НАВЧАЛЬНО-МЕТОДИЧНА ЛІТЕРАТУРА

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя

Факультет прикладних інформаційних технологій
та електроінженерії
Кафедра автоматизації технологічних
процесів і виробництв

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи №24
«Розробка проекту автоматизації у середовищі програмного забезпечення «Factory I/O»-CODESYS»

з курсу «Проектування систем автоматизації »

для студентів спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології»

Тернопіль

Міністерство освіти і науки України

Тернопільський національний технічний університет
ім. Івана Пулюя

Факультет прикладних інформаційних технологій
та електроінженерії
Кафедра автоматизації технологічних
процесів і виробництв

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи №24
«Розробка проекту автоматизації у середовищі програмного забезпечення «Factory I/O»-CODESYS»

з курсу «Проектування систем автоматизації »

для студентів спеціальності 151 – «Автоматизація та комп'ютерно-інтегровані технології»

Тернопіль, 2018

Методичні вказівки до лабораторної роботи № 24 «Розробка проекту автоматизації у середовищі програмного забезпечення Factory I/O -CODESYS» з курсу «Проектування систем автоматизації» / Шкодзінський О.К., Пісьціо В.П., Сікора Д.А. - Тернопіль: ТНТУ, 2018 - 22 с.
Для студентів спеціальності 151 Автоматизація та комп'ютерно-інтегровані технології

Укладачі: Шкодзінський О.К., Пісьціо В.П., Сікора Д.А.

Методічні вказівки розглянуті, схвалені і затверджені на засіданні кафедри автоматизації технологічних процесів та виробництв Тернопільського національного технічного університету імені Івана Пулюя
(протокол № 11 від 26 лютого 2018 року).

Рецензент: доцент кафедри автоматизації технологічних процесів та виробництв Тернопільського національного технічного університету імені Івана Пулюя, к.т.н. Коноваленко І.В.

Тема: Розробка проекту автоматизації у середовищі програмного забезпечення «Factory I/O» - CODESYS.

Мета: Навчитися розробляти різноманітні проекти автоматизації відповідно до поставленого завдання у середовищі програмного забезпечення «Factory I/O» - CODESYS.

1.ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1. Парадигма програмування

Парадигма програмування — це система ідей і понять, які визначають стиль написання комп'ютерних програм, а також спосіб мислення програміста. Також, це — спосіб концептуалізації, що визначає організацію обчислень і структурування роботи, яку виконує комп'ютер.

Парадигма програмування унаочнює те, як програміст розглядає роботу програми; наприклад, за ООП (Об'єктно-орієнтоване програмування) — як множини об'єктів, тоді як за ФП (Функційне програмування) — як послідовності обчислень функцій без станів.

Кожну окрему парадигму програмування характеризує наявність у ній методу та зв'язку із моделлю життєвого циклу. Спільним для різних парадигм програмування є загальні принципи проектування програмного продукту. Користувач може вибирати ту чи іншу парадигму програмування з позицій зручності застосування для створення конкретного програмного продукту.

Важливо відзначити, що парадигма програмування не визначається однозначно мовою програмування; практично всі сучасні мови програмування в тій чи іншій мірі допускають використання різних парадигм (мульти-парадигмальність програмування).

1.2. Дискретний автомат

Цифровий (дискретний) автомат – пристрій, який здійснює прийом, зберігання та перетворення дискретної інформації за деяким алгоритмом. У певному розумінні до автоматів можна віднести як реальні пристрої (обчислювальні машини, живі організми тощо), так і абстрактні системи (математичні машини, аксіоматичні теорії тощо).

Загальну теорію автоматів поділяють на *абстрактну* і *структурну*. Відмінність між ними полягає в тому, що абстрактна теорія, відсторонюючись від структури автомата (тобто не цікавлячись способом його побудови), вивчає лише поведінку автомата відносно зовнішнього середовища. Абстрактна теорія автоматів, отже, близька до теорії алгоритмів, будучи, по суті, її подальшою

деталізацією. В протилежність абстрактній теорії структурна теорія цікавиться як структурою самого автомата, так і структурою вхідних дій і реакцією автомата на них. У структурній теорії вивчаються способи побудови автоматів, способи копіювання вхідних дій і реакцій автомата. Таким чином, структурна теорія автоматів є продовженням і подальшим розвитком абстрактної теорії. Спираючись на апарат булевих функцій і на абстрактну теорію автоматів, структурна теорія дає ефективні рекомендації щодо розробки реальних пристроїв обчислювальної техніки. Найвідомішим прикладом де використовуються автомати є машина Тюрінга.

Автомат складається з: вхідної стрічки, керуючого пристрою, допоміжної або робочої пам'яті. Приклад взаємодії цих елементів можна побачити на наступному рисунку (рис.1).

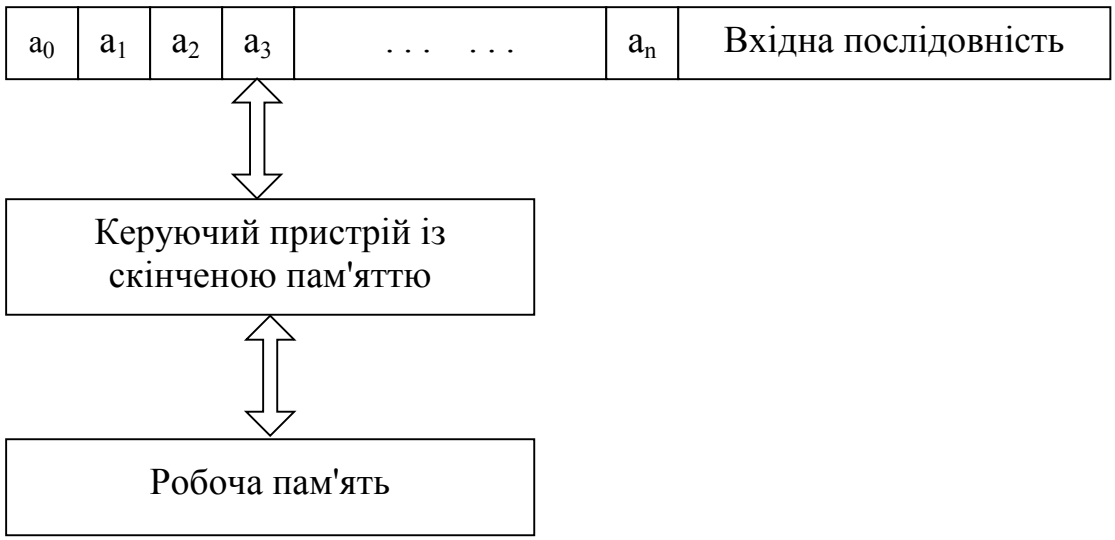


Рис.1. Взаємодія елементів автомата

Вхідна стрічка

Її можна розглядати як лінійну послідовність клітинок, або комірок, причому кожна комірка точно містить один вхідний символ з деякого скінченного вхідного алфавіту. За допомогою вхідної стрічки зображується інформація, що поступає на вхід автомата.

