Tomasz GIDLEWICZ, **Arkadiusz BUKOWIEC**
UNIVERSITY OF ZIELONA GÓRA, Faculty of Electrical Engineering, Computer Science and Telecommunications,
Institute of Computer Engineering and Electronics, ul. Podgórna 50, 65-246 Zielona Góra

Implementation of algorithm of Petri net architectural synthesis into FPGA

Tomasz GIDLEWICZ, B.Sc.

Tomasz Gidlewicz obtained B.Sc. degree (2013) from University of Zielona Góra with specialization engineering of microcomputers systems. Now, he is a master student at the same University.

e-mail: tgidlewicz@gmail.com

Arkadiusz BUKOWIEC, Ph.D.

Arkadiusz Bukowiec obtained B.Sc. degree (2001) from Technical University of Zielona Góra and M.Sc. degree (2004) with a specialization in computer engineering at University of Zielona Góra. In 2008 he defended his Ph.D. thesis with distinction at Faculty of Electrical Engineering, Computer Science and Telecommunications of University of Zielona Góra in the field of computer science. Since 2004, he works at Institute of Computer Science and Electronics at University of Zielona Góra, as an assistant professor.

e-mail: a.bukowiec@iie.uz.zgora.pl

Abstract

In the paper an implementation of algorithm of Petri net array-based synthesis is presented. The method is based on decomposition of colored interpreted Petri net into subnets. The structured encoding of places in subnets is done of using minimal numbers of bits. Microoperations, which are assigned to places, are written into distributed and flexible memories. It leads to realization of a logic circuit in a two-level concurrent structure, where the combinational circuit of the first level is responsible for firing transitions, and the second level memories are used for generation of microoperations. This algorithm is implemented in C# and delivered as library.

Keywords: C#, decomposition, FGPA, logic synthesis, Petri net.

Implementacja algorytmu syntezy blokowej sieci Petriego do układu FPGA

Streszczenie

W artykule przedstawiono implementację algorytmu syntezy sieci Petriego z wykorzystaniem metod dekompozycji blokowej. Metoda opiera się o dekompozycję pokolorowanej sieci Petriego na podsieci automatowe. Kodowanie miejsc w każdej podsieci wykonane jest na minimalnej ilości bitów. Mikrooperacje przypisane do miejsc zostają umieszczone w pamięci wewnętrznej układu. Prowadzi to do realizacji układu logicznego w dwu poziomowej strukturze, gdzie układ pierwszego poziomu odpowiedzialny jest za generowanie funkcji odpalania tranzycji a układ drugiego poziomu za generowanie mikrooperacji. Algorytm ten został zaimplementowany w języku C# i dostarczony jako niezależna biblioteka.

Słowa kluczowe: C#, dekompozycja, FGPA, synteza logiczna, sieć Petriego

1. Introduction

Application specific logic controllers (ASLC) [1] are one of the most popular group of electronic devices. They can be designed as dedicated software for microcontroller or as dedicated hardware. The second approach gives more possibilities of system integration as system on programmable chip (SoPC) with use of field programmable gate arrays (FPGAs). The most classical way of designed such controllers is application of hardware description languages (HDLs) but it is unconformable for designer and potentially it gives high risk of human mistake. The usage of graphical representation of algorithm is much more conformable [2]. In this case Petri nets (PNs) [3] are one of the most adequate methods for formal design of such devices. It gives easy way for representation of concurrent processes and additionally there could be applied mathematical algorithms for formal analysis and verification of the designed model [4]. There are also several algorithms of direct synthesis of Petri net model into FPGA devices [5, 6]. The most typical implementation of Petri nets into FPGA devices use one-hot local state encoding where each single place is represented by a flip-flop [6]. Such an approach requires hardware implementation of a large number of several logic functions and flip-flops included in macrocells.

One of the main features of FPGA is an existence of separated logic elements (look-up tables) with restricted fixed number of inputs.

Very frequently logic functions have more arguments than number of inputs of such logic element. It forces a functional decomposition during a synthesis process and consumes a large number of logic elements. One of the methods of decreasing a number of such functions is architectural decomposition of a sequential circuit [7]. Such methods introduce several additional internal variables and very often consume more hardware than typical direct implementation. This issue can be resolved by using logic elements together with embedded memory blocks that are available in modern FPGA devices.

There is presented implementation of a synthesis algorithm [7] that allow to decrease the number of implemented logic functions depending on inputs and internal variables of Petri net-based logic controller. The logic functions are going to be synthesized with use of logic elements and embedded memory blocks. To permit the minimal local state encoding the Petri net is initially colored and it is compacted into macro Petri net [8]. Macroplaces that are colored by the same color create one state machine module. Each memory block controls only microoperations that belong to the subnet with the same color.

