Chapter 8

Control Interpreted Petri Nets – Model Checking and Synthesis

Iwona Grobelna

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/47797

1. Introduction

The chapter presents a novel approach to formal verification of logic controller programs [2], focusing especially on reconfigurable logic controllers (RLCs). Control Interpreted Petri Nets [8] are used as formal specification of logic controller behavior. The approach proposes to use an abstract rule-based logical model presented at RTL-level. A Control Interpreted Petri Net is written as a logical model, and then processed further. Proposed logical model (Figure 1) is suitable both for formal verification [14] (model checking in the NuSMV tool [19]) and for logical synthesis (using hardware description language VHDL).

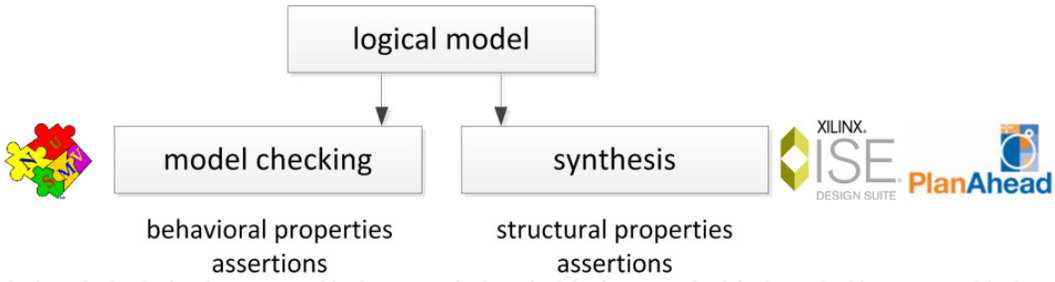

Figure 1. Logical model for model checking and synthesis purposes

Model checking [7, 10] of prepared logical model allows to validate the primary specification of logic controller. It is possible to verify some user-defined properties, which are supposed to be satisfied in designed system.

Logical model derived from a Control Interpreted Petri Nets presented at RTL-level (*Register Transfer Level*) in such a way, that it is easily synthesizable as reconfigurable logic controller or PLC (*Programmable Logic Controller*) without additional changes.

Design methodology at RTL-level allows to convert an algorithm into hardware realization and to use the conception of variables and sequential operation performing. Project

description in VHDL language is a specification accepted by synthesis tools at RTL-level [23]. Therefore, logical model is transformed into synthesizable code in VHDL language.

Presented approach to formal verification of reconfigurable logic controllers was tested on several examples of industrial specifications by means of Control Interpreted Petri Nets. Specifications were firstly written as logical models, then transformed into appropriate formats, and finally formally verified (with some properties added) and synthesized.

As a support for testing, a tool has been developed, which allows automatic transformation of logical model into model description in the NuSMV format and into synthesizable code in hardware description language VHDL.

Rules for definition of rule-based logical model and model description in the NuSMV tool are described in section 3, while rules for synthesizable model definition in VHDL are given in section 4.

2. Description and illustration of proposed RLCs design system

Logic controller development process usually starts with specification, further goes through verification [16] and simulation, finally ending with implementation. Schema of proposed system for designing of logic controllers is presented in Figure 2.

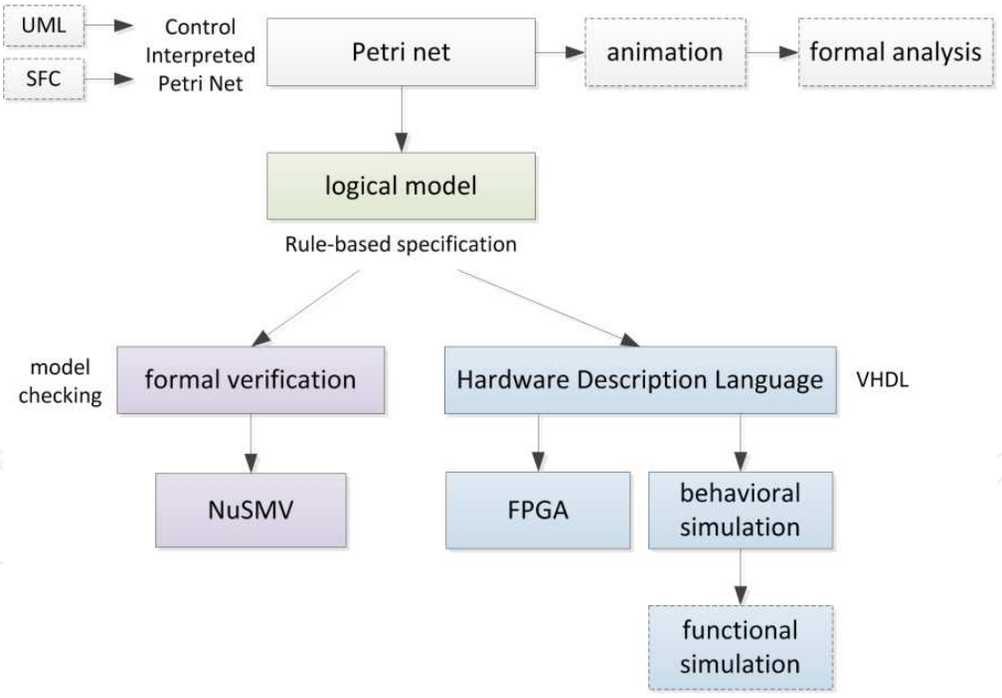

Figure 2. Schema of proposed system for designing of logic controllers

Formal specification is prepared by means of Control Interpreted Petri Nets [8]. They specify and model the behaviour of concurrent logic controllers and take into account properties of controlled objects. Local states, as in typical P/T Petri nets, may change after firing of transitions, if some events occur. Additionally, transition guards are associated with input signals of controller, while places are associated with its output signals.

