ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 1992
Seria: AUTOMATYKA z. 109 Nr kol. 1175

Lech Grodzki
Politechnika Białostocka

METODA OPROGRAMOWANIA STEROWNIKÓW BINARNO-CIĄGŁYCH PROCESÓW PRZEMYSŁOWYCH NA POZIOMIE ASEMBLERA

PROGRAMMING METHOD FOR BINARY-CONTINUOUS INDUSTRIAL PROCESS CONTROLLERS AT ASSEMBLER LEVEL

МЕТОД ПРОГРАММИРОВАНИЯ КОНТРОМЕРОВ ВИНАРНО-НЕПРЕРЫВНЫХ ПРОМЫШЛЕННЫХ ПРОЦЕССОВ НА УРОВНЕ АССЕМБЛЕРА

Streszczenie: Artykuł przedstawia jedną z możliwych metod opisu procesu binarnego na potrzeby sterownika mikroprocesorowego. Metoda ta, wykorzystując podstawowe pojęcia z teorii automatów, cechuje się prostotą i elastycznością w stosowaniu. Omówiono powstały język opisu automatów, generujący kod z informacją o algorytmie pracy automatu.

Summary: The article presents one of the possible method of binary process description for microprocessor controller. This method, based on the basic ideas of automata theory, is characterized by simplicity and flexibility of usage. New binary automata description language, producing code with information about the automata, is presented, too.

Резюме: Работа представляет один из возможных способов описания бинарного процесса для микропроцессорного контромера. Этот способ основан на базе понятий из теории автоматов, характеризуется простотой и эластичностью применения. В работе представлен тоже язык описания автоматов, который генерирует код заключающий информацию об алгоритме работы автомата.

Automatyzując jakiś proces, urządzenie technologiczne, mamy bardzo często do czynienia z co najmniej dwoma rodzajami sterowań: ciągłym i binarnym. Pierwsze z nich polega na regulacji stałowartościowej lub nadążnej, drugie zaś ma za zadanie włączanie i wyłączanie maszyn i urządzeń zgodnie z zadanym stanem pracy. Wymienione rodzaje sterowań składają się na najniższą warstwę struktury systemu automatyki [1]. W konkretnych rozwiązaniach są one realizowane przez lokalne sterowniki mikroprocesorowe. Nietrudno zauważyć, że złożoność sterowania binarnego, związana z ilością możliwych stanów pracy urządzenia oraz wejściowych i wyjściowych zmiennych logicznych, bezpośrednio wiąże się ze złożonością oprogramowania sterownika. Szczególnie uciążliwe jest tworzenie i modyfikacja ciągów instrukcji, składających się na wyrażenia logiczne i skoki warunkowe, a realizujących właśnie to sterowanie. Niniejszy artykuł zawiera propozycję rozwiązania powyższego problemu.

Opis zadań sterowania binarnego

Pierwszym elementem proponowanego rozwiązania jest zastąpienie jednego dużego automatu przez zespół kilku mniejszych, a tym samym łatwiejszych do opanowania, automatów. Przez automat rozumie się tu fragment urządzenia, podzespół, grupę mechanizmów ściśle ze sobą współpracujących w celu realizacji określonego zadania technologicznego. Dokonując podziału na automaty, należy kierować się następującymi regułami:

- liczba stanów w automacie winna być możliwie najmniejsza, ale pozwalająca na prawidłowy opis jego pracy;
- każdy automat może oddziaływać jedynie na "swoje" wyjścia binarne, w przeciwnym razie wystąpią konflikty w pracy poszczególnych automatów;
- ilość sygnałów wymienianych z pozostałymi automatami winna być jak najmniejsza (maksimum kilka).

Takie potraktowanie automatyzowanego obiektu ułatwia nam tworzenie oprogramowania systemowego: zmniejsza liczbę zmiennych logicznych, na których musimy jednocześnie operować, upraszcza wyrażenia logiczne i skoki warunkowe w programach obsługi automatu. Poszczególne automaty mogą być obsługiwane przez jeden sterownik albo przez odrębne jednostki. W tym drugim przypadku możliwe jest budowanie systemów zdecentralizowanych nawet dla dużych obiektów (linii technologicznych). Wymianę informacji, potrzebnej do współpracy pomiędzy automatami, można zorganizować przy użyciu wspólnych obszarów pamięci RAM albo łączy transmisji szeregowej.