2. Colored interpreted macronet

An interpreted Petri net [6,7, 8] is an extension of a simple Petri net [3] about a feature for information exchange. This exchange is made by use of binary signals. It is required for a models of concurrent logic controllers [1, 6] to establish communication with environment.

Now, a transition can be fired if all its input places are marked and a condition assigned to this transition returns value true. Like in simple Petri nets, firing of a transition removes tokens from its input places and puts one token in each output place. The condition is defined as Boolean function of the input variables.

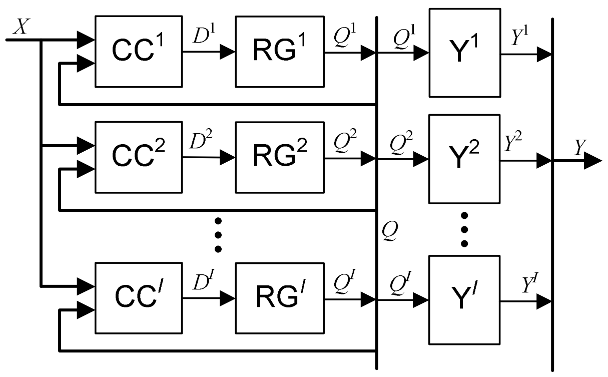
Elementary conjunctions of affirmation of some output variables are associated to places. If the place is marked the output variables from corresponding conjunction are being set otherwise they are being reset.

An interpreted Petri net can be extended with a hierarchy by application of macroplaces [4]. A macroplace correspond to a part of a net. In this article we make frequent use of mono-active macroplaces, that are limited to have one input and one output and consist of only sequential places.

A Petri net can be enhanced by assigning colors to places and transitions [3, 8]. In state machine (SM) colored Petri net colors help to validate intuitively and formally the consistency of all sequential processes covering the considered Petri net. Each color recognizes one SM-subnet.

3. Synthesis algorithm overview

The idea of synthesis method is based on the minimal local states encoding of places together with functional parallel decomposition of the Petri net-based logic circuit. Places are encoded separately in every colored subset. Output variables assigned to places are placed in configured memories of FPGA. It leads to realization of a logic circuit in two-level structure (Fig. 1), where the combinational circuits of first level are responsible for generation of the excitation functions. The memory of the circuit is built from concurrent D-type registers which hold a current state of each subnet. The second level decoders are responsible for generation of microoperations and they are implemented using memory blocks.



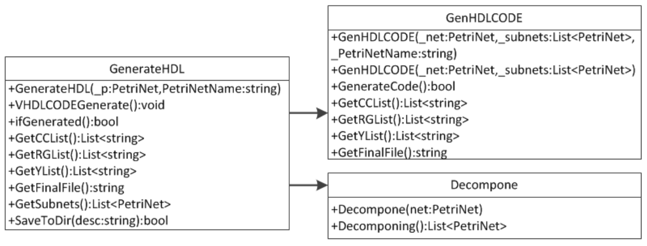
Rys. 1. Układ logiczny sterownika logicznego
Fig. 1. Logic circuit of ASLC

The synthesis process includes following steps:

1. Formation of subnets,
2. Encoding of places,
3. Formation of conjunctions and logic equations,
4. Formation of memory contents,
5. Formation of logic circuit and implementation.

4. Implementation of synthesis algorithm

The algorithm was implemented in C# in Microsoft .NET environment. The whole algorithm was implemented with use of three classes (Fig. 2).


GenHDLCODE
+GenHDLCODE(_net:PetriNet,_subnets:List<PetriNet>,
_PetriNetName:string)
+GenHDLCODE(_net:PetriNet,_subnets:List<PetriNet>)
+GenerateCode():bool
+GetCCList():List<string>
+GetRGList():List<string>
+GetYList():List<string>
+GetFinalFile():string
GenerateHDL
+GenerateHDL(_p:PetriNet,PetriNetName:string)
+VHDLCODEGenerate():void
+ifGenerated():bool
+GetCCList():List<string>
+GetRGList():List<string>
+GetYList():List<string>
+GetFinalFile():string
+GetSubnets():List<PetriNet>
+SaveToDir(desc:string):bool
Decompone
+Decompone(net:PetriNet)
+Decomponing():List<PetriNet>

