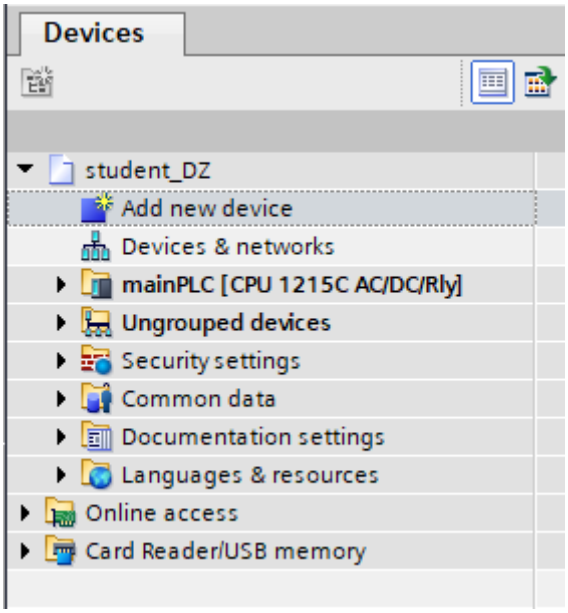
Instrukcja dla stanowiska FESTO.

Obsługa środowiska.

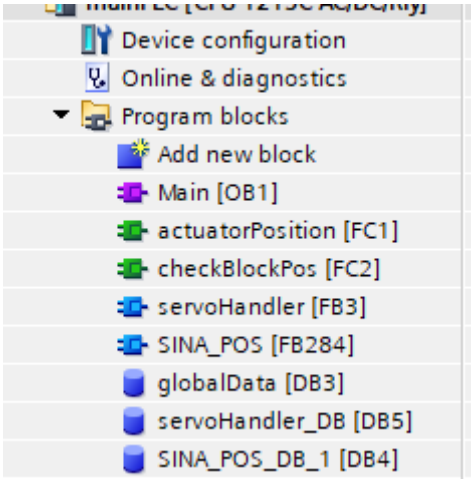
1. Uruchomienie TIA-Portal

oraz wybranie projektu startowego. Projekt ten powinien być zapisany pod własną nazwą grupy i tylko taki edytowany.

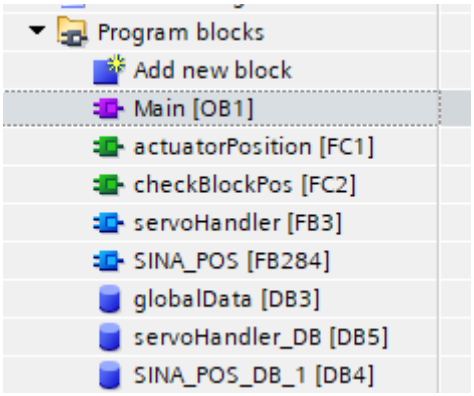
2. We własnej kopii projektu startowego otwórz MainPLC



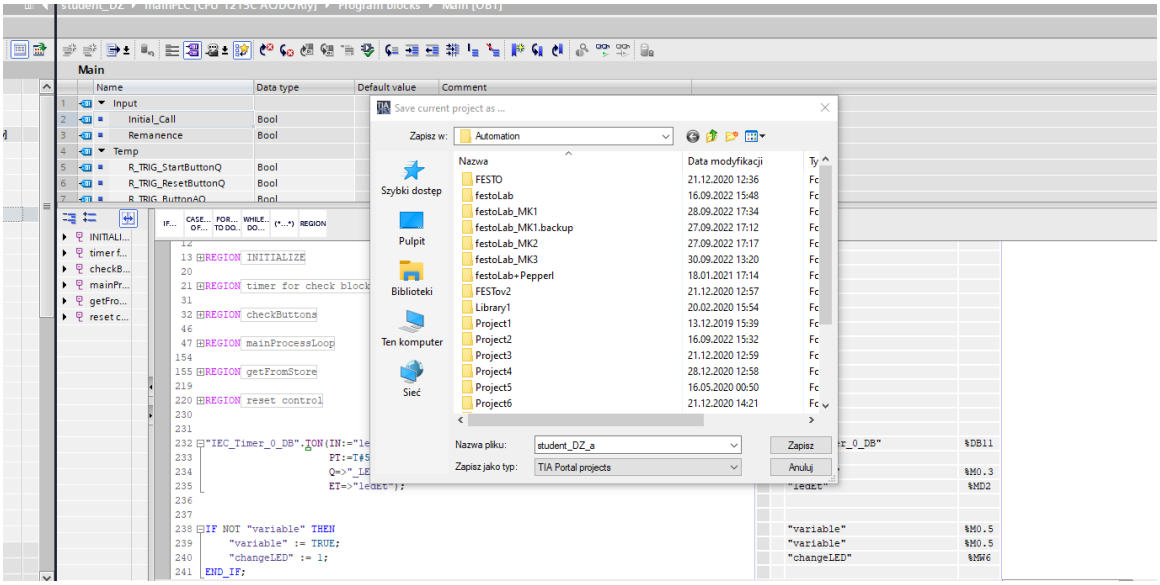
3. Rozwinąć Program blocks



4. Wybrać Main [OB1] – tu dokonywana jest edycja programu.

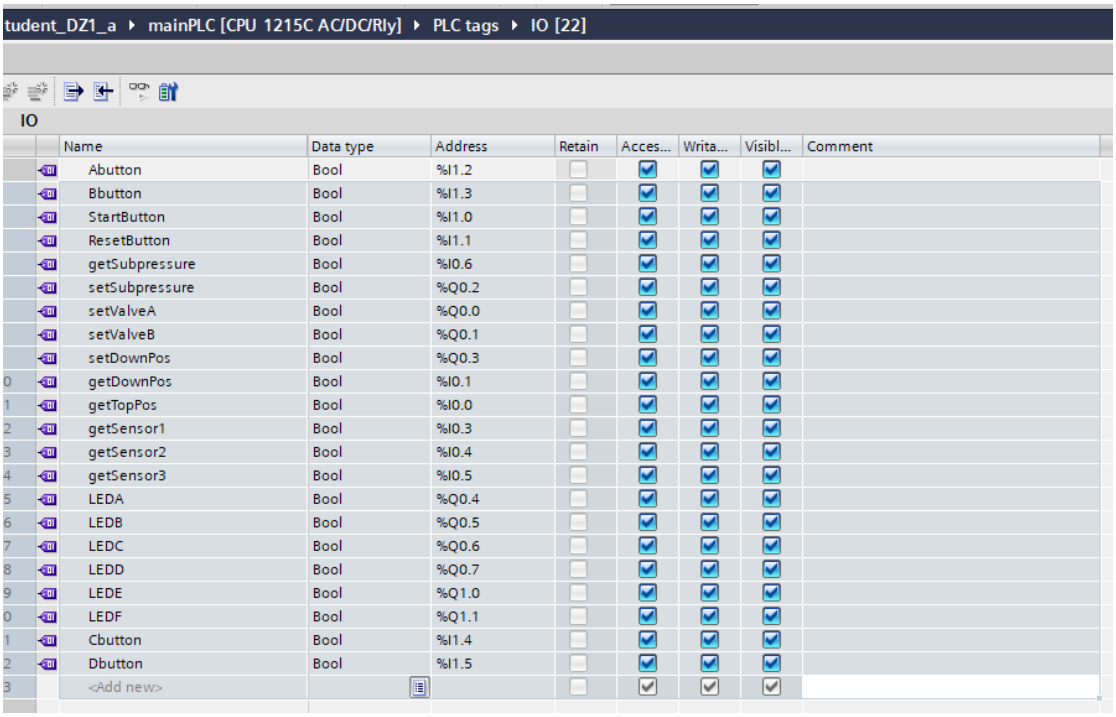


Należy koniecznie zapisywać zmiany edycyjne w projekcie pod własną nazwą



6. Wprowadzić zmiany w programie

Przy edycji należy wykorzystywać PLC tags.