Вхідна голівка

У кожний момент **вхідна голівка** читає одну вхідну комірку. За один крок роботи автомата вхідна голівка може зміститися на одну комірку вліво, вправо або залишитися нерухомою. Автомат, який не переміщує свою голівку називається одностороннім.

Пам'ять автомата

Пам'ять автомата - структура, в якій записуються, зберігаються і зчитуються додаткові дані, що використовуються автоматом при роботі. Для кожного виду автоматів строго визначено тип пам'яті.

Робота автомата складається з послідовності тактів. Кожен такт складається з таких дій:

1. Читається поточний вхідний символ.
2. За допомогою функції доступу досліджується пам'ять, і до неї заноситься деяка інформація.
3. Змінюється стан керуючого пристрою.
4. Записується вихідна інформація у комірки вхідної стрічки.
5. Вхідна голівка зміщується на одну комірку вліво, вправо або зберігається у початковому стані.

Поточний вхідний символ та інформація, що витягнута з пам'яті, разом з поточним станом керуючого пристрою визначають, яким повинен бути цей такт. Таким чином, на кожному такті визначається поточна конфігурація автомата, до якої входять:

- поточний стан керуючого пристрою;
- поточний зміст вхідної стрічки;
- поточний зміст робочої пам'яті.

Конфігурація автомата змінюється на кожному такті під впливом керуючого пристрою.

Керуючий пристрій

Складається зі скінченної множини станів $S = \{s_0...s_n\}$ і відображень, які залежно від попередньої конфігурації дозволяють визначити нову конфігурацію автомата, напрямок переміщення вхідної голівки (якщо вона не одностороння), інформацію для друку (якщо в автоматі передбачено функцію друку), інформацію, що заносять у пам'ять і витягують звідти (якщо автомат має робочу пам'ять). Є два види керуючого пристрою:

- **Недетермінований** - для кожної конфігурації існує скінченна множина можливих конфігурацій (більше однієї), так що в будь-яку з них автомат може перейти на наступному кроці.
- **Детермінований** - для кожної конфігурації існує не більше однієї можливої наступної конфігурації.

1.3. Автоматне програмування

Автоматне програмування — парадигма програмування, відповідно до якої програма або її фрагмент розглядаються як модель деякого

формального **автомата** (покрокового перетворювача інформації), а вибір операції обробки вхідних даних визначається не лише поточним значенням даних, але й поточним значенням стану автомата.

Залежно від конкретної задачі в автоматному програмуванні можуть використовуватися як кінцеві автомати, так і автомати більш складної структури.

Визначальними для автоматного програмування є такі особливості:

1. Послідовність операцій виконання програми розбивається на *кроки автомата*, кожен з яких являє собою виконання певної (однієї і тієї ж для кожного кроку) секції коду з єдиною точкою входу; така секція може бути оформлена, наприклад, у вигляді окремої функції і, в свою чергу, може бути розділена на підсекції, відповідно до окремих станів або категорій станів;
2. Збереження інформації про стан між кроками автомата здійснюється виключно через явно визначений набір змінних, призначених для зберігання *стану автомата*, таких, що стани програми на будь-якому окремому моменті входу в *крок автомата* можуть різнитися між собою виключно значеннями цих і тільки цих змінних.
3. Зміна станів автомата відбувається в циклі (можливо, неявно заданому) виконання кроків автомата.

Основним поняттям в автоматному програмуванні є стан. При написанні автоматних програм пропонується розділяти стани на два класи: *керуючі* і *обчислювальні*. При цьому за допомогою невеликого числа керуючих станів, як і в машині Тюрінга, можна управляти як завгодно великим числом обчислювальних станів. У введеній класифікації керуючі стани можуть бути названі *якісними*, а обчислювальні - *кількісними*. В рамках автоматного програмування основна увага приділяється керуючим станам, які і будуть розглядаються в подальшому.

При цьому справедливим є співвідношення: «Стан + вхідний вплив = кінцевий автомат без виходу». Також справедливо: «Автомат без виходу + вихідні впливи = автомат».

Автомати можуть бути абстрактними (вхідні та вихідні впливи формуються послідовно) і структурними (вхідні та вихідні впливи формуються «паралельно»). Час в автоматах у явному вигляді не використовується. При потребі застосовуються елементи затримки, вони розглядаються як об'єкти керування. При цьому затримки запускаються і скидаються з автоматів, а інформація про закінчення часу надходить до них у вигляді вхідних впливів.

Автомати можуть задаватися в різному вигляді, проте при їх проектуванні та використанні людиною, вони повинні володіти когнітивними властивостями, що досягається при заданні поведінки автоматів у вигляді графів переходів

(діаграм станів). У разі якщо автомати генеруються автоматично, то вони можуть задаватися інакше, наприклад, в табличній формі.

При цьому потрібно відзначити, що навіть при порівняно невеликому числі станів і переходів, таке задання ускладнює розуміння роботи автоматів людиною, так як відображення переходів в них не є наочним, проте їх реалізація може стати більш швидкодіючою.

Зрозумілість графів переходів досягається за рахунок того, що стани всіх вхідних впливів відповідного автомата декомпонуються на групи, кожна з яких визначає переходи з розглянутого стану.

