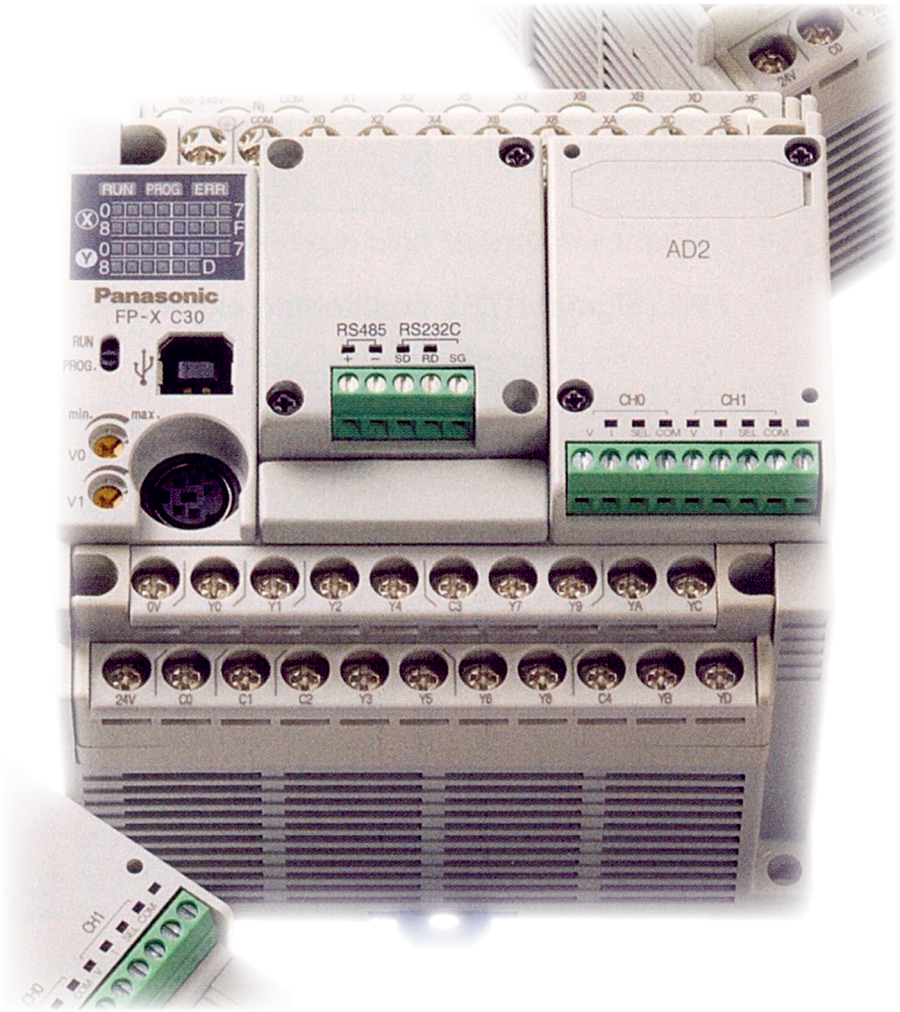
Programowanie PLC zgodnie z normą IEC61131 – języki programowania

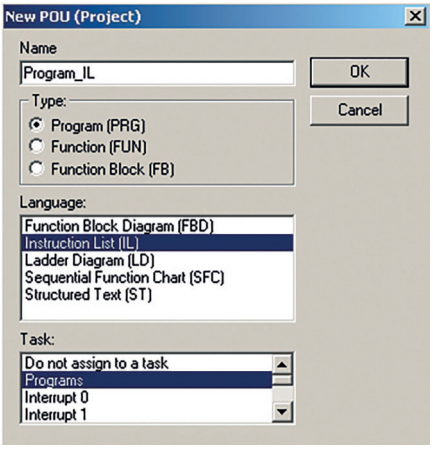
Norma IEC61131 definiuje pięć języków programowania PLC. Wśród nich znaleźć można zarówno języki graficzne, jak i tekstowe. Wybór stosownego języka zależy zarówno od preferencji użytkownika oraz ograniczeń narzuconych przez producenta sprzętu, bowiem nie każde środowisko programistyczne pozwala na tworzenie aplikacji w dowolnym z języków. W niniejszym artykule omówiono podstawowe cechy wspólne i różnice wszystkich z nich, posługując się przykładami opracowanymi z zastosowaniem środowiska Control FPWinPro.



