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 програмування послідовного порту у навчальних цілях”

Виконав здобувач вищої освіти
IV курсу, групи КМ-19
ОПП «Комп’ютерна інженерія»
спеціальності 123 «Комп’ютерна інженерія»
_____ Хрусталенко С.М.
«____» _____ 2023 р.

Керівник проекту
кандидат технічних наук
_____ Буравченко К.О.
«____» _____ 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

Хрусталенку Станіславу Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи візуалізації програмування послідовного порту у навчальних цілях*

2. Керівник роботи *Буравченко Костянтин Олегович, канд. техн. наук*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 10-02 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 програмування послідовного порту у навчальних цілях*

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
_____ Буравченко К.О.
(прізвище та ініціали)

Завдання прийнято до виконання
« 17 » січня 2023 р.

Підпис здобувача
_____ Хрусталенко С.М.
(прізвище та ініціали)

АНОТАЦІЯ

Хрусталенко С.М. Програмне забезпечення системи візуалізації програмування послідовного порту у навчальних цілях. 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 програмування послідовного порту у навчальних цілях.

Метою розробки є програмне забезпечення системи візуалізації програмування послідовного порту у навчальних цілях.

Результат роботи – програмна реалізація системи візуалізації програмування послідовного порту у навчальних цілях.

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, послідовний порт

ABSTRACT

Khrustalenko S.M. Serial port programming visualization system software for educational purposes. 123 Computer engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2023.

In this graduation thesis for the first (bachelor) level of higher education, software is developed, which is intended for the visualization system of serial port programming for educational purposes.

The goal of the development is a serial port programming visualization system software for educational purposes.

The result of the work is the software implementation of the serial port programming visualization system for educational purposes.

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, serial port

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ 2
ВСТУП..... 3
1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ 6
1.1 Призначення системи..... 6
1.2 Область застосування..... 8
2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ 11
2.1 Огляд існуючих систем, технологій, архітектур та програмних рішень за профілем теми випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти..... 11
2.2 Обґрунтування вибору засобів для побудови системи та мови програмування..... 25
2.3 Розгорнута постановка завдання 27
3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ 28
3.1 Опис функціонування системи 28
3.2 Розробка структурної схеми..... 52
3.3 Розробка функціональної схеми 55
3.4 Розробка діаграми процесів..... 57
4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ..... 59
4.1 Розробка блок-схем та опис алгоритмів функціонування системи..... 59
4.2 Захист розробленого програмного забезпечення..... 67
5 ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ..... 70
6 ОСНОВНІ ВИСНОВКИ..... 75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 77

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ІТ	–	інформаційні технології
ОТ	–	обчислювальна техніка
ПЗ	–	програмне забезпечення
ПІТ	–	перспективні інформаційні технології

ВСТУП

Актуальність теми. Тенденції розвитку сучасного суспільства, його яскраво виражена інформатизація пояснюють необхідність усе більше широкого використання інформаційних технологій у сфері освіти. У цей час культурній людині незалежно від її професії й особливостей діяльності необхідно мати вміння роботи з електронними засобами обробки й передачі інформації.

Сучасні інформаційні технології є основою процесу інформатизації навчання, реалізація якого припускає:

– поліпшення якості навчання за допомогою більше повного використання доступної інформації;

– підвищення ефективності навчального процесу на основі його індивідуалізації й інтенсифікації;

– розробку перспективних засобів, методів і технологій навчання з орієнтацією на розвиваюче, випереджальне й персоніфіковане утворення;

– досягнення необхідного рівня професіоналізму в оволодінні засобами інформатики й обчислювальної техніки;

– інтеграцію різних видів діяльності (навчальної, учбово-дослідницької, методичної, наукової, організаційної) у рамках єдиної методології, заснованої на застосуванні інформаційних технологій;

– підготовку учасників освітнього процесу до життєдіяльності в умовах інформаційного суспільства;

– підвищення професійної компетентності й конкуренто здатності майбутніх фахівців різних галузей;

– подолання кризових явищ у системі утворення.

Використовуючи комп'ютер, можна вивчити процеси, які в умовах навчального кабінету продемонструвати неможливо, або занадто дорого або небезпечно. Практика показує, що на етапі тренування, де переважає самостійна

робота, комп'ютер має більші переваги, допомагаючи здійснити диференційований підхід до кожного учня, вчасно помітити пробіли в знаннях і усунути їх. Володіючи "нескінченним терпінням", машина ніколи не "утомлюється", і, якщо це буде потрібно, вона може повторювати вправи багаторазово. Таким чином, виділимо наступні основні умови реалізації принципу доцільності створення навчальних програм:

– установлення тих властивостей і можливостей наявної комп'ютерної техніки, які дозволять підвищити якість навчання;

– обґрунтований вибір змісту, методів і форм комп'ютеризованого навчання, їхня відповідність цілям навчального процесу й раціональному використанню можливостей ЕОМ;

– чітке визначення конкретної ролі, завдань, місця й часу застосування навчальної програми;

– установлення зв'язків і відносин з іншими засобам і методами навчання;

– ретельна організація й технічна бездоганність функціонування навчальної програми.

Одним з напрямків навчального процесу, який реалізується на кафедрі кібербезпеки та програмного забезпечення, являється підготовка висококваліфікованих фахівців з системного програмування. Спеціаліст з системного програмування повинен у повній мірі володіти будовою та архітектурою сучасних ЕОМ.

Одним з елементів архітектури ЕОМ є послідовний порт, який використовується у сучасній обчислювальній техніці. Тому вважається доцільним реалізація програмного забезпечення яке візуалізує механізм праці послідовного порту, та передачі через нього інформації, що б допомогло при підготовці системного програміста.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи візуалізації програмування послідовного порту у навчальних цілях.

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

– Огляд існуючих систем візуалізації програмування послідовного порту у навчальних цілях.

– Дослідження системи візуалізації програмування послідовного порту у навчальних цілях.

– Програмна реалізація системи візуалізації програмування послідовного порту у навчальних цілях.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі візуалізації програмування послідовного порту у навчальних цілях.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи візуалізації програмування послідовного порту у навчальних цілях, 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 реалізації програмного забезпечення візуалізації програмування послідовного порту у навчальних цілях. Послідовний порт, серійний порт або COM-порт – двонаправлений послідовний інтерфейс, призначений для обміну байтовою інформацією.

Послідовним даний порт називається тому, що інформація через нього передається по одному біту, біт за бітом (на відміну від паралельного порту). Хоча деякі інші інтерфейси комп'ютера – такі як Ethernet, FireWire і USB – також використовують послідовний спосіб обміну, назва «послідовний порт» закріпилося за портом, що має стандарт RS-232C, і призначеним для обміну інформацією з модемом.

Призначення

Найбільше часто для послідовного порту персональних комп'ютерів використовується стандарт RS-232C. Раніше послідовний порт використовувався для підключення термінала, пізніше для модему або миші. Зараз він використовується для з'єднання із джерелами безперебійного живлення, для зв'язку з апаратними засобами розробки обчислювальних систем, що вбудовуються.

За допомогою COM-порту можна з'єднати два комп'ютери, використовуючи так званий «нуль-модемний кабель».

Роз’єм

Прийнята умовна позначка послідовного порту:  . Найбільше часто використовуються D-образні роз’єми: 9-ти й 25-тиконтактні, (DB-9 і DB-25 відповідно). Раніше використовувалися також DB-31 і круглі восьмиконтактні DIN-8. Максимальна швидкість передачі звичайно становить 115 200 бод.

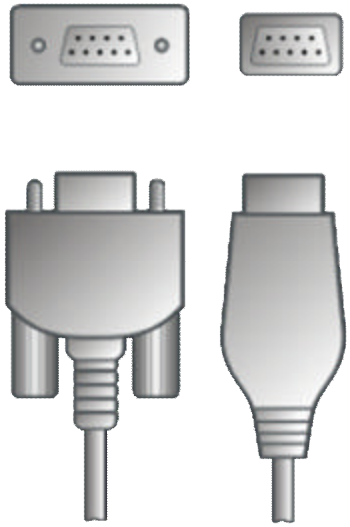

Рисунок 1.1 – Варіанти роз’єми COM-порту типу DB-9F

Актуальність

Є присутнім на сучасних комп'ютерах і використовується в промисловому й вузькоспеціальному встаткуванні. Є присутнім на сучасних ноутбуках. У цей час активно витісняється інтерфейсом USB.

Програмний доступ до COM-порту

POSIX

COM-порти в операційній системі Unix (Linux) – це файли символьних пристроїв. Звичайно ці файли розташовуються в каталозі /dev і називаються

– tty0, tty1, tty2 і т.д. в Linux.

– ttyd0, ttyd1, ttyd2 і т.д. в FreeBSD.

– ttya, ttyb, ttyc і т.д. в Solaris.

– ttyf1, ttyf2, ttyf3 і т.д. в IRIX.

– tty1p0, tty2p0, tty3p0 і т.д. в HP-UX.

– tty01, tty02, tty03 і т.д. в Digital Unix.

– ser1, ser2, ser3 і т.д. в QNX.

Для програмного доступу до COM-порту необхідно відкрити на читання/запис відповідний файл і зробити виклики спеціальних функцій tcgetattr (для того, щоб довідатися поточні настроювання) і tcsetattr (щоб установити нові

настроювання). Також може знадобитися зробити виклики ioctl з певними параметрами. Після цього при записі у файл дані будуть відправлятися через порт, а при читанні програма буде одержувати вже прийняті дані з буфера COM-порту.

Windows

COM-порти в операційній системі Windows – це іменовані канали для передачі даних, називані звичайно COM1, COM2 і т.д. один по одному виявлення драйверів відповідних пристроїв. Наприклад, для обміну інформації через Bluetooth багато драйверів представляються операційній системі як COM-порт, і резервують схоже ім'я. Слід також зазначити, що організація взаємодії по послідовному порту з погляду програмування реалізується значно легше, ніж інші способи.

1.2 Область застосування

Областю застосування розроблювального, у результаті виконання бакалаврського проектування, системного програмного забезпечення є система навчання та підготовка фахівців у галузі інформаційних технологій та програмування, з використання комп’ютерної техніки.

Особлива увага науковців і практиків до нових засобів навчання обумовлено їх значно більшою ефективністю в порівнянні з іншими засобами навчання. У випадку застосування програмних засобів обчислювальної техніки:

Підвищується якість навчання за рахунок:

а) індивідуалізації навчання, – індивідуальний темп і метод навчання, адаптація системи до вихідного рівня знань студента, що підготовляється як системний програміст, характеру й причин помилок, особливостей мислення студента, що підготовляється як системний програміст;

б) аналізу передісторії навчання і її обліку при організації наступного навчання, обліку психофізіологічних характеристик тих, кого навчають, шляхом тестування;

в) постійного індивідуального контролю якості знань на кожному етапі навчання, при цьому збільшується об'єктивність контролю знань.