1.4. Переваги автоматного програмування

До сильних сторін автоматного програмування (АП) можна віднести таке:

1. *Простота і, як наслідок, надійність.* Прагнення триматися рамок єдиного автоматного формалізму змушує строгіше ставитися до створюваного програмного коду. Цей код створюється з невеликих за обсягом «конструктивних» елементів. В цьому і полягає простота. Короткі, уніфіковані, побудовані за єдиним принципом процедури легше контролювати.
2. *Наочність алгоритму.* Складність алгоритму поведінки нікуди не ділась. Просто вона перенесена з програмного коду назовні, в граф переходів (діаграму станів), - яким користується програміст, реалізуючи автомат. А з програмної точки зору сам автомат разом з логікою його роботи представлений простою структурою - матрицею переходів. Аналізувати поведінку об’єкту керування, виходячи з графа автомата дійсно зручно і наочно. Це в якійсь мірі випливає з особливостей психології людського сприйняття.
3. *Гнучкість алгоритму.* Автоматне уявлення дозволяє поступово нарощувати складність і гнучкість поведінки об’єкту керування, не ламаючи при цьому логіку роботи всієї програми, що є вкрай важливо.

2. РІЗНОВИДИ АВТОМАТНОГО ПРОГРАМУВАННЯ

Автоматне програмування розвивається в трьох основних напрямках: *логічне керування*, *програмування з явним виділенням станів* та *об'єктно - орієнтоване програмування з явним виділенням станів.*

2.1. Логічне керування

Найбільш природньо застосовувати автомати в системах логічного керування, у яких вхідні і вихідні змінні є двійковими (0 та 1). Природність застосування автоматів при програмуванні цього класу систем пояснюється тим, що програми в них заміняються логічними схемами, проектування яких з

використанням автоматів є поширеним і розвивається давно, починаючи ще з релейно-контактних схем.

Однак простота і природність застосування автоматів, мабуть, далеко не очевидна, так як навіть при програмуванні логічних контролерів, практично ніхто не пропонував спочатку проектувати автомати, а потім реалізовувати їх на обраній мові програмування.

2.2. Програмування з явним виділенням станів

Більш широким класом у порівнянні з системами логічного керування є реактивні системи. Для опису поведінки таких систем, куди входить, більшість вбудованих систем, застосування автоматів також є природнім, як і для систем логічного керування. Реактивні системи є складнішими у порівнянні з системами логічного керування. У них:

- у якості вхідних впливів, поряд з вхідними змінними, використовуються стани;
- запуск програм здійснюється за станами, а не циклічно;
- в якості вихідних впливів можуть використовуватися не лише виконавчі, але й інші функції, що дозволяє називати автомати, що застосовуються - гібридними;
- автомати можуть містити не лише вкладені стани, а й вкладені автомати;
- автомати можуть взаємодіяти не лише за рахунок перевірки номерів станів, як було запропоновано для систем логічного керування, але також і за рахунок вкладеності, яка викликана обміном подіями (повідомленнями).

2.3. Об'єктно-орієнтоване програмування з явним виділенням станів

Вже два десятиліття об'єктно-орієнтоване програмування (ООП) є найбільш широко використовуваним стилем програмування у світі. При застосуванні об'єктної парадигми програми будують з об'єктів, що взаємодіють за рахунок обміну повідомленнями.

В теорії об'єктно-орієнтованого програмування вважається, що об'єкт має внутрішній стан і здатний отримувати повідомлення, відповідати на них, відправляти повідомлення іншим об'єктам і в процесі обробки повідомлень змінювати свій внутрішній стан. Більш наближене до практики, поняття виклику методу об'єкта, вважається синонімом поняття відправки повідомлення об'єкту.

Отже, з одного боку, об'єкти в об'єктно-орієнтованому програмуванні можуть розглядатися як кінцеві автомати (або, якщо завгодно, моделі кінцевих автоматів), стан яких являє собою сукупність внутрішніх полів, у якості кроку

автомата можуть розглядатися один або декілька методів об'єкта при умові, що ці методи не викликають ні самі себе, ні один одного ні прямо, ні побічно.

З іншого боку, очевидно, що поняття об'єкта є вдалим інструментом для реалізації моделі кінцевого автомата. При застосуванні парадигми автоматного програмування в об'єктно-орієнтованих мовах зазвичай моделі автоматів представляються у вигляді класів, стан автомата описується внутрішніми (закритими) полями класу, а код кроку автомата оформляється у вигляді методу класу, причому такий метод швидше за все виявляється єдиним відкритим методом (не рахуючи конструкторів і деструкторів), що змінюють стан автомата. Інші відкриті методи можуть служити для отримання інформації про стан автомата, але не змінюють його. Всі допоміжні методи (наприклад, методи-обробники окремих станів чи їх категорій) в таких випадках зазвичай прибирають в закриту частину класу.

3. РЕАЛІЗАЦІЯ АВТОМАТНОГО ПРОГРАМУВАННЯ

3.1. Етапи автоматного програмування

Процес реалізації АП можна розбити на 3 етапи:

1) Виділення з системи об’єктів, які можуть відповідати фізичним об’єктам, їх групам або їх частинам. В процесі такого виділення необхідно дотримуватись мети досягнення найбільш простого вигляду системи та найбільш простого її подальшого програмування. Зазвичай можна притримуватись правила: «один елемент системи - один об’єкт – одна змінна стану».

2) Після декомпозиції системи слід виділити стани кожного об’єкта. Бажано при цьому щоб зміна вихідних сигналів для керування об’єктами здійснювалась лише при змінної стану об’єкта.

3) Після виділення станів всіх об’єктів необхідно виділити умови переходу об’єкта з одного стану в інший. Такі умови можуть визначатись станом датчиків або набуттям якимось іншим об’єктом заданого свого стану.

Розглянемо декілька прикладів використання автоматного програмування.

3.2. Приклад задачі із однією змінною стану

3.2.1 Формалізація задачі

Спочатку розглянемо найпростішу задачу із використанням одного об’єкта та однієї змінної стану.

Нехай на конвеєрі потрібно згрупувати 3 коробки (рис.2), які будуть відправлені на вихід. Коробки поступають незалежно з вхідної позиції і

розміщуються на вхідному конвеєрі. По мірі надходження коробки передаються на буферний конвеєр, де накопичуються у кількості 3 штуки. Після накопичення (3-х коробок) вони передаються на вихід.

