Lesław GNIEWEK, Zbigniew HAJDUK
POLITECHNIKA RZESZOWSKA, KATEDRA INFORMATYKI I AUTOMATYKI,
Al. Powstańców Warszawy 12, 35-959 Rzeszów

Sprzętowo-programowa realizacja rozmytej interpretowanej sieci Petriego

Dr inż. Lesław GNIEWEK

Absolwent Wydziału Elektrycznego Politechniki Rzeszowskiej (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 jest adiunktem w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Jego publikacje dotyczą projektowania sprzętu cyfrowego, logiki rozmytej, sieci Petriego i programowalnych sterowników logicznych.

e-mail: lgniewek@prz-rzeszow.pl

Dr inż. Zbigniew HAJDUK

Absolwent Wydziału Elektrycznego Politechniki Rzeszowskiej (specjalność aparatura elektroniczna). Od 2006 jest adiunktem w Katedrze Informatyki i Automatyki Politechniki Rzeszowskiej. Zainteresowania badawcze obejmują projektowanie aplikacji z układami programowalnymi, języki opisu sprzętu - w tym szczególnie język Verilog oraz systemy mikroprocesorowe.

e-mail: zhajduk@kia.prz.edu.pl

Streszczenie

W artykule przedstawiono algorytm sterowania mieszalnikiem, który przygotowano w formie rozmytej interpretowanej sieci Petriego. Algorytm ten zaimplementowano w specjalizowanym sterowniku programowalnym, zbudowanym w oparciu o struktury programowalne FPGA, co znacznie skróciło czas wykonywania kodu. Sterownik programowany jest zgodnie z normą IEC 61131-3 dzięki środowisku inżynierskiemu CPDev. Zaprezentowano ogólny sposób realizacji rozmytej interpretowanej sieci Petriego w języku tekstowym ST, co pozwala uzyskać przenośność programów pomiędzy sterownikami PLC różnych producentów.

Słowa kluczowe: modelowanie, sieci Petriego, układy FPGA.

Hardware-software realization of Fuzzy Interpreted Petri Net

Abstract

Fuzzy Interpreted Petri Net is a synchronized, low-level net, which can be used for formal description of control algorithms. Formal bases of the net and a transformation method to the logic circuit were presented in [1]. Software realization of the net, using Siemens Step 7 language, was described in [2]. Some practical application of the net for controls and diagnostics can be found in [3]. In this article, general realization method of Fuzzy Interpreted Petri Net in ST language was proposed. The method directly uses the transition firing and dynamic equations of the net. As a hardware, specific programmable controller, based on FPGA structures, was applied. Description of main CPU unit of the controller is shortly presented in this article. More detail of the controller can be found in [6], [7, 10]. Prototype of the controller was shown in [12] as well. FPGA programmable controller is programmed using CPDev control software [4]. Example of Fuzzy Interpreted Petri Net for control of the mixer plant is also included in this article. The net consists of 18 places *p*', 5 places *p*" and 17 transitions. Total computation time of the control algorithm, implemented in FPGA programmable controller, is very short and equal to 86μs. Such a time is almost unreachable to the typical, industrial PLCs. Another advantage of proposed realization method of Fuzzy Interpreted Petri Net is a portability of the code between PLCs of different producers, which is impossible to obtain using PLC programming languages, such as LD or FBD.

Keywords: modelling, fuzzy Petri nets, FPGA.

1. Wprowadzenie

Rozmyta interpretowana sieć Petriego jest synchronizowaną siecią niskiego poziomu, którą można wykorzystać do formalnego opisu algorytmów sterowania. Należy do klasy sieci rozmytych, w których wartości logiczne są liczbami z przedziału [0, 1], umożliwiając uwzględnienie nie tylko binarnych sygnałów procesowych, ale i wartości analogowych. W [1] przedstawiono formalne podstawy tej sieci oraz podano metodę jej transformacji na schemat układu logicznego. W kolejnej pracy [2] pokazano programową realizację wszystkich elementów potrzebnych do tego przekształcenia oraz omówiono organizację programu sterującego, który zaimplementowano w sterowniku Simatic S7. W [3] przedstawiono praktyczne zastosowanie tej sieci w układzie sterowania i diagnostyki.

