Міністерство освіти і науки України
Донбаська державна машинобудівна академія (ДДМА)

О. О. Сердюк, О. В. Разживін

РОЗПОДІЛЕНІ СИСТЕМ НА БАЗІ ПЛК

Краматорськ
ДДМА
2020

ББК 32.965.7
УДК 658.516
С–32

Сердюк О. О.
С 32 Розподілені систем на базі ПЛК: навч. посіб. / – О. О. Сердюк, О. В. Разживін. – Краматорськ : ДДМА, 2020. – 210 с.
ISBN

Розглянуто апаратні засоби систем автоматизації SIMATIC. Викладено принципи і методики конфігурування станцій SIMATIC S7-300/400 та розподіленої периферії у програмній системі STEP 7. Висвітлено особливості структурної організації програм у системах автоматизації SIMATIC. Наведено методики створення користувальницької програми у середовищі STEP 7 на мовах програмування STL, SCL, LAD, FBD, Graph і Higraph.

Для студентів спеціальності 123 – «Комп'ютерна інженерія»».

ББК 32.965.7
УДК 658.516

ISBN

ЗМІСТ

ВСТУП

Згідно з освітньо-кваліфікаційною характеристикою фахівця напряму 6.050202 «Автоматизація та комп‘ютерно-інтегроване управління» при виконанні професійних задач він повинен розробляти технічні та програмні засоби систем автоматизації.

З цією метою освітньо-професійною програмою підготовки фахівця передбачено навчальну дисципліну «Проектування систем автоматизації» об‘ємом 8 кредитів, яка належить до нормативної частини програми.

Мета дисципліни «Проектування систем автоматизації» – освоєння сучасних принципів і правил розроблення проектної документації стосовно систем автоматизації технологічних процесів. Вивчення дисципліни забезпечує вміння фахівця виконувати одну з основних виробничих задач його діяльності – проектування апаратних і програмних засобів систем автоматизації.

Для вивчення дисципліни «Проектування систем автоматизації» необхідно засвоєння таких дисциплін:

«Електроніка та мікропроцесорна техніка»;

«Теорія автоматичного управління»;

«Автоматизація технологічних процесів і виробництв»;

«Технічні засоби автоматизації»;

«Контролери та їх програмне забезпечення».

Завдання дисципліни полягають у тому, що на основі вимог освітньо-кваліфікаційної характеристики й освітньо-професійної програми підготовки бакалавра у результаті вивчення дисципліни студент повинен вміти:

- обґрунтовано вибирати технічні засоби автоматизації, безпеки та діагностики;
- виконувати конфігурування центральних станцій і розподіленої периферії систем автоматизації;
- програмувати логічні контролери для управління обладнанням;
- застосовувати ПЕОМ для розроблення проектної документації систем автоматизації.

Слід зазначити, що український ринок насичений великою кількістю технічних засобів автоматизації різних виробників. Для спрощення та прискорення монтажних робіт апарати та пристрої мають зазвичай модульну конструкцію та визначені функції. Для задоволення широкого спектру вимог користувачів виробники збільшують номенклатуру своїх виробів. Сьогодні розроблення якісного проекту системи автоматизації без застосування спеціальних програмних інструментів викликає складнощі через великий обсяг рутинної роботи зі знаходження раціонального рішення, а також спричиняє виникнення небезпеки прийняття некоректного рішення. Тому для підтримки своєї продукції на ринку провідні компанії розробляють необхідні програмні засоби для прискорення та спрощення процесу проектування систем автоматизації.

Враховуючи те, що у галузях машинобудування та металургії під час вибору апаратних засобів автоматизації головним критерієм є надійність апаратури: найбільшого розповсюдження в цих галузях набувають системи автоматизації SIMATIC всесвітньо відомого концерну SIEMENS. Слід також зазначити, що для підтримки вирішення завдань проектування систем автоматизації SIMATIC у концерні SIEMENS створено програмне забезпечення STEP 7.

На сайтах представництв цього концерну в Україні та Росії є величезна кількість інформаційних матеріалів, але їх надлишок стає перешкодою для використання студентами із-за часових обмежень на вивчення дисципліни. При цьому практично відсутні методичні матеріали українською мовою.

Мета навчального посібника – у стислому виді подати студентам методичні рекомендації щодо вирішення найважливіших питань у проектуванні апаратного та програмного забезпечення системи автоматизації SIMATIC. У результаті освоєння цього матеріалу студенти отримають знання про сучасні принципи побудови систем автоматизації та набудуть умінь створювати проекти систем автоматизації з використанням засобів будь-яких сучасних виробників. Набуті знання та уміння будуть використані у процесі розроблення дипломних проектів.

Для самостійної роботи студента будуть корисні методичні рекомендації та приклади із застосування програми STEP 7 для конфігурування центральних станцій і розподіленої периферії систем автоматизації SIMATIC, а також для розроблення користувальницьких програм на мовах STL, SCL, LAD, FBD, S7-Graph і S7-Higraph, що відповідають міжнародному стандарту IEC 6-1131-3.

1 ПРОЕКТУВАННЯ ЦЕНТРАЛЬНИХ СТАНЦІЙ СИСТЕМ АВТОМАТИЗАЦІЇ SIMATIC

1.1 Ієрархічна організація проекту системи автоматизації SIMATIC

SIMATIC – це торговельна марка концерну Siemens AG, що відома своїми засобами автоматизації технологічних процесів, виробництв і підприємств: Назва SIMATIC поєднує «Siemens» і «Automatic» в одному слові.

До систем автоматизації SIMATIC належать:

- лінійки програмувальних логічних контролерів (ПЛК) – SIMATIC S5, SIMATIC S7;
- мережні розв'язки SIMATIC NET на основі промислових мереж Profinet, Industrial Ethernet, PROFIBUS, As-interface;
- засоби людино-машинного інтерфейсу SIMATIC HMI та HMI-додатки (SIMATIC Protool; SIMATIC WinCC);
- DCS-система SIMATIC PCS 7;
- промислові ПК (SIMATIC IPC);
- програмна платформа для розроблення MES (системи оперативного керування виробництвом) – SIMATIC IT.

Першою вільно програмувальною системою автоматизації стала система SIMATIC S5 (1978–1979 рр.). В 2005 р. вона була замінена на систему SIMATIC S7, яка розвивається й підтримується дотепер.

До програмувальних логічних контролерів SIMATIC S7 належать контролери S7-200, S7-1200, S7-300 та S7-400. Контролери S7-200 та S7-1200 призначені для рішення відносно простих завдань нижнього рівня управління, а контролери S7-300 та S7-400 можуть застосовуватися для вирішення будь-яких завдань автоматизації виробництва.

Для програмування контролерів Siemens розробив власне ПО:

- для ПЛК SIMATIC S7-200 – ПО SIMATIC STEP 7 Micro/Win;
- для ПЛК SIMATIC S7-1200 – ПО SIMATIC STEP 7 Basic;
- для ПЛК SIMATIC S7-300 і SIMATIC S7-400 – ПО SIMATIC STEP 7.

SIMATIC STEP 7 – це найбільш потужна на теперішній час система програмування контролерів. За допомогою цієї програми виконується комплекс робіт зі створення й обслуговування систем автоматизації на основі ПЛК S7-300 і S7-400.

В основі STEP 7 є концепція проекту, під яким розуміється комплексне розв'язання завдання автоматизації. Роботу із проектом у цілому забезпечує головна утиліта STEP 7 – SIMATIC Manager.

Проект являє собою сукупність апаратних і програмних ресурсів, задіяних у вирішенні завдання автоматизації. Ці засоби відображаються у SIMATIC Manager у вигляді деревоподібної структури, найбільш важливі компоненти якої показані на рисунку 1.1.

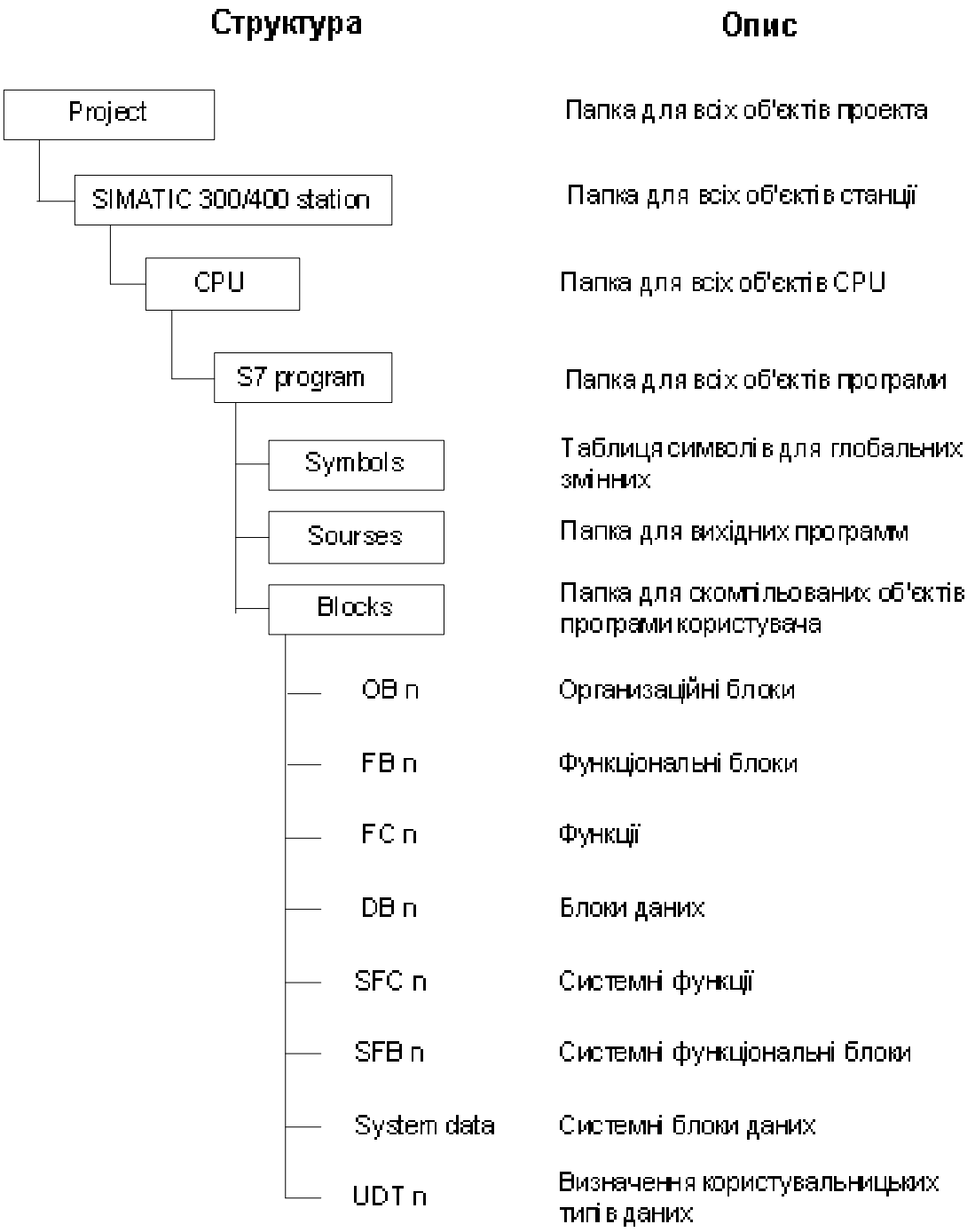


Рисунок 1.1 – Ієрархічне дерево об'єктів проекту

У верхній частині дерева розташовані об'єкти апаратури – інтерфейси, станції та процесорні модулі. Об'єкти, які виділені жирним шрифтом, містять інші об'єкти. Всі показані на рисунку об'єкти доступні користувачеві.

Програма користувача (S7-програма) завантажується у процесорний модуль. У папці S7-програми розташовані її компоненти – таблиця символів глобальних змінних (Symbols), за допомогою якої формальні параметри зв‘язуються з фактичними, а також папки з файлами програми (Sourses) та блоками (Blocks).

У зв‘язку з необхідністю вирішення різноманітних завдань у STEP 7 передбачені різні блоки – організаційні блоки OB для взаємодії з операційною системою, блоки FB та FC з програмними кодами, блоки DB для збереження фактичних значень змінних, системні блоки SFB, SFC та System data для виконання службових завдань і сервісних функцій, а також масиви користувальницьких типів даних UDT.

Головне призначення програми STEP 7 – це *програмування* контролерів. Розроблення програми здійснюється у базовому редакторі STEP 7, який забезпечує написання програм на трьох мовах міжнародного стандарту IEC 61131-3: релейно-контактної логіки LAD; функціональних блокових діаграм FBD і списку інструкцій STL.

На додаток до цих трьох основних мов існує можливість використовувати ще чотири додаткові мови, редактори до яких поставляються окремо: SCL – структурована мова керування, за синтаксисом близька до Pascal; S7-GRAPH – керування послідовними технологічними процесами; S7-Higraph – керування на основі графа станів системи та CFC – мова постійних функціональних схем.

Можливість спостереження за поточним станом програми, доступна при використанні будь-якої мови програмування, забезпечує не тільки налагодження програмного забезпечення, але й пошук несправностей в устаткуванні, що підключається, навіть якщо воно не має засобів діагностики.

STEP 7 дозволяє також робити *конфігурування* програмувальних логічних контролерів і мереж (утиліти HWconfig і NetPro). У процесі конфігурування визначається склад устаткування в цілому, здійснюється розбивання на модулі, визначаються способи підключення та використовувані мережі, задаються настроювання для використовуваних модулів. Система перевіряє правильність використання й підключення окремих компонентів. Завершується конфігурування завантаженням обраної конфігурації в устаткування, що по суті є настроюванням устаткування. Утиліти конфігурування дозволяють здійснювати діагностику встаткування, виявляти апаратні помилки або неправильний монтаж устаткування.

У сімействі програмних продуктів компанії Siemens для розв'язку завдань автоматизації система STEP 7 виконує інтеграційні функції. У проект STEP 7 можуть бути включені різноманітні технічні засоби, наприклад, системи людино-машинного інтерфейсу або персональний комп'ютер. Система дозволяє створити програмне забезпечення для настроювання й керування складними вимірювальними або виконавчими пристроями автоматизації, виконати мережні настроювання, здійснити з'єднання й передачу даних між пристроями автоматизації в розподіленій периферії.

Для освоєння всіх можливостей STEP 7 необхідно спочатку розглянути базові компоненти системи автоматизації – програмувальні логічні контролери S7-300 та S7-400.

1.2 Організація програмувальних контролерів S7-300/400

Апаратні компоненти контролера

Програмувальний контролер SIMATIC S7-300/400 має модульну конструкцію. Модулі, з яких складається необхідна конфігурація контролера, можуть бути центральними (розташовуватися по сусідству з CPU) або розподіленими.

Програмувальний контролер SIMATIC S7-300/400 містить у собі такі компоненти:

1. Стійка (Rack) для розміщення модулів і їхнього з'єднання. Стійки поділяються на три типи:

UR – універсальна;

- CR – центральна;
- ER – розширення;

2. Джерело живлення (PS – power supply), яке забезпечує робочу напругу 24 В постійного струму для живлення контролеру, датчиків і виконавчих пристроїв;

3. Центральний процесор (CPU – central processing unit), що використовується для розміщення й оброблення програми користувача, а також забезпечує зв'язок з іншими CPU і з програматором PG за допомогою шинного кабелю SINEC 1.2;

4. Інтерфейсні модулі (IM – interface module), які використовуються для з'єднання стійок;

5. Сигнальні модулі (SM – signal module), які використовуються для введення й виведення дискретних сигналів, а також для перетворення вхідних аналогових сигналів у дискретні або вихідних дискретних сигналів в аналогові сигнали керування;

6. Функціональні модулі (FM – function module), які незалежно від CPU використовуються для виконання різних завдань керування, пов'язаних з тимчасовими характеристиками процесів;

7. Комунікаційні процесори (CP – communication processor), які забезпечують зв'язок контролера з підмережами.

Організація пам'яті

Основу системи становить CPU, що містить три області пам'яті для обробки програм користувача:

- завантажувальна пам'ять використовується для програм користувача без призначень символьних адрес або коментарів (вони залишаються у пам'яті пристрою програмування). Завантажувальна пам'ять може бути типу RAM (ОЗП) або EPROM (ЕППЗП);
- робоча пам'ять (вбудований ОЗП) містить частини програми S7, які є істотними для виконання користувальницької програми. Програма виконується тільки в областях робочої й системної пам'яті;
- системна пам'ять (ОЗП) містить елементи пам'яті, надавані кожним CPU програмі користувача, такі як таблиці входів і виходів образа процесу, меркери, таймери й лічильники. Системна пам'ять містить також стек блоків і стек переривань.

Файл конфігурації та програма користувача із програматора спочатку пересилається в завантажувальну пам'ять (load memory). Операційна система CPU копіює релевантні частини програмного коду й даних у робочу пам'ять (work memory).

Завантажувальна пам'ять у CPU для S7-300 зазвичай (за винятком CPU 318) являє собою RAM-пам'ять і може вміщати програму цілком. Поточні значення з області пам'яті користувача (блоки даних) і системної пам'яті (меркери, таймери, лічильники) варто розміщувати в енергонезалежній формі, щоб в умовах можливих перебоїв електроживлення користувальницькі дані зберігалися без застосування резервної батареї.

Системна пам'ять містить адреси (змінні), за якими здійснюється обіг у програмі. Всі адреси розподіляються в адресному просторі так, що утворюють області, розміри яких залежать від конкретного CPU.

Системна пам'ять CPU містить такі адресні області:

- входи (I) – це *відображення процесу за входами* дискретних вхідних модулів;
- виходи (Q) – це *відображення процесу за виходами* дискретних вихідних модулів;
- меркери (M) – це деякі проміжні стани, інформація про які повинна бути доступною з будь-якої точки програми;
- таймери (T) зберігають інформацію, яка визначає параметри часу для функцій очікування й моніторингу;
- лічильники (C) зберігають інформацію для функції прямого та зворотного рахунку;
- тимчасові локальні дані (L) використовуються в якості динамічних проміжних буферів під час оброблення блоків. L-стек динамічно займається й звільняється CPU при виконанні програми.

У системній пам'яті зберігаються також буфери даних для комунікаційних завдань і системних повідомлень (буфери діагностики). Розміри цих буферів даних, як і розміри областей зберігання відображення процесу за входами й виходами, у нових центральних процесорах для S7-400 може визначати користувач.

1.3 Механічна конфігурація контролера (станції)

Програмувальний контролер (станція) може складатися і центральної стійки й стійок розширення. Стійки з'єднуються за допомогою інтерфейсних модулів.

Монтажні стійки для контролерів S7-400

Монтажні стійки є несучою основою, що призначена для установлення модулів, а також для їх підключення до ланцюгів живлення та до внутрішньої шини контролера.

Монтажна стійка UR (universal rack) може бути використана як базова стійка контролера S7-400, а також як стійка розширення. Вона має два

варіанти виконання – UR1 на 18 слотів (*слоти* – посадкові місця у монтажній стійці) та UR2 на 9 слотів. Стійка підтримує можливість використання стандартних або резервованих схем живлення контролера. У першому випадку в неї встановлюється один, у другому – два блоки живлення. У будь-якому разі установлення модулів блоків живлення починається з першого роз'єму монтажної стійки, тобто з першого слоту.

За модулем живлення встановлюються модуль CPU і далі інші модулі. При цьому інтерфейсні модулі (передавальні та приймальні) повинні встановлюватися в останні слоти. Принцип установлення модулів на стійку UR1 показаний на рисунку 1.2.

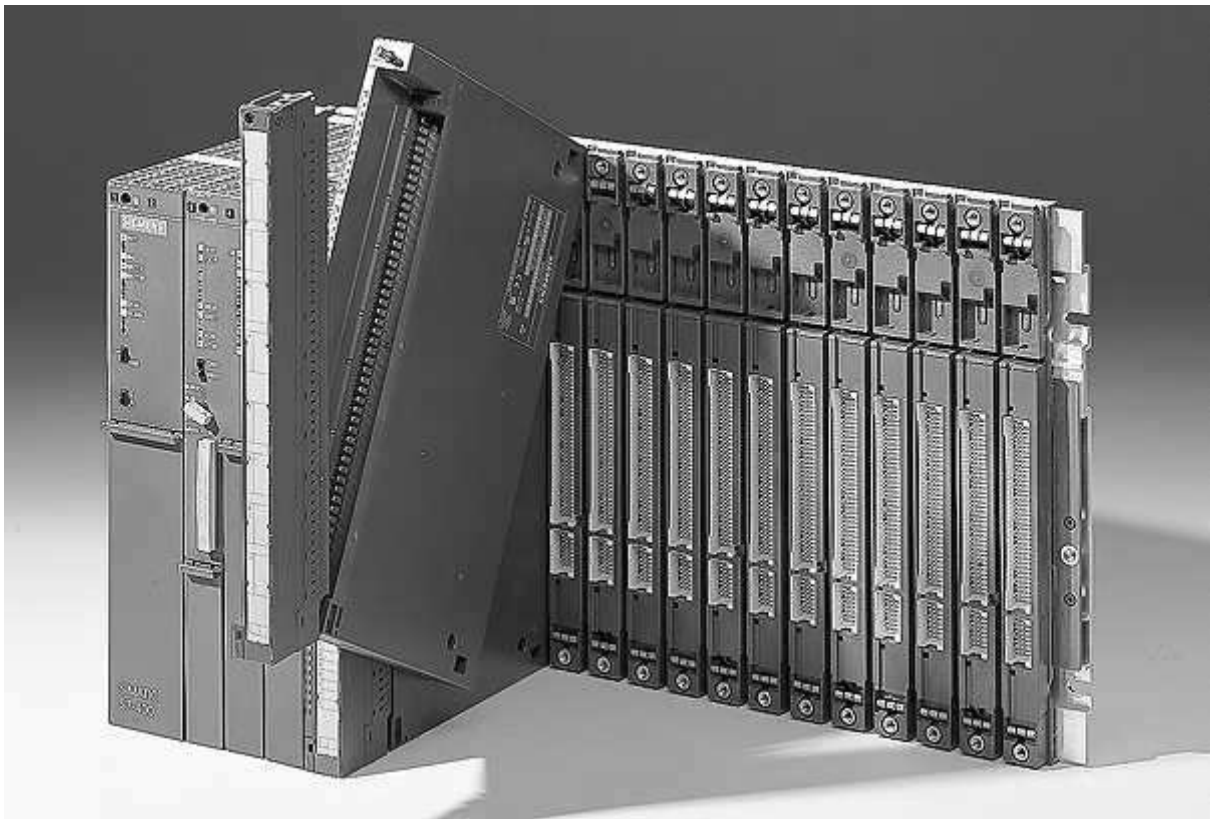


Рисунок 1.2 – Принцип установлення модулів контролера S7-400 у монтажну стійку UR1

Модулі різних типів вимагають різної кількості слотів для установлення. Потреба в кількості слотів представлена в таблиці 1.1.

Для зв'язку модулів у стійках UR служать дві шини: шина входів/виходів I/O (P-шина) і комунікаційна шина (K-шина).

P-шина призначена для високошвидкісного обміну даними із процесором, а комунікаційна K-шина забезпечує зв‘язок модуля CPU з комунікаційними процесорами та функціональними модулями.

Монтажна стійка базового блоку CR (central rack) може бути використана для розміщення до 18 модулів *(CR2)* або 4 модулів *(CR3)* базового блоку контролера.

Таблиця 1.1 – Кількість слотів, що необхідна для установлення модулів

Тип модуля	Потрібна кількість слотів
Модулі живлення: PS 405 4A, PS 407 4A PS 405 10A, PS 407 10A PS 405 20A, PS 407 20A	 1 2 3
Модулі процесорів: CPU 412-1, CPU 413-1, CPU 414-1, CPU 416-1 CPU 413-2 DP, CPU 414-2 DP	 1 2
Сигнальні модулі (SM)	1
Інтерфейсні модулі (IM)	1

Р-шина (шина введення-виведення) стійки CR2 розділена на два сегменти. Один сегмент охоплює 10, другий – 8 слотів (рознімань). Процесорний модуль потрібно встановити в кожний сегмент CR2 з відповідним набором модулів введення-виведення. Ланцюги живлення та К-шина (комунікаційна шина) є загальними для обох сегментів.

Стійка підтримує можливість використання стандартних або резервованих схем живлення контролера. У першому випадку в неї встановлюється один, у другому – два блоки живлення. У кожному разі установлення модулів блоків живлення починається з першого слота.

Інтерфейсні та сигнальні модулі, а також модулі процесора можуть установлюватися в будь-який слот, крім першого.

Стійка розширення ER1 (extension rack) застосовується для побудови недорогих пристроїв розширення введення-виведення. Вона дозволяє розміщати до 18 модулів S7-400.

В ER1 відсутня К-шина, що виключає можливість установлення комунікаційних процесорів і функціональних модулів. Крім того, обмежені функціональні можливості Р-шини – відсутня підтримка переривань і буферизація модулів. Відсутня також внутрішня шина живлення = 24 В. Установлення модулів блоків живлення починається з першого роз'єму монтажної стійки. Приймальний інтерфейсний модуль (один) повинен бути встановлений в останнє гніздо стійки.

Варіанти конфігурування станції S7-400

Для підключення стійок розширення UR та ER використовуються передавальні інтерфейсні модулі IM 460-0, IM 460-1, IM 460-3 та приймальні інтерфейсні модулі IM 461-0, IM 461-1, IM 461-3. Варіанти використання цих модулів наведені на рисунку 1.3.

Застосування інтерфейсних модулів IM 460-1 і IM 461-1 дозволяє додати до центральної стійки *одну* стійку розширення. Відмінною рисою цих інтерфейсних модулів є можливість подачі 5-вольтової напруги живлення від центральної стійки в стійку розширення, що дозволяє не встановлювати блок живлення в стійку розширення. При цьому довжина лінії зв'язку між стійками не повинна перевищувати 1,5 м.

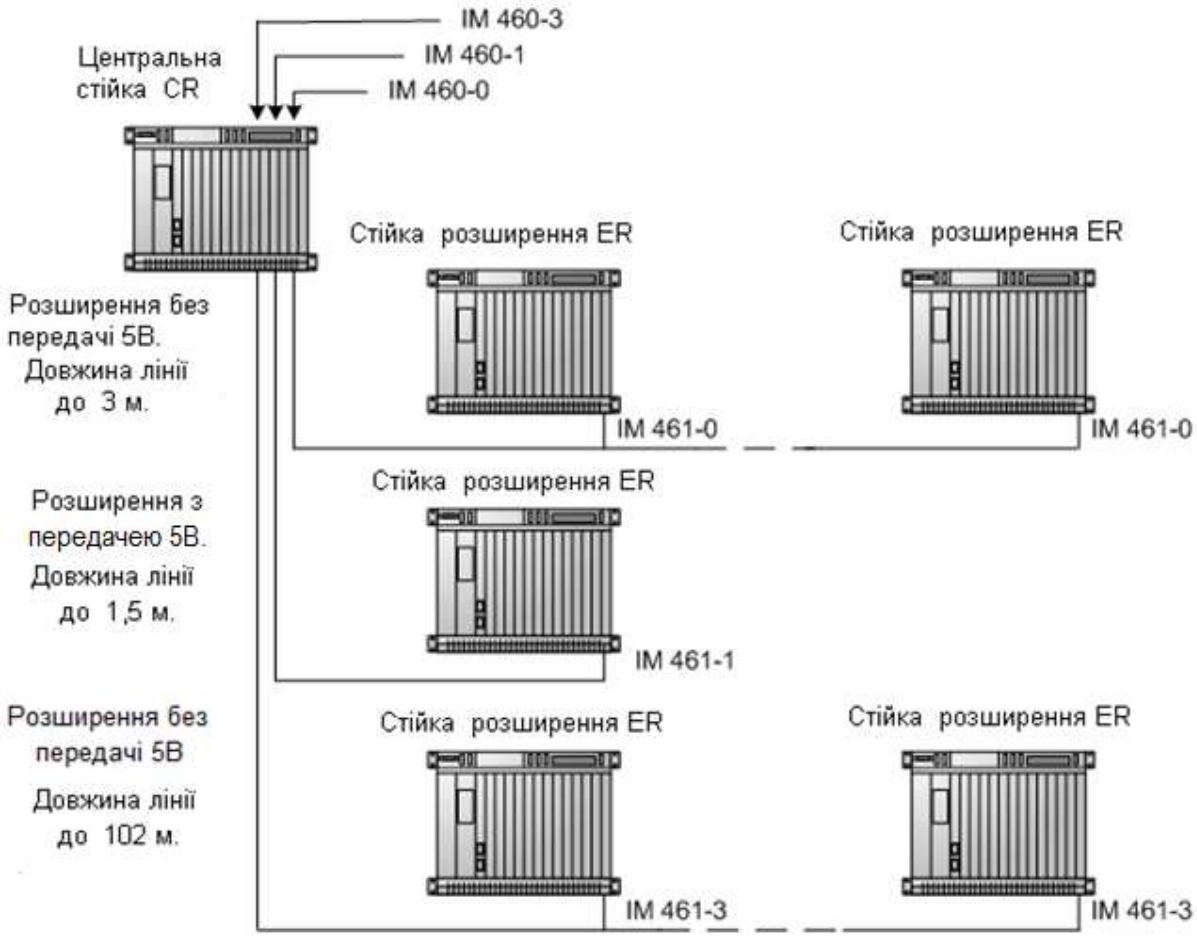


Рисунок 1.3 – Варіанти використання інтерфейсних модулів

Застосування інтерфейсних модулів IM 460-0 і IM 461-0 дозволяє додати до центральної стійки *від 1 до 4 стійок* розширення з довжиною лінії зв'язку до 3 метрів.

І, нарешті, за допомогою інтерфейсних модулів IM 460-3 і IM 461-3 до центральної стійки можливо підключення *від однієї до 4 стійок* розширення з довжиною лінії зв'язку до 102 метрів.

Слід прийняти до уваги, що в базовій стійці можна встановити до 6 передавальних інтерфейсних модулів. Приймальний модуль повинен бути установлений тільки в останній слот стійки розширення.

Варіанти конфігурування станції S7-300

Програмувальний контролер S7-300 монтується на стійку одного типу – стандартну профільну рейку (Rail) на 11 слотів. У базову (центральну) стійку встановлюються зліва направо: модуль живлення PS (слот 1), процесорний модуль CPU (слот 2), інтерфейсний передавальний модуль IM (слот 3), а також до 8 вхідних або вихідних модулів. Якщо така однорядна конфігурація контролера не є достатньою, то можливі два варіанти розширення конфігурації (рис. 1.4):

- варіант дворядної конфігурації, що має центральну стійку й одну стійку розширення. Цей варіант реалізується у разі використання інтерфейсних модулів IM 365, якщо відстань між стійками не перевищує одного метра;

- варіант конфігурації, що складається максимально з 4 рядів, тобто крім центральної стійки підключається до 3 стійок розширення. Цей варіант реалізується у разі використання передавального інтерфейсного модуля IM 360 і передавально-приймальних модулів IM 361 при відстані між стійками до 10 метрів.

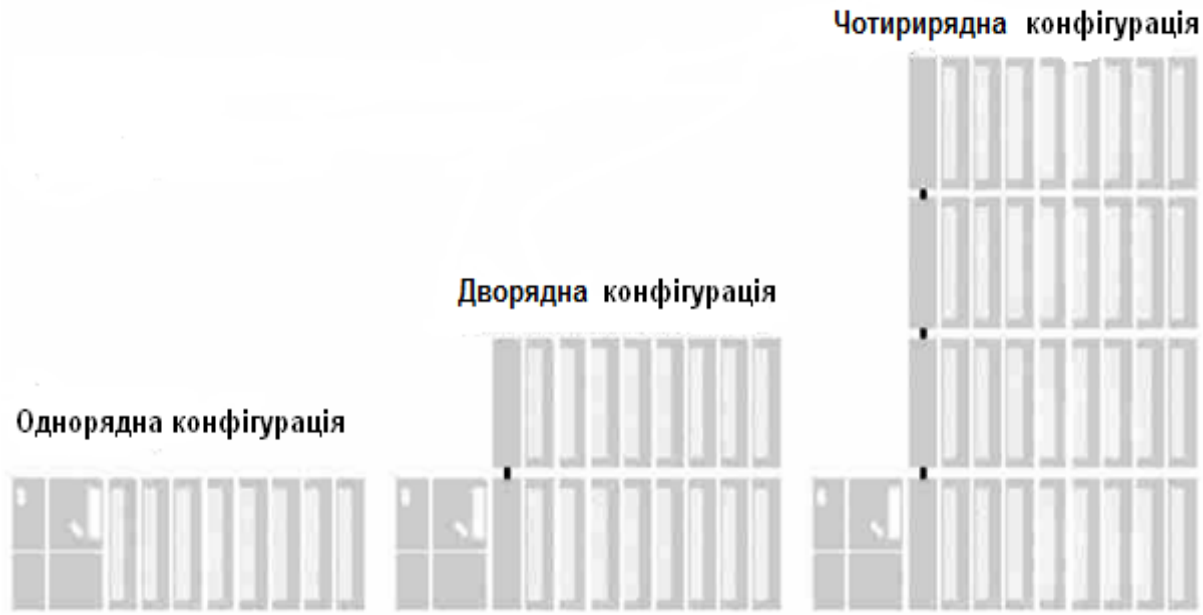


Рисунок 1.4 – Варіанти конфігурації станції S7-300

У кожній стійці можна задіяти до восьми модулів. Кількість модулів у стійках розширення може бути обмежена максимально припустимим струмом споживання на одну стійку, що становить 1,2 А.

Модулі з'єднуються між собою спеціальним шинним з'єднувачем, який забезпечує функції P- і K-шин.

1.4 Принципи адресації та організація роботи в адресному просторі контролера

Кожний слот із установленим модулем має фіксовану адресу в S7-станції, яка визначається номером монтажної стійки й номером слота. При цьому кожний модуль, установлений у слоті, може бути однозначно описаний через адресу слота, так звану «географічну адресу».

Якщо модуль містить інтерфейсні плати, кожна з них також описується адресою. І, нарешті, кожний дискретний або аналоговий сигнал у системі має свою власну унікальну адресу.

Початкова адреса модуля

Адресний простір входів/виходів починається з адреси 0 і закінчується деяким значенням, що відповідає верхній границі, яка визначається типом CPU.

У випадку дискретних модулів окремі сигнали (біти) збираються в групи по 8 біт, тобто в *байти*. Ці байти мають відповідні адреси 0, 1, 2 і 3. Адресація байтів починається з початкового модуля. Якщо початковий модуль введення має 32 канали, то в адресному просторі він отримує 4 байти, тобто байти 0, 1, 2 та 3.

В аналогових модулях кожний з аналогових каналів, які передають сигнали у вигляді напруги або струму, займає 2 байти, тобто слово, що позначається символом W. Аналогові модулі, залежно від конструкції, мають 2, 4, 8 і 16 каналів. При цьому кожний модуль, відповідно, буде займати 4, 8, 16 або 32 байти адресного простору.

При включенні живлення, якщо не було передустановок, CPU встановлює початкову адресу кожного модуля залежно від типу модуля, номера слоту й номера стійки. Значення адрес можна побачити в таблиці конфігурації.

Принцип адресації розподіленої периферії

Розподілені I/O модулі також мають «географічні адреси», які визначаються адресою системи провідного DP-пристрою й номером веденої станції (замість номера стійки).

Як і централізовані модулі, модулі розподіленої периферії (станції) резервують відповідні номери байтів в області I/O-адрес. При цьому адреси централізованих модулів і розподілених I/O не повинні перекриватися.

Ведені DP-пристрої можуть бути параметризовані таким чином, щоб особливі номери байтів забезпечували консистентність (логічну зв'язаність) даних під час їх пересилання. Для цього кожному веденому DP-пристрою відповідає один байт I/O адреси, яким вони адресуються у разі використання системних функцій SFC 14 (читання даних розподіленої периферії) і SFC 15 (запис даних розподіленої периферії).

MPI-адресація

Модулі, що є вузлами в MPI-мережі (CPU, FM або CP), мають MPI-адресу. Ця адреса відіграє вирішальну роль для зв'язку з програматорами PG, HMI-пристроями й для обміну глобальними даними. Модулі можуть бути адресовані програматором через номер стійки та номер слоту.

Необхідно враховувати, що адреси MPI функціональних модулів і комунікаційних процесорів в S7-300 автоматично визначаються центральним процесором таким алгоритмом:

- перший CP або перший FM після CPU ⇨ MPI-адреса CPU + 1;
- другий CP або другий FM після CPU ⇨ MPI-адреса CPU + 2.

Діагностичні адреси

Модулі з убудованою функцією діагностики забезпечують користувача діагностичними даними, які можуть оцінюватися в користувальницькій програмі.

Адреса даних діагностики завжди є адресою в I/O області входів і займає один байт пам'яті. STEP 7 автоматично призначає діагностичні адреси, відраховуючи довжину даних долілиць від верхнього максимального значення для адрес в області I/O.

Адресний простір контролера

Адресний простір кожного програмувального контролера містить:

- області периферійних входів і виходів (PI та PQ, відповідно);
- області відображення процесу за входом (I) й виходом (Q);
- область меркерів (M);
- області таймерів (T) і лічильників (C);
- область L-стека.

У SIMATIC S7 кожний модуль може мати дві області адрес:

- область даних користувача, що може бути безпосередньо адресована за допомогою операторів LOAD і TRANSFER;
- область системних даних для запису даних передачі.

Область даних користувача

Обсяг даних користувача визначається типом процесорного модуля. Адресація в цій області завжди починається з байта 0.

Властивості даних користувача в модулі також залежать від типу модуля. Для сигнальних модулів дані є дискретними або аналоговими вхідними / вихідними сигналами. Для інших модулів це можуть бути дані про стани (статуси).

У станціях S7 послідовність адрес *для цифрових модулів* починається з 0 і триває не більше ніж до 68 (18-й слот). Варто врахувати, що адреса 0 призначається тому слоту, в який встановлено перший цифровий модуль.

Кожний *канал* цифрового модуля представляється одним бітом, тобто кожний біт резервується за окремим входом. Адреса кожного біта визначається автоматично. Для цього система фіксує номер слота з першим цифровим модулем, з якого починається адресація, та визначає тип модуля. Якщо модуль має 32 канали, то йому призначається адресне поле від I 0.0 до I 3.7 (чотири байта). Наступний модуль матиме адресне поле від I 4.0 до I 7.7 і т. д.

Приклад. Нехай модуль *цифрового введення* з 32 каналами (4 байти), установлений у слоті 14. Якщо перший цифровий модуль було встановлено в слот 4 (слоти 1 та 2 зайняті блоком живлення, а слот 3 – процесорним модулем), то початковій адрес модуля в слоті 14 за замовчуванням дорівнює:

$(14 - 3) \times 4 = 44$.

Адреси каналів в модулі автоматично установлюються зверху долілиць.

Послідовність адрес за замовчуванням *для аналогових модулів* починається з адреси 512 і закінчується адресою 1600 (максимум).

Враховуючи те, що аналоговий модуль може мати 2, 4, 8 та 16 каналів, а для кожного каналу потрібно два байти, то для розрахунку адреси аналогового модуля за замовчуванням необхідно перемножити кількість каналів модуля на 2 та додати 512.

Приклад. Нехай в стійку встановлено три модулі аналогового введення. Перший має 8 каналів, другий – 4 канали, а третій – 2 канали. Тоді початкові адреси модулів дорівнюють:

Модуль AI 8×14 bit 512;

Модуль AI 4×13 bit *(8 ×2) + 512 = 528;*
Модуль AI 2×15 bit *(12 ×2) + 512 = 536.*

Для кожного каналу аналогового модуля передбачено формат виводу інформації словами (W). Тому адреси окремих каналів введення (I) та виведення (Q) збільшуються на два байти.

Дані користувача відображуються в області I/O адрес. Залежно від напрямку передачі ця область ділиться на PI-область (peripheral inputs – область периферійних входів) і PQ-область (peripheral outputs – область периферійних виходів).

Адресна область *периферійних входів* використовується під час читання даних вхідних модулів. Частина адрес PI-області відповідає області відображення даних процесу. Ця частина завжди починається з 0-ї адреси I/O, при цьому розмір області визначається типом CPU.

Адресна область *периферійних виходів* використовується під час запису даних вихідних модулів. Частина адрес PQ-області відповідає області відображення даних процесу. Ця частина завжди починається з 0-ї адреси I/O, а розмір області також визначається типом CPU.

Варто врахувати, що периферійні входи та периферійні виходи мають одинакові розміри областей, які відрізняються тільки символами I і Q.

Відображення процесу (образ процесу)

Відображення процесу (образ процесу) складається з образу дискретних вхідних і дискретних вихідних модулів і, таким чином, підрозділяється на *образ входів* процесу й *образ виходів* процесу.

До образу входів процесу доступ здійснюється в адресній області входів (I), а до образу виходів процесу – в адресній області виходів (Q).

Образ входів – це відображення відповідного біта в дискретному вхідному модулі під час сканування. Сканування входу – це те ж саме, що й сканування біта в самому модулі. Перед виконанням програми в кожному програмному циклі операційна система CPU копіює значення сигналу із модуля в образ входів процесу, що представляється таблицею відображення.

На рисунку 1.5 представлена схема процесу на прикладі сканування кнопки включення мотора. Ця кнопка підключена до модуля на 16 входів за адресою I 5.2.

Вираз «I 5.2» – це абсолютна адреса сигналу включення. У таблиці символів цій адресі можна привласнити символічне ім'я, наприклад, «Switch motor on». Тоді вираз «Switch motor on» є символьною адресою.

Використання образу входів процесу має такі переваги:

- входи можуть бути просканованої й записані послідовно біт за бітом. Завдяки цьому програма виконується швидше, тому що сканування входів виконується швидше, ніж процедура одержання доступу до модуля;
- стан входу не міняється протягом усього циклу програми. У разі зміни біта вхідного модуля ця зміна стану сигналу буде перенесена на відповідний вхід образу процесу лише на початку наступного циклу;
- біти дискретних вхідних модулів доступні тільки для читання,

у той час, як біти входу перебувають в RAM-пам'яті й доступні як для читання, так і для записування. Завдяки цьому з'являється можливість змінювати вхідні біти з метою налагодження під час тестування програми.

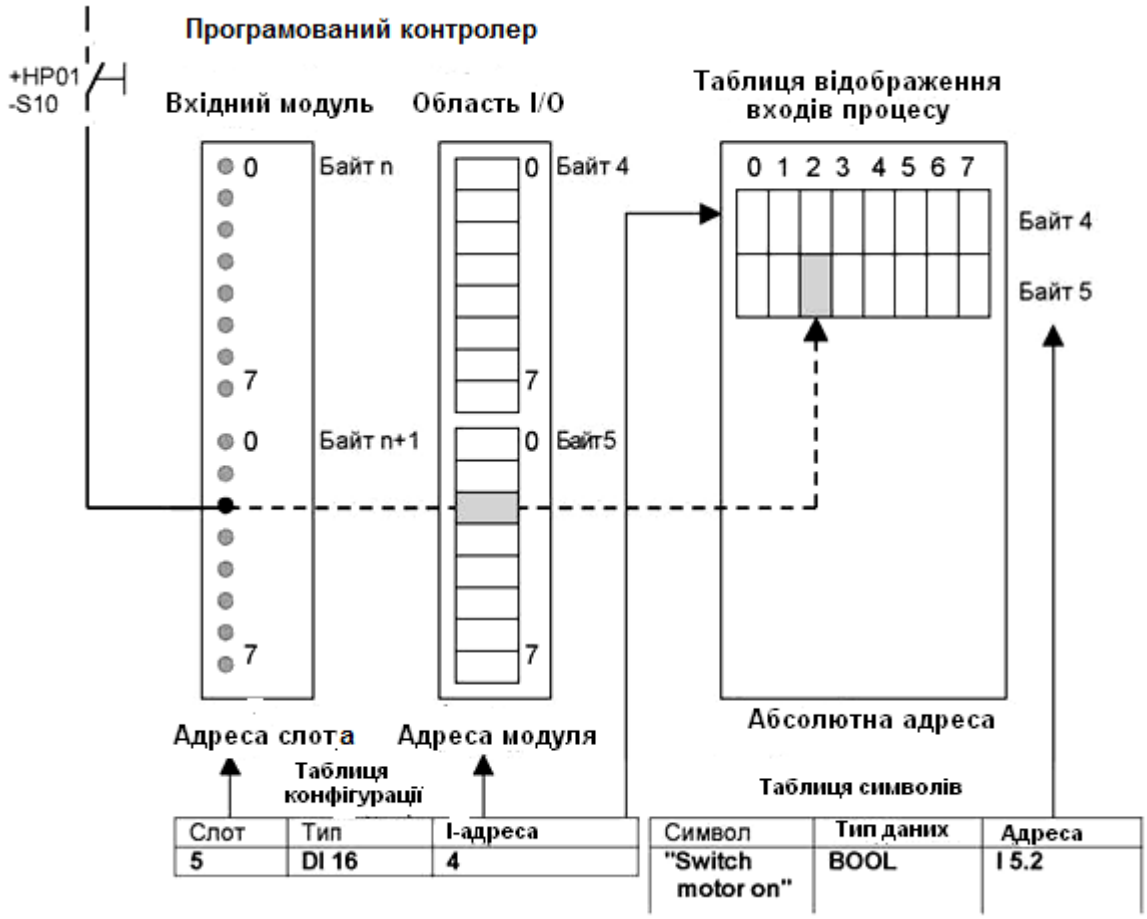


Рисунок 1.5 – Схема проходження сигналу від кнопки включення до області відображення входів в RAM-пам'яті

Образ виходів – це відображення відповідних бітів у дискретному вихідному модулі. Установлення виходу – це те ж саме, що й установлення біта в самому модулі. Операційна система CPU копіює значення бітів з образу виходів процесу у вихідний модуль.

Використання образу виходів процесу дає такі переваги:

1. Виходи можуть бути встановлені або скинуті біт за бітом. Установка виходів здійснюється швидше, ніж процедура одержання доступу до вихідного модуля. Отже, програма також виконується швидше;

2. Стан виходу може багаторазово мінятися протягом усього циклу програми, при цьому стан сигналу біта вихідного модуля залишається без зміни. І лише останній стан сигналу буде перенесено у відповідний вихідний модуль на початку нового програмного циклу;

3. Виходи можуть бути прокановані, тому що вони перебувають у RAM-пам'яті, тоді як біти дискретних вихідних модулів доступні тільки для запису, але недоступні для читання.

Область системних даних

В області системної пам'яті CPU розташовуються *меркери*. Меркери можуть розглядатися як додаткові «пускові реле» контролера. Кількість меркерів залежить від типу обраного CPU.

Меркери використовуються для зберігання проміжних результатів, які можуть знадобитися в інших блоках. Для зберігання проміжних результатів підходять:

- тимчасові локальні дані, що можливі у всіх блоках, але є дійсними тільки для поточного виклику блоку;
- статичні локальні дані, що можливі тільки у функціональних блоках, але є дійсними для багатьох викликів блоків.

Деякі меркери можуть бути призначені реманентними меркерами. Такі меркери зберігають свій стан навіть в умовах вимикання живлення. Реманентна область завжди починається з адреси «0» й закінчується в заданому місці пам'яті (задається при параметризації CPU).

1.5 Конфігурування станцій у програмному середовищі STEP 7

Програмна система STEP 7 призначена для виконання всього комплексу проектування – конфігурування, програмування й тестування системи автоматизації. Створення проекту здійснюється в графічній оболонці SIMATIC Manager, а конфігурування системи виконується в додатках Hardware Configuration (*Конфігурування апаратури*) і NetPro (*Проектування мереж*).

Під час проектування необхідно задовольнити такі вимоги:

1. Обрані блоки живлення повинні забезпечити необхідний струм споживання;
2. Модулі CPU повинні задовольняти вимогам до комунікацій;
3. При виборі інтерфейсного модуля для з'єднання стійок необхідно визначити доцільність передачі струму в стійки розширення, відстань, кількість стійок розширення, а також необхідність комунікаційної шини;
4. При виборі сигнальних модулів обґрунтувати типи модулів з урахуванням напруг, навантажувальних здатностей і типів з'єднань із зовнішніми пристроями (чи можливо групування каналів, яка кількість точок з'єднання, який опір навантаження);
5. При виборі комунікаційних процесорів необхідно звернути увагу на тип мережі, підтримувані цими мережами комунікаційні функції, а також пристрої, з якими комунікаційний процесор може взаємодіяти.

У результаті конфігурування центральної станції треба отримати файл конфігурації з розширенням «.cfg». Цей файл створюється командою «Station ⇨ Export...» у форматі XML. Під час створення файлу необхідно вказати директорію для його збереження. Відкрити файл можна в редакторі XML Editor.

Для конфігурування системи автоматизації в додатку Hardware Configuration використовуються два вікна:

- вікно SIMATIC Station (*Станція SIMATIC*), у якому конфігуруються стійки;

- вікно HW Config, у якому є розділ Hardware Catalog (*Каталог апаратури*) для вибору необхідних апаратних компонентів (стійок, сигнальних й інтерфейсних модулів, тощо).

На рис. 1.6 центральна стійка позначена (0)UR, де 0 – порядковий номер стійки, UR – тип стійки (Universal Rack – універсальна стійка).

Процес конфігурування полягає в тім, що необхідні компоненти вибираються у вікні Hardware Catalog і переносяться у вікно станції. Компонування станції відображається в конфігураційній таблиці стійки, розташованій під вікном станції (рис. 1.7). У таблиці відображаються номери слотів, найменування модулів, їх адреси й замовні номери.

Основний порядок параметрування

Після того як компонент розміщено у вікні станції, можна перейти в режим діалогу для зміни встановлених за замовчуванням параметрів або адрес (режим параметрування).

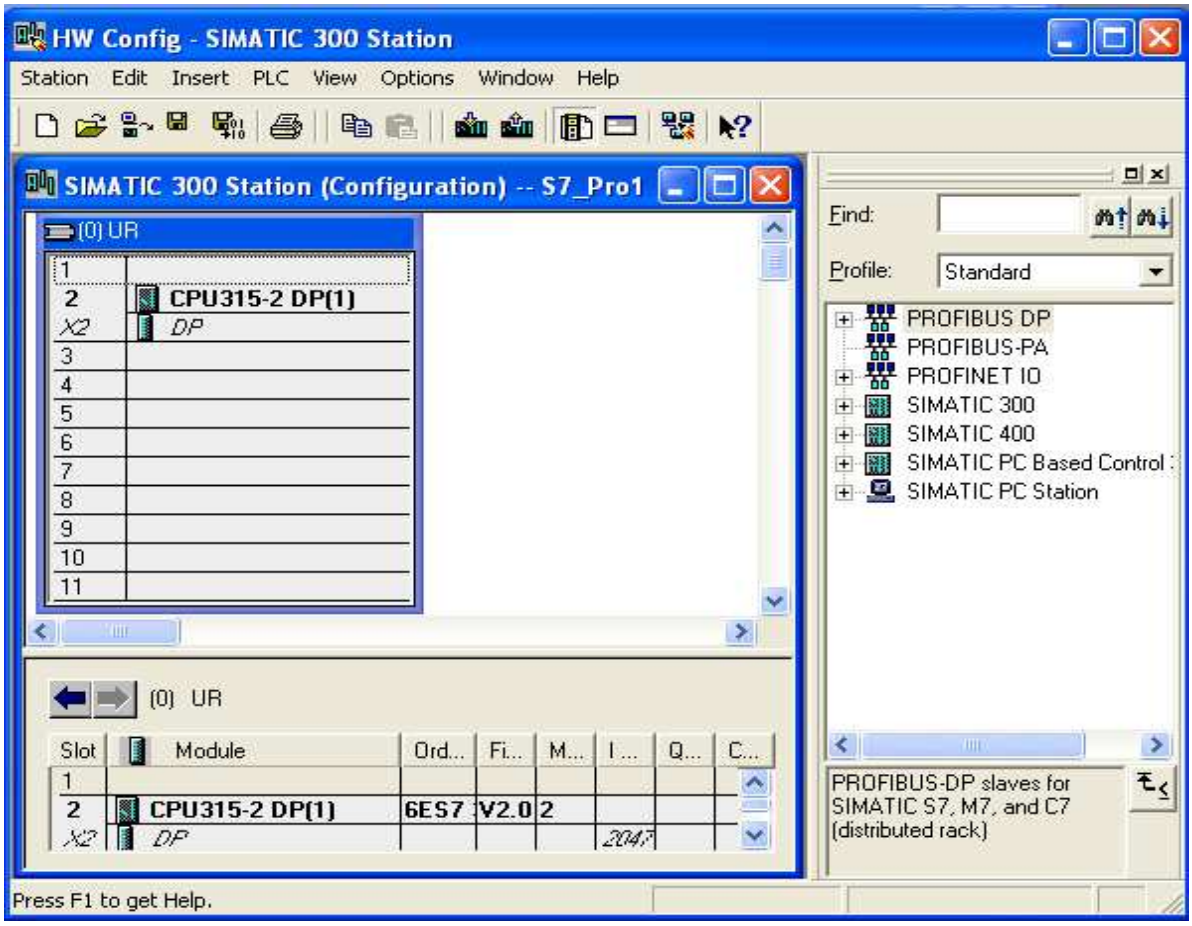


Рисунок 1.6 – Розташування вікон інтерфейсу в середовищі конфігурування станції HW Config

Для переходу у вікно установки властивостей компонента можна застосувати один із способів:

- двічі клацнути на компоненті лівою кнопкою миші;
- вибрати команду меню Edit ⇨ Object Properties (*Редагування* ⇨ *Властивості об'єкта*);

• за допомогою правої кнопки миші вибрати зі спливаючого меню команду Object Properties (*Властивості об'єкта*).

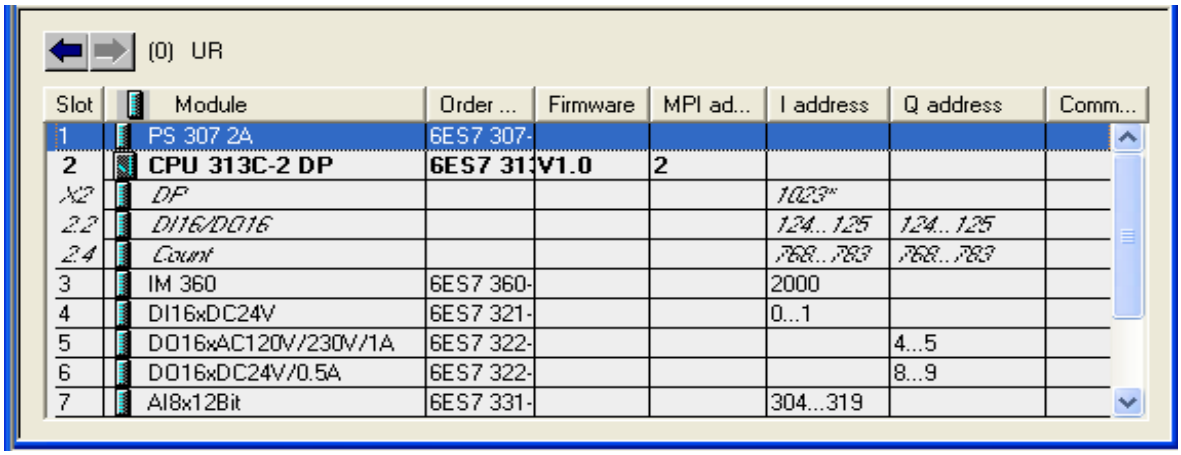


Рисунок 1.7 – Вид таблиці конфігурування стійки

Для настроювання поводження системи особливе значення мають властивості CPU. На закладках CPU можна встановити характеристики запуску, області локальних даних і пріоритети для переривань, області пам'яті, характеристики реманентності (збереження даних у пам'яті після вимикання живлення), тактові меркери, рівень захисту й пароль.

У закладці General (*Загальне*) CPU можна параметрувати інтерфейси, наприклад, MPI або убудований інтерфейс PROFIBUS-DP.

Збереження конфігурації

Щоб зберегти конфігурацію з усіма встановленими параметрами, необхідно вибрати команду меню Station ⇨ Save або команду меню Station ⇨ Save and Compile (*Станція ⇨ Зберегти й компілювати*). Для збереження незакінченої конфігурації виберіть команду Station ⇨ Save.

Створення станції

Станція може створюватися тільки безпосередньо під проектом. Тому спочатку необхідно виділити у лівій частині вікна проект, а потім вибрати команду меню Insert ⇨ Station ⇨ SIMATIC 300-Station, (*Вставити ⇨ Станція ⇨ Станція SIMATIC 300*) або ... SIMATIC 400-Station.

Станція створюється з ім'ям, даним за замовчуванням. Це ім'я можна замінити іншим, більш інформативним.

Після створення станції необхідно:

• виділити у вікні проектів об'єкт Station, після чого в правій частині вікна станції стає видимим об'єкт Hardware (*Апаратура*);

 – об'єкт «Station»,  – об'єкт «Hardware»;

• можна також виділити об'єкт Station і вибрати команду меню Edit ⇨ Open Object (*Редагувати ⇨ Відкрити об'єкт*);

• у результаті на екрані з'являються вікно станції й каталог модулів;

• у вікні станції можна помістити стійку й інші компоненти у відповідності зі структурою станції, а з каталогу модулів у вікні Hardware Catalog вибрати необхідні для побудови станції компоненти;

• командою меню Station ⇨ New (*Станція ⇨ Нова*) можна сконфігурувати у тім же проекті ще одну станцію.

Проектування центральної стійки

Центральна стійка проектується в такій послідовності:

1. У вікні Hardware Catalog потрібно вибрати центральну стійку (Rack). Для SIMATIC 300 це профільна рейка (Rail), для SIMATIC 400 може бути, наприклад, універсальна стійка (UR1);

2. Використовуючи метод Drag&Drop, варто відбуксирувати стійку у вікно станції. Стійка *з'являється у вигляді невеликої конфігураційної таблиці* у верхній частині вікна станції. У нижній частині вікна станції з'являється докладне подання стійки з додатковими даними – номером для замовлення, адресою MPI, адресами входів/виходів.

Компонування стійки здійснюється в такій послідовності:

1. Вибирається модуль із вікна Hardware Catalog. При цьому слоти, у які можна встановити цей модуль, виділяються кольором;

2. З використанням Drag&Drop модуль буксирується у відповідний рядок стійки. При цьому STEP 7 перевіряє, чи не порушені правила для слотів;

3. Кроки 1 і 2 повторюються доти, поки стійка не буде повністю оснащена бажаними модулями.

Примітка. Під час виділення слота в стійці можна побачити список всіх можливих для установлення модулів. Для цього необхідно правою кнопкою миші відкрити контекстно-залежне меню й у ньому вибрати Insert Object (*Вставити об'єкт*) або Replace Object (*Замінити об'єкт*). Ця можливість рятує від необхідності пошуку апаратури в каталозі.

Відображення інтерфейсів й інтерфейсних модулів

Інтерфейси або интерфейсні модулі відображаються в конфігураційній таблиці у власному рядку. Цей рядок позначений так само, як і коннектор інтерфейсу, наприклад, *X1*.

При наявності *вбудованих* інтерфейсів ім'я інтерфейсу з'являється у стовпчику Module (*Модуль*). Для установлення інтерфейсних модулів потрібно перенести відповідний інтерфейсний модуль (IF) з вікна Hardware Catalog у відповідний рядок, використовуючи Drag&Drop.

Якщо CPU має більше однієї версії операційної системи, то він показується у вікні Hardware Catalog як папка з іконками, що мають різні порядкові номери.

Конфігурування стійок розширення для SIMATIC 300

Для станцій SIMATIC 300 як в якості центральної стійки, так і в якості стійки розширення використовуються тільки профільні шини. Кількість профільних шин визначається реальною конструкцією, однак їх не повинно бути більше чотирьох.

Стійки розширення з'єднуються в STEP 7 шляхом установлення відповідних інтерфейсних модулів:

• *для розширення тільки на одну стійку* в стійках 0 (центральна стійка) і 1 (стійка розширення) установлюються модулі IM 365;

- *для підключення до трьох стійок розширення* в стійці 0 установлюється модуль IM 360, а в стійках з 1 по 3 модулі IM 361.

Правила розміщення модулів у станції SIMATIC-400

Правила розміщення модулів у станції S7-400 залежать від типу застосовуваної стійки. У центральній стійці модулі розміщуються за такими правилами:

- блоки живлення встановлюються тільки в слоті 1 і наступних слотах;
- кількість інтерфейсних модулів повинна бути не більше 6, з них не більше двох з передачею живлення;
- до центральної стійки через інтерфейсні модулі можна підключити не більше 21 стійки розширення;
- до інтерфейсу *передавального* IM 460-1 можна підключити не більше 1 стійки розширення з *передачею струму;*
- до інтерфейсу *передавального* IM 460-0 або IM 460-3 можна підключити не більше 4 стійок розширення *без передачі струму*.

У стійках можуть установлюватися резервовані блоки живлення. Для резервованих блоків живлення станції S7-400 варто враховувати такі правила:

- установлення резервованих блоків живлення можливо тільки у призначені для цього стійки (розпізнаються за більшим замовленим номером і текстом у вікні каталогу апаратури Hardware Catalog);
- резервовані блоки живлення можуть експлуатуватися тільки разом із призначеними для цього CPU (непридатні CPU, наприклад, більш старої версії, при конфігуруванні відкидаються);
- резервовані блоки живлення повинні встановлюватися зі слота 1 без пропусків;
- резервовані й не резервовані блоки живлення не можуть вставлятися в ту ж саму стійку, тобто змішана експлуатація неможлива.

Порядок конфігурування стійок розширення

1. Виберіть відповідну стійку розширення з каталогу апаратури Hardware Catalog;
2. Перетягніть стійку, використовуючи Drag&Drop, у вікно станції;
3. Розмістіть в стійці модулі. При цьому блок живлення повинен бути встановлений у першому слоті, а інтерфейсний модуль – *в останньому слоті*;
4. Зробіть з'єднання між інтерфейсними модулями, установленими в центральній стійці й стійці розширення. Із цією метою клацніть двічі на передавальному IM. У закладці *Connect* відображаються всі стійки із установленими приймальними IM. Вибравши стійку розширення, підключіть її за допомогою екранної кнопки *Connect* до інтерфейсу передавального IM (C1 або C2). Підтвердіть з'єднання кнопкою OK.

Після виконання цих дій між інтерфейсними модулями з'явиться сполучна лінія.

На рисунку 1.8, як приклад, показано вікно станції з відображенням центральної стійки (0) і стійки розширення (1), з'єднаних між собою інтерфейсними модулями IM 460-0 і IM 461-0.

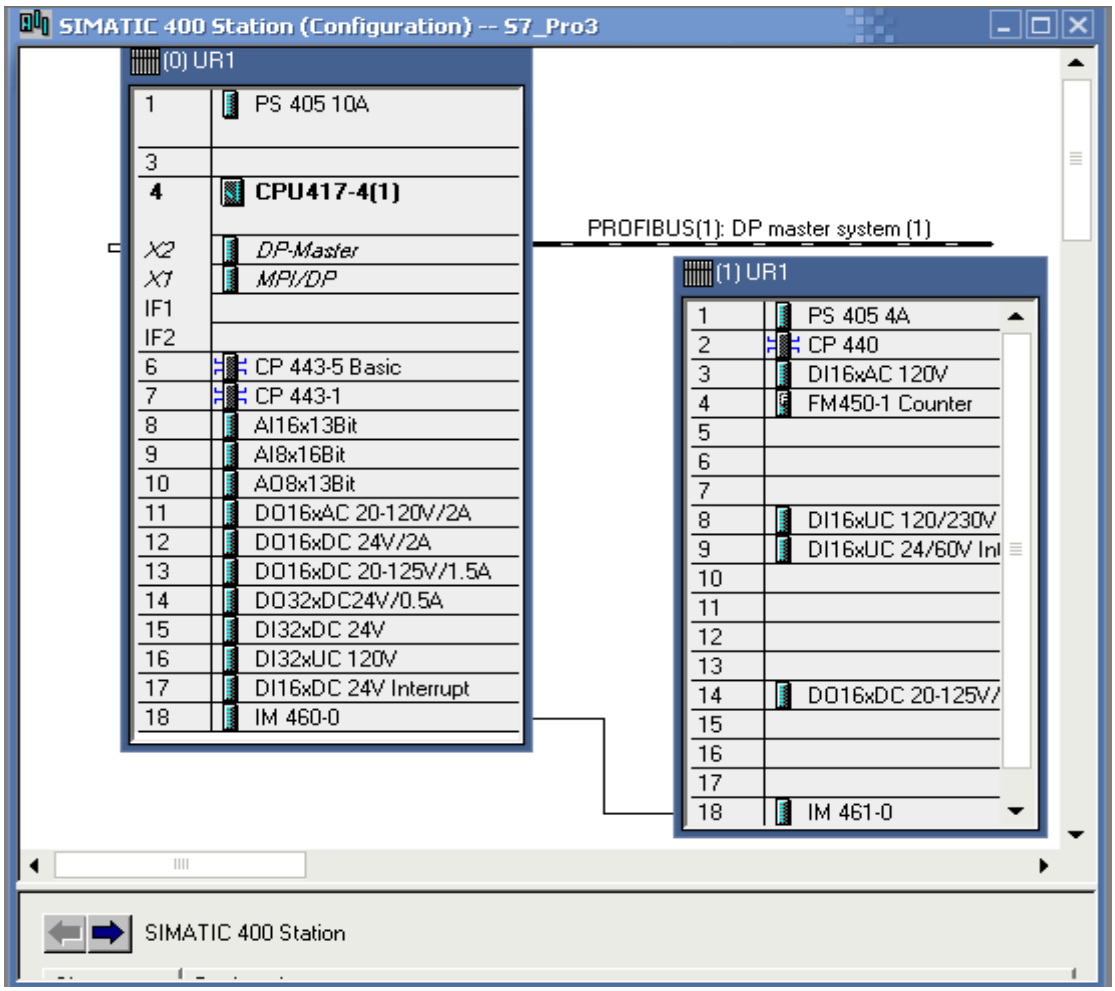


Рисунок 1.8 – Приклад конфігурування стійок

Заміна стійок у станції

Заміна стійок станції SIMATIC S7-400 виправдана, якщо в результаті цієї заміни функціональні можливості станції розширюються. Це відбувається в таких випадках:

- заміна стійки, що не підтримує резервування системи живлення, на стійку, що підтримує резервування;
- заміна короткої стійки (9 слотів) на довгу (18 слотів) для установлення додаткових модулів. Для стійок, що конфігуруються як стійки розширення (UR або ER) із *приймальними* IM, приймальні IM автоматично переміщаються в останній слот.

Правила заміни стійок

Стійка станції SIMATIC 400 може бути замінена на іншу, тільки у разі дотримання таких *основних правил* (якщо хоча б одне правило не виконуються, STEP 7 не дозволяє заміну й перериває процедуру з повідомленням про помилку):

• сегментована стійка (CR2) не може бути замінена несегментованою (наприклад, UR1) і навпаки. Слоти із двох сегментів не можуть бути однозначно зіставлені слотам в не сегментованій стійці. Тому стійка CR2 може бути замінена тільки на стійку CR2 з іншим порядковим номером, наприклад, щоб забезпечити установлення модулів з резервуванням живлення без того, щоб проводити все конфігурування знову;

• під час установлення модулів у нову стійку не повинні порушуватися ніякі правила слотів;

• не дозволяється заміна стійки UR1 із установленим CPU на стійку розширення ER1, тому що установлення CPU в ER1 заборонено правилами слотів.

Якщо станція має складну структуру, наприклад, містить кілька стійок, то можна настроїти станцію на мінімальний розмір.

Для мінімізації розміру станції необхідно виконати таке:

1. Виділити конфігураційну таблицю;

2. Натиснути праву кнопку миші й вибрати в спливаючому меню команду Minimize (*Мінімальний розмір*).

Заміна компонентів

Якщо потрібно замінити компонент у стійці із вже встановленими модулями, виконайте таке:

1. Виділіть у конфігурації станції компонент, який потрібно замінити;

2. Виділіть у вікні каталогу апаратури ідентичний компонент, сумісний із замінним. Наприклад, для ведених DP можна виділити інтерфейсний модуль IM 153-2;

3. Двічі клацніть на необхідному компоненті в каталозі апаратури. Якщо компоненти сумісні, вони заміняються, причому модулі з вихідної конфігурації зберігають настроювання адрес і параметрів.

Заміна компонента можлива також з використанням буксирування методом drag-and-drop з каталогу апаратури.

Якщо з'являється повідомлення The slot is already occupied (*Слот уже зайнятий*), потрібно спочатку активізувати функцію заміни командою меню Options ⇨ Customize (*Можливості* ⇨ *Настроювання*), а потім вибрати настроювання Enable Module Exchange (*Дозволити заміну модулів*).

1.6 Параметрування модулів й інтерфейсів

Параметрування модулів

Модулі повинні мати властивості, які представляються адресами й параметрами. Зазвичай ці властивості встановлюються за замовчуванням. Однак у багатьох випадках установлені за замовчуванням значення не відповідають конкретним вимогам. Так, наприклад, заздалегідь установлені види й діапазони вимірів в аналогових модулях навряд чи будуть відповідати бажаним.

Якщо потрібно змінити ці настроювання, треба діяти в такий спосіб:

1. Двічі клацніть у конфігураційній таблиці на компоненті, що підлягає параметризації, наприклад, на інтерфейсному модулі, або виділіть

рядок і виберіть команду меню Edit ⇨ Object Properties (*Редагувати* ⇨ *Властивості об'єкта*);

2. Використовуючи діалогове вікно, що з'явилося, встановіть властивості компонента.

Призначення адрес вузлів і входів – виходів

Під час призначення адрес варто розрізняти призначення адрес *вузлам* і призначення адрес *входам – виходам*.

Адреси вузлів призначаються програмувальним модулям у мережах MPI, PROFIBUS, Industrial Ethernet.

Адреси входів – виходів (I/O) призначаються модулям для того, щоб у програмі користувача зчитувати входи або встановлювати виходи.

Уже використані адреси входів і виходів можна відобразити в такий спосіб:

1 відкрити станцію, адреси якої потрібно переглянути;

2 вибрати команду меню View ⇨ Address Overview [*Вид* ⇨ *Огляд адрес*];

3 виділити в діалоговому вікні Address Overview модуль, у якому повинні бути відображені призначені входи й виходи, наприклад, CPU;

4 якщо необхідно, слід відфільтрувати відображення за видами адрес, наприклад, «тільки адреси входів».

Адресні області входів і виходів відображаються із вказівкою місця розміщення модулів – номером майстер-системи DP, номером стійки, слоту або гнізда. Адреси входів, що мають нульову довжину, наприклад, адреси інтерфейсних модулів, позначаються зірочкою (I*).

На рисунку 1.9 показаний приклад розподілу адрес входів (I) і виходів (Q) для системи S7-400 (CPU 417-4), що складається із центральної стійки (0) і стійки розширення (1). Область адресного простору центрального процесора становить 16383 байта.

У таблиці вікна Address Overview наведені такі дані (зліва направо):

- тип даних – вхід (I), вихід (Q);
- області адрес кожного модуля (Addr. from і Addr. to);
- найменування модуля;
- тип поділу образа процесу PIP (Process image partition), у якому зазначений організаційний блок OB1 (циклічне виконання);
- адреси в PROFIBUS DP (колонка DP) і PROFINET (колонка PN);
- номер стійки R (Rack) і номер слота S (Slot);
- стартові адреси модулів інтерфейсів DP-Master і MPI (колонка IF).

