the other hand, the logical model is transformed into hardware description language (i.e. VHDL) with structural assertions. The 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.

It is also possible to use other specification formalisms, beside Interpreted Petri Nets, such as Sequential Function Charts (SFC) or UML diagrams, especially UML 2.x Activity Diagrams [9].

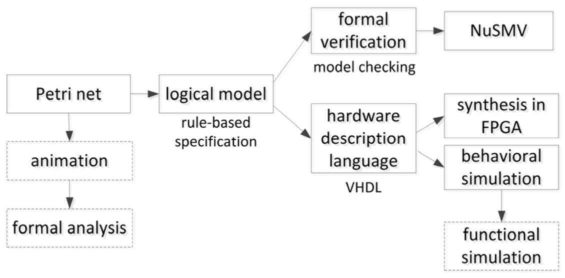


Fig. 2. Schema of the proposed system
Rys. 2. Schemat proponowanego systemu

The rule-based logical model for a (Control) Interpreted Petri Net can be defined as a seven-tuple *LM = {P, X, Y, S, T, O, I}*, where:

1) *P* stands for places (internal local states),

2) *X* stands for inputs (input signals to logic controller),

3) *Y* stands for outputs (output signals to logic controller),

4) *S* stands for initial values of places, inputs and outputs,

5) *T* stands for rules describing transitions (indicating changes of local states),

6) *O* stands for active outputs corresponding to appropriate places,

7) *I* stands for inputs supposed to be active in appropriate places (for formal verification simplification).

The logical model includes definition of variables with their initial values. It specifies also some rules (transitions). The rules define how the model variables change over time. Each rule (transition) is described separately and contains firing condition (with active places and input signals) and marking changes of Petri net places (transition input places become inactive, while transition output places become active). Output signals (if defined) are assigned to the corresponding places. The logical model also defines possible changing of input signal values. 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.

The following code in Fig. 3 presents a sample logical model based on a net from Fig. 1. There are four places and four input signals defined, but no output signal.

```
VARIABLES
places: p0, p1, p2, p3
inputs: x0, x1, x2, x3
outputs: no
INITIALLY
p0; p1; !p2; !p3;
!x0; !x1; !x2; !x3;
no;
TRANSITIONS
t0: p0 & x0 -> X (!p0 & p2);
t1: p1 & !p0 & x1 -> X (!p1 & p3);
t2: p2 & p3 & x2 -> X (!p2 & p0);
t3: p3 & !p2 & x3 -> X (!p3 & p1);
OUTPUTS
p0 -> no;
p1 -> no;
p2 -> no;
p3 -> no;
INPUTS
p0 -> !x0 | x0;
p1 -> !x1 | x1;
p2 -> !x2 | x2;
p3 -> !x3 | x3;
```

Fig. 3. Rule-based logical model
Rys. 3. Regułowy model logiczny

The inhibitor and enabling arcs are interpreted in appropriate transitions (rules), in this case the *t1* and *t2* transitions. So, the *t1* transition has an inhibitor arc from place *p0*. The preconditions for rule realization are active marking of the *p1* place, inactive marking of the *p0* place and the *x1* input signal active. The postconditions are inactive marking of the *p1* place and active marking of the *p3* place (token flow from place *p1* to *p3*).

Analogously, the above happens for the *t2* transition with an enabling arc from place *p3*, with the difference that place *p3* must be active in order to fire the transition, but its marking does not change.

4. Interpretation in UML Activity Diagrams

Alternatively to the Petri net, UML Activity Diagrams as specification technique of logic controller behavior are considered. Description nature of UML Activity Diagram syntax ensures better readability of specification. Thus, usage of UML diagrams complementarily to Petri nets may positively influence communication with non-technicians (e.g. clients). Both specification techniques have a lot in common and may be used parallel in one project, with the help of transformation [9]. Moreover, not all UML Activity Diagram elements fit to design control processes. Furthermore, some of them are even useless or do not have equivalent elements in control process specification rules. Therefore, a necessary and in the same time sufficient subset of elements dedicated to describe control process is desirable [9].

Hence, inhibitor and enabling arcs available in Petri net are not easily achievable in UML version 2.x Activity Diagrams with visual elements of discussed technique. There is no possibility to make a simple graphical connection between state of the system and a transition. It would be possible to visually connect flows (somehow corresponding to states) with actions, but it would be beyond the syntax of UML. Therefore, it is reasonable to employ additional signals to realize enabling and disabling of action execution. Moreover, enabling and inhibitor arcs in Petri nets are partly a graphical representation of such additional signals without employing them. Sample diagrams with realization of enabling and inhibition of action execution with additional signal (corresponding to enabling and inhibitor arcs from Petri net from Fig. 1) is shown in Fig. 4a. An additional signal *disable* is used to achieve inhibitor arc equivalence. As it is shown in Fig. 4a the *disable* signal is set after start of control process and then every time (in every cycle of process) just after the execution of action *t2*. Enabling arc is realized with usage of additional *enable* signal, which is set just after the execution of *t1* action and is reset after the execution of *t3* action. Moreover, both presented constructs (enabling and disabling of action execution) are fully corresponding to inhibitor and enabling arcs in Petri net presented in Fig. 1.