Скорочується час навчання за рахунок:

а) зменшення часу на технічні операції, – виконання обчислень, контроль правильності відповідей, звернення за довідкою, допомогою або роз'ясненням;

б) миттєвої реакції програмної системи на допущені помилки;

в) індивідуалізації темпу навчання, з урахуванням рівня знань студента, що підготовляється як системний програміст;

г) адаптації до типу мислення студента, що підготовляється як системний програміст.

З погляду студента, що підготовляється як системний програміст, застосування навчальних програмних засобів обчислювальної техніки підвищує інтерес до навчання, збільшує мотивацію за рахунок новизни й сполучення більш різноманітних і наочних методів навчання в сукупності із традиційними.

Навчальні програмні засоби обчислювальної техніки в цілому ряді випадків дозволяють працювати в індивідуальному темпі, знімаючи при цьому психологічні бар'єри спілкування. При цьому враховується початковий рівень підготовки, склад розуму студента, стиль мислення, особливості виховання, пам'яті, уваги, темперамент, властивості нервової системи й. т. ін.

Студент має можливість під час навчання звернутися за довідкою, допомогою або роз'ясненням. Студент одержує можливість перервати навчання й відновити його з місця переривання зі збереженням передісторії навчання. Це дозволяє більш ефективно використовувати час, організувати додаткові заняття з відстаючими й одержати більший обсяг знань устигаючими.

З погляду викладання навчальні програмні засоби обчислювальної техніки надають педагогові наступні можливості:

а) реалізувати й поширити у вигляді навчальних програмних засобів свій досвід викладання, свою модель навчання;

б) забезпечити оперативність внесення корективів у навчальні програмні засоби обчислювальної техніки, обробку статистичних даних і прийняття тих або інших навчальних і дидактичних рішень, при цьому з'являється можливість більше гнучкого керування пізнавальною діяльністю навчання;

в) вивчити досвід ведучих у своїй області викладачів;

г) підвищити ефективність праці викладачів за рахунок виконання рутинної роботи за допомогою електронно-обчислювальних машин у короткі проміжки часу й автоматичній оцінці й реєстрації всіх параметрів процесу навчання всіх учнів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи візуалізації програмування послідовного порту у навчальних цілях, 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

Проведені дослідження дозволили виявити наступні програми, які візуалізують роботу складових ЕОМ:

– емулятор ПЗП;

– емулятор EMU;

– емулятор систем QEMU.

Розглянемо більш детально ці емулятори.

Емулятор ПЗП

Емулятор ПЗП являє собою пристрій, що з однієї сторони з'єднується з комп'ютером IBM PC, а з іншої сполучений з контролером через розетку РПЗП. Основою емулятора є мікросхема ОЗП заданого обсягу пам'яті. Достоїнством цього типу мікросхем є незрівнянно більша кількість циклів запису/зчитування ніж у мікросхем РПЗП, а також малий час циклу перезапису.

Рисунок 2.1 – Емулятор ПЗП

Принцип функціонування емулятора ПЗП, у загальному випадку, що випливає: програмний код, отриманий після вищеописаного процесу за допомогою спеціально розробленого драйвера заноситься в емулятор ПЗП. Після чого емулятор активізується для роботи з контролером і контролеру стає доступним програмний код, збережений в ОЗП емулятора, так само, як якби він був запрограмований у звичайне ПЗП. При необхідності зміни програмного коду потрібно лише обновити вміст ОЗП, що займає частки секунд, а не десятки мінут як у випадку із РПЗП.

При його виконанні було розглянуто кілька варіантів реалізації емулятора ПЗП: з використанням COM або LPT портів, або шини комп'ютера. Через простоту реалізації був обраний метод обміну через LPT порт.

Основним елементом емулятора є мікросхема ОЗП обсягом пам'яті в 32Кб. Також присутні:

– блок керування записом даних;

– регістр управління емулятора, за допомогою якого всі інші елементи переводяться в заданий стан;

– регістри формування адреси осередку ОЗП;

– регістр формування даних для запису в ОЗП;

– мультиплексор шини даних, що визначає, чи можуть дані бути зчитанні чи контролером ні;

– мультиплексор адрес, що визначає, чи будуть адреси формуватися комп'ютером або контролером;

– мультиплексор байта даних, за допомогою якого дані надходять у комп'ютер;

– мультиплексор керування, що визначає, чи буде комп'ютер управляти ОЗП або контролер.

Загальний алгоритм функціонування:

– Спочатку вміст файлу із програмним кодом заноситься в спеціально відведену область пам'яті ЕОМ.

– Далі в ОЗП емулятора з 0-го по 32767-ту адресу записуються відповідні значення з пам'яті ЕОМ.

– Після цього, з ОЗП емулятора з 0-го по 32767-ту адресу зчитуються значення й рівняються з відповідними з пам'яті ЕОМ.

– Якщо всі значення збіглися, то програма виводить повідомлення «ВСЕ НОРМАЛЬНО» і переводить емулятор у режим роботи під управлінням контролера.

– Інакше, якщо не збіглося хоч одне значення, програма намагається ще 2 рази повторити цикл запису-читання, щораз перевіряючи збіг значень; якщо після 3-го циклу знову відбулася помилка при порівнянні, програма виводить повідомлення «ПОМИЛКА ЗАПИСУ» і завершує роботу.

Емулятор EMU v1.xx

Призначення

Емулятор EMU версії 1.01 – потужний інструментальний засіб, що дозволяє детально досліджувати різноманітне програмне забезпечення, що функціонує на апаратній платформі Intel x86. З погляду користувача емулятор виглядає як класичний відладник програм, що включає традиційний набір можливостей для програм цього класу:

– покроковий і безперервний режим виконання досліджуваної програми;

– можливість зупинки виконання програми в заданих контрольних точках;

– перегляд значень регістрів центрального процесора;

– перегляд умісту оперативної пам'яті й стека.

Використання режиму емуляції дозволило реалізувати ряд принципово нових можливостей, які відсутні у звичайних відладниках:

– можливість дослідження програм, що виконуються до повного завантаження операційної системи (елементи BIOS, компоненти ОС);

– перегляд параметрів поточного стану апаратних пристроїв ПЕОМ (наприклад, умісту пам'яті CMOS, значень регістрів і внутрішніх буферів контролерів);

– завантаження в режимі емуляції різних операційних систем, під керуванням яких виконуються досліджувані програми.

Опис програми

Емулятор EMU по своїй суті є віртуальною машиною, аналогічній реальній апаратній платформі Intel x86. Досліджувана програма функціонує зовсім так само, як і на звичайному комп'ютері й нічого "не підозрює" про те, що вона працює в режимі емуляції. Досягається це за рахунок досить повного моделювання процесора, тобто реалізації набору його команд, а так само моделювання того, над чим ці команди оперують: регістрів процесора, адресного простору процесора (у яке попадають ОЗП комп'ютера, ПЗП системної плати й зовнішніх пристроїв, ОЗП зовнішніх пристроїв, наприклад, відеопам'ять, і т.д), а так само простору портів уведення/виводу процесора, що забезпечує процесору доступ до регістрів зовнішніх пристроїв.

Природно, що крім усього перерахованого вище для нормального функціонування програмам потрібні зовнішні пристрої, такі як контролер переривань, таймер, тверді й гнучкі диски, клавіатура, відеоплата й т.д. Мінімально необхідні для роботи будь-яких програм, а так само найпоширеніші пристрої представлені в емуляторі EMU у вигляді відповідних програмних моделей.

Емулятор містить у собі програмні моделі наступних основних частин ПЕОМ:

– центральний процесор (Intel Pentium);

– чипсет Intel i815EP у складі:

– міст Host-Hub Interface, що забезпечує доступ до оперативної пам'яті комп'ютера (розмір моделюємої оперативної пам'яті обмежується можливостями апаратної платформи, що здійснює емуляцію);

– міст PCI-to-AGP Bridge (повною мірою не реалізований);

– Hub Interface to PCI Bridge;

– міст PCI-to-LPC, що забезпечує доступ до пристроїв на LPC шині (аналог Super I/O або ISA шини);

– контролер прямого доступу до пам'яті 8237 (DMA controller);

– таймер 8254 (PIT);

– контролер переривань 8259 (PIC);

– годинники реального часу (RTC) + енергонезалежна пам'ять КМОП (CMOS, до 128 байт, образ CMOS зчитується із зовнішнього файлу при початку емуляції);

– контролер FWH, що забезпечує доступ до ПЗП системної плати, що містить BIOS (образ BIOS зчитується з файлу);

– контролер розширеного керування живленням (ACPI);

– контролер IDE (один "Primary" канал, підтримується тільки режим "Programmed I/O") з можливістю підключення до нього моделей жорстких дисків (ATA-4, образи дисків зберігаються у вигляді зовнішніх файлів) або моделі привода CD-ROM (ATAPI-4, для моделювання використовується реальний привод емулюючого комп'ютера);

– контролер шини LPC Super I/O типу Winbond W83627HF, що містить наступні контролери:

– контролер клавіатури PS/2 8042 (з підтримкою миші, для моделювання використовується клавіатура й миша емулюючого комп'ютера);

– контролер накопичувача на гнучких магнітних дисках 82077/ 765, (FDD, образи дискет зберігаються в зовнішніх файлах);

– контролер послідовного порту (не реалізований повною мірою);

– контролер паралельного порту (не реалізований повною мірою);

– монітор стану процесора й системної плати (Hardware Monitor);

– відеоадаптер VGA (із власної BIOS у вигляді файлу-образа);

– монітор (представлений у вигляді окремого вікна програми).

На моделюємому комп'ютері виконуються всі фази роботи звичайного комп'ютера – "включення живлення", завантаження операційної системи, запуск

програм і ін. Саме за рахунок даної "віртуалізації" і забезпечується замкнутість програмного середовища досліджуваних програм , що дозволяє істотно розширити клас розв'язуваних завдань. Під замкнутістю програмного середовища досліджуваних програм розуміються наступне: які б "шкідливі" дії не спробувала виконати досліджувана програма, результат цих дій позначиться тільки на функціонуванні інших програм, що працюють у режимі емуляції, але не на роботі емулюючої платформи. У такий спосіб експерт, що проводить дослідження, завжди перебуває "зовні" даного середовища й при будь-яких обставинах має можливість зберігати контроль над досліджуваною програмою.

Емулятор EMU функціонує під керуванням операційних систем Windows 9x/NT/2000. Це звичайна прикладна програма, що не накладає яких-небудь істотних (спеціальних) обмежень на операційну систему й апаратні можливості емулюючого комп'ютера. У програми є розвитий графічний інтерфейс для найбільш зручного дослідження емулюємих програм.

