МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

КОНТРОЛЕРНІ ЗАСОБИ АВТОМАТИЗАЦІЇ: ЧАСТИНА 1. ПРОГРАМОВАНІ ЛОГІЧНІ КОНТРОЛЕРИ

Комп'ютерний практикум

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів,
які навчаються за спеціальністю 151 «Автоматизація та комп'ютерно-інтегровані технології»,
освітньо-професійною програмою «Автоматизація та комп'ютерно-інтегровані технології кібер-енергетичних систем»

Київ
КПІ ім. Ігоря Сікорського
2020

Контролерні засоби автоматизації: Частина 1. Програмовані логічні контролери [Електронний ресурс] : навч. посіб. для студ. студ. спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», освітньо-професійної програми «Автоматизація та комп'ютерно-інтегровані технології кібер-енергетичних систем» / КПІ ім. Ігоря Сікорського ; уклад.: О. В., Степанець, Ю. І. Маріяш ;. – Електронні текстові дані (1 файл: 1,37 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 52 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 10 від 18.06.2020 р.) за поданням Вченої ради факультету (протокол № 10 від 25.05.2020 р.)

Електронне мережне навчальне видання

КОНТРОЛЕРНІ ЗАСОБИ АВТОМАТИЗАЦІЇ: ЧАСТИНА 1. ПРОГРАМОВАНІ ЛОГІЧНІ КОНТРОЛЕРИ
Комп'ютерний практикум

Укладачі: *Степанець Олександр Васильович, канд.техн.наук, доц.*
Маріяш Юрій Ігорович

Відповідальний редактор: *Волощук Володимир Анатолійович., д-р. техн. наук, доц.*

Рецензент: *Гагарін Олександр Олександрович, канд.техн.наук, доц., КПІ ім. Ігоря Сікорського*

Посібник призначений для студентів, які навчаються за спеціальністю «Автоматизація та комп'ютерно-інтегровані технології», освітньо-професійна програма «Автоматизація та комп'ютерно-інтегровані технології кібер-енергетичних систем», які вивчають курс «Контролерні засоби автоматизації – 1: програмовані логічні контролери».
Метою посібника «Контролерні засоби автоматизації – 1: програмовані логічні контролери» є висвітлення основних принципів функціонування промислових програмованих логічних контролерів, структури прикладного програмного забезпечення, отримання практичних навичок розробки програмного коду промислового призначення.

ЗМІСТ

Вступ 4
1 Лабораторна робота №1. 5
2 Лабораторна робота №2. 14
3 Лабораторна робота №3. 20
4 Лабораторна робота №4. 28
5 Лабораторна робота №5. 34
Навчально-методичні матеріали 43
Додаткок А. Перелік операторів IL 44
Додаток Б. Зміст звіту 52

ВСТУП

Сучасні системи автоматичного керування технологічними процесами побудовані, як правило, із застосуванням програмованих логічних контролерів. Вони дозволяють реалізовувати алгоритми керування будь-якої складності, тим самим значно піднімаючи планку якості керування об’єктами.

Якість реалізації алгоритмів регулювання, захистів, програмного керування значною мірою залежить від володіння розробником сучасного інструментарію програмування систем керування.

Основний зміст лабораторних робіт складають практичні задачі, з якими стикаються спеціалісти з промислової автоматизації. Вирішення задачі обернене в необхідність вивчення інженерних мов програмування стандарту МЕК-61131, у результаті чого студенти на практичних задачах вивчають особливості технологічного програмування.

У методичних вказівках наведені приклади алгоритмів керування та їх реалізації на текстових мовах IL та ST, графічних LD, FBD, SFC, розглянуті питання використання середовища програмування CoDeSys, критерії вибору оптимальних інструментів для вирішення виникаючих задач.

1 ЛАБОРАТОРНА РОБОТА №1. МОВА ПРОГРАМУВАННЯ IL

Мета роботи: ознайомитись з інженерною мовою програмування IL; отримати навички програмування на мові IL.

1.1 Теоретичні відомості

1.1.1 Загальні дані

Текстова мова IL (Instruction List) відноситься до ассемблероподібних мов, тобто до мов низького (машинного) рівня. Вона дозволяє створювати високоефективні і оптимізовані функції. ЇЇ можна рекомендувати для написання найбільш критичних місць в програмі.

Згідно стандарту IEC 61131-3 ключові слова мови повинні бути введені в символах верхнього регістру. Проміжки і мітки табуляції не впливають на синтаксис, вони можуть використовуватися скрізь.

Текст на IL - це текстовий список послідовних інструкцій. Кожна інструкція записується на окремому рядку.

Згідно стандарту IEC 61131-3 ключові слова мови повинні бути введені в символах верхнього регістру. Проміжки і мітки табуляції не впливають на синтаксис, вони можуть використовуватися скрізь.

Інструкція може включати 4 поля, розділені пробілами або знаками табуляції:

```
Мітка:     Оператор Операнд  Коментар
```

Мітка інструкції не є обов'язковою, вона ставиться тільки там, де потрібно. Оператор присутній обов'язково. Операнд необхідний майже завжди. Коментар — необов'язкове поле, записується в кінці рядка. Ставити коментарі між полями інструкції не можна.

Якщо використовується декілька операндів, вони відокремлюються комами.

Приклад IL-програми:

```
LABEL1: LD Sync (* приклад IL *)
AND Start
S Q
(* Для краси мітку можна поставити в окремий рядок *)
LABEL2:
LD 2 (* Y = 2 +2 *)
```

```
ADD 2
ST Y
```

1.1.2 Акумулятор

Мова IL - акумуляторно-орієнтована мова, тобто кожна команда використовує або змінює поточне вміст акумулятора (тип тимчасової пам'яті). У стандарті МЕК замість терміна «акумулятор» використовується термін результат (result). Так, інструкція бере «поточний результат» і формує «новий результат». Тим не менше майже всі керівництва з програмування різних фірм широко використовують термін «акумулятор». Акумулятор IL є універсальним контейнером, що здатен зберігати значення змінних будь-якого типу.

Абсолютна більшість інструкції IL виконують деяку операцію з вмістом акумулятора. Операнд, звичайно, теж бере участь в інструкції, але результат знову поміщається в акумулятор. Наприклад, інструкція SUB 10 віднімає число 10 від значення аккумулятора і поміщає результат в акумулятор. Команди порівняння порівнюють значення операнда і акумулятора, результат порівняння (TRUE або FALSE) знову поміщається в акумулятор.

Команди переходу на мітку здатні аналізувати акумулятор і приймати рішення - виконувати перехід чи ні.

Список команд повинен завжди починатися з оператора LD (команда завантаження акумулятора) і закінчуватися оператором збереження ST.

Приклад додавання:

```
LD 10
ADD 25
ST A
```

Приклад показує завантаження літерала 10 в акумулятор, додавання літерала 25 і внесення результату в змінну A. Вміст змінної та акумулятора тепер 35. Будь-яка наступна команда працювала б з вмістом акумулятора 35, якщо вона не починається з LD.

Операції порівняння також завжди стосуються акумулятора. Булевий результат порівняння вноситься в акумулятор, отже, це є поточним вмістом акумулятора.

Приклад порівняння:

```
LD B
GT 10
```

У прикладі значення змінної B завантажене в акумулятор і порівнюється з літералом 10. Якщо B менше або дорівнює 10, вміст акумулятора дорівнює 0 (FALSE). Якщо B більше ніж 10, вміст акумулятора є 1 (TRUE).

1.1.3 Операнди

Операндом може бути літерал, змінна, структурована змінна, елемент структурованої змінної, вихід функціонального блоку або пряма адреса.

1.1.1Оператори

Оператор є символом для арифметичної або логічної операції, яка буде виконана, або для виклику функції.

Оператори є узагальненими, тобто вони автоматично коригуються до типу даних операнда.

Оператори мови програмування IL наведені у додатку.

1.1.1Мітки та переходи

Програма на IL виконується підряд, зверху вниз. Для зміни порядку виконання і організації циклів застосовується перехід на мітку. Перехід на мітку може бути безумовним JMP - виконується завжди, незалежно від будь-чого. Умовний перехід JMPC виконується тільки при значенні акумулятора ІСТИНА. Перехід можна виконувати як вгору, так і вниз. Мітки є локальними, іншими словами, перехід на мітку в іншому POU не допускається.

Властивості міток:

• Мітки завжди повинні бути першими елементами в рядку.
• Ім'я має бути вільним у межах POU і не є чутливим до регістру.
• Мітки можуть бути завдовжки до 32 символів (максимум).
• Мітки повинні відповідати IEC угодам про іменування.
• Мітки відокремлюються двокрапкою : від наступної інструкції.
• Мітки дозволені тільки на початку "Виразів", інакше в акумуляторі може бути виявлено невизначене значення.

Приклад:

```
start: LD A
AND B
OR C
ST D
JMP start
```

Властивості стрибків:

• За допомогою JMP стрибок до мітки може бути обмеженим або необмеженим.
• JMP може використовуватися з модифікаторами C і CN (тільки якщо вміст акумуляторів типу даних BOOL).
• Стрибки можуть проводиться в межах програми.
• Стрибки можливі тільки в поточній секції.

Переходи потрібно організовувати досить акуратно, щоб не отримати нескінченний цикл:

```
LD 1
ST Counter
loop1:
(* Тіло циклу *)
LD Counter
ADD 1
ST Counter
LE 5
JMP loop1
```

У прикладі показана реалізація циклу на 5 повторень з однією очевидною помилкою. Замість безумовного переходу JMP повинен бути JMPC.

1.1.2 Модифікатори

Модифікатори впливають на виконання попереднього оператора.

Модифікатор N використовується, щоб побітово інвертувати значення операнда. Модифікатор N може застосовуватися тільки до операндів базових типів даних.

Приклад модифікатора N:

```
LD A
ANDN B
ST C
```

У прикладі C = 1, якщо A = 1 і B = 0.

Модифікатор C використовується, щоб виконати відповідну команду, якщо значення акумулятора дорівнює 1 (TRUE). Модифікатор C може застосовуватися тільки до операндів типу даних BOOL.

Приклад модифікатора C:

```
LD A
AND B
JMPC START
```

У прикладі перехід до START виконується, тільки якщо A = 1 (TRUE) і B = 1 (TRUE).

Якщо модифікатор C об'єднаний з модифікатором N, відповідна команда виконується, тільки якщо значення акумулятора дорівнює булевому 0 (FALSE).

Модифікатор ((ліва кругла дужка) використовується, щоб затримати оцінку операнда до появи оператора) (права кругла дужка). Круглі дужки можуть бути вкладеними.

1.1.3 Виклик функціональних блоків і функцій

Викликати екземпляр функціонального блоку або програму в IL можна з одночасним присвоюванням змінних. Наприклад:

```
CAL CTD_1 (CD: = TRUE, LOAD: = FALSE, PV: = 10)
LD CTD_1.CV
ST Y
```

Аналогічний виклик можна виконати з попередніми присвоюванням значень вхідних змінних:

```
LD TRUE
ST CTD_1.CD
LD FALSE
ST CTD_1.LOAD
LD 10
ST CTD_1.PV
CAL CTD_1
LD CTD_1.CV
ST y
```

При виклику функції з перерахуванням параметрів в IL існує одна важлива особливість. В якості першого параметра використовується акумулятор:

```
LD TRUE
SEL 3,4
```

На ST це рівносильно виклику SEL (TRUE, 3,4). Очевидно, при виклику функції або оператора з одним параметром список параметрів взагалі не потрібен:

```
LD ivar1
INT_TO_BOOL
ST bvar1
```

1.1.4 Коментування тексту

Коментування тексту — це додаткова пояснююча інформація про алгоритм програми (чи будь-які інші повідомлення програміста). Коментар починається символами «(*» і закінчується «*)».