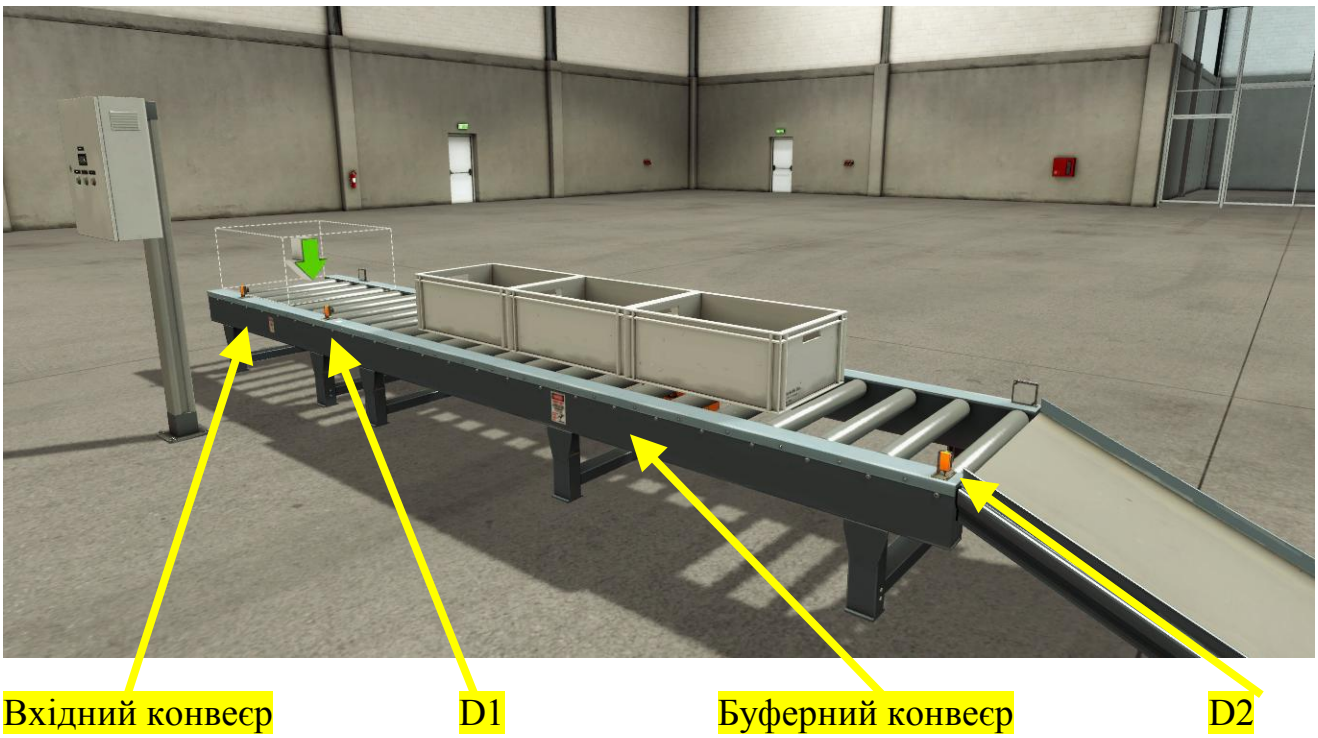


Рис.2. Приклад сцени у середовищі «Factory I/O»

В даній задачі будемо використовувати один об’єкт та одну змінну стану котра його описує. Легко зрозуміти, що в системі можливі 5 станів:

0) Усі конвеєри виключені. Цей стан є початковим. Вихід з даного стану здійснюється при запуску програми.

1) Система очікує надходження коробки. У даному стані вхідний конвеєр має бути включений, буферний виключений. Умовою виходу з даного стану буде спрацювання датчика (D1) на виході вхідного конвеєра, що буде вказувати на появу коробки на буферному конвеєрі. В такому випадку система переходить у стан 2.

2) У цьому стані коробка передається з вхідного конвеєра на буферний, виходом із даного стану буде відключення датчика (D1) на виході вхідного конвеєра. Зі стану 2 можливі переходи в стан 1 та в стан 3. Якщо на буферному конвеєрі находиться менше 3-х коробок, то здійснюється перехід на завантаження наступної коробки. Якщо ж на буферному конвеєрі є 3 коробки , система переходить у стан 3.

3) У 3 стані вхідний конвеєр виключається і здійснюється розвантаження буферного конвеєра. З 3 стану здійснюється перехід у 1 стан після вивантаження останньої коробки, що контролюється переключенням датчика (D2) встановленого на виході буферного конвеєра. В стан 4 здійснюється перехід у тому випадку коли датчик (D2) спрацьовує, але на конвеєрі ще залишились коробки.

4) У стані 4 продовжується вивантаження коробки з буферного конвеєра і здійснюється перехід у стан 3 після проходження коробки повз датчика.

Тобто система спочатку 3 рази переходить між станами 1 та 2, накопичуючи коробки на буферному конвеєрі, а потім шляхом переходів між станами 3 і 4 здійснює їх вивантаження. Фактично система працює між станами 1-2-1-2-1-2-3-4-3-4-3-4, а потім знову повторює цикл. При реалізації такого циклу нам необхідно буде мати лічильник кількості коробок, котрий буде збільшуватись на 1 (k+1) при переході між станами 1 та 2, і зменшуватись на 1 (k-1) при переході між станами 3 і 4. Діаграму станів до нашого прикладу зображена на рисунку нижче (рис.3).

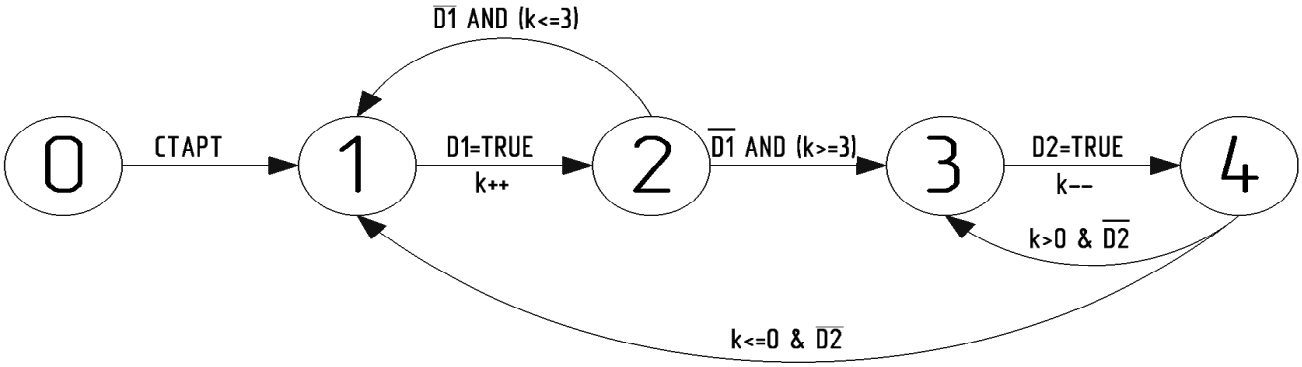


Рис.3. Діаграма станів до прикладу задачі із однією змінною стану

Як видно з опису крім змінних стану при автоматному програмуванні можна використовувати довільні змінні, котрі тим чи іншим чином визначають параметри об’єктів. В даному випадку такою додатковою змінною є лічильник кількості коробок.

