Центральноукраїнський національний технічний університет
Механіко-технологічний факультет
Кафедра кібербезпеки та програмного забезпечення

”Допущено до захисту”
Завідувач кафедри кібербезпеки
та програмного забезпечення
д.т.н., професор
_____ Олексій СМІРНОВ
“____” _____ 2023 р.

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА
за першим (бакалаврським) рівнем вищої освіти
на тему
“Програмне забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних”

Виконав здобувач вищої освіти
IV курсу, групи КІ-20-3СК
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Ляшенко Є.В.
«____» _____ 2023 р.

Керівник проекту
доктор технічних наук, доцент
_____ Коваленко О.В.
«____» _____ 2023 р.
Рецензент _____
_____

м. Кропивницький

Центральноукраїнський національний технічний університет
Факультет *Механіко-технологічний*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Галузь знань ._12 _____*"*Інформаційні технології*"*_____
Спеціальність *123 "*Комп'ютерна інженерія*"*
Освітньо-професійна (освітньо-наукова) програма _*"*Комп'ютерна інженерія*"*_

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____ Олексій СМІРНОВ
« 17 » січня 2023 року

ЗАВДАННЯ НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ ЗА ПЕРШИМ (БАКАЛАВРСЬКИМ) РІВНЕМ ВИЩОЇ ОСВІТИ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Ляшенку Євгену Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних*

2. Керівник роботи *Коваленко Олександр Володимирович, докт. техн. наук, доцент*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 8-02 від 5.01.2023 року

3. Строк подання студентом роботи до захисту *23.05.2023 р.*

4. Мета та завдання випускної кваліфікаційної роботи*: Метою роботи є розробка програмного забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних*

5. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

6. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

7. Дата видачі завдання « 17 » січня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Строк виконання етапів випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти	Примітка
1.	Аналіз існуючих систем	10.03.2023 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2023 р.	
3.	Розробка моделі компонента	20.03.2023 р.	
4.	Розробка структур даних	25.03.2023 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2023 р.	
6.	Програмування алгоритмів	10.04.2023 р.	
7.	Оформлення ПЗ	17.04.2023 р.	
8.	Попередній захист роботи	23.05.2023 р.	

Дата видачі завдання
« 17 » січня 2023 р.

Підпис керівника
_____ Коваленко О.В.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Ляшенко Є.В.
(прізвище та ініціали)

АНОТАЦІЯ

Ляшенко Є.В. Програмне забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2023.

В даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти розроблено програмне забезпечення, яке призначено для системи візуалізації процесу діагностики паралельного порту передачі даних.

Метою розробки є програмне забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних.

Результат роботи – програмна реалізація системи візуалізації процесу діагностики паралельного порту передачі даних.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows 10/11.

Програму розроблено в середовищі Builder C++.

Ключові слова: комп'ютерна інженерія, діагностика, паралельний порт передачі даних

ABSTRACT

Liashenko E.V. Software of the visualization system of the process of diagnostics of the parallel port of data transmission. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this final qualification work for the first (bachelor) level of higher education, software is developed, which is intended for the visualization system of the process of diagnostics of the parallel data transfer port.

The purpose of the development is the software of the visualization system of the process of diagnostics of the parallel port of data transmission.

The result of the work is the software implementation of the visualization system of the diagnostic process of the parallel data transfer port.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

A convenient user interface has been developed. Instructions for working with software tools are provided.

The program can be used on PCs of IBM PC architecture with Windows 10/11 OS.

The program was developed in the Builder C++ environment.

Keywords: computer engineering, diagnostics, parallel data transfer port

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 5
1.1 Призначення системи..... 5
1.2 Область застосування..... 7
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 10
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти..... 10
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування..... 18
2.3 Розгорнута постановка завдання 19
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 20
3.1 Опис функціонування системи 20
3.2 Розробка структурної схеми..... 28
3.3 Розробка функціональної схеми 30
3.4 Розробка діаграми процесів..... 34
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ..... 36
4.1 Розробка блок-схем та опис алгоритмів функціонування системи..... 36
4.2 Захист розробленого програмного забезпечення..... 48
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 49
6 ОСНОВНІ ВИСНОВКИ..... 53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 55

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ЕОМ	–	електронно-обчислювальна машина
ЛОМ	–	локальна обчислювальна мережа
ОС	–	операційна система
НС	–	надзвичайна ситуація
ПП	–	периферійний пристрій
ТТЛ	–	транзисторно-транзисторна логіка
ЦО	–	цивільна оборона
AES	–	покращений стандарт шифрування
Byte Mode	–	двунаправлений байтний режим
Bi-Di	–	двунаправлений режим паралельного порту
Centronics	–	базовий інтерфейс паралельного порту
DMA	–	прямий доступ до пам'яті
ECP	–	порт із розширеними можливостями
EPP	–	поліпшений паралельний порт
HostBusy	–	сигнал квитування
LPT	–	паралельний порт
Nibble Mode	–	напівбайтний режим введення
PC	–	персональний комп'ютер
SPP	–	стандартний паралельний порт
VCL	–	інтегрована бібліотека візуальних компонентів

ВСТУП

Актуальність теми. Паралельний порт (LPT) – один із самих старих портів комп'ютера, але разом з тим і самий зручний для керування електронними пристроями, тому що сигнали порту постійні й не мають потреби в якихось перетвореннях. Найчастіше через LPT-порт до комп'ютера приєднують принтер, сканер чи модем. Але область застосування цього порту насправді значно ширша. На базі даного порту можна організувати автоматизовану систему з датчиками й робочими елементами або просто керувати будь-яким електронним пристроєм.

З огляду на широку сферу застосування паралельного порту, прогнозування й своєчасне визначення його несправностей є актуальним завданням.

Для роботи на ЕОМ необхідна наявність кваліфікованих спеціалістів у сфері системного програмування. Для їх якісного навчання не достатньо лише вивчення теоретичних основ. Велику роль відіграє практика.

Розроблене програмне забезпечення візуалізації процесу діагностики паралельного порту передачі даних значно полегшить вивчення методів програмування та діагностики паралельного інтерфейсу.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем візуалізації процесу діагностики паралельного порту передачі даних.

– Дослідження системи візуалізації процесу діагностики паралельного порту передачі даних.

– Програмна реалізація системи візуалізації процесу діагностики паралельного порту передачі даних.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі візуалізації процесу діагностики паралельного порту передачі даних.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Система призначена для емуляції процесу діагностики паралельного порту.

Паралельний порт (порт принтера, LPT) – міжнародний стандарт паралельного інтерфейсу для підключення периферійних пристроїв персонального комп'ютера.

В основному використовується для підключення до комп'ютера принтера, сканера та інших зовнішніх пристроїв (часто використовувався для підключення зовнішніх пристроїв зберігання даних), однак може застосовуватися й для інших цілей (організація зв'язку між двома комп'ютерами, підключення механізмів телесигналізації та телеуправління).

Порт на стороні керуючого пристрою (комп'ютера) має 25-контактний 2-рядний роз'єм DB-25-female (IEEE 1284-A), що показаний на рисунку 1.1.

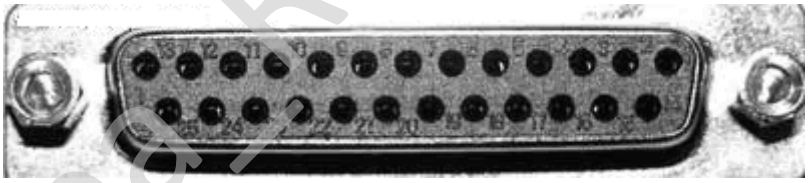

Рисунок 1.1 – 25-контактний роз'єм DB-25, що використовється як LPT-порт на персональних комп'ютерах (IEEE 1284-A)

На периферійних пристроях звичайно використовується 36-контактний роз'єм Centronics (IEEE 1284-B), який зображений на рисунку 1.2.

Тому кабелі для підключення периферійних пристроїв до комп'ютера по паралельному порту звичайно виконуються з 25-контактним роз'ємом DB-25-male на одній стороні й 36-контактним IEEE 1284-B на іншій (AB-кабель). Зрідка

застосовується AC-кабель із 36-контактним роз’ємом MiniCentronics (IEEE 1284-C).

Існують також CC-кабелі з роз’ємами MiniCentronics на обох кінцях, призначені для підключення приладів у стандарті IEEE 1284-II, що застосовується рідко.

Рисунок 1.2 – Кабельний 36-роз’єм Centronics для підключення зовнішнього пристрою (IEEE 1284-B)

Довжина сполучного кабелю не повинна перевищувати 3 метри. Конструкція кабелю: кручені пари в спільному екрані, або кручені пари в індивідуальних екранах. Зрідка використовуються стрічкові кабелі.

Для підключення сканера, і деяких інших пристроїв використовується кабель, у якого замість роз’єма (IEEE 1284-B) встановлений роз’єм DB-25-male. Звичайно сканер оснащується другим інтерфейсом з роз’єм DB-25-female (IEEE 1284-A) для підключення принтера (оскільки звичайно комп'ютер оснащується тільки одним інтерфейсом IEEE 1284). LPT являється самим зручним портом для керування електронними пристроями, тому що сигнали порту постійні й не мають потреби в яких-небудь перетвореннях. На базі даного порту можна організувати автоматизовану систему з датчиками й робочими елементами або просте керування якими-небудь електронними пристроями.

Тому важливо вміти програмувати та діагностувати паралельний порт. Дана програма розроблена для навчання студентів діагностувати неполадки паралельного порту.

1.2 Область застосування

Розроблене програмне забезпечення можна використовувати для навчання студентів спеціальності 123 «Комп'ютерна інженерія» діагностиці паралельного порту передачі даних.

Паралельний порт – це тип інтерфейсу на персональному комп'ютері (ПК), який передає або приймає дані на периферійний пристрій, наприклад принтер. Дані передаються по паралельному кабелю довжиною не більше ніж стандартні 6 футів. Якщо кабель занадто довгий, цілісність даних може бути втрачена. Рекомендація Hewlett-Packard – максимум 10 футів.

Спочатку паралельний порт був односпрямованим і передавав вісім біт даних за раз по кількох жилах мідного кабелю. Він був представлений CentronicsData Computer Corporation у 1970 році. Паралельний порт був розроблений для використання з принтерами та міг передавати лише 300 Кбіт/с. Стандартом для односпрямованого порту принтера був стандартний порт принтера (SPP) або звичайний порт, розроблений у 1981 році. У 1987 році з'явився PS/2, який підключав інші периферійні пристрої, такі як миші та клавіатури. PS/2 був двонаправленим паралельним портом (BPP), який міг одночасно передавати та отримувати вісім бітів даних.

У 1994 році було представлено два нових типи паралельних портів – розширений паралельний порт (EPP) і порт розширених можливостей (ECP). Розширений паралельний порт (EPP) був трохи швидшим, ніж старі паралельні порти, зі швидкістю передачі від 500 Кбіт/с до 2 Мбіт/с. Порт використовується для нових моделей принтерів і сканерів. ECP також підтримує 8-бітний двонаправлений порт. Це як EPP, але використовує прямий доступ до пам'яті (DMA). Він використовується для периферійних пристроїв, які не є принтерами, наприклад мережевих адаптерів або дисководів.

Крім того, у 1994 році було прийнято стандартний метод сигналізації для двонаправленого паралельного периферійного інтерфейсу для персональних

комп’ютерів (IEEE 1284), щоб уникнути проблем несумісності з новим апаратним забезпеченням різноманітних паралельних портів. П'ять режимів роботи були визначені як режим ECP, режим EPP, байтовий режим, режим полубайтів і режим сумісності. Кожен режим повинен підтримувати передачу даних у прямому, зворотному або двонаправленому напрямку. Щоб забезпечити збереження цілісності даних, IEEE 1284 встановив стандарти для роз’єму, інтерфейсу та кабелю.

Паралельний порт передає один біт даних по кожному з двох проводів, що збільшує швидкість передачі даних (DTR). Як правило, існують додаткові дроти, що регулюють сигнали, щоб вказати, коли доступна передача або отримання даних.

Спочатку паралельні порти призначалися для принтерів. Перший порт паралельного інтерфейсу для принтерів був створений для Centronics Model 101 (випущений у 1970 році), який передавав дані по вісім біт за раз. Цей паралельний порт міг лише передавати дані, але не приймати їх. Пізніше паралельний порт став двонаправленим і використовувався як для пристроїв введення, так і для принтерів. Двонаправлений паралельний порт (BPP) може обмінюватися даними з декількома периферійними пристроями, такими як сканери, zip-накопичувачі, жорсткі диски, модеми та приводи CD-ROM. BPP зазвичай використовується для швидкої передачі даних на невеликі відстані. Додаткові паралельні порти зазвичай позначаються LPT1, LPT2 тощо.

Коли в 1994 році був представлений стандарт IEEE 1284, довжина кабелів, логічні напруги та інтерфейси були стандартизовані. У стандарті IEEE 1284 було визначено п’ять режимів роботи для підтримки передачі даних у прямому, зворотному або двонаправленому напрямку. П’ять режимів роботи: порт розширених можливостей (режим ECP), режим розширеного паралельного порту (EPP), байтовий режим, режим полубайтів і режим сумісності (стандартний паралельний порт або SPP).

Сумісність односпрямована і використовується переважно для принтерів. Режим полубайта є двонаправленим, що дозволяє передавати чотири послідовні біти за допомогою однієї лінії даних. Він використовується для покращеного стану принтера, що дозволяє пристрою передавати дані чотири біти за раз. Байтовий режим є двонаправленим, у якому дані передаються вісім біт за раз, використовуючи одну лінію даних. Режим EPP має 8-бітний двонаправлений інтерфейс, який передає дані зі швидкістю від 500 Кбіт/с до 2 МБ/с. Режим ECP має 8-бітний двонаправлений інтерфейс, який використовує DMA і може забезпечити пропускну здатність до 2,5 МБ/с.

Сьогодні універсальна послідовна шина (USB) замінила паралельний порт. Насправді кілька виробників повністю виключили паралельний інтерфейс. Хоча для старих персональних комп'ютерів (ПК) і ноутбуків доступний USB-паралельний адаптер для паралельних принтерів або інших периферійних пристроїв, що мають паралельний інтерфейс. Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних, є актуальною задачею, яка потребує вирішення у даній випускній кваліфікаційній роботі за першим (бакалаврським) рівнем вищої освіти.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти

Для роботи з паралельним портом та його діагностики існує багато програм, але серед них немає емуляторів. Всі вони працюють безпосередньо з самим портом.

Найвідоміша зарубіжна програма, що ілюструє роботу з LPT-портом та дозволяє на основі прикладів розробити власний драйвер паралельного порту – це RapidDriver, а конкретніше, її модуль RapidLPT (рисунок 2.1).

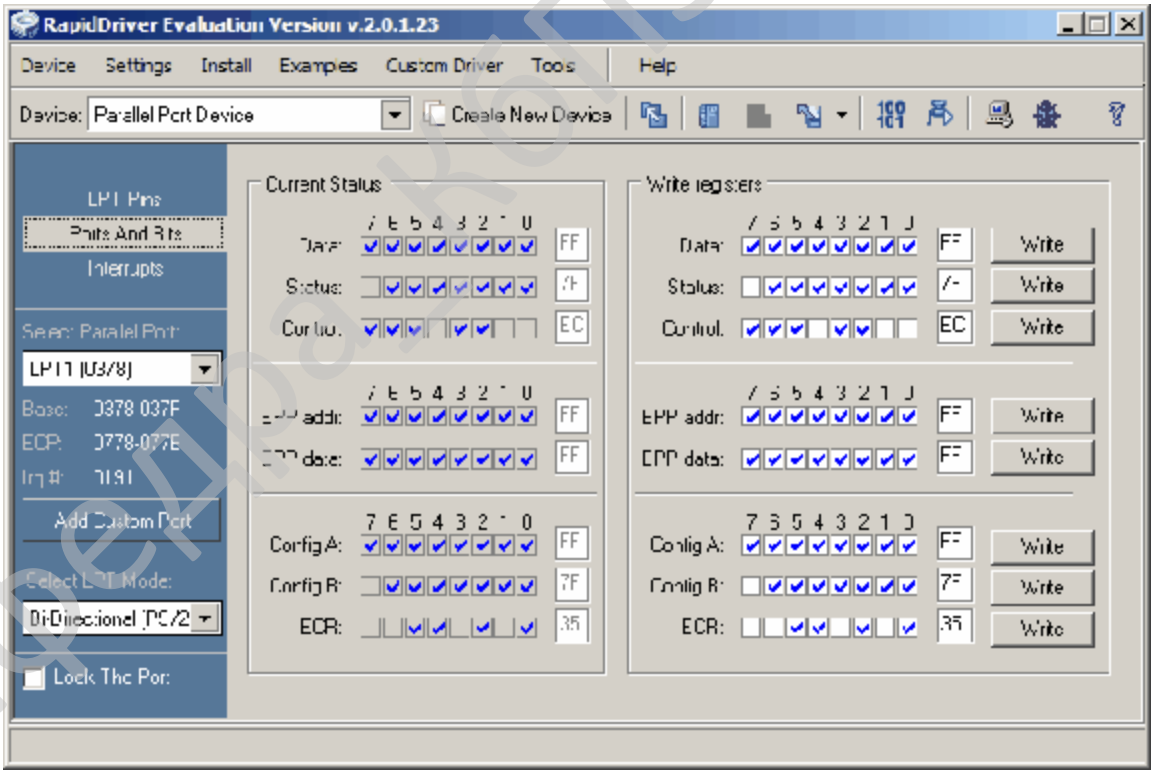

Рисунок 2.1 – RapidLPT (модуль програми RapidDriver для роботи з послідовним портом)

RapidDriver – інструмент для розробки драйверів і налагодження апаратури. Достатньо лише запустити RapidDriver, створити новий проект, вибрати один з пристроїв: LPT, USB, ISA або PCI зі списку розпізнаних PnP-пристроїв – і Ви вже повністю готові для роботи з ним.

Програма дозволяє управляти пристроєм безпосередньо з RapidDriver, або написати власну програму керування. RapidDriver включає багато прикладів програм, написаних з використанням MSVC/C++, Delphi, C++Builder, VB, VB.NET і MS C#.

Для більш «тонкого» дослідження поводження пристрою, RapidDriver включає вбудований відлагоджувач, що працює з командного рядка, а також дозволяє писати програми на C, Pascal, або Basic-подібній мові.

Програма RapidDriver має виключно англомовний інтерфейс і є платною. На сьогоднішній день модуль для програмування паралельного порту RapidLPT з відкритим програмним кодом коштує 999 доларів.

LPT 3D hard analyzer v1.0 призначена для зняття цифрових сигналів та протоколів працюючих пристроїв, що використовують чи підключаються до паралельного порту (LPT 1-3) комп'ютера. Програма призначена для роботи в ОС Windows 9x/2000/XP.

Цю програму рекомендується застосовувати при знятті закодованих сигналів інфрачервоних систем охоронної сигналізації, інфрачервоних кодових замків, найбільш сучасних пультів дистанційного керування, коли швидкість аналізу має вирішальне значення.

Модуль керування портами заснований на ядрі коду програми «XP LPT» (рисунок 2.2), що використовує драйвер введення-виведення LPTWDMIO і має відповідні можливості:

1. Автоматична реєстрація драйвера в Windows XP на правах адміністратора системи.

2. Автоматична перевірка встановлених портів і внесення в список.

3. Одночасне читання регістрів даних, контролю й стану вибраного LPT-

порту.

4. Одночасне відображення вмісту регістрів у форматах Hex і Byte, що рятує від необхідності перерахування.

5. Запис даних у порт у форматах Hex і Byte.

6. Відображення логічного (1/0) стану бітів (Pin) обраного порту в реальний період часу.

7. Керування логічним станом бітів (Pin) LPT-порту за допомогою Pin клавіатури.

