Metoda modelowania procesów sekwencyjnych i współbieżnych w środowisku sterowników PLC

Krzysztof Franczok

Fabryka Maszyn ROTOX Sp. z o.o., Pokój k. Opola

Streszczenie: W artykule zaproponowano metodę modelowania i realizacji procesów sekwencyjnych i współbieżnych w środowisku sterowników programowalnych PLC, z wykorzystaniem instrukcji CASE OF, języka tekstowego ST. Pokazano metodę transformacji modelu procesu sterowania, zbudowanego w postaci sieci Petriego, wprost do postaci kodu programu sterownika przemysłowego, bez konieczności pośredniego budowania schematu logicznego. Wskazano korzyści płynące z zastosowania tej metody, w realizacji zarówno prostych jak i złożonych algorytmów sterowania procesami dyskretnymi w systemach czasu rzeczywistego. Przedstawiono rozwiązanie algorytmu sterowania procesami sekwencyjnymi i współbieżnymi na przykładzie procesu ważenia. Wskazano kierunki dalszego wykorzystania i rozwoju zaprezentowanej metody do realizacji procesów sekwencyjnych i współbieżnych.

Słowa kluczowe: język strukturalny ST, sieci Petriego, modelowanie procesów dyskretnych, instrukcja CASE OF, procesy sekwencyjne i współbieżne

DOI: 10.14313/PAR_203/98

1. Wprowadzenie

W praktyce inżynierskiej bardzo często występują sekwencyjne i współbieżne procesy dyskretnie. Zadaniem inżynierów automatyków jest szybkie i bezbłędne tworzenie, na podstawie specyfikacji wymagań, modelu algorytmu automatycznego sterowania dyskretnym procesem produkcyjnym, a następnie zaimplementowanie go w systemach sterujących procesem, jakim zwykle jest system czasu rzeczywistego oparty na sterowniku PLC.

Do modelowania i badania właściwości dynamicznych takich procesów przydatnym narzędziem są sieci Petriego i programy do ich analizy. Sieci Petriego dobrze ilustrują relacje zachodzące w procesach sterowania i umożliwiają formalną analizę ich właściwości behawioralnych.

Wśród obecnie stosowanych rozwiązań najbardziej popularny jest język SFC, ale ma on pewne wady:

- język SFC nie zawsze jest dostępny w sterownikach PLC różnych producentów,
- z uwagi na graficzną formę, programy napisane w języku SFC mają gorsze parametry wydajnościowe niż programy pisane językami tekstowymi [1],

- nie jest to język samodzielny – zarówno akcje wykonywane w ramach miejsc, jak i tranzycje pisane są z pomocą innych języków programowania, jak język IL, FBD, LD czy ST [2].

Do transformacji modelu procesu opisanego interpretowaną siecią Petriego, można jednak wykorzystać język strukturalny ST.

W artykule [3] opisana została sprzętowo-programowa realizacja rozmytej interpretowanej sieci Petriego za pomocą języka ST. Autorzy wskazali korzyści stosowania języka ST do takich celów oraz zaprezentowali metodę jego wykorzystania do realizacji modelu sieci Petriego.

W pracy [4] przedstawiono w interesujący sposób metodę modelowania sieci Petriego w języku VHDL, którą można również zastosować w języku ST. Wskazano na występującą w większości projektów inżynierskich konieczność synchronizowania niezależnych procesów sekwencyjnych i związane z tym trudności, zaproponowano ich rozwiązanie.

Zastosowanie połączenia języka SFC z językiem ST do realizacji modelu maszyny stanowej UML przedstawiono w pracy [2], gdzie wskazano potrzebę określenia formy translacji diagramu stanów na program dla sterownika PLC.

Proponowana w niniejszym artykule metoda modelowania i realizacji sekwencyjnych i współbieżnych procesów dyskretnych stanowi próbę rozwiązania problemów, na które wskazują autorzy wspomnianych publikacji i może być alternatywą dla dotychczas proponowanych rozwiązań.

Celem jest zaproponowanie wydajnej, wygodnej i funkcjonalnej dla projektanta systemów sterowania, metody konwersji modelu procesu, z postaci sieci Petriego, na program sterownika PLC, z zachowaniem struktury i charakteru modelu, z wykorzystaniem wydajnego tekstowego języka programowania ST.

