МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

А.М.Ніколаєнко

ПРОГРАМУВАННЯ ПЛК У Softlogic-СИСТЕМІ KW MULTIPROG

Навчальний посібник

Рекомендовано
Міністерством освіти і науки України
як навчальний посібник для студентів
вищих навчальних закладів

Запоріжжя ЗДІА, 2008

УДК 62.529(075.8)

Ніколаєнко А.М. Програмування ПЛК у Softlogic-системі KW MULTIPROG: Навч.посіб.- Запоріжжя: Видавництво Запорізької державної інженерної академії, 2008. - 202 с.

Р е ц е н з е н т и:

Бєляєв Ю.Б., доктор технічних наук, професор кафедри автоматизації та компютерно-інтегрованих технологій Національного університету харчових технологій.

Зіновкін В.В., доктор технічних наук, професор кафедри електроприводу і автоматизації промислових установок Запорізького національного технічного університету.

Ковриго Ю.М., кандидат технічних наук, професор, завідувач кафедри автоматизації теплоенергетичних процесів і виробництв теплоенергетичного факультету Національного технічного університету «Київський політехнічний інститут».

Рекомендовано до друку:
рішенням вченої ради Запорізької державної інженерної академії (протокол №36 від 01.11. 2007р.)

Описано загальні елементи і мови програмування діючого стандарту IEC 61131-3. Викладено Softlogic-система програмування промислових логічних контролерів KW MULTIPROG, а також методика створення і налагодження програмного забезпечення ПЛК у цій інструментальній системі. Наведено приклади програмування різними мовами міжнародного стандарту.

Навчальний посібник призначений для студентів, які здобувають вищу освіту за напрямом «Комп'ютерно-інтегровані системи управління», а також інженерно-технічним працівникам і розробникам АСК ТП.

Рекомендовано Міністерством освіти і науки України як навчальний посібник для студентів вищих навчальних закладів за напрямом «Автоматизація та комп'ютерно-інтегровані технології». Гриф № 1.4/18-Г-330 від 06.02.08р.

ЗМІСТ

ПЕРЕДМОВА

Історія розвитку технологічних мов починається з появи наприкінці 70-х років XX-століття програмованих логічних контролерів (ПЛК).

Перші технологічні контролери були переважно логічними, тому при створенні мови програмування було прийняте рішення - імітувати проектування тих схем, замість яких прийшли контролери. Так з'явилася мова релейно – контактних схем (РКС). Приведення нової суті до старої звичної форми забезпечило психологічно безболісний перехід на нову техніку. РКС – мова мала ще одну перевагу – процедури вводу та налагодження програм легко реалізувалися на примітивних пультах з однорядковим дисплеєм і півтора десятками кнопок.

Успіх програмованих логічних контролерів невдовзі розділили одно-канальні та багатоканальні програмовані цифрові регулятори, які програмувалися мовою функціональних блоків. Ця мова повторювала методику створення систем регулювання на технічній базі, що використовувалася раніше, коли окремі електронні або пневматичні функціональні блоки з'єднували між собою для отримання більш складних функціональних можливостей.

В Україні РКС - мова і мова функціональних блоків були застосовані при програмуванні відповідно МікроДАТів і Реміконтів.

Проте, як показав досвід, РКС – мова виявилася незручною для створення програм керування кроковими процесами, де послідовність кроків порушувалася розгалуженнями, паралелізмом і рециклами. Програми, що створювалися мовою функціональних блоків для багатоканальних схем регулювання, були занадто громіздкі і невиразно зображені на примітивних пультах. Змішування цих мов у деякий мірі розширило можливості програмування, але було безсилим у випадку їх багатофункціонального застосування. Тому почались роботи по вдосконаленню мов програмування, і перш за все, на шляху організації загальної структури програми, яка б дозволяла введення модульності і ієрархічного зображення. Вдалим прикладом цього підходу стала мова Графсет, яку створила французька фірма "Telemechaniq". Основні властивості мови Графсет- це зображення програми у вигляді кроків і переходів, а також наявність декількох одночасно працюючих програм під керівництвом головної. При цьому логіка самих кроків і переходів може бути описана різними мовами, в тому числі РКС і функціональних блоків. Можливість створення крокових послідовностей, наявність в програмі розгалужень, а в деяких версіях і циклів, дозволили користувачам за допомогою мови Графсет разом із структуризацією задач створювати складні програми керування технологічними процесами.

Необхідність поєднання неперервного та дискретного управління на більшості промислових об'єктів спонукала фахівців до створення універсальної мови програмування. Початком на цьому шляху стала спроба адаптації існуючих мов програмування. У перших вільно-програмованих контролерах

застосовувався асемблер. Потім почали використовувати мови високого рівня (Си, Бейсик). Принцип адаптації універсальних мов програмування був простим: вводилися нові типи змінних – “входи” і “виходи” контролера, а також створювалась певна бібліотека підпрограм, що реалізовували найбільш популярні алгоритми. Програмування спростилося, але ще було складним для звичайного технолога або “автоматчика – киповца”. Тому для розв’язання задач автоматизації запрошувалися професійні програмісти. У той же час використання складних мов і потреба великих ресурсів, необхідних для компіляції та рекомпіляції програм, повністю виключило можливість використання пультів програмування.

Стрімкий розвиток обчислювальної техніки і її можливостей показав, що кращим пультом програмування для будь-якого контролера може бути звичайний комп’ютер. Але таки події не наблизили традиційного фахівця з автоматизації до передових технологій в керуванні технологічними процесами, тому що використання комп’ютера знімає будь які обмеження на застосування складних мов програмування, а значить і залишається потреба в професійних програмістах.

І тоді був зроблений наступний крок – створення на базі розповсюджених мов програмування процедурного типу спеціалізованих технологічних мов, орієнтованих саме на кінцевого користувача на об’єкті. Основною різницею текстових технологічних мов від універсальних стало, по-перше, різке спрощення синтаксису та семантики (скорочення типів операторів, складності виразів та ін.), а по-друге, введення в мову спеціальних технологічних понять, які реалізують типові функції контролю й управління. У зв’язку з цим відкрився великий простір для творчості, й кількість мов такого типу на першому етапі майже співпало з кількістю фірм, які бажали їх застосувати.

Щоб уніфікувати технологічні мови, Міжнародний електротехнічний комітет МЕК (IEC) розробив стандарт IEC 61131-3, який визначив п’ять мов програмування – три графічні (LD,FBD,SFC) і дві текстові (ST і IL).

Усі основні виробники інструментальних програмних систем для промислової автоматики прийняли умови нового стандарту. Ядро реального часу (РЧ) таких систем не залежить від марки контролера, а лише змінюються драйвери для доступу до плат ПЗО. Ці пакети програм отримали назву Softlogic.

У списку інструментальних систем програмування, що реалізують стандарт IEC 61131-3, більш ніж два десятки найменувань. Серед них системи CoDeSys (Німеччина), PUMA (Австрія), PDS7 (Нідерланди), SELECONTROL (Швейцарія), ISaGRAF (Франція), KW MULTIPROG (Німеччина)[1-4].

Інструментальна Softlogic-система програмування KW MULTIPROG або просто MULTIPROG на українському ринку з’явилася після того, як фірма Advantech придбала в німецької фірми-розробника KW SoftWare GmbH право на використання цього пакета для програмування контролерів серії ADAM-5510KW (ADAM-5510E/KW, ADAM-5510KW/TCP, ADAM-5510EKW/TCP та ін.) власного виробництва. Проте, така зручна в роботі, дуже виразлива і красива у налагодженні система програмування MULTIPROG поки ще мало

відома користувачам, про що свідчить відсутність будь-якої україномовної або російськомовної літератури, де б викладалася методика роботи в цьому пакеті програмування.

На кафедрі автоматизованого управління технологічними процесами Запорізької державної інженерної академії самостійно опанували цю технологію програмування і вже декілька років студенти проходять лабораторний практикум на базі контролера ADAM-5510KW фірми Advantech, отримують навички програмування і використовують їх у курсовому та дипломному проектуванні. Проте систематизованої методичної розробки, яка б надавала студентам можливість самостійно оволодіти технікою програмування ПЛК у Softlogic – системі MULTIPROG досі немає.

Усе це, а також потреба більш детальніше ознайомити читача з міжнародним стандартом IEC 61131-3, який дає детальний опис загальних елементів програмного проекту і синтаксису мов програмування сучасних мікропроцесорних контролерів, спонукали на створення цього навчального посібника. У посібнику надається значна кількість прикладів програмування різними мовами міжнародного стандарту, контрольні запитання, завдання для самостійної роботи, тому посібник буде корисним усім хто намагається вивчити технологічні мови і придбати навички програмування.

Автор висловлює подяку всім студентам спеціальності «Автоматизоване управління технологічними процесами» Запорізької державної інженерної академії, які приймали участь в опануванні технології програмування промислових логічних контролерів за допомогою Softlogic – системи MULTIPROG і особисто А. Жиманову, А. Янченко, В. Ключкову, О. Теліну, К. Москаленко і К. Помітій, фрагменти з дипломного проекту, курсових і лабораторних робіт яких, використані в даному посібнику.

Вважаю також за необхідне висловити вдячність рецензентам посібника за корисні поради та пропозиції, що зроблені при рецензуванні.

1. МІЖНАРОДНИЙ СТАНДАРТ IEC 61131-3

Існує багато стандартів і тільки невелика їх кількість має дійсно історичне значення. Один з таких стандартів – стандарт МЕК 61131 (IEC 61131), який визначає таке поняття, як “програмовані контролери”. Цей міжнародний стандарт був опублікований у 1993р. і з тих пір широко використовується у всьому світі. Спочатку він мав номер 1131 і тільки з 1997р., коли Міжнародна Електротехнічна Комісія (МЕК) (International Electrotechnical Committee (IEC)) перейшла на 5-цифрове позначення, найменування міжнародної версії стандарту прийняло вигляд IEC 61131.

Процес розробки стандарту для програмованих контролерів почався у 1979 році, коли національні комітети доручили спеціальній робочій групі технічних експертів виконати перший варіант стандарту. Пізніше, коли стало зрозумілим, що створений стандарт дуже складний і не зручний в роботі, було сформовано п’ять проблемно зорієнтованих підрозділів для розробки складових стандарту в наступних п’яти напрямах:

- загальні положення;
- специфікації і випробування обладнання;
- мови програмування;
- рекомендації користувачам;
- специфікації сервісних служб повідомлення.

Найбільш відомою і популярною частиною створеного стандарту стала третя частина - IEC 61131-3, яка визначає мнемоніку мов програмування.

При розробці IEC 61131-3 робоча група знайшла так багато варіацій мов для програмованих контролерів, що було неможливо вибрати одну з існуючих варіацій як спільну мову. Тому вона намагалась розробити нову спільну мову із застосуванням сучасних принципів розробки програмного забезпечення. Але кінцевим результатом цієї роботи стало затвердження у якості стандартної не однієї, а п’яти мов програмування, створених на базі найбільш популярних мов програмування, найбільш поширених у світі контролерів. Це дало змогу зберегти вже напрацьовані програмні рішення і використовувати їх з мінімальними переробками при модернізації систем автоматизації, а також програмувати контролери мовою, яка найбільш зручніша для певних задач автоматизації. Проте, всі мови стандарту мають загальний зовнішній інтерфейс і єдині об’єкти, тому частини прикладної програми можуть бути запрограмовані на будь-якій мові і скомпоновані в єдину програму, що виконується.

Умовно кажучи, стандарт IEC 61131-3 описує, так звані, загальні елементи і мови програмування. Загальні елементи включають такі поняття, як конфігурація, ресурс, задача, тип даних, змінні, стандартні функції і блоки. Вони створюють основу, яка дозволяє об’єднати багатомовні компоненти в одному програмному проекті. Друга частина стандарту присвячена опису синтаксису трьох графічних мов програмування **LD**, **FBD**, **SFC** і двох текстових **– IL, ST**[1-5].

1.1. **Загальні елементи стандарту**

1.1.1. **Архітектура програмного проекту**

Відповідно до міжнародного стандарту програмний проект складається з конфігурацій, які створюють апаратно-програмну платформу. У конфігурації можуть бути визначені один або декілька ресурсів. Ресурс можна порівнювати з процесором, який має пам'ять і систему виконання. Він програмується і у складі конфігурації повністю виконує програмний код в єдиному циклі роботи контролера. Ресурс складається з компонентів - програмних модулів, в оригіналі званих POU(Program Organisation Unit - Одиниця організації програми). POUs – це маленькі і незалежні одиниці програмного забезпечення, які описують дані, що обробляються, і комунікаційні взаємодії, а також містять код програми. Ім’я POU в межах проекту має бути унікальним. Існують три види POUs - програми, функціональні блоки, функції. Кожен POU складається з двох різних частин - розділу описів і розділу тіла коду. У розділі описів задекларовані всі необхідні змінні, а в розділі тіла коду POU записана програма проекту.

POUs ресурсу зв'язуються між собою, утворюючи подібність деревовидної архітектури і можуть бути описані з використанням будь-яких графічних (**SFC**, **FBD**, **LD**) або текстових (**ST**, **IL**) мов.

Планування часу програм і керування їх роботою здійснюють задачі, які можуть бути кількох типів.

Типова задача – це задача з найнижчим пріоритетом (нижче ніж циклічні задачі) і з не плановaним часом. Якщо типова задача виконана і потім немає циклічної задачі або якої-небудь іншої задачі з вищим пріоритетом, типова задача автоматично виконується знову. Це означає, що типова задача завжди виконується, поки немає задачі з вищим пріоритетом.

Циклічні задачі активізуються у певному інтервалі часу і програма виконується періодично.

Системні задачі викликаються операційною системою ПЛК автоматично, коли з’являється помилка або відбувається зміна оперативного стану ПЛК. Системні задачі також відомі як системні програми .

Задачі подій або переривання активізуються, якщо трапилась певна подія.

Кожна задача має певний пріоритет. У системах програмування з пріоритетним плануванням поточна задача з низьким пріоритетом негайно переривається, якщо задача з вищим пріоритетом стає активною завдяки певним подіям. У системах програмування з непріоритетним плануванням, переривання задач задачами з більш високим пріоритетом неможливі.

Типи задач, що підтримуються ПЛК, залежать від самого ПЛК.

1.1.2. **Одиниці організації програми (POUs)**

Функції FU – це POUs з багатьма вхідними параметрами і одним вихідним. Функція не має внутрішньої пам’яті, тому виклик функції з певними

значеннями вхідних параметрів завжди дає незмінний результат. Для опису функцій може бути використана будь-яка мова, за винятком SFC. Інтерфейс функції повинен бути визначений явно з типами і унікальними іменами кожного параметра, що викликається (вхідного) і що повертається (вихідного). Тип функції (тип значення, що повертається) може бути будь-яким. Для того, щоб підтримати стандарт мови ST, параметр, що повертається, повинен мати те ж ім'я, що і функція. Функція викликається за допомогою її ім’я і одночасно вказуються значення вхідних параметрів.

Виконанням функції керує її визивна програма. Виконання визивної програми припиняється до тих пір, поки не закінчить свою роботу функція. В межах функції можна викликати лише іншу функцію. Рекурсивний виклик не дозволений.

В IEC 61131-3 описані стандартні функції, які можуть використовуватися при редагуванні програми ПЛК. Список стандартних функцій містить**:**

1. Функції перетворення типів.
2. Математичні функції.
3. Стандартні арифметичні функції.
4. Логічні функції.
5. Функції виділення і порівняння.
6. Функції рядка символів.
7. Функції типів даних часу.

Функціональні блоки FB – це POUs з багатьма вхідними та вихідними параметрами і оперативною пам’яттю. Значення на виході функціонального блока залежать від вмісту його оперативної пам’яті, тобто функціональний блок, що викликається з тими самими параметрами, має різні вихідні значення. При створенні функціональних блоків можуть використовуватися LD-, FBD-, ST- або IL-мови. Функціональні блоки є тиражованими. Це означає, що локальні змінні функціональних блоків копіюються для кожного її екземпляра. Коли програма викликає функціональний блок, насправді, викликається екземпляр блоку, тобто викликається той же код, але використовуються дані, що призначені спеціально для цього екземпляра. Значення змінних екземпляра зберігаються від одного циклу до іншого. Інтерфейс функціонального блока повинен бути визначений явно з типами і унікальними іменами кожного параметра, що викликається (або вхідного) і що повертається (або вихідного).В межах функціонального блока можна викликати інший функціональний блок або функцію. Рекурсивний виклик неможливий.

Стандарт IEC 61131-3 описує функціональні блоки, які можуть використовуватися при редагуванні програми ПЛК. Список стандартних функціональних блоків містить:

1. Двопозиційні елементи.
2. Функціональні блоки визначення границі.
3. Функціональні блоки лічильників.
4. Функціональні блоки таймерів.

Програми – це POUs, які містять логічну комбінацію функцій і функціональних блоків. Відповідно до програми контролер формує керувальні дії. Програми подібні функціональним блокам. Вони мають вхідні і вихідні параметри, можуть мати оперативну пам'ять і прив'язані до задач програмного проекту. Програми описують послідовні або циклічні операції. Циклічні програми виконуються в кожному циклі цільової системи. Виконання послідовних програм визначається динамічними правилами мови.

В межах програми можна викликати функції і функціональні блоки. Рекурсивний виклик не можливий.

Основні послідовні програми створюються SFC-мовою. Будь-яка SFC-програма може мати одну або більше дочірніх SFC-програм. Циклічні програми, функції і функціональні блоки не можуть бути описані за допомогою SFC-мови.

У той же час функції і функціональні блоки можуть бути викликані з дій або умов SFC-програм.

1.1.3. **Типи даних**

Типи даних характеризують будь-який вираз, змінну або константу, що використовуються в POU (написаному будь-якою мовою). Вони визначають початкове значення змінних, а також діапазон їх зображення і кількість бітів, які змінні цього типу займають у пам'яті.

Типи мають бути узгоджені в графічних зображеннях і текстових виразах програми.

IEC 61131-3 встановлює три види типів даних:

- елементарні типи даних;
- загальні типи даних;
- типи даних користувача.

Елементарні типи даних, які завжди визначаються у проекті, показані у табл.

Таблиця 1.1. Елементарні типи даних

Тип даних	**Опис**	**Розмір у бітах**	**Діапазон припустимих значень**
BOOL	**Логічна величина**	1	0…1
SINT	**Коротке ціле число із знаком**	8	**-128…127**
INT	**Ціле число із знаком**	16	**-32768…32767**
DINT	**Ціле число подвійної точності із знаком**	32	**-2.147.483.648 ¸ 2.147.483.647**
USINT	**Коротке ціле число без знаку**	8	**0 ¸ 255**
UINT	**Ціле число без знаку**	16	**0 ¸ 65535**

Продовження таблиці 1.1

UDINT	**Ціле число подвійної точності без знаку**	32	**0 ¸ 4.294.967.295**
REAL	**Дійсні числа**	32	**.402823466E+38 ¸ -1.175494351E-38 і +1.175494351E-38 ¸ +3.402823466E+38**
TIME	**Часова величина**	32	**0…4.294.967.295ms**
BYTE	**Рядок битів довжиною**	8	**0…255 (16#00…16#FF)**
WORD	**Рядок битів довжиною**	16	**0…65.535(16#00…16#FFFF)**
DWORD	**Рядок битів довжиною**	32	**0…4.294.967.295(16#00…16# FFFFFFFF)**

Використання елементарних типів даних залежить від технічних засобів. Тому необхідно подивитися в документацію технічних засобів, щоб дізнатися, які типи даних підтримуються.

Тип даних STRING (Рядок символів) є також елементарним типом даних, але не належить до наведеної вище групи. Його формат в пам'яті залежить від операційної системи ПЛК.

Загальні типи даних – це типи даних, які мають найменування ANY_... і включають ієрархічні групи елементарних типів даних:

ANY
ANY_NUM
ANY_REAL
REAL
ANY_INT
DINT, INT, SINT
UDINT, UINT, USINT
ANY_BIT
DWORD, WORD, BYTE, BOOL
STRING
TIME

Кожне найменування ANY_... об'єднує певну кількість типів. Так, ANY_INT включає елементарні типи даних DINT, INT, SINT, UDINT, UINT, і USINT. Якщо функція зв'язана з ANY_INT, то це означає, що всі цілочислові змінні типів даних DINT, INT, SINT, UDINT, UINT, і USINT можуть нею оброблятися. Використання ANY_... при оголошенні змінних не можливо.

Використання загальних типів даних залежить також, від технічних засобів. Тому необхідно подивитися документацію технічного засобу, щоб дізнатися, які обмеження встановлені до загальних типів даних.

Тип даних користувача завжди оголошується зі слова TYPE і закінчується рядком END_TYPE. Здійснюється це у робочому листку типу даних, використовуючи текстовий редактор системи програмування. До типу даних користувача відносяться перелічення, структури, масиви і масиви структур.

Тип даних масиву включає декілька елементів однакового типу. Масив оголошується одним рядком, в якому елементи ідентифіковані і доступні за індексом. Приклад оголошення типу даних масиву має наступний вигляд:

```
TYPE
    mass : ARRAY (1…18) OF INT;
END_TYPE
```

У прикладі тип даних “mass” містить 18 елементів типу даних “INT”. Усі елементи масиву запам’ятовуються послідовно у пам’яті ПЛК.

Для того, щоб оголосити масив, необхідно надрукувати його в робочому листку опису типів даних.

Для створення багатовимірних масивів використовуються масиви масивів.

Приклад оголошення типу даних масиву масивів:

```
TYPE
    graph: ARRAY (1…10) OF INT;
    my array : ARRAY (1…3) OF graph;
END_TYPE
```

У масиві масивів елемент стає доступним, коли використовують два індекси. Опис змінної при цьому має вигляд:

```
VAR
   var1 : my_array;
   var2 : INT;
END_VAR
```

Масиви можна ініціалізувати, тобто кожному елементу масиву може бути надане початкове значення. Як згадувалось раніше, для доступу до кожного окремого елементу використовується його індекс. Ініціалізація масиву може бути зроблена при редагуванні декларації тіла коду.

У цьому випадку опис змінної має вигляд:

```
VAR
    graph : ARRAY (1…10) OF INT;
END_VAR
```

А тіло коду ST –мовою представляється як:

```
graph [1] :=7;
graph [2] :=1092;
.
.
.
graph [10] :=13;
```

Ініціалізувати усі елементи масиву немає необхідності. Якщо не задано початкове значення, елемент масиву ініціалізується значенням за умовчанням при запуску програми.

Приклад програмування масиву.

Масив потрібно використовувати для даних, що описують один і той же об’єкт. Уявимо собі, що значення параметру на вході контролера змінюється кожні три секунди. Необхідно запам’ятати кожне значення і порівняти його із заданим. Всі вхідні значення відносяться до одного типу даних. В даному випадку корисно оголосити масив, тому що при створенні тіла коду два

значення можуть бути легко порівняні, завдяки використанню оператора циклу (наприклад FOR). Окремі компоненти масиву можуть бути доступними завдяки індексу масиву.

Оголошення типу має наступний вигляд:

```
TYPE
      graph : ARRAY (1…23) OF INT;
      set_point : ARRAY (1…23) OF INT;
END_TYPE
```

А опис змінної зображується як:

```
VAR
      input    : graph;          (* значення, що поступають *)
      values   : set_point;      (*значення для порівняння з*)
       i         : INT :=1;        (*змінний індекс масиву*)
       run      : BOOL :=TRUE;
ERROR       : BOOL
timer          : FB_TIMER    (*оголошення зразка FB*)
END_VAR
```

Тоді тіло коду ST -мовою:

```
timer (pt :=t#3s ; in :=run);
IF timer. Q THEN   (*забезпечення вхідних значень, щоб збудувати
                          масив “graph”*)
       input[i] :=% IWO;    (*присвоювання вхідного значення масиву*)
        run :=0              (*визначення початкового значення для
                               повторного запуску таймера*)
          i := i+1           (*наступний індекс масиву*)
ELSE
       run := 1              (*підрахунок*)
END_IF;
IF i =23
    FOR i :=1 TO 23 BY 1 DO
          IF input [i] < > values [i] THEN   (*порівняння масиву
                                              “graph” до “set point”*)
          ERROR := TRUE;
          END_IF;
     END_FOR;
i :=1;
END_IF
```

Структурований тип даних включає декілька елементів одного або різних типів даних. На відміну від масивів структура дійсно створює новий тип даних. До використання конкретної змінної потрібно здійснити два оголошення. Спочатку необхідно описати структуру на рівні проекту. І тільки після того, як структура отримує ідентифікатор (ім’я) типу даних, оголошуються змінні.

Оголошення структури починається зі слову **STRUCT** і закінчується **END_STRUCT.**

Наприклад:

```
TYPE
      machine     :
      STRUCT
```

```
        x_pos    : INT;
        y_pos    : INT;
        depth    : INT;
        rpm      : UINT;
    END_STRUCT;
END_TYPE
```

У прикладі структурований тип даних “machine” складається з компонентів x_pos, y_pos, depth і rpm. Всі компоненти описують характеристики машини.

Структури використовуються, коли потрібно декларувати дані, що описують однакові об’єкти. Наприклад, необхідно просвердлити декілька отворів на одному робочому місці. Всі отвори мають положення в Х*У просторі, різну глибину і інтервал свердління отворів. Усі отвори різні, але змінні, що описують їх однакові. У цьому випадку корисно оголосити структуру, яка складається з трьох компонентів – положення свердла, глибини і інтервалу свердління. Для кожного отвору ці компоненти можуть бути різних значень. Функціональний блок процесу свердління працює з однією і тією ж змінною - machine, що була структурована.

При програмуванні використаємо структуру як елемент масиву:

```
TYPE
    machine     :
    STRUCT
        x_pos    : REAL;
        y_pos    : REAL;
        depth    : INT;
        rpm      : INT;
    END_STRUCT;
     my array    : ARRAY [1…10]  OF machin;
END_TYPE
```

Наведений приклад масиву структур забезпечує послідовність свердління декількох отворів. В масиві індексується конкретний станок, який буде працювати, а через структуру компонентів призначаються різні значення для свердління.

Можна використовувати масиви в структурах, як це показано в наступному прикладі:

```
TYPE
   graph  : ARRAY [1…10]  of INT;
   drive  :
   STRUCT
          rpm        : INT;
          inputs     : IN_BOOL;
       performance : graph;
   END_STRUCT;
END_TYPE
```

Структури ініціалізуються при редагуванні тіла коду проекту програми, коли компонентам призначаються значення . Приклад декларування змінної має наступний вигляд:

```
VAR
      var1   :  machine;
      first    :  BOOL  := TRUE;
END_VAR
```
Тоді тіло коду ST- мовою :
```
IF first THEN
         var1   :  x_pos  := REAL#1.3E+2;
          var1   :  rpm    := 3000;
        ……..
          first    := FALSE;
END_IF
```
Рядковий тип даних – це ряд з певною кількістю символів. При оголошені певного ряду довжина його вказується в круглих дужках після типу даних.

Прикладом оголошення рядкового типу даних є:
```
TYPE
      STRING 10    : STRING (10)
END_TYPE
```
У цьому прикладі довжина рядка складає 10 символів. Найкоротший рядок має довжину 1, найдовший рядок має довжину 32766.

Перелічний тип даних обмежує можливі значення змінної значеннями, вказаними при визначенні цього типу. Змінна, що використовується у цьому типі даних, може мати тільки перелічені значення. Використання і формат (пам'ять) структурованого типу даних, визначеного користувачем рядка і перелічного типу даних, залежать від типу ПЛК. Тому в документації контролера має бути все те, що стосується використання цих типів даних.

Прикладом перелічного типу даних є:
```
TYPE
      light      :  {red, yellow, green};
END_TYPE
```
У прикладі тип даних "light" може мати тільки red, yellow або green значення.

1.1.4. Константи

Константи використовуються для зображення чисел, характеристики рядків і часових даних. Константні вирази можуть відноситися тільки до одного типу. Один і той же запис не може бути використаний для зображення константних виразів різних типів.

Числові константі вирази можуть бути цілочисловими, дійсними і логічними.

Цілочислові константи можуть бути зображені в десятковому, шістнадцятковому, вісімковому, двійковому численні. При цьому, окрім десяткових, всі значення констант починаються з префікса, який ідентифікує основу системи числення. Щоб розділити групи цифр, використовується символ

підкреслення «_». Він не має особливого значення і використовується для поліпшення читаності констант.

Наприклад:

-Константа цілого числа - 120123_456+986

-Двійкова константа - INT#2#1111_1111

-Вісімкова константа - INT#8#377

-Шістнадцяткова константа - INT#16#FF SINT#16#FF

Дійсні вирази констант можуть бути записані в десятковому або інженерному уявленні**.** Десяткова крапка «.» розділяє цілу і десяткову частини. Десяткову крапку потрібно використовувати для того, щоб відрізнити дійсну константу від цілочислової. Інженерне уявлення використовує букви 'E' або 'F' для того, щоб відділити мантису від експоненти. Експоненціальна частина в інженерному уявленні повинна бути знаковою цілою величиною в діапазоні від -37 до +37.

Наприклад:

-Дійсна константа -12.00.00.45603.14159_26

-Дійсна константа з показником ступіню -1.34E-12-1.34e-121.0E+6

Логічні константи існують тільки двох типів:

- TRUE - еквівалентна цілочисловій величині із значенням 1;
- FALSE - еквівалентна цілочисловій величині із значенням 0.

Константи, що використовуються у робочих листках змінних, константи типу даних INT або BOOL можуть використовуватися без ключового слова, як це показано у наступних прикладах:

для INT#16#FF можна використовувати 16#FF;

для BOOL#FALSE можна використовувати FALSE.

При опису змінної можна використовувати “var1: DINT:=10”, але в тілі коду доведеться використовувати “LD DINT#10”.

Константи часу можуть бути зображені в годинах (h), хвилинах (m), секундах (s), мілісекундах (ms) і в їх комбінації.

Вони мають починатися з префікса "T#" або "TIME#"**.**

Наприклад:

Короткий префікс
T#14mst#14mst#12m18s3.5msT#25h_15mt#25h15m;

Довгий префікс
TIME#14mstime#14msTIME#25h_15mtime#25h_15.

Константи дати і часу дня починаються префіксами відповідно "D#" або "DATE#" і "TOD#" або "TIME_OF_DAY#".

Наприклад:

-Дата - DATE#1996-01-24 - Час - TIME_OF_DAY#15:36:55:36

date#1996-01-24 time_of_day#15:36:55:36

D#1996-01-24 TOD#15:36:55:36

d#1996-01-24 tod#15:36:55:36

-Дата і час - DATE_AND_TIME#1996-01-24-15:36:55:36

date_and_time#1996-01-24-15:36:55:36

DT#1996-01-24-15:36:55:36

Константи рядка символів – це послідовність символів, яка обмежена двома одинарними лапками.

Попередження: символ лапки «’» не може бути використаний всередині рядкового константного виразу. Рядковий константний вираз має бути записаний в одному рядку початкового тексту програми. Довжина рядка не повинна перевищувати 255 символів, включаючи пропуски.

Порожній рядковий константний вираз зображається двома лапками без пропуску або табуляції всередині.

Наприклад:

-Порожній рядок ,,

-Рядок з вільним місцем , ,

-Непорожній рядок ’this is a text’

1.1.5. **Змінні**

Змінні використовуються при програмуванні замість входів прямої адресації, виходів і прапорів. Кожна змінна обов’язково має ім’я і тип. Імена (ідентифікатори) змінних повинні задовольняти наступним правилам:

- ім'я не може мати більше 128 символів;
- першим символом повинна бути літера;
- подальшими символами можуть бути літери, цифри або символ підкреслення.

В IEC-61131-3 описані три типи змінних:

- символьні змінні;
- безпосередньо зображені змінні;
- локалізовані змінні.

Змінні потрібно декларувати в робочому листку змінних POU, використовуючи ключові слова. Компетенція кожної змінної, яка визначена з використанням ключового слова, обмежується POU або цілим проектом. Тому змінні поділяють на локальні та глобальні.

Якщо змінна використовується тільки в межах POU, вона називається локальною змінною. У цьому випадку мають використовуватися ключові слова змінної VAR, VAR_INPUT, VAR_OTPUT.

Якщо змінна використовується в межах всього проекту, вона називається глобальною змінною і декларується як VAR_GLOBAL у глобальних деклараціях і як VAR_EXTERNAL у кожному POU, де вона використовується.

Повний перелік ключових слів надається у табл 1.2.

Таблиця 1.2. Ключові слова

Ключові слова	Опис
VAR	**-для внутрішніх змінних, які можуть використовуватися в межах POU;** **-для оголошення екземплярів функціональних блоків;** **-для декларації безпосередньо представлених і символьних змінних;**

Продовження таблиці 1.2

	-може використовуватися з ключовим словом RETAIN для оголошення затримуючих змінних;
VAR_INPUT	**-для змінних, які є входами до функції, функціональних блоків і програм;** **-надається змінній POU, що прибуває від іншого POU;** **-для декларації тільки символьних змінних в межах POU;**
VAR_OUTPUT	**-для змінних, які є виходами функціональних блоків і програм;** **-для вихідних змінних,які призначені, наприлад, для іншогоPOU;** **-для змінних, що використовуються тільки в межах POU;** **-може використовуватися з ключовим словом RETAIN;**
VAR_IN_OUT	**- для оголошення затримуючих змінних;** **-адреса змінної передається довідкою;** **-змінна може бути зчитана або записана;** **- для складних типів даних, таких якрядки, масиви і структури;**
VAR_EXTERNAL	**-для глобальних змінних в POU;** **-така змінна потрапляє в декларацією VAR_GLOBAL;** **-така змінна може бути змінена в межах POU;** **-може використовуватися тільки для декларації символьних змінних;**
VAR_EXTERNAL_PG	**-для глобальних змінних в програмі;** **-така змінна потрапляє в декларацією VAR_GLOBAL_PG;** **-не можуть бути ініціалізовані;** **-така змінна може бути змінена в межах програми;** **-може використовуватися тільки для декларації символьних змінних;**
VAR_EXTERNAL_FB	**-для глобальних змінних у функціональному блоці;** **-така змінна потрапляє в декларацією VAR_GLOBAL_FB;** **-не може ініціалізуватися;** **-така змінна може змінюватися в межах функціонального блоку;** **-може використовуватися тільки для декларації символьних змінних;**
VAR_GLOBAL	**-для глобальних змінних, які можуть використовува-**

Продовження таблиці 1.2

	тися в усіх програмах і функціональних блоках проекту; **-можуть використовуватися для декларації безпосередньо представлених і символьних змінних;** **-можуть використовуватися з ключовим словом RETAIN для декларації затримуючих змінних;**
VAR_GLOBAL_ PG	**-для глобальних змінних, які можуть використовуватися в усіх програмах проекту;** **-можуть використовуватися для декларації безпосередньо представлених і символьних змінних;** **-можуть використовуватися з ключовим словом RETAIN для декларації затримуючих змінних;**
VAR_GLOBAL_FB	**-для глобальних змінних, які використовуються в усіх функціональних блоках проекту;** **-можуть використовуватися для декларації безпосередньо представлених і символьних змінних;** **-можуть використовуватися з ключовим словом RETAIN для декларації затримуючих змінних;**
END_VAR	**-кінцівка декларації блока;**

На додаток до цих ключових слів, ще два ключових слова використовують при декларуванні змінних: RETAIN для затримуючих змінних і ключове слово AT, яке потрібно для декларування безпосередньо зображених змінних.

Ключові слова VAR_GLOBAL_PG, VAR_GLOBAL_FB, VAR_EXTERNAL_PG, VAR_EXTERNAL_FB є розширенням стандарту IEC.

Символьні змінні декларуються з символьним ім'ям і типом даних. Початкове значення необов'язкове. Система програмування запам'ятовує змінну у вільній області пам'яті ОЗП ПЛК, яка невідома користувачеві.

Наступний приклад показує декларацію двох символьних змінних:

```
VAR
    var1:  BOOL;
    var2:  INT (-22…12);
END_VAR
```

Символьні змінні можуть бути ініціалізовані та/або декларовані як затримуючі змінні, які використовують ключове слово RETAIN.

Безпосередньо зображені змінні декларуються без символьного імені, але з використанням логічної адреси.

Локалізовані змінні декларуються з символьним ім'ям і логічною адресою.

Як безпосередньо зображені, так і локалізовані змінні запам'ятовуються у декларованій логічній адресі. Це створює програмісту можливість перевірки того, що ніяка адреса пам'яті не використовується двічі. Декларація (опис) розташування складається з ключового слова AT, процентного знака %,

приставки (префікса) розташування, розміру приставки (префікса) та ім’я логічної адреси.

Безпосередньо зображені й локалізовані змінні можуть бути декларовані в робочому листку глобальних змінних програми, використовуючи VAR_GLOBAL.

У табл. 1.3 показані префікси розташування та розміру безпосередньо зображених та локалізованих змінних.

Таблиця 1.3. Префікси розташування та розміру змінних

Префікс розташування	**Опис**
I	**Фізичний вхід**
Q	**Фізичний вихід**
Префікс розміру	**Опис**
X	**Однобітний розмір (тільки з типом даних BOOL)**
None	**Однобітний розмір**
B	**Розмір байта (8 біт)**
W	**Розмір слова (16 біт)**
D	**Розмір подвійного слова (32 біти)**
L	**Розмір довгого слова (64 біта)**

Для створення безпосередньо зображених та локалізованих змінних використовується така форма оголошення:

Ім’я змінної AT% пряма адреса : тип

Пряма адреса починається з літери, яка визначає область пам’яті, далі записується символ, що визначає пряму адресу. Завершується пряма адреса числом. Якщо пряма адреса визначає байт, то завершується вона номером байта, якщо пряма адреса визначає слово то завершується номером слова.

У стандарті передбачена зручна форма роботи з окремими бітами, що визначають локалізовані змінні. Необхідний біт вказується через крапку після ідентифікатора. У найпростішому варіанті це номер слота і номер біта. Коли вказується адреса одного біта, тип змінної може бути тільки **BOOL.**

У наступному прикладі показана декларація безпосередньо зображених і локалізованих змінних:

```
VAR
    var1   AT%QX2.4   :BOOL;
           AT%IW4     :WORD;
           AT%QB7     :BYTE;
END_VAR
```

Затримуючі змінні – це змінні, значення яких зберігається навіть якщо вимкнене живлення. У випадку теплого запуску останнє значення змінної готове до використання.

Затримуючі змінні декларуються, використовуючи ключове слово RETAIN, як це показано у наступному прикладі:

```
VAR RETAIN
    var1    : BOOL := TRUE;
END_VAR
```

У цьому прикладі змінна набула початкове значення TRUE, яке є початковим значенням для холодного запуску.

У разі теплого запуску використовується поточне значення змінної.

Ключове слово RETAIN може використовуватися в комбінації з ключовими словами VAR, VAR_OTPUT і VAR_GLOBAL. Неможливо декларувати затримуючі змінні з ключовими словами VAR_INPUT і VAR_EXTERNAL.

Згідно до стандарту IEC 61131-3 змінним можуть бути призначені початкові значення. Це означає, що змінна, яка використовується в програмі ПЛК вперше, має початкове значення. Початкові значення можуть бути надані усім видам змінних, окрім в деклараціях VAR_EXTERNAL.

Початкові значення мають бути вставлені в кінці рядка декларованої змінної, використовуючи " := ", як це показано в наступному прикладі:

```
VAR
    var1 :    INT := 28;
    var2 :    TIME := T#1s;
    var3 :    AT%QX0.0 := TRUE;
END_VAR
```

Не можна ініціалізувати змінні, які розміщені у фізичних входах.

Початкове значення має відповідний тип даних. Не можна використовувати, наприклад, тип даної BOOL і початкове значення "5". Початкове значення необов'язкове. Якщо ніякого значення немає, змінна ініціалізується з невстановленим початковим значенням типу даних або із значенням, що зберігається у разі затримуючих змінних[6].

Контрольні запитання

1. Чому призначений стандарт МЕК 61131 (IEC 61131)?
2. З чого складається програмний проект стандарту?
3. Що таке одиниця організації програми?
4. Чим відрізняються функції від функціональних блоків?
5. Які бувають типи даних і, як вони оголошуються?
6. Коли використовують константи і які вони бувають?
7. Які типи змінних описані в стандарті IEC-61131-3?
10. Як декларуються символьні змінні?
11. Що таке softlogic-система програмування?

1.2. **Мови програмування**

Стандарт IEC 61131-3 визначає синтаксис п’яти мов програмування і декларує різні елементи, які можуть використовуватися мовами.

За зовнішньою ознакою мови програмування можуть бути диференційовані на дві текстові - Список Інструкцій (IL) та Структурованого Тексту (ST) і три графічні - Крокових діаграм (LD), Функціональних блокових діаграм (FBD) і Послідовних функціональних схем (SFC).

Кожна з цих мов має свої особливості і найбільш пристосована для розв'язання тих або інших алгоритмів керування. Наприклад, мова крокових діаграм, безперечно, має переваги при реалізації алгоритму керування, який раніше був реалізований на релейно-контактних схемах. Мова функціональних блокових діаграм найбільш зрозуміла для фахівців-практиків, які мають досвід створення систем автоматизації з окремих функціональних блоків: автоматичних регуляторів, задавачів, таймерів, лічильників, блоків арифметичних операцій і т. ін. Якщо у програмі користувача необхідно виконувати велику кількість операцій по обробці інформації, краще використовувати мову структурованого тексту - ST. IL-мова дозволяє створювати високоефективні та оптимізовані функції, тому її використовують для написання найбільш критичних місць програми. Для реалізації алгоритму керування послідовно-паралельними процесами пристосована мова послідовних функціональних схем.

Якщо у програмі користувача зустрічаються різні за алгоритмами задачі, то доцільно для їх написання використовувати різні технологічні мови. А вже при запису програми користувача у ПЛК різні фрагменти програми поєднуються у один машинний код. Проте досвід показує, що вибір мов програмування в першу чергу визначається особистою схильністю користувачів і мало залежить від об’єкту автоматизації. Мови міжнародного стандарту в більшості випадків взаємозамінні, а це означає, що при різному рівні підготовленості в області чистого програмування користувачі можуть створювати програми рівної функціональності.

1.2.1. **Стандартні мовні елементи**

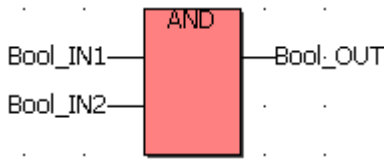
До стандартних мовних елементів відносяться оператори, функції та функціональні блоки. Правильне їх використання при створені програм не можливе без розуміння призначення цих елементів та порядку їх функціонування. Тому розглянемо стандартні елементи, які частіше за усе використовуються при програмуванні контролерів[4,7].

1.2.1.1. **Оператори**

Майже усі стандартні оператори у виразах текстових мов мають символьну форму запису. Для зручності використання в графічних мовах оператори подаються у вигляді функцій.

Булеві оператори

Оператор AND (Логічне І)



Для цього оператора кількість булевих входів може бути більше двох.

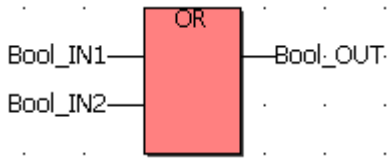
(*STеквівалент:*)

```
Bool_OUT:= Bool_IN1AND Bool_IN2;
```

(* IL еквівалент: *)

```
LD    Bool_IN1
AND Bool_IN2
ST    Bool_OUT
```

Оператор OR (АБО)



Для цього оператора кількість булевих входів може бути більше двох.

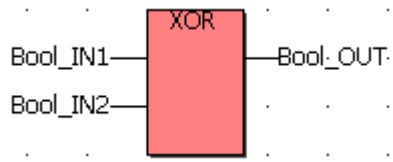
(*STеквівалент:*)

```
Bool_OUT:= Bool_IN1ORBool_IN2;
```

(* IL еквівалент: *)

```
LD Bool_IN1
OR Bool_IN2
ST Bool_OUT
```

Оператор XOR (Виключне АБО)



Для цього оператора кількість булевих входів може бути більше двох.

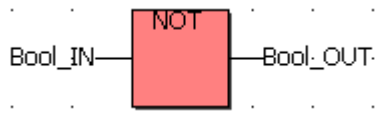
(*STеквівалент:*)

```
Bool_OUT:= Bool_IN1XORBool_IN2;
```

(* IL еквівалент: *)

```
LD    Bool_IN1
XOR Bool_IN2
ST    Bool_OUT
```

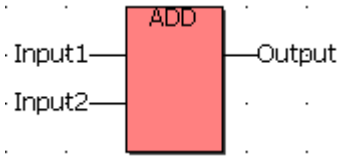
Оператор NOT (Логічне заперечення НІ)



Повертає заперечення повного булевого виразу.
Bool_IN – будь-яка булева змінна або складний вираз;
BooL_OUT-TRUE, коли BooL_IN-FALSE і FALSE,
коли BooL_IN-TRUE.

Арифметичні оператори

Оператор ADD (Додавання)



Складає дві або декілька змінних.

Входи – DINT-REAL-STRING-TIME можуть бути цілими, дійсними, рядковими або часовими (усі входи мають бути одного формату).

Виходи - DINT-REAL-STRING-TIME складання значень входів із знаком.

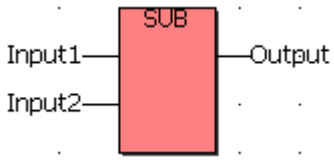
(*STеквівалент:*)

```
Output:= Input1 + Input2;
```

(* IL еквівалент: *)

```
LD   Input1
ADD Input2
ST    Output
```

Оператор SUB (Віднімання)



Віднімає дві змінні (другу з першої).

Входи: Input1 – DINT-REAL-TIME може бути цілим, дійсним, або часовим;

Input2- DINT-REAL-TIME (всі входи мають бути одного формату).

Вихід: Output - DINT-REAL-TIME віднімання.

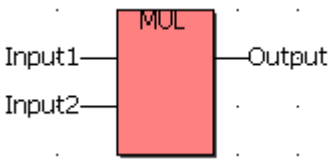
(*STеквівалент:*)

```
Output:= Input1 - Input2;
```

(* IL еквівалент: *)

```
LD   Input1
SUB Input2
ST    Output
```

Оператор MUL (Множення)



Для цього оператора кількість входів може бути більше двох.

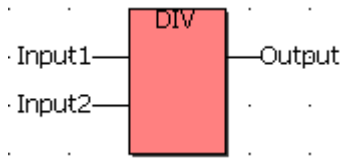
Входи DINT-REAL можуть бути цілими або дійсними (усі входи мають бути одного формату).

Вихід DINT-REAL множення значень входів із знаком.

Опис: Помножує дві або декілька цілих, або дійсних змінних.

(*ST еквівалент*)	(*IL еквівалент*)
*Output:= Input1*Input2*	*LD Input1* *MUL Input2* *ST Output*

Оператор DIV (Ділення)



Input1 - DINT-REAL може бути цілим або дійсним (операнд);

Input2 - DINT-REAL ненульове ціле або дійсне значення (дільник);

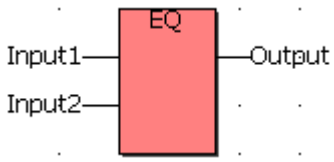
Output - DINT-REAL знакове ціле або дійсне ділення Input1 на Input2.

Опис: Ділить дві змінні (першу на другу).

(*ST еквівалент*)	(*IL еквівалент*)
Output:= Input1/Input2	*LD Input1* *DIV Input2* *ST Output*

Оператори порівняння

Оператор «=» (Дорівнює)



Input1 - BOOL-SINT-DINT-TIME-STRING;

Input2 - BOOL-SINT-DINT-TIME-STRING;

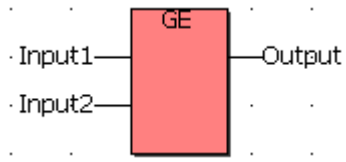
Output - BOOL TRUE, якщо Input1= Input2,
FALS, якщо Input1≠ Input2.

Вхід і вихід мають бути одного типу.

Опис: Перевірити, що одна величина дорівнює іншій. Не рекомендується перевіряти на рівність у SFC-діаграмі виходи часових блоків типу TON,TP,TOF,BLINK .

(*ST еквівалент*)	(*IL еквівалент*)
(Output=TRUE):=((Input1=5)=(Input2=5))	*LD Input1* *EQ Input2*
(Output=FALS):= ((IN1=2)≠(IN2=6))	*ST Output(*FALS або TRUE*)*

Оператор « >=» (Більше або дорівнює)



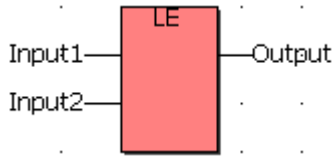
Input1 - DINT-REAL-TIME-STRING;
Input2 - DINT-REAL-TIME-STRING;
Output - BOOL TRUE, якщо IN1>=IN2.
Вхід і вихід мають бути одного типу.

Опис: Перевірити, що одна величина більше або дорівнює іншій. Не рекомендується перевіряти виходи часових блоків типу TON, TP, TOF, BLINK у SFC-діаграмі.

(*ST еквівалент*) (*IL еквівалент*)

(Output=TRUE):=((Input1=10)>=(Input2=10)) *LD Input1*
(Output=FALS):=((Input1=23)>=(Input2=28)) *GE Input2*
*S T Output(*FALS або TRUE*)*

Оператор «<=» (Менше або дорівнює)



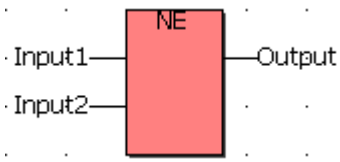
Input1 - DINT-REAL-TIME-STRING;
Input2 - DINT-REAL-TIME-STRING;
Output - BOOL TRUE, якщо Input1<=Input2.
Вхід і вихід мають бути одного типу.

Опис: Перевірити, що одна величина менше або дорівнює іншій. Не рекомендується перевіряти виходи часових блоків типу TON, TP, TOF, BLINK у SFC-діаграмі.

(*ST еквівалент*) (*IL еквівалент*)

(Output=TRUE):=((Input1=10)<=(Input2=19)) *LD Input1*
(Output=FALS):=((Input1=23)<=(Input2=18)) *LE Input2*
*ST Output(*FALSабо TRUE*)*

Оператор «<>» (Не дорівнює)



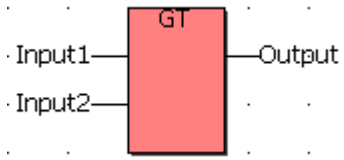
Input1 - BOOL-SINT-DINT-REAL-STRING;
Input2 - BOOL-SINT-DINT-REAL-STRING;

Output - BOOL TRUE, якщо перший <> другому.
Обидва входи мають бути одного формату.
Опис: Перевірити, що одна величина не дорівнює іншій.

(*ST еквівалент*) (*IL еквівалент*)

(Output=TRUE):=((Input1=10)<>(Input2=19)) *LD Input1*
(Output=FALS):=((Input1=23)<>(Input2=23)) *NE Input2*
*STOutput(*FALSабо TRUE*)*

Оператор «>» (Більше ніж)

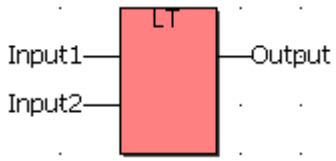


Input1 - DINT-REAL-TIME-STRING;
Input2 - DINT-REAL-TIME-STRING;
Output - BOOL TRUE, якщо Input1>Input2.
Вхід і вихід мають бути одного типу.
Опис: Перевірити, що одна величина більше за іншу

(*ST еквівалент*) (*IL еквівалент*)

(Output=TRUE):=((Input1=10)>(Input2=4)) *LD Input1*
(Output=FALS):=((Input1=23)>(Input2=28)) *GT Input2*
*ST Output(*FALS або TRUE*)*

Оператор « <» (Менше ніж)



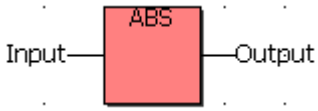
Input1 - DINT-REAL-TIME-STRING;
Input2 - DINT-REAL-TIME-STRING;
Output - BOOL TRUE, якщо Input1< Input2.
Вхід і вихід мають бути одного типу.
Опис: Перевірити, що одна величина менше за іншу.

(*ST еквівалент*) (*IL еквівалент*)

(Output=TRUE):=((Input1=10)<(Input2=14)) *LD Input1*
(Output=FALS):=((Input1=13)<(Input2=8)) *LT Input2*
*ST Output(*FALS або TRUE*)*

1.2.1.2. Стандартні функції
Математичні функції

Функція ABS

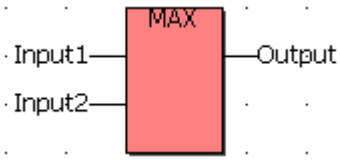


Input REAL будь-яка знакова аналогова величина.
Output REAL абсолютне значення (завжди позитивне).
Опис: Дає абсолютне значення (позитивне) дійсної величини.

(*ST еквівалент*)
Output:=ABS(Input)

(*IL еквівалент*)
LD Input
ABS
ST Output

Функція MAX

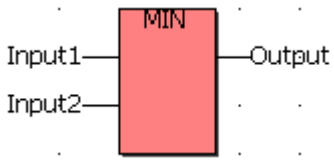


Input1 DINT будь-яка знакова ціла величина.
Input2 DINT (не може бути REAL)
Output DINT максимум з двох входів
Опис: Дає максимальне значення з двох цілих.

(*ST еквівалент*)
Output:=MAX(Input1,Input2)

(*IL еквівалент*)
LD Input1
MAX Input2
ST Output

Функція MIN

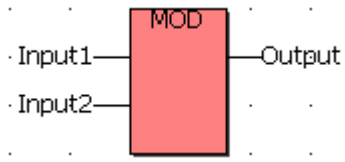


Input1 DINT будь-яка знакова ціла величина.
Input2 DINT (не може бути REAL)
Output DINT мінімум з двох входів
Опис: Дає мінімальне значення з двох цілих.

(*ST еквівалент*)
Output:=MIN(Input1,Input2)

(*IL еквівалент*)
LD Input1
MIN Input2
ST Output

Функція MOD

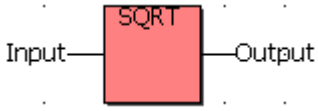


Input1 DINT будь-яка знакова ціла величина.
Input2 DINT має бути більше нуля.
Output DINT обчислення модуля
Опис: Обчислює модуль цілого значення.

(*ST еквівалент*) (*IL еквівалент*)

```
Output:=MOD(Input1,Input2)        LD    Input1
                                  MOD   Input2
                                  ST    Output
```

Функція SQRT

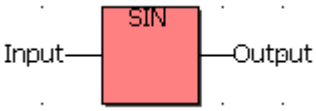


Input REAL має бути більше або дорівнювати нулю.
Output REAL квадратний корінь вхідної величини.
Опис: Обчислює квадратний корінь дійсного значення.

(*ST еквівалент*) (*IL еквівалент*)

```
Output:=SQRT(Input)        LD    Input
                           SQRT
                           ST    Output
```

Функція SIN

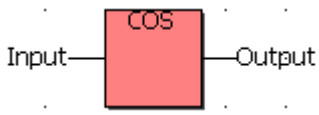


Input REAL будь-яка дійсна величина.
Output REAL синус входу (у діапазоні -1.0…+1.0.
Опис: Обчислює синус дійсної величини.

(*ST еквівалент*) (*IL еквівалент*)

```
Output:=SIN(Input)        LD    Input
                          SIN
                          ST    Output
```

Функція COS

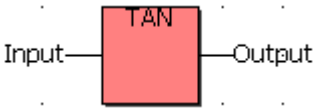


Input REAL будь-яка дійсна величина.
Output REAL косинус входу (у діапазоні -1.0…+1.0.
Опис: Обчислює косинус дійсної величини.

(*ST еквівалент*) (*IL еквівалент*)

```
Output:=COS(Input)        LD   Input
                          COS
                          ST   Output
```

Функція TAN

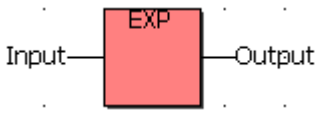


Input REAL не може дорівнювати π/2 за модулем π.
Output REAL тангенс вхідного значення =1E+38 для неправильного входу.
Опис: Обчислює тангенс дійсної величини.

(*ST еквівалент*) (*IL еквівалент*)

```
Output:=TAN(Input)        LD   Input
                          TAN
                          ST   Output
```

Функція EXP



Input REAL
Output REAL
Опис: Обчислює експоненту вхідної величини.

(*ST еквівалент*) (*IL еквівалент*)

```
Output:=EXP(Input)        LD   Input
                          EXP
                          ST   Output
```

1.2.1.3. Стандартні функціональні блоки

Детектори імпульсів

Детектори імпульсів застосовуються, коли має бути реакція не на стан дискретного сигналу, а на його зміну.

*Детектор переднього фронту **R_TRIG***

CLK	BOOL	будь-яка булева змінна.
Q	BOOL	TRUE, коли CLK змінюється з FALS на TRUE. У решті випадків – FALS.

Опис: Генерує одиничний імпульс по передньому фронту вхідного сигналу, тобто вихід Q установлюється в TRUE, коли у попередньому циклі вхід CLK дорівнював FALS, а у поточному циклі він вже має значення TRUE.

*Детектор заднього фронту **F_TRIG***

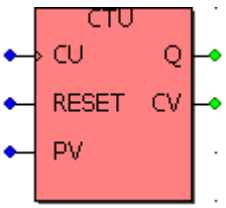
CLK	BOOL	будь-яка булева змінна.
Q	BOOL	TRUE, коли CLK змінюється з TRUE на FALS. У решті випадків – FALS.

Опис: Генерує одиничний імпульс по задньому фронту вхідного сигналу, тобто вихід Q установлюється в TRUE, коли у попередньому циклі вхід CLK дорівнював TRUE, а у поточному циклі він вже має значення FALS.

Попередження: Блок F_TRIG має властивість формувати хибний імпульс при перезапуску.

Лічильники

*Інкрементний лічильник **CTU***

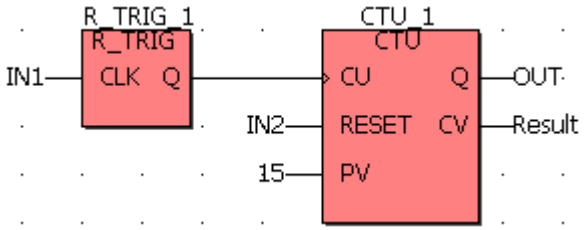


CU	BOOL	– вхід для лічіння (лічити, коли CU дорівнює TRUE).
RESET	BOOL	– команда скидання (домінанта).
PV	DINT	- програмоване максимальне значення.
Q	BOOL	- переповнення: TRUE, коли CV>=PV.
CV	DINT	- результат лічіння.

Опис: По кожному фронту на CU-вході значення лічильника на CV-виході збільшується на 1. Вихід Q установлюється у TRUE, коли лічильник досягає або перевищує задане значення на PV-вході. Логічна одиниця на RESET-вході зупиняє лічильник та скидає його показання (CV:=0).

Попередження: блок CTU не визначає передній або задній фронт входу CU. Для того, щоб створити імпульсний лічильник, його потрібно сполучити з блоками R_TRIG або F_TRIG.

Приклад:



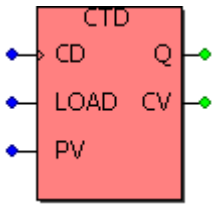
(*ST еквівалент: R_TRIG_1- це екземпляр блока R_TRIG, а CTU_1- це екземпляр блоку CTU*)

CTU_1(R_TRIG_1(IN1),IN2,15);
OUT:=CTU_1.Q;
Result:=CTU_1.CV;

(*IL еквівалент*)

LD	*IN1*	*LD*	*15*
ST	*R_TRIG_1.CLR*	*ST*	*CTU_1.PV*
CAL	*R_TRIG_1*	*CAL*	*CTU_1*
LD	*R_TRIG.Q*	*LD*	*CTU_1.Q*
ST	*CTU_1.CU*	*ST*	*OUT*
LD	*IN2*	*LD*	*CTU_1CV*
ST	*CTU_1.Reset*	*ST*	*Result*

Декрементний лічильник CTD



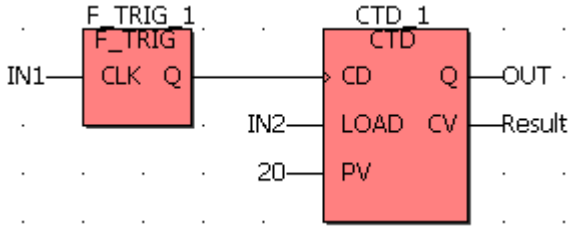
CD BOOL – вхід для лічіння (лічити, коли CD дорівнює TRUE).
LOAD BOOL – команда завантаження (домінанта) .
PV DINT - програмоване максимальне значення.
Q BOOL - переповнення: TRUE, коли CV=0.
CV DINT - результат лічіння.