Як і традиційні відладники, емулятор дозволяє виконувати досліджувану програму в безперервному й у покроковому режимах. Допускається використання контрольних точок для припинення роботи програми. Кількість контрольних точок дуже велика (65536) і різноманітна (наприклад, зупинка програми при читанні даних з регістра зовнішнього пристрою, при переході центрального процесора в захищений режим, при виникненні в процесорі виключення й ін.), що дозволяє здійснювати набагато більше повний контроль за ходом виконання досліджуваних програм. Можливості налагодження, надавані самим центральним процесором, так само моделюються, що дозволяє використовувати в режимі емуляції й традиційні відладники. Уміст регістрів центрального процесора, осередків оперативної пам'яті, регістрів зовнішніх пристроїв виводиться в окремих вікнах. У процесі роботи програми натискання клавіш на клавіатурі комп'ютера сприймається як натискання клавіш емулюємого комп'ютера. Для відображення вмісту відеопам'яті адаптера VGA так само використовується окреме вікно емулятора.

Типовий вид емулятора під час роботи представлений на рисунку 2.2.

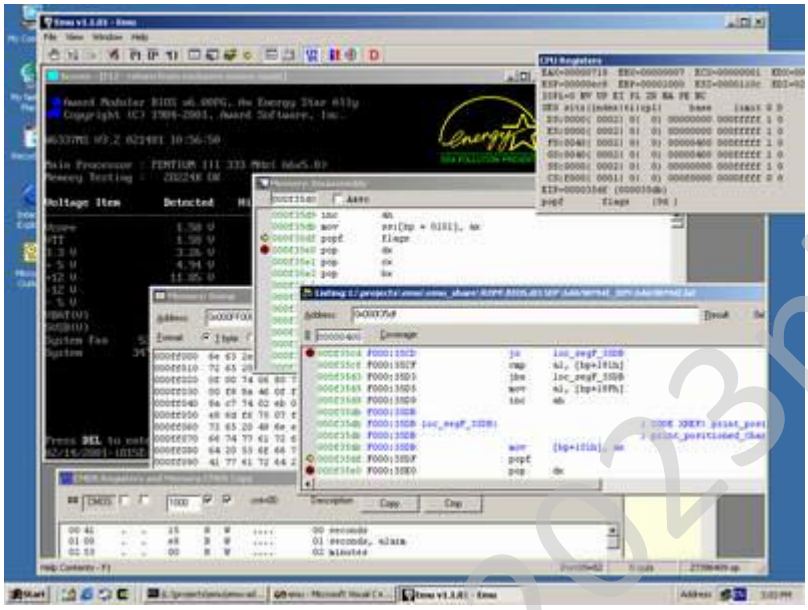

Рисунок 2.2 – Типовий вид емулятора

Емулятор має убудований дизассемблер, що дозволяє динамічно дизасемблювати будь-які ділянки оперативної пам'яті. Користувач також може застосовувати для трасування лістинги, отримані за допомогою дизассемблерів інших фірм.

Область застосування, що рекомендується

Емулятор є програмою із широкою областю застосування. З його допомогою можна вирішувати наступні завдання:

– Дослідження особливостей функціонування елементів різних операційних систем, що працюють на платформі Intel x86 (наприклад, Windows).

– Дослідження роботи прикладних програм в умовах відсутності вихідних текстів з метою виявлення алгоритму функціонування.

– Налагодження програм, що виконуються до повного завантаження операційної системи й, отже, без використання стандартних відладників (компоненти ОС).

– Інтерактивне налагодження програмного забезпечення BIOS і користувальницьких ПЗП в умовах інтенсивної взаємодії з апаратурою комп'ютера.

– Дослідження програм, які містять зашифровані (упаковані) ділянки або мають засоби захисту від налагодження.

– Дослідження систем реального часу в покроковому (!) режимі.

– Антивірусні дослідження.

– Налагодження програмного забезпечення, взаємодіючого з апаратними пристроями, які реально відсутні на інструментальному комп'ютері, але представлені в емуляторі своєю програмною моделлю.

– Виконання робіт із сертифікації й атестації програмно-апаратного забезпечення.

Обмеження

Емулюємий комп'ютер у порівнянні з реальними комп'ютерами має наступні обмеження:

– кількість пристроїв IDE – не більше 2 (у реальних – до 4);

– розмір жорсткого диска – не більше 8 гігабайт, режим доступу – тільки Programmed I/O;

– образ BIOS, що завантажується в емулятор, повинен бути сполучимо з BIOS для материнської плати із чипсетом Intel i815EP і чипом Super/O Winbond W83627HF.

При роботі програмного забезпечення в режимі емуляції спостерігається природне зниження швидкодії в порівнянні зі швидкодією інструментального комп'ютера. Так, наприклад, на комп'ютері із процесором Pentium II ~400 Мгц швидкість емуляції становить близько 1.000.000 команд у секунду, що цілком прийнятно для нормальної роботи

Емуляція систем за допомогою QEMU

QEMU – це додаток з відкритим вихідним кодом, повністю емулююче персональний комп'ютер. Крім емуляції процесора, QEMU дозволяє емулювати також всі необхідні підсистеми, такі як мережні адаптери й відеоплати. Він також може емулювати більше просунуті речі, такі як симетричні багатопроцесорні системи (аж до 255 процесорів) і інші процесорні архітектури, такі як ARM або PowerPC. У цій статті розглядається QEMU і його архітектура, і показується, як емулювати гостьову операційну систему на Linux-хості.

Не буде перебільшенням сказати, що віртуализація – це дуже затребувана в цей час технологія. Сьогодні по запиту слова віртуализація в пошуковій системі видається приблизно 22 мільйона результатів. Наприклад, тільки за один місяць корпорація EMC оголосила про вивід на ринок цінних паперів підрозділу VMware, Citrix System оголосила про плани покупки XenSource, з'явилася безліч нових компаній у сегменті віртуалізації. На цьому, як виявилося, колосальному ринку постійно виявляються нові ніші. Але за всіма розмовами про первісні публічні пропозиції й поглинання в області віртуалізації в ці дні легко забути про деяких інших уже існуючих технологіях віртуалізації.

QEMU – це додаток, яке можна застосовувати для багатьох цілей. Його можна використовувати для віртуалізації гостьової операційної системи або в якості повнофункціонального машинного емулятора, що запускає операційні системи, призначені для процесора хост-системи або інших процесорних архітектур.

Почнемо з короткого введення в технологію віртуалізації, щоб закласти основу для розгляду QEMU.

Під віртуализацією у цій статті мається на увазі віртуализація платформи. Для фізичного встаткування, що контролює програма може бути операційною системою-хазяїном або гіпервізором. У деяких випадках сама хазяйська операційна система і є гіпервізором. Гостьові операційні системи "проживають" на гіпервізорі. У деяких випадках гостьові операційні системи орієнтовані на той

же процесор, що й контролююча програма, але в інших випадках платформи можуть бути відмінними (наприклад, гостьова система PowerPC працює на x86 платформі).

Реалізувати віртуалізацію можна безліччю шляхів, але найбільше часто зустрічаються три основних методи. Перший називається "рідною" (або повною) віртуализацією. У цьому варіанті гіпервізор реалізує основні елементи ізоляції, відокремлюючи фізичне встаткування від гостьової операційної системи. Цей підхід уперше був продемонстрований в 1966 році в операційній системі віртуальних машин/віртуальної пам'яті IBM® CP – 40, а зараз той же метод використовується в VMware ESX Server.

Інший популярний метод віртуалізації називається паравіртуализацією. У випадку паравіртуалізації контролююча програма реалізує інтерфейсу прикладних програм (API) гіпервізора, що використовується гостьовою операційною системою. Паравіртуалізацію використовують Xen і Linux Kernel-based Virtual Machine (KVM).

Третій корисний метод називається емуляцією. Емуляція, як видно з назви, віртуалізує гостьову платформу завдяки повній імітації апаратного середовища. Емуляція здійснюється в різних формах, навіть у межах того самого додатка. Прикладами віртуалізації за допомогою емуляції -є QEMU і Bochs.

Архітектура QEMU

Розглянемо, як QEMU забезпечує емуляцію. У цьому розділі описуються два режими роботи QEMU, а також деякі цікаві аспекти динамічного транслятора QEMU.

Основні операції QEMU

QEMU підтримує два режими емуляції: користувальницький режим [User-mode] і системний режим [System-mode]. Користувальницький режим емуляції дозволяє процесу, створеному на одному процесорі, працювати на іншому (виконується динамічний переклад інструкцій для приймаючого процесора й

конвертація системних викликів Linux). Системний режим емуляції дозволяє емулювати систему цілком, включаючи процесор і різноманітну периферію.

При емуляції коду для x86 на хост-системі з архітектурою x86 можна досягти ефективності, близької до рідного, за допомогою так званого акселератора QEMU. Він дозволяє виконувати емулюємий код безпосередньо на центральному процесорі хоста (на Linux через модуль ядра).

Але що робить QEMU дійсно цікавим з технічної точки зору, так це його швидкий і компактний динамічний транслятор [dynamic translator]. Динамічний транслятор дозволяє виконувати під час виконання переводити інструкції цільового (гостьового) процесора в інструкції центрального процесора хоста для забезпечення емуляції. Це може бути зроблено методом грубої сили (просто заміняючи інструкції одного процесора іншими), але це не завжди легко зробити, а в деяких випадках одна інструкція може зажадати декількох інструкцій або навіть змін у їхньому порядку проходження для трансльованої архітектури.

QEMU забезпечує динамічну трансляцію перетворенням цільової інструкції в мікрооперації. Ці мікрооперації являють собою елементи З-коду, які компілюються в об'єкти. Потім вступає в справу основний транслятор. Він відображає цільові інструкції на мікрооперації для динамічної трансляції. Такий підхід не тільки ефективний, але й забезпечує переносимість.

Динамічний транслятор QEMU також кеширует блоки трансльованого коду для зниження накладних витрат транслятора. Коли блок цільового коду зустрічається вперше, він переводиться й зберігається у вигляді трансльованого блоку [translated block]. Кеш QEMU зберігає недавно переведені блоки в буфері обсягом 16 МБ. QEMU може навіть підтримувати код, що самомодифікується, анулюючи трансльовані блоки в кеші.

Підтримувані периферійні пристрої

Використання QEMU як емулятор персонального комп'ютера забезпечує підтримку різноманітних периферійних пристроїв. Природно, сюди входять стандартні периферійні пристрої – емулятор апаратного відеоадаптера (VGA),

миші й клавіатури PS/2, інтерфейс IDE для жорстких дисків, інтерфейс CD-ROM і емуляція дисководу. Крім того, QEMU має можливість емуляції мережних адаптерів NE2000 (PCI), послідовних портів, численних звукових плат і контролера PCI Universal Host Controller Interface (UHCI) Universal Serial Bus (USB) (з віртуальним USB концентратором). Також підтримується до 255 процесорів з підтримкою симетричної багатопроцесорності (SMP).

