Kompaktowy sterownik programowalny

Chmiel Mirosław
Hrynkiewicz Edward
Stefański Łukasz*

Wszystkie firmy, liczące się na rynku sterowników, mają w swojej ofercie przynajmniej jednego typu niewielki sterownik kompaktowy, zwykle wyposażony we własny interfejs użytkownika umożliwiający programowanie bez konieczności korzystania z komputera osobistego. W artykule omówiono projekt takiego urządzenia.

Od chwili pojawienia się pierwszych tego typu urządzeń w Instytucie Elektroniki Politechniki Śląskiej rozpoczęto również prace mające na celu opracowanie takiego sterownika. W ich wyniku powstał projekt oparty na mikrokontrolerze 8-bitowym z rodziny MCS '51, programowalny za pomocą języka LD (*Ladder Diagram*) z niewielkiej klawiatury na płycie czołowej [9, 14]. Ze względu na prekursorski charakter prac mających na celu sprawdzenie ogólnych koncepcji, sterownik ten miał jedynie wejścia i wyjścia dyskretne, a w skład zaimplementowanego języka wchodziły tylko najprostsze funkcje logiczne. Kolejnym krokiem miało być zaprojektowanie sterownika kompaktowego wyposażonego w mikrokontroler 16-bitowy oraz wejścia i wyjścia analogowe. Niniejszy artykuł dotyczy projektu takiego urządzenia [15]. W powstałym sterowniku wykorzystano część koncepcji opracowanych przy okazji projektu pierwszego ze wspomnianych wyżej urządzeń. Jest więc on również programowany za pomocą języka LD przy wykorzystaniu 6-przyciskowej klawiatury i wyświetlacza LCD. Skanowanie schematu drabinkowego odbywa się kolumnami, przy czym procedura analizująca nie dokonuje kompilacji programu (głównie ze względu na ograniczoną pamięć RAM oraz fakt, iż w wersji skompilowanej program użytkownika zajmowałby dużo większy obszar niż w przyjętym rozwiązaniu), a jedynie pewnego rodzaju interpretację umożliwiającą jego szybsze wykonanie. Autorzy postawili sobie za cel stworzenie sterownika szybszego od pierwowzoru. W tym celu zastosowano 16-bitowy mikrokontroler z rodziny MCS '196 oraz starano się jak najbardziej zoptymalizować przyjęty sposób wykonywania programu użytkownika. Założeniem było także zwiększenie zakresu zastosowań sterownika poprzez wyposażenie go w wejścia i wyjścia analogowe, zwiększenie liczby dostępnych elementów języka oraz obszaru pamięci przeznaczonego na schemat drabinkowy. Ponadto zdecydowano się na dalsze zwiększenie autonomii urządzenia, czego wynikiem było dodanie wymiennego modułu pamięciowego umożliwiającego przechowywanie do 50 programów użytkownika. Priorytetem było także lepsze rozwiązanie metod programowania sterownika, tak aby był on bardziej przyjazny użytkownikowi, co jest szczególnie trudne przy niewielkich rozmiarach wyświetlacza oraz kilkuklawiszowej klawiaturze.

* *dr inż. Mirosław Chmiel, dr inż. hab. Edward Hrynkiewicz, prof. Politechniki. Śląskiej, mgr inż. Łukasz Stefański - Politechnika Śląska w Gliwicach, Instytut Elektroniki*

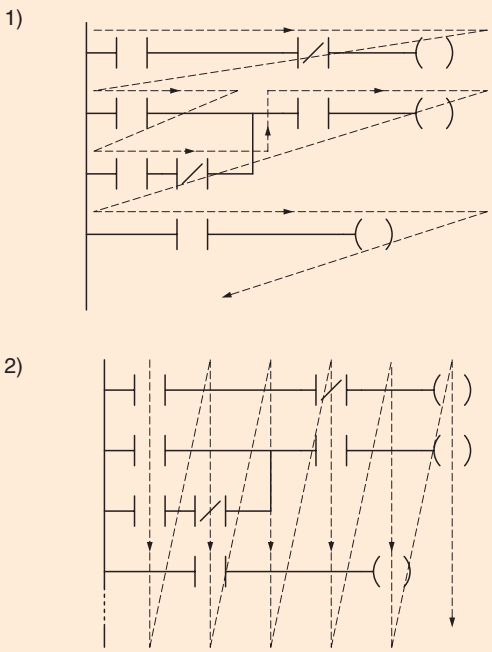
Wybór sposobu analizy schematu drabinkowego

Sterowniki PLC wykonują program aplikacyjny w sposób szeregowo-cykliczny. Metoda ta polega na ciągłym wykonywaniu zespołu określonych czynności tworzących cykl programowy (*Program Sweep*). Podstawowymi etapami tego cyklu są: odczyt stanów wejść do odpowiadającego im obszaru w pamięci danych sterownika zwanego obszarem odwzorowania wejść (PII – *Process Image Input*); realizacja programu użytkownika i zapis wyników jego wykonania do obszaru odwzorowania wyjść (PIQ – *Process Image Output*) oraz wysłanie danych wyjściowych z obszaru odwzorowania do wyjść fizycznych [11, 12, 13]. Sposób wykonania programu zależy od rodzaju języka, w którym napisano program aplikacyjny. Języki graficzne wykorzystują ideę przepływu odpowiedniej wielkości przez kolejne obwody reprezentujące pewną strategię postępowania. W języku LD stany elementów łączących oznaczanych jako ON lub OFF – odpowiednio do wartości logicznych 1 i 0 – są analogią do przepływu prądu w systemach sterowania opartych na przekaźnikach elektromechanicznych. Ponieważ struktura elementów graficznych tworzących schemat drabinkowy jest rozbudowywana zarówno w kierunku poziomym jak i pionowym obszaru edycji, konieczne jest opracowanie sposobu jego przeglądania i realizowania (*Scanning*). Analizując materiały udostępniane przez producentów sterowników PLC, można zauważyć, że praktycznie przyjęły się dwie metody: skanowanie wierszami [2, 5] oraz skanowanie kolumnami [3, 12]. Pierwsza z nich polega na przeglądaniu schematu w kierunku poziomym – od lewej listwy prądowej w prawo, aż do ostatniego elementu w wierszu. W ten sposób są przeglądane kolejno wszystkie wiersze schematu. W drugiej metodzie stan wszystkich elementów schematu jest sprawdzany od elementu położonego najwyżej po położony najniżej w skrajnej lewej kolumnie,