Опис: По кожному фронту на CD-вході значення лічильника на CV-виході зменшується на 1. Вихід Q установлюється у TRUE, коли лічильник досягає нуля. CV-лічильник завантажується початковим значенням, яке дорівнює PV, коли вхід LOAD= TRUE.

Попередження: блок CTD не визначає передній і задній фронти входу CD. Для того, щоб створити імпульсний лічильник, його потрібно сполучити з блоками R_TRIG або F_TRIG.

Приклад:

(*Програма FBD, що використовує блок CTD*)



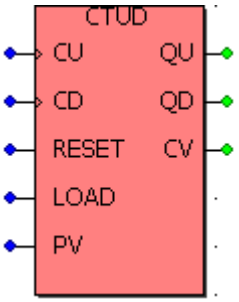
(* ST еквівалент: F_TRIG_1- це екземпляр блока F_TRIG, а CTD_1- це екземпляр блоку CTD*)

CTD(F_TRIG(IN1),IN2,20);
OUT:=CTD_1.Q;
Result:=CTD_1.CV

(*IL еквівалент*)

LD	*IN1*	*LD*	*20*
ST	*F_TRIG_1.CLR*	*ST*	*CTD_1.PV*
CAL	*F_TRIG_1*	*CAL*	*CTD_1*
LD	*F_TRIG.Q*	*LD*	*CTD_1.Q*
ST	*CTD_1.CD*	*ST*	*OUT*
LD	*IN2*	*LD*	*CTD_1CV*
ST	*CTD_1.LOAD*	*ST*	*Result*

Інкрементний-декрементний лічильник CTUD



CU BOOL – вхід для лічіння угору (лічити, коли CU дорівнює TRUE).

CD BOOL – вхід для лічіння униз (лічити, коли CD дорівнює TRUE).

RESET BOOL – команда скидання (домінанта). (CV=0, коли RESET дорівнює TRUE).

LOAD BOOL – команда завантаження (CV=PV, коли LOAD дорівнює TRUE).

PV DINT - програмоване максимальне значення.

QU BOOL - переповнення зверху.TRUE, коли CV>=PV.

QD BOOL - переповнення знизу. TRUE, коли CV=0.

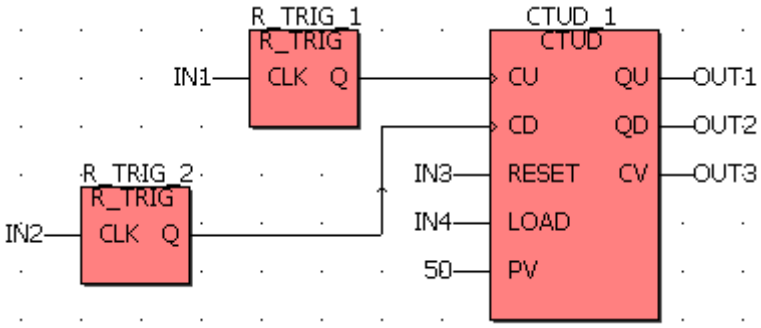
CV DINT - результат лічіння.

Опис: По фронту на CU-вході лічильник збільшується на 1. По фронту на CD-вході лічильник зменшується на 1 до 0. Коли RESET- вхід дорівнює TRUE значення CV-виходу скидається у 0. Коли LOAD-вхід дорівнює TRUE CV-вихід завантажується значенням, що дорівнює PV- входу.

Попередження: блок CTUD не визначає передній або задній фронт входів лічильника (CU I CD). Для створення імпульсного лічильника його потрібно сполучити з блоками R_TRIG або F_TRIG.

Приклад:

(* Програма FBD, що використовує блок CTUD*)



(* ST еквівалент: R_TRIG_1 і R_TRIG_2 – це екземпляри блока R_TRIG, а CTUD_1 – це екземпляр блока CTUD*).

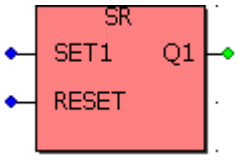
```
CTUD_1 (R_TRIG_1(IN1), R_TRIG_2(IN2), IN3,IN4,50);
OUT1:=CTUD_1.QU;
OUT2:=CTUD_1.QD;
OUT3:=CTUD_1.CV;
```

(*IL еквівалент *)

```
LD    IN1
ST    R_TRIG_1.CLK
CAL   R_TRIG_1
LD    R_TRIG_1.Q
ST    CTUD_1.CU
LD    IN2
ST    R_TRIG_2.CLK
CAL   R_TRIG_2
LD    R_TRIG_2.Q
ST    CTUD_1.CD
LD    IN3
ST    CTUD_1.RESET
LD    IN4
ST    CTUD_1.LOAD
LD    50
ST    CTUD_1.PV
CAL   CTUD_1
LD    CTUD_1.QU
ST    OUT1
LD    CTUD_1.QD
ST    OUT2
LD    CTUD_1.CV
ST    OUT3
```

Тригери

Перемикач із домінантою вмикання SR



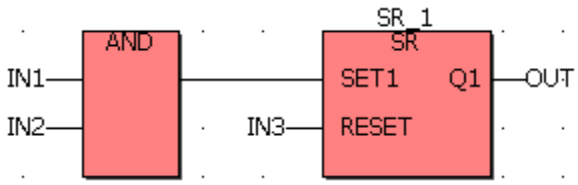
SET1 BOOL - якщо TRUE, встановлює Q1 у TRUE (домінанта).
RESET BOOL - якщо TRUE, скидає Q1 у FALS.
Q1 BOOL - логічний стан пам'яті.

Опис: Блок SR має на Q1-виході два стійких стана – TRUE і FALS. SET1-вхід вмикає вихід, а RESET- вхід вимикає його. При одночасному впливі обох входів SET1-вхід є домінантним.

Set1	Reset	Q1	result Q1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Приклад:

(*FBD-програма , що використовує блок SR*)



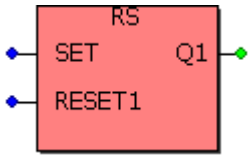
(*ST - еквівалент: SR_1- це екземпляр блока SR*)

SR_1((IN1 AND IN2),IN3);
OUT:=SR_1.Q1;

(* IL – еквівалент*)

LD	*IN1*	*ST*	*SR_1.RESET*
AND	*IN2*	*CAL*	*SR_1*
ST	*SR_1.SET1*	*LD*	*SR_1.Q1*
LD	*IN3*	*ST*	*OUT*

Перемикач із домінантою вимкнення RS



SET BOOL - якщо TRUE встановлює Q1 у TRUE .

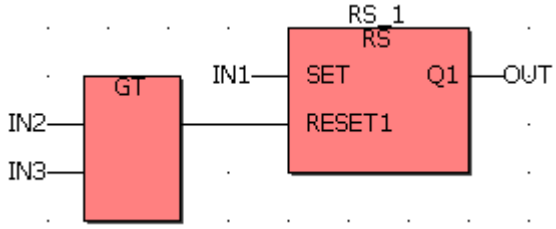
RESET1 BOOL - якщо TRUE, скидає Q1 у FALS (домінанта).
Q1 BOOL - логічний стан пам'яті.

Опис: Блок RS має на Q1-виході два стійких стана – TRUE і FALS. SET-вхід вмикає вихід, а RESET1- вхід вимикає його. При одночасній дії обох входів RESET1-вхід є домінантним.

Set	Reset1	Q1	result Q1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Приклад:

(*FBD-програма , що використовує блок RS*)



(*ST - еквівалент: RS_1- це екземпляр блока RS*)

RS_1(IN1, (IN2 GT IN3));

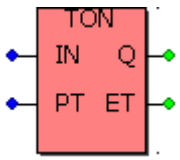
OUT:=RS_1.Q1;

(* IL – еквівалент*)

LD	*IN1*	*ST*	*RS_1.RESET1*
ST	*RS_1.SET*	*CAL*	*RS_1*
LD	*IN2*	*LD*	*RS_1.Q1*
GT	*IN3*	*ST*	*OUT*

Таймери

Таймер із затримкою вмикання TON



IN BOOL - якщо передній фронт, то починає збільшувати внутрішній таймер, якщо задній фронт, то зупиняється і скидає внутрішній таймер.

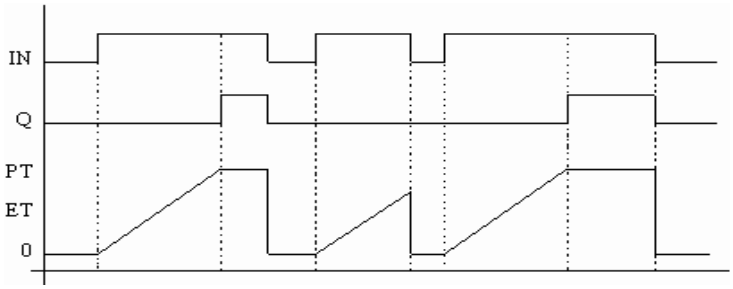
PT TIME - максимальний програмований час.

Q BOOL - якщо TRUE, програмований час використаний.

ET TIME - поточний використаний час.

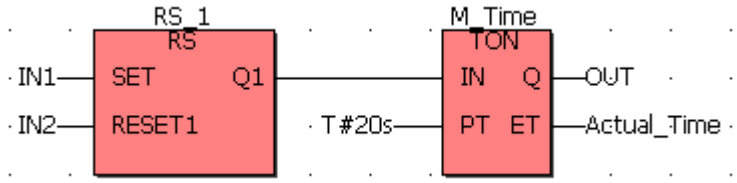
Опис: По передньому фронту IN- входу виконується скид лічильника ET і починається новий відлік часу. Q- вихід установлюється у TRUE через заданий на PT-вході час, якщо IN-вхід продовжує залишатися у стані TRUE. По задньому фронту IN- входу зупиняється відлік часу і Q-вихід скидається у FALS. Таким чином, Q- вихід вмикається логічною одиницею, яка триває протягом часу показаному у PT – вході, а вимикається по задньому фронту IN- входу.

Часова діаграма має вигляд:



Приклад:

(*FBD-програма , що використовує блок TON*)



(*ST - еквівалент: RS_1- це екземпляр блока RS, а M_Time - екземпляр блока TON*)

M_Time(RS_1(IN1,IN2),T#20s)

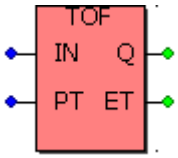
OUT:=M_TIME_1.Q;

Actual_Time:=M_TIME_1.ET;

(* IL – еквівалент*)

```
 LD   IN1                    LD   T#20s
 ST   RS_1.SET               ST   M_Time.PT
 LD   IN2                    CAL  M_Time
 ST   RS_1.RESET1            LD   M_Time.Q
 CAL  RS_1                   ST   OUT
 LD   RS_1.Q1                LD   M_Time.ET
 ST   M_Time.IN              ST   Actual_Time
```

Таймер із затримкою вимикання TOF



IN BOOL - якщо задній фронт, то починає збільшувати внутрішній таймер, якщо передній фронт, то зупиняється і скидає внутрішній таймер.

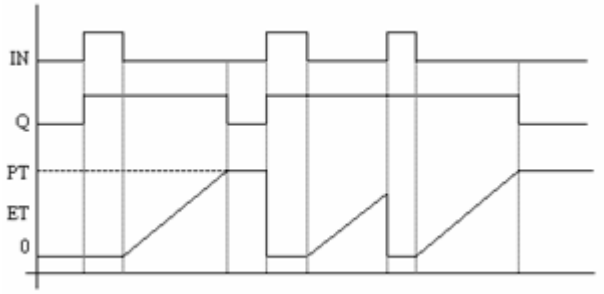
PT TIME - максимальний програмований час.

Q BOOL - якщо TRUE, програмований час використаний.

ET TIME - поточний використаний час.

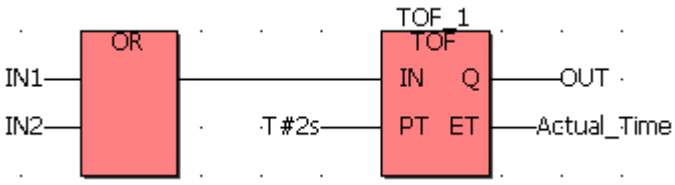
Опис: По задньому фронту IN- входу виконується скид лічильника ET, Q- вихід установлюється у TRUE і починається новий відлік часу. Через заданий на PT- вході час Q- вихід скидається. Якщо під час відліку IN- вхід установлюється у стані TRUE, то відлік призупиняється. Таким чином, Q-вихід вмикається по задньому фронту, а вимикається логічним нулем тривалістю не менш, як на PT-вході.

Часова діаграма має вигляд:



Приклад:

(*FBD-програма , що використовує блок TOF*)



(*ST - еквівалент: TOF_1- екземпляр блока TOF*)

TOF_1((IN1 OR IN2), T#2s);

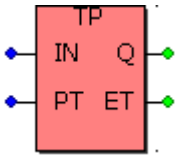
OUT:= TOF_1.Q;

Actual_Time:=TOF_1.ET

(* IL – еквівалент*)

```
LD   IN1                CAL TOF_1
OR   IN2                LD   TOF_1.Q1
ST   TOF_1.IN           ST   OUT
LD   T#2s               LD   TOF_1.ET
ST   TOF_1.PT           ST   Actual_Time
```

Час пульсування TP



IN BOOL - якщо передній фронт, то починає збільшувати внутрішній таймер (якщо вже не збільшується). Якщо FALSE і лише, якщо таймерний інтервал вичерпано, скидає внутрішній таймер.
Будь-яка змінна IN під час лічіння не має ефекту.

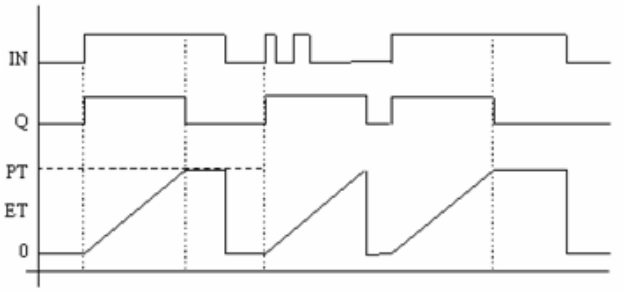
PT TIME - максимальний програмований час.

Q BOOL - якщо TRUE таймер лічить.

ET TIME поточний використаний час.

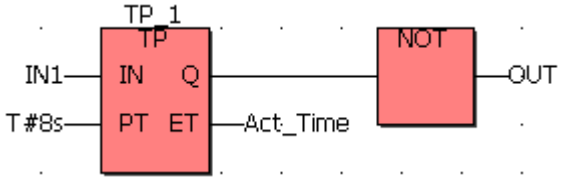
Опис: Запуск таймера здійснюється по передньому фронту імпульсу на IN-вході, PT-вхід задає тривалість імпульсу на Q-виході . Після запуску таймер не реагує на зміну значення IN- входу. ET-вихід лічить час і, коли значення досягає установленого на PT-вході, лічильник зупиняється і Q-вихід скидається у 0.

Часова діаграма має вигляд:



Приклад:

(*FBD-програма , що використовує блок TP*)



(*ST - еквівалент: TP_1- екземпляр блока TP*)

TP_1(IN1,T#8s);
Act_Time:=TP_1.ET;
OUT:=NOT(TP_1.Q);

(* IL – еквівалент*)

LD	*IN1*	*LD*	*TP_1.Q*
ST	*TP_1.IN*	*NOT*	
LD	*T#8s*	*ST*	*OUT*
ST	*TP_1.PT*	*LD*	*TP_1.ET*
CAL	*TP_1*	*ST*	*Act_Time*

Контрольні запитання

1. Які мови визначені стандартом IEC 61131-3?
2. Які існують стандартні мовні елементи?
3. Які оператори складають групу булевих?
4. Коли використовуються арифметичні оператори?
5. Якими символами зображаються оператори порівняння?
6. Які функції є стандартними?
7. Коли застосовуються детектори імпульсів?
8. Чим відрізняються функціональні блоки R_TRIG і F_TRIG?
9. Які бувають лічильники і в чому між ними різниця?
10. Чим характеризуються тригери SR і RS?
11. Який таймер із затримкою вмикання?
12. Як працює таймер TOF?
13. Що таке TP-таймер?

1.2.2. **LD-мова**

Мова Крокових діаграм LD (Ladder Diagram) або релейних діаграм, або релейної логіки у графічному вигляді зображає алгоритм розв'язання задач логіко-командного управління. Це одна з мов програмування мікропроцесорних контролерів, які вперше були використані для заміни релейно-контактних схем керування. Тому у вітчизняній техніці ця мова мала саме таку назву –"Мова релейно - контактних схем".

Для опису логічних виразів різного рівня складності ця мова використовує, як базові елементи програмування, графічні елементи "контакти" (contacts) і "котушки" (coils), сполучені відповідно з вхідними та вихідними каналами. Зважаючи на свої обмежені можливості, LD-мова доповнена привнесеними засобами – (таймерами, лічильниками, блоками порівняння, обчислювальними блоками, ПІД-регуляторами та ін.), що дає можливість розв'язувати інші більш складні задачі.

Основою створення технологічної LD-мови є різні варіанти мови релейно-контактних схем фірм виробників ПЛК- контролерів (Allen-Bredley, AEG Schneider Automation, GE-Fanuc, Siemens).

1.2.2.1. **Синтаксис LD-мови**

Графічні символи LD-мови зображаються всередині програми так саме, як і елементи релейної автоматики в електричній схемі.

Для зображення вхідних контактів використовуються символи:

 прямий контакт;

 інвертований контакт.

А для зображення вихідних котушок використовуються символи:

 пряма котушка;

 інвертована котушка;

SET котушка;

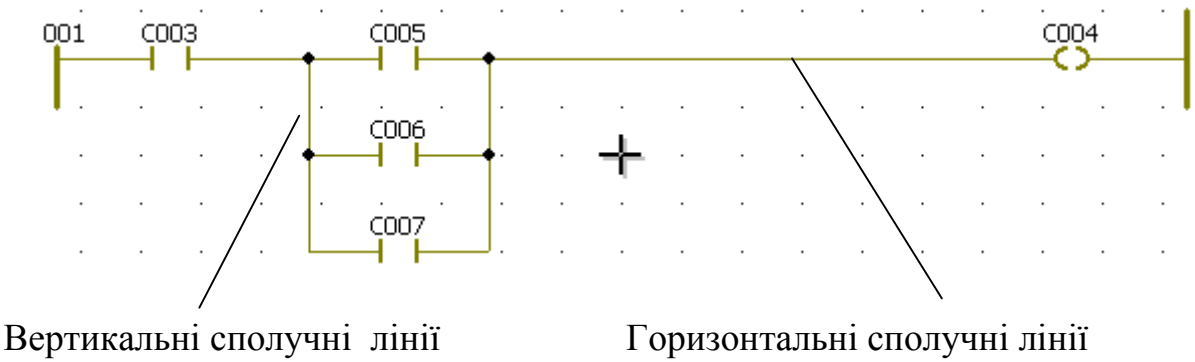
RESET котушка.

Над графічними символами пишеться ім'я змінної, яке фактично є його назвою. Зазвичай, при створені програми графічним символам автоматично присвоюється типове ім’я з відповідним номером, яке може коригуватися користувачем на власний розсуд.

Кількість вхідних контактів у LD- діаграмі може бути скільки завгодно, а вихідна котушка тільки одна. При необхідності паралельно можна вмикати декілька котушок, проте послідовне вмикання їх не допускається.

LD-діаграма обмежена справа і зліва вертикальними лініями, які називаються відповідно лівою шиною живлення і правою шиною живлення.

Графічні символи LD-діаграми сполучені з шинами живлення або іншими символами за допомогою сполучних ліній. Сполучні лінії можуть бути горизонтальними або вертикальними.

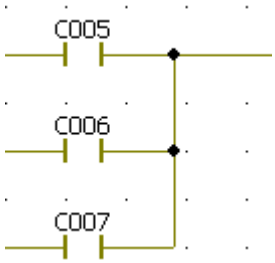


Кожен сегмент лінії має стан FALSE або TRUE. Сегменти, сполучені разом, мають один і той же логічний стан. Будь-яка горизонтальна лінія, сполучена з лівою шиною живлення, має стан TRUE.

Комбінуючи вертикальні і горизонтальні лінії, можна будувати множинні сполучення. Логічний стан кінців множинного сполучення визначається правилами логіки**.**

Ліве множинне сполучення об'єднує декілька горизонтальних ліній, сполучених з вертикальною лінією зліва і одну горизонтальну лінію, сполучену з її правою стороною. Булевий стан правого кінця є ЛОГІЧНИМ АБО (OR) між всіма лівими кінцями.

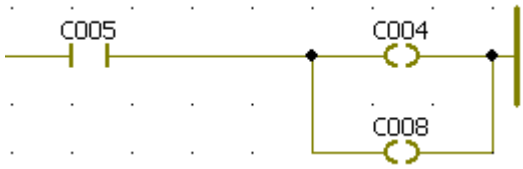
Приклад:



Стан правого кінця:= (C005 OR C006OR C007) *)

Праве множинне сполучення об'єднує одну горизонтальну лінію, сполучену з вертикальною лінією з лівого боку, і декілька горизонтальних ліній, що підходять до її правої сторони. Булевий стан лівого кінця розповсюджується на всі праві кінці.

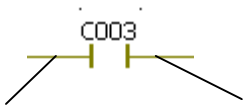
Приклад:



Наведена схема еквівалентна виразу: C004:=C005; C008:=C005.

Множинні сполучення зліва і справа об'єднують декілька горизонтальних ліній, сполучених з вертикальною лінією з лівого боку і декілька горизонтальних ліній, що підходять до її правої сторони. Булевий стан кожного правого кінця є ЛОГІЧНИМ АБО (OR) між всіма лівими кінцями.

Прямий контакт дозволяє виробляти логічні операції між станом лінії і логічної змінної.



Ліве сполучення Праве сполучення

Стан правого кінця лінії сполучення є логічним ’I’ (AND) між станом лівого кінця і значенням змінної, асоційованої з контактом.

Приклад:

Наведена схема еквівалентна виразу: C004:=C003 AND C005

Інвертований контакт дозволяє виробляти логічні операції між станом лінії і запереченням логічної змінної.

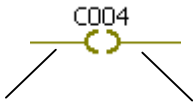
Ліве сполучення Праве сполучення

Стан правого кінця лінії сполучення є логічним ’I’ (AND) між станом лівого кінця і запереченням значення змінної, асоційованої з контактом.

Приклад:

Наведена схема еквівалентна виразу: C004 := NOT (C003) AND NOT (C005);

Пряма котушка визначає логічний вихід стану лінії сполучення**.**

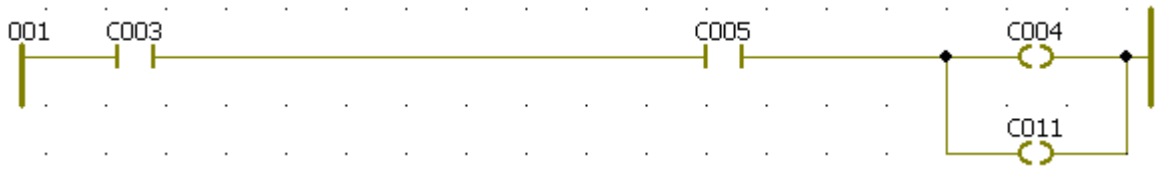


Ліве сполучення Праве сполучення

Змінній котушки призначається стан лівого сполучення. Стан лівого сполучення розповсюджується на праве сполучення. Праве сполучення може бути приєднане до правої вертикальної шини живлення.

Асоційована логічна змінна повинна бути ВИХОДОМ або ВНУТРІШНЬОЮ.

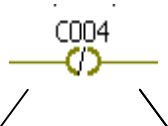
Приклад:



Наведена схема еквівалентна виразу: C004:= C003 AND C005;

C011:= C003 AND C005.

Інвертована котушка визначає логічний вихід, відповідний запереченню стану лінії сполучення**.**

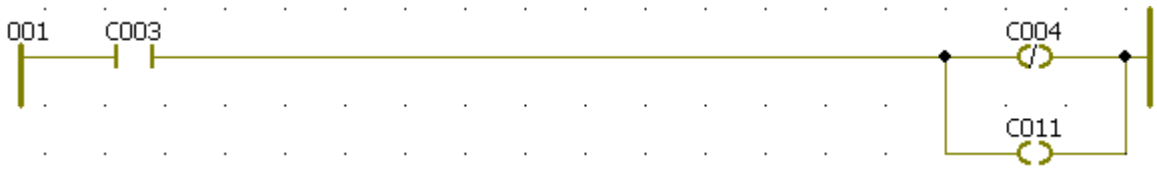


Ліве сполучення Праве сполучення

Змінній котушки призначається заперечення стану лівого сполучення. Стан лівого сполучення розповсюджується на праве сполучення. Праве сполучення може бути підключене до правої вертикальної шини живлення.

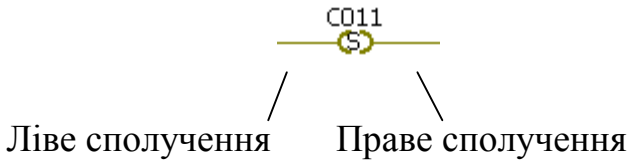
Асоційована логічна змінна повинна бути ВИХОДОМ або ВНУТРІШНЬОЮ.

Приклад:



Наведена схема еквівалентна виразу: C004:= NOT(C003); C011:= C003.

'Set' котушка визначає логічний вихід стану лінії сполучення**.**



Асоційована змінна встановлюється в TRUE (SET TO TRUE), коли стан лівого сполучення стає рівним TRUE. Вихідна змінна утримує цей стан до тих пір, поки вона не буде інвертована котушкою 'RESET'. Стан лівого сполучення розповсюджується на праве сполучення. Праве сполучення може бути підключене до правої вертикальної шини живлення.

Асоційована логічна змінна повинна бути ВИХОДОМ або ВНУТРІШНЬОЮ.

Приклад:

Наведена схема еквівалентна виразу:

```
IF C003 THEN
    C004 := TRUE;
END_IF;
IF C012 THEN
    C004 := FALSE;
END_IF;
```

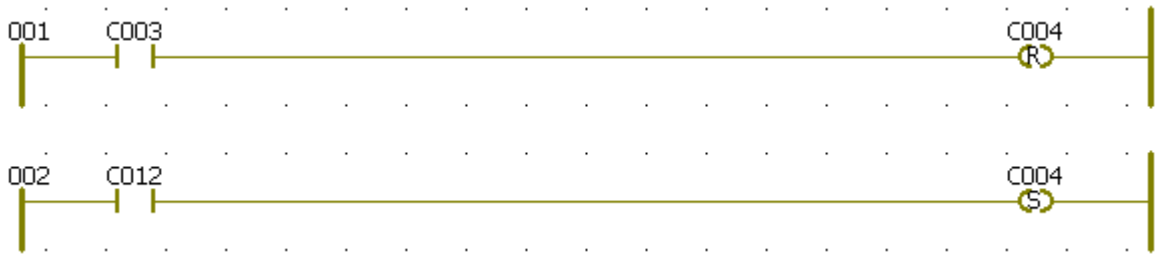
"Reset" котушка визначає логічний вихід стану лінії сполучення**.**

Ліве сполучення Праве сполучення

Асоційована змінна встановлюється в FALSE (RESET TO FALSE), коли стан лівого сполучення стає рівним TRUE. Вихідна змінна утримує цей стан до тих пір, поки вона не буде інвертована котушкою "SET". Стан лівого сполучення розповсюджується на праве сполучення. Праве сполучення може бути підключене до правої вертикальної шини живлення.

Асоційована логічна змінна повинна бути ВИХОДОМ або ВНУТРІШНЬОЮ.

Приклад:



Наведена схема еквівалентна виразу:

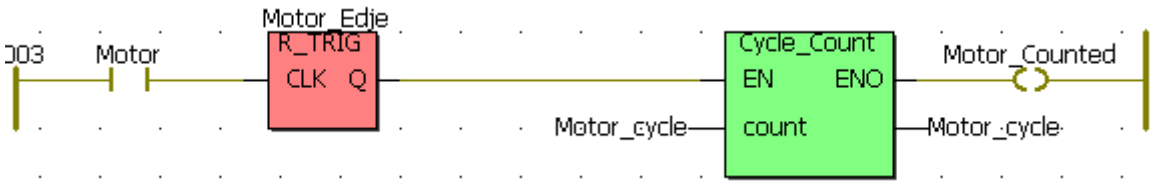
```
IF C003 THEN
    C004 := FALSE;
END_IF;
IF C012 THEN
    C004 := TRUE;
END_IF;
```

Окрім розглянутих елементів LD – мови, першою редакцією стандарту IEC 61131-3 ще визначалися контакти та котушки, які керуються фронтами імпульсів: контакт −│P│− і котушка –(P)− переднього фронту , контакт −│N│− і котушка –(N)− заднього фронту. У теперішній час підтримка таких контактів і котушок засобами програмування не є обов’язковою, оскільки аналогічна LD – діаграма легко створюється за допомогою функціональних блоків R_TRIG і F_TRIG.

Таким чином, у LD разом з контактами та котушками можна використовувати функції та функціональні блоки. Оскільки блоки не завжди мають логічні входи і/або виходи, введення блоків в LD- діаграму призводить до додавання в інтерфейс блока нових параметрів – EN та ENO .

У деяких функціях і функціональних блоках перший вхід або перший вихід не є булєвим. Тому при використанні їх в LD- програмах штучно створюються логічний вхід '**EN**' і логічний вихід '**ENO'.** Блок виконується тільки тоді, коли вхід '**EN**' дорівнює TRUE. Вихід '**ENO**' завжди має те саме

значення, що і перший вхід блока. У деяких випадках потрібні обидва параметри '**EN**' і '**ENO'.** Нижче наведено приклад використання функціонального блока R_TRIG разом з функцією користувача Cycle_Count (Рахувати цикли), яка має логічний вхід '**EN**' і логічний вихід '**ENO**'.



Послідовність виконання LD–діаграм можна примусово змінити, використовуючи мітки та переходи.

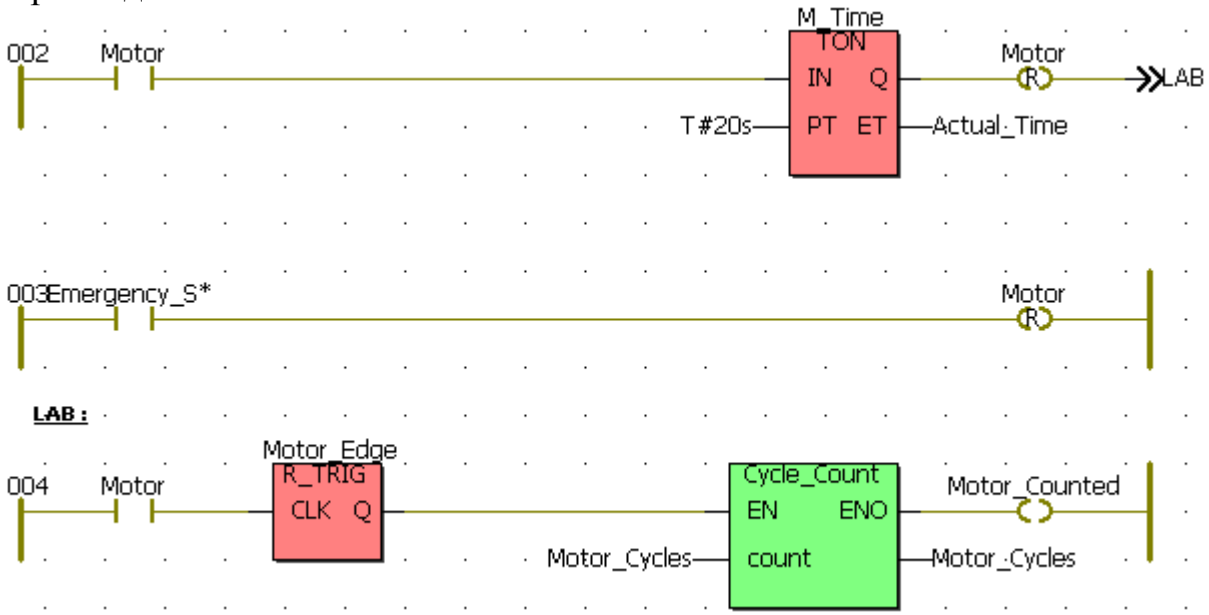
Мітка ставиться тільки на початку LD – діаграми, при цьому для наочності після неї можна ставити дві крапки.

>>LAB - перехід на мітку "LAB";

(LAB:) – мітка на початку діаграми.

LD – діаграма може мати тільки одну мітку та один перехід. Перехід розглядається як вихідне реле і виконується, коли вихідна змінна встановлюється в TRUE . Використовуючи перехід, можна пропустити виконання деяких діаграм.

Приклад:

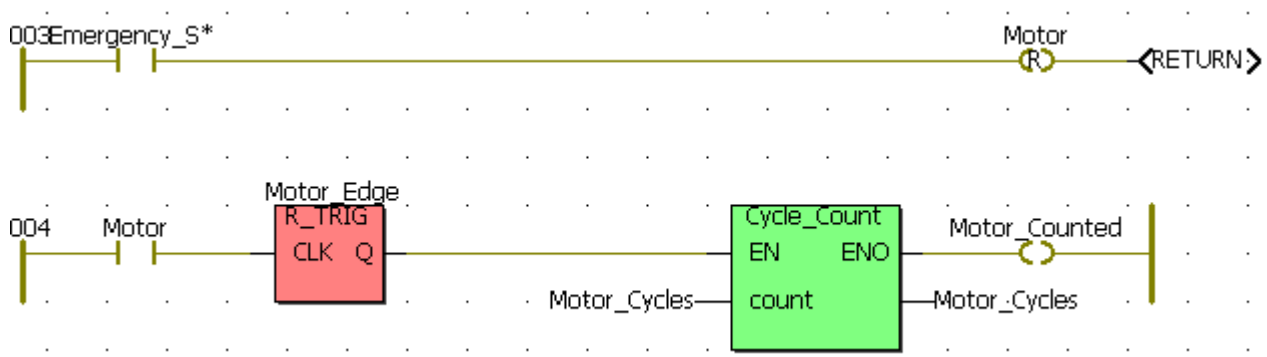


Проте, бажано не займатися керуванням послідовності виконання LD – діаграм, оскільки це суперечить їх аналогії з релейними схемами. Простіше порядок виконання описувати SFC-мовою.

Спеціальний перехід RETURN використовується для умовного завершення програми. Ніяких символів до правого кінця RETURN підключати не можна.

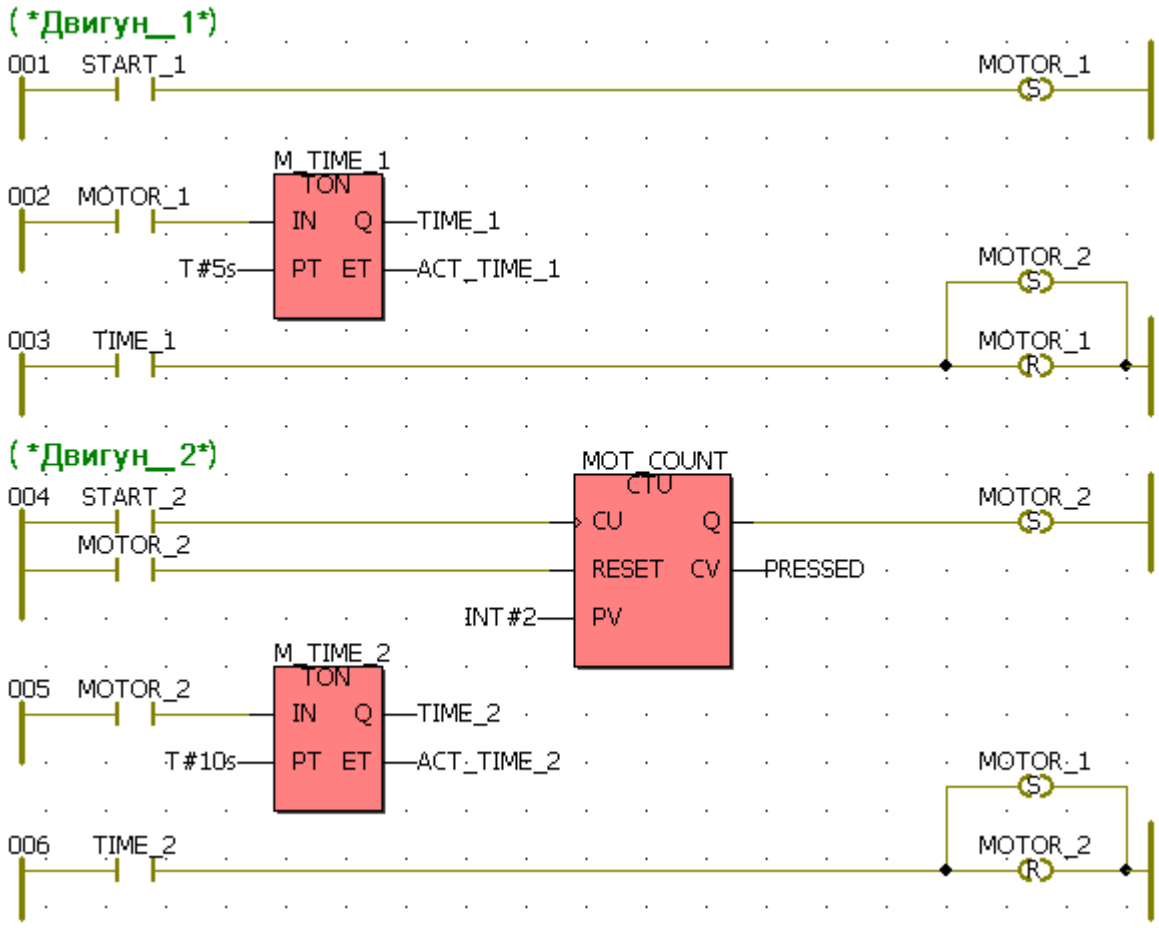
Якщо лівий кінець лінії сполучення має стан **TRUE**, то програма завершується, не виконуючи подальші рядки діаграми[6,7].

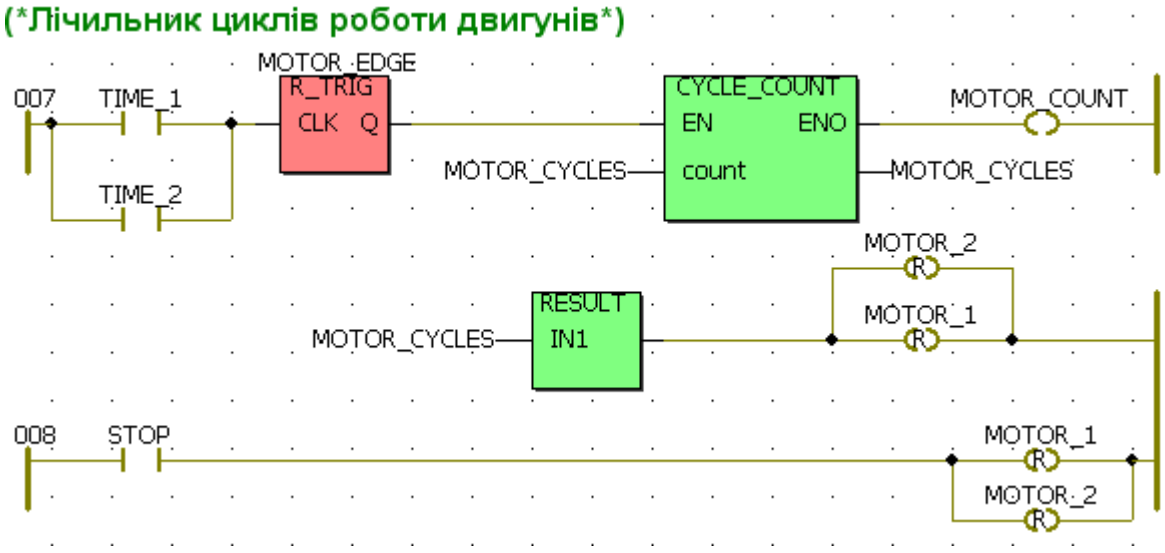
Приклад:



1.2.2.2. **Приклад програмування LD-мовою**

Розробимо програму керування роботою двох двигунів, які після запуску одного з двигунів безперервно по черзі вмикаються і вимикаються. Тривалість роботи двигунів відповідно 5 і 10 секунд. Початковий запуск першого двигуна здійснюється одноразовим натисненням пускової кнопки, а початковий запуск другого двигуна – дворазовим. При цьому двигуни можна у будь-яку мить зупинити, а кожне вмикання двигунів підраховується лічильником. Коли кількість вмикань двигунів досягає десяти, їх робота автоматично зупиняється. Проектний код, що реалізує наведений алгоритм, має вигляд:





Тут:
START_1 і START_2 - кнопки початкового пуску двигунів;
MOTOR_1 і MOTOR_2 – перший і другий двигуни;
M_TIME_1 і M_TIME_2 – екземпляри функціонального блока таймера TON, які відлічують час роботи двигунів;
MOT_COUNT – екземпляр функціонального блока лічильника CTU, який підраховує кількість натисків на кнопку START_2;
MOTOR_EDGE – екземпляр функціонального блока детектора імпульсу переднього фронту R_TRIG.
CYCLE_COUNT(*CYCLE_COUNT:=count+1;)*– функція користувача ST-мовою, яка підраховує активізації двигунів;
RESULT*(RESULT:=IN1=10;)* – функція користувача ST-мовою, яка порівнює поточну кількості вмикань двигунів з уставкою 10;
STOP – кнопка непередбаченої зупинки двигунів.

Контрольні запитання

1. Які графічні символи використовуються при редагуванні LD-мовою?
2. Що таке шина живлення?
3. Які бувають множинні сполучення?
4. У чому різниця між прямим і інвертованим контактами?
5. Коли змінній котушки призначається стан лівого сполучення?
6. Коли асоційована змінна 'Set' котушки встановлюється в TRUE?
7. Що трапляється з асоційованою змінною "Reset" котушки, коли стан лівого сполучення стає рівним TRUE?
8. Коли створюються логічний вхід 'EN' і логічний вихід 'ENO'?
9. Чи можна змінити послідовність виконання LD–діаграм?
10. Коли використовується ключове слово RETURN?

1.2.3. **FBD-мова**

Мова Функціональних блокових діаграм FBD (Functional Block Diagram) або функціональних блоків призначена для інженерів-технологів, які

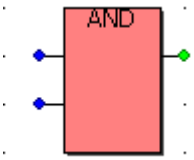
розв’язують задачі керування технологічними процесами. FBD-мова дозволяє створити програмну одиницю практично будь-якої складності на основі стандартних «цеглинок» (арифметичні, тригонометричні та логічні блоки, ПІД-регулятори і т.д.), які сполучаються між собою за допомогою входів і виходів.

Коріння мови з’ясувати складно, проте більшість фахівців вважають, що це не що інше, як перенесення ідей мови релейно-контактних схем на іншу елементну базу. Дійсно, замість реле в цій мові використовуються функції, функціональні блоки, а також стандартні оператори у вигляді функцій, які знаходяться у бібліотеках. Додатково користувач може створювати власні блоки, які він будує із стандартних елементів або С-мовою і оформляє їх, як об’єкти бібліотек.

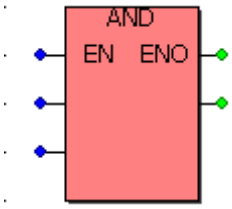
1.2.3.1. **Синтаксис FBD-мови**

FBD – програма (діаграма) – це блок-схема з певної кількості елементарних функцій та функціональних блоків, які зображаються прямокутниками.

Функції – це комбінаційні схеми. Вони не мають внутрішніх умов, тому при кожному їх виконанні одному і тому ж значенню вхідних величин відповідають ті ж самі значення вихідної величини, як це трапляється, наприклад, з функцією складання двох величин. Функція має декілька входів і тільки один вихід. Входи завжди розташовані зліва, виходи - справа. Назва функції записується усередині прямокутника. Графічне зображення функції має вигляд:



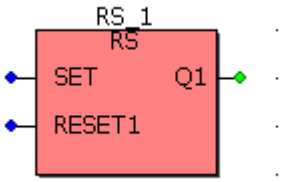
У деяких функціях перший вхід або перший вихід не є булевим. Тому в процесі програмування при необхідності можна штучно створити логічний вхід '**EN**' і логічний вихід '**ENO**'**.**



Тоді алгоритм, визначений функцією, буде виконуватися, коли вхід '**EN**' дорівнює TRUE. Після виконання алгоритму без помилок значення виходу '**ENO**' автоматично стає рівним одиниці. Коли виникають помилки при виконанні алгоритму, значення '**ENO**' стає рівним нулю.

Функціональні блоки – це програмні одиниці з багатьма вхідними та вихідними параметрами і оперативною пам’яттю. Тому вони можуть мати різний стан, завдяки чому при неодноразових виконаннях функціональних блоків одним і тим же значенням вхідних величин можуть відповідати різні

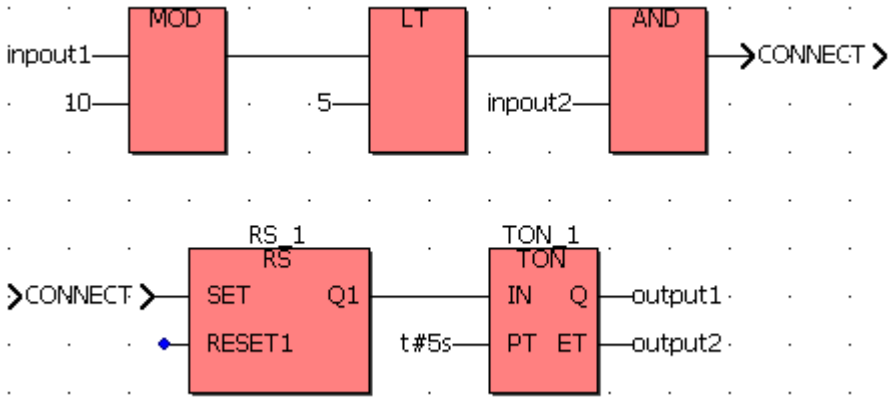
значення на виході. Наприклад, значення виходу лічильника збільшується, тобто має бути різним при незмінних одиничних вхідних імпульсах. Графічне зображення функціонального блока має вигляд:



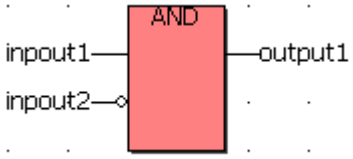
Назва функціонального блока, а також назви усіх входів і виходів записуються всередині прямокутника. Над блоком показується назва екземпляра функціонального блока.

Входом FBD- блока може бути константа, будь-яка внутрішня вхідна або вихідна змінна. Виходом – будь-яка внутрішня або вихідна змінна, або ім'я функції (тільки для функцій).

Вхідні та вихідні змінні сполучаються з відповідними входами та виходами блок. При цьому тип кожної змінної такий самий, як і тип відповідного входу або виходу. Вхідні і вихідні змінні, входи і виходи блоків сполучаються разом лініями або з'єднувачами. З'єднувачі (connectors) використовуються при обмеженій ширині екрана, коли є необхідність розірвати сполучення і перенести його в наступну мережу. Тому FBD-програма не обов'язково зображає собою велику єдину схему, частіше вона має вигляд декількох мереж, які послідовно виконуються.



Одинична лінія правим кінцем, приєднаним до входу блока, може закінчуватися логічним запереченням. Заперечення зображається маленьким кільцем.



(* ST еквівалент: *)

output1 := input1 AND NOT (input2).

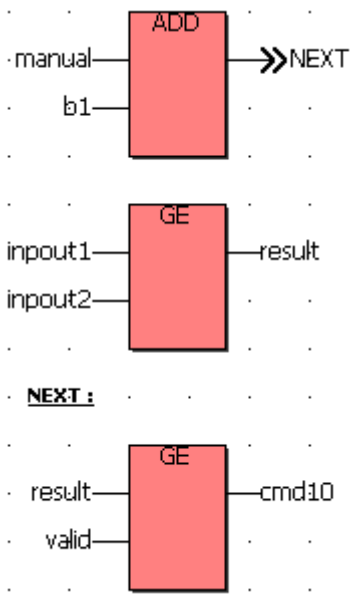
Коли використовується логічне заперечення, лівий і правий кінці лінії зв'язку повинні мати тип BOOL.

Виконання FBD-програми здійснюється зліва направо, зверху вниз. Блок починає обчислюватися тільки після того, як будуть знайдені значення усіх його входів. Подальші обчислювання не будуть продовжені поки значення на всіх виходах не будуть обчислені. Обчислення мережі програми є закінченим тільки після обчислення значень на виходах усіх складових елементів.

Послідовність виконання FBD-програми можна примусово змінити, використовуючи мітки та переходи, які обов'язково мають однакові імена.

Мітка з двома крапками вставляється попереду будь-якої мережі і стає її назвою, а перехід на мітку у вигляді стрілки показується біля виходу блока, з якого треба змінити хід програми. Перехід обов'язково зв'язаний з логічною змінною і виконується, коли вона має значення TRUE. Для створення безумовного переходу використовується константа TRUE, що зв'язана з переходом.

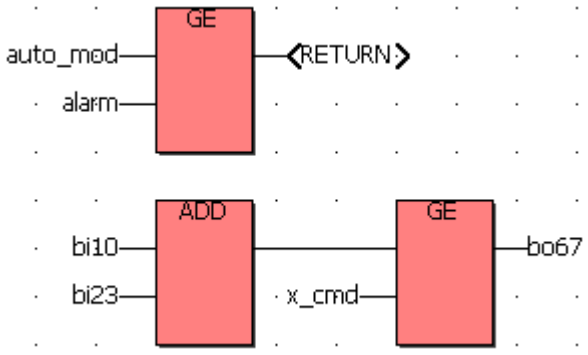
Наприклад:




```
(* IL еквівалент: *)
LD      manual
ADD     b1
JMPC    NEXT
LD      input1
OR      input2
ST      result
LD      result
OR      valid
ST      cmd10
```

Для виходу із програми використовується ключове слово <RETURN>.

Якщо вихід блока, сполученого з оператором <RETURN>, має тип TRUE, решта частини діаграми не виконується[4,7-9].



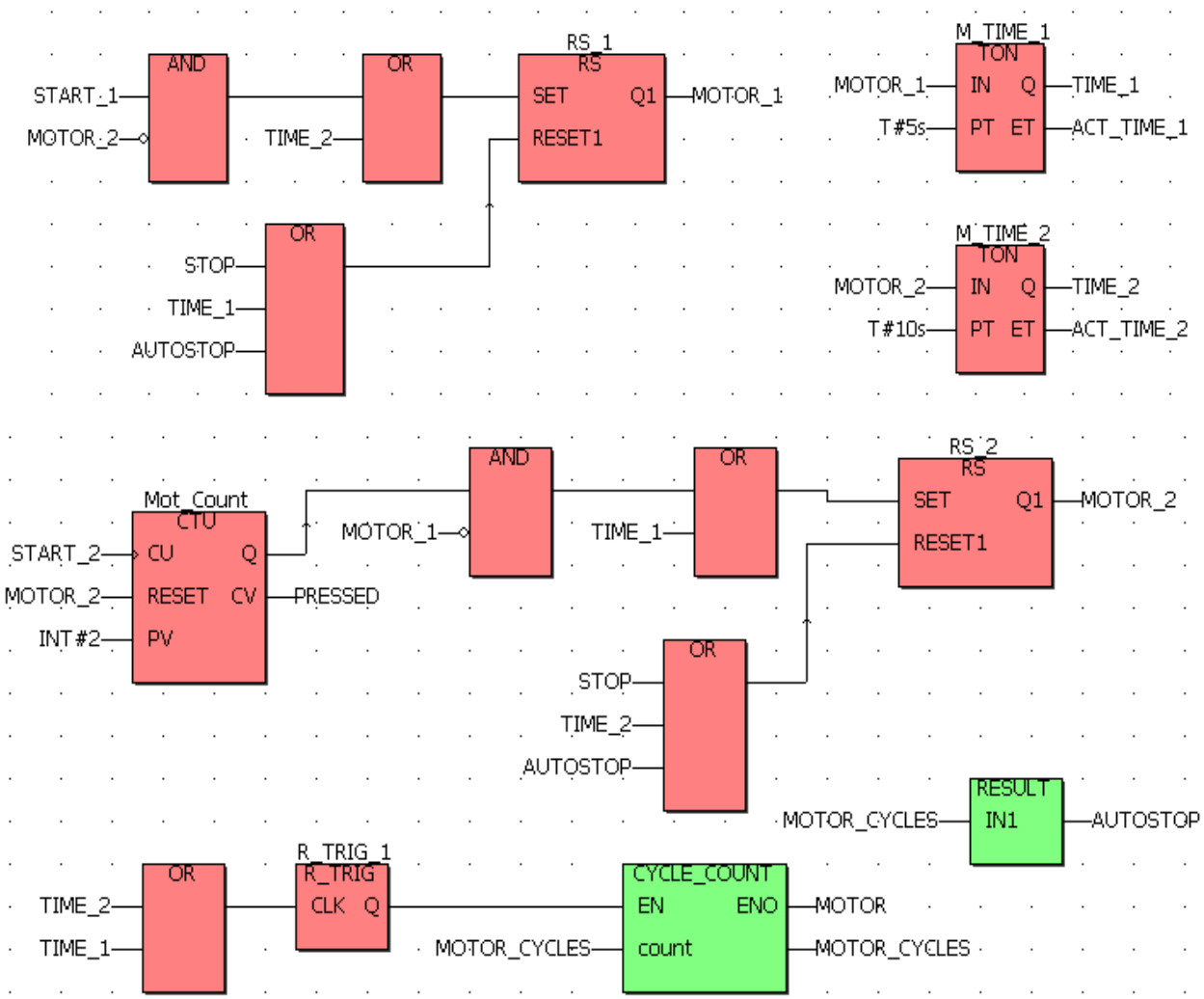
(* ST еквівалент: *)

If auto_mode OR alarm Then
Return;
End_if;
bo67 := (bi10 ADD bi23) OR x_cmd;

1.2.3.2. **Приклад програмування FBD - мовою**

Як приклад програмування FBD – мовою розробимо програму керування роботою двох двигунів згідно з алгоритмом наведеним у параграфі 1.2.2.2.

Проектний код FBD-мовою має вигляд:



Тут:

START_1 і START_2 - кнопки початкового пуску двигунів;

MOTOR_1 і MOTOR_2 – перший і другий двигуни;

M_TIME_1 і M_TIME_2 – екземпляри функціонального блока таймера TON, які відлічують час роботи двигунів;

Mot_Count – екземпляр функціонального блока лічильника CTU, який підраховує кількість натисків на кнопку START_2;

RS_1 і RS_2 – екземпляри функціонального блока детектора імпульсу переднього фронту R_TRIG.

CYCLE_COUNT(*CYCLE_COUNT:=count+1;)*–функція користувача, яка підраховує активізації двигунів;

RESULT*(RESULT:=IN1=10;)* – функція користувача, яка порівнює поточну кількість вмикань двигунів з уставкою 10;

STOP – кнопка непередбаченої зупинки двигунів.

Контрольні запитання

1. Який вигляд має FBD-програма?
2. Як зображається функція в FBD-програмі?
3. Як зображається функціональний блок в FBD-програмі?
4. Що може бути входом, а що виходом FBD-блока?
5. Які вимоги до типів змінних входів і виходів блоків, що сполучаються між собою?
6. Коли використовуються з'єднувачі?
7. Чи можна примусово змінити послідовність виконання FBD- програми?

1.2.4. **IL-мова**

Список інструкцій IL (Instruction List) або командний список є текстовою мовою низького рівня. Це найпростіша мова мнемонічних інструкцій, яка нагадує мову Асемблера і призначена для програмування контролерів малої потужності. Програми, що створені IL-мовою, легко транслюються в машинні коди будь-якого процесора і це дозволяє створювати дуже швидкі програми.

Проте, реально IL-мова не має ніяких переваг перед ST або FBD-мовами, оскільки проблема продуктивності вже давно розв'язана іншими методами. Тому самостійного значення не має і використовується тільки спільно з SFC-мовою. Походження – мова програмування STEP-5 німецької фірми Siemens.

1.2.4.1. **Синтаксис IL-мови**

IL-програма - це список послідовних інструкцій. Кожна інструкція програми починається з нового рядка і містить оператор, доповнений необов'язковими модифікаторами і, якщо необхідно для специфічних операцій, один або декілька операндів, розділених комами. Інструкції може передувати мітка з двокрапкою на кінці. Вона не є обов'язковою і ставиться тільки там, де потрібно. Якщо інструкція супроводжується коментарем, то він має бути останнім компонентом рядка. Коментар завжди починається символами '(*' і закінчується символами '*)'. Між інструкціями можуть бути введені порожні рядки, де записуються коментарі, якщо є в цьому необхідність.

Синтаксис інструкції має вигляд:

< Мітка:> < оператор> < модифікатор>< операнд> (коментар*)*

Інструкції завжди відносяться до поточного результату, який знаходиться в IL- регістрі (в асемблері це аккумулятор, а стандарт IEC 61131-3 маркірує

його, як результат). Оператор визначає операцію, яка має бути виконана з поточним результатом і операндом. Результат операції знову запам'ятовується в IL- регістрі. Через те, що існує тільки один IL- регістр (поточний результат), деякі операції можуть бути затримані, таким чином, порядок виконання інструкцій може бути змінений. Для того, щоб виділити затримані операції, використовуються дужки.

В табл. 1.4 приведені IL - оператори з поясненнями і допустимими модифікаторами:

Таблиця 1.4. Оператори IL-мови

Оператор	Модифі-катор	Операнд	Опис
LD	N	Змінна, константа	Завантажує значення операнда у поточний результат
ST	N	Змінна	Запам'ятовує поточний результат у змінній. Поточний результат цією операцією не змінюється.
S		BOOL змінна	Зберігає значення TRUE в булевій змінній, якщо поточний результат має значення TRUE. Ніяка операція не виконується, якщо поточний результат рівний FALSE. Поточний результат не модифікується
R		BOOL змінна	Зберігає значення FALSE в булевій змінній, якщо поточний результат має значення TRUE. Ніяка операція не вико-нується, якщо поточний результат рівний FALSE. Поточний результат не модифікується.
AND	N (	BOOL	Логічне AND
&	N (	BOOL	Логічне AND
OR	N (	BOOL	Логічне OR
XOR	N (	BOOL	Виключне OR
ADD	(	Змінна, константа	Складання
SUB	(	Змінна, константа	Віднімання
MUL	(	Змінна, константа	Множення

Продовження таблиці 1.4

DIV	(	Змінна, константа	Ділення
GT	(	Змінна, константа	Більше: >
GE	(	Змінна, константа	Більше або дорівнює: >=
EQ	(	Змінна, константа	Дорівнює: =
LE	(	Змінна, константа	Менше або дорівнює: <=
LT	(	Змінна, константа	Менше: <
NE	(	Змінна, константа	Не рівно: <>
CAL	C N	Ім'я екземпляра функц. блока	Виклик функціонального блока
JMP	C N	Мітка	Стрибок на мітку
RET	C N		Закінчення виконання поточної IL програми. Якщо IL послідовність є функцією, то поточний результат повертається у визивну програму.
)			Виконує затриману операцію. Затримана операція позначається "("

Список команд має завжди починатися з оператора LD (команда завантаження IL- регістра) і закінчуватися оператором збереження ST.

Приклад:

```
LD   20
SUB 8
ST    M
```

Тут в IL- регістр (акумулятор) завантажується число 20, з якого віднімається 8, а результат запам’ятовується у змінній М. Після цього вміст змінної та IL- регістра (акумулятора) дорівнює 12.

Операції порівняння також завжди виконуються з акумулятором (IL-регістром), а булевий результат дії залишається в акумуляторі.

Приклад:

```
LD  M
LT  5
```

Тут значення змінної М завантажено у акумулятор і порівнюється з числом 5. Якщо М більше або дорівнює 5, вміст акумулятора (IL- регістра) або результат - 0 (FALS), якщо менше 5 – 1 (TRUE).

