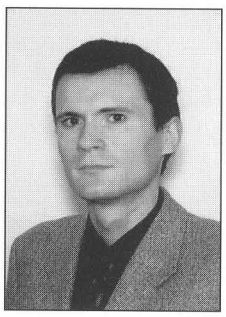
Grzegorz DEC, Zbigniew HAJDUK
POLITECHNIKA RZESZOWSKA, KATEDRA INFORMATYKI I AUTOMATYKI

Sprzętowa implementacja rozmytego systemu ekspertowego

mgr inż. Grzegorz DEC

Studia ukończył na Wydziale Elektrycznym Politechniki Rzeszowskiej w roku 1998. Obecnie pracuje jako asystent w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Zainteresowania zawodowe obejmują zastosowanie układów scalonych w procesach sterowania logicznego.

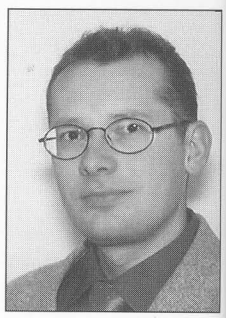
e-mail: gdec@prz-rzeszow.pl



mgr inż. Zbigniew HAJDUK

Studia ukończył na Wydziale Elektrycznym Politechniki Rzeszowskiej w roku 1998. Obecnie pracuje jako asystent w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Interesuje się układami FPGA, systemami mikroprocesorowymi, logiką rozmyta i sieciami Petriego.

e-mail: zhajduk@prz-rzeszow.pl



Streszczenie

W pracy zaproponowano metodę syntezy układu sterującego, specyfikowanego za pomocą rozmytego systemu ekspertowego, zawierającego reguły działania. Przedstawiono sposób modelowania i badania właściwości SE przy użyciu rozmytej sieci Petriego. Pokazano praktyczny przykład zastosowania metody do syntezy sterownika, wykonanego w układzie FPGA.

Abstract

This paper presents a synthesis method of a control system, that is modeled by a production-rule-based expert system. As a tool for analysis properties of the expert system a fuzzy Petri net is proposed. We show practical application of the described method to develop a parallel controller, that is implemented in a FPGA chip.

Słowa kluczowe: systemy ekspertowe, sterowanie rozmyte, rozmyte sieci Petriego, FPGA.

Keywords: expert systrems, fuzzy control, fuzzy Petri nets, FPGA.

1. Wstęp

Systemy ekspertowe służą do rozwiązywania problemów wymagających profesjonalnej ekspertyzy. Znajdują szerokie zastosowanie w różnych dziedzinach, takich jak diagnostyka, planowanie czy sterowanie. Podstawowymi elementami struktury systemów ekspertowych są baza wiedzy i maszyna wnioskująca. Tworzenie systemu ekspertowego opartego na bazie wiedzy polega na pozyskaniu wiedzy eksperta, który znajduje rozwiązanie na podstawie informacji o problemie i swoim doświadczeniu. Podstawą efektywnego działania systemu ekspertowego jest obszerna baza wiedzy, wyposażona w informacje dobrej jakości [1]. W związku z tym istnieje potrzeba opracowania sposobów przechowywania wiedzy. Zaproponowano kilka modeli reprezentacji wiedzy, takich jak rachunek zdań, stwierdzenia, reguły, ramy [2, 3] oraz sieci Petriego. Sieci Petriego dzięki szerokim możliwościom modelowania i analizy systemów znajdują zastosowanie dziedzinach takich jak reprezentacja wiedzy [4], wnioskowanie [5], pozyskiwanie wiedzy [6] oraz są dobrym mechanizmem modelowania rozmytych systemów ekspertowych opartych na regułach [5, 7].

2. System ekspertowy

Rozpatrzmy system ekspertowy złożony z bazy wiedzy opisanej za pomocą zbioru reguł

$$L=\{l_1, l_2, ..., l_s\} \qquad (1)$$

i zbioru faktów

$$F=\{f_1, f_2, ..f_k\}. \qquad (2)$$

Każda reguła l_i ma postać:

$$l_i : \text{IF } \textit{przesłanka} \text{ THEN } \textit{konkluzja}, \qquad (3)$$

gdzie *przesłanka* jest wyrażeniem złożonym ze zdań logicznych połączonych spójnikiem AND, tj.: *przesłanka:* s^i_1 AND ... AND s^i_n; podobnie konkluzja: d^i_1 AND ... AND d^i_m; s^i_n, d^i_m są zdaniami logicznymi i $n, m \in \{1, 2, ...\}$. Każde zdanie logiczne s^i_j może być prawdziwe w pewnym stopniu $M(s^i_j)\in\langle 0; 1\rangle$, podobnie jak zdanie d^i_j. Zdania są opisami pewnych czynności lub stanów, np. *składniki są mieszane* czy *stanowisko jest gotowe.*