8. Перевірка порту на двунаправленість (якщо ввімкнено в BIOS).

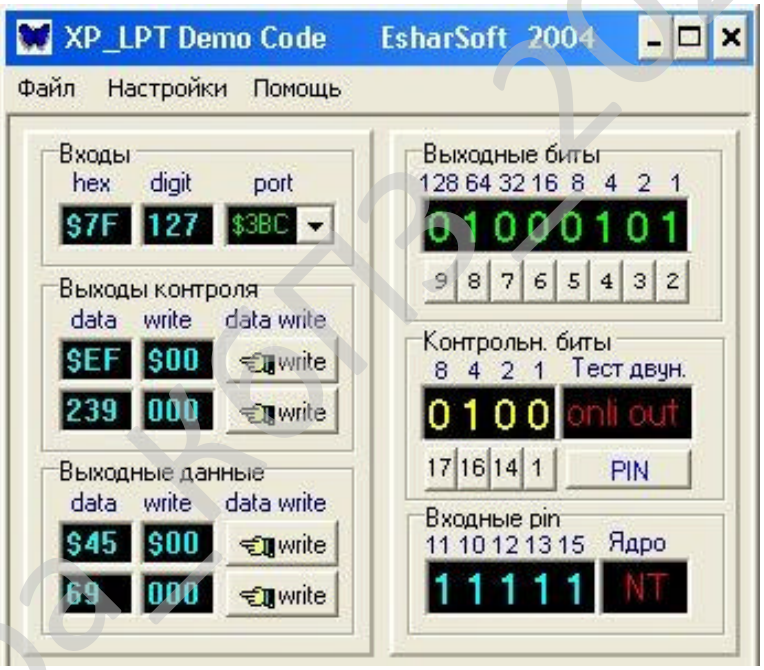

Рисунок 2.2 – Програма «XP LPT», призначена для керування LPT-портами комп'ютера з ОС Windows 9x/2000/XP

PinRegistrator – програма спостереження й реєстрації логічного стану бітів LPT-порту (рисунок 2.3). Написана під Windows 95/98/ME і має наступні можливості:

1. Спостереження за логічним станом кожного з 17 бітів (pin) LPT-порту на вибір і ведення відповідного LOG-файлу зі збереженням індивідуальних записів і

настроювань для кожного обраного біта.

2. Автоматичне створення LOG-файлу в каталозі програми за поточну добу й збереження всіх попередніх LOG-ів роботи, у яких фіксується:

- дата й час запуску програми;
- номер порту й обраного біта LPT;
- час старту процесу реєстрації (перхід високого логічного стану біта (pin)=5 в у низький (1/0) за допомогою підключених зовнішніх датчиків (пристроїв) або вручну з панелі керування програми;
- загальний звіт роботи за минулі місяці, дні, години, хвилини, секунди;
- поточний звіт роботи;
- відстеження змін бітів порту.

3. Перегляд LOG-файлів.

4. Прихований фоновий режим роботи, з можливістю відключення реєстрації у вікні зняття процесів Windows (CTRL+ALT+DEL), вихід зі прихованого режиму по таймеру.

5. Візуальний моніторинг логічного стану бітів за допомогою панелі світлодіодів.

6. Можливість запуску багатьох копій програми й ведення відповідних LOG-ів, попередньо скопіювавши програму PinRegistrator.exe в окрему папку.

Розглянемо режими роботи програми.

Режим "Тест". Для запуску натисніть кнопку "Настроювання таймерів", виберіть меню "Тест" і поставте прапорець "Працювати в тестовому режимі". У цьому режимі відлік часу відбувається за 0,001с, замість 1с, що дуже зручно для перегляду роботи програми. Для зручності настроювання виберіть кожний з вихідних бітів, встановивши номери Pin відповідно до документації LPT-порту. За замовчуванням це п'ять входів, дванадцять виходів.

Режим "Вручну". Вмикається відповідною кнопкою. За допомогою нього можна перевести обраний вихідний біт (але не вхідний) у стан логічного нуля (низького рівня, 0В) з логічної одиниці (високого рівня, 5В) або навпаки.

Реєстратор запускається по низькому рівні, при цьому світлодіод обраного біта (Pin) гасне. Це вибрано не випадково, тому що за замовчуванням всі вхідні біти мають високий рівень і коли підключений зовнішній пристрій переводить їх у низький – запускається реєстратор (відлік часу). Крім того, це зручніше і безпечніше з погляду сполучення комп'ютера (LPT-порту) і зовнішніми пристроями, тому що не вимагає подачі керуючої напруги 5 Вольт і можна обійтися навіть найпростішою схемою керування з повною розв'язкою: коли від пристрою спрацьовує малопотужне реле, своїми контактами приєднуючи резистор 300-500Ом до обраного біта вхідного регістра, переводячи його в логічний нуль.

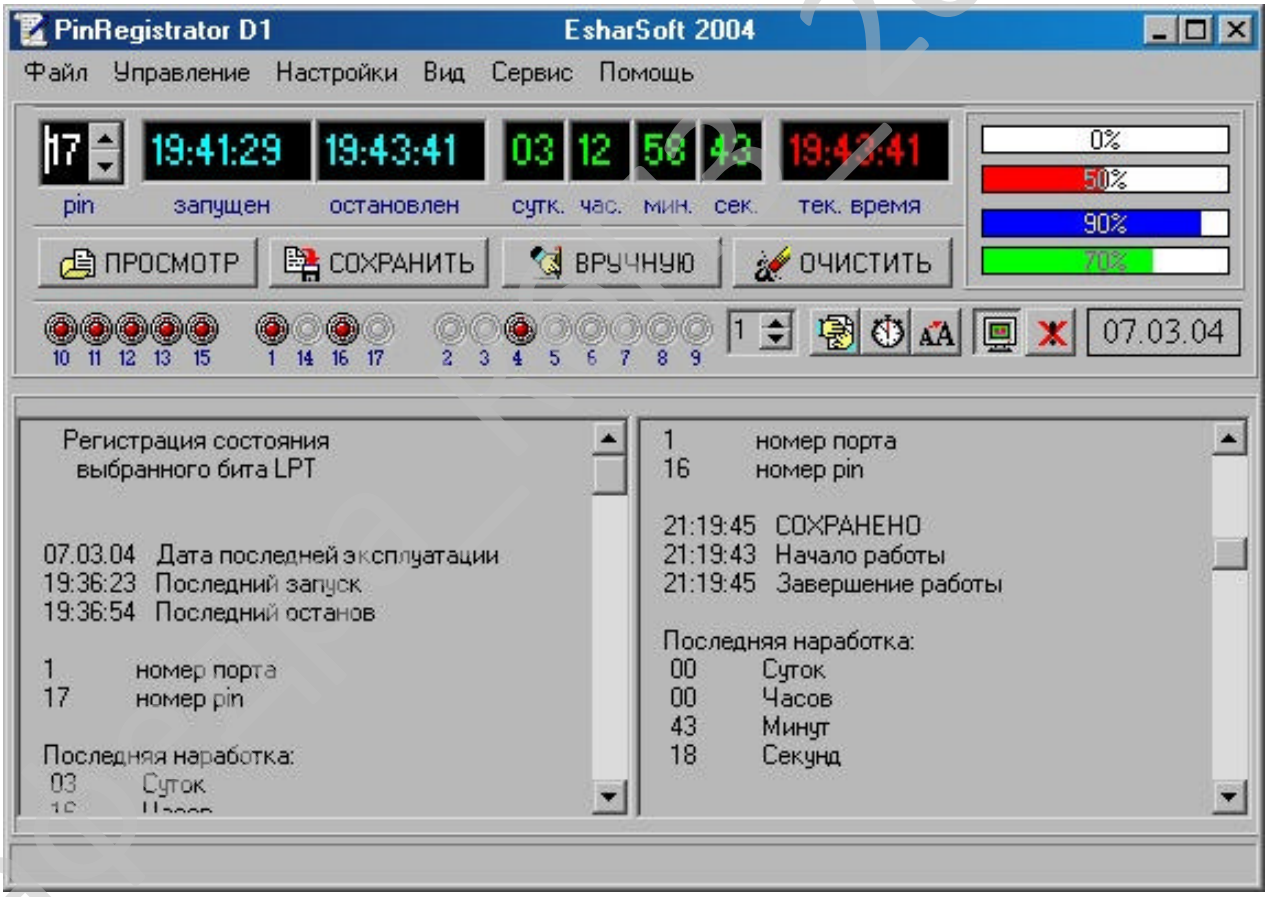

Рисунок 2.3 – PinRegistrator, програма для спостереження й реєстрації логічного стану бітів LPT-порту

Використання паралельних портів

Існує кілька варіантів виконань принтерних портів у комп'ютерах:

1. SPP (стандартний паралельний порт).
2. EPP (поліпшений паралельний порт).
3. ECP (порт із розширеними можливостями).

Найпоширенішим застосуванням LPT-порту є, звичайно, підключення принтера. Практично всі принтери можуть працювати з портом у режимі SPP, але застосування розширених режимів дає додаткові переваги:

Двунаправлений режим (Bi-Di) дає додаткові можливості для повідомлення стану й параметрів принтера.

Швидкісні режими (Fast Centronics) істотно підвищують продуктивність практично будь-якого принтера (особливо лазерного), але можуть потребувати більш якісного кабелю.

Режим ECP потенційно найефективніший, і він має системну підтримку у всіх варіантах Windows.

З розповсюджених сімейств ECP підтримують принтери HP DeskJet моделей BXX, LaserJet починаючи з 4-го, сучасні моделі фірми Lexmark вимагають застосування кабелю по частотних властивостях відповідного IEEE 1284.

Найпростіший варіант кабелю підключення принтера – 18-дротовий кабель із неперевитими проводами з успіхом може використовуватися для роботи порту в режимі SPP.

Ідеальним варіантом є кабелі, у яких всі сигнальні лінії перевиті із спільними проводами й укладені в спільний екран – те, що вимагає IEEE 1248. Такі кабелі гарантовано працюють на швидкостях до 2 Мбайт/с, і допускається їхня довжина до 10 метрів.

У таблиці 2.1 приводиться розпаювання кабелю підключення принтера з роз'ємом XI типу A (DB-25P) з боку PC і X2 типу B (Centronics-36) або типу C (мініатюрний) з боку принтера.

Таблиця 2.1 – Кабель підключення принтера

XI, роз'єм PC типу A	Сигнал	X2, роз'єм PRN типу B	X2, роз'єм PRN типу З
1	Strobe	1	15
2	Data0	2	6
3	Data1	3	7
4	Data2	4	8
5	Data3	5	9
6	Data4	6	10
7	Data5	7	11
8	Data6	8	12
9	Data7	9	13
10	Ack	10	3
11	Busy	11	1
12	PaperEnd	12	5
13	Select	13	2
14	Auto LF	14	17
15	Error	32	4
16	Imt	31	14
17	Sict In	36	16
18	GND(l)	19	33
19	GND(2 3)	20 21	24 25
20	GND(4 5)	22 23	26 27
21	GND(6 7)	24 25	28 29
22	GND(8 9)	26 27	30 31
23	GND(11 15)	29	19 22
24	GND(10 12 13)	28	20 21 23
25	GND(14 16 17)	30	32 34 35

Високошвидкісний зв'язок двох комп'ютерів може виконуватися й у режимі ECP (режим EPP для цих цілей незручний, оскільки він вимагає синхронізації шинних циклів введення/виведення двох комп'ютерів). У таблиці 2.2 наведено розпаювання кабелю для цього режиму. У ній як допоміжна інформація наведені імена сигналів, які апаратно генеруються адаптерами портів. Цей же кабель може використовуватися й для зв'язку в режимі Byte Mode (при наявності двунаправлених портів).

Таблиця 2.2 – Кабель зв'язку PC-PC у режимі ECP і Byte Mode

Роз'єм XI		Роз'єм X2	
Контакт	Ім'я в ECP	Ім'я в ECP	Контакт
1	HostClk	PeriphClk	10
14	HostAck	PeriphAck	11
17	1284Active	PeriphRequest	15
16	Reverse Request	AckReverse	12
10	PeriphClk	HostClk	1
11	PeriphAck	HostAck	14
12	AckReverse	ReverseRequest	16
13	Xflag	-	-
15	PeriphRequestf	284Active	17
2-9	Data[0:7]	Data[0:7]	2-9

Підключення сканера до LPT-порту ефективне, тільки якщо порт забезпечує хоча б двунаправлений режим, оскільки в основному тут використовується введення. Але краще використовувати порт ECP, якщо цей режим підтримується сканером.

Підключення зовнішніх накопичувачів (lomega Zip Drive, CD-ROM), адаптерів ЛОМ та інших симетричних пристроїв введення/виведення має спільну специфіку. Більшість таких пристроїв здатні працювати в кожному з режимів порту (крім ECP), що забезпечує їхнє необмежене застосування на будь-яких PC.

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Оскільки потрібно розробити просту та легку у користуванні програму, яка б виконувалась під операційною системою Windows, то для її реалізації я обрав Builder C++. Існує велике число бібліотек написаних під Builder C++ , тому це одна з важливих причин вибору мови програмування. Середовище Builder C++ досить просте в користуванні, його вихідний код значно менше по об'єму в порівнянні з Delphi чи деякими іншими програмами такого типу. Досить легко організувати взаємодію між модулями програм, об'єктно-орієнтований підхід дає можливість значно скоротити код програми, а отже і час його виконання.

На заміну старого розробленого набору елементів управління у Builder C++ інтегрована бібліотека візуальних компонентів VCL, представлених на палітрі компонентів. Після переносу на форму методом перетягування (drag-and-drop) компоненти відразу становляться діючими об'єктами вашої програми. Окрім типізованих інтерфейсних елементів Windows (кнопки, смуги прокручування, редагуємі текстові області, прості та комбіновані списки, та інше) у бібліотеку включені елементи підтримки діалогових вікон, обслуговування баз даних та багато іншого. Можливо не тільки модифікувати поведінку існуючих компонентів, але і будувати нові.

Builder C++ підтримує останні розширення стандарту мови C++ та забезпечує швидку компіляцію та складання 32-розрядних програм для Windows. Результуючі програми оптимізовані з точки зору швидкості виконання програм та затрат пам'яті. Зручний відладгоджувальник (з асемблерним вікном, можливістю крокового виконання, завдання точок зупинки, трасування та інше) повністю інтегрований у систему проектування. Дизайнер форм, редактор коду, інспектор об'єктів та інші інструменти зостаються доступними під час виконання програми, саме через це вносити зміни до коду можна прямо у процесі відлагодження.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, реалізації підлягає програмне забезпечення, яке призначено для системи візуалізації процесу діагностики паралельного порту передачі даних.

В процесі розробки випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Різниця між послідовними і паралельними портами

У комп’ютерній системі порт забезпечує інтерфейс для підключення периферійного пристрою до комп’ютерної системи. Таким чином, порти в апаратному забезпеченні комп’ютера – це гнізда або роз’єми, через які периферійні пристрої підключаються до комп’ютерної системи. Ці порти стандартизовані для кожної мети. У комп’ютерній системі є кілька типів портів, наприклад порти USB, порти Ethernet, порти дисплея, послідовні порти, паралельні порти тощо. Через ці порти ми під’єднуємо різні апаратні пристрої, такі як монітор, клавіатура, миша тощо. комп'ютер.

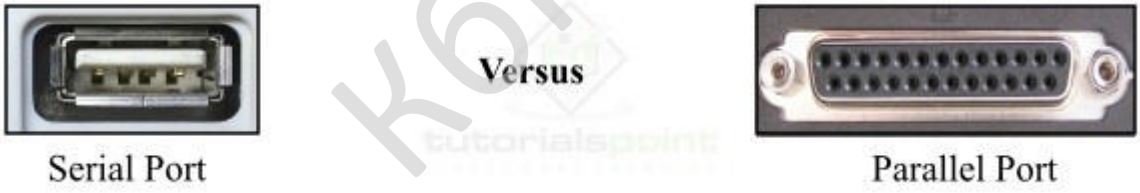

Рисунок 3.1 – Послідовні та паралельні порти та чим вони відрізняються один від одного

Що таке послідовні порти?

Послідовні порти забезпечують інтерфейс для підключення послідовних ліній для підготовки послідовного зв’язку. Послідовні порти – це типи комп’ютерних портів, через які біти даних передаються як єдиний потік двійкових 0 і 1 у формі електричних сигналів. Послідовні порти забезпечують лише один шлях передачі, який може бути одним проводом, парою проводів або одним каналом у разі бездротового зв’язку.

Послідовні порти – це найстаріші комунікаційні інтерфейси, які в основному використовуються для підключення принтерів і модемів до комп'ютерної системи. Але в сучасних комп'ютерах послідовні порти використовуються для підключення сучасних пристроїв, таких як монітори з плоским екраном, камери безпеки, GPS-пристрої тощо. Послідовні порти іноді також називають **COM-портами** (або **портами зв'язку**).

Послідовний порт використовує роз'єм DB-9, 9-контактний D-подібний роз'єм, який під'єднується до лінії передачі. Послідовний порт забезпечує послідовний зв'язок за допомогою однієї лінії і, отже, не залежить від швидкості іншого дроту, а його довжину можна збільшити відповідно до потреби.

Що таке паралельні порти?

Паралельний порт – це інший тип комп'ютерного порту для підключення периферійного пристрою до комп'ютерної системи. Як випливає з назви, паралельний порт може передавати кілька бітів даних одночасно. Таким чином, у випадку паралельних портів швидкість передачі даних є відносно високою порівняно з послідовними портами, оскільки вони передають дані без будь-яких затримок.

Паралельні порти в основному використовуються для підключення тих периферійних пристроїв комп'ютера, яким потрібна висока пропускна здатність. Найпоширенішими прикладами таких пристроїв є принтери, монітори, проектори тощо.

Паралельні порти забезпечують інтерфейс для підключення кількох ліній для підготовки паралельного зв'язку для надсилання великих даних за раз. Паралельні порти використовуються для підключення принтерів, жорстких дисків, компакт-дисків тощо. Швидкість усіх ліній має бути однаковою, щоб уникнути помилок і перехресних перешкод. Щоб уникнути таких проблем, дроти мають малу довжину. У паралельному порту використовується роз'єм D-25, 25-контактний роз'єм D-подібної форми, який під'єднується до проводів передачі.

Різниця між послідовними і паралельними портами

Таблиця 3.1 – Відмінності між послідовними портами та паралельними портами

Параметр	**Послідовні порти**	**Паралельні порти**
Визначення	Послідовний порт – це порт, який передає дані за раз одним потоком тривалістю 0 і 1 с.	Паралельний порт – це порт, який передає декілька бітів даних одночасно.
призначення	Послідовний порт використовується для послідовної передачі даних.	Паралельний порт використовується для паралельної передачі даних.
Швидкість передачі	Швидкість передачі через послідовний порт низька порівняно з паралельним портом.	Швидкість передачі паралельного порту досить висока порівняно з послідовним портом.
Кількість бітів даних	Послідовний порт може передавати один біт за раз по одному проводу.	Паралельний порт може передавати набір бітів даних (наприклад, 8-біт, 16-біт) одночасно через окремі дроти.
Пропускна здатність	Послідовні порти використовуються для підключення тих периферійних пристроїв, яким потрібна низька пропускна здатність.	Паралельні порти використовуються для підключення тих пристроїв, яким потрібна відносно висока пропускна здатність.
Роз'єм	Послідовний порт використовує роз’єм DB-9, 9-контактний D-подібний роз’єм, який під’єднується до лінії передачі.	У паралельному порту використовується роз’єм D-25, 25-контактний роз’єм D-подібної форми, який під’єднується до проводів передачі.