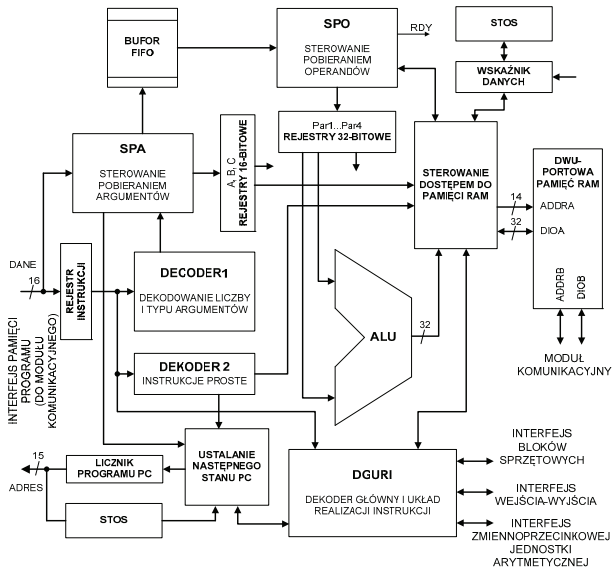
W niniejszej pracy przedstawiono algorytm sterowania mieszalnikiem w formie rozmytej interpretowanej sieci Petriego oraz omówiono jego realizację z wykorzystaniem dedykowanego sterownika PLC implementowanego w strukturach programowalnych FPGA oraz środowiska inżynierskiego CPDev [4]. Program sterujący napisano w języku tekstowym ST, zgodnym z normą IEC61131-3, i przetestowano na obiekcie laboratoryjnym.

2. Sterownik PLC implementowany w układach FPGA

W środowisku inżynierskim CPDev [4], programy napisane w jednym z dostępnych języków programowania (ST, FBD, LD) są kompilowane do postaci uniwersalnego kodu pośredniego, który po stronie sterownika wykonuje maszyna wirtualna. Oprócz programowych realizacji maszyny wirtualnej, wykorzystujących popularne mikrokontrolery AVR i ARM [5], opracowano również sprzętową implementację tej maszyny [6, 7], która jest kilkadziesiąt razy szybsza od istniejących rozwiązań programowych. Sprzętowa implementacja maszyny wirtualnej, zwana również maszyną sprzętową, jest specjalizowanym 32-bitowym mikrokontrolerem, którego lista rozkazów jest identyczna z listą instrukcji uniwersalnego kodu pośredniego generowanego przez kompilator środowiska CPDev. Maszyna sprzętowa istnieje w postaci tzw. wirtualnego komponentu (*IP core*), opisanego w języku Verilog [8, 9] i może być zaimplementowana układach programowalnych FPGA dowolnego producenta, posiadających odpowiednio dużą liczbę zasobów logicznych. Maszyna sprzętowa, wyposażona w zmiennoprzecinkową jednostkę arytmetyczną [10] oraz moduł komunikacyjny [11], zaimplementowana w układach programowalnych PFGA wraz z zewnętrznymi modułami wejść-wyjść cyfrowych oraz analogowych, tworzy sterownik programowalny PLC [12], programowany z wykorzystaniem środowiska CPDev.

Na rys. 1 przedstawiono uproszczony schemat architektury maszyny sprzętowej [6, 7]. Istotnym elementem architektury jest podukład odpowiedzialny za pobranie argumentów z pamięci programu oraz operandów z pamięci danych. Podukład ten składa się z bloku wyznaczania liczby i typu argumentów do pobrania (DECODER 1), bloku sterowania pobieraniem argumentów z pamięci programu (SPA), sprzętowego bufora FIFO, bloku sterowania pobieraniem operandów z pamięci danych (SPO) oraz zespołu 16 i 32 bitowych rejestrów. Bloki SPA i SPO pracują współbieżnie: z pamięci programu pobierane są kolejne argumenty (wskaźniki do operandów), które poprzez bufor FIFO trafiają do bloku sterowania pobieraniem operandów SPO. Wartości operandów umieszczane są w zespole 32-bitowych rejestrów. Zakończenie pobierania argumentów i operandów sygnalizowane jest ustawieniem sygnału gotowości (RDY) i dopiero wówczas rozpoczyna się kolejna faza wykonania danego rozkazu: urucha-