Zdefiniujmy zbiory: $S_i = \{s^i_1, ..., s^i_n\}$, $D_i = \{d^i_1, ..., d^i_m\}$,

$$S=\bigcup_{i=1}^{s} S_i,\ D=\bigcup_{i=1}^{s} D_i,\ G_i = S_i \setminus D,\ G=\bigcup_{i=1}^{s} G_i,\ Q_i = S_i \setminus G_i = \left\{q^i_1,\ldots,q^i_r\right\}.$$

Znaczenia zbiorów są następujące: S_i - stwierdzenia przesłanki reguły i; S - stwierdzenia występujące w przesłankach wszystkich reguł; D_i - stwierdzenia konkluzji reguły i; D - stwierdzenia występujące w konkluzjach wszystkich reguł; G_i - warunki związane z regułą i (takie stwierdzenia z przesłanki, które nie występują w żadnej konkluzji); G - warunki występujące w przesłankach wszystkich reguł; Q_i - stwierdzenia przesłanki reguły i, które nie są warunkami.

Wprowadzimy pewne ograniczenia na zbiór reguł, służące uproszczeniu metody syntezy układu sterującego specyfikowanego za pomocą SE. Zbiór L nie powinien zawierać reguł zapętlonych (np.: IF s_1 THEN s_1) i sprzecznych (np. l_1: IF s_1 THEN d_1; l_2: IF s_1 THEN d_2). Każde stwierdzenie występujące w konkluzji reguły musi wystąpić w przesłance przynajmniej jednej innej reguły. Możemy zapisać:

$$\forall(i=1,2,\ldots,s): D_i \cap S_i = \varnothing,$$
$$\bigcap_{i=1}^{s} D_i = \varnothing,\ \bigcap_{i=1}^{s} Q_i = \varnothing,\ D \subseteq S. \qquad (4)$$

Reguła l_i jest aktywna (może być użyta w procesie wnioskowania), od chwili gdy stopień prawdy stwierdzeń ze zbioru Q_i przesłanki jest równy 1 i stopień prawdy stwierdzeń konkluzji jest równy 0, do chwili gdy stopień prawdy stwierdzeń ze zbioru Q_i przesłanki jest równy 0 i stopień prawdy stwierdzeń konkluzji jest równy 1. Przyjmijmy następujący sposób wnioskowania: jeżeli reguła l_i jest aktywna i stopień prawdy warunku ze zbioru G_i jest równy ϑ_i, to zmianie ulegają stopnie prawdy stwierdzeń konkluzji D_i jak i stwierdzeń przesłanki Q_i. Nowy stopień prawdy każdego stwierdzenia q^i_j przesłanki jest równy $M'(q^i_j)=M'(q^i_j)\wedge(1-\vartheta_i)$. Nowy stopień prawdy każdego stwierdzenia d^i_j konkluzji jest równy $M'(d^i_j)= M(d^i_j) \vee \vartheta_i$.

Taka procedura wnioskowania pozwala na proste reprezentowanie wiedzy o stanie procesów, które składają się z sekwencji działań.

3. Reprezentacja wiedzy SE

Rozmyta sieć Petriego FPN może modelować rozmyte reguły systemu ekspertowego SE oraz wnioskowanie. W tym artykule przedstawiona zostanie propozycja modelowania bazy wiedzy SE i sposobu wnioskowania za pomocą FPN, opisanej w pracy [8]. W artykułach [9, 10] pokazano zastosowanie tej sieci w zadaniach sterowania. Projektant specyfikuje model sterownika za pomocą formalizmu sieci Petriego, po czym dokonuje przekształcenia modelu na kod w języku opisu sprzętu, wykorzystując techniki projek-

towania strukturalnego: miejsca i tranzycje FPN zastępowane są odpowiednimi blokami funkcjonalnymi [11]. Autorzy uważają, że dla projektanta prostszym sposobem opisu zachowania sterownika może okazać się opis regułowy. Jednakże ze względu na duże możliwości sieci Petriego w dziedzinie analizy systemów, proponowana jest ona jako dodatkowy model układu sterującego. Ma to szczególne znaczenie w przypadku badania układów sterowania, gdzie dopuszczalna jest współbieżna praca procesów.

