Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Оптимізація технології програмування контролерів паралельної дії

(тема)

Виконав:

студент II курсу, групи СПм-20-1

Бура С. А.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма

Системне програмування
(повна назва освітньої програми)

Керівник: доц. Бовчалюк С. Я.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А. А.
(підпис) (прізвище, ініціали)

2021 р.

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 «Комп’ютерна інженерія»
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

Харківський національний університет радіоелектроніки

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Бурі Сергію Анатолійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Оптимізація технології програмування контролерів паралельної дії

затверджена наказом по університету від “ 5 ” листопада 2021 р. № 1657 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2021 р.

3. Вхідні дані до роботи 1) мова програмування ЯППЛК і ЯПЛК-М
2) TVP-технологія
3) технологія Rust
4) інструмент побудови графічних інтерфейсів iced
5) інструмент генерації таблиць xlsx

4. Перелік питань, що потрібно опрацювати у роботі
1) Аналіз сучасних методів та засобів керування промисловим технологічним обладнанням та технологій їх програмування
2) Аналіз технологій програмування технічних засобів керування паралельної дії
3) Огляд сучасних середовищ і технологій програмування та обґрунтування вибору засобів для створення технології програмування контролерів паралельної дії
4) Розробка архітектури програмного продукту та опис його програмної реалізації
5) Опис інтерфейсу користувача для заповнення циклограми мовою ЯПЛК-М та програмного продукту

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайд-презентація – 13 слайдів.

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз сучасних технологій програмування	09.11.21-12.11.21	
	ПЛК паралельної дії		
2	Вибір програмного забезпечення	13.11.21-16.11.21	
3	Аналіз та модифікація методів програмування	17.11.21-19.11.21	
	ПЛК паралельної дії		
4	Розробка програмного забезпечення	20.11.21-03.12.21	
5	Оформлення результатів кваліфаційної роботи	04.12.21-07.12.21	
6	Подання кваліфікаційної роботи керівникові	08.12.21-09.12.21	
	та її попередній захист		
7	Подання кваліфікаційної роботи на	10.12.21-11.12.21	
	рецензування		

Дата видачі завдання 08 листопада 2021 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Бовчалюк С. Я.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 77 с., 34 рис., 2 дод., 10 джерел.

ПРОГРАМОВАНІ ЛОГІЧНІ КОНТРОЛЕРИ, ППЛК, ПЛІС, HDL, ЯППЛК, ЯПЛК-М, TVP-ТЕХНОЛОГІЯ, ЦИКЛОГРАМА.

Метою кваліфікаційної роботи є реалізація графічного програмного додатку для створення та генерації циклограм мовою ЯПЛК-М для ПЛК паралельної дії.

У ході виконання кваліфікаційної роботи проведено аналіз стану сучасних методів керування та засобів програмування ПЛК паралельної дії, розглянуто технології програмування ППЛК. Виконано аналіз технологій програмування технічних засобів керування паралельної дії, та визначені їх основні недоліки. Розглянуто сучасні середовища і технології програмування для створення програмного забезпечення для ПК, та виконано обґрунтування вибору програмних засобів для створення технології програмування контролерів паралельної дії. Розроблено архітектуру програмного додатку та виконано його реалізацію. Наведено опис інтерфейсу користувача для заповнення циклограми мовою ЯПЛК-М та проведено тестування програмного продукту.

ABSTRACT

Explanatory note of the qualification work: 77 pp., 34 fig., 2 appendix, 10 sources.

PROGRAMMED LOGICAL CONTROLLERS, PLC OF PARALLEL ACTION, CPLD, HDL, ЯППЛК, ЯПЛК-М, TVP-TECHNOLOGY, CYCLOGRAM.

The purpose of the qualification work is the implementation of a graphical software application for the creation and generation of cyclograms in the language ЯПЛК-М for PLC parallel operation.

During the qualification work the analysis of the state of modern control methods and means of PLC programming of parallel action is carried out, the technologies of PLC of parallel action programming are considered. The analysis of technologies of programming of technical means of control of parallel action is executed, and their basic lacks are defined. The modern environments and programming technologies for the creation of software for the PC are considered, and the substantiation of the choice of software for the creation of the programming technology of the parallel controllers is performed. The architecture of the software application was developed and its implementation was performed. The description of the user interface for filling in the cyclogram in the ЯПЛК-М language is given and the software product is tested.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АСК ТП – автоматизована система керування технологічними процесами

ВІС – велика інтегральна схема

МЕК 61131-3 – міжнародний стандарт мов програмування програмованих логічних контролерів

ПЗ – програмне забезпечення

ПЛІС – програмована логічна інтегральна схема

ПЛК – програмований логічний контролер

ППЛК – паралельний програмований логічний контролер

ЯППЛК – мова програмування ПЛК паралельної дії

ЯПЛК-М – вдосконалена мова програмування ПЛК паралельної дії

DCS – розподілена система керування (англ., Distributed Control System)

EEPROM – постійна енергозалежна пам’ять(англ., Electrically Erasable Programmable Read-Only Memory)

EPROM – постійна пам’ять (англ., Erasable Programmable Read Only Memory)

FBD – графічна мова функціональних блоків (англ., Functional Block Diagram)

GUI – графічний інтерфейс користувача (англ., Grpahical User Interface)

HDL – мова опису обладнання (англ., Hardware Description Language)

IDE – інтегроване середовище розробки програмного забезпечення (англ., Integrated Drive Electronics)

IL – текстова мова низького рівня (англ., Instruction List)

IT – інформаційні технології (англ., Information Technology)

LD – мова сходових діаграм (англ., Ladder Diagram)

MOSFET – металооксидний напівпровідниковий польовий транзистор

(англ., metal-oxide-semiconductor field effect transistor)

PAL – програмований масив (англ., programmable array logic)

PLC – програмований логічний контролер (англ., Programmable Logic Controller)

PLD – програмована логічна інтегральна схема (англ., programmable logic device, PLD)

SCADA – диспетчерське управління і збір даних (англ., Supervisory Control And Data Acquisition)

SFC – графічна мова описання алгоритму управління у вигляді набору пар, крок і перехід (англ, Sequential Function Chart)

ST – текстова мова програмування ПЛК високого рівня (англ, Structured Text)

SRAM – оперативна пам’ять з довільним доступом (англ, static random access memor)

TVP – технологічне візуальне програмування (англ., Technological Visual Programing)

ВСТУП

Підхід до промислового виробництва змінювався протягом усієї історії людства, від механізованих конвеєрів до складних роботизованих систем. І на кожній стадії розвитку головною метою промислової галузі було підвищення продуктивності виробництва.

Сьогодні використання мікроелектронних засобів в системах автоматизованого керування призводить не тільки до підвищення техніко-економічних показників систем (вартість, надійність, габаритні розміри) і скорочення термінів розробки, а й надають їм принципово нові якості (розширені функціональні можливості, модифікованість, адаптивність тощо). За останні роки в мікроелектроніці бурхливий розвиток отримав напрямок, пов’язаний з випуском програмованих логічних контролерів (ПЛК).

Використання ПЛК значно підвищило продуктивність виробництва, але залишаються етапи, продуктивність котрих можливо підвищити ще більше, до таких етапів можна віднести методи програмування ПЛК [3].

Стандарт МЕК 6113-3 сприяв тому, що методи програмування ПЛК відрізняються між собою, оскільки окрім двох текстових мов, стандарт має ще три графічні мови. Звісно кожна мова використовується для різних задач, але на сьогодні промисловій галузі не потрібні всі п’ять мов стандарту МЕК 6113-3, оскільки більшість задач пов’язанні з виконанням відносно простих алгоритмів роботи умовного конвеєру.

Також окрім різних методів програмування ПЛК ще одним недоліком є сам процес програмування, який займає відносно багато часу через те що в процесі розробки беруть участь декількох спеціалістів які відповідають за різні етапи розробки. Інженер-технолог який відповідає за створення алгоритмів та залежностей між ними, та програміст який просто конвертує представлені йому алгоритми в код. Такий підхід розподілення роботи має свої переваги та недоліки на відміну від підходу коли весь обсяг праці

виконується одним працівником. Серед переваг можна відмітити відносне підвищення швидкості розробки, оскільки поки програміст буде зайнятий конвертацію алгоритму у код, інженер може перевіряти уже створенні програмістом програми якщо вони є, або ж створювати інші алгоритми для програміста, коли у випадку з підходом що один працівник займається розробкою алгоритмів та їх реалізацією. Але підхід розподіленої праці можна вдосконалити шляхом оптимізації програмування ПЛК, а саме пришвидшення конвертації алгоритму у програмний код.

На відміну від методів програмування які є стандартизованими, методи створення алгоритму роботи майбутньої програми не мають єдиного стандарту, що призводить до різних методів відображень майбутнього алгоритму. Зазвичай алгоритм роботи може бути представленим у вигляді електросхем. Мінусом такого підходу є те, що для розуміння алгоритму програміст повинен мати знання з електротехніки.

Опис алгоритму управління у вигляді таблиць циклограми є простим, наочним та доступним для читання інженерам-технологам. Таким чином, безпосередньо інженери-технологи на цій табличній мові створювали б програму управління, перевіряли створений алгоритм управління і усували логічні помилки в програмі. Але на етапі розробки циклограми зі збільшенням вхідних та вихідних змінних збільшується й розмір самої таблиці що підвищує шанси допустити помилку через велику кількість інформації яка відображається в таблиці.

Отже основною метою кваліфікаційної роботи є реалізація графічного програмного додатку для створення та генерації циклограм мовою ЯПЛК-М для ПЛК паралельної дії.

1 АНАЛІЗ СТАНУ ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз сучасних методів та засобів керування промисловим технологічним обладнанням

Головною складовою будь-якого виробництва є технологічний процес, а якщо точніше, то методи та засоби його керування. Всього можна визначити декілька систем керування технологічним процесом:

- ручне керування (всі функції управління виконує людина-оператор);
- автоматизоване керування (частину функцій виконує людина, а іншу частину – автоматичні пристрої);
- автоматичне керування (всі функції керування виконують автоматичні пристрої).

Серед усього списку найпоширенішою системою керування технологічним процесом є автоматизована система керування технологічними процесами (АСК ТП) яка представляється у вигляді сукупності апаратно-програмних засобів, що здійснюють контроль і керування виробничими й технологічними процесами, підтримуючи зворотній зв’язок і активно впливаючи на хід процесу при відхиленні його від заданих параметрів; забезпечуючи регулювання й оптимізацію керованого процесу. АСК ТП ділиться на три глобальні класи:

SCADA (англ. Supervisory Control And Data Acquisition) – диспетчерське управління і збір даних. Основне призначення системи – контроль і моніторинг об’єктів за участю диспетчера. У вузькому сенсі під терміном «SCADA» розуміють програмний пакет візуалізації технологічного процесу. У широкому розумінні – це клас автоматизованих систем керування технологічним процесом (рисунок 1.1).

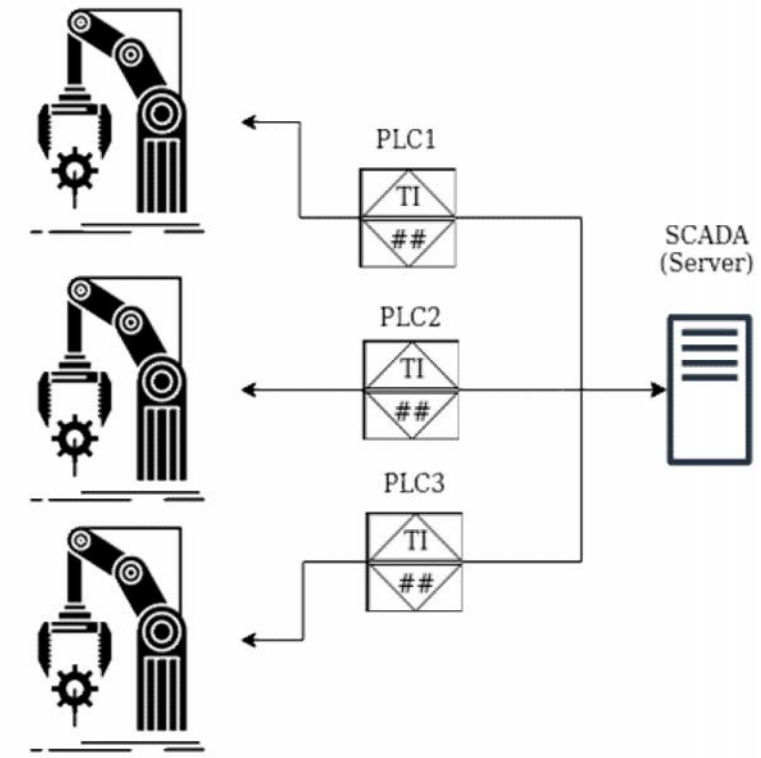


Рисунок 1.1 – Архітектура SCADA

DCS (англ. Distributed Control System) – розподілена система керування (РСК). Це система керування технологічним процесом, що характеризується побудовою розподіленої системи вводу/виводу та децентралізацією обробки даних. РСК застосовуються переважно для керування неперервними і гібридними технологічними процесами. У першу чергу це стосується процесів, що тривають місяцями і навіть роками, при цьому зупинка процесу, навіть на короткочасний період, може привести до псування продукції, що виготовляється, поломки технологічного устаткування чи нещасних випадків.

PLC (англ. Programmable Logic Controller) – програмований логічний контролер (ПЛК). У вузькому розумінні це – апаратний модуль для реалізації алгоритмів автоматизованого керування з використанням логічних операцій, таймерів, і (в деяких моделях) неперервне регулювання відповідно до заданого закону. У широкому розумінні під ПЛК розуміється клас систем. Хоча ПЛК може управляти компонентами системи, що використовуються в

SCADA і DCS систем, вони часто є основним компонентом у структурах невеликих систем керування у багатьох галузях виробництва.

Багато в чому, архітектура програмованих логічних контролерів (ПЛК) нагадує архітектуру ПК загального призначення зі спеціалізованими модулями вводу/виводу (I/O). Однак, деякі важливі характеристики відрізняють ці пристрої. По-перше, ПЛК набагато надійніші та розраховані на безвідмовну роботу протягом багатьох років – і це їхня найважливіша особливість. По-друге, ПЛК можуть використовуватися в умовах промислового виробництва, де їм доводиться працювати в умовах серйозного електромагнітного випромінювання, вібрації, екстремальних температур та вологості. По-третє, ПЛК легко обслуговуються технічним персоналом з виробництва. Оскільки ПЛК – це промисловий контролер на основі мікропроцесора з програмованою пам’яттю, що використовується для зберігання програмних інструкцій та різних функцій то він складається з наступних складових частин (рисунок 1.2):

- процесор (CPU), який інтерпретує вхідні дані, виконує керуючу програму, збережену в пам’яті, і посилає вихідні сигнали;
- блок живлення, який перетворює змінну напругу в постійний;
- блок пам’яті, що зберігає дані з входів і програму;
- інтерфейс входу та виходу, де контролер отримує та відправляє дані з/на зовнішні пристрої;
- комунікаційний інтерфейс для отримання та передачі даних у мережах зв’язку від/до віддалених ПЛК.

Процесорний модуль складається з власне процесора (CPU) та пам’яті. Крім того, є інтерфейс до програматора, а також можуть бути інтерфейси до віддалених пристроїв вводу/виводу та комунікаційних мереж. Живлення зазвичай реалізується за допомогою окремого модуля, як і введення/виведення.

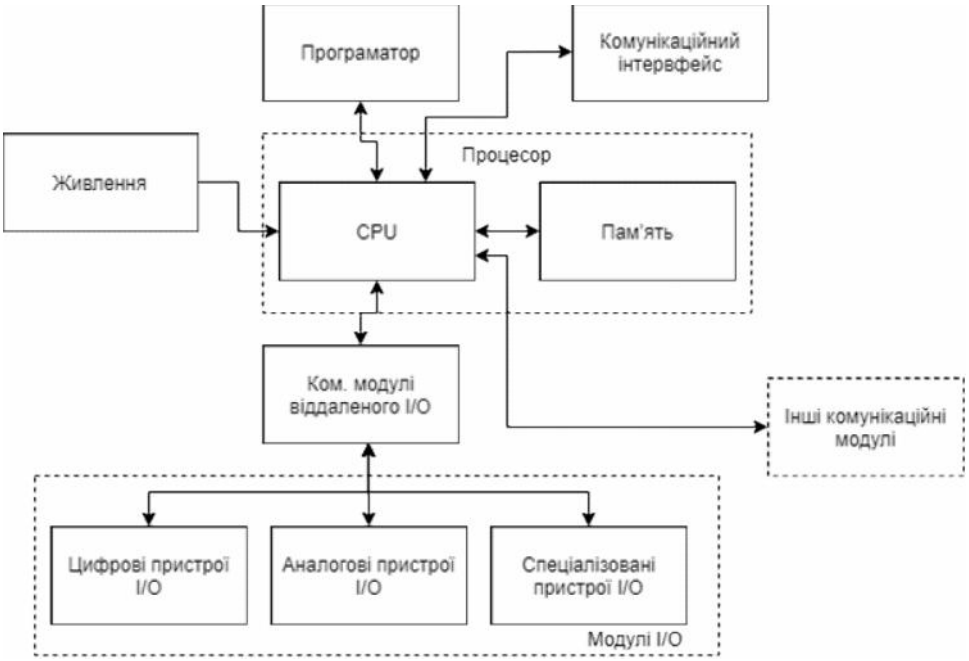


Рисунок 1.2 – Архітектура ПЛК

Модулі введення/виводу бувають дискретними (вкл./викл.), і аналоговими, а також існують спеціалізовані модулі, наприклад, для керування переміщеннями або швидкісні лічильники. До модулів введення/виводу підключаються польові пристрої. Залежно від кількості каналів введення/виводу, і конкретного процесора ПЛК, модулі введення/виводу можуть бути на тому ж шина, що і процесор, та/або на іншому або навіть кількох шинах.

