Jacek KLUSKA, Zbigniew HAJDUK, Lesław GNIEWEK
POLITECHNIKA RZESZOWSKA, KATEDRA INFORMATYKI I AUTOMATYKI

Synteza rozmytych sieci Petriego jako sprzętowych układów sterowania i diagnostyki

Dr hab. inż. Jacek KLUSKA

W 1983 r. uzyskał stopień doktora w ICT Politechniki Wrocławskiej. Stopień dra hab. uzyskał w 1995 r. na Wydziale Elektrotechniki, Automatyki, Informatyki i Elektroniki, AGH w Krakowie. Obecnie prof. nadzw. Politechniki Rzeszowskiej. Jego publikacje dotyczą projektowania systemów czasu rzeczywistego, stabilności i projektowania układów regulacji rozmytej, sztucznej inteligencji, sieci Petriego i rozmytych systemów ekspertowych.

e-mail: jacklu@prz-rzeszow.pl

Mgr inż. Zbigniew HAJDUK

Absolwent Wydziału Elektrycznego Politechniki Rzeszowskiej (1998, specjalność aparatura elektroniczna). Od 1999 r. zatrudniony w Politechnice Rzeszowskiej jako asystent. Zamknął przewód doktorski na Wydziale Elektrotechniki, Informatyki i Telekomunikacji Uniwersytetu Zielonogórskiego. Jego publikacje dotyczą projektowania sprzętu cyfrowego z wykorzystaniem układów programowalnych i mikrokontrolerów jednoukładowych.

e-mail: zhajduk@prz-rzeszow.pl

Dr inż. Lesław GNIEWEK

Absolwent Wydziału Elektrycznego Politechniki Rzeszowskiej (1991, specjalność automatyka i metrologia). Od 1991 r. zatrudniony w Politechnice Rzeszowskiej. W 1999 r. uzyskał stopień doktora nauk technicznych na Wydziale Informatyki i Zarządzania Politechniki Wrocławskiej. Obecnie adiunkt w Politechnice Rzeszowskiej w Katedrze Informatyki i Automatyki. Jego publikacje dotyczą projektowania sprzętu cyfrowego, logiki rozmytej, sieci Petriego i programowalnych sterowników logicznych.

e-mail: lgniewek@prz-rzeszow.pl

Streszczenie

Rozmyte sieci Petriego (FPN) dobrze nadają się do modelowania algorytmów sterowania procesów złożonych. Dzięki takim modelom można dokonać syntezy zarówno programowych, jak i sprzętowych układów sterujących. Zastosowanie logiki wielowartościowej (rozmytej), prowadzi do układów, które mogą przetwarzać zarówno sygnały analogowe, jak i binarne. Sprzętowe układy sterujące są szczególnie atrakcyjne ze względu na dużą szybkość działania i niski koszt. Dotychczasowe prace dotyczące FPN dotyczyły modelowania algorytmów sterowania. Niniejsza praca zawiera dwa elementy nowości – pokazuje, jak można wykorzystać wartość rozmytych znaczników sieci Petriego, oraz wyjaśnia, że FPN daje się łatwo rozbudować o funkcje diagnostyki w układzie sterowania.

Słowa kluczowe: logika rozmyta, modelowanie, sieci Petriego, układy stero-wania, diagnostyka uszkodzeń, FPGA, Verilog.

Synthesis of Fuzzy Petri Nets as hardware control and diagnosis systems

Abstract

Fuzzy Petri nets are useful for modeling of complex systems. Owing to such models we can synthesize both software and hardware control devices. Application of multivalued logic (fuzzy logic) leads to the systems, which are able to processing both analog, and binary signals. Hardware control systems are especially attractive for the sake of very high speed and low cost. Until now, the works associated with FPN, were applied to modeling of control algorithms. This work provides two elements of novelty – it shows how to use the values of fuzzy markers in the Petri net, and explains that one can easily extend the FPN to solve the diagnostic functions in the control system.

Keywords: fuzzy logic, modeling, Petri nets, control systems, fault diagnosis, FPGA, Verilog.