1.2 Завдання

1.2.1 Загальні завдання

Розробити функціональні блоки, описані нижче. Розробити варіант візуалізації, що включає в себе засоби зміни значень входів (кнопки для логічних сигналів, поля введення – для аналогових) та тренд для відображення зміни виходів.

1) Interlock

Блок Interlock змінює значення своїх логічних виходів так, щоб «1» була виході, пов'язаному з останнім активованим логічним входом.

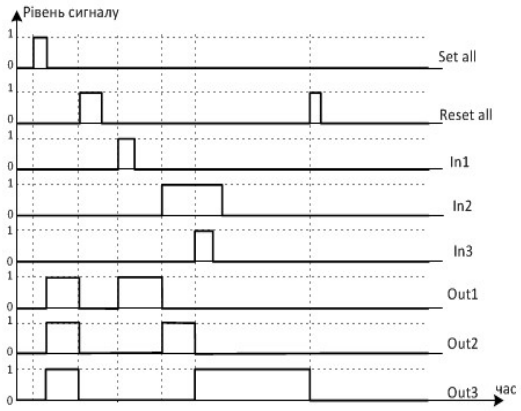


Рис. 2.1. Діаграма роботи блоку для випадку 3-х входів/виходів

Блок має певну кількість логічних входів та таку ж кількість виходів.

Логічний вхід пов'язаний з логічним виходом, що має той самий номер (вхід №1 – з виходом №1 і т.д.) Блок виставляє логічну одиницю на той вихід, на зв'язаний вхід якого в останній раз прийшла логічна «1». Інші виходи в цей час встановлюються рівними «0».

Блок має додаткові входи «Set all» та «Reset all». Вхід Set all надає кожному виходу значення «1», а вхід Reset all - виставляє «0» на кожному виході.
Блок зберігає значення виходів до того часу, поки на інший вхід не прийде «1». Зміна виходів у кожному випадку відбувається по передньому фронту імпульсу на входах блоку.

2) Ramp

Блок Ramp генерує сигнал на виході Out, який лінійно змінюється при наявності на одному з логічних входів Up чи Down значення «1».

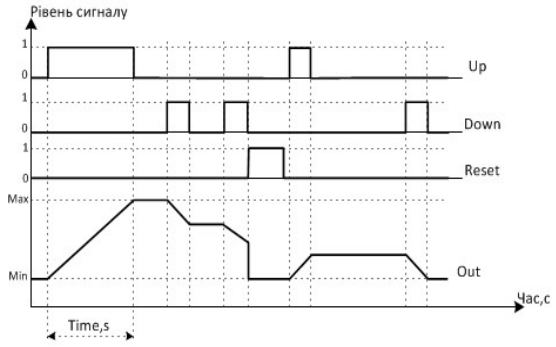


Рис.2.2. Діаграма роботи блоку Ramp

При подаванні логічної одиниці на вхід Up значення аналогового виходу Out його виходу зростає, а при наявності «1» на вході Down – спадає. Зміна Out припиняється при наявності на входах Up та Down логічного «0».

Швидкість зростання та спадання визначається по значенням на входах Min, Max, Time,s. Вхід Min обмежує мінімальне значення виходу Out, Max –

максимальне. Time,s визначає час збільшення Out від мінімального до максимального значення.

Логічний вхід Reset безумовно миттєво присвоює виходу Out значення, яке знаходиться на вході Min.

3) Toggle

Блок виставляє та зберігає значення своїх логічних виходів залежно від стану логічних входів.

Блок має виходи логічного типу Out та ^Out. Вихід ^Out завжди інвертований відносно виходу Out.

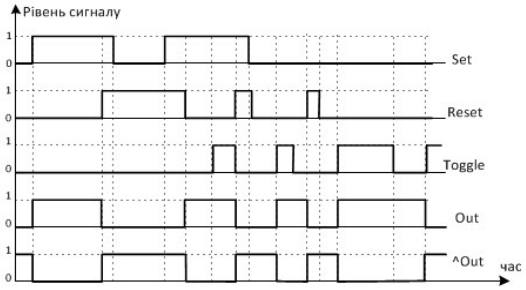


Рис. 2.3 Діаграма роботи блоку Toggle

Серед входів блоку є: Set, який безумовно виставляє на виході Out логічну «1»; Reset, що виставляє на Out «0», та Toggle, який змінює значення Out на протилежне.

Пріоритети входів у порядку спадання: Reset, Set, Toggle.

4) Buffer

Блок виставляє на своїх аналогових виходах значення, що приходять на відповідні входи (виходу №1 відповідає вхід №1 і т.д.), при наявності сигналу на логічному вході On. За відсутності сигналу на вході On на вихід передаються значення, отримані по задньому фронту сигналу On.

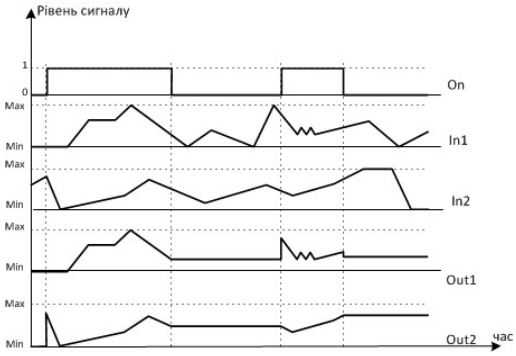


Рис.2.4 Діаграма роботи блоку Buffer

1.2.2 Додаткові (альтернативні) завдання

1. Реалізувати таймер, що фіксує час ввімкнення пристрою.
2. Реалізувати таймер, що вмикає пристрій (вихідну булеву змінну) через певний час після подання керуючого сигналу (вхідна булева змінна).
3. Розробити блок, що дозволяє вмикати певний пристрій (вихідна булева змінна) за допомогою одного з кількох зовнішніх сигналів (вхідних булевих змінних). Якщо пристрій ввімкнено, то надходження сигналу на будь-який з мулевих входів вимикає пристрій.
4. Написати програму, яка реалізує ввімкнення певних пристроїв за принципом гірлянди (почергове ввімкнення пристроїв із заданими часовими проміжками ввімкнення та вимкнення). Необхідні параметри роботи гірлянди повинні бути доступні для зміни користувачем.

5. Створити функціональний блок, який відловлює зовнішній аварійний сигнал, тобто фіксує аварійну ситуацію. Блок має булевий вхід, зв'язаний із зовнішнім аварійним сигналом (наприклад, від блоку-аналізатору стану аналогового давача). У разі надходження сигналу на цей вхід та витримки наперед заданого часу на булевий вихід блоку посилається сигнал про аварійну ситуацію. Після факту аварійної ситуації втрата вхідного сигналу про можливу аварію не повинна призводити до скидання вихідного мулевого сигналу. Аварійна ситуація може бути скинута лише по надходженню булевого сигналу на додатковий вхід Reset блоку. Час може змінювати користувач.

1.3 Порядок виконання роботи

1. Підібрати програмований логічний контролер та апаратне забезпечення, на яких можуть бути реалізоване поставлені завдання;
2. виконати загальні завдання;
3. виконати додаткове завдання (згідно номеру бригади);
4. оформити звіт про виконання роботи.

1.4 Методичні вказівки

Для оператора умовного вибору IF-ELSE можна використовувати конструкцію:

```
    LD      <Умова>
    JMPCN   No
        <Оператори TRUE>
    JMP         YesEnd
No:
        <Оператори FALSE>
YesEnd:
```

Для оператора циклу з передумовою можна використовувати конструкцію:

```
Loop_begin:
        LD <Умова продовження>
        JMPCN Loop_end
        <Оператори циклу>
        JMP Loop_begin
Loop_end:
```

Для оператора циклу з постумовою можна використовувати конструкцію:

```
Loop:
        <Оператори циклу>
        LD <Умова завершення>
        JMPCN Loop
```

1.5 Контрольні запитання

1. Властивості акумулятора.
2. Структура інструкції мови IL.
3. Мітки та переходи.
4. Організація циклів та вибору.
5. Модифікатори.
6. Виклик POU.

2 ЛАБОРАТОРНА РОБОТА №2. МОВА ПРОГРАМУВАННЯ ST

Мета роботи: ознайомитись з інженерною мовою програмування ST; отримати навички програмування на мові ST.

2.1 Теоретичні відомості

2.1.1 Загальні відомості

Мова ST (Structured Text) є мовою високого рівня (типу Паскаля). Призначена для універсального аналізу даних. Зручна для програм, що включають числовий аналіз або складні алгоритми. Може використовуватися в головних програмах, в тілі функції або функціонального блоку, а також для опису дій всередині елементів мови SFC. Проста у супроводі, якщо імена змінних зрозумілі, код добре структурований.

Згідно IEC 61131-3 ключові слова повинні бути введені в символах верхнього регістра. Пробіли і мітки табуляції не впливають на синтаксис, вони можуть використовуватися скрізь.

Основою ST-програми служать 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:

```
iVar1: = 1 + iVar2 / ABS (iVar2);
iVar3:=EXPT(iVar1, 3);
```

Стандартні оператори в виразах ST мають символьне представлення, наприклад математичні дії: +, -, *, /, операції порівняння і т. д.

Вираз може включати інший вираз, вкладений в дужки. Вираз, що знаходиться у дужках, обчислюється в першу чергу.

Тип виразу визначається типом результату обчислень.

Обчислення виразу відбувається відповідно з правилами пріоритета операцій. Першими виконуються операції з вищим пріоритетом.

У порядку зменшення пріоритету операції розташовуються так: вираз в дужках; виклик функції; степінь EXPT; заміна знака (-); заперечення NOT; множення, ділення й ділення по модулю MOD; додавання і віднімання (+, -); операції порівняння (<,>, <=,> =); рівність (=); нерівність (<>); логічні операції AND, XOR і OR.

Оператори, що виконують математичні та логічні операції, приклади їх використання, наведені у додатку.

2.1.2 Оператори вибору

Оператор вибору IF дозволяє виконати різні групи виразів в залежності від умов, виражених логічними виразами.

Якщо <логічний вираз IF> ІСТИНА, то виконуються вирази першої групи - <вирази IF>. Інші вирази пропускаються, альтернативні умови не перевіряються.

Синтаксис	Приклад

IF <логічний вираз IF>
THEN
<Вирази IF>;
[
ELSIF <логічний вираз ELSEIF 1>
THEN
<Вирази ELSEIF 1>;
...
ELSIF <логічний вираз ELSEIF n>
THEN
<Вирази ELSEIF n>;
ELSE
<Вирази ELSE>;
]
END_IF
	IF A > B THEN
	C:=1;
	ELSIF A < B
	THEN
	C:=2;
	ELSE
	C:=3;
	END_IF

Частина конструкції в квадратних дужках є необов'язковою і може бути відсутньою. Якщо <логічний вираз IF> FALSE, то одна за одною перевіряються умови ELSIF. Перша умова, яка справдилася, призведе до виконання відповідної групи виразів. Інші умови ELSIF аналізуватися не будуть. Груп ELSIF може бути кілька або не бути зовсім.

Якщо всі логічні вирази дали хибний результат, то виконуються вирази групи ELSE, якщо вона є. Якщо групи ELSE немає, то не виконується нічого.

Оператор множинного вибору CASE дозволяє виконати різні групи виразів в залежності від значення однієї цілочисельної змінної або виразу.

Синтаксис	Приклад

CASE <цілочисельний вираз> OF
<Значення 1>:
<Вирази 1>;
<Значення 2>, <значення 3>:
<Вирази 3>;
<Значення 4> .. <значення 5>:
<Вирази 4>;
...
[
ELSE
<Вирази ELSE>;
]
END_CASE
	CASE byLeft/2 OF
	0,42:
	bReset: = TRUE;
	Var1: = 0;
	16 .. 24:
	Var1: = 1;
	ELSE
	Var1: = 2;
	END_CASE

Якщо значення виразу збігається із заданою константою, то виконується відповідна група виразів. Інші умови не аналізуються (<значення 1>: <вирази 1> ;). Якщо кілька значень констант повинні відповідати одній групі виразів, їх можна перелічити через кому (<значення 2>, <значення 3>: <вирази 3> ;). Діапазон значень можна визначити через двокрапку (<значення 4> .. <значення 5>: <вирази 4> ;).