by w kolejnym kroku przesuwając się w prawo, powtarzać czynność przeglądania od góry ku dołowi. Obydwie metody są przedstawione na rys. 1 na przykładowym fragmencie schematu drabinkowego. Porównując zaznaczoną w obu przypadkach kolejność skanowania poszczególnych elementów schematu, można zauważyć, że przy analizie poziomej algorytm „nawigacji" ma nieco bardziej skomplikowaną formę. Wynika to z tego, że oprócz połączeń poziomych (a więc zgodnych z kierunkiem skanowania) schemat zawiera także połączenia pionowe. Ponieważ połączenie takie spełnia funkcję sumy logicznej stanów wszystkich gałęzi podłączonych do niego z lewej strony, konieczne jest ich wyznaczenie przed obsługą połączenia pionowego. Ponadto, można się spodziewać, że dłuższe połączenia pionowe (łączące więcej niż dwa szczeble schematu drabinkowego), a także łączące szczeble niepodłączone bezpośrednio do listwy prądowej, spowodują dalsze odstępstwa od poziomego kierunku skanowania i skomplikowanie programu analizy [8, 10].

Przedstawiony problem obsługi połączeń pionowych nie występuje przy skanowaniu kolumnami. Jest to spowodowane tym, że stany wszystkich gałęzi znajdujących się po lewej stronie połączenia pionowego są wyznaczane podczas analizy poprzedniej kolumny. Niewątpliwą zaletą analizy poziomej jest natomiast to, że teoretycznie wystarcza jedna zmienna do zapamiętywania stanów analizowanych wierszy schematu (lub nawet jeden znacznik bitowy, jeżeli mikrokontroler ma obszary adresowane bitowo), ponieważ przed rozpoczęciem skanowania kolejnego wiersza stan wiersza poprzedniego jest przekazywany zmiennej skojarzonej z cewką znajdującą się na jego końcu. Oczywiście przy omówionych powyżej odstępstwach od poziomego kierunku skanowania konieczne są dodatkowe zmienne umożliwiające zapamiętanie stanów wierszy, których analiza została przerwana. Nie ulega jednak wątpliwości, że liczba wykorzystywanych zmiennych będzie dużo mniejsza niż przy skanowaniu kolumnami, ponieważ wówczas z każdym wierszem musi być skojarzona odrębna zmienna określająca jego status. Za wadę analizy w kierunku pionowym można uważać także efekt kolumnowy polegający na tym, że cewka znajdująca się w kolumnie położonej bliżej lewej szyny prądowej zostanie obsłużona wcześniej niż cewka znajdująca się w kolumnie położonej bardziej na prawo, nawet jeżeli szczebel schematu z pierwszą cewką znajduje się poniżej szczebla z cewką drugą (w przedstawionym przykładzie z rys. 1 cewka położona w ostatnim wierszu zostanie obsłużona przed pozostałymi cewkami).

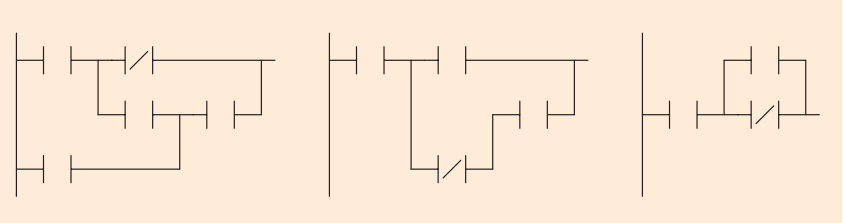
Biorąc pod uwagę wady i zalety przedstawionych metod skanowania schematu drabinkowego, zdecydowano się ostatecznie na zastosowanie w projektowanym sterowniku analizy kolumnami. Uznano, że technika ta umożliwia realizację algorytmu analizy, który będzie dużo prostszy (a więc prawdopodobnie także i szybszy) niż przy skanowaniu w kierunku poziomym, a jednocześnie będzie umożliwiał większą wszechstronność w budowaniu struktury połączeń schematu drabinkowego. Wszechstronność ta jest wynikiem dużo większej dowolności wstawiania połączeń pionowych – stosunkowo łatwa jest analiza i obsługa połączeń pionowych występujących w strukturach takich jak przedstawione na rys. 2. (przedstawione fragmenty schematu drabinkowego nie zostałyby poprawnie obsłużone w większości sterowników dokonujących analizy wierszami).



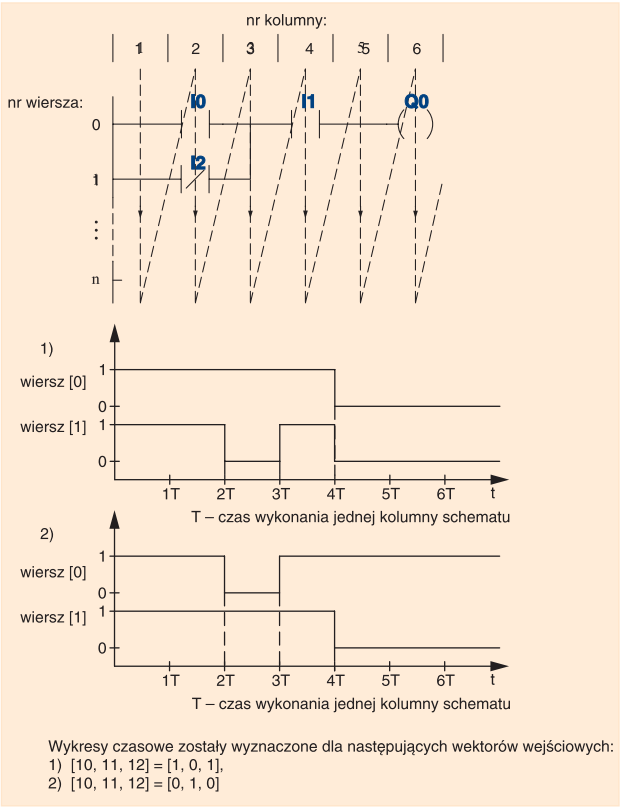
Rys. 1. Sposoby analizy schematu drabinkowego: 1) wierszami, 2) kolumnami

