МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ
УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

І.Ш. Невлюдов, С.П. Новоселов, О.В. Сичова

ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ ПРОМИСЛОВИХ КОНТРОЛЕРІВ В ІНТЕГРОВАНОМУ СЕРЕДОВИЩІ CODESYS

НАВЧАЛЬНИЙ ПОСІБНИК

Харків 2019

УДК 681.51
Н40

Рекомендовано Вченою радою
Харківського національного університету радіоелектроніки
(протокол № 6 від 31.05 2019 року)

Невлюдов І.Ш. Технологія програмування промислових контролерів в інтегрованому середовищі CODESYS: Навчальний посібник / І.Ш. Невлюдов, С.П. Новоселов, О.В. Сичова. – Харків: ХНУРЕ, 2019 . – 264 с.

ISBN 978-966-659-265-4
DOI: 10.30837/978-966-659-265-4

У навчальному посібнику подані відомості про сучасні технології організації виробництва, зокрема розглянуті перспективи переходу до Індустрії 4.0 у вітчизняних виробництвах. Розглянуті особливості побудови сучасних ПЛК. Дано практичні рекомендації щодо створення складних програмно-апаратних комплексів керування ТП.

Навчальний посібник дає можливість студентам ознайомитись із сучасними інструментами підготовки технологічних програм керування ТП, вивчити перспективні мови програмування промислових контролерів, такі як ST, LD, FBD, SFC.

Вивчення принципів програмування промислових контролерів, зокрема, структури і принципу дії ПЛК, які застосовуються для функціонування АСУ ТП, технологічних мов програмування є необхідним у процесі підготовки фахівців спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології» та 172 «Телекомунікації та радіотехніка», практична діяльність яких орієнтована на виробництво, обслуговування, автоматизацію та створення комп'ютерно-інтегрованих технологій і промислової автоматики.

Навчальний посібник призначено для підготовки студентів денної та заочної форм навчання спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології», 172 «Телекомунікації та радіотехніка». Може бути корисний аспірантам і фахівцям у промисловості, робота яких пов'язана з розробкою та організацією виробництв галузі радіоелектронного приладобудування.

УДК 681.51
Н40

Рецензенти:

В.М. Свищ, д-р техн. наук, професор, радник з науки Генерального директора, Державне науково-виробниче об'єднання «Комунар»;

Л.І. Нефьодов, д-р техн. наук, професор, завідувач кафедри автоматизації та комп'ютерно-інтегрованих технологій, Харківський національний автомобільно-дорожний університет;

Г.В. Карпов, канд. техн. наук, доцент, керівник групи головного технолога ТОВ «НВО Вертикаль».

ISBN 978-966-659-265-4
DOI: 10.30837/978-966-659-265-4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАК, СКОРОЧЕНЬ І ТЕРМІНІВ

АРМ – автоматизоване робоче місце;

АСУ – автоматизована система управління;

АСУТП – автоматизована система управління технологічними процесами;

АЦП – аналого-цифровий перетворювач;

ВМ – виконавчий механізм;

ВП – виконавчий пристрій;

ВПр – вимірювальний перетворювач;

ЗП – захоплювальний пристрій;

ІвП – інтелектуальний вимірювальний перетворювач;

ІВП – інтелектуальні виконуючі пристрої;

ІД – інтелектуальні датчики;

ІЗ – інформаційне забезпечення (бази даних);

Інд – індикатор;

КЗ – комунікаційне забезпечення (послідовний порт і ПЗ);

МП – мікропроцесор;

ОЗП – оперативний запам’ятовувальний пристрій;

ПЗ – програмне забезпечення;

ПЗП – постійний запам’ятовувальний пристрій;

ПЗв – програмне забезпечення виробника;

ПЗк – програмне забезпечення користувача;

ПЛК – програмований логічний контролер;

ПК – персональний комп’ютер;

ПрК – програмований контролер;

ТЗ – технічне забезпечення (апаратна частина);

УП – програма, що управляє;

ЦАП – цифро-аналоговий перетворювач;

ШІМ – широтно-імпульсна модуляція.

Бітовий канал – елемент структурування каналу. Створюється автоматично. Тип завжди BOOL.

Ім’я параметра – унікальний (у даній конфігурації) набір символів, однозначно визначає доступ до параметра в контролері.

Індекс параметра – числове значення, що відрізняє параметри однотипних елементів з однаковими іменами.

Канал конфігурації – елемент конфігурації, що описує змінну в пам’яті вводу/виводу, через яку модуль, що містить канал, взаємодіє з програмою ПЛК. Мінлива каналу може бути типу BOOL, BYTE, WORD, DWORD, FLOAT, STRING. Канал не має параметрів.

Конфігуратор ПЛК (PLC-Configuration – редактор CODESYS, в якому визначається склад апаратних засобів і проводиться налаштування певних параметрів вводу/виводу.

Конфігурація – сукупність модулів, каналів і значень їх параметрів, що визначає структуру області вводу/виводу і функціонування ПЛК.

Модуль конфігурації (модуль) – основний елемент, що конфігурується, який підключається в PLC-Configuration. Модуль відповідає одиниці апаратних/програмних засобів, з яким працює контролер.

Назва параметра – словесний опис параметра, що відображає його суть.

Параметр – атрибут каналу або модуля. Значення параметра встановлюється інтерактивно до завантаження проекту в контролер. Воно передається в ПЛК і впливає на його роботу.

Програмний модуль – блок програми приладу, призначений для виконання конкретної дії.

Формат даних – тип значень параметрів. Розрізняють такі формати: ціле число, число з плаваючою точкою тощо.

Цільовий файл (Target file) – набір файлів, які поставляються виробником ПЛК та описують апаратні і програмні особливості конкретного ПЛК, що дозволяє середовищу розробки коректно взаємодіяти з ПЛК.

ВСТУП

Зростання продуктивності праці, в тому числі шляхом її автоматизації, стає найважливішим джерелом розширення виробництва радіоелектронної апаратури. Зазначені обставини висувають нові вимоги до масштабів використання ПЛК і до технічного рівня АСУТП, до забезпечення їх надійності, точності, швидкодії, економічності, тобто до ефективності їх функціонування. Ефективне вирішення цих завдань можливе лише за умови знання сучасних технічних і програмно-технічних засобів автоматизації, їх функціональних можливостей і техніко-експлуатаційних показників.

Отже, вивчення принципів програмування промислових контролерів, зокрема, структури і принципу дії ПЛК, які застосовується для функціонування АСУ ТП, технологічних мов програмування є необхідним у процесі підготовки фахівців спеціальностей 151 «Автоматизація та комп'ютерно-інтегровані технології» і 172 «Телекомунікації та радіотехніка», практична діяльність яких орієнтована на виробництво, обслуговування, автоматизацію та створення комп'ютерно-інтегрованих технологій і промислової автоматики, що підтверджує актуальність написання даного навчального посібника.

Навчальний посібник дає можливість студентам ознайомитись з сучасними інструментами підготовки технологічних програм керування ТП, вивчити перспективні мови програмування промислових контролерів, такі як ST, LD, FBD, SFC.

У навчальному посібнику подано відомості про сучасні технології організації виробництва, зокрема розглянуто перспективи переходу до Індустрії 4.0 у вітчизняних виробництвах. Розглянуті особливості побудови сучасних ПЛК. Подано практичні рекомендації щодо створення складних програмно-апаратних комплексів керування ТП.

Навчальний посібник складається з шести розділів. Матеріал подано у зручній і доступній для вивчення студентами формі з великою кількістю ілюстрацій та практичними прикладами.

Перший розділ навчального посібника присвячено висвітленню ролі промислових контролерів у четвертій промисловій революції. Розкрито питання застосування комунікаційних технологій і цифровізації на інтелектуальному заводі. Велика увага приділяється особливостям

організації виробничих процесів і контролю стану технологічного процесу з використанням ПЛК та інтелектуальних датчиків, як невід'ємної складової частини Індустрії 4.0.

У другому розділі розглядаються особливості внутрішньої організації ПЛК, типи контролерів і принцип роботи даного пристрою. Надається інформація про інтеграцію ПЛК у виробничий процес та систему управління підприємством.

У третьому розділі розглядаються характеристики програмованих логічних контролерів і програмованих реле ОВЕН. Подано відомості про найбільш масові серії ПЛК100 та ПЛК150.

У четвертому розділі описані особливості програмування промислових контролерів та інструменти для виконання цієї задачі.

П'ятий розділ присвячено технології програмування ПЛК. Розглянуті основні технологічні мови програмування, які відповідають МЕК-61131 – ST, LD, FBD, SFC.

У шостому розділі наводяться приклади вирішення практичних задач. Розглянуті питання використання різних версій інтегрованого середовища розробника CODESYS. Розглянута робота з дискретними виходами на прикладі вирішення задач управління світловою сигнальною колоною.

Зміст навчального посібника «Технологія програмування промислових контролерів в інтегрованому середовищі CODESYS» відповідає програмі підготовки бакалаврів, що навчаються за спеціальностями 151 «Автоматизація та комп'ютерно-інтегровані технології» і 172 «Телекомунікації та радіотехніка».

1 РОЛЬ ПРОМИСЛОВИХ КОНТРОЛЕРІВ У ЧЕТВЕРТІЙ ПРОМИСЛОВІЙ РЕВОЛЮЦІЇ

1.1 Головні характеристики Індустрії 4.0

Четверта промислова революція або перехід до «Індустрії 4.0» передбачає максимально широке застосування інформаційних технологій на виробництві [1].

Індустрія 4.0 – це більше, ніж бачення майбутнього. Інтелектуальна мережа являє собою величезні можливості для промисловості. Гнучке виробництво може допомогти оптимально використовувати можливості промислового обладнання.

Перша промислова революція почалася з винаходом парового двигуна в кінці XVIII століття і переходом від ручної праці до механічного виробництва (рис. 1.1).

Рисунок 1.1 – Стадії розвитку промисловості

Друга промислова революція настала приблизно через 100 років із застосуванням конвеєрного виробництва з електроприводом. З першої третини XX-го століття воно зробило можливим економічно ефективне серійне виробництво.

Електронні системи управління, інформаційні технології, електроніка, роботи і розширення використання датчиків роблять можливим подальшу

автоматизацію виробничих, складальних і логістичних процесів та перехід до третьої промислової революції.

Важливо розрізняти терміни «Четверта промислова революція» і «Індустрія 4.0» (Industry 4.0) [1]. Перший визначає впровадження нових технологій 4.0 і їх вплив на всю економіку і соціальну сферу – розумні міста, будинки, сільське господарство, енергетика, інфраструктурні об'єкти, фінанси, державне управління, охорона здоров'я, освіта тощо. Індустрія 4.0 належить, насамперед, до сфери виробництва матеріальних продуктів (рис. 1.2).

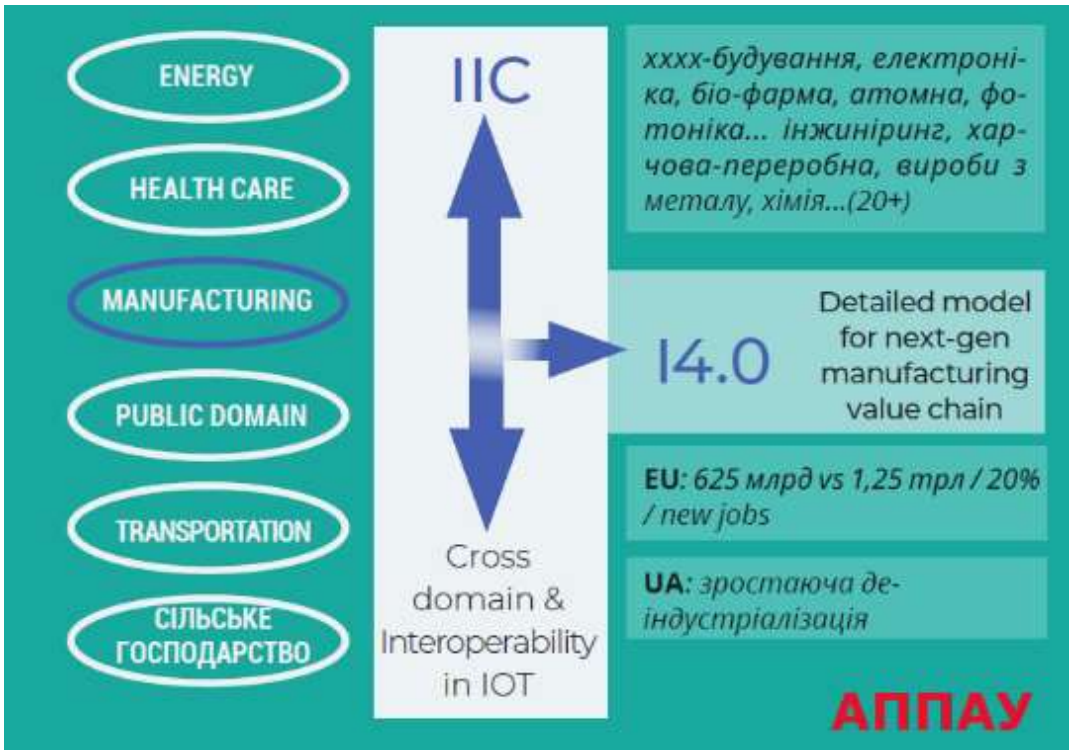


Рисунок 1.2 – Різниця між термінами
«Четверта промислова революція» й «Індустрія 4.0»

Водночас невірно ізолювати Індустрію 4.0 від інших сфер економіки. Delloite вказує, що Розумні Фабрики є дотичними до багатьох сфер, пов'язаних з промисловими виробництвами, і утворюють цілісну технологічну екосистему (рис. 1.3).

Більшість експертів в області світової Індустрії 4.0 єдині в розумінні трьох загальних характеристик (рис. 1.4). Цей фреймворк також показує, що 4.0 певною мірою є еволюцією (продовженням 3.0).

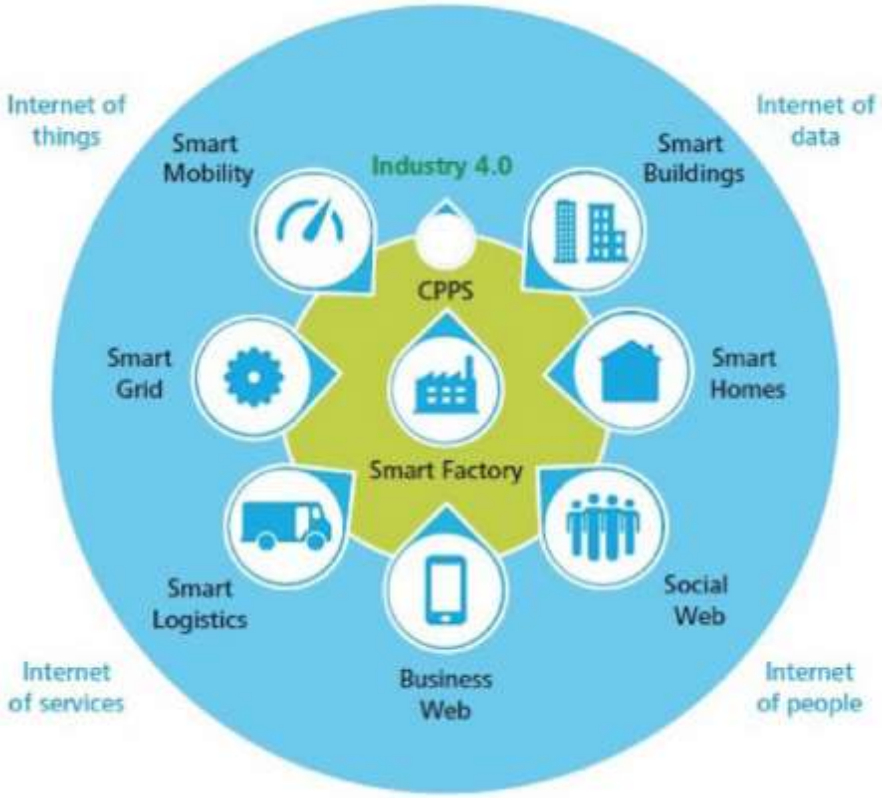


Рисунок 1.3 – Цілісна технологічна екосистема

Інші важливі характеристики, які не наведені на рис. 1.4 і які в результаті роблять фабрики і заводи «розумними», це:

– інтероперабельність: кіберфізичні системи дозволяють людям і розумному обладнанню (фабрикам) ефективніше поєднуватися один з одним;

– віртуалізація: в 4.0 можна створювати віртуальні копії розумних фізичних об’єктів (масштабованих від окремих пристроїв або машин до цілих заводів) і, відповідно, запускати різні механізми симуляцій, моделювання, а також оцінки реального стану;

– децентралізація: на відміну від високоцентралізованих підходів, у 3.0 і 4.0 кожна кіберфізична підсистема може робити власні рішення і взаємодіяти з іншим найбільш оптимальним способом;

– реальний час: всі дані та їх аналітику можливо отримувати в реальному часі;

– орієнтація на сервіси: кількість різних сервісів, як із взаємодії пристроїв і систем між собою, так і з взаємодії з людьми та учасниками екосистеми, зростає в рази;

– модульність: гнучка адаптація розумних фабрик до зовнішніх змін так само зростає, оскільки можна легко змінювати або розширювати окремі модулі систем керування.

Рисунок 1.4 – Головні характеристики і технології 4.0

У той час, коли зазначені характеристики є абсолютно новими, зв’язок 3.0 з 4.0, наведений на рис. 1.4, є вкрай важливим для цілого ряду індустрій. Він стверджує, що не можна перестрибувати через рівень 3.0, якого до цих пір на 100% немає в більшості промислових галузей України.

Велика частина впровадження технології 4.0 – особливо це стосується великих даних і штучного інтелекту – базується на тому, що ці дані вже оцифровані на польовому рівні. Тобто на підприємствах вже налагоджений облік і встановлені датчики.

1.2 Комунікаційні технології та цифровізація на інтелектуальному заводі

Комунікація на виробництві в рамках технології 4.0 займає передове місце. В інтелектуальному виробництві обладнання, контролери і датчики взаємодіють як один з одним, так і безпосередньо з Ethernet або у хмарі. Закрита система стає відкритою. Але змінюється не тільки кількість інформації, що обробляється безпосередньо на місці. Підвищується і якість до абсолютно нового рівня.

Інформація про стан виробничого обладнання та пов'язане з ним прогнозування про можливі простої виробництва за допомогою інноваційних систем зворотного зв'язку подає тільки один із прикладів. Це стало можливим завдяки швидкому збільшенню обчислювальної потужності, яку також можна вже використовувати децентралізовано в периферії, на краю мережі або в основах виробництва.

Результатом цього стає більш гнучке й динамічне виробництво, яке в будь-який час може індивідуально і швидко реагувати на вимоги клієнтів.

Поширена досі форма комунікації між датчиками і блоками управління (ПЛК), і розташованими вище рівнями управління технологічними процесами, виробництвом і підприємством, є закритою системою (рис. 1.5).

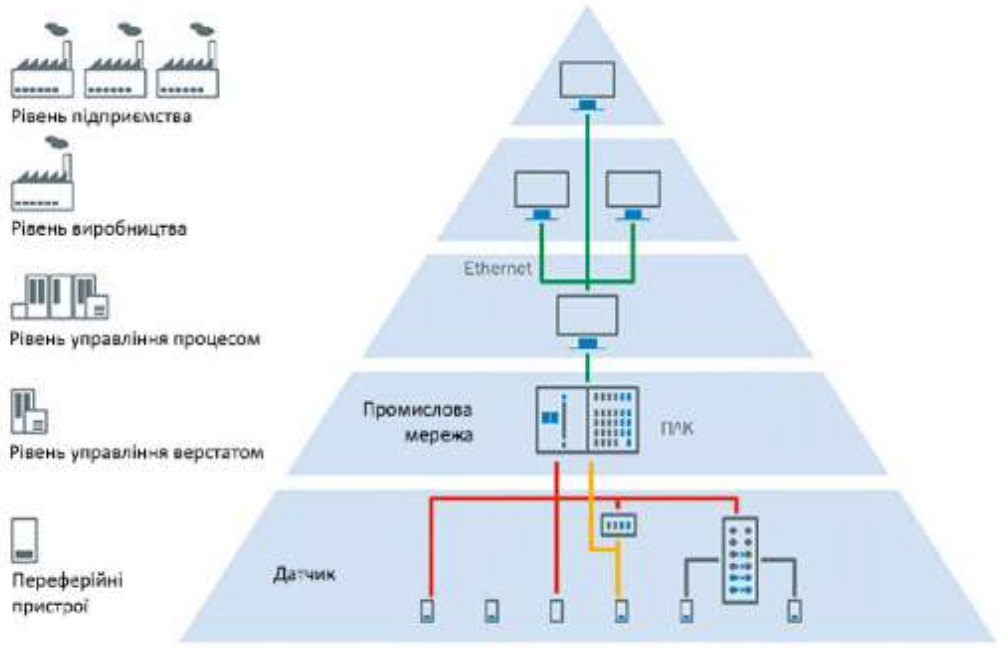


Рисунок 1.5 – Рівні комунікації на інтелектуальному заводі сьогодні

При цьому дані передаються від периферійних пристроїв, тобто датчиків і виконавчих механізмів, у програмований логічний контролер (ПЛК).

Децентралізована обчислювальна потужність в ідеології Індустрії 4.0 переробляє дані в інформацію безпосередньо в датчику.

Рішення приймаються децентралізовано. Релевантна для технологічного процесу, виробництва і підприємства інформація направляється безпосередньо в Ethernet і хмарний сервіс.

На рис. 1.6 показана схема децентралізованого управління виробництвом.

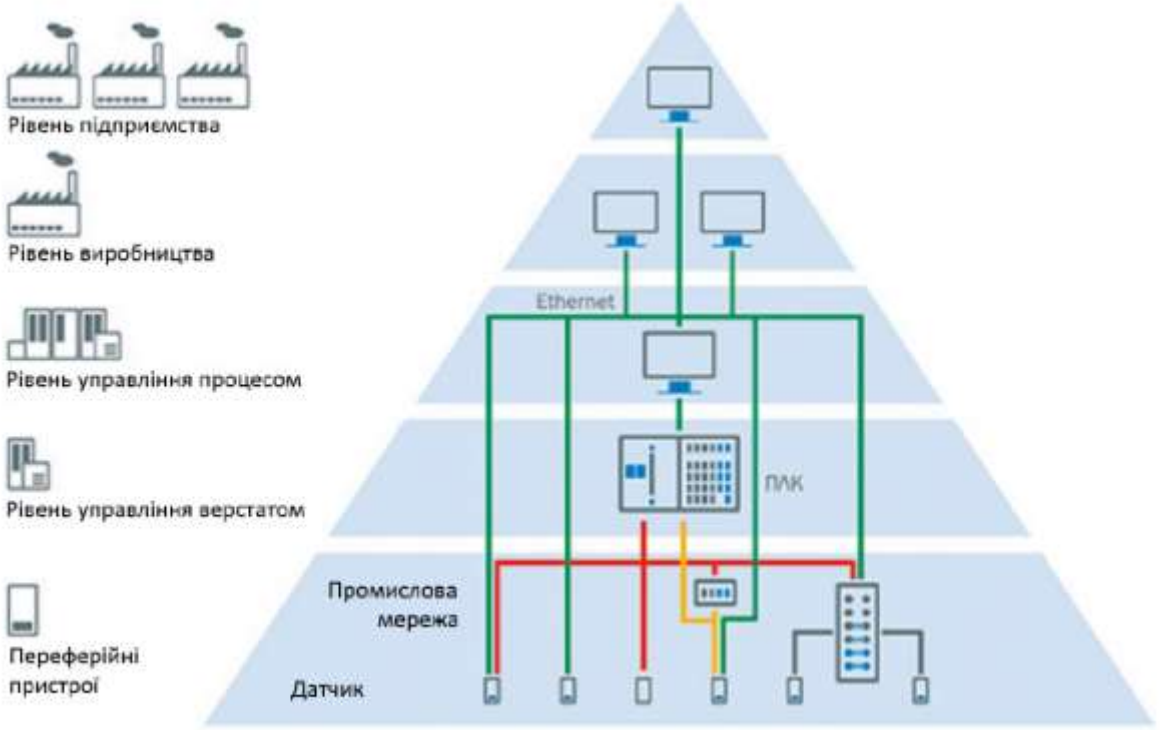


Рисунок 1.6 – Схема децентралізованого управління виробництвом

Подальший розвиток цифрового виробництва передбачає перенесення обчислень у хмарний сервіс, як показано на рис. 1.7.

Таким чином, хмарний сервіс стає все більш важливим для управління загальними процесами. Але основна обчислювальна потужність все більше переміщується на периферію. Датчики готують зібрані дані для передачі інформації, яка потім обробляється в Ethernet або у хмарному сервісі для подальшого процесу.

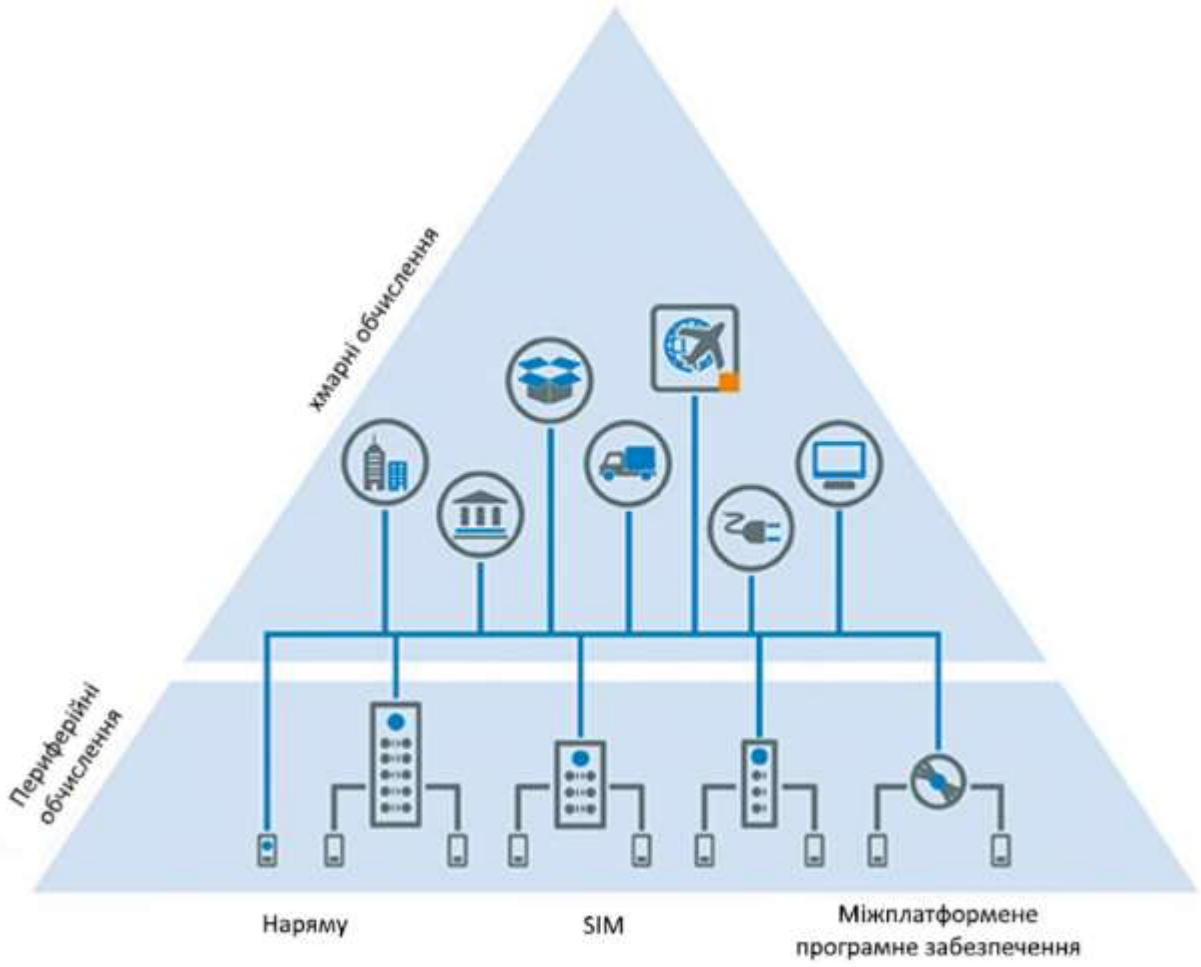


Рисунок 1.7 – Мережна інформація на цифровому виробництві

1.3 Виробничі процеси та контроль стану технологічного процесу

1.3.1 Прозорі виробничі та логістичні процеси

Позитивні ефекти об’єднання в мережу в рамках Індустрії 4.0 описуються як послідовне прозоре виробництво на всьому виробничому процесі [2].

За успішного об’єднання в мережу всіх компонентів такий вид прозорого виробництва забезпечує огляд усіх виробничих і логістичних процесів за усім ланцюжком поставок, аж до оформлення замовлень і доставки продукції замовникам. Це зменшує споживання матеріалів і ресурсів. Додатково цілісно оптимізуються виробничі і постачальні мережі.

Інтелектуальні рішення з відстеження та контролю за проходженням генерують дані та інформацію, які в мережному технологічному ланцюжку забезпечують повне виявлення, ідентифікацію та відстеження продукту і матеріалу.

Технічні можливості щодо реалізації рішень з відстеження та контролю за проходженням різноманітні. Вибір відповідної технології ідентифікації

для найкращої ефективності зчитування і системної інтеграції варіюється залежно від вимог.

Для використання в Індустрії 4.0 рішень на розумному заводі використовуються, перш за все, RFID і програмовані камери.

Датчики по всьому виробничому ланцюжку на основі носіїв даних безпосередньо розпізнають, які кроки зі збірки необхідно ініціювати, і забезпечують постійну прозорість аж до відвантаження.

Сьогодні інтелектуальні датчики означають не тільки збір даних про реальність, а й відповідну підготовку інформації вже в датчику.

Так, наприклад, за допомогою гнучкого формату виведення за допомогою налаштування і зв'язування логічних умов виведення даних можна точно привести у відповідність до вимог.

На цьому фоні кожна технологія матиме своє повноваження в майбутньому: RFID, наприклад, дозволяє зчитування і запис а, отже, багаторазове використання носіїв даних, крім того, немає необхідності в прямому «візуальному контакті».

З іншого боку, зчитувачі коду на основі камери також зчитують 2D-коди і звичайний текст. Збережені зображення можуть бути заархівовані та проаналізовані.

1.3.2 Динамічне і гнучке виробництво

Прогресивна автоматизація сприяє гнучкому виробництву і дуже невеликим обсягам партій. Побажання клієнтів стають головними. Розмір партії в одну штуку більше не є дорогою складністю, а повсякденним успішним бізнесом.

Невеликі обсяги партій та індивідуалізовані продукти масового виробництва є девізами Індустрії 4.0. Для її виправдання машина або устаткування повинні вміти обходитися зі змінною подачею продукту і бути адаптованими до різних форматів. Тільки тоді можна гнучко й ефективно виробляти товари відповідно до побажань замовника, навіть розміром партії в одну штуку, і в адаптації до коливань попиту.

Інтелектуальні датчики забезпечують нову якість гнучкості. Вони створюють дані з виробництва в режимі реального часу. Датчики підтримують і спрощують обробку даних за рахунок того, що результати вимірювань обробляються за допомогою інтелектуальних функцій відразу на місці і

передаються у вигляді підготовленої інформації, що містить тільки корисні дані.

Зі збільшенням міри автоматизації устаткування також зростають і завдання окремих компонентів: у різних галузях вже використовуються, наприклад, фотоелектричні датчики з гнучкими настройками і діагностичними функціями.

Наприклад, індуктивні датчики наближення з IO-Link вирішують складні завдання безпосередньо в самому датчику. Датчики контрасту, датчики рівня та електронні реле тиску обмінюються даними налаштувань параметрів за допомогою інтегрованих інтерфейсів IO-Link.

Вимірювальні високоавтоматизовані світлові завіси знижують витрати на проводку у виробничих умовах і забезпечують доступ до діагностики та переналаштування формату.

Енкодери з EtherNet/IP™ мають як активний веб-сервер, так і функціональні блоки для інтеграції в польову шину.

Компактні датчики 2D-LiDAR (також лазерні 2D-сканери) надійно виявляють об’єкти під час контролю території.

Стандарт зв’язку з датчиками IO-Link дає можливість легкої заміни кінцевих пристроїв [3]. Ведучій пристрій завантажує параметри у встановлений як заміна пристрій, що значно спрощує технічне обслуговування – не потрібне додаткове налаштування. Серед доступних різновидів датчиків IO-Link: виконавчі датчики, вимірювальні датчики, фотоелектричні датчики, RFID-датчики, лазерні датчики відстані, датчики кольору. Стандарт IO-Link регулюється відкритим консорціумом виробників.

Комунікації IO-Link на фізичному рівні реалізуються за допомогою звичайного провідникового кабелю і стандартних конекторів, таких як M12, M8 і M5. Система IO-Link складається з пристроїв IO-Link, включаючи датчики і приводи, а також провідного пристрою. Оскільки IO-Link має архітектуру типу «точка-точка», тільки один пристрій може бути під’єднаний до кожного порту ведучого пристрою.

Ведучі пристрої IO-Link можуть бути різними, включаючи карту в стійці ПЛК, або окремий пристрій з інтерфейсами до стандартних польових шин, таких, як PROFIBUS або PROFINET.

Протокол IO-Link підтримує різні типи даних – циклічні (процесні), ациклічні, сервісні, події. Пристрій IO-Link посилає дані тільки після запиту від ведучого пристрою IO-Link. Ациклічні дані і події безпосередньо запитуються ведучим пристроєм, а циклічні дані відправляються після телеграми IDLE від ведучого пристрою.

Об’єднання в мережу всіх задіяних пристроїв і безпечний, децентралізований обмін даними призводить до широкого спектру варіантів застосування. Їх можна запропонувати як через хмару, так і через програмовані логічні контролери на рівні машин і систем.

Покращені обчислювальні можливості також змінюють візуальні здібності рішень на основі камери для забезпечення якості та управління виробництвом на основі датчиків.

Забезпечення якості є необхідною умовою для стійкого бізнесу і стабільного доходу. Воно включає в себе як управління матеріалами, так і функціональний контроль, контроль за машиною і виробництвом. Це дозволяє знизити рівень складських запасів і скоротити час обробки.

Сенсорні рішення для контролю за технологічним процесом і забезпечення якості підвищують гнучкість завдяки автономному коригуванню зі зміною якості і зміною продукту. Вони забезпечують ресурсоефективність, скорочення виробничого браку і високу пропускну здатність.

1.3.3 Взаємодія людини і робота на мережному заводі

У розумному підприємстві (Smart Factory) люди і роботи зближуються ще дужче [4]. У розумінні сучасного розподілу праці датчики підтримують роботів у їх роботі – даючи їм очі для виконання завдань у промислових умовах. Більш сильна взаємодія між людиною і машиною вимагає абсолютно надійних і гнучких рішень для забезпечення безпеки. Кооперація і співіснування мають стати справжньою співпрацею.

Розумне підприємство робить ставку на тісну взаємодію роботів з людьми. У цих взаємодіючих сценаріях сила, швидкість, траєкторії руху робота і заготовки становлять небезпеку для робітника. Ці небезпеки мають бути обмежені застосуванням спеціальних запобіжних заходів.

Уже сьогодні датчики безпеки дозволяють реалізувати тонку адаптацію під поточний автоматизований процес.

Інтелектуальні алгоритми дозволяють, наприклад, перехід від техніки безпеки з цифровою технологією перемикання до безперервної машинної реакції залежно від руху людини.

Отже, наближення робочого більше не призводить до загального відключення машини, а скоріше до розумного зниження робочої швидкості або коригування напрямків руху.

«Safe Motion» нині є найкращою технологією. Вона в будь-який час забезпечує безпеку людей і, незважаючи на це, немає необхідності переривати виробництво. Внаслідок цього значно скорочується час простою і кількість помилкових відключень, часи циклів стають коротшими, а ефективність та експлуатаційна готовність машин і систем збільшуються.

Якщо людина і машина співпрацюють щільно й надійно, функціональна безпека в сучасних виробничих системах є важливим кроком до більшої гнучкості.

На шляху до повноцінного співробітництва – люди і роботи ділять між собою один і той самий робочий простір і працюють у ньому водночас – існують також рішення для співіснування або кооперації.

За одночасний захист великої кількості небезпечних об’єктів відповідає, наприклад, програмований контролер безпеки з супровідним програмним забезпеченням, також у поєднанні з безпечним каскадом датчиків.

Небезпечні зони, входи і небезпечні об’єкти абсолютно надійно захищає нове покоління лазерних сканерів безпеки. Високопродуктивні світлові завіси безпеки підходять як компактні альтернативи вибіркового відключення без додаткових датчиків, а також для забезпечення високодоступного захисту небезпечних об’єктів і зон.

1.4 Інтелектуальні датчики як невід’ємна складова частина Індустрії 4.0

1.4.1 Збір інформації для автоматизації промислових процесів

Цифровізація та об’єднання обладнання в мережу – головна риса четвертої промислової революції. Нові технології роблять можливим злиття фізичного і віртуального світу в області виробництва і логістики в кіберфізичні системи (CPS). З 2011 року ця розробка об’єднується під терміном «Індустрія 4.0». Машини можуть автономно обмінюватися одна з одною даними і, таким чином, оптимізувати перебіг процесів. При цьому Індустрія 4.0 чітко посилається на створення мереж у промисловому секторі.

Сенсорна техніка створює умови для прозорих процесів в Індустрії 4.0. При цьому датчик утворює основу всіх наступних застосувань.

На відміну від класичних, немережних датчиків, датчики Індустрії 4.0 надають більше, ніж просто дані вимірювань. Інтегрована, децентралізована обчислювальна потужність і гнучка програмованість є ключовими

характеристиками, які роблять виробництво більш гнучким, динамічним і ефективним.

Технологічний прогрес завжди є передумовою для змін у промисловості. Основою для динамічних, оптимізованих у реальному часі і самоорганізованих промислових процесів є отримання інформації та її подальша обробка. У них датчики виконують роль постачальників даних для інтелектуального виробництва. Сенсорна техніка є обов'язковою умовою для успішної реалізації Індустрії 4.0 (рис. 1.8).

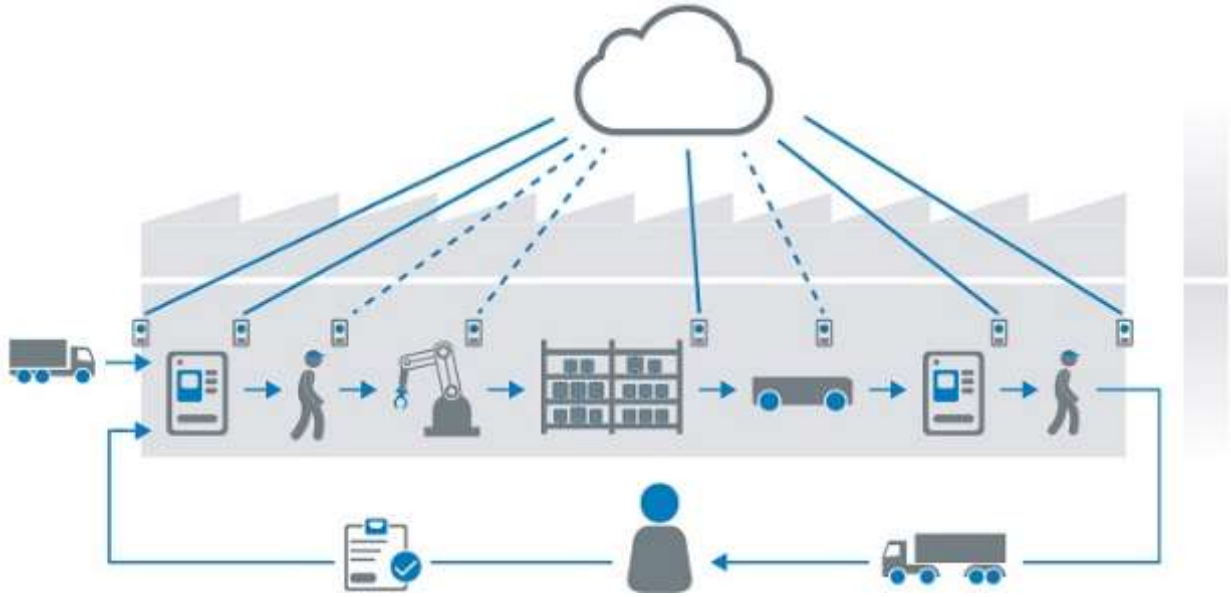


Рисунок 1.8 – Інтелектуальні датчики на мережевому заводі

Мережний завод є обов'язковою умовою для Індустрії 4.0. Кожен сенсор, кожна машина і всі залучені люди в будь-який час можуть спілкуватися між собою. Але цей обмін інформацією не закінчується у заводських воріт. Взаємодія між периферією і хмарним сервісом дозволяє управляти виробництвом і даними навіть ззовні.

Таким чином, це інтенсивне співробітництво між технологією і людиною робить процес більш прозорим, продуктивним і прибутковим.

1.4.2 Діагностична функція інтелектуальних датчиків

Діагностика дозволяє завчасно розпізнати відхилення від технологічних процесів та уникнути незапланованих простоїв обладнання.

Smart Sensors дають знати про себе навіть самостійно з порушенням надійної експлуатації. Профілактичне обслуговування дозволяє складати гнучкі плани техобслуговування, орієнтовані на поточні потреби, і тим самим знижувати витрати на сервіс.

Якщо проблеми все ж таки виникли, їх причину легко визначити завдяки широким можливостям візуалізації – це дозволяє уникнути незапланованих простоїв обладнання.

Розглянемо види діагностики, які доступні під час використання інтелектуальних датчиків.

Самоконтроль датчика в ході налаштування та експлуатації

Даний вид діагностики дозволяє досягти таких переваг під час експлуатації:

– завчасне виявлення перешкод запобігає незапланованому простою обладнання;

– превентивне дистанційне техобслуговування дозволяє точно планувати сервісні роботи, що економить витрати й час;

– відпадає потреба у тривалому пошуку несправностей завдяки точній локалізації повідомлення про необхідність техобслуговування.

Датчики Smart Sensor мають вбудовані функції діагностики, наприклад, самоконтроль, розпізнавання величини перешкод і автоматичне підстроювання порога перемикання. Це забезпечує можливість профілактичного техобслуговування, що підвищує експлуатаційну готовність обладнання і скорочує час простоїв.

Контроль важливих технологічних параметрів

Ця функція забезпечує високу якість продукції завдяки контролю технологічних параметрів. Наприклад, завдяки вбудованій функції діагностики датчик наближення вже в процесі формування контролює результат окремого етапу технологічного процесу. Таким чином, регульовані точки перемикання можуть безпосередньо сигналізувати про перевищення величини перешкод. Це знижує відбраковування виробів і спрощує пошук несправностей, тому що дозволяє швидко локалізувати причину несправності.

Статистичний моніторинг сигналів дозволяє розпізнавати тенденції і заздалегідь планувати технічне обслуговування та налагодження формувальних інструментів, виходячи з потреб технологічного процесу.

Візуалізація сигналів і параметрів розпізнавання для детального аналізу технологічних процесів і процесу виявлення

Як правило, автоматизовані комплексні технологічні процеси вимагають використання в обладнанні великої кількості датчиків. Цілеспрямоване виведення інформації інтегрованими в обладнання датчиками категорії Smart

Sensor, наприклад, візуалізація сигналів і параметрів виявлення, допомагає кінцевим споживачам оптимізувати процеси машинного виробництва, швидко розпізнавати й усувати несправності.

Візуалізації виміряних величин і отриманих сигналів має ряд переваг:

– підвищена прозорість у ході виробництва для кращого розуміння процесів;

– швидке усунення проблем у разі несправностей;

– візуалізація змін технологічних процесів.

1.4.3 Розширене сприйняття інтелектуальних датчиків

Розширене сприйняття (Enhanced Sensing) забезпечує надійні результати виявлення і вимірювання, що безпосередньо позитивно впливає на експлуатаційну готовність обладнання.

Інтелектуальні датчики автоматично визначають несправності, що виникають під час роботи та активно протидіють їм. Наприклад, вже під час установки вони активно підтримують монтажника в пошуку оптимальної робочої точки.

Багато Smart Sensors пропонують різні режими роботи, аж до повністю ручного регулювання параметрів вимірювання та виявлення таким чином, щоб за необхідності їх можна було динамічно адаптувати під поточні завдання.

Функція розширеної можливості налаштування дозволяє досягти таких результатів:

– надійне й точне виявлення об’єктів для найкращих результатів вимірювань;

– швидка індивідуальна адаптація для кожного конкретного випадку застосування, аж до ручного режиму роботи;

– стабільність виробничих процесів;

– захист від несанкціонованих маніпуляцій завдяки цілеспрямованій деактивації елементів управління.

Наприклад, типографські позначки слугують для правильного позиціонування аркушів в автоматизованих виробничих процесах. Завдяки динамічному підстроюванню порога перемикання інтелектуальний датчик контрасту надійно й швидко розпізнає типографські позначки.

Як і всі датчики категорії Smart Sensor, він налаштовується через IO-Link. Завдяки цьому інші процеси запускаються в потрібному місці робочого процесу.

Функція попереднього задання режиму експлуатації для складних завдань дозволяє досягти таких результатів:

– швидкий і простий шлях введення в експлуатацію;

– надійне виявлення об’єкта навіть у складних умовах.

На рис. 1.9 показаний приклад використання інтелектуальних датчиків на харчовому виробництві.

Рисунок 1.9 – Приклад використання інтелектуальних датчиків на харчовому виробництві

Як видно з даного рисунка, під час виробництва пляшок для напоїв можливе виготовлення найрізноманітніших варіантів продукції. Вироби, що виготовляються, подаються транспортером на високій швидкості.

Для швидкого, безперебійного транспортування необхідне надійне розпізнавання виробів, які можуть значно відрізнятися за формою і типом поверхні. Висока швидкість і особливі оптичні властивості пляшок для напоїв вимагають точності виявлення. Інтелектуальні датчики категорії Smart Sensor, зокрема, фотоелектричний датчик W12G, мають встановлені профілі виявлення для оптимального та швидкого налаштування датчика на різні форми й поверхні пляшок.

Функція простої компенсації перешкод має такі переваги:

– запобігання помилкових спрацьовувань;

– надійне виявлення об’єкта навіть у складних умовах;

– стабільні й надійні сигнали від датчика.

У харчовій промисловості датчики, як правило, знаходяться безпосередньо усередині виробничого процесу і постійно піддаються значній запиленості, впливу води і вібрацій, а також інших несприятливих впливів. Отже, для надійного

функціонування датчиків, а відтак, для безпечної роботи машини або установки, необхідні регулярне очищення і техобслуговування датчиків.

Інтелектуальний датчик Smart Sensor W4S-3 Glass завдяки реалізованій функції AutoAdapt динамічно адаптує поріг перемикання до оптичних поверхонь датчика і відбивача, які поступово забруднюються. Це значно збільшує проміжок між процедурами очищення й відповідно знижує витрати на обслуговування.

Функція зворотного зв'язку під час установки дозволяє отримати такі переваги:

– швидкість юстування і введення в експлуатацію;

– запобігання небажаної експлуатації датчика на межі допустимих параметрів.

На рис. 1.10 показаний приклад застосування інтелектуальних датчиків у складі загорткових машин, для раціонального використання ресурсів у рулонах, наприклад, плівки.

Для відділення матеріалу в потрібному місці положення різання позначено на плівці друкарською міткою. Для забезпечення оптимального функціонування датчика і для уникнення браку в ході переналаштування важливе швидке й точне регулювання датчика з переходом на інший тип плівки.

Наприклад, датчик Smart Sensor KTS Prime в ході переналаштування постійно перевіряє, чи вивірене положення датчика відносно друкарських міток на плівці, і виводить дані про актуальну якість регулювання датчика на вбудований дисплей, а також додатково (через IO-Link) на панель, закріплену на установці. Це дозволяє безперервно контролювати правильність регулювання.

Рисунок 1.10 – Приклад застосування інтелектуальних датчиків у складі загорткових машин

1.4.4 Ефективний обмін даними в інтелектуальних датчиках

Ефективні процеси комунікації дозволяють здійснювати обмін даними в обох напрямках між системою управління і датчиком – для можливості штучного виробництва, гнучкості процесів і простоти обслуговування.

Інтелектуальні датчики – це пристрої збору даних та інтелектуальні пристрої аналізу даних. Завдяки вбудованому інтерфейсу IO-Link вони дозволяють навколишньому середовищу бути учасником у цих знаннях в режимі реального часу.

Smart Sensors завжди готові для прийому команд управління всіх видів. Наприклад, вони за лічені секунди можуть отримувати нові набори параметрів для гнучкого виробництва, аж до штучних партій. Навіть за несправності пристрою, за допомогою IO-Link останній використаний набір параметрів може бути автоматично переданий на запасний датчик.

Використання інтелектуальних датчиків дозволяє створювати гнучкі виробництва і працювати зі штучними партіями. Це дозволяє досягти таких переваг:

– підвищення продуктивності за рахунок зниження простоїв обладнання з переходом на інший тип продукції;

– максимально можлива гнучкість і безпека завдяки динамічному коригуванню параметрів датчика в процесі виробництва – навіть для штучних партій;

– автоматичне параметрування датчика з системи управління дозволяє запобігти помилкам у настройках, які можливі за ручного налаштування;

– підвищення варіативності обладнання дозволяє знизити витрати.

Розглянемо практичний приклад використання інтелектуальних датчиків.

Наприклад, у секторі вторинного пакування виготовлення різних варіантів продукту вимагає гнучкого використання машин та устаткування. У виробничому процесі можливі різні формати пакування для продуктів, що виробляються, характеристики яких визначаються технічним завданням. Зокрема, пакування може значно відрізнятися за формою і властивостями поверхні.

Інтелектуальний датчик Smart Sensor WTT12LC, показаний на рис. 1.11, з лінійки продуктів PowerProx дозволяє реєструвати й архівувати в системі управління набори параметрів датчика для окремих форматів пакування.

Рисунок 1.11 – Інтелектуальний датчик Smart Sensor WTT12LC

Це дозволяє здійснювати зміну форматів пакування під час роботи швидко, з повною відтворюваністю і без ручного втручання: система управління просто надсилає новий набір параметрів у Smart Sensor через IO-Link.

Швидке введення в експлуатацію і заміна пристроїв за принципом plug and play, завдяки автоматичному налаштуванню параметрів датчика, призводить до досягнення таких показників:

– спрощений порядок введення в експлуатацію завдяки параметруванню датчика з системи управління;

– швидка заміна датчика за принципом plug and play підвищує експлуатаційну готовність обладнання;

– заміну датчика може здійснити навіть непідготовлений персонал;

– віддалене параметрування датчиків, встановлених у недоступних місцях.

Наскрізна цифрова передача даних веде до таких позитивних результатів:

– підвищення якості сигналу за рахунок повністю цифрової передачі даних від датчика в систему управління;

– відпадає необхідність у традиційній аналоговій передачі параметрів (0 ... 10 В, 4 ... 20 мА);

– застосування стандартних неекранованих кабелів знижує витрати;

– висока електромагнітна сумісність (EMC).

Наприклад, на аналогову передачу сигналу оптичних датчиків внаслідок автоматизації процесів транспортування в промисловості значно впливають поля електромагнітних перешкод. На відміну від неї, цифрова передача сигналу датчиків Smart Sensor дозволяє надійно передавати дані без використання екранованих кабелів завдяки IO-Link.

Валідація пристроїв, протоколювання та електронні специфікації дозволяють значно підвищити надійність переданих даних:

– підвищена прозорість: можливість протоколювання заміни датчика і зміни параметрів;

– захист від несанкціонованих маніпуляцій завдяки цілеспрямованій деактивації елементів управління;

– надійність експлуатації: можливість запобігти введенню в експлуатацію недозволених приладів;

– автоматичне складання електронної специфікації на датчики Smart Sensors, встановлені в обладнанні на даний момент.

Функції протоколювання і генерації електронної специфікації забезпечують повну прозорість у питанні датчиків, які використовуються на виробничих лініях. Функція валідації пристроїв дозволяє гарантувати, що будуть використані лише пристрої, попередньо схвалені виробником машин.

1.4.5 Вирішення інтелектуальних завдань

Інтелектуальні датчики дозволяють здійснювати обробку даних безпосередньо в датчику. Це забезпечує прискорену передачу даних, підвищення надійності структури та економічності технологічних операцій.

Інтелектуальні датчики обробляють різноманітні сигнали виявлення і вимірювання, за необхідності поєднують їх з сигналами зовнішнього датчика і на цій основі генерують дійсно необхідну інформацію про технологічні процеси з урахуванням відповідних завдань для конкретного обладнання. Це економить час для обробки даних у системі управління, прискорює процеси машин і робить непотрібним застосування високопродуктивного додаткового обладнання, що вимагає великих витрат.

Розглянемо деякі приклади інтелектуальних завдань.

Вимірювання швидкості і довжини матеріалу

Smart Task-функція «вимірювання довжини» дозволяє точно відміряти довжину шпону на деревообробних підприємствах незалежно від проковзування транспортерної стрічки. Для цього використовується мініатюрний фотоелектричний датчик W4S-3. Вимірювальний блок, який не вимагає техобслуговування, також може використовуватися для контролю якості після відмірювання довжини і для регулювання летючого пилу з відхиленням параметрів шпону від заданих допусків.

Отже, застосування інтелектуального датчика дозволяє досягти таких переваг:

– вимірювання швидкості об’єкта незалежно від проковзування для більш точних результатів вимірювання;

– простота сортування та класифікації об’єктів виявлення на основі довжини об’єкта незалежно від швидкості подачі;

– висока гнучкість у виборі місця вимірювання;

– відсутність спотворення результату вимірювання як наслідок часу циклу системи управління.

Моніторинг об’єктів і проміжків

Під час подачі продукту в горизонтальні формувальні, фасувальні машини необхідно контролювати належний розмір або положення об’єкта, що транспортується на конвеєрі (рис. 1.12).

Завдяки датчику Smart Sensor та інтегрованій Smart Task функції об’єкти і проміжки між ними визначаються безпосередньо датчиком.

З відхиленням від заданого значення Smart Sensor надсилає відповідні сигнали в пристрій управління.

Рисунок 1.12 – Моніторинг об’єктів і проміжків

Застосування інтелектуального датчика дозволяє досягти таких переваг:

– контроль довжини об’єктів та відстаней між ними для швидкого виявлення помилок;

– простота генерації сигналів для більш високого рівня системи управління або для безпосереднього й швидкого управління вибраковуванням;

– відсутність спотворення результату вимірювання як наслідок часу циклу системи управління.

Аналіз завантаження

Інтелектуальне завдання «аналіз завантаження» датчика Smart Sensor дозволяє здійснювати надійний та економічний контроль якості меблевих плит у шкантозабивних машинах (рис. 1.13). При цьому правильність розташування дюбелів контролюється в потоці.

Рисунок 1.13 – Інтелектуальне завдання «аналіз завантаження»

Таким чином, застосування інтелектуального датчика дозволяє досягти таких переваг:

– надійне розпізнавання завантаження в процесі виробництва – навіть за різних швидкостей транспортування;

– економічність і простота.

1.5 Контрольні запитання та завдання

1. Опишіть головні характеристики Індустрії 4.0.
2. У чому різниця між термінами «Четверта промислова революція» і «Індустрія 4.0»?
3. Як використовуються комунікаційні технології та цифровізація на інтелектуальному заводі?
4. У чому полягає децентралізація управління виробництвом?
5. Як організовані виробничі процеси та контроль стану технологічного процесу в рамках Індустрії 4.0?
6. Опишіть стандарт комунікації IO-Link.
7. Як організована взаємодія людини і робота на мережному заводі?
8. Яка роль інтелектуальних датчиків в Індустрії 4.0?
9. Які задачі вирішують інтелектуальні датчики?

2 ПРОГРАМОВАНІ ЛОГІЧНІ КОНТРОЛЕРИ

2.1 Визначення ПЛК

Програмований логічний контролер (ПЛК) – це спеціалізований мікропроцесорний (МП) керуючий пристрій, пристосований до використання безпосередньо у виробничих умовах і програмований спрощеними мовами, доступними користувачам, які не мають спеціальної підготовки з програмування.

За принципом дії ПЛК є спрощеною моделлю комп'ютера, в якій програмним шляхом реалізується цифровий керуючий автомат.

ПЛК, як правило, має блочно-модульну конструкцію, що дозволяє користувачеві компонувати необхідну об'єктно-орієнтовану конфігурацію контролера шляхом доукомплектування деякого базового (керуючого) модуля необхідним набором модулів вводу/виводу з номенклатури, запропонованої виробником ПЛК.

Особливості ПЛК:

– простота спілкування з користувачем, що полягає в можливості програмування ПЛК за принциповою електричною схемою, за логічним (булевим) рівнянням і за допомогою простої алгоритмічної мови;

– пристосованість до роботи у важких виробничих умовах за рахунок застосування оптоелектронної гальванічної розв'язки входів і виходів від зовнішніх електричних ланцюгів, за рахунок пристосованості ПЛК до розширеного діапазону умов експлуатації;

– модульність конструкції, що дозволяє компонувати з обмеженої кількості уніфікованих модулів контролери різного функціонального призначення і необхідної конфігурації, що відкриває широкі перспективи в області підвищення гнучкості створюваних систем управління складними технологічними модулями;

– різке скорочення витрат на проектування за рахунок істотного зниження вартості програмування, а також за рахунок спрощення прив'язки модульної конструкції до конкретного об'єкта управління;

– значне скорочення термінів розробки і проектування систем управління обладнанням за рахунок можливості паралельного проведення робіт з їх

проектування (в тому числі програмування) і виготовлення систем управління на основі ПЛК;

– можливість коригування алгоритмів управління безпосередньо в цехових умовах (під час монтажу, пуску, випробування і в модернізації обладнання), що істотно покращує адаптивні якості систем управління технологічним обладнанням;

– широка номенклатура модулів вводу/виводу для зв'язку контролерів з датчиками, керованими механізмами, а також керуючими пристроями, виконаними за іншими принципами та іншою елементною базою, що позитивно позначається на ефективності побудови комплексних систем управління з використанням ПЛК;

– наявність вбудованої автоматичної функціональної діагностики дозволяє істотно спростити процес експлуатації і підвищити ремонтопридатність як власне контролера, так і керованого технологічного устаткування.

До числа основних характеристик (параметрів), за якими користувач здійснює вибір ПЛК для конкретного застосування, належать:

– кількість входів-виходів – одна з головних характеристик, яка визначає максимально можливу кількість контрольованих датчиків і керованих механізмів, які можуть бути підключені до ПЛК;

– номенклатура пропонованих модулів вводу/виводу – характеризує можливості адаптації ПЛК різних фірм до умов промислового використання в частині номенклатури і величин напруг живлення і комутованих струмів органів управління, датчиків і виконавчих механізмів;

– обсяг пам'яті для зберігання програм користувача – визначає можливості даного ПЛК в частині створення прикладного програмного забезпечення;

– швидкодія ПЛК – як правило, характеризує тривалість циклу однократного обслуговування всіх входів-виходів;

– типи використовуваних мов і технологій програмування – характеризує ступінь складності освоєння прикладного програмного забезпечення та зручність введення і коригування записаних у пам'яті керуючих програм;

– оснащеність стандартними інтерфейсами – характеризує пристосованість ПЛК до використання його в ієрархічних системах управління з можливістю дистанційного введення і коригування програм і даних, а також до використання у складі контролерних мереж.

Технічно контролери реалізуються по-різному. Це може бути механічний пристрій, пневматичний або гідравлічний автомат, релейна або електронна схема або навіть комп'ютерна програма.

В ПЛК застосовуються переважно три види пам'яті: ППЗУ типу PROM для зберігання базової (незмінної) частини керуючих програм; ОЗП з підживленням на акумуляторах на етапах налагодження і корегування програмного забезпечення, а також для зберігання оперативної інформації; ППЗУ типу REPROM як основна пам'ять для зберігання керуючих програм.

У разі, коли контролер вбудовано в машину масового випуску, вартість його проектування розділена на велику кількість виробів і є низькою відносно вартості виготовлення. У разі машин, що виготовляються в одиничних екземплярах, ситуація інша. Вартість проектування контролера домінує відносно вартості його фізичної реалізації.

Під час створення машин, зайнятих у сфері промислового виробництва, як правило, доводиться мати справу не більше ніж з одиницями однотипних пристроїв. Крім того, дуже суттєвою тут є можливість швидкого переналаштування устаткування на випуск іншої продукції.

Контролери, побудовані на основі реле або мікросхем з «жорсткою» логікою, неможливо переналаштувати на іншу роботу без суттєвої переробки. Очевидно, що таку можливість мають тільки програмовані логічні контролери. Ідея створення ПЛК виникла практично відразу з появою мікропроцесора.

Фізично типовий ПЛК є блоком з певним набором виходів і входів, для підключення датчиків і виконавчих механізмів (рис. 2.1).

Залежно від максимально можливої кількості входів-виходів, усі ПЛК поділяють на моделі, що відрізняються їх кількістю:

– мікро – до 64;

– малі – до 128;

– середні – до 512;

– великі – до 1024;

– надвеликі – понад 1024.

Логіка управління описується програмно на основі мікрокомп'ютерного ядра. Абсолютно однакові ПЛК можуть виконувати абсолютно різні функції. Причому для зміни алгоритму роботи не потрібно будь-яких переробок апаратної частини. Апаратна реалізація входів і виходів ПЛК орієнтована на поєднання з уніфікованими пристроями і мало придатна до змін.

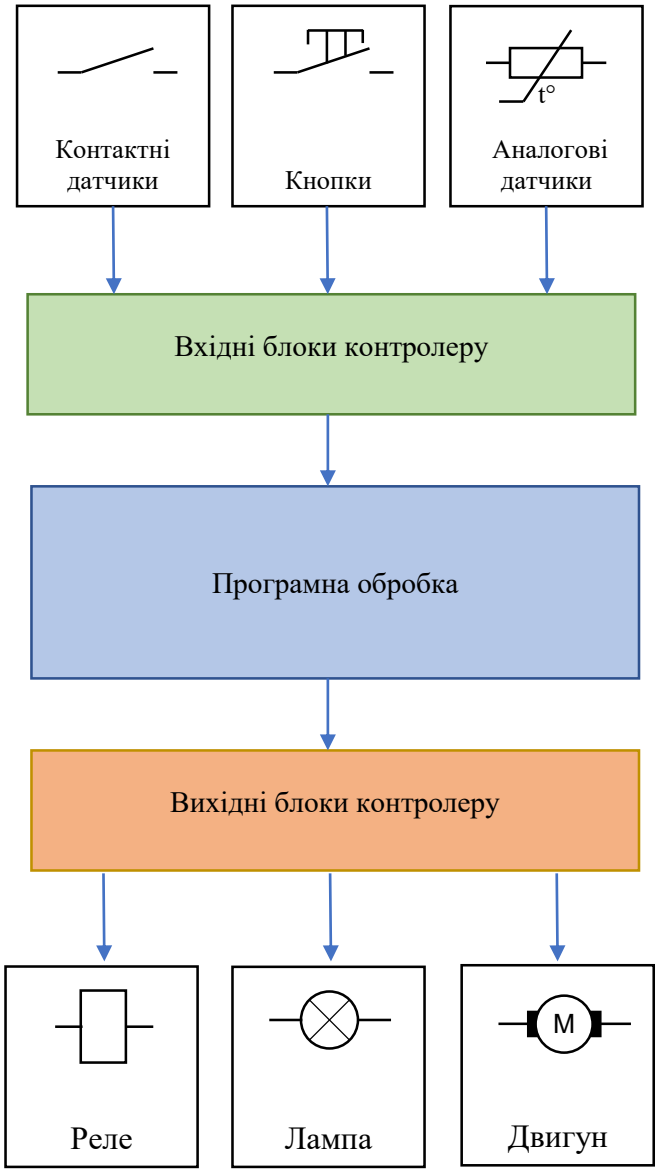


Рисунок 2.1 – Принцип роботи ПЛК

Завданням прикладного програмування ПЛК є тільки реалізація алгоритму управління конкретною машиною. Опитування входів і виходів контролер здійснює автоматично, незалежно від способу фізичного з'єднання. Цю роботу виконує системне програмне забезпечення. В ідеальному випадку прикладного програміста абсолютно не цікавить, як приєднані і де розташовані датчики і виконавчі механізми. Його робота не залежить від того, з яким контролером і якої фірми він працює. Завдяки стандартизації мов програмування додаток можна використовувати в будь-якому ПЛК, що підтримує даний стандарт.

Програмований контролер – це програмно керований дискретний автомат, який має певну кількість входів, підключених за допомогою датчиків

до об’єкта управління, і певну кількість виходів, підключених до виконавчих пристроїв. ПЛК контролює стан входів і виробляє певні послідовності програмно заданих дій, що впливають на змінення виходів.

ПЛК призначений для роботи в режимі реального часу в умовах промислового середовища і має бути доступний для програмування неспеціалістом у галузі інформатики [5].

На рис. 2.2 наведено приклад сучасного ПЛК від фірми ОВЕН типу ПЛК110 [М02].

Рисунок 2.2 – Контролер фірми ОВЕН ПЛК110 [М02]
для середніх систем автоматизації

Конструкція ПЛК може бути найрізноманітнішою – від стійки, заповненої апаратурою, до мініатюрних ПЛК. Спочатку ПЛК призначалися для управління послідовними логічними процесами, що і зумовило слово «логічний» у назві ПЛК.

Сучасні ПЛК крім простих логічних операцій здатні виконувати цифрову обробку сигналів, управління приводами, регулювання, функції операторського управління тощо.

2.2 Типи ПЛК

Сучасні ПЛК, що використовують інноваційні технології, далеко пішли від перших спрощених реалізацій промислового контролера, але закладені в систему управління універсальні принципи були стандартизовані та успішно розвиваються вже на базі новітніх технологій.

Найбільшими світовими виробниками ПЛК сьогодні є компанії Siemens AG, Allen-Bradley, Rockwell Automation, Schneider Electric, Omron. Крім них ПЛК випускають і багато інших виробників, включаючи компанії ТОВ КОНТАР, Овен, Сегнетікс, Fastwel Груп, група компаній Текон та інші.

Апаратно ПЛК є обчислювальною машиною. Тому архітектура його процесорного ядра практично не відрізняється від архітектури комп'ютера.

Відмінності спостерігаються у складі периферійного обладнання, також відсутні відеокарта, засоби ручного вводу і дискова підсистема. Замість них ПЛК має блоки входів і виходів.

За конструктивним виконанням ПЛК поділяють на:

– моноблочні (рис. 2.3);

– модульні (рис. 2.4);

– розподілені.

Моноблочні (одноплатні) ПЛК мають фіксований набір входів-виходів. У корпусі моноблочного ПЛК поряд з центральним процесором (ЦП), пам'яттю і блоком живлення розміщується фіксований набір входів-виходів.

Рисунок 2.3 – Моноблочні програмовані логічні контролери

У модульних ПЛК використовують модулі вводу/виводу, що встановлюються окремо (рис. 2.4). Модулі вводу/виводу встановлюються у різній комбінації і кількості залежно від необхідної конфігурації. Таким чином, досягається мінімальна апаратна надмірність.

Згідно з вимогами МЕК 61131, тип і кількість модулів вводу/виводу може змінюватися залежно від поставленого завдання та оновлюватися з часом. ПЛК, наведені на рис. 2.4, можуть діяти в режимі «ведучого» і розширюватися «веденими» ПЛК через інтерфейс Ethernet.

Рисунок 2.4 – Програмовані логічні контролери з розширеними можливостями

У розподілених системах модулі вводу/виводу, які утворюють єдину систему управління, можуть бути рознесені на значні відстані.

Моноблочні функціонально завершені ПЛК можуть включати в себе невеликий дисплей і кнопки управління. Дисплей призначений для відображення поточних робочих параметрів і команд робочих програм і технологічних установок, що вводяться за допомогою кнопок. Більш складні ПЛК комбінуються з окремих функціональних модулів, спільно закріплюються на стандартній монтажній рейці. Залежно від кількості обслуговуваних входів і виходів, встановлюється необхідна кількість модулів вводу/виводу.

Характерним для сучасних контролерів є використання багатопроцесорних рішень. У цьому випадку модулі вводу/виводу мають власні мікропроцесори, які виконують необхідну попередню обробку даних. Модуль центрального процесора має виділену швидкісну магістраль даних для роботи з пам’яттю і окрему магістраль (мережу) для спілкування з модулями вводу/виводу.

Ще одним варіантом побудови ПЛК є мезонін-технологія. Всі силові ланцюги, пристрої захисту контролера реалізуються на несучій платі. Процесорне ядро контролера, включаючи систему виконання, побудовано на окремій змінній (мезонін) платі. Внаслідок з’являється можливість складати кілька комбінацій процесорного ядра і різних силових плат без необхідності

коригування програмного забезпечення. За необхідності процесор можна замінити навіть у готовій системі.

Джерело живлення може бути вбудованим в основний блок ПЛК, але частіше воно виконане у вигляді окремого блоку живлення (БЖ), який закріплюється поруч на стандартній рейці. Блок живлення невеликої потужності подано на рис. 2.5.

Рисунок 2.5 – Блок живлення для ПЛК

Первинним джерелом для БЖ найчастіше слугує промислова мережа 24/48/110/220/400 В, 50 Гц. Інші моделі БЖ можуть використовувати як первинне джерело постійної напруги на 24/48/125 В.

Стандартними для промислового обладнання та ПЛК є вихідні напруги БЖ: 12, 24, 48 В. У системах підвищеної надійності можлива установка двох спеціальних резервованих БЖ для дублювання електроживлення.

Для збереження інформації під час аварійних відключень мережі електроживлення в ПЛК використовують додаткову батарею.

Як відомо, початкова концепція програмованого логічного контролера сформувалася за часів переходу з релейно-транзисторних систем управління промисловим обладнанням на мікроконтролери, що тоді з’явилися. Подібні ПЛК з 8- і 16-розрядних МП обмеженою продуктивності досі успішно експлуатуються і знаходять нові сфери застосування.

Величезний прогрес у розвитку мікроелектроніки торкнувся всієї елементної бази ПЛК. У них значно розширився діапазон функціональних можливостей.

Таким чином, стає зрозуміло, що ПЛК – це просто особливим чином спроектована цифрова система управління на основі процесорів різної потужності і з різною функціональною оснащеністю, залежно від призначення. Таку систему можна також вважати спеціалізованим міні-комп'ютером. Причому вона спочатку орієнтована на експлуатацію в цехах промислових підприємств, де є безліч джерел електромагнітних перешкод, а температура може бути як позитивною, так і негативною.

Додатково до мінімізації впливу вищевказаних факторів необхідно передбачити і захист від агресивного зовнішнього середовища, що включає пил, бризки технологічних рідин і пароповітряні суспензії. У таких випадках передбачена установка ПЛК в захисні шафи або у віддалених приміщеннях. Окремі модулі можуть розміщуватися на відстані до сотень метрів від основного комплекту ПЛК і експлуатуватися за екстремальних зовнішніх температур. Згідно з МЕК 61131, для ПЛК з зовнішньою установкою допустима температура 5...55 °С. Для встановлюваного в закритих шафах ПЛК необхідно забезпечити робочий діапазон 5...40 °С за відносної вологості 10...95% (без утворення конденсату).

Тип ПЛК обирається в ході проектування системи управління і залежить від поставлених завдань та умов виробництва. В окремих випадках це може бути моноблочний ПЛК з обмеженими функціями, який має достатню кількість входів і виходів. В інших умовах будуть потрібні ПЛК з розширеними можливостями, що дозволяють використовувати розподілену конфігурацію з віддаленими модулями входу-виходу та з віддаленими пультами управління технологічним процесом.

Зв'язок між віддаленими блоками і основним ядром ПЛК здійснюється через захищені польові шини мідними кабелями та оптичними лініями зв'язку. В окремих випадках, наприклад, для зв'язку з рухомими об'єктами, застосовують безпровідні технології, найчастіше це мережі і канали Wi-Fi. Для взаємодії з іншими ПЛК можуть застосовуватися як широко відомі інтерфейси RS-232 і RS-485, так і більш захищені промислові варіанти типу Profibus і CAN.

2.3 Входи-виходи ПЛК

На початку своєї появи ПЛК мали тільки бінарні входи, тобто входи, значення сигналів на яких здатні приймати тільки два стани – логічний нуль і логічну одиницю. Наявність струму (або напруги) в ланцюзі входу вважається зазвичай логічною одиницею. Відсутність струму (напруги) означає логічний нуль. Датчиками, що формують такий сигнал, можуть бути кнопки ручного управління, кінцеві датчики, датчики руху, контактні термометри і багато інших.

Бінарний вихід також має два стани – підключений і відключений. Сфера застосування бінарних виходів очевидна: електромагнітні реле, силові пускачі, електромагнітні клапани, світлові сигналізатори тощо.

У сучасних ПЛК широко використовуються аналогові входи й виходи. Аналоговий або безперервний сигнал відображає рівень напруги або струму, відповідний деякій фізичній величині в кожен момент часу. Цей рівень може стосуватися температури, тиску, ваги, стану, швидкості, частоти тощо, тобто, будь-якої фізичної величини.

Аналогові входи контролерів можуть мати різні параметри і можливості. До таких параметрів належать: розрядність АЦП, діапазон вхідного сигналу, час і метод перетворення, несиметричний або диференційний вхід, рівень шуму і нелінійність, можливість автоматичного калібрування, програмний або апаратний коефіцієнт посилення, фільтрація. Особливі класи аналогових входів представляють входи, призначені для підключення термометрів опору і термопар. Тут потрібне застосування спеціальної апаратної підтримки (триточкове підключення, джерела зразкового струму, схеми компенсації холодного спаю, схеми лінеаризації тощо).

У сфері застосування ПЛК бінарні входи й виходи зазвичай називають дискретними. Але багато ПЛК мають спеціалізовані входи-виходи. Вони орієнтовані на роботу з конкретними специфічними датчиками, які вимагають певних рівнів сигналів, живлення і спеціальної обробки. Наприклад, квадратурні шифратори, блоки управління двигунами, інтерфейси дисплейних модулів тощо.

Входи-виходи ПЛК не обов’язково мають бути фізично зосереджені в загальному корпусі з процесорним ядром. Останніми роками все більшої популярності набувають технічні рішення, що дозволяють повністю відмовитися від прокладки кабелів для аналогових ланцюгів. Входи-виходи

виконуються у вигляді мініатюрних модулів, розташованих у безпосередній близькості від датчиків і виконавчих механізмів. З'єднання підсистеми вводу/виводу з ПЛК виконується за допомогою одного загального цифрового кабелю. Наприклад, в інтерфейсі Actuators/Sensors interface застосовується плоский профільований кабель («жовтий кабель») для передачі даних і живлення тільки за двома проводами.

2.4 Режим реального часу і обмеження на використання ПЛК

Для математичних систем характеристикою якості роботи є правильність знайденого рішення. У системах реального часу крім правильності рішення визначальну роль відіграє час реакції. Логічно, що правильне рішення, яке отримано з більшою затримкою ніж припустимо, не є прийнятним.

Прийнято розрізняти системи жорсткого і м'якого реального часу. У *системах жорсткого реального часу* існує виражений часовий поріг. З його перевищенням настають незворотні катастрофічні наслідки. У *системах м'якого реального часу* характеристики системи погіршуються зі збільшенням часу керуючої реакції. Система може працювати погано або ще гірше, але нічого катастрофічного при цьому не відбувається.

Класичний підхід для задач жорсткого реального часу вимагає побудови подієво керованої системи. Для кожної події в системі встановлюється певний час реакції і певний пріоритет. Практична реалізація таких систем складна і завжди вимагає ретельної переробки моделювання.

Для ПЛК істотне значення має не тільки швидкодія самої системи, а й час проектування, впровадження та можливого оперативного переналаштування.

Абсолютна більшість ПЛК працюють за методом періодичного опитування вхідних даних (сканування). ПЛК опитує входи, виконує призначену для користувача програму і встановлює необхідні значення виходів. Специфіка застосування ПЛК обумовлює необхідність одночасного вирішення декількох завдань. Прикладна програма може бути реалізована у вигляді безлічі логічно незалежних завдань, які мають працювати водночас.

Насправді ПЛК має зазвичай один процесор і виконує декілька завдань псевдопаралельно, послідовними порціями. Час реакції на подію виявляється залежним від кількості водночас оброблюваних подій. Розрахувати мінімальне й максимальне значення часу реакції, звичайно, можна, але додавання нових завдань або збільшення обсягу програми призведе до збільшення часу реакції.

Така модель більше підходить для систем м’якого реального часу. Сучасні ПЛК мають типове значення часу робочого циклу, яке вимірюється одиницями мілісекунд і менше. Оскільки час реакції більшості виконавчих пристроїв значно вище, з реальними обмеженнями можливості використання ПЛК за часом доводиться стикатися рідко.

У деяких випадках обмеженням слугує не час реакції на подію, а обов’язковість його фіксації, наприклад робота з датчиками, що формують імпульси малої тривалості. Це обмеження долається спеціальною конструкцією входів. Так, лічильний вхід дозволяє фіксувати і підраховувати імпульси з періодом у багато разів менше часу робочого циклу ПЛК. Спеціалізовані інтелектуальні модулі у складі ПЛК дозволяють автономно відпрацьовувати задані функції, наприклад модулі управління сервоприводом.

До негативних факторів, що визначають промислове середовище, належать: температура й вологість, удари та вібрації, корозійно-активне газове середовище, мінеральний і металевий пил, електромагнітні перешкоди. Перераховані фактори характерні для виробничих умов і обумовлюють жорсткі вимоги, що визначають схемотехнічні рішення, елементну і конструктивну базу ПЛК. У процесі серійного виробництва ПЛК обов’язковий технічний прогін готових виробів, включаючи кліматичні, вібраційні та інші випробування.

ПЛК – це конструктивно закінчений виріб, практичне виконання якого визначається необхідним ступенем захисту, починаючи від контролерів у легких пластикових корпусах, призначених для монтажу в шафі (ступінь захисту IP20), і до герметичних пристроїв у литих металевих корпусах, призначених для роботи в особливо жорстких умовах.

Правильно підібраний за умовами експлуатації контролер не можна пошкодити ззовні без застосування екстремальних методів. Штатними для ПЛК є такі апаратні рішення, як повна гальванічна розв’язка входів-виходів, захист за струмом і напругою, дзеркальні вихідні канали, сторожовий таймер завдань і мікропроцесорного ядра.

2.5 Інтеграція ПЛК у систему управління підприємством

Контролери традиційно працюють у нижній ланці автоматизованих систем управління підприємством (АСУП) – систем, безпосередньо пов’язаних з технологією виробництва. ПЛК зазвичай є першим кроком у побудові систем АСУ. Це пояснюється тим, що необхідність автоматизації окремого

механізму або установки завжди найбільш очевидна. Вона дає швидкий економічний ефект, покращує якість виробництва, дозволяє уникнути фізично важкої і рутинної роботи. Контролери створені саме для такої роботи.

Повністю автоматичну систему вдається створити не завжди. Часто необхідне «загальне керівництво» з боку кваліфікованої людини – диспетчера. На відміну від технічних засобів АСУ ТП, такі системи називають автоматизованими.

Зараз використовується цілий клас програмного забезпечення реалізації інтерфейс людина-машина (HMI). Це так звані системи збору даних і оперативного диспетчерського управління (Supervisory Control And Data Acquision System – SCADA).

Приклад пристрою для реалізації інтерфейсу людина-машина показаний на рис. 2.6.

Рисунок 2.6 – Сенсорні панельні контролери СПК1ХХ

Сучасні SCADA-системи створюються з обов’язковим застосуванням засобів мультимедіа. Крім живого відображення процесу виробництва, диспетчерські системи дозволяють накопичувати отримані дані, проводять їх зберігання і аналіз, визначають критичні ситуації і здійснюють оповіщення персоналу телефонними каналами або радіомережею, дозволяють створювати сценарії управління, формують дані для аналізу економічних характеристик виробництва.

Створення систем диспетчерського управління є окремим видом бізнесу. Розподіл виробництва ПЛК, засобів програмування і диспетчерських систем призвів до появи стандартних протоколів обміну даними. Найбільшу популярність отримала технологія OPC (OLE for Process Control), заснована на механізмі DCOM Microsoft Windows. Механізм динамічного обміну даними (DDE) ще досить поширений, незважаючи на те, що вимогам систем реального часу не задовольняє.

У комплекс програмування ПЛК входить OPC-сервер. Він вміє отримувати доступ до даних ПЛК так само прозоро, як і відладчик. Достатньо забезпечити канал передачі даних ПЛК-OPC-сервер. Зазвичай такий канал існує, він використовується в ході налаштування. Уся подальша робота зводиться до визначення переліку доступних змінних, правильного налаштування мережі, конфігурації OPC-сервера і SCADA-системи. В цілому операція дуже нагадує налаштування загальнодоступних пристроїв локальної мережі ПК.

ПЛК є засобами системної інтеграції і складовою частиною базового програмного забезпечення сучасної АСУ ТП (рис. 2.7).

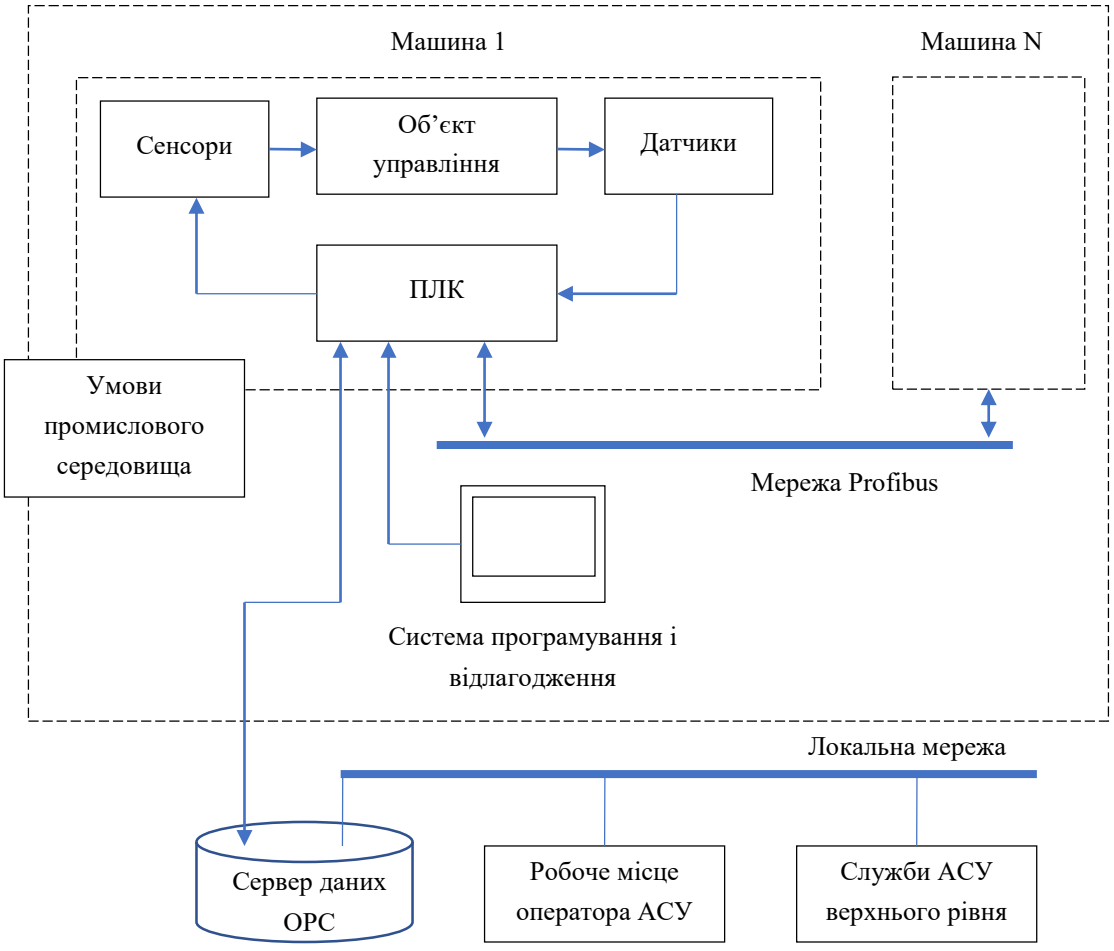


Рисунок 2.7 – Місце ПЛК в АСУ ТП

Іншим важливим завданням є інтеграція декількох ПЛК з метою синхронізації їх роботи. Тут з'являються мережі, які мають ряд специфічних вимог.