Efektem pracy automatu jest zmieniający się stan urządzenia, co jest widoczne np. jako zmiany wyjść binarnych. Traktując nasz automat jako znany z teori układów logicznych automat Moore'a [2], możemy wyróżnić poszczególne stany pracy, uwzględniając różne wartości wyjść binarnych w tych stanach. Z kolei algorytm pracy takiego automatu mozemy przedstawić za pomocą tablicy lub grafu przejść. Ponieważ graf jest bardziej czytelną formą opisu, zajmiemy się właśnie nim. Jego wierzchołki odpowiadają poszczególnym stanom automatu, a krawędzie - możliwym przejściom pomiędzy nimi, przy czym są one opisane wyrażeniami logicznymi warunkującymi te przejścia.

Do pracy automatu potrzebne są pewne wejściowe zmienne logiczne. Z reguły są to: sygnały z zewnętrznych wejść binarnych, informacje pochodzące od innych automatów, integralne zmienne logiczne generowane w obrębie samego automatu (np. o szczególnych stanach wielkości analogowych, stanach awaryjnych dwustanowych elementów wykonawczych itp.). Ogół wymienionych zmiennych będziemy nazywali dalej zmiennymi istotnymi. Wszystkie zmienne istotne można zgrupować w tzw. słowie zmiennych istotnych (SZI). Jest to struktura jedno- lub wielobajtowa, której pojedyncze bity odpowiadają poszczególnym zmiennym. Wyróżnione zmienne istotne winny być jedynymi zmiennymi logicznymi występującymi w wyrazeniach opisujących graf przejść automatu.

Opis automatu w systemie mikroprocesorowym

Pozostaje jeszcze wybór sposobu realizacji w sterowniku mikroprocesorowym przygotowanego grafu przejść. Sprowadza się on do wyboru sposobu zapisu wyrażeń logicznych. Jedną z lepszych metod zunifikowanego zapisu takich wyrażeń jest zastosowanie techniki masek i wzorów. Polega ona na tym, że słowo maski wybiera z zestawu wszystkich możliwych zmiennych (w naszym przypadku z SZI) tylko te, które występują w wyrazeniu. Otrzymany w ten sposób zbiór wartości 0 i 1 jest porównywany ze wzorem - pomyślny wynik

porównania oznacza spełnienie danego wyrażenia. Aby móc zastosować technikę masek i wzorów do wartościowania wyrażenia logicznego, należy je najpierw przekształcić do alternatywnej postaci normalnej (APN - sumy iloczynów zmiennych prostych lub zanegowanych). Można tego dokonać rozpisując wszystkie iloczyny sum występujące w wyrażeniu. Przy kodowaniu grafu przejść każdej jego krawędzi będzie odpowiadać co najmniej jedna para słów maska-wzór, zaś zbiorowi N krawędzi wychodzących ze stanu S_i będzie odpowiadać struktura danych przedstawiona na rys.1. W strukturze tej po każdej parze maska-wzór umieszczany jest kod stanu następnego S_n, do którego należy przejść z S_i, jeżeli zachodzi:

$$SZI \wedge maska = wzór$$

M	$maska_1$	$wzór_1$	S_{n_1}	$maska_2$	$wzór_2$	S_{n_2}	. . .	$maska_M$	$wzór_M$	S_{n_M}	S_i

Rys.1. Struktura zakodowanych wyrażeń dla pojedynczego stanu
Fig.1. The structure of coded expresions for one state

Dodatkowo na początku umieszcza się liczbę $M \geq N$ określającą ilość par maska-wzór, a na końcu powtarza się kod stanu bieżącego S_i. Zestaw wszystkich zakodowanych w powyższy sposób wyrażeń będzie nazywany w dalszej części artykułu tablicą przejść. W praktycznej realizacji na mikroprocesory ośmiobitowe ustawia się w pary nie całe słowa masek i wzorów, ale ich poszczególne bajty. Można wskazać kilka zalet przyjęcia powyższego zapisu wyrażeń logicznych:

- jest to najzwięźlejszy z możliwych sposób zapisu;
- wymagana jest tylko jedna, wspólna dla wszystkich wyrażeń, w miarę prosta procedura wartościująca, której wynikiem jest kod stanu następnego;
- nie trzeba pisać indywidualnych dla każdego automatu procedur, będących ciągami sprawdzeń logicznych i skoków;
- proces zmiany stanu nie zajmuje dużo czasu maszynowego;
- w prosty sposób można zrealizować pomijanie stanów niestabilnych;
- powyższa metoda może być stosowana nie tylko na poziomie asemblera;
- uzyskuje się dużą łatwość zmian w pracy automatu, co jest istotne zwłaszcza na etapie uruchamiania sterownika.