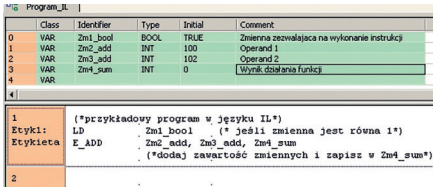
Obecnie największą popularnością wśród programistów sterowników PLC cieszy się język drabinkowy – głównie ze względu na podobieństwo do elektrycznych schematów sterowania opartych na przekaźnikach. Przed wprowadzeniem normy IEC61131 był to jeden z głównych języków programowania z tym, że każdy producent posiadał swoje unikalne narzędzie do programowania, co stanowiło niedogodność dla programistów i użytkowników. Wprowadzenie normy IEC61131 standaryzuje sterowniki oraz narzędzia do programowania. W normie przedstawiono pięć języków programowania, których dokładny opis przedstawiono w dalszej części artykułu.

Lista rozkazów

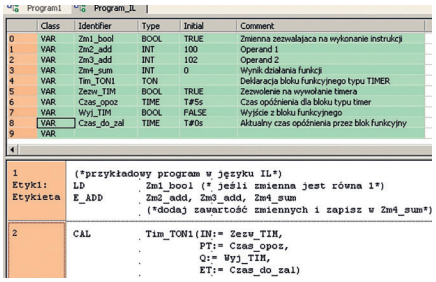
Język IL (*Instruction List*) jest tekstowym językiem programowania, którego składnia jest zbliżona do języka typu asembler. Program napisany w IL składa się z sekwencji rozkazów, a każdy z nich zawiera operator, modyfikator oraz operand. Operator określa działania, jakie mają być wykonane, natomiast operand reprezentuje stałe lub zmienne. Rozkaz może być poprzedzony etykietą, która reprezentuje skok. Dodatkowo może wystąpić komentarz, który służy do opisu zawartości linii programu. W języku IL jest dostępnych szereg standardowych operatorów, takich jak np.: przesłania, mnożenia, dzielenia, odejmowania, dodawania, porównywania, skoków, powrotu z podprogramu, boolowskich (AND, OR, XOR), ustawiania i zerowania wyjścia. Przed wprowadzeniem instrukcji programu w oprogramowaniu FPWinPro, należy utworzyć nową jednostkę oprogramowania POU, tj. kliknąć prawym przyciskiem myszki w „Project navigator" na „POUs" i wybrać „New POU". W dalszej kolejności trzeba wprowadzić nazwę programu oraz wybrać język programowania IL. Przykład wprowadzenia nowego programu zaprezentowano na **rysunku 1**. Przykład pro-



Rysunek 1. Widok okna z podczas tworzenia nowego programu typu IL



Rysunek 2. Widok okna z przykładowym programem w języku IL



Rysunek 3. Przykład programu z wywołanym blokiem funkcyjnym

stego programu w języku IL przedstawiono na **rysunku 2**. Obrazuje on działanie funkcji dodawania dwóch operatorów. Litera „E" w operatorze „E_ADD" oznacza, że do wywołania jest potrzebny warunek zezwalający (EN), w tym przypadku jest zmienna „Zm1_bool".

Podobne zasady obowiązują również podczas tworzenia funkcji i bloku funkcyjnego w oprogramowaniu FPWinPro. Wywołanie funkcji odbywa się poprzez umieszczenie jej nazwy oraz podaniu parametrów wejściowych. Wartość zwracana przez funkcję jest zapisywana przez następny rozkaz. Należy pamiętać, aby typ zmiennej zwracanej przez funkcję był zgodny z typem danej, do której jest przepisywany wynik działania funkcji. Bloki funkcjonalne FB mogą być wywoływane z poprzez instrukcję CAL, CALC lub CALCN. Przykład wywołania bloku funkcjonalnego typu TON (Timer z opóźnionym załączeniem) przedstawiono na **rysunku 3**. Przed wprowadzeniem kodu do programu muszą być najpierw zadeklarowane zmienne oraz blok funkcyjny.