2.6 Використання ПЛК під час створення автоматизованої системи управління технологічними процесами

Автоматизовану систему управління технологічними процесами (АСУ ТП) можна подати у вигляді окремих її частин, які називаються підсистемами. Підсистема – це частина основної системи, виділена за будь-якою ознакою. Сукупність підсистем автоматизованої системи управління технологічними процесами, незалежно від сфери застосування, як правило, одна й та сама.

Структура будь-якої автоматизованої системи управління технологічними процесами може бути подана сукупністю підсистем, що її забезпечують, серед яких зазвичай виділяють програмно-технічне забезпечення, інформаційне, математичне (іноді алгоритмічне), організаційно-методичне та правове.

Основною змінною величиною, що характеризує автоматизовану систему управління технологічними процесами, є критерій управління.

Критерієм управління називається співвідношення, яке залежить від якості функціонування системи в цілому і приймає конкретні числові значення залежно від використовуваних керуючих впливів.

Як було сказано вище, автоматизована система управління технологічними процесами складається з підсистем. Розглянемо докладніше значення кожної з них:

– технічне забезпечення, яке включає обчислювальні та керуючі пристрої, засоби отримання інформації (датчики), засоби перетворення, зберігання, відображення та реєстрації інформації, передачі і перетворення сигналів, та виконавчі пристрої. Як приклад таких пристроїв можуть виступати комп'ютери, програмовані логічні контролери, маршрутизатори, електронні реєстратори, промислові панелі оператора;

– програмне забезпечення, що складається із сукупності програм, необхідних для реалізації функцій АСУ ТП і забезпечення керуючого алгоритму, без якого неможливе функціонування комплексу технічних засобів;

– інформаційне забезпечення містить відомості, що характеризують стан системи управління, системи класифікації та кодування технологічної та техніко-економічної інформації, масиви даних і документів, необхідних для виконання функцій АСУ ТП, в тому числі нормативно-довідкову і правову інформацію;

– організаційне забезпечення – це сукупність описів функціональних, технічних і організаційних структур, а також інструкцій для оперативного персоналу; дана сукупність має забезпечити належне функціонування перерахованих структур;

– оперативний персонал – це технологи-оператори, які здійснюють контроль за управлінням системи;

– експлуатаційний персонал – це персонал, який забезпечує експлуатацію, технічне обслуговування та ремонт системи.

Типова структура автоматизованої системи управління промисловим підприємством наведена на рис. 2.8.

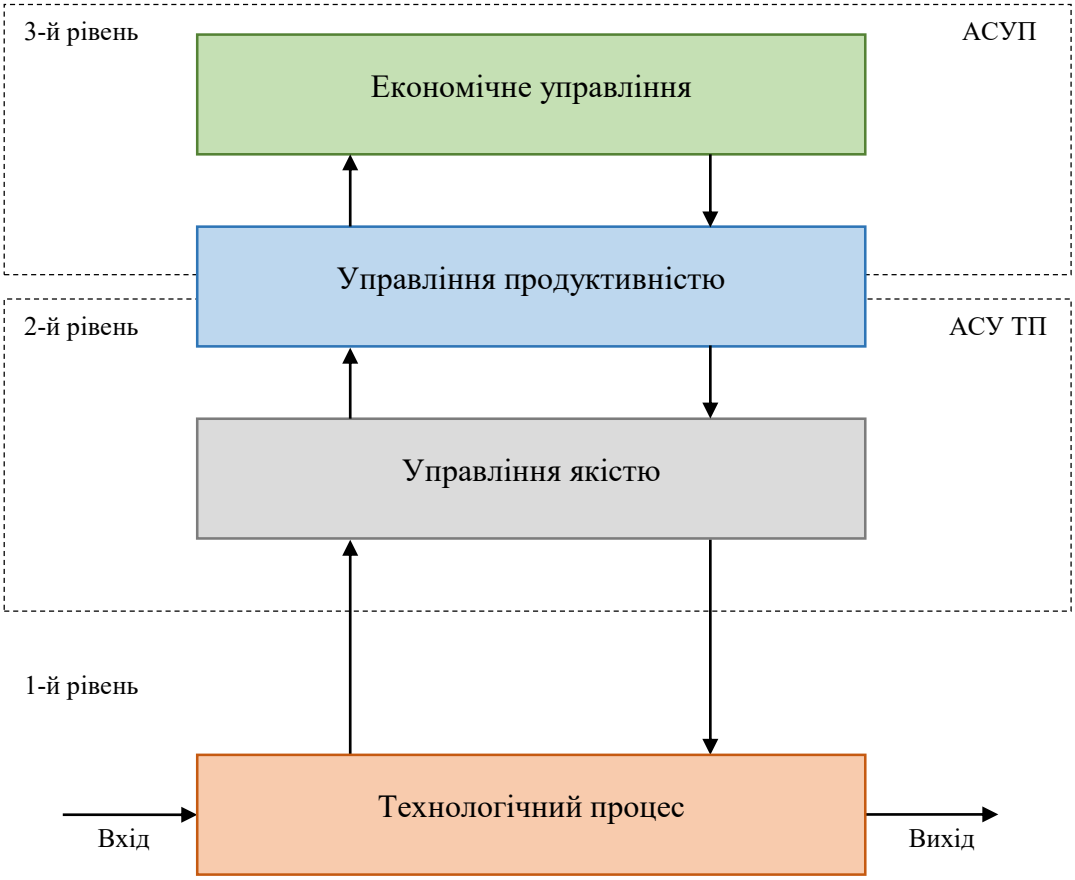


Рисунок 2.8 – Структура АСУ промислового підприємства

На даному рисунку показано місце підсистем і компонент у складі системи управління підприємством. Виділено рівні технологічного процесу, АСУ ТП і АСУП. Виділено компоненти економічного управління, управління якістю і продуктивністю.

У ході побудови засобів сучасної промислової автоматики у вигляді автоматизованої системи управління технологічними процесами, використовується ієрархічна інформаційна структура із застосуванням на різних рівнях обчислювальних засобів різної потужності.

На рис. 2.9 наведена загальна структура автоматики АСУ ТП.

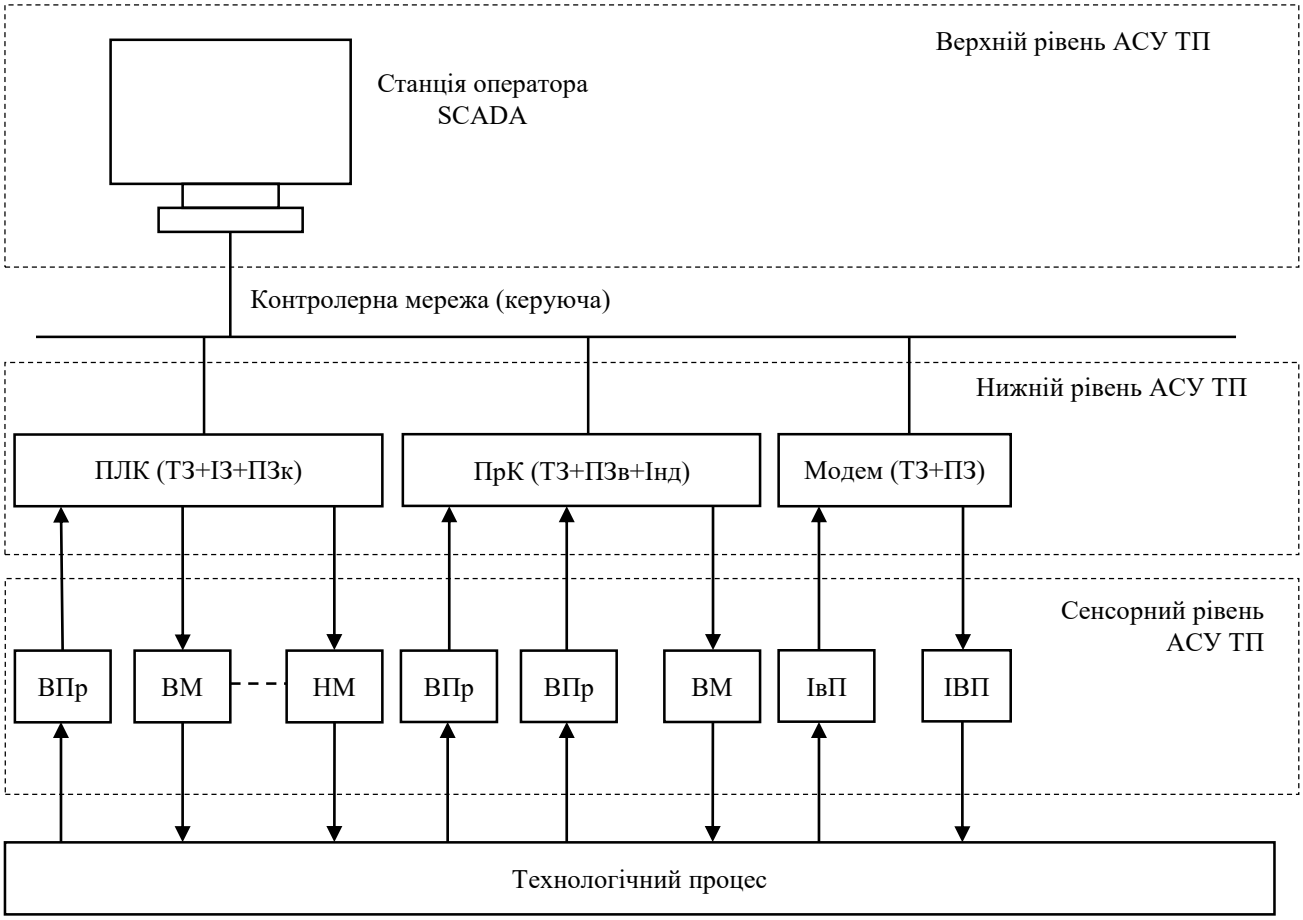


Рисунок 2.9 – Загальна схема автоматики АСУ ТП

На рисунку 2.9 наведені такі позначення:

– ВПр – вимірювальні перетворювачі (датчики);

– ВМ – виконуючі механізми;

– ПЛК – програмований логічний контролер;

– ПрК – програмований контролер;

– ІвП – інтелектуальний вимірювальний перетворювач;

– ІВП – інтелектуальні виконуючі пристрої;

– Модем – модулятор/демодулятор сигналів;

– ТЗ – технічне забезпечення (апаратна частина);

– ІЗ – інформаційне забезпечення (бази даних);

– ПЗ – програмне забезпечення;

– КЗ – комунікаційне забезпечення (послідовний порт і ПЗ);

– ПЗк – програмне забезпечення користувача;

– ПЗв – програмне забезпечення виробника;

– Інд – індикатор.

Частіше автоматизовані системи управління технологічними процесами створюють за такими типовими схемами:

– однорівнева схема (децентралізована, або іншими словами, локальна система), що містить програмований логічний контролер (ПЛК), або моноблочний, що налаштовується (МНК), або програмований логічний контролер і панель оператора. Крім безпосередньо самого завдання управління, всі ці технічні засоби забезпечують індикацію і сигналізацію стану контрольованого або регульованого технологічного параметра за допомогою систем людино-машинного інтерфейсу. Це можуть бути як прості текстові панелі, так і сенсорні панелі оператора з великою функціональністю;

– дворівнева схема (централізована система), що включає в себе: на нижньому рівні – декілька ПЛК з підключеними до них датчиками і виконавчими пристроями, на верхньому рівні – одна, можливо декілька, операторських (робочих) станцій. Робоча станція являє собою комп’ютер для створення автоматизованого робочого місця (АРМ) оператора. Зазвичай робоча станція або АРМ укомплектована системою збору та візуалізації даних (SCADA-системою) того чи іншого рівня складності.

Дворівневі системи промислової автоматики дають величезні переваги у глибині зберігання інформації, створенні архівів, доступу до інформації з будь-якої точки світу через Web-інтерфейс. Також перевагою дворівневої системи є розподіл завдань між вузлами системи, що дозволяє підвищити надійність її функціонування. Типова функціональна схема дворівневої АСУ технологічного процесу показана на рис. 2.10.

Усі ПЛК та автоматизовані робочі станції об’єднуються промисловими інформаційними мережами, що забезпечують постійний обмін даними.

Основні функції нижчого рівня:

– збір, електрична фільтрація і перетворення сигналів з первинних перетворювачів (датчиків) з аналогової в цифрову форму;

– реалізація локальних автоматизованих систем управління конкретного технологічного процесу в об’ємі функцій ПЛК однорівневої системи;

– реалізація аварійної і попереджувальної сигналізації;

– організація системи захисту і блокувань;

– обмін поточними даними з ПК верхнього рівня через промислову мережу за запитами ПК.

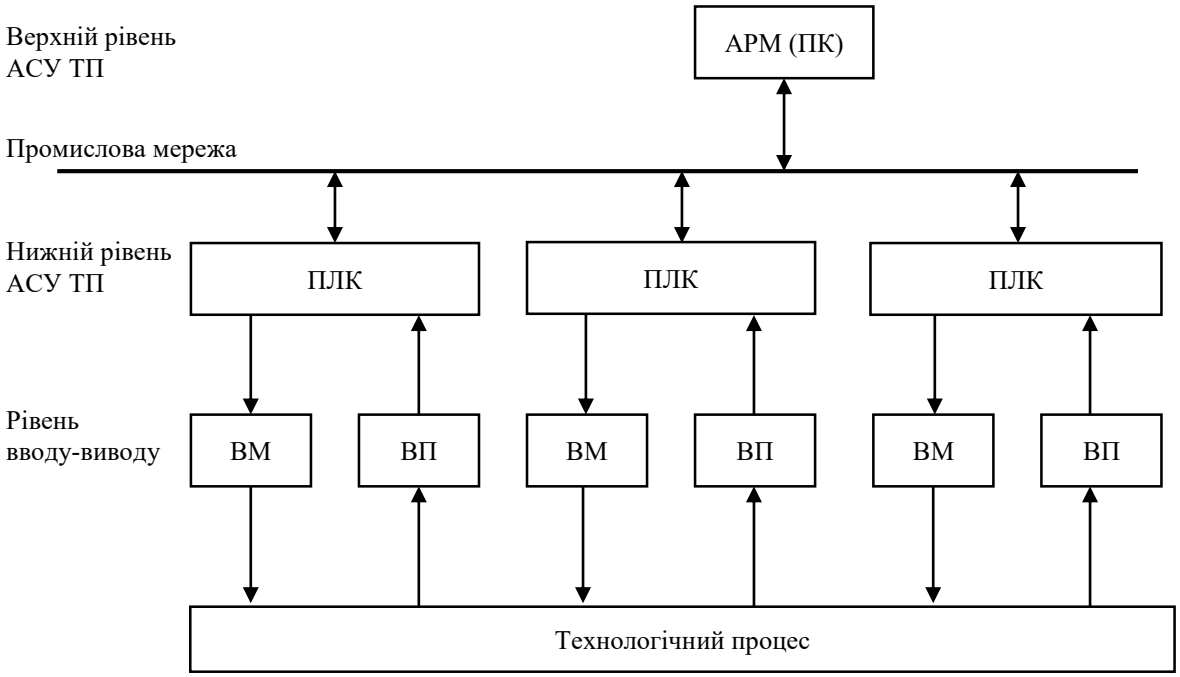


Рисунок 2.10 – Типова функціональна схема дворівневої АСУ ТП

Основні функції верхнього рівня:

– візуалізація стану технологічного процесу;

– поточна реєстрація характеристик технологічного процесу;

– оперативний аналіз стану устаткування і технологічного процесу;

– реєстрація дій оператора, у тому числі під час аварійних повідомлень;

– архівація і тривале зберігання значень протоколів технологічного процесу;

– реалізація алгоритмів «системи порадника»;

– супервізорне управління;

– реалізація безпосереднього цифрового управління;

– управління в режимі збору даних – зберігання і ведення баз цих параметрів техпроцесів, критичних параметрів устаткування, ознак аварійних станів технологічного процесу;

– реалізації системи контролю доступу операторів і їх паролів.

Таким чином, нижній рівень реалізує алгоритми управління устаткуванням, верхній – вирішення стратегічних питань функціонування АСУ ТП.

Наприклад, рішення підключити, відключити насос, змінити його частоту обертання починається на верхньому рівні, а подання усіх необхідних сигналів, що управляють, перевірка теплового перевантаження насосу, реалізація механізму блокувань і захисту по «сухому» ходу виконується на нижньому рівні.

Ієрархічна структура автоматизованої системи управління технологічними процесами передбачає:

– потік команд спрямований від верхнього рівня до нижнього;

– нижній відповідає верхньому за його запитами.

Ця структура підвищує надійність усієї системи управління в цілому, оскільки з виходом з ладу комплексу технічних засобів верхнього рівня працездатність нижнього рівня АСУ ТП зберігається. Такі несправності сприймаються нижнім рівнем лише як відсутність нових команд і запитів, але безпосередньо на безпечний хід технологічного процесу не впливають.

Під час конфігурації ПЛК встановлюється: до якого часу після отримання останнього запиту ПЛК продовжує функціонувати, підтримуючи останній заданий режим, після чого переходить у потрібний за цієї аварійної ситуації режим роботи.

2.7 Принцип роботи ПЛК

2.7.1 Принцип програмування ПЛК

В основу структурної організації ПЛК покладена типова структура МП-пристрою, оснащеного модулями зв’язку з керованим об’єктом, а також пультом (пристроєм) користувача, за допомогою якого реалізуються функції програмування, відлагодження, діагностування керуючої програми і відображення станів керованого об’єкта.

Для зберігання програми, що управляє (УП), використовуються різного типу ПЗП або ОЗП з підживленням. Для пам’яті даних використовується ОЗП, а іноді і ОЗП з підживленням. Послідовний інтерфейс використовується для стикування з іншими мікропроцесорними системами.

На рис. 2.11 показана узагальнена структура ПЛК.

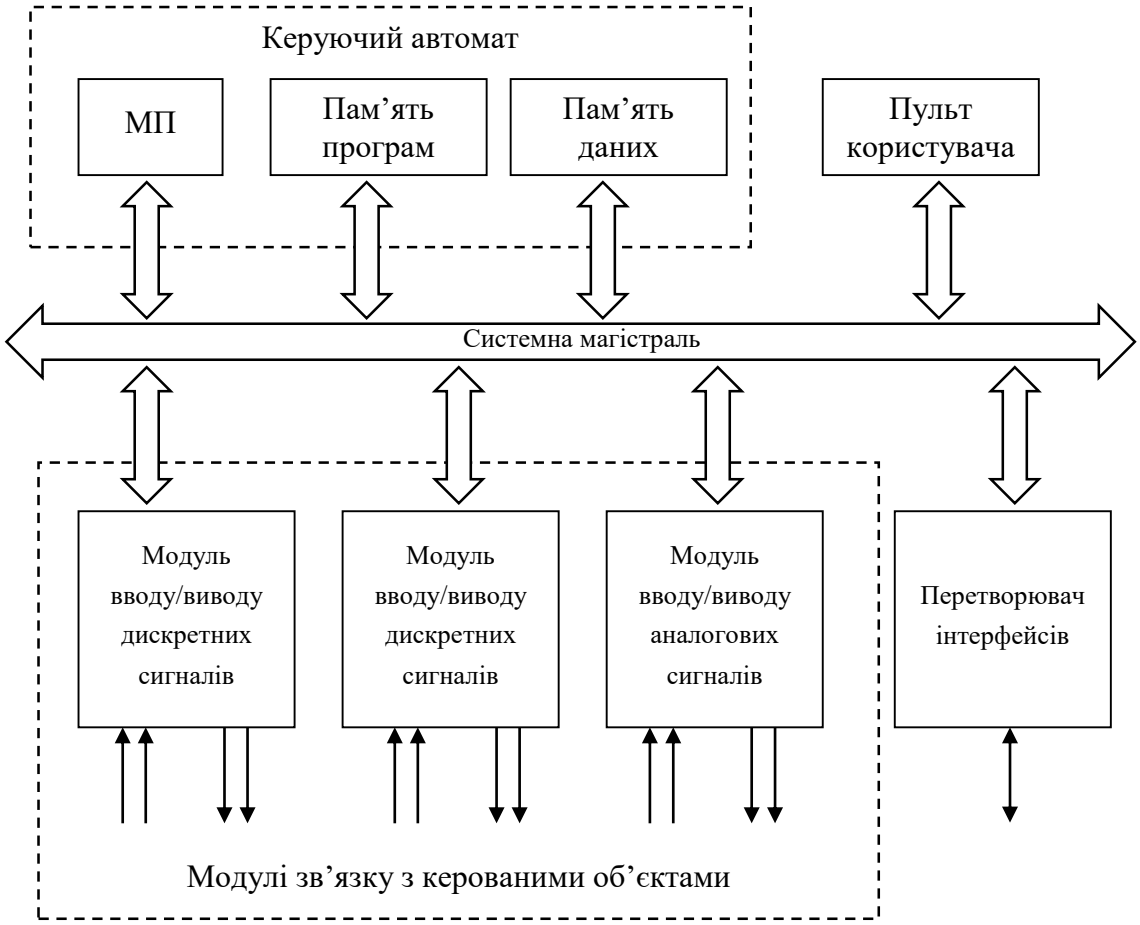


Рисунок 2.11 – Узагальнена структура ПЛК

Основною особливістю функціонування ПЛК є його циклічна робота. Кожен цикл складається з трьох основних етапів. На першому етапі проводиться опитування станів входів і запам’ятовування цієї інформації. На другому етапі виконується аналіз отриманої інформації відповідно до тієї, що зберігається у пам’яті ПЛК керуючою програмою, тобто проводиться спроба розв’язання сукупності записаних у програмі логічних рівнянь. На третьому етапі на основі результатів розв’язання логічних рівнянь формуються команди збудження виходів, сигнали запуску і скидання таймерів і лічильників, а також внутрішніх операторів програми.

Відпрацювання частин (ділянок) УП (окремих логічних рівнянь) здійснюється одна за одною в порядку їх розміщення у програмі з поверненням до початку УП після закінчення усього циклу.

Циклічна обробка (сканування) УП здійснюється за допомогою процесора. Одноразове обслуговування відповідно до програми усіх входів-виходів ПЛК називається циклом сканування або робочим циклом, а час, що

витрачається на це, – тривалістю циклу сканування $T_ц$, яка характеризує швидкодію ПЛК.

На рис. 2.12 показаний принцип роботи ПЛК.

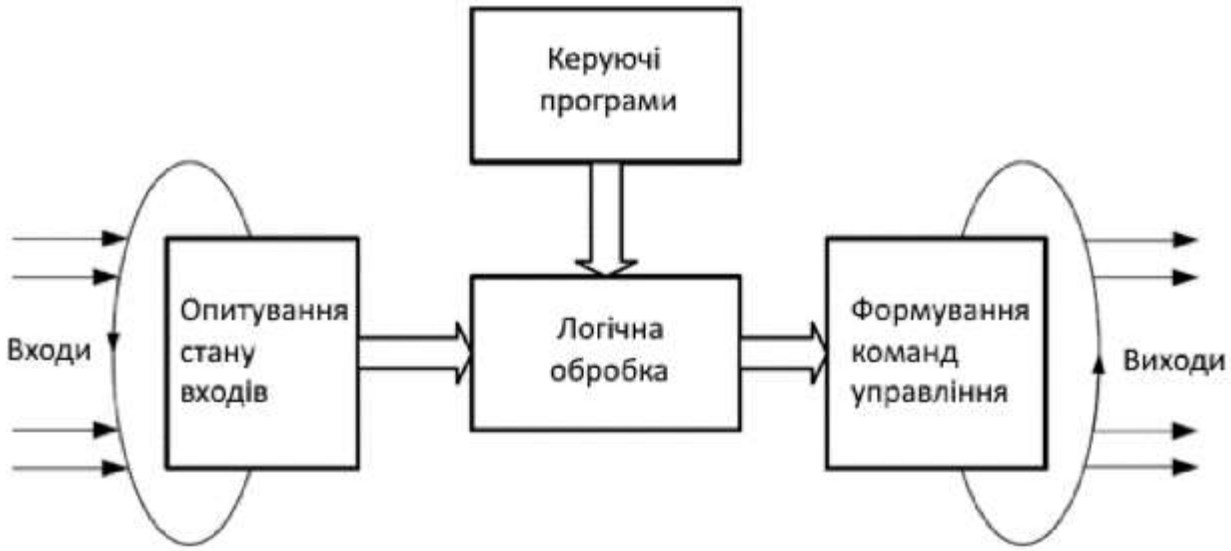


Рисунок 2.12 – Принцип роботи ПЛК

Для нормального функціонування ПЛК обов'язково має виконуватися умова: $T_ц < T_{спр.вм}$, де $T_{спр.вм}$ – час спрацьовування виконавчих механізмів.

Необхідність виконання цієї умови викликана тим, що на входах ПЛК можлива поява перешкод, а, отже, і помилкових сигналів про спрацьовування того або іншого датчика, що може призвести до помилкового формування вихідних сигналів ПЛК і до аварії на керованому об'єкті.

Якщо $T_ц < T_{спр.вм}$, то виконавчий механізм не встигає увімкнутися за один робочий цикл ПЛК, а в наступному циклі, якщо перешкода носить випадковий характер, помилковий сигнал, що управляє, виданий не буде.

Рекомендується опитування станів входів проводити не один раз, а *n* разів і тільки після *n*-разового підтвердження стану конкретного входу використати цю інформацію для подальшої логічної обробки.

До вихідних функцій насамперед належать функції увімкнення/вимкнення (запуску/зупинки) виконавчих механізмів керованого об'єкта. До них також належать функції пуску-скидання таймерів і лічильників, увімкнення-вимкнення внутрішніх операторів, які використовуються як для запам'ятовування імпульсних вхідних сигналів, так і проміжних станів УП, а також проміжних результатів, що отримуються в ході відпрацювання ділянок програми.

До вихідних належать також функції, пов’язані з організацією зон ігнорування ділянок програми, з організацією умовних і безумовних переходів у програмі, з обнуленням і без обнулення вказаних вище вихідних функцій.

Головною вимогою до ПЛК завжди була і залишається можливість його експлуатації існуючим технічним персоналом і можливість швидкої заміни старого устаткування. Тому мови програмування комп’ютерів і вбудованих мікропроцесорних систем управління погано підходять для програмування ПЛК. Тут потрібні простіші й наочніші мови, що дозволяють викладати задачі у близьких до вживаних технологій категоріях. Залучення ж до програмування спеціалізованої фірми призводить до залежності, якщо реалізація не є досить прозорою. Складна мова програмування ПЛК знижує його шанси на конкурентному ринку істотно більше, ніж масогабаритні показники.

Програмні застосування, що імітують технологію ПЛК на комп’ютері (оснащеному платами вводу/виводу), отримали назву програмний ПЛК (soft PLC). Програмна емуляція ПЛК зручна тим, що завдяки наявності багатозадачної операційної системи можна об’єднати в одному місці контролер, середовище програмування і систему диспетчерського управління.

Істотний мінус такого рішення – великий час виходу на робочий режим після увімкнення живлення або зависання комп’ютера. Особливо небезпечно, якщо перезапуск ініціював «сторожовий таймер» в автоматичному режимі, тоді як стан виконавчих механізмів не відповідає вихідним позиціям. Завантаження операційної системи може займати декілька хвилин, весь цей час система залишається некерованою. Для ПЛК час «холодного» запуску вимірюється мілісекундами.

2.7.2 Робочий цикл ПЛК

Завдання управління вимагають безперервного циклічного контролю. У будь-яких цифрових пристроях безперервність досягається за рахунок застосування дискретних алгоритмів, що повторюються через досить малі проміжки часу. Таким чином, обчислення в ПЛК завжди повторюються циклічно.

Одна ітерація, що включає замірювання, обрахування і вироблення дії, називається робочим циклом ПЛК.

Програма ПЛК виконується як частина процесу, що повторюється, який називається скануванням (рис. 2.13). У стандартному ПЛК сканування розпочинається з того, що процесор зчитує стан вхідних контактів. Далі

виконується технологічна програма. Після закінчення виконання програми процесор виконує внутрішні діагностичні та комунікаційні завдання.

Рисунок 2.13 – Стандартний цикл роботи ПЛК

Наступним кроком є оновлення статусу усіх виходів. Цей процес повторюється постійно, поки ПЛК знаходиться у режимі роботи.

Після увімкнення живлення ПЛК виконує самотестування і налаштування апаратних ресурсів, очищення оперативної пам’яті даних (ОЗП), контроль цілісності програми користувача. Якщо програма збережена в пам’яті, ПЛК переходить до основної роботи, яка складається з постійного повторення послідовності дій, що входять у робочий цикл.

Робочий цикл ПЛК складається з декількох фаз:

1. Початок циклу.
2. Читання стану входів.
3. Виконання коду програми користувача.
4. Запис стану виходів.
5. Обслуговування апаратних ресурсів ПЛК.
6. Монітор системи виконання.
7. Контроль часу циклу.
8. Перехід у початок циклу.

На самому початку циклу ПЛК проводить фізичне читання входів. Отримані значення розміщуються в області пам’яті входів. Таким чином, створюється повна одномоментна дзеркальна копія значень входів.

Далі виконується код програми управління. Ця програма працює з копією значень входів і виходів, розташованих в оперативній пам’яті. Якщо програма не завантажена або зупинена, то ця фаза робочого циклу не виконується. У тому випадку, якщо використовується режим налаштування, система програмування має доступ до образу входів-виходів, дозволяє управляти виходами вручну і проводити дослідження роботи датчиків.

Після виконання призначеного для користувача коду фізичні виходи ПЛК приводяться у відповідність до розрахункових значень (фаза 4).

Обслуговування апаратних ресурсів полягає в забезпеченні роботи системних таймерів, годинника реального часу, оперативного самотестування, індикації стану та інших апаратно-залежних завдань.

Монітор системи виконання містить велику кількість функцій, необхідних у процесі налаштування програми і забезпечення взаємодії з системою програмування, сервером даних і мережі. У функції системи виконання зазвичай входить:

– завантаження коду програми в оперативну і електрично перепрограмовану пам’ять;
– управління послідовністю виконання завдань;
– відображення процесу виконання програм;
– покрокове виконання;
– забезпечення перегляду і редагування значень змінних;
– фіксація і трасування значень змінних;
– контроль часу циклу тощо.

Призначена для користувача програма працює тільки з миттєвою копією входів. Таким чином, значення входів у процесі виконання призначеної для користувача програми не змінюються в межах одного робочого циклу. Це фундаментальний принцип побудови ПЛК скануючого типу. Такий підхід виключає неоднозначність алгоритму обробки даних у різних його гілках. Крім того, читання копії значення входу з ОЗП виконується значно швидше, ніж пряме зчитування входу. Апаратно читання входу може бути пов’язане з формуванням певних часових інтервалів, передачею послідовності команд для конкретної мікросхеми або навіть запиту мережею.

Необхідно зазначити, що не завжди робота зі зчитування входів повністю локалізована у фазі зчитування входів. Наприклад, АЦП зазвичай вимагає

певного часу з моменту запуску до зчитування виміряного значення. Частину роботи системне програмне забезпечення контролера виконує за перериваннями. Грамотно реалізована система виконання ніде і ніколи не використовує порожні цикли очікування готовності апаратури. Для прикладного програміста усі ці деталі не важливі. Значення має лише те, що значення входів оновлюються автоматично виключно на початку кожного робочого циклу.

Загальна тривалість робочого циклу ПЛК називається часом сканування. Час сканування значною мірою визначається тривалістю фази коду, призначеної для користувача програми. Час, зайнятий іншими фазами робочого циклу, практично є величиною постійною. Для задання середнього об'єму в ПЛК з системою виконання CODESYS час розподілиться приблизно так: 98% призначена для користувача програма, 2% – усе інше.

Існують завдання, в яких «плаває» час циклу, що істотно впливає на результат. Наприклад, це автоматичне регулювання. Для усунення цієї проблеми в сучасних ПЛК передбачений контроль часу циклу. Якщо окремі гілки коду керуючої програми виконуються дуже швидко, в робочий цикл додається штучна затримка. Якщо контроль часу циклу не передбачений, подібні завдання доводиться вирішувати виключно таймерами.

2.7.3 Особливості функціонування ПЛК як керуючого пристрою промислового призначення

Процес управління промисловим устаткуванням за допомогою ПЛК зводиться, як правило, до розв'язання логічних рівнянь, при цьому необхідно чітко розуміти, які функції ПЛК, що реалізовуються, слід вважати вихідними і які стани керованого об'єкта, елементів системи управління і вихідні сигнали ПЛК можуть або мають розглядатися як аргументи вказаних функцій.

До вихідних функцій насамперед належать функції увімкнення/вимкнення (запуску/зупинки) виконавчих механізмів керованого об'єкту. До вихідних функцій належать також функції запуску-скидання таймерів і лічильників, увімкнення-вимкнення внутрішніх операторів, які використовуються як для запам'ятовування імпульсних вхідних сигналів, так і проміжних станів керуючої програми, а також проміжних результатів, що отримуються для відпрацьовування ділянок програми. Крім цього, до вихідних належать також функції, пов'язані з організацією зон ігнорування ділянок програми, з

організацією умовних і безумовних переходів у програмі з обнуленням і без обнулення при цьому вказаних вище вихідних функцій.

Аргументами логічних функцій, що реалізовуються в ПЛК, є сигнали від органів управління, від датчиків положення виконавчих механізмів, з систем управління суміжним технологічним або транспортним устаткуванням, з систем верхнього або нижнього рівня (з використанням ПЛК в ієрархічних керуючих системах). Крім того, як аргументи логічних функцій використовуються сигнали з виходів таймерів, лічильників, стану внутрішніх операторів програми.

Однією з найважливіших особливостей функціонування ПЛК як керуючого пристрою промислового призначення є те, що на відміну від усіх керуючих пристроїв попередніх поколінь, в ПЛК один і той самий аргумент (стан входу контролера) може бути використаний на різних етапах програми довільну кількість разів (що рівнозначно релейно-контактному апарату з безкінечною кількістю контактів).

Зазначена особливість належить і до вихідних функцій, для «розмноження» яких не потрібні традиційні блок-контакти пускачів або їх твердотілі аналоги. Подібним же чином у ході реалізації того або іншого алгоритму в ПЛК є можливість багаторазового використання в програмі одного й того самого таймера, лічильника, а також звернення до будь-яких проміжних значень таймерів і лічильників.

Практично в усіх ПЛК програмно можна задати увімкнення його виходів або внутрішніх операторів як з пам’яттю (фіксацією стану після зникнення умов увімкнення), так і без неї.

У ході реалізації функцій увімкнення-вимкнення механізмів програмним шляхом не має значення, яким сигналом здійснюється запуск механізму «1» або «0» у той час, як в ході реалізації такої операції за допомогою релейної апаратури необхідно щоразу вводити додаткове реле, що виконує функцію інвертора.

У ПЛК використовується як енергозалежна, так і енергонезалежна пам’ять. Очевидно, що для нормальної роботи ПЛК поточні стани виходів, таймерів, лічильників і внутрішніх операторів запам’ятовуються в оперативній пам’яті, яка може бути як енергозалежною, так і енергонезалежною (з підживленням від акумуляторів). Використання в даному випадку енергозалежної пам’яті має як недоліки, так і переваги.

З одного боку, відключення електроживлення може призводити до втрати значного обсягу інформації. Втім з іншого, енергозалежна пам'ять може ефективно виконувати роль «нульового» захисту. Це запобігає прямому (без урахування заданої послідовності підключення механізмів і усіх необхідних блокувальних залежностей) повторному увімкненню механізмів у ході відновлення подання електроживлення.

Функції таймера (реле часу) в ПЛК реалізуються різними способами. Є моделі ПЛК, в яких функції таймера реалізуються апаратним шляхом, при цьому таймер підключається послідовно з певним виходом ПЛК. На таких таймерах часові уставки задаються за допомогою регуляторів або програмних задатчиків (наприклад, програмних перемикачів барабанного типу) вручну. Таке рішення є дуже зручним під час налагоджування і налаштування технологічного устаткування, оскільки дозволяє оперативно змінювати часові уставки без зміни керуючої програми.

Організація таймерів у ПЛК програмним шляхом є гнучкішим і більш універсальним рішенням, тому що в цьому випадку немає необхідності у жорсткій (монтажній) прив'язці таймера до конкретного виходу і є можливість програмного вибору типу часової функції (затримка на увімкнення, затримка на вимкнення тощо). Крім того, програмно можна задавати й різні умови запуску і скидання таймера. Так, наприклад, програмно можна задати або не задати умову, за якої запущений таймер скидається в початковий стан, якщо з певних причин зникли умови його запуску в період часу, що відповідає запрограмованій часовій затримці.

По-різному виконується і скидання таймера в початковий (стартовий) стан. У багатьох моделях ПЛК відлік часу заданої уставки здійснюється від уставки до нуля, тому в таких таймерах операції примусового скидання сенсу не має. У тих самих ПЛК, де в таймерах відлік уставок виконується від нуля до уставки, необхідно вводити команди примусового обнулення таймерів.

Дуже істотною для ПЛК як керуючого пристрою промислового призначення є функція рахунку кількості подій, що відбуваються на керованому об'єкті (наприклад, кількості виготовленої продукції). Проте, лічильники в ПЛК можуть встановлюватися і для підрахунку кількості внутрішніх подій (наприклад, кількості циклів відпрацьовування окремих ділянок програми).

Лічильники, як і таймери, можуть бути реалізовані й апаратно, і програмно, можуть працювати як ті, що підсумовують, віднімають і як реверсивні. Як правило, під час використання в ПЛК лічильників усі вони обнуляються за допомогою спеціальної команди «Скидання лічильника».

На відміну від таймера (програмна реалізація якого здійснюється виключно за рахунок внутрішніх ресурсів ПЛК) для рахунку кількості зовнішніх подій хоча б один зі входів контролера має бути задіяний для прийому імпульсів від датчика рахунку, встановленого на технологічному устаткуванні.

У зв'язку з тим, що лічильники в ПЛК виконують підрахунок імпульсів, які надходять до ПЛК із зовнішніх електричних ланцюгів, то існує досить серйозна проблема забезпечення точності роботи лічильників. З цією метою прийом одиничного імпульсу в лічильник фіксується не менше, ніж за двома станами (імпульс, пауза). Крім того, в каналах рахунку імпульсів використовується спеціальна схема (або її програмний аналог) виділення поодинокого імпульсу, що дозволяє позбутися «брязкоту» контактів і багатьох видів імпульсних перешкод.

Інші схемні рішення, що підвищують достовірність передачі імпульсів рахунку від джерела інформації до входів ПЛК, як правило, реалізуються в місці їх формування (у датчиках, перетворювачах), тому на входи ПЛК імпульси рахунку надходять зазвичай вже у «відфільтрованому» вигляді. Проте, в самих ПЛК також застосовують різні програмні засоби, що забезпечують підвищення достовірності інформації, яка приймається, і точності рахунку, що здійснюється програмно організованими лічильниками.

З метою підвищення швидкодії, а також надійності функціонування ПЛК у них застосовується спеціальна операція – пропускання ділянки програми, яка в цьому випадку називається «Зоною ігнорування». Сенс організації зон ігнорування і можливість програмного їх пропускання полягає в наступному. Під час відпрацювання окремих частин програми (тобто на певних кроках технологічного циклу), інші частини програми можуть бути опущені (проігноровані), що і дозволяє істотно скоротити тривалість циклу сканування частини програми, з якою активно взаємодіє процесор на цьому кроці технологічної циклограми, що призводить до підвищення як швидкодії ПЛК, так і надійності його функціонування.

Використання зон ігнорування дозволяє спростити структуру програми. У багатьох моделях ПЛК з метою підвищення надійності їх функціонування

розгалуження програми з використанням умовних і безумовних переходів ефективно замінюють керуючою програмою, що не містить розгалуження, з введенням необхідної кількості зон ігнорування.

Під час використання ПЛК у промислових об'єктах особливе значення має автоматизація діагностування несправностей як у каналах вводу/виводу інформації (у датчиках, виконавчих механізмах і лініях зв'язку), так і в самому контролері. І в цьому випадку в ПЛК не використовується додаткова апаратура, оскільки абсолютна більшість проблем, пов'язаних з діагностикою і відповідним аварійним відключенням ПЛК і технологічного устаткування, вирішується програмним шляхом. Дуже часто обсяг пам'яті, який займається діагностичними програмами і програмами аварійної зупинки, значно перевищує обсяг пам'яті основної керуючої програми.

2.7.4 Схемотехніка модулів вводу/виводу

Однією з основних функцій, що реалізовуються модулями вводу/виводу, є надійна гальванічна розв'язка внутрішніх (низьковольтних і слабкострумових) електричних ланцюгів контролера від зовнішніх (високовольтних і сильнострумових) електричних ланцюгів керованого об'єкта, які є ще й джерелами потужних електромагнітних перешкод, в умовах яких без гальванічної розв'язки контролер стійко функціонувати не може. Існує багато різних варіантів технічної реалізації схем гальванічних розв'язок входів і виходів ПЛК для ланцюгів постійного і змінного струму.

Нижче наведені узагальнені типові електричні схеми розв'язок, використовуваних у ПЛК. Простим і найдешевшим є релейний варіант гальванічної розв'язки входів (рис. 2.14), виконаної на мініатюрному герконовому реле в корпусі, подібному до корпусу мікросхеми, і встановлюваному безпосередньо на друкованій платі модуля входів. Живлення обмоток реле може здійснюватися як постійним, так і змінним (через діод) струмом. $К_д$ – контакт датчика (органу управління) керованого об'єкта. Для сигналізації збудженого стану входу використовується світлодіод V_i, що встановлюється на лицьовій панелі модуля входів.

Єдиним недоліком цієї простої схеми є лише обмежений ресурс реле, проте в системах програмного управління нескладним технологічним устаткуванням з малою частотою спрацьовування виконавчих механізмів ця схема досить часто й ефективно застосовується на практиці.

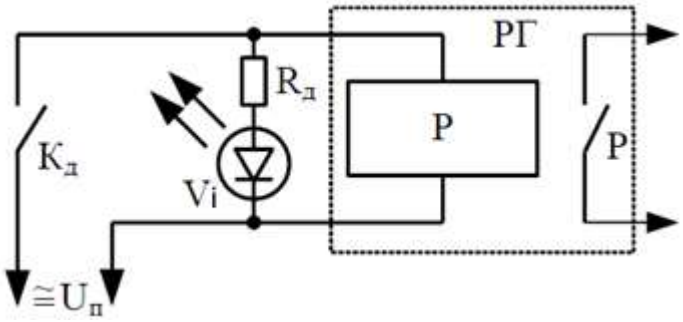


Рисунок 2.14 – Релейна гальванічна розв’язка вхідного ланцюга

Для вхідних ланцюгів постійного струму широко застосовується оптронна гальванічна розв’язка (ОГР), основний елемент якої – транзисторна або діодна оптопара, яка складається зі світлодіода (випромінювача) і фототранзистора (рис. 2.15).

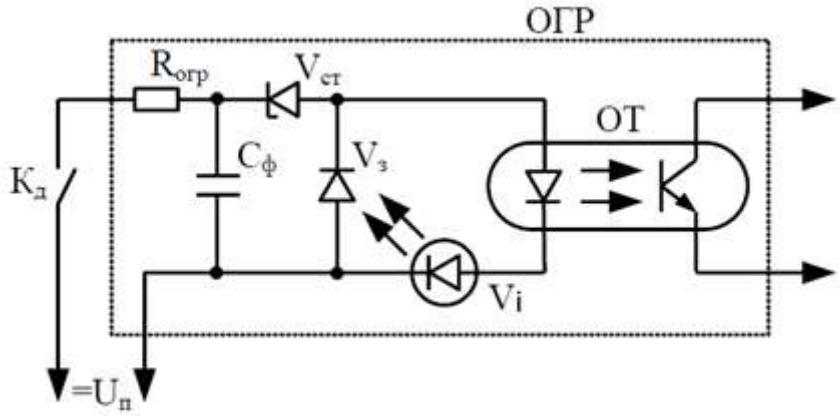


Рисунок 2.15 – Оптронна гальванічна розв’язка
вхідного ланцюга постійного струму ПЛК

$R_{обм}$ обмежує вхідний струм оптрона (у межах від 14 до 16 мА). $V_{ст}$ встановлює поріг спрацьовування вхідного ланцюга за напругою, що дуже важливо в процесі підключення до ПЛК безконтактних датчиків з великим рівнем залишкової напруги. V_3 забезпечує захист від пробою випромінювача оптопари і світлодіода V_i за помилкового порушення полярності вхідного сигналу. $R_{обм}$ і $C_ф$ утворюють фільтр для захисту вхідного ланцюга від короткочастотних імпульсних перешкод.

Для вхідних ланцюгів змінного струму застосовують такі самі схеми, але з введенням у них випрямляча V змінного струму в постійний (рис. 2.16).

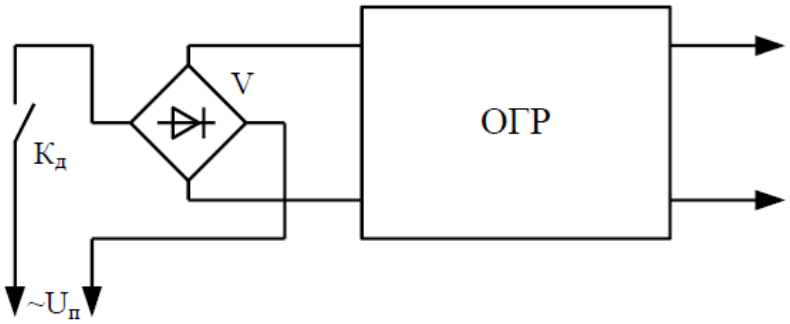


Рисунок 2.16 – Оптронна гальванічна розв’язка
вхідного ланцюга змінного струму

У вихідних ланцюгах постійного струму застосовується транзисторний оптрон, що забезпечує не лише гальванічне розділення, але й посилення вихідного сигналу ПЛК (рис. 2.17).

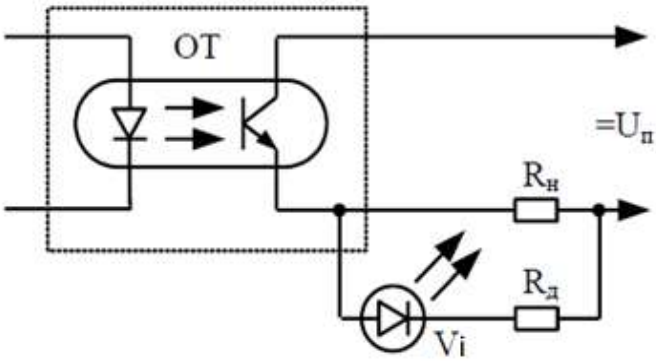


Рисунок 2.17 – Гальванічна розв’язка вихідного ланцюга
з використанням транзисторного оптрона

У ланцюгах змінного струму застосовується така сама схема, але доповнена, наприклад, симістором (рис. 2.18).

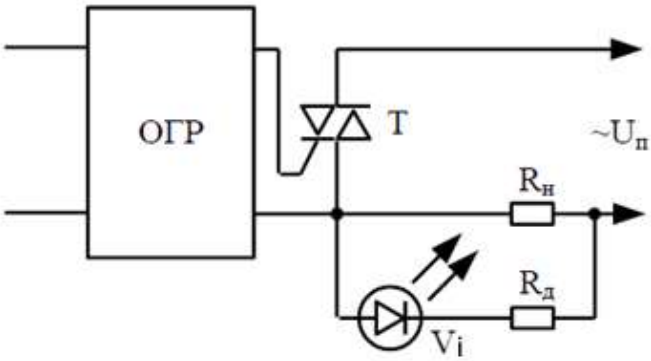


Рисунок 2.18 – Схема вихідного каналу
з гальванічною розв’язкою і підсилювальним симістором

Універсальним є варіант подвійної гальванічної розв’язки – оптронної, на вході якої встановлюється, наприклад, потужне герконове реле (РГ), що забезпечує комутацію ланцюгів управління виконавчих механізмів як для постійного струму, так і змінного струму (рис. 2.19).

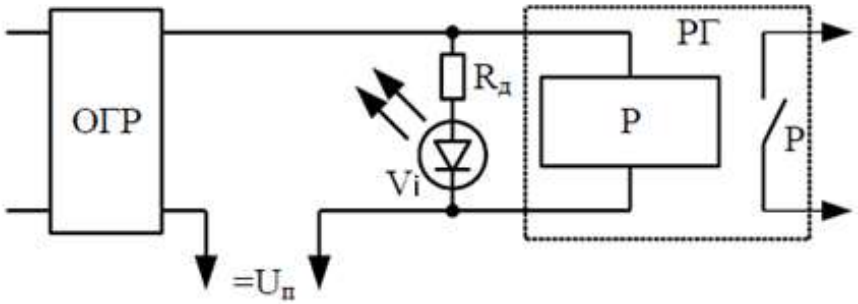


Рисунок 2.19 – Схема подвійної гальванічної розв’язки вихідного ланцюга

Вибір модулів входів-виходів з певними варіантами гальванічної розв’язки здійснюється користувачем під час проектування систем програмного управління конкретним технологічним устаткуванням.

У процесі проектування систем промислової автоматики необхідно мати на увазі одну важливу особливість електричного стикування органів управління та інших контактних пристроїв, що підключаються до ПЛК, з вхідними ланцюгами модулів вводу/виводу. Струм навантаження вхідного ланцюга модуля вводу/виводу складає 14…16 мА, а багато органів управління промислового призначення (кнопки, тумблери тощо) та інші релейно-контактні пристрої, що підключаються до входів ПЛК, гарантують надійний електричний контакт зі струмом навантаження не менше 100 мА. У зв’язку з цим багато моделей ПЛК доукомплектовувалися спеціальними блоками додаткових резисторів, що підключаються за схемою, наведеною на рис. 2.20.

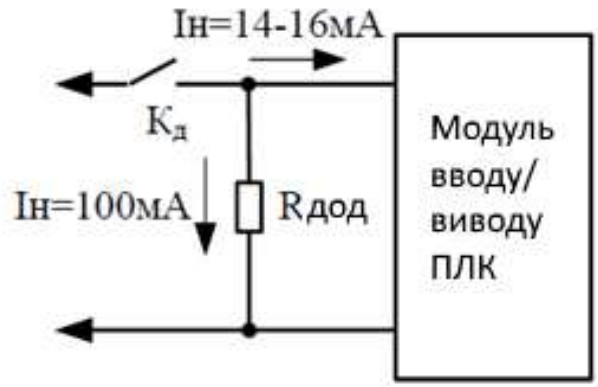


Рисунок 2.20 – Схема введення додаткового (навантаження) резистора

Разом з простими елементами вводу/виводу дискретних сигналів у ПЛК можуть використовуватися складніші пристрої для обробки й перетворення вхідних сигналів, формування різних затримок часу, підрахунку імпульсів, зв'язку з іншими ПЛК або комп'ютерами. До складу сучасних ПЛК можуть входити багатоканальні аналого-цифрові або цифро-аналогові перетворювачі. Крім того, розроблені різні модулі управління виконавчими механізмами з багатокоординатним управлінням.

2.8 Вимоги до надійності експлуатації систем програмного управління на основі ПЛК

2.8.1 Способи і засоби підвищення надійності функціонування процесора

Оскільки процесор є центральним керуючим пристроєм системи з програмним управлінням (СПУ), то вихід його з ладу або помилки і збої в його роботі можуть призводити не лише до відмов у роботі технологічних агрегатів, але й до аварійних ситуацій на керованому об'єкті. Тому забезпеченню надійності функціонування процесора завжди має приділятися первинне значення. Відомі методи підвищення надійності функціонування процесора умовно можна поділити на апаратні, програмні та структурні.

До апаратних методів підвищення надійності функціонування процесора належать:

– забезпечення безперебійного електроживлення (при цьому можуть використовуватися акумуляторні батареї або резервні мережні джерела живлення);

– тестування функціонування апаратної частини процесора, у тому числі і з використанням «сторожового» елемента, який контролює тривалість відпрацьовування команди і видає сигнал «несправність» з перевищенням максимально допустимої тривалості циклу виконання команди.

У сучасних ПЛК велика частина самодіагностики процесора реалізується програмним шляхом. З увімкненням ПЛК виконується тестування внутрішнього (системного) програмного забезпечення (за спеціальними тестовими програмами), пам'яті даних (наприклад, за контрольними сумами), інтерфейсів вводу/виводу, функціональних модулів ПЛК, фрагментів програми (програм користувачів).

У процесі роботи ПЛК багаторазово (циклічно) здійснюються такі перевірки: роботи інтерфейсів вводу/виводу, функціонування усіх модулів ПЛК, а також контроль функціонування центрального процесора.

До структурних належать методи, спрямовані на вдосконалення архітектури СПУ на базі ПЛК, а саме:

– застосування архітектури з потрійним резервуванням ПЛК;

– застосування дуальної архітектури (архітектури з резервуванням);

– застосування комбінованої та оригінальної архітектури підвищеної живучості.

Потрійне резервування архітектури з використанням мажоритарного принципу (перевірка за збігом двох з трьох) досить широко використовується у бортових системах управління літальними апаратами, в ядерній енергетиці, на залізничному транспорті, проте потрійне резервування призводить до істотного ускладнення систем та їх значного дорожчання, тому у промислових системах управління загального призначення, побудованих на ПЛК, застосовується вкрай рідко.

Частіше за все під час управління складними технологічними комплексами використовується дуальна архітектура (рис. 2.21), що містить два ПЛК, один з яких є ведучим (головним), а інший – веденим (допоміжним), тому така архітектура називається асиметричною.

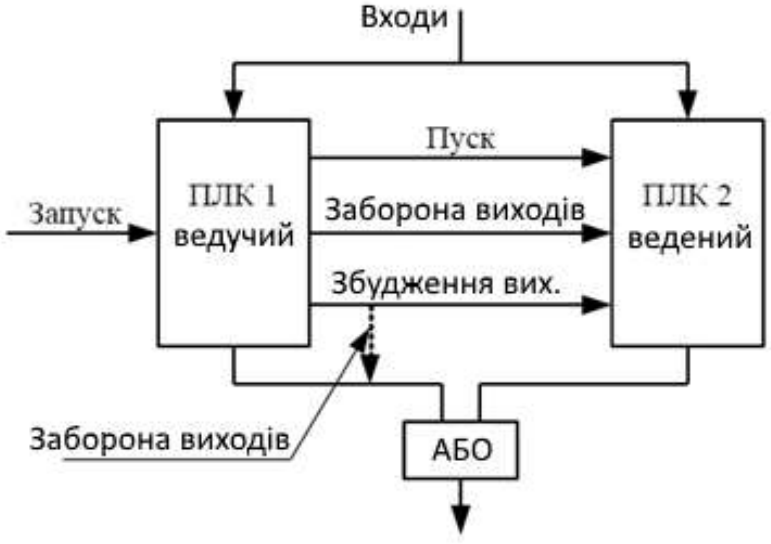


Рисунок 2.21 – Дуальна структура ПЛК

Входи системи (інформаційні сигнали від датчиків, органів управління) підключаються водночас до входів обох ПЛК, виходи системи (команди

управління механізмами) видаються через схему «АБО». У ЗП обох ПЛК записуються однакові керуючі програми. Запуск ведучого ПЛК здійснюється користувачем, а запуск веденого – ведучим за допомогою спеціальної команди. Виходи системи спочатку збуджуються тільки ведучим ПЛК, виводи веденого при цьому заблоковані.

Якщо ведучий ПЛК виявляє збій у своїй роботі, то він генерує спеціальне (аварійне) слово-стан, за яким відключаються його виходи від схеми «АБО» і водночас активізуються виходи веденого ПЛК, який бере на себе управління об’єктом.

Оригінальною є архітектура, використовувана фірмою Alsthom (Франція), в якій система ПЛК складається з двох однакових структур зв’язку і двох блоків відпрацьовування, які сполучені між собою блоком прийняття рішень БПР (рис. 2.22).

Сигнали управління на виходи видаються тільки тоді, коли вони однакові в обох ПЛК. У випадку незбіжності за допомогою тестування локалізується несправна структура, відповідний контролер тимчасово відключається. Об’єкт продовжує функціонувати під управлінням справного ПЛК. Ремонт несправності може здійснюватися без зупинки об’єкта. За повторного запуску відремонтованого ПЛК здійснюється і його автоматична синхронізація.

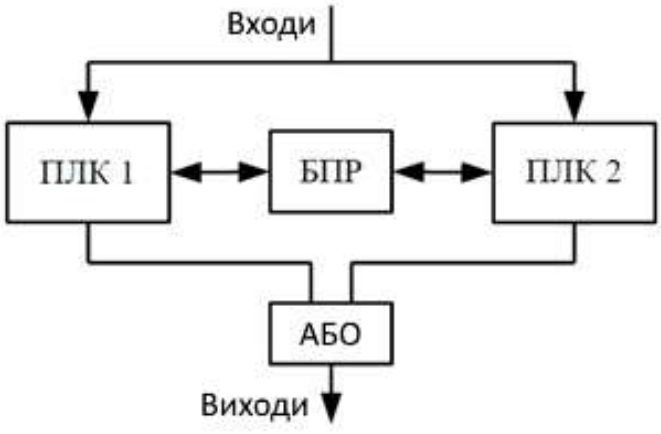


Рисунок 2.22 – Двоканальна архітектура з блоком прийняття рішень

2.8.2 Способи підвищення достовірності отримання, обробки і зберігання інформації

Одним з найпоширеніших способів підвищення достовірності, які використовуються в ПЛК, є контроль інформації за парністю (непарністю). У цьому випадку під час формування або передачі інформації в канал зв’язку до

коду повідомлення (слова), що передається, додається (або ні) одиниця для створення в коді повідомлення парної (непарної) кількості одиниць, а в ході прийому інформації перевіряється наявність цього парного (непарного) числа одиниць. Такий контроль дозволяє з дуже високою мірою достовірності виявляти поодинокі помилки в повідомленнях, що передаються.

У зв'язку з тим, що обсяг пам'яті, займаний керуючими програмами, нестримно зростає, з'явилася необхідність у застосуванні методів контролю, що дозволяють виявляти дві і більше помилок у повідомленні, для чого до інформаційних біт додається не один, а декілька (група) контрольних бітів. Крім того, для виявлення двох і більше помилок у повідомленнях застосовують також коригувальні коди.

Досить широко використовується у ПЛК і метод контрольної суми. У цьому випадку пакет повідомлень, що містить велику кількість числових даних, зберігається, або передається каналами зв'язку спільно з контрольною сумою цих даних, яка перевіряється щоразу перед використанням даних.

Дуже ефективним є також простий і надійний спосіб контролю інформації, що полягає у багатократному (циклічному) виконанні операцій з перевіркою на збіг результатів окремих опитувань або обчислень. Ефективність застосування цього способу останнім часом різко зросла у зв'язку з появою нових швидкодіючих мікропроцесорів, використання яких в ПЛК істотно збільшило часовий резерв під час вирішення завдань управління технологічними процесами.

2.8.3 Надійність системи входів-виходів ПЛК

Досвід практичного використання ПЛК на виробництві показав, що від 80 до 100% усіх несправностей контролерів припадає на модулі входів-виходів. При цьому усі несправності, що мають місце, можна розділити на три групи:

– несправності, пов'язані з вхідними ланцюгами модулів входів;

– несправності, пов'язані з вихідними ланцюгами модулів виходів;

– несправності, пов'язані з переробкою інформації в модулях входів-виходів.

Несправності у вхідних ланцюгах модулів входів-виходів, як правило, є наслідком некваліфікованого підключення, обслуговування і ремонту. Типовими причинами відмов є:

– подання на входи ПЛК підвищеної напруги;

– порушення полярності вхідних сигналів;

– підключення до вхідних ланцюгів постійного струму джерел змінної напруги;

– прокладення вхідних інформаційних каналів зв'язку (дротів, джгутів, кабелів) у безпосередній близькості із зварювальним устаткуванням, що є джерелом потужних електромагнітних перешкод;

– підключення до входів ПЛК контактних апаратів (зокрема, органів управління), для яких не гарантується якісний електричний контакт під час комутації малих вхідних струмів ПЛК (13...16 мА).

Запобігання вказаним несправностям у більшості випадків забезпечується кваліфікованим обслуговуванням ПЛК з дотриманням усіх вказівок і рекомендацій, наведених в інструкціях з експлуатації. Хоча для запобігання виходу з ладу елементів вхідних ланцюгів і насамперед оптопар усі вхідні ланцюги ПЛК мають відповідні захисти (обмеження напруги на вході оптопари, захист від порушення полярності вхідного сигналу тощо).

Несправності у вихідних ланцюгах модулів входів-виходів, як правило, аналогічні несправностям у вхідних ланцюгах (як за причинами їх виникнення, так і за методами їх запобігання).

Для запобігання відмовам і збоям, пов'язаним з обробкою інформації в модулях входів-виходів, застосовуються описані нижче способи.

Перевірка джерел вхідних сигналів здійснюється шляхом *n*-кратного опитування: спочатку виконується двократне опитування, у випадку незбіжності результатів опитування додатково здійснюється п'ятикратне опитування, а за негативного результату і цього контролю вхід вважається несправним.

Аналогічно організовується перевірка достовірності вихідних сигналів, що видаються. Елемент, що отримав вихідну інформацію, повертає її в центральний блок (процесор), який порівнює її з виданою. У випадку незбіжності даних операція порівняння повторюється п'ять разів і за негативного результату вихід залишається в попередньому стані або скидається в нуль.

У деяких ПЛК частина виходів спеціально з'єднується з частиною входів, і з виходів на входи циклічно подається відома послідовність сигналів (символів), яка порівнюється з еталонною. У деяких випадках використовують резервування найбільш відповідальних входів і виходів ПЛК.

2.8.4 Запобігання аварійним ситуаціям на керованому об’єкті

Забезпечення безпеки управління технологічним процесом завжди було і буде однією з найважливіших вимог до усіх керуючих систем, у тому числі й систем програмного управління, побудованих на основі ПЛК.

Запобігання аварійним ситуаціям на керованому об’єкті досягається за рахунок комбінованого використання програмних і апаратних засобів.

Програмне забезпечення практично усіх ПЛК разом з програмами, що управляють, обов’язково має містити програми автоматичної діагностики та аварійного відключення або аварійного перемикання на напівавтоматичний або ручний режим, причому аварійні перериваючі програми мають найвищий пріоритет і за допомогою контролера переривань практично миттєво реалізують аварійну зупинку керованого об’єкта.

Крім того, з метою забезпечення можливості гарантованої зупинки технологічного агрегата навіть за повного виходу з ладу процесора ПЛК або серйозних збоїв у роботі програмного забезпечення в СПУ з використанням ПЛК передбачають наявність ввідного автоматичного вимикача з дистанційним відключенням безпосередньо від кнопки аварійної зупинки.

2.9 Контрольні запитання та завдання

1. Що таке ПЛК? Поясніть принцип роботи ПЛК.
2. Які особливості використання ПЛК на виробництві можете назвати?
3. Які види пам’яті застосовуються в ПЛК?
4. Перелічіть основні типи ПЛК. Як і на якому етапі вибирається тип ПЛК?
5. Поясніть, як організовані входи та виходи ПЛК.
6. Які існують обмеження на використання ПЛК?
7. Які існують типи режиму реального часу?
8. Які ви можете назвати обмеження, що виникають у ході використання режиму реального часу в ПЛК?
9. Як інтегрувати ПЛК в систему управління підприємством?
10. Як використовують ПЛК під час створення автоматизованої системи управління технологічними процесами?
11. Наведіть типову функціональну схему дворівневої АСУ ТП.
12. Що таке робочий цикл ПЛК?

13. Поясніть особливості функціонування ПЛК як керуючого пристрою промислового призначення.

14. Як схематично організовані модулі вводу-виводу ПЛК?

15. Поясніть вимоги до надійності експлуатації систем програмного управління на основі ПЛК.

16. Назвіть способи підвищення достовірності отримання, обробки і зберігання інформації.

17. Як запобігати аварійним ситуаціям на керованому об’єкті?

3 ХАРАКТЕРИСТИКИ ПРОГРАМОВАНИХ ЛОГІЧНИХ КОНТРОЛЕРІВ ОВЕН

3.1 ПЛК100 контролер для малих систем автоматизації з DI/DO

ОВЕН ПЛК100 – моноблочний контролер з дискретними входами/виходами на борту для автоматизації малих систем.

Призначення контролера ОВЕН ПЛК100:

– створення систем управління малими і середніми об’єктами;

– побудова систем диспетчеризації.

На рис. 3.1 показаний зовнішній вигляд ПЛК100.

Рисунок 3.1 – Зовнішній вигляд ПЛК100

Особливості ОВЕН ПЛК100:

– компактний DIN-корпус;

– вбудовані дискретні входи/виходи;

– наявність послідовних портів (RS-485, RS-232) і Ethernet;

– розширення кількості точок вводу/виводу здійснюється шляхом підключення зовнішніх модулів вводу/виводу за будь-яким з вбудованих інтерфейсів;

– два варіанти живлення: 220 В змінного струму і 24 В постійного струму.

На рис. 3.2 показана структурна схема ПЛК100.

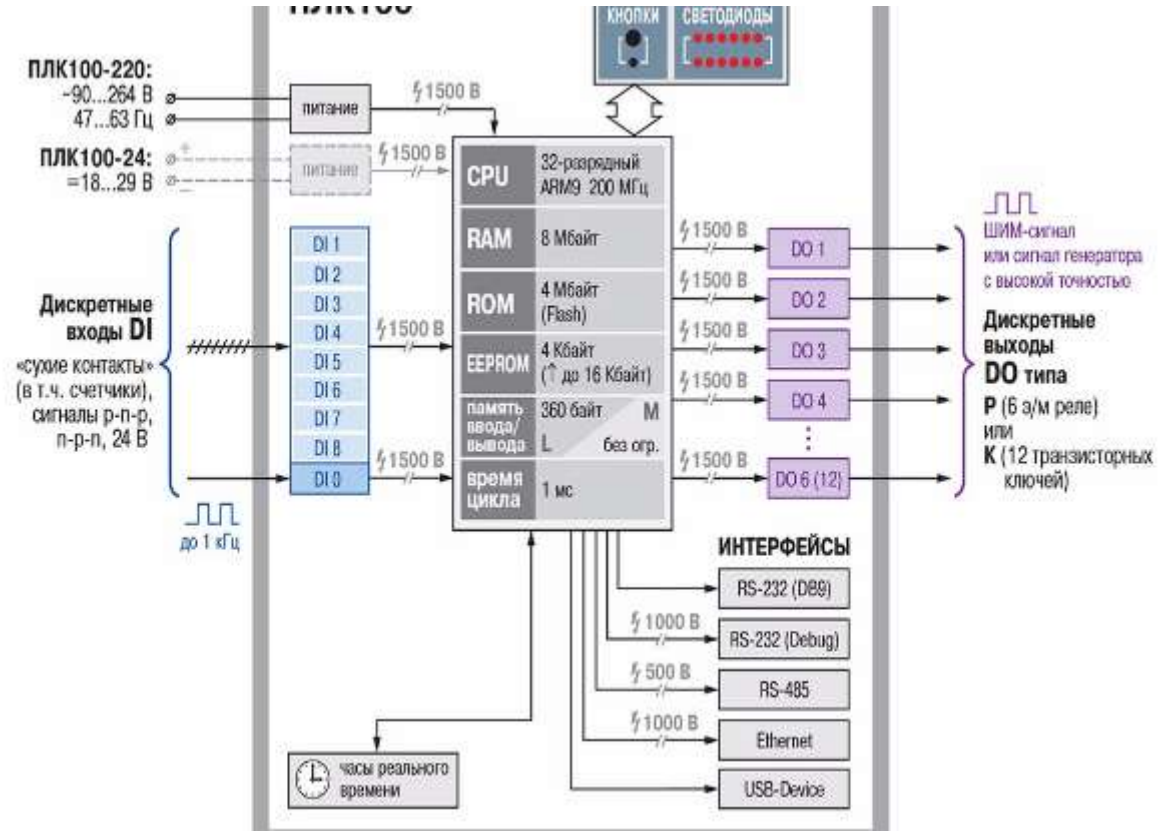


Рисунок 3.2 – Структурна схема ПЛК100

Як можна бачити з цього рисунку, ПЛК працює на основі 32-розрядного контролера, виконаного на ядрі ARM9 з тактовою частотою 200 МГц. ПЛК оснащений 8 Мб ОЗП і 4 Мб ПЗП (Flash) для зберігання програми управління. Також є 4 Кб енергонезалежної пам’яті EEPROM. Мінімальний час машинного циклу може досягати 1 мс.

ПЛК має 8 дискретних входів і 6 дискретних виходів релейного або транзисторного типу. В останньому випадку використовується шість здвоєних транзисторних ключів, що відповідає 12 вихідним сигналам.

Пристрій підтримує такі інтерфейси:

– RS-232 для підключення зовнішніх пристроїв;

– RS-232 для програмування пристрою;

– RS-485;

– Ethernet;

– USB.

Серед відмінних характеристик цього пристрою можна виділити:

– відсутність ОС, що підвищує надійність роботи контролерів;

– швидкість роботи дискретних входів – до 10 кГц під час використання підмодулів лічильника;

– велика кількість інтерфейсів на борту, працюючих незалежно один від одного: Ethernet, три послідовні порти, USB Device для програмування контролера;

– розширений температурний діапазон роботи: від -20 до +70 °С;

– вбудований акумулятор, що дозволяє «перечікувати» зникнення живлення: виконання програми зі зникненням живлення і переведення вихідних елементів в «безпечний стан»;

– вмонтований годинник реального часу;

– контролер підтримує роботу з нестандартними протоколами за будь-яким з портів, що дозволяє підключати такі пристрої як електро-, газо-, водолічильники, зчитувачі штрих-кодів тощо.

Програмування контролерів ОВЕН ПЛК100 здійснюється професійною системою програмування CODESYS v.2. Стислі технічні характеристики ПЛК100 зведено до таблиці 3.1.

Таблиця 3.1 – Технічні характеристики ПЛК100

Найменування параметра	Характеристика
Ресурси	
Центральний процесор	32-розрядний RISC-процесор 200 МГц на базі ядра ARM9
Обсяг оперативної пам’яті	8 Мб
Обсяг енергонезалежної пам’яті зберігання ядра CODESYS, програм і архівів	4 Мб**
Розмір Retain-пам’яти	4 Кб***
Час виконання циклу ПЛК	Мінімальний – 250 мкс (нефіксований), типовий – від 1 мс
Дискретні входи	
Кількість дискретних входів	8
Гальванічна розв’язка дискретних входів	є, групова
Електрична міцність ізоляції дискретних входів	1,5 кВ
Максимальна частота сигналу, що подається на дискретний вхід: – під час програмної обробки – із застосуванням апаратного лічильника – із застосуванням обробника енкодера	 1 кГц 10 кГц 1 кГц

Закінчення таблиці 3.1

Найменування параметра	Характеристика
Дискретні виходи	
Кількість дискретних виходів у: ПЛК100-24.Р і ПЛК100-220.Р ПЛК100-24.К	 6 е/м реле 6 здвоєних транзисторних ключів (всього 12 вихідних сигналів)
Гальванічна розв’язка дискретних виходів	є, індивідуальна
Електрична міцність ізоляції дискретних виходів	1,5 кВ
Інтерфейси зв’язку	
Інтерфейси	Ethernet 100 Base-T RS-232 – два канали RS-485 USB 2.0-Device
Швидкість обміну за інтерфейсами RS	від 4800 до 115200 bps
Протоколи, які підтримуються	ОВЕН Modbus RTU, Modbus ASCII DCON Modbus TCP GateWay (протокол CODESYS)
Програмування	
Середовище програмування	CODESYS 2.3.8.1 (і старше)
Інтерфейс для програмування і відладки	RS-232 USB-Device Ethernet

У таблиці 3.2 перераховані протоколи та інтерфейси, які підтримуються в даному пристрої.

Контролери ОВЕН ПЛК дозволяють організувати шлюз між приладами з протоколом ОВЕН (RS-485) і промисловими мережами з протоколами Modbus, Modbus TCP, DCON. Користувач має можливість реалізувати в середовищі програмування CODESYS власний протокол, який не підтримується ОВЕН ПЛК. В цьому випадку він може скористатися спеціальною бібліотекою, яка відкриває низькорівневий доступ до послідовних портів ОВЕН ПЛК (бібліотека входить до комплекту постачання контролера).

Таблиця 3.2 – Інтерфейси і протоколи, що підтримуються

Протокол	Інтерфейс	Застосування
ОВЕН	RS-232 RS-485	Підтримка модулів вводу/виводу лінійки ОВЕН Мх110 Робота в мережах ОВЕН спільно з ТРМ2хх
Modbus RTU Modbus ASCII	RS-232 RS-485	Підтримка модулів вводу/виводу - ОВЕН Мх110 і операторських панелей (ОВЕН СП307/СП310), зв'язок з SCADA-системами.
Modbus TCP	Ethernet 10/100 Мбіт/с	Передача даних на верхній рівень (у SCADA-системи).
DCON	RS-232 RS-485	Підтримка модулів вводу/виводу ICP DAS I – 7ххх, ADAM – 4хххх
GateWay (протокол CODESYS)	RS-232 Ethernet 10/100 Мбіт/с USB Device	Програмування контролера, відлагодження призначеної для користувача програми. Робота з OPC-сервером CODESYS GateWay Зв'язок з контролерами інших виробників на базі CODESYS.

Контролери ОВЕН ПЛК випускаються в двох модифікаціях за ліцензійним обмеженням розміру області пам'яті вводу/виводу (області %I+%Q+%M або області відображення процесу):

– ОВЕН ПЛК-Х.Х-М – контролери з обмеженням обсягу області пам'яті вводу/виводу до 25 Кб;

– ОВЕН ПЛК-Х.Х-L – контролери з ліцензійним обмеженням обсягу області пам'яті вводу/виводу до 360 байт.

Розташування портів вводу/виводу і елементів управління ПЛК100 показане на рис. 3.3. З обох боків контролера розташовані клеми для підключення дискретних датчиків і виконавчих механізмів.

Будь-який дискретний вхід ПЛК100 може працювати в режимі апаратного лічильника або тригера (частота до 10 кГц при шпаруватості 50%).

До двох дискретних входів можна підключити енкодер (частота імпульсів до 10 кГц). Частота обробки апаратних лічильників і обробників енкодера не залежить від часу виконання циклу ПЛК.

Вихід 11 ПЛК100-К може працювати як апаратний генератор заданої кількості імпульсів частотою до 10 кГц. Інші виходи контролера управляються з призначеної для користувача програми, тому частота управління ними пов'язана з часом виконання циклу ПЛК.

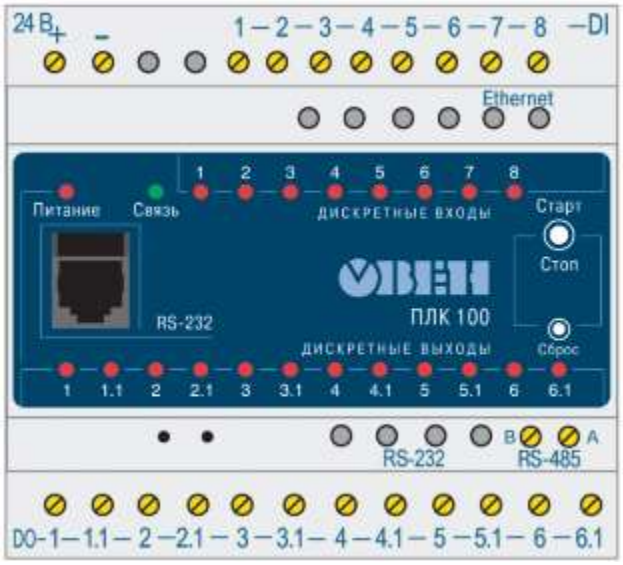


Рисунок 3.3 – Розташування портів вводу/виводу
і елементів управління ПЛК100

На передню панель контролера виведена світлодіодна індикація про стан дискретних входів і виходів («Дискретні входи», «Дискретні виходи»), про наявність живлення («Живлення») і про наявність зв’язку («Зв’язок») з середовищем програмування CODESYS.

Також на передній панелі є дві кнопки:

– кнопка «Старт/Стоп», призначена для запуску і зупинки програми в контролері, може бути використана як додатковий дискретний вхід контролера;

– прихована кнопка «Скидання», призначена для перезавантаження контролера. Натиснути кнопку «Скидання» можливо тільки тонким загостреним предметом.

Світлодіодна індикація слугує для відображення:

– наявності живлення;

– ознаки роботи програми;

– наявності зв’язку з середовищем програмування CODESYS;

– станів дискретних входів;

– станів дискретних виходів.

Залежно від модифікації пристрою існує декілька схем підключення.

У контролері передбачений малопотужний звуковий випромінювач, керований з призначеної для користувача програми як додатковий дискретний вихід. Звуковий випромінювач може бути використаний для аварійної або іншої сигналізації, або в процесі відлагодження програми. Частота звукового сигналу випромінювача фіксована і не підлягає зміні.

Контролер ПЛК100 оснащений вмонтованим годинником реального часу, що має власне акумуляторне джерело живлення. Енергії повністю зарядженого акумулятора вистачає на безперервну роботу годинника реального часу впродовж 6 місяців (за температури від +15 до +35 °С). У разі зношення акумулятора, не повної його зарядки, а також під час роботи за нижчих або більших температур час роботи годинника реального часу може скоротитися.

На рис. 3.4 показана схема підключення ОВЕН ПЛК100-24.К. Літера «К» в позначенні означає, що в контролері використовуються здвоєні транзисторні ключі. Згідно зі специфікацією, контролер може управляти дванадцятьма дискретними виходами на транзисторних ключах.

Максимальний струм навантаження на один канал не повинен перевищувати 150 мА. Ключі комутують плюс джерела живлення ($+U_{живл}$). Тому використовується негативний загальний провід. Комутована напруга – до 30 В постійного струму.

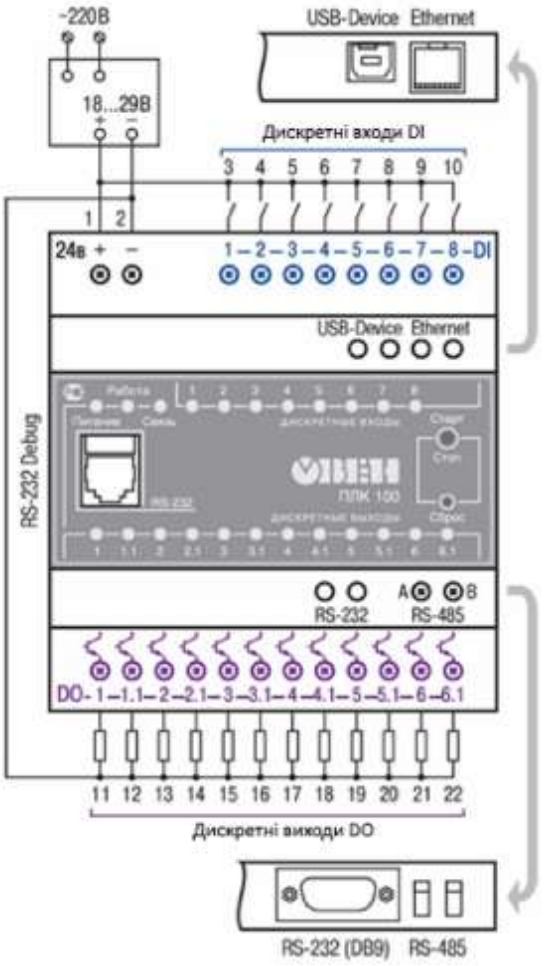


Рисунок 3.4 – Схема підключення ОВЕН ПЛК100-24.К

На рис. 3.5 показана схема підключення ОВЕН ПЛК100-24.Р. Для управління входами використовується електромагнітне реле. Дискретні виходи є нормально відкритими контактами реле, виведеними на клемники. Із спрацьовуванням реле контакти замикаються.

На рис. 3.5 показаний випадок з'єднання виходів за схемою із загальним проводом, проте виходи незалежні і можуть працювати кожен зі своїм навантаженням. Струм комутації дискретного виходу – до 4 А за напруги не більше 220 В, частотою 50 Гц і $\cos \phi > 0{,}4$. Напруга живлення цієї версії контролера 24 В постійного струму.