1. Wstęp

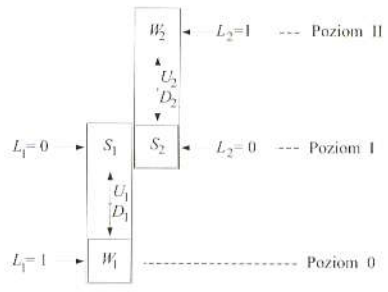
Formalizm sieci Petriego jest wciąż atrakcyjny dla projektantów systemów automatyki. Wykorzystując sieci Petriego i metody oparte na logice Boole'a, rozwinięto wiele procedur służących do syntezy rekonfigurowalnych sterowników logicznych i sterowników równoległych, które mogą sterować procesami złożonymi [1, 6, 13].

Opracowano kilka pożytecznych koncepcji tych sieci do modelowania systemów dynamicznych, które są bliskie idei sieci Petriego rozpatrywanych w niniejszym artykule, z tym, że wykorzystują one na logikę dwuwartościową. Do nich należą 'interpretowalne sieci Petriego'. Idea tych sieci oraz metody projektowania wykorzystujące logikę Booole'a zaowocowały powstaniem Grafcet-u i Sekwencyjnych Diagramów Funkcyjnych (SFC) [2, 5], które wykorzystuje się w programowalnych sterownikach logicznych.

Modele wykorzystujące sieci Petriego i logikę tradycyjną, mogą być bezpośrednio wykorzystane do konstrukcji programowych lub sprzętowych układów sterowania. W przeciwieństwie do rozwiązań programowych opartych na mikroprocesorach, rozwiązania sprzętowe są tanie i gwarantują dużą szybkość przetwarzania informacji. W pracy [12] podano atrakcyjną metodę transformacji sieci Petriego (żywych, bezpiecznych i ograniczonych [14]) do układu logicznego. Jednak otrzymany w ten sposób układ jest zdolny do przetwarzania jedynie informacji binarnej. Procedura ta została znacznie rozszerzona do logiki wielowartościowej (rozmytej) i szerzej opisana w pracach [7, 8, 10, 11]. Opierając się na niej, w niniejszej pracy opisana będzie metoda prawie automatycznego projektowania rozmytej sieć Petriego, jako sprzętowego układu, służącego zarówno do sterowania, jak i diagnostyki. Aby uczynić metodę czytelną, wyjaśnimy ją na przykładzie.

2. Przykład układu sterowania i diagnostyki

Rozpatrzmy układ dwóch wind służących do przewozu towaru z jednego poziomu do drugiego, jak na rys. 1.



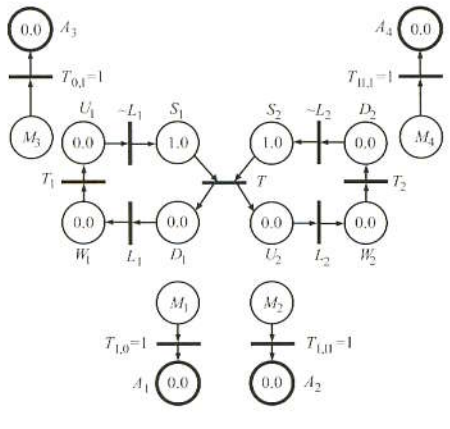
Rys. 1. Układ dwóch wind towarowych
Fig. 1. The system of two elevators

Podobny problem rozpatrywany był w pracy [9], jednak obejmował tylko zagadnienie sterowania. Tutaj rozpatrzymy dodatkowo problem diagnostyki. Dla układu z rys. 1 przyjmujemy następujące oznaczenia:

S_i – stan początkowy i-tej windy, i=1,2,
W_1 – czas postoju pierwszej windy na poziomie 0,
W_2 – czas postoju drugiej windy na poziomie II,
T – wspólny czas postoju wind w stanie początkowym S_1, S_2,
T_1 – czas postoju pierwszej windy na poziomie 0,

T_2 – czas postoju drugiej windy na poziomie II,
L_i – sygnał wskazujący położenie i-tej windy, i=1,2,
D_i – sygnał sterujący dla i-tej windy w kierunku "w dół", i=1, 2; $D_i \geq 0$, ($D_i = 0$ oznacza brak sygnału sterującego),
U_i – sygnał sterujący dla i-tej windy w kierunku "do góry", i=1;2; $U_i \geq 0$, ($U_i = 0$ oznacza brak sygnału sterującego),
$T_{i,j}$ – maksymalny dopuszczalny czas jazdy windy z poziomu „i" na poziom „j". Jest to sygnał binarny. Jeżeli $T_{i,j}$ = 1, to znaczy, że czas został przekroczony, jeżeli $T_{i,j}$ = 0, to czas nie został przekroczony. Dla windy pierwszej brane są pod uwagę czasy: $T_{1,0}$ i $T_{0,I}$, a dla windy drugiej - $T_{I,II}$ oraz $T_{II,I}$.