Принцип роботи ПЛК є дуже простим, коротко кажучи це циклічна програма зчитування входів та встановлення виходів. На початку кожного циклу програма зчитує стани входів контролера та записує їх (таблиця стану входів процесу). Після виконання всіх команд і визначення (підрахунку) актуального для даної ситуації стану виходів, контролер вписує стани виходів до пам’яті, що є таблицею стану виходів процесу а операційна система виставляє відповідні сигнали на виходи, котрі керують виконавчими механізмами. Отже всі сигнальні комбінації подаються у вхідний модуль контролера, а програма слідкує за їх станами та реагує змінами виходів на основі закладеного алгоритму (рисунок 1.3).

Рисунок 1.3 – Алгоритм роботи ПЛК

Використання зазначених технологій значно підвищує продуктивність виробництва, та сприяє загальному технологічному розвитку шляхом удосконалення методів керування промисловим виробництвом, що викликає потребу в оновленні підходу до програмування ПЛК.

1.2 Аналіз технологій програмування технічних засобів керування промисловим технологічним обладнанням

Безперервний ріст рівня автоматизації систем керування процесами й об’єктами в найрізноманітніших областях економіки стимулює потребу ринку праці у кваліфікованих фахівцях, що забезпечують проектування, конструювання, експлуатацію і ремонт сучасних засобів і систем автоматизації.

Для стандартизації великої кількості варіантів мов у 1993 році міжнародна електротехнічна комісія випустила стандарт МЕК 61131-3, який

описує мовні засоби програмування промислових контролерів. У стандарт увійшли три графічні та дві текстові мови:

- SFC (Sequential Function Chart) – графічна мова, що дозволяє описати алгоритм управління у вигляді набору пар крок і перехід;
- LD (Ladder Diagram) – мова сходових діаграм, графічна мова побудований на принципах релейно-контактних схем;
- FBD (Functional Block Diagram) – графічна мова функціональних блоків;
- ST (Structured Text) – текстова мова високого рівня з паскалеподібним синтаксисом;
- IL (Instruction List) – текстова мова низького рівня, що представляє собою спрощений асемблер для завдань логічного управління.

При створенні стандарту розробники описали на той момент популярні мови програмування ПЛК [6]. Під час створення стандарту було зрозумілим, що мови програмування програмованих контролерів повинні бути простими й наочними, для того, щоб самі інженери-технологи могли створювати керуючі програми. Проте зараз створенням програм управління для ПЛК займаються професійні програмісти.

Професійні програмісти під час створення програм в більшості випадків використовують приватні підходи до написання коду, що залежать від суб’єктивного досвіду та переваг конкретного розробника. Розробка систем логічного управління передбачає проблемно-орієнтований підхід з тісною участю на всіх етапах життєвого циклу створюваного програмного забезпечення інженерів-технологів, які є експертами в галузі автоматизованого процесу.

Сьогодні найпоширенішим мовним засобом створення програмного забезпечення систем управління є набір мов МЕК 61131-3. Кожна з п’яти мов стандарту орієнтована одну конкретну область завдань логічного управління. При описі складних алгоритмів управління програмісти змушені в одному проекті використовувати кілька мов стандарту МЕК 61131-3. У такій

програмі частина алгоритму описується однією мовою програмування, а решта програми описується іншою мовою програмування. Багато фахівців вказують на явні недоліки використання мов стандарту, «закритість» для розуміння та використання цих мов для інженерів-технологів. Тому з’являються різні роботи, що описують альтернативні стандарту мови програмування ПЛК. Описуються ідеї використання універсальної візуальної мови моделювання UML для створення програм логічного управління. Іншим напрямом, є адаптація універсальних текстових мов програмування до завдань логічного управління. Однак використання універсальних мов програмування та моделювання доступне тільки професійним програмістам.

Також пропонується спосіб автоматичної генерації програмного коду на стандартних мовах програмування ПЛК за математичними моделями, наприклад з математичного пакету MathCAD. Пропонуються різні спеціалізовані мови моделювання для вирішення певного класу завдань автоматизації. Проте програмна помилка в логіці управління складним технологічним об’єктом може призвести не тільки до серйозних матеріальних втрат, але створити аварійну ситуацію, що загрожує здоров’ю і життю людей.

З появи перших промислових контролерів різні фахівці вказували необхідність створення наочних проблемно-орієнтованих мов програмування. Ці мови повинні бути простими та повинні дозволяти описати алгоритм логічного управління тільки термінами технологічного процесу. Такий підхід дозволить інженерам-технологам самим створювати та редагувати програми управління не звертаючись за допомогою до програмістів тим самим пришвидшуючи етап розробки завдяки чому зростає й загальна продуктивність.

1.3 Аналіз технологій програмування технічних засобів керування паралельної дії

ПЛК паралельної дії, або паралельні ПЛК (ППЛК) є вдосконаленням

звичайних ПЛК. Річ у тім що при аналізі структурної організації та елементної бази відомих ПЛК показує, що при їх побудові функціональні можливості сучасних програмованих ВІС з однорідною (матричною) структурою використовуються не повністю. Крім того, для програмування ПЛК практично не застосовуються природні для користувача промислового обладнання технологічні мови опису алгоритмів керування в їх явному вигляді (таблиці рішень, технологічні такто- і циклограми та т.п.), які також характеризуються високим рівнем регулярності.

Тобто, у методології і практиці побудови ПЛК явно не використовуються властивості регулярності, притаманні як однорідним мікроелектронним структурам, так і технологічним мовам опису алгоритмів керування об’єктами дискретної циклічної дії.

ППЛК складається з наступних блоків (рисунок 1.4):

- блоку індикації – БІ;
- схеми порівняння – СП;
- блоку вибору операції – БВО;
- блоку логічного керування – БЛК;
- лічильника адреси – ЛкА;
- вихідного регістра – ВР;
- блоків пам’яті станів, команд, переходів і заборонених комбінацій – БПС, БПК, БПП, БПЗК.

ППЛК працює наступним чином. Встановлення його у початковий стан здійснюється за допомогою зовнішнього імпульсного сигналу початкової установки "ПУ", що встановлює в 0 лічильник адреси. Процес відпрацювання керуючої програми складається з двох етапів:

- аналіз комбінацій станів датчиків умов переходів і формування початкової адреси підпрограми;
- власне відпрацювання обраної підпрограми. Причому аналіз станів зовнішнього середовища здійснюється паралельно і незалежно від відпрацювання підпрограми.

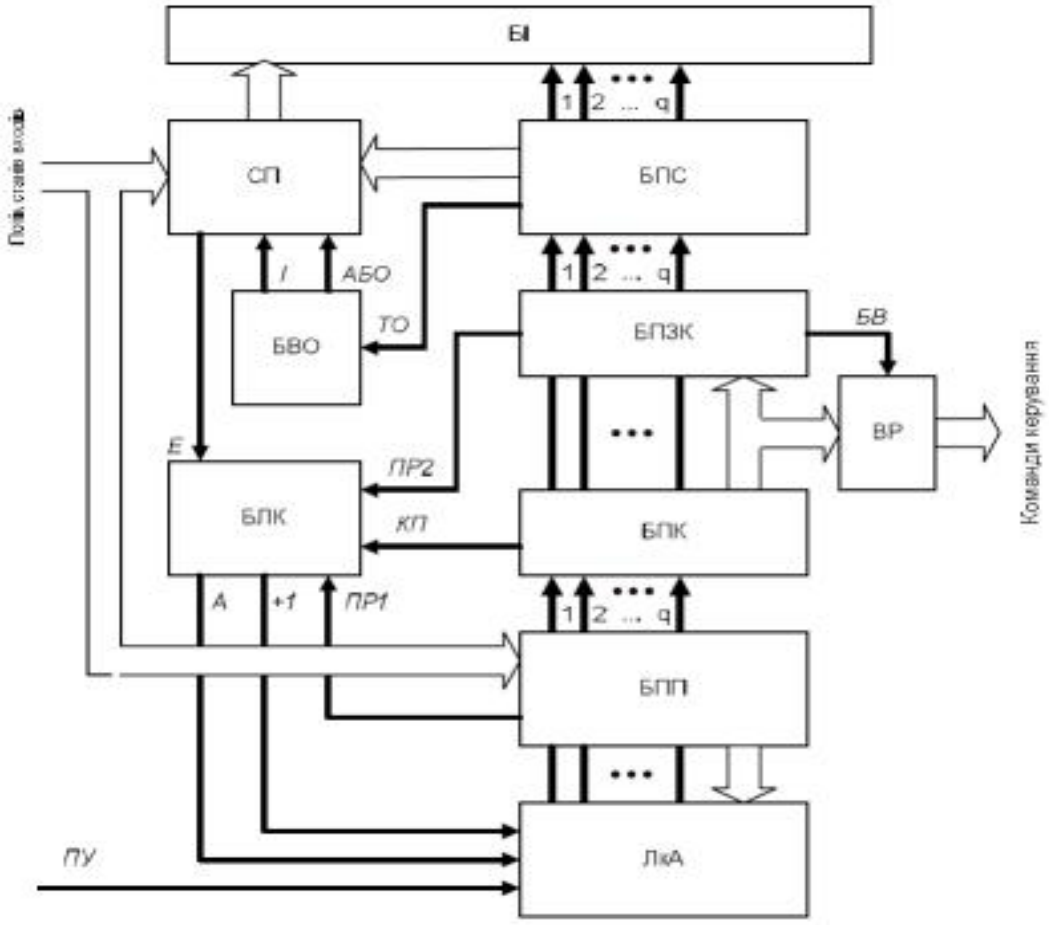


Рисунок 1.4 – Архітектура ППЛК

Основою ППЛК є кристал ПЛІС (рисунок 1.5), який використовується для побудови реконфігурованих цифрових схем. На відміну від інтегральних схем (ІС), які складаються з логічних елементів і мають фіксовану функцію, ПЛІС має невизначену функцію на момент виготовлення. Перш ніж ПЛІС можна буде використовувати в ланцюзі, його необхідно переконфігурувати [2].

ПЛІС є комбінацією логічного пристрою та пристрою пам’яті. Пам’ять використовується для зберігання шаблону, який був наданий мікросхемі під час програмування. Більшість методів зберігання даних в інтегральній схемі були адаптовані для використання в ПЛІС. До них належать:

- кремнієві антизапобіжники;

- SRAM;
- комірки пам’яті EPROM або EEPROM;
- флеш-пам’ять.

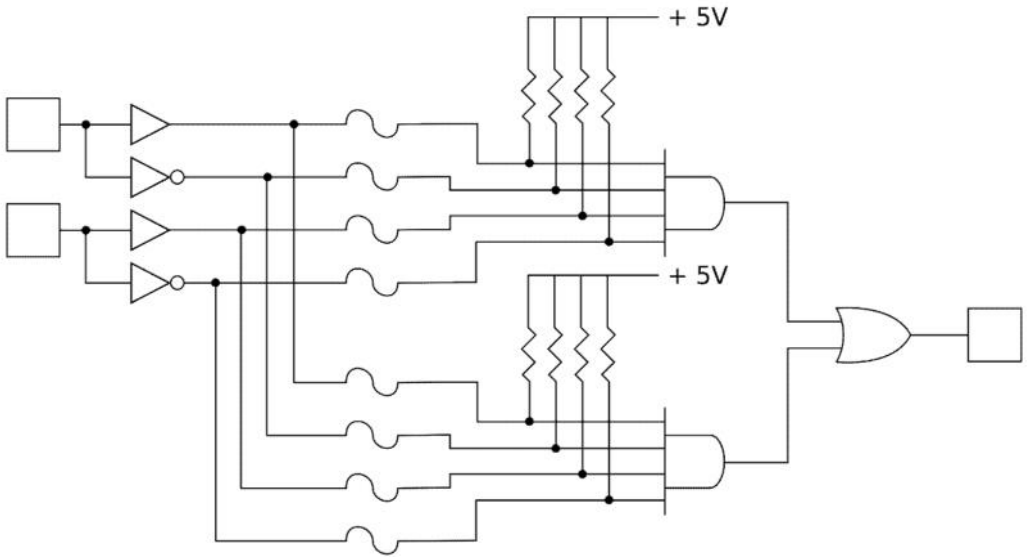


Рисунок 1.5 – Спрощена структура базового елементу ПЛІС

Кремнієві антизапобіжники – це з’єднання, які створюються шляхом подачі напруги на змінену ділянку кремнію всередині мікросхеми. Вони називаються анти-запобіжниками, тому що вони працюють протилежно звичайним запобіжникам, які починають функціонувати як з’єднання, поки їх не розриває електричний струм.

SRAM, або статична оперативна пам’ять, є енергонезалежним типом пам’яті, що означає, що її вміст втрачається при кожному вимкненні живлення. Тому PLD на основі SRAM необхідно програмувати щоразу, коли ланцюг вмикається. Зазвичай це робиться автоматично іншою частиною схеми.

Комірка пам’яті EPROM – це MOSFET (металооксидний напівпровідниковий польовий транзистор або МОП-транзистор), який можна увімкнути шляхом постійного уловлювання електричного заряду на його електроді затвора. Це робить програміст PAL. Заряд залишається протягом

багатьох років, і його можна видалити, лише піддавши чіп впливу сильного ультрафіолетового світла в пристрої, який називається ластик EPROM.

Флеш-пам’ять є енергонезалежною, зберігаючи свій вміст навіть при вимкненому живленні. Він зберігається в осередках пам’яті MOSFET з плаваючим затвором і може бути стертий і перепрограмований за потреби. Це робить його корисним у PLD, які можна часто перепрограмувати, наприклад, PLD, що використовуються в прототипах. Флеш-пам’ять – це різновид EEPROM, який зберігає інформацію за допомогою захоплених електричних зарядів, подібних до EPROM. Отже, флеш-пам’ять може зберігати інформацію роками, але, можливо, не стільки років, скільки EPROM.

Використання ПЛІС контролерів в концепції ПЛК дозволяє розширити можливості керуючого контролера, а саме розпаралелити виконання програм автоматизацій, оскільки ПЛІС може виконувати операції паралельно: зчитувати вхідні сигнали та встановлювати вихідні.

Методи програмування ПЛК паралельної дії значно відрізняються від методів звичайних ПЛК, через те що ППЛК базується на ПЛІС контролері, то мови зі стандарту МЕК 61131-3 не підходять для програмування даного контролеру. Оскільки на відміну від звичайних контролерів ПЛІС контролера надає можливість змінювати цифрову схему контролера. У таких випадках слід використовувати так звані мови опису обладнання (HDL).

HDL використовуються як для моделювання, так і для синтезу. Логічне моделювання – це потужний спосіб перевірити систему на комп’ютері, перш ніж її перетворити на апаратне забезпечення. Симулятори дозволяють перевірити значення сигналів у вашій системі, які неможливо виміряти на фізичному пристрої. Логічний синтез перетворює код HDL в цифрові логічні схеми.

VHDL і Verilog вважаються звичайними мовами розробки цифрових пристроїв, тоді як SystemVerilog представляє розширену версію Verilog. Кожна мова має свій стиль та індивідуальні характеристики. VHDL

успадкував від мови програмування Ada концепцію та синтаксис, у той час як Verilog посилається на ранній HDL, який називався Hilo, та має стиль мови програмування C.

У Verilog та VHDL інженери можуть уявити необхідну функціональність у вигляді тексту програми. Потім модель симулюється для перевірки, що дизайн працює так, як потрібно. У моделі можуть бути підкориговані будь-які проблеми і симуляція перевірить цю корекцію. VHDL це багата і строго типізована і строго детерміністична мова, більш детальна порівняно з Verilog. В результаті проекти, написані на VHDL, вважаються само документованими. Синтаксис дуже відрізняється від стилю мови C, і інженери, що працюють у VHDL, постійно стикаються з необхідністю явного перетворення з одного типу даних на інший. VHDL часто показує помилки, які пропускає Verilog. VHDL має підкреслено однозначно недвозначну семантику, тому легше переноситься між різними системами розробки (у тому сенсі, що перенесення точніше переносить всі тонкощі роботи вихідного проекту).