2. Korzyści wynikające ze stosowania języka tekstowego ST

W pracy [1] wykazano, że z punktu widzenia wymagań stawianych systemom czasu rzeczywistego, do realizacji zadań spełniających te wymagania, jest zalecane stosowanie języków tekstowych. Czasy minimalne, maksymalne oraz odchylenie standardowe czasu wykonania są znacznie mniejsze dla języków tekstowych niż graficznych [1]. Wyjaśniono, że w przy-

Rys. 1. Tranzycje wejściowe i wyjściowe miejsca p [4]

Fig. 1. In/out transitions for place p [4]

Rys. 2. Miejsca wejściowe i wyjściowe tranzycji t [4]

Fig. 2. In/out places for transition t [4]

0002	CASE P OF
0003	0:
0004	instrukcja_0;
0005	1:
0006	instrukcja_1;
0007	2:
0008	instrukcja_2;
0009	n:
0010	instrukcja_n;
0011	ELSE
0012	instrukcja_else;
0013	END_CASE
0014	

Rys. 3. Fragment programu napisanego za pomocą instrukcji CASE OF

Fig. 3. A fragment of the program written using the CASE OF statement

padku języków tekstowych operacje wykonywane są wyłącznie w akumulatorach jednostki centralnej, do których czas dostępu jest krótki, natomiast w przypadku języków graficznych wykonanie złożonych operacji wymaga stosowania zmiennych pomocniczych do zapisu wyników pośrednich w pamięci o dłuższym czasie dostępu.

Autor pracy [1] zwraca uwagę, że w przypadku języków graficznych dodatkowym czynnikiem pogarszającym szybkość i determinizm czasowy działania, jest wykonywanie operacji innych niż logiczne, tylko przez wywołanie funkcji. Wykazano przy tym, że z punktu widzenia wydajności, język listy instrukcji IL oraz język strukturalny ST są równoważne. Wskazano jednak, że język ST jest bardziej funkcjonalny i zapewnia dostęp do wszystkich zasobów CPU.

Zalety języka ST:

- szybkość wykonywania programu przez sterownik PLC – jak wykazano w [1], wśród języków opisanych normą PN-EN 61131-3 programy napisane w językach IL oraz ST są wykonywane w najkrótszym czasie,
- łatwość implementacji skomplikowanych operacji matematycznych [1],
- łatwość stosowania pętli i obróbki skomplikowanych struktur wielowymiarowych o zróżnicowanych typach danych, pojedynczych i w postaci tablic,
- możliwość budowania reguł logiki rozmytej [3],
- łatwość wprowadzania obszernych komentarzy,
- łatwość przenoszenia kodu, co pozwala na uniezależnienie się od dostawcy sprzętu [3].

3. Transformacja sieci Petriego do kodu programu w języku ST

3.1. Sieć Petriego

Przed przedstawieniem możliwości modelowania sieci Petriego w języku ST należy przypomnieć wybrane zagadnienia z teorii sieci określające ich strukturę, których opis przedstawiono w [4]:

Zbiory tranzycji wejściowych i wyjściowych miejsca p , definiowane są następująco:

$$\bullet p = \{t \in T: (t, p) \in F\} \quad (1)$$

$$p \bullet = \{t \in T: (p, t) \in F\} \quad (2)$$

gdzie: T – zbiór tranzycji sieci, F – relacja przepływu w danej sieci (zbiór łuków), t – rozpatrywana tranzycja, p – rozpatrywane miejsce.

Zbiory miejsc wejściowych i wyjściowych danej tranzycji t określane są zależnościami:

$$\bullet t = \{p \in P: (p, t) \in F\} \quad (3)$$

$$t \bullet = \{p \in P: (t, p) \in F\} \quad (4)$$

gdzie: P – zbiór miejsc sieci,

3.2. Metoda CASE OF

Proponowana metoda transformacji modelu w postaci sieci Petriego do kodu programu, opiera się na szczególnym zastosowaniu instrukcji CASE OF języka ST, która z założenia jest elementem wyboru.

Znając zasadę działania sterownika programowalnego, który powtarza cykl instrukcji oraz etapy:

- autodiagnostyka,
- odczyt wejść,
- wykonanie programu,
- zadania komunikacyjne,
- ustawienie wyjść,

można wykorzystać właściwości instrukcji CASE OF do budowy programu sekwencyjnego. Odpowiednie sterowanie wieloma pojedynczymi programami sekwencyjnymi umożliwia realizację procesów współbieżnych i ich synchronizację.