	Name	Data type	Address	Retain	Acces...	Writa...	Visibl...	Comment
	Abutton	Bool	%I1.2		☑	☑	☑	
	Bbutton	Bool	%I1.3		☑	☑	☑	
	StartButton	Bool	%I1.0		☑	☑	☑	
	ResetButton	Bool	%I1.1		☑	☑	☑	
	getSubpressure	Bool	%I0.6		☑	☑	☑	
	setSubpressure	Bool	%Q0.2		☑	☑	☑	
	setValveA	Bool	%Q0.0		☑	☑	☑	
	setValveB	Bool	%Q0.1		☑	☑	☑	
	setDownPos	Bool	%Q0.3		☑	☑	☑	
0	getDownPos	Bool	%I0.1		☑	☑	☑	
1	getTopPos	Bool	%I0.0		☑	☑	☑	
2	getSensor1	Bool	%I0.3		☑	☑	☑	
3	getSensor2	Bool	%I0.4		☑	☑	☑	
4	getSensor3	Bool	%I0.5		☑	☑	☑	
5	LEDA	Bool	%Q0.4		☑	☑	☑	
6	LEDB	Bool	%Q0.5		☑	☑	☑	
7	LEDC	Bool	%Q0.6		☑	☑	☑	
8	LEDD	Bool	%Q0.7		☑	☑	☑	
9	LEDE	Bool	%Q1.0		☑	☑	☑	
0	LEDF	Bool	%Q1.1		☑	☑	☑	
1	Cbutton	Bool	%I1.4		☑	☑	☑	
2	Dbutton	Bool	%I1.5		☑	☑	☑	
3	<Add new>				☑	☑	☑	

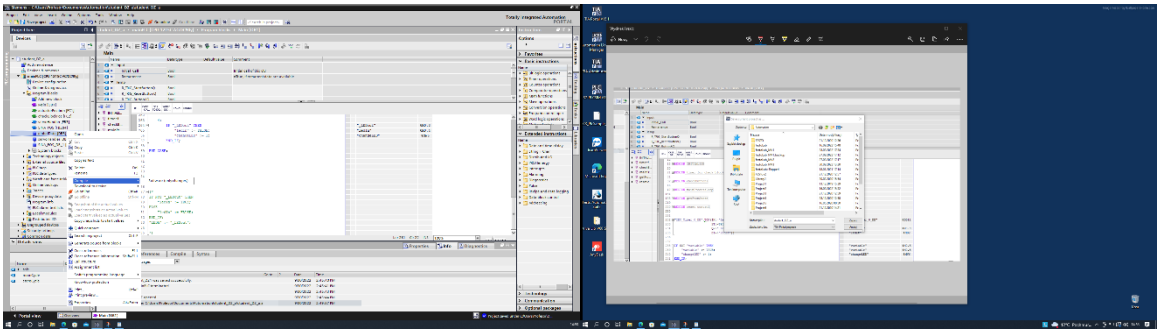
I oznacza wejście, Q wyjście.

Bbutton	Bool	%I1.3	False	True	True	True
StartButton	Bool	%I1.0	False	True	True	True
ResetButton	Bool	%I1.1	False	True	True	True
getSubpressure	Bool	%I0.6	False	True	True	True
setSubpressure	Bool	%Q0.2	False	True	True	True
setValveA	Bool	%Q0.0	False	True	True	True
setValveB	Bool	%Q0.1	False	True	True	True
setDownPos	Bool	%Q0.3	False	True	True	True
getDownPos	Bool	%I0.1	False	True	True	True
getTopPos	Bool	%I0.0	False	True	True	True
getSensor1	Bool	%I0.3	False	True	True	True
getSensor2	Bool	%I0.4	False	True	True	True

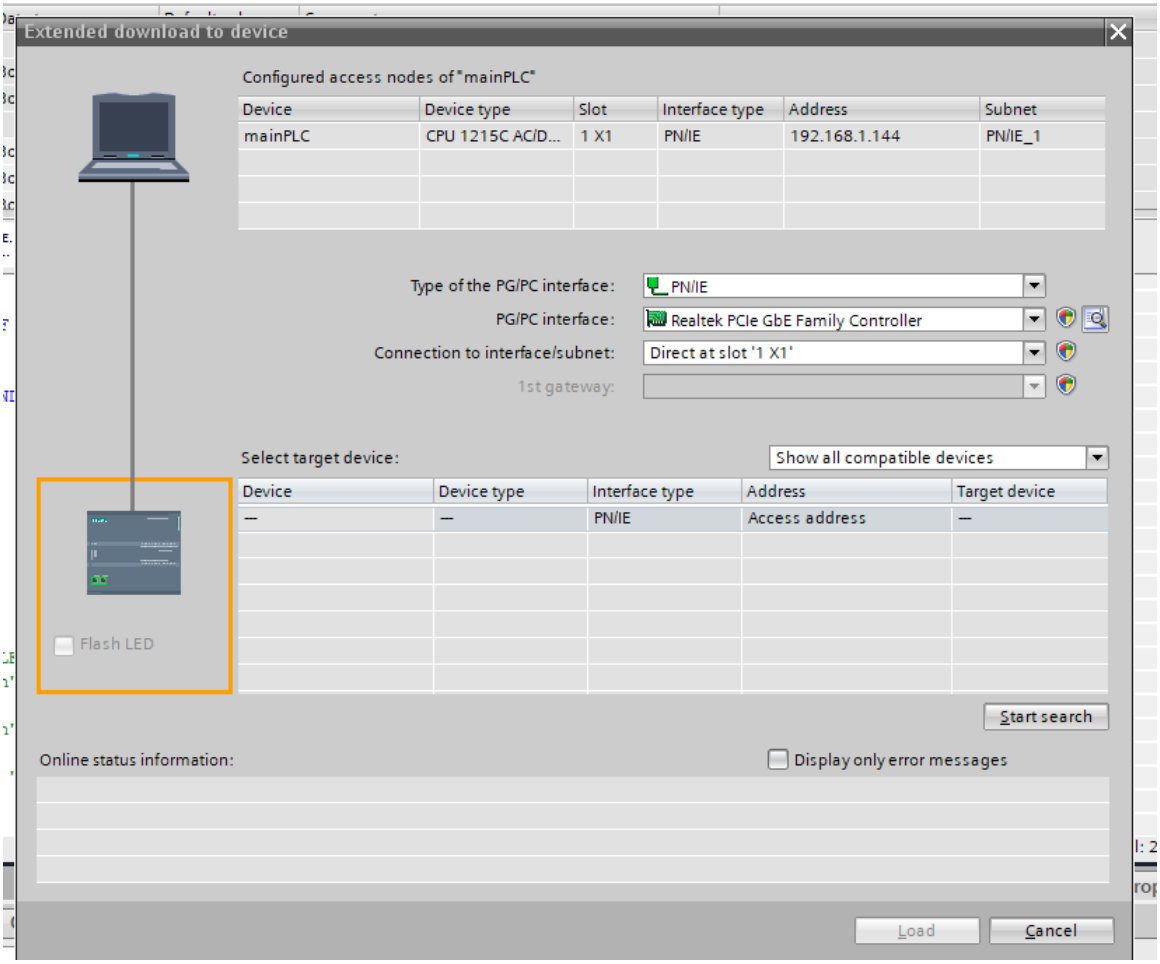
getSensor3	Bool	%I0.5	False	True	True	True	
LEDA	Bool	%Q0.4	False	True	True	True	
LEDB	Bool	%Q0.5	False	True	True	True	
LEDC	Bool	%Q0.6	False	True	True	True	
LEDD	Bool	%Q0.7	False	True	True	True	
LEDE	Bool	%Q1.0	False	True	True	True	
LEDF	Bool	%Q1.1	False	True	True	True	
Cbutton	Bool	%I1.4	False	True	True	True	
Dbutton	Bool	%I1.5	False	True	True	True	

Po edycji:

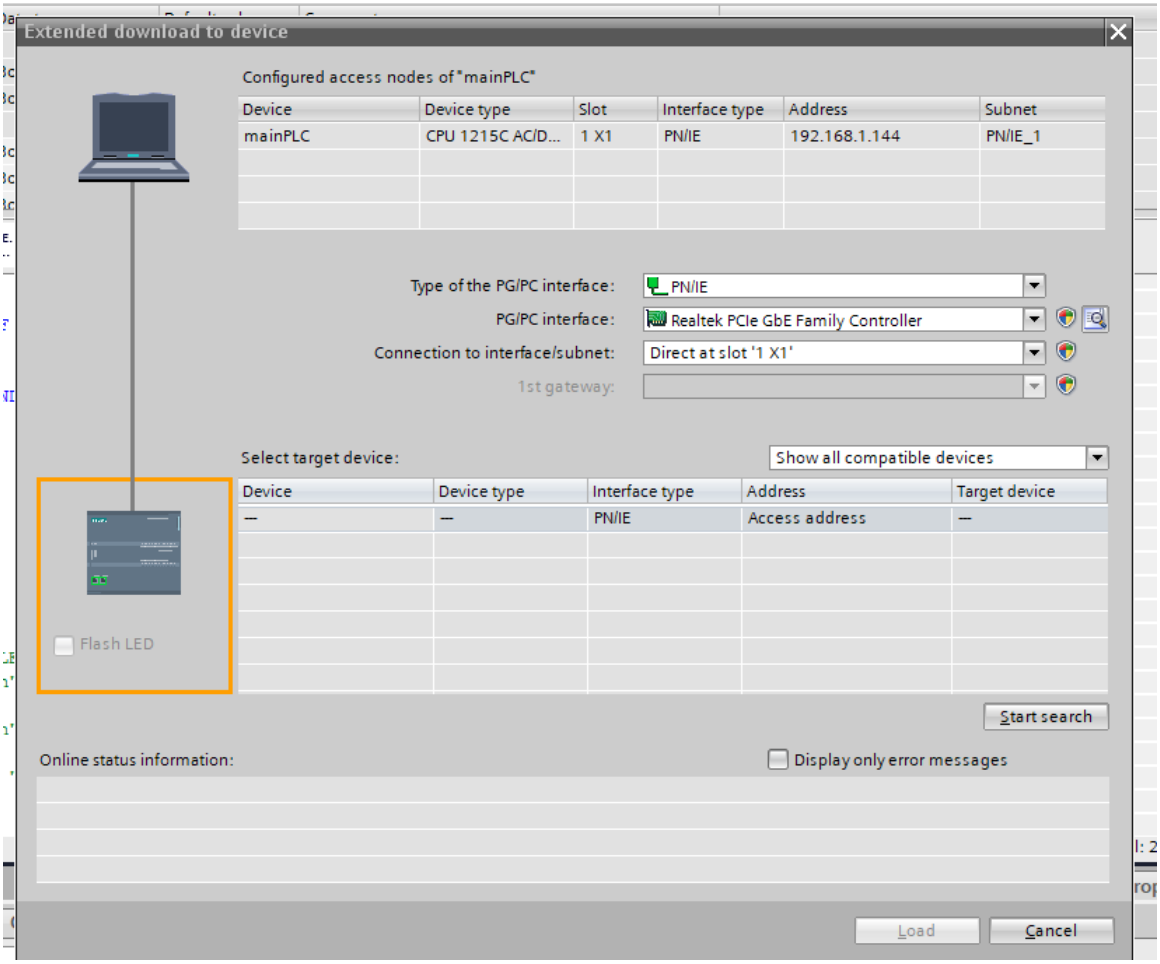
7. Prawym przyciskiem rozwinąć menu ładowania



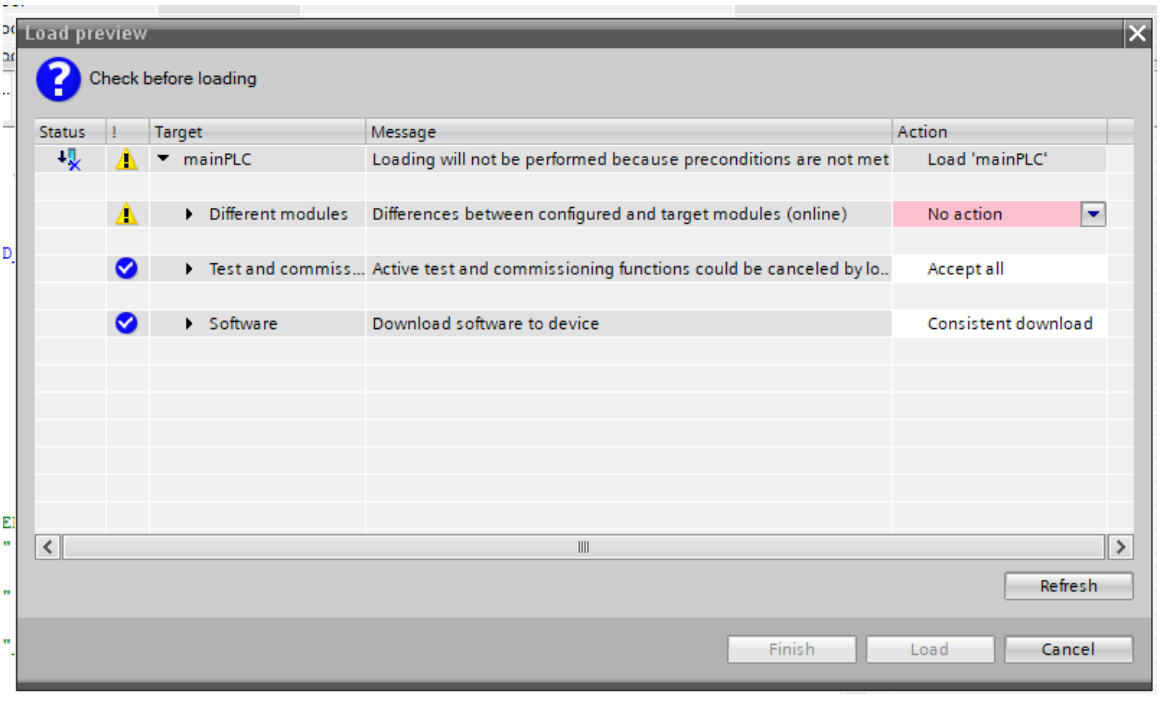
8. Wejść w procedurę ładowania do sterownika



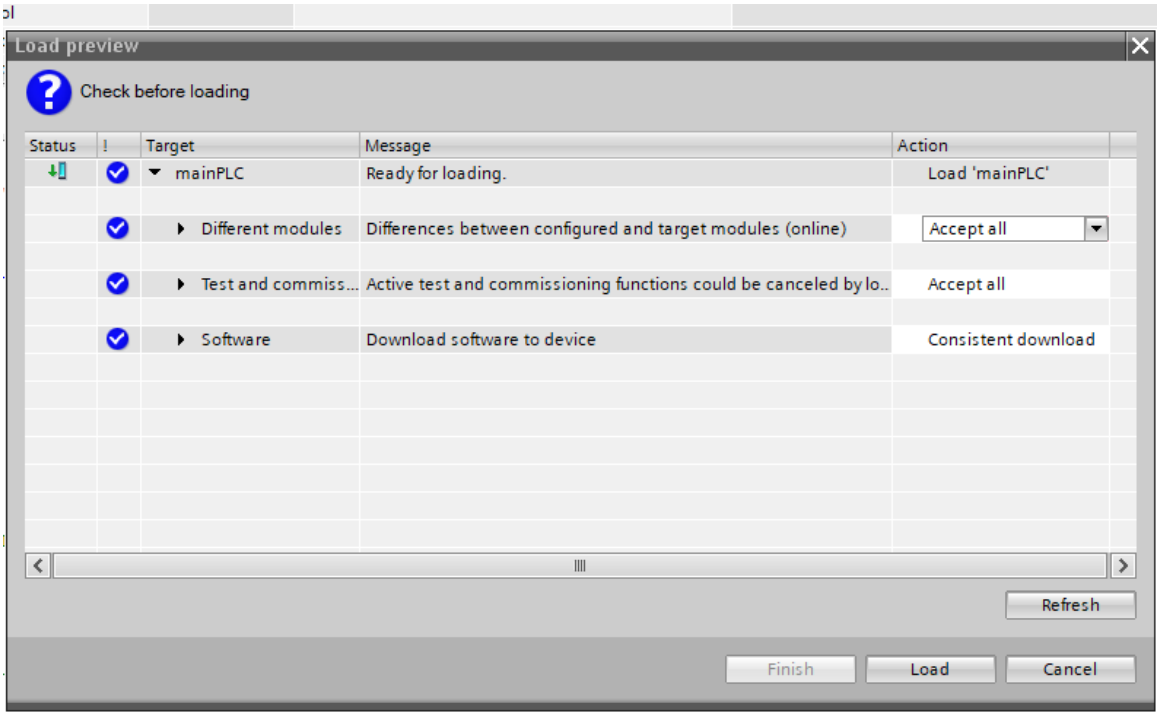
9. Start Search żeby odnaleźć urządzenie.