Також для спрощення процесу розробки існують середовища програмування ПЛІС контролерів, які окрім аналізатору коду мають додатковий функціонал. Наприклад середовище розробки Active-HDL дозволяє окрім можливості редакції коду мовами VHDL, Verilog, System Verilog та SytemC й аналізу коду на помилки (рисунок 1.6), чого вже достатньо для комфортного користування даним середовищем.

Окрім описаного вище функціоналу, середовище розробки Active-HDL дозволяє моделювати та верифікувати проект для програмованих логічних інтегральних схем (рисунок 1.7).

Додатково цьому середовище має потужний симулятор змішаної мови загального ядра (рисунок 1.8), який підтримує VHDL, Verilog, SystemVerilog і SystemC. Що дозволить дуже швидко перевірити щойно розроблену програму без необхідності в перепрограмуванні фізичного контролеру бо у випадку допущення критичної помилки при розробці програми шанс

зруйнувати контролер підвищується. Емулятор в свою чергу надає можливість виявити проблеми які можуть статися з фізичним об’єктом без додаткових ризків.

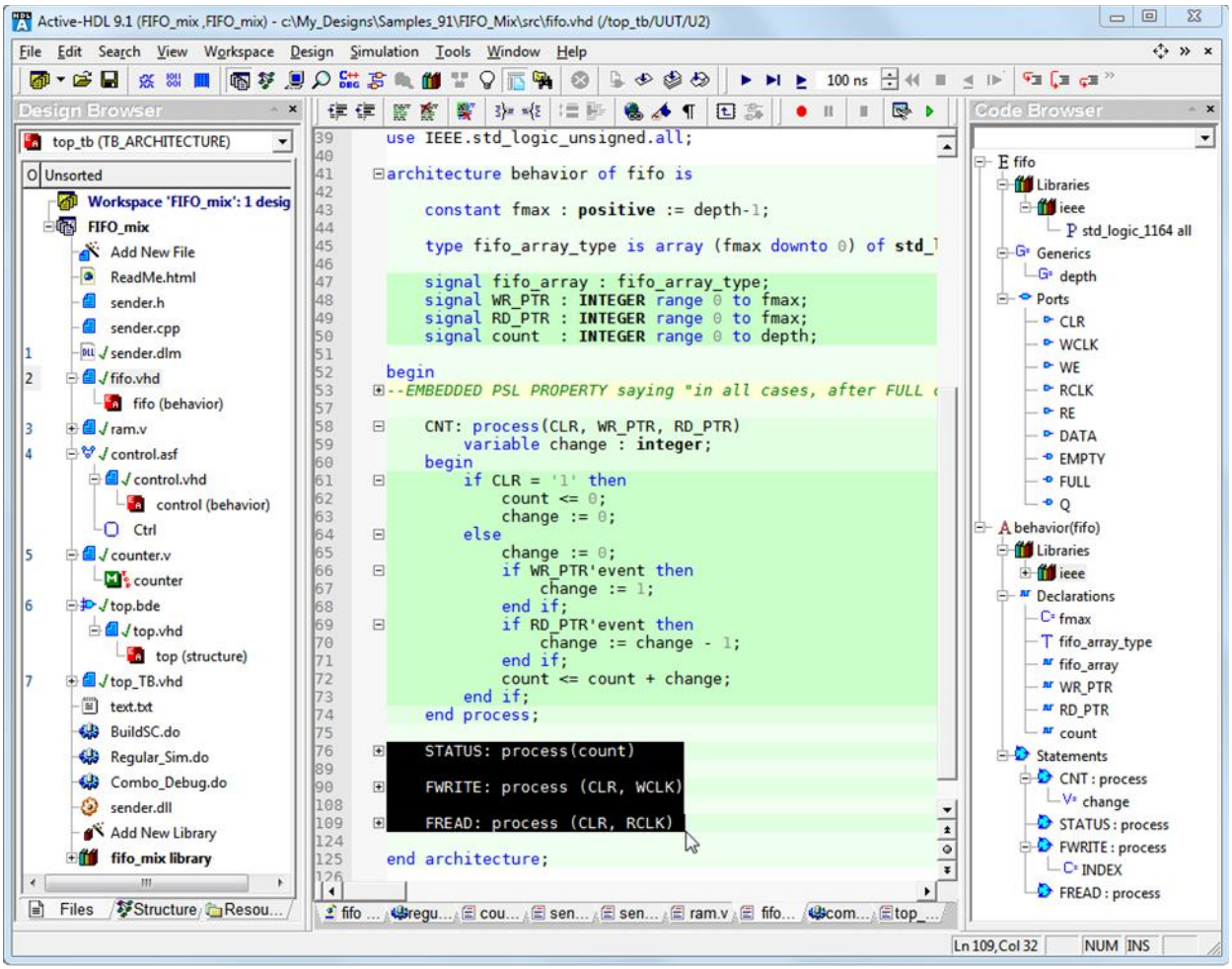


Рисунок 1.6 – Редактор коду HDL

До того ж, можливість використання перевірки на основі показників, задля визначення невикористаних частин створеного дизайну за допомогою інструментів аналізу покриття коду. Моделювання розширених конструкцій перевірки, таких як SV Functional Coverage, обмежена рандомізація та UVM. Інтелект абстрактного дизайну та представлення його у зручній для розуміння графічній формі за допомогою конвертера HDL у схему.

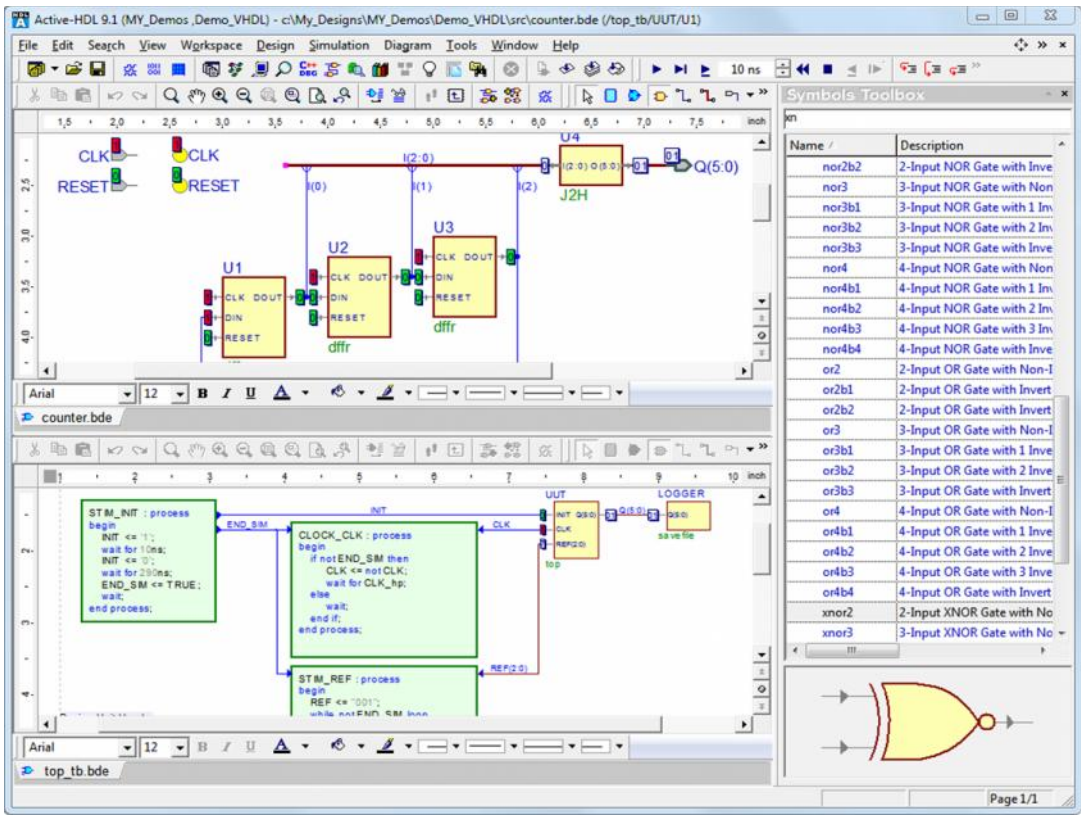


Рисунок 1.7 – Редактор інтегральних схем

Отже, Active-HDL – це потужний інструмент для програмування ПЛІС контролерів і використання його при програмування ПЛК паралельної дії є доречним та правильним рішенням, але в ньому не має можливості у моделюванні алгоритму роботи контролеру не тільки за допомогою логічних інтегральних схем, а й моделювання алгоритму роботи мовою ЯПЛК-М.

У своїй суті мова програмування ЯПЛК є використанням циклограми у табличному вигляді як алгоритмічної мови, де рядок таблиці відповідає робочому кроці циклу, а в кожному рядку записується сукупність (вектор) команд управління виконавчими механізмами та комбінація станів, у які повинні прийти датчики при спрацьовуванні механізмів (рисунок 1.9).

Такий підхід значно покращує візуальне сприйняття логіки роботи керуючого пристрою, та дозволяє знаходити помилки в алгоритмі ще до його реалізації мовами програмування та емуляції його роботи в середовищі Active-HDL. Також дане відображення логіки роботи програми є більш

зрозумілим та простим для програміста який може не мати достатньо кваліфікаційного досвіду для розуміння логіки роботи ПЛК у вигляді інтегральних схем (рисунок 1.7).

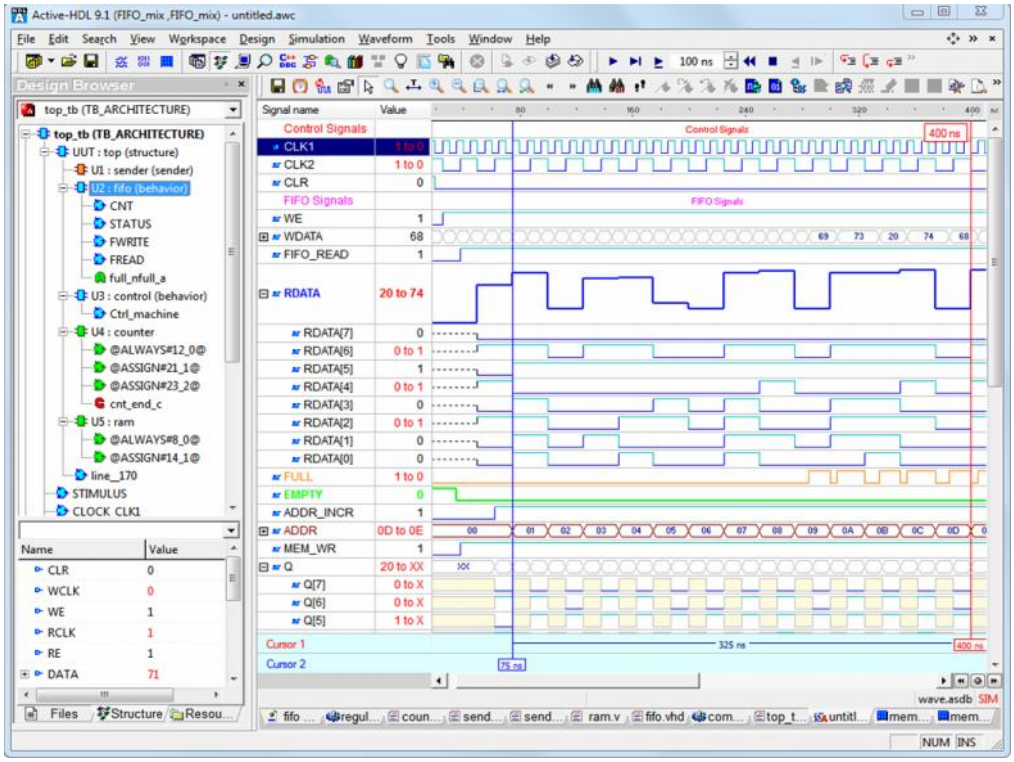


Рисунок 1.8 – Емулятор

Рисунок 1.9 – Мова програмування ЯПЛК

Концепція мови ЯППЛК являє собою виконану за визначеними правилами формалізований (табличний) запис технологічної циклограми. Циклограма має змінні які поділяються на вхідні, вихідні та внутрішні. Усі вхідні і вихідні змінні є булевими векторами; вхідними змінними є стани входів ПЛК (стани керованого об’єкта і зовнішнього середовища (ЗС)), вихідними – стани виходів ПЛК (команди керування), внутрішніми – адреси переходів.

Програма складається з двох основних частин, а саме описової до якої відносяться прив’язка елементів програми до об’єкта керування й операторної яка відображається у вигляді формалізованого запису циклограми роботи технологічного агрегату.

Кожен рядок операторної частини є етапом циклограми, кожен з яких виступає простим оператором. Набір з логічно зв’язаних рядків, що має самостійне значення, являє собою складений оператор. Рядок операторної частини програми (оператор) складається з номера рядка і 4-х кадрів:

- стану (містить слова, що описують стани датчиків циклу (стан керованого об’єкта));
- умови (складається зі слів, що описують стани датчиків умов переходів (стан ЗС));
- керування (складається зі слів, що описують команд керування);
- адреси переходу (являє собою код переходу складається з двійкових цифр і являє собою код початкової адреси і-го складеного оператора).

Слова у кадрах стану, умови і керування використовуються для програмування станів датчиків циклу, датчиків умов переходів і команд керування, причому змістовне значення слів визначається складом і послідовністю (зліва направо) розташування в них символів "1" і "0". Значення слів у кадрах стану й умови:

"10" – активний стан входу (включений стан датчика);

"01" – неактивний стан входу (відключений стан датчика);

"00" – стан входу (сенсора) несуттєвий для даного рядка циклу;

"11" – використовується для "викреслювання" рядка.

У свою чергу слова у кадрах керування концептуально є схожими зі словами стану та умови, але все ж є і відмінності:

"10" – збудження виходу (команда включення механизма);

"01" – дезбуждення виходу (команда виключення механізму);

"00" – керуючий вплив не змінюється;

"11" – заборонена комбінація (невизначеність).

На відміну від операторної, описова частина програми являє собою послідовність рядків, що описують назви датчиків станів, умов, команд керування і розряди адрес переходів. Крім того, на початку опису вводиться додатковий рядок "номер рядка" для позначення номера оператора або кроку циклу (рисунок 1.10).

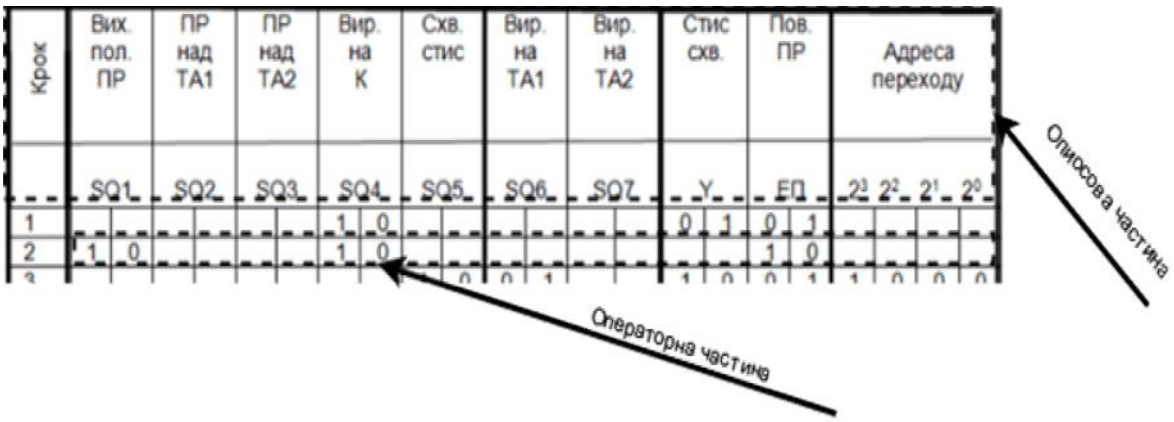


Рисунок 1.10 – Фрагмент програми мовою ЯППЛК

Складання програми мовою ЯППЛК зводиться до концептуально зрозумілого алгоритму: для кожної вхідної і вихідної змінної на кожному рядку програми вказуються стани і дії змінних, котрими ставляться у відповідність стани датчиків (включено, виключено, несуттєво і заборонено для даного рядка) і команд керування (уключити, виключити, не впливати).