Продовження таблиці 3.1

Надмірність	Модель Bottom-Up краще підходить, оскільки вона забезпечує мінімальну надлишковість даних і фокусується на повторному використанні.	Модель «зверху вниз» має високий коефіцієнт надмірності зі збільшенням розміру проекту.
Кількість проводів	Дротове підключення до послідовного порту є менш тихим порівняно з паралельним портом.	Кількість дротів, підключених до паралельного порту, досить висока порівняно з послідовним портом.
Можливість	Послідовний порт здатний передавати один потік даних за раз.	Паралельний порт може передавати кілька потоків даних одночасно.
Механізм передачі даних	Послідовний порт надсилає дані побітово після надсилання побітово.	Паралельний порт надсилає дані шляхом надсилання кількох бітів паралельним способом.
Тип порту	Послідовний порт використовує порти Male.	Для паралельного порту використовуються роз'єми.
Додатки	Модеми, камери безпеки, контролери пристроїв використовують послідовні порти.	Принтери, жорсткі диски, приводи компакт-дисків використовують паралельні порти.

У комп’ютерній системі доступні кілька типів портів для підключення різних типів периферійних пристроїв. Послідовні та паралельні порти є найбільш часто використовуваними портами комп’ютера. Найсуттєвіша відмінність між послідовними портами та паралельними портами полягає в тому, що послідовний

порт передає один біт даних за раз, тоді як паралельний порт передає кілька бітів даних за раз. Завдяки цьому швидкість передачі даних через паралельний порт вище, ніж у послідовного.

Паралельний порт

Модуль Parallel Port дозволяє використовувати LaboratoryBenchComputer для керування контактами паралельного порту на основі змінних LaboratoryBenchComputer. Нижче наведено короткий опис контактів паралельного порту та їх можливого використання.

Паралельний порт – це недорогий спосіб зв'язку із зовнішніми пристроями TTL 0–5 В за допомогою паралельного порту ПК. ЗВЕРНІТЬ УВАГУ: якщо ви експериментуєте з новими схемами, ви можете придбати зовнішній паралельний порт на випадок, якщо ваша проводка може бути неправильною. Замінити зовнішній паралельний порт набагато дешевше, ніж материнську плату!

Виводи паралельного порту поділяються на три групи контактів для цілей відмінності. Три набори: DATA (вхід/вихід), CONTROL (вихід) і STATUS (вхід). Виводи DATA призначені для введення та виведення даних (зазвичай для надсилання на принтер інформації для друку). Порт CONTROL призначений для виведення керуючої інформації на принтер, а штифти STATUS були вхідними сигналами назад від принтера до ПК, призначені для передачі інформації про те, як «закінчився папір» тощо.

Кожен пін використовується буквально для передачі 1 біта інформації. Пін – це або «0», або «1». Усі контакти можуть передавати інформацію одночасно паралельно або послідовно (один за одним), як це робиться через послідовний порт. Стандартний паралельний порт здатний передавати від 50 до 100 кілобайт даних на секунду.

Рівень напруги на контакті становить близько 5 вольт, і його можна використовувати для безпосереднього керування світлодіодом. Щоб перевірити свій паралельний порт за допомогою LaboratoryBenchComputer, ви можете

налаштувати свій модуль паралельного порту, як показано в інтерфейсі нижче, під'єднати один кінець світлодіода до контакту 3 (data2), а інший – до контакту 25 (земля). Коли кількість зображень збільшується, світлодіод має блимати та вимикатися.

Паралельний порт може виводити/надсилати 8 біт за раз і вводити 5 біт. На більш сучасних портах стандартного паралельного порту (SPP) 8 контактів даних можуть стати входами, якщо біт DIRECTION на порту CONTROL має високий рівень. Дивіться нижче, як змінити цей біт.

Існує кілька версій паралельних портів, відмінних від оригінального дизайну. Розширений паралельний порт (EPP) був створений Intel, Xircom і Zenith у 1991 році. EPP дозволяє передавати значно більше даних, від 500 кілобайт до 2 мегабайт, щосекундно. Стандартний паралельний порт (SPP) і повністю замінив оригінальний дизайн. Двонаправлений зв'язок дозволяє кожному пристрою отримувати дані, а також передавати їх. Виводи з 18 по 25, які спочатку використовувалися лише як заземлення, також можна використовувати як виводи даних. Це забезпечує повнодуплексний (в обох напрямках одночасно) зв'язок. Порт розширених можливостей (ECP) забезпечує двонаправлений зв'язок.

Майте на увазі, що початковим призначенням паралельного порту було з'єднання ПК з комп'ютером. Отже, ім'я шпильки відображатиме цю конкретну мету. Якщо ви використовуєте паралельний порт для з'єднання з вашими власними пристроями, головною якістю контактів, яка буде важливою для вас, буде те, чи є вони вхідними чи вихідними. Тобто, якщо ви використовуєте висновок 11 «Busy» для введення деяких даних із вашої схеми, це не повинно відображати, що ваша схема «зайнята».

Більшість значень бітів логічно включені, коли рівень напруги високий, але 4 біти міняються місцями.

Таблиця 3.2 – Розпіновка

Pin #	**Ім'я**	**Введення-виведення**	**біт**	**Перевернутий**
1	nСтроб	Вийти	Контроль-0	Так
2	Дані0	In out	Дані-0	Немає
3	Дані1	In out	Дані-1	Немає
4	Дані2	In out	Дані-2	Немає
5	Дані3	In out	Дані-3	Немає
6	Дані4	In out	Дані-4	Немає
7	Дані5	In out	Дані-5	Немає
8	Дані6	In out	Дані-6	Немає
9	Дані7	In out	Дані-7	Немає
10	nAck	в	Статус-6	Немає
11	Зайняте	в	Статус-7	Так
12	Папір закінчився	в	Статус-5	Немає
13	Виберіть	в	Статус-4	Немає
14	Перехід рядка	Вийти	Контроль-1	Так
15	nПомилка	в	Статус-3	Немає
16	nІніціалізувати	Вийти	Контроль-2	Немає
17	nSelect-Printer	Вийти	Контроль-3	Так
18	Земля	Вийти	(Контроль-4)	Немає
19	Увімкнути двонаправлений порт	Вийти	(Контроль-5)	Немає
20-25	Земля	-	-	-

Зауважте, що ОС забороняє безпосереднє підключення до паралельного порту за допомогою Windows 10/11 тощо. Вам потрібен драйвер системного пристрою для взаємодії з паралельним портом. Для цього LaboratoryBenchComputer використовує популярний inpout32.dll від LOGIX4U.

Інтерфейс

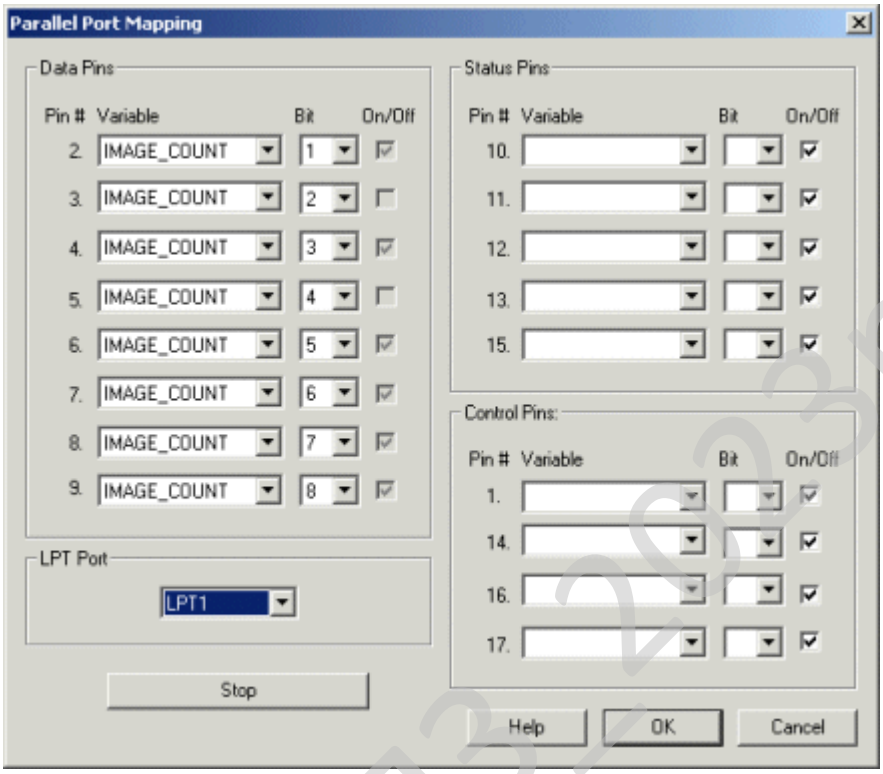

Рисунок 3.1 – Інтерфейс

Інструкції

1. Укажіть, які змінні мають відповідати яким бітам на паралельному порту. Якщо жодна змінна не вибрана, ви можете вручну встановити прапорець для кожного штифта. Зміна прапорця повинна змінити відповідний контакт на паралельному порту. Якщо вибрано змінну, вам потрібно буде вибрати, який біт у значенні змінної представлятиме значення піна. Наприклад, якщо значення змінної дорівнює 2, а вибраний біт – 1, тоді відповідним виводом буде 0. Якщо вибраний біт дорівнює 2, відповідним виводом буде 1. Таким чином, щоб надіслати 8-бітове число через паралельний порт, ви повинні вибрати біти, як показано на зображенні вище інтерфейсу.

2. Виберіть номер паралельного порту. Типовим є LPT1

Зауважте, що на деяких машинах адресний простір паралельного порту

відрізняється. У LaboratoryBenchComputer наступна адреса використовується у співвідношенні зі спадним меню LPTx. Якщо конфігурація вашого комп'ютера інша, вам, можливо, доведеться вказати інший паралельний порт у спадному меню, ніж той, який ви вважаєте вірним, щоб указати правильний адресний простір. Зверніть увагу, що LaboratoryBenchComputer приймає конфігурацію за замовчуванням. Віртуальні паралельні порти тощо змінять базову конфігурацію.

lpt1Data = 0x378

lpt1Status = 0x379

lpt1Control = 0x37a

lpt2Data = 0x278

lpt2Status = 0x279

lpt2Control = 0x27a

lpt3Data = 0x3bc

lpt3Status = 0x3bd

lpt3Control = 0x3be

Щоб перевірити адресу LPT1, яку використовує ваш комп'ютер, клацніть правою кнопкою миші «Мій комп'ютер», виберіть «Керувати», клацніть піддерево «Диспетчер пристроїв», потім розгорніть «Порти (COM і LPT), клацніть правою кнопкою миші порт принтера (LPT1).) і виберіть «Властивості». Адреса вводу/виводу знаходиться на вкладці «Ресурси». Зверніть увагу лише на першу адресу. Зіставте цю адресу з адресою вище та виберіть відповідний LPT1-3 зі спадного меню.

3.2 Розробка структурної схеми

Структурна схема системи зображена на рисунку 3.2. Стандартний паралельний порт (SPP) використовує електричні сигнали ТТЛ-рівня. Для зв'язку з зовнішніми пристроями через LPT порт використовуються різного роду адаптери, що дозволяють за допомогою паралельного порту сформувати шину

адреси й шину даних для подальшого керування яким-небудь електронним модулем або приладом.

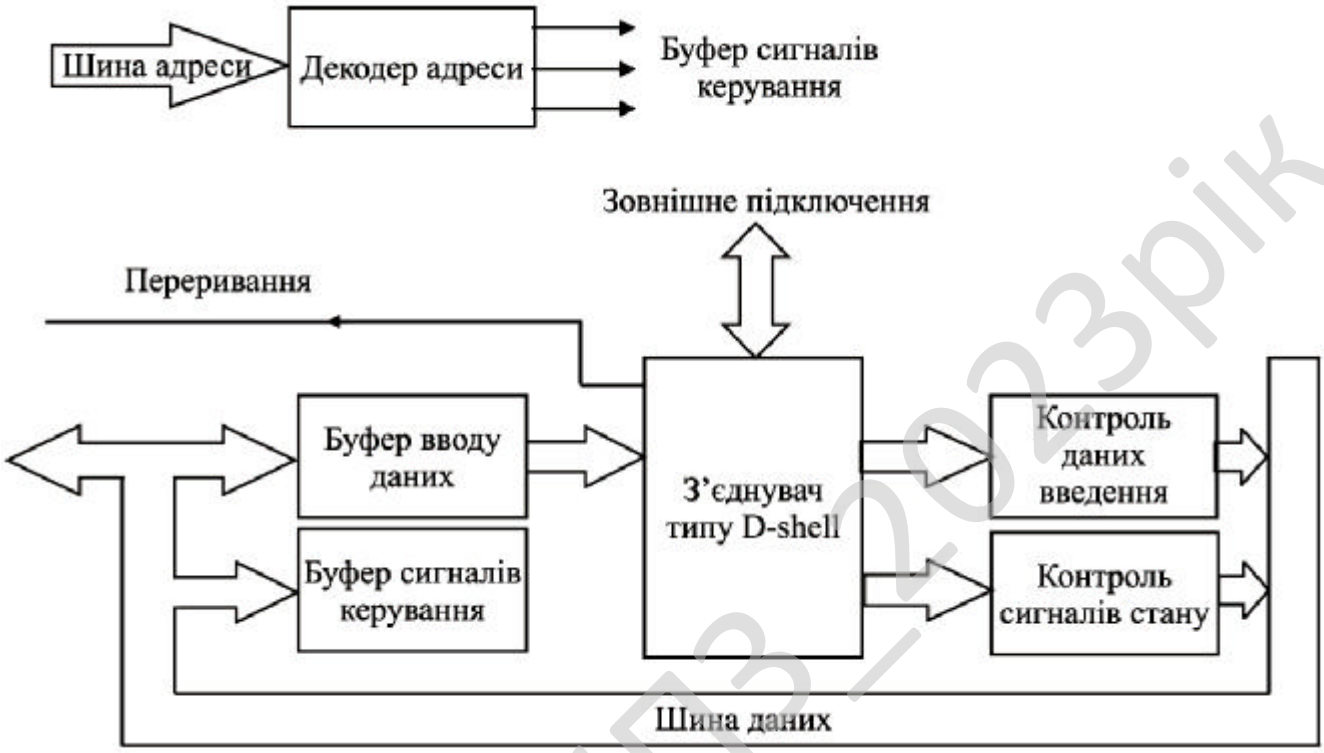

Рисунок 3.2 – Структурна схема розробленої системи

Приєднання кабеля до адаптера паралельного інтерфейсу здійснюється через 25-контактний роз'єм типу D-shell (DB-25).

З шини даних інформація потрапляє до буфера вводу даних та буфера сигналів керування. Програма здійснює контроль даних, що вводяться та контроль сигналів стану.

Широкого розповсюдження здобули паралельні адаптери, в яких більшість функцій окремих ТТЛ-мікросхем об'єднані в одну, типу 82С11, що виконані по КМОП-технології.

Для керування пристроями, що споживають малий струм необхідність у схемі сполучення відпадає, тому що потужність порту може забезпечити нормальну роботу таких пристроїв. Такими є наприклад світлодіоди, електронні

наручні годинники, чорно-білі РК дисплеї, керуючі входи приладів. Виключенням є використання пристроїв, які навантажують порт до межі. Такі пристрої краще підключати через резистори для відводу тепла й запобігання порту (основне навантаження буде доводиться саме на ці резистори, а не на резистори порту), при відсутності цих резисторів Ви наражаєте на небезпеку сам порт. Також вони обмежують споживання струму в ланцюзі. Опір резисторів – у діапазоні 470Ом – 2кОм (залежно від того, що буде підключатися).

3.3 Розробка функціональної схеми

Функціональна схема системи зображена на рисунку 3.6.

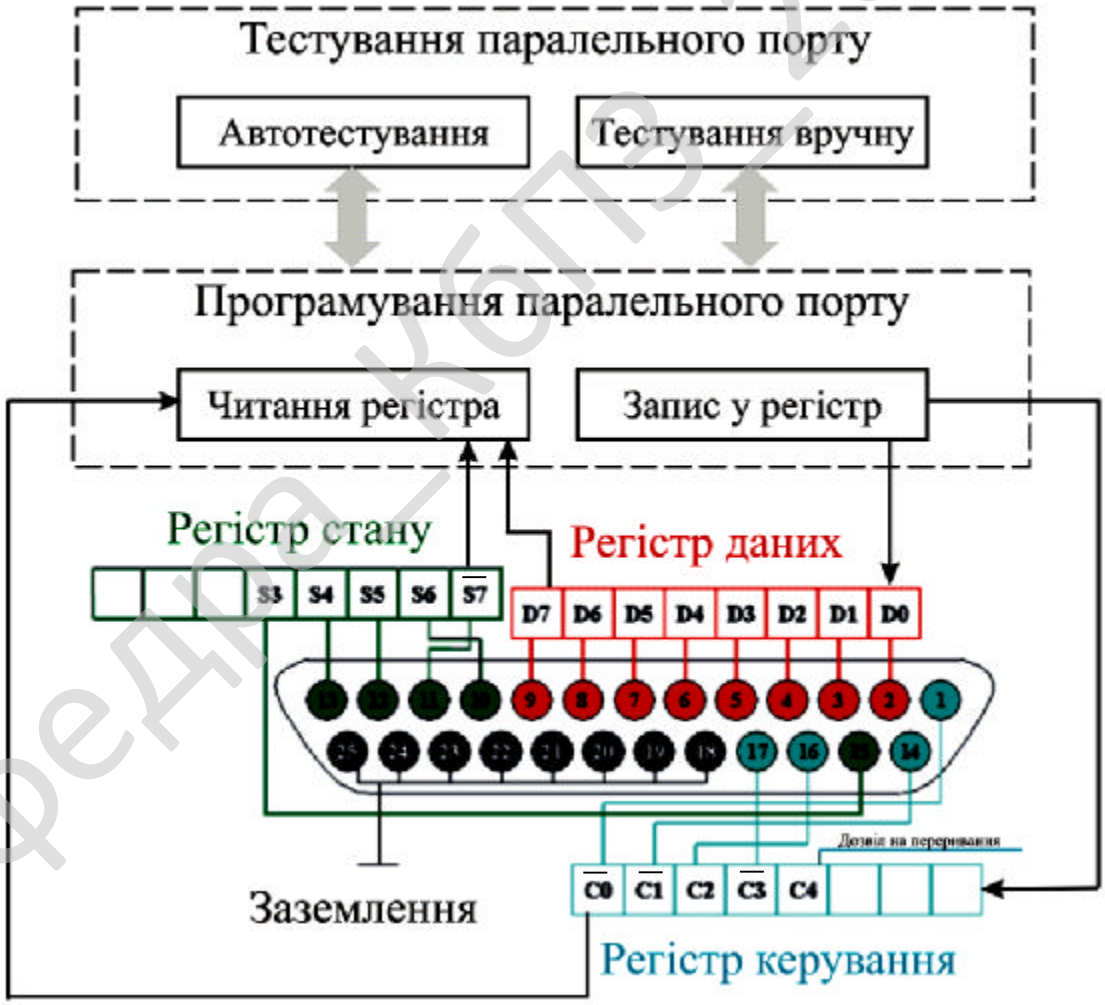

Рисунок 3.6 – Функціональна схема розробленої системи

На схемі видно зв'язок між роз'ємом паралельного порту, регістрами та модулями розробленої програми.

Програма складається з наступних модулів:

– автотестування;

– тестування вручну;

– читання регістра;

– запис у регістр.

Тестування вручну полягає у тому, що користувач записує у регістр будь-яке значення, а потім програма зчитує значення цього регістра та перевіряє чи коректно пройшов запис.

Автотестування здійснюється послідовним записом та зчитуванням бітів регістрів керування та даних. Якщо усі біти були записані коректно, то автотестування пройшло успішно.

Регістри даних та керування доступні для як для читання так і для запису. Регістр стану доступний лише для читання.