Drugą grupą informacji niezbędnych do pracy automatu jest tzw. tablica wyjść - zawiera ona wartości wyjściowych zmiennych logicznych dla wszystkich stanów automatu. Na potrzeby sterowania procesami binarno-ciągłymi może również przechowywać inne dane istotne w poszczególnych stanach pracy. Przykładowo: od stanu automatu może zależeć rodzaj stosowanego algorytmu regulacji analogowej, parametry regulatora, wartości zadane itp. W systemie mikroprocesorowym wartości wyjściowych zmiennych logicznych w poszczególnych stanach mogą być po prostu stablicowane i umieszczone w pamięci stałej.

Krótkiego omówienia wymaga również sposób opisu bieżącego stanu automatu w pamięci systemu mikroprocesorowego. Całość związanej z tym informacji

można zgrupować w tzw. stronie opisu automatu (SOA). Jej przykładową strukturę przedstawia rys.2. Podstawowymi jej składnikami są: kod bieżącego stanu automatu, słowo zmiennych istotnych, słowo wyjściowych zmiennych binarnych. Mogą się tam również znaleźć: liczniki czasu przebywania automatu w danym stanie, informacja związana ze stanami alarmowymi (kod przyczyny alarmu, numery (adresy procedur) tzw. algorytmów specjalnych (różnych regulatorów wielkości analogowych, procedur rozruchu i zatrzymania itd.) oraz inne informacje wykorzystywane przez program obsługi automatu.

stan bieżący	SZI	słowo wyjść	timer(y)	informacja dodatkowa

Rys.2. Przykładowa strona opisu automatu
Fig.2. Example of automata description page

Zarezerwowanie w SOA fragmentu pamięci przeznaczonego dla tzw. algorytmów specjalnych ma niebagatelne znaczenie w przypadku sterowania procesami binarno-ciągłymi. W rzeczywistości bardzo często mamy do czynienia z automatyzowaniem urządzenia, w którym obok szeregu elementów dwustanowych, takich jak zawory czy silniki, należy jednocześnie regulować jedną lub więcej wielkości analogowych. W tego typu obiektach niejednokrotnie sposób regulacji ciągłej jest uzależniony od ogólnego stanu urządzenia, fazy jego pracy (włączanie, zatrzymanie itp.), stanu urządzeń współpracujących. Traktując numer wariantu regulacji ciągłej także jako zmienną wyjściową automatu, mamy możliwość zarządzania regulacją wielkosci analogowych z poziomu logicznego automatu. Obszar informacji dodatkowej w SOA może przechowywać szereg wielkości związanych z realizowanymi algorytmami regulacji (wartości wielkości zadanej i mierzonej, błędów regulacji, współczynników regulatora itd.).

Oprócz przedstawionych powyżej struktur danych w skład oprogramowania sterownika mikroprocesorowego wchodzą jeszcze: procedury obsługi urządzeń we/wy (z reguły są to proste procedury), oprogramowanie systemu przerwań oraz procedury generujące SZI (na ogół różne dla różnych automatów). Procedury te analizują szereg wielkości wejściowych tak binarnych, jak i analogowych (jeśli takie występują). W niektórych przypadkach należy także uwzględnić procedury algorytmów specjalnych. Nie wymieniono tu procedur analizujących tablice przejść i dokonujących zmiany stanu automatu, ponieważ mają one charakter niezmiennych modułów bibliotecznych, których jedynymi parametrami są: długość słowa SZI i kod bieżącego stanu. Podobnie rzecz ma się ze standardowymi procedurami regulatorów wielkości analogowych - mogą być one umieszczone w bibliotece i wywoływane z danymi o obwodzie regulacji jako parametrami.