Група виразів ELSE не є обов'язковою. Вона виконується при невиконанні жодної з умов (<вирази ELSE> ;).

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

2.1.3 Цикли

Цикли WHILE і REPEAT забезпечують повторення групи виразів, поки вірний умовний логічний вираз. Якщо умовний вираз завжди істинний, то цикл стає нескінченним.

Умова в циклі WHILE перевіряється до початку циклу. Якщо логічний вираз спочатку має значення ХИБНІСТЬ, тіло циклу не буде виконано жодного разу.

Синтаксис	Приклад

WHILE <Умовний логічний вираз> DO
<Вирази - тіло циклу>
END_WHILE
	ci: = 64;
	WHILE ci > 1 DO
	Var1: = Var1 + 1;
	ci: = ci / 2;
	END_WHILE

Умова в циклі REPEAT перевіряється після виконання тіла циклу. Якщо логічний вираз спочатку має значення ХИБНІСТЬ, тіло циклу буде виконане один раз.

Синтаксис	Приклад
REPEAT <Вирази - тіло циклу> UNTIL <Умовний логічний вираз> END_REPEAT	REPEAT Sum:=Sum+Var1; i:=i+1; UNTIL i<5 END_REPEAT

Цикл FOR забезпечує задану кількість повторень групи виразів.

Синтаксис	Приклад
FOR <Цілий лічильник>: = <Поч. значення> TO <Кінцеве значення> [BY <Крок>] DO <Вирази - тіло циклу> END_FOR	Var1: = 0; FOR cw: = 1 TO 10 DO Var1: = Var1 + 1; 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 і 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 при додатному прирості цикл не буде виконаний жодного разу. При однакових початковому і кінцевому значеннях тіло циклу буде виконане один раз.

Частина конструкції BY в дужках необов'язкова, вона визначає крок збільшення лічильника. За замовчуванням лічильник збільшується на одиницю в кожній ітерації. В якості лічильника можна використовувати змінну будь-якого цілого типу.

2.1.4 Переривання ітерацій операторами EXIT і RETURN

Оператор EXIT, розміщений в тілі циклів WHILE, REPEAT і FOR, призводить до негайного закінчення циклу.

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 дуже непросто.

2.2 Завдання

2.2.1 Загальні завдання

Розробити функціональні блоки, що реалізують роботу елементарних ланок передавальних функцій:

1. аперіодична першого порядку з коефіцієнтом підсилення;
2. аперіодична другого порядку з коефіцієнтом підсилення;
3. інтегральна;
4. реально-диференцююча;
5. транспортне запізнення.

Необхідні параметри блоків (сталі часу, коефіцієнти передачі тощо) повинні бути доступні для зміни ззовні.

Створити візуалізацію, що демонструє поведінку виходів блоків та має засоби впливу на їх входи.

Оформити блоки у вигляді бібліотеки.

2.2.2 Додаткові (альтернативні) завдання

1. Реалізувати FIFO-чергу з наперед заданою кількістю елементів.
2. Розробити блок, який у реальному часі отримує дані на свій аналоговий вхід та видає максимальне значення за весь час роботи блоку.
3. Розробити блок, який у реальному часі отримує дані на свій аналоговий вхід та видає максимальне значення за певний час роботи блоку (наприклад, найбільше значення за останні 30 секунд становить 15,68).
4. Написати ПІД-регулятор.
5. Написати ПДД^2-регулятор з ШИМ-модуляцією для керування трипозиційним виконавчим механізмом типу МЕО.

2.3 Методичні вказівки

Частою задачею є виклик функціональних блоків та визначенням періоду цих викликів (період дискретизації). Такий функціонал блоку корисний, якщо функціональний блок прив’язаний до робочого циклу контролера. Наприклад, якщо в алгоритмі блоку розраховується похідна певної величини, адже у реалізації визначення похідних використовується крок по часу між сусідніми значеннями величини.

Сам шаблон має вигляд:

```
(*розділ об’явлення змінних*)
VAR_INPUT
…
```

```
END_VAR
VAR_OUTPUT
  …
END_VAR
VAR
  init: BOOL; (*зміна-індикатор завершеної ініціалізації блоку*)
  cur_time: TIME; (*час початку роботи блоку на поточному циклі контролера*)
  last_time: TIME; (*час початку роботи блоку на минулому циклі контролера*)
  dtime: TIME; (*часовий проміжок між викликами блоку у двох суміжних циклах*)
END_VAR
(*розділ тіла блоку*)
cur_time:=TIME();
IF NOT init THEN (* гілка ініціалізації*)
  init := TRUE;
  last_time:=cur_time; (*останньому моменту виклику блооку присвоюється значення часу поточного значення*)
ELSE
  dtime:=cur_time – last_time; (*проміжок часу між викликами*)
        (* основний алгоритм *)
  last_time:=cur_time; (*зафіксоване значення останнього виклику блоку*)
END_IF;
```

2.4 Порядок виконання роботи

5. Підібрати програмований логічний контролер та апаратне забезпечення, на яких можуть бути реалізоване поставлені завдання;
6. виконати загальні завдання;
7. виконати додаткове завдання (згідно номеру бригади);
8. оформити звіт про виконання роботи.

2.5 Контрольні запитання

1. Оператори та вирази мови ST.
2. Пріоритети операцій в мові ST.
3. Оператори вибору мови ST.
4. Оператори циклу мови ST.
5. Оператори переривання ітерацій.

3 ЛАБОРАТОРНА РОБОТА №3. МОВА ПРОГРАМУВАННЯ LD

Мета роботи: ознайомитись з інженерною мовою програмування LD; отримати навички програмування на мові LD.

3.1 Теоретичні відомості

3.1.1 Загальні відомості

Мова LD (Ladder Diagram) — графічна мова, що за базується на принципах релейно-контактних схем (елементами релейно-контактної логіки є контакти, обмотки реле, вертикальні і горизонтальні перемички та ін.) з можливістю використання великої кількості різних функціональних блоків. Символіка цієї мови була запозичена з проектування в області електротехніки.

Перевагами мови LD є: подання програми у вигляді електричного потоку (близький фахівцям з електротехніки), наявність простих правил, використання тільки булевих виразів. Вона має значне поширення серед користувачів, раціональна для ручної оптимізації специфічних критичних місць коду. Найкраще LD підходить для побудови логічних перемикачів, але досить легко можна створювати і складніші ланцюги, як в FBD. Крім того, LD достатньо зручна для у правління іншими компонентами POU.

Діаграма LD складається з ряду ланцюгів.

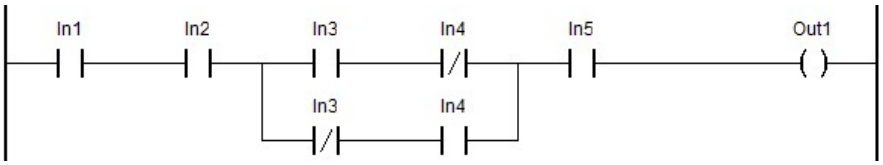


Рис. 3.1 Приклад LD-ланцюга

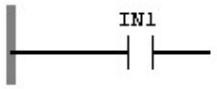
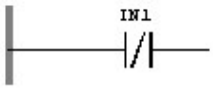
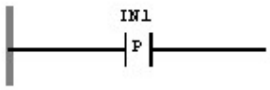
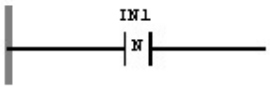
Ліворуч і праворуч схема обмежена вертикальними лініями — шинами живлення. Між ними розташовані ланцюги, утворені контактами і обмотками реле, за аналогією зі звичайними електронними ланцюгами.

Зліва будь-який ланцюг починається набором контактів, які посилають зліва направо стан "ON" або "OFF", що відповідають логічним значенням TRUE або FALSE. Кожному контакту відповідає логічна змінна. Якщо змінна має значення ІСТИНА, то стан передається через контакт. Інакше праве з’єднання отримує значення вимкнено ("OFF").

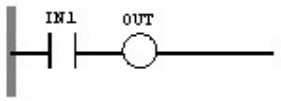
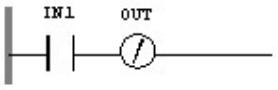
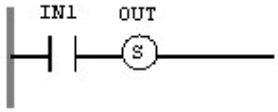
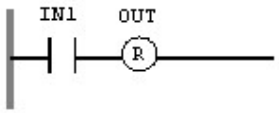
Кількість контактів в ланцюзі довільно, реле одне. Якщо послідовно з'єднані контакти замкнуті, струм йде по ланцюгу і реле включається. При необхідності можна включити паралельно кілька реле, послідовне включення не допускається.

3.1.2Елементи мови LD

Контакти. Контактом є LD-елемент, який передає стан горизонтального зв'язку лівого боку на праву сторону. Контакт не змінює значення пов'язаної з ним змінної або прямої адреси.

Тип контакту	Опис
	Для нормально розімкнутих контактів стан лівого зв'язку передається в правий зв'язок, якщо стан пов'язаного з контактом логічного фактичного параметра ON. Інакше, стан правого зв'язку OFF.
	Для нормально замкнених контактів стан лівого зв'язку передається в правий зв'язок, якщо стан пов'язаного з контактом логічного фактичного параметра OFF. Інакше, стан правого зв'язку ON.
	У контактах для визначення позитивних переходів правий зв'язок встановлюється в стан ON, якщо перехід пов'язаного з контактом фактичного параметра відбувається з OFF в ON, і в той же час стан лівого зв'язку ON. Інакше, стан правого зв'язку OFF. В CoDeSys замінений на ФБ R_TRIG.
	контактах для визначення задніх фронтів сигналу правий зв'язок встановлюється в стан ON, якщо перехід пов'язаного з контактом фактичного параметра відбувається з ON в OFF, і стан лівого зв'язку ON в той же час. Інакше, стан правого зв'язку OFF. В CoDeSys замінений на ФБ F_TRIG.

Котушки. Котушка є LD-елементом, який передає стан горизонтального зв'язку на лівій стороні горизонтальному зв'язку на правій стороні, не змінюючи його. У цьому процесі стан зв'язаної з котушкою змінної або прямої адреси буде збережено. Тобто у котушках стан лівого зв'язку передається в пов'язаний з котушкою логічний фактичний параметр і в правий зв'язок.

Тип котушки	Опис
	У котушці стан лівого зв'язку копіюється в пов'язаний логічний фактичний параметр стану зв'язку. Якщо зв'язок перебуває в стані ON, тоді правий зв'язок теж буде знаходитися в стані ON і пов'язаний логічний фактичний параметр буде знаходитися в стані ON.
	У котушці з інверсією стан лівого зв'язку копіюється в пов'язаний логічний фактичний параметр з інвертуванням стану зв'язку. Якщо зв'язок перебуває в стані OFF, тоді правий зв'язок теж буде знаходитися в стані OFF і пов'язаний логічний фактичний параметр буде знаходитися в стані ON.
	У котушці установки стан лівого зв'язку копіюється в праву зв'язок. Пов'язаний з котушкою логічний фактичний параметр встановлюється в стан ON, якщо лівий зв'язок має стан ON, інакше він не змінюється. Пов'язаний логічний фактичний параметр може скидатися тільки котушкою скидання.
	У котушці скидання стан лівого зв'язку копіюється в праву зв'язок. Пов'язаний з котушкою логічний фактичний параметр встановлюється в стан OFF, якщо лівий зв'язок має стан ON, інакше він не змінюється. Пов'язаний логічний фактичний параметр може встановлюватися тільки котушкою установки.