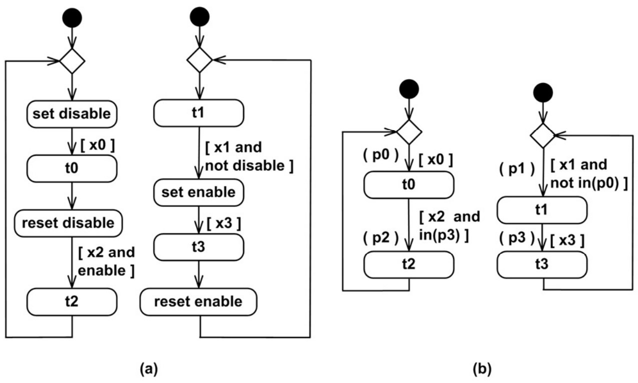


Fig. 4. Sample UML Activity Diagram with additional signals (a) and with flow labeling (b)
Rys. 4. Przykładowy diagram aktywności UML z dodatkowym sygnałem (a) oraz z etykietowaniem przepływów (b)

Representation of inhibitor arc without additional signal is proposed in [10], but it considers specification with usage of UML in version 1.4, where such a construct was much easier achievable. Disabling of action execution is there achieved with use of action state nature of actions and activities in this version of UML specification. However, in UML 2.x there is also a possibility to omit additional signal employment. Ought to achieve similar result there have to be done a flow labeling. Flows of the prepared diagram with specification have to be marked and identified by own labels attached to them. Flow labels are corresponding to states (places) of Petri nets. Then, there is a possibility to refer to a flow label into action execution condition achieving equivalent construct to inhibitor and enabling arcs of Petri nets. Realization of the construct is implemented with the use of keyword *in* (e.g. *in(p3)* enabling condition of the *t2* action in Fig. 4b) in case of enabling nature of condition and with the keyword *not in* (e.g. *not in(p0)* inhibitor condition of the *t0* action in Fig. 4b) in case of inhibitor/disabling nature of condition. Unnecessarily presented flow labeling is not compatible with standard UML Activity Diagram syntax and may not be compatible with all specification tools and techniques.

5. Formal verification and synthesis

In this section a formal verification method of the rule-based abstract logical model using model checking technique [11] together with its synthesis method (in form of rapid prototyping) [12] is considered. Some properties of Interpreted Petri Nets with inhibitor and enabling arcs may also be checked using other methods, as i.e. stubborn sets [13].

Model checking is one of formal verification methods, like e.g. *theorem proving* or *equivalence checking*, and is currently used in the industry in software and hardware production. The 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 localizing the error source. Model checking process can be performed on the whole system, or just on a part of it (so called *partial verification*), which has an important meaning especially for complex systems.

The rule-based logical model (example in Fig. 3) is transformed into format of the NuSMV model checker according to some strictly defined rules [11].

Model description is the first part needed for model checking. Additionally, it is necessary to specify some requirements, which are supposed (expected) to be true in the defined model. Structural properties can also be checked on the Petri net level (and does not require model checking technique). However, the most important are here behavioral properties, which describe system functionality, impact of input signals and output signals activity. Properties to be checked are defined using temporal logic – either LTL or CTL [14]. Properties [2] 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.

Sample properties expressed in LTL temporal logic [14] correspond to the interpretation of inhibitor arc (Fig. 5 for a net from Fig. 1). It is formally verified what happens if places *p0* and *p1* include tokens and conditions *x0* and *x1* are fulfilled. Which place will be active in the next state (temporal operator X) – the *p2* or *p3* place? Model checking process gives the answer. The first defined property is satisfied in described model, what confirms the interpretation of inhibitor arc. The second requirement cannot be satisfied (as the *t1* transition cannot be fired) and as the result appropriate counterexample is generated.

```
LTLSPEC G (p0 & p1 & x0 & x1 -> X p2);
LTLSPEC G (p0 & p1 & x0 & x1 -> X p3);
```

Fig. 5. Properties expressed in LTL temporal logic
Rys. 5. Właściwości określone z wykorzystaniem liniowej logiki temporalnej

The proposed rule-based logical model is also suitable for logic synthesis. Interpreted Petri Net, which is the core for logical model, is a safe net. Places can be then implemented using simple flip-flops, as their marking is expressed by a binary value (*1 / 0* – active / inactive marking) [15]. Flip-flops amount (for places) using one-hot encoding is equal to the amount of places (and so for the amount of local states).