Крім стандартних PC і ISA PC (без шини PCI), QEMU можуть емулювати і інші апаратні платформи, не пов'язані із ПК, такі як базові плати APM Versatile (з використанням 926E) і плати на основі Malta million instructions per second (MIPS). У цей час ведеться робота з реалізації підтримки ряду інших платформ, включаючи Power Macintosh G3 (Blue & White) і Sun 4u.

Збірка й установка QEMU

Збірка й установка QEMU виконується дуже просто за допомогою стандартних інструментів GNU. Після закачування й розархівування дистрибутива QEMU потрібно виконати configure, make, а потім make install (див. лістинг 1).

Лістинг 1. Збірка емулятора QEMU

```
$ wget http://fabrice.bellard.free.fr/qemu/ qemu-0.9.0.tar.gz
$ tar xfvz qemu-0.9.0.tar.gz
$ cd qemu-0.9.0
$ ./configure
$ make
$ make install
$
```

Цей процес створює не образ, що завантажується тільки, QEMU для поточної цільової архітектури, але й образи для інших архітектур, включаючи ARM, MIPS, PowerPC, 68k і SPARC. Після цього можна завантажити ядро Linux уже зібраним для різних цільових архітектур.

Якщо гостьова й операційна система хоста націлені на одну архітектуру, то можна збільшити швидкість до близької до рідного, використовуючи акселератор QEMU (KQEMU). KQEMU – це драйвер (модуль ядра для Linux), що дозволяє користувальницькому коду й коду ядра запускатися прямо на

центральному процесорі хоста. Зібрати акселератор QEMU також просто, як і сам QEMU (див. лістинг 2).

Лістинг 2. Збірка акселератора QEMU

```
$ wget http://fabrice.bellard.free.fr/qemu/ kqemu-1.3.0pre11.tar.gz
$ tar xvfz kqemu-1.3.0pre11.tar.gz
$ cd kqemu-1.3.0pre11
$ ./configure
$ make
$ make install
```

KQEMU можна скомпілювати й установити на безлічі операційних систем, включаючи Microsoft® Windows®, FreeBSD® і Linux. Після Збірка акселератора QEMU його потрібно встановити в Linux, використовуючи наступну команду:

```
$ insmod kqemu.ko
$
```

Використання QEMU

Тепер розглянемо використання QEMU для віртуалізації іншої машини в типовому середовищі настільного ПК під GNU/Linux. Емуляція іншої машини схожа на початок роботи з тільки що купленим новим комп'ютером. Перший крок – це установка операційної системи. Новий комп'ютер, звичайно, повинен мати місце для установки операційної системи, тому необхідно жорсткий диск.

QEMU надає спеціальну команду для створення жорсткого диска, що називається qemu-img. Ця утиліта створює образи різних форматів, але кращий (для qemu) з них називається qcow (або qemu copy-on-write). Перевагою даного формату є те, що образ емулюємого диска не обов'язково повинен займати фізичний файл такого ж обсягу. Інакше кажучи, формат допускає пропуски, що дозволяє зробити образ диска більше компактним. Наприклад, порожній образ диска обсягом 4 ГБ займе всього 16 КБ.

Для quemu-img необхідно вказати операцію (create для створення нового образа диска), формат (qcow для форматування образа qemu), розмір і ім'я образа диска. Наступний приклад емулює машину для невеликого дистрибутива Linux, передбачуваного для використання на Flash. Отже, створюємо образ диска на 128 МБ:

```
$ qemu-img create -f qcow disk.img 128M
Formating 'disk.img', fmt=qcow, size=131072 k
$
```

Необхідно пам'ятати, що якщо планується установка операційної системи загального призначення, такий як Windows, Linux або FreeBSD, те потрібний набагато більший диск. Результат виконаної операції – файл disk.img – буде містити емулюємий диск розміром 128 МБ.

Тепер, коли жорсткий диск створений, можна встановити на нього нову операційну систему. Для демонстрації цього процесу я використовую невеликий дистрибутив Linux, називаний cfLinux, призначений для застосування в якості невеликий що вбудовується Linux-Системи в таких пристроях, як шлюзи, бездротові точки доступу, брандмауери й маршрутизатори.

Тепер у нас є емулюємий диск (disk.img) і CD-ROM, з якого можна інсталювати операційну систему. Наступним кроком буде інсталяція системи на жорсткий диск. Це робиться дуже просто за допомогою qemu:

```
$ qemu -hda disk.img -cdrom /root/ cflinux-1.0.iso -boot d
$
```

При використанні qemu образ жорсткого диска задається за допомогою опції hda, а компакт-диск (файл, де розташовується образ) – за допомогою опції cdrom. Опція boot дозволяє завантажитися з CD-ROM. Аргумент d указує завантажуватися з CD-ROM, а – із флоппі-диска, с указує на завантаження з жорсткого диска (за замовчуванням), а n – завантаження з мережі. Якщо команда уведена правильно, то з'явиться нове вікно QEMU з емулюємою машиною.

Дотримуючись інструкцій по установці з CD-ROM, легко закінчити установку з ISO-Образа на емулюємий жорсткий диск. Установка вимагає перезавантаження. У цьому місці можна закінчити емуляцію (Ctrl-C у вікні qemu). Тепер можна завантажити встановлену операційну систему за допомогою наступної команди:

```
$ qemu -hda disk.img
$
```

Ця команда просто емулює стандартний PC (опція за замовчуванням) з жорстким диском, представленим файлом образа disk.img.

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.

Дизайнер форм, Інспектор об'єктів і інші засоби залишаються доступними під час роботи програми, тому вносити зміни можна в процесі відлагодження.

Builder C++ поставляється в трьох варіантах: Standard (стандартний), Professional (для професіоналів розробників, орієнтованих на мережеву архітектуру) і Client/Server Suite (для розробки систем в архітектурі клієнт/сервер). Останні два варіанти доповнюють стандартний початковими текстами візуальних компонентів, різномасштабним словником даних, новими функціями мови запитів SQL для бази даних, пакетом підтримки систем Internet, службою моніторингу програм, а також рядом інших засобів.

Builder C++ підтримує зв'язок з різними базами даних 3-х видів: dBASE і Paradox: Sybase, Oracle, InterBase і Informix; Excel, Access, FoxPro і Btrieve. Механізм BDE (Borland Database Engine) додає обслуговуванню зв'язків з базами даних дивовижну простоту і прозорість. Провідник Database Explorer дозволяє зображати зв'язки і об'єкти баз даних графічно. Використовуючи компоненти баз даних, я побудував електронний записник згідно таблиці dBASE за півгодини роботи на комп'ютері. Спадкоємство готових форм і їх "підгонка" під специфічні вимоги помітно скорочують тимчасові витрати на вирішення подібних завдань.

Довідкова служба Builder C++ надавала мені допомогу в цій і багатьох інших подібних ситуаціях. Є повний опис кожного управляємого компонента, включаючи списки властивостей і методів, а також численні приклади. Виклад матеріалу в книзі був значно покращуваний і систематизований завдяки відомостям, почерпнутим мною з довідкової служби.

Завдяки засобам управління проектами, двосторонній інтеграції застосунку і синхронізації між засобами візуального і текстового редагування, а також вбудованому відладнику (з асемблерним вікном прокрутки, покрокового виконання, точок останову, трасуванням і тому подібне) – Builder C++ корпорації Borland надає собою вражаюче середовище розробки, яка, мабуть, витримає конкурентну боротьбу з такими модними продуктами як Developer Studio фірми Microsoft.

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 програмування послідовного порту у навчальних цілях.

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

Опис роботи COM порту

Розглянемо порт послідовної передачі даних. Його називають ще портом RS-232-C, або асинхронним адаптером RS-232-C. Комп'ютер IBM PC підтримує інтерфейс RS-232-C не повною мірою, скоріше роз’єм, позначений на корпусі комп'ютера як порт послідовної передачі даних, містить деякі із сигналів, що входять в інтерфейс RS-232-C і що мають відповідному цьому стандарту рівні напруги. Практично кожний комп'ютер обладнаний хоча б одним портом для послідовної передачі даних.

Основні поняття й терміни

Послідовна передача даних означає, що дані передаються з використанням єдиної лінії. При цьому біти байта даних передаються по черзі з використанням одного проведення. Для синхронізації групі бітів звичайно передує спеціальний стартовий біт, після групи бітів випливають біт перевірки на парність і один або два стопових біти. Іноді біт перевірки на парність може бути відсутнім.

Вихідний стан лінії послідовної передачі даних – рівень логічної 1. Стартовий біт START сигналізує про початок передачі даних. Далі передаються біти даних, спочатку молодші, потім старші. Якщо використовується біт парності P, то передається й він. Біт парності має таке значення, щоб у пакеті бітів загальна кількість одиниць (або нулів) була парна або непарна.

У самому кінці передаються один або два стопових біти STOP, що завершують передачу байта. Потім рівень лінії передачі знову встановлюється в 1 до приходу наступного стартового біта.

Використання парності, стартових і стопових бітів визначають протокол передачі даних. Очевидно, що передавач і приймач даних повинні використовувати той самий протокол, інакше зв'язок буде неможливим.

Інша важлива характеристика – швидкість передачі даних. Вона також повинна бути однаковою для передавача й приймача.

Швидкість передачі даних звичайно виміряється в бодах. Боди – це кількість переданих бітів у секунду. При цьому враховуються й старт/стопні біти, а також біт парності.

Іноді використовується інший термін – біти в секунду (bps). Тут мається на увазі ефективна швидкість передачі даних, без обліку службових бітів.

Апаратна реалізація

Комп'ютер може бути оснащений одним або двома портами послідовної передачі даних. Ці порти розташовані або на материнській платі, або на окремій платі, що вставляється в слоти розширення материнської плати.

Бувають також плати, що містять 4 або 8 портів послідовної передачі даних. Їх часто використовують для підключення декількох комп'ютерів або терміналів до одного, центрального, комп'ютера.

В основі послідовного порту передачі даних лежить мікросхема Intel 8250. Це універсальний асинхронний приймально-передавач (UART – Universal Asynchronous Receiver Transmitter). Мікросхема містить кілька внутрішніх регістрів, доступних через команди уведення/виводу.

Мікросхема 8250 містить регістри передавача й приймача даних. При передачі байта він записується в буферний регістр передавача, звідки потім листується в регістр зсуву передавача. Байт "висувається" зі регістра зсуву по бітах.

Аналогічно є зсувовий і буферний регістри приймача.

Програма має доступ тільки до буферних регістрів, копіювання інформації в регістри зсуву й процес зрушення виконується мікросхемою 8250 автоматично.

Зовнішні пристрої підключаються до порту уведення/виводу через роз'єм DB25P (що має 25 виводів) або DB9P (щомає 9 виводів). Приведемо розведення роз'єму послідовної передачі даних DB25P.