Na typowy cykl obsługi automatu składa się kilka operacji: wczytanie informacji wejściowej, przetworzenie jej na zawartość SZI, porównanie SZI z tablicą przejść, ewentualna zmiana stanu automatu i jego wyjść, a na koniec

realizacja algorytmów specjalnych, jeżeli są one przewidziane w danym stanie. Naturalnym rozwiązaniem jest, aby w/w cykl obsługi był realizowany w przerwaniach zegarowych ze stałą częstotliwością, dopasowaną do szybkości zmian procesu i uwzględniającą najdłuższy możliwy czas trwania jednego cyklu obsługi. Innym wariantem jest uruchamianie programu obsługi automatu jedynie po wystąpieniu zmiany w stanie wejść binarnych, co odpowiada pracy z przerwaniami zewnętrznymi. Z punktu widzenia zarządzania procesami w systemie mikroprocesorowym program nadzorujący może w najprostszym przypadku ograniczać się do uruchamiania w każdym przerwaniu zegarowym kolejno wszystkich programów obsługi automatów - tzw. obsługa synchroniczna. Przyjęta koncepcja realizacji sterowania binarno-ciągłego nie koliduje także z bardziej zaawansowanymi sposobami zarządzania procesami.

Język opisu automatów JODA

Stosowanie, przedstawionego powyżej, sposobu opisu pracy automatu - kodowanie na poziomie asemblera - jest w niektórych przypadkach kłopotliwe. Utrudnienia te wynikają z obecności w procesie kodowania tablic opisujących automat człowieka, którego znamienną cechą jest omylność. Prawdopodobieństwo błędów wzrasta co najmniej liniowo wraz ze zwiększaniem się rozmiaru zadania. Oczywiście, wszelkie błędy kodowania uwidaczniają się w trakcie uruchamiania urządzenia i mogą być następnie skorygowane. Poprawki tablic przejść i wyjść mogą też wynikać ze zmieniającej się w trakcie rozruchu koncepcji pracy automatu. Konieczność przeprowadzania korekt struktur opisujących automat wydłuża czas trwania etapu rozruchu urządzenia - nawet przy założeniu nieomylności osoby kodującej, czas potrzebny na "ręczne" przygotowanie odpowiednich struktur binarnych wydłuża się niewspółmiernie do rozmiaru zadania. Dlatego też powstał zamysł stworzenia narzędzia umożliwiającego automatyczne generowanie tablic przejść i wyjść.

Efektem prac jest Język Opisu Automatów (JODA). Umożliwia on opisanie algorytmu pracy automatu w prostym, zrozumiałym dla człowieka zapisie symbolicznym. Punktem wyjściowym przygotowania danych dla translatora JODA jest analiza automatu, tzn. podział na stany, określenie tablicy wyjść, zdefiniowanie słowa zmiennych istotnych i funkcji przejść. Są to więc czynności, które wykonuje się zwykle przy projektowaniu automatu. Tak przygotowaną informację należy zapisać zgodnie z przyjętą składnią języka, a pozwala ona między innymi na:

- jednoczesny opis wielu automatów (każdy z nich może posiadać do 255 stanów i maksimum 32-bajtowe słowo zmiennych istotnych);
- stosowanie mnemotechnicznych nazw stanów i zmiennych logicznych;
- stosowanie komentarzy objaśniających;
- swobodne budowanie wyrażeń logicznych opisujących funkcje przejść.

Translator ma rozbudowaną diagnostykę danych wejściowych, ze szczególnym uwzględnieniem kontroli wyrażeń logicznych, wskazywaniem miejsca i rodzaju

wykrytych błędów (jeden błąd w danych nie powoduje przerwania procesu sprawdzania ich poprawności). Przy braku błędów w danych wejściowych translator generuje, oprócz raportu z przebiegu translacji, także plik tekstowy zawierający tablice wyjść i przejść, który może być następnie dołączony do głównego programu sterownika mikroprocesorowego. Możliwy jest przy tym wybór programu asemblującego (aktualnie dostępne są dwa warianty: AZ80 dla Z80 i metaasembler C16).