miana jest jednostka arytmetyczno-logiczna ALU. Jeżeli dany rozkaz należy do grupy instrukcji, które mogą być bezpośrednio zrealizowane przez układy kombinacyjne tej jednostki, wówczas już w kolejnym takcie zegara generowany jest sygnał zapisu wyniku do pamięci danych oraz jednocześnie, w tym samym takcie zegara, pobierany jest z pamięci programu kod następnego rozkazu. Rozkazy bardziej złożone wykonywane są przez układy sekwencyjne głównego dekodera i układu realizacji instrukcji DGURI. W tym przypadku faza zapisu wyniku przetwarzania danych również przebiega współbieżnie z fazą pobrania kolejnego rozkazu.



Rys. 1. Uproszczony schemat architektury maszyny sprzętowej
Fig. 1. Simplified block diagram of the hardware machine

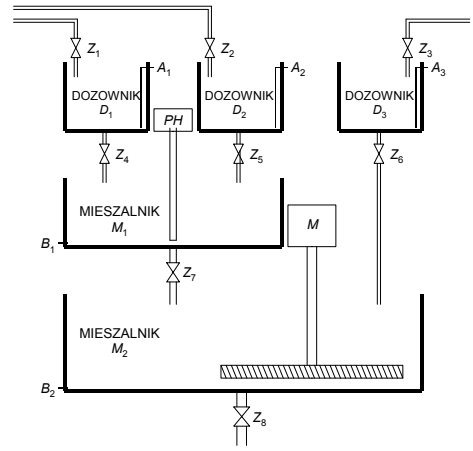
Jednostka arytmetyczno-logiczna maszyny sprzętowej może bezpośrednio przetwarzać dane zarówno 8, 16, jak i 32 bitowe. Realizuje operacje arytmetyczne, logiczne, porównania, rotacji i przesunięć bitowych. Większość operacji realizowana jest przez odpowiednie układy kombinacyjne, dzięki czemu wynik dostępny jest już w kolejnym takcie zegara.

Maszyna sprzętowa stanowi specyficzną jednostkę centralną sterownika PLC, programowanego za pomocą środowiska CPDev. Załadowanie kodu wykonywalnego do pamięci programu sterownika zapewnia moduł komunikacyjny [11]. Umożliwia on również m.in. monitorowanie pracy sterownika, w tym pozwala na pełny dostęp (zapis i odczyt) do pamięci programu oraz pamięci operacyjnej maszyny sprzętowej, odczyt rejestrów specjalnych maszyny, a także pewne funkcje pomocne podczas uruchamiania oprogramowania dla maszyny sprzętowej.

Część sprzętowa sterownika programowalnego ma konstrukcję modułową [12]. Jej zasadniczym elementem jest tzw. moduł główny z układem FPGA do którego za pomocą wielostykowych złączy dołączane są dedykowane moduły wejścia-wyjścia. W module głównym, zależnie od wersji, wykorzystywane są układy FPGA Xilinx z rodziny Spartan-3A oraz Spartan 6. Sterownik wyposażony jest również w dedykowane moduły wejść i wyjść analogowych oraz cyfrowych.

3. Obiekt sterowania

Obiektem sterowania jest laboratoryjny układ mieszalnika, zbudowany z tablicy, na której zainstalowano pięć zbiorników wyposażonych w binarne i analogowe czujniki poziomu oraz zawory i pompy dozujące (rys. 2).