Formally, a Control Interpreted Petri Net can be defined as a six-tuple:

$$CIPN = (PN, X, Y, \varrho, \lambda, \gamma) \quad (1)$$

where:

- PN is an alive and safe Petri net,
- X is a set of input states,
- Y is a set of output states,
- ρ is a function $T \rightarrow 2^X$, that each transition assigns the subset of input states $X(T)$; 2^X states for the set of all possible subsets of X,
- λ is a function of Moore outputs $M \rightarrow Y$, that each marking M assigns the subset of output states $Y(M)$,
- γ is a function of Mealy outputs $(M \times X) \rightarrow Y$, that each marking M and input states X assigns the subset of output states Y.

3. Novel approach to formal verification of logic controller specification

Control Interpreted Petri Net is first written as an abstract rule-based logical model. Then, basing on that model two other models are built – a verifiable model for the NuSMV model checker (described in details in section 3.2, together with requirements list definition expressed in temporal logic) and a synthesizable model in VHDL (for reconfigurable logic controllers, discussed in section 4). Thanks to proposed methodology, synthesized model is formally verified before the implementation and the two models are fully consistent with each other.

3.1. Rule-based logical model of a Control Interpreted Petri Net

Proposed rule-based logical model used for synthesis and verification purposes is an intermediate format describing desired behaviour of designed logic controller [13, 14]. Model includes variables definition and their initial values, rules describing net functionality, changes of logic controller output and input signal values.

Proposed logical model reflects the behaviour of Moore digital automaton with inputs register (optionally) and outputs register (Figure 3). Combinational circuit (*CC*) controls system behaviour and operates on internal system states.

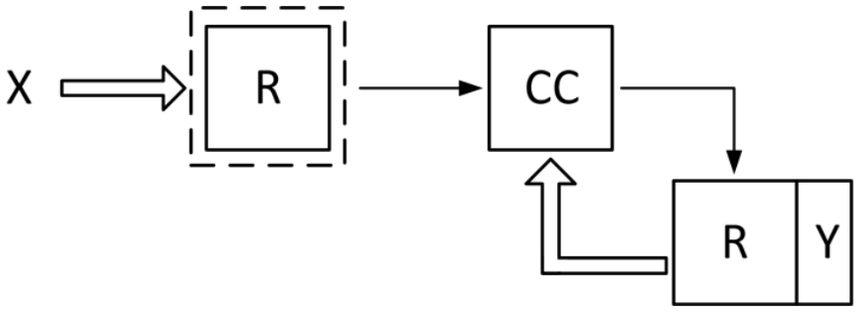

Figure 3. Moore digital automaton with inputs and outputs register

Formally, rule-based logical model can be defined as a seven-tuple:

$$LM = \{P, X, Y, S, T, O, I\} \quad (2)$$

where:

- *P* stands for places of a Control Interpreted Petri Net (internal local states),
- *X* stands for inputs of a Control Interpreted Petri Net (input signals to logic controller),
- *Y* stands for outputs of a Control Interpreted Petri Net (output signals to logic controller),
- *S* stands for initial values of places, inputs and outputs,
- *T* stands for rules describing transitions (indicating changes of local states),
- *O* stands for active outputs corresponding to appropriate places,
- *I* stands for inputs supposed to be active in appropriate places (for formal verification simplification).

As an example to demonstrate proposed solution a sample control process was chosen, described by means of Control Interpreted Petri Nets, then formally verified for behavioral properties and synthesized. Control process example was taken from. It was verified using CTL temporal logic and the NuSMV model checker in 2.5.2 version [19].

A simple embedded system for drink production is considered (Figure 4).