3.2.2. Класичний підхід до програмування

В даному випадку роботу системи можна також описати у вигляді класичного алгоритму (**додаток А**).

Як видно з рисунку додатка А алгоритм є досить складним заплутаним і тяжким для модифікації. Тому опис роботи системи у вигляді алгоритму в даному випадку є незручним.

3.2.3. Програмування з використання змінної стану

Найпростіший приклад реалізації на мові ST (CODESYS) вказаної логіки роботи описаної вище системи показаний у **додатку Б**. Як видно основна управляюча змінна котра визначає стан системи реалізується в операторі CASE. Кожен підблок оператора CASE розділяється на 2 частини. У першій частині вказується, що необхідно включити і виключити для отримання даного стану. Наприклад включити вхідний конвеєр, виключити вихідний конвеєр. У другій частині вказується чим визначається умова виходу із даного стану, стан в який необхідно перейти, а також дії котрі здійснюються при даному переході (наприклад, збільшення або зменшення лічильника на 1 тощо). Умови можуть бути вкладеними якщо це потрібно (приклад умови реалізується у стані 2).

Лічильник збільшується та зменшується лише при переходів між станами, тому оператор збільшення та зменшення знаходиться у програмі в частині де здійснюється аналіз можливості переходу у новий стан.

3.3. Приклад програмування з використанням декількох змінних стану

Розглянемо більш складний випадок. Нехай система має 2 конвеєри для подачі палет та коробок (рис. 4), робота для перестановки коробок та вихідний конвеєр. Система призначена для складання коробок по дві штуки на одну палету і видачі їх на позицію розвантаження. Зрозуміло, що в такій задачі використовувати одну змінну стану не раціонально, так як фактично описуються три незалежних об’єкти:

- вхідний конвеєр на котрий поступають коробки;
- конвеєр для подачі палет;
- робот, що переставляє коробки.

Отже, будемо використовувати три об’єкти і три незалежні змінні: **BoxState**, **PaletState**, **RobotState** котрі описують відповідні об’єкти. У загальному, кількість об’єктів і відповідність змінних стану буде залежати від кількості матеріальних об’єктів і вимог, що пред'явлені до системи та до програми.

Для спрощення модифікації системи будемо притримуватись правила, що різні об’єкти можуть взаємодіяти між собою визначаючи стан один одного, але при цьому можуть змінювати лише свій стан.

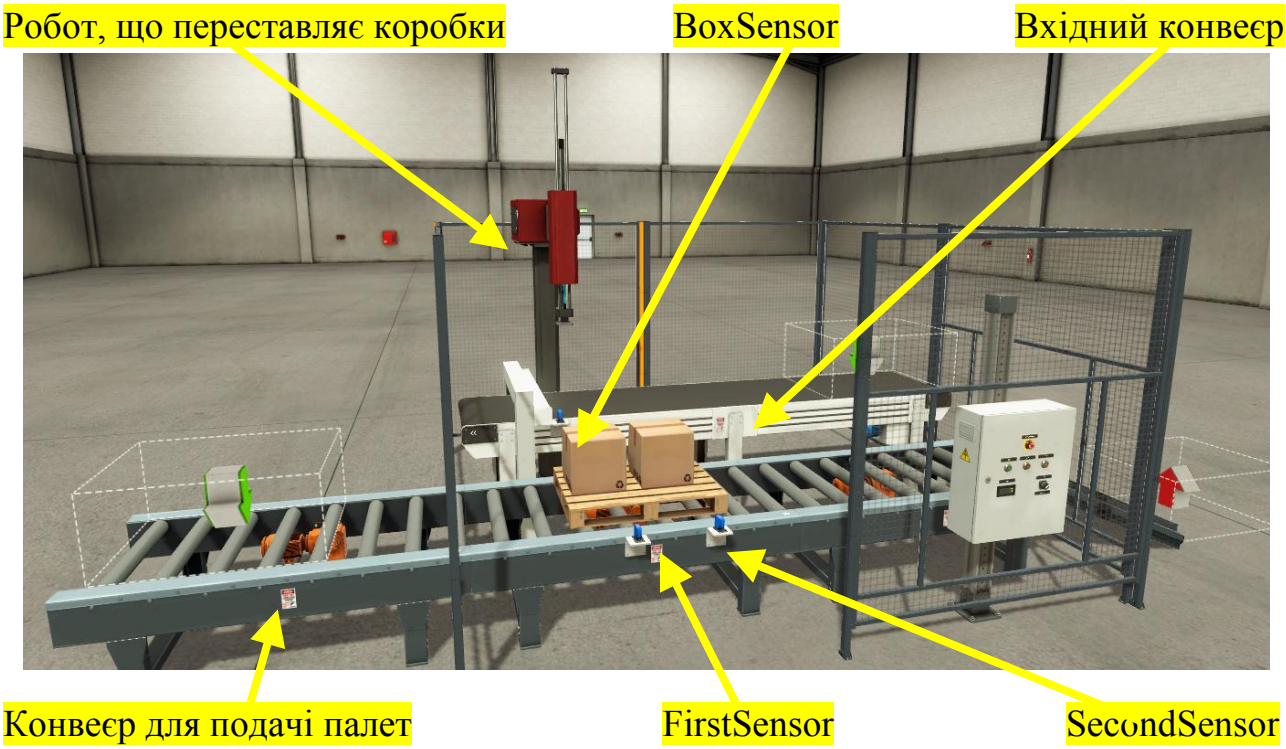


Рис.4. Приклад сцени у середовищі «Factory I/O»

Тепер опишемо відповідні стани об’єктів одночасно вказуючи умови переходів між станами.