Rys. 2. Układ mieszalnika z 5 zbiornikami
Fig. 2. The mixer plant with 5 containers

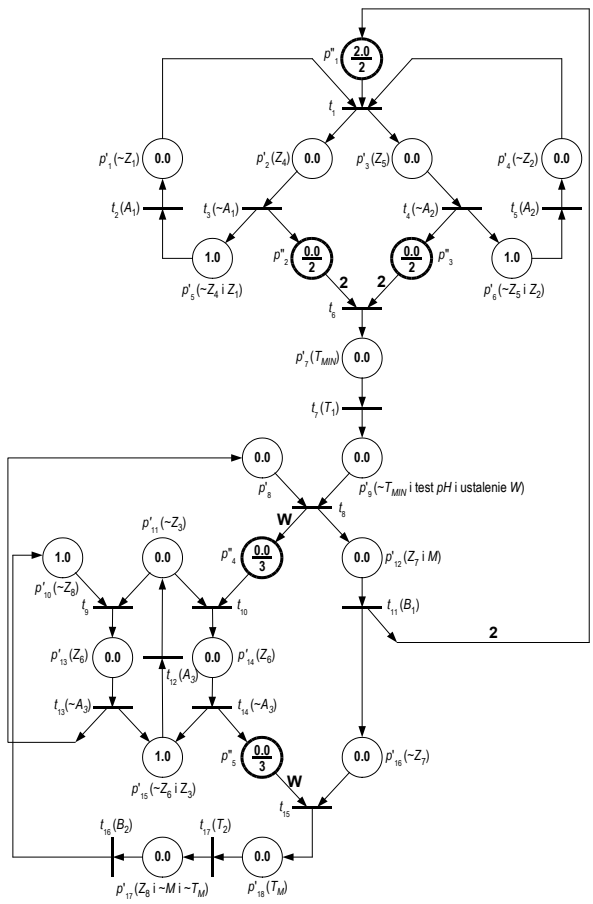
Przeanalizowano przykładowy algorytm działania układu mieszalnika, przyjmując oznaczenia czujników i elementów wykonawczych zgodnie z rys. 2. Trzy zbiorniki umieszczone w górnej części tablicy pełnią rolę dozowników D_i, $i = 1, 2, 3$, odmierzających odpowiednie porcje składników płynnych. Analogowe czujniki poziomu podają sygnały A_i, znormalizowane do przedziału [0, 1], określające stopnie ich napełnienia. Dozowniki są napełniane niezależnie od siebie poprzez otwarcie zaworów Z_i, gdy tylko zostaną opróżnione ($A_i = 0$). Gdy dozowniki D_1 i D_2 są pełne i mieszalnik M_1 jest opróżniony, rozpoczyna się proces jego napełniania dwoma porcjami każdego ze składników chemicznych. Pompy dozujące Z_4 i Z_5 są zawsze jednocześnie załączane, rozpoczynając proces przelewania kolejnych porcji składników. Kiedy po dwie porcje każdego z dwóch składników są już w mieszalniku M_1, następuje uruchomienie elementu czasowego T_{MIN}, odliczającego minimalny czas T_1 potrzebny na ustanie reakcji chemicznych zachodzących wskutek połączenia tych składników. Następnie przeprowadzany jest odczyt z pehametru i na jego podstawie ustalana jest liczba porcji rozpuszczalnika, jaką należy podać z dozownika D_3. Proces rozcieńczania otrzymanej w mieszalniku M_1 substancji chemicznej przeprowadza się w mieszalniku M_2. Wstępnie do tego mieszalnika nalewa się jedną porcję rozpuszczalnika, a następnie jednocześnie wlewa się substancję z mieszalnika M_1 i określoną liczbę dodatkowych porcji rozpuszczalnika, włączając jednocześnie mieszadło M. Gdy mieszalnik M_1 jest opróżniony, co wskazuje binarny czujnik poziomu B_1, wyłączana zostaje pompa dozująca Z_7 i jednocześnie rozpoczyna się ponowne napełnianie tego mieszalnika. Dzięki równoczesnemu wykonywaniu tych czynności zwiększa się wydajność urządzenia. Gdy do mieszalnika M_2 zostanie przelana cała substancja z mieszalnika M_1 i odpowiednia liczba porcji rozpuszczalnika, wówczas uruchamiany jest element czasowy T_M, odmierzający czas, przez jaki te substancje mają być dodatkowo mieszane. Po jego upływie roztwór jest już gotowy, a zatem wyłączane jest mieszadło M i opróżniany mieszalnik M_2 poprzez załączenie pompy Z_8. Po zakończeniu tej operacji pompa Z_8 jest wyłączana i jedna porcja rozpuszczalnika z dozownika D_3 jest wlewana do mieszalnika M_2, aby jak najszybciej mógł rozpocząć się kolejny proces opróżniania mieszalnika M_1.