З рисунку 3.6 видно, що виводи порту можна розділити на чотири групи. Виводи заземлення позначені чорним кольором (контакти 18-25). Всі вони з'єднані між собою, тому для заземлення можна використовувати кожний з них. Червоним кольором позначені виводи регістра даних (контакти 2-9). У регістрі даних їх 8 штук. Регістр стану – контакти 10-13, 15. Тобто йому ставляться у відповідність 5 контактів. Регістр керування – 1, 14, 16-17. Він має всього 4 контакти.

Тепер розглянемо як відбувається запис і читання даних у регістри LPT-порту, тобто як нам встановити на потрібних виводах "0" або "1".

Запис/читання інформації у регістр даних

Розглянемо відразу практичне завдання. Потрібно записати у регістр даних послідовного порту число 245. Пишемо код:

```
int Address=0x378;
int data=245;
Out32(Address, data);
```

В даному випадку наведена функція Out32 бібліотеки inpout32.dll, подальші приклади будуть також основані на цій бібліотеці.

Першим параметром функції Out32 ми передаємо число 0x378 (адресу регістра даних LPT порту). Тепер функції відомо куди писати дані. Далі другим параметром передаємо число 245, яке нам треба записати у регістр.

Переводимо 245 у двійкову систему числення й справа наліво записуємо розряди числа у відповідні біти регістра. У підсумку одержимо, що на контактах LPT-порту під номерами 2,4, 6-9 є присутньою напруга +5В, на контактах 3,5 – нуль (рисунок 3.7).

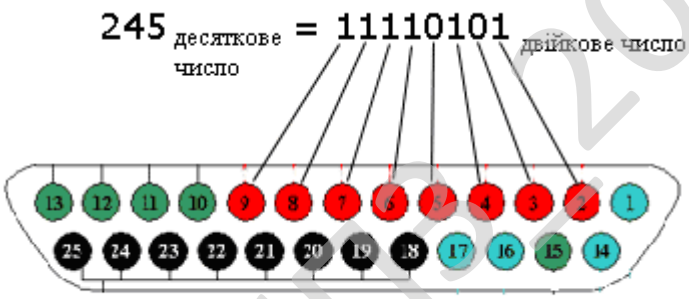

Рисунок 3.7 – Запис інформації у регістр даних

Оскільки цей регістр 8-ми розрядний, діапозон десяткових чисел, які можна записати в нього даних лежить у межах від 0 до 255.

Тепер розглянемо як читати інформацію з регістру даних. Припустимо, необхідно довідатися на яких виводах регістра даних зараз високий рівень напруги, а на яких низький. Пишемо код:

```
int Address=0x378;
int data;
data = Inp32(Address);
```

Inp32 це функція для читання даних з порту бібліотеки inpout32.dll. Єдиним параметром для неї є адреса того регістра, звідки треба прочитати дані. На виході вона повертає десяткове число, що відповідає поточному значенню

регістра. Після виконання цього коду, змінна data буде містити число 245, що було записане в регістр у попередньому прикладі.

В регістр даних інформацію може записати й зовнішній пристрій. Однак це потребує зовнішніх джерел живлення.

Запис/читання даних у регістр Control

Регістр керування односпрямований, дані в нього може записати тільки програма. Цей регістр має декілька особливостей. По-перше, він відповідає всього чотирьом контактам. Значить у нього можна записати число в діапозоні від 0 до 24-1=16-1=15. По-друге, він має дуже неприємну особливість: деякі з його виводів інвертовані, тобто якщо Ви на цей вивід пишете "1", то на ньому установлюється "0". І навпаки, читаєте "1", а насправді там "0". Тому, значення записуваної і прочитаної інформації не зовсім очевидне. Приклад запису числа в регістр керування:

```
int Address=0x37A; //адреса регістра керування
int data=10;
Out32(Address, data);
```

І приклад читання:

```
int Address=0x37A; //адреса регістра керування
int data;
data = Inp32(Address);
```

Запис/читання даних у регістр стану

Регістр стану односпрямований, дані в нього може записати тільки зовнішній пристрій, тобто в програмі можна тільки читати вміст цього регістра. Прочитавши дані з регістра стану, і перевівши їх у двійкове число, відразу досить важко зрозуміти що ж реально діється з напругами на виходах цього регістра. По-перше, він теж має інвертовані виводи, а по-друге робочими є біти під номерами 4-7, а біти 0-3 не використовуються, і отже число записується досить хитро. Приклад читання:

```
int Address=0x379; //адреса регістра стану
int data;
data = Inp32(Address);
```

Виникає питання, а як ці дані на ньому встановити? Це можна зробити виступивши в ролі зовнішнього пристрою. Після виконання коду, наведеного вище, у змінній data буде міститися деяке число. Тепер візьмемо провідник і з'єднаємо їм кожний із земляних виводів порту (18-25) з яким-небудь виводом регістра Status (10-13, 15), наприклад з десятим. І знову виконаємо читання. Після цього буде одержите інше число. Заберемо провідник. Прочитавши, напівпарі одержимо вихідне число. Як це працює? Напочатку, на всіх виводах цього регістра перебуває високий рівень напруги +5В. При з'єднанні одного з його виводів із землею, на напруга ньому стає рівною нулю. Тепер двійковим даним у регістрі відповідає інше десяткове число. Можна спробувати замикати й інші виводи регістра стану на землю, замикати відразу декілька. Щоразу при читанні вийде різний результат.

3.4 Розробка діаграми процесів

Розглянемо діаграму процесів зображену на рисунку 3.8.

Робота програми починається з перегляду наявних на комп'ютері LPT-портів та виведення їхнього списку. Потім користувач обирає з них один для діагностики. Діагностика здійснюється за допомогою процесів автотестування та тестування вручну, що в свою чергу тісно пов'язані з процесами читання та запису у регістри. Після автотестування, його результати виводяться на екран. Тестування вручну вимагає від користувача самостійного аналізу операцій читання/запису.

Якщо під час діагностики була виявлена помилка, і користувач хоче дізнатися більш детально про причини її виникнення, то він може скористатися довідкою.

В роз'ємах інтерфейсу Centronics, які фіксуються за допомогою металевих скоб, іноді пропадає контакт, навіть якщо з'єднання виглядає цілком надійним. Так що при підключенні кабелю до принтера необхідно досить сильно нажати на

роз'єм з боку кабелю, щоб гарантувати гарний контакт. Тут, звичайно, важливо не перестаратися, тому що інші неполадки зв'язані звичайно з обривами або замиканнями проводів у кабелі або роз'ємі через недбале з ним обходження.

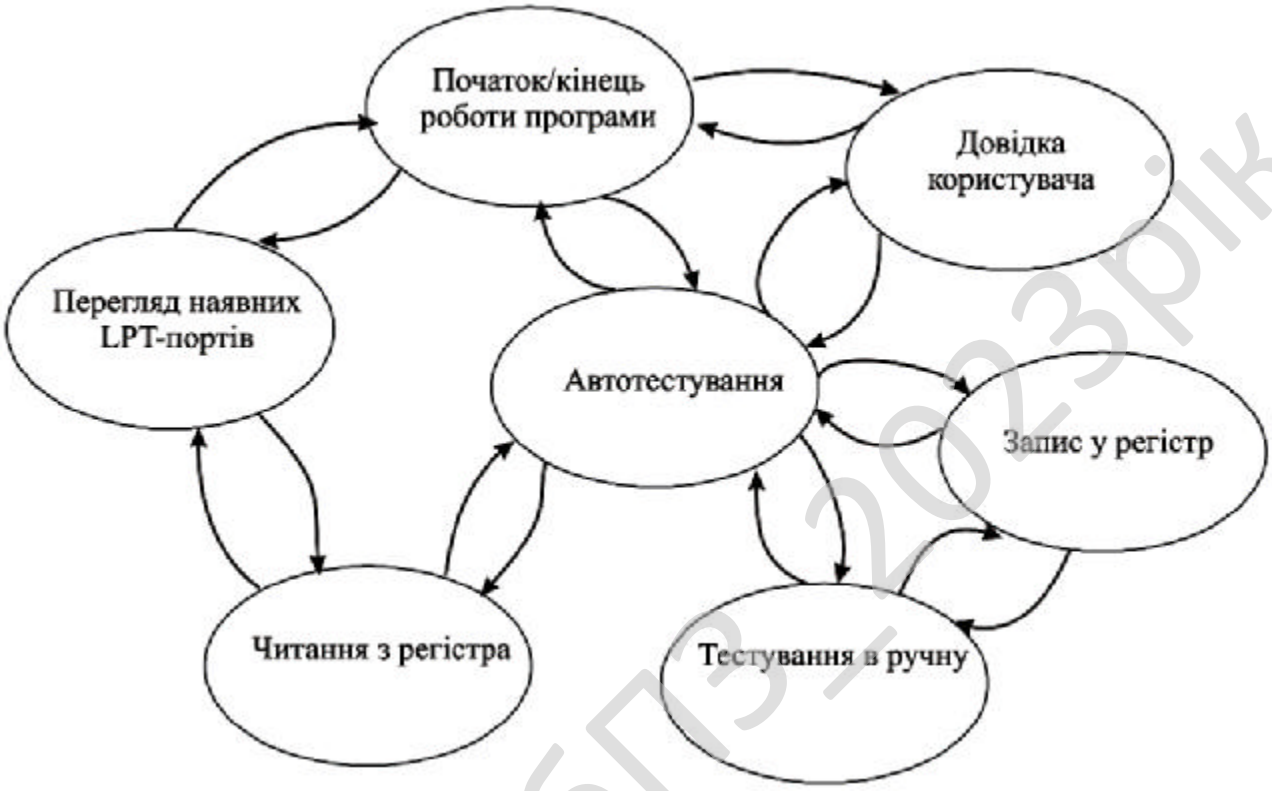

Рисунок 3.8 – Діаграма процесів розробленої системи

Якщо обрив відбувся в лініях даних, то свідомо справний принтер починає «говорити з турецьким акцентом» замість одних символів будуть друкуватися зовсім інші. До речі, така ж ситуація може мати місце при виході з ладу мікросхеми на самому адаптері. Якщо ж обрив відбувся в лініях квитування й статусу, то виявити несправність досить важко, тому що в деяких випадках принтер тривалий час може працювати нормально. Для перевірки буферної мікросхеми й ліній кабелю, призначених для сигналів квітування й статусу, можна виготовити спеціальну «заглушку» (loopback або wrap-plug).

Таким чином, розглянувши опис системи, структурну, функціональну схеми системи, та діаграму взаємодії процесів перейдемо до опису блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

Блок-схема роботи основної програми показана на рисунку 4.1. З нього видно, що після запуску програми відбувається ініціалізація LPT-портів та їх регістрів та вивід початкових значень регістрів.

Після цього користувач може здійснити автотестування або тестування вручну, чи переглянути довідку про паралельний порт.

При автотестуванні відбувається послідовне встановлення та обнуління бітів регістрів даних та керування. Потім значення цих регістрів зчитуються та порівнюються з введеними. Якщо введені та прочитані значення співпали, то тестування пройшло успішно і на екран виводиться відповідне повідомлення. Якщо ж між введеними та прочитаними значеннями є розбіжності, то LPT-порт несправний, і на екран виводиться повідомлення про помилку, а по бітах, у яких виникнув збій, можна встановити, які саме контакти порту несправні.

Підпрограма автотестування виглядає наступним чином:

```
void __fastcall TForm1::SpeedButton3Click(TObject *Sender)
{
    bool io = true;
    int in, out;
    for (int i=0; i<9; i++)
    {
        if (i==8)
            in = 0;
        else
            in = pow(2, i);
        Frame31->Frame22->Tag = in;
        Frame31->Frame22->GetTag();
        Frame31->process(Sender);
```

```
            out = Frame31->Frame21->Tag;
            if (in != out)
                io = false;
            Sleep(500);
        }
        if (io)
            ShowMessage("Тестування регістра даних пройшло УСПІШНО!");
        else
            ShowMessage("Тестування регістра даних пройшло НЕ УСПІШНО!");
        for (int i=0; i<6; i++)
        {
            if (i==5)
                in = 192;
            else
                in = 192 + pow(2, i);
            Frame33->Frame22->Tag = in;
            Frame33->Frame22->GetTag();
            Frame33->process(Sender);
            out = Frame33->Frame21->Tag;
            if (in != out)
                io = false;
            Sleep(500);
        }
        if (io)
            ShowMessage("Тестування регістра керування пройшло УСПІШНО!");
        else
            ShowMessage("Тестування регістра керування пройшло НЕ УСПІШНО!");
    }
```

При тестуванні вручну користувач побітно, або побайтно вводить значення у регістри даних та керування. Значення можна вводити у десятковому, шістнадцятковому чи вісімковому форматі. Тестування у ручному режимі також засноване на порівнянні записаних та прочитаних значень регістрів.

Підпрограма введення значень у регістр даних виглядає наступним чином:

```
void __fastcall TChangeRegDat::Button1Click(TObject *Sender)
{
    AnsiString data;
    data = Edit1->Text;
    char *s;
   s[0]=0; s[1]=0; s[2]=0; s[3]=0;
    s = data.c_str();
```

```
    byte RegData;
    if ((s[0] == '0')&(data != '0'))
    {
        if ((s[1] == 'x') | (s[1] == 'X'))
            RegData = hex(s[2], s[3]);
        else
        {
            if (s[1] != NULL)  RegData = oct(s[1], s[2], s[3]);
        }
 }
    else   RegData = dec(s[0], s[1], s[2], s[3]);
    Form1->Frame31->Frame22->Tag = RegData;
    Form1->Frame31->Frame22->GetTag();
    Form1->Frame31->process(Sender);
    Close();
}
```

Підпрограма встановлення регістра керування:

```
void __fastcall TChangeRegContr::Button1Click(TObject *Sender)
{
    AnsiString data;   int r;
    data = Edit1->Text;
    r=StrToInt(data);
    if(r>31){
      ShowMessage("Значення перевищило 31! Воно буде замінене на 0.");
      data = '0'; }
          char *s;
    s[1]=0; s[2]=0; s[3]=0;
    s = data.c_str();
    byte RegData;

   if ((s[0] == '0')&(data != '0'))
    {
        if ((s[1] == 'x') | (s[1] == 'X'))
            RegData = hex(s[2], s[3]);
        else
        {
            if (s[1] != NULL)  RegData = oct(s[1], s[2], s[3]);
        }
    }
    else    RegData = dec(s[0], s[1], s[2], s[3]);
    Form1->Frame33->Frame22->Tag = RegData | 192;
    Form1->Frame33->Frame22->GetTag();
```

```
    Form1->Frame33->process(Sender);
    Close();
}
```

Для роботи з числами у шістнадцятковій та вісімковій системі ліку створені функції:

```
byte switch_case (byte, char); //визначення системи ліку числа
byte hex (char, char); //переведення рядка в 16-ве число
byte oct (char, char, char); //переведення рядка в 8-ве число
byte dec (char, char, char, char); //переведення рядка в десяткове число
```

Функція switch_case призначена для визначення системи ліку до якої відноситься число введене користувачем:

```
byte switch_case(byte ci, char ch)    // ci = 0 - oct
{   // ci = 1 - dec
    int ch_i = (int)ch;          // ci = 2 - hex
    byte x;
    if (ci==2)
        switch (ch_i)
        {
            case '0':   case ' ':   case '\0':  x = 0;   break;
            case '1':   x = 1;   break;
            case '2':   x = 2;   break;
            case '3':   x = 3;   break;
            case '4':   x = 4;   break;
            case '5':   x = 5;   break;
            case '6':   x = 6;   break;
            case '7':   x = 7;   break;
            case '8':   x = 8;   break;
            case '9':   x = 9;   break;
            case 'a':   case 'A':   x = 10;   break;
            case 'b':   case 'B':   x = 11;   break;
            case 'c':   case 'C':   x = 12;   break;
            case 'd':   case 'D':   x = 13;   break;
            case 'e':   case 'E':   x = 14;   break;
            case 'f':   case 'F':   x = 15;   break;
            default:    ShowMessage("Це не шістнадцяткове число!");
             x = 21;    // 21 - код помилки
        }
    if (ci==1)
        switch (ch_i)
```

```
    {
        case '0':   case ' ':   case '\0':  x = 0;   break;
        case '1':   x = 1;   break;
        case '2':   x = 2;   break;
        case '3':   x = 3;   break;
        case '4':   x = 4;   break;
        case '5':   x = 5;   break;
        case '6':   x = 6;   break;
        case '7':   x = 7;   break;
        case '8':   x = 8;   break;
        case '9':   x = 9;   break;
        default:    ShowMessage("Це не десяткове число!");
         x = 21;
    }
if (ci==0)
    switch (ch_i)
    {
        case '0':   x = 0;   break;
        case '1':   x = 1;   break;
        case '2':   x = 2;   break;
        case '3':   x = 3;   break;
        case '4':   x = 4;   break;
        case '5':   x = 5;   break;
        case '6':   x = 6;   break;
        case '7':   x = 7;   break;
        case ' ':   case '\0':  x = 0;   break;
        default:    ShowMessage("Це не вісімкове число!");
        x = 21;
    }
return x;}
```

Функція hex призначена для переведення рядка в 16-ве число:

```
byte hex(char s3, char s4)
{
    byte s3_i, s4_i, hex_temp=0;
    s3_i = switch_case (2, s3);
    if (s3_i == 21)
        return hex_temp = 0;
    s4_i = switch_case (2, s4);
    if (s4_i == 21)
        return hex_temp = 0;
```

```
    if ((s3==' ')|(s3=='\0'))      hex_temp = 0;
        else
            if ((s4==' ')|(s4=='\0'))
                      hex_temp = s3_i;
            else   hex_temp = s3_i*16 + s4_i;
    return hex_temp;}
```

Функція oct призначена для переведення рядка в 8-ве число:

```
byte oct(char s2, char s3, char s4)
{
    byte s2_i, s3_i, s4_i;
    int oct_temp;
    s2_i = switch_case (0, s2);
    if (s2_i == 21)
        return oct_temp = 0;
    s3_i = switch_case (0, s3);
    if (s3_i == 21)
        return oct_temp = 0;
    s4_i = switch_case (0, s4);
    if (s4_i == 21)
        return oct_temp = 0;
    if ((s2==' ')|(s2=='\0'))     oct_temp = 0;
        else
            if ((s3==' ')|(s3=='\0'))
                      oct_temp = s2_i;
            else
                if ((s4==' ')|(s4=='\0'))
                          oct_temp = s2_i*8 + s3_i;
                else
                oct_temp = s2_i*64 + s3_i*8 + s4_i;
    if (oct_temp > 255)
    {
        ShowMessage("Значення перевищило 255! Воно буде замінене на 0.");
        oct_temp = 0;
    }
    return (byte)oct_temp;
}
```

Функція dec призначена для переведення рядка в десяткове число:

```
byte dec(char s1, char s2, char s3, char s4)
```

```
{
    byte s1_i, s2_i, s3_i, s4_i;
    int dec_temp;
    s1_i = switch_case (1, s1);
    if (s1_i == 21)
        return dec_temp = 0;
    s2_i = switch_case (1, s2);
    if (s2_i == 21)
        return dec_temp = 0;
    s3_i = switch_case (1, s3);
    if (s3_i == 21)
        return dec_temp = 0;
    s4_i = switch_case (1, s4);
    if (s4_i == 21)
        return dec_temp = 0;
    if ((s1==' ')|(s1=='\0'))      dec_temp = 0;
        else
            if ((s2==' ')|(s2=='\0'))
                      dec_temp = s1_i;
            else
                if ((s3==' ')|(s3=='\0'))
                            dec_temp = s1_i*10+s2_i;
                else
                    if ((s4==' ')|(s4=='\0'))
                               dec_temp = s1_i*100+s2_i*10+s3_i;
                    else
           dec_temp = s1_i*1000+s2_i*100+s3_i*10+s4_i;
    if (dec_temp > 255)
    {
        ShowMessage("Значення перевищило 255! Воно буде замінене на 0.");
        dec_temp = 0;
    }
    return (byte)dec_temp;}
```

З алгоритму основної програми видно, що найважливішими у розробленій системі є операції запису та читання регістрів, адже на них оснований процес діагностики паралельного порту. Саме тому блок-схеми підпрограм запису та читання виведені на рисунках 4.2 та 4.3.