Для програмування ПЛІС контролерів паралельної дії є вдосконалена версія мови ЯППЛК – ЯПЛК-М яка має дещо схожу конструкцію з її попередньою версією. Оскільки ПЛІС є контролером паралельної дії для його програмування за технологією TVP, то однієї таблиці для коректного програмування контролера буде недостатньо. Тому керуюча програма являє собою два виконані за певними правилами формалізовані (табличні) записи [8].

Перша таблиця (рисунок 1.11) складається із двох частин: ліва описує можливі стани керуємого об’єкта і зовнішнього середовища, а також заборонені комбінації керуючих сигналів контролера; і права, що однозначно визначає початкові адреси керуючих і аварійних підпрограм, а також ознаки переривань.

Таблиця 1

Найменування комбінацій станів датчиків і команд керування, що визначають початкові адреси технологічних і аварійних підпрограм, а також заборонені комбінації керуючих сигналів		Стани датчиків (стохастичних входів)																				Команди керування								Ознаки переходу ПР1	Адреси переходів	Сформ. заборонені комбінації ПР2
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	...	n	1	2	3	4	5	6	...	m			
Підпрограма 1: переміщення з Р1-1 (ЗС14)	на Зав2-1(ЗС14)	1		0		1																								С	7	0
Підпрограма 2: переміщення з Р1-1 (ЗС15)	на Зав2-2(ЗС15)	1		0			1																							С	18	0
Підпрограма 3: переміщення з Р1-1 (ЗС16)	на Зав2-3(ЗС16)	1		0				1													1									С	28	0
Підпрограма 4: переміщення з Р1-2 (ЗС12)	на Зав2-1(ЗС14)	1				1	1																							С	41	0
Підпрограма 5: переміщення з Р1-2 (ЗС13)	на Зав2-2(ЗС15)	1				0		1																						С	51	0
Підпрограма 6: переміщення з Р1-2 (ЗС12)	на Зав2-3(ЗС16)	1				0			1												1									С	62	0
Підпрограма 7: повернення ПР у вихідне положення	з Р1-1	1							1	0																				С	.	0
Підпрограма 8: повернення ПР у вихідне положення	з Р1-2	1								0																				С	1	0
Підпрограма 9: повернення ПР у вихідне положення	зі Зав2-1	1									0																			С	3	0
Підпрограма 10: повернення ПР у вихідне положення	зі Зав2-2	1										0																		С	3	0
Підпрограма 11: повернення ПР у вихідне положення	зі Зав2-3	1											0																	С	3	0
Аварійна ситуація 1: скат у низу, ПР рухається вправо		1													1	1														1	73	0
Аварійна ситуація 2: скат у низу, ПР рухається вліво		1													1			1												1	75	0
Аварійна ситуація 3: зустрічний рух ПР: П1-п1		1														1	1													1	72	0
Аварійна ситуація 4: зустрічний рух ПР: ПЗп1		1																1	1											1	73	0
Аварійна ситуація 5: аварійна зупинка обладнання		0	1																											1	72	0
Заборонена комбінація 1: одночасна видача команд на вмикання приводу П.' вправо і вліво																						1	1							С		1
Заборонена комбінація 2: одночасна видача команд на вмикання приводу підняття : опускання скату																								1	1					С		1
Заборонена комбінація 3: одночасна видача команд на вмикання стискання : розтискання скату																										1	1			С		1

Рисунок 1.11 – Перша таблиця мови ЯПЛК-М

Друга (рисунок 1.12) в свою чергу є сукупністю технологічних і аварійних підпрограм, що підлягають реалізації у залежності від поточних значень змінних першої таблиці. Таблиця 2 також складається із двох частин:

у лівій записуються очікувані стани сигналів від датчиків детермінованих входів і тип логічної операції, виконуваної над зазначеними сигналами; у правій – вказуються команди керування технологічним обладнанням.

Використовувані у мові змінні поділяються на вхідні, вихідні і внутрішні. Вхідними змінними є стани входів контролера, вихідними – стани виходів контролера, внутрішніми – адреси переходів, ознаки переривань ПР1 і ПР2, а також адреса і тип логічної операції [7].

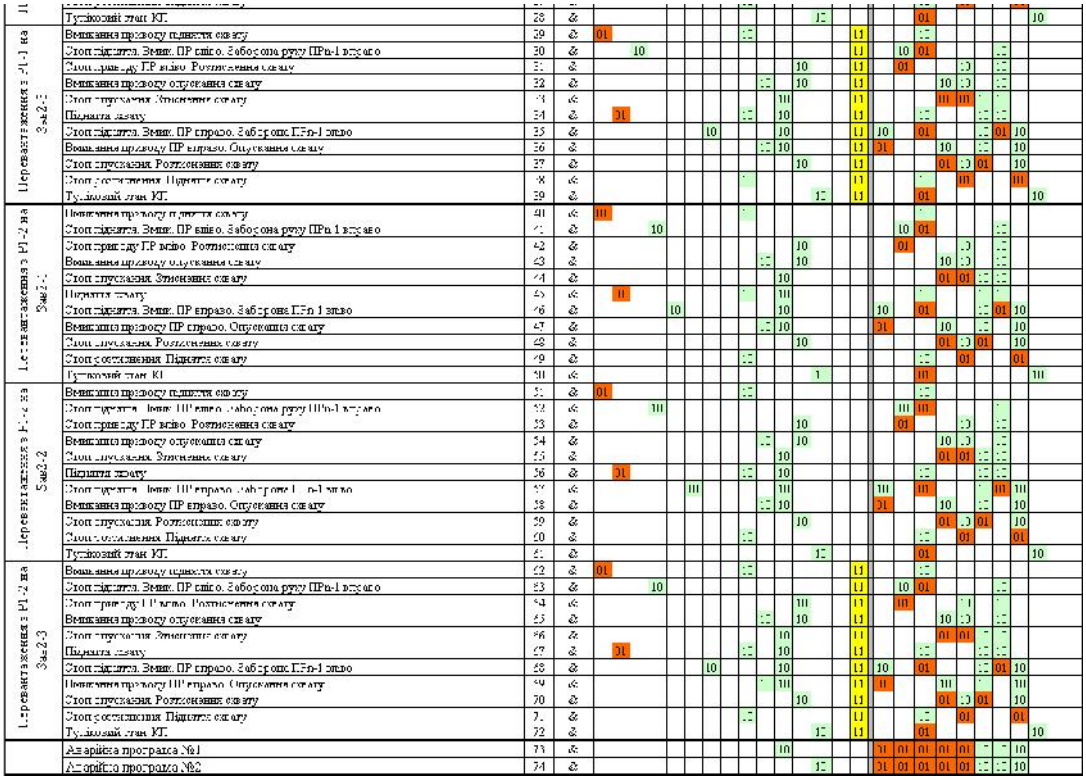


Рисунок 1.12 – Друга таблиця мови ЯПЛК-М

Таблиці в ЯПЛК-М, як і в ранній версії складаються з описової та операторної частин.

Описова частина програми першої таблиці виконує прив’язку елементів програми до об’єкту керування, а також описує комбінації станів датчиків і команд керування, що визначають адреси технологічних й аварійних підпрограм і заборонених комбінацій керуючих сигналів. В свою чергу операторна частина виконує запис станів датчиків стохастичних входів і

команд керування, що призводять до формування внутрішніх змінних;

Друга таблиця має суміжну описову частини програми. Так само як і в першій таблиці, у другій здійснюється прив’язка елементів програми до об’єкта керування, також на відміну від попередньої таблиця 2 приводить найменування технологічних і аварійних підпрограм. Операторній частина таблиці 2 виконує формалізований запис циклограми роботи технологічного обладнання.

Операторна частина таблиці 1 в мові ЯПЛК-М подібно операторній частині в мові ЯППЛК складається з рядків за числом підпрограм і заборонених комбінацій керуючих сигналів. Рядок операторної частини програми таблиці 1 складається з п’яти кадрів: умови, керування, ознаки переходу ПР1, адреси переходу, ознаки забороненої комбінації ПР2 і визначає початкові адреси технологічних і аварійних підпрограм, а також умови виникнення заборонених комбінацій вихідних сигналів.

Таблиця 2 має дещо іншу концепцію операторної частини програми та собою послідовність із рядків, кожний з яких відповідає певному етапу виконання технологічної операції або аварійної підпрограми і являє собою простий оператор. Рядок операторної частини програми таблиці 2 складається з номера рядка і чотирьох кадрів – адреси, типу логічної операції, стану й керування і реалізує виконання технологічних операцій або аварійних підпрограм.

Основна відмінність мов ЯПЛК-М та ЯППЛК є не тільки в кількості таблиць, але й у кількості та призначенні кадрів операторної частини. Також кадри відрізняються між першою та другою таблицями.

Кадр умови у таблиці 1 складається зі слів, що описують стани датчиків умов переходів.

Кадр забороненої комбінації у таблиці 1 складається зі слів, що визначають комбінацію команд керування, поява якої на виході контролера може призвести до аварійної ситуації.

Кадр ознаки переходу ПР1 у таблиці 1 являє собою символ, що

позначає можливість завершення виконання поточної підпрограми при відсутності ознаки аварійного переривання, або безумовний перехід до відпрацьовування аварійної підпрограми, при його наявності.

Кадр адреси переходу в таблиці 1 являє собою десяткове число, що визначає початкову адресу і-го мікроциклу (технологічної або аварійної підпрограми) таблиці 2.

Кадр ознаки забороненої комбінації ПР2 у таблиці 1 являє собою символ, що визначає наявність забороненої комбінації вихідних сигналів.

Кадр адреси у таблиці 2 являє собою десяткове число, що позначає поточну адресу виконуємої технологічної операції або поточного етапу виконуємої аварійної підпрограми.

Кадр типу логічної операції в таблиці 2 позначає логічну операцію ("І" чи "АБО"), виконувану над вхідними змінними.

Кадр стану в таблиці 2 містить k слів, які описують стан k датчиків детермінованих входів.

Кадр керування в таблиці 2 складається з m слів, які описують m команд керування технологічним обладнанням.

В описовій частині програми пояснюються природною мовою назви датчиків стохастичних входів і команд керування, для яких можливі заборонені їхні комбінації; а також назви датчиків детермінованих входів і команд керування технологічним обладнанням. Крім того, таблиця 1 містить опис природною мовою назв підпрограм, а також заборонених комбінацій вихідних сигналів. Аналогічно таблиця 2 містить опис технологічної операції або етапу аварійної підпрограми, виконуємої на даному етапі.

Складання програми мовою ЯПЛК-М полягає у наступному – для кожної вхідної, вихідної і внутрішньої змінних на кожному рядку програми ставиться у відповідність їхній стан і дія над ними. При цьому для кожної змінної вказується стан датчиків (включений, відключений і несуттєвий для даного рядка), команд керування (включити, відключити, не впливати) і значення внутрішніх сигналів.

1.4 Постановка завдань дослідження

Проаналізувавши сучасні методи програмування ПЛК та ПЛК паралельної дії можна зробити висновок що використання технологій значно підвищує продуктивність АСК ТП. Але також залишаються етапи удосконалення яких може значно спростити та знизити затрати обслуговування та програмування ПЛК паралельної дії.

Традиційна технологія програмування ПЛК на сьогодні має ряд недоліків які ускладнюють процес створення та відлагодження програми. Наявність декількох спеціалістів для створення простого алгоритму роботи ППЛК значно підвищує рівень затрат на розробку необхідної замовникові програми.

На сьогодні для процес створення програми для ППЛК будь-якої складності потребує як мінімум двох спеціалістів це інженер-технолог та програміст. Інженер створює алгоритм керування для ППЛК, який може бути представлений у вигляді релейних схем, блок схем, і т.д.. У свою чергу програміст опираючись на отриманий алгоритм та власне уявлення про структуру системи створює програмний код мовою зручною саме програмістові (рисунок 1.13).

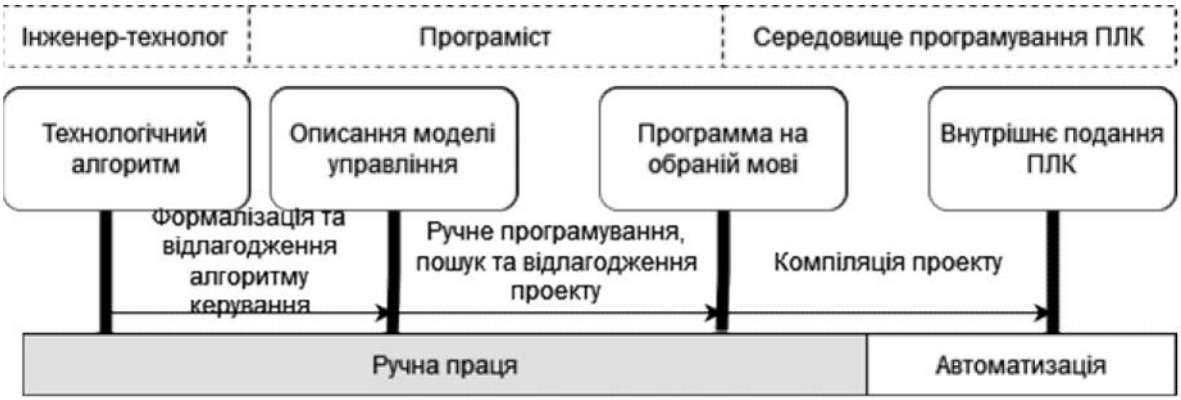


Рисунок 1.13 – Традиційна технологія розробки ПЛК або ППЛК

Такий підхід значно ускладнює процес відлагодження розробленої програми. Оскільки окрім виправлення помилок у коді необхідно також знаходити помилки в алгоритмі які неможливо перевірити без стабільно працюючого коду. Це впливає на час та на собівартість розробки.

Створення спеціалізованого середовища програмування, що дозволяє користувачеві безпосередньо в табличному процесорі технологічними термінами формалізувати модель логічного управління та налагодити логіку управління шляхом виконання табличної моделі (Віртуального контролера), після чого автоматично згенерувати відповідний отриманої моделі програмний код на вибраному стандартною мовою програмування для використовуваної апаратної платформи. Такий підхід до розробки ПЗ отримав назву технологічного візуального програмування чи TVP-технології (Technological visual programming), яка розвивається на цей час у двох основних напрямках (рисунок 1.14) [8].

Рисунок 1.14 – TVP технологія розробки

Спираючись на схему TVP технології можна помітити що автоматизація охоплює більшу кількість етапів розробки ніж традиційна технологія розробки.

Використання технології розробки TVP є ідеальним розв’язання цієї проблеми оскільки конвертація циклограми відбувається автоматично, що значно пришвидшує можливість перевірки створеної автоматизації на

емуляторі або на тестових стендах з ПЛК .

Проаналізувавши сучасні методи програмування ПЛК можна зробити висновок що для програмування та налаштування контролеру необхідно мати навички не тільки у створенні логічних схем роботи автоматизованої системи та знання мов програмування. Такий підхід значно сповільнює роботу з налаштування систем автоматизації та робить її більш ресурсовитратною. Мова програмування ЯППЛК є ідеальним інструментом для створення логіки управління автомату яка є показовою та зручною для реалізації мовами для розробки програм для ПЛК або ПЛК паралельної дії.

Але все ж таки при використанні стандартної технології програмування ПЛК залишається проміжок між інженером та програмістом і під час реалізації може відштовхуватися від своїх власних методів розробки, але головним недоліком цієї технології є те що написання коду відбувається в ручну що може забрати значну кількість часу та ресурсів.

Використання технології розробки TVP є ідеальним рішенням цієї проблеми оскільки конвертація циклограми відбувається автоматично, що значно пришвидшує можливість перевірки створеної автоматизації на емуляторі або на тестових стендах з ПЛК.

Як розглядалося вище використання мови ЯПЛК-М можна стверджувати, що застосування її у якості табличної системи опису моделі керування ПЛК паралельної дії є доречним, адже відображення алгоритму керування є простим та інтуїтивно зрозумілим. Але процес створення таблиць мови ЯПЛК-М займає багато часу через великий обсяг даних які необхідно заповнювати.

Таким чином процес створення програмного забезпечення для ПЛК паралельної дії можна у значній степені пришвидшити, створивши програмний додаток для генерації таблиць (циклограм) мовою ЯПЛК-М.

2 ОГЛЯД СУЧАСНИХ СЕРЕДОВИЩ ПРОГРАМУВАННЯ

2.1 Огляд сучасних середовищ і технологій програмування для створення програмного забезпечення для ПК

Сучасна IT індустрія дуже швидко розвивається, кожен день є можливість релізу нового середовища програмування або технології що безперечно дозволяє дуже точно підібрати для конкретного проекту все необхідне для його спрощення розробки. Для створення десктопних додатків існує дуже багато технологій які дозволяють створювати додатки під конкретну платформу так і кросплатформний додаток. Окрім технологій розробки та контролю проекту дуже важливим компонентом є середовище розробки (IDE) яке має спростити процес розробки.