Таблиця 3.1 – Призначення контактів роз'єм DB25P

Номер контакту	Призначення контакту	Вхід або вихід
1	Захисне заземлення	-
2	Передані дані (Transmitted Data)	Вихід
3	Прийняті дані (Received Data)	Вхід
4	Запит для передачі (Request to send, RTS)	Вихід
5	Скидання для передачі (Clear to Send, CTS)	Вхід
6	Готовність даних (Data Set Ready, DSR)	Вхід
7	Сигнальне заземлення	-
8	Детектор прийнятого з лінії сигналу (Data Carrier Detect, DCD)	Вхід
9-19	Не використовуються	
20	Готовність вихідних даних (Data Terminal Ready, DTR)	Вихід
21	Не використовується	
22	Індикатор виклику (Ring Indicator, RI)	Вхід
23-25	Не використовується	

Поряд з 25-контактним роз'єм часто використовується 9-контактний роз'єм.

Таблиця 3.2 – Призначення контактів роз'єму DB9P

Номер контакту	Призначення контакту	Вхід або вихід
1	Детектор прийнятого з лінії сигналу (Data Carrier Detect, DCD)	Вхід
2	Прийняті дані (Received Data)	Вхід
3	Передані дані (Transmitted Data)	Вихід
4	Готовність вихідних даних (Data Terminal Ready, DTR)	Вихід
5	Сигнальне заземлення	-
6	Готовність даних (Data Set Ready, DSR)	Вхід
7	Запит для передачі (Request to send, RTS)	Вихід
8	Скидання для передачі (Clear to Send, CTS)	Вхід
9	Індикатор виклику (Ring Indicator, RI)	Вхід

Рівні напруги на лініях роз'ємів становлять для логічного нуля -15 вольтів, для логічної одиниці – +15 вольтів.

Доступ до окремих ліній можливий через порти введення/виводу асинхронного адаптера, які ми розглянемо в наступному розділі. Там же буде описане призначення окремих ліній роз'єму.

Порти асинхронного адаптера

На етапі ініціалізації системи модуль POST BIOS тестує наявні асинхронні адаптери й ініціалізує перші два. Їхні базові адреси розташовуються в області даних BIOS починаючи з адреси 0000:0400h.

Перший адаптер COM1 має базову адреса 3F8h і займає діапазон адрес від 3F8h до 3FFh. Другий адаптер COM2 має базова адреса 2F8h і займає адреси 2F8h...2FFh.

Асинхронні адаптери можуть виробляти переривання:

– COM1 – IRQ4 (відповідає INT 0Ch)

– COM2 – IRQ3 (відповідає INT 0Bh)

Розглянемо призначення окремих бітів цих портів.

Порт 3F8h.

Цей порт відповідає регістру переданих даних. Для передачі в порт 3F8h необхідно записати переданий байт даних. Після прийому даних від зовнішнього пристрою вони можуть бути прочитані із цього порту.

Залежно від стану старшого біта керуючого слова, виведеного в керуючий регістр із адресою 3FBh, призначення порту 3F8h може змінюватися. Якщо цей біт дорівнює 0, порт використовується для запису переданих даних. Якщо ж біт дорівнює 1, порт використовується для виводу значення молодшого байта дільника частоти тактового генератора. Змінюючи вміст дільника, можна змінювати швидкість передачі даних. Старший байт дільника записується в порт 3F9h.

Залежність швидкості передачі даних від значення дільника частоти представлено в наступній таблиці.

Таблиця 3.3 – Призначення дільника

Дільник	Швидкість передачі в бодах	Дільник	Швидкість передачі в бодах
1040	110	24	4800
768	150	12	9600
384	300	6	19200
192	600	3	38400
96	1200	2	57600
48	2400	1	115200

Порт 3F9h.

Порт використовується або як регістр керування перериваннями від асинхронного адаптера або (після виводу в порт 3F9h байти із установленим в 1 старшим бітом) для виводу значення старшого байта дільника частоти тактового генератора.

У режимі регістра керування перериваннями порт має наступний формат.

Таблиця 3.4 – Призначення біт порту

Номер біта	Значення біта
0	1 – дозвіл переривання при готовності прийнятих даних
1	1 – дозвіл переривання після передачі байта (коли вихідний буфер передачі порожній)
2	1 – дозвіл переривання по виявленню стану "BREAK" або помилково
3-6	1 – дозвіл переривання по зміні стану вхідних ліній на рознімані RS-232-C (CTS, DSR, RI, DCD)
7	Не використовуються, повинні бути рівні 0

Порт 3FAh.

Регістр ідентифікації переривання. Зчитуючи його вміст, програма може визначити причину переривання.

Формат регістра.

Таблиця 3.5 – Призначення біт порту

Номер біта	Значення біта
0	1 – Немає переривань, що очікують обслуговування
1-2	00 – Переривання по лінії стану приймача, виникає при переповненні приймача, помилках парності або формату даних або при стані "BREAK". Скидається після читання стану лінії з порту 3FDh. 01 – Дані прийняті й доступні для читання. Скидається після читання даних з порту 3F8h. 10- Буфер передавача порожній. Скидається при записі нових даних у регістр даних передавача, порт 3F8h. 11 – Стан модему. Установлюється при зміні стану вхідних ліній CTS, RI, DCD, DSR. Скидається після читання стану модему з порту 3FEh.
3-7	Повинне дорівнювати 0.

Порт 3FBh.

Керуючий регістр, доступний по запису й читанню.

Таблиця 3.6 – Призначення біт порту

Номер біта	Значення біта
0-1	Довжина слова в бітах: 00 – 5 біт; 01 – 6 біт; 10 – 7 біт; 11 – 8 біт.
2	Кількість стопових біт: 0 – 1 біт; 1 – 2 біти.
3-4	Парність: X0 – контроль на парність не використовується; 01 – контроль на непарність; 11 – контроль на парність.
5	Фіксація парності. При установці цього біта біт парності завжди приймає значення 0 (якщо біти 3-4 рівні 11) або 1(якщо біти 3-4 рівні 01).
6	Установка переривання. Викликає вивід рядка нулів як сигнал "BREAK" для підключеного пристрою.
7	1 – порти 3F8h і 3F9h використовуються для завантаження дільника частоти тактового генератора; 0 – порти використовуються як звичайно.

Порт 3FCh.

Регістр керування модемом. Управляє станом вихідних ліній DTR, RTS , ліній, специфічних для модемів OUT1 і OUT2, для запуску діагностики з'єднаних разом замкнутих вході й виході асинхронного адаптера. Формат порту.

Таблиця 3.7 – Призначення біт порту

Номер біта	Значення біта
0	лінія DTR;
1	лінія RTS;
2	лінія OUT1 (запасна);
3	лінія OUT2 (запасна);
4	запуск діагностики при вході асинхронного адаптера, замкнутому на його вихід;
5-7	Повинне дорівнювати 0.

Порт 3FDh.

Регістр стану лінії.

Таблиця 3.8 – Призначення біт порту

Номер біта	Значення біта
0	Дані отримані й готові для читання, скидається при читанні даних.
1	Помилка переповнення. Був прийнятий новий байт даних, а попередній ще не був зчитаний програмою. Попередній байт загублений.
2	Помилка парності, скидається після читання стану лінії.
3	Помилка синхронізації.
4	Виявлено запит на переривання передачі "BREAK" – довгий рядок нулів.
5	Регістр зберігання передавача порожній, у нього можна записувати новий байт для передачі.
6	Регістр зсуву передавача порожній. Цей регістр одержує дані з регістра зберігання й перетворить їх у послідовний вид для передачі.
7	Таймаут (пристрій не пов'язане з комп'ютером).

Порт 3FEh.

Регістр стану модему.

Таблиця 3.9 – Призначення біт порту

Номер біта	Значення біта
0	лінія CTS змінила стан
1	лінія DSR змінила стан
2	лінія RI змінила стан
3	лінія DCD змінила стан
4	стан лінії CTS
5	стан лінії DSR
6	стан лінії RI
7	стан лінії DCD

Підтримка асинхронного адаптера в BIOS

Ми опишемо функції BIOS, що полегшують обслуговування двох асинхронних адаптерів, COM1 і COM2. Ці функції доступні через переривання INT 14h.

Перша функція призначена для ініціалізації портів асинхронного адаптера:

На вході: AH = 00h;

DX = номер порту: 0 – COM1, 1 – COM2;

AL = параметри ініціалізації (див. далі).

На виході: AH = стан порту асинхронного адаптера;

AL = стан модему.

При виклику цієї функції регістр AL повинен містити параметри ініціалізації.

Таблиця 3.10 – Призначення біт

Номер біта	Значення біта
0-1	Довжина слова в бітах: 00 – 5 біт; 01 – 6 біт; 10 – 7 біт; 11 – 8 біт.
2	Кількість стопових біт: 0 – 1 біт; 1 – 2 біти.
3-4	Парність: X0 – контроль на парність не використовується; 01 – контроль на непарність; 11 – контроль на парність.
5-7	Швидкість передачі даних у бодах: 000 – 110 001 – 150 010 – 300 011 – 600 100 – 1200 101 – 2400 110 – 4800 111 – 9600

Після виклику функції в регістр AH записується стан порту асинхронного адаптера.

Таблиця 3.11 – Призначення біт

Номер біта	Значення біта
0	таймаут, якщо встановлено цей біт, інші біти не мають значення
1	регістр зсуву передавача порожній;
2	буферний регістр передавача порожній;
3	виявлено стан "BREAK";
4	помилка синхронізації;
5	помилка парності;
6	помилка переповнення вхідного регістра;
7	дані готові.

Регістр AL містить байт стану модему.

Таблиця 3.12 – Призначення біт порту

Номер біта	Значення біта
0	лінія CTS змінила стан;
1	лінія DSR змінила стан;
2	лінія RI змінила стан;
3	лінія DCD змінила стан;
4	стан лінії CTS;
5	стан лінії DSR;
6	стан лінії RI;
7	стан лінії DCD.

Для передачі байта використовується наступна функція:

На вході: AH = 01h;

DX = номер порту: 0 – COM1, 1 – COM2;

AL = переданий байт.

На виході: AL зберігається;

AH = стан порту асинхронного адаптера,

якщо біт 7 регістра AH установлений в 1,

відбулася помилка.

Функція 02h призначена для прийому байта:

На вході: AH = 02h;

DX = номер порту: 0 – COM1, 1 – COM2.

На виході: AL = прийнятий байт;

AH = стан порту асинхронного адаптера,

якщо регістр AH не дорівнює 0,

відбулася помилка.

Стан порту асинхронного адаптера можна довідатися за допомогою функції 03h:

На вході: AH = 03h;

DX = номер порту: 0 – COM1, 1 – COM2.

На виході: AH = стан порту асинхронного адаптера;

AL = стан модему.

Програмування асинхронного адаптера

На жаль, MS-DOS не містить скільки-небудь серйозної підтримки асинхронного адаптера. Дві функції переривання INT 21h з номерами 3 і 4 призначені для читання й запису байтів через асинхронний адаптер. Обидві ці функції мають справа з адаптером COM1 або AUX. Функція 3 одержує в регістрі AL символ, прийнятий з адаптера, функція 4 посилає в адаптер символ, записаний у регістр DL.