Приклади використання деяких операторів IL-мови :

Оператора LD

```
LD    false      (* результат в IL- регістрі:= булева константа FALSE *)
LD    true       (* результат в IL- регістрі := булева константа TRUE *)
LD    123        (* результат в IL- регістрі := цілочислова константа *)
LD    123.1      (* результат в IL- регістрі := дійсна константа *)
LD    t#3ms      (* результат в IL- регістрі:= часова константа *)
LD    boo_var1   (* результат в IL- регістрі := булева змінна *)
LD    ana_var1   (* результат в IL- регістрі:= цілочислова змінна *)
LD    tmr_var1   (* результат в IL- регістрі:= часова змінна *)
```

Оператора ST:

```
LD    false      (* поточний результат в IL- регістрі:= FALSE *)
ST    boo_var1 (* boo_var1 := FALSE *)
LD    123        (* поточний результат в IL- регістрі:= 123*)
ST    ana_var1 (* ana_var1 := 123 *)
LD    t#12s      (* поточний результат в IL- регістрі:= t#12s*)
ST    tmr_var1 (* tmr_var1 := t#12s *)
```

Оператора S:

```
LD    true       (* поточний результат в IL- регістрі:= TRUE *)
S     boo_var1 (* boo_var1 := TRUE *)
                 (* поточний результат в IL- регістрі не змінюється *)
LD    false      (* поточний результат в IL- регістрі:= FALSE *)
S     boo_var1 (* нічого не зроблене - boo_var1 не змінюється *)
```

Оператора R:

```
LD    true       (* поточний результат в IL- регістрі:= TRUE *)
R     boo_var1 (* boo_var1 := FALSE *)
                 (* поточний результат в IL- регістрі не змінюється *)
ST    boo_var2 (* boo_var2 := TRUE *)
LD    false      (* поточний результат в IL- регістрі:= FALSE *)
R     boo_var1 (* нічого не зроблене - boo_var1 не змінюється *)
```

Додавання до мнемоніки деяких операторів символів - модифікаторів 'C', 'N' і '(' модифікує значення інструкції. Символ модифікатора завершує ім'я оператора без пропуску, наприклад: JMPC, ANDN, CALCN.

Модифікатор 'N' (negation) інвертує значення операнда до виконання інструкції. Операнд має бути типів BOOL, BYTE, WORD або DWORD.

Приклад:

```
LDN     boo_var2    (* результат := NOT ( булева змінна ) *)
```

Модифікатор 'C' (condition) вказує на те, що відповідна інструкція має бути виконана тільки за умови, що поточний результат має значення TRUE (відмінне від 0 для небулевих значень).

Приклад:

*JMPC test1 (*виконується тільки, коли результат:= TRUE*)*

Модифікатор 'C' може комбінуватися з модифікатором 'N'. Тоді інструкція виконується тільки за умови, що поточний результат має значення FALSE (або 0 для не булевих значень).

Приклад:

*CALCN trigb1 (*виконується тільки коли результат:= FALSE*)*

Модифікатор відкриваюча дужка '(' вказує на те, що оцінка інструкції має бути затримана до тих пір, поки не зустрінеться оператор закриваюча дужка ')', який виконує затриману операцію.

Приклад:

```
(* res := a1 + (a2 * (a3 - a4) * a5) + a6; *)
LD    a1 (* результат := a1; *)
ADD(  a2 (* затримане складання ADD - результат := a2; *)
MUL(  a3 (* затримане множення MUL - результат := a3; *)
SUB   a4 (* результат := a3 - a4; *)
)        (* виконання затриманого множ. MUL - результат:=a2*(a3-a4); *)
MUL   a5 (* результат := a2 * (a3 - a4) * a5; *)
)        (* виконання затриманого складання ADD *)
         (* результат := a1 + (a2 * (a3 - a4) * a5); *)
ADD   a6 (* результат := a1 + (a2 * (a3 - a4) * a5) + a6; *)
ST    res  (* збереження поточного результату в змінній res *)
```

Інструкції може передувати мітка з двокрапкою (':'). Мітка використовується при необхідності переводу виконання програми на вказану інструкцію. Щоб перехід відбувся, необхідно використати операцію стрибка, а в якості операнда вказати ім’я мітки. При цьому:

- ім'я не може бути довшим 16 символів;
- першим символом повинна бути буква;
- подальшими символами можуть бути букви, цифри або символ підкреслення ('_').

У одній IL-програмі те саме ім'я не може бути використане для позначення більше однієї мітки. Ім'я мітки може співпадати з ім'ям змінної.

Операнд і поточний вміст акумулятора (IL-регістра) повинні мати однаковий тип даних. Якщо є потреба опрацювати операнди різних типів даних, то спочатку здійснюється перетворення типів за допомогою наступних операторів:

ANY_TO_DINT-Перетворити будь-яку змінну в цілу;
ANY_TO_REAL-Перетворити будь-яку змінну в дійсну;
ANY_TO_SINT- Перетворить будь-яку змінну в коротку цілу;
ANY_TO_STRING-Перетворити будь-яку змінну в рядок;
ANY_TO_TIME-Перетворить будь-яку змінну в часову.

Виключенням є тип даних TIME разом з арифметичними операторами MUL і DIV. Ці оператори дозволяють опрацьовувати операнд типу даних TIME разом з операндом типу даних ANY_NUM. У цьому випадку результат цих команд має тип даних TIME.

Приклад:

Перевірити значення цілочислового селектора (0, 1 або 2) і за допомогою оператора JMPC встановити булевий вихід, що дорівнює 0.

```
JMPex:  LD          selector    (* селектор - 0 або 1 або 2 *)
        ANY_TO_BOOL             (* перетворення у Boolean *)
        JMPC        test1       (* if selector = 0 then *)
        LD          true
        ST          bo0         (* bo0 := true *)
        JMP         JMPend      (* кінець програми *)
test1:  LD          selector
        SUB         1           (* зменшити селектор: тепер 0 або1 *)
        ANY_TO_BOOL             (* перетворення у Boolean *)
        JMPC        test2       (* if selector = 0 then *)
        LD          true
        ST          bo1         (* bo1 := true *)
        JMP         JMPend      (* кінець програми *)
test2:  LD          true        (* остання можливість *)
        ST          bo2         (* bo2 := true *)

JMPend:                         (* кінець IL програми *)
```

При IL-програмуванні можна використовувати функції та функціональні блоки.

Виклик функції в IL (написаної на будь-якій з мов IL, ST, LD, FBD або 'C') використовує її ім'я як оператора. При цьому перший параметр виклику має бути збережений у поточному результаті перед викликом. Решта параметрів записується в полі операнда, розділяючись комами. Після виконання функції, значення що нею повертається, запам'ятовується у поточному результаті IL.

Приклад:

```
(*Визивна функція : перетворити цілочислове значення в часове *)
Main:  LD          bi0
       MYFUNC      (* виклик функції для отримання цілочислового
```

```
                    значення *)
ST                  (* результат := значення, що повертається
                    функцією *)
GT                  (* перевірка переповнення значення*)
RETC                (* повернення по переповненню *)
LD
MUL                 (* перетворити секунди в мілісекунди *)
ANY_TO_TIME         (* перетворити в таймер *)
ST                  (* запам'ятати перетворене значення в таймері *)
```

Функція, що викликається з ім'ям MYFUNC, оцінює цілочислове значення, задане як двійкове число трьох булевих входів: in0, in1, in2 - три вхідні параметри функції:

```
LD                  in2
ANY_TO_DINT                     (* result = ANY_TO_DINT (in2); *)
MUL                 2           (* result := 2 * ANY_TO_DINT (in2); *)
ST                  temporary   (* temporary := result *)
LD                  in1
ANY_TO_DINT
ADD                 temporary   (* result := 2 * ANY_TO_DINT (in2) +
                                ANY_TO_DINT (in1); *)
MUL                 2           (* result := 4 * ANY_TO_DINT (in2) + 2 *
                                ANY_TO_DINT (in1); *)
ST                  temporary   (* temporary := result *)
LD                  in0
ANY_TO_DINT
ADD                 temporary   (* result := 4 * ANY_TO_DINT (in2) + 2 *
                                ANY_TO_DINT (in1) + ANY_TO_DINT (in0); *)
```

Виклик функціонального блока здійснюється за допомогою оператора CAL, при цьому в якості операнда використовується ім'я екземпляра функціонального блока.

Вхідні параметри блока повинні бути призначені до виклику за допомогою послідовності дій LD/ST. Вихідні параметри, якщо використовуються, відомі[4,7-9].

Приклад:

Викликати функціональний блок SR : SR1 є екземпляром SR

```
LD      auto_mode
AND     start_cmd
ST      SR1.set1
LD      stop_cmd
```

ST	*SR1.reset*
CAL	*SR1*
LD	*SR1.Q1*
ST	*command*

1.2.4.2. **Приклад програмування IL-мовою**

Як приклад програмування IL – мовою розробимо програму керування роботою двох двигунів згідно з алгоритмом наведеним у параграфі 1.2.2.2.

Проектний код IL-мовою має вигляд:

```
(*Програмування запуску двигуна_1*)
LD  start_1
S   motor_1
(*Програмування таймера роботи двигуна_1*)
LD  motor_1
ST  M_TIME_1.IN
LD  T#5s
ST  M_TIME_1.PT
CAL M_TIME_1
LD  M_TIME_1.Q
ST  time_1
LD  time_1
R   motor_1
S   motor_2
LD  M_TIME_1.ET
ST  act_time_1
(*Програмування лічильника натисків кнопки запуску двигуна_2*)
LD  start_2
ST  MOT_COUNT.CU
LD  motor_2
ST  MOT_COUNT.RESET
LD  int#2
ST  MOT_COUNT.PV
CAL MOT_COUNT
LD  MOT_COUNT.Q
S   motor_2
LD  MOT_COUNT.CV
ST  pressed
```

```
ST  pressed
(*Програмування таймера роботи двигуна_2*)
LD  motor_2
ST  M_TIME_2.IN
LD  t#10s
ST  M_TIME_2.PT
CAL M_TIME_2
LD  M_TIME_2.Q
ST  time_2
LD  time_2
R   motor_2
S   motor_1
LD  M_TIME_2.ET
ST  act_time_2
(*Програмування зупинення двигуна_1 і двигуна_2*)
LD  stop
R   motor_1
R   motor_2
(*Програмування підрахунку циклів роботи двигунів та їх
зупинення*)
LD  time_1
OR  time_2
ST  Cycle_Count.CU
LD  Reset
ST  Cycle_Count.RESET
LD  int#10
ST  Cycle_Count.PV
CAL Cycle_Count
LD  Cycle_Count.Q
R   motor_1
R   motor_2
LD  Cycle_Count.CV
ST  Motor_Cycles
```

Тут:

START_1 і START_2 - кнопки початкового пуску двигунів;

MOTOR_1 і MOTOR_2 – перший і другий двигуни;

M_TIME_1 і M_TIME_2 – екземпляри функціонального блока таймера TON, які відлічують час роботи двигунів;

MOT_COUNT – екземпляр функціонального блока лічильника CTU, який підраховує кількість натисків на кнопку START_2;

CYCLE_COUNT(*CYCLE_COUNT:=count+1;)*– функція користувача, яка підраховує кількість активізацій двигунів;

STOP – кнопка непередбаченої зупинки двигунів.

Контрольні запитання

1. Який вигляд має синтаксис IL-інструкції?
2. Що визначає оператор в IL-інструкції?
3. Як стандарт IEC 61131-3 трактує вираз – «поточний результат»?
4. Що таке операнд і модифікатор?
5. Для чого при створені IL-програм використовуються дужки?
6. З якого оператора починається список команд і яким закінчується?
7. Який опис мають оператори S і R?
8. Чим відрізняються оператори GT і GE?
9. Який оператор забезпечує виклик функціонального блока?
10. У яких випадках використовуються модифікатори “C” і “N”?
11. Коли при програмуванні використовується мітка?
12. У яких випадках потрібно перетворювати типи даних?
13. Чим характеризується тип даних TIME?
14. Як при програмуванні викликаються функції та функціональні блоки?

1.2.5. **ST- мова**

Мова структурованого тексту ST (Structured Text) - це мова, яка відноситься до класу мов високого рівня, схожа на Паскаль і орієнтована на програмістів. У цій мові підтримуються масиви (також і багатовимірні), контроль перетворення типів, є такі конструкції, як DO-WHILE, REPEAT-UNTIL, FOR-TO-DO, IF-THEN-ELSE, CASE-OF, а також інші зрозумілі будь-якому програмісту оператори. Самостійно ця мова використовується лише для створення складних процедур, які не можуть бути легко виражені за допомогою графічних мов.

Зазвичай мова структурованого тексту (ST) використовується тільки для програмування послідовних кроків і переходів SFC-мови. Ця мова має походження від мови програмування Графсет (Grafcet) французької фірми Telemechanique - Groupe Schneider.

1.2.5.1. **Синтаксис** ST-**мови**

Основою ST-програми є вирази. Вираз складається із операндів (змінних, констант, функцій і функціональних блоків), розділених операторами. Результат обчислення виразу, як і у Паскалі, присвоюється змінній за допомогою оператора «:=». Кожний вираз обов'язково закінчується крапкою з комою «;». Стандартні оператори у ST- виразах мають символьний вигляд, наприклад математичні дії: +, −, *, /, операції порівняння: <, >, <=, >= і т. ін. Окрім операторів, елементи виразу для кращого сприйняття можна відділяти пропусками і табуляціями . У текст можуть бути введені коментарі. Коментар повинен починатися символами "(*",а закінчуватися символами"*)". Коментарі можна вставляти там, де допустимі пасивні роздільники (пропуски і табуляції).

Обчислення виразів виконується відповідно до правил пріоритету. Оператор з найвищим приоритетом виконується першим, оператор з нижчим пріоритетом –другим і т.д., доки не будуть виконані усі оператори. Оператори з однаковим пріоритетом виконуються зліва направо. У табл. 1.5 приведено список ST- операторів, розташованих у порядку пріоритету.

Таблиця 1.5. Список ST- операторів

Оператор	Приклад	Значення прикладу	Опис	Пріоритет
( )	(1+2)*(4+5)	45	Круглі дужки	Най-вищий
**	3.0**4	81.0	Піднесення	
-	-10	-10	Заміна знаку	
NOT	NOT TRUE	False	Числове доповнення	
*	10*3	30	Множення	
/	6/2	3	Ділення	
MOD	17MOD 10	7	За модулем	
+	2+3	5	Додавання	
-	4-2	2	Віднімання	
<,>,<=,>=	4>12	False	Порівняння	
=	T#26h=T#1d2h	True	Рівність	
<>	8<>16	True	Нерівність	
&, AND	TRUE&FALSE	False	Логічне „І”	
OR	TRUE OR FALSE	True	Логічне „АБО”	
XOR	TRUE XOR FALSE	True	Логічне„ВиключнеАБО”	Най-нижчий

Для кожного окремого виразу, об'єднуючого операнди з одним оператором тип операндів має бути одним і тим же. При цьому цей окремий вираз має той самий тип, що і його операнди, і може бути використаним у складнішому виразі.

Дужки використовуються для того, щоб відділити складові виразу і явно визначити пріоритетність операцій. Коли в складному виразі немає дужок, послідовність операцій задається неявно пріоритетністю операторів ST.

Основні оператори ST-мови :

- присвоєння;
- виклик підпрограми або функції;
- виклик функціонального блока;
- оператори вибору (IF, THEN, ELSE, CASE...);
- оператори циклу (FOR, WHILE, REPEAT...);
- управляючі оператори (RETURN, EXIT...);

- спеціальні оператори для зв'язку з такими мовами як SFC.

Оператор присвоєння є найпростішим оператором, який частіше всього застосовується при програмуванні. Він призначений для обчислення нового значення певної змінної, а також визначення значення функції, що повертається.

Перед оператором присвоєння «:=» знаходиться операнд (змінна або адреса), якому присвоюється значення виразу, що стоїть після оператора присвоєння. Синтаксис оператора присвоєння має традиційний для мов програмування вигляд:

<змінна> := <будь- який вираз >;

Змінна має бути внутрішньою або вихідною.

Приклад:

*Var1: = Var2 * 10;*

Після виконання цієї операції Var1 приймає значення вдесятеро більше, ніж Var2.

Виклик функції може бути використаний у будь-якому виразі. Щоб викликати функцію, необхідно знати її ім’я і параметри, якими вона характеризується. Функції, що викликаються, мають бути написані мовами IEC-стандарту або мовою "C".

Синтаксис виклику функції такий:

<змінна> := *<ім’я функції> (<nap1>, ... <napN>*);

Тут *(<nap1>, ... <napN>*)- параметри функції.

Змінна набуває значення функції, що повертається. Тип значення і параметри виклику повинні відповідати інтерфейсу, визначеному для функції.

Приклад : виклик IEC функцій

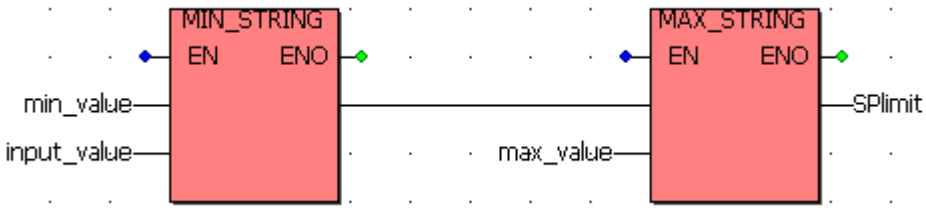
(Головна ST програма *)*

(набуває ціле значення і перетворює його на обмежене часове значення *)*

ana_timeprog := SPlimit (tprog_cmd);

*appl_timer := ANY_TO_TIME (ana_timeprog * 100);*

(FBD функція, що викликається, з ім'ям 'SPlimit' *)*



Виклик функціонального блока із стандартної бібліотеки або бібліотеки користувача здійснюється після оголошення його унікальної копії (екземпляра), визначення імені копії і типу блока. Функціональний блок викликається за допомогою імені екземпляра і списку вхідних параметрів з присвоюванням даних у круглих дужках.

Синтаксис виклику функціонального блока:

(* викликати функціональний блок *)

< ім'я блока> (<пар1>, <пар2> ...);

(* одержати параметри, що повертаються *)

<результат> := *<ім'я блока>*. *<возвр_пар1>*;

...

<результат> := *<ім'я блока>*. *<возвр_парN>*;

Приклад:

(Виклик функціонального блоку ST- програмою *)*

(оголосити екземпляр функціонального блоку в словнику: *)*

(trigb1 : блок R_TRIG - визначення переднього фронту *)*

(активізація функціонального блоку з мови ST *)*

trigb1 (b1);

(доступ до параметрів, що повертаються *)*

If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;

Оператор циклу з передумовою **WHILE** (*ДОКИ, У ТОЙ ЧАС ЯК*) забезпечує повторне виконання виразу, доки булева умова ***перед*** виконанням кожного циклу має значення TRUE. У протилежному випадку, коли значенням умови є FALSE, виконання оператора циклу закінчується. Тіло циклу зовсім не виконується, якщо булева умова із самого початку має значення FALSE. Якщо ж булева умова ніколи не приймає значення FALSE, тіло циклу виконується безкінцево. Щоб такого не трапилося, значення змінної булевої умови в тілі циклу обов'язково має змінюватися, наприклад, шляхом інкременту або декременту лічильника.

Синтаксис WHILE має вигляд:

WHILE *<булева умова>* ***DO***

<вираз>;

<вираз>;

… ;

END_WHILE

Приклад:

```
WHILE Counter<>0 DO
    Var1:=Var1*2;
    Counter:=Counter-1;
END_WHILE
```

У цьому прикладі доки Counter <>0 змінна Var1 подвоюється.

Оператор циклу з постумовою **REPEAT (***ПОВТОРИТИ***)** аналогічний попередньому оператору WHILE тільки його умова перевіряється ***після***

виконання чергової ітерації, тобто гарантується хоча б одноразове виконання циклу. Окрім того, критерієм завершення циклу є рівність виразу константі TRUE. Якщо вираз дорівнює FALSE, то цикл повторюється Щоб не було безкінцевого виконання циклу, значення змінній булевої умови у тілі циклу, як і у попередньому випадку, обов'язково має змінюватися.

Синтаксис REPEAT:

REPEAT

<вираз>;

<вираз>;

… ;

UNTIL** <булева умова>**;

END_REPEAT;

Приклад:

REPEAT
Proc1(x, y+1);
I:=i-1
UNTIL
I=0

END_REPEAT;

Цикл виконується до тих пір, доки "і" не стане більше нуля.

Оператор циклу з параметром **FOR** *(ДЛЯ)* використовується, коли кількість повторювань циклу може бути визначеною перед його початком. Таким чином, перед виконанням циклу лічильник отримує початкове значення і тіло циклу повторюється, доки значення лічильника збільшуючись не перевищить кінцеве значення.

Синтаксис FOR:

***FOR**<лічильник>:=<початкове значення>**TO**<кінцеве значення>*
{BY**<крок>**}DO
<вираз>;
*<вираз>**;***
… ;
END_FOR;

Частина конструкції у фігурних дужках не обов'язкова, вона визначає крок приросту лічильника. За умовчанням лічильник збільшується на одиницю у кожній ітерації. Лічильником може бути змінна будь-якого цілого типу.

Крок зміни лічильника ітерацій може бути і від'ємний. У цьому випадку початкове значення має бути більше за кінцеве. Тоді цикл закінчується, коли значення лічильника стає менше кінцевого.

Приклад:

```
Varl := 0
FOR Counter := 1 TO 5 DO
   Varl := Varl *2;
END_FOR
Erg:= Var1
```

Даний цикл буде виконуватися 5 разів і Varl матиме значення 32.

Використовуючи умовний оператор **IF** *(ЯКЩО)*, можна перевірити умову і залежно від цієї умови виконати будь-які дії. У якості умови використовується значення логічного виразу.

Синтаксис IF:

IF <*булева умоваIF*> **THEN**
<*виразIF*>;
{ELSIF <*булева умова ELSIF1*> **THEN**
<ELSIF- вираз1>;
… ;
ELSIF <*булева умоваELSIF n*> **THEN**
<ELSIF_вираз n>
ELSE
<ELSE-вираз>**}**
END_IF;

Якщо <*булева умоваIF*> TRUE, тоді виконується вираз першої групи <*виразIF*>. Інші вирази пропускаються, альтернативні умови не перевіряються.

У випадку, коли <*булева умоваIF*> є FALSE, послідовно перевіряється решта логічних виразів. Перша знайдена умова TRUE призведе до виконання виразів, що стоять після цієї логічної умови, до наступного ELSIF або ELSE.

ELSIF може бути декілька або зовсім не бути.

Коли усі логічні вирази помилкові, то виконуються інструкції, що стоять після ELSE, якщо вона є. Якщо ELSE немає, то нічого не виконується.

Частина конструкції у фігурних дужках не обов'язкова.

У найпростішому випадку оператор IF містить тільки одну умову.

Приклад:

```
IF temp < 20
THEN heating_on: = TRUE;
ELSE heating_on: = FALSE;
END_IF
```

У даному прикладі нагрівання (heating) вмикається, коли температура знизиться нижче 20 градусів, інакше воно залишиться вимкненим.

За допомогою оператора множинного вибору CASE *(У ВИПАДКУ)* можна виконати один з декількох виразів. Вибір здійснюється відповідно до значення поточного обчислення цілочислової змінної або виразу, що записані після оператора CASE .

Синтаксис оператора множинного вибору має вигляд:

CASE < *цілочислова змінна* > **OF**
< *значення 1*>:< *вираз 1*> ;
< *значення 2* > , < *значення 3* >,< *значення4* >:< *вираз 2* > ;
< *значення 5* > .. < *значення10* >:< *вираз 3* >;
…
< *значення n* >:<*вираз n* >
ELSE
< *вираз* ELSE> ;
END_CASE

Оператор CASE виконується відповідно до наступних правил:

- У випадку <цілочислова змінна > має < значення і >, то виконується < вираз і>, тобто <значення і>:< вираз і> . Інші умови не аналізуються.
- У випадку <цілочислова змінна > не дорівнює жодному з вказаних значень, то виконується <вираз ELSE>. Цей вираз не є обов'язковим.
- Щоб той самий вираз виконувався при різних значеннях <цілочислової змінної >, необхідно перелічити ці значення через кому: < значення 2 >,< значення 3 >,< значення4 >:< вираз2 > ;
- Щоб той самий вираз виконувався для певного діапазону значень <цілочислової змінної >, необхідно вказати початкове та кінцеве значення і відокремити їх двома крапками: < значення 5 > .. < значення10 >:< вираз 3 >.

Значеннями оператору множинного вибору CASE можуть бути тільки цілі константи, змінні використовувати не можна. Однакові значення в альтернативах вибору задавати не можна , навидь у діапазонах.

Приклад:

```
CASE INT1 OF
1,5: BOOL1:=TRUE;
        BOOL3:=FALSE;
2: BOOL2:=FALSE;
        BOOL3:=TRUE;
10..20: BOOL1:=TRUE;
ELSE
        BOOL1:= NOT BOOL1;
        BOOL2:= BOOL1 OR BOOL2;
END_CASE;
```

Якщо INT1 дорівнює 1 або 5, то BOOL1:=TRUE; і BOOL3:=FALSE. Якщо INT1 дорівнює 2, то BOOL2:=FALSE; BOOL3:=TRUE. Якщо INT1 дорівнює значенню, що знаходиться поміж 10 і 20 включно, то BOOL1:=TRUE. Якщо ж жодної умови не виконується, то BOOL1:= NOT BOOL1; BOOL2:= BOOL1 OR BOOL2;

Якщо в циклах FOR, WHILE, REPEAT є оператор **EXIT** *(ВИХІД)*, то цикл закінчує свою роботу незалежно від значення умови виходу. EXIT

звичайно використовується з оператором IF усередині блоку FOR, WHILE або REPEAT.

Приклад:

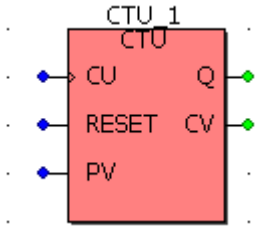
```
length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
    code := ascii (message, index);
    IF (code = searched_char) THEN
        found := YES;
        EXIT;
    END_IF;
END_FOR;
```

У наведеній програмі здійснюється пошук символу в рядку.

Оператор **RETURN** *(ПОВЕРНЕННЯ)* припиняє виконання поточної програми і здійснює негайне повернення з POU. У блоці дій SFC-програми оператор RETURN позначає кінець виконання тільки даного блоку[4,7-9].

Приклад:

Програма, яка реалізує стандартний функціональний блок лічильника,



має вигляд:

```
 If NOT (CU) then
     Q := false;
     CV := 0;
     RETURN; (* закінчити програму *)
 end_if;
 if RESET then
     CV := 0;
 else
     if (CV < PV) then
         CV := CV + 1;
     end_if;
end_if;
Q := (CV >= PV);
```

1.2.5.2. **Приклад програмування ST – мовою**

Як приклад програмування ST – мовою розробимо програму керування роботою двох двигунів згідно з алгоритмом, наведеним у параграфі 1.2.2.2.

Проектний код ST-мовою має вигляд:

```
(*Програмування запуску двигуна_1*)
RS_1(SET:=(start_1 AND NOT motor_2)OR time_2,
RESET1:=stop OR time_1 OR autostop);
motor_1:=RS_1.Q1;
(*Програмування таймера роботи двигуна_1*)
M_TIME_1(IN:=motor_1,PT:=t#5s);
time_1:=M_TIME_1.Q;
act_time_1:=M_TIME_1.ET;
(*Програмування лічильника кількості натисків кнопки запуску
двигуна_2*)
MOT_COUNT(CU:=start_2,RESET:=motor_2,PV:=int#2);
out:=MOT_COUNT.Q;
Pressed:=MOT_COUNT.CV;
RS_2(SET:=(out AND NOT motor_1)OR time_1,
RESET1:=stop OR time_2 OR autostop);
motor_2:=RS_2.Q1;
(*Програмування таймера роботи двигуна_2*)
M_TIME_2(IN:=motor_2,PT:=t#10s);
time_2:=M_TIME_2.Q;
act_time_2:=M_TIME_2.ET;
(*Програмування лічильника циклів роботи двигуна та його
зупинення*)
if time_1 OR time_2 then
COUNTER:=COUNTER+1;
end_if;
if COUNTER>=10 then
autostop:=INT_TO_BOOL (COUNTER);
end_if;
```

Тут:

START_1 і START_2 - кнопки початкового пуску двигунів;

MOTOR_1 і MOTOR_2 – перший і другий двигуни;

M_TIME_1 і M_TIME_2 – екземпляри функціонального блока таймера TON, які відлічують час роботи двигунів;

MOT_COUNT – екземпляр функціонального блока лічильника CTU, який підраховує кількість натисків на кнопку START_2;

RS_1 і RS_2 – екземпляри функціонального блока детектора імпульсу переднього фронту R_TRIG.

COUNTER– лічильник циклів роботи двигунів;

STOP – кнопка непередбаченої зупинки двигунів.

Контрольні запитання

1. Що є основою ST-програми?
2. Який вигляд у ST-програмі мають стандартні оператори?
3. Як у ST-програмі записуються коментарі?
4. Як виконуються оператори з однаковим пріоритетом?
5. Який оператор має найвищий пріоритет?
6. Який синтаксис має оператор присвоєння?
7. Як у ST-програмі викликаються функції?
8. Як здійснюється виклик функціонального блока?
9. Коли виконується оператор циклу з передумовою WHILE?
10. Який має вигляд синтаксис WHILE?
11. Чим відрізняється оператор REPEAT від WHILE?
12. Який має вигляд синтаксис REPEAT?
13. Який синтаксис має оператор FOR і коли він використовується?
14. Який синтаксис має оператор IF і коли він використовується?
15. За допомогою якого оператора можна виконати один з декількох виразів?
16. Коли використовують оператори EXIT і RETURN?

1.2.6. SFC-мова

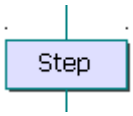
Мова послідовних функціональних схем SFC (Sequential Function Chart) є найважливішою із всього сімейства стандартних мов. Вона використовується для опису алгоритму у вигляді набору зв'язаних пар – крок (step) і перехід (transition). Крок є набором операцій над змінними. Перехід – набір логічних умовних виразів, визначаючих передачу управління до наступної пари крок-перехід. На вигляд опис SFC-мовою нагадує добре відомі логічні блок-схеми алгоритмів. Строго кажучи, SFC-мова не є мовою програмування. Це засіб проектування прикладного програмного забезпечення, що складається з комплексу великої кількості одиниць: програм, функціональних блоків, функцій. Основними перевагами цієї мови є можливість описувати паралельно-послідовні процеси в наочній і компактній формі й автоматичне отримання за цим описом (за наявності транслятора) керуючих програм. SFC-мова дуже зручна для програмування стадійних процесів, систем дозування та бізнес-додатків. Проте SFC-мова не має засобів для опису кроків і переходів. Вони можуть бути виражені тільки засобами інших мов стандарту IEC.

Теоретичною основою SFC-мови є мережі Петрі, за допомогою яких в теорії кінцевих автоматів описується стан складних процесів управління і, які складають основу французької мови програмування контролерів Grafcet.

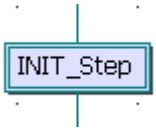
1.2.6.1. Синтаксис SFC-мови

Основними компонентами SFC-мови є кроки і початкові кроки, переходи, орієнтовані сполучення, стрибок на крок, сходження і розбіжності.

Крок зображається прямокутником. Кожному кроку привласнюється ім'я, написане усередині прямокутника.

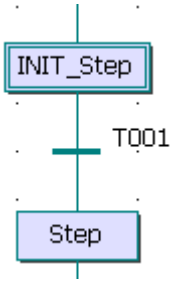


Початковий крок позначається графічним символом з подвійною рамкою.



Кроки не можуть слідувати один за одним. Обов'язково поміж ними має бути перехід.

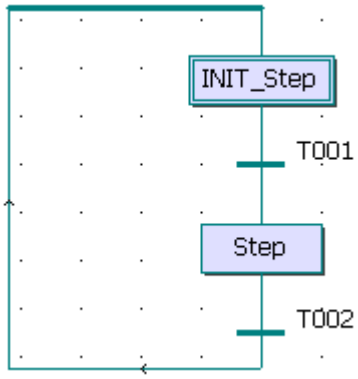
Переходи зображаються маленькими горизонтальними смужками, які перетинають лінії сполучення. Кожному переходу присвоюється ім'я, яке записується поряд.



Перехід виконується, коли перехід дозволено (відповідний йому крок активний) и при цьому умова переходу має значення TRUE.

Для сполучення кроків і переходів використовуються *орієнтовані лінії*. Коли орієнтація не задана явно, сполучення орієнтовано зверху вниз.

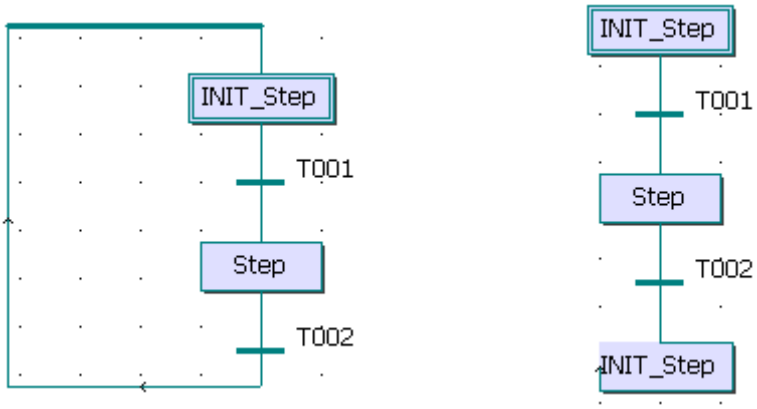
Явна орієнтація від переходу T002 до кроку INIT_Step



Неявна орієнтація від кроку INIT_Step до переходу T001

Щоб позначити лінію сполучення від переходу до будь-якого кроку, не малюючи лінії, використовується символ *стрибка* з іменем кроку призначення. У той же час за допомогою символу стрибка не можна позначити лінію сполучення від кроку до переходу.

Наступні схеми еквівалентні:



Дії у SFC- програмі зображуються у вигляді прямокутників, розташованих праворуч від символу кроку і приєднаних до них лінією. У лівій частині прямокутника знаходиться класифікатор, можливо, з константою часу, а права містить ім'я дії або логічній змінній. Кроки можуть мати декілька дій.

Дії кроків описуються окремо від них і не належать певному кроку, тому та сама дія може багаторазово використовуватися в межах одного POU.

Класифікатор визначає спосіб впливу активного кроку на дію, табл.1.6.

Таблиця 1.6. Класифікатори SFC-мови

Класифі-катор	Дія	Коментар
N (Non-stored)	Що не зберіга-ється	Дія виконується у кожному робочому циклі протягом активності кроку
R (Reset)	Позачерговий скид	Дія деактивується
S (Stored)	Що зберігається	Дія активна доки немає скиду. Дія продовжує виконання у кожному циклі, навіть тоді, коли крок вже не активний
L (time Limited)	Що обмежена за часом	Дія активна протягом вказаного часу, але не триваліше часу активності кроку
D (Delayed)	Що відкладена	Дія активується через заданий час після активації кроку і залишається бути активною, доки крок є активним
P (Pulse)	Імпульс	Дія виконується один раз при активації і другий раз після деактивації кроку
SD (Stored and timeDelayed)	Що зберігається і відкладена	Дія активується через заданий час після активації кроку, навіть якщо крок вже не активний
DS (Delayed and Stored)	Що відкладена і зберігається	Дія активується через заданий час після активації кроку і залишається активною до скиду
SL (Stored and Limited)	Що зберігається і обмежена за часом	Дія активується разом з кроком і залишається бути активною заданий час незалежно від активності кроку

Класифікатори L, D, SD, DS і SL потребують установлення константи у форматі TIME.

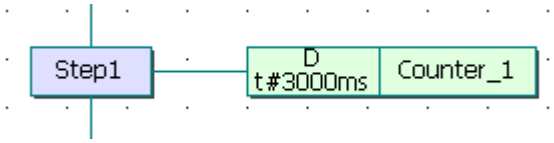
Синтаксис дій на ST та IL- мовах має вигляд:

Action(Класифікатор):

*(*Оператори ST-мови або блок команд IL-мовою*)*

End _Action;

Приклад:



Тут дія Counter_1 написана IL-мовою :

Action(D):

LD Count

ADD INT#1

ST Count

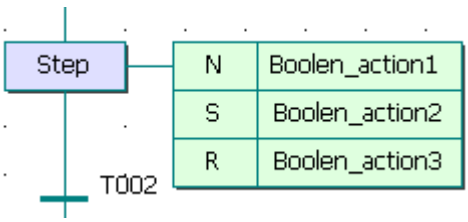
End_Action;

Опис дій, що виконуються протягом активності кроку, здійснюється з використанням текстових властивостей мови SFC і інших мов міжнародного стандарту. Основні типи дій:

- булеві дії із класифікаторами: 'Set', 'Reset' або 'Non-stored';
- імпульсні дії або дії, що не зберігаються із класифікаторами відповідно 'Pulse' і 'Non-Stored';
- SFC дії (управління дочірніми SFC програмами) із класифікаторами 'Set', 'Reset' або 'Non-Stored'.
- Виклик функцій, функціональних блоків і підпрограм, створених будь- якою мовою, окрім SFC.

Булеві дії присвоюють вихідним (OUTPUT) або внутрішнім (MEMORY) логічним змінним значення активності кроку кожного разу, коли поточний крок стає активним. Синтаксис основних логічних дій такий:

N	boolen var	присвоює змінній boolen var сигнал активності кроку.
S	boolen vari	присвоює змінній boolen var значення TRUE, коли сигнал активності кроку стає TRUE.
R	boolen vari	присвоює змінній boolen var значення FALSE, коли сигнал активності кроку стає TRUE.



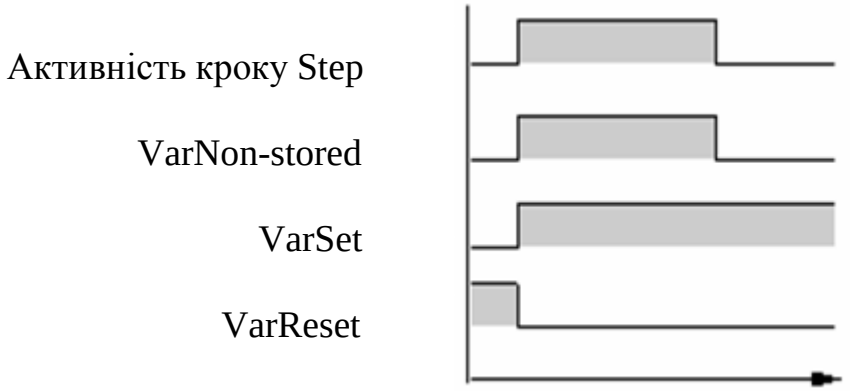
Boolen_actions:

Action(N): *Action(S):* *Action(R):*

VarNon-stored; VarSet; VarReset;

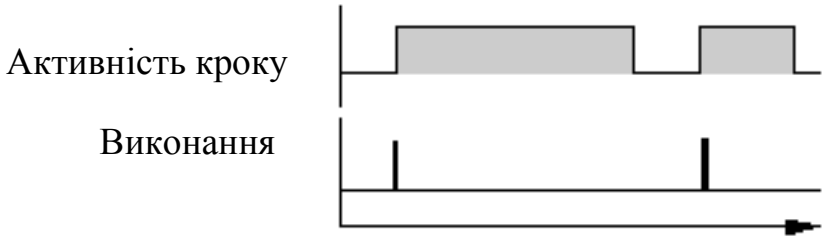
End_Action; *End_Action;* *End_Action;*

Часова діаграма зміни значень булевих змінних VarNon-stored; VarSet; VarReset; має вигляд:

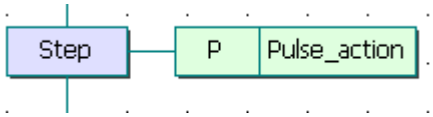


Імпульсні дії **(**Pulse) - це список ST, IL – інструкцій або LD- схем , які виконуються тільки одного разу при активізації кроку .

Часова діаграма імпульсної дії із класифікатором “P” має вигляд:



Приклад:



Pulse_action:

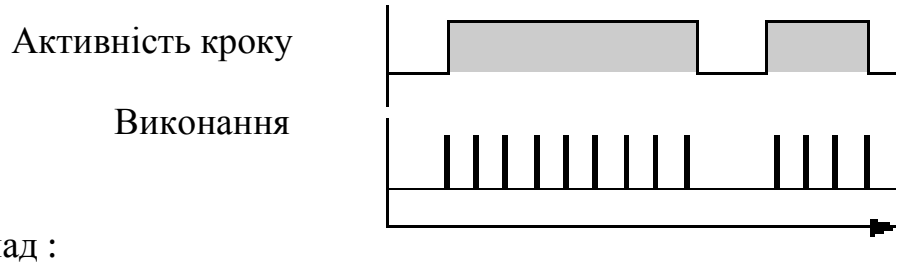
Action(P):

Lamp1: TRUE

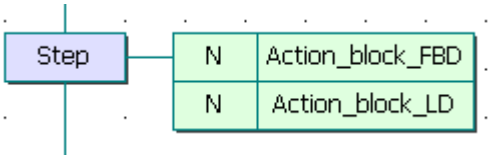
End _Action;

Дія, що не зберігається (Non-Stored), - це список ST-, IL- інструкцій або LD-, FBD- схем, які виконуються в кожному циклі протягом усього періоду активності кроку. Інструкції пишуться відповідно до синтаксису використовуваної мови.

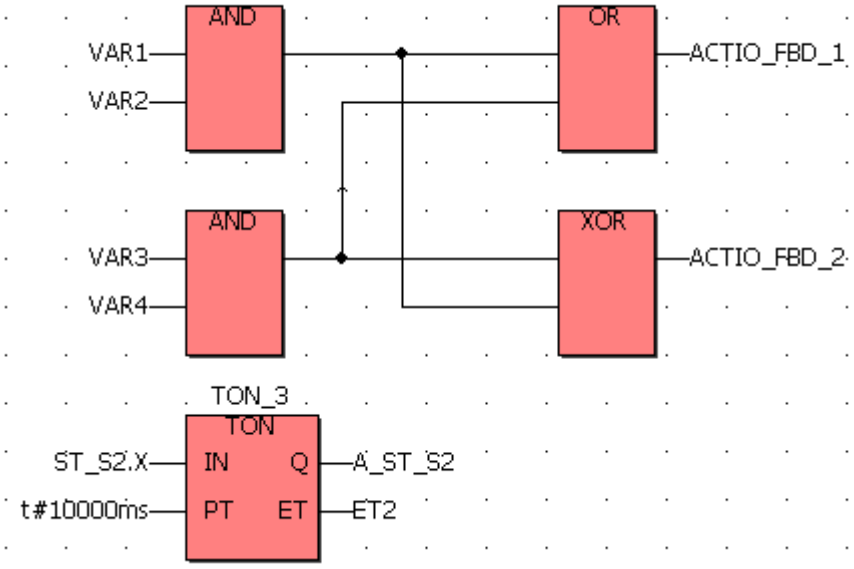
Нижче показаний результат дії, що не зберігається із класифікатором “N”:



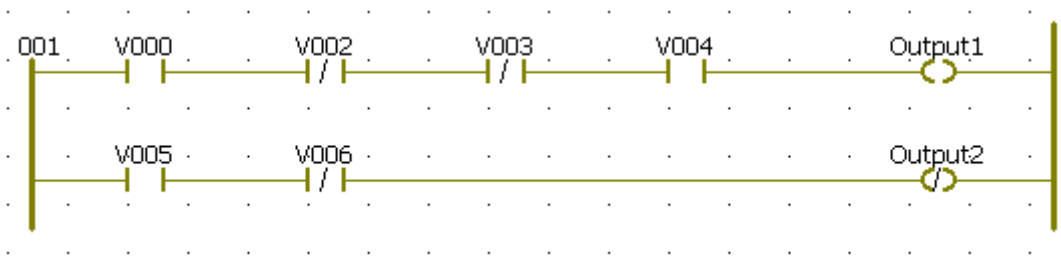
Приклад :



Action_block_FBD-мовою:



Action_block_LD-мовою:



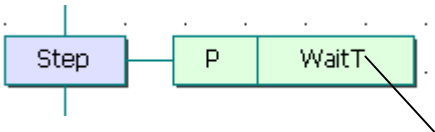
SFC дія - це дочірня послідовність SFC, що стартує і знищується відповідно до зміни сигналу активності кроку. SFC дія може мати класифікатор N (що не зберігається), S (установка), або R (скидання). Нижче приведений синтаксис основних SFC дій:

N on a child запустити дочірню послідовність, коли крок стає активним, і знищити її, коли крок стає пасивним.

S on a child запустити дочірню послідовність, коли крок стає активним, і нічого не робити, коли крок стає пасивним.

R on a child знищити дочірню послідовність, коли крок стає активним, і нічого не робити, коли крок стає пасивним.

SFC- послідовність, що визначена як дія, повинна бути дочірньою SFC-програмою поточної редагованої програми. Використання класифікаторів **S** (установка) або **R** (скидання) для SFC дій має той же самий ефект, що і використання операторів **GSTART** і **GKILL** в імпульсній дії на мові **ST**.



Ім’я дочірньої SFC програми

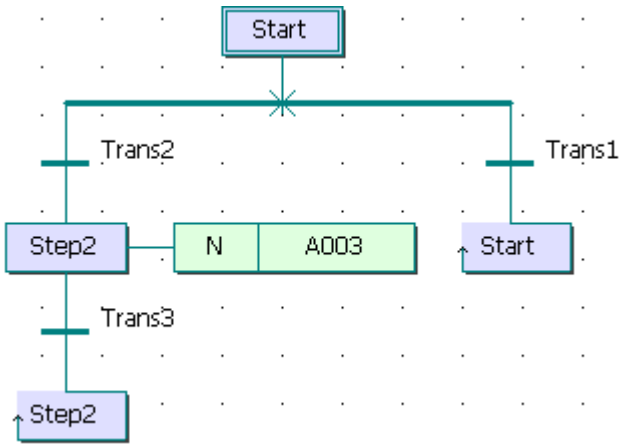
Дія WaitT:

Action (P):

GKILL(WaitT);

End_Action;

Дочірня програма WaitT:



Дія A003 :

(**Перемикання режимів**)

Action (N) :

Mode:=NewMode;

End_Action

Підпрограми, функції або функціональні блоки (написані ST-, IL-, LD- або FBD- мовами), а також 'C' функції і 'C' функціональні блоки можуть бути безпосередньо викликані з блока дій SFC на основі синтаксису мови, що використовується .

Наприклад, підпрограма, що створена ST-мовою, викликається з використанням наступного синтаксису:

Action (P): *(* або Action (N):*)*

< результат > : = < ім'я підпрограми >( );

End_Action.

Тобто

Action (P):

Init: = SPinit( ) ;

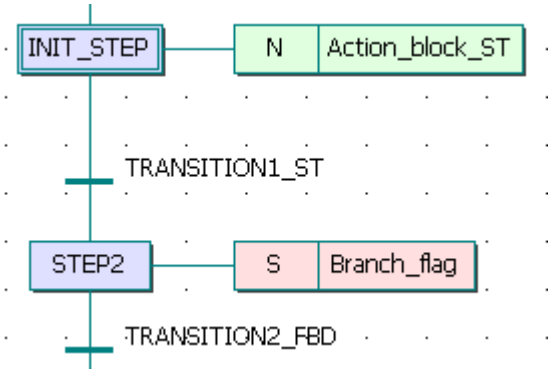
Run: = SPRun ( );

Err: = Error ( );

End_Action

Ім'я кожного переходу має логічний вираз, який є умовою переходу. Якщо до переходу вираз не приєднаний, то за умовчанням умова є істиною (TRUE).

Умови переходів створюються ST-, IL-, LD- і FBD- мовами відповідно до їх синтаксису. Наприклад:

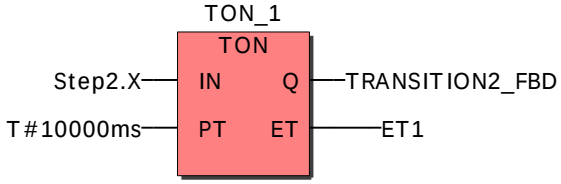


Тут умови переходів TRANSITION_ST і TRANSITION_FBD створені відповідно ST- і FBD – мовами:

TRANSITION_ST:

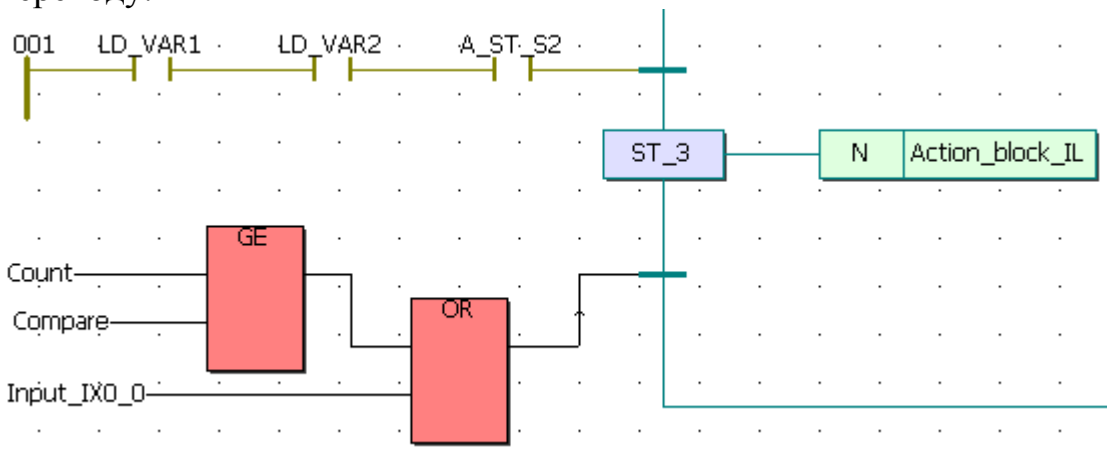
```
TON_SFC(IN:=INIT_STEP.X,PT:=T#10000ms);
Out := TON_SFC.ET;
Transition1_ST := TON_SFC.Q;
```

TRANSITION_FBD:



LD - діаграма, що описує умови переходів, складається з одиничної мережі з котушкою, значення якої представляє значення переходу.

Умови переходів LD - і FBD- мовами можуть бути прив'язані до точки переходу:



Опис умов, приєднаних до переходів, які зроблені IL-мовою, здійснюється згідно її синтаксису:

Action:
*(*блок команд IL-мовою*);*
End _Action.

Значення, яке містить поточний результат (IL регістр) в кінці IL послідовності, буде умовою, приєднаною до переходу:

Поточний результат = 0 > Умова
і або FALSE ' FALSE

Поточний результат <> > Умова
) або TRUE TRUE

Наприклад:

Action:
LD Run
ANDN Error
End _Action.

Будь-яка функція , що написана FBD-, LD-, ST- або IL-мовою, а також "C"- функція можуть бути викликані для обчислення значення умови, приєднаної до переходу, згідно з таким синтаксисом на ST і IL:

Action:
< function > ( ) ;
End _Action.

Значення, що повертається функцією, повинне бути логічним і давати результуючу умову:

значення, що повертається = **FALSE** -> умова **FALSE;**
значення, що повертається = **TRUE** -> умова **TRUE** .

SFC-програма може мати розбіжності і сходження.

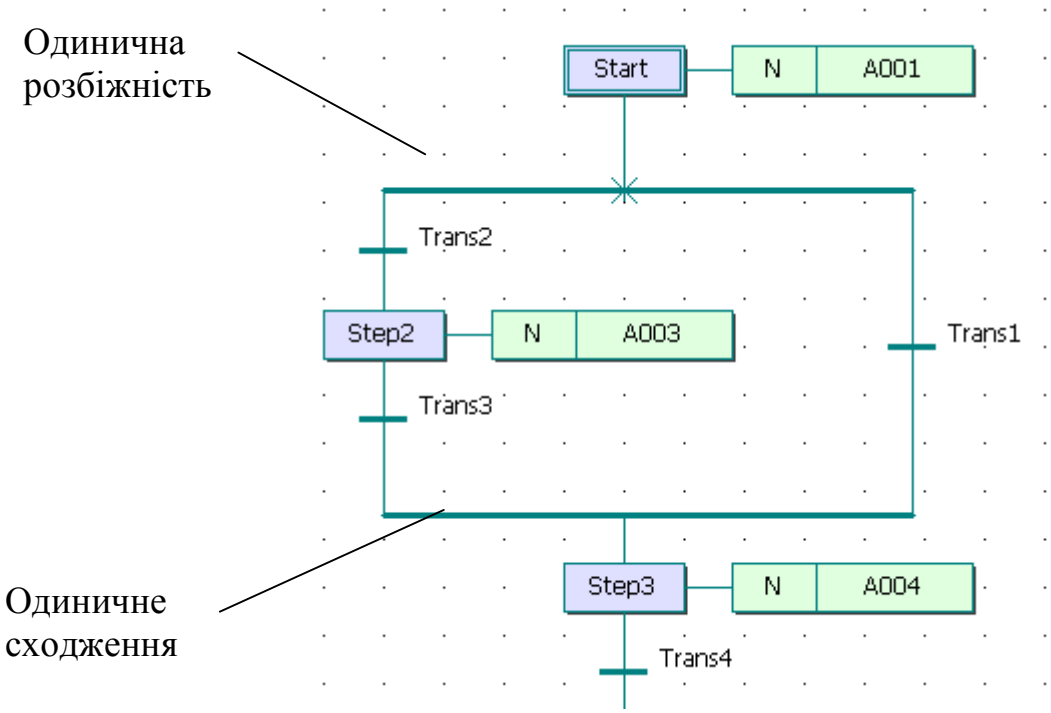
Розбіжності - це множинні сполучення від одного символу SFC (Кроку або Переходу) до багатьох інших символів.

Сходження - це множинні сполучення від більш ніж одного символу SFC до одного іншого символу. Сходження і розбіжності можуть бути одиночними або подвійними.

Одинична розбіжність (альтернативні гілки) - це множинний зв'язок від одного кроку до декількох переходів. Вона дозволяє маркеру активності пройти по одній з безлічі гілок. Перевірка альтернативних умов виконується зліва направо. Якщо вірна умова знайдена, то інші альтернативи не розглядаються. У альтернативних гілках завжди працює тільки одна, тому, коли вона закінчується, здійснюється перехід до наступного за альтернативною групою кроку.

Одиничне сходження - це множинний зв'язок від декількох переходів до одного і того ж кроку. Одиничне сходження звичайно використовується для того, щоб об'єднати декілька гілок SFC, що почалися з одиничної розбіжності.

Одиничні розбіжності і сходження позначаються одиничними горизонтальними лініями.

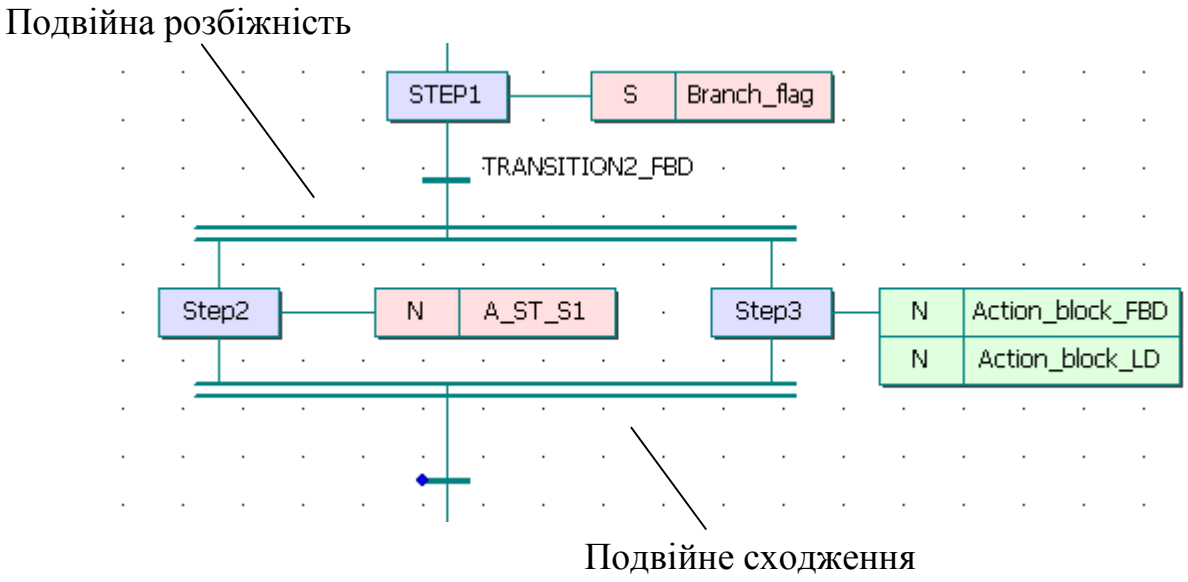


Умови, що приєднані до переходів, не є взаємовиключними. Для того, щоб програма пішла по одній гілці, треба явно визначити винятковість умови переходу.

Подвійна розбіжність - це множинний зв'язок від одного переходу до декількох кроків. Вона відповідає паралельній роботі процесу.

Подвійне сходження - це множинний зв'язок від декількох кроків до одного і того ж переходу. Подвійне сходження використовується для того, щоб об'єднати декілька SFC-гілок , що почалися в подвійній розбіжності.

Подвійні розбіжності і сходження позначаються подвійними горизонтальними лініями.



Кожна паралельна гілка починається і закінчується кроком, тобто умова входу в паралельність завжди єдина, як і єдина на всіх умова виходу.
Паралельні гілки виконуються теоретично одночасно – в одному робочому циклі, зліва направо.

При програмуванні існують п'ять динамічних поведінок SFC-мови **:**

- ***Початкова ситуація***

Початкова ситуація характеризується початковими кроками, які, за визначенням, знаходяться в активному стані на початку роботи. Принаймні, один початковий крок повинен бути в кожній SFC -програмі.

- ***Очищення переходу***

Перехід або дозволений, або заборонений. Перехід дозволений, якщо усі попередні кроки, пов'язані з відповідним символом переходу- активні, інакше, перехід заборонений. Перехід не може бути очищений, поки він не дозволений і відповідна умова переходу не є TRUE.

- ***Зміна стану активних кроків***

Очищення переходу одночасно веде до активного стану безпосередньо наступних кроків і пасивного стану безпосередньо попередніх кроків.

- ***Одночасне очищення переходів***

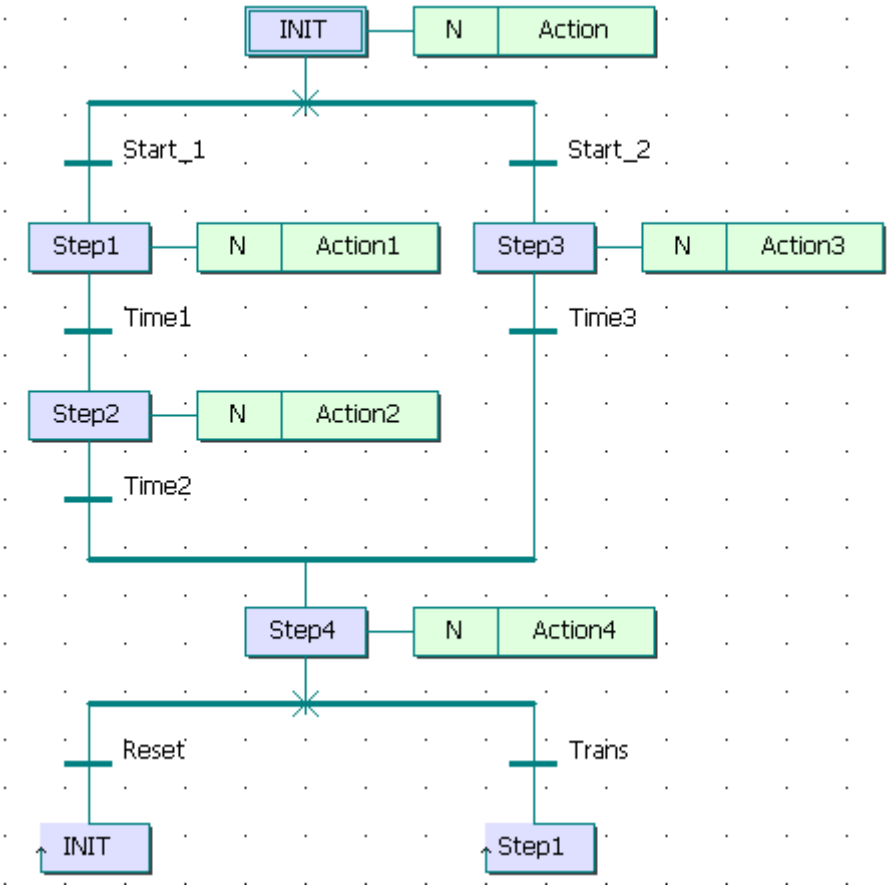
Всі переходи (всіх SFC програм), які можуть бути очищені (тобто вони дозволені і їх умова true), очищаються одночасно.

