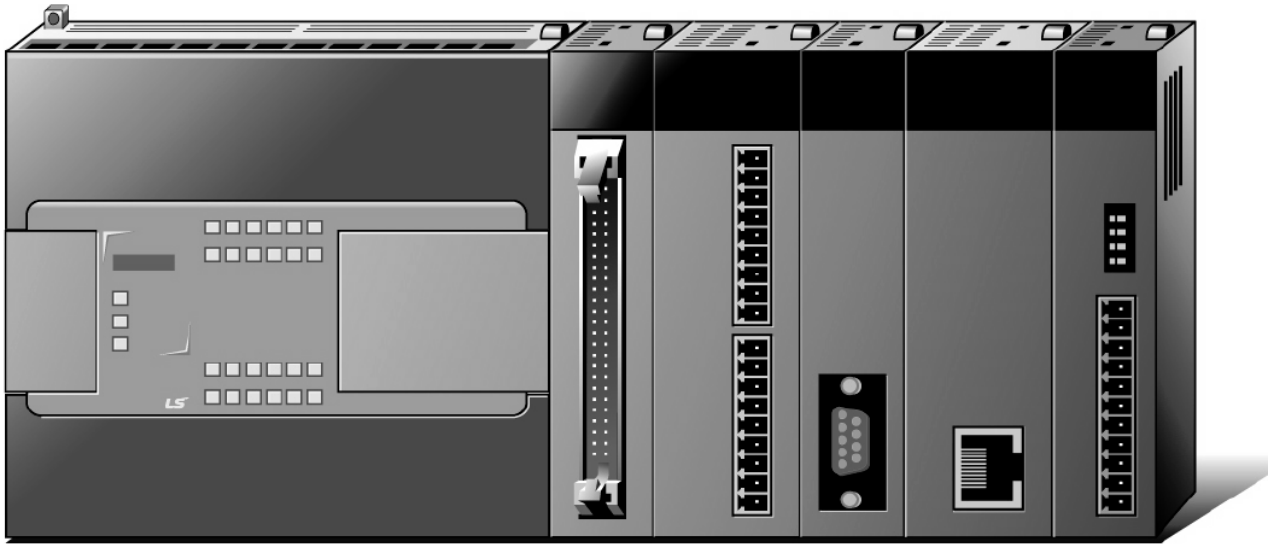
ANIRO

Podstawy programowania

Sterowników XGB-XEC

IEC 61131-3



Spis treści

Tabela szybkiego dostępu

Poniższa tabela przedstawia listę problemów często napotykanych przy pracy ze sterownikami PLC. Skorzystaj z tabeli aby łatwo znaleźć odpowiedź na swoje pytania

1 Wstęp

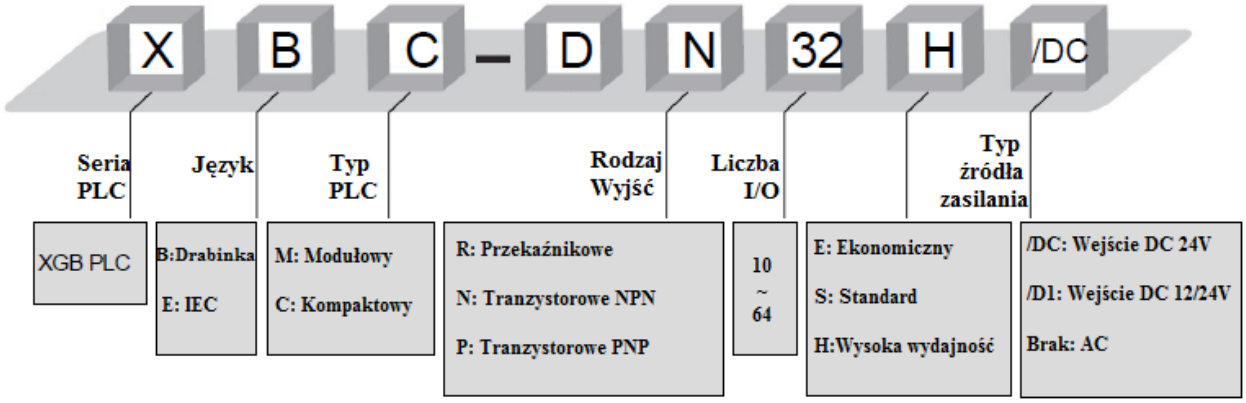
1.1 Opis

Sterowniki PLC serii XGB występują w dwóch typach: XBC oraz XEC

Typ sterownika	Język programowania	Komentarz
XBC	Ladder	Język drabinkowy LSIS (dedykowany)
XEC	Ladder, ST, SFC	Języki programowania zgodne z IEC 61131-3

Instrukcja skupia się na programowaniu sterowników serii XEC, przeprowadza użytkownika przez tworzenie nowego projektu, wybór języka programowania, wykorzystanie funkcji zależnie od języka, adresacje zmiennych,tworzenie funkcji użytkownika i inne.

1.2 Identyfikacja sterownika



1.3 Oprogramowanie

Do programowania sterowników PLC firmy LSIS należy zainstalować oprogramowanie XG5000. Najnowsza wersja oprogramowania jest dostępna na stronie producenta www.lsis.com.

1.4 Istotne instrukcje

Instrukcja jest tylko podsumowaniem informacji na temat programowania sterowników typu IEC. W celu szczegółowych informacji należy odnieść się do ogólnej instrukcji użytkownika.

Nazwa producenta	Nazwa Aniro	Opis
XGB Hardware(IEC) Manual	Instrukcja sprzętowa XGB(XEC)	Opisuje podstawy użytkowania, użycie modułów rozszerzeń, konfiguracje systemu oraz wbudowanych liczników sterownika XBG(IEC).
Manual XBC Economic/Standard	Instrukcja XBC - Typ Standardowy_Ekonomiczny	Opisuje podstawy użytkowania, użycie modułów rozszerzeń, konfiguracje systemu oraz wbudowanych liczników sterownika XBG.
Manual XEC Economic/Standard	Instrukcja XEC - Typ Standawdowy_Ekonomiczny	Opisuje podstawy użytkowania, użycie modułów rozszerzeń, konfiguracje systemu oraz wbudowanych liczników sterownika XBG(IEC).
Manual XGB Cnet	Instrukcja Cnet	Opisuje użycie wbudowanej funkcji komunikacji XGB oraz użycie zewnętrznych modułów Cnet I/F.
XGB Posotioning Manual	XGB - wbudowane pozycjonowanie	Opisuje jak korzystać z wbudowanej funkcji pozycjonowania sterownika XGB

Instrukcje te są dostępne do pobrania ze strony ANIRO lub bezpośrednio ze strony producenta.

2 IEC 61131-3

IEC 61131-3 jest częścią międzynarodowej normy opisującą graficzne i tekstowe języki programowania. Sterowniki XEC mogą być programowane w trzech językach opisanych w normie. Programy, zadania i bloki użytkownika zawarte w jednym projekcie mogą być napisane w różnych językach co ułatwia pracę umożliwiając większą elastyczność programowania sterowników.

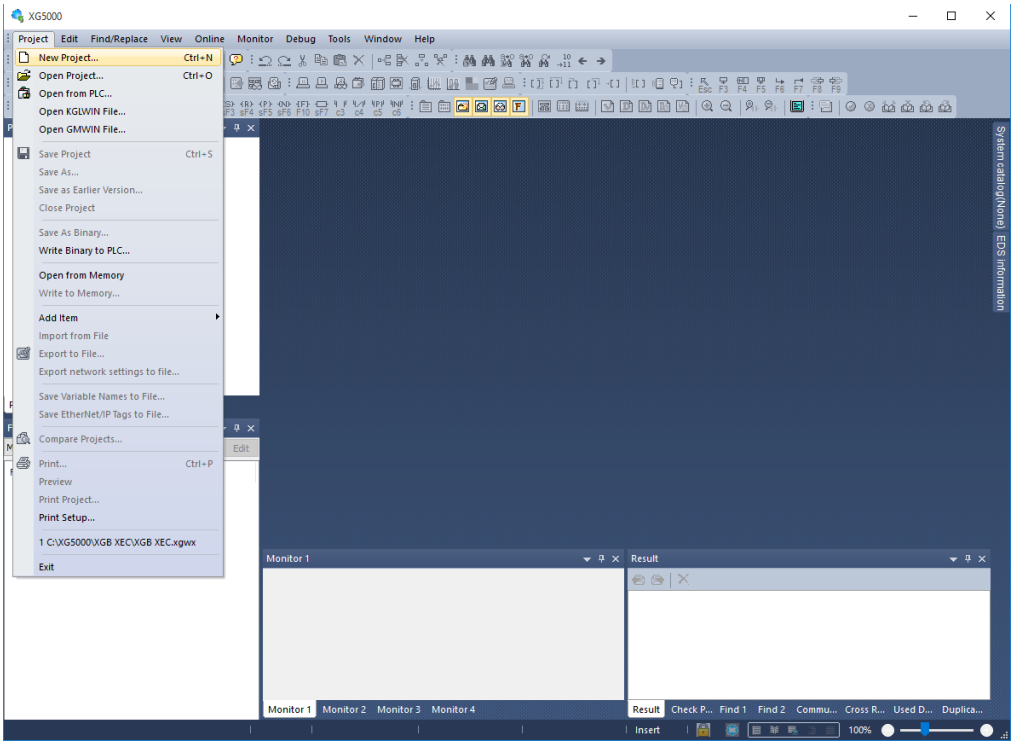
Tabela przedstawia języki programowania zgodne z IEC 61131-3 obsługiwane przez sterowniki XEC:

Nazwa producenta	Nazwa Aniro
Ladder Diagram (LD)	Schemat Drabinkowy
Sequencial Function Chart (SFC)	Sekwencyjna Karta Funkcji
Structured Text (ST)	Tekst Strukturalny

3 Utworzenie projektu

W celu utworzenia projektu zawierającego sterownik IEC należy uruchomić program XG5000.

Po ukazaniu się okna programu należy klinąć [Project]→ [New Project…]



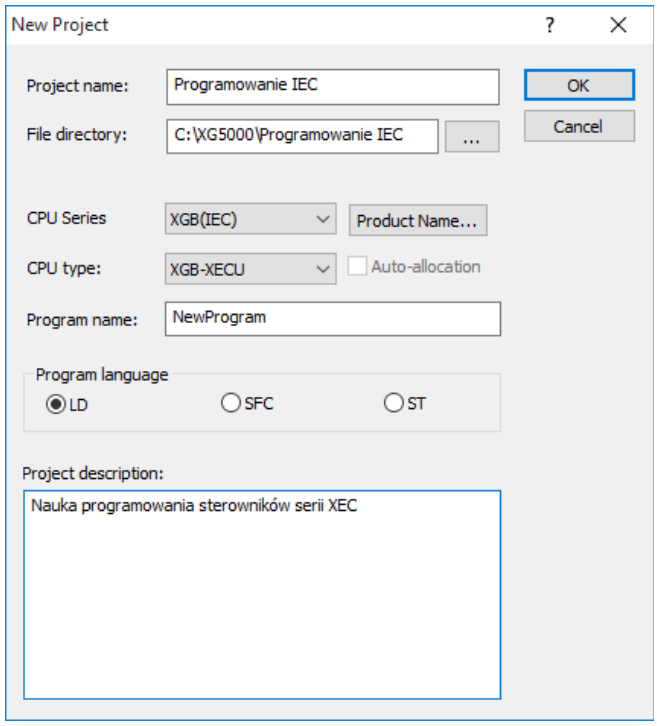
W oknie nowego projektu podajemy:

nazwę projektu, katalog zapisu, Serie oraz typ CPU.

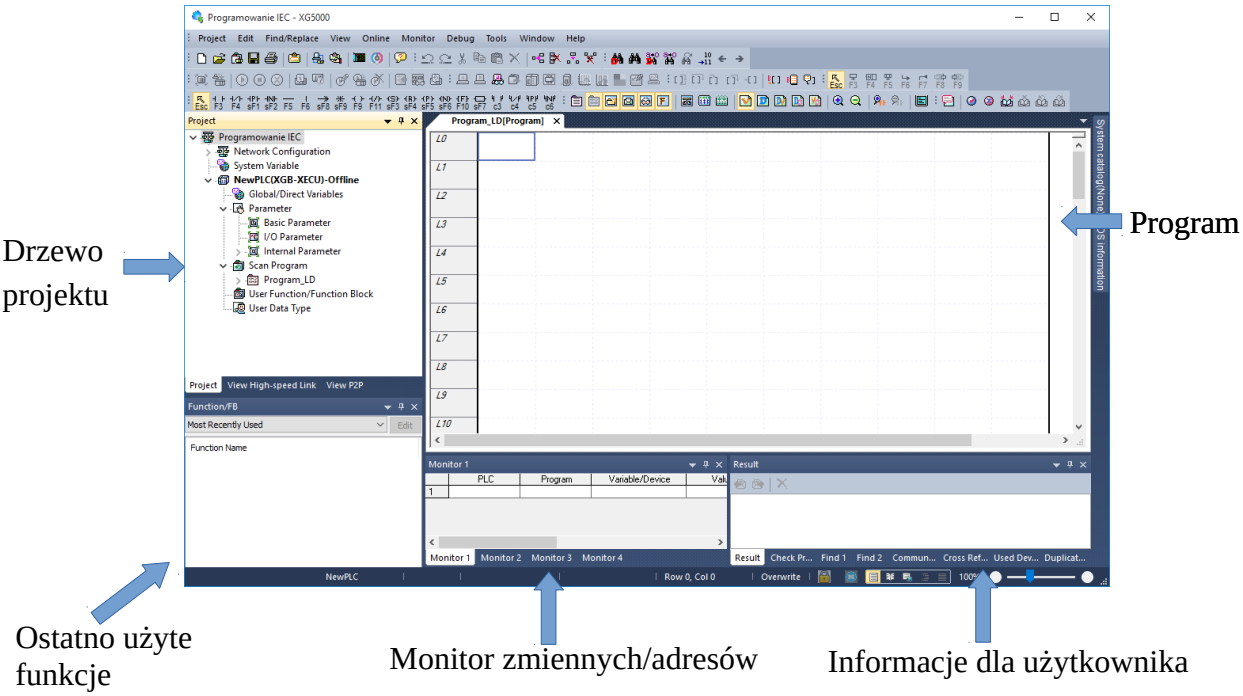
Nazwa pola	Opis
Project name	Nazwa pod którą zostanie zapisany projekt
File directory	Ścieżka zapisu katalogu projektu
CPU Series	Seria CPU, Należy wybrać XGB(IEC)
CPU type	Typ CPU zależnie od serii PLC (E, S, H, U)
Program name	Nazwa pierwszego programu
Program language	Język pierwszego programu
Project description	Opis programu

Wszystkie powyższe dane możemy zmienić po utworzeniu projektu.