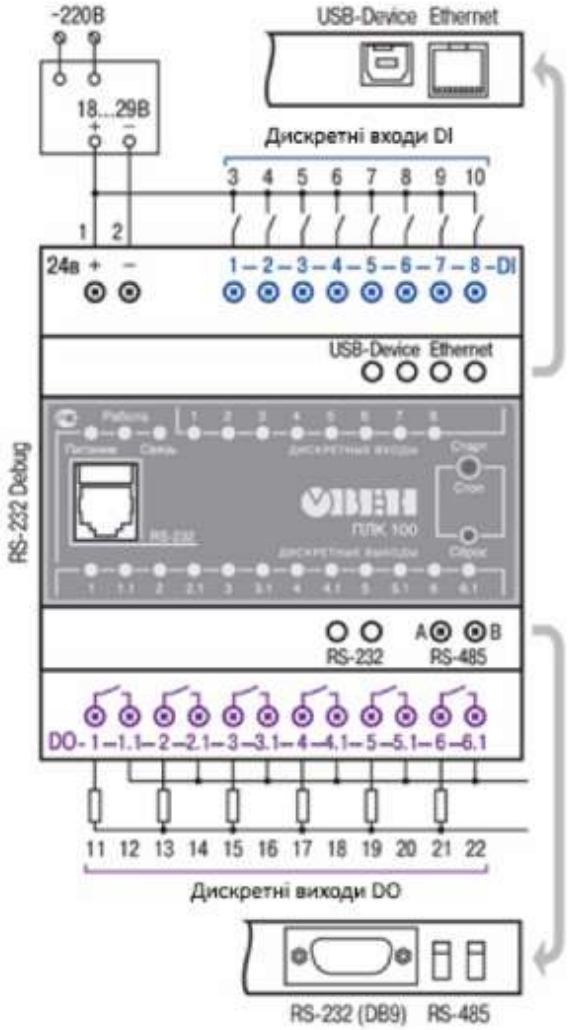


Рисунок 3.5 – Схема підключення ОВЕН ПЛК100-24.Р

На рис. 3.6 показана схема підключення ОВЕН ПЛК100-220.Р. Відмінністю цього контролера є вбудоване джерело живлення. Тому цей пристрій підключається безпосередньо до мережі 220 В змінного струму.

Цей контролер має дискретні виходи, що є сухими контактами реле. Аналогічно попередньому пристрою, струм комутації кожного дискретного виходу – до 4 А за напруги не більше 220 В, частоти 50 Гц і cos ϕ > 0,4.

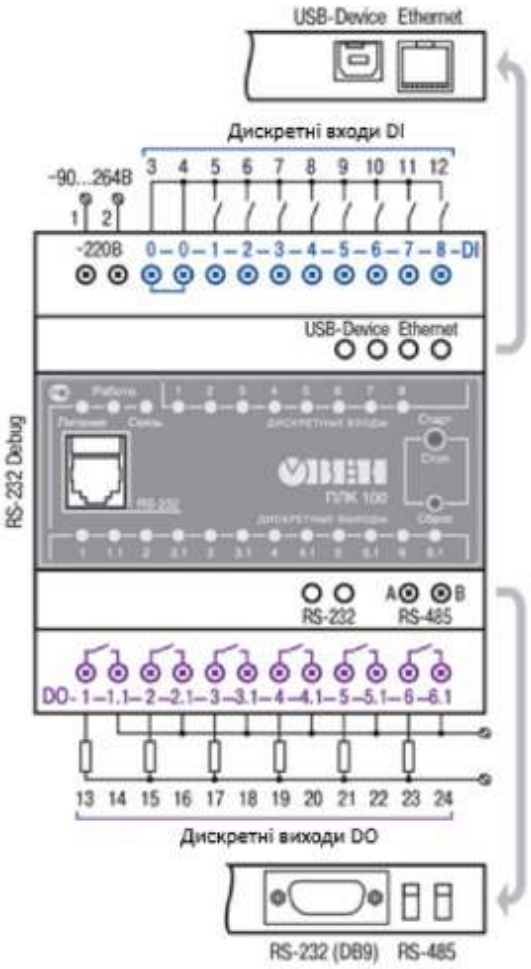
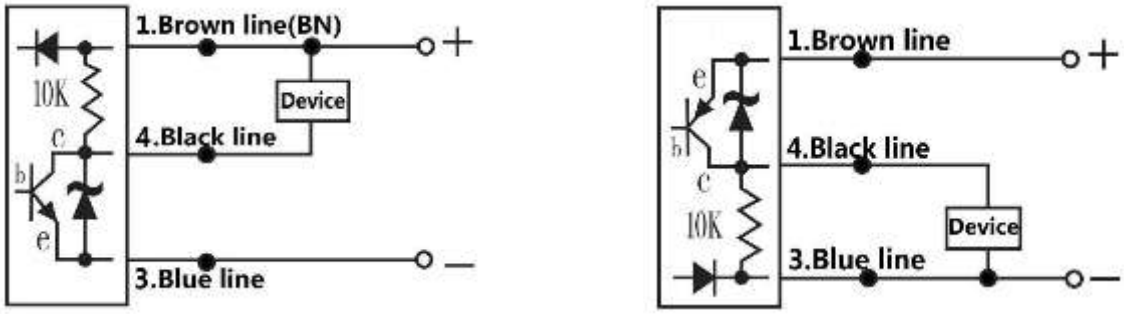


Рисунок 3.6 – Схема підключення ОВЕН ПЛК100-220.Р

Дискретні входи контролера ПЛК100 мають гальванічну розв'язку і виконані із застосуванням оптичних пар. Електрична міцність ізоляції дискретних входів 1,5 кВ. До дискретних входів ПЛК можуть підключатися як датчики з транзисторними ключами, так і з виходами типу «сухий контакт». На рис. 3.7 показана схема підключення дискретних входів ПЛК100.

Рисунок 3.7 – Схема підключення дискретних входів ПЛК100

До дискретних входів ПЛК підключаються різні датчики. Такими датчиками можуть бути, наприклад, індуктивний датчик наближення LJ12A3 -4-X-XX. Датчики випускаються в різних модифікаціях і мають різні вихідні ланцюги. На рис. 3.8 показані різні схеми вихідних ланцюгів індуктивних датчиків наближення LJ12A3 -4-X-XX.



а) N-P-N вихід датчика

б) P-N-P вихід датчика

Рисунок 3.8 – Схеми вихідних ланцюгів
індуктивних датчиків наближення LJ12A3 -4-X-XX

З підключенням до ПЛК датчиків, що мають на виході транзисторні ключі з відкритим колектором, використовується схема, зображена на рис. 3.9. Як видно з цього рисунку, загальним є негативний провідник джерела живлення.

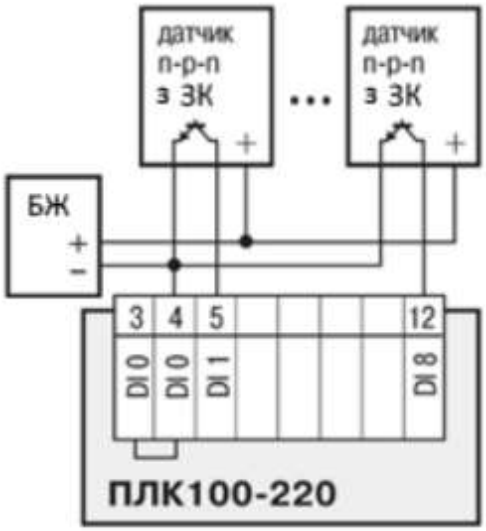


Рисунок 3.9 – Підключення датчиків з виходом типу N-P-N

Датчики, які комутують плюс джерела живлення і мають на виході транзисторні ключі з відкритим колектором, підключаються до ПЛК100 за схемою, яка зображена на рис. 3.10.

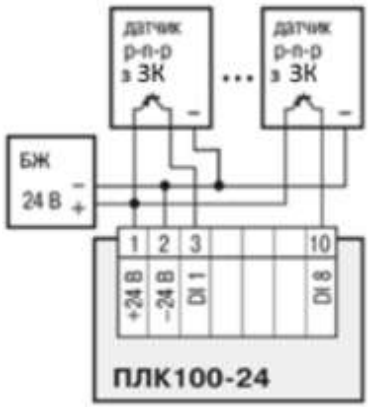


Рисунок 3.10 – Підключення датчиків з виходом типу P-N-P

Як видно з цього рисунку, загальним є позитивний провідник джерела живлення. Датчики підключаються за трипровідною схемою. Два контакти використовуються для живлення, а один – це вихід, відкритий колектор ключового транзистора.

Також використовуються схеми, в яких застосовується підтягувальний резистор у вихідному ланцюзі. Він використовується для підняття вихідного рівня до рівня напруги живлення за відсутності вихідного сигналу. У такому разі датчик підключається до ПЛК100 за схемою, яка зображена на рис. 3.11.

Рисунок 3.11 – Підключення датчика N-P-N
з додатковим підтягувальним резистором

Враховуючи вищесказані особливості схем відключення датчиків і виконавчих механізмів, можна запропонувати типову схему роботи контролера ПЛК100 у промисловій мережі, показану на рис. 3.12.

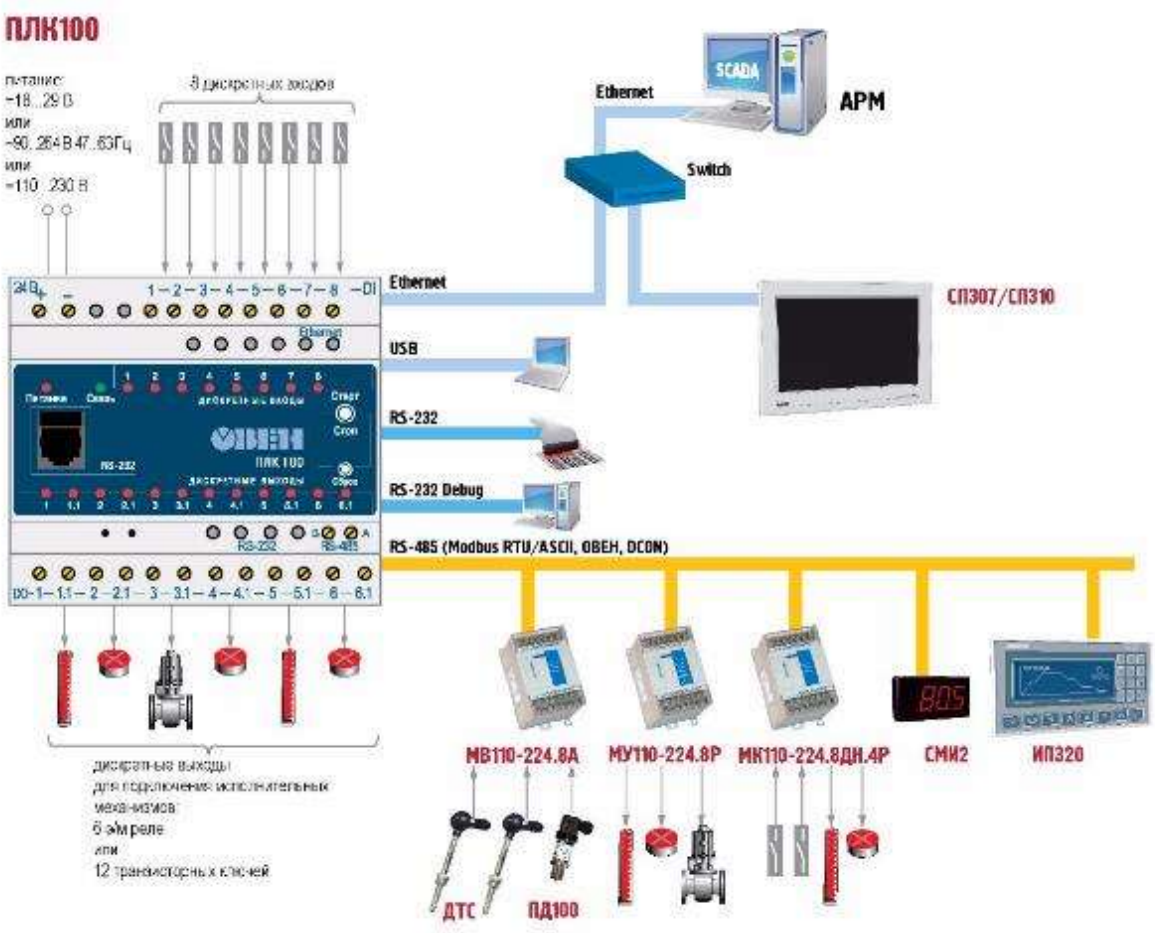


Рисунок 3.12 – Схема роботи контролера ОВЕН ПЛК100
у промисловій мережі

Як видно з цього рисунка, для розширення кількості портів вводу/виводу у схемі використовуються:

– модулі аналогового вводу з універсальними входами (з інтерфейсом RS-485) МВ110;

– модулі дискретного виводу (з інтерфейсом RS-485) МУ110;

– модулі дискретного вводу/виводу (з інтерфейсом RS-485) МК110;

– графічна монохромна панель оператора ИП320;

– сенсорні панелі оператора з серії СП3хх.

Для контролю за станом технологічного процесу використовується автоматизоване робоче місце на базі персонального комп’ютера. АРМ диспетчера підключається до ПЛК через інтерфейс Ethernet.

Через інтерфейс RS-232 підключається сканер штрих-кодів, а також ПК зі встановленим програмним середовищем CODESYS.

3.2 ПЛК150 контролер для малих систем автоматизації з AI/DI/DO/AO

ОВЕН ПЛК150 – моноблочний контролер з дискретними і аналоговими входами/виходами на борту для автоматизації малих систем.

Призначення контролера ОВЕН ПЛК150 :

– вимір і автоматичне регулювання температури (з використанням як первинного перетворювача термометрів опору), а також інших фізичних параметрів, значення яких первинними перетворювачами (датчиками) може бути перетворене в напругу постійного струму, уніфікований електричний сигнал постійного струму або активний опір;

– вимір аналогових сигналів струму або напруги;

– вимір дискретних вхідних сигналів;

– управління дискретними (релейними) виходами:

– управління аналоговими виходами;

– прийом і передача даних за інтерфейсами RS-485, RS-232, Ethernet;

– виконання призначеної для користувача програми за аналізом результатів вимірювання дискретних і аналогових входів;

– управління дискретними входами і виходами, передача і прийом даних за інтерфейсами RS-485, RS -232, Ethernet.

Контролер може застосовуватися для створення систем автоматизованого управління технологічним устаткуванням в енергетиці, на транспорті, в т.ч. залізничному, в різних областях промисловості, житлово-комунального і сільського господарства.

Логіка роботи ПЛК150 визначається користувачем у процесі програмування пристрою за допомогою інтегрованого середовища розробника CODESYS 2.3.8.1 і старше.

На рис. 3.13 показаний зовнішній вигляд ПЛК150.

Особливості ОВЕН ПЛК150:

– компактний DIN корпус;

– дискретні входи/виходи;

– наявність послідовних портів (RS-485, RS-232) і Ethernet;

– розширення кількості точок вводу/виводу здійснюється шляхом підключення зовнішніх модулів вводу/виводу за будь-яким із вбудованих інтерфейсів;

Рисунок 3.13 – Зовнішній вигляд ПЛК150

– у цій версії ПЛК відсутня операційна система (ОС), що підвищує надійність роботи контролерів;

– швидкість роботи дискретних входів – до 10 кГц з використанням підмодулів лічильника. Велика кількість інтерфейсів на борту: Ethernet, два послідовні порти;

– вбудований акумулятор, що дозволяє «перечікувати» зникнення живлення, виконання програми зі зникненням живлення і переведення вихідних елементів в «безпечний стан»;

– вмонтований годинник реального часу;

– контролер підтримує роботу з нестандартними протоколами за будь-яким з портів, що дозволяє підключати такі пристрої як електро-, газо-, водолічильники, зчитувачі штрих-кодів тощо.

На рис. 3.14 показана структурна схема ПЛК150.

Як видно з рисунка, ця серія ПЛК має схожу на ПЛК100 структуру. Так само він працює на основі 32-розрядного контролера, виконаного на ядрі ARM9 з тактовою частотою 200 МГц. ПЛК оснащений 8 Мб ОЗП і 4 Мб ПЗП (Flash) для зберігання програми управління. Пристрій має 4 Кб енергонезалежної пам’яті EEPROM. Мінімальний час машинного циклу може досягати 1 мс.

Відмінності торкнулися портів вводу/виводу. Так, у цій серії ПЛК використовується шість дискретних входів з гальванічною розв’язкою, чотири релейних дискретних виходи із струмом комутації до 2 А з напругою не більше 220 В 50 Гц і cos j > 0,4.

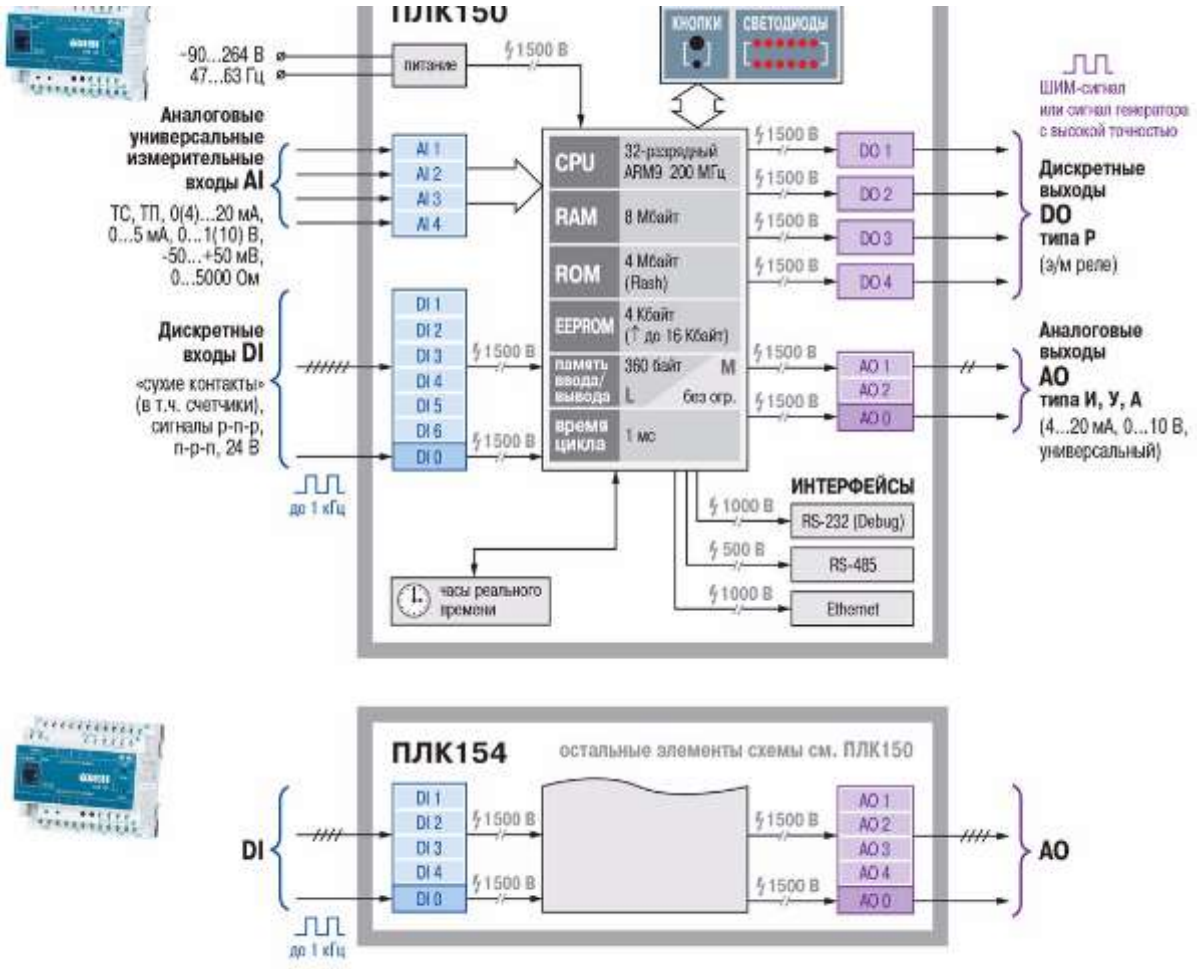


Рисунок 3.14 – Структурна схема ПЛК150

Основною відмінністю цього ПЛК є наявність аналогових входів і виходів.

У ПЛК150 використовується чотири аналогових входи. Розрядність АЦП аналогового входу складає 16 біт. Гальванічна розв’язка на вході відсутня.

У пристрої застосовується два аналогових виходи з розрядністю 10 біт. Залежно від модифікації відрізняється тип вихідного сигналу:

– ПЛК 150-И – струм 4...20 мА;

– ПЛК 150-У – напруга 0...10 В;

– ПЛК 150-А – струм 4...20 мА або напруга 0...10 В.

Для живлення аналогових виходів використовується вбудована, загальна на усі виходи напруга.

Пристрій підтримує такі інтерфейси:

– RS-232 для підключення зовнішніх пристроїв;

– RS-232 для програмування пристрою;

– RS-485;

– Ethernet.

Програмування контролерів ОВЕН ПЛК150 здійснюється професійною системою програмування CODESYS v.2. Стислі технічні характеристики ПЛК150 зведено до таблиці 3.3.

Таблиця 3.3 – Технічні характеристики ПЛК150

Найменування параметра	Характеристика
Ресурси	
Центральний процесор	32-розрядний RISC-процесор 200 Мгц на базі ядра ARM9
Обсяг оперативної пам’яті	8 Мб
Обсяг енергонезалежної пам’яті зберігання ядра CODESYS, програм і архівів	4 Мб
Розмір Retain-пам’яті	4 Кб
Час виконання циклу ПЛК	Мінімальний – 250 мкс (нефіксоване), типовий від 1 мс
Дискретні входи	
Кількість дискретних входів	8
Гальванічна розв’язка дискретних входів	є, групова
Електрична міцність ізоляції дискретних входів	1,5 кВ
Максимальна частота сигналу, що подається на дискретний вхід: – у ході програмної обробки – під час застосування апаратного лічильника – під час застосування обробника енкодера	 1 кГц 10 кГц 1 кГц
Дискретні виходи	
Кількість дискретних виходів	4 е/м реле
Характеристики дискретних виходів	Струм комутації до 2 А з напругою не більше 220 В 50 Гц і cos j > 0,4
Гальванічна розв’язка дискретних виходів	є, індивідуальна
Електрична міцність ізоляції дискретних виходів	1,5 кВ

Продовження таблиці 3.3

Найменування параметра	Характеристика
Аналогові входи	
Кількість аналогових входів	4
Розрядність вбудованого АЦП	16 біт
Внутрішній опір аналогового входу: – у режимі вимірювання струму – у режимі вимірювання напруги 0...10 В	 50 Ом близько 10 кОм
Час опитування одного аналогового входу	0,5 с
Межа основної приведеної погрішності вимірювання аналоговими входами	0,5 %
Гальванічна ізоляція аналогових входів	відсутня
Аналогові виходи	
Кількість аналогових виходів	2
Розрядність ЦАП	10 біт
Тип вихідного сигналу: – ПЛК 150-И – ПЛК 150-У – ПЛК 150-А	 Струм 4...20 мА Напруга 0...10 В Струм 4...20 мА або напруга 0...10 В
Живлення аналогових виходів	вбудоване, загальне на усі виходи
Гальванічна ізоляція аналогових виходів	є, групова
Електрична міцність ізоляції аналогових виходів	1,5 кВ
Інтерфейси зв’язку	
Інтерфейси	Ethernet 100 Base -T RS-232 – два канали RS-485
Швидкість обміну за інтерфейсами RS	від 4800 до 115200 bps
Протоколи, які підтримуються	ОВЕН Modbus RTU, Modbus ASCII DCON Modbus TCP GateWay (протокол CODESYS)
Програмування	
Середовище програмування	CODESYS 2.3.8.1 (і старше)
Інтерфейс для програмування і відладки	RS-232 або Ethernet

Користувач має можливість реалізувати в середовищі програмування CODESYS власний протокол, який не підтримується ОВЕН ПЛК150. У цьому випадку він може скористатися спеціальною бібліотекою, яка відкриває

низькорівневий доступ до послідовних портів ОВЕН ПЛК150. Розташування портів вводу/виводу і елементів управління ПЛК150 показано на рис. 3.15.

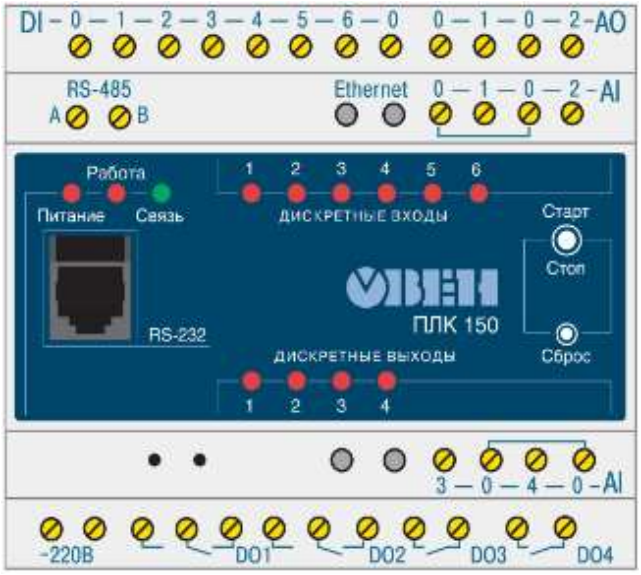


Рисунок 3.15 – Розташування портів вводу/виводу
і елементів управління ПЛК150

На верхній стороні пристрою розташовані клеми для підключення дискретних входів, аналогових входів (два канали) і виходів. Також зверху розташовані контакти для підключення інтерфейсу RS-485 і Ethernet.

На нижній стороні розташовані клеми для підключення напруги живлення (220 В), релейні дискретні виходи і аналогові входи (два канали).

На передню панель контролера виведена світлодіодна індикація про стан дискретних входів і виходів («Дискретні входи», «Дискретні виходи»), про наявність живлення («Живлення») і про наявність зв’язку («Зв’язок») з середовищем програмування CODESYS, а також індикатор «Робота». Також на передній панелі є дві кнопки:

– кнопка «Старт/Стоп», призначена для запуску і зупинки програми в контролері, може бути використана як додатковий дискретний вхід контролера;

– прихована кнопка «Скидання», призначена для перезавантаження контролера. Натиснути кнопку «Скидання» можливо тільки тонким загостреним предметом.

У таблиці 3.4 наведені характеристики вбудованих вихідних елементів.

Таблиця 3.4 – Характеристики вбудованих вихідних елементів

Позначення	Найменування	Характеристики
И	Цифро-аналоговий перетворювач «параметр – струм 4...20 мА»	Опір навантаження від 0 до 900 Ом
У	Цифро-аналоговий перетворювач «параметр – напруга 0...10 В»	Опір навантаження від 2 кОм
А	Цифро-аналоговий перетворювач «параметр – струм 4...20 мА або напруга 0...10 В»	Опір навантаження від 150 до 900 Ом для струмового сигналу і понад 10 кОм для сигналу напруги

На рис. 3.16 показана схема підключення ОВЕН ПЛК150.

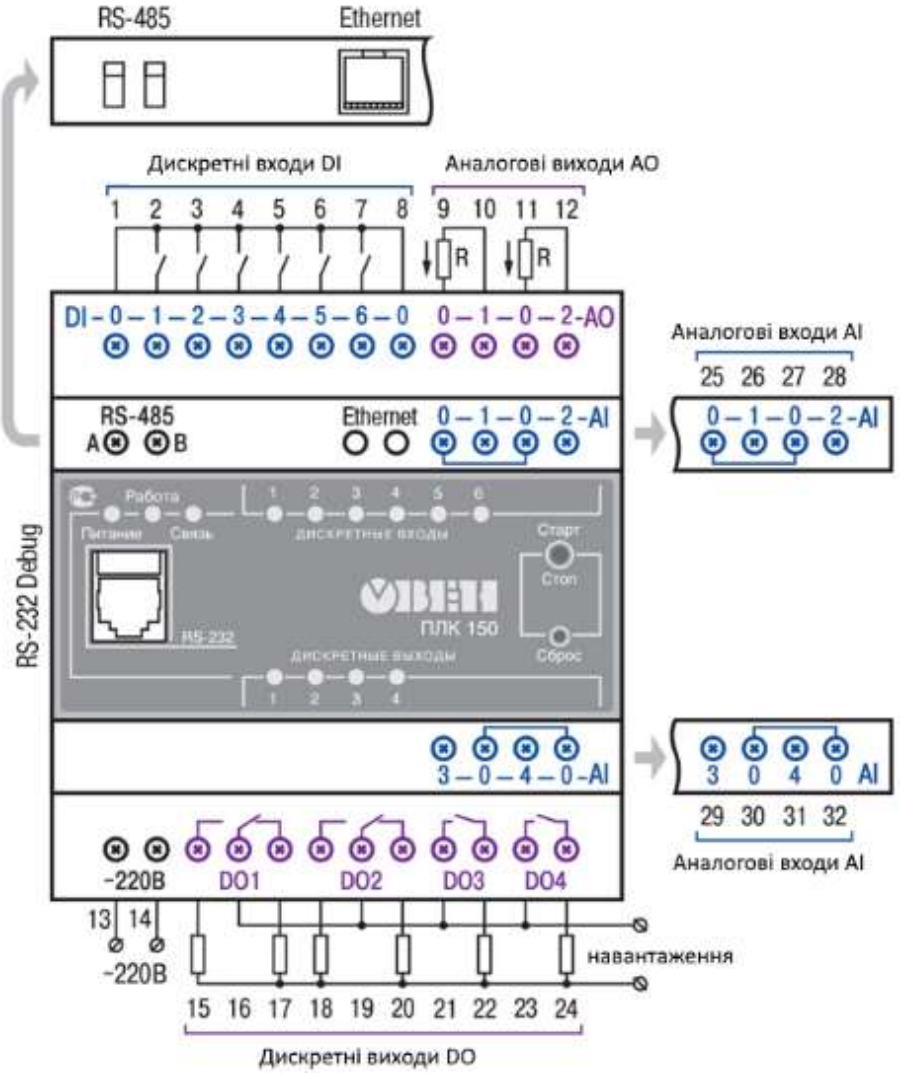


Рисунок 3.16 – Схема підключення ОВЕН ПЛК150

Опір навантаження аналогового виходу ПЛК150:

– R ≤ 900 Ом за вихідного сигналу «струм 4...20 мА»;

– R > 2 кОм за вихідного сигналу «напруга 0...10 В».

Підключення зовнішнього блоку живлення для аналогових виходів не потрібне, оскільки блок живлення вбудований у контролер. Підключення дискретних входів до вхідних клем пристрою показано на рис. 3.17.

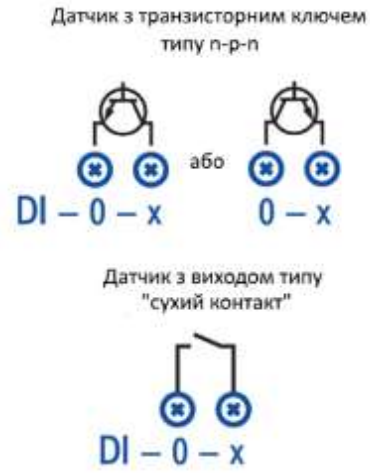


Рисунок 3.17 – Підключення дискретних входів до вхідних клем пристрою

На рис. 3.18 показана схема підключення до ПЛК150 дискретних датчиків з напівпровідниковим вихідним каскадом.

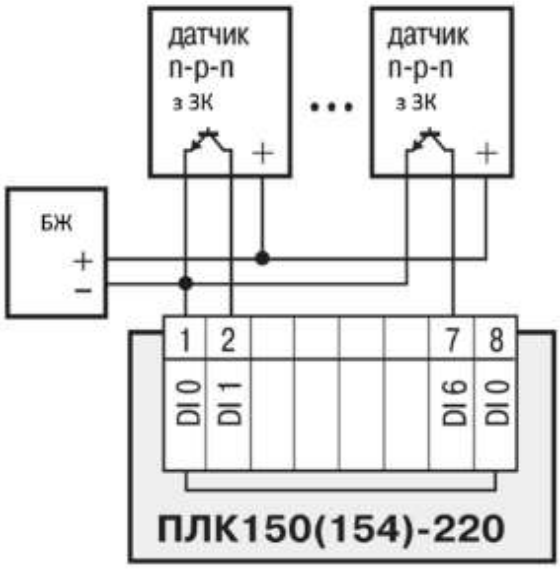


Рисунок 3.18 – Схема підключення до ПЛК150 дискретних датчиків з напівпровідниковим вихідним каскадом

Під час вимірювання напруги або струму датчики підключаються до аналогового входу, як показано на рис. 3.19.

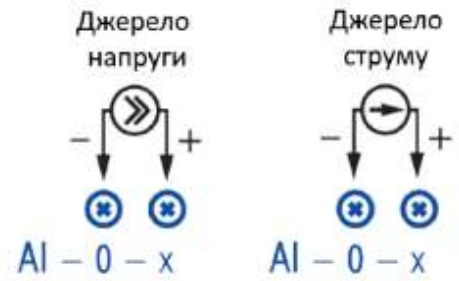


Рисунок 3.19 – Підключення датчиків до аналогового входу
для вимірювання напруги або струму

У разі необхідності вимірювання температури використовуються термопари або термоопір. У такому разі ці датчики також підключаються до аналогового входу ПЛК. На рис. 3.20 показаний приклад підключення термопари і термоопору до ПЛК150 фірми ОВЕН.

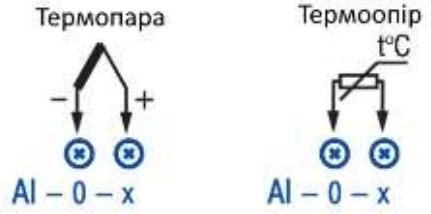


Рисунок 3.20 – Підключення термопари і термоопору до ПЛК150

Приклад схеми роботи контролера ПЛК150 у промисловій мережі показаний на рис. 3.21.

Як видно з цього рисунку, спільно з ПЛК150 у мережі використовуються декілька модулів розширення портів вводу/виводу:

– модулі аналогового вводу з універсальними входами (з інтерфейсом RS-485) МВ110;

– модулі дискретного виводу (з інтерфейсом RS-485) МУ110;

– модулі дискретного вводу/виводу (з інтерфейсом RS-485) МК110;

– графічна монохромна панель оператора ИП320;

– сенсорні панелі оператора з серії СП3хх.

Шість дискретних входів ПЛК використовуються для підключення датчиків з виходом типу «сухий контакт».

Виконавчі пристрої підключаються як до дискретних виходів релейного типу, так і до аналогових виходів (4…20 мА, 0...10 В).

Вимірювальні датчики рівня з аналоговими виходами підключаються до відповідних входів ПЛК.

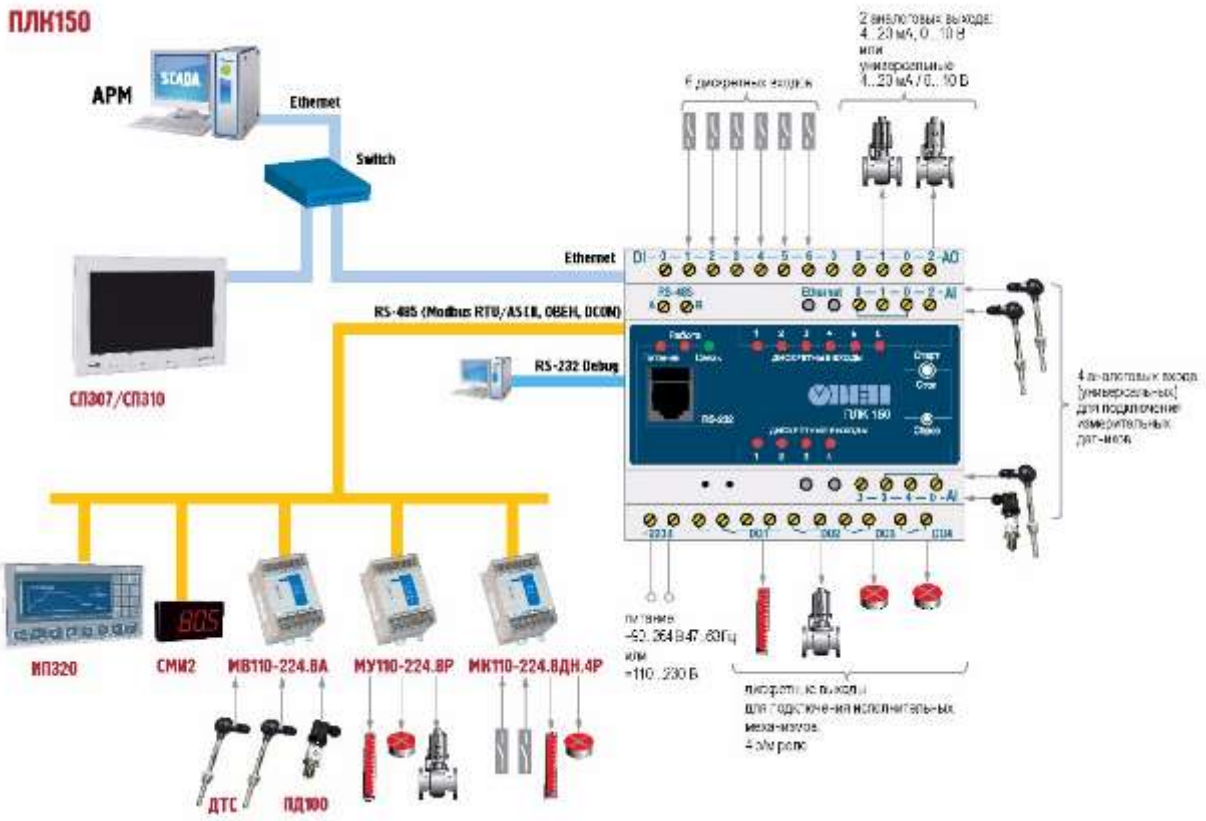


Рисунок 3.21 – Можлива схема роботи контролера ОВЕН ПЛК150 у промисловій мережі

Для контролю за станом технологічного процесу використовується автоматизоване робоче місце на базі персонального комп'ютера. АРМ диспетчера підключається до ПЛК через інтерфейс Ethernet.

Через інтерфейс RS-232 підключається сканер штрих-кодів, а також ПК зі встановленим програмним середовищем CODESYS.

3.3 Контрольні запитання та завдання

1. Наведіть характеристики контролера для малих систем автоматизації ПЛК100.

2. Як підключаються датчики з виходом типу N-P-N до ПЛК100?

3. Наведіть схему роботи контролера ОВЕН ПЛК100 у промисловій мережі.

4. Наведіть структурну схему ПЛК150.

5. Наведіть схему роботи контролера ОВЕН ПЛК150 у промисловій мережі.

4 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ РОБОТИ З ПЛК

4.1 Загальні принципи програмування ПЛК

Програмовані логічні контролери, програмування яких здійснюється з вбудованого або виносного пульта, зустрічаються сьогодні вкрай рідко. Як правило, це прості, спеціалізовані ПЛК. Усе програмування таких контролерів зводиться зазвичай до задання набору констант. На рис. 4.1 показаний приклад використання терміналу для програмування ПЛК.

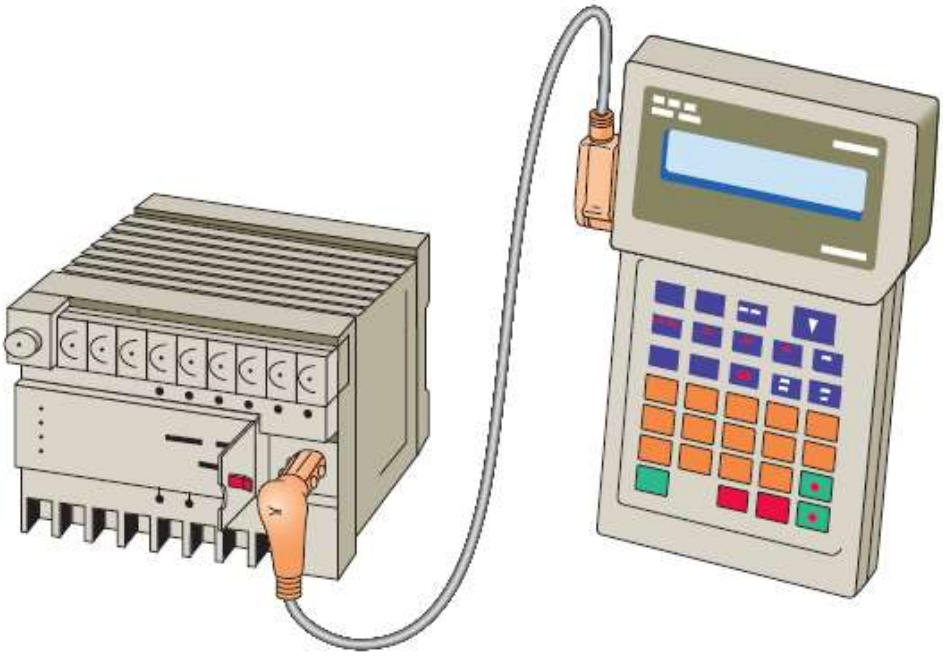


Рисунок 4.1 – Використання терміналу для програмування ПЛК

На рис. 4.1 показаний програматор ручного типу. Цей запатентований пристрій для програмування має сполучний кабель, так що його можна підключити до порту для програмування ПЛК. Деякі контролери можуть використати вбудовану панель для набору команд, а не окремий ручний пристрій.

Ручні програматори компактні, дешеві і прості у використанні. Ці пристрої містять багатофункціональні клавіші і вікно з рідкокристалічним дисплеєм (LCD) або світлодіодним екраном (LED). Зазвичай це клавіатура для введення і редагування команд, а також клавіші навігації для переміщення за програмою. Ручні програматори мають обмежені можливості відображення. Деякі пристрої відображатимуть тільки останню запрограмовану інструкцію,

тоді як інші відображатимуть від двох до чотирьох східців релейної логіки. Так звані інтелектуальні портативні програматори призначені для підтримки певного сімейства ПЛК від конкретного виробника.

Для програмування ПЛК універсального призначення застосовують персональний комп'ютер. Процес розробки і відлагодження програмного забезпечення відбувається за допомогою спеціалізованих комплексів програм, що забезпечують комфортне середовище для роботи прикладного програміста (рис. 4.2).

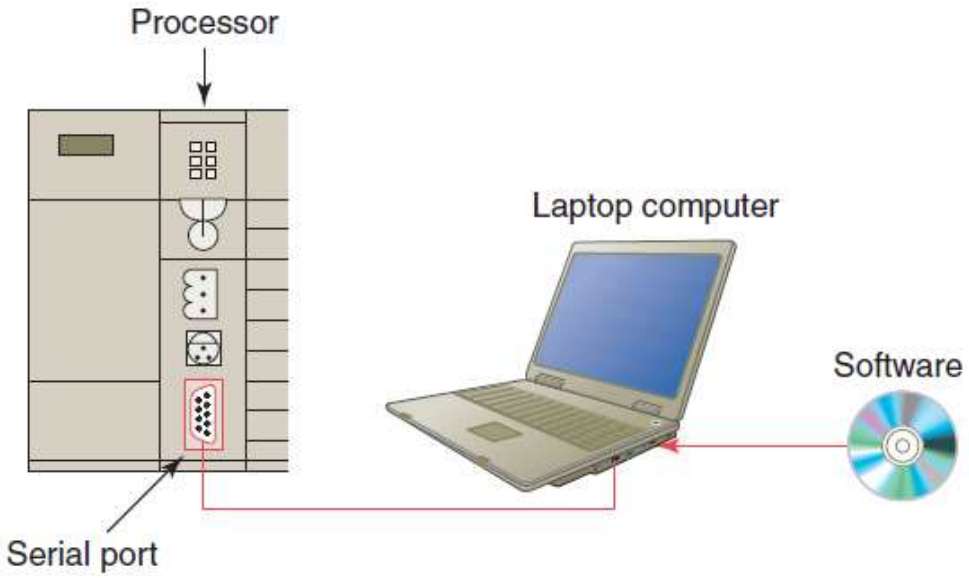


Рисунок 4.2 – Для програмування ПЛК використовується персональний комп'ютер

Типові можливості програмного забезпечення включають онлайн і офлайн редагування програм, онлайн моніторинг програм, документування роботи програм, діагностику несправностей, що виникають у ПЛК, і усунення помилок програмування контрольованої системи.

Традиційно усі провідні виробники ПЛК мають власні фірмові напрацювання в області інструментального програмного забезпечення. Більшість програмних пакетів не дозволяють розробляти програми на ПЛК іншого виробника. В деяких випадках у одного виробника може бути декілька сімейств ПЛК, для кожного з яких потрібно власне програмне забезпечення для програмування.

Відкритість стандарту МЕК 61131, з одного боку, і складність реалізації висококласних комплексів програмування з іншого, привели до появи універсальних інструментів програмування ПЛК.

Одним з найпопулярніших універсальних інструментів програмування є інструментальний програмний комплекс промислової автоматизації CODESYS, який підтримує стандарт МЕК і враховує фірмові особливості більше 150 моделей ПЛК.

Комплекс CODESYS (Controllers Development System) фірми 3S (Smart Software Solutions) надає проектувальникові зручне середовище для програмування контролерів мовами МЕК [5, 6]. Використовувані редактори і налагоджувальні засоби базуються на широко відомих принципах розробки технологічних програм.

CODESYS дозволяє використовувати мови: IL, ST, LD, SFC, FBD і CFC. Текстові редактори CODESYS проводять автоматичне оголошення змінних, тип яких задається в діалоговому вікні.

Графічний редактор автоматично виконує розставляння компонентів схеми (контактів, котушок реле, таймерів тощо) і трасування їх з'єднань; нумерацію ланцюгів; масштабування зображення, що дозволяє побачити усю LD-діаграму (програму) або певну її частину і виділяти кольором активні ланцюги.

Вбудовані емулятор і елементи візуалізації дають можливість виконувати відладку проекту без самих апаратних засобів.

На рис. 4.3 показана структура проекту в CODESYS.

Особливості проекту в редакторі CODESYS:

– проект зберігається в одному файлі (name.pro);

– проект містить програмні компоненти (POU), візуалізацію, ресурси тощо;

– виконання додатка починається з POU PLC_PRG (аналог функції main);

– програма управління виконується циклічно.

Усі програмні компоненти мають викликатися прямо або побічно з головної програми PLC_PRG (рис. 4.4).

Створення проекту розпочинається з вибору цільової платформи, як показано на рис. 4.5

Для того, щоб у списку, що випадає, з'явився потрібний тип ПЛК, необхідно заздалегідь встановити target-файли від виробника цього пристрою. У target-файлах міститься інформація про ресурси програмованих контролерів, з якими працює CODESYS. Target-файл поставляється виробником контролера.

Однією з умов завантаження програми в ПЛК є виконання його конфігурації. Для цього настроюється час циклу виконання програми,

конфігуруються вхідні й вихідні порти, ставляться у відповідність вхідні й вихідні змінні.

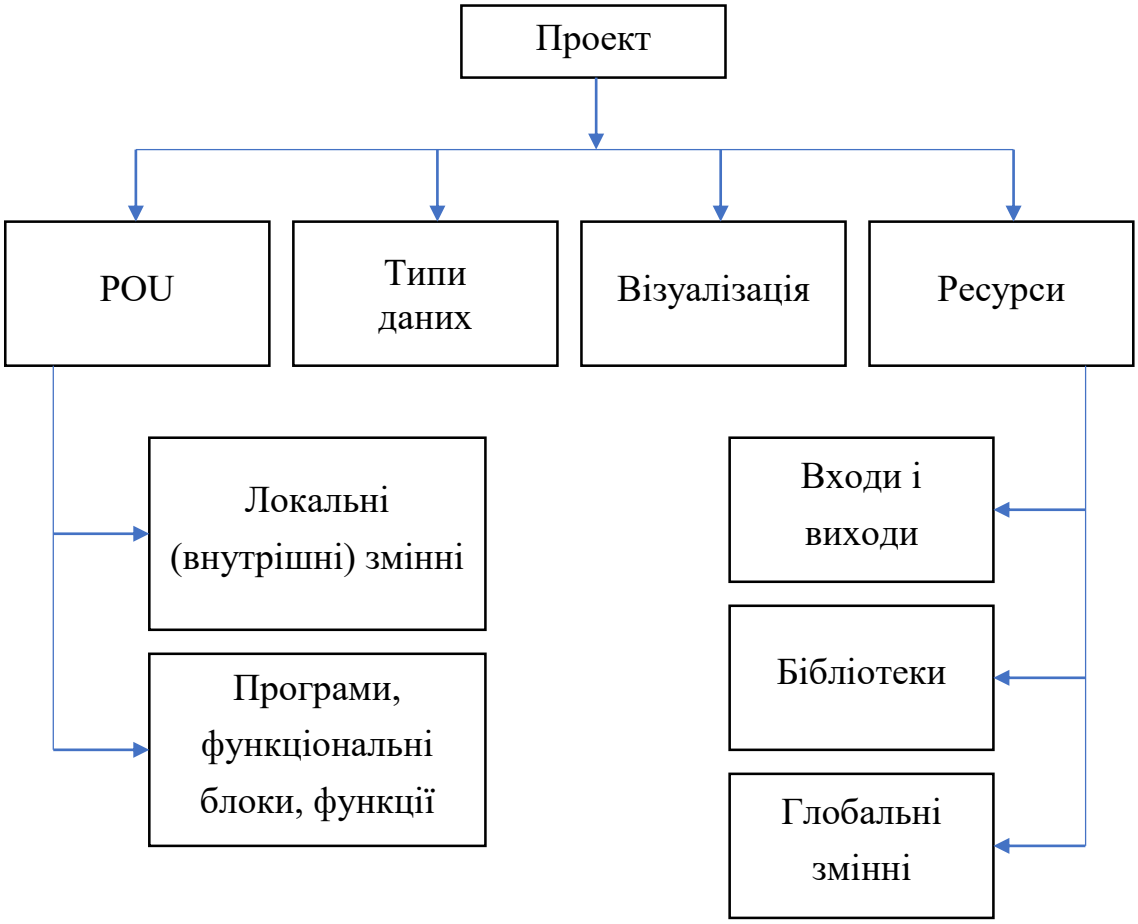


Рисунок 4.3 – Структура проекту в CODESYS

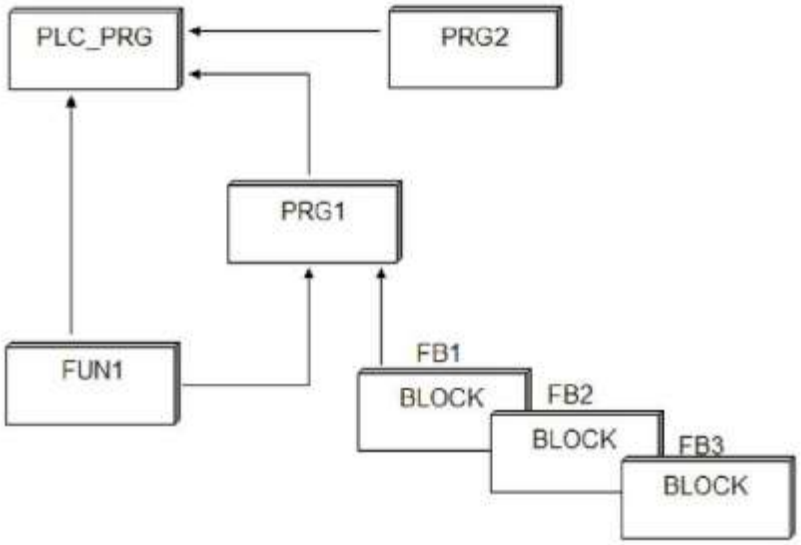


Рисунок 4.4 – Виклик програмних компонентів

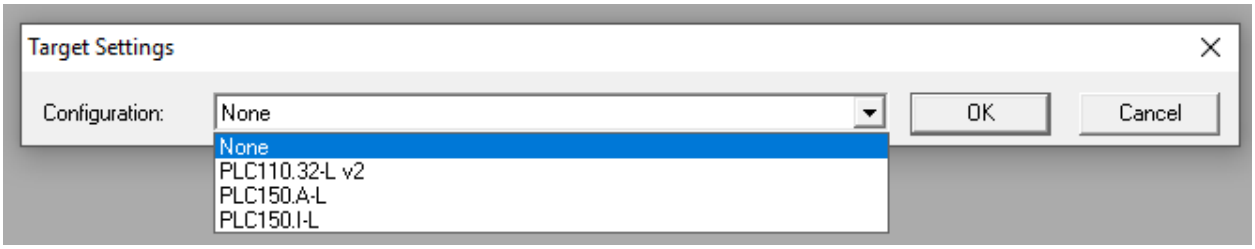


Рисунок 4.5 – Вибір цільової платформи

Після створення керуючої програми і виконання процедури конфігурації, створений проект можна відкомпілювати. Якщо усе зроблено вірно, то має з'явитися повідомлення «0 помилок» або «Проект актуальний». Інакше необхідно виправити допущені помилки. У цьому допоможуть розгорнуті повідомлення про помилки.

Наступним етапом є підключення контролера до комп'ютера. Контролери ОВЕН зазвичай мають такі комунікаційні інтерфейси: Ethernet 10/100 Мбіт/с, RS232 (два канали), RS485 і USB 2.0 Device. Зв'язок з контролером може бути організований по-різному: за інтерфейсами Ethernet, Debug RS-232, USB Device.

Один з варіантів організації зв'язку ПЛК з комп'ютером забезпечується через порт Debug RS-232. Цей порт розташований на корпусі контролера і призначений для зв'язку ПЛК з середовищем програмування, завантаження призначеної для користувача програми і її відладки. Підключення до цього порту здійснюється кабелем, що входить у комплект постачання ОВЕН ПЛК.

Для завантаження програми використовується протокол GateWay (протокол системи CODESYS).

Після організації зв'язку виконується налаштування каналу з'єднання з контролером у вікні «Параметри зв'язку» (Communication parameters), меню Online, що викликається командою.

Далі в меню «Онлайн» вибирається рядок «Підключення». При цьому режим емуляції має бути заздалегідь відключений. Встановлюється зв'язок між ПЛК і комп'ютером (системою CODESYS). Якщо компіляція проекту командою «Компілювати» ще не виконувалася, то вона буде виконана автоматично зі з'єднанням контролера з комп'ютером за командою «Підключення». При цьому виводяться виявлені помилки і попередження.

Команда «Завантаження» в меню «Онлайн» завантажує код проекту (програму) в контролер. Запускає програму в ПЛК команда «Старт» з меню «Онлайн», а зупиняє команда «Стоп».

4.2 Конфігурація роботи ПЛК

Утиліта PLC Configuration (Конфігуратор ПЛК) доступна користувачеві на вкладці ресурсів (Resources) Організатора об’єктів середовища розробки CODESYS. Конфігуратор є редактором ресурсів ПЛК, що належать області вводу/виводу, через яку програма ПЛК здійснює інформаційний обмін із зовнішнім середовищем.

У конфігурації є присутніми модулі, що відповідають за структуризацію областей вводу і/або виводу, кожен з яких може містити вкладені піделементи (субмодулі і канали). Для каналів можуть бути призначені символічні імена. Прямі МЕК адреси відображаються в конфігурації для кожного символічного імені.

Первинний вид екранної форми редактора конфігурації у вікні CODESYS задають файл (файли) конфігурації *.cfg. Вони розташовуються в директорії, визначеній у цільовому файлі (Target file) і зчитуються з відкриттям проекту в CODESYS. На рис. 4.6 показаний приклад вікна конфігурації ПЛК 150.

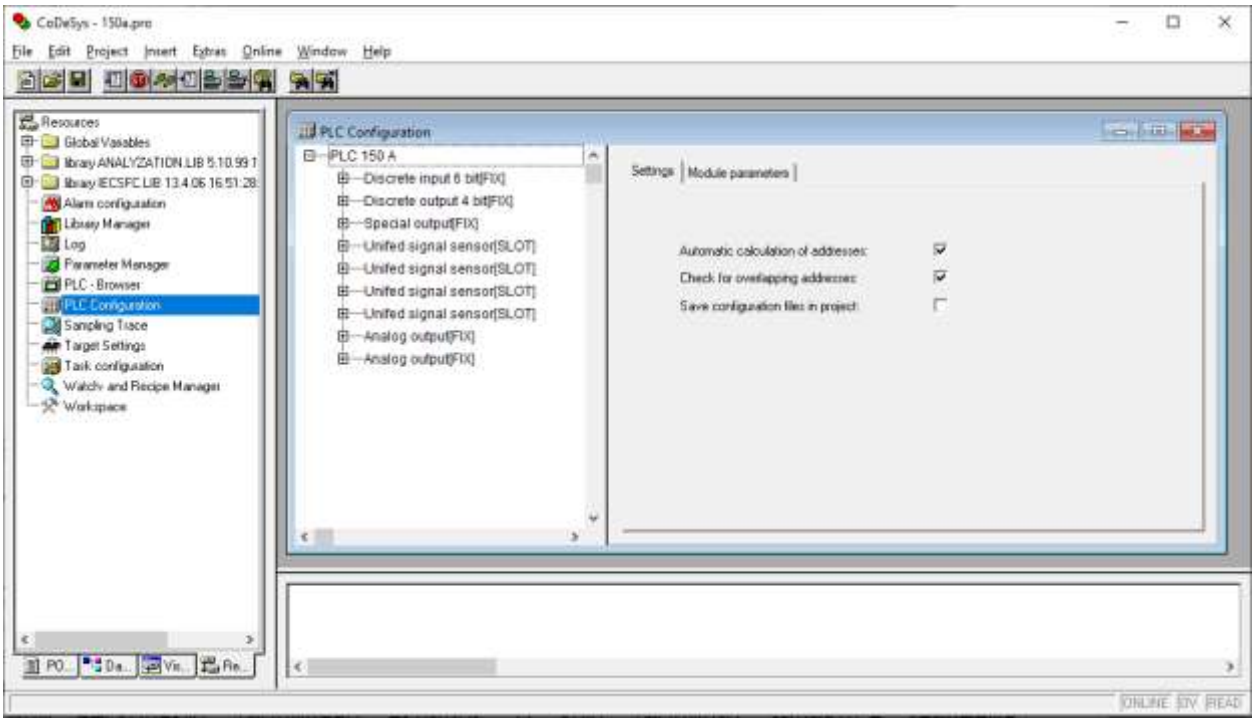


Рисунок 4.6 – Приклад вікна конфігурації ПЛК 150

Вікно редактора конфігуратора ПЛК розділене на дві частини. У лівій частині вікна відображається ієрархічна структура – дерево Конфігурації ПЛК. Структура і компоненти дерева можуть бути змінені користувачем CODESYS.

У правому вікні показані доступні у момент роботи користувача діалоги конфігурації у вигляді однієї або декількох табличних вкладок. У цих таблицях задаються значення параметрів елементів Конфігурації ПЛК. Праву частину вікна видно за замовчуванням, але воно може бути приховано через опції меню «Extras» => «Properties».

Задання часу циклу ПЛК

У ході налаштуванні конфігурації користувач може змінити параметри функціонування ПЛК, встановлені за замовчуванням. Для цього використовується вікно властивостей ПЛК, яке відображається з подвійним натисканням кнопкою мишки на опцію з назвою типу ПЛК. На рис. 4.7 показано вікно завдання часу циклу ПЛК.

Мінімальне значення циклу роботи ПЛК (MinCycleLength, вимірюється в мілісекундах) – параметр, який визначає мінімальний період, з яким ПЛК виконує повний цикл своєї роботи. Діапазон значень може змінюватися від 0 до 50 мс, а значення за замовчуванням встановлено 1 мс.

Під час установлення значення періоду циклу ПЛК, рівного нулю, відключається контроль періоду циклу, і цикл ПЛК виконується з максимально можливою частотою.

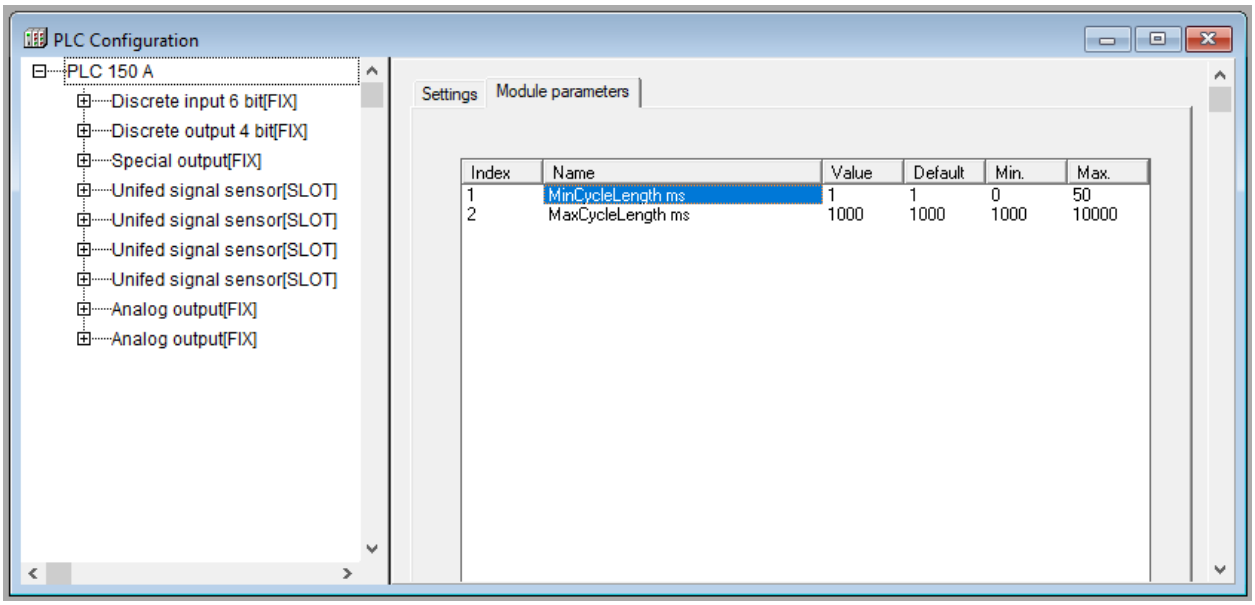


Рисунок 4.7 – Вікно задання часу циклу ПЛК

Максимальне значення циклу роботи ПЛК (MaxCycleLength, вимірюється в мілісекундах) – параметр визначає максимально допустимий час, за який ПЛК виконує повний цикл своєї роботи. У разі перевищення цієї величини під час роботи ПЛК буде примусово перезавантажений. Діапазон значень від 1000 до 10000 мс, а значення за замовчуванням встановлено 1000 мс.

Типи модулів у вікні Конфігурації

Існує два види модулів:

– фіксовані модулі – жорстко задаються і не можуть бути видалені або замінені. Допускається тільки редагування їх параметрів;

– модулі, які додаються – вставляються за бажанням користувача під час конфігурації.

Модулі, які додаються, поділяються на два типи:

а) тип SLOT – означає, що зарезервовано місце для модуля, яке може бути зайняте або залишене порожнім. На одне зарезервоване місце може бути встановлений один модуль;

б) вільний тип (VAR) – означає можливість встановити будь-яку кількість модулів (з урахуванням фізичних можливостей області вводу/виводу).

Можливість додавати модулі може бути заборонена або обмежена, тобто додати можна лише певні типи модулів.

Додавання модулів здійснюється вибором опції «Додавання піделемента» (Append Subelement) в контекстному меню, що викликається натисненням правої кнопки маніпулятора «мишка» із встановленням курсора в дереві Конфігурації ПЛК, і подальшим вибором конкретного модуля зі списку, що випадає. Після появи екранної форми модуля його параметри відповідним чином налаштовуються, як це необхідно для функціонування цього піделемента.

Додавання й видалення модулів, а також налаштування їх параметрів здійснюються за відключеного від середовища програмування контролера. Для відключення контролера необхідно викликати команду меню Online => Logout або скористатися відповідною кнопкою на панелі інструментів.

Вікно екранної форми Конфігуратора ПЛК розділене на дві частини. У лівій частині вікна відображається ієрархічна структура Конфігурації ПЛК. У правій частині – одна з двох вкладок параметрів конкретного модуля, виділеного курсором в ієрархічній структурі. Це вкладка базових параметрів і вкладка параметрів модуля (рис. 4.8).

Вкладка базових параметрів (Base parameters) ідентична для усіх модулів і містить такі значення:

– Modul id – ідентифікаційний номер модуля;

– Node id – положення модуля на його рівні ієрархії в загальній конфігурації. Це значення можна редагувати, у такому разі аналогічні ідентифікатори інших модулів одного рівня ієрархії зміщуватимуться;

– Input, Output, Diagnostic Address – адреси областей вводу/виводу, наводяться конкретні номери. Вони можуть знадобитися в ході програмування, якщо буде необхідно звертатися до них. Значення недоступні для редагування.

На рис. 4.8 на прикладі модуля дискретних входів подана екранна форма Конфігуратора ПЛК, що з'являється з відкриттям вікна конфігурації модуля, із вкладкою базових параметрів у правій частині екранної форми.

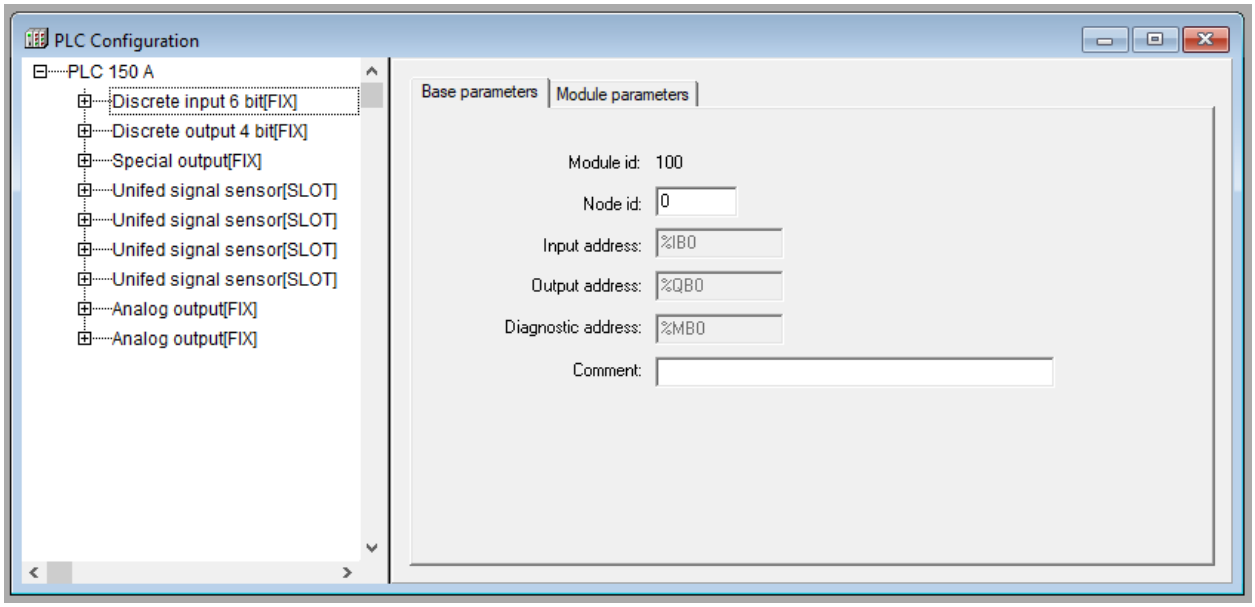


Рисунок 4.8 – Вікно конфігурації модуля дискретних входів і вкладка «Базові параметри»

Друга вкладка правої частини екранної форми – «Параметри модуля» (Module parameters), показана на рис. 4.9, містить параметри модуля, подані у вигляді таблиці, що містить стовпчики:

– Index – номер параметра;

– Name – ім'я;

– Value – поточне значення;

– Default – значення за замовчуванням;

– Min, Max – мінімальна й максимальна величини діапазону можливих значень. Значення параметрів можуть бути цифровими, символьними і такими, що обираються зі списку.

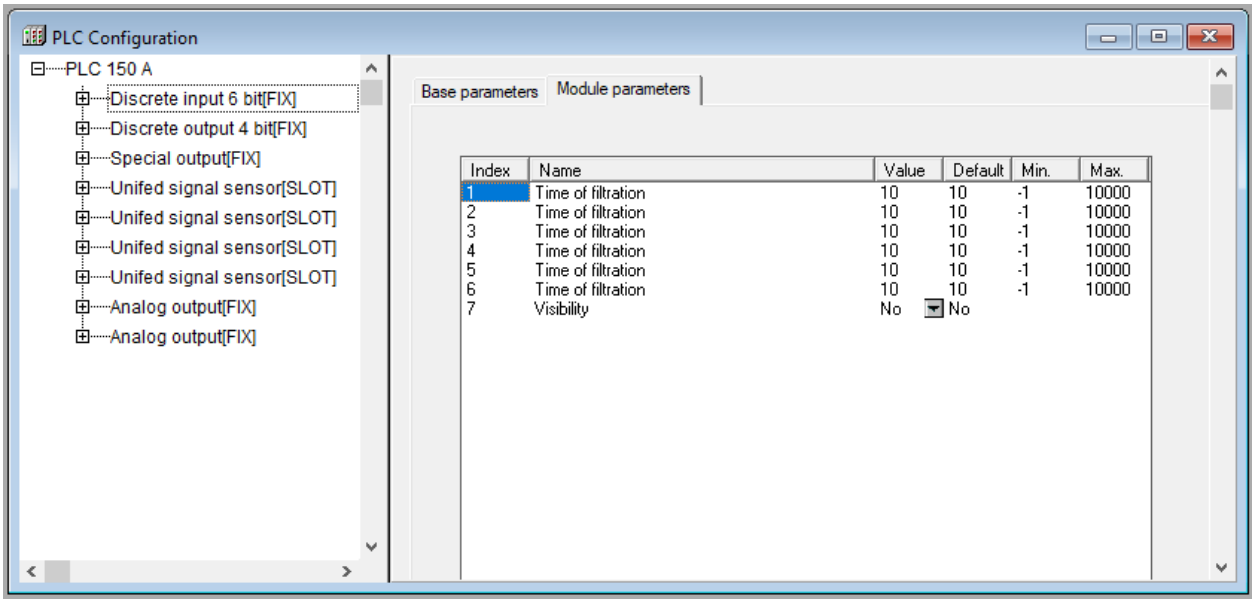


Рисунок 4.9 – Вікно конфігурації модуля дискретних входів і вкладка «Параметри модуля»

У складі модуля є канали – бітові і байтові. Канал – це переносник одиниці інформації (даних) від зовнішнього устаткування в область пам’яті вводу/виводу. Кожному каналу відповідає змінна в області вводу/виводу.

Канал і змінна, що відповідає йому, можуть бути пойменовані, і за присвоєним ім’ям до змінної можна звертатися у програмі. Або в програмі можливий виклик змінної каналу за тією адресою, яка у неї встановлена апаратно, наприклад, %1X 0.0.1.

Восьмибітовий канал може бути використаний у програмі як восьмибітове число, або як чотири або вісім окремих один від одного бітів.

Іменування каналу проводиться так: подвійним клацанням маніпулятора «мишка» із курсором, встановленим на початку рядка назви каналу, здійснюється перехід у режим редагування і вводиться ім’я змінної каналу.

Ім’я може складатися з латинських літер, цифр і знаку «_» (підкреслення). Ім’я має розпочинатися з літери або знаку «_» і має бути унікальним. У деяких випадках редагування імен каналів може бути заборонене. Екранна форма, представлена на рис. 4.10, ілюструє процес іменування каналу – появу поля введення символів.

Дані, наявні у вкладці «Базові параметри», носять інформаційний характер і не редагуються. Для каналу програма виводить таку інформацію:

– коментар – характеристика каналу (наприклад, для модуля дискретних входів – «8 discrete inputs» = «8 дискретних входів»);

– ідентифікаційний номер каналу в загальній ієрархії;

– клас;

– розмір (у бітах).

Для бітового каналу програма виводить тільки коментар з номером бітового каналу, наприклад, «Bit 3».

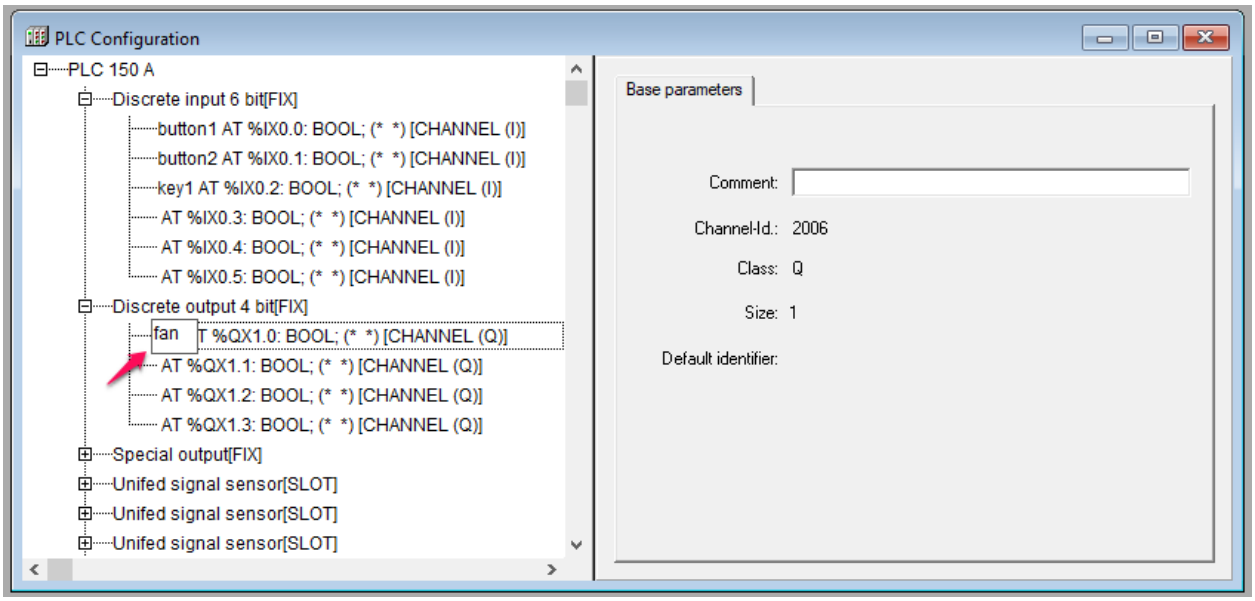


Рисунок 4.10 – Введення і редагування імені змінної каналу

У Конфігураторі реалізована можливість додавання до модулів підлеглих ним підмодулів, які розширюють функціонал або змінюють алгоритм роботи. Підключення до модуля підлеглого йому підмодуля продемонстровано на рис. 4.11 на прикладі модуля дискретних входів. Додавання підмодулів здійснюється з використанням опції Append Subelement з установленням курсору на імені модуля в дереві Конфігурації ПЛК, в який додається підмодуль.

4.2.1 Конфігурація модулів дискретного входу

Модуль дискретних входів (Discrete input) відображає в області вводу/виводу значення, що характеризують стани дискретних входів ПЛК. Модуль має 6-бітовий канал.

Параметри модуля:

– «Час фільтрації» (Time of filtration) – діапазон значень від 0 до 10000, значення за замовчуванням – 10 (1 од. = 100 мкс, 10 од. = 1 мс). На вкладці модуля дискретних входів є шість однойменних параметрів «Час фільтрації» – для кожного бітового каналу (входу), відповідно;

– «Видимість» (Visibility) – задає видимість параметрів модуля у програмі EasyWorkPLC. Значення обираються зі списку «yes» і «no», значення за замовчуванням – «no».

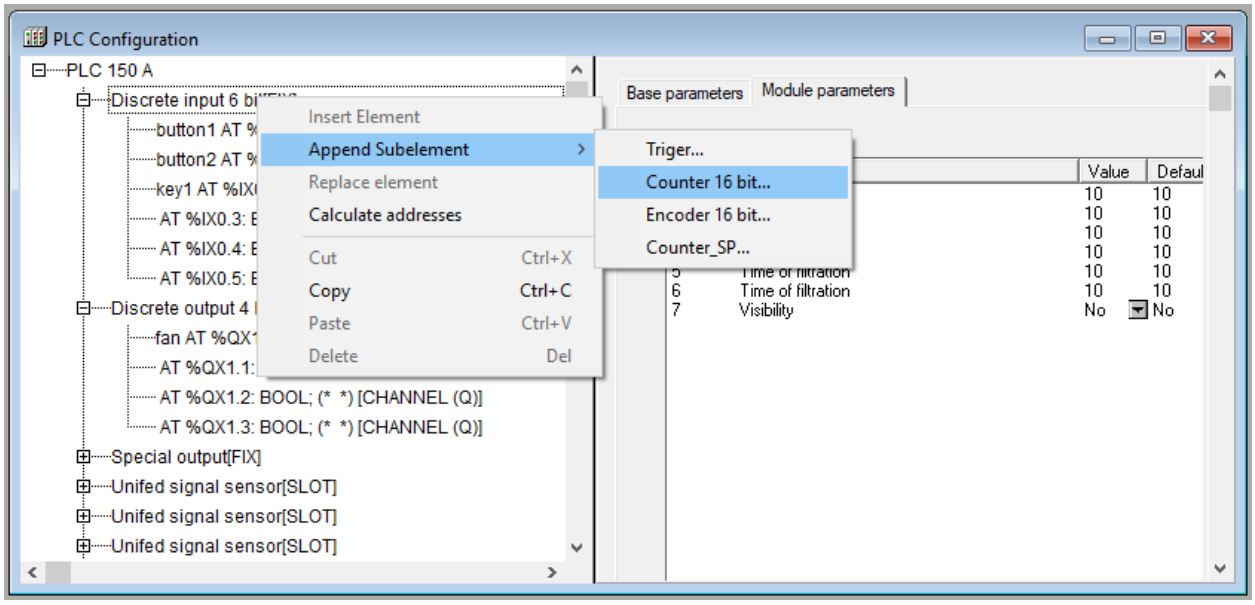


Рисунок 4.11 – Підключення до модуля підлеглого йому підмодуля

Розглянемо детальніше параметри модуля дискретного входу.

Параметр «Час фільтрації» (Time of filtration). Фільтрація застосовується переважно для пригнічення «брязкоту» контактів. Час фільтрації – це період опитування стану одного дискретного входу, задається в сотнях мікросекунд (1 од. = 100 мкс, 10 од. = 1 мс).

Принцип дії фільтрації:

– у зсувному регістрі в драйвері кожного дискретного входу накопичуються значення восьми останніх станів, отриманих внаслідок опитування з періодом, заданим у параметрі «Час фільтрації»;

– якщо стан бітового каналу дискретного входу дорівнює 1 (TRUE), а кількість одиниць у зсувному регістрі менше двох, то бітовий канал перемикається на 0 (FALSE);

– якщо стан бітового каналу дорівнює 0 (FALSE), а кількість одиниць у зсувному регістрі більше п’яти, то бітовий канал перемикається на 1 (TRUE);

– якщо кількість одиниць у зсувному регістрі від 2 до 5, то стан бітового каналу дискретного входу не змінюється.

Режим фільтрації може бути відключений встановленням у параметрі значення, рівного «–1». Відключення фільтрації потрібне під час роботи з підпорядкованими модулями енкодерів для того, щоб не пропускати високочастотні сигнали, а також у тих випадках, коли ПЛК функціонує без обмеження циклу за частотою, тобто на максимально можливій частоті.

Параметр «Видимість» (Visibility). Робота кінцевого користувача з Конфігуратором може здійснюватися за допомогою спеціалізованої програми EasyWorkPLC розробки ПО «ОВЕН». Під час установлення для конкретного модуля значення «yes» параметра «Видимість» параметри цього модуля стають видні у програмі EasyWorkPLC.

Список підмодулів для модуля дискретних входів :

– тригер (Trigger);

– лічильник (Counter);

– енкодер (Encoder);

– лічильник спеціалізований (Counter_SP).

Підмодуль «Тригер» (Trigger)

Тригер – програмний модуль, що дозволяє стежити за станом входу і подає відповідний сигнал про зміну стану входу.

Модуль «Тригер» (Trigger) є підпорядкованим підмодулем модуля дискретних входів і виконує функцію тригера. Модуль має бітовий канал.

На рис. 4.12 показаний приклад вікна конфігурації для модуля «Тригер».

Параметри модуля:

– «Номер входу» (Number of Input) – діапазон значень від 0 до 7, значення за замовчуванням – 0;

– «Фронт сигналу» (SenseEdge) – значення обираються зі списку «RISEEDGE», «FALL EDGE» і «BOTH EDGE», значення за замовчуванням – «RISE EDGE»;

– «Видимість» (Visibility) – задає видимість параметрів модуля в програмі EasyWorkPLC. Значення обираються зі списку «yes» і «no», значення за замовчуванням – «no».

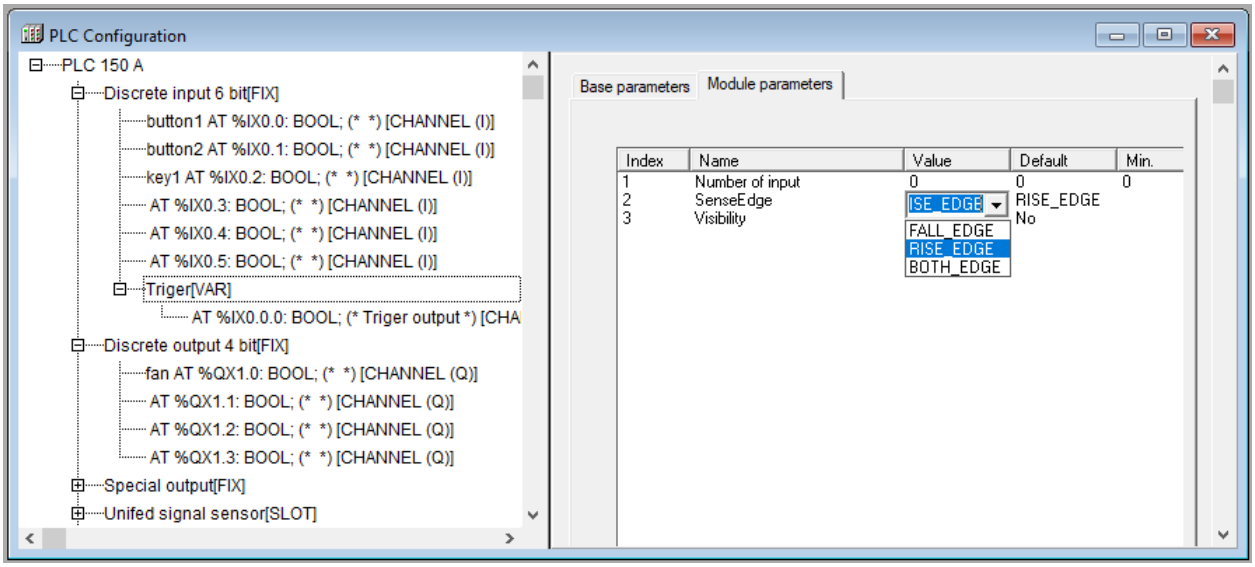


Рисунок 4.12 – Вікно конфігурації для модуля «Тригер»

Параметр «Номер входу» (Number of Input) вказує, який дискретний вхід контролера (вважаючи від 0) оброблятиметься.

Параметр «Фронт сигналу» (SenseEdge) вказує фронт, за яким здійснюватиметься робота. Якщо сигнал представити як прямокутний імпульс, то, очевидно, що у нього зростають і убувають фронти. Відповідно, може бути задане спрацьовування тригера в мить, коли значення сигналу змінюється з 0 на 1, – зростаючий фронт імпульсу RISE_EDGE, коли змінюється з 1 на 0, – спадаючий фронт FALL_EDGE, або за будь-яким фронтом, – BOTH_EDGE.