Згідно до статистики PYPL PopularitY на сьогоднішній день серед популярних IDE для розробки десктопних додатків є наступні:

- Visual Studio;
- Eclipse;
- Visual Studio Code;
- pyCharm;
- Sublime Text;
- Xcode;
- Atom;
- Code::Blocks;
- Vim;
- Qt Creator.

Серед зазначеного списку багато програм не зовсім відносяться до того що вважається IDE у цьому випадку їх можна вважати як звичайні текстові редактори, але завдяки можливості додавати необхідні інструменти вони можуть перетворитися на потужні середовища розробки. Також більшість зазначених IDE створенні під конкретну технологію та платформу.

Наприклад Visual Studio є потужним IDE для розробки десктопних додатків мовами C/C++ або C# під платформу Windows. Коли своєю чергою QTCretator є кросплатформним IDE, але дане середовище створено для фреймворку QT для мови C++. Visual Studio Code є кросплатформним рішенням та відноситься то того типу IDE які зазначалися вище, дане IDE може бути використаним для розробки ПЗ на будь-якій мові завдяки системі розширень яку поповнюють самі користувачі цієї IDE оскільки проєкт є відкритим. Подібними можливостями володіють ще такі IDE як Vim та SublimeText, та все-таки Vim дещо відрізняється від двох інших тим що він є швидшим оскільки не потребує дуже високих затрат на зображення контексту тому що Vim можна запустити в терміналі. Також завдяки дуже поширеному суспільству користувачів існують багато плагінів під будь-яку мову розробки. Але все ж необхідно вибрати технологію для того аби вибрати найбільш комфортну для неї IDE.

Існує безліч технологій для створення ПЗ, але оскільки метою роботи є створення GUI додатку то список технологій значно звужується. Розглянемо представлений список з 6 відомих мов які понад усе підходять для створення GUI додатку згідно зі статистикою сайту PYPL PopularitY:

- Python;
- Java;
- JavaScript;
- C#;
- C/C++;
- Rust.

Як можна бачити на першому місці стоїть Python, що не є випадковістю оскільки дана мова має низький рівень входу, на відміну від усіх інших представлених мов. Але окрім популярності для розробки GUI додатку стартових можливостей недостатньо, як і усіх мов зазначених нижче. Отже, слід розглянути інструменти для створення GUI додатку для даного списку мов, технологія Java розглядатися не буде, оскільки подібна та більш

стабільна технологія реалізована компанією Microsoft та має назву .Net, дана технологія буде розглянута нижче.

У випадку використання мови Python для розробки GUI додатків є три графічні інтерфейси, які дещо відрізняються між собою підходами до створення GUI, але усі рішення є кросплатформними:

- Tkinter;
- wxPython;
- PyQt.

На відміну від багатьох фреймворків графічного інтерфейсу, Tkinter є єдиним фреймворком, вбудованим у стандартну бібліотеку Python. Візуальні елементи відображаються з використанням нативних елементів операційної системи, тому програми, створені за допомогою Tkinter, виглядають так, ніби вони належать платформі, на якій вони запускаються. Однак можна зазначити що графічні інтерфейси, створені за допомогою Tkinter, виглядають застарілими. Однак Tkinter легкий і відносно безболісний у використанні порівняно з іншими фреймворками.

Набір інструментів графічного інтерфейсу wxPython – це обгортка Python навколо бібліотеки C++ під назвою wxWidgets. Основна відмінність wxPython від інших наборів інструментів, таких як PyQt або Tkinter, полягає в тому, що wxPython використовує фактичні віджети на рідній платформі, коли це можливо. Завдяки цьому програми wxPython виглядають рідними для операційної системи, на якій вони працюють.

PyQt і Tkinter самі малюють свої віджети, тому вони не завжди збігаються з рідними віджетами, хоча PyQt дуже близький.

Це не означає, що wxPython не підтримує спеціальні віджети. Насправді, набір інструментів wxPython містить багато користувацьких віджетів, а також десятки основних віджетів. На сторінці завантажень wxPython є розділ «Додаткові файли», який варто перевірити.

PyQt – це зв’язка Python для Qt, яка являє собою набір бібліотек C++ та інструментів розробки, які включають незалежні від платформи абстракції

для графічних інтерфейсів користувача (GUI), а також мережу, потоки, регулярні вирази, бази даних SQL, SVG, OpenGL, XML та багато інших потужних функцій.

У випадку якщо оптимальним буде використання JavaScript технології для розробки GUI додатку фреймворк який понад усе підходить для виконання подібних задач є Electron – це фреймворк для створення GUI додатків за допомогою JavaScript, HTML і CSS. Вбудовуючи Chromium і Node.js у свій двійковий файл, Electron дозволяє вам підтримувати одну кодову базу JavaScript і створювати між платформні програми, які працюють на Windows, macOS та Linux.

Наступною в списку йде мова C# яку неможливо сприймати без технології .NET. дана технологія дозволяє створювати ПЗ майже усіх видів, стандартний .NET Framework дозволяє дуже швидко створювати GUI додатки для платформи Windows, для створення кросплатформного ПЗ слід використати технологію .NET Core, але ця технологія не має вбудованих графічних інтерфейсів для створення GUI додатку і в такому випадку використовується або зв’язка .NET Core з Electron, технологія Electron розглядалася вище, також можливо створення локального веб-додатку завдяки технології ASP.NET та запускати створений графічний додаток у браузері. Також є кросплатформний GUI framework з відкритим кодом для .NET Core під назвою Avalonia, який дещо схожий на стандартний фреймворк для платформи Windows WPF оскільки для верстання використовує файл формату xaml.

Найпоширенішим фреймворком для розробки графічного додатку мовами C/C++ є Qt. Це кросплатформний та дуже потужний фреймворк який дозволяє швидко створювати не тільки графічні додатки. Технологія QML є дуже зручною та простою в розумінні, також модулі QML можуть підтримувати JavaScript код, що надає можливість реалізувати графічний додаток не використовуючи C/C++. В загалі Qt використовується не тільки для мов C/C++, вище зазначалося використання Qt мовою Python, також

.NET Core може використовувати даний фреймворк що робить його універсальним. Але окрім всіх переваг, як зазначалося при описі графічного інтерфейсу PyQt, інтерфейс дуже сильно виділяється від звичного для системи. Єдине місце на якій інтерфейс Qt не буде виділятися це графічна оболонка KDE, яка була створена за допомогою Qt.

Останньою мовою є Rust, що не є дивним, адже дана мова є наймолодшою з усіх перерахованих. Перший реліз мови був 7 Липня у 2010 році, на цей час мові всього 11 років і вона вже займає високе місце у сьогоднішній IT індустрії. За цей час було створено близько 23 графічних бібліотек для створення GUI, але на жаль дизайн кожної бібліотеки відрізняється від сучасного дизайну будь-якої ОС, більшість бібліотек створюють зовсім застарілий графічний дизайн, однією кросплатформною бібліотекою для Rust є бібліотека iced перший реліз бібліотеки був 2 Квітня 2020 року, і не зважаючи на це бібліотека працює дуже швидко та є зручною для розробки на відміну від багатьох інших бібліотек.

2.2 Обґрунтування вибору програмних засобів для створення технології програмування контролерів паралельної дії

Головною проблемою при створенні додатку є вибір інструментів для його створення, через те, що обраний стек технологій та інструментів у майбутньому може створювати додаткові проблеми при розробці нового функціоналу або відлагодження проблем у вже створеному. Розробка додатку під конкретну операційну систему є найпростішим методом, але на жаль це зменшить коло можливостей використання додатку для користувачів інших операційних систем, створення кросплатформного додатку створює інші проблеми, оскільки є можливість що деякий функціонал не зможе бути виконаний на конкретній системі.

Ідеальним розв’язання проблеми може стати створення додатку з використанням технології Electron, такий підхід може одразу розв’язувати

проблему кросплатформного додатку, детально технологія Electron розглядалася вище. Але окрім всіх плюсів вибору створення додатку з використанням технології Electron, є декілька моментів які можуть не відповідати вимогам до створюємого додатку. Оскільки у випадку використання технології загальна складність проекту ускладнюється через те що для розробки такого додатку необхідно встановити додатковий стек технологій таких як nodejs та менеджер пакетів для нього під назвою npm. Також збільшиться сам проект оскільки npm має завантажити увесь необхідний для розробки код фреймворку Electron для створюємого додатку це занадто, оскільки розробка з використанням інших технологій буде значно простішою та не займатиме стільки місця. Отже, дана технологія не підходить для реалізації необхідного додатку.

Розглядаючи технологію .NET було зазначено що для створення кросплатформного рішення може бути використана технологія Electron, але як було зазначено вище вона не підходить, також був згаданий кросплатформний інтерфейс для розробки графічних додатків під назвою Avalonia який розглядався вище. Для комфортної розробки графічного додатку використовуючи технології .NET Core та Avalonia, рекомендується встановити спеціалізоване середовище розробки JetBrains Raider яке є пропріетарним та потребує оплати, хоча і даються 30 днів безкоштовного користування.

Використання Qt для розробки додатку мовою C++ є гарним рішенням, через те що на відміну від Raider Qt Creator є безплатний, також при використанні C++ можна гарантувати що додаток буде швидко працювати на всіх платформах. Але не глядачі на всі переваги дана технологія потребує встановлення спеціалізованого IDE яке відображає структуризований проект коли на справді всі файли знаходяться в одній дерикторії та це все необхідно змінювати вручну. Але на відміну від попередньо розглянутих технологій, для створення додатку за допомогою Qt необхідно лише встановити framework і цього почати розробку. Але кількість затраченого часу може

підвищиться завдяки особливостям мови C++, а саме під час відлагодження. Використання технології Python замість C++ може розв’язувати проблему оскільки кількість затраченого часу на відлагодження коду написаного на Pyrhon значно менше у зв’язку з парадигмами мови. Але як зазначалося вище інтерфейс Qt значно відрізняється від стандартних системних інтерфейсів.

Rust є більш оптимальним для розробки графічного додатку. Наявність пакетного менеджеру cargo значно спрощує процес розробки завдяки інструментам які дозволяють спростити процес розробки. До інструментів, завдяки яким процес спрощується відноситься так званий файл маніфесту з додатковими метаданими які використовуються задля вказівки необхідних авторові залежностей, до більш вагомих відносяться наступні:

- додаткові бібліотеки;
- шляхи до локальних бібліотек;
- конфігурація виконавчих файлів.

2.3 Складові технології програмування контролерів паралельної дії та етапи її створення

Проаналізувавши сучасні методи програмування ПЛК паралельної дії та технології розробки ПЗ слід визначитися зі складовими особливостями розроблюємого додатку. В основі своїй розроблюємий додаток є генератором циклограм для мови ЯПЛК-М, тому додаток має мати можливість редагувати створені циклограми, але оскільки розмір таблиці може бути занадто великим для комфортної розробки є сенс у створенні більш компактного графічного зображення циклограми.

Спочатку слід реалізувати графічний інтерфейс для заповнення циклограм для мови ЯПЛК-М. Опираючись на структуру циклограм в мові то, процес створення таблиць можна значно спростити. Оскільки ліва частина таблиці 2 дуже схожа на таблицю мови ЯПЛК тільки відсутній кадр умови. Отже перше що необхідно зробити це сконфігурувати вхідні та

вихідні сигнали, адже без них усі інші дані таблиці не матимуть смислу. Оскільки сигнал в мові ЯПЛК-М ділиться не лише за видом вхідний або вихідний, ще є розділення за типом його використання у мові як сигнал стану або сигнал керування, тому графічне відображення конфігурації сигнал пропонується реалізувати так, як зображено на рисунку 2.1.

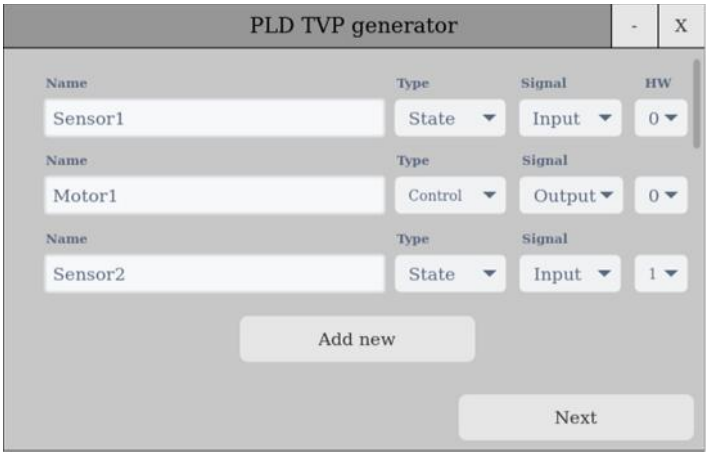


Рисунок 2.1 – Вікно конфігурації сигналів

Після того як усі необхідні сигнали буде сконфігуровано, тоді ці сигнали можуть бути використані при створені таблиць мови ЯПЛК-М. Замість прописування одних й тих самих сигналів по декілька разів під час заповнення таблиць тепер це необхідно зробити 1 раз і надалі використовувати ці дані. Перший етап конфігурації у подальшому може зекономити багато часу, тому для розроблюваної програми є обов’язковим.

Як зазначалося при огляді мови ЯПЛК-М в ній присутні 2 таблиці. Перша таблиця створена для виклику підпрограм які описані в другій таблиці опираючись на описані статусні сигнали першою, тому створення другої таблиці має бути першим.

Отже, процес заповнення даних другої таблиці для комфортного користування було розподілено на 3 етапи. Першим етапом є процес додавання нових підпрограм який має виглядати приблизно так як зображено на рисунку 2.2.

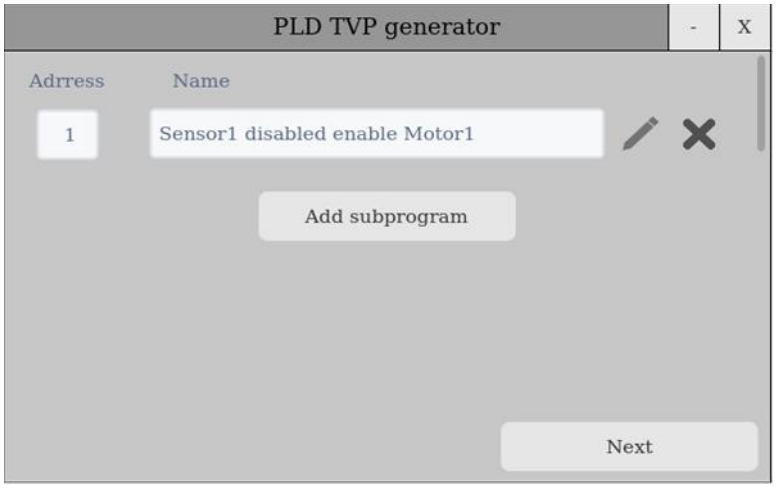


Рисунок 2.2 – Вікно додавання нових підпрограм

Під час цього етапу має бути надана можливість додавати нові підпрограми та редагувати вже існуючи. Таке графічне рішення дозволить краще орієнтуватися поміж підпрограм оскільки відображається лише загальна інформація про підпрограму приховуючи деталі які в даному етапі були б зайвими. Оскільки у випадку коли необхідно буде внести зміни у підпрограму під час її редакції в табличному форматі є шанс не наміряно змінити код іншої підпрограми. У випадку коли деталі підпрограми винесені окремо, то її зміна не зможе зачепити інші підпрограми.

Етап створення підпрограми має містити у собі можливість додавання або видалення кроків. Також мати можливість вибирати сигнали з сигналів, що були сконфігуровані раніше. Приблизний вигляд даного етапу зображено на рисунку 2.3.

Наведене рішення має надати змогу користувачеві доволі швидко створювати підпрограми обираючи зі списків необхідні елементи. Такий метод відображення є більш зручним лише на етапі створення циклограм в подальшому використані відображення у вигляді таблиці є зручнішим.

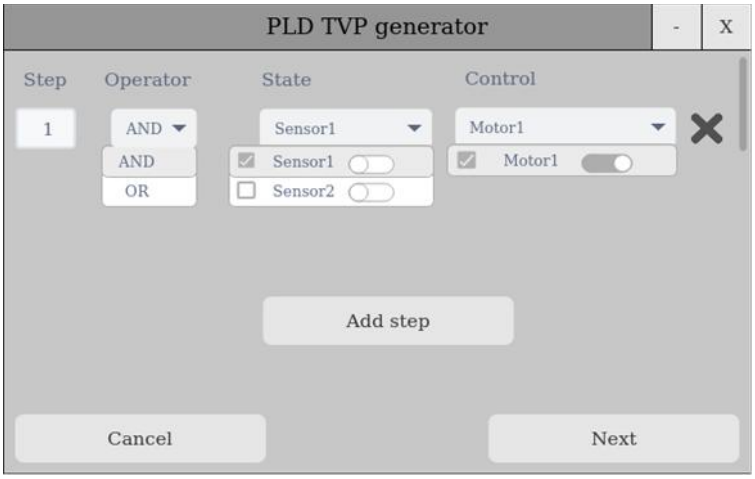


Рисунок 2.3 – Етап конфігурації підпрограми

При завершені редагуванні підпрограми наступним кроком має бути етап надання підпрограмі опису адреси та типу підпрограми які були описані мовою ЯПЛК-М для другої таблиці (рисунок 2.4).