Przykładowe uzupełnienie nowego projektu:

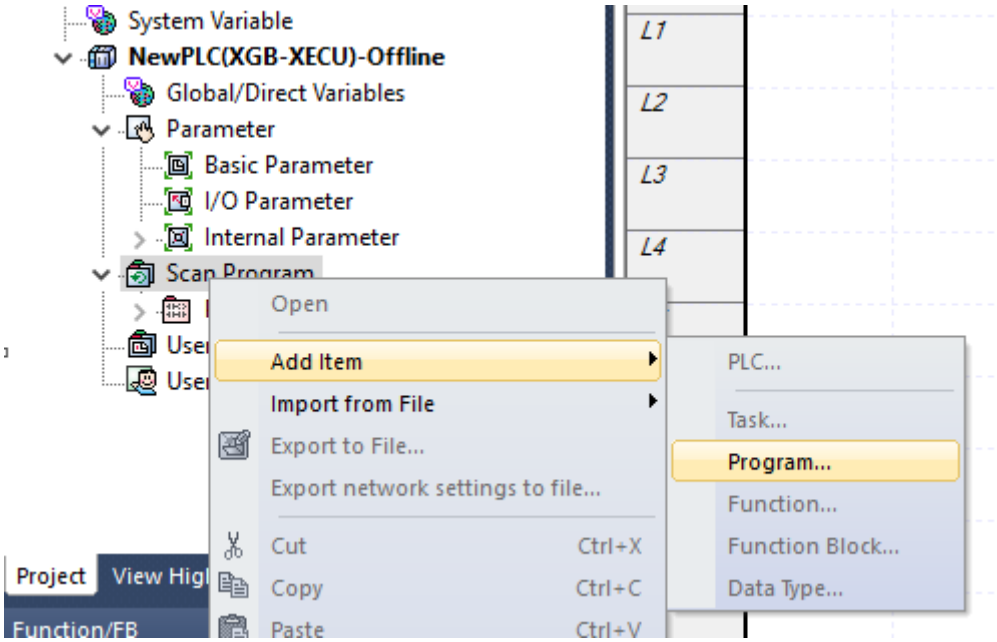


Po wciśnięciu przycisku OK zostanie otworzone pole edycji utworzone programu. Po lewej stronie zostanie uzupełnione drzewo projektu które pozwala na poruszanie się po jego częściach.

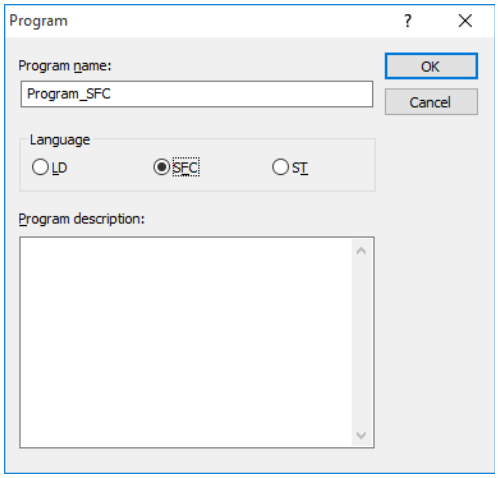


3.1 Tworzenie nowego programu

Aby utworzyć nowy program należy kliknąć prawym przyciskiem myszy na elemencie drzewa [Scan Program] i wybrać [Add item]→[Program…]



W oknie „Program” należy wpisać nazwę programu oraz język programowania

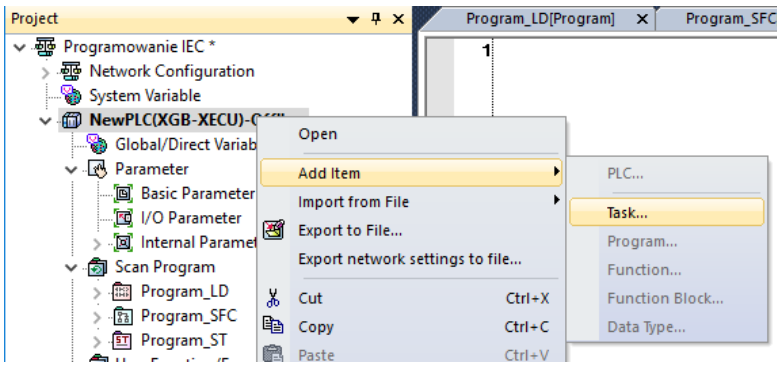


Tworzenie nowego programu jest niedostępne przy podłączonym sterowniku PLC.

4 Zadania

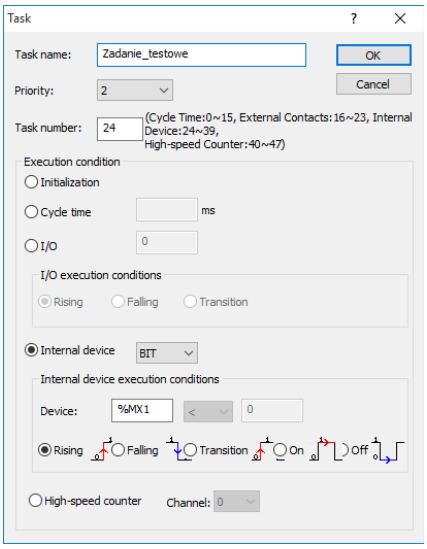
4.1 Tworzenie zadania

Aby utworzyć nowe zadanie należy kliknąć prawym przyciskiem myszy na elemencie drzewa zawierającym nazwę sterownika i wybrać [Add item]→[Task…]

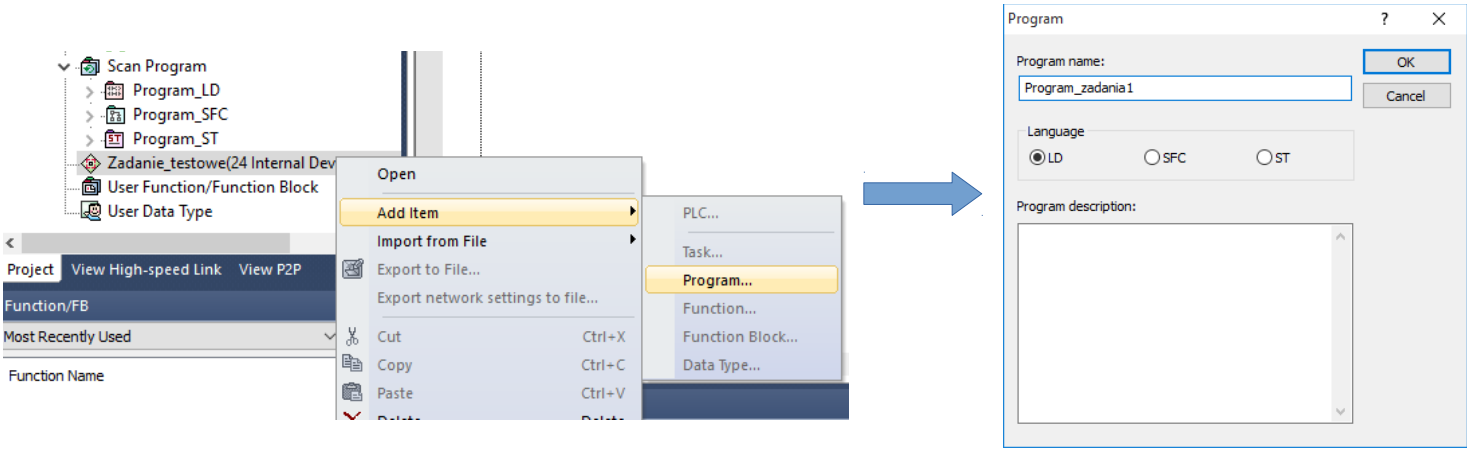


W oknie nowego zadania znajdują się następujące pola

Nazwa pola	Opis
Task name	Nazwa pod którą zostanie zapisane zadanie
Priority	Priorytet zadania
Task number	Numer zadania
Execution condition	Metoda wywołania zadane
Inicialization	Wywołanie podczas przejścia PLC w tryb RUN
Cycle time	Wywołanie co określony czas
I/O	Wywołanie na podstawie wybranego We/Wy
Internal device	Wywołanie na podstawie wybranego adresu wewnętrznego
High-speed counter	Wywołanie na podstawie szybkiego licznika



Domyślnie utworzone zadanie nie posiada zdefiniowanego programu. Należy kliknąć prawym przyciskiem myszy na utworzone zadanie i wybrać [Add item]→[Program…]. W otworzonym oknie należy wpisać nazwę programu, wybrać język programowanie i kliknąć OK.



Program wykonywany przez zadanie należy tworzyć tak samo jak programy główne. Należy pamiętać, że każdy utworzony program musi być niepusty i poprawnie napisany nawet jeżeli nigdy nie zostanie wywołany. Błąd i programie zadanie spowoduje błąd kompilacji.

W skład jednego zadania może wchodzić kilka programów napisanych w różnych językach programowania.

5 Zmienne oraz adresy

5.1 Adresy

Obszary pamięci wykorzystywane w XG5000 do programowania XEC są podobne do obszarów zdefiniowanych dla programowania sterowników typu XBC:

%I	Wejścia cyfrowe
%Q	Wyjścia cyfrowe
%M	Pamięć wewnętrzna
%K	Obszar podtrzymany, zawiera flagi PID, Logowania danych oraz szybkiego licznika
%U	Obszar modułów specjalnych(np. Wejścia analogowe)
%F	Flagi systemowe
%L	Komunikacja, P2P, połączenie High-speed
%R	Obszar pamięci flash
%W	Obszar pamięci flash oraz rejest plików

Do obszarów pamięci należy się odwołać dodając definicje typu danych. Niezależnie od wybranego typu uzyskujemy dostęp do tego samego obszaru danych.

X/brak	1 bit
B	1 bajt (8 bitów)
W	1 słowo (16 bitów)
D	1 Podwójne słowo (32 bity)
L	1 Długie słowo (64 bity)

Przykład:

%MW0 odwołuje się do pierwszego słowa pamięci M na które składają się bity %MX0 Do %MX15

Zmieniając wartość bitu %MX2 z 0 na 1 jednocześnie zmienimy wartość trzeciego bitu słowa %MW1 zwiększając jego wartość o 4.

Bit %MW1.0 jest pierwszym bitem słowa i jest równoważny z %MX16.

5.2 Obszar pamięci wejść/wyjść cyfrowych

Odwołanie do wejść oraz wyjść sterownika musi zostać dokonane poprzez określenie bazy sterownika, gniazda We/Wy oraz numeru pinu w danym gnieździe.

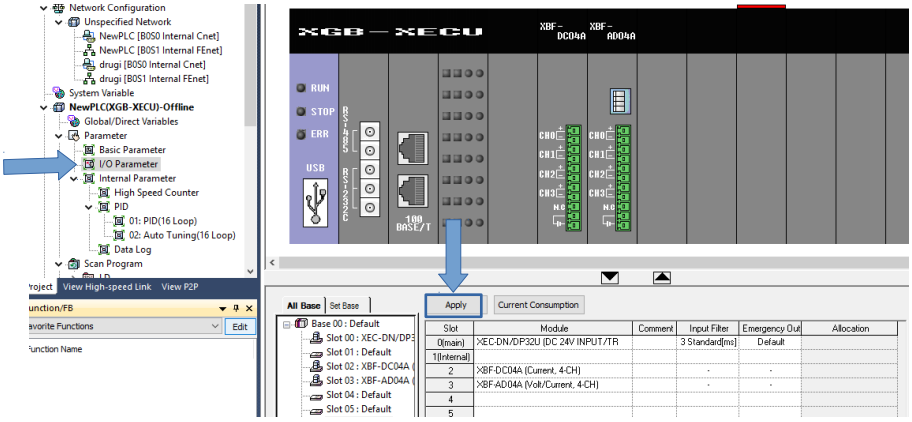
Przykład:

%IX0.0.15 – wejście nr 15 sterownika (gniazdo 0)
%QX.03.2 – wyjście nr 2 modułu rozszerzeń umieszczonego w gnieździe nr 3

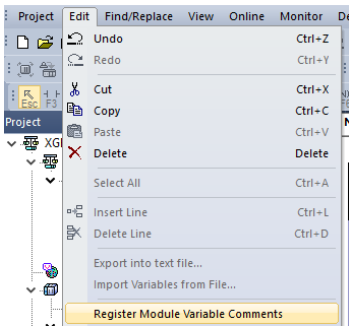
5.3 Obszar pamięci wejść/wyjść analogowych

Dane modułów specjalnych(We/Wy analogowe, moduły temperaturowe itd) są zwykle przechowywane w obszarze U w formie podobnej do obszaru cyfrowego. Aby uzyskać dane z dołączonego modułu należy w adresie określić numer bazy (domyślnie 0) oraz numer gniazda modułu.