Основний недолік функцій MS-DOS, призначених для роботи з адаптером, полягає у відсутності їхньої функціональної повноти. Використовуючи тільки функції MS-DOS, ви не зможете проаналізувати помилкові ситуації й змінити режим роботи асинхронного адаптера – немає відповідних засобів.

Функції BIOS, що обслуговують адаптер, більше різноманітні. Однак і їм властиві недоліки. Наприклад, ви не зможете встановити швидкість передачі більше 9600 бод або використовувати режим фіксації парності. Ні можливості довідатися поточний режим асинхронного адаптера, відсутня підтримка модему.

Тому для програмування асинхронного адаптера ми рекомендуємо використовувати порти уведення/виводу мікросхеми 8250.

Ініціалізація асинхронного адаптера

Перше, що повинна зробити програма, що працює з асинхронним адаптером – установити протокол обміну й швидкість передачі даних. Після завантаження операційної системи для асинхронних адаптерів установлюється швидкість 2400 бод, не виконується перевірка на парність, використовуються один стоповий біт і восьмибітова довжина переданого символу. Ви можете змінити цей режим командою MS-DOS MODE.

Виконавши уведення з порту 3FBh, програма може одержати поточний режим адаптера. Для установки нового режиму змініть потрібні вам поля й запишіть новий байт режиму за адресою 3FBh.

Якщо вам треба задати нове значення швидкості обміну даними, перед записом байта режиму встановите старший біт цього байта в 1. Потім послідовно двома командами виводу завантажите дільник частоти тактового генератора. Молодший байт запишіть у порт 3F8h, старший – у порт 3F9h.

Перед початком роботи необхідно також проініціалізувати регістр керування перериваннями (порт 3F9h), навіть якщо у вашій програмі не використовуються переривання від асинхронного адаптера. Якщо переривання вам не потрібні, запишіть у цей порт значення 0.

На цьому ініціалізацію можна вважати закінченою.

Передача даних

Перед записом байта даних у регістр передавача необхідно переконатися в тому, що регістр зберігання передавача вільний, тобто переконатися в тому, що передача попереднього символу завершена.

Ознакою того, що регістр передавача вільний, є встановлений в 1 біт 5 регістра стану лінії з адресою 3FDh.

Прийом даних

Аналогічно передачі даних, перед уведенням символу з порту приймача 3F8h необхідно переконатися в тому, що біт 0 порту 3FDh установлений в 1. Це означає, що символ прийнятий з лінії й перебуває в буферному регістрі приймача.

Використання переривань

Тому що процес послідовної передачі даних протікає досить повільно, має сенс виконувати його у фоновому режимі, використовуючи переривання по закінченню передачі або прийому символу. Нагадаємо, що порту COM1 відповідає апаратне переривання INT 0Ch, а COM2 – INT 0Bh.

Для дозволу переривань необхідно встановити в 1 біти порту керування перериваннями 3F9h, що відповідають тим перериванням, які ми бажаємо обробляти.

Коли відбулося переривання, програма-оброблювач переривання повинна проаналізувати причину переривання, прочитавши вміст порту ідентифікації переривання з адресою 3FAh.

Не забудьте, що наприкінці оброблювача апаратного переривання повинна перебуває послідовність команд:

```
mov     al, 20h
out 20h, al
iret
```

Може трапитися так, що одночасно відбудеться кілька переривань. У цьому випадку біт 0 регістра ідентифікації переривання буде встановлений в 1. Якщо така ситуація має місце, перед завершенням обробки переривання вам треба знову прочитати регістр ідентифікації переривання й обробити наступне

переривання. Так варто надходити доти, поки біт 0 регістра ідентифікації переривання не стане рівним нулю.

Програмування COM-порту в ОС Windows

Написати програму, керуючим пристроєм через COM-порт, для MS-DOS не так складно. Із платформою Win32 справа дещо складніша. Але тільки на перший погляд. Звичайно прямо працювати з регістрами портів не можна, Windows це не дозволяє, зате можна не обертати уваги на тонкості різних реалізацій (i8055, 16450, 16550A) і не займатися обробкою переривань.

Відкриття порту

З послідовними й паралельними портами в Win32 працюють як з файлами. Для відкриття порту використовується функція `CreateFile`. Ця функція надається Win32 API. Її прототип виглядає так:

```
HANDLE CreateFile(
  LPCTSTR         lpFileName,
  DWORD           dwDesiredAccess,
  DWORD           dwShareMode,
  LPSECURITY_ATTRIBUTES lpSecurityAttributes,
  DWORD           dwCreationDistribution,
  DWORD           dwFlagsAndAttributes,
  HANDLE          hTemplateFile
);
```

lpFileName

Покажчик на рядок з ім'ям файлу, що відкривається або створюваного. Формат цього рядка може бути дуже «хитрим». Зокрема можна вказувати мережні імена для доступу до файлів на інших комп'ютерах. Можна відкривати логічні розділи або фізичні диски й працювати в обхід файлової системи.

Послідовні порти мають імена "COM1", "COM2", "COM3", "COM4", "COM5", "COM6", "COM7", "COM8", "COM9". Для доступу до портів, чий номер більше 9, необхідно указувати ім'я порту як "\\.\COMx", де x – номер порту. Наприклад, "\\.\COM72". Такий синтаксис підходить для будь-якого номера порту. Точно так само вони називалися в MS-DOS. Паралельні порти називаються «LPT1», «LPT2» і так далі.

dwDesiredAccess

Задає тип доступу до файлу. Можливе використання наступних значень:

– `0`. Опитування атрибутів пристрою без одержання доступу до нього.

– `GENERIC_READ`. Файл буде зчитуватися.

– `GENERIC_WRITE`. Файл буде записуватися.

– `GENERIC_READ|GENERIC_WRITE`. Файл буде й зчитуватися й записуватися.

dwShareMode

Задає параметри спільного доступу до файлу. Комунікаційні порти не можна робити поділюваними, тому даний параметр повинен бути дорівнює 0.

lpSecurityAttributes

Задає атрибути захисту файлу. Підтримується тільки в Windows NT. Однак при роботі з портами повинен у кожному разі рівнятися `NULL`.

dwCreationDistribution

Управляє режимами автостворення, автоусічення файлу і їм подібними. Для комунікаційних портів завжди повинне задаватися `OPEN_EXISTING`.

dwFlagsAndAttributes

Задає атрибути створюваного файлу. Також управляє різними режимами обробки. При роботі з портом цей параметр повинен бути або рівним `0`.

hTemplateFile

Задає описувач файлу-шаблона. При роботі з портами завжди повинен бути дорівнює `NULL`.

При успішному відкритті файлу, у цьому випадку порту, функція повертає дескриптор (`HANDLE`) файлу. При помилці `INVALID HANDLE VALUE`. Код помилки можна отримати викликавши функцію `GetLastError`.

Закриття порту

Відкритий порт повинен бути закритий перед завершенням роботи програми. В Win32 закриття об'єкта по його дескриптору виконує функція `CloseHandle`:

```
BOOL CloseHandle(
   HANDLE hObject
```

```
};
```

При успішному завершенні функція повертає не нульове значення, при помилці нуль.

Структура DCB

Основні параметри послідовного порту описуються структурою DCB. Часові параметри структурою COMMTIMEOUTS. Існує ще трохи інформаційних і керуючих структур, але вони використовуються рідше. Настроювання порту полягає в заповненні керуючих структур і наступному виклику функцій настроювання.

Основну інформацію містить структура DCB:

```
typedef struct _DCB {
  DWORD DCBlength;      // Задає довжину, у байтах, структури DCB.
  DWORD BaudRate;       // Швидкість передачі даних
  DWORD fBinary:1;      // Включає двійковий режим обміну
  DWORD fParity:1;      // Включає режим контролю парності
  DWORD fOutxCtsFlow:1;  // Включає режим спостереження за сигналом CTS
  DWORD fOutxDsrFlow:1;  // Включає режим спостереження за сигналом DSR
  DWORD fDtrControl:2;   // Задає режим керування обміном для сигналу DTR
  DWORD fDsrSensitivity:1;  // Задає чуттєвість комунікаційного драйвера до стану
                                лінії DSR
  DWORD fTXContinueOnXoff:1; // Задає, чи припиняється передача при переповненні
                                прийомного буфера й передачі драйвером
                                символу XoffChar
  DWORD fOut:1;         // Задає використання XON/XOFF керування потоком при
                                передачі
  DWORD fIn:1;          // Задає використання XON/XOFF керування потоком при
                                прийманні
  DWORD fErrorChar:1;  // Указує на необхідність заміни символів з помилкою
                                парності
  DWORD fNull:1;        // Визначає дія виконуване при прийманні нульового байта
  DWORD fRtsControl:2;  // Задає режим керування потоком для сигналу RTS
  DWORD fAbortOnError:1;  // Задає ігнорування всіх операцій читання/запису при
                                виникненні помилки
  DWORD fDummy2:17;       // Зарезервовано й не використовується
  WORD wReserved;         // Не використовується, повинне бути встановлене в 0
  WORD XonLim;          // Задає мінімальне число символів у прийомному буфері
                                перед посилкою символу XON
```

```
  WORD XoffLim;         // Визначає максимальна кількість байт у прийомному буфері
                                перед посилкою символу XOFF
  BYTE ByteSize;        // Визначає число інформаційних біт у переданих і
                                прийнятих байтах
  BYTE Parity;          // Визначає вибір схеми контролю парності
  BYTE StopBits;        // Задає кількість стопових біт
  char XonChar;         // Задає символ XON використовуваний як для прийому, так і
                                для передачі
  char XoffChar;        // Задає символ XOFF використовуваний як для прийому, так
                                і для передачі
  char ErrorChar;       // Задає символ, що використовується для заміни символів з
                                помилковою парністю
  char EofChar;         // Задає символ, що використовується для сигналізації про
                                кінець даних
  char EvtChar;         // Задає символ, що використовується для сигналізації про
                                подію
  WORD wReserved1;      // Зарезервовано й не використовується
} DCB;
```

Ця структура містить майже всю керуючу інформацію, що у реальності розташовується в різних регістрах послідовного порту.

DCBlength

Задає довжину, у байтах, структури `DCB`. Використовується для контролю коректності структури при передачі її адреси у функції настроювання порту.