- ***Одночасна активація і деактивація кроку***

Якщо під час роботи крок одночасно активується і деактивується, пріоритет віддається активізації[4-9].

1.2.6.2. **Приклад програмування SFC – мовою**

Як приклад програмування SFC – мовою розробимо програму керування роботою двох двигунів згідно з алгоритмом наведеним у параграфі 1.2.2.2.
Проектний код SFC-мовою має вигляд:



Тут:

Action (ST-мовою) - motor_1:=FALSE; motor_2:=FALSE; Cycle_Count:=0;

Start_1 (ST-мовою) - Motor_Counter1(CU:=Motor_Start1(* BOOL *),
RESET:=start_1(*BOOL *),PV:=int#1(* INT *));
Start_1(* BOOL *):=Motor_Counter1.Q;
Pressed1(* INT *):=Motor_Counter1.CV;

Start_2(ST-мовою) - Motor_Counter2(CU:=Motor_Start2(* BOOL *),
RESET:=start_2(* BOOL *),PV:=int#2(* INT *));
Start_2(* BOOL *):=Motor_Counter2.Q;
Pressed2(* INT *):=Motor_Counter2.CV;

Action1(ST-мовою) - Motor_1:=TRUE; Motor_2:=FALSE;

Action3(ST-мовою) - Motor_2:=TRUE; Motor_1:=FALSE;

Time1(ST-мовою) - TON_1(IN:=step1.X(* BOOL *),PT:=t#5s(* TIME *));
time1(* BOOL *):=TON_1.Q;
A_time1(* TIME *):=TON_1.ET;

Time3(ST-мовою) - TON_3(IN:=step3.X(* BOOL *),PT:=t#10s(* TIME *));
time3(* BOOL *):=TON_3.Q;
A_time3(* TIME *):=TON_3.ET;

Action2(ST-мовою) - Motor_2:=TRUE; Motor_1:=FALSE;

Time2(ST-мовою) - TON_2(IN:=step2.X(* BOOL *),PT:=t#10s(* TIME *));
Time2(* BOOL *):=TON_2.Q;
A_time2(* TIME *):=TON_2.ET;

Action4(ST-мовою) - IF Time1 or Time2 or Time3 (*EXPRESSION (must return a boolean value)*)
THEN Cycle_count:=Cycle_Count+1 (*If returned value of EXPRESSION = TRUE*)
(*STATEMENT*);
END_IF;

Reset(IL-мовою) - LD Cycle_Count(*N*) (*IN as ANY*)
GT 10
ST Reset

Trans(IL-мовою) - LD Cycle_Count(*N*) (*IN as ANY*)
LE 5
ST Trans

Контрольні запитання

1. З яких компонентів складається SFC-мова?
2. Як зображається крок і які кроки бувають?
3. Як зображаються переходи і коли вони виконуються?
4. Яке призначення мають орієнтовані лінії та стрибки?
5. Де і як у SFC- програмі показуються дії?
6. Що визначає класифікатор?
7. Як описуються дії кроків?
8. Як характеризуються булеві дії?
9. Як характеризуються імпульсні дії?
10. Як характеризується дія, що не зберігається?
11. Як викликаються функції і функціональні блоки?
12. Як створюються умови переходів?
13. Що таке розбіжності і сходження і які вони бувають?
14. Чим характеризуються паралельні гілки і одиничні?

2. СИСТЕМА ПРОГРАМУВАННЯ MULTIPROG

MULTIPROG – це SoftLogic-система програмування промислових логічних контролерів, яка відповідає стандарту IEC 61131-3. Система базується на віконній технології, яка використовує графічні можливості MS - Windows. Тому принципи стандарту здебільшого реалізуються графічно з символами і іконами або діалогами, в яких можуть бути визначені властивості елементів.

За допомогою MULTIPROG можна програмувати на двох текстових ST та IL і трьох графічних – LD, FBD та SFC мовах стандарту IEC 61131-3. Здійснюється це в діалоговому режимі, використовуючи, так званого, Project Wizard (Майстр Проекту).

Графічний редактор системи дозволяє вільне розміщення об'єктів на робочому полі проекту, вибирати мову програмування та використовувати бібліотеку операторів, функцій і функціональних блоків. Він підтримує змішування графічних мов в одному робочому листку, вставлення нових елементів в існуючі мережі, переміщення об'єктів або мереж.

Текстовий редактор дозволяє легко і швидко створювати програми, завдяки кольоровому зображенню ключових слів, використанню бібліотечних блоків і елементів.

Система програмування має могутній засіб налагодження із зручним інтерфейсом користувача. За допомогою MULTIPROG перевірка програми контролера здійснюється на симуляторі без участі самого контролера.

MULTIPROG працює разом з програмованою операційною системою реального часу ProConOs (Programmable Controller Operating System), яка вбудована в контролер[6].

2.1. Проекти в системі програмування

Проект у MULTIPROG містить усі необхідні елементи системи автоматизації. Він складається з бібліотеки(Libraries), типу даних(Data Type), програмних модулей (Logical POUs) і набору елементів конфігурації, які зображені у піддереві Physical Hardware (Фізична апаратура). Дерево проекту приведено на рис.2.1.

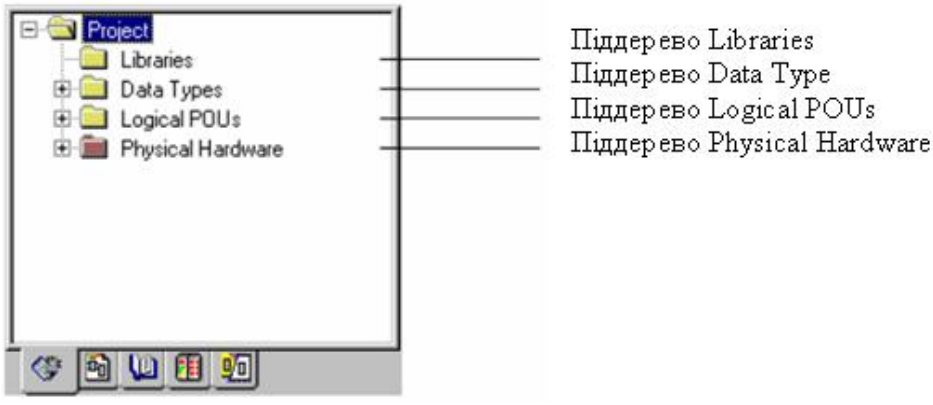


Рис.2.1. Дерево проекту

Редактор проектного дерева дозволяє створювати, вставляти, видаляти, копіювати та переміщувати об’єкти в проектному дереві.

Щоб створити функцію або функціональний блок користувача, необхідно:

- вставити новий POU в проектне дерево, викликаючи правою клавішею миші у піддереві Logical POUs (Логічні POUs) діалог Insert (Вставка);
- відкрити робочий листок тіла коду новоствореного POU;
- відредагувати тіло коду, вставляючи інструкції, оператори, контакти, котушки і т.і. (залежно від мови програмування), а також функції і функціональні блоки та змінні;
- створити робочий листок сітки змінних POU;
- зберегти відредаговані робочі листки;
- компілювати редагований POU.

Після компіляції нова функція або функціональний блок поміщаються в область вибору Edit Wizard (Майстер редагування) - група Project name (Ім’я проекту) і можуть вибиратися і вставлятися, як певна функція або функціональний блок, в інші POUs поточного проекту.

Об’єкти можуть бути вставлені в проектне дерево за допомогою контекстного меню ікон проектного дерева. Контекстне меню кожного об’єкта проектного дерева містить підменю Insert (Вставка), яке показує об’єкти, що можуть бути вставлені в проектне дерево.

Наприклад, підменю Insert (Вставка) піддерева Logical POUs (Логічні POUs) містить пункти Program(Програма), Function Blok(Функціональний блок), Function(Функція), а контекстне підменю ікони робочого листка тіла коду дозволяє вставити тільки інший робочий листок тіла коду.

Щоб вставити об’єкт:

- позначте курсором у проектному дереві бажану ікону і клацніть по неї правою клавішею миші;
- відкриється підменю Insert (Вставка), виберіть об’єкт, який має бути вставлений. З’явиться відповідне діалогове вікно Insert (Вставка);
- після призначення імені та вибору мови програмування натисніть ОК і новий об’єкт з’явиться у проектному дереві.

З проектного дерева можна видалити програми, функціональні блоки, функції і всі робочі листки. Можна також видалити переходи і дії, що створені при програмуванні SFC-мовою. Перед видаленням POU або окремого робочого листа необхідно закрити всі відкриті робочі листки POU. Після цього позначте об’єкт або декілька об’єктів, які потрібно видалити. Виберіть пункт Delet (Видалення) в підменю Edit (Редагування) або натисніть клавішу “DEL”.

Іноді швидше і легше копіювати існуючий об’єкт, ніж створювати новий.

Щоб це зробити:

- позначте лівою клавішею миші об’єкт або декілька об’єктів, що копіюються;

- в підменю Edit (Редагування) виберіть Copy (Копія) і клацніть лівою клавішею миші;
- позначений об’єкт скопіюється у буфер обміну і може бути встановлений в інше місце, використовуючи команду Paste (Вставити) у підменю Edit (Редагування).

Не можна вставити об’єкт у несумісні структури, програми. Наприклад, робочі листи FBD - мовою, в POU ST – програми. Тому команда Paste (Вставити) активізується тільки після того, як буде визначене правильне місце для вставлення скопійованого об’єкта.

Для вставлення об’єкта:

- у проектному дереві позначте місце, куди бажано додати скопійований об’єкт, наприклад, для POU це може бути піддерево Logical POUs(Логічні POU);
- в підменю Edit (Редагування) виберіть Paste (Вставити);
- копійований об’єкт буде вставлений з буфера обміну в позначене положення.

Щоб перемістити об’єкт в проектному дереві, необхідно виконати два кроки – спочатку вирізати об’єкт, а потім вставити його в точку призначення.

Для переміщення об’єкта:

- позначте об’єкт, який переміщується, якщо треба позначте декілька об’єктів;
- в підменю Edit (Редагування) виберіть Cut (Вирізати);
- позначений об’єкт копіюється в буфер обміну і може бути вставлений в інше положення, використовуючи команду Paste (Вставити).

Після вставлення об’єкта в нове положення, зі старого положення він буде видалений.

Система програмування може забезпечити захист проекту проти несанкціонованого вставлення або вилучення об’єктів (таких як POUs і робочі листки) у проектному дереві. Залежно від установок безпеки, зроблених за допомогою Project owner (Власник проекту), доступ до проекту можна обмежити завдяки паролю. Після цього у дерево об’єкта, в якому було визначене обмеження, додається ікона з символом ключа.

2.1.1. **Бібліотеки**

Бібліотеки (Libraries) – це POUs , що оголошені як бібліотеки. Існують бібліотеки мікропрограм і бібліотеки користувача.

Бібліотеки мікропрограм – це бібліотеки, які створені виробниками PLC. Файл такої бібліотеки має розширення *.fwl.

Бібліотеки, що призначені користувачем, – це проекти, функціональні блоки та функції, які створені раніше. Маючи вже закінчений проект, створений функціональний блок або функцію, можна використовувати їх POUs і робочі листки у нових проектах багато разів. Тобто користувачеві немає потреби створювати тіло коду, яке вже існує.

Бібліотека користувача знаходиться у файлі, з розширенням *.mwl.

Бібліотеки мають власне піддерево у проектному дереві. Можна показати повне проектне дерево або тільки піддерево “Бібліотеки”, клацаючи по етикетці Libraries (Бібліотеки), що внизу вікна проектного дерева.

Піддерево Libraries (Бібліотеки), рис.2.2, складається з двох або більше ікон. Перша ікона – це вузол директорії. Елемент цього вузла директорії являє оголошену бібліотеку.

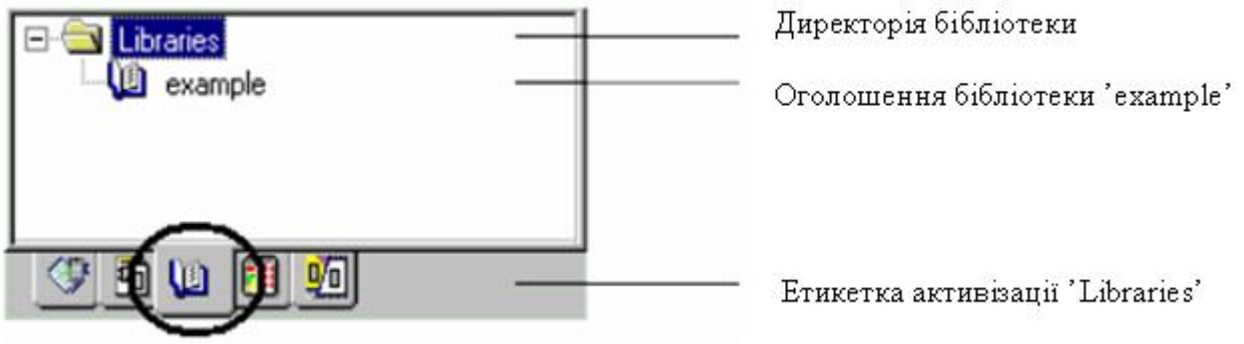


Рис. 2.2. Піддерево бібліотеки користувача

При створенні бібліотеки потрібно дотримуватися наступного:

- для оголошення бібліотек можуть використовуватися тільки POUs;
- в оглядовому режимі робочі листки POUs оголошених бібліотек користувача можуть бути тільки відкриті, тобто тільки показані й не можуть бути редагованими;
- одночасно робочі листки бібліотеки користувача, що відкриті в оглядовому режимі, можуть перемикатися в оперативний режим, в якому можливе їх налагодження;
- робочі листки бібліотеки мікропрограм не можуть бути відкритими ні в оглядовому режимі роботи, ні в оперативному;
- робочі листки бібліотеки не можуть бути надрукованими;
- право для оголошення бібліотек і огляду їх робочих листків може обмежуватися паролем користувача.

Щоб оголосити бібліотеку, позначте у проектному дереві теку піддерева Libraries (Бібліотеки) .

- В панелі інструментів клацніть по іконі Add Object (Додати об’єкт). Альтернативно можна вибрати пункт меню Add Library (Додати бібліотеку) в підменю Project (Проект).

Примітка. Якщо ікона Add Object (Додати об’єкт) інертна, то доступ до бібліотеки захищений. Для отримання доступу необхідно ввести пароль, використовуючи пункт меню Enter password (Ввести пароль) в підменю File (Файл).

- У діалоговому вікні Include library (Вставити бібліотеку), що з’явилося, виберіть бажану бібліотеку, тобто mwt-файл бібліотеки користувача, або fwl-файл бібліотеки мікропрограм.

- Виберіть у теці KWSoft системи програмування MULTIPROG теку Project (Проект), а в неї - файл, який бажано оголосити як бібліотеку, і у діалоговому вікні Include library (Включити бібліотеку) натисніть кнопку Include . Ікона створеної бібліотеки вставиться у проектне дерево. Щоб переглянути бібліотеку, необхідно відкрити проектне дерево “Libraries”, а в ньому - необхідні робочі листки.

Для видалення бібліотеки позначте її ікону в проектному дереві і натисніть клавішу “DEL”, з’явиться попередження, в якому потрібно підтвердити видалення.

2.1.2. **Типи даних**

Типи даних у системі програмування – це, в основному, ті самі типи даних, які описані в стандарті IEC 61131-3. Певні типи даних, які призначені користувачем, редагуються в робочих листках типу даних, що знаходяться у піддереві Data Types (Типи даних), рис. 2.3, дерева проекту.

Рис. 2.3. Піддерево типів даних

Щоб вставити робочі листки типу даних у проектне дерево, позначте теку Data Types (Типи даних). В панелі інструментів клацніть по іконі Add Objiect (Додати об’єкт) або виберіть в підменю Project (Проект) пункт Add Data Type (Додати тип даних).

Примітка. Якщо ікона інертна, захист з використанням пароля активізований. Тому треба ввести пароль використовуючи пункт меню Enter pass word (Ввести пароль) в підменю File (Файл).

З’явиться діалогове вікно Insert (Вставка). Введіть ім’я нового робочого листка типу даних і підтвердіть діалог. Новий робочий листок типу даних буде вставлено у відповідне піддерево.

Щоб відкрити робочий листок типу даних з текстовим редактором, в дереві проекту подвійно клацніть по іконі Data Types Worksheet (Робочий листок типу даних). Може статися, що виробник ПЛК визначив типи даних, які можуть використовуватися при редагуванні програми. Тому треба заздалегідь подивитися в документацію технічного засобу.

Піддерево Data Type (Тип даних) можна оглядати, показуючи повне проектне дерево або тільки піддерево Data Type (Тип даних) і Logical POUs (Логічні POUs). Для цього клацніть по етикетці POUs в меню проектного дерева, що розташоване внизу вікна.

2.1.3. **POUs**

Програми, функціональні блоки і функції можуть редагуватися у піддереві Logical POUs (Логічні POUs), рис. 2.4, проектного дерева.

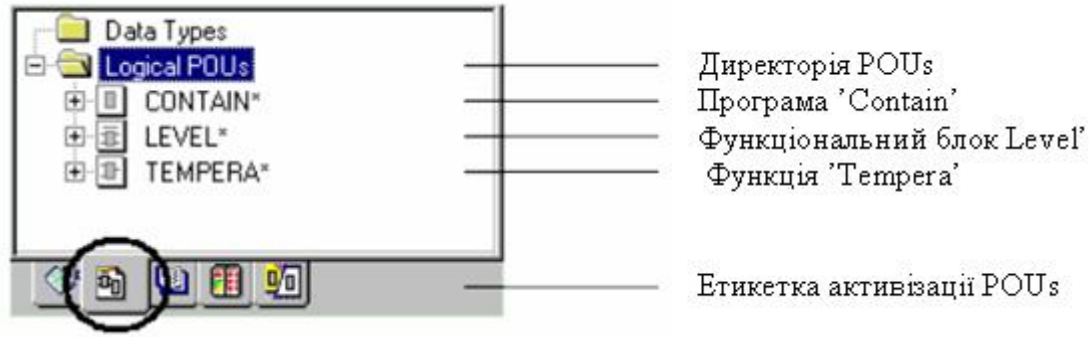


Рис. 2.4. Піддерево одиниць організації програми (POUs)

Кожне POU складається з декількох робочих листків, рис. 2.5:

- Робочий листок опису створюється для документації і містить текст, введений користувачем. В ньому описується POU або елементи конфігурації, проте він не є обов'язковим;
- Робочий листок сітки змінних містить декларацію локальних змінних;
- Робочий листок тіла коду містить фактичний код, що редагувався однією з найзручніших мов програмування.

Рис. 2.5. Структура директорії POUs

Ці робочі листки у піддереві Logical POUs (Логічні POUs) зображаються графічно іконами і мають назву, яка співпадає з назвою власного POU.

При програмуванні SFC - мовою робочі листки POUs доповнюються робочими листками дій та переходів. Якщо робочий листок коду відкритий,

можна легко відкрити зв'язаний з ним робочий листок сітки змінних, двічі клацнувши по відповідній іконі у піддереві Logical POUs (Логічні POUs) або вибравши пункт меню Open Variables Worksheet (Відкрити робочий листок змінних) у підменю View (Огляд), або клацаючи по іконі Variables Worksheet (Змінні робочого листка) в панелі інструментів.

Піддерево Logical POUs (Логічні POUs) містить усі POUs, які використовуються в проекті.

Можна оглядати піддерево Logical POUs (Логічні POUs), показуючи повне проектне дерево або тільки піддерева Data types (Тип даних) і Logical POUs (Логічні POUs). Для цього необхідно клацнути по етикетці POUs в меню проектного дерева, що розташоване внизу вікна.

Щоб вставити POUs у проектне дерево, позначте теку Logical POUs (Логічні POUs) і в панелі інструментів клацніть по одній з наступних ікон :

Add program (Додати програму), щоб вставити програму.

Add FB (Додати FB), щоб вставити функціональний блок.

Add Function (Додати функцію), щоб вставити функцію.

Альтернативно можна вибрати відповідний пункт меню у підменю Project > Add POU(Проект > Додати POU) або клацнути правою клавішею миші;

- з'явиться діалог Insert (Вставка);
- введіть ім'я нового POU;
- виберіть бажану мову, активізуючи відповідний перемикач в зоні Language (Мова);
- якщо потрібно введіть PLC і/або тип процесора;
- позначте перемикач Use Reserve (Використання резерву), якщо потрібно.

Use Reserve (Використання резерву) означає, який резерв пам'яті POU використовується для Patch POU (Вставка POU). Значення Use Reserve (Використання резерву) записано у діалозі Data Area (Область даних), який викликається натисненням кнопки Data Area (Область даних) у діалозі Resource setting (Установки ресурсу) вашого ПЛК.

- Підтвердіть діалог.

Новий POU є вставлений у проектне дерево. Він містить робочий листок тіла коду, робочий листок змінних і робочий листок опису.

Примітка. Якщо право на вставлення POU у проектне дерево обмежене паролем користувача, з'явиться повідомлення про помилку. В даному випадку необхідно зареєструватися в проекті, використовуючи дійсний пароль проекту. Для цього потрібно відкрити підменю File (Файл) і вибрати пункт меню Enter password (Ввести пароль).

Нові робочі листки в проектному дереві позначені зірочкою.

Зірочки означають, що ці робочі листки можуть бути вставлені або замінені, але ще не компільовані.

Система забезпечує захист структури піддерева Logical POUs (Логічний POUs) проти несанкціонованої вставки або видалення POUs і робочих листків, використовуючи пароль.

В POU можна вставити робочі листки коду програми, проте тільки того самого типу. Наприклад, можна вставити в IL- POU тільки IL-робочі листки.

Щоб вставити новий робочий листок:

- відкрийте бажаний POU;
- позначте один з робочих листків POU;
- в панелі інструментів клацніть по іконі Add Object (Додати об'єкт), з'явиться діалогове вікно Insert (Вставка).

Примітка. Якщо ікона інертна, в проекті використовується захист паролем. Треба ввести пароль. Для цього виберіть пункт меню Inter password (Ввести пароль) у підменю File (Файл).

- У діалоговому вікні введіть ім'я нового робочого листка;
- визначтесь, новий робочий листок буде вставлений після (перемикач Append (Приєднати)) чи до (перемикач Insert (Вставка)) позначеного робочого листка;
- підтвердіть діалог Insert (Вставка).

Новий робочий листок є вставленим у проектне дерево.

Нові робочі листки позначені у проектному дереві зірочками, які означають, що робочі листки встановлені або змінені, але не компільовані.

Можна переглянути у проектному дереві властивості усіх існуючих POUs і робочих листків. Деякі властивості можуть бути змінені.

Мова програмування існуючого POUs або робочого листка не може бути змінена оскільки вона визначається при створенні POU.

Для перегляду або зміни властивостей:

- клацніть правою клавішею миші по бажаній іконі теки POU або іконі робочого листка у проектному дереві. З' явиться контекстне меню ікони;
- виберіть пункт меню Properties (Властивості) для відкриття діалогу Properties (Властивості);
- змініть властивості, якщо бажаєте.

Перейменування POU або робочого листка не можливо, коли робочий лист відкритий.

2.1.4. **Елементи конфігурації у системі програмування**

Елементи конфігурації у проектному дереві зображені графічно. Система програмування відображає структуру елементів конфігурації у піддереві Physical Hardware (Фізична апаратура).

Взагалі можуть використовуватися одна або декілька конфігурацій. У кожній конфігурації можуть бути оголошені один або декілька ресурсів. У межах одного ресурсу можуть використовуватися декілька задач, зв'язаних з їх програмами. Задачі визначають планування часу програм.

У ресурсі можуть бути оголошені глобальні змінні, які дійсні тільки в межах цього ресурсу.

Протягом створення робочого коду програм и функціональних блоків у піддереві Physical Hardware (Фізична апаратура), рис. 2.6, додаються їх екземпляри, а також редагуються окремі робочі листки глобальних змінних. Як результат кожний екземпляр програми або функціонального блока має власний набір даних з безпосереднім доступом.

Рис. 2.6. Структура піддерева Physical Hardware

Екземпляри програми і функціонального блока корисні, якщо маємо, наприклад, дві ідентичні машини свердління, які керуються одним ПЛК через різні модулі входу-виходу. Кожний екземпляр працює з даними його власної декларації глобальних змінних.

У піддереві Physical Hardware (Фізична апаратура) можна створити тільки екземпляр функціонального блока в програмі і не можна створити екземпляр функціонального блока в межах функціонального блоку.

Ім'я екземпляра, яке використовується у піддереві Physical Hardware (Фізична апаратура) має відповідати імені програми, яке використовується у піддереві Logical POUs (Логічні POUs).

Приклад екземпляра програми та функціонального блока з відповідним робочим листком глобальних змінних приведено на рис. 2.7:

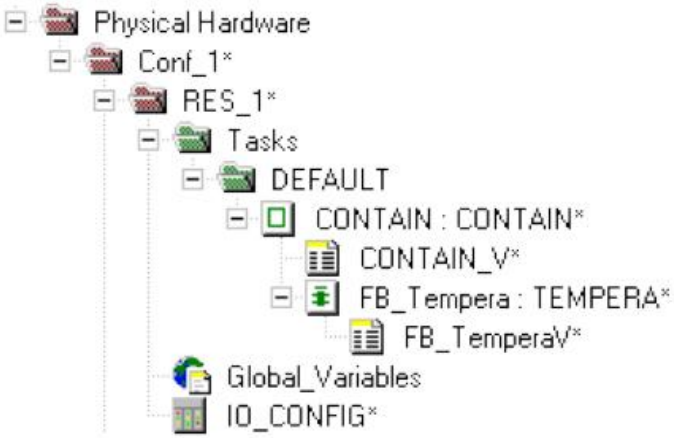


Рис. 2.7. Екземпляр програми і функціонального блока

Дерево екземпляра показує доступні ресурси, задачі і зв'язані з ними програми з усіма функціями і функціональними блоками ресурсу. Структура дерева створюється протягом компіляції ресурсу. Тому дерево екземпляра не може редагуватися.

Щоб зробити видимим дерево екземпляра, необхідно клацнути по етикетці Instances (Екземпляри) в меню, що розташоване внизу вікна проектного дерева.

Для оголошення екземплярів програми або функціонального блока, доведеться використовувати додаткову змінну – ключові слова.

Ці додаткові ключові слова можуть бути вибрані зі стовпчика Usage (Вжити) робочого листка сітки змінних, коли оголошуються екземпляри. Пояснення наведені у наступній табл.2.1.

Таблиця 2.1. Ключові слова

Ключові слова	**Опис**
VAR_EXTERNAL_FB	**-для визначення глобальних змінних в функціональному блоці, який вставлений у піддерево Physical Hardware (Фізична апаратура);** **-така змінна потрапляє в декларацією VAR_GLOBAL_FB екземпляра функціонального блока в піддереві Physical Hardware (Фізична апаратура);** **-їх значення можуть бути змінені в межах POU;** **-може використовуватися тільки для декларації символьних змінних;**
VAR_EXTERNAL_PG	**-для глобальних змінних, що використовуються в програмах або функціональних блоках, які вставлені у піддерево Physical Hardware (Фізична апаратура);** **-така змінна потрапляє в декларацією VAR_GLOBAL_PG глобальної змінної екземпляра програми у піддереві Physical Hardware (Фізична апаратура);** **-їх значення можуть бути змінені в межах POU;** **-може використовуватися тільки при декларації символьних змінних;**
VAR_GLOBAL_FB	**-для глобальних змінних, що використовуються відповідними екземплярами функціональних блоків;** **-можуть використовуватися для декларації безпосередньо зображених, розміщених і символьних змінних;** **-можуть використовуватися з ключовим словом RETAIN для декларації затримуючих змінних;**

Продовження таблиці 2.1

VAR_GLOBAL_PG	**-для глобальних змінних, що використовуються відповідними екземплярами програм;** **-можуть використовуватися для декларації безпосередньо зображених, розміщених і символьних змінних;** **-можуть використовуватися з ключовим словом RETAIN для декларації затримуючих змінних;**

Контрольні запитання

1. Що собою являє система програмування MULTIPROG?
2. Який вигляд має проект, створений у MULTIPROG?
3. Що таке бібліотека у системі програмування MULTIPROG?
4. Як створити бібліотеку?
5. Як зображаються типи даних у системі програмування MULTIPROG?
6. Який вигляд має директорія POUs і що вона містить?
7. Як система програмування відображає структуру елементів конфігурації?
8. Як вставити проект у проектне дерево?
9. Як створити функцію або функціональний блок?
10. Як у проектному дереві об'єкти видаляються, копіюються та переміщуються?
11. Як вставити POU у проектне дерево?
12. Для чого створюються екземпляри програм і функціональних блоків?

2.2. Інструменти програмування ПЛК

2.2.1. Редактор LD- мови

Щоб редагувати робочий листок тіла коду LD-мовою, необхідно відкрити його в дереві проекту, двічі клацнувши лівою клавішею миші по відповідній іконі теки Logical POUs(Логічні POUs)[6].

LD-мережу можна скласти з окремих елементів LD-мови, а можна вставити її в робочий листок автоматично в складі одного контакту, однієї котушки і двох шин живлення. LD-мережа завжди має заданий розмір, але коли в мережу вставляються додаткові мовні елементи, її розмір коригується автоматично. Якщо з LD-мережею сполучаються SFC- і FBD- мережі, то її розмір коригується вручну.

LD- мережа завжди вставляється у робочий листок тіла коду з лівого краю, а положення вставки визначається місцем розташування мітки.

Для того, щоб створити LD- мережу:

- у зручному місці робочого листка тіла коду клацніть лівою клавішею миші, щоб позначити вихідну точку програмування;

- в панелі інструментів клацніть по іконі Contact network (Контактна мережа) і в робочому листку тіла коду з'явиться перша LD- мережа з одним контактом, одною котушкою і двома шинами живлення:

LD- мережу можна завжди розширити за рахунок додаткового контакту вставленого зліва або справа від існуючих контакту і котушки, а також котушки, яка вставляється тільки справа від контакту або котушки.

Щоб вставити інший стандартний об'єкт:

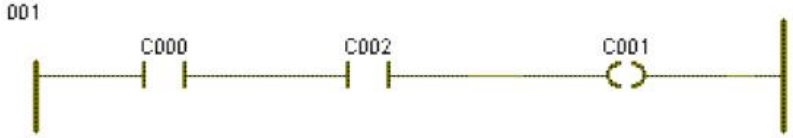
- у LD- мережі позначте існуючий контакт або котушку;

- щоб вставити контакт справа, в панелі інструментів клацніть по іконі Add contact right (Додати контакт справа);

- щоб вставити контакт зліва, в панелі інструментів клацніть по іконі Add contact left (Додати контакт зліва);

- щоб вставити котушку справа, в панелі інструментів клацніть по іконі Add coil right (Додати котушку справа).

Перша LD- мережа зі вставленим додатковим контактом C002 має вигляд:



Елементи LD- мови можна вставити паралельно існуючим контактам і котушкам. LD-редактор автоматично вставляє той елемент, який був позначений заздалегідь. Якщо позначити контакт, новий контакт і буде вставлений, якщо позначити котушку, буде встановлена нова котушка.

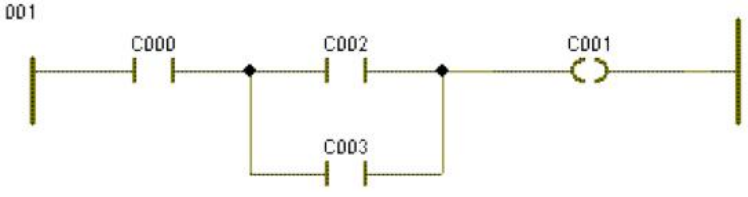
Щоб вставити паралельно інший елемент:

- позначте лівою клавішею миші існуючий контакт або існуючу котушку;

- для вставлення нового елемента нижче вибраного в панелі інструментів клацніть по іконі Add contact/coil below (Додати контакт/ котушку знизу);

- для вставлення нового елемента вище вибраного в панелі інструментів клацніть по іконі Add contact/coil above (Додати контакт/ котушку зверху).

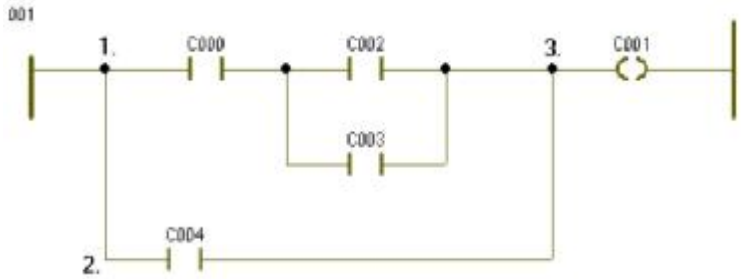
LD-мережа з новим паралельним контактом C003, вставленим нижче вибраного існуючого C002, має вигляд:



Графічний редактор дозволяє вставляти у LD-мережу паралельні гілки. Для цього використовується режим редагування LD-гілок, який є зручним інструментом для створення складних LD-мереж. Режим редагування LD-гілок не може використовуватися, якщо робочий листок пустий. Тому необхідно спочатку створити LD- мережу, як мінімум з одним контактом або котушкою.

Для вставлення паралельної гілки:

- в панелі інструментів клацніть лівою клавішею миші по іконі Insert LD branch (Вставити LD-гілку), щоб активізувати режим редагування LD-гілок. Коли режим редагування LD-гілок активізовано, ікона в панелі інструментів є „натиснутою";
- клацніть лівою клавішею миші по лінії сполучення між двома об'єктами, де бажано почати паралельну гілку (крок 1);



- просувайте мишу вперед або назад, виводячи нову сполучну лінію у вільну зону (крок 2);
- клацніть лівою клавішею миші на рівні майбутнього розташування нового мовного об'єкта;
- перемістіть мишею курсор до бажаного кінця паралельної гілки (крок 3);
- клацніть лівою клавішею миші, щоб встановити зв'язок ліній.

У LD-мережу можна вставити шини живлення, якщо їх немає, або видалити, коли вони залишилися не зв'язаними з будь-яким елементом.

Щоб вставити шину живлення:

- у LD- мережі позначте елемент, біля якого бажаєте зробити вставку;
- в панелі інструментів клацніть по іконі Left power rail (Ліва шина живлення), щоб вставити ліву шину живлення;
- в панелі інструментів клацніть по іконі Right power rail (Права шина живлення), щоб вставити праву шину живлення.

Для видалення шини живлення позначте її лівою клавішею миші і натисніть "Delete" на клавіатурі комп'ютера.

Окремі LD- мережі можна з'єднати між собою. Ліві шини живлення сполучаються тільки з лівими, а праві - тільки з правими.

Для з'єднання LD- мереж:

- в панелі інструментів клацніть лівою клавішею по іконі Connect objects (З'єднати об'єкти) ;
- позначте курсором першу шину живлення;

- клацніть послідовно по інших шинах живлення - і вони з'єднаються, а їх положення і розміри встановляться автоматично.

Для створення декількох паралельних мереж однакового розміру і вже з'єднаних одна з одною необхідно нижче лівої шини живлення попередньої мережі встановити мітку і в панелі інструментів клацнути по іконі Contact network (Контактна мережа). Зробити це потрібно стільки разів, скільки додаткових мереж необхідно вставити.

Для сполучення віддалених об'єктів можна застосовувати з'єднувачі замість ліній зв'язку. З'єднувачі входу і виходу, що мають однакові імена, сполучаються автоматично. Можна мати один з'єднувач виходу і декілька вхідних з'єднувачів з однаковими іменами.

З'єднувачі можуть використовуватися тільки в межах одного робочого листка.

Щоб вставити з'єднувач:

- у робочому листку позначте вихід об'єкта, до якого має бути встановлений з'єднувач;
- в панелі інструментів клацніть по іконі Connector/Jump or Label (З'єднувач /перехід або мітка). З'явиться діалог Connector/ Jump (З'єднувач/ Перехід);
- введіть ім'я з'єднувача;
- активізуйте кнопку Connector (З'єднувач);
- підтвердіть діалог - і з'явиться з'єднувач, зв'язаний з позначеним об'єктом.

З'єднувач можна вставити у будь-якому місці робочого листка. Потрібно тільки встановити мітку і потім клацнути по Connector/ Jump or Label (З'єднувач/ Перехід або мітка) .

При редагуванні робочого листка тіла коду можуть використовуватися переходи між двома об'єктами. Для реалізації переходів необхідно вставити у робочий листок перехід і мітку, яка вказує точку переходу. Перехід і мітка повинні мати однакові імена.

Щоб вставити перехід:

- позначте в робочому листку вихід об'єкта, де має бути вставлений перехід;
- в панелі інструментів клацніть по іконі Connector/ Jump or Label (З'єднувач/ Перехід або мітка). З'явиться діалог Connector/ Jump (З'єднувач/ Перехід);
- введіть ім'я переходу;
- активізуйте кнопку Jump (Перехід);
- підтвердіть діалог.

З'явиться перехід, зв'язаний з позначеним об'єктом .

Перехід можна вставити в будь-якому місці робочого листка. Потрібно вибрати точку вставки і потім в панелі інструментів клацнути по іконі Connector/ Jump or Label (З'єднувач/ Перехід або мітка).

Щоб вставити мітку:

- у робочому листку клацніть лівою клавішею миші для позначення місця вставки мітки ;

- в панелі інструментів клацніть по іконі Connector/ Jump or Label (З'єднувач/ перехід або мітка). З'явиться діалог Connector/ Jump (З'єднувач/ Перехід);

- активізуйте кнопку Label (Мітка);

- введіть ім'я мітки таке саме, як і ім'я переходу;

- підтвердіть діалог і мітка буде встановлена.

У робочих листках тіла коду можуть використовуватися повернення назад у POU, якщо логічна змінна у LD-мережі - TRUE (Істинна). Якщо змінна - FALSE (Хибна), то виконання поточної мережі продовжується.

Для вставлення повернення:

- позначте в робочому листку вихід об'єкта, куди бажано вставити повернення;

- в панелі інструментів клацніть по іконі Return (Повернення) і символ повернення буде вставленим.

Повернення можна вставити в робочому листку де завгодно. Для цього потрібно зробити мітку і продублювати в панелі інструментів клацання по іконі Return (Повернення) .

Розмір LD-мережі можна змінити, використовуючи можливості підменю Layout (Формат). Після коригування розміру всі нові LD-мережі мають нову ширину.

Щоб змінити розмір LD-мережі:

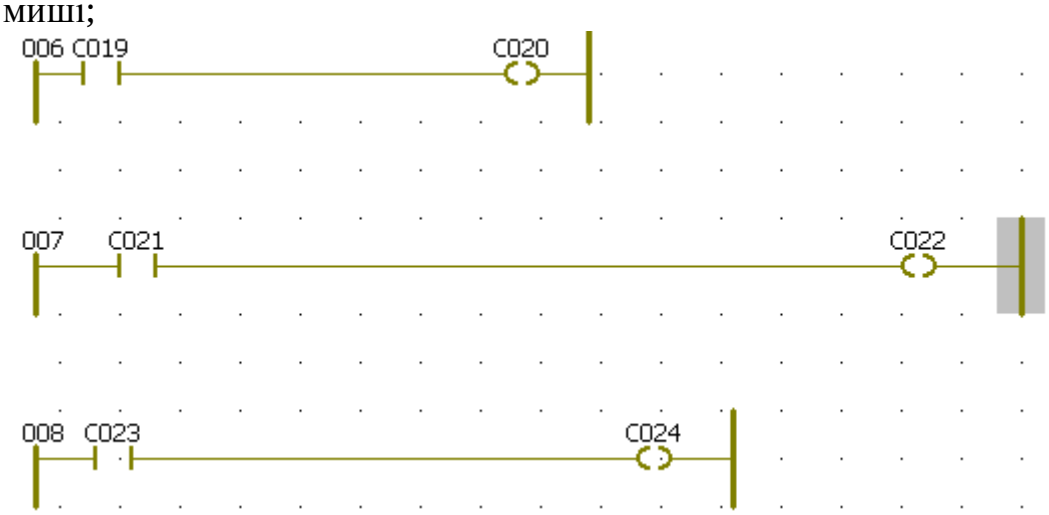
- у підменю Layout (Формат) виберіть пункт Contact Width (Ширина контакту). З'явиться діалог Contact Width (Ширина контакту);

- введіть нове значення ширини мережі і натисніть ОК.

Можна вирівняти LD-мережі, якщо їх праві шини живлення по-різному розташовані. Редактор забезпечує дві можливості вирівнювання правих шин живлення:

1. Вирівнювання шин живлення за самою правою шиною. Для цього:

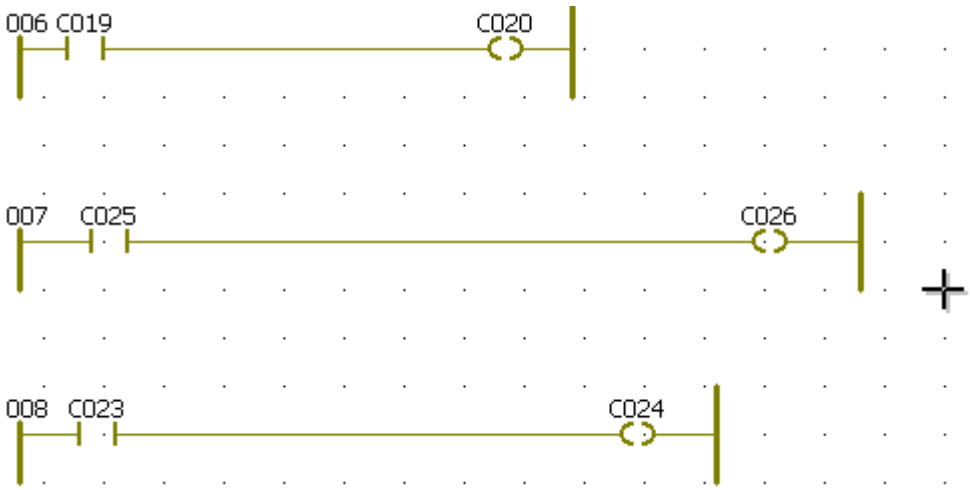
- установіть курсор на саму праву шину живлення і клацніть лівою клавішею миші;



- відкрийте підменю Edit (Редагування). Виберіть і активізуйте пункт Stretch/ Compress > Arrange Power rails (Розтягнення/ Стиснення > Видалення шин живлення). Усі шини живлення змістяться до рівня позначеної шини.

2. Вирівнювання шин живлення за міткою у робочому листку:

- клацніть лівою клавішею миші у точці робочого листка, яка вказує мітку вирівнювання шин живлення. Слід враховувати, що вирівнювання здійснюється тільки зміщенням шин праворуч;



- відкрийте підменю Edit (Редагування). Виберіть і активізуйте пункт меню Stretch/ Compress > Arrange Power rails (Розтягнення/ стиснення > Видалення шин живлення). Усі шини змістяться на рівень встановленої мітки.

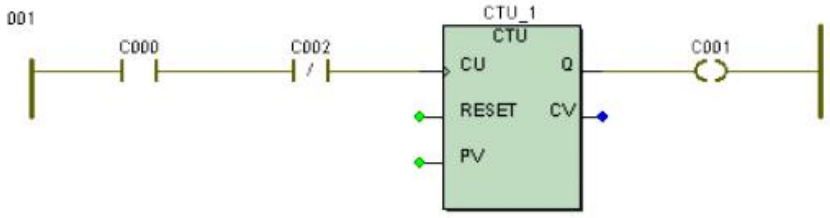
В LD- мережу можна викликати функції та функціональні блоки, які є елементами FBD- мови, тобто в тілі коду LD- програми змішуються елементи LD- і FBD- мови.

Функцію або функціональний блок можна вставити в будь-якому місці робочого листа, а потім з'єднати їх з іншими елементами. Можна також вставити функцію або функціональний блок безпосередньо в існуючу LD-мережу.

Найзручніший шлях встановлення FBD функцій або функціональних блоків в LD- програму - це використання Edit Wizard (Майстра редагування).

- Позначте лівою клавішею миші місце на LD- мережі, де бажаєте вставити функцію або функціональний блок;
- в панелі інструментів натисніть ікону Edit Wizard (Майстра редагування;
- у вікні, що з'явилося, зі списку Group виберіть потрібну функцію або функціональний блок і двічі клацніть по ньому лівою клавішею миші, щоб вставити в позначене місце.

Приклад LD- мережі зі вставленим FBD функціональним блоком CTU має вигляд:



Вільні параметри функціонального блока можуть бути зв'язані з іншими контактами, котушками або змінними.

В LD- мережі можна змінити властивості контактів і котушок. Наприклад, замінити прямий контакт на інвертований або змінити їх початкове ім'я на інше.

Щоб змінити властивості контакту/ котушки:

- подвійно клацніть по контакту або котушці, що змінюється;
- у діалоговому вікні Contact/ Coil Properties (Властивості контакт/ котушка), що з'явилося, на сторінці Contact (Контакт) змініть тип контакту або котушки, використовуючи перемикачі Contact (Контакт) і Coil (Котушка), а також можливості вікна Type (Тип).

Для оголошення змінних контактів або котушок існує дві можливості:

- використання змінних, які вже декларовані в робочому листку змінних;
- використання змінних, які ще не декларовані в робочому листку змінних.

Щоб змінити типове початкове ім'я контакту або котушки на вже існуюче:

- подвійно клацніть по контакту/котушці, для якого/якої бажаєте оголосити вже декларовану змінну. З'явиться діалогове вікно Contact/ Coil Properties (Властивості Контакт/ котушка).

На сторінці Contact (Контакт) список змінних у полі Name (Ім'я) містить усі локальні або глобальні змінні, які були раніше декларовані у відповідних локальних або глобальних робочих листках змінних;

- у вікні списку змінних Name (Ім'я) клацніть по імені бажаної змінної, і позначене ім'я увійде у перший рядок текстового поля змінних;
- натисніть ОК, і вибране ім'я змінної з'явиться у робочому листку тіла коду, як ім'я позначеного контакту/ котушки.

Щоб оголосити нову змінну для контакту або котушки:

- подвійно клацніть по контакту/котушці, для якого/якої бажаєте декларувати нову змінну. З'явиться діалогове вікно Contact/ Coil Properties (Властивості контакт/ котушка);
- на сторінці Contact (Контакт) у полі Name (Ім'я) введіть нове ім'я і натисніть кнопку Apply (Застосувати). Діалогова сторінка Common (Загальний) відкриється автоматично;

- у діалоговій сторінці Common (Загальний) виберіть для змінної контакт/ котушка відповідні установки Usage(Вжити), Data Type(Тип Даних), I/O address(Адреса В/В) і клацніть по Apply (Застосувати), підтверджуючи їх;

- для визначення компетенції нової змінної відкрийте діалогову сторінку Local scope (Локальна компетенція), якщо оголошується локальна змінна або Global scope (Глобальна компетенція) у разі глобальної змінної.

У піддереві відповідної діалогової сторінки виберіть групу змінних Default(За умовчанням), в яку бажаєте вставити опис нової змінної, і клацніть лівою клавішею миші.

Примітка. Якщо декларується глобальна змінна (тобто використовується VAR_EXTERNAL, або VAR_EXTERNAL_PG) доведеться вибрати, як локальну так і глобальну компетенцію.

- Після вибору компетенції клацніть ОК. Змінна з'явиться у робочому листку тіла коду, як ім'я контакту/ котушки, а її декларація автоматично буде вставлена у відібрану групу робочого листка сітки змінних.

У разі глобальної змінної декларація вставляється у сітку локальних змінних, використовуючи ключове слово VAR_EXTERNAL і в таблицю глобальних змінних робочого листа, використовуючи VAR_GLOBAL.

Щоб вставити і декларувати нову змінну, яка не є контактом/ котушкою:

- у робочому листку тіла коду лівою клавішею миші позначте місце вставки нової змінної;

- в панелі інструментів клацніть по іконі Variable (Змінна) і у діалоговому вікніVariable Properties (Властивості змінної), що з'явилося, зробіть все те, що робилося при оголошені змінної контакту або котушки.

Після закриття діалогового вікна змінна з'явиться у робочому листку тіла коду, а її декларація автоматично вставиться у сітку відповідних змінних.

Автовставка декларації змінної не відбувається, якщо змінна має таке ж ім'я, як і тіло коду POU, дії або переходу, а також, коли вона зображається структурою або масивом.

Біля лівої шини живлення LD- мережі можна вставити коментарі. Для цього:

- подвійно клацніть по лівій шині живлення, з'явиться діалог Comment (Коментар);

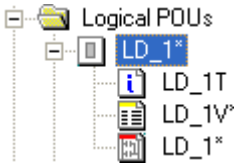
- введіть необхідний коментар і натисніть ОК.

Система програмування може здійснити перекомпіляцію існуючого POU, створеного IL-, FBD- або LD-мовою, кожною з інших двох мов цього переліку.

Для реалізації мовної конверсії:

- переконайтесь, що проект вже побудований і компільований;

- в проектному дереві відкрийте піддерево Logical POUs (Логічні POUs) і позначте ікону існуючого POU;



- клацніть правою клавішею миші, щоб відкрити контекстне меню ікони;

- виберіть пункт Source conversion (Початок конверсії) і клацніть лівою клавішею;

- у діалоговому вікні Source conversion (Початик конверсії), що з'явилося, активізуйте перемикач Overwrite source POU (Переписати початковий POU), якщо бажаєте, визначте нове ім'я POU і виберіть мову перекомпіляції;

- закрийте діалогове вікно і у дереві проекту на місці початкового POU з'явиться новий POU - конвертований вибраною мовою;

- в панелі інструментів клацніть лівою клавішею миші по іконі Compile Worksheet (Компілювати Робочий лист), щоб компілювати POU після конверсії.

У табл.2.2 показани можливі мовні конверсії.

Таблиця 2.2. Можливі мовні конверсії

Початкова мова	**Мови конверсії**
IL	FBD, LD
FBD	IL, LD
LD	IL, FBD

Контрольні запитання

1. Як створити LD-мережу?
2. Як додати до LD-мережі паралельні гілки?
3. Як об'єднати декілька LD-мереж?
4. Для чого і як створюють з'єднувачі?
5. Чи можна змінити розмір LD-мережі?
6. Як вирівняти LD-мережі?
7. Як в LD-мережу можна викликати функцію або функціональний блок?
8. Як в LD-мережі можна змінити властивості контактів і котушок?
9. Як зробити перекомпіляцію проекту?

2.2.2. Редактор FBD-мови

Графічний редактор FBD-мови [6] має багато можливостей, які полегшують створення тіла коду програми, функціонального блока або функції:

– функції і функціональні блоки можуть бути вставлені у робочий листок за допомогою Edit Wizard (Майстра Редагування), а відредаговані завдяки діалогу Variables Praperties (Властивості змінних);

– вставлення і переміщення об'єктів здійснюється простими операціями з клавіатури;

– дублювання і заперечення входів-виходів можна зробити з клавіатури, панелі інструментів і меню;

– елементи тіла коду можуть бути вставлені безпосередньо в лінію сполучення або можуть бути приєднані до входів і виходів інших елементів;

– для налагодження програми викликаються вікна спостереження;

– елементи тіла коду можуть бути створені у будь-якому місці робочого листка;

– подвійне клацання лівою клавішею миші по вибраних користувачем функціях або функціональних блоках відкриває вміст їх POU.

Щоб вставити функцію або функціональний блок у робочий листок тіла коду за допомогою Edit Wizard (Майстер редагування) необхідно:

– у підменю Layout (Розташування) активізувати Grid (Сітка), якщо на робочому листку бажано мати сітку;

– у робочому листку тіла коду визначити положення нової функції або функціонального блока і клацнути лівою клавішею миші;

- в панелі інструментів натиснути ікону Edit Wizard (Майстер редагування), щоб Майстер редагування був видимий;

– у вікні Group (Група) Майстра редагування відкрити список функцій або функціональних блоків і подвійно клацнути по бажаному об'єкту.

Коли вставляється функція, то вона миттєво з'являється у робочому полі після подвійного натиснення лівої клавіші миші.

Коли вставляється функціональний блок, з'являється діалогове вікно Variables Properties (Властивості Змінних), де у полі Name (Ім'я) сторінки Variables (Змінні) пропонується типове ім'я екземпляра блока (наприклад, для CTU пропонується ім'я CTU_n, де n – порядковий номер екземпляра вибраного блока).

Щоб присвоїти унікальне ім'я новому функціональному блоку, введіть його у перший рядок поля Name (Ім'я) замість запропонованого;

– ім'я блока можна вибрати зі списку вже декларованих змінних, що приведені нижче у полі Name (Ім'я);

– за необхідністю можна ввести коментар на новий функціональний блок у текстове поле Description (Опис) сторінки Common (Загальний) діалогового вікна Variables Properties (Властивості Змінних);

– на сторінці Local Scope (Локальна компетенція) діалогового вікна Variables Properties (Властивості Змінних) лівою клавішею миші позначте Default (За умовчанням) і натисніть кнопку Apply (Застосувати), щоб вставити декларацію екземпляра функціонального блока у робочий листок сітки локальних змінних;

– закрийте діалогове вікно Variables Properties (Властивості змінних) і новий функціональний блок буде автоматично вставлений у робочий листок.

На робочому полі тіла коду стандартні функції та функціональні блоки, а також функції та функціональні блоки бібліотеки системи програмування зображаються у червоному кольорі.

Функції та функціональні блоки, що створені користувачем у поточному проекті, позначаються зеленим кольором, а функції та функціональні блоки з бібліотеки користувача – синім.

Може виникнути необхідність заміни вставлених функцій або функціональних блоків на інші. Заміна однієї функції або функціонального блока здійснюється, використовуючи Майстра редагування. Заміна декількох функцій або функціональних блоків одного типу відбувається за допомогою діалогу Local Replace FB/FU (Локальна заміна FB/FU).

Щоб замінити одну функцію або функціональний блок, використовуючи Майстра редагування:

– у робочому полі тіла коду клацніть по функції або функціональному блоку, який потрібно замінити. Позначений об’єкт змінює свій колір;

– використовуючи Майстра редагування виберіть бажану функцію або функціональний блок з вікна Group(Група);

– подвійно клацніть по вибраній функції або функціональному блоку і зміна відбудеться.

Щоб замінити декілька функцій або функціональних блоків одного типу:

– у підменю Edit (Редагування) виберіть пункт Replace FB/FU (Заміна FB/FU). З’явиться діалогове вікно Local Replace FB/FU (Локальна заміна FB/FU);

– виберіть функцію чи функціональний блок у рядку Find What (Що Знайти), які підпадають під заміну;

– виберіть функцію або функціональний блок у рядку Replace With (Замінити з), на який здійснюється заміна;

– натисніть кнопку Find Next (Знайти наступний), щоб замінити першу функцію або функціональний блок;

– натисніть кнопку Replace all (Замінити все), якщо бажаєте замінити всі функції або функціональні блоки цього типу без їх огляду;

– натисніть кнопку Replace (Замінити), якщо бажаєте замінити їх крок за кроком.

Властивості функцій та функціональних блоків при необхідності можна змінити.

Для цього:

– у робочому листі FBD- коду правою клавішею миші клацніть по функції або функціональному блоку , щоб відкрити контекстне меню;

– виберіть пункт Object Properties (Властивості об’єкту) і клацніть лівою клавішею, з’явиться діалог FB/FU Properties (Властивості FB/FU). Коли змінюються властивості функції, з’являється тільки діалогова таблиця FB/FU,

яка показує формальні параметри для зміни. У разі функціонального блока діалогове вікно FB/FU Properties (Властивості FB/FU) передбачає чотири сторінки, в яких потрібно змінити бажані властивості;

– закрийте діалогове вікно.

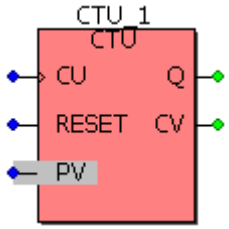
Після зміни властивостей екземпляра функціонального блока, його нова декларація автоматично вставляється у групу локальних змінних.

Оголошення змінних може бути зроблено в будь-якому місці робочого листка тіла коду або після безпосередньої прив'язки їх до входу/виходу (формальному параметру) функції або функціонального блока.

Якщо є необхідність оголосити змінну не прив'язану до функції або функціонального блока, клацніть лівою клавішею миші у робочому листку тіла коду, щоб позначити місце вставки змінної.

Якщо є необхідність оголосити змінну вже зв'язану з функцією або функціональним блоком, позначте формальний параметр об'єкта.

Приклад:



– в панелі інструментів клацніть по іконі Variable (Змінна), з'явиться діалогове вікно Variable Properties (Властивості змінної);

– на сторінці Variable (Змінні) активізуйте кнопку Local (Локальна), якщо бажаєте вставити локальну змінну. Активізуйте кнопку Global (Глобальна), якщо бажаєте вставити глобальну змінну.

Залежно від вибраної компетенції вікно списку змінних нижче рядка Name (Ім'я) містить усі локальні або глобальні змінні, які були задекларовані раніше;

– клацніть по бажаній змінній у вікні списку змінних і позначене ім'я увійде у перший рядок поля Name(Ім'я);

– закрийте діалогове вікно і змінна вставиться у позначеному місці робочого листка тіла коду.

Щоб оголосити і вставити нову змінну:

– в панелі інструментів клацніть по іконі Variable (Змінна), з'явиться діалогове вікно Variable Properties (Властивості змінної);

– на сторінці Variable (Змінні) у рядку Name(Ім'я) введіть нове ім'я і клацніть по Apply (Застосувати);

– у діалоговій сторінці Common (Загальний), що з'явилася, заповніть необхідні рядки для нової змінної - Usage(Вжити), Data Type(Тип Даних), I/O address(Адреса В/В) і клацніть по Apply (Застосувати), щоб підтвердити їх;

- у сторінках Local scope (Локальна компетенція) або Global scope (Глобальна компетенція) визначтесь оголошується локальна чи глобальна

змінна. Для цього при перегляді сторінок у піддереві позначте Default (За умовчанням) і натисніть кнопку Apply (Застосувати).

Примітка. Якщо оголошується глобальна змінна (тобто використовується VAR_EXTERNAL або VAR_EXTERNAL_PG), доведеться вибрати як локальну так і глобальну компетенцію.

− Після вибору компетенції клацніть ОК і змінна вставиться у робочий листок тіла коду, а її декларація автоматично – у вибрану групу робочого листка сітки змінних. У разі глобальної змінної декларація вставляється у сітку локальних змінних, використовуючи ключове слово VAR_EXTERNAL, і у робочий листок сітки глобальних змінних, використовуючи VAR_GLOBAL.

Автовставка опису змінної не відбувається, якщо змінна має таке саме ім'я, як і POU, дія або перехід, а також коли це структура або масив.

Можна зробити заміну змінних, якщо з входом або виходом функціонального блока з'єднали помилкову змінну. Для цього:

Подвійно клацнути по змінній, яка замінюється. З'явиться діалог Variable Properties (Властивості змінних).

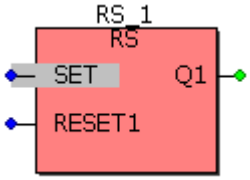
− Для заміни вже декларованою змінною, в діалоговій сторінці Variables (Змінні) виберіть ім'я зі списку змінних, натисніть кнопку Apply (Застосувати) і закрийте діалогове вікно.

Для заміни поточної змінної новою введіть нове ім'я, натисніть кнопку Apply (Застосувати), відкрийте сторінку Common (Загальний) і зробіть необхідні записи, визначтесь з компетенцією у сторінках Local scope (Локальна компетенція) чи Global scope (Глобальна компетенція), а потім закрийте діалогове вікно.

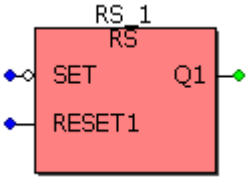
Використовуючи графічний редактор, легко зробити заперечення входів, виходів функцій та функціональних блоків, а також відмінити їх.

Для створення заперечення можна використати діалог FB/FU Properties (Властивості FB/FU) або виконати процедуру, яка описана нижче.