Algorytm opisujący działanie mieszalnika przedstawiono na rys. 3 w postaci rozmytej interpretowanej sieci Petriego. Miejscom sieci typu p' przyporządkowano sygnały sterujące, które oznaczono następująco:

Z_n – otwarcie zaworu lub załączenie pompy dozującej Z_n, $n = 1, 2, 3, 4, 5, 6, 7, 8$ (~Z_n – zamknięcie zaworu lub wyłączenie pompy dozującej Z_n),

T_{MIN} – uruchomienie timera T_{MIN} odliczającego czas T_1 pozostawania roztworu w mieszalniku M_1 (~T_{MIN} – wyłączenie timera T_{MIN}),

T_M – uruchomienie timera T_M odliczającego czas T_2 mieszania roztworu w mieszalniku M_2 ($\sim T_M$ – wyłączenie timera T_M),
M – załączenie mieszadła ($\sim M$ – wyłączenie mieszadła),
Test PH i ustalenie W – obliczenie wagi $W(t_8, p''_4) = W(p''_5, t_{15})$ na podstawie wskazań pehametru.



Rys. 3. Sieć Petriego opisująca algorytm sterowania układem mieszalnika z rys. 3
Fig. 3. The Petri Net describing control algorithm of the mixer plant from Fig. 3

Tranzycjom t przypisano sygnały pochodzące z czujników i elementów czasowych, przyjmując następujące oznaczenia:
A_i – napełnienie dozownika D_i, $i = 1, 2, 3$ – sygnał analogowy ($\sim A_n = 1 - A_n$),
B_k – opróżnienie mieszalnika M_k, $k = 1, 2$ – sygnał binarny,
T_1 – upłynął minimalny czas pozostawania roztworu w M_1,
T_2 – upłynął czas mieszania roztworu w mieszalniku M_2.

4. Realizacja sieci w języku ST

Proponowana metoda realizacji rozmytej interpretowanej sieci Petriego w języku ST polega na bezpośrednim wykorzystaniu równań opisujących moment przygotowania tranzycji do uaktywnienia. W tej sieci tranzycja $t \in T$ jest przygotowana do uaktywnienia dla znakowania M: $P \rightarrow [0,1]$ od momentu, gdy stopnień spełnienia warunku $\Theta(t) = \vartheta$, związanego z tą tranzycją, jest większy od zera i są spełnione warunki:

$$\forall p \in {}^\bullet t, M(p) \geq \frac{W(p,t)}{K(p)} \quad \text{i} \quad \forall p \in t^\bullet, M(p) \leq \frac{K(p)\text{-}W(t,p)}{K(p)} \quad (1)$$

do momentu, gdy:

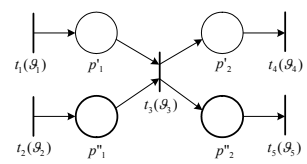
$$\exists p' \in {}^\bullet t, M(p) = 0 \quad \text{lub} \quad \exists p' \in t^\bullet, M(p) = 1, \quad (2)$$

gdzie: $K(p)$ – pojemność miejsca p, $W(p, t)$ i $W(t, p)$ – wagi łuków łączących miejsce p z tranzycją t i odwrotnie, $^\bullet t = \{p \in P \mid (p, t) \in R\}$ – zbiór miejsc wejściowych tranzycji t, $t^\bullet = \{p \in P \mid (t, p) \in R\}$ – zbiór jej miejsc wyjściowych, p' – miejsce modelujące działanie lub proces; p'' – miejsce modelujące zasoby, p – miejsce p' lub p''.