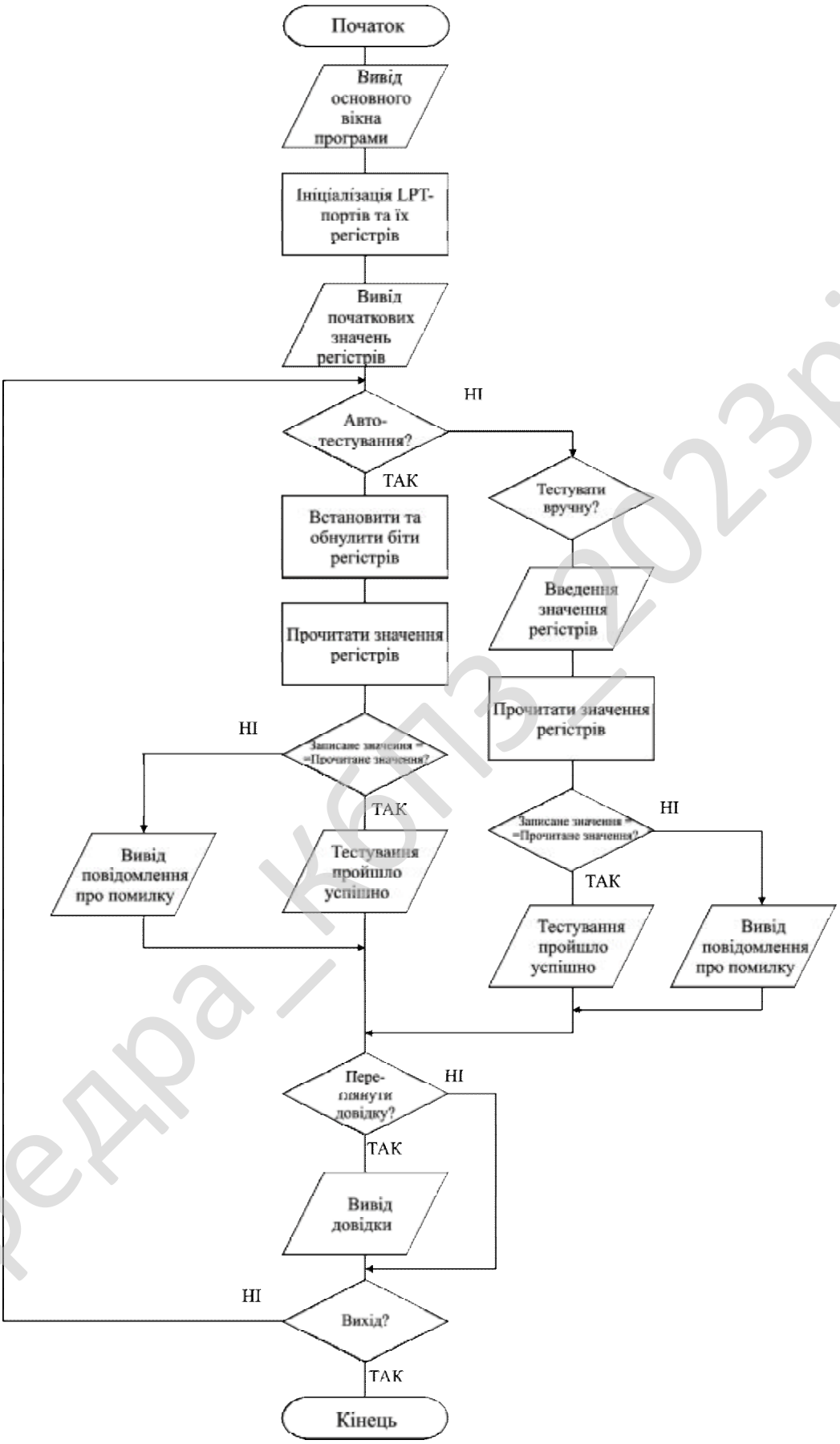

Рисунок 4.1 – Блок-схема основної програми

Розглянемо підпрограми запису та читання регістрів. Перед записом до порту треба здійснити його ініціалізацію. Для забезпечення обміну інформацією *базова адреса + 2* (B+2) безпосередньо не використовується, однак у ній необхідно визначити стан бітів C4 і C5.

При використанні апаратного переривання по сигналу /ACK (ніжка 10 роз'єму порту) біт C4 адреси B+2 необхідно встановити (C4=1).

При використанні інших алгоритмів біт C4 адреси B+2 варто скинути (D4=0). Також слід зазначити, що при використанні переривання, необхідно ініціалізувати контролер переривань.

Біт C5 адреси B+2 у старих моделях комп'ютерів не використовується, але для сумісності з останніми ЕОМ його необхідно скинути (C5=0), або зовсім не змінювати.

Стан інших бітів адреси B+2 може бути довільним, тому що в даній конфігурації вони залишаються непідключеними.

Крім того, необхідно визначити стан *базової адреси* (B), зокрема сигналу, що буде надалі використовуватися для стропування інформації, у противному випадку можливе порушення синхронної роботи портів.

При забезпеченні зв'язку через паралельний порт необхідно розробити протокол обміну даними, або використовувати один із пропонованих. Можуть бути використані алгоритми з використанням стробуючого імпульсу, або протоколи, що самосинхронізуються, подібні тим, які використовуються для запису на магнітний носій.

При використанні алгоритмів зі стробучим імпульсом із всіх інформаційних ліній необхідно вибрати одну для стробування виведеної інформації, а інші задіяти для передачі інформації.

У цьому випадку використовуються п'ять інформаційних ліній, одну з яких можна вибрати для стробування інформації, а по інших чотирьох передавати байт у вигляді двох напівбайтів.

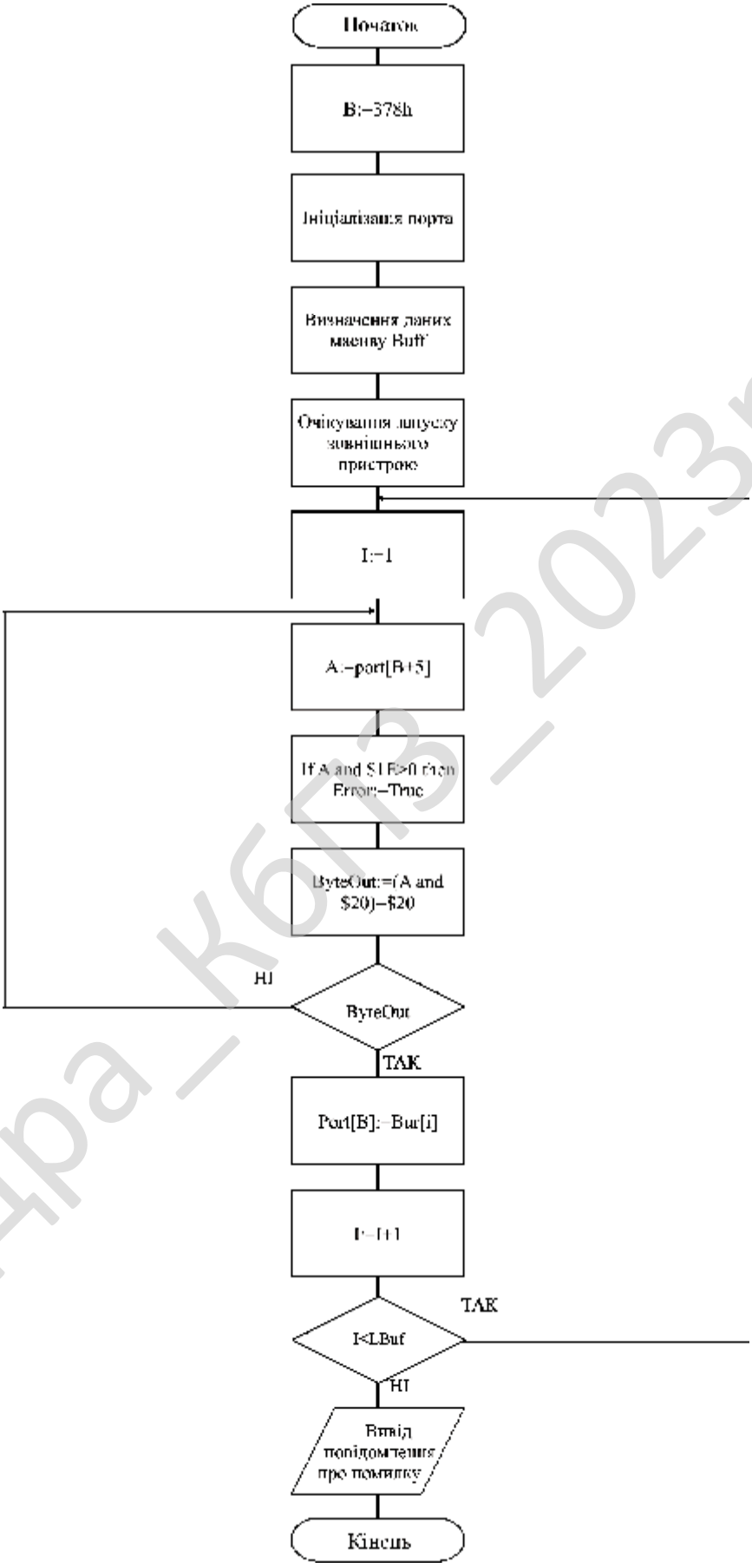

Рисунок 4.2 – Блок-схема підпрограми запису в регістр

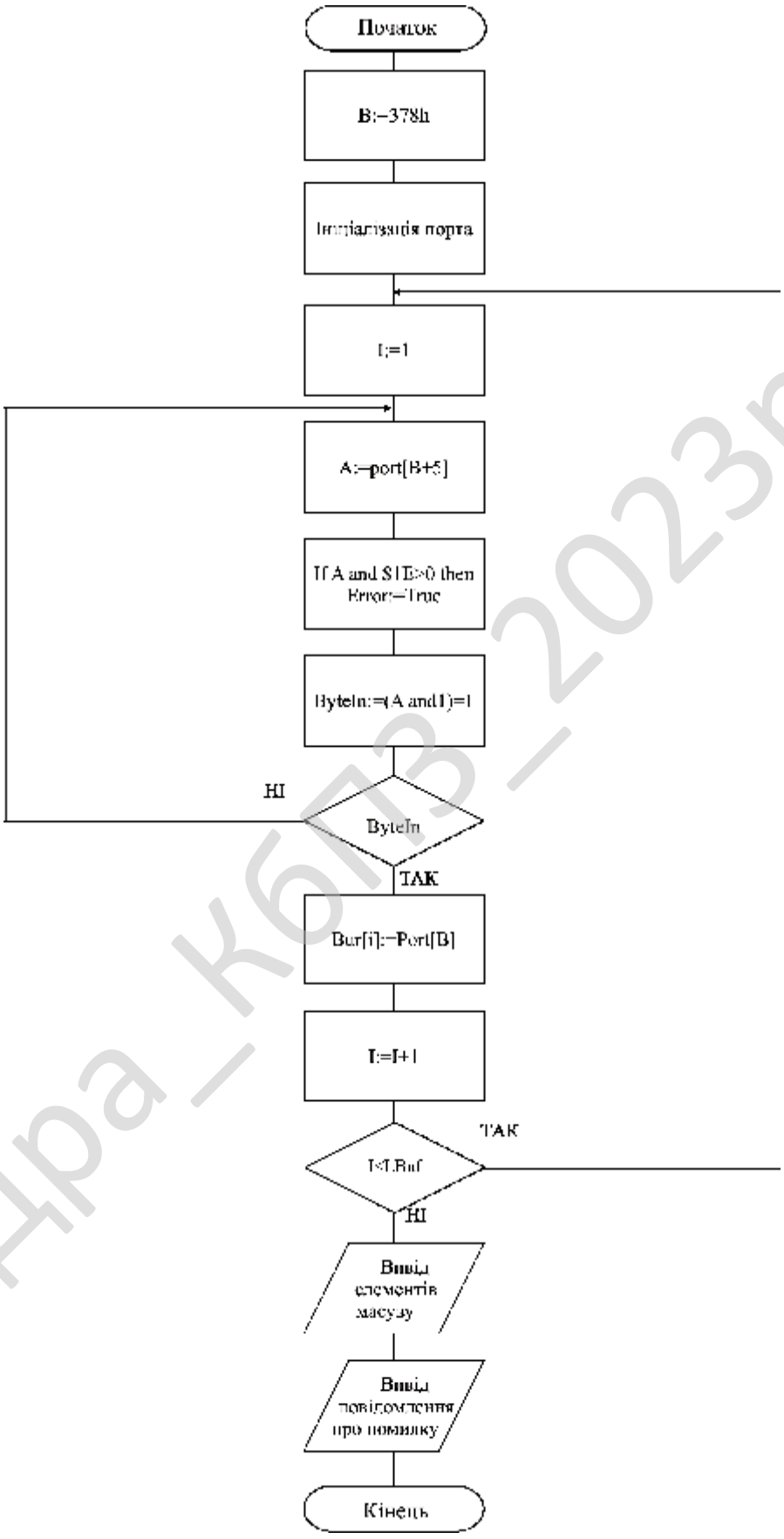

Рисунок 4.3 – Блок-схема підпрограми читання регістра

При використанні сигналу переривання (IRQ7 для LPT1 або IRQ5 для LPT2) у якості строба краще вибрати лінію D3, позитивний перепад на якій викличе переривання.

При використанні алгоритмів перевірки біта шляхом опитування його стану вибір номера інформаційної лінії для строба довільний.

Алгоритм підпрограми запису у регістр:

1. Визначити базову адресу комунікаційного порту.
2. Виконати ініціалізацію комунікаційного порту.
3. Підготувати масив даних для передачі.
4. Передати підготовлений масив даних у порт.

Для передачі даних необхідно організувати цикл. У даному циклі необхідно виводити дані в порт, попередньо очікуючи (перевіряючи) готовність зовнішнього пристрою прийняти дані.

Передані дані повинні супроводжуватися сигналом Strobe. Очікування готовності реалізується шляхом організації циклу, вихід з якого здійснюється шляхом аналізу сигналу Ack від зовнішнього пристрою.

Алгоритм підпрограми читання регістра.

1. Визначити базова адреса комунікаційного порту.
2. Виконати ініціалізацію комунікаційного порту.
3. Відновити значення масиву даних шляхом читання з порту.
4. Вивести отриманий масив даних на екран.

Для прийому з порту необхідно організувати цикл. У даному циклі необхідно послідовно читати дані з порту й розміщати їх у масиві даних.

Дані варто читати у випадку, якщо сигнал Strobe зовнішнього пристрою активний.

По завершенні читання даних необхідно підтвердити прийом даних активізацією сигналу Ack.

Очікування сигналу Strobe і даних реалізується шляхом організації циклу, вихід з якого здійснюється за результатами аналізу сигналу Strobe.

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою алгоритму FEAL – блоковий шифр, запропонований Акіхіро Симідзу і Седзі Міягуті.

У ньому використовуються 64-бітовий блок і 64-бітовий ключ. Його ідея полягає і в тому, щоб створити алгоритм, подібний DES, але з більш сильною функцією етапу. Використовуючи менше етапів, цей алгоритм міг би працювати швидше. На жаль, дійсність виявилася далекою від цілей проекту.

Як вхід процесу шифрування використовується 64-бітовий блок відкритого тексту. Спочатку блок даних підлягає операції XOR з 64 бітами ключа. Потім блок даних розщеплюється на ліву і праву половини. Об'єднання лівої і правої половин за допомогою XOR утворює нову праву половину. Ліва половина і нова права половина проходять через N етапів (спочатку 4). На кожному етапі половина об'єднується за допомогою функції F[1] з 16 бітами ключа і за допомогою XOR – з лівою половиною, створюючи нову праву половину. Вихідна права половина (на початок етапу) стає новою лівою половиною. Після N етапів (ліва і права половини не переставляти після N-го етапу) ліва половина знову об'єднується з допомогою XOR з правою половиною, утворюючи нову праву половину, потім ліва і права об'єднуються разом в 64-бітове ціле. Блок даних об'єднується за допомогою XOR з іншими 64 бітами ключа і алгоритм завершується.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1.

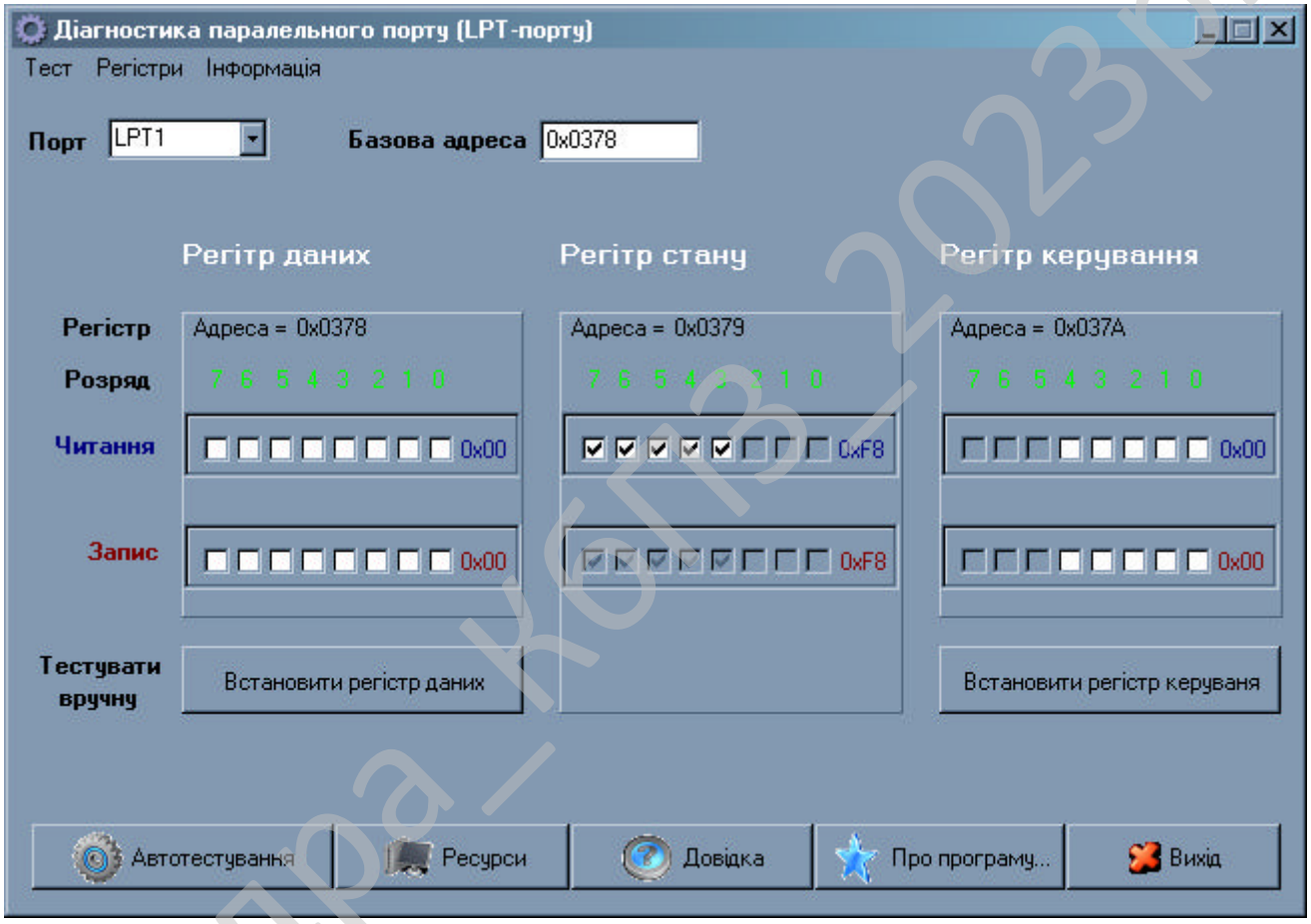

Рисунок 5.1 – Головне вікно програми

Маємо три регістри – даних, статусу й керування. У перший і третій можемо писати побітно, клацаючи мишкою в чекбоксах рядоку – «запис», або побайтно, за допомогою кнопок «Встановити регістр даних» та «Встановити регістр керування» (рисунки 5.2, 5.3). При побайтному записі значення байта можна вводити в десятковому, вісімковому або в шістнадцятковому форматі.