Używana FPN jest uogólnieniem klasycznej sieci binarnej [12] na logikę rozmytą. FPN jest siecią bez pętli i bezkonfliktową, składa się z miejsc, tranzycji i łuków. Z każdym miejscem i tranzycją związane są odpowiednio stwierdzenie i warunek. Łuki określone są za pomocą relacji incydencji R. Znacznik może przybierać wartości z przedziału ⟨0; 1⟩.

W literaturze spotyka się odwzorowanie stwierdzeń SE na miejsca, reguł na tranzycje i faktów na znakowanie FPN [5, 7]. Postępowanie takie będzie podstawą prezentowanego przekształcenia SE na FPN. Każdej regule $l_i \in L$ odpowiada tranzycja t_i, której przyporządkowany jest warunek ze zbioru G_i lub stała PRAWDA, jeżeli jest $G_i=\varnothing$. Każdemu stwierdzeniu $d_i \in D$ odpowiada miejsce p_i z przypisanym stwierdzeniem d_i. Stopnie prawdy stwierdzeń modelowane za pomocą znakowania. Do relacji R FPN należą następujące pary (łuki sieci):

- para (p_i, t_j), jeżeli stwierdzenie przyporządkowane miejscu p_i należy do zbioru stwierdzeń Q_j występujących w przesłance reguły przyporządkowanej tranzycji t_j,
- para (t_j, p_i), jeżeli stwierdzenie przyporządkowane miejscu p_i należy do zbioru stwierdzeń D_j występujących w konkluzji reguły przyporządkowanej tranzycji t_j.

Aktywnej regule SE odpowiada aktywna tranzycja FPN. Znakowanie następne FPN wyznacza reguła: jeżeli tranzycja t_i jest aktywna i stopień prawdy warunku przypisanego tej tranzycji jest równy ϑ_i, to nowe znakowanie M' każdego miejsca wejściowego p_j^{i-} tranzycji jest równe $M'(p_j^{i-})=M'(p_j^{i-}) \wedge (1-\vartheta_i)$. Nowe znakowanie M' każdego miejsca wyjściowego p_j^{i+} jest równe $M'(p_j^{i+})= M(p_j^{i+}) \vee \vartheta_i$. Sposób wyznaczania znakowania następnego FPN (szczegóły w [8]) odpowiada wnioskowaniu SE.

Wykorzystując przedstawioną metodę, można przekształcić regułową specyfikację SE na FPN, a następnie za pomocą istniejącego oprogramowania [13] dokonać analizy systemu pod kątem takich właściwości, jak np. istnienie współbieżności, osiągalność, żywotność, bezkonfliktowość.

4. Implementacja SE w języku opisu sprzętu

Posiadając specyfikację układu sterującego, należy zdecydować się na realizację programową lub sprzętową. Pokażemy sposób realizacji sprzętowej, dający w wyniku kod w języku Verilog, który może być zaimplementowany w szerokiej klasie układów scalonych, jak CPLD, FPGA czy ASIC. Wybór języka powodowany jest dostępnością narzędzi i przyzwyczajeniami autorów.

Załóżmy, że w celu zamodelowania pewnego procesu określono system ekspertowy z bazą wiedzy L i zbiorem faktów F. Zbiór F definiuje stan początkowy procesu. Reguły zbioru L spełniają warunki wyrażone przez (4). Rozpatrzmy regułę l_1 postaci:

$$l_1: \text{IF } q_2 \text{ AND } g_4 \text{ THEN } d_1. \qquad (5)$$

Przyjmijmy oznaczenia zmiennych: M2 - przyjmuje wartość 1 od chwili kiedy stwierdzenie q_2 jest prawdziwe w stopniu 1, do chwili, kiedy stwierdzenie q_2 jest prawdziwe w stopniu 0; M1 - identyczne znaczenie jak M2, ale dla stwierdzenia d_1; mp2, mp1 - przechowują stopnie prawdy stwierdzeń odpowiednio q_1 i d_1; g_4 - stopień prawdy warunku g_4. Przykładowa reguła może być zapisana w języku Verilog za pomocą instrukcji:

```
parameter TRUE=4'b1111;
parameter FALSE=4'b0000;
input [3:0] g4;
output[3:0] mp1,mp2;   reg[3:0] mp1,mp2;
output M1,M2;          reg M1,M2;
allways@(posedge clock) begin
    if(M2&~M1) begin     // reguła aktywna?
      if(g4>mp1) begin // nowe stopnie
        mp1=g4; mp2=~g4; // prawdy stwierdzeń
      end
    end
    case(mp1)TRUE:M1=1;FALSE:M1=0;endcase
    case(mp2)TRUE:M2=1;FALSE:M2=0;endcase
  end
```