Metoda realizacji sieci uwzględnia także zmiany zachodzące w znakowaniu poszczególnych miejsc sieci. Dowolną strukturę sieci rozpatruje się biorąc kolejno pod uwagę każdą tranzycję. Gdy wraz z nadejściem sygnału synchronizującego pracę sieci, przygotowana tranzycja zostanie uaktywniona, to nowe znakowanie sieci, które wystąpi, gdy stopień spełnienia warunku związanego z tą tranzycją $\Theta(t) = \vartheta \in [0,1]$ zwiększy się o $\Delta\vartheta$, można wyznaczyć z zależności:

$$M'(p) = \begin{cases} M(p) - \Delta\vartheta \cdot \dfrac{W(p,t)}{K(p)} & \Leftrightarrow \quad p \in {}^\bullet t, \\ M(p) + \Delta\vartheta \cdot \dfrac{W(t,p)}{K(p)} & \Leftrightarrow \quad p \in t^\bullet, \\ M(p) & \Leftrightarrow \quad p \notin {}^\bullet t \cup t^\bullet. \end{cases} \quad (3)$$

Dla przykładu przeanalizowano fragment rozmytej interpretowanej sieci Petriego pokazany na rys. 4. Fragment ten składa się z dwóch miejsc typu p' i dwóch miejsc typu p'', przy czym obydwa typy miejsc stanowią zarówno miejsca wejściowe (p'_1, p''_1), jak i wyjściowe (p'_2, p''_2) tranzycji t_3. Przedstawiony na rys. 4 diagram sieci jest na tyle ogólny, że przez analogię można w języku ST opisać dowolnie złożoną jej strukturę.



Rys. 4. Wybrany fragment rozmytej interpretowanej sieci Petriego
Fig. 4. Selected part of Fuzzy Interpreted Petri Net

Realizację w języku ST fragmentu rozmytej interpretowanej sieci Petriego rozpoczyna się od deklaracji zmiennych, w sposób pokazany na list. 1. Znaczenie poszczególnych zmiennych podano w tab. 1.

Listing 1. Deklaracje zmiennych dla fragmentu sieci z rys. 3
Listing 1. Variables declaration for the Net fragment from Fig. 3

```
VAR
 MMAX : INT := INT#16#3FF;
 MP1, MP2 : INT;
 MPPK1, MPPK2 : INT;
 KPP1, KPP2, WPP1T3, WT3PP2 : INT;
 T3, DT3 : INT :=0;
 AP1, AP2, APP1_I, APP2_O : BOOL := FALSE;
END_VAR
```

Na list. 2 pokazano program główny w języku ST, realizujący fragment sieci z rys. 4, wprowadzając dodatkowe numerowanie linii kodu. W linii 1 następuje sprawdzenie warunku przygotowania tranzycji t_3 do aktywacji. Zgodnie z równaniem (1) tranzycja ta będzie przygotowana do uaktywnienia, gdy wystąpi koniunkcja wartości zmiennych: *AP1*, zanegowanej *AP2*, *APP1_I* oraz *APP2_O*. Wtedy realizowany jest kod zawarty w liniach 2…7. W linii 2 obliczany jest przyrost stopnia spełnienia warunku związanego z tranzycją t_3. Wystarczy tutaj obliczyć różnicę pomiędzy bieżącą wartością tego stopnia (zmienna *T3*) a wartością znakowania w miejscu wyjściowym typu p' tej tranzycji (zmienna *MP2*). Jeżeli ten przyrost jest większy od zera, należy zaktualizować znakowanie we wszystkich miejscach wejściowych i wyjściowych tranzycji t_3. Jest to realizowane w liniach 4…7, bezpo-

średnio na podstawie równania (3). Należy tu zwrócić uwagę, że zmienne *MPPK1* oraz *MPPK2* zawierają stan znakowania miejsc p''_1 i p''_2 pomnożony przez pojemność tych miejsc.