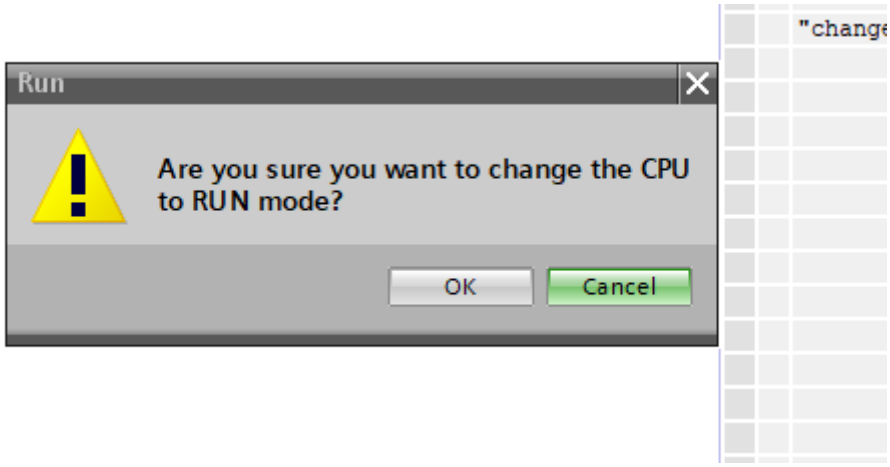
10. Zmienić akcję na Accept All



11. Załadować



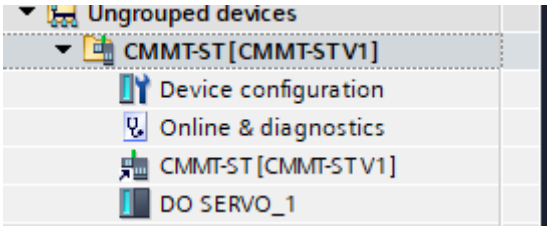
12. Uruchom z menu start_stop



13. po wykonaniu programu zatrzymaj.

Informacje dodatkowe:

Serwonapęd:

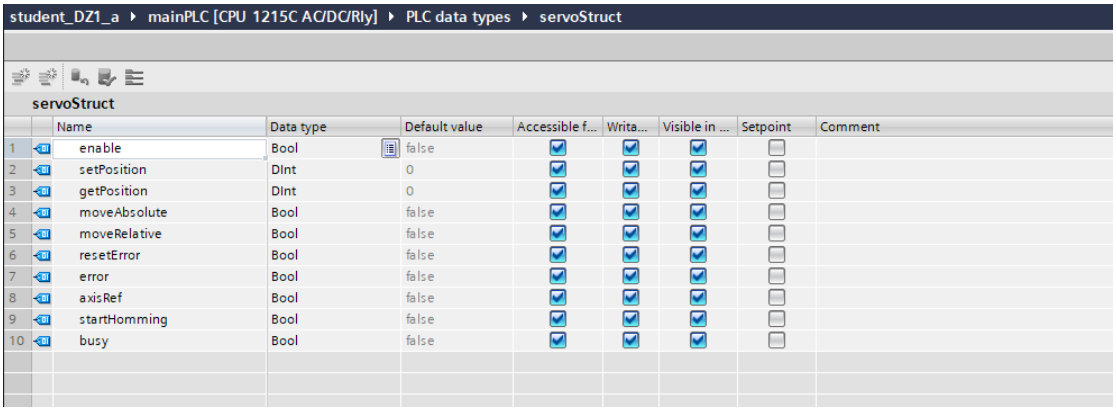


https://www.festo.com/cat/pl_pl/products_CMMT_ST

https://manualmachine.com/festo/cmmost/1571793-user-manual/

https://iautomatyka.pl/komunikacja-siemens-s7-1200-z-napedem-festo-cmmt-st-za-pomoca-profinet/

PLC data types \ ServoStruct



setPosition	DInt	0	True	True	True	False
getPosition	DInt	0	True	True	True	False
moveAbsolute	Bool	false	True	True	True	False
moveRelative	Bool	false	True	True	True	False

resetError	Bool	false	True	True	True	False
error	Bool	false	True	True	True	False
axisRef	Bool	false	True	True	True	False
startHomming	Bool	false	True	True	True	False
enable	Bool	false	True	True	True	False
busy	Bool	false	True	True	True	False

Przykład sterowania :

```
// bazowanie
IF #R_TRIG_StartButtonQ THEN
   "globalData".axis.startHomming := TRUE;
END_IF;
// ruch w kierunku dodatnim ( obecnie ok. 8 mm)
IF #R_TRIG_ButtonAQ THEN
   "globalData".axis.setPosition := 1000;
   "globalData".axis.moveRelative := TRUE;
ELSIF #R_TRIG_ButtonBQ THEN

// ruch w kierunku ujemnym
   "globalData".axis.setPosition := -1000;
   "globalData".axis.moveRelative := TRUE;
END_IF;
```

Sterowanie urządzeniami binarnymi:

(na przykładzie zapalenia/gaszenia diody przez naciśnięcia przycisku A – wyzwalanie zboczem narastającym – wymaga deklaracji R_TRIG_ButtonAQ)

```
IF #R_TRIG_ButtonAQ AND NOT "LEDA" THEN
            "LEDA":= TRUE;
        ELSIF #R_TRIG_ButtonAQ AND "LEDA" THEN
            "LEDA" := FALSE;
```

END_IF;

Można testować też urządzenia przez akcję wyzwalaną nie zboczem, a poziomem sygnału co jest prostsze – w tym celu wystarczy tylko sprawdzić wartość True/False danego przycisku.

Na podobnej zasadzie sterowane są siłowniki pneumatyczne i przyssawka:

setSubpressure	Bool	%Q0.2	False	True	True	True	// przyssawka
setValveA	Bool	%Q0.0	False	True	True	True	// oś Y - ustawienie True/False
setValveB	Bool	%Q0.1	False	True	True	True	// oś Y - lub False/True
setDownPos	Bool	%Q0.3	False	True	True	True	// oś Z

Zaleca się przetestowanie pneumatyki, przycisków oraz diod sygnalizacyjnych **przed** testowaniem serwonapędu osi X.

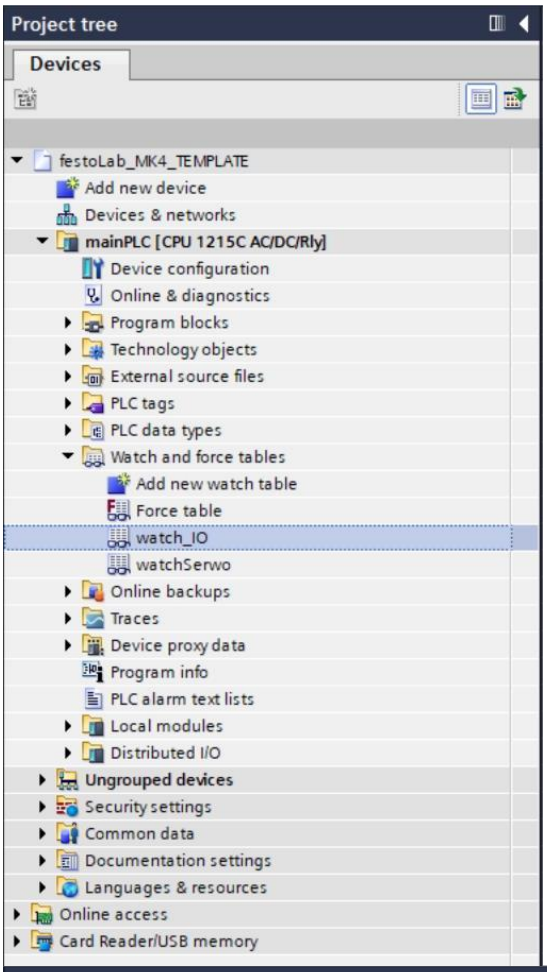
Część II

W drzewie projektu w folderze „Watch and force tables" znajduje się tablica „watch_IO". Tablice można wykorzystać do obserwacji sygnałów wejść/wyjść oraz możliwość wymuszania sygnałów wyjściowych. Istnieje możliwość dodawania własnych tablic w których można dodawać i obserwować interesujące nas zmienne. Nową tablicę dodajemy poprzez dwukrotne kliknięcie „Add new watch table".