Графіки роботи тригера для різних фронтів і сигналів наведено на рис. 4.13.

Вибір з трьох варіантів спрацьовування можливий тільки у тому випадку, якщо заданий режим фільтрації входу, тобто значення часу фільтрації більше 0. Якщо час фільтрації не задано (встановлено в 0), особливості апаратної реалізації ОВЕН ПЛК забороняють роботу в режимі BOTH_ EDGE. Дана примітка дійсна для роботи не лише тригера, але й лічильника та енкодера.

Після читання програмою ПЛК значення тригера воно обнуляється, тобто за цикл ПЛК значення не лише прочитуються, але й обнуляються, відповідно, накопичення у тригері немає, і, за необхідності підсумовування, виконання цієї функції здійснюється в програмі.

Підмодуль «Лічильник» (Counter 16bit)

Лічильник – програмний модуль, що здійснює ведення обліку вхідних імпульсів та експорт облікових даних програми ПЛК, при цьому реалізована можливість задання методу підрахунку імпульсів.

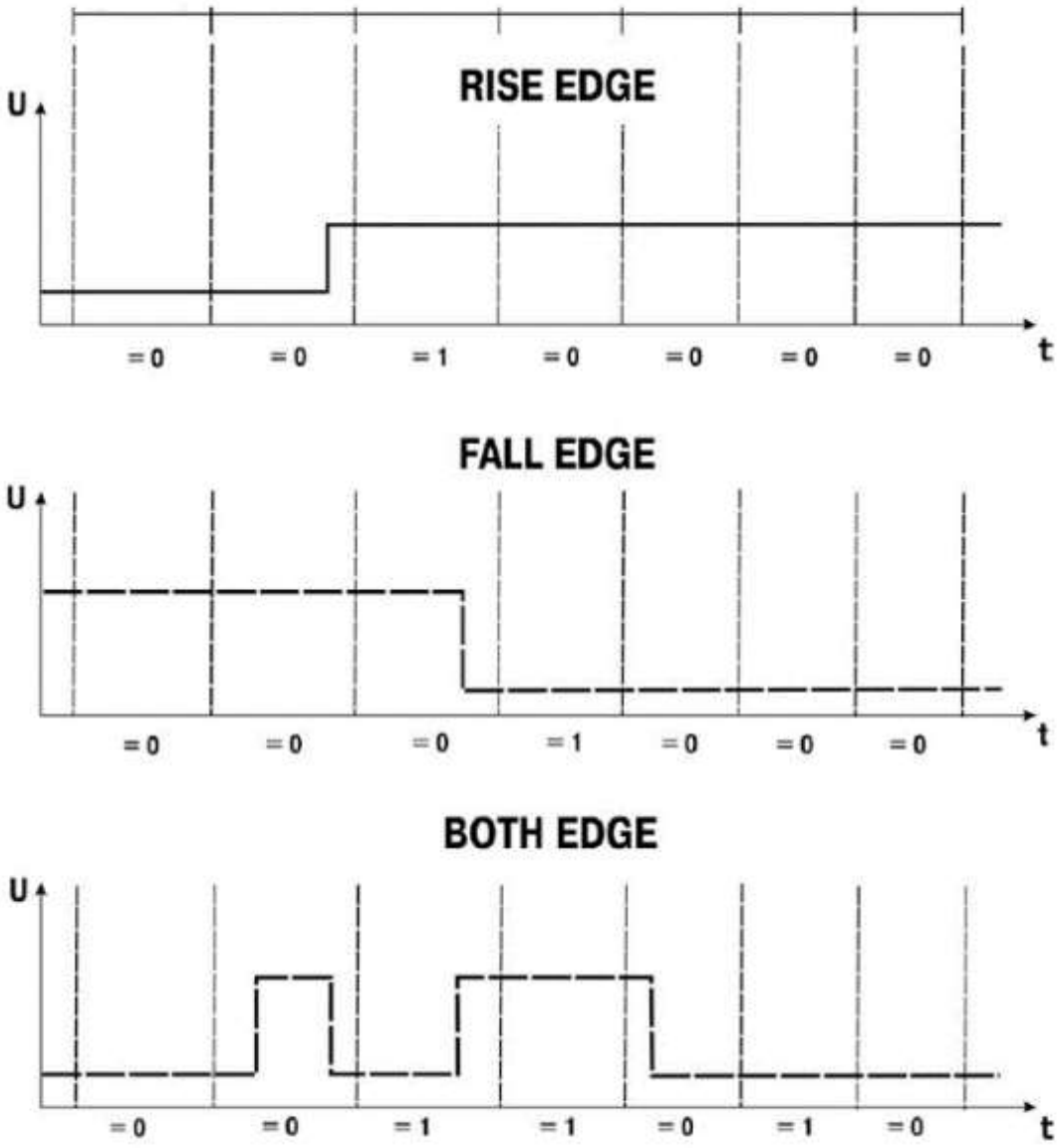


Рисунок 4.13 – Графіки роботи тригера для різних фронтів і сигналів

Підмодуль «Лічильник» (Counter) є підлеглим для модуля дискретних входів і виконує функції лічильника з розрядністю 16 біт.

Параметри модуля:

– «Номер входу» (Number of Input) – діапазон значень від 0 до 7, значення за замовчуванням – 0;

– «Фронт сигналу» (SenseEdge) – значення обираються зі списку «RISE_EDGE», «FALL_EDGE» і «BOTH_EDGE», значення за замовчуванням – «RISE_EDGE»;

– «Видимість» (Visibility) – задає видимість параметрів модуля у програмі EasyWorkPLC. Значення обираються зі списку «yes» і «no», значення за замовчуванням – «no».

Лічильник рахує кількість імпульсів, що надійшли на дискретний вхід за один цикл ПЛК. Після завершення кожного циклу ПЛК значення в лічильнику обнуляється.

Для датчиків термоперетворювачів опору (ТПО) вимагається задавати параметр більше або рівним 1,5 с. Для датчиків термопар (ТП) та уніфікованих сигналів струму або напруги задавати параметр більше або рівним 1,0 с. Із заданням значення менше 1,0 с необхідно збільшити час між вимірами для інших входів так, щоб сумарний час для усіх входів був не менше 4 с.

Підмодуль «Енкодер» (Encoder 16bit)

Енкодер – програмний модуль, що дозволяє здійснювати підключення на двох дискретних входах відносного енкодера для отримання з його допомогою даних про обертання або лінійне переміщення контрольованого механізму з подальшою передачею інформації в цифровій формі у програму ПЛК. Для роботи з високочастотними енкодерами необхідно відключати режим фільтрації за входом.

Модуль «Енкодер» (Encoder) є підпорядкованим підмодулем модуля дискретних входів. Модуль має 16-бітовий канал (формат WORD). Вікно конфігурації модуля «Енкодер» (Encoder) подано на рис. 4.14.

Параметри модуля:

– «Перший вхід» (First Input) – діапазон значень від 0 до 7, значення за замовчуванням – 0;

– «Другий вхід» (Second Input) – діапазон значень від 0 до 7, значення за замовчуванням – 1;

– «Діапазон» (Range) – діапазон значень від 0 до 65000, значення за замовчуванням – 255;

– «Тип енкодера» (Encoder Typi) – значення обираються зі списку «RING», «LINEAR», значення за замовчуванням – «RING»;

– «Видимість» (Visibility) – задає видимість параметрів модуля в програмі EasyWorkPLC. Значення обираються зі списку «yes» і «no», значення за замовчуванням – «no».

Параметри «Перший вхід» (First Input) і «Другий вхід» (Second Input) визначають номери входів ПЛК, до яких підключений енкодер.

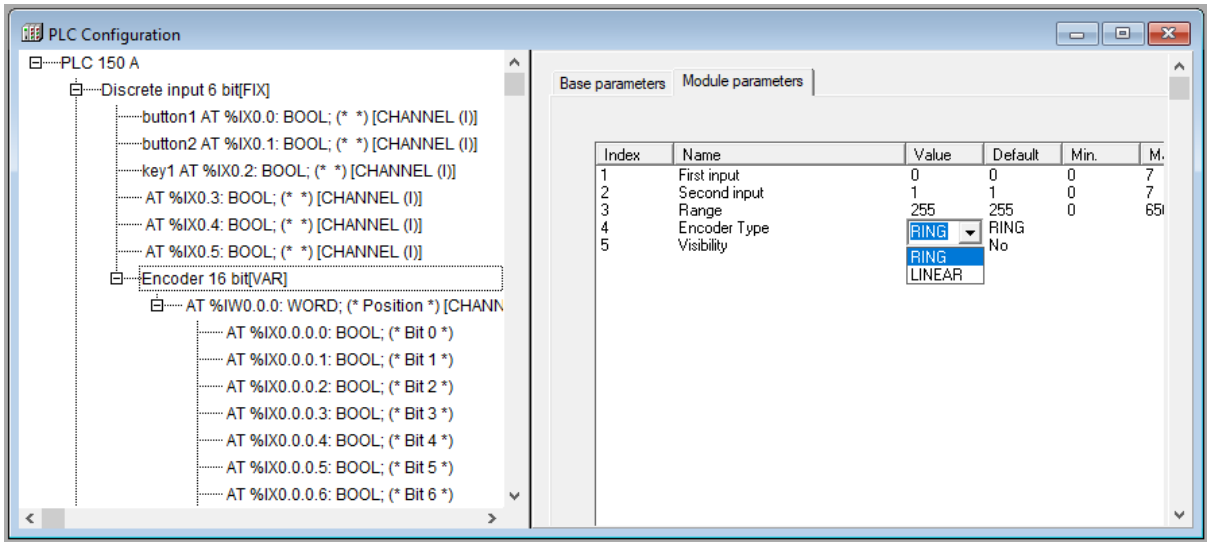


Рисунок 4.14 – Вікно конфігурації модуля «Енкодер»

Під час підключення енкодера необхідно дотримуватися обережності, щоб не помінявся напрям обертання: на виходах А і Б підключення здійснюється так:

1-й вхід = А,

2-й вхід = Б.

Параметр «Діапазон» (Range) визначає кількість імпульсів на повний оберт кругового енкодера або на повний хід лінійного.

Параметр «Тип енкодера» (Encoder Type) визначає тип енкодера: або круговий, або лінійний. Необхідно враховувати особливість підрахунку для типів енкодера:

– у круговому типі – якщо під час обертання здійснюється перехід через умовний нуль, показання лічильника скидаються, і починається новий відлік з 0;

– у лінійному типі – здійснюється фіксація досягнення максимуму або мінімуму діапазону, далі приріст не йде.

Зі зчитуванням інформації в програму ПЛК позиція не обнуляється, тобто енкодер є датчиком положення і фіксує позицію постійно в діапазоні від 0 до значення, заданого в параметрі «Діапазон» (Range).

Підмодуль «Лічильник спеціалізований» (Counter SP)

Лічильник спеціалізований – програмний модуль, що здійснює обробку цифрових входів і керований спеціальним чином. Його відмінність від Лічильника (Counter 16bit), що підключається до якого-небудь входу, полягає в

тому, що Лічильник спеціалізований додатково використовує два апаратні входи (які задаються через параметри лічильника) для того, щоб ззовні можливо було управляти стартом і зупинкою функціонування лічильника.

Модуль «Лічильник спеціалізований» (Counter SP) є підпорядкованим підмодулем модуля дискретних входів. Вікно конфігурації модуля «Лічильник спеціалізований» (Counter SP) подано на рис. 4.15.

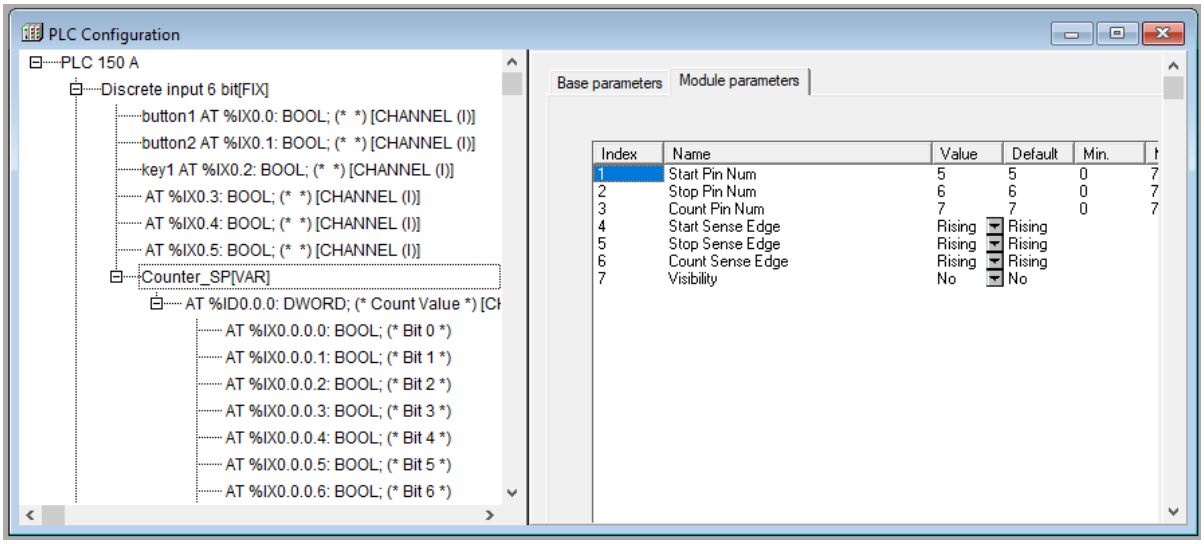


Рисунок 4.15 – Вікно конфігурації модуля «Лічильник спеціалізований»

Параметри модуля:

– «Номер входу старту» (Start Pin Num) – діапазон значень від 0 до 5, значення за замовчуванням – 5;

– «Номер входу зупинки» (Stop Pin Num) – діапазон значень від 0 до 6, значення за замовчуванням – 6;

– «Номер входу рахунку» (Count Pin Num) – діапазон значень від 0 до 7, значення за замовчуванням – 7;

– «Фронт сигналу на вході старту» (Start Sense Edge) – значення обираються зі списку «RISING_EDGE», «FALL_EDGE» і «BOTH_EDGE», значення за замовчуванням – «RISING_EDGE»;

– «Фронт сигналу на вході зупинки» (Stop Sense Edge) – значення обираються зі списку «RISING_EDGE», «FALL_EDGE» і «BOTH_EDGE», значення за замовчуванням – «RISING_EDGE»;

– «Фронт сигналу на вході рахунку» (Count Sense Edge) – значення вибираються зі списку «RISING_EDGE», «FALL_EDGE» і «BOTH_EDGE», значення за замовчуванням – «RISING_EDGE»;

Спеціалізований Лічильник рахує кількість імпульсів, що надійшли на дискретний вхід. У каналі Спеціалізованого Лічильника значення не обнуляється у кожному циклі ПЛК (на відміну від модуля Лічильника), а накопичується. Обнуління даних у каналі проводиться з кожним новим стартом.

Параметри «Номер входу старту» (Start Pin Num) і «Номер входу зупинки» (Stop Pin Num) визначають номери входів ПЛК, через які здійснюється управління стартом і зупинкою функціонування лічильника.

Параметр «Номер входу рахунку» (Count Pin Num) визначає номер входу ПЛК, на якому проводиться накопичення (підсумовування) імпульсів, врахованих лічильником.

Параметри «Фронт сигналу на вході старту» (Start Sense Edge). «Фронт сигналу на вході зупинки» (Stop Sense Edge) і «Фронт сигналу на вході рахунку» (Count Sense Edge) визначають фронти сигналів, якими здійснюватиметься робота – старт і зупинка рахунку і власне рахунок імпульсів.

Вибір з трьох варіантів спрацьовування можливий тільки у тому випадку, якщо заданий режим фільтрації входу, тобто значення часу фільтрації більше 0. Якщо час фільтрації не задано (встановлено в 0), особливості апаратної реалізації ОВЕН ПЛК забороняють роботу в режимі BOTH_EDGE.

4.2.2 Конфігурація модулів дискретного виходу (Discrete output)

Модуль дискретного виходу (Discrete output) відображає в області пам’яті вводу/виводу значення дискретного виходу ПЛК.

Модуль має декілька бітових каналів (кількість каналів залежить від варіанта виконання контролера).

Параметри модуля:

– «Безпечне значення» (Save Value) – TRUE або FALSE для дискретного виходу;

– «Видимість» (Visibility) – задає видимість параметрів модуля в програмі EasyWorkPLC. Значення обираються зі списку «yes» і «no», значення за замовчуванням – «no».

Залежно від виконання ПЛК може бути оснащений електромагнітними реле або здвоєними транзисторними ключами. Під час комплектації здвоєними ключами один канал дискретного виходу має два бітові канали. Під час комплектації електромагнітними реле один канал дискретного виходу може

утримувати один бітовий канал або декілька бітових каналів, рівних кількості реле в контролері.

На рис. 4.16 наведені варіанти модуля дискретного виходу, що утримує один бітовий канал і чотири бітові канали.

У варіантах модуля дискретних входів, що містять два бітових і чотири бітові канали, значення параметра «Безпечний стан виходу» (Save Value) встановлюється для кожного бітового каналу.

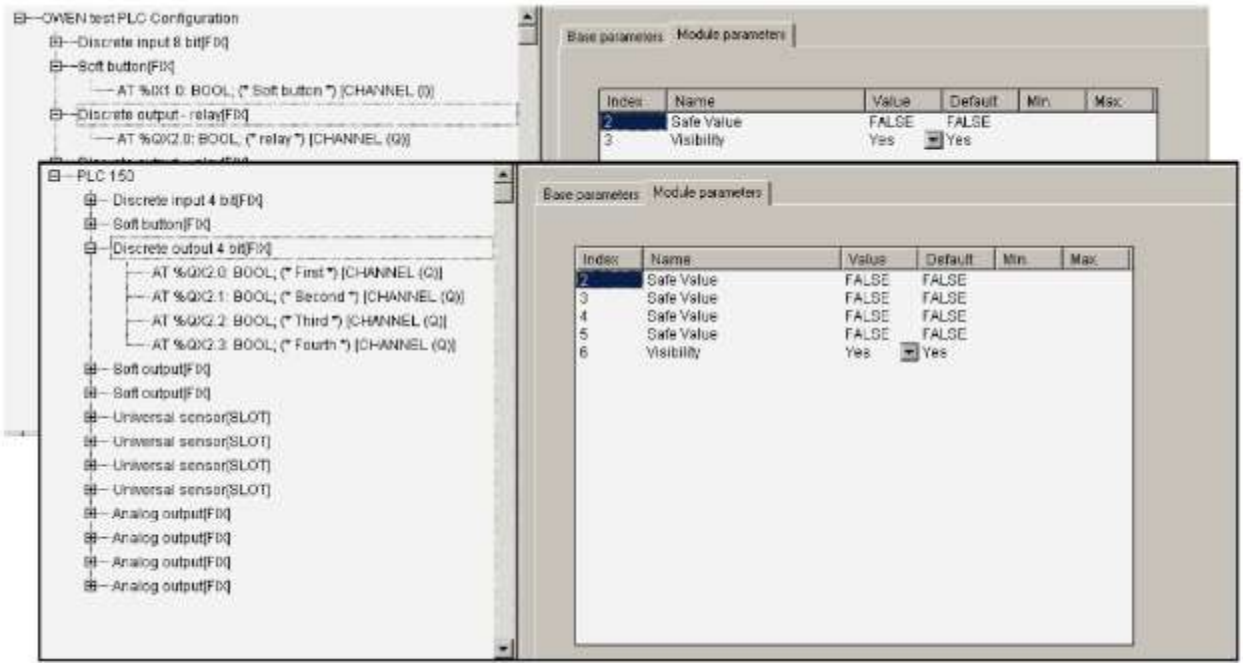


Рисунок 4.16 – Вікно конфігурації модуля дискретного виходу

Параметр «Безпечний стан виходу» (Save Value)

Призначення параметра «Безпечний стан виходу» такий.

У момент завантаження ПЛК або за якого-небудь серйозного збою, наприклад із «зависанням» ПЛК тощо, його виходи можуть опинитися в невизначеному стані: вимкнені або увімкнені. Проте цей стан може виявитися неприпустимим у ході експлуатації керованого устаткування. Для виключення такої ситуації ПЛК переводить виходи під час «зависання» або під час завантаження в стан, заданий у параметрі «Безпечний стан виходу» (Save Value). Значення параметра FALSE означає, що вихід вимкнений (=0). TRUE – вихід увімкнений (=1).

Список підмодулів для модуля дискретного виходу:

– ШІМ (Pulse – wide modulator);

– генератор (Generator).

Підмодуль ШІМ (Pulse – wide modulator)

Модуль ШІМ – програмний модуль, призначений для забезпечення функціонування генератора широтно-імпульсної модуляції, підключеного до дискретного виходу.

Модуль ШІМ (Pulse – wide modulator) є підпорядкованим підмодулем модуля дискретного виходу. Модуль має 16-бітовий канал (формат WORD).

Вікно конфігурації модуля ШІМ (Pulse – wide modulator) подане на рис. 4.17.

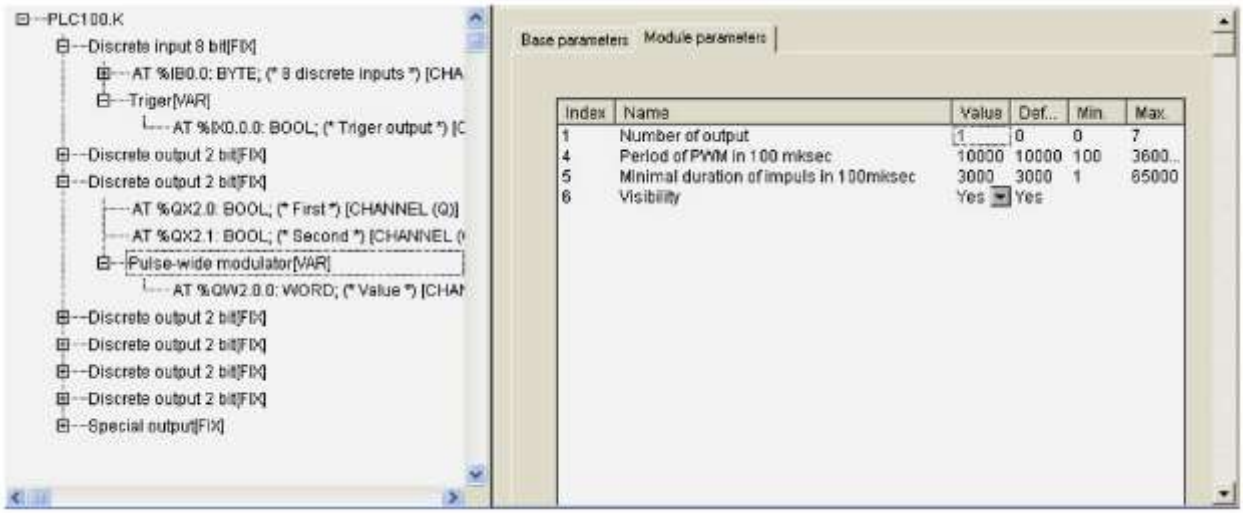


Рисунок 4.17 – Вікно конфігурації модуля ШІМ

Параметри модуля:

– «Номер виходу» (Number of output) – діапазон значень від 0 до 7, значення за замовчуванням – 0. Параметр набуває значень від 0 до (n – 1), де *n* – кількість бітових каналів у модулі, до якого здійснюється підключення;

– «Період ШІМ в 100 мкс» (Period of PWM in 100 mksec) – діапазон значень від 100 до 360000, значення за замовчуванням – 100 мкс;

– «Мінімальна тривалість імпульсу ШІМ в 100 мкс» (Minimal duration of impulse in 100 mksec) – діапазон значень від 1 до 65000, значення за замовчуванням – 30 од. 1 од. = 100 мкс;

– «Видимість» (Visibility) – задає видимість параметрів модуля в програмі EasyWorkPLC. Значення вибираються зі списку «yes» і «no», значення за замовчуванням – «no».

З додаванням модуля ШІМ з’являється канал, у який записується значення шпаруватості ШІМ (від 0 до 65535 (від 0 до 100% потужності)).

У параметрі «Номер виходу» (Number of output) в модулі ШІМ – задається номер виходу ПЛК, до якого підключений ШІМ-генератор.

У параметрі «Період ШІМ в 100 мкс» (Period of PWM in 100 mksec) – задається тривалість одного періоду ШІМ-регулювання. Набуває значень від 100 до 360000 од. відповідно, задаючи період ШІМ від 10 мс до 36 с.

У параметрі «Мінімальна тривалість імпульсу ШІМ у 100 мкс» (Minimal duration of impulse in 100 mksec) встановлюється обмеження на мінімальну тривалість імпульсу ШІМ.

Підмодуль «Генератор» (Generator)

Модуль «Генератор» (Generator) – програмний модуль, що розширює функціонал дискретного виходу і здійснює за рахунок апаратного забезпечення процесора функціонування виходу в режимі високоточного і високошвидкісного генератора зі змінюваною шпаруватістю.

Модуль «Генератор» (Generator) може бути застосований і як ШІМ-генератор за рахунок зміни шпаруватості.

Особливості апаратної реалізації ОВЕН ПЛК дозволяють здійснювати генерування сигналу тільки для ПЛК100-К і тільки через одинадцятий вихід. В інших модифікаціях ОВЕН ПЛК модуль «Генератор» (Generator) не реалізований. При цьому вихід ПЛК100-К здатний видавати імпульси, як одиничні, так і нульові, тривалістю не менше 30 мкс.

Модуль «Генератор» (Generator) є підпорядкованим підмодулем модуля дискретного виходу і має три канали виводу, в яких записуються значення:

– частоти («Frequency») – від 1 до 10 кГц;

– шпаруватості («Q – Duty Circle») – від 0 до 999 (від 0 до 99,9% з точністю до 0,1%);

– число імпульсів/стан модуля («Amount Ticks»).

В останній канал записується число імпульсів, яке вимагається згенерувати. Під час запису числа 0x0 генерація імпульсів припиняється, під час запису числа 0xffffffff відбувається нескінченно.

Параметри модуля:

– «Інверсія сигналу» (Invertion) – значення обираються зі списку «yes» і «no», значення за замовчуванням – «no».

Вікно конфігурації модуля «Генератор» (Generator) подано на рис. 4.18.

У параметрі «Інверсія сигналу» (Invertion) в модулі «Генератор» (Generator) визначається необхідність інверсії сигналу на виході модуля.

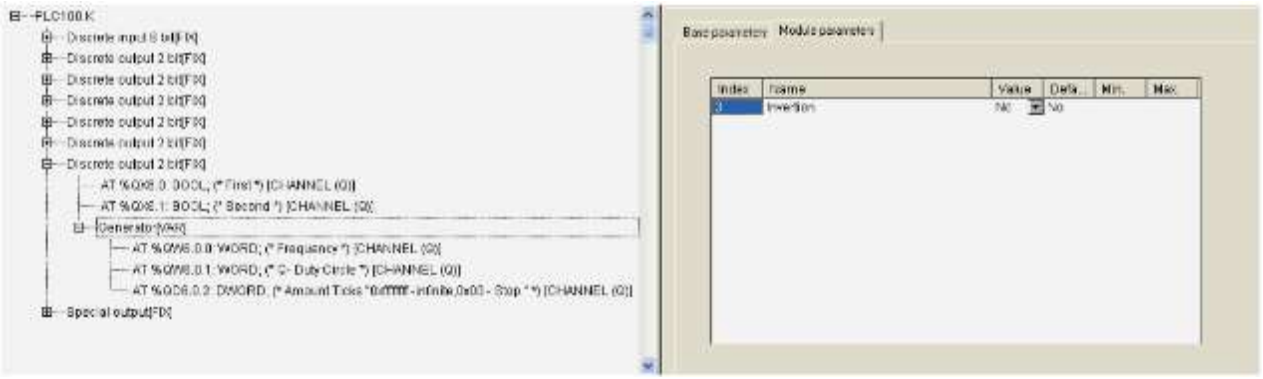


Рисунок 4.18 – Вікно конфігурації модуля «Генератор»

4.2.3 Модулі аналогових входів

Модулі аналогових входів належать до стаціонарних модулів. Залежно від конкретної моделі ПЛК, їх кількість може бути різною. Для них у Конфігураторі ПЛК закладена відповідна кількість модулів аналогових входів. Залежно від поставленого завдання можуть бути використані різні модулі.

Користувач обирає необхідні модулі функцією контекстного меню Replace Elements з чотирьох можливих варіантів у випадному списку:

– датчик уніфікованого сигналу;

– датчик типу «термопара» (перетворювач термоелектричний);

– датчик типу «термоопір» (термоперетворювач опору);

– контактний датчик.

Модулі призначені для наведення результатів виміру до значень фізичної величини, вимірюваної датчиком. Приведене значення фізичної величини може бути прочитане в програму ПЛК.

Рис. 4.19 ілюструє можливість вибору варіанта аналогового входу.

У кожного типу модуля аналогових входів є як співпадаючі з іншими параметри, так і ті, що відрізняються, характерні тільки для конкретного датчика.

Модуль «Датчик уніфікованого сигналу» (Unified Signal Sensor)

Параметри модуля:

– «Тип датчика» (Type of sensor) – значення вибираються зі списку, що містить сім можливих типів датчиків, значення за замовчуванням, – IT_4_20 (датчик з уніфікованим сигналом постійного струму від 4 до 20 мА);

– «Час між вимірами в секундах» (Measure interval, s) – максимальний час між вимірами не обмежений, значення може бути будь-яким, в т.ч. дробовим, мінімальне – 1 с, значення за замовчуванням – 1;

– «Нижня і верхня межі» (Ain low & Ain high) – межі діапазону виміру фізичної величини, значення за замовчуванням – 0;

– «Перша, друга і третя точка» і «зміна, що вводиться» (First point, Second point, Third point & Delta), – три точки корекції вироблюваного виміру і зміни, що вводяться, значення за замовчуванням – 0.

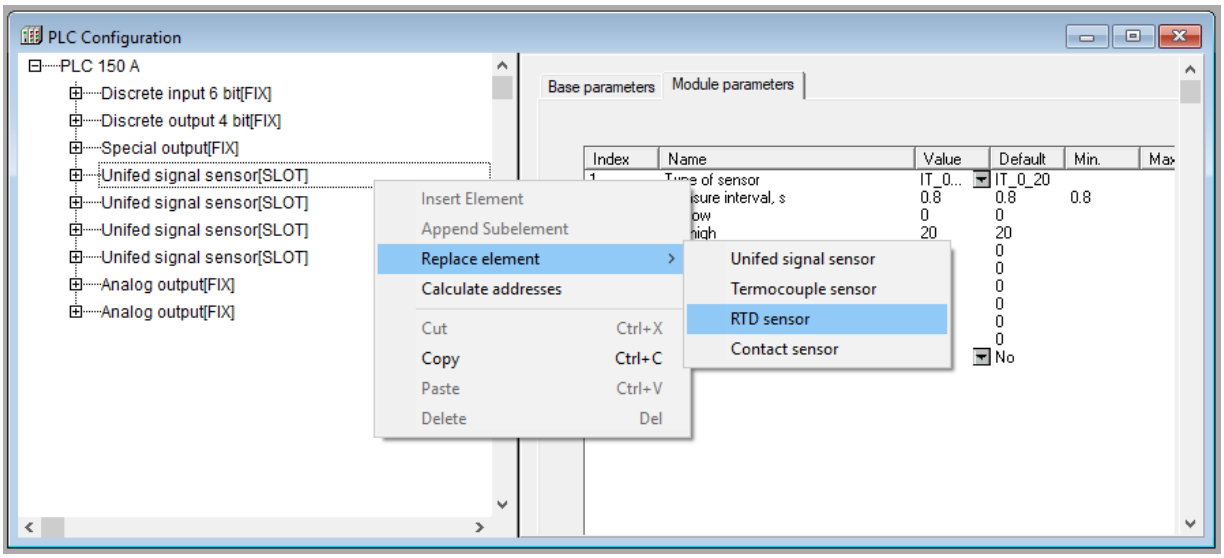


Рисунок 4.19 – Вікно вибору типу модуля аналогового входу

Модуль має два канали:

– канал Value («Значення»), формат Real – значення, що отримується на виході ПЛК з урахуванням усіх поправок, приведене до значень вимірюваної фізичної величини. У випадку помилки датчика (і виході за діапазон) значення містить код помилки в спеціальному форматі;

– канал Circular time («Циклічний час»), формат Word – значення циклічного часу виміру в діапазоні від 0 до 65536 од. (1 од.=10 мс). Відлік часу починається від моменту старту програми ПЛК і обнуляється під час переходу через максимальне значення.

Analog Input – підмодуль, необхідний для організації структури.

Вікно конфігурації модуля аналогового входу «Датчик уніфікованого сигналу» з вкладкою параметрів подано на рис. 4.20.

У параметрі «Тип датчика» (Type of sensor) задається тип датчика, показання якого оброблятимуться модулем «Датчик уніфікованого сигналу». Користувач здійснює вибір з семи можливих варіантів у випадному списку:

– датчики з уніфікованим сигналом струму 0...20 мА, 4...20 мА, 0...5 мА;

– датчики з уніфікованим сигналом напруги -50...+50 мВ, 0...1 В, 0...10 В;

– опір від 0 до 5000 Ом.

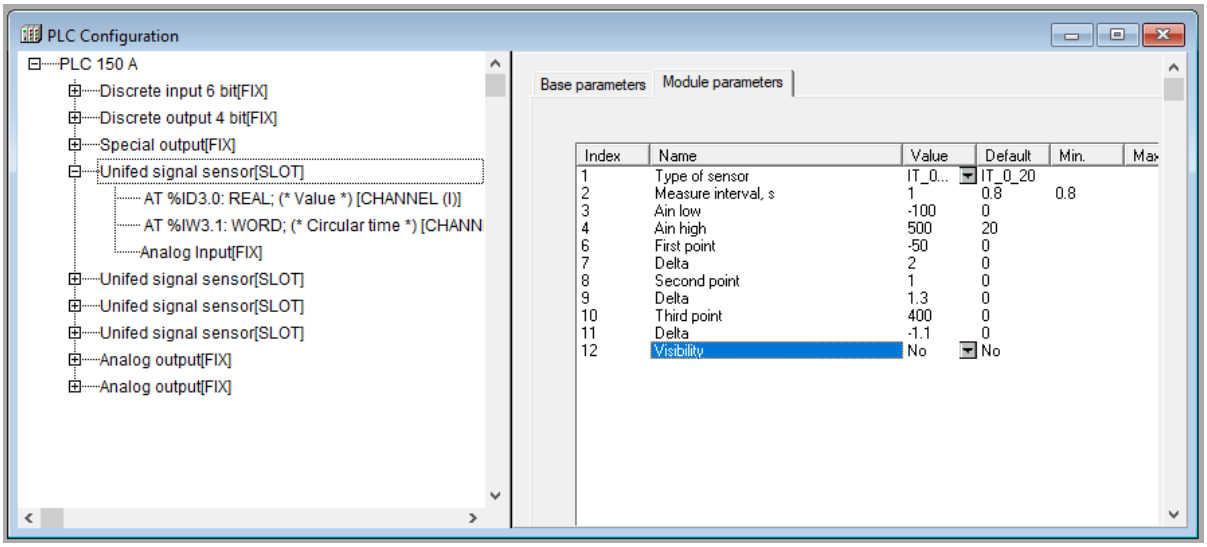


Рисунок 4.20 – Вікно конфігурації модуля аналогового входу «Датчик уніфікованого сигналу» з вкладкою параметрів

У параметрі «Час між вимірами в секундах» (Measure interval, s) задають інтервал часу між опитуваннями датчика.

У параметрах «Нижня і верхня межі» (Ain low & Ain high) задають діапазон вимірів датчика уніфікованого сигналу.

Наприклад, датчик з уніфікованим сигналом «параметр – струм 0...20мА» вимірює температуру в діапазоні – -100 °С...+500 °С, і 0 мА відповідає нижній межі діапазону виміру – - 100 °С, а 20 мА відповідає верхній межі діапазону виміру – +500 °С. Тоді вихідна величина модуля змінюватиметься від - 100 до +500. Перетворення лінійне на усьому діапазоні.

У параметрах «Перша, друга і третя точка» і «зміна, що вводиться» (First point і Delta), (Second point і Delta) і (Third point і Delta) задають коригування лінійності датчика.

Модуль дозволяє виконувати корекцію за трьома точками (поліном другого ступеня). Три групи параметрів визначають три точки корекції вироблюваних вимірів. За замовчуванням значення точок корекції нульові, що означає, що жодна точка не використовується. Як тільки значення [xxx]_Point встановлюються ненульовими, датчик їх використовує, проводячи коригування полінома за заданими точкам виміру і поправками Delta у цих точках.

Технологія корекції така. Наприклад, під час калібрування визначено, що з виміром у першій точці мінус 50 °С, датчик показує значення мінус 48 °С. Вноситься поправка мінус 2 °С, із складанням з якою виходить реальне значення, що використовується в подальшому коригуванні роботи датчика. Так само і з іншими точками. Бажано, щоб під час калібрування точки корекції

були розташовані далеко одна від одної. Інакше можуть бути великі погрішності, нахил графіку буде невірним.

Якщо корекція відбувається за однією точкою, то уся крива зміщується на одну величину, що коригує, за двома точками – зміщується і змінюється нахил, за трьома – для внесення корекції використовується поліном другого ступеня (змінюється форма кривої).

Точки задаються, починаючи з першої. Якщо буде задана друга або третя точки, але не задана перша, то контролер сприйме це як помилку.

Модуль «Датчик типу Термопара» (Thermocouple sensor)

Параметри модуля:

– «Тип датчика» (Type of sensor) – значення вибираються зі списку, що містить можливі типи датчиків, значення за замовчуванням, – TP_L (термопара ТХК (L) з діапазоном виміру [– 200 °С...+800 °С]);

– «Час між вимірами в секундах» (Measure interval, s) – максимальний час між вимірами не обмежений, значення може бути будь-яким, в т.ч. дробовим, мінімальне – 1 с, значення за замовчуванням – 1;

– «Компенсація холодного спаю» (Cold junction compensation) – значення обираються зі списку «On» (Увімкнути) і «Off» (Вимкнути), значення за замовчуванням – «On»;

– «Нижня і верхня межі» (Ain low & Ain high) – межі діапазону виміру фізичної величини, значення за замовчуванням – 0;

– «Перша, друга і третя точка» і «зміна, що вводиться» (First point, Second point, Third point & Delta), – три точки корекції вироблюваного виміру і зміни, що вводяться, значення за замовчуванням – 0.

Модуль має два канали:

– канал Value («Значення»), формат Real – значення, що отримується на виході ПЛК з урахуванням усіх поправок, приведене до значень вимірюваної фізичної величини. З помилкою датчика (і виходом за діапазон) значення містить код помилки у спеціальному форматі;

– канал Circular time («Циклічний час»), формат Word – значення циклічного часу виміру в діапазоні від 0 до 65536 од. (1 од.=10 мс). Відлік часу починається від моменту старту програми ПЛК і обнуляється під час переходу через максимальне значення.

Analog Input – підмодуль, необхідний для організації структури.

Перелік параметрів аналогового входу «Термопара» ідентичний переліку параметрів модуля аналогового входу «Датчик уніфікованого сигналу». Різниця полягає у виборі можливих значень параметра «Тип датчика» (Type of sensor). Користувачеві дається на вибір випадний список можливих термопар різних типів. Ці самі типи термопар використовуються в усіх приладах з універсальним входом розробки ПО «ОВЕН», наприклад, у ТРМ148.

Вікно конфігурації аналогового входу «Термопара» з вкладкою параметрів подано на рис. 4.21.

У параметрі «Компенсація холодного спаю» (Cold junction compensation) визначають необхідність компенсації холодного спаю. Відключення компенсації холодного спаю потрібно для проведення метрологічної перевірки аналогових входів ПЛК. У звичайному режимі роботи рекомендується вмикати компенсацію холодного спаю.

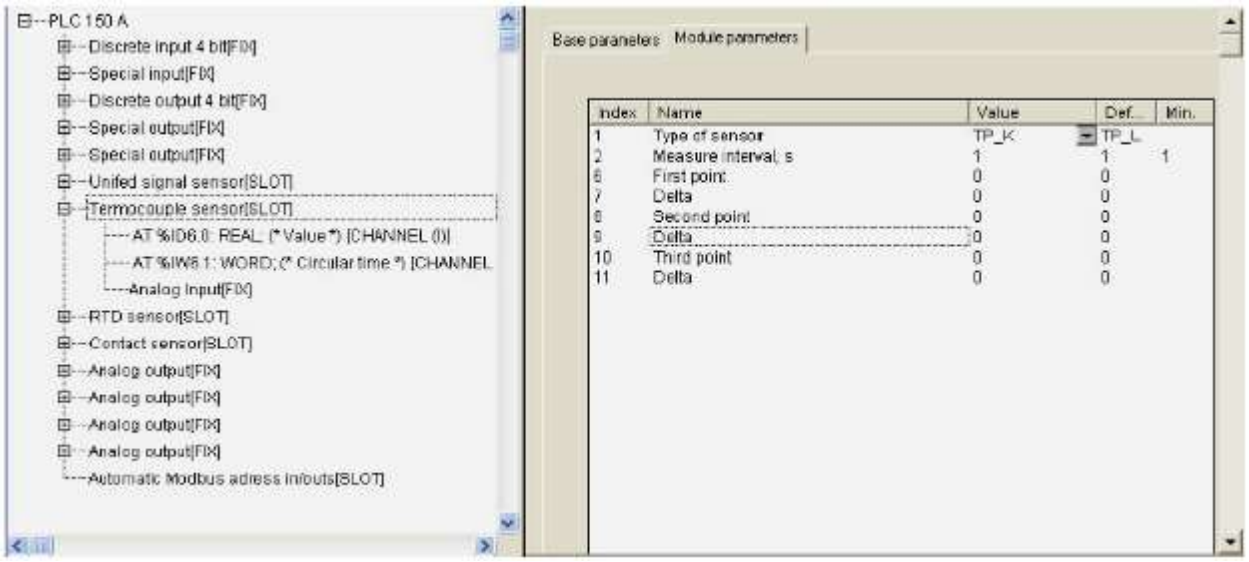


Рисунок 4.21 – Вікно конфігурації аналогового входу «Термопара»

Модуль «Датчик типу Термоопір» (RTD sensor)

З підключенням датчика типу «Термоопір» за двопровідною схемою необхідно виконати компенсацію опору сполучних проводів за методикою, викладеною нижче.

Датчики типу ТСМ50, ТСП100 і ТСМ100 мають відносно невеликий опір, порівнянний з опором сполучних проводів. Через це опір проводів вносить велику додаткову погрішність. Звичайне підключення таких термоперетворювачів опору здійснюється за трипровідною схемою, що

дозволяє вимірювати і компенсувати опір проводів, але ПЛК150 не має можливості підключення датчиків за трипровідною схемою. Тому з підключенням датчиків ТСМ50, ТСП100 і ТСМ100 за двопровідною схемою необхідно виконати такі дії:

1. Підключити лінію зв'язку «прилад-датчик» до контролера згідно зі схемами підключення.

2. Налаштувати контролер на роботу з датчиком необхідного типу.

3. З боку датчика до лінії зв'язку підключити магазин опорів з класом точності не менше 0,1 або еталонний резистор.

4. Встановити на магазині опорів значення 50 Ом для датчиків ТСМ50 або 100 Ом для датчиків ТСМ100 і ТСП100.

5. Зафіксувати результати виміру на аналоговому вході контролера. (Результати виміру можна викликати на вкладці, подвійним натисканням лівою кнопкою маніпулятора «мишка» із встановленням курсора на каналі модуля).

6. У параметрі First point модуля необхідного аналогового входу задати значення 0 (що відповідає значенню 0 °С) і в першому параметрі Delta задати значення, рівне виміряному, але із зворотним знаком.

7. Повторно зафіксувати результати виміру на аналоговому вході контролера і переконатися, що вони рівні або близькі до 0.

8. Під'єднати замість магазину опорів датчик.

Параметри модуля:

– «Тип датчика» (Type of sensor) – значення вибираються зі списку, що містить можливі типи датчиків, значення за замовчуванням, – r385_50 (термоперетворювач опору ТСП Pt 50 ($\alpha = 0{,}00385\ °C^{-1}$);

– «Час між вимірами в секундах» (Measure interval, s) – максимальний час між вимірами не обмежений, значення може бути будь-яким, в т.ч. дробовим, мінімальне – 1 с, значення за замовчуванням – 1;

– «Нижня і верхня межі» (Ain low & Ain high) – межі діапазону виміру фізичної величини, значення за замовчуванням – 0;

– «Перша, друга і третя точка» і «зміна, що вводиться» (First point, Second point, Third point & Delta), – три точки корекції вироблюваних вимірів і зміни, що вводяться, значення за замовчуванням – 0.

Модуль має два канали:

– канал Value («Значення»), формат Real – значення, що отримується на виході ПЛК з урахуванням усіх поправок, приведене до значень вимірюваної

фізичної величини. З помилкою датчика (і виходом з діапазону) значення містить код помилки у спеціальному форматі.

– канал Circular time («Циклічний час»), формат Word – значення циклічного часу виміру в діапазоні від 0 до 65536 од. (1 од.=10 мс). Відлік часу починається від моменту старту програми ПЛК і обнуляється під час переходу через максимальне значення.

Analog Input – підмодуль, необхідний для організації структури.

Перелік параметрів аналогового входу «Термоопір» ідентичний перелікам параметрів аналогових входів «Термопара» і «Датчик уніфікованого сигналу». Різниця полягає у виборі можливих значень параметра «Тип датчика» (Type of sensor). Користувачеві дається на вибір випадний список можливих типів термоперетворювача опору.

Перелік використовуваних типів датчиків подано в таблиці 4.1.

Таблиця 4.1 – Перелік підтримуваних датчиків

Тип датчика або вхідного сигналу	Позначення в параметрі «Typi of sensor»	Діапазон вимірів
Термоперетворювачі опору за ДСТУ 2858:2015 (Termoresistor sensors)		
ТСМ Cu 50 (α = 0,00426 °C^{-1})	R426_50	–50 °C...+200 °C
ТСМ 50М (α = 0,00428 °C^{-1})	R428_50	–190 °C...+200 °C
ТСП Pt 50 (α = 0,00385 °C^{-1})	R385_50	–200 °C...+750 °C
ТСП 50П (α = 0,00391 °C^{-1})	R391_50	–200 °C...+750 °C
ТСМ Cu 100 (α = 0,00426 °C^{-1})	R426_100	–50 °C...+200 °C
ТСМ 100М (α = 0,00428 °C^{-1})	R428_100	–190 °C...+200 °C
ТСП Pt 100 (α = 0,00385 °C^{-1})	R385_100	–200 °C...+750 °C
ТСП 100П (α = 0,00391 °C^{-1})	R391_100	–200 °C...+750 °C
ТСН 100Н (α = 0,00617 °C^{-1})	R617_100	–60 °C...+180 °C
ТСМ Cu 500 (α = 0,00426 °C^{-1})	R426_500	–50 °C...+200 °C
ТСМ 500М (α = 0,00428 °C^{-1})	R428_500	–190 °C...+200 °C
ТСП Pt 500 (α = 0,00385 °C^{-1})	R385_500	–200 °C...+750 °C
ТСП 500П (α = 0,00391 °C^{-1})	R391_500	–200 °C...+750 °C
ТСН 500Н (α = 0,00617 °C^{-1})	R617_500	-60 °C...+180 °C
ТСМ Cu 1000 (α = 0,00426 °C^{-1})	R426_1000	–50 °C...+200 °C
ТСМ 1000М (α = 0,00428 °C^{-1})	R428_1000	–190 °C...+200 °C
ТСП Pt 1000 (α = 0,00385 °C^{-1})	R385_1000	–200 °C...+750 °C
ТСП 1000П (α = 0,00391 °C^{-1})	R391_1000	–200 °C...+750 °C
ТСН 1000Н (α = 0,00617 °C^{-1})	R617_1000	–60 °C...+180 °C

Продовження таблиці 4.1

Тип датчика або вхідного сигналу	Позначення в параметрі «Type of sensor»	Діапазон вимірів
Перетворювачі термоелектричні за ДСТУ EN 60584-1:2016 (Termocouple sensors)		
ТХК (L)	TP_L	–200 °C...+800 °C
ТЖК (J)	TP_J	–200 °C...+1200 °C
ТНН (N)	TP_N	–200 °C...+1300 °C
ТХА (К)	TP_ДО	–200 °C...+1300 °C
ТВП (S)	TP_S	0°C...+1600 °C
ТВП (R)	TP_R	0°C...+1600 °C
ТВР (А-1)	TP_WR1	0 °C...+2500 °C
ТВР (А-2)	TP_WR2	0 °C...+1800 °C
ТВР (А-3)	TP_WR3	0°C...+1600 °C
ТМК (Т)	TP_T	–200 °C...+400 °C
Уніфіковані сигнали постійної напруги і струму за ГОСТ 26.011-80 (Universal sensors)		
0...5.0 мА	IT 0 5	0...100 %
0...20,0 мА	IT 0 20	0...100 %
4,0...20,0 мА	IT 4 20	0...100 %
–50,0...+50,0 мВ	mV	0...100 %
0...1.0 В	U0 1	0...100 %
0...10.0 В	U0 10	0...100 %
Датчики опору (Universal sensors)		
резистивний (0... 5000 Ом)	R0_5000	0...100 %

Вікно конфігурації аналогового входу «Термоопір» подано з вкладкою параметрів на рис. 4.22.

Модуль «Контактний датчик» (Contact sensor)

Тип модуля аналогового входу «Контактний датчик» використовується в тих випадках, коли за конкретної схеми застосування ПЛК не достатньо дискретних входів і доводиться використовувати один з аналогових входів.

Параметри модуля:

– «Тип датчика» (Type of sensor) – значення обираються зі списку, що містить можливі типи датчиків, значення за замовчуванням – КТ;

– «Час між вимірами в секундах» (Measure interval, s) – максимальний час між вимірами не обмежений, значення може бути будь-яким, в т.ч. дробовим, мінімальне – 1 с, значення за замовчуванням – 1 с.

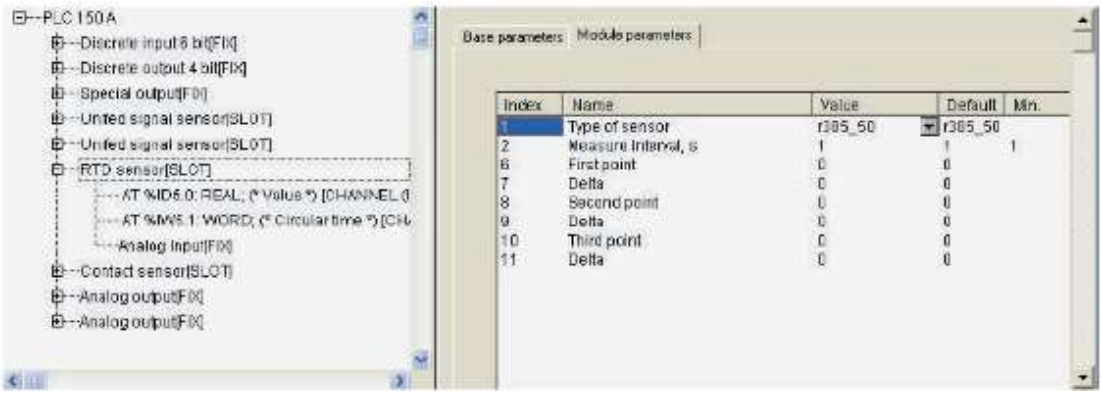


Рисунок 4.22 – Вікно конфігурації аналогового входу «Термоопір»

Модуль має два канали:

– канал Value («Значення»), формат Real – значення, що отримується на виході ПЛК з урахуванням усіх поправок, приведене до значень вимірюваної фізичної величини;

– канал Circular time («Циклічний час»), формат Word – значення циклічного часу виміру в діапазоні від 0 до 65536 од. (1 од.=10 мс).

Відлік часу починається від моменту старту програми ПЛК і обнуляється під час переходу через максимальне значення.

Analog Input – підмодуль, необхідний для організації структури.

Вихідне значення модуля «Контактний датчик» може дорівнювати 1 або 0. Значення 1 означає, що опір контакту, який замикає загальну клему аналогових входів і клему конкретного входу, менше 50 Ом.

Розмикання контакту, а також опори більше 50 Ом інтерпретуються як нульове вихідне значення. Вікно конфігурації аналогового входу «Контактний датчик» з вкладкою параметрів подано на рис. 4.23.

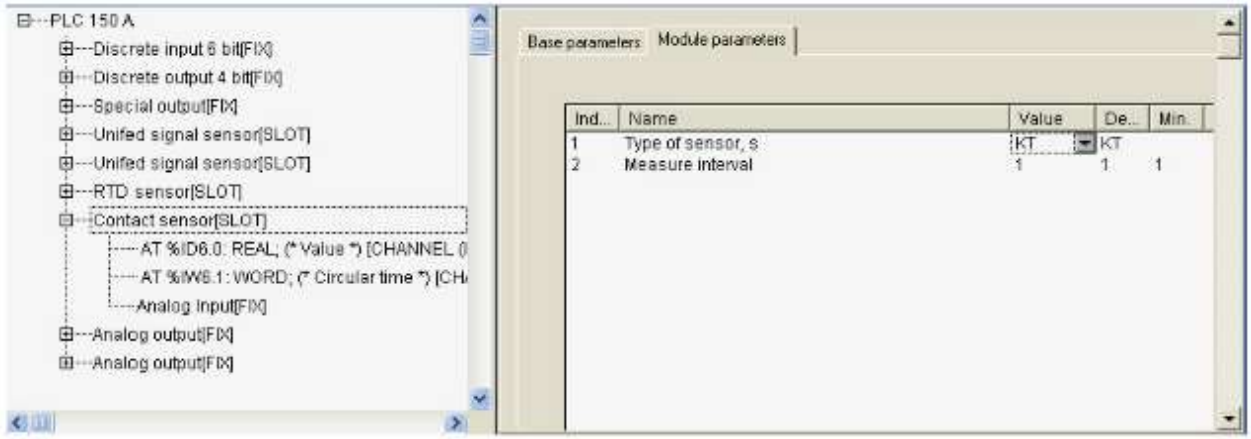


Рисунок 4.23 – Вікно конфігурації аналогового входу «Контактний датчик»

4.2.4 Модулі аналогових виходів (Analog output)

Аналогові виходи належать до стаціонарних модулів. Залежно від типу встановлених у ПЛК ЦАП існують три варіанти налаштування модуля аналогових виходів :

– універсальний – програмно-налаштовуваний або на ЦАП «параметр – струм», або на ЦАП «параметр – напруга»;

– фіксований, призначений для ЦАП «параметр – струм»;

– фіксований, призначений для ЦАП «параметр – напруга».

Параметри модуля:

– «Тип виходу» (Type) – значення обираються зі списку «Current 0-20 mA» і «Voltage 0-10 V», значення за замовчуванням – залежить від типу ЦАП;

– «Корекція в початковій точці» (Null correction) і «Корекція у кінцевій точці діапазону» (Full range correction) – значення обираються, виходячи з необхідного коригування, значення за замовчуванням – 0;

– «Безпечне значення» (Save Value) – значення задається у тих самих одиницях, що й основні значення за замовчуванням – 0;

– «Видимість» (Visibility) – задає видимість параметрів модуля в програмі EasyWorkPLC. Значення обираються зі списку «yes» і «no», значення за замовчуванням – «no».

На рис. 4.24 наведено вікно налаштування модуля аналогового виходу з вкладкою параметрів.

Параметр «Тип виходу» (Typi) задає тип аналогового виходу. Можливі значення: «параметр – струм 4...20 мА» або «параметр – напруга 0...10 В».

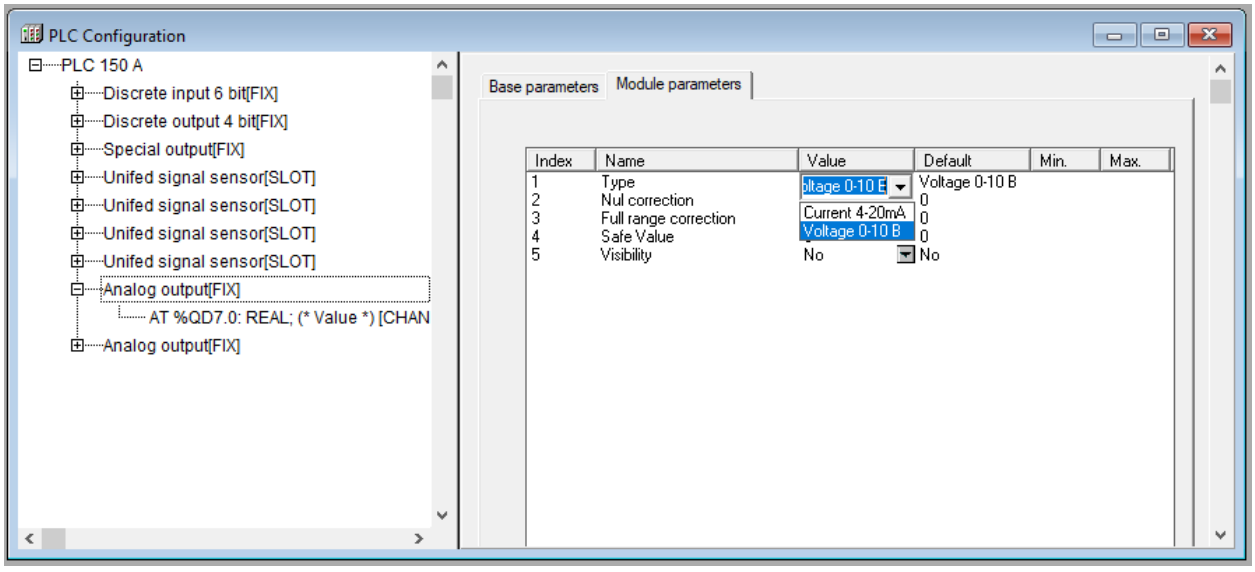


Рисунок 4.24 – Вікно налаштування модуля аналогового виходу з вкладкою параметрів

Зміна значення параметра може бути заблокована із встановленням у ПЛК ЦАП фіксованого типу.

У канал аналогового модуля з програми ПЛК передається число, що дорівнює бажаному вихідному значенню фіксованого модуля аналогового входу. Для ЦАП типу «Струм» – це значення від 4 до 20, для напруги – від 0 до 10. Значення передаються у форматі Real.

Група параметрів «Корекція в початковій точці» (Null correction) і «Корекція в кінцевій точці діапазону» (Full range correction) визначає дві точки корекції. У параметрах корекції можуть бути задані значення, що коригують, у разі неточної роботи ЦАП. Точки обрані так: для ЦАП «параметр – струм» 4 і 15 мА, а для ЦАП «параметр – напруга» – 1 і 10 В.

4.3 Контрольні запитання та завдання

1. Поясніть загальні принципи програмування ПЛК.
2. Наведіть структуру проекту в CODESYS.
3. Як відбувається конфігурація модулів дискретного входу?
4. Які типи модулів існують у CODESYS?
5. Які підмодулі можна використовувати для модуля дискретних входів?
6. Що таке ШІМ та як можна налаштувати роботу дискретного виходу в режимі ШІМ?
7. Як відбувається конфігурація модулів дискретного виходу (Discrete output)?
8. Які параметри впливають на роботу аналогових входів?
9. Поясніть методику компенсації опору сполучних проводів.
10. Як відбувається конфігурація модулів аналогових виходів?

5 ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ ПЛК

5.1 Загальні елементи програмування мовами стандарту МЕК

Перш ніж переходити до вивчення програмування мовами стандарту МЕК, необхідно ознайомитися із загальними елементами цих мов. Загальні елементи є єдиним фундаментом, що дозволяє об’єднати багатомовні компоненти в одному проекті.

5.1.1 Типи даних

Тип даних змінної визначає рід інформації, діапазон представлення і безліч допустимих операцій. Мови МЕК використовують ідеологію чіткої перевірки типів даних. Це означає, що будь-яку змінну можна використовувати тільки після її оголошення. Привласнювати значення однієї змінної іншій можна, тільки якщо вони обидві одного типу. Допускається також присвоювання значення змінної сумісного типу, що має більш широку множину допустимих значень. У цьому випадку відбувається неявне перетворення типу без втрат. Неявні перетворення типів даних із втратами заборонені. Так, наприклад, логічну змінну, здатну приймати тільки два значення (логічні 0 і 1), можна привласнити змінній типу SINT (-128...+127), але не навпаки.

Під час трансляції програми всі подібні спроби відслідковуються і вважаються грубими помилками. Якщо ж це дійсно необхідно, то виконати присвоювання з втратами можливо, але тільки за допомогою спеціальних операторів. Оператори перетворення в МЕК виконують також і більш складні операції, наприклад перетворення числа або календарної дати в текстовий рядок, і навпаки.

Найбільша різноманітність типів даних у стандарті передбачена для представлення цілих чисел. Сенс застосування широкого спектра цілочисельних змінних полягає насамперед в оптимізації коду програми. Швидкість обчислень залежить від того, як мікропроцесор оперує зі змінними даного типу. Так, цілком очевидно, що 16-розрядний процесор виконує складання двох 16-розрядних значень однією командою. Додавання ж двох значень 32-розрядних змінних – це підпрограма з декількох команд.

Додаткові затримки можуть утворюватися за рахунок мультиплексування шини даних, що зв'язують процесор і пам'ять, особливостей мікросхем пам'яті тощо. У загальному випадку менші за діапазоном представлених значень типи змінних вимагають менше пам'яті, менше коду, і обчислення за їх участі виконуються значно швидше.

Типи даних МЕК поділяються на дві категорії – елементарні і складені. Елементарні або базові типи є основою для побудови складених типів. До складених типів належать перерахування, масиви, структури, масиви структур тощо.

5.1.2 Елементарні типи даних

Цілочисельні типи

Цілочисельні змінні відрізняються різним діапазоном даних, що зберігається і, відповідно, різними вимогами до пам'яті. Детально дані характеристики наведені в таблиці 5.1.

Таблиця 5.1 – Основні типи даних та їх характеристики

Тип	Нижня границя	Верхня границя	Розмір, у байтах
BYTE	8 біт		1
WORD	16 біт		2
DWORD	32 біта		4
LWORD	64 біта		8
SINT	–128	127	1
INT	–32768	32767	2
DINT	-2^{31}	$2^{31}-1$	4
LINT	-2^{63}	$2^{63}-1$	8
USINT	0	255	1
UINT	0	65535	2
UDINT	0	232–1	4
ULINT	0	264–1	8

Нижня границя діапазону цілих без знаку 0, верхня границя визначається як $(2^n) - 1$, де n – число розрядів числа.

Для чисел зі знаком нижня границя – (2^{n-1}), верхня границя $(2^{n-1}) - 1$.

Найменування цілих типів даних утворюються із застосуванням префіксів, що виражають відношення розміру до 16-розрядних слів: S (short *1/2) коротке, D (double *2) подвійне, L (long *4) довге. Префікс U (unsigned) вказує на представлення цілих без знаку.

Змінні типів BYTE, WORD, DWORD і LWORD визначаються стандартом як бітові рядки ANY_BIT. Говорити про діапазон значень чисел для цих змінних взагалі некоректно. Вони представляють рядки з 8, 16 і 32 біт, відповідно. Крім звернення до таких змінних як до єдиних цілих, їх можна використовувати побітно.

Цілі числа можуть бути представлені в двійковій, вісімковій, десятковій або шістнадцятковій системі числення. Числові константи, відмінні від десяткових, вимагають зазначення основи системи числення перед знаком «#». У таблиці 5.2 наведено приклад запису чисел в різних системах числення.

Таблиця 5.2 – Приклад запису чисел в різних системах числення

Система числення	Приклад запису чисел
двійкова	2#0100_1110
вісімкова	8#116
шістнадцяткова	16#4E
десяткова	78

Для позначення шістнадцяткових цифр від 10 до 15 використовуються латинські літери від A до F.

Символ підкреслення «_» не впливає на значення і використовується виключно для поліпшення зорового сприйняття числа. Наприклад: 10_000, 16#01_88. Підкреслення можна застосовувати тільки між цифрами або в кінці числа. Два або більш підкреслення поспіль застосовувати не можна.

За початкової ініціалізації цілочисельні змінні отримують нульові значення. Якщо необхідно задати інші початкові значення, це можна зробити безпосередньо з оголошенням змінної.

Приклади:

```
VAR
wVar0, wVar1:    WORD;      (*2 змінних типу WORD*)
byVar3 :         BYTE ;     (*тип BYTE початкове значення 0*)
byVar2:          BYTE := 16#55; (*тип BYTE початкове значення 55h*)
END_VAR
```

byVar2 := 2#1_0_0_0_1_0_0_0; (*рівнозначно 2#1000_1000*) byVar3 := 2#1_0_0_0__1_0_0_0; (*помилка*)

Логічний тип

Логічні змінні оголошуються ключовим словом BOOL. Це означає їх приналежність до алгебри Буля. Вони можуть приймати тільки значення логічного нуля FALSE (ХИБНІСТЬ) або логічної одиниці TRUE (ІСТИНА). За початкової ініціалізації логічне значення за замовчуванням – ХИБНІСТЬ.

Приклад:

```
VAR
bVarl:      BOOL := TRUE;
wVar2:      WORD;
END VAR
```

З перетворенням значення логічної змінної в цілу FALSE повертає 0, а TRUE 1:

```
wVar2 := BOOL_TO_WORD(bVarl);      (*результат 1 *)
```

Зі зворотним перетворенням будь-якого цілого в логічну змінну істину утворює будь-яке ненульове значення:

```
wVar2 := 0;
bVarl := WORD _TO_ BOOL (wVar2);      (*результат FALSE*)
```

Результати операцій, що дають логічне значення, можна присвоювати змінним типу BOOL:

```
bVarl := wVar2 > 5000;
```

За визначенням BOOL – це рядок з одного біта, але з міркувань ефективності коду під час автоматичного розподілу пам’яті транслятором під бітову змінну виділяється, як правило, 1 байт пам’яті повністю. Змінні типу BOOL, пов’язані з дискретними входами-виходами або визначені з прямою бітовою адресою, дійсно фізично представлені одним бітом.

Дійсні типи

Змінні дійсного типу REAL є дійсні числа в діапазоні $\pm 10^{\pm 38}$. З 32 біт, що займається числом, мантиса займає 23 біти. В результаті точність подання

приблизно становить 6-7 десяткових цифр. Довгий дійсний формат LREAL займає 64 біти. Число містить 52-бітову мантису. Точність подання приблизно становить 15-16 десяткових цифр. Діапазон чисел довгого дійсного $\pm 10^{\pm 307}$.

Числа з плаваючою комою записуються в форматі з точкою: 14.0, 0.33_, –120.2, або в експоненційній формі: –1.2E10, 3.1e7.

Інтервал часу

Змінні типу TIME використовуються для вираження інтервалів часу. На відміну від часу доби (TIME OF DAY) часовий інтервал не обмежений максимальним значенням у 24 години.

Числа, які виражають часовий інтервал, мають починатися з ключового слова TIME# або в скороченій формі T#. У загальному випадку подання часу складається з полів днів (d), годин (h), хвилин (т), секунд (s) і мілісекунд (ms). Порядок подання має бути саме такий, хоча непотрібні елементи можна опускати. Для кращого зорового сприйняття поля допускається розділяти символом підкреслення. Наприклад:

```
VAR
        TIME1:      TIME := t#10h_14m_5s;
END_VAR
```

Старший елемент може перевищувати верхню межу діапазону подання. Отже, якщо у поданні присутні дні або години, то секунди не можуть перевищувати значення 59. Якщо секунди стоять першими, то їх значення може бути й більшим.

```
TIME1 := t#1m65s;       (*помилка*)
TIME1 := T#125s;        (*правильно*)
```

Молодший елемент можна подати у вигляді десяткового дробу:

```
TIME1 := T#1.2S;        (*рівносильне T#1s200ms*)
```

Час доби і дата

Типи змінних, що виражають час дня чи дату, подаються згідно з ISO 8601.

У таблиці 5.3 подані типи змінних, що виражають час дня або дату.

Таблиця 5.3 – Типи змінних, що виражають час дня або дату

Тип	Коротке позначення	Початкове значення
DATE	D	1 січня 1970 р.
TIME_OF_DAY	TOD	00:00
DATE_AND_TIME	DT	00:00 1 січня 1970 р.

Дата записується у форматі «рік»-«місяць»-«число». Час записується у форматі «години»:«хвилини»:«секунди».«соті». Дата визначається ключовим словом DATE# (скорочено D#), час дня TIME_OF_DAY# (скорочено TOD#), дата і час DATE_AND_TIME# (скорочено DT#).

Приклад запису дати і часу:

DATE#2002-01-31 або D#2002-01-31

TIME_OF_DAY# 16:03:15.47 або TOD#16:03:15.47

DATE_AND_TIME#2002-01-31-16:03:15.47 або

DT#2002-01-31-16:03:15.47

Всі три типи даних фізично займають 4 байти (DWORD). Тип TOD містить час доби в мілісекундах, починаючи з 0 годин. Типи DATE і DT містять час у секундах, починаючи з 0 годин 1 січня 1970 року.

Рядки

Тип рядкових змінних STRING визначає змінні, що містять текстову інформацію. Розмір рядка задається з оголошенням. Наприклад, оголошення рядка str1, що вміщує до 20 символів, і str2 – до 60 символів:

```
VAR
    str1: STRING(20);
    str2: STRING(60) := ‘Протягування’;
END_VAR
```

Якщо початкове значення не задано, то в ході ініціалізації буде створений порожній рядок.

Кількість необхідної пам’яті визначається заданим з оголошенням розміром рядка. Для типу STRING кожен символ займає 1 байт (WSTRING 2 байти). Рядкові константи задаються між одинарних лапок:

```
str1 := ‘Включення насосу’;
```

За необхідності розміщення в рядок коду, що не має друкованого відображення, використовується знак ($) і наступний за ним код з двох цифр у шістнадцятковій системі числення. У таблиці 5.4 наведені позначення для поширених керуючих термінальних кодів.

Таблиця 5.4 – Позначення для термінальних кодів

Позначення	Код
$$	Знак долара
$’	Одиночна лапка
$L або $l	Перевід рядка
$N або $n	Новий рядок
$P або $p	Перевід сторінки
$R або $r	Розрив рядка
$T або $t	Табуляція

Ієрархія елементарних типів

Наведена в таблиці 5.5 ієрархія елементарних типів застосовується виключно для зручності опису програм. Кожне найменування ANY_... об’єднує деяку множину типів. Так, у ході опису будь-якої бітової операції зручніше вказати, що вона може бути застосовна для ANY_BIT, ніж перераховувати щоразу допустимі елементарні типи. Застосовувати ANY_ з оголошенням змінних не можна.

Таблиця 5.5 – Ієрархія елементарних типів

<table>
<tr><td rowspan="6">ANY</td><td rowspan="2">ANY_NUM</td><td>ANY_INT</td><td>SINT, INT, DINT, LINT, USINT, UINT, UDINT, ULINT</td></tr>
<tr><td>ANY_REAL</td><td>REAL, LREAL</td></tr>
<tr><td colspan="2">ANY_BIT</td><td>BOOL, BYTE, WORD DWORD, LWORD</td></tr>
<tr><td colspan="3">STRING</td></tr>
<tr><td colspan="3">TIME</td></tr>
<tr><td>ANY_DATE</td><td></td><td>DATE, TIME_OF_DAY, DATE_AND_TIME</td></tr>
</table>

5.1.3 Призначені для користувача типи даних

Опис призначених для користувача типів даних (крім масивів) має виконуватися на рівні проекту (в CODESYS на вкладці «Типи даних» =>

«Організатор об’єктів»). Оголошення типу завжди починається з ключового слова TYPE і закінчується рядком END_TYPE.

Масиви

Масиви – це множина однотипних елементів з довільним доступом. Масиви можуть бути багатовимірними. Розмірність масиву і діапазони індексів задаються з оголошенням. Приклад оголошення тривимірного масиву:

<ім’я масива>:ARRAY

[<li1>…<hi1>,<li2>…<hi2>,<li3>…<hi3>] OF <тип елемента>;

де li1, li2, li3 вказують нижні границі індексів; hi1, hi2 і hi3 – верхні границі.

Індекси мають бути цілого типу і тільки позитивні. Негативні індекси використовувати не можна.

Приклади оголошення масивів:

```
XYbass:    ARRAY [1. .10,1..20] OF INT;
TxtMsg:    ARRAY [0..10] OF STRING(32);
Massl:     ARRAY [1..6] OF SINT := 1,1,2,2,2,2;
Mass2:     ARRAY [1..6] OF SINT := 1,1,4(2);
```

Два нижніх приклади показують, як можна виконати ініціалізацію елементів масиву з оголошенням. Обидва приклади створюють однакові масиви. У першому прикладі всі початкові значення наведені через кому. У другому прикладі присутнє скорочення N(a,b,c..) яке означає – повторити послідовність a, b, c.. N раз. Багатовимірні масиви ініціалізуються порядково:

```
Mass2d:    ARRAY [1..2,1..4] OF SINT := 1,2,3,4,5,6,7,8;
```

Результат ініціалізації Mass2d показаний на рис. 5.1.

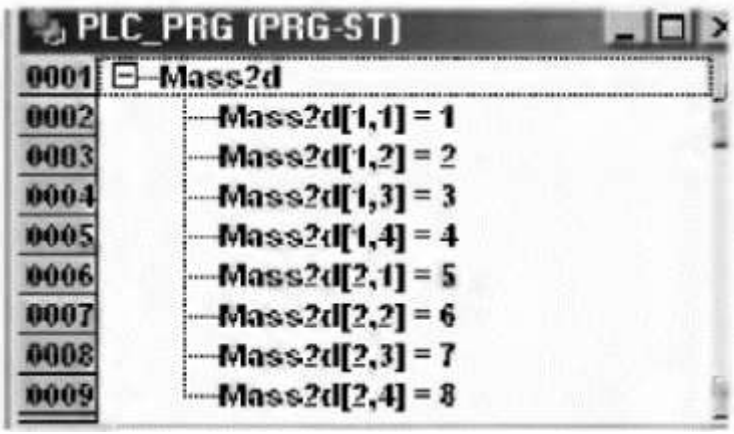


Рисунок 5.1 – Результат ініціалізації Mass2d

Для доступу до елементів масиву застосовується такий синтаксис:

<Ім’я_масива>[Індекс1, Індекс2, Індекс3]

Для двовимірного масиву використовуються два індекси. Для одновимірного, очевидно, достатньо одного. Наприклад:

```
XYbass[2,12] := 1;
i := STR_TO_INT(TxtMsg[4]);
```

Рекомендується використовувати в масивах нумерацію з нуля. У цьому випадку обчислення фізичної адреси елемента в ході виконання простіше. Внаслідок цього код буде коротшим.

Структури

Структури призначені для створення нових типів даних на основі елементів різних базових типів. Зі змінною типу структура можна поводитися як з єдиним елементом, передавати як параметр, створювати покажчики, копіювати тощо.

На відміну від масивів структура дійсно вводить новий тип даних. Це означає, що до застосування конкретної змінної потрібно виконати як мінімум два оголошення. Спочатку потрібно описати структуру.

Опис структури відбувається глобально, на рівні проекту. Описана структура отримує ідентифікатор (ім’я структури). Але це ще не змінна, це новий тип даних. Тепер, використовуючи новий ідентифікатор, потрібно оголосити одну або скільки завгодно змінних, так само, як і для базових типів. Тільки тепер змінна нового типу отримує «тілесну оболонку» або, іншими словами, конкретне місце в пам’яті даних.

Оголошення структури має починатися з ключового слова STRUCT і закінчуватися END_STRUCT. Синтаксис оголошення виглядає так:

```
TYPE <Ім’я _структури>:
    STRUCT
        < Оголошення змінної 1>
        < Оголошення змінної n>
    END_STRUCT
END_TYPE
```

Приклад оголошення структури на ім’я Trolley:

```
TYPE Trolley:
    STRUCT
        Start:      TIME;
        Distance:   INT;
        Load, On:   BOOL$
        Articl:     STRING(16);
    END_STRUCT
END_TYPE
```

Оголошення в програмі змінної Telega1 типу Trolley і початкова ініціалізація структури виглядає так:

Telegal: Trolley := (Articl:=‘Порожній’);

Стан елементів після початкової ініціалізації Telega1 показано на рис. 5.2.

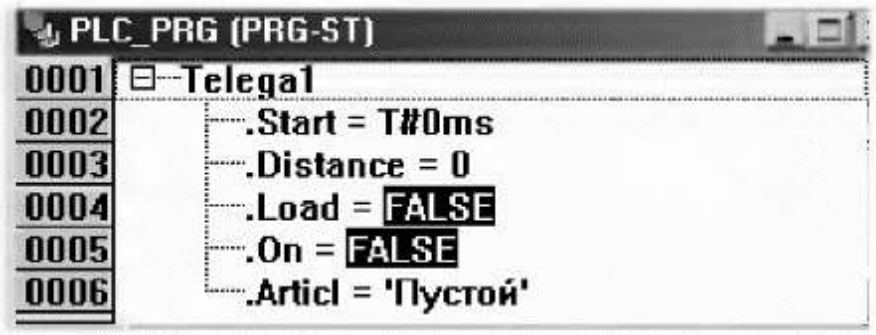


Рисунок 5.2 – Стан елементів після початкової ініціалізації Telega1

За початкової ініціалізації не обов’язково ставити значення для всіх елементів. Елементи, які не мають чітко зазначених початкових значень, за замовчуванням отримають нульові значення.

Для доступу до елементів структури використовується такий синтаксис:

<Ім’я_змінної>.<Ім’я_елемента>

Наприклад:

Telegal.On := True;

Структури можуть включати інші структури, масиви й самі утворювати масиви. Приклад оголошення та ініціалізації масиву структур:

TrolleySet: ARRAY[0..2] OF Trolley := (Articl := ‘T 1’),

(Articl := ‘T 2’), (Articl := ‘T 3’);

TrolleySet[i].On := TRUE;

Якщо структура містить вкладену структуру, то доступ до елементів вкладеної структури здійснюється із застосуванням складного імені, що містить дві точки:

train.wagon[5]. weight; (*wagon[] вкладений масив структур*)

Оскільки фізичний розмір елементів структури відомий транслятору заздалегідь, звернення до елемента структури не дає жодних накладних витрат порівняно з простою змінною. Транслятор має можливість розрахувати абсолютні адреси елементів у ході компіляції. Це не стосується масивів структур. Щоб не мати проблем у використанні декількох різних змінних однієї структури, застосовувати прямі адреси в структурі не можна.

Перерахування

Перерахування дозволяє визначити кілька послідовних значень змінної і присвоїти їм найменування. Перерахування – це зручний інструмент, що дозволяє обмежити множину значень змінної і посилити контроль під час трансляції. Як і структура, перерахування створює новий тип даних, визначення якого виконується на рівні проекту:

TYPE <Ім’я перерахувань>:

(<Елемент 0>, <Елемент 1>, ... <Елемент n>);

ENDTYPE

Оголошена пізніше змінна типу <Ім’я перерахування> може приймати тільки перераховані значення. В ході ініціалізації змінна отримує перше зі списку значення. Якщо числові значення елементів перерахування не вказані явно, їм присвоюються послідовно зростаючі числа, починаючи з 0. Фактично елемент перерахування – це число типу INT з обмеженим набором значень. Якщо необхідно, значення елементам можна присвоїти явно під час оголошення типу перерахування.

Наприклад:

TYPE TEMPO: (Adagio := 1, Andante := 2, Allegro := 4);

END_TYPE

Ідентифікатори елементів перерахування використовуються в програмі як значення змінної:

VAR

LiftTemp : TEMPO := Allegro;

END VAR

Якщо в різні перерахування включені елементи з однаковими іменами, виникає неоднозначність. Для вирішення цієї проблеми застосовується префікс, що містить перерахування: TEMPO#Adagio. У CODESYS всі найменування елементів перерахування мають бути унікальними.

Обмеження діапазону

Тип змінних з обмеженим діапазоном значень дозволяє визначити допустиму множину значень змінної. Оголошення типу змінної з обмеженим діапазоном має відбуватися безпосередньо між ключовими словами TYPE і END_TYPE:

```
TYPE <Ім'я> :
    <Цілий тип> (<від>..<до>)
END_TYPE
```

Наприклад:

```
TYPE DAC10:
    INT (0..16#3FF);
ENDTYPE
```

Застосування змінної з обмеженням діапазону покажемо на прикладі:

```
VAR
    dac: DAC10;
END_VAR
dac := 2000;
```

Зі спробою трансляції даного прикладу виникає помилка:

Error: Type mismatch: Cannot convert '2000' to 'INT(0..1023)'.

Псевдоніми типів

Проблема вибору відповідного типу даних не завжди вирішується легко. Припустимо, ви працюєте з температурою, заміряною 16-розрядним АЦП. Чи може бути температура тільки вище нуля або коли-небудь буде потрібно працювати в негативній області – ще не зовсім очевидно. В одному випадку потрібно використовувати тип змінних UINT, а в іншому – INT. Тут зручно визначити новий тип даних:

```
TYPE TEMPERATURA : UINT;
END TYPE
```

Далі скрізь у програмі використовується тип TEMPERATURA під час оголошення змінних. Якщо раптом знадобиться змінити тип температури на INT, то це легко і швидко можна буде зробити в одному місці.

Аналогічні псевдоніми типів зручно створювати для будь-яких часто використовуваних у програмі типів. Наприклад, для масивів або інших типів, що мають довге і невиразне визначення.

Специфіка реалізації типів даних CODESYS

Стандарт тільки визначає сумісні типи даних, але не вимагає обов'язкової підтримки усіх типів для всіх реалізацій систем МЕК-програмування. CODESYS має найбільш повну підтримку стандартних типів. Але навіть він у версії 2.x не підтримував 64-розрядні цілі і текстові рядки Unicode.

Крім того, обмеження підтримуваних типів даних можливо навіть у рамках одного комплексу програмування для різних контролерів. Так, восьмизарядний генератор коду CODESYS не підтримує дійсні змінні, перерахування та змінні, які виражають час доби і календарну дату. Обмеження підтримки типів обґрунтовується досягненням мінімальної вартості за максимальної ефективності ПЛК різних категорій. Так, повна реалізація ядра системи виконання CODESYS (включаючи налагодження функції і трасування значень змінних) для Intel 8051 сумісного мікроконтролера вимагає загалом 6 Кб пам'яті коду. Тому і код прикладної програми має бути максимально компактним, для чого доводиться йти на певні компроміси. Але за необхідності будь-які спеціалізовані типи даних можна визначити на базі елементарних типів і підтримати за допомогою бібліотек.

У CODESYS немає обмеження на спосіб застосування бітових рядків. Внаслідок цього типи BYTE, WORD і DWORD можна застосовувати в операціях, що вимагають цілих без знака (USINT, UINT і UDINT), але не навпаки.