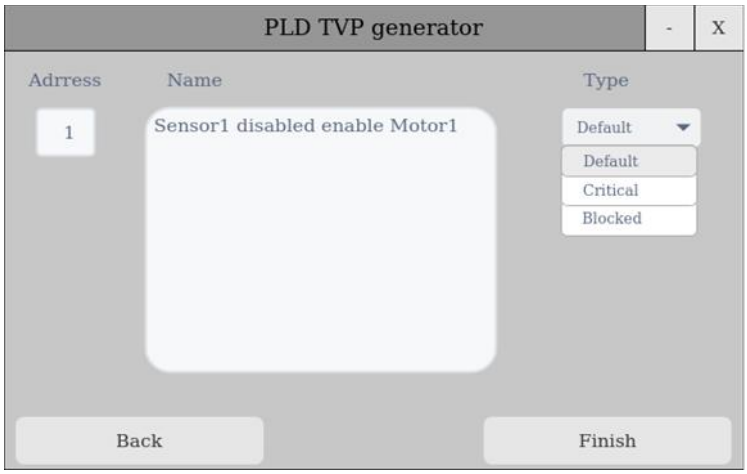


Рисунок 2.4 – Останній етап редагування підпрограми

Після конфігурації другої таблиці заповнення першої тепер є значно легшим, оскільки усі необхідні дані вже створенні та можуть бути використаними. Оскільки перша таблиця мови ЯПЛК-М має менше унікальних даних та являє собою циклограму умов, то розбивати її немає сенсу (рисунок 2.5).

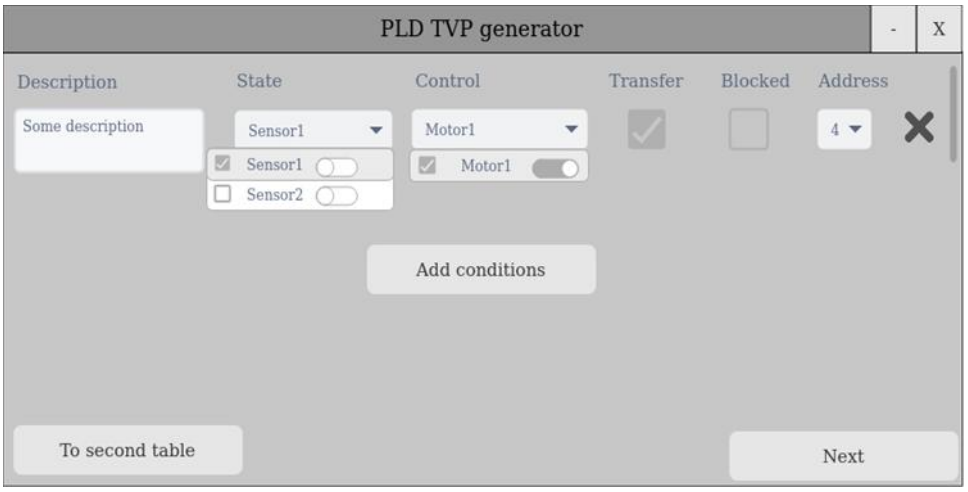


Рисунок 2.5 – Етап конфігурації першої таблиці мови ЯПЛК-М

Після конфігурації обох таблиць фінальний етап це генерація циклограм у табличний формат відображення. Проаналізувавши всі головні етапи розроблюваної технології можна підбити підсумок.

Описані етапи створені для спрощенні процесу розробки циклограм для мови ЯПЛК-М. Кожен етап відповідає за конфігурацію певного модулю який буде використаний наступними. Також можливість завантажувати готові циклограми у створений додаток з метою їх зміни, може стати однією з ключових функцій.

3 РОЗРОБКА ДОДАТКУ РЕАЛІЗАЦІЇ МОВИ ЯПЛК-М

3.1 Опис необхідного програмного забезпечення

Підсумовуючи серед розглянутих вище технологій та середовищ які найбільше підходять для розробки графічного додатку, час визначити кінцевий стек програмного забезпечення для розробки GUI. Оскільки програмне рішення є кросплатформним, то для реалізації буде використовуватися платформа Linux. Як зазначалося вище обрана платформа значно більше підходить для розробки ПЗ, використання платформи значно підвищить процес розробки.

Серед усіх розглянутих середовищ розробки, для реалізації додатку використовуватиметься Vim оскільки є менш ресурсозатратним та швидшим серед усього списку розглянутих середовищ.

Також при розробці використовуватиметься система контролю версій Git. Яка дозволить швидко знаходити проблемні ділянки програми. Використання Git Flow дозволить у майбутньому дуже просто розширювати функціонал та контролювати версії самого додатку. Окрім цього система контролю версій Git дозволяє додати віддалені ресурси для зберігання репозиторію.

Окрім цього для створення архітектури додатку використовувався онлайн сервіс для створення UML діаграм daraw.io. Обраний сервіс дозволяє швидко та зручно створювати UML діаграми та надає можливість зберігати створену діаграму в форматах PNG та SVG.

Для реалізації пропонується використання мови Rust, що також додає до неї пакетний менеджер cargo. Оскільки стандартна бібліотека мови не має кросплатформних рішень для створення графічного інтерфейсу, то цей функціонал буде виконувати графічна бібліотека iced з відкритим кодом, що дозволяє використовувати її безкоштовно.

3.2 Розробка архітектури програмного продукту

Архітектура системи описує її основні компоненти, їх взаємозв’язки (структури) і те, як вони взаємодіють один з одним. Архітектура та дизайн програмного забезпечення охоплює кілька факторів, що сприяють, таких як бізнес-стратегія, атрибути якості, людська динаміка, дизайн та IT-середовище.

Структурні елементи архітектури охоплюють статистичний набір підсистем і компонентів що становлять систему, а також відповідальності цих підсистем та їх організацію у більші підсистеми. Важливо розглядати також залежності між компонентами та іншими зовнішніми системами. На цьому рівні й з цієї точки зору організація архітектури важлива також для координування робіт і планування проекту. Команда розробників проекту організовується згідно зі структурою – різні групи відповідають за реалізацію різних підсистем. Постійність у вибраній архітектурі проекту дозволяє ефективно управляти плануванням і розробкою.

Для розроблюваного додатку архітектура має бути простою та гнучкою у випадку коли знадобиться додати функціонал.

«Шаблон архітектури Model-View-Controller (MVC)» передбачає відокремлення моделі даних програми, рівня презентації та аспектів управління. Нижче наведено його характеристики:

- є три будівельні блоки, а саме модель, подання та контролер;
- дані програми знаходяться в моделі;
- користувачі бачать дані програми через подання, однак подання не може впливати на те, що користувач буде робити з даними;
- контролер є будівельним блоком між моделлю і представленням.

Перегляд ініціює події, згодом на нього діє контролер. Дія зазвичай є викликом методу моделі. Відповідь відображається у вікні.

Оскільки графічне відображення в розроблюємому програмному

додатку, то внутрішня архітектура значно спрощується до простих типів даних та методів їх обробки.

Отже, процес генерації циклограм для мови ЯПЛК-М поділяється на декілька етапів. Таким чином данні які будуть записані на кожному етапі можна розділити три основні модулі:

- модуль конфігурації вхідних або вихідних сигналів;
- модуль конфігурації підпрограми;
- модуль конфігурації залежностей між підпрограмами.

Також слід зазначити що, модулі не повинні мати прямого доступу для зміни даних іншого модулю аби уникнути колізій при роботі програми. Усі основні зміни даних повинні робитися на необхідних модулеві етапах. Наприклад сконфігуровані вхідні або вихідні сигнали можна змінити лише на етапі їх створення, на інших етапах ці дані можуть лише бути доступні тільки для зчитування, а не для зміни.

Спираючись на задані вимоги розроблено архітектуру додатку (рисунок 3.1).

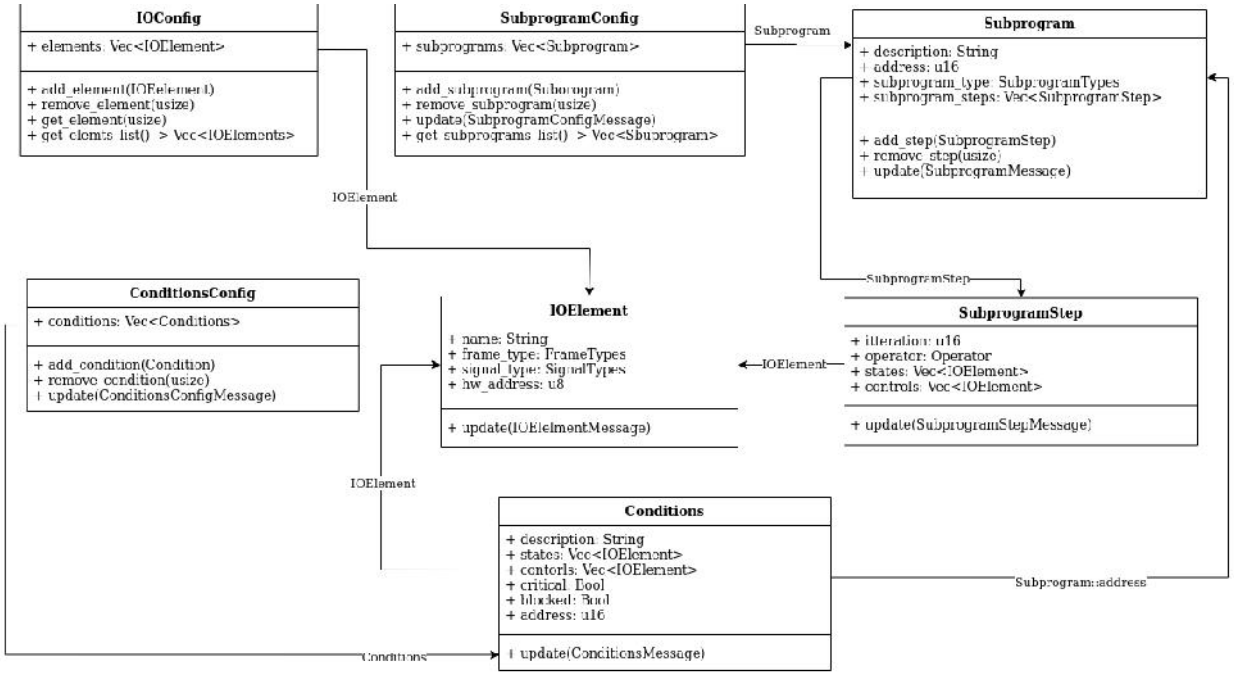


Рисунок 3.1 – Архітектура програмного додатку

Наведена архітектура дозволяє використовувати данні з різних модулів та підмодулів як і було зазначено в вимогах. Також можна зазначити що найбільше залежностей використовують підмодуль IOConfigElement модулю IOConfig це пов’язано з особливостями мови ЯПЛК-М, а саме з тим що обидві таблиці мови використовують вхідні та вихідні сигнали для створення залежностей між станом сигналу та кроком у таблиці з підпрограмами й станом сенсору або ж керуючого пристрою та адресою підпрограми у таблиці з залежностями.

Також можна зазначити що, адреса підпрограми в таблиці залежностей не є випадковим числом яке може вписати користувач. Навпаки, користувачеві надається список з адрес які були записані на етапі створення підпрограм. Таким чином користувач не може задати адреси якої не існує.

Отже програмна архітектура задовольняє встановлені вимоги, але окрім цього слід розробити архітектуру графічної частини програми для спрощення відображення конфігураційних етапів.

Створення архітектури графічного інтерфейсу базується на внутрішній архітектурі графічної бібліотеки iced. По-перше кожен етап має своє унікальне відображення, тобто можна їх винести в окремий enum для зручного перемикання та виклику необхідних методів між ними.

Мова Rust в цьому випадку є ідеальним рішенням, оскільки у зв’язку з особливостями своєї архітектури цей, на перший погляд примітивний, тип даних є потужним інструментом, через те що кожен елемент переліку має можливість містити внутрішні данні.

Виходячи із цього можна сказати, що кожне відображення конкретного етапу є пресетом цього етапи, тобто створивши enum Presets та записавши в ньому усі пресети та данні які будуть використанні для графічного відображення даних етапу, дозволить реалізувати паттерн проектування під назвою стратегія (лістинг 3.1).

Лістинг 3.1 – Представлення паттерну стратегія

```
#[derive(Debug)]
pub enum Presets {
    Entry {
        create_new_button: button::State,
        load_table_button: button::State,
    },
    IOConfig {
        scroll: scrollable::State,
        create_new_button: button::State,
        elements: Vec<IOConfigElement>,
    },
    SubprogramConfig {
        scroll: scrollable::State,
        create_new_button: button::State,
        subprogramms: Vec<Subprogram>,
    },
    SubprogramStepsConfig {
        scroll: scrollable::State,
        create_new_button: button::State,
        steps: Vec<Step>
    },
    ConditionsConfig {
        scroll: scrollable::State,
        create_new_button: button::State,
        conditions: Vec<Conditions>
    }
}
```

Шаблон стратегія передбачає, що ви берете клас, який виконує щось конкретне багатьма різними способами, і виділяє всі ці алгоритми в окремі класи, які називаються стратегіями. Початковий клас, який називається контекстом, повинен мати поле для зберігання посилання на одну зі стратегій. Контекст делегує роботу пов’язаному об’єкту стратегії замість того, щоб виконувати її самостійно. Контекст не відповідає за вибір відповідного алгоритму для роботи. Замість цього клієнт передає потрібну стратегію в контекст. Насправді, контекст не знає багато про стратегії. Він працює з усіма стратегіями через той самий загальний інтерфейс, який надає лише один метод для запуску алгоритму, інкапсульованого у вибраній стратегії. Таким чином контекст стає незалежним від конкретних стратегій, тож ви можете додавати нові алгоритми або змінювати існуючі, не змінюючи

код контексту чи інших стратегій.

Використання паттерну стратегія дозволить під час роботи програми змінювати пресети. Тобто таким чином можливість змінювати пресети зводиться до вказання необхідного пресету із вказаного переліку Presets. Оскільки при виклику методу view завдяки оператору match можливо в залежності від типу Presets викликати необхідний метод відображення конкретного пресету (лістинг 3.2).

Лістинг 3.2 – Відображення роботи патерну стратегія

```
pub fn view(&mut self) -> Element<PresetMessage> {
        match self {
           Presets::Entry {
                create_new_button,
                load_table_button
            } => Self::entry_view(create_new_button,
load_table_button),
            Presets::IOConfig {
                scroll,
                create_new_button,
                elements
            } => Column::new()
                    .push(Self::ioconfig_view(
                            create_new_button,
                          ).map(PresetMessage::IOConfigMessage))
           Presets::SubprogramConfig {
                create_new_button,
                load_table_button
            } =>
Self::submprogtramconfig_view(create_new_button,
load_table_button),
           Presets::SubprogramStepsConfig {
                create_new_button,
                load_table_button
            } => Self::subprogram_steps_view(create_new_button,
load_table_button),
           Presets::Subprogram {
                create_new_button,
                load_table_button
            } => Self::subprogram_view(create_new_button,
load_table_button),
           Presets::ConditionsConfig {
                create_new_button,
                load_table_button
            } => Self::conditions_view(create_new_button,
load_table_button),
```

```
    }
    .into()
```

Також бібліотека використовує повідомлення для обміну даними між об’єктами такий підхід дозволяє відокремити пряму залежність між собою, що дозволяє підвищити швидкість роботи графічного інтерфейсу.

Визначившись з основними архітектурними рішеннями для розробки додатку слід приступити до її реалізації.

3.3 Опис програмної реалізації розробленого програмного додатку

Отже, у зв’язку з особливостями архітектури графічної бібліотеки iced та самої мови Rust, повністю реалізовувати шаблон MVC не є доцільним, через те, що при реалізації з’являються проблеми з явним дублюванням коду та порушенням правил шаблону. Тому було прийняте рішення спростити шаблон.

Перший етап конфігурації входів та виходів створює масив з сконфігурованих елементів наступні етапи використовують розумний покажчик Rc який відстежує кількість посилань на значення, щоб визначити, чи використовується це значення. Якщо посилання на значення нульові, значення можна очистити. Таким чином самі дані не будуть передаватися між етапами, а лише тільки їх посилання.