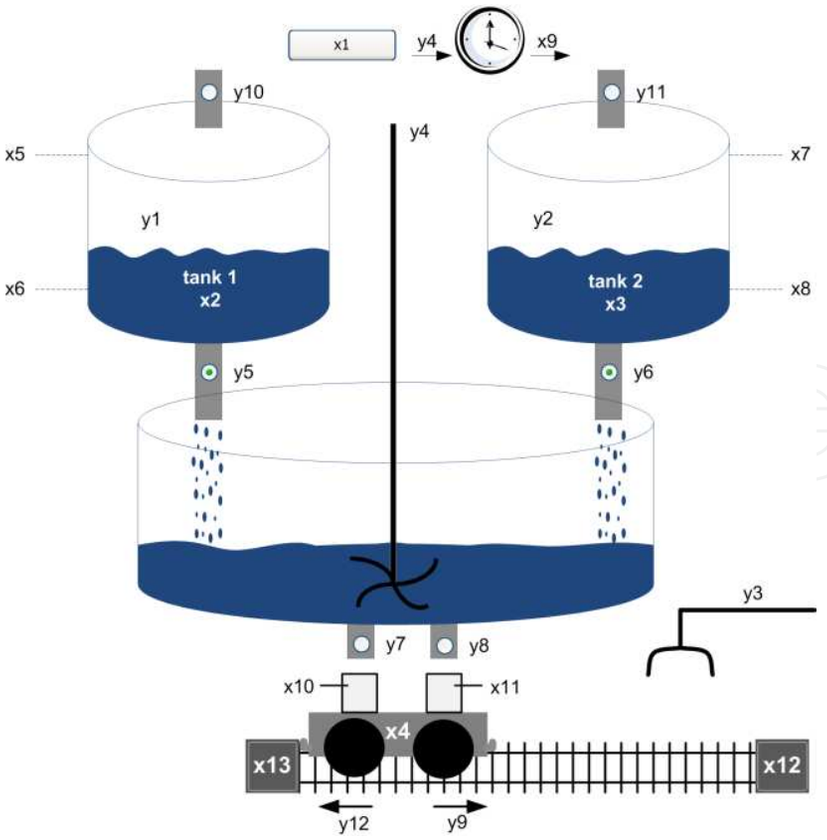

Figure 4. Real model of process for drink production

Logic controller schema with input and output signals (Table 1) is presented in Figure 5. A Control Interpreted Petri Net for drink production process is presented in Figure 6. It has 20 local states and initial marking involves two places – *P1* and *P14*.

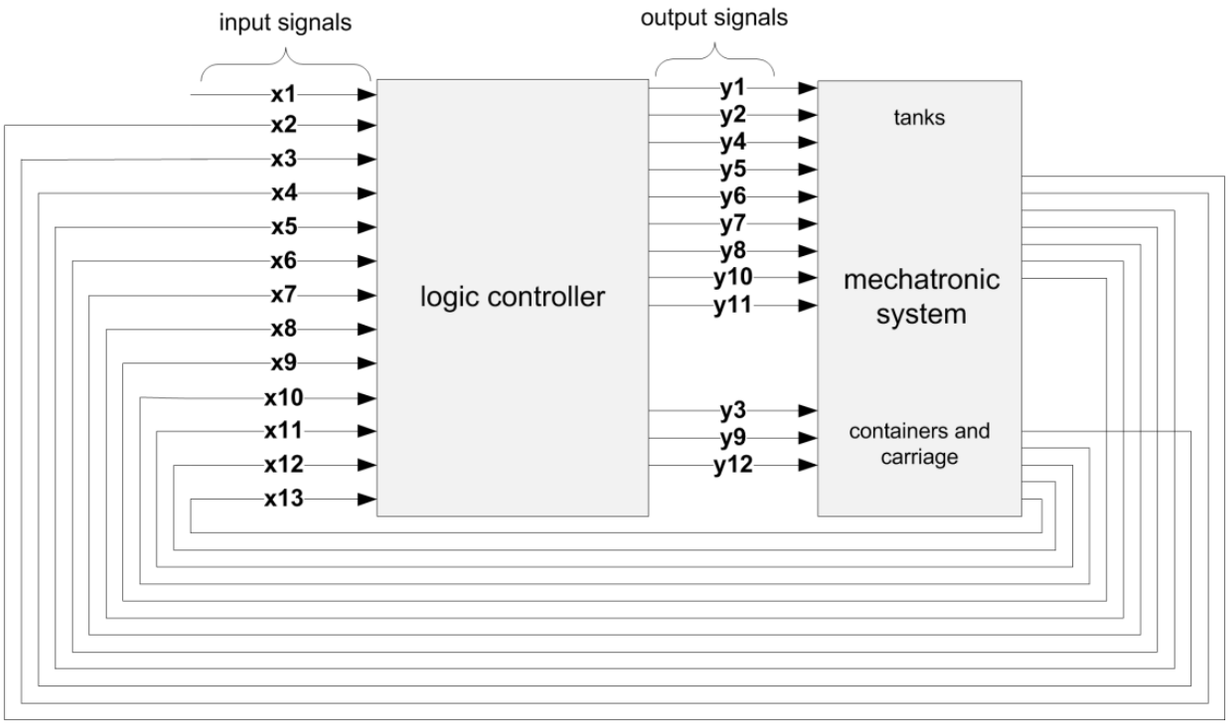

Figure 5. Logic controller schema

Initially, both tanks are empty and process can be started. After pressing the *x1* button, drink production process starts. Valves *y10* and *y11* are opened and target containers are loaded on the carriage (*y3*). Filling tanks process (active signals *y1* and *y2*) is a concurrent process. When a tank is already full (signalized by sensor *x5* or *x7* respectively), the appropriate valve for filling tank is closed. Meanwhile, loaded containers and transported (*y12*) to a proper location (sensor *x13*). When the ingredients are ready, it is signalized by sensors *x2*, *x3* and *x4*. Then, ingredients from both tanks are dropped into the main tank (signals *y5* and *y6*), where they are mixed (signal *y4*). Emptying of small tanks is signalized by sensors *x6* and *x8*. When the drink is well mixed, it is indicated by sensor *x9*. Then, ready drink is filled into containers (*y7*, *y8*). When containers filling process ends (sensors *x10*, *x11*), they are transported (signal *y9*) to their starting location (sensor *x12*).