Instrukcja CASE OF jest wygodnym zastępnikiem w języku strukturalnym instrukcji warunkowej IF THEN, szczególnie w przypadku występowania wielu alternatywnych warunków. Konstrukcja programu oparta na instrukcji CASE OF, w języku strukturalnym przyjmuje postać przedstawioną na rys. 3, gdzie p jest liczbą całkowitą nieujemną.

W zależności od wartości zmiennej p wykonywana jest tylko jedna z instrukcji z przedziału $[0 \dots n]$

Ten sam program, zrealizowany za pomocą instrukcji IF THEN przyjąłby postać, jak na rys. 4.

Jeśli kod programu z rys. 3 rozbudujemy w sposób przedstawiony na rys. 5, otrzymamy program sekwencyjny, w którym jeśli:

- $p = 0$, to zostanie wykonana instrukcja_0 oraz jeśli spełniony będzie warunek ($\text{warunek_0_1} = \text{true}$), to zmienna p przyjmie wartość 1,
- $p = 1$, to zostanie wykonana instrukcja_1 oraz jeśli spełniony będzie warunek ($\text{warunek_1_2} = \text{true}$), to zmienna p przyjmie wartość 2,
- $p = 2$, to zostanie wykonana instrukcja_2 a zmienna p przyjmie bezwarunkowo wartość 0 i ponownie zostanie rozpoczęta sekwencja.

Z wykorzystaniem instrukcji CASE OF, w zależności od nieujemnej wartości całkowitej zmiennej p , wykonywane są instrukcje zawarte tylko dla danego przypadku określonego wartością zmiennej p . W ramach tego przypadku może być wykonana dowolna liczba instrukcji, pętli iteracyjnych, operacji arytmetycznych, porównawczych i innych. Wszystkie pozostałe przypadki, określone innymi niż aktualna wartość zmiennej p , nie są brane pod uwagę podczas wykonywania programu.

Można zatem założyć, że jeśli:

$$p \in P, p \geq 0 \quad (5)$$

to dla określonej wartości całkowitej p program znajduje się w określonym stanie – miejscu reprezentowanym przez zdarzenie określone wartością p .

Dopóki zmienna p będzie przyjmowała tę samą wartość, sterownik programowalny w każdym cyklu pracy będzie wykonywał instrukcje zadeklarowane dla przypadku p o tej wartości. Zmiana wartości zmiennej p spowoduje, że sterownik programowalny wykona w następnym cyklu pracy instrukcje określone dla nowej wartości zmiennej p , a zatem zmieni stan – miejsce w sekwencji.

Aby możliwe było przejście do nowego stanu procesu sekwencyjnego, konieczna jest zmiana wartości zmiennej p . W tym celu konieczne jest zadeklarowanie warunków odpalenia tranzycji. Taka deklaracja może przyjąć postać instrukcji *if then end_if* i musi być umieszczona w ramach aktywnego przypadku p .

Spełnienie warunków *if then end_if* w czasie aktywności miejsca p spowoduje zmianę wartości zmiennej p na taką,

```

0001
0002 IF      p=0 THEN instrukcja_0;
0003 ELSIF   p=1 THEN instrukcja_1;
0004 ELSIF   p=2 THEN instrukcja_2;
0005 ELSIF   p=n THEN instrukcja_n;
0006 ELSE
0007 END_IF   instrukcja_els;

```

Rys. 4. Fragment programu napisanego za pomocą instrukcji IF THEN

Fig. 4. A fragment of the program written using IF THEN statement

```

0001
0002 CASE P OF
0003 0:
0004     instrukcja_0;
0005     IF  warunek_0_1
0006 THEN P:=1;
0007 END_IF
0008 1:
0009     instrukcja_1;
0010     IF  warunek_1_2
0011 THEN P:=2;
0012 END_IF
0013 2:
0014     instrukcja_2;
0015     P:=0;
0016 ELSE
0017     instrukcja_else;
0018     P:=0;
0019 END_CASE
0020

```

Rys. 5. Przykład programu sekwencyjnego napisanego za pomocą instrukcji CASE OF

Fig. 5. An example of a sequential program written using the CASE OF statement

jaka odpowiada pożądanemu miejscu procesu sekwencyjnego. Po zmianie wartości zmiennej p , w kolejnym cyklu pracy, sterownik wykona instrukcje zawarte w ramach nowego przypadku określonego nową wartością p .