Внутрішній формат змінних типу TIME стандартом не обмежений. У CODESYS інтервали часу зберігаються у змінній типу DWORD в мілісекундах, що забезпечує надання інтервалів майже 50 діб. Якщо довжина рядка STRING з оголошенням не зазначена, то приймається значення за замовчуванням – 80 символів. У CODESYS використовуються нультерміновані (як у С-компіляторах) рядки. Тобто під рядок завжди заздалегідь виділяється область пам'яті заданого максимального розміру.

Будь-який рядок закінчується нульовим байтом, який не входить до складу рядка, а слугує виключно для визначення кінця рядка функціями, які оперують із рядками. Порожній рядок складається з єдиного нульового байта. З оголошенням рядка необхідно ставити розмір на одиницю більше необхідного для символу «кінець рядка». Така форма подання найбільш компактна (лише 1 допоміжний байт), але, очевидно, не оптимальна в плані швидкодії. Якщо, наприклад, потрібно об'єднати два рядки, то функція конкатенації рядків зобов'язана спочатку знайти, де закінчується перший рядок. В інших системах програмування можна зустріти реалізацію рядків у вигляді структури, що містить максимальний розмір, поточну довжину рядка і сам рядок (масив байт). Взагалі ж робота із рядками в ПЛК потребується не часто.

Для підтримки перевірки значень змінних з обмеженим діапазоном під час роботи система виконання має надавати засоби контролю. У CODESYS це завдання вирішується так – дії, які виконуються зі спробою виходу за діапазон, визначаються програмістом. Для цього призначені спеціальні функції (CheckRangeSigned, CheckRangeUnsigned), які необхідно включити до проекту. На вході функції отримують три параметра: дві границі діапазону і значення. Будь-яка необхідна реакція на порушення границь (обмеження змінної, індикація помилки тощо) описується в тілі функцій контролю.

5.1.4. Змінні

Кожна змінна обов'язково має найменування й тип. Сутність змінної може бути різною. Змінна може представляти вхід або вихід ПЛК, дані в оперативній або незалежній пам'яті. Далі ми розглянемо правила оголошення та деякі практичні складнощі й тонкощі, які виникають під час роботи зі змінними.

Ідентифікатори

Ім'я змінної (її ідентифікатор) має бути складено з друкованих символів і цифр. Цифру не можна ставити на перше місце. Пробіли в найменуванні використовувати не можна. Замість них зазвичай застосовується символ підкреслення. Символ підкреслення є значущим. Так імена 'Var1', 'Var_l' і '_Varl' є різними. Два підкреслення поспіль використовувати не можна. Регістр літер не враховується. Так 'VAR1' і 'Var1' – одне й те саме. Як мінімум, шість перших знаків ідентифікатора є значущими для всіх систем програмування.

У CODESYS такого обмеження немає – всі символи найменування є значущими. Символи кирилиці в ідентифікаторах застосовувати не можна. Це обмеження характерне для всіх програмних систем.

Аналогічні вимоги стосуються і будь-яких ідентифікаторів МЕК-програм (компоненти, мітки, типи тощо).

Розподіл пам'яті змінних

Контролер з точки зору МЕК програми має кілька областей пам'яті, що мають різне призначення:

– область входів ПЛК;

– область виходів ПЛК;

– область пам'яті з прямою адресацією;

– оперативна пам'ять користувача (ОЗП).

Апаратні ресурси ПЛК присутні в МЕК-проектах у неявній формі. Розміщення змінної в одній з трьох перших областей призводить до її зв'язку з певною апаратурою – входами, виходами або змінними системи виконання (діагностика модулів, настроювання параметрів ядра тощо). Розподіл змінних у цих областях визначається виробником ПЛК. Прив'язка до конкретних адрес задається за допомогою прямої адресації.

Для забезпечення переносимості програмного забезпечення прямі адреси потрібно використовувати тільки в розділі оголошень. У мовах програмування стандарту не передбачено операцій прямого читання входів-виходів. Цю роботу виконує система виконання. За необхідності для низькорівневого звернення виробником ПЛК поставляються спеціальні бібліотеки.

Оголошення змінної без префікса AT фізично означає виділення їй певної пам'яті в області ОЗП. Розподіл доступної пам'яті ОЗП транслятор здійснює автоматично.

Змінні прийнято розділяти на глобальні та локальні за областю видимості. Глобальні змінні визначаються на рівні ресурсів проекту (VAR_GLOBAL) і доступні для всіх програмних компонентів проекту. Локальні змінні описуються з оголошенням компонента і доступні тільки всередині нього.

Опис будь-якого програмного компонента містить, як мінімум, один розділ оголошення локальних змінних VAR, змінних інтерфейсу VAR_INPUT, VAR_OUTPUT, VAR_IN_OUT і зовнішніх глобальних змінних VAR_EXTERNAL.

Найменування розділів оголошення змінних можуть містити додаткові ключові слова, уточнюючи спосіб застосування, як показано в таблиці 5.6.

Таблиця 5.6 – Ключові слова для уточнення способу застосування змінної

Ключове слово	Застосування змінної
RETAIN	Змінні потрібно розмістити в незалежній пам’яті, що зберігає значення з вимкненим живленням. Така пам’ять не є обов’язковою і є не у всіх ПЛК
CONSTANT	Константи, доступні тільки для читання

Пряма адресація

Для створення змінної, що прямо адресується, використовується таке оголошення:

ім’я змінної AT %пряма адреса тип;

Пряма адреса починається з літери, яка визначає область пам’яті, як показано в таблиці 5.7.

Таблиця 5.7 – Визначення області пам’яті

Символ	Область пам’яті
I	Область входів
Q	Область виходів
M	Пам’ять, що прямо адресується

Далі йде символ, який визначає тип прямої адреси. У таблиці 5.8 наведено повний список символів, що визначає тип прямої адреси.

Таблиця 5.8 – Повний список символів, що визначає тип прямої адреси

Символ	Область пам’яті
ні	Біт
X	Біт
B	Байт
W	Слово
D	Подвійне слово
L	Довге слово

Завершує пряму адресу число – складена ієрархічна адреса, поля якої розділені крапкою. У найпростішому випадку використовується два поля адреси: номер елемента і номер біта. На рис. 5.3 наведена схема побудови імені змінної, що прямо адресується.

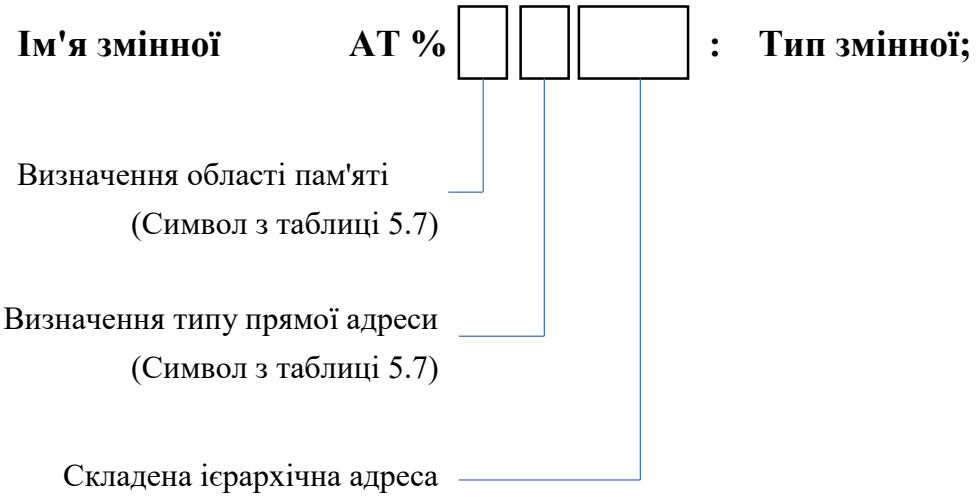


Рисунок 5.3 – Схема побудови імені змінної, що прямо адресується

В кінці оголошення, як і для змінних, що автоматично розміщуються, необхідно вказати тип змінної. Із зазначенням адреси одного біта тип змінної може бути тільки BOOL.

У прямій адресі вказується саме номер елемента. Це докорінно відрізняється від фізичних адрес мікропроцесора. Якщо пряму адресу визначає байт, то номер елемента – це номер байта. Якщо пряму адресу визначає слово, то номер елемента – це номер слова, і, відповідно, один елемент займає два байти. Так, наступні три записи адресують один і той самий байт:

```
dwHeat    AT %MD1:    BYTE;
wbHeat    AT %MW2:    BYTE;
byHeat    AT %MB4:    BYTE;
```

Нумерацію елементів пам’яті для даного прикладу ілюструє таблиця 5.9.

Таблиця 5.9 – Приклад розподілу змінних у пам’яті

D	0				**1**			
W	0		1		**2**		3	
B	0	1	2	3	**4**	5	6	7

На випадок, якщо така схема виявиться чомусь неприйнятною, компілятор CODESYS має спеціальний прапорець, який примусово вмикає байтову адресацію для всіх типів.

У кожній області пам’яті адресація елементів починається з нуля. Фізичне розміщення областей, які прямо адресуються в ОЗП, визначається конфігурацією контролера.

Очевидно, що зіставлення ідентифікаторів змінних прямим адресам є справою, що вимагає великої акуратності. Тому для складних модульних контролерів застосовуються спеціальні фірмові конфігуратори, що підключаються до оболонки комплексу програмування і дозволяють графічно «зібрати» ПЛК і визначити всі необхідні інтерфейсні змінні.

Входи ПЛК – це змінні з прямими адресами в області I. Вони доступні в прикладних програмах тільки за читанням. Виходи Q – тільки за записом. Змінні в області M доступні за записом і читанням.

В області пам’яті M розміщують зазвичай змінні, які не можна однозначно віднести до входів або виходів. Це можуть бути діагностичні ресурси модулів, параметри системи виконання тощо.

Прямі адреси можна використовувати в програмах безпосередньо, наприклад:

```
IF %IW4 > 1 THEN ...    (*Значення входу IW4*)
```

Проте все ж бажано компактно зосередити в проекті всі апаратно-залежні моменти.

Зверніть увагу, що пряма адресація дозволяє розмістити кілька різнотипних змінних в одній і тій самій пам’яті. Наприклад, спеціально для швидкого обнуління 16-дискретних виходів (BOOL) можна використовувати змінну типу WORD. Або, наприклад, поєднати змінну STRING і декілька змінних типу BYTE, що дасть можливість організувати форматування виводу без застосування рядкових функцій. Оскільки фізичний розподіл адрес відомий на етапі трансляції, компілятор формує максимально компактний код для таких об’єднань, чого не вдається досягти у роботі з елементами масиву, де потрібна динамічна адресація.

Порозрядна адресація

У стандарті передбачена зручна форма роботи з окремими бітами змінних типу бітових рядків – порозрядна адресація. Необхідний біт вказується через точку після ідентифікатора. Аналогічно можна використовувати окремі біти пам’яті, що прямо адресується. Молодшому біту відповідає нульовий номер. Порозрядна нумерація не має перевищувати межі відповідного типу числа.

```
VAR
        a: WORD;
        bStop AT %IX64.3:       BOOL;
ENDVAR
a := 0;
a.3 := 1;               (*або a.3 := TRUE; – результат 2#0000_1000*)
a.18 := TRUE;           (*помилка, в WORD не може бути біт a.18*)
IF a.15 THEN ...        (*а менше нуля?*)
```

Перетворення типів

Перетворення типів відбувається з присвоєнням значення змінної одного типу змінній іншого типу. Перетворення змінює фізичне подання значення у пам'яті даних, але не має змінювати саме значення. Якщо це неможливо, то перетворення призводить до часткової втрати даних. Але в такому випадку транслятор вимагає явної вказівки необхідності виконання такої операції.

Розглянемо спочатку роботу з цілими числами. Нехай, наприклад, оголошена змінна siVar типу – коротке ціле (SINT 8 біт) і змінна iVar типу – ціле (INT 16 біт). Припустимо siVar = 100 а iVar = 1000. Вираз iVar := siVar є цілком допустимим, оскільки числа типу SINT є підмножиною INT (iVar прийме значення 100). Тут перетворення типу буде виконано транслятором автоматично, без будь-яких додаткових вказівок. Зворотне присвоювання siVar := iVar призведе до переповнення і втрати даних. Змусити транслятор виконати перетворення з імовірною втратою даних можна тільки в явній формі за допомогою спеціального оператора siVar := INT_TO_SINT(iVar). Результат дорівнює 24 (в шістнадцятковій формі 1000 це 16#03E8 і тільки молодший його байт перейде в SINT, значення 16#E8 відповідає десятковому числу 24).

Аналогічна ситуація виникає під час роботи з дійсними числами довгого LREAL і короткого REAL типів.

Оператори явного перетворення базових МЕК-типів утворюють свої найменування з двох частин. Спочатку вказується «вихідний тип», потім «_TO_» і «тип результату». Наприклад:

```
si := INT_TO_SINT(16#55AA);       (* Результат 16#AA*)
si := TIME_TO_SINT(T# 120ms);     (*120*)
i := REAL_TO_I NT(2.7);           (*Результат 3*)
```

```
i := TRUNC(2.7);                    (*Результат 2*)
t := STRING_TO_TIME('T#216ms'); (*Результат T#116ms*)
```

Операція TRUNC виконує відкидання дробової частини на відміну від перетворення REAL_TO_INT, що виконує округлення.

Зверніть увагу, що операції перетворення допустимі для будь-яких комбінацій базових типів, а не тільки для сумісних типів (наприклад, дату в рядок). Так, перетворення <...> _ TO_STRING фактично замінюють оператор PRINT, поширений у мовах загального застосування.

У конкретній реалізації окремі перетворення можуть не підтримуватися або мати певні особливості, насамперед це стосується перетворень рядків в інші типи і назад.

Формат BCD

Двійково-кодований десятковий формат подання BCD (binary coded decimal) являє собою числа в позиційній десятковій системі, де кожна цифра числа займає 4 біти. Наприклад, десяткове число 81 буде подано у вигляді 2#1000_0001. Арифметичні операції з BCD-числами вимагають застосування спеціального математичного апарату, малоефективні порівняно зі звичайним двійковим поданням. Але, з іншого боку, BCD виявляється дуже зручним у процесі організації клавіатурного вводу та індикації. Наприклад, функції виведення числа на принтер або навіть на сегментний індикатор виходять тривіальними (одна одновимірна таблиця на 10 констант).

Для зберігання чисел у форматі BCD стандарт МЕК пропонує використовувати змінні типів ANY BIT (крім BOOL, звичайно). Арифметика BCD-обчислень зазвичай не підтримується у стандартному комплекті бібліотек систем програмування ПЛК. У бібліотеці утиліт CODESYS реалізовані дві прості функції BCD перетворення: BCD TO INT і INT TO BCD.

5.1.5 Визначення компонента

Компоненти організації програм є базовими елементами, з яких будується код проекту. Аналогічно електронні пристрої складаються зазвичай з модулів. Кожен компонент програми має власне найменування, певний інтерфейс і опис однією з МЕК-мов.

Один компонент може викликати інші компоненти. Виклик самого себе (рекурсія) у стандарті МЕК не дозволена. Комбінувати різні мови в одному проекті можна з описом різних компонентів, але окремий компонент цілком реалізується однією мовою МЕК. Під час виклику компонента мова його реалізації значення не має.

До компонентів організації програм у МЕК-стандарті належать функції, функціональні блоки і програми. Всі вони багато в чому схожі, але мають певні особливості й різне призначення.

Компонент має властивість інкапсуляції – працює як «чорний ящик», приховуючи деталі реалізації. Для роботи з компонентом достатньо знати його інтерфейс, що включає опис входів і виходів. Внутрішню його будову знати необов'язково. У графічній формі подання компонент виглядає як прямокутник з входами зліва і виходами справа. Локальні (внутрішні) змінні компонента недоступні ззовні і в графічному поданні не відображаються.

Завдяки інкапсуляції компоненти успішно вирішують завдання структурної декомпозиції проекту. На верхньому рівні подання ми працюємо з більшими компонентами. Кожен з них виконує значну для даного проекту задачу. Зайві подробиці на цьому рівні тільки заважають розумінню проблеми. Розкриваючи вкладені компоненти один за одним, ми можемо отримати найдетальніше подання.

Готовий компонент завжди можна розкрити, вивчити й поправити. Це стосується тільки призначених для користувача компонентів і відкритих бібліотек. Деякі стандартні компоненти включені в транслятор і не доступні для перегляду і зміни. Це стосується і зовнішніх бібліотек. Зовнішні бібліотеки реалізуються у вигляді об'єктного коду за допомогою зовнішніх засобів, наприклад компілятор С або асемблера. Можливо навіть, що компонент реалізований не тільки програмно, а використовує допоміжні апаратні засоби, наприклад годинник реального часу або математичний співпроцесор.

Ще одним завданням, що вирішується компонентами, є локалізація імен змінних. Це означає, що в різних компонентах можна використовувати повторювані імена. Так, наприклад, улюблену змінну з оригінальним ідентифікатором «X» можна використовувати в кожному компоненті, і щоразу це буде нова змінна. Область видимості локальних змінних визначається рамками одного компонента. Обмеження області видимості є обов'язковим у всіх сучасних системах програмування.

Екземпляри функціональних блоків, оголошені всередині інших компонентів, також мають локальну область видимості. Програми та функції завжди визначені глобально.

Оголошення POU

Реалізації будь-якого POU завжди має передувати розділ оголошень. Оголошення функції, функціонального блоку і програми починаються відповідно з ключових слів FUNCTION, FUNCTION_BLOCK і PROGRAM. За ним йде ідентифікатор (ім’я компонента). Далі визначається інтерфейс POU. До інтерфейсу компонента належать входи VAR_INPUT, виходи VAR_OUTPUT і змінні типу вхід-вихід VAR_IN_OUT. Завершують розділ оголошень локальні змінні VAR.

У функціях розділи VAR_OUTPUT і VAR_IN_OUT відсутні. Виходом функції є єдина змінна, що збігається з ім’ям функції. Тип значення, що повертається, вказується під час визначення ідентифікатора через двокрапку. Наприклад: FUNCTION iNearby : INT.

Структура розділу оголошень POU показана в таблиці 5.10.

Таблиця 5.10 – Структура розділу оголошень POU

Тип POU	Функція	Функціональний блок	Програма
	FUNCTION ім’я: ТИП	FUNCTION_BLOCK ім’я	PROGRAM ім’я
Інтерфейс	VARINPUT	VAR_INPUT	VAR_INPUT
	–	VAR_OUTPUT	VAR_OUTPUT
	–	VAR_IN_OUT	V AR_IN_OUT
Локальні змінні	VAR	VAR	VAR

Всі розділи змінних є не обов’язковими.

Формальні й актуальні параметри

Інтерфейс компонента утворюється вхідними та вихідними змінними. Інтерфейсні вхідні змінні називають формальними параметрами. З використанням компонента його формальні параметри зв’язуються з

актуальними параметрами. І нарешті, під час виклику параметри компонента набувають актуальних або поточних значень. Ці поняття необхідні для усунення двозначності під час опису техніки роботи з компонентами.

Пояснимо їх відмінності на прикладі. Візьмемо стандартний блок R_TRIG. Він має вхід з назвою CLK. Ми використовуватимемо його в програмі, в якій визначена певна підходяща змінна, наприклад bPulse. Під час виклику блоку з нашої програми ми подаємо bPulse на вхід CLK. Далі програма компілюється і завантажується в контролер. Змінна bPulse набуває деякого значення, наприклад TRUE. Вхід CLK, відповідно, теж матиме значення TRUE. Тут відмінності вже практично очевидні. CLK – це формальний параметр, bPulse – актуальний параметр, а TRUE – фактичне значення. З формальними параметрами доводиться мати справу в ході проектування POU й опису його інтерфейсу. Актуальні параметри працюють під час використання компонента. Поточні значення народжуються тільки в «залізі» у процесі виконання.

Параметри і змінні компоненти

Під час оголошення POU можна зустріти такі заголовки:

Формальні вхідні параметри VAR_INPUT

Передаються POU за значенням шляхом копіювання. Під час виклику блоку такій змінній можна привласнити значення іншої змінної (сумісного типу), константи або виразу. Будь-які зміни такої змінної всередині POU жодним чином не відображаються на дані компонента, який викликає. Застосовується в будь-яких POU. Можуть мати значення за замовчуванням. Відображаються в графічному поданні з лівого боку компонента.

Формальні вихідні параметри VARJDUTPUT

Відображають результати роботи компонента. Передаються з POU за значенням шляхом копіювання. Читання значення виходів зазвичай має сенс після виконання блоку. Поза компонента параметри VAR_OUTPUT доступні тільки за читанням. Не використовуються у функціях, оскільки функція має тільки одне значення, що повертається. Можуть мати початкові значення. Відображаються у графічному поданні справа.

Параметр типу VAR_IN_OUT

Цей параметр водночас є входом і виходом. Передача змінної екземпляру блоку виконується за посиланням. Це означає, що зовнішня змінна немовби сама працює всередині блоку на правах внутрішньої змінної. До компонента

передається тільки адреса її розташування в пам’яті даних. Для змінної VAR_IN_OUT не можна:

– використовувати її у функціях;

– привласнювати початкове значення;

– звертатися як до елемента структури даних, через точку;

– привласнювати константу як актуальний параметр.

Присвоєння зовнішньої змінної для VAR_IN_OUT можна робити тільки з викликом блоку.

Найважливішою властивістю VAR_IN_OUT є відсутність копіювання зовнішніх даних. Параметри VAR_INPUT і VAR_OUTPUT можуть оперувати з масивами і структурами, але щоразу зі зверненням до компонента відбуватиметься повне копіювання даних. Це може забирати багато часу. Присвоєння одного масиву іншому для VAR_IN_OUT означає фактично переключення компонента з одного масиву на інший. Локальна копія даних у цьому випадку не створюється.

Як і глобальні змінні, параметри VAR_IN_OUT порушують ідеологію незалежності компонентів. Правильний компонент не повинен мати можливості зіпсувати чужу пам’ять. Тому застосовувати їх потрібно дуже акуратно й тільки у випадках, коли це дійсно необхідно.

Локальні змінні VAR

Доступні тільки всередині компонента, поза компонента доступу немає. Можуть мати початкові значення. Для функцій локальні змінні розміщуються в динамічній пам’яті (зазвичай у стеку). Після закінчення роботи функції пам’ять звільняється і може використовуватися в інших функціях. У програмах і екземплярах функціональних блоків змінні VAR зберігають свої значення між викликами програм і екземплярів. У графічному поданні компонента локальні змінні не відображаються.

5.1.6 Функції

Функція – це програмний компонент, що відображає множину значень вхідних параметрів на вихід. Функція завжди повертає тільки одне значення. З оголошенням функції вказується тип значення, ім’я функції і список вхідних параметрів. Виклик функції проводиться за ім’ям із зазначенням вхідних параметрів. Функція може використовуватися в математичних виразах поряд з операторами і змінними. Функція не має внутрішньої пам’яті. Це означає, що

функція з одними й тими самими значеннями вхідних параметрів завжди повертає одне й те саме значення. Функція – це чистий код. Багаторазове використання функції не призводить до повторного вмикання коду функції в ході компонування. Реалізація функції присутня в коді проекту тільки один раз. Щоразу з виконанням функції процесор виконує той самий поіменований код. Функція може мати локальні (тимчасові) змінні. Але по закінченні своєї роботи функція звільняє локальну пам'ять.

Тип функції (тип значення) може бути будь-яким з числа стандартних типів даних або типів, створених користувачем. Тіло функції може бути описано мовами IL, ST, LD або FBD. Використовувати в ГС не можна. З функції можна викликати бібліотечні функції та інші функції поточного проекту. Викликати функціональні блоки і програми з функцій не можна.

Виклик функції з перерахуванням значень параметрів

У прабатьків мови ST – мовах Паскаль і С виклик функції проводиться за ім'ям з перерахуванням в дужках списку актуальних вхідних параметрів, через кому, зліва направо. Аналогічний спосіб прийнятний і в мові ST. Наприклад:

Y := MUX(0, x1, x2); (*Повертає нульовий вхід – X_1*)

Тут необхідно звернути увагу на те, що найменування параметрів не важливі. З перерахуванням параметрів важливо тільки дотримуватися правильної послідовності відповідно до визначення в оголошенні функції. У графічних мовах порядок вхідних параметрів заданий напрямком зверху донизу (рис. 5.4).

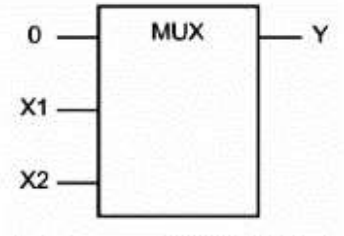


Рисунок 5.4 – Графічне відображення виклику функції

Деякі функції можуть мати рівноцінні параметри, тоді їх порядок очевидно неважливий. Наприклад:

```
y := MAX(x1, x2);        (*Повертає найбільше зі значень
                          вхідних параметрів *)
y := MAX(x2, x1);        (*Результат такий самий*)
```

Присвоєння значень параметрам функції

Другий спосіб виклику функції передбачає безпосереднє присвоювання значень параметрам функції за іменами:

stResult := CONCAT(STR1 := ‘Висока’, STR2 := ‘температура’);

Це те саме:

stResult:= CONCAT(‘Висока’, ‘температура’);

або:

stResult := CONCAT(‘Висока’, STR2 := ‘температура’);

Якщо у програмі вже визначена змінна з ім’ям, що збігається з найменуванням вхідного параметра (STR1 := ‘Висока’;), то такий запис може викликати нерозуміння:

stResult:= CONCAT(STR1 := STR1, STR2 := ‘температура’);

Насправді тут все правильно: зліва від знака присвоювання – параметр функції, праворуч – змінна.

Описаний спосіб виклику функції передбачає можливість задавати параметри в довільному порядку і опускати деякі з них. Версія CODESYS 2.x не забезпечує функції таку можливість. Єдиний сенс такої нотації – в універсальності, прийнятній для функціональних блоків і програм.

Передача параметрів функції завжди відбувається шляхом копіювання. За будь-якого способу виклику функція отримує локальні копії значень змінних.

Функції зі змінною кількістю параметрів

Для багатьох функцій важко передбачити, скільки значень потрібно буде обробити в конкретному випадку. Наприклад, для функції AND можна обмежитися двома входами і використовувати «драбинку» викликів функцій для обробки більшої кількості змінних (рис. 5.5).

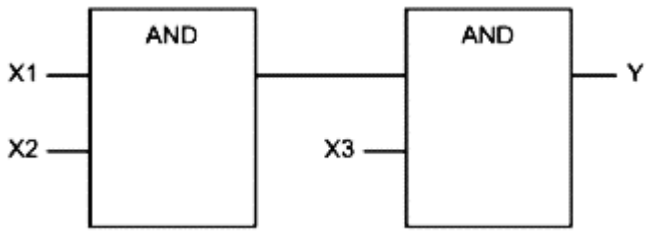


Рисунок 5.5 – З’єднання двовходових AND

На рис. 5.5 подана не дуже практична конструкція. Було б значно зручніше мати функцію, яка могла б «розширюватися» і адаптуватися під змінну кількість параметрів. Така реалізація показана на рис. 5.6.

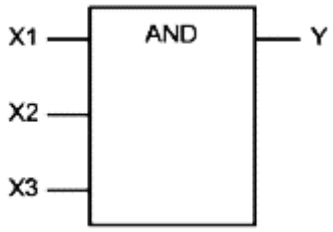


Рисунок 5.6 – Розширювана реалізація AND

Стандарт МЕК дійсно передбачає таку можливість. У текстових мовах розширення проводиться додаванням змінних у кінець списку параметрів:

y := MUX(x_n, x1, x2, x3, x4, x5);

Через складність реалізації транслятора змінна кількість параметрів у призначених для користувача функціях не використовується.

Оператори і функції

Оператори – це символи певних операцій. Але їх можна визначити і як функції, наділені певними привілеями. По-перше, код для операторів транслятор створює сам і не вимагає підключення будь-яких бібліотек. По-друге, багато операторів мають особливі форми запису у виразах ST. Наприклад, математичні оператори (додавання, віднімання, множення і ділення) мають традиційне символьне подання, у текстових мовах (+, – , *, /). У графічних мовах оператори виглядають як звичайні функції.

Можна обходитися без символьного подання операторів. Наприклад:

Y := SUB(MUL(4,x),3);

Але символьне подання у ST виглядає значно краще:

Y := 4 * X – 3;

Можна записати ще коротше:

Y = 4x – 3.

Всі три записи рівноцінні за змістом. Символьні вирази зрозуміліші і дають змогу більше сконцентруватися на суті виразу, а не на формі його подання.

Під час роботи з операторами необхідно звертати увагу на наявність символьної форми подання. Так, для математичних і логічних операторів у мові ST, як правило, допускається тільки символьне подання. Вираз

```
Y := AND(x1, x2)
```

викличе помилку компіляції. Необхідно писати так:

```
Y := x1 AND x2;
```

Якщо оператор не має символьного подання, то на нього поширюються звичайні правила виклику функцій. Наприклад:

```
y := SQRT(x);
```

Зверніть увагу, що імена вхідних параметрів для операторів в описі не задані. Це означає, що викликати такі функції в ST можна тільки перерахуванням параметрів.

5.1.7 Функціональний блок

Функціональний блок – програмний компонент, що відображає множину значень вхідних параметрів на множину вихідних. Після виконання екземпляру функціонального блоку всі його змінні зберігаються до наступного виконання. Отже, функціональний блок, що викликається з одними й тими самими вхідними параметрами, може виробляти різні вихідні значення. Зберігаються всі змінні, включаючи вхідні та вихідні. Так, якщо ми викличемо екземпляр функціонального блоку, не визначаючи значення деяких вхідних параметрів, він використовуватиме раніше встановлені значення. Можливість задання змінної кількості вхідних значень закладена за визначенням і не вимагає будь-яких додаткових зусиль. Ззовні доступні тільки входи й виходи функціонального блоку, отримати доступ до внутрішніх змінних блоку не можна.

З позицій об’єктно-орієнтованого програмування (ООП) функціональні блоки – це об’єкти, які добре реалізують інкапсуляцію, тобто приховування деталей реалізації. Об’єднання коду і даних в одному модулі споріднює функціональні блоки з класами ООП. Можливість успадкування і поліморфізм, на жаль, поки відсутні.

Створення екземпляра функціонального блоку

Перш ніж використовувати функціональний блок, необхідно створити його екземпляр. Ця операція аналогічна за змістом оголошенню змінної.

Описавши новий блок, ми фактично створили новий тип даних, подібний структурі. Кожен функціональний блок може мати будь-яку кількість екземплярів. Так, різні екземпляри блоку «таймер» абсолютно незалежні один від одного. Кожен з них має власні настройки і живе власним життям.

Кожен екземпляр функціонального блоку має свій власний ідентифікатор і свою область у статичній пам’яті даних. Оголошення ще одного екземпляру блоку призводить до виділення ще однієї області в пам’яті даних. Код, очевидно, як і для функції, залишається спільним (рис. 5.7).

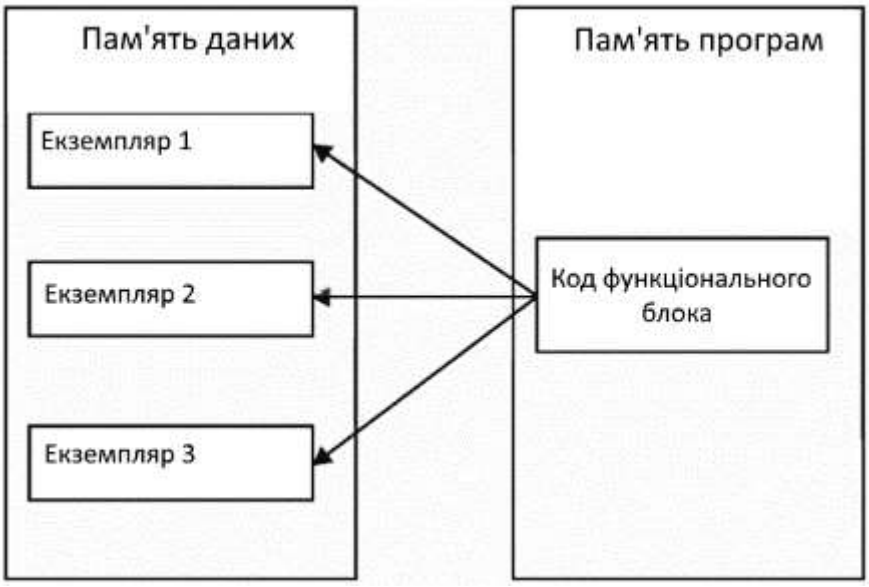


Рисунок 5.7 – Розподіл пам’яті для екземплярів функціональних блоків

Екземпляр функціонального блоку створюється в розділі оголошень змінних функціонального блоку, програми або в розділі глобальних змінних проекту. Як і змінні, він має отримати унікальний ідентифікатор. Наприклад, створення екземпляра стандартного функціонального блоку «інкрементний лічильник» з ідентифікатором ctuTimeMeter має такий вигляд:

ctuTimeMeter: CTU;

Очевидно, що створювати екземпляри можна тільки для відомих системі блоків. Це бібліотечні блоки або блоки, раніше реалізовані користувачем. З точки зору транслятора, створення екземпляра означає виділення необхідної пам’яті для розміщення змінних блоку.

Екземпляр функціонального блоку можна не тільки викликати, але й використовувати як вхідні змінні інших функціональних блоків.

Функціональним блоком іноді називають екземпляр функціонального блоку. У даному посібнику такі неоднозначні скорочення не застосовуватимуться. Дозволимо собі лише називати іноді функціональний блок просто блоком, а екземпляр функціонального блоку – екземпляром.

Доступ до змінних екземпляра

Після створення екземпляра функціонального блоку можна відразу почати працювати з його даними. При цьому зовсім не обов'язково викликати його. Звертатися до змінних екземпляра можна так само, як до елементів структури даних, через точку:

```
ctuTimeMeter.RESET := FALSE;
ctuTimeMeter.PV := 100;
x := ctuTimeMeter.CV;
```

Входи екземпляра блоку доступні для запису і читання ззовні. Виходи – тільки для читання. Змінювати значення виходів можна тільки з тіла блоку, ззовні не можна. Транслятор відстежує такі спроби і видає повідомлення про помилку.

Виклик екземпляра блоку

Викликати екземпляр функціонального блоку з перерахуванням параметрів як функцію не можна. Значення вхідних змінних мають присвоюватися безпосередньо. У текстових мовах вхідні змінні перераховуються в дужках, після імені екземпляра. Присвоєння вхідних значень виконується операцією «:=».

Мовою ST:

```
ctuTimeMeter (RESET := FALSE);
```

Мовою IL:

```
CAL ctuTimeMeter (RESET := FALSE)
```

Спеціальний символ «=>» дозволяє отримати значення виходів після виконання блоку:

```
ctuTimeMeter (RESET := FALSE, CU := Inp1, CV => x);
```

Під час виклику екземпляра можна визначити тільки необхідні параметри, причому в довільному порядку. У графічних мовах не використовувані входи й виходи екземпляра блоку просто залишаються не підключеними. Як показано на рис. 5.8, вхід PV залишився не підключеним.

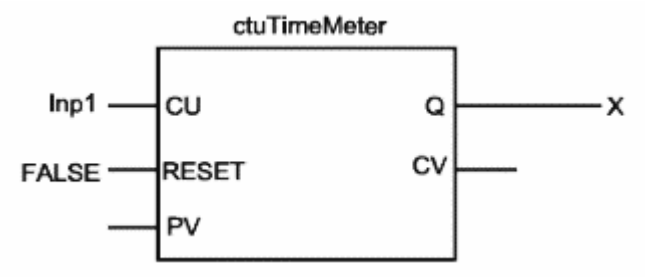


Рисунок 5.8 – Виклик екземпляра функціонального блоку (FBD)

У мові ST за відсутності параметрів порожні дужки після імені екземпляра ставити не потрібно:

```
ctuTimeMeter();     (*зайві дужки*)
```

Використовувати екземпляри функціональних блоків у виразах не можна, але можна використовувати їх входи і виходи:

```
X := ctuTimeMeter.PV – ctuTimeMeter.CV + 1;
```

Можна визначити значення входів заздалегідь і викликати екземпляр функціонального блоку взагалі без параметрів.

Мовою ST:

```
 ctuTimeMeter.RESET := FALSE;
 ctuTimeMeter;
```

Мовою IL:

```
 LD    FALSE
 ST    ctuTimeMeter.RESET
 CAL   ctuTimeMeter
```

Потрібно звернути увагу на те, що МЕК не заохочує використання елементів даних окремо від виклику екземпляра блоку, оскільки це може привести до проблем з використанням екземпляра блоку в багатозадачних проектах. З іншого боку, багаторазове повторне присвоювання вхідних значень збільшує розмір коду і знижує ефективність програми.

Ініціалізація даних екземпляра

Під час опису блоку в розділі оголошень можна явно привласнити початкові значення змінним. Наприклад:

```
FUNCTION_BLOCK SyncSwitch
VAR_INPUT
…
Sync:          BOOL := TRUE;
```

Зі створенням екземпляра функціонального блоку SyncSwitch вхідна змінна Sync отримає значення TRUE. Якщо початкові значення не задані, використовуються нульові значення.

Екземпляр функціонального блоку може потребувати індивідуальної ініціалізації, відмінної від тієї, яка визначена в процесі реалізації. Установку початкових значень змінних найпростіше виконати під час створення екземпляра. Значення, задані у ході створення екземпляра, сильніше значень, заданих під час реалізації блоку.

SyncSwl: SyncSwitch := (Sync := FALSE);

Тепер змінна SyncSwl.Sync отримає початкове значення FALSE, незважаючи на значення, вказане в оголошенні блоку.

Фізично початкові значення змінних отримують ще до першого використання екземпляра. Операція початкової ініціалізації змінних проводиться зі скидання, яке виконується безпосередньо після завантаження проекту в пам’ять ПЛК, за командою відладчика або з перезапуском контролера.

Деякі транслятори мають опцію відключення ініціалізації за замовчуванням з метою прискорення запуску ПЛК. У цьому випадку покладатися на те, що змінні, які не мають явно зазначених початкових значень, отримуватимуть однакові значення з перезапуском системи, не можна.

Можливі випадки, коли екземпляру функціонального блоку потрібна розумна ініціалізація. Наприклад, для настроювання блоку необхідно провести деякі обчислення. Спеціальної процедури ініціалізації в функціональних блоках не передбачено. Тут доведеться витратити на ініціалізацію один або кілька перших циклів виконання екземпляра. Закінчення складної процедури ініціалізації повідомляють зазвичай індикацією виходом готовності (ENO). Часто буває, що зручно застосувати для ініціалізації дію і зосередити контроль над ініціалізацією в одному місці (зазвичай за крок Init SFC діаграми). Такий метод дозволяє проводити ініціалізацію даних, екземплярів блоків і програм у необхідній послідовності і взаємозв’язку. У більшості ж практичних випадків для блоків, які потребують певної настройки, виявляється достатнім ввести кілька спеціальних входів (уставок). Так зроблено в усіх стандартних блоках.

Тиражування екземплярів

За необхідності отримати копію екземпляра функціонального блоку можна, використовуючи оператор присвоювання:

SyncSw, SyncSw2: SyncSwitch; (*працюємо з екземпляром SyncSwl*)

…

Мовою ST:

SyncSw2 := SyncSwl;

Мовою LD реалізація цього прикладу показана на рис. 5.9.

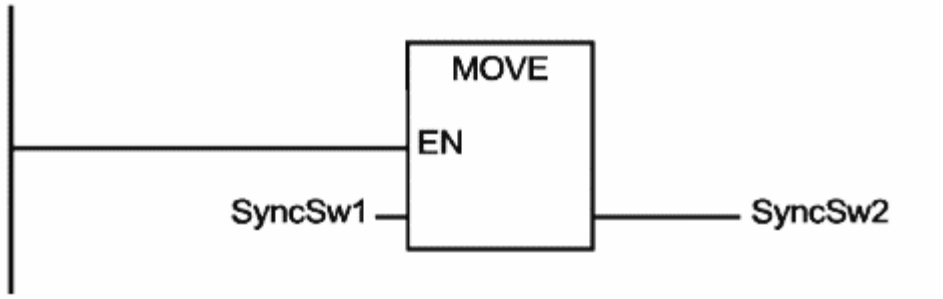


Рисунок 5.9 – Присвоювання екземплярів функціонального блоку

Оператор присвоювання виконує побайтне копіювання даних екземпляра блоку за аналогією зі структурами даних. Очевидно, що така техніка копіювання підходить тільки для екземплярів одного й того самого функціонального блоку.

Особливості реалізації та застосування функціональних блоків

Вхідні змінні всередині блоку доступні для запису. Це викликає певний інтерес для програміста. Так, наприклад, вхідну змінну зручно застосувати як лічильник ітерацій, якщо вона саме відображає кількість потрібних повторень. Це дозволить уникнути створення додаткової локальної змінної. Оскільки під час виклику екземпляра блоку ввідна змінна має отримати нове значення, нічого страшного на перший погляд немає. Виклик екземпляра не зобов’язаний супроводжуватися привласненням значень усім формальним параметрам.

Можливо, у певний момент ви вирішите, що вхідний параметр вже визначено, і можна не ставити його повторно. В результаті значення параметра дорівнюватиме тим значенням, які він мав під час роботи в ролі локальної змінної, при попередньому виклику екземпляра. Звичайно, можна придумати багато прикладів, коли зміна значення вхідної змінної безпечна.

Проте використовувати такий прийом потрібно виключно обдумано й обережно. У загальному випадку в ході реалізації блоку вхідні змінні потрібно розглядати як константи.

Застосування глобальних змінних у функціональних блоках викликає ті самі проблеми, що і в функціях. Екземпляри блоку перестають бути незалежними. Зміни змінної, виконані одним екземпляром, проявлять себе зовсім в іншому місці. Іноді це дійсно необхідно, але у звичайній практиці бажано обмежувати таке застосування глобальних змінних.

Якщо екземпляри функціонального блоку використовують глобальну змінну тільки для читання, то жодних побічних явищ виникнути не може. Аналогічна ситуація виникає в ході застосування змінних, що мають пряму адресацію.

Зі входами проблем немає. Єдине обмеження – це погіршення можливостей блоку в плані його повторного застосування. З використанням блоку в іншій програмі або проекті прямі адреси доведеться поправити. Добре вирішення цієї проблеми дають шаблонні змінні.

Шаблонні змінні

Шаблонні змінні або, як їх іноді називають, конфігураційні змінні (в CODESYS variable configuration) є частково визначеними змінними з прямою адресацією. Прямий доступ замінюється зірочкою. Визначення повної адреси шаблонної змінної дається у спеціальному розділі ресурсів.

Розглянемо, як це робиться на прикладі. Створимо блок SHABLON, який матиме стандартну змінну bMarvel. Для нашого прикладу достатньо розділу оголошень:

```
FUNCTION_BLOCK SHABLON
VAR
bMarvel AT %I*   : BOOL;
END_VAR
```

Далі створимо два екземпляри блоку SHABLON у головній програмі:

```
PROGRAM PLC_PRG
VAR
Shablon1:   SHABLON;
Shablon2:   SHABLON;
END_VAR
```

Їх потрібно прописати в ресурсах проекту:

```
VAR_CONFIG
PLC PRG.Shablon1.bMarvel    AT %IX1.0 : BOOL;
PLC_PRG.Shablon1.bMarvel    AT %IX1.1 : BOOL;
END_VAR
```

Типи даних шаблонної змінної, які зазначені в оголошенні блоку і в ході настроювання адреси в ресурсах, мають співпадати.

Приклад функціонального блоку

Як простий приклад реалізуємо блок синхронного перемикача SyncSwitch. Алгоритм його роботи такий: вихід перемикача Q приймає значення, які рівні входу Start, але перемикання виходу дозволено тільки при Sync: = TRUE. Графічно це відображено на рис. 5.10.

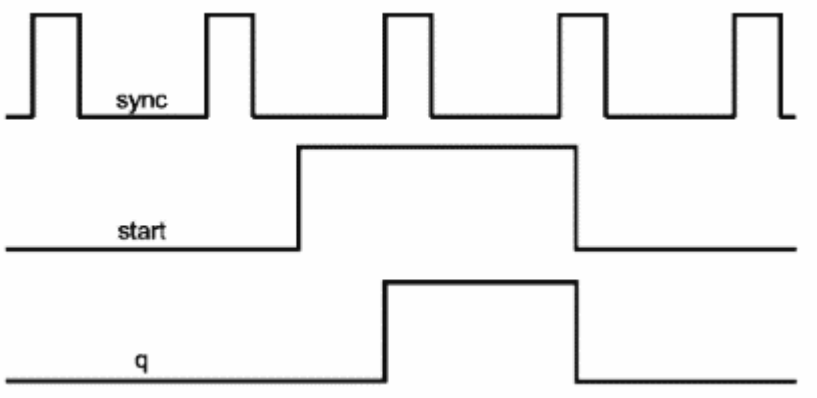


Рисунок 5.10 – Діаграма роботи синхронного перемикача

Умова включення виходу виражається рівнянням: Q = start AND sync, а умова виключення Q = NOT start AND sync. Значення виходу має зберігатися між синхроімпульсами, тому використовувати тут функцію не можна. Мовою IL блок SyncSwitch можна реалізувати так:

```
FUNCTION_BLOCK SyncSwitch VAR INPUT
    Sync:       BOOL
    Start:      BOOL
END_VAR
VAR_OUTPUT
    Q:          BOOL
END_VAR
```

```
LD      Sync
AND     Start
S       Q
LD      Sync
ANDN    Start
R       Q
```

Такий блок корисний під час реалізації «безударного» перемикання в ланцюзі змінного струму. Імпульси синхронізації мають відображати інтервали, коли миттєве значення напруги близьке до нуля. Тоді перемикання силового ланцюга з виходу SyncSwitch відбуватиметься без кидка струму.

Звичайно, практичне значення нашого нового блоку дещо програє, якщо згадати про стандартний домінантний перемикач – SR. З ним завдання вирішується в один рядок (ST):

SR_1(SET1:= Start AND Sync, RESET:= Sync, Ql=> Q);

Дії

У функціональних блоках МЕК дуже не вистачає можливості виконувати декілька різних операцій. Особливо, якщо блок містить об'ємні дані. Можна, звичайно, зробити додатковий вхід і по ньому аналізувати, що ми хочемо від блоку. У CODESYS ця проблема вирішена так. Функціональні блоки і програми можна доповнювати діями. Дія працює всередині блоку з повним правом доступу до всіх даних. Його можна викликати як з тіла блоку, так і ззовні. Дія вказується через точку після назви екземпляра блоку і може мати перелік значень входів і виходів. Під час виклику дії з тіла блоку найменування екземпляра не потребується. Дія не має власних даних і використовує входи, виходи і локальні змінні блоку. Мова реалізації дії може бути довільною.

Для прикладу доповнимо вищеописаний блок SyncSwitch дією EmergencyBreak. Нехай його виклик призводить до миттєвого безумовного виключення виходу. Для визначення дії потрібно обрати блок в органайзері об'єктів CoDeSys і дати команду «Add Action». Опишемо дію мовою ST:

Q := FALSE;

Викликати дану дію з ST-програми можна так:

SyncSwitchl.EmergencyBreak(Q => q);

У графічних мовах прямокутник, який представляє дану дію, матиме заголовок SyncSwitch1.EmergencyBreak.

Зверніть увагу, що вікно редактора для дій не має розділу оголошень. Компоненти програм з діями мають списки дій, що розкриваються в органайзері об’єктів. Список дій у розділі оголошень POU не відображається.

Дії аналогічні методам класу в C ++. Термін «дія» навіть більш зрозумілий, ніж «метод». Своїм походженням дії зобов’язані SFC. У CODESYS дії можна використовувати як підпрограми.

5.2 ПЛК, як кінцевий автомат

Як було описано вище, ПЛК функціонує циклічно – читання входів, виконання прикладної програми і запис виходів. В результаті прикладне програмування для МЕК ПЛК істотно відрізняється від традиційної моделі, яка застосовується під час роботи мовами високого рівня ПК. Розглянемо як ілюстрації найпростішу задачу: необхідно запрограмувати мигаючий світловий індикатор.

Очевидно, що алгоритм має бути приблизно такий:

– увімкнути вихід;

– витримати паузу;

– вимкнути вихід;

– витримати паузу;

– перехід до кроку 1 (початок програми);

– кінець програми.

Реалізована за цим алгоритмом програма для ПЛК не працюватиме. По-перше, вона містить безкінечний цикл. Весь код прикладної програми виконується від початку і до кінця в кожному робочому циклі. Будь-яка прикладна програма ПЛК є частиною робочого циклу і має повертати управління системі виконання. Тому крок 5 «перехід на початок програми» зайвий.

Якщо в нашому алгоритмі видалити «перехід на початок», програма працюватиме. Хоча і не так, як задумано. Вихід завжди залишатиметься у вимкненому стані, оскільки фізично встановлення значень виходів здійснюється після закінчення прикладної програми один раз. Проміжні зміни значень виходів не відображаються на апаратні засоби. Звичайно, значення

змінної змінюватиметься багаторазово, але визначальним для виходу стане тільки останнє значення.

Що ще поганого для ПЛК у даному алгоритмі, так це затримка часу. Цілком імовірно, що, крім мигання одним виходом, ПЛК має виконувати ще й іншу роботу. Тобто програму необхідно буде доповнювати. Але якщо контролер зайнятий очікуванням, то в даному алгоритмі це означає, що нічого іншого він робити не зможе. Отже, затримку часу необхідно організувати інакше. Достатньо засікти час і зайнятися іншими справами, контролюючи періодично годинник. Тут немає нічого особливого. Так роблять зазвичай і більшість людей в очікуванні призначеного часу.

З урахуванням наведених міркувань алгоритм мигаючого індикатора для ПЛК має бути таким:

1. Перевірити таймер, якщо час паузи вийшов, то:
   – інвертувати вихід (включити, якщо вимкнений, і навпаки);
   – почати відлік нової паузи;
2. Кінець програми.

Незважаючи на описані складності, алгоритм вийшов в результаті простіше. Так і має бути. Технологія ПЛК спеціально орієнтована на подібні завдання.

Щоб писати гарні програми для ПЛК, потрібно навчитися думати певним чином. Секрет полягає в тому, щоб уявляти собі контролер не як машину, що послідовно виконує команди програми, а як кінцевий автомат.

У будь-якому автоматі існує множина входів (X), множина виходів (Y) і множина можливих станів (S). У нашому випадку це кінцеві множини, оскільки кількість входів-виходів ПЛК обмежена, так само, як і обсяг пам’яті змінних (що визначають можливі стани). Початковий стан ($s_0 \in S$) однозначно визначено. Автомат працює за тактами, для ПЛК це робочий цикл. У кожному такті значення входів відомі. Значення виходів визначаються (функція виходів λ) значеннями входів і поточним станом. Реакція автомата залежить тільки від поточного стану без передісторії, тобто не має значення, як він прийшов у даний стан. Разом з тим поточний стан також змінюється за тактами, автомат переходить у новий стан (функція переходів δ). У теорії автоматів описані шість об’єктів $A = \{X, Y, S, s_0, \lambda, \delta\}$ прийнято називати кінцевим автоматом Мілі.

Класична сфера застосування ПЛК – це програмна реалізація автоматів. Саме це і зумовило підхід до програмування ПЛК. Контролер обчислює програмно задану функцію виходів і функцію переходів. У кожному робочому циклі ПЛК виконує розрахунок нових значень для виходів, які необхідно змінити. В результаті класична прикладна програма ПЛК виявляється більш схожою на обчислення за формулою.

Дещо розширивши поняття автомата, ми можемо розглядати переходи як функції подій. Події не обов'язково мають бути пов'язані зі входами, це досить абстрактне поняття. Тоді закінчення часу очікування можна буде просто розуміти як подію, причому цілком не важливо, як саме реалізований сам таймер. Модель такої системи зручно подати у вигляді спрямованого графа станів (state charts). Стани відображаються овалами, які містять значення набору змінних, а переходи – спрямованими дугами (рис. 5.11). Діаграми станів дуже ефективний інструмент проектування й аналізу автоматів.

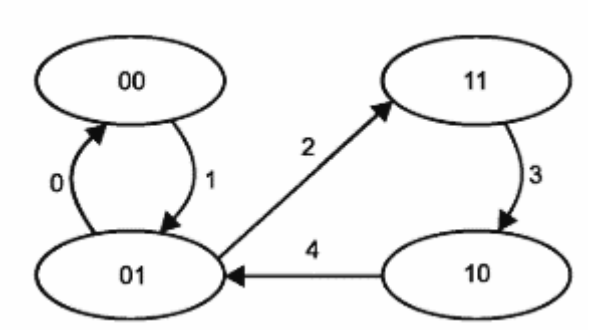


Рисунок 5.11 – Граф станів для двох змінних

Технічна база для побудови автоматів досить широка. Це механічні вузли, пневматичні елементи, реле або логічні мікросхеми тощо. Але на відміну від будь-яких інших реалізацій автоматів технологія ПЛК забезпечує швидке й виключно гнучке рішення. Безумовно, в ході побудови автоматів на базі програмованих логічних матриць і мікропроцесорів перепрограмування також можливо, але значно більш трудомістке. Це можна зробити тільки за наявності відповідного обладнання і спеціальної підготовки.

Реально можливості ПЛК суттєво перевищують кінцеві автомати. Далеко не все, що можна зробити на ПЛК, вписується в рамки кінцевих автоматів. Це функції управління за часом, математична обробка даних, регулювання тощо. Проте застосування формалізму кінцевих автоматів дозволяє значно спростити процес проектування.

Причому це стосується не тільки ПЛК. Подібний підхід лежить в основі універсальної мови моделювання Unified Modeling Language (UML). Пакет розширення Stateflow матричної системи комп'ютерної математики MATLAB забезпечує побудова анімаційних діаграм стану моделей різних пристроїв і систем. Він дозволяє виконувати ситуаційне моделювання додатково до імітаційного моделювання, що виконується потужним пакетом розширення системи MATLAB-Simulink.

5.3 Структурований текст (ST)

5.3.1 Особливості мови ST

Мова ST (Structured Text) – це мова високого рівня. Синтаксично ST – це дещо адаптована мова Паскаль. Замість процедур Паскаля в ST використовуються компоненти програм стандарту МЕК.

Для фахівців, знайомих з мовою С, освоєння ST також не буде занадто складним.

У більшості комплексів програмування ПЛК мова ST за замовчуванням пропонується для опису дій і умов переходів SFC. Це дійсно максимально потужний тандем, що дозволяє ефективно вирішувати будь-які завдання. Розглянемо основні елементи мови ST.

5.3.2 Вирази

Основою ST-програми є вирази. Результат обчислення виразу присвоюється змінній за допомогою оператора «: =», як і в Паскалі. Кожен вираз обов'язково закінчується крапкою з комою «;». Вираз складається із змінних констант і функцій, розділених операторами:

iVarl := 1 + iVar 2 / ABS(iVar2);

Стандартні оператори у виразах ST мають символьне подання, наприклад математичні дії: +, – *, /, операції порівняння тощо.

Крім операторів, елементи виразу можна відокремлювати пробілами і табуляціями для кращого сприйняття. У текст можуть бути введені коментарі. Скрізь, де допустимі пасивні роздільники, можна вставляти і коментарі:

iVar1 := 1 + (*отримати знак*) iVar2 / ABS(iVar2); (*перевірка на 0 була вище*)

Декілька виразів можна записати поспіль в один рядок. Проте гарним стилем вважається запис одного виразу в рядку. Довгі вирази можна перенести на наступний рядок. Перенесення рядка рівноцінно пасивному роздільнику.

Вираз може включати інший вираз, який записують у дужках. Вираз, який записаний у дужках, обчислюється насамперед. Тип виразу визначається типом результату обчислень:

```
bAlarm := byInp1 > byInp2 AND byInp1 + byInp2 <> 0 OR bAlarm2;
```

5.3.3 Порядок обчислення виразів

Обчислення виразу відбувається відповідно до правил пріоритету операцій. Першими виконуються операції з найвищим пріоритетом.

У порядку зменшення пріоритету операції розташовуються так: вираз в дужках; виклик функції; ступінь EXPT; заміна знаку (–); заперечення NOT; множення, ділення й ділення за модулем MOD; додавання і віднімання (+, –); операції порівняння (<, >, <=, >=); рівність (=); нерівність (<>); логічні операції логічні операції AND, XOR і OR. Пріоритет операцій у виразах дуже важливий. Насамперед з математичної точки зору:

```
X := 2 + 2 * 2;              (* = 6*)
X := (2 + 2) * 2;            (* = 8*)
```

Тут результати очевидні. Множення має більш високий пріоритет, ніж складання, і виконується раніше. Дужки змінюють порядок обчислень, і результат виявляється іншим.

У ході складання виразів обов’язково необхідно враховувати можливий діапазон зміни значень і типи змінних. Помилки, пов’язані з переповненням, виникають у процесі виконання і не можуть бути виявлені транслятором.

```
iVar1:      SINT;
siVar2:     SINT := 120;
siVarl := 120 – siVar2 + 20;          (*120 – 120 = 0, 0 + 20 = 20*)
siVarl := 120 – ( siVar2 + 20);       (*120 + 20 = -116, 120 +116 = -20*)
```

5.3.4 Порожній вираз

Порожній вираз складається з крапки з комою «;». Для крапки з комою транслятор не генерує жодного коду. Якщо випадково поставити зайву «;», це не викличе помилки. Єдине осмислене застосування порожнього виразу –

це забезпечення правильності мовних конструкцій. Наприклад, може знадобитися відтранслювати проект, що містить ще не реалізований POU. Для коректної трансляції достатньо написати в тілі POU один порожній оператор.

Ще один приклад, де порожній оператор виявляється до речі, – це умова IF, що не містить розділ THEN:

```
IF x = Threshold THEN
        ;                               (*так правильно*)
ELSIF x > Threshold THEN
        bMarker := bMarker – 1;         (*крок униз*)
ELSE
        bMarker := bMarker + 1;         (*крок вгору*)
END_IF;                                 (*зайва ; *)
```

5.3.5 Оператор вибору IF

Оператор вибору дозволяє виконати різні групи виразів залежно від умов, виражених логічними виразами. Повний синтаксис оператора IF (ЯКЩО) виглядає так:

```
IF <логічний вираз IF>
THEN
<вираз IF> ;
[
ELSIF < логічний вираз ELSEIF 1>
THEN
        <вираз ELSEIF 1> ;
        …
ELSIF < логічний вираз ELSEIF n>
THEN
        <вираз ELSEIF n> ;
ELSE
        <вираз ELSE> ;
]
END_IF
```

Якщо <логічний вираз IF> ІСТИНА, то виконуються вирази першої групи – <вираз IF>. Інші вирази пропускаються, альтернативні умови не перевіряються.

Частина конструкції в квадратних дужках є необов'язковою і може бути відсутня.

Якщо <логічний вираз IF> ХИБНІСТЬ, то одне за одним перевіряються умови ELSIF. Перша істинна умова призведе до виконання відповідної групи виразів. Інші умови ELSIF не аналізуватимуться. Груп ELSIF може бути декілька або не бути зовсім.

Якщо всі логічні вирази дали помилковий результат, то виконуються вирази групи ELSE, якщо вона є. Якщо групи ELSE немає, то не виконується нічого.

У найпростішому випадку оператор IF містить тільки одну умову. Наприклад:

```
IF bReset THEN
    iVarl := 1;
    iVar2 := 0;
END_IF
```

На перший погляд конструкція IF з декількома групами ELSIF виглядає складною, але насправді виявляється досить виразною, наприклад:

```
IF bReset THEN
    iVarl := 1;
    ELSIF byLeft <16 THEN
        iVarl := 2;
    ELSIF byLeft < 32 THEN
        iVarl := 3;
    ELSIF byLeft < 64 THEN
        iVarl := 4;
ELSE
    bReset := TRUE;
END IF
```

5.3.6 Оператор множинного вибору CASE

Оператор множинного вибору CASE дозволяє виконати різні групи виразів залежно від значення однієї цілочисельної змінної або виразу. Синтаксис:

```
CASE < цілочисельний вираз> OF
    <значення 1>:
        <вираз 1> ;
```

```
        <значення 2> , <значення 3> :
                <вираз 3> ;
        <значення 4>..<значення 5> :
                <вираз 4> ;
        …
[
ELSE
        <вираз ELSE> ;
]
END CASE
```

Якщо значення виразу збігається із заданою константою, то виконується відповідна група виразів. Інші умови не аналізуються (<значення 1>: <вираз 1>;).

Якщо кілька значень констант мають відповідати одній групі виразів, їх можна перерахувати через кому (<значення 2>, <значення 3>: <вираз 3> ;).

Діапазон значень можна визначити через двокрапку (<значення 4> .. <значення 5>: <вираз 4> ;).

Група виразів ELSE є необов'язковою. Вона виконується під час незбігу жодної з умов (<вираз ELSE> ;).

Наприклад:

```
CASE byLeft/2 OF
0,127:
        bReset := TRUE;
        Var1 := 0;
16..24:
        Var1 := 1;
ELSE
        Var1 := 2;
END_CASE
```

Значеннями вибору CASE можуть бути тільки цілі константи, змінні використовувати не можна.

Однакові значення в альтернативах вибору задавати не можна, навіть у діапазонах.

Так, наступний приклад викликає помилку в ході трансляції:

```
CASE byLeft OF
20:         Varl := 0;
16..24:     Varl := 1;
END_CASE
```

Безумовно, оператор CASE «слабкіше» оператора IF, який не має подібних обмежень. Але формат CASE не тільки виразніший для програміста, але й більш ефективний. Використання цілочисельних констант дозволяє транслятору виконати оптимізацію коду, часто досить суттєву.

5.3.7 Цикли WHILE і REPEAT

Цикли WHILE і REPEAT забезпечують повторення групи виразів, поки вірний умовний логічний вираз. Якщо умовний вираз завжди істинний, то цикл стає безкінечним. Синтаксис WHILE:

```
WHILE <Умовний логічний вираз> DO
    <Вираз – тіло циклу>
END WHILE
```

Умова в циклі WHILE перевіряється до початку циклу. Якщо логічний вираз спочатку має значення ХИБНІСТЬ, тіло циклу не буде виконано жодного разу.

Синтаксис REPEAT:

```
REPEAT
    <Вираз – тіло циклу>
UNTIL <Умовний логічний вираз>
END_REPEAT
```

Умова в циклі REPEAT перевіряється після виконання тіла циклу. Якщо логічний вираз спочатку має значення ХИБНІСТЬ, тіло циклу буде виконано один раз, наприклад:

```
ci := 64;
WHILE ci > 1 DO
    Varl := Varl + 1;
    ci := ci/2;
END_WHILE
```

Правильно побудований цикл WHILE або REPEAT обов'язково має змінювати змінні, які складають умову закінчення в тілі циклу, поступово наближаючись до умови завершення. Якщо цього не зробити, цикл не закінчиться ніколи.

Потрібно намагатися не використовувати точні рівність або нерівність для припинення циклу. Інакше є ймовірність помилково проскочити граничну умову. Краще використовувати умови більше й менше. У наступному прикладі помилка добре видна виключно завдяки його простоті:

```
ci := 1;
WHILE ci <> 100 DO
        Varl := Varl + 1;
        ci := ci + 10;
END_WHILE
```

Очевидно, лічильник *ci* за початкового значення 1 і приросщення 10 ніколи не стане рівним 100.

Для реалізації мінімального часу виконання циклу необхідно уникати в тілі циклу і в умовному виразі обчислень, які можна було зробити заздалегідь. Такі обчислення повторюються в циклі, щоразу забираючи час. Наприклад:

```
WHILE ci < 5 + x DO
        Var := Varl + 2*x*x + 1;
        ci := ci + 1;
END_WHILE
```

Даний цикл можна оптимізувати за швидкістю:

```
iMax := 5+x;
iPoly := 2*x*x + 1;
WHILE ci < iMax DO
        Var := Varl + iPoly;
        ci := ci + 1;
END_WHILE
```

5.3.8 Цикл FOR

Цикл FOR забезпечує задану кількість повторень групи виразів. Синтаксис:

FOR <Цілий лічильник> := <Початкове значення>

TO <Кінцеве значення>
[BY <Крок>] DO
    <Вираз – тіло циклу>
END_FOR

Перед виконанням циклу лічильник отримує початкове значення. Далі тіло циклу повторюється, поки значення лічильника не перевищить кінцевого значення. Лічильник збільшується в кожному циклі. Початкове й кінцеве значення і крок можуть бути як константами, так і виразами.

Лічильник змінюється після виконання тіла циклу. Тому якщо задати кінцеве значення менше початкового, то за позитивного приросту цикл не буде виконаний жодного разу. За однакових початкового й кінцевого значень тіло циклу буде виконано один раз.

Частина конструкції BY у дужках не обов'язкова, вона визначає крок збільшення лічильника. За замовчуванням лічильник збільшується на одиницю в кожній ітерації.

Як лічильник можна використовувати змінну будь-якого цілого типу.

Приклад:

```
Var1 := 0;
FOR cw := 1 TO 10 DO
    Varl := Varl + 1;
END_FOR
```

Даний цикл буде виконано 10 разів і відповідно Varl матиме значення 10.

Крок зміни лічильника ітерацій може бути і негативним. Початкова умова в цьому випадку має бути більше кінцевої. Цикл буде закінчений, коли значення лічильника стане менше кінцевого значення. Наприклад:

```
Varl := 0;
FOR ci := 10 TO 1 BY -1 DO
    Varl := Varl + 1;
END_FOR
```

Цикл FOR зручний для ітерацій із заздалегідь відомою кількістю повторів. Причому, щоб створити безкінечний цикл FOR, потрібно добре постаратися. Можна, наприклад, спробувати поставити нульовий крок збільшення (в CODESYS це не допомагає) або скинути лічильник в тілі циклу.

Для побудови правильного циклу достатньо дотримуватися двох простих формальних вимог:

– не змінюйте лічильник циклу і умову закінчення в тілі циклу. Лічильник та змінні, які утворюють кінцеву умову в циклі, можна використовувати тільки для читання;

– не ставте як кінцеву умову максимальне для типу змінної лічильника значення. Так, якщо для однобайтного цілого без знака задати константу 255, то умова закінчення ніколи не буде виконана. Цикл стане безкінечним.

У CODESYS лічильник змінюється в тілі циклу завжди, включаючи завершальну ітерацію, коли умова закінчення вже досягнута. Але в стандарті такі тонкощі не обумовлені. З метою оптимізації за швидкістю транслятор може обійти змінення лічильника зайвий раз. Тому не рекомендується використовувати значення лічильника поза тіла циклу. Переносність такої програми гарантувати не можна. Уникайте використовувати цикл FOR зі складними умовами закінчення і у випадках, коли після закінчення циклу необхідно визначити причину закінчення. Наприклад, коли цикл може бути перерваний оператором EXIT і є необхідність дізнатися, скільки ітерацій насправді було виконано (див. нижче приклад з EXIT).

Інший момент, який необхідно враховувати під час створення переносної програми, – це негативне прирощення лічильника. На жаль, далеко не всі системи програмування підтримують таку можливість.

5.3.9 Переривання ітерацій операторами EXIT і RETURN

Оператор EXIT, поміщений у тілі циклів WHILE, REPEAT і FOR, призводить до негайного закінчення циклу. Гарний стиль програмування закликає уникати такого прийому, але іноді він дуже зручний. Розглянемо, наприклад, пошук елемента масиву з певним значенням (x). Найпростіше організувати лінійний перебір за допомогою циклу FOR:

```
bObtained:= FALSE;
FOR cN := 1 TO Maxlndex DO
        IF x = aX[cN] THEN
                Index := cN;
                bObtained := TRUE;
                EXIT;
        END_IF
```

```
END_FOR
IF bObtained THEN        (*елемент знайдено, його індекс – Index*)
```

Для вкладеного циклу оператор EXIT завершує тільки «свій» цикл, зовнішній цикл продовжуватиме роботу. Наприклад:

```
FOR y := O TO 9 DO
    FOR x := O TO 99 DO (*обробляємо рядок масиву Arr[y][x]*)
        ;
        IF ... THEN EXIT;
    END_FOR
ENDFOR
```

За необхідності завершення зовнішнього циклу за умови, яка з’явилася у вкладеному циклі, можна використовувати пару синхронізованих операторів EXIT:

```
bBreakY := FALSE;
    FOR y := O TO 9 DO
        FOR x := 0 TO 99 DO
            ;
            IF ... THEN bBreakY := TRUE; EXIT;
            (*перервати обробку*)
        END_FOR
    IF bBreakY THEN EXIT;
END_FOR
```

Оператор RETURN здійснює негайне повернення з POU. Це єдиний спосіб перервати вкладені ітерації без введення додаткових перевірок умов. Оператор RETURN виконується дуже швидко, фактично це одна машинна команда процесора. Але не варто ним зловживати, оскільки в тексті компонента, що має, наприклад, 50 виходів, розібратися досить непросто.

Іноді буває зручно створити безумовний цикл, а умови виходу формувати в тілі циклу з використанням EXIT. Наприклад, можуть знадобитися кілька рівноймовірних, але не взаємопов’язаних умов виходу з циклу. Створити безумовний (безкінечний) цикл у ST найпростіше так: WHILE TRUE DO...

5.3.10 Ітерації на базі робочого циклу ПЛК

Використовувати для умови виходу з циклів WHILE і REPEAT входи, виходи або інші апаратно залежні змінні ПЛК не можна. Дані змінні не змінюють своїх значень у межах одного робочого циклу користувальницької програми, тому цикл завжди буде безкінечним. Якщо подібна необхідність все ж існує, використовуйте для ітерацій робочий цикл ПЛК. Виконання ітерації задається простою умовою IF.

Аналогічно можна робити за необхідності побудови тривалих циклів. Наприклад, ініціалізація або копіювання великих масивів даних. Розподілення об'ємних операцій на декілька робочих циклів є стандартним прийомом, що дозволяє уникнути небажаного уповільнення інших, паралельно виконуваних завдань.

Так, ініціалізація даних функціонального блоку з виставленням сигналу готовності може виглядати так:

```
ENO:        BOOL := FALSE;
niCounter:  INT := 1000;
aiVar:      ARRAY[0..999] OF INT;

IF niCounter = 0 THEN
        ENO := TRUE;
        …     (*основна робота блоку*)
ELSE
        niCounter := niCounter – 1;
        aiVar[niCounter] := GetlnitVal(niCounter);
END IF
```

5.3.11 Оформлення тексту

Оформлення текстів ST-програм може бути абсолютно довільним. Розташування операторів і виразів в рядку не впливає на правильність програм. Але дуже важливо виробити свій власний стиль і чітко дотримуватися його.

Найважливішу роль в оформленні відіграють відступи на початку рядків. Відступи візуально об'єднують рядки, що містять вирази одного рівня вкладення. Текст, вирівняний у вигляді драбинки, кожна сходинка якої належить до одного циклу або умови, читається легко. Незважаючи на можливість горизонтальної прокрутки в редакторі, бажано, щоб за шириною

текст містився на одній сторінці. Не варто розташовувати кілька виразів в один рядок. Нічого страшного немає в тому, що текст виявиться розтягнутим по вертикалі: лаконічні вирази і навіть порожні рядки тільки допомагають зоровому аналізу. Наприклад:

```
FOR icY := 0 TO 8 DO
        FOR icX := 0 TO 16 DO
                IF iaPos[icY,icX] > iLevel THEN
                        iBalance := iBalance + 1;
                ELSE
                        IF iaPos[icY,icX] < iLevel THEN
                                iBalance := iBalance – 1;
                        END_IF
                END_IF
        END_FOR
        iLevel := iLevel *2;
END_FOR
```

Погано оформлений ST-текст читати вкрай важко, навіть редактор з колірним виділенням інструкцій тут не допоможе. Мало того, помилки у схемі відступів здатні заплутати розуміння сенсу:

```
FOR icY := O TO 8 DO
        FOR icX := O TO 16 DO
                IF iaPos[icY,icX] > iLevel THEN
                iBalance := iBalance + 1;
                ELSE
        IF iaPos[icY,icX] < iLevel THEN
                iBalance := iBalance – 1;
        END_IF
        END_IF
        END_FOR
iLevel := iLevel *2;
END_FOR
```

Для оформлення ST текстів цілком можна застосувати рекомендації, які зустрічаються в літературі з програмування на Паскалі і С. Зверніть увагу, що в

ST відсутні програмні дужки (в Паскалі: begin, end; в С: {}). Замість них кожен вираз мови має власну кінцівку (WHILE .. END_WHILE, IF .. END_IF). Тобто, програмна дужка, що закриває, є інформативною. Візуально такий текст сприймається явно краще. Зі створенням складних вкладень у мові С дужки, що закривають, часто розташовані суцільною драбинкою. У таких випадках досвідчені програмісти застосовують стислі коментарі після кожної дужки, що закриває. Коментарі підказують, з чого розпочато даний рівень відступу. Наприклад: (*FOR x*). Це гарний прийом, але при грамотному застосуванні відступів у рядках ST така необхідність виникає значно рідше, ніж у С і Паскалі.

5.4 Релейні діаграми (LD)

5.4.1 Загальні відомості про мову LD, поняття ланцюгів

Мова релейних діаграм LD (Ladder Diagram) або релейно-контактних схем (РКС) – графічна мова, реалізує структури електричних ланцюгів. На початку 70-х рр. XX ст. релейні автомати складальних конвеєрів почали поступово витіснятися програмованими контролерами. Деякий час ті й інші працювали водночас й обслуговувалися одними й тими самими людьми. Так з’явилася задача прозорого перенесення релейних схем у ПЛК. Різні варіанти програмної реалізації релейних схем створювалися практично всіма провідними виробниками ПЛК. Завдяки простоті подання РКС знайшов заслужену популярність, що й стало основною причиною включення його в стандарт МЕК.

Слова «релейна логіка» звучать сьогодні досить архаїчно, майже як «ламповий комп’ютер». Тим більше у зв’язку зі створенням численних швидкодіючих і надійних безконтактних (зокрема, оптоелектронних) реле і потужних перемикальних приладів, таких як потужні польові транзистори, керовані тиристори й прилади IGBT [--36]. Але, незважаючи на це, релейна техніка все ще дуже широко застосовується.

Релейна схема являє собою дві вертикальні шини живлення, між ними розташовані горизонтальні ланцюги, утворені контактами та обмотками реле. Кількість контактів у ланцюзі довільна, реле одне. Кожне реле має контакти, які можна використовувати в інших ланцюгах. Якщо послідовно з’єднані контакти замкнуті, струм йде ланцюгом і реле вмикається (у прикладі

на рис. 5.12 Lamp1). За необхідності можна підключити паралельно декілька реле, послідовне підключення не допускається.

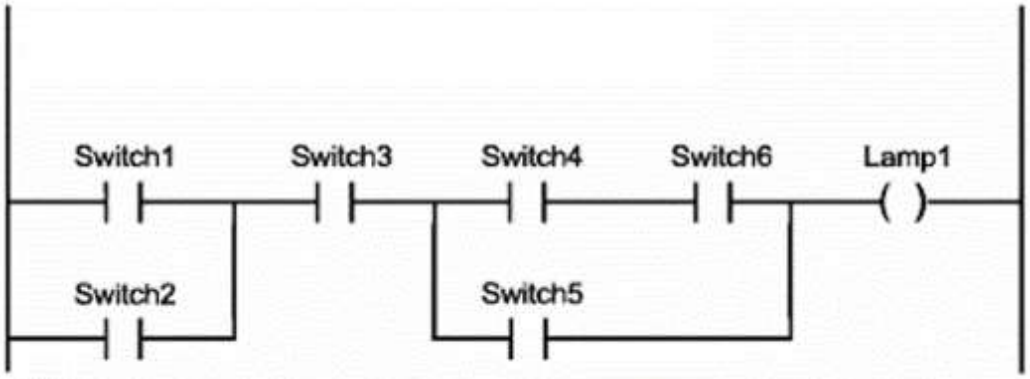


Рисунок 5.12 – Схема LD з одним ланцюгом

Логічно послідовне (І), паралельне (АБО) з'єднання контактів та інверсія (НІ) утворюють базис Буля. В результаті LD ідеально підходить не тільки для побудови релейних автоматів, а й для програмної реалізації комбінаційних логічних схем. Завдяки можливості підключення в LD функцій і функціональних блоків, виконаних іншими мовами, сфера застосування мови практично не обмежена.

У LD кожному контакту ставиться у відповідність логічна змінна, яка визначає його стан. Якщо контакт замкнутий, то змінна має значення ІСТИНА. Якщо розімкнутий – ХИБНІСТЬ. Ім'я змінної пишеться над контактом і фактично є його назвою.

Послідовне з'єднання контактів або ланцюгів рівноцінно логічній операції І. Паралельне з'єднання утворює монтажне АБО.

Ланцюг може бути або замкнутим (ON), або розімкнутим (OFF). Це саме й відображається на обмотці реле і відповідно на значенні логічної змінної обмотки (ІСТИНА / ХИБНІСТЬ).

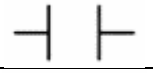
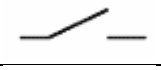
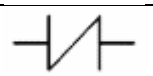
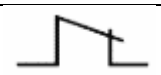
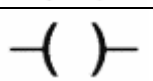
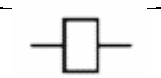
Наведена на рис. 5.12 схема еквівалентна виразу:

Lampl := (Switchl OR Switch2) AND Switch3 AND ((Switch4 AND Switch6) OR Switch5);

Зорове сприйняття LD-діаграм має бути інтуїтивно зрозумілим. Вітчизняним розробникам цього певною мірою перешкоджає прийнята система умовних графічних позначень, що базується на американському стандарті NEMA. Перевага таких позначень полягає в можливості застосування символів псевдографіки для побудови LD-діаграм.

Порівняння позначень базових елементів LD і позначень ЄСКД наведено в таблиці 5.11.

Таблиця 5.11 – Порівняння позначень базових елементів LD і позначень ЄСКД

LD	ЄСКД	Позначення
		Нормально розімкнутий контакт
		Нормально замкнутий контакт
		Обмотка реле

Контакт може бути інверсним – нормально замкнутим. Такий контакт позначається за допомогою символу |/| і замикається, якщо значення змінної ХИБНІСТЬ. Інверсний контакт рівнозначний логічній операції НІ.

Перемикаючий контакт утворюється комбінацією прямого й інверсного контактів (див. приклад на рис. 5.13).

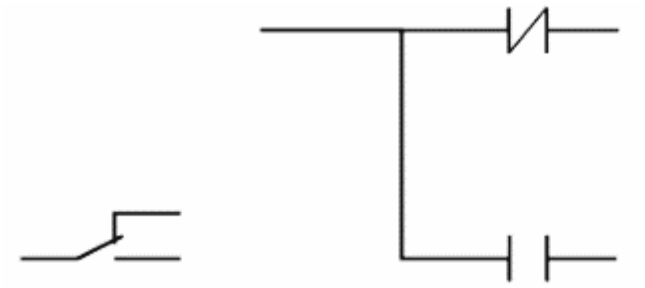


Рисунок 5.13 – Перемикаючий контакт

Обмотки реле також можуть бути інверсними, що позначається символом (/). Якщо обмотка інверсна, то у відповідну логічну змінну копіюється інверсне значення стану ланцюга.

5.4.2 Реле з самофіксацією

Крім звичайних реле, в релейних схемах часто застосовуються поляризовані реле. Таке реле має дві обмотки, які перемикають його з одного положення в інше. Перемикання проводиться імпульсами струму. З відключенням струму живлення поляризоване реле залишається в заданому положенні, що реалізує елементарну комірку пам’яті.

У LD таке реле реалізується за допомогою двох спеціальних обмоток SET і RESET. Обмотки типу SET позначаються літерою S всередині круглих

дужок (S). Обмотки типу RESET позначаються літерою R. Якщо відповідна обмотці (S) змінна приймає значення ІСТИНА, то зберігає його безкінечно. Повернути дану змінну в ХИБНІСТЬ можна тільки обмоткою (R).

Очевидно, що повної аналогії з поляризованим реле програмно досягти неможливо. Навіть якщо значення логічного виходу зберігається в енергонезалежній пам'яті, стан самого електричного ланцюга за вимкненого живлення ПЛК визначається його схематикою. Фіксація безпечного положення апаратури під час аварії живлення системи управління може бути досягнута тільки апаратно.

Умова вимкнення реле не завжди рівнозначна відсутності умови увімкнення. Завдяки (R) і (S) обмоткам умови увімкнення і вимкнення реле можна формувати абсолютно незалежно, причому в будь-якому ланцюзі і скільки завгодно разів. Обмотки (R) і (S) забезпечують фіксацію умов управління, що необхідно в ході реалізації автоматів з пам'яттю.

Самофіксацію нескладно організувати і на простому реле, використовуючи додатковий контакт у ланцюзі живлення. Приклад цього наведено на рис. 5.14.

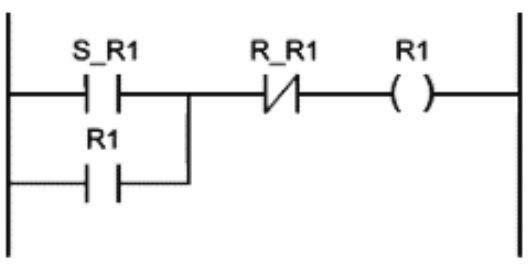


Рисунок 5.14 – Реле з самофіксацією

Контакт S_R1 вмикає, а R_R1 вимикає реле R1. Завдяки контакту R1 реле отримує живлення після розмикання S_R1. Застосування SET / RESET-обмоток не дає нічого принципово нового, але робить LD-діаграму простіше й красивіше.

5.4.3 Порядок виконання і зворотні зв'язки

Ідеологія релейних схем передбачає паралельну роботу всіх ланцюгів. Струм в усі ланцюги подається водночас.

У LD рішення діаграми виконується послідовно зліва направо і зверху вниз. У кожному робочому циклі одноразово виконуються всі ланцюги діаграми, що і створює ефект паралельності роботи ланцюгів. Будь-яка змінна

в рамках одного ланцюга завжди має одне й те саме значення. Якщо навіть реле в ланцюзі змінить змінну, то нове значення надійде на контакти тільки в наступному циклі. Ланцюги розташовані нижче, отримають нове значення змінної відразу. Ланцюги розташовані вище – тільки в наступному циклі. Суворий порядок виконання схеми дуже важливий. Випадковий або навіть істинно паралельний порядок виконання ланцюгів міг би приводити до ефекту «гонок», що зустрічається в електронних схемах з тригерами. Завдяки жорсткому порядку виконання LD-діаграми зберігають стійкість за наявності зворотних зв’язків.

У наведеній на рис. 5.15 схемі вмикання Key викличе миттєве (в тому ж циклі) вмикання P2 і вимикання P3. Реле P1 буде увімкнене тільки в наступному циклі, причому навіть якщо Key вже в обриві (ХИБНІСТЬ).