A Control Interpreted Petri Net is written formally using temporal logic [15]. Logic representation well corresponds to net structure and behavior, and at the same time is easy to formally verify and to synthesize.

Logical model includes variables definition and their initial values. The following elements of Control Interpreted Petri Net are interpreted as model variables: places, input and output signals. Logical model involves also set of rules, which describe how defined variables change over time. Set of rules influences the system behaviour. Each rule (transition) is presented in a separate row and starts with transition name (a label for particular rule).

	Signal	Description
Inputs	*x1*	Signal to start the process
	x2	Ingredients preparation in the first tank is finished
	x3	Ingredients preparation in the second tank is finished
	x4	Containers preparation is finished
	x5	Maximal fluid level in the first tank
	x6	Minimal fluid level in the first tank
	x7	Maximal fluid level in the second tank
	x8	Minimal fluid level in the second tank
	x9	Drink preparation is finished
	x10	Filling of the first container is finished
	x11	Filling of the second container is finished
	x12	The carriage is in its starting location (the right side)
	x13	The carriage is in its target location (the left side)
Outputs	*y1*	Preparation of the first ingredient
	y2	Preparation of the second ingredient
	y3	Loading containers
	y4	Mixing ingredients
	y5	Valve for emptying the first tank
	y6	Valve for emptying the second tank
	y7	Valve for filling the first container
	y8	Valve for filling the second container
	y9	Carriage movement to the right
	y10	Valve for filling the first tank
	y11	Valve for filling the second tank
	y12	Carriage movement to the left

Table 1. Logic controller input and output signals

A rule consists of two separated parts. The first part contains conditions for transition firing, namely names of active places and input signals (if required) needed to fire the transition. If the condition involves more than one variable, variables are usually connected with a logical operator *and* (written as &). It is also possible to connect the variables with logical operator *or* (written as |), what can be used by transition activation with one of many input signals. Similar as by initial values of variables, a variable can take the *TRUE* value (active place / input signal) or the *FALSE* value (inactive input signal). The second part describes marking changing of Petri net places. Usage of a temporal logic operator *X* indicates that marking changing will take place in the next system state. Analogously to previous possible variable values, after transition firing some places can become active (transition output places) or inactive (transition input places, names of these variables are preceded by an exclamation mark). Proposed solution is focused on transitions. Here, transition input places, firing conditions (corresponding to appropriate combinations of input signals) and transition output places are taken into account.

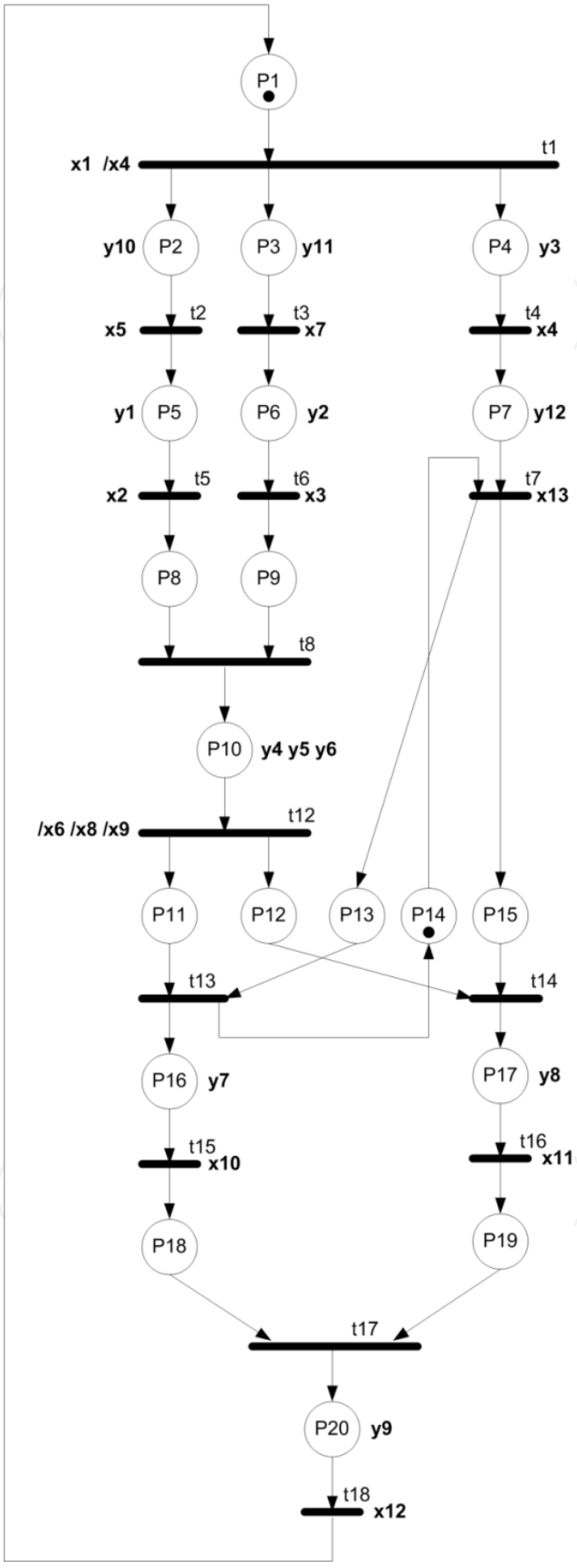

Figure 6. Control Interpreted Petri Net

Transitions from net from Figure 6 are described as separate rules (Figure 7). Firing of each transition changes marking of its input and output places. For example, firing of the *t1* transition removes token from the *p1* place (expressed by *!p1*) and adds a token into three places staring three concurrent processes: *p2*, *p3* and *p4* (expressed by *p2 & p3 & p4*).

```
t1: p1 & x1 & !x4 -> X (!p1 & p2 & p3 & p4);
t2: p2 & x5 -> X (!p2 & p5);
t3: p3 & x7 -> X (!p3 & p6);
...
```