Ідеологія релейних схем має на увазі паралельну роботу всіх ланцюгів. Струм у всі ланцюги подається одночасно. У LD рішення діаграми виконується послідовно зліва направо і зверху вниз. В 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 ж значення. Якщо навіть реле в ланцюзі змінить змінну, то нове значення надійде на контакти тільки в наступному циклі. Ланцюги, що розташовані нижче, отримають нове значення змінної відразу. Ланцюги, які розташовані вище - тільки в наступному циклі. Завдяки жорсткому порядку виконання LD-діаграми зберігають стійкість при наявності зворотних зв'язків.

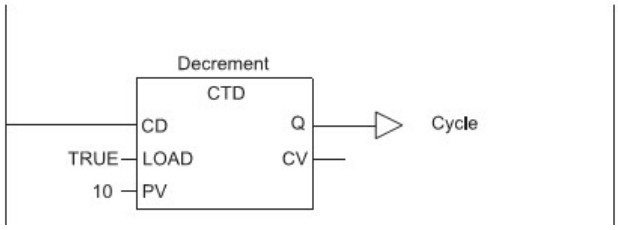
Час виконання одного LD-ланцюга не постійний. Якщо процесор виявив, що ланцюг розімкнутий, то вже немає сенсу аналізувати його далі, можна

відразу присвоїти значення FALSE вихідній змінній. Компілятор коду CoDeSys так і робить.

У LD-діаграму можна вставити функції та функціональні блоки. Функціональні блоки повинні мати логічні вхід і вихід.

Функція або функціональний блок у LD розглядаються як виконавчий пристрій - аналог реле.

Для функціональних блоків CoDeSys дозволяє графічно підключати тільки один вхід в логічний ланцюг, хоча стандарт не накладає таких обмежень.



Для включення в діаграму функцій в них штучно вводиться додатковий логічний вхід, позначуваний EN (Enable). Логічне значення на вхід EN дозволяє або забороняє виконання функції. Сама функція не терпить ніяких змін при додаванні входу EN.

3.2 Завдання

3.2.1 Загальні завдання

Розробити програму керування групою насосних агрегатів, що складається з двох насосів. Насосна група працює в режимі «основний – резервний», тобто один насос ввімкнений, інший вмикається при виході з ладу першого.

Інформаційні та керуючі сигнали в системі.

Start — дискретний сигнал, що запускає насоси;

Stop — дискретний сигнал, який зупиняє роботу насосів;

Pump1 — керуючий сигнал ввімкнення насосу №1;

Pump2 — керуючий сигнал ввімкнення насосу №2;

IsPressure1 — дискретний сигнал про наявність перепаду тиску до та після насосу №1 (індикація його роботи, у завданні імітується кнопкою);

IsPressure2 — дискретний сигнал про

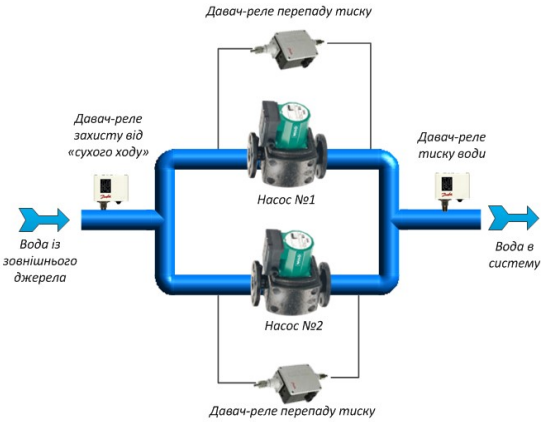


Рис.4.2 Схема насосної групи

наявність перепаду тиску до та після насосу №2 (індикація його роботи, у завданні імітується кнопкою);
Alarm1 — сигнал про аварію насоса №1;
Alarm2 — сигнал про аварію насоса №2;
DryFlow — сигнал про відсутність у системі води (небезпека «сухого ходу» насосів, у завданні імітується кнопкою).

Алгоритм.

1) Насоси працюють в автоматичному режимі. Дозвіл на роботу насосної групи надходить при замиканні контакта «Start» (кнопка без фіксації, розташована на пульті керування).

2) Насосна група зупиняється по натисненню на кнопку «Зупин» («Stop») на пульті керування (кнопка без фіксації).

3) При отриманні дозволу включається в роботу насос №1. Якщо через період часу «timeDelayAlarm» не замкнеться контакт «IsPressure1», пов’язаний з дискретним сигналом від диференційного давача-реле тиску до та після насосу, то насос №1 вважається таким, що вийшов з ладу. Насос №1 вимикається, запалюється світлова сигналізація «Аварія насоса №1» (лампа на пульті керування), вмикається насос №2.

4) Якщо за час «timeDelayAlarm» від моменту запуску насоса №2 не з’явився сигнал «IsPressure2», то насос №2 вважається таким, що вийшов з ладу. Насос №2 вимикається, на пульті керування вмикається лампа «Аварія насоса №2».

5) З метою забезпечення рівномірної роботи насосів потрібно передбачити зміну робочого насосу з основного на резервний з періодом часу «timeRotation». Наприклад, якщо насос №1 працював впродовж timeRotation, то по завершенню цього часу він вимкнеться і запуститься насос №2. І так далі.

6) Заборонити ввімкнення аварійного насосу. Так, якщо насос №1 зупинився у зв’язку з аварією, то, насос №2, відпрацювавши період ротації, не перейде в резерв. Аналогічно, якщо з ладу вийшов насос №2.

7) Насоси можуть працювати, тільки якщо в трубопроводі є вода. Наявність води у трубопроводі визначається по замкненому контакту давача-реле «сухого ходу». Реле тиску встановлене перед насосною групою, контакт реле — нормально замкнений. Якщо у системи э вода, то контакт «DryFlow» знаходиться у замкненому стані і дозволяє роботу насосів.

3.2.2 Додаткові (альтернативні) завдання

1. Розробити програму, що керує освітленням в приміщенні. Кожне джерело світла (люмінесцентна лампа, освітлювальна точка тощо) має власний вимикач. Всі джерела світла можуть бути вимкнені одним майстер-вимикачем. Візуалізувати роботу програми у вигляді плану приміщення з розташованими

на ньому джерелами світла та вимикачами. Джерела світла, що розташовані біля вікна, автоматично вимикаються при появі сигналу від давача зовнішнього освітлення (дискретний сигнал), але потім можуть бути примусово ввімкнені користувачем.

2. Написати програму, що керує температурою в холодильнику. Виконавчий механізм – компресор холодильника, закон керування — двопозиційний.

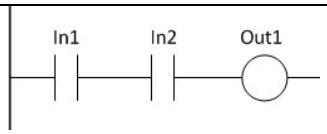
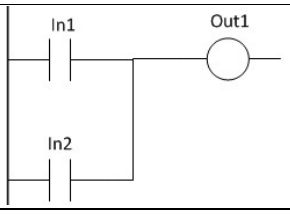
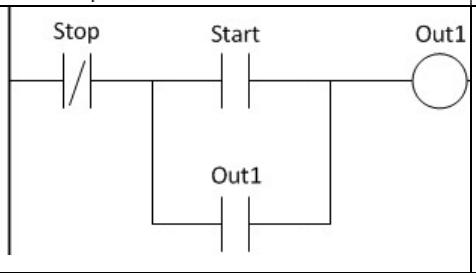
3. Розробити функціональний блок, що керує роботою асинхронного реверсивного трифазного виконавчого механізму в залежності від регулятора з імпульсними вихідними сигналами «більше-менше». Блок повинен блокувати ввімкнення механізму при появі сигналів від кінцевих вимикачів.

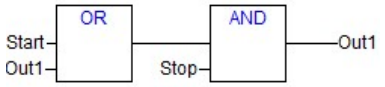
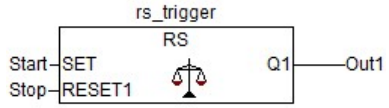
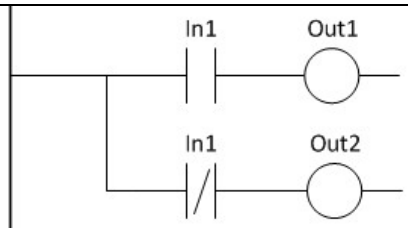
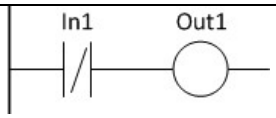
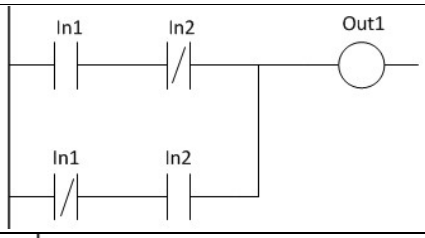
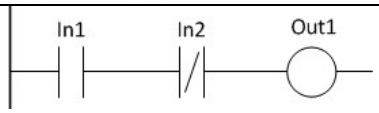
4. Розробити блок, що видає усереднене значення з кількох своїх входів. Блок повинен не враховувати значення зі входів, пов’язаних з дефектними давачами (визначається за виходом показів давачів за допустимі межі).

5. Розробити програму, яка реалізує кодовий замок із 3-х послідовних натиснень кнопок. Всього кнопок 10. Натиснення на невірну кнопку чи правильних кнопок, але в невірному порядку призводить до ввімкнення сирени. Натиснення у вірному порядку відчиняє сейф. Правильна комбінація прошивається у програму на етапі створення і не підлягає зміні користувачем.

3.3 Методичні вказівки

Приклади елементарних операцій та алгоритмів на мові LD:

Операція чи функціонал	Реалізація на LD	Опис
Логічне І		Реле Out1 замкнеться, коли обидва контакти In1 та In2 замкнені. На мові ST: Out1:=In1 AND In2;
Логічне АБО		Реле Out1 замкнеться, коли хоча б один з контактів In1 чи In2 замкнений. Out1:=In1 OR In2;
Реле із самопідхватом (триггер)		Реле Out1 замкнеться, якщо на контакті Stop немає сигналу та замкнено контакт Start. Реле буде замкнено до моменту, поки не подадуть сигнал на контакт Stop, який розірве ланцюг. На мові ST:

		Out1:=NOT Stop AND (Start OR Out1); На мові FBD:  або 
Перекидний контакт		Реалізує реле з перекидним контактом, тобто реле, що має два контакти Out1 і Out2, які завжди знаходяться у протилежному стані. Їх стан залежить від зовнішнього сигналу In1. На мові ST: Out1:=In1; Out2:= NOT In1;
Логічне НЕ		Реле Out1 замкнене, якщо на In1 не подається сигнал. На мові ST: Out1:=NOT In1;
Логічне виключаюче АБО (XOR)		Реалізація логічної операції XOR на мові LD. На мові ST матиме вигляд: Out1:=In1 XOR In2;
Лоічне І-НЕ		Реле Out1 замкнене, якщо на In1 подається сигнал, а на контакт In1 — ні. На мові ST виглядає так: Out1:=In1 AND (NOT In2)

3.4 Порядок виконання роботи

9. Підібрати програмований логічний контролер та апаратне забезпечення, на яких можуть бути реалізоване поставлені завдання;
10. виконати загальні завдання;
11. виконати додаткове завдання (згідно номеру бригади);
12. оформити звіт про виконання роботи.

3.5 Контрольні запитання

1. Елементи мови LD.
2. Типи контактів та реле.
3. Реалізація елементарних операцій за допомогою мови LD.
4. Використання функціональних блоків.
5. Час виконання програми на мові LD.

4 ЛАБОРАТОРНА РОБОТА №4. МОВА ПРОГРАМУВАННЯ FBD

Мета роботи: ознайомитись з інженерною мовою програмування FBD; отримати навички програмування на мові FBD.