Sterowanie. Zakładamy, że na początku obie windy znajdują się na poziomie I i są odpowiednio w stanach S_1 i S_2 (rys. 2).



Rys. 2. Rozmyta sieć Petriego jako układ sterowania i diagnostyki
Fig. 2. The fuzzy Petri net as control and diagnosis system

Czujniki wskazują L_1=0 i L_2=0. Po upływie czasu T, który jest wspólnym czasem oczekiwania dla obu wind, pierwsza winda opuszczana jest w dół dzięki sygnałowi D_1, który steruje jej napędem (D_1>0). W tym samym czasie druga winda porusza się do góry dzięki sygnałowi U_2>0. Czujniki L_1 i L_2 wskazują aktualną pozycję wind i zakładamy, że obydwa sygnały z czujników są znormalizowane do przedziału [0,1] (lub [0,100%]). Gdy pierwsza winda osiągnie poziom 0, (L_1=1), wówczas następuje stan oczekiwania W_1, który trwa przez czas T_1. Gdy druga winda osiąga poziom II, (L_2=1), wówczas będzie ona w stanie oczekiwania W_2 przez czas T_2. Po czasie T_1, pierwsza winda porusza się do góry dzięki sygnałowi sterującemu U_1>0 tak długo, aż sygnał z czujnika osiąga wartość L_1=0, (~L_1=1). Następnie pierwsza winda przechodzi do stanu oczekiwania S_1. Po czasie T_2, druga winda porusza się w dół dzięki sygnałowi sterującemu D_2>0 tak długo, aż zostanie spełniony warunek L_2=0, (~L_2=1). Następnie druga winda przechodzi do stanu oczekiwania S_2. Gdy obydwie windy osiągną odpowiednio stany S_1 i S_2, wówczas cały proces może być powtórzony.

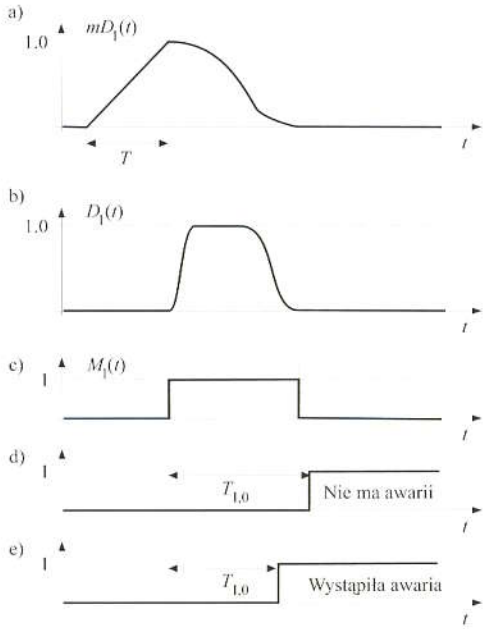
Diagnostyka. Jeżeli sygnał sterujący dla pierwszej windy - D_1 lub U_1, względnie dla drugiej windy - U_2 lub D_2 są niezerowe (dodatnie), to spodziewamy się, że odpowiednia winda osiągnie określony poziom. Tak będzie, o ile układ sterowania jest sprawny. Jednak może się zdarzyć, że pomimo sterowania D_1>0, pierwsza winda startująca z poziomu I nie dotrze do poziomu 0 w czasie dopuszczalnym $T_{1,0}$. Oznacza to awarię. Są 4 takie sytuacje awaryjne, które łatwo zamodelować za pomocą binarnych sieci Petriego o dwóch miejscach i jednej tranzycji. Rozmytą sieć Petriego, która modeluje algorytm sterowania i diagnostyki pokazano na rys. 2.