Diagram drabinkowy

Język LD jest graficznym językiem programowania, w którym schemat sterowania przedstawia się w postaci symboli. Są one umieszczane w obwodach, podobnie jak w szczeblach w schemacie drabinkowym układu przekaźnikowego i wzajemnie połączone pionowo lub poziomo, tworząc odpowiedni algorytm sterowania. Do obwodu może być również przypisana etykieta (*label*). Obwód jest ograniczony z lewej i prawej strony przez szyny prądowe. Prawa szyna nie musi być rysowana, może pozostać w domyśle. W programie napisanym w języku LD instrukcje są wykonywane z lewej strony do prawej, przy czym wyjścia mogą być wyznaczone tylko wtedy, gdy wszystkie wejścia zostaną odczytane oraz zakończy się przetwarzanie całego obwodu. Kolejne obwody są przetwarzane po kolei, tak jak są narysowane na schemacie drabinkowym, z wyjątkiem wprowadzenia elementów, które te kolejność zmieniają. Podstawowymi elementami wykorzystywanymi w języku LD są styki i cewki. Styk jest elementem, który przekazuje stan do połączenia w schemacie. Styk nie modyfikuje wartości przypisanej do niego zmiennej. Cewka przekazuje stan połączeń ze schematu, powodując, że zmienna, która jest przypisana do danej cewki przyjmuje wartość wynikającą z algorytmu połączeń oraz zasady działania. Dostępne są następujące rodzaje standardowych styków i cewek: styk normalnie otwarty oraz normalnie zamknięty, styk reagujący na zbocze narastające oraz opadające, cewka normalna oraz negująca, cewka ustawiają oraz kasująca, cewka reagująca na zbocze narastające oraz opadające.

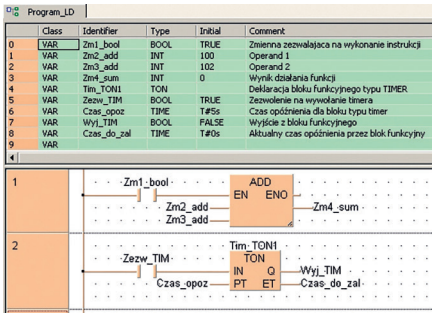
W języku LD oprócz realizacji prostych funkcji boolowskich istnieje możliwość wykonywania operacji złożonych, które wymagają odpowiednich funkcji lub bloków funkcjonalnych. Mogą to być bloki zdefiniowane w bibliotece lub przez użytkownika. Wywołana funkcja lub blok funkcyjny jest przedstawiany w postaci prostokąta. Funkcja ma tylko jedno wyjście, podczas gdy blok FB może ich mieć kilka. Parametry aktualne mogą być przekazywane za pomocą odpowiednich stałych lub zmiennych przez połączenie do bloku. Blok powinien zawierać przynajmniej jedno wejście i wyjście boolowskie, aby umożliwić przepływ prądu. W standardowych blokach jest dostępne wyjście, które nosi nazwę „Q". W blokach użytkownika funkcję taką może pełnić para „EN" i „ENO". W przypadku bloków funkcyjnych FB, nad ich symbolami graficznymi wprowadza się nazwę konkretnego egzemplarza. Przykład wywołania bloku funkcyjnego oraz funkcji przedstawiono na **rysunku 4**. Wykonuje on te same operacje, co program z rys. 3, a co więcej użyto w nich identycznych nazw zmiennych. Kompilator prawidłowo przetwarza instrukcje programu na kod wynikowy. Jeśli w tej samej jednostce programu POU pojawiłyby się identyczne nazwy zmiennych, wówczas kompilatora poinformuje o błędzie.

