**Wydział Elektryczny**
**Zespół Automatyki (ZTMAiPC)**

LABORATORIUM KOMPUTEROWYCH UKŁADÓW STEROWANIA

Ćwiczenie 4

# Implementacja prostych algorytmów sterowania w środowisku Modicon Concept

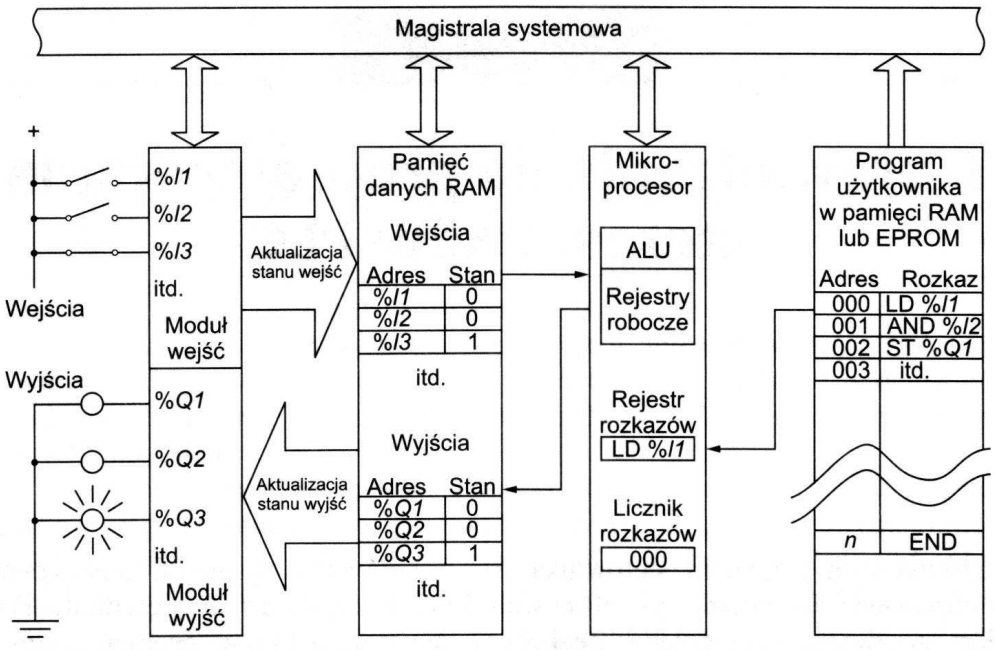
## 1. Wprowadzenie

Celem ćwiczenia jest zapoznanie się z procedurą tworzenia i uruchamiania programu sterowania PLC składającego się z sekcji napisanych w różnych językach programowania zgodnych z normą IEC 61131-3, w szczególności w języku schematu sekwencyjnych zmian stanu SFC. Do implementacji algorytmów wykorzystywane jest zintegrowane środowisko Modicon Concept.

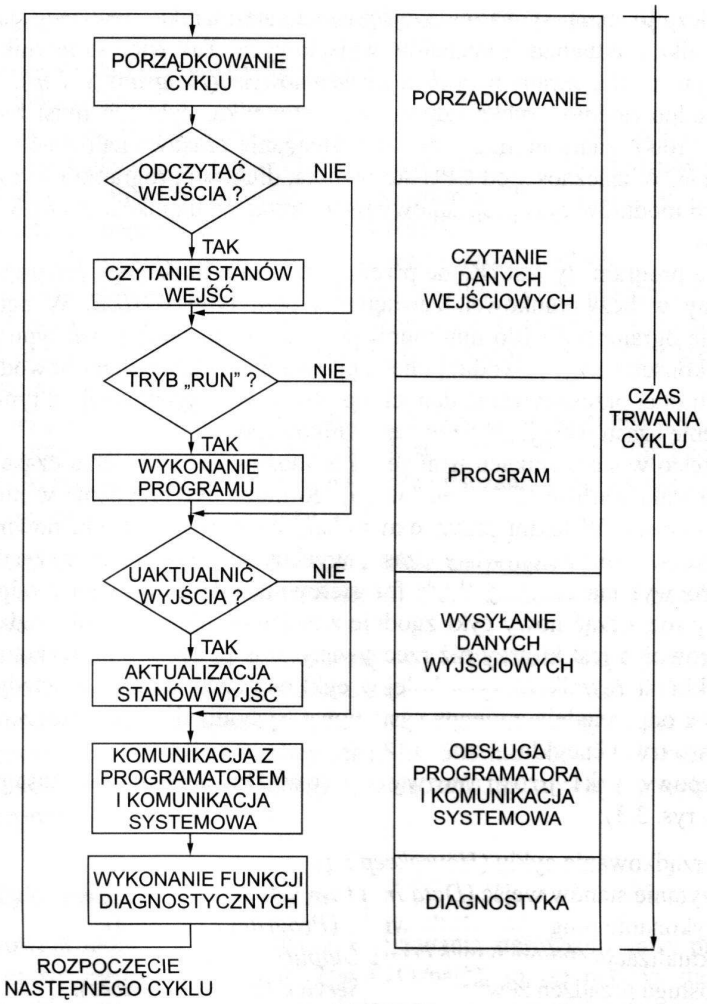
### 1.1. Norma IEC 61131 – *Sterowniki programowalne*

W związku z coraz powszechniejszym stosowaniem sterowników PLC, a zarazem różnorodnością rozwiązań programistycznych stosowanych przez różnych producentów, ze strony użytkowników zaczęło pojawiać się żądanie pewnej standaryzacji systemów PLC, w szczególności jeśli chodzi o metody programowania. Odpowiedzią na te postulaty było opracowanie przez *International Electrotechnical Commission* i wydanie w 1993r. normy *Programmable Controllers*, od 1998r. oznaczonej jako IEC 61131.

Część 1 normy (*General Information*) określa ogólne właściwości funkcjonalne sterowników PLC, m.in. cykliczne przetwarzanie programu użytkownika z wykorzystaniem przechowywanego w pamięci obrazu stanu wejść i wyjść sterownika, przydział czasu racy na komunikację z programatorem i/lub urządzeniami interfejsu operatora MMI (*Man Machine Interface*) (rys. 1.1-2).




Rys. 1.1. Schemat ideowy sterownika PLC [1]



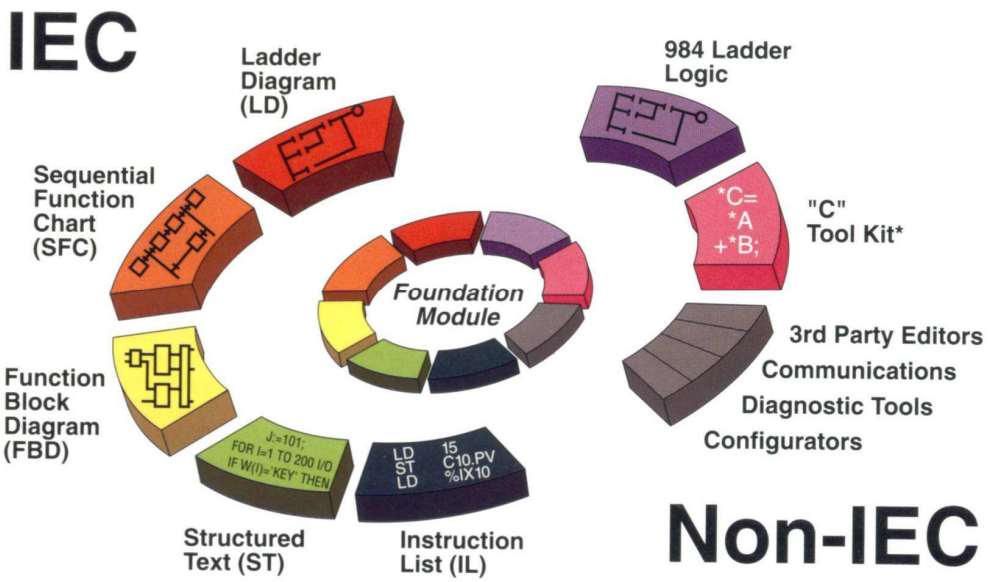

Rys. 1.2. Cykl pracy sterownika PLC [1]

Współcześnie sterowniki PLC pracują coraz częściej w warstwie sterowania bezpośredniego (*Direct Control*, prostsze jednostki) i/lub w warstwie sterowania nadrzędnego (*Supervisory Control*) rozproszonych (sieciowych), hierarchicznych systemów sterowania. Integracja węzłów takich systemów jest realizowana za pomocą stacji roboczych z oprogramowaniem SCADA (*Supervisory Control and Data Acquisition*) zbierającym i wizualizującym dane zbierane ze sterowników.

Część 3 normy IEC 61131 (*Programing Languages*) dotyczy zasad programowania sterowników PLC i jest najważniejsza z punktu widzenia użytkownika. Ujednolicono w niej zasady programowania na tyle, aby użytkownik zaznajomiony z programowaniem jednej rodziny sterowników PLC był w stanie programować również różne inne systemy pochodzące od różnych producentów. Norma IEC 61131-3 stała się wzorcem dla nowych rozwiązań. W 2003r. opublikowano jej drugie, oprawione wydanie.

Norma IEC 61131-3 określa dwie grupy języków programowania: graficzne i tekstowe (rys. 1.3). Do języków tekstowych należą:

- Język IL (*Instruction List* – lista rozkazów) – język niskiego poziomu, odpowiednik asemblera, którego zbiór instrukcji obejmuje operacje logiczne, arytmetyczne, relacyjne, jak również funkcje przerzutników, timerów, liczników. Umożliwia bezpośredni dostęp do rejestrów, opracje na bitach, przesłania itp.
- Język ST (*Structured Text* – tekst strukturalny) – odpowiednik języka algorytmicznego wysokiego poziomu, takiego jak Basic, Pascal czy C. Zawiera struktury programowe i polecenia podobne do występujących w tych językach.




Rys. 1.3. Języki programowania sterowników PLC określone przez normę IEC 61131-3 i inne

Do języków graficznych należą:

- Język LD (*Ladder Diagram* – schemat drabinkowy) – wywodzący się ze schematów stykowych obwodów przekaźnikowych. Oprócz styków, cewek i połączeń między nimi douszcza się też użycie funkcji (arytmetycznych, logicznych, relacyjnych) oraz bloków funkcjonalnych (przerzutniki, timery, liczniki).
- Język FBD (*Function Block Diagram* – schemat bloków funkcyjnych) – odpowiednik schematu przepływu sygnałów w układach logicznych złożonych z bramek i bloków funkcjonalnych takich jak w języku LD
- Język SFC (*Sequential Function Chart* - schemat sekwencyjnych zmian stanu) – metoda reprezentacji wewnętrznej struktury programu i opisu algorytmu sterowania w sposób sekwencyjny, za pomocą grafów zawierających kroki (*steps*) i warunki przechodzenia od kroku do kroku – tranzycje (*transitions*). W krokach określa się akcje sterowania realizowane po uaktywnieniu danego kroku. Definicje akcji i tranzycji programuje się w językach wymienionych wcześniej. W odróżnieniu od innych języków, w sieci SFC nie występuje wykonywanie kolejnych instrukcji, ale cykliczne ustalanie wartości statusu kroków i tranzycji.

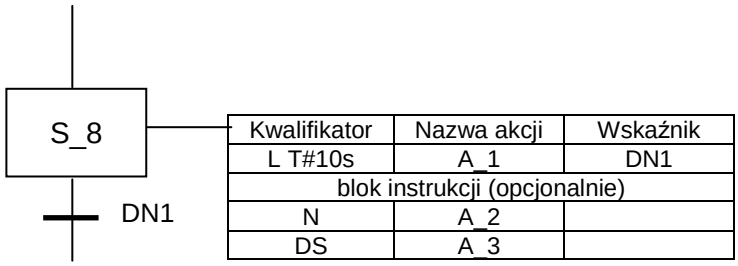
## 1.2. Język SFC – *Sequential Function Chart*

Metoda SFC nadaje się szczególnie do programowania algorytmów sterowania procesów, które dają się podzielić pewne sekwencje działań realizowanych krok po kroku, tzn. da się dobrze zdefiniować etapy (stany) i określić warunki przechodzenia pomiędzy nimi (często podawany przykład: proces prania w pralce automatycznej). W poszczególnych fazach pewne akcje mogą być wykonywane współbieżnie. SFC wywodzi się z metody *Grafcet* modelowania złożonych systemów sterowania sekwencyjnego opracowanej w 1977r. we francuskiej firmie *Telemecanique* (ulepszenie tej metody pod nazwą *Grafpol* opracowano na Politechnice Wrocławskiej, ale nie zdobyła ona popularności). Zarówno Grafcet jak i SFC stanowią modyfikacje sieci Petriego typu P/T (Pozycja/Tranzycja).