Клацніть лівою клавішею миші по входу або виходу, в якому потрібно зробити заперечення. Позначений формальний параметр отримає кольорову позначку:



– в панелі інструментів клацніть по іконі Toggle negation of FP (Перемикати заперечення FP) і на позначеному вході або виході (формальному параметрі) з'явиться маленьке коло - символ заперечення:



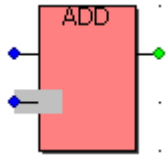
Щоб відмінити заперечення, необхідно знову зробити такі самі кроки для того ж формального параметру.

При використанні діалогу FB/FU Properties (Властивості FB/FU), який викликається подвійним клацанням по функції або функціональному блоці, необхідно в ньому лише активізувати відповідну альтернативу вибраного формального параметра (Formal Parameters) і послідовно натиснути кнопки Apply (Застосувати) та ОК.

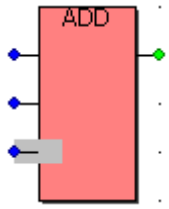
Можна додавати входи функціям, але тільки тим, які працюють із декількома входами. При цьому дублюється тільки останній вхід функції.

Для дублювання входу функції можна використати діалог FB/FU Properties (Властивості FB/FU) або виконати процедуру, яка описана нижче.

– Клацніть по останньому входу функції, який треба дублювати, і він змінить колір:



– в панелі інструментів клацніть по іконі Duplicate FP (Подвійний FP) щоб встановити новий вхід:



При використанні діалогу FB/FU Properties (Властивості FB/FU), необхідно подвійним клацанням лівою клавішею миші по функції викликати діалогове вікно і позначити в ньому ім’я останнього входу функції, а потім послідовно натиснути кнопки Duplicate (Дублювати) та ОК.

Щоб видалити входи, які раніше були продубльовані, необхідно позначити їх і натиснути клавішу Delete(Видалити) на клавіатурі ЕОМ або позначити їх імена у діалоговому вікні FB/FU Properties (Властивості FB/FU) і послідовно натиснути кнопки Delete(Видалити) та ОК.

Замість ліній зв’язку між віддаленими об’єктами, можна вставити з’єднувачі, але тільки в межах одного робочого листка. При виконанні програми сполучення вихідного і вхідного з’єднувачів, які мають однакові імена, здійснюється автоматично. Можна мати з однаковими іменами один з’єднувач виходу і декілька вхідних з’єднувачів.

Щоб зробити вставку з’єднувача:

– у робочому листку тіла коду позначте вихід об’єкта, до якого бажаєте приєднати з’єднувач;

– в панелі інструментів клацніть по іконі Conector/Iumpol (З’єднувач/Перехід або мітка), з’явиться діалогове вікно Conector/Iumpol (З’єднувач/Перехід);

– введіть ім’я з’єднувача і активізуйте кнопку Connector (З’єднувач);

– натисніть ОК і з'єднувач приєднається до позначеного об'єкта.

Не обов'язково з'єднувач приєднувати до об'єкта – його можна вставити у робочому листку, де завгодно. Для цього потрібно на екрані зробити мітку вставки і в панелі інструментів клацнути по Conector/Iumpol (З'єднувач/Перехід або мітка) .

При редагуванні FBD – коду можна використовувати переходи. Для реалізації переходів між двома об'єктами необхідно вставити в робочий листок тіла коду перехід і мітку, яка вказує на місце продовження програми. Перехід і мітка повинні мати однакові імена.

Щоб вставити перехід:

– у робочому листку позначте вихід об'єкта, де має бути вставлений перехід;

– в панелі інструментів клацніть по іконі Conector/Iumpol Label (З'єднувач/Перехід або мітка). З'явиться діалогове вікно Conector/Iumpol (З'єднувач/Перехід);

– введіть ім'я переходу;

– активізуйте кнопку Jump (Перехід), натисніть ОК і перехід приєднається до позначеного об'єкта.

Перехід можна вставити у будь-якому місці робочого листка, потрібно тільки вказати місце вставки і в панелі інструментів клацнути по іконі Conector/Iumpol Label (З'єднувач/Перехід або мітка) .

Щоб вставити мітку:

– у робочому листку тіла коду клацніть лівою клавішею миші, щоб позначити місце вставки мітки;

– в панелі інструментів клацніть по іконі Conector/Iumpol Label (З'єднувач/Перехід або мітка), з'явиться діалогове вікно Conector/Iump (З'єднувач/Перехід);

– активізувати кнопку Label (Мітка);

– введіть ім'я мітки, яке має бути таким, як і ім'я переходу;

– натисніть ОК і мітка є вставленою.

Для повернення назад викликаного POU у робочих листках мають бути вставлені повернення, які спрацьовують, коли логічна змінна є істиною (TRUE). Якщо змінна є хибною (FALSE), виконання поточної мережі проводжується.

Для вставлення повернення:

– у робочому листку позначте вихід об'єкта, де бажано вставити повернення;

– в панелі інструментів клацніть по іконі Return (Повернення) і повернення вставиться на позначене місце.

Повернення можна вставити у робочому листку, де завгодно. Потрібно тільки зробити мітку вставки і в панелі інструментів клацнути лівою клавішею миші по іконі Return (Повернення) .

Якщо функції зв'язані безпосередньо лінією одна з одною (тобто пряма лінія від точки до точки зв'язку), для компіляції програми іноді потрібен внутрішній тимчасовий прапор.

У тих випадках, коли тип даних внутрішнього прапору не визначений структурою мережі, це потрібно зробити вручну, використовуючи діалогове вікно Flag Type Properties (Властивості типу прапора).

Це вікно містить список для конкретизації типу даних прапору.

Звичайно, при компіляції в інформаційному вікні з'являється застереження про наявність помилки. Для визначення місця, де потрібен прапор, двічі клацніть лівою клавішею миші по кнопці Errors (Помилки) і з'явиться відповідне повідомлення.

Щоб змінити тип прапору:

Подвійно клацніть по лінії або виберіть пункт меню Object Properties (Властивості об'єкта) в контекстному меню лінії, яка використовувалась для з'єднання функцій. З'явиться діалог Flag Type Properties (Властивості типу прапора).

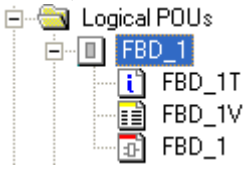
Змініть тип даних прапору і закрийте діалогове вікно Flag Type Properties (Властивості типу прапора).

Система програмування може здійснити перекомпіляцію існуючого POU, створеного IL-, FBD- або LD-мовою, кожною з інших двох мов цього переліку.

Для реалізації мовної конверсії:

– переконайтесь, що проект вже побудований і компільований;

– в проектному дереві відкрийте піддерево Logical POUs (Логічні POUs) і позначте ікону існуючого POU;



– клацніть правою клавішею миші по іконі POU, щоб відкрити його контекстне меню;

– виберіть пункт Source conversion (Початок конверсії) і клацніть лівою клавішею. З'явиться діалогове вікно Source conversion (Початок конверсії);

– у діалоговому вікні Source conversion (Початик конверсії), що з'явилося, активізуйте перемикач Overwrite source POU (Переписати початковий POU), якщо бажаєте, визначте нове ім'я POU і виберіть мову перекомпіляції;

– закрийте діалогове вікно і у дереві проекту на місці позначеного POU з'явится новий POU - конвертований вибраною мовою;

- В панелі інструментів клацніть лівою клавішею миші по іконі Compile Worksheet (Компілювати Робочий листок), щоб компілювати POU після конверсії.

Контрольні запитання

1. Як вставити функцію або функціональний блок у робочий листок тіла FBD-коду?
2. Як замінити властивості функції або блока?
3. Як оголошуються змінні?
4. Як додавати входи функціям і як зробити заперечення входів?
5. Як користуватися з'єднувачами і переходами?

6 Як здійснити перекомпіляцію проекту?

2.2.3. Редактор IL – мови

Тіло коду IL-мовою редагується текстовим редактором, який друкує інструкції або вставляє їх у робочий листок за допомогою Edit Wizard (Майстер редагування). При цьому елементи синтаксису IL-мови зображаються різними кольорами: оператори – блакитні; змінні й імена зразків – чорні; коментарі – зелені [6].

Для того, щоб редагувати тіло коду IL-мовою, необхідно у проектному дереві відкрити робочий листок, подвійно клацаючи по відповідній іконі теки Logical POUs(Логічні POUs).

За допомогою Майстра редагування інструкції вставляються у робочий листок тіла коду із вже завершеною структурою і користувач її просто заповнює змінними.

Якщо Edit Wizard (Майстер редагування) не активізований, треба натиснути на клавіатурі <SHIFT>+<F2> або <Alt>+<3>, або в панелі інструментів клацнути по іконі Edit Wizard.

Щоб вставити інструкцію:

– відкрийте робочий листок тіла коду, визначте положення нової інструкції на екрані і лівою клавішею миші встановіть курсор;

– у відкритому вікні Майстра редагування зі списку Group (Група), виберіть Operators (Оператори) і клацніть лівою клавішею. Майстер редагування покаже доступні оператори;

– виберіть необхідний оператор і двічі клацніть по ньому, щоб вставити у позначене місце. Деякі оператори вставляються окремо, а деякі - разом з іншими, відповідно до синтаксису IL-мови. Наприклад, при вставлені оператора ADD, в робочому полі з'являється така мовна структура:

```
LD   (* IN1 as ANY_NUM *)
ADD  (* IN2 as ANY_NUM *)
ST   (* Result as ANY_NUM *)
```

Щоб декларувати фактичну змінну біля оператора, позначте курсором змінну, що з'явилася з мовною структурою за умовчанням і натисніть<F5> або в панелі інструментів клацніть по іконі Variable (Змінна). Відкриється діалогове вікно Variables (Змінні).

Для того, щоб вставити в мовну структуру змінну, яка вже декларована:

– на сторінці Variables (Змінні) діалогового вікна Variables (Змінні) у списку Name (Ім’я) вже декларованих змінних позначте бажане ім’я і клацніть лівою клавішею миші;

– позначене ім’я увійде у перший рядок списку Name (Ім’я);

– активізуйте перемикач Local scope (Локальна компетенція), якщо бажаєте вставити локальну змінну. Активізуйте Global scope (Глобальна компетенція), якщо бажаєте встановити глобальну змінну;

– закрийте діалог і змінна опиниться у позначеному місці робочого листка тіла коду.

Щоб декларувати нову змінну:

– надрукуйте нову змінну у бажаній позиції коду;
наприклад, LD T_value;

– позначте курсором нове ім’я змінної і в панелі інструментів клацніть по іконі Variable (Змінна);

– з’явиться діалогове вікно Variables (Змінні) з надрукованим іменем T_value у першому рядку поля Name (Ім’я);

– клацніть по кнопці Apply (Застосувати). Діалогова сторінка Common (Загальний) відкриється автоматично;

– на діалоговій сторінці Common (Загальний) виберіть відповідні установки для змінної у рядках Usage (Вжити), Data Type (Тип Даної), а також в інших, якщо потрібно, і клацніть по Apply (Застосувати), щоб підтвердити їх;

– відкрийте діалогову сторінку Local scope (Локальна компетенція), якщо декларуєте локальну змінну або Global scope (Глобальна компетенція) у разі глобальної змінної;

– у дереві діалогової сторінки для вибраної групи змінних лівою клавішею миші позначте Default (За умовчанням) .

Примітка. Якщо оголошуєте глобальну змінну (тобто використовуєте VAR_EXTERNAL або VAR_EXTERNAL_PG) необхідно вибрати як локальну, так і глобальну компетенцію.

– Після вибору компетенції закрийте діалогове вікно.

Змінна вставиться у робочий листок тіла коду, а її декларація автоматично увійде у робочий листок сітки відповідних змінних.

У разі глобальної змінної, декларація вставляється у сітку локальних змінних з ключовим словом VAR_EXTERNAL і у робочий листок сітки глобальних змінних з ключовим словом VAR GLOBAL.

Автовставка декларації змінної не відбувається, якщо змінна має таке саме ім’я, як і POU, дія або тіло коду переходу, а також, якщо вона є структурою або масивом.

Можна замінити змінну на таку, яка вже оголошена в робочому листку сітки змінних. Можна також замінити ім’я змінної на нове і таким чином декларувати при заміні нову змінну.

Щоб замінити змінну:

– позначте змінну, яка змінюється, курсором і натисніть <F5>. З'явиться діалогове вікно Variables (Змінні);

– для заміни поточної змінної на вже декларовану змінну виберіть у діалоговій сторінці Variables (Змінні) зі списку змінних бажане ім'я і далі продовжуйте, як при вставленні вже оголошеної змінної.

Для заміни поточної змінної новою, у діалоговій сторінці Variables (Змінні) введіть нове ім'я, а потім продовжуйте, як при декларуванні нової змінної.

Для зміни властивостей поточної змінної відкрийте діалогову сторінку Common (Загальний), зробіть необхідні заміни і закрийте діалогове вікно.

Щоб в текстовому редакторі викликати функцію, використовується її ім'я у якості оператора та відповідні параметри, як це показано в наступному прикладі:

```
LD Inpar1
Funktion name   par2,par3
ST  var1
```

Перший декларований вхідний параметр «inpar1» завантажується у попередній рядок викликаної функції. Усі інші вхідні параметри «par2, par3» записуються через кому в другому рядку, як операнди. Результат зберігається змінною «var1», як це показано в останньому рядку прикладу.

Більш зручніше і надійніше редагування викликаної функції здійснювати за допомогою Майстра редагування (Edit Wizard). Якщо Майстер редагування невидимий на екрані, натисніть <Shift>+<F2>або <Alt>+<3> або в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування).

– Виберіть в робочому листку тіла коду позицію, де нова функція має бути вставлена, і клацніть лівою клавішею миші;

– відкрийте список Group (Група) у вікні Майстра редагування і виберіть курсором Functions (Функції). З'явиться список доступних функцій;

– виберіть бажану функцію і подвійно клацніть по ній лівою клавішею миші. Функція автоматично з'явиться у зазначеному місці робочого поля;

– замініть зелені коментарі (оточені круглими дужками і зірочками) необхідними елементами.

Наступний приклад показує вже редаговану функцію GE, яка вставлена в робочий листок, використовуючи Майстра редагування.

```
LD   (* IN1 as ELEMENTARY *)
GE   (* IN2 as ELEMENTARY *)
ST   (* Result as BOOL *)
```

Функціональні блоки викликаються в IL-редакторі за допомогою оператора CAL й імені зразка функціонального блока, як операнда.

Уявимо собі функціональний блок „FB_exam", екземпляр якого має назву „instance", з двома вхідними параметрами „inpar1" та „inpar2" і двома вихідними параметрами „outpar1" та „outpar2". Тоді синтаксис виклику функціонального блока має вигляд:

```
LD  var1
ST  instance. inpar1
LD  var2
ST  instance. inpar2
CAL FB_exam
LD  instance. outpar3
ST  var3
LD  instance. outpar4
ST  var4
```

Тобто виклик функціонального блока складається з трьох частин: введення вхідних параметрів, відповідний виклик оператором CAL, зберігання вихідних параметрів.

Більш зручніше, і це перешкоджає появі помилок, викликати функціональний блок за допомогою Майстра редагування (Edit Wizard).

Якщо Майстер редагування невидимий на екрані, натисніть <Shift>+<F2> або <Alt>+<3>,або в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування) .

– Виберіть позицію в тілі коду, де буде вставлений новий функціональний блок. Клацніть лівою клавішею миші;

– відкрийте список Group (Група) у вікні Майстра редагування (Edit Wizard) і виберіть групу Function blooks (Функціональні блоки). Майстер редагування покаже доступні функціональні блоки;

– подвійно клацніть по бажаному функціональному блоку. З'явиться діалогове вікно Variables (Змінні), де у полі Name (Ім'я) за умовчанням пропонується ім'я екземпляра блока (наприклад, для функціонального блока CTU пропонується ім'я „CTU_n" де n – порядковий номер цього імені екземпляра).

Щоб призначити екземпляру функціонального блока бажане ім'я, необхідно:

– ввести у перший рядок діалогової сторінки Variables (Змінні) нове ім'я зразка або вибрати ім'я з текстового поля Name (Ім'я);

– натиснути кнопку Apply (Застосувати);

– у діалоговій сторінці Common (Загальний) ввести, при необхідності, коментар, що стосується нового функціонального блока;

– у діалоговій сторінці Local scope (Локальна компетенція) позначте Default (За умовчанням), щоб вставити декларацію зразка у робочий листок сітки змінних;

– натисніть ОК, і функціональний блок автоматично вставиться у позицію, вказану текстовим курсором, а його декларація автоматично увійде у групу локальних змінних ;

– змініть зелені коментарі (оточені круглими дужками і зірочками) необхідними елементами.

Наступний приклад показує вже редагований CTU- функціональний блок, викликаний за допомогою Майстра редагування.

```
LD   (* BOOL *)
ST   CTU_1.CU
LD   (* BOOL *)
ST   CTU_1.RESET
LD   (* INT *)
ST   CTU_1.PV
CAL  CTU_1
LD   CTU_1.Q
ST   (* BOOL *)
LD   CTU_1.CV
ST   (* INT *)
```

Вже редагований функціональний блок CTU з іменем CTU_1

Зелені коментарі займають місця, які мають бути переписані фактичними значеннями й іменами

В IL робочих листках можуть використовуватися стрибки, які редагуються за допомогою оператора JMP, а також модифікаторів „C” або „CN” і мітки.

Щоб редагувати стрибки:

– встановіть текстовий курсор у бажане місце робочого листка тіла коду;
– введіть оператор JMP, додаткові модифікатори і мітку;
– виберіть в тілі коду місце, куди доведеться ставити мітку;
– введіть мітку, двокрапки й інструкцію.

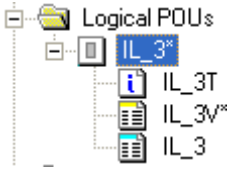
Приклад використання стрибка має вигляд:

```
LD    var1
EQ    INT#100
JMPC  label
LD    var2
ADD   var3
ST    var4
Label:LD %IX2.2
```

Система програмування може здійснити перекомпіляцію існуючого POU, створеного IL-, FBD- або LD-мовою, кожною з інших двох мов цього переліку.

Для реалізації мовної конверсії:

– переконайтесь, що проект вже побудований і компільований;
– у проектному дереві відкрийте піддерево Logical POUs (Логічні POUs) і позначте ікону існуючого POU;



- клацніть правою клавішею миші по іконі POU, щоб відкрити його контекстне меню;
– виберіть пункт Source conversion (Початок конверсії) і клацніть лівою клавішею. З’явиться діалогове вікно Source conversion (Початок конверсії).

– У діалоговому вікні Source conversion (Початок конверсії), що з'явилося, активізуйте перемикач Overwrite source POU (Переписати початковий POU), якщо бажаєте, визначте нове ім'я POU і виберіть мову перекомпіляції.

– Закрийте діалогове вікно і у дереві проекту на місці позначеного POU з'явится новий POU - конвертований вибраною мовою.

- В панелі інструментів клацніть лівою клавішею миші по іконі Compile Worksheet (Компілювати Робочий листок), щоб компілювати POU після конверсії.

Контрольні запитання

1. Як вставити IL-інструкцію у робочий листок тіла коду?
2. Як замінити змінну на таку, що вже оголошена?
3. Як у текстовому редакторі викликати функцію?
4. Як у текстовому редакторі викликати функціональний блок?
5. Як редагуються стрибки в робочих листках тіла коду?
6. До яких мов можна застосувати перекомпіляцію?

2.2.4. **Редактор ST – мови**

Робочі листки тіла коду редагуються ST–мовою, використовуючи текстовий редактор [6]. Інструкції та вирази користувач може друкувати або вставляти за допомогою Майстра редагування. Зручніше користуватися Майстром редагування тому, що він містить багато стандартних ключових слів, функцій та функціональних блоків, а це перешкоджає появі помилок при створені ST-програми. При друкуванні інструкцій та виразів рекомендується використовувати абзаци. Кожний рядок починається з номера, а кожна інструкція має закінчуватися крапкою з комою. При використанні Майстра редагування це здійснюється автоматично.

Коментарі створюються у круглих дужках із зірочками. Різні елементи синтаксису мають свій колір: ключові слова - блакитні, змінні та імена екземплярів блоків – чорні, коментарі – зелені.

Для редагування ST-мовою необхідно в проектному дереві відкрити робочий листок тіла коду, подвійно клацаючи по відповідній іконі теки Logical POUs(Логічні POUs).

Якщо Майстра редагування на екрані не видно, натисніть <SHIFT> + <F2> або <Alt>+<3> або в панелі інструментів клацніть по іконі Edit Wizard (Майстра редагування) .

Щоб зробити вставку інструкції:

– у робочому листку тіла коду визначте курсором місце, де нова інструкція має бути вставлена;

– відкрийте вікно Group (Група) Майстра редагування і виберіть Keywords (Ключові слова), де приведено список ключових слів;

– виберіть зі списку бажану інструкцію, наприклад CASE, і двічі клацніть по ній лівою клавішею миші. Синтаксис інструкції автоматично вставиться в робоче поле тіла коду:

```
CASE (*EXPRESSION (must return an INT value*) OF
    (* VALUE  a*):    (*STATEMENTS*);(* VALUE can be a single value  *)
    (* VALUES b*):    (*STATEMENTS*);(* or a set of VALUES          *)
    (*      .    :         .      *) (* for Example:         *)
    (*      .    :         .      *) (*              1   : ....; *)
    (* VALUE  x*):    (*STATEMENTS*);(*              2..4: ....; *)
    ELSE (*STATEMENTS*);
END_CASE;
```

– у наведеній мовній структурі замість фактичних змінних і їх значень приведені зеленим текстом коментарі, які оточені круглими дужками і зірочками.

При редагуванні коментарі замінюються фактичними операндами (змінними та значеннями).

Щоб декларувати нову змінну, надрукуйте її ім'я у бажаній позиції тіла коду і встановіть на неї курсор;

– в панелі інструментів клацніть по іконі Variable (Змінна), з'явиться діалогове вікно Variables (Змінні), де у першому рядку поля Name (Ім'я) показане ім'я позначеної змінної;

- натисніть кнопку Apply (Застосувати) і автоматично відкриється діалогова сторінка Common (Загальний);

– На сторінці Common (Загальний) виберіть відповідні установи для змінної (Usage, Data Type та ін.) і натисніть кнопку Apply (Застосувати), підтверджуючи їх.

Щоб визначити компетенцію нової змінної, відкрийте діалогову сторінку Local scope (Локальна компетенція), якщо декларується локальна змінна або Global scope (Глобальна компетенція) у разі глобальної змінної.

У піддереві, що з'явилося, позначте лівою клавішею миші Default (За умовчанням).

Примітка. Якщо декларується глобальна змінна (тобто використовуються VAR_EXTERNAL або VAR_EXTERNAL_PG), треба вибирати як локальну, так і глобальну компетенцію.

Після вибору компетенції натисніть ОК і змінна вставиться у робочий листок тіла коду, а її декларація автоматично увійде у робочий листок сітки змінних. Коли декларується глобальна змінна, вона записується у сітку локальних змінних з ключовим словом VAR_EXTERNAL, а у робочий листок сітки глобальних змінних - з ключовим словом VAR _GLOBAL.

Автовставка декларації змінної не відбувається, якщо змінна має таке саме ім'я, як і POU, як тіло коду дії або переходу у SFC-програмі, а також, якщо змінна зображена структурою чи масивом.

При редагування ST – тіла коду можна використовувати змінні, які вже декларовані.

Щоб вставити змінну, яка вже декларована:

– встановіть текстовий курсор у таке місце тіла коду, де бажаєте вставити змінну;

– в панелі інструментів клацніть по іконі Variable (Змінна), з'явиться діалогове вікно Variables (Змінні).

Вікно списку змінних у полі Name (Ім'я) містить всі локальні або глобальні змінні, які декларовані у відповідному локальному чи глобальному робочому листі змінних.

- Клацніть по бажаній змінній, що у списку поля Name (Ім'я), і позначене ім'я увійде у перший рядок поля Name (Ім'я);

– закрийте діалогове вікно і змінна вставиться у робочий листок тіла коду.

Можна замінити змінну на змінну, яка вже декларована в робочому листку сітки змінних.

Для цього:

- двічі клацніть по змінній, яка замінюється, щоб позначити її;

- натисніть <F5>. З'явиться діалогове вікно Variables (Змінні);

- для заміни змінної вже декларованою змінною виберіть бажане ім'я у вікні списку змінних діалогової сторінки Variables (Змінні). Якщо потрібно, визначтесь з компетенцією, використовуючи сторінки Local scope (Локальна компетенція) або Global scope (Глобальна компетенція);

– для зміни властивостей поточної змінної відкрийте діалогову сторінку Common (Загальний) і зробіть необхідні корективи (Usage, Data Type та ін.), і натисніть кнопку Apply (Застосувати), підтверджуючи їх;

– закрийте діалогове вікно, натиснувши ОК.

Для того, щоб викликати функцію, використовують її ім'я. При цьому до імені функції ще додаються імена вхідних змінних у круглих дужках:

Ім'я вихідної змінної : = функціональне ім'я (var 1, var 2);

Більш зручніше викликати функцію, використовуючи Edit Wizard (Майстер редагування).

Якщо Майстер редагування невидимий на екрані, натисніть <SHIFT> + <F2> або <ALT> + <3>, або в панелі інструментів клацніть по іконі Edit Wizard (Майстра редагування) .

Щоб викликати функцію, використовуючи Edit Wizard (Майстра редагування):

– в робочому листку тіла коду курсором виберіть місце, куди нова функція має бути вставлена;

– відкрийте вікно Group (Група) Майстра редагування і виберіть Functions (Функції);

– клацніть лівою клавішею миші, майстер редагування покаже список функцій;

– виберіть бажану функцію і двічі клацніть по ній, функція автоматично вставиться у позицію, яку показує текстовий курсор;

– замініть зелені коментарі , що оточені круглими дужками і зірочками, необхідними елементами.

Функціональні блоки викликаються за іменами їх екземплярів, а також вказуються їх входи і виходи. Виклик екземпляра функціонального блока «instance» має вигляд:

instance (invar 1: = 1 invar2: = 2);
a: = instance. outvar 1.

Більш зручніше викликати функціональний блок, використовуючи Майстра редагування.

Якщо Майстра редагування не видно на екрані, натисніть <SHIFT> + <F2> або <ALT> + <3>, або в панелі інструментів клацніть по іконі Edit Wizard (Майстра редагування) .

Щоб викликати функціональний блок, використовуючи Майстра редагування:

– виберіть курсором місце у робочому листку тіла коду, де новий функціональний блок має бути вставлений;

– відкрийте вікно Group (Група) Майстра редагування і виберіть Functions blocks (Функціональні блоки). Майстер редагування (Edit Wizard) покаже список функціональних блоків;

- подвійно клацніть по бажаному функціональному блоці і з'явиться діалогове вікно Variables (Змінні) з іменем екземпляра функціонального блока у першому рядку поля Name (Ім'я). Наприклад, для функціонального блока CTU, Майстер редагування (Edit Wizard) пропонує ім'я CTU_n, де n – порядковий номер імені екземпляра блока).

Для визначення нового імені необхідно:

- надрукувати у першому рядку поля Name (Ім'я) нове ім'я екземпляра функціонального блока або вибрати вже існуюче ім'я зі списку імен поля Name (Ім'я) і клацнути по ньому лівою клавішею миші;

– на сторінці Common (Загальний), що з'явилася, введіть коментар по новому функціональному блоку, якщо бажаєте;

– на сторінці Local Scope (Локальна компетенція) діалогового вікна Variables (Змінні) позначте Default (За умовчанням), щоб вставити в робочий листок сітки змінних декларацію екземпляра функціонального блока;

– натисніть ОК, і мовна структура функціонального блока автоматично вставиться у робочий листок тіла коду, а його декларація увійде в групу локальних змінних робочого листка сітки змінних;

– замініть зелені коментарі, що оточені круглими дужками і зірочками, необхідними елементами ST-мови.

Наступний приклад показує функцію MAX і функціональний блок CTU, що вставлені, використовуючи Майстра редагування (Edit Wizard).

```
(* Result as ELEMENTARY *):=MAX((* IN1 as ELEMENTARY *),
(* IN2 as ELEMENTARY *));

CTU_1(CU:=(* BOOL *),RESET:=(* BOOL *),PV:=(* INT *));
(* BOOL *):=CTU_1.Q;
(* INT *):=CTU_1.CV;
```

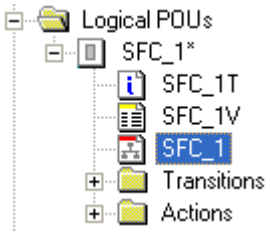
Контрольні запитання

1. Як відкрити робочий листок тіла коду для редагування ST–мовою?
2. Як декларувати нову змінну?
3. Як можна використати вже декларовану змінну при редагуванні тіла коду?
4. Як викликаються функції та функціональні блоки?
5. Які можливості у Edit Wizard (Майстра редагування)?

2.2.5. Редактор SFC – мови

Одною з головних особливостей редагування SFC – мовою є те, що система програмування завжди прагне створити закриту SFC-мережу [6]. Коли вставляються нові кроки і переходи, частина мережі під точкою вставлення автоматично зміщується вертикально вниз. Проте структуру SFC –мережі необхідно організовувати так, щоб був і достатній простір для її розширення убік.

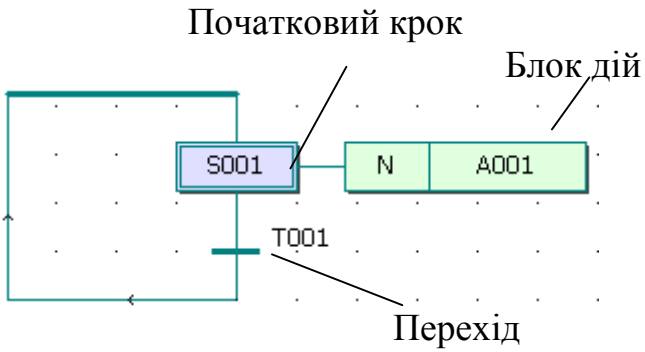
Для редагування робочого листка SFC- коду, необхідно відкрити його подвійним клацанням лівої клавіші миші по відповідній іконі теки Logical POUs(Логічні POUs) у дереві проекту:



Протягом редагування робочого листка створюється SFC- мережа, яка складається з кроків і переходів, редагованих LD-,FBD-,IL- або ST-мовою. Робочі листки тіла коду кроків і переходів розташовуються у відповідній піддиректорії проектного дерева і називаються деталлю.

Для створення SFC- мережі у робочому листку тіла коду курсором позначте місце вставки;

– в панелі інструментів клацніть по іконі Insert step transition sequence (Вставити послідовність крок - перехід) і на екрані з'явиться SFC- мережа з одним кроком і одним переходом. Перший вставлений крок автоматично є початковим кроком, він має подвійну рамку.



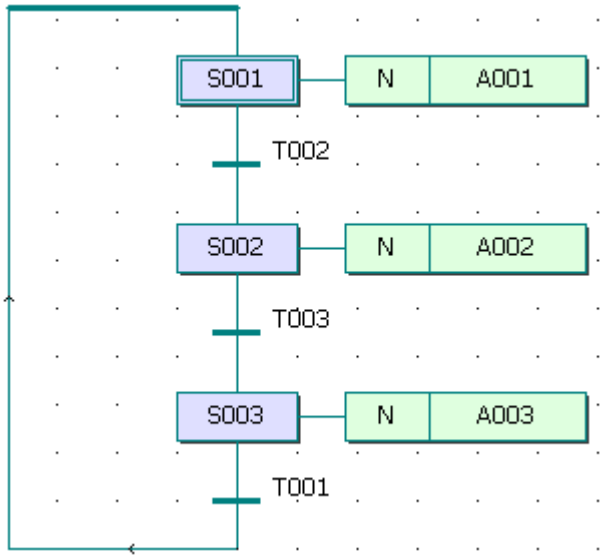
Щоб розширити існуючу мережу за рахунок нових кроків і переходів:

– клацніть лівою клавішею миші по бажаному кроці, щоб позначити місце розширення мережі;

– в панелі інструментів клацніть по іконі Insert step transition sequence (Вставити послідовність крок - перехід) і інша пара (крок-перехід) з'явиться між позначеним кроком і наступним переходом.

Якщо позначити перехід замість кроку і вставити нову послідовність (крок – перехід), то вона вставиться нижче позначеного переходу.

У наступній SFC-мережі дві нових послідовності (крок – перехід) вставлені між кроком SOO1 і переходом TOO1 після того, як був позначений крок SOO1 і в панелі інструментів двічі натиснута ікона Insert step transition sequence (Вставити послідовність крок-перехід).



Ширина кроків і дій SFC-мережі завжди однакова, якщо у підменю Loyout (Розташування) альтернатива Fixed SFC objects Width (Постійна ширина SFC об'єкта)- активізована. Якщо ні, ширина їх коригується автоматично відповідно до довжини імені кроку або дії.

При необхідності можна змінити початковий крок на звичайний, а звичайний – на початковий.

Щоб зробити зміну:

– двічі клацніть лівою клавішею миші по кроці, який бажано змінити. З’явиться діалог Step (Крок);

– активізуйте перемикач Initial step (Початковий крок), щоб змінити звичайний крок на початковий;

– вимкніть перемикач Initial step (Початковий крок), щоб змінити початковий крок на звичайний. Звичайний крок зображається в робочому листку прямокутником з одинарною рамкою.

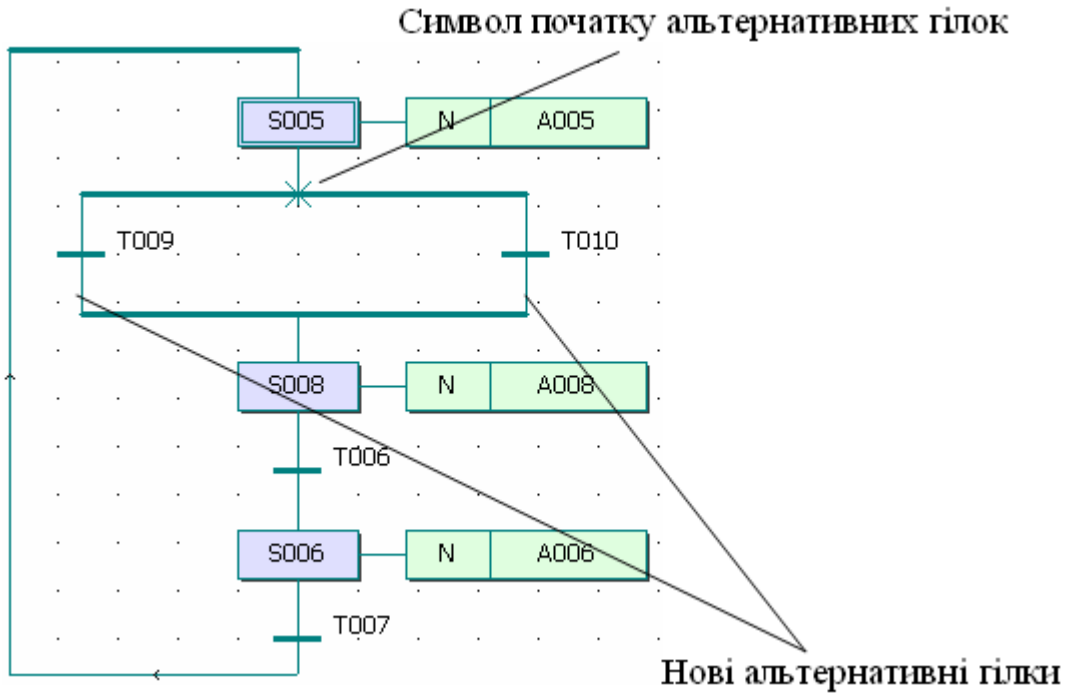
Протягом редагування можна вставити у SFC–мережу альтернативні або паралельні гілки.

Щоб вставити альтернативні гілки:

– позначте крок, де бажано вставити гілки;

– в панелі інструментів клацніть по іконі Insert Simultaneous/Alternativ Branches (Вставка Одночасної/Альтернативної гілки). З’явиться діалог Divergence (Розходження);

– введіть бажану кількість гілок у вікні діалогу Branches Count (Рахунок гілок) і натисніть ОК:

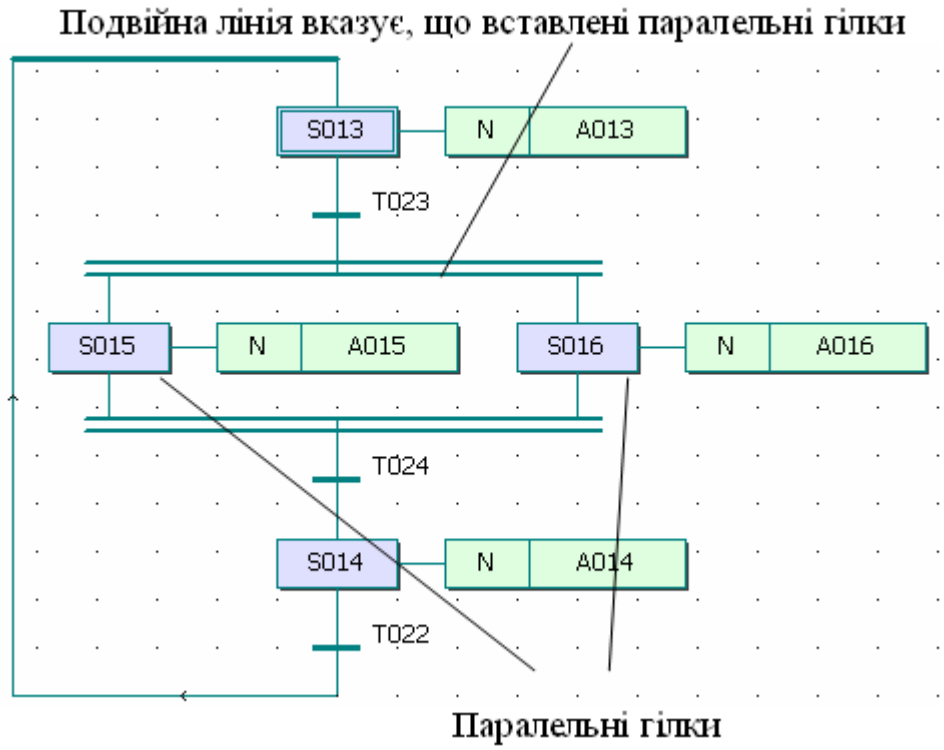


Щоб вставити паралельні гілки:

– позначте перехід, де бажаєте вставити гілки;

– клацніть по іконі Insert Simultaneous/Alternativ Branches (Вставка Одночасної/Альтернативної гілки);

– введіть бажану кількість гілок у вікні діалогу Branches Count (Рахунок гілок) і натисніть ОК:



Паралельні гілки

Якщо бажаєте збільшити кількість гілок, повторіть процедуру встановлення розходження в SFC-мережі і в панелі інструментів клацніть по іконі Insert SFC branch (Вставка SFC – гілки). У діалозі Divergence (Розходження) скоригуйте кількість гілок.

Вставлення гілок можна здійснювати, використовуючи режим редагування SFC – гілок.

Щоб вставити паралельні гілки:

- в панелі інструментів клацніть лівою клавішею миші по іконі Insert SFC branch (Вставка SFC – гілки). З'явиться курсор у вигляді SFC – гілки;
- спрямуйте курсор на перехід, після якого має бути розходження гілок і клацніть лівою клавішею миші. З'явиться подвійна лінія. Перемістіть курсор вбік на вільне місце, де бажаєте встановити новий крок з блоком дії;
- клацніть лівою клавішею миші, щоб зафіксувати верхню подвійну лінію;
- перемістіть курсор на перехід, де бажаєте встановити нижню подвійну лінію, і клацніть лівою клавішею миші.

Режим редагування SFC – гілок завершується автоматично після того, як нова гілка вставлена. Для вставлення іншої гілки повторно виберіть режим редагування SFC - гілок, знову клацаючи по іконі Insert SFC branch (Вставка SFC – гілки).

Щоб вставити альтернативну гілку, треба повторити такі самі дії, як і у попередньому варіанті, тільки спочатку спрямуйте курсор у вигляді SFC – гілки не на перехід, а на крок.

В існуючих SFC – мережах можна вставити стрибки. Стрибки вказують шлях виконання програми у розімкненій SFC-мережі. Стрибок має ім'я мети, куди він прямує.

Щоб вставити стрибок:

– двічі клацніть лівою клавішею миші по кроці, з якого має бути стрибок. З'явиться діалог Step (Крок);

– у рядку Name(Ім'я) введіть ім'я кроку, на який здійснюється стрибок:

– активізуйте перемикач Jump (Стрибок) і натисніть ОК;

– з'явиться діалог Jump/End step insert control (Стрибок/Кінець управління кроком вставки);

– активізуйте перемикач Show marked objects for delete (Показ позначених об'єктів для видалення) і натисніть ОК, щоб побачити усі об'єкти SFC-мережі, що видаляються при вставленні стрибка;

– знову викличте діалог Jump/End step insert control (Стрибок/Кінець управління кроком вставки);

– натисніть кнопку Change (Заміна) для видалення усіх непотрібних об'єктів і вставлення стрибка з іменем мети.

Можна вставити кінцеві кроки, якщо бажаєте закінчити SFC - мережу.

Для цього:

– клацніть по кроці, який бажаєте зробити кінцевим. З'явиться діалог Step (Крок);

– активізуйте перемикач End step (Кінцевий крок) і натисніть ОК;

– з'явиться діалог Jump/End step insert control (Стрибок/Кінець управління кроком вставки);

- активізуйте перемикач Show marked objects for delete (Показ позначених об'єктів для видалення) і натисніть ОК, щоб побачити усі об'єкти SFC-мережі, що видаляються при вставленні стрибка;

– знову викличте діалог Jump/End step insert control (Стрибок/Кінець управління кроком вставки);

– натисніть кнопку Change (Заміна) для видалення усіх непотрібних об'єктів і вставлення кінцевого кроку.

Можна вставити блок дій до кроку, якщо він був видалений. Можна додати нові блоки дій до одного існуючого і створити, таким чином, у робочому листку тіла коду крок із складеним блоком дій. Щоб видалити існуючий блок дій, клацніть по ньому лівою клавішею миші і натисніть «Delete» на клавіатурі комп'ютера.

Щоб вставити новий блок дії:

- позначте крок, до якого бажаєте вставити новий блок дії або позначте існуючий блок дії, якщо бажаєте мати складений блок дій;

– в панелі інструментів клацніть по іконі Create action (Створити дію). З'явиться діалогове вікно Action Properties (Властивості дії);

– введіть нове ім’я дії, якщо бажаєте і натисніть ОК. Новий блок дії з’явиться у робочому листку тіла коду.

Робочий листок тіла коду дії, так звану деталь, можна присвоїти будь-якому кроку існуючої SFC – мережі.

Для цього:

– позначте крок, якому бажаєте присвоїти нову деталь;

– в панелі інструментів клацніть по іконі Create action (Створити дію) . З’явиться діалог Action Properties (Властивості дій);

– у першому рядку поля Name (Ім’я) введіть ім’я нового робочого листка тіла коду (деталі);

– активізуйте кнопку Detail (Деталь) і натисніть ОК.

Позначений крок отримує нову дію з власним іменем.

Дії можуть мати логічну змінну або тіло коду.

Протягом редагування проекту можна змінити властивості кроку **і** ***блоку дії, а також переходу***.

Для зміни властивостей кроку:

– двічі клацніть по кроку лівою клавішею миші. З’явиться діалогове вікно Step(Крок);

– змініть ім’я або тип кроку і натисніть ОК.

Щоб змінити властивості блока дій:

– правою клавішею миші клацніть по блоку дії і виберіть пункт контекстного меню Objects Properties (Властивості об’єкту). З’явиться діалогове вікно Action Properties (Властивості дії);

– зробіть зміну імені або класіфікатора і натисніть ОК;

Для зміни властивостей переходу:

– клацніть правою клавішею миші по переходу, щоб викликати контекстне меню, і виберіть пункт Objects Properties (Властивості об’єкту). З’явиться діалог Transition (Перехід);

– змініть ім’я або тип переходу і натисніть ОК.

Переходи можуть мати деталі або прямі зв’язки. Коли перехід має власне тіло коду, тобто деталь, у діалозі Transition (Перехід) треба активізувати тип Detail (Деталь).

Прямий зв'язок означає, що перехід немає тіла коду, і він безпосередньо з’єднується з LD – або FBD – мережею, або має змінну. У цьому випадку в діалозі Transition (Перехід) потрібно активізувати тип Direct connection (Прямий зв’язок). У разі прямого зв’язку перехід не має імені.

Для створення деталі дії або переходу в робочому листку SFC-коду подвійно клацніть по дії або переходу, з’явиться діалогове вікно Insert (Вставити);

– виберіть мову програмування деталі;

– натисніть ОК і робочий листок дії або переходу відкриється для редагування і одночасно з’явиться у проектному дереві.

Дії в робочих листках тіла коду можуть бути зображені логічними змінними.

Щоб присвоїти дії ім'я змінної або замінити на ім'я змінної, яка вже декларована:

– правою клавішею миші клацніть по дії, щоб відкрити контекстне меню;

– виберіть пункт Object Properties (Властивості об'єкту). З'явиться діалог Action Properties (Властивості дії);

– у діалоговій сторінці Action (Дія) активізуйте кнопку Variable (Змінна);

– зі списку поля Name (Ім'я), що містить усі оголошені змінні, виберіть бажану і клацніть по ній лівою клавішею миші;

– відкрийте сторінку Local Scope (Локальна компетенція), якщо бажаєте оголосити локальну змінну або Global Scope (глобальна компетенція), якщо бажаєте оголосити глобальну змінну, і активізуйте Default(За умовчанням).

– Натисніть ОК і змінна буде вставлена у блок дії.

Щоб декларувати нову змінну дії:

- клацніть по дії правою клавішею миші, щоб відкрити контекстне меню;

– виберіть Object Properties (Властивості об'єкту) і клацніть лівою клавішею миші. З'явиться діалог Action Properties (Властивості дії);

– у діалоговій сторінці Action (Дія) активізуйте перемикач Variable (Змінна);

– у першому рядку поля Name (Ім'я) надрукуйте ім'я дії і клацніть по Apply (Застосувати). Діалогова сторінка Common (Загальний) відкриється автоматично;

– виберіть відповідні установки для змінної і клацніть по Apply (Застосувати), щоб підтвердити їх.

Відкрийте сторінку Local scope (Локальна компетенція), якщо декларується локальна зміна, або Global scope (Глобальна компетенція) у разі глобальної змінної і активізуйте Default (За умовчанням).

Якщо оголошується глобальна змінна (тобто використовуєте VAR_EXTERNAL або VAR_EXTERNAL_PG), потрібно вибрати як локальну, так і глобальну компетенцію.

Після вибору компетенції клацніть ОК, змінна вставиться у робочий листок тіла коду. Автоматично її декларація потрапляє у робочий листок відповідної сітки змінних. У разі глобальної змінної декларація вставляється у сітку локальних змінних, використовуючи ключове слово VAR_EXTERNAL, а у робочий листок сітки глобальних змінних, використовуючи VAR_GLOBAL.

У разі декларування змінних переходів можна вставити змінну вже приєднаною до переходу або з'єднати з переходом змінну, яка встановлена у будь-якому місці робочого листка тіла коду.

Щоб присвоїти переходу ім'я вже декларованої змінної або надати йому нове ім'я, або замінити існуюче ім'я переходу:

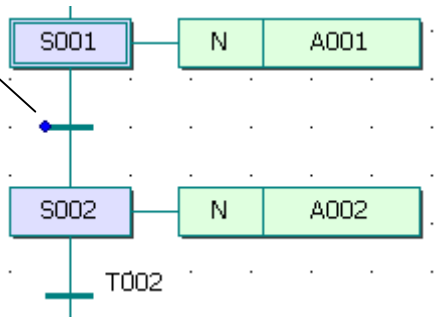
– клацніть по переходу правою клавішею миші, щоб відкрити контекстне меню;

– виберіть пункт Object Properties (Властивості об’єкту). З’явиться діалогове вікно Transition (Перехід);

– активізуйте перемикач Direct connection (Прямий зв'язок) і натисніть ОК.

Перехід показується синьою точкою з’єднання.

Синя точка



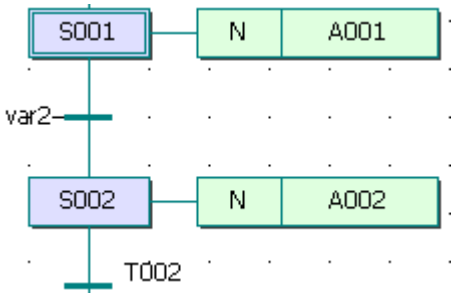
– Позначте перехід, лівою клавішею миші;

– в панелі інструментів клацніть по іконі Variable (Зміна), з’явиться діалогове вікно Variable Properties (Властивості змінної);

– у вікні вже декларованих змінних поля Name (Ім’я) клацніть по бажаному імені, і воно увійде у перший рядок. Якщо немає - надрукуйте нове ім’я.

Коли переходу надаєте нове ім’я на сторінці діалогу Common (Загальний) виберіть відповідні установки і натисніть Apply (Застосувати). Визначтесь з компетенцією змінної на сторінках Local scope (Локальна компетенція) і Global scope (глобальна компетенція).

Закрийте діалогове вікно і змінна потрапить у робочий листок тіла коду, а її декларація автоматично потрапить у відповідний робочий листок сітки змінних:



Замість змінних до переходу можуть бути приєднані функції або LD- мережі за допомогою прямих зв’язків. У цих випадках спочатку необхідно в робоче поле тіла коду вставити функцію чи LD- мережу, а потім з’єднати її з точкою зв’язку переходу, активізуючи в панелі інструментів ікону Connect Objects (З’єднати об’єкти).

При видаленні фрагментів SFC-коду, в які входять більш ніж один крок і один перехід, або цілі гілки, на робочому полі тіла коду іноді залишаються дуже довгі лінії сполучення. У цих випадках довжину ліній можна оптимізувати автоматично.

Для цього:

– позначте за допомогою лівої клавіші миші частину лінії, яку потрібно оптимізувати;

– у підменюEdit>Stretch/Compress (Редагування>Розтягання/ Стиснення) виберіть пункт Optimize SFC Lines (Оптимізація SFC –лінії) і довжина ліній оптимізується автоматично.

Контрольні запитання

1. Як створити першу SFC-мережу?
2. Як розширити існуючу мережу?
3. Як змінити початковий крок на звичайний, а звичайний – на початковий?
4. Як вставити в SFC-мережу альтернативні гілки?
5. Як вставити в SFC-мережу альтернативні гілки?
6. Як вставити в SFC-мережу паралельні гілки?
7. Як користуватися режимом редагування SFC – гілок?
8. Як вставити в SFC-мережу стрибки?
9. Як додати нові блоки дій до одного існуючого?
10. Як змінити властивості кроку, блоку дії, переходу?
11. Як створити деталь дії або переходу?
12. Як вставити змінну вже приєднаною до переходу?
13. Як до переходу можуть бути приєднані функції або LD- мережі?
14. Як видалити фрагменти SFC-коду?

3. ПРОГРАМУВАННЯ У СИСТЕМІ MULTIPROG

3.1. Розробка проекту – загальні кроки

Щоб реалізувати проект, необхідно зробити декілька загальних кроків.

Крок 1. Визначення структури проекту у проектному дереві

Визначення структури проекту у проектному дереві означає вставлення у проект необхідних одиниць організації програми, так званих POUs, якими можуть бути програми, функціональні блоки і функції.

Крок 2. Редагування POUs

Редагування POUs означає декларування параметрів і змінних та редагування тіла коду за допомогою графічного та текстового редакторів і Edit Wizerd (Майстр Редагування).

Крок 3. Зв'язок програм і задач

Для того, щоб генерувати код до контролера, кожну програму необхідно зв'язати з задачею, яка транслює програму до ПЛК. Для цього у проектному дереві треба позначити теку задачі і в панелі інструментів клацнути по іконі Add Object (Додати об'єкт), щоб викликати діалогове вікно Insert (Вставка). У діалоговому вікні необхідно активізувати перемикач Program (Програма), ввести ім'я зразка програми і вибрати бажану програму у вікні списку Program Type (Тип Програми).

Після підтвердження діалогу зразок програми автоматично вставиться у теку задачі.

Крок 4. Компіляція проекту

Коли редагування проекту закінчено, він компілюється, використовуючи в панелі інструментів ікону Compile Worksheet (Компілювати Робочий лист). При цьому тіло коду перевіряється на правильний синтаксис.

Крок 5. Завантаження

Компільований проект має бути завантаженим до ПЛК або симулятора. Для цього в панелі інструментів необхідно клацнути по іконі Project Control Dialog (Діалог Управління Проектом). Відкриється діалогове вікно Resource (Ресурс), в якому натисніть Download (Завантаження).

Крок 6. Налагодження програми ПЛК

Наявність завантаженого проекту дає змогу перевірити програму, використовуючи оперативний режим налагодження. Перехід у цей режим роботи здійснюється за допомогою ікони Debug on/off (Налагодження В/В) в панелі інструментів після того, як відповідні робочі листки будуть відкриті (робочий листок тіла коду програми або робочий листок сітки змінних). За допомогою команд налагодження Step (Крок) і Trace (Відслідковувати), а також встановлюючи контрольні точки і, активізуючи змінні, можна відстежити коректність виконання програми.

Крок 7. Друкування проекту

Для створення документації корисно надрукувати проект. Тому система забезпечує могутній інтерфейс друкування, який дозволяє використовувати

вбудований макет сторінок і вибирати для друку необхідні листки замість всього проекту.

У деяких випадках, для більш зручнішого створення проекту, можна використовувати в панелі інструментів ікону Patch POU (Вставка POU) замість Make (Створювати). Після створення основи проекту і завантаження його до ПЛК або симулятора можна поповнювати програму шляхом редагування різних робочих листків коду POU за рахунок, так би мовити, латання їх в процесі виконання контролером програми [6].

3.2. Методика програмування LD-мовою

Знайомство з програмуванням у системі MULTIPROG почнемо на прикладі розробки програми LD-мовою (Крокових діаграм), яка здійснює циклічне управління роботою двигуна, що вмикається триразовим натисненням стартової кнопки, а вимикається автоматично за 20с роботи [6].

Розробка проекту складається з кількох етапів, по яких веде Project Wizard (Майстер проекту) системи програмування MULTIPROG..

Для створення проекту запустіть систему програмування, двічі натиснувши піктограму MULTIPROG.

Щоб отримати кращий результат в процесі створення проекту, будемо використовувати запропоновані ідентифікатори й імена.

3.2.1. Створення нового проекту

Визначимо ім’я та шлях створення проекту, мову програмування і тип ПЛК, що використовується.

Використовуючи пункт меню File (файл), відкрийте діалогове вікно New Project (Новий проект), рис. 3.1;

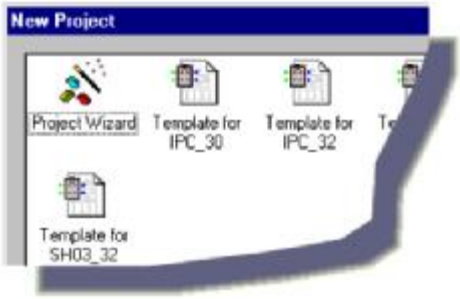


Рис. 3.1. Фрагмент діалогового вікна

- двічі клацніть по Project Wizard (Майстер проекту) і з’явиться відповідний діалог;
- введіть ім’я проекту My_first_Project (Мій перший проект). Майстер проекту збереже проект у відповідному файлі My_first_Project .mwt і створить підтеку того ж імені, де будуть запам’ятовуватися тіло коду і файли змінних.

Максимальна довжина імені проекту може складати 24 символи і не повинна містити пропуски або спецсимволи.

За умовчанням шлях проекту вводиться автоматично. Якщо бажаєте створити інший шлях, натисніть кнопку, на якій є три крапки;

- у діалозі Celect Directory(Вибрати директорію) виберіть теку для нового проекту і клацніть ОК;
- натисніть кнопку Next (Далі) і у рядок Name of POU (Імя POU) введіть ім'я першого організатора програми (POU)- Main (Головний);
- оскільки зразковий проект будемо програмувати графічною LD-мовою активізуйте відповідний перемикач;
- натисніть кнопку Next (Далі) і введіть бажане ім'я конфігурації. У нашому прикладі це - Configuration (Конфігурація);
- у вікні списку виберіть тип конфігурації проекту. У нашому прикладі вибираємо IPC_32 , тому що компілятор генерує intel- код для ProConOS 3.2.
- натисніть кнопку Next (Далі) і введіть ім'я ресурсу – Resource (Ресурс).
- у вікні списку виберіть тип ресурсу - PCOS_NT. Вікно списку пропонує тільки типи CPU , які належать конфігурації, що визначена в діалозі Майстра проекту;
- натисніть кнопку Next (Далі) і введіть ім'я задачі -Task (Задача);
- у вікні списку виберіть тип задачі - Cyclic (Циклічний);
- натисніть кнопку Next (Далі). На екрані з'явиться зміст зроблених установок. Якщо помилок немає, натисніть Finish (Готов), в іншому разі скоригуйте кроки проекту, де є помилка, користуючись кнопкою „Back" (Назад).

Створений новий проект Main з'явиться у вікні проектного дерева, рис.3.2, у піддеревах Logical POUs і Physical Hardware.

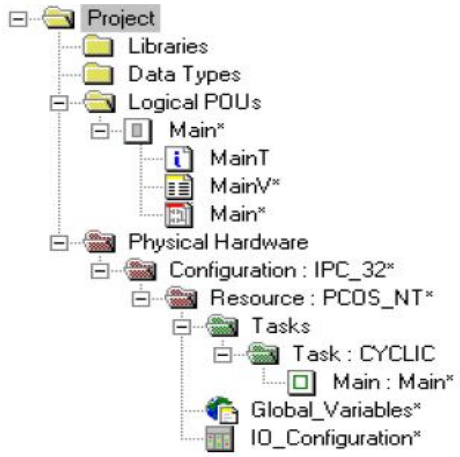


Рис. 3.2. Дерево проекту

3.2.2. **Розробка коду програми**

Тепер, коли новий проект створено і на екрані комп'ютера з'явилось його дерево, розробимо проектний код LD-мовою. У наступних кроках пояснимо, як створити першу LD-мережу, як оголосити властивості об'єктів, що автоматично з'явилися з першою LD- мережею, як вставити і з'єднати функціональний блок у робочому листку тіла LD- коду за допомогою Edit

Wizard (Майстер редагування), як вставити і з'єднати контакт у робочому листку коду програми, як оголосити властивості контактів і котушок, як вставити другу LD-мережу і відредагувати коментар мережі.

Вставлення першої LD- мережі:

- клацніть лівою клавішею миші у зручному місці робочого листка для установки мітки першої вставки;
- в панелі інструментів клацніть по іконі Contact network (Мережа контактів) і в установленому місці з'явиться LD- мережа 001 певної ширини з контактом і котушкою.

Оголошення властивостей змінних, які автоматично з'явилися з першою LD- мережею:

- двічі клацніть лівою клавішею миші по контакту C000, щоб оголосити змінну, яка має запустити двигун. У діалоговому вікні Contact/Coil Properties (Властивості Контакт/Котушка), що з'явилося, замініть типове ім'я змінної C000 на Motor_Start;
- натисніть кнопку Apply (Застосувати), автоматично відкриється сторінка діалогу Common (Загальний). У вікні списку Usage (Вжити) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL.

Таким чином, змінна оголошується як локальна, а тому вона може використовуватися тільки в POU проекту.

Призначимо тепер змінній Motor_Start фізичну адресу симулянта модуля вводу/виводу ПЛК:

- введіть у поле I/O address (Адреса вводу/виводу) %IX0.0, тут 0.0– позначає перший модуль і його перший канал; I – фізичний вхід; X – одно-бітний розмір;
- відкрийте по черзі сторінки діалогів Local scope (Локальна компетенція) і Global scope (Глобальна компетенція) і активізуйте в них Default;
- клацніть ОК, підтверджуючи діалог Contact/Coil Properties (Властивості Контакт/Котушка). Після цього змінна є оголошеною і вставленою у відповідний список, а на робочому листку проекту замість C000 з'явиться глобальна зміна Motor_Start .

Вставлення в LD- мережу лічильника натисків стартової кнопки:

Оскільки двигун має увімкнутися після триразового натиснення стартової кнопки вставимо в LD- мережу лічильник.