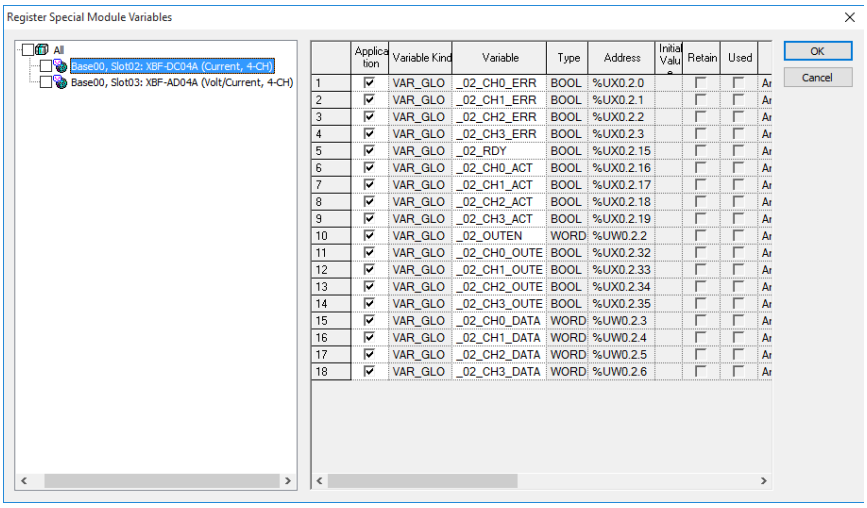
Uzyskanie danych z modułów specjalnych jest ułatwione dzięki opcji automatycznej rejestracji zmiennych modułu. Aby tego dokonać trzeba w oknie [I/O Parameter] zdefiniować sterownik oraz dołączone do niego moduły a następnie kliknąć [Apply]. W poniższym przypadku dodano moduł wyjść analogowych XBF-DC04A w gnieździe 2 oraz moduł wejść analogowych XBF-AD04A.



Następnie należy otworzyć zakładkę [Edit] nacisnąć [Register Module Variable Comments]



Wyświetlona zostanie informacja o automatycznej rejestracji zmiennych w oparciu o zdefiniowane wcześniej moduły. Po zatwierdzeniu otworzy się okno zawierające wszystkie adresy dotyczące zdefiniowanych modułów specjalnych, komentarze informujące o ich przeznaczeniu oraz ich domyślne nazwy.



	Application	Variable Kind	Variable	Type	Address	Initial Value	Retain	Used	
1	☑	VAR_GLO	_02_CH0_ERR	BOOL	%UX0.2.0		☐	☐	Ar
2	☑	VAR_GLO	_02_CH1_ERR	BOOL	%UX0.2.1		☐	☐	Ar
3	☑	VAR_GLO	_02_CH2_ERR	BOOL	%UX0.2.2		☐	☐	Ar
4	☑	VAR_GLO	_02_CH3_ERR	BOOL	%UX0.2.3		☐	☐	Ar
5	☑	VAR_GLO	_02_RDY	BOOL	%UX0.2.15		☐	☐	Ar
6	☑	VAR_GLO	_02_CH0_ACT	BOOL	%UX0.2.16		☐	☐	Ar
7	☑	VAR_GLO	_02_CH1_ACT	BOOL	%UX0.2.17		☐	☐	Ar
8	☑	VAR_GLO	_02_CH2_ACT	BOOL	%UX0.2.18		☐	☐	Ar
9	☑	VAR_GLO	_02_CH3_ACT	BOOL	%UX0.2.19		☐	☐	Ar
10	☑	VAR_GLO	_02_OUTEN	WORD	%UW0.2.2		☐	☐	Ar
11	☑	VAR_GLO	_02_CH0_OUTE	BOOL	%UX0.2.32		☐	☐	Ar
12	☑	VAR_GLO	_02_CH1_OUTE	BOOL	%UX0.2.33		☐	☐	Ar
13	☑	VAR_GLO	_02_CH2_OUTE	BOOL	%UX0.2.34		☐	☐	Ar
14	☑	VAR_GLO	_02_CH3_OUTE	BOOL	%UX0.2.35		☐	☐	Ar
15	☑	VAR_GLO	_02_CH0_DATA	WORD	%UW0.2.3		☐	☐	Ar
16	☑	VAR_GLO	_02_CH1_DATA	WORD	%UW0.2.4		☐	☐	Ar
17	☑	VAR_GLO	_02_CH2_DATA	WORD	%UW0.2.5		☐	☐	Ar
18	☑	VAR_GLO	_02_CH3_DATA	WORD	%UW0.2.6		☐	☐	Ar

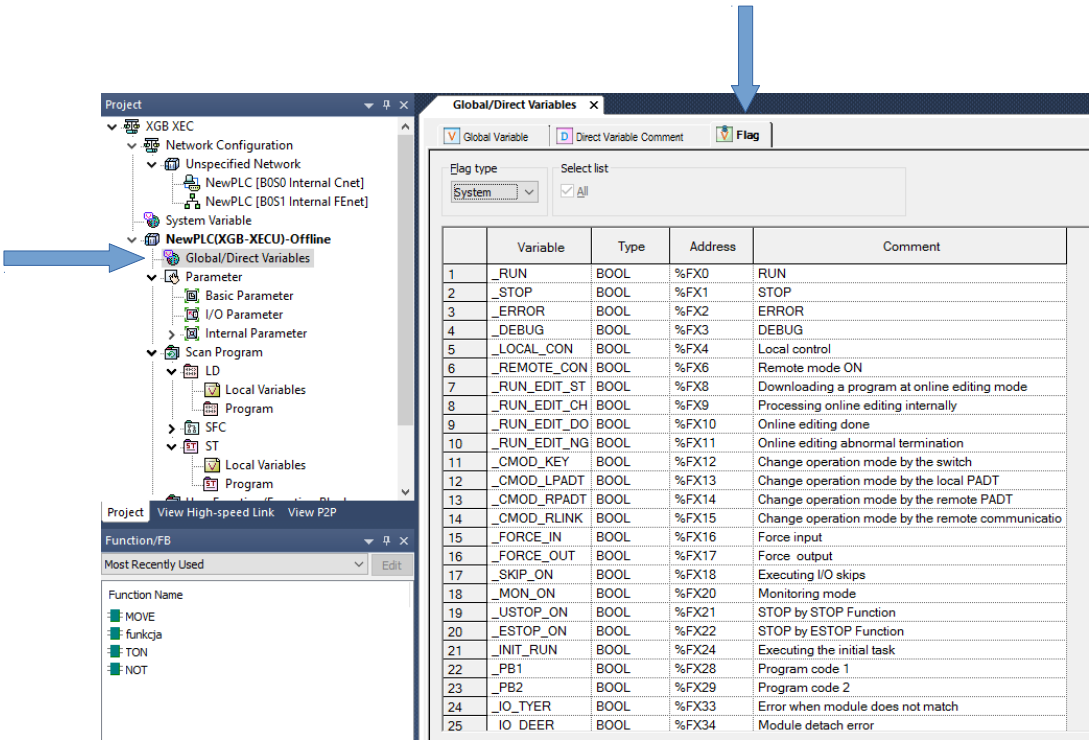
W powyższym oknie należy zaznaczyć okno [Application] dla wszystkich parametrów potrzebnych w projekcie oraz zmienić nazwy zmiennych w zależności od potrzeb. Po kliknięciu ok wszystkie zaznaczone zmienne zostaną automatycznie utworzone w oknie [Global Variable] umożliwiając użycie ich w programie.

6 Zmienne

Zmienne – (adres pamięci posiadający symboliczną nazwę) w sterownikach XEC są podzielone są na zmienne lokalne oraz globalne. Każdy utworzony program oraz zadanie posiada własne zmienne lokalne. Zmienne te są widoczne tylko z poziomu danego programu, zdefiniowanie zmiennej lokalnej nie daje możliwości użycia jej w innym podprogramie. Należy pamiętać, że zmienna jest powiązaniem adresu pamięci z nazwą(identyfikatorem). Inny program może odwołać się adresu użytego jako zmienna lokalna i dokonać zmiany wartości. Dlatego należy zachować ostrożność w używaniu bezpośrednich adresów a nie zmiennych.

6.1 Zmienne systemowe

Dostęp do zdefiniowanych znaczników(flag) wewnętrznych sterownika można uzyskać poprzez wejście w okno Global/Direct Variables i przejście do zakładki [Flag]



	Variable	Type	Address	Comment
1	_RUN	BOOL	%FX0	RUN
2	_STOP	BOOL	%FX1	STOP
3	_ERROR	BOOL	%FX2	ERROR
4	_DEBUG	BOOL	%FX3	DEBUG
5	_LOCAL_CON	BOOL	%FX4	Local control
6	_REMOTE_CON	BOOL	%FX6	Remote mode ON
7	_RUN_EDIT_ST	BOOL	%FX8	Downloading a program at online editing mode
8	_RUN_EDIT_CH	BOOL	%FX9	Processing online editing internally
9	_RUN_EDIT_DO	BOOL	%FX10	Online editing done
10	_RUN_EDIT_NG	BOOL	%FX11	Online editing abnormal termination
11	_CMOD_KEY	BOOL	%FX12	Change operation mode by the switch
12	_CMOD_LPADT	BOOL	%FX13	Change operation mode by the local PADT
13	_CMOD_RPADT	BOOL	%FX14	Change operation mode by the remote PADT
14	_CMOD_RLINK	BOOL	%FX15	Change operation mode by the remote communicatio
15	_FORCE_IN	BOOL	%FX16	Force input
16	_FORCE_OUT	BOOL	%FX17	Force output
17	_SKIP_ON	BOOL	%FX18	Executing I/O skips
18	_MON_ON	BOOL	%FX20	Monitoring mode
19	_USTOP_ON	BOOL	%FX21	STOP by STOP Function
20	_ESTOP_ON	BOOL	%FX22	STOP by ESTOP Function
21	_INIT_RUN	BOOL	%FX24	Executing the initial task
22	_PB1	BOOL	%FX28	Program code 1
23	_PB2	BOOL	%FX29	Program code 2
24	_IO_TYER	BOOL	%FX33	Error when module does not match
25	IO_DEER	BOOL	%FX34	Module detach error

6.2 Zmienne lokalne

Zmienne lokalne są zbiorem zmiennych udostępnionym do użycia jednemu programowi, zadaniu lub funkcji. Dostęp do zmiennych lokalny można uzyskać poprzez rozwinięcie folderu w drzewku projektu odpowiadającemu danemu programowi i wybór elementu [Local Variables]

Dostępne typy zmiennych

VAR	Zmienna lokalna
VAR_CONSTANT	Zmienna lokalna o stałej wartości (stała z identyfikatorem)
VAR_EXTERNAL	Zmienna zewnętrzna(globalna)
VAR_EXTERNAL_CONSTANT	Zmienna zewnętrzna a o stałej wartości (stała z identyfikatorem)

6.3 Zmienne globalne

Zmienne globalne mogą być użyte w każdym progamie i zadaniu po uprzednim wpisaniu ich w tabelę zmiennych lokalnych(VAR_EXTERNAL). Należy pamiętać, że można zdefiniować zmienną lokalną o adresie istniejącej już zmiennej lokalnej (tylko w przypadku nie definiowania zmiennej globalnej w postaci VAR_EXTERNAL w tabeli lokalnej)

Aby zdefiniować zmienną globalną należy wybrać element [Global/Drirect Variables] z drzewa projektu.

7 Języki programowania

7.1 Schemat drabinkowy (LD)

U podstaw języka drabinkowego leży przedstawienie założeń logicznych w postaci styków i cewek przekaźników. Odniesienie do drabinki wynika z wyglądu powstałego schematu.

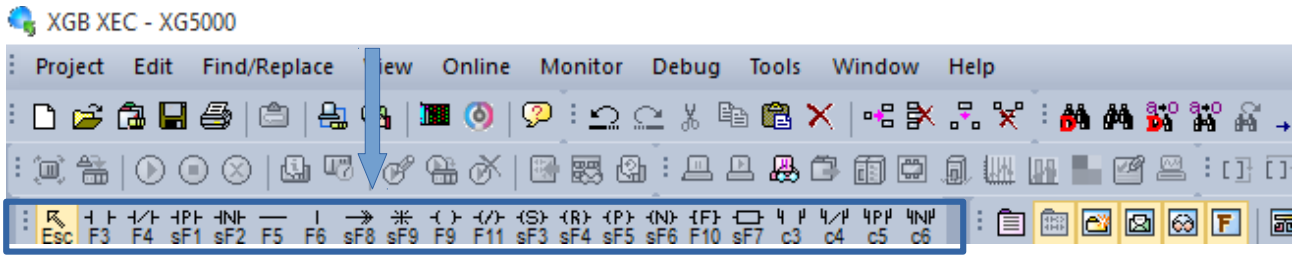
7.1.1 Podstawowe elementy

Podstawowe elementy możliwe do umieszczenia w oknie programu drabinkowego:

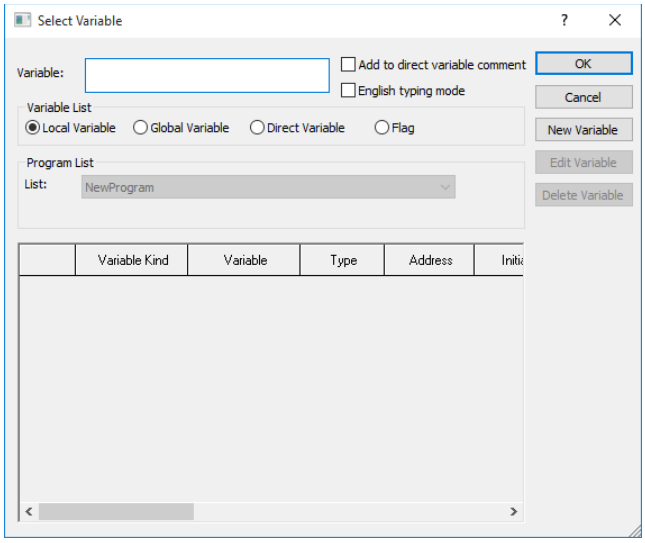
F3	Styk zwierny NO
F4	Styk rozwierny NC
sF1	Styk zwierny reagujący na zbocze
sF2	Styk rozwierny reagujący na zbocze
F5	Linia pozioma
F6	Linia pionowa
sF8	Wypełnij linią poziomą do końca
sF9	Negacja
F9	Cewka
F11	Cewka zanegowana
sF3	Cewka typu SET
sF4	Cewka typu RESET
sF5	Cewka reagująca na zbocze rosnące
sF6	Cewka reagująca na zbocze opadające
F10	Funkcja
sF7	Funkcje programowe
c3	Styk zwierny OR
c4	Styk rozwierny OR
c5	Styk zwierny OR reagujący na zbocze rosnące
c6	Styk rozwierny OR reagujący na zbocze opadające

Elementy reagujące na zbocze zostają aktywowane tylko w jednym cyklu przy którym wykryto zbocze. Zapobiega to wielokrotnemu wykonaniu operacji bez potrzeby używania dodatkowych bitów pamięci jako flag.