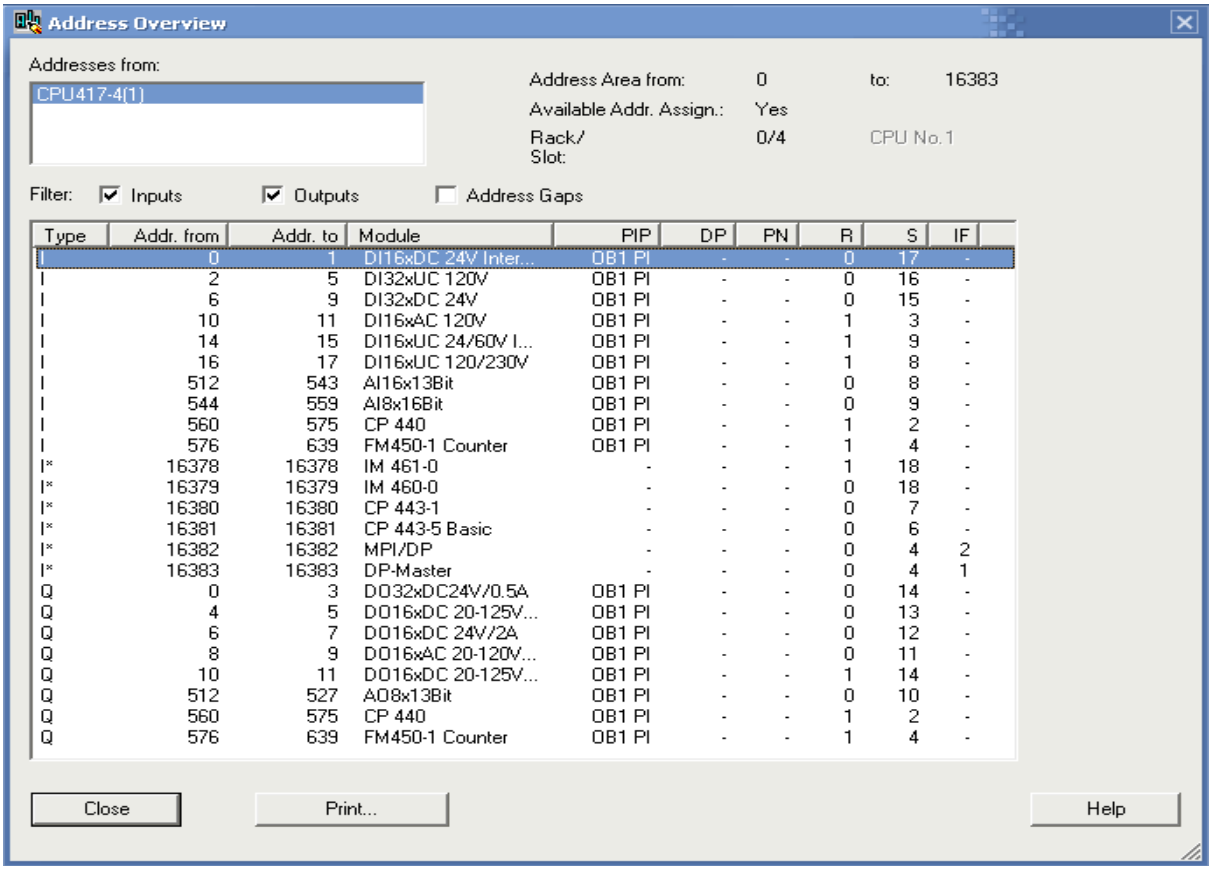
Призначення символьних імен адресам входів і виходів

Уже при конфігуруванні *цифрових або аналогових модулів* їхнім входам і виходам можна призначити символьні імена, не відкриваючи для цього таблицю символів.

При завданні символьних імен необхідно:

1 виділити цифровий або аналоговий модуль, адресам якого привласнюються символьні імена;

2 вибрати команду меню Edit ⇨ Symbols і у вікні Edit Symbols внести символічні імена входів або виходів. Якщо клацнути на наявній у діалоговому вікні кнопці Add Symbol, то як символ буде внесена адреса операнда.



Type	Addr. from	Addr. to	Module	PIP	DP	PN	R	S	IF
I	0	1	DI16xDC 24V Inter...	OB1 PI	-	-	0	17	-
I	2	5	DI32xUC 120V	OB1 PI	-	-	0	16	-
I	6	9	DI32xDC 24V	OB1 PI	-	-	0	15	-
I	10	11	DI16xAC 120V	OB1 PI	-	-	1	3	-
I	14	15	DI16xUC 24/60V I...	OB1 PI	-	-	1	9	-
I	16	17	DI16xUC 120/230V	OB1 PI	-	-	1	8	-
I	512	543	AI16x13Bit	OB1 PI	-	-	0	8	-
I	544	559	AI8x16Bit	OB1 PI	-	-	0	9	-
I	560	575	CP 440	OB1 PI	-	-	1	2	-
I	576	639	FM450-1 Counter	OB1 PI	-	-	1	4	-
I*	16378	16378	IM 461-0	-	-	-	1	18	-
I*	16379	16379	IM 460-0	-	-	-	0	18	-
I*	16380	16380	CP 443-1	-	-	-	0	7	-
I*	16381	16381	CP 443-5 Basic	-	-	-	0	6	-
I*	16382	16382	MPI/DP	-	-	-	0	4	2
I*	16383	16383	DP-Master	-	-	-	0	4	1
Q	0	3	DO32xDC24V/0.5A	OB1 PI	-	-	0	14	-
Q	4	5	DO16xDC 20-125V...	OB1 PI	-	-	0	13	-
Q	6	7	DO16xDC 24V/2A	OB1 PI	-	-	0	12	-
Q	8	9	DO16xAC 20-120V...	OB1 PI	-	-	0	11	-
Q	10	11	DO16xDC 20-125V...	OB1 PI	-	-	1	14	-
Q	512	527	AO8x13Bit	OB1 PI	-	-	0	10	-
Q	560	575	CP 440	OB1 PI	-	-	1	2	-
Q	576	639	FM450-1 Counter	OB1 PI	-	-	1	4	-

Рисунок 1.9 – Приклад відображення адрес у вікні Address Overview

Контрольні питання

1. Перелічіть компоненти, з яких складається програмувальний контролер SIMATIC.
2. Як улаштована і для чого призначена завантажувальна пам'ять CPU?
3. Що являє собою робоча пам'ять і для чого вона призначена?
4. Які області містить системна пам'ять?
5. За якими правилами встановлюються модулі в центральну стійку?
6. За якими правилами встановлюються модулі в стійку розширення?
7. Які шини використовуються в стійках для зв'язку модулів?
8. Які варіанти конфігурації застосовуються при створенні контролера S7-300?
9. Які інтерфейсні модулі застосовуються для з'єднання стійок в S7-300?
10. Які варіанти конфігурації застосовуються при створенні контролера S7-400?
11. Які інтерфейсні модулі застосовуються для з'єднання стійок в S7-400?
12. Що являють собою монтажні стійки UR1 і UR2?

13. Що являє собою монтажна стійка базового блоку CR2?

14. Які особливості має стійка розширення ER1?

15. Як визначаються адреси дискретних модулів введення – виведення?

16. Як визначаються адреси аналогових модулів введення – виведення?

17. Як визначаються географічні адреси розподіленої периферії?

18. Як визначаються адреси вузлів MPI-мережі?

19. Як визначаються діагностичні адреси модулів з убудованими функціями діагностики?

20. З яких областей складається адресний простір програмувального контролера?

21. Що містить у собі область даних користувача?

22. За яким принципом призначається адреса модуля й каналу дискретного введення?

23. За яким принципом призначається адреса модуля й каналу виведення аналогового сигналу?

24. Для чого призначені області периферійних входів і периферійних виходів?

25. Для чого призначена область відображення входів процесу та які переваги вона створює?

26. Для чого призначена область відображення виходів процесу та які переваги вона створює?

27. Для чого призначаються меркери?

2 ОСОБЛИВОСТІ ЗАСТОСУВАННЯ БАЗОВИХ МОДУЛІВ

2.1 Центральні процесори (CPU)

Центральні процесори S7-300

Програмувальні контролери S7-300 можуть комплектуватися 17 типами центральних процесорів. Ці процесори характеризуються часом виконання логічної інструкції (0,1...0,2 мкс) і обсягом робочої пам'яті (від 16 Кбайт в CPU 312 до 512 Кбайт в CPU 317).

Центральні процесори стандартного виконання розрізняються в такому:

- CPU 312 – центральний процесор для побудови невеликих систем керування, які містять до 8 модулів локального введення – виведення S7-300;
- CPU 314 – центральний процесор для побудови систем керування, у яких потрібна швидкісна обробка інформації й підтримка до 32 модулів локального введення – виведення S7-300;
- CPU 315-2 DP – центральний процесор з вбудованим інтерфейсом провідного / веденого пристрою PROFIBUS DP для побудови систем автоматизації з розвинутою системою локального й розподіленого введення – виведення;
- CPU 315-2 PN/DP – центральний процесор з вбудованим інтерфейсом Industrial Ethernet і PROFIBUS DP, що забезпечує підтримку комунікаційного стандарту PROFINET і призначений для використання в системах CBA (Component Based Automation);
- CPU 317-2 DP – центральний процесор з вбудованими інтерфейсами MPI/PROFIBUS DP і PROFIBUS DP, більшим обсягом пам'яті програм і даних, високою продуктивністю. Призначений для побудови високопродуктивних систем автоматизації з розвинутою системою локального й розподіленого введення – виведення;
- CPU 317-2 PN/DP – центральний процесор з вбудованим інтерфейсом Industrial Ethernet, PROFINET і PROFIBUS DP, призначений для використання в системах з розподіленим інтелектом CBA (Component Based Automation) на основі PROFINET;
- CPU 318-2 DP – потужний центральний процесор з високою швидкодією, більшим обсягом пам'яті програм і даних, здатний обслуговувати системи розподіленого введення – виведення на основі мережі PROFIBUS DP. Вимагає для своєї роботи карту пам'яті й буферну батарею.

Центральні процесори S7-300 Compact розрізняються в такому:

- CPU 312C – це компактний центральний процесор з 10 дискретними входами й 6 дискретними виходами, а також вбудованими функціями швидкісного рахунку (2х10 кГц) і виміру частоти (2х10 кГц) або тривалості періоду;
- CPU 313C – центральний процесор з 24 дискретними входами, 16 дискретними виходами, 4 аналоговими входами для виміру уніфікованих

сигналів сили струму або напруги, 1 аналоговим входом для підключення датчика температури Pt100 і 2 аналоговими виходами. Набір вбудованих функцій містить швидкісний рахунок (3×30 кГц), вимір частоти (3 × 30 кГц) або тривалості періоду, а також ПІД-регулювання;

- CPU 313 C-2 PtP та CPU 313 C-2 DP – центральні процесори з 16 дискретними входами, 16 дискретними виходами й додатковим інтерфейсами RS 422/ RS 485 (CPU 313 C-2 PtP) та вбудованим інтерфейсом провідного / веденого пристрою PROFIBUS DP (CPU 313 C-2 DP). Набір вбудованих функцій аналогічний CPU 313C;
- CPU 314 C-2 PtP та CPU 314 C-2 DP – центральні процесори з 24 дискретними входами, 16 дискретними виходами, 4 аналоговими входами для виміру уніфікованих сигналів сили струму або напруги, 1 аналоговим входом для підключення датчика температури Pt100, 2 аналоговими виходами й вбудованим послідовним інтерфейсом RS422/RS485 (CPU 314 C-2 PtP) та вбудованим інтерфейсом провідного – веденого пристрою PROFIBUS DP (CPU 314 C-2 DP). Набір вбудованих функцій містить у своєму складі швидкісний рахунок (4×60 кГц), вимір частоти (4×60 кГц) або тривалості періоду, ПІД-регулювання та позиціонування по одній осі.

Всі центральні процесори S7-300 Compact можуть використовуватися як готові системи автоматизації.

Для побудови систем керування в небезпечних виробництвах застосовуються процесори S7-300F:

- CPU 315 F-2 DP – центральний процесор з вбудованим інтерфейсом PROFIBUS DP для побудови F-систем із середніми вимогами до обсягу й швидкості виконуваних програм, а також обслуговуванню систем розподіленого введення – виведення;
- CPU 317 F-2 DP – центральний процесор з вбудованим інтерфейсом PROFIBUS DP для побудови F-систем з високими вимогами до обсягу й швидкості виконуваних програм.

Особливе місце посідають центральні процесори CPU 315 T-2 DP/ CPU 317 T-2 DP, які оснащені набором вбудованих входів і виходів, вбудованими інтерфейсами MPI/PROFIBUS DP і PROFIBUS DP/DRIVE, що підтримують функції позиціонування й керування переміщенням.

Центральні процесори S7-400

Програмувальні контролери SIMATIC S7-400 можуть комплектуватися 7 типами центральних процесорів. Центральні процесори відрізняються обчислювальною потужністю, обсягами пам'яті, кількістю вбудованих інтерфейсів й іншими параметрами.

Центральні процесори охоплюють розв’язання завдань різного рівня складності й класифікуються в такий спосіб:

CPU 412-1, CPU 412-2 – призначені для побудови невеликих систем керування й розв’язання завдань середнього ступеня складності;

CPU 414-2, CPU 414-3 – призначені для побудови систем керування середнього ступеня складності із програмами великого обсягу, швидкісним виконанням інструкцій та інтенсивним мережним обміном даними;

CPU 416-2, CPU 416-3 – призначені для побудови складних систем автоматичного керування зі складними алгоритмами оброблення інформації й інтенсивним мережним обміном даними;

CPU 417-4 – призначені для побудови найбільш потужних систем автоматичного керування. Потужні центральні процесори оснащені одним (CPU 41x-3) або двома (CPU 417-4) відсіками для установлення інтерфейсного субмодуля IF 964-DP і одержання додаткових інтерфейсів PROFIBUS DP.

Центральні процесори S7-400 характеризуються обсягами робочої пам'яті від 288 Кбайт в CPU 412-1 до 30 Мбайт в CPU 417-4, паралельним доступом до пам'яті програм і даних, а також високою швидкодією. Так, наприклад, час виконання логічної операції в CPU 417-4 становить 18 нс.

У всіх моделях центральних процесорів є вбудований комбінований інтерфейс MPI/PROFIBUS DP, який використовується для підключення до мереж MPI або PROFIBUS DP. Поряд із цим є додаткові інтерфейси:

- у CPU 41x-2 – один додатковий інтерфейс провідного / веденого пристрою PROFIBUS DP;
- у CPU 41x-3 – два додаткових інтерфейси провідного / веденого пристрою PROFIBUS DP;
- у CPU 417-4 – три додаткових інтерфейси провідного / веденого пристрою PROFIBUS DP;
- у CPU 41x-3 PN/DP – один додатковий інтерфейс PROFIBUS DP, інтерфейс PROFINET з 2-канальним комутатором, а також Web-сервер.

Центральні процесори підтримують від 32 до 64 активних комунікаційних з'єднань, одночасну роботу з декількома комунікаційними процесорами, виконання функцій шлюзового пристрою між різними промисловими мережами.

STEP 7 дозволяє робити настроювання комунікаційних інтерфейсів (установлення мережних адрес, режимів роботи, швидкостей передачі даних, комунікаційних з'єднань і т. п.), розподіл адресного простору введення – виведення (установлення адрес модулів), визначення розмірів областей пам'яті, а також установлення глибини діагностичного буфера й інше.

2.2 Інтерфейсні модулі (IM)

Інтерфейсні модулі IM 460-0 і IM 461-0 дозволяють створювати системи локального введення – виведення, у яких відстань від базового блоку до останньої стійки розширення не перевищує 5 м. IM 460-0 виконує функції передавача, а IM 461-0 – функції приймача. Модулі забезпечують підтримку P- і K-шин контролера, що дозволяє встановлювати в стійки розширення сигнальні, функціональні й комунікаційні модулі S7-400. Напруга живлення через сполучний кабель не передається, тому кожна стійка розширення повинна комплектуватися власним блоком живлення.

Передавальний інтерфейсний модуль IM 460-0 установлюється в базовий блок програмувального контролера S7-400. В одну монтажну стійку UR1, UR2 і CR2 може встановлюватися до 6, а в одну монтажну стійку CR3 до 2 модулів IM 460-0.

IM 460-0 оснащений двома вбудованими інтерфейсами для підключення ліній розширення. Кожний інтерфейсний модуль IM 460-0 дозволяє підключити до 4 стійок розширення. Під час установлення в базовий блок 6 інтерфейсних модулів IM 460-0 загальна кількість стійок розширення не повинна перевищувати 21.

Приймальний інтерфейсний модуль IM 461-0 установлюється в стійки розширення. Він може підключатися до інтерфейсного модуля IM 460-0 базового блоку контролера або попередньої стійки розширення. У кожну стійку розширення (UR1, UR2, ER1, ER2) установлюється тільки один інтерфейсний модуль IM 461-0.

Інтерфейсні модулі IM 460-1 і IM 461-1 дозволяють створювати системи локального введення – виведення програмувальних контролерів S7-400, у яких відстань від базового блоку до стійки розширення не перевищує 1,5 м. Інтерфейсний модуль IM 460-1 виконує функції передавача, IM 461-1 – функції приймача. Модулі забезпечують підтримку тільки P-шини контролера, що дозволяє встановлювати в стійки розширення тільки сигнальні модулі. Комунікаційні процесори й функціональні модулі в ці стійки встановлюватися не можуть.

Напруга живлення(= 5 В) передається в стійку розширення через сполучний кабель від блока живлення базового блоку контролера. Установлення власних блоків живлення в стійках розширення не потрібне. Струм навантаження ланцюга живлення може досягати 5 А.

Передавальний інтерфейсний модуль IM 460-1 установлюється в базовий блок програмувального контролера S7-400. В одну монтажну стійку може встановлюватися до 2-х модулів IM 460-1.

IM 460-1 оснащений двома вбудованими інтерфейсами для підключення ліній розширення. До кожного інтерфейсу може підключатися по одній стійці розширення.

Приймальний інтерфейсний модуль IM 461-1 установлюється в стійку розширення (UR1, UR2, ER1, ER2), що підключається до базового блоку контролера S7-400 через інтерфейсний модуль IM 460-1.

Інтерфейсні модулі IM 460-3 і IM 461-3 дозволяють створювати системи локального введення – виведення програмувальних контролерів S7-400, у яких відстань від базового блоку до останньої стійки розширення не перевищує 102 м. Інтерфейсний модуль IM 460-3 виконує функції передавача, IM 461-3 – функції приймача. Модулі забезпечують підтримку P- і K-шин контролера, що дозволяє встановлювати в стійки розширення не тільки сигнальні, але й функціональні та комунікаційні модулі.

Напруга живлення через сполучний кабель не передається, тому кожна стійка розширення повинна комплектуватися власним блоком живлення.

Передавальний інтерфейсний модуль IM 460-3 установлюється в базовий блок програмувального контролера S7-400. В одну монтажну стійку UR1, UR2 і CR2 може встановлюватися до 6, в одну монтажну стійку CR3 – до 2 модулів IM 460-3.

IM 460-3 оснащений двома вбудованими інтерфейсами для підключення ліній розширення. До кожного інтерфейсу може підключатися до 4 стійок розширення. Під час установлення в базовий блок 6 інтерфейсних модулів IM 460-3 загальна кількість стійок розширення не повинна перевищувати 21.

Приймальний інтерфейсний модуль IM 461-3 установлюється в стійки розширення. Він може підключатися до інтерфейсного модуля IM 460-3 базового блоку контролера або до інтерфейсного модуля IM 461-3 попередніх стійок розширення. У кожну стійку розширення (UR1, UR2, ER1, ER2) установлюється тільки один інтерфейсний модуль IM 461-3.

Інтерфейсні *модулі IM 467 і IM 467 FO* призначені для підключення програмувальних контролерів S7-400 до мережі PROFIBUS DP і використовуються для збільшення кількості мереж PROFIBUS DP, які обслуговуються одним програмувальним контролером.

На відміну від комунікаційних процесорів інтерфейсні модулі вимагають керування своєю роботою з боку центрального процесора. Тому збільшення кількості використовуваних модулів IM 467 і IM 467 FO супроводжується збільшенням навантаження на центральний процесор при обслуговуванні комунікаційних завдань.

У мережі PROFIBUS інтерфейсні модулі IM 467 і IM 467 FO забезпечують підтримку:

- протоколу PROFIBUS DP під час роботи в режимі провідного DP пристрою;
- функцій зв'язку із програматором PG і панеллю оператора OP.

За необхідності обидва протоколи можуть використовуватися паралельно.

У режимі провідних пристроїв PROFIBUS DP інтерфейсні модулі IM 467 і IM 467 FO забезпечують підтримку функцій синхронізації (SYNC), заморожування (FREEZE), а також сталості часу циклу мережі.

У контролерах S7-300 застосовуються інтерфейсні модулі IM 360, IM 361 і IM 365.

Інтерфейсний *передавальний модуль IM 360* установлюється в базову стійку S7-300 при максимальній відстані до стійки розширення 10 м.

Інтерфейсний *модуль IM 361* має такі характеристики:

- може виконувати функції приймального та передавального модуля;
- містить джерело живлення 24 В постійного струму, що виводить на задню шину S7-300 напругу силою струму 0,8 А;
- допускає максимальну відстань 10 м між IM 360 і IM 361 або між двома IM 361.

Інтерфейсний *модуль IM 365* має такі характеристики:

- конструктивне виконання – попередньо зібрана пара модулів для базової стійки і стійки розширення (стійки 0 і 1);

- постачений загальним джерелом живлення на 1,2 А, з яких до 0,8 А може бути використано в одній стійці;
- сполучний кабель довжиною 1 м, підключений постійно;
- IM 365 не продовжує комунікаційну шину в стійку 1, тобто робота функціональних модулів FM в стійці розширення не підтримується.

2.3 Комунікаційні процесори (CP)

Комунікаційний процесор CP 443-1 забезпечує можливість підключення програмувальних контролерів SIMATIC S7-400 до мережі Industrial Ethernet. Він оснащений вбудованим мікропроцесором і виконує автономне керування мережним обміном даними, розвантажуючи від цих завдань центральний процесор контролера.

CP 443-1 установлюється в монтажну стійку S7-400 та підключається до внутрішньої шини контролера через один роз'єм. Модуль може встановлюватися на будь-яке вільне посадкове місце. Комунікаційний процесор CP 443-1 має 15-полюсне гніздо з'єднувача D-типу з автоматичним перемиканням між інтерфейсами AUI- і ITP, а також гніздо RJ45 для підключення до мережі Industrial Ethernet з використанням технології FastConnect.

Передача даних здійснюється на транспортних рівнях 1...4 з урахуванням вимог міжнародних стандартів. Підтримується робота в комбінованому режимі з одночасною підтримкою транспортних протоколів ISO, TCP/IP і UDP.

Для контролю працездатного стану системи зв'язку на основі TCP-з'єднань може активізуватися функція відстеження часу передачі між кожним активним і пасивним партнером по зв'язку.

Комунікаційному процесору CP 443-1 привласнюється власна Ethernet-адреса, що дозволяє підключати його до мережі підприємства.

Під час роботи в комбінованому режимі CP 443-1 здатний підтримувати такі комунікаційні функції:

- PG/OP функції зв'язку, які забезпечують можливість дистанційного програмування всіх мережних S7 станцій;
- використання процедур S7 routing дозволяє організувати міжмережний обмін даними й забезпечити «прозорість» мережі;
- S7 функції зв'язку дозволяють організувати зв'язки між S7-300 і S7-400 (сервер і клієнт), пристроями людино-машинного інтерфейсу й комп'ютерами. У системах автоматизації SIMATIC S7-400 комунікаційні процесори CP 443-1 можуть використовуватися для побудови резервованих систем зв'язку на базі Industrial Ethernet;
- синхронізацію за датою й часом всіх мережних пристроїв, що підтримують виконання цієї функції;
- передавання даних з використанням інтерфейсів SEND/RECEIVE і FETCH/WRITE транспортного протоколу ISO. При цьому інтерфейс

SEND/RECEIVE забезпечує зв'язок між контролерами SIMATIC S7 і комп'ютерами, а функції FETCH/WRITE забезпечують прямий доступ до даних центрального процесора. Обсяг переданих даних може досягати 8 Кбайт.

Для контролерів SIMATIC S7-300 застосовується аналогічний модуль – комунікаційний процесор CP 343-1 V2.

Мережа Ethernet має високу пропускну здатність, однак істотним недоліком цієї мережі є те, що вона не гарантує час доставлення повідомлень. Цей недолік усунутий у мережі PROFINET.

Комунікаційний процесор CP 443-1 Advanced призначений для підключення програмувального контролера SIMATIC S7-400 до мережі PROFINET. Система розподіленого введення – виведення на основі PROFINET IO забезпечує роботу в реальному масштабі часу. Обмін даними між технологічними модулями в системах автоматизації з розподіленим інтелектом підтримується PROFINET CBA (Component Based Automation).

CP 443-1 Advanced дозволяє підтримувати зв'язок між SIMATIC S7-400 і такими пристроями:

- програматорами й комп'ютерами;
- засобами людино-машинного інтерфейсу;
- системами автоматизації SIMATIC S7/C7;
- контролерами PROFInet;
- приладами систем розподіленого введення – виведення на основі PROFINET й інтелектуальними модулями системи PROFINET CBA.

Комунікаційний процесор CP 443-1 Advanced виконує такі комунікаційні функції:

- HTTP-функції, що забезпечують доступ до даних контролера зі стандартного Web-браузера;
- FTP-функції, що дозволяють використовувати програмно-керований обмін даними FTP-клієнта, доступ до блоків даних через FTP-сервер, оброблення даних файлової системи через FTP, передавання повідомлень по каналах електронної пошти;
- установлення IP адреси через DHCP (Dynamic Host Configuration Protocol) з використанням інструментальних засобів комп'ютера або через програмний блок;
- захист доступу, що базується на використанні IP адреси;
- підключення до внутрішньої шини контролера через один роз'єм монтажної стійки;
- підключення до мережі за допомогою 4-х комутованих портів;
- інтеграцію в систему керування шляхом підтримки протоколу SNMPI.

Механічна конфігурація CP 443-1 Advanced характеризується такими особливостями:

- 4 гнізда RJ45 для підключення до Industrial Ethernet;
- знімний модуль пам'яті C-PLUG для збереження інформації (без модуля C-PLUG процесор CP 443-1 Advanced працювати не може).

Для контролерів SIMATIC S7-300 застосовується аналогічний модуль – комунікаційний процесор CP 343-1 Advanced.

Комунікаційний процесор CP 443-5 Basic призначений для підключення контролерів SIMATIC S7-400 до мережі PROFIBUS.

Він здатний підтримувати:

- функції FMS-зв'язку з PROFIBUS FMS-станціями;
- інтерфейс SEND/RECEIVE;
- функції зв'язку із програматором, пристроями й системами людино-машинного інтерфейсу;
- функції зв'язку з іншими системами автоматизації SIMATIC S7/C7;
- синхронізацію дати й часу всіх мережних станцій.

Для контролерів SIMATIC S7-300 є аналогічний модуль – комунікаційний процесор CP 343-5.

Комунікаційний процесор CP 443-5 Extended виконує функції ведучого DP пристрою й дозволяє одержувати від 4 до 10 додаткових ліній PROFIBUS DP на один базовий блок програмувального контролера.

Він здатний підтримувати:

- функції ведучого пристрою PROFIBUS DP відповідно до вимог міжнародних стандартів IEC 61158/ EN 50170;
- функції зв'язку із програматором, пристроями й системами людино-машинного інтерфейсу;
- функції зв'язку з іншими системами автоматизації SIMATIC S7/C7.

Для контролерів SIMATIC S 7-300 є аналогічний модуль – комунікаційний процесор CP 342-5.

Комунікаційні процесори CP 440 призначені для організації швидкісного обміну даними в послідовному форматі через PtP (Point-to-Point) інтерфейс.

PtP-інтерфейс дозволяє встановлювати зв'язок з такими партнерами:

- програмувальними контролерами SIMATIC S7, а також контролерами інших виробників;
- персональними комп'ютерами й програма торами;
- принтерами, сканерами, модемами й т. д.;
- вимірювальними приладами.

CP 440 оснащений вбудованим послідовним інтерфейсом RS422/RS485 (X.27), через який може підключатися до 31 пристрою.

Модуль CP 440 здатний підтримувати два стандартних протоколи обміну даними:

- ASCII – для організації найпростіших варіантів зв'язку із системами інших виробників;
- 3964(R) – для організації зв'язку із пристроями SIEMENS або апаратурою інших виробників, які підтримують обмін даними за протоколом 3964 (R).

Для контролерів SIMATIC S7-300 є аналогічний модуль – комунікаційний процесор CP 340.

Комунікаційні процесори CP 441 призначені для організації швидкісного обміну даними в послідовному форматі через PtP-інтерфейс.

Модуль випускається у двох модифікаціях:

- CP 441-1 з одним PtP портом для розв'язання простих комунікаційних завдань;
- CP 441-2 із двома PtP портами для побудови високопродуктивних систем зв'язку.

Комунікаційні процесори CP 441 здатні підтримувати найпоширеніші протоколи передачі даних:

- 3964 (R) – для зв'язку із приладами й пристроями виробництва фірми SIEMENS;
- RK 512 – для зв'язку з комп'ютерами (тільки CP 441-2);
- ASCII – для простого зв'язку з апаратурою різних фірм-виготовлювачів.

Для контролерів SIMATIC S7-300 є аналогічний модуль – комунікаційний процесор CP 341.

2.4 Функціональні модулі (FM)

Функціональні модулі призначені для розв'язання типових завдань автоматичного керування, до яких належать завдання швидкісного рахунку, позиціонування, автоматичного регулювання й інше. Більшість функціональних модулів наділено інтелектом, що дозволяє зменшити навантаження центрального процесора контролера.

Функціональні модулі містять:

- модуль швидкісного рахунку FM 450-1;
- модуль позиціонування FM 451;
- модуль електронного командоконтролера FM 452;
- модуль позиціонування крокових двигунів FM 453;
- модуль автоматичного регулювання FM 455;
- модуль розв'язання прикладних завдань FM 458-1DP.

Функціональний модуль FM 450-1 – це інтелектуальний 2-канальний модуль швидкісного рахунку. Він дозволяє робити:

- підрахунок імпульсів інкрементальних датчиків позиціонування;
- контролювати дискретні сигнали датчиків положення, наприклад, фотоелектронних бар'єрів;
- виконувати функції порівняння вмісту лічильників із заданими значеннями;
- видавати дискретні сигнали на вбудовані дискретні виходи.

Всі операції виконуються автономно, живлення датчиків здійснюється від вбудованого в модуль блока живлення.

FM 450-1 здатний обробляти сигнали двох інкрементальних датчиків позиціонування із частотами до 500 кГц. Напрямок рахунку задається зовнішніми імпульсними сигналами. Вплив на керований процес може здійснюватися двома способами:

1. Через вбудовані дискретні виходи, стан яких визначається результатами операцій порівняння поточних значень рахунку із заданими величинами. Для кожного лічильника може встановлюватися три величини: вихідний стан (попереднє установлення), верхнє й нижнє граничне значення рахунку;

2. Передачею сигналів у центральний процесор по внутрішній шині контролера з використанням механізму переривань.

Обидва лічильники можуть використовувати для своєї роботи два числових діапазони:

- нереверсивний рахунок – від 0 до +4294967295;
- реверсивний рахунок – від –2147483648 до +2147483647.

Інтелектуальний модуль позиціонування FM 451 застосовується для розв'язання завдань позиціонування по 3 осях із прискореною подачею робочого органа. Він здатний управляти роботою приводів, оснащених стандартними двигунами. Впливи на двигуни формуються контакторами або перетворювачем частоти. Поточні координати переміщення контролюються за допомогою інкрементальних або синхронно-послідовних (SSI) датчиків положення.

Модуль знаходить застосування в системах керування пакувальними машинами, ліфтами, конвеєрами, устаткуванням для деревооброблення й виробництва паперу, друкувальними машинами, устаткуванням для виробництва виробів з гуми й пластмас.

На рисунку 2.1 наведена типова схема застосування модуля позиціонування для розв'язання завдання циклічних рухів виконавчих органів.

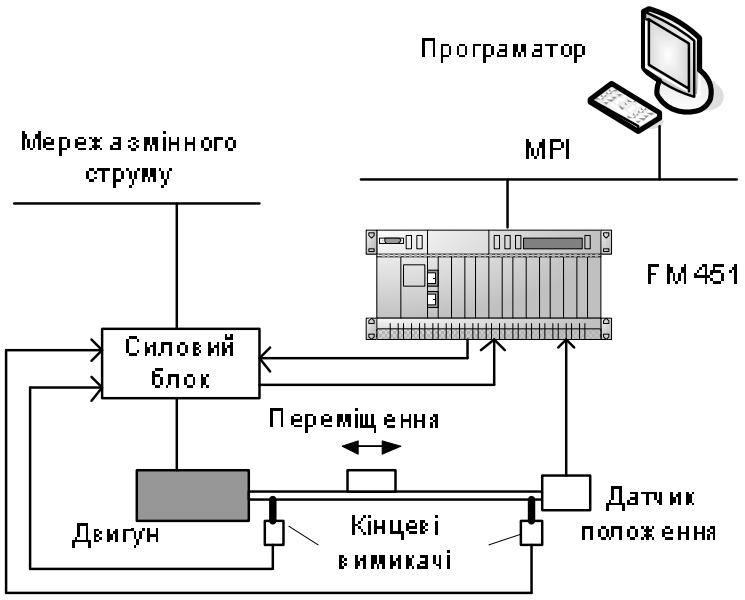


Рисунок 2.1 – Типова функціональна схема позиціонування

Кожний канал оснащений 4 дискретними виходами, які дозволяють управляти напрямком обертання двигуна, вибирати високу або низьку швидкість переміщення, робити запис координат поточної точки, дозволяти або забороняти роботу системи позиціонування.

Швидкість переміщення вибирається залежно від відстані до точки зупинки за сигналом датчика проходження деякої контрольної точки. Під час досягнення точки зупинки модуль перевіряє точність позиціонування з урахуванням заданих допусків і посилає повідомлення в центральний процесор.

Модуль електронного командоконтролера FM 452 призначений для формування послідовності команд за аналогією з кулачковим командоконтролером. Запуск послідовності операцій робиться за сигналом датчика положення, підключеного до входу модуля.

Модуль здатний працювати з інкрементальними й синхронно-послідовними датчиками позиціонування та дозволяє використовувати для формування команд до 32 кулачків, які впливають на стани 16 вбудованих дискретних виходів. Модуль знаходить застосування в системах керування свердлувальними й фрезерними верстатами, пресами й іншим устаткуванням.

На рисунку 2.2 показаний приклад застосування модуля командоконтролера FM 452 для керування приводом, який виконує складний рух у функції часу та шляху.

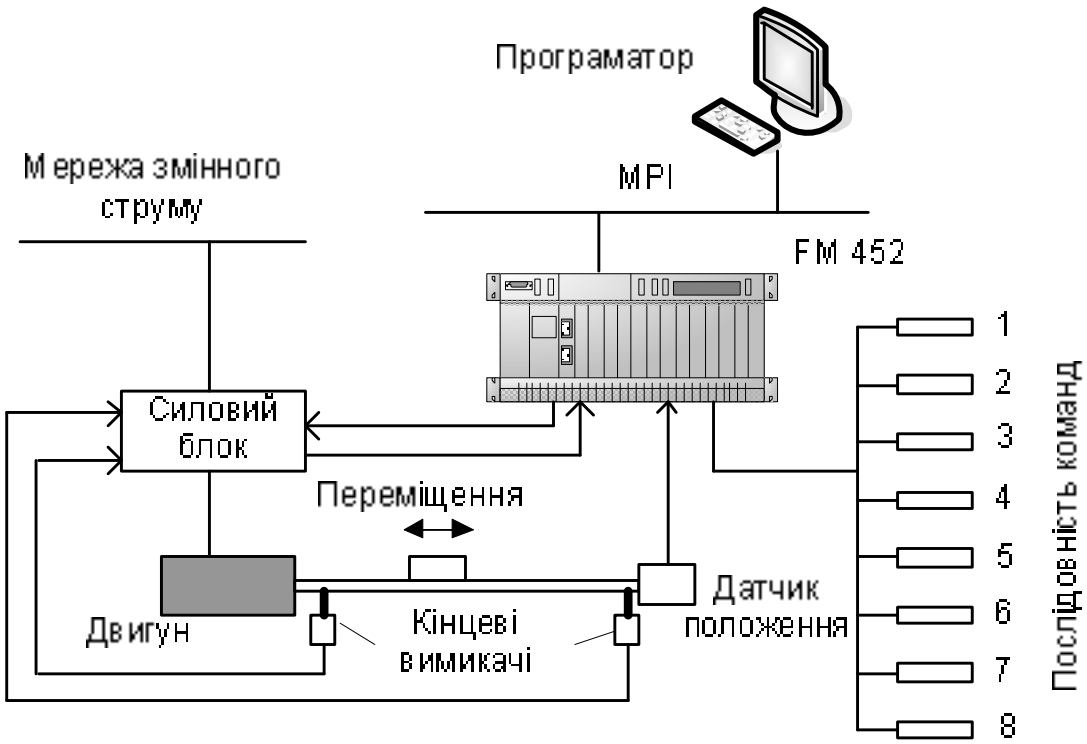


Рисунок 2.2 – Функціональна схема системи керування приводом за допомогою командоконтролера FM 452

Модуль забезпечує:

- вимір пройденого шляху;
- установлення контрольних точок;
- установлення поточних значень параметрів;
- зчитування миттєвих значень параметрів;
- зсув нуля;
- зміну керуючих фронтів;
- режим імітації.

Інтелектуальний модуль для керування кроковими двигунами FM 453 призначений для вирішення широкого кола завдань позиціонування електроприводів із кроковими та (або) серводвигунами: від простого покрокового позиціонування до складних комплексних завдань із високими вимогами до часу реакції, точності й швидкості позиціонування. До одного модуля може підключатися до трьох приводів.

Модуль знаходить застосування в системах керування машинами для виробництва паперу, текстильними й пакувальними машинами, типографськими верстатами, устаткуванням у харчовій промисловості, складальним устаткуванням.

У випадку простого переміщення від точки до точки задаються кінцева точка позиціонування й швидкість переміщення. Для більш складних завдань створюється програма переміщення, що за допомогою програматора вводиться у FM 453. Можливе програмування в режимі навчання. Параметри настроювання зберігаються в пам'яті модуля FM 453. Ці дані містять відомості про параметри машин, необхідні компенсації, програми керування рухом або опис кроків руху від точки до точки.

Для виконання завдань позиціонування FM 453 може формувати:

- аналогові сигнали $\pm 10\ B$ для керування роботою електроприводів із серводвигунами;
- імпульси керування електроприводами із кроковими двигунами, а також сигнал вибору напрямку обертання.

Контроль процесу позиціонування здійснюється за допомогою синхронно-послідовних (SSI) або інкрементальних датчиків позиціонування. У приводах із кроковими двигунами датчики позиціонування можуть не застосовуватися.

Функціональний модуль FM 455 є універсальним інтелектуальним 16-канальним регулятором, що застосовується для вирішення широкого кола завдань автоматичного регулювання. На його основі можуть бути побудовані системи регулювання температури, тиску, потоку й інших параметрів. Модуль випускається у двох модифікаціях:

- FM 455C – для побудови систем автоматичного регулювання з аналоговими виконавчими пристроями, що підключаються до 16 аналогових виходів модуля;
- FM 455S – для побудови систем імпульсного регулювання із впливом на процес через 32 вбудовані дискретні виходи.

Обидва модулі дозволяють створювати програмні структури автоматичного регулювання й використовувати інтерактивну систему адаптації систем регулювання температури. Регулятори, побудовані на основі FM 455, здатні продовжувати свою роботу навіть у випадку зупинки центрального процесора контролера.

Модуль FM 455 здатний виконувати такі функції:

- здійснювати регулювання із використанням готових структур: регулятора з фіксованим настроюванням, системи каскадного регулювання, регуляторів пропорційної дії, а також систем трикомпонентного регулювання;
- реалізувати різні режими роботи – автоматичний, ручний, а також режими захищеного й безпечного керування;
- реалізувати регульований крок квантування, що залежить від розрядності перетворення – для 12-розрядного перетворення від 20 до 180 мс, для 14-розрядного перетворення – від 100 до 1 700 мс (визначається кількістю використовуваних аналогових каналів);
- здійснювати оптимальне адаптивне регулювання температури, а також ПІД-регулювання у системах, у яких не спостерігається великих відхилень регульованого параметра від заданих значень, наприклад, систем автоматичного регулювання парових казанів, ливарних машин і т. д.

2.5 Цифрові модулі введення - виведення (SM)

Цифрові модулі введення

Основні параметри сигнальних цифрових модулів введення SM321 (SM421 для контролерів S7-400) присутні в їхніх позначеннях. Так, наприклад, *SM 321; DI 32*x*24 VDC* являє собою модуль *введення* (позначення DI) на 32 канали для підключення датчиків з напругою живлення 24 *В постійного струму* (позначення VDC), а *SM 321; DI 8*x*120/230 VAC* – модуль введення на 8 каналів для підключення датчиків з номінальною напругою 120/230 В *змінного струму* (позначення VAC).

Датчики підключаються до модулів введення кабелем. Довжина неекранованого кабелю може досягати 600 м, а екранованого – 1000 м.

Всі модулі придатні для підключення двохпроводних датчиків типу перемикачів і трьохпроводних датчиків безконтактної дії типу BERO. Модулі відрізняються схемою з'єднання датчиків із задньою шиною модуля. Задня шина – це послідовна шина даних, через яку модулі обмінюються даними один з одним і через яку вони одержують живлячу напругу.

Найпростішим варіантом з'єднання є резистивний дільник у сполученні з оптопарою (рис. 2.3).

Модулі введення, призначені для роботи із датчиками, які живляться від джерел змінного струму, мають трохи іншу схему дільника. Тут застосовується резистивно-ємнісний дільник і випрямний міст (рис. 2.4).

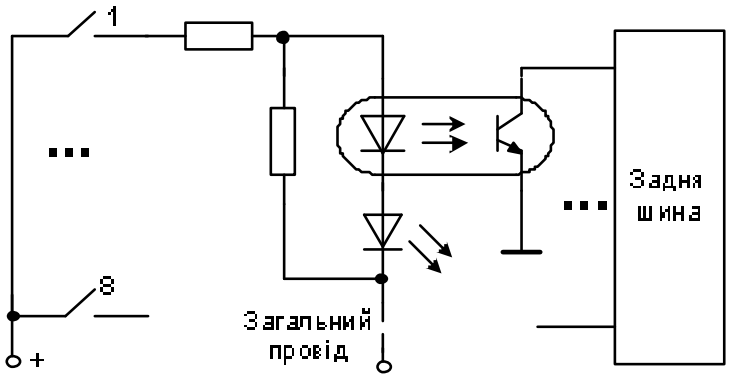


Рисунок 2.3 – Схема з'єднання датчика із задньою шиною через резистивний дільник і оптопару

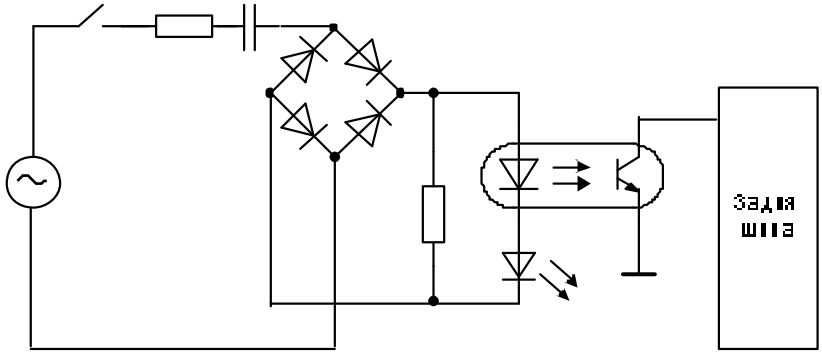


Рисунок 2.4 – Схема з'єднання датчика з модулем при живленні датчика напругою змінного струму

У системах автоматизації SIMATIC передбачені модулі для різних рівнів сигналів логічного нуля й логічної одиниці.

Наприклад, у модулі *SM 321; DI 32*x*24 VDC* (номінальна напруга 24 *В* постійного струму) значенню сигналу «1» відповідає напруга від 13 до 30 *В*, а значенню сигналу «0» – від мінус 30 до плюс 5 *В*.

Входи модулів можуть бути як ізольованими (*SM 321; DI 16*x*24/48 VUC*), коли кожний датчик підключається до модуля зі своїм джерелом живлення, так і згрупованими, коли живлення подається на групу датчиків (4, 8 або 16 одиниць).

Групове живлення датчиків дозволяє скоротити витрати дротів для підключення датчиків.

Цифрові модулі виведення

У цифрових модулях виведення SM 322 або SM 422 із вихідними ланцюгами *постійного* струму передача сигналу із задньої шини модуля в ланцюги навантаження здійснюється через оптронну пару діод-транзистор і підсилювач постійного струму, який повинен одержувати живлення від зовнішнього джерела. Модулі розрізняються напругою живлення, припустимою силою струму в ланцюгах навантаження і частотою зміни вихідного сигналу.

Напруги живлення джерел постійного струму перебувають у діапазоні від 24 *В* до 125 *В*.

Модулі з вихідними ланцюгами постійного струму застосовують для підключення електромагнітних клапанів, контакторів постійного струму й сигнальних ламп. Максимальна частота перемикання навантаження становить 100 Гц для активного навантаження і 0,5 Гц для індуктивного навантаження.

Модулі із вихідними ланцюгами на *змінному* струмі застосовуються для керування електромагнітними вентилями, контакторами, пускачами, двигунами малої потужності та сигнальними лампами.

У модулях виведення змінного струму передача сигналів від задньої шини в ланцюг навантаження здійснюється через оптопару діод-семістор. Така схема сполучення дозволяє комутувати напруги до 230 *В* змінного струму із навантаженням до 2 А (SM 322; DO 8x120/230 VAC/2 A).

У сімействі модулів виведення є також модулі з релейним виходом. У цих модулях у ланцюг колектора транзистора оптопари включено реле, контакти якого використовуються для керування зовнішніми реле або магнітними пускачами. Така схема дозволяє використовувати для живлення виконавчих пристроїв джерела постійного та змінного струму.

Цифрові модулі введення – виведення SM 323

До цієї групи належать три модулі:

1. SM 323; Dl 16/DO 16x24 VDC/0.5 A – модуль на 16 входів (одна група) і 16 виходів (дві групи);

2. SM 323; Dl 8/DO 8x24 VDC/0.5 A – модуль на 8 входів і 8 виходів з потенційною розв'язкою групами по входам і виходам;

3. SM 327; Dl 8/DO 8x VDC 24/0.5 A – модуль на 8 входів (одна група) і 8 виходів з розв'язкою по кожному виходу.

Всі модулі введення – виведення розраховані на підключення до джерел постійного струму напругою 24 В.

2.6 Аналогові модулі введення – виведення (SM)

Аналогові модулі введення SM 331/SM 431

Аналогові модулі введення SM 331 у своєму позначенні містять інформацію про кількість каналів і кількість розрядів цифрового коду. Так, наприклад, модуль SM 331; AI 8×16 Bit має 8 каналів введення аналогового сигналу з перетворенням в 16-розрядний код. Старший розряд коду зазвичай використовується для позначення знака сигналу, тому числове значення буде мати на один розряд менше.

Канали зазвичай групуються по 4 або по 8. Залежно від типу модуля кожна група каналів може настроюватися на вимір струму, напруги, опору або температури. На рисунку 2.5, як приклад, показана функціональна схема підключення датчиків до модуля SM 331; AI 8×14 Bit; High Speed.

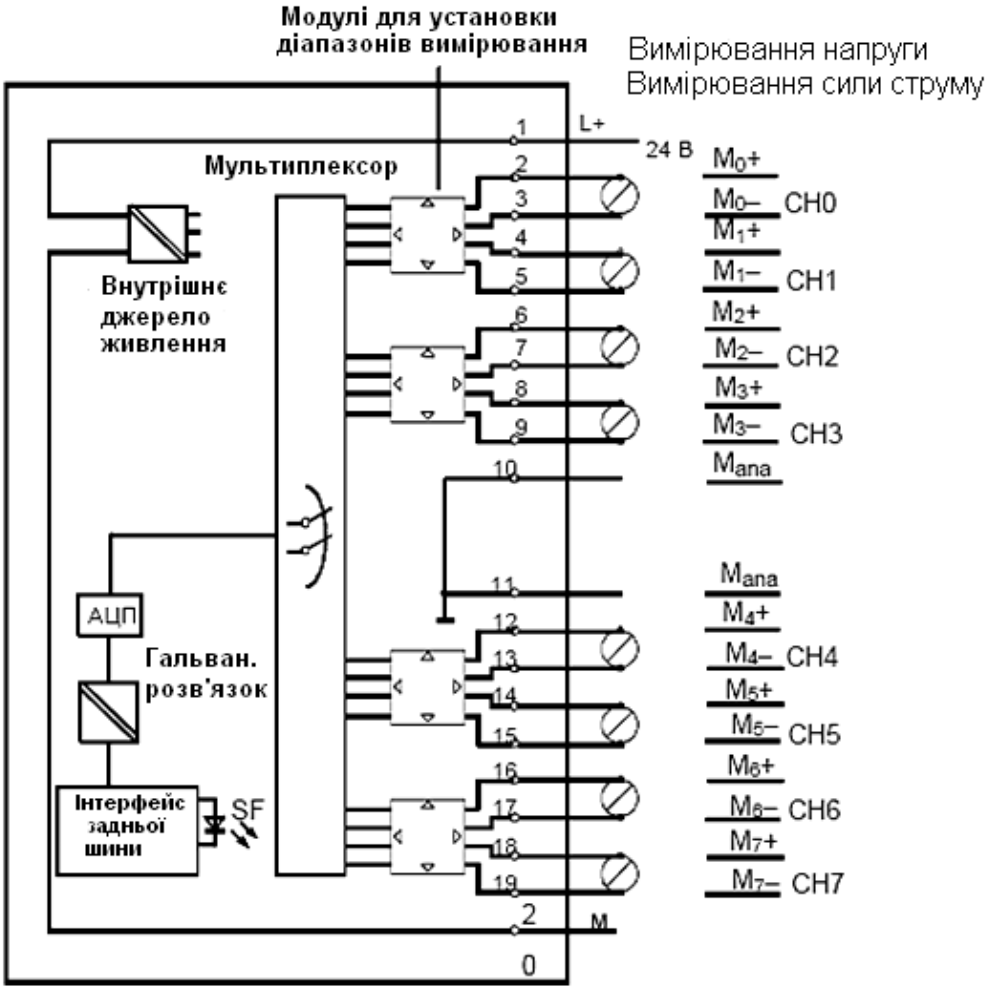


Рисунок 2.5 – Функціональна схема підключення модуля SM 331 AI 8

Для установлення діапазону виміру використовуються *спеціальні* модулі настроювання діапазонів, які являють собою конфігураційну вставку. Процес установлення діапазону показано на рис. 2.6.

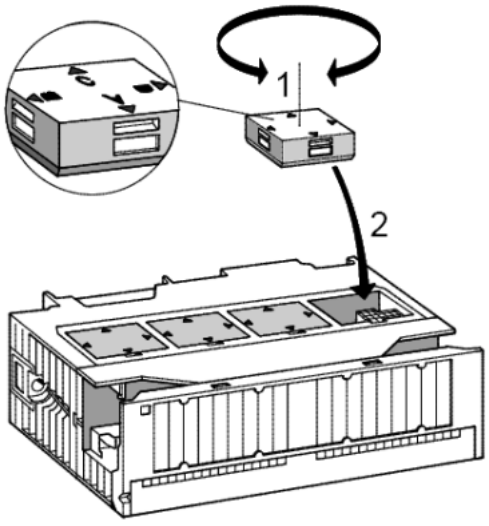


Рисунок 2.6 – Установка діапазону вимірів у модулю аналогового введення

Крок 1. За допомогою викрутки модуль установки діапазону вимірів виймається із аналогового модуля.

Крок 2. Модуль для установки діапазону вимірів позиціонується (1) щодо гнізда аналогового модуля так, щоб зазначений на ньому діапазон був спрямований на мітку в корпусі аналогового модуля.

Крок 3. Модуль для установки діапазону вимірів вставляється в аналоговий модуль введення (2).

Модуль установки має чотири положення: A, B, C, D. Положення A и B дозволяють установити діапазон вимірюваних напруг (± 1 В, ± 5 В, ± 10 В та 1...5 В), а положення C і D – призначити діапазони виміру струмів (± 20 mA, 0...20 mA та 4...20 mA). Під час використання чотирипровідного вимірювального перетворювача струму модуль установки повинен перебувати у положенні C.

Для виміру напруги використовуються двопровідні лінії підключення датчиків. Датчик напруги підключається з дотриманням полярності. На наведеній нижче схемі (рис. 2.7) L+ позначає позитивну клему джерела живлення модуля аналогового введення (24 В), а M+ позначає позитивний провід схеми вимірювання.

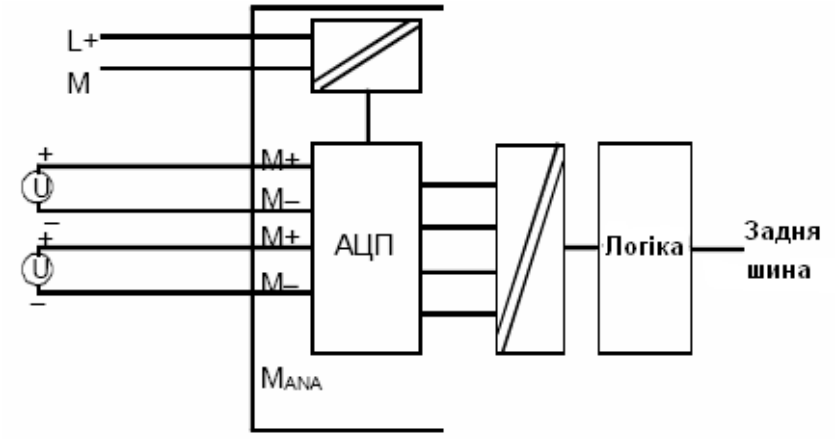


Рисунок 2.7 – Схема двопровідного підключення датчиків напруги

Трипровідне або чотирипровідне підключення варто застосовувати з метою підвищення точності вимірів опорів і температури. Під час чотирипровідного підключення модуль подає у схему вимірювання через клеми IC+ і IC– струм постійної величини, завдяки чому компенсується спадання напруги, що виникає на сполучних кабелях. Важливо, щоб сполучні кабелі зі струмом постійної величини були безпосередньо підключені до термометра опору або резистора.

Приклад схеми чотирипровідного з'єднання термометра опору з модулем вводу показаний на рисунку 2.8.

Для підключення термопар використовуються модулі, які підтримують роботу з необхідним типом термопари. Це пов'язано з необхідністю компенсації температури холодного спаю, характер якої залежить від типу термопари.

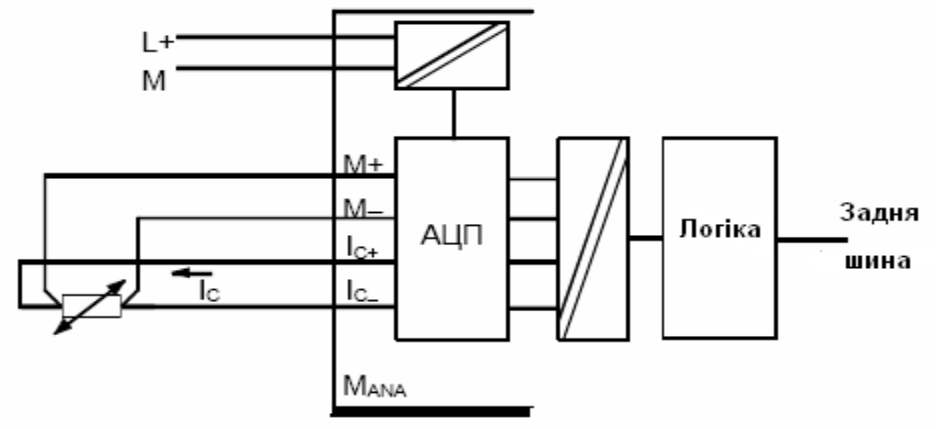


Рисунок 2.8 – Схема чотирипровідного підключення до аналогового модуля термометра опору

Якщо вимірювальний спай термопари (рис. 2.9) піддається дії температури, відмінної від температури вільних кінців термопари (точка підключення), то між вільними кінцями виникає напруга, або термо-ЕРС (термоелектрорушійна сила). Величина термо-ЕРС залежить від різниці між температурами вимірювального спаю та «холодного спаю» (вільних кінців), а також від комбінації матеріалів термопари.

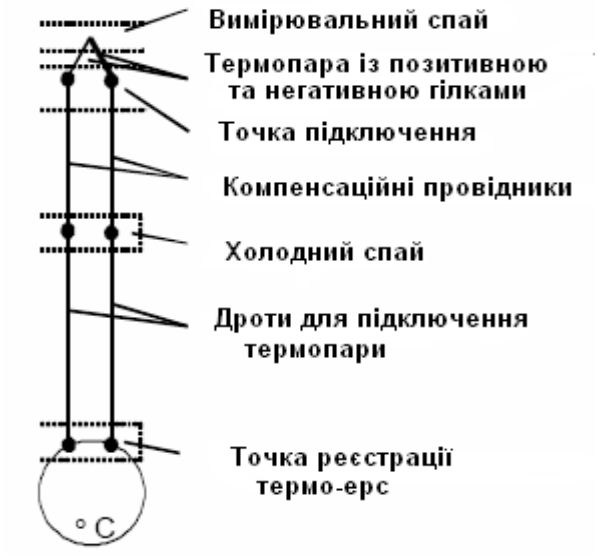


Рисунок 2.9 – Схематичне подання термопари

Враховуючи те, що термопара завжди вимірює різницю температур, то вільні кінці повинні втримуватися при відомій температурі, наприклад 0°С, щоб можна було визначати температуру вимірювального спаю.

Термопари можуть бути подовжені від точки їхнього підключення до точки з відомою температурою (температурою холодного спаю) за допомогою *компенсаційних* дротів. Ці компенсаційні дроти виконуються з того ж матеріалу, що й дроти термопари. В якості дротів з'єднання застосовуються мідні провідники.

Для компенсації температури холодного спаю (установки початку відліку температури, тобто 0°С) застосовуються два способи – внутрішня компенсація й зовнішня компенсація.

Внутрішня компенсація будується за таким принципом: компенсаційні дроти підключаються до модуля аналогового введення і холодний спай, таким чином, опиняється усередині модуля та його температура може бути виміряна за допомогою термістора.

Зовнішня компенсація здійснюється за допомогою спеціального зовнішнього компенсаційного блока, до якого підключаються компенсаційні дроти всіх термопар (рис. 2.10). Компенсаційний блок використовується для *стабілізації* температури холодного спаю. Підключення компенсаційного блоку до модуля здійснюється мідними дротами.

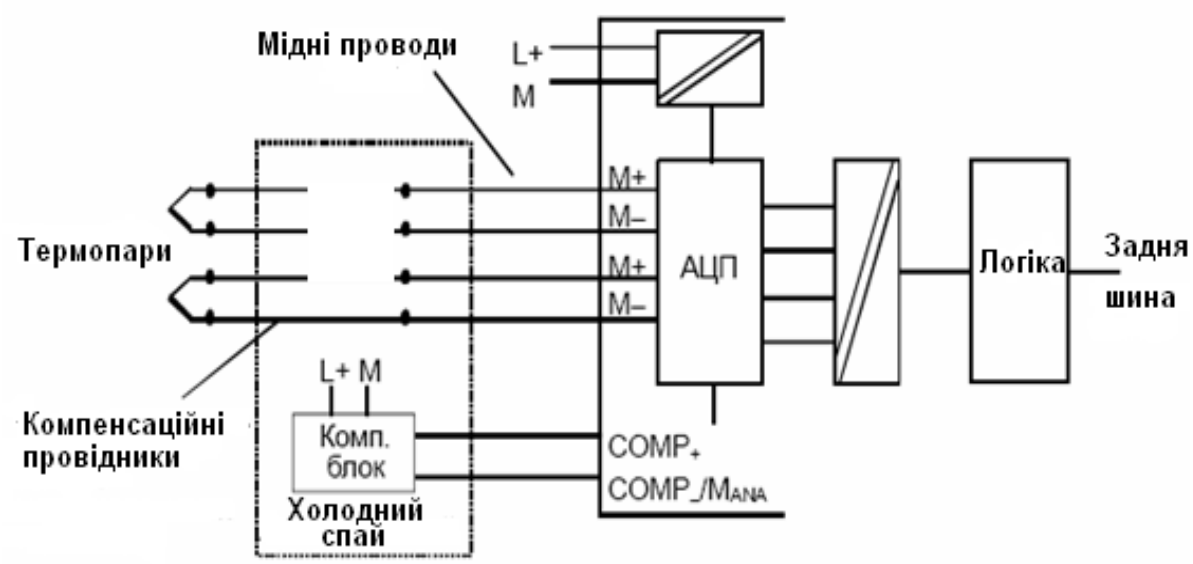


Рисунок 2.10 – Підключення термопар з компенсаційним блоком

При підключенні датчиків довжина неекранованого мідного кабелю може досягати 50 м, а екранованого – 200 м.

Основний час перетворення аналогового сигналу в цифровий залежить від типу модуля й становить 10…100 мс. Потім модуль повинен перемкнутися на інший канал у групі за допомогою оптичного МОП-реле. На це перемикання потрібно близько 12 мс. Тому основний час реакції модуля, що визначає цикл його опитування, перебуває в межах 24…816 мс. У цілому час реакції модуля залежить від часу інтегрування, що встановлюється при параметризації модуля.

Для придушення синфазних перешкод на частоті 50 Гц рекомендується встановлювати час інтегрування 190–200 мс.

Деякі модулі аналогового введення дозволяють робити діагностику модуля, а також набудовувати апаратні переривання при виході вимірюваної величини за встановлені межі.

Модулі аналогового виведення SM 332/SM 432

До модулів аналогового виведення належать:

1. Аналоговий модуль виведення SM 332; AT 8x12 Bit забезпечує дво- та чотирипровідне підключення навантаження на 8 каналів виведення;

2. Аналоговий модуль виведення SM 332; AT 4x16 Bit забезпечує тактову синхронізацію (4 канали виведення);

3. Аналоговий модуль виведення SM 332; AT 4x12 Bit забезпечує можливість параметризації каналів (потенційні або струмові виходи);

4. Аналоговий модуль виведення SM 332; AT 2x12 Bit забезпечує роздільну параметризацію й можливість чотирипровідного підключення навантаження.

Аналогові модулі введення – виведення SM 334

До аналогових модулів введення – виведення належать два модулі:

1 Аналоговий модуль введення – виведення SM 334; AI 4/AT 2x8/8 Bit призначений для виміру та виведення напруги або струму;

2 Аналоговий модуль введення – виведення SM 334; AI 4/AT 2x12 Bit призначений для виміру напруг, опорів і температури (4 канали), а також для виведення напруги (2 канали).

Контрольні питання

1. Чим відрізняються центральні процесори CPU контролера S7-300?
2. Які типи центральних процесорів застосовуються в небезпечних виробництвах?
3. Як поділяються центральні процесори контролерів S7-400?
4. Якими додатковими інтерфейсами забезпечуються центральні процесори контролерів S7-400?
5. Які функції реалізують інтерфейсні модулі IM 460-x і IM 461-x?
6. Для чого призначені інтерфейсні модулі IM 467 і IM 467 FO?
7. Які функції виконують інтерфейсні модулі IM 360, IM 361 і IM 365?
8. Які можливості забезпечують комунікаційні процесори CP 443-1?
9. Які функції реалізують комунікаційні процесори?
10. Які завдання можна вирішувати за допомогою функціональних модулів FM 450-1, FM 451, FM 452 та FM 453?
11. Чим відрізняються цифрові модулі введення, як вони позначаються?
12. Чим відрізняються цифрові модулі виведення, як вони позначаються?
13. Чим відрізняються аналогові модулі введення, як вони позначаються?
14. Чим відрізняються аналогові модулі виведення, як вони позначаються?
15. Як установлюється діапазон виміру в аналогових модулях введення?
16. Які переваги забезпечує трипровідне й чотирипровідне підключення датчиків до аналогових модулів введення?
17. Як здійснюється компенсація температури холодного спаю при застосуванні термопари?

3 ПРОЕКТУВАННЯ ДЕЦЕНТРАЛІЗОВАНОЇ ПЕРИФЕРІЇ

3.1 Правила проектування децентралізованої периферії

Послідовність проектування периферії

Можливість створення комунікацій у розподіленій периферії визначається якістю конфігурування мережі.

Конфігурування мережі здійснюється в такій послідовності:

1. Створюється графічне зображення мережі, що складається з однієї або декількох підмереж;
2. Установлюються властивості й параметри для кожної під мережі;
3. Установлюються властивості й параметри для кожного вузла, що включається в мережу;
4. Установлюються властивості й параметри для кожного модуля, що включається у вузол;
5. Створюється документація з конфігурації мережі.

Варіанти проектів PROFIBUS-DP

Мережа PROFIBUS-DP забезпечує стандартний інтерфейс для передавання переважно двійкових даних процесу між інтерфейсним модулем у центральному програмувальному контролері та приладами польового рівня. При цьому *провідним* DP-пристроєм (DP-master) є процесорний або інтерфейсний модуль центрального контролера, а прилади польового рівня називаються *веденими* DP-пристроями (DP-slave).

Провідний DP-пристрій і всі керовані їм ведені DP-пристрої утворюють *систему провідного DP-пристрою* – майстер-систему (DP-master system).

При проектуванні PROFIBUS-DP можливі такі варіанти проектних рішень:

- проект із *простим* веденим DP (модульним або компактним), що забезпечує обмін даними за схемою Slave ⇔ Master;
- проект із *інтелектуальним* веденим DP, що забезпечує обмін даними за схемою Slave ⇔ Master;
- проект із *інтелектуальними* веденими DP і прямим обміном даними Slave ⇨ ISlave;
- проект із *двома майстер-системами* DP і прямим обміном даними Slave ⇨ Master;
- проект із *двома майстер-системами* DP і прямим обміном даними Slave ⇨ ISlave.

Розглянемо особливості цих проектів.

Проект із простим (модульним або компактним) веденим DP і обміном даними за схемою Slave ⇔ Master

У цій конфігурації (рис. 3.1) обмін даними між провідним DP і простими веденими DP, наприклад, станцією введення – виведення, відбувається під керуванням майстра DP.

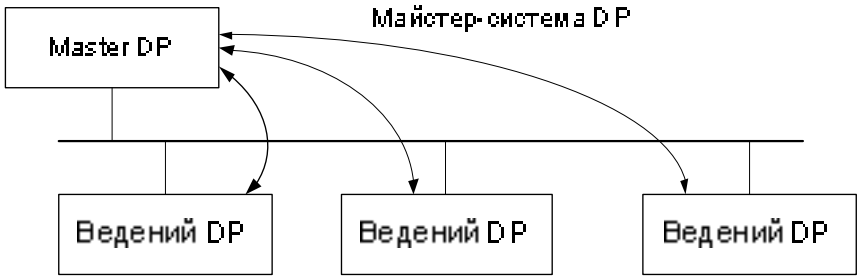


Рисунок 3.1 – Структура майстер-системи із простими модулями DP

Майстер DP послідовно опитує ведені DP, *конфігуровані в списку майстер-системи DP*. Наприкінці робочого циклу майстер DP здійснює передачу даних на ведені DP, внесені в список майстер-системи. Адреси входів і виходів ведених DP призначаються при конфігуруванні системи автоматично. Ця конфігурація відома як *мономастер-система*, тому що до однієї фізичної підмережі PROFIBUS DP підключений один майстер зі своєю системою ведених DP.

Проект із інтелектуальним веденим DP і обміном даними за схемою ISlave ⇔ Master

Завдання автоматизації можна розділити на окремі задачі, які можна реалізувати на інтелектуальних ведених DP. Приклад структури такої системи показано на рисунку 3.2.

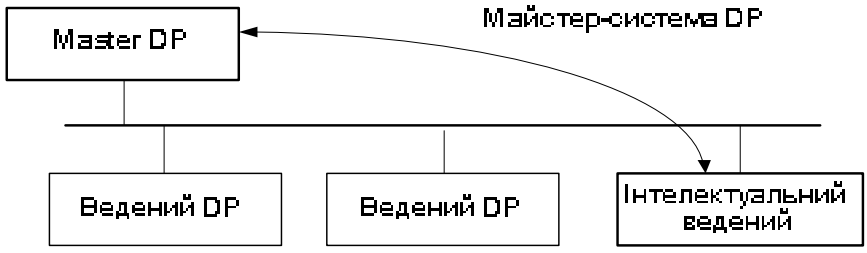


Рисунок 3.2 – Структура майстер-системи з інтелектуальним веденим DP і схемою обміну даними ISlave ⇔ Master

Особливість проекту полягає в тім, що при конфігуруванні інтелектуальних ведених DP (I-slaves), таких як CPU 315-2DP, кожному вхідному й вихідному модулю повинна бути визначена *адресна область*. Тому майстрові DP призначається тільки *визначення області* адресного простору веденого модуля. При обміні даними із майстром DP інтелектуальний модуль звертається до цієї області, а при реалізації задачі – до відповідних областей власної системної пам'яті.

Проект із інтелектуальними веденими DP і прямим обміном даними за схемою Slave ⇨ I Slave

У цій конфігурації (рис. 3.3) вхідні дані з ведених DP можуть бути дуже швидко передані інтелектуальним веденим DP.

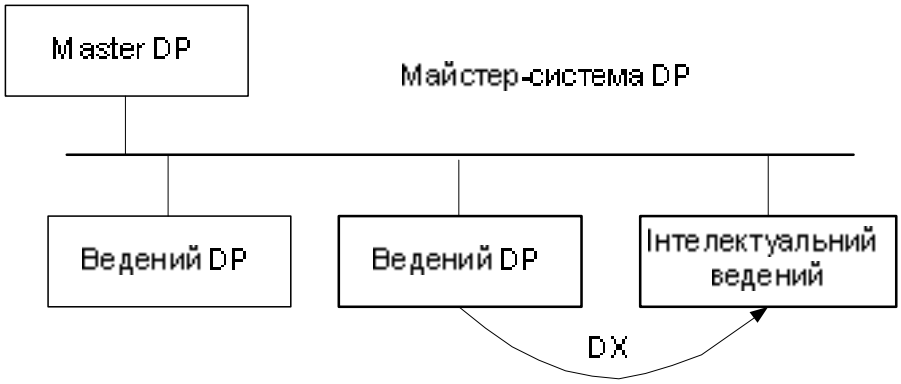


Рисунок 3.3 – Структура майстер-системи з інтелектуальним веденим DP і схемою обміну даними Slave ⇨ I Slave

Цим способом, у принципі, можна забезпечити й пряме передавання даних (DX) *між простими веденими* DP. Однак для реалізації проекту повинні бути використані тільки такі ведені DP, як CPU 315-2DP.

Проект із двома майстер-системами DP і прямим обміном даними за схемою Slave ⇨ I Slave

Якщо на одній фізичній підмережі PROFIBUS-DP є декілька майстрів DP, то така система називається *мультимастерною системою*. У цій конфігурації (рис. 3.4) інтелектуальні ведені DP, такі як CPU 315-2DP, передбачають пряме передавання у свою область вхідних даних з веденого DP, навіть якщо вони перебувають в інших майстер-системах DP.

Проект із двома майстер-системами DP і прямою передачею даних Slave ⇨ Master

У мультимастерних системах однієї фізичної підмережі PROFIBUS-DP вхідні дані з інтелектуального або простого веденого DP можуть бути безпосередньо прочитані ведучим DP іншої майстер-системи (рис. 3.5).