- Клацніть по лінії між Motor_Start і C001, щоб позначити місце вставлення лічильника;
- в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування) і з таблиці Group (Група), що з'явилася на екрані, виберіть функціональний блок лічильника CTU;

- двічі клацніть по блоку CTU, у діалоговому вікні Variable properties (Властивості змінної), що з'явилося, змініть типове ім'я CTU-1 на Motor-Count;

- натисніть послідовно кнопки Apply (Застосувати) і ОК, підтверджуючи діалог, на позначеному місці LD- мережі з'явиться лічильник;

- в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування), щоб сховати Майстра редагування.

Вставлення контакту скидання лічильника:

- клацніть на функціональному блоці CTU по синій точці Reset-входу;

- в панелі інструментів клацніть по іконі Add contact left (Додати контакт зліва) і на вході лічильника з'явиться контакт C002;

- активізуйте в панелі інструментів кнопку Connect objects (З'єднати об'єкти);

- клацніть по контакту C002 і перемістіть його курсором ліворуч до шини живлення 001так, щоб він опинився під контактом Motor_Start;

- в панелі інструментів клацніть по іконі Mark (Позначка), щоб деактивізувати кнопку Connect objects (З'єднати об'єкти);

Оголошення властивостей контакту скидання лічильника:

- двічі клацніть по контакту C002 і у діалоговому вікні Contact/Coil Properties (Властивості Контакт/котушка), що з'явилося, змініть ім'я C002 на Motor;

- клацніть по Apply (Застосувати), з'явиться сторінка діалогу Common(Загальний);

- у вікні списку Usage (Вжити) виберіть VAR, тобто змінна оголошується, як локальна і може використовуватися тільки в POU проекту, тип змінної – BOOL виберіть у рядку Data Type (Тип даних);

- для призначення змінній Motor фізичної адреси симулятора модуля вводу/виводу I/0 PLC у рядку I/0 Address введіть %QX0.0,

тут Q – позначає фізичний вихід; 0.0 – перший вихідний модуль і його перший канал; X – позначає однобітний розмір.

Оскільки група змінних визначена раніше, немає потреби конкретизувати Local scope (Локальна компетенція) та Globel scope (Глобальна компетенція);

- клацніть ОК, і на робочому листку з'явиться змінна Motor замість C002;

- для визначення параметрів лічильника двічі клацніть по блакитній точці PV-входу, з'явиться діалогове вікно Variable properties (Властивості змінної);

- у рядку Name (Ім'я) введіть INT#3, щоб двигун (Motor) спрацював після триразового натискання стартової кнопки, тут INT – означає ціле число; # - означає константу; 3 – фактичне значення константи;

- клацніть ОК і на PV-вході лічильника з'явиться INT#3, тобто ціле число є вставленим в тіло коду;

- двічі клацніть по зеленій точці CV-виходу лічильника, з'явиться діалогове вікно Variable properties (Властивості змінної);

- в рядку Name (Ім’я) введіть Pressed - це змінна, яка запам’ятовує поточне значення лічильника;

- натисніть кнопку Apply (Застосувати), відкриється сторінка діалогу Common(Загальний);

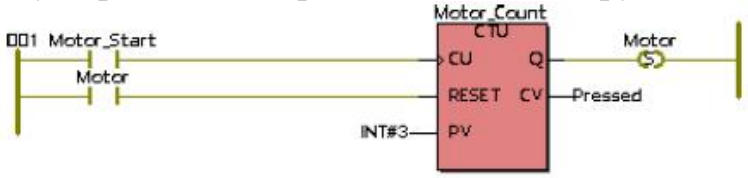
- у рядку Usage (Вжити) виберіть VAR, а У рядку Date type (Тип даних) – INT, оскільки змінною є число;

- клацніть ОК і у робочому листку поряд з CV-виходом з’явиться Pressed;

- двічі клацніть по C001, з’явиться діалогове вікно Contact/Coil Properties (Властивості Контакт/Котушка);

- зі списку вже оголошених змінних, виберіть Motor. Оскільки двигун має працювати безперервно після його увімкнення, у рядку Type (Тип) виберіть –(S)–(увімкнути вихід і зафіксувати);

- клацніть ОК, у першій LD- мережі з’явиться інструкція –(S)–.



Вставлення другої LD- мереж:.

- клацніть лівою клавішею миші нижче першої LD- мережі, на робочому листку з’явиться мітка установки нової мережі;

- в панелі інструментів клацніть по іконі Contact Network (Мережа контактів) і на екрані з’явиться друга LD- мережа;

- двічі клацніть по контакту C003 для оголошення його властивостей;

- у діалозі Contact/Coil Properties (Властивості Контакт/Котушка) із списку вже оголошених змінних виберіть Motor і клацніть лівою клавішею миші;

- клацніть ОК для зміни у LD- мережі імені контакту C003 на Motor .

Вставлення таймера, що керує тривалістю роботи двигуна:

- в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування) , на екрані з'явиться вікно Group (Група);

- клацніть по лінії другої LD- мережі між контактом Motor і C004, щоб позначити місце вставки таймера;

- за допомогою курсору у вікні Group (Група) виберіть Function blocks (Функціональні боки);

- зі списку блоків курсором виберіть таймер TON і двічі клацніть лівою клавішею миші, з'явиться діалогове вікно Variable properties (Властивості змінної);

- у рядку Name (Ім’я) введіть M_Time і послідовно натисніть кнопки Apply (Застосувати) і ОК, підтверджуючи завершення діалогу;

- клацніть по іконі Edit Wizard (Майстер редагування) для приховування вікна Group (Група);

- для визначення часу роботи таймера лівою клавішею миші двічі клацніть по блакитній точці його PT-входу;

- у діалоговому вікні Variable properties (Властивості замінної), що з'явилося, у рядку Name (Ім'я) введіть T#20s;

Тут T – позначає префікс часу; # - означає константу; 20s – час роботи двигуна (20с);

- клацніть ОК і на PT-вході лічильника з'явиться константа часу.

Щоб визначити змінну для ідентифікації фактичного часу роботи таймера двічі клацніть по зеленій точці ET-виходу таймера;

- у діалоговому вікні Variable properties (Властивості змінної), що з'явилося, у рядку Name (Ім'я) введіть Actual_Time (Фактичний час), як ім'я локальної змінної;

- натисніть кнопки Apply (Застосувати), відкриється сторінка діалогу Common (Загальний);

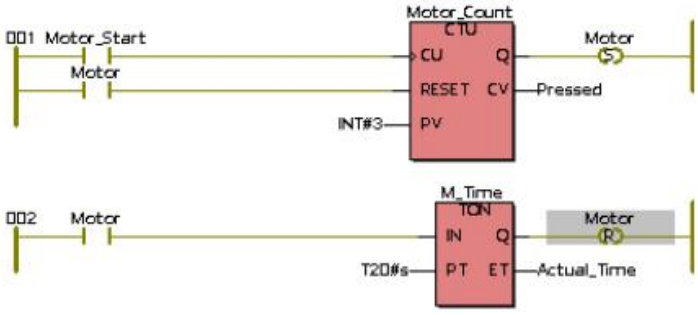
- зі списку Data_Type (Тип даних) виберіть Time (Час);

- клацніть ОК, на ET-виході таймера з'явиться Actual_Time;

- двічі клацніть по котушці C004, відкриється діалогове вікно Contact/Coil Properties (Властивості Контакт/Котушка);

- для зупинки двигуна виберіть курсором зі списку змінних ім'я Motor, а у вікні Type (Тип) - котушку –(R)– (вимкнути вихід і зафіксувати);

- клацніть ОК.



Присвоєння імені розробленої мережі:

- двічі клацніть по лівій шині живлення першої LD- мережі, з'явиться діалогове вікно Comment (Коментар);

- у діалоговому вікні Comment (Коментар) надрукуйте Circuit (Схема) і клацніть Fout>> (Шрифт);

- для зміни властивостей шрифту у вікні, що з'явилося, виберіть блакитний колір і розмір 20;

- клацніть ОК у вікні Fout (Шрифт) і у вікні Comment (Коментар). Над розробленою мережею з'явиться ім'я Circuit.

3.2.3. **Компіляція проекту**

Тепер, коли редагування закінчено, зробимо компіляцію проекту. Протягом компіляції вміст робочих листків відтрансльовується і перетворюється у спеціальний код, який може бути виконаний ПЛК.

Оскільки у цьому прикладі замість ПЛК використовується симулятор, активізуємо його:

- у дереві проекту клацніть по теці Resource (Ресурс);
- клацніть правою клавішею миші і виберіть з контекстного меню Settings (Установки);
- у вікні Resource settings for IRC_32 (Установки ресурсу для IPC-32), що з'явилося, активізуйте Simulation 1 (Симуляція 1), якщо це необхідно і закрийте діалог, клацнувши ОК;
- в панелі інструментів клацніть по іконі Make (Утворювати) ;
- у вікні Build (Конструкція) внизу екрана відобразяться помилки (Error) і попередження (Warning), якщо будуть знайдені протягом компіляції.

Помилки синтаксичного або структурного характеру перешкоджають завершенню процесу компіляції.

Попередження вказують на потенційні проблеми, подібно змінній, якої немає, а вона використовується. Попередження не перешкоджають завершенню процесу компіляції. Попередженнями можна нехтувати, а помилки потрібно виправити.

- Для виводу на екран списку знайдених помилок у вікні Build (Конструкція) клацніть по Error (Помилки);
- для виводу на екран списку попереджень у вікні Build (Конструкція) клацніть по Warning (Попередження);
- двічі клацніть по вказаній помилці або попередженню, відкриється відповідний робочий листок;
- виправіть усі помилки і попередження і повторно компілюйте проект, клацнувши в панелі інструментів по іконі Make (Утворювати);

Тільки після цього можна завантажувати програму до ПЛК або симулятора.

3.2.4. **Завантаження проекту до ПЛК або симулятора**

Зв'язок з ПЛК або симулятором здійснюється завдяки діалогу Resource (Ресурс), який керує контролером.

- Клацніть по іконі Project Control Dialog (Діалогове управління проектом), з'явиться діалогове вікно Resource (Ресурс) для управління ПЛК або симулятором;
- натисніть кнопку Download (Завантаження), з'явиться відповідне діалогове вікно ;
- продублюйте у цьому вікні натиснення на кнопку Download і проект завантажиться у пам'ять ПЛК або симулятор;
- успішний процес завантаження проекту супроводжується бігом синьої стрічки внизу екрана;
- у діалозі управління Resource (Ресурс) натисніть кнопку Cold (Холодний) для холодного пуску симулятора;

Стан діалогового вікна Resource (Ресурс) змінюється кнопками Stop (Зупинка) і Cold (Холодний).

3.2.5. **Налагодження проекту**

Робочі листки проекту можуть перемикатися з режиму редагування на режим налагодження і навпаки, за допомогою ікони Debug on/off (Налагодження В/В) в панелі інструментів. Коли режим налагодження активізовано, стан і поточні значення змінних на робочому листку проекту показані різними кольорами: блакитний – хибний, червоний – істинний.

Режим налагодження використовується для знаходження помилок програмування і переконання, що програма ПЛК працює правильно. Для запуску програми у діалозі управління Resource (Ресурс) натисніть кнопку Cold (Холодний) .

- Для виводу на екран симулятора модулів I/O (Вводу/виводу) клацніть по Demoio - Driver, що видно внизу монітора;
- розмістіть симулятор таким чином, щоб проект на робочому листку не був прихований;
- тричі клацніть по зеленому віртуальному світлодіоду з адресою IX0.0 і спостерігайте реакцію схеми в робочому листку проекту.

Motor (Двигун) почне працювати після того, як з'явиться цифра три біля Pressed, тому що поточне значення CV- виходу на блоці Motor_Count досягає установленого значення на PV- вході того ж блока.

Коли котушка Motor у мережі 001 вмикає двигун, вмикається контакт Motor у мережі 002 і таймер M_Time починає відлік часу протягом 20с. Тривалість руху двигуна продовжується доки час, що фіксує Actual_Time (Фактичний час) біля ET-виходу таймера, не співпаде з уставкою біля PT-входу (T#20s). По завершенню роботи таймера котушка –(R)– у мережі 002 вимикає двигун (Motor), що у мережі 001.

3.2.6. **Оперативне редагування**

Оперативне редагування можливе без зупинки програми, яка реалізується на ПЛК або симуляторі. Цей режим викликається, клацаючи в панелі інструментів по іконі Patch POU (Вставка POU) .

Коли використовується режим Patch POU (Вставка POU), зміни, що зроблені в коді програми автоматично завантажуються до ПЛК або симулятора. Протягом всього часу роботи в режимі Patch POU (Вставка POU) виконання коду програми не припиняється.

Як приклад роботи в режимі Patch POU (Вставка POU), вставимо у програму змінну Emergency_Stop(Непередбачена зупинка). Активізація Emergency_Stop (Непередбачена зупинка) негайно зупинить двигун.

- Клацніть по іконі Debug on/off (Налагодження В/В), щоб перевести робочий листок коду програми в режим редагування. Проте, ресурс так само, як і реальний контролер, працює;

- установіть курсор на робочому листку коду програми нижче шини живлення OO2 і клацніть лівою клавішею;

- в панелі інструментів клацніть по іконі Contact Network (Мережа контактів), і на робочому листку з'явиться нова LD- мережа 003;

- двічі клацніть по контакту C005, відкриється діалог Contact/Coil Properties (Властивості Контакт/Котушка). У рядку Name (Ім'я) замініть типове ім'я C005 на Emergency_Stop (Непередбачена зупинка);

- клацніть OK, автоматично відкриється сторінка діалогу Common(Загальний);

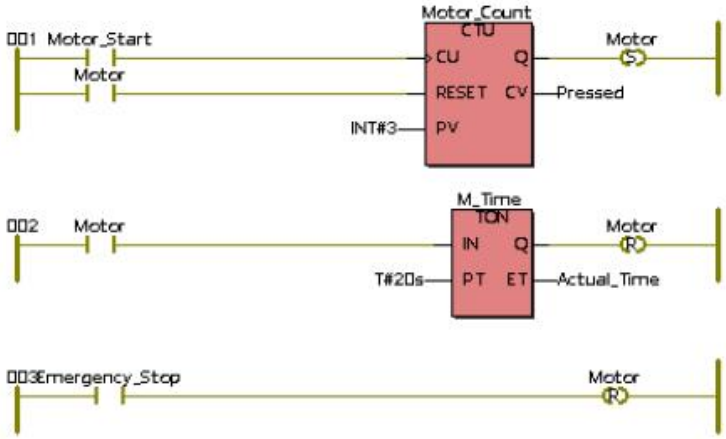
- з вікна списку Usage (Вжити) виберіть VAR_EXTERNAL, декларуючи таким чином, Emergency_Stop (Непередбачена зупинка), як глобальну змінну.

Для непередбаченої зупинки двигуна використаємо другий канал симулятора I/O:

- у рядку I/O Address (Адреса B/B) введіть % IXO.1, клацніть OK, у мережі 003 з'явиться змінна Emergency_Stop замість C005;

- двічі клацніть по контакту C006, з'явиться діалог Contact/Coil Properties (Властивості Контакт/Котушка);

- з вікна списку Type (Тип) виберіть –(R)–, а з вікна списку Variable (Змінна) – Motor, клацніть OK, і на робочому листку з'явиться третя відкоригована LD- мережа.



Тепер, коли код програми змінений, використаємо режим Patch POU (Вставка POU), щоб прокомпілювати змінні і завантажити їх до симулятора без його зупинки.

- В панелі інструментів клацніть по іконі Patch POU (Вставка POU), змінений код проекту прокомпілюється і завантажиться у симулятор I/O.

Після того, як режим Patch POU (Вставка POU) успішно завершиться, робочий листок буде автоматично переведений до оперативного режиму.

- Клацніть по Demoio-Driver, відкриється симулятор I/O;

- тричі подвійним клацанням по нульовому віртуальному світлодіоду нульового модуля I/O змініть стан біту;

- використовуючи новий контакт Emergency_Stop (Непередбачена зупинка), негайно зупиніть двигун Motor, клацаючи по світлодіоду 1-го нульового модуля вводу.

3.2.7. **Створення певної функції користувача**

Система програмування дозволяє користувачеві створювати власну функцію. У цьому проекті створимо функцію користувача, яка має перелічити кількість активізацій двигуна. Зробимо це, використовуючи текстову ST-мову. Перед створенням функції виберіть в меню Extras (Доповнення) і увійдіть в Options → Graphical Editor (Опції → Графічний редактор);

- активізуйте Function with EN/ENO і клацніть ОК; тут EN/ENO – означає додатковий логічний вхід EN і додатковий логічний вихід ENO для функцій стандарту EIC61131 в мовах програмування LD і FBD;
- в панелі інструментів клацніть по іконі Debug on/off (Налагодження В/В) для переходу в режим редагування;
- в панелі інструментів клацніть по іконі Add Function (Додати функцію) для створення функції користувача, з'явиться вікно Insert (Вставка);
- в полі Name (Ім'я) введіть Cycle_Count (Рахувати цикли), як ім'я функції, крапками позначте Function (Функція) у групі Type (Вид) і ST у групі Language (Мова). У рядку Data Type of Return Value (Тип даних , значення, що повертається) конкретизуйте, який тип даних застосовується до виходу функцій. Змінна, що зв'язана з виходом функції, має відповідати типу даних, що повертаються. У нашому прикладі використовуємо INT зі списку, що надається;
- клацніть ОК, функція додається до дерева проекту. Зірка наприкінці імені створеної функції вказує, що новий POU ще не скомпільований;
- клацніть по Cycle_Count (Рахувати цикли), відкриється робочий листок Cycle_Count (Рахувати цикли) для редагування ST-коду;
- у відкритому листку надрукуйте наступний код:

1. Cycle_Count := Count+1;

Цей рядок коду генеруватиме значення, яке має безперервно збільшуватися після кожного запуску двигуна;

- розмістіть курсор на змінній Count (Рахувати) і в панелі інструментів клацніть по іконі Variables(Змінні), з'явиться діалогове вікно Variables(Змінні);
- активізуйте Common (Загальний) у вікні Variables (Змінні) і декларуйте локальну змінну:
- у рядку Usage (Вжити) виберіть VAR_INPUT;
- у рядку Data Tape (Тип даних) виберіть INT;
- клацніть ОК, підтверджуючи діалог. Декларована змінна автоматично вставиться у робочий листок змінних нового ST-POU;
- закрийте ST- робочий листок і збережіть змінні.

Після закриття нового ST-робочого листка користувача функція залишається доступною Майстру редагування (Editor Wizard) і може бути встановленою в інші робочі листки проекту.

Щоб викликати створену функцію користувача Cycle_Count (Рахувати цикли) із головного POU програми:

- у дереві проекту подвійним клацанням по Main (Головний) відкрийте робочий листок;
- клацніть нижче LD- мережі 003, щоб позначити місце вставки нової мережі;
- в панелі інструментів клацніть по іконі Contact Network (Мережа контактів) , з'явиться мережа 004;
- позначте курсором лінію сполучення між C007 та C008 і клацніть лівою клавішею миші;
- в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування) , щоб активізувати Майстра редагування;
- у вікні Group (Група) виберіть My_First_Project, який містить функцію користувача проекту Cycle_Count (Рахувати цикли);
- двічі клацніть по Cycle_Count (Рахувати цикли) і функція з'явиться на позначеному місці мережі 004;
- в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування), вікно Group (Група) закриється;

З'єднаємо нову змінну з входом Count (Підрахунок), щоб бачити її значення:

- двічі клацніть по блакитній точці на вході Count (Підрахунок) функції Cycle_Count (Рахувати цикли), з'явиться діалог Variable Properties (Властивості змінної);
- у полі Name (Ім'я) декларуйте локальну змінну Motor_Cycle;
- клацніть ОК, підтверджуючи діалог, автоматично активізується вікно Common (Загальний);
- у діалозі Common (Загальний) у рядку списку Name (Ім'я) виберіть VAR, а у вікні списку Data Type виберіть INT;
- клацніть ОК, закриється вікно Variable properties (Властивості змінної) і змінна Motor_Cycle потрапляє у середину тіла коду проекту, а її декларація – у список локальних змінних робочого листка.

З'єднаємо також змінну Motor_Cycle з виходом функції Cycle_Count (Рахувати цикли):

- двічі клацніть по зеленій точці на виході функції Cycle_Count (Рахувати цикли), з'явиться діалог Variable Properties (Властивості змінної);
- зі списку змінної виберіть курсором Motor_Cycle і клацніть лівою клавішею миші;
- клацніть ОК, діалог Variable Properties закриється і змінна Motor_Cycle увійде у відповідний список.

Змінимо ім'я контакту 007 у мережі 004 на Motor (Двигун):

- двічі клацніть по контакту C007 у робочому листку проекту, відкриється діалог Contact/Coil Properties (Властивості Контакт/Котушка);
- зі списку змінних виберіть курсором Motor (Двигун) і клацніть лівою клавішею миші;
- клацніть ОК, і вибране ім'я замінить C007;

- в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування), з'явиться таблиця Group (Група);

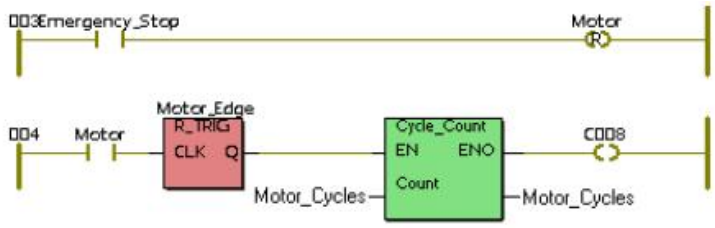
- з наведеного списку виберіть курсором Function Blocks (Функціональні блоки) і клацніть лівою клавішею, з'явиться перелік блоків;

- подвійно клацніть по функціональному блоку детектора імпульсів R_TRIG, з'явиться діалог Variable Properties (Властивості змінної);

- у полі Name (Ім'я) введіть Motor_Edge (Двигун_Фронт), клацніть ОК;

- закрийте діалог Variable Properties (Властивості змінної), клацніть ОК і функціональний блок R_TRIG потрапить у LD- мережу С004, а його декларація – у код програми;

- клацніть по іконі Edit Wizard (Майстер редагування), таблиця Group (Група) зникне з екрана;



- двічі клацніть по котушці С008, з'явиться діалогове вікно Contact/Coil Properties (Властивості Контакт/Котушка);

- у рядку Name (Ім'я) введіть Motor_Counted (Мотор_Рахівник) і клацніть ОК, активізується діалогове вікно Common (Загальний);

- у вікні списку Usage (Вжити) виберіть VAR, а у вікні списку Data Type (Тип даних) виберіть BOOL і клацніть ОК, діалогове вікно зникне;

- в панелі інструментів клацніть по іконі Make (Створювати) і проект скомпілюється.

Якщо помилки є, відкоригуйте проект, якщо їх немає, то створення зразкового проекту завершене і його треба завантажити, використовуючи відомий шлях Project Control Dialog → Download → Download;

- клацніть Cold (Холодний) у діалозі Resource (Ресурс) для холодного пуску симулятора ПЛК.

Перевіримо поведінку програми:

- в панелі інструментів клацніть по іконі Debug on/off (Налагодження В/В), щоб перейти в оперативний режим (всі змінні позначаються різними кольорами);

- клацніть по Demoio_Driver, що внизу екрана і симулятор I/O - відкриється;

- тричі подвійно клацніть по нульовому світлодіоду нульового модуля вводу In.

Програма почне виконуватися, а біля виходу Actual_Time (Фактичний час) таймера M_Time будуть змінюватися секунди затримки. При цьому з

кожним новим запуском програми значення змінної Motor_Cycles у функції Cycles_Count (Рахувати_Цикли) збільшується на одиницю, рис.3.3.

З поточного робочого листка можна перейти на функцію користувача, якщо позначити Cycles_Count (Рахувати_Цикли):

- двічі клацніть по функції Cycles_Count (Рахувати_Цикли) і в LD-робочому листку з'явиться діалог;
- підтвердіть його, натиснувши Yes(Так), робочий листок тіла коду функції Cycles_Count (Рахувати – Цикли) відкриється і в ньому буде показано поточне значення лічильника;
- закрийте робочий листок тіла коду, щоб повернутися до LD- мережі.

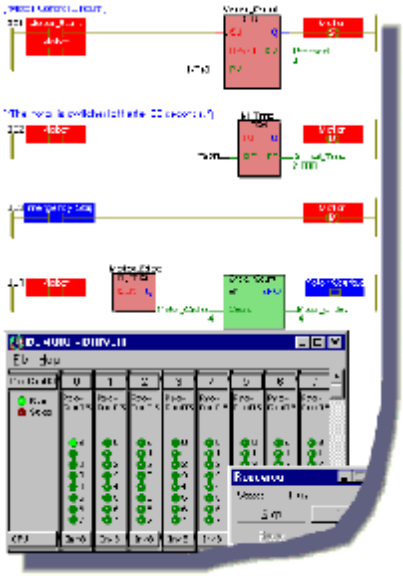


Рис. 3.3. Фрагмент робочого листка проекту в режимі налагодження.

3.2.8. **Додаткові можливості оперативного редагування**

3.2.8.1.**Вікно довідкове і спостереження змінних**

Вікно спостереження змінних – це могутній інструмент, що дозволяє легко вставити різні змінні у список і спостерігати динаміку їх поведінки. Як тільки змінна додана у вікно спостереження, не потрібно відкривати відповідний робочий листок для контролю за нею. Таким чином, можна зосередитися лише на змінних, які потрібні для аналізу роботи програми.

- Клацніть по іконі Debug on/off (Налагодження В/В), щоб перевести робочий листок в оперативний режим;
- в панелі інструментів клацніть по іконі Watch Window (Вікно спостереження), унизу екрана з'явиться вікно спостереження;
- установіть курсор на змінну Motor_Start і клацніть правою клавішею миші;
- у контекстному меню, що з'явилось, виберіть Add to Watch Window (Додати до вікна спостереження) і клацніть лівою клавішею миші, змінна Motor_Start буде внесена до списку;

- продублюйте такі самі дії зі змінними Pressed (Натиснути) і Actual_Time (Фактичний час). У вікні спостереження з'явиться список з трьох змінних;

- натисніть Demoio-Driver для виклику симулятора I/O;

- тричі подвійно клацніть по нульовому світлодіоду нульового модуля I/O, щоб запустити програму, і спостерігайте стан змінних одночасно на робочому листку і у вікні спостереження.

Окрім вікна спостереження змінних при оперативному редагуванні дуже зручним є перехресне довідкове вікно. Перехресний довідковий список містить всі змінні, функціональні блоки, переходи, мітки та зв'язки, які використовувалися у проекті. Цей інструмент особливо корисний при налагодженні і визначенні помилок.

- В панелі інструментів клацніть по іконі Cross Reference (Перехресна довідка), унизу екрана відкриється перехресне довідкове вікно, якщо вже не відкрито;

- установіть курсор на полі довідкового вікна і клацніть правою клавішею миші, відкриється контекстне меню;

- виберіть пункт меню Build Cross Reference (Створення перехресної довідки) і клацніть лівою клавішею миші, відкриється довідковий список;

- двічі клацніть по будь-якій змінній, що у вікні перехресної довідки. Вікно відкриє робочий листок, в якому ця змінна підсвічується. Підсвічується також ця змінна у вікні перехресної довідки;

- закрийте вікно перехресної довідки і вікно повідомлення, клацаючи послідовно в панелі інструментів по іконах Cross Reference (Перехресна довідка) і Message (Повідомлення) .

3.2.8.2. **Контрольні точки**

Контрольні точки можуть бути встановлені в робочі листки, використовуючи праву частину діалогу Debug: Resource (Налагодження: Ресурс). Коли контрольна точка установлена, виконання програми зупиниться у місці її установки і поновлюється, коли користувач змушує це поновлення. Система програмування забезпечує виконання програми поки не буде досягнута наступна контрольна точка (Єдиний крок) або поки таж сама контрольна точка не буде досягнута знову (Єдиний цикл).

Якщо контрольна точка досягається, стан ПЛК змінюється на HALT (Зупинник). З'являється вікно з кнопками GO (Пуск), Step (Крок) і Trace (Відслідковування).

Пуск. Натискання кнопки GO (Пуск) відновлює виконання програми до наступної контрольної точки.

Крок. Натискання кнопки Step (Крок) примушує програму виконати наступний крок.

Відслідковування. Якщо функція або функціональний блок, що визначив користувач, досягнуті у процесі виконання програми, тіло коду функції або функціонального блока відкривається для покрокового налагодження.

Треба бути дуже обережними при використанні контрольних точок під час роботи ПЛК, оскільки контрольна точка фактично зупиняє виконання програми.

Щоб спостерігати дію контрольної точки, робочий листок повинен знаходитися в оперативному режимі, тобто ікона Debug on/off (Налагодження В/В) має бути активізована:

- клацніть по іконі Project Control Dialog (Діалогове управління проектом), щоб відкрити діалог Resource (Ресурс) управління ресурсом;
- в робочому листку проекту двічі клацніть по Motor_Start , з'явиться діалог Debug: Resource (Налагодження: Ресурс);
- виберіть Set (Вставити) для встановлення точки на вибраній змінній. При цьому в робочому листку змінна Motor_Start підсвічується помаранчевим кольором;
- у діалозі Resource (Ресурс) натисніть кнопку GO (Пуск) для поновлення виконання програми до наступної контрольної точки.

Оскільки установлена тільки одна точка, програма знову зупиняється на Motor_Start, завершуючи перший цикл.

- Клацніть Step (Крок) декілька разів і помітьте, що помаранчеве підсвічування кожного разу переміщується на наступну інструкцію, вказуючи точку, де виконання програми зупиниться. При цьому змінна Motor_Star має червоний колір, показуючи, де була встановлена контрольна точка;
- двічі клацніть по Motor_Start, з'явиться діалог Debug: Resource (Налагодження: Ресурс);
- натисніть Reset (Скидання), червона відмітка на Motor_Start зникне, залишиться тільки помаранчева, вказуючи на нове місце контрольної точки;
- для продовження програми у діалозі Resource (Ресурс) клацніть GO (Пуск);
- для переходу в режим редагування проекту клацніть по іконі Debug on/off (Налагодження В/В) ;
- для зупинення симулятора клацніть Stop (Стоп) у діалозі Resource (Ресурс);
- клацніть Close (Закрити) у діалозі Resource (Ресурс), щоб закрити діалогове вікно керування.

3.2.8.3. **Зміна часу циклу задачі**

Система програмування дозволяє змінювати цикл задачі, тобто інтервал часу, в якому задача виконується. Зменшення часу циклу задачі збільшує швидкість процесу її виконання. У даному прикладі змінимо час циклу задачі від 100ms до 90ms.

Найменший цикл задачі залежить від типу ПЛК, що використовується:

- в панелі інструментів клацніть по іконі Debug on/off (Налагодження В/В) для переходу в режим редагування, якщо це потрібно. Щоб змінити конфігурацію задачі у піддереві Physical Hardware (Фізична апаратура) дерева

проекту, активізуйте Task: Cyclic (Задача: Цикл) і клацніть правою клавішею миші, з'явиться контекстне меню;

- з контекстного меню виберіть Setting (Установки), відкриється діалог Task setting for IPC_32 (Задача установки для IPC_32);
- у рядку Interval (Інтервал) введіть 90 замість 100, клацніть ОК для підтвердження діалогу;
- в панелі інструментів клацніть по іконі Make (Створювати) і проект скомпілюється;
- для завантаження проекту, в панелі інструментів клацніть по іконі Project Control Dialog (Діалог управління проектом), і послідовно натисніть Download (Завантаження) → Download (Завантаження) у діалогах, що з'являються на екрані;
- підтвердіть завантаження, натискаючи Yes (Так).

3.2.8.4. **Зміна конфігурації симулятора**

Configuration I/O (Конфігурація В/В) у проектному дереві використовується для редагування I/O у робочому листку конфігурації. Пояснення з використання Configuration I/O (Конфігурація В/В) зробимо на прикладі збільшення кількості вхідних модулів в існуючому симуляторі з 8 до 10:

- у піддереві Physical Hardware (Фізична апаратура) двічі клацніть по Configuration I/O (Конфігурація В/В). З'явиться діалог Configuration I/O (Конфігурація В/В);
- у діалозі Configuration I/O (Конфігурація) натисніть кнопку Properties (Властивості);
- у полі Length (Довжина) введіть 10 і на клавіатурі ЕОМ натисніть клавішу <<Тав>>. У рядку End address (Кінцева адреса) з'явиться адреса %IB9;
- клацніть ОК для підтвердження нових властивостей і клацніть ОК у діалозі Configuration I/O (Конфігурація В/В), щоб повернутися до програмування;
- в панелі інструментів клацніть по іконі Make (Створювати), щоб скомпілювати проект.
- завантажте проект, як було наведено раніше;
- клацніть по Demoio-Driver внизу екрана і на симуляторі I/O, що з'явився, замість 8 модулей буде 10.

3.2.9. **Друкування проектної документації**

Після створення проекту корисно надрукувати документацію. Система програмування проходить декілька етапів у створенні документації: попереднього перегляду поточної сторінки, визначення установки принтера і друку повного проекту або одного робочого листка. Система програмування за бажанням користувача дозволяє друкувати різні сторінки проекту. При друкуванні створеного проекту автоматично використовується типова сторінка фірми виробника системи програмування.

Вибір принтера:

- виберіть пункт меню File (Файл) і клацніть лівою клавішею миші;
- увійдіть в підменю Print (Друк), відкриється стандартний Windows (Вікна);
- введіть назву принтера і клацніть ОК;
- з меню File (Файл) виберіть Print Project (Друк проекту) і клацніть лівою клавішею миші;
- у діалоговому вікні Print Project (Друк проекту), що з'явилося, зніміть позначки частин проекту, які не бажано друкувати;
- увімкніть принтер і клацніть Print (Друк).

Попередній перегляд і друк єдиного робочого листка

Попередній перегляд друку дозволяє побачити, який вигляд матиме робочий листок після друкування і якщо потрібно скоригувати його.

Щоб викликати попередній перегляд:

- переконайтесь, що робочий листок, який треба переглянути дійсно у вікні;
- виберіть в меню шлях File → Print Preview(Файл → Перегляд друку) і зробіть попередній огляд друку активного робочого листка;
- щоб надрукувати один робочий листок, що показаний, в головному меню натисніть кнопку Print (Друк) .

Контрольні запитання

1. З яких етапів складається створення нового проекту?
2. Як розробляється код програми?
3. Як здійснюється компіляція і завантаження проекту в симулятор або ПЛК?
4. Що таке режим налагодження і оперативного редагування?
5. Як створюється власна функція користувача?
6. Як створюється перехресне довідкове вікно і вікно спостереження змінних?
7. Як здійснюється налагодження проекту за контрольними точками?
8. Як змінити час циклу задачі?
9. Для чого змінюється конфігурація симулятора?
10. Як здійснюється друкування проектної документації?

3.3. Методика програмування FBD-мовою

Створимо FBD-мовою програму керування роботою двигуна згідно з алгоритмом наведеним у 3.2.

Для цього відкрийте діалог New Project (Новий проект), використовуючи пункт меню File (Файл). Двічі клацніть по Project Wizard (Майстер проекту) і пройдіть увесь шлях створення проекту аналогічно попередньому варіанту, тобто призначте ім'я проекту та його POU, виберіть FBD-мову програмування, бажані імена і типи конфігурації, ресурсу і задачі.

Коли новий проект буде створено, він з'явиться у вікні проектного дерева. Щоб розробити проектний код, необхідно послідовно вибрати з бібліотеки системи програмування MULTIPROG відповідні стандартні компоненти, з'єднати їх між собою та оголосити властивості змінних. Згідно з алгоритмом керування роботою двигуна для створення програми потрібні функціональні блоки лічильника CTU - для підрахунку кількості натисків на кнопку старту двигуна, таймера TON – для визначення часу роботи двигуна, детектора імпульсів R_TRIG – для керування роботою лічильника кількості активізацій двигуна, домінантного перемикача RS – для зупинки двигуна, а також логічний оператор OR і функція користувача Cycle_Count для підрахунку кількості активізацій двигуна.

Установимо на робочому полі проекту функціональний блок лічильника для підрахунку кількості натисків на кнопку старту двигуна:

- Виберіть на робочому полі місце установки першого функціонального блока і клацніть лівою клавішею миші;.
- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть функціональний блок CTU;
- подвійно клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я CTU-1 на Motor-Count;
- клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишиться функціональний блок Motor_Count.

Оголосимо змінні блока Motor_Count:

- Лівою клавішею миші активізуйте на функціональному блоці CU-вхід для установлення на ньому змінної запуску двигуна.
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Motor_Start, натисніть кнопку Apply (Застосувати);
- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;
- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вхідного каналу модуля вводу %IX0.0;
- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;
- натисніть ОК, діалогове вікно закриється, а біля CU-входу лічильника з'явиться змінна Motor_Start;
- лівою клавішею миші активізуйте на функціональному блоці RESET-вхід для оголошення змінної, яка має скидати лічильник;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Motor, натисніть кнопку Apply (Застосувати);
- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL.
- натисніть ОК, діалогове вікно закриється, а біля RESET-входу лічильника з'явиться змінна Motor.

- лівою клавішею миші активізуйте на функціональному блоці PV-вхід для визначення параметрів лічильника;

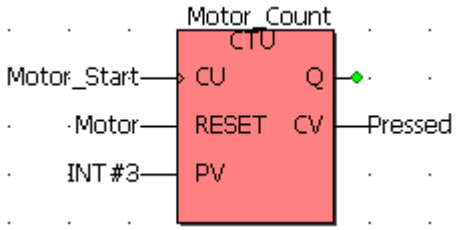
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте INT#3, натисніть послідовно кнопки Apply (Застосувати) та ОК і біля PV -входу лічильника з'являться його параметри;

- лівою клавішею миші активізуйте на функціональному блоці CV-вихід для оголошення змінної, яка запам'ятовує поточне значення лічильника;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Pressed, натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - INT;

- натисніть ОК, діалогове вікно закриється, а біля CV-виходу лічильника з'явиться змінна Pressed:



З'єднаємо функціональний блок лічильника Motor_Count з функціональним блоком домінантного перемикача RS і оголосимо його вихідну змінну:

- Лівою клавішею миші активізуйте Q-вихід на функціональному блоці Motor_Count;

- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть функціональний блок перемикача RS, який має домінанту вимикача;

- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) у рядку Name з'явиться ім'я екземпляра блока RS_1;

- клацніть ОК, діалогове вікно закриється, а на робочому листку проекту з'явиться екземпляр функціонального блока RS_1 сполучений з функціональним блоком Motor_Count;

- лівою клавішею миші активізуйте Q-вихід на функціональному блоці RS_1 для оголошення вихідної змінної, яка увімкне у роботу двигун;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, серед існуючих змінних виберіть Motor і натисніть ліву клавішу миші, у рядку Name з'явиться назва змінної Motor;

- натисніть кнопку Apply (Застосувати) і відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вихідного каналу модуля виводу %QX0.0;

- натисніть ОК, діалогове вікно закриється, а біля виходу функціонального блока RS_1 з'явиться змінна Motor.

Оголосимо змінні, які мають зупиняти роботу двигуна:

Оскільки двигун має зупинятися після закінчення установленого часу роботи, а також примусово у будь-який момент, використаємо з бібліотеки MULTIPROG логічний оператор OR.

- Виберіть на робочому полі проекту місце розташування логічного оператора OR і клацніть лівою клавішею миші;

- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть логічний оператор OR;

- двічі клацніть по ньому і на робочому полі проекту з'явиться оператор OR;

- активізуйте перший вхід логічного оператора;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Emergency_Stop(Непередбачена зупинка), натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вхідного каналу модуля вводу %IX0.1;

- натисніть ОК, діалогове вікно закриється, а біля першого входу логічного оператора OR з'явиться змінна Emergency_Stop;

- активізуйте другий вхід логічного оператора;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Mot_Time, натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- натисніть ОК, діалогове вікно закриється, а біля другого входу логічного оператора OR з'явиться змінна Mot_Time.

Визначимо час роботи двигуна:

- Виберіть на робочому полі проекту місце установки таймера TON і клацніть лівою клавішею миші;

- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть функціональний блок TON;

- подвійно клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я TON-1 на M_Time;

- натисніть кнопку Apply (Застосувати) і клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишиться функціональний блок M_Time.

Оголосимо змінні таймера M_Time:

- Лівою клавішею миші активізуйте на функціональному блоці IN-вхід для встановлення на ньому змінної запуску таймера;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, виберіть змінну Motor і клацніть лівою клавішею миші;

- натисніть послідовно кнопки Apply (Застосувати) та ОК і біля IN-входу з'явиться змінна Motor;

- активізуйте PT-вхід таймера для визначення часу його роботи;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте T#20s;

- натисніть кнопку Apply (Застосувати) і закрийте діалогове вікно, на PT-вході таймера з'явиться константа часу;

- активізуйте ET-вихід таймера для визначення змінної, що ідентифікує поточний час;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Actual_Time ;

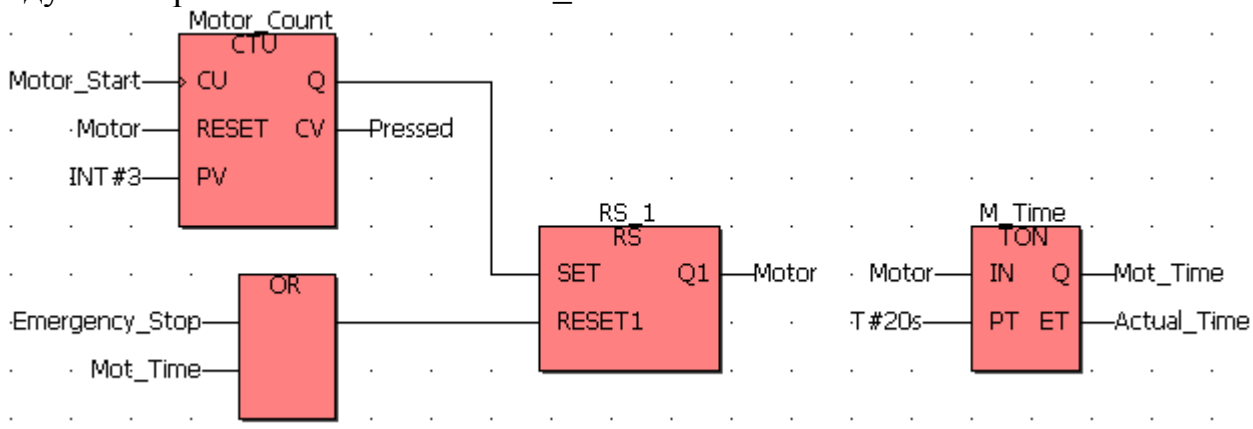
- Натисніть кнопку Apply (Застосувати) і відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - TIME;

- натисніть ОК, діалогове вікно закриється, а біля ET- виходу таймера з'явиться змінна Actual_Time;

- активізуйте основний Q-вихід таймера ;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, виберіть змінну Mot_Time і клацніть лівою клавішею миші;

- натисніть послідовно кнопки Apply (Застосувати) та ОК і біля Q-виходу таймера з'явиться змінна Mot_Time.



Створимо функцію користувача, яка має перелічувати кількість активізації двигуна:

- Аналогічно попередній LD - програмі створіть функцію користувача Cycle_Count, яка має перелічувати кількість вмикань двигуна.

Щоб це відбувалося, на її EN-вході має виникати імпульс при кожному вмиканні двигуна, який формується функціональним блоком R_TRIG. Для створення цього блока:

- лівою клавішею миші позначте перед функцією користувача місце його установки;
- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть функціональний блок R_TRIG;
- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної), що з'явилося, змініть типове ім'я функціонального блока R_TRIG_1 на Motor_Edge;
- натисніть кнопку Apply (Застосувати) і клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишиться функціональний блок Motor_Edge.

Оголосимо змінну функціонального блока Motor_Edge і з'єднаємо його вихід з функцією користувача:

- Лівою клавішею миші активізуйте CLK-вхід на функціональному блоці Motor_Edge для установлення на ньому змінної керування лічильником вмикань двигуна;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, виберіть вже оголошену змінну Motor і клацніть лівою клавішею миші;
- натисніть послідовно кнопки Apply (Застосувати) та ОК і біля CLK-входу з'явиться змінна Motor;
- в панелі інструментів активізуйте ікону Connect objects і з'єднайте Q-вихід функціонального блока Motor_Edge з EN-входом функції користувача Cycle_Count.

Оголосимо змінну, яка відображає значення лічильника Cycle_Count:

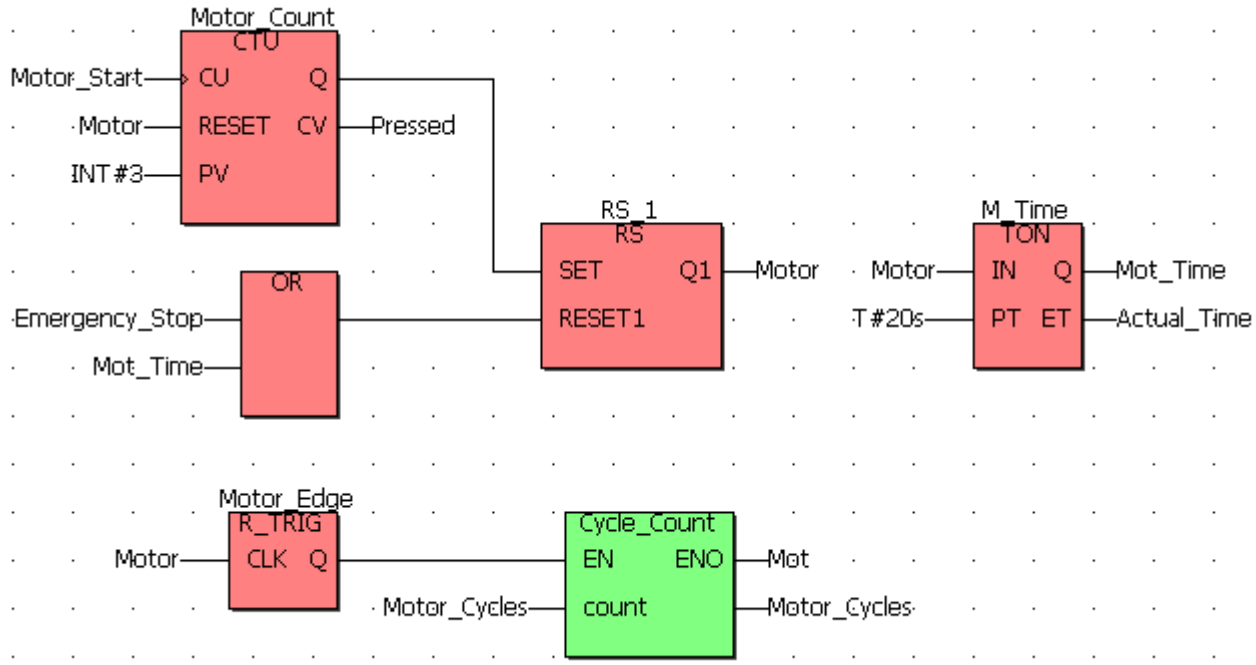
- Активізуйте Count-вхід функції користувача Cycle_Count;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я змінної Motor_Cycles;
- натисніть кнопку Apply (Застосувати), відкрийте сторінку Common, і у рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - INT;
- натисніть ОК, діалогове вікно закриється, а біля Count-входу функції користувача Cycle_Count з'явиться ім'я змінної Motor_Cycles;

Так саме оголосить змінну Motor_Cycles для другого виходу функції користувача Cycle_Count.

- Активізуйте логічний ENO-вихід функції користувача Cycle_Count;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я змінної Mot;

- натисніть кнопку Apply (Застосувати), відкрийте сторінку Common, у рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- натисніть ОК, діалогове вікно закриється, а біля ENO-виходу функції користувача Cycle_Count з'явиться ім'я змінної Mot;



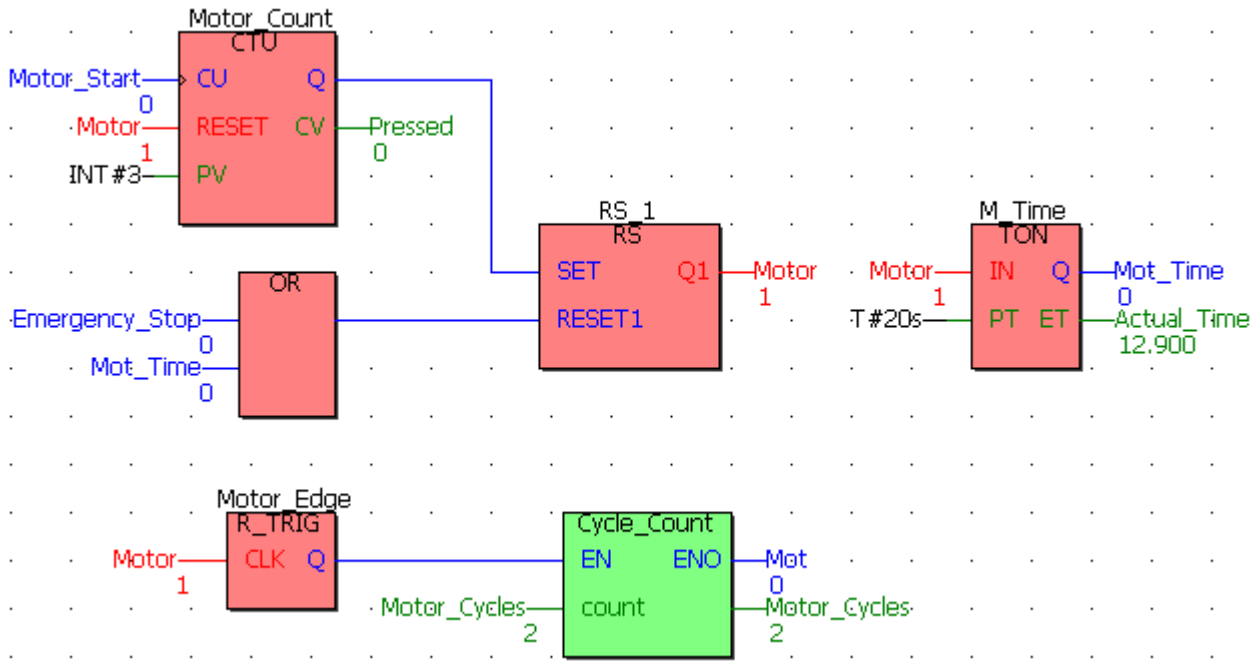
Після цього в панелі інструментів натисніть ікону Make (Створювати), для компіляції проекту.

Якщо помилки є, відкоригуйте проект, якщо немає - завантажте його, використовуючи відомий шлях: Project Control Dialog → Download→Download.

Перевіримо роботу програми:

- В панелі інструментів клацніть по іконі Project Control Dialog (Управління діалогом проекту), з'явиться діалогове вікно Resource (Ресурс);
- натисніть кнопку Cold (Холодний) у діалоговому вікні Resource (Ресурс) для холодного запуску симулятора ПЛК;
- в панелі інструментів клацніть по іконі Debug on/off (Налагодження В/В), щоб перейти в оперативний режим роботи. При цьому усі змінні позначаться різними кольорами;
- клацніть лівою клавішею миші по Demoio_Driver унизу екрана, для відкриття симулятора I/O;
- тричі подвійно клацніть лівою клавішею миші по нульовому світлодіоду нульового модуля вводу In.

Програма почне виконуватися, а біля змінної Actual_Time таймера M_Time будуть змінюватися секунди затримки. При цьому з кожним новим запуском програми значення змінної Motor_Cycles у функції Cycle_Count збільшується на одиницю.



Контрольні запитання

1. Як створюється FBD-проект?
2. Як здійснюється програмування FBD-мовою?
3. Як оголошуються змінні у функціональному блоці?
4. Як перевірити роботу програми?
5. Як створити функцію користувача?

3.4. Методика програмування IL-мовою

Створимо IL-мовою програму керування роботою двигуна згідно з алгоритмом наведеним у 3.2.

Для цього, відкрийте діалог New Project (Новий проект), використовуючи пункт меню File (файл). Двічі клацніть по Project Wizard (Майстер проекту) і пройдіть увесь шлях створення проекту аналогічно варіанту програмування LD-мовою, тобто призначте ім’я проекту та його POU, виберіть у даному випадку IL-мову програмування, бажані імена і типи конфігурації, ресурсу і задачі.

Коли новий проект з’явиться у вікні проектного дерева, можна починати програмування.

Створюючи проектний код, відповідно до алгоритму керування роботою двигуна, будемо з бібліотеки системи програмування MULTIPROG послідовно вибирати функціональні блоки лічильника CTU - для підрахунку кількості натисків на кнопку старту двигуна , таймера TON – для визначення часу роботи двигуна і ще раз лічильника CTU – для підрахунку його активізацій.

Запрограмуємо IL-мовою функціональний блок лічильника для підрахунку кількості натисків на кнопку старту двигуна:

- На робочому полі проектного коду позначте курсором місце початку програмування;
- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group(Група), що з'явилася на екрані, виберіть функціональний блок CTU.
- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я CTU-1 на Motor-Count;
- клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишиться послідовність IL-інструкцій з коментарями, яка програмує функціональний блок Motor_Count.

```
LD   (* BOOL *)
ST   Motor_Count.CU
LD   (* BOOL *)
ST   Motor_Count.RESET
LD   (* INT *)
ST   Motor_Count.PV
CAL Motor_Count
LD   Motor_Count.Q
ST   (* BOOL *)
LD   Motor_Count.CV
ST   (* INT *)
```

Оголосимо змінні блока Motor_Count:

У наведеній послідовності IL-інструкцій місця змінних займають коментарі – підказки їх типів.

- Установіть курсор перед коментарем першого рядка і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);
- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Motor_Start і натисніть кнопку Apply (Застосувати);
- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;
- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вхідного каналу модуля вводу %IX0.0;
- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;
- натисніть ОК, діалогове вікно закриється, а у першому рядку між інструкцією LD і коментарем з'явиться змінна Motor_Start, яка наступною інструкцією ST запам'ятовується у CU - вході лічильника Motor_Count.
- установіть курсор перед коментарем третього рядка, щоб оголосити змінну, яка призначена для RESET(Скид)-входу лічильника Motor_Count;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Motor, натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вихідного каналу модуля виводу %QX0.0;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а у третьому рядку між інструкцією LD і коментарем з'явиться змінна Motor, яка наступною інструкцією ST запам'ятовується у RESET- вході лічильника Motor_Count;

- установіть курсор перед коментарем п'ятого рядка для визначення параметрів PV - входу;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте константу INT#3, натисніть послідовно кнопки Apply (Застосувати) та ОК і у п'ятому рядку IL-інструкцій з'являться параметри лічильника;

- установіть курсор перед коментарем дев'ятого рядка, щоб оголосити змінну, якою керує Q-виход лічильника Motor_Count;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я Out і натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а у дев'ятому рядку між інструкцією ST і коментарем з'явиться змінна Out;

- установіть курсор перед коментарем останнього рядка, щоб оголосити змінну, яка запам'ятовує імпульси лічильника Motor_Count;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Pressed, натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - INT.

- натисніть ОК, діалогове вікно закриється, а у останньому рядку з'явиться змінна Pressed.

Запрограмуємо таймер роботи двигуна:

- Позначте курсором новий рядок для продовження програми;

- активізуйте в панелі інструментів ікону Edit Wizard (Майстер проекту) і у таблиці Group(Група), що з'явилася на екрані, виберіть функціональний блок TON.

- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я TON-1 на M_Time;

- натисніть кнопку Apply (Застосувати) і клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишаться IL – інструкції, що програмують таймер M_Time.

```
LD   (* BOOL *)
ST   M_Time.IN
LD   (* TIME *)
ST   M_Time.PT
CAL M_Time
LD   M_Time.Q
ST   (* BOOL *)
LD   M_Time.ET
ST   (* TIME *)
```

Оголосимо змінні таймера M_Time:

- Установіть курсор перед коментарем першого рядка і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);
- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, виберіть ім'я Motor і натисніть послідовно кнопки Apply (Застосувати) і ОК, оскільки змінна Motor вже оголошена;
- діалогове вікно закриється, і у першому рядку між інструкцією LD і коментарем з'явиться змінна Motor, яка наступною інструкцією ST запам'ятовується в IN - вході таймера M_Time;
- установіть курсор перед коментарем третього рядка, щоб оголосити параметри таймера;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте константу часу T#20s;
- натисніть кнопку Apply (Застосувати) і закрийте діалогове вікно, у третьому рядку IL – інструкції з'явиться константа часу;
- установіть курсор перед коментарем сьомого рядка і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);
- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, зі списку змінних виберіть ім'я Motor вже оголошеної змінної і послідовно натисніть кнопку Apply (Застосувати) та ОК;
- перед коментарем у сьомому рядку з'явиться змінна Motor;
- установіть курсор перед коментарем останнього рядка, щоб оголосити змінну, яка показує поточний час роботи таймера;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Actual_Time;
- натисніть кнопку Apply (Застосувати) і відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - TIME;

- натисніть ОК, діалогове вікно закриється, а в останньому рядку залишиться змінна Actual_Time;

Запрограмуємо непередбачену зупинку двигуна:

- Позначте курсором новий рядок для продовження програми;
- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group(Група), що з'явилася на екрані, виберіть функціональний блок перемикача RS;
- подвійно клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) у рядку Name з'явиться типове ім'я екземпляра блока RS-1;
- натисніть кнопку Apply (Застосувати) і клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишаться IL – інструкції, що програмують перемикач RS-1.

```
LD   (* BOOL *)
ST   RS_1.SET
LD   (* BOOL *)
ST   RS_1.RESET1
CAL  RS_1
LD   RS_1.Q1
ST   (* BOOL *)
```

Оголосимо змінні блока RS-1:

- Установіть курсор перед коментарем першого рядка і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);
- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, виберіть ім'я Out вже оголошеної змінної і натисніть послідовно кнопки Apply (Застосувати) та ОК;
- діалогове вікно закриється, а у першому рядку між інструкцією LD і коментарем з'явиться змінна Out, яка наступною інструкцією ST запам'ятовується у SET - вході перемикача RS-1;
- установіть курсор перед коментарем третього рядка, щоб оголосити змінну, яка призначена для RESET1 (Скид)-входу перемикача RS-1;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Emergency_Stop(Непередбачена зупинка), натисніть кнопку Apply (Застосувати);
- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;
- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вхідного каналу модуля вводу %IX0.1;
- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;
- натисніть ОК, діалогове вікно закриється, а у третьому рядку між інструкцією LD і коментарем з'явиться змінна Emergency_Stop (Непередбачена

зупинка), яка наступною інструкцією ST запам'ятовується у RESET1- вході перемикача RS-1. Оскільки двигун має зупинятися не тільки під впливом вхідної змінної Emergency_Stop(Непередбачена зупинка), а ще і після 20 секунд роботи, додамо з клавіатури до змінної Emergency_Stop оператора LD альтернативну вихідну змінну Mot_Time таймера M_Time, використовуючи оператор OR.

- Установіть курсор перед коментарем останнього рядка IL – інструкцій перемикача RS-1 і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, виберіть ім'я Motor і натисніть послідовно кнопки Apply (Застосувати) і ОК, оскільки змінна Motor вже оголошена;

- діалогове вікно закриється, а у останньому рядку між інструкцією LD і коментарем з'явиться змінна Motor, якою керує Q- вихід перемикача RS-1.

Запрограмуємо лічильник кількості активізацій двигуна:

- На робочому полі проектного коду позначте курсором новий рядок програмування;

- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть функціональний блок CTU;

- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я CTU-1 на Cycle_Count;

- клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишиться послідовність IL-інструкцій, яка програмує функціональний блок Cycle_Count:

```
LD   (* BOOL *)
ST   Cycle_count.CU
LD   (* BOOL *)
ST   Cycle_count.RESET
LD   (* INT *)
ST   Cycle_count.PV
CAL  Cycle_count
LD   Cycle_count.Q
ST   (* BOOL *)
LD   Cycle_count.CV
ST   (* INT *)
```

Оголосимо змінні блока Cycle_Count:

- Установіть курсор перед коментарем першого рядка і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, виберіть ім'я Motor вже оголошеної змінної і натисніть послідовно кнопки Apply (Застосувати) та ОК;

- діалогове вікно закриється, а у першому рядку між інструкцією LD і коментарем з'явиться змінна Motor, яка наступною інструкцією ST запам'ятовується у CU - вході лічильника Cycle_Count;

- установіть курсор перед коментарем третього рядка, щоб оголосити змінну призначену для RESET-входу лічильника Cycle_Count;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Reset, натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вихідного каналу модуля виводу %IX0.5;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а в третьому рядку між інструкцією LD і коментарем з'явиться змінна Reset, яка наступною інструкцією ST запам'ятовується у RESET- вході лічильника Cycle_Count;

- установіть курсор перед коментарем п'ятого рядка для визначення параметрів PV - входу;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте константу INT#10, натисніть послідовно кнопки Apply (Застосувати) та ОК і у п'ятому рядку IL-інструкцій з'являться параметри лічильника;

- установіть курсор перед коментарем дев'ятого рядка, щоб оголосити змінну, якою керує Q-вихід лічильника Cycle_Count;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я внутрішньої змінної Mot і натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а у дев'ятому рядку між інструкцією ST і коментарем з'явиться змінна Mot;

- установіть курсор перед коментарем останнього рядка, щоб оголосити змінну, яка запам'ятовує активації двигуна Cycle_Count;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте Motor_Cycles, натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - INT;

- натисніть ОК, діалогове вікно закриється, а в останньому рядку з’явиться змінна Motor_Cycles.