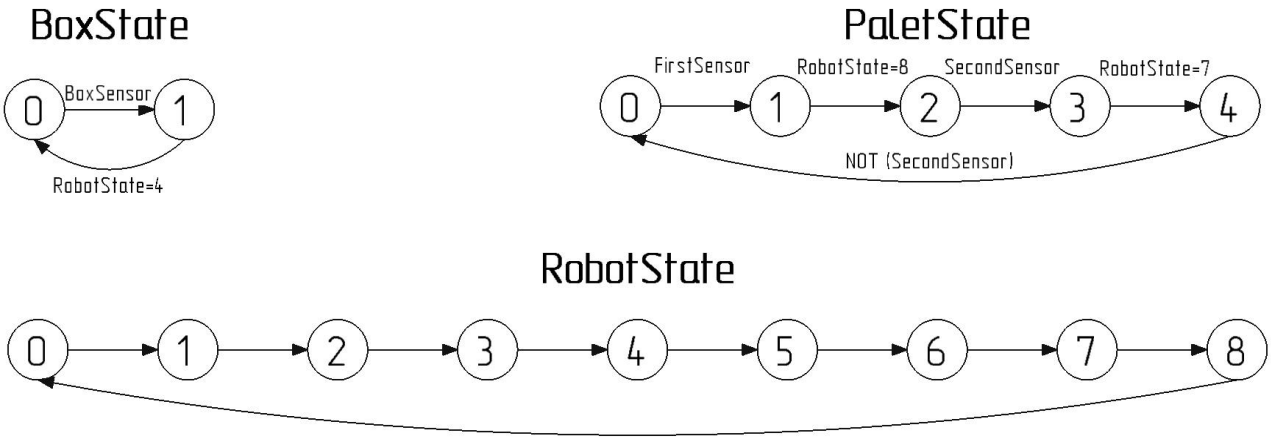


Рис.5. Діаграма станів до приклад з використанням декількох змінних стану

Вхідний конвеєр та відповідно змінна **BoxState** може набувати 2-х станів:

0) На вивантажувальній позиції відсутня коробка для вивантаження, конвеєр рухається. Умовою виходу з даного стану є спрацювання датчика **BoxSensor**.

1) Конвеєр зупинений, на вихідній позиції знаходиться коробка. Умовою виходу з даного стану служить стан робота який відповідає

переміщенню коробки. Тобто, вхідний конвеєр перейде із 0 в 1 доти, доки робот не забере коробку.

Конвеєр для подачі палет і відповідна змінна **PaletState** може набувати 5 значень:

0) Немає жодної палети для завантаження, двигуни відповідних конвеєрів включені. Виходом з даного стану буде спрацювання першого датчика позиціонування палети (**FirstSensor**). Коли датчик спрацює змінна **PaletState** набуває значення 1.

1) Палета знаходиться на 1 позиції. Двигун вхідного конвеєра виключений та двигун конвеєра завантажувача вимкнений. Перехід з 1-го стану у 2-й відбувається тоді коли робот поклав коробку на відповідну палету (і набув свій стан рівний 8).

2) У стані 2 палета рухається на позицію завантаження. Для цього включаються відповідний конвеєр для завантаження. Умовою виходу буде спрацювання 2-го датчика (**SecondSensor**), котрий відповідає 2-ій позиції завантаження.

3) У 3-му стані двигун конвеєра виключається і конвеєр очікує встановлення другої коробки роботом. Умовою виходу з даного стану буде встановлення коробки роботом. Після чого конвеєр для подачі палет переходить у стан вивантаження палет (стан 4).

4) У стані 4 вихідний конвеєр палет включається до тих пір поки датчик 2 позиції не виключиться. Після чого змінна **PaletState** (і відповідний конвеєр) набуває стан 0.

Робот та відповідна змінна **RobotState** може набувати 9 станів:

0) В 0-му стані захоплюючий пристрій виключений, рука робота втягнута і піднята. Умовою виходу з відповідного стану буде наявність коробки на позиції завантаження та спрацювання датчика наявності палети на першій чи другій позиції завантаження. У такому випадку робот переходить до стану 1.

1) У стані 1 робот здійснює опускання руки до тих пір доки це можливо (і поки рука рухається). Коли рух руки вниз закінчений, робот переходить у стан 2.

2) У стані 2 здійснюється захоплення коробки. Так як захоплюючий пристрій є повільним, то перехід в наступний стан здійснюється після спрацювання таймера. Після витримки часу робот переходить у стан 3.

3) У стані 3 рука робота піднімається вгору. Рука робота піднімається до тих пір поки це можливо. Після закінчення руху руки вгору робот переходить у стан 4.

4) У стані 4 робот витягує руку до спрацювання обмежувача. Коли рука зупиниться здійснюється перехід у стан 5.

5) У стані 5 робот опускає руку з коробкою вниз на відповідну палету. Вихід з даного стану здійснюється коли рука досягнула нижньої позиції. У цьому випадку здійснюється перехід у 6-й стан.

6) У стані 6 виключається захоплюючий пристрій і через певний час робот переходить у стан 7.

7) У стані 7 робот піднімає руку вгору. Умовою виходу зі стану буде неможливість руху руки. Перехід із стану 7 здійснюється у стан 8.

8) У стані 8 робот втягує руку. Умовою виходу є неможливість руху руки. Після закінчення циклу роботи робота він знову повертається до стану 0.

Діаграми станів, що описують зміни станів наших об’єктів показано на рисунку рис.5. Лістинг програми можна переглянути у **додатку В.**

Логіку роботи даної системи практично неможливо описати у вигляді єдиного алгоритму, тому він і не приводиться.

4. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Створити сцену згідно варіанту (таблиця 9) та вирішити поставлену проблему. Сцени НЕ є базовими і створювати їх прийдеться самим вручну. Також потрібно розробити програму керування для даної сцени та налагодити її на коректну роботу за допомогою програмного коду на мові ST.