Ze względu na charakterystykę projektowanego sterownika i zastosowanie tego typu urządzeń, pamięć przeznaczona na program użytkownika nie będzie miała dużych rozmiarów, a więc wada wybranego sposobu analizy związana z koniecznością stosowania dużej liczby zmiennych statusowych wierszy nie będzie stanowić dużego utrudnienia. Z kolei efektu kolumnowego można uniknąć, jeżeli użytkownik wprowadza schemat ze świadomością sposobu jego późniejszej analizy oraz możliwości wystąpienia takiego zjawiska.

Stosując przyjętą koncepcję skanowania schematu drabinkowego, opracowano algorytm umożliwiający wykonanie programu użytkownika. Jest to rodzaj interpretacji polegający na rozpoznawaniu kolejnych elementów schematu (zgodnie z przyjętym kierunkiem



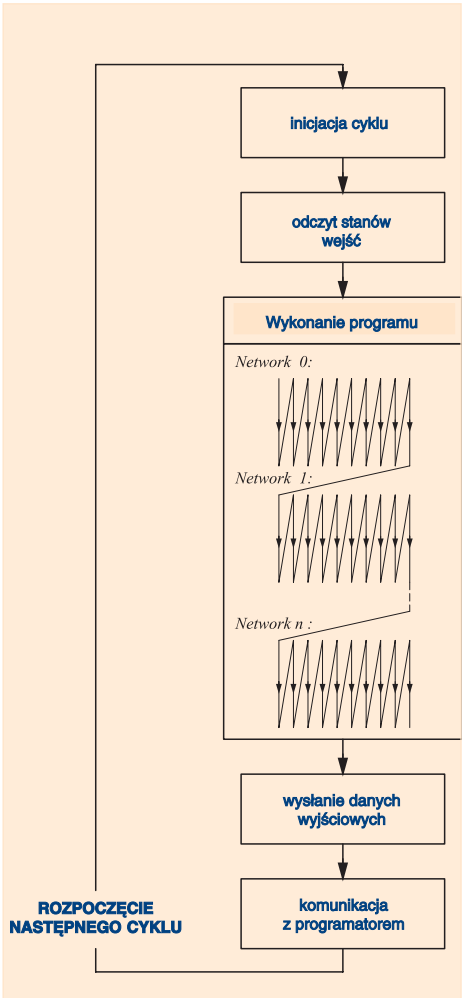
Rys. 2. Przykłady struktur obsługiwanych poprawnie przy kolumnowej analizie schematu drabinkowego



Rys. 3. Ilustracja zmian wartości zmiennych odpowiadających za stan wierszy podczas analizy schematu drabinkowego

analizy) i wywoływaniu procedur obsługi tych elementów. Procedura tego typu, odpowiednio do funkcji wykonywanej przez konkretny element, modyfikuje zmienną statusową wiersza, w którym się on znajduje. Zmienne statusowe wierszy są ustawiane na początku każdego cyklu, co oznacza podpięcie do listwy prądowej elementów umieszczonych w pierwszej kolumnie. Rys. 3 ilustruje zmiany zmiennych statusowych wierszy schematu przy analizie pokazanego fragmentu schematu drabinkowego. Wykresy czasowe opracowano, dokonując pewnego uproszczenia polegającego na założeniu, że czasy obsługi poszczególnych kolumn przedstawionego fragmentu schematu są równe.

Przedstawione oznaczenia zmiennych statusowych wierszy schematu drabinkowego wynikają z tego, że będą one umieszczone w tablicy o indeksie odpowiadającym numerowi wiersza. Aby skrócić proces analizy, zdecydowano się na podział programu użytkownika na segmenty. Wówczas, jeżeli dany segment jest pusty lub żaden ze znajdujących się w nim elementów nie jest podłączony do listwy prądowej, można go pominąć w procesie analizy. Podział na segmenty ma również na celu ułatwienie obsługi sterownika, ponieważ – dzięki realizacji trybu umożliwiającego przeskok o segment – pozwoli na szybkie poruszanie się po schemacie. Na rys. 4 przedstawiono cykl pracy sterownika wraz z zaznaczonym sposobem skanowania schematu drabinkowego.



Rys. 4. Cykl pracy sterownika

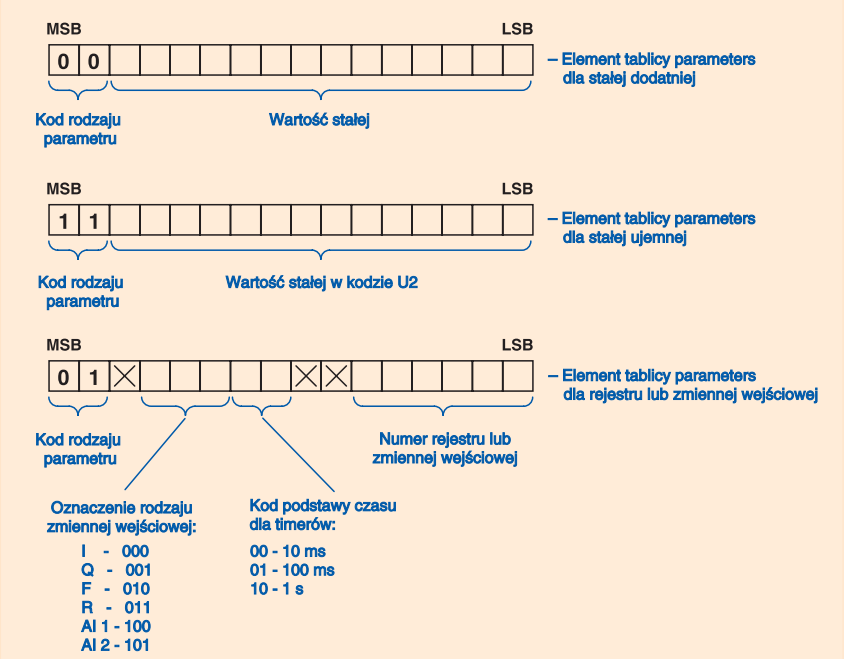
Wprowadzanie i przechowywanie programu użytkownika