Możliwe jest wprowadzenie więcej niż jednej instrukcji warunkowej *if then end_if* w ramach jednego aktywnego miejsca p , co odpowiada kilku tranzycjom stanowiącym wybór alternatywny. Za pomocą instrukcji warunkowej *if then end_if*, możliwe jest uruchomienie więcej niż jednego nowego procesu sekwencyjnego, co odpowiada tranzycji dzielącej sekwencje na dwa procesy współbieżne.

4. Modelowanie procesu sekwencyjnego

4.1. Wykorzystanie instrukcji CASE OF do implementacji sieci Petriego

Aby zilustrować zasadę modelowania programu sekwencyjnego na bazie instrukcji CASE OF, posłużono się przykładem zaczerpniętym z pracy [5], przedstawiającym metodę transformacji sieci SFC na sieć Petriego i dla tego przykładu zaprezentowano proponowany sposób, z wykorzystaniem instrukcji CASE OF.

Elementy graficzne przedstawione na rys. 6, właściwe dla języka SFC oraz dla sieci Petriego, w języku strukturalnym przyjmują postać, jak na rys. 7.

Program inicjowany jest w miejscu P0, gdy zmienna p typu całkowitego przyjmuje wartość 0. Dopóki nie zostanie spełniony warunek $C = \text{true}$, sterownik PLC będzie w każdym cyklu programu wykonywał instrukcje przewidziane w miejscu P0. Inne części programu nie są wykonywane przez sterownik. W momencie spełnienia warunku $C = \text{true}$, program przypisuje zmiennej p wartość 9, co powoduje, że w kolejnym cyklu pracy program będzie wykonywany tylko w ramach miejsca P9.

4.2. Przykład dla prostego procesu sekwencyjnego ważenia substancji

Dla zobrazowania metody przekształcenia modelu prostego procesu dyskretnego, opisanego za pomocą sieci Petriego, na postać programu napisanego w języku strukturalnym, posłużono się przykładem zaczerpniętym z pracy [5]. Na rys. 8, przedstawiono przykład procesu ważenia substancji, w posta-

Rys. 6. Porównanie elementów graficznych sieci SFC i sieci Petriego [5]

Fig. 6. Comparison of graphics SFC network and Petri net

Rys. 7. Przykład przekształcenia sieci Petriego do języka ST

Fig. 7. An example of the transformation of Petri nets as ST language

Rys. 8. Sieć SFC opisująca proces ważenia substancji [5]

Fig. 8. SFC network describing the process of weighing of substance [5]

Rys. 9. Sieć Petriego po transformacji z sieci SFC [5]

Fig. 9. Petri net after transformation of the SFC network [5]

ci języka SFC, a na rys. 9, jego odwzorowanie w postaci sieci Petriego.

Na rys. 10 przedstawiono ten sam proces, zaimplementowany w sterowniku PLC w języku strukturalnym z wykorzystaniem instrukcji CASE OF. Dzięki właściwościom instrukcji CASE OF można w pełni i bezpośrednio odwzorować strukturę opisaną za pomocą sieci Petriego.

Program rozpoczyna pracę w miejscu P1 i oczekuje na spełnienie warunków umożliwiających odpalenie tranzycji T(P1, P20) – sygnalizowanie awarii lub T(P1, P30) – rozpoczęcie właściwego procesu. Jeśli zmienna p przyjmuje wartość 1, to program PLC pomija kod programu znajdujący się w miejscach odpowiadających wartościom innym niż 1. Jeśli spełnione zostaną warunki umożliwiające odpalenie tranzycji, w następnym cyklu sterownika wykonywany będzie tylko program zawarty w miejscu p, osiągnięty przez odpalenie danej tranzycji.

4.3. Modelowanie procesów współbieżnych

Implementacja przedstawiona na rys. 10, realizuje hierarchiczną sieć Petriego w postaci jednego programu sekwencyjnego i z wykorzystaniem tej samej zmiennej p, określającej poszczególne miejsca, również w ramach makromiejsca P30. Taka forma jest dopuszczalna w przypadku, gdy w ramach makromiejsca P30 wykonywany jest tylko jeden proces sekwencyjny. W przypadku gdyby w ramach makromiejsca P30, realizowany był współbieżnie, więcej niż jeden proces sekwencyjny, program musiałby zostać zapisany w postaci przedstawionej na rys. 11.