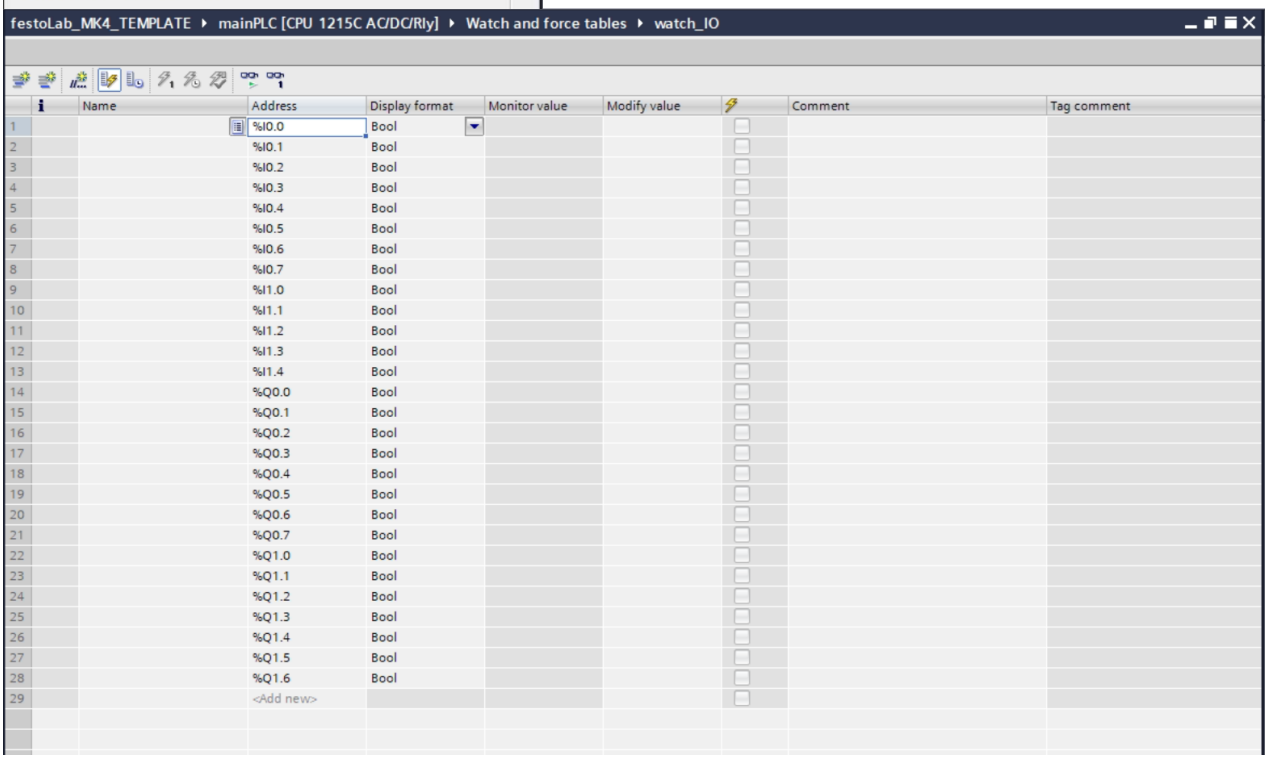
Po wejściu w „watch_IO" zobaczymy listę wejść/wyjść. Rejestry wejściowe definiowane są poprzez dodanie przed wartością rejestru znaku „%" oraz litery „I" np. %I0.1. W przypadku rejestrów sterujących wyjściami dodajemy literę „Q" np. %Q0.1.

Opcja sterowania i odczytywania wartości jest możliwa w momencie, gdy pracujemy w trybie online oraz gdy włączana jest opcja „Monitor all" znajdująca się w górnej części okna tablicy  .

Gdy znajdujemy się w trybie online oraz został uruchomiony podgląd zmiennych kolumna „Monitor value" zmieni się na kolor żółty oraz pojawią się wartości sygnałów. Możliwość zmiany sygnałów wyjściowych umożliwia kolumna „Modify value". Jest to kolumna, w której można przygotować wartość logiczną lub liczbową czy też tekstową w przypadku zmiennych. Klikając w komórkę w kolumnie „Modify value" komórka staje się aktywna i umożliwia edycję wartości. W przypadku wartości binarnych wartość TRUE reprezentowana jest przez cyfrę 1 natomiast FALSE przez cyfrę 0. Po wpisaniu nowej wartości w kolumnie obok pojawi się zaznaczony checkbox z wykrzyknikiem, który informuje o tym, że chcemy zmienić wartość rejestru. Jeśli nie chcemy zmieniać wartość danego rejestru należy odznaczyć checkbox. Następnie aby wartość została przepisana należy kliknąć ikonę znajdującą na górze okna tabeli  . Na przedstawionych zrzutach ekranu nie widzimy nazw rejestrów.

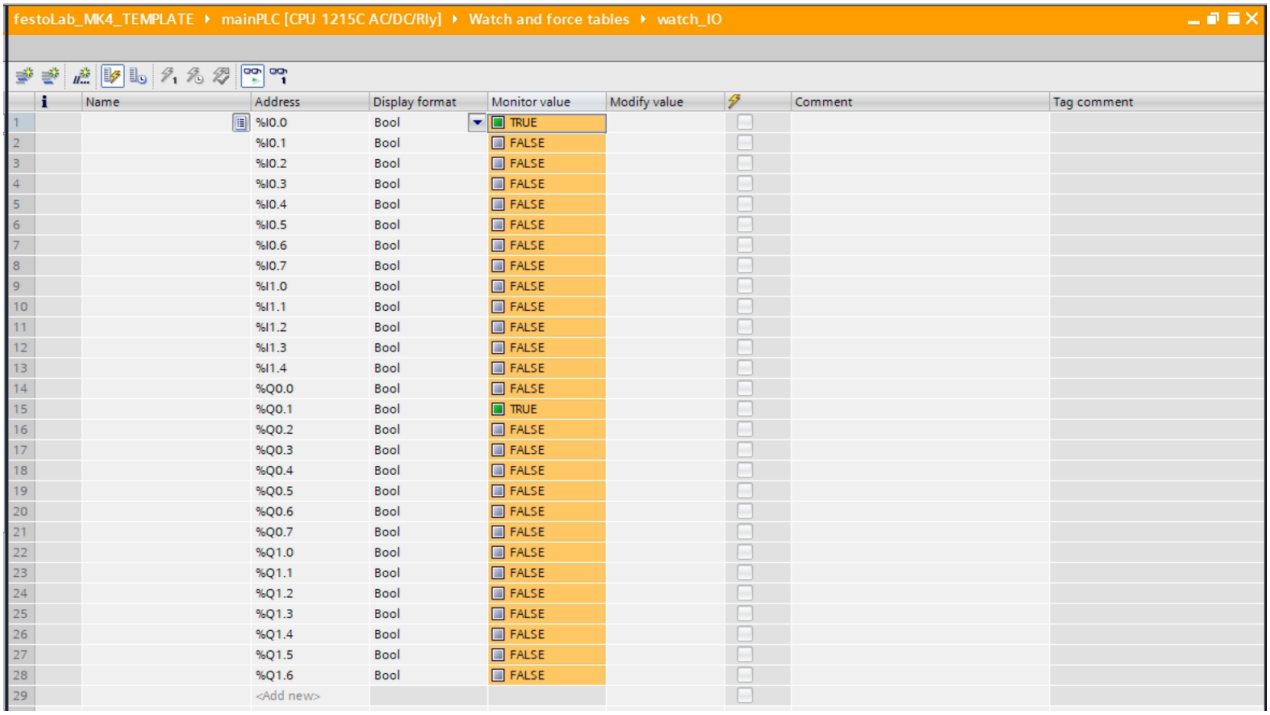


Nazwy te pojawią się w momencie gdy w folderze „PLC tags” znajdującym się w oknie projektu dodamy nową tablicę tagów, w której zdefiniujemy rejestry wejść/wyjść.