W Tabeli 1.1 przedstawiono sekwencje kroków i tranzycji stosowane w schematach SFC. Tranzycja jest dozwolona (*enabled*) tylko wówczas, gdy wszystkie bezpośrednio poprzedzające ja kroki są aktywne. Jeżeli tranzycja jest dozwolona i jednocześnie spełniony jest związany z nią warunek, to następuje tzw. kasowanie tranzycji i deaktywacja (reset) wszystkich poprzedzających ją bezpośrednio kroków oraz aktywacja wszystkich kroków następujących bezpośrednio po tranzycji. Po aktywacji krok wykonywany jest co najmniej raz, łącznie ze wszystkimi skojarzonymi z nim akcjami. Taki sposób działania, w którym atrybut stanu aktywności jest przekazywany do kolejnych kroków, jest w pewnym sensie analogiczny do przekazywania żetonu pomiędzy kolejnymi stacjami w sieci komputerowej (*token passing*).

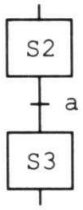
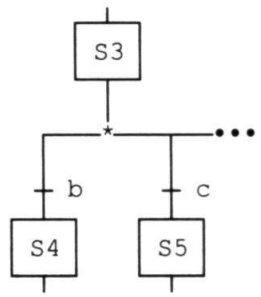
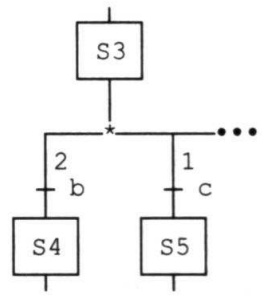
Prosta kombinacja kroków i tranzycji jest nazywana sekwencją.

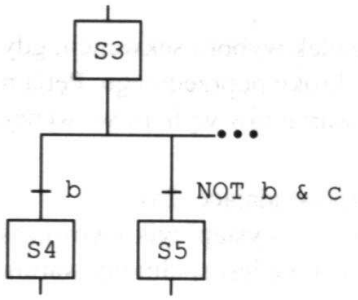
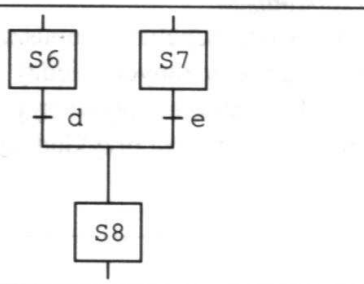
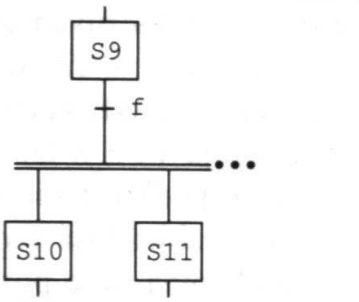
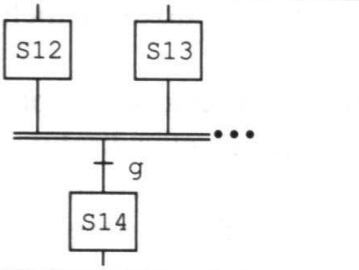
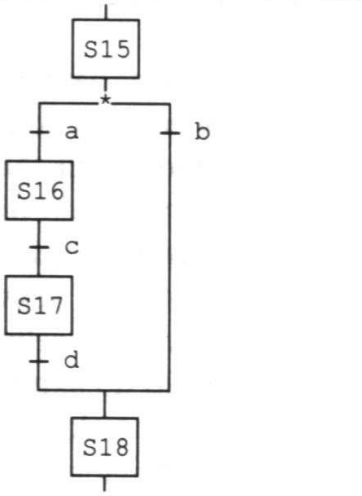
Kroki i tranzycje muszą zawsze występować naprzemiennie: dwa kroki muszą być przedzielone tranzycją, a dwie tranzycje muszą być przedzielone krokiem. Każdy krok jest skojarzony z akcjami podejmowanymi po jego uaktywnieniu (rys. 1.4). Kwalifikator akcji stanowi określenie warunków zezwolenia na wykonywanie skojarzonej akcji. Logiczne zmienne wskaźnikowe służą głównie do przekazywania na zewnątrz informacji o stanie wykonania danej akcji.

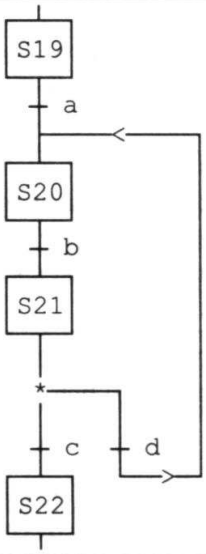



Rys. 1.4. Opis skojarzenie kroku z akcjami

Tabela. 1.1. Sekwencje kroków i tranzycji (przejść) stosowane w schematach SFC [1]

| Lp. | Przykład | Reguła |
|---|---|---|
| 1 |   | *Sekwencja pojedyncza*<br>Kroki i przejścia są powtarzane kolejno po sobie.<br>Przykład:<br>Przejście z kroku *S2* do *S3* jest możliwe tylko wówczas, gdy krok *S2* jest aktywny i jest spełniony warunek *a* |
| 2a |   | *Wybór sekwencji – rozbieżność*<br>Wybór między różnymi sekwencjami jest reprezentowany przez tyle przejść umieszczonych pod linią poziomą, ile jest możliwych sekwencji. Gwiazdka oznacza pierwszeństwo wyboru od lewej strony do prawej.<br>Przykład:<br>Przejście z kroku *S3* do *S4* wystąpi tylko wówczas, gdy *S3* jest aktywny i jest spełniony warunek *b*, a przejście z *S3* do *S5* wtedy, gdy *S3* jest aktywny i jest spełniony warunek *c*, a nie jest spełniony warunek *b* |
| 2b |   | *Wybór sekwencji – rozbieżność*<br>Gwiazdka i numerowane gałęzie wskazują na zadeklarowane pierwszeństwo sprawdzania przejścia. Gałęzie o niższym numerze mają wyższy priorytet.<br>Przykład:<br>Przejście z kroku *S3* do *S5* wystąpi tylko wówczas, gdy *S3* jest aktywny i jest spełniony warunek *c*, a przejście z *S3* do *S4* wtedy, gdy *S3* jest aktywny i jest spełniony warunek *b*, a nie jest spełniony warunek *c* |

| Lp. | Przykład | Reguła |
|---|---|---|
| 2c |  | *Wybór sekwencji – rozbieżność*<br>Połączenie gałęzi oznacza, że kolejne warunki przejścia muszą się wzajemnie wykluczać (gdyż nie ma tu określonego priorytetu gałęzi).<br>Przykład:<br>Przejście z kroku *S3* do *S4* wystąpi tylko wówczas, gdy *S3* jest aktywny i jest spełniony warunek *b*, a przejście z *S3* do *S5* tylko wówczas, gdy *S5* jest aktywny i nie jest spełniony warunek *b*, a jest spełniony warunek *c* |
| 3 |  | *Wybór sekwencji – zbieżność*<br>Wszystkie sekwencje kończą się symbolami przejścia umieszczonymi nad linią poziomą.<br>Przykład:<br>Przejście do kroku *S8* jest możliwe wówczas, gdy krok *S6* jest aktywny i jest spełniony warunek *d*, albo gdy *S7* jest aktywny i jest spełniony warunek *e* |
| 4 |  | *Sekwencje równoczesne – rozbieżność*<br>Jeden wspólny symbol przejścia umieszczony bezpośrednio nad podwójną poziomą linią synchronizacji.<br>Przykład:<br>Przejście z kroku *S9* do kroków *S10*, *S11*, … następuje tylko wówczas, gdy *S9* jest aktywny i jest spełniony wspólny warunek *f*. Po równoczesnej aktywacji *S10*, *S11* itd., realizacja każdej sekwencji jest niezależna |
| 5 |  | *Sekwencje równoczesne – zbieżność*<br>Jeden wspólny symbol przejścia umieszczony bezpośrednio pod podwójną poziomą linią synchronizacji.<br>Przykład:<br>Przejście z kroków *S12*, *S13*, … do kroku *S14* następuje tylko wówczas, gdy wszystkie kroki dołączone do linii podwójnej są aktywne i jest spełniony wspólny warunek *g* |
| 6 |  | *Przeskok sekwencji*<br>Jest to szczególny przypadek wyboru sekwencji, gdy któraś z gałęzi nie zawiera żadnego kroku. Przeskok może wystąpić dla każdego z wariantów wyboru sekwencji (pozycje 2a, 2b i 2c).<br>Przykład (dla opcji odpowiadającej 2a):<br>Przejście z kroku *S15* do *S18* z pominięciem kroków *S16* oraz *S17* wystąpi tylko wówczas, gdy *S15* jest aktywny i nie jest spełniony warunek *a*, natomiast jest spełniony warunek *b* |

| Lp. | Przykład | Reguła |
|---|---|---|
| 7 |  | *Pętla sekwencji*<br>Jest to szczególny przypadek wyboru sekwencji, gdy któraś z gałęzi wraca do kroku poprzedniego. Pętla może wystąpić dla każdego z wariantów wyboru sekwencji (pozycje 2a, 2b i 2c).<br>Przykład (dla opcji odpowiadającej 2a):<br>Przejście z kroku *S21* do *S20* wystąpi tylko wówczas, gdy jest spełniony warunek *d*, a nie jest spełniony warunek *c*<br>*Oznaczanie kierunku strzałkami*<br>W celu zwiększenia przejrzystości schematu dopuszcza się użycie znaków „<" oraz „>" umieszczonych między znakami „-" dla określenia kierunku przepływu sygnału sterującego, jak pokazano w załączonym przykładzie |

## 2. Środowisko Concept programowania sterowników PLC Modicon

Pakiet programowy Concept firmy Schneider Automation GmbH jest jednym z typowych narzędzi do programowania i uruchamiania programów w sterownikach PLC. Pakiet ten umożliwia programowanie popularnych sterowników Schneider–Modicon z rodzin: Quantum, TSX Compact, Momentum i Atrium. Podstawowe zalety tego narzędzia to:

- zasady programowania sterowników są zgodne z zaleceniami normy IEC 61131-3,
- pakiet jest łatwy w użytkowaniu – wykorzystano w nim wiele udogodnień dostępnych w aplikacjach pracujących w środowisku Windows. Wyposażony jest także w bardzo obszerny system pomocy (*Help*),
- korzystając z pakietu Concept można testować napisane programy bez konieczności posiadania fizycznego sterownika – pakiet został wyposażony w znakomity symulator sterowników wymienionych rodzin, który jest osobną aplikacją systemu Windows (sterownik wirtualny, zawiera zadajnik stanów logicznych i wartości rejestrów wejściowych - „sygnałów analogowych"),
- wersja demonstracyjna (*Trial*) jest w dużej mierze funkcjonalna: umożliwia tworzenie i zapis projektu, chociaż zawiera tylko podstawowe biblioteki funkcji (IEC, EXTENDED i SYSTEM), obsługuje do 100 zmiennych i nie pozwala na komunikację ze sterownikiem fizycznym. Tym niemniej jest bardzo przydatnym narzędziem treningowym.
- możliwość modyfikacji programu *on-line*.

Pakiet *Concept* tworzy we współpracy z systemem *Windows 95, 98, 2000, NT, XP* środowisko programowe, które udostępnia sześć języków programowania przeznaczonych do tworzenia programów użytkownika dla sterowników programowalnych. Językami tymi są zdefiniowane w normie IEC 61131-3: *FBD*, *LD*, *IL*, *ST*, *SFC* oraz dodatkowo, dla zachowania ciągłości z oprogramowaniem *Modsoft*, język *984 LL*, odpowiadający standardowi języka programowania *Modicon 984 Ladder Logic*.
Dla każdego z wymienionych języków programowania istnieje odpowiedni edytor języka.
Ponadto, niezależnie od wybranego języka, Concept zawiera:

- edytor typów danych (*Data type editor*),
- edytor zmiennych (*Variables editor*),
- edytor informacji o danych (*Reference data editor*).

W zależności od potrzeb użytkownik ma do dyspozycji wiele bibliotek bloków funkcji FFB (*Function/Function Block*):

- IEC – standardowe funkcje zdefiniowane w normie IEC 61131-3,
- EXTENDED – dodatkowe funkcje uzupełniające inne biblioteki (np. interpolacja, strefa

martwa, wartość średnia),
- SYSTEM – funkcje użytkowe systemu, takie jak określenie czasu cyklu programowego, podanie statusu systemu itp.
- CONT_CTL (*Continuous Control*) – bloki regulacji procesów ciągłych (m.in. regulatory PID, elementy różniczkujące, całkujące),
- DIAGNOSIS – funkcje diagnostyki błędów w programach sterowania,
- COMM – funkcje komunikacji sieciowej PLC z innymi węzłami sieci Modbus, Modbus Plus, Ethernet),
- ANA_IO – funkcje przetwarzania sygnałów wejść/wyjść analogowych,
- FUZZY – bilioteka funkcji sterowania rozmytego,
- EXPERTS – funkcje obsługi modułów ekspertowych,
- LIB984 – emulacja funkcji środowiska Modsoft

Oprócz głównej aplikacji, pakiet zawiera kilka osobnych narzędzi działających pod Windows:
- *Concept DFB* – program do tworzenia funkcji pochodnych (*Derived Function Blocks*) na podstawie istniejących i definiowanie makr programowych,
- *Concept EFB* – program do tworzenia własnych bibliotek funkcji elementarnych z wykorzystaniem języka C,
- *PLCSIM32* – programowy symulator (*IEC Simulator*) sterowników Modicon na PC z zadajnikiem stanów, dostosowuje się do konfiguracji PLC zdefiniowanej przez użytkownika,
- *Concept Converter* – narzędzie do konwersji FFB pomiędzy różnymi językami programowania.