Odwzorowanie reguł na kod w języku Verilog może być zautomatyzowane. Algorytm tego przekształcenia jest następujący:

1. Niech $T=\{t_1, t_2, ..., t_{|G|}\}$ będzie zbiorem zmiennych wejściowych. Zdefiniujmy bijekcyjną funkcję α: $G \rightarrow T$.
2. Niech $M=\{M_1, M_2, ..., M_{|D|}\}$, $MP=\{mp_1, mp_2, ..., mp_{|D|}\}$ będą zbiorami zmiennych M_i i mp_i o znaczeniu wyjaśnionym poprzednio. Zdefiniujmy bijekcyjne funkcje β: $D \rightarrow M$ i β':$D \rightarrow MP$.
3. Dla każdej reguły $l_i \in L$:

3.1. Określmy zbiór zmiennych wejściowych $T_i=\{t: \forall g^i_i \in G_i: t=\alpha(g^i_i)\}$. Zbiór ten powinien być jednoelementowy. Jeżeli jest $G_i=\varnothing$ to $T_i=\{\text{TRUE}\}$.

3.2. Określmy zbiór zmiennych mp^- odpowiadających stwierdzeniom zbioru Q_i: $MP^-_i=\{mp^-: \forall q^i_i \in Q_i: mp^-=\beta'(q^i_i)\}$.

3.3. Określmy zbiór zmiennych mp^+ odpowiadających stwierdzeniom zbioru D_i: $MP^+_i=\{mp^+: \forall d^i_i \in D_i: mp^+=\beta'(d^i_i)\}$.

3.4. Należy zakodować następujące instrukcje:

3.5. Jeżeli jest $\beta(q^i_1) \wedge ... \wedge \beta(q^i_{|Qi|}) \wedge \beta(d^i_1) \wedge ... \wedge \beta(d^i_{|Di|})=1$ to {

3.6. Jeżeli jest $t > mp^+_1$ to {

$\forall mp^+ \in MP^+_i$: $mp^+=t$

$\forall mp^- \in MP^-_i$: $mp^-=$ not t

}

}

4. Dla każdego mp_i ze zbioru MP:

4.1. Jeżeli $mp_i=1$ to $\beta(\beta'^{-1}(mp_i))=1$, w przeciwnym przypadku $mp_i=\beta(\beta'^{-1}(mp_i))=0$

Zaproponowany algorytm został zrealizowany w języku C. Program fpngen przekształca plik wejściowy zawierający opis reguł SE na model sterownika w języku Verilog. Format pliku wejściowego jest następujący:

```
plik wejściowy ::= fakty lista_reguł
fakty ::= element {, element}
lista_reguł ::= reguła { reguła}
reguła ::= przesłanka : konkluzja warunek
przesłanka ::= element {, element}
konkluzja ::= element {, element}
```

Znaczniki element są liczbami całkowitymi, natomiast znacznik warunek jest dowolnym ciągiem znaków. Dla reguły l_i elementy przesłanki tworzymy z indeksów elementów zbioru Q_i, elementy konkluzji z indeksów elementów zbioru D_i, a warunkiem jest element zbioru G_i. Weźmy jako przykład regułę (5). Jej zapis w przedstawionym formacie wygląda następująco:

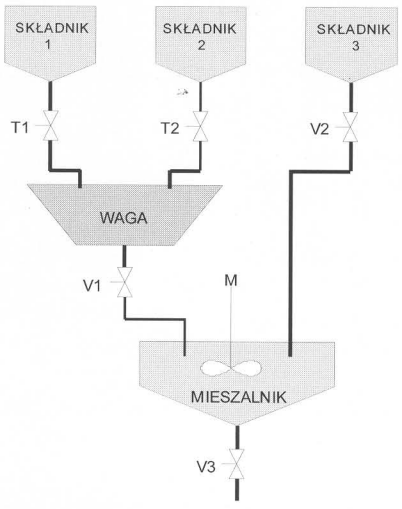
```
2:1  g4
```

Narzędzie fpngen dokonuje sprawdzenia warunków poprawności SE określonych przez (4), jednak do pełnej analizy właściwości systemu wskazane jest użycie dedykowanych pakietów oprogramowania, jak np. [13].