4.1 Теоретичні відомості

4.1.1Загальні відомості

Мова FBD (Function Block Diagram) - графічна мова програмування високого рівня, що забезпечує керування потоками даних усіх типів. Дозволяє використовувати дуже потужні алгоритми простим викликом функцій і функціональних блоків (ФБ). FBD є більш ефективним для подання структурної інформації, ніж мова релейно-контактних схем.

Розробка програми для ПЛК здійснюється за допомогою графічного редактора FBD шляхом формування блок-схеми. У ній блоки об'єднуються один з одним або за допомогою зовнішніх (фактичних) параметрів ФБ (змінні, приєднані до відповідних входів і виходів ФБ), або безпосередньо лініями зв'язку - графічними зв'язками. 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 чи скасована безпосередньо в діаграмі. У CoDeSys це робиться командою «Negate». На рис. 5.1 змінна A подається на вхід блоку OR з інвертованим значенням.

Послідовність (черговість) обробки окремих ФБ в програмі (в кожній FBD-секції) визначається потоком даних усередині секції. Діаграма FBD будується з компонентів, що відображаються на схемі прямокутниками. Входи POU зображуються зліва від прямокутника, виходи праворуч. Всередині прямокутника вказується тип POU і найменування входів і виходів. Ім'я примірника (екземпляра) показано над рамкою. Ім'я екземпляра є унікальним ідентифікатором для функціонального блоку в проекті. У графічних системах програмування прямокутник компонента може містити зображення, що відображає його тип. Розмір прямокутника залежить від числа входів і виходів і встановлюється графічним редактором автоматично.

CoDeSys дозволяє записувати вирази ST на вході графічних блоків. Такий прийом розширює стандартний FBD і часто виявляється досить зручним. Компактна форма представлення слів полегшує запис і читання функциональних діаграм.

На рисунку 5.1 наведено загальний вигляд секції FBD.

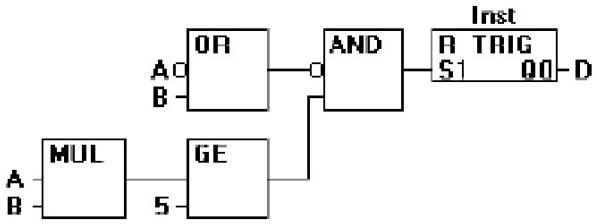


Рис. 5.1. Представлення секції FBD

Програма в FBD не обов’язково повинна становити велику єдину схему. Як і в LD, діаграма утворюється з множини ланцюгів, які виконуються одна за одною. У CoDeSys всі ланцюги одного POU відображаються в єдиному графічному вікні (рис. 5.2), пронумеровані і розділені горизонтальними лініями. Значення змінних, обчислені в одного ланцюга, доступні в наступних ланцюгах відразу в тому ж робочому циклі.

На відміну від FBD редактор безперервних функціональних схем (CFC) не використовує ланцюга, але дає можливість вільно розміщувати компоненти та сполуки, що дозволяє створювати зворотні зв'язки, як показано в прикладі (рис. 5.3).

Прямокутники POU в FBD з’єднані лініями зв’язку. З’єднання мають спрямованість зліва направо. Вхід блоку може бути з’єднаний з виходом блока, розташованого ліворуч від нього. Крім цього, вхід може бути з’єднаний із змінною або константою. З’єднання має пов’язувати змінні або входи і виходи одного типу. На відміну від компонента, змінна зображується на діаграмі без прямокутної рамки. Ширина сполучної лінії в FBD ролі не грає.

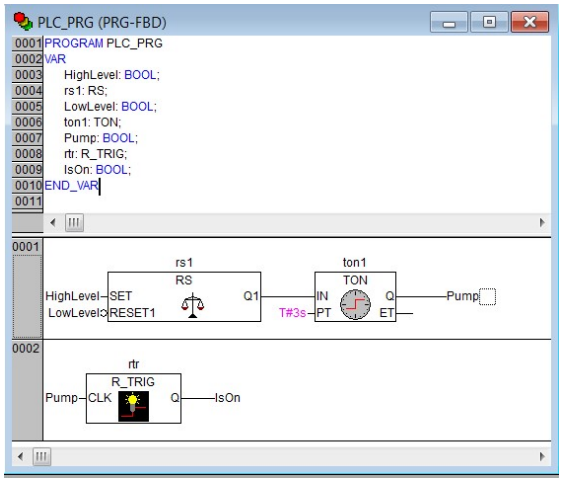


Рис. 5.2. Діаграма FBD з двох ланцюгів

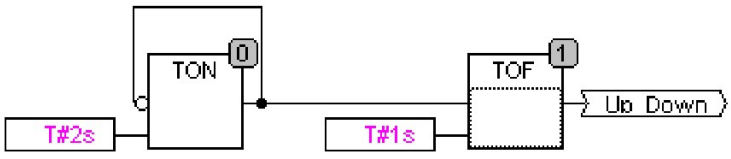


Рис. 5.3. Ланцюг CFC-діаграми

4.1.2 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. Подальші обчислення не будуть продовжені до обчислення значень на всіх виходах. Іншими словами, значення на всіх виходах графічного блоку з'являються одночасно. Обчислення ланцюга вважається закінченим тільки після обчислення значень на виходах всіх вхідних в неї елементів.

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 CoDeSys автоматично розставляє блоки в порядку виконання.

4.1.3 Мітки, переходи і повернення

Порядок виконання FBD-ланцюгів діаграми можна примусово змінювати, використовуючи мітки і переходи, точно так само, як і в релейних схемах. Мітка ставиться на початку будь ланцюга, будучи, по суті, назвою даного ланцюга. 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 заміняти мітки.

Перехід обов'язково пов'язаний з логічною змінною і виконується, якщо змінна має значення ІСТИНА. Для створення безумовного переходу використовується константа ІСТИНА, пов'язана з переходом.

Оператор повернення RETURN можна використовувати в FBD так само, як і перехід на мітку, тобто у зв'язці з логічної змінної. Повернення призводить до негайного закінчення роботи програмного компонента і поверненню на верхній рівень вкладень. Для основної програми це початок робочого циклу ПЛК.

4.2 Завдання

4.2.1 Загальні завдання

Деаератор як теплоенергетичне устаткування має дві технологічні величини, що повинні підтримуватися на заданому рівні. Це рівень води у баці та тиск пари у горішній частині баку над водою. Ця задача вимагає створення САР.

Реалізувати контури керування технологічних змінних деаератора: тиску пари та рівня води в баці.

Виконавчий механізм контуру керування тиском має плавне керування в діапазоні 0-100%, модель об'єкта керування: $W_{\text{PRES}}(s) = \frac{K}{Ts+1}$.

Регулятор контуру регулювання рівня води повинен мати релейно-імпульсні виходи «більше»-«менше». Він керує виконавчим механізмом типу МЕО. Постійна часу виконавчого механізму – 30 с. Об'єкт керування: $W_{\text{LEVEL}}(s) = \frac{K}{s}$.

Для імітації об'єкта керування використати функціональні блоки з лабораторної роботи №2. Візуалізувати роботу контурів керування, відобразити на тренді технологічні змінні та завдання (уставки). Передбачити можливість змінювати уставки.

Для реалізації регуляторів дозволяється використання стандартної бібліотеки ControlTecnology.

4.2.2 Додаткові (альтернативні завдання)

1. Написати програму, що реалізує каскадну систему керування з двома ПІ-регуляторами та виконавчим механізмом з аналоговим керуванням у межах 0-100%. Об'єкт керування — аперіодичні ланки першого порядку з коефіцієнтом передачі
2. Написати програму, що реалізує каскадну систему керування з двома ПІ-регуляторами та трипозиційним виконавчим механізмом постійної швидкості. Об'єкт керування — аперіодичні ланки першого порядку з коефіцієнтом передачі.
3. Написати програму, що реалізує систему автоматичного керування з Під-регулятором та компенсацією збурень. Об'єкт керування — аперіодична ланка першого порядку з коефіцієнтом передачі.
4. Написати програму, яка реалізує двоконтурну систему керування з регулятором та диференціатором. Об'єкт керування — аперіодичні ланки першого порядку з коефіцієнтом передачі.
5. Написати програму, що реалізує систему керування з ПІД-регулятором. Об'єкт керування — аперіодична ланка першого порядку з коефіцієнтом передачі. Завдання регулятору змінюється в залежності від зовнішнього сигналу (представляє собою кусково-задану функцію з трьох пар значень «зовнішній сигнал — завдання регулятору»).

4.3 Методичні вказівки

Для реалізації замкненого контуру автоматичного регулювання необхідно розробити функціональні блоки, які відповідають об'єктам керування кожного контуру.

Для об'єкта контуру керування тиском, який у даному спрощеному випадку є інерційною ланкою першого порядку, можна скористатися різницевою формулою:

$$y_i = \frac{T}{T + \Delta t} y_{i-1} + K \frac{\Delta t}{T + \Delta t} x_i ,$$

де

y_i - значення виходу на і-му кроці (зараз),

x_i - значення входу на і-му кроці (зараз),

y_{i-1} - значення виходу на і-1 -му кроці (попереднє значення),

T - стала часу об'єкта, с,

K - коефіцієнт передачі об'єкта, %ВМ/кПа,

Δt - період дискретизації, с.

Рівень води в баку деаератора можна спрощено представити як інерційну ланку. Її реалізація у вигляді функціонального блоку можлива за формулою:

$$y_i = y_{i-1} + K \Delta t x_i ,$$

де

y_i - значення виходу на і-му кроці (зараз),

x_i - значення входу на і-му кроці (зараз),

y_{i-1} - значення виходу на і-1 -му кроці (попереднє значення),

K - стала інтегрування, мм/с,

Δt - період дискретизації, с.

Виведення цих формул можна подивитися в матеріалах курсу ТАУ.

4.4 Порядок виконання роботи

13. Підібрати програмований логічний контролер та апаратне забезпечення, на яких можуть бути реалізоване поставлені завдання;
14. виконати загальні завдання;
15. виконати додаткове завдання (згідно номеру бригади);
16. оформити звіт про виконання роботи.

4.5 Контрольні запитання

1. Функціональний блок мови FBD.
2. Використання операцій ST на входах функціональних блоків.
3. Відмінності і схожість FBD та CFC.
4. Порядок виконання функціональних блоків.
5. Мітки, переходи та повернення.

5 ЛАБОРАТОРНА РОБОТА №5. МОВА ПРОГРАМУВАННЯ SFC

Мета роботи: ознайомитись з інженерною мовою програмування SFC; отримати навички програмування на мові SFC.

5.1 Теоретичні відомості

5.1.1Загальний опис

SFC - це графічна мова, що дозволяє описати хронологічну послідовність різних дій в програмі. Для цього дії зв'язуються з кроками (етапами), а послідовність роботи визначається у слові переходів між кроками.

Застосування SFC в об’ємних компонентах дозволяє скоротити час виконання і відповідно час реакції системи. За допомогою кроків монолітна програма розбивається на короткі фрагменти, що виконуються в різних робочих циклах ПЛК.

Будь SFC-схема складається з елементів, що представляють кроки і умови переходів (рис. 6.1). Кроки показані на схемі прямокутниками. Реальна робота кроку (дії) описується в окремому вікні системи програмування і не відбивається на діаграмі. Про призначення кроку SFC говорить тільки його назва або, якщо цього не достатньо, короткий текстовий опис (коментар).

SFC POU складається з набору кроків, пов'язаних переходами. Існують 2 види кроків:

- Крок простого типу (спрощений SFC) може включати єдину дію. Графічний прапорець (невеликий трикутник у верхньому кутку кроку) показує, пустий крок чи ні.
- МЕК крок (стандартний SFC) пов’язаний з довільним числом дій або логічних змінних.