Funkcjonalny schemat blokowy

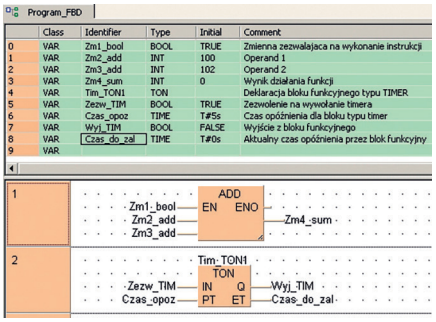
W języku FBD (Functional Block Diagram – funkcjonalny schemat blokowy) realizacja programu opiera się na przepływie sygnału poprzez elementy przetwarzania sygnałów. Przepływ sygnału następuje z wyjścia jednej funkcji lub bloku funkcjonalnego do wejścia przyłączonego następnego elementu. Są one wprowadzane w postaci prostokątów i elementów sterujących połączone liniami. Obwód stanowi zespół połączonych ze sobą elementów. Do każdego obwodu można przypisać etykietę o zasięgu lokalnym. Program napisany w języku FBD jest wykonywany w kolejności wprowadzenia obwodów, przy czym wartość wyjść zostanie wyznaczona, jeśli wszystkie wejścia zostaną wprowadzone do programu oraz zakończy się jego przetwarzanie. Program dla danego obwodu powinien się zakończyć przed rozpoczęciem wykonywania następnego obwodu. Wyjścia bloków funkcjonalnych nie mogą być łączone ze sobą, nie może być realizowana operacja sumy boolowskiej „OR" poprzez równoległe łączenie wyjść elementów. Rozwiązaniem tego problemu jest jawne użycie funkcji „OR". Podczas wprowadzania programu należy dążyć do tworzenia przejrzystej struktury, najlepiej poprzez liniowe ułożenie programu z jak najmniejszą liczbą przecinających się linii. Na **rysunku 5** zaprezentowano okna z wprowadzonym programem w języku FBD. Należy zauważyć, że program jest bardzo podobny do napisanego w języku LD.

Tekst strukturalny

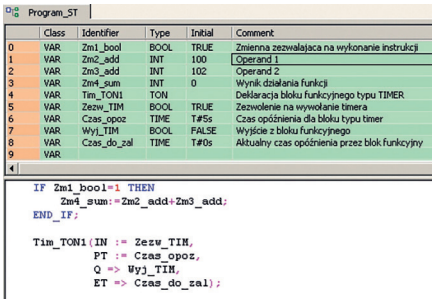
Język ST (*Structured Text*) należy do grupy języków wyższego poziomu, w którym używa się jako podstawowych elementów, wyrażeń i instrukcji. Dzięki temu struktura programu jest bardzo przejrzysta. W jednej linii może być więcej niż tylko jedna instrukcja, a poszczególne z nich są oddzielane średnikami. Wyrażenia składają się z operandów i operatorów. Operandami mogą być zmienne, stałe, wywoływane funkcje lub inne wyrażenia. Operatorami są symbole, które decydują, w jaki sposób ma być obliczany wynik.



Rysunek 4. Widok okna programu w języku LD z wprowadzoną funkcją oraz blokiem funkcyjnym



Rysunek 5. Widok okna programu w języku FBD z wprowadzoną funkcją oraz blokiem funkcyjnym



Rysunek 6. Widok okna programu w języku ST z wprowadzoną funkcją oraz blokiem funkcyjnym

Dostępny jest szereg operatorów takich jak: potęgowanie, negacja, mnożenie, dzielenie, dodawanie, odejmowanie, porównywanie, iloczyn i suma boolowska. Najwyższy priorytet mają wyrażenia umieszczone w nawiasach, następnie funkcje obliczające wartość funkcji (np. SIN(A)). Jeśli wyrażenie ma więcej niż dwa operandy, to najpierw jest obliczana wartość operandu znajdującego się z lewej strony. Instrukcje w języku ST są zakończone znakiem średnika. Do zapisu nowej wartości w zmiennej lub przepisaniu wartości wynikającej z działania funkcji służy instrukcja przypisania, która jest oznaczana symbolem „:=". Należy pamiętać, aby po obu stronach funkcji przypisania typy zmiennych były zgodne. Instrukcja ta jest również używana do nadawania wartości funkcji oraz przepisania wartości przed powrotem z funkcji.