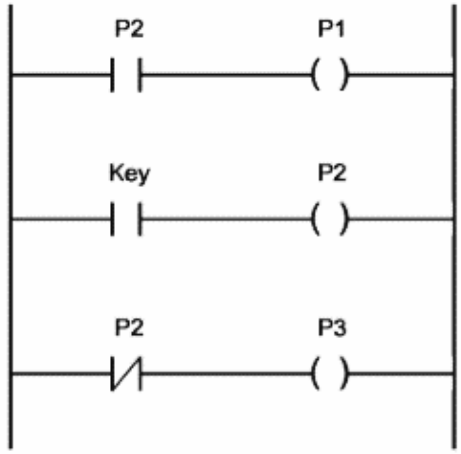


Рисунок 5.15 – LD-діаграма зі зворотним зв’язком

Використовуючи вищеописаний принцип циклічності виконання LD-діаграм, дуже легко побудувати генератор одиничних імпульсів. Приклад такої побудови показаний на рис. 5.16.

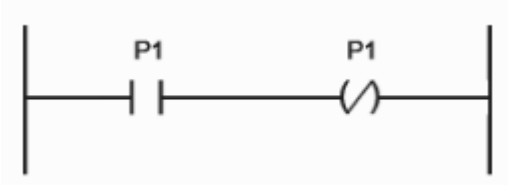


Рисунок 5.16 – Генератор одиничних імпульсів

Період імпульсів на реле P1 дорівнюватиме подвоєній тривалості робочого циклу ПЛК.

5.4.4 Управління порядком виконання

Порядок виконання ланцюгів діаграми можна примусово змінювати, використовуючи мітки (labels) і переходи (jumps).

Мітку можна ставити тільки в початок ланцюга. Імена міток підпорядковані правилам найменування змінних. Для наочності можна закінчити мітку двокрапкою. Двокрапка не утворює нової мітки. Таким чином, Ml: і Ml – це одне й те саме.

Ланцюг може мати тільки одну мітку і один перехід. Перехід рівнозначний вихідному реле і виконується, якщо вихідна змінна має значення ІСТИНА. Перехід може бути інверсним, у цьому випадку він виконується зі значенням ланцюга ХИБНІСТЬ. Використовуючи перехід, можна пропустити виконання частини діаграми. Пропущені ланцюги не скидаються, і не виконуються – залишаються в тому положенні, в якому були раніше. Перехід вгору допускається і дозволяє створювати цикли (рис. 5.17).

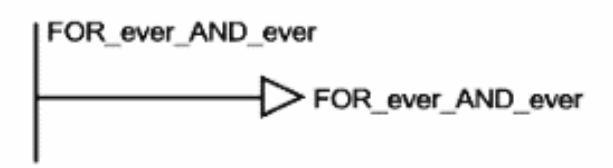


Рисунок 5.17 – Простий безкінечний цикл

Ідеологічно переходи суперечать аналогії LD з релейними схемами, порушуючи закони електричних ланцюгів. У схемі LD з переходами розібратися буває складно. Бажано не займатися управлінням порядком виконання LD-діаграми в ній самій, а використовувати для цього більш виразні засоби. Наприклад, розділити LD-діаграму на модулі (дії), а порядок виконання описати в SFC.

5.4.5 Розширення можливостей LD

У LD-діаграму можна вставити функції та функціональні блоки. Функціональні блоки повинні мати логічні вхід і вихід. На рис. 5.18 показаний приклад організації циклу на 10 повторів на базі функціонального блоку декрементний лічильник. Перший ланцюг завантажує лічильник кількістю повторів. Другий ланцюг – генератор одиничних імпульсів. Третій – декрементний лічильник з перевіркою умови закінчення циклу. Тіло циклу на рисунку не наведено.

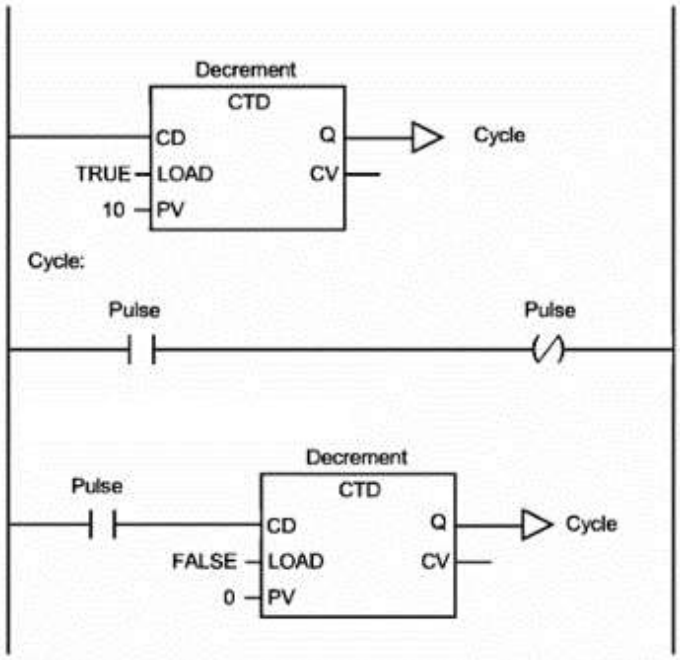


Рисунок 5.18 – Цикл на 10 повторів на базі функціонального блоку CTD

Для включення в діаграму функцій у них штучно вводиться додатковий логічний вхід, що позначається EN (Enable) (рис. 5.19). Логічне значення на вхід EN дозволяє або забороняє виконання функції. Сама функція не змінюється з додаванням входу EN.

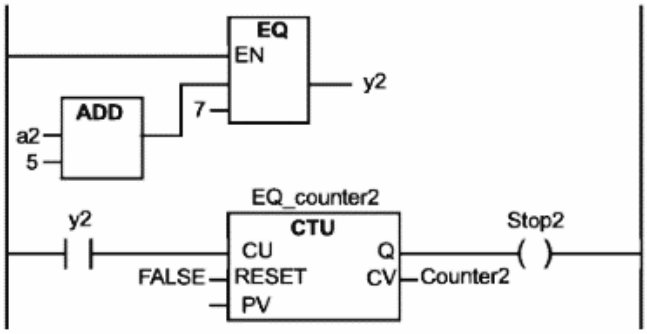


Рисунок 5.19 – Використання оператора EQ, який управляється входом EN

У першій редакції стандарт МЕК визначав контакти й обмотки, які управляються фронтами імпульсів: контакт |P| і обмотка (P) переднього фронту, контакт |N| і обмотка (N) заднього фронту. Зараз підтримка таких контактів і обмоток не є обов'язковою, тому що аналогічний ланцюг легко можна побудувати за допомогою функціональних блоків R_TRIG і F_TRIG.

5.4.6 Особливості реалізації LD в CODESYS

Неважко помітити, що довжина LD-ланцюгів різна залежно від складності. Наприклад, нехай перший ланцюг складається з 50 контактів, а

другий з двох. У цьому випадку шини живлення на схемі мають бути широко рознесені, щоб вмістити всі з'єднання першого ланцюга. Другий ланцюг виглядатиме невиправдано розтягнутим. Ця особливість РКС завжди викликала труднощі у реалізації станцій ПЛК, що програмують. Якщо ланцюг не вміщувався в один екран, його розривали і переносили залишок нижче. Перенесення легко читається для одного проводу, але ланцюг може бути розгалуженим.

Стандарт МЕК допускає не зображувати загальну праву шину взагалі і вирівнювати ланцюг вліво для кращого зорового сприйняття. Графічний редактор COSESYS не обмежує можливу ширину LD-ланцюга і не вимагає застосування переносів. Складні ланцюги зображуються злито і для роботи з ними необхідно користуватися горизонтальною прокруткою екрана (рис. 5.20). Права шина зображується вертикальними відрізками в межах кожного ланцюга і вирівнюється вліво, ланцюги пронумеровані й розділені горизонтальними лініями.

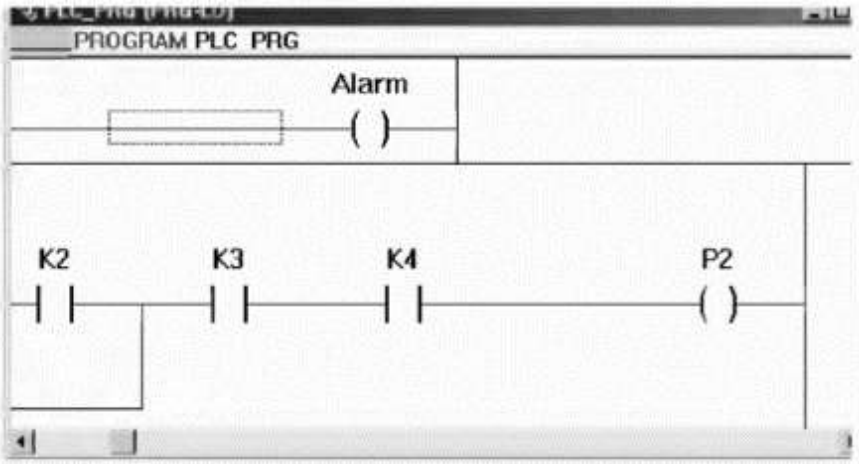


Рисунок 5.20 – Правий край двох ланцюгів різної довжини

Час виконання одного LD-ланцюга не постійний. Якщо процесор виявив, що ланцюг розімкнутий, то вже немає сенсу аналізувати його далі, можна відразу присвоїти значення FALSE вихідній змінній. Компілятор коду CODESYS так і робить.

Для функціональних блоків CODESYS дозволяє графічно підключати тільки один вхід у логічний ланцюг. Стандарт не накладає таких обмежень. Наприклад, у системі MULTIPROG перший ланцюг, показаний на рис. 5.18, може виглядати, як на рис. 5.21.

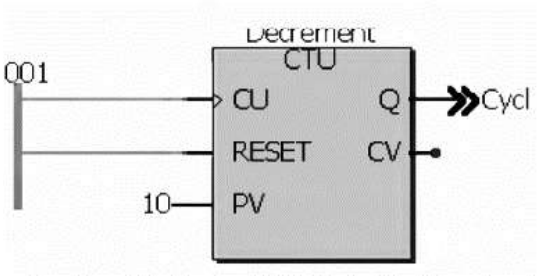


Рисунок 5.21 – Зображення декрементного лічильника в MULTIPROG

Вхід RESET тут отримує значення ІСТИНА безпосередньо від шини живлення. Така схема більш наочна, але прийнятна тільки для логічних змінних. Аналогові змінні все одно не вписуються в ідеологію LD і вимагають безпосереднього присвоювання.

Функція або функціональний блок у LD розглядаються як виконавчий пристрій – аналог реле. Застосування функції фактично означає кінець LD-діаграми і початок FBD. Правила побудови LD-діаграм тут вже не працюють.

Крім входу EN, стандарт пропонує визначати для функцій і додатковий вихід ENO (Enable Out), що показує подальше проходження струму в ланцюзі. Вихід ENO не є обов'язковим. У CODESYS така можливість не підтримується. Вихід ENO має слугувати для індикації помилок у функції. В CODESYS контроль помилок виконання реалізований інакше.

За визначенням функція має тільки один вихід. Завдяки цьому функції можна використовувати у виразах ST. Функції з додатковим виходом ENO вирішують одну проблему, але створюють іншу.

5.4.7 LD-діаграми в режимі виконання

У режимі Online обмотки реле, контакти і провідники, що знаходяться в стані On (під струмом) блакитного (кольори за замовчуванням) кольору. CODESYS дозволяє змінювати значення логічних змінних (ІСТИНА / ХИБНІСТЬ) безпосередньо в графічній діаграмі подвійним клацанням мишки на імені змінної. Значення входів-виходів функціональних блоків відображаються числовими значеннями.

Точка зупинника може встановлюватися тільки в цілому на ланцюг. Для установки або скидання точки зупинника необхідно клацнути кнопкою мишки по номеру ланцюга. У режимі зупинника номер ланцюга підсвічений червоним. Покроково – по одному ланцюгу виконання досягається командами «Step over» і «Step in».

5.5 Функціональні діаграми FBD

5.5.1 Відображення POU

FBD (Function Block Diagram) – це графічна мова програмування. Діаграма FBD дуже нагадує принципову схему електронного пристрою на мікросхемах. На відміну від LD, «провідники» у FBD можуть проводити сигнали (передавати змінні) будь-якого типу (логічний, аналоговий, час тощо). Іноді говорять, що в релейних схемах з'єднувальні провідники передають енергію. Провідники FBD теж передають енергію, але в більш широкому сенсі. Очевидно, що шини живлення і контакти тут вже не ефективні. Шини живлення на FBD-діаграмі не відображаються.

Виходи блоків можуть бути подані на входи інших блоків або безпосередньо на виходи ПЛК. Самі блоки, подані на схемі як «чорні ящики», можуть виконувати будь-які функції.

FBD-схеми дуже чітко відображають взаємозв'язок входів і виходів діаграми. Якщо алгоритм від початку добре описується з позиції сигналів, то його FBD-подання завжди виходить наочніше, ніж текстовими мовами.

Діаграма FBD будується з компонентів, що відображаються на схемі прямокутниками. Входи POU зображуються зліва від прямокутника, виходи справа. Усередині прямокутника вказується тип POU і найменування входів і виходів. Для екземпляра функціонального блоку його найменування вказується у верхній частині, над прямокутником.

У графічних системах програмування прямокутник компонента може містити зображення, що відображає його тип. Розмір прямокутника залежить від кількості входів і виходів і встановлюється графічним редактором автоматично.

Приклад графічного подання екземпляра Blinker функціонального блоку BLINK дано на рис. 5.22.

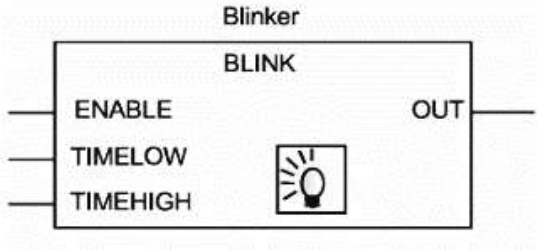


Рисунок 5.22 – Графічне подання екземпляра функціонального блоку

Програма у FBD не обов'язково має представляти велику єдину схему. Як і в LD, діаграма утворюється з ланцюгів, які виконуються один за одним.

У CODESYS усі ланцюги одного POU відображаються в єдиному графічному вікні, пронумеровані і розділені горизонтальними лініями (рис. 5.23). Значення змінних, які обчислені в одному ланцюзі, доступні в наступних ланцюгах відразу в тому самому робочому циклі.

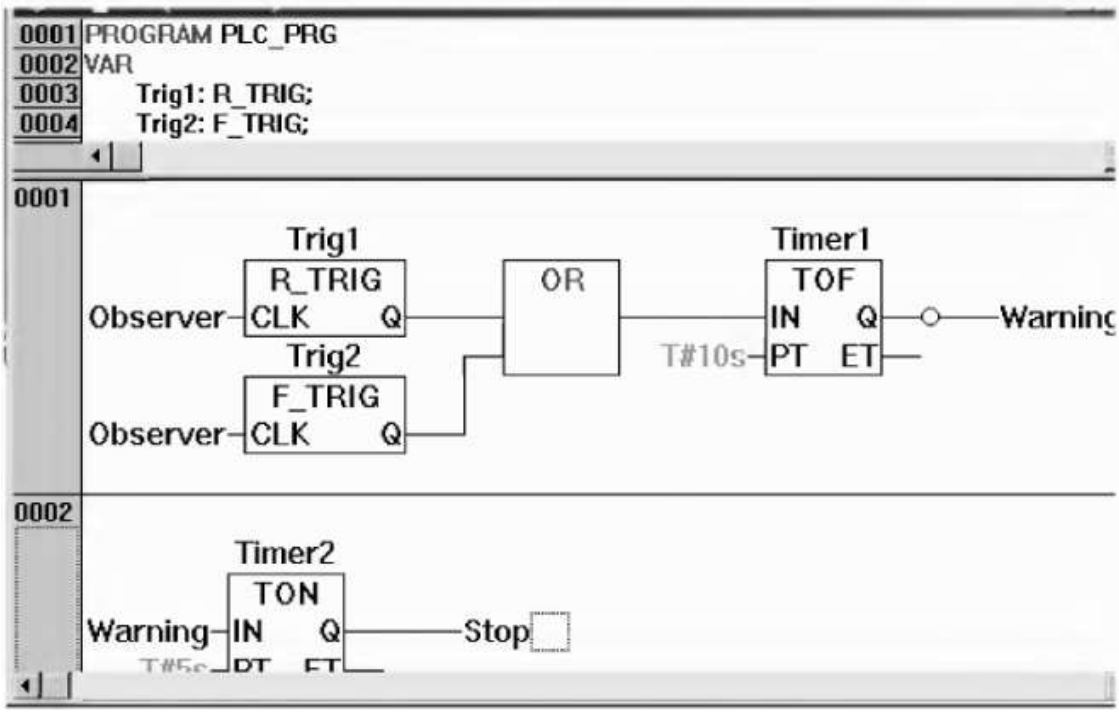


Рисунок 5.23 – Діаграма FBD з двох ланцюгів

5.5.2 З'єднувальні лінії

Прямокутники POU у FBD з'єднані лініями зв'язку. З'єднання мають спрямованість зліва направо. Вхід блоку може бути з'єднаний з виходом блоку, розташованим зліва від нього. Крім цього, вхід може бути з'єднаний зі змінною або константою. З'єднання має пов'язувати змінні або входи й виходи одного типу.

На відміну від компонента, змінна зображується на діаграмі без прямокутної рамки. Ширина сполучної лінії в FBD не важлива. Стандарт допускає використання з'єднувальних ліній різної ширини і стилю для з'єднань різного типу.

5.5.3 Порядок виконання FBD

Виконання FBD-ланцюгів йде зліва направо, зверху вниз. Блоки, розташовані лівіше, виконуються раніше. Блок починає обчислюватися тільки після обчислення значень усіх його входів. Подальших обчислень не буде, доки

не будуть обчислені значення на всіх виходах. Іншими словами, значення на всіх виходах графічного блоку з'являються водночас. Обчислення ланцюга вважається закінченим тільки після обчислення значень на виходах усіх елементів, які входять до нього.

У деяких системах програмування користувач має можливість вільно пересувати блоки зі збереженням зв'язків. У цьому випадку орієнтуватися потрібно, виходячи з порядку з'єднань. Редактор FBD COSESYS автоматично розставляє блоки в порядку виконання.

5.5.4 Інверсія логічних сигналів

Інверсія логічного сигналу в FBD зображується у вигляді кола на з'єднанні, перед входом або змінною. Інверсія не є властивістю самого блоку і може бути легко додана або скасована безпосередньо в діаграмі. У CODESYS це робиться командою «Negate». На рис. 5.23 вихід Q екземпляра функціонального блоку TOF інвертується перед присвоєнням його значення змінній Warning.

5.5.5 З'єднувачі і зворотні зв'язки

З'єднувачі (connectors) являють собою пойменоване з'єднання, яке можна розірвати і перенести в наступний ланцюг. Такий прийом може знадобитися за обмеженої ширини вікна редактора FBD. У CODESYS ширина вікна не обмежена, тому з'єднувачі тут не потрібні.

Стандарт не забороняє з'єднання, що надходять з виходу блоку на свій вхід або вхід раніше виконуваних блоків. Зворотний зв'язок не утворює цикл, подібний FOR, просто деяке обчислене значення надійде на вхід з наступним викликом діаграми. Фактично це означає неявне створення змінної, яка зберігає своє значення між викликами діаграми.

В редакторі FBD CODESYS зворотні з'єднання заборонені. Для створення зворотного зв'язку використовуйте явно оголошену внутрішню змінну.

За необхідності перенесення або розгалуження з'єднання в інші ланцюги також необхідно використовувати проміжні локальні змінні.

5.5.6 Мітки, переходи і повернення

Порядок виконання FBD-ланцюгів діаграми можна примусово змінювати, використовуючи мітки й переходи, так само, як і в релейних схемах.

Мітка ставиться на початку будь-якого ланцюга, і є назвою даному ланцюгу. Ланцюг може містити тільки одну мітку. Імена міток підпорядковані загальним правилам найменування ідентифікаторів МЕК. Графічний редактор автоматично нумерує ланцюги діаграми. Ця нумерація застосовується виключно для документування і не може замінювати мітки.

Перехід обов’язково пов’язаний з логічною змінною і виконується, якщо змінна має значення ІСТИНА. Для створення безумовного переходу використовується константа ІСТИНА, яка пов’язана з переходом. Мітки та переходи в FBD наведені у прикладі на рис. 5.24.

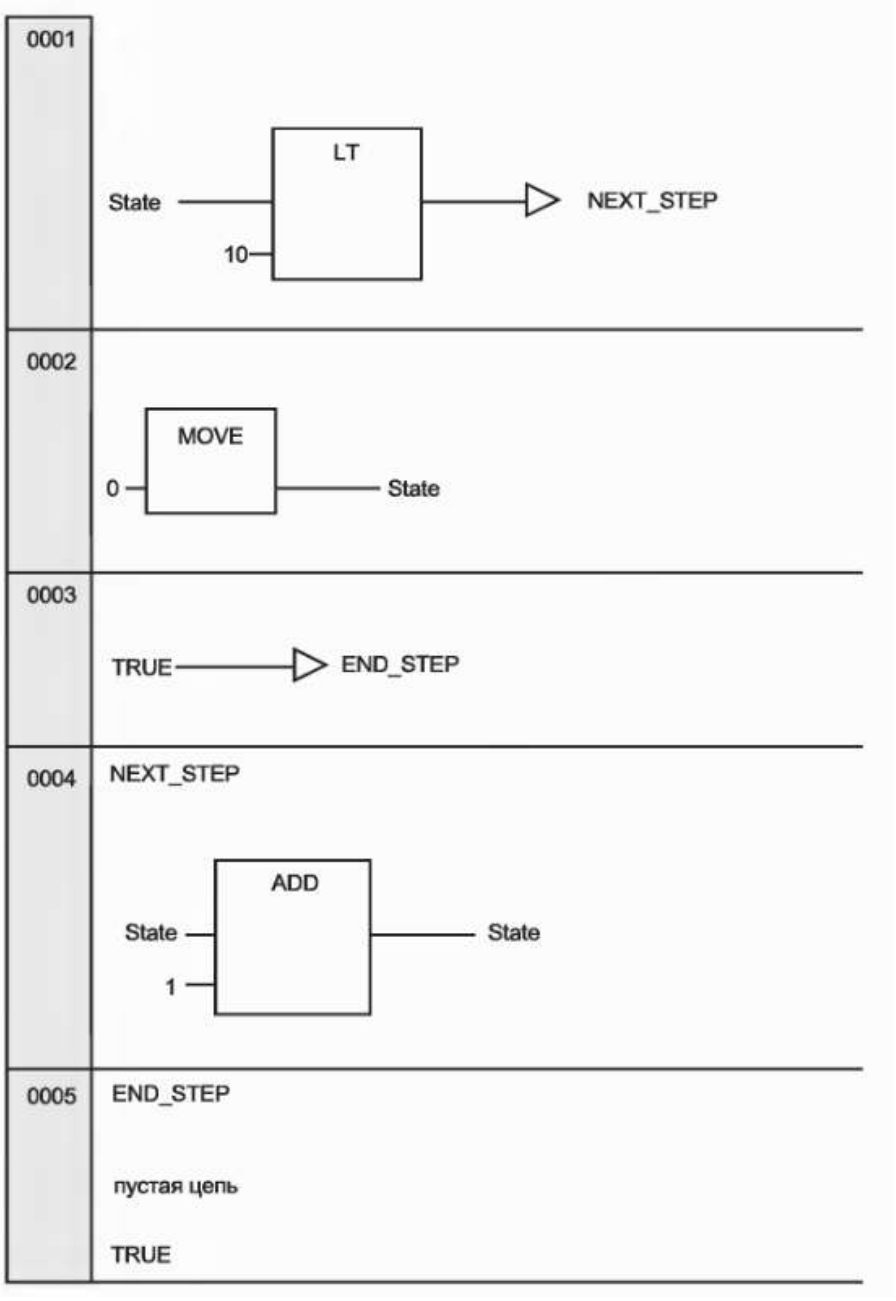


Рисунок 5.24 – Мітки та переходи в FBD

Зверніть увагу на останній ланцюг на рис. 5.24 – він є порожнім. Порожній ланцюг позначається єдиною константою TRUE.

Оператор повернення RETURN можна використовувати в FBD так само, як і перехід на мітку, тобто у зв'язці з логічною змінною. Повернення призводить до негайного закінчення роботи програмного компонента і повернення на верхній рівень вкладень. Для основної програми – це початок робочого циклу ПЛК.

5.5.7 Вирази ST в FBD

COSEDYS дозволяє записувати вирази ST на вході графічних блоків. Такий прийом розширює стандартний FBD і часто виявляється досить зручним. Компактна форма подання слів полегшує запис і читання функціональних діаграм (рис. 5.25).

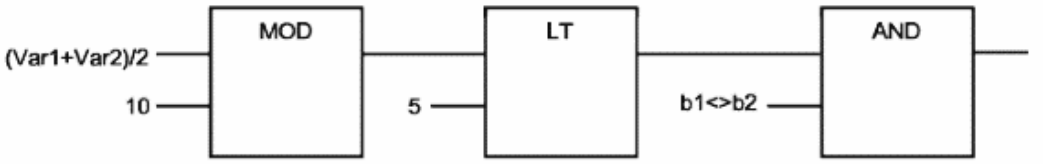


Рисунок 5.25 – Фрагмент FBD-діаграми з виразами

5.6 Контрольні запитання та завдання

1. Які типи даних МЕК ви можете навести?
2. Для чого призначені структури?
3. Що таке змінні? Як розподіляється пам'ять змінних?
4. Чим відрізняється пряма від порозрядної адресації?
5. Що таке функція?
6. Як створити екземпляр функціонального блоку?
7. Як здійснюється виклик екземпляра блоку?
8. Які особливості реалізації та застосування функціональних блоків?
9. Поясніть роботу ПЛК, як кінцевого автомату.
10. Що таке структурований текст?
11. Наведіть загальні відомості про мову LD.
12. Дайте визначення поняттю ланцюгів.
13. Поясніть особливості реалізації LD в CODESYS.
14. Як використовуються функціональні діаграми FBD?

6 ПРИКЛАДИ ВИРІШЕННЯ ПРАКТИЧНИХ ЗАДАЧ

6.1 Налаштування та підготовка до роботи інтегрованого середовища розробки CODESYS v2.3

Як було зазначено раніше, CODESYS – це сучасний інструмент для програмування контролерів, який надає програмісту зручне середовище для програмування мовами стандарту МЕК 61131-3. Редактори і налагоджувальні засоби, що використовуються, базуються на широко відомих принципах, які добре себе зарекомендували в інших популярних середовищах професійного програмування.

Проект включає такі об'єкти: POU, типи даних, візуалізації, ресурси, бібліотеки. Кожен проект зберігається в окремому файлі.

До *програмних компонентів* (POU – Program Organization Unit) належать функціональні блоки, функції та програми. Окремі POU можуть включати дії (підпрограми).

Кожен програмний компонент складається з розділу оголошень і коду. Для написання всього коду POU використовується тільки одна з МЕК мов програмування (IL, ST, FBD, SFC, LD або CFC).

CODESYS підтримує всі описані стандартом МЕК компоненти. Для їх використання достатньо включити в свій проект бібліотеку standard.lib.

POU можуть викликати інші POU, але рекурсії неприпустимі.

Функція – це POU, що повертає тільки єдине значення (яке може складатися з декількох елементів, якщо це бітове поле або структура). У текстових мовах функція викликається як оператор і може входити у вираз.

З оголошенням функції необхідно вказати тип значення, що повертається. Для цього після імені функції потрібно написати двокрапку і тип. Правильно оголошена функція має такий вигляд:

```
FUNCTION Fet: INT;
```

Ім'я функції використовується як вихідна змінна, якій присвоюється результат обчислень.

Оголошення функції має починатися з ключового слова FUNCTION і закінчуватися ключовим словом END_FUNCTION. На рис. 6.1 показаний

приклад функції, написаної на IL, яка використовує три вхідних змінних і повертає результат ділення добутку перших двох на третю.

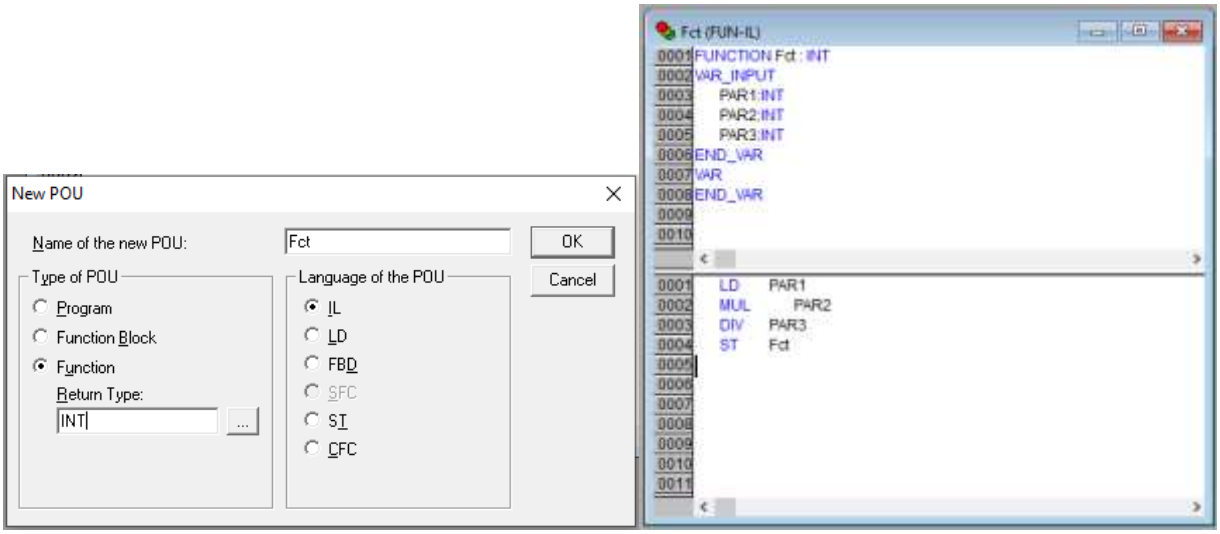


Рисунок 6.1 – Приклад функції, написаної на IL

У мові ST виклик функції може бути присутнім у виразах як операнд.

У SFC функція викликається тільки з кроку або переходу.

Функція не має внутрішньої пам'яті. Це означає, що функція з одними і тими самими значеннями вхідних змінних завжди повертає одне й те саме значення.

На рис. 6.2 показаний приклад використання функції різними мовами програмування.

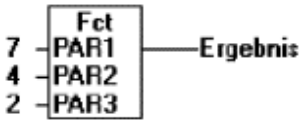
Мова програмування IL	Мова програмування ST	Мова програмування FBD
LD 7 Fct 2,4 ST Result	Result := Fct(7,2,4);	

Рисунок 6.2 – Приклад використання функції
різними мовами програмування

Функціональний блок – це POU, який приймає і повертає довільне число значень. На відміну від функції, функціональний блок не формує значення, яке повертається. Оголошення функціонального блоку починається з ключового слова FUNCTION_BLOCK і закінчується ключовим словом END_FUNCTION_BLOCK.

На рис. 6.3 наведено приклад функціонального блоку, написаного на IL, який має дві вхідних і дві вихідних змінних. Значення вихідних змінних MULERG дорівнює добутку значень двох вхідних змінних, а значення VERGL визначається в результаті порівняння значень вхідних змінних.

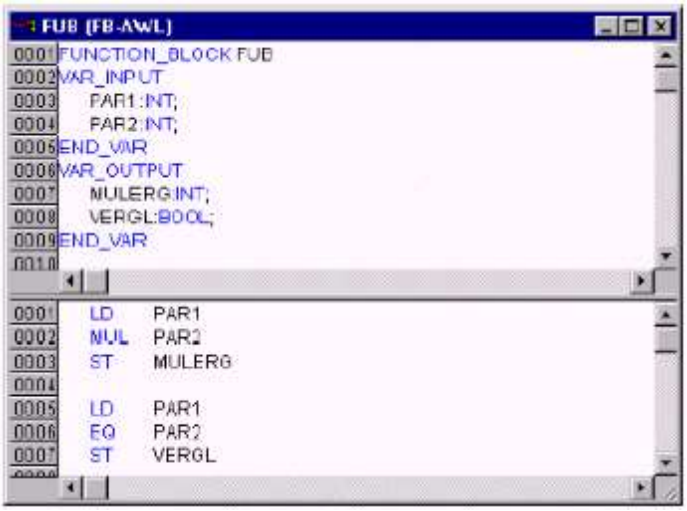


Рисунок 6.3 – Приклад функціонального блоку

Визначення функціонального блоку подібно визначенню типу даних. Для роботи з функціональним блоком необхідно оголосити (створити) його екземпляр. Один функціональний блок може мати довільну кількість екземплярів, кожен з яких має власні незалежні дані (пам’ять).

Кожен екземпляр функціонального блоку отримує свій власний ідентифікатор (ім’я екземпляра) і свої дані, що містять вхідні, вихідні та внутрішні змінні. Примірники функціонального блоку оголошуються глобально або локально як змінні, що мають тип відповідного функціонального блоку. Приклад оголошення екземпляра з ідентифікатором INSTANCE функціонального блоку FUB:

INSTANCE: FUB;

Виклик екземпляра функціонального блоку відбувається за допомогою його імені. Вхідні й вихідні змінні доступні поза функціонального блоку, а внутрішні змінні доступні тільки в самому блоці.

На рис. 6.4 наведено приклад використання екземпляра функціонального блоку. На даному рисунку показано створення екземпляра функціонального блоку fb. Екземпляр оголошується з ідентифікатором inst1. Функціональний блок fb має вхідну змінну int типу INT.

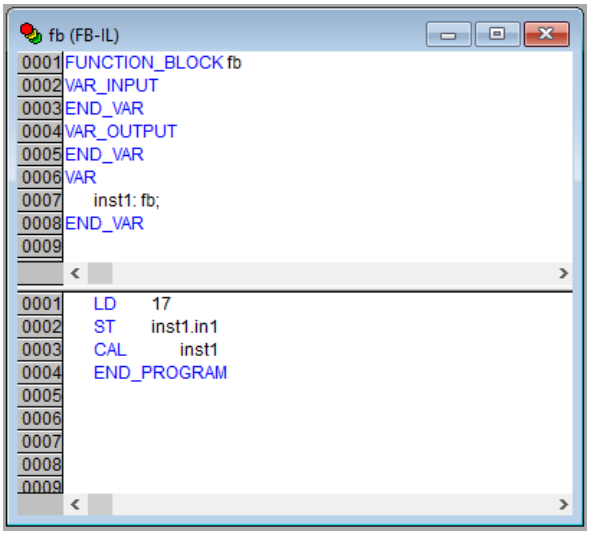


Рисунок 6.4 – Приклад використання екземпляра функціонального блоку

Під час роботи в середовищі CODESYS він автоматично пропонує асистент вибору типів змінних, або способів опису функціональних блоків після натискання кнопки ENTER в кінці рядка.

На рис. 6.5 показаний приклад і порядок дій під час роботи з асистентом вводу екземпляра функціонального блоку.

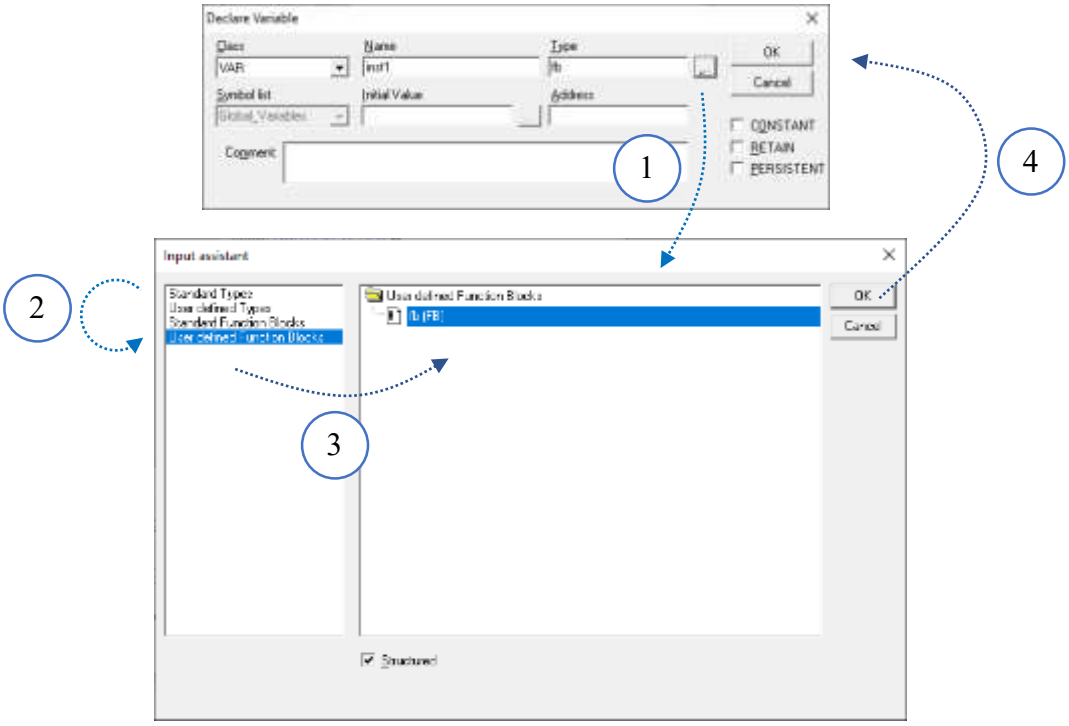


Рисунок 6.5 – Приклад і порядок дій під час роботи з асистентом вводу екземпляра функціонального блоку

Екземпляри функціонального блоку можуть бути оголошені в іншому функціональному блоці або у програмі. Оголошувати екземпляр функціонального блоку в тілі функції не можна. Примірники функціонального блоку доступні в тому POU, в якому вони оголошені, якщо вони не оголошені глобально. Екземпляри функціональних блоків можуть бути використані як вхідні змінні інших функціональних блоків або функцій.

Після виконання функціонального блоку всі його змінні зберігаються до наступного виконання. Отже, функціональний блок, що викликається з одними і тими самими вхідними параметрами, може виробляти різні вихідні значення.

Якщо хоча б одна змінна функціонального блоку оголошена як RETAIN, то всі дані екземплярів повністю вміщуються в енергонезалежний сегмент.

Виклик функціонального блоку

Для звернення до вхідних і вихідних змінних функціонального блоку ззовні необхідно вказати ім’я екземпляра функціонального блоку, наступної за нею крапкою та ім’ям змінної:

<Ім’я екземпляра>.< Ім’я змінної >

У текстових мовах (IL, ST) задати актуальні параметри і зчитати значення виходів можна безпосередньо з викликом екземпляра функціонального блоку. Для вхідних змінних застосовується присвоювання виду «:=», а виходи зчитуються за допомогою «=>». Цей процес спрощується, якщо використовувати асистент вводу (<F2>) з увімкненою опцією вставки з аргументами (With arguments).

Наприклад, припустимо, FBINST – це локальна змінна типу функціональний блок, що має вхідну змінну *xx* і вихідну змінну *yy*. Із вставкою FBINST у ST за допомогою асистента вводу отримуємо:

FBINSTl(xx:=, yy=>);

Необхідно звернути увагу, що змінні вхід-вихід (VAR_IN_OUT) передаються в екземпляр функціонального блоку через покажчики. Тому таким змінним можна присвоювати константи під час виклику.

Наприклад:

```
VAR
    inst: fubo;
    var: int;
END_VAR
```

varl := 2;

inst(instout1 := var1^);

Не можна привласнювати константи так:

inst(instout1 :=2);

або

inst.inout1 :=2;

Приклад виклику екземпляра вищеописаного функціонального блоку FUB наведено на рис. 6.6. Результат множення зберігається у змінній ERG, а результат порівняння у змінній QUAD. Екземпляр функціонального блоку FUB називається INSTANZ.

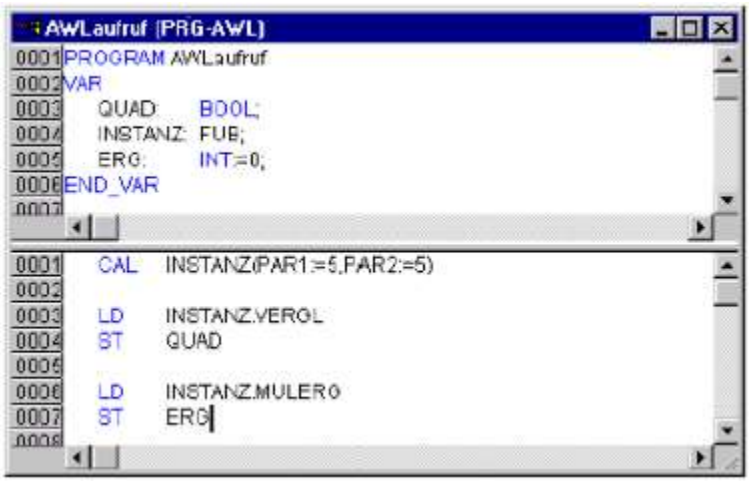


Рисунок 6.6 – Приклад виклику екземпляра функціонального блоку FUB

Приклад виклику екземпляра функціонального блоку в ST показаний на рис. 6.7.

Рисунок 6.7 – Приклад виклику екземпляра функціонального блоку в ST

Приклад виклику екземпляра функціонального блоку в FBD показаний на рис. 6.8.

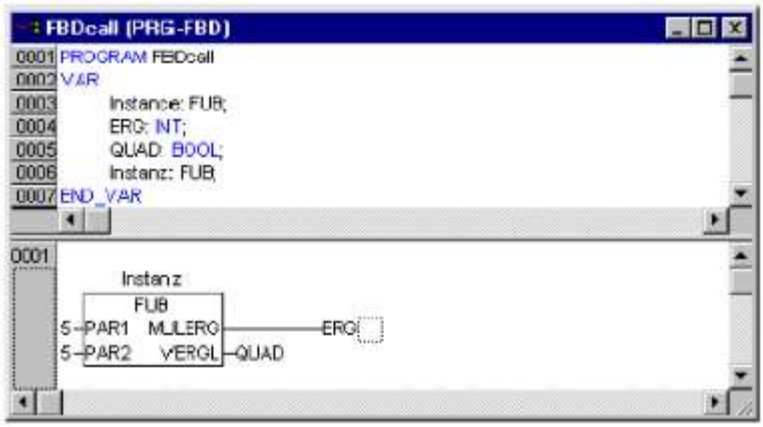


Рисунок 6.8 – Приклад виклику екземпляра функціонального блоку в FBD

Програма – це POU, здатний формувати довільне значення під час обчислень. Значення усіх змінних програми зберігаються між викликами. На відміну від функціонального блоку, екземплярів програми не існує. Програма є глобальною у всьому проекті.

Приклад програми наведено на рис. 6.9.

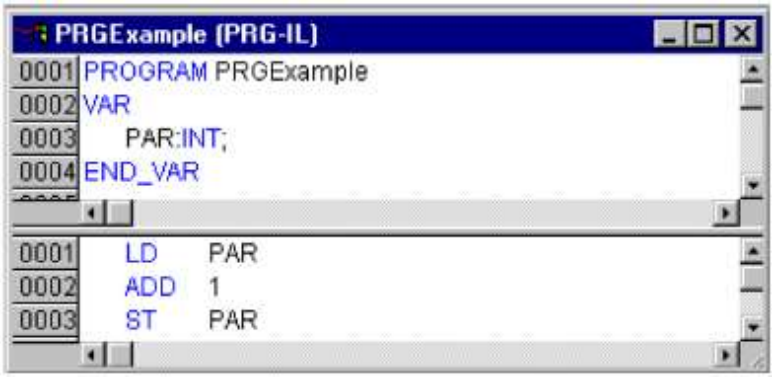


Рисунок 6.9 – Приклад програми

Не можна викликати програму з функції. Якщо викликати програму, яка змінить значення своїх змінних, то з наступним викликом її змінні матимуть ті самі значення, навіть якщо вона викликана з іншого POU. У цьому полягає головна відмінність між програмою і функціональним блоком, в якому змінюються тільки значення змінних даного екземпляра функціонального блоку.

Оголошення програми починається ключовим словом PROGRAM і закінчується ключовим словом ENDPROGRAM.

Так само, як і для екземплярів функціональних блоків, у текстових мовах (IL, ST) задати актуальні параметри і зчитати значення виходів можна безпосередньо під час виклику програми.

Для вхідних змінних застосовується присвоювання виду «: =», а виходи зчитуються за допомогою «=>».

Приклади виклику програми для різних мов наведені нижче.

Для мови IL:

```
CAL PRGexample2
LD PRGexample2.out_var
ST ERG
```

Або з присвоюванням параметрів:

```
CAL PRGexample2(in_var:=33, out_var=>erg );
```

Для мови ST:

```
PRGexample2;
Erg :=PRGexample2.out_var;
```

Або з присвоюванням параметрів:

```
PRGexample2(in_var:=33, out_var=>erg );
```

Для мови FBD:

PLC PRG

Програма PLC_PRG – це спеціальний POU, який має бути в кожному проекті. Ця програма викликається один раз за цикл управління.

Зі створенням нового проекту автоматично відкривається діалог «Project» «Object Add», що пропонує створити нову POU-програму з ім’ям PLC_PRG.

Не слід змінювати запропоновані установки.

Якщо визначити послідовність виконання завдань у Task Configuration, то проект може не містити PLC_PRG.

Не можна видаляти або перейменовувати POU PLC_PRG (якщо Task Configuration не використовується). PLC_PRG є головною програмою в однозадачному проекті.

Дія

Програми або функціональні блоки можуть бути доповнені діями. Фактично дії – це додатковий набір вбудованих у POU підпрограм. Дії можуть описувтися мовою, відмінною від тієї, якою виконується відповідний функціональний блок або програма.

Дія оперує з тими самими даними, що й функціональний блок або програма, до якої вона належить. Приклад дії функціонального блоку показаний на рис. 6.10.

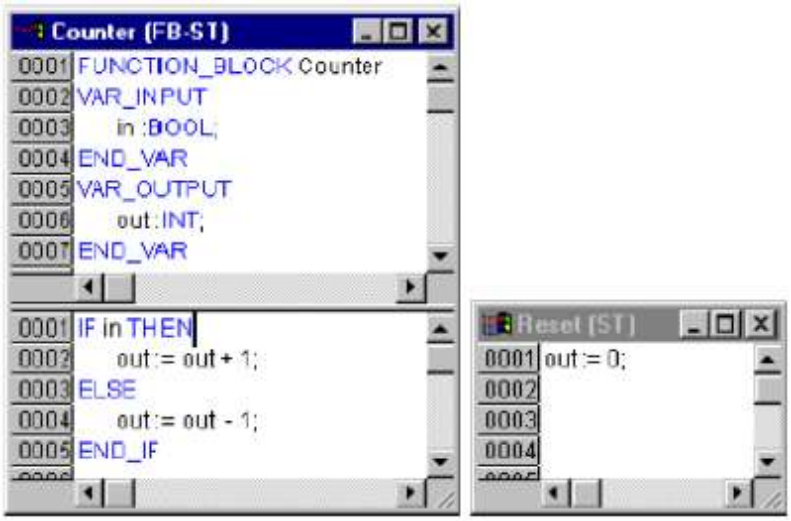


Рисунок 6.10 – Приклад дії функціонального блоку

У даному прикладі функціональний блок Counter збільшує або зменшує вихідну змінну «out» залежно від значення входу «in». З викликом дії Reset вихідна змінна приймає значення 0. Одна й та сама змінна «out» використовується в обох випадках.

Дія викликається за допомогою ідентифікатора:

<Ім’я_програми>.<Ім’я_дії>

або

<Ім’я_екземпляра>.<Ім’я_дії >

Приклад виклику вищеописаної дії наведено на рис. 6.11.

```
        PROGRAM PLC_PRG
        VAR
           Inst :  Counter;
        END_VAR
IL:
        CAL  Inst.Reset(In : = FALSE)
        LD    Inst.out
        ST    ERG
ST:
        Inst.Reset(In := FALSE);
        Erg := Inst.out;
FBD:
```

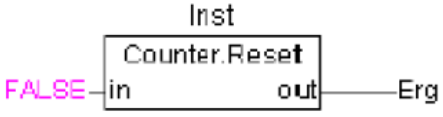


Рисунок 6.11 – Приклад виклику вищеописаної дії

Якщо потрібно викликати дію з POU, до якого воно належить, то в текстових мовах використовується ім’я дії, а в графічних – функціональний блок без вказівки імені екземпляра.

Ресурси

Ресурси відповідають за конфігурацію проекту, включаючи:

– глобальні змінні, які використовуються в усьому проекті;

– менеджер бібліотек (Library manager) для підключення необхідних бібліотек до проекту;

– журнал запису дій під час виконання;

– конфігуратор тривог (Alarm Configuration) для конфігурації обробки тривог у проекті;

– конфігуратор ПЛК (PLC Configuration) для конфігурації апаратури контролера;

– конфігуратор задач (Task Configuration) для управління задачами;

– менеджер рецептів (Watch and Receipt Manager) для перегляду і замовлення наборів значень змінних;

– опції цільової системи (Target Settings);

– робочу область для відображення опцій проекту.

Залежно від системи виконання та її опцій можуть підключатися додаткові об’єкти:

– sampling Trace – для задання графічного трасування значень змінних;

– parameter Manager – для взаємодії з іншими контролерами у мережі;

– PLC-Browser – монітор ПЛК;

– Tools – для виклику зовнішніх, специфічних для кожної платформи інструментів;

– SoftMotiou – компоненти системи управління рухом (відповідно до ліцензії), редактори CNC і CAM.

Бібліотеки

Проект може використовувати декілька бібліотек, в які входять POU, необхідні їм типи даних і глобальні змінні. Бібліотечні POU можна використовувати так само, як і визначенні користувачем.

Бібліотеки «standard.lib» і «util.lib» обов’язково входять до стандартного комплекту поставки.

Типи даних

Крім стандартних типів даних, ви можете використовувати визначені користувачем типи даних. Ними можуть бути структури, переліки та посилання.

Візуалізація

За допомогою візуалізації користувач може створити графічне подання проекту. Форма і колір графічних елементів будуть змінюватися під час роботи програми залежно від значень змінних.

Візуалізація може здійснюватися в системі програмування, в окремому додатку CODESYS HMI або як Web або цільова (в ПЛК) візуалізація.

6.2 Особливості налаштування і роботи інтегрованого середовища розробки CODESYS v3.x

6.2.1 Особливості середовища програмування CODESYS 3

У новому середовищі програмування CODESYS 3 реалізовано більше сотні різних нововведень. До них належать:

– можливість «згорнути» в рядок найскладніші конструкції мовою програмування ПЛК – ST (VAR …END_VAR тощо);

– можливість моніторингу виразів на предмет проміжних значень;

– реалізація функцій, що відповідають стандарту МЕК 61131 (IEC 61131) зі змінною кількістю параметрів і багато інших.

Проте серед усіх наявних нововведень є п’ять основних, які виділяють середовище програмування CODESYS 3 на фоні інших програм для роботи з ПЛК:

– розширення МЕК об’єктно-орієнтовані;

– можливість побудови мережних структур ПЛК;

– профілювання версій;

– захист програм CODESYS;

– єдина платформа для побудови систем автоматизації (АСУ ТП).

В оновленому середовищі програмування CODESYS оптимізовані основні особливості програмування, орієнтованого на створення різних об’єктів: класи, поліморфізм, інтерфейси, динамічне зв’язування, процес успадкування.

Компанії 3S вдалося розширити стандарт МЕК 61131-3 (IEC 61131-3), не змінюючи його, за рахунок введення в нього додаткового набору функцій.

У середовищі програмування CODESYS була впроваджена можливість адміністрування усіх промислових контролерів (ПЛК), об’єднаних в одному проекті в єдину мережну структуру.

Зміни не обійшли стороною і конфігуратор мережі. Тепер він може здійснювати налаштування не тільки інтелектуальних, але й пасивних пристроїв і елементів (різні датчики і виконавчі механізми).

Також оновлене середовище програмування CODESYS 3 може створювати безпечні системи на ПЛК, до складу яких входять – спеціальний редактор, система виконання і відповідний редактору компілятор. Всі компоненти проходять обов’язкову сертифікацію на предмет відповідності стандарту МЕК 61508, SIL 3.

Контролеру безпеки і стандартному ПЛК тепер набагато простіше обмінюватися між собою даними, оскільки працюють на одній і тій самій апаратній платформі, а обмін даними відбувається за допомогою PROFI-Safe.

Структура компонентів середовища програмування CODESYS 3 дозволяє зібрати всі необхідні інструменти в одному розширюваному середовищі, який у подальшому подається кінцевому користувачеві. Дану систему можна легко розширити за допомогою додаткових бібліотек CODESYS.

Всі функції системи строго задокументовані й доповнені шаблонами, завдяки яким досвідчений користувач може самостійно розширити систему, створити необхідні компоненти, які будуть забезпечені захистом авторських прав. При цьому зведено до мінімуму можливий загальний вплив компонентів і прояв побічних ефектів від даного впливу. Завдяки цьому розробники мають можливість здійснення своєї роботи швидко й якісно.

6.2.2 Особливості установки ПЗ CODESYS

CODESYS 3.x для своєї роботи потребує наявність Microsoft framework версії 3.5 і вище. Тому перед початком роботи з CODESYS необхідно встановити Microsoft framework на ПК. Якщо ПК має підключення до мережі Internet, то ці програми буде завантажено автоматично під час установки CODESYS. Також вони є для скачування на сайті виробника www.microsoft.ru.

Для установки ПЗ CODESYS 3 слід запустити програму-інсталятор. Безкоштовні оновлення версій ПЗ CODESYS доступні на сайтах www.codesys.ru, www.3s-software.com и www.owen.ru.

Після інсталяції ПЗ CODESYS 3 слід виконати інсталяцію Target-файлів. У Target-файлах міститься інформація про ресурси програмованих контролерів, з якими працює CODESYS. Target-файл поставляється виробником контролера.

Для кожної модифікації необхідний свій target-файл. Завантажити їх можна з сайту компанії (http://www.owen.ru/catalog/32050189 для ПЛК3xx, http://www.owen.ru/catalog/17844335 для СПК2xx):

– ПЛК 304/ПЛК308 – owen_plc30x.devdesc.xml;

– СПК207-03.CS – owen_spc207_x.03.x-cs.devdesc.xml;

– СПК207-03.CS-WEB – owen_spc207_x.03.x-cs-web.devdesc.xml;

– СПК207-04.CS – owen_spc207_x.04.x-cs.devdesc.xml;

– СПК207-04.CS-WEB – owen_spc207_x.04.x-cs-web.devdesc.xml.

Установка Target-файлу здійснюється в програмному середовищі CODESYS 3. Для цього перед створенням нового проекту оберіть команду «Tools | Device repository ... » головного меню ПЗ CODESYS, як це показано на рис. 6.12.

Рисунок 6.12 – Вибір пункту Device repository головного меню CoDeSys

Після вибору команди «Tools | Device repository ... » з'являється діалогове вікно, вид якого наведено на рис. 6.13.

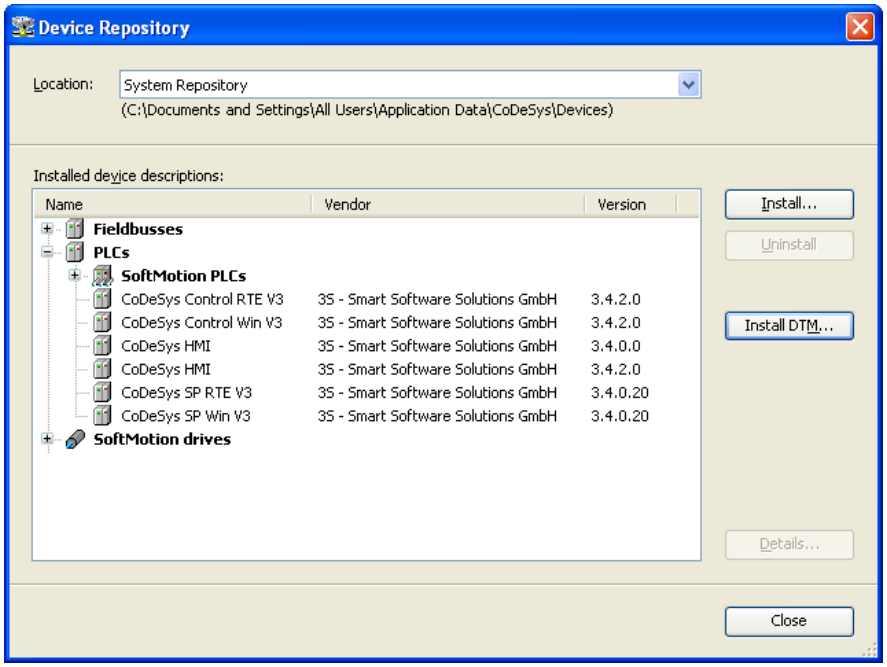


Рисунок 6.13 – Діалогове вікно установки Target-файлу в базу пристроїв (Device repository)

Для установки target-файлу у вікні «Device repository ...» натиснути кнопку «Install ...». У діалоговому вікні обрати необхідний target-файл, як це показано на рис. 6.14.

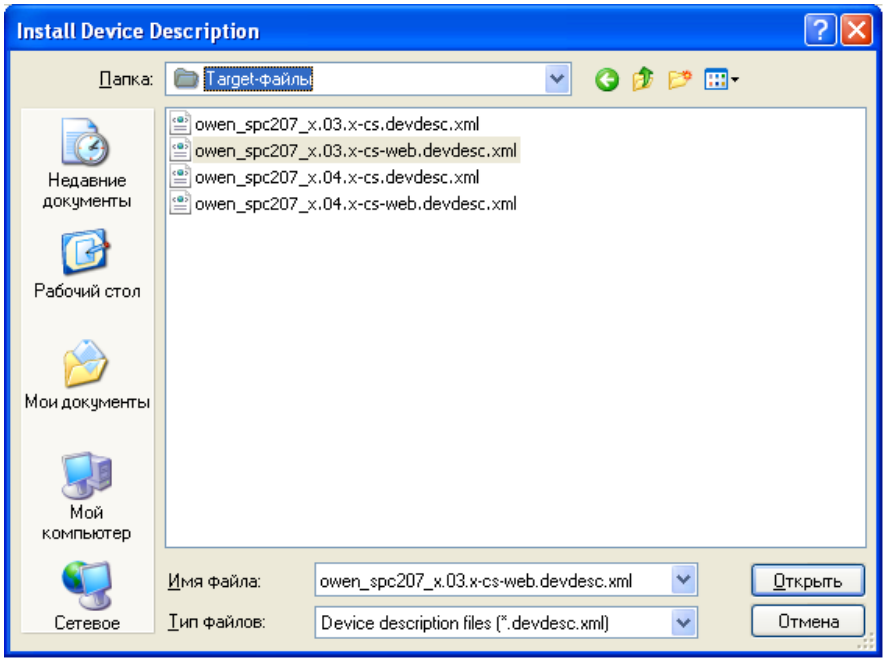


Рисунок 6.14 – Вибір адреси розміщення target-файла на ПК

Після вибору шляху розміщення target-файла і його відкриття в CODESYS 3 інформація про структуру СПК 207 буде додана в базу пристроїв CODESYS 3, як це показано на рис. 6.15.

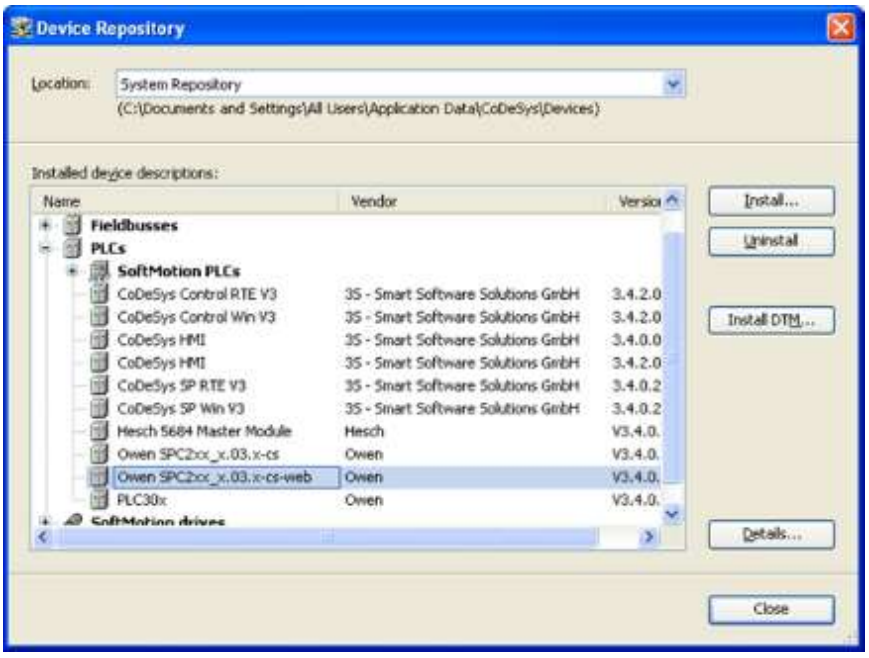


Рисунок 6.15 – Діалогове вікно Device repository
після установки Target-файла СПК 207

Після цього можна завершити роботу з меню Device repository натисканням кнопки Close. Тепер із заданням параметрів стандартного проекту (Standard project) в CoDeSys можна використовувати як виконавчу платформу (Device) СПК 207. Приклад такого вибору показаний на рис. 6.16.

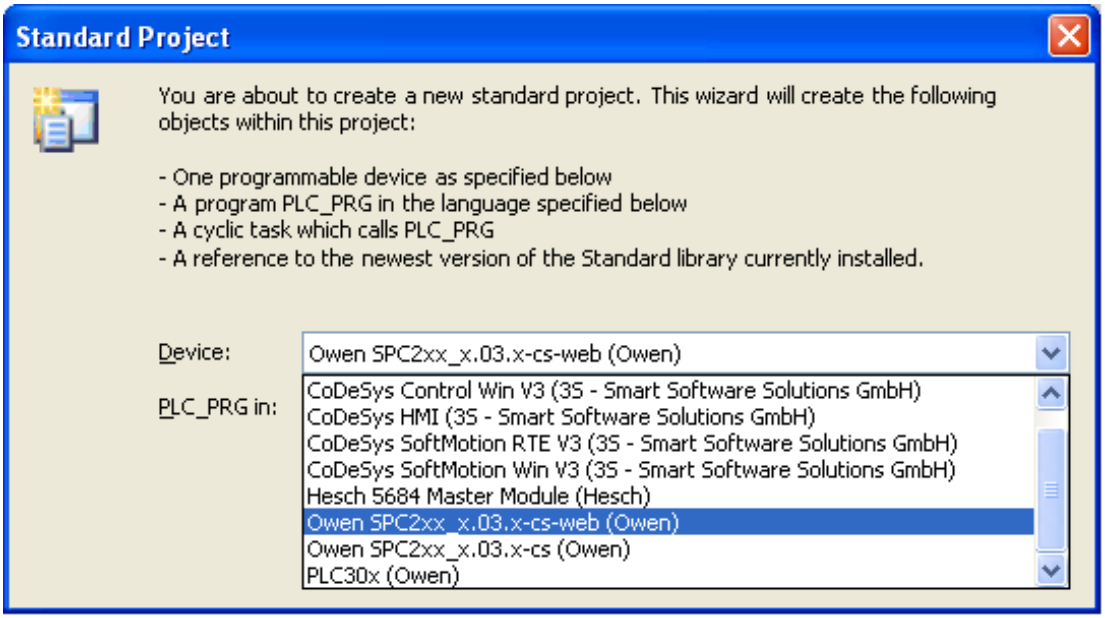


Рисунок 6.16 – Вибір СПК 207 як виконавчого пристрою для Standard project

Результатом такого вибору має стати конфігурація дерева проекту, яка наведена на рис. 6.17.

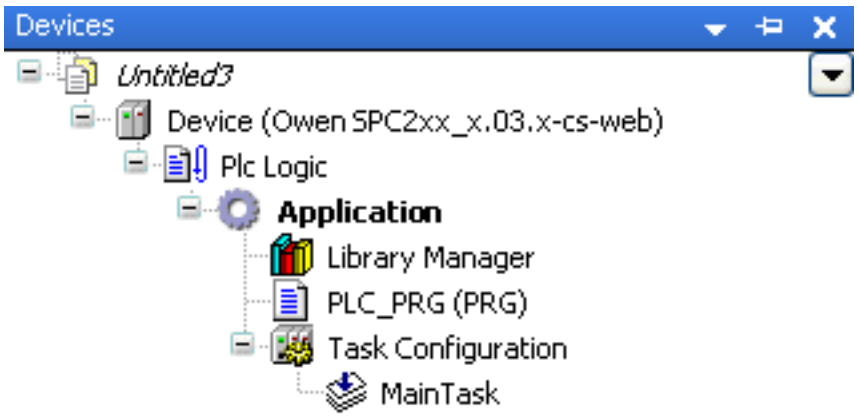


Рисунок 6.17 – Дерево проекту з СПК 207 як виконавча платформа

У разі помилкового вибору target-файла для використовуваного ПЛК його можна змінити, обравши в контекстному меню, що викликається правою кнопкою мишки, команду Update Device, як це показано на рис. 6.18.

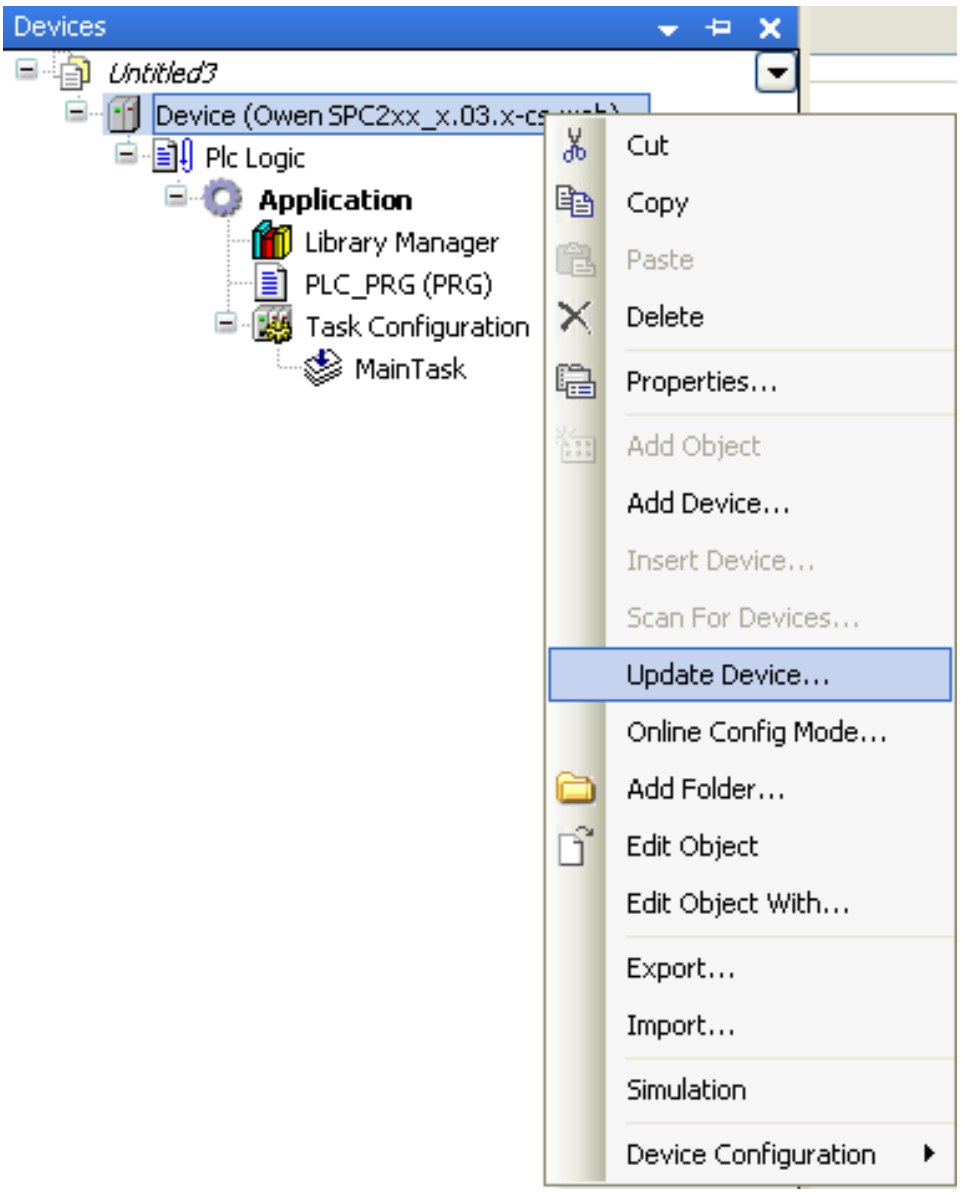


Рисунок 6.18 – Контекстне меню визначення / додавання пристрою в дереві проекту CODESYS 3

Використання цієї команди приведе до виклику вікна визначення ПЛК, показаного на рис. 6.19.

У ньому можна обрати необхідний target-файл, відповідний підключеному ПЛК. Використання команди Add Device дозволить додати в проект новий пристрій, якщо передбачається використання декількох ПЛК для одного проекту.

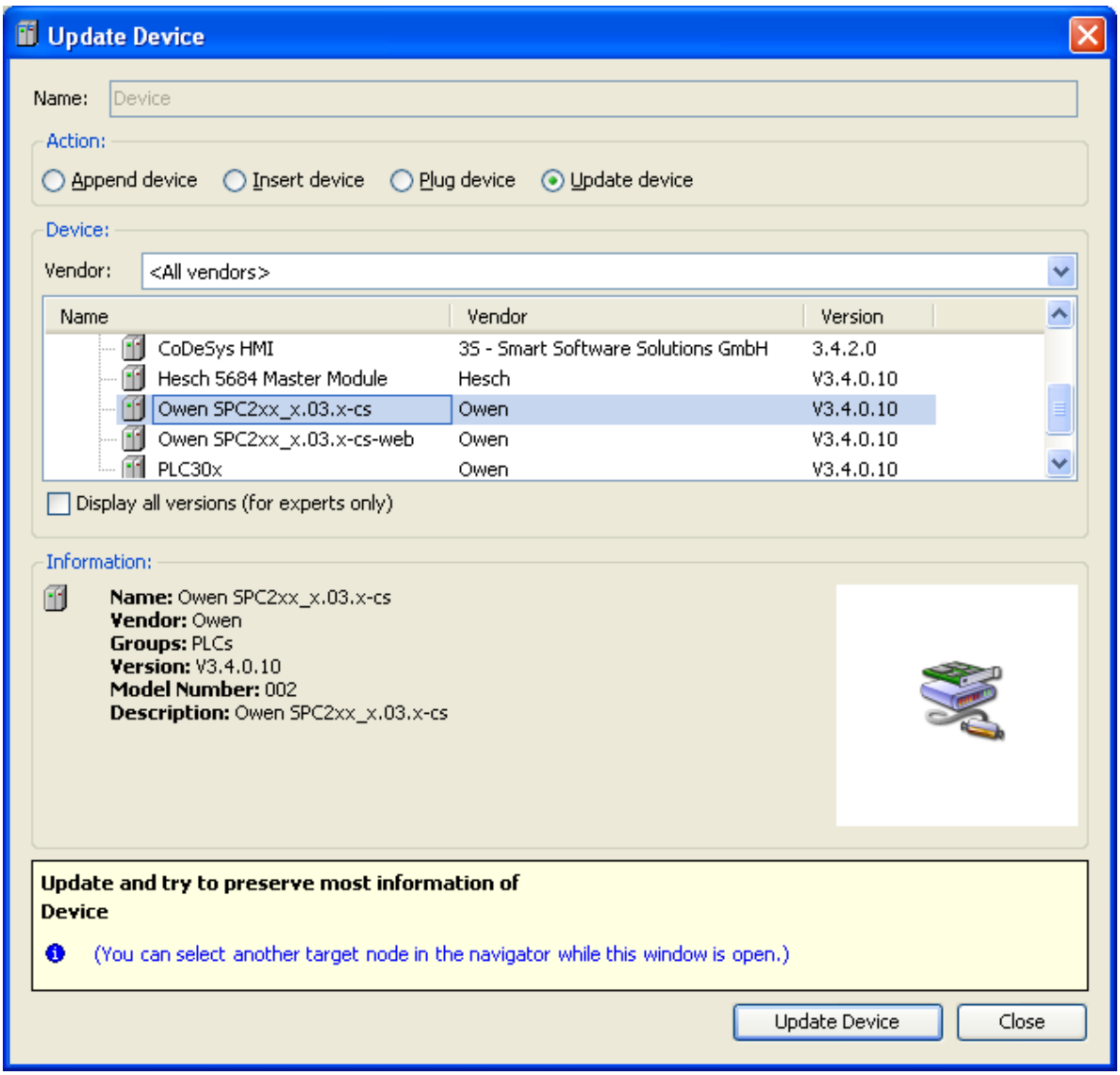


Рисунок 6.19 – Вікно поновлення target-файла для підключеного ПЛК

6.2.3 Налаштування зв'язку ПК і ПЛК

З підключенням контролера до мережі необхідно зробити його налаштування. Більш докладно ця процедура розглянута в керівництві з експлуатації конкретного пристрою. На даному етапі розглянемо приклад налаштування мережі для панелі оператора СПК.

Для входу в режим налаштування СПК необхідно перевести тумблер на задній стороні приладу в нижнє положення (0) і подати живлення на СПК або перезавантажити його за допомогою кнопки «Скид», яка теж розташована на задній стороні. Після завантаження СПК у цьому режимі буде запропоновано обрати мову інтерфейсу. Після вибору мови та натискання кнопки «ОК» буде

потрібно ввести пароль. Пароль за замовчуванням – «owen», він вводиться за допомогою віртуальної клавіатури.

За замовчуванням увімкнений режим набору великих літер, щоб переключитися на рядкові натисніть на кнопку «Shift».

Якщо пароль введений вірно, на екрані конфігуратора з'явиться інформація, відображена на рис. 6.20. Порт Ethernet може отримувати мережні настройки від DHCP-сервера, про те, що ця функція увімкнена, свідчить зелений колір кнопки «DHCP» і відсутність мережних налаштувань.

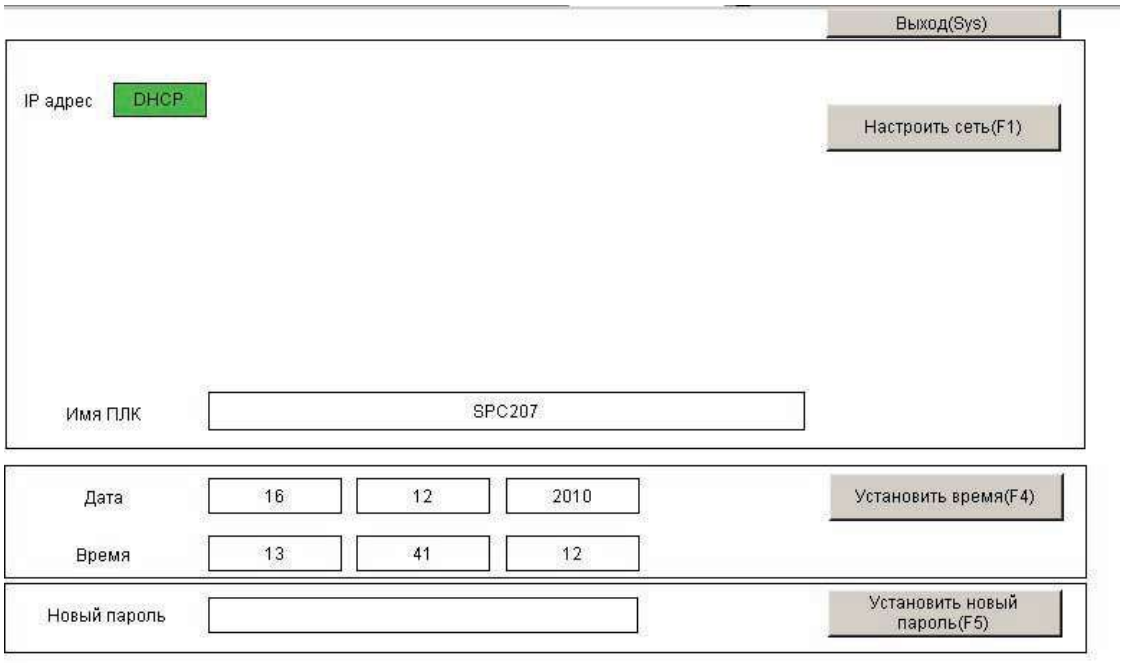


Рисунок 6.20 – Вигляд вікна конфігуратора СПК 207 за замовчуванням

Якщо параметри мережі автоматично отримати неможливо або необхідно працювати зі статичною IP-адресою, то слід відключити цю функцію, натиснувши кнопку «DHCP». На екрані з'явиться інформація, подана на рис. 6.21, а кнопка стане білого кольору.

Далі необхідно налаштувати мережні параметри СПК «IP-адреса», «Маска», «Широкомовна адреса» відповідно до логіки роботи мережі. Вони встановлюються у вікні конфігуратора введенням значень у своїх вікнах за допомогою віртуальної клавіатури або кнопок «▲» і «▼» над і під полями введення.

В полі «Ім'я ПЛК» слід вказати ім'я панелі (під цим ім'ям панель буде вказана під час сканування мережі в середовищі CoDeSys). Слід врахувати, що ім'я може містити тільки латинські літери, цифри і знак підкреслення «_».

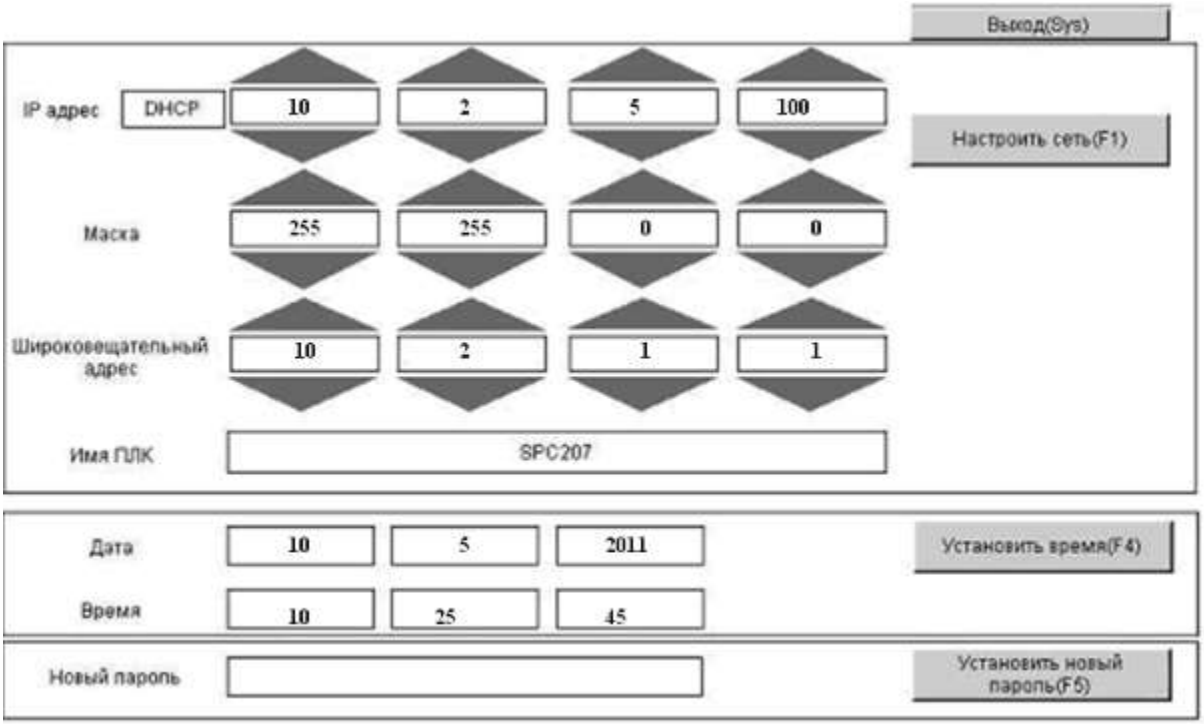


Рисунок 6.21 – Налаштування зв’язку для СПК 207

Після установки всіх параметрів слід натиснути кнопку «Налаштувати мережу» на екрані або кнопку «F1» на передній панелі СПК 207.

Аналогічно в панелі можна встановити значення годин реального часу і змінити пароль входу в конфігуратор.

Для підключення ПЛК з мережі необхідно виконати налаштування CODESYS. Для цього слід відкрити налаштування цільової платформи подвійним клацанням по назві ПЛК в дереві проекту, відкриється діалог «Device» зі вкладкою «Communication Settings» («Параметри з’єднання»). Далі необхідно додати новий вузол (Gateway) зв’язку командою «Add gateway» і налаштувати його параметри, як показано на рис. 6.22.

Раніше створені вузли зберігаються і будуть відображені в діалозі параметрів з’єднання. Якщо налаштування було проведено раніше, то можна пропустити цей крок і перейти відразу до установки каналу зв’язку з пристроєм («В результаті пошуку ...»).

З додаванням нового сайту або редагуванням старого відкриється вікно, наведене на рис. 6.22. У даному вікні необхідно ввести IP-адресу ПЛК, наприклад, для розглянутого прикладу «10.2.5.100» (для відкриття поля редагування необхідно двічі клацнути по полю колонки). Установку Port можна залишити без змін. Для підтвердження введення даних потрібно натиснути на кнопку «OK».