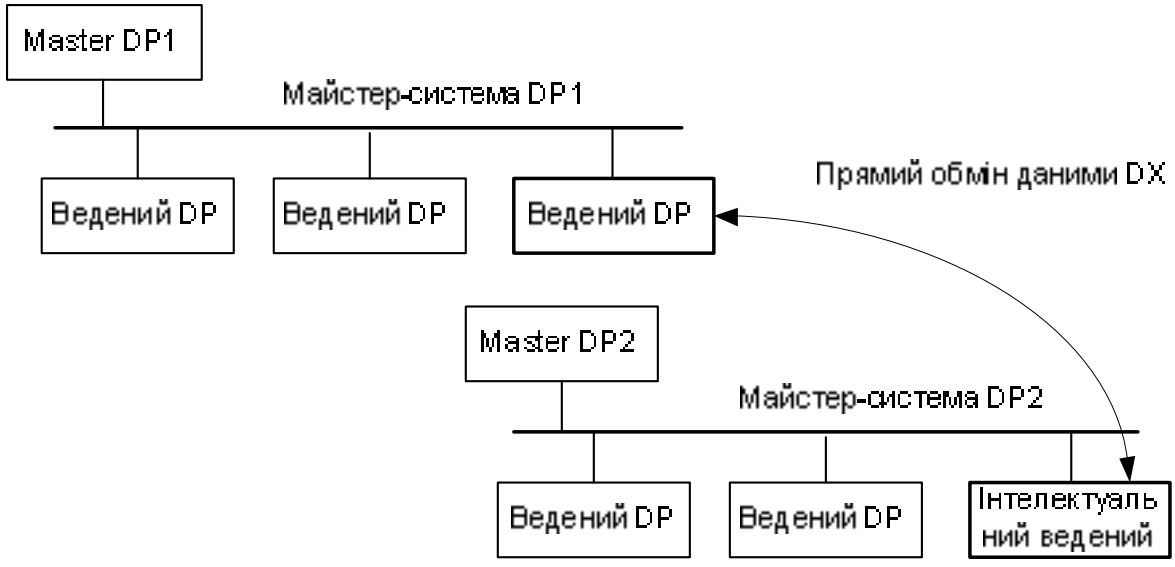


Рисунок 3.4 – Структура мультимастерної системи

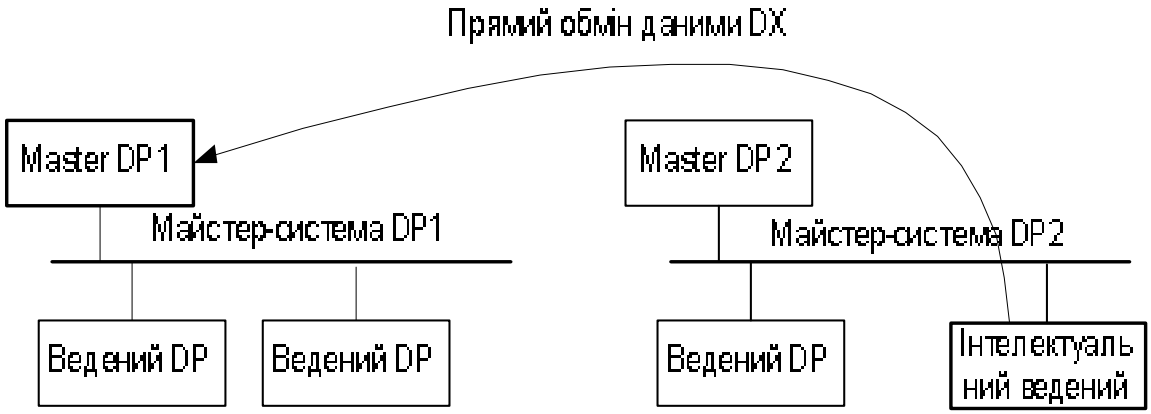


Рисунок 3.5 – Структура мультимастерної системи із прямою передачею даних Slave ⇨ Master

Цей механізм називається «загальний вхід», оскільки вхідні дані використовуються за межами деякої майстер-системи DP.

3.2 Проектування розподіленої периферії в мережі PROFIBUS-DP

Засоби мережі PROFIBUS-DP

PROFIBUS-DP – це протокол розподіленої периферії (Decentralized Peripheral), орієнтований на забезпечення швидкісного обміну даними між провідними DP-пристроями й веденими DP-пристроями. Протокол характеризується високою стійкістю до впливу зовнішніх електромагнітних полів. Він розроблений для високошвидкісних і недорогих систем. Швидкість передачі даних може знаходитись у межах від 9,6 кбіт/с до 12 Мбіт/с.

За електричними параметрами інтерфейс PROFIBUS відповідає параметрам RS-485, але відрізняється наявністю мережних карт, у яких використовується двохпортова рефлексивна пам'ять, що дозволяє пристроям обмінюватися даними без завантаження процесора контролера.

Провідному DP-пристрію сегмента мережі відповідає певна кількість керованих ним ведених DP-пристроїв. В одному сегменті мережі може бути до 32 станцій, у всій мережі допускається включення до 127 станцій.

Найбільше поширення одержали системи PROFIBUS-DP з одним провідним DP-пристроєм (mono master system). Практичний приклад такої системи показаний на рисунку 3.6.

Варіант мережі PROFIBUS-DP з декількома провідними DP-пристроями (multi master system) гірше тому, що він створює проблему доступу – у той час, коли один провідний DP-пристрій ініціює ведені DP-пристрої, інший ведучий DP-пристрій не має до них доступу.

Провідний DP-пристрій (DP Master) є активним вузлом мережі PROFIBUS. Цей пристрій обмінюється даними з веденими DP-пристроями циклічно.

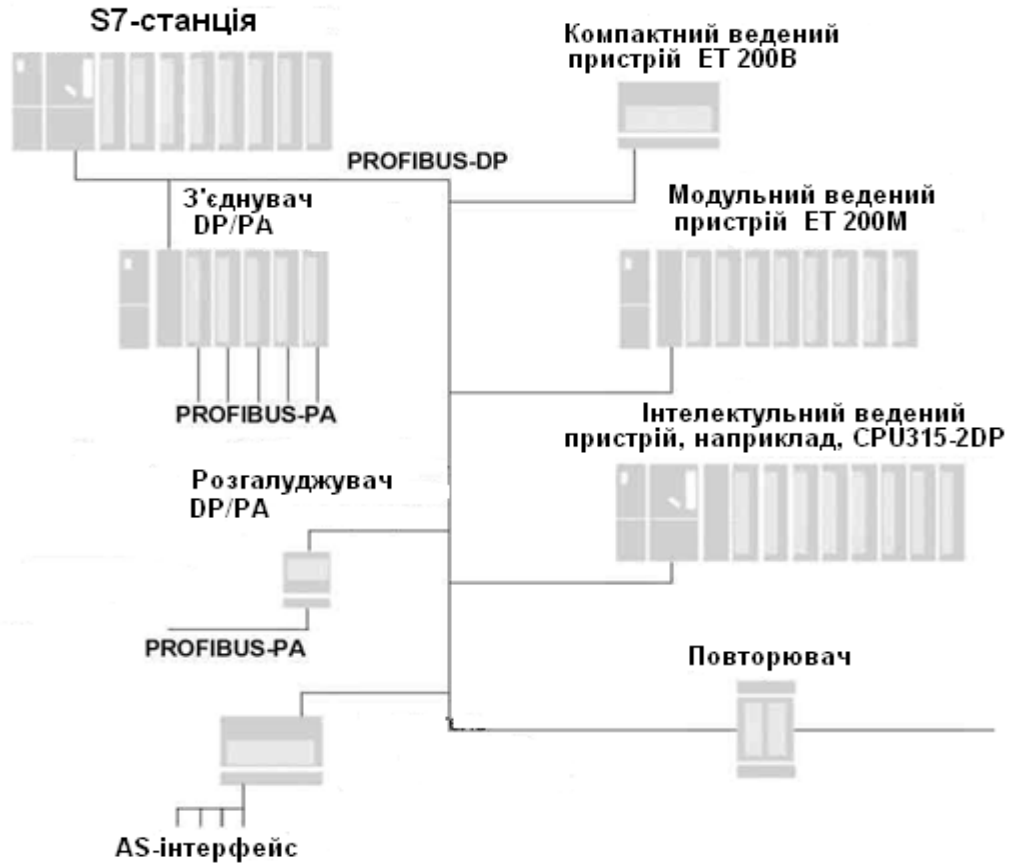


Рисунок 3.6 – Приклад системи з одним провідним DP-пристроєм

Провідним DP-пристроєм може бути:

- CPU, наприклад, CPU 315-2DP або CPU 417 з убудованим інтерфейсом провідного DP-пристрою або із вставленим інтерфейсним модулем;
- інтерфейсний модуль, наприклад, IM 467 у поєднанні з CPU;
- комунікаційний процесор, наприклад, CP 342-5 або CP 443-5 у поєднанні з CPU.

Існують провідні DP-пристрої 1 класу, призначені для обміну даними під час оброблення процесу, і провідні DP-пристрої 2 класу, призначені для обслуговування й діагностики, наприклад, програматори.

Ведені DP-пристрої (DP Slaves) є пасивними вузлами мережі PROFIBUS. У SIMATIC S7 розрізняють такі ведені DP-пристрої:

- компактні ведені пристрої, які поводяться як окремі модулі стосовно ведучого DP-пристрою;
- модульні ведені пристрої, що складаються з декількох модулів або під модулів;
- інтелектуальні ведені пристрої, що містять програму керування для власних підлеглих модулів.

До *компактних* ведених DP-пристроїв належать такі пристрої:

- ET200B і ET200C у версії для дискретних вхідних / вихідних модулів або аналогових вхідних / вихідних модулів, що забезпечують максимальну швидкість передавання даних 12 Мбіт/с;

- ET200 L-SC у дискретно-модульній конструкції з можливістю вільного комбінування кількості дискретних і аналогових модулів введення – виведення, що забезпечує швидкість передавання даних 1,5 Мбіт/с;
- шинні шлюзи, такі як з'єднувач DP/AS-I (DP/AS-I Link).

Прикладом *модульних* ведених DP-пристроїв може служити пристрій ET200M. Його конструкція аналогічна конструкції станції S7-300. Пристрій ET200M має профільну шину DIN, модуль блока живлення, інтерфейсний модуль IM 153 (на місці CPU) і до 8 сигнальних модулів (SM) або функціональних модулів (FM).

Іншим прикладом модульного веденого DP-пристрою є станція ET200S. У цілому одна станція ET200S дозволяє встановлювати до 63 модулів різного призначення, у тому числі силових модулів і перетворювачів, а також обслуговувати до 128 дискретних або до 64 аналогових каналів введення – виведення.

Інтелектуальні ведені PROFIBUS DP-пристрої

Прикладом інтелектуальних (програмувальних) ведених DP-пристроїв може бути станція S7-300, у якій задіяний CPU з DP-інтерфейсом і режимом веденого, а також станція S7-300 з комунікаційним процесором CP 342-5 у режимі веденого (slave) пристрою. У якості інтелектуального веденого DP-пристрою може працювати також станція ET200X з базовим модулем BM 147/CPU.

Підключення до PROFIBUS-PA

PROFIBUS-PA (Process Automation) – це шинна система для автоматизації процесу у вибухонебезпечних зонах або в так званих Ex-зонах.

Існують два способи з'єднання PROFIBUS-DP і PROFIBUS-PA:

- DP/PA *відгалужувач* (DP/PA coupler), який забезпечує одну швидкість передачі даних – 45,45 кбіт/с;
- DP/PA *з'єднувач* (DP/PA link), який забезпечує узгодження різних швидкостей обміну в PROFIBUS-DP і PROFIBUS-PA.

DP/PA *відгалужувач* дозволяє підключати PA-прилади польового рівня до мережі PROFIBUS-DP. У мережі PROFIBUS-DP *відгалужувач* DP/PA має статус веденого DP-пристрою. До одного DP/PA *відгалужувача* можна підключити не більше 31 PA-приладів польового рівня. Така сукупність польових приладів створює сегмент PROFIBUS-PA зі швидкістю обміну даними 31,25 Кбіт/с. Узяті разом сегменти PROFIBUS-PA утворюють шинну систему загального використання (shared).

DP/PA *відгалужувач* може мати два варіанти виконання:

- звичайний з вихідним струмом до 400 мА;
- Ex-версії з вихідним струмом до 100 мА.

DP/PA *з'єднувач* дозволяє підключати PA-прилади польового рівня до мережі PROFIBUS-DP і забезпечувати швидкість обміну даними від 9,6 Кбіт/с до 12 Мбіт/с. DP/PA з'єднувач має у своєму складі інтерфейсний модуль IM 157 і допускає установлення до 5 одиниць DP/PA відгалужувачів.

Підключення до послідовного інтерфейсу

Для з'єднання інтерфейсу PROFIBUS-DP із інтерфейсом RS 232C (V.24) використовується з'єднувач PROFIBUS-DP/RS-232C (PROFIBUS-DP/RS-232C link). З'єднувач DP/RS-232C підтримує протоколи 3964R та ASCII і є конвертором. У фреймі передається до 224 байт даних користувача.

З'єднувач DP/RS-232C забезпечує підключення приладів способом «точка до точки». Дані передаються зі збереженням консистентності в обох напрямках. Швидкість передачі даних по з'єднанню PROFIBUS-DP/RS-232C становить 38,4 Кбіт/с.

3.3 Принципи організації розподіленої периферії з AS-інтерфейсом

AS-інтерфейс (Actuator-Sensor Interface, інтерфейс привод-датчик) – це мережна система для обміну даними з устаткуванням процесу нижнього рівня керування.

Провідний пристрій AS-i може керувати групою, що містить до 31 одиниць ведених пристроїв AS-i. Керування забезпечується по двохпроводній AS-i лінії, по якій передається як живляча напруга, так й інформаційні сигнали (рис. 3.7). Кабель AS-i має конструкцію, що забезпечує легкість з'єднання з будь-яким модулем і неможливість підключення з неправильною полярністю.

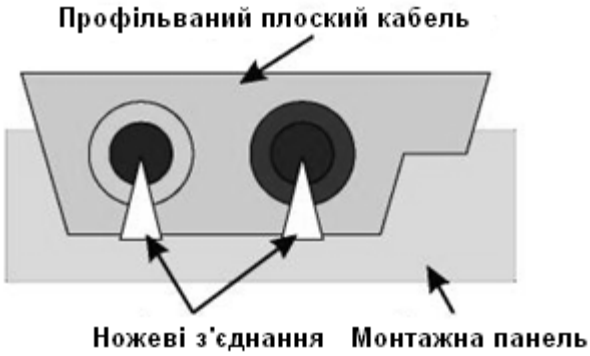


Рисунок 3.7 – Форма AS-i кабелю й спосіб його з'єднання з модулем

Ведені пристрої AS-i – це приводи або датчики із шинною організацією або AS-i модулі, до яких можна підключити до 8 двійкових датчиків або приводів. Максимальна довжина сегмента AS-i складає 100 м. Ця довжина сегмента може бути збільшена вдвічі при застосуванні повторювача або розширника.

При застосуванні повторювача ведені пристрої AS-i та джерела живлення повинні бути присутніми на вхідній і вихідній лініях повторювача. У разі застосування розширника ведені пристрої AS-i та джерело живлення повинні бути тільки на лінії, що йде від провідного пристрою AS-i.

Провідний пристрій AS-i (AS-i master) обновляє свої дані й дані всіх підключених до нього ведених пристроїв AS-i з інтервалом часу, що не перевищує 5 мс. Шина AS-i може бути підключена безпосередньо до SIMATIC S7 за допомогою комунікаційного процесора CP 342-2. Підключення до мережі PROFIBUS-DP виконується за допомогою DP/AS-інтерфейсного з'єднувача (рис. 3.8).

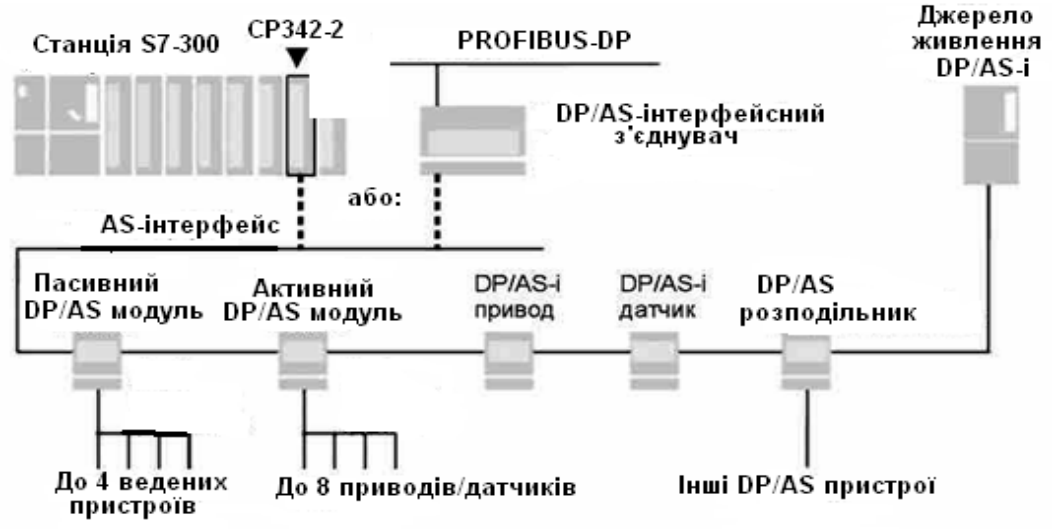


Рисунок 3.8 – Варіанти організації шини AS-інтерфейсу

Комунікаційний процесор CP 342-2 може бути використаний як провідний пристрій AS-i не тільки в станції S7-300, але й у станції ET200M. Він підтримує два робочих режиму – стандартний і розширений.

У *стандартному режимі* CP 342-2 поводиться як модуль введення – виведення. Він займає 16 вхідних байтів і 16 вихідних байтів (в аналоговому адресному просторі починається з адреси 256). Ведені пристрої AS-i параметризуються даними в CP.

У *розширеному режимі* реалізується повний набір функціональних можливостей провідного пристрою AS-i. Виклики провідного пристрою можуть виконуватися із програми користувача з використанням функціональних блоків FC.

DP/AS-інтерфейсний з'єднувач забезпечує підключення AS-i приводів і AS-i датчиків до мережі PROFIBUS-DP, причому в мережі PROFIBUS-DP з'єднувач має статус модульного веденого DP-пристрою, а в мережі AS-інтерфейсу він має статус провідного AS-i пристрою, який може контролювати до 31 пристроїв.

За максимальну кількість ведених AS-i пристроїв DP/AS-інтерфейсний з'єднувач займає 16 вхідних байтів і 16 вихідних байтів. Для входів і виходів кожного Slave завжди резервується 4 біта.

DP/AS-інтерфейсний з'єднувач може виконуватися у двох варіантах:

- версія 65 для жорстких умов експлуатації зі ступенем захисту IP66/67;
- версія 20 зі ступенем захисту IP20 з можливістю установлення додаткового командного інтерфейсу, при якому й для входів і для виходів діапазон адрес зростає до 20 байт.

3.4 Конфігурування станції децентралізованої периферії ET200M

Принципи використання станцій ET200M

У разі значних віддаленнях введення – виведення від системи автоматизації електричний монтаж може стати дуже об'ємним, а електромагнітні перешкоди можуть завдати шкоди надійності роботи. Для таких установок рекомендується використовувати систему децентралізованої периферії ET200.

Пристрій децентралізованої периферії ET200M є Slave-пристроєм DP у системі децентралізованої периферії ET200. Він складається з таких компонентів:

- джерело живлення (PS);
- підлеглий інтерфейсний модуль IM 153-1, IM 153-2 або IM 153-3;
- до 8 сигнальних модулів (SM), функціональних модулів (FM) або комунікаційних процесорів (CP). При цьому модулі FM і CP взаємодіють тільки з модулем DP-Master.

Вибір інтерфейсного модуля

Залежно від мети використання можна застосовувати три різних підлеглих інтерфейсних модуля IM 153.

На рисунку 3.9 показаний приклад конфігурації ET200M с CPU 315-2DP у якості Master-пристрою та IM 153-1 як пристрій децентралізованої периферії. У такій конфігурації обслуговуються *тільки сигнальні модулі* станції ET200M.

На рисунку 3.10 показаний приклад конфігурації ET200M с IM 153-2. Через IM 153-2 Master-пристрій DP або PG/OP можуть безпосередньо обмінюватися інформацією з *сигнальними й функціональними модулями.* Сіра лінія показує можливі «комунікаційні шляхи».

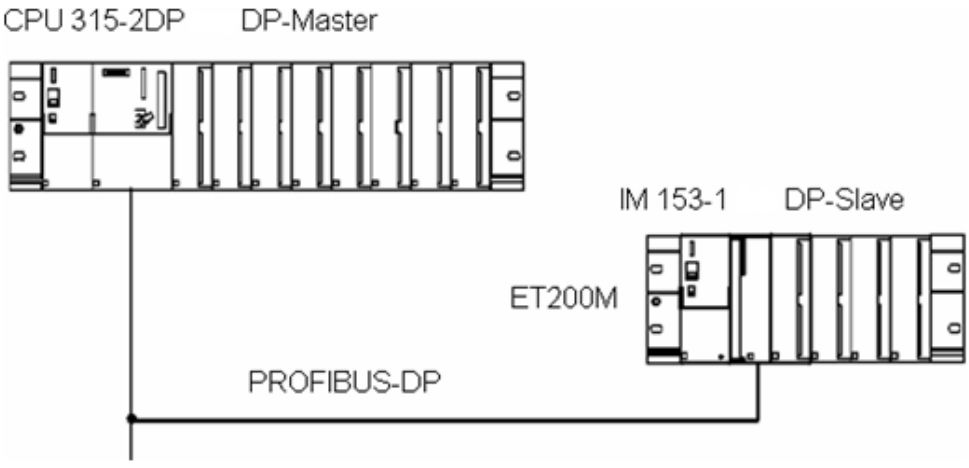


Рисунок 3.9 – Приклад конфігурації з IM 153-1

На рисунку 3.11 показаний приклад конфігурації ET 200M с IM 153-3 і резервною PROFIBUS на H-системі. Інтерфейсний модуль IM 153-3 дозволяє зв'язати станцію ET200M с двома Master-пристроями.

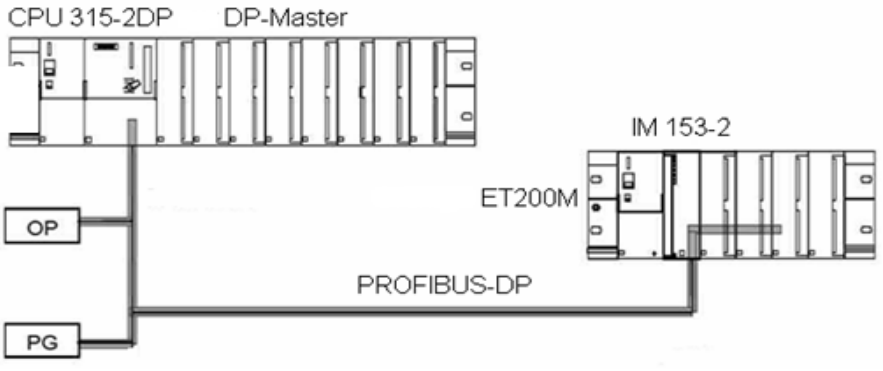


Рисунок 3.10 – Приклад структури з IM 153-2

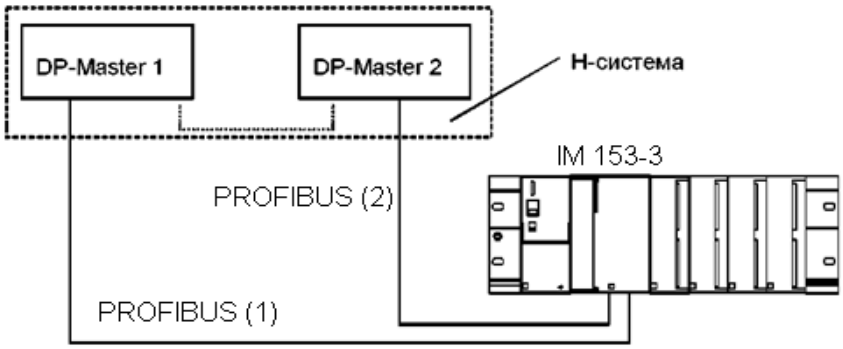


Рисунок 3.11 – Приклад конфігурації з IM 153-3

Таким чином, інтерфейсний модуль вибирається відповідно до вимог комунікацій.

Механічна конфігурація станції ET200M

Під час проектування механічної конфігурації, насамперед, потрібно брати до уваги споживання струму з боку модулів.

ET200M може бути змонтований не більше ніж на одному носії модулів (профільній шині), тому що з'єднання через інтерфейсні модулі з іншими носіями неприпустимо.

Для розміщення модулів на носії варто враховувати, що праворуч від IM 153 можна вставити не більше 8 модулів (сигнальних, функціональних і комунікаційних).

На рисунку 3.12 показано розміщення модулів у структурі ET200M.

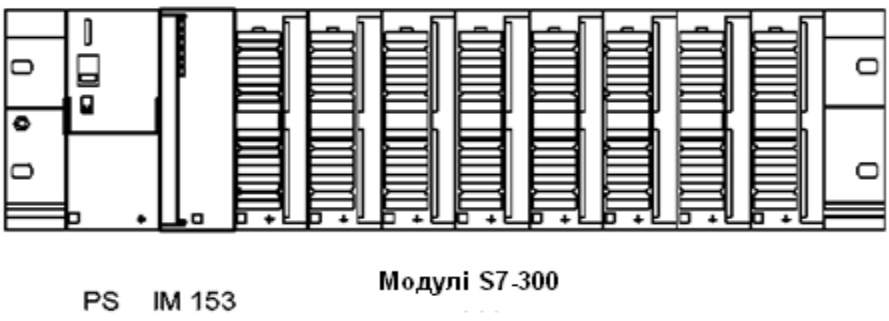


Рисунок 3.12 – Розміщення модулів ET200M

Модулі одержують необхідний для їхньої роботи струм із задньої шини, а також із зовнішнього джерела живлення навантаження. Варто врахувати, що IM 153 поставляє для задньої шини живлення, обмежене струмом 1 А. Тому при проектуванні ET200M необхідно визначити споживання струму й потужність втрат у станції. Визначення споживаного струму дозволить вибрати блок живлення, а визначення потужності втрат дозволить обґрунтовано вибрати розміри шафи й способи її вентиляції.

Розглянемо приклад визначення споживаної сили струму для станції ET200M.

Нехай станція ET200M складається з таких модулів:

- 1 джерело живлення PS 307 зі струмом 2 А;
- 1 інтерфейсний модуль IM 153-1 для підключення Slave-пристроїв;
- 4 цифрових модулі введення SM 321; DI 16x24 VDC;
- 3 цифрових модулі виведення SM 322; DO 16x24 VDC.

Знайдемо споживання струму для вищенаведеної структури ET200M. Занесемо всі модулі й споживані ними струми в таблицю 3.1. Баланс струмів визначимо підсумовуванням значень.

Таблиця 3.1 – Баланс струмів і потужності втрат у станції ET200M

Модуль станції		Споживання струму із задньої шини 5 В	Споживання струму від джерела живлення 24 В
Джерело живлення	PS 307 (2 А)		поставляє 2 А
	IM 153-1	поставляє 1А	споживає 650 мА
4 цифрових модулі введення SM 321 (5 В и 24 В)		(4×25) = 100 мА	(4×25) = 100 мА
4 цифрових модулі виведення (5 В и 24 В)		4×80 мА = 320 мА	4×120 мА = 480 мА
Сума		420 мА	1230 мА

З таблиці 3.1 виходять такі результати:

1. Споживання струму із задньої шини всіма сигнальними модулями в цілому становить 420 мА й, таким чином, не перевищує 1А, що поставляє в задню шину IM 153-1;

2. Споживання струму із джерела живлення навантаження 24 В всіма модулями становить 1,23 А. Таким чином, живлення навантаження 24 В може здійснюватися від джерела живлення PS 307 (2 А).

Після вибору модулів їм потрібно привласнити номери слотів. Схема нумерації слотів наведена у таблиці 3.2.

Таблиця 3.2 – Нумерація слотів у станції ET200M

№ слоту	Модуль	Примітка
1	Джерело живлення (PS)	*Використання джерела живлення не обов'язково*
2	IM 153	Займає 2 слоти
3	-	
4	1-й модуль S7-300	Праворуч поруч з IM 153
5	2-й модуль S7-300	
…	…	…
11	8-й модуль S7-300	

Визначення часу реакції модуля

Час реакції для ET200M залежить від таких факторів:

- час оброблення даних в ET200M;
- запізнювання входів і виходів.

Час оброблення даних усередині ET200M становить 1 мс. За цей час відбувається оброблення даних в IM 153-1 і передача даних усередині станції на інші модулі. Запізнювання в цифрових електронних модулях дискретного введення можна не враховувати. Якщо використовуються релейні виходи модулів виведення, то слід зважати на типовий час запізнювання від 10 до 20 мс.

Для аналогових виходів потрібно враховувати час перетворення аналогової величини.

І, нарешті, під час застосування модуля IM 153-3 необхідно враховувати час перемикання між провідними модулями PROFIBUS, який становить близько 30 мс плюс 2 цикли DP.

3.5 Конфігурування станції децентралізованої периферії ET200S

Станції ET 200S використовуються для побудови систем розподіленого введення – виведення у промислових мережах PROFIBUS DP або PROFINET із використанням програмувальних контролерів SIMATIC S7-300, S7-400, а також систем автоматизації С7.

Станція ET 200S може містити такі модулі:

- інтерфейсний модуль IM 151 для підключення станції до мережі PROFIBUS DP або PROFINET, а також для підтримки обміну даними із провідним пристроєм;
- електронні модулі введення – виведення дискретних і аналогових сигналів;
- технологічні модулі для рішення завдань позиціонування, зважування, швидкісного рахунку, комунікаційного обміну даними;
- фідери навантаження, призначені для комутації трифазних ланцюгів змінного струму потужністю до 7,5 кВт;

- перетворювачі частоти потужністю до 4 кВт;
- додаткові модулі для забезпечення безпеки роботи (PM-E, PM-D).

Примітка. Для моніторингу зовнішніх ланцюгів електронних і технологічних модулів у складі станції повинен використовуватися хоча б один модуль PM-E. Для моніторингу ланцюгів живлення силових модулів необхідний хоча б один модуль PM-D.

Приклад конфігурації станції ET200S наведений на рисунку 3.13.

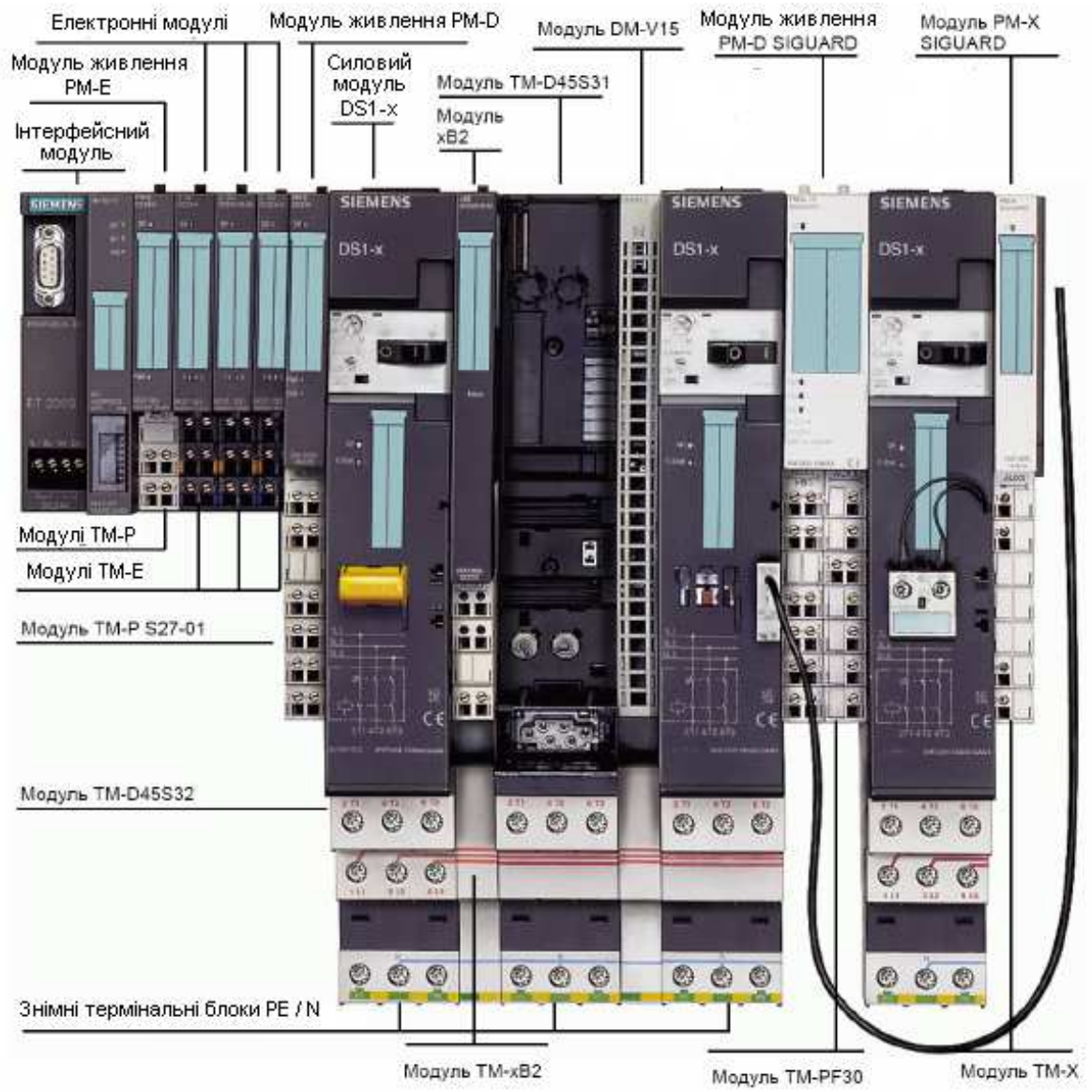


Рисунок 3.13 – Приклад конфігурації станції ET200S

Інтерфейсні модулі

ET200S може комплектуватися інтерфейсними модулями декількох типів. Вибір типу інтерфейсного модуля визначається кількістю використовуваних у станції модулів, видом інтерфейсу для підключення до мережі, можливостями виконання попереднього оброблення даних на рівні станції.

Під час вибору інтерфейсного модуля необхідно враховувати такі особливості його застосування:

1. Інтерфейсні модулі IM 151-1 використовуються для підключення ET200S до мережі PROFIBUS DP (інтерфейс RS-485) і підтримки обміну даними із провідним DP-пристроєм. Модуль IM 151-1 Basic забезпечує передачу 88 байт на телеграму, IM 151-1 Standard – 128 байт, IM 151-1 HF – 244 байта. Модуль IM 151-1 FO підключається до *оптичного каналу* й забезпечує введення – виведення128 байт даних на телеграму;

2. Інтелектуальні інтерфейсні модулі IM 151-7 CPU підтримують всі функції IM 151-1 і здатні виконувати попереднє оброблення даних на рівні станції. Обсяг даних на телеграму складає 244 байт. Модулі IM 151-7 оснащені вбудованим центральним процесором, аналогічним за своїми характеристиками CPU 314. Спільне застосування модулів IM 151-7 CPU та 6ES7 138-4HA00-0AB0 дозволяє використовувати ET200S як ведений пристрій в одній мережі і як провідний пристрій в іншій мережі PROFIBUS DP. IM 151-7 F-CPU підтримує функції протиаварійного захисту й автоматики безпеки на рівні операційної системи;

3. Інтерфейсні модулі IM 151-3 PN використовуються для підключення ET200S до мережі PROFINET IO з обсягом передачі даних 128 байт на телеграму.

Термінальні модулі TM-E

Електронні й технологічні модулі встановлюються на термінальні модулі TM-E. Модулі TM-E монтуються на 35 мм профільну шину DIN і містять вбудовані ділянки внутрішньої шини станції, вбудовані ділянки шини AUX1, гнізда для установлення електронного або технологічного модуля, а також контакти для підключення зовнішніх ланцюгів електронного або технологічного модуля. Шина AUX1 може використовуватися як шина заземлення або як шина допоміжного ланцюга живлення напругою до 220 В.

Термінальні модулі TM-E можуть збиратися в потенційні групи, що мають загальну шину живлення зовнішніх ланцюгів.

Кожна потенційна група починається термінальним модулем TM-P, на якому встановлюється модуль контролю живлення PM-E. Модуль PM-E здійснює моніторинг напруги живлення зовнішніх ланцюгів (датчиків і виконавчих пристроїв) електронних і технологічних модулів відповідної потенційної групи. Кількість потенційних груп у межах однієї станції ET 200S не обмежується.

Термінальні модулі TM-P

Кожна *силова* потенційна група починається силовим термінальним модулем TM-P, на який установлюється модуль PM-D.

Живлення на внутрішню силову шину подається через силовий термінальний модуль PM-D із 6 клемами (три з них використовується для підключення до мережі змінного струму, три – для підключення навантаження). Інші термінальні модулі цієї потенційної групи мають тільки клеми для підключення навантаження.

Силові модулі

У станції ET200S використовуються силові модулі двох видів:

- фідери навантаження для 3-фазних ланцюгів змінного струму напругою до 400 В;

- перетворювачі частоти для керування роботою 3-фазних асинхронних двигунів.

Керування силовими модулями та їхньою діагностикою виконується через внутрішню шину станції ET200S. За необхідності силові модулі можуть доповнюватися модулями керування електромагнітним гальмом.

Фідери навантаження

Фідери навантаження ET200S – це готові пускові комбінації для комутації ланцюгів 3-фазного змінного струму з навантаженням до 7,5 кВт.

Кожний фідер може містити:

- автоматичний вимикач;
- електромагнітний контактор (реверсивний або нереверсивний);
- пристрій плавного пуску.

Фідери типів F-DS1e-x (нереверсивний) і F-RS1e-x (реверсивний) оснащені вбудованими компонентами автоматики безпеки.

У силових модулях використовуються автоматичні вимикачі й контактори серії SIRIUS 3R. Кожний силовий модуль оснащений дискретними входами для підключення зовнішніх органів ручного керування, а також дискретними виходами для сигналізації про свій стан і виникаючі помилки. Силові модулі для нереверсивного керування встановлюються на термінальні модулі TM-DS, а силові модулі для реверсивного керування – на термінальні модулі TM-RS. Живлення силових модулів здійснюється від внутрішніх шин термінальних модулів. Термінальні модулі постачені силовою трифазною шиною, яка допускає струм навантаження 40 або 50 А.

На рисунку 3.14 показана схема з'єднання всіх компонентів фідера й установлення фідерного складання на профільну шину DIN.

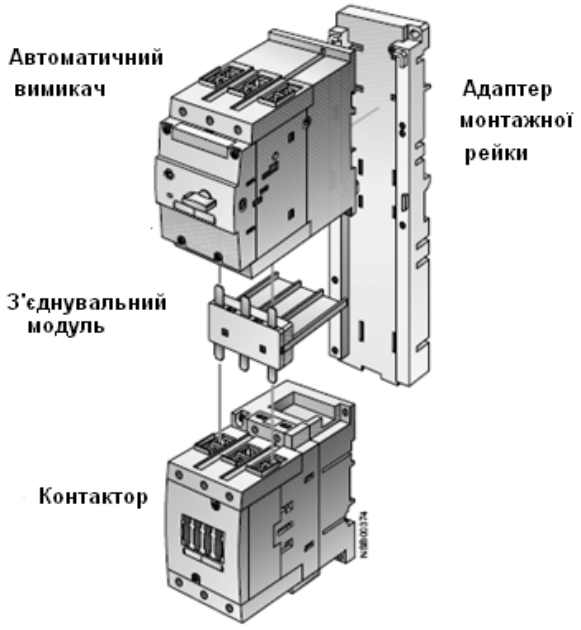


Рисунок 3.14 – Схема складання фідера

За необхідності використання 4-провідної схеми кожний термінальний модуль доповнюється модулем PE/N, що дозволяє формувати шину нейтрального проводу N і захисного заземлення PE.

Модулі перетворювачів частоти ET200S FC

Модулі перетворювачів частоти ET200S FC складаються з таких компонентів:

- модуль керування ICU24;
- силовий модуль IPM25;
- термінальні модулі для установлення й підключення модулів керування й силових модулів.

Силові модулі перетворювача частоти IPM25 здатні підтримувати лінійний або квадратичний закон регулювання U/f, а також векторне керування швидкістю зі зворотним зв'язком або без зворотного зв'язку. Керування перетворювачем виконує модуль ICU24. Якщо необхідна підтримка не тільки стандартних функцій керування приводом, але й функції автоматики безпеки та протиаварійного захисту, варто застосовувати модуль ICU24F. Силові модулі встановлюються на термінальні модулі 65 мм і 130 мм. Модулі керування встановлюються на термінальні модулі шириною 25 мм.

Параметри настроювання привода можуть задаватися з комп'ютера або за допомогою мікрокарти пам'яті. Частота перетворювача настроюється в діапазоні 2–16 кГц через 2 кГц. Рекомендоване значення – 8 кГц. Діапазон регулювання перетворювача – 0–650 Гц.

На рисунку 3.15 наведений приклад конфігурації станції ET200S на два модулі перетворювачів частоти ET 200S FC.

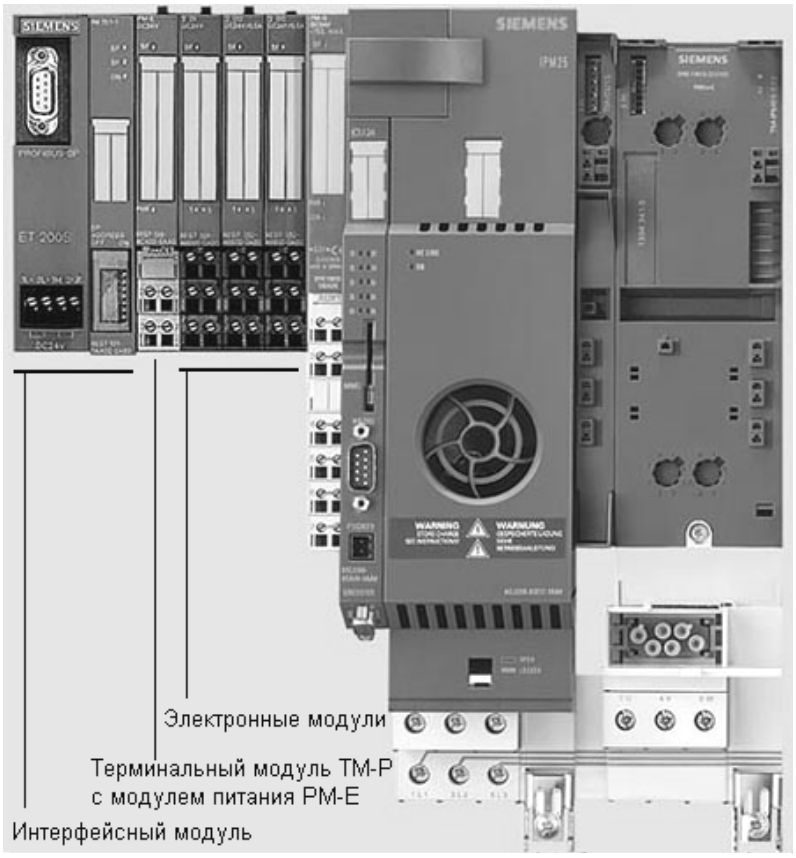


Рисунок 3.15 – Конфігурація станції ET 200S на два перетворювачі частоти ET200S FC (справа тільки термінальні модулі)

Перетворювач установлюється на термінальний блок L1/L2/L3, який застосовується для формування трифазної силової шини змінного струму. Ліворуч від перетворювача встановлений термінальний модуль TM-P15S 27-01 з модулем контролю живлення PM-D, який контролює наявність живлення на силовій шині. Зліва від цього блоку встановлені три електронних модулі. Інтерфейсний блок завжди встановлюється в крайньому зліва слоті.

3.6 Конфігурування децентралізованої периферії у системі STEP 7

В цілому завдання проектування розподіленої периферії зводиться до забезпечення необхідної структури системи керування, призначенню швидкості передачі даних у підмережах, а також формуванню адресного простору.

На першому етапі у програмному додатку NetPro системи STEP 7 створюється мережна структура, яка являє собою верхній рівень організації системи. Саме така послідовність забезпечує правильний вибір модуля центрального процесора й модулів для організації інтерфейсів.

На наступному етапі з використанням інтерактивного діалогу вибираються засоби комунікації, а також встановлюються властивості й параметри підмереж.

Робота завершується установленням необхідних сигнальних і технологічних модулів у вузлах мережі.

Під час конфігурування розподіленої периферії необхідно враховувати такі правила:

1. Всі абоненти підмереж повинні мати унікальні адреси. При цьому варто враховувати, що кількість абонентів MPI повинна бути не більше 32, у мережах PROFIBUS – до 126, а в Industrial Ethernet – до 1 024;

2. Під час установлення модуля процесора CPU йому за замовчуванням привласнюється адреса «2». У зв'язку з тим, що ця адреса може використовувати тільки один раз, у випадку застосування декількох CPU їм необхідно буде змінювати встановлені за замовчуванням адреси;

3. Під час призначення адрес для абонентів MPI/DP адреса «0» резервується для програматора, адреса «1» – для панелі оператора.

Для відображення мережної конфігурації в NetPro використовуються спеціальні графічні засоби мови конфігурування – символи.

Призначення символів на прикладі мережі MPI показано на рис. 3.16.

Символи є елементами керування. У разі подвійного клацання лівої кнопки миші по обраному символу відбувається таке:

- символ станції запускає додаток для конфігурування станції;
- символ підключення до мережі відкриває вікно завдання властивостей інтерфейсу;
- символ модуля відкриває вікно установки параметрів модуля.

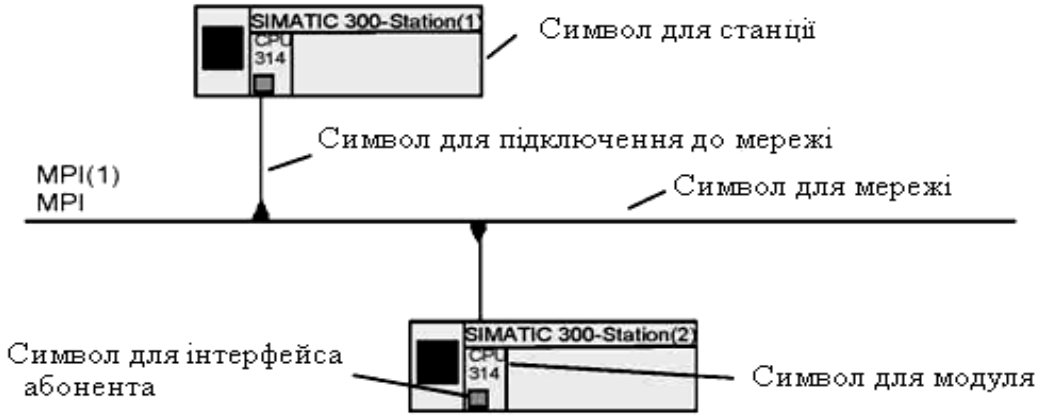


Рисунок 3.16 – Розташування основних символів графічного відображення на прикладі мережі MPI

У мережах PROFIBUS зазвичай створюють два профілі – DP і Standard. Профіль DP – це система настроювання параметрів, яку доцільно використовувати в мономайстерних системах із еквідістантним циклом шини, а профіль Standard застосовується для роботи із мультипроцесорними системами, які вимагають узгодження швидкодії через *різну тривалість* процесів оброблення даних.

Майстер-система складається із ведучого (провідного) пристрою DP-Master і одного або декількох ведених пристроїв DP-Slave.

У якості ведучого DP можна використовувати CPU з вбудованим інтерфейсом ведучого DP, інтерфейсний субмодуль, наприклад, IF 964-DP в CPU 488-4, інтерфейсний модуль із інтерфейсом ведучого DP, наприклад, IM 467 а також комунікаційний процесор CP у поєднанні з CPU, наприклад, CP 342-5.

У якості ведених DP використовуються *компактні ведені* DP, наприклад, ET200B, м*одульні ведені* DP, наприклад, ET200M а також і*нтелектуальні ведені* (I-Slaves) – станції, оснащені процесорними модулями й своїми користувальницькими програмами, наприклад, S7-300.

Процес створення майстер-системи рекомендується здійснювати у такій послідовності:

1. Запустити SIMATIC Manager. У діалоговому вікні установки виду проекту вибрати Finish і в новому вікні (S7_Proj*) ввести ім'я проекту;

2. Вибрати у меню Options мережне подання – Configure Network. У вікні конфігурування мережі NetPro, яке відкривається при цьому, за замовчуванням установлена мережа MPI і станція S7-300. Якщо необхідно встановити станцію S7-400, то видалити станцію S7-300 і перетягнути станцію S7-400 з розділу каталогу у вікно Network;

3. У контекстному меню вибрати команду Object Propertis і в діалоговому вікні Propertis на вкладці General ввести ім'я станції, наприклад, DP-Master. На вкладці Interface установити типи інтерфейсів, які повинна підтримувати система – MPI, PROFIBUS, Industrial Ethernet, PtP. Закрити вікно Propertis, підтвердивши зроблені установки кнопкою ОК;

4. У контекстному меню вибрати команду Open Object. Перехід у вікно конфігурування станції вимагає збереження в пам'яті мережного подання. Підтвердити свою згоду кнопкою ОК;

5. У порожнім вікні з ім'ям станції (DP-Master) конфігурування станції повинно бути почате з установлення необхідної стійки. Викликати контекстне меню, вибрати команду Insert Object і в діалоговому інтерактивному режимі вибрати SIMATIC 400 і стійку, наприклад, UR1;

6. Для вибору центрального процесорного модуля також використовувати інтерактивний діалог, що запускається командою Insert Object. По закінченні процедури вибору STEP-7 виводити вікно Propertis для установлення параметрів інтерфейсу PROFIBUS. На вкладці Parameters за замовчуванням установлена адреса «2», яку можна не міняти;

7. На цій же вкладці для створення майстер-системи натиснути кнопку New і в новому вікні властивостей на вкладці General ввести ім'я підмережі, а на вкладці Network Setting установити необхідну швидкість обміну даними по шині, наприклад, 6 Мбіт/с. Тут же можна установити профіль шини PROFIBUS (DP, Standard чи User Defined);

8. Для завдання постійного часу циклу шини (еквідистантності шини) натиснути кнопку Options і установити цикл шини (Constant Bus Cycle Time), наприклад таким, що дорівнює 10 мс. Підтвердити установки кнопкою ОК;

9. Під час установлення комунікаційного процесора для організації мережі Industrial Ethernet вибрати тип CP, наприклад, CP 443-1.

У результаті виконаних дій на відображенні стійки станції створюється рознімання DP (*X2*) і з'явиться символ майстра-системи . Цей символ є «якорем» для ведених модулів майстер-системи DP. Відображення створеної станції у вікні NetPro показано на рисунку 3.17, а у вікні станції DP-Master – на рисунку 3.18.

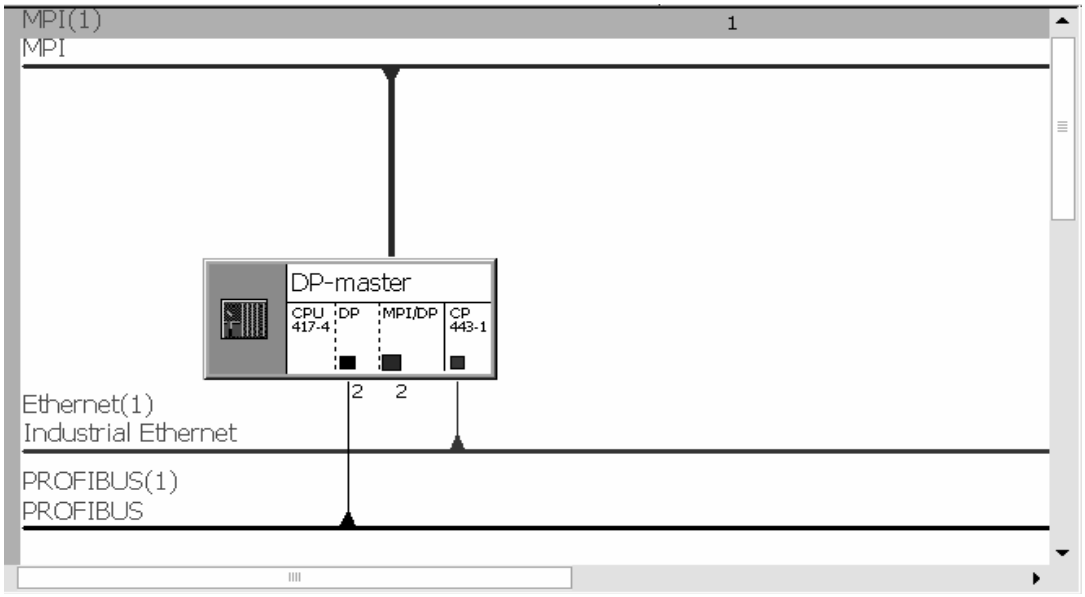


Рисунок 3.17 – Мережне подання станції у вікні NetPro

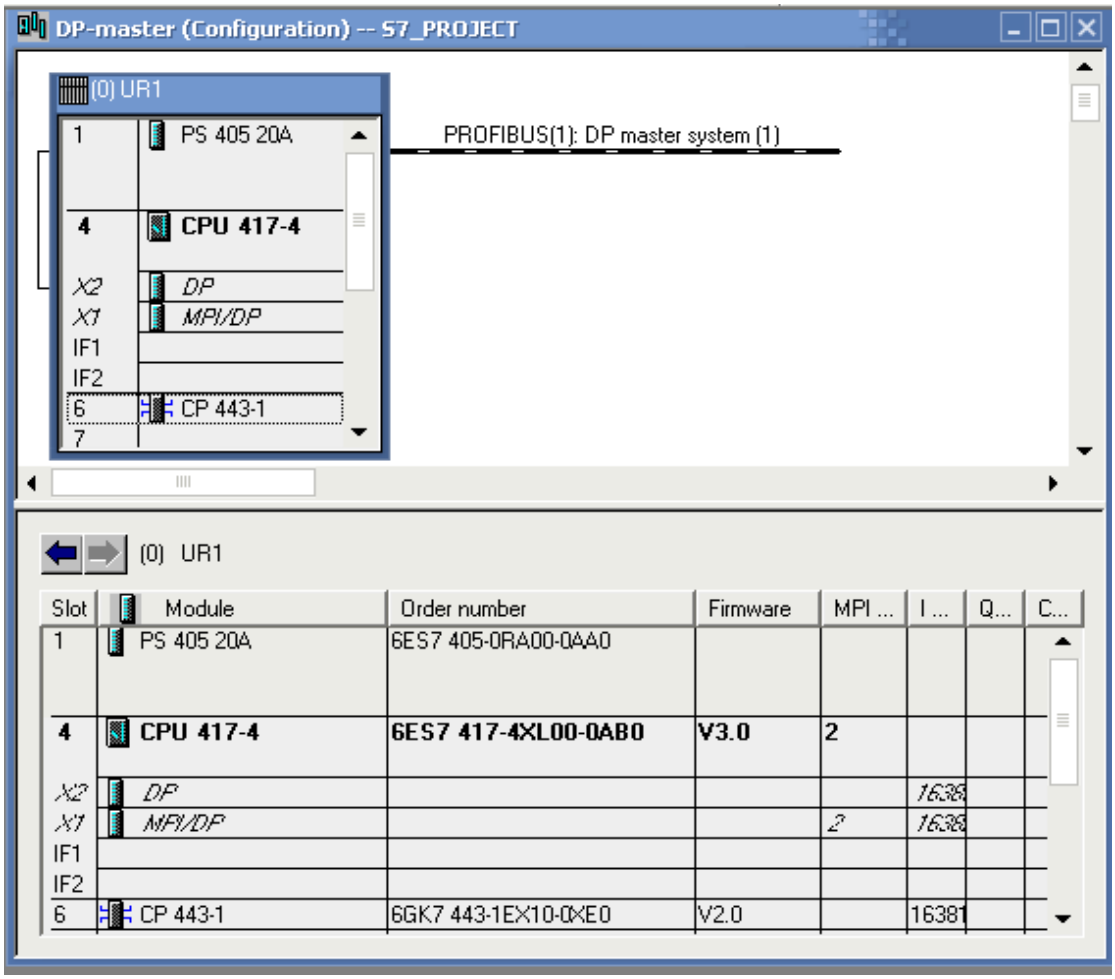


Рисунок 3.18 – Відображення станції майстер-системи у вікні DP-Master

Якщо символ майстер-системи у вікні станції не видно, то він, можливо, закритий конфігураційною таблицею. Треба зменшити висоту конфігураційної таблиці, у якій установлений ведучий DP. Якщо символ для майстер-системи DP знову не видно, треба вибрати команду меню Insert ⇨ DP Master System (*Вставити ⇨ Майстер-система DP*).

Після створення ведучого DP можна зробити вибір і проектування ведених DP. Ведені DP, що відповідають установленому ведучому модулю, можна перетягувати з вікна каталогу апаратури Hardware Catalog (розділ PROFIBUS-DP) і розміщати на майстер-системі.

При розміщенні ведених модулів DP до майстер-системи DP додається символ, що представляє ведений DP.

Приступаючи до конфігурування ведених модулів, варто врахувати специфічні особливості, пов'язані з конструктивними відмінностями модулів.

Конфігурування станції децентралізованої периферії ET200M

Модульна станція розподіленого введення – виведення ET200M комплектується інтерфейсними, сигнальними й функціональними модулями.

Для забезпечення обміну по мережі PROFIBUS DP на станції встановлюється інтерфейсний модуль IM 153. Зв'язок з контролером може здійснюватися також через окремий комунікаційний процесор CP 443-5 Basic.

Станція дозволяє підключити до 8 сигнальних або функціональних модулів. Для живлення модулів станції використовуються або блоки живлення сімейства PS, або блоки живлення сімейства SITOP Power.

Під час конфігурування станції ET 200M варто враховувати такі правила до слотів станції:

- власна периферія (входи / виходи станції) завжди починається зі слота 4;
- незалежно від того, чи установлено блок живлення (PS) у реальній структурі, чи ні, слот 1 завжди резервується для PS;
- слот 2 завжди резервується для інтерфейсного модуля DP;
- слот 3 завжди резервується для інтерфейсного модуля розширення (IM), незалежно від того, чи є реальний периферійний пристрій розширюваним, чи ні.

Правила до слотів необхідно враховувати при конфігуруванні всіх типів ведених DP: як модульних, так і компактних. Призначення слотів важливо також для аналізу діагностичних повідомлень, які запускаються слотами.

Станція ET200M підтримує «гарячу» заміну модулів, тобто заміну без зупинки станції. Для цього сигнальні модулі повинні бути встановлені на активні шинні модулі, які, у свою чергу, монтуються на спеціальну профільну шину DIN. Активні шинні модулі поєднуються між собою, створюючи внутрішню шину станції. Якщо «гарячої» заміни не потрібно, сигнальні модулі монтуються на стандартну профільну шину контролерів S7-300/400.

Приклад

Нехай майстер-система мережі PROFIBUS DP уже створена й потрібно сконфігурувати ведену станцію ET200M з одним аналоговим модулем введення, одним аналоговим модулем виведення, а також із цифровим модулем введення й цифровим модулем виведення. Крім того, станція повинна забезпечити підрахунок імпульсів з інкрементного датчика, а також з'єднання з AS-шиною.

Конфігурування станції ET200M треба здійснювати в такій послідовності:

1. Вибрати IM 153-2, що забезпечує «гарячу» заміну модулів (папка PROFIBUS DP ⇨ ET 200M) і відбуксирувати цей модуль на DP master system. При цьому STEP-7 виводить діалогове вікно Properties – PROFIBUS node ET200 IM 153-2 (*Властивості – вузол PROFIBUS ET200 IM 153-2*). У полі PROFIBUS Address (*Адреса PROFIBUS*) треба вибрати адресу для slave-пристрою DP, наприклад, 3. Закрити вікно кнопкою OK.

2. Вставка модулів. Розкрити дерево каталогу (папка PROFIBUS DP / ET200M / IM 153-2) і, буксируючи потрібні модулі з каталогу, вставити їх у слоти ET200M, починаючи зі слота 4. Функціональний модуль FM 350 для підрахунку імпульсів і комунікаційний процесор для зв'язку із шиною AS знаходяться у цій же папці.

Результати конфігурування станцій представлені на рисунку 3.19.

Особливості конфігурування станції ET200S

Станція ET 200S призначена для побудови систем розподіленого введення – виведення на основі PROFIBUS DP або PROFINET.

Станція ET 200S може комплектуватися:

1. Звичайними або інтелектуальними інтерфейсними модулями для підключення до електричних або оптичних каналів PROFIBUS;
2. Модулями введення – виведення дискретних і аналогових сигналів.
3. Технологічними модулями для рішення завдань позиціонування, швидкісного підрахунку, обміну даними через послідовні інтерфейси;
4. Силовими модулями для керування споживачами 3-фазного змінного струму, наприклад, 3-фазними електродвигунами.

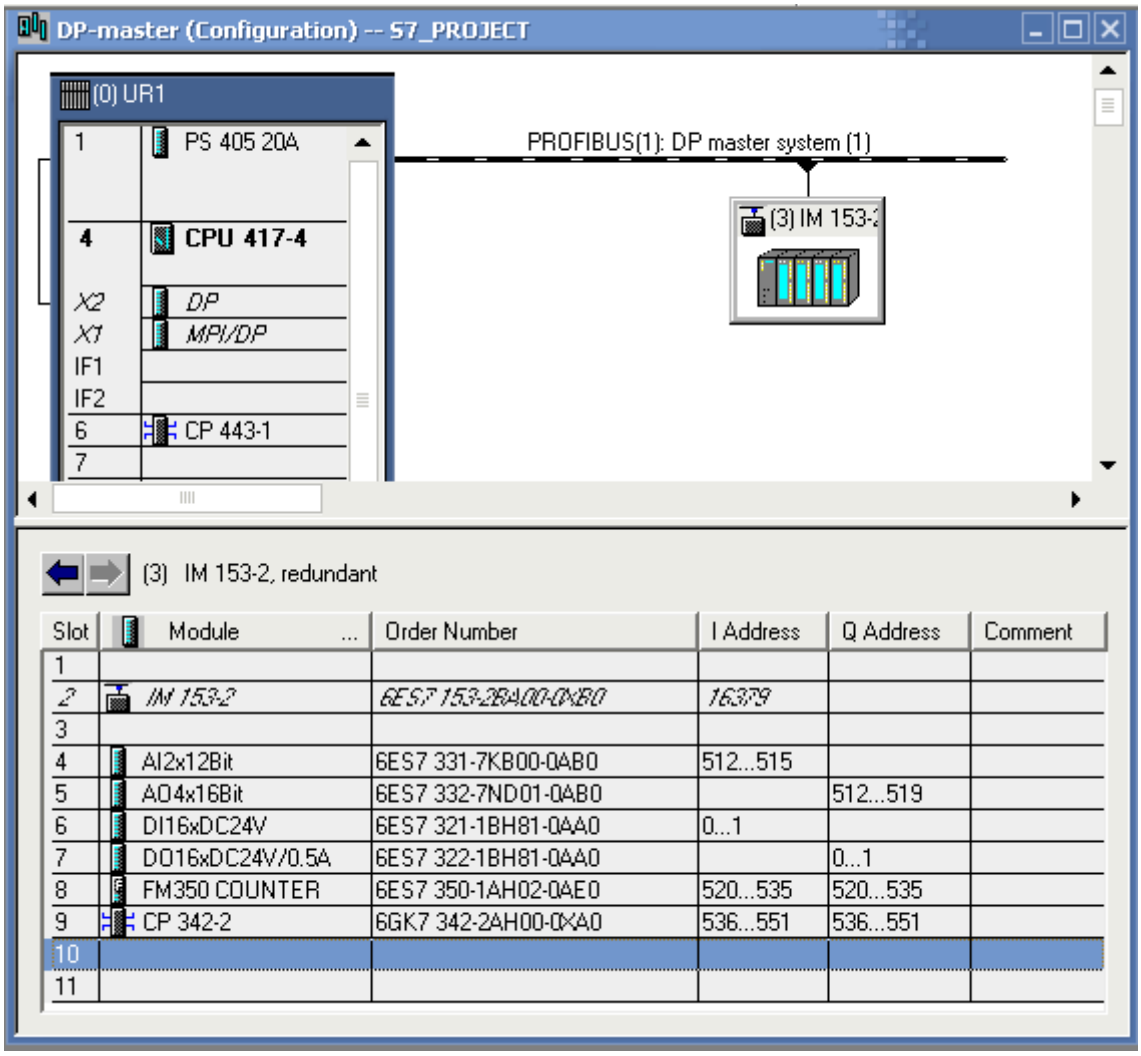


Рисунок 3.19 – Приклад конфігурування станції ET200M

Під час конфігурування станції ET200S варто враховувати правила розміщення модулів:

- першим ліворуч установлюється інтерфейсний модуль IM 151;
- праворуч від інтерфейсного модуля встановлюється модуль контролю живлення *електронних* модулів PM-E;

• праворуч від модуля PM-E розташовуються електронні модулі (SM), а також технологічні модулі (FM), що виконують функції підрахунку, генерації імпульсів, позиціонування й т. п. Електронні й технологічні модулі встановлюються на термінальні модулі TM-E.

Після електронних і технологічних модулів встановлюються засоби мотор-стартера – спочатку *модуль контролю живлення силових модулів PM-D*, а праворуч від нього один із стартерів:

• на термінальний модуль TM-D, призначений для нереверсивного привода встановлюється силовий модуль (direct starter) DS 1-х нереверсивного керування;

• на термінальний модуль TM-R, призначений для реверсивного привода, встановлюється силовий модуль RS 1-х реверсивного керування.

Якщо навантаження вимагає чотирипровідного силового ланцюга, силові модулі забезпечуються знімними термінальними блоками PE/N, які дозволяють сформувати нульовий провід N.

Варто врахувати, що кожна *силова група* вимагає окремого модуля контролю живлення PM-D. Цей модуль повинен установлюватися на термінальний модуль TM-P15S 27-01.

Станція допускає установлення на один інтерфейсний модуль максимум до 63 модулів.

Конфігурування інтелектуальних ведених DP

Ознакою інтелектуального веденого DP є наявність програмного забезпечення, завдяки чому вхідні й вихідні дані надаються в розпорядження ведучого DP не безпосередньо від реального входу – виходу, а від виконуючого попереднє оброблення інтелектуального веденого DP.

Інакше кажучи, якщо застосовується звичайний ведений DP, наприклад, модульний ET200M, то ведучий DP звертається до децентралізованих входів-виходів модуля, а якщо застосовується інтелектуальний ведений DP, то ведучий звертається до *області операндів* CPU, який виконує попереднє оброблення даних.

Приклад конфігурування S7-300 як інтелектуального веденого

Нехай потрібно створити таку конфігурацію:

• ведуча станція (ім'я «DP Master») з CPU 416-2 DP і вбудованим інтерфейсом DP;

• ведена станція (ім'я «DP Slave») з CPU 315-2 DP у якості інтелектуального веденого DP.

Процес конфігурування містить у собі такі кроки:

Крок 1. Створюємо проект із ім'ям, наприклад, S7_Pro1 із двома станціями – SIMATIC 400 і SIMATIC 300 (рис. 3.20). Клацаємо в дереві проекту по рядку SIMATIC 400, потім по символу Hardware і в порожнім вікні, що відкрилося, командою Insert Object запускаємо інтерактивний діалог, у якому спочатку вибираємо SIMATIC 400, а далі стійку UR2.

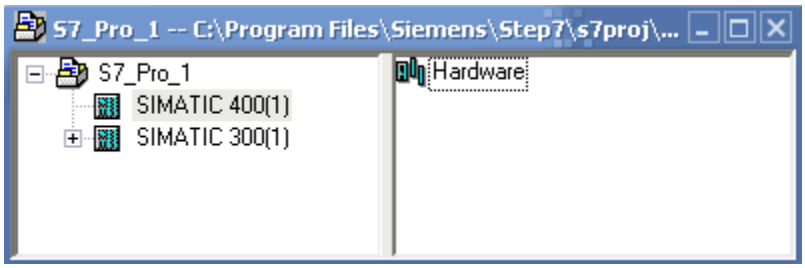


Рисунок 3.20 – Створення проекту

Крок 2. Створюємо ведучу станцію SIMATIC 400 і мережу PROFIBUS. Спочатку для слота 1 вибираємо джерело живлення PS 407 10A. Далі для слота 3 по ланцюжку списків: CPU 400 ⇨ CPU 416-2 ⇨ 6ES7 416-2XK01-0AB0 ⇨ V3.0 вибираємо модуль центрального процесора CPU 416-2. У вікні Properties PROFIBUS interface DP на вкладці Parameters установлюємо адресу – «2». Далі кнопкою New відкриваємо вікно, у якому задаємо ім'я мережі – PROFIBUS(1). І, нарешті, на вкладці Network Setting задаємо швидкість передачі по мережі (1,5 Мбіт/с) і профіль (DP).

Крок 3. Створюємо майстер-систему PROFIBUS. Після виконання попереднього кроку у вікні HW Configuration з'являється відображення станції та символ шини: *PROFIBUS(1): DP master-system*. Клацнувши двічі по рядку слота 4 (вивід *X3*), відкриваємо вікно Properties DP і на вкладці General у полі Name установлюємо ім'я станції – DP-Master. Далі на вкладці Operating Mode призначаємо режим – DP-master. У результаті одержимо відображення провідної станції, що показано на рисунку 3.21.

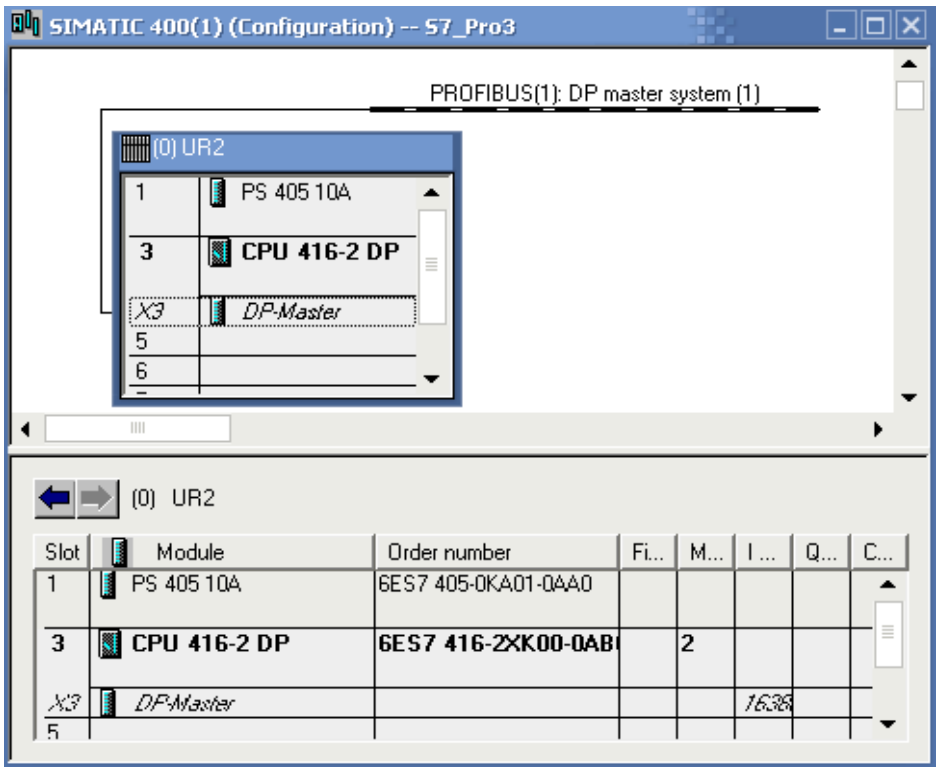


Рисунок 3.21 – Відображення ведучої станції PROFIBUS(1)

Крок 4. Створюємо ведену станцію SIMATIC 300 з CPU 315-2 DP. Для цього переходимо в SIMATIC Manager (у вікно проекту, де перебуває станція) і, вибравши станцію у вікні браузера, двічі клацаємо лівою кнопкою миші по її символу Hardware. У результаті відкривається порожнє вікно з ім'ям станції – SIMATIC 300.

Використовуючи команду Insert Object контекстного меню, установлюємо в інтерактивному діалозі стійку (SIMATIC 300 ⇨ RACK-300 ⇨ Rail). Далі робимо тим же способом заповнення стійки модулями – у слот 1 установлюємо PS 307 2A, а у слот 2 – процесорний модуль CPU 315-2 DP. У вікні Properties вибираємо вже створену шину PROFIBUS(1) і встановлюємо адресу станції в цій шині – 3.