BaudRate

Швидкість передачі даних. Можлива вказівка наступних констант: `CBR_110`, `CBR_300`, `CBR_600`, `CBR_1200`, `CBR_2400`, `CBR_4800`, `CBR_9600`, `CBR_14400`, `CBR_19200`, `CBR_38400`, `CBR_56000`, `CBR_57600`, `CBR_115200`, `CBR_128000`, `CBR_256000`. Ці константи відповідають всім стандартним швидкостям обміну. Насправді, це поле містить числове значення швидкості передачі, а константи просто є символічними іменами. Тому можна вказувати, наприклад, і `CBR_9600`, і просто `9600`. Однак рекомендується вказувати символічні константи.

fBinary

Включає двійковий режим обміну. Win32 не підтримує недвійковий режим, тому дане поле завжди Повинне дорівнювати `1`, або логічній константі

TRUE. У цьому режимі символ, що надійшов на вхід порту, заданний полем EofChar свідчив про кінець прийнятих даних.

fParity

Включає режим контролю парності. Якщо це поле дорівнює TRUE, то виконується перевірка парності, при помилці, у зухвалу програму, видається відповідний код завершення.

fOutxCtsFlow

Включає режим спостереження за сигналом CTS. Якщо це поле дорівнює TRUE і сигнал CTS скинутий, передача даних припиняється до установки сигналу CTS. Це дозволяє підключеному до комп'ютера приладу призупинити потік переданої в нього інформації, якщо він не встигає неї обробляти.

fOutxDsrFlow

Включає режим спостереження за сигналом DSR. Якщо це поле дорівнює TRUE і сигнал DSR скинутий, передача даних припиняється до установки сигналу DSR.

fDtrControl

Задає режим керування обміном для сигналу DTR. Це поле може приймати наступні значення:

– DTR_CONTROL_DISABLE. Забороняє використання лінії DTR.

– DTR_CONTROL_ENABLE. Дозволяє використання лінії DTR.

– DTR_CONTROL_HANDSHAKE. Дозволяє використання рукостискання для виходу з помилкових ситуацій. Цей режим використовується, зокрема, модемами при відновленні в ситуації втрати зв'язку.

fDsrSensitivity

Задає чуттєвість комунікаційного драйвера до стану лінії DSR. Якщо це поле дорівнює TRUE, то всі прийняті дані ігноруються драйвером (комунікаційний драйвер розташований в операційній системі), за винятком тих, які приймаються при встановленому сигналі DSR.

fTXContinueOnXoff

Задає, чи припиняється передача при переповненні прийомного буфера й передачі драйвером символу XoffChar. Якщо це поле дорівнює TRUE, то передача триває, незважаючи на те, що прийомний буфер містить більше XoffLim символів і близький до переповнення, а драйвер передав символ XoffChar для призупинення потоку прийнятих даних. Якщо поле дорівнює FALSE , то передача не буде продовжена доти, поки в прийомному буфері не залишиться менше XonLim символів і драйвер не передасть символ XonChar для поновлення потоку прийнятих даних. У такий спосіб це поле вводить якусь залежність між керуванням вхідним і вихідним потоками інформації.

fOut

Задає використання XON/XOFF керування потоком при передачі. Якщо це поле дорівнює TRUE, то передача зупиняється при прийманні символу XoffChar, і відновляється при прийманні символу XonChar.

fIn

Задає використання XON/XOFF керування потоком при прийманні. Якщо це поле дорівнює TRUE, то драйвер передає символ XoffChar, коли в прийомному буфері перебуває більше XoffLim, і XonChar, коли в прийомному буфері залишається менш XonLim символів.

fErrorChar

Указує на необхідність заміни символів з помилкою парності на символ задаваємий полем ErrorChar. Якщо це поле дорівнює TRUE, і поле fParity дорівнює TRUE, то виконується заміна.

fNull

Визначає дію виконувану при прийманні нульового байта. Якщо це поле TRUE, то нульові байти відкидаються при передачі.

fRtsControl

Задає режим керування потоком для сигналу RTS. Якщо це поле дорівнює 0, то за замовчуванням мається на увазі RTS_CONTROL_HANDSHAKE. Поле може

приймати одне з наступних значень: RTS_CONTROL_DISABLE Забороняє використання лінії RTS RTS_CONTROL_ENABLE Дозволяє використання лінії RTS RTS_CONTROL_HANDSHAKE Дозволяє використання RTS рукостискання. Драйвер установлює сигнал RTS коли прийомний буфер заповнений менш, ніж на половину, і скидає, коли буфер заповнюється більш ніж на три чверті. RTS_CONTROL_TOGGLE Задає, що сигнал RTS установлений, коли є дані для передачі. Коли всі символи з передавального буфера передані, сигнал скидається.

fAbortOnError

Задає ігнорування всіх операцій читання/запису при виникненні помилки. Якщо це поле дорівнює TRUE, драйвер припиняє всі операції читання/запису для порту при виникненні помилки. Продовжувати працювати з портом можна буде тільки після усунення причини помилки й виклику функції ClearCommError.

fDummy2

Зарезервовано й не використовується.

wReserved

Не використовується, повинне бути встановлене в 0.

XonLim

Задає мінімальне число символів у прийомному буфері перед посилкою символу XON.

XoffLim

Визначає максимальна кількість байт у прийомному буфері перед посилкою символу XOFF. Максимально припустима кількість байт у буфері обчислюється вирахуванням даного значення з розміру прийомного буфера в байтах.

ByteSize

Визначає число інформаційних біт у переданих і прийнятих байтах.

Parity

Визначає вибір схеми контролю парності. Дане поле повинне містити одне з наступних значень: EVENPARITY Доповнення до парності MARKPARITY Біт парності

завжди 1 NOPARITY Біт парності відсутній ODDPARITY Доповнення до непарності SPACEPARITY Біт парності завжди 0

StopBits

Задає кількість стопових біт. Поле може приймати наступні значення: ONESTOPBIT Один стоповий біт ONE5STOPBIT Півтора стопових біта TWOSTOPBIT Два стопових біти

XonChar

Задає символ XON використовуваний як для прийому, так і для передачі.

XoffChar

Задає символ XOFF використовуваний як для прийому, так і для передачі.

ErrorChar

Задає символ, що використовується для заміни символів з помилковою парністю.

EofChar

Задає символ, що використовується для сигналізації про кінець даних.

EvtChar

Задає символ, що використовується для сигналізації про подію.

wReserved1

Зарезервовано й не використовується.

Заповнення структури DCB

Структура COMMTIMEOUTS

winbase.h

```
typedef struct _COMMTIMEOUTS {
  DWORD ReadIntervalTimeout;     /* час у мілісекундах, що задає максимальний час,
                                    для інтервалу між надходженням
                                    двох символів у комунікаційну
                                    лінію */
  DWORD ReadTotalTimeoutMultiplier;  /* Множник, використовуваний, щоб обчислити
                                    повний період часу простою для
                                    операцій читання, у
                                    мілісекундах   */
  DWORD ReadTotalTimeoutConstant;   /* Константа, використовувана, щоб обчислити
                                    повний (максимальний) період
```

```
                                                часу простою для операцій
                                                читання, у мілісекундах    */
  DWORD WriteTotalTimeoutMultiplier; /* Множник, використовуваний, щоб обчислити
                                                повний період часу простою для
                                                операцій запису, у
                                                мілісекундах.    */
  DWORD WriteTotalTimeoutConstant;  /* Константа, використовувана, щоб обчислити
                                                повний період часу простою для
                                                операцій запису, у мілісекундах
                                                */
} COMMTIMEOUTS,*LPCOMMTIMEOUTS;
```

ReadIntervalTimeout – час у мілісекундах, що задає максимальний час, для інтервалу між надходженням двох символів у комунікаційну лінію. Якщо інтервал між надходженням будь-яких двох символів буде більше цієї величини, операція ReadFile завершується й будь-які буферизовані дані вертаються.

Щоб операція ReadFile негайно повертала керування з усіма отриманими даними (асинхронний режим) варто задавати наступні значення:

```
ReadIntervalTimeout=0xFFFFFFFF;
ReadTotalTimeoutConstant=0;
ReadTotalTimeoutMultiplier=0;
```

ReadTotalTimeoutMultiplier – Множник, використовуваний, щоб обчислити повний період часу простою для операцій читання, у мілісекундах. Для кожної операції читання, це значення множиться на викликане число байтів, які читаються.

ReadTotalTimeoutConstant – Константа, використовувана, щоб обчислити повний (максимальний) період часу простою для операцій читання, у мілісекундах. Для кожної операції читання, це значення додається до добутку члена структури ReadTotalTimeoutMultiplier і прочитаного числа байтів.

Значення нуля й для члена ReadTotalTimeoutMultiplier, і для члена ReadTotalTimeoutConstant указує, що повний час простою не використовуються для операцій читання.

WriteTotalTimeoutMultiplier – Множник, використовуваний, щоб обчислити повний період часу простою для операцій запису, у мілісекундах. Для кожної операції запису, це значення множиться на число записуваних байтів.

WriteTotalTimeoutConstant – Константа, використовувана, щоб обчислити повний період часу простою для операцій запису, у мілісекундах. Для кожної операції читання, це значення додається до добутку члена структури WriteTotalTimeoutMultiplier і записаного числа байтів.

Значення нуля й для члена WriteTotalTimeoutMultiplier, і для члена WriteTotalTimeoutConstant указує, що повний час простою не використовуються для операцій запису.

Заповнення структури COMMTIMEOUTS

Варіант 1: (максимальна затримка при читанні й записі = TIMEOUT)

```
COMMTIMEOUTS CommTimeOuts;
CommTimeOuts.ReadIntervalTimeout = 0xFFFFFFFF;
CommTimeOuts.ReadTotalTimeoutMultiplier = 0;
CommTimeOuts.ReadTotalTimeoutConstant = TIMEOUT;
CommTimeOuts.WriteTotalTimeoutMultiplier = 0;
CommTimeOuts.WriteTotalTimeoutConstant = TIMEOUT;
```

Варіант 2: Ініціалізація значеннями (без затримки при читанні)

```
COMMTIMEOUTS CommTimeOuts={0xFFFFFFFF,0,0,0,1500};
```

3.2 Розробка структурної схеми

Структурна схема розробленої системи зображена на рисунку 3.1. На ньому показано структуру обраного роз'єму та контролера послідовного порту.

З рисунку видно, що для емуляції послідовного порту обрано роз'єм типу DB-9, що містить наступні контакти:

1. Детектор сигналу, DCD.
2. Прийняті дані, RD.
3. Передані дані, TD.
4. Готовність передавача, DTR.
5. Заземлення, SG.
6. Готовність приймача, DSR.
7. Запит для передачі, RTS.

8. Скидання для передачі, CTS.
9. Індикатор виклику, RI.