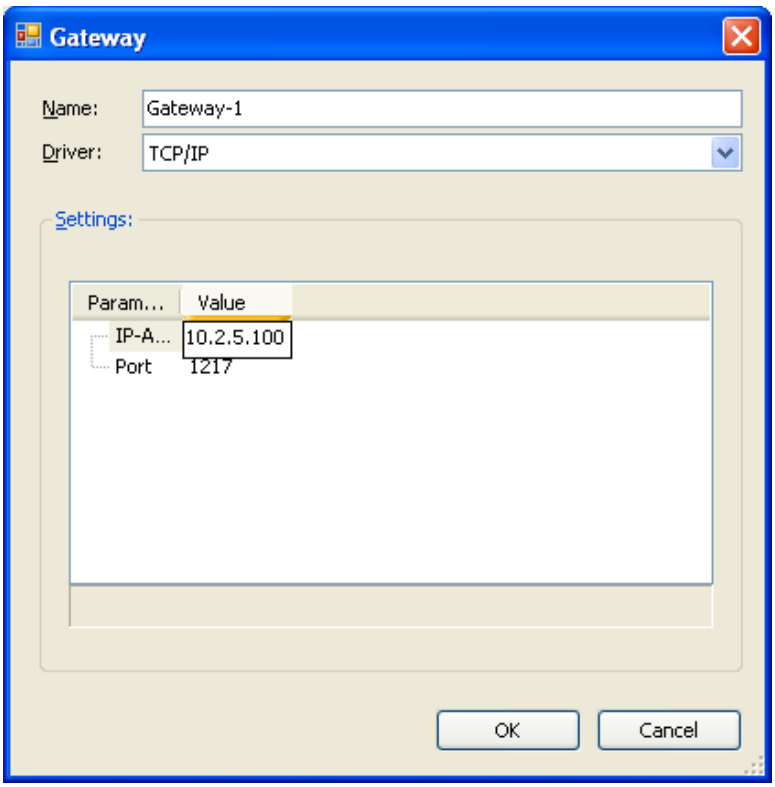


Рисунок 6.22 – Установка IP-адреси ПЛК в мережі

Тепер Gateway (вузол) з'явився в полі в лівій частині діалогу. Якщо gateway працює нормально, то маркер поруч із позначкою – зеленого кольору, в іншому випадку – червоного. Для отримання актуальної інформації про доступні за даним geteway пристрої необхідно виконати сканування мережі, натиснувши кнопку «Scan network».

Внаслідок пошуку за IP-адресою може бути виявлено декілька пристроїв заданого класу (якщо вони підключені до загальної підмережі). З них необхідно обрати те, яке реально задіяно в проекті, і натиснути кнопку «Set active path» для вибору його активним, як показано на рис. 6.23.

Цей перелік містить тільки ті пристрої, які збігаються з типом використовуваного в проекті, якщо необхідно відобразити всі пристрої, то слід переключити параметри фільтру з «Target ID» на «None».

Після підключення й запису програми в ПЛК можна перевірити правильність вибору пристрою на вкладці «Log» діалогу «Device». Для цього необхідно оновити інформацію, натиснувши на відповідну кнопку, обрати інформаційні повідомлення і в фільтрі обрати «CmpBlkDrvUdp», отримати у другому рядку повідомлень налаштування IP-адреси і маски мережі. Приклад результатів такої перевірки показаний на рис. 6.24.

Рисунок 6.23 – Вибір активного ПЛК

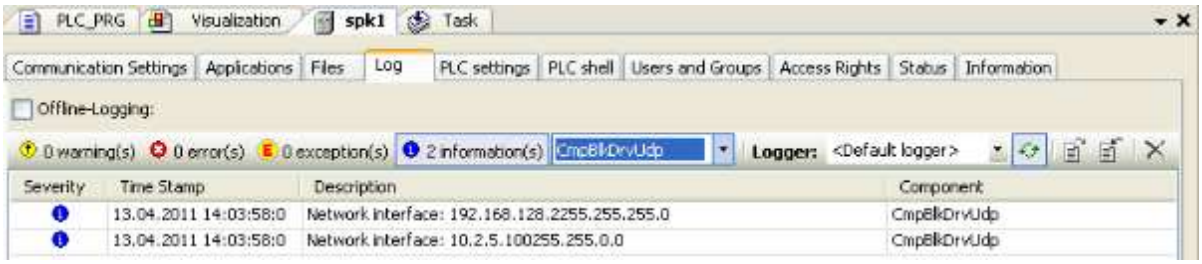


Рисунок 6.24 – Перевірка параметрів зв’язку

Існує можливість підключення ПК безпосередньо до ПЛК. За такого підключення крім конфігурації каналу зв’язку з ПЛК в середовищі CODESYS 3 необхідно налаштувати мережну карту ПК так, щоб ПК і ПЛК стали двома різними приладами однієї підмережі. Якщо мережна карта вже налаштована, то можна пропустити цей крок і перейти відразу до налаштування Geteway («Налаштування зв’язку в ...»). Приклад таких налаштувань подано в таблиці 6.1.

Таблиця 6.1 – Приклад мережних налаштувань для ПК

IP	10	2	5	1
Mask	255	255	255	0
Broadcast	10	2	1	1

Єдиною відмінністю мережних налаштувань ПК і ПЛК в даному прикладі є четверте число IP-адреси, що визначає адресу приладу в підмережі.

Для налаштування мережних параметрів ПК необхідно налаштувати параметри мережі в розділі Пуск / Налаштування / Мережні підключення, як показано на рис. 6.25.

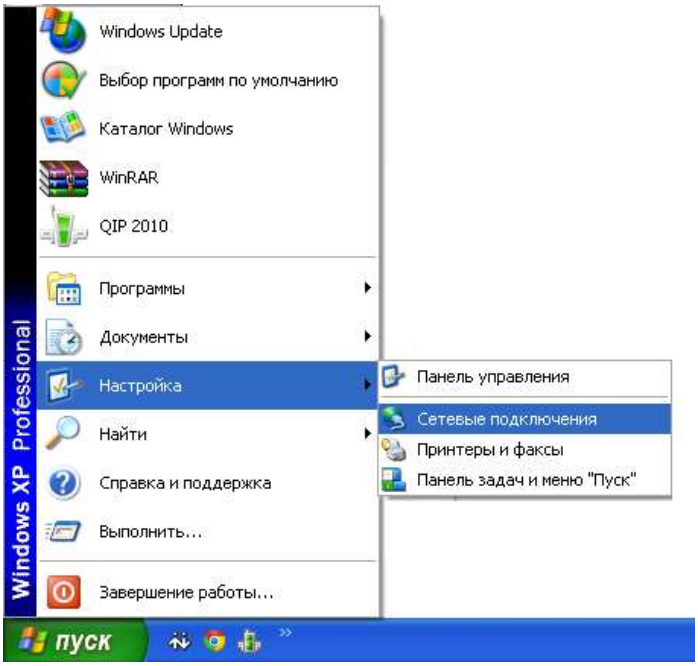


Рисунок 6.25 – Зміна параметрів мережі в ПК

Після цього подвійним клацанням лівої кнопки мишки обрати з'єднання, що використовується, і натиснути кнопку «Властивості», як показано на рис. 6.26. Далі необхідно в меню Компоненти, що використовуються цим підключенням, обрати пункт Протокол Інтернету (TCP/IP) і натиснути кнопку «Властивості».

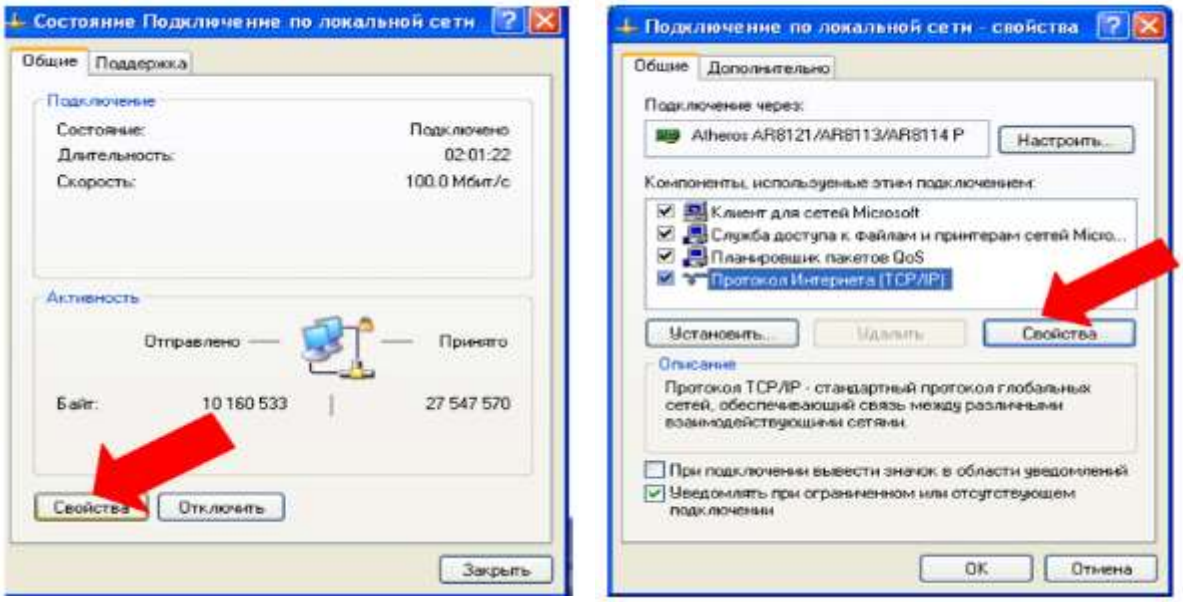


Рисунок 6.26 – Вибір властивостей мережного підключення і налаштувань TCP/IP

У діалоговому вікні Властивості: Протокол Інтернету (TCP/IP) потрібно зробити налаштування параметрів підключення згідно з таблицею 6.1 (див. рис. 6.27). Завершивши налаштування, необхідно натиснути кнопку «ОК» для прийняття їх як робочих.

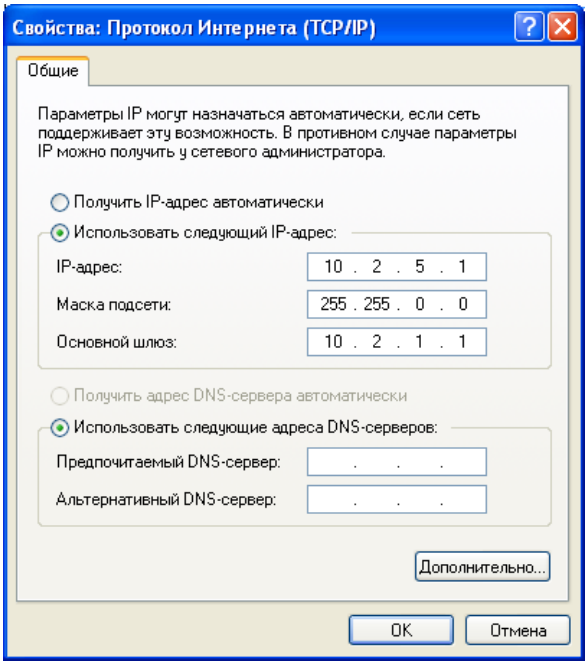


Рисунок 6.27 – Введення параметрів TCP/IP у ПК

З додаванням нового Gateway в налаштуваннях зв’язку (Communication settings) припустимо залишити налаштування за замовчуванням, як показано на рис. 6.28.

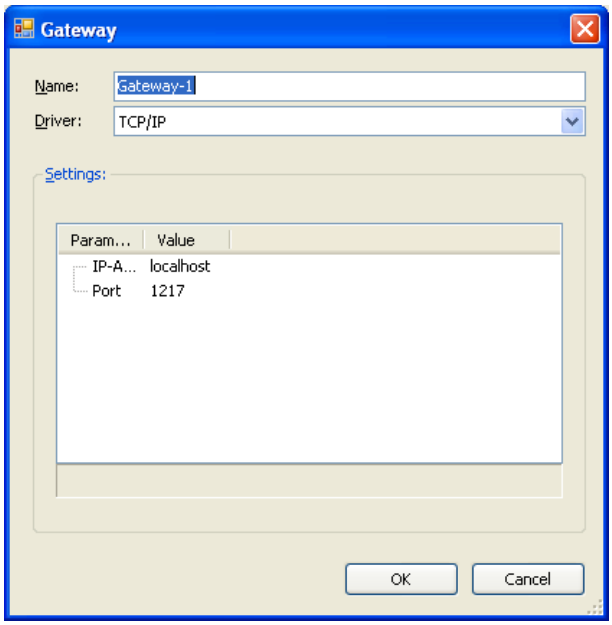


Рисунок 6.28 – Налаштування Gateway для визначення ПЛК

Результатом команди «Scan network» буде єдиний підключений контролер, який за замовчуванням буде обраний як активний пристрій, як показано на рис. 6.29 (за відключеного WinPLC).

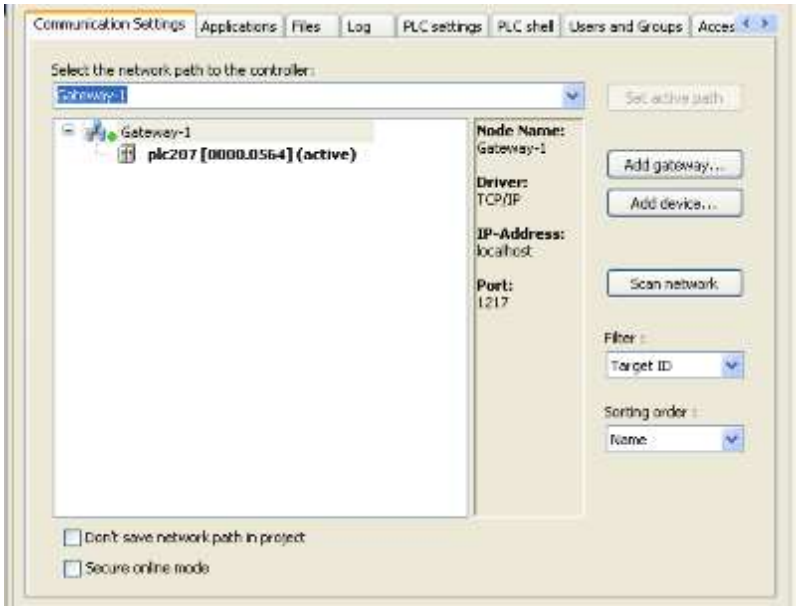


Рисунок 6.29 – Автовизначення активним пристроєм знайденого ПЛК

На цьому налаштування зв’язку завершені. Можна робити завантаження програми в ПЛК за допомогою команд меню Online і запуск / налаштування програми за допомогою команд меню Debug.

6.2.4 Створення проекту в CODESYS 3

Запуск програмного середовища CODESYS 3 може бути проведений з меню Пуск так: Програми => 3S CODESYS => CODESYS => CODESYS V3.x, або подвійним клацанням по іконці на робочому столі. Результатом запуску буде вікно, подане на рис. 6.30.

Для створення нового проекту можна обрати пункт New project вкладки File, клацнути лівою кнопкою мишки по іконці на панелі швидкого виклику або скористатися поєднанням клавіш <Ctrl> + <N>. Результатом цих дій має стати поява вікна New Project, вид якого наведено на рис. 6.31.

У цьому вікні потрібно обрати пункт Standard project категорії General. У розділі Name можна дати ім’я проекту, а в розділі Location – шлях його збереження на ПК.

Після збереження проекту з’являється вікно вибору виконавчого пристрою і мови основної програми. Його вигляд подано на рис. 6.32.

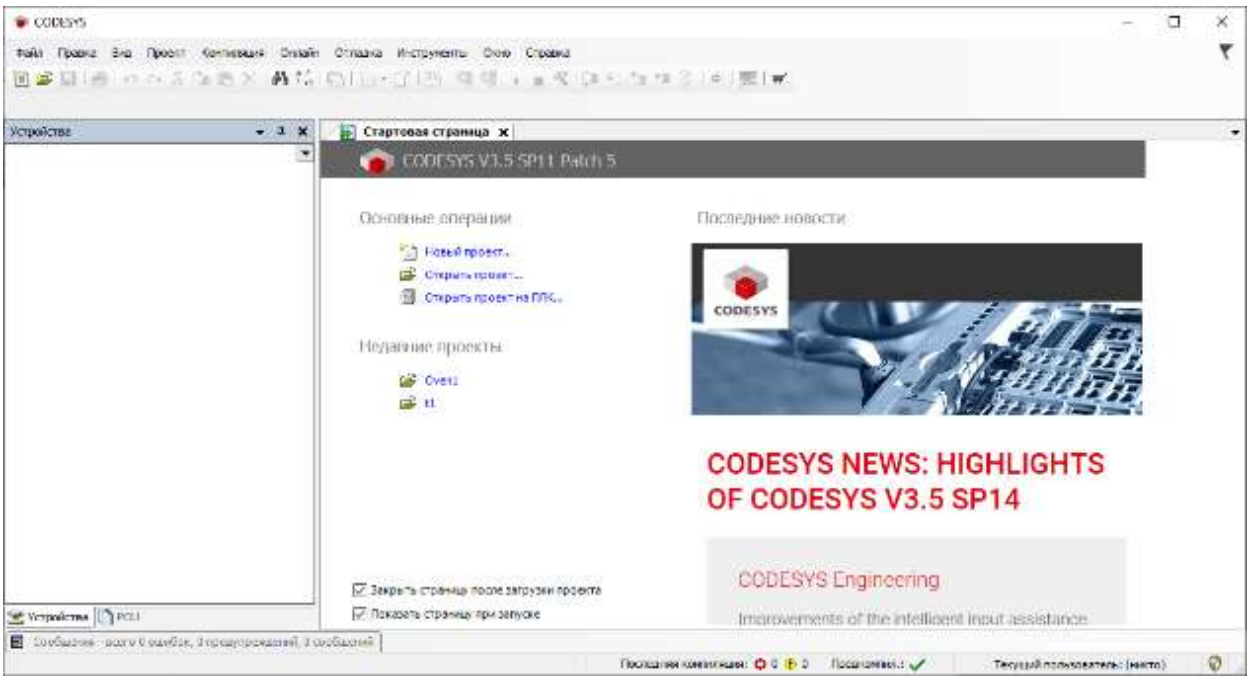


Рисунок 6.30 – Вікно CODESYS 3 під час запуску

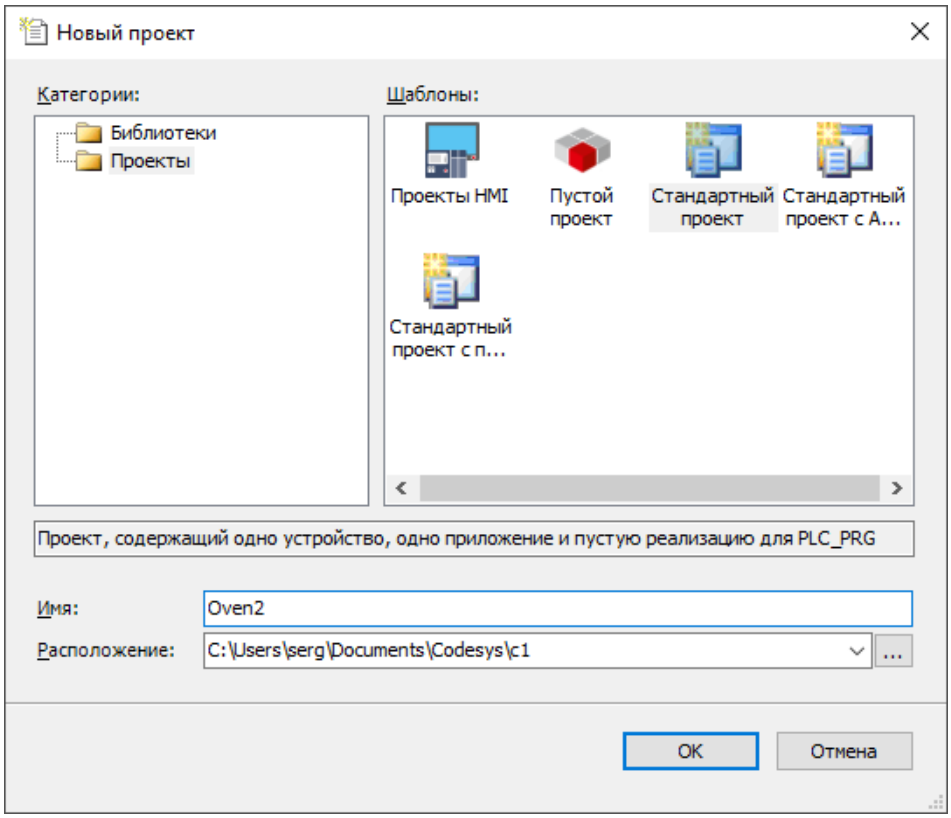


Рисунок 6.31 – Створення нового проекту і визначення адреси його збереження

Як виконавчий пристрій у цьому прикладі буде використаний внутрішній віртуальний ПЛК середовища CODESYS 3. Цей пристрій реалізує повнофункціональний ПЛК на ПК, дозволяючи працювати в тому числі і з

зовнішніми пристроями, підключеними до COM-портів комп'ютера. Для його використання в пункті Device оберемо CODESYS SP Win V3.

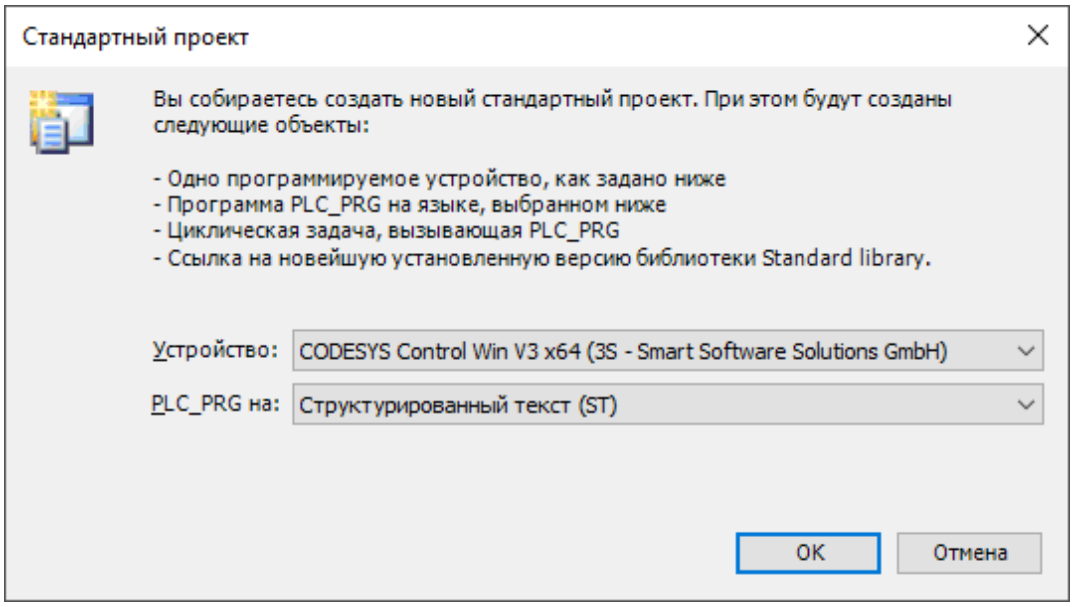


Рисунок 6.32 – Вибір пристрою і тексту основної програми

Мовою проекту PLC_PRG оберемо мову структурованого тексту ST. Після задання цих параметрів дерево проекту набуде вигляду, показаного на рис. 6.33.

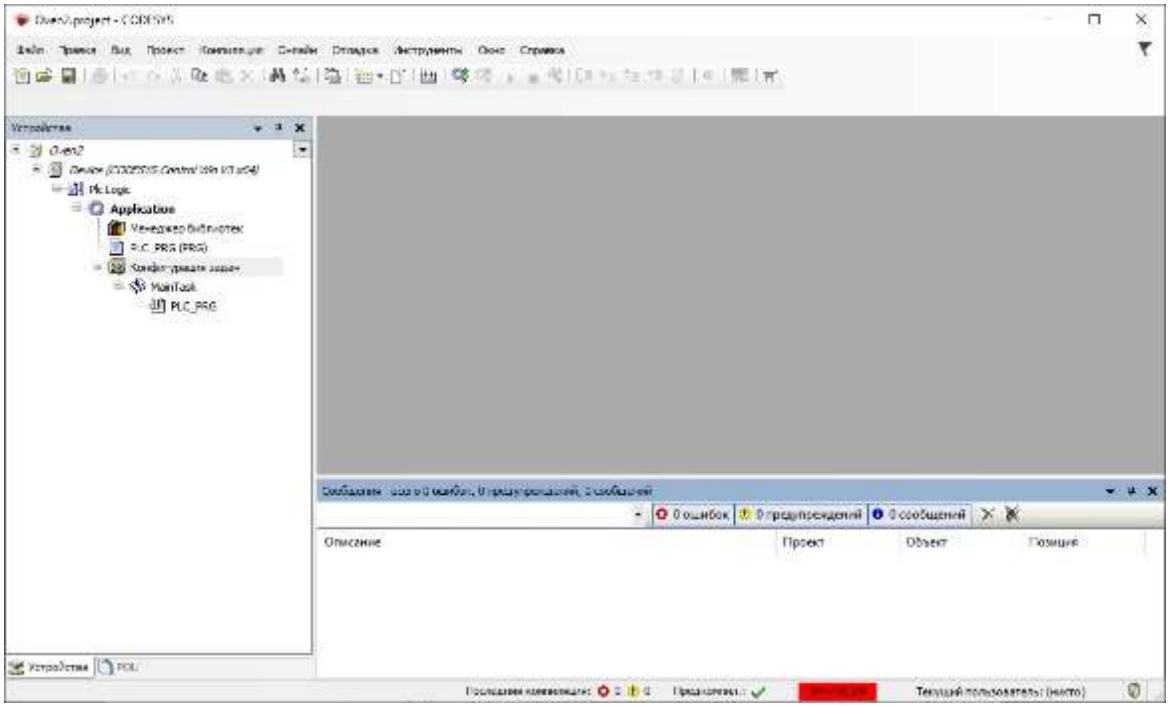


Рисунок 6.33 – Дерево проекту

Головне вікно включає в себе такі основні частини проекту:

– вкладки Devices (дерево пристроїв) і POU (функції, функціональні блоки і програми проекту) в лівій частині вікна екрану;

– робочу область (на рис. 6.33 неактивна, пофарбована сірим кольором, розташована у правій частині екрану);

– Messages (рядок повідомлень), що показує кількість і статус службових повідомлень, розташований у правій центральній частині екрану;

– Description, що включає опис етапів роботи CODESYS і розшифровку службових повідомлень і розташована в нижньому правому куті екрана.

У дереві проекту з'явилася його назва (Oven2), вид пристрою (CODESYS Control Win V3) та наявні програми (Application), включаючи менеджер завдань і бібліотек, а також основну програму PLC_PRG.

Зовнішній вигляд менеджера бібліотек у конфігурації за замовчуванням наведено на рис. 6.34. Перелік містить бібліотеку Standard, що включає лічильники, таймери, перемикачі та інші основні функціональні блоки.

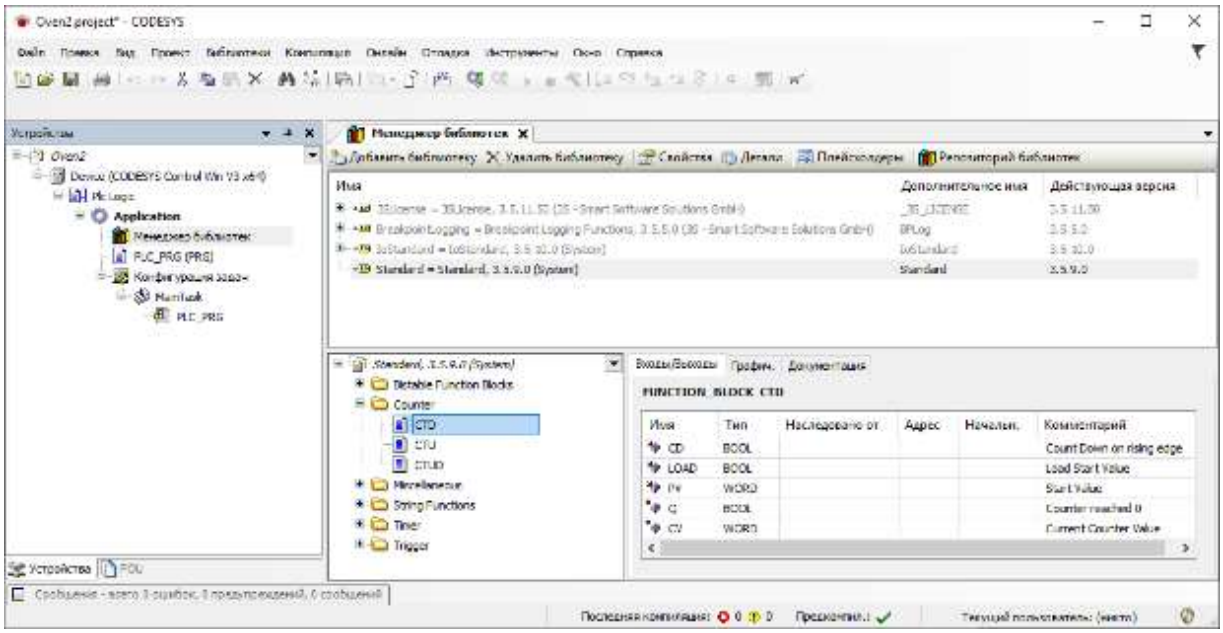


Рисунок 6.34 – Стандартна бібліотека CODESYS 3
(додається за замовчуванням)

Для прикладу створимо найпростіший проект з двома змінними *x* і *y* як показано на рис. 6.35. У цьому проекті змінна *x* збільшується на 1 кожен цикл роботи ПЛК, а змінна *y* аналогічно зменшується.

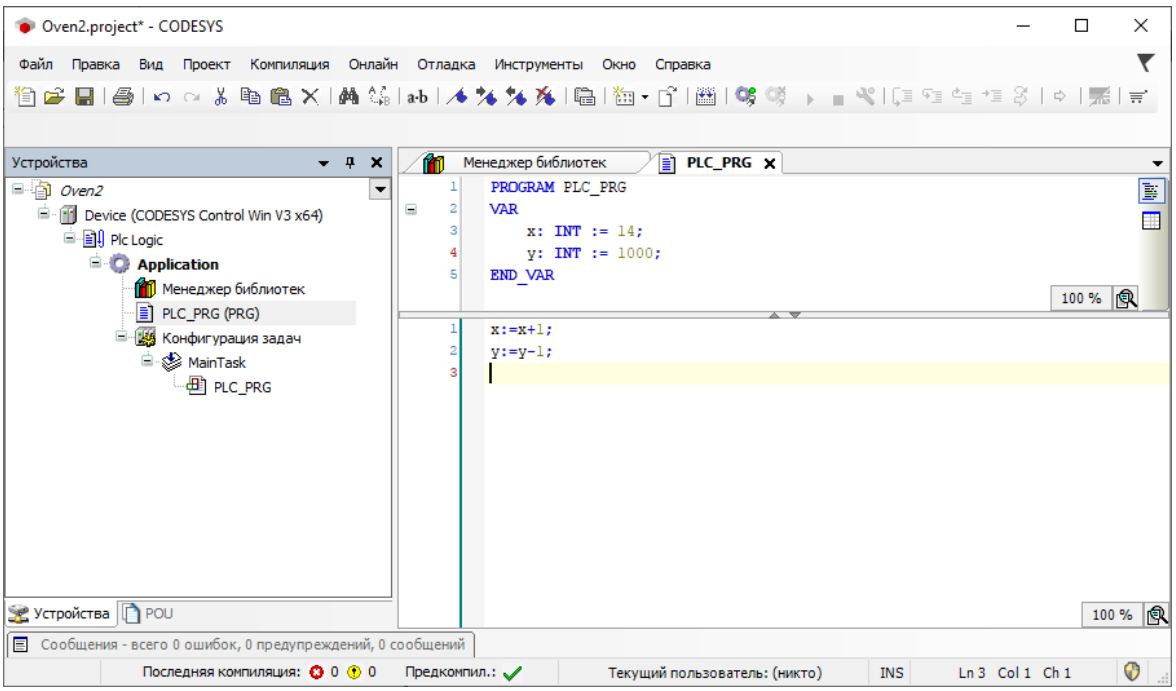


Рисунок 6.35 – Вигляд програми інкремента / декремента

Задання типів і початкових значень змінних може бути здійснено в розділі змінних PLC_PRG (між службовими словами VAR і END_VAR) або за допомогою вікна автооголошення, вид якого подано на рис. 6.36.

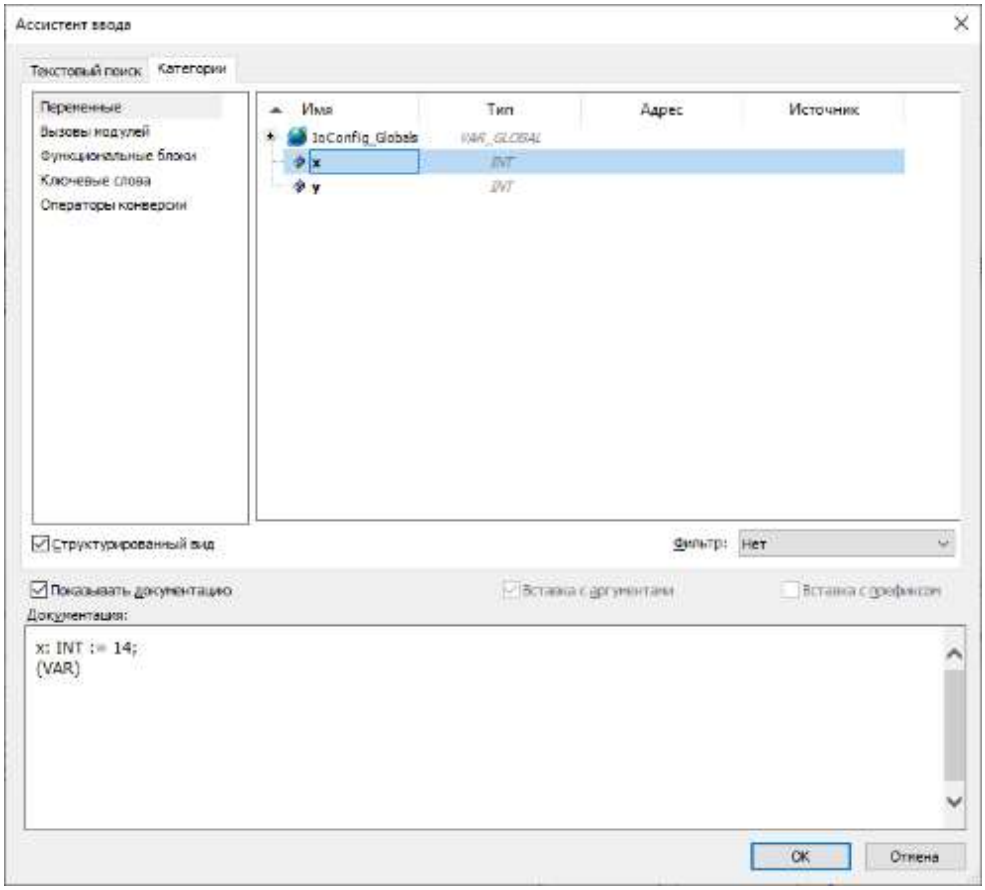


Рисунок 6.36 – Вікно асистента введення змінної

Після створення програми виконується її компіляція за допомогою виклику опції головного меню або після натискання на кнопку F11.

6.2.5 Робота в режимі емуляції

Для перевірки працездатності проекту без підключення до контролера може бути використаний режим емуляції. У ньому можна провести налаштування і моделювання роботи програми без завантаження в ПЛК.

Вибір режиму роботи «Емуляція» проводиться встановленням позначки в пункті Online / Simulation, як показано на рис. 6.37.

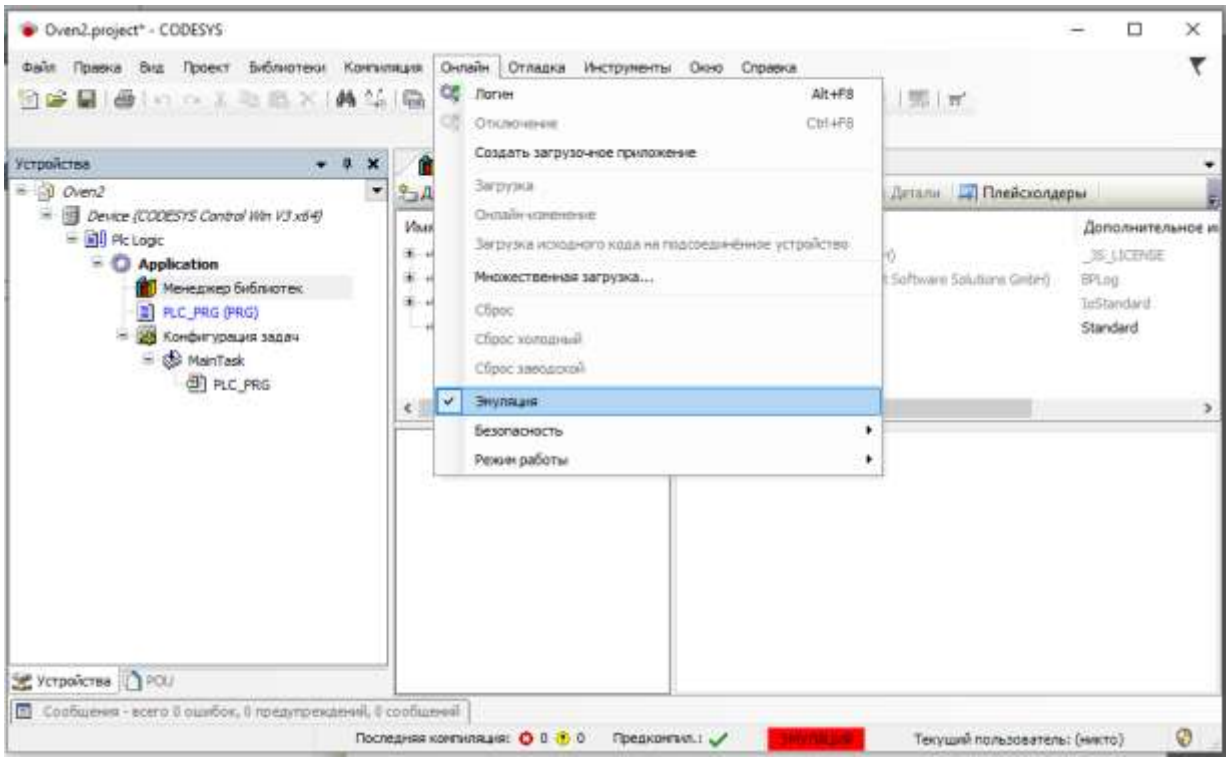


Рисунок 6.37 – Вибір режиму емуляції

Після вибору цього пункту із запуском програми на виконання в меню статусу з’являється напис «ЕМУЛЯЦІЯ». Запуск програми здійснюється двома командами: логін (Alt + F8) або старт (F5). Після запуску програма виконує свій код. Біля кожної змінної відображається її поточне значення, як показано на рис. 6.38.

Примусово змінити значення змінної можна в таблиці змінних у полі Prepared Value (Підготовлене значення) (рис. 6.39).

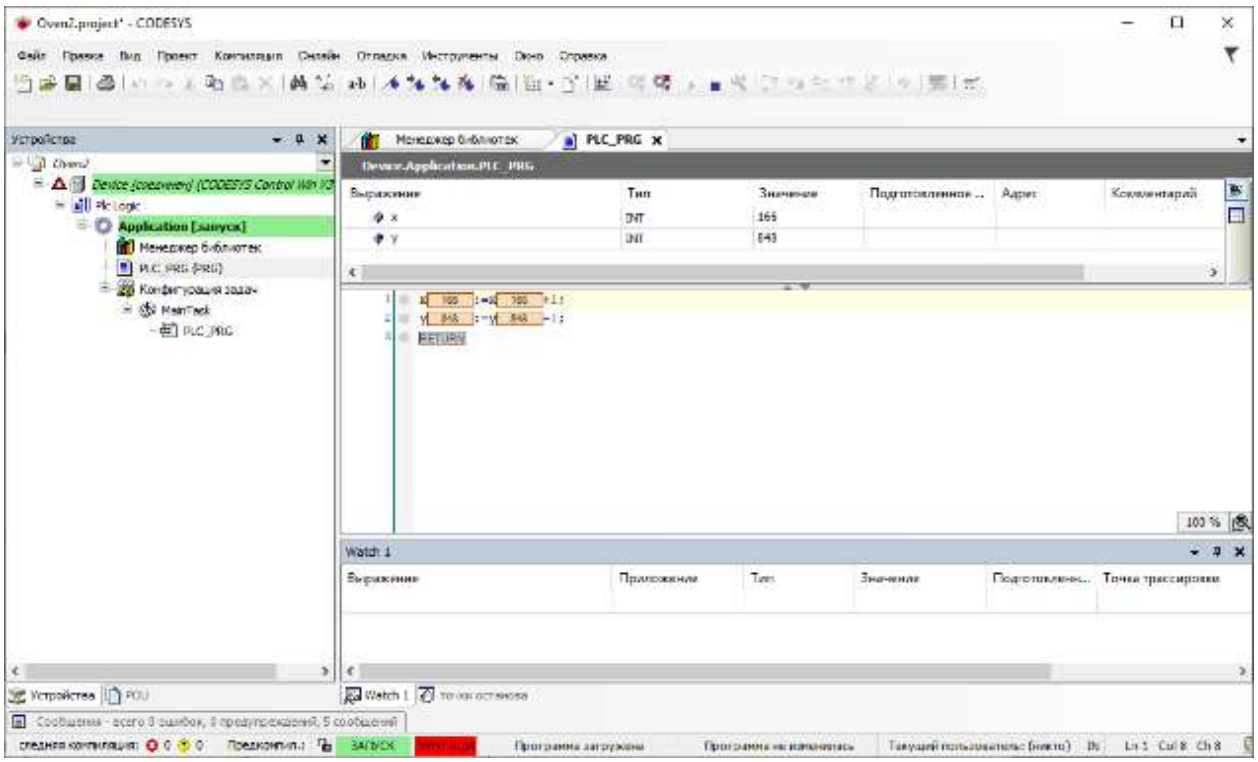


Рисунок 6.38 – Робота програми в режимі емуляції

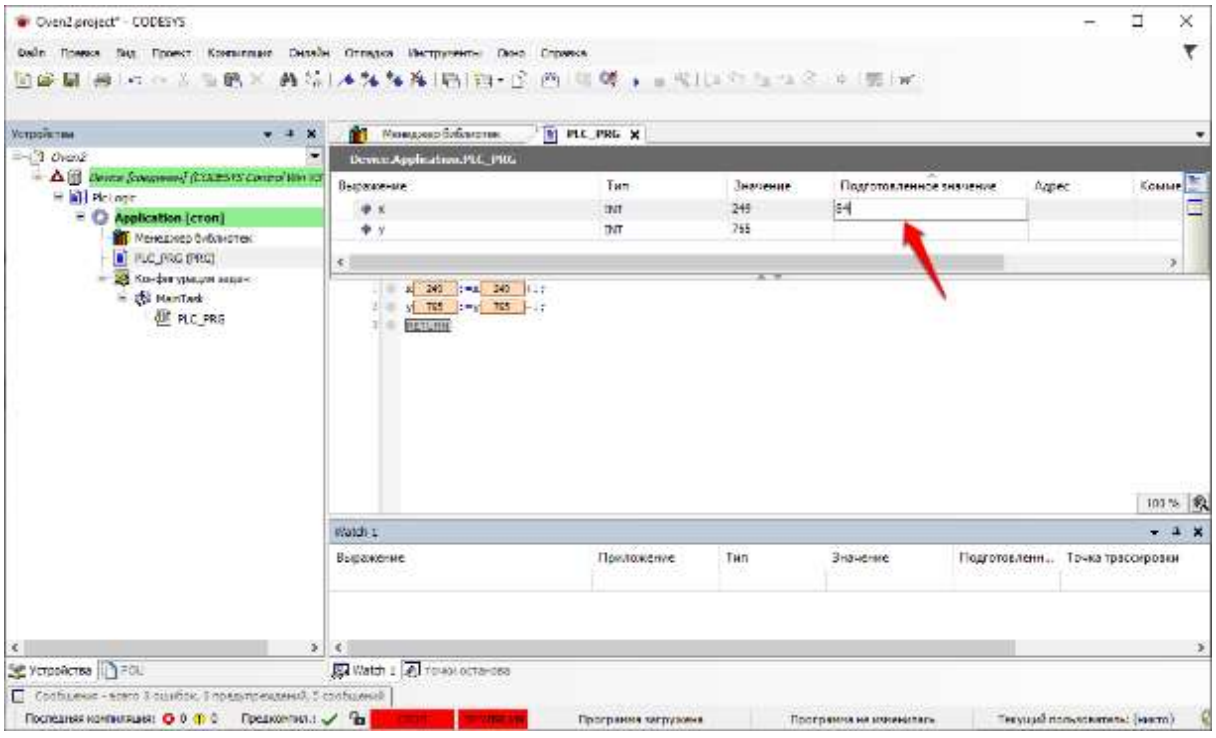


Рисунок 6.39 – Змінення значення змінної в проекті

Також змінити значення змінної можна, викликавши вікно Prepare Value подвійним клацанням лівої кнопки мишки на необхідній змінній (рис. 6.40).

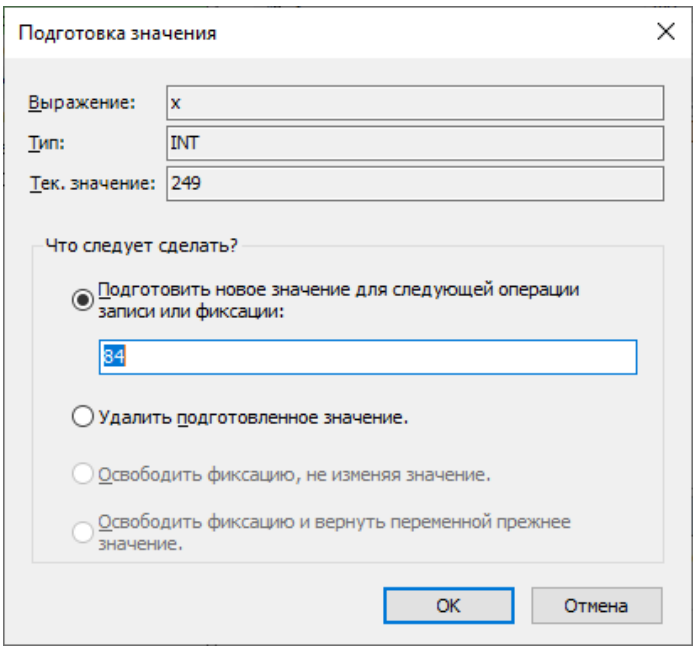


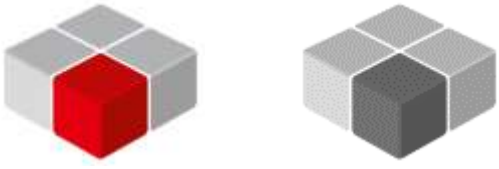
Рисунок 6.40 – Вікно змінення значення змінної

Встановлені значення з’являться в кутових дужках (< >) праворуч від дійсного значення змінної. Встановити змінені значення змінних можна вибором пункту меню Debug / Write values або поєднанням клавіш Ctrl + F7.

6.2.6 Робота з віртуальним контролером CODESYS 3

Можливості користувача під час роботи в режимі симуляції обмежені. Розширити їх без підключення реального ПЛК дозволяє використання віртуального контролера CODESYS 3. Він встановлюється разом з програмним середовищем CODESYS 3 і запускається за допомогою Gateway-сервера.

Gateway-сервер автоматично запускається як сервіс із запуском системи. Переконайтеся, що на панелі задач є кольорова іконка, яка вказує на те, що сервер запущено. На рис. 6.41 показано зовнішній вигляд іконки в різних станах серверу.



Сервер запущено

Сервер зупинено

Рисунок 6.41 – Зовнішній вигляд іконки в різних станах Gateway-серверу

Сіра іконка свідчить про те, що gateway на даний момент зупинений. Ця іконка є частиною програми GatewaySysTray, призначеної для контролю й спостереження за сервісом Gateway. Вона включає в себе меню, яке містить команди start і stop, що дозволяє користувачеві зупиняти і перезапускати сервіс вручну.

Меню також містить команду Exit Gateway Control, яка закриває тільки програму GatewaySysTray, але не сервіс Gateway. Програма GatewaySysTray запускається автоматично із запуском Windows. Проте її можна також запустити з меню Програми.

ПЛК (CODESYS SP Win V3) доступний як сервіс після запуску системи. На панелі задач він поданий іконкою, показаною на рис. 6.42, для стану «зупинено» і для стану «запущено».

PLC-сервіс запущено

PLC-сервіс зупинено

Рисунок 6.42 – Стан віртуального PLC

ПЛК-сервіс може автоматично запускатися під час запуску системи, якщо це підтримується самою системою. В іншому випадку для його запуску необхідно вручну застосувати команду ‘Start PLC’ з меню, яке відкривається клацанням мишки по іконці.

Для підключення віртуального контролера в дереві проекту подвійним клацанням по пристрою Device (CODESYS SP Win V3) відкриємо діалог PLCWinNT з вкладкою Communication settings. Якщо встановлюється з’єднання в CODESYS V3.x в перший раз, то спочатку вам необхідно поставити локальний Gateway-сервер. Для цього використовується кнопка Add gateway, після чого з’явиться вікно, вигляд якого подано на рис. 6.43.

Якщо під час попередніх сесій вже налаштовувався сервер, він буде відображений у діалозі параметрів з’єднання, як показано на рис. 6.44. У такому випадку ви можете пропустити цей крок і перейти відразу до установки каналу зв’язку з пристроєм.

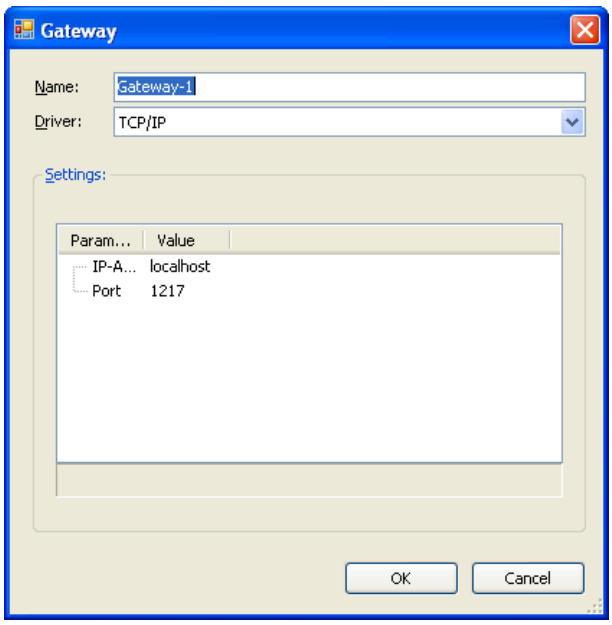


Рисунок 6.43 – Вікно додавання каналу Gateway

Рисунок 6.44 – Діалогове вікно параметрів з’єднання

У діалоговому вікні необхідно натиснути кнопку Scan network для отримання списку доступних мережних пристроїв, як це показано на рис. 6.45. У разі невдалого підключення необхідно перевірити налаштування цільової платформи.

Далі натисканням кнопки Set active path зробимо вибір використовуваного контролера. Праворуч від імені пристрою з’явиться напис active, як показано на рис. 6.46.

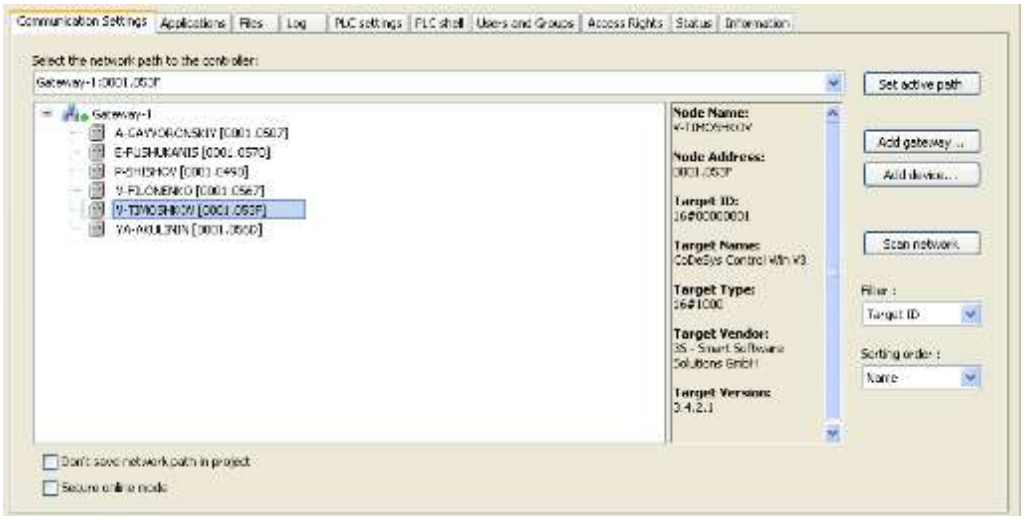


Рисунок 6.45 – Пошук віртуального контролера

Рисунок 6.46 – Вибір активного контролера

Віртуальний контролер дозволяє не тільки робити налаштування програми, але й може бути використаний як цільова платформа для підключення зовнішніх пристроїв і взаємодії з ними за COM-портами ПК.

6.2.7 Робота з бібліотеками в CoDeSys 3

У CODESYS 3 використовується два види бібліотек:

– *.library – це бібліотека з відкритим кодом, тобто можна подивитися її код і простежити його виконання;

– *.compiled-library – це відкомпільована бібліотека з закритим кодом, її використовують, коли хочуть захистити свої розробки.

Library Repository використовується для додавання нової бібліотеки. CODESYS веде свою базу бібліотек і стежить за їх версіями. Додавання бібліотеки в Library Repository – це її оголошення, що тепер вона доступна для використання в проектах. Library Manager використовується для додавання оголошеної через Library Repository бібліотеки безпосередньо в проект.

Разом з кожною бібліотекою йде файл з її описом і керівництвом з установки.

У проекті CODESYS 3 після створення проекту доступні бібліотеки, що підключаються за замовчуванням. Це насамперед бібліотека Standard.library. Вона містить всі функції і функціональні блоки, які потрібні для системи програмування відповідно до стандарту МЕК 61131-3.

Бібліотека UTIL.Library містить додатковий набір різних функцій і функціональних блоків, що застосовуються для BCD і біт / байт перетворень, додаткових математичних функцій, а також регуляторів, генераторів і перетворень аналогових сигналів.

Для додаткової функціональності (візуалізація, робота з профілями тощо) потрібні додаткові бібліотеки. Ці бібліотеки використовуються неявно і додаються в проект автоматично.

Розділ «Менеджер бібліотек» містить основні команди для роботи з бібліотекою, які доступні за рахунок плагіна Library Manager Editor. За замовчуванням вони підключені в меню Бібліотеки.

Менеджер бібліотек використовується для додавання бібліотек і управління ними в проекті. Установка бібліотек, а також задання директорій (репозиторіїв) бібліотек виконується в діалозі Репозиторій бібліотек, який є частиною Менеджера бібліотек. Його можна відкрити командою з панелі меню (за замовчуванням меню Інструменти) або з вікна редактора.

Редактор Менеджера бібліотек доступний за рахунок плагіна Library Manager Editor. Його можна додати в проект за допомогою команди Додати об’єкт. При цьому він буде вставлений у вікно POU або у вікно пристроїв і призначений конкретному пристрою або додатку. Можна вставити тільки один об’єкт Менеджера бібліотек. Ім’я об’єкта задається в діалозі «Додати об’єкт».

На рис. 6.47 показаний зовнішній вигляд програми CODESYS з відкритою вкладкою «Менеджер бібліотек».

Менеджер бібліотек можна відкрити у вікні редактора за допомогою команди «Редагувати об’єкт» або подвійним клацанням мишки по об’єкту в дереві вікна «Пристрої».

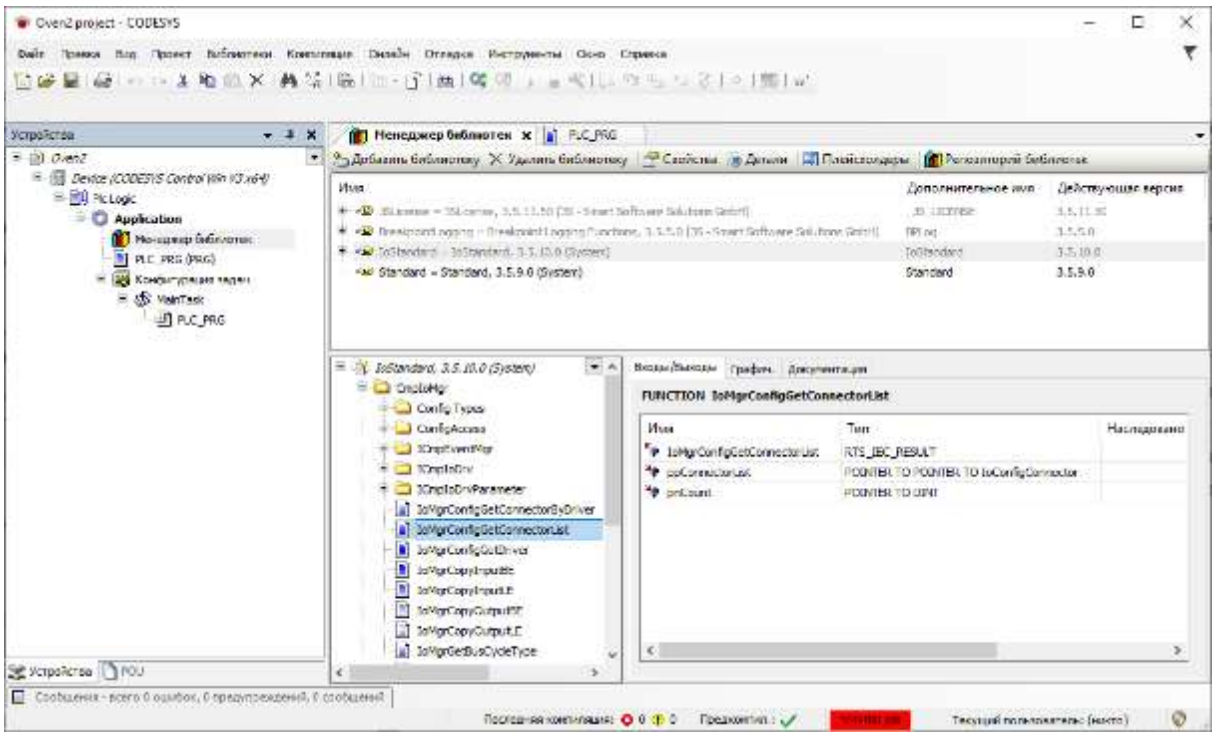


Рисунок 6.47 – Вкладка «Менеджер бібліотек»

У верхній частині діалогу подані бібліотеки, включені у проект. Для них показана така інформація: назва, версія і, можливо, ім’я компанії.

Діюча версія – версія бібліотеки, яка використовуватиметься відповідно до завдання у властивостях бібліотеки.

Бібліотеки, які були автоматично додані в проект за рахунок плагіна, показані сірим, ті, які були додані вручну, показані чорним.

Іконка біля імені бібліотеки вказує на її тип, як показано на рис. 6.48.

– CODESYS V3.x – бібліотека (містить інформацію про версію).

– файл не знайдений або не є підходящою бібліотекою (див. відповідне повідомлення в списку ‘Менеджер бібліотек’ у вікні Повідомлення).

Рисунок 6.48 – Позначення типів бібліотек

Якщо бібліотека містить залежності від інших бібліотек, вони також автоматично вмикаються, якщо доступні, і відображаються з іконкою в піддереві елемента. Піддерево можна відкрити або закрити, натиснувши на відповідну позначку плюса або мінуса. У нижній частині діалогу показані модулі поточної обраної бібліотеки. Тут також розташовані стандартні кнопки для організації модулів (сортування, пошук тощо).

CODESYS 3 дозволяє переглядати властивості елементів стандартних бібліотек, як це показано на рис. 6.49.

Справа доступні три вкладки:

– документація;

– входи / виходи;

– графічне зображення.

На вкладці «Документація» в таблиці подані компоненти модуля, обраного в лівій частині. Для них показані Ім’я, Тип даних і Коментар, який можна додати в оголошення компонента під час створення бібліотеки. За рахунок цих коментарів можна забезпечити користувачів документацією за модулем.

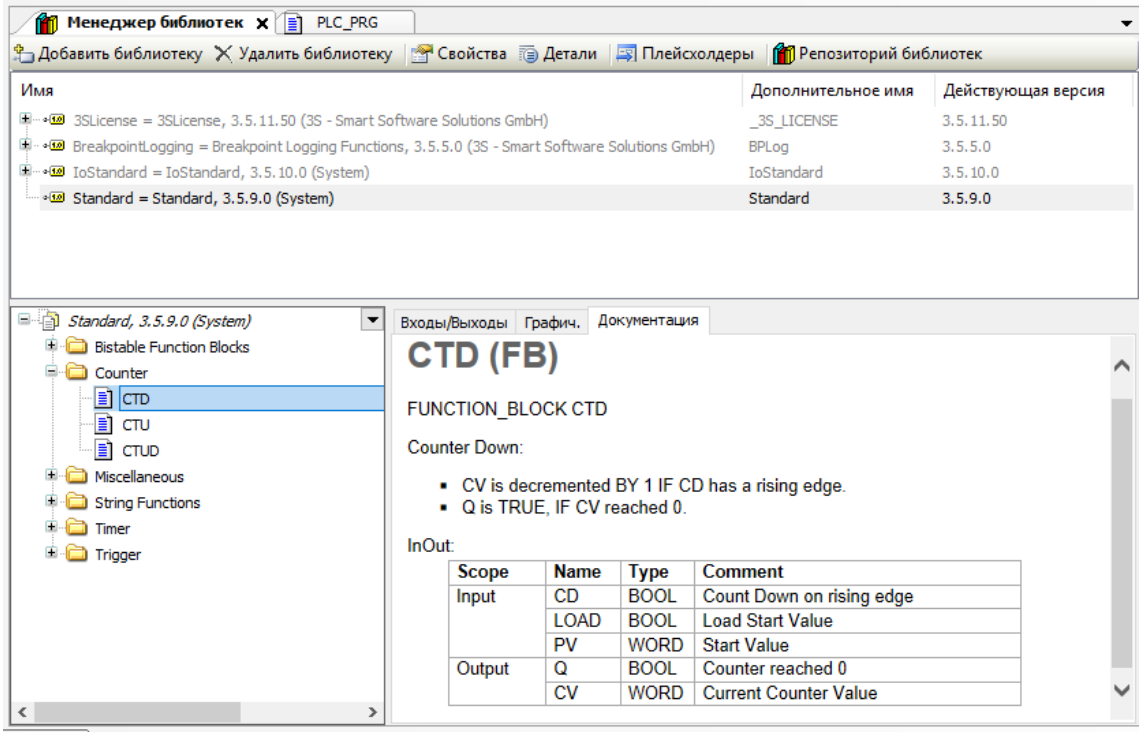


Рисунок 6.49 – Властивості елементів стандартних бібліотек

На вкладці «Входи / виходи» в таблиці перераховані компоненти поточного обраного модуля бібліотеки (Ім’я, Тип даних, Адреса, Початкове значення і Коментар). На рис. 6.50 показаний приклад вкладки «Входи / виходи» для модуля CTUD.

На вкладці «Графічне зображення» показана МЕК-реалізація модуля. На рис. 6.51 показаний приклад графічного подання модуля CTUD.

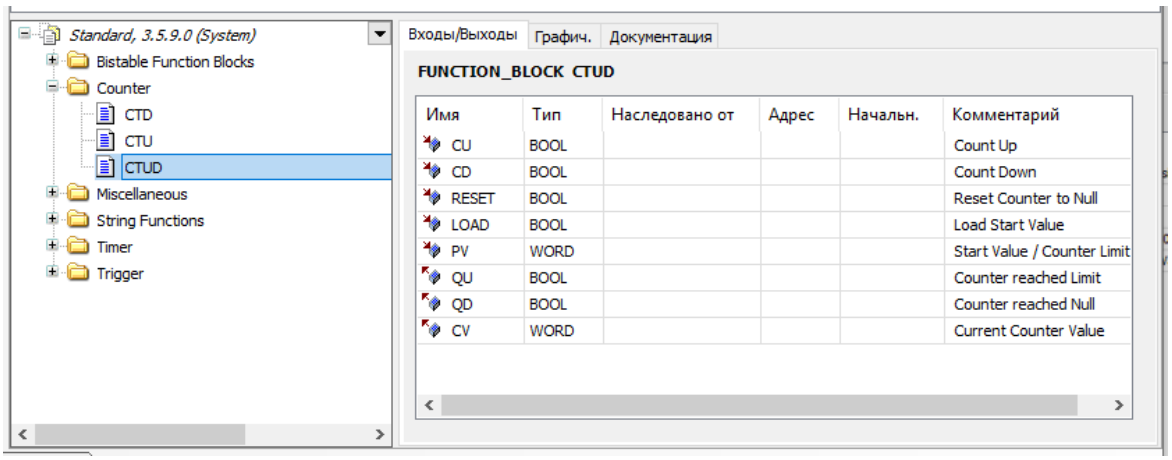


Рисунок 6.50 – Приклад вкладки «Входи / виходи»
для модуля CTUD з розділу лічильників

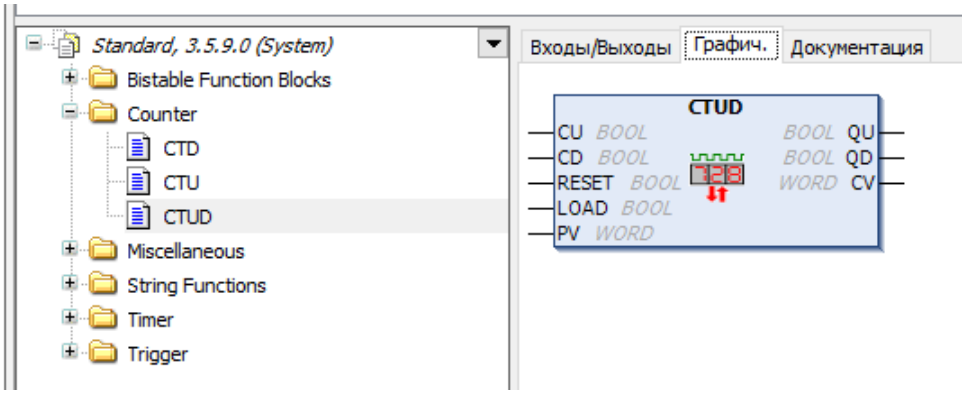


Рисунок 6.51 – Приклад графічного подання модуля CTUD

У закладці «Менеджер бібліотек» на панелі інструментів є кнопка «Репозиторій бібліотек» (див. рис. 6.49). Вона доступна за рахунок плагіна Library Manager Object і призначена для роботи з директоріями бібліотек, а також для їх установки і видалення.

За замовчуванням опція «Репозиторій бібліотек» включена в меню Інструменти головного меню програми.

Репозиторій бібліотек – це база даних бібліотек, встановлених у локальній системі для подальшого включення в проекти CODESYS.

На рис. 6.52 показано вікно «Репозиторій бібліотек».

У діалозі представлені задані директорії бібліотек (репозиторії) і встановлені бібліотеки. Тут можна додавати, змінювати і видаляти репозиторії, а також встановлювати й видаляти бібліотеки. У списку відображатимуться встановлені бібліотеки обраної директорії і компанії. У списку показані

імена бібліотек (Тема), номер версії та ім’я компанії відповідно до інформації про проект.

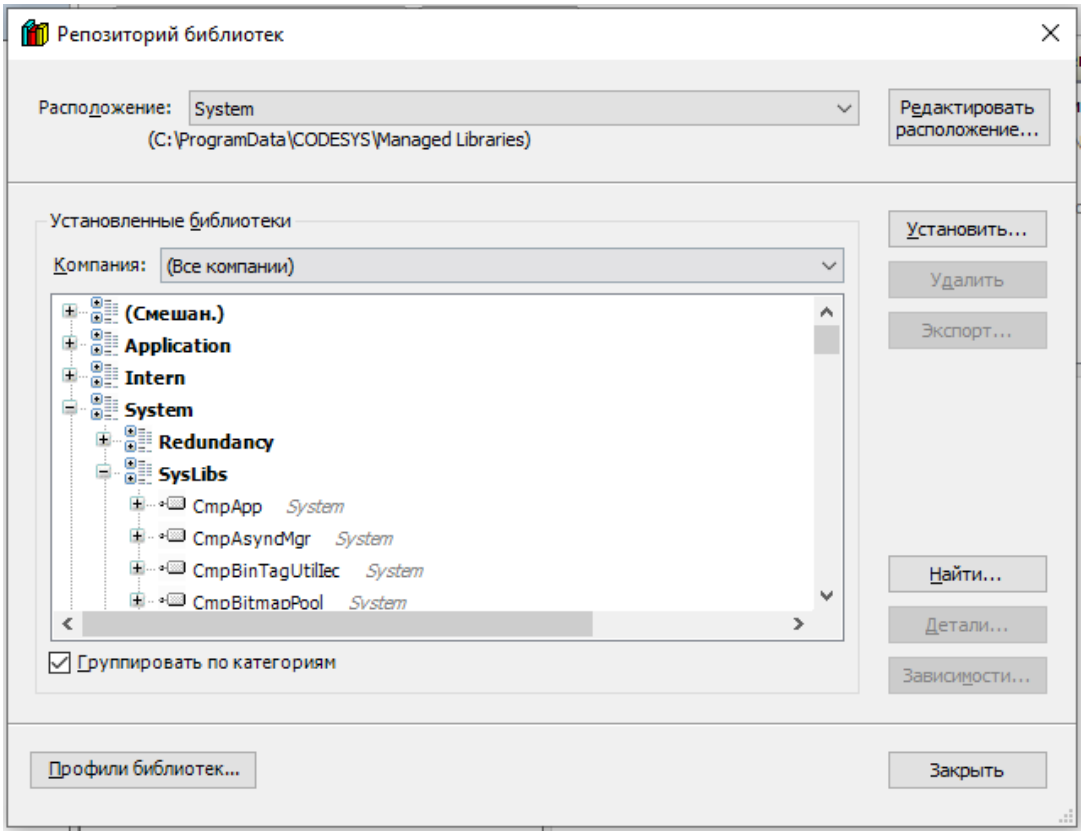


Рисунок 6.52 – Вікно «Репозиторій бібліотек»

Якщо активована опція «Групувати за категоріями», список буде впорядкований відповідно до категорій бібліотек, причому ці категорії можна згортати й розгортати. Якщо опція відключена, бібліотеки будуть впорядковані за алфавітом. Категорії бібліотек задаються зовнішніми файлами опису *.libcat.xml.

На рис. 6.53 показано вікно додавання нової бібліотеки.

Додавати можна тільки бібліотеки, встановлені у вашій системі. До проекту можна водночас включати декілька версій бібліотек.

Можна обрати розширений режим вибору бібліотек, натиснувши на кнопку «Додатково». Зовнішній вигляд вікна додавання бібліотек з розширеними функціями показаний на рис. 6.54.

Якщо активована опція «Групувати за категоріями», то бібліотеки обраної компанії будуть впорядковані за заданими категоріями, в іншому випадку – за алфавітом. Із підключеною опцією категорії будуть показані як вузли, що містять у собі бібліотеки або інші категорії.

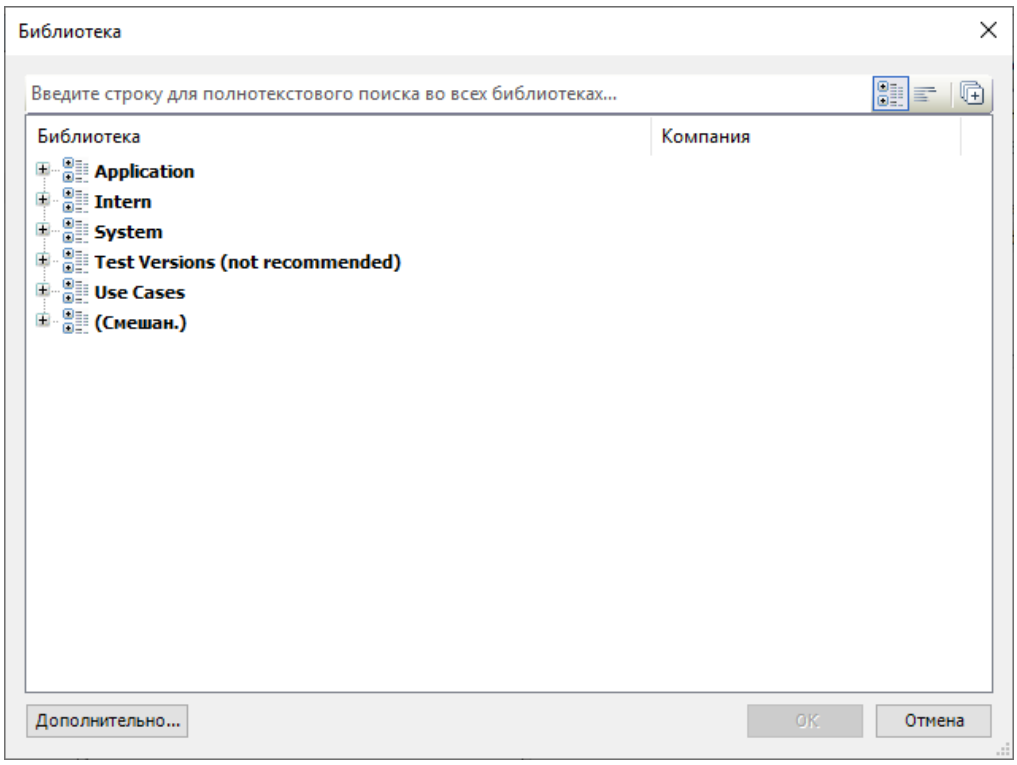


Рисунок 6.53 – Вікно додавання нової бібліотеки

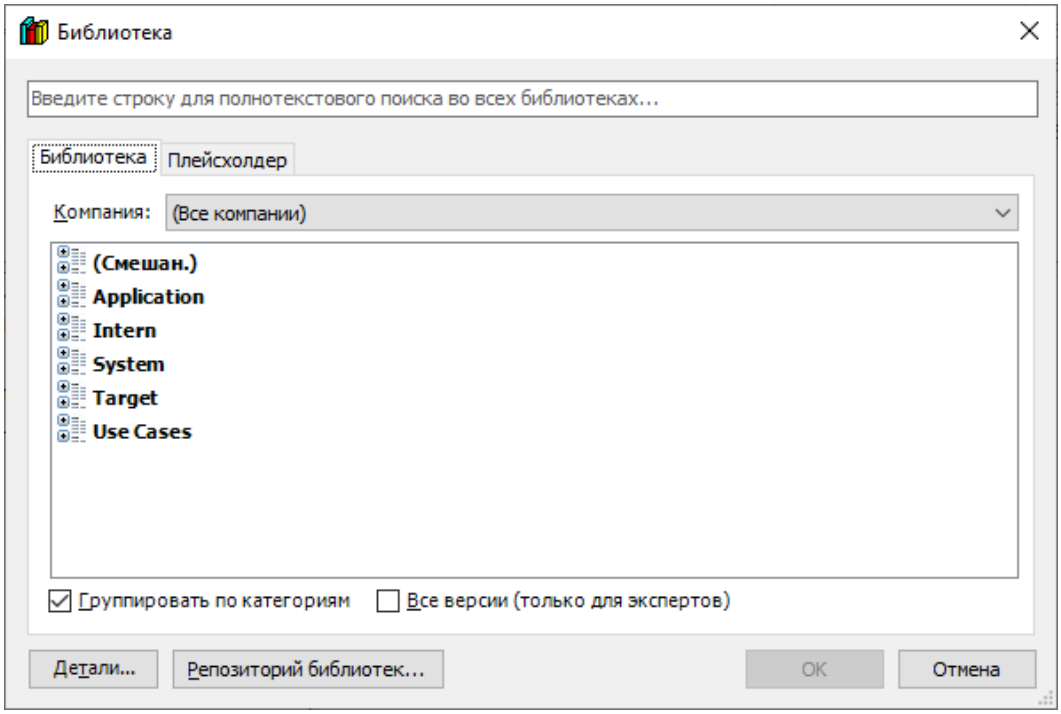


Рисунок 6.54 – Зовнішній вигляд вікна додавання бібліотек
з розширеними функціями

Тут перераховані всі бібліотеки, встановлені у вашій системі. Можна задати відображення бібліотек конкретної Компанії, обравши її зі списку. Якщо вибрано «(Всі компанії)», то будуть показані всі доступні бібліотеки.

Якщо активована опція «Показувати всі версії» (тільки для експертів), то під обраною бібліотекою відображатимуться всі її встановлені версії. Новіша версія в цьому випадку позначається зірочкою («*»). За замовчуванням ця опція відключена, і показуються тільки новіші версії. У цьому випадку можливий вибір кількох бібліотек, для цього потрібно натиснути й утримувати клавішу <Shift> і клацати мишкою по потрібним бібліотекам. Після натискання кнопки ОК вибрані бібліотеки будуть додані у вікно Менеджера бібліотек.

Якщо необхідно включити у проект бібліотеку, яка не встановлена в системі, можна скористатися кнопкою «Репозиторій бібліотек». При цьому відкриється діалог «Репозиторій бібліотек», в якому можна виконати необхідну установку (див. рис. 6.52).

Вкладка «Заповнювач» використовується у двох випадках:

– під час створення проекту, який не залежить від пристрою;

– під час створення проекту бібліотеки <library_xy>, що посилається на іншу бібліотеку, тобто залежить від конкретного пристрою.

Якщо планується використовувати проект для декількох взаємозамінних цільових пристроїв у «Менеджер бібліотек», використовуючи заповнювачі, необхідно підключити бібліотеки для окремих пристроїв.

На рис. 6.55 показано вікно вибору бібліотек з відкритою вкладкою «Заповнювач».

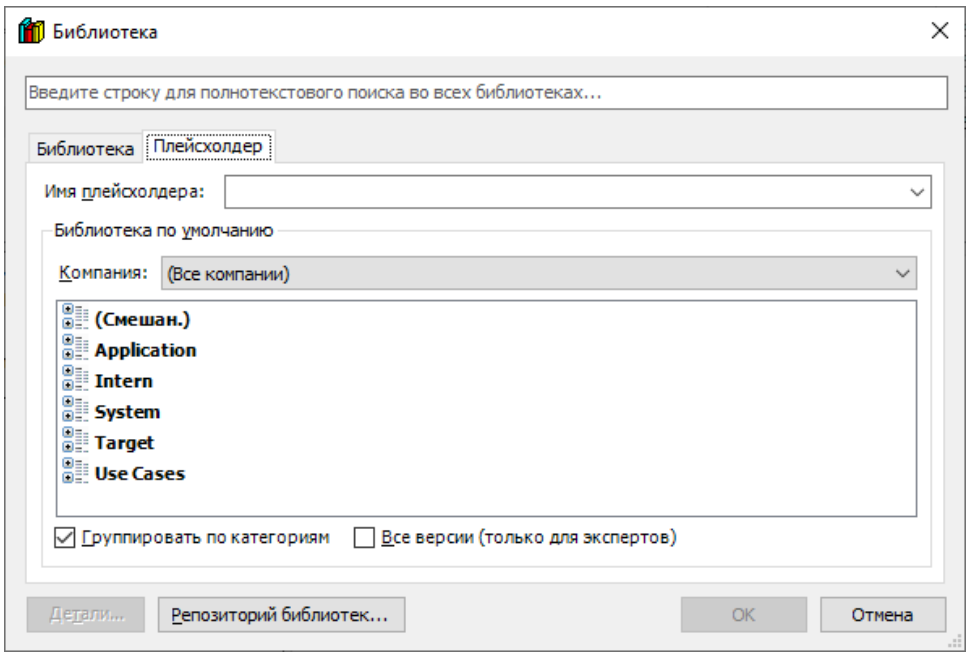


Рисунок 6.55 – Вікно вибору бібліотек з відкритою вкладкою «Заповнювач»

Після того, як задано цільовий пристрій, заповнювачі будуть перетворені у відповідності з описами конкретних пристроїв. Навіть якщо не доступно жодного опису, буде виконана синтаксична перевірка.

Щоб підключити бібліотеку у менеджер бібліотек, використовуючи заповнювач, її необхідно обрати у вікні Бібліотека за замовчуванням в нижній частині діалогу «Заповнювач». При цьому можна скоротити перелік бібліотек, які показуються, залишивши в ньому лише бібліотеки конкретної компанії.

Крім того, у відповідному полі необхідно ввести Ім’я заповнювача. Для вставлення коректного імені можна використовувати перелік, що містить імена всіх заповнювачів, заданих в описах пристроїв. Заповнювач може бути налаштований і для проекту бібліотеки. Якщо проект-бібліотека заснований на інших бібліотеках, залежних від конкретних пристроїв, ці бібліотеки мають бути включені до проекту з використанням заповнювачів.

Це означає, що замість однієї конкретної бібліотеки буде вставлений заповнювач. Коли бібліотека <library_xy> буде використана в іншому проекті для конкретного пристрою, заповнювач буде замінений на ім’я бібліотеки, заданої для конкретного пристрою. Це ім’я має бути задано у відповідному файлі опису пристроїв <library_xy>, який імені «реальної» бібліотеки привласнює ім’я заповнювача.

Якщо з якої-небудь причини менеджер бібліотек не призначено жодному пристрою, заповнювач буде замінено на бібліотеку за замовчуванням, яка задається у цьому діалозі. (Це дозволяє без помилок виконати компіляцію проекту бібліотеки, навіть якщо на даний момент не доступно жодного опису).

У поле «Ім’я заповнювач» можна ввести будь-який рядок і потім, обравши Бібліотеку за замовчуванням. Тут також можна використовувати опцію «Показувати всі версії» (тільки для експертів). Після натискання ОК бібліотека-заповнювач буде вставлена в дерево менеджера бібліотек. Відкривши діалог «Властивості бібліотеки-заповнювача», отримаємо інформацію про задану бібліотеку за замовчуванням.

У наведеному далі прикладі заповнювач ще не замінений ("resolved"); якщо згодом менеджер бібліотек буде поміщений під пристроєм з відповідним файлом опису, в стовпці «Ім’я» з’явиться ім’я бібліотеки на конкретний пристрій.

Команда «Властивості» (категорія «Менеджер бібліотек») за замовчуванням доступна у вікні редактора «Менеджера бібліотек». З використанням цієї команди відкриється діалог «Властивості бібліотеки», обраної у вікні «Менеджер

бібліотек». У ньому можна задавати установки, що стосуються додаткового імені (namespace), управління версіями, доступності та видимості Посилань на бібліотеку. На рис. 6.56 наведено приклад вікна властивостей бібліотеки.