Регістр статусу обновляється автоматично.

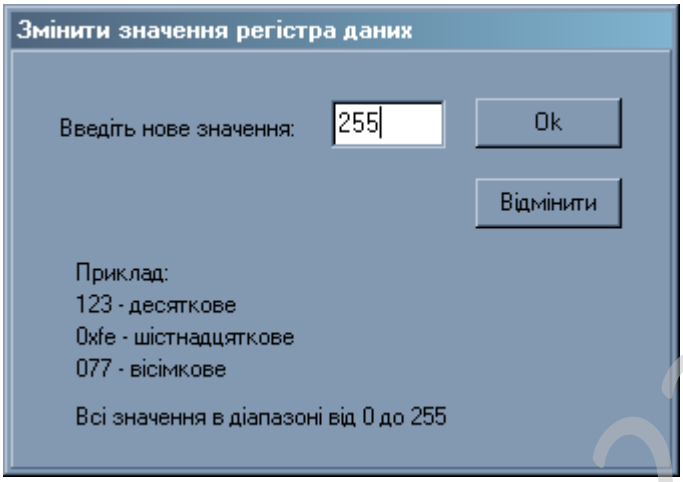

Рисунок 5.2 – Ручне тестування регістра даних

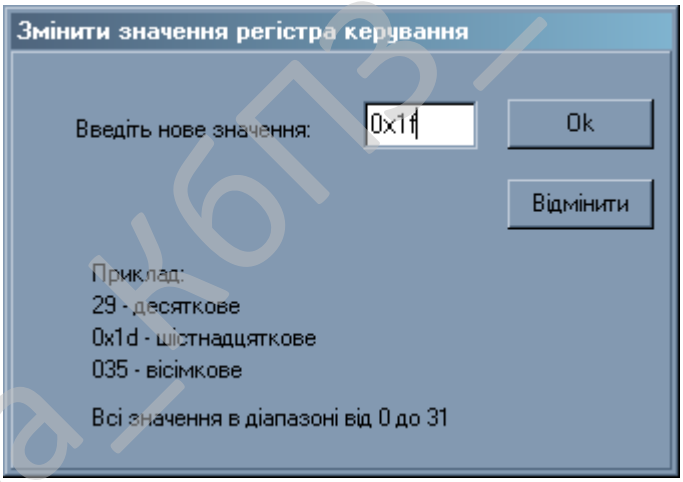

Рисунок 5.3 – Ручне тестування регістра керування

Якщо регістр справний, то у відповідних розрядах рядка «читання» встановляться такі значення, які користувач ввів у рядок «запис». Якщо ж значення рядків «запис» та «читання» не співпадають, то порт несправний.

Наявність галочки – логічна «1», відсутність – логічний «0».

Крім тестування вручну реалізоване *автотестування*. Воно вмикається завдяки відповідній кнопці. Під час автотестування програма сама послідовно

встановлює та обнулює всі біти регістру даних та керування, а потім, порівнявши значення рядків «запис» та «читання», виводить результат (рисунок 5.2)

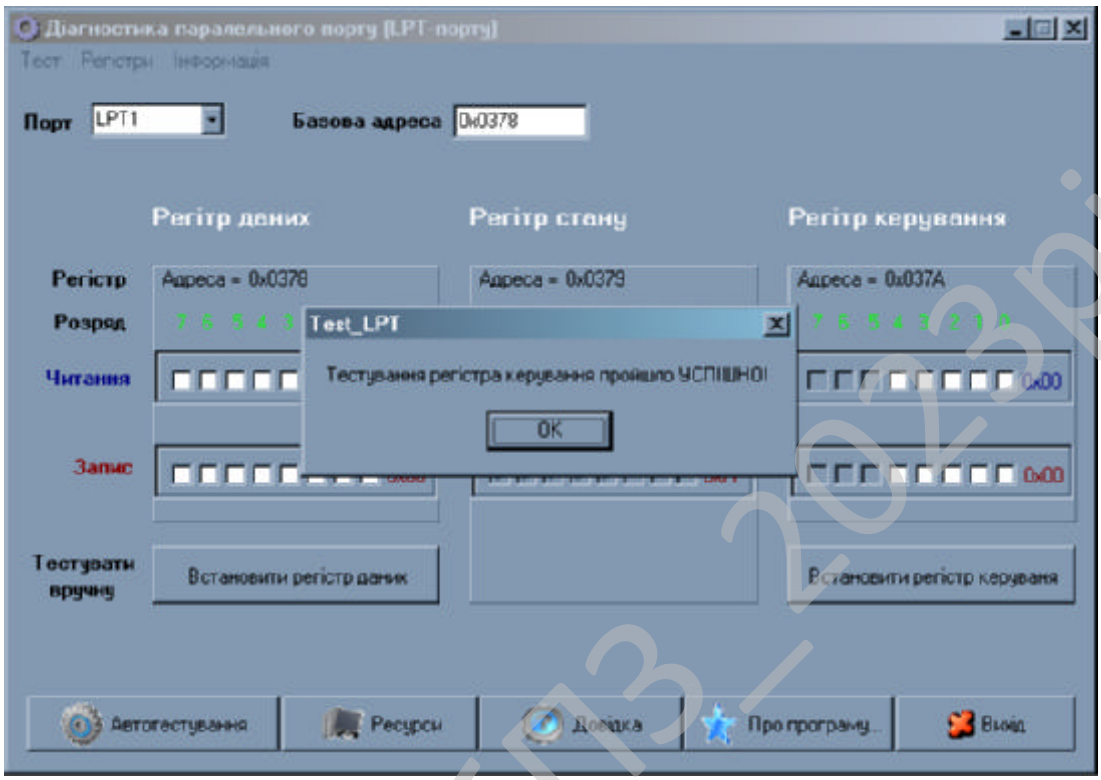

Рисунок 5.4 – Автотестування

Програма також дозволяє довідатися про наявні на комп'ютері LPT-порти (рисунок 5.3), для отримання цієї інформації треба натиснути кнопку «Ресурси».

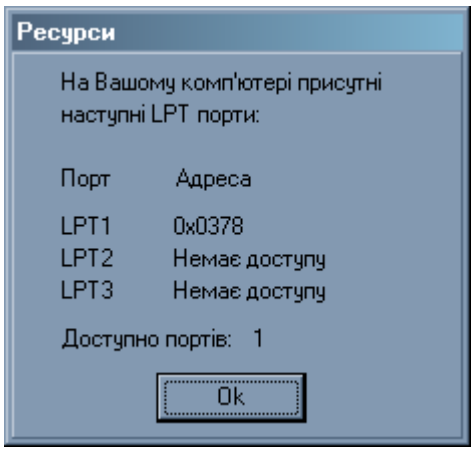

Рисунок 5.5 – Ресурси

Натиснувши на кнопку «Довідка», можна переглянути призначення бітів регістрів паралельного порту, та побачити, яким виводам порту вони відповідають (рисунок 5.6). Якщо в процесі тестування виникла помилка в якомусь біті, то за допомогою довідки можна визначити який саме вивід LPT-порту несправний.

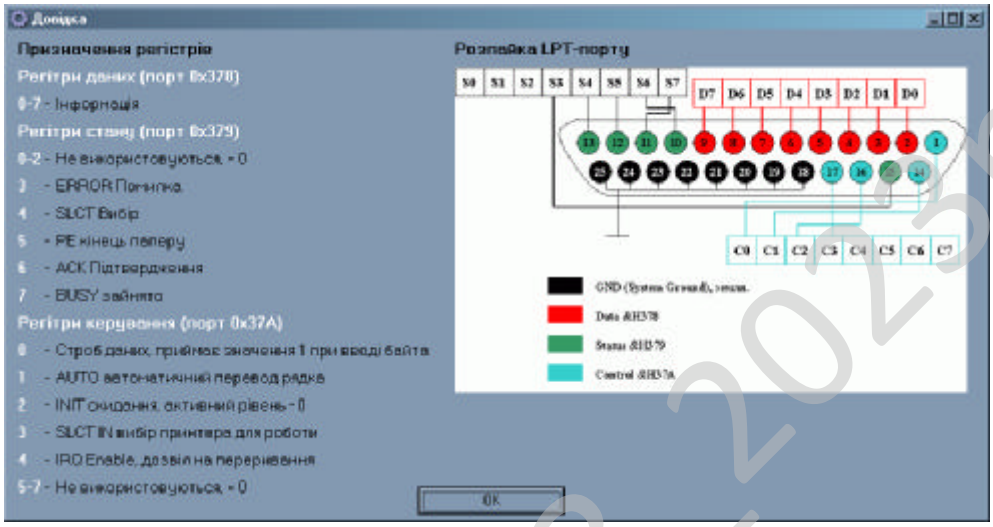

Рисунок 5.6 – Довідка

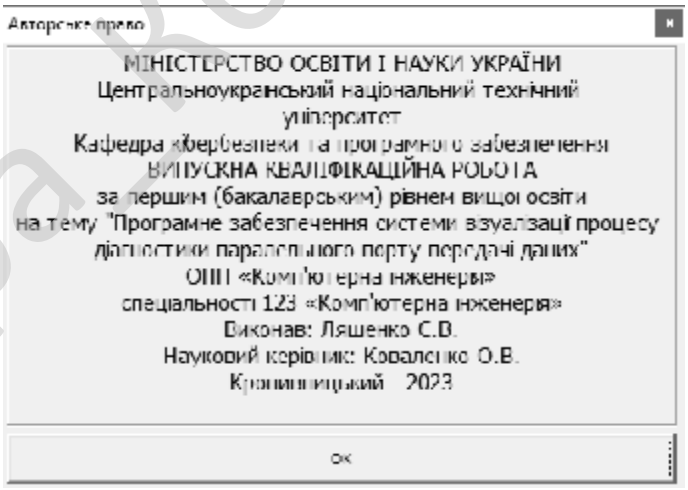

Рисунок 5.7 – Про програму

Призначення програми та ім’я автора можна побачити, натиснувши кнопку «Про програму…» (рисунок 5.7).

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи візуалізації процесу діагностики паралельного порту передачі даних.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем візуалізації процесу діагностики паралельного порту передачі даних.

– Досліджена система візуалізації процесу діагностики паралельного порту передачі даних.

– На основі отриманих результатів досліджень створена програмна реалізація системи візуалізації процесу діагностики паралельного порту передачі даних.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання візуалізації процесу діагностики паралельного порту передачі даних.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня Builder C++. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи візуалізації процесу діагностики паралельного порту передачі даних. Це

дозволило мінімізувати строк розробки програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows 10/11.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм FEAL.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Аппаратние средства IBM PC. Энциклопедия, 2-е. – СПб.: Питер, 2001 -928 с.: ил. Автор – Михаил Гук
2. Атовмян И.О. Архитектура вичислительних систем М.: МИФИ, 2002
3. Бененсон Е.П., Витенберг И.М., Мельников В.В. и д.р. Печатающие устройства персональных для ЭВМ: Справочник // Под ред. И.М. Витенберга. – М.: Радио и связь, 1992.-208 с.
4. Браун Р., Кайл Дж. Справочник по прерываниям для IBM PC: В 2 Т. Т.1: Пер. с англ. – М.: Мир, 1994. – 558 с.
5. Вильямс А. Системное программирование Windows 2000. – СПб.: Питер, 2001.
6. Гук М. Аппаратние средства IBM PC. – СПб.: Питер, 2001.
7. Гук М. Интерфейси ПК: Энциклопедия. – СПб.: Питер, 2001.
8. Коваленко А.С. Разработка структуры базы данных интегрированной информационной системы / А.С. Коваленко, А.В. Коваленко // Информационные технологии и защита информации в информационно-коммуникационных системах: монографія / Под редакцией профессора В.С. Пономаренко. – Х.: Вид-во ТОВ «Щедра садиба плюс», 2015. – С. 54-64.
9. Кожанова А.С. Обґрунтування необхідності створення систем технічної діагностики інтегрованих інформаційних систем / О.А. Смірнов, А.С. Кожанова, О.В. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2013. – Вип. 6(113). – С. 255-257.
10. Коваленко А.С. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко., А.А. Смірнов, А.С. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2014. – Вип. 4(120). – С. 161-164.
11. Коваленко А.С. Підсистема технічної діагностики для автоматизації процесів керування в інтегрованих інформаційних системах / А.С. Коваленко ,

О.А.Смірнов, О.В. Коваленко // Системи озброєння і військова техніка.– Х.: ХУПС, 2014. – № 1(37). – С. 126-129.

12. Коваленко А.С. Анализ эффективности использования экспертной системы технической диагностики с традиционной структурой / А.С. Коваленко, А.А. Смирнов, А.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2014. – № 2(38). – С. 106-108.

13. Коваленко А.С. Разработка структуры экспертной системы технической диагностики интегрированной информационной системы / А.С. Коваленко, А.А. Смирнов, А.В. Коваленко // Наука і техніка Повітряних Сил Збройних Сил України. – Харків: ХУПС, 2014. – № 2(15). – С.154-157.

14. Коваленко А.С. Разработка структуры экспертной системы технической диагностики интегрированной информационной системы / А.С. Коваленко, А.А. Смирнов, А.В. Коваленко // Наука і техніка Повітряних Сил Збройних Сил України. – Харків: ХУПС, 2014. – № 2(15). – С.154-157.

15. Коваленко А.С. Структура системи технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Збірник наукових праць Кіровоградського національного технічного університету / техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Кіровоград: Вид-во КНТУ, 2014. – Вип. 27. – С. 245-251.

16. Коваленко А.С. Дослідження будови інтегрованої інформаційної системи та її елементів / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2014. – № 4(40). – С. 85-88.

17. Коваленко А.С. Розробка структури бази даних для обліку технічного стану елементів інтегрованої інформаційної системи з урахуванням вимог споживачів інформації / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2015. – Вип. 1(126). – С. 75-79.

18. Коваленко А.С. Обґрунтування набору даних для оцінки технічного стану інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В.

Коваленко // Збірник наукових праць Харківського університету Повітряних Сил. – Харків: ХУПС, 2015. – Вип. 1(42). – С.39-41.

19. Коваленко А.С. Експертна система технічного діагностування інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Системи озброєння і військова техніка. – Х.: ХУПС, 2015. – № 1(41). – С. 106-111.

20. Коваленко А.С. Удосконалення методу технічного обслуговування об'єктів інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко, О.П. Доренський // Системи озброєння і військова техніка. – Х.: ХУПС, 2016. – № 2(46). – С. 109-114.

21. Коваленко А.С. Метод визначення оптимального комплексу робіт з відновлення працездатності інтегрованої системи технічної діагностики в умовах ресурсних обмежень / А.С. Коваленко // Системи обробки інформації. – Х.: ХУПС, 2016. – Вип. 3(140). – С. 69-72.

22. Kovalenko A.S. Information model and its element for displaying information on technical condition of objects of integrated information system / A.S. Kovalenko, A.A. Smirnov, A.V. Kovalenko, A.P. Dorensky // International Journal of Computational Engineering Research (IJCER). – India: Delhi, 2016. – Volume 6, Issue 1. – P. 21-27.

23. Кожанова А.С. Система технічної діагностики інтегрованих інформаційних систем – обґрунтування необхідності створення, визначення понятійного апарату та напрямів досліджень / А.С. Кожанова, О.А. Смірнов, М.П. Савченко, Д.М. Ізосімов, В.В. Мороз // Створення та модернізація озброєння і військової техніки в сучасних умовах: Тринадцята наук.-техн. конф., 5-6 вер. 2013 р., м. Феодосія: тези доп. – Феодосія: ДНВЦ, 2013. – С. 187-188.

24. Кожанова А.С. Визначення основних напрямків досліджень щодо створення системи технічної діагностики інтегрованих інформаційних систем / А.С. Кожанова, О.А. Смірнов, А.В. Челпанов // Проблемні питання розвитку

озброєння та військової техніки Збройних Сил України: IV наук.-техн. конф., 16-20 груд. 2013 р., м. Київ: зб. тез. – Київ: ЦНДІ ОВТ ЗСУ, 2013. – С. 293.

25. Коваленко А.С. Обґрунтування необхідності створення систем технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Інформатика та системні науки : V Всеукр. наук.-практ. конф., 13–15 бер. 2014 р., м. Полтава : зб. тез. – Полтава: ПУЕТ, 2014. – С. 292-294.

26. Коваленко А.С. Задачи распознавания ситуаций в системах организационной стратегии интеграции производства и операций / А.С. Коваленко, А.В. Коваленко // Комбінаторні конфігурації та їх застосування: XVI міжнар. наук.-практ. сем., 11-12 квіт. 2014 р., м. Кіровоград: зб. тез. – Кіровоград: КНТУ, 2014. – С. 53-55.

27. Коваленко А.С. Створення систем технічної діагностики для автоматизації процесів керування в інтегрованих інформаційних системах / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Проблеми і перспективи розвитку ІТ-індустрії: VI між нар. наук.-практ. конф., 17-18 квіт. 2014 р., м. Харків: зб. тез. – Харків: ХНЕУ, 2014. – С. 241.

28. Коваленко А.С. Визначення понятійного апарату та напрямів досліджень для синтезу систем технічної діагностики інтегрованих інформаційних систем / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Комп'ютерне моделювання у наукоємних технологіях (КМНТ-2014): наук.-техн. конф. з міжнар. участю, 28-31 трав. 2014 р., м. Харків: зб. наук. праць. – Харків: ХНУ, 2014. – С. 190-193.

29. Коваленко А.С. Основні складові та функції системи технічної діагностики інтегрованих інформаційних систем / Коваленко А.С. // Інформаційні технології та комп'ютерна інженерія: наук.-практ. конф., 4 груд. 2014 р., м. Кіровоград: зб. тез доп. – Кіровоград: КНТУ, 2014. – С. 236.

30. Коваленко А.С. Розробка структури бази даних інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко //

Проблеми і перспективи розвитку ІТ-індустрії: VII міжнар. наук.-практ. конф., 17-18 квіт. 2015 р., м. Харків: зб. тез. – Харків: ХНЕУ, 2015. – С. 15.

31. Коваленко А.С. Дослідження елементів інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Комбінаторні конфігурації та їх застосування: XVII між нар. наук.-практ. сем., 17-18 квіт. 2015 р., м. Кіровоград: зб. тез – Кіровоград: КНТУ, 2015. – С. 5.

32. Коваленко А.С. Метод автоматизованої перевірки результатів вимірювання параметрів об'єкті в інтегрованої інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Стратегія якості у промисловості і освіті: XI міжнар. конф., 1 – 5 черв. 2015 р., м. Варна, Болгарія.: зб. матер. – Варна: ТУВ, 2015. – С. 423-426.

33. Коваленко А.С. Обґрунтування необхідності створення розподіленої бази даних для забезпечення захисту рухомих повітряних об`єктів / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Перспективні напрями захисту інформації: І всеукр. наук.-практ. конф., 07 вер. 2015 р., м. Одеса: зб. тез доп. – Одеса: ОНАЗ, 2015. – С. 35-39.