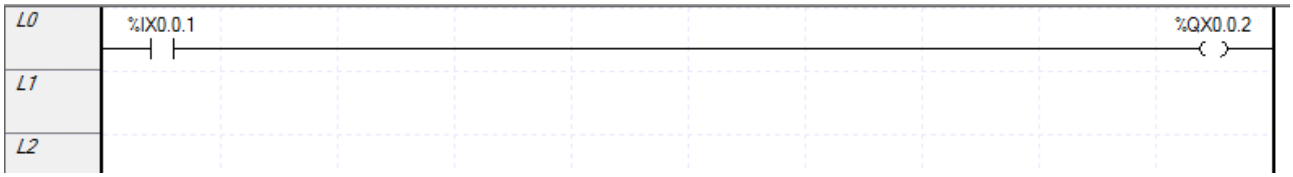
Lista tych elementów jest domyślnie umieszczona na pasku narzędzi



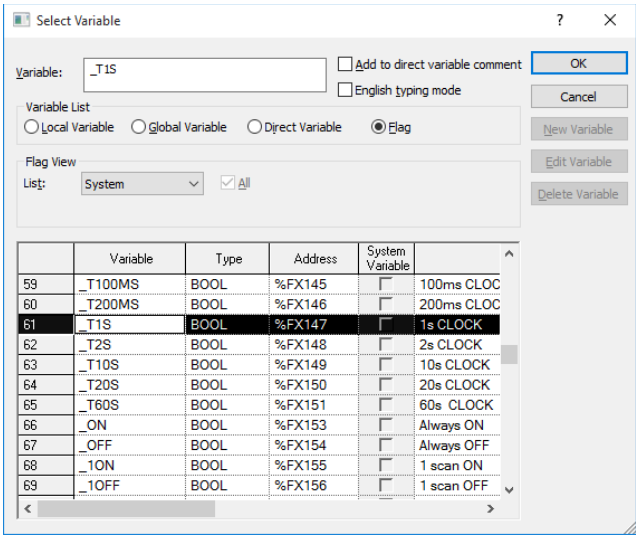
Poniżej podano przykład programu który po otrzymaniu sygnału na wejście nr 1 sterownika ustawi stan wysoki wyjścia nr 2. Po wybraniu elementu styku zwiernego i cewki z paska narzędzi umieszczamy je na jednym poziomie z po programu. Po umieszczeniu każdego z nich zostanie wyświetlone okno pozwalające na przypisanie elementu do zmiennej lub wejścia.



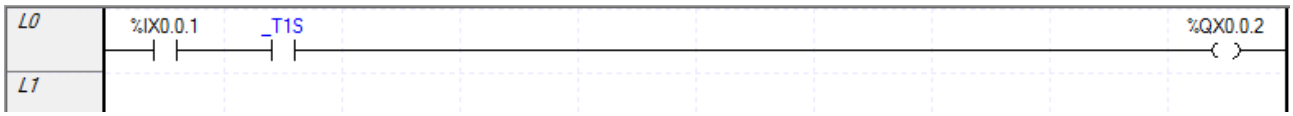
W powyższym oknie możemy utworzoną wcześniej zmiennej lokalną lub globalną(lista zmiennych lokalnych znajduje się w dolnej części okna), użyć flagi systemowej, utworzyć nową zmienną klikając [New Variable] lub odwołać się bezpośrednio do adresu pamięci. W tym przypadku użyto odwołania bezpośredniego wpisując adres %IX0.0.01 dla styku oraz adres %QX0.0.2 dla cewki. Oba elementy połączono poziomą linią (skrót klawiszowy F5). Efektem jest prosty program zilustrowany poniżej:



W sytuacji, gdy chcemy aby wyjście zmieniało cyklicznie swoją wartość możemy posłużyć się jedną z flag systemowych dotyczących czasu. Wybrano element styku umieszczając go pomiędzy dwoma istniejącymi już elementami. Z listy wybrano flagę zegara o okresie 1s:



Efektem jest poniższa linijka:



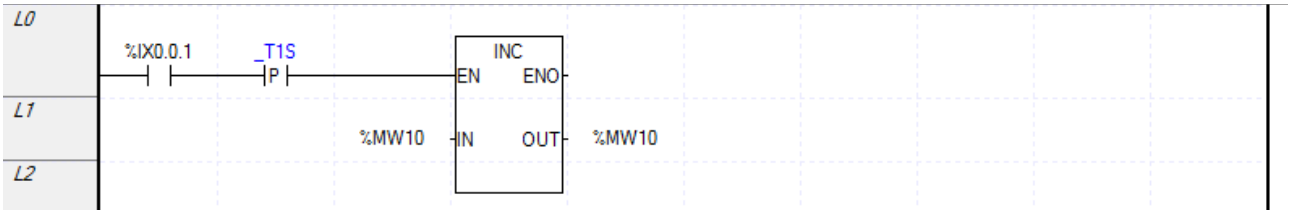
Program ten będzie cyklicznie zmieniał stan wyjścia %Q0.0.2 co 0.5s tak długo jak aktywne jest wejście %IX.0.01

7.1.2 Funkcje i bloki funkcyjne

Każda funkcja w programowaniu w schemacie drabinkowym posiada wejścia po lewej oraz wyjścia po prawej stronie bloku. Przykład wstawienia funkcji do programu zostanie wykonany przy użyciu funkcji inkrementowania INC.

Po wybraniu z paska narzędzi obiektu funkcji [Function/FB] bądz kliknięciu F10 mamy możliwość umieszczenia funkcji w programie. Najpierw należy wybrać miejsce w programie a następnie wymaganą funkcję z wyświetlonej listy.

Poniższy program zwiększa wartość słowa %MW10 o 1 co jedną sekundę tak długo jak aktywne jest wejście %IX0.0.1. Należy zwrócić uwagę na użycie cewki reagującej na zbocze przy zmiennej cyklicznej _T1S. Założeniem jest jedna aktywacja funkcji INC w przejściu _T1S w stan wysoki. Efektem użycia zwykłego styku byłaby wielokrotna inkrementacja %MW10 podczas jednego stanu wysokiego _T1S.



Aby wstawić adres wejścia i wyjścia (w tym przypadku %MW10) należy dwukrotnie kliknąć na pustą komórkę obok danego wejścia/wyjścia.

Wyjcie ENO

Wyjście ENO posiada taki sam stan jak wejście EN. Pozwala to na łączenie szeregowe kilku bloków funkcji oraz dodanie cewki do wyjścia bloku.

Wyjście Q

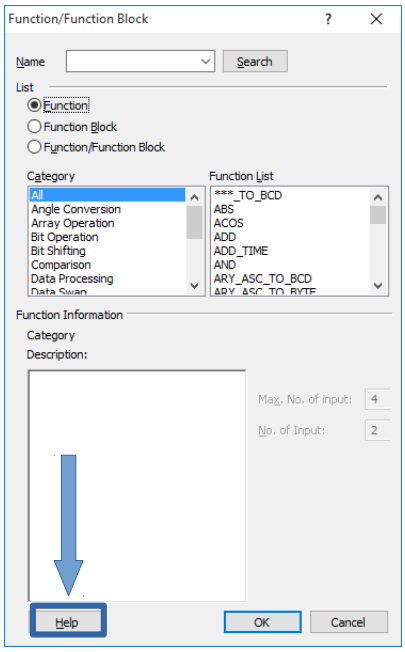
Wyjście Q jest wyjściem binarnym funkcji. Zwykle oznacza spełnienie postawionych warunków (np. zliczenie zadanej ilości impulsów przez funkcje CTU)

Wyście DONE

Wyjście DONE występuje w blokach funkcyjnych które są realizowane przez więcej niż 1 cykl programu. Wyjście to informuje o zakończeniu operacji.

7.1.3 Pomoc

Podczas pisania programu można uzyskać pomoc dotyczącą sposobu działania funkcji lub bloku funkcyjnego poprzez kliknięcie przycisku [Help] w oknie wyboru funkcji.

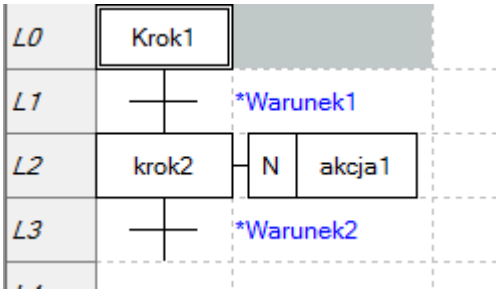


Po naciśnięciu przycisku Help zostanie uruchomiony domyślny program obsługujący pliki z rozszerzeń .pdf oraz wczytany plik pomocy instrukcji programowania

7.2 Sekwencyjna karta funkcji(SFC)

Podstawowe elementy SFC:
- Blok
- Warunek
- Akcja

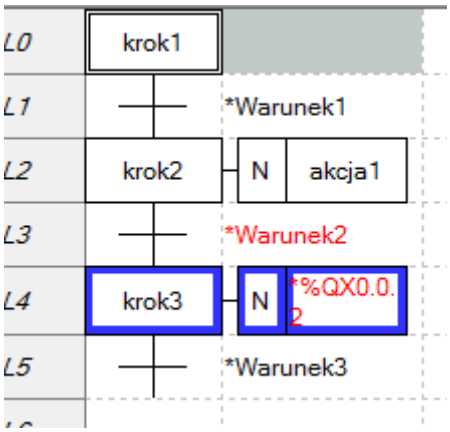
Przykładowy program z użyciem elementów podstawowych:



7.2.1 Opis działania

Krok oznaczony podwójną linią jest krokiem startowym (w tym przypadku Krok1). Przejście do kolejnego bloku następuje po spełnieniu warunku pomiędzy krokiem aktywnym a krokiem kolejnym. Jeżeli aktywowany krok posiadający akcję zostanie ona rozpatrzona.

Po spełnieniu ostatniego warunku programu następuje automatyczne przejście do kroku pierwszego.

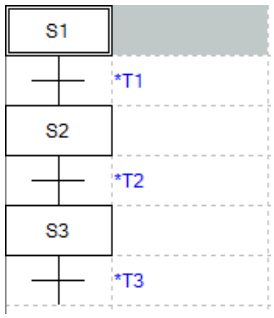


Na powyższym przykładzie aktywny jest krok 3. Po spełnieniu warunku (Warunek3=1) aktywny będzie krok1.

7.2.2 Umieszczanie elementów

Umieszczanie elementów zawsze należy rozpocząć od wiersza L0. Wybór elementów jest możliwy poprzez skrótu klawiszowe lub pasek narzędzi:

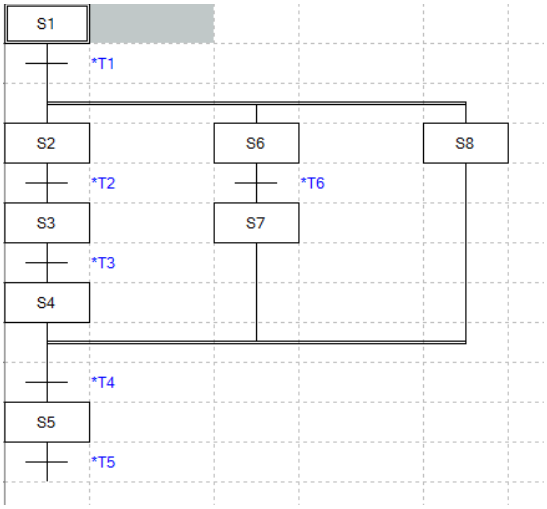
Pierwszym umieszczonym elementem powinien być blok(skrót F3) lub element blokowy (F5). Wybrany element należy ułożyć klikając na polu wiersza L0. Umieszczenie bloku automatycznie tworzy warunek przejścia do kolejnego bloku. Aby dołączyć kolejny krok, po jego wyborze należy kliknąć na istniejący warunek lub blok. Krok zostanie wstawiony bezpośrednio pod warunkiem.



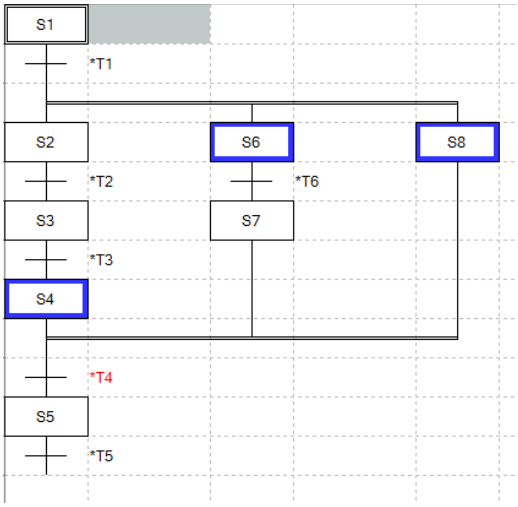
7.2.3 Tworzenie bocznych gałęzi

Gałęzie boczne można utworzyć używając narzędzia Left Branch(F8) lub Right Branch(F9). Istnieją dwa typy gałęzi bocznych:

Gałęzie równoległe -W tym przypadku następuje rozszczepienie gałęzi po spełnieniu pojedynczego warunku. Efektem jest kilka jednocześnie aktywnych gałęzi.
Aby to uzyskać należy zaznaczyć narzędzie wstawiania gałęzi, kliknąć warunek pod którym ma nastąpić rozszczepienie, a następnie kliknąć istniejący krok. Zostanie utworzona gałąź boczna obejmująca elementy od wybranego warunku do wybranego kroku. Poniższy przykład przedstawia gałąź powstałą po wyborze warunku T1 i bloku S4 i ponowne kliknięcie na punkcie rozszczepienia w celu utworzenia trzeciej gałęzi.:



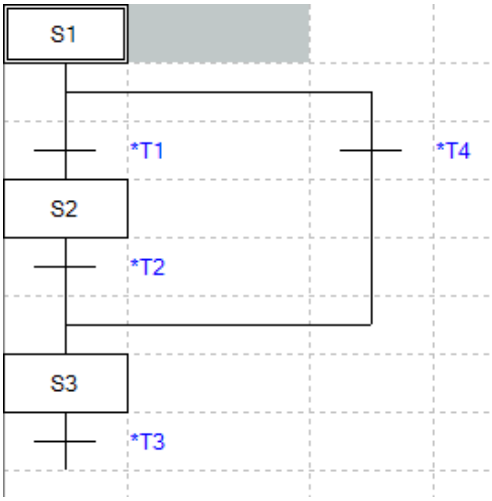
Po spełnieniu warunku T1 aktywowane zostaną kroki S2, S6 oraz S8. Warunkiem wyjścia z gałęzi równoległych i przejścia do S5 jest spełnienie warunku T4 gdy zostaną osiągnięte ostatnie kroki w każdej S4, S7 oraz S8.