Але окрім етапів графічного відображення для функціонування самої бібліотеки необхідно реалізувати методи інтерфейсу Application для цього було створено структуру Generator яка міститиме у собі перелік із пресетів та декілько керуючих елементів.

Лістинг 3.3 – Реалізація методів інтерфейсу Application

```
#[derive(Debug)]
pub struct Generator {
    active_preset: usize,
    scroll: scrollable::State,
```

```
    presets: Vec<Presets>,
    next_preset: button::State,
    back_preset: button::State,
}
impl Application for Generator {
    type Executor = executor::Default;
    type Message = Message;
    type Flags = ();

    fn new(_flags: ()) -> (Generator, Command<Message>) {
    }

    fn title(&self) -> String {
    }

    fn update(
        &mut self,
        message: Message,
        _clipboard: &mut Clipboard,
    ) -> Command<Message> {

    }

    fn view(&mut self) -> Element<Message> {
    }
}

fn main() -> iced::Result {
    Generator::run(Settings {
        antialiasing: true,
        ..Settings::default()
    })
}
```

Для створення простого графічного додатку достатньо реалізувати метод view, саме цей метод відповідає за відображення усіх даних. Не важко дізнатися який метод відповідає за оновлення даних графічного відображення. Як можна зазначити метод update приймає повідомлення яке необхідно опрацювати та змінити необхідні дані.

Отже, оскільки зрозуміло яким чином буде працювати графічна частина додатку, то стає яснішим реалізація інших модулів майбутнього додатку. Використання повідомлень як один із засобів обміну даними дозволить спростити сам процес пересилання даних, а також даний підхід не буде здаватися нав’язаним оскільки сама бібліотека використовує таку

систему.

Розглянемо приклад (лістинг 3.4) використання технології повідомлень для модулю конфігурації вхідних або вихідних сигналів.

Лістинг 3.4 – Опис повідомлень для зв’язку графічної частини з частиною зберігання даних

```
#[derive(Debug, Clone)]
pub enum IOElementMessage {
    NameInputChanged(String),
    FrameTypeSelected(FrameTypes),
    SignalTypeSelected(SignalTypes),
    HwSelected(String),
    DeleteElement,
}
```

Як можна зазначити наведена структура повідомлень відповідає за зміни які відбуваються в графічному модулі додатку. Прив’язання повідомлення до графічного віджету відбувається шляхом передавання необхідного типу повідомлення в момент створення віджету.

Лістинг 3.5 – Прив’язка повідомлення до графічного віджету

```
let type_list = PickList::new(
    mut self.type_list,
    FRAME_TYPES_ALL,
    Some(frame_type),
    IOElementMessage::FrameTypeSelected
);
```

Як можна бачити під час виклику методу new для віджету PickList останнім аргументом передається повідомлення яке віджет має надіслати коли користувач вибере з випадного списку необхідний йому елемент.

Також окрім зберігання та відображення даних додаток має мати можливість генерувати таблиці мовою ЯПЛК-М. Для цієї задачі було обрано бібліотеку xlsxeditor, яка дозволить генерувати таблиці з розширенням xlsx.

При генерації таблиць використовуються сконфігуровані раніше дані і оскільки використання шаблону MVC, генерації непотрібні дані графічних елементів оскільки вони зберігаються в іншому місці і лише використовуються для відображення графічними елементами.

Лістинг 3.6 – Генерація таблиць для мови ЯПЛК-М

```
use std::error::Error;
use csv::Writer;

fn generate<'a>(
    conditions: &'a Conditions,
    subprogramms: &'a Subprograms
) -> Result<(), Box<dyn Error>> {
    let mut cinditions_wtr =
Writer::from_path("conditions.csv")?;
    let mut subprograms_wtr =
Writer::from_path("subprogramms.csv")?;

    wtr.write_record(prepare_subprograms(subprogramms))?;
    wtr.write_record(get_subprograms_content(subprogramms))?;
    wtr.write_record(prepare_conditions(conditions))?;
    wtr.write_record(get_conditions_content(conditions))?;

    wtr.flush()?;

    Ok(())
}
```

Отже, можна зазначити що використання мови Rust та бібліотек iced, json й xlsx надають можливість повністю реалізувати поставлену задачу та були вдалим вибором в якості інструментів розробки програмного додатку. Також використання системи контролю версій git надало змогу зручно додавати новий та змінювати старий функціонал.

4 ОСОБЛИВОСТІ ІНТЕРФЕЙСУ КОРИСТУВАЧА

4.1 Опис інтерфейсу користувача для заповнення циклограми мовою ЯПЛК-М

Графічний додаток було створено з орієнтацією на мультиплатформність тому стиль дещо відрізняється від стандартного стилю системи на якій він запускається. Також структура проекту поділяється на незалежні модулі (рисунок 4.1).

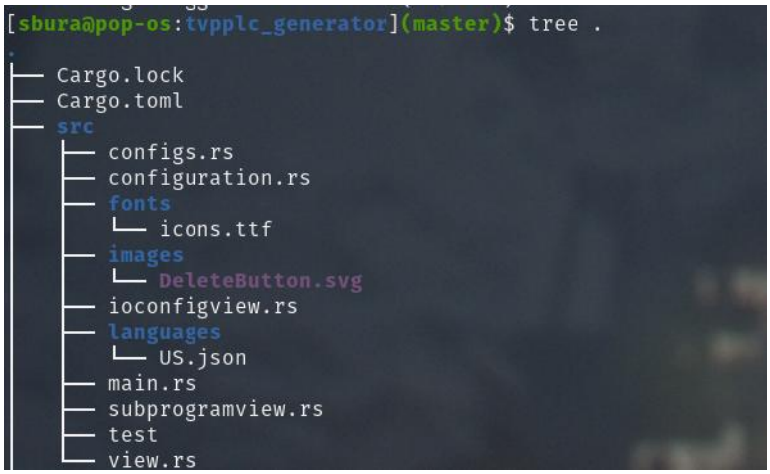


Рисунок 4.1 – Структура проекту

Як можна бачити в структурі є файл локалізації US.json, даний файл містить англійську локалізацію додатку. Це надає можливість легко додавати необхідну локалізацію для програми не витрачаючи час на заміну даних якщо вони б було прописані прямо у коді. Також це надає можливість змінювати локалізацію під час роботи програми.

Як зазначалося вище додаток має бути крос-платформним. Оскільки використана графічна бібліотека сама по собі є крос-платформною, то нічого не потрібно додаткового завантажувати, бібліотека вже містить все необхідне.

Стосовно цього програма добре працює однаково на всіх системах, для прикладу наведено роботу додатку на платформах Linux (рисунок 4.2) та Windows (рисунок 4.3).

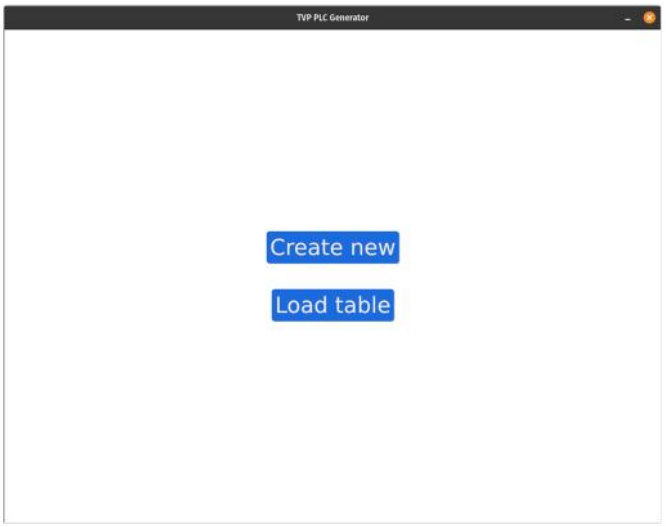


Рисунок 4.2 – Робота додатку на платформі Linux

Для того щоб, запустити програму на платформі слід встановити мову Rust та пакетний менеджер cargo. В майбутньому на сервісі Github будуть розміщенні виконавчі файли для систем Windows та Linux.

Як можна бачити графічний стиль додатку виділяється серед стандартного, для кожної системи, стилю, але різниця не така вже й велика для того щоб якось відволікати користувача.

Як зазначалося вище додаток дозволяє дуже легко змінювати локалізацію шляхом додавання файлу в форматі JSON поля об’єкту якого слід заповнити необхідною мовою та додати його в директорію languages. В прикладі нижче наведено запуск програми з українською локалізацією додатку (рисунок 4.4).

В майбутньому заплановано реалізувати можливість перемикатися між локалізаціями за допомогою графічного інтерфейсу, що спростить процес зміни локалізації. Також буде надана можливість редагувати вже згенеровані таблиці шляхом їх завантаження у додаток.

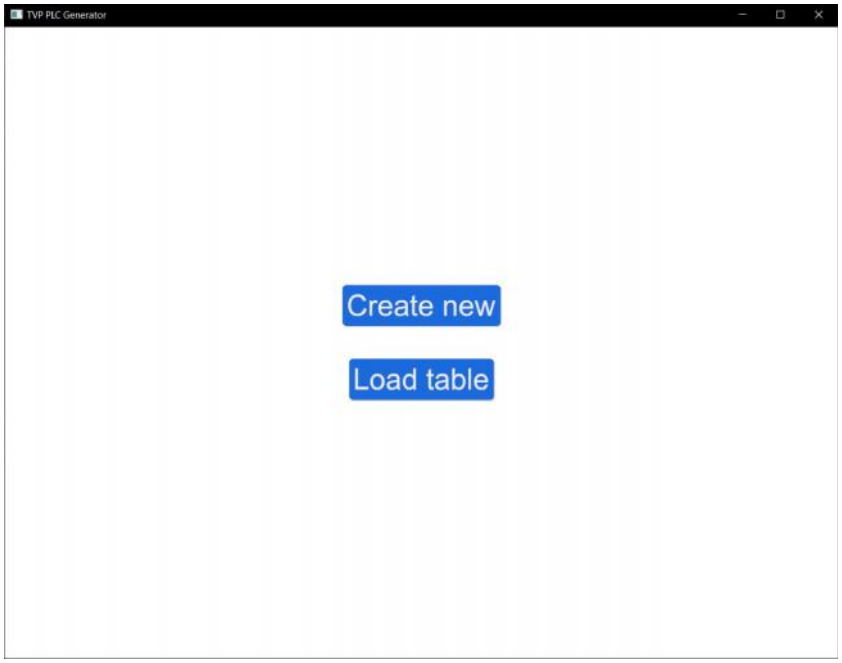


Рисунок 4.3 – Робота програми на платформі Windows

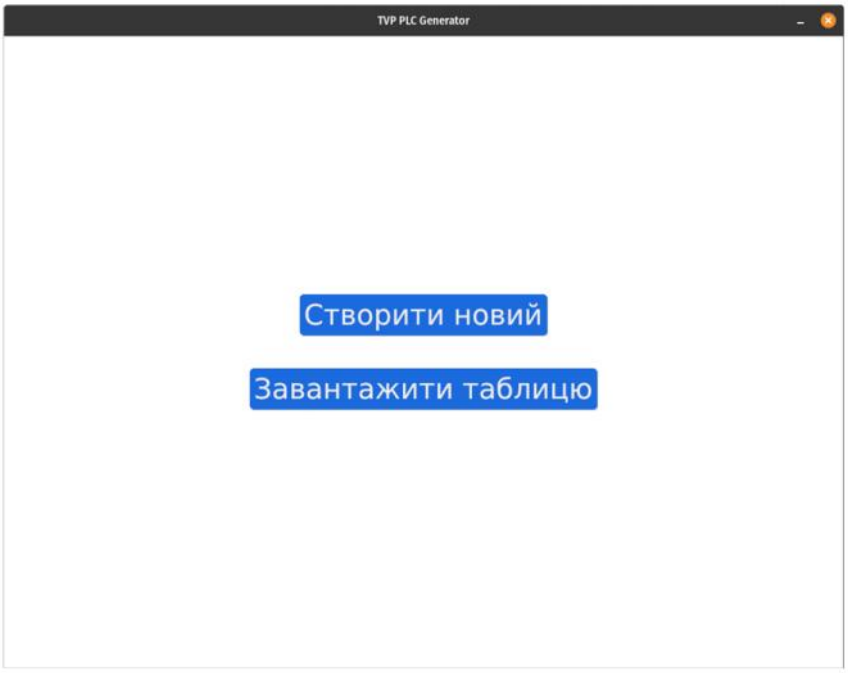


Рисунок 4.4 – Робота додатку з українською локалізацією

4.2 Тестування програмного продукту

Після запуску програмного додатку на зручній користувачеві платформі та з обраною локалізацією програма відкриває стартове вікно (рисунок 4.5).

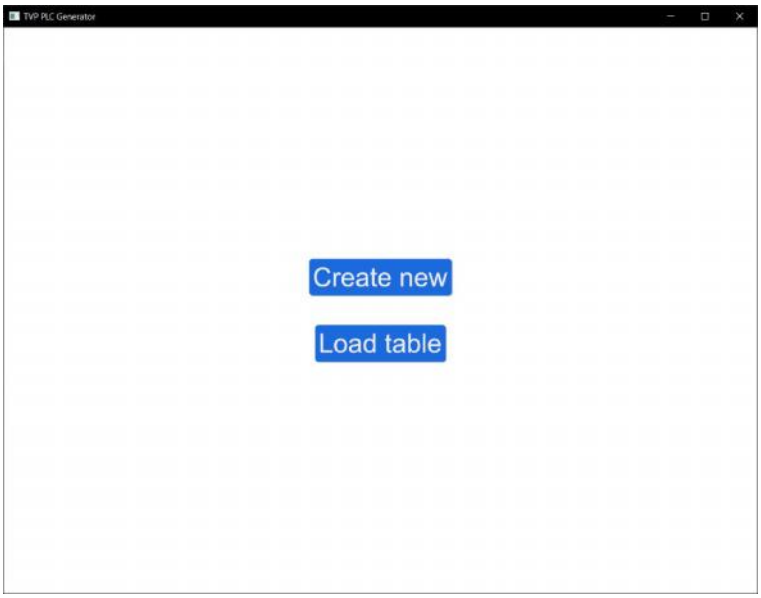


Рисунок 4.5 – Стартове вікно додатку

На стартовому вікні знаходяться дві клавіші кожна з яких у будь якому випадку відкриє вікно з конфігурацією вихідних та вхідних елементів (рисунок 4.6). Але на відміну від клавіші “Create new”, клавіша “Load table” ініціалізує внутрішні дані які програма зчитує з вказаних користувачем таблиць (рисунок 4.7).

Отже, після входу до вікна конфігурації користувачеві надається інтерфейс для конфігурації вхідних або ж вихідних параметрів, у випадку якщо елементи не будуть додані чи навпаки будуть усі видалені, графічний інтерфейс виведе відповідне повідомлення про відсутність елементів.

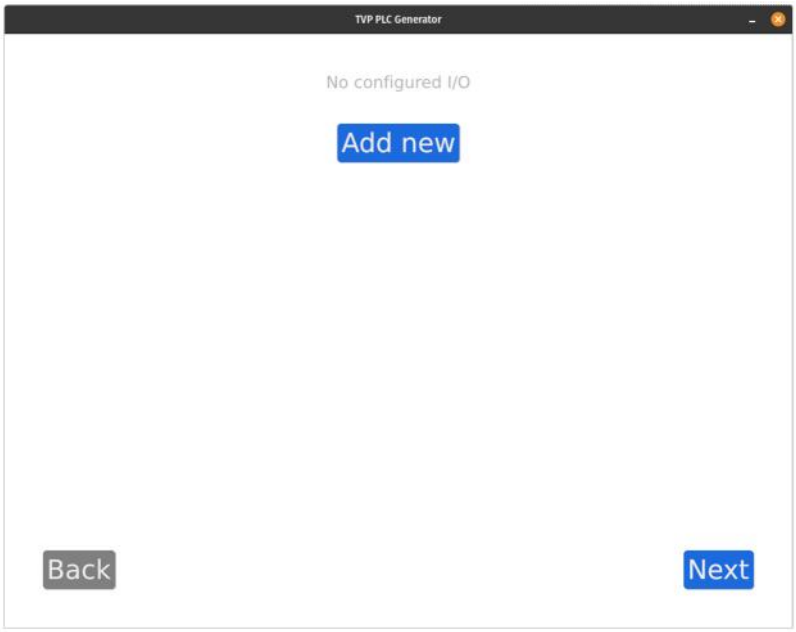


Рисунок 4.6 – Вікно конфігурації вхідних або вихідних сигналів

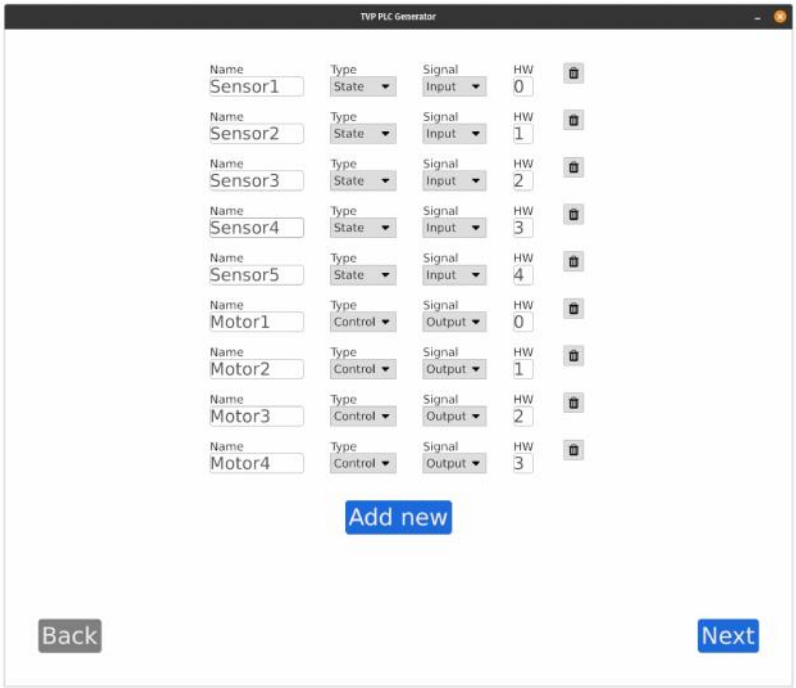


Рисунок 4.7 – Вікно з заповненими даними входів та виходів

Після заповнення даних входів та виходів при натисканні клавіші Next, користувач потрапляє до вікна створення підпрограм для другої таблиці мови ЯПЛК-М, це пов’язано з тим, що перша таблиця має прямий зв’язок з

підпрограмами, оскільки до першої таблиці заносяться адреси підпрограм, тому було прийняте рішення конфігурувати таблицю з підпрограмами раніше.