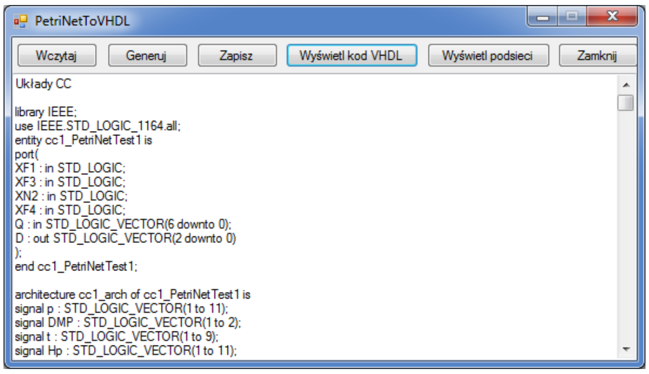
Rys. 2. Diagram klas algorytmu syntezy
Fig. 2. Class diagram of synthesis algorithm

The class *Decompone* is responsible for decomposition of macronet into concurrent macrosubnets. Then each macrosubnet is extracted into flat subnet.

The other steps of synthesis algorithm are performed by methods from other two classes. All equations are generated directly into VHDL syntax. It allows easy and fast further generation of description of hole circuit in VHDL.

The whole algorithm was compiled into *DecomponeAndVHDLCodeGen.dll* library. There are public only methods for generation VHDL code. Other methods, which perform particular steps of synthesis algorithm, are internal and not available for end-user. They are invoked automatically by main method.

The designed library was used for implementation sample application *PetriNetToVHDL*. The application reads Petri net in PNML format extended with nodes for interpretation. Such description can be obtained from graphical editors for Petri nets. After reading the synthesis can be performed and the result can be saved in chosen director or displayed on the screen (Fig. 3).



```
Układy CC

library IEEE;
use IEEE.STD_LOGIC_1164.all;
entity cc1_PetriNetTest1 is
port(
XF1 : in STD_LOGIC;
XF3 : in STD_LOGIC;
XN2 : in STD_LOGIC;
XF4 : in STD_LOGIC;
Q : in STD_LOGIC_VECTOR(6 downto 0);
D : out STD_LOGIC_VECTOR(2 downto 0)
);
end cc1_PetriNetTest1;

architecture cc1_arch of cc1_PetriNetTest1 is
signal p : STD_LOGIC_VECTOR(1 to 11);
signal DMP : STD_LOGIC_VECTOR(1 to 2);
signal t : STD_LOGIC_VECTOR(1 to 9);
signal Hp : STD_LOGIC_VECTOR(1 to 11);
```

Rys. 3. Główne okno aplikacji do syntezy
Fig. 3. Main window of synthesis application

5. Summary

The implementation of the method of synthesis of application specific logic controllers into FPGAs with embedded memory blocks was presented in this article. The special method of logic synthesis [7] is applied. The usage of designed library is fully automated and it could be easily integrated with design tools in CAD system.

The research was financed from budget resources intended for science in 2010–2013 as an own research project No. N N516 513939.

6. References

[1] M. Węgrzyn, P. Wolański, M. Adamski, J. Monteiro, "Coloured Petri net model of application specific logic controller programs", w: Proceedins of IEEE International Symposium on Industrial Electronics – ISIE'97, vol. 1. Guimarães, Portugalia: Piscataway, 1997, ss. 158-163.

[2] G. Łabiak, "From UML statecharts to FPGA – the HiCoS approach", w: Proceedings of the Forum on Specification & Design Languages – FDL'03. Frankfurt, Niemcy: ECSI, 2003, ss. 354–363.

[3] T. Murata, "Petri nets: Properties, analysis and applications", Proceedings of the IEEE, vol. 77, no. 4, ss. 541–580, 1989.

[4] J. Esparza, M. Silva, "On the analysis and synthesis of free choice systems", w: Advances in Petri Nets 1990, ser. Lecture Notes in Computer Science, G. Rozenberg, Ed. Berlin/Heidelberg: Springer-Verlag, 1991, vol. 483, ss. 243–286.

[5] L. Gomes, A. Costa, J. Barros, P. Lima, "From Petri net models to VHDL implementation of digital controllers", w: 33rd Annual Conference of the IEEE Industrial Electronics Society – IECON'07. Taipei, Taiwan: IEEE, 2007, ss. 94–99.

[6] M. Adamski, M. Węgrzyn, "Petri nets mapping into reconfigurable logic controllers," Electronics and Telecommunications Quarterly, vol. 55, no. 2, ss. 157–182, 2009.

[7] A. Bukowiec, M. Adamski, "Synthesis of Macro Petri Nets into FPGA with Distributed Memories", International Journal of Electronics and Telecommunications, vol. 58, no. 4, ss. 403–410, 2012.

[8] M. Adamski, J. Tkacz, "Formal reasoning in logic design of reconfigurable controllers", w: Proceedings of 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems – PDeS'12, Brno, Czechy, 2012, ss. 1–6.