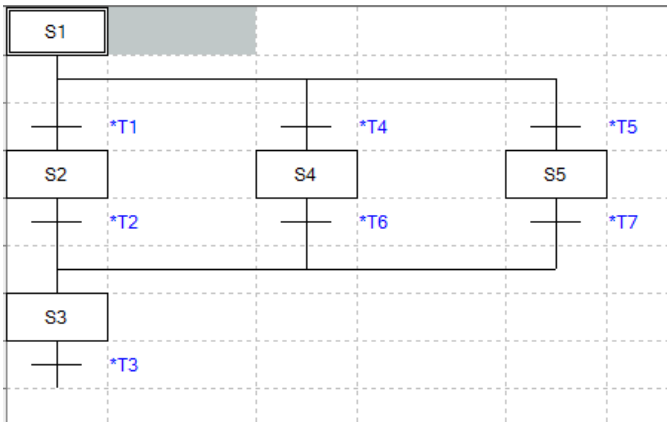
W powyższym przykładzie warunek T4 jest spełniony. Mimo to przejście do S5 nie nastąpi aż osiągnięcia ostatniego kroku w środkowej gałęzi (S7).

Warunki równoległe – W tym przypadku każda równoległa gałąź posiada osobny warunek wejściowy. Gałęzie nie mogą być aktywne jednocześnie i nastąpi wejście do gałęzi z aktywnym warunkiem.

Aby utworzyć warunki równoległe należy wybrać krok pod którym ma nastąpić rozszczepienie oraz na warunek pod którym gałęzie mają się zejść. Poniżysz przykład uzyskano poprzez wybranie narzędzie Right Branch(F9) i kliknięcie na S0 i T2. Warunek w nowej gałęzi nazwano T4.



W tej sytuacji będąc w kroku S1 i spełniając warunek T4 program przejdzie bezpośrednio do S3 z pominięciem S2. Program został rozwinięty poprzez rozszerzenie o kolejną gałąź klikając na istniejące rozszczepienie i dodając kolejne kroki w nowych gałęziach:

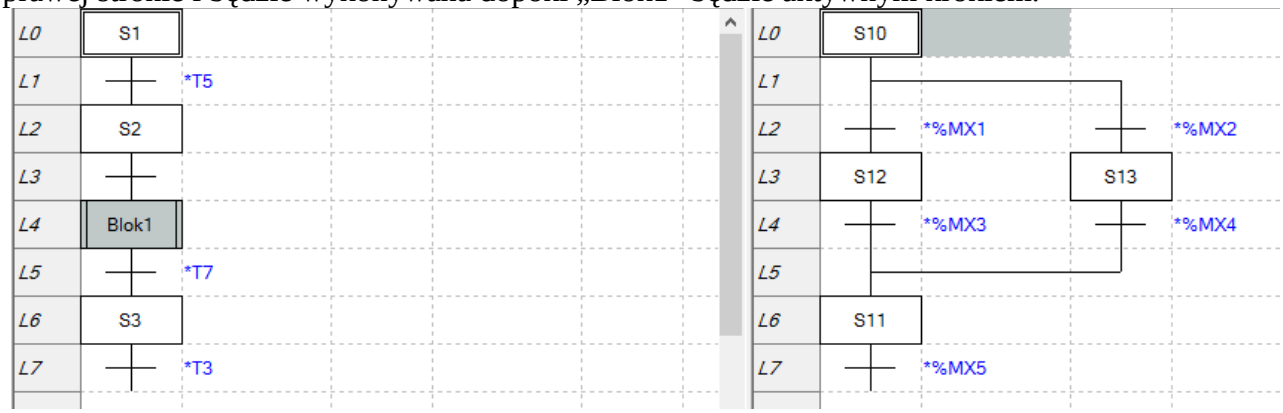


Istnieją trzy drogi przejścia: przez S2, S4 lub S5. W sytuacji jednoczesnego spełnienia kilku równoległych warunków pierwszeństwo ma ten po lewej. Przykład: W przypadku jednoczesnego spełnienia T1 oraz T4 aktywowany zostanie krok S2.

7.2.4 Krok blokowy

Krok blokowy(F5) jest elementem wstawianym w miejsce zwykłego kroku. Jest to krok równoważny z podprogramem. Po jego wstawieniu, nazwaniu i kliknięciu zostanie otworzone równoległe okno programowania SFC. Program zawarty w tym oknie będzie podprogramem wykonywanym przez krok blokowy tak długo jak nie jest spełniony kolejny warunek programu głównego SFC. W momencie spełnienia warunku niezależnie od sytuacji w kroku blokowym program przechodzi do następnego kroku.
Dostęp do wszystkich zdefiniowanych kroków blokowych jest możliwy poprzez kliknięcie prawym klawiszem myszy na pole programu SFC, kliknięcie [Block/Action/Transition List] i wybór zakładki [Block].

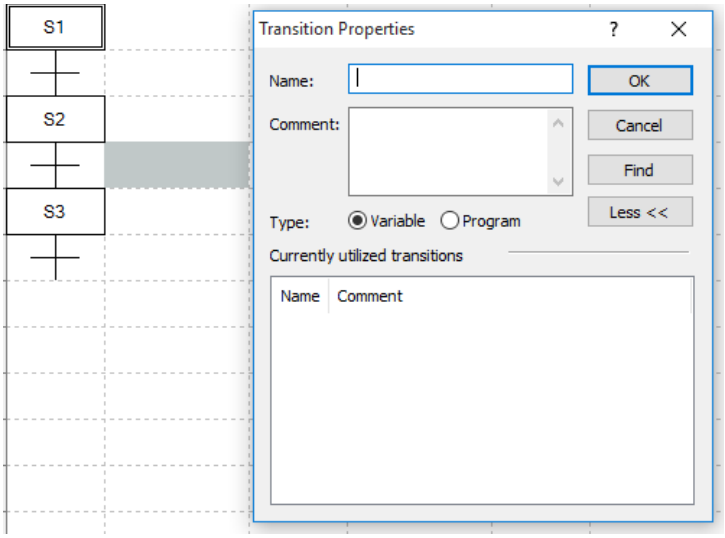
Poniższy przykład przedstawia użycie kroku blokowego „Blok1”. Jego zawartość znajduje się po prawej stronie i będzie wykonywana dopóki „Blok1” będzie aktywnym krokiem.



7.2.5 Warunki

Warunkiem przejścia do kolejnego kroku może być zmienna binarna lub adres bitu pamięci

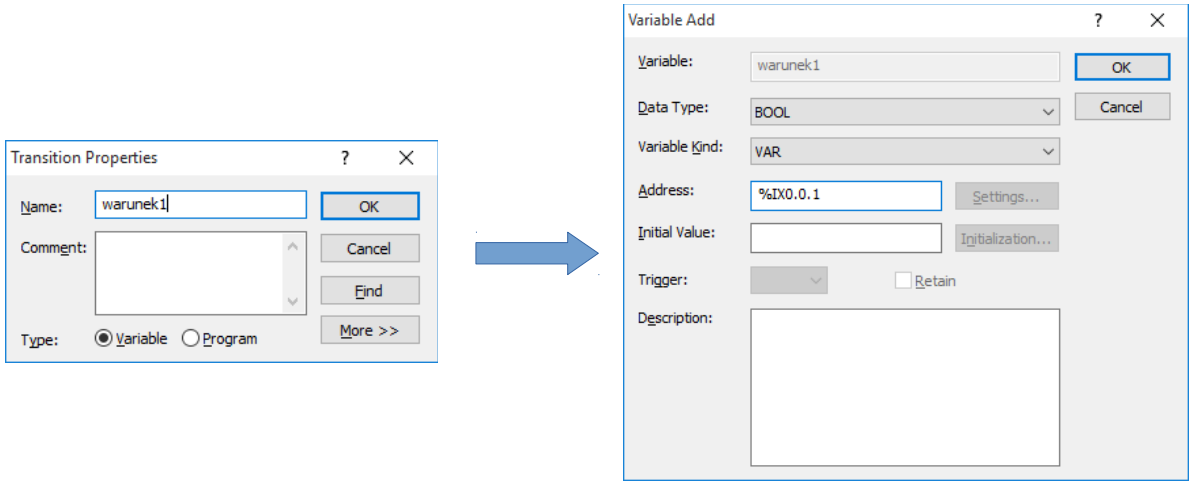
(np. %MX0, %MW2.3%IX0.0.2) albo program definiujący wartość zmiennej wewnętrznej SFC „TRANS”. W celu definicji typu warunku należy dwukrotnie kliknąć na pole po prawej stronie waniu.



Name: W tym polu należy podać nazwę zmiennej/programu lub zdefiniować adres pamięci

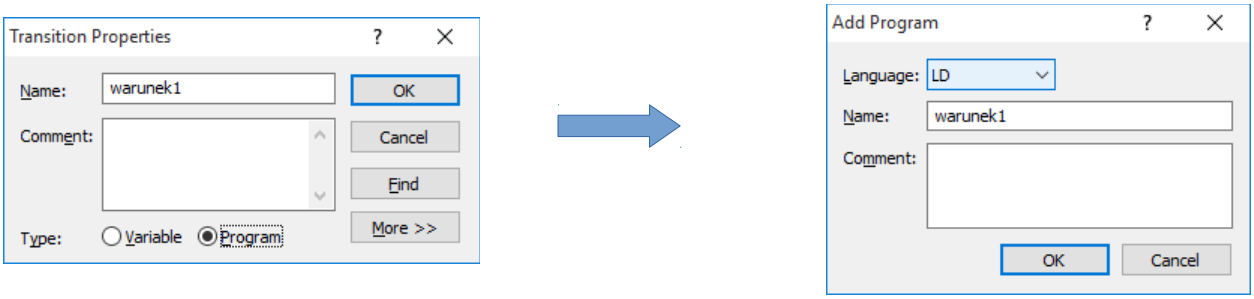
Type: Wybór pola [Variable] spowoduje utworzenie warunku na podstawie zmiennej. Wybór pola program spowoduje utworzenie warunku na podstawie programu. Jeżeli podano nazwę nieistniejącej zmiennej/programu zostanie wyświetlone pole dodawania danego elementu.

Przykład dodania warunku zmiennej warunek1(%IX0.0.1):



Zmienna zostanie automatycznie dodana do zmiennych lokalnych SFC.

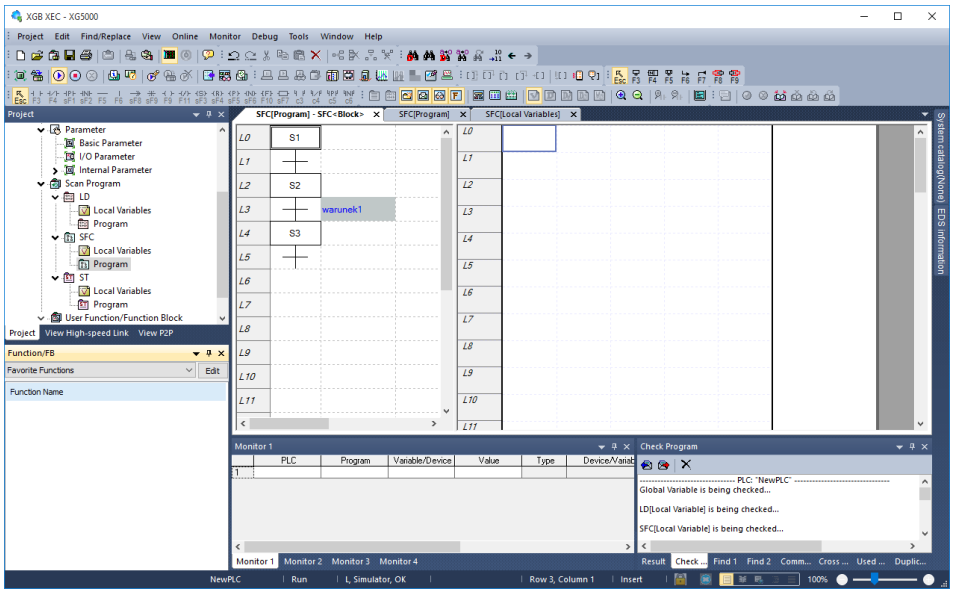
Przykład utworzenia warunku programu:



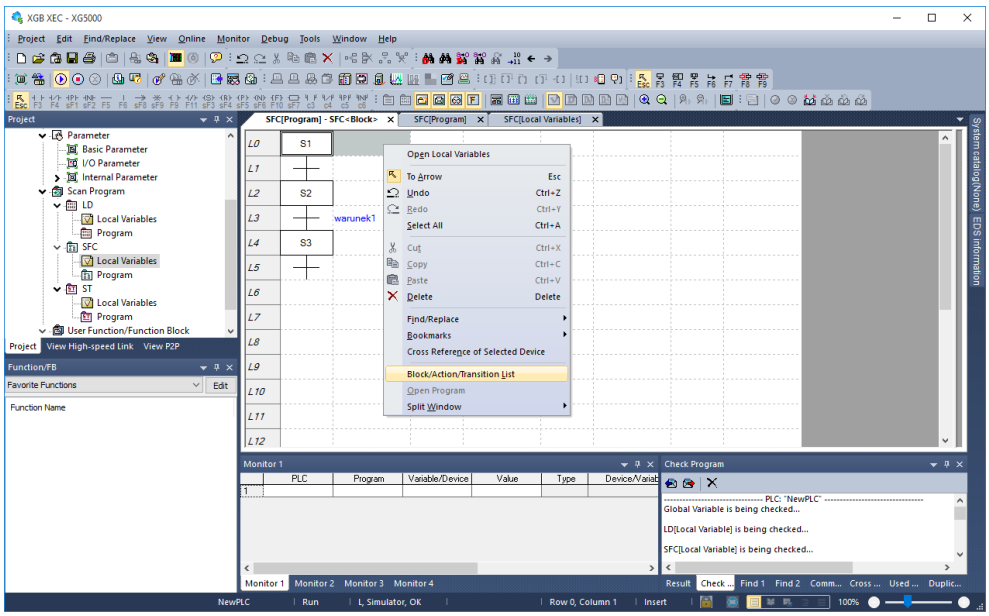
W polu [Language] należy wybrać język programu. Przykład pokazuje programowanie w języku drabinkowym [LD].