Do realizacji ćwiczenia będzie wykorzystywany pakiet Concept v.2.6 Trial.

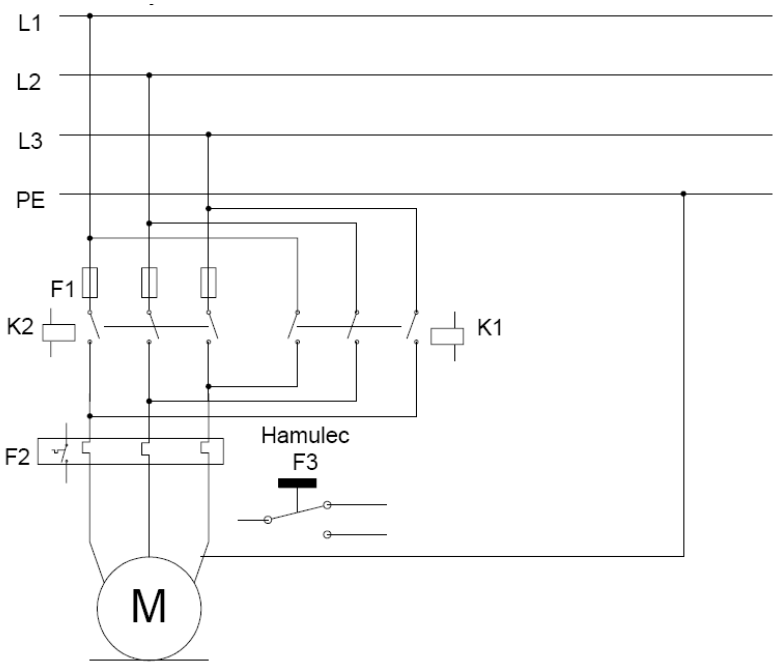
# 3. Program realizacji ćwiczenia

## 3.1. Sterowanie silnikiem asynchronicznym. Sekcje programu w różnych językach

Obwód sterowania trójfazowym silnikiem asynchronicznym zasilanym z sieci trójfazowej jest pokazany na rys. 3.1.

Silnik jest połączony z siecią przez styki przekaźników K1 albo K2, w zależności od wybranego przez przyciski sterownicze kierunku obrotu: S1 (w prawo) lub S2 (w lewo), zaś jego wyłączenie następuje za pomocą przycisku wyłączającego S0 (stop). Załóżmy, że jest to silnik jednobiegowy z jednopoziomową ochroną termiczną. Silnik wyposażony jest w zabezpieczenie termiczne uruchamiane pod wpływem nagrzania paska bimetalowego działającego na styk wyłączający F2 umieszczony jest w obwodzie sterowania. Dodatkowo możliwe jest wykorzystanie do sterowania styku przekaźnika F3 hamulca (wyłącznika) nadprędkościowego, sterowanego cewką zasilaną po przekroczeniu przez silnik określonego ograniczenia prędkości obrotowej.

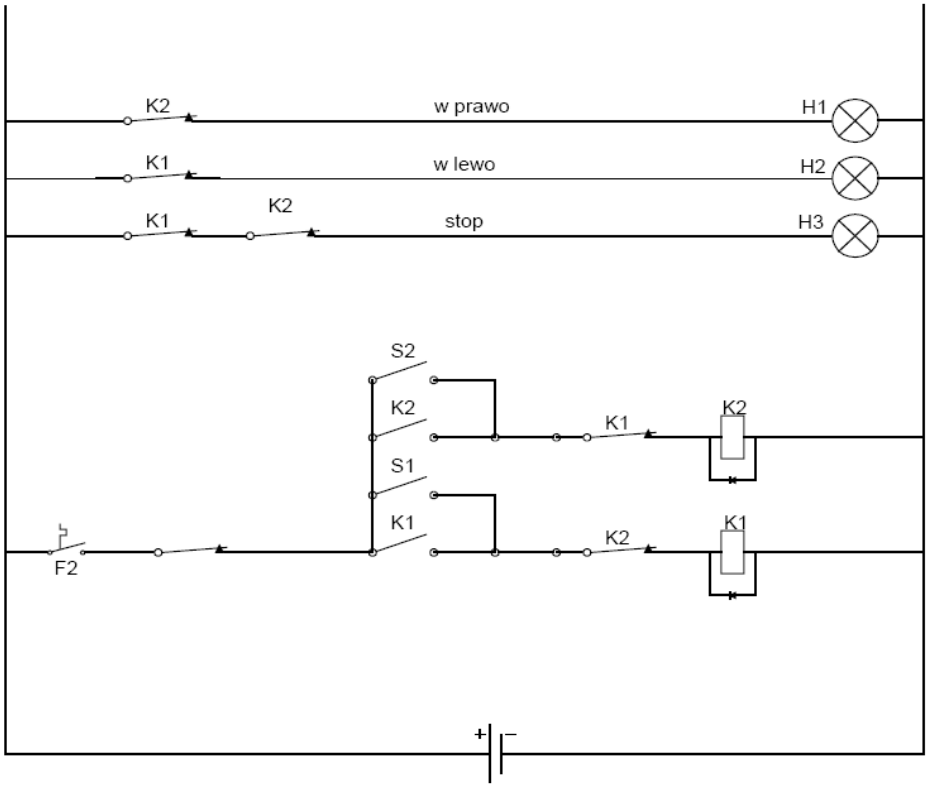
Rys. 3.2 przedstawia przykładowy prosty schemat stykowy układu sterowania takim silnikiem dla zmiennych zdefiniowanych w Tabeli 3.1 (bez wykorzystania F3).




Rys. 3.1. Schemat obwodu zasilania i sterowania trójfazowego silnika prądu przemiennego

Tabela 3.1. Definicje zmiennych (typu BOOL) sterowania silnika

| Nazwa zmiennej | Opis | Adres | Liczba styków | |
|---|---|---|---|---|
| | | | zwiernych | rozwiernych |
| S0 | Przycisk sterowniczy STOP | 1:01 | | 1 |
| S1 | Przycisk sterowniczy kierunek PRAWO | 1:02 | 1 | |
| S2 | Przycisk sterowniczy kierunek LEWO | 1:03 | 1 | |
| F2 | Styk wyłączający zabezpieczenia termicznego | 1:04 | | 1 |
| F3 | Cewka i styk wyłączający hamulca | 0:03 | | 1 |
| K1 | Cewka i styki przekaźnika załączającego PRAWO | 0:01 | 2 | 2 |
| K2 | Cewka i styki przekaźnika załączającego LEWO | 0:02 | 2 | 2 |
| H1 | Lampka sygnalizacji kier. obrotów PRAWO | 0:11 | | |
| H2 | Lampka sygnalizacji kier. obrotów LEWO | 0:12 | | |
| H3 | Lampka sygnalizacji stanu STOP | 0:13 | | |




Rys. 3.2. Schemat stykowy (przekaźnikowy) obwodu sterowania

## *Konfiguracja sterownika PLC i opracowanie programu sterowania w języku drabinkowym LD i języku bloków funkcyjnych FBD*

- Uruchomić program Concept (v.2.6 Trial) i otworzyć nowy projekt (**File | New project**).

- Określić rodzinę i typ sterownika w *PLC Selection*, np. Modicon Compact z CPU PC-E984-258, i zapoznać się z podziałem przestrzeni pamięci (*PLC Memory Partition*) na zmienne binarne (w tym wejścia) *Coils* (cewki) – adresy rozpoczynające się od 0:01, wejścia binarne *Discrete Inputs*– adresy rozpoczynające się od 1:01, rejestry wejściowe *Input Registers* (w tym m.in. wejścia analogowe) – adresy rozpoczynające się od 3:01, i rejestry podtrzymujące *Holding Registers* (w tym m.in. wyjścia analogowe) – adresy rozpoczynające się od 4:01.

- Określić moduły sterownika i przeprowadzić konfigurację przestrzeni wejść/wyjść: kliknąć na *I/O Map* i w otwartym oknie wcisnąć przycisk *Edit* w wierszu *Compact I/O*. Pod slotami przeznaczonymi dla CPU przypisać kolejne sloty kolejnym modułom:
  - *Discrete In* (moduł wejść binarnych DI, np. DC 16-Input 24V True High AS-BDE0216),
  - *Discrete Out* (moduł wyjść binarnych DO, np. DC 16-Output 24V 0.5A AS-BDA0216),
  - *Analog In* (moduł wejść analogowych AI, np. Analog 4 In AS-BADU204),
  - *Analog Out* (moduł wyjść analogowych AO, np. Analog Out 2 Channel Volt/Current AS-BDAU2x2),

Przypisać wejścia/wyjścia modułów adresom w pamięci sterownika. Zakresy adresów *In Ref* do *In End* oraz *Out Ref* do *Out End* muszą mieścić się w przedziałach określonych przez podział przestrzeni pamięci sterownika, np.:
- slot modułu DI – adres referencyjny (początkowy) *In Ref* = 1:01 (16 adresów),
- slot modułu DO – adres referencyjny *Out Ref* = 0:01 (16 adresów),
- slot modułu AI – adres referencyjny *In Ref* = 3:01 (4 rejestry),
- slot modułu AO – adres referencyjny *Out Ref* = 4:01 (2 rejestry),

Po zatwierdzeniu odwzorowania przestrzeni adresowej pamięci zapisać projekt na dysku (**File | Save project as**), np. jako `silnik.prj` (nazwa może składać się z maksimum 8 znaków).

- Zadeklarować zmienne algorytmu sterowania, menu **Project | Variable declarations...** (lub klawisz F8). Nazwy (*Variable name*) zmiennych binarnych (*Data Type* = BOOL) oraz ich adresy określić jak w Tabeli 3.1. Zmienne można usuwać (o ile nie pojawiają się w algorytmie przez

ustawienie kursora np. na nazwie, wciśnięcie klawisza **Delete** (spowoduje to zaznaczenie zmiennej do usunięcia, można przeznaczyć do usunięcia po kolei kilka zmiennych) i zatwierdzenie operacji przez OK.

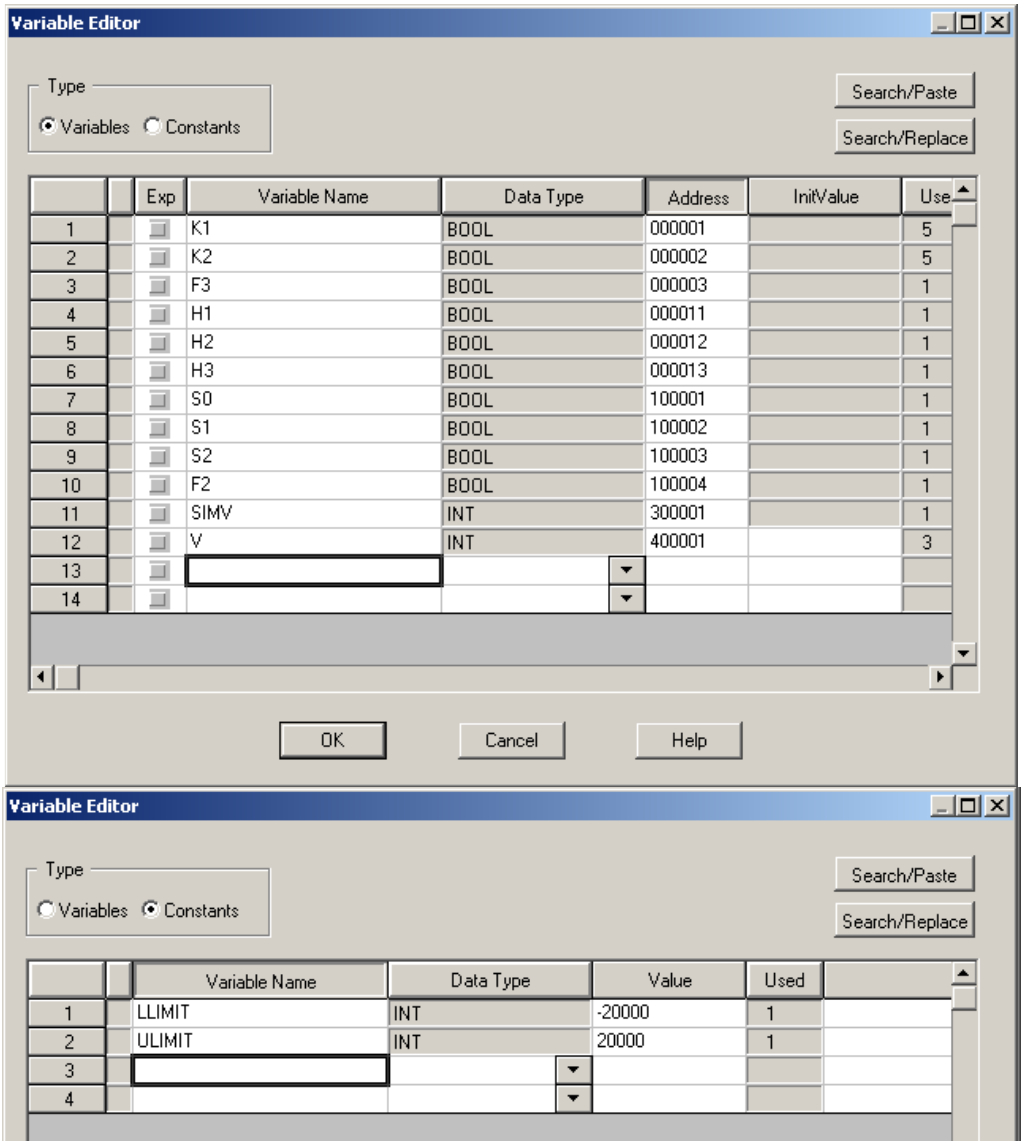
- Zadeklarować dodatkowe zmienne (*Variables*) i stałe (*Constans*) typu *Integer* (16 bitów, zakres -32768 do +32767) do symulacji i odczytu prędkości obrotowej silnika i uruchamiania wyłączenia nadprędkościowego:

  - zmienna *SIMV,* typ *INT,* rejestr wejściowy 3:01 (wejście 1 modułu Analog In),

  - zmienna *V,* typ *INT,* rejestr wyjściowy 4:01 (prędkość obrotowa, wyjście 1 modułu Analog Out),

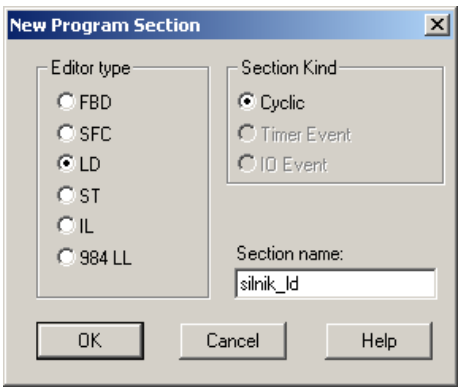
  - stałe *LLIMIT* (=-20000), *ULIMIT* (=20000), typ *INT,* określają dolny (obroty w lewo) i górny (obroty w prawo) limit prędkości obrotowej, powyżej których uaktywnia się zabezpieczenie nadprędkościowe.

Okno edytora zmiennych z listą wszystkich zmiennych i stałych programu jest pokazane na rys. 3.3.



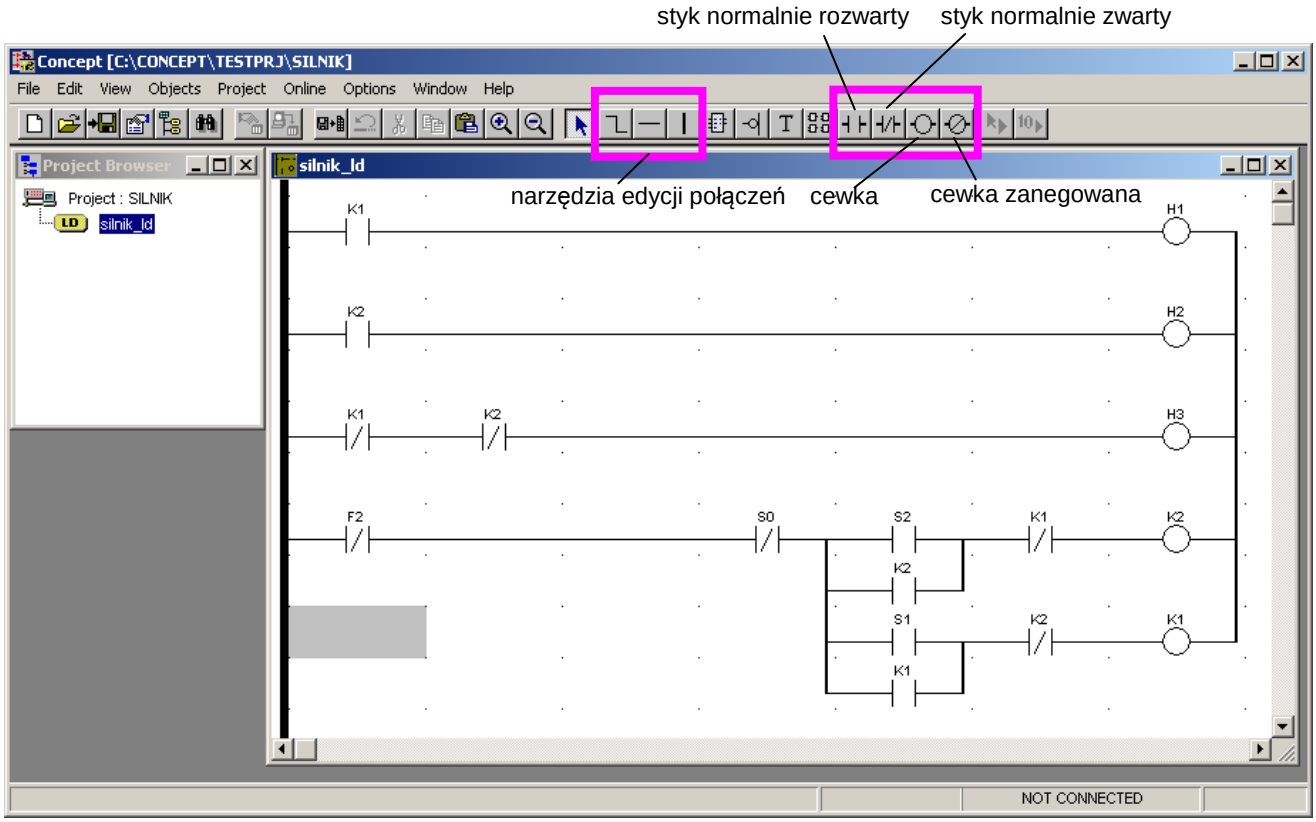

Rys. 3.3. Okno edytora zmiennych programu Concept: deklaracje zmiennych algorytmu sterowania silnikiem

- Utworzyć w projekcie główną sekcję programu sterowania w formie schematu drabinkowego LD poleceniem **File | New section** (rys. 3.4). Nadać sekcji nazwę *silnik_ld*.




Rys. 3.4. Tworzenie nowej, realizowanej cyklicznie, sekcji programu o nazwie *silnik_ld* w formie schematu drabinkowego LD

- W oknie utworzonej sekcji przeprowadzić edycję schematu drabinkowego zgodnego ze schematem stykowym układu sterowania z rys. 3.2 za pomocą narzędzi z górnej belki edytora LD. Przykładowy efekt końcowy jest pokazany na rys. 3.4. Zapisać projekt na dysk.

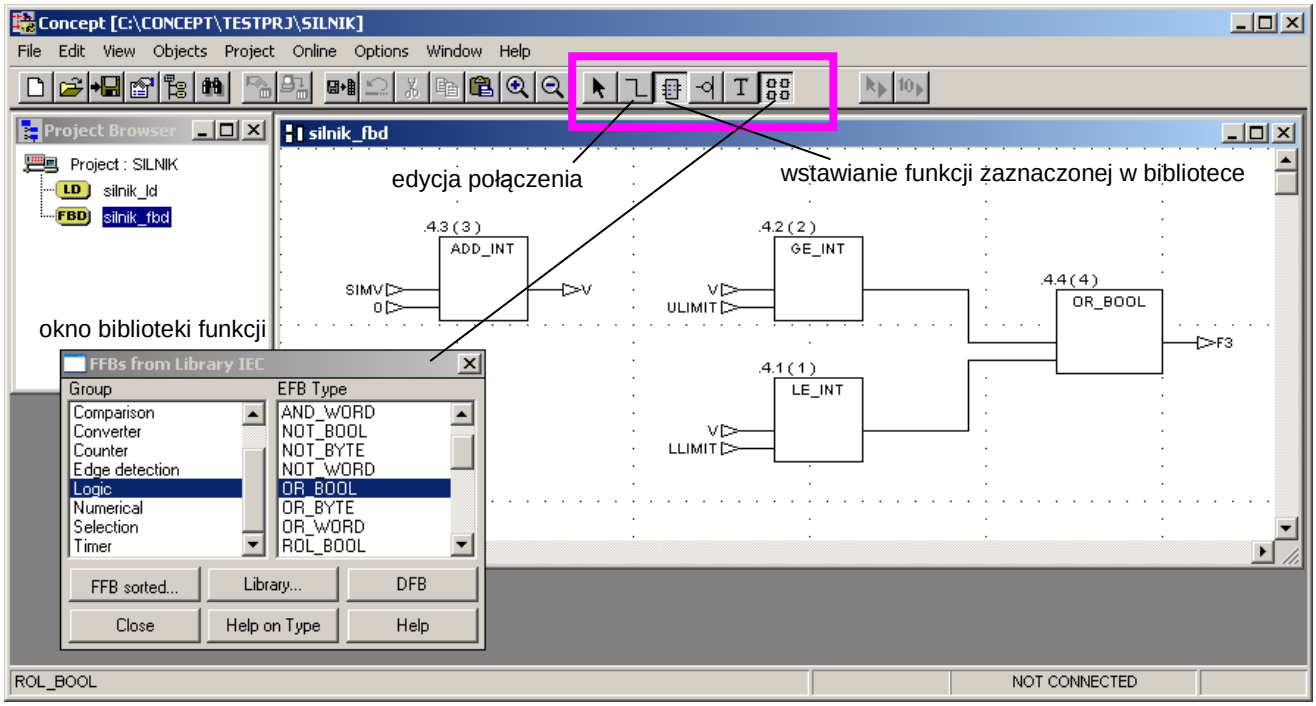



Rys. 3.4. Realizacja drabinkowa schematu stykowego układu sterowania z rys. 3.2.

- Utworzyć w projekcie dodatkową sekcję programu w formie schematu funkcji logicznych FBD o nazwie *silnik_fbd*. Będzie to sekcja symulacji i pomiaru prędkości obrotowej silnika oraz logiki uaktywniania wyłączania nadprędkościowego (hamulca).

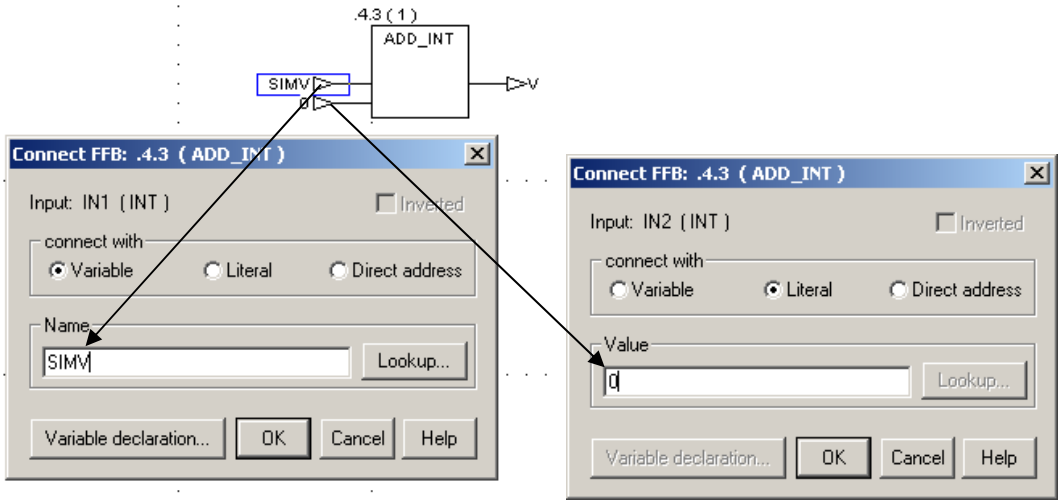
- W oknie utworzonej sekcji przeprowadzić edycję schematu logicznego jak na rys. 3.5. za pomocą narzędzi z górnej belki edytora LD.
  - *ADD_INT* – funkcja dodawania liczb typu *Integer* ze znakiem (grupa funkcji *Arithmetic* w bibliotece FFB). Zadaniem bloku jest odczyt sygnału (analogowego) symulującego prędkość obrotową silnika z wejścia 3:01 modułu AI (zmienna *SIMV*) do zmiennej *V* - rejestru pamięciowego 4:01 (przy zdefiniowanej konfiguracji sterownika rejestr ten jest odwzorowywany na jedno z wyjść modułu AO). Stan wejścia AI będzie dalej symulowany położeniem slidera na

panelu symulatora sterownika.
- *GE_INT* (*Greater than or Equal to* – większy lub równy), *LE_INT* (*Less than or Equal to* – mniejszy lub równy) – funkcje porównania dwóch liczb typu *Integer* (grupa funkcji *Comparison* w bibliotece FFB). Zadaniem bloków jest sprawdzenie, czy prędkość silnika *V* nie przekracza ograniczenia *ULIMIT* (lub *LLIMIT*). Wyjście typu Boolean jest aktywne jeśli *V≥ULIMIT* (odpowiednio *V≤ULIMIT*).
- *OR_BOOL* – funkcja logiczna *OR* argumentów typu *Boolean* (grupa funkcji *Logic* w bibliotece FFB). Wyjście jest aktywne, jeżeli prędkość silnika przekracza którekolwiek z ograniczeń.