Пов'язані дії розташовуються з правого боку від кроку.

Кожна SFC-схема починається з кроку, виділеного графічно подвійними вертикальними лініями або по всьому периметру. Це - початковий крок. Найменування початкового кроку може бути довільним (за замовчуванням Init). Початковий крок присутній обов'язково, хоча і може бути порожнім.

Дія може містити список інструкцій на IL або ST, схеми на FBD або LD, або знову схеми на SFC.

При використанні простих кроків дія завжди пов’язується з цим кроком. Для того, щоб редагувати дію, необхідно двічі клацнути лівою клавішею мишки на кроці. Або виділити крок і вибрати команду меню «Extras» «ZoomAction / Transition». Крім основної дії, крок може включати одну вхідну і одну вихідну дія.

Дії МЕК кроків показані в Організатора Об’єктів, безпосередньо під POU, що їх викликає. Редагування дії запускається подвійним клацанням миші або клавішею <Enter>. Нові дії додаються командою головного меню «Project» «Add Action». Одному кроку можна співставити до до 9 дій.

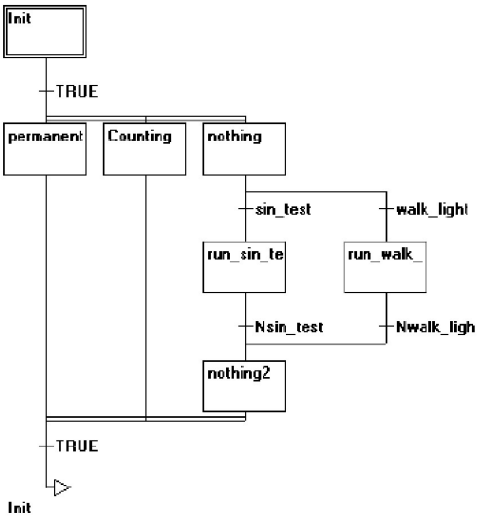


Рис. 6.1 Приклад структури SFC-програми.

Між кроками знаходяться так звані переходи. Умовою переходу може бути логічна змінна або константа, логічна адреса або логічний вираз, описаний на будь-якій мові. Умова може включати серію інструкцій, що утворюють логічний результат, у вигляді ST виразу (тобто (i<=100) AND b) або на будь-якій іншій мові. Але умова не повинна містити присвоювання, виклик програм і примірників функціональних блоків.

У редакторі SFC умову переходу можна записати безпосередньо поряд з символом переходу (допустима тільки мова ST) або в окремому вікні редактора для введення умови (тут можливе використання будь-якої мови)

Перехід виконується при дотриманні двох умов:

1) перехід дозволений (відповідний йому крок активний);

2) умова переходу має значення TRUE.

В якості умови переходу може бути задана логічна константа. Якщо задано TRUE, то крок буде виконаний одноразово, за один робочий цикл, далі управління перейде до наступного кроку. Якщо задана умова FALSE, то крок буде виконуватися нескінченно.

5.1.2 Паралельні та альтернативні гілки

Кілька гілок SFC можуть бути паралельними (рис. 6.2).

Ознакою паралельних гілок на схемі є подвійна горизонтальна лінія. Кожна паралельна гілка починається і закінчується кроком. Тобто умова входу

в паралельність завжди одна, умова виходу теж одна на всіх. Паралельні гілки виконуються теоретично одночасно, тобто в одному робочому циклі, зліва направо.

Умова переходу, яка завершує паралельність, перевіряється тільки в разі, якщо в кожній паралельній гілці активні останні кроки. У даному прикладі (рис.) Shaking буде виконаний одноразово, далі Digestion і Mixing будуть працювати паралельно до виконання умови Ready.

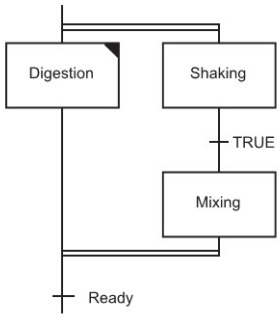


Рис. 6.2. Паралельні гілки

Кілька гілок SFC можуть бути альтернативними гілками (рис. 6.3). Ознакою альтернативних гілок на схемі є одинарна горизонтальна лінія. Кожна альтернативна гілка починається і закінчується власною умовою переходу. Перевірка альтернативних умов виконується зліва направо. Якщо вірна умова знайдена, то інші альтернативи не розглядаються. В альтернативних гілках завжди працює тільки одна з гілок, тому її закінчення і буде означати перехід до слідуючого за альтернативною групою кроку.

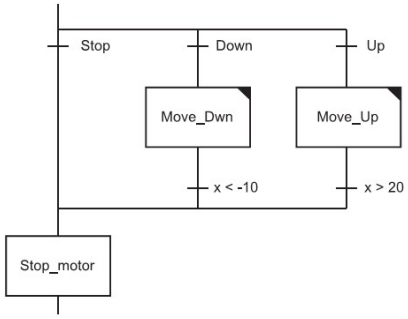


Рис. 6.3 Альтернативні гілки

У даному прикладі (рис. 6.3) альтернатива Stop оцінюється першою. Кроки Move_Dwn і Move_Up мають шанс стати активними, тільки якщо Stop дорівнює FALSE.

При створенні альтернативних гілок бажано ставити взаємовиключні умови. У цьому випадку ймовірність допустити помилку при аналізі або в процесі доопрацювання діаграми значно нижче.

5.1.3 Перехід на довільний крок

У загальному випадку SFC-схема виконується зверху вниз. Стандартом допускається створення переходів на довільний крок. Для цього застосовуються сполучні лінії з проміжними стрілками або пойменовані переходи. Тобто перехід виконується на крок, ім’я якого зазначено під стрілкою. В англомовних джерелах перехід на довільний крок називається «стрибок» (jump).

У прикладі, показаному на рис. 6.4, кроки Move_Dwn і Move_Up послідовно активують один одного. Зауважте, що умова Stop при цьому перевірятися не буде, кроки Move_Dwn і Move_Up з'єднані в логічне кільце, яке має 2 варіанти входу, але жодної можливості виходу. Маркер активності буде переміщатися виключно в цьому кільці.

Стрибок з однієї гілки паралельного блоку назовні викликає ефект розмножения маркера. Стрибок всередину паралельного блоку порушує паралельність гілок. Подібних трюків необхідно уникати.

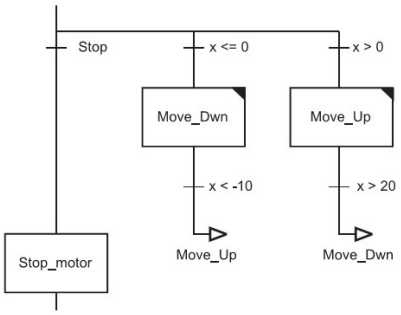


Рис. 6.4 Стрибок

5.1.4 Спрощений SFC

У CoDeSys, окрім стандартної МЕК – реалізації SFC, реалізована спрощена версія (easy mode SFC). Сенс її полягає в застосуванні більш простого, компактного і швидкого послідовного SFC-виконавця. Крім цього, самі діаграми виходять компактніше і часто простіше для розуміння. Безумовно, можливості спрощеної реалізації кілька вже - не можна вмикати і вимикати дії в різних кроках і управляти активністю дій по часу.

Дії можуть бути трьох класів - поточна, вхідна і вихідна. Графічно дії на діаграмі ніяк не відображаються, їх редагування виконується в окремих вікнах.

У спрощеній реалізації дії належать кроку. Тобто дія не можна викликатися з іншого кроку або звідкись ще.

Можна вважати, що кожен прямокутник кроку при його збільшеному розгляді містить 3 розділи, відповідні трьом можливим діям. Якщо крок видалити, то і всі його дії будуть втрачені. Такі дії не вимагають окремих ідентифікаторів і називаються за іменами кроків.

Для створення нового або редагування існуючого дії в CoDeSys досить клацнути мишкою по прямокутнику кроку. Це призведе до відкриття відповідного редактора або викликом діалогу створення нового дії, якщо крок ще не описаний.

Кроки, які містять дію, на схемі відрізняються тим, що верхній правий кут прямокутника зафарбований. Поки крок активний, поточна дія виконуватиметься один раз в кожному робочому циклі.

Досить імовірний випадок, коли певні дії потрібно виконати в кроці тільки один раз. З цією метою і передбачені вхідна і вихідна дії. Вхідна дію позначається сегментом 'E' (Entry) у нижньому лівому куті прямокутника кроку і виконується одноразово при активізації кроку. Вихідна позначається сегментом 'X' (eXit) в нижньому лівому куті прямокутника кроку.
Вихідна дія виконується одноразово при завершенні роботи кроку.

Для кожного кроку CoDeSys створює дві логічні змінні. Припустимо, крок називається Step1. Для нього будуть визначені змінні _Step1 і Step1. Оголошення змінних відбуваються неявно, тобто в розділі оголошень ніяких додаткових записів робити непотрібно. Змінна з лідируючим підкресленням (_Step1) отримує значення TRUE, коли крок активується (вхідна умова виконана), і скидається при деактивації (відразу при виконанні вихідної умови). Змінна без підкреслення (Step1) відстає на один робочий цикл, тобто отримує значення TRUE після виконання вхідної дії і скидається після виконання вихідної. Комбінації двох цих змінних (_Step1, Step1) послідовно утворюють 4 можливих стани кроку: не виконується (00), вхідна дія (10), поточна дію (11 і 10), вихідна дія (01).

Дані змінні можна використовувати для визначення активності кроку, наприклад, з метою синхронізації паралельних гілок (рис. 6.5).

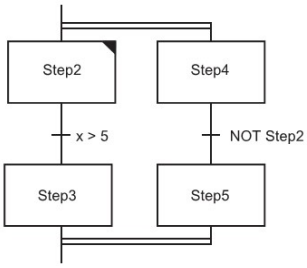


Рис. 6.5 Синхронізація кроків

Для доступу до змінних кроку поза даного компонента і з відладчика їх необхідно оголосити як логічні змінні.

Описаний спрощений механізм SFC продовжує працювати і при використанні МЕК кроків. Тобто CoDeSys дозволяє в будь-який час підключити повний SFC-виконавець без переробки того, що вже реалізоване.

5.1.5 Стандартний SFC

При застосуванні МЕК-дій спочатку визначаються дії (види робіт), які повинна виконувати система, а потім уже складається діаграма, в якій визначається їх порядок і взаємозв'язок. Кожна дія співставляється одному або декільком крокам. Причому цілком можливо, що деяка дія повинна запускатися в одному кроці і зупинятися в іншому. Також можливо, що розпочата дія повинна закінчити свою роботу взагалі незалежно ні від яких кроків. Наприклад, почавши рух, кабіна ліфта повинна як мінімум доїхати до найближчого поверху і випустити пасажирів, навіть якщо дана команда на закінчення роботи.

Дії МЕК показуються на SFC-діаграмі у вигляді прямокутників, розташованих праворуч від кроку і прив'язаних до нього графічно (рис. 6.6).

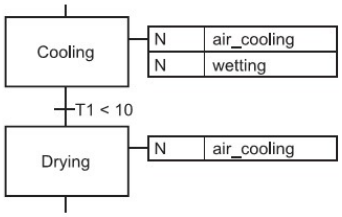


Рис. 6.6. Дії в стандртному SFC

Суттєво важливим тут є те, що одну і ту ж дію можна багаторазово використовувати в різних кроках. Так, в даному прикладі кроки Cooling (охолодження) і Drying (сушка) використовують дію air_cooling (повітряний обдув). На відміну від описаних вище спрощених дій, дії МЕК не належать конкретному кроку, а є самостійними програмними елементами SFC-компонента.

Ідентифікатори дій повинні бути унікальні в межах компонента POU і не повинні збігатися з ідентифікаторами кроків і переходів.