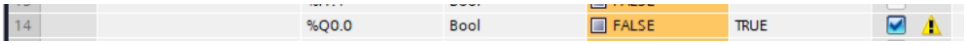
	Name	Address	Display format	Monitor value	Modify value	Comment	Tag comment
1		%I0.0	Bool				
2		%I0.1	Bool				
3		%I0.2	Bool				
4		%I0.3	Bool				
5		%I0.4	Bool				
6		%I0.5	Bool				
7		%I0.6	Bool				
8		%I0.7	Bool				
9		%I1.0	Bool				
10		%I1.1	Bool				
11		%I1.2	Bool				
12		%I1.3	Bool				
13		%I1.4	Bool				
14		%Q0.0	Bool				
15		%Q0.1	Bool				
16		%Q0.2	Bool				
17		%Q0.3	Bool				
18		%Q0.4	Bool				
19		%Q0.5	Bool				
20		%Q0.6	Bool				
21		%Q0.7	Bool				
22		%Q1.0	Bool				
23		%Q1.1	Bool				
24		%Q1.2	Bool				
25		%Q1.3	Bool				
26		%Q1.4	Bool				
27		%Q1.5	Bool				
28		%Q1.6	Bool				
29		<Add new>					

Okno ”watch table”, w tym przypadku tablica pozwalająca na podgląd rejestrów wejść/wyjść.



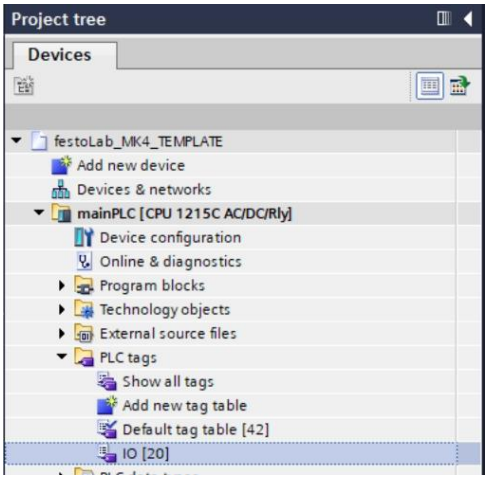
	Name	Address	Display format	Monitor value	Modify value	Comment	Tag comment
1		%I0.0	Bool	TRUE			
2		%I0.1	Bool	FALSE			
3		%I0.2	Bool	FALSE			
4		%I0.3	Bool	FALSE			
5		%I0.4	Bool	FALSE			
6		%I0.5	Bool	FALSE			
7		%I0.6	Bool	FALSE			
8		%I0.7	Bool	FALSE			
9		%I1.0	Bool	FALSE			
10		%I1.1	Bool	FALSE			
11		%I1.2	Bool	FALSE			
12		%I1.3	Bool	FALSE			
13		%I1.4	Bool	FALSE			
14		%Q0.0	Bool	FALSE			
15		%Q0.1	Bool	TRUE			
16		%Q0.2	Bool	FALSE			
17		%Q0.3	Bool	FALSE			
18		%Q0.4	Bool	FALSE			
19		%Q0.5	Bool	FALSE			
20		%Q0.6	Bool	FALSE			
21		%Q0.7	Bool	FALSE			
22		%Q1.0	Bool	FALSE			
23		%Q1.1	Bool	FALSE			
24		%Q1.2	Bool	FALSE			
25		%Q1.3	Bool	FALSE			
26		%Q1.4	Bool	FALSE			
27		%Q1.5	Bool	FALSE			
28		%Q1.6	Bool	FALSE			
29		<Add new>					

Okno „watch table” w trybie online.



14	%Q0.0	Bool	FALSE	TRUE	

Przygotowana wartość do zmiany w rejestrze.



Tablica tagów IO.

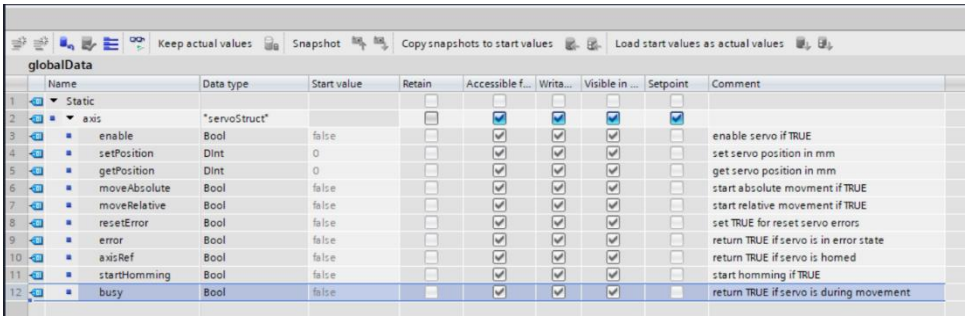
Wykorzystane FB oraz funkcje:

Blok funkcyjny odpowiedzialny za sterowanie serwomechanizmem:

```
1  //servo instance
2  "servoHandler_DB"(startHomming:="globalData".axis.startHomming,
3                    "moveAbsolute":="globalData".axis.moveAbsolute,
4                    moveRelative:="globalData".axis.moveRelative,
5                    resetError:="globalData".axis.resetError,
6                    setPosition:="globalData".axis.setPosition,
7                    actPosition=>"globalData".axis.getPosition,
8                    "axisRef"=>"globalData".axis.axisRef,
9                    "error"=>"globalData".axis.error,
10                   "axisBusy"=>"globalData".axis.busy
11 );
```

W głównym programie „Main” znajduje się instancja bloku funkcyjnego odpowiedzialnego za sterowanie serwomechanizmem. Wszystkie wejścia binarne, którymi sterujemy reagują na zbocze narastające, co więcej są one zdefiniowane jako zmienna VAR_IN_OUT co oznacza, że jest to referencja. Oznacza to, że do wejścia zawsze musimy przypisać zmienną oraz umożliwia to bloku funkcyjnemu samoczynną zmianę sygnału na stan niski po wykonaniu polecenia np. bazowania.

Struktura zmiennych przypisana do sygnału bloku zostały zadeklarowane w bloku danych DB o nazwie „globalData”. Należy pamiętać, że w głównym programie nie możemy deklarować zmiennych statycznych. Struktura zmiennych sterujących silnikiem została nazwana „axis” oraz posiada odpowiedni komentarz przy każdej zmiennej.



	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...	Setpoint	Comment
1	Static								
2	axis	"servoStruct"							
3	enable	Bool	false						enable servo if TRUE
4	setPosition	DInt	0						set servo position in mm
5	getPosition	DInt	0						get servo position in mm
6	moveAbsolute	Bool	false						start absolute movment if TRUE
7	moveRelative	Bool	false						start relative movement if TRUE
8	resetError	Bool	false						set TRUE for reset servo errors
9	error	Bool	false						return TRUE if servo is in error state
10	axisRef	Bool	false						return TRUE if servo is homed
11	startHomming	Bool	false						start homming if TRUE
12	busy	Bool	false						return TRUE if servo is during movement

Aby odwołać się do tych zmiennych w głównym programie należy odwołać się do odpowiedniego bloku danych, a następnie zmiennej w strukturze:

“blokDanych”.nazwaStruktury.nazwaZmiennej

np. Wpisanie wartości TRUE do zmiennej startHomming:

"globalData".axis.startHomming := TRUE;

Zmienne pozwalające na sterowanie serwonapędem:

- enable - pozwala na załączenie lub zwolnienie momentu silnika.
- setPosition – pozwala na wpisanie zadanej wartości pozycji napęu w mm dla ruchu relatywnego lub absolutnego.
- getPosition – zmienna zwraca aktualną absolutną wartość pozycji napędu.
- moveAbsolute – zmiana wartości z FALSE na TRUE pozwala na rozpoczęcie ruchu absolutnego.
- moveRelative – zmiana wartości z FALSE na TRUE pozwala na rozpoczęcie ruchu relatywnego.
- resetError – wpisanie wartości TRUE resetuje błędy napędu.
- error – zwraca informację czy napęd znajduje się w błędzie.
- axisRef – zwraca informację czy napęd został zbazowany.
- startHomming - zmiana wartości z FALSE na TRUE pozwala na rozpoczęcie bazowania napędu.
- busy – zwraca TRUE, gdy serwo znajduje się w ruchu. UWAGA: Po rozpoczęciu ruchu w sekwencji zaleca się najpierw sprawdzenie czy napęd rozpoczął ruch, a następnie dla wykonania reszty logiki cyklu sprawdzenie czy ruch został zakończony!!! Sprawdzenie w kolejnym kroku jedynie czy napęd zakończył ruch może powodować błędy w logice, ponieważ program sprawdzi warunek w chwili, gdy napęd jeszcze nie rozpoczął ruchu.