Tab. 1. Znaczenie zmiennych występujących w programie
Tab. 1. Meaning of the used variables

Zmienna	Znaczenie
MMAX	Maksymalna wartość zbioru liczb naturalnych {0, 1, ..., *MMAX*}, na który rzutowany jest przedział jednostkowy [0, 1].
MP1, MP2	Stan znakowania w miejscach p'_1, p'_2: $M(p'_1)$, $M(p'_2)$.
MPPK1, MPPK2	Stan znakowania w miejscach typu p'' pomnożony przez pojemność tych miejsc: $M(p''_1)\cdot K(p''_1)$, $M(p''_2)\cdot K(p''_2)$.
KPP1, KPP2	Pojemność miejsc p''_1 i p''_2: $K(p''_1)$, $K(p''_2)$.
WPP1T3	Waga łuku $W(p''_1, t_3)$.
WT3PP2	Waga łuku $W(t_3, p''_2)$.
T3	Stopień spełnienia warunku związanego z tranzycją t_3: ϑ_3. Wartość tej zmiennej musi zawierać się w zbiorze {0, 1, ..., *MMAX*}.
DT3	Przyrost stopnia spełnienia warunku związanego z tranzycją t_3: $\Delta\vartheta_3$.
AP1, AP2	Dodatkowe zmienne związane z miejscami p'_1 *i* p'_2, wykorzystywane do obliczania warunku przygotowania tranzycji do aktywacji. Mogą także służyć do binarnego sterowania elementami wykonawczymi. Zmienna przyjmuje wartość *TRUE* od momentu, gdy znakowanie w miejscu osiągnie wartość *MMAX*, do momentu, gdy znakowanie będzie wynosiło 0.
APP1_I	Zmienna związana z miejscem p''_1, wykorzystywana do obliczania warunku przygotowania tranzycji do aktywacji. Przyjmuje wartość *TRUE*, gdy $M(p''_1)\cdot K(p''_1) \geq W(p''_1, t_3)$.
APP2_O	Zmienna związana z miejscem p''_2, wykorzystywana do obliczania warunku przygotowania tranzycji do aktywacji. Przyjmuje wartość *TRUE*, gdy $M(p''_2)\cdot K(p''_2) \leq K(p''_2) - W(t_3, p''_2)$.

Listing 2. Program główny realizujący fragment sieci z rys. 3
Listing 2. The main program describing Petri Net fragment form Fig. 3

```
1: IF AP1 AND (NOT AP2) AND APP1_I AND APP2_O THEN
2:  DT3:=T3-MP2;
3:  IF DT3>0 THEN
4:  MP1:=MP1-DT3;
5:  MP2:=MP2+DT3;
6:  MPPK1:=MPPK1-WPP1T3*DT3;
7:  MPPK2:=MPPK2+WT3PP2*DT3;
8: END_IF
9:
10: IF MP1=MMAX THEN AP1:=TRUE; END_IF
11: IF MP1=0 THEN AP1:=FLASE; END_IF
12: IF MP2=MMAX THEN AP2:=TRUE; END_IF
13: IF MP2=0 THEN AP2:=FALSE; END_IF
14: IF MPPK1>=(WPP1T3*MMAX) THEN APP1_I:=TRUE;
   ELSE APP1_I:=FALSE; END_IF
15: IF MPPK2<=((KP2-WT3PP2)*MMAX) THEN
   APP2_O:=TRUE; ELSE APP2_O:=FALSE; END_IF
```

W kolejnych liniach kodu 10…15 obliczane są wartości dodatkowych zmiennych, które wykorzystuje się do określenia warunku przygotowania tranzycji do aktywacji. Odbywa się to bezpośrednio na podstawie równań (1) i (2). Z każdym miejscem p'_i związana jest jedna zmienna *APi*, natomiast z miejscem p''_j dwie zmienne *APPj_I*, *APPj_O*. W rozważanym przykładzie opisywane jest uaktywnienie tranzycji t_3, dlatego też wykorzystano tylko zmienne *APP*1*_I* i *APP*2*_O*, gdyż miejsce p''_1 jest miejscem wejściowym, a miejsce p''_2 miejscem wyjściowym tranzycji t_3. Zmienna *APP*1*_O* byłaby wykorzystywana w przypadku sprawdzaniu przygotowania tranzycji t_2 do aktywacji. Podobnie zmienna *APP*2*_I* dla tranzycji t_5. Dla poprawnego działania programu ważne jest, aby fragment kodu odpowiedzialny za obliczanie wartości tych zmiennych występował po fragmencie realizującym sprawdzanie warunków przygotowania wszystkich tranzycji do uaktywnienia i aktualizowanie znakowania miejsc.