34. Коваленко А.С. Розробка інформаційної моделі автоматизованої оцінки технічного стану інтегральної інформаційної системи / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Інформаційні технології та взаємодії (ІТ & І): ІІ між нар. наук.-практ. конф., 3-5 лист. 2015 р., м. Київ: тези доп. – Київ: КНУ ім. Т. Шевченка, 2015. – С. 41-42.

35. Коваленко А.С. Разработка метода усовершенствования технического обслуживания интегрированной информационной системы / А.С. Коваленко, О.А. Смірнов, О.В. Коваленко // Информационные и телекоммуникационные технологии: образование, наука, практика: ІІ междунар. научн.-практ. конф., 3-4 дек. 2015 г., г. Алматы, Казахстан: сб. труд. – Алматы: КазНИТУ им. К.И. Сатпаева, 2015. – Т.2. – С. 423-427.

36. Королюк Н.А. Оценка временных интервалов работы лица, принимающего решение, на автоматизированном командном пункте / Н.А.

Королюк, А.И. Тимочко // Системи обробки інформації. – Х.: ХУПС, 2005. – Вип. 8 (48). – С. 51-54.

37. Костерев В.В. Надёжность технических систем и управление риском: учебн. пособ. / В.В. Костерев. – М.: МИФИ, 2008. – 280 с.

38. Костюков А.В. Підвищення операційної ефективності підприємств на основі моніторингу в реальному часі. / А.В. Костюков, В.М. Костюков. – М.: Машинобудування, 2009. – 192 с.

39. Лазарев А.А. Выбор показателя затрат для анализа сравнительной экономической эффективности техники конечного потребления / А.А. Лазарев, М.В. Бейлин // Сборник научных трудов ХГПУ.– Х.: ХГПУ, 1999. – Вып. 74. – С. 27-29.

40. Ланецкий Б.Н. Основы теории надежности, технического обслуживания и ремонта вооружения и военной техники: Справочные материалы, часть 1. / Б.Н. Ланецкий, А.А. Посудевский. – Харьков: ХВУ, 1993. – 308 с.

41. Ланецкий Б.Н. Основы теории надежности, технического обслуживания и ремонта вооружения и военной техники: Справочные материалы, часть 2. / Б.Н. Ланецкий, А.А. Посудевский. – Харьков: ХВУ, 1993. – 208 с.

42. Лапсарь А.П. Метод оценки состояния сложных технических объектов для синтеза быстродействующих прогнозирующих систем / А.П. Лапсарь // Измерительная техника. – 2004. – № 2. – С. 7-10.

43. Линейные задачи оптимизации: Учеб. пособие / С.В. Лутманов. – Пермь: ЛИТЕР-А, 2004. – Ч.1. – Линейное программирование. – 128 с.

44. Литвак Б.Г. Экспертные технологии в управлении. Учебное пособие / Б.Г. Литвак. – М.: Дело, 2014. – 318 с.

45. Локазюк В.М. Надійність, контроль, діагностика і модернізація ПК: Посібн. / В.М. Локазюк, Ю.Г. Савченко. – К.: Видавничий центр «Академія», 2004. – 376 с.

46. Лопатников Л. И. Экономико-математический словарь: Словарь современной экономической науки / Л.И. Лопатников. – М.: Дело, 2003. – 520 с.

47. Манухина С.Ю. Инженерная психология и эргономика: хрестоматия / С.Ю Манухина. – М.: Изд. центр ЕАОИ, 2009. –224 с.

48. Мартыненко М.В. Человекомашинные процедуры поддержки организационно–управленческих решений: учеб. пособие СПбГЭТУ / М.В. Мартыненко, О.И. Шеховцов. – СПб, 2012. – 250 с.

49. Мунипов О.В. Эргономика: человекоориентированное проектирование техники, программных средств и среды: Учебник / О.В. Мунипов, В.П. Зинченко. – М.: Логос, 2001. – 356 с.

50. Надеев А.И. Математическая модель эксплуатационной надежности интеллектуальных датчиков / А.И. Надеев, Р.А. Юсупов, Ю.К. Свечников, Д.Р. Юсупов // Измерительная техника. – М: Стандартинформ, 2004. – № 1. – С. 8-11.

51. Надійність техніки. Аналіз надійності. Основні положення: ДСТУ 2861-94 – [Чинний від 1997–01–01]. – Київ: Держстандарт України, 1995. – 33 с. – (Національний стандарт України).

52. Надійність техніки. Терміни та визначення: ДСТУ 2860-94 – [Чинний від 1996–01–01]. – Київ: Держстандарт України, 1994. – 36 с. – (Національний стандарт України).

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування.....2
2 Підстава для розробки.....2
3 Мета та призначення розробки.....2
4 Джерела розробки.....2
5 Технічні вимоги.....2
5.1 Вміст проекту.....2
5.2 Показники призначення.....3
5.3 Вимоги до функціональних характеристик.....3
5.4 Вимоги до архітектури.....3
5.5 Вимоги до надійності.....3
5.6 Умови експлуатації.....4
5.7 Вимоги до складу та параметрів технічних засобів.....4
5.8 Вимоги до інформаційної і програмної сумісності.....4
5.8.1 Обладнання.....4
5.8.2 Мова програмування.....4
5.8.3 Вхідні дані.....5
5.8.4 Вихідні дані.....5
6 Вимоги до програмної документації.....5
7 Перелік документів, що розробляються.....5
8 Етапи розробки.....6
9 Порядок контролю та приймання.....6

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи візуалізації процесу діагностики паралельного порту передачі даних.

2 Підстава для розробки

Підставою для розробки служить завдання на випускну кваліфікаційну роботу за першим (бакалаврським) рівнем вищої освіти, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 8-02 від 5.01.2023 року).

3 Мета та призначення розробки

Метою випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є розробка програмного забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних.

4 Джерела розробки

Джерелом цієї випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи візуалізації процесу діагностики паралельного порту передачі даних;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows 10/11 і з сумісними з цією платформою пристроями і прикладним програмним забезпеченням.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows 10/11.

5.8.1 Обладнання

Комп'ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14" 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище Builder C++.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 61 аркуш.

8 Етапи розробки

8.1 Збір і обробка інформації по темі випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти. Постановка задачі на виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

9 Порядок контролю та приймання

9.1 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на попередній захист 23.05.2023 р.

9.2 Подання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти на захист 8.06.2023 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник випускної кваліфікаційної роботи за

першим (бакалаврським) рівнем вищої освіти

_____ Коваленко О.В.

Програмне забезпечення системи візуалізації процесу діагностики паралельного порту передачі даних

Лістинг програми

Код документу 12

Носій: CD/DVD-диск / USB-флеш-накопичувач

Загальна кількість аркушів: 28

Літера: РП

Кропивницький – 2023 року

Основна програма

Файл Test_LPT.cpp основної програми

```
//---------------------------------------------------------------------
#include <vcl.h>
#include "winio.h"
#pragma hdrstop
USEFORM("Unit1.cpp", Form1);
USEFORM("Byte.cpp", Frame2); /* TFrame: File Type */
USEFORM("Registr.cpp", Frame3); /* TFrame: File Type */
USEFORM("About.cpp", AboutBox);
USEFORM("Resursi.cpp", Resurses);
USEFORM("ChangeRegData.cpp", ChangeRegDat);
USEFORM("ChangeRegControl.cpp", ChangeRegContr);
USEFORM("Help.cpp", Form1);

//---------------------------------------------------------------------

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
   bool bResult;  // результат ініціалізації WinIO
   bResult = InitializeWinIo();
   if (bResult == false)
   {
      ShowMessage("Не вдалося здійснити ініціалізацію WinIO!");
   }
   else
   {
      try
      {
         Application->Initialize();
         Application->Title = "Test_LPT";
         Application->CreateForm(__classid(TForm1), &Form1);
                 Application->CreateForm(__classid(TAboutBox), &AboutBox);
                 Application->CreateForm(__classid(TResurses), &Resurses);
                 Application->CreateForm(__classid(TChangeRegDat),
&ChangeRegDat);
                 Application->CreateForm(__classid(TChangeRegContr),
&ChangeRegContr);
                 Application->CreateForm(__classid(TForm2), &Form2);
                 Application->Run();
      }
      catch (Exception &exception)
      {
         Application->ShowException(&exception);
      }
      ShutdownWinIo();
   }
   return 0;
}
//---------------------------------------------------------------------
```

Файл Unit1.cpp основної програми

```
//---------------------------------------------------------------------
#include <vcl.h>
#include <math.h>
#pragma hdrstop

#include "Unit1.h"
#include "Help.h"
#include "About.h"
#include "Resursi.h"
#include "ChangeRegData.h"
#include "hex_dec_oct.h"
#include "ChangeRegControl.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "Byte"
#pragma link "Registr"
#pragma resource "*.dfm"
TForm1 *Form1;
WORD lpt_address; // адреса вибраного LPT порту
//---------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm1::changeLabel(void)
{
    Frame31->Label1->Caption = "0x" + IntToHex(lpt_address, 4);
    Frame31->address = lpt_address;
    Frame32->Label1->Caption = "0x" + IntToHex(lpt_address+1, 4);
    Frame32->address = lpt_address+1;
    Frame33->Label1->Caption = "0x" + IntToHex(lpt_address+2, 4);
    Frame33->address = lpt_address+2;
}
//---------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    DWORD dwMemVal;
    short x=0;
    GetPhysLong((PBYTE)0x0408, &dwMemVal);
    WORD lpt1, lpt2, lpt3;
    lpt1 = (WORD)dwMemVal;
    if (lpt1 > 0)
        x = 1;
    GetPhysLong((PBYTE)0x040a, &dwMemVal);
    lpt2 = (WORD)dwMemVal;
    if ((lpt2>0) & (x==0))
        x = 2;
    GetPhysLong((PBYTE)0x040c, &dwMemVal);
    lpt3 = (WORD)dwMemVal;
    if ((lpt3>0) & (x==0))
        x = 3;
    switch (x)
    {
        case 1: ComboBox1->Text = "LPT1";
                lpt_address = lpt1;
                Edit1->Text = "0x" + IntToHex(lpt1, 4);
                ComboBox1->Items->Add("LPT1");
                changeLabel();
                if (lpt2>0)
                    ComboBox1->Items->Add("LPT2");
                if (lpt3>0)
                    ComboBox1->Items->Add("LPT3");
                break;
```

```
        case 2: ComboBox1->Text = "LPT2";
                lpt_address = lpt2;
                Edit1->Text = "0x" + IntToHex(lpt2, 4);
                ComboBox1->Items->Add("LPT2");
                changeLabel();
                if (lpt3>0)
                    ComboBox1->Items->Add("LPT3");
                break;
        case 3: ComboBox1->Text = "LPT3";
                lpt_address = lpt3;
                Edit1->Text = "0x" + IntToHex(lpt3, 4);
                ComboBox1->Items->Add("LPT3");
                changeLabel();
                break;
        default: ComboBox1->Text = "No LPT!";
    }
    Frame31->process(Sender);
    Frame32->process(Sender);
    Frame33->Frame22->CheckBox7->State = cbChecked;
    Frame33->Frame22->CheckBox8->State = cbChecked;
    Frame33->process(Sender);
    Frame32->Frame22->Label1->Caption = Frame32->Frame21->Label1->Caption;
}
//---------------------------------------------------------------------

void __fastcall TForm1::ComboBox1Change(TObject *Sender)
{
    short vibor;
    if (ComboBox1->Text=="LPT1")    vibor=1;
    if (ComboBox1->Text=="LPT2")    vibor=2;
    if (ComboBox1->Text=="LPT3")    vibor=3;
    DWORD dwMemVal;
    switch (vibor)
    {
        case 1: GetPhysLong((PBYTE)0x0408, &dwMemVal);
                lpt_address = (WORD)dwMemVal;
                Edit1->Text = "0x" + IntToHex(lpt_address, 4);
                changeLabel();
                break;
        case 2: GetPhysLong((PBYTE)0x040a, &dwMemVal);
                lpt_address = (WORD)dwMemVal;
                Edit1->Text = "0x" + IntToHex(lpt_address, 4);
                changeLabel();
                break;
        case 3: GetPhysLong((PBYTE)0x040c, &dwMemVal);
                lpt_address = (WORD)dwMemVal;
                Edit1->Text = "0x" + IntToHex(lpt_address, 4);
                changeLabel();
                break;
    }
}
//---------------------------------------------------------------------

void __fastcall TForm1::N7Click(TObject *Sender)
{
        AboutBox->ShowModal();
}
//---------------------------------------------------------------------

void __fastcall TForm1::N4Click(TObject *Sender)
{
        Resurses->ShowModal();
}
//---------------------------------------------------------------------


void __fastcall TForm1::N8Click(TObject *Sender)
{
    Close();
```

```
}
//---------------------------------------------------------------------

void __fastcall TForm1::DR1Click(TObject *Sender)
{
    ChangeRegDat->ShowModal();
    ChangeRegDat->Edit1->Text = "";
}
//---------------------------------------------------------------------

void __fastcall TForm1::N6Click(TObject *Sender)
{
    ChangeRegContr->ShowModal();
    ChangeRegContr->Edit1->Text = "";
}
//---------------------------------------------------------------------

void __fastcall TForm1::ExitClick(TObject *Sender)
{
    Close();
}
//---------------------------------------------------------------------

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Frame32->process(Sender);
}
//---------------------------------------------------------------------

void __fastcall TForm1::SpeedButton3Click(TObject *Sender)
{
    bool io = true;
    int in, out;
    for (int i=0; i<9; i++)
    {
        if (i==8)
            in = 0;
        else
            in = pow(2, i);
        Frame31->Frame22->Tag = in;
        Frame31->Frame22->GetTag();
        Frame31->process(Sender);
        out = Frame31->Frame21->Tag;
        if (in != out)
            io = false;
        Sleep(500);
    }
    if (io)
        ShowMessage("Тестування регістра даних пройшло УСПІШНО!");
    else
        ShowMessage("Тестування регістра даних пройшло НЕ УСПІШНО!");
    for (int i=0; i<6; i++)
    {
        if (i==5)
            in = 192;
        else
            in = 192 + pow(2, i);
        Frame33->Frame22->Tag = in;
        Frame33->Frame22->GetTag();
        Frame33->process(Sender);
        out = Frame33->Frame21->Tag;
        if (in != out)
            io = false;
        Sleep(500);
    }
    if (io)
        ShowMessage("Тестування регістра керування пройшло УСПІШНО!");
    else
        ShowMessage("Тестування регістра керування пройшло НЕ УСПІШНО!");
```

```
}
//---------------------------------------------------------------------

void __fastcall TForm1::SpeedButton6Click(TObject *Sender)
{
Form2->Show();
}
//---------------------------------------------------------------------

void __fastcall TForm1::N9Click(TObject *Sender)
{
Form2->Show();
}
//---------------------------------------------------------------------
```

Файл Unit1.h – бібліотека для файлу Unit1.cpp

```
//---------------------------------------------------------------------

#ifndef Unit1H
#define Unit1H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include "Byte.h"
#include "Registr.h"
#include "WinIO.h"
#include <ExtCtrls.hpp>
#include <Grids.hpp>
#include <Buttons.hpp>
//---------------------------------------------------------------------
class TForm1 : public TForm
{
__published:      // IDE компоненти керування
    TMainMenu *MainMenu1;
    TMenuItem *N1;
    TMenuItem *N2;
    TMenuItem *N3;
    TMenuItem *N4;
    TMenuItem *N5;
    TMenuItem *DR1;
    TMenuItem *N6;
    TMenuItem *N7;
         TComboBox *ComboBox1;
         TLabel *Label1;
         TLabel *Label2;
         TEdit *Edit1;
         TPanel *Panel1;
         TLabel *Label3;
         TLabel *Label4;
         TLabel *Label5;
         TLabel *Label6;
         TFrame3 *Frame32;
         TFrame3 *Frame33;
     TMenuItem *N8;
     TSpeedButton *Exit;
     TSpeedButton *SpeedButton1;
     TSpeedButton *SpeedButton2;
     TSpeedButton *SpeedButton3;
     TSpeedButton *SpeedButton5;
     TTimer *Timer1;
         TLabel *Label9;
         TLabel *Label10;
         TLabel *Label11;
         TFrame3 *Frame31;
         TSpeedButton *SpeedButton4;
```

```
        TLabel *Label7;
        TLabel *Label8;
        TSpeedButton *SpeedButton6;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TCheckBox *CheckBox9;
        TCheckBox *CheckBox10;
        TCheckBox *CheckBox11;
        TCheckBox *CheckBox12;
        TMenuItem *N9;
    void __fastcall FormCreate(TObject *Sender);
    void __fastcall ComboBox1Change(TObject *Sender);
    void __fastcall N7Click(TObject *Sender);
    void __fastcall N4Click(TObject *Sender);
    void __fastcall N8Click(TObject *Sender);
    void __fastcall DR1Click(TObject *Sender);
    void __fastcall N6Click(TObject *Sender);
    void __fastcall ExitClick(TObject *Sender);
    void __fastcall Timer1Timer(TObject *Sender);
    void __fastcall SpeedButton3Click(TObject *Sender);
        void __fastcall SpeedButton6Click(TObject *Sender);
        void __fastcall N9Click(TObject *Sender);
private:    // задаэться користувачем
    void __fastcall changeLabel(void);
public:           // задається користувачем
    __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------
#endif
```

Робота з регістрами

Файл Registr.cpp – ініціалізація регістрів

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Registr.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma link "Byte"
#pragma resource "*.dfm"
extern WORD lpt_address;
TFrame3 *Frame3;
//---------------------------------------------------------------------
__fastcall TFrame3::TFrame3(TComponent* Owner)
        : TFrame(Owner)
{
        Frame21->io = false;
        Frame22->io = true;
}
//---------------------------------------------------------------------
void __fastcall TFrame3::process(TObject *Sender)
{
    byte port_data = Frame22->Tag;
    if (address == lpt_address+2)
        port_data ^= 11;
    SetPortVal(address, port_data, 1);
    DWORD d_port_data;
    GetPortVal(address, &d_port_data, 1);
    port_data = (byte)d_port_data;
    if (address == lpt_address+1)
        port_data ^= 128;
    if (address == lpt_address+2)
        port_data ^= 11;
    Frame21->Tag = port_data;
    Frame21->CheckBox1Click(Sender);
}
//---------------------------------------------------------------------
void __fastcall TFrame3::Frame22CheckBox1Click(TObject *Sender)
{
    Frame22->CheckBox1Click(Sender);
    process(Sender);
}
//---------------------------------------------------------------------
```

Файл Registr.h – бібліотека для файлу Registr.cpp