Figure 7. Set of rules in logical model

Places not mentioned in particular rule do not change marking after firing of the (considered) transition. It means that the particular rule does not change marking of not mentioned places. Rules correspond therefore to Petri net transitions firings, and ipso facto marking changing of places. Situations, when a transition cannot be realized are not considered, supposing that active places hold then their marking. Proposed approach is an inertial description oriented on transitions (based on publications [1]. In the paper [11], because of the presence of Mealy outputs (where output signals values depend on input signal values and current internal system state), additionally output signals connected with particular transition firing are taken into account, besides places and input signals.

Output signals are considered for successive Petri net places. If the activity of particular output signal is connected with more than one place, this signal occurs multiple times on the right side of an arrow, by different places. Proposed notation concerns Moore outputs, where output state depends only from internal state of the system.

Output signals from the net in Figure 6 are therefore assigned to places, in which they are active (Figure 8). For example, the *y10* output signal is active only by active marking of the *p2* place, and active marking of the *p10* place implies the activity of output signals *y4*, *y5* and *y6*. The other output signals, which are not present on the right side of particular rule (for particular places) remain default inactive. It is also possible to evidently indicate the activity or inactivity of output signal, as in [1], proposed solutions seems however to be intuitive and does not enforce additional information, which could negative influence its readability.

```
p2 -> y10;
p3 -> y11;
p4 -> y3;
...
p10 -> y4 & y5 & y6;
...
```

Figure 8. Output signals in logical model

Input signals changes are also defined in logical model. 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 coming from different objects, supervising system or system operator are considered analogously like output signals for successive Petri net places.

Input signals from the net in Figure 6 are assigned to places, where they are essential and may become active (Figure 9). In each other state, the signals remains by default inactive. For example, input signal *x5* can be activated, when Petri net marking involves the place *p2*.

```
p1 -> (!x1 | x1) & (!x4 | x4);
p2 -> !x5 | x5;
p3 -> !x7 | x7;
...
```

Figure 9. Input signals in logical model

3.2. Model checking of rule-based logical model

Model checking technique [7, 10] is one of formal verification methods among others like e.g. *theorem proving* or *equivalence checking* and is currently used in the industry in software and hardware production [12]. System model is compared with defined properties and an answer whether they are satisfied or not is given. In case of any detected errors, appropriate counterexamples are generated which allow to localize error source.

Model checking process can be performed on the whole system or just on a part of it (so-called *partial verification*), what has an important meaning especially by complex systems which can be divided into subsystems.

Logical model derived from Control Interpreted Petri Net is transformed into format of the NuSMV model checker according to some strictly specified rules [13, 14]:

a. Each place $p \in P$ is a variable of Boolean type,
b. Each input signal $x \in X$ is a variable of Boolean type,
c. Each output signal $y \in Y$ is a variable of Boolean type,
d. Defined variable take some initial values. Each variable takes any of two values (*TRUE* or *FALSE*),
e. Each place changes according to the rules defined in the transitions T and the function $\rho: T \rightarrow 2^X$; conditions of changes between places (token flow) occur in pairs (groups) – in the previous place(s) and in the next place(s),
f. Each output signal changes according to the rules defined in the function $\lambda: M \rightarrow Y$,
g. Each input signal changes randomly, but can take the expected values connected with Petri net places or change adequately to the situation.

Logical model into NuSMV model description translation is done automatically using implemented software application.

Similar like in logical model, model description for verification starts with variables definition, which correspond to places, input and output signals. Then, initial values are assigned to the variables. Rules describing net behaviour and token flow correspond to values changes of appropriate places (active/inactive marking of places, Figure 10). Input signals change their value only in expected situations (Figure 11). Output signals are in turn active when appropriate places include token (Figure 12).

```
next(p1) := case
   p1 & x1 & !x4  : FALSE;
   p20 & x12      : TRUE;
   TRUE           : p1;
esac;
next(p2) := case
   p2 & x5        : FALSE;
   p1 & x1 & !x4  : TRUE;
   TRUE           : p2;
esac;
...
```

Figure 10. Rules in verifiable model (assignment of next values to places)

```
next(x1) := case
   p1    : {FALSE, TRUE};
   TRUE  : FALSE;
esac;
next(x2) := case
   p5    : {FALSE, TRUE};
   TRUE  : FALSE;
esac;
...
```

Figure 11. Assignment of next values to input signals in verifiable model

```
next(y1) := case
   p5    : TRUE;
   TRUE  : FALSE;
esac;
next(y2) := case
   p6    : TRUE;
   TRUE  : FALSE;
esac;
...
```

Figure 12. Assignment of next values to output signals in verifiable model

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 defined model. Structural properties can also be checked on the Petri net level (and do not require model checking technique). However, the most important are here behavioural properties, which describe system functionality, impact of input signals and output signals activity.