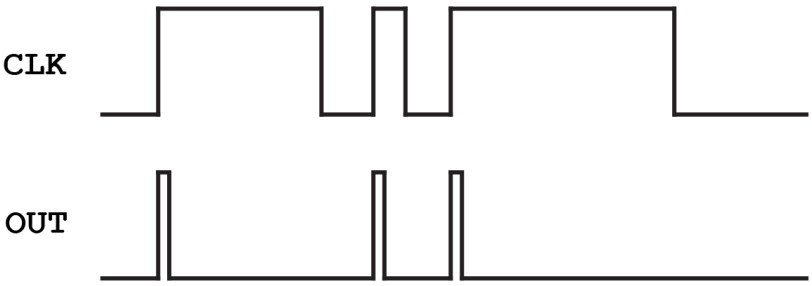
Przykład ruchu relatywnego oraz resetu błędu wywoływanych przez przyciski:

```
IF #R_TRIG_ButtonAQ THEN
   "globalData".axis.setPosition := 1000;
   "globalData".axis.moveRelative := TRUE;
ELSIF #R_TRIG_ButtonBQ THEN
   "globalData".axis.setPosition := -1000;
   "globalData".axis.moveRelative := TRUE;
ELSIF #R_TRIG_ResetButtonQ THEN
   "globalData".axis.resetError := TRUE;
END_IF;
```

Opis bloku wykrycia zbocza dla konkretnego sygnału:

Istnieją dwa bloki funkcyjne umożliwiających detekcję zbocza narastającego lub opadającego, odpowiednie R_TRIG, F_TRIG.

```
"R_TRIG_DB"(CLK:=_bool_in_,
        Q=>_bool_out_);
```



Zmienna CLK przyjmuje wartości zmiennej binarnej, którą chcemy obserwować np. Realizacja wykrycia wciśnięcia przycisku. Wyjście Q przepisuje na wyjście informację o tym, czy zostało wykryte zbocze. W przypadku wykrycia wyjście Q zwraca TRUE, należy pamiętać że wartość ta jest tymczasowa i w kolejnej pętli programu zwróci wartość FALSE.

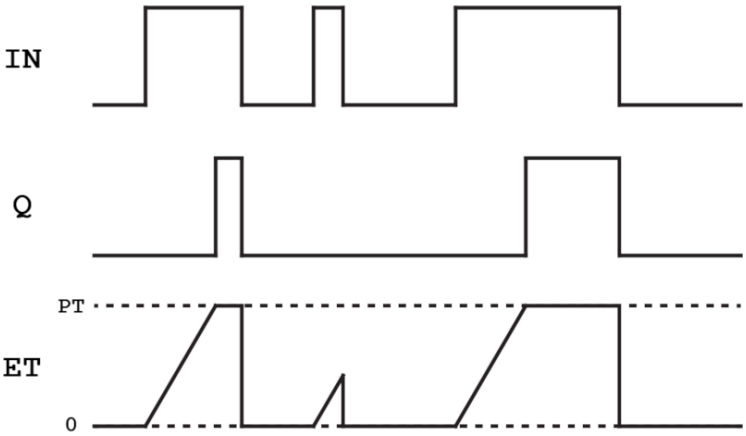
Opis bloku obsługującego licznik TON:

"IEC_Timer_0_DB".TON(IN:=_bool_in_,

PT:=_time_in_,

Q=>_bool_out_,

ET=>_time_out_);



Zmienna „IN” reaguje stan wysoki w momencie, gdy na tym wejściu pojawi się stan wysoki licznik zaczyna odliczać czas „PT”. Możliwe jest bezpośrednie wpisanie wartości lub przekazanie jej przez zmienną. Format czasu to **T#wartośćCzasuJednostkaCzasu** np. T#500ms. Wyjście „Q” przyjmie

wartość TRUE po odliczeniu zadeklarowanego czasu do momentu, gdy „IN” będzie w stanie wysokim. Wyjście „ET” zwraca aktualną wartość czasu, który upłynął od momentu odliczania.

Zalecane funkcje do napisania:

Sterowanie zaworem dwucewkowym.

Ze względu na to, że jedna z osi pneumatycznych manipulatora kartezjańskiego (oś Y) jest sterowana przez zawór dwucewkowy zaleca się wypełnienie przygotowanej funkcji „actuatorPosition”, która przyjmowałaby argument decydujący o tym w jakiej pozycji będzie przesterowany siłownik. Pozwoli to na jednokrotne napisanie sekwencji i następnie wykorzystanie jej w głównej pętli programu.

Sprawdzenie orientacji pobranego elementu.

Zaleca się wypełnienie przygotowanej funkcji „checkBlockPos”, która wykorzystując sygnały zebrane z trzech czujników indukcyjnych zwróci informację o orientacji pobranego elementu. Zaleca się, aby funkcja zwracała wartość liczbową np. BYTE. Należy pamiętać, że być może przed wykryciem elementu konieczne stanie się dodanie minimalnego opóźnienia, aby poprawnie odczytać wartości ze wszystkich czujników.

Tabela IO:

Adres	**Funkcjonalność**
%I0.0	siłownik w górnym położeniu
%I0.1	siłownik w dolnym położeniu
%I0.2	
%I0.3	czujnik indukcyjny nr. 1
%I0.4	czujnik indukcyjny nr. 2
%I0.5	czujnik indukcyjny nr. 3
%I0.6	czujnik podciśnienia
%I0.7	
%I1.0	przycisk START
%I1.1	przycisk RESET
%I1.2	przycisk A
%I1.3	przycisk B
%I1.4	przycisk C
%I1.5	przycisk D
%Q0.0	zawór dwucewkowy siłownika cewka A - oś Y
%Q0.1	zawór dwucewkowy siłownika cewka B - oś Y
%Q0.2	Eżektor dla przyssawki zamontowanej w osi Z

%Q0.3	zawór siłownika osi Z
%Q0.4	dioda A
%Q0.5	dioda B
%Q0.6	dioda C
%Q0.7	dioda D
%Q1.0	dioda E
%Q1.1	dioda F

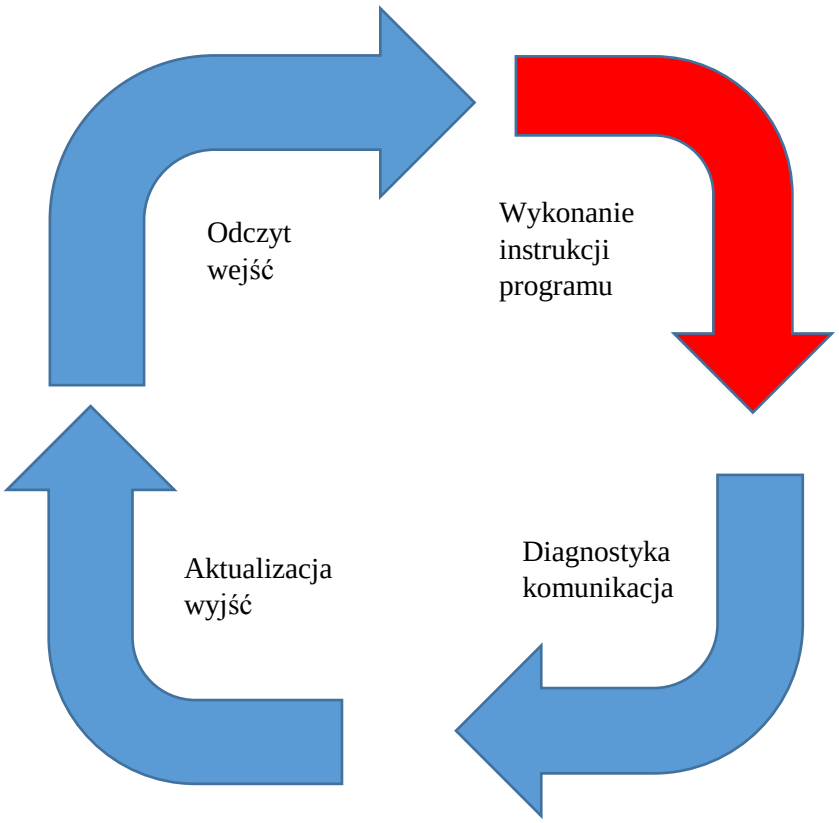
Dodatek: Metodyka programowania z uwzględnieniem podstawowego cyklu pracy sterownika.

Przy tej metodzie programowaniu sterownika, jeżeli nie korzystamy bezpośrednio z funkcji zegarowych do realizacji automatu sekwencyjnego, można wykorzystać **podstawowy cykl sterowanika** (Rys. 1).