Po potwierdzeniu przyciskiem [OK] istnieją dwie możliwości uzyskania dostępu do edycji programu.

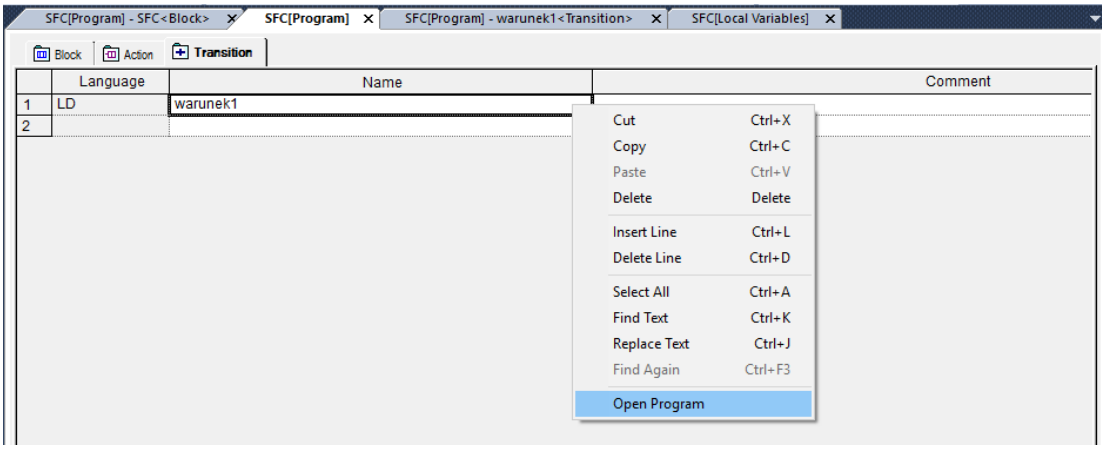
1) kliknięcie na nazwie warunku w programie SFC otwiera równoległe okno programu:



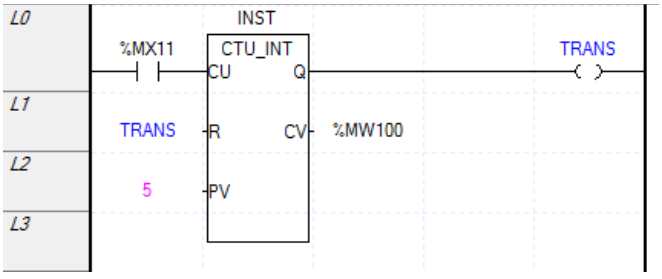
2)Kliknięcie prawym przyciskiem myszy w programie SFC i wybranie [Block/Action/Transition List]:



W otwartym oknie należy przejść do zakładki [Transition] w której znajduje się lista wszystkich programów przejść. Należy wybrać program, kliknąć na niego prawym przyciskiem myszy i wybrać [Open Program]:



Każdy utworzony program musi być poprawny pod kątem składni oraz nie być pusty(**nawet jeżeli nie został użyty jako warunek SFC**) oraz definiować wartość zmiennej TRANS. Nadanie zmiennej TRANS stanu wysokiego(”1”) spowoduje przejście do następnego kroku w SFC. Przykład:



Powyższy program przy każdym zboczu rosnącym adresu %MX11 zwiększy wartość adresu %MW100 o 1. Kiedy %MW100 osiągnie wartość 5 zmienna TRANS uzyska wartość 1 i nastąpi przejście do następnego bloku SFC. Wejście Reset(„R”) powoduje wyzerowanie adresu %MW100 w celu przygotowania dla następnego cyklu.

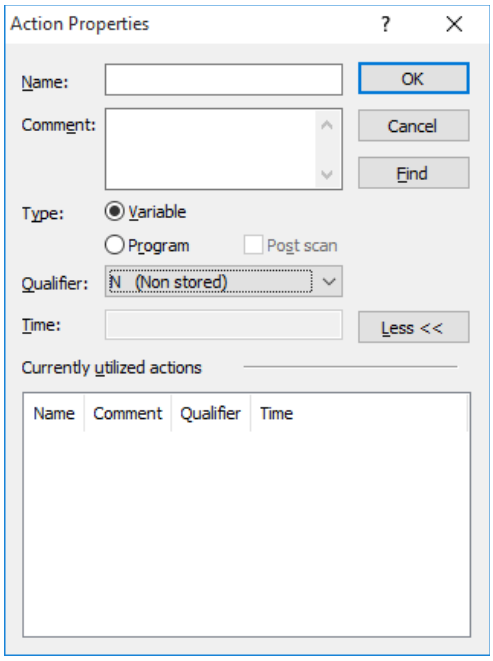
Poniżej zamieszczono program wykonujący to samo zadanie napisany w tekście strukturalnym.

```
1   INST2(CU:=%MX11, R:=TRANS, PV:=5, Q=>TRANS, CV=>%MW100);
```

7.3 Akcje

Akcja może zostać dodana do dowolnego kroku(poza krokiem blokowym). Element ten mówi o akcji podjętej w sytuacji aktywowania danego bloku. Należy go dodać poprzez wybranie elementu Action(F4) z paska narzędzi i kliknięcie na istniejącym kroku. Do jednego kroku można dodać wiele akcji.

7.3.1 Właściwości akcji



Name	Nazwa akcji
Comment:	Komentarz
Type:	Typ akcji (Zmienna/Program)
Qualifier:	Metoda wykonywania akcji
Time:	Czas zdefiniowany dla metod czasowych
Currenty utilized actions	Wcześniej zdefiniowane akcje

7.3.2 Metoda akcji

Każda z akcja posiada określoną metodę działania(Qualifier). Dostępny metody:

N) (nieprzechowywana)

Akcja zostanie wykonana dopóki krok jest aktywny.

S) (Set)

Akcja jest wykonywana do momentu wystąpienia akcji R (Reset).

R (Reset)

Zatrzymuje wykonanie akcji S, SD, SL lub DS.

L(Limit czasowy)

Akcja jest wykonywana przez określony czas.

D(Opóźnienie czasowe)

Wykonanie akcji jest rozpoczęte po upływie określonego czasu.

P (Puls)

Akcja zostaje wykonana tylko raz (w jednym skanie).

SD (Opóźniona i przechowana)

Wykonanie akcji jest rozpoczęte po upływie określonego czasu. Wykonywanie jest kontynuowane do czasu do wystąpienia akcji R(Reset).

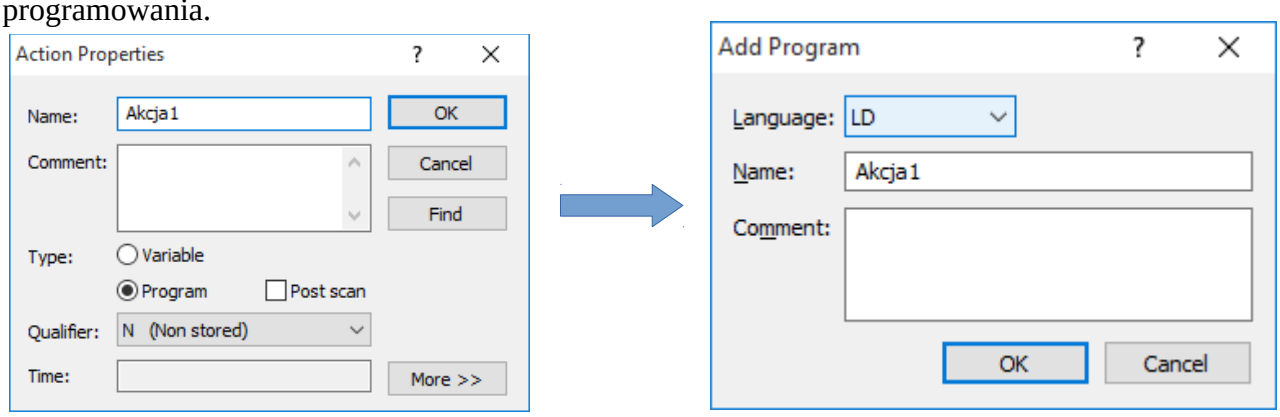
SL (Opóźniona i przechowana)

Akcja jest wykonywana do momentu wystąpienia akcji R (Reset) lub upłynięcia określonego czasu.

7.3.3 Typ Akcji

Akcja typu [Variable] wiąże się z ustawieniem stanu zmiennej binarnej.

Akcja typu [Program] jest nowym podprogramem utworzonym w jednym z dostępnych języków(LD, SFC, ST). Po wybraniu typu [Program] i wciśnięciu OK należy wybrać język programowania.



Dostęp do utworzonego podprogramu może być uzyskany poprzez kliknięcie elementu akcji lub kliknięcie prawym przyciskiem myszy w programie SFC i wybranie [Block/Action/Transition List]:W otwartym oknie należy przejść do zakładki [Action] w której znajduje się lista wszystkich programów akcji. Należy wybrać program, kliknąć na niego prawym przyciskiem myszy i wybrać [Open Program].

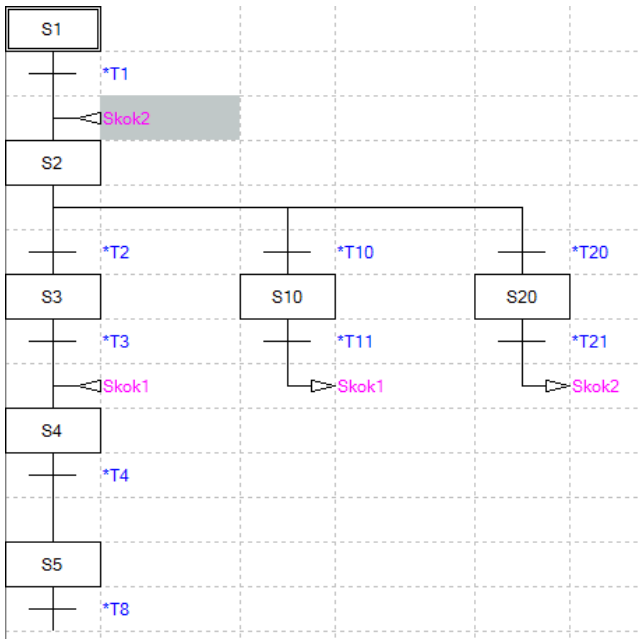
7.3.4 Skok

Skok pozwala na zmianę aktywnego kroku bez potrzeby zachowania ciągłości. Aby dokonać skoku należy użyć dwóch elementów oraz nadać im tę samą nazwę.

- Etykiety – Label(F6)
- Skoku – Jump(F7)

Skok jest wykonywany od elementy skoku do elementu etykiety. Element skoku Jump(F7) musi zostać umieszczony pod warunkiem przejścia na końcu gałęzi. Umieszcza się go poprzez kliknięcie na ostatnim warunku gałęzi. Etykieta zostaje umieszczona ponad istniejącym krokiem poprzez kliknięcie na wybrany krok.

Oba elementy muszą zostać nazwane. Zmiana nazwy może zostać dokonana poprzez dwukrotne kliknięcie na umieszczonym elemencie. Skok zostanie wykonany pomiędzy elementami o tej samej nazwie. Przykład:



Jeżeli aktywny jest krok S20 i warunek T21 zostanie spełniony kolejnym aktywowanym krokiem będzie S2.

7.4 Tekst Strukturalny(ST)

Język strukturalny jest językiem tekstowym wysokiego poziomu umożliwiającym tworzenie skomplikowanych wyrażeń i algorytmów. Jest szczególnie wygodny dla osób z doświadczeniem w innych językach programowania np. C, Cpp, Pascal.

7.4.1 Komentarze

Umieszczenie komentarza w kodzie jest możliwe poprzez wstawienie „//”(komentarz obejmuje tekst do końca linii) lub zastosowanie „(*” jako początek komentarza i „*)” jako jego koniec(może dotyczyć wielu linii). Przykład:

```
1   //linia komentarza
2   (*komentarz
3   złożony z wielu
4   linii*)
5
```

7.4.2 Wyrażenia

- Każde wyrażenie posiada operator (np. +,-,*,/) oraz operand(stała, funkcja, ciąg znaków, zdefiniowana zmienna, adres pamięci itd.).
- W przypadku kilku operatorów o tym samym priorytecie są one wykonywane po kolei od lewej strony. Np. A+B-C najpierw doda A i B a następnie od wyniku dodawania odejmie C.
- Jeżeli operator posiada kilka operandów są one rozpatrywane od lewej
- Np. SIN(A)*COS(B) najpierw wykona SIN(A), następnie COS(B)
- Następujące operacje spowodują błąd:

  Dzielenie przez 0,

  Operand jest niepoprawnego lub niezdefiniowanego typu (Np. parametry funkcji ADD(1,2,3)nie mają zdefiniowanego typu, więc wyrażenie spowoduje błąd kompilacji),

  Przekroczenie zakresu zmiennej np. próba zapisania wartości większej niż 65,535 w zmiennej typu UINT.