Таблиця 1. Варіанти завдань

Варіант	Завдання
1	Транспортування палет з коробками з другого поверху на перший
2	Транспортування палет з коробками з першого поверху на другий
3	Вкладання поряд двох коробок на одну палету та відвантаження палети
4	Переміщення коробок на палетах, що відрізняються за висотою від заданої на паралельний конвеєр.
5	Перекладання вищих коробок з палети на палету на паралельному конвеєрі
6	Переміщення палет з паралелиних конієрів на один спільний конвеєр

5. ЗМІСТ ЗВІТУ

1) На титульній сторінці креслярським шрифтом вказати назву закладу, кафедри, назву і номер роботи, прізвище, ініціали, номер групи виконавця, прізвище та ініціали викладача, який керував роботою, рік виконання роботи.
2) Вказати тему та мету роботи.
3) Продемонструвати виконану роботу за допомогою скріншотів.
4) Подати таблиці з описом виконавчих механізмів і датчиків.
5) Подати діаграму станів обладнання.
6) Подати текст програми, доповнивши його коментарями.
7) Написати висновок до лабораторної роботи.

6. ПЕРЕЛІК ПОСИЛАНЬ

1. Методичні вказівки до лабораторної роботи № 21 на тему «Ознайомлення з основами роботи у середовищі програмного забезпечення «Factory I/O» та запуск готового проекту» з курсу «Проектування систем автоматизації» / Шкодзінський О.К., Пісьціо В.П., Сікора Д.А., Герасимів Ю.О. - Тернопіль: ТНТУ, 2018 - 20 с.

2. Методичні вказівки до лабораторної роботи № 22 на тему «Модифікація та відлагодження проекту у середовищі програмного забезпечення «Factory I/O»-COSESYS» з курсу «Проектування систем автоматизації» / Шкодзінський О.К., Пісьціо В.П., Сікора Д.А., Герасимів Ю.О. - Тернопіль: ТНТУ, 2018 - 18 с.

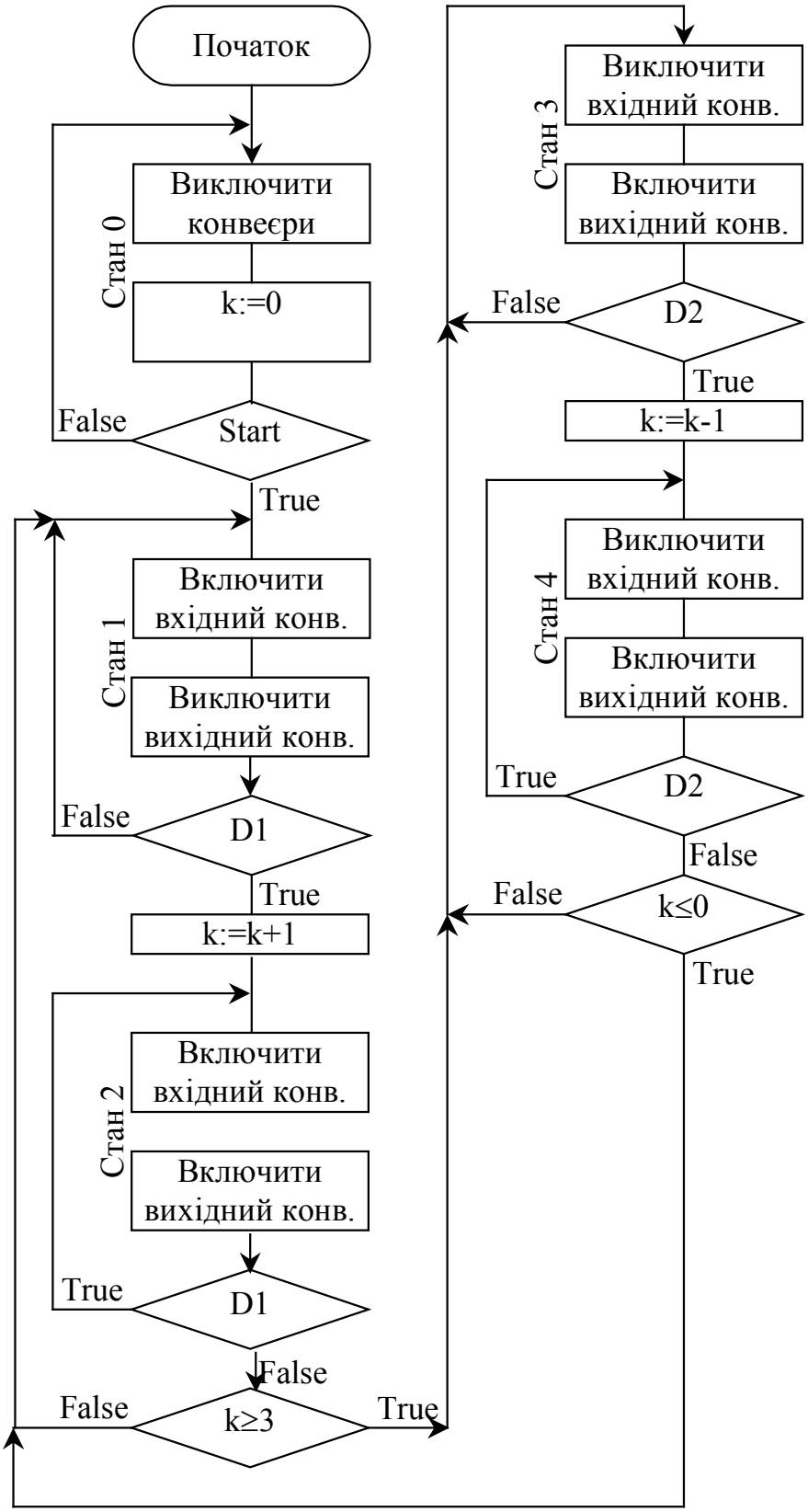
3. Методичні вказівки до лабораторної роботи № 23 на тему «Розробка та відлагодження програми керування технологічним обладнанням у середовищі програмного забезпечення «Factory I/O»-CODESYS» з курсу «Проектування систем автоматизації» / Шкодзінський О.К., Пісьціо В.П., Сікора Д.А. - Тернопіль: ТНТУ, 2018 - 17 с.

4. Шалыто, А. А. Автоматное програмирование [Текст] – Санкт-Петербургский государственный университет информационных технологий, механики и оптики 197101. – С.44.