Początkowe znakowanie sieci jest takie, jak na rys. 2. Sieć modelująca sterowanie i diagnostykę nie jest spójna. Istotną rolę odgrywają 4 specjalne stany znakowania miejsc sieci na rys. 2, jako sygnały: M_1, M_2, M_3 i M_4, które są zawsze dostępne przy zastosowaniu proponowanej metody syntezy technicznej i są istotne z punktu widzenia diagnostyki układu sterowania. Sygnały te zdefiniujemy kolejno, zaczynając od M_1 (patrz rys. 3):

$M_1 = 1$ wtedy i tylko wtedy, gdy $[(mD_1) > 0]$ & $[d(mD_1)/dt \leq 0]$,
$M_1 = 0$ wtedy i tylko wtedy, gdy $[(mD_1) = 0]$ lub $[d(mD_1)/dt > 0]$,

gdzie mD_1 jest liczbą z przedziału [0, 1], oznaczającą wartość znacznika w miejscu z etykietą D_1. Pochodna $d(mD_1)/dt \leq 0$ wskazuje, że wartość tego znacznika nie rośnie w czasie. Gdy pochodna ta jest mniejsza lub równa zero, wówczas pierwsza winda sterowana jest sygnałem (np. analogowym) – równym $D_1(t)$, który jest niezerowy przez czas nie dłuższy, niż $T_{1,0}$ (patrz rys. 3). Jednak czas $T_{1,0}$ wynika ze strategii diagnostyki – jest z góry ustalony i nie ma nic wspólnego z czasem zmniejszania się znacznika mD_1 od wartości 1 do 0 (rys. 3). Gdy nie ma awarii, to czas, po którym wystąpi $T_{1,0}$ = 1 jest na tyle długi, że miejsce A_1 nie zdąży się wypełnić znacznikiem (tranzycja $T_{1,0}$ = 1 nie odpali) – winda 1 pokona drogę z poziomu I do poziomu 0 w dopuszczalnym czasie. Może się jednak zdarzyć, że $(mD_1) > 0$, czyli $D_1(t)$>0 - winda zjeżdża, lecz trwa to zbyt długo. Wtedy wystąpi $T_{1,0}$ = 1 zanim pojawi się $(mD_1) = 0$; tranzycja $T_{1,0}$ = 1 natychmiast odpali i miejsce A_1 zapełni się, wskazując awarię. W takim przypadku układ sterowania zostanie zatrzymany, a stan rozmytej sieci Petriego dość dobrze odzwierciedli przyczynę awarii.

Zauważmy ponadto, że w ciągu czasu T, wartość rozmytego znacznika (mD_1) liniowo rośnie w czasie, co wynika z ogólnej zasady działania FPN [7].



Rys. 3. (a) - Przebieg wartość znacznika w miejscu z etykietą D_1, (b) - Przykładowy przebieg wartości sterowania D_1 w czasie, (c) – Wartość znacznika M_1 jako funkcji czasu, (d) i (e) – wyjaśnienie detekcji awarii
Fig. 3. (a) – The value of the marker in the place labeled by D_1, (b) – Examples of the control signal D_1 as a function of time, (c) – The value of the marker M_1 as a function of time, (d) and (e) – explanation of fault detection

Następne wartości znaczników M_i definiuje się analogicznie (rys. 2 i 3):

$M_2 = 1$ wtedy i tylko wtedy, gdy $[(mU_2) > 0]$ & $[d(mU_2)/dt \leq 0]$,
$M_2 = 0$ w przeciwnym razie,

$M_3 = 1$ wtedy i tylko wtedy, gdy $[(mU_1) > 0]$ & $[d(mU_1)/dt \leq 0]$,
$M_3 = 0$ w przeciwnym razie,

$M_4 = 1$ wtedy i tylko wtedy, gdy $[(mD_2) > 0]$ & $[d(mD_2)/dt \leq 0]$,
$M_4 = 0$ w przeciwnym razie.