Rys. 3.5. Sekcja pomocnicza programu w formie schematu logicznego FBD.

Po wstawieniu w obszar schematu bloku funkcyjnego należy zdefiniować jego zmienne (stałe) wejściowe i zmienne wyjściowe. Po dwukrotnym kliknięciu portu otwiera się okno *Connect FFB* jak na rys. 3.6. Nazwę zmiennej można wpisać bezpośrednio lub wybrać z listy (przycisk *Lookup...*).

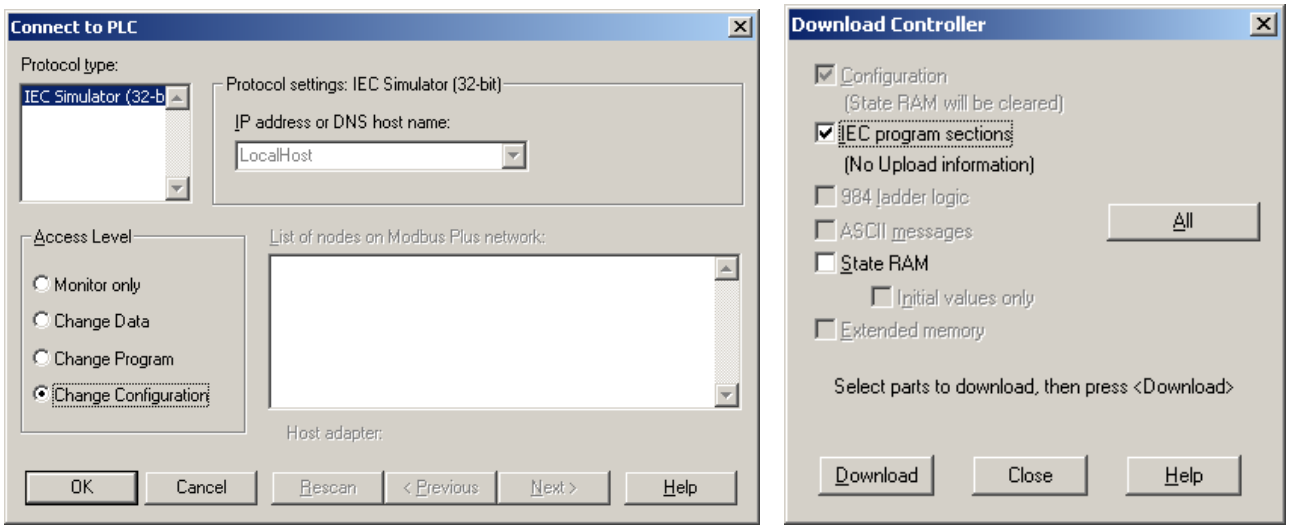



Rys. 3.6. Określanie zmiennych (*Variable*) wejściowych/wyjściowych i stałych (*Literal*) wejściowych bloków funkcyjnych schematu FDB.

- Ustawić kolejność realizacji sekcji programu poleceniem **Project | Execution order**. Ponieważ w przypadku naszego programu wynik działania sekcji *silnik_fbd*, tj. ewentualne stwierdzenie

przekroczenia ograniczenia prędkości, może oddziaływać na logikę sterowania, powinna ona być realizowana jako pierwsza (*Execute as First*), żeby uniknąć opóźnień.

- Zapisać projekt na dysk i przejść do uruchamiania programu na symulatorze sterownika PLC. Realizuje się to przez wybranie polecenia **Online | Connect** (rys. 3.7), które wywołuje symulator PLC. W trakcie ustanawiania połączenia (TCP/IP) należy potwierdzać za pomocą OK ewentualne ostrzeżenia. W przypadku niewłaściwego zdefiniowania w symulatorze typu sterownika, należy ustawić taki sam sterownik jak w uruchamianym projekcie, tj. w naszym przypadku PC-E984-258.




Rys. 3.7. Łączenie środowiska Concept z programowym symulatorem *IEC Simulator* i ładowanie programu.

- Załadować do symulatora program i konfigurację sterownika poleceniem **Online | Download** i uruchomić sterownik (*Download complete. Start controller?* → TAK).

- W głównym oknie środowiska Concept otworzyć okna obu sekcji programu (menu **File | Open section**, jeśli były zamknięte, lub przez **Project | Project Browser**). Dla obu sekcji uruchomić animację zmiennych binarnych zaznaczeniem **Online | Animate booleans** (Ctrl-Y) (rys. 3.8)

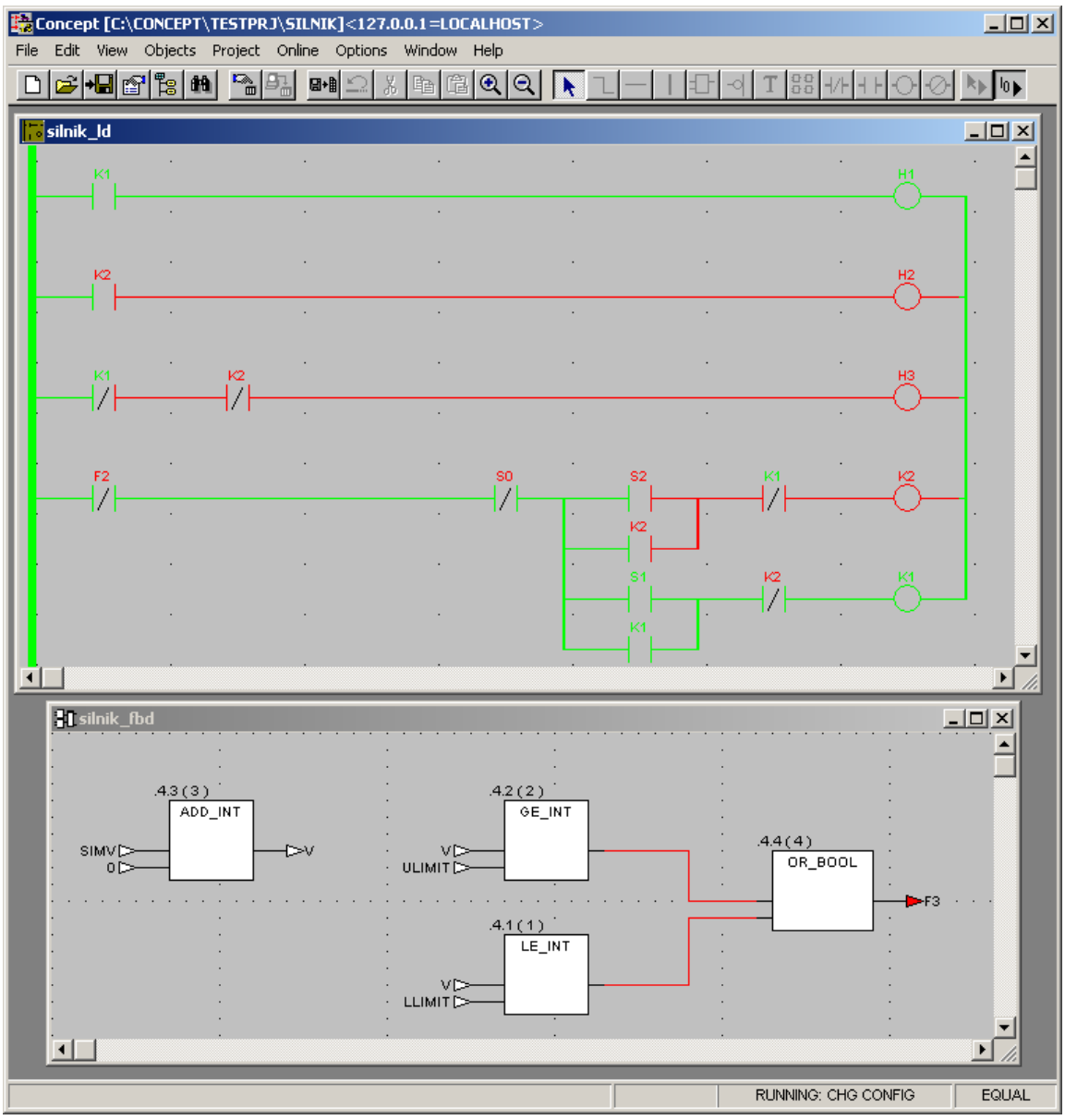
- Przeanalizować działanie algorytmu sterowania zmieniając w oknie symulatora (rys. 3.9) stany wejść odpowiadające przyciskom sterowniczymi (PRAWO, LEWO, STOP) obserwując spowodowane tym reakcje algorytmu obrazowane przez animacje schematów. Za pomocą suwaka można symulować prędkość obrotową silnika (jest to połączone ze zmianą stanu - wysokości słupka – rejestru wyjściowego sprzęgniętego z tym wejściem) sprawdzić efekt przekroczenia limitu prędkości.

Sprawdzić, że zakładka I*/O Modules* symulatora pokazuje (po załadowaniu konfiguracji sterownika i programu) stany rejestrów zdefiniowanych modułów wejścia-wyjścia sterownika, zakładka *Connections* – stany połączeń komunikacyjnych.

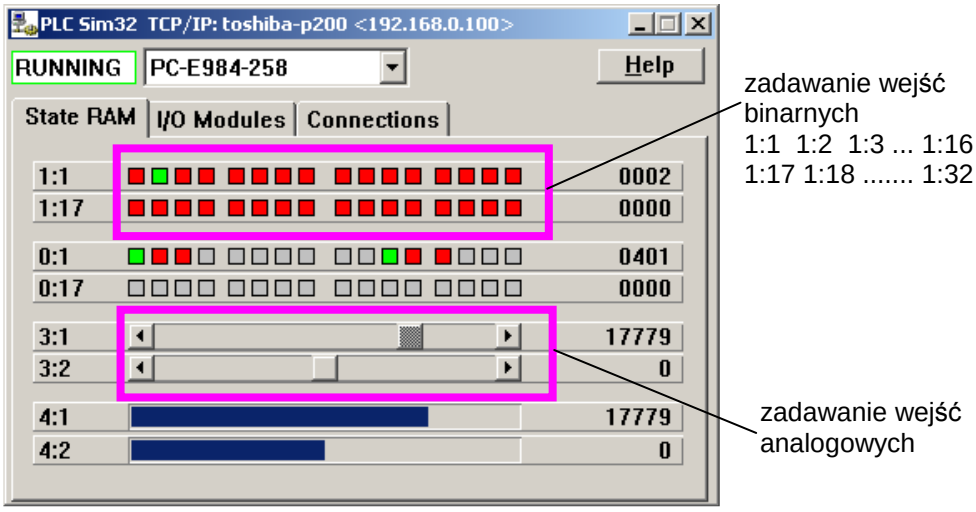
Modyfikacji zmiennych programu można też dokonać z poziomu edytora danych - menu **Online | Reference Data Editor (**lub *Ctrl-R***).**

- Działanie programu przerywa się poleceniem **Online | Online Control Panel** → *Stop controller*. Połączenie z symulatorem przerywa się poleceniem **Online | Disconnect**.

- Zmodyfikować w prosty sposób sekcję logiki sterowania w taki sposób, aby aktywny stan zmiennej F3 (przekroczenie limitu prędkości) powodował zatrzymanie silnika. Uruchomić na symulatorze i przeanalizować zmodyfikowany program.




Rys. 3.8. Okna sekcji uruchomionego programu z włączoną animacją zmiennych logicznych (kolor czerwony = stan 0, kolor zielony = stan 1)

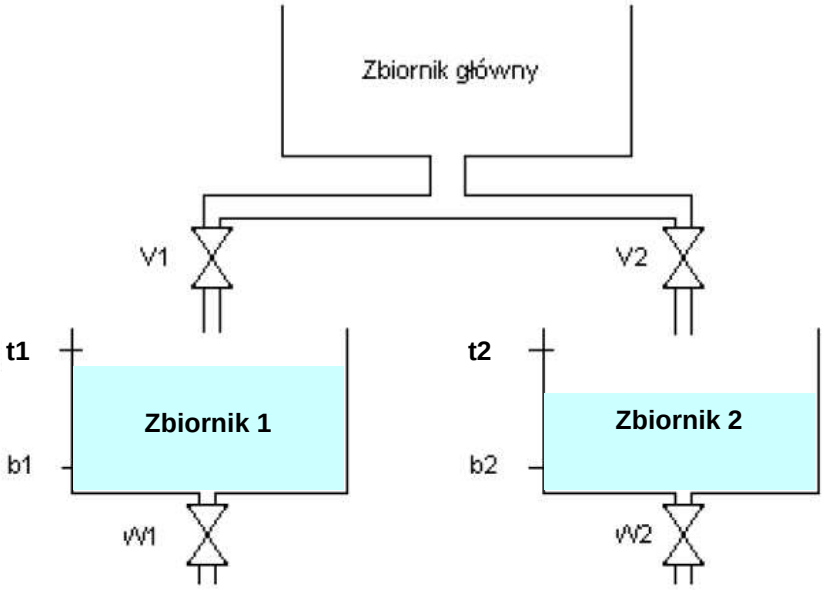



Rys. 3.9. Okno symulatora w trakcie działanie programu (RUNNING). Zaznaczone zadajniki zmiennych binarnych (kliknięcie na kwadracik zmienia stan wejścia o danym adresie, czerwony = 0, zielony = 1) analogowych (przez przesuwanie suwaków, rejestry wejściowe interpretowane jako zmienne typu *Integer*)

## 3.2. Sterowanie napełnianiem i opróżnianiem dwóch zbiorników – algorytm SFC

Zadanie polega na cyklicznym napełnianiu i opróżnianiu cieczą dwóch zbiorników (rys. 3.10). Napełnianiem i opróżnianie realizuje się poprzez włączanie odpowiednio zaworów górnych (napełniających) V1, V2 i dolnych (opróżniających) W1, W2. W stanie początkowym oba zbiorniki są próżne. Napełnianie rozpoczyna się po wciśnięciu (w każdym cyklu) przycisku sterującego Key1. Zakładamy, że zbiornik jest napełniony, jeżeli poziom cieczy w zbiorniku znajduje się powyżej pewnego poziomu górnego. Napełnienie będzie sygnalizowane ustawieniem (stan 1) zmiennych logicznych, odpowiednio t1 dla zbiornika 1 i t2 dla zbiornika 2. Po napełnieniu zbiornika ma być wyłączony jego zawór górny. Po napełnieniu obu zbiorników należy odczekać pewien czas i uruchomić opróżnianie zbiorników. Analogicznie, przyjmujemy, że zbiornik jest opróżniony, jeżeli poziom cieczy w zbiorniku znajduje się poniżej pewnego poziomu dolnego. Opróżnienie będzie sygnalizowane ustawieniem zmiennych logicznych, odpowiednio b1 dla zbiornika 1 i b2 dla zbiornika 2. Po opróżnieniu zbiornika ma być wyłączony jego zawór dolny.