Program użytkownika w postaci schematu drabinkowego jest wprowadzany interfejsem złożonym z wyświetlacza alfanumerycznego o rozmiarze 4x20 oraz klawiatury, w skład której wchodzi: 6 przycisków, 4 klawisze kursorów, klawisz akceptacji – OK oraz rezygnacji – ESC. Ekran jest zorganizowany w taki sposób, że dwie skrajne kolumny z lewej zajmują numery poszczególnych wierszy, następnie znajduje się obszar wprowadzania schematu złożony z 10 kolumn oraz listwy prądowej, natomiast pozostałe kolumny z prawej strony są używane do wprowadzania i późniejszego wyświetlania parametrów elementów schematu. Obszar użyteczny został podzielony na segmenty ułatwiające programowanie i poruszanie się po ekranie.

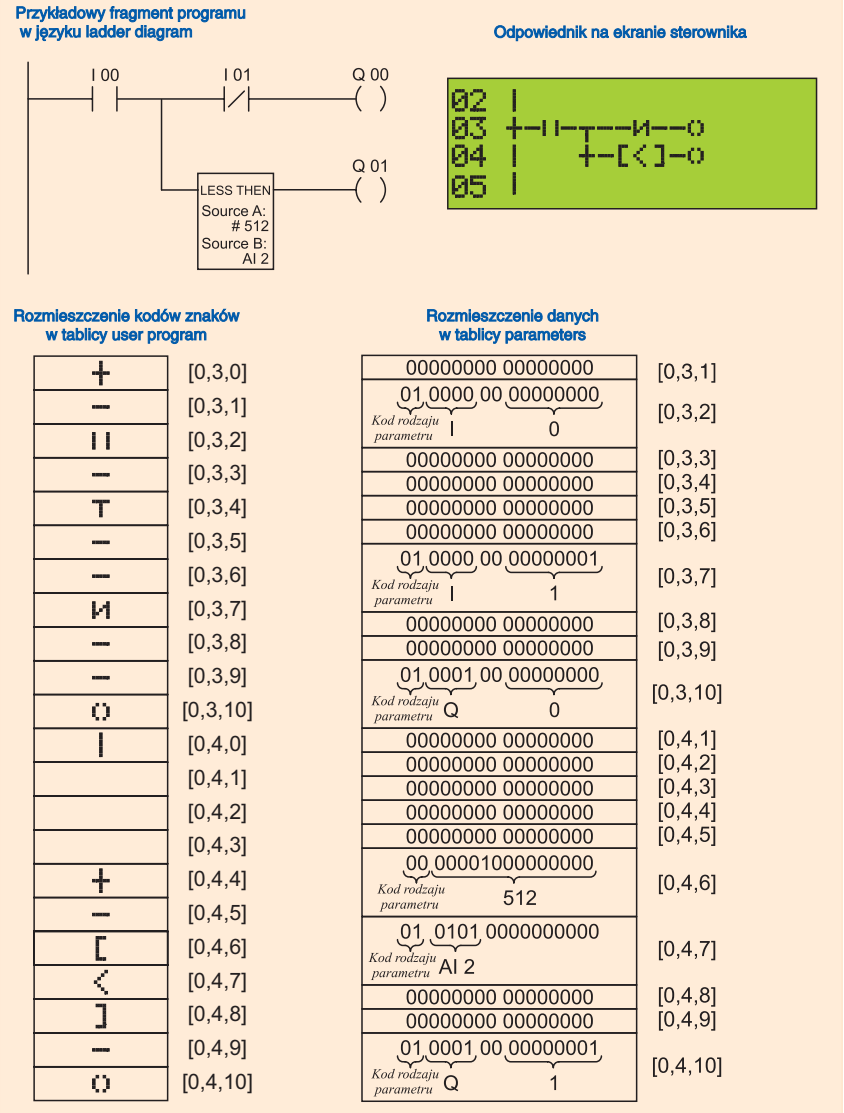
Do przechowywania programu użytkownika stworzono dwie tablice o nazwach *user_program* oraz *parameters*. Pierwsza z tych tablic zawiera kody znaków wyświetlanych na ekranie tworzących schemat drabinkowy (tablica zmiennych typu char), a jej indeksy mają zakres identyczny ze zmiennymi określającymi położenie kursora w polu edycji schematu (numer segmentu, wiersza oraz kolumny). Druga tablica jest zorganizowana w sposób identyczny, z tym że jej zawartość stanowią wartości typu *integer* będące parametrami elementów schematu. Element tablicy parameters może mieć jedną z trzech postaci, w zależności od tego jaki rodzaj parametru jest w nim przechowywany. Rodzaje elementów tej tablicy i znaczenie poszczególnych pól przedstawia rys. 5.

Dwa najstarsze bity stanowią kod oznaczający czy parametr zawiera stałą, czy numer skojarzonej z elementem schematu zmiennej logicznej lub numer rejestru. Ponadto w komórce zawierającej adres akumulatora (numer rejestru wybranego na akumulator) dla timera znajduje się dodatkowa informacja o podstawie czasu.