```
//---------------------------------------------------------------------
#ifndef RegistrH
#define RegistrH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include "Byte.h"
#include "winio.h"
//---------------------------------------------------------------------
class TFrame3 : public TFrame
{
__published:      // IDE- компоненти керування
        TPanel *Panel1;
        TLabel *Label1;
        TLabel *Label2;
        TFrame2 *Frame21;
        TFrame2 *Frame22;
    TLabel *Label3;
    void __fastcall Frame22CheckBox1Click(TObject *Sender);
private:
public:
        __fastcall TFrame3(TComponent* Owner);
        WORD address;
        void __fastcall process(TObject *Sender);
};
//---------------------------------------------------------------------
extern PACKAGE TFrame3 *Frame3;
//---------------------------------------------------------------------#endif
```

Файл Byte.cpp – відображення значень регістрів

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "Byte.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TFrame2 *Frame2;
//---------------------------------------------------------------------
void TFrame2::SetTag(void)
{
    Tag = 0;
    if (CheckBox1->State == cbChecked)   Tag += 1;
    if (CheckBox2->State == cbChecked)   Tag += 2;
    if (CheckBox3->State == cbChecked)   Tag += 4;
    if (CheckBox4->State == cbChecked)   Tag += 8;
    if (CheckBox5->State == cbChecked)   Tag += 16;
    if (CheckBox6->State == cbChecked)   Tag += 32;
    if (CheckBox7->State == cbChecked)   Tag += 64;
    if (CheckBox8->State == cbChecked)   Tag += 128;
    Label1->Caption = AnsiString(Tag);
    Label1->Hint = "0x"+IntToHex(Tag, 2);
}
//---------------------------------------------------------------------
void TFrame2::GetTag(void)
{
    div_t x;
    byte i = Tag;
    x = div(i, 2);
    if (x.rem == 1) CheckBox1->State = cbChecked;
    else CheckBox1->State = cbUnchecked;
    i = x.quot;
    x = div (i, 2);
    if (x.rem == 1) CheckBox2->State = cbChecked;
    else CheckBox2->State = cbUnchecked;
    i = x.quot;
    x = div (i, 2);
    if (x.rem == 1) CheckBox3->State = cbChecked;
    else CheckBox3->State = cbUnchecked;
    i = x.quot;
    x = div (i, 2);
    if (x.rem == 1) CheckBox4->State = cbChecked;
    else CheckBox4->State = cbUnchecked;
    i = x.quot;
    x = div (i, 2);
    if (x.rem == 1) CheckBox5->State = cbChecked;
    else CheckBox5->State = cbUnchecked;
    i = x.quot;
    x = div (i, 2);
    if (x.rem == 1) CheckBox6->State = cbChecked;
    else CheckBox6->State = cbUnchecked;
    i = x.quot;
    x = div (i, 2);
    if (x.rem == 1) CheckBox7->State = cbChecked;
    else CheckBox7->State = cbUnchecked;
    i = x.quot;
    x = div (i, 2);
    if (x.rem == 1) CheckBox8->State = cbChecked;
    else CheckBox8->State = cbUnchecked;
}
//---------------------------------------------------------------------
void __fastcall TFrame2::if_else(void)
{
    if (io)
    {
        SetTag();
```

```
    }
    else
    {
        GetTag();
    }
    Label1->Caption = "0x"+IntToHex(Tag, 2);
    Label1->Hint = AnsiString(Tag);
}
//---------------------------------------------------------------------
__fastcall TFrame2::TFrame2(TComponent* Owner)
   : TFrame(Owner)
{
    if_else();
}
//---------------------------------------------------------------------

void __fastcall TFrame2::CheckBox1Click(TObject *Sender)
{
    if_else();
}
//---------------------------------------------------------------------
```

Файл Byte.h – бібліотека для файлу Byte.cpp

```
//---------------------------------------------------------------------
#ifndef ByteH
#define ByteH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include "winio.h"
//---------------------------------------------------------------------
class TFrame2 : public TFrame
{
__published:      // IDE компоненти керування
   TPanel *Panel1;
   TCheckBox *CheckBox1;
   TCheckBox *CheckBox2;
   TCheckBox *CheckBox3;
   TCheckBox *CheckBox4;
   TCheckBox *CheckBox5;
   TCheckBox *CheckBox6;
   TCheckBox *CheckBox7;
   TCheckBox *CheckBox8;
   TLabel *Label1;

    void __fastcall CheckBox1Click(TObject *Sender);
private:    // Визначається користувачем
   void SetTag(void);   // CheckBox's -> tag
   void __fastcall if_else(void);
public:           // Визначається користувачем
   __fastcall TFrame2(TComponent* Owner);
   bool io;     // io = true – дозволена зміна CheckBox'ов
                // io = false – заборонена зміна CheckBox'ов
   void GetTag(void);   // tag -> CheckBox's
};
//---------------------------------------------------------------------
extern PACKAGE TFrame2 *Frame2;
//---------------------------------------------------------------------#endif
```

Файл ChangeRegData.cpp – введення значення регістру даних

```
//---------------------------------------------------------------------#include
<vcl.h>
#pragma hdrstop

#include "ChangeRegData.h"
#include "hex_dec_oct.h"
#include "Unit1.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TChangeRegDat *ChangeRegDat;
//---------------------------------------------------------------------
__fastcall TChangeRegDat::TChangeRegDat(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TChangeRegDat::Button1Click(TObject *Sender)
{
    AnsiString data;
    data = Edit1->Text;
    char *s;
   s[0]=0; s[1]=0; s[2]=0; s[3]=0;
    s = data.c_str();
    byte RegData;
    if ((s[0] == '0')&(data != '0'))
    {
        if ((s[1] == 'x') | (s[1] == 'X'))
            RegData = hex(s[2], s[3]);
        else
        {
            if (s[1] != NULL)  RegData = oct(s[1], s[2], s[3]);
        }
    }
    else   RegData = dec(s[0], s[1], s[2], s[3]);
    Form1->Frame31->Frame22->Tag = RegData;
    Form1->Frame31->Frame22->GetTag();
    Form1->Frame31->process(Sender);
    Close();
}
//---------------------------------------------------------------------

void __fastcall TChangeRegDat::Button2Click(TObject *Sender)
{
    Close();
}
//---------------------------------------------------------------------

void __fastcall TChangeRegDat::FormShow(TObject *Sender)
{
    ActiveControl = Edit1;
}
//---------------------------------------------------------------------

void __fastcall TChangeRegDat::Edit1KeyPress(TObject *Sender, char &Key)
{
    if (Key==13)
        Button1Click(Sender);
}
//---------------------------------------------------------------------
```

Файл ChangeRegData.h – бібліотека для файлу ChangeRegData.cpp

```
//---------------------------------------------------------------------

#ifndef ChangeRegDataH
#define ChangeRegDataH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------
class TChangeRegDat : public TForm
{
__published:      // IDE-managed Components
    TPanel *Panel1;
    TLabel *Label1;
    TEdit *Edit1;
    TButton *Button1;
    TButton *Button2;
    TLabel *Label2;
    TLabel *Label3;
    TLabel *Label4;
    TLabel *Label5;
    TLabel *Label6;
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
    void __fastcall FormShow(TObject *Sender);
    void __fastcall Edit1KeyPress(TObject *Sender, char &Key);
private:
public:
    __fastcall TChangeRegDat(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TChangeRegDat *ChangeRegDat;
//---------------------------------------------------------------------
#endif
```

Файл ChangeRegControl.cpp – введення значення регістру керування

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "ChangeRegControl.h"
#include "hex_dec_oct.h"
#include "Unit1.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TChangeRegContr *ChangeRegContr;
//---------------------------------------------------------------------
__fastcall TChangeRegContr::TChangeRegContr(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TChangeRegContr::Button1Click(TObject *Sender)
{
    AnsiString data;   int r;
    data = Edit1->Text;
    r=StrToInt(data);
    if(r>31){
      ShowMessage("Значення перевищило 31! Воно буде замінене на 0.");
      data = '0'; }
          char *s;
    s[1]=0; s[2]=0; s[3]=0;
    s = data.c_str();
    byte RegData;

   if ((s[0] == '0')&(data != '0'))
    {
        if ((s[1] == 'x') | (s[1] == 'X'))
            RegData = hex(s[2], s[3]);
        else
        {
            if (s[1] != NULL)  RegData = oct(s[1], s[2], s[3]);
        }
    }
    else    RegData = dec(s[0], s[1], s[2], s[3]);
    Form1->Frame33->Frame22->Tag = RegData | 192;
    Form1->Frame33->Frame22->GetTag();
    Form1->Frame33->process(Sender);
    Close();
}
//---------------------------------------------------------------------

void __fastcall TChangeRegContr::Button2Click(TObject *Sender)
{
    Close();
}
//---------------------------------------------------------------------

void __fastcall TChangeRegContr::FormShow(TObject *Sender)
{
    ActiveControl = Edit1;
}
//---------------------------------------------------------------------

void __fastcall TChangeRegContr::Edit1KeyPress(TObject *Sender, char &Key)
{
    if (Key==13)
        Button1Click(Sender);
}
//---------------------------------------------------------------------
```

Файл ChangeRegControl.h – бібліотека для файлу ChangeRegControl.cpp

```
//---------------------------------------------------------------------

#ifndef ChangeRegControlH
#define ChangeRegControlH
//---------------------------------------------------------------------#include
<Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------
class TChangeRegContr : public TForm
{
__published:      // IDE- компоненти керування
    TPanel *Panel1;
    TLabel *Label1;
    TEdit *Edit1;
    TButton *Button1;
    TButton *Button2;
    TLabel *Label2;
    TLabel *Label3;
    TLabel *Label4;
    TLabel *Label5;
    TLabel *Label6;
    void __fastcall Button1Click(TObject *Sender);
    void __fastcall Button2Click(TObject *Sender);
    void __fastcall FormShow(TObject *Sender);
    void __fastcall Edit1KeyPress(TObject *Sender, char &Key);
private:    // задається користувачем
public:           // задається користувачем
    __fastcall TChangeRegContr(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TChangeRegContr *ChangeRegContr;
//---------------------------------------------------------------------
#endif
```

Файл hex_dec_oct.cpp – забезпечує роботу з числами у шістнадцятковій та вісімковій системі ліку

```
//---------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "hex_dec_oct.h"

//---------------------------------------------------------------------

#pragma package(smart_init)


byte switch_case(byte ci, char ch)      // ci = 0 - oct
{                                       // ci = 1 - dec
    int ch_i = (int)ch;                      // ci = 2 - hex
    byte x;
    if (ci==2)
        switch (ch_i)
        {
            case '0':   case ' ':   case '\0':  x = 0;   break;
            case '1':   x = 1;   break;
            case '2':   x = 2;   break;
            case '3':   x = 3;   break;
            case '4':   x = 4;   break;
            case '5':   x = 5;   break;
            case '6':   x = 6;   break;
            case '7':   x = 7;   break;
            case '8':   x = 8;   break;
            case '9':   x = 9;   break;
            case 'a':   case 'A':   x = 10;   break;
            case 'b':   case 'B':   x = 11;   break;
            case 'c':   case 'C':   x = 12;   break;
            case 'd':   case 'D':   x = 13;   break;
            case 'e':   case 'E':   x = 14;   break;
            case 'f':   case 'F':   x = 15;   break;
            default:    ShowMessage("Це не шістнадцяткове число!");
                        x = 21;    // 21 - код помилки
        }
    if (ci==1)
        switch (ch_i)
        {
            case '0':   case ' ':   case '\0':  x = 0;   break;
            case '1':   x = 1;   break;
            case '2':   x = 2;   break;
            case '3':   x = 3;   break;
            case '4':   x = 4;   break;
            case '5':   x = 5;   break;
            case '6':   x = 6;   break;
            case '7':   x = 7;   break;
            case '8':   x = 8;   break;
            case '9':   x = 9;   break;
            default:    ShowMessage("Це не десяткове число!");
                        x = 21;
        }
    if (ci==0)
        switch (ch_i)
        {
            case '0':   x = 0;   break;
            case '1':   x = 1;   break;
            case '2':   x = 2;   break;
            case '3':   x = 3;   break;
            case '4':   x = 4;   break;
            case '5':   x = 5;   break;
            case '6':   x = 6;   break;
```

```
            case '7':   x = 7;   break;
            case ' ':   case '\0':  x = 0;   break;
            default:    ShowMessage("Це не вісімкове число!");
                        x = 21;
        }
    return x;
}
//---------------------------------------------------------------------
byte hex(char s3, char s4)
{
    byte s3_i, s4_i, hex_temp=0;
    s3_i = switch_case (2, s3);
    if (s3_i == 21)
        return hex_temp = 0;
    s4_i = switch_case (2, s4);
    if (s4_i == 21)
        return hex_temp = 0;
    if ((s3==' ')|(s3=='\0'))      hex_temp = 0;
        else
            if ((s4==' ')|(s4=='\0'))      hex_temp = s3_i;
            else                hex_temp = s3_i*16 + s4_i;
    return hex_temp;
}
//---------------------------------------------------------------------
byte oct(char s2, char s3, char s4)
{
    byte s2_i, s3_i, s4_i;
    int oct_temp;
    s2_i = switch_case (0, s2);
    if (s2_i == 21)
        return oct_temp = 0;
    s3_i = switch_case (0, s3);
    if (s3_i == 21)
        return oct_temp = 0;
    s4_i = switch_case (0, s4);
    if (s4_i == 21)
        return oct_temp = 0;
    if ((s2==' ')|(s2=='\0'))      oct_temp = 0;
        else
            if ((s3==' ')|(s3=='\0'))      oct_temp = s2_i;
            else
                if ((s4==' ')|(s4=='\0'))      oct_temp = s2_i*8 + s3_i;
                else                oct_temp = s2_i*64 + s3_i*8 + s4_i;
    if (oct_temp > 255)
    {
        ShowMessage("Значення перевищило 255! Воно буде замінене на 0.");
        oct_temp = 0;
    }
    return (byte)oct_temp;
}
//---------------------------------------------------------------------byte dec(char
s1, char s2, char s3, char s4)
{
    byte s1_i, s2_i, s3_i, s4_i;
    int dec_temp;
    s1_i = switch_case (1, s1);
    if (s1_i == 21)
        return dec_temp = 0;
    s2_i = switch_case (1, s2);
    if (s2_i == 21)
        return dec_temp = 0;
    s3_i = switch_case (1, s3);
    if (s3_i == 21)
        return dec_temp = 0;
    s4_i = switch_case (1, s4);
    if (s4_i == 21)
        return dec_temp = 0;
    if ((s1==' ')|(s1=='\0'))      dec_temp = 0;
        else
```

```
            if ((s2==' ')|(s2=='\0'))      dec_temp = s1_i;
            else
                if ((s3==' ')|(s3=='\0'))      dec_temp = s1_i*10+s2_i;
                else
                    if ((s4==' ')|(s4=='\0'))      dec_temp =
s1_i*100+s2_i*10+s3_i;
                    else      dec_temp = s1_i*1000+s2_i*100+s3_i*10+s4_i;
    if (dec_temp > 255)
    {
        ShowMessage("Значення перевищило 255! Воно буде замінене на 0.");
        dec_temp = 0;
    }
    return (byte)dec_temp;
}
//---------------------------------------------------------------------
```

Файл hex_dec_oct.h – бібліотека для файлу hex_dec_oct.cpp

```
//---------------------------------------------------------------------
#ifndef hex_dec_octH
#define hex_dec_octH
//---------------------------------------------------------------------
#endif
byte switch_case (byte, char); //визначення системи ліку числа
byte hex (char, char); //переведення рядка в 16-ве число
byte oct (char, char, char); //переведення рядка в 8-ве число
byte dec (char, char, char, char); //переведення рядка в 10-ве //число
```

Додаткові функції програми

Файл Resursi.cpp – відображення наявних на комп'ютері LPT-портів

```
//---------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Resursi.h"
#include "WinIO.h"
//---------------------------------------------------------------------

#pragma package(smart_init)
#pragma resource "*.dfm"
TResurses *Resurses;

//---------------------------------------------------------------------
__fastcall TResurses::TResurses(TComponent* Owner)
        : TForm(Owner)
{
}

//---------------------------------------------------------------------
void __fastcall TResurses::Button1Click(TObject *Sender)
{
        Close();
}

//---------------------------------------------------------------------

void __fastcall TResurses::FormCreate(TObject *Sender)
{
    DWORD dwMemVal;
    byte count_port=0;
    GetPhysLong((PBYTE)0x0408, &dwMemVal);
    WORD lpt1, lpt2, lpt3;
    lpt1 = (WORD)dwMemVal;
    if (lpt1 > 0)
    {
        Resurses->Label_lpt1->Caption = "0x" + IntToHex(lpt1, 4);
        count_port++;
    }
    else
        Resurses->Label_lpt1->Caption = "Немає доступу";
    GetPhysLong((PBYTE)0x040a, &dwMemVal);
    lpt2 = (WORD)dwMemVal;
    if (lpt2 > 0)
    {
        Resurses->Label_lpt2->Caption = "0x" + IntToHex(lpt2, 4);
        count_port++;
    }
    else
        Resurses->Label_lpt2->Caption = "Немає доступу";
    GetPhysLong((PBYTE)0x040c, &dwMemVal);
    lpt3 = (WORD)dwMemVal;
    if (lpt3 > 0)
    {
        Resurses->Label_lpt3->Caption = "0x" + IntToHex(lpt3, 4);
        count_port++;
    }
    else
        Resurses->Label_lpt3->Caption = "Немає доступу";
    Label7->Caption = AnsiString(count_port);
}
//---------------------------------------------------------------------
```

Файл Resursi.h – бібліотека для файлу Resursi.cpp

```
//---------------------------------------------------------------------------

#ifndef ResursiH
#define ResursiH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TResurses : public TForm
{
__published:      // IDE компоненти керування
        TPanel *Panel1;
        TLabel *Label1;
        TButton *Button1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label_lpt1;
        TLabel *Label_lpt2;
        TLabel *Label_lpt3;
    TLabel *Label7;
    TLabel *Label8;
        void __fastcall Button1Click(TObject *Sender);
    void __fastcall FormCreate(TObject *Sender);
private:    // задається користувачем
public:           // задається користувачем
        __fastcall TResurses(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TResurses *Resurses;
//---------------------------------------------------------------------------
                                    #endif
```

Файл About.cpp – відображення вікна "Про програму..."

```
//---------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "About.h"
//---------------------------------------------------------------------
#pragma resource "*.dfm"
TAboutBox *AboutBox;
//---------------------------------------------------------------------
__fastcall TAboutBox::TAboutBox(TComponent* AOwner)
      : TForm(AOwner)
{
}
//---------------------------------------------------------------------
```

Файл About.h – бібліотека для файлу About.cpp

```
//---------------------------------------------------------------------
#ifndef AboutH
#define AboutH
//---------------------------------------------------------------------
#include <vcl\System.hpp>
#include <vcl\Windows.hpp>
#include <vcl\SysUtils.hpp>
#include <vcl\Classes.hpp>
#include <vcl\Graphics.hpp>
#include <vcl\Forms.hpp>
#include <vcl\Controls.hpp>
#include <vcl\StdCtrls.hpp>
#include <vcl\Buttons.hpp>
#include <vcl\ExtCtrls.hpp>
//---------------------------------------------------------------------
class TAboutBox : public TForm
{
__published:
      TPanel *Panel1;
      TLabel *ProductName;
      TLabel *Copyright;
      TButton *OKButton;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
private:
public:
      virtual __fastcall TAboutBox(TComponent* AOwner);
};
//---------------------------------------------------------------------
extern PACKAGE TAboutBox *AboutBox;
//---------------------------------------------------------------------
#endif
```

Файл Help.cpp – відображення вікна Довідки про LPT-порт

```
//---------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop

#include "Help.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
//---------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm2::Button1Click(TObject *Sender)
{
Form2->Close();
}
//---------------------------------------------------------------------
```

Файл Help.h – бібліотека для файлу Help.cpp

```
#ifndef HelpH
#define HelpH
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
//---------------------------------------------------------------------
class TForm2 : public TForm
{
__published:      // IDE-managed Components
        TImage *Image1;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TLabel *Label11;
        TLabel *Label12;
        TLabel *Label13;
        TLabel *Label14;
        TLabel *Label15;
        TLabel *Label16;
        TLabel *Label19;
        TLabel *Label20;
        TLabel *Label21;
        TLabel *Label22;
        TLabel *Label23;
        TLabel *Label24;
        TLabel *Label25;
        TLabel *Label26;
        TLabel *Label27;
        TLabel *Label28;
        TLabel *Label29;
        TLabel *Label30;
        TLabel *Label31;
        TLabel *Label17;
        TLabel *Label18;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:    // User declarations
public:           // User declarations
        __fastcall TForm2(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm2 *Form2;
//---------------------------------------------------------------------
#endif
```