Properties to be checked are defined using temporal logic [6, 15, 20] – either LTL (*Linear Temporal Logic*) or CTL (*Computation Tree Logic*). Properties 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. It is often written basing on an informal specification. In the best practices, it is specified by customer (in textual form) or by engineers not involved in design process (in more or less formalized way).

Using CTL temporal logic the requirements list for considered case study was defined (properties are listed in Figure 13 and described in details in Table 2). All specified requirements are satisfied in the corresponding model description. Some properties concern Petri net structure itself (properties 1 – 20). It is checked, whether particular places are reachable. Next properties describe output signals, which cannot be active at the same time (properties 21 – 23). The last part of properties regards the correlation of input and output signals.

```
CTLSPEC EF p1;                --1
...                           ...
CTLSPEC EF p20;               --20
CTLSPEC AG !(y5 & y10);       --21
CTLSPEC AG !(y6 & y11);       --22
CTLSPEC AG !(y9 & y12);       --23
CTLSPEC AG (x5 -> AF !y10);   --24
CTLSPEC AG (x7 -> AF !y11);   --25
CTLSPEC AG (x13 -> AF !y12);  --26
CTLSPEC AG (x12 -> AF !y9);   --27
```

Figure 13. Requirements list

Property	Description
1	It is possible to reach the *p1* place
...	...
20	It is possible to reach the *p20* place
21	The *y5* and *y10* output signals can never be active at the same time
22	The *y6* and *y11* output signals can never be active at the same time
23	The *y9* and *y12* output signals can never be active at the same time
24	Always, when the *x5* input signal is active (maximal fluid level in the first tank), finally the *y10* output signal (controlling valve for filling the first tank) becomes inactive
25	Always, when the *x7* input signal is active (maximal fluid level in the second tank), finally the *y11* output signal (controlling valve for filling the second tank) becomes inactive
26	Always, when the *x13* input signal is active (carriage location on the left), finally the *y12* output signal (carriage movement to the left) becomes inactive
27	Always, when the *x12* input signal is active (carriage location on the right), finally the *y9* output signal (carriage movement to the right) becomes inactive

Table 2. Requirements list description

By introducing a subtle modification into Control Interpreted Petri Net, which regards initial marking removing from place *p14* (initial marking involves then only the *p1* place), the corresponding part of logical model and NuSMV model description is also changed. However, such a subtle change dramatically changes net behavior, and thereby designed logic controller behavior. Model checking of the same properties shows now another results. User receives multiple generated counterexamples indicating unsatisfied requirements. Places *p1* to *p12* are reachable, but it is not possible to reach active marking of further places. Next to last requirement is also not satisfied (*CTLSPEC AG (x13 -> AF !y12)*). Summarizing the report – an error occurs starting from transitions *t7* and t*13*, what confirms the fact, that it is indeed correlated with additional initial marking (and actually the lack of it) of Control Interpreted Petri Net.

When model checking process does not indicate any errors, it is then possible and advisable to focus on synthesizable code. Basing on logical model, model in hardware description language VHDL is built. The model is fully synthesizable and may be then implemented in FPGA for a reconfigurable logic controller.

4. Synthesis of rule-based logical model

Combining FPGA [18] as a target hardware platform with hardware description language VHDL ensures high reliability, speed and safety. Additionally, it is possible to modify anytime the already running system, what has a practical sense. Direct implementation of concurrent logic controllers in FPGA is similar to rule-based realization based on classical sequence diagrams. Transition firings are synchronized with clock rising edge.

Control 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. Flip-flops amount (for places) using one-hot encoding is equal to the amount of places (and so to the amount of local states).

Logical model can be easy synthesized as reconfigurable logic controller. Logical model, derived from Control Interpreted Petri Net, is transformed into VHDL language according to some strictly defined rules [13]:

a. Each place is an internal signal of *std_logic* type,
b. Each input signal is an input port of *std_logic* type,
c. Each output signal is an output port of *std_logic* type,
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,
f. Input signals are not considered, as they are inputs to the logic controller,
g. Each output signal changes its value according to active places; output signals are active by active marking of corresponding places.

Model in VHDL is oriented on places and transitions. It can be simulated and synthesized. Synthesis is performed in form of rapid prototyping [5], what in modern methodology for

digital circuits design allows for frequent verification (simulation, analysis) of developed system. Its main goal is to check, whether designed system works at all, but the circuit might be not optimized. Circuit optimization and minimization of resources usage are here out of scope, however they may be important in some fields [9, 18].

Logical model into VHDL model translation is done automatically using implemented software application. Generated VHDL file for considered drink production process is fully synthesizable.

Model for synthesis starts with input and output signals definition. Petri net places are defined as internal signals. By clock rising edge and active reset signal, some initial values are assigned to places, which correspond to initial marking of a Control Interpreted Petri Net. Additionally, by each clock rising edge places hold their heretofore marking.