Після цього, клацнувши на рядку слота з інтерфейсом DP, у вікні Properties DP на вкладці General у полі Name призначаємо ім'я станції – DP-Slave, а на вкладці Operating Mode режим – DP-slave.

Результат проектування станцій показаний на рисунку 3.22 (відображення у вікні HW Configuration) і на рисунку 3.23 (відображення у вікні Network).

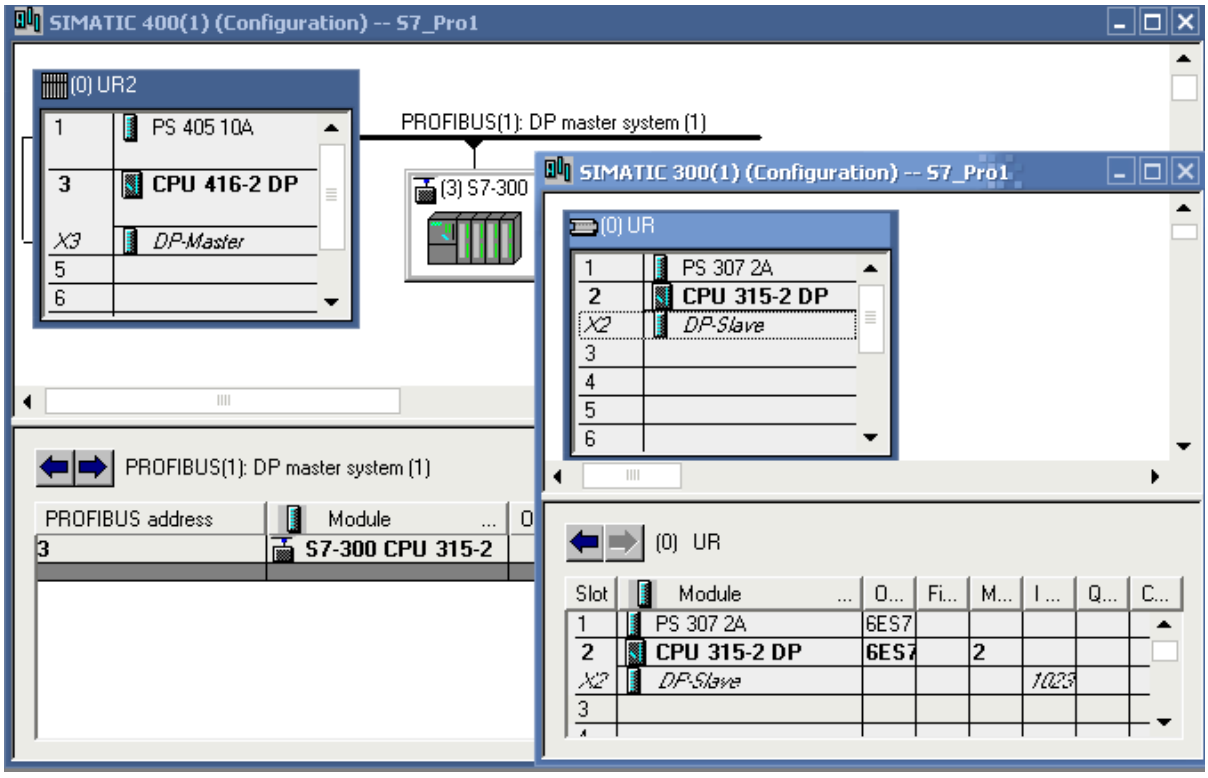


Рисунок 3.22 – Відображення ведучої і веденої інтелектуальної станцій у вікні HW Configuration

Крок 5. Установлюємо з'єднання між станціями. Для цього вертаємося у вікно провідної станції (SIMATIC 400). Тут виділяємо символ шини PROFIBUS і командою Insert Object входимо в режим інтерактивного діалогу, де вибираємо команду Configured Station ⇨ CPU 31x. У вікні DP slave properties на вкладці Connect повинна перебувати інформація про ведену

станцію, що підключається – ім'я, адреса, тип процесорного модуля та номери слотів. Після натискання на кнопку Connect цей запис повинен зникнути, що свідчить про створення з'єднання.

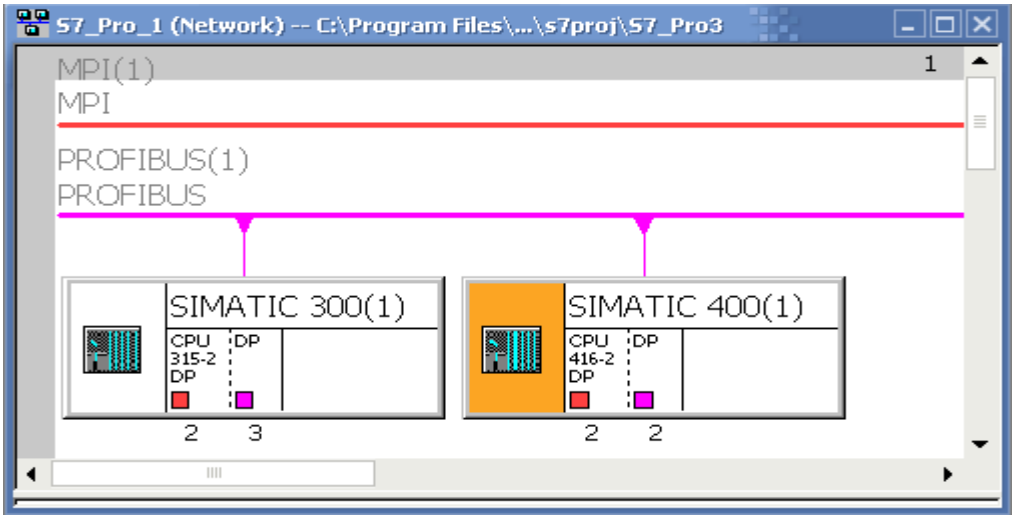


Рисунок 3.23 – Відображення ведучої і веденої інтелектуальної станцій у вікні Network

Крок 6. Параметруємо з'єднання. У тім же вікні Properties переходимо на вкладку Configuration і натискаємо на кнопку New. У вікні настроювання параметрів зв'язку (DP slave properties → Configuration → Row 1) є два поля: ліве для ведучої станції, праве – для веденої.

Установивши для DP-Master напрямок передачі Output, введемо в поле адреси *адресу пам'яті*, з якої дані будуть виводитися, наприклад, 248. Для веденої станції (режим Input) призначимо адресу в області введення, наприклад, 15. Підтверджуємо дані натисканням кнопки ОК. У результаті у вікні DP slave properties на вкладці Configuration з'явиться перший ряд даних (Row 1), що визначає властивості з'єднання.

Для параметрування операції читання з веденого пристрою необхідно знову натиснути кнопку New і, змінивши напрямок передачі (в DP-Master встановити Input), потрібно ввести нові значення адресації, наприклад, для ведучого встановити адресу 252, а для веденого – 16.

Всі виконані установлення підтверджуємо кнопкою ОК.

Вид вікна DP slave properties після введення параметрів зв'язку наведений на рисунку 3.24.

Для перевірки правильності конфігурування майстер-системи з інтелектуальним веденим вузлом необхідно виділити одну станцію й вибрати в головному меню Station ⇨ Consistency Check.

Правильність конфігурування оцінюється за консістентністю (погодженістю) станцій одна відносно іншої. Тому така перевірка повинна здійснюватися для обох станцій.

Результати оцінки приводяться у вікні Consistency Check для кожної станції. На рисунку 3.25 вони наведені для станції SIMATIC 400.

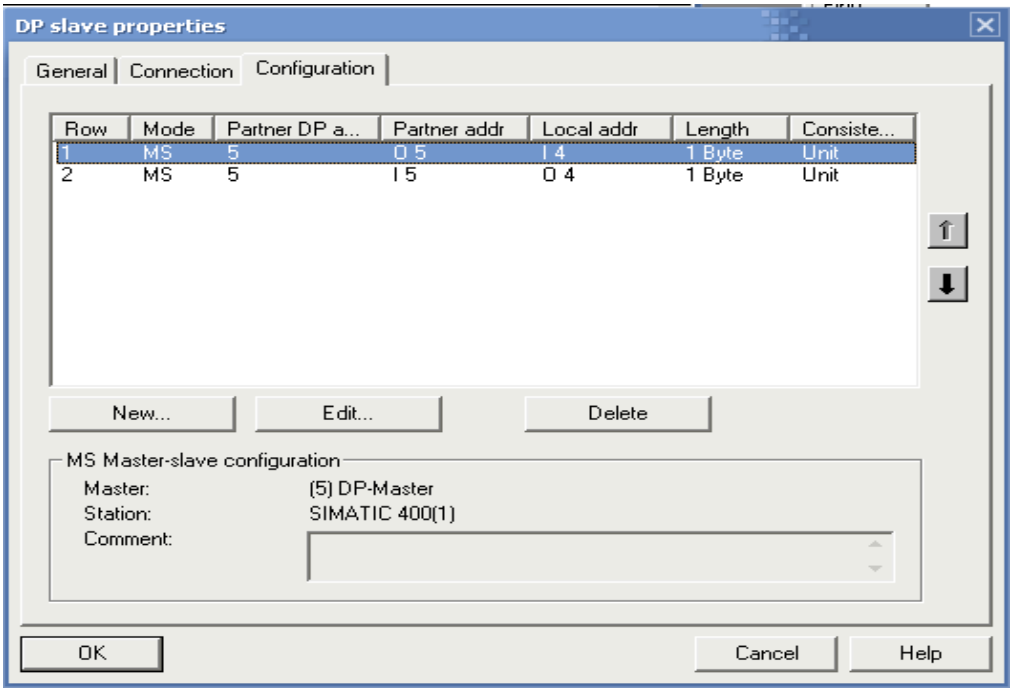


Рисунок 3.24 – Вид вікна DP slave properties з параметрами з'єднання

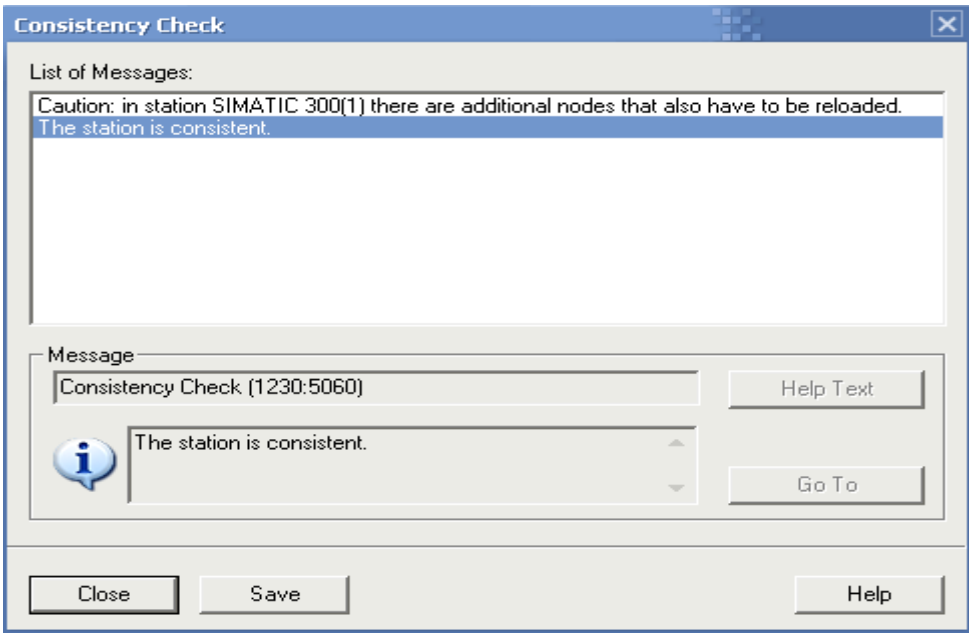


Рисунок 3.25 – Вид вікна з виводом результату перевірки консістентності станцій

Конфігурування ведених з функціями ведучих для підмережі

Особливості конфігурування DP/AS-i Link

Під час конфігурування ведених DP/AS-i Link (розподілений інтерфейс виконавчих пристроїв і датчиків) варто врахувати, що DP/AS-i Link конфігурується з веденими AS-i. У разі розміщення пристроїв DP/AS-i Link у нижній частині вікна станції автоматично відображається конфігураційна таблиця, у яку потрібно розмістити ведені AS-i з вікна каталогу апаратури Hardware Catalog.

Особливості конфігурування PROFIBUS PA

Щоб сконфігурувати розподілену периферію на польових пристроях PROFIBUS-PA (PROFIBUS для автоматизації процесів), необхідно встановити в мережу PROFIBUS-DP з'єднувач (шлюз) DP/PA.

Конфігурувати з'єднувач DP/PA в утиліті HW Config не потрібно – він невидимий для станції. Потрібно тільки відбуксирувати DP/PA-Link, наприклад, IM 157 з вікна каталогу апаратури на майстер-систему DP. Під час установлення з'єднувача DP/PA виводиться попередження, що швидкість передачі мережі PROFIBUS повинна дорівнювати 45,45 Кбод. Для польових пристроїв PA з'єднувач DP/PA зменшить швидкість передачі до 31,25 Кбод.

Відображення DP/PA-Link містить у собі поряд із символом для самого пристрою також і символ майстер-системи PA подібно майстер-системі DP. На цей символ і призначаються польові пристрої PA. Однак для цього необхідний додатковий програмний пакет SIMATIC PDM.

Контрольні питання

1. В якій послідовності проектується розподілена периферія?
2. Що являє собою проект із *простим* та *інтелектуальним* веденим DP?
3. Що являє собою проект із *двома майстер-системами* DP?
4. Для чого застосовується мережа PROFIBUS-DP?
5. Які засоби можна застосовувати для створення пристрою DP-Master?
6. Які засоби можна застосовувати для створення пристрою DP-Slave?
7. Які засоби можна застосовувати для створення інтелектуального веденого DP-пристрою (DP ISlave)?
8. За допомогою яких засобів можна створити мережу PROFIBUS-PA?
9. За допомогою яких засобів можна створити мережу AS-i?
10. Як підключаються модулі (датчики й виконавчі пристрої) до AS-i?
11. Для рішення яких завдань призначені станції ET 200M?
12. За якими правилами встановлюються модулі в станцію ET 200M?
13. Як визначається баланс струмів втрат у станції ET 200M?
14. Які запізнювання варто враховувати під час організації роботи зі станцією ET 200M?
15. Для чого призначена й що може містити в собі станція ET 200S?
16. Для чого призначені та як установлюються термінальні модулі TM-E?
17. Для чого призначені термінальні модулі TM-P і TM-D?
18. Які силові модулі можна встановити в станцію ET 200S?
19. Для чого призначені фідери навантаження ET 200S?
20. Які функції реалізує модуль перетворювача частоти ET 200S FC?

4 ПРОЕКТУВАННЯ СТРУКТУРИ ПРОГРАМИ

4.1 Структурна організація програмного забезпечення в CPU

В CPU завжди виконуються дві програми: *операційна система* та *програма користувача*.

Операційна система

Кожний CPU містить операційну систему, яка організує функції та послідовності в CPU, не пов'язані з конкретним завданням керування.

Операційна система забезпечує такі функції:

- виклик програми користувача;
- оброблення «теплого» і «гарячого» перезапуску;
- відновлення таблиці образу процесу для входів і виведення таблиці образу процесу для виходів;
- виявлення переривань і виклик організаційних блоків переривань;
- виявлення й оброблення помилок;
- керування областями пам'яті;
- обмін інформацією із пристроями програмування й іншими комунікаційними партнерами.

Параметри операційної системи встановлюються за замовчуванням і змінювати їх не можна.

Програма користувача

Програма користувача містить ті функції, які необхідні для реалізації конкретного завдання автоматизації.

Завдання програми користувача полягають у такому:

- оброблення даних процесу (зчитування й аналізування вхідних сигналів, обчислення функцій переходу й значень вихідних сигналів);
- реакція на переривання;
- оброблення порушень у нормальному виконанні програми.

Програмне забезпечення STEP 7 дозволяє структурувати *користувальницьку* програму, іншими словами, розбивати програму на окремі автономні програмні секції, які простіше модифікувати й налагодити. Важливо також і те, що такі програмні секції можна використовувати неодноразово для повторюваних технологічних функцій процесу. Під час структурного програмування програмні секції представляються організаційними блоками (OB), функціональними блоками (FB), функціями (FC) і блоками даних (DB).

Функціонування структурованої програми користувача полягає у викликах функціональних блоків (рис. 4.1).

Кількість блоків може бути будь-якою. Для організації циклічного виконання програми обов'язково застосування організаційного блоку OB1.

У принципі можна записати всю користувальницьку програму в одному блоці OB1. Таке програмування називається лінійним. Лінійне програмування доцільно застосовувати тільки під час створення простих програм.

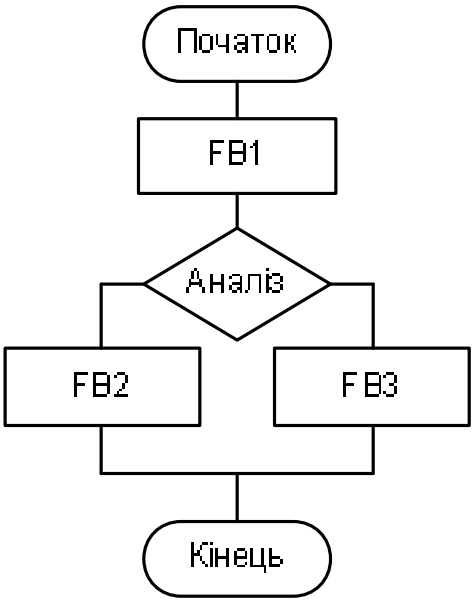


Рисунок 4.1 – Сутність функціонування структурованої програми

Фази циклічного оброблення програми

Користувальницька програма, записана в OB1, складається з функціональних блоків, які працюють із поточними значеннями входів і формують поточні значення виходів. Для того щоб під час циклічного обробляння програми мати несуперечливий образ сигналів процесу, CPU звертається не безпосередньо до периферійних входів (PI) і виходів (PQ) на модулях введення/виведення, а до області внутрішньої пам'яті CPU, яка містить образ входів (I) і виходів (Q).

Циклічне обробляння програми містить такі фази:

1. Операційна система запускає час контролю циклу;
2. CPU переписує значення з таблиці образу виходів процесу в модулі виведення;
3. CPU зчитує стани входів і записує їх у таблицю образу процесу на входах;
4. CPU виконує програму користувача;
5. Операційна система виконує завдання, що чекають своєї черги;
6. CPU перезапускає час контролю циклу й починає новий цикл.

Циклічне обробляння програми може бути перервано в результаті подачі команди STOP, виходу з ладу живлення, а також виникнення несправності або помилки в програмі.

Час виконання циклу

Час виконання циклу – це час, необхідний операційній системі для виконання циклічної програми й системних операцій, наприклад, переривань.

Час виконання циклу ТС не однаковий в кожному циклі. На рисунку 4.2 показаний випадок збільшення часу виконання циклу CPU. Тут час виконання циклу TC_1 більше часу циклу TC_2 через переривання за часом дня, яке виконується організаційним блоком OB10.

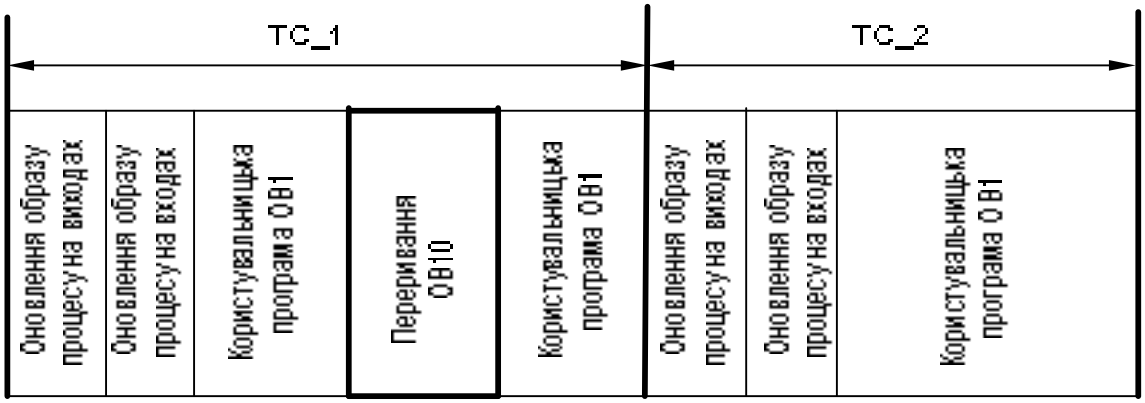


Рисунок 4.2 – Виникнення різного часу виконання циклів CPU

За допомогою STEP 7 можна задавати максимальний і мінімальний час циклу. Якщо мине максимальний час циклу, то можливі дві процедури CPU – або він переходить у режим STOP, або буде викликаний блок OB80, щоб визначити, як CPU повинен реагувати на таку помилку.

Якщо час робочого циклу повинен бути однаковим, наприклад, у завданнях регулювання, то варто встановити мінімальну тривалість циклу. Різниця між максимальним і мінімальним часом буде являти собою резерв для обслуговування переривань і фонових завдань.

Ієрархія створення та викликів блоків у програмі користувача

Функціонування структурованої програми користувача полягає у викликах функціональних блоків. Для цього використовуються спеціальні команди, які можуть бути запущені тільки в логічних блоках.

Порядок викликів блоків називається ієрархією викликів. Кількість блоків, які можуть бути вкладені один в один (глибина вкладення) залежить від конкретного CPU.

Порядок викликів блоків диктує відповідно порядок їх створення.

Якщо побудувати ієрархію викликів у організаційному блоці OB1, як показано, наприклад, на рисунку 4.3, то сформулювати порядок створення блоків можна таким чином:

- блоки створюються зверху вниз, тобто починати потрібно з верхнього ряду блоків;
- оскільки кожний викликуваний блок уже повинен існувати, то усередині ряду блоки повинні створюватися *справа наліво*.

Враховуючи цей порядок, для наведеного на рисунку 4.9 приклада визначимо таку послідовність створення блоків:

1. Для верхнього ряду спочатку повинен бути створений блок з функцією FC1, потім функціональний блок FB1 з його екземплярним блоком даних DB1;

2. Для наступного ряду спочатку створюється блок системних даних DB1 для системної функції SFC1, після цього складається програма функціонального блоку FB21, для якого потім створюється блок даних DB21. Далі розробляється функціональний блок FB2 з його екземплярним блоком даних DB2.

3. Для останнього ряду створюється блок з функцією FC3.

4. На закінчення створюється блок OB1.

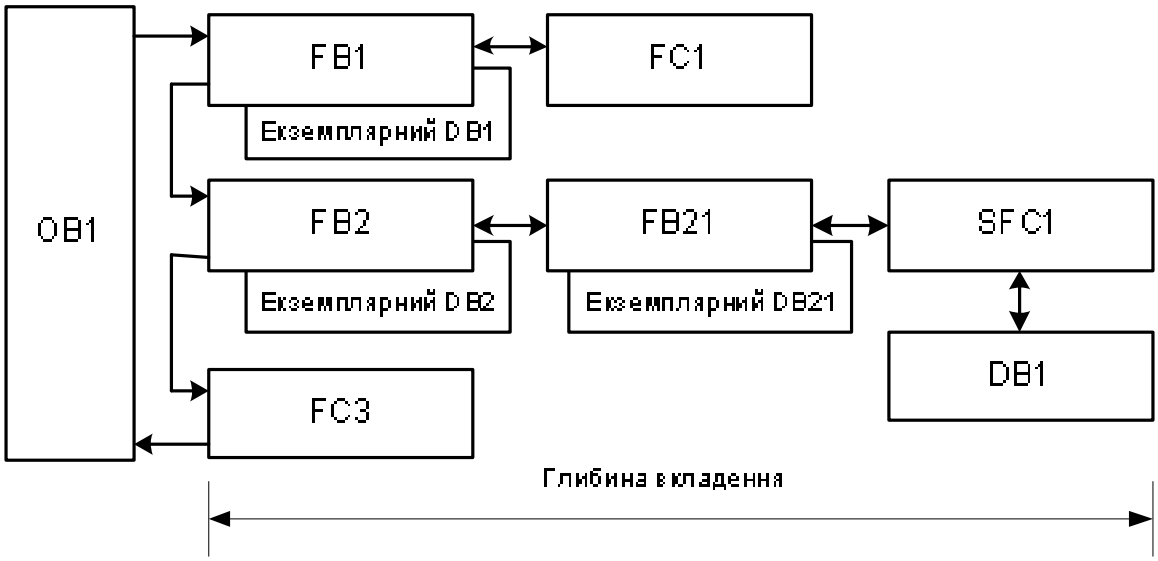


Рисунок 4.3 – Порядок викликів блоків усередині циклу програми

У програмі STEP 7 використовуються різноманітні операнди – сигнали входів / виходів, меркери, лічильники, таймери, блоки даних і функціональні блоки. Звертання до цих операндів здійснюється через їхні абсолютні адреси. Однак програма буде значно легше читатися, якщо замість цих адрес застосувати символьні імена. STEP 7 може автоматично перетворювати символьні імена в необхідні абсолютні адреси. Однак для цього символьні імена уже повинні бути призначені абсолютним адресам. Так, наприклад, якщо призначити символьне ім'я Motor_On адресі Q 4.0, то потім можна використовувати Motor_On, як адресу в операторі програми.

Варто врахувати, що символьні адреси полегшують розуміння відповідності елементів програми й апаратних засобів керування процесом.

Розрізняють локальні та глобальні символи. Їх розходження зводяться до такого:

1 Глобальні символи діють у всій програмі користувача, а локальні – у межах блоку, для якого вони визначені;

2 Глобальні символи повинні бути записані в таблиці символів, а локальні символи призначаються в таблиці опису змінних блоку даних;

3 Глобальні символи можуть визначати всі змінні, а локальні – тільки параметри блоку (вхідні, вихідні, статичні й тимчасові).

У розділі кодів програми глобальні символьні імена з таблиці символів відображаються в лапках «...», а локальним символьним іменам з таблиці опису змінних блоку передує символ «#».

Вводити лапки або символ «#» немає необхідності. Під час введення програми редактор додає ці символи автоматично.

По всій таблиці символів необхідно використовувати тільки мнемонічні позначення, пропоновані редактором.

4.2 Особливості використання блоків

Організаційні блоки OB (Organization blocks)

Організаційні блоки служать своєрідним *інтерфейсом* між операційною системою й користувальницькою програмою.

Операційна система CPU викликає організаційні блоки при виникненні особливої події, наприклад, під час запуску програми користувача, або при апаратному перериванні.

Головна програма перебуває в організаційному блоці OB1. Інші організаційні блоки також мають постійні призначені номери, прив'язані до певних подій.

Функціональні блоки FB (Function blocks) і блоки даних DB (Data blocks)

Функціональні блоки є частинами програми й створюються для рішення певних завдань. Вони мають область пам'яті для змінних (variable memory), яка розташована в блоці даних DB. Кожному функціональному блоку або, точніше, *виклику* функціонального блоку, призначений свій блок DB. Таким чином, якщо один функціональний блок викликається, наприклад, п'ять разів, то буде створено п'ять екземплярів DB.

Постійно призначений блок даних називається *екземплярним блоком даних, або локальним екземпляром*. Виклик функціонального блоку й екземплярного блоку даних називається *екземпляром виклику* або, для стислості, *екземпляром*. Функціональні блоки можуть зберігати свої змінні не тільки у своєму екземплярному блоці даних, але й в екземплярному блоці даних того блоку, який зробив виклик.

Крім екземплярних блоків даних, існують блоки глобальних даних (global data blocks). Блоки глобальних даних у користувальницькій програмі кодовому блоку не призначаються.

Створюючи блок даних, необхідно визначити, у якій формі будуть зберігатися дані, тобто вказати типи даних. При цьому операційна система контролера відводить у пам‘яті відповідний ресурс для зберігання даних у потрібному форматі. Варто врахувати, що перед створенням екземплярного блоку даних відповідний FB уже *повинен існувати*, інакше неможливо буде вказати *номер FB*, для якого створюється екземплярний блок даних.

Функції FC (Functions)

Функції використовуються для програмування часто повторюваних або складних завдань автоматики. Функція може мати призначені їй параметри, значення яких вона може повертати в блок, який викликав функцію. Функції не зберігають інформацію й не мають призначених блоків даних.

Оскільки FC створюється для формального параметра (деякого образа), а обробляє фактичне значення, то формальні параметри потрібно зіставити фактичним значенням.

Якщо, наприклад, формальному параметру «Start» відповідає вхід «E 3.6» (фактичний параметр), то під час виклику функції вона замінить «Start» значенням, установленим на вході E 3.6. Інакше кажучи, формальні

вхідні й вихідні параметри, використовувані FC, зберігаються як покажчики на фактичні параметри логічного блоку, що викликав FC.

Системні й стандартні блоки

Системні й стандартні блоки є компонентами *операційної системи*. Системні блоки (функції SFC і функціональні блоки SFB) можуть містити дані в системних блоках даних (SDB). Вони забезпечують важливі системні функції, доступні користувачеві, наприклад, функції керування внутрішніми годинниками CPU або комунікаційні функції. При цьому системні й стандартні блоки не займають місця в користувальницькій пам'яті – вони розташовуються в операційній системі. Однак для системних блоків потрібно створювати екземплярні блоки даних і завантажувати їх в CPU як частину програми користувача.

Використання декількох екземплярних DB для одного FB

Схема з окремими екземплярами DB дозволяє за допомогою одного FB керувати декількома однотипними пристроями. Так, наприклад, FB, створений для деякого класу двигунів, може керувати різними двигунами, використовуючи для кожного з них певний набір екземплярів даних. При використанні цього методу для декількох двигунів потрібний тільки один функціональний блок (рис. 4.4).

Використання одного екземплярного DB для декількох екземплярів FB

Екземпляри даних для декількох двигунів можна одночасно передавати в один екземплярний DB. Для цього потрібно запрограмувати виклики в одному FB і описати статичні змінні для окремих екземплярів у розділі описів того FB, який буде викликано, типом FB.

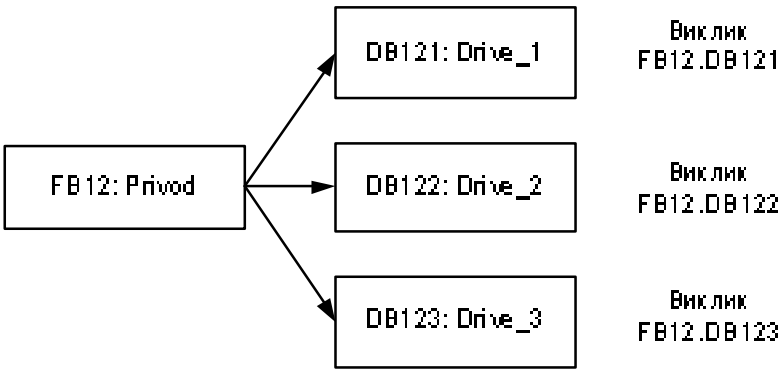


Рисунок 4.4 – Декілька екземплярних блоків даних для одного FB

На рисунку 4.5 виклик здійснює блок FB12 «Privod», а змінні мають тип даних FB13, тобто блока, який викликається. Екземпляр виклику визначається за допомогою призначень Drive_1, Drive_2 і Drive_3.

У цьому прикладі FB13 не має потреби у власному екземплярному блоці даних, тому що дані його екземплярів зберігаються в блоці даних DB20, який належить FB12 – визивному блоку.

При використанні одного екземплярного DB для декількох екземплярів FB заощаджується пам'ять і оптимізується використання блоків даних.

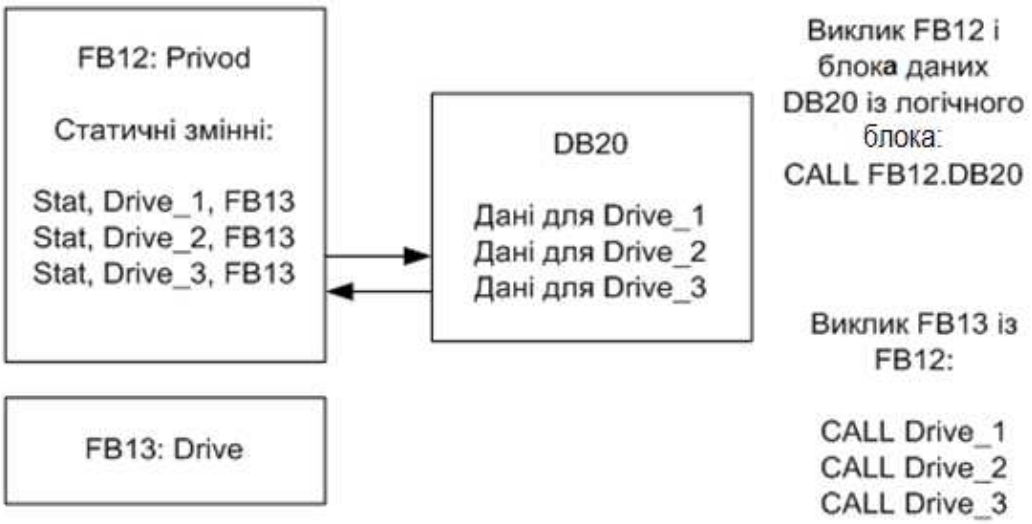


Рисунок 4.5 – Використання одного екземплярного DB для декількох екземплярів FB

Використання глобальних блоків даних

Глобальні блоки даних застосовуються для зберігання користувальницьких даних, до яких можуть звернутися всі інші блоки.

Кожний FB, FC або OB може читати дані із глобального DB або записувати дані в цей DB. Ці дані зберігаються в DB після виходу з нього. Глобальний і екземплярний DB можуть бути відкриті одночасно.

На рисунку 4.6 показані різні методи доступу до блоків даних.

Слід враховувати, що коли викликається логічний блок FC, FB або організаційний блок OB, то він повинен на час виклику зайняти місце в області локальних даних, тобто в L-стеці. При цьому логічний блок відкриває область пам'яті у DB. На відміну від даних, що перебувають в L-стеці, дані в DB не видаляються після завершення роботи логічного блоку.

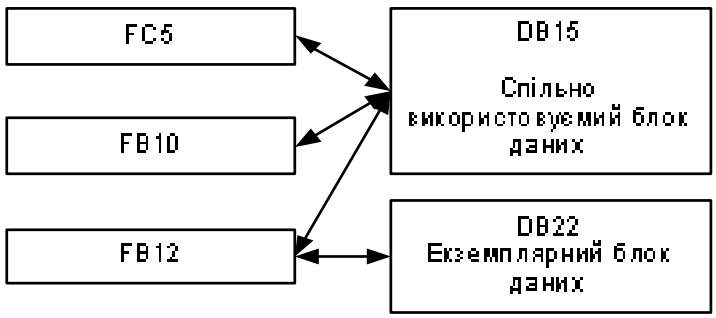


Рисунок 4.6 – Варіанти доступу до блоків даних

4.3 Методика створення логічних блоків

Логічні блоки OB, FB і FC містять у собі три розділи: розділ опису змінних, розділ кодів, а також певні властивості. У таблиці опису змінних

визначаються параметри, їхні системні атрибути й локальні змінні блоку. У розділі кодів створюється користувальницька програма. Властивості блоку містять мітки часу або шлях, що вводиться системою. Крім того, у властивостях блоку можна ввести системні атрибути для блоків, а також його власні деталі, що ставляться до ім'я, сімейства, версії та автора.

У принципі, не має значення, в якому порядку ці частини логічного блоку редагуються.

Результатом описування змінних є таке:

- під час описування локальних даних блоку в L-стеці резервується пам'ять для тимчасових змінних, а при описуванні статичних змінних виділяється пам'ять в екземплярному DB, що приєднується пізніше;
- при призначенні вхідних і вихідних параметрів визначається інтерфейс для виклику блока в програмі;
- при описуванні змінних у функціональному блоці формується також структура даних для кожного екземплярного блоку даних DB, пов'язаного із цим функціональним блоком.

Установлення системних атрибутів передбачає призначення спеціальних властивостей параметрам передачі повідомлень і конфігурації з'єднань, а також визначення функцій взаємодії з оператором.

Таблиця оголошення змінних і розділ кодів логічних блоків тісно пов'язані один з одним, тому що імена з таблиці опису змінних використовуються в розділі кодів. Для контролю зроблених оголошень використовується інтерфейс блока.

Вікно інтерфейсу блока відображає в лівій частині (браузері) список дозволених для декларування типів (IN, OUT, IN_OUT, STAT, TEMP), а у правій частині представляє детальний огляд змінних (рис. 4.7).

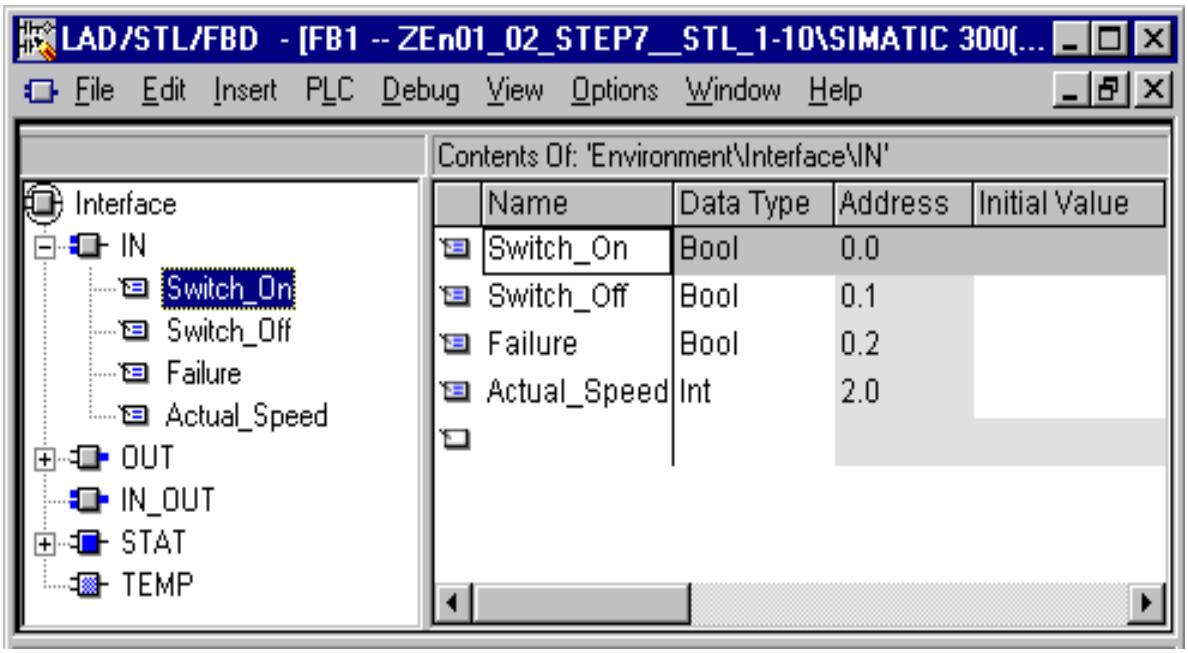


Рисунок 4.7 – Приклад таблиці опису змінних

Програмування послідовності операцій для логічного блока здійснюється в *розділі кодів* шляхом введення відповідних команд. Після введення команди редактор негайно виконує перевірку синтаксису й відображає помилку червоним курсивом.

Розділ кодів логічного блоку зазвичай містить у собі низку сегментів (ланцюгів). Окремі частини розділу кодів можна редагувати в будь-якому порядку. При цьому можна вказати ім'я сегмента, а також ввести коментарі до сегментів або окремих команд.

Завдяки коментарям програма легше читається, що підвищує ефективність пошуку помилок. Коментарі є важливою частиною програмної документації й повинні використовуватися скрізь.

Інтерфейс блоків (Block Interface)

Таблиця оголошення змінних містить інтерфейс блока, який складається з вхідних і вихідних параметрів блока, а також статичних локальних даних. Тимчасові локальні дані не належать інтерфейсу блока. Змінні, які входять до інтерфейсу блока, необхідно ініціалізувати під час виклику блока.

Редактор програм перевіряє відповідність параметрів, заданих при ініціалізації блока, його інтерфейсу. Для цього редактор використовує *мітки часу*. Мітки часу необхідні для того, щоб інтерфейс викликуваного блока створювався раніше, ніж інтерфейс блока, що зробив виклик. Іншими словами, останні зміни інтерфейсу повинні бути виконані раніше його об'єднання із блоком. Редактор програм оновлює мітку часу інтерфейсу у разі зміни кількості параметрів або зміни типу даних, або зміни значень параметрів, прийнятих за замовчуванням.

Конфлікт тимчасових міток (Time stamp conflict)

Якщо інтерфейс викликуваного блока має пізнішу відносно визивного блока тимчасову мітку, виникає «конфлікт тимчасових міток» – Time stamp conflict. Це може відбутися, наприклад, при спробі відкрити для редагування вже скомпільований блок. У цьому випадку редактор виділить некоректний виклик блоку червоним кольором.

Конфлікт тимчасових міток виникає в таких випадках:

- інтерфейс викликуваного блоку має пізнішу тимчасову мітку (younger), ніж код викликуваного блока. Конфлікту не буде, якщо спочатку змінні будуть оголошені, а після цього буде написаний програмний код, у якому ці змінні застосовуються;
- інтерфейс ініціалізації не погоджений із інтерфейсом блока;
- функціональний блок має пізнішу тимчасову мітку, ніж його екземплярний блок даних;
- інтерфейс локального екземпляра має пізнішу тимчасову мітку, ніж екземпляр, який його викликав (стосується функціональних блоків);
- користувальницький тип даних UDT має пізнішу тимчасову мітку, ніж блок, у якому ці змінні оголошені як UDT.

Перевірка блока на консистентність (Check Block Consistency)

Редактор програм лише інформує про наявність конфлікту тимчасових міток. Для цілковитої перевірки програми потрібно використовувати функцію перевірки консистентності блока – Check Block Consistency. Ця функція знімає більшість конфліктів інтерфейсу й указує на місця в програмі, що вимагають редагування.

4.4 Адресація змінних у блоці

При адресації змінних використовуються два основних варіанти:

- абсолютна адресація (absolute addressing);
- символьна адресація (symbol addressing).

Абсолютна адресація змінних

При абсолютній адресації використовуються чисельні адреси, починаючи з адреси «0». Такий принцип застосовується для кожної адресної області.

При символьній адресації використовуються символьні імена, які задаються для глобальних адрес у таблиці символів (Symbol Table), а для локальних – у розділі оголошення змінних (declaration section) усередині блоків.

Розширенням абсолютної адресації є непряма адресація (indirect addressing), при якій адреса (місце розташування) у пам'яті обчислюються під час виконання програми.

Абсолютні адреси входів і виходів розраховуються, виходячи з початкової адреси модуля, що встановлена у таблиці конфігурації, та типу сигналу, що вводиться до модуля.

Дискретний сигнал містить один біт інформації. Прикладами дискретних сигналів є *вхідні* сигнали від кінцевих вимикачів, кнопок і т. п., які надходять на дискретні вхідні модулі, а також *вихідні* сигнали, що керують лампами, контакторами й т. п., які надходять на дискретні вихідні модулі.

Аналоговий сигнал містить 16 біт інформації, що відповідає каналу (channel). Він займає в контролері машинне слово (word), тобто 2 байти. Динамічному діапазону зміни сигналу відповідає динамічний діапазон змінної, у вигляді якої сигнал зберігається й обробляється. Динамічний діапазон зміни сигналу й інтерпретація цього сигналу, наприклад, відносне положення, взяті разом, визначають тип даних для відповідної змінної.

Дискретні сигнали зберігаються в змінних типу BOOL (булева змінна), аналогові сигнали – у змінних типу INT (цілочисельна змінна).

Визначальним фактором для адресації змінної є її тип, від якого залежить необхідна величина області пам'яті для розміщення змінної.

У системі STEP 7 існують 4 типи даних для абсолютної адресації:

1 біт тип даних BOOL;

8 біт тип даних BYTE або інший 8-бітовий тип даних;

16 біт тип даних WORD або інший 16-бітовий тип даних;
32 біта тип даних DWORD або інший 32-бітовий тип даних.

На рисунку 4.8 показана схема адресації для вихідних даних, що представляються бітом, байтом, словом і подвійним словом.

Посилання на змінні типу BOOL здійснюються за допомогою ідентифікатора адреси, номера байта й відділеного десятковою крапкою номера біта. Нумерація байтів починається з нуля у кожній адресній області. Біти усередині байтів нумеруються від 0 до 7.

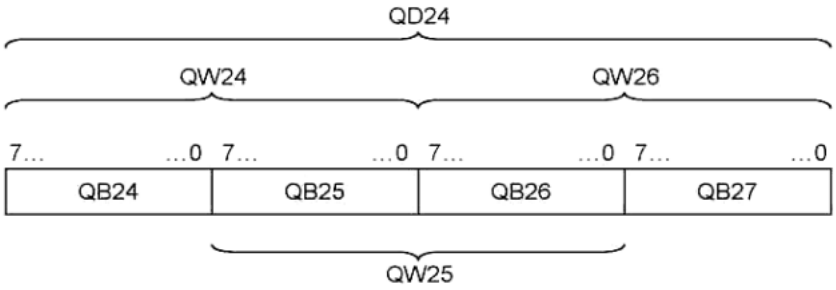


Рисунок 4.8 – Схема адресації вихідних даних, представлених бітом, байтом, словом і подвійним словом

Приклади:

I 1.0 вхідний біт з номером 0 у байті номер 1.

Q 16.4 вихідний біт з номером 4 у байті номер 16.

Для змінних типу BYTE як абсолютна адреса використовується ідентифікатор адреси й номер байта, у якому втримується властиво значення змінної. Ідентифікатор адреси доповнений символом B.

Приклади:

IB 2 вхідний байт номер 2.

QB 18 вихідний байт номер 18.

Змінні типу WORD складаються із двох байтів (слово). Як абсолютна адреса використовується ідентифікатор адреси й номер молодшого байта машинного слова, у якому власно і втримується значення змінної. Ідентифікатор адреси такої змінної доповнюється символом W.

Приклади:

IW 4 вхідне слово номер 4, містить байти 4 і 5.

QW 20 вихідне слово номер 20, містить байти 20 і 21.

Змінні типу DWORD складаються із чотирьох байтів (подвійне слово). Як абсолютна адреса використовується ідентифікатор адреси й номер молодшого байта подвійного слова, у якому власно і втримується значення змінної. Ідентифікатор адреси доповнюється символом D.

Приклади:

ID 8 вхідне подвійне слово містить байти 8, 9, 10 і 11.

QD 24 вихідне подвійне слово містить байти 24, 25, 26 і 27.

При адресації даних DB указується блок даних.

Приклади:

DB 10.DBX 2.0 біт даних 2.0 у блоці даних DB 10.

DB 11.DBB 14 байт даних 14 у блоці даних DB 11.
DB 20.DBW 20 слово даних 20 у блоці даних DB 20.
DB 22.DBD 10 подвійне слово даних 10 у блоці даних DB 22.

Непряма (побічна) адресація

Непряма адресація (indirect addressing) дозволяє розраховувати адреси в області даних під час виконання програми. Мови програмування STL і SCL використовують різні методи для непрямої адресації. У STL розрізняють такі види адресації:

- непряма адресація *за допомогою пам'яті* (Memory-indirect-addressing). Так наприклад, IW [MD 200] означає, що адреса перебуває у подвійному слові пам'яті меркерів;
- непряма *внутрішньозонна адресація за допомогою регістра* (Register-indirect area-internal addressing). Так наприклад, IW [AR1, P#2,0] означає, що адреса із *області входів*, яка перебуває в адресному регістрі AR1, одержує при виконанні оператора збільшення на величину зсуву (offset) P#2,0;
- непряма *міжзонна адресація за допомогою регістра* AR. Так наприклад, W [AR1, P#0,0] означає, що *область і сама адреса* знаходиться в адресному регістрі AR1. При виконанні оператора вона одержує збільшення на величину зсуву (offset) P#0,0.

Подвійні слова адресної області для даних (DBD і DID), меркерів (MD) і тимчасових локальних даних (LD) можуть використовуватися для зберігання адрес при непрямій адресації за допомогою пам'яті.

Непряму адресацію за допомогою регістра можна застосовувати з використанням двох адресних регістрів – AR1 і AR2.

При використанні мови програмування SCL адресні області складаються з поля, елементи якого доступні побічно й окремо.

Наприклад, MW[*index*] – це звертання до слова пам'яті, адреса якого розміщена у змінній *index*. Змінна *index* може визначається у процесі виконання програми.

Символьна адресація змінних

Символьна адресація (symbolic addressing) використовує імена (символи) замість абсолютних адрес. Розроблювач призначає ці імена самостійно. Ім'я повинне починатися з букви й може містити до 24 символів. У STL не дозволено використовувати ключові слова як імена (символи). Для того щоб використовувати ключові слова як імена в SCL, потрібно вставити символ ґрат "#" перед ім'ям.

При присвоєнні імен входам ураховується регістр написання символу. Для імен виходів редактор використовує регістр і нотацію (форму запису), які були застосовані при оголошенні символу.

Символи розрізняються за місцем призначення: глобальні символи дійсні у всій програмі, тоді як локальні символи дійсні тільки в блоці, у розділі оголошення змінних якого вони описані.

У таблиці символів (symbol table) можна призначити *глобальні* символи таким об'єктам:

- блоки даних і кодові блоки;
- входи, виходи, периферійні входи й периферійні виходи;
- меркери, таймери й лічильники;
- користувальницькі типи даних;
- таблиці змінних.

Глобальний символ може також містити пробіли, спеціальні символи й національні символи, наприклад, умляут. Кожний такий символ повинен бути унікальним (однозначно належати одній адресі) у цій програмі.

Імена *локальних* даних визначаються в розділі оголошення змінних відповідного блоку. Ці імена можуть містити тільки букви, цифри й знак підкреслення.

Локальні символи діють тільки усередині блоку, в якому вони описані. Такі ж символи можуть бути застосовані в іншому блоці в іншому контексті для позначення зовсім інших об'єктів. Редактор відображає локальні символи (імена), вставляючи зі спереду символ "#".

У випадку використання *масивів* доступ до окремих елементів масивів забезпечується використанням ім'я масиву з індексом. Так, наприклад, ім'я MSERIES[1] належить першому елементу масиву MSERIES. У випадку програмування на STL індекс повинен бути константою (INT). У випадку програмування на SCL індекс може бути як цілою змінною (INT), так і цілим виразом.

У *структурах* кожний елемент ім'я (subname) відділяється від інших елементів десятковою крапкою, наприклад, FRAME.HEADER.CNUM.

Компоненти користувальницьких типів даних адресуються так само як і компоненти структур.

Символьна адресація даних припускає використання повної адреси, включаючи адресу блоку даних. Наприклад, якщо блок даних із символьною адресою MVALUES містить змінні MVALUE1, MVALUE2 і MTIME, то ці змінні можуть бути адресовані в такий спосіб:

MVALUES. MVALUE1;
MVALUES. MVALUE2;
MVALUES. MTIME.

4.5 Призначення типів даних

Всі дані в програмі користувача повинні бути ідентифіковані типом даних. Доступні такі типи даних:

- елементарні типи даних;
- складені типи даних, які комбінуються з елементарних типів;
- параметричні типи даних.

Команди працюють із об'єктами даних певного розміру. Команди двійкової логіки працюють із бітами. Команди завантаження й передачі у STL, а також команди пересилання в LAD і FBD працюють із байтами (B), словами (W) і подвійними словами (DW).

Елементарні типи даних мають певний розмір, варіант подання (формат) і діапазон. Елементарні типи можуть представлятися в таких форматах:

- тип BOOL представляється одним бітом зі значеннями TRUE і FALSE;
- тип BYTE – це один байт, який представляється шістнадцятирічним числом без знаку, наприклад, B#16#1F;
- тип WORD представляється двома байтами у двійковому (2#0001_0000_0000_0000), шістнадцятирічному (W#16#1000) або BCD (C#157) форматі;
- тип DWORD представляється подвійним словом двійковим, шістнадцятирічним, а також десятковим числом без знака, наприклад, B#(1, 14, 100, 120);
- INT – це ціле десяткове число зі знаком і діапазоном від –32768 до +32767 (розмір 16 біт);
- DINT – це ціле десяткове число зі знаком і діапазоном від –2.147.483.648 до +2.147.483.647 (розмір 32 біта);
- REAL – число із плаваючою крапкою розміром 32 біта й форматом IEEE, наприклад, 1.254567e+12.

Складені типи даних

Складені типи даних визначають групи даних, що займають більше 32 бітів, або групи даних, що складаються з інших типів даних.

STEP 7 допускає такі складені типи даних:

- DATE_AND_TIME – дата й час. Цей тип даних зберігає рік, місяць, день, години, хвилини, секунди, мілісекунди та день тижня;
- STRING – символьний рядок. Цей тип визначає одномірний масив довжиною максимум 254 символів (тип даних CHAR). Символьний рядок може передаватися тільки як одне ціле;
- ARRAY – масив. Цей тип поєднує групу даних одного типу, утворюючи таким чином одне ціле;
- STRUCT – структура поєднує дані різного типу, утворюючи одне ціле;
- UDT – типи даних, обумовлені користувачем.

Дані типу DATE_AND_TIME (DT)

Дані типу DATE_AND_TIME зберігаються у двійково-десятковому форматі в 8 байтах. Можливі два варіанта представлення даних:

- DATE_AND_TIME# 12-25-8:01:1.23;
- DT# 12-25-8:01:1.23.

Для роботи з типом даних DATE_AND_TIME використовуються спеціальні функції стандарту МЕК (ICE):

- перетворення дати й часу доби у формат DATE_AND_TIME за допомогою функції FC3: D_TOD_DT;
- отримання дати з формату DATE_AND_TIME за допомогою функції FC6: DT_DATE;
- отримання дня тижня з формату DATE_AND_TIME за допомогою функції FC7: DT_DAY;
- отримання часу доби з формату DATE_AND_TIME за допомогою функції FC8: DT_TOD.

Масиви

При створенні масиву необхідно зробити таке:

- описати масив за допомогою ключового слова ARRAY та привласнити масиву ім'я;

- визначити розмір масиву, використовуючи індекси, тобто номери першого й останнього елемента за окремими вимірами масиву (максимум 6 вимірів). Індекс вводять у квадратних дужках, розділяючи виміри за допомогою коми, а номери першого й останнього елемента виміру – двома крапками. Наприклад, тривимірний масив буде визначати індекс [1..5, –2..3, 30..32];
- указати тип даних, які повинні втримуватися в масиві.

Звертання до даних у масиві здійснюється через індекс певного елемента в масиві. Індекс використовується в сполученні із символьним ім'ям.

Так, наприклад, якщо масив має ім'я Motor, а елемент масиву – символьне ім'я Heat_2x3, то звертання до другого елемента першої розмірності цього масиву запишеться в такий спосіб:

Motor.Heat_2x3[1,2].

Структури

При визначенні структури необхідно описати дані в DB або в розділі опису змінних логічного блоку.

Таблиця 4.1 ілюструє опис структури MotCont, що складається із двох булевих змінних On і Off, тимчасової затримки Delay типу S5TIME і цілочисельної змінної maxSpeed.

Таблиця 4.1 – Приклад опису структури

Ім'я	Тип даних	Початкове значення	Коментар
MotCont	STRUCT		Змінна простої структури
On	BOOL	FALSE	Змінна MotCont.On
Off	BOOL	TRUE	Змінна MotCont.Off
Delay	S5TIME	S5TIME#5s	Змінна MotCont.Delay
maxSpeed	INT	5000	Змінна MotCont.maxSpeed
	END_STRUCT		

Типи даних, обумовлені користувачем

Типи даних, обумовлені користувачем (UDT), можуть поєднувати елементарні й складені типи даних. Можна привласнювати UDT імена й використовувати їх більше одного разу.

Замість уведення всіх типів даних по одному або у вигляді структури досить лише визначити «UDT20» як тип даних і STEP 7 автоматично виділити відповідний простір пам'яті.

Параметричні типи

Крім елементарних і складених типів даних, використовуються також параметричні типи даних.

До цього типу належать такі дані:

- TIMER або COUNTER визначає конкретний таймер або конкретний лічильник, що буде використовуватися під час виконання блока. При параметризації формальний параметр типу TIMER або COUNTER забезпечується значенням, тобто після введення «T» або «C» уводиться позитивне ціле число;

- BLOCK визначає конкретний блок, використовуваний як вхід або вихід. Опис цього параметра визначає використовуваний тип блока (FB, FC, DB і т. д.) і значення, що задається як фактична адреса блока, наприклад, «FC101» при використанні абсолютної адресації або «Valve» при символьній адресації;
- POINTER указує адресу змінної замість її значення. Так наприклад, покажчик для адресації даних, що починаються з M 50.0, буде мати такий формат: P#M50.0;
- ANY використовується, коли тип даних фактичного параметра невідомий або коли можна використовувати будь-який тип даних.

Обмеження типів даних для блоків

У STEP 7 є обмеження на призначення елементарних, складених і параметричних типів даних.

Оскільки викликати OB не можна, то в OB не може бути вхідних, вихідних або прохідних параметрів. Таким чином, в OB можуть бути тільки *тимчасові* змінні елементарних або складових типів.

Для FB кількість обмежень на призначення типів менша. Під час описування вхідних параметрів FB обмежень немає, але для вихідного параметра не можна повідомляти параметричні типи. Тимчасові змінні можуть мати тип даних ANY. Всі інші параметричні типи заборонені.

Контрольні питання

1. Які функції забезпечує операційна система?
2. Які функції забезпечує програма користувача?
3. Що являє собою *структурування* користувальницької програми?
4. Чим відрізняється структурне програмування від лінійного?
5. Які фази містить циклічна обробка програми?
6. Від чого залежить час виконання циклу?
7. Для чого призначаються організаційні блоки програми?
8. Для чого створюються функціональні блоки FB?
9. Для чого використовуються функції FC?
10. Для чого створюються блоки даних DB?
11. У чому полягають особливості системних функцій та блоків SFC, SFB?
12. У чому полягає різниця між формальними і фактичними параметрами даних?
13. Які варіанти кількісного відношення можна застосовувати при створенні DB і FB?
14. Які основні об'єкти утворюють ієрархію структури проекту?
15. У якому порядку створюються блоки програми?
16. У чому полягає різниця між глобальними і локальними змінними?
17. Які три частини необхідно редагувати при створенні блоків?
18. Що утворюється операційною системою контролера у результаті описування змінних?

19. Яку роль відіграють мітки часу при створенні блоків даних і кодових блоків?

20. За яких умов виникає конфлікт тимчасових міток?

21. Як здійснюється адресація даних?

22. Які дані належать до елементарних типів?

23. Які дані належать до складених типів?

24. Які дані належать до параметричних типів?

5 ПРОГРАМУВАННЯ ПРИСТРОЇВ ЛОГІЧНОГО КЕРУВАННЯ

Для програмування пристроїв логічного керування зазвичай застосовуються текстова мова STL (Statement List) і дві графічні мови – LAD (Ladder Diagram) і FBD (Function Block Diagram). Розроблення програми на вибраній мові здійснюється в одному редакторі – «LAD, STL, FBD: Programming S7 Blocks». Редактор дозволяє переходити від одного представлення програми до другого, наприклад, від представлення програми на мові LAD до представлення на мові FBD чи STL. Такий перехід стає можливим завдяки однаковому набору функцій у цих трьох мовах.

У мовах LAD, STL, FBD використовуються *базові функції, функції для оброблення чисел і функції керування в програмі*.

Базові функції містять у собі двійкові логічні операції, операції з пам'яттю, функції передачі даних, а також функції таймерів і лічильників.

Функції для обробляння чисел – це функції порівняння, математичні й арифметичні функції, а також функції перетворення й зсуву.

Функції керування в програмі містять у собі функції переходів і функції обробляння блоків.

5.1 Програмування двійкових логічних операцій

Особливості бінарної логіки в STL

У мові STL застосовуються двійкові логічні операції A (AND – логічне І), O (OR – логічне АБО) і X (Exclusive OR – Виключальне АБО). Під час обчислення булевих функцій із цими операціями перевіряється результат на рівні логічної одиниці «1». Для перевірки логічного рівня «0» використовуються функції з модифікатором N, тобто функції AN, ON і XN, відповідно.

Слід прийняти до відома, що в схемах SIMATIC S7 стан сигналу має значення «1» за наявності напруги на вході модуля введення, а значення «0», якщо напруга на вході відсутня.

Отже, під час обчислення булевої функції CPU не зв'язує стан сигналу перевіреного біта, а скоріше формує результат перевірки. При цьому у випадках перевірки на стан сигналу «1» результат перевірки ідентичний стану сигналу, а у випадках перевірки на стан сигналу «0» результат перевірки інвертується щодо стану сигналу.

Перевірку стану біта ініціює вираз, що містить оператор перевірки (check statement). Це вираз містить правило логічної операції, тобто алгоритм, відповідно до якого результат перевірки стану біта буде прирівнюватися збереженому у процесорі *результату логічної операції* (RLO).

Результат логічної операції CPU надалі використовує для оброблянння двійкових сигналів. Значення RLO формується й модифікується за допомогою операторів перевірки. Якщо RLO має значення «1», то це означає, що умова двійкової логічної операції виконана, якщо RLO має значення «0», то умова двійкової логічної операції не виконана.

У процесі виконання перевірки логічної операції беруть участь вхідний модуль, вихідний модуль і CPU. Після перевірки стану сигналу (status) датчика, підключеного до каналу вхідного модуля, CPU зв'язує його з результатом логічної операції RLO, що був збережений після виконання попередньої логічної операції, а потім формує відповідний вихідний сигнал. Ця схема показана на рисунку 5.1.

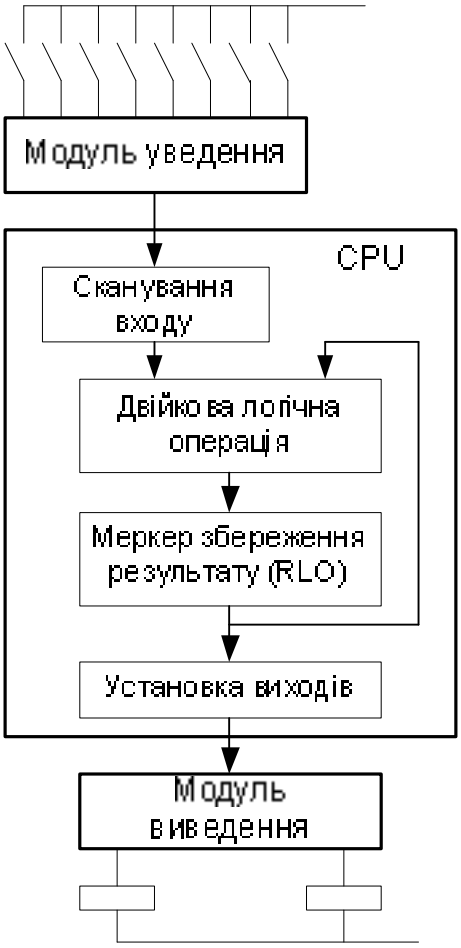


Рисунок 5.1 – Схема виконання перевірки двійкової логічної операції

Результат *поточної* логічної операції запам'ятовується й зберігається як *новий результат логічної операції*. Таким чином, обробляння RLO здійснюється на кожному логічному кроці операції.

Кожний логічний крок складається з виражень із операторами перевірки, які називаються операторами сканування, і виражень із умовними операторами. Перший оператор перевірки, який виконується за умовним оператором, називається *первинним опитуванням* (first check).

Умовні оператори – це такі оператори, виконання яких залежить від результату логічної операції RLO. Ці оператори містять операції призначення (assign), установлення (set) і скидання (reset) двійкових розрядів, а також запуск таймерів і лічильників.

Умовні оператори (за рідкісним винятком) виконуються тільки тоді, коли результат логічної операції RLO має стан «1», і не виконуються, коли

RLO має стан «0». Умовні оператори не впливають на результат логічної операції RLO, і, таким чином, RLO протягом послідовного виконання декількох умовних операторів залишається незмінним.

Приклади логічних кроків з операторами A:

= Q 4.0 умовний оператор (conditional statement), що призначає поточний RLO вихідному сигналу із адресою Q 4.0 (на схемі рис. 5.1 перехід від меркера збереження до установки виходів);

A I 2.0 первинне опитування (first check) – це сканування входів на схемі рис. 5.1;

A I 2.1 оператор перевірки (check statement) – це двійкова логічна операція на рис. 5.1;

= Q 4.3 умовний оператор (conditional statement).

У наведеному прикладі використаний умовний оператор присвоєння «=». Цей оператор призначає результат логічної операції RLO біту, зазначеному у вираженні. Якщо результат логічної операції має значення «1», то біт установлюється, якщо результат логічної операції має значення «0», то біт скидається.

Перший оператор перевірки, який виконується за умовним оператором, має особливе значення, тому що він створює новий результат логічної операції. При цьому *старе* значення результату логічної операції RLO губиться. *Первинне опитування завжди відповідає початку логічної операції*. Алгоритм первинного опитування (AND, OR або Exclusive OR) не грає при цьому ніякій ролі.

Якщо датчик, підключений до входу, має нормально розімкнуті контакти, то значення сигналу, що дорівнює «1», установлюється під час його активації. Якщо датчик має нормально замкнуті контакти, то значення «1» є присутнім на вході в неактивному стані датчика. Тип використовуваного датчика необхідно враховувати під час розроблення програми.

У випадку, якщо один з нормально розімкнутих контактів замінити на нормально замкнутий, то для збереження логіки керування (керування активними значеннями сигналів) потрібно замінити оператор перевірки A на AN. Із цього випливає, що урахування типу датчика проявляється у правильному виборі оператора перевірки.

Під час програмування складних двійкових логічних операцій оператор AND має більш високий пріоритет і виконується перед операторами OR і Exclusive OR, які мають однаковий пріоритет. Для збереження необхідного порядку обчислення складного логічного виразу іноді потрібно тимчасово зберегти значення RLO для деякої точки програми. З цією метою можуть використовуватися вкладені вирази. Як і при записі виразів булевої алгебри, вкладені оператори забезпечують певний порядок виконання функцій.

У мові програмування STL можна використовувати такі вкладені оператори:

O функція OR для функцій AND;

A(відкривальна дужка з функцією AND;

O(відкривальна дужка з функцією OR;
X(відкривальна дужка з функцією Exclusive OR;
AN(відкривальна дужка з функцією NOT-AND;
ON(відкривальна дужка з функцією NOT-OR;
XN(відкривальна дужка з функцією NOT-Exclusive OR;
) закривальна дужка.

Коли CPU зустрічає *відкривальну* дужку, він запам'ятовує поточне значення RLO, а потім обробляє вираз в дужках, тобто вкладений вираз.

Коли CPU зустрічає *закривальну* дужку, він зв'язує значення RLO для вкладеного виразу з раніше запам'ятованим значенням RLO. Це зв'язування здійснюється відповідно до функції, яка записана при відкривальній дужці.

Оператор перевірки, який виконується за *відкритою* дужкою, є *первинним опитуванням*, тому що CPU повинний створити новий результат логічної операції RLO *для вкладеного виразу*.

У практиці обчислення булевих виражень нерідко виникає необхідність в об'єднанні окремих функцій. При об'єднанні AND-функцій оператором OR логічні операції можуть бути записані в булевій алгебрі без використання дужок. Тут діє правило, відповідно до якого функції AND виконуються у першу чергу, а функція OR, що зв'язує функції AND, виконується в другу чергу.

Приклад:

```
A    Key0
A    Key1
O
A    Var1
A    Var2
=    Out1.
```

У цьому прикладі оператор O перебуває між двома функціями AND. Сигнал Out1 установлюється в «1», якщо (Key0 І Key1) АБО (Var1 І Var2) установлені в «1».

При об'єднанні OR оператором AND функції повинні бути записані в булевій алгебрі *з використанням дужок*, за допомогою яких вказується, що функції OR виконуються у першу чергу.

На рисунку 5.2 показано вікно STL-редактора із фрагментом програми, в якій застосовується дужки для зміни порядку формування результату логічної операції RLO.

Особливості операцій бінарної логіки в LAD

Графічна мова програмування LAD заснована на поданні комутаційних схем. Елементами комутаційної схеми є нормально розімкнуті й нормально замкнуті контакти, а також реле. З'єднання цих елементів групується в ланцюг (network). Один або кілька ланцюгів Network 1, Network 2 і т. д. утворюють розділ кодів логічного блоку. На рисунку 5.3 представлено вид вікна LAD-редактора. Треба звернути також увагу на те, що програмний код на рисунках 5.2–5.3 однаковий.

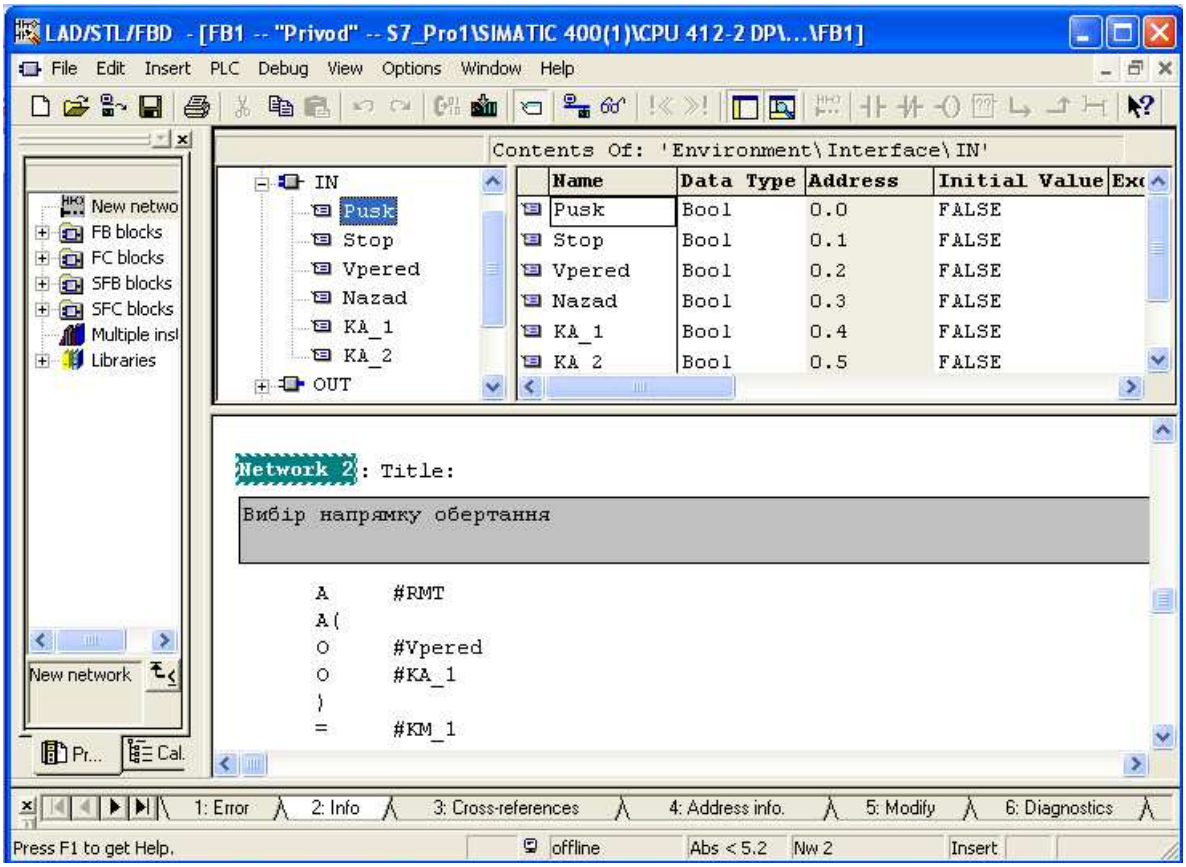


Рисунок 5.2 – Вид вікна програмування на мові STL

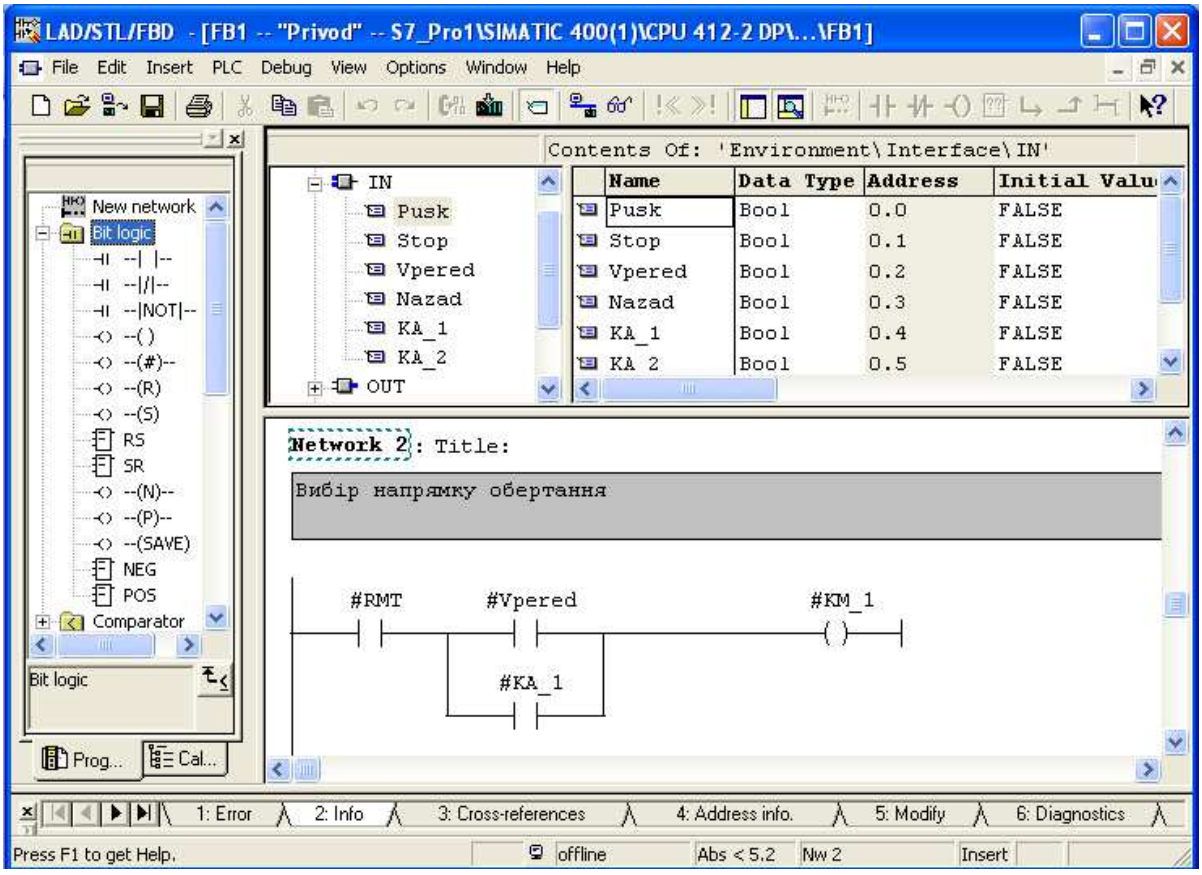


Рисунок 5.3 – Вид вікна програмування на мові LAD

У мові LAD застосовуються тільки два типи контактів – *нормально розімкнуті контакти типу NO*, які скануються з очікуванням стану «1», і *нормально замкнуті контакти типу NC*, які скануються з очікуванням стану «0». Ланцюг може складатися з одного контакту або ж великої кількості з'єднаних контактів. Ланцюг завжди повинен бути завершеним логічною змінною, наприклад, котушкою (coil). Котушка керує двійковим операндом за допомогою *результату логічної операції RLO*.