Składnia języka JODA w chwili pisania niniejszego artykułu przedstawia się w zmodyfikowanej notacji Backusa-Naura następująco:

```
opis = "asembler" rodzaj_asemblera ";"
       opis_automatu { ";" opis_automatu } "." .
rodzaj_asemblera = "az80" | "c16"
opis_automatu = nagłówek
                deklaracja_wyjść
                deklaracja_SZI
                deklaracja_stanów
                opis_przejść
                "koniec" .
nagłówek = "automat" tekst ";" .
tekst = ciąg_znaków .
ciąg_znaków = {znak ASCII} .
deklaracja_wyjść = "wyjścia" { bajt_wyjść ";" } .
bajt_wyjść = bajt .
bajt = bit "," bit "," bit "," bit "," bit "," bit "," bit "," bit .
bit = "-" | identyfikator .
identyfikator = litera { litera | cyfra } .
litera = "a" | .. | "z" | "A" | .. | "Z" .
cyfra = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9" .
deklaracja_SZI = "bity" bajt_SZI ";" { bajt_SZI ";" } .
bajt_SZI = bajt .
deklaracja_stanów = "stany" opis_stanu ";" { opis_stanu ";" } .
opis_stanu = identyfikator_stanu ":"
             { bit_wyjściowy { "," bit_wyjściowy }} ";" .
identyfikator_stanu = identyfikator .
bit_wyjściowy = identyfikator .
opis_przejść = "funkcje" przejście { przejście } .
przejście = identyfikator_stanu "-" identyfikator_stanu ":"
            wyrażenie_logiczne ";" .
wyrazenie_logiczne = składnik { "+" składnik } .
składnik = czynnik { "*" czynnik } .
czynnik = identyfikator | "/" czynnik | "(" wyrazenie_logiczne ")" .
komentarz = "{" tekst "}" .
```

Uwagi dodatkowe:

- identyfikatory muszą być obiektami unikalnymi w obrebie danego automatu;

- bity w bajtach wyjściowych oraz SZI podaje się w kolejności od najstarszego do najmłodszego;
- znak "-" przy deklarowaniu tych bitów oznacza nieużywaną przez dany automat pozycję w bajcie;
- znaki "/","*" i "+" to operatory: negacji, iloczynu i sumy logicznej;
- translator dopuszcza także następujące uproszczenia w zapisie:

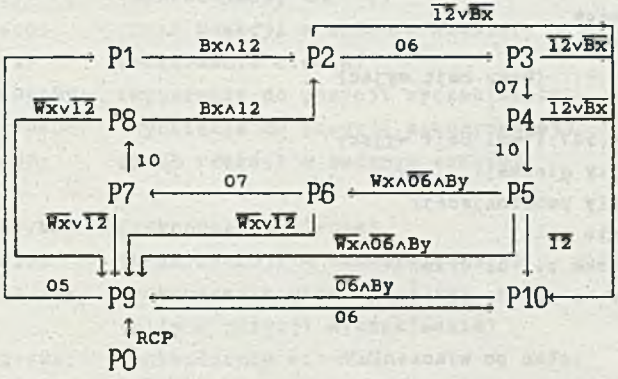
  ")" *czynnik* ≡ ")" "*" *czynnik*

  czynnik "(" ≡ *czynnik* "*" "("

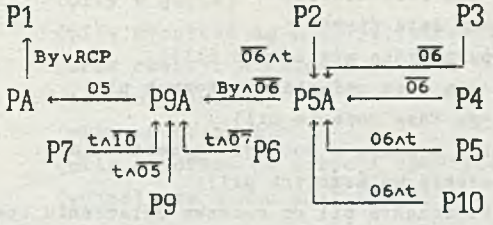
  czynnik "/" ≡ *czynnik* "*" "/"

- komentarz może rozpoczynać się w dowolnym miejscu opisu z wyjątkiem "wnętrza" identyfikatorów i słów kluczowych.

Opisany translator potwierdził już swoją użyteczność w konkretnej realizacji systemu automatyki formatyzerki do cięcia arkuszy sklejki [3]. Poniżej przedstawiony zostanie przykład jego użycia do oprogramowania najprostszego automatu wymienionego urządzenia, a mianowicie zespołu pił tnących. Rysunki 3 i 4 przedstawiają grafy pracy zespołu pił, rysunek 5 demonstruje przyjętą strukturę SZI oraz słowa wyjściowego, a tablica 1 - stany wyjść binarnych w poszczególnych stanach pracy. Na końcu przedstawiona jest treść pliku z danymi wejściowymi automatu PIŁY dla translatora JODA.