5. Przykład

Urządzenie do produkcji betonu [12] (rys. 1), składa się z transporterów kruszywa T1 i cementu T2, wagi do odważania składników, zbiornika z wodą i betoniarki. Sekwencja czynność potrzebnych w procesie produkcji jest następująca (w nawiasach podano symbole używane w dalszej części):

- ważenie kruszywa (e_1),
- opróżnianie wagi z kruszywa (e_2),
- ważenie cementu (e_3),
- opróżnianie wagi z cementu (e_4),
- odmierzanie wody (e_5),
- mieszanie składników (e_6),
- opróżnianie betoniarki (e_7).



Rys. 1. Schemat modelowanego urządzenia
Fig. 1. Schema of the modelled device

Pewne czynności mogą być wykonywane równolegle, np. e_1 i e_6, jednakże należy wprowadzić dodatkowe stwierdzenia, opisujące oczekiwanie na zwolnienie zajętych zasobów. Po zakończeniu czynności e_1 czynność e_2 może być wykonana tylko wtedy, gdy betoniarka jest opróżniona (e_8). Do tego czasu należy zaczekać na opróżnienie betoniarki (e_9). Po odmierzeniu wody (e_5) należy zaczekać na cement i kruszywo w betoniarce (e_{10}). Podobnie, przed włączeniem mieszalnika, a po wykonaniu czynności e_4, musimy zaczekać na nalanie wody (e_{11}). Posiadając zidentyfikowane stwierdzenia opisujące stany przyjmowane przez proces, można podać reguły sterowania:

l_1: IF e_1 AND żądana waga kruszywa (g_1) THEN e_9
l_2: IF e_9 AND e_8 THEN e_2
l_3: IF e_2 AND waga opróżniona z kruszywa (g_2) THEN e_3
l_4: IF e_3 AND żądana waga cementu (g_3) THEN e_4
l_5: IF e_4 AND waga opróżn. z cementu (g_4) THEN e_{11} AND e_1
l_6: IF e_{11} AND e_{10} THEN e_6
l_7: IF e_6 AND składniki wymieszana (g_5) THEN e_7
l_8: IF e_7 AND betoniarka pusta (g_6) THEN e_8 AND e_5
l_9: IF e_5 AND woda nalana (g_7) THEN e_{10}

W stanie początkowym betoniarka jest opróżniona, kruszywo jest ważone i woda jest odmierzana ($F=\{e_1, e_5, e_8\}$). Plik wejściowy dla programu fpngen wygląda następująco:

```
1,5,8                          \\ stan początkowy
1          : 9       g1        \\ reguła L1
9, 8       : 2       TRUE      \\ reguła L2
2          : 3       g2        \\ reguła L3
3          : 4       g3        \\ reguła L4
4          : 11, 1   g4        \\ reguła L5
11, 10     : 6       TRUE      \\ reguła L6
6          : 7       g5        \\ reguła L7
7          : 8, 5    g6        \\ reguła L8
5          : 10      g7        \\ reguła L9
```

Wynikiem pracy programu jest model sterownika w języku Verilog, którego fragmenty prezentowane są poniżej. Zakłada się, że sygnały wielowartościowe reprezentowane są jako liczby całkowite 4-bitowe. W stanie początkowym prawdziwe są stwierdzenia e_1, e_5, e_8, czemu odpowiadają wartości maksymalne zmiennych mp1, M1, mp5, M5, mp8, M8. Aktywność reguły l_2 zależy od stopni prawdy stwierdzeń e_8, e_9 i e_2 (zmienne M8, M9, M2). Jeżeli reguła ta jest aktywna, zmienna mp2, odpowiadająca stopniowi prawdy stwierdzenia e_2 z konkluzji reguły, przyjmuje wartość TRUE, a zmienne mp8 i mp9 przyjmują wartość 0.

```
module fpn(mp1,mp2,mp3,mp4,mp5,mp6,mp7,mp8,
 mp9,mp10,mp11,M1,M2,M3,M4,M5,M6,M7,M8,M9,
 M10,M11,g1,g2,g3,g4,g5,g6,clock,start,reset);
input [3:0] g1,g2,g3,g5,g6,g4;
output [3:0] mp1,mp2,mp3,mp4,mp5,
mp6,mp7,mp8,mp9,mp10,mp11;
reg [3:0] mp1,mp2,mp3,mp4,mp5,
mp6,mp7,mp8,mp9,mp10,mp11;
output M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11;
reg M1,M2,M3,M4,M5,M6,M7,M8,M9,M10,M11;
input clock,start,reset;
parameter TRUE=4'b1111; parameter FLS=4'b0000;
always @(posedge clock or negedge reset) begin
  if(~start) begin    // stan początkowy
    mp1=TRUE; M1=1; mp5=TRUE; M5=1;
    mp8=TRUE; M8=1;
  end else begin
    ...
    if(M9&M8&~M2) begin    // reguła 2
       mp9=0;  mp8=0; mp2=TRUE;
    end
    ...
    case(mp9)TRUE:M9=1; FLS:M9=0;endcase
    ...
  end
end
```