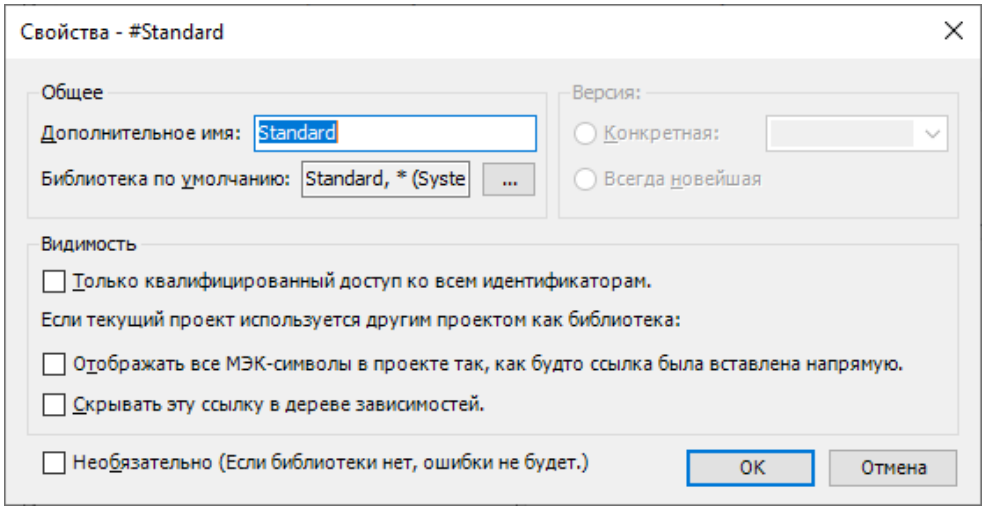


Рисунок 6.56 – Приклад вікна редагування властивостей бібліотеки

Змінні властивості бібліотеки поділяються на декілька розділів.

1. Загальні:

– «додаткове ім’я» – за замовчуванням додаткове ім’я бібліотеки ідентично самому імені бібліотеки, проте, його можна явно задати під час створення проекту в Інформації про проект. Також можна в будь-який момент змінити це ім’я безпосередньо в цьому діалозі;

– «бібліотека за замовчуванням» – якщо в Менеджері бібліотек обрана бібліотека-заповнювач, то в цьому полі буде введене ім’я бібліотеки, яка має замінити заповнювач у випадку, якщо не доступно жодної бібліотеки для конкретного пристрою.

2. Версія:

Тут задається версія бібліотеки, яка використовуватиметься в проекті:

– «дана версія» – буде використана версія, введена тут (можна обрати зі списку);

– «завжди новіша версія» – завжди використовуватиметься новіша версія, знайдена в репозиторії бібліотек. Таким чином, використовувані модулі можуть змінюватися з появою більш нових версій бібліотеки.

3. Видимість:

Ці установки мають значення, коли бібліотека вставляється, тобто викликається іншою бібліотекою. За замовчуванням вони відключені.

Поки опція «відображати всі МЕК-символи у проекті так, ніби посилання було вставлено сюди безпосередньо» відключена, до компонентів поточної бібліотеки можна здійснювати унікальний доступ за рахунок використання відповідного шляху (складається з додаткового імені «батьківської» бібліотеки і власного додаткового імені, а також з ідентифікатора модуля або змінної).

Команда Спробувати перезавантажити бібліотеку (категорія «Менеджер бібліотек») за замовчуванням підключена в меню «Бібліотеки» і в вікно редактора Менеджер бібліотек.

Якщо бібліотека, підключена у проект, за будь-якої причини не доступна в заданій директорії з відкриттям проекту в CoDesys, буде видано відповідне повідомлення про помилку. Коли помилка виправлена, а бібліотека знову доступна, можна застосувати команду «Спробувати перезавантажити бібліотеку», попередньо обравши потрібну бібліотеку в Менеджері бібліотек. Бібліотеку можна перезавантажити, не залишаючи проект.

6.3 Робота з дискретними виходами на прикладі вирішення задач управління світловою сигнальною колоною

6.3.1 Застосування світлових колон на виробництві

Автоматизовані й роботизовані виробництва мають ряд особливостей, що впливають на безпеку. Вони мають у своєму складі найрізноманітніші пристрої і машини, в тому числі численні транспортні системи. Гнучкі автоматизовані виробництва (ГАВ) займають значні площі. Обладнання, що входить до їх складу, може займати різні рівні за висотою приміщення, розміщуючись навіть на двох і більше поверхах, при цьому управління ведеться з одного пульта, одним або більшою кількістю постачальників. При цьому змінився характер праці, для працюючого на ГАВ переважають форми розумової праці, праця стає менш важкою, але більш напруженою.

Забезпечення безпеки й комфортності праці в гнучкій виробничій системі (ГВС) має ряд особливостей. У тому числі:

– автоматичні пристрої, роботи, електронні системи – можуть виходити з ладу, створюючи небезпечні ситуації, як правило, поза візуальним контролем людини безпосередньо в місці поломки;

– ГВС мають у своєму складі різноманітні пристрої і машини, в тому числі численні транспортні системи;

– ГВС займають значні площі, тому обладнання, що входить до їх складу, може займати різні рівні за висотою приміщення, навіть на два й більше поверхів, при цьому управління ведеться з одного пульта одним або більшою кількістю приладів керування;

– під час налагодження, переналагодження, програмування засобів управління, а також у ході профілактичних і ремонтних робіт працівнику доводиться перебувати в робочих зонах обладнання, у тому числі в зоні рухомих частин.

Застосування пристроїв автоматичного контролю і сигналізації (інформаційні, попереджувальні, аварійні) – найважливіша умова безпечної і надійної роботи обладнання. Пристрої контролю – це прилади для вимірювання тиску, температури, статистичних і динамічних навантажень та інших параметрів, що характеризують роботу устаткування і машин.

Для виключення зіткнення промислових роботів з людиною вони оснащуються безконтактними датчиками, які виявляють присутність людини в безпосередній близькості від рухомих частин. Датчики викликають спрацьовування системи зупинки робота. Ефективність використання датчиків контролю значно підвищується в ході об’єднання з системами сигналізації (звуковими, світловими, колірними, знаковими або комбінованими).

Сигналізаційні пристрої слугують для інформування персоналу про появу виробничої небезпеки і про роботу технологічного обладнання.

Одним з найпоширеніших пристроїв світлової та звукової сигналізації є світлова колона, яка показана на рис. 6.57.

Світлодіодна сигнальна колона слугує для світлової та звукової сигналізації стану обладнання, подачі дозвільних або заборонних світлових сигналів і попередження персоналу про аварійні та надзвичайні ситуації, пов’язані з безпечною роботою обладнання.

Світлосигнальні колони використовуються для візуального та звукового оповіщення персоналу про стан технологічного процесу. Колони застосовуються у різних галузях промисловості і можуть видавати сигнали різного типу й тривалості – мигалки, спалахи, постійне горіння, поворотні сирени тощо залежно від важливості сигналу. Світлодіодні сигнальні колони одноколірні й багатоколірні, із звуковою сигналізацією або без неї застосовуються на конвеєрах та інших транспортних системах,

термопластавтоматах і екструдерах, фасувальному і пакувальному обладнанні, на обладнанні, де потрібні підвищені заходи безпеки обладнання.

Рисунок 6.57 – Зовнішній вигляд світлової колони

Світлові й звукові сигнали є основними елементами безпеки системи. Для забезпечення коректного тлумачення, за Європейськими стандартами прийнято використання звукових і візуальних пристроїв у системах.

Сигнальні колони можуть мати до семи модулів, послідовно зібраних з таких колірних елементів: червоний, жовтий, помаранчевий, синій, зелений і білий.

Кожен колір або звуковий сигнал вказує на подію в робочій системі і сигналізує про рівень небезпеки відповідно до стандартів за EN 981/EN 60073. Модуль білого кольору не має специфічного значення і використовується користувачем на свій розсуд. У таблиці 6.2 наведено пояснення щодо кольорової індикації, що відображається даними пристроями.

Окремі елементи колони можуть мати такі виконання:

– елемент, що постійно світиться (лампа розжарювання, світлодіод);

– миготливий елемент (лампа розжарювання, світлодіод);

– світловий елемент з одним спалахом;

– елемент, що обертається (світлодіод);

– звуковий елемент (гудок);

– звуковий елемент (сирена).

Таблиця 6.2 – Пояснення щодо кольорової індикації, що відображається даними пристроями

Візуальний код	Червоний	Жовтий	Помаранчевий	Зелений	Білий
Значення	Небезпека. Аварія	Попередження і увага Ненормальна ситуація	Обов'язкова команда	Нормальна ситуація. Звичайна робота	Немає специфіч-ного значення
Звук	Часта модуляція повторення високої частоти	Короткий сигнал, повторюється	Змінний сигнал постійного струму	Постійний довгий сигнал після аварії	Інші звуки
Дія	Негайне втручання для усунення аварії	Необхідне контрольне втручання	Необхідне втручання для виконання команди	Втручання не потрібне	Залежно від обставин

Сигнальні колони можуть бути безпосередньо підключені до шинної системи, наприклад, AS-Interface через адаптер, який може бути вбудованим. Це зменшує витрати на передачу даних. Двожильний кабель фіксується в гвинтових затисках у сполучному елементі.

Адаптер має бути першим модулем, який буде підключений у сполученому елементі. Максимальна кількість наступних сигнальних елементів може бути не більше чотирьох.

Загальна архітектура взаємодії між модулями виробничої системи з використанням блока світової та звукової сигналізації показана на рис. 6.58.

Як можна бачити з рис. 6.58, промислова ділянка складається з декількох верстатів (промислове обладнання), логічного контролера, що програмується, промислової мережі, блоків світлової та звукової сигналізації.

Кожен верстат має свою робочу зону, де виконуються технологічні операції. Доступ до даної зони контролюється набором датчиків. З появою людини в межах робочої зони датчики надсилають тривожні сигнали до контролера промислового обладнання, а воно, в свою чергу, через промислову мережу до логічного контролера, що програмується.

Рисунок 6.58 – Загальна архітектура взаємодії
між модулями виробничої системи
з використанням блока світової та звукової сигналізації

6.3.2 Опис конструкції світлової колони

Блоки світлової та звукової сигналізації можуть підключатися безпосередньо до обладнання, або до промислової мережі.

Залежно від рангу датчика, що спрацював, приймається рішення про рівень сигналізації.

Структурна схема модуля керування блоком світлової та звукової сигналізації подана на рис. 6.59.

Як можна бачити з наведеного рисунку, до складу модуля керування входять:

– мікроконтролер;

– блоки узгодження з вхідними контактами;

– блоки узгодження з пристроєм індикації;

– блоки індикації;

– блок узгодження з пристроєм випромінювання звуку;

– блок звукової сигналізації;

– перетворювач інтерфейсу.

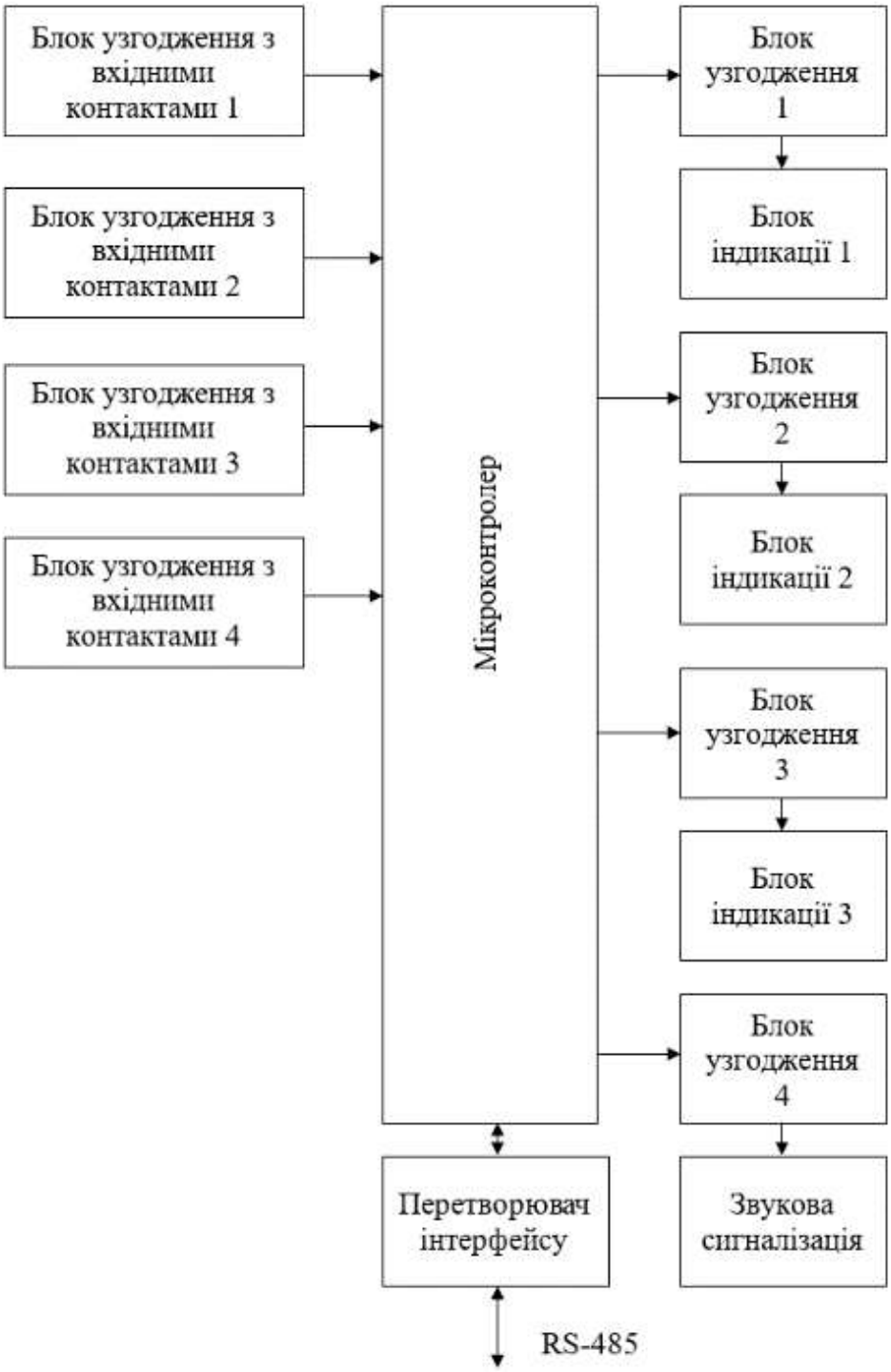


Рисунок 6.59 – Структурна схема модуля керування блоком світлової та звукові сигналізації

Мікроконтролер приймає та обробляє команди від головного контролера, або персонального комп'ютера. Залежно від прийнятої команди, виконує вмикання відповідних блоків звукової, або світлової індикації.

Перетворювач інтерфейсів виконує функцію узгодження рівнів сигналів мережі RS-485 та транзисторної логіки мікроконтролера.

Через цей блок до даного модуля світлової та звукової індикації надходять сигнали керування.

Блоки узгодження із вхідними контактами виконані на базі оптичних перетворювачів. Вони призначені для захисту входів мікроконтролера від виходу з ладу за неправильного підключення кнопок керування.

Блоки узгодження з пристроєм індикації призначені для підключення потужного світлового приладу до виходів мікроконтролера. Вони виконані на потужних транзисторних ключах.

Блок індикації виконаний на дискретних світлодіодах, що запаяні на гнучкі друковані плати, та скручені за периметром кольорового стакану, в який вони вставляються.

Блок звукової сигналізації виконано на спеціалізованому звуковому пристрої BUZZER.

Для керування блоком звукової сигналізації також використовується потужний транзистор, який увімкнено за ключовою схемою.

На рис. 6.60 показана конструкція навчального макету, який використовуватиметься для тестування програми керування світловою колоною.

До складу макету входять:

– блок індикації червоного кольору (1);

– блок індикації жовтого кольору (2);

– блок індикації зеленого кольору (3);

– складна стійка (4);

– основа для кріплення (5);

– стопорний гвинт (6);

– кронштейн (7);

– блок керування (8).

Конструкція пристрою передбачає використання його як у настільному варіанті (рис. 6.60), так і у випадку кріплення на DIN-рейку (рис. 6.61). Для зміни варіанта кріплення передбачено рухомий механізм зміни орієнтації стійкі 4. Стопорний гвинт 6 може фіксувати основу для кріплення 5 в одному з двох положень.

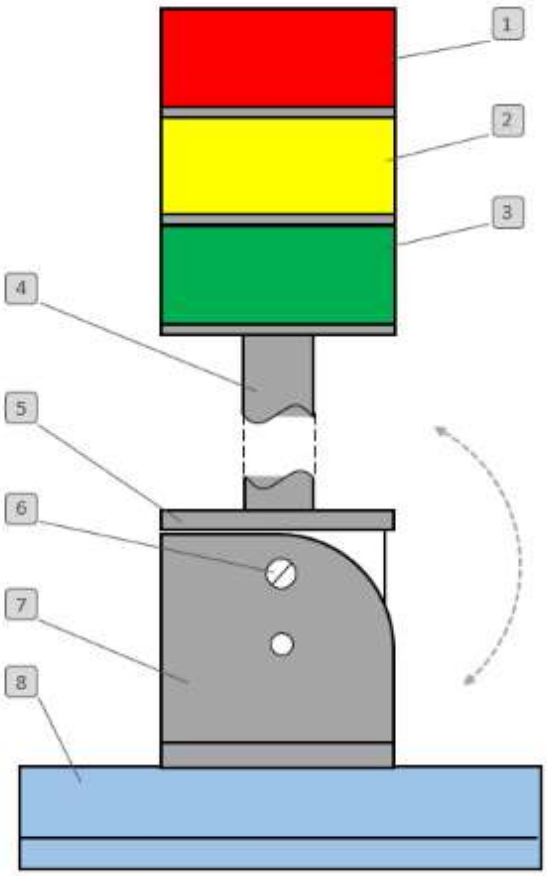


Рисунок 6.60 – Конструкція навчального макету «Світлова колона»

На рис. 6.61 показано варіант кріплення пристрою на DIN-рейку.

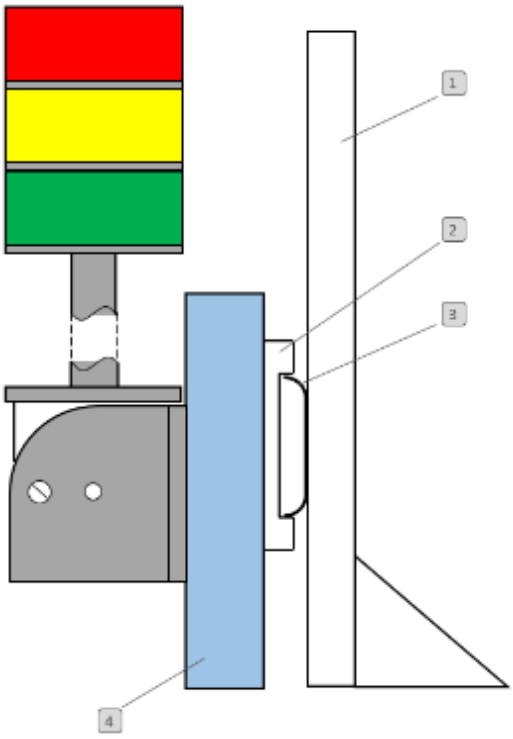


Рисунок 6.61 – Варіант кріплення пристрою на DIN-рейку

З рис. 6.61 можна бачити, що модуль керування 4 за допомогою скоби 2 кріпиться на DIN-рейку 3, яка в свою чергу, закріплена на жорсткій основі 1.

Даний варіант використовується для стаціонарного розміщення макету, наприклад, у навчальній лабораторії.

Для підключення пристрою до ПЛК використовується ряд комутаційних пристроїв – клемників, що розташовані з двох боків модуля керування, як показано на рис. 6.62.

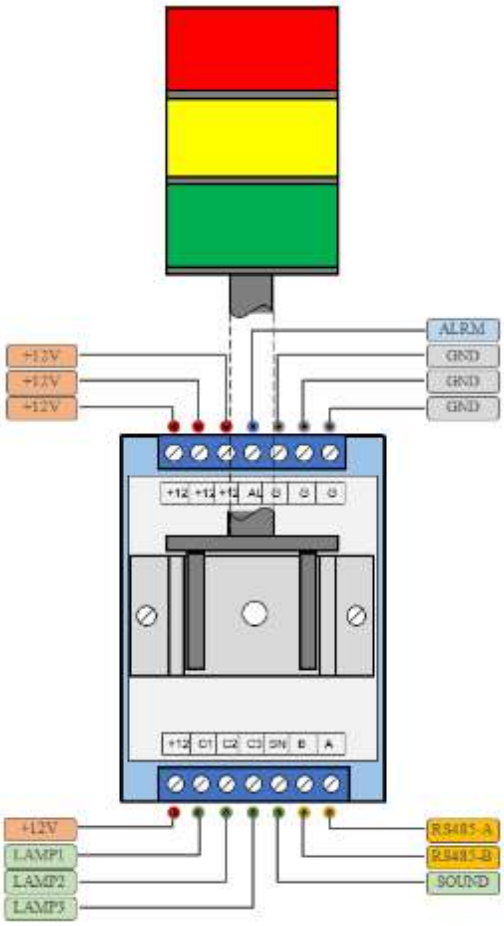


Рисунок 6.62 – Схема розташування контактів на клемниках блоку керування

На клемники виведено контакти підключення:

– напруги живлення 12 В;

– зовнішнього сигналу звукової сигналізації (ALRM);

– входу керування сигнальною лампою 1 (червоний);

– входу керування сигнальною лампою 2 (жовтий);

– входу керування сигнальною лампою 3 (зелений);

– контакту A інтерфейсу RS-485;

– контакту B інтерфейсу RS-485;

– входу керування звуковою сигналізацією (SOUND).

Пристроєм можна керувати як за допомогою команд, які передаються промисловою мережею через інтерфейс RS-485, або безпосередньо підключенням кнопок, або виходів ПЛК до вхідних контактів пристрою.

Схема підключення кнопок керування до модуля керування світловою колоною зображена на рис. 6.63.

Для роботи модуля керування потрібне зовнішнє джерело живлення напругою 12 В. Воно підключається до відповідних контактів правого з'єднувача (рис. 6.63).

Вхідна частина модуля живиться від ізольованого джерела, що підключається позитивним полюсом до контакту «+12 В» лівого з'єднувача. Кнопки керування мають замикати входи LAMP 1 – LAMP3, або SUND до мінусу даного джерела живлення.

Для керування макетом використовуватимемо ПЛК 110-60 фірми ОВЕН.

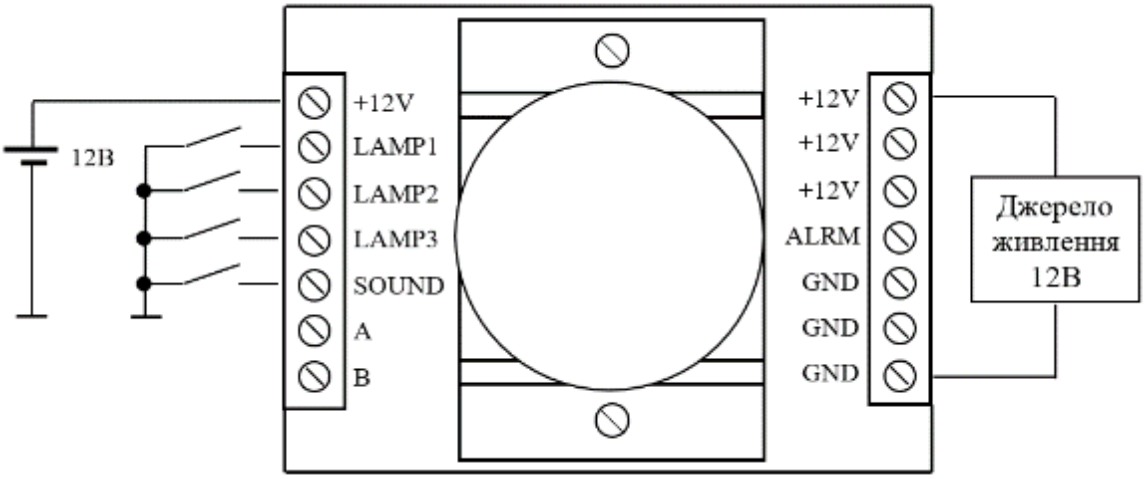


Рисунок 6.63 – Схема підключення зовнішніх контактів для керування світловою колоною

На рис. 6.64 показана схема розташування вхідних та вихідних контактів на корпусі пристрою.

Згідно з рис. 6.63, схема підключення макету світлової колони до ПЛК 110-60 показана на рис. 6.65.

Релейні виходи підключаються до загальної шини та до мінусу джерела живлення. Кожен з виходів DO1 – DO4 підключається до окремого входу модуля керування світловою колоною:

– DO1 підключено до контакту SOUND;

– DO2 підключено до контакту LAMP3 (зелений);

– DO3 підключено до контакту LAMP2 (жовтий);

– DO4 підключено до контакту LAMP1 (червоний);

Для живлення модуля керування використовується окреме джерело напругою 12 В.

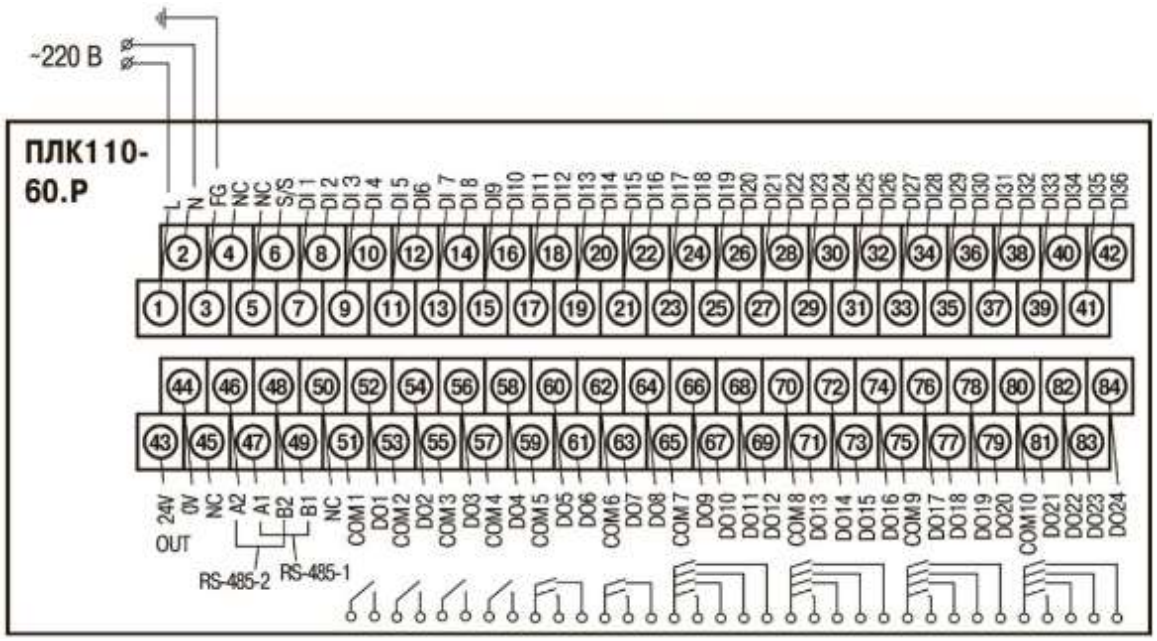


Рисунок 6.64 – Схема розташування вхідних і вихідних контактів на корпусі пристрою

6.3.3 Розробка простої програми керування світловою колоною

Перша програма керування світловою колоною вирішуватиме задачу управління трафіком переміщення мобільних платформ у виробничому приміщенні за простим алгоритмом:

– крок 1 – вмикається червоне світло на 45 секунд;

– крок 2 – на 2 секунди вмикається жовте світло;

– крок 3 – вмикається зелене світло на 45 секунд;

– крок 4 – на 2 секунди вмикається жовте світло;

– крок 5 – перехід на крок 1.

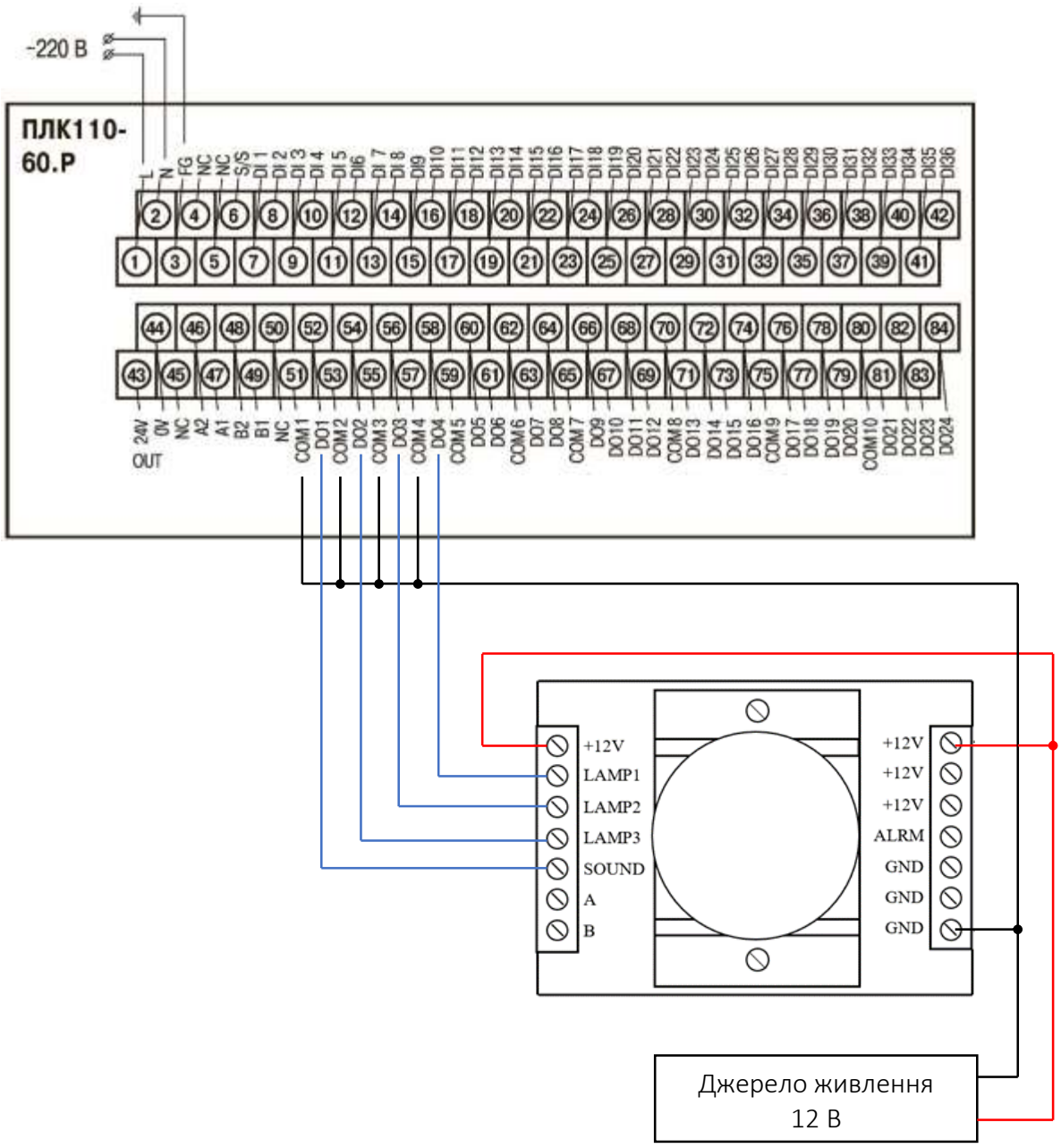


Рисунок 6.65 – Схема підключення макету світлової колони до ПЛК 110-60

На рис. 6.66 показана діаграм стану автомату перемикання кольору в світлофорі.

На початку створення проекту відкриваємо CODESYS та в меню File натискаємо кнопку New. Далі слід обрати цільову платформу (пристрій), де виконуватиметься наша програма. Зі списку, який випадає, обираємо PLC110.60-M, як показано на рис. 6.67.

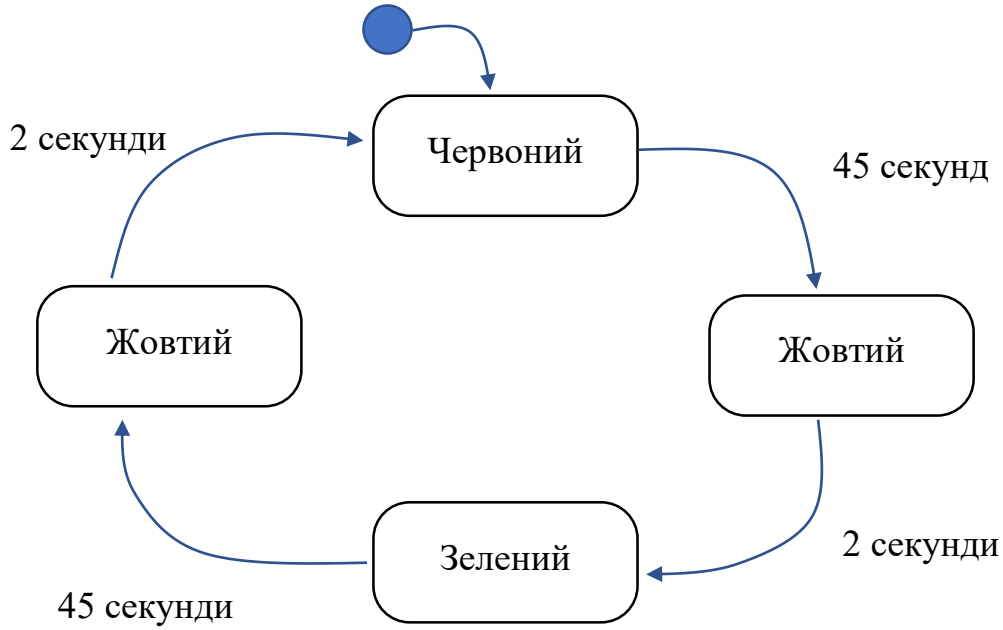


Рисунок 6.66 – Діаграма стану автомату перемикання кольору в світлофорі

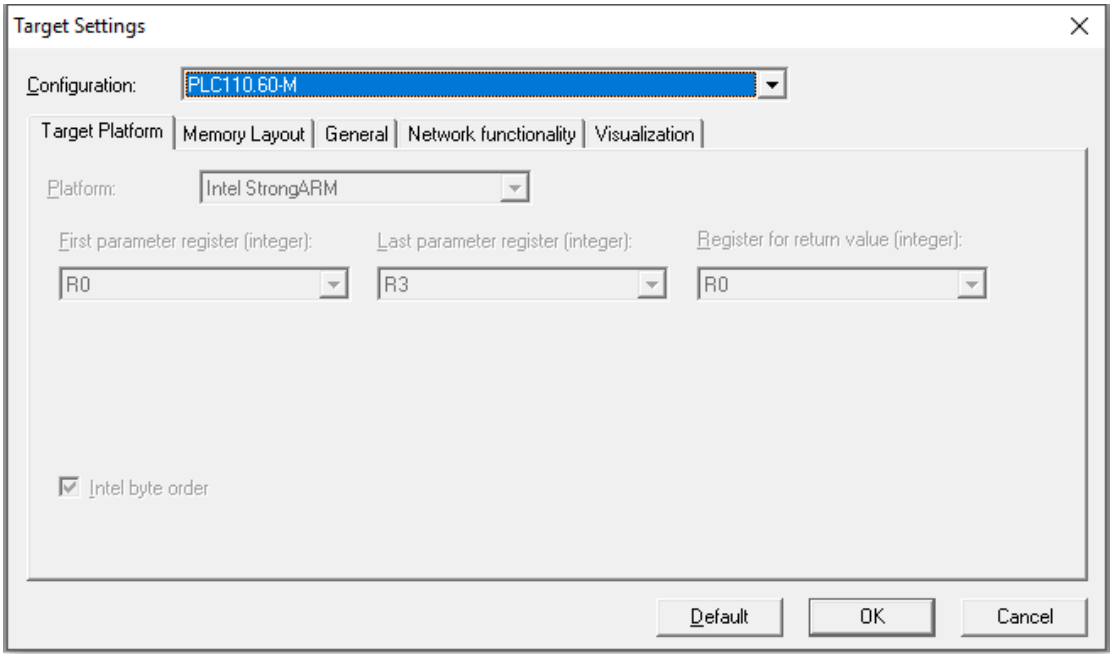


Рисунок 6.67 – Обирання цільової платформи

Після цього з’являється вікно, в якому нам пропонується створити POU PLC_PRG – головний будівельний блок нашої програми. PLC_PRG – це програма, яка викликається в кожному циклі контролера. Обираємо мову програмування FBD, як показано на рис. 6.68.

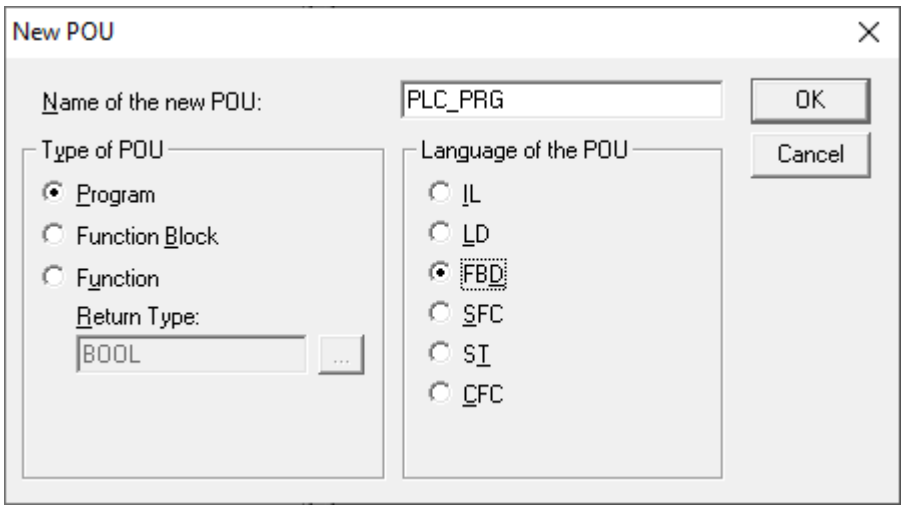


Рисунок 6.68 – Обирання мови написання програми

На даному етапі наш проект матиме такий вигляд (див. рис. 6.69):

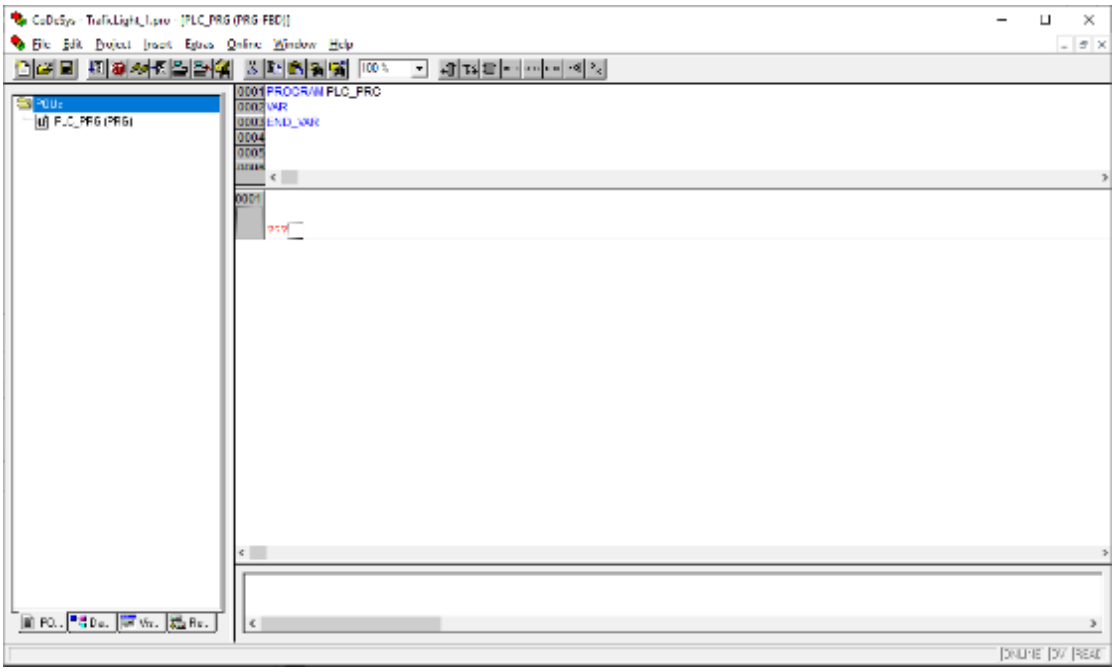


Рисунок 6.69 – Початок створення проекту

Для створення нашої програми насамперед необхідно створити функціональний блок для перемикання сигналів світлофора. Оскільки даний блок використовуватиметься також і в інших програмах, його необхідно зробити універсальним. Необхідно передбачити можливість задання будь-якого часу роботи червоного та зеленого світла.

Таким чином функціональний блок TraficLight матиме такий інтерфейс (рис. 6.70) .

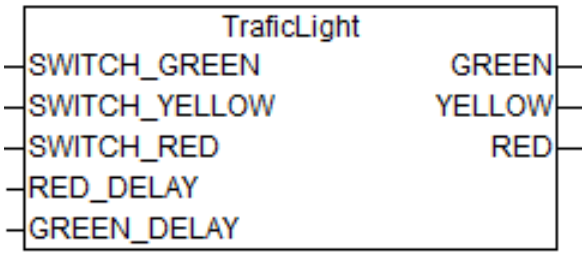


Рисунок 6.70 – Інтерфейс функціонального блоку TraficLight

Вхід SWITCH_GREEN вмикає зелений сигнал світлофора, SWITCH YELLOW – жовтий, SWITCH RED – червоний. Після вмикання зеленого сигналу він горітиме протягом часу GREEN_DELAY, а червоний сигнал горить протягом часу RED_DELAY. Час роботи жовтого сигналу не подається на вхід, оскільки він фіксований і становить 2с.

Виходи GREEN, YELLOW та RED включають сигнали відповідних кольорів.

Тепер опишемо логіку роботи даного функціонального блоку мовою FBD. Щоб керувати світлофором, нам знадобиться таймер TP, який входить у бібліотеку standard.lib. Ця бібліотека приєднується до проекту автоматично під час його створення.

На рис. 6.71 показано зовнішній вигляд функціонального блоку генератора імпульсу PT.

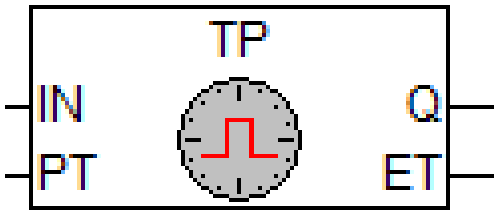


Рисунок 6.71 – Зовнішній вигляд функціонального блоку генератора імпульсу PT

Запуск таймера відбувається за фронтом імпульсу на вході IN. Вхід PT задає тривалість формованого імпульсу. Після запуску таймер не реагує на зміну значення входу IN. Вихід ET відраховує час, що минув. З досягненням ET значення PT лічильник зупиняється, і вихід скидається в 0.

Логіка роботи таймера показана на рис. 6.72.

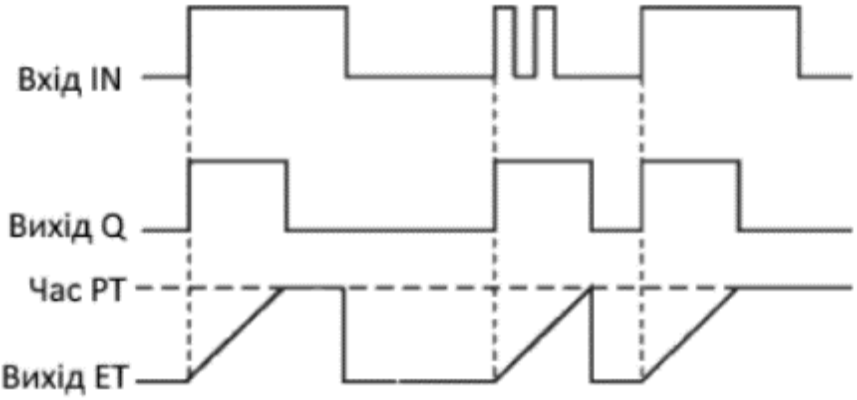


Рисунок 6.72 – Часова діаграма роботи таймера TP

Використовуючи такий таймер, можемо описати роботу світлофора за допомогою трьох FBD-схем. Для цього створимо три блоки змінних: вхідні змінні, вихідні змінні та внутрішні змінні. Наприклад:

```
PROGRAM PLC_PRG
VAR_INPUT
    SWITCH_GREEN:     BOOL;
    SWITCH_YELLOW:    BOOL;
    SWITCH_RED:       BOOL;
    RED_DELAY:        TIME;
    GREEN_DELAY:      TIME;
END_VAR
VAR_OUTPUT
    GREEN:                  BOOL;
    YELLOW:                 BOOL;
    RED:                    BOOL;
END_VAR
VAR
    TOF_GREEN:              TP;
    TOF_YELLOW    :         TP;
    TOF_RED:                TP;
END_VAR
```

Логіка роботи функціонального блоку показана на рис. 6.73.

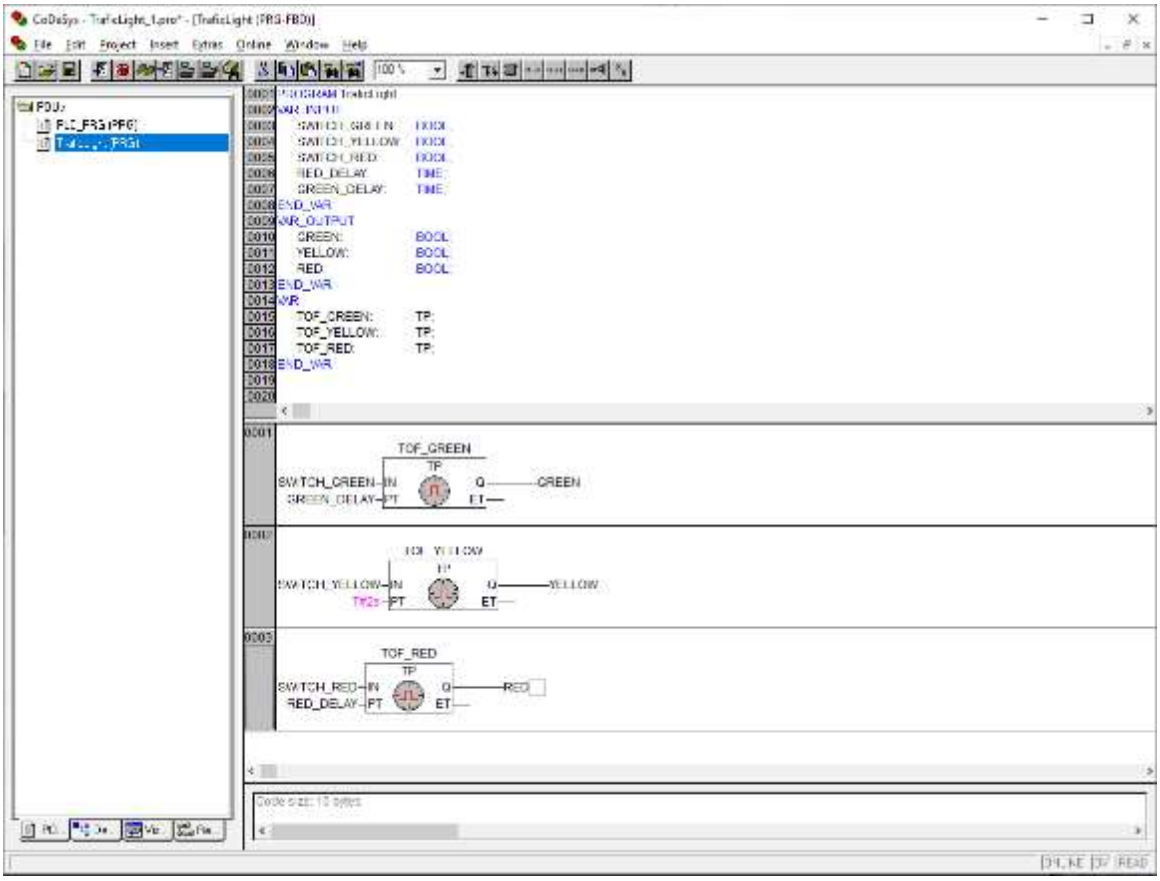


Рисунок 6.73 – Логіка роботи функціонального блоку TraficLight

Як можна бачити з даного рисунку, функціональний блок складається з двох розділів:

– з розділу оголошень;

– зі схем, які описують логіку його роботи.

У розділі оголошення описуються входи-виходи функціонального блоку, а також внутрішні змінні. В даному випадку внутрішніми змінними є екземпляри функціональних блоків TP – по одному на кожен сигнал.

Для реалізації діаграми станів, яка зображена на рис. 6.66, використовуватимемо такі функціональні блоки:

– таймер з затримкою увімкнення TON;

– блок порівняння GT (більше);

– блок порівняння LT (менше);

– блок логічного оператора AND;

– блок логічного оператора OR.

Зовнішній вигляд блоку таймера з затримкою TON показано на рис. 6.74.

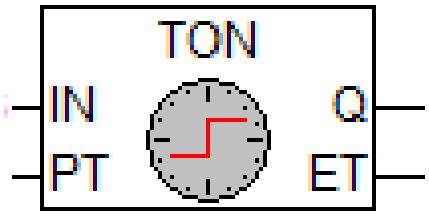


Рисунок 6.74 – Зовнішній вигляд блоку таймера з затримкою TON

За фронтом входу IN виконується скидання лічильника і починається новий відлік часу. Вихід Q буде встановлений у TRUE через заданий PT час, якщо IN продовжуватиме залишатися у стані TRUE. Спад входу IN зупиняє відлік і скидає вихід Q в FALSE. Таким чином, вихід Q вмикається логічною одиницею тривалістю не менше PT, а вимикається за спадом входу IN.

Часова діаграма роботи таймера TON показана на рис. 6.75.

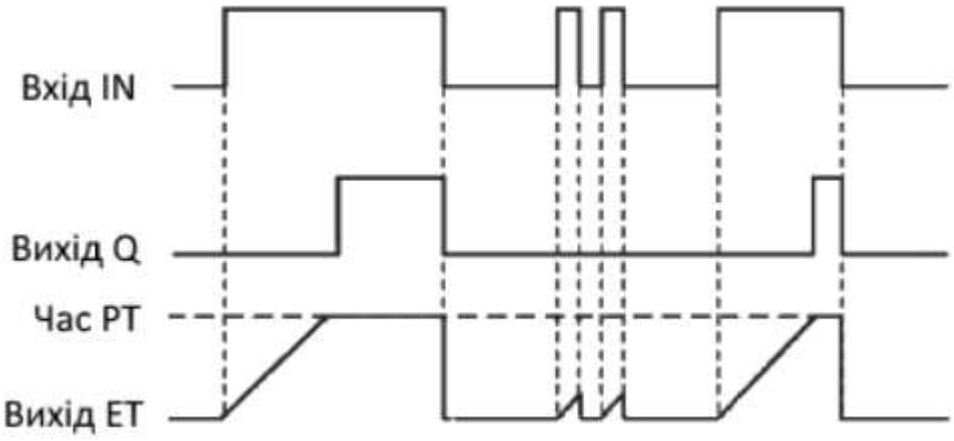


Рисунок 6.75 – Часова діаграма роботи таймера TON

Для нашої задачі даний блок вироблятиме часові імпульси на виході ET, доки значення не дорівнюватиме заданому часу на вході PT. Для циклічної роботи таймера встановимо максимальне значення часу на рівні 120 секунд. Для скидання таймера слід створити на вході IN умову, за якою на ньому підтримуватиметься значення TRUE в межах часу всіх стадій світлофору. Ця умова запишеться так: tProcessTime < T#94S.

На рис. 6.76 наведено приклад функціонального блоку TON після виконання настроювання.

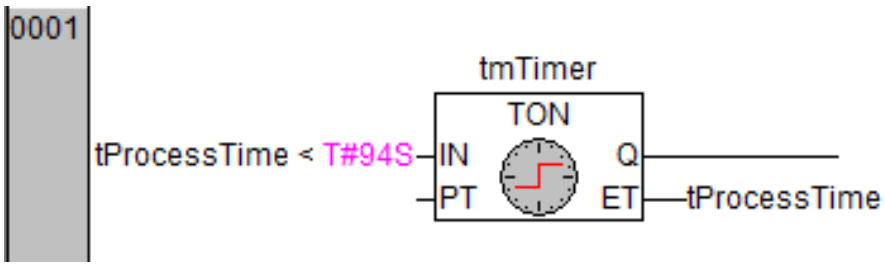


Рисунок 6.76 – Функціональний блок TON після виконання настроювання

Блок порівняння GT використовується для порівняння двох параметрів A та B, як показано на рис. 6.77.

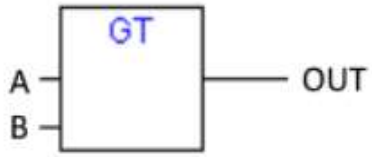


Рисунок 6.77 – Функціональний блок GT

Двійковий GT-оператор повертає TRUE, якщо значення першого параметра більше другого.

Блок порівняння LT використовується для порівняння двох параметрів A та B, як показано на рис. 6.78.

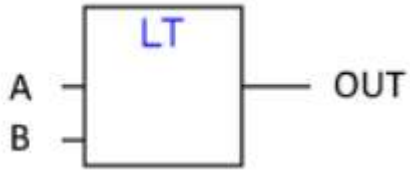


Рисунок 6.78 – Функціональний блок LT

Двійковий LT-оператор повертає TRUE, якщо значення першого параметра менше другого.

На рис. 6.79 наведено приклади функціональних блоків для виконання побітових операторів AND та OR.

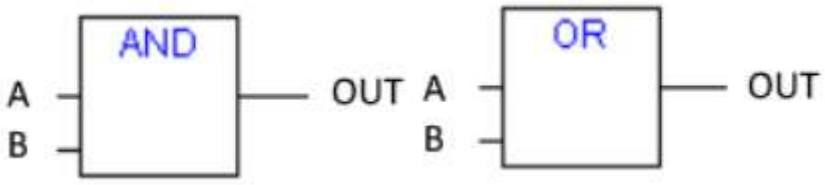


Рисунок 6.79 – Приклади функціональних блоків
для виконання побітових операторів AND та OR

Для реалізації заданого алгоритму побудуємо циклограму управління світлофором, яка зображена на рис. 6.80.

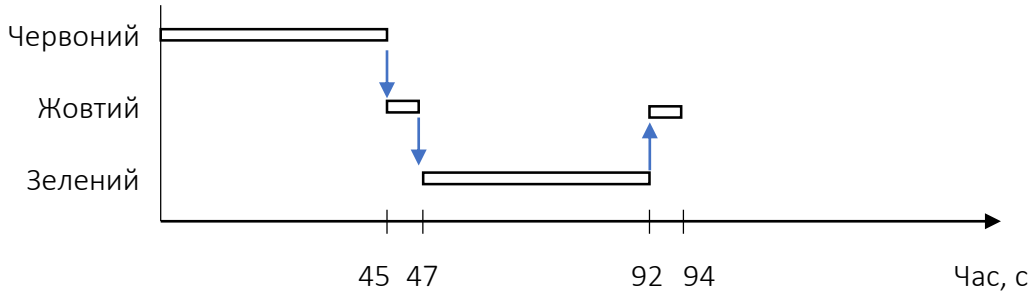


Рисунок 6.80 – Циклограма управління світлофором

Після підключення всіх блоків загальна схема матиме вигляд, як показано на рис. 6.81.

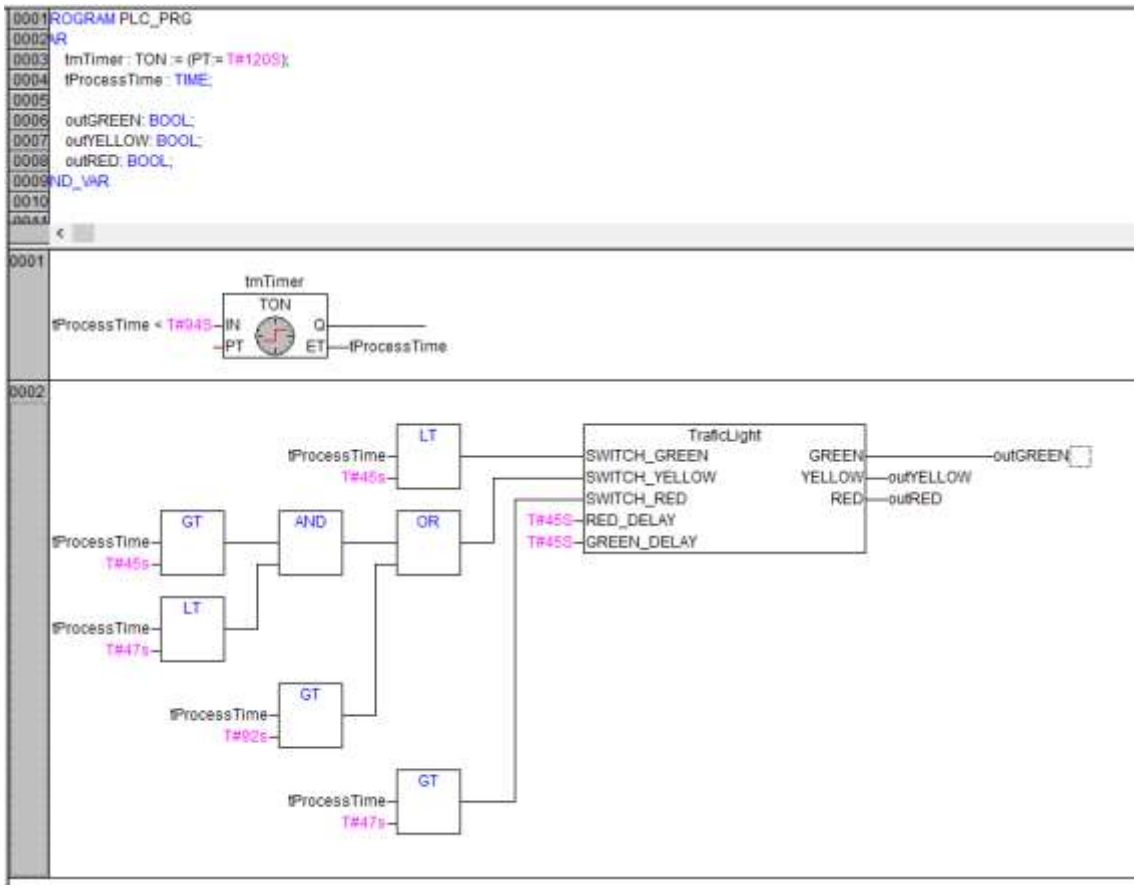


Рисунок 6.81 – Загальна схема управління світлофором

Якщо слід модифікувати алгоритм перемикання світлофора і зробити так, аби жовтий колір вмикався з закінченням часу світіння червоного (на останніх двох секундах) то циклограма матиме вигляд, як показано на рис. 6.82.

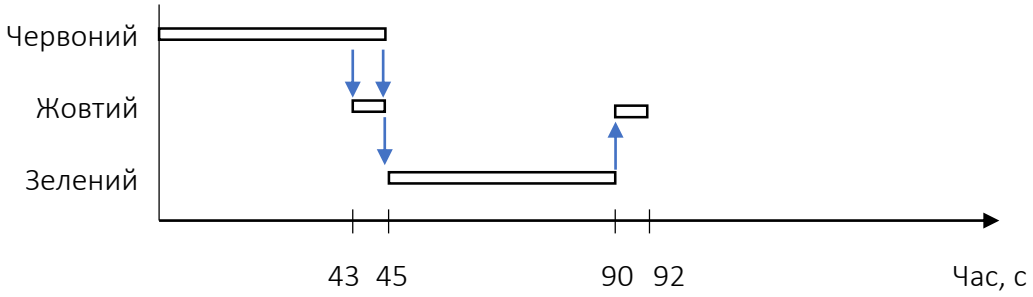


Рисунок 6.82 – Циклограма управління світлофором

Після зміни алгоритму роботи програми зміниться і загальний час на виконання всієї програми, який становитиме 92 с.

6.3.4 Розробка програми керування групою світлових колон

Розглянемо приклад вирішення задачі, коли рухом транспорту керує кілька світлофорів, що розташовані на перехресті доріг. На рисунку 6.83 показано приклад розташування світлофорів.

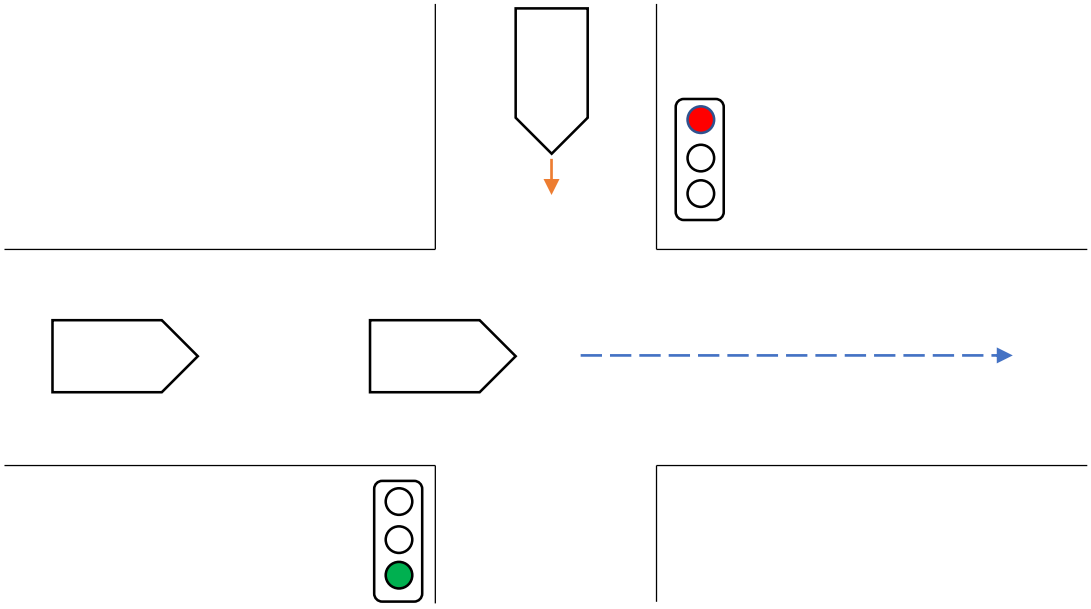


Рисунок 6.83 – Схема розташування світлофорів

Постановка задачі. Є перехрестя двох доріг. Рух мобільних платформ на перехресті управляється двома світлофорами, кожен з яких дозволяє або

забороняє рух платформ по одному з напрямків. Світлофори працюють у протифазі – тобто коли світлофор А дозволяє рух, світлофор В забороняє.

Інтенсивність руху кожною з доріг різна в різний час роботи цеху. Наприклад, якщо інтенсивність руху по дорозі А більше, ніж по дорозі В, то слід зробити так, щоб зелений сигнал світлофора А горів довше, ніж у світлофора В. Це дозволить підвищити пропускну здатність перехрестя. Також потрібно зробити так, щоб період роботи світлофора змінювався б динамічно залежно від поточної ситуації на дорозі.

Для створення даної програми використовуватимемо функціональний блок TraficLight, але дамо йому іншу назву TRAFFICSIGNAL.

Для вирішення цього завдання використовуватимемо спрощену мову SFC – вона підходить для задач, в яких дії виконуються послідовно.

У розділі оголошень потрібно оголосити дві копії функціонального блоку TRAFFICSIGNAL: TS1 та TS2 (рис. 6.84).

```
0001 PROGRAM PLC_PRG
0002 VAR
0003     TS1:TRAFFICSIGNAL;
0004     TS2:TRAFFICSIGNAL;
0005
```

Рисунок 6.84 – Створення об’єктів типу TRAFFICSIGNAL

Об’єкт TS1 використовуватиметься для управління світлофором А, а TS2 використовуватиметься для управління світлофором В.

За замовчуванням схема SFC складається з одного кроку INIT. Додамо ще чотири та створимо послідовність дій, що складається з п’яти стадій, як показано на рис. 6.85.

Кожен крок описує одну стадію роботи світлофорів.

Почнемо з кроку INIT. Створимо опис дії даного кроку:

```
TS1.GREEN_DELAY:=T#6s;
TS1.RED_DELAY:=T#6s;
TS2.GREEN_DELAY: =T#6s;
TS2.RED_DELAY: = T#6s;
```

Тут ми задали час увімкнення кожного з сигналів світлофорів.

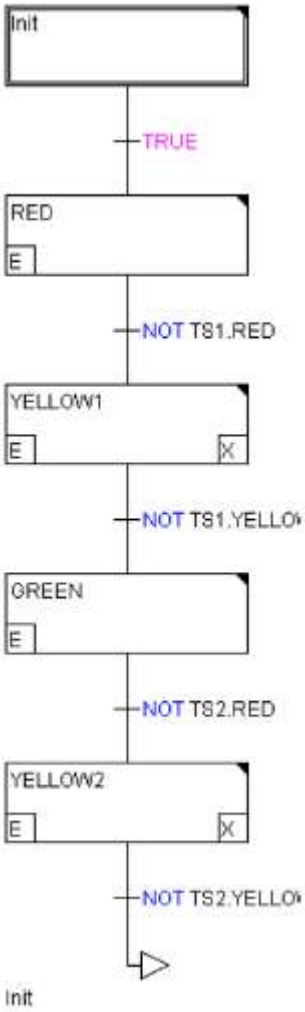


Рисунок 6.85 – Послідовність дій на діаграмі SFC

Наступний крок RED включає червоний сигнал світлофора A і зелений сигнал світлофора B. Для реалізації цього кроку нам знадобиться дві дії – вхідна й основна. Вхідна дія виконується один раз із входом у крок. Основна – в кожному циклі, поки не буде досягнута умова переходу. У вхідній дії ми просто підключаємо потрібні сигнали світлофора:

```
TS1(SWITCH_GREEN:=FALSE,
SWITCH_YELLOW:=FALSE,
SWITCH_RED:=TRUE);
TS2(SWITCH_GREEN:=TRUE,
SWITCH_YELLOW:=FALSE,
SWITCH_RED: =FALSE);
```

В основній дії функціональний блок TS1 викликається періодично, до тих пір, доки таймер не вимкне червоний сигнал світлофора A.

Крок Yellow1 складається з трьох дій. Вхідний крок вмикає жовтий сигнал світлофора А:

```
TS1(SWITCH_GREEN:=FALSE,
SWITCH_YELLOW:=TRUE,
SWITCH_RED:=FALSE);
```

В основному кроці викликається TS1 до тих пір, доки жовтий сигнал світлофора А не вимкнеться.

Вихідна дія виконується, коли перехід вже спрацював. У ньому вимикається зелений сигнал світлофора В:

```
TS2(SWITCH_GREEN:=FALSE);
```

Крок GREEN вмикає зелений сигнал світлофора А і червоний сигнал світлофора В. Для цього у вхідній дії кроку прописуємо:

```
TS1(SWITCH_GREEN:=TRUE,
SWITCH_YELLOW:=FALSE,
SWITCH_RED:=FALSE);
TS2(SWITCH_GREEN:=FALSE,
SWITCH_YELLOW:=FALSE,
SWITCH_RED:=TRUE);
```

В основній дії:

```
TS2;
```

Останній крок YELLOW2 вмикає жовтий сигнал світлофора В і вимикає зелений сигнал світлофора А.

Вхідна дія YELLOW2:

```
TS2 (SWITCH_GREEN: = FALSE,
SWITCH_YELLOW: = TRUE,
SWITCH_RED: = FALSE);
```

Основна дія YELLOW2:

```
TS2;
```

Вихідна дія YELLOW2

```
TS1 (SWITCH_GREEN: = FALSE);
```

Описана послідовність кроків виконується циклічно.

Тепер додамо в нашу програму можливість увімкнення/вимкнення світлофора. Жовтий сигнал вимкненого світлофора повинен мигати. Вимкнення світлофора можна проводити тільки після завершення повного циклу роботи світлофора. За увімкнення/вимкнення світлофора відповідатиме глобальна логічна змінна ON.

Додамо в нашу SFC-схему альтернативну гілку, яка виконується, коли світлофор вимкнений (ON = FALSE). На рис. 6.86 показана схема SFC з додатковою гілкою.

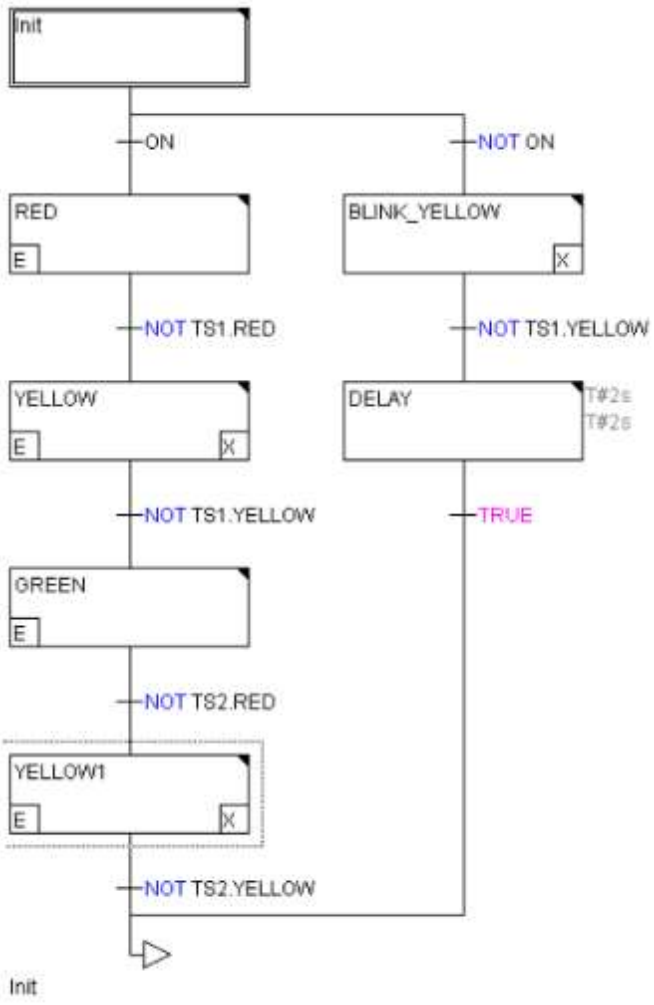


Рисунок 6.86 – PLC_PRG з альтернативною гілкою

В альтернативній гілці є два кроки. Крок BLINK_YELLOW вмикає жовтий сигнал обох світлофорів на дві секунди:

```
TS1(SWITCH_YELLOW:=TRUE);
TS2(SWITCH_YELLOW:=TRUE);
```

Після того як жовті сигнали будуть вимкнені, у вихідній дії скидаються вхідні сигнали

TS1(SWITCH_YELLOW:=FALSE);
TS2(SWITCH_YELLOW:=FALSE);

Це зроблено для того, щоб у наступному циклі забезпечити передній фронт на цих входах.

Крок DELAY порожній. Він потрібен тільки для затримки, яка триватиме протягом 2 с. Протягом цього часу жовтий сигнал світлофора не горить. Для задання цієї затримки потрібно у властивостях кроку визначити параметр Minimum time (рис. 6.87).

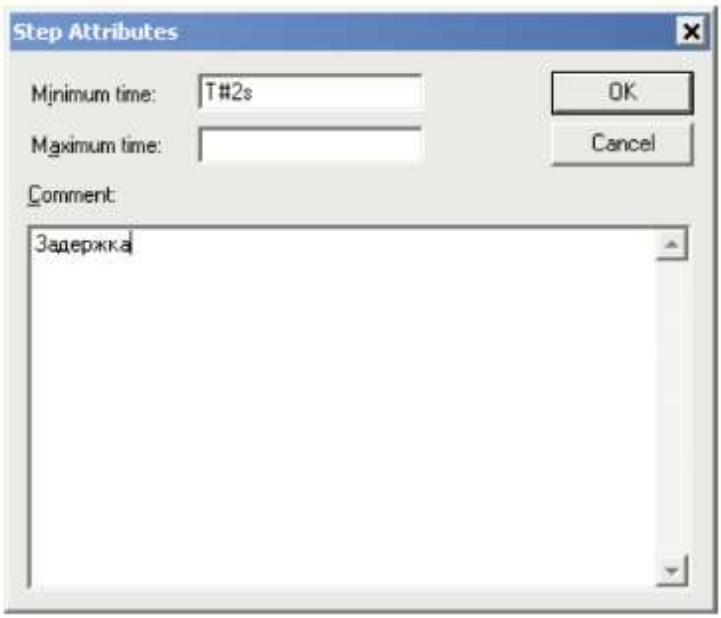


Рисунок 6.87 – Налаштування часу виконання кроку

Далі слід модифікувати програму так, щоб можна було ставити період роботи світлофора і час увімкнення зеленого сигналу світлофора. Вважатимемо, що період роботи світлофора можна задавати від 20 с до 6 хв. Достатньо лише задавати час увімкнення зеленого сигналу світлофора А, оскільки час увімкнення червоного можна порахувати через період.

Час зеленого сигналу В відповідатиме часу червоного сигналу А, а час червоного В – зеленому А. Це пов’язано з тим, що світлофори працюють у протифазі.

Період роботи світлофора зберігатиметься у змінній TS_period, а час дозволяючого сигналу світлофора А (у відсотках від періоду) у змінній TS1_greentime.

Установку часу увімкнення сигналів обох світлофорів проводитимемо у кроці INIT:

```
TS1.GREEN_DELAY:=(TS_period*TS1_greentime)/100;
TS1.RED_DELAY:=TS_period-T#4s-TS1.GREEN_DELAY;
TS2.GREEN_DELAY:=TS1.RED_DELAY;
TS2.RED_DELAY:=TS1.GREEN_DELAY;
```

Тепер, коли логіка роботи нашої системи повністю описана, необхідно пов’язати наш додаток із зовнішнім світом – зв’язати імена змінних з входами/виходами контролера і створити інтерфейс, за допомогою якого диспетчер керуватиме світлофором.

Почнемо з визначення входів/виходів. Зазвичай входи/виходи оголошуються за допомогою ПЛК конфігуратора, як показано на рис. 6.88.

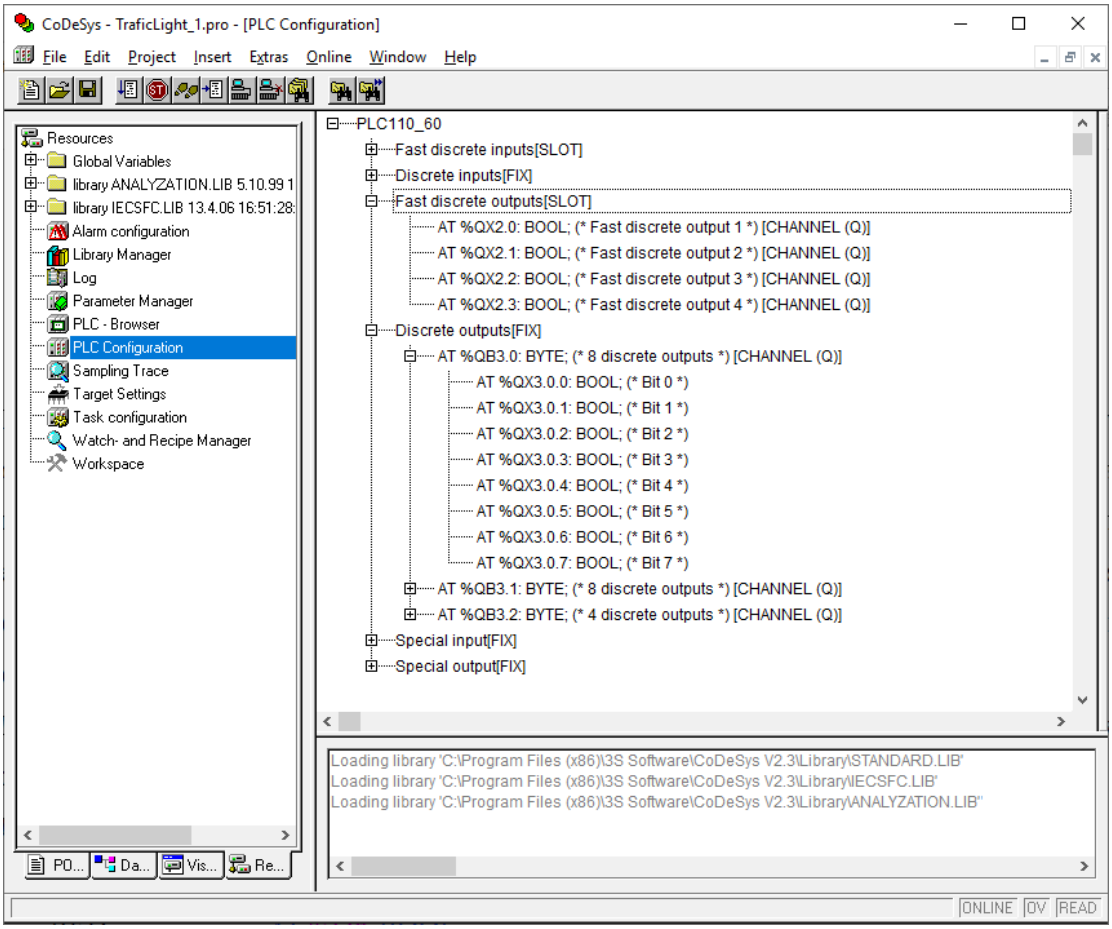


Рисунок 6.88 – Конфігуратор входів/виходів

За МЕК-адресою оголошується глобальна змінна, яка в проекті буде входом або виходом. Але ми використовуватимемо інший спосіб задання змінних. Справа в тому, що змінні, які у нас відповідають фізичним сигналам світлофора, оголошені всередині функціональних блоків. Тобто, щоб вивести їх значення на виходи контролера, значення цих змінних потрібно скопіювати у розділі оголошення глобальних змінних у ПЛК-конфігураторі. Це не дуже зручно. Тому ми скористаємося конфігураційними змінними.

Для цього у функціональному блоці TLLAFFLCSIGNAL потрібно модифікувати оголошення виходів функціонального блоку так:

```
VAR OUTPUT
    GREEN AT %Q*: BOOL;
    YELLOW AT %Q*: BOOL;
    RED AT %Q*: BOOL;
END_VAR
```

Цей запис свідчить про те, що дані виходи функціонального блоку будуть виходами контролера. Тепер визначимо, якими саме. Для цього у глобальних змінних проекту в списку variable configuration напишемо:

```
PLC_PRG.TS1.RED AT %QX0.0:BOOL;
PLC_PRG.TS1.YELLOW AT %QX0.1:BOOL;
PLC_PRG.TS1.GREEN AT %QX0.2:BOOL;
PLC_PRG.TS2.RED AT %QX0.3:BOOL;
PLC_PRG.TS2.YELLOW AT %QX0.4:BOOL;
PLC_PRG.TS2.GREEN AT %QX0.5:BOOL;
```

Цей запис свідчить про те, що PLC_PRG.TS1.RED відповідає нульовому виходу контролера, PLC PRG.TS1.YELLOW – першому і т.д. Тобто коли значення змінної PLC_PRG.TS1.RED дорівнює TRUE, нульовий вихід буде активний. Аналогічно і для інших виходів.

6.4 Контрольні запитання та завдання

1. Як налаштувати та підготувати до роботи інтегроване середовище розробки CODESYS v2.3?

2. Поясніть особливості налаштування і роботи інтегрованого середовища розробки CODESYS v3.x.

3. Як налаштувати зв’язок ПК і ПЛК?

4. Як створити проект в CODESYS 3?

5. Як відбувається робота з бібліотеками в CoDeSys 3?

6. Наведіть приклад роботи з дискретними виходами на прикладі вирішення задач управління світловою сигнальною колоною.

7. Наведіть структурну схему модуля керування блоком світлової та звукової сигналізації.

8. Наведіть схему розташування контактів на клемниках блоку керування.

9. Поясніть логіку роботи функціонального блоку TraficLight.

10. Поясніть принцип роботи програми керування групою світлових колон.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Асоціація підприємств промислової автоматизації України. URL: https://appau.org.ua/ (дата звернення: 04.04.2019).

2. Прозрачные производственные и логистические процессы. URL: https://www.sick.com/ru/ru/transparent-production-and-logistics-processes/w/smartfactory-transparent-production-process/ (дата звернення: 04.04.2019).

3. Билл Лайдон (Bill Lydon). Стандарт связи с датчиками IO-LINK набирает популярность. URL: http://ua.automation.com/content/standart-svjazi-s-datchikami-io%E2%80%90link-nabiraet-populjarnost/ (дата звернення: 14.04.2019).

4. Взаимодействие человека и робота на сетевом заводе. URL: https://www.sick.com/ru/ru/human-robot-collaboration-in-the-networked-factory/w/smartfactory-human-robot-collaboration/ (дата звернення: 15.04.2019).

5. Петров И.В. Программируемые контроллеры. Стандартные языки и приемы прикладного программирования / Под ред. проф. В.П. Дьяконова. – М.: СОЛОН-Пресс, 2004. – 256 с.

6. Минаев И.Г., Самойленко В.В. Программируемые логические контроллеры: практическое руководство для начинающего инженера. – Ставрополь: АРГУС, 2009. – 100 с.

7. Codesys products. URL: https://www.codesys.com/products.html/ (дата звернення: 30.09.2018).

8. ОВЕН. Обладнання для автоматизації. URL: https://owen.ua/ (дата звернення: 25.09.2018).

9. Черевко О.І., Кіптела Л.В., Михайлов В.М. та ін. Автоматизація виробничих процесів: підручник. – Харків: ХДУХТ, 2014. – 186 с.

10. Шишов О.В. Интеллектуальные датчики в системах промышленной автоматизации. Электроника и информационные технологии. – 2011. Вып. 2. URL: http//fetmag. mrsu.ru/ (дата звернення: 25.03.2019).

11. Макаров И.М. Интеллектуальные системы автоматического управления / Под ред. И.М. Макарова, В. М. Лохина. – М.: ФИЗМАТЛИТ, 2001. – 576 с.

Навчальне видання

НЕВЛЮДОВ Ігор Шакирович
НОВОСЕЛОВ Сергій Павлович
СИЧОВА Оксана Володимирівна

ТЕХНОЛОГІЯ ПРОГРАМУВАННЯ ПРОМИСЛОВИХ КОНТРОЛЕРІВ В ІНТЕГРОВАНОМУ СЕРЕДОВИЩІ CODESYS

Навчальний посібник

263

Відповідальний випусковий І.Ш. Невлюдов
Редактор О.Г. Троценко
Комп’ютерна верстка Л.Ю. Свєтайло

План 2019 (друге півріччя), поз. 17

Підп. до друку 26.06.2019	Формат 60×84 1/16.	Спосіб друку – ризографія.
Умов. друк. арк. 15,6	Облік. вид. арк. 13,9	Тираж 50 прим.
Ціна договірна.	Зам. № 1-17.	

ХНУРЕ. Україна. 61166, Харків, просп. Науки, 14

Віддруковано в редакційно-видавничому відділі ХНУРЕ
61166, Харків, просп. Науки, 14