Kolejny cykl napełniania może rozpocząć się dopiero wtedy, gdy oba zbiorniki będą opróżnione i zostanie naciśnięty przycisk Key1.




Rys. 3.10. Schemat układu napełniania dwóch zbiorników

Algorytm sterowania zostanie zrealizowany w formie schematu sekwencyjnych zmian stanu SFC (*Sequential Function Chart*) z dwiema realizowanymi równolegle (współbieżnie) gałęziami akcji napełniania i opróżniania zbiornika 1 i zbiornika 2.

### *Opracowanie programu sterowania w języku sekwencyjnych zmian stanu SFC*

- W programie Concept utworzyć nowy projekt (**File | New project**).
- Określić rodzinę i typ sterownika w *PLC Selection* (np. Modicon Compact z CPU PC-E984-258 jak w pkt. 3.1). Pominiemy etap konfiguracji modułów wejść/wyjść i przestrzeni zmiennych sterownika przyjmując konfigurację domyślną *State RAM* symulatora *IEC Simulator*.
- Zadeklarować zmienne algorytmu sterowania, menu **Project | Variable declarations...** (lub klawisz F8). Nazwy (*Variable name*) typy (*Data Type*) i adresy (*Address*) zmiennych określić jak w Tabeli 3.2. Końcowa lista deklaracji zmiennych jest pokazana na rys. 3.10. Zapisać projekt na dysku (**File | Save project as**), np. jako `zbiornik.prj` (nazwa maksimum 8 znaków).

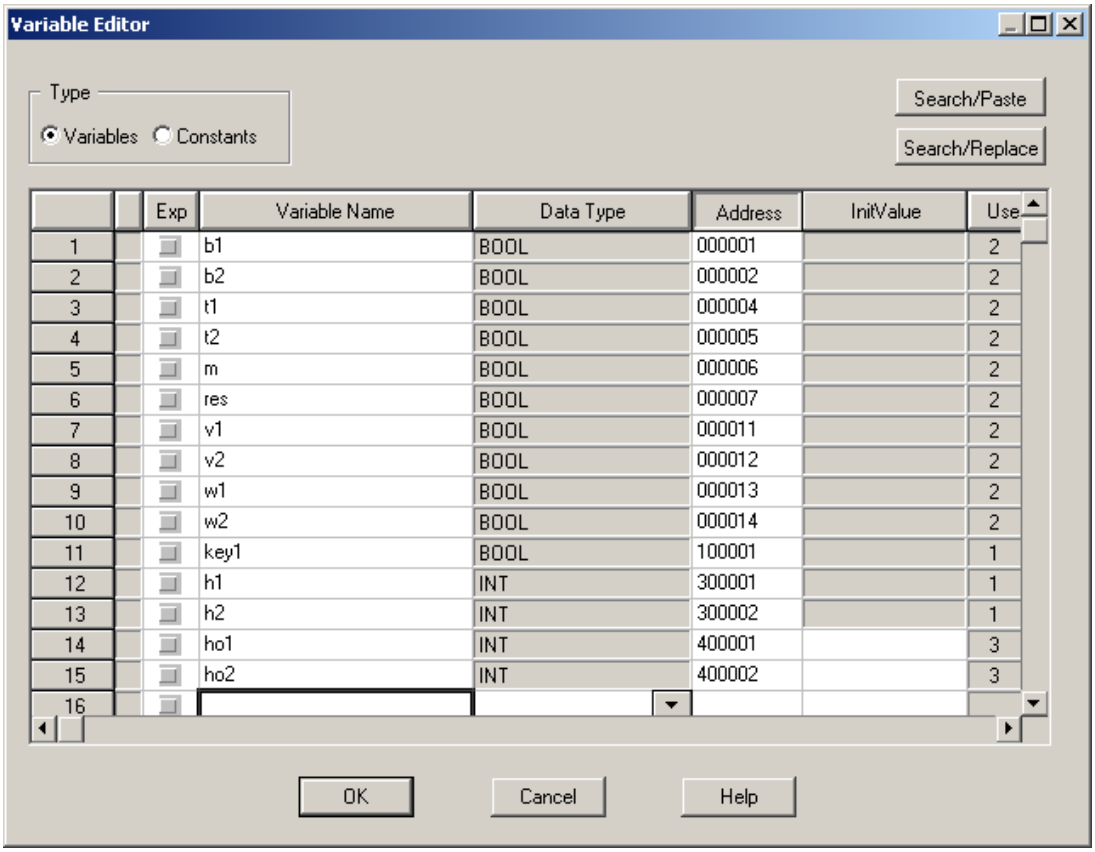
Uwaga: Concept dopuszcza deklarowanie zmiennych (pomocniczych) nie adresowanych, o ile nie będzie potrzeby mapowania ich na przestrzeń adresową wejść/wyjść.

Dodatkowe zmienne typu *Integer*: *h1, h2* (wejścia „analogowe" pomiaru poziomu) oraz *ho1, ho2* (rejestry wewnętrzne) są przewidziane do symulacji i odczytu poziomu cieczy w zbiornikach i wyznaczania stanów zmiennych binarnych sygnalizujących napełnienie lub opróżnienie zbiornika.

Uwaga: Zmienne te są zdefiniowane jako typ *Integer* (zakres -32768÷32767), a nie *Word* (nieujemny zakres 0÷65535) z powodu sztywnego ustawienia zakresów rejestrów w oknie symulatora *IEC Simulator* jak dla typu *Integer*. Będziemy rozumieć, że wartość równa -32768 oznacza poziom cieczy równy 0 – zbiornik pusty.

Tabela 3.2. Opis zmiennych algorytmu sterowania procesem napełniania i opróżniania zbiorników

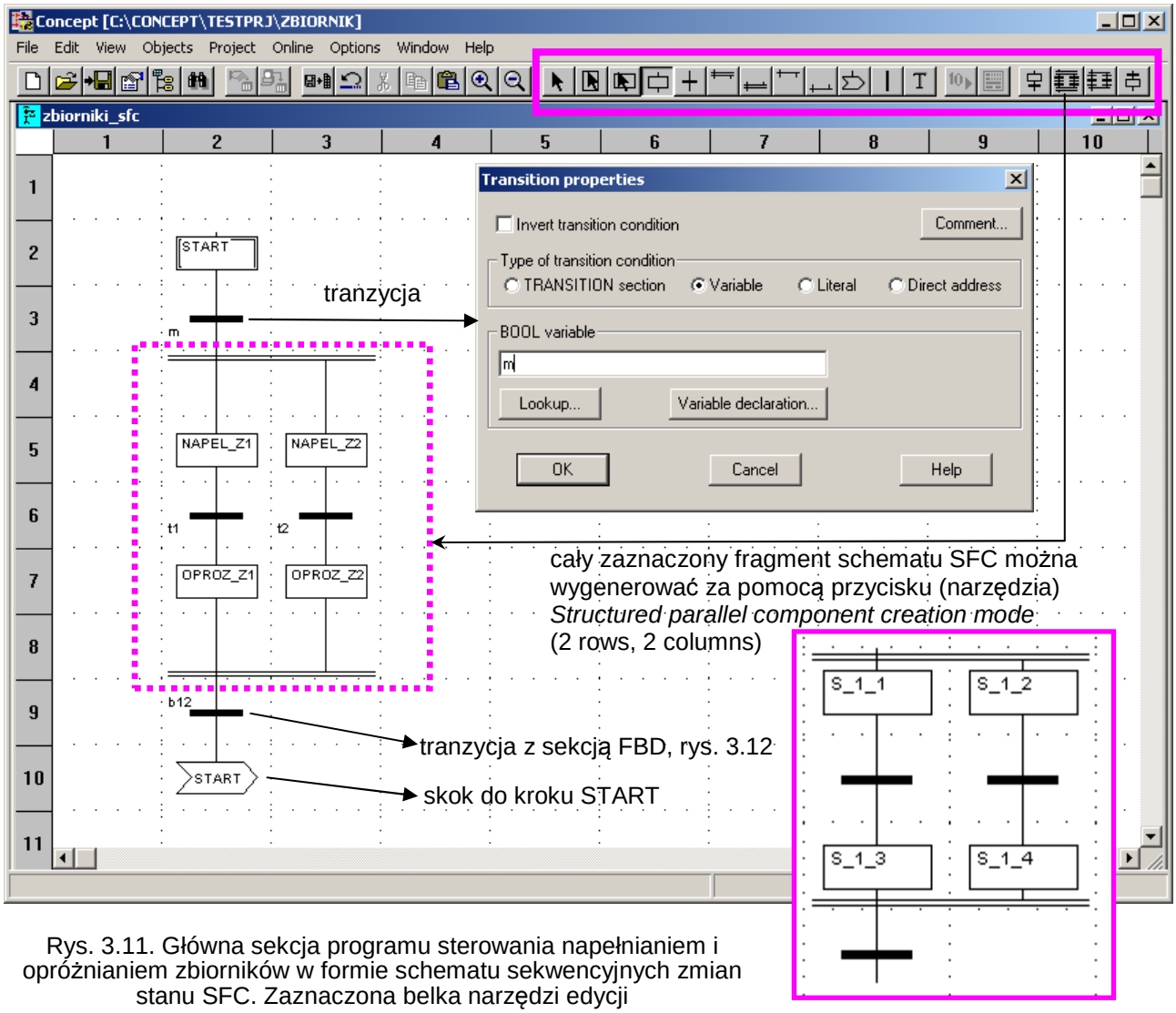
| Nazwa zmiennej | Opis | Adres |
|---|---|---|
| b1 | Zbiornik 1 opróżniony (bottom 1) | |
| b2 | Zbiornik 2 opróżniony (bottom 2) | |
| t1 | Zbiornik 1 napełniony (top 1) | |
| t2 | Zbiornik 2 napełniony (top 2) | |
| m | Zmienna (tranzycji) startu cyklu | |
| res | Reset zmiennej m tranzycji startu cyklu | |
| v1 | Zawór górny zbiornika 1 otwarty (napełnianie) | |
| v2 | Zawór górny zbiornika 2 otwarty (napełnianie) | |
| w1 | Zawór dolny zbiornika 1 otwarty (opróżnianie) | |
| w2 | Zawór dolny zbiornika 2 otwarty (opróżnianie) | |
| key1 | Przycisk sterujący startu cyklu napełniania | |
| h1 | Czujnik pomiarowy napełnienia zbiornika 1 (rejestr wej.) | |
| h2 | Czujnik pomiarowy napełnienia zbiornika 2 (rejestr wej.) | |
| ho1 | Poziom napełnienia zbiornika 1 (rejestr pamięciowy wyj.) | |
| ho2 | Poziom napełnienia zbiornika 2 (rejestr pamięciowy wyj.) | |




Rys. 3.10. Deklaracje zmiennych algorytmu cyklicznego napełniania i opróżniania zbiorników

- Utworzyć w projekcie główną sekcję programu sterowania w formie schematu sekwencyjnych zmian stanu SFC poleceniem **File | New section**. Nadać sekcji nazwę *zbiornik_sfc*.