Wywołanie bloku funkcyjnego zawiera nazwę bloku oraz umieszczone w nawiasach parametry wejściowe. Kolejność przypisania parametrów nie ma znaczenia i jeśli któryś z nich został pominięty, to zostanie nadana mu wartość początkowa zgodnie z wartością zadeklarowaną w bloku funkcyjnym. Na **rysunku 6** przedstawiono widok okna z programem w języku ST. Jest to taka sama aplikacja jak przedstawiona wyżej w innych językach.

W języku ST jest dostępnych kilka instrukcji związanych z wyborem (instrukcje „IF" oraz „CASE") oraz pętlami. Instrukcja wyboru „IF" prowadzi do wykonania grupy poleceń, jeśli spełniony jest określony warunek. W przeciwnym razie mogą być wykonane instrukcje umieszczone po słowie „ELSE" lub „ELSEIF". W instrukcji „CASE" wybór wykonywanych poleceń zależy od wartości parametru wywołania.

W instrukcji „FOR" polecenia powtarzane są do momentu napotkania słowa kluczowego „END_FOR". Przy każdym wykonaniu instrukcji jest modyfikowana zadeklarowana zmienna licznikowa, począwszy od wartości pierwotnej do końcowej, zgodnie z określonym przyrostem. Instrukcja „EXIT" umożliwia zakończenie pętli „FOR" przed warunkiem określonym w zmiennej kontrolnej. Pętle „WHILE" i „REPEAT" są używane, jeśli liczba powtórzeń nie jest znana.

Schemat funkcjonalny

W normie IEC61131-3 określono sposób działania sterownika w postaci sekwencyjnego schematu funkcjonalnego SFC (*Sequential Functional Chart*). W SFC zadanie sterowania przedstawia się za pomocą kroków i warunków przejścia między nimi. Metoda ta jest szczególnie przydatna do realizacji zadań, w których występują pewne powtarzających się sekwencje działań. W SFC możliwy jest podział na mniejsze elementy POU, którymi są kroki i przejścia wzajemnie połączone. W ten sposób tworzona jest sieć, która stanowi podstawę strukturalną. Z każdym krokiem jest zawarty odpowiedni zestaw instrukcji, który nazywa się akcjami (*actions*). Przy przejściach między krokami znajdują się warunki przejścia (*transition condition*). Kroki mogą być aktywne lub nieaktywne. W danej chwili stan POU jest określony przez zbiór aktywnych kroków. Naturalną formą przedstawiania schematu SFC jest forma graficzna, ponieważ reprezentuje ona zależności pomiędzy elementami w sposób bardziej czytelny. Stan początkowy POU jest określony przez wartości początkowe jego zmiennych wewnętrznych i wyjściowych oraz przez zbiór kroków aktywnych w chwili początkowej. Każda sieć SFC powinna zawierać tylko jeden krok początkowy, który aktywowany jest z chwilą inicjowania programu lub bloku funkcyjnego. Zmiana stanu polega na przechodzeniu pomiędzy krokami aktywnymi a następnymi, zależnie od spełnienia warunków przejścia. Przejście jest to warunek przepływu sygnału pomiędzy kolejnymi elementami, a jego działanie wynika z rozwiązania wyrażenia boolowskiego. Jeśli przejście jest dozwolone i jednocześnie spełniony jest odpowiedni warunek to następuje kasowanie przejścia oraz wyłączenie kroków poprzedzających i aktywacja kroków występujących po danym symbolu. Kolejne kroki i przejścia muszą występować naprzemiennie, tzn. dwa kroki muszą być rozdzielone przejściem oraz dwa przejścia muszą być rozdzielone krokiem. Jeśli krok jest aktywny to powinny być wykonywane związane z nim działania, które są określane jako akcje. Blok akcji określa, co powinno zostać wykonane, gdy krok staje się aktywny. Najprostsze akcje składają się z prostych operacji przypisania wartości do zmiennej (np. VAR_OUTPUT). Mogą również być rozbudowanymi programami napisanymi w postaci języka tekstowego lub graficznego i są one wykonywane dopóki aktywny jest związany z nim krok.