Numer	Operacja	Symbol
1	Nawias	( )
2	Funkcja	Nazwa_funkcji(parametry) Np. ADD(X,Y)
3	negacja	NOT
4	potęgowanie	**
5	Mnożenie Dzielenie Reszta z dzielenia	* / MOD
6	Dodawanie Odejmowanie	+ -
7	Porównanie	<.>,<=,>=
8	Równy Nierówny	= <>
9	AND	& AND
10	XOR	XOR
11	OR	OR

7.4.3 Terminator instrukcji

Każda instrukcja powinna być zakończona terminatorem instrukcji „ ; ” Jedną instrukcję można zapisać w kilku liniach pod warunkiem poprawnego użycia terminatora. Przykład jednej instrukcji w dwóch liniach:

```
%MW1:=ADD(IN1:=%MW2, IN2:=%MW3, IN3:=%MW4,
IN4:=%MW5, IN5:=%MW6, IN6:=%MW7);
```

7.4.4 Instrukcja przypisania

Instrukcja przypisania powoduje przypisanie pewnej wartości do lokacji. Jest ona używana za każdym razem gdy nadawana jest wartość zmiennej lub adresu pamięci. Przykłady:

```
%MX1:=1;
%MW50:=30;
D:=E-100;
%MW100:=ABS(IN:=%MW100);
```

Nie należy mylić przypisania z operacją porównania „=” której wynikiem jest 0 (nie równe) lub 1(równe). Przykładowo zastosowanie przypisania w miejscu podania warunku pętli spowoduje błąd kompilacji programu.

7.4.5 Instrukcja warunkowa IF

Instrukcja składa się z kilku podstawowych elementów:

- IF „Warunek” - Warunek który determinuje wykonanie akcji
- THEN „Akcja”; - akcja zostanie wykonana jeżeli warunek ma wartość 1
- END_IF; Zakończenie instrukcji

Przykład:

```
IF %MX4=0
THEN INST_TON(IN:=%MX0, PT:=T#15s, Q=>%MX5, ET=>A);
END_IF;
```

Opcjonalne elementy instrukcji IF:

- ELSE „Akcja”; - akcja zostanie wykonana gdy warunek początkowy ma wartość 0
- ELSIF „Warunek”- jest to zagnieżdżenie kolejnej funkcji IF które jest rozpatrzone jeżeli warunek główny nie został spełniony.

Przykład 1:

```
IF %MX4=0
THEN Stan_alarmu:=0;
ELSE Stan_alarmu:=1;
END_IF;
```

Przykład 2:

```
IF %MX4=0
THEN Stan_alarmu:=0;
ELSIF %MX5=1
THEN Stan_alarmu:=2;
END_IF;
```

Przykład 3:

```
IF %IX0.0.1=1 AND %IX0.0.2=1
THEN
    Stan_alarmu:=1;
ELSIF %IX0.0.1=1
THEN
        Stan_alarmu:=2;
ELSIF %IX0.0.2=1
THEN
    Stan_alarmu:=3;
ELSE Stan_alarmu:=0;
END_IF;
```

Należy pamiętać, że instrukcje można zapisać w innej formie pod warunkiem zachowania kolejności i struktury. Przykład 2 można zapisać w następującej postaci:

```
IF %MX4=0 THEN
    Stan_alarmu:=1;
    ELSIF %MX5=1 THEN
        Stan_alarmu:=2;
END_IF;
```

7.4.6 Instrukcja warunkowa CASE

Instrukcja CASE wykonuje jedną z akcji zależnie od wartości wyrażenia wejściowego. Element Lista: Akcja składa się z minimum 1 linii przypisującej akcje do wartości wyrażenia. Opcjonalnie może zostać umieszczony element ELSE wykonujący akcję jeżeli żaden element z listy nie odpowiada wartości wyrażenia.. Funkcję kończy się komendą END_CASE.

CASE 'Wyrażenie' OF
'Lista': 'Akcja'
[ELSE
'Lista': 'Akcja']
END_CASE

Element w nawiasie kwadratowym jest opcjonalny. Przykład:

```
CASE Wejscie1 OF
    1 : Wynik:=10;
    2 : Wynik:=15;
    3,4 : Wynik:=25;
ELSE
    Wynik:=0;
END_CASE;
```

Powyższa instrukcja nadaje zmiennej wynik jeden z czterech stanów:

- Wynik:=10 jeżeli Wejście1=1
- Wynik:=15 jeżeli Wejście1=2
- Wynik:=25 jeżeli Wejście1=3 lub Wejście1=4
- Wynik:=0 jeżeli Wejście1 posiada inną wartość.

7.4.7 Pętla FOR

Instrukcja FOR jest jedną w pętli obsługiwanych przez ST. Każda wykonana iteracja wiąże się ze zmianą wartości zdefiniowanego licznika. Przed wykonaniem kolejnej iteracji następuje sprawdzenie czy licznik spełnia zadany warunek. Jeżeli go nie spełni akcja w pętli nie zostanie wykonana i program przejdzie do następnej instrukcji. Przykład:

```
FOR licznik1:=1 TO 5 DO
    Stan_alarmu:=Stan_alarmu*5;
END_FOR;
```

Pętla wykona mnożenie wartości zmiennej Stan_alarmu pięciokrotnie.

```
FOR licznik1:=0 TO 10 BY 2 DO
    Stan_alarmu:=Stan_alarmu+5;
END_FOR;
```

Powyższa pętla wykona dodawanie 6 razy. Warunek BY definiuje inkrementacje zmiennej licznika inną niż jeden.

Przy użyciu instrukcji pętli należy zwrócić szczególną uwagę na ilość iteracji. Zbyt duża ilość powtórzeń spowoduje zadziałanie zabezpieczenia Watchdog.

7.4.8 Pętla WHILE

Pętla wykonuje zawartą w niej instrukcje tak długo jak postawiony warunek jest prawdziwy. Format pętli:

WHILE 'Warunek' DO
'Akcja'
END_WHILE

Przykład:

```
licznik1:=0;
WHILE licznik1<20 DO
    licznik1:=licznik1+1;
END_WHILE;
```

Pętla będzie zwiększać wartości licznika aż do osiągnięcia wartości 20. Należy zwrócić uwagę na postawiony warunek. W przypadku gdy warunek nie zostanie spełniony program nie wyjdzie z pętli i dojdzie do uruchomienia zabezpieczenia Watchdog.

7.4.9 Pętla REPEAT

Petla wykonuje akcje tak długo jak warunek nie jest spełniony(„powtarzaj do czasu spełnienia warunku”). Format pętli:

REPEAT
'Akcja'
UNTIL 'Warunek'
END_REPEAT

Przykład:

```
licznik1:=0;
REPEAT
    licznik1:=licznik1+1;
UNTIL licznik1>20
END_REPEAT;
```

Program będzie zwiększał wartość licznika do czasu osiągnięcia wartości większe od 20(w stym przypadku pętla zostanie zakończona dla wartości 21). Tak jak w innych pętlach wywołanie pętli nieskończonej lub o bardzo dużej ilości iteracji spowoduje zadziałanie zabezpieczenia Watchdog.

7.4.10 EXIT

Instrukcja EXIT powoduje wymuszenie wyjścia z pętli bez względu na postawiony warunek. Można ją stosować jako akcje wykonywaną w dowolnej instrukcji pętli (FOR, WHILE, REPEAT).
Przykład:

```
licznik1:=0;
WHILE licznik1<20 DO
licznik1:=licznik1 + 1 ;
IF licznik1 = 10 THEN
EXIT;
END_IF;
END_WHILE ;
```

W powyższym przykładzie pętla będzie zwiększać wartość licznika co 1 oraz sprawdzać, czy licznik osiągnął wartość 10. W przypadku osiągnięcia wartości 10 zostanie aktywowana instrukcja EXIT i pętla zostanie zakończona pomimo spełniania warunku wykonywania pętli(licznik1<20).

7.4.11 Funkcje i bloki funkcyjne

Funkcje w ST mogą zostać zdefiniowane ręcznie w całości lub wybrane z menu funkcji języka LD (F10) – zostanie wtedy wygenerowany szkielet funki z komentarzami podpowiadającymi typy wejść i wyjść. Poniżej pokazano funkcje ADD wybraną z menu funkcji:

```
6   (*ANY_NUM*):=ADD(IN1:=(*ANY_NUM*), IN2:=(*ANY_NUM*))
```

Argumenty funkcji mogą być podane jeden po drugim po przecinku w stałej kolejności lub przypisane do nazw wejść i wyjść funkcji. Przykład:

```
INST_TON(%MX0, T#15s, %MX5, A);
INST_TON(IN:=%MX0, PT:=T#15s, Q=>%MX5, ET=>A);
```

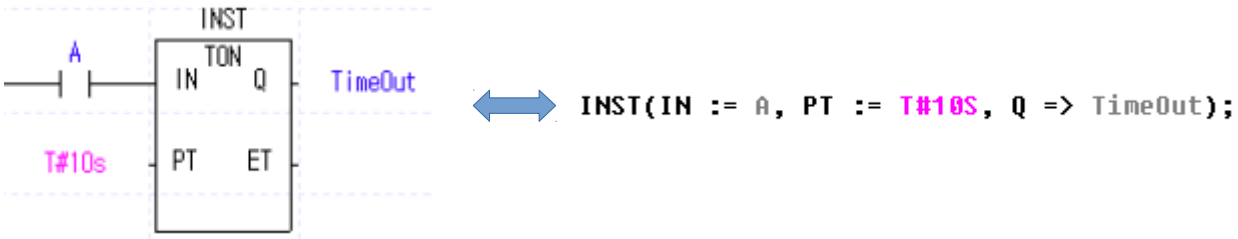
Argumenty w linii drugiej mogą być zamienione miejscami pod warunkiem zachowania nazw We/Wy. Elementy typu := (Np. IN:=) są wejściami, elementy typu => (Np. Q=> są wyjściami).

Jeżeli funkcja posiada argumenty, ich typ powinien zostać zdefiniowany. Przykład:

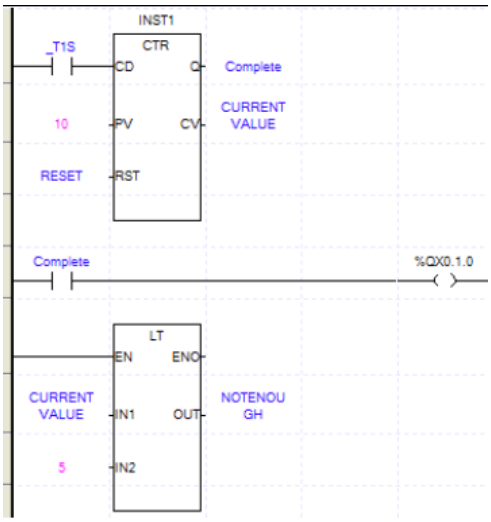
```
licznik1:=ADD(INT#1,2,3);
licznik1:=ADD(B,2,3);
licznik1:=ADD3_INT(1,2,3);
```

W pierwszej linii typ stałych został zadeklarowany poprzez użycia przedrostka „INT#”. W drugiej linii typ stałych został dopasowany do typu zmiennej B zadeklarowanej jako INT. W trzeciej linii użyto funkcji sumującej 3 elementy typu INT.

Poniżej znajduje się porównanie deklaracji bloku funkcyjnego w języku LD i ST:



Przykład prostego programu:



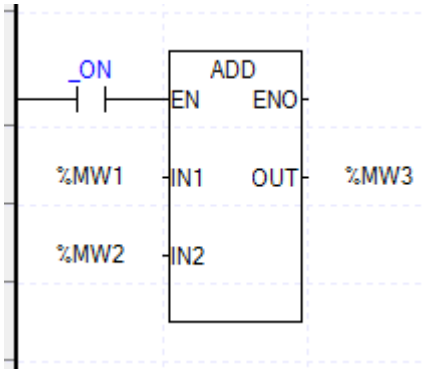

```
INST1(CD:=_T1S, PV:=10, RST:= RESET,
Q=>COMPLETE, CV=>CURRENTVALUE);

%QX0.1.0:=Complete;

NOTENOUGH:=LT(IN1:=CURRENTVALUE, IN2:=5);
```

8 Funkcje i bloki funkcyjne użytkownika

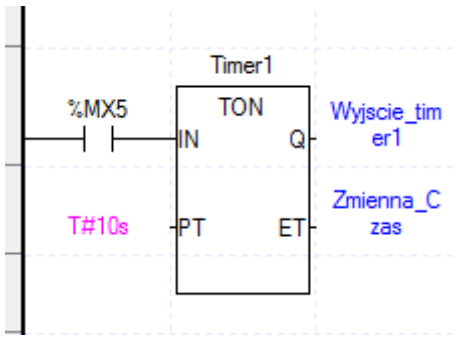
Funkcja jest zbiorem instrukcji posiadającym wejścia i wyjścia. Użycie funkcji w programie powoduje jej wykonanie i zwrócenie przez nią wartości podczas danego cyklu programu. Dodatkowo funkcje nie mogą korzystać ze zmiennych globalnych

Blok funkcyjny jest zbiorem instrukcji rozpatrywanym w sposób ciągły przez wiele skanów programu. Podczas każdego skanu programu blok funkcyjny jest aktualizowany i w momencie zakończenia pracy lub spełnienia postawionych warunków zmienia się stan jego wyjścia. Każdy blok funkcyjny musi posiadać swoją nazwę(identyfikator).

Przykład:



ADD jest funkcją. Program sprawdza, czy na wejście jest podany stan wysoki. Jeżeli tak, funkcja wykonuje działanie, wynik dodawania jest zapisany w adresie wyjściowym (%MW3) i sygnał EN jest przekazany na wyjcie ENO.