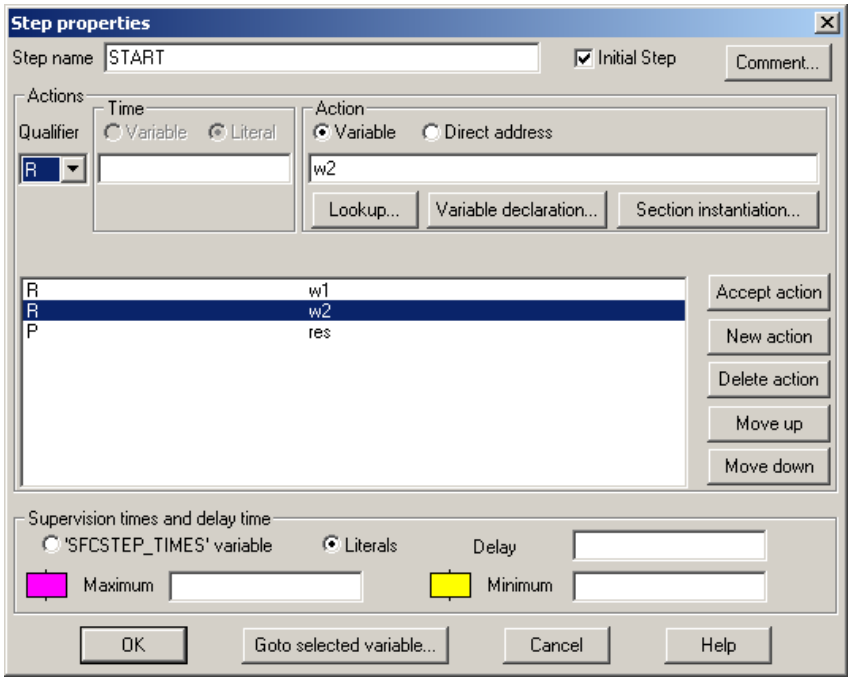
- W oknie utworzonej sekcji przeprowadzić edycję schematu SFC za pomocą narzędzi z górnej belki edytora SFC. Końcowy schemat SFC jest pokazany na rys. 3.11. Zapisać projekt na dysk.




Rys. 3.11. Główna sekcja programu sterowania napełnianiem i opróżnianiem zbiorników w formie schematu sekwencyjnych zmian stanu SFC. Zaznaczona belka narzędzi edycji

## Okna edycji kroków (*Step*) schematu SFC

- *Krok START*

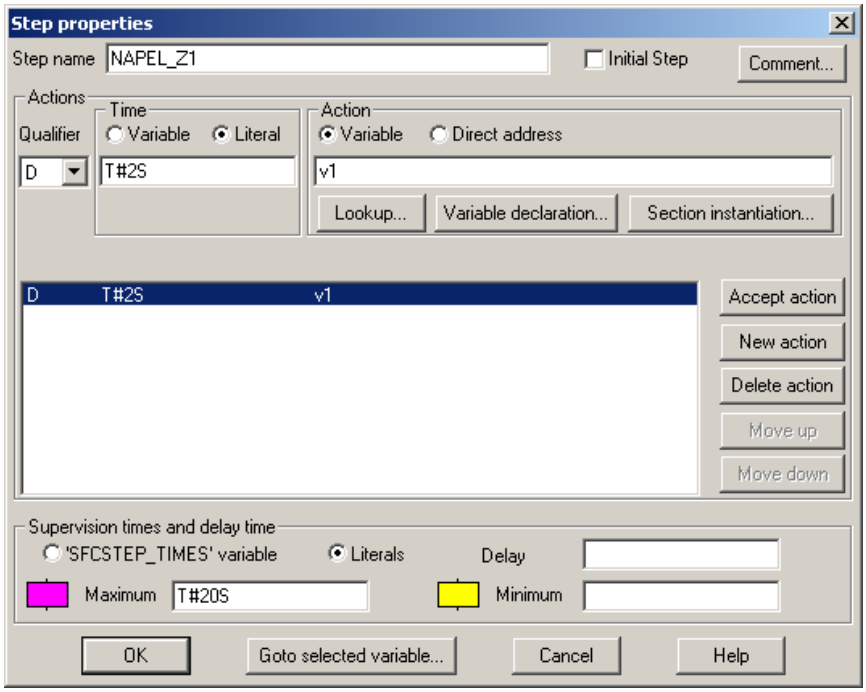



Krok początkowy schematu (zaznaczyć opcję *Initial Step*).

Realizuje reset (kwalifikator akcji R – *overriding reset*) akcji (zmiennych logicznych) *w1, w2*, czyli zamknięcie zaworów dolnych oraz ustawia impulsowo (qualifier *P – pulse*) akcję *res* wykorzystywaną do resetowania przerzutnika RS przycisku sterującego *Key1*, por. schemat FBD na rys. 3.13.

*Tranzycja m* (wyjście z kroku *START*) następuje po spełnieniu warunku *m*=1 (wciśnięciu przycisku *Key1*).

➢ *Kroki NAPEL_Z1, NAPEL_Z2*




Realizowane równolegle kroki napełniania zbiorników.

*NAPEL_Z1* realizuje ustawienie (nadanie wartości =1) akcji (zmiennej) *v1* (akcji *v2* w *NAPEL_Z2*), tzn. otwarcie zaworu napełniającego z zadanym opóźnieniem T=2s od momentu uaktywnienia kroku (kwalifikator *D* - *delayed*). Akcja *v1* jest ustawiona *tylko do końca aktywności kroku*, czyli napełnienia zbiornika.

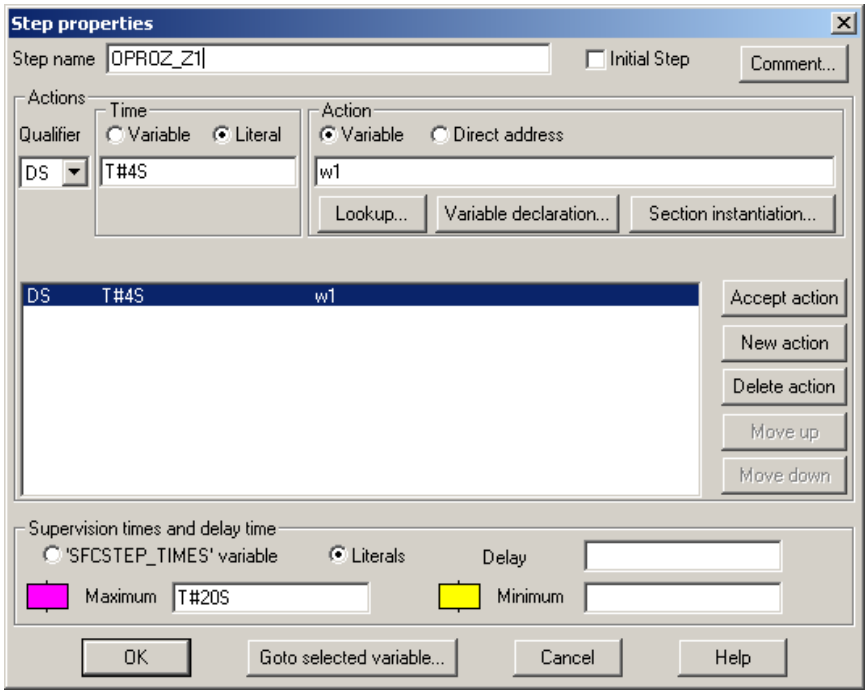
Warunkiem *tranzycji t1*, czyli przejścia do następnego kroku *OPROZ_Z1 jest n*apełnienie zbiornika, tzn. spełnienie warunku *t1*=1. Analogiczny jest warunek tranzycji *t2*. Po wyjściu z kroku *NAPEL_Z1* zawór napełniający jest zamykany.

W kroku *NAPEL_2* ustawić inną wartość opóźnienia otwarcia zaworu, np. T=3s.

W dolnym polu *Supervision times and delay time* można określić kontrolę minimalnego i maksymalnego czasu trwania kroku (*Delay<Minimum<Maximum*). Przekroczenie ograniczenia powoduje reakcję nadzoru – na animacji schematu SFC zmienia się kolor bloku kroku (w pokazanym przykładzie nadzór zareaguje zmianą koloru bloku, jeżeli krok będzie trwał dłużej niż 20s). Często stosowanymi kwalifikatorami akcji kroku (nie wykorzystywanymi w realizowanym programie) są:

- kwalifikator N (*neutral/not saved*) – akcja jest wykonywana tylko w czasie aktywności kroku (działa jak D z opóźnieniem 0),
- kwalifikator L (*time limited*) – akcja wykonywana jest w ograniczonym, określonym przedziale czasu, ale nie dłużej niż do końca aktywności kroku.

➢ *Kroki OPROZ_Z1, OPROZ_Z2*

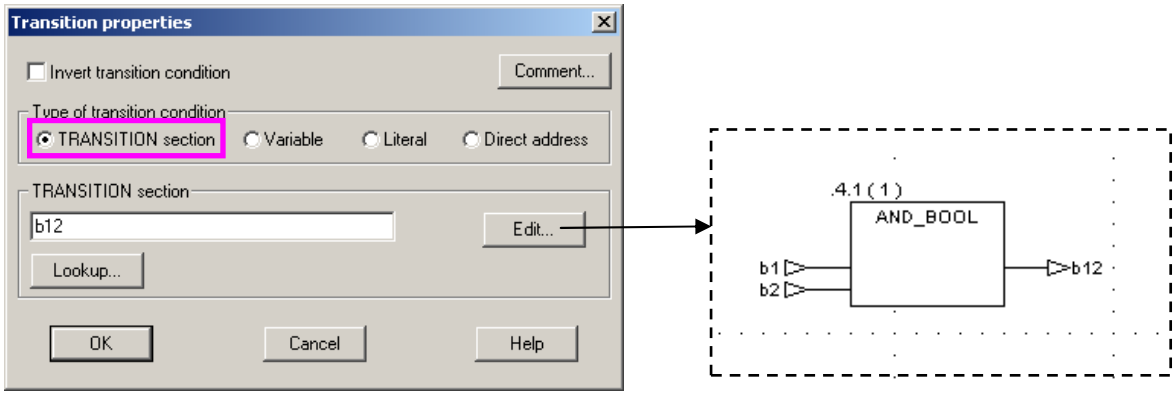



Realizowane równolegle kroki opróżniania zbiorników.

*OPROZ_Z1* realizuje ustawienie zmiennej *w1* (*w2* w *OPROZ_Z2*), tzn. otwarcie zaworu opróżniającego z zadanym opóźnieniem T=4s od momentu uaktywnienia kroku (kwalifikator *DS* – *delayed and set/saved*). Zmienna *w1* pozostaje w tym przypadku ustawiona *również po wyjściu z kroku*, czyli opróżnieniu zbiornika.

Warunkiem *tranzycji b12* kończącej cykl (rys. 3.12) jest *b1 AND b2 = 1 (TRUE)*, czyli opróżnienie obu zbiorników. Po wyjściu z kroku zawór

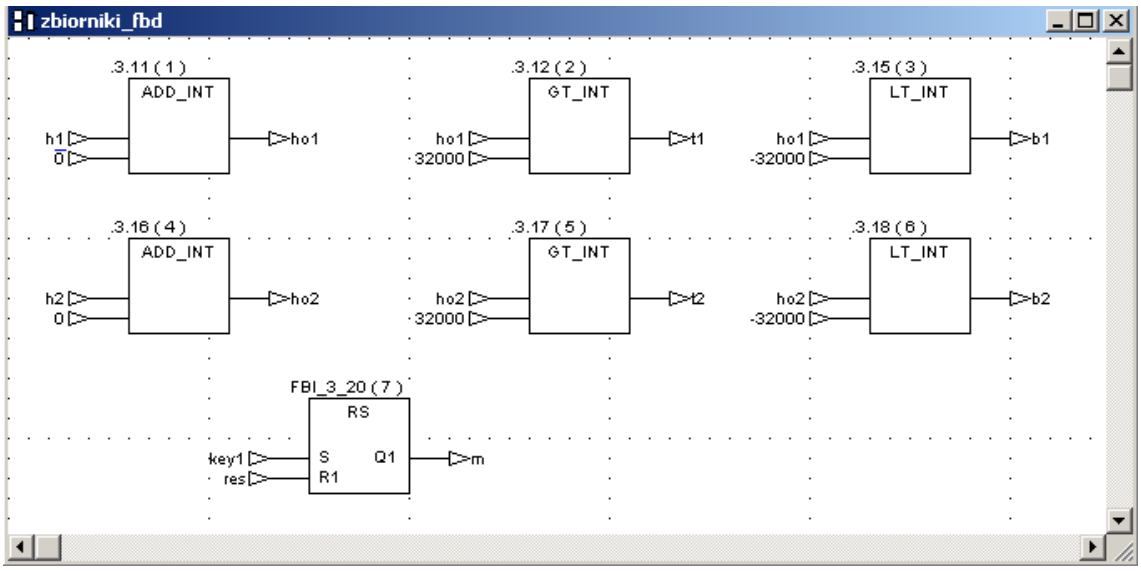
opróżniający pozostaje otwarty - aż do resetu zmiennej *w1* (*w2* dla zbiornika 2) w kroku *START*.