Uzyskany plik posłużył do zaprogramowania układu scalonego klasy FPGA Xillinx XC2S100, na którym praktycznie zweryfikowano poprawność proponowanej metody. Kompilacja wykonana została w środowisku Xillinx ISE ver. 5.2i. Sterownik wykonany w układzie cyfrowym kontroluje działanie fizycznego urządzenia znajdującego się w laboratorium.

6. Podsumowanie

W pracy zaproponowano metodę implementacji układu sterującego specyfikowanego za pomocą rozmytego systemu ekspertowego SE. Baza wiedzy SE może być modelowana przez rozmytą sieć Petriego, co pozwala na analizę właściwości SE. Zaproponowano i zrealizowano algorytm automatycznej konwersji regułowej specyfikacji SE na model w języku opisu sprzętu Verilog. Dokonano praktycznej weryfikacji proponowanej metody projektując system sterujący obiektem laboratoryjnym. System wykonano na układzie klasy FPGA.

7. Literatura

[1] J. J. Mulawka: Systemy ekspertowe. WNT, Warszawa, 1996.
[2] J. Korbicz, J. M. Kościelny, Z. Kowalczuk, W. Cholewa: Diagnostyka procesów. Modele. Metody sztucznej inteligencji. Zastosowania. WNT, 2002.
[3] A. Niederliński: Regułowe systemy ekspertowe. Pracownia Komputerowa Jacka Skalmierskiego, 2000.
[4] T. Murata, D. Zhang: A Predicate-Transition Net Model for Parallel Interpretation of Logic Programs. IEEE Trans. Softw. Eng., vol. 14, no. 4, pp. 481-497, 1988.
[5] A. Bugarin, S. Barro: Fuzzy Reasonig Supported by Petri Nets. IEEE Trans. On Fuzzy Systems, vol. 2, no. 2, pp. 135-150, 1994.
[6] S. M. Chen: A Knowledge Acquisition Scheme for Rule-Based Expert Systems Based on Fuzzy Petri Nets. Intl. Journal of Information Management and Engineering, vol. 1, no. 3, pp. 45-56, 1994.
[7] S. M. Chen, J. S. Ke, J. F. Chang: Knowledge Representation Using Fuzzy Petri Nets. IEEE Transactions on Knowledge and Data Engineering, vol. 2, no. 3, pp. 311-319, 1990.
[8] L. Gniewek, J. Kluska: Hardware Implementation of Fuzzy Petri Net as a Controller. IEEE Trans. on Systems, Man, and Cybernetics, Part B: Cybernetics, vol. 34, no. 3, pp. 1315-1324, 2004.
[9] G. Dec: Sprzętowa realizacja rozmytej sieci Petriego w reprogramowalnych układach scalonych. VI Krajowa Konferencja Naukowa RUC, str. 71-78, Szczecin, 2003.
[10] J. Kluska, Z. Hajduk: Hardware Implementation of a Fuzzy Petri Net Based on VLSI Digital Circuits. Proc. of 3rd European Society for Fuzzy Logic and Technology, pp. 789-793, Zittau, 2003.
[11] L. Gniewek, J. Kluska: Familly of Fuzzy J-K Flip-Flops Based on Bounded Product, Bounded Sum and Complementation. IEEE Trans. on Systems, Man. and Cybernetics, vol. 28, no. 6, pp. 861-868, 1998.
[12] P. Misiurewicz: Zagadnienia projektowania cyfrowych układów sterowania binarnego. VIII Krajowa Konferencja Automatyki, część 1, str. 664-670, 1980.
[13] Z. Suraj, M. Szpyrka: Sieci Petriego i PN-TOOLS, Wydawnictwo WSP, Rzeszów 1999.

Title: Hardware Implementation of a Fuzzy Expert System

Artykuł recenzowany