W takim przypadku w chwili odpalenia tranzycji T(P1, P30) należy aktywować wszystkie współbieżne procesy sekwencyjne, a tranzycję T(P30, P1) odpalić dopiero, gdy zostaną one zakończone. W ten sposób można uruchomić za pomocą zmiennej P1_Start, P2_Start, ..., Pn_Start szereg procesów sekwencyjnych wykonywanych współbież-


```

0001
0002 CASE P OF
0003 (*
0004 (* P1 Krok (Miejsce) inicjujace *)
0005 *)
0006 1:
0007
0008 (*
0009 (* T(P1,P20) tranzycja *)
0010 *)
0011 IF Awaria
0012 THEN
0013     P:=20;
0014 END_IF
0015 (*
0016 (* T(P1,P30) tranzycja *)
0017 *)
0018 IF Start
0019 AND NOT Awaria
0020 THEN
0021     P:=30;
0022 END_IF
0023 (*
0024 (* P20 Krok (Miejsce) Awaria *)
0025 *)
0026 20:
0027     Status_Awaria:=TRUE;
0028
0029 (*
0030 (* T(P20,P1) tranzycja *)
0031 *)
0032 IF Zerowanie
0033 THEN
0034     Status_Awaria:=FALSE;
0035     P:=1;
0036 END_IF
0037 (*
0038 (* P30 Krok (Miejsce) Napelnianie *)
0039 *)
0040 30:
0041     Napelnianie:=TRUE;
0042 (*
0043 (* T(P30,P31) tranzycja *)
0044 *)
0045 IF Pojemnik_Pelen
0046 THEN
0047     Napelnianie:=FALSE;
0048     P:=31;
0049 END_IF
0050 (*
0051 (* P31 Krok (Miejsce) Wazenie *)
0052 *)
0053 31:
0054     Wazenie:=TRUE;
0055 (*
0056 (* T(P31,P32) tranzycja *)
0057 *)
0058 IF Pojemnik_Swazon
0059 THEN
0060     Wazenie:=FALSE;
0061     P:=32;
0062 END_IF
0063 (*
0064 (* P32 Krok (Miejsce) Oproznianie *)
0065 *)
0066 32:
0067     Oproznianie:=TRUE;
0068 (*
0069 (* T(P32,P33) tranzycja *)
0070 *)
0071 IF Pojemnik_Pusty
0072 THEN
0073     Oproznianie:=FALSE;
0074     P:=33;
0075 END_IF
0076 (*
0077 (* P33 Krok (Miejsce) Zapam masy *)
0078 *)
0079 33:
0080     Zapamietanie_Masy:=TRUE;
0081 (*
0082 (* T(P33,P1) tranzycja *)
0083 *)
0084 IF Koniec
0085 THEN
0086     Zapamietanie_Masy:=FALSE;
0087     P:=1;
0088 END_IF
0089
0090 END_CASE

```

Rys. 10. Implementacja procesu ważenia w sterowniku PLC, z wykorzystaniem instrukcji CASE OF

Fig. 10. Implementation of the weighing process in the PLC environment, using CASE OF statement

nie, a w miejscu P30 oczekiwać na spełnienie warunków zakończenia tych procesów w postaci wartości logicznej true dla zmiennych P1_Koniec, P2_Koniec, ..., PN_Koniec.

Uruchomienie poszczególnych procesów wykonywanych współbieżnie nie musi być realizowane przez zmienne pomocnicze Pn_Start, Pn_Koniec. W tym celu można użyć wartości zmiennej p_n określającej aktywne miejsce procesu sekwencyjnego n.

5. Istotne cechy metody CASE OF

5.1. Pełna kontrola nad przebiegiem procesu

Każde miejsce sieci Petriego ma swój numer i zawiera tranzycje do kolejnych miejsc. Ponieważ zawsze tylko jedno miejsce jest aktywne w ramach danego programu, w przypadku błędnego działania programu można łatwo zlokalizować przyczynę na podstawie numeru aktywnego miejsca i analizy warunków odpalenia tranzycji wyjściowej miejsca p.