Потрапивши до вікна конфігурації підпрограм користувачеві надається можливість створювати підпрограми. Оскільки зовнішньо порожнє вікно не дуже відрізняється від порожнього вікна при конфігурації входів, цей етап буде пропущений до моменту натискання клавіші Add new. Після чого на екрані відобразиться нова, порожня, підпрограма.

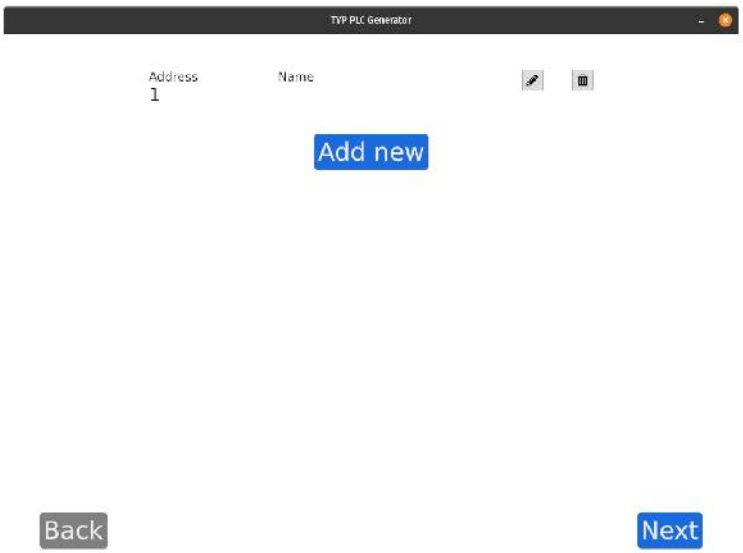


Рисунок 4.8 – Вікно конфігурації підпрограм

Для редагування підпрограми необхідно натиснути на клавішу з зображенням олівця, після чого відкриється вікно редагування кроків обраної підпрограми (рисунок 4.9). Який відображає номер кроку, його короткий опис, тип логічної операції та дві кнопки для додавання залежностей між датчикими стану та виходами керування. Цей етап зовнішньо відрізняється від того що було показано у розділі 2.3 через те що обрана графічна бібліотека не дозволяє додавати сторонні елементи відображення до стандартних графічних об’єктів.

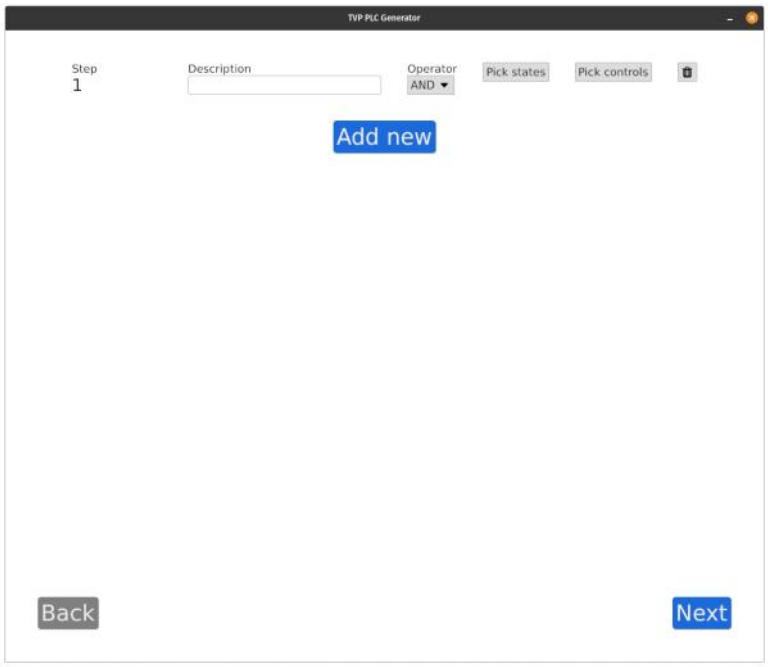


Рисунок 4.9 – Вікно конфігурації кроків підпрограм

Також однією з причин змінення графічного відображення є те, що наведенні у розділі 2.3 приклади не покривають усі можливі стани I/O елементів, оскільки всього вони мають 4 стани, тому відображення стану у вигляді перемикача не є правильним (рисунок 4.10).

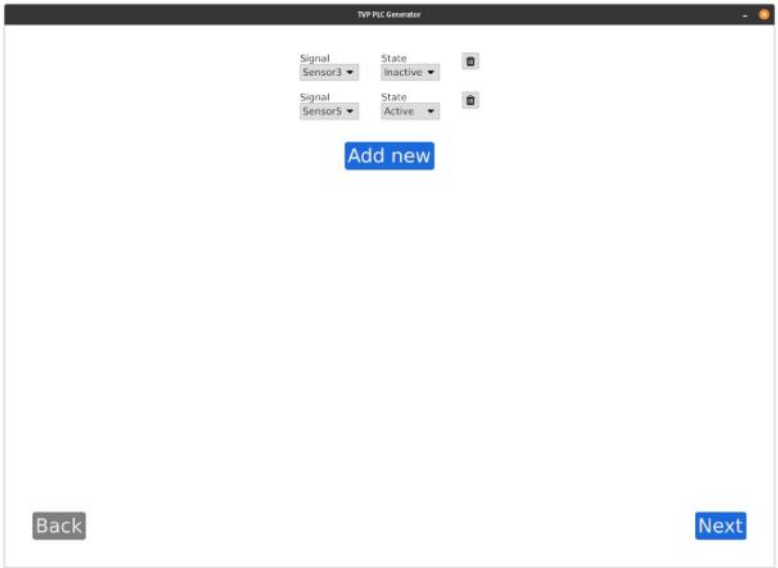


Рисунок 4.10 – Вікно додавання I/O елементів

Для того аби повернутися до вікна конфігурації кроків підпрограми можна натискати на будь-яку кнопку.

Після конфігурації підпрограм при натисканні кнопки Next із вікна конфігурації підпрограм, користувач потрапляє до вікна конфігурації першої таблиці (рисунок 4.11). У ньому знаходиться вся необхідна для таблиці 1 інформація. Аналогічно з вікном конфігурації кроків таблиці, дане вікно також використовує кнопки для завдання залежностей станів сенсорів та елементів керування між адресами підпрограм.

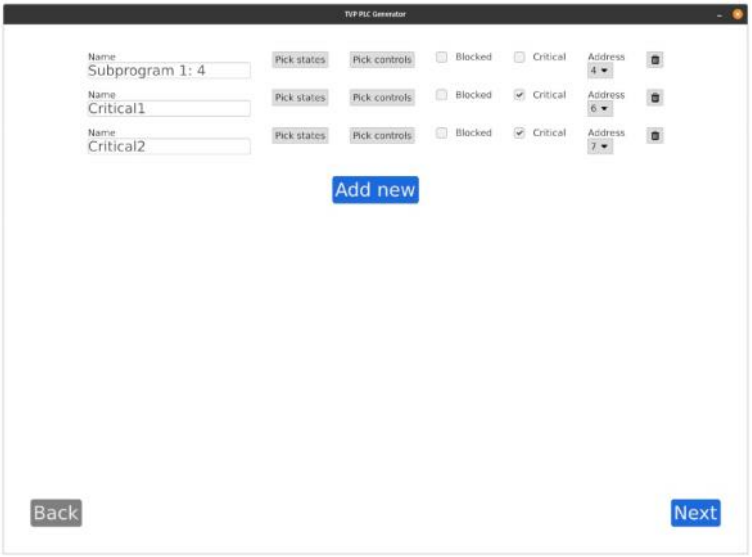
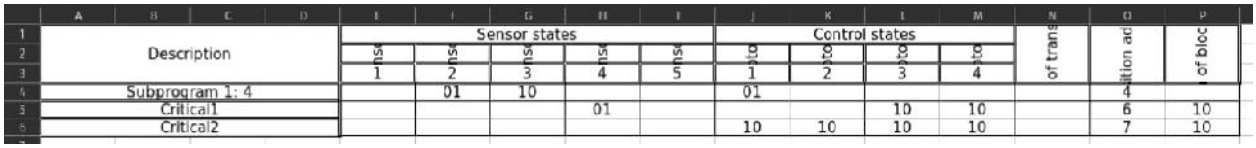


Рисунок 4.11 – Вікно конфігурації залежностей

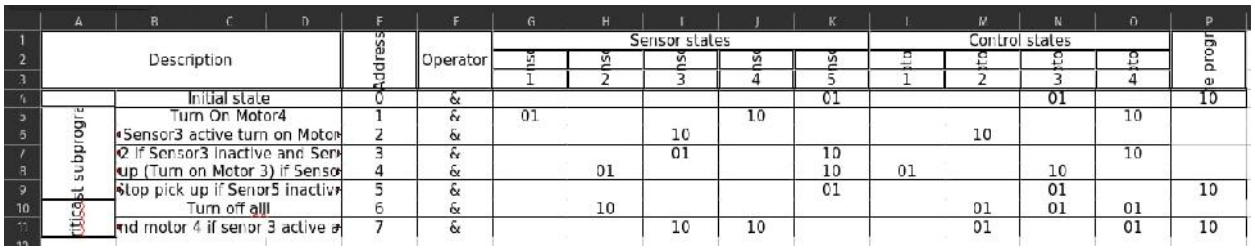
Після того як користувач заповнює усі необхідні дані для генерації таблиць мови ЯПЛК-М, у вікні редагування таблиці 1 слід натиснути кнопку Next після чого відкриється вікно з кнопкою генерації програми натискання якої створює в папці створеного додатку таблицю в форматі xlsx.

Як можна бачити на рисунках 4.12 і 4.13 згенеровані таблиці потребують постредакції у плані розтягування до зручного зчитування комірки та стовбці і на цьому все, все інші данні були записані так як було необхідно.



Description	Sensor states					Control states						
	ins	ins	ins	ins	ins	sto	sto	sto	sto	of trans	dition ad	of bloc
	1	2	3	4	5	1	2	3	4			
Subprogram 1: 4		01	10			01					4	
Critical1				01				10	10		6	10
Critical2						10	10	10	10		7	10

Рисунок 4.12 – Фрагмент згенерованої таблиці 1 мови ЯПЛК-М



	Description	Address	Operator	Sensor states					Control states					
				ins				ins	sto			sto		e progr
				1	2	3	4	5	1	2	3	4		
	Initial state	0	&					01			01			10
	Turn On Motor4	1	&	01			10					10		
	Sensor3 active turn on Motor	2	&			10				10				
	If Sensor3 inactive and Sen	3	&			01		10				10		
	up (Turn on Motor 3) if Senso	4	&		01			10	01		10			
	top pick up if Senor5 inactiv	5	&					01			01			10
	Turn off all!	6	&		10					01	01	01		
	nd motor 4 if senor 3 active a	7	&			10	10			01		01		10

Рисунок 4.13 – Фрагмент згенерованої таблиці 2 мови ЯПЛК-М

Підсумовуючі можна зазначити, що на даному етапі функціонал розробленого додатку дозволяє використовувати його для створення і заповнення технологічних циклограм мовою програмування ЯПЛК-М. У майбутньому планується покращення візуального дизайну, додавання інших графічних елементів наприклад, таких як меню локалізації, аби дозволити користувачеві зручно змінювати мову інтерфейсу.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було виконано аналіз стану сучасних методів керування та засобів програмування ПЛК паралельної дії, розглянуто технології програмування ППЛК.

Виконано аналіз технологій програмування технічних засобів керування паралельної дії, та визначені їх недоліки, основними з яких є повільність та ресурсовитратність під час використання традиційних технологій розробки. У якості альтернативи прийнято рішення використати TVP технологію мовою ЯПЛК-М, що дозволяє автоматизувати значну кількість процесів створення програмного забезпечення для ПЛК паралельної дії.

Виконано огляд сучасних середовищ і технологій програмування для створення програмного забезпечення для ПК та виконано обґрунтування вибору програмних засобів для створення технології програмування контролерів паралельної дії.

Розроблено архітектуру програмного додатку та виконано його реалізацію. Наведено опис інтерфейсу користувача для заповнення циклограми мовою ЯПЛК-М та проведено тестування програмного продукту.

Таким чином завдання, що поставлені в кваліфікаційній роботі успішно виконані, що дозволяє у значній мірі спростити та оптимізувати технологію програмування контролерів паралельної дії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Андрюшенко, О. А. Электронные программируемые реле серий EASY и MFD-Titan [Текст] / О. А. Андрюшенко, В. А. Водичев – Одесса: Одесский национальный политехнический университет, 2006. — 223 с.

2. Бовчалюк, С. Я. Вдосконалення архітектури, мови програмування та технічної реалізації ПЛК паралельної дії [Текст] / С. Я. Бовчалюк // Збірник міжнародної науково-технічної конференції «ІКТМ-2005». – Харків: ХАІ, 2005. – С. 290.

3. Бура, С. А. Оптимізація технології програмування контролерів паралельної дії [Текст] / С. А. Бура, С. Я. Бовчалюк // Проблеми інформатизації : зб. тез доп. IX міжнар. наук.-практ. конф. том 2, м. Харків, 18-19 листопада 2021 р. Харків, 2021. С. 86.

4. Выжигин, А. Ю. Гибкие производственные системы учеб. Пособие для вузов [Текст] / А. Ю. Выжигин. // 1. – 2009. – С. 286.

5. Галкін, П. В. Програмування ПЛК в CODESYS: навчальний посібник [Текст] / П. В. Галкін, І. І. Ключник // Харків: ФОП Панов А. М., –2019. – 92 с.

6. Зюбин, В. Е. Программирование ПЛК: языки МЭК 61131-3 и возможные альтернативы [Текст] / В. Е. Зюбин // Промышленные АСУ и контроллеры. – 2005. – No 11. – С. 31–35.

7. Мишель, Ж. Программируемые контроллеры: Архитектура и применение: пер. с фр. [Текст] / Ж. Мишель ; Переводчик И. В. Федотов ; Ред. Б. И. Лыткин . – М. : Машиностроение, 1992 . – 320 .

8. Роспутний В. О., Бовчалюк С. Я. Перспективна мова і технологія програмування контролерів паралельної дії [Текст] / В. О. Роспутний, С. Я. Бовчалюк //. Проблеми енергозабезпечення та енергозбереження в АПК України : Науково-практ. студент. конф., м. Харків, 2 квіт. 2020 р. Харків, – 2020. – С. 166.

9. Фурман, И. А. Технологическое визуальное программирование – новое средство автоматизации разработки программного обеспечения ПЛК [Текст] / И. А. Фурман, С. А. Колесников // Інформаційно- керуючі системи на залізничному транспорті. – 2003. – N 4. – С. 46–48.

10. Фурман, И. А. Контроллеры и процессоры с параллельной архитектурой [Текст] / И. А. Фурман, В. А. Краснобаев, М. Л. Малиновский // Учебник для ВУЗов. – Харьков: УкрГАЖТ, 2006. – 416 с.