Procedura obsługująca ruch kursora na ekranie wyświetlacza odczytuje zmienną, do której wczytywany jest stan klawiatury, a następnie, w zależności od rodzaju naciśniętego klawisza, inkrementuje lub dekrementuje zmienne odpowiedzialne za położenie kursora będące jednocześnie indeksami tablic *user_program* i *parameters*. Każdemu z pól wyświetlacza przypisuje się po jednej komórce tych tablic. Elementy schematu drabinkowego w opisywanym sterowniku zajmują różną liczbę pól wyświetlacza (od 1 do 4) w zależności od rodzaju elementu. Cecha ta sprawia, że przypisanie jednemu polu tylko po jednej komórce tablic zawierających program użytkownika jest wystarczające. Na przykład element realizujący dodawanie arytmetyczne zajmuje na ekranie wyświetlacza trzy pola: dwa nawiasy kwadratowe i znak plus ([+]), a więc dla opisu tego elementu mamy do dyspozycji po trzy komórki każdej z tablic. Tablica *user_program* będzie zawierać wszystkie wyżej wymienione znaki ('[', '+' oraz ']') w postaci kodów ASCII natomiast tablica *parameters* dwa argumenty dodawania oraz miejsce przeznaczenia wyniku, przy czym argumenty mogą mieć każdą z trzech opisywanych wyżej postaci komórek tablicy *parameters* (mogą mieć postać stałych lub być przekazane przez rejestr). Rys. 6 przedstawia przykładowy fragment programu oraz jego reprezentacje w tablicach *user_program* i *parameters*. Jak można zauważyć na znajdującej się po prawej stronie ilustracji, reprezentującej wygląd przedstawionego fragmentu programu na ekranie sterownika, przy poszczególnych elementach schematu drabinkowego nie ma opisów określających rodzaj i numer zmiennych skojarzonych z danym elementem oraz innych parametrów, takich jak numery rejestrów czy wartości stałych. Wszystkie te dane pojawiają się w menu bocznym w chwili ustawienia kursora na elemencie, który one opisują. Napis 'AI 2' w bloku operatora relacji oznacza drugie wejście analogowe sterownika (źródło drugiego argumentu).



Rys. 5. Rodzaje elementów tablicy *parameters*



Rys. 6. Przykładowy fragment programu oraz jego reprezentacje w tablicach *user_program* i *parameters*

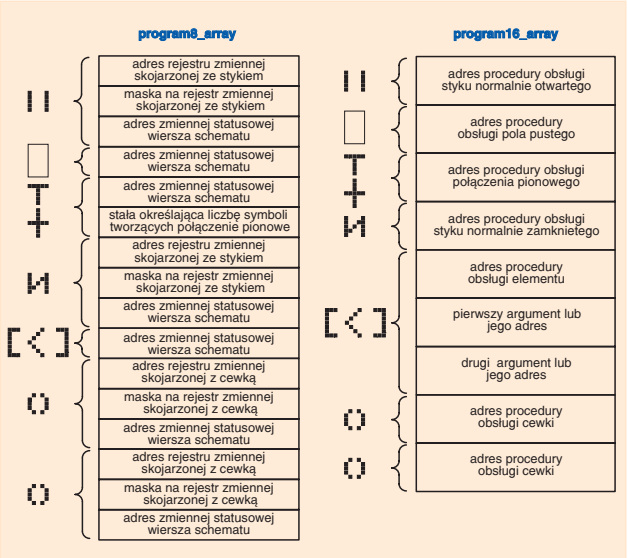
Wykonywanie programu

Sterownik wykonuje program użytkownika w sposób szeregowo-cykliczny. Początkowo do obszaru odwzorowania wejść są odczytywane stany wejść dyskretnych, następnie są realizowane funkcje zapisane za pomocą schematu drabinkowego przez użytkownika, a ich wyniki zapisywane w obszarze odwzorowania wyjść, z którego są przepisywane do wyjść dyskretnych pod koniec cyklu programowego. Nieco inaczej są obsługiwane wejścia i wyjścia analogowe, które nie mają obszarów odwzorowań w pamięci sterownika. Odczyt wejścia analogowego następuje bezpośrednio w trakcie obsługi elementu realizującego ten odczyt. Podobnie wygląda sytuacja przy zapisie do wyjścia analogowego, który następuje bezpośrednio, bez oczekiwania na koniec cyklu programowego. Taki sposób obsługi we/wy analogowych był podyktowany głównie stosunkowo długim czasem konwersji przetworników, o który wydłużony byłby każdy cykl pracy, nawet jeżeli program nie korzysta z we/wy analogowych lub w danym cyklu nie są one obsługiwane.

Sposób wykonywania programu użytkownika przez sterownik polega na modyfikacji wartości zmiennych przydzielonych do każdego wiersza schematu drabinkowego w zależności od wyników funkcji realizowanych przez poszczególne elementy wchodzące w skład programu. Zmienne statusowe wierszy są ustawiane na początku każdego cyklu programowego.