Jeśli zmienna p zostanie zadeklarowana w sterowniku PLC jako zmienna nieulotna, stan w którym znajduje się proces, może być pamiętany nawet po wyłączeniu zasilania sterownika. Po ponownym załączeniu zasilania sterownika, program rozpocznie pracę od miejsca, w którym proces się znajdował przed wyłączeniem zasilania.

W przypadku wystąpienia sytuacji awaryjnych, możliwe jest wpłynięcie na zachowanie się programu za pomocą programu nadrzędnego. Program nadrzędny może, w sytuacji zadziałania wyłącznika bezpieczeństwa, ustawić wartość zmiennej p tak, by program wszedł do miejsca P0, realizującego awaryjne zatrzymanie procesu dyskretnego, bez względu na to, w jakim miejscu p proces się w danej chwili znajdował.

5.2. Efektywność obliczeniowa sterownika

W pracy [1] autor określa górne oszacowanie czasu wykonania procedury obliczeniowej w środowisku PLC jako:

$$T_e \in F \sum T_{instr} \quad (6)$$

gdzie: T_e – szacowany czas wykonania procedury, T_{instr} – maksymalny czas wykonania poszczególnych instrukcji, zależnie od typu CPU oraz typu danych, na jakich są wykonywane działania; $F > 1$ – stałą zależną od typu użytej CPU.

Ponieważ w przypadku instrukcji CASE OF sterownik PLC wykonuje tylko tę część kodu programu, która zawiera się w ramach aktywnego miejsca p, pozostała część programu jest przez sterownik pomijana. Oznacza to, że dla programu, który zawiera n miejsc p, czas wykonania programu T_e zależy tylko i wyłącznie od liczby instrukcji m, zawartych w miejscu p, a nie od liczby n miejsc p.

Dzięki tej właściwości, przy założeniu, że dane miejsce p zawiera maksymalnie m instrukcji, w przypadku, gdy $n \rightarrow \infty$, maksymalny czas wykonania programu będzie niezmienny względem liczby n miejsc p, a jedynie zależny od maksymalnego czasu T_e dla aktywnego miejsca p. Właściwość ta pozwala na maksymalne skrócenie czasu cyklu nawet dla bardzo rozbudowanych sekwencji. Pozwala również na wykonanie w jednym cyklu sterownika wielu współbieżnych pro-

```

0001
0002 CASE P OF
0003 (* ----- *)
0004 (* P1 Krok (Miejsce) inicjujace *)
0005 (* ----- *)
0006 1:
0007
0008 (* ----- *)
0009 (* T(P1,P20) tranzycja *)
0010 (* ----- *)
0011 IF Awaria
0012 THEN
0013 P:=20;
0014 END_IF
0015 (* ----- *)
0016 (* T(P1,P30) tranzycja *)
0017 (* ----- *)
0018 IF Start
0019 AND NOT Awaria
0020 THEN
0021 P1_Start:=TRUE;
0022 P :=30;
0023 END_IF
0024 (* ----- *)
0025 (* P20 Krok (Miejsce) Awaria *)
0026 (* ----- *)
0027 20:
0028 Status_Awaria:=TRUE;
0029
0030 (* ----- *)
0031 (* T(P20,P1) tranzycja *)
0032 (* ----- *)
0033 IF Zerowanie
0034 THEN
0035 Status_Awaria :=FALSE;
0036 P :=1;
0037 END_IF
0038 (* ----- *)
0039 (* P30 Krok (Miejsce) Napełnianie *)
0040 (* ----- *)
0041 30:
0042 (* ----- *)
0043 (* T(P30,P31) tranzycja *)
0044 (* ----- *)
0045 IF P1_Koniec
0046 THEN
0047 P1_Koniec:=FALSE;
0048 P :=1;
0049 END_IF
0050 END_CASE

```

Rys. 11. Implementacja procesu ważenia w sterowniku PLC, z wykorzystaniem instrukcji CASE OF, z zachowaniem hierarchii sieci

Fig. 11. Implementation of the weighing process, using the CASE OF instruction, keeping the network hierarchy

gramów realizujących procesy sekwencyjne, przy zastosowaniu sterowników PLC o mniejszej wydajności.