5. Podsumowanie

Realizacja rozmytej interpretowanej sieci Petriego z wykorzysta-niem tekstowego języka programowania ST – w porównaniu z dotychczasowymi metodami [2] – pozwala znacznie uprościć proces opisu sieci. Dodatkowo uzyskany kod programu może być przenoszony pomiędzy sterownikami PLC różnych producentów, spełniającymi wymagania normy IEC 61131-3.

Zastosowanie specjalizowanego sterownika programowalnego, zbudowanego w oparciu o układy FPGA pozwala uzyskać bardzo krótki czas obliczeń nowego znakowania sieci. W przypadku układu sterowania mieszalnikiem, realizacja kompletnego algorytmu sterowania, opisanego rozmytą interpretowaną siecią Petriego, wraz ze skalowaniem wartości sygnałów z czujników analogowych itp. zajmuje średnio tylko 86 µs (czas ten może nieznacznie się zmieniać w zależności od ilości aktywnych tranzycji w sieci). Czas cyklu sterownika może więc wynosić poniżej 100 µs, co jest wartością nieosiągalną dla typowych, przemysłowych sterowników PLC.

6. Literatura

[1] Gniewek L.: Transformacja rozmytej interpretowanej sieci Petriego na schemat układu logicznego. Pomiary Automatyka Kontrola, vol. 56, nr 11, s. 1368-1371, 2010.

[2] Gniewek L.: Implementacja rozmytej interpretowanej sieci Petriego w sterowniku PLC. Metody wytwarzania i zastosowania systemów czasu rzeczywistego (red. Trybus L., Samolej S.), WKiŁ, Warszawa 2010, s. 71-80.

[3] Gniewek L.: Modelowanie układu sterowania i diagnostyki za pomocą rozmytej interpretowanej sieci Petriego. Systemy wykrywające, analizujące i tolerujące usterki (red. Kowalczyk Z.), PWNT, Gdańsk 2009, s. 105-112.

[4] Rzońca D., Sadolewski J., Trybus B.: Prototype environment for controller programming in the IEC 61131-3 ST language, Computer Science and Information Systems, vol. 4, no. 2, December 2007.

[5] Sadolewski J., Trybus B.: Wieloplatformowa maszyna wirtualna dla systemów sterowania. Modele i zastosowania systemów czasu rzeczywistego. (red. Z. Mazur, Z. Huzar), WKŁ, Warszawa 2008, s. 293-302.

[6] Hajduk Z., Trybus B., Sadolewski J.: Sprzętowa implementacja maszyny wirtualnej dla sterowników programowalnych, Metody Wytwarzania i Zastosowania Systemów Czasu Rzeczywistego (red. Trybus L., Samolej S.), WKŁ, 2010, s. 333-343.

[7] Hajduk Z., Sadolewski J., Trybus B.: FPGA-based Execution Platform for IEC 61131-3 Control Software, Przegląd Elektrotechniczny, nr 8, s. 187-191, 2011.

[8] Thomas D. E., Moorby P. R.: The Verilog Hardware Desciption Language, Fifth Edition, Kluwer Academic Publishers, 2002.

[9] Hajduk Z.: Wprowadzenie do języka Verilog, Wydawnictwo BTC, Legionowo 2009.

[10] Hajduk Z.: Zmiennoprzecinkowa jednostka arytmetyczna dla sprzętowej maszyny wirtualnej, Pomiary Automatyka Kontrola, vol. 57, nr 1, s. 82-85, 2011.

[11] Hajduk Z.: Moduł komunikacyjny dla sprzętowej implementacji maszyny wirtualnej, Elektronika konstrukcje technologie zastosowania, nr 5, s. 172-175, 2011.

[12] Hajduk Z.: Prototyp programowalnego sterownika logicznego dla sprzętowej maszyny wirtualnej, Elektronika konstrukcje technologie zastosowania, nr 4, s.114-118, 2011.

otrzymano / received: 25.05.2012
przyjęto do druku / accepted: 01.11.2012 *artykuł recenzowany / revised paper*