3. Automatyzacja procesu projektowania FPN

W celu automatyzacji procesu syntezy rozmytej sieci Petriego opracowano specjalny program, napisany w języku C++, o nazwie *fpngen*. Program ten na podstawie specyfikacji struktury rozmytej sieci Petriego zawartej w pliku tekstowym o specyficznym formacie, generuje kod (tzw. wirtualny komponent – *Intellectual Property Core*) w języku opisu sprzętu Verilog HDL jednoznacz-

nie opisujący daną rozmytą sieć Petriego. Odwzorowanie struktury zadanej sieci Petriego w odpowiedni kod w języku Verilog program *fpngen* wykonuje według kilku opracowanych metod:

- z zastosowaniem synchronicznych lub asynchronicznych modułów miejsca,
- z wykorzystaniem synchronicznych lub asynchronicznych przerzutników RS,
- z użyciem rozmytych, synchronicznych przerzutników JK,
- z wykorzystaniem opisu behawioralnego w języku Verilog, specyfikującego działanie rozmytej sieci Petriego wynikające wprost z definicji.

Struktura rozmytej sieci Petriego zawarta w pliku tekstowym, stanowiącym dane wejściowe dla programu fpngen, opisywana jest z punktu widzenia każdej tranzycji występującej w sieci. Format opisu jest następujący:

lista_miejsc_ze_znakowaniem_początkowym
lista_miejsc_we : lista_miesjc_wy nazwa_tranzycji

Listę miejsc sieci stanowią numery poszczególnych miejsc (będące liczbami nieujemnymi) oddzielone przecinkiem, dlatego też wszystkie miejsca występujące w sieci należy wcześniej odpowiednio ponumerować. Listę miejsc od nazwy tranzycji przedziela jeden lub więcej znaków spacji lub tabulacji. Etykiety tranzycji mogą zawierać dowolne znaki ze zbioru liter i cyfr z wyjątkiem słowa TRUE, które ma specjalne znaczenie identyfikujące bezwarunkową tranzycję w sieci.

Program *fpngen* został również wyposażony w prosty analizator syntaktyczny sprawdzający składnię i spójność danych wejściowych zawartych w pliku tekstowym.

Odwzorowanie miejsc sieci z rys. 2 dla programu *fpngen*:

D_1→P1, W_1→P2, U_1→P3, S_1→P4, U_2→P5, W_2→P6, D_2→P7, S_2→P8, M_1→P9, A_1→P10, M_2→P11, A_2→P12, M_3→P13, A_3→P14, M_4→P15, A_4→P16.

Specyfikacja rozmytej sieci Petriego dla programu *fpngen*:

```
4,8
4,8:1,5     T
1:2         L1
2:3,11      T1
3:4         ~L1
5:6         L2
6:7,15      T2
7:8         ~L2
9:10        T_I_0
11:12       T_I_II
13:14       T_0_I
15:16       T_II_I
```

Niżej podajemy fragment kodu wygenerowanego przez program dla metody syntezy opartej na synchronicznym module miejsca:

```
module fpn(mp1,mp2,mp3,mp4,mp5,mp6,mp7,mp8,mp9,mp10,
mp11,mp12,mp13,mp14,mp15,mp16,M1,M2,M3,M4,M5,M6,M7,
M8,M9,M10,M11,M12,M13,M14,M15,M16,T,L1,T1,L2,T2,
T_I_0,T_I_II,T_0_I,T_II_I,tin_p9,tout_p10,tout_p12,
tin_p13,tout_p14,tout_p16, Ein_p9,Eout_p10,Eout_p12,
Ein_p13,Eout_p14,Eout_p16,
clock,set,reset);
//...
parameter TRUE=4'b1111; supply1 High; pl P1(.tin(T),
.tout(L1),.mp(mp1),.Ein(M4&M8&~M1&~M5),.Eout(M2),
.M(M1),.clk(clock),.set(High),.reset(reset));
pl P2(.tin(L1),.tout(T1),.mp(mp2),.Ein(M1),
.Eout(~(M2&~M3&~M11)),.M(M2),.clk(clock),.set(High),
.reset(reset));
//...
pl P16(.tin(T_II_I),.tout(tout_p16),.mp(mp16),.Ein(M15),
.Eout(Eout_p16),.M(M16),.clk(clock),.set(High),
.reset(reset));
endmodule
```

Uzyskany w wyniku działania programu wirtualny komponent, opisujący rozmytą sieć Petriego, może być następnie bezpośrednio zaimportowany do dowolnego systemu wspomagającego projektowanie, przeznaczonego dla układów programowalnych i tam zaimplementowany w wybranym układzie FPGA lub CPLD.