5. Автоматное програмирование [Електронний ресурс] / Википедия — свободная энциклопедия. Режим доступу: https:// ru.wikipedia.org /wiki/ Автоматное_программирование

ЗМІСТ

Додаток А


Початок
Стан 0
Виключити конвеєри
k:=0
Start
False
True
Стан 1
Включити вхідний конв.
Виключити вихідний конв.
D1
False
True
k:=k+1
Стан 2
Включити вхідний конв.
Включити вихідний конв.
D1
True
False
k≥3
False
True
Стан 3
Виключити вхідний конв.
Включити вихідний конв.
D2
False
True
k:=k-1
Стан 4
Виключити вхідний конв.
Включити вихідний конв.
D2
True
False
k≤0
False
True

Додаток Б

```
PROGRAM PLC_PRG;
VAR State:BYTE:=0;
    k:INT;  //Лічильник кількості коробок
END_VAR

CASE state OF
0:  //Початковий стан
    GVL.InpConv:=FALSE;
    GVL.OutConv:=FALSE;
      IF gvl.Start THEN state:=1;
      END_IF
1: //стан 1
    GVL.InpConv:=TRUE; //Включити вхідний конвеєр
    GVL.OutConv:=FALSE;
    IF NOT(GVL.D1) THEN
       k:=k+1;       //Лічильник збільшується при переході із стану 1 у стан 2
       State:=2;
    END_IF;
 2:
     GVL.InpConv:=TRUE;
     GVL.OutConv:=TRUE;
       IF (GVL.D1) THEN
        IF (k<3) THEN
           State:=1;
        ELSE
           state:=3;
       END_IF;
      END_IF;
 3:
    GVL.OutConv:=TRUE;
    GVL.InpConv:=FALSE; //Виключити вхідний конвеєр
      IF NOT(GVL.D2) THEN
      k:=k-1;
      state:=4;
       END_IF;
  4:
    GVL.OutConv:=TRUE;
    GVL.InpConv:=FALSE;
    IF (GVL.D2) THEN
        IF (k>0) THEN  state:=3;
           ELSE state:=1;
         END_IF;
    END_IF;
   ELSE state:=0;
END_CASE
```

Додаток В

```
PROGRAM PLC_PRG
VAR
      BoxState: INT:=0;
      PaletState:INT:=0;
      RobotState:INT:=0;
      GTon:TON;
      GGrabOn:ton;
      TimerOn:BOOL:=FALSE;
      fq: BOOL;
END_VAR

GTon(In := TimerOn,PT:=T#0.5S);
GGrabOn(In:=TimerOn,PT:=T#3S);

CASE BoxState OF
      0:
      G.BoxConv:=TRUE;
      IF G.BoxSensor=TRUE THEN
            G.BoxConv:=FALSE;
            BoxState:=1;
      END_IF
      1:
      G.BoxConv:=FALSE;
      IF RobotState = 4
      THEN
         BoxState:=0;
      END_IF;
     ELSE BoxState:=0;
END_CASE

CASE RobotState OF
      0: // Скинути робота в 0
      G.MoveX:=FALSE;
      G.MoveY:=FALSE;
      G.Grab:=FALSE;
      TimerOn:=FALSE;
      IF (boxstate =1) AND ((paletState = 3)OR(PaletState=1)) THEN
      robotstate:=1;
      END_IF;

      1: //Опустити руку
      G.MoveY:=TRUE;
      TimerOn:=TRUE;
      IF NOT(G.MovingZ) AND GTOn.Q  THEN
            RobotState:=2;
            TimerOn:=FALSE;
      END_IF
      2://захват
      G.Grab:=TRUE;
      TimerOn:=TRUE;
            IF GGrabOn.Q THEN
            RobotState:=3;
            TimerOn:=FALSE;
      END_IF
```

```
      3://рука вверх
      G.MoveY:=FALSE;
      TimerOn:=TRUE;
      IF NOT(G.MovingZ) AND GTon.Q THEN
            RobotState:=4;
            TimerOn:=FALSE;
      END_IF

      4://руку витягнути
      G.MoveX:=TRUE;
      TimerOn:=TRUE;
      IF (g.MovingX= FALSE) AND GTon.Q THEN
            RobotState:=5;
            TimerOn:=FALSE;
      END_IF

      5://руку вниз
      G.MoveY:=TRUE;
      TimerOn:=TRUE;
      IF (g.MovingZ= FALSE) AND GTon.Q THEN
            RobotState:=6;
            TimerOn:=FALSE;
      END_IF

      6://захоплючий пристій виключити
      G.Grab:=FALSE;
      TimerOn:=TRUE;
      IF Gton.Q THEN
        RobotState:=7;
        TimerOn:=FALSE;
      END_IF

      7://руку вверх
      G.Grab:=FALSE;
      g.MoveY:=FALSE;
      TimerOn:=TRUE;
      IF NOT(g.MovingZ) AND Gton.Q THEN
        RobotState:=7;
        TimerOn:=FALSE;
      END_IF

  8: //руку втянути
      g.MoveX:=FALSE;
      TimerOn:=TRUE;
        IF NOT(g.MovingX) AND Gton.Q THEN
           RobotState:=0;
           TimerOn:=FALSE;
        END_IF;
     ELSE RobotState:=0;

END_CASE
```

```
CASE PaletState OF
      0://Немає палет для завантаження
      G.StartConv:=TRUE;
      G.SensorConv:=TRUE;
      IF G.FirstSensor THEN PaletState:=1; END_IF;
      1://Палета на 1 позиції
      G.StartConv:=FALSE;
      G.SensorConv:=FALSE;
        IF RobotState=7 THEN PaletState := 2; END_IF;
      2://палета рухається на позицію № 2
      G.StartConv:=FALSE;
      G.SensorConv:=TRUE;
        IF G.SecondSensor THEN PaletState:=3; END_IF;
      3://палета на позиції № 2
      G.StartConv:=FALSE;
      G.SensorConv:=FALSE;
        IF RobotState=6 THEN PaletState := 4; END_IF;
      4://палета готова та переходить на позицію вивантаження
        G.SensorConv:=TRUE;
        IF NOT(G.SecondSensor) THEN PaletState:=0; END_IF;
     ELSE PaletState:=0;
END_CASE
```