В цілому IL- програма керування роботою двигуна має такий вигляд:

```
 (*Програмування лічильника кількості натисків стартової кнопки*)
LD  Motor_Start (* BOOL *)
ST  Motor_Count.CU
LD  Motor       (* BOOL *)
ST  Motor_Count.RESET
LD  int#3       (* INT *)
ST  Motor_Count.PV
CAL Motor_Count
LD  Motor_Count.Q
S   out       (* BOOL *)
LD  Motor_Count.CV
ST  Pressed     (* INT *)
 (*Програмування таймера*)
LD  Motor       (* BOOL *)
 ST M_Time.IN
LD  t#20s       (* TIME *)
ST  M_Time.PT
CAL M_Time
LD  M_Time.Q
R   Mot_Time    (* BOOL *)
LD  M_Time.ET
ST  Actual Time (* TIME *)
 (*Програмування непердбаченої зупинки двигуна*)
 LD  out(* BOOL *)
 ST  RS_1.SET
 LD  Emergency_Stop(* BOOL *)
 OR  Mot_Time
 ST  RS_1.RESET1
 CAL RS_1
 LD  RS_1.Q1
 ST  Motor(* BOOL *)
  (*Програмування лічильника циклів роботи двигуна*)
 LD  Motor       (* BOOL *)
 ST  Cycle_count.CU
 LD  Reset       (* BOOL *)
 ST  Cycle_count.RESET
 LD  int#10      (* INT *)
 ST  Cycle_count.PV
 CAL Cycle_count
 LD  Cycle_count.Q
 ST  Mot         (* BOOL *)
 LD  Cycle_count.CV
 ST  Motor_Cycles(* INT *)
```

Для компіляції проекту в панелі інструментів натисніть ікону Make (Створювати).

Якщо помилки є, відкоригуйте проект, якщо немає - завантажте його, використовуючи відомий шлях: Project Control Dialog → Download→Download.

Перевіримо роботу програми:

- Натисніть кнопку Cold (Холодний) у діалоговому вікні Resource (Ресурс) для холодного запуску симулятора PLC;
- в панелі інструментів клацніть по іконі Debug on/off (Налагодження В/В), щоб перейти в оперативний режим роботи. При цьому усі змінні позначаться різними кольорами;
- клацніть лівою клавішею миші по Demoio_Driver унизу екрана для відкриття симулятора I/O;
- тричі подвійно клацніть лівою клавішею миші по нульовому світлодіоду нульового модуля вводу In.

Програма почне виконуватися, а біля змінної Actual_Time таймера M_Time будуть змінюватися секунди затримки. При цьому з кожним новим запуском програми значення змінної Motor_Cycles у функції Cycle_Count збільшується на одиницю.

```
        (*Програмування лічильника кількості натисків
  TRUE    стартової кнопки*)
  TRUE LD  Motor_Start (* BOOL *)
  TRUE ST  Motor_Count.CU
  TRUE LD  Motor       (* BOOL *)
       ST  Motor_Count.RESET
     3 LD  int#3       (* INT *)
       ST  Motor_Count.PV
 FALSE CAL Motor_Count
  TRUE LD  Motor_Count.Q
     0 S   out       (* BOOL *)
     0 LD  Motor_Count.CV
       ST  Pressed     (* INT *)
  TRUE  (*Програмування таймера*)
  TRUE LD  Motor       (* BOOL *)
        ST M_Time.IN
20.000 LD  t#20s       (* TIME *)
       ST  M_Time.PT
 FALSE CAL M_Time
 FALSE LD  M_Time.Q
 4.200 R   Mot_Time    (* BOOL *)
 4.200 LD  M_Time.ET
       ST  Actual_Time (* TIME *)
```

```
 TRUE (*Програмування непердбаченої зупинки двигуна*)
 TRUE LD  out(* BOOL *)
FALSE ST  RS_1.SET
FALSE LD  Emergency_Stop(* BOOL *)
FALSE OR  Mot_Time
      ST  RS_1.RESET1
 TRUE CAL RS_1
 TRUE LD  RS_1.Q1
      ST  Motor(* BOOL *)
 TRUE  (*Програмування лічильника циклів роботи двигуна*)
 TRUE LD  Motor        (* BOOL *)
FALSE ST  Cycle_count.CU
FALSE LD  Reset        (* BOOL *)
      ST  Cycle_count.RESET
   10 LD  int#10       (* INT *)
      ST  Cycle_count.PV
FALSE CAL Cycle_count
FALSE LD  Cycle_count.Q
     1 ST  Mot          (* BOOL *)
     1 LD  Cycle_count.CV
       ST  Motor_Cycles(* INT *)
```

Контрольні запитання

1. Як запрограмувати IL-мовою функціональний блок?
2. Як оголосити змінні функціонального блока?
3. Як запрограмувати таймер?
4. Як запрограмувати лічильник кількості натисків стартової кнопки?
5. Як запрограмувати екстрену зупинку двигуна?

3.5. Методика програмування ST-мовою

Створимо ST-мовою програму керування роботою двигуна згідно з алгоритмом наведеним у 3.2.

Для цього відкрийте діалог New Project (Новий проект), використовуючи пункт меню File (файл). Подвійно клацніть по Project Wizard (Майстер проекту) і пройдіть увесь шлях створення проекту аналогічно варіанту програмування LD-мовою, тобто призначте ім’я проекту та його POU, виберіть у даному випадку ST-мову програмування, бажані імена і типи конфігурації, ресурсу і задачі.

Коли новий проект з’явиться у вікні проектного дерева, можна починати програмування.

Створюючи проектний код відповідно до алгоритму керування роботою двигуна, будемо з бібліотеки системи програмування MULTIPROG послідовно вибирати функціональні блоки лічильника CTU - для підрахунку кількості

натисків на кнопку старту двигуна, RS перемикача з домінантою вимкнення - для створення можливості раптової зупинки двигуна, таймера TON – для визначення часу роботи двигуна, а також запрограмуємо лічильник активізацій двигуна.

Запрограмуємо ST-мовою функціональний блок лічильника для підрахунку кількості натисків на кнопку старту двигуна:

- На робочому полі проектного коду позначте курсором місце початку програмування;
- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані виберіть функціональний блок CTU;
- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я CTU-1 на Motor-Count;
- клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишиться програма функціонального блока лічильника Motor_Count створеного ST- мовою:

```
Motor_Count(CU:=(* BOOL *),RESET:=(* BOOL *),PV:=(* INT *));
(* BOOL *):=Motor_Count.Q;
(* INT *):=Motor_Count.CV;
```

Оголосимо змінні блока Motor_Count:

У наведеній ST - програмі місця змінних займають коментарі – підказки їх типів.

- Установіть курсор перед коментарем CU- входу лічильника і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);
- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я змінної Motor_Start, за допомогою якої буде активізуватися двигун, і натисніть кнопку Apply (Застосувати);
- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;
- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вхідного каналу модуля вводу %IX0.0;
- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;
- натисніть ОК, діалогове вікно закриється, а CU- входу лічильника Motor_Count присвоїться змінна Motor_Start;
- установіть курсор перед коментарем RESET- входу лічильника, щоб оголосити його змінну;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я змінної Motor, натисніть кнопку Apply (Застосувати);
- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а RESET- входу лічильника Motor_Count присвоїться змінна Motor;

- установіть курсор перед коментарем PV- входу для визначення параметрів лічильника;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте константу INT#3, натисніть послідовно кнопки Apply (Застосувати) та ОК і PV- входу лічильника присвояться параметри, що визначають кількість натисків кнопки;

- установіть курсор перед коментарем змінної, що присвоюється Q- виходу лічильника Motor_Count;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, зі списку імен виберіть вже оголошену змінну Motor і натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вихідного каналу модуля виводу %QX0.0;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а Q- виходу лічильника Motor_Count присвоїться змінна Motor;

- установіть курсор перед коментарем змінної, яка присвоюється CV- виходу лічильника Motor_Count;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я внутрішньої змінної –Pressed, яка буде відображати кількість натисків на кнопку пуску двигуна, і клацніть по кнопці Apply (Застосувати);

- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - INT;

- натисніть ОК, діалогове вікно закриється, а CV- виходу лічильника Motor_Count присвоїться змінна Pressed.

Запрограмуємо непередбачену зупинку двигуна:

- Позначте курсором новий рядок для продовження програми;

- в панелі інструментів активізуйте іконку Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані виберіть функціональний блок RS, який має властивості перемикача із домінантою вимикача;

- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я блока на ім'я його екземпляру - RS-1;

- натисніть кнопку Apply (Застосувати) і клацніть ОК, діалогове вікно закриється, а у робочому листку проекту залишиться ST- програма функціонального блока RS:

```
RS_1(SET:=(* BOOL *),RESET1:=(* BOOL *));
(* BOOL *):=RS_1.Q1;
```

Оголосимо змінні блока RS:

У наведеній ST - програмі місця змінних займають коментарі – підказки їх типів.

- Установіть курсор перед коментарем SET- входу функціонального блока RS і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);
- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я внутрішньої змінної Out і натисніть кнопку Apply (Застосувати);
- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;
- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;
- натисніть ОК, діалогове вікно закриється, а SET- входу функціонального блока RS-1 присвоїться змінна Out;
- установіть курсор перед коментарем RESET1-входу.Оскільки двигун має зупинятися після закінчення установленого часу роботи, а також примусово у будь-який момент, для RESET1-входу оголосимо дві альтернативні змінні;
- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я першої альтернативної змінної Emergency_Stop (Непередбачений_Останов) і натисніть кнопку Apply (Застосувати);
- відкрийте сторінку діалогу Common(Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;
- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вхідного каналу модуля вводу %IX0.1;
- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;
- натисніть ОК, діалогове вікно закриється, а RESET1-входу присвоїться перша альтернативна змінна Emergency_Stop;
- надрукуйте поряд з нею логічний оператор OR і в панелі інструментів клацніть по іконі Variable (Змінна);
- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я другої альтернативної змінної Mot_Time, яка зупинить двигун за командою таймера;
- послідовно натисніть кнопки Apply (Застосувати) та ОК і друга альтернативна змінна Mot_Time присвоїться RESET1-входу;
- установіть курсор перед коментарем змінної, яка присвоюється виходу Q1 функціонального блока RS;

- в панелі інструментів клацніть по іконі Variable (Змінна), у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, зі списку імен виберіть вже оголошену змінну Motor і натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вихідного каналу модуля виводу %QX0.0;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а Q1-виходу функціонального блока RS-1 присвоїться змінна Motor.

Запрограмуємо час роботи двигуна:

- Позначте курсором новий рядок для продовження програми;

- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть функціональний блок таймера TON;

- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я TON1 на M_Time;

- натисніть кнопку Apply (Застосувати) і клацніть ОК, діалогове вікно закриється, а на робочому листку проекту залишаться ST- програма функціонального блока таймера M_Time:

```
M_Time(IN:=(* BOOL *),PT:=(* TIME *));
(* BOOL *):=M_Time.Q;
(* TIME *):=M_Time.ET;
```

- установіть курсор перед коментарем IN- входу функціонального блока M_Time і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося у рядку Name надрукуйте ім'я вже оголошеної змінної Motor і послідовно натисніть кнопки Apply (Застосувати) і ОК;

- діалогове вікно закриється, а IN- входу функціонального блока таймера M_Time присвоїться змінна Motor;

- установіть курсор перед коментарем PT- входу і в панелі інструментів клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте константу T#20s, яка встановлює час роботи двигуна і натисніть послідовно кнопки Apply (Застосувати) і ОК;

- діалогове вікно закриється, а PT- входу таймера присвоюється константа затримки роботи двигуна;

- установіть курсор перед коментарем змінної, якої присвоюється Q-вихід таймера і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я вже оголошеної змінної Mot_Time і послідовно натисніть кнопки Apply (Застосувати) і ОК;

- діалогове вікно закриється, і змінна Mot_Time присвоїться Q- виходу функціонального блока таймера;

- установіть курсор перед коментарем ET- виходу таймера і в панелі інструментів клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я внутрішньої змінної Actual_Time, яка має відображати фактичний час роботи двигуна і натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) – Time;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а ET-виходу функціонального блока M_Time присвоїться змінна Actual_Time.

Запрограмуємо лічильник активізацій двигуна:

- Позначте курсором новий рядок для продовження програми;

- активізуйте в панелі інструментів ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані виберіть меню Keywords;

- подвійно клацніть по оператору IF і на робочому листку проекту з'явиться його синтаксис з відповідним коментарем:

```
IF  (*EXPRESSION (must return a boolean value)*)
    THEN        (*If returned value of EXPRESSION = TRUE*)
    (*STATEMENT*);
END_IF;
```

- установіть курсор після оператора IF і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я вже оголошеної змінної Out і послідовно натисніть кнопки Apply (Застосувати) і ОК;

- діалогове вікно закриється, а біля оператора IF з'явиться логічна змінна Out;

- установіть курсор після оператора THEN і надрукуйте вираз Cycle_Count:=Cycle_Count+1;

Для компіляції проекту в панелі інструментів натисніть ікону Make (Створювати).

Якщо помилки є, відкоригуйте проект, якщо немає - завантажте його, використовуючи відомий шлях: Project Control Dialog → Download→Download.

Після цієї дії ST-програма набуває завершений вигляд:

```
(*Програмування лічильника кількості натисків стартової кнопки*)
Motor_Count(CU:=Motor_Start,RESET:=Motor,PV:=INT#3);
out:=Motor_Count.Q;
Pressed(* INT *):=Motor_Count.CV;
(*Програмування непередбаченої зупинки двигуна*)
RS_1(SET:=out,RESET1:=Emergency_Stop OR Mot_Time);
Motor:=RS_1.Q1;
(*Програмування таймера часу роботи двигуна*)
M_Time(IN:=Motor,PT:=T#20s);
Mot_Time:=M_Time.Q;
Actual_Time:=M_Time.ET;
(*Програмування лічильника активацій двигуна*)
IF  out
    THEN Cycle_Count:=Cycle_Count+1;
END_IF;
```

Перевіримо роботу програми:

- Натисніть кнопку Cold (Холодний) у діалоговому вікні Resource (Ресурс) для холодного запуску симулятора ПЛК;
- в панелі інструментів клацніть по іконі Debug on/off (Налагодження В/В), щоб перейти в оперативний режим роботи. При цьому усі змінні позначаться різними кольорами;
- клацніть лівою клавішею миші по Demoio_Driver, що унизу екрана, для відкриття симулятора I/O;
- тричі подвійно клацніть лівою клавішею миші по нульовому світлодіоду нульового модуля вводу In.

Програма почне виконуватися, а біля змінної Actual_Time таймера M_Time будуть змінюватися секунди затримки. При цьому з кожним новим запуском програми вміст лічильника Cycle_Count збільшується на одиницю.

```
        (*Програмування лічильника кількості натисків стартової
         кнопки*)
  FALSE Motor_Count(CU:=Motor_Start,RESET:=Motor,PV:=INT#3);
  FALSE out:=Motor_Count.Q;
      0 Pressed(* INT *):=Motor_Count.CV;
        (*Програмування непередбаченої зупинки двигуна*)
  FALSE RS_1(SET:=out,RESET1:=Emergency_Stop OR Mot_Time);
   TRUE Motor:=RS_1.Q1;
        (*Програмування таймера часу роботи двигуна*)
   TRUE M_Time(IN:=Motor,PT:=T#20s);
  FALSE Mot_Time:=M_Time.Q;
  5.700 Actual_Time:=M_Time.ET;
        (*Програмування лічильника активацій двигуна*)
  FALSE IF  out
      2     THEN Cycle_Count:=Cycle_Count+1;
        END_IF;
```

Контрольні запитання

1. Як запрограмувати ST-мовою лічильник активізацій двигуна?
2. Як оголосити змінні функціонального блока CTU?
3. Як запрограмувати час роботи двигуна?
4. Навіщо у програмі використовується функціональний блок RS?
5. Як здійснюється налагодження програми?

3.6. Методика програмування SFC-мовою

Створимо SFC-мовою програму керування роботою двигуна згідно з алгоритмом наведеним у 3.2.

Для цього відкрийте діалог New Project (Новий проект), використовуючи пункт меню File (Файл). Двічі клацніть по Project Wizard (Майстер проекту) і пройдіть увесь шлях створення проекту аналогічно варіанту програмування LD-мовою, тобто призначте ім’я проекту та його POU, виберіть у даному випадку SFC-мову програмування, бажані імена і типи конфігурації, ресурсу і задачі.

Коли новий проект з’явиться у вікні проектного дерева, можна починати програмування.

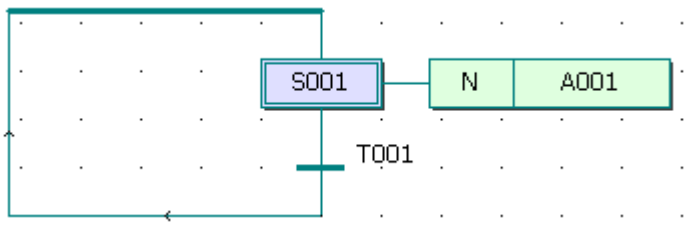
Будь- який проектний код, що створений SFC – мовою, складається з кроків і переходів, об’єднаних SFC – схемою. Тому перед тим, як їх програмувати необхідно побудувати структуру цієї схеми.

Створення SFC – схеми керування роботою двигуна:

Спочатку побудуємо послідовну мережу кроків та переходів, які забезпечують ініціалізацію змінних, програмування умов вмикання в роботу двигуна, програмування перемикача для створення можливості непередбаченого вимкнення двигуна та лічильника кількості активізацій двигуна.

На робочому полі проектного коду позначте курсором місце початку програмування;

- в панелі інструментів клацніть лівою клавішею миші по іконі Creat step transition sequence(Створити послідовні крок і перехід). На робочому полі з’явиться початкова SFC-схема з одного кроку S001 і дії A001, одного переходу T001, а також повернення назад:



Якщо з’явився не початковий крок, а звичайний, тобто прямокутник кроку створений не подвійною лінією, а одинарною, то:

- двічі клацніть лівою клавішею миші по прямокутнику і у діалоговому вікні Step (Крок), що з'явилося, активізуйте Initial step (Початковий крок);
- закрийте вікно Step (Крок), натиснувши ОК, і на робочому полі звичайний крок перетвориться у початковий.

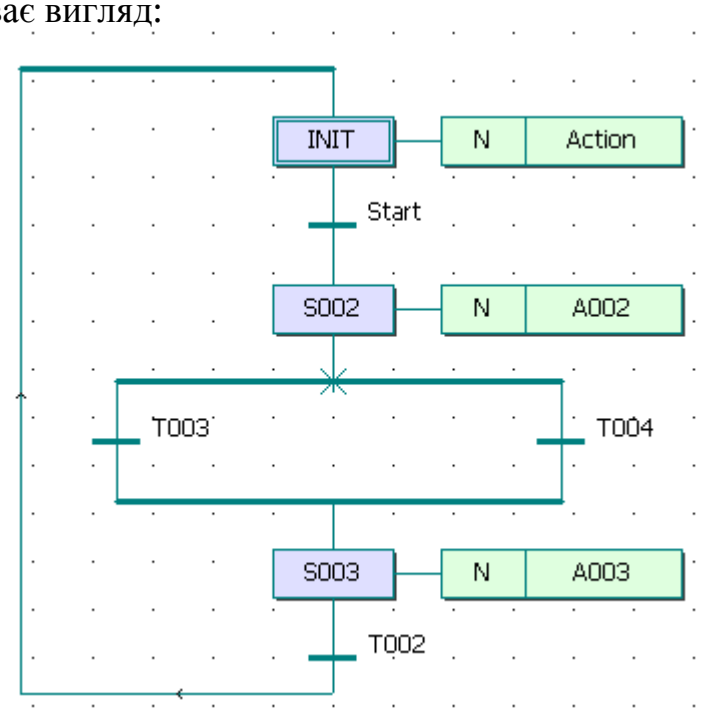
Для призначення імені компонентам SFC-схеми :

- клацніть правою клавішею миші по кроку S001 і в меню, що з'явилося, лівою клавішею миші активізуйте Object Properties… У вікні Step у рядку Name надрукуйте ім'я INIT і клацніть ОК;
- таке саме зробіть послідовно з дією A001 та переходом T001, привласнюючи їм імена відповідно Action з класифікатором (Qualifier) N замість A001у діалоговому вікні Action Properties (Властивості дії) та Start замість T001у діалоговому вікні Transition (Перехід), при цьому в обох вікнах альтернатива Detail (Деталь) має бути активізована;

Для створення другої пари (крок і перехід) клацніть лівою клавішею миші спочатку по переходу Start, а потім в панелі інструментів по іконі Creat step transition sequence(Створити послідовні крок і перехід). В наслідок цього існуюча SFC-схема збільшиться ще на один крок S002 і перехід T002. При цьому присвоювати імена їм поки не будемо, оскільки далі на базі другої пари кроку і переходу створимо у SFC-схемі розгалуження;

Для створення альтернативних гілок розгалуження позначте курсором другий крок S002 SFC-схеми і в панелі інструментів клацніть по іконі Insert Simultaneous/Alternative Divergence(Вставити Паралельне/Альтернативне розгалуження);

- у діалоговому вікні Divergence (Розгалуження), що з'явилося, у полі Branches Count (Кількість гілок) надрукуйте 2 і клацніть ОК. Проектний код у робочому полі набуває вигляд:



У лівій та правій гілках альтернативного розгалуження створимо стрибки на початковий крок INIT. Для цього:

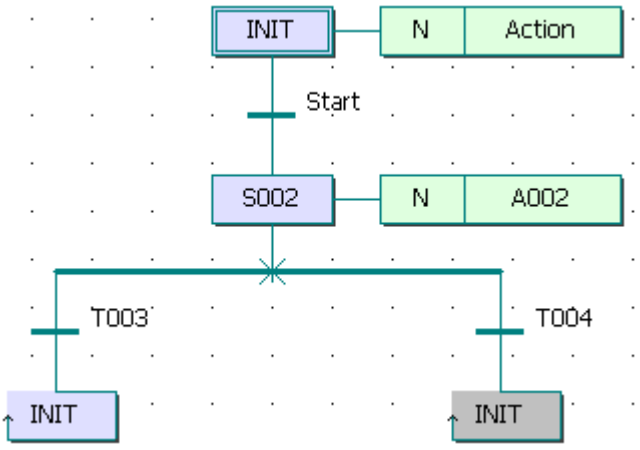
- позначте курсором перехід T003 і в панелі інструментів клацніть по іконі Creat step transition sequence(Створити послідовні крок і перехід), у лівій гільці з'являться крок S004 з дією A004 і перехід T005;

- правою клавішею миші клацніть по кроку S004 і в меню, що з'явилося, лівою клавішею миші виберіть Object Properties… У полі Type активізуйте альтернативу Jump(Стрибок) і клацніть ОК;

- у діалоговому вікні Jump/End step insert control (Стрибок/Кінець управління кроком вставки), що з'явилося, натисніть Change(Змінити) і в лівій гілці залишаться тільки перехід T003 і безадресний стрибок S004 у вигляді прямокутника зі стрілкою;

- щоб призначити адресу стрибка S004 лівою клавішею миші подвійно клацніть по ньому і у діалоговому вікні Step(Крок), що з'явилося, вкажіть у полі Name ім'я адреси – початковий крок INIT. Натисніть ОК і у прямокутнику стрибка замість S004 з'явиться INIT;

- такі самі перетворення, тільки на базі переходу T004, зробіть і з правою гілкою SFC-схеми. Після цього SFC-схема набуває вигляд:



- призначте новим компонентам SFC-схеми, що з'явилися на робочому полі (крок S002, дія A002 і переходи T003 і T004), імена відповідно Step1, Action1, Trans і Trans1, користуючись вже описаними раніше правилами.

Програмування кроків і переходів:

Після створення SFC-схеми необхідно запрограмувати дії кроків і умови переходів відповідно до алгоритму керування роботою двигуна.

Згідно з принципами SFC – мови, перший крок є ініціалізаційним. Тому змінній керування роботою двигуна Motor необхідно присвоїти значення FALSE, а лічильник скинути. Для цього:

- лівою клавішею миші двічі клацніть по дії початкового кроку Action, у діалоговому вікні Insert (Вставлення), що з'явилося, виберіть ST-мову програмування і натисніть ОК. Відкриється робоче поле дії Action;

- користуючись синтаксисом ST-мови присвойте змінній motor значення FALSE, тобто надрукуйте: Motor :=FALSE;

- перейдіть на новий рядок і в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту), у таблиці Group, що з'явилася на екрані, виберіть меню Keywords;

- подвійно клацніть по оператору IF і на робочому листку проекту з'явиться його синтаксис з відповідним коментарем:

```
IF  (*EXPRESSION (must return a boolean value)*)
    THEN        (*If returned value of EXPRESSION = TRUE*)
    (*STATEMENT*);
END_IF;
```

- установіть курсор після оператора IF і надрукуйте вираз Cycle_Count>10;

- установіть курсор після оператора THEN і надрукуйте вираз Cycle_Count:=Cycle_Count+1.

Створений фрагмент програми дії Action забезпечує обнуління лічильника після 10 активізацій двигуна.

Після цього ST-програма дії Action набуває завершений вигляд:

```
motor:=FALSE;
IF Cycle_Count>10   (*EXPRESSION (must return a boolean value)*)
    THEN Cycle_Count:=0(*If returned value of EXPRESSION = TRUE*
    (*STATEMENT*);
END_IF;
```

Тут: Motor - змінна, що керує роботою двигуна;

Cycle_Count – змінна, що рахує кількість активізацій двигуна.

Щоб оголосити ці змінні, установіть курсор спочатку на ім'я Motor і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variables (Змінні), що з'явилося, у рядку Name побачите ім'я змінної Motor, натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний) і у рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;

- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вихідного каналу модуля виводу %QX0.0;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а змінна Motor автоматично з'явиться у списку змінних проекту.

Після цього установіть курсор на ім'я внутрішньої змінної Cycle_Count і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variables (Змінні), що з'явилося, у рядку Name побачите ім'я змінної Cycle_Count, натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний) і у рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - INT;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- закрийте робоче поле дії Action і запам’ятайте зроблену ST-програму, натиснувши у діалоговому вікні MULTIPROG, що з’явилося, відповідну кнопку.

Програмування переходу Start:

Умовою початку роботи двигуна є триразове натиснення стартової кнопки. Запрограмуємо її LD-мовою, використовуючи прямий зв'язок, коли перехід не має тіла коду, а безпосередньо сполучається з LD-мережею. У даному випадку перехід не повинен мати ім’я.

Щоб реалізувати цей намір:

- клацніть правою клавішею миші по переходу Start, у діалоговому вікні Trasition (Перехід), що з’явилося, активізуйте альтернативу Direct Connection (Безпосереднє сполучення), рядок Name (Ім’я) зробіть порожнім і натисніть ОК;

- діалогове вікно закриється, а у робочому полі SFC - програми після початкового кроку INIT залишиться безіменна смужка переходу з блакитною точкою;

- для створення умови переходу LD-мовою клацніть лівою клавішею миші у зручному місці робочого поля зліва від безіменного переходу;

- в панелі інструментів клацніть по іконі Contact network (Мережа контактів) і в установленому місці робочого поля з’явиться LD- мережа 001 певної ширини з контактом і котушкою;

- вилучте з LD- мережі праву шину живлення і котушку C001 послідовно помічаючи їх лівою клавішею миші і натискаючи клавішу Delet на клавіатурі ЕОМ;

- в панелі інструментів активізуйте ікону Connect objects (З’єднати об’єкти) і за допомогою курсору з’єднайте прямий контакт C000 LD- мережі з блакитною точкою смужки безіменного переходу;

- лівою клавішею миші позначте лінію, що сполучає прямий контакт і перехід;

- в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування) і з таблиці Group (Група), що з’явилася на екрані, виберіть функціональний блок лічильника CTU;

- двічі клацніть по блоку CTU, у діалоговому вікні Variable properties (Властивості змінної), що з’явилося, змініть типове ім’я CTU-1 на Motor-Count;

- натисніть послідовно кнопки Apply (Застосувати) і ОК, підтверджуючи діалог;

- в панелі інструментів клацніть по іконі Edit Wizard (Майстер редагування) , щоб сховати Майстра редагування.

Таким чином на позначеному місці LD- мережі з’явиться лічильник.

- Клацніть на функціональному блоці CTU по синій точці Reset-входу;

- в панелі інструментів клацніть по іконі Add contact left (Додати контакт зліва) і на Reset -вході лічильника з’явиться контакт C002;

- в панелі інструментів активізуйте кнопку Connect objects (З'єднати об'єкти);

- клацніть по контакту C002 і перемістіть його курсором ліворуч до шини живлення 001 так, щоб він опинився під контактом C000;

- в панелі інструментів клацніть по іконі Mark (Позначка), щоб деактивізувати кнопку Connect objects (З'єднати об'єкт).

Оголошення властивостей прямих контактів LD- мережі:

- двічі клацніть лівою клавішею миші по контакту C000, щоб оголосити змінну, яка має запустити двигун. У діалоговому вікні Contact/Coil Properties (Властивості Контактів/Котушок), що з'явилося, замініть типове ім'я змінної C000 на Motor_Start;

- натисніть кнопку Apply (Застосувати), автоматично відкриється сторінка діалогу Common (Загальний). У вікні списку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL.

Призначімо змінній Motor_Start фізичну адресу симулянта модуля вводу/виводу I/O ПЛК:

- введіть у поле I/O address (Адреса вводу/виводу) фізичну адресу симулянта модуля вводу ПЛК - %IX0.0;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте в неї Default;

- клацніть ОК, підтверджуючи діалог Contact/Coil Properties (Властивості Контактів/Котушок). Після цього змінна є оголошеною і вставленою у відповідний список, а на робочому листку проекту замість C000 з'явиться локальна зміна Motor_Start;

- двічі клацніть по контакту C002 і у діалоговому вікні Contact/Coil Properties (Властивості Контакт/котушка), що з'явилося, з існуючого списку вже оголошених змінних виберіть Motor і клацніть лівою клавішею миші.

Оскільки змінна Motor визначена раніше, немає потреби конкретизувати Local scope (Локальна компетенція).

- Клацніть ОК і на робочому листку з'явиться змінна Motor замість C002;

- для визначення параметрів лічильника, двічі клацніть по блакитній точці PV-входу, з'явиться діалогове вікно Variable properties (Властивості змінної);

- у рядку Name (Ім'я) введіть константу INT#3, щоб двигун (Motor) спрацював після триразового натиснення стартової кнопки;

- клацніть ОК і на PV-вході лічильника з'явиться INT#3, тобто ціле число є вставленим в тіло коду;

- двічі клацніть по зеленій точці CV-виходу лічильника, з'явиться діалогове вікно Variable properties (Властивості змінної);

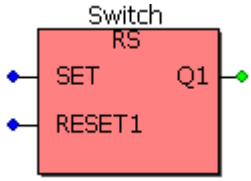
- у рядку Name (Ім'я) надрукуйте Pressed - це внутрішня змінна, яка запам'ятовує поточне значення лічильника;

- натисніть кнопку Apply (Застосувати), відкриється сторінка діалогу Common(Загальний);
- у рядку Usage (Вживання) виберіть VAR, а в рядку Date type (Тип даних) – INT, оскільки змінною є число;
- клацніть ОК і у робочому листі поряд з CV-виходом з'явиться Pressed;

Програмування дії Action1 кроку Step1:

Після триразового натиснення кнопки Motor_Start має працювати двигун. Тому у наступному кроці Step1 запрограмуємо роботу двигуна, передбачивши можливість його екстреного зупинення. Для цього:

- лівою клавішею миші двічі клацніть по дії першого кроку Action1, у діалоговому вікні Insert (Вставлення), що з'явилося, виберіть FBD-мову програмування і натисніть ОК;
- у робочому полі дії Action1, що з'явилося, лівою клавішею миші позначте місце створення програмного коду;
- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть функціональний блок перемикача з домінантою вимкнення RS;
- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я RS -1 на Switch;
- клацніть ОК, діалогове вікно закриється, а у робочому листку дії Action1 залишиться функціональний блок перемикача Switch створений FBD-мовою:



Оголошення змінних блоку Switch:

- двічі клацніть по SET- входу перемикача і у діалоговому вікні Variable properties (Властивості змінної), що з'явилося, у полі Name надрукуйте ім'я виходу лічильника кількості натисків стартової кнопки - Motor_Count.Q;
- послідовно натисніть кнопки Apply (Застосувати) і ОК, діалогове вікно закриється, а SET- входу перемикача присвоїться змінна Motor_Count.Q ;
- двічі клацніть по RESET1- входу перемикача і у діалоговому вікні Variable properties (Властивості змінної), що з'явилося, у полі Name надрукуйте ім'я нової змінної Emergency_Stop(Непередбачене зупинення);
- натисніть кнопки Apply (Застосувати) і відкрийте сторінку діалогу Common (Загальний блок). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) - BOOL;
- у рядку I/O address (Адреса вводу/виводу) надрукуйте фізичну адресу вхідного каналу модуля вводу %IX0.1;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а RESET1- входу перемикача Switch присвоїться змінна Emergency_Stop (Непередбачене зупинення);

- двічі клацніть по Q1- виходу перемикача і у діалоговому вікні Variable properties (Властивості змінної), що з'явилося, у полі Name надрукуйте ім'я вже оголошеної змінної Motor;

- послідовно натисніть кнопки Apply (Застосувати) і ОК, діалогове вікно закриється, а Q1- виходу перемикача присвоїться змінна Motor;

- закрийте робоче поле дії Action1 кроку Step1, а його вміст запам'ятайте, натиснувши відповідну кнопку у діалоговому вікні MULTIPROG.

Разом з початком роботи двигуна має працювати лічильник його активізацій. Тому для першого кроку створимо ще одну дію, яка має виконуватися разом з дією Action1. Для цього:

- на SFC-схемі лівою клавішею миші клацніть по прямокутнику дії Action1, а в панелі інструментів по іконі Create action (Створення дії). Поряд з прямокутником дії Action1 з'явиться прямокутник іншої дії, якому присвойте ім'я Counter, керуючись вже відомою схемою.

Програмування дії Counter кроку Step1:

Створимо лічильник, який рахує кожне вмикання двигуна. При цьому після 10 циклів роботи показання лічильника скидаються у нуль.

- Лівою клавішею миші двічі клацніть по дії Counter;

- у діалоговому вікні, що з'явилося, виберіть ST-мову, закрийте вікно і на екрані з'явиться робоче поле для програмування лічильника;

- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, виберіть альтернативу Keywords (Ключові слова);

- двічі клацніть по оператору IF і у робочому полі з'явиться його синтаксис з відповідним коментарем:

```
IF   (*EXPRESSION (must return a boolean value)*)
     THEN        (*If returned value of EXPRESSION = TRUE*)
     (*STATEMENT*);
END_IF;
```

Оголошення змінних дії Counter:

- Установіть курсор після оператора IF і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я виходу лічильника кількості натисків стартової кнопки - Motor_Count.Q і послідовно натисніть кнопки Apply (Застосувати) і ОК;

- діалогове вікно закриється, а біля оператора IF з'явиться логічна змінна Motor_Count.Q;

- установіть курсор після оператора THEN і надрукуйте вираз Cycle_Count:=Cycle_Count+1.

Оскільки змінна Cycle_Count вже оголошена, лічильник активізацій двигуна є створеним. Закрийте робоче поле переходу, а його код запам'ятайте.

Програмування переходу Trans:

Щоб показання лічильника скидалися після 10 активізацій двигуна створимо IL – мовою відповідну програму переходу Trans . Для цього:

- лівою клавішею миші двічі клацніть по переходу Trans;
- у діалоговому вікні, що з'явилося, виберіть IL-мову, закрийте його і на екрані з'явиться робоче поле для програмування;
- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані, в альтернативі Operators (Оператори) виберіть оператор GT (Більш ніж);
- двічі клацніть по ньому і на робочому полі переходу Trans з'явиться синтаксис цього оператора:

```
LD  var_1
GT  var_2
ST  var_out
```

Замість var1 надрукуйте вже оголошену змінну Cycle_Count, замість var2 – константу 10, а замість var_out – ім'я переходу Trans;

- установіть курсор на ім'я переходу Trans і клацніть лівою клавішею миші, з'явиться діалогове вікно Variables, у рядку Name якого надруковано ім'я переходу Trans;
- лівою клавішею миші послідовно клацніть по Apply (Застосувати) і OK;
- закрийте робоче поле переходу Trans і запам'ятайте створену програму.

Програмування часу роботи двигуна:

Оскільки двигун після вмикання працює 20 секунд, умовою переходу Trans1 має бути саме 20с, які відлічує таймер.

Для програмування таймера ST- мовою:

- лівою клавішею миші двічі клацніть по переходу Trans1;
- у діалоговому вікні, що з'явилося, виберіть ST-мову, закрийте вікно і на екрані з'явиться робоче поле для програмування таймера;
- в панелі інструментів активізуйте ікону Edit Wizard (Майстер проекту) і у таблиці Group, що з'явилася на екрані виберіть функціональний блок таймера TON;
- двічі клацніть по ньому і у діалоговому вікні Variable properties (Властивості змінної) змініть типове ім'я TON_1 на Motor_Time;
- натисніть кнопку Apply (Застосувати) і клацніть OK, діалогове вікно закриється, а в робочому полі залишиться ST- програма функціонального блока таймера Motor_Time:

```
Motor_Time(IN:=(* BOOL *),PT:=(* TIME *));
(* BOOL *):=Motor_Time.Q;
(* TIME *):=Motor_Time.ET;
```

- установіть курсор перед коментарем IN- входу функціонального блока Motor_Time і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я Step1.X, яке відповідає назві кроку Step1 і послідовно натисніть кнопки Apply (Застосувати) і ОК.

- діалогове вікно закриється, а IN- входу функціонального блока таймера Motor_Time присвоїться змінна Step1.X;

- установіть курсор перед коментарем PT- входу і в панелі інструментів клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте константу T#20s, яка встановлює час роботи двигуна і натисніть послідовно кнопки Apply (Застосувати) і ОК;

- діалогове вікно закриється, а PT- входу таймера присвоїться константа затримки роботи двигуна;

- установіть курсор перед коментарем змінної, яка присвоюється Q- виходу таймера і в панелі інструментів лівою клавішею миші клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я переходу Trans1 і послідовно натисніть кнопки Apply (Застосувати) і ОК;

- діалогове вікно закриється, і змінна Trans1 присвоїться Q- виходу функціонального блока таймера;

- установіть курсор перед коментарем ET- виходу таймера і в панелі інструментів клацніть по іконі Variable (Змінна);

- у діалоговому вікні Variable Properties (Властивості змінної), що з'явилося, у рядку Name надрукуйте ім'я внутрішньої змінної Act_Time, яка має відображати фактичний час роботи двигуна і натисніть кнопку Apply (Застосувати);

- відкрийте сторінку діалогу Common (Загальний). У рядку Usage (Вживання) виберіть VAR, а у рядку Data Type (Тип даних) – Time;

- відкрийте сторінку діалогу Local scope (Локальна компетенція) і активізуйте Default;

- натисніть ОК, діалогове вікно закриється, а ET-виходу функціонального блока Motor_Time присвоїться змінна Act_Time.

Таким чином, відповідно до створеної програми після триразового натиснення кнопки Motor_Start вмикається двигун і лічильник його активізацій. Оскільки у лівій гілці альтернативного розгалуження умови для переходу Trans відбуваються тільки після 10 вмикань двигуна програма виконується відповідно до умов переходу Trans1 правої гилки розгалуження, тобто після 20секунд роботи двигуна у виконанні програми здійснюється стрибок на початковий крок

і робота двигуна зупиняється. Для повторного запуску двигуна необхідно знову тричі натиснути стартову кнопку. Після десятої активізації роботи двигуна виконуються умови переходу Trans лівої гілки розгалуження і стрибок на початковий крок вже здійснюється через неї. Це призводить не тільки до зупинки двигуна, а і до обнуління лічильника його активізацій.

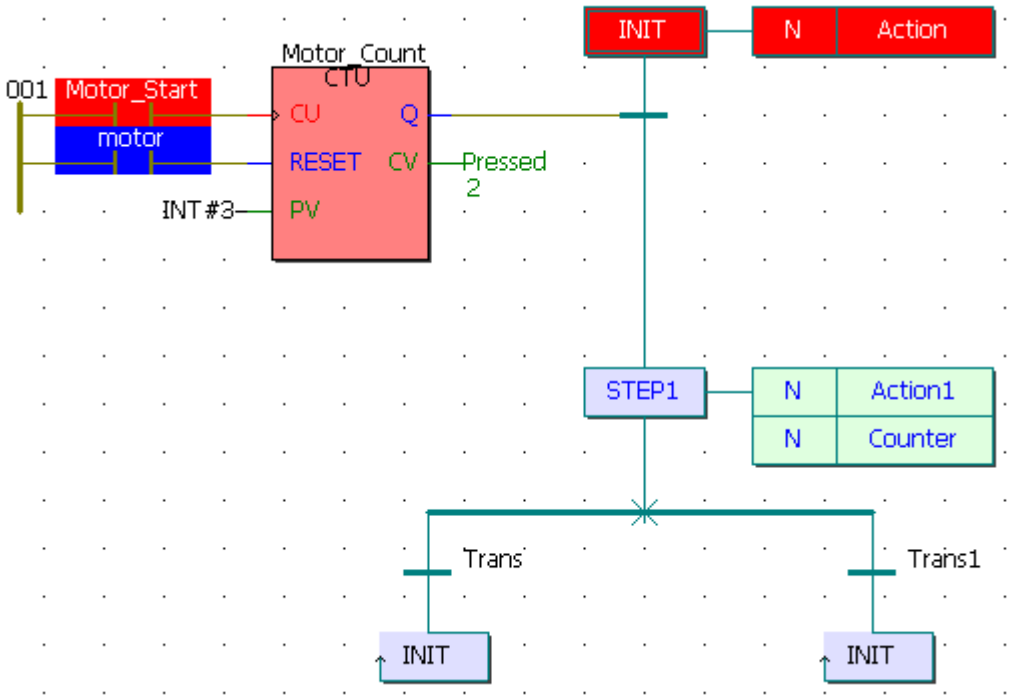
Після створення проектного коду в панелі інструментів натисніть ікону Make (Створювати), для компіляції проекту.

Якщо помилки є, відкоригуйте проект, якщо немає - завантажте його, використовуючи відомий шлях: Project Control Dialog → Download→Download.

Перевірка роботи програми:

- Для холодного запуску симулятора ПЛК натисніть кнопку Cold (Холодний) у діалоговому вікні Resource (Ресурс), що з'являється після активізації ікони Project Control Dialog (Діалог управління проектом) в панелі інструментів;
- в панелі інструментів клацніть по іконі Debug on/off (Налагодження В/В), щоб перейти в оперативний режим роботи. При цьому усі змінні позначаться різними кольорами;
- лівою клавішею миші клацніть по Demoio_Driver унизу екрана, для відкриття симулятора I/O;
- тричі подвійно клацніть лівою клавішею миші по нульовому світлодіоду нульового модуля вводу In.

Програма почне виконуватися, про що свідчить перехід червоного кольору від початкового кроку INIT до кроку STEP1 після триразового натиснення на нульовий світлодіод нульового модуля вводу симулятора.



Після затримки на 20 секунд червоний колір від кроку STEP1 через перехід Trans1 перестрибує на початковий крок INIT. При цьому з кожним

новим запуском програми у дії Counter змінна лічильника Cycle_Count збільшується на одиницю. Для екстреної зупинки двигуна необхідно натиснути на перший світлодіод нульового модуля вводу. Коли кількість активізацій двигуна перевищує цифру 10, виконання програми продовжується через перехід Trans лівої гилки розгалуження на початковий крок INIT. При цьому робота двигуна зупиняється, а лічильник скидається у нуль.

Контрольні запитання

1. Як створюється SFC-схема?
2. Як здійснюється програмування кроків і переходів SFC-схеми?
3. Як запрограмувати умови переходу LD-мовою безпосередньо на SFC- схемі?
4. Як запрограмувати ST-мовою дію Counter кроку Step1?
5. Коли у режимі налагодження елементи SFC-схеми позначаються червоним кольором?

4. ПРИКЛАДИ ПРОГРАМУВАННЯ

4.1. Керування роботою ліфта

Ліфт призначений для обслуговування триповерхового будинку. Для виклику кабіни ліфта потрібно натиснути кнопку, яка є на кожному поверсі. При цьому, кнопка знаходиться у натиснутому стані до тих пір, поки кабіна ліфта не приїде на поверх. Виклик ліфта можна здійснити з різних поверхів одночасно, тоді ліфт зупиняється на кожному з них. Зайнятість ліфта сигналізує лампочка, що знаходиться біля кнопки виклику. На кожному поверсі можна бачити напрям руху ліфта – уверх чи униз, а також положення кабіни ліфта. Роботою ліфта можна керувати зсередини кабіни за допомогою кнопок відповідних поверхів.

Змінні, що використовуються:

go – Вхідна змінна, що визначає позицію ліфта на 1-му поверсі;

go1-go3 – Вхідні змінні, що визначають положення кнопок виклику кабіни на кожному поверсі;

C_go1-C_go3 - Вхідні змінні, що визначають положення кнопок замовленого поверху всередині кабіни;

Level_1-Level_3 – Вихідні змінні, стан яких визначається кнопками виклику ліфта на кожному поверсі;

Door_L_1- Door_L_3 – Вихідні змінні, що керують роботою дверей кабіни ліфта на відповідному поверсі;

Lift_Work – Вихідна змінна, яка відображає стан світлового індикатора роботи ліфта;

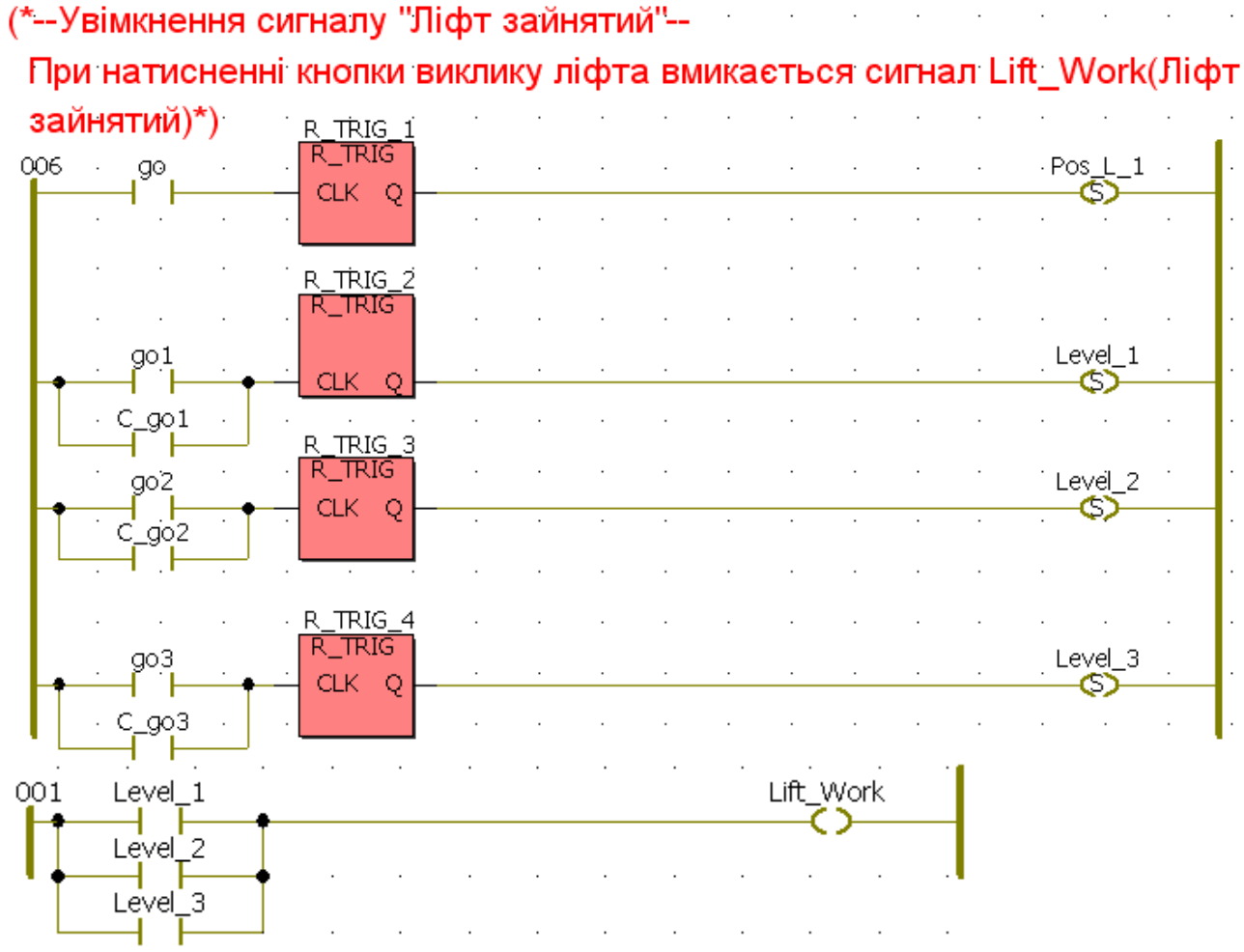
Matr_Ready – Внутрішня вихідна змінна, яка відображає стан матриці;

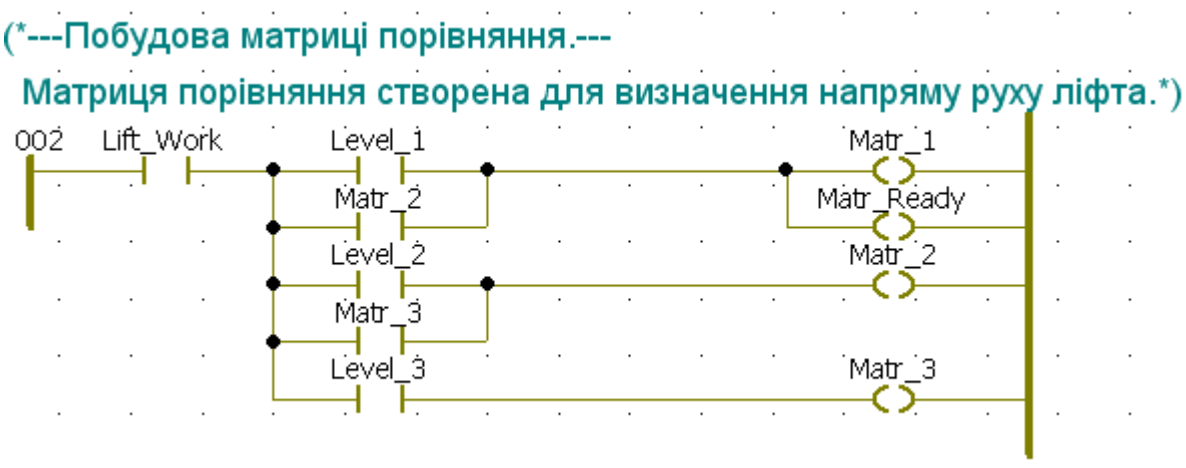
Matr_1- Matr_3 – Внутрішні вихідні змінні матриці порівняння;

Pos_L1 - Pos_L3 – Вихідні змінні , які відповідають індикаторам розташування ліфта на поверхах;

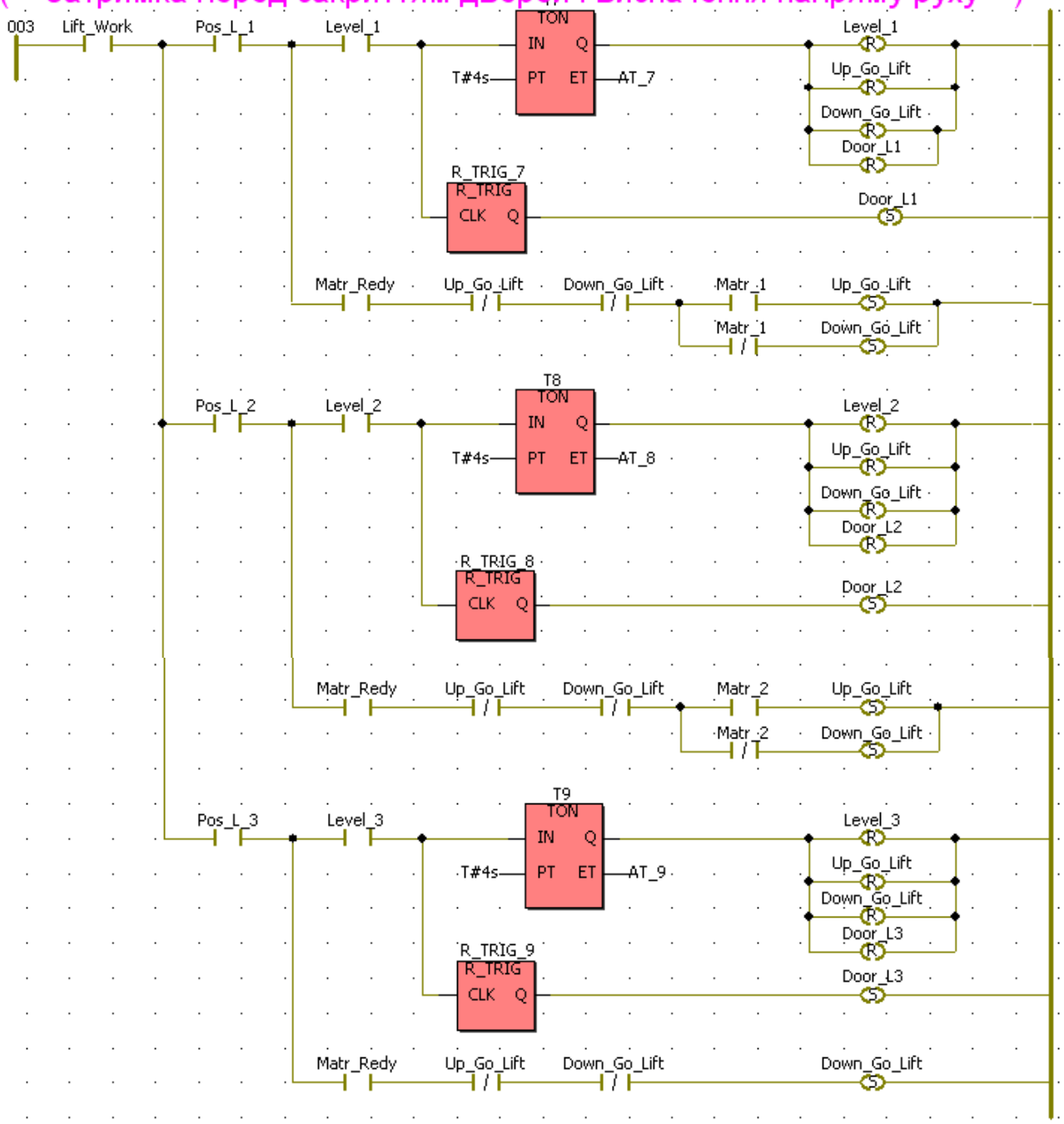
Up_Go_Lift – Вихідна змінна, що відповідає індикатору руху ліфта уверх; Down_Go_Lift – Вихідна змінна, що відповідає індикатору руху ліфта униз.

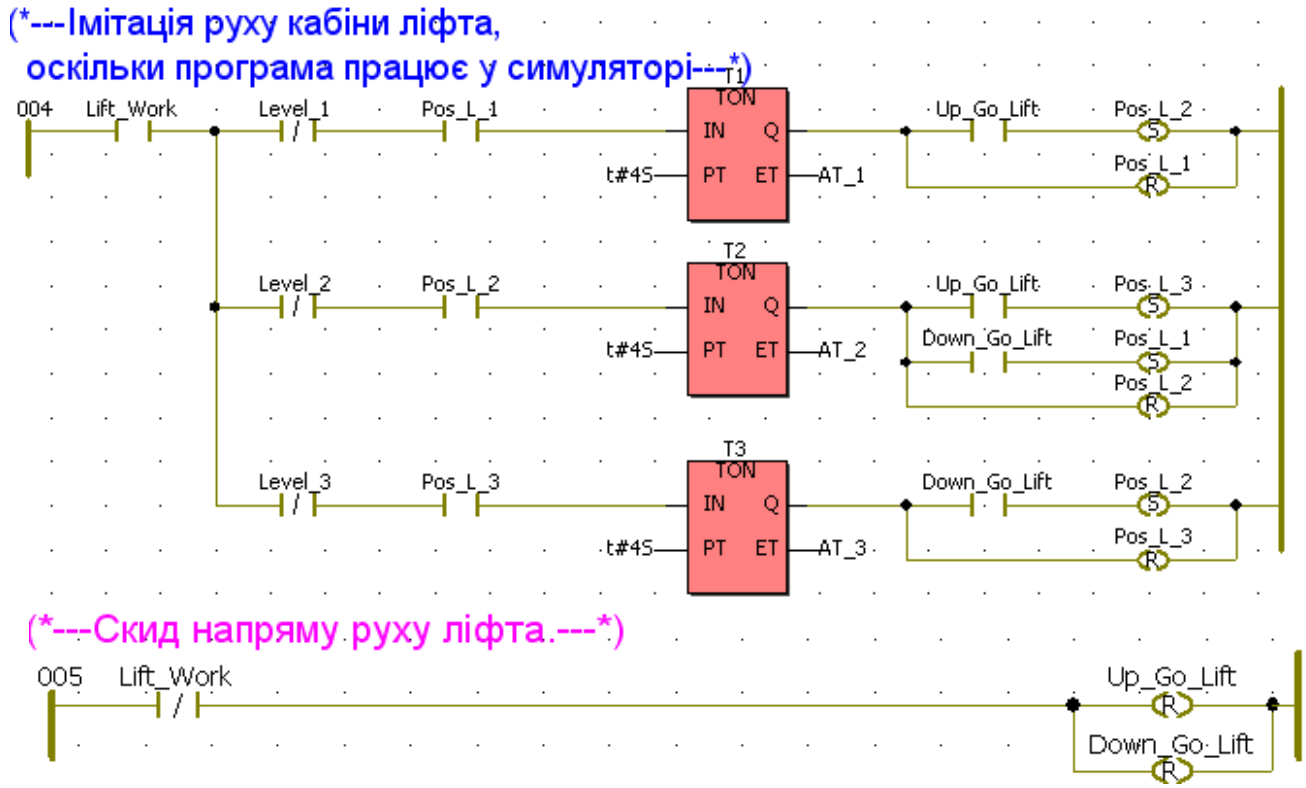
Програма керування роботою ліфта LD-мовою має вигляд:





(*--Затримка перед закриттям дверей і визначення напряму руху--*)





Для роботи з програмою керування ліфтом потрібно активізувати, а потім скинути вхідну змінну «go», що визначає позицію ліфта на 1-му поверсі. Потім активізувати і скинути – вхідні змінні «go1-go3» , що визначають положення кнопок виклику кабіни на кожному поверсі. Здійснюється це тому, що при роботі програми в режимі симулятора вхідні змінні програмно не скидаються.

Програма працює наступним чином.

При натисненні кнопки виклику кабіни ліфта на будь-якому поверсі, стан відповідної вхідної змінної (go1-go3) набуває істинного значення. При цьому активізується відповідна вихідна змінна (Level_1-Level_3) і фіксується. Змінна Lift_Work також набуває істинного значення. Тобто світловий індикатор роботи ліфта показує, що ліфт зайнятий.