For places the one-hot encoding was used (called also *isomorphic places encoding*), which is the most accurate (and the simplest) representation of logical model, however it can cause bigger resources usage. For each place one flip-flop is generated, which label corresponds to particular place etiquette. Flip-flop sets the *1* value, if a place contains token, otherwise it holds the *0* value. Additionally, one-hot encoding is recommended by implementation in FPGA circuits, and even seen as the most effective method for states encoding [23], i.e. in FPGA circuits of Xilinx [21], especially for small automata. It is also possible to extend the work to any other encoding.

Places marking can change after transitions firing. Conditions connected with transitions correspond to values of input signals and active marking of particular places. If a condition is satisfied, Petri net transition is realized, and thereby its input and output places change their marking (Figure 14).

```
if p1 = '1' and x1 = '1' and x4 = '0' then
   p1 <= '0';
   p2 <= '1';
   p3 <= '1';
   p4 <= '1';
end if;
```

Figure 14. The *t1* transition firing in VHDL model

Output signals are active by active marking of appropriate places, what is denoted as shown in Figure 15.

```
y1 <= p5;
y2 <= p6;
y3 <= p4;
...
```

Figure 15. Outputs assignment in VHDL model

Input signals value changes come from outside and are not modified inside VHDL model file. Their values are just read out by conditions related to transition firings.

VHDL file can also be simulated i.e. in *Active-HDL* environment [4]. Simulation confirms the proper functionality of designed logic controller (simulation results are presented in Figure 16).

It is then possible to perform logic synthesis and implementation, i.e. in *Xilinx PlanAhead* environment, in version *13.1* [21]. Sample resources usage for the *xa6slx4csg225-2* circuit from *Spartan6* family of *XILINX* [22] is listed in Table 3.

Resource	Utilization	Available	Utilization
Register	18	4800	1%
LUT	18	2400	1%
Slice	6	600	1%
IO	27	132	20%
Global Clock Buffer	1	16	6%

Table 3. Resources usage

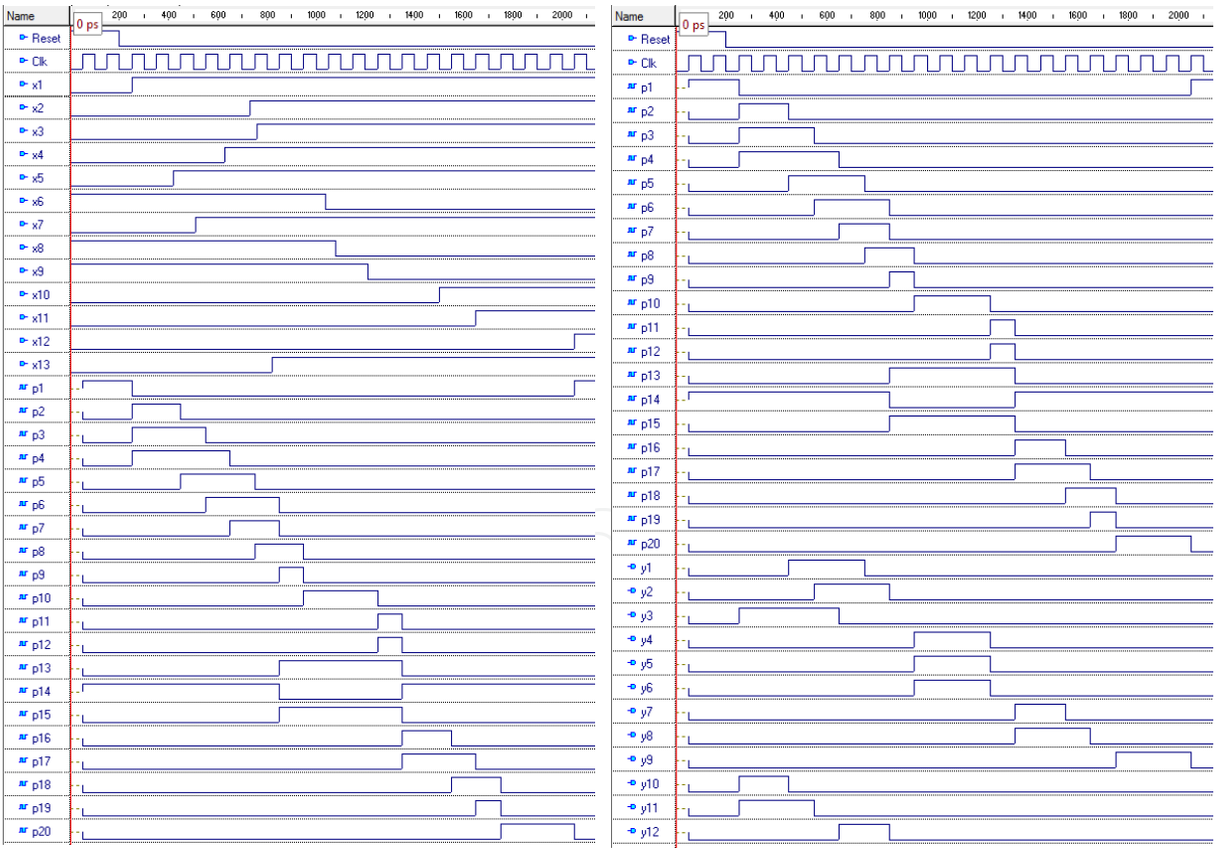

Figure 16. Simulation results in *Active-HDL*

It is also possible to transform logical model into synthesizable code in Verilog language [9, 17], this aspect is however not discussed further in this chapter.