Прямокутник, що відображає дію, містить у лівій частині спеціальне поле-класифікатор (qualifier), який визначає спосіб впливу активного кроку на дану дію.

Можливі наступні класифікатори:

Класифікатор	**Призначення**	**Опис**
N	Без зберігання	Дія активна під час активності кроку
R	Позачергове скидання	Деактивація дії
S	Установка	Дія активна аж до скидання
L	Обмежена по часу	Дія активна впродовж вказаного проміжку, але не довше часу активності кроку

Класифікатор	Призначення	Опис
D	Відкладена	Дія активується з вказаною затримкою, якщо крок ще активний та продовжує бути активним
P	Імпульс	Дія виконується одноразово, якщо крок активний
SD	Зі збереженням та відкладенням	Дія активна після вказаного часу до скидання
DS	З відкладенням та збереженням	Дія активна після вказаного часу, якщо крок ще активний, аж до скидання
SL	Збереження та обмеження по часу	Активна після вказаного часу

Класифікатори L, D, SD, DS та SL вимагають указання константи часу в форматі TIME.

Кожна активна дія виконується ще один раз вже після деактивації. Це необхідно для того, щоб дії могли відпрацювати втрату активності і виконати деякі завершальні операції.

Визначити стан деактивації можна всередині дії шляхом аналізу ознаки активності. Така необхідність виникає особливо часто при реалізації імпульсних дій, коли різні операції потрібно виконати на початку і в кінці кроку.

5.2 Завдання

5.2.1 Основні завдання

Деаератор для коректного пуску вимагає строгого виконання послідовності дій, що описуються в паспорті обладнання.

Створити програму на мові SFC, яка реалізує підказки оператору про порядок дій. Підказки виводяться у вікно візуалізації. Оператор повинен підтверджувати виконання дії, натискаючи кнопку підтвердження. Якщо виконання дії має заздалегідь відому тривалість, то під час її активності повинен додатково відображатися час дії.

Підказку зобразити у вигляді строкової змінної.

Перелік операцій наступний:

- відкрити засувку на підводі пари в деаератор;

- прогріти деаератор протягом 20-30 хвилин. Тиск в деаераторі при цьому не повинен перевищувати робочий. При підігріві періодично продувати покажчики рівня;
- злити конденсат з бака через дренажну лінію;
- подати в деаератор хімочищену воду, встановити мінімальну її витрату (при наявності підігрівачів хімочищеної води включити їх в роботу), збільшивши одночасно витрату пари в деаератор за допомогою регулюючого клапана тиску;
- включити в роботу систему автоматичного регулювання тиску в деаераторі;
- Подати в деаераційну колонку основний конденсат (некиплячий);
- включити в роботу охолоджувач випару;
- встановити нормальний рівень води в деаераторному баці та увімкнути систему автоматичного регулювання рівня;
- відкрити засувку на лінії відводу деаерованої води з бака до живильних насосів;
- встановити номінальну витрату випара.

5.2.2 Додаткові (альтернативні) завдання

1. Реалізувати світлофор.
2. Реалізувати гірлянду.
3. Реалізувати керування конвеєром.
4. Реалізувати систему керування, що у нормальному режимі працює згідно ПІ-алгоритму, а в аварійному (визначається за зовнішнім сигналом) переходить на ручне керування. Об’єкт керування — аперіодична ланка другого порядку з коефіцієнтом передачі, виконавчий механізм -постійної швидкості типу МЕО.
5. Розробити систему керування, що, в залежності від виду продукту, що зберігається в холодильнику (м’ясо, овочі, морозиво, напої) обирає різні уставки (завдання) для регулятора температури. Температура регулюється компресором холодильника за принципом «ввімкнено»-«вимкнено». Передбачити захист компресора від занадто частого ввімкнення/вимкнення для запобігання його швидкого виходу з ладу.

5.3 Методичні вказівки

Реалізація функціональних блоків і програм в SFC має істотну особливість. Відсутні перша і остання інструкції. Оператор RETURN також не використовується. Програма як би не має кінця. Кожен виклик SFC POU

рівноцінний виконанню одного циклу. Що конкретно буде виконувати POU, залежить від його попереднього стану.

Примусово повернути компонент в початковий стан можна тільки шляхом скидання ПЛК. Примусова активація початкового кроку в SFC не означає автоматичне скидання компонента. Вона приведе тільки до того, що, окрім поточних активних дій, активним ще стане і початковий крок. Початковий крок не містить прихованих дій. Він не забороняє інші кроки і дії.

Відпрацювання реакції на всі необхідні події, включаючи екстрені, повинне бути передбачене в SFC явним чином. Переведення системи в початковий або безпечний стан передбачає для ПЛК установку заданого положення виконавчих механізмів і управління ними. Натискання аварійної клавіші, що знеструмлює виконавчі механізми, повинно коректно відпрацьовуватися програмним забезпеченням. Установка і підтримка безпечного стану системи - це така ж робота, як і нормальне функціонування. Не варто для цих цілей використовувати програмне скидання ПЛК, тим більше що така функція в стандартних бібліотеках відсутня.

У CoDeSys екстренне скидання SFC-програм і функціональних блоків все ж таки можливе. Проблема вирішується за допомогою спеціальних системних прапорців (SFCInit, SFCReset), керуючих роботою SFC-виконавця.

5.4 Порядок виконання роботи

17. Підібрати програмований логічний контролер та апаратне забезпечення, на яких можуть бути реалізоване поставлені завдання;
18. виконати загальні завдання;
19. виконати додаткове завдання (згідно номеру бригади);
20. оформити звіт про виконання роботи.

5.5 Контрольні запитання

1. Концепція мови SFC.
2. Кроки і переходи.
3. Алгоритми в кроках.
4. Спрощений варіант SFC.
5. МЕК-варіант SFC.
6. Класифікатори дій.

НАВЧАЛЬНО-МЕТОДИЧНІ МАТЕРІАЛИ

1. Руководство пользователя по программированию ПЛК в CoDeSys 2.3 – 455 с. // Електронний архів кафедри АТЕП ТЕФ НТУУ “КПІ”
2. Керівництво користувача FirstSteps в CoDeSys 2.3 – 15 с. // Електронний архів кафедри АТЕП ТЕФ НТУУ “КПІ”
3. Деменков Н.П. Языки программирования промышленных контроллеров: Учебное пособие / Н.П. Деменков, под редакцией К.А. Пупкова. — М. : МГТУ им.Н.Э.Баумана, 2004.- 172 с.
4. Петров И.В. Программируемые логические контроллеры / И.В. Петров. — М.: СОЛОН-Пресс, 2004. – 256 с.
5. Денисенко В.В. Компьютерное управление технологическим процессом, экспериментом, оборудованием. / В.В. Денисенко – М.: Горячая линия–Телеком, 2009. – 608 с.

ДОДАТКОК А. ПЕРЕЛІК ОПЕРАТОРІВ IL

Таблиця – Оператори завантаження та зберігання

Оператор	Модифікатор	Призначення	Операнди	Опис
LD	N (тільки для операндів типу BOOL, BYTE, WORD або DWORD)	Завантаження значень операндів до акумулятора	Літерал, змінна, пряма адреса будь-якого типу даних	Значення операнда завантажується в акумулятор, використовуючи LD. Розмір даних акумулятора авто етично налаштовується до типу даних операції. Це також дійсно для похідних типів даних. Приклад: значення A завантажується в акумулятор з додаванням значення B, і результат зберігається в E. LD A ADD B ST E
ST	N (тільки для операндів типу BOOL, BYTE, WORD или DWORD)	Збереження значення акумулятора в операнді	Змінна, пряма адреса будь-якого типу даних	Поточне значення акумулятора зберігається в операнді, використовуючи ST. Тип даних операнда повинен відповідати «типу даних» акумулятора. «Старий» результат використовується надалі, в залежності від того, чи використовується Ld за ST чи ні. Приклад: значення A завантажується в акумулятор з додаванням значення B, результат зберігається в E. Значення E (поточне значення акумулятора) віднімається від значення B, а результат зберігається в C. LD A ADD B ST E SUB 3 ST C

Таблиця – Оператори встановлення та скидання

Оператор	Модифікатор	Призначення	Операнди	Опис
S	-	Установка операнда в 1, коли вміст акумулятора = 1	Змінна, пряма адреса даних типу BOOL	S установка операнда в "1", коли поточне значення акумулятора дорівнює логічній 1. Приклад: значення A завантажується в акумулятор, якщо вміст акумулятора (значення A) =1, тоді OUT встановлюється в 1. LD A S OUT Зазвичай цей оператор використовується у парі з оператором скидання R. Приклад: тут показаний RS тригер (з приорітетом по скиданню), який керується логічними змінними A та C. LD A

Оператор	Модифікатор	Призначення	Операнди	Опис
				S OUT LD C R OUT
R	-	Установка операнда в 0, коли вміст акумулятора = 1	Змінна, пряма адреса даних типу BOOL	R установка операнда в «0», коли поточний вміст акумулятора дорівнює логічній 1. Приклад: значеннння A завантажується в акумулятор; якщо вміст акумулятора (значення A) = 1, тоді OUT встановлюється 0. LD A R OUT

Таблиця – Логічні оператори

Оператор	Модифікатор	Призначення	Операнди	Опис
AND	N, N(, (	Логічне І	Літерал, змінна, пряма адреса датин типу BOOL, BYTE, WORD чи DWORD	За допомогою AND виконується логічна операція І між вмістом акумулятора та операндом. Для BYTE, WORD і DWORD типів даних операція виконується побітово. Приклад: D = 1, якщо A, B та C = 1. LD A AND B AND C ST D
OR	N, N(, (	Логічне АБО	Літерал, змінна, пряма адреса датин типу BOOL, BYTE, WORD чи DWORD	За допомогою OR виконується логічна операція АБО між вмістом акумулятора та операндом. Для BYTE, WORD та DWORD типів даних, операція виконується побітово. Приклад: D = 1, якщо A або B = 1 та C = 1. LD A OR B OR C ST D
XOR	N, N(, (	Логічне виключаюче АБО	Літерал, змінна, пряма адреса датин типу BOOL, BYTE, WORD чи DWORD	За допомогою XOR виконується логічна операція Виключаюче АБО між вмістом акумулятора та оерандом. Для BYTE, WORD та DWORD типів даних, зв’язок виконується побітово. Приклад: D = 1, якщо A або B = 1. Якщо A та B мають однаковий стан (обидва 0 чи 1), то D = 0. LD A XOR B ST D Якщо зв’язуються більше двох операндів, результат для непарного числа 1-стану = 1, та 0 для парної кількості одинарних станів. Приклад: F = 1, якщо 1 або 3 операнди = 1. F = 0, якщо 0, 2 чи 4 операнди = 1. LD A

Оператор	Модифікатор	Призначення	Операнди	Опис
				XOR B XOR C XOR D XOR E ST F
NOT	-	Логічне заперечення	Вміст акумулятора типу даних BOOL, BYTE, WORD або DWORD	Вміст акумулятора інвертується NOT. Приклад: B = 1, якщо A = 0; B = 0, якщо A = 1. LD A NOT ST B