Sterownik wyposażono w dwa tryby wykonywania programu użytkownika: z podglądaniem stanu gałęzi lub bez (docelowy). Obydwa tryby pracy sterownika korzystają z tych samych procedur obsługi poszczególnych elementów schematu drabinkowego. Procedury korzystają z kolei z danych zawartych w dwóch specjalnie utworzonych do tego celu jednoindeksowych tablicach: *program8_array* (o wartościach typu *char*) i *program16_array* (o wartościach typu *integer*), w których zostają one, po odpowiednich przekształceniach i uporządkowaniu, umieszczane z tablic *user_program* i *parameters*. Ten proces odwzorowania ma miejsce przy wejściu do każdego z trybów pracy sterownika. W wyniku przeprowadzenia takiego przekształcenia uzyskuje się znaczne zwiększenie szybkości pracy sterownika z kilku powodów. Po pierwsze, korzystanie z tablic, w których znajdują się dane jednoindeksowe znacznie przyspiesza pracę w porównaniu do korzystania z tablic *user_program* i *parameters*, do których odwołanie odbywa się poprzez trzy indeksy i podczas pobierania danych do procesu każdorazowo mikrokontroler oblicza sumę trzech indeksów. Po drugie, w procesie przekształcania jest możliwe dokonanie pewnych obliczeń, które mogą zostać wykonane jednorazowo, a przy korzystaniu z tablic *user_program* i *parameters* musiałyby być wykonywane wielokrotnie (na przykład obliczenie adresu rejestru czy uzyskanie z odpowiedniej komórki tablicy parameters podstawy czasu dla licznika). Wykonując proces przekształcenia, można także wykluczyć z procesu obsługi niektóre z rozpoznanych symboli widniejących na ekranie. Takimi symbolami są przykładowo połączenia poziome, których obsługa nie jest konieczna (ale tylko dla trybu bez wizualizacji stanu połączeń, gdyż dla trybu z wizualizacją konieczne jest umieszczenie w tablicach odwzorowania miejsca położenia symbolu połączenia poziomego na ekranie), gdyż nie zmieniają one stanu zmiennych określających stan wierszy schematu. Podobnie jest w przypadku większości pól pustych. Zadaniem pola pustego na schemacie drabinkowym jest wyzerowanie zmiennej statusowej rzędu, w którym to pole się znajduje. Wyzerowanie tej zmiennej ma jednak sens tylko wtedy, gdy jakikolwiek inny element schematu znajduje się w tym rzędzie, na prawo od pola pustego. W przeciwnym wypadku jest to operacja niepotrzebna i powodująca wydłużenie czasu cyklu, gdyż żaden element nie będzie sprawdzał stanu wyzerowanej zmiennej statusowej rzędu, a więc jej stan jest obojętny. Ponadto nie ma sensu wielokrotne zerowanie zmiennej opisującej stan danego wiersza, gdy kilka pól pustych następuje po sobie. Po natrafieniu na pole puste wystarczy sprawdzać, czy kolejny element w wierszu jest elementem innym niż pole puste i tylko w takim wypadku dokonywać odwzorowania. Kolejnym czynnikiem umożliwiającym przyspieszenie wykonywania programu jest możliwość umieszczenia w tablicy *program16_array* adresów kolejno wykonywanych procedur obsługi poszczególnych elementów, ponieważ proces rozpoznawania elementu poprzez jego porównywanie z wszystkimi możliwymi kodami jest dokonywany w procedurze odwzorowującej. Ma to szczególne znaczenie w przypadku symboli elementów zajmujących kilka pól wyświetlacza, ponieważ proces ich rozpoznawania trwa dłużej. Zawartość tablic odwzorowania schematu dla przedstawionego w poprzednim punkcie przykładu jest przedstawiona na rys. 7 (dla trybu pracy bez wizualizacji).

W przypadku odwzorowania dla trybu z wizualizacją stanu gałęzi powyższe tablice zawierałyby dodatkowo informacje o położeniu wszystkich symboli wchodzących w skład gałęzi schematu drabinkowego (zarówno poziomych jak i pionowych). Tryb ten jest nieco wol-



Rys. 7. Zawartość tablic odwzorowania dla przykładowego programu z rys. 6

niejszy ze względu na konieczność obsługi klawiatury, odświeżanie ekranu i dodatkową obsługę połączeń poziomych wraz ze wspomnianą wizualizacją stanu wszystkich połączeń.

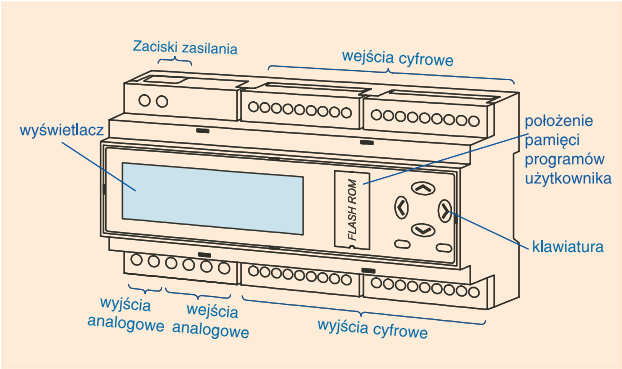
W procedurze analizy schematu tablice *user_program* i *parameters* są skanowane kolumnami zgodnie z wcześniej przyjętą koncepcją. Każdy segment jest skanowany oddzielnie, przy czym początkowo jest sprawdzane, czy w danym segmencie jakikolwiek element jest podpięty do listwy prądowej (w tym celu sprawdzane jest czy w pierwszej kolumnie tablicy *user_program* – o pierwszym indeksie odpowiadającym badanemu networkowi – występuje przynajmniej jeden znak +) i jeżeli tak, to jest dokonywane odwzorowanie tego networku. W przeciwnym razie badany segment jest pomijany w procesie skanowania, co skraca czas konwersji całego programu. Po przekształceniu całego schematu do tablicy *program16_array* jest wstawiany adres procedury kończącej program, procedura ta jest odpowiedzialna za wykonanie m.in. takich operacji jak: wysłanie wyników do wyjść, pobranie do obszaru odwzorowania stanów wejść, ustawienie znaczników stanu wierszy i kilka innych zależnie od trybu pracy.

Po wypełnieniu tablic odwzorowania w wyżej przedstawiony sposób, wykonanie programu użytkownika sprowadza się jedynie do pobierania kolejnych adresów procedur obsługi poszczególnych elementów schematu drabinkowego i ich wywoływaniu. Adresy początków tablic odwzorowania są przypisywane dwóm odpowiadającym im zmiennym wskaźnikowym. Ponieważ w trakcie wykonywania każdej z procedur zmienne wskaźnikowe są na bieżąco inkrementowane, aby pobrać wszystkie potrzebne dane z tablic *program8_array* i *program16_array*, po wyjściu z każdej z procedur zmienne te wskazują komórki, od których zaczynają się bloki danych dla kolejnego elementu.

Podsumowanie i wnioski

W wyniku przeprowadzonych prac powstał sterownik o bardzo prostej i wygodnej obsłudze, przy jednoczesnym znacznym rozszerzeniu możliwości w stosunku do pierwowzoru wspomnianego we wstępie. Wygląd sterownika przedstawia schematycznie rys. 8.