Rys.3. Graf przejść automatu PIŁY - stany normalne
Fig.3. Transition graph for automata SAWS - normal states



Rys.4. Graf przejść automatu PIŁY - stany krytyczne
Fig.4. Transition graph for automata SAWS - critical states

a)

we07	we06	we05	—	—	—	RCP	t
—	By	—	Bx	Wx	we12	we11	we10

b)

—	x6	x5	—	—	—	—	—
—	—	—	—	—	—	x9	—
x30	—	—	—	—	—	—	x27

Rys.5. Przyjęta struktura słowa zdarzeń istotnych (a), słowa wyjściowego (b) dla automatu PIŁY
Fig.5. Taken structure of essential events word (a), output word (b) for automata SAWS

Tabela 1

Aktywne wyjścia w poszczególnych stanach automatu PIŁY

Stan automatu	Wykaz aktywnych wyjść binarnych
p0,p1,p10,p5a,pa	
p2	x5,x6,x9
p3,p4	x5,x6,x9,x30
p5,p8	x5,x6
p6,p7	x5,x6,x27
p9,p9a	x27

```
asembler az80
automat pily tnace
wyjscia
-,x6,x5,-,-,-,-,-;  {0-wy bajt wyjsc}
-,-,-,-,-,-,-,x9,-;
x30,-,-,-,-,-,-,-,x27;{2-gi bajt wyjsc}
{x5 - silnik pily glownej;
 x6 - silnik pily podcinajacej;
 x9 - podnoszenie pily;
 x27 - jazda wozka pil do tylu;
 x30 - jazda wozka pil do przodu;}
stany
p0:              {stan po wlaczeniu};
p1:              {stan spoczynkowy};
p2:x5,x6,x9      {wlaczenie i podnoszenie pil};
p3:x5,x6,x9,x30  {1-sza faza ciecia};
p4:x5,x6,x9,x30  {2-ga faza ciecia};
p5:x5,x6         {opuszczanie wlaczonych pil};
p6:x5,x6,x27     {1-sza faza cofania wlaczonych pil};
p7:x5,x6,x27     {2-ga faza cofania pil};
p8:x5,x6         {pily pracuja w pozycji wyczekiwania};
p9:x27           {cofanie wylaczonych pil};
p10:             {opuszczanie pil po recznym wylaczeniu (pedalem)};
p5a:             {awaryjne opuszczanie pil};
p9a:x27          {awaryjne wycofanie pil};
```

```
pa:              {stan awarii automatu};
bity
we07,we06,we05,-,-,-,RCP,t;  {0-wy bajt SZI}
-,By,-,Bx,Wx,we12,we11,we10; {1-szy bajt SZI}
{ RCP-stan klawisza recznego sterowania,
  t-flaga upluniecia zadanego czasu,
  we05,we06,we07,we10,we11,we12-sygnaly z krancowek,
  Bx,By,Wx-wybrane stany pozostalych automatow }
funkcje
p0-p9=RCP;           {start po wlaczeniu zasilania}
p1-p2=Bx*we12;       {start do ciecia}
p2-p5a=t/we06;       {brak reakcji w zadanym czasie}
p2-p10=/we12+/Bx;    {przerwanie ciecia}
p2-p3=we06;          {pily uniesione}
p3-p5a=/we06;        {awaria mechanizmu unoszenia pil}
p3-p10=/we12+/Bx;    {przerwanie ciecia}
p3-p4=we07;
p4-p5a=/we06;        {awaria mechanizmu unoszenia pil}
p4-p10=/we12+/Bx;    {przerwanie ciecia}
p4-p5=we10;          {koniec jazdy tnacej}
p5-p5a=t*we06;       {brak reakcji w zadanym czasie}
p5-p10=/we12;        {przerwanie ciecia}
p5-p6=Wx/we06*By;    {wycofanie do pozycji wyczekiwania}
p5-p9=By/Wx/we06;    {wycofanie do pozycji spoczynkowej}
p6-p9a=t/we07;       {brak reakcji w zadanym czasie}
p6-p7=we07;
p6-p9=/we12+/Wx;     {zakonczenie ciecia}
p7-p9a=t/we10;       {brak reakcji w zadanym czasie}
p7-p9=/we12+/Wx;     {zakonczenie ciecia}
p7-p8=we10;          {pily w pozycji wyczekiwania}
p8-p9=/we12+/Wx;     {zakonczenie ciecia}
p8-p2=Bx*we12;       {start do ciecia}
p9-p9a=t/we05;       {brak reakcji w zadanym czasie}
p9-p10=we06;         {pily w gorze}
p9-p1=we05;          {pily wycofane do pozycji spoczynkowej}
p10-p5a=t*we06;      {brak reakcji w zadanym czasie}
p10-p9=By*/we06      {pily opuszczone}
p5a-p9a=By/we06;     {pily opuszczone}
p9a-pa=we05;         {pily wycofane do pozycji spoczynkowej}
pa-p1=By+RCP;        {wyjscie ze stanu awarii pil}
koniec.
```