Таблиця – Арифметичні оператори

Оператор	Модифікатор	Призначення	Операнди	Опис
ADD	(	Додавання	Літерал, змінна, пряма адреса даних типу INT, DINT, UINT, UDINT, REAL або TIME	За допомогою ADD значення операнда додається до значення акумулятора.. Приклад: реалізація формули D = A + B + C виглядатиме так: LD A ADD B ADD C ST D
SUB	(	Віднімання	Літерал, змінна, пряма адреса даних типу INT, DINT, UINT, UDINT, REAL або TIME	За допомогою SUB значення операнда віднімається від вмісту акумулятора. Приклад: Формула D = A - B – C виглядає так: LD A SUB B SUB C ST D
MUL	(	Множення	Літерал, змінна, пряма адреса даних типу INT, DINT, UINT, UDINT, REAL або TIME	За допомогою MUL вміст акумулятора множиться на значення операнда. Приклад: Формула D = A * B * C запишеться так: LD A MUL B MUL C ST D
DIV	(	Ділення	Літерал, змінна, пряма адреса даних типу INT, DINT, UINT, UDINT, REAL або TIME	За допомогою DIV міст акумулятора ділиться на значення операнда. Приклад: Вираз D = A / B / C прийме вигляд: LD A DIV B DIV C ST D
MOD	(	Залишок від ділення	Літерал, змінна, пряма адреса даних типу INT, DINT, UINT, UDINT, REAL або TIME	Для MOD значення першого операнда ділиться на значення другого операнда, а залишок від ділення повертається як результат. Приклад: C = 1, якщо A = 7 та B = 2 C = 1, якщо A = 7 та B = -2 C = -1, якщо A = -7 та B = 2

Оператор	Модифікатор	Призначення	Операнди	Опис
				C = -1, якщо A = -7 та B = -2 LD A MOD B ST C

Таблиця – Оператори порівняння

Оператор	Модифікатор	Призначення	Операнди	Опис
GT	(	Порівняння: >	Літерал, змінна, пряма адреса даних типу BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT або TOD	За допомогою GT вміст акумулятора порівнюється з містом операнда. Якщо вміст акумулятора більше, ніж вміст операнда, результат – логічна 1, інакше – логічний 0. Приклад: D = 1, якщо A більше 10, інакше значення D = 0. LD A GT 10 ST D
GE	(	Порівняння: >=	Літерал, змінна, пряма адреса даних типу BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT або TOD	За допомогою GE вміст акумулятора порівнюється з містом операнда. Якщо вміст акумулятора більше або рівний вмісту операнда, результат – логічна 1, інакше – логічний 0. Приклад: D = 1, якщо A більше або рівне 10, інакше значення D = 0. LD A GE 10 ST D
EQ	(	Порівняння: =	Літерал, змінна, пряма адреса даних типу BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT або TOD	За допомогою EQ вміст акумулятора порівнюється з містом операнда. Якщо вміст акумулятора рівний вмісту операнда, то результат – логічна 1, інакше – логічний 0. Приклад: D = 1, якщо A дорівнює 10, інакше значення D = 0. LD A EQ 10 ST D
NE	(	Порівняння: <>	Літерал, змінна, пряма адреса даних типу BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT або TOD	За допомогою NE вміст акумулятора порівнюється з містом операнда. Якщо вміст акумулятора не дорівнює вмісту операнда, то результат – логічна 1, інакше – логічний 0. Приклад: D = 1, якщо A не дорівнює 10, інакше значення D = 0. LD A NE 10 ST D
LE	(	Порівняння: <=	Літерал, змінна, пряма адреса даних типу BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT,	За допомогою LE вміст акумулятора порівнюється з містом операнда. Якщо вміст акумулятора менший або рівний за вміст операнда, результат – логічна 1, інакше – логічний 0. Приклад: D = 1, якщо A менше або

Оператор	Модифікатор	Призначення	Операнди	Опис
			UDINT, REAL, TIME, DATE, DT або TOD	рівне 10, інакше значення D = 0. LD A LE 10 ST D
LT	(	Порівняння: <	Літерал, змінна, пряма адреса даних типу BOOL, BYTE, WORD, DWORD, STRING, INT, DINT, UINT, UDINT, REAL, TIME, DATE, DT або TOD	За допомогою LT вміст акумулятора порівнюється з містом операнда. Якщо вміст акумулятора менше за вміст операнда, то результат – логічна 1, інакше – логічний 0. Приклад: D = 1, якщо A менше 10, інакше значення D = 0. LD A LT 10 ST D

Таблиця – Оператори виклику

Оператор	Модифікатор	Призначення	Операнди	Опис
CAL	C, CN (тільки якщо вміст акумулятора BOOL типу даних)	Виклик функціонального блоку, або програми	Ім’я екземпляра функціонального блоку або програми	Викликається функціональний блок и програма
FUNCTIONSNAME	-	Виконання функції	Літерал, змінна, пряма адреса (ти даних залежить від функції)	Функція виконується за ім’ям функції

Таблиця – Оператори структурування

Оператор	Модифікатор	Призначення	Операнди	Опис
JMP	C, CN (тільки якщо вміст акумулятора BOOL типу даних)	Стрибок до мітки	TAG (Мітка)	За допомогою JMP перехід до мітки може бути обмеженим чи необмеженим.
RET	C, CN (тільки якщо вміст акумулятора BOOL типу даних)	Повернення до наступної більш високої організаційної одиниці програми	-	Кожна підпрограма залишається після виконання всіх операторів, тобто відбувається повернення до основної програми, що викликала поточну. Якщо підпрограма залишається передчасно, повернення до основної програми може бути форсоване, використовуючи RET (Повернення).
)	-	Редагування відкладених операцій	-	За допомогою дужки, що закриває,) починається виконання відновленого оператора. Число дужок, що закривають, повинне бути рівним числу дужок, що відкривають. Дужки можуть бути вкладеними. Приклад: E = 1, якщо C та/або D = 1 та A і B = 1.

Оператор	Модифікатор	Призначення	Операнди	Опис
				LD A AND B AND(C OR D) ST E

Перелік операторів ST

Таблиця – Операторы мови програмування ST

Оператор	Призначення	Операнд	Пріоритет	Опис
()	Взяття в дужки	Вираз	1 (найвищий)	Зміна порядку операцій у виразі. Якщо операнди A, B, C та D мають значення 1, 2, 3 й -4, то A+B-C*D= -9, а (A+B-C)*D = 0.
EXPT	Піднесення до степеня	Вираз, літерал, змінна, пряма адреса базових числових типів	3	При піднесенні до степеня значення першого операнда (основа) зводиться до степеня другого операнда (показника). OUT := EXPT(IN1, IN2); У прикладі змінна OUT дорівнюватиме 625.0, якщо IN1=5.0 та IN2=4.0.
–	Зміна знаку	Вираз, літерал, змінна, пряма адреса базових числових типів	4	За допомогою зміни занку відбувається реверсування знака значення операнда. OUT := - IN1 ; OUT дорівнюватиме -4, якщо IN1=4.
NOT	Інверсія	Вираз, літерал, змінна, пряма адреса базових логічних типів	4	За допомогою операнда NOT виконується порозрядна інверсія операнда. OUT := NOT IN1 ; OUT дорівнюватиме 0011001100, якщо IN1=1100110011.
*	Множення	Вираз, літерал, змінна, пряма адреса базових числових типів	5	При множенні значенн першого операнда множиться на значення другого OUT := IN1 * IN2 ;
/	Ділення	Вираз, літерал, змінна, пряма адреса базових числових типів	5	При діленні значення першого операнда ділиться на значення другого операнда. OUT := IN1 / IN2 ;
MOD	Остача від ділення (модуль ділення)	Вираз, літерал, змінна, пряма адреса базових числових типів	5	В операторі MOD значення першого операнда ділиться на значення другого, а остача повертається як результат OUT := IN1 MOD IN2 ; OUT дорівнюватиме 1, якщо IN1=7 та IN2=2.

Оператор	Призначення	Операнд	Пріоритет	Опис
+	Додавання	Вираз, літерал, змінна, пряма адреса базових числових типів та типу TIME	6	При додаванні значення першого операнда шумується зі значенням другого операнда. OUT := IN1 + IN2 ;
—	Віднімання	Вираз, літерал, змінна, пряма адреса базових числових типів та типу TIME	6	При відніманні значення другого операнда віднімається від значення першого операнда. OUT := IN1 - IN2 ;
<	Менше ніж	Вираз, літерал, змінна, пряма адреса	7	Операція порівняння двох операндів. Результат логічна «1», якщо перший операнд менший за другий, інакше результат — логічний «0». Якщо A=3, а B=5, то результат A<B буде «1» (TRUE)
>	Більше ніж	Вираз, літерал, змінна, пряма адреса	7	Операція порівняння двох операндів. Результат логічна «1», якщо перший операнд більший за другий, інакше результат — логічний «0». Якщо A=3, а B=5, то результат A>B буде «0» (FALSE)
<=	Менше або рівне	Вираз, літерал, змінна, пряма адреса	7	Операція порівняння двох операндів. Результат логічна «1», якщо перший операнд менший за другий або рівний йому, інакше результат — логічний «0». Якщо A=3, а B=5, то результат A<=B буде «1» (TRUE)
>=	Більше або рівне	Вираз, літерал, змінна, пряма адреса	7	Операція порівняння двох операндів. Результат логічна «1», якщо перший операнд більший за другий або рівний йому, інакше результат — логічний «0». Якщо A=3, а B=5, то результат A>=B буде «0» (FALSE)
=	Рівність	Вираз, літерал, змінна, пряма адреса	8	Операція порівняння двох операндів. Результат логічна «1», якщо перший операнд дорівнює другому, інакше результат — логічний «0». Якщо A=3, а B=5, то результат A=B буде «0» (FALSE)
<>	Не дорівнює	Вираз, літерал, змінна, пряма адреса	8	Операція порівняння двох операндів. Результат логічна «1», якщо перший операнд не рівний другому, інакше результат — логічний «0». Якщо A=3, а B=5, то результат A<>B буде «1» (TRUE)
AND	Логічне І	Вираз, літерал, змінна, пряма адреса булевого типу	9	При використанні оператора «Логічне І» відбувається логічна операція І між операндами

Оператор	Призначення	Операнд	Пріоритет	Опис
				OUT := IN1 AND IN2 AND IN3 ; OUT дорівнюватиме 1, якщо змінні IN1, IN2 та IN3 рівні 1.
XOR	Логічне виключаюче АБО	Вираз, літерал, змінна, пряма адреса булевого типу	10	За допомогою оператора XOR виконується логічна операція «логічне виключаючи АБО» між операндами. OUT := IN1 XOR IN2 ; OUT дорівнюватиме 1, якщо змінні IN1, IN2 не рівні. Якщо змінні IN1 та IN2 мають однаковий стан (обидві 0 чи 1), змінна OUT дорівнюватиме 0.
OR	Логічне АБО	Вираз, літерал, змінна, пряма адреса булевого типу	11 (найнижчий)	За допомогою оператора OR реалізується логічна операція «АБО» між операндами. OUT := IN1 OR IN2 OR IN3 ; Тут OUT буде рівною 1, кщо хоча б одна зі змінних IN1, IN2 чи IN3 дорівнюватиме 1.

ДОДАТОК Б. ЗМІСТ ЗВІТУ

Звіт про виконання лабораторної роботи оформлюється у наступному вигляді

1. Титульний аркуш
2. Зміст
3. Постановка завдання (загального та додаткового)
4. Теоретичні відомості
5. Виконання роботи

5.1. Підбір технічного обладнання.

Описати, які засоби потрібні для реалізації вашої задачі - який ПЛК, які модулі, які датчики та виконавчі механізми/електрокомутуюча апаратура. Чому прийняли саме такі рішення.

5.2. Програмна реалізація завдання

У ході роботи обов'язково наводити:

- словесний опис Алгоритму вирішення задачі (декомпозиція, логічні міркування, акцент на особливостях тощо);
- програмний код (з коментарями про суть роботи відкоментованого фрагменту коду);

5.3. Результати симуляції:

обов'язково наводити результат симуляції/моделювання (в статиці та динаміці) для всіх передбачених випадків роботи програми з поясненням, що ми бачимо на ілюстраціях.

6. Висновки.

У висновках концентруватися на суті - яка задача поставлена, як її вирішили, які інструментарії застосували, які алгоритми використали, які нюанси зустріли, які проблеми вирішили