Cykl ten wykonuje się w pętli nieskończonej każdorazowo po uruchomieniu programu. Typowy czas wykonania dla sterownika S7-1200 wynosi **około** 1 ms, ale może się zmieniać wraz z kodem programu. Należy mieć więc na uwadze, iż w ciągu tego krótkiego czasu wykonuje się **cały** napisany kod programu, w następnej **umownej** milisekundzie znowu cały i tak dalej w pętli nieskończonej … aż do zatrzymania programu. Oczywiście wykonanie kodu instrukcji w ciągu owej milisekundy nie oznacza, że będzie po tej instrukcji widoczny od razu końcowy **efekt** jej wykonania – ten może pojawić się znacznie później.

Z podanych uwarunkowań wynika, iż około 1000 cykli sterownika (tysiąc wykonań napisanego kodu programu), przy stosunkowo mało rozbudowanych programach będzie się wykonywać **około** 1 sekundy. Warto sprawdzić samodzielnie czy tak rzeczywiście jest.

Należy raz jeszcze tu podkreślić, że czasy te są **przybliżone,** mogą nieco się zmieniać i nie powinny być wykorzystywane do **dokładnego** odmierzania czasu – jeżeli jest taka potrzeba, należy wykorzystać **timery.**



Rys. D1 Podstawowy cykl sterownika

W kodzie programu można umieścić instrukcję pozwalającą zliczać wykonane od początku cykle.

licznik_cykli := licznik_cykli+1; // instrukcja ta powinna być wykonywana w każdym cyklu

Jeżeli zadeklarujemy zmienną licznik_cykli (np. jako PLC tag) wtedy możemy w okienku **Watch** podglądać **numer** aktualnie wykonywanego cyklu (najczęściej przy założeniu, że licznik startuje od zera). Mechanizm śledzenia numeru aktualnego cyklu (nazywanego w tym dokumencie również dalej krokiem) można wykorzystać do generowania większych **makro-cykli** w programie.

W tym mechanizmie tworzenia sekwencji operacji, wykonanie niektórych instrukcji powinno być blokowane na skutek tego, iż aktualny licznik cykli będzie poza zakresem danego makro cyklu. W ten sposób uzyskamy przypisanie instrukcji do danego makro-cyklu.

Przykładowo możemy cały program podzielić na 3 makro cykle (rys. 2):

1. Cykl inicjalizacji pracy urządzenia
2. Cykl oczekiwania na start programu
3. Cykl właściwego programu.

- W cyklu inicjalizacji powinny być wykonane operacje inicjalizujące pracę robota, najważniejsze jest tu przygotowania napędu osi X do pracy.
- W cyklu oczekiwania program oczekuje na uruchomienie właściwego procesu poprzez naciśnięcie np. przycisku START.
- W cyklu właściwego programu wykonane są kolejno odpowiednie instrukcje programu.

Podział na **makro cykle** odbywa się poprzez zaznaczenie w programie odpowiedniego **zakresu** makro cyklu.

Przykładowo makro cykl inicjalizacji może trwać, jeśli licznik cykli jest w zakresie 1-899, makro cykl oczekiwania, jeśli jest w zakresie 900-999, właściwy makro cykl w zakresie 1000 – 30 000. Wykonanie danej instrukcji należącej do danego makro cyklu jest zatem możliwe przy wykorzystaniu konstrukcji:

```
if licznik_cykli = numer_cyklu_dla_danej_instrukcji then instrukcja
End_if ;.
```

Przykładowo:

```
If  licznik_cykli = 300 then  "globalData".axis.startHomming := TRUE End_if ;
```
// oznacza, że bazowanie zostanie rozpoczęte w cyklu nr 300.

Należy tu zwrócić uwagę na to, aby daną instrukcję mającą efekt wykonawczy wykonać w **pojedynczym** cyklu (sprawdzanie tu np. warunku nierówności, mniejszości, większości zamiast **równości** powodowałoby, że dana instrukcja była aktywizowana wielokrotnie przy każdym nowym cyklu). Sprawdzanie natomiast stanu wejść powinno się zazwyczaj odbywać **w określonym zakresie** licznika_cykli – jest to więc podstawowa **różnica** między ustawianiem wyjścia a odczytywaniem wejścia.

Aby mieć pewność, że program znajduje się w danym makro cyklu należy wprowadzić do kodu programu odpowiednią reinicjalizację cyklu, jeśli licznik_cykli dojdzie do maksymalnej wartości dla danego makro_cyklu np.

```
If licznik_cykli=999 then licznik_cykli=900
End_if;
```

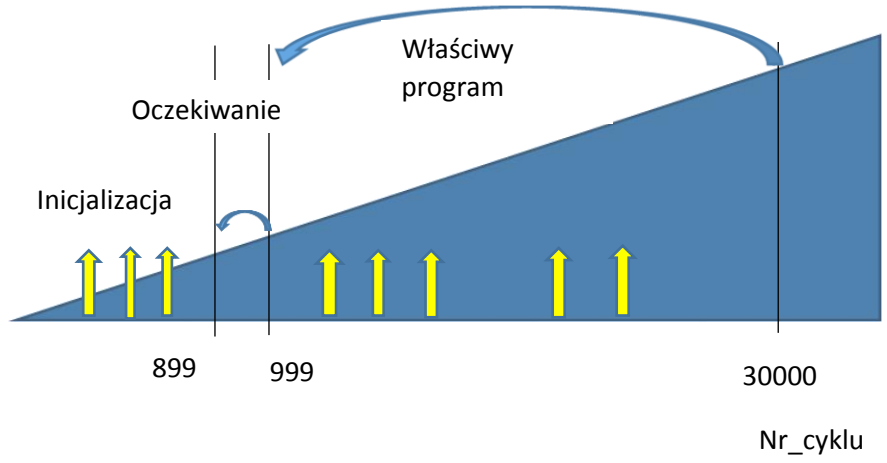
W ten sposób na danym etapie programu: w makro cyklu oczekiwania, będziemy mieli pewność, że licznik_cykli zawiera się w zakresie <900;999>.

Jeżeli takie ograniczenie nie zostanie wprowadzone, to po dojściu do liczby 999 nastąpi **bezwarunkowe** wyjście z danego makro_cyklu i przejście do następnego. Jeżeli ograniczenie jest wprowadzone program znajduje się w pętli nieskończonej danego makro_cyklu, chyba że nastąpi zdarzenie zmieniające wartość licznika_cykli na liczbę spoza zakresu makro_cyklu np. przez naciśnięcie wybranego przycisku.

Przykładowo:

If przycisk_start=TRUE then licznik_cykli = 1000 End_if; // w przypadku kiedy program znajdowałby się w zakresie <900;999> przestawienie licznika spowodowałoby wyjście poza ten makro_cykl i już brak możliwości powrotu do niego, chyba że nastąpiłoby później jakieś nowe zdarzenie ustawiające licznik_cykli na liczbę z zakresu <900;999>. Należy tu zauważyć, że trzymanie statyczne przycisku_startu, przy aktywnej powyższej instrukcji, de facto zatrzyma nam zmiany licznika_cykli (na skutek ciągłej reinicjalizacji), który zacznie znów się zwiększać dopiero, kiedy zwolnimy przycisk.

Przy podanych wyżej makro_cyklach w następujący sposób graficznie możemy przedstawić przykładowy harmonogram operacji:



Rys. D2 Przykładowy harmonogram operacji

Strzałki symbolizują tu operacje, które mogą być wykonane w danym kroku makro_cyklu:

np. wysłanie odpowiedniego sygnału na wyjście, pozycjonowanie w osi X, itp.

Jeżeli efekt działania instrukcji powinien wystąpić **przed** jakąś drugą instrukcją, do tej drugiej instrukcji należy przypisać **odpowiednio wyższy** numer kroku makro_cyklu, tak aby była pewność zaistnienia owego efektu (można np. zaczekać na rozpoczęcie ruchu w osi Z dopiero po zakończeniu ruchu w osi Y).

Należy zaznaczyć, iż **upływ określonego czasu** nie jest jedynym możliwym do wykorzystania **warunkiem przejścia** do następnej instrukcji, można wykorzystać również inne warunki zakończenia akcji (np. aktywizację określonego sensora w wyniku pracy urządzenia) – wtedy należy zadbać o odpowiednią strukturę logiczną programu. Standardowo są wykorzystywane do tego celu różnego rodzaju flagi (zmienne typu booolowskiego False/True) zezwalające lub blokujące wykonanie określonych instrukcji.