До операнда можна звернутися через контакт, використовуючи абсолютну або символічну адресу.

Нормально розімкнутий контакт відповідає скануванню з очікуванням сигнального стану «1». Якщо струм тече в точці контактного плану, то це означає, що в цій точці бітова логіка виконується й відповідний двійковий операнд має сигнальний стан «1». Результатом логічної операції (RLO) також є «1».

Операції бінарної логіки в FBD

Мова програмування FBD (Function Block Diagram), інакше функціональний план, заснована на використанні графічних символів логічних елементів, відомих з булевої алгебри й мікросхемотехніки.

Приклад програми у FBD представлений на рисунку 5.4.

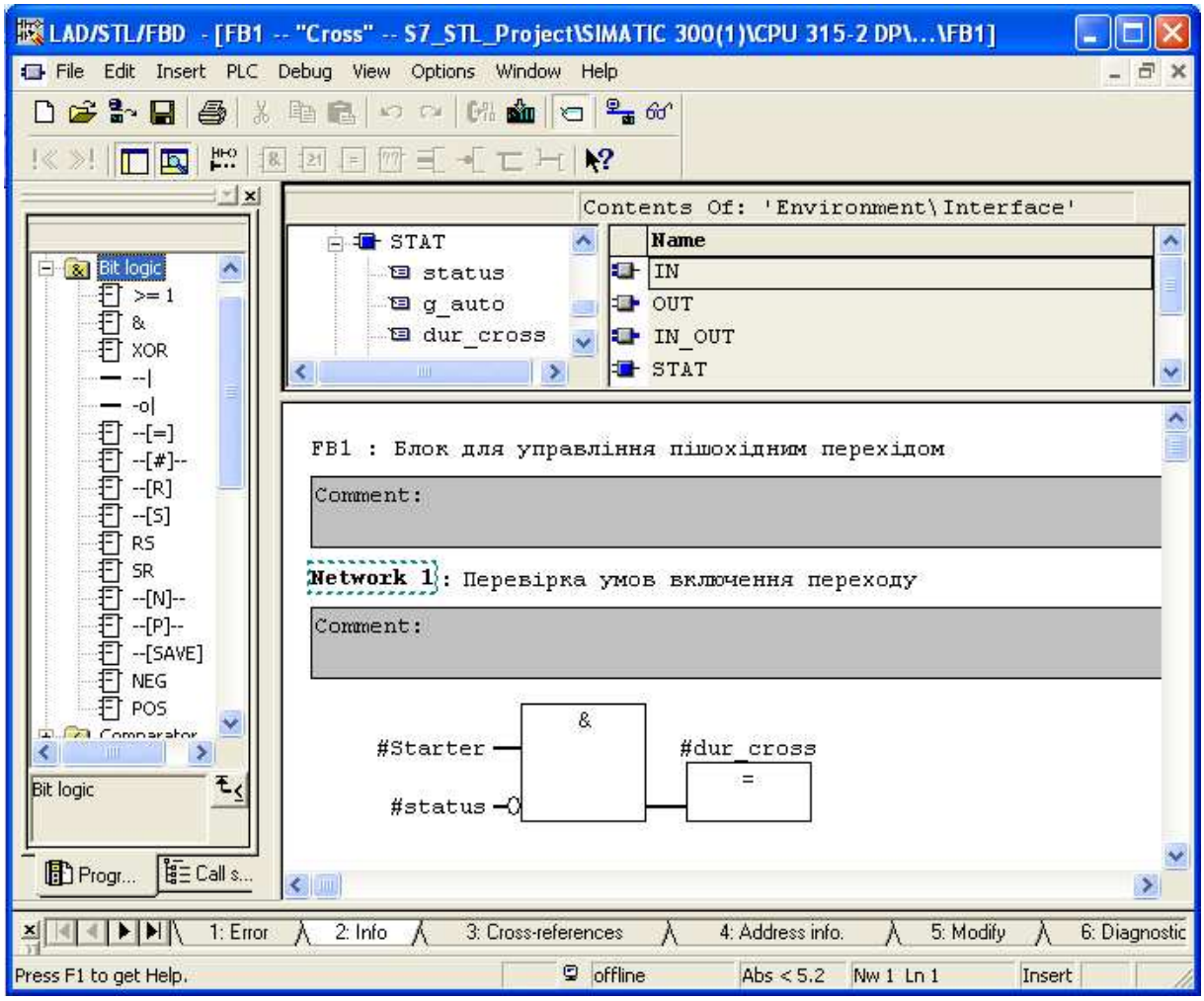


Рисунок 5.4 – Приклад програми в FBD

Для виконання логічних операцій на входи блоків можна підключити такі операнди:

- вхідні й вихідні біти, меркери;
- таймери й лічильники;
- біти глобальних і локальних даних;
- біти слова стану (результати оцінювання й обчислень).

Кожний операнд може бути адресований абсолютно або символічно.

Логічна схема або логічна операція завжди повинна бути завершена оператором, наприклад, присвоювання. У результаті присвоювання бінарний операнд отримує значення результату логічної операції (RLO).

Операнд може скануватися з очікуванням «1» або «0» (рис. 5.5). Сканування з очікуванням «0» розпізнається за символом заперечення на вході функції.

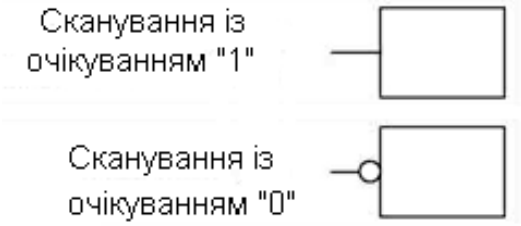


Рисунок 5.5 – Позначення сканування з очікуванням «1» і «0»

З погляду функціональності два методи сканування бінарних операндів дозволяють використовувати NO-контакти й NC-контакти ідентично.

Вихід бінарної функції завжди повинен бути приєднаний до наступного функціонального блоку. У найпростішому випадку можна просто з'єднати вихід із блоковим елементом Assign (*Присвоєння*). Щоб привласнити сигнальний стан бінарного операнда іншому бінарному операнду зазвичай використовується функція AND. При цьому операнд підключається до одного із входів функції, а інший вхід видаляється.

Бінарні функції можна вільно комбінувати одну з іншою, наприклад, можна об'єднати кілька функцій AND за функцією OR. Кількість функцій у логічній операції (схемі або ланцюгу) теоретично не обмежено.

Приклад комбінації операцій бінарної логіки представлений на рисунку 5.6.

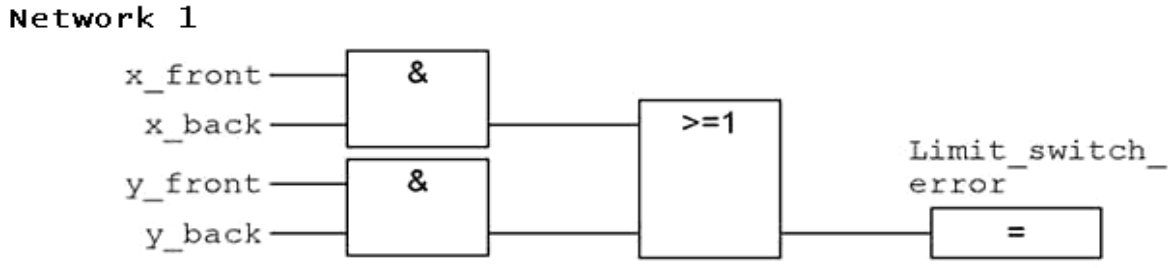


Рисунок 5.6 – Схема виявлення помилки кінцевих вимикачів

У ланцюгу Network 1 здійснюється спостереження кінцевих вимикачів (limit switches) осей X і Y, які не можуть бути активовані попарно. Схема формує повідомлення про помилку роботи кінцевих вимикачів.

Інвертування (заперечення) результату логічної операції можна використовувати при скануванні бінарного операнда, що еквівалентно скануванню з очікуванням сигнального стану «0», а також на виході бінарної функції, коли умова не виконана, тобто коли RLO = 0.

5.2 Програмування операції з пам'яттю та передачі даних

Операції з пам‘яттю в STL

Операції з пам‘яттю використовуються в сполученні з двійковими логічними операціями, щоб впливати на значення сигналів (стану) бітів з використанням результату логічної операції RLO.

До операцій з пам‘яттю належать:

- функція Assign (*Присвоєння*) для *динамічного* керування бітами;
- функції *статичного* керування бітами – Set (*Установлення біта*) і Reset (*Скидання біта*);
- функції перевірки наявності фронту сигналу.

Функція Assign має такий синтаксис:

= *Bit*.

Ця функція привласнює RLO вказаному біту.

Функції Set і Reset мають синтаксис:

S *Bit;*

R *Bit*.

Функції Set і Reset виконуються тільки у випадку, якщо результат логічної операції RLO має значення «1». Якщо результат логічної операції «0», то інструкції Set і Reset не змінюють стани біта, зазначеного як операнд у цих інструкціях.

Для перевірки наявності фронту сигналу використовуються дві функції:

FP *Bit* – функція перевірки наявності переднього (зростаючого) фронту сигналу;

FN *Bit* – функція перевірки наявності заднього (спадного) фронту сигналу.

Наявність переднього фронту сигналу свідчить про перехід сигналу від рівня «0» до рівня «1». Відповідно, наявність заднього фронту сигналу свідчить про перехід сигналу від рівня «1» до рівня «0».

У логічних перемикаючих схемах еквівалентом функції перевірки наявності фронту сигналу є *контактний формувач імпульсу*. Під час включення реле цей формувач генерує імпульс, що свідчить про наявність зростаючого фронту сигналу. Під час вимикання реле цей формувач генерує імпульс, що свідчить про наявність спадного фронту сигналу.

До біта, зазначеного як операнд, звертаються як до «меркера фронту». Стан цього сигналу може бути перевірено в будь-який момент

у наступних циклах сканування програми. Бітом у функції перевірки наявності фронту сигналу може бути меркер, біт із блоку глобальних даних або біт зі статичних локальних даних у функціональних блоках.

Отже, меркер фронту зберігає «старе» значення RLO, яке CPU використовував при останній перевірці наявності фронту сигналу. Під час кожної нової перевірки наявності фронту сигналу CPU порівнює поточне значення RLO зі станом меркера фронту. Фронт сигналу буде виявлений, якщо сигнали першої й другої перевірки будуть мати різні стани. Таким чином, стан біта «1» означає факт виявлення фронту сигналу. Цей стан біта зберігається, як правило, протягом одного циклу сканування програми, тому що при наступній перевірці CPU фронту сигналу не буде.

Перевірка наявності фронту у двійковій логічній операції може бути використана для керування таймером, лічильником або операцією з пам'яттю. Двійкові операції перевірки можуть розташовуватися між операцією перевірки наявності фронту і функцією, яка запускається.

Приклад:

```
O     Var_1
O     Var_2
FP    Merker_1
A     Var_3
S     Out_1
A     Var_4
FN    Merker_2
R     Out_1.
```

У прикладі вихід Out_1 установлюється в момент, коли виконується OR-умова й вхід Var_3 установлений в «1». Вихід Out_1 скидається в момент, коли приходить негативний фронт на вхід Var_4.

Функції передачі даних у STL

У всіх операціях передачі даних використовується акумулятор – спеціальний регістр у процесорі, який виконує функції проміжного буфера.

Напрямок, у якому відбувається передача даних, вказується у використовуваній для передачі інструкції.

Функція завантаження складається з оператора L (код операції завантаження) і константи, змінної або адреси з ідентифікатором адреси, вміст якого функція буде завантажувати в акумулятор 1.

Приклади:

```
L     +1500       // завантаження константи;
L     IW 32       // завантаження слова із прямою адресацією;
L     ActVal      // завантаження змінної (символьна адресація).
```

CPU виконує функцію завантаження незалежно від результату логічної операції RLO і бітів стану. Функція завантаження не впливає на результат логічної операції й не впливає на біти стану. При завантаженні адреси, константи або змінної в акумулятор 1, поточний вміст акумулятора 1 пересилається в акумулятор 2. Попередній вміст акумулятора 2 при цьому губиться.

Функція вивантаження складається з оператора T і адреси в області пам'яті, по якій дані повинні бути відправлені з акумулятора 1.

Приклади:

```
T    MW120     //переносить вміст акумулятора в область пам‘яті
               //меркерів (байти 120 та 121), тобто абсолютна адресація
T    Setpoint  //переносить вміст акумулятора в змінну
               //(символьна адресація).
```

CPU виконує функцію вивантаження незалежно від результату логічної операції RLO і бітів стану. Функція вивантаження не впливає на результат логічної операції й не впливає на біти стану.

Функція вивантаження пересилає вміст акумулятора 1 по одному байту, слову, або подвійному слову. При цьому вміст акумулятора 1 залишається незмінним (копіюється), що дозволяє багаторазово пересилати дані з акумулятора 1.

Для одночасного оброблення двох числових значень потрібні два проміжних буфери. У ролі таких буферів виступають акумулятор 1 і акумулятор 2. Всі CPU мають такі спеціальні регістри. Крім того, CPU S7-400 мають два додаткові проміжні буфери – акумулятор 3 і акумулятор 4, які використовуються переважно в арифметичних операціях.

Кожний акумулятор містить 32 розряди, тоді як всі області пам'яті мають байтову структуру. Обмін інформацією між областями пам'яті й акумулятором 1 може відбуватися побайтно, по 1 машинному слову й по 1 подвійному машинному слову.

В операціях пересилання можуть брати участь:

IB	вхідний байт;
IW	вхідне слово;
ID	вхідне подвійне слово;
QB	вихідний байт;
QW	вихідне слово;
QD	вихідне подвійне слово;
MB	байт меркерів;
MW	слово меркерів;
MD	подвійне слово меркерів;
LB	байт локальних даних;
LW	слово локальних даних;
LD	подвійне слово локальних даних;
DBB	байт глобальних даних;
DBW	слово глобальних даних;
DBD	подвійне слово глобальних даних;
DIB	байт в екземплярному DB;
DIW	слово в екземплярному DB;
DID	подвійне слово в екземплярному DB;
STW	слово стану.

Функції акумуляторів дозволяють пересилати значення з одного акумулятора в інший або переміщати байти усередині акумулятора 1.

До функцій акумуляторів належать:

PUSH зсув вмісту акумулятора «вперед»;

POP зсув вмісту акумулятора «назад»;

TAK обмін вмістом між акумуляторами 1 і 2;

ENT зсув вмісту акумулятора «вперед» (без акумулятора 1);

LEAVE зсув вмісту акумулятора «назад» (без акумулятора 1).

Перші три функції PUSH, POP і TAK використовуються в CPU, які мають тільки два акумулятори (S7-300 CPU). Всі п'ять функцій використовуються в CPU, що мають чотири акумулятори (S7-400 CPU).

Функції обміну байтами в акумуляторі 1:

CAW міняє місцями два байти в молодшому слові акумулятора 1. При цьому байти старшого слова акумулятора залишаються незмінними.

CAD міняє місцями всі байти в акумуляторі 1. При цьому самий старший байт стає самим молодшим за номером, а середні два байти міняються місцями.

Для пересилання даних застосовуються також системні функції SFC 20 BLKMOV (копіювання області даних), SFC 21 FILL (вставка даних в область призначення) та SFC 81 UBLKMOV (безперервне копіювання області даних).

Елементи пам'яті LAD

Для роботи з пам'яттю в мові LAD доступні такі функції:

- одиночна котушка, якій привласнений (призначений) RLO;
- котушки R і S як операції з пам'яттю;
- блокові елементи RS і SR як функції, які працюють із пам'яттю;
- коннектори (midline outputs) як проміжні буфери;
- котушки P и N як елементи виявлення фронту імпульсу;
- блокові елементи POS і NEG як елементи виявлення фронту операндів.

Одиночна котушка як термінатор, тобто завершальний елемент ланцюга, призначає потік електроенергії (електричний струм) прямо операнду, розташованому при котушці. Котушки установлення й скидання S і R (set coil, reset coil) також можуть завершувати ланцюг.

Позначення котушок наведені на рисунку 5.7.

Рисунок 5.7 – Позначення котушок установлення й скидання

Якщо струм тече в котушці установлення S, то операнд, записаний над котушкою, установлюється в сигнальний стан «1». Якщо струм тече в котушці скидання, тоді операнд над котушкою має сигнальний стан «0». За відсутності струму в котушці установлення або скидання бінарний операнд залишається незмінним.

Функції котушок установлення й скидання можуть бути об'єднані в блоковому елементі функції для роботи із блоком пам'яті (memory box), як показано на рисунку 5.8. Загальний бінарний операнд розташовується над блоковим елементом.

Рисунок 5.8 – Блокові елементи пам'яті SR і RS

Вхід S (set input) блокового елемента в цьому випадку відповідає котушці установлення, вхід R (reset input) – котушці скидання. Сигнальний стан двійкового операнда, призначеного функції для роботи з пам'яттю, перебуває на виході Q функції пам'яті.

Блокові елементи SR і RS відрізняються пріоритетом входу скидання. Якщо обидва входи одночасно дорівнюють «1», то функції пам'яті реагують по-різному – функція пам'яті SR скидається, а функція пам'яті RS установлюється. Оскільки оператори виконуються послідовно, то в блоковому елементі SR CPU установить операнд пам'яті, тому що цей вхід обробляється першим, однак потім знову скине його, коли обробляє вхід скидання. Функція пам'яті із пріоритетом скидання є «нормальною» формою функції для роботи з пам'яттю, тому що стан скидання (сигнальний стан «0») зазвичай безпечніше або менш ризикований стан.

Якщо операнд пам'яті є виходом, то його установлення відображається тільки у вихідній таблиці образу процесу. CPU не передає вихідну таблицю образу процесу в модулі виходів до початку наступного програмного циклу.

Коннектори (midline outputs) є проміжними буферами в контактному або функціональному планах. Коннектор є одиночною котушкою в ланцюзі. Двійковий операнд над коннектором зберігає RLO для цієї точки. Сам коннектор не робить впливу на електричний струм.

Позначення коннектора наведено на рисунку 5.9.

Рисунок 5.9 – Позначення коннектора в LAD і FBD

Сканувати бінарний операнд над коннектором можна в іншій точці програми за допомогою NO- і NC-контактів (рис. 5.10). В одному ланцюзі можуть бути запрограмовані декілька коннекторів.

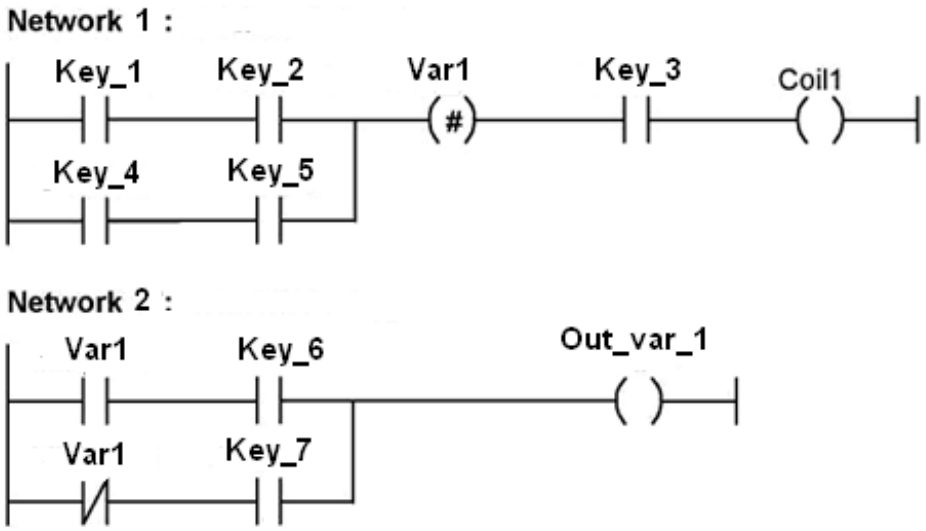


Рисунок 5.10 – Приклад використання коннектора в LAD

Блокові елементи пам'яті у FBD

У FBD блокові елементи пам'яті використовуються з метою впливу на сигнальні стани бінарних операндів за допомогою результату логічної операції RLO, що генерується CPU.

Для роботи з пам'яттю доступні такі функції:

- блоковий елемент присвоювання (assign box);
- блокові елементи установлення S (set) і скидання R (reset) як індивідуально програмувальні функції пам'яті;
- блокові елементи RS і SR як цілком закінчені функції пам'яті.
- блоковий елемент коннектора (midline output box) як проміжний буфер;
- блокові елементи P и N як елементи оцінювання (виявлення) фронту результату логічної операції.
- блокові елементи POS і NEG як визначники фронту операндів.

Блоковий елемент присвоювання як термінатор ланцюга привласнює результат логічної операції безпосередньо операнду, суміжному із блоковим елементом.

Позначення блокового елемента присвоювання наведено на рис. 5.11.

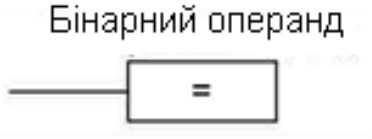


Рисунок 5.11 – Позначення блокового елемента присвоювання

Блокові елементи установлення й скидання також можуть завершувати логічну операцію (рис. 5.12). Ці блокові елементи активуються тільки тоді, коли результат логічної операції, що направляється в блоковий елемент, дорівнює «1».

Рисунок 5.12 – Позначення блокових елементів установлення й скидання

Елементи оцінювання фронту імпульсу в LAD і FBD

Виявлення фронту сигналу відбувається в програмі шляхом порівняння поточного RLO зі збереженим RLO. Якщо сигнальні стани різні, то це є наслідком фронту сигналу.

Для оцінювання фронту застосовуються чотири елементи (рис. 5.13):

- меркер позитивного фронту;
- меркер негативного фронту;
- позитивний фронт операнда;
- негативний фронт операнда.

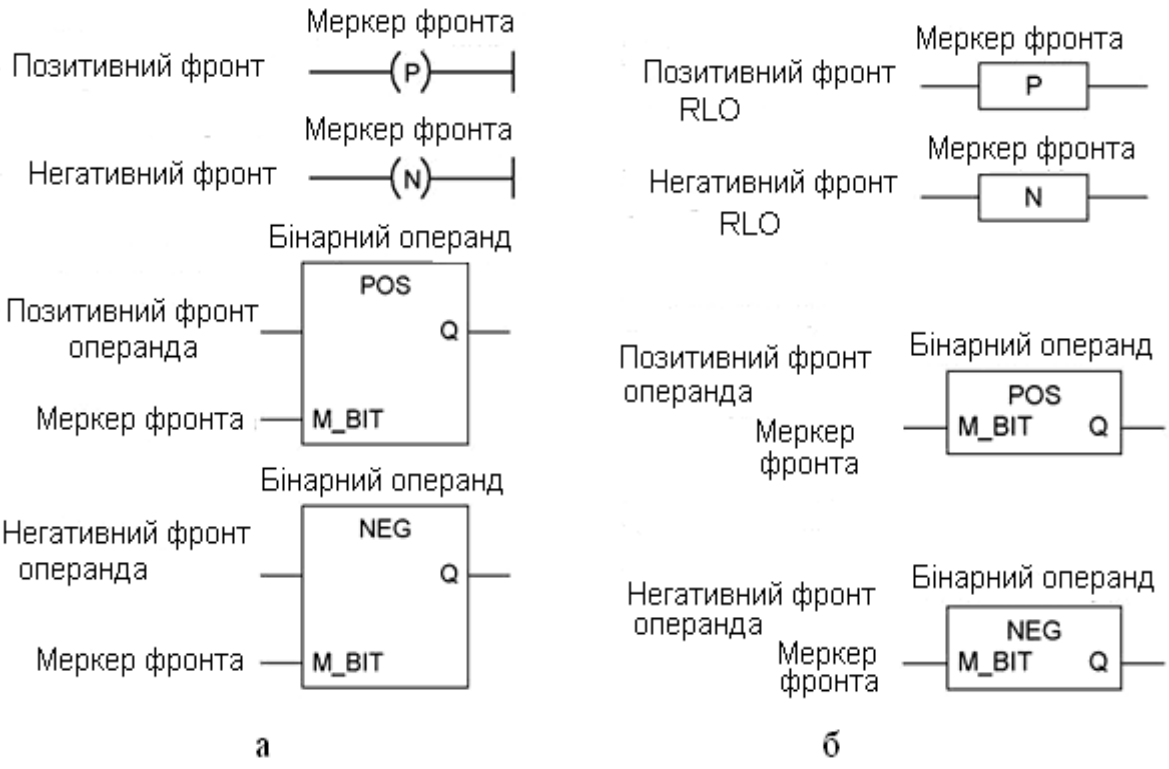


Рисунок 5.13 – Елементи оцінювання фронту в LAD (а) і FBD (б)

5.3 Програмування таймерів

Таймери використовуються для керування за часом, наприклад, для забезпечення заданого часу очікування, виміру відрізків часу або генерації імпульсів.

Користувачеві доступні такі типи таймерів:

- таймер з керованим імпульсом (Pulse timer);
- таймер з розширеним імпульсом (Extended pulse timer);
- таймер із затримкою включення (On-delay timer);
- таймер із затримкою включення і пам'яттю (Retentive On-delay timer);
- таймер із затримкою вимикання (Off-delay timer).

Позначення типів і тимчасові діаграми таймерів наведені на рисунку 5.14.

Інтерфейс таймера складають вхідні сигнали запускання, скидання, та завдання тривалості, а також вихідні сигнали стану таймера та його поточних значень. Призначення входів і виходів наведено у таблиці 5.1.

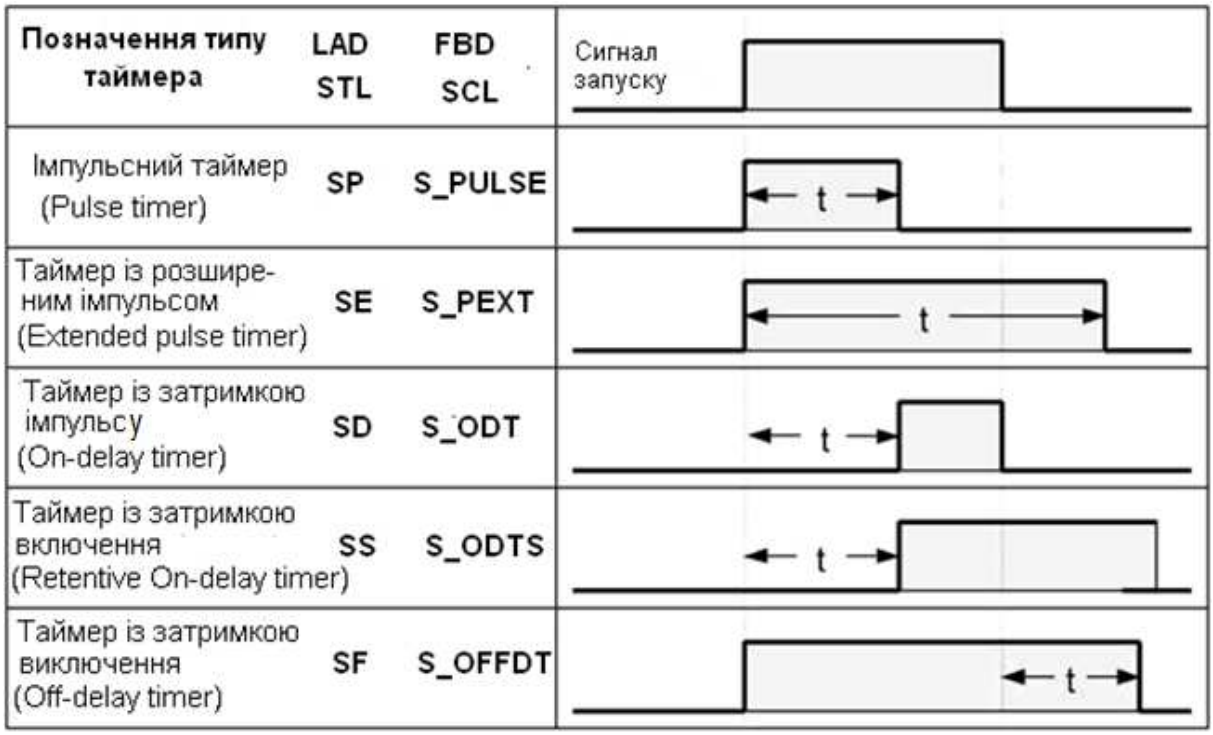


Рисунок 5.14 – Тимчасові діаграми таймерів

Таблиця 5.1 – Призначення входів і виходів таймера

Назва	Тип даних	Опис
S	BOOL	Вхід запуску
TV	S5TIME	Специфікація тривалості
R	BOOL	Вхід скидання
BI	WORD	Поточне значення у двійковому коді
BCD	WORD	Поточне значення в BCD-форматі
Q	BOOL	Стан таймера

Особливості використання таймерів у мові STL

Таймер існує у STL-програмі як блоковий елемент. Завантаження значення часу здійснюється через акумулятор 1.

Таймер запускається, якщо відбулася зміна значення результату логічної операції RLO. При цьому для таймера із затримкою вимикання (Off-delay timer) RLO повинен змінити свій стан зі значення «1» на «0», а для всіх інших типів таймерів RLO повинен змінити свій стан зі значення «0» на «1».

Під час запускання таймера з акумулятора 1 вибирається час запуску (running time) або тривалість роботи (duration). Ці параметри рекомендується завантажувати в акумулятор 1 безпосередньо перед запусканням функції таймера або у вигляді константи, або у вигляді змінної.

Приклади завдання тривалості імпульсу таймера константою:

```
L     S5TIME#10s   // Завантажити тривалість 10 с
L     S5T#1m10ms   // Завантажити тривалість 1 хв 10 мс.
```

Приклади завдання тривалості імпульсу таймера змінною:

```
L     S5T#10m      //Задати тривалість 10 хв
T     MW20         //Зберегти тривалість роботи
L     MW20         //Завантажити тривалість роботи.
```

Внутрішня структура тимчасового параметра «тривалість імпульсу» складається зі значення часу й тимчасової бази. Тривалість імпульсу таймера дорівнює добутку цих величин.

Значення тимчасової бази (величин кроку за часом) використовується операційною системою CPU для декрементування таймера (рис. 5.15).

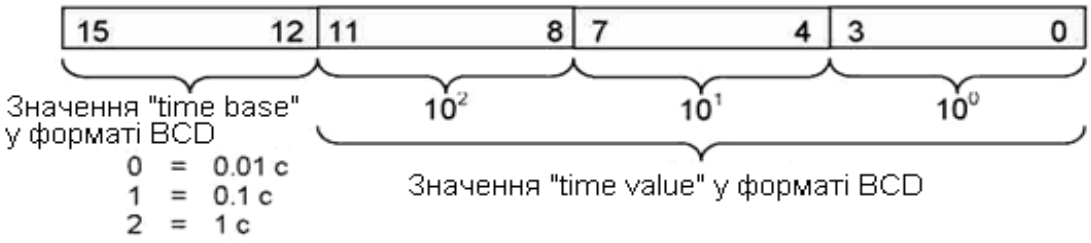


Рисунок 5.15 – Формування параметра «тривалість роботи»

Таким чином, зменшення значення тимчасової бази забезпечує більш точне обчислення проміжків часу, тому що CPU робить східчасте (декрементне) зменшення значення функції таймера (time value) відповідно до заданого значення тимчасової бази (time base).

Варто враховувати, що таймери можуть обновлятися асинхронно стосовно процесу сканування програми користувача. При цьому стан таймера на початку циклу сканування відрізняється від його стану наприкінці циклу. Щоб зменшити помилку через асинхронне відновлення таймера у програмі, таймер необхідно обновляти тільки в одному місці програми.

Скидання таймера виконується за інструкцією:

```
R     T n.
```

За цією інструкцією таймер скидається при результаті логічної операції RLO, що дорівнює «1».

Запустити знову таймер можна інструкцією:

FR T n.

В операційну систему CPU вбудовані IEC-функції таймерів. Вони доступні як системні функціональні блоки SFB:

- SFB 3 TP – генерація імпульсів;
- SFB 4 TON – генерація імпульсу із затримкою включення;
- SFB 5 TOF – генерація імпульсу із затримкою вимикання.

Параметри IEC-функцій таймерів представлені в табл. 5.2.

Таблиця 5.2 – Параметри IEC-функцій таймерів

Назва	Призначення	Тип даних	Опис
IN	INPUT	BOOL	Вхід запуску (Start input)
PT	INPUT	TIME	Тривалість імпульсу (Pulse length) або затримка включення (Delay duration)
Q	INPUT	BOOL	Стан таймера (Timer status)
ET	INPUT	TIME	Минулий час (Elapsed time)

Особливості програмування таймерів у мовах LAD і FBD

У програмах LAD і FBD таймер представляється або як блоковий елемент, як показано на рисунку 5.16, або окремими програмними елементами.

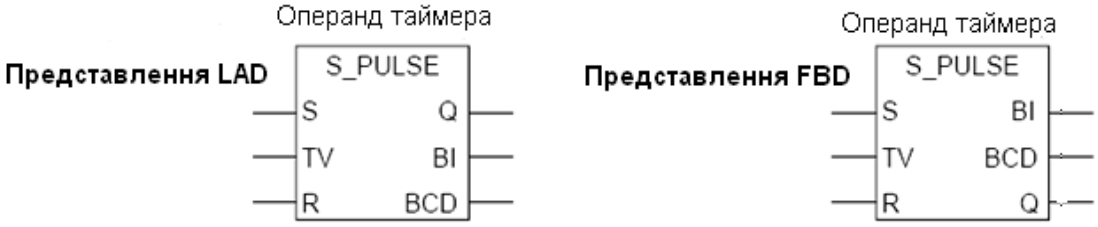


Рисунок 5.16 – Представлення таймера блоковим елементом

Представлення окремими елементами LAD показано на рисунку 5.17, а представлення окремими елементами в FBD – на рисунку 5.18.

Над блоковим елементом розташована абсолютна або символічна адреса таймера. У самому блоковому елементі як заголовок зазначений режим таймера (S_PULSE означає Start pulse або запуск імпульсу).

Таймер запускається, коли результат логічної операції (RLO) змінюється на вході запуску S (start input) блокового елемента або перед котушкою. Таймери запускаються при зміні RLO з «0» на «1», однак для таймера *затримки вимикання* RLO повинен помінятися з «1» на «0».

Рисунок 5.17 – Представлення таймера окремими елементами в LAD

Рисунок 5.18 – Представлення таймера окремими елементами в FBD

Як завдання тривалості таймер приймає значення, зазначене під котушкою (блоковим елементом) або на вході TV. Задати тривалість можна константою, операндом розміром у слово або змінною типу S5TIME. Слід прийняти до уваги, що призначення для входів S і TV обов'язкові.

На рисунку 5.19 показаний приклад LAD-програми запуску таймера T1 з часом, що визначений операндом «Del_1».

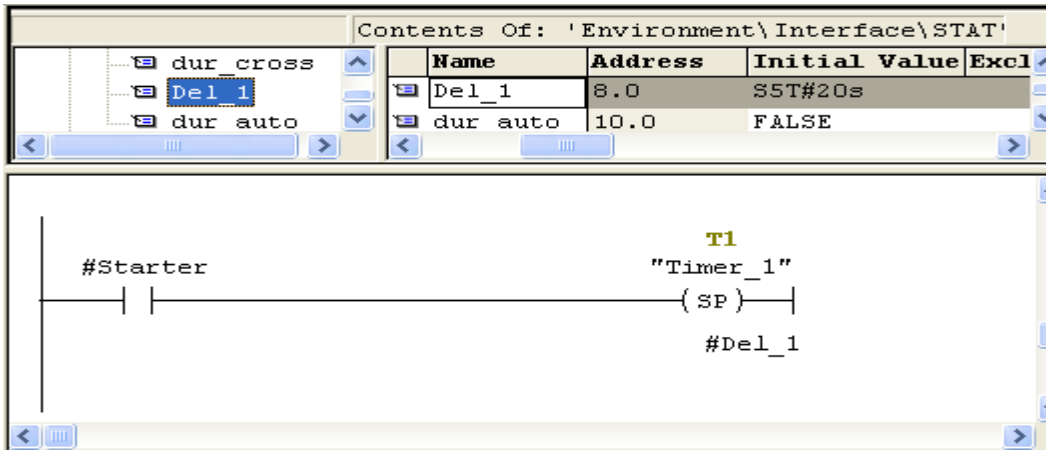


Рисунок 5.19 – Приклад програми запуску таймера, представленого котушкою

У LAD-програмі таймер скидається, коли електричний струм тече на вході скидання або в котушці скидання. У FBD-програмі таймер скидається, коли на вході скидання присутній сигнал «1». У скинутому стані перевірка стану таймера поверне «0» (інвертоване сканування поверне «1»).

Виходи BI і BCD (рис. 5.16) надають значення часу таймера у двійковому (BI) і двійково-десятковому (BCD) виді. Значення виходу є поточним у момент зчитування. Якщо таймер активний, то значення часу відлічується від установленого убік зменшення до нуля. Значення зазвичай зберігається в заданому операнді, тобто передається як до блокового елемента MOVE.

5.4 Програмування лічильників

Функції лічильників дозволяють вирішувати завдання рахунку безпосередньо в CPU. Лічильники дозволяють виконувати прямий і зворотний рахунок. Під час виконання рахунку використовується «тридекадний» формат значення лічильника, що визначає діапазон значень від 000 до 999.

Швидкість рахунку лічильників залежить від часу сканування програми. Це пов'язано з тим, що для забезпечення процесу рахунку CPU повинен виявляти на вході лічильника зміни вхідного сигналу, інакше кажучи, вхідний імпульс (або пауза) на вході повинний бути присутнім принаймні один цикл сканування програми. Таким чином, чим триваліше цикл сканування програми, тим повільніше швидкість рахунку лічильника.

Лічильник представляється, зазвичай, блоковим елементом, який показано на рисунку 5.20.

Рисунок 5.20 – Блоковий елемент лічильника

Блоковий елемент лічильника містить всі операції рахунку у формі функціональних входів і функціональних виходів. Над блоковим елементом розташована абсолютна або символічна адреса лічильника. У блоковому елементі як заголовок вказується тип лічильника. Наприклад, S_CUD означає up-down counter – лічильник прямого і зворотного рахунку.

Призначення входів і виходів лічильника представлені в таблиці 5.3.

Таблиця 5.3

Назва	Тип даних	Опис
CU	BOOL	Вхід прямого рахунку
CD	BOOL	Вхід зворотного рахунку
S	BOOL	Вхід установки
PV	WORD	Вхід передустановки
R	BOOL	Вхід скидання
CV	WORD	Поточне значення у двійковому коді
CV_BCD	WORD	Поточне значення в BCD-коді
Q	BOOL	Стан лічильника

Програмування лічильника мовою STL

Під час програмування лічильника можна включати режим прямого або зворотного рахунку, скидати лічильник, задавати для лічильника початкове значення (initial value). Існує також можливість визначати стан лічильника.

Установлення лічильника здійснюється за інструкцією:

S C n.

Лічильник установлюється, якщо RLO переходить від «0» до «1» перед операцією установлення S, тобто для установлення лічильника завжди потрібен позитивний фронт.

Установити лічильник – це значить завантажити в лічильник початкове значення. Початкове значення (від 0 до 999), що завантажується в лічильник, повинно перебувати в акумуляторі 1.

Для завдання константи можна використовувати C# або W#16# (тільки з десятковими числами).

Приклад:

```
L     C#200          //Завантажити значення лічильника 200
T     MW 36          //Зберегти значення лічильника.
```

Скидання лічильника здійснюється командою:

R C n.

Лічильник скидається, якщо RLO має значення «1» перед тим, як у програмі зустрінеться операція скидання R лічильника (Reset). Скидання встановлює лічильник у нульове значення.

Прямий рахунок виконується за інструкцією:

CU C n.

Зворотний рахунок виконується за інструкцією:

CD C n.

Приклади програмування лічильника під номером C2:

1) установлення прямого рахунку лічильника C2:

```
A   I 2.1
CU  C 2
```

2) установлення зворотного рахунку лічильника C2:

```
A     I 2.2
CD    C 2
```
3) установлення лічильника С2 із завантаженням константи 500:
```
A     I 2.3
L     C#500
S     C 2
```
4) скидання лічильника С2:
```
A     I 2.4
R     C 2
```
5) опитування лічильника С2 з передачею рівня на вихід Q 13.0:
```
A     C 2
=     Q 13.0.
```

Особливості програмування лічильника в LAD і FBD

Лічильник установлюється, коли сигнал на вході установки S змінюється з «0» на «1».

Коли лічильник установлюється, він приймає (як значення рахунку) значення на вході PV або значення під котушкою установки.

Лічильник скидається, коли струм протікає на вході скидання або в котушці скидання. У цьому випадку перевірка лічильника NO-контактом поверне результат зчитування «0», а перевірка NC-контактом поверне «1».

Стан лічильника подається на вихід Q блокового елемента лічильника. Стан лічильника можна перевірити з використанням NO-контакту або NC-контакту. Вихід Q містить «1», коли поточне значення рахунку більше нуля. На виході Q утримується «0», якщо поточне значення рахунку дорівнює нулю. Вихід Q у блоковому елементі лічильника можна не підключати.

IEC-функції лічильників

IEC-функції лічильників (IEC Counter Functions) убудовані в операційну систему CPU як системні функціональні блоки SFB.

Під час розроблення програми доступні такі функції лічильників:

- SFB 0 CTU – функція прямого рахунку;
- SFB 1 CTD – функція зворотного рахунку;
- SFB 2 CTUD – функція прямого і зворотного рахунку.

Ці SFB можна викликати з екземплярними блоками даних або використовувати їх як локальні екземпляри у функціональному блоці.

5.5 Використання функцій порівняння

Функції порівняння використовуються при обробці чисел типів INT, DINT і REAL. Перелік та позначення функцій порівняння наведено у таблиці 5.4.

Таблиця 5.4 – Функції порівняння для даних типу INT, DINT і REAL

Операція порівняння	Тип даних		
	INT	DINT	REAL
дорівнює	==I	==D	==R
Не дорівнює	<>I	<>D	<>R
Більше ніж	>I	>D	>R
Більше ніж або дорівнює	>=I	>=D	>=R
Менше ніж	<I	<D	<R
Менше ніж або дорівнює	<=I	<=D	<=R

Програмування операцій порівняння в STL

Під час програмування функцій порівняння необхідно керуватися такою схемою операцій:

1) завантаження першого числа;
2) завантаження другого числа;
3) застосування функції порівняння;
4) присвоєння результату.

Під час виконання першої операції число завантажується в акумулятор 1. Під час виконання другої операції вміст акумулятора 1 переміщається в акумулятор 2, а число із другої адреси завантажується в акумулятор 1. Тепер операція порівняння виконується між вмістом двох акумуляторів.

Функція порівняння повертає двійковий результат типу BOOL, що може бути призначений двійковій адресі або може бути використаний у якій-небудь іншій двійковій перевірці. Функції порівняння не змінюють вмісту акумуляторів. Вони завжди виконуються поза всяким зв'язком з якими-небудь умовами.

Приклад:

```
L     #Actval     //Завантаження першої змінної
L     #Calibra    // Завантаження другої змінної
>=R               //Перевірка умови «більше або дорівнює»
S     #Recali     //Установити змінну Recali в «1».
```

Функції порівняння повертають двійковий результат логічної операції RLO і, отже, вони можуть бути використані в сполученні з іншими двійковими функціями. Функція порівняння на початку логічної операції завжди є первинним опитуванням (first check). Значення RLO, що повертається функцією порівняння, може бути безпосередньо використане в логічних операціях.

Приклад:

```
L     MW 120
L     512
>     I
A     Input1
=     Output1.
```

У прикладі вихід *Output1* установлюється, якщо виконується умова порівняння, а вхід *Input1* при цьому має стан «1».

Функція порівняння усередині логічної операції повинна бути укладена в дужки, тому що ця функція починає первинне опитування.

Приклад:

```
O     Input2
O     (
L     MW 12
L     200
<=    I
)
O     Input3
=     Output2.
```

У цьому прикладі вихід *Output2* установлюється, якщо вхід *Input2* або вхід *Input3* має стан «1», або якщо виконується умова порівняння.

Програмування операцій порівняння в LAD і FBD

На рисунку 5.21 як приклад показаний блоковий елемент операції «дорівнює» між числами INT.

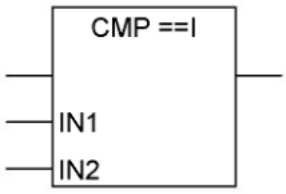


Рисунок 5.21 – Позначення блокового елемента порівняння «дорівнює»

Блоковий елемент функції порівняння має два входи – IN1 і IN2, а також немарковані вхід і вихід. Немарковані вхід і вихід служать для приєднання інших (двійкових) програмних елементів.

Заголовок CMP у блоковому елементі ідентифікує операцію порівняння (compare) і тип виконуваного порівняння.

У ланцюзі LAD компаратор (блок порівняння) можна поставити замість контакту. Порівнювані значення подаються на входи IN1 і IN2, результат порівняння виводиться у вигляді булевої змінної на виході. Успішне порівняння еквівалентно замкнутому контакту (струм протікає через компаратор). Якщо порівняння не успішно, то контакт розімкнутий. Вихід компаратора завжди повинний бути підключений.

У прикладі на рисунку 5.22 меркер M 99.0 скидається, якщо значення в слові MW 92 дорівняє 120, інакше він залишається без змін.

Контакти можна приєднати до й після функції порівняння. Самі блокові елементи порівняння можуть бути з'єднані послідовно або паралельно.

У випадку послідовного з'єднання функцій порівняння оба порівняння повинні бути успішно виконані, щоб у ланцюзі протікав струм.

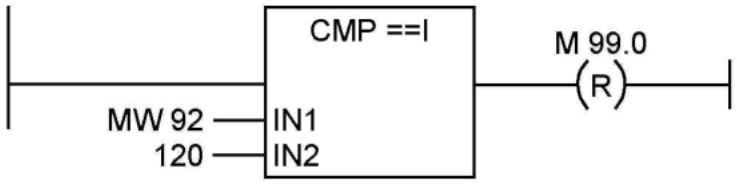


Рисунок 5.22 – Приклад застосування функції порівняння чисел INT

У паралельній схемі струм буде протікати при виконанні будь-якої умови.

5.6 Програмування арифметичних і математичних функцій

Програмування арифметичних функцій

Арифметичні функції забезпечують виконання базових арифметичних операцій із двома числовими значеннями, одне з яких перебуває в акумуляторі 1, а друге – в акумуляторі 2. Результат арифметичної операції записується в акумулятор 1. Біти стану CC0, CC1, OV і OS забезпечують інформацію, яка стосується виконання обчислень і результату.

Огляд доступних користувачеві арифметичних функцій наведений у таблиці 5.5.

Таблиця 5.5 – Арифметичні функції

Назва функції	Тип даних		
	INT	DINT	REAL
Додавання (Addition)	+I	+D	+R
Вирахування (Substraction)	-I	-D	-R
Множення (Multiplication)	*I	*D	*R
Ділення (Division with quotient as result)	/I	/D	/R
Залишок від ділення (Division with remainder as result)	-	MOD	-

Арифметичні функції програмуються за схемою:

1) завантаження першого числа;
2) завантаження другого числа;
3) виконання арифметичної функції;
4) запис результату.

При виконанні першої операції число записується в акумулятор 1. При завантаженні другого числа спочатку вміст акумулятора 1 переміщається в акумулятор 2, а потім друге число завантажується в акумулятор 1. Після цього виконується арифметична операція, у якій беруть участь два акумулятори. Результат операції зберігається в акумуляторі 1.

Арифметичні функції виконуються поза всяким зв'язком з якими-небудь умовами.

Приклад програмування арифметичної операції:

```
L     MW12      // Завантаження першого числа
L     250       // Завантаження другого числа
/I              // Ділення другого числа на перше (ціле число)
T     MW120     // Запис результату.
```

Інструкція ділення /I інтерпретує вміст молодших слів акумуляторів 1 і 2 як числа цілого типу (INT). Інструкція виконує ділення числа, що перебуває в акумуляторі 2 (ділене), на число, що перебуває в акумуляторі 1 (дільник), і зберігає два результати ділення – частку (молодше слово) і залишок (старше слово). Обидва значення мають тип INT.

Частка є цілим результатом операції розподілу. Вона дорівнює нулю у двох випадках:

1) ділене дорівнює нулю;
2) ділене менше, ніж дільник.

Частка від розподілу негативна, якщо дільник менше нуля.

Після виконання інструкції біти стану CC0 і CC1 показують, яка частка від ділення – негативна, дорівнює нулю або позитивна. Біти стану OV і OS указують на порушення дозволеного діапазону.

У випадку ділення на нуль частка від розподілу й залишок вертаються з нульовими значеннями, а біти стану CC0, CC1, OV і OS установлюються в «1».

Якщо потрібно виконати кілька арифметичних операцій, то їх можна запрограмувати послідовно. У цьому випадку результат виконання першої операції використовується для оброблення в наступної операції. Тимчасове зберігання даних забезпечується акумуляторами.

Приклад: Result1:= Value1+Value 2-Value3

```
L     Var1
L     Var2
+I;                 //Var1 + Var2 = сума
L     Var3
-I                  //Сума – Var3
T     Result1.
```

При виконанні арифметичної операції в CPU із двома акумуляторами перше завантажене число перебуває в акумуляторі 2 і може бути використано без повторного завантаження.

*Приклад: Result2: = Var4 * (Var5)2*

```
L     Var5
L     Var4
*D                  //Var4 * Var5
*D                  //Var4 * Var5 * Var5
T     Result2.
```

Операції декременту та інкременту

Синтаксис інструкцій:

```
DEC    n        //Декремент;
INC    n        //Інкремент.
```

Ці операції варто програмувати за такою схемою:

1) завантаження адреси;
2) вказівка операції (декремент або інкремент) і кроку;
3) передача результату Result.

Операції декременту та інкременту виконуються незалежно від значення RLO.

Приклади операцій:

```
L      IncVar
INC 5
T      IncRes.
```

У прикладі значення змінної IncVar збільшується на 5 і передається в змінну IncRes.

```
L      DecValue
DEC 7
T      DecRes.
```

У прикладі значення змінної DecValue зменшується на 7 і зберігається в змінній DecRes.

Програмування математичних функцій

До математичних функцій належать такі функції:

- синус (SIN), косинус (COS), тангенс (TAN);
- арксинус (ASIN), арккосинус (ACOS), арктангенс (ATAN);
- зведення у квадрат (SQR), добування квадратного кореня (SQRT);
- експонента (EXP), логарифм (LN).

Всі математичні функції обробляють числа у форматі REAL. Залежно від результату ці функції встановлюють біти стану CC0, CC1, OV і OS.

Як вхідне значення математичні функції використовують число, що перебуває в акумуляторі 1. Це число обробляється відповідно до інструкції функції й знову зберігається в акумуляторі 1.

Математичні функції змінюють тільки вміст акумулятора 1, вміст всіх інших акумуляторів залишається незмінним. Математичні функції завжди виконуються поза всяким зв'язком з якими-небудь умовами.

Приклади математичних функцій:

```
L      MD 10      //Значення кута в подвійному слові
SIN
T      MD 14      //Запис синуса кута в подвійному слові
…      …
L      #Exponent
EXP
T      #Result.
```

Математичні функції виконуються відповідно до правил оброблення чисел типу REAL, навіть коли застосовується абсолютна адресація й тип вхідного числа не описаний.

Особливості програмування арифметичних і математичних функцій в LAD і FBD

Арифметичні функції представляються у програмах блоковими елементами. Приклад арифметичного блокового елемента для функції підсумовування даних типу INT наведений на рисунку 5.23.

Блоковий елемент арифметичної функції має, крім входу дозволу EN (enable input) і виходу дозволу ENO (enable output), два входи – IN1 і IN2, а також вихід OUT. Заголовок у блоковому елементі ідентифікує арифметичну дію, що виконується.

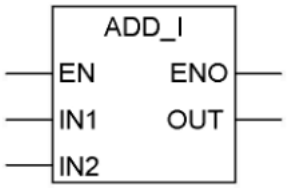


Рисунок 5.23 – Позначення блокового елемента підсумовування даних типу INT

У таблиці 5.6 наведені позначення арифметичних функцій у відповідності із типом даних.

Таблиця 5.6 – Позначення арифметичних функцій

Арифметична функція	Відповідність типу даних		
	INT	DINT	REAL
Додавання	ADD_I	ADD_DI	ADD_R
Вирахування	SUB_I	SUB_DI	SUB_R
Множення	MUL_I	MUL_DI	MUL_R
Ділення (результат – частка)	DIV_I	DIV_DI	DIV_R
Ділення із остачею	-	MOD_DI	-

Значення, що обчислюються, подаються на входи IN1 і IN2, результат обчислення виводиться на вихід OUT. Використовувані змінні повинні бути того самого типу даних, що входи елемента.

Арифметична функція виконується, якщо на вході дозволу присутній сигнал «1». Якщо під час обчислення виникає помилка, то вихід дозволу встановлюється в «0. Якщо виконання функції не дозволене (EN = 0), то обчислення не виконується. Якщо головне реле керування (MCR) активовано, то вихід OUT установлюється в нуль.

Під час виконання арифметичної функції можуть виникнути такі помилки:

- переповнення в обчисленнях з типами INT і DINT;
- зникнення значущих розрядів і переповнення в обчисленнях з типом REAL;
- неприпустиме число REAL в обчисленнях з типом REAL.

На рисунку 5.24 значення слова у пам'яті меркерів MW 100 ділиться на 250, цілочисельний результат зберігається в меркері MW 102.

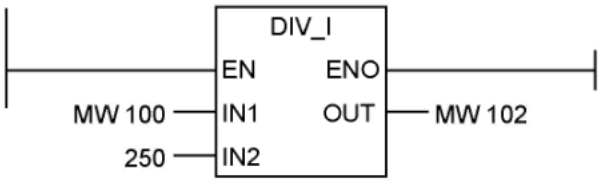


Рисунок 5.24 – Приклад застосування функції поділу чисел INT

Арифметичний блоковий елемент може бути поміщений у будь-якому місці ланцюга. Пряме з'єднання з лівою (живильною) напрямною дозволяє підключити арифметичні блокові елементи паралельно. При цьому блокові елементи самого верхнього ланцюга обробляються зліва направо, потім зліва направо обробляються елементи другого ланцюга й так далі. Для завершення ланцюга потрібно встановити котушку, якій можна призначити, наприклад, біт тимчасових локальних даних.

Якщо вихід ENO попереднього блокового елемента з'єднаний із входом EN наступного, то останній спрацьовує тільки за тією умовою, що попередній елемент був оброблений без помилок.

Математичні функції

У LAD і FBD представлені такі математичні функції:

- синус, косинус, тангенс;
- арксинус, арккосинус, арктангенс;
- зведення у квадрат, добування квадратного кореня;
- експонентна функція за основою *e*, натуральний логарифм.

Всі математичні функції працюють із числами типу даних REAL.

Блоковий елемент математичної функції має вхід IN і вихід OUT, а також вхід дозволу EN і вихід дозволу ENO. Заголовок у блоковому елементі ідентифікує виконувану математичну функцію.

Приклад позначення блочного елементу математичної функції *синус* наведений на рисунку 5.25.

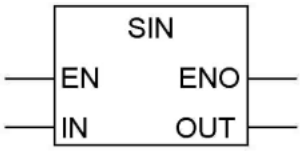


Рисунок 5.25 – Позначення блокового елемента «синус»

Вхідне значення подається на вхід IN, а результат математичної функції виводиться на вихід OUT. Вхід і вихід належать до типу даних REAL. Операнди, які адресуються з використанням абсолютних адрес, повинні мати розмір подвійного слова.

Математична функція виконується, якщо на вході дозволу (EN) присутній сигнал «1». Якщо під час обчислення виникне помилка, то вихід дозволу ENO установлюється в «0». Якщо виконання функції не дозволене (EN = 0), то обчислення не виконується й ENO також дорівнює «0».

Якщо головне реле керування (MCR) активно, то під час виконання математичної функції (EN = 1) вихід OUT установлюється в нуль. При цьому MCR не впливає на ENO.

У математичній функції можуть виникнути такі помилки:

- вихід за межі діапазону й переповнення;
- неприпустиме число типу REAL як вхідне значення.

Приклад обчислення функції *синус* показано на рисунку 5.26. Значення подвійного слова пам'яті (меркерів) MD 110 містить величину в радіанах. Від цього значення береться синус і зберігається в подвійному слові пам'яті MD 104.

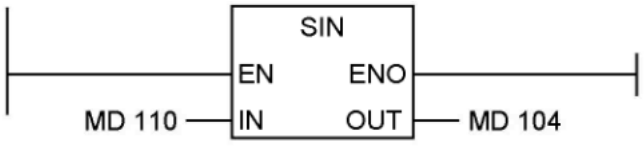


Рисунок 5.26 – Приклад обчислення функції синус

5.7 Застосування функцій перетворення типів даних

Функції перетворення конвертують тип даних, які перебувають в акумуляторі 1. Схема функцій перетворення представлена на рисунку 5.27.

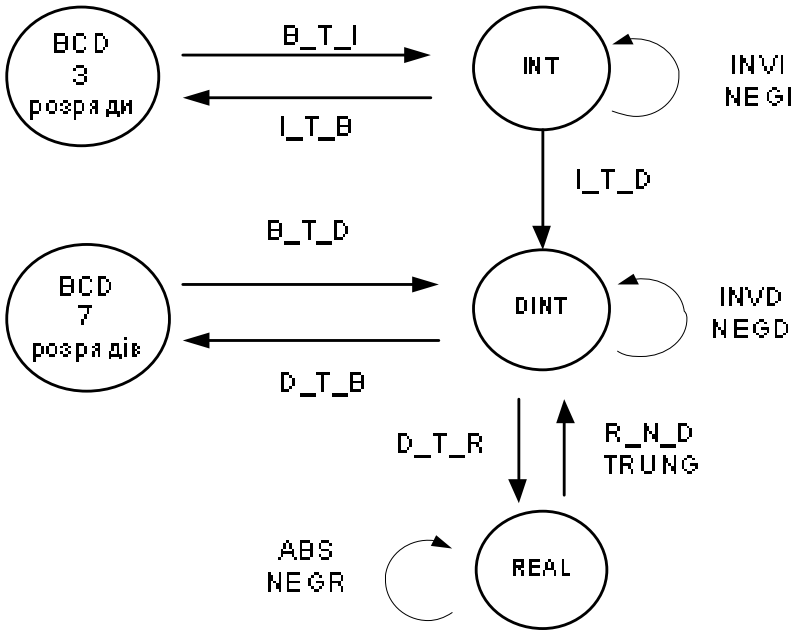


Рисунок 5.27 – Схема використання функцій перетворення

Функції перетворення впливають тільки на дані, що перебувають в акумуляторі 1. При цьому окремі функції перетворення впливають тільки на вміст молодшого слова в акумуляторі 1 (біти з 0 по 15), інші функції впливають на вміст акумулятора в цілому. Функції перетворення завжди виконуються поза всяким зв'язком з якими-небудь умовами.

Програмування функцій перетворення в STL

Функції перетворення програмуються за такою схемою:

1) завантаження операнда в акумулятор;
2) запис функції перетворення;
3) передача результату.

Приклад програмування функцій перетворення:

```
L     MW 120
ITB                  //Перетворення INT в BCD
T     MW 122.
```

Вміст акумулятора 1 можна використовувати в декількох послідовно виконуваних функціях перетворення.

Приклад:

```
L     BCD_Number
BTI             //Перетворення BCD в INT
ITD             // Перетворення INT в DINT
DTR             // Перетворення DINT в REAL
T     REAL_Number.
```

У цьому прикладі BCD-число перетвориться в число формату REAL.

Існує кілька спеціальних функцій перетворення:

- INVI знаходження зворотного коду двійкового числа типу INT;
- INVD знаходження зворотного коду числа типу DINT;
- NEGI інвертування числа типу INT;
- NEGD інвертування числа типу DINT;
- NEGD інвертування числа типу REAL;
- ABS знаходження абсолютного значення числа типу REAL.

Всі функції перетворення не встановлюють бітів стану.

Програмування функцій перетворення в LAD і FBD

На рисунку 5.28 показаний приклад блокового представлення функції перетворення INT в BCD (I_BCD).

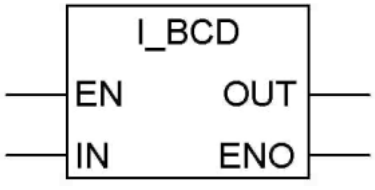


Рисунок 5.28 – Позначення функції перетворення I_BCD

Значення, що підлягає конвертуванню, подається на вхід IN, результат конвертування формується на виході OUT. Тип даних входу й виходу залежить від функції перетворення. У функції перетворення DI_R (DINT в REAL), наприклад, вхід має тип DINT, а тип виходу – REAL.

Функція перетворення виконується, якщо на вході дозволу присутній сигнал «1». Якщо під час перетворення виникне помилка, то вихід ENO установлюється в «0», в іншому випадку він установлюється в «1». Якщо виконання функції не дозволене (EN = 0), перетворення не відбувається й на виході ENO також установлюється нуль.

У таблиці 5.7 наведені функції перетворення для чисел типів INT і DINT. Змінні на входах і виходах повинні мати певні типи даних, а операнди, які адресуються абсолютно, відповідати їх розміру.

Таблиця 5.7 – Функції перетворення чисел типів INT і DINT

Перетворення типу даних	Блоковий елемент	Тип даних параметра	
		IN	**OUT**
INT в DINT	I_DI	INT	DINT
INT в BCD	I_BCD	INT	WORD
DINT в BCD	DI_BCD	DINT	DWORD
DINT в REAL	DI_R	DINT	REAL

Не всі функції перетворення повідомляють про помилку. Помилка виникає тільки тоді, коли перевищується припустимий діапазон чисел (I_BCD, DI_BCD) або визначене недійсне число REAL.

Якщо вхідне значення для перетворення BCD_I або BCD_DI містить псевдотетраду, то виконання програми переривається, і викликається організаційний блок оброблення помилок OB 121 (синхронні помилки роботи програми).

На рисунку 5.29 показаний приклад перетворення числа INT. Значення слова MW 120 інтерпретується як число типу INT і зберігається як BCD-число у пам'яті меркерів MW 122.

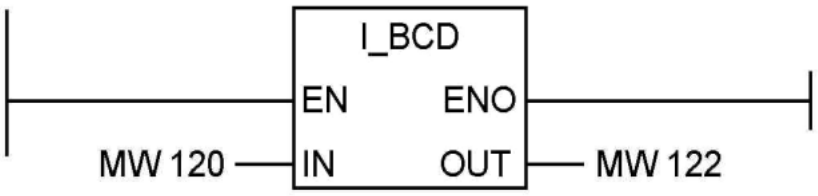


Рисунок 5.29 – Приклад перетворення числа INT

Є кілька функцій для перетворення числа у форматі REAL у число формату DINT (перетворення дробового значення в ціле число). Ці функції наведені в таблиці 5.8. Вони відрізняються за способом округлення.

Таблиця 5.8 – Перетворення чисел типу REAL у числа типу DINT

Перетворення типу даних з округленням	Блоковий елемент	Тип даних параметра	
		IN	OUT
До наступного більшого цілого числа	CEIL	REAL	DINT
До наступного меншого цілого числа	FLOOR	REAL	DINT
До наступного цілого числа	ROUND	REAL	DINT
Без округлення	TRUNC	REAL	DINT

До входів і виходів повинні бути застосовані змінні певного типу даних розміром у подвійне слово.

Функції CEIL, FLOOR, ROUND і TRUNC інтерпретують значення на вході IN як число у форматі REAL і перетворюють його в число формату DINT на виході OUT.

CEIL повертає ціле, що більше або дорівнює конвертованому числу. Якщо значення на вході IN перебуває поза діапазоном, припустимим для чисел формату DINT, або воно не відповідає числу у форматі REAL, то CEIL установлює біти стану OV і OS. Перетворення при цьому не виконується.

Функція FLOOR повертає ціле, що менше або дорівнює конвертованому числу. Якщо значення на вході IN перебуває поза діапазоном, припустимого для чисел у форматі DINT, або воно не відповідає числу у форматі REAL, то FLOOR установлює біти стану OV і OS. Перетворення при цьому не виконується.

Функція ROUND повертає наступне ціле число. Якщо результат перебуває точно між непарним і парним числом, перевага надається парному числу. Якщо значення на вході IN перебуває поза діапазоном, припустимим для чисел у форматі DINT, або воно не відповідає числу у форматі REAL, то ROUND установлює біти стану OV і OS. Перетворення при цьому не виконується.

Функція TRUNC повертає цілу складову перетвореного числа, а дробова складова відтинається.

5.8 Програмування функцій зсуву

Функції зсуву дозволяють побітно зсувати вліво або вправо дані, які перебувають в акумуляторі 1 (слово або подвійне слово). Біти, що зсуваються за межі слова (подвійного слова), або губляться при операціях простого зсуву, або переносяться на іншу сторону. Функції зсуву не впливають на вміст інших акумуляторів і на результат логічної операції (RLO).

Програмування функцій зсуву в STL

Доступні користувачеві функції зсуву наведені у таблиці 5.9.

Таблиця 5.9 – Огляд функцій зсуву

Функція зсуву	W (слово)		DW (подвійне слово)	
	Число поз. як параметр	Число поз. у Accum2	Число позиц. як параметр	Число поз. у Accum2
Зсув вліво	SLWn	SLW	SLDn	SLD
Зсув вправо	SRWn	SRW	SRDn	SRD
Зсув зі знаком	SSIn	SSI	SSDn	SSD
Циклічний зсув вліво			RLDn	RLD
Циклічний зсув вправо			RRDn	RRD
Циклічний зсув вліво через біт CC1			RLDA	
Циклічний зсув вправо через біт CC1			RRDA	

Позначення функцій залежить від способу завдання кількості розрядів, на яку здійснюється зсув. Число позицій для функції зсуву може бути задано двома шляхами – в акумуляторі 2 або у параметрі інструкції.

Так, наприклад, зсув слова даних вліво здійснюється інструкціями:

SLW n // Зсув слова даних вліво на *n* розрядів;

SLW // Зсув слова даних вліво на кількість розрядів, зазначених в акумуляторі 2.

Функція зсуву SLW дозволяє біт за бітом зсувати вліво дані, що перебувають у молодшому слові акумулятора 1 (з 0 по 15). При цьому розряди, що звільняються під час зсуву зазначених бітів заповнюються нулями. Біти, що перебувають у старшому слові акумулятора, залишаються без зміни. Перенесення даних у біт 16 не виконується.

Якщо вміст акумулятора 1 (молодшого слова) інтерпретується як ціле типу INT, то зсув вліво еквівалентний множенню на 2.

Для зсуву подвійного слова даних *вліво* варто застосовувати інструкції:

SLD n //зсув подвійного слова вліво на *n* розрядів;

SLD //зсув подвійного слова даних вліво на кількість
//розрядів, зазначених в акумуляторі 2.

Під час зсуву слова даних *вправо* застосовуються інструкції:

SRW n

SRW.

Якщо вміст акумулятора 1 (молодшого слова) інтерпретується як ціле число формату INT, то зсув вправо еквівалентний поділу на 2.

Функції зсуву встановлюють біт стану CC0 в «0», а біт CC1 у стан, в якому перебував останній переміщений біт.

Біти стану перевіряються за допомогою двійкового опитування або в операціях переходу.

Функція *циклічного* зсуву з бітом стану CC1 зміщає вміст акумулятора 1 на одну розрядну позицію.

Приклади програмування функцій зсуву:

```
L     MW 130
SLW   4          //зсув слова на 4 розряди вліво
T     MW 132
…     …
L     #Actval
SSI   2          //зсув слова даних на 2 розряди вправо зі знаком
T     #Display.
```

Під час зсуву числа INT зі знаком *інструкцією SSI* вивільнювані розряди заповнюються значенням знакового біта 15, тобто значенням «0», якщо число позитивне, й значенням «1», якщо число негативне.

Під час зсуву подвійного слова даних (числа типу DINT) зі знаком застосовується інструкція SSD. Розряди, що звільняються тут при зсуві, заповнюються значенням біта 31, який містить знак числа формату DINT.

Операції циклічного зсуву RLD і RRD дозволяють біт за бітом зсувати вліво (RLD) або вправо (RRD) дані, що перебувають у всіх розрядах акумулятора 1. При цьому знову звільнений при зсуві розряд заповнюється значенням біта, що був «виштовхнутий» з акумулятора останнім.

Циклічний зсув може також виконуватися через біт стану CC1.

Програмування функцій зсуву мовами LAD і FBD

У мовах LAD і FBD операції зсуву представляються блоковими елементами. На рисунку 5.30 у якості приклада показано елемент для зсуву слова вліво, а у таблиці 5.10 – огляд функцій зсуву.