Starano się tak rozwiązać wprowadzanie programu użytkownika, aby było ono sprawne i szybkie, mimo prostoty interfejsu użytkownika. Z tego powodu wprowadzono specjalne tryby rysowania i mazania schematu drabinkowego, umożliwiające łatwe wstawianie i usuwanie całych szczebli schematu. Tryby te wybiera się z menu elementów; pozostają one aktywne aż do naciśnięcia klawisza ESC lub wyboru innego elementu schematu. Obsługa trybu rysowania połączeń polega jedynie na naciskaniu klawiszy kursorów, co powoduje ruch znacznika na ekranie z automatycznym wstawianiem struktury połączeń. Jednocześnie jest dokonywane automatyczne podłączanie szczebli schematu do listwy prądowej oraz modyfikacja symboli łączników znajdujących się na skrzyżowaniach gałęzi poziomych i pionowych. Tryb mazania schematu działa w sposób analogiczny. Poruszanie się po ekranie ułatwia wprowadzona repetycja oraz tryb przeskoku o segment. Dodatkowo sygnalizacja akustyczna umożliwia orientację co do prawidłowości wykonywanych czynności. Dobrym rozwiązaniem okazało się także wprowadzenie rejestrów uniwersalnych, które spełniają jednocześnie funkcję akumulatorów dla timerów i liczników (dzięki temu, to od użytkownika zależy rozdzielenie rejestrów pomiędzy liczniki, układy czasowe i pozostałe elementy). Sterownik jest wyposażony ponadto w tryb symulacji schematu drabinkowego poprzez wizualizację stanów połączeń, a także w opcję umożliwiającą podgląd wybranego rejestru uniwersalnego bez przerywania pracy urządzenia (w menu parametrów).



Rys. 8. Wygląd sterownika

Sterownik został wyposażony w dwa wejścia analogowe i jedno wyjście analogowe o rozdzielczości 10 bitów oraz zakresie napięć od 0 do 10 V. Zarówno wejścia jak i wyjścia są izolowane galwanicznie. Izolacja została zrealizowana po stronie analogowej za pomocą wzmacniaczy izolacyjnych. Ponieważ zastosowane wzmacniacze wymagają zasilania po obu stronach bariery, wykorzystano do tego celu izolowane przetwornice impulsowe. Taka budowa obwodów we/wy analogowych jest bardzo wygodna dla użytkownika, gdyż nie wymagają one zasilania po stronie obiektowej.

Liczba elementów schematu drabinkowego została rozszerzona do 24 (w pierwowzorze było ich tylko 8). Oczywiście część z tych elementów, jak np. cewki ustawiające czy zerujące są możliwe do zrealizowania za pomocą podstawowych elementów języka (styków i cewek standardowych), jednak ich dodanie umożliwia oszczędność miejsca na ekranie, zwiększa przejrzystość schematu i wygodę programowania. W tablicy 1 przedstawiono wszystkie elementy, których można używać, tworząc program sterowania.

Szybkość sterownika jest porównywalna z urządzeniami podobnej klasy będącymi na rynku [4, 6, 7]. Czas wykonania 1000 instrukcji binarnych wynosi 15 ms, a 1000 instrukcji mieszanych (75 % binarnych oraz 25 % słownych) – 24 ms. Przy tego typu budowie systemu (standardowy system mikroprocesorowy) ciężko jest osiągnąć czasy dużo krótsze. Rozwiązaniem tego problemu mogłaby być kompilacja programu i ładowanie go do pamięci w wersji skompilowanej. Oczywiście zwiększenie szybkości można osiągnąć poprzez zastosowanie szybszego (ale i droższego) mikrokontrolera,

Tabela 1. Wykaz elementów schematu drabinkowego

Nazwa elementu	Symbol w schemacie drabinkowym	Opis
Połączenie poziome	—	Połączenie poziome przekazuje stan elementu znajdującego się bezpośrednio po lewej stronie do elementu po stronie prawej.
Połączenie pionowe	| + T	Stan połączenia pionowego odpowiada sumie logicznej stanów połączeń poziomych występujących po jego lewej stronie i jest przekazywany połączeniom poziomym dołączonym po prawej stronie. Połączenie pionowe stanowi odpowiednią kombinację symboli pokazanych w sąsiedniej kolumnie.
Styk normalnie otwarty	| |	Stan połączenia z lewej strony styku jest przenoszony na prawą stronę, jeżeli skojarzona zmienna logiczna ma wartość 1. W przeciwnym razie prawe połączenie jest w stanie OFF.
Styk normalnie zamknięty	|/|	Stan połączenia z lewej strony styku jest przenoszony na prawą stronę, jeżeli skojarzona zmienna logiczna ma wartość 0. W przeciwnym razie prawe połączenie jest w stanie OFF.
Styk reagujący na zbocze narastające	|P|	Połączenie z prawej strony styku jest w stanie ON w czasie jednego wykonania, jeśli połączenie z lewej strony jest w stanie ON a skojarzona zmienna logiczna zmieniła wartość z 0 na 1. Poza tym stan połączenia z prawej strony jest OFF.
Styk reagujący na zbocze opadające	|N|	Połączenie z prawej strony styku jest w stanie ON w czasie jednego wykonania, jeśli połączenie z lewej strony jest w stanie ON a skojarzona zmienna logiczna zmieniła wartość z 1 na 0. Poza tym stan połączenia z prawej strony jest OFF.
Cewka	( )	Stan połączenia z lewej strony cewki jest przenoszony na prawą stronę i zapamiętywany w skojarzonej zmiennej logicznej.
Cewka ustawiająca	(S)	Skojarzona zmienna logiczna przyjmuje wartość 1, jeżeli połączenie z lewej strony jest w stanie ON i nie zmieni się aż do chwili wyzerowania przez cewką kasującą.
Cewka kasująca	(R)	Skojarzona zmienna logiczna przyjmuje wartość 0, gdy połączenie z lewej strony jest w stanie ON i nie zmieni się do chwili wyzerowania przez cewką ustawiającą.
Moduł czasowy załączający z opóźnieniem	TN – wejście ustawiające rT – wejście zerujące	Element odmierza zaprogramowany czas, gdy wynik warunku uruchomienia zmieni wartość z 0 na 1 oraz warunek zerowania nie jest spełniony. Wyjście modułu czasowego jest w stanie ON, jeżeli upłynął zaprogramowany czas, warunek uruchomienia jest spełniony oraz nie jest spełniony warunek zerowania.
Moduł czasowy wyłączający po wyłączeniu	TF – wejście ustawiające rT – wejście zerujące	Element odmierza zaprogramowany czas, gdy wynik warunku uruchomienia zmieni wartość z 1 na 0 oraz warunek zerowania nie jest spełniony. Wyjście modułu czasowego jest w stanie ON, jeżeli warunek uruchomienia jest spełniony lub jeżeli nie jest spełniony warunek zerowania i zaprogramowany czas nie upłynął.
Licznik dodający	UC – wejście zliczające rC – wejście zerujące	Element umożliwia zliczanie impulsów, przy czym przy każdej zmianie z 0 na 1 warunku zliczania zawartość akumulatora licznika jest zwiększana o 1, jeżeli nie jest spełniony warunek zerowania. Wyjście licznika UC jest w stanie ON po zliczeniu zadanej wartości impulsów aż do momentu spełnienia warunku zerowania.
Licznik odejmujący	DC – wejście zliczające rC – wejście zerujące	Element umożliwia zliczanie impulsów, przy czym przy każdej zmianie z 0 na 1 warunku zliczania zawartość akumulatora licznika jest zmniejszana o 1, jeżeli nie jest spełniony warunek zerowania. Wyjście licznika DC jest w stanie ON, gdy zawartość akumulatora licznika osiągnie 0 i pozostaje w tym stanie aż do momentu spełnienia warunku zerowania.
Element przesyłu danych	(→)	Element realizuje przesył danej pomiędzy podany źródłem a miejscem przeznaczenia, jeżeli połączenie z lewej strony elementu jest w stanie ON.
Dodawanie	[+]	Jeżeli połączenie z lewej strony jest w stanie ON wykonywana jest operacja dodawania a wynik umieszczany jest w podanym rejestrze.
Odejmowanie	[–]	Jeżeli połączenie z lewej strony jest w stanie ON wykonywana jest operacja odejmowania a wynik umieszczany jest w podanym rejestrze.
Mnożenie	[*]	Jeżeli połączenie z lewej strony jest w stanie ON, to jest wykonywana operacja mnożenia a wynik umieszczany jest w podanym rejestrze.
Dzielenie	[/]	Jeżeli połączenie z lewej strony jest w stanie ON, to jest wykonywana operacja dzielenia a wynik umieszczany jest w podanym rejestrze. Reszta z dzielenia umieszczana jest w rejestrze o numerze o jeden większym.
Komparatory	[=], [>], [<] [>=], [<=]	Jeżeli połączenie z lewej strony jest w stanie ON, to jest sprawdzana wybrana zależność pomiędzy argumentami. Jeżeli argumenty operacji spełniają zadaną relację, to wyjście elementu jest ustawiane w stan ON.