5. Summary and conclusions

Proposed novel approach to verification of reconfigurable logic controller programs and specification by means of Control Interpreted Petri Nets allows to detect even subtle errors on an early stage of system development. Rule-based representation of Control Interpreted Petri Nets in temporal logic is presented at RTL-level and is easy to formally verify using model checking technique and to synthesize using hardware description languages.

Results of the work include the assurance that verified behavioural specification in temporal logic will be an abstract program of matrix reconfigurable logic controller. Hence, logic controller program (its implementation) will be valid according to its primary specification. This may shorten the duration time of logic controllers development process (as early discovered errors are faster corrected) and, consequently, save money (as project budgets will not be exceeded).

Furthermore, formal verification can improve the quality of final products, making them work more reliable. And even if a logic controller, already delivered to customer, will not work properly (it can always happen that some subtle error was overseen or that the specification was incomplete), it is possible to find error source using available techniques (verification, simulation, etc.). Then, some part of corrected system (or the whole system) may be one more time formally verified using extended requirements list and modified logical model.

Future research directions include i.e. (but are not limited to) model checking of other forms of logic controllers specification and mechanisms for behavioural properties specification.

Author details

Iwona Grobelna
University of Zielona Góra, Poland

6. Acknowledgement

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.

7. References

[1] Adamski, M & Monteiro, J. L., From Interpreted Petri net specification to Reprogrammable Logic Controller Design, In: *Proceedings of the IEEE International Symposium on Industrial Electronics*, 2000, Vol. 1, pp. 13 – 19.

[2] Adamski, M.A.; Karatkevich, A. & Węgrzyn, M.. *Design of embedded control systems*, Springer Verlag ; 2005.
[3] Adamski, M.; Kołopieńczyk, M. & Mielcarek, K.. Perfect Petri Net in parallel control circuits (in Polish), *Measurement Automation and Monitoring*, 2011, Vol. 57, No. 6, pp. 656 – 660.
[4] Aldec home page. The producer of Active-HLD environment. http://www.aldec.com/ (access 15.04.2012)
[5] Andreu, D.; Souquet, G. & Gil, T.. Petri Net based rapid prototyping of digital complex system, *IEEE Computer Society Annual Symposium on VLSI 2008*, pp. 405 – 410.
[6] Ben-Ari, M., *Mathematical logic for computer science*, Springer Verlag ; 2001.
[7] Clarke, E.M.; Grumberg, O. &Peled, D.A., Model checking, The MIT Press ; 1999.
[8] David, R. & Alla, H., *Discrete, Continuous, and Hybrid Petri Nets*, Springer Verlag ; 2010.
[9] De Micheli, G., *Synthesis and Optimization of Digital Circuits*, McGraw-Hill Higher Education; 1994.
[10] Emerson, E.A., The Beginning of Model Checking: A Personal Perspective, In: *25 Years of Model Checking: History, Achievements, Perspectives*, O. Grumberg, H. Veith (Ed.), Springer Verlag ; 2008, pp. 27 – 45.
[11] Fernandes, J.M. ; Adamski, M. & Proenca, A.J., VHDL generation from hierarchical Petri net specifications of parallel controllers, *IEE Proceedings – Computers and Digital Techniques*, 1997, Vol. 144, No. 2, pp. 127 – 137.
[12] Fix, L., Fifteen years of formal property verification in Intel, In: O. Grumberg, H. Veith (Ed.), *25 Years of Model Checking: History, Achievements, Perspectives*, Springer Verlag ; 2008, pp. 139 – 144.
[13] Grobelna, I., Formal verification of embedded logic controller specification with computer deduction in temporal logic, Electrical Review, 2011, nr 12a, 2011, pp. 47 – 50.
[14] Grobelna, I. & Adamski, M., Model Checking of Control Interpreted Petri Nets, *Proceedings of the 18th International Conference Mixed Design of Integrated Circuits and Systems 2011*, pp. 621 – 626 (available in IEEE Xplore).
[15] Huth, M. & Ryan, M., *Logic in Computer Science. Modelling and Reasoning about Systems*, Cambridge University Press ; 2004.
[16] Kropf, T., *Introduction to Formal Hardware Verification*, Springer Verlag ; 1999.
[17] Minns, P. & Elliott, I., *FSM based Digital Design using Verilog HDL*, Wiley ; 2008.
[18] Nemec, J., Stoke the fires of FPGA design, *Electronic design*, 1994, Vol. 42, Issue 22, pp. 97 – 105.
[19] NuSMV model checker homepage: http://nusmv.fbk.eu/ (access 15.04.2012)
[20] 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, Lecture Notes in Computer Science, Vol. 2031, Springer Verlag; 2001, pp. 1 – 22.
[21] Xilinx homepage. The producer of XILINX ISE and XILINX PlanAhead software. http://www.xilinx.com (access 15.04.2012)
[22] Xilinx FPGA Spartan6 family home page. http://www.xilinx.com/products/silicon-devices/fpga/spartan-6/index.htm(access 15.04.2012)
[23] Zwoliński, M., *Digital System Design with VHDL*, Prentice Hall; 2004.