На наступному етапі програми відбувається побудова матриці порівняння, яка необхідна для визначення напряму руху кабіни ліфта. Як тільки стан матриці визначився відповідно до натиснутої кнопки (вихідної змінної Level_1-Level_3), спрацьовують відповідні змінні (Matr_1- Matr_3) і внутрішня вихідна змінна Matr_Ready. Істинний стан внутрішніх змінних Matr_Ready, Matr_1- Matr_3 і Level_1-Level_3, визначений натиснутою кнопкою, а значить відповідною вхідною змінною (go1-go3) забезпечує рух кабіни у бажаному напрямі.

У фрагменті програми «Затримка перед закриттям дверей і визначення напряму руху» відбувається порівняння поточного положення кабіни ліфта – стан змінних (Pos_L1 - Pos_L3) з натиснутою кнопкою на відповідному поверсі – стан змінних (Level_1-Level_3). При збігу цих параметрів відкриваються двері кабіни ліфта і запускається відповідний таймер на чотири секунди. Коли час спливає, двері закриваються, відновлюється у

первинний стан положення натиснутої кнопки виклику кабіни ліфта, а також скидаються змінні Up_Go_Lift і Down_Go_Lift, що визначають роботу світлових індикаторів напряму руху кабіни ліфта.

Якщо знову здійснено виклик кабіни ліфта - з будь-якого іншого поверху (змінні кнопок go1-go3), або з середини ліфта (змінні кнопок C_go1-C_go3), знову спрацьовує матриця порівняння і визначається напрям руху ліфта. При істинному стані внутрішньої вихідної змінної (Matr_1- Matr_3) і матриці порівняння (Matr_Ready), а також змінній (Level_1-Level_3), які визначають поточний поверх, напрям руху ліфта встановлюється уверх, інакше, коли значення змінної Lift_Work істинне – униз.

Оскільки програма працює у симуляторі, необхідно створити фрагмент програми, що імітує положення кабіни ліфта, відповідно до натиснутої кнопки виклику.

У фрагменті «Імітація руху кабіни ліфта» відбувається порівняння положення кабіни ліфта з кнопкою виклику кабіни ліфта на цьому ж поверсі. При не натиснутій кнопці виклику кабіни ліфта і знаходженні кабіни ліфта на тому ж поверсі, що і кнопка, запускається таймер і по закінченню 4-х секунд змінюється позиція ліфта залежно від напряму руху кабіни ліфта.

Коли стан змінної Lift_Work – ХИБНИЙ, тобто ліфт не працює, скидаються змінні, що визначають напрям руху кабіни ліфта.

4.2.Керування температурою рідини в технологічному апараті

Нагрів рідини в технологічному апараті відбувається парою шляхом збільшення або зменшення її витрати при досягненні температурою рідини значень - відповідно Tmin і Tmax. Керування роботою виконавчого механізму (ВМ) має бути заблоковано і додатково передбачена сигналізація при повністю відкритому або закритому регулювальному органі. Окрім того, схема керування має формувати сигнал застереження при одночасній появі сигналів Tmin і Tmax, а також при наявності мінімальної температури Tmin і сигналізації відкритого стану регулювального органу.

Програма керування температурою рідини IL-мовою , використовує такі змінні:

Tmin - мінімальна температура рідини;

Tmax – максимальна температура рідини;

D_Pr – прямий хід ВМ на відкриття регулювального органу;

D_Rev – зворотний хід ВМ на закриття регулювального органу;

Z_On – сигналізація відкритого стану регулювального органу;

Z_Off - сигналізація закритого стану регулювального органу;

Timer_1 – таймер тривалості роботи ВМ на відкриття регулювального органу;

Timer_2 - таймер тривалості роботи ВМ на закриття регулювального органу;

F – сигнал застереження при одночасній появі сигналів Tmin і Tmax;

F1- сигнал застереження при одночасній появі сигналів Tmin і Z_On;

```
(*Програмування керувального впливу на збільшення температури
  рідини*)
LD      Tmin
ANDN    Tmax
S       D_Pr
R       D_Rev
LD      D_Pr
ST      Timer_1.IN
LD      TIME#10s
ST      Timer_1.PT
CAL     Timer_1
LD      Timer_1.Q
R       D_Pr
S       Z_On
LD      Timer_1.ET
ST      Real_Time_1
(*Програмування керувального впливу на зменшення температури
  рідини*)
LD      Tmax
ANDN    Tmin
S       D_Rev
R       D_Pr
LD      D_Rev
ST      Timer_2.IN
LD      TIME#10s
ST      Timer_2.PT
CAL     Timer_2
LD      Timer_2.Q
R       D_Rev
S       Z_Off
LD      Timer_2.ET
ST      Real_Time_2
(*Блокування роботи ВМ на відкриття при відкритому стані
  заслінки*)
LD      Z_On
R       D_Pr
(*Блокування роботи ВМ на закриття при закритому стані
  заслінки*)
LD      Z_Off
R       D_Rev
(*Блокування появи сигналізації про відкриту заслінку при
 наявності керувального впливу на її закриття*)
LD      D_Rev
R       Z_On
(*Блокування появи сигналізації про закриту заслінку при
 наявності керувального впливу на її відкриття*)
LD      D_Pr
R       Z_Off
```

```
(*Програмування аварійних сигналів*)
LD      Tmax
AND     Tmin
ST      F
LD      Tmin
AND     Z_On
ST      IL_Variable_1
LD      Tmax
AND     Z_Off
OR      IL_Variable_1
ST      F1
LD      F
R       F1
```

Програма працює наступним чином.

Коли температура рідини досягає значення Tmin, виконавчий механізм починає відкривати (D_Pr) регулювальний орган. Витрата пари на нагрів збільшується і температура рідини росте. При цьому сигнал, відповідний значенню Tmin, зникає. За 10с регулювальний орган повністю відкриється і спрацює сигналізація (Z_On). При збільшені температури рідини до Tmax, виконавчий механізм починає закривати (D_Rev) регулювальний орган і за 10с спрацьовує сигналізація повного його закриття (Z_Off). В наслідок такої роботи температура рідини починає зменшуватися, сигнал, що відповідає її максимальному значенню - зникає, а при досягненні температурою значення Tmin процес керування повторюється. В програмі передбачені відповідні сигналізації, блокування та аварійні застереження, які супроводжуються коментарем.

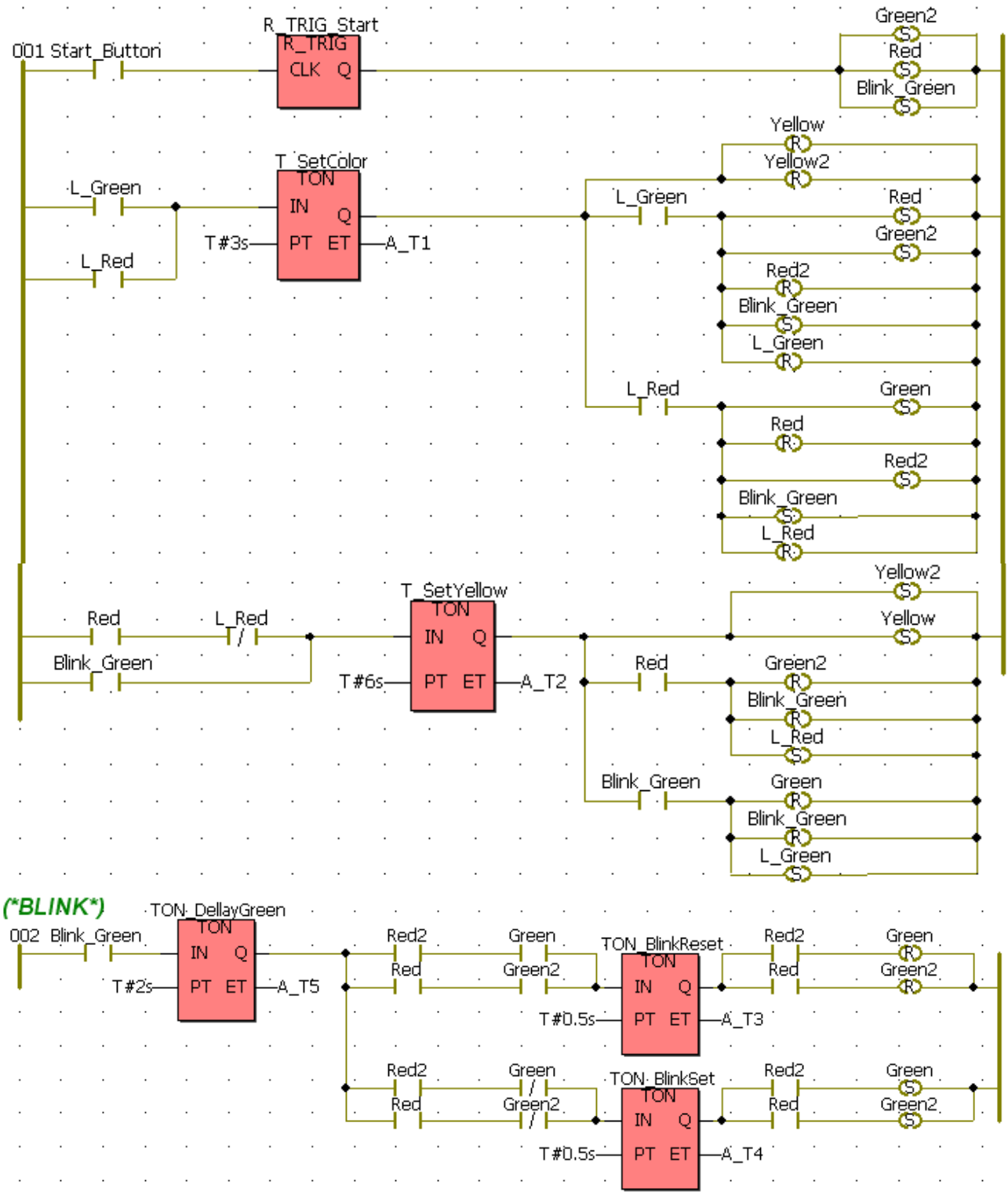
4.3. **Керування роботою світлофора**

Створимо програму роботи світлофора, який керує рухом на перехресті двох доріг. Зрозуміло, що світлофори повинні мати два протилежні стани – червоний і зелений з міготінням. Щоб уникнути небажаних випадків, додамо загальноприйняті перехідні стадії- жовтий і жовто-червоний. Остання стадія повинна бути довше за попередньою.

У якості змінних використовуються такі:

Start_Button – запуск програми;
Red, Red2 –червоний колір 1-го та 2-го світлофорів;
Yellow, Yellou2 –жовтий колір 1-го та 2-го світлофорів;
Green, Green2 –зелений колір 1-го та 2-го світлофорів;
Blink_Green- ознака миготіння зеленого кольору світлофорів;
L_Green – ознака того, що останнім кольором світлофору був зелений;
L_Red – ознака того, що останнім кольором світлофору був червоний.

LD-програма керування роботою світлофора має наступний вигляд:



1. Програма запускається в роботу, коли змінна Start_Button набуває стан TRUE. Внаслідок цього спрацьовують котушки Red і Green2, тобто з’являються червоний колір на першому світлофорі та зелений – на другому. Одночасно спрацьовує котушка Blink_Green, яка є внутрішньою вихідною програми і керує роботою таймера T_SetYellow, а також фрагментом програми (*BLINK*), який забезпечує миготіння зеленого кольору.

2. Таймер T_SetYellow починає відлік часу, за яким установляться котушки Yellow, Yellow2 і L_Red, а також скинуться змінні Green2, Blink_Green. Поки таймер T_SetYellow відлічує установлені 6с, фрагмент програми (*BLINK*) після затримки у 2с, створеної таймером TON_DelayGreen, тричі за допомогою таймерів TON_BlikReset і TON_BlinkSet вмикає і вимикає зелений колір 2-го світлофору. Коли відлік таймером T_SetYellow досягає заданого часу, зелений колір на 2-му світлофорі змінюється на жовтий, а на першому - жовтий колір додається до червоного.

3. Під впливом того, що змінна L_Red установилася, вмикається в роботу таймер T_SetColor. За 3с скинуться котушки Yellow, Yellow2, Red і L_Red, а установляться Green, Blink_Green і Red2, тобто на першому світлофорі червоний і жовтий кольори зміняться на зелений, а на другому замість жовтого з'явиться червоний.

4. Завдяки тому, що змінна Blink_Green знову встановилася, починають повторну роботу таймери T_SetYellow і TON_DelayGreen. Проте в цьому випадку після затримки часу спрацьовують ті LD-мережі, які тричі вмикають і вимикають зелений колір 1-го світлофору, а коли відлік таймером T_SetYellow досягає заданого часу, змінюють зелений колір першого світлофора на жовтий, а на другому - до червоного кольору додають жовтий.

5. Оскільки по завершенню роботи таймера T_SetYellow установлюється котушка L_Green, то одразу починає працювати таймер T_SetColor. Після відліку установленого часу, завдяки роботі відповідних LD-мереж на другому світлофорі червоний і жовтий кольори зміняться на зелений, а на першому замість жовтого з'явиться червоний.

6. Далі робота програми повертається на п.2.

4.4. **Керування роботою грейферного крана**

Грейферний кран в автоматичному режимі при натисненні кнопки «Пуск» з вихідної позиції у місці розвантаження рухається до місця завантаження. При досягнені його грейфер опускається, захоплює сипкий матеріал, піднімає його уверх і рухається до місця розвантаження. Там він опускається і після розвантаження знову піднімається у вихідний стан.

Повторне натиснення кнопки «Пуск» забезпечує новий цикл роботи крана.

При створені програми використовуються такі змінні:

Pusk- кнопка «Пуск» грейфера;

P_Konceviy- кінцевий вимикач у місці розвантаження грейфера;

L_Konceviy- кінцевий вимикач у місці завантаження грейфера;

V_Konceviy- кінцевий вимикач у місці верхнього положення грейфера;

Automat- автоматичний режим роботи грейферного крана;

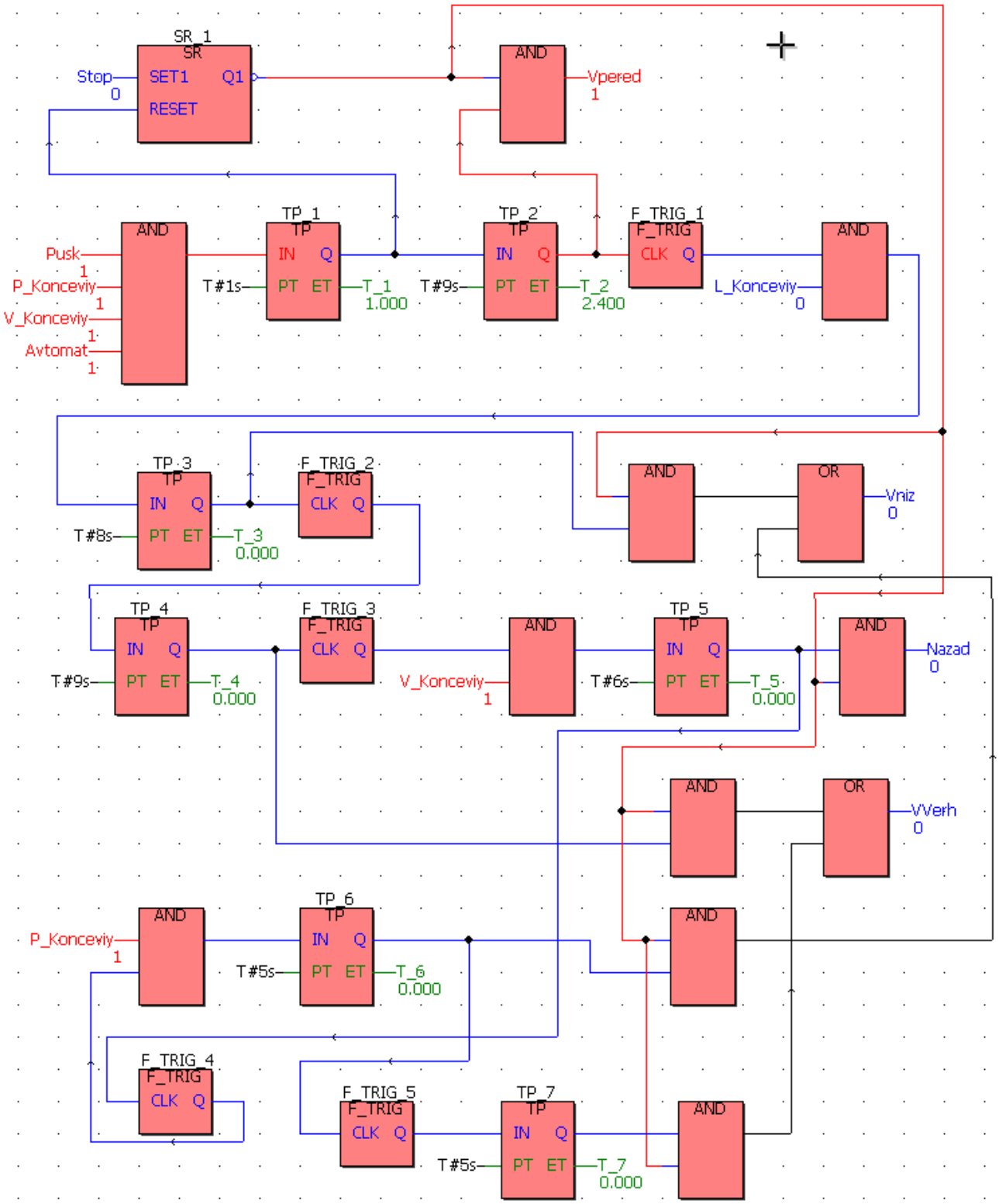
Vpered- рух грейфера до місця завантаження;

Nazad- рух грейфера до місця розвантаження;

Vverh- рух грейфера уверх;

Vniz- рух грейфера униз;
STOP- зупинка роботи грейфера.

FBD-програма керування роботою грейферного крана має вигляд:



Програма працює наступним чином.

У вихідному стані грейферний кран знаходиться у місці розвантаження, кінцеві вимикачі P_Konceviy, L_Konceviy і V_Konceviy – замкнені. При натисненні кнопки «Пуск» активізується сигнал Pusk і вихід блока AND набуває істинного значення. Генератор імпульсу TP_1 скидає тригер SR_1 і запускає таймер TP_2. Істинне значення сигналів на виході SR_1 і TP_2 забезпечує появу сигналу Vpered, який вмикає в роботу грейфер і той рухається до місця завантаження. За 9с детектор заднього фронту вхідного сигналу F_TRIG_1 генерує імпульс на логічний блок AND. За умови істинного значення змінної L_Konceviy, тобто досягнення грейфером місця завантаження, цей блок на виході має істинне значення. Запускається таймер TP_3, який формує сигнал керування Vniz, і грейфер опускається для завантаження. За 8с, коли сплине час роботи таймера TP_3, запускається таймер TP_4 і на виході ланцюжка складеного з блоків AND і OR з'являється сигнал Vverh. Грейфер з вантажем піднімається вгору. За 9с TP_4 вмикає детектор заднього фронту вхідного сигналу F_TRIG_3, який при істинності вхідної змінної V_Konceviy на блоці AND, запускає таймер TP_5. Таймер генерує імпульс і на виході блока AND набуває істинне значення змінна Nazad, яка забезпечує рух грейфера до місця розвантаження. За 6с детектор заднього фронту вхідного сигналу F_TRIG_4 подає імпульс на логічний елемент AND, який при наявності замкненого кінцевого вимикача в місці розвантаження P_Konceviy видає істинний сигнал. Цей сигнал запускає таймер TP_6, який через ланцюжок складений з блоків AND і OR, формує сигнал Vniz, і грейфер опускається для розвантаження. Коли час роботи TP_6 сплине, F_TRIG_5 запускає таймер TP_7, який забезпечує появу сигналу Vverh на виході блока OR. Грейфер піднімається і за 5с він займає вихідне положення.

Робота грейфера може бути вимкнена у будь-який час, для чого необхідно натиснути кнопку «Стоп». Як наслідок, активізується змінна STOP і спрацьовує перемикач із домінантою на вмикання SR_1. Але завдяки інверсії його вихідного сигналу перший вхід логічних елементів, що формують сигнали керування Vpered, Nazad, Vverh, Vniz, стає хибним і рух грейфера відновити не можливо. Програма забезпечує керування роботою крана після установлення грейфера у вихідний стан.

4.5. **Керування завантаженням приймальних бункерів**

На ділянці приймальних бункерів спікального відділення аглофабрики виконується розподіл потоку шихти між трьома паралельно розташованими бункерами. Як розподільний пристрій використовується автостела. Під кожним бункером є віброживильник, який безперервно видає шихту у подальшому технологічному напрямі.

Алгоритм завантаження бункерів наступний: спочатку передбачається, що всі 3 бункери порожні і автостела знаходиться над першим бункером.

Оператор перемиканням ключа вибирає режим управління - “ручний” або “автоматичний”.

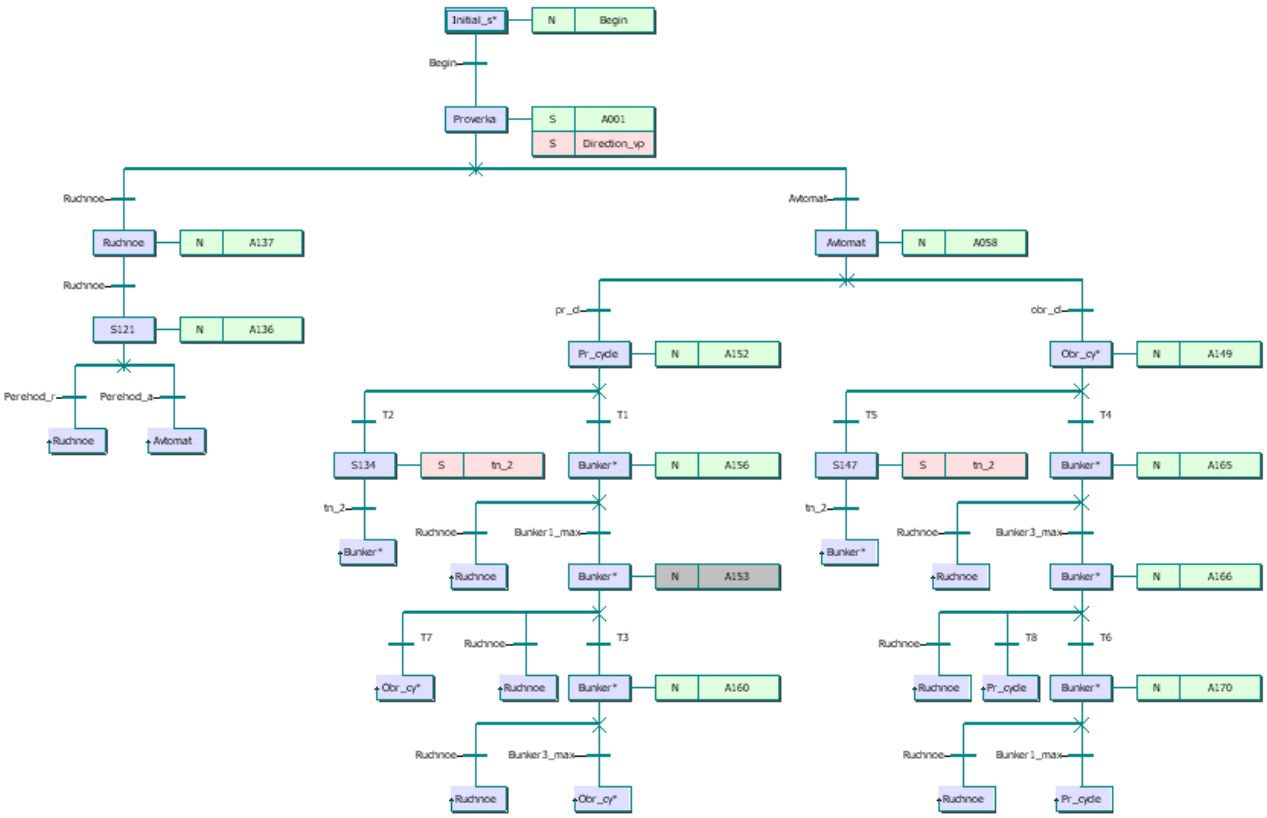
У автоматичному режимі завантаження бункерів відбувається за циклічною схемою, коли автостела рухається уперед (прямий цикл завантаження) і послідовно заповнює усі бункери до спрацьовування датчиків верхнього рівня. Після цього цикл завантаження повторюється у зворотному напрямі (зворотний цикл). Переміщення автостели від одного бункера до іншого здійснюється за часом.

У режимі ручного керування оператор має нагоду завантажити будь-який бункер натисненням відповідної кнопки. Якщо автостела при переході з автоматичного на ручне керування, рухалася від одного бункера до іншого, то вона зупиниться і відновить рух, коли оператор натисне кнопку завантаження потрібного бункера.

При переході з ручного на автоматичне керування завантаження продовжується у тому ж циклі, при якому воно було перерване при переході на ручний режим.

Якщо відбулася аварійна зупинка конвеєра подачі шихти до бункерів, то рух автостели у будь-якому напрямі блокується.

Загальна структура алгоритму управління автостелою зображена у вигляді SFC- програми[14], яка реалізує послідовне виконання усіх етапів її роботи - прямий і зворотний цикли завантаження, а також режим ручного керування:

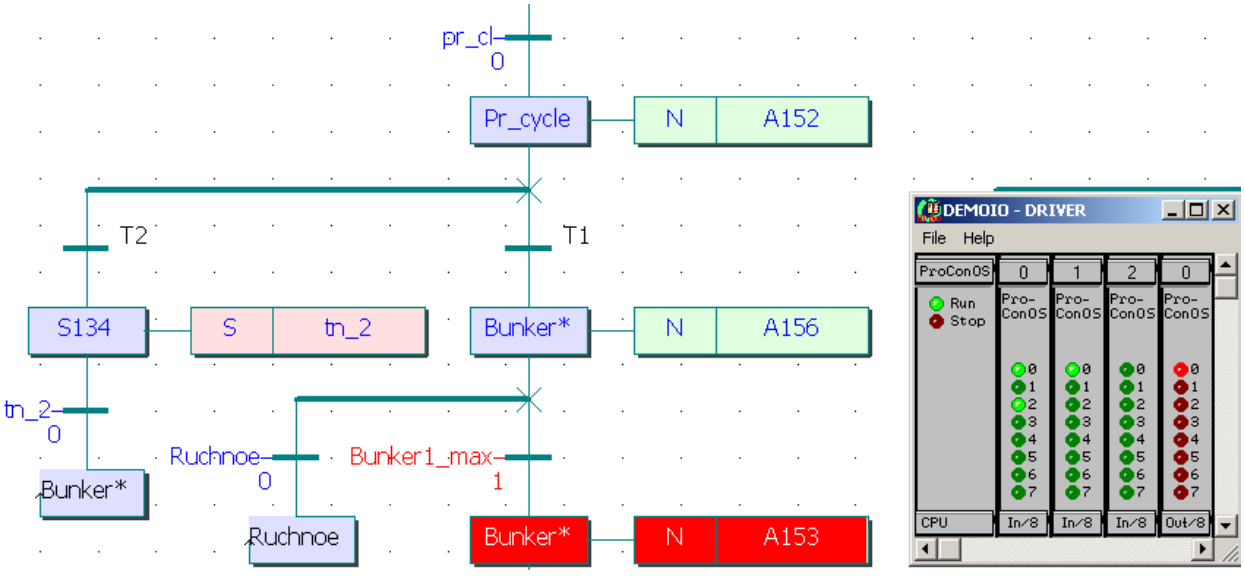


SFC-схема є комбінацією кроків і переходів, що поєднуються у певній послідовності. Альтернативні гілки, що йдуть після кроку Proverka, відповідають за ручний режим роботи автостели і автоматичний. Умова переходу визначається станом змінних Ruchnoe і Avtomat. У автоматичному режимі, залежно від стану змінних pr_cl і obr_cl, здійснюється перехід на одну з альтернативних гілок, що відповідають за прямий або зворотний цикл завантаження. У цих гілках SFC-схеми шляхом послідовного виконання кроків і переходів, реалізується алгоритм прямого або зворотного циклу завантаження. При виконанні останньої умови переходу стає активним крок, що здійснює “стрибок” на гілку наступного циклу завантаження. Проміжні альтернативні гілки перевіряють умову переходу на ручний режим, який є пріоритетним по відношенню до інших умов переходу.

Розглянемо роботу програми.

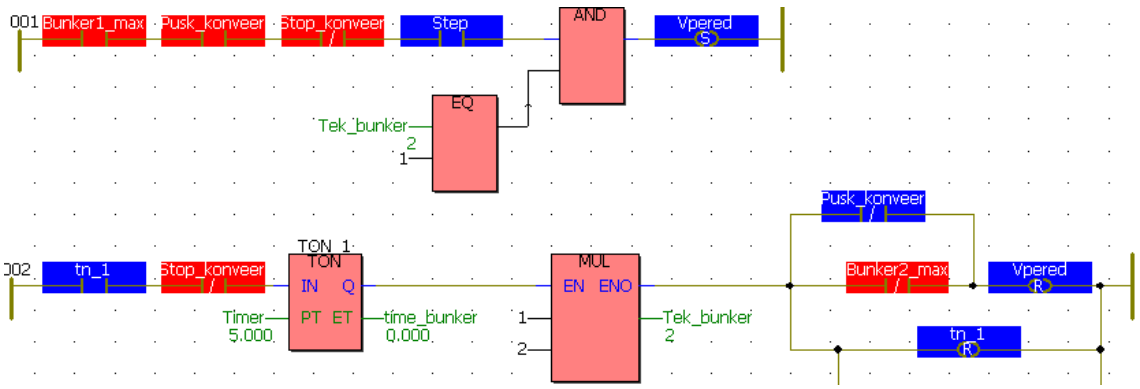
Вихідний стан - усі бункери порожні і автостела знаходиться над першим бункером.

При подачі сигналу про увімкнення конвеєра (0-ий вхід 0-го модуля дискретного вводу) і переводу оператором ключа управління в положення “автоматичне” (2-ій вхід 0-го модуля) починається прямий цикл завантаження. Заповнюється 1-ий бункер і після досягнення максимального рівня шихти, що фіксує датчик верхнього рівня, (0-ий вхід 1-го модуля дискретного вводу), автостела починає рух уперед до 2-го бункера, про що свідчить червоний колір 0-го виходу 0-го модуля дискретного виводу:

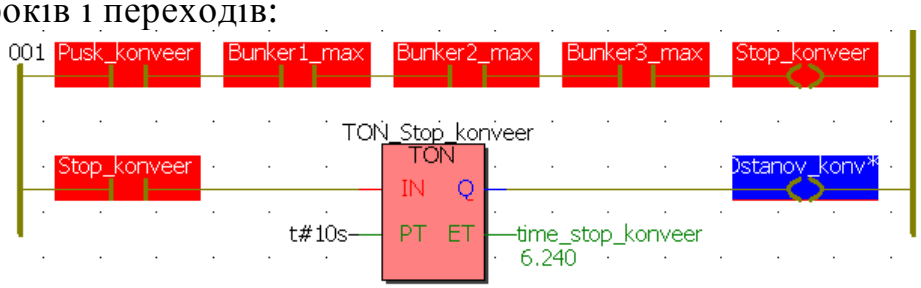


Для зручності уявлення припустимо, що час руху між бункерами дорівнює 5-ти секундам. Після закінчення цього часу автостела зупиниться над другим бункером і почне його завантаження.

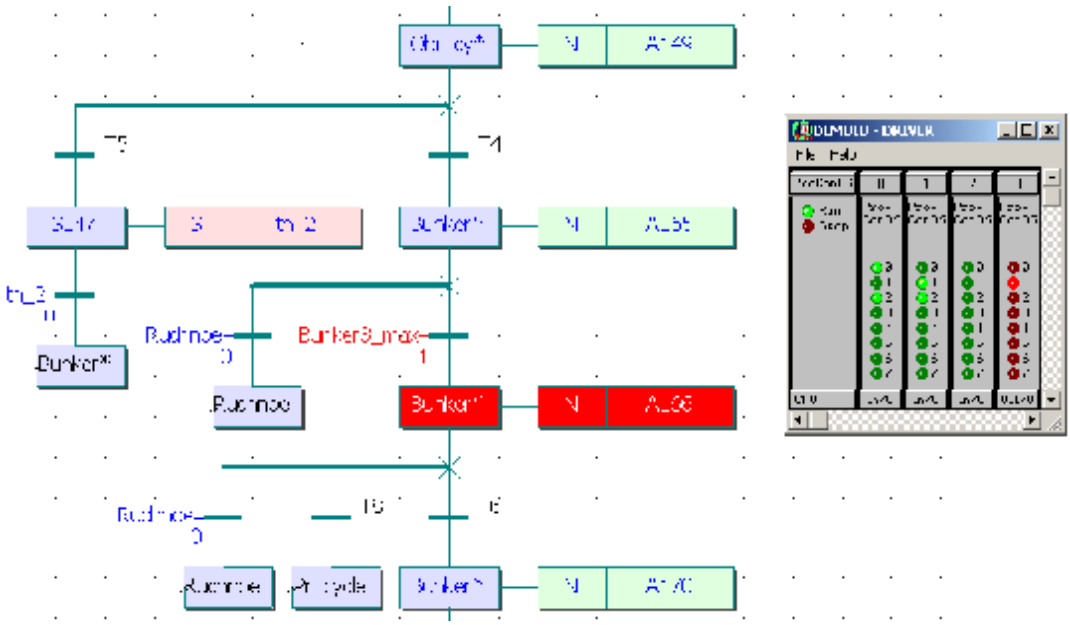
Дія A153 відповідає за переміщення автостели від 1-го бункера до 2-го, написана LD- і FBD-мовами. Фрагмент її має вигляд:



Після спрацьовування датчика верхнього рівня 2-го бункера (1-ий вхід 1-го модуля вводу) автостела рухається до 3-го бункера і починає його заповнювати. У разі, якщо пуск віброживильників бункерів не відбувся і усі бункери заповнені, щоб уникнути переповнення 3-го бункера, через інтервал часу 10с у шихтове відділення подається сигнал про виключення конвеєра подачі шихти на ділянку. (2-ий вихід 0-го модуля дискретного виводу). Цю перевірку виконує дія A001, яка є активною і незалежна від подальших кроків і переходів:

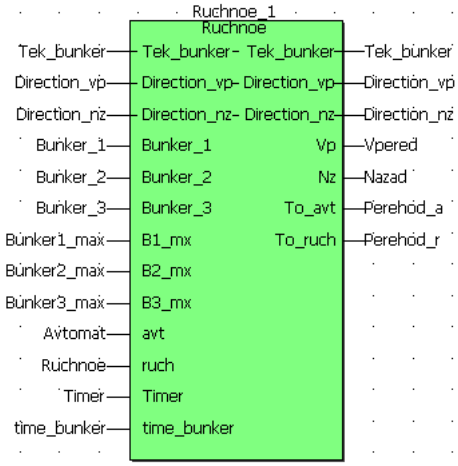


Якщо рівень у бункерах убуває, то починається цикл завантаження, при якому автостела рухається у зворотному напрямі і по черзі завантажує незаповнені бункери до максимального рівня. У наведеному нижче фрагменті програми, автостела рухається назад (1-ий вихід 0-го модуля дискретного виводу) від 3-го бункера до 1-го, не зупиняючись над заповненим 2-им. Час пересування дорівнює 10 с.

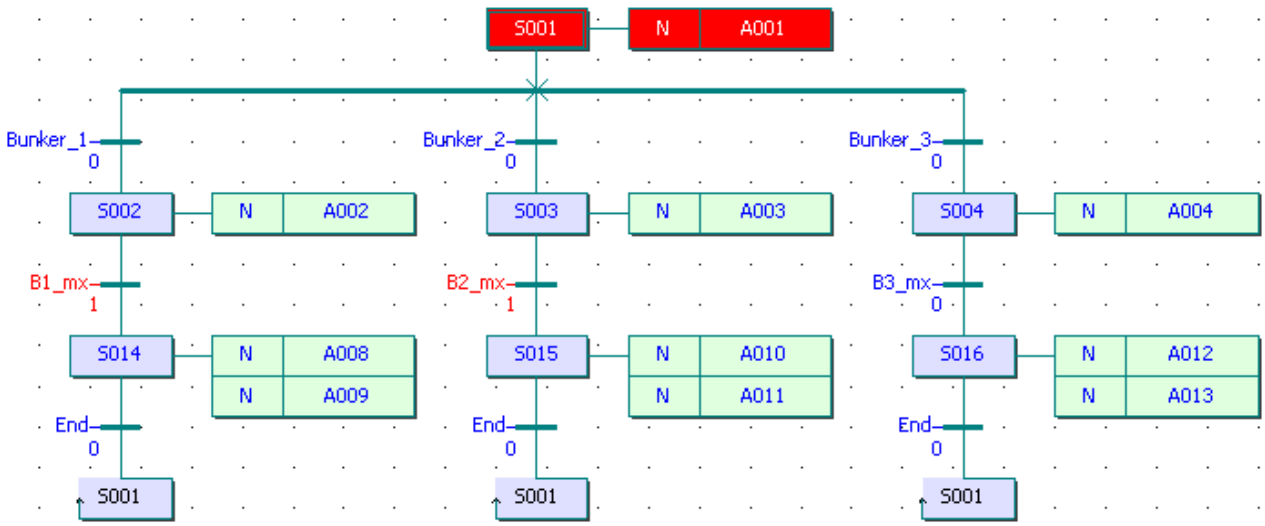


Після завантаження 1-го бункера цикл роботи змінюється, і автостела рухається уперед до найближчого незаповненого бункера. За змістом дії прямого і зворотного циклу аналогічні одна одній, відмінність полягає тільки в напрямі руху автостели.

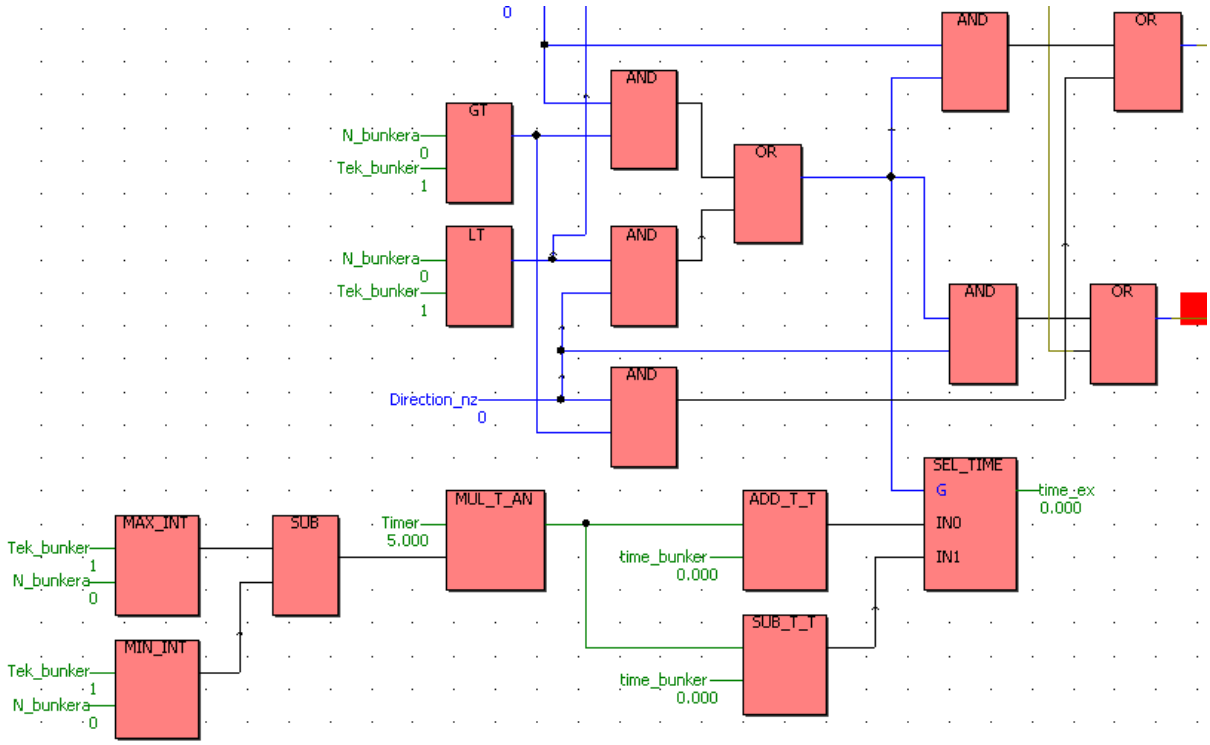
При переході у режим ручного керування, циклічне завантаження бункерів припиняється, і в програмі відбувається перехід до дії, яка відповідає за цей режим роботи. Дія містить функціональний блок, який визначає номер поточного бункера, що завантажується, номер бункера визначеного оператором для завантаження, напрям і час руху автостели до нього:



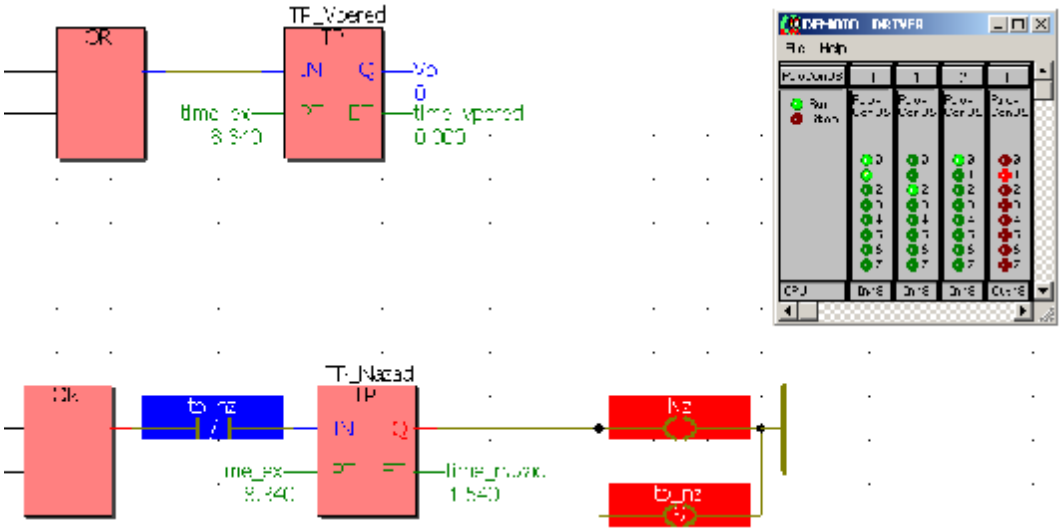
Якщо автостела, при переході з автоматичного на ручне керування, рухалася від одного бункера до іншого, то вона зупиняється. У цьому випадку час її руху до необхідного бункера у ручному режимі визначається, виходячи з поточного напряму руху автостели, часу, що пройшов з моменту початку її руху і до переходу у ручний режим, а також напряму, в якому автостела має рухатися при натисненні кнопки оператором. Загальний алгоритм функціонального блока ручного управління написаний SFC-мовою:



Основні дії – FBD-мовою, із застосуванням блоків порівняння, логічних операцій «І», «АБО», блоків арифметичних операцій, таймерів і блоку вибору SELECT. Фрагмент програми дії A002 має вигляд:

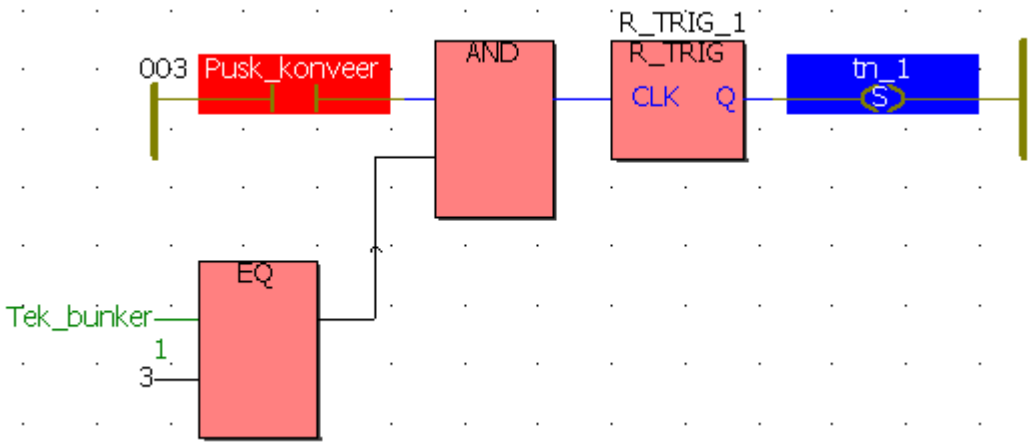


Промоделюємо ситуацію, коли під час пересування від 3-го бункера до 2-го, оператору знадобилося завантажити 1-ий бункер. При переході з автоматичного режиму в ручний (1-ий вхід 0-го модуля вводу), автостела зупинилася. Коли оператор натиснув кнопку завантаження 1-го бункера (0-ий вхід 2-го модуля вводу), автостела почала рухатися уперед. Час необхідний для пересування автостели визначається, як час руху її від 3-го бункера до 1-го (10 сек.) мінус час, який автостела вже пройшла від 3-го бункера до 2-го (1.66 сек). Тобто загальний час пересування автостели до 1-го бункера після її зупинки дорівнює 8.34 с:



Після закінчення завантаження 1-го бункера і переходу назад в автоматичний режим, автостела повернеться у циклічний режим завантаження і почне рух уперед до найближчого незаповненого бункера, тобто до 2-го.

Якщо під час пересування автостели між бункерами відбулася раптова зупинка конвеєра подачі шихти, то автостела доїде до найближчого бункера і цикл завантаження припиниться до появи сигналу про включення конвеєра. Фрагмент програми, що відповідає за дану операцію має вигляд:



З наведеного фрагменту видно, що змінна «Pusk_konveer» - у червоному кольорі, тобто знаходиться у стані TRUE (кнопка пуску конвеєра розімкнена). Як наслідок - транспортер не працює (змінна «tn_1»- хибна).

4.6. **Керування рівнем рідини в резервуарі**

Керування рівнем в резервуарі, що заповнюється рідиною, здійснюється в автоматичному та ручному режимах. При роботі в автоматичному режимі відбувається двопозиційне регулювання витратою рідини з резервуару, тобто після досягнення рівнем установленого мінімального або максимального значення автоматично закривається чи відкривається регулювальний орган на виході з резервуара.

В ручному режимі керування рівнем в резервуарі забезпечується шляхом впливу на регулювальний орган витрати рідини за допомогою певних кнопок.

В програмі використані такі змінні:

Autofill – автоматичне збільшення рівня;
Autodrain – автоматичне зменшення рівня рідини;
Level – поточне значення рівня рідини в резервуарі;
Fill – кнопка ручного збільшення рівня;
Drain – кнопка ручного зменшення рівня;
Mode – кнопка вибору режиму керування.

ST-програма керування рівнем рідини в резервуарі має вигляд:

```
 (*Програмування автоматичного збільшення рівня рідини
 в резервуарі*)
if autofill then
level:=level+10;
end_if;
(*Програмування автоматичного зменшення рівня рідини
в резервуарі*)
if autodrain then
level:=level-10;
end_if;
case level of
0:autofill:=true;
  autodrain:=false;
100:autofill:=false;
   autodrain:=true;
40:autofill:=true;
   autodrain:=false;
end_case;
(*Програмування ручного збільшення рівня рідини
в резервуарі*)
if fill then
autodrain:=false;
autofill:=true;
level:=level+10;
else
if level>100 then
autofill:=false;
autodrain:=true;
end_if;
case level of
0:autofill:=true;
  autodrain:=false;
100:autofill:=false;
   autodrain:=true;
40:autofill:=true;
   autodrain:=false;
end_case;
end_if;
```

```
(*Програмування ручного зменшення рівня в резервуарі*)
if drain then
autofill:=false;
level:=level-10;
if level<0 then
level:=0;
return;
end_if;
else
 if level<0 then
level:=0;
return;
end_if;
case level of
0..39:autofill:=true;
  autodrain:=false;
40:autofill:=true;
   autodrain:=false;
end_case;
end_if;
(*Програмування переходу між режимами управління*)
if mode then
autofill:=false;
autodrain:=false;
else
case level of
0..39:autofill:=true;
      autodrain:=false;
100:autofill:=false;
   autodrain:=true;
40:autofill:=true;
   autodrain:=false;
end_case;
end_if;
```

Програма працює наступним чином.

В автоматичному режимі, коли рівень рідини досягає максимального установленого значення - 100, відкривається регулювальний орган на виході з резервуару і рівень рідини починає зменшуватися. Після досягнення мінімального установленого значення – 40, регулювальний орган закривається і рівень рідини знову починає збільшуватися. Досягнувши позначки 100, процес управління повторюється.

В ручному режимі за допомогою кнопки Fill резервуар заповнюється, а за допомогою кнопки Drain, здійснюється злив рідини.

Кнопка Stop використовується для зміни режиму роботи. Перехід в автоматичний режим управління здійснюється після вимкнення кнопок ручного керування.

Завдання для самостійної роботи

1. Запрограмуйте FBD-мовою алгоритм керування компресором відповідно до наведеної на рис. 4.1 релейно-контактної схеми:

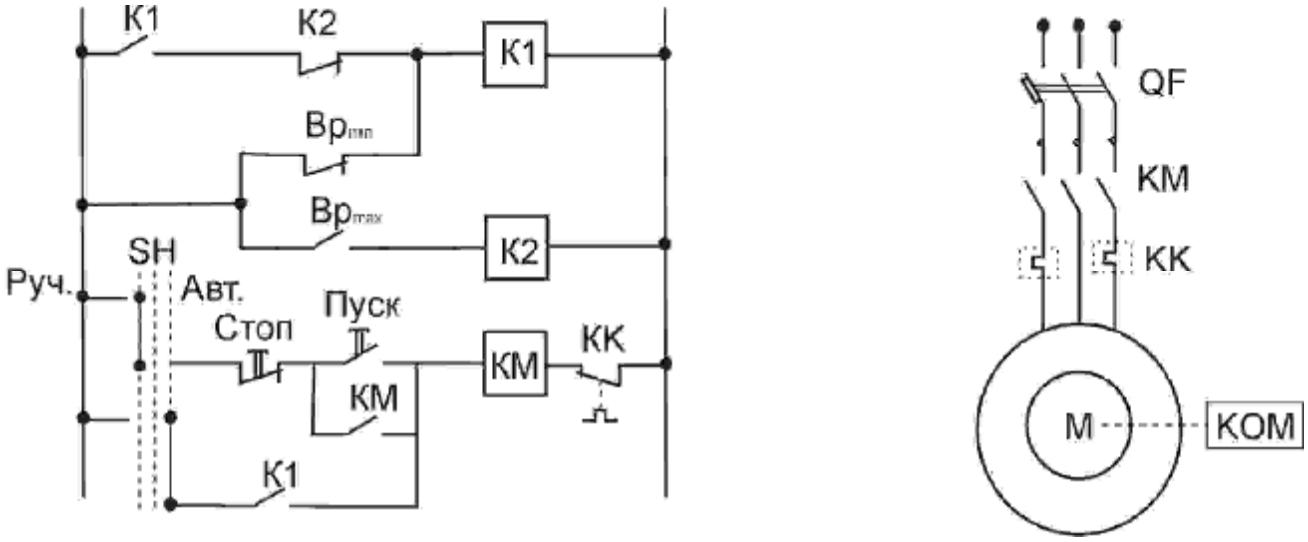


Рис. 4.1 Схема автоматичного керування компресором.

Датчиком тиску є електроконтактний манометр ВР. Компресор вмикається кнопкою "Пуск" при ручному керуванні (перемикач SH у лівому положенні) або замиканням контакту реле К1 у колі магнітного пускача КМ при автоматичному керуванні (перемикач SH повернутий праворуч). Якщо тиск у системі падає до встановленого на $ВР_{min}$ мінімального значення, то подається сигнал на вмикання компресора. При роботі компресора тиск у системі починає підвищуватися, що призведе до розмикання контактів $ВР_{min}$. Проте завдяки самоблокуванню реле К1 не вимкнеться і компресор буде продовжувати працювати. Якщо тиск у системі перевищує встановлене на $ВР_{max}$ максимальне значення, то компресор вимикається. Внаслідок його періодичного вмикання та вимикання забезпечується двопозиційне регулювання тиску у межах від мінімального до максимального заданого значення. Теплове реле КК і роз'єднувачі автоматів QF призначені для вимикання електродвигуна при перевантаженні та коротких замиканнях.

2. Запрограмуйте SFC-мовою керування роботою вантажного ліфта, який обслуговує 10-ти поверховий будинок. Для управління ліфтом на кожному з поверхів є дві кнопки керування - для подачі команди на підйом і для подачі

команди на спуск. Для індикації положення ліфтової кабіни встановлені кінцеві вимикачі. При досягненні ліфтом 10-го поверху рух угору неможливий. Аналогічно, при досягненні ліфтом 1-го поверху рух униз неможливий. Пересування ліфтової кабіни з одного поверху на іншій займає 5 секунд.

3. Розробіть ST-програму, яка забезпечує неперервне послідовне вмикання чотирьох двигунів і здійснює зупинку на 5с після п’яти циклів роботи.

4. Реалізуйте SFC-мовою алгоритм керування роботою світлофора на перехресті доріг.

5. Запрограмуйте LD- мовою процес регулювання температури рідини у технологічному апараті, алгоритм якого надається релейно-контактною схемою:

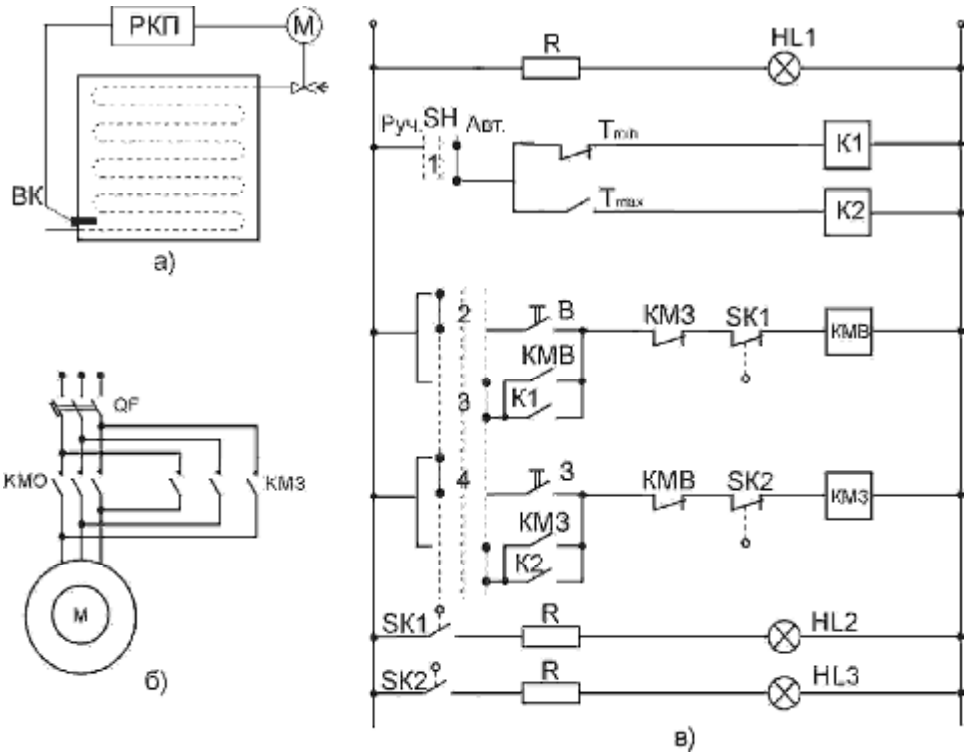


Рис. 4.2. Схема автоматичного регулювання температури:

а) - функціональна; б) – силових кіл виконавчого механізму М;
в) – релейно-контактна схема керування двигуном

Нагрівання відбувається парою або гарячою водою. Датчиком температури ВК є електроконтактний манометричний термометр (ЕКТ), виконавчим механізмом - заслінка з електроприводом. Керування приводом здійснюється релейно-контактним пристроєм РКП. Контакти T_{min} і T_{max} ЕКТ малопотужні, тому в схемі передбачені проміжні реле К1 і К2, що використовуються як підсилювальні елементи. Універсальний перемикач SH

перемикає кола керування з ручного режиму на автоматичний. Кінцеві вимикачі SK1 і SK2 вимикають контактори КМЗ і КМВ при повному закритті або відкритті заслінки. Їх замикальні контакти використовуються для сигналізації положення заслінки. У схемі передбачено електричне блокування, яке виключає одночасне вмикання контакторів КМВ і КМЗ (послідовно з КМВ увімкнений робочий контакт КМЗ, а послідовно з КМЗ - робочий контакт КМВ).

В автоматичному режимі перемикач SH знаходиться у положенні "Авт." Якщо температура рідини в апараті дорівнює або менше за нижню межу, то контакти ЕКТ замкнені. При вмиканні схеми спрацьовує реле К1, яке через контакти 3 перемикача SH, контакт КМЗ і кінцевий вимикач SK1 подає сигнал на вмикання КМВ. Вал виконавчого механізму М при спрацьовуванні КМВ починає обертатися у бік відкриття заслінки, забезпечуючи приплив теплоносія в апарат. При повному відкритті заслінки кінцевий вимикач SK1 вимикає виконавчий механізм М. При нагріванні рідини спочатку розмикається контакт T_{min}, а потім при досягненні температурою верхньої встановленої межі замикається контакт T_{max}, який подає керувальний сигнал на закриття заслінки.

Таким чином, розглянута релейна схема здійснює регулювання температури на двох рівнях - максимальному та мінімальному.

6. Складіть програми IL- і ST- мовами, які б реалізовували алгоритм управління рухом горизонтального крана.

Протягом переміщення кран не сприймає команди на зміну напряму руху. Щоб змінити напрям руху, кран необхідно зупинити. Одночасна подача сигналів керування на обидва виходи не допускається.

Час переміщення крана в кожному напрямі контролюється. Переміщення вліво припиняється через 5с, а переміщення вправо – через 10 с.

Якщо сумарна кількість переміщень в обох напрямах досягає 20, управління роботою крана припиняється .

7. Запрограмуйте LD-мовою алгоритм керування роботою дробарки, бункер якої заповнюється породою верхнім транспортером, а розвантаження її здійснюється на нижній транспортер. Подача породи відбувається за сигналами датчиків нижнього і верхнього рівнів в бункері дробарки. Для уникнення небажаних наслідків, вмикання дробарки в роботу здійснюється дворазовим натисненням кнопки «Пуск». При цьому на 10с спрацьовує сирена, яка за 30с після вимкнення знову вмикається на 10с і тільки після цього починає працювати грабарка. Якщо в бункері є залишки породи, про що свідчить сигнал з датчика нижнього рівня, спрацьовує нижній конвеєр. Після спорожнювання бункера вмикається верхній конвеєр і починається позиційне керування завантаженням дробарки відповідно до сигналів датчиків рівня. Разом з цим на пульту оператора загоряється сигнальна лампа «Норма».

ЛІТЕРАТУРА

1. *Языки* программирования контроллеров по стандарту МЭК 1131.3/PL7-Junior. Schneider automation club. Информационный бюллетень представительства Schneider Electric на Украине. 1997,№1, с.4-5.
2. *Д. Христенсен*. Знакомство со стандартом на языки программирования PLC: IEC 1131-3. Мир компьютерной автоматизации, 1995, №1, с. 3-5.
3. *Ельперін І.В*. Промислові контролери: Навч. посіб.-К: НУХТ,2003.-320с.
4. *Петров И.В*. Программируемые контроллеры. Стандартные языки и приёмы прикладного проектирования/Под ред. проф. В.П. Дьяконова.-М.:СОЛОН-Пресс, 2004.-256 с.: ил. -(Серия «Библиотека инженера»).
5. *Зюбин В.Е*.Программирование ПЛК: языки МЭК 61131-3 и возможные альтернативы. Промышленные АСУ и контроллеры. 2005, №11, с. 31-35.
6. ***KW MULTIPROG***. User Manual Klopper und Witgt Software GmbH.-1998.
7. *Руководство* пользователя ISa GRAF PRO.
8. *Деменков Н.П.* Языки программирования промышленных контроллеров: Учебное пособие/Под ред.К.А. Пупкова. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. – 172 с.: ил.
9. *Руководство* пользователя по программированию ПЛК в CoDeSys 2.3.-М.: СОЛОН-Пресс 2004.-253с.