jednak celem pracy była przede wszystkim optymalizacja (pod względem czasu trwania cyklu pracy) oprogramowania systemu sterownika. W tabeli 2 są zebrane osiągnięte czasy obsługi poszczególnych elementów schematu drabinkowego.

Tabela 2. Czasy obsługi poszczególnych elementów schematu drabinkowego

Połączenie poziome	0 µs
Połączenie pionowe	20 µs
Pole puste	9 µs
Styki *(styki zwierne, rozwierne oraz reagujące na zbocza narastające i opadające zmiennych skojarzonych)*	14 µs
Cewki *(zwykłe, ustawiające i zerujące)*	21 µs
Liczniki *(liczący w górę oraz liczący w dół)*	30 µs (wejście zliczające) 12 µs (wejście zerujące)
Timery *(czasomierz załączający z opóźnieniem oraz wyłączający z opóźnieniem po wyłączeniu)*	26 µs (wejście ustawiające) 12 µs (wejście zerujące)
Element transferu danych	25 µs (przesłanie międzyrejestrowe) 36 µs (z obsługą przetwornika C/A) 74 µs (z obsługą przetwornika A/C)
Elementy arytmetyczne *(dodawanie, odejmowanie, mnożenie, dzielenie na liczbach całkowitych)*	32 µs (dodawanie i odejmowanie) 34 µs (mnożenie), 38 µs (dzielenie)
Komparatory *(wszystkie rodzaje operatorów relacji)*	32 µs
Czas obsługi pustego programu	370 µs

Bibliografia

1. Easy 400, 600, 800 Sortimentskatalog. Moeller, sierpień 2003
2. Oprogramowanie Cscape — Instrukcja obsługi. GE Fanuc Automation 2000
3. Pico Controllers. User Manual. Bulletin 1760. Rockwell Automation, czerwiec 2001
4. Podręcznik programowania sterownika LOGO! Siemens AG 1996
5. S7-200 Programmable Controller System Manual. Siemens AG 2003
6. Sterownik Programowalny ZEN. Omron, Wydanie 2004
7. Zelio-Logic Relays Catalog. Schneider Electric 2001
8. Chmiel M., Hrynkiewicz E.: Sposób analizy programu sterowania w postaci schematu drabinkowego dla małego sterownika programowalnego typu compact'. Krajowa Konferencja Elektroniki — KKE 2002, Kołobrzeg — Dźwirzyno 10–12.06.2002
9. Chmiel M., Hrynkiewicz E., Milik A.: A New Compact Programmable Logic Controller with Integrated Programming Equipment. IFAC Workshop on Programmable Devices and Systems — PDS 2001, Gliwice, Polska 22–23.11.2001
10. Chmiel M., Hrynkiewicz E., Muszyński M.: The Way of Ladder Diagram Analysis for Small Compact Programmable Controller. The 6th Russian-Korean International Symposium on Science and Technology — KORUS 2002, Nowosybirsk, Rosja 24–30.06.2002
11. Kwaśniewski J.: Programowalne sterowniki przemysłowe w systemach sterowania. Kraków 1999
12. Legierski T., Wyrwał J., Hajda J., Kasprzak J.: Programowanie sterowników PLC. Wydawnictwo Pracowni Komputerowej Jacka Skalmierskiego, Gliwice 1998
13. Michel G.: Programmable Logic Controllers. Architecture and Applications. John Wiley&Sons 1990
14. Muszyński M.: Programowalny sterownik logiczny (PLC) typu compact. Praca magisterska, Instytut Elektroniki Politechniki Śląskiej, Gliwice 1999
15. Stefański Ł.: Kompaktowy sterownik programowalny z 16-bitowym procesorem 80C196. Praca magisterska, Instytut Elektroniki Politechniki Śląskiej, Gliwice 2004