Dla zobrazowania korzyści warto posłużyć się danymi podawanymi przez firmę Beckhoff, producenta systemu czasu rzeczywistego Twincat. Według opublikowanych danych [9], czas wykonania 1000 rozkazów PLC w przypadku standardowego sterownika sprzętowego wynosi ok. 800 μ s, dla szybkiego sterownika sprzętowego jest to czas rzędu 70 μ s, a dla sterownika SoftPLC opartego na przestarzałym już procesorze Pentium III wynosi 15 μ s. W przypadku procesora Intel Core 2 Duo czas ten jest rzędu 1 μ s. W praktyce typowo

```

0052
0053 CASE P1 OF
0054 (* ----- *)
0055 (* P1_0 Krok (Miejsce) Napełnianie *)
0056 (* ----- *)
0057 0:
0058 IF P1_Start
0059 THEN
0060 P1_Start:=FALSE;
0061 P1 :=30;
0062 END_IF
0063 (* ----- *)
0064 (* P1_30 Krok (Miejsce) Napełnianie *)
0065 (* ----- *)
0066 30:
0067 Napełnianie:=TRUE;
0068 (* ----- *)
0069 (* T(P1_30,P1_31) tranzycja *)
0070 (* ----- *)
0071 IF Pojemnik_Pelen
0072 THEN
0073 Napełnianie :=FALSE;
0074 P1 :=31;
0075 END_IF
0076 (* ----- *)
0077 (* P1_31 Krok (Miejsce) Wazenie *)
0078 (* ----- *)
0079 31:
0080 Wazenie :=TRUE;
0081 (* ----- *)
0082 (* T(P1_31,P1_32) tranzycja *)
0083 (* ----- *)
0084 IF Pojemnik_Zwazony
0085 THEN
0086 Wazenie :=FALSE;
0087 P1 :=32;
0088 END_IF
0089 (* ----- *)
0090 (* P1_32 Krok (Miejsce) Oproznianie *)
0091 (* ----- *)
0092 32:
0093 Oproznianie:=TRUE;
0094 (* ----- *)
0095 (* T(P1_32,P1_33) tranzycja *)
0096 (* ----- *)
0097 IF Pojemnik_Pusty
0098 THEN
0099 Oproznianie :=FALSE;
0100 P1 :=33;
0101 END_IF
0102 (* ----- *)
0103 (* P1_33 Krok (Miejsce) Zapam masy *)
0104 (* ----- *)
0105 33:
0106 Zapamietanie_Masy :=TRUE;
0107 (* ----- *)
0108 (* T(P1_33,P1_1) tranzycja *)
0109 (* ----- *)
0110 IF Koniec
0111 THEN
0112 Zapamietanie_Masy:=FALSE;
0113 P1_Koniec :=TRUE;
0114 P1_Koniec :=0;
0115 END_IF
0116
0117 END_CASE

```

wy czas cyklu programu sterownika przyjmuje się 1 ms, co pozwala na wykonanie na procesorze Intel Core 2 Duo do 1 miliona rozkazów sterownika, w jednym cyklu programowym. Według danych firmy Beckhoff możliwe jest uzyskanie czasu cyklu sterownika już od 50 μ s [9].

5.3. Spełnienie kryteriów poprawności formalnej programu

Można założyć, że w przedstawionych przykładach implementacji, jeśli model w postaci sieci Petriego będzie spełniał kryteria formalne, co można zbadać za pomocą dostępnych

metod analizy sieci Petriego, to program dokładnie odwzorowujący sieć Petriego też będzie je spełniał.

5.4. Czytelna struktura programu

Według [6] program ma czytelną strukturę, jeśli w sposób prosty i jednoznaczny można określić, co sterownik PLC ma wykonywać w danej chwili i co będzie wykonywał w dalszych etapach. Program napisany z wykorzystaniem instrukcji CASE OF dokładnie odwzorowuje model opisany siecią Petriego. Jeśli w danej chwili aktywne jest miejsce p , to podczas analizy programu ważna jest tylko ta jego część, która zawarta jest w miejscu p , bo tylko ona jest wykonywana przez procesor sterownika. Na podstawie analizy tranzycji można też jednoznacznie określić, co będzie wykonane w przypadku spełnienia określonych warunków odpalenia tranzycji.

Jeśli nastąpi zatrzymanie procesu, znając wartość zmiennej p , można jednoznacznie określić na jakim etapie programu proces się zatrzymał i ograniczyć się do sprawdzenia aktywnego miejsca p oraz zlokalizowania warunków, które nie zostały spełnione, aby możliwe było odpalenie tranzycji do kolejnego miejsca p (kolejnego etapu procesu dyskretnego). Stanowi to znaczne ułatwienie z punktu widzenia operatora sterowanego procesu. Dzięki tym właściwościom, można szybko znaleźć przyczynę zatrzymania procesu dyskretnego.