Podsumowując, w języku SFC nie występuje sekwencyjne wykonywanie instrukcji programu, ale cyklicznie ustalenie wartości statusu kroków i przejść. Inaczej jest w klasycznym języku programowania, w którym klasyczna jednostka POU zawiera sekwencję instrukcji wykonywanych jedna za drugą.

Funkcje i bloki standardowe

W normie IEC61131-3 ujednolicono sposób implementacji, wywołania oraz zachowanie standardowych funkcji i bloków funkcjonalnych takich jak: czasomierze, liczniki czy też operacje arytmetyczne. Producenci oprogramowania lub systemu sterownika PLC mogą oferować, oprócz zdefiniowanych w normie, również inne funkcje i bloki w celu zwiększenia możliwości systemu. Dodatkowe funkcje i bloki są dostępne również w oprogramowaniu FPWinPro i zostały podzielone na odpowiednie grupy w bibliotece. W normie przedstawiono siedem grup funkcji standardowych: funkcje konwersji typów, liczbowe, wyboru i porównywania, funkcje na bitach oraz na ciągach znaków, funkcje na typach danych związanych z czasem, na wyliczeniowych typach danych.

Niektóre funkcje mogą nie mieć określonej liczby wejść. Liczba parametrów wejściowych jest wtedy określana przez użytkownika i są one nazywane funkcjami rozszerzalnymi. Funkcjami tymi są: ADD, MUL, AND, OR, XOR oraz funkcje porównywania. Liczba wejść w takim symbolu graficznym jest zmieniana poprzez wysokość prostokąta reprezentującego funkcję.

Funkcje umożliwiające zmianę konwersję typów danych oznaczane są w postaci „ARGWEJ_TO_ARGWYJ", gdzie ARGWEJ oznacza typ argumentu wejściowego (np. INT, REAL), ARGWYJ oznacza argument wyjściowy (np. INT, DINT). Przykłady funkcji konwersji typów: INT_TO_REAL, DINT_TO_WORD.

W funkcjach liczbowych wyróżnia się te, w których jest podawania jedna zmienna oraz funkcje arytmetyczne. Do tych pierwszych należą funkcje logarytmiczne, trygonometryczne oraz wartość bezwzględna i pierwiastek kwadratowy. Do drugiej grupy zalicza się dodawanie, odejmowanie, mnożenie, dzielenie, resztę z dzielenia, potęgowanie oraz funkcję przepisania.

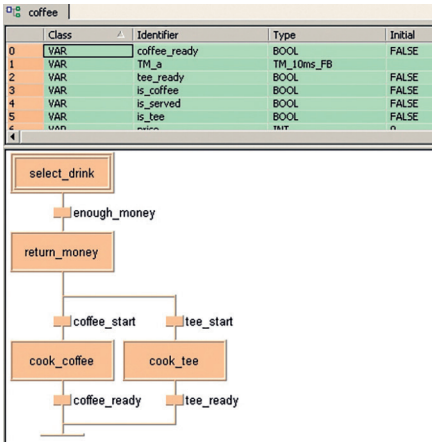
Funkcje operujące na ciągach bitów wykonują przesuwanie bitów lub działania boolowskie. Operują na danych należących do typu ANY_BIT. Funkcje przesuwania bitów w prawo lub lewo oraz cyklicznego przesuwania zawierają nazwy parametrów formalnych. Funkcje działające na bitach (AND, OR, XOR) są rozszerzalne.