Doświadczenia ze stosowaniem translatora JODA przy tworzeniu oprogramowania sterownika formatyzerki wskazują na to, że program ten:

- automatyzuje i przyśpiesza radykalnie proces generacji tablic przejść i wyjść;
- eliminuje ryzyko popełnienia błędu, właściwe "ręcznym" metodom kodowania;
- ułatwia i przyśpiesza ewentualne zmiany algorytmu pracy automatu, co jest istotną właściwością przy uruchamianiu rozwiązań prototypowych.

Podsumowanie

Zaprezentowana koncepcja opisu binarno-ciągłych procesów potwierdziła swe praktyczne znaczenie. Oprócz wspomnianej już formatyzerki, została ona zastosowana do dużego systemu sterowania linią uzysku białka [4]. W systemie tym zadania sterowania binarnego (synchronizacja pracy urządzeń, automatyczny rozruch i wyłączanie, obsługa stanów alarmowych) przeplatały się z koniecznością regulacji wielkości analogowych (regulatory klasyczne i nietypowe). Przyjęcie zunifikowanych form opisu procesu ułatwiło tworzenie oprogramowania sterowników. Podobnie, w przypadku formatyzerki, gdzie występowało jedynie sterowanie binarne, zastosowanie translatora JODA znacząco usprawniło proces uruchomienia urządzenia. Na podstawie zdobytych doświadczeń ze stosowania opisanej metody można stwierdzić, iż ma ona znaczenie praktyczne, szczególnie przy uruchamianiu sterowników programowanych na poziomie asemblera. Może być również z powodzeniem użyta przy programowaniu w językach wyższego poziomu. Dlatego też aktualna wersja translatora JODA nie jest traktowana jako ostateczna - przewiduje się dalszą rozbudowę jego możliwości.

LITERATURA

[1] Niederliński A.: Systemy komputerowe automatyki przemysłowej. WNT, Warszawa 1985.

[2] Siwiński J.: Układy przełączające w automatyce. WNT, Warszawa 1980.

[3] Raport z realizacji pracy: Mikroprocesorowy sterownik formatyzerki, dla Białostockich Zakładów Przemysłu Sklejek, Białystok 1991.

[4] Leszczyński J.,Grodzki L.: Oprogramowanie systemu sterowania doświadczalną linią produkcji kazeiny. Materiały XI KKA, Białystok 1991.

Recenzent: Prof.dr h.inż. Franciszek Marecki
Wpłynęło do Redakcji do 30.04.1992 r.

Abstract: The paper presents one of the possible method of binary process description for microprocessor controller. It is based on treatment of all processes (objects, devices), as one or more automatas co-operating each with other. The work of such automatas can be described by transition graphs and output tables - basic ideas of automata theory. The author proposes standardized treatment of the automatas, served by one controller, for simplification of the system software. The standard description elements are: methods of coding transition graphs and output table, data structure containing information about the current state of device and a part of the system software procedures. Since coding process for bigger automata is not easy and fast in real applications, the special description language and translator have been worked out. The paper presents this language and its translator possibilities. The experience in usage of the mentioned language shows, that it is quite usefull. There is an example of usage description language translator in paper, too.