Rys. 3.12. Tranzycja zrealizowana z wykorzystaniem sekcji FBD. Tranzycja realizowana pod warunkiem (b1 AND b2) = TRUE (oba zbiorniki opróżnione)

- Utworzyć w projekcie (**File | New section**) dodatkową sekcję programu w formie schematu funkcji logicznych FBD o nazwie *zbiornik_fbd*. Będzie to sekcja symulacji zmian poziomów cieczy (rejestry *ho1, ho2*) w zbiornikach oraz ustawiania flag napełnienia i opróżnienia zbiorników (generowania stanów zmiennych logicznych odpowiednio *t1, t2* i *b1, b2*).

- W oknie utworzonej sekcji przeprowadzić edycję schematu logicznego FBD jak na rys. 3.13.
  - *ADD_INT* – funkcja dodawania liczb typu *Integer* ze znakiem (grupa funkcji *Arithmetic* w bibliotece FFB). Zadaniem bloków jest odczyt sygnałów (analogowych) symulujących poziomy cieczy w zbiornikach i zapisanie ich w rejestru sterownika. Stan wejść analogowych będą symulowane położeniami suwaków na panelu symulatora sterownika.
  - *GT_INT* (*Greater Than* – większy niż), *LT_INT* (*Less Than* – mniejszy niż) – funkcje porównania dwóch liczb typu *Integer* (grupa funkcji *Comparison* w bibliotece FFB). Bloki porównują poziomy cieczy z wartościami granicznymi: *ho*>32000 ustawia zmienną napełnienia zbiornika *t1* (*t2*), *ho*<-32000 ustawia zmienną opróżnienia zbiornika *b1* (*b2*).
  - *RS* – przerzutnik *RS* (grupa funkcji *Bistable* w bibliotece FFB). Jeżeli *res*=0 i *key1* przyjmuje wartość 1, to wyjście *m* przyjmuje wartość 1, która pozostanie nawet jeśli *key1* powróci do stanu 0. Jeżeli *res*=0, to *m*=0 niezależnie od stanu *key1* (dominujący reset stanu przycisku startu cyklu). Ponowne ustawienie wyjścia po resecie nastąpi dopiero po zmianie *key1* z 0 na1.




Rys. 3.13. Sekcja pomocnicza programu sterowania: symulacja zmian poziomów cieczy (rejestry *ho1, ho2*), ustawianie flag napełnienia i opróżnienia zbiorników oraz przerzutnik przycisku sterującego *Key1*

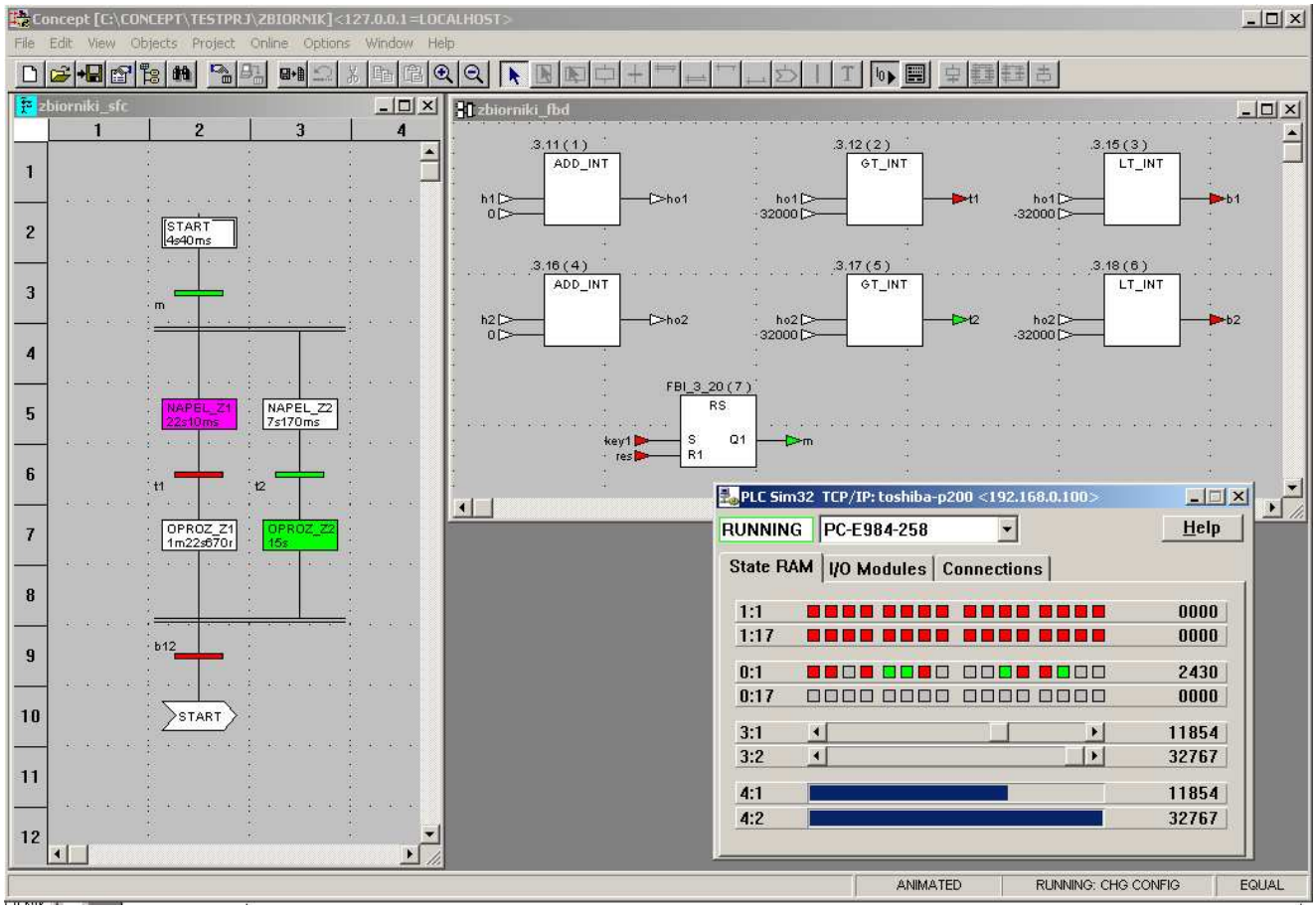
- Ustawić kolejność realizacji sekcji programu poleceniem **Project | Execution order**. Sekcja popmocnicza *zbiornik_fbd* ma być realizowana jako pierwsza (*Execute as First*).

- Zapisać projekt na dysk i przejść do uruchamiania programu na symulatorze sterownika PLC poleceniem **Online | Connect** (w trakcie ustanawiania połączenia potwierdzać przez OK ewentualne ostrzeżenia).

- Załadować do symulatora program i konfigurację sterownika poleceniem **Online | Download** i uruchomić sterownik (*Download complete. Start controller?* → TAK).

- W głównym oknie środowiska Concept otworzyć okna obu sekcji programu (menu **File | Open section**, jeśli były zamknięte, lub przez **Project | Project Browser**). Uruchomić animację schematu SFC przez **Online | Animate** i animację schematu FBD przez **Online | Animate booleans** (Ctrl-Y).

- Przeanalizować działanie algorytmu sterowania zmieniając w oknie symulatora (rys. 3.9) stan wejścia 1:01 odpowiadającego przyciskowi sterowniczemu *Key1,* oraz poziomy cieczy w zbiornikach za pomocą przesuwania suwaków (3:01 dla zbiornika 1, 3:02 dla zbiornika 2). Zacząć od przesuwania suwaków w prawo, a po osiągnięciu napełnienia (ustawienie *t1* i/lub *t2*) – w lewo (opróżnianie). Kolejny cykl rozpoczyna ponowne wciśnięcie (zmiana na kolor zielony) przycisku Key1, przy czym musi on być zwolniony (kolor czerwony) od poprzedniego wciśnięcia.




Rys. 3.14. Symulacja działania sekcji SFC i FBD programu napełniania i opróżniania zbiorników. Kroki i tranzycje aktywne oznaczone są kolorem zielonym. W blokach kroków schematu SFC podawane są czasy timerów kroków.

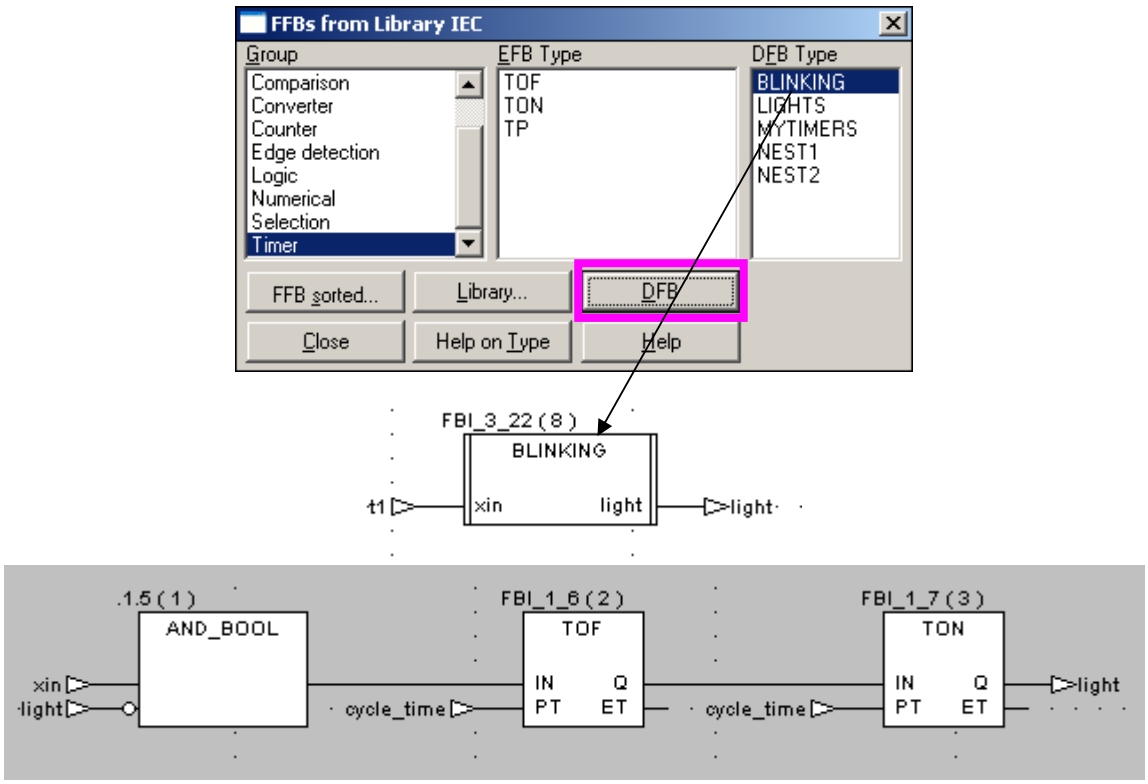
- Rozbudować algorytm w sekcji logiki FBD o sygnalizację napełnienia zbiornika 1 w postaci migającej lampki kontrolnej. W tym celu zadeklarować (**Project | Variable declarations**) dodatkową zmienną powiązaną z wyjściem, np. 0:17, sterującym kontrolką:

  - zmienna *light,* typ *BOOL,* adres (rejestr wyjściowy) 0:17

Zadaną funkcję realizuje blok złożonej funkcji pochodnej DFB (*Derived Function Block*) o nazwie *BLINKING* (rys. 3.15). Funkcje DFB mogą być definiowane i zapisywane przez użytkownika tak jak zwykłe projekty. Strukturę wewnętrzną funkcji DFB można obejrzeć po dwukrotnym kliknięciu bloku funkcji i wybranie przycisku *Refine*. W funkcji BLINKING wykorzystano standardowe funkcje z

grupy *Timer*: TOF i TON, timery odpowiednio z opóźnionym wyłączeniem i z opóźnionym załączeniem.

W przypadku naszego programu jako wejście funkcji BLINKING wybieramy zmienną *t1* sygnalizującą napełnienie zbiornika 1, a jako wyjście – nowo zdefiniowaną zmienną *ligth*. Jeżeli *t1* przyjmie wartość 1, to stan zmiennej *light* będzie się zmieniał co *cycle_time*=0.5s.




Rys. 3.15. Schemat logiczny funkcji złożonej BLINKING.DFB cyklicznego włączania i wyłączania zmiennej logicznej *ligth* co *cycle_time* = 500ms (*cycle_time* jest zdefiniowany jako stała (Constant) typu TIME równa T#500ms), pod warunkiem, że zmienna logiczna *xin*=1

## 4. Opracowanie sprawozdania

W sprawozdaniu należy opisać i skomentować działanie implementowanych algorytmów ze szczególnym uwzględnieniem modyfikacji zadanych przez prowadzącego.

### Literatura

1. Kasprzyk J.: *Programowanie sterowników przemysłowych*, WNT, 2006.
2. Legierski T., Wyrwał J., Kasprzyk J., Hajda J.: *Programowanie sterowników PLC*, Wyd. Pracowni Komputerowej J.Skalmierskiego, 1998.
3. Mikulczński T., Samsonowicz Z.: *Automatyzacja dyskretnych systemów produkcyjnych*, WNT, 1997.
4. Pliki pomocy pakietu *Concept* firmy *AEG Schneider Electric*.

Opracowanie:
Dr inż. Janusz Baran
Dr inż. Sebastian Dudzik