6. Wnioski

Przedstawiona metoda implementacji algorytmów sterowania dyskretnego, umożliwia efektywną syntezę programów w środowisku PLC na podstawie modelu algorytmu w postaci sieci Petriego. Model w postaci sieci Petriego jest zrozumiały również dla osób nieznających zasady działania sterowników programowalnych. Istnieją również opracowania pozwalające na transformację diagramów stanów języka UML na sieci Petriego [2, 7], co znacznie ułatwia komunikację między specjalistami różnych branż.

Zaproponowana metoda projektowania algorytmów sterowania posłużyła autorowi, do realizacji algorytmów sterowania urządzeniami takimi jak: centrum obróbki ościennic i skrzydeł okiennych, oraz maszyna magazynująca i sortująca szyby, której zasadę działania oparto na idei magazynu chaotycznego, której celem jest dostarczanie ściśle określonych szyb na stanowiska szklenia, zgodnie z zasadą Just in time.

Dalsze badania mają wykazać możliwość stosowania opisanej metody w realizacji złożonych algorytmów sterowania procesami automatyzacji produkcji, wykorzystującymi elementy teorii złożoności [8].

Bibliografia

1. Oprędkiewicz K., *Porównanie języków programowania*, „Pomiary Automatyka Robotyka” 12/2006, 5–9.
2. Doligalski M., Adamski M., *Projektowanie strukturalnych programów dla PLC na podstawie modelu maszyny stanowej UML*, „Pomiary Automatyka Kontrola”, 5/2007, 42–44.
3. Gniewek L., Hajduk Z., *Sprzętowo-programowa realizacja rozmytej interpretowanej sieci Petriego*, „Pomiary Automatyka Kontrola”, 12/2012, 1113–1116.

4. Węgrzyn M., *Modelowanie sieci Petriego w języku VHDL*, „Przegląd Elektrotechniczny”, 1/2010, 212–216.
5. Węgrzyn A., *Symboliczna analiza układów sterowania binarnego z wykorzystaniem wybranych metod analizy sieci Petriego*, Rozprawa Doktorska, Politechnika Warszawska Wydział Elektroniki i Technik Informacyjnych, 2003.
6. Minas M., Frey G., *Visual PLC-Programming using signal interpreter Petri nets*, American Control Conference 6/2002, 5019–5024.
7. Lasota A., *Modelowanie procesów produkcyjnych z wykorzystaniem diagramów aktywności UML i sieci Petriego*, Akademicka Oficyna Wydawnicza EXIT, Warszawa 2012.
8. Cyklis J., Słota A., *Modelowanie systemów wytwarzania przy pomocy sieci Petriego z wykorzystaniem teorii złożoności*, SOP'2008 CA Systems and Technologies, ISBN 978-83-7242-481-5, 47–52.
9. www.beckhoff.pl. ■

The method for modeling of sequential and concurrent processes in the PLC environment

Abstract: This paper describes a method for modeling and execution of sequential and concurrent processes in the environment of programmable controllers (PLC) with the use of CASE OF statements of the Structured Text language (ST). It depicts the method of transformation of the model of the process control, built on the basis of the Petri net, which leads directly to the form of the program code of industrial controller without the need of creating the intermediate logical schema. The paper reveals the benefits of the implementation of the presented method in the realisation of both simple and complex control algorithms for discrete processes in real time systems. The solution of the control algorithm for sequential and concurrent processes is shown on the example of the weighing process. The paper presents the directions of further development and implementation of the mentioned method concerning the execution of the sequential and concurrent processes.

Keywords: structured text, Petri nets, modeling discrete processes, CASE OF statement, sequential and concurrent processes

Artykuł recenzowany, nadesłany 19.11.2013 r., przyjęty do druku 20.12.2013 r.

mgr inż. Krzysztof Franczok

Programista automatyk w Fabryce Maszyn ROTOX Sp. z o.o. Autor wielu projektów komputerowych systemów sterowania maszyn, przeznaczonych dla przemysłu stolarki budowlanej. Zainteresowania: opracowywanie zaawansowanych projektów sterowania dyskretnymi procesami przemysłowymi.
e-mail: kfranczok@op.pl