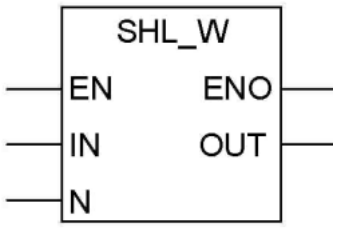


Рисунок 5.30 – Позначення блокового елемента функції зсуву

Таблиця 5.10 – Огляд функцій зсуву

Функція зсуву	Змінна – слово	Змінна – подвійне слово
Зсув вліво	SHL_W	SHL_DW
Зсув вправо	SHR_W	SHR_DW
Зсув зі знаком	SHR_I	SHR_DI
Циклічний зсув вліво Циклічний зсув вправо		ROL_DW ROR_DW

Крім входу дозволу EN і виходу ENO, блоковий елемент функції зсуву має вхід IN, вхід N і вихід OUT. Заголовок у блоковому елементі ідентифікує виконувану функцію зсуву.

Значення, що підлягає зсуву, подається на вхід IN. Кількість розрядів для зсуву задається на вході N. Результат видається на вихід OUT. Типи даних на вході й виході залежать від функції зсуву. Так, наприклад, вхід і вихід для функції зсуву SHR_DW (зсув вправо змінної розміром у подвійне слово) повинні належати до типу DWORD.

Приклад застосування функцій зсуву

Зсув змінних розміром у слово показаний на рисунку 5.31. Значення в слові пам'яті MW 130 зсувається на 4 позиції вліво й зберігається в слові пам'яті MW 132.

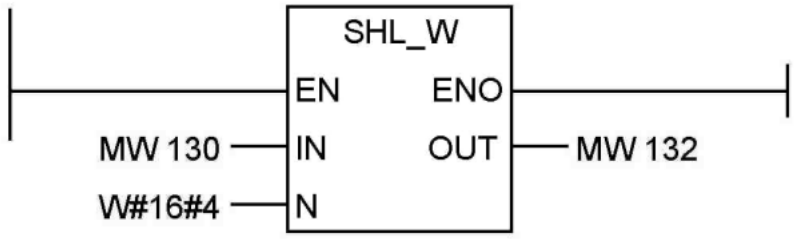


Рисунок 5.31 – Приклад зсуву числа INT

Число зсувів на вході N може бути константою або змінною. Якщо число зсувів дорівнює 0, то функція не виконується, якщо більше 15, то вихідна змінна після виконання функції містить нуль.

Під час зсуву подвійного слова, наприклад функцією SHR_DI, вміст змінної типу DINT на вході IN зсувається побітно вправо на кількість позицій, що обумовлена числом на вході N. Бітові розряди, звільнені при зсуві, заповнюються сигнальним станом біта 31, який є знаком числа DINT. Зсув вправо числа у форматі DINT еквівалентний поділу на 2^n, де показник ступеня n є числом зсувів.

Циклічні зсуви змінної *вліво* здійснюються за допомогою функції ROL_DW. Функція зсуває вліво вміст змінної типу DWORD на вході IN побітно на кількість позицій, обумовлену числом на вході N. Бітові позиції, які вивільняються при зсуві, заповнюються витиснутими бітовими розрядами. Результат утримується в змінній типу DWORD на виході OUT.

Число зсувів на вході N може бути константою або змінною. Якщо число зсувів дорівнює 0, то функція не виконується. Якщо воно дорівнює 32, то вміст вхідної змінної зберігається й установлюються біти стану. Якщо число зсувів дорівнює 33, здійснюється зсув одного розряду. Якщо воно дорівнює 34, зсуваються два розряди й так далі (зсуви виконується за модулем 32).

Циклічний зсув змінної *вправо* здійснюється за допомогою функції ROR_DW.

5.9 Контроль стану операції та програмування переходу у програмі

Біти стану – це двійкові прапори (індикаторні біти), які втримуються в слові стану (status word) і використовуються для керування операціями. У таблиці 5.11 представлені біти стану, доступні при програмуванні.

Таблиця 5.11 – Формат слова стану (status word)

Біт	Двійкові прапорці (binary flags)	
0	/FC	Первинне опитування (first check)
1	RLO	Результат логічної операції
2	STA	Стан (status)
3	OR	Біт стану OR (OR status bit)
8	BR	Двійковий результат
	Числові прапорці (digital flags)	
4	OS	Для збереження інформації про переповнення
5	OV	Переповнення (Overflow)
6	CC0	Умовний код
7	CC1	Умовний код

Перша колонка показує номер біта в слові стану. Біти з 0 по 3 і 8 – це «двійкові прапори». Вони застосовуються для керування двійковими функціями. Біти з 4 по 7 – це «числові прапори». Вони використовуються, насамперед, для індикації результатів арифметичних і математичних функцій.

Біт 0 – первинне опитування FC (first check)

Біт стану /FC керує двійковими логічними операціями усередині арифметичного логічного пристрою керування. Двійковий логічний крок завжди починається із двійкової інструкції перевірки (первинного опитування) при /FC = «0». Первинне опитування встановлює /FC у стан «1». Двійковий логічний крок закінчується або присвоєнням двійкового значення, або умовним переходом, або зміною блоку. При цьому біт стану /FC скидається. Наступна двійкова перевірка (опитування) починається з нової двійкової логічної функції.

Біт 1 – результат логічної операції (RLO)

Біт стану RLO є проміжним буфером у двійкових логічних операціях. При первинному опитуванні процесор передає результат опитування в RLO, комбінує цей результат зі збереженим в RLO значенням і потім зберігає нове значення в RLO. Значення біта RLO можна встановлювати, скидати, інвертувати або зберігати в біті двійкового результату BR (біт 8).

Біт 2 – стан STA (status)

Значення біта STA відповідає стану сигналу зазначеного двійкового розряду (двійкової адреси) або стану «умовного біта», що перевіряється при аналізі двійкової логічної операції. Біт стану STA не впливає на оброблення STL-операторів. Його можна використовувати для трасування двійкових логічних послідовностей або для налагодження програми.

Біт 3 – стан OR (OR status bit)

Біт OR status зберігає результат виконаної двійкової логічної операції AND (I) і показує наступній операції OR (АБО), що результат уже зафіксований.

Біти переповнення 4 і 5 – OV (Overflow) і OS (Overflow stored)

Біт стану OV відображує факт перевищення діапазону припустимих числових значень (переповнення) або факт використання некоректних дійсних чисел (REAL). На стан біта OV впливають арифметичні й математичні функції, функції порівняння дійсних чисел типу REAL, а також окремі функції перетворення. Перевірити стан біта OV можна за допомогою операцій перевірки або оператором переходу JO.

Біт стану OS дублює й зберігає встановлений стан біта OV. При цьому, якщо біт стану OV надалі може бути скинутий під час виконання відповідної операції, то біт OS збереже інформацію про факт перевищення.

Біти станів CC0, CC1 (6 і 7)

Біти станів CC0, CC1 – це умовні біти, які відображають результати виконання функцій порівняння, арифметичних і математичних функцій, логічних операцій для слів даних, а також станів бітів в операціях зсуву.

Біт 8 – BR (двійковий результат) допомагає реалізувати механізм EN/ENO для викликів блоків у сполученні із графічними мовами.

Всі біти станів, тобто слово стану STW можна завантажити в акумулятор 1 інструкцією:

```
L     STW.
```

Програмування переходів у STL-програмі

За допомогою функцій переходу (jump functions) можна перервати лінійне виконання програми й продовжити її виконання в іншій точці блоку. Таке розгалуження програми може бути організовано або з перевіркою виконання певної умови, або без перевірки яких-небудь умов. Відповідно, у програмі повинен використовуватися або умовний, або безумовний перехід.

У STL використовуються такі функції переходів:

JU	*мітка*	безумовний перехід;
JC	*мітка*	перехід, якщо RLO = «1»;
JCN	*мітка*	перехід, якщо RLO = «0»;
JCB	*мітка*	перехід, якщо RLO = «1», і збереження RLO;
JNB	*мітка*	перехід, якщо RLO = «0», і збереження RLO;
JBI	*мітка*	перехід, якщо BR = «1»;
JNBI	*мітка*	перехід, якщо BR = «0»;
JZ	*мітка*	перехід, якщо результат дорівнює «0»;
JN	*мітка*	перехід, якщо результат не дорівнює «0»;
JP	*мітка*	перехід, якщо результат більше «0»;
JPZ	*мітка*	перехід, якщо результат більше або дорівнює «0»;
JM	*мітка*	перехід, якщо результат менше «0»;
JMZ	*мітка*	перехід, якщо результат менше або дорівнює «0»;
JUO	*мітка*	перехід, якщо результат невірний;
JO	*мітка*	перехід при переповненні (перевірка біта OV);
JOS	*мітка*	перехід при переповненні (перевірка біта OS);
LOOP	*мітка*	перехід циклічний.

Інструкція для кожної функції переходу містить оператор переходу й мітку переходу. Оператор переходу визначає умову, яка перевіряється, а мітка показує, у якій точці програми варто продовжувати оброблення програми у випадку, коли умова для переходу виконується.

Мітка переходу може містити до 4 символів алфавіту й числового ряду, а також символ підкреслення. Мітка переходу не може починатися із цифри. Після мітки переходу ставиться двокрапка. Мітка переходу вказує на вираз, який повинно оброблятися після виконання операції переходу.

Приклад програмування з розгалуженням:

```
L     C 1          //Завантажити значення лічильника
L     50           //Завантажити умовне значення
>I                 //Програмування умови
JC    M1           //Умовний перехід
…     …      //Продовження програми при невиконанні умови
JU    M2           //Безумовний перехід
M1: …        …//Мітка продовження програми при виконанні умови
M2: …        …//Мітка основної програми.
```

У наведеному прикладі умовою переходу є позитивний результат операції порівняння RLO. При RLO = «1» виконується перехід до мітки продовження програми M1. При негативному результаті операції порівняння (RLO = «0») перехід до M1 не виконується.

Перехід може виконуватися в будь-якому напрямку програми, однак тільки усередині блоку. Мітка переходу повинна мати унікальний ідентифікатор.

У процесі програмування мовою STL ідентифікатори міток зберігаються у відповідній частині блоку на носіях даних програматора PG. За цієї причини зміни в програмі блоку, зроблені в CPU в інтерактивному режимі, повинні бути також виконані й у програматорі PG.

Особливості керування переходами у мовах LAD і FBD

Для читання бітів стану в LAD передбачені як нормально розімкнуті контакти, так і нормально замкнуті. Ці контакти перебувають у розділі Status bits браузера програмних елементів вікна редагування. Для читання бітів стану в FBD передбачені відповідні блокові елементи.

У мовах програмування LAD і FBD низка блокових елементів має вхід дозволу EN і вихід дозволу ENO. Якщо на вході EN присутній стан «1», то функція блокового елемента обробляється. Якщо блоковий елемент оброблений коректно, вихід має сигнальний стан «1». Якщо під час обробки блокового елемента виникає помилка, наприклад, переповнення при виконанні арифметичної функції, то ENO установлюється в «0». Якщо EN має сигнальний стан «0», то ENO буде також установлений в «0».

Ці характеристики EN і ENO можуть бути використані для з'єднання в ланцюг декількох блокових елементів з підключенням виходу дозволу до входу дозволу наступного елемента. Приклад такого з'єднання елементів в FBD наведений на рисунку 5.32.

Якщо, наприклад, вхід Input0 має стан «0», весь ланцюг буде виключеним. Керувати виходом ENO можна також за допомогою бінарного результату BR.

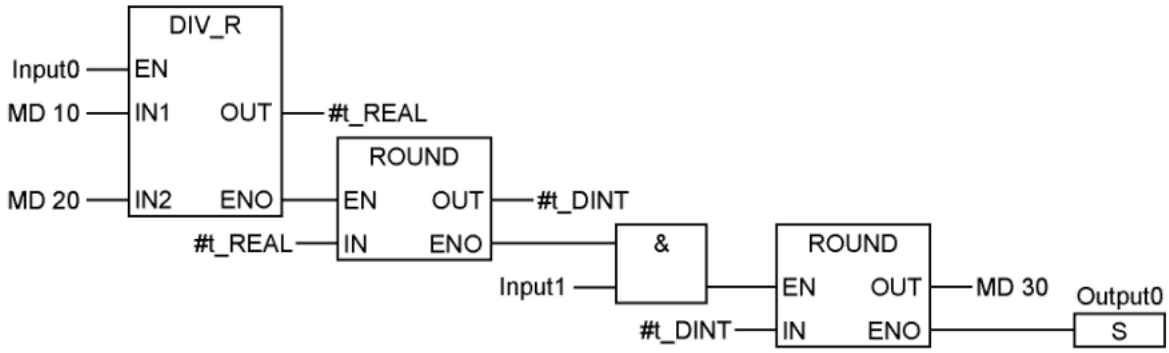


Рисунок 5.32 – Приклад послідовного з'єднання EN і ENO

Функція переходу складається з операції переходу у формі котушки (у LAD) або блокового елемента (у FBD), а також мітки переходу, що позначає місце в програмі для продовження оброблення. Мітка переходу вказується над операцією переходу (рис. 5.33).

Рисунок 5.33 – Позначення функцій переходу (LAD)

Мітка переходу може містити в собі до 4 символів, які можуть бути буквами, цифрами й знаками підкреслення. Починається вона з букви. Мітка переходу *в блоковому елементі* позначає ланцюг, що буде оброблятися після закінчення операції переходу. Блоковий елемент із міткою переходу повинен розташовуватися на початку ланцюга. Вибрати цей елемент можна в каталозі програмних елементів у розділі LABEL (*Мітка*).

Переходи можна виконувати як уперед (у напрямку обробки програми), так і назад. Перехід може здійснюватися тільки в межах блоку, тобто місце призначення переходу повинне бути в тім же блоці, що й функція переходу. Якщо використовується головне реле керування MCR, мітка переходу повинна перебувати в тій же MCR-зоні або в MCR-області, що й функція переходу.

Місце призначення переходу повинне бути позначене однозначно. Програмний редактор зберігає імена міток переходів у розділах відповідних блоків.

Абсолютним переходом, що завжди виконується, є функція переходу JMP. У програмі LAD котушка JMP повинна бути з'єднана з лівою живильною шиною (рис. 5.34).

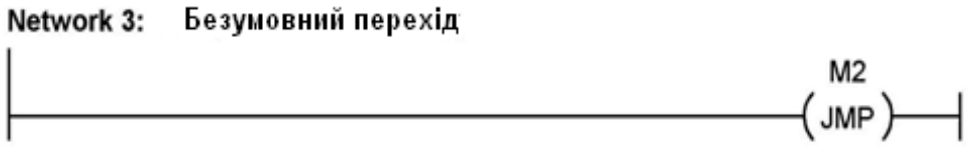


Рисунок 5.34 – Правило застосування безумовного переходу в LAD

Якщо котушка не має прямого з'єднання з лівою шиною, то виконується умовний перехід.

Якщо RLO = 1 (у випадку виконання попередньої логічної операції), то CPU переривається лінійний потік програми й продовжує оброблення в ланцюзі, позначеному міткою переходу. Якщо попередня логічна операція не виконана, то CPU продовжує виконувати програму в наступному ланцюзі.

Умовний перехід при RLO = 0 виконується функцією JMPN, чия котушка не з'єднана прямо з лівою живильною шиною. Якщо RLO = 0 у випадку незадоволення попередньої логічної операції, CPU переривається лінійний потік програми й продовжує оброблення в тому ланцюзі, який позначений міткою переходу.

5.10 Застосування інструкцій для виклику і завершення блоків

Інструкції для виклику й завершення оброблення блоків належать до функцій обробки кодових блоків.

Кодові блоки викликаються оператором CALL. Під час виклику функціонального блока відкривається екземплярний блок даних, за допомогою якого можна передавати дані у викликуваний блок або одержувати дані від викликуваного блока. Якщо кодові блоки не мають параметрів, то їх можна викликати за допомогою операторів UC або CC. Оброблення блока припиняється за допомогою оператора кінця блока. Оператор BEC завершує оброблення програми в блоці залежно від стану RLO, а оператори BEU і BE закінчують блок незалежно від умов.

Функції для кодових блоків наведені в таблиці 5.12.

Програма блоку обробляється, поки не зустрінеться оператор закінчення блока. По закінченні викликаного блоку CPU вертається до виконання програми в блоці, що зробив виклик. Виконання цієї програми триває з оператора, який слідує за оператором виклику блока. Якщо виконання програми організаційного блока завершується, CPU передає керування операційній системі.

Інформація про те, що потрібно CPU для повернення у визивний блок, зберігається в B-стеці блоків. При кожному виклику блока у B-стеці генерується новий елемент, який містить адресу повернення, вміст регістра даних і адресу локальних даних блока, який зробив виклик.

В інструкції виклику може бути список параметрів блока. Під час введення вихідного тексту програми список параметрів блока знаходиться між круглими дужками. Параметри блока, наведені у списку, повинні бути розділені комами. Під час виклику функціонального блока немає необхідності ініціалізувати всі параметри викликуваного блока.

Таблиця 5.12 – Функції для кодових блоків

Виклик функціонального блока		
Із блоком даних і параметрами	Як локальний екземпляр з параметрами	Виклик безумовний і за умовою
CALL FB1, DB1 (In1: =Num1; In2: =Num2; In3: =Num3);	CALL name (In1: =Num1; In2: =Num2; In3: =Num3);	UC FB 5 CC FB 5
Виклик функції		
Зі значенням функції та з параметрами блока	Без значення функції, з параметрами блока	Виклик безумовний і за умовою
CALL FB1 (In1: =Num1; In2: =Num2; Ret_Val: = Num3);	CALL FB1 (In1: =Num1; In2: =Num2; Out: = Num3);	UC FB 5 CC FB 5
Оператори завершення блока		
Умовне завершення блока	Безумовне завершення блока	Кінець блока
BEC	BEU	BE

Функції FC викликаються шляхом завдання ідентифікатора функції абсолютним або символьним способом після оператора CALL. Під час виклику функції потрібно ініціалізувати всі параметри. Викликувані функції з функціональним значенням мають таку ж саму форму, як і функції без функціонального значення. Єдиний вихідний параметр, що відповідає функціональному значенню, має ім'я RET_VAL.

Системні функціональні блоки можна викликати в такий же спосіб, як і блоки, створені користувачем. Системні блоки доступні тільки в операційній системі CPU. Під час виклику системних блоків під час програмування в автономному (offline) режимі, редактору потрібен опис інтерфейсу виклику для того, щоб ініціалізувати параметри.

Опис інтерфейсу розташований у стандартній бібліотеці *Standard library* у системних функціональних блоках *System Function Blocks*. Звідси редактор копіює опис інтерфейсу в папку (розділ) автономного режиму «Blocks», коли викликається системний блок. Після цього скопійований опис інтерфейсу виклику з'являється як «нормальний» об'єкт блока.

5.11 Методика створення програми мовами LAD, STL, FBD у редакторі STEP 7

Для складання програми на мові STL треба спочатку визначитися – де буде розміщена ця програма. Якщо це буде головна програма з циклічним виконанням, то її слід записати в організаційному блоці OB1. Якщо програма розробляється для локальної задачі логічного керування, то для її розміщення можливо потрібен функціональний блок.

У будь-якому випадку спочатку треба створити проект.

Створення проекту відбувається у середовищі Simatic Manager. Після завдання ім‘я проекту, вибору типу станції та типу процесорного модуля з‘являється можливість розміщення програмних блоків.

Для цього у дереві проекту знайдемо та відкриємо папку Blocks. На правому полі вікна проекту в Simatic Manager знаходиться перший блок – блок організації циклічного виконання програми OB1.

Нехай, наприклад, треба створити програму для керування пішохідним переходом і розмістити її в окремому функціональному блоці.

Клацнемо на полі правою кнопкою миші та виберемо у контекстному меню команди «Insert New Object» → «Function Block», як показано на рисунку 5.35.

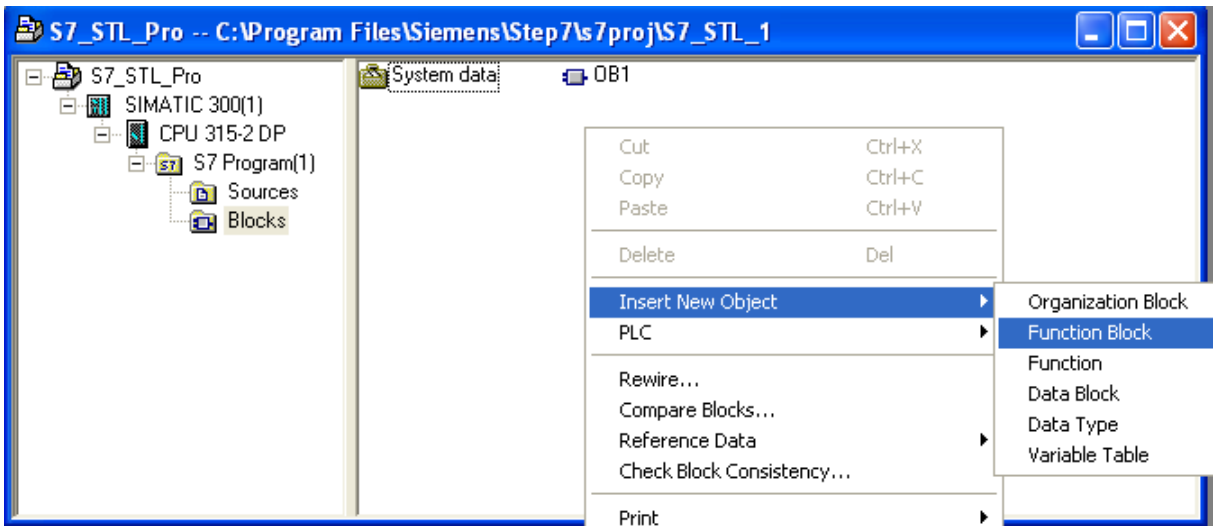


Рисунок 5.35 – Вибір команд у вікні Simatic Manager для створення функціонального блока

При цьому відкривається вікно для завдання параметрів блока. На вкладці General – Part 1 у відповідних полях вносимо: номер блока (нумерація блоків виконується автоматично, але її можна змінити), його символічне ім‘я (Cross) і мову програмування (STL). Вікно Properties після виконання цих операцій показано на рисунку 5.36.

Після створення функціонального блока треба визначити склад змінних, які будуть використовуватися у програмі. Перш за все, необхідно визначити глобальні змінні, які потрібно внести до таблиці Symbol Table.

Щоб відкрити цю таблицю, треба у вікні редактора вибрати меню Options → Symbol Table. У таблицю символів можна включати вхідні та вихідні дані, меркери, периферійні дані, таймери, лічильники, функціональні блоки та функції, організаційні блоки, блоки даних і типи даних UDT. Редагувати Symbol Table можна у будь-який момент.

Далі визначаються локальні змінні блока – його інтерфейс із програмою. До інтерфейсу належать вхідні дані (IN), вихідні дані (OUT), вхідні та вихідні параметри (IN_OUT), а також статичні змінні (STAT). Крім цього, блок може мати тимчасові дані (TEMP), які не належать до інтерфейсу.

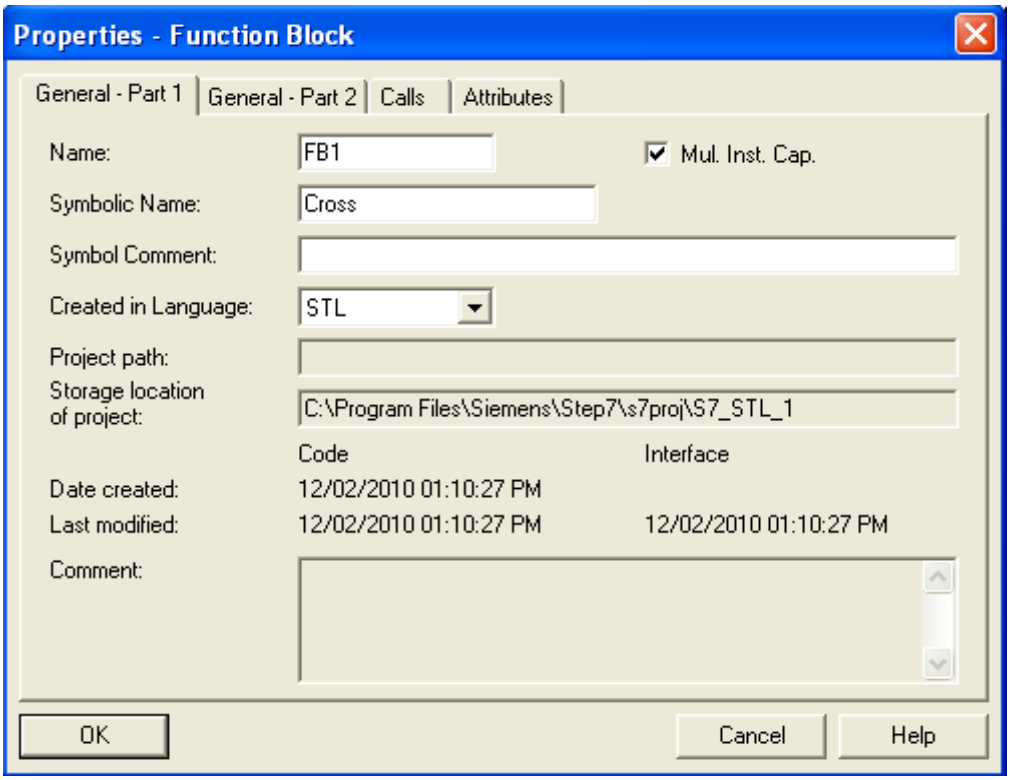


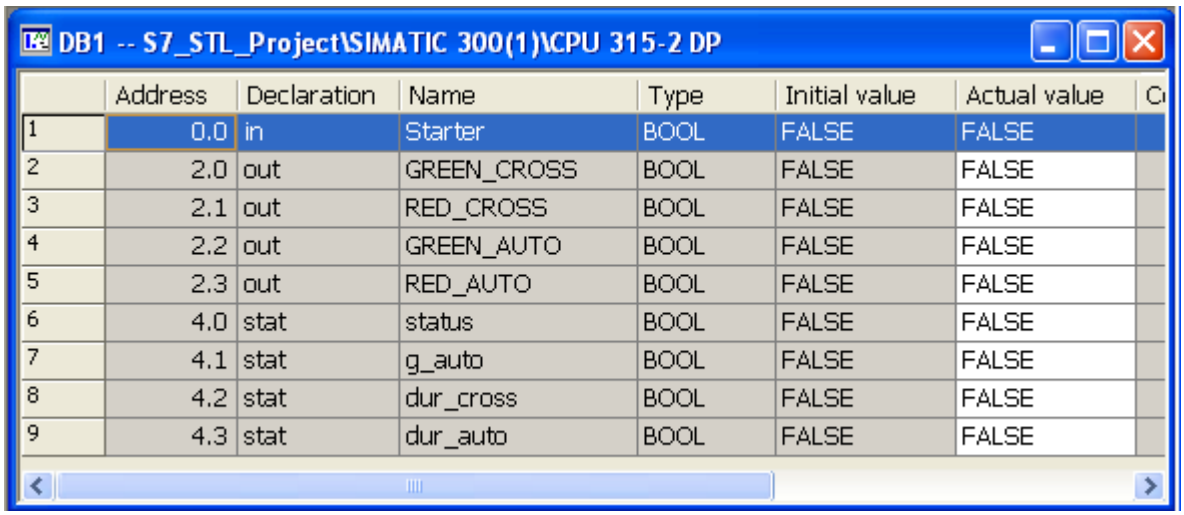
Рисунок 5.36 – Вигляд вікна Properties після завдання параметрів блока

Оголошення символічних імен і типів всіх необхідних змінних здійснюється у розділі оголошень вікна редактора STL-програми, яке можна відкрити подвійним клацанням лівої кнопки на піктограмі створеного функціонального блока у вікні Simatic Manager.

Після оголошення змінних у функціональному блоці треба створити блок даних DB. Слід взяти до уваги, що у випадку внесення змін у розділ оголошень функціонального блока після створення блока даних, інтерфейс блока даних уже не буде відповідати інтерфейсу функціонального блоку, тому блок даних повинен бути створений заново.

Для створення блока даних потрібно у вікні проекту Simatic Manager розкрити папку Blocks, клацнути правою кнопкою миші на полі розташування блоків і у списку, що відкрився, вибрати команди «Insert New Object» → «Data Block».

У вікні параметрів, що відкривається при цьому, треба призначити номер і символічне ім‘я блока даних, перемкнути тип блока із «Shared DB» на «Instance DB» (екземплярний) і вказати номер функціонального блока. Після внесення цих даних слід закрити вікно кнопкою ОК. При цьому інтерфейс функціонального блока буде скопійовано у блок даних. На рисунку 5.37 для прикладу показаний вміст блока даних DB1, створеного для функціонального блока FB1.



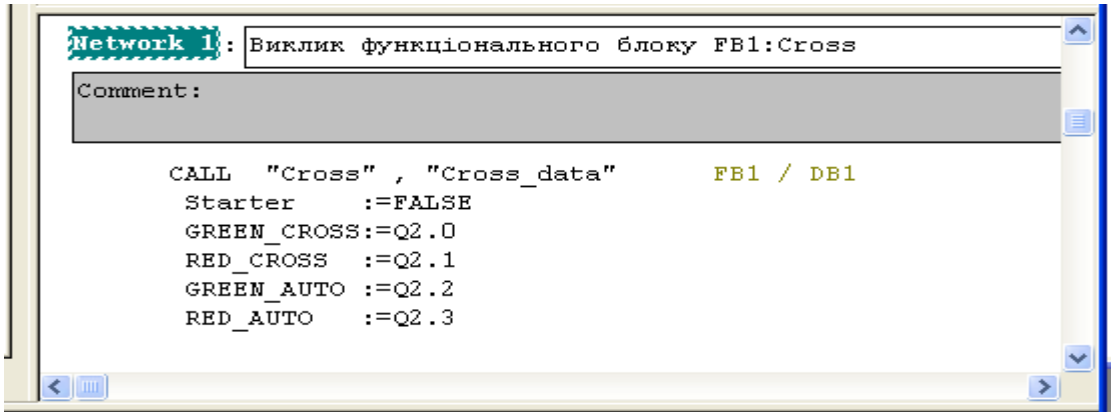
DB1 -- S7_STL_Project\SIMATIC 300(1)\CPU 315-2 DP

	Address	Declaration	Name	Type	Initial value	Actual value	C
1	0.0	in	Starter	BOOL	FALSE	FALSE	
2	2.0	out	GREEN_CROSS	BOOL	FALSE	FALSE	
3	2.1	out	RED_CROSS	BOOL	FALSE	FALSE	
4	2.2	out	GREEN_AUTO	BOOL	FALSE	FALSE	
5	2.3	out	RED_AUTO	BOOL	FALSE	FALSE	
6	4.0	stat	status	BOOL	FALSE	FALSE	
7	4.1	stat	g_auto	BOOL	FALSE	FALSE	
8	4.2	stat	dur_cross	BOOL	FALSE	FALSE	
9	4.3	stat	dur_auto	BOOL	FALSE	FALSE	

Рисунок 5.37 – Відображення даних у блоці даних DB1

Після створення функціонального блока й екземплярного блока даних залишається організувати виклик цих блоків у головній програмі керування, яка знаходиться в OB1.

Відкривши OB1, у першому сегменті вводимо команду CALL FB1, DB1 і натискуємо Enter. При цьому редактор автоматично змінює абсолютні номери блоків на символічні імена «Cross» і «Cross_data», де лапки означають, що це глобальні символи, а також виводить для ініціалізації вхідні та вихідні змінні. Вхідні змінні повинні мати конкретне значення, а для вихідних указується адреса розташування. Приклад програмного елемента виклику показано на рисунку 5.38.



```
Network 1: Виклик функціонального блоку FB1:Cross
Comment:

      CALL  "Cross" , "Cross_data"       FB1 / DB1
       Starter     :=FALSE
       GREEN_CROSS:=Q2.0
       RED_CROSS   :=Q2.1
       GREEN_AUTO  :=Q2.2
       RED_AUTO    :=Q2.3
```

Рисунок 5.38 – Організація виклику функціонального блока у програмі

Контрольні питання

1. Якими операціями здійснюється перевірка станів бітів?
2. Що містить логічний крок двійкової операції?
3. Яким чином враховуються типи контактів при програмування перевірки стану?
4. Як програмуються вкладені операції перевірки бітів?
5. Які функції належать до операцій з пам'яттю?
6. Як перевіряється наявність фронту сигналу?
7. Яким способом можна зберегти факт виявлення фронту сигналу?
8. Якими операціями здійснюється пересилання даних?
9. Які типи таймерів доступні користувачу?
10. Як задається тривалість для таймера?
11. Яке значення має тимчасова база для завдання тривалості?
12. Від чого залежить швидкість роботи лічильників?
13. Які операції можна застосовувати при програмуванні лічильника?
14. У якому порядку програмуються операції порівняння?
15. У якому порядку програмуються арифметичні операції?
16. У якому порядку програмуються математичні операції?
17. У якому порядку програмуються функції перетворення?
18. Як програмуються функції зсуву?
19. З яких бітів складається слово станів?
20. Які біти дозволяють використовувати функції переходів?
21. За яких умов виконується перехід в іншу точку програми?
22. Які функції застосовуються для оброблення кодових блоків?
23. З яких процедур складається оброблення виклику блока?
24. Якими процедурами завершується оброблення блока?
25. Що необхідно для виклику функції?
26. Що необхідно для виклику системного блока?

6 ПРОГРАМУВАННЯ НА МОВІ SCL

6.1 Призначення адрес і типів даних у мові SCL

Структурована мова програмування SCL (Structured Control Language) є мовою високого рівня для SIMATIC S7. Мова базується на стандарті 61131-3 і містить у собі елементи мови Паскаль поряд з такими типовими для PLC елементами, як «вхід» і «вихід».

SCL доцільно застосовувати для програмування складних алгоритмів або для завдань, що належать до області керування даними.

Створення програми здійснюється в редакторі SCL. Вихідна програма компілюється за допомогою SCL-компілятора в програму користувача.

Незважаючи на те, що елементи мови SCL (оператори, вирази, присвоєння значень) відрізняються в синтаксисі від інструкцій STL, у SCL використовуються ті самі типи даних, адресні області, символьні імена й блокова структура.

Для того щоб створити програму на SCL, потрібно спочатку створити проект. Далі в проекті призначити CPU, тому що створювана програма залежна від типу CPU. Тільки після цього створюються необхідні блоки програми.

Вихідна SCL-програма може містити, як мінімум, один блок. Можна також створити декілька вихідних SCL-програм, які потім потрібно скомпілювати у певному порядку з використанням керуючого файлу компілятора.

Під час програмування мовою SCL застосовуються такі адреси й змінні:

- входи I та виходи Q;
- периферійні входи PI та периферійні виходи PQ;
- адреси глобальних даних D;
- меркери M;
- тимчасові та статичні локальні дані із символьною адресацією;
- організаційні блоки OB, функціональні блоки FB, функції FC, а також блоки даних DB.

Функції таймерів T і функції лічильників C обробляються в SCL-програмах як *стандартні функції*.

Під час призначення типу даних визначаються:

- характер і значення даних;
- дозволені діапазони, наприклад, числовий діапазон чи довжина рядка символів;
- дозволені операції для оброблення даних.

У SCL є можливість групування значень типів даних, які представляють однакове поводження усередині одного класу. Групування дозволяє одержати такі класи:

- клас ANY_INT, що містить у собі дані типів INT і DINT;
- клас ANY_NUM, що містить у собі дані типів INT, DINT і REAL;
- клас ANY_BIT, що містить дані типів BOOL, BYTE, WORD і DWORD.

Зазначені класи типів даних дозволяють ясніше описати оператори. Однак для описування змінних при їх оголошенні застосовувати зазначені класи не можна.

Особливий клас даних представляють константи. Константи – це фіксовані значення, які при виконанні програми не змінюються. Константи використовуються для завдання початкових значень змінних або для їхнього об'єднання (комбінування) у програмі з іншими змінними, наприклад, як граничні значення.

У мові SCL константа не визначає «свій» тип даних, поки вона не буде оброблена в арифметичній операції. Наприклад, константа 1234 може належати до типу даних INT або до типу даних REAL, залежно від застосування:

```
int1 := int2 + 1234;        //константа INT
real1 := real2 + 1234;      //константа REAL.
```

У мові SCL можна призначати тип даних для константи зі специфікацією «type-defined», використовуючи відповідний префікс. Можна, наприклад, визначити змінну WORD у розділі оголошень за допомогою десяткового, шістнадцятиричного, восьмеричного або двійкового числа.

Приклади:

```
W1:   WORD  :=W#1234 ;                          //десяткове
W2:   WORD  :=W#16#04D2 ;                       // шістнадцятирічне
W3:   WORD  :=W#8#2322 ;                        //восьмеричне
W4:   WORD  :=W#2#0000_0100_1101_0010;  //двійкове.
```

Абсолютна адреса завжди належить до класу типів даних ANY_BIT. Так, наприклад, подвійне слово меркерів MD10 має тип даних DWORD.

Операнд може мати тип даних, що відрізняються від ANY_BIT тільки у двох випадках: якщо має символьне ім'я і якщо тип даних перетворений.

```
MW14 := SHL(IN := MW12, N := 2);                //Операнд має
//символьне ім'я
real1 := real2 + DWORD_TO_REAL(MD10);       //Тип даних операнда
//перетворений.
```

Рядок символів повинен вводитися в одинарних лапках. З таким типом можуть також використовуватися керуючі символи, що не друкуються; вони повинні вводитися у форматі $hh, де hh означає значення ASCII символу:

```
string1:= '$0A$0D';                 //новий рядок.
```

Під час абсолютної адресації адреси призначаються відповідно до початку адресної області; наприклад, I 0.0 (перший біт вхідного байта 0). Абсолютна адресація в SCL відповідає абсолютній адресації в стандартних мовах програмування за винятком способу ідентифікації адрес глобальних даних (табл. 6.1).

У мові програмування SCL доступ до адрес *глобальних даних* можливий тільки способом повної адресації. Доступ до блока даних здійснюється викликом змінної типу BLOCK_DB.

Таблиця 6.1 – Ідентифікація адрес при абсолютній адресації

Адресна область	Біт	Байт	Слово	Подвійне слово
Входи	Iy.x	IBy	IWy	IDy
Виходи	Qy.x	QBy	QWy	QDy
Периферійні входи		PIBy	PIWy	PIDy
Периферійні виходи		PQBy	PQWy	PQDy
Меркери	My.x	MBy	MWy	MDy
Адреси глобальних даних	DBz.DXy.x DBz.Dy.x	DBz.DBy	DBz.DWy	DBz.DDy

Примітки: x – адреса біта, y – адреса байта, z – номер блоку даних

При символьній адресації символьні імена повинні бути призначені абсолютним адресам і змінним. Для глобальних даних імена призначаються у таблиці символів; для локальних даних імена призначаються в розділі оголошення змінних блока.

Символьна адресація в SCL відповідає символьній адресації в стандартних мовах програмування. Можна також використовувати змішані абсолютно-символьні ідентифікатори, наприклад:

DB10.Setpoint;

“Motor1Data”.DW12.

Непряме використання глобальних адрес ґрунтується на абсолютній адресації. При цьому для вказівки розташування даних у пам'яті в квадратних дужках вказується:

- одна змінна INT, якщо дані представлені байтом;
- дві змінні INT, якщо дані представлені бітом.

Наприклад:

I [*byteindex.bitindex*]; //Дані представлені бітом;

MB [*byteindex*]; //Дані представлені байтом.

Тут *byteindex* і *bitindex* є виразами типу INT.

Таким способом можна адресувати такі області:

- периферійні входи PI і периферійні виходи PQ (в обох цих випадках адреса біта не вказується);
- входи I, виходи Q, меркери M;
- адреси глобальних даних D (блок даних і адреса даних);
- тимчасові й статичні локальні дані (тільки символьна адресація);
- функції таймерів T і лічильників C (адреса біта не вказується).

Синтаксис адресації блока:

DB10.DX [*byteindex.bitindex*]; //Адреса блока й адреса даних;

MotorData.DW [*byteindex*]; //Символьне ім'я блока, адреса даних.

Тут *byteindex* і *bitindex* є виразами типу INT.

Застосовуючи функцію перетворення WORD_TO_BLOCK, можна призначити непряму адресу блока даних. Номер блока даних DB визначається або як змінна, або як вираз з типом даних WORD:

WORD_TO_BLOCK_DB [*dbindex*].DW0.

Тут *dbindex* є змінною або виразом з типом даних WORD.

Якщо блок даних адресований непрямим способом, то для доступу до адреси даних не може використовуватися символьне ім'я. Так, наприклад, вхідний параметр Data можна викликати такими способами:

- Data.DW0;
- Data.DX2.0;
- Data.DW[*byteindex*];
- Data.DX[*byteindex.bitindex*].

Тут *byteindex* і *bitindex* є або константами, або змінними, або виразами типу INT. Індекс може бути змінений у процесі виконання програми.

6.2 Правила використання виражень і операторів

Оператори використовуються у виразах для одержання певних значень.

Приклад виразу: a + b.

Тут *a* і *b* – ідентифікатори даних (змінні), «+» – оператор.

Список операторів, які підтримує мова програмування SCL, наведений у таблиці 6.2.

Таблиця 6.2 – Список операторів і їхніх пріоритетів у мові SCL

Комбінування	Назва	Оператор	Пріоритет
Дужки	<Вираз>	(…)	1
Арифметичні операції	Степінь	**	2
	Знак (плюс, мінус)	+, –	3
	Множення, ділення	*, /, MOD, DIV	4
	Додавання, вирахування	+, –	5
Порівняння	Менше, менше або дорівнює	<, <=	6
	Більше, більше або дорівнює	>, >=	6
	Дорівнює, не дорівнює	=, <>	7
Двійкові	Логічне «НІ»	NOT	3
	Логічне «І»	AND, &	8
	«Виключальне АБО»	XOR	9
	Логічне «АБО»	OR	10
Присвоєння		:=	11

Оператори, що належать до одного пріоритетного класу, виконуються послідовно зліва направо.

Вирази розділяються на арифметичні, логічні й вирази порівняння.

Вираз – це формула, у якій існує певний порядок обчислення змінної. Цей порядок зазвичай регулюється за допомогою дужок. Результат, отриманий під час виконання виразу, може бути привласнений змінній або параметру блоку, або може бути використаний для перевірки критерію умови в інструкції керування.

Арифметичний вираз може складатися або із числових значень, або з їхніх символьних імен. Типи даних, які припустимі в арифметичних виразах, наведені в таблиці 6.3.

Приклади використання арифметичних виразів:

```
Power      := Voltage * Current;
Volume     := 4/3*PI*Radius**3;
MeanValue  := (Motor[1].Power + Motor[2]. Power)/2;
Solution1  := -P/2 + SQRT(SQR (P/2) – Q).
```

Таблиця 6.3 – Типи даних, припустимі в арифметичних операціях

Операція	Оператор	1 операнд	2 операнд	Результат
Множення	*	ANY_NUM TIME	ANY_NUM ANY_INT	ANY_NUM TIME
Ділення	/	ANY_NUM	ANY_NUM	ANY_NUM
Ділення цілих	DIV	ANY_INT	ANY_INT	ANY_INT
Додавання Вирахування	+	ANY_NUM TIME	ANY_NUM TIME	ANY_NUM TIME
Зведення в степінь	**	ANY_NUM	INT	REAL

Логічний вираз поєднує два операнда або вирази, що належать до класу типів даних ANY_BIT, відповідно до логіки AND (І), OR (АБО) або XOR (Виключальне АБО).

Приклади:

```
Q 4.0      := I 1.0 & I 1.1;
Pulses           := (Edge_mem_bits XOR ID 16) AND ID 16;
MW 30      := MW 32 AND Mask.
```

Логічний вираз видає значення, що належить до класу типів даних ANY_BIT. Результат логічного виразу належить до типу BOOL, якщо обидва операнда також мають тип BOOL. Якщо один або обидва операнди мають тип BYTE, WORD або DWORD, то результат буде мати той тип даних, що більш «вимогливий» до пам'яті операнда.

До логічного виразу належить також інвертування. Ця операція аналогічна зміні знака значення.

Приклад:

```
Automatic AND NOT Manual_on.
```

Вираз порівняння дозволяє одержати результат у вигляді булевого значення – при виконанні умови порівняння результат дорівнює TRUE (ІСТИНА), при невиконанні результат дорівнює FALSE (НЕПРАВДА).

Приклади:

```
ToolLarge:= VoltageAct > VoltageSet;
Warning:= (Voltage*Current) > 20_000;
IF Deviatin > 2_000 THEN Display := 16#F002; END_IF.
```

Порівнювані операнди повинні належати до одного типу даних або до одного класу типів даних (ANY_INT, ANY_NUM або ANY_BIT).

За допомогою операції *присвоєння значення* одна змінна одержує значення іншої змінної або значення виразу. Ліворуч від оператора присвоювання «:=» перебуває змінна, яка набуває значення змінної, що знаходиться праворуч від оператора присвоювання.

Типи даних, які перебувають із двох сторін від знака присвоювання, повинні бути ідентичними. Виключення становить тільки випадок явного присвоювання типу даних.

Приклади:

```
Automatic   := TRUE;
Deviation   := ActualValue – SetPoint;
Display     := INT_TO_WORD (Deviation).
```

Абсолютні адреси мають типи даних ANY_BIT, які обумовлені розміром займаної області – BOOL, BYTE, WORD, DWORD. Якщо необхідно призначити інший тип даних, то доцільно використовувати оператори перетворення.

У випадку використання масивів можна:

- звертатися до масиву в цілому;
- звертатися до частини масиву;
- звертатися до елемента масиву.

При перетворенні типів даних у масивах варто враховувати, що типи даних елементів масивів повинні бути погоджені. При цьому повинні збігатися граничні значення індексів у кожній розмірності.

6.3 Особливості застосування операторів керування програмою

За допомогою операторів керування (control statements) користувач може організувати розгалуження програми, циклічне виконання окремих фрагментів програми й здійснювати перехід у програмі блока для виконання іншої її частини.

Мова програмування SCL підтримує такі оператори керування:

- IF – оператор для виконання розгалуження в програмі за умовою, що перевіряється у відношенні до булевої змінної;
- CASE – оператор для виконання розгалуження в програмі за умовою, що перевіряється у відношенні цілої змінної або параметра типу INT;
- FOR – оператор для організації в програмі циклів зі змінною, що є лічильником циклів;
- WHILE – оператор для організації в програмі циклів, які ініціюються при виконанні певної умови;
- REPEAT – оператор для організації в програмі циклів із завершенням за умовою;
- CONTINUE – оператор для завершення поточного проходу циклу в програмі;
- EXIT – оператор для виходу із циклу в програмі;

- GOTO – оператор для переходу в нову точку програми, зазначену міткою;
- RETURN – оператор для виходу із програми блока.

Оператор IF

Оператор IF керує виконанням тієї або іншої частини програми залежно від стану булевої змінної. За допомогою оператора IF користувач може запрограмувати виконання програми різними, обумовленими умовами, гілками.

```
IF condition
        THEN statements;
END_IF;
```

Тут condition – це адреса або вираз з типом BOOL. Якщо condition має значення TRUE, то виконуються оператори після ключового слова THEN. Якщо condition має значення FALSE, то виконуються оператори після ключового слова END_IF. Ключове слово END_IF завершує оператор IF.

```
IF condition
        THEN statements1;
        ELSE statements0;
END_IF;
```

У цьому прикладі, як і в попередньому, condition має значення TRUE або FALSE. Якщо condition має значення TRUE, то виконуються оператори після ключового слова THEN. Якщо condition має значення FALSE, то виконуються оператори після ключового слова ELSE.

```
IF condition1
        THEN statements1;
        ELSEIF condition2
              THEN statements2;
        ELSE statements0;
END_IF;
```

Можна використовувати будь-яку кількість комбінацій ключових слів ELSEIF ... THEN … між ключовими словами IF ... THEN … і ELSE. Ключове слово ELSE і наступні оператори не є обов'язковими.

Приклад:

```
IF Actual_value > Setpoint
        THEN greater_than:= TRUE;
              less_than    := FALSE;
              equal_to     := FALSE;
        ELSEIF Actual_value < Setpoint
              THEN greater_than := FALSE;
                    less_than:= TRUE;
                    equal_to:= FALSE;
              ELSE greater_than:= FALSE;
                    less_than    := FALSE;
                    equal_to     := TRUE;
END_IF.
```

У наведеному прикладі, якщо змінна *Actual_value* більше, ніж змінна *Setpoint*, то виконуються оператори, що йдуть після ключового слова THEN. Якщо, навпаки, змінна *Actual_value* менше, ніж змінна *Setpoint*, то виконуються оператори, що йдуть після ключового слова ELSEIF. Якщо обидва вирази порівняння не виконуються, то виконуються оператори після ключового слова ELSE.

Оператор CASE

Оператор CASE дозволяє вибрати для виконання потрібну послідовність операторів у програмі залежно від значення цілої змінної – параметра типу INT.

Загальна структура програми з оператором CASE може мати таку форму:

```
CASE Selection OF
        CONST1 : statements1;
        CONST2 : statements2;
        …
        CONSTx : statementsx;
        ELSE : statements0;
END_CASE;
```

В цьому прикладі Selection – це адреса або вираз із типом INT. Якщо Selection має значення CONST1, то виконуються оператори statements1. Якщо Selection має значення CONST2, то виконується ланцюжок операторів statements2 і так далі. Якщо Selection має значення, що перебуває за межами списку значень, зазначених для перевірки, то виконується ланцюжок операторів після ключового слова ELSE. При цьому ланцюжок операторів, що починається із ключового слова ELSE, не є обов'язковим.

Список значень CONST1, CONST2 і т. д. складається із цілих (INT) констант. Для цих констант можуть використовуватися кілька варіантів форматів запису. В якості константи-перемикача CONSTx можуть бути зазначені:

- одиночне ціле число;
- діапазон цілих чисел, наприклад, 15…20;
- суміш розділених комами окремих цілих чисел і діапазонів цілих чисел, наприклад, 21,25,31…33.

При цьому кожне значення константи-перемикача CONSTx повинне бути унікальним.

Приклад

Нехай значення, що привласнюється змінній *Error_number*, залежить від змінної *ID*. Тоді програма може виглядати так:

```
CASE ID OF
        0       : Error_number := 0;
        1,3,5   : Error_number := ID + 128;
        …
        6..10   : Error_number := ID;
        ELSE : Error_number := 16#7F;
END_CASE;
```

Оператор FOR

Оператор FOR застосовується для організації циклів з лічильником циклів. Виконання внесеного в цикл фрагмента програми буде повторюватися стільки довго, скільки змінна (лічильник циклів) буде залишатися в зазначеному діапазоні значень.

Загальна структура програми з оператором FOR може мати таку форму:

```
FOR i := limit1
        TO limit2
        BY step
        DO statements;
END_FOR;
```

Під час оброблення цього оператора початкове значення limit1 привласнюється лічильнику циклів «i». Змінна, яка призначена лічильником циклів, повинна бути типу INT або DINT. Вона повинна мати початкове й кінцеве значення, а також крок зміни step.

Після кожного проходу програми (циклу) лічильник циклу збільшується на величину кроку збільшення step, якщо крок зазначений як позитивне число, або зменшується на величину кроку step, якщо крок зазначений як негативне число.

Під час програмування циклу рядок BY step не є обов'язковим. Якщо така умова для кроку лічильника циклу відсутня, то крок (за замовчуванням) приймається таким, що дорівнює +1. Якщо величина змінної-лічильника циклу виходить за межі зазначеного діапазону, то програма переходить до оператора, що стоїть після ключового слова END_FOR.

Оператор FOR може бути вкладеним, тобто усередині циклу з оператором FOR можна запрограмувати інші цикли з оператором FOR, у яких як лічильники циклу використовуються інші змінні.

Усередині циклу з оператором FOR може бути запрограмований перехід до початку циклу (з використанням оператора керування CONTINUE) або повний вихід із циклу для продовження виконання програми, починаючи відразу ж після ключового слова END_FOR, (з використанням оператора керування EXIT).

Приклад

Нехай, наприклад, необхідно привласнити значення слів з PIW 128 по PIW 142 області периферії словам в області меркерів – з MW 128 по MW 142.

Програма цієї процедури буде мати такий вигляд:

```
FOR i := 128 TO 142 BY 2 DO
        MW[i] := PIW[i];
END_FOR;
```

Оператор WHILE

Оператор WHILE служить для організації циклів, виконання яких триває весь час, поки виконується певна умова.

Загальна структура програми з оператором WHILE може мати таку форму:

```
WHILE Condition DO
    Statements;
END_WHILE.
```

У цьому прикладі Condition – це адреса або вираз типу BOOL. Поки виконується певна умова, тобто поки Condition = TRUE, будуть циклічно виконуватися вирази, представлені операторами Statements.

Перед кожним проходом виконується перевірка умови Condition. Якщо умова не виконується (Condition = FALSE), програма переходить до оператора, розташованого після ключового слова END_WHILE. Такий перехід можливий навіть без проходу програми циклу. Оператори Statements при цьому жодного разу не будуть виконані.

Оператор WHILE може бути вкладеним. При цьому усередині одного циклу з оператором WHILE можуть розміщатися інші цикли з оператором WHILE.

Усередині циклу з використанням оператора WHILE може бути запрограмований перехід до початку циклу з використанням оператора керування CONTINUE або повний вихід із циклу з використанням оператора керування EXIT. Після виходу із циклу у програмі буде виконуватися оператор, розташований після ключового слова END_WHILE.

Оператор REPEAT

Оператор REPEAT служить для організації циклів, виконання яких триває увесь час, поки не зустрінеться умова завершення оброблення циклу.

Загальна структура програми з оператором REPEAT може мати таку форму:

```
REPEAT
    Statements;
    UNTIL Condition
END_REPEAT;
```

У цьому прикладі Condition – це адреса або вираз типу BOOL. Поки не виконується певна умова, тобто поки Condition = FALSE, будуть циклічно виконуватися вирази Statements.

Після кожного проходу циклу виконується перевірка умови Condition. Якщо умова виконується (Condition = TRUE), то цикл далі не обробляється й виконання програми буде продовжено відразу ж після ключового слова END_REPEAT.

Таким чином, програма циклу буде оброблена, принаймні, один раз, навіть якщо при першому проході циклу виконується умова завершення його оброблення.

Оператор REPEAT може бути вкладеним. При цьому усередині одного циклу з оператором REPEAT можуть розміщатися інші цикли з оператором REPEAT.

Усередині циклу з використанням оператора REPEAT може бути запрограмований перехід до початку циклу з використанням оператора керування CONTINUE або повний вихід із циклу для продовження виконання програми, починаючи відразу ж після ключового слова END_REPEAT (з використанням оператора керування EXIT).

Нехай, наприклад, необхідно викликати системну функцію SFC 25 COMPRESS у програмі перезапуску, поки не буде завершено «стискання» пам'яті користувача.

```
REPEAT
        SFC_ERROR       := COMPRESS(
                BUSY    := busy,
                DONE    := done);
        UNTIL done
END_REPEAT;
```

Оператор CONTINUE

Оператор CONTINUE служить для завершення поточного проходу циклу в програмі, організованого за допомогою операторів FOR, WHILE або REPEAT.

Після виконання оператора CONTINUE у програмі виконується одна з таких дій:

- якщо цикл організований за допомогою операторів REPEAT або WHILE, перевіряється умова для виконання наступного проходу циклу;
- якщо цикл організований з оператором FOR, проводиться зміна лічильника циклу на величину кроку збільшення й перевіряється умова: чи перебуває змінна лічильника циклів у припустимих для неї межах.

Нехай, наприклад, необхідно встановити меркери за допомогою двох вкладених циклів з використанням оператора FOR. Установлення бітів повинно починатися з меркера M 0.3. Умова завершення циклу формулюється при цьому в такий спосіб: якщо байтова адреса (i) дорівнює 0, а бітова адреса (k) менше 2, оператори тіла внутрішнього циклу FOR не виконуються.

Програма:

```
FOR i := 0 TO 2 DO
        FOR k := 0 TO 7 DO
                IF (k<2 & i=0) THEN CONTINUE;
                END_IF;
                M[i,k] := TRUE;
                END_FOR;
        END_FOR;
```

Оператор EXIT

Оператор EXIT служить для повного завершення оброблення циклу, організованого за допомогою операторів FOR, WHILE або REPEAT. При цьому вихід із циклу з оператором EXIT не залежить від виконання умов, що перевіряються в циклі, і може виконуватися з будь-якої точки циклу. При виході із циклу з оператором EXIT програма продовжує виконуватися відразу ж після ключових слів END_FOR, END_WHILE або END_REPEAT.

Вихід із циклу з оператором EXIT здійснюється негайно із точки програми циклу, де цей оператор зустрівся.

Приклад

Нехай необхідно встановити меркери за допомогою двох вкладених циклів з використанням оператора FOR. Якщо байтова адреса (i) дорівнює

2, а бітова адреса (k) більше 5, то виконання внутрішнього циклу FOR переривається. Установлення бітів повинно закінчуватися на меркері M 2.5.

```
FOR i := 0 TO 2 DO
    FOR k := 0 TO 7 DO
        IF (k=2 & i>5) THEN EXIT;
        END_IF;
        M[i,k] := TRUE;
    END_FOR;
END_FOR;
```

У цьому прикладі виконання циклу FOR припиняється за певної умови для лічильника циклу *k* за допомогою оператора EXIT.

Оператор RETURN

Оператор RETURN служить для безумовного виходу з поточного блока й переходу в основну програму. При цьому оператор RETURN пересилає стан сигналу змінної OK на вихід ENO блока, який завершується з ним.

Приклад:

```
IF Error <> 0 THEN RETURN;
END_IF;
```

6.4 Особливості програмування SCL-блоків

У користувальницькій SCL-програмі використовуються кодові блоки і блоки даних. Кодові блоки – це викликувані для виконання підпрограми, а блоки даних – це описи атрибутів змінних, які беруть участь у виконанні цих підпрограм.

У мові програмування SCL використовується стандартна структура й стандартний інтерфейс блока. Тому окремі блоки, запрограмовані з використанням SCL, можна викликати, наприклад, в FBD-блоці й, навпаки, з SCL-блоків можна викликати блоки, створені з використанням, наприклад, мови STL.

Система STEP 7 підтримує функції FC і функціональні блоки FB як кодові блоки. На відміну від функцій FC, функціональні блоки FB викликаються разом із блоками даних, у яких зберігаються локальні змінні блока (пам'ять блока).

Параметри блока

Параметри блока забезпечують зв'язок між двома блоками, один із яких являється викликаючим, а другий – викликуваним. Ці параметри можуть бути оголошені як вхідні (Input), вихідні (Output) і вхідні – вихідні (In/Out) параметри. Вхідні параметри можуть бути тільки прочитані, вихідні параметри можуть бути тільки записані. Тому, якщо параметри повинні бути спочатку прочитані, потім змінені й, нарешті, знову записані, то варто використовувати тип In/Out.

У функціях FC параметри блоку є покажчиками на фактичні параметри або на інші покажчики.

У функціональних блоках FB параметри блока зберігаються в екземплярних блоках даних. Доступ до цих даних вимагає визначення блока даних і адреси даних.

Приклади:

Result := DB1.DW2;

Result := DB2.Total;

Result := Privod_data.Speed.

Формальні параметри

Для адресації блока використовуються формальні параметри. Вони мають такі ж імена, як і параметри блока, і використовуються у виразах програми в якості операндів.

Формальні параметри параметричних типів TIMER і COUNTER можуть бути оброблені з використанням функцій таймерів і функцій лічильників. Значення формальних параметрів таких типів можуть бути передані у параметри викликуваних блоків.

За допомогою формальних параметрів типу BLOCK_xx можна одержати доступ до адрес даних у блоці даних. Формальні параметри параметричних типів BLOCK_FB і BLOCK_FC при використанні мови програмування SCL можуть тільки передаватися у викликувані блоки, тому що команд для оброблення формальних параметрів такого типу в блоці немає.

Формальні параметри типів даних POINTER і ANY можуть бути передані у викликувані блоки цілком. Виключення становлять фактичні параметри, розміщені в тимчасових локальних даних, передача яких не допускається.

Статичні локальні дані

Статичні локальні дані – це пам'ять функціонального блока. Ці дані розташовуються в *екземплярному блоці даних*. При цьому значення даних зберігаються доти, поки не будуть змінені програмою. У статичних локальних даних можна розміщати також локальні екземпляри функціональних блоків і системних функціональних блоків.

Статичні локальні дані оголошуються за допомогою ключових слів VAR і END_VAR.

В якості статичних локальних даних допускається застосовувати всі прості, складні, користувальницькі (UDT) типи даних, а також типи даних POINTER і ANY.

Якщо задавати значення ініціалізації не потрібно, змінні можна повідомляти списком.

Приклад:

```
VAR
        VAR1, VAR2, VAR3 : INT;
        VAR4: INT:=3;
END_VAR.
```

Необхідно врахувати, що під час програмування мовою SCL статичні локальні дані у функціональному блоці можуть бути адресовані тільки символьним способом.

Попередня ініціалізація параметрів блоків

Попередня ініціалізація параметрів блоків не обов'язкова й допускається тільки у відношенні таких функціональних блоків, параметри яких зберігаються як значення. Це поширюється на будь-які параметри блоків із простими типами даних, а також на вхідні (Input) і вихідні (Output) параметри складних типів даних.

Якщо ініціалізація параметрів блоків не зроблена, то редактор буде використовувати як початкові значення параметрів або нульове значення, або найменше значення, або пробіл (залежно від типу даних). Для параметрів типу BLOCK_DB як значення за замовчуванням приймається DB1 (DB0 не існує).

Виклик SCL-блоків

Під час програмування блоків мовою програмуванні SCL розрізняють блоки з функціональним значенням і блоки без функціонального значення.

Функціональні блоки FB і функції FC без функціонального значення є просто «галузями» програми, тобто підпрограмами.

Функції FC з функціональним значенням можуть використовуватися у виразах присвоювання, а також в інших виразах у якості операндів.

Системні функціональні блоки SFB викликаються так само, як і функціональні блоки FB, а системні функції SFC викликаються так само, як функції FC. Варто врахувати, що системний функціональний блок SFB викликається із блоком даних При цьому блок даних розміщається в користувальницькій програмі.

Під час виклику блока його параметри ініціюються *фактичними параметрами*, які являють собою константи, змінні або вирази. Саме із цими параметрами блок обробляється при виконанні програми, і саме в них зберігаються результати цієї обробки.

Під час виклику функцій FC і системних функцій SFC всі параметри блока повинні бути ініційовані. У разі виклику функціональних блоків FB і системних функціональних блоків SFB ініціалізація параметрів блока не обов'язкова.

Вихідні параметри під час виклику функціональних блоків FB і системних функціональних блоків SFB ініціюються за допомогою прямого звертання до екземплярних даних.

Приклад виклику функції FC без функціонального значення:

FC291 (MAX := Maximum, IN := InputVar, MIN := Minimum, OUT := Result).

Під час виклику використовується або абсолютна, або символьна адреса блока. Список параметрів наводиться в дужках. Дужки повинні бути зазначені, навіть якщо функція FC не має параметрів.

Порядок ініціалізації параметрів може бути довільним. Якщо функція має єдиний вхідний параметр, то ім'я параметра при ініціалізації може бути опущено.

Приклад ініціалізації з перетворенням INT-змінної Speed в STRING-змінну Display: Display := I_STRING(Speed).

Приклад виклику функції FC з функціональним значенням:

Result := FC2 (MAX := Maximum, IN := InputVar, MIN := Minimum).

У цьому прикладі глобальній змінній Result привласнюється функціональне значення функції FC2. Функції FC з функціональним значенням можуть використовуватися у якості операндів у будь-яких вираженнях.

Якщо під час виклику блока використовується вхідний параметр EN і цей вхід має значення FALSE, то функціональне значення не буде визначено, тобто функціональному значенню не буде привласнено ніякої величини.

Екземплярний блок даних специфікується під час виклику функціонального блоку.

Приклад виклику функціонального блоку з екземплярним блоком даних:

FB2.DB2 (IN:= InputValue);

Result := DB2.OUT.

У записі виклику спочатку записується адреса функціонального блоку, а потім відділена крапкою адреса екземплярного блоку даних. Далі в дужках представляється список параметрів.

Оскільки вхідні та вихідні параметри складних типів зберігаються як покажчики, вони повинні бути ініційовані значущими величинами при першому виклику функціонального блоку. Якщо параметр блоку не ініційований, то він зберігає своє останнє певне значення.

Всі параметри блока даних можуть бути адресовані як глобальні дані із вказівкою імені екземплярного блока даних й імені параметра. Вони також можуть бути ініційовані до виклику функціонального блока за допомогою операторів присвоювання:

DB2.MAX := Maximum;

DB2.MIN := Minimum.

Функціональний блок як локальний екземпляр

Функціональні блоки можуть бути оголошені як *локальні екземпляри* й викликані в іншому функціональному блоці. Такі викликувані функціональні блоки зберігають свої локальні дані в екземплярному блоці даних визивного функціонального блока.

Приклад

```
FUNCTION_BLOCK FB2
…
VAR
        Delimiter : FB3;
END_VAR
BEGIN
Delimiter (IN := InputVar);
Result := Delimiter.OUT;
…
END_FUNCTION_BLOCK.
```

Оголошення екземплярів виконується в статичних локальних даних за ключовим словом VAR. Тут локальному екземпляру (FB3) призначається ім'я, наприклад, Delimiter. До моменту компілювання викликуваний

функціональний блок повинен уже існувати або у вигляді раніше скомпільованого блока в розділі Blocks (*Блоки*), або у вигляді вихідної програми.

Аналогічні дії виконуються під час виклику системного функціонального блоку SFB.

Застосування механізму EN/ENO

Під час програмування на SCL користувач має можливість перевірити окремі вирази (операції) на правильність їхнього виконання й залежно від цього приймати рішення – виконувати наступний блок чи не виконувати.

Для реалізації цієї можливості в мові програмування SCL існує змінна з ім'ям «ОК» і типом даних «BOOL». Ця змінна повідомляє про помилки, що виникають під час виконання програми в SCL-блоці. На початку блоку ОК-змінна має значення TRUE. При виникненні програмної помилки вона скидається в стан FALSE. Результат перевірки повідомляється за допомогою спеціального виходу блока ENO (Enable output – *вихід розблокований*) типу BOOL.

Перевіряти ОК-змінну можна за допомогою відповідних SCL-операцій. Нижче наведено приклад одержання перевірки результату арифметичного виразу:

```
SUM := SUM + IN;
IF OK
        THEN (* не відбулося помилок *);
        ELSE (* помилка в операції додавання *);
END_IF.
```

Для того щоб викликуваний блок зберіг значення ОК-змінної на виході ENO, необхідно передбачити такі призначення:

```
FC15 (In1:= ..., In2:= ...);
OK : = ENO.
```

Після виклику блока значення ENO може бути використане для визначення того, чи правильно був оброблений блок. Якщо блок був оброблений правильно, то ENO = TRUE. Якщо під час оброблення блока відбулася помилка, то ENO = FALSE.

Приклад програмування аналізу ENO:

```
FC5 (In1:= ..., In2:= ...);
IF ENO
        THEN (* не відбулося помилок *);
        ELSE (* відбулася помилка *);
END_IF.
```

Для керування викликами блока використовується вхід EN типу BOOL. Якщо вхід EN ініційований значенням TRUE, то блок буде викликаний для виконання, якщо EN ініційований значенням FALSE, то блок не буде викликаний для виконання. При цьому буде виконаний перехід до наступного оператора після виклику блока.

Приклад виклику блока із вказівкою значення входу EN:

```
FC1 (EN:= I1.0, In1:= ..., In2:= ...); (*FC15 виконується тільки у випадку, коли I1.0 = «1» *)
```

Якщо EN не використовується, блок буде виконуватися завжди.

Приклад програмування виклику блоку FC2 у випадку, якщо оброблення FC1 успішно завершено:

FC1 (EN := I1.0, In1:= ..., In2:= ...);

FC2 (EN := ENO, In1:= ..., In2:= ...).

6.5 Особливості програмування SCL-функцій

До SCL-функцій належать функції *таймерів, лічильників, математичні, зсуву й перетворення*.

Програмування таймерів

У системній пам'яті CPU підтримується функціонування низки таймерів, які відрізняються режимам роботи:

S_PULSE – режим керованого імпульсу (pulse timer);

S_PEXT – режим розширеного імпульсу (extended pulse);

S_ODTON – із затримкою включення (delay);

S_ODTS – із затримкою включення з пам'яттю (latching ON delay);

S_OFFDT – із затримкою вимикання (OFF delay).

Всі функції таймера мають параметри, показані в таблиці 6.4.

Приклад виклику функцій таймерів:

```
Time_BCD:= S_PULSE (
T_NO:= Timer_address,
S:= Start_input,
TV:= Timer_duration,
R:= Reset,
Q:= Timer_status,
BI:= Binary_time);
```

Під час ініціалізації параметрів функцій таймерів варто враховувати, що параметр T_NO повинен бути ініційований завжди, а інші можна не ініціювати.

Таблиця 6.4 – Параметри SIMATIC-функцій таймерів

Параметр	Оголошення	Тип даних	Значення
T_NO	INPUT	TIMER	Адреса таймера
S	INPUT	BOOL	Параметр запуску
TV	INPUT	S5TIME	Встановлене значення
R	INPUT	BOOL	Скидання таймера
Функціональне значення	OUTPUT	S5TIME	Поточне значення у BCD форматі
Q	OUTPUT	BOOL	Стан таймера
BI	OUTPUT	WORD	Поточне значення таймера у двійковому коді

На додаток до SIMATIC-функцій таймерів у спеціалізованих CPU підтримуються також IEC-функції таймерів. Ця підтримка забезпечується системними функціональними блоками SFB:

SFB 3 TP – функція генерації імпульсу;

SFB 4 TON – функція генерації імпульсу із затримкою включення;

SFB 5 TOF – функція генерації імпульсу із затримкою вимикання.

Ці функціональні блоки зберігаються в бібліотеці *Standard library* у розділі *System Function Blocks (Системні функціональні блоки).*

Програмування лічильників

У системній пам'яті CPU підтримуються три функції лічильників:

S_CU – функція лічильника прямого рахунку (up counter);

S_CD – функція лічильника зворотного рахунку (down counter);

S_CUD – функція лічильника прямого і зворотного рахунку (up-down counter).

Параметри SIMATIC-функцій лічильників представлені в таблиці 6.5.

Приклад виклику функцій лічильників:

```
BCD_Count_Value   := S_CU (
C_NO              := Count_address,
CU                := Count_up,
S                 := Set_input,
PV                := Count_value,
R                 := Reset,
Q                 := Counter_status,
BI                := Binary_count_value);
```

Таблиця 6.5 – Параметри SIMATIC-функцій лічильників

Параметр	Оголошення	Тип даних	Значення
C_NO	INPUT	COUNTER	Адреса лічильника
CU	INPUT	BOOL	Прямий рахунок
CD	INPUT	BOOL	Зворотний рахунок
S	INPUT	BOOL	Параметр запуску
PV	INPUT	S5TIME	Встановлене значення
R	INPUT	BOOL	Скидання лічильника
Функціональне значення	OUTPUT	WORD	Поточне значення у BCD форматі
Q	OUTPUT	BOOL	Стан лічильника
CV	OUTPUT	WORD	Поточне значення у двійковому коді

Під час ініціалізації параметрів функцій лічильників застосовуються такі правила:

- не допускається застосування параметра CD разом з функцією лічильника S_CU, а також параметра CU разом з функцією лічильника S_CD (повинен бути встановлений один з параметрів CD або CU);

- параметр C_NO повинен бути ініційований завжди;
- параметри S і PV, а також параметри Q і CV можна не ініціювати.

Для ініціалізації параметра PV лічильника може бути застосоване ціле число типу INT як константа.

У спеціалізованих CPU підтримуються також IEC-функції лічильників (як системні функціональні блоки SFB):

SFB 0 CTU – функція лічильника прямого рахунку;
SFB 1 CTD – функція лічильника зворотного рахунку;
SFB 2 CTUD – функція лічильника прямого і зворотного рахунку.