Elementy takie jak timery czy liczniki to funkcje blokowe. Powyższy przykład przedstawia użycie timera typu TON. Jeżeli wejście %MX5 zostanie aktywowane timer rozpoczyna zliczanie czasu. Czas jest zliczany tak długo jak wejście posiada stan wysoki(przez wiele cykli). W momencie gdy czas zliczania jest większy niż czas PT timer ustawia stan wysoki wyjścia Q. Reset bloku jest możliwy poprzez zdjęcie sygnału %MX5. Dodatkowo zmienne bloku funkcyjnego mogą być powiązane z innymi elementami programu. Odwołanie do zmiennej bloku wygląda następująco:

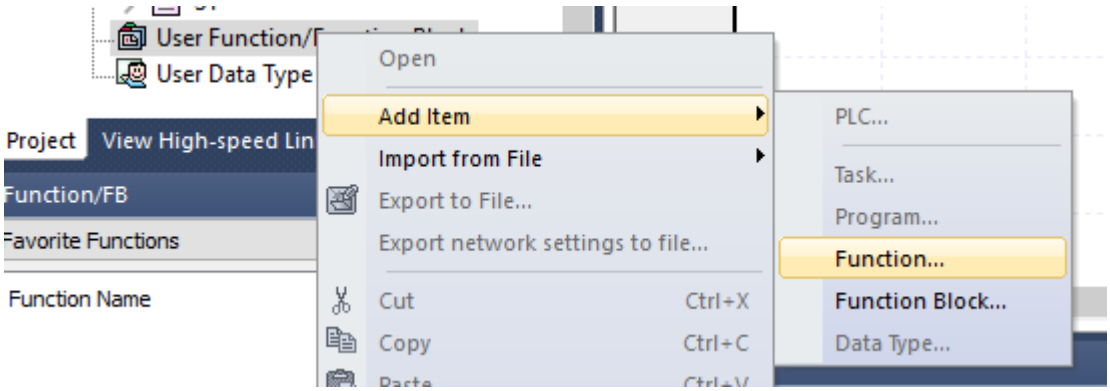
NAZWA_BLOKU.NAZWA_ZMIENNEJ

Na przykład chcąc wykorzystać czas zliczony przez powyższy blok funkcyjny jako wejście innego bloku należy użyć zmiennej:

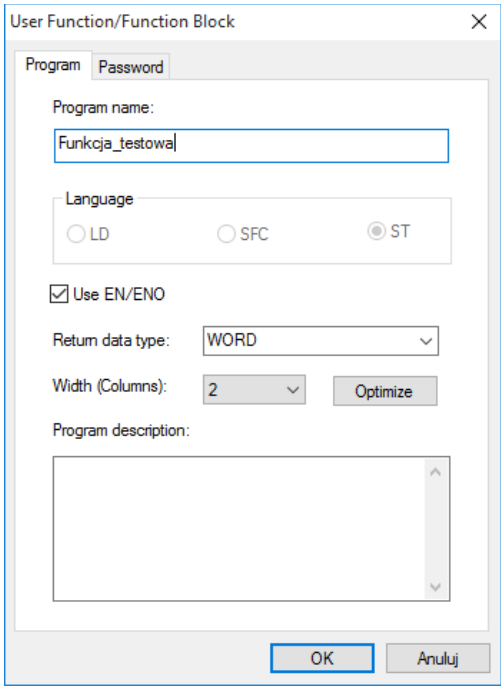
Timer1.ET

8.1 Tworzenie funkcji użytkownika

Tworzenie funkcji lub bloku użytkownika odbywa się poprzez kliknięcie prawym przyciskiem myszy na elemencie projektu [User Function/Function Block] i wybranie z listy [Add Item] elementu funkcji lub bloku funkcyjnego. Poniższy przykład przedstawia utworzenie funkcji:



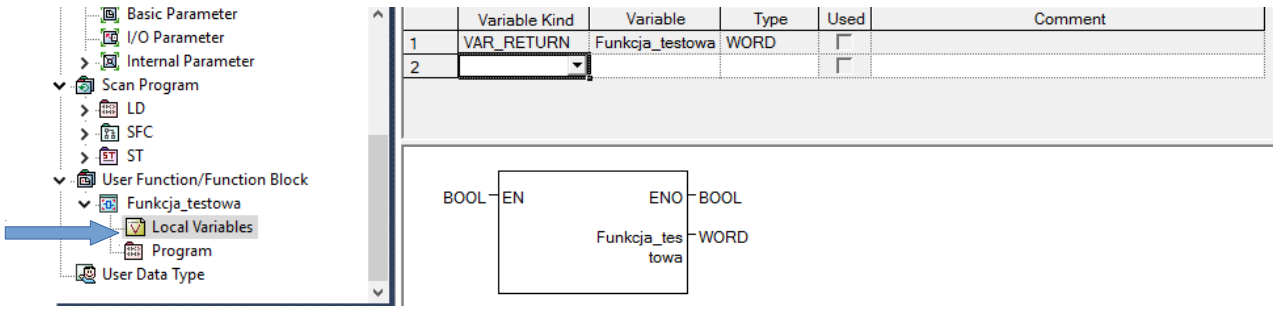
Okno tworzenia funkcji wygląda następująco:



Program name	Nazwa programu
Language	Język programowania
Use EN/ENO	Po zaznaczeniu blok będzie posiadał wejście EN i wyjście ENO
Return data type	Określenie typu danych wyniku funkcji
Width (Colums)	Określa szerokość bloku funkcji

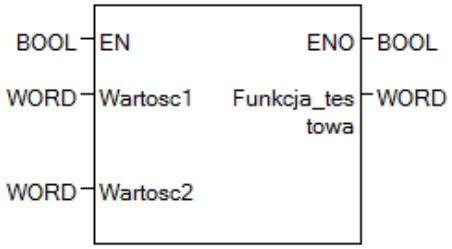
8.1.1 Definiowanie wejść i wyjść

Aby określić wejścia i wyjścia funkcji należy przejść do zmiennych lokalnych utworzonej funkcji:



W tabeli zmiennych lokalnych od razu zdefiniowana jest zmienna wyniku o takiej nazwie jak nazwa funkcji i typie VAR_RETURN. W poniższym przykładzie utworzona zostanie funkcja dodawania.

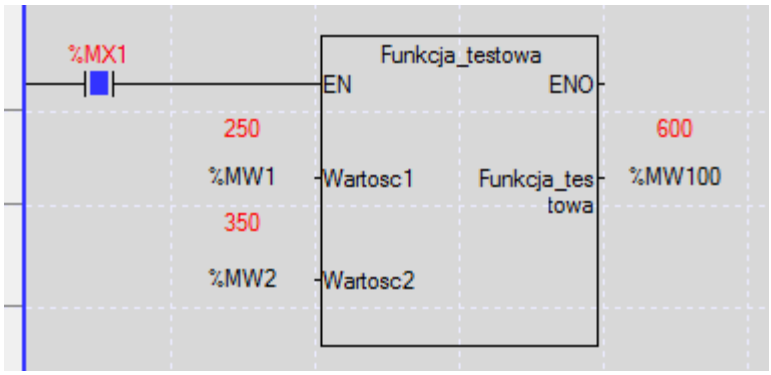
	Variable Kind	Variable	Type	Used
1	VAR_RETURN	Funkcja_testowa	WORD	☐
2	VAR_INPUT	Wartosc1	WORD	☐
3	VAR_INPUT	Wartosc2	WORD	☐



Program funkcji dodawania:

```
1  Funkcja_testowa:=Wartosc1+Wartosc2;
2  RETURN ;
```

Ta prosta funkcji przypisuje zmiennej wyjściowej wartość równą sumie wartości wyjściowych. Wywołanie instrukcji RETURN kończy funkcję i nadaje wyjściu wartość. Przykład użycia fukncji:

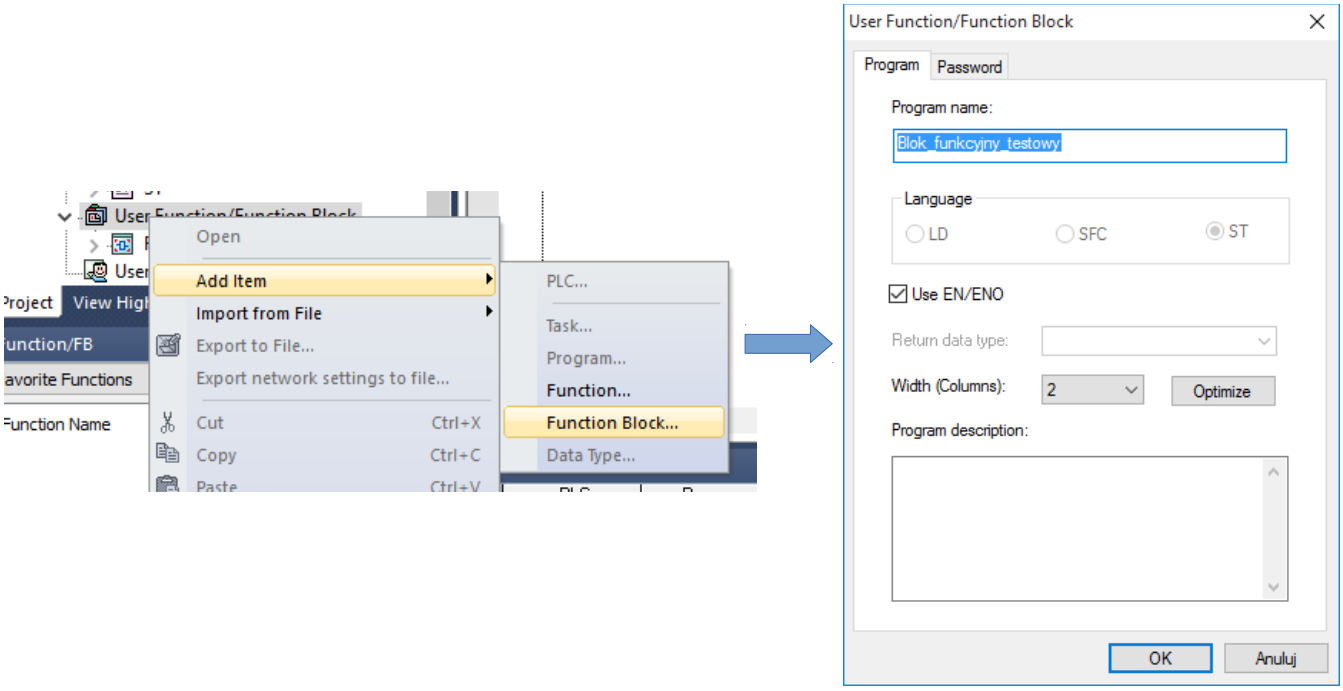


Dzięki monitorowi pracy PLC widać, że funkcja działa poprawnie. W adresie %MW100 została zapisana suma adresów podanych jako wejścia Wartosc1 oraz Wartosc2.

Należy pamiętać, że w programie funkcji użytkownika można korzystać w funkcji XG5000, jednak bloki funkcyjne są niedostępne ze względy na ich metodę działania.

8.2 Tworzenie bloku funkcyjnego użytkownika

Tworzenie bloku funkcyjnego przebiega tak samo jak funkcji:



Przykładowy blok funkcyjny będzie sprawdzał, czy wartość wejściowa czasu mieści się w podanym zakresie. Zmienne bloku funkcyjnego:

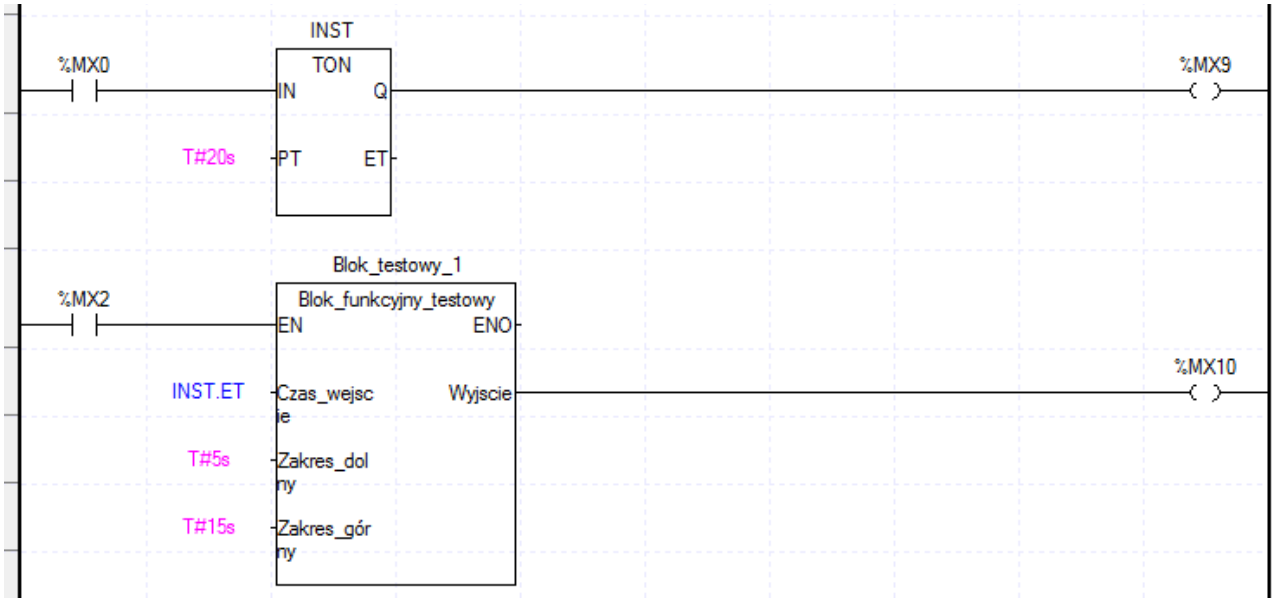
	Variable Kind	Variable	Type	Trigger	Address	Initial Value	Retain	Used
1	VAR_INPUT	Czas_wejscie	TIME				☐	☑
2	VAR_INPUT	Zakres_dolny	TIME				☐	☑
3	VAR_INPUT	Zakres_górny	TIME				☐	☑
4	VAR_OUTPUT	Wyjscie	BOOL				☐	☑

Program bloku funkcyjnego:

```
IF Czas_wejscie>Zakres_dolny AND Czas_wejscie<Zakres_górny THEN
    Wyjscie:=1;
ELSE
    Wyjscie:=0;
END_IF;
```

W przypadku bloku funkcyjnego nie używamy instrukcji RETURN.

Użycie utworzonego bloku w programie głównym:



W powyższym przykładzie Nasz blok funkcyjny sprawdza, czy czas zliczony przez funkcję TON znajduje się w zakresie 5-15s. Jeżeli tak, wyjście ma stan 1. W tym celu jako wejście bloku użyto adresu bloku timera INST.ET gdzie 'INST' jest nazwą bloku a 'ET' zmienną użytą w tym bloku. W ten sam sposób można powoływać się na zmienne bloku funkcyjnego użytkownika posługując się zapisem:

Nazwa_bloku.Zmienna

Rewizja	Data
V1.0	25.01.2016