The rule-based logical model can be easy synthesized as reconfigurable logic controller. The logical model is transformed into VHDL language according to some strictly defined rules [12]. Model in VHDL is oriented on places and transitions. It can be simulated (using i.e. *Active HDL* environment) and synthesized (using i.e. *XILINX Plan Ahead* environment).

Synthesis is performed in the form of rapid prototyping and its main goal is to check, whether the designed system works at all, but the circuit may be not optimized. Circuit optimization and minimization of resources usage are here out of scope, however they may be important in some fields [15].

6. Conclusions

The proposed novel approach to verification of embedded logic controller specification allows detecting some subtle errors on an early stage of system development. Rule-based representation 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. Hence, a logic controller program (its implementation) will be valid according to its primary specification. This may shorten the duration time of logic controller development process (as early discovered errors are faster corrected) and, consequently, save money (as project budgets will not be exceeded).

The approach differs from the one presented in [8], where logic controller specification is verified in SPIN model checker. Although the referred approach allows verifying Signal Interpreted Petri Nets, it does not guarantee that the resulting implementation will be valid according to primary specification. In the novel approach presented in the paper, an additional established rule-based logical model makes the specification abstract and more general, enabling as well formal verification as logic synthesis.

Direct representation of Petri net inhibitor and enabling arcs is not possible in UML Activity Diagrams (in version 2.x). Alternative solutions have to be found. They should allow interpreting arcs and making the transformation between two considered logic controller behavior specification techniques possible. The paper presents two different approaches – one with additional signals, and the second with flow labeling.

HUMAN CAPITAL
HUMAN – BEST INVESTMENT!
Lubuskie
Worth your while
EUROPEAN UNION
EUROPEAN SOCIAL FUND

Authors are scholars 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.

7. References

[1] Adamski M., Karatkevich A., Wegrzyn M. (ed.): Design of embedded control systems. Springer 2005 (USA).
[2] Kropf T.: Introduction to Formal Hardware Verification. Berlin Heidelberg: Springer-Verlag, 1999.
[3] Gomes L., Barros J.P., Costa A.: Modeling formalisms for embedded system design. Embedded Systems Handbook, Taylor & Francis Group, 2006.
[4] David R., Alla H.: Discrete, Continuous, and Hybrid Petri Nets. Berlin Heidelberg: Springer-Verlag, 2010.
[5] Girault C., Valk R.: Petri Nets for Systems Engineering, A Guide to Modelling, Verification and Applications. Berlin Heidelberg: Springer-Verlag, 2003.
[6] Unified Modelling Language homepage: www.uml.org
[7] Clarke E. M., Grumberg O., Peled D. A.: Model checking. Cambridge, The MIT Press, 1999.
[8] Ribeiro Ó. R., Fernades J.M.: Translating synchronous Petri Nets into PROMELA for verifying behavioural properties. International Symposium on Industrial Embedded Systems, 2007, pp. 266-273.
[9] 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 : proceedings of the 17th international conference, 2010, pp. 607-612.
[10] Eshuis R.: Semantics and Verification of UML Activity Diagrams for Workflow Modelling (PhD thesis), University of Twente, Wierden, 2002.
[11] Grobelna I., Adamski M.: Model checking of control interpreted Petri nets. Mixed Design of Integrated Circuits and Systems - MIXDES 2011 : proceedings of the 18th international conference, 2011, pp. 621-626.
[12] Grobelna I.: Formal verification of embedded logic controller specification with computer deduction in temporal logic. Electrical Review, 2011, nr 12a, pp. 47-50.
[13] Karatkevich A., Andrzejewski G.: Analiza wybranych własności interpretowanej sieci Petriego metodą optymalnej symulacji, Pierwsza Krajowa Konferencja Elektroniki - KKE 2002, Polska, 2002, T. 2, pp. 685-690 (in Polish).
[14] Ben-Ari M.: Mathematical Logic for Computer Science. Berlin Heidelberg: Springer-Verlag, 2001.
[15] Pardey J., Bolton M.: Logic synthesis of synchronous parallel controllers. Proc. International Conference on Computer Design: VLSI in Computer & Processors, Cambridge, 1991, pp. 454-457.

otrzymano / received: 15.03.2012
przyjęto do druku / accepted: 01.05.2012 *artykuł recenzowany / revised paper*