Програмування математичних функцій

У мові програмування SCL підтримуються такі математичні функції:

тригонометричні функції:

SIN	функція синуса;
COS	функція косинуса;
TAN	функція тангенса;
ASIN	функція арксинуса;
ACOS	функція арккосинуса;
ATAN	функція арктангенса;

логарифмічні функції:

EXP	експонентна функція по основі *e*;
EXPD	експонентна функція по основі 10;
LN	натуральний логарифм;
LOG	десятковий логарифм;

інші функції:

ABS	функція виділення абсолютного значення;
SQR	функція знаходження квадрата числа;
SQRT	функція узяття квадратного кореня.

Математичні функції обробляють числа форматів INT, DINT і REAL. Під час використання вхідного параметра у форматі INT або DINT, число автоматично перетвориться в формат REAL. Виключення становить функція ABS, що видає результат того ж типу, до якого належало вихідне число.

Тригонометричні функції розглядають вхідні параметри як кути, виражені в радіанах, у діапазоні від 0 до 2π (де π = 3,141593e + 00), що відповідає діапазону в градусах від 0° до 360°.

Приклади програмування математичних функцій:

1. Розрахунок реактивної потужності Reactive_power, що визначається добутком напруги Voltage на струм Current і на синус фазового зсуву між ними:

```
Reactive_power := Voltage * Current * SIN(φ);
```

2. Розрахунок обсягу рідини Volume, що визначається добутком числа PI на квадрат радіуса основи посудини Radius і на рівень заповнення посудини Level:

```
Volume  := PI * SQR(Radius) * Level;
```

3. Розрахунок довжини гіпотенузи за відомими величинами катетів:

c := SQRT(SQR(a) + SQR(b)).

Програмування функцій зсуву

Загальний спосіб виклику функцій зсуву (Shifting) і циклічного зсуву (Rotating) має такий вигляд:

Result := Function(IN := Input_value, N := Shift_number).

Функції зсуву й циклічного зсуву мають *два вхідних параметри*.

Параметр IN визначає змінну, з якої необхідно виконати операцію зсуву або циклічного зсуву. Цей параметр належить до класу ANY_BIT, тобто, до типів BOOL, BYTE, WORD, DWORD. При цьому значення функції належить до того ж типу, що й вхідне значення.

Параметр N показує число бітів, на яке необхідно зробити операцію зсуву або циклічного зсуву. Цей параметр належить до типу INT.

До функцій зсуву належать:

SHL – зсув вліво;

SHR – зсув вправо.

До функцій циклічного зсуву належать:

ROL – циклічний зсув вліво;

ROR – циклічний зсув вправо.

Приклади:

MW14 := SHL(IN := MW12, N := 2);

res_dword := ROR(IN := in_dword, N := shift_int).

Використання функцій перетворення

У мові програмування SCL підтримуються *два типи функцій* перетворення – неявні та явні.

Неявні функції виконуються в SCL автоматично (неявно), тому що вони не пов'язані з втратою інформації. Прикладом може служити перетворення даних типу BYTE у дані типу WORD.

Явні функції користувач повинен ініціювати самостійно, явно. Прикладом може служити перетворення даних типу REAL у дані типу INT. Будь-яка втрата інформації може бути попереджена за допомогою відповідного контролю даних або ж користувач може перевіряти OK-змінну для перевірки результатів виконання операцій.

Варто врахувати, що під час використання деяких явних функцій перетворення, по-перше, перетворення даних по суті не відбувається й не виконується ніякий код, а по-друге, деякі з функцій перетворення впливають на стан змінної OK.

Контрольні питання

1. Для рішення яких завдань застосовується мова SCL?

2. Що треба визначити для призначення типу даних у SCL-операндів?

3. Які області даних використовуються при адресації у мові SCL?

4. Який тип даних визначає константа?

5. Який синтаксис застосовується для адресації змінних у SCL?

6. Що являють собою *логічні вираження* у мові SCL?

7. Що являють собою *математичні вираження* у мові SCL?

8. Як розподіляються приорітети операторів?

9. За допомогою яких операторів можна організувати розгалуження програми?

10. За яких умов застосовується оператор IF?

11. Які завдання вирішуються з використанням оператора CASE?

12. За допомогою яких операторів можна організувати циклічні процедури?

13. Якими операторами можна завершити циклічні процедури?

14. Як здійснюється виклик і завершення роботи блоків програми?

7 ПРОГРАМУВАННЯ МОВОЮ S7-GRAPH

7.1 Особливості мови S7-GRAPH

Мова програмування S7-GRAPH призначена для створення систем послідовного керування.

У разі використання цієї мови процес розділяється на окремі кроки, забезпечуючи наочний огляд функціонування системи. Графічне подання функціонального блоку FB S7-GRAPH називається *секвенсором*.

Секвенсор містить послідовність кроків, які повинні виконуватися у певному порядку, і послідовності переходів, які визначають умови для переходу до наступного кроку. Найпростіша структура секвенсора – лінійна послідовність кроків і переходів без розгалуження. Лінійний секвенсор починається із кроку й завершується переходом.

Приклад лінійного секвенсора показаний на рисунку 7.1.

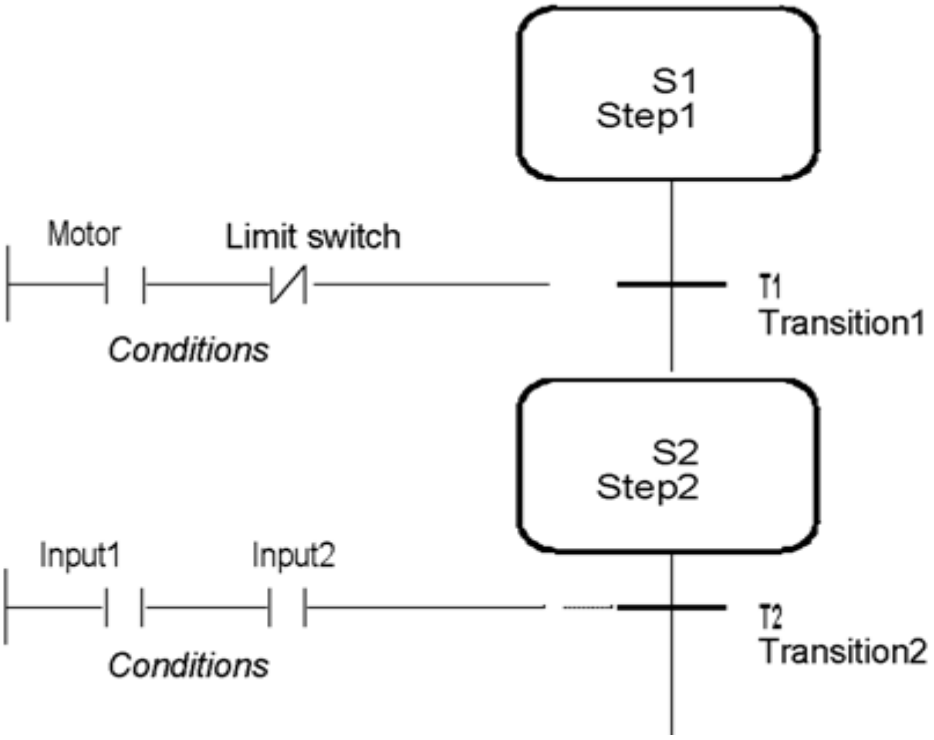


Рисунок 7.1 – Лінійна структура секвенсора

До створення програми секвенсора необхідно розробити концепцію керування технологічним процесом, розбивши процес на окремі кроки.

Основою для розроблення концепції служить технологічна схема процесу й тимчасова діаграма процесу (рис. 7.2).

Складність системи послідовного керування залежить від розв'язуваного завдання автоматизації. Однак у загальному випадку буде потрібно, принаймні, три блоки:

1. Блок STEP 7, у якому викликається функціональний блок S7-GRAPH. Це може бути організаційний блок (OB), функція (FC) або інший функціональний блок (FB);

2. Функціональний блок FB S7-GRAPH, що описує окремі підзадачі й взаємозалежності системи послідовного керування;

3. Екземплярний блок DB, який містить дані й параметри системи послідовного керування. Екземплярний DB призначається функціональному блоку FB S7-GRAPH і може бути створений системою автоматично.

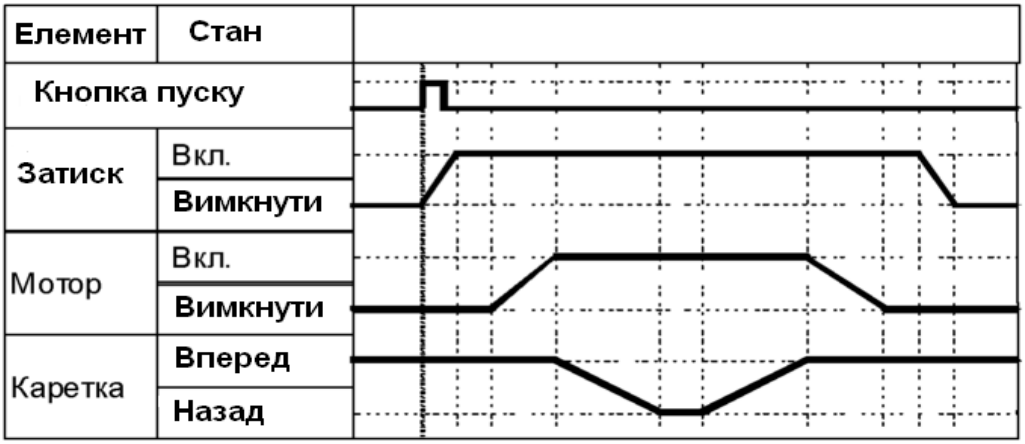


Рисунок 7.2 – Приклад тимчасової діаграми процесу

Структура секвенсора повинна задовольняти таким правилам:

- FB S7-GRAPH може містити до 256 кроків і переходів. Кроки й переходи вставляються тільки парами;
- секвенсор може містити максимум 256 гілок, у тому числі до 125 альтернативних або до 249 паралельних гілок. Практично недоцільно створювати більш ніж 20–30 гілок;
- гілка може бути замкнута тільки на гілку, яка розташована ліворуч;
- наприкінці гілки після переходу можуть бути встановлені перегони. Вони спричиняють з'єднання з попереднім кроком у тій же послідовності або в іншій послідовності в тім же FB;
- останов секвенсора може бути доданий після переходу наприкінці гілки;
- постійні інструкції можуть бути визначені до або після секвенсора в спеціальних полях. Вони викликаються однократно в кожному циклі;
- програмувати структуру секвенсора необхідно на рівні відображення «Секвенсор».

Пари крок – перехід

За замовчуванням FB S7-GRAPH завжди містить *одну* пару крок – перехід, до якої можна додати додаткові пари. Коли вставляються кроки й переходи, їм автоматично привласнюється номери.

Початковий крок – це крок секвенсора, що стає активним без попереднього опитування умов, тобто коли FB S7-GRAPH запускається у перший раз. Початковий крок – це не обов'язково *перший* крок секвенсора.

Коли секвенсор виконується циклічно, початковий крок, як і будь-який крок, стає активним тільки тоді, коли виконуються умови попереднього переходу, наприклад, «Повернення» (*Return*).

Запуск секвенсора з початкового кроку здійснюється при значенні вхідного параметра FB INIT_SQ = 1.

Стрибок – це перехід до іншого кроку в тому чи іншому секвенсорі в межах одного FB. Стрибок завжди встановлюється за переходом і закриває секвенсор або шлях гілки до певної точки. Стрибок і точка його призначення графічно відображаються як стрілка, але саме з'єднання не зображується.

Лінійний секвенсор може бути розширений альтернативним і паралельним розгалуженням.

Альтернативна гілка

Альтернативна гілка починається з *переходу*. Виконується тільки та гілка, перехід до якої включається першим. Альтернативна гілка відповідає, таким чином, логіці АБО, у якій активний тільки один шлях. Кожна приватна альтернативна гілка закінчується переходом (рис. 7.3).

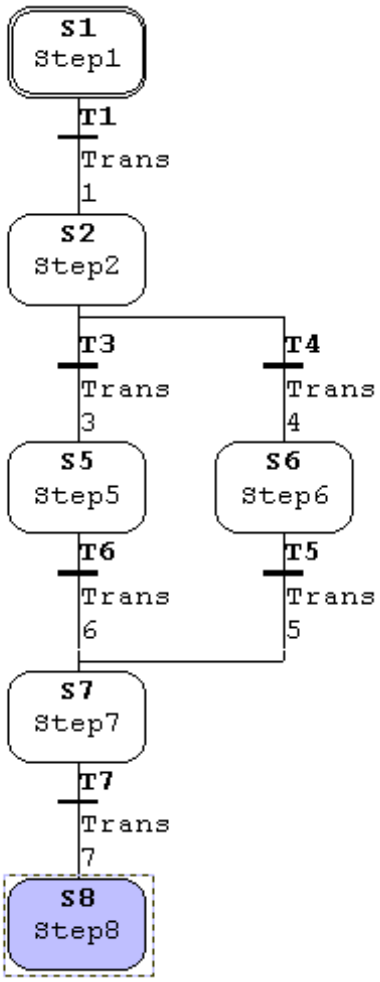


Рисунок 7.3 – Секвенсор з альтернативними гілками

Якщо на початку альтернативних гілок одночасно відкрито кілька переходів, пріоритет має крайня ліва гілка із відкритим переходом.

Паралельне розгалуження

Паралельне розгалуження відповідає обробленню гілок за логікою І. Паралельне розгалуження починається *загальним переходом*, що активує перший крок *у всіх* паралельних гілках.

На рисунку 7.4 загальними переходами є переходи Т3 і Т7.

Кожний шлях у паралельному розгалуженні закінчується кроком, підключеним до *загального* фінального переходу. Фінальний перехід дозволяє наступний крок тільки тоді, коли виконані всі паралельні гілки.

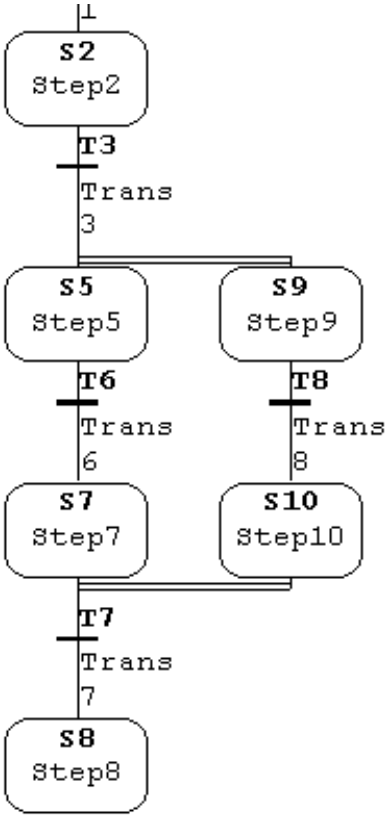


Рисунок 7.4 – Формування паралельних гілок

Постійні умови

Умови, які потрібно виконувати в більш ніж одній точці секвенсора, можуть бути запрограмовані як постійні умови.

Для програмування умов можна використовувати елементи контактної схеми – нормально розімкнуті або нормально замкнуті контакти, компаратори або елементи FBD. У постійній умові можна використовувати до 32 елементів.

Результат обчислення умов зберігається елементом «котушка» в LAD або блоком пам'яті в FBD.

Виклики блоку

Блоки, створені на інших мовах програмування, можуть бути викликані з використанням постійних інструкцій або дій в FB S7-GRAPH. Після того як викликаний блок буде виконаний, виконання FB S7-GRAPH буде продовжено.

У S7-GRAPH можна викликати функції (FC) і функціональні блоки (FB), запрограмовані на LAD, FBD, STL або SCL, а також системні функції (SFC) і системні функціональні блоки (SFB). Функціональним блокам і системним функціональним блокам повинні бути призначені екземплярні блоки даних DB. Імена блоків можна використовувати в абсолютному виді, наприклад, FC1 або символьно, наприклад, Motor1. Під час виклику блоків потрібно забезпечити формальні параметри викликуваного блоку дійсними значеннями.

7.2 Програмування дій і умов

Редагування пари крок – перехід

Після визначення структури секвенсора в FB S7-GRAPH можна почати програмувати окремі кроки й переходи.

Порядок програмування не має великого значення.

На рисунку 7.5 показані поля й області розміщення елементів програми під час програмування кроку й переходу.

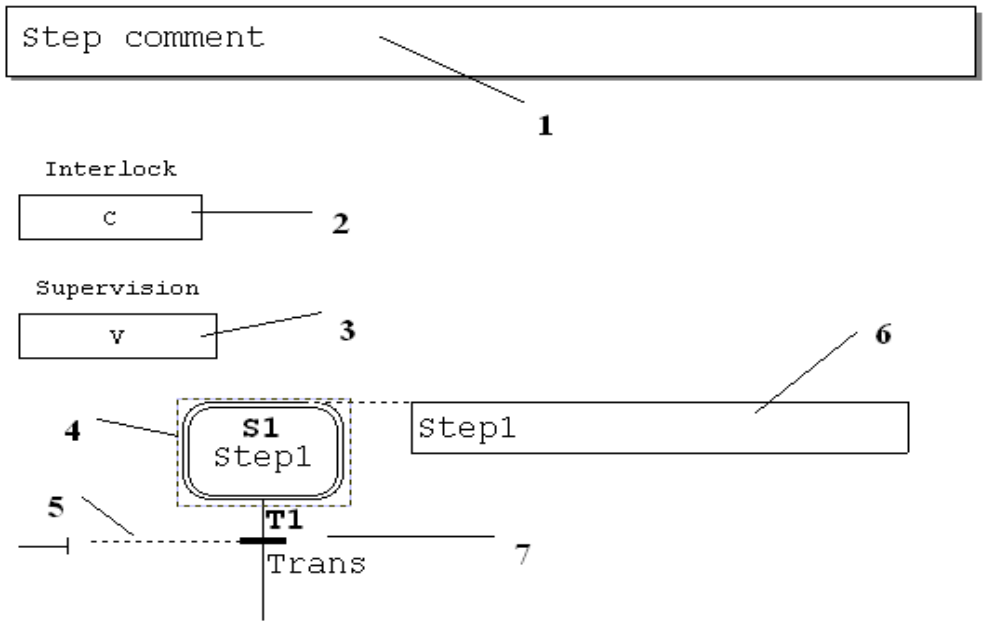


Рисунок 7.5 – Схема розташування областей програмування кроку й переходу

Програмування кроку починається з коментарю в полі 1. Коментар може містити до 2048 символів і не впливає на виконання програми.

У полях 2 і 3 вводяться умови блокування й супервізора, відповідно. У полі 4 перебуває символ кроку, а в полі 7 – символ переходу. Поле 5 призначене для описування умов переходу на мовах LAD, FBD, STL і SCL.

Дії, які повинні бути виконані в кроці, заносяться в таблицю, що розміщається в зоні 6.

Дії керують входами, виходами й меркерами, активують і деактивують кроки секвенсора, викликають блоки.

Отже, дії містять команди керування процесом. Вони виконуються в порядку «зверху донизу». Дії можуть складатися з події, наприклад, «S1 N» на рисунку 7.6 або команди, наприклад, «N M 4.2». У діях вказується адреса (M 4.2) або присвоєння, наприклад A:=B+C.

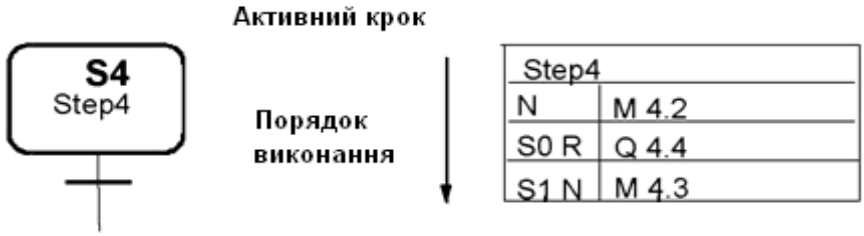


Рисунок 7.6 – Приклад програмування кроку

Дії можна розділити на такі категорії:

- стандартні дії;
- дії, що залежать від подій;
- дії для активації й деактивації кроків;
- лічильники й таймери;
- арифметичні дії.

Кроки, які не містять запрограмованих дій, – це порожні кроки. Порожні кроки обробляються так само, як і активні.

Стандартні дії

Стандартні дії наведені в таблиці 7.1.

Таблиця 7.1

Команда	Ідентифікатор	Розташування	Значення
N[C]	Q, I, M, D	m, n	Адреса однократно одержує значення 1
S[C]	Q, I, M, D	m, n	Адреса встановлюється в 1
R[C]	Q, I, M, D	m, n	Адреса скидається в 0
D[C]	Q, I, M, D	m, n	Затримка включення на *n* секунд
L[C]	Q, I, M, D	m, n	Обмеження тривалості адреси на *n* секунд
CALL[C]	FB, FC, SFB, SFC	Номер блока	Виклик блока

Всі стандартні дії (команди) можуть комбінуватися з умовою блокування (додається символ C). Такі дії виконуються тільки тоді, коли блокування зняте.

У таблиці використовуються такі позначення:

D адреса у форматі: DBi.DBXm.n;
m адреса байта;
n адреса біта;

SFB, FB системний функціональний блок або функціональний блок;

SFC, FC системна функція або функція.

Константа часу

Інструкції D[C] або L[C] вимагають завдання часу. Час програмується як константа із синтаксисом T#<const> і може бути заданий в необхідній комбінації:

<const> = nD (n діб), nH (n годин), nM (n хвилин), nS (n секунд), nMS (n мілісекунд),

де n – ціле число.

Приклад:

T#2D3H – константа часу: 2 доби й 3 години.

Дії, які залежать від подій

Дії можуть бути логічно скомбіновані з подіями.

Подія – це зміна стану кроку, супервізора, блокування, квітування, повідомлення або установлення реєстрації.

S1: Крок активується.

S0: Крок деактивується.

V1: З'явилася помилка супервізора (неполадка).

V0: Припинилася помилка супервізора (немає неполадок).

L0: Умова блокування включена.

L1: Умова блокування відключена, наприклад, неполадка.

C: Умови блокування задоволені.

A1: Повідомлення квітовано.

R1: Установлена реєстрація (позитивний фронт на вході REG_EF або REG_S).

Якщо дія логічно скомбінована з подією, сигнал стану визначається за *виявленням фронту*. Це означає, що інструкції можуть виконуватися тільки в тім циклі, у якому відбувається подія:

У разі настання подій S0, V0, L0, L1 можливе виконання таких команд:

N – адреса однократно одержує значення 1;

S – адреса однократно встановлюється в 1;

R – адреса однократно скидається в 0;

CALL – однократно викликається блок.

Кроки можна активувати й деактивувати, використовуючи команди ON і OFF. Момент активації й деактивації кроку визначають події.

Команди активації й дезактивації кроків можна комбінувати із блокуванням. У цьому випадку дії виконуються тільки при виконаних умовах блокування.

Принципи комбінування представлені в таблиці 7.2.

Таблиця 7.2

Подія	Команда	Ідентифікатор адреси	Значення
S1, V1, A1, R1	ON[C], OFF[C]	S	Активація (ON) або деактивація (OFF) кроку залежно від події
S1, V1	OFF[C]	S_ALL	Деактивація всіх кроків залежно від події
S0, V0, L0, L1	ON, OFF	S	Активація (ON) або деактивація (OFF) кроку залежно від події
L1	OFF	S_ALL	Деактивація всіх кроків залежно від події

Лічильники й таймери в діях

Лічильники в діях поводяться, як і в інших мовах програмування S7 – вони не переповняються зверху й знизу. У разі, якщо значення лічильника дорівнює 0, біт стану лічильника також дорівнює 0, а в інших випадках дорівнює 1.

Для всіх типів подій (S, R, V, L, A) можна застосовувати чотири інструкції:

CS[C] – установлення лічильника (завантаження початкового значення);

CU[C] – рахунок вверх;

CD[C] – рахунок вниз;

CR[C] – скидання.

На відміну від лічильників, таймери для всіх типів подій (S, R, V, L, A) можуть виконувати три інструкції:

TL[C] – розширений імпульс, старт і рестарт у момент настання події при дотриманні умови C;

TD[C] – запуск під час виникнення події й дотриманні умови C;

TR[C] – скидання таймера при настанні події.

Арифметика в діях

У діях можна передбачити також інструкції із простими арифметичними виразами – присвоєння у вигляді A:=B, A:=func(B) і A:=B<operator>C.

Дії з арифметичними вираженнями вимагають інструкції N.

При прямому присвоєнні із синтаксисом A:=B використовуються такі типи даних:

8 біт – BYTE, CHAR;

16 біт – ORD, INT, DATE, S5TIME;

32 біта – WORD, DINT, REAL, TIME, TIME_OF_DAY.

Присвоєння з убудованою функцією записується у вигляді A: = func(B). Під убудованими функціями маються на увазі функції перетворення й складні математичні функції. Призначувана адреса A визначає тип виразу.

Убудовані функції *перетворення* наведені в таблиці 7.3.

Таблиця 7.3

Функція перетворення	Коментар
A:=BCD_TO_NUM(B)	BCD в INT або в DINT (команди STL – BTI, BTD)
A:=NUM_TO_BCD(B)	INT або DINT в BCD (команди STL – ITB, DTB)
A:=INT_TO_DINT(B)	INT в DINT (команда STL – ITD)
A:=DINT_TO_REAL(B)	DINT в REAL (команда STL -DTR)
A:=ROUND(B)	REAL в DINT (команда STL – RND)
A:=TRUNC(B)	REAL в DINT, відкидання залишку (команда STL – TRUNC)

Умови

Умови – це двійкові стани процесу, які з'єднуються з елементами LAD або блоками FBD. Умови – це події й стани, наприклад, вхід I 2.0 установлений. Умови використовуються у переході для дозволу наступного кроку, під час виклику блоку, введення блокувань і в супервізорі.

Перехід передає керування наступному кроку секвенсора, якщо виконується логічна умова переходу, тобто коли результат виконання сегмента дорівнює 1. Крок, який прямує за переходом, стає активним.

Убудовані *математичні* функції наведені в таблиці 7.4.

Блокування – це програмувальна умова для заборони виконання окремих дій у кроках, підданих блокуванню. Якщо логічні умови блокування виконуються, дії, скомбіновані із блокуванням, виконуються. Якщо логічні умови блокування не виконуються, це вважається порушенням. Запрограмоване блокування позначається буквою С поруч із кроком.

Таблиця 7.4

Функція	Коментар
A:=NEGR(B)	REAL, заперечення
A:=ABS(B)	REAL, абсолютне значення
A:=SQR(B)	REAL, квадрат
A:=SQRT(B)	REAL, квадратний корінь
A:=LN(B)	REAL, логарифм по основі *e*
A:=EXP(B)	REAL, експонента
A:=SIN(B)	REAL, синус
A:=ASIN(B)	REAL, арксинус
A:=COS(B)	REAL, косинус
A:=ACOS(B)	REAL, арккосинус
A:=TAN(B)	REAL, тангенс
A:=ATAN(B)	REAL, арктангенс
A:=ANEG(B)	Зміна знака

Супервізор – це програмувальна умова для контролю кроку, що впливає на спосіб, яким секвенсор передає керування від одного кроку до наступного. Запрограмований супервізор позначається на всіх рівнях подання буквою V ліворуч від кроку.

Якщо логічна умова супервізора виконується, це означає неполадку й сигналізує про подію V1. Секвенсор не передає керування наступному кроку, активним залишається поточний крок. Якщо логічна умова супервізора не виконується, то неполадок немає й секвенсор передає керування наступному кроку.

7.3 Установлення параметрів

Ім'я й номер кроку призначаються системою автоматично від step1 до step999, однак його можна змінити. Ім'я переходу призначається системою також автоматично від Trans1 до Trans999, його також можна змінювати.

Ім'я кроку й переходу може містити максимум 24 символів (букв і цифр). Перший символ повинен бути буквою.

Призначення параметрів FB S7-GRAPH

Множина параметрів блоку S7-GRAPH залежить від передбаченого використання секвенсора й доступної пам'яті CPU.

У вікні опису змінних існує чотири стандартних набори параметрів:

- мінімальний набір з 3 параметрами (Minimum);
- стандартний набір з 21 параметром (Standard);
- максимальний набір з 31 параметром (Maximum);
- користувальницький набір з 53 параметрами (User-specific).

Зі стандартних наборів можна видалити не використовувані параметри. Та або інша множина параметрів вибирається відповідно до розв'язуваного завдання. Існують такі рекомендації:

- множина параметрів Minimum використовується тоді, коли секвенсор працює тільки в автоматичному режимі й не потрібно додаткових функцій керування й моніторингу;
- множина параметрів Standard використовується тоді, коли секвенсор працює в різних режимах і потрібен зворотний зв'язок від процесу;
- множина параметрів Maximum використовується тоді, коли секвенсор працює в різних режимах. При цьому потрібен зворотний зв'язок від процесу, можливість квітування повідомлень і додаткові функції моніторингу для обслуговування системи;
- користувальницький набір параметрів User-specific надає ті ж можливості, що й набір Maximum.

На рисунку 7.7 показаний функціональний блок S7-GRAPH зі стандартним набором вхідних і вихідних параметрів.

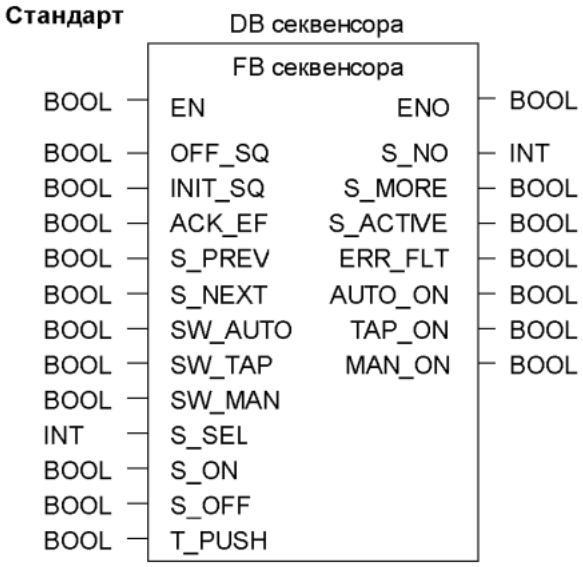


Рисунок 7.7 – Відображення блоку з набором параметрів Standard

Вхідні параметри FB S7-GRAPH

Перелік вхідних параметрів для наборів Minimum, Standard і Maximum наведений у таблиці 7.5, а вихідних – у таблиці 7.6.

Функціональний блок S7-GRAPH реагує на позитивний фронт вхідного параметра (за винятком EN).

Таблиця 7.5 – Вхідні параметри блока S7-GRAPH

Параметр	Тип даних	Опис	Minim.	Stand.	Maxim.
1	2	3	4	5	6
EN	BOOL	Вхід дозволу	•	•	•
OFF_SQ	BOOL	Деактивація секвенсора		•	•
INIT_SQ	BOOL	Скидання секвенсора	•	•	•
ACK_EF	BOOL	Квітування помилок, примусове перемикання до наступного кроку		•	•
HALT_SQ	BOOL	Останов секвенсора			•
HALT_TM	BOOL	Останов всіх таймерів			•
ZERO_OP	BOOL	Скидання в 0 всіх адрес операторів і команди CALL			•
EN_IL	BOOL	Деактивація блокувань			•
EN_SV	BOOL	Деактивація умов супервізора			•
S_PREV	BOOL	В автоматичному режимі крок назад. У ручному режимі – показати номер попереднього кроку на S_NO		•	•

Продовження таблиці 7.5

1	2	3	4	5	6
S_NEXT	BOOL	В автоматичному режимі крок уперед. У ручному режимі – показати номер наступного кроку на S_NO		•	•
SW_AUTO	BOOL	Включити автоматичний режим		•	•
SW_TAP	BOOL	Включити режим з підштовхуванням		•	•
SW_MA	BOOL	Включити ручний режим		•	•
S_SEL	INT	Уведення номера кроку		•	•
S_ON	BOOL	Відобразити крок у ручному режимі		•	•
S_OFF	BOOL	Виключити відображення кроку		•	•
T_PUSH	BOOL	Передача керування у покроковому режимі		•	•

Таблиця 7.6 – Вихідні параметри блока S 7-GRAPH

Параметр	Тип дани	Опис	Minim.	Stand.	Maxim.
ENO	BOOL	Дозвіл виходу	•	•	•
S_NO	INT	Відображення номера кроку		•	•
S_MORE	BOOL	Вибір іншого кроку		•	•
S_ACTIVE	BOOL	Відображення активного кроку		•	•
ERR_FLT	BOOL	Групова неполадка		•	•
SQ_HALTED	BOOL	Секвенсор зупинений			•
TM_HALTED	BOOL	Таймери зупинені			•
OP_ZEROED	BOOL	Адреси скинуті			•
IL_ENABLED	BOOL	Блокування дозволене			•
SV_ENABLED	BOOL	Супервізор дозволений			•
AUTO_ON	BOOL	Індикація автоматичного режиму		•	•
TAP_ON	BOOL	Індикація режиму з підштовхуванням		•	•
MAN_ON	BOOL	Індикація ручного режиму		•	•

7.4 Створення структури й установлення режимів системи керування

Визначення структури програми й вбудовування секвенсора

Для кожного секвенсора S7-GRAPH створюється FB з екземплярним DB. Оскільки разом із програмами, створеними в S7-GRAPH, зазвичай потрібні інші програми, найкращий спосіб – це виклик усіх FB, створених в S7-GRAPH в одному блоці, як показано на рисунку 7.8.

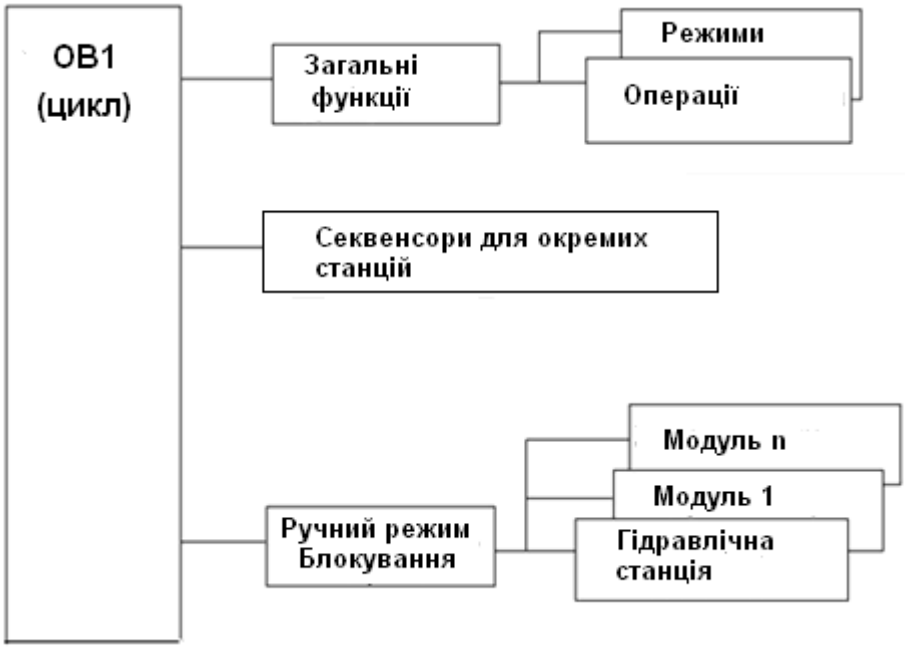


Рисунок 7.8 – Структура програми з використанням секвенсорів

Різні функції на окремих рівнях виконуються циклічно у тому порядку, у якому вони викликаються.

Рекомендується такий порядок виклику:

- секвенсори як загальні функції більш високого рівня;
- секвенсори для окремих станцій, які викликаються в FB «Sequencers»;
- за секвенсорами викликаються розділи програми для ручного режиму, блокувань і постійних функцій моніторингу для модулів.

Режими роботи

Залежно від ситуації, користувачеві потрібні різні режими роботи системи й установлення.

У незв'язаних виробничих осередках, наприклад, при складанні корпусу, можливі такі режими:

- автоматичний (SW_AUTO). В автоматичному режимі керування передається наступному кроку, якщо виконується умова переходу;
- з підштовхуванням (SW_TAP) – це різновид автоматичного режиму із зупинкою після *кожного* обробленого кроку (напівавтомат). Секвенсор

передає керування, коли виконується умова переходу й виникає позитивний фронт (перехід з 0 в 1) у параметрі T_PUSH;

- автомат або перемикання до наступного SW_TOP – це автоматичний режим з додатковими умовами дозволу кроку початкового запуску проекту або покрокового тестування керуючої системи. Секвенсор виконується завжди, коли відкритий перехід або коли виникає позитивний фронт параметра T_PUSH;
- ручний (SW_MAN) – це пряме керування модулями й функціями. Ручний режим вибирається, наприклад, для перевірки секвенсора. Ручний режим характеризується тим, що кроки можна вибирати й скасовувати вручну.

Якщо є панель вибору режиму й перемикач на керуючій панелі забезпечує установлення окремих сигналів автоматичного режиму, режиму з підштовхуванням і ручним режимом, сигнали режиму для секвенсора можна одержати за допомогою простої логіки.

Контрольні питання

1. Що називається секвенсором?
2. Що являється основою для створення програми секвенсора?
3. Які блоки потрібні для застосування секвенсора у S7-програмі?
4. Які правила необхідно виконати при створенні секвенсора?
5. Що являє собою початковий крок секвенсора?
6. Які правила застосовуються для побудови альтернативних гілок?
7. Які правила застосовуються для побудови паралельних гілок?
8. Що розуміється під постійними умовами?
9. Як можна викликати функціональний блок FB S7-GRAPH?
10. Що програмується в кроці секвенсора?
11. Що програмується в переході секвенсора?
12. Які стандартні дії передбачені при програмуванні секвенсора?
13. Які події аналізуються секвенсором?
14. Яку арифметику можна застосовувати в діях при програмуванні секвенсора?
15. Що розуміється під терміном «блокування» у секвенсорі?
16. Якими наборами параметрів можна скористатись при програмуванні блока S7-GRAPH?
17. Який порядок передбачено під час виклику секвенсорів у програмі?
18. Які режими роботи можливі при реалізації секвенсора?

8 ПРОГРАМУВАННЯ МОВОЮ S7-HIGRAPH

8.1 Принцип програмування мовою S7-Higraph

Графічна нотація широко використовується для описування поведінки автоматів, що здійснюють логічне керування встаткуванням. Алгоритми керування часто представляються блок-схемами, графами станів, перемикальними схемами, мережами Петрі й т. п. Мова програмування S7-Higraph дозволяє розширити функціональну область середовища програмування STEP 7 шляхом застосування графічного методу програмування на основі використання графів станів.

Для застосування цієї мови об'єкт автоматизації розділяється на функціональні матеріальні одиниці. Поведінка кожної функціональної одиниці описується графом станів. Для організації взаємодії графів станів створюється координуючий граф станів (граф-диспетчер). Усі процедури процесу створення графів станів здійснюються в редакторі S7-Higraph. Виконаний етап роботи з кожного графа зберігається в контейнері «Source files» (вихідні файли) S7-програми без перевірки синтаксису.

Higraph-програма структурована в такий спосіб:

- у графах станів визначаються дії, які можуть бути зроблені під час входу в стан, під час стану та під час виходу зі стану;
- керування переходом (транзакцією) з одного стану в інший здійснюється розв'язною умовою із заданим рівнем пріоритету;
- дії в станах і умови в транзакціях описуються мовою програмування STEP 7 STL;
- для забезпечення взаємодії графів станів з координуючим графом станів програмуються повідомлення – вихідні в станах і вхідні в транзакціях;
- графи станів разом з координуючим графом вставляються в груповий граф. У груповому графові призначаються поточні параметри всім змінним і повідомленням;
- груповий граф компілюється зі створенням функції керування (FC) і блоку даних поточних значень параметрів (DB), які розміщаються в контейнері Blocks S7-програми.

Увесь алгоритм керування може бути задокументований у графічній і текстовій формі.

Функція Higraph FC повинна викликатися із циклічно працюючого блоку OB1. Структура готової програми показана на рисунку 8.1.

Рисунок 8.1 – Структура Higraph-програми

Редактор Higraph забезпечує такі функції програмування:

- програмування станів і транзакцій мовою STL;
- програмування викликів FC-функції Higraph і логічних блоків STEP 7 (FC, SFC, FB, SFB), створених із застосуванням мов STL, LAD, FBD, а також SCL-команд;
- програмування часу очікування завершення дії й контрольного часу перебування в стані;
- тестування функціональних блоків з визначенням активного стану, попереднього стану й останньої транзакції, а також з виставою інформації щодо команд у станах і транзакціях;
- виявлення помилок процесу, блокувань за часом і аварійних ситуацій з виводом інформації на пристрій зв'язку з оператором.

Графи станів і групові графи зберігаються в SIMATIC Manager у контейнері «Source files», а скомпільовані групові графи у вигляді функції FC, блоку даних DB і додаткових блоків – у папці Blocks.

Редактор Higraph забезпечує такі опції програмування:

- вставляння будь-якої кількості графів стану в груповий граф;
- одночасне редагування декількох групових графів;
- програмування умов у транзакціях;
- програмування дій у станах, причому дії характеризуються подіями на вході (E), діями на виході (X) і циклічними діями (C и C-);
- використання для програмування всього спектра STL команд. Перелік STL-команд наведений у додатку;
- введення контрольного часу й часу очікування у формі змінних або констант;
- перемикання між символьною й абсолютною виставою адрес.

Послідовність створення програми містить у собі такі кроки:

1. Створення проекту програми в Simatic Manager;
2. Вставлення графа станів у програму;
3. Визначення сигналів, необхідних для керування;
4. Програмування станів;
5. Програмування транзакцій;
6. Програмування постійних інструкцій;
7. Програмування операційних режимів (автоматичний, ручний);
8. Створення координуючого графа;
9. Створення групового графа;
10. Установлення послідовності виконання;
11. Призначення фактичних параметрів;
12. Програмування повідомлень;
13. Компіляція «первинників» («исходников» російською мовою) і створення блоків програми; вихідних кодів
14. Завантаження програми в контролер;
15. Налагодження програми в інтерактивному режимі.

8.2 Аналіз об‘єкта та визначення задач керування

Розглянемо приклад, наведений у [8]. У прикладі створюється проста програма для свердлильного верстата, показаного на рисунку 8.2.

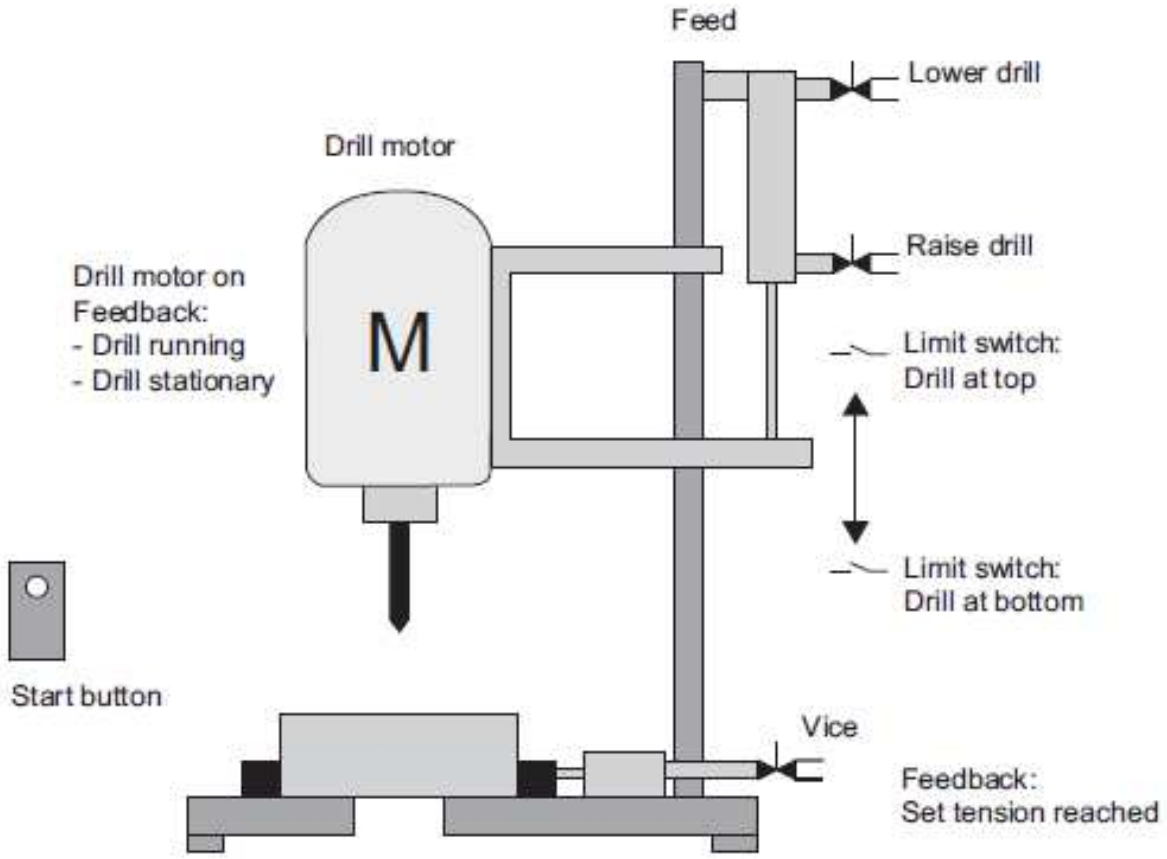


Рисунок 8.2 – Функціональні елементи свердлильного верстата

Свердлильний верстат містить гідравлічний затискач заготовки (Vice), керований за допомогою золотника, електропривод обертання свердла (Drill motor) і гідравлічний привід подачі свердлильної головки (Feed) із золотниками керування підйомом (Raise drill) і опусканням (Lower drill). Контроль процесу здійснюється кінцевими вимикачами переміщення свердлильної головки (Limit switch), датчиком швидкості обертання вала двигуна (Motor Running) і тензодатчиком (Tension Reached), що сигналізують про досягнення необхідної сили затискача заготовки. Пускання процесу свердління здійснюється кнопкою Start button.

Вихідний стан верстата визначений у такий спосіб:

- мотор привода свердла в стані останову;
- свердлильна головка в крайньому верхньому положенні;
- заготовка встановлена в затискне пристосування, не затиснута.

На рисунку 8.3 показана функціональна діаграма процесу свердління, що складається з 8 станів.

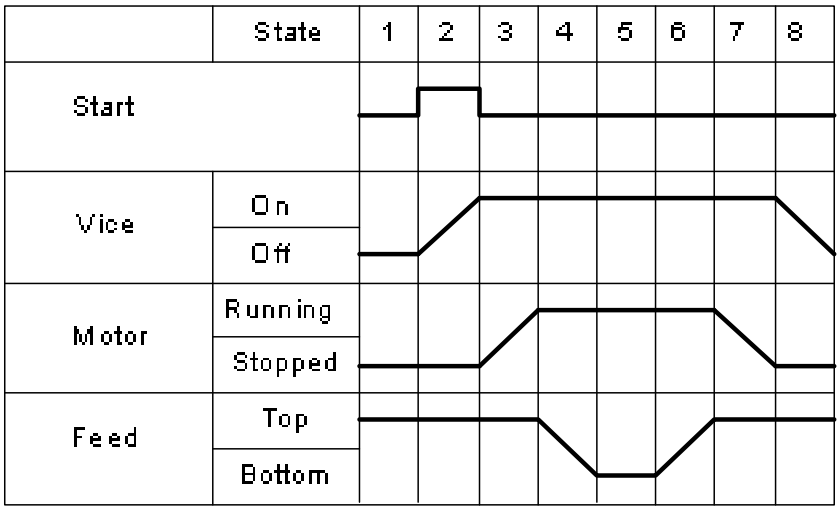


Рисунок 8.3 – Функціональна діаграма процесу свердління

Процес свердління в автоматичному режимі починається із включення верстата пусковою кнопкою Start button і складається з таких операцій:

- затискання заготовки, доки не досягнутий тиск фіксації;
- пускання двигуна обертання свердла;
- подача свердла вниз, поки не досягнута крайня нижня точка;
- витримка в нижньому положенні;
- подача свердла нагору, поки не досягнута крайня верхня точка;
- останов двигуна обертання свердла;
- розтискання заготовки;
- видалення заготовки оператором.

Виконавчі пристрої свердлильного верстата керуються через виходи модуля цифрового виведення з адресами Q 0.0 - Q 0.7. Вхідні сигнали подаються на модуль уведення з адресами входів I 0.0 - I 0.7.

Призначення вхідних і вихідних сигналів наведено в таблиці 8.1.

Таблиця 8.1 – Призначення вхідних і вихідних сигналів

Адреса		Опис
Символьна	Абсол.	
Drill_Motor_Running	I 0.0	Свердло обертається із заданою швидкістю
Drill_Motor_Stopped	I 0.1	Двигун привода свердла зупинений
Drill_at_Bottom	I 0.2	Свердлильна головка в нижньому положенні
Drill_at_Top	I 0.3	Свердлильна головка у верхньому положенні
Tension_Reached	I 0.4	Заготовка затиснута (тиск досягнутий)
Start_Button	I 0.7	Сигнал пускової кнопки
Drill_Motor_On	Q 0.0	Вмикання двигуна привода свердла
Lower_Drill	Q 0.1	Вмикання подачі свердла вниз
Raise_Drill	Q 0.2	Вмикання подачі свердла нагору
Clamp_Workpiece	Q 0.3	Затискання заготовки

З опису завдання випливає, що процес свердління заготовки здійс-

нюється за допомогою трьох функціональних одиниць: пристрою затискача деталі (Vice), привода обертання свердла (Motor) і пристрою подачі свердлильної головки (Feed). Для кожної функціональної одиниці потрібний один граф станів. Для координації роботи цих функціональних одиниць потрібно ще один граф станів – свердління (Drilling).

Порядок створення графа станів можна розглянути на прикладі привода подачі «Feed».

На рисунку 8.4 показаний функціональний блок гідроциліндра подачі з діаграмами сигналів керування й зворотного зв'язку. Він містить два електромагнітні клапани (Up і Down) і два кінцеві вимикачі (Top і Bottom).

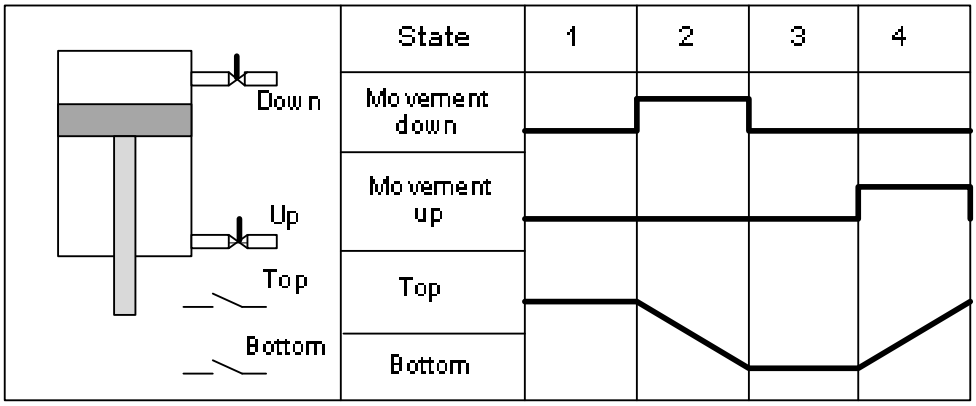


Рисунок 8.4 – Функціональний блок гідроциліндра подачі свердла

Як видно з діаграми, процес керування гідроциліндром складається із чотирьох станів – 1, 2, 3, 4. На діаграмі не показаний стан 0, який повинен бути в кожному графові. Стан 0 призначений для перевірки поточного положення функціонального блоку і його переводу (за необхідності) у стан готовності, коли свердлильна головка перебуває у верхньому положенні.

Виведення свердла у верхнє положення повинно проводитися по закінченні кожного циклу, тобто в стані 4. Якщо з якоїсь причини в момент включення верстата свердло перебуває в іншому положенні, то перехід зі стану 0 повинен бути зроблений саме в стан 4, щоб забезпечити переміщення свердла нагору. З обліком цього складена послідовність зміни станів графа «Feed» (Подача), яка наведена на рисунку 8.5.

Передбачається, що клапани з електромагнітним керуванням повинні використовуватися тільки для фази руху й гідроциліндр залишається у верхньому кінцевому положенні під час зняття сигналів керування.

8.3 Послідовність створення графа станів у Higraph

Розглянемо деталі створення графа станів у кожному кроці.

Крок 1. Створення проекту програми в Simatic Manager.

Для створення нової програми необхідно відкрити Simatic Manager. Далі у вікні «New Project», що відкрилося, ввести ім'я нового проекту й натиснути ОК.

Рисунок 8.5 – Послідовність виконання графа станів «Feed»

У лівій панелі центрального вікна Simatic Manager установити курсор на імені проекту й правою кнопкою миші викликати контекстне меню. У списку, що відкрився, вибрати Insert New Object ► SIMATIC 300 Station. Далі у вікні HW-Config під станцію SIMATIC 300 потрібно вставити рейку RACK 300 (Rail), блок живлення й процесорний модуль, наприклад, CPU315-2 PN/DP. Процес компонування елементів проекту закінчується вставкою S7 Program, яка буде відображена в правій панелі.

Крок 2. Вставлення графа станів у програму.

Для вставлененя графа станів потрібно подвійним клацанням лівої кнопки миші розкрити структуру S7 Program, вибрати контейнер «Sources» і правою кнопкою викликати контекстне меню, у якому вибрати Insert New Object ► State graph. При цьому в правій панелі з'явиться об'єкт «State graph(1)», якому тут же слід привласнити ім'я, наприклад, «Motor». Після подвійного клацання по піктограмі «Motor» тільки що створений початковий граф станів буде відображений в основному вікні редактора Higraph (рис. 8.6). Під вікном редактора показане відкрите вікно інструкцій (Instructions), у якому можна ввести команди для програмування станів і транзакцій. За допомогою кнопок унизу вікна можна також відкрити вікна оголошення змінних (Variables) і повідомлень (Application messages і Document messages).

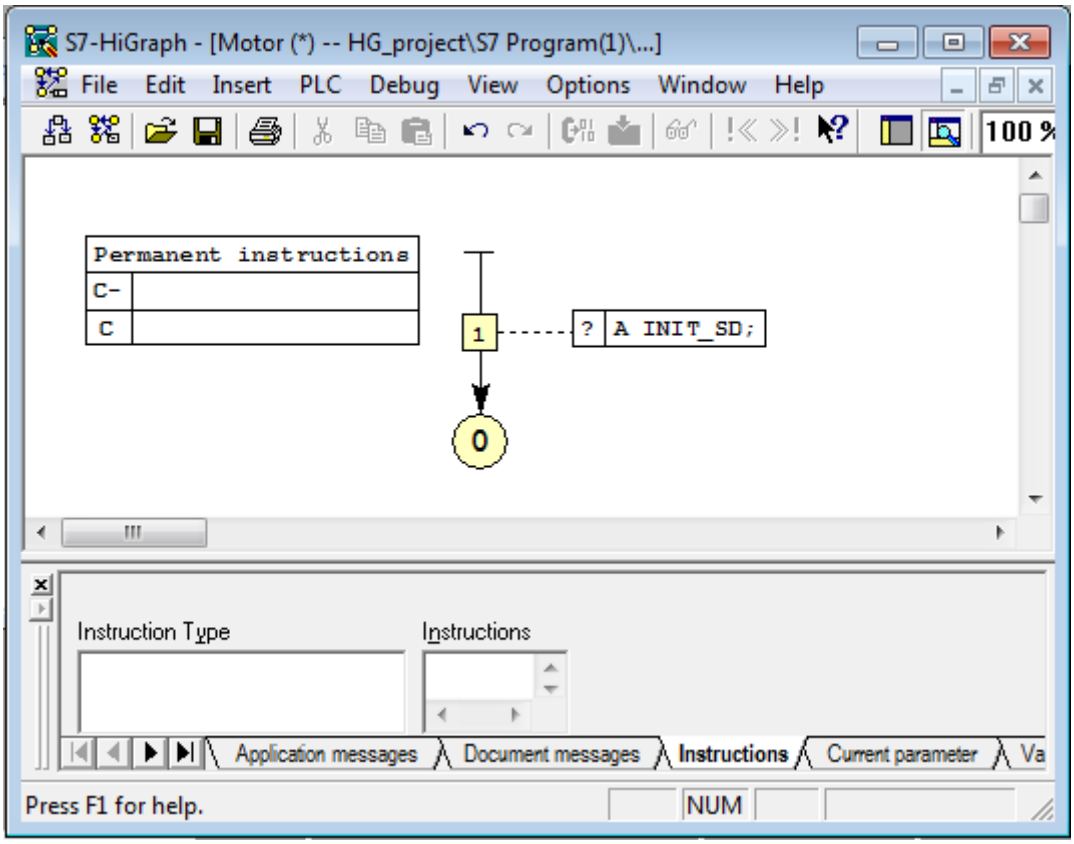


Рисунок 8.6 –Вікно редактора з початковим графом станів

Крок 3. Визначення сигналів, необхідних для керування.

Для відображення розділу змінних виберемо в меню View опцію Details. При цьому відкриється ліва панель Environment (навколишнє середовище), у якій відображаються елементи програми, згруповані в розділи. Для перегляду змінних слід розкрити розділ Interface. У нижній частині вікна редактора перебуває вікно редагування змінних, яке відкривається кнопкою Variables. Це вікно являє собою таблицю із чотирма колонками – ім'я, тип даних, коментар і початкове значення. При цьому початкове значення не редагується, а встановлюється системою.

Вид вікна редагування після зроблених у цьому кроці настроювань показаний на рисунку 8.7.

Розділ змінних містить такі значення:

- *вхідні змінні IN*. Під час створення графа редактор автоматично вводить у список вхідних змінних три змінні – Automaticmode, Manualmode й Usrmsgquit, видаляти які не можна. При цьому вхідна змінна Automaticmode при значенні «1» дозволяє оброблення транзакцій тільки затрибутом «Auto» і забороняє оброблення транзакцій з атрибутом «Manual». Аналогічно, змінна Manualmode при значенні «1» дозволяє оброблення транзакцій тільки з атрибутом «Manual». Змінна Usrmsgquit служить для підтвердження помилки або повідомлення;
- *вихідні змінні OUT*. У список вихідних змінних уводяться імена вихідних параметрів графа станів;

- *вхідні / вихідні змінні IN_OUT*. У цей список включаються змінні для обміну повідомленнями між графами станів;
- *змінні STAT*. Для спрощення процесу програмування редактор Higraph автоматично встановлює в цьому списку 15 необхідних для роботи змінних, у тому числі INIT_SD, яка забезпечує запуск програми. Значення змінних STAT представлені в таблиці 8.2.

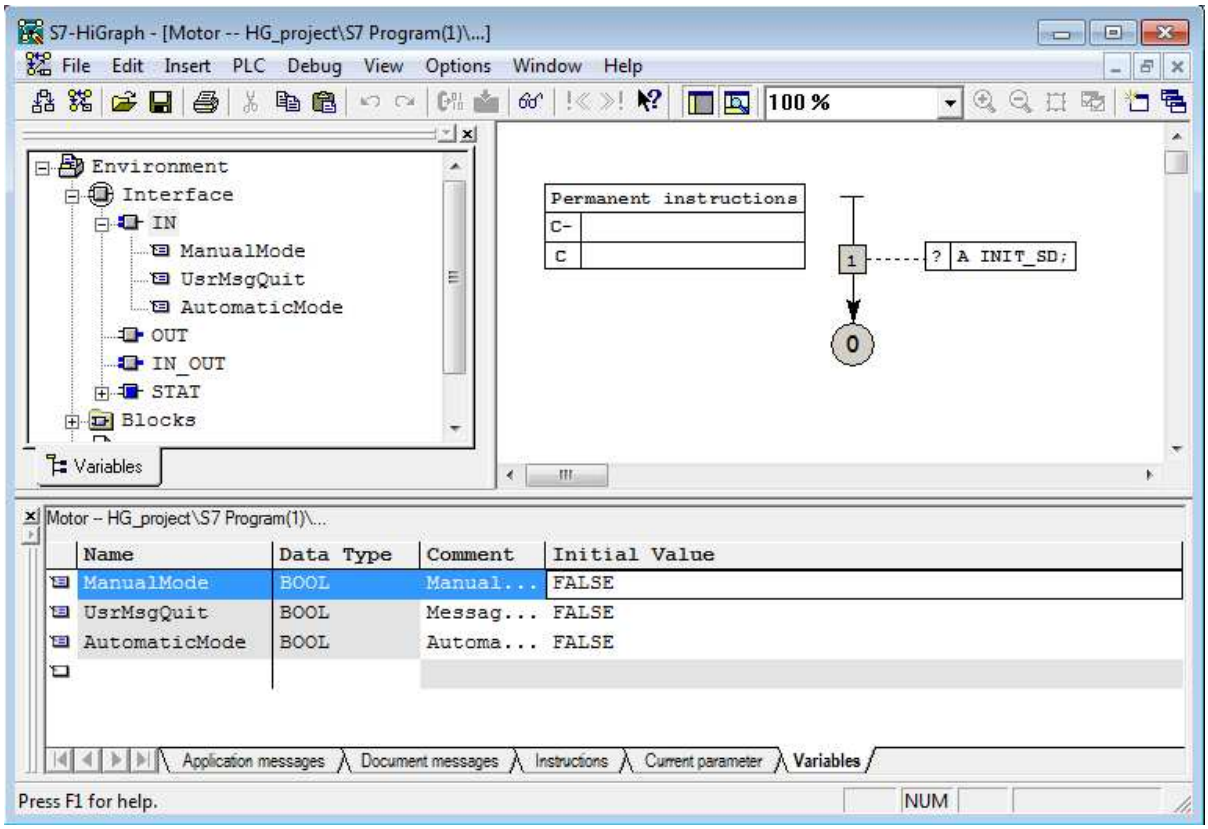


Рисунок 8.7 – Вид вікна редагування на кроці 3

Під час оголошення змінних дозволяється таке.

В імені змінної дозволяються текстові символи й символи підкреслення, причому символ підкреслення може стояти на початку імені, але не повинен стояти наприкінці імені.

При оголошенні типу даних редактор пред'являє на вибір BOOL, INT, WORD, CHAR, String, TIME і т. д.

Для повідомлень, які оголошуються в розділі IN_OUT, у вікні оголошень відкривається колонка Message Type (тип повідомлення).

У розділі коментаря допускається будь-яка вистава тексту.

У вікні оголошення змінних не передбачене введення адреси. Адреса змінної, її формальний параметр, буде визначений при заповненні таблиці ідентифікаторів Symbols, яка перебуває в папці S7 Program Simatic Manager.

Відразу після того, як новий граф стану створений, змінні Currentstate, Previousstate і Statechange деактивовані.

Таблиця 8.2 – Значення змінних розділу STAT

Ім'я змінної	Значення	Тип даних	Визначається	
			Користувачем	Системою
WT_Expired	Витікання часу очікування	BOOL		Так
WT_Valid	Час очікування активний	BOOL		Так
WT_Stop	Зупинка часу очікування	BOOL	Так	
WT_Currvalue	Збереження часу очікування	DWORD		Так
Usrmsgsend	Повідомлення стану активно	BOOL		Так
Usrmsgstat	Для внутрішнього використання	WORD		
ST_Expiredprev	Витікання контрольного часу попереднього стану	BOOL		Так
ST_Expired	Витікання контрольного часу	BOOL		Так
ST_Valid	Контрольний час активний	BOOL		Так
ST_Stop	Зупинка контрольного часу	BOOL	Так	
ST_Currvalue	Збереження контрольного часу	DWORD		Так
INIT_SD	**Параметр запуску**	**BOOL**	**Так**	
Currentstate	Номер поточного стану	WORD		Так
Previousstate	Номер попереднього стану	WORD		Так
Statechange	Зміна стану	BOOL		Так

Для активізації цих змінних необхідно:

- виділити змінну у вікні оголошення змінних і в контекстному меню вибрати команду Object Properties**;**
- у діалоговому вікні перейти на вкладку «Attributes» і призначити значення «true» на атрибут «S7_active», як показано на рисунку 8.8.

Крок 4. Програмування станів.

Програмування станів містить у собі:

- присвоєння імені стану (не обов'язково);
- уведення команд;
- визначення часу очікування й контрольного часу (не обов'язково);
- вставлення наступного стану.

Вставлення стану здійснюється командою контекстного меню Insert State. Стани нумеруються в порядку, у якому вони вводяться. Присвоєння імені стану проводиться у вікні Object Propertis, виклик якого можливий після виділення стану правою кнопкою миші. У цьому ж вікні можна змінити номер стану.

Для введення команд потрібно відкрити вікно Instructions, двічі клацнувши по стану. При цьому у вікні Instructions (рис. 8.9) відобразиться список типів інструкцій. Типи інструкцій, які вводяться в стан графа, представлено в таблиці 8.3.

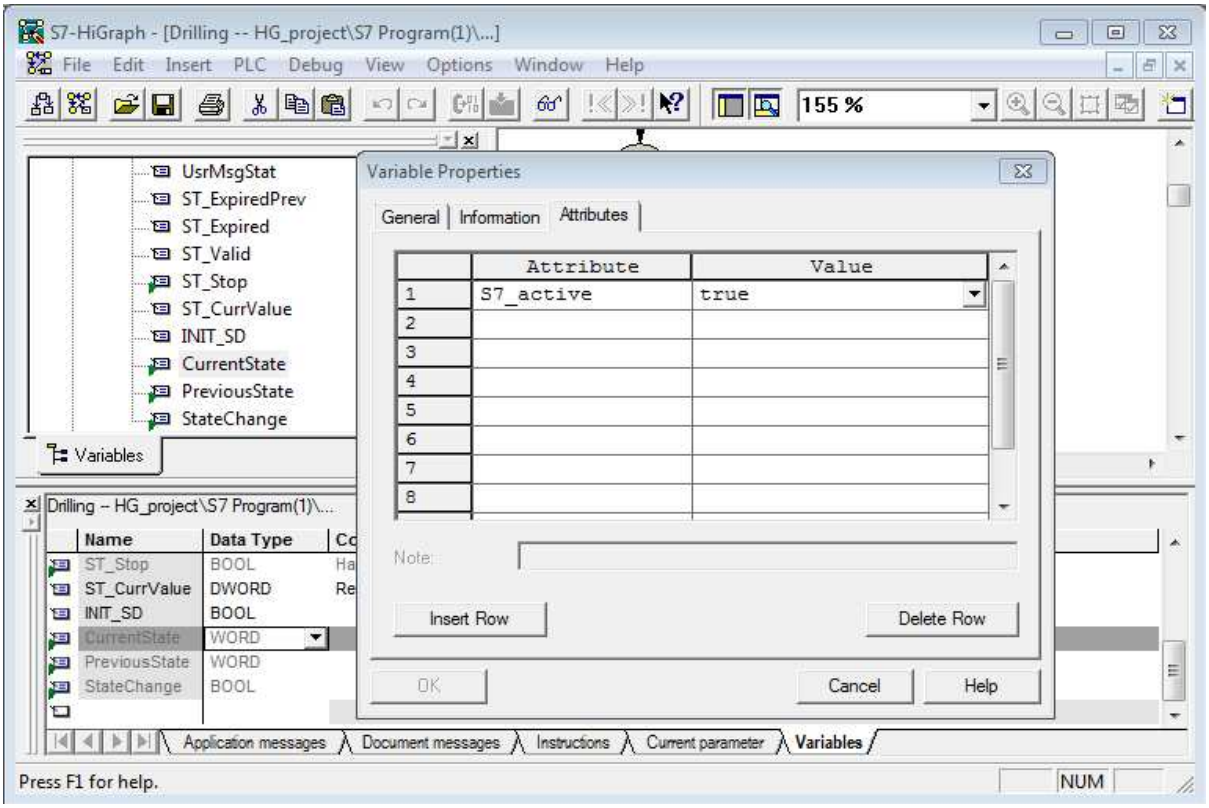


Рисунок 8.8 – Приклад активування змінної Currentstate

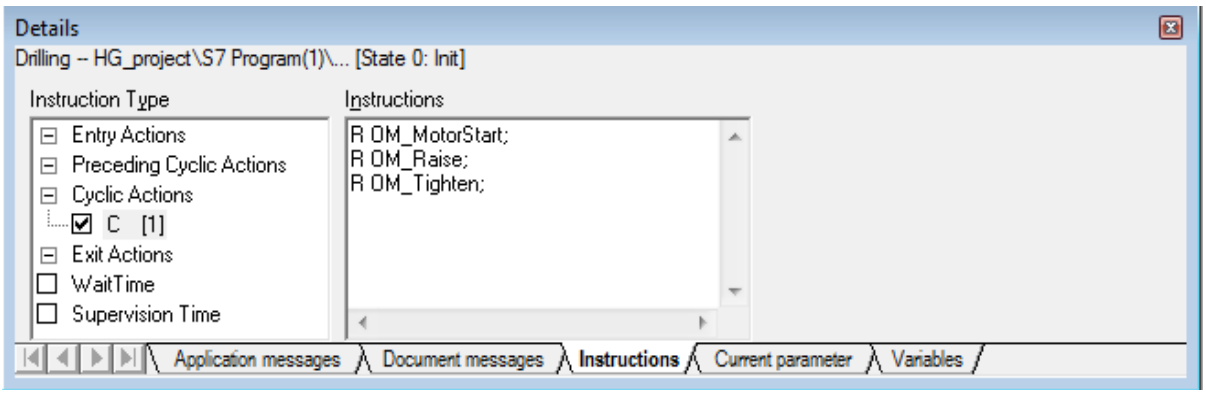


Рисунок 8.9 – Вигляд вікна введення команд Instructions під час програмуванні стану 0

Уведення інструкції здійснюється в правому полі, яке стає активним після виділення типу інструкції й вибору в контекстному меню єдиної команди Insert.

Під час уведення STL-команди можна використовувати символьні й формальні параметри. Введення повинно закінчуватися крапкою з комою. Результат програмування стану 0 показаний на рис. 8.10.

Під час програмування стану можна визначити, чи повинен контролер залишатися в стані якийсь час до того, як буде перевірена наступна транзакція. Установлення часу очікування здійснюється при виборі команди «Wait Time». При цьому в правому вікні за замовчуванням установлюється час T#500 ms, який може бути змінений.

Таблиця 8.3 – Типи інструкцій, які вводяться в стані

Тип команди (інструкції)	Ідентифікатор	Опис команди
Дія входу	E	Дія, яка виконується один раз тільки під час входу в стан
Попередня циклічна дія	З-С-	Дія, яка містить певні умови й виконується перед перевіркою транзакції
Циклічна дія	C	Дія, яка виконується після перевірки транзакції
Дія виходу	X	Дія, яка виконується один раз тільки під час виходу зі стану

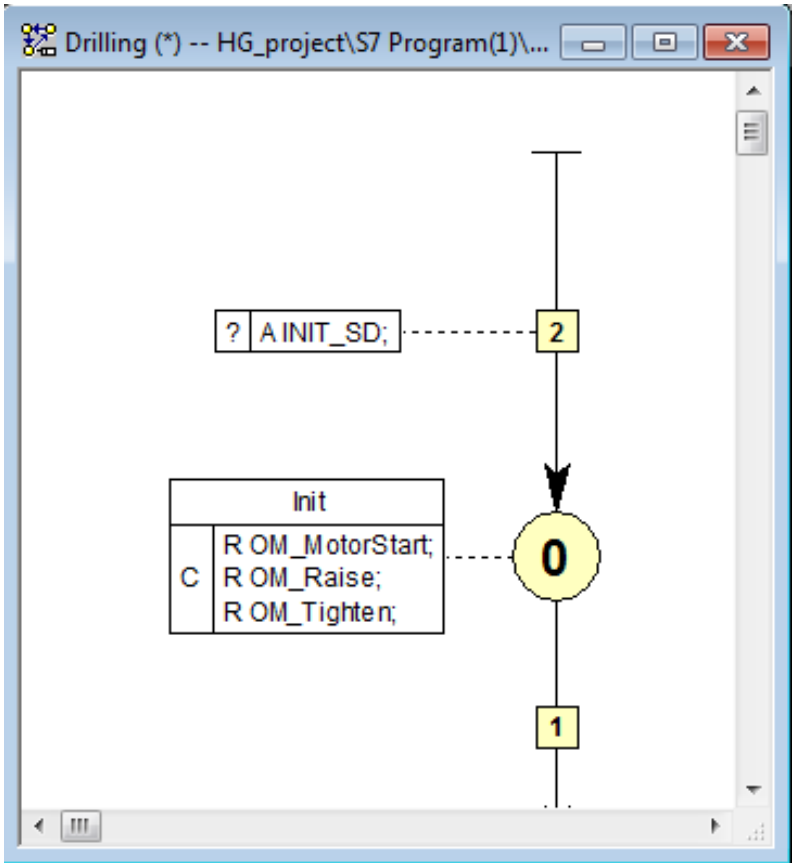


Рисунок 8.10 – Вигляд вікна після програмування стану 0

Якщо необхідно задати час, протягом якого процес може перебувати в стані, то слід вибрати команду «Supervision Time». Установлений за замовчуванням контрольний час 500 ms можна змінити. Якщо фактичний час перебування перевищить заданий контрольний час, то визначена попередньо змінна «ST_Expired» установиться в стан «1». При цьому в діагностичний буфер CPU буде видане повідомлення про помилку.