Do funkcji wyboru należą:

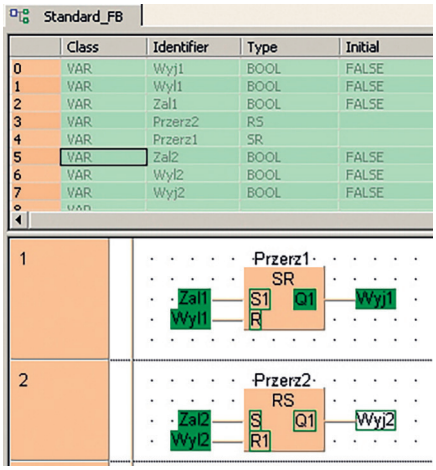
- wybór określonej wartości (SEL),
- wartości maksymalnej (MAX),
- minimalnej (MIN),
- ogranicznik wyboru (LIMIT),
- multiplekser (MUX).

Natomiast funkcje porównywania to:

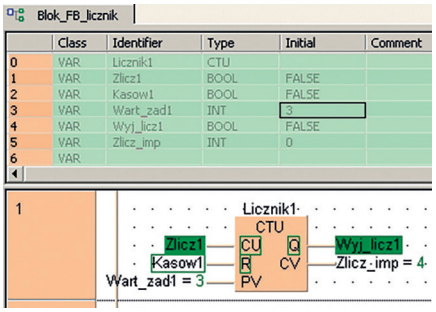
- większy (GT),
- większy lub równy (GE),
- równy (EQ),
- mniejszy lub równy (LE),
- mniejszy (LT),
- różny (NE).



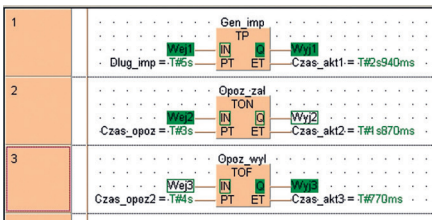
Rysunek 7. Przykład programu napisanego w języku SFC



Rysunek 8. Przykład użycia przerzutników SR



Rysunek 9. Przykład użycia bloku funkcyjnego typu licznik CTU



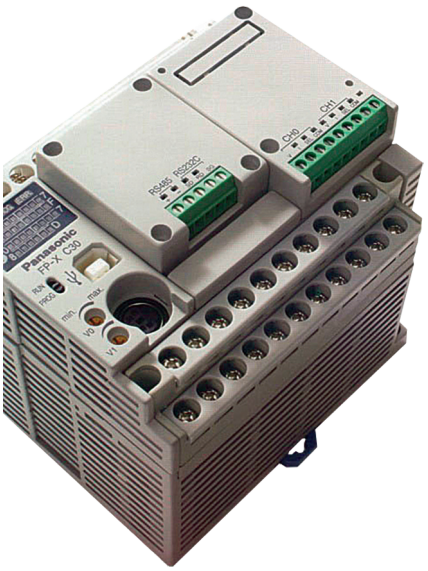
Rysunek 10. Przykład użycia bloków funkcyjnych czasomierzy

W obu grupach mogą być stosowane jako argumenty wejściowe określonego lub dla dowolnego typu danych należącego do typu uniwersalnego (ANY) z tym, że wszystkie z nich muszą być tego samego typu.

Funkcje porównywania i wyboru mogą być również stosowane dla ciągów znaków. Oprócz tego dostępne są funkcje obliczające długość ciągu znaków, obcinania znaków, łączenia ciągów, wstawiania ciągu znaków, kasowania znaków, zastąpienia znaków, znalezienia ciągu znaków.

Dla funkcji operujących na typach danych związanych z czasem (TIME, DATE, TIME_OF_DAY, DATE_AND_TIME) dostępne są operacje dodawania, odejmowania, mnożenia i dzielenia oraz łączenia czasu i daty. Należy zaznaczyć, że dla funkcji odejmowania jest dostępnych więcej kombinacji dla parametrów wejściowych niż dla funkcji dodawania. Funkcje MUL i DIV służą do mnożenia i dzielenia czasu trwania (TIME) przez liczbę typ liczby ANY_NUM.