4. Podsumowanie

Na przykładzie pokazano, że rozmyte sieci Petriego mogą być stosowane do modelowania algorytmów sterowania, jak również diagnostyki. Dzięki takim modelom można dokonać syntezy układów sprzętowych, jako sterujących i diagnostycznych. Projektowane układy mogą przetwarzać zarówno sygnały analogowe, jak i binarne. Pokazano, że wartości rozmytych znaczników sieci Petriego można wykorzystać do syntezy sygnałów sterujących. Rozmyta sieć opisująca algorytm sterowania, daje się łatwo rozbudować o funkcje diagnostyki.

Proces projektowania układów sprzętowych dla FPN daje się znacznie zautomatyzować. Opracowany program na podstawie specyfikacji struktury rozmytej sieci Petriego, generuje wirtualny komponent w języku opisu sprzętu, który jednoznacznie opisuje daną sieć. Odwzorowanie struktury sieci w odpowiedni kod w języku Verilog wykonuje się automatycznie według kilku wariantów projektowych.

5. Literatura

[1] Chang N., Kwon W. H., Park J.: Hardware implementation of real-time Petri-net-based controllers, Control Eng. Pactice, vol. 6, pp. 889-895, 1998.

[2] David R., Alla H.: Petri Nets and Grafcet: Tools for Modelling Discrete Event Systems. London: Prentice Hall, 1992.

[3] Dec G., Hajduk Z., Zakorczmenny D.: Model do sterowania linią produkcyjną procesu kucia na gorąco, PAK, Nr 9, s.25-28, 2004.

[4] Fernandes J.M., Adamski M., Proenca A.J.: VHDL Generation from Hier-archical Petri Net Specifications of Parallel Controllers, IEE Proc.-Comput. Digit. Tech., vol. 144, no. 2, pp. 127-137, 1997.

[5] Ferrarini L., Piroddi L.: Modular design and implementation of a logic control system for a batch process, Computers & Chemical Engineering, vol. 27, pp. 983-996, 2003.

[6] Gniewek L., Kluska J.: Family of Fuzzy J-K Flip-Flops Based on Bounded Product, Bounded Sum and Complementation, IEEE Trans. Syst., Man, Cybern., Part B, vol. 28, pp. 861-868, Dec. 1998.

[7] Gniewek L., Kluska J.: Hardware implementation of Fuzzy Petri Net as a controller, IEEE Trans. Syst., Man, Cybern., Part B, vol. 34, pp. 1315-1324, June 2004.

[8] Kluska J., Gniewek L.: A New Method of Fuzzy Petri Net Synthesis and its Application for Control Systems Design, in Fuzzy Control, Advances in Soft Computing, R. Hampel, M. Wagenknecht, and N. Chaker, Eds. Hei-delberg: Springer-Verlag, 2000, pp. 222-227.

[9] Kluska J., Gniewek L: Fuzzy Petri nets as control systems, Pomiary, Automatyka, Kontrola, Vol. 1, pp.7 - 10, 2005.

[10] Kluska J., Hajduk Z.: Digital implementation of fuzzy Petri net based on asynchronous fuzzy RS flip-flop. Proc. of 7th Int. Conf. Artificial Intelli-gence and Soft Computing ICAISC, Zakopane, pp. 314-319, 2004.

[11] Kluska J., Hajduk Z.: Hardware implementation of a fuzzy Petri net based on VLSI digital circuits. Proc. of 3rd EUSFLAT Conf. Zittau, pp. 789-793, 2003.

[12] Misiurewicz P.: Zagadnienia projektowania cyfrowych układów sterowania binarnego, VIII Kraj. Konf. Automatyki, tom 1, s. 664-670, Szczecin, 1980.

[13] Park E., Tilbury D.M., Khargonekar P.P.: A Modeling and Analysis Methodology for Modular Logic Controllers of Machining Systems Using Petri Net Formalism, IEEE Trans. Syst., Man, Cybern., Part C, vol. 31, no. 2, pp. 168-188, May 2001.

[14] Peterson J.L.: Petri Net Theory and the Modelling of Systems, Prentice Hall, Englewood Cliffs, NJ, 1981.

Artykuł recenzowany