Для діагностики процесу виконання програми в стані можна призначити дві характеристики – помилка (функція F) і повідомлення (функція M). Ці призначення здійснюються у вікні Object Properties. У цьому ж вікні можна призначити коментар для стану.

Крок 5. Програмування транзакцій.

Транзакція містить розв'язну умову для переходу від одного стану до іншого. Стану може бути призначено одну або кілька транзакцій. Якщо виконуються умови більш ніж однієї транзакції, то перемикання відбудеться по транзакції з найвищим пріоритетом (1).

У мові Higraph використовуються три типи транзакцій (рис. 8.11) – стандартна, довільна й транзакція повернення.

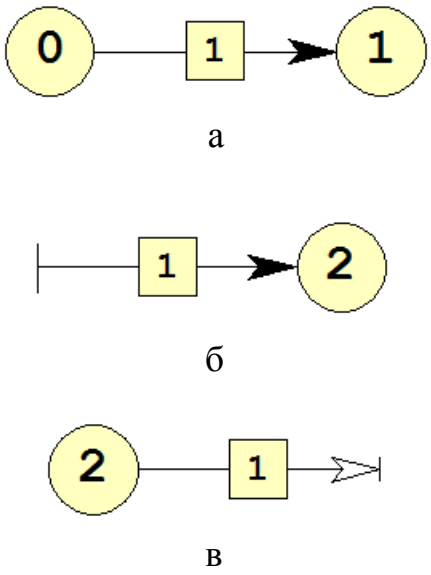


Рисунок 8.11 – Типи транзакцій

Стандартна транзакція (рис. 8.11, *а*) здійснює перехід зі стартового стану в наступний стан.

Довільна транзакція (рис. 8.11, *б*) іде від будь-яких станів до цільового стану. Вона має більш високий пріоритет, ніж інші типи транзакцій і обробляється безупинно незалежно від поточного стану графа стану. Такі транзакції використовуються для постійного контролю важливих умов з високим пріоритетом. Якщо запрограмована в довільній транзакції контролююча функція виконується, то здійснюється перехід на галузі процесу із цільовим станом.

Транзакція повернення (рис. 8.11, *в*) іде від поточного стану до попереднього активного стану.

Вставлення транзакції здійснюється командою контекстного меню Insert Transition. Тип транзакції залежить від позиції кінця транзакції.

Під час програмування транзакцій виконуються такі кроки:

- визначення пріоритету транзакції (не обов'язково);
- введення умов;
- введення дій транзакцій (не обов'язково);
- призначення імені транзакції (не обов'язково);
- введення коментарів (не обов'язково);
- установлення часу очікування (не обов'язково);

Якщо одному стану призначено кілька транзакцій, то редактор автоматично призначає цим транзакціям різні пріоритети. Бажаний рівень пріоритету можна встановити у вікні Object Properties.

Умови (Conditions) у транзакціях записуються з ідентифікатором «?» (знак питання). Умови програмуються у форматі команд мови STL.

Дії в транзакціях (Transition actions) програмуються з ідентифікатором «!» (знак оклику) теж у форматі команд мови STL. Ці команди виконуються однократно, коли транзакція перемикає стан.

Для введення команд необхідно двічі клацнути по транзакції, щоб відкрити вікно редагування. Далі в лівій частині вікна редагування вибрати зі списку Conditions або Transition actions і ввести STL-команду в правій області вікна. Введені команди відображаються у вікні графа станів як таблиця, «прикріплена» до транзакції. Слід урахувати, що оброблення команд починається з результатом логічної операції RLO = 1.

Крок 6. Програмування постійних інструкцій.

Постійні команди виконуються один раз у цикл, незалежно від поточного стану. У постійних командах можна програмувати такі основні процеси:

- обчислення змінних процесу;
- реєстрація й обробляння подій, на які процес повинен завжди реагувати, незалежно від поточного стану, наприклад, контроль захисту й блокувань;
- для редагування доступні такі типи постійних команд:

  1) циклічні дії (Preceding Cyclic Actions), які завжди виконуються на початку циклу (ідентифікатор C-).

  2) циклічні дії (Cyclic Actions), які завжди виконуються наприкінці циклу (ідентифікатор C).

Щоб програмувати ці дії, потрібно клацнути по таблиці команд із заголовком «Permanent Instructions». При цьому відкриється вікно редагування, у лівій області якого необхідно вибрати тип інструкції, а в правій – увести команду STL. Додавання команд здійснюється так: у лівій області потрібно виділити тип інструкції, потім натиснути праву кнопки й вибрати єдину команду Insert.

Після введення всіх команд вони будуть відображатися у вікні графа стану як таблиця.

Крок 7. Програмування операційних режимів.

Якщо в транзакції запрограмувати режим Manual (ідентифікатор M), то транзакція буде перемикатися тільки в ручному режимі, а якщо запрограмувати режим Auto (ідентифікатор A), то транзакція буде перемикатися тільки в автоматичному режимі. Необхідна особливість режиму встановлюється у вікні Object Properties. Після установлення режиму транзакція зафарбовується в рожевий (Auto) або блакитний (Manual) колір і забезпечується відповідним ідентифікатором (A або M).

Крок 8. Створення координувального графа.

З функціональної діаграми, представленої на рисунку 8.3, видно, що процес свердління складається з 8 станів.

Стан 1 характеризує вихідну позицію (ініціалізацію) – лещата розтиснуті, свердло знаходиться у верхньому положенні й мотор виключений. У цій позиції можливе установлення та зняття заготовки.

Перехід у стан 2 здійснюється пусковою кнопкою (Start_Button). Коли процес затискання закінчиться й спрацює тензодатчик зусилля затискача (Tension_Reached), транзакція перемкне процес у стан 3. У цьому стані запускається мотор обертання свердла й після досягнення заданої швидкості (сигнал Drill_Motor_running) відбувається перехід у стан 4 – вмикання привода подачі свердла.

Подача проводиться до моменту спрацьовування кінцевого вимикача в крайньому нижньому положенні свердлильної головки (стан 5). У цьому положенні свердло повинне якийсь час обертатися без подачі для зменшення пружних деформацій від осьової подачі свердла, а потім автомат повинен перейти в стан 6, де реалізується команда руху свердлильної головки нагору. По закінченні процесу, коли спрацює кінцевий вимикач крайнього верхнього положення головки, автомат перейде в стан 7. Тут відбувається вимикання мотора, й після його зупинки (сигнал з датчика швидкості Drill_Motor_stopped) перемикання в стан 8, у якому здійснюється розтискання лещат.

На рисунку 8.12 показаний координуючий граф станів Drilling, який відповідає функціональній діаграмі процесу свердління.

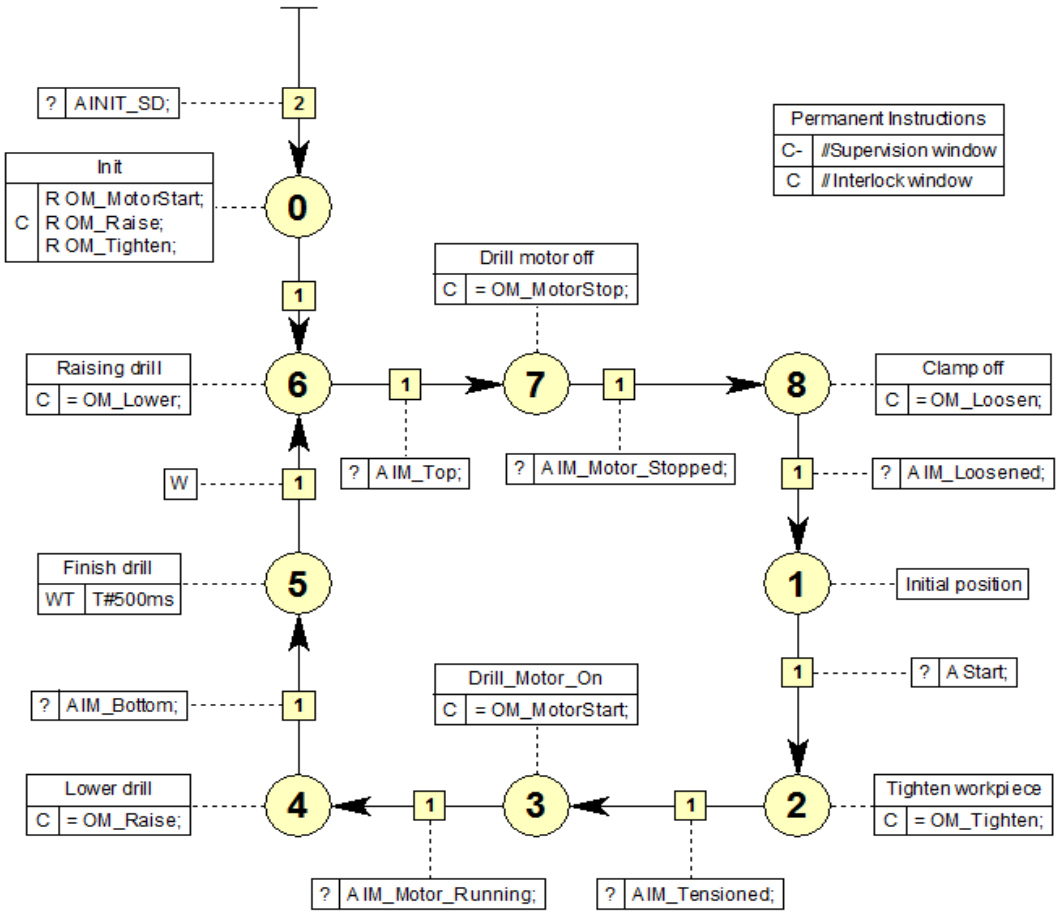


Рисунок 8.12 – Вигляд координувального графа станів Drilling для свердлильного верстата

8.4 Створення групового графа

Груповий граф визначає задану послідовність запитів до графів станів (вихідних кодів), які виконуються циклічно. Вихідні коди обробляються у програмувальному контролері як змінні групового графа області STAT.

Для створення групового графа потрібно виконати таке: у Simatic Manager відкрити папку Source Files із графами станів. Потім клацнути на порожньому місці правою кнопкою й у контекстному меню вибрати команду Insert New Object ► Graph group. Вставленому груповому графові слід привласнити ім'я, наприклад, Drilling_machine. Присвоєння імені здійснюється у вікні Object Propertis, яке відкривається за допомогою контекстного меню. Створити груповий граф можна у вікні S7-Higraph. Для цього потрібно відкрити будь-який граф станів і вибрати в меню File ► New Graph Group.

Вставляння вихідних кодів (графів станів) у груповий граф

Створений груповий граф відкривається з порожньою робочою областю. Для вставлення в цю область графів станів потрібно клацнути правою кнопкою на порожньому полі й у контекстному меню вибрати **Insert Instance**. При цьому відкриється вікно вибору файлу «Open», у якому відображаються всі створені до цього графи станів. Першим вибирається координуючий граф. Закривши вікно «Open» кнопкою ОК, потрібно вставити цей граф на робоче поле групового графа. Вставлений граф станів відображається у вікні групового графа прямокутником з іменем і номером. Імена вставлених графів станів відображаються як змінні в області оголошення STAT.

Операції вставлення потрібно повторити для всіх створених графів станів. На рисунку 8.13 показане вікно групового графа із вставленими графами станів, створеними для розглянутого тут прикладу.

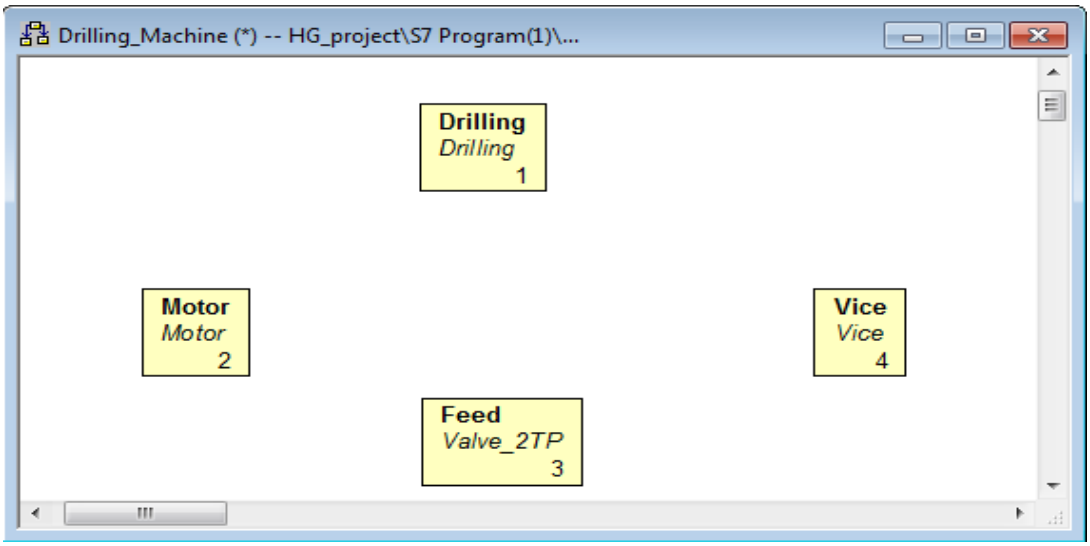


Рисунок 8.13 – Вигляд вікна групового графа «Drilling_Machine» із всіма вставленими графами станів

Установлення послідовності виконання процесу

Послідовність запускання графів станів задається у вікні Run Sequence, яке відкривається однойменною командою контекстного меню. Визначити цю послідовність можна за координувальним графом станів. Для розглянутого прикладу прийнята послідовність виконання: Drilling (1), Motor (2), Feed (3), Vice (4).

Призначення фактичних параметрів

Графи станів для окремих функціональних одиниць являють собою вихідні коди, які можна вставляти в будь-які проекти програм. Зазвичай розроблювач графа дає змінним ті імена, які відбивають сутність елементарного процесу керування, наприклад, для змінної «Вмикати мотор» призначає ім'я «Motor_On». Однак під час створення програми керування конкретним устаткуванням, у якому є кілька моторів, буде потрібна інша система іменування змінних. Тому в груповому графові передбачене зв'язування вихідних імен змінних вставлених графів станів зі змінними, які призначені в створюваній програмі. Процедура зв'язування називається призначенням поточних параметрів. Вона зводиться до такого: у вікні групового графа знаходяться вставлені графи станів (вихідні коди). У меню View вибираємо команду Details, яка відкриває в нижній області екрана вікно редагування. Це вікно містить вкладку «Current parameters». Виділяємо один із графів стану й відкриваємо цю вкладку з відображенням списку всіх змінних для обраного графа станів. Далі відкриваємо таблицю ідентифікаторів командою Options ► Symbol Table і робимо оголошення всіх вхідних і вихідних змінних із вказівкою їх фактичних адрес і типів даних. Приклад заповнення таблиці представлений на рис. 8.14.

S7 Program(1) (Symbols) -- HG_project

	Status	Symbol	Address	Data type	Comment
1		Clamp_Workpiece	Q 0.3	BOOL	Clamp/hold workpiece with set tension
2		CYCL_EXC	OB 1	OB 1	
3		DB_GG_Drilling	DB 1	DB 1	DB for drilling graph group
4		Drill_at_Bottom	I 0.2	BOOL	Limit switch for "drill at lowest position"
5		Drill_at_Top	I 0.3	BOOL	Limit switch for "drill at highest position"
6		Drill_Motor_On	Q 0.0	BOOL	Switch on drill motor
7		Drill_Motor_Runn...	I 0.0	BOOL	Feedback signal for "drill running at set speed"
8		Drill_Motor_Stop...	I 0.1	BOOL	Feedback signal for "drill stationary"
9		GG_Drilling	FC 1	FC 1	FC for drilling graph group
1		Lower_Drill	Q 0.1	BOOL	Lower drill via feed to lowest limit
1		Raise_Drill	Q 0.2	BOOL	Raise drill via feed to highest limit
1		Start_Button	I 0.7	BOOL	Start button for drilling machine
1		Tension_Reached	I 0.4	BOOL	Feedback signal for "workpiece set tension reached"
1					

Рисунок 8.14 – Приклад заповнення таблиці Symbols

Для того щоб призначити нове ім'я змінної (це ім'я визначене у таблиці символів), потрібно на вкладці «Current parameters» вибрати ім'я в стовпці Name, а в стовпці Current parameters правою кнопкою викликати команду контекстного меню Insert Symbol/Message.

При цьому відкриється список змінних таблиці Symbols, у якому потрібно вибрати відповідну змінну й натиснути Enter. На рисунку 8.15 показані відкриті списки змінних *вставленого і нового графів станів* у вікні групового графу.

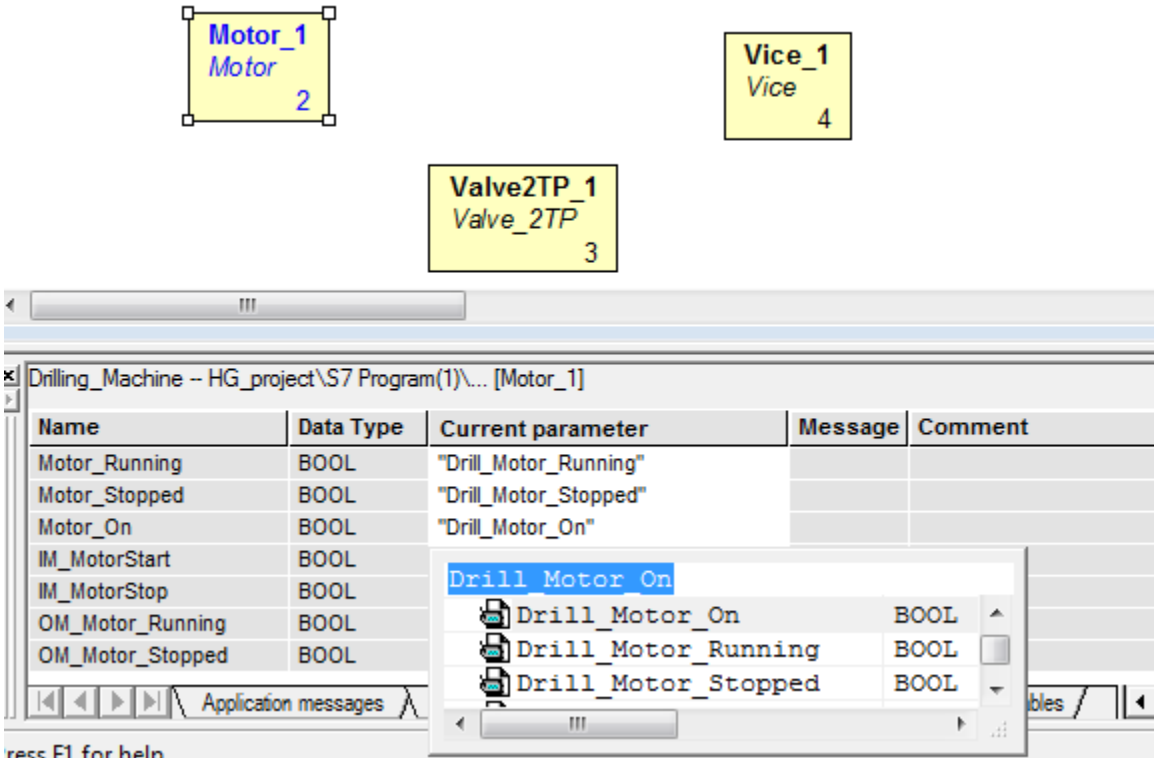


Рисунок 8.15 – Вибір змінної для присвоєння нового імені

Призначення поточних параметрів необхідно виконати для всіх змінних у всіх графах станів, причому в стовпець «Current parameters» потрібно внести навіть ті імена, які не мають відмінності від стовпця «Name». Нові параметри не призначаються тільки для змінних типу «in».

Програмування повідомлень

Повідомлення – це бінарна змінна, яка встановлюється графом-передавачем і обробляється в транзакціях графа-приймача. У транзакціях приймача програмується також дія, яка скидає біт отриманого повідомлення. Повідомлення служать засобами зв'язку між графами станів і використовуються для координації взаємодії графів стану в груповому графові.

Залежно від області дії використовуються два типи повідомлень:

Internal message – внутрішнє повідомлення для зв'язку між графом станів і груповим графом. Зв'язок здійснюється через бітову адресу блоку даних DB групового графа. Цей вид повідомлень використовується під час створення програми керування з одним груповим графом.

External message – зовнішнє повідомлення для зв'язку між графами станів, що перебувають у різних групових графах, або між Higraph-функціями FC й іншими програмами. Зв'язок здійснюється через загальнодоступну бітову адресу, установлену програмістом.

В якості повідомлень використовуються булеві змінні, оголошені як Message Type. Під час оголошення змінних разом з іменем слід указати ознаки – вихідне повідомлення позначити OM (output message), вхідне повідомлення позначити IM (input message).

Для пояснення механізму обміну повідомленнями розглянемо приклад. Нехай зі стану 3 (рис. 8.5) графа-передавача «Feed» у координуючий граф Drilling передається *вихідне* повідомлення OM_Bottom (досягнуте дно). Поточний параметр цього повідомлення записується у вікні Current parameter групового графа з адресою й іншим типом, тобто так: Drilling.IM_Bottom.

Координуючий граф Drilling ухвалює це *вхідне* повідомлення й використовує його в транзакції перемикання зі стану 4 у стан 5 (рис. 8.12), де передбачена логічна операція І між цим повідомленням і умовою переходу, що перебувають в акумуляторі (команда A IM_Bottom). Якщо RLO цієї операції буде дорівнювати «1», відбудеться перехід у стан 5, з якого після закінчення часу очікування W буде виконане перемикання в стан 6. Тут процес піде в іншому напрямку – уже координувальний граф Drilling повинен передати вихідне повідомлення OM_Lower, поточним параметром якого буде Feed.IM_Lower. Граф Feed ухвалює це вхідне повідомлення й використовує його в транзакції перемикання в стан 4 (команда A IM_Lower).

Таким чином, вихідні повідомлення відправляються зі станів, а вхідні використовуються в транзакціях.

Вхідні (in) і вихідні (out) повідомлення призначаються у вікні оголошення змінних Variables групового графа в стовпці Message Type. На рисунку 8.16 показані вхідні й вихідні повідомлення для графа станів Motor.

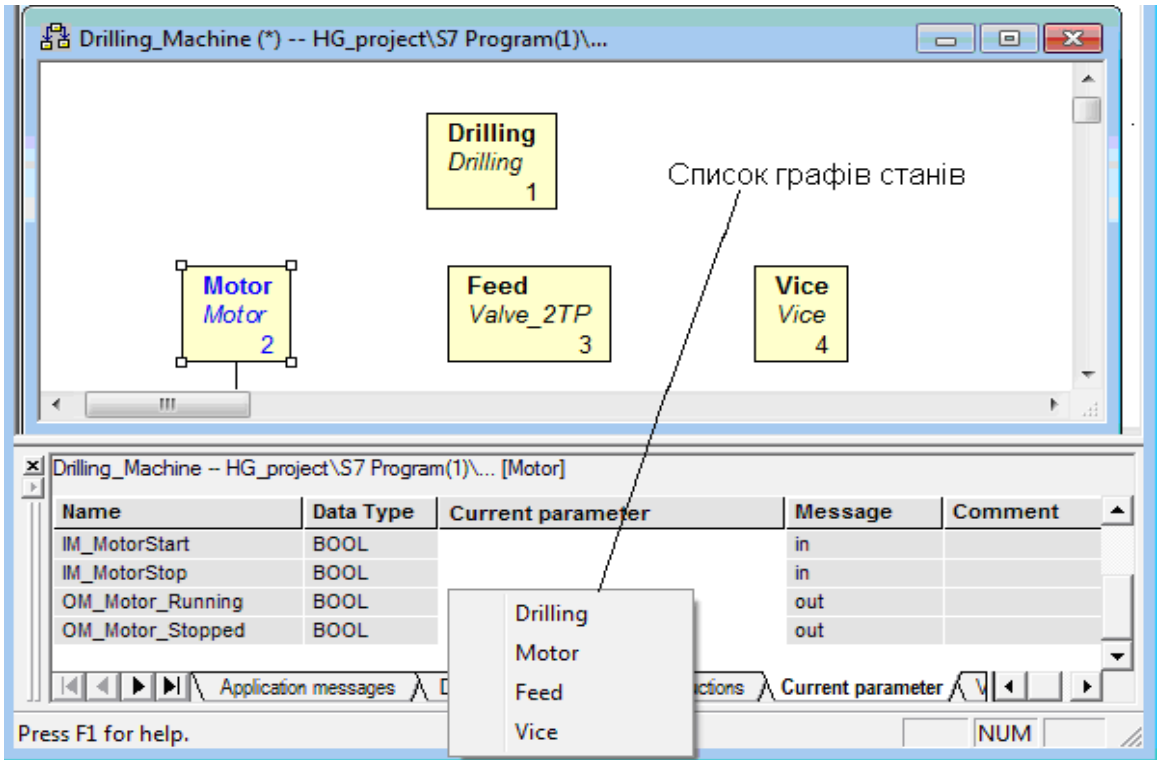


Рисунок 8.16 – Вигляд вікна оголошення змінних із вхідними й вихідними повідомленнями

Слід урахувати, що для змінних типу OUT програмуються дії присвоєння, наприклад, «= OM_Motorstart» (рис. 8.12, стан 3) або установки, наприклад, «S OM_Motorstart», а для вхідних змінних типу IN програмуються умови, наприклад, «A IM_Bottom» (перехід у стан 5).

Призначення поточних параметрів для вихідних повідомлень здійснюється у вікні Current parameter групового графа. Процес призначення здійснюється в такий спосіб: спочатку необхідно виділити граф станів. Далі у вікні оголошення змінних прокрутити список змінних і знайти вихідні повідомлення з типом «out». На перетинанні імені повідомлення й стовпця «Current parameter» клацанням правої кнопки викликати контекстне меню, у якому треба вибрати команду Insert Symbol/Message. За цією командою відкривається невелике вікно зі списком графів, вставлених у груповий граф. На наведеному вище рисунку 8.16 показаний список графів для передачі вихідного повідомлення OM_Motor_Running.

У зв‘язку з тим, що повідомлення OM_Motor_Running повинне відправлятися в координуючий граф станів Drilling, клацаємо по імені Drilling і воно вставляється редактором у гніздо таблиці. Далі додаємо до імені крапку й редактор автоматично виводить список змінних для призначення цьому вихідному повідомленню поточного параметра. Цей стан показано на рисунку 8.17.

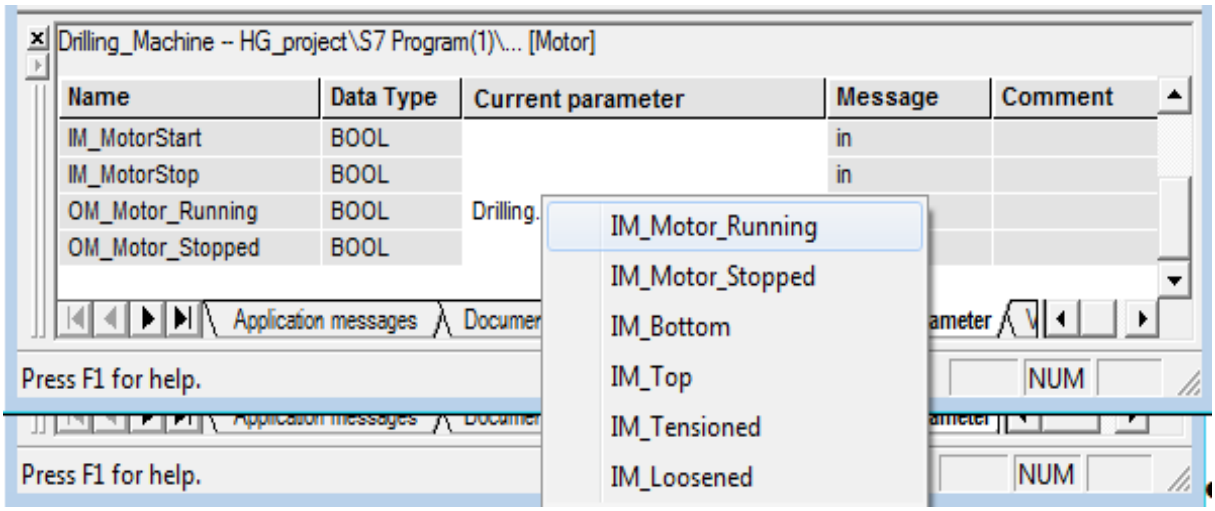


Рисунок 8.17 – Вигляд вікна для призначення поточного параметра

Вибираємо зі списку IM_Motor_Running і одержуємо результат – Drilling.IM_Motor_Running. Після призначення всіх повідомлень груповий граф приймає вигляд, показаний на рисунку 8.18.

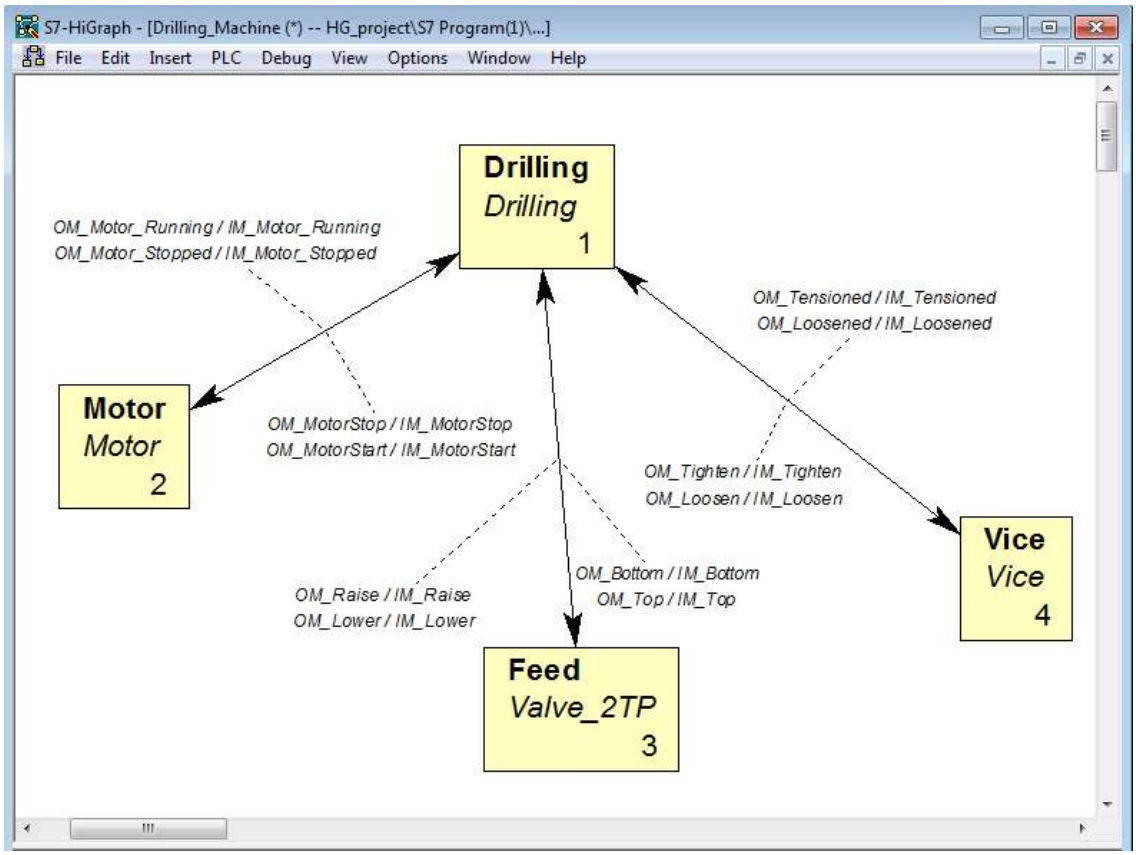


Рисунок 8.18 – Вигляд групового графа після призначення всіх повідомлень

8.5 Компіляція вихідних кодів і створення блоків програми

Під час збереження графів станів у контейнері «Source Files» програми S7 ніякої перевірки синтаксису не відбувається. Тому процес редагування може здійснюватися протягом будь-якої кількості сеансів роботи. По закінченні роботи зберегти об'єкти можна командою File > Save.

Процес компілювання застосовується тільки для групових графів, а індивідуальні графи станів не компілюються. Під час компіляції груповий граф спочатку зберігається, потім редактор Higraph перевіряє синтаксис програми, створює функцію (FC) і блок даних (DB), а потім запам'ятовує їх у контейнері «Blocks» програми S7.

Якщо під час компіляції виявлені синтаксичні помилки, то вони будуть відображені у вікні повідомлення і логічний блок, що призначений до виконання, не буде створений. Якщо після перевірки синтаксису з'являться тільки попередження, то логічний блок буде створений.

Для компіляції групового графа необхідно виконати таке: У меню Options вибрати команду «*ім'я_групового_графа* Settings». У вікні, що відкрилося, перейти на закладку Compile (рис. 8.19) і ввести ім'я функції (FC) і блоку даних (DB). Під час призначення імені можна вказувати абсолютне ім'я, наприклад, FC 1. Тут же можна встановити інші параметри настроювання.

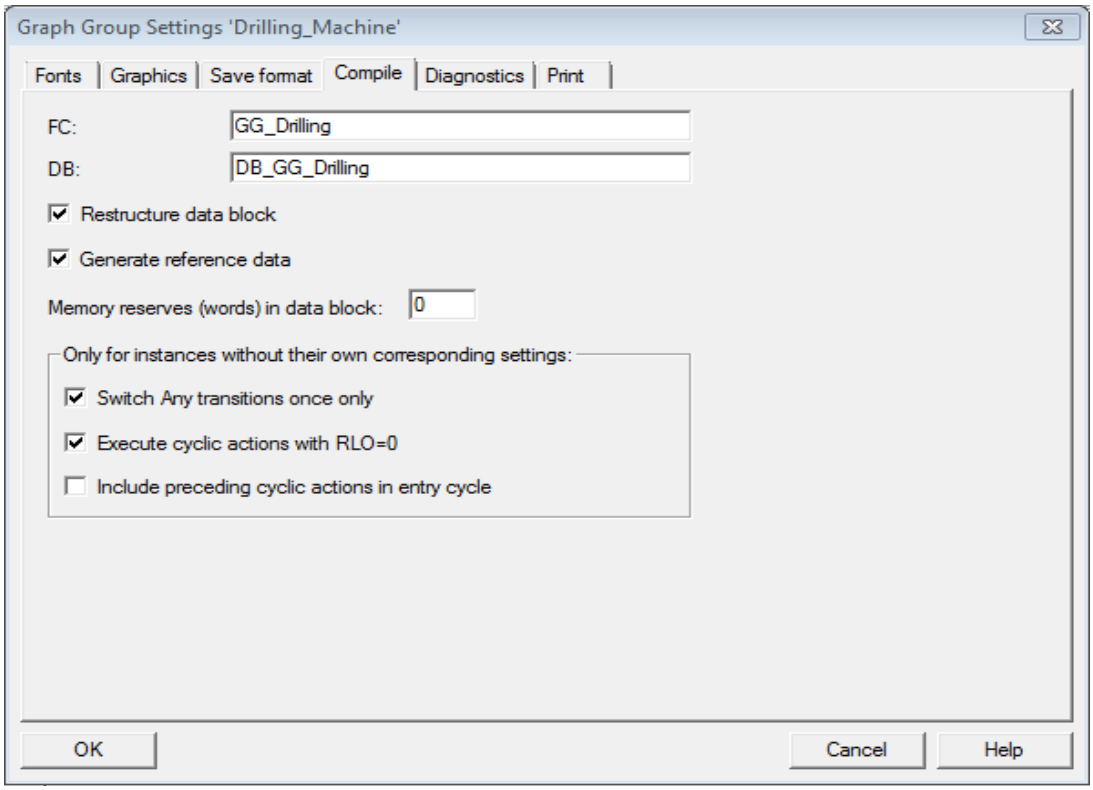


Рисунок 8.19 – Вигляд вікна настроювання на закладці Compile

Опції настроювань забезпечують такі ефекти:

- Restructure data block – блок даних DB створюється в процесі трансляції;
- Generate reference data – довідкові дані генеруються автоматично;
- Memory reserves (words) in data block – установлюється резерв для додаткових графів станів і повідомлень;
- Switch Any transitions once only – опція запобігає повторному перемиканню транзакції запуску;
- Execute cyclic actions with RLO=0 – опція змушує виконати циклічні дії до виходу зі стану при RLO=0;
- Include preceding cyclic actions in entry cycle – опція змушує виконати дії, передбачені до входу в стан (C-).

Після настроювань на вкладці Compile слід перемкнутися на вкладку Diagnostics, установити прапорець на опцію «Format converter diagnostics» і закрити вікно.

Для переходу до процесу компіляції потрібно вибрати команду меню File ► Compile. Процес компіляції відображається у вікні повідомлень (нижня область екрана). Під час виводу помилок потрібну позицію можна знайти, двічі клацнувши на повідомленні про помилки. Після виправлення помилок слід перекомпілювати груповий граф. На рисунку 8.20 показаний вигляд вікна групового графа з результатами компіляції.

Для циклічного оброблення програми Higraph у програмувальному контролері вона повинна викликається з організаційного блоку OB1.

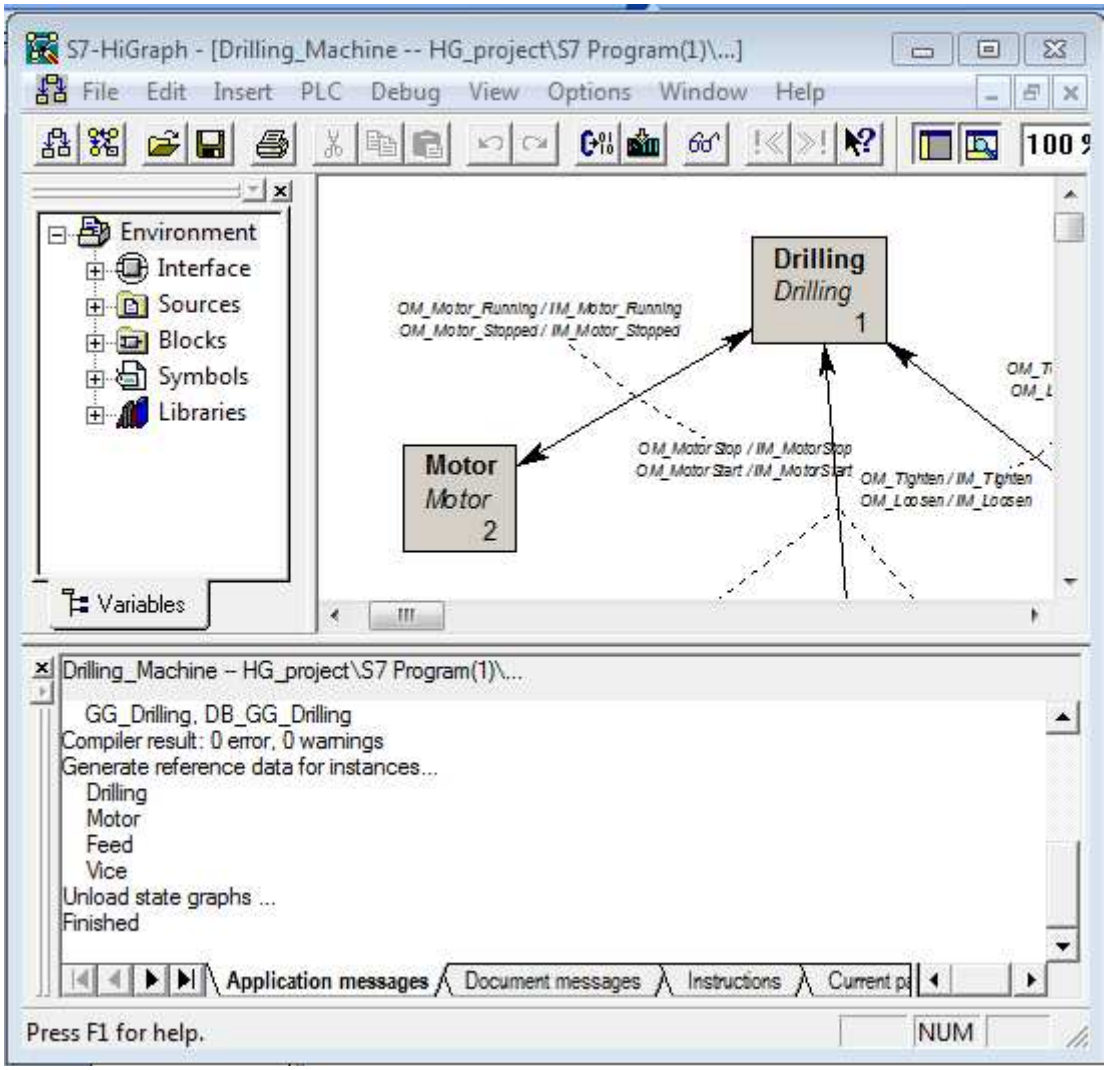


Рисунок 8.20 – Вигляд вікна групового графа з результатами компіляції

Блок OB1 можна програмувати в LAD/STL/FBD редакторі базового пакета STEP 7. Функція (FC), створена в Higraph, має вхідний параметр «INIT_SD». Цей параметр повинен установлюватися в «1» при включенні контролера, інакше графи станів у груповому графі ініціалізуватися не будуть. Сигнал установки «INIT_SD» може бути сформований з використанням стартової інформації OB1 (змінна #OB1_SCAN_1) і збережений у тимчасовій змінній OB1, наприклад, в #startup.

Коли блок OB1 вставляється в програму, необхідні змінні вже оголошені в розділі TEMP і залишається дописати цей розділ стартовою змінною #startup (рис. 8.21).

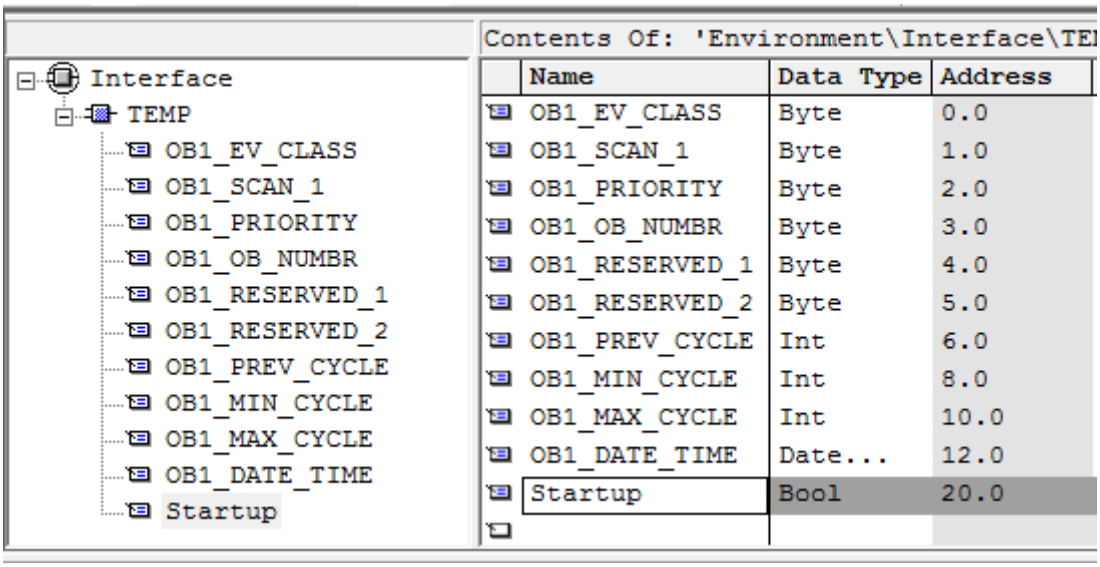


Рисунок 8.21 – Розділ оголошення змінних блоку OB1

Програма організаційного блоку OB1 містить у собі генерацію біта запуску, виклик функції FC1 й ініціалізацію змінної INIT_SD:

```
L OB1_SCAN_1              // генерація біта запуску
L 1
==I
=     #startup            //установка першого циклу OB1
      CALL  "GG_Drilling" //виклик функції FC1
      INIT_SD:=#startup   //ініціалізація сигналу установки.
```

8.6 Завантаження програми в контролер та її налагодження

Для завантаження програми користувача в контролер повинні бути виконані такі вимоги:

- програма, яка буде завантажуватися, повинна бути відкомпільована без помилок;
- повинен бути запрограмований виклик Higraph FC із циклічно виконуваного блоку;
- пристрій програмування й програмувальний контролер повинні бути з'єднані.

Завантаження програми здійснюється в такій послідовності:

1. Установити CPU у режим STOP;
2. Відкрити Higraph програму в SIMATIC Manager;
3. Вибрати необхідні блоки в контейнері блоків:
   - Higraph FC;
   - Higraph DB;
   - блок виклику (OB або FB);
   - функцію FC101, якщо потрібна діагностика;
4. Вибрати команду меню PLC > Download.

Для завантаження в програмувальний контролер тільки функції (FC) з блоком даних (DB) необхідно при відкритому груповому графові вибрати команду меню PLC > Download і в діалоговім вікні «Download» вибрати завантаження блоку даних DB разом з функцією FC.

Існує можливість контролю й зміни програми, у той час як вона виконується в CPU. Це дозволяє знайти помилки, які не були відображені формальною логікою перевірки під час створення програми або перевірки синтаксису протягом трансляції.

Редактор Higraph дозволяє виявити такі помилки:

- програмні помилки, наприклад, неправильно встановлений контрольний час;
- логічні помилки в структурі програми, які свідчать, що запрограмовані стани й транзакції не відповідають необхідній послідовності технологічних операцій.

Слід урахувати, що функція налагодження сповільнює проходження програми й може спричинити збої або перевищення часу циклу.

Доступні такі функції налагодження й контролю:

- контроль стану програми;
- контроль і зміна значень змінних;
- оцінювання довідкових даних.

Перш ніж використовувати контролювальні функції, повинні бути виконані такі вимоги:

- пристрій програмування повинен бути інтерактивно пов'язаний із процесорним модулем (CPU);
- програма повинна бути відкомпільована без помилок;
- програма (включаючи FC, DB, і OB) повинна бути завантажена у CPU;
- CPU повинен бути в режимі RUN (читання) або в режимі RUN-P (читання й записування);
- Higraph FC повинна викликатися із блоку OB1.

Просування програми через індивідуальні стани й транзакції, а також поточна інформація щодо оброблюваних команд показується на екрані. Вікна Higraph надають такі можливості контролю:

- у вікні групового графа видно стани й усі графи станів групового графа, причому поточний стан відображений у кожному графові;
- у вікні графа станів активний стан позначений кольором, а транзакція, яка спричинила цей стан, а також попередній активний стан, затінена.

Щоб запустити контроль стану програми, необхідно:

- при відкритому груповому графові вибрати команду меню Debug ► Monitor для відображення стану групового графа;
- вибрати один або декілька графів станів, а потім команду меню Edit ► Open Object. Кожний обраний граф стану буде відкритий інтерактивно. При цьому відображається детальна інформація стану;

- таблиця з інформацією стану відображається спочатку для переміщення з найвищим пріоритетом, що веде від активного стану. Якщо потрібно, можна вибрати іншу активну таблицю команд, щоб відобразити її інформацію;
- для контролю інших графів станів потрібно повернутися до короткого огляду стану групового графа, вибрати інший граф стану й знову застосувати команду меню Edit ► Open Object;
- щоб вийти з відображення стану програми, слід дезактивувати команду меню Debug ► Monitor.

Для редагування змінних потрібно вибрати команду меню Debug ► Select Variable. У діалоговім вікні, що відкрилося, потрібно вибрати необхідні графи станів і їх змінні, а після редагування натиснути кнопку «ОК».

Під час налагодження програми можна використовувати різні довідкові дані. Для їхнього перегляду слід вибрати команду Options ► Reference.

Контрольні питання

1. Яке призначення має граф станів?
2. Яка мова застосовується для програмування графу станів?
3. З якою метою створюється груповий граф?
4. Що потрібно визначити за допомогою функціональної діаграми?
5. Що дозволяється вносити в ім‘я змінної?
6. Де вводиться адреса змінної?
7. Які позначення мають ідентифікатори інструкцій в станах?
8. На які типи поділяються транзакції?
9. Яку роль відіграють повідомлення у груповому графі?
10. Звідки відправляються вихідні повідомлення і де вони використовуються?
11. У чому полягає процедура призначення фактичних параметрів?
12. За яких умов створюється логічний блок програми Higraph?

ЛІТЕРАТУРА

1. **Бергер, Ганс**. Автоматизация посредством STEP 7 с использованием STL и SCL и программируемых контроллеров SIMATIC S7-300/400. [Электронный ресурс] / Ганс Бергер – 2001. – Режим доступа: http://mirknig.com/knigi/nauka_ucheba/1181227029-avtomatizaciya-posredstvom-step-7-s-ispolzovaniem.html.

2. **Бергер, Ганс.** Автоматизация с помощью программ STEP7 LAD и FBD [Электронный ресурс] /Ганс Бергер. – 2-е изд., перераб. – 2001. – Режим доступа: http://www.twirpx.com/file/119315/.

3. SIMATIC S7. Введение в STEP 7 [Электронный ресурс]. – Режим доступа: http://www.twirpx.com/file/60377/.

4. SIMATIC. Программирование с помощью STEP 5 V5.3. Руководство. Редакция 01/2004, A5E00261405-01.

5. Интерактивный каталог продуктов Siemens IA&DT. Техника автоматизации SIEMENS [Электронный ресурс]. – Режим доступа: https://eb.automation.siemens.com/goos/catalog/Pages/ProductData.aspx?catalogRegion=RU&language=ru&nodeid=10045207&tree=CatalogTree&regionUrl=%2fru#activetab=product&.

6. Программируемые контроллеры SIMATIC S7 [Электронный ресурс]. – Режим доступа: *http://automation-drives.ru/as/products/simatic_s7/.*

7. Автоматизация в промышленности. Каталог Siemens CA01 2010 [Электронный ресурс]. – Режим доступа: *http://www.sms-automation.ru/distribution/Siemens/catalog/index.php?nodeid=1102*.

8. SIMATIC. HiGraph для S7-300/400. Руководство [Электронный ресурс]. – Режим доступа: http://automation-drives.ru /as/download/doc/software/eng/HiGraph_V4.1_r.pdf.

9. SIMATIC. S7-GRAPH V5.3 для S7-300/400. Программирование систем последовательного управления. Руководство. Редакция 02/2004, A5E00290656-01.

10. SIMATIC. Программируемые контроллеры S7-400, M7-400. Руководство пользователя C79000–G7076–C400–01. Выпуск 2.

ГЛОСАРІЙ

Адреса. Адреса – це позначення для певного операнда або області операндів, наприклад: вхід E 12.1; меркерне слово MW 25; блок даних DB 3.

Адреса PROFIBUS. Кожний абонент шини для однозначної ідентифікації в PROFIBUS-DP повинен одержати адресу PROFIBUS. Програматор PG має адресу PROFIBUS ”0”. Master- і Slave-пристрої DP мають адреси PROFIBUS у діапазоні від 1 до 125.

Акумулятор. Акумулятори – це регістри в CPU, які служать як проміжна пам'ять для операцій завантаження, передачі, а також порівняння, перетворення й арифметичних операцій.

Аналоговий модуль. Аналогові модулі перетворюють аналогові параметри процесу, наприклад, температуру у цифрові величини, які можуть далі оброблятися процесором, або перетворюють цифрові величини в аналогові керуючі впливи.

Апаратні засоби. Під апаратними засобами розуміють все фізичне й технічне встаткування програмувального контролера.

Блок даних. Блоки даних (DB) – це області даних у прикладній програмі, що містять дані користувача. Є глобальні блоки даних, до яких можна звертатися із всіх кодових блоків і екземпляри блоків даних, які поставлені у відповідність певному виклику FB.

Блок живлення. Блок живлення сигнальних і функціональних модулів і підключеної до них процесної периферії.

Буферна пам'ять. Буферна пам'ять резервує області пам'яті CPU при зникненні напруги живлення й певних областей даних і меркери, таймери й лічильники залишаються реманентними.

Буферізація модуля. Забезпечення модуля буферною пам'яттю.

Відображення процесу. Відображення процесу – це складова частина системної пам'яті CPU. На початку циклічної програми сигнальні стани модулів уведення передаються відображенню процесу на входах, а відображення процесу на виходах передається модулям виведення як сигнальний стан.

Глибина вкладення . За допомогою виклику блоків один блок може викликатися з іншого. Під глибиною вкладення розуміють кількість одночасно викликаних кодових блоків.

Глобальні дані. Глобальні дані – це дані, до яких можна звернутися з будь-якого кодового блоку (FC, FB, OB). Зокрема, це меркери M, входи E, виходи A, таймери, лічильники й блоки даних DB. До глобальних даних можна звертатися абсолютно або символічно.

Дані статичні. Статичні дані – це дані, використовувані тільки усередині функціонального блоку. Ці дані зберігаються в екземплярі блоку даних, що належить функціональному блоку. Дані, що перебувають в екземплярі блоку даних, зберігаються до наступного виклику функціонального блоку.

Дані тимчасові. Тимчасові дані – це локальні дані блоку, які під час обробки блоку накопичуються в L-стеці й після обробки стають недоступними.

Джерело живлення. Пристрій для живлення сигнальних і функціональних модулів і підключених до них входів/виходів процесу.
Діагностичний буфер. Діагностичний буфер – це буферизована область пам'яті CPU, у якій накопичуються діагностичні події в послідовності їхньої появи.
Діагностичне переривання. Модулі, здатні до діагностики, повідомляють розпізнані системні помилки CPU через діагностичні переривання.
Екземпляр блоку даних. Кожному виклику функціонального блоку в прикладній програмі STEP 7 ставиться у відповідність блок даних, що генерується автоматично. В екземплярі блоку даних зберігаються значення вхідних, вихідних і прохідних параметрів, дані, локальні в блоці.
Завантажувальна пам'ять. Завантажувальна пам'ять – це складова частина центрального модуля. Вона містить об'єкти, створені пристроєм програмування і реалізується або як вставна плата пам'яті, або як жорстко убудована пам'ять.
Задня шина. Задня шина – це розташована на задній стінці модулів послідовна шина даних, через яку модулі здійснюють зв'язок один з одним і одержують необхідне живлення. Зв'язок між модулями створюється за допомогою шинних з'єднувачів.
Земля. Струмопровідний ґрунт, електричний потенціал якого в будь-якій точці може бути встановлений на нуль. У районі заземлювача ґрунт може мати потенціал, відмінний від нуля. У зв'язку із цією обставиною часто застосовується термін "Еталонна земля"
Ізольовані модулі. У випадку ізольованих модулів уведення-виведення опорні потенціали ланцюгів керування й навантаження гальванічно розв'язані один з одним, наприклад, за допомогою оптронів, контактів реле або трансформаторів. Ланцюги уведення-виведення можуть бути підключені до загального потенціалу.
Індикація помилок. Індикація помилок – це одна з можливих реакцій операційної системи на помилки виконання програми. Інші можливі реакції: реакція на помилку в прикладній програмі, стан STOP CPU.
Інтерфейсні субмодулі. Субмодулі, які забезпечують комп'ютер для рішення завдань автоматизації додатковими інтерфейсами, наприклад, VGA, COM і т.д.
Інтерфейс багатоточечний. див. MPI
Клас пріоритету. Операційна система S7-CPU надає до 26 класів пріоритету ("рівнів обробки програми"), яким поставлені у відповідність різні організаційні блоки. Класи пріоритету визначають, які OB можуть переривати інші OB. Якщо клас пріоритету охоплює декілька OB, то вони один одного не переривають, а обробляються послідовно.
Кодовий блок. Кодовий блок – це блок в SIMATIC S7, що містить частину прикладної програми STEP 7 (на противагу блоку даних, що містить тільки дані).
Комунікаційний процесор. Комунікаційні процесори – це модулі з'єднань "точка-до-точки" і з'єднань за допомогою шини.

Консистентні дані. Дані, які змістовно зв'язані й не можуть бути розділені, називаються консистентними даними. Наприклад, значення, одержувані від аналогових модулів, завжди повинні оброблятися консистентно, тобто значення з аналогового модуля не може бути перекручене при зчитуванні у два різних моменти часу.

Контроль спрацьовування. Це параметр Slave-пристрою. Якщо до Slave-пристрою протягом часу контролю не здійснюється звернень, то він переходить у безпечний стан, тобто встановлює свої виходи в "0".

Конфігурація. Призначення модулів стійкам (слотам) і адресам. Розрізняють фактичну конфігурацію (фактично встановлені модулі) і задану. Задана конфігурація визначається при конфігуруванні за допомогою STEP 7, COM PROFIBUS (або COM ET 200 Windows). Завдяки цьому операційна система при запуску може розпізнати можливу невірну комплектацію.

Конфігурування. Призначення модулів носіям модулів/слотам і адресам, наприклад, у випадку сигнальних модулів.

Лічильники. Лічильники – це складові частини системної пам'яті CPU. Уміст лічильників може бути змінений за допомогою команд STEP 7, наприклад, рахувати вперед / назад).

Маркер. Право доступу до шини.

Меркери. Меркери – це складова частина системної пам'яті CPU для зберігання проміжних результатів. До них можна звертатися як до бітів, байтів, слів або подвійних слів.

Метод Master-Slave. Метод доступу до шини, при якому в міру потреби тільки один абонент є Master-пристроєм DP, а всі інші абоненти є Slave-пристроями DP.

Незаземлений. Не має гальванічного зв'язку із землею.

Новий пуск. При запуску центрального процесора, наприклад, при повороті перемикача режимів роботи з положення STOP в RUN або при включенні напруги, перед циклічною обробкою програми (OB 1) спочатку обробляється організаційний блок OB 100 (новий пуск). При новому пуску зчитується відображення процесу на входах і прикладна програма STEP 7 обробляється, починаючи з першої команди OB 1.

Обробка помилок через OB. Якщо операційна система розпізнає певну помилку, наприклад, помилку доступу в STEP 7, то вона викликає передбачений для цього випадку організаційний блок (OB помилок), у якому може бути визначене подальше поводження CPU.

Операційна система CPU. Операційна система CPU організує всі функції й процеси CPU, не зв'язані зі спеціальним завданням керування.

Організаційний блок. Організаційні блоки (OB) утворять інтерфейс між операційною системою CPU і прикладною програмою. В організаційних блоках установлюється послідовність обробки прикладної програми.

Пам'ять користувача. Пам'ять користувача містить кодові блоки й блоки даних прикладної програми. Пам'ять користувача може бути убудована в CPU або перебувати на вставних платах або модулях пам'яті. Однак прикладна програма в принципі обробляється з робочої пам'яті CPU.

Параметр. 1. Змінна кодового блоку STEP 7. 2. Змінна для настроювання режиму роботи модуля (одна або декілька на модуль). Кожний модуль при поставці має деяке раціональне основне настроювання, що може бути змінене конфігуруванням за допомогою STEP 7. Параметри бувають статичні й динамічні.

Параметри динамічні.

Динамічні параметри модулів, на противагу статичним, можуть бути змінені під час роботи викликом SFC у прикладній програмі, наприклад, граничні значення сигнального модуля аналогового уведення.

Параметри модуля. Параметри модуля – це величини, за допомогою яких можна набудовувати характеристики модуля. Розрізняють статичні й динамічні параметри модуля.

Параметри статичні. Статичні параметри модулів, на противагу динамічним, не можуть бути змінені за допомогою прикладної програми, а тільки шляхом конфігурування в STEP 7, наприклад, вхідне запізнювання сигнального модуля цифрового уведення.

Параметризація. Під параметризацією розуміють настроювання поводження модуля.

Параметруюча програма. Кожен Slave-пристрій DP має одну параметруючу програму. При запуску параметруюча програма передає дані про параметри на DP-Slave. Вона має доступ до Slave-пристрою DP на читання й запис і може змінювати конфігурацію Slave-пристрою DP.

Переривання. Операційна система CPU знає 10 різних класів пріоритетів, що регулюють обробку прикладної програми. До цих класів пріоритетів належать, серед іншого, переривання, наприклад, переривання від процесу. З появою переривання операційною системою автоматично викликається відповідний організаційний блок, у якому користувач може запрограмувати бажану реакцію.

Переривання за часом. Переривання за часом ставиться до одного із класів пріоритету при обробці програми SIMATIC S7. Воно генерується залежно від певної дати (або щодня) і часу доби, наприклад, 9:50 або щогодини, щохвилини. Потім обробляється відповідний організаційний блок.

Переривання із затримкою. Переривання із затримкою належить до одного із класів пріоритету при обробці програми SIMATIC S7. Воно генерується при спрацьовуванні запущеного в прикладній програмі таймера. Потім обробляється відповідний організаційний блок.

Переривання циклічне. Циклічне переривання генерується CPU періодично через параметруємі проміжки часу. Потім викликається відповідний організаційний блок.

Периферійна шина. Складова частина задньої шини S7–300 у системі автоматизації, оптимізована для швидкого обміну сигналами між IM 153 і сигнальними модулями. Через периферійну шину передаються корисні дані, наприклад, вхідні цифрові сигнали сигнального модуля і системні дані, наприклад, набори параметрів сигнального модуля, установлені за замовчуванням.

Плата пам'яті. Плати пам'яті – це засоби запам'ятовування у форматі пластикових карток для CPU і CP. Вони реалізуються як RAM або FEPROM.

Помилка виконання. Помилка, що виникає при обробці прикладної програми в системі автоматизації, тобто не в керованому процесі.

Потенційна розв'язка. В потенційно розв'язаних модулях уведення опорні потенціали керуючих і виконуючих ланцюгів струму гальванічно розділені; наприклад, оптичним елементом зв'язку, контактом реле або трансформатором.

Потенційний зв'язок. У потенційно зв'язаних модулів уведення опорні потенціали керуючих і виконуючих ланцюгів електрично з'єднані.

Провідні пристрої. Пристрої DP–Master, які можуть, якщо вони мають маркер, посилати дані іншим абонентам і запитувати дані від інших абонентів (активних абонентів).

Прикладна програма. В SIMATIC існує розходження між операційною системою CPU і прикладними (користувальницькими) програмами. Останні створюються за допомогою програмного пакета STEP 7 на можливих мовах програмування і зберігаються в кодових блоках.

Пристрій децентралізованої периферії. Це блоки уведення-виведення, які використовуються не в центральному модулі, а змонтовані децентралізовано у віддаленні від CPU. Пристрої децентралізованої периферії з'єднуються з Master-пристроєм DP через PROFIBUS-DP.

Пристрій програмування. Пристрій програмування – це, в сутності, персональні комп'ютери, придатні до промислового використання, компактні й транспортабельні. Вони характеризуються наявністю спеціального апаратного й програмного забезпечення для роботи із програмувальною апаратурою керування SIMATIC.

Пріоритет ОВ. Операційна система CPU розрізняє класи пріоритету, наприклад, циклічну обробку програми, обробку програми, керовану перериваннями від процесу. Кожному класу пріоритету поставлені у відповідність організаційні блоки (ОВ), у яких користувач S7 може запрограмувати деяку реакцію. У відповідності зі стандартом ОВ мають різні пріоритети, що визначають у якій послідовності вони повинні оброблятися або, навпаки, переривати один одного у випадку одночасного виклику.

Програмувальна апаратура керування с пам'яттю. Програмувальна апаратура керування з пам'яттю (SPS) – це електронні пристрої керування, функції яких зберігаються у вигляді програми в пристрої керування. Тому механічний і електричний монтаж пристрою не залежать від виконуваної їм функції. Програмувальна апаратура керування з пам'яттю має структуру обчислювальної машини. Вона складається із CPU (центрального процесора) з пам'яттю, модулів уведення-виведення й внутрішньої системи шин. Периферія й мова програмування орієнтуються на потреби техніки керування.

Робоча пам'ять. Робоча пам'ять – це RAM-пам'ять в CPU, до якої процесор звертається під час обробки прикладної програми.

Робочий режим. Системи автоматизації SIMATIC S7 знають наступні робочі режими: STOP, ANLAUF , RUN.

Реакція на помилку. Операційна система може реагувати на помилку виконання в такий спосіб: переведення системи автоматизації в стан STOP, виклик організаційного блоку, у якому користувач може запрограмувати реакцію або відображення помилки.

Резервна пам'ять. Резервна пам'ять забезпечує буферизацію областей пам'яті в CPU без буферної батареї. Буферизується параметруєма кількість таймерів, лічильників, меркерів і байтів даних, реманентні таймери, лічильники, меркери й байти даних.

Реманентні дані. Реманентні дані не губляться при зникненні живлення, якщо є в розпорядженні буферна батарея.

Сегмент. див. Шинний сегмент

Сигнальний модуль. Сигнальні модулі (SM) утворюють інтерфейс між процесом і системою автоматизації. Є цифрові модулі уведення й виведення, а також аналогові модулі уведення й виведення.

Системна діагностика. Системна діагностика – це розпізнавання, аналіз і формування повідомлень про помилки, що виникають усередині системи автоматизації. Прикладами таких помилок є програмні помилки або несправності в модулях. Системні помилки можуть відображатися за допомогою індикаторів або в повідомленнях програми STEP 7.

Системна пам'ять. Системна пам'ять убудована в центральний процесор і виконана у вигляді RAM. У системній пам'яті зберігаються області операндів, наприклад, таймери, лічильники, меркери, а також області даних, потрібних операційній системі, наприклад, буфер для комунікації.

Системна функція. Системна функція (SFC) – це функція, убудована в операційну систему CPU, що при необхідності може бути викликана в прикладній програмі STEP 7.

Системний функціональний блок. Системний функціональний блок (SFB) – це функціональний блок, убудований в операційну систему CPU, що при необхідності може бути викликаний у прикладній програмі STEP 7.

Структура децентралізована. Децентралізована структура має місце, якщо пристрої периферії не перебувають безпосередньо в одній стійці із центральним модулем або в сусідній розподільній шафі, а просторово віддалені друг від друга й зв'язані один з одним через комунікаційну шину.

Структура централізована. Централізована структура має місце, коли пристрої периферії й центральний модуль поміщені в одній стійці або в сусідній розподільній шафі.

Сумарний струм. Сума струмів всіх вихідних каналів цифрового модуля виведення.

Таблиця входів процесу. Область пам'яті, в якій фіксуються дані, що надійшли від вхідних пристроїв (датчиків, кнопок управління та ін.) у момент їх читання, тобто образ входів.

Таблиця виходів процесу. Область пам'яті, в якій фіксуються дані, що призначені для виведення на вихідні пристрої на початку робочого циклу, тобто образ виходів.

Таймери. Таймери – це складові частини системної пам'яті CPU. Уміст "таймерних осередків" обновляється операційною системою автоматично асинхронно стосовно прикладної програми. За допомогою команд STEP 7 визначається точна функція таймерного осередку й ініціюється її обробка, наприклад, запуск таймера.

Термінатор. Термінатор – це опір, що замикає кабель передачі даних щоб уникнути відбиття.

Утиліта STEP 7. Утиліта STEP 7 – це інструмент STEP 7, пристосований до певного завдання.

Флеш-EPROM. див. FEPROM.

Функціональний блок. Функціональний блок (FB) – це, відповідно до IEC 1131–3, кодовий блок зі статичними даними. FB надає можливість передачі параметрів у прикладній програмі. Завдяки цьому функціональні блоки придатні для програмування часто повторюваних складних функцій, наприклад, регулювання, завдання режиму роботи.

Функція. Функція (FC) - це, відповідно до IEC 1131–3, кодовий блок без статичних даних. Функція надає можливість передачі параметрів у прикладній програмі. Завдяки цьому функції придатні для програмування часто повторюваних складних функцій, наприклад, розрахунків.

Центральний процесорний модуль. Центральний модуль системи автоматизації S7 з керуючим і арифметичним пристроєм, пам'яттю, операційною системою й інтерфейсом для пристрою програмування.

Час реакції. Час реакції – це середній час, що проходить між зміною входу й відповідною зміною виходу.

Час циклу. Час циклу – це час, необхідний CPU для однократної обробки програми.

Швидкість передачі. Швидкість при передачі даних (біт/с)

Шина. Шина – це засіб передачі, що з'єднує між собою декількох абонентів. Передача даних може відбуватися послідовно або паралельно через електричні провідники або світловоди.

Шинний сегмент. Шинний сегмент – це замкнута ділянка послідовної системи шин. Шинні сегменти з'єднуються один з одним повторювачами.

ANLAUF. Робочий режим ANLAUF (ЗАПУСК) виконується при переході з робочого режиму STOP у робочий режим RUN. Може бути ініційований перемикачем режимів роботи, або після включення напруги мережі, або командою із пристрою програмування. В S7-300 при цьому виконується новий пуск.

CP. див. Комунікаційний процесор.

DP-master. див. Master-пристрій DP.

DP-Slave. див. Slave-пристрій DP.

ET200. Система децентралізованої периферії ET 200 із протоколом PROFIBUS-DP.

FB. див. Функціональний блок.
FC. див. Функція.
FEPROM (Flash-EPROM). FEPROM мають властивості зберігати дані при відключенні живлення, стираючі (електрично) EEPROM, однак стираються істотно швидше (FEPROM = Flash Erasable Programmable Read Only Memory). Вони використовуються на платах пам'яті.
FREEZE. Це команда керування Master-пристрою DP групі Slave-пристроїв DP. Після одержання команди керування FREEZE Slave-пристрій DP заморожує поточний стан входів і передає їх циклічно на DP-Master. Новий цикл виконання передачі командою керування FREEZE можливий тільки тоді, коли DP-Master пошле команду керування UNFREEZE.
Master. Активний абонент, якщо він має маркер, може посилати дані іншим абонентам і вимагати даних від інших абонентів.
Master-пристрій DP. Основна (провідна) станція.
Memory Card. див. Плата пам'яті.
MPI. Багатоточечний інтерфейс (MPI) – це інтерфейс пристрою програмування SIMATIC S7. Він дає можливість одночасної роботи декількох абонентів (пристроїв програмування, текстових дисплеїв, панелей оператора) з одним або декількома центральними модулями, кожний абонент ідентифікується однозначною адресою – адресою MPI.
OB. див. Організаційний блок.
PG. див. Пристрій програмування.
RAM. Random Access Memory – це напівпровідникова пам'ять із вільним доступом на запис і читання.
SFB. див. Системний функціональний блок.
SFC. див. Системна функція.
Slave. Slave (підлегла, ведена станція) може обмінюватися даними з Master-пристроєм тільки по запиту останнього.
Slave-пристрій DP. Підлегла (ведена) станція, що приводиться в дію на шині PROFIBUS за допомогою протоколу PROFIBUS-DP.

Навчальний посібник

СЕРДЮК Олександр Олександрович,

РАЗЖИВІН Олексій Валерійович

РОЗПОДІЛЕНІ СИСТЕМ НА БАЗІ ПЛК

Навчальний посібник

Редагування С. П. Шнурік

Комп'ютерне верстання О. С. Орда

40 /2010. Підп. до друку . Формат 60 х 84/16.
Папір офсетний. Ум. друк. арк. . Обл.-вид. арк. .
Тираж прим. Зам. №

Донбаська державна машинобудівна академія
84313, м. Краматорськ, вул. Шкадінова, 72.
Свідоцтво суб'єкта видавничої справи

серія ДК №1633 від 24.12.2003