Standardowe bloki funkcjonalne są dostępne we wszystkich językach programowania sterowników. Norma IEC61131-3 wyróżnia następujące grupy:



- elementy dwustanowe i detekcji zbocza,
- liczniki,
- czasomierze.

Do grupy elementów dwustanowych należą przerzutniki i semafory. Przerzutniki RS dostępne są w dwóch rodzajach, z wejściem S (*Set* – ustawiającym) jako dominującym lub z wejściem R (*Reset* – zerującym) jako dominującym. Przykłady użycia elementów typu przerzutnik SR w języku LD zilustrowano na **rysunku 8**. Blok semafora jest używany głównie w celu kontroli dostępu do zasobów systemu operacyjnego.

Dostępne są również bloki funkcjonalne, które umożliwiają wykrycie zbocza narastającego (R_TRIG) oraz opadającego (F_TRIG). W bloku zbocza narastającego wyjście Q zostaje ustawione na wartość 1 przy wykryciu przejścia na z 0 na 1 na wejściu CLK. Wyjście zostaje ustawione na 0 przy ponownym wywołaniu bloku. Blok detekcji zbocza opadającego działa podobnie z tym, że wykrywa zmianę stanu na wejściu CLK z 1 na 0. W języku LD zamiast tych bloków używa się odpowiednich styków |P|, |N| lub cewek (P) i (N).

W grupie liczników dostępne są następujące bloki funkcjonalne:

- licznik dodający (CTU – Counter Up),
- licznik odejmujący (CTD – Counter Down),
- licznik dodająco-odejmujący (CTUD – Counter Up Down).

Licznik dodający CTU zlicza impulsy pojawiające się na wejściu CU. Jeśli ich liczba (CV – Current Value) osiągnie lub przekroczy wartość zadaną PV (Preset Value) to na wyjściu pojawi się boolowska jedynka. Pojawienie się na wejściu R (Reset) wartości 1 spowoduje zerowanie licznika. Przykład użycia licznika dodającego został zilustrowany na **rysunku 9**. Na podobnej zasadzie działa licznik odejmujący CTD z tym, że zamiast dodawania impulsów odejmuje od wartości nastawionej PV. Jeśli wartość CV spadnie do 0 lub poniżej, to na wyjściu pojawi się boolowska jedynka. Licznik dodająco-odejmujący CTUD stanowi połączenie obydwu liczników i ma odpowiednio więcej wejść i wyjść. Należy zwrócić uwagę, że impulsy w licznikach są dodawane tylko przy zmianie stanu wejścia zliczającego z 0 na 1.

Bloki funkcyjne z grupy czasomierzy umożliwiają załączanie lub wyłączanie obwodów w zależności od odmierzonego czasu. Dostępne są następujące bloki:

- generator impulsu (TP),
- opóźnione załączenie (TON)
- opóźnione wyłączenie (TOF).

Blok funkcyjny TP generuje na wyjściu Q impulsy w momencie, gdy na wejściu nastąpi zmiana wartości z „0” na „1”. Czas trwania jest określony na wejściu PT (Preset Time). Po upływie tego czasu wyjście jest znów ustawiane na 0. Blok TON realizuje funkcję opóźnionego załączenia wyjścia w stosunku do sygnału wyzwalającego, pojawiającego się na wejściu IN. Wartość opóźnienia jest określona przez wejście PT. Wyjście Q jest utrzymywane w stanie 1 dopóki wejście IN nie przyjmie wartości 0. Jeśli w czasie odliczania wejście IN uzyska wartość 0, to czasomierz przerwie odliczanie. W na wyjściu Q bloku TOF pojawi się wartość 1 w chwili wystąpienia jedynki na wejściu IN. Wyjście zmieni stan na przeciwny, gdy wejście IN przyjmie wartość 0 z opóźnieniem zadanym na wejściu PT. Na **rysunku 10** zademonstrowano przykład użycia bloków funkcyjnych czasomierzy TP, TON i TOF.

Sławomir Kacprzak

Artykuł ukazał się w EP+