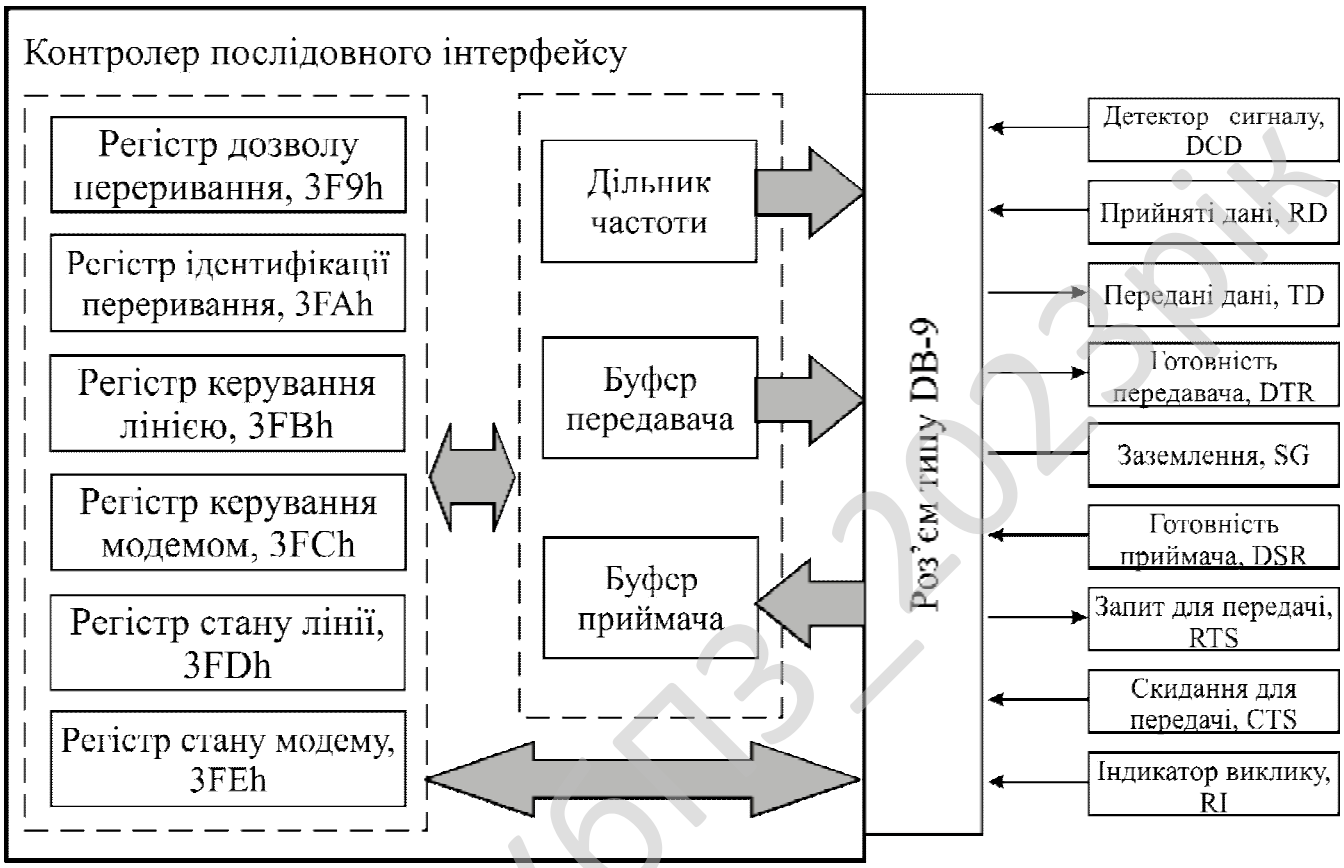

Рисунок 3.1 – Структурна схема розробленої системи

Сигнали послідовного інтерфейсу підрозділяються на наступні: послідовні дані, керуючі сигнали квитування та сигнали синхронізації.

Послідовні дані (TD, RD) – це два незалежних послідовних канали даних: первинний і вторинний. Обидва канали можуть працювати в дуплексному режимі, тобто одночасно здійснюють передачу й прийом інформації.

Керуючі сигнали квитування (RTS, CTS) – засіб, за допомогою якого обмін сигналами дозволяє комп'ютеру почати діалог з модемом до фактичної передачі або прийому даних по послідовній лінії зв'язку.

Сигнали синхронізації (DCD, DTR, DSR, RI). У синхронному режимі (на відміну від більш розповсюдженого асинхронного) між пристроями необхідно

передавати сигнали синхронізації, які спрощують синхронізм прийнятого сигналу з метою його декодування.

Контролер послідовного інтерфейсу містить:

– Дільник частоти, LSB/MSB.

– Буфер передавача, THR.

– Буфер приймача, RBR.

– Регістр дозволу переривання, IER.

– Регістр ідентифікації переривання, IIR.

– Регістр керування лінією, LCR.

– Регістр керування модемом, MCR.

– Регістр стану лінії, LSR.

– Регістр стану модему, MCR.

Значення дільника частоти міститься в двох регістрах: регістру буфера молодшого байта дільника (LSB) та регістру буфера старшого байта дільника (MSB). Ці регістри доступні по читанню й запису тільки при значенні біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR), рівному 1. При записі в цей регістр нового значення дільник відразу перезавантажується.

Регістр буфера передавача (THR) доступний тільки під час запису й при значенні біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR), рівному 0.

Регістр буфера приймача (RBR) доступний під час читання (IN) і при значенні біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR), рівному 0.

Регістр дозволу переривань (IER) доступний по читанню й запису, але тільки при значенні біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR), рівному 0. Цей регістр дозволяє управляти чотирма типами переривань, породжуваними контролером послідовного інтерфейсу.

Регістр ідентифікації переривання (IIR) доступний тільки по читанню й дозволяє одержати інформацію від контролера про очікуюче переривання.

Регістр керування лінією (LCR) доступний і під час читання, і під час запису. Значення даного регістра визначає формат переданих даних у лінію передачі даних контролером послідовного інтерфейсу.

Регістр керування модемом (MCR) доступний по читанню й запису. За допомогою регістра можна управляти роботою модему.

Регістр стану лінії (LSR) надає інформацію про стан обміну даних. Доступний тільки по читанню.

Регістр стану модему (MSR) надає інформацію про стан керуючих ліній модему. Крім того, цей регістр містить 4 біти, які відображають зміну стану модему й встановлюються в значення 0 після операції читання з регістра MSR.

3.3 Розробка функціональної схеми

На рисунку 3.2 зображена функціональна схема системи. Нижче розглянемо її більш докладно.

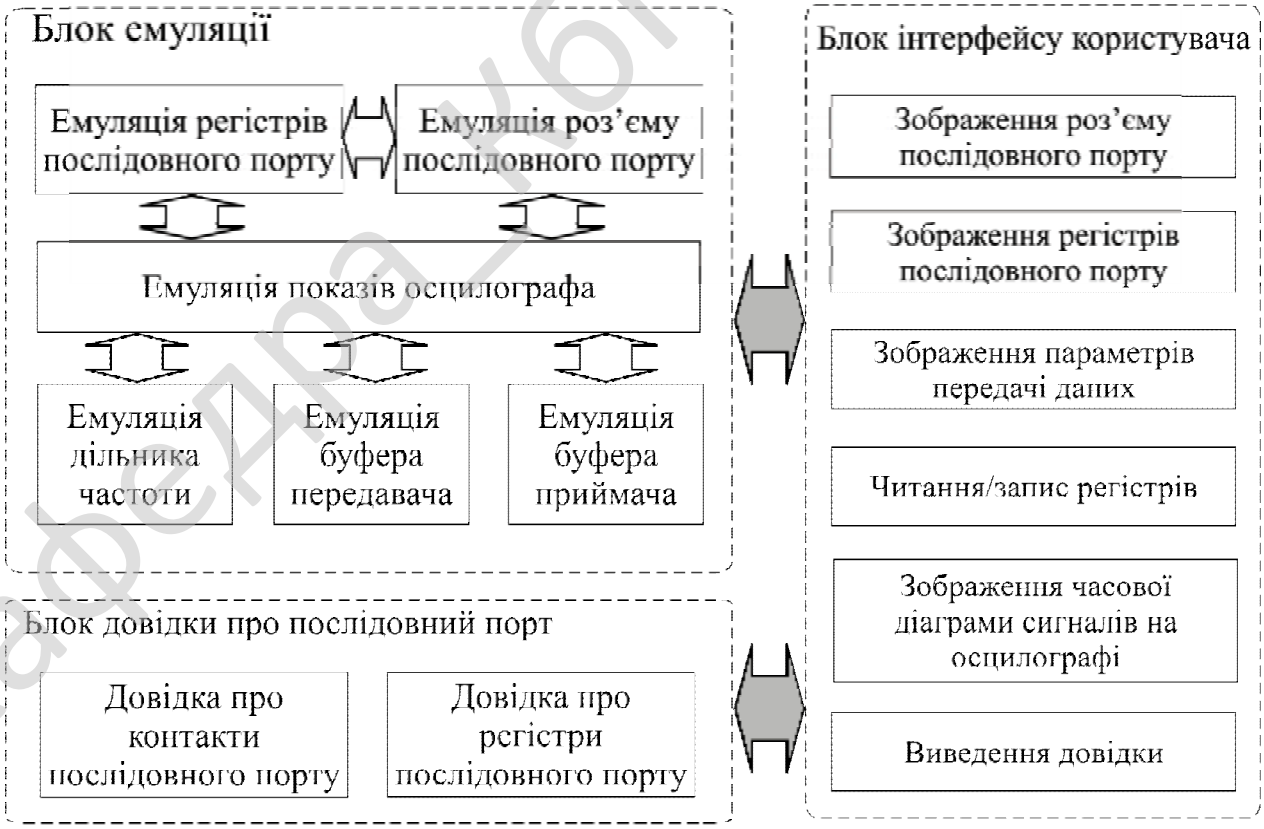

Рисунок 3.2 – Функціональна схема розробленої системи

З рисунка 3.2 бачимо, що функціонально схема розробленої системи складається з наступних функціональних блоків:

– Блок емуляції.

– Блок інтерфейсу користувача.

– Блок довідки про послідовний порт.

Блок емуляції складається з наступних функціональних блоків:

– Функціональний блок емуляції регістрів послідовного порту.

– Функціональний блок емуляції роз’єму послідовного порту.

– Функціональний блок емуляції показів осцилографа.

– Функціональний блок емуляції дільника частоти.

– Функціональний блок емуляції буфера передавача.

– Функціональний блок емуляції буфера приймача.

Блок інтерфейсу користувача складається з наступних функціональних блоків:

– Функціональний блок зображення роз’єму послідовного порту.

– Функціональний блок зображення регістрів послідовного порту.

– Функціональний блок зображення параметрів передачі даних.

– Функціональний блок читання/запису регістрів.

– Функціональний блок зображення часової діаграми сигналів на осцилографі.

– Функціональний блок виведення довідки.

Блок довідки про послідовний порт складається з наступних функціональних блоків.

– Функціональний блок довідки про контакти послідовного порту.

– Функціональний блок довідки про регістри послідовного порту.

Розглянувши усі блоки функціональної схеми перейдемо до розгляду діаграми взаємодії процесів, які відбуваються у системі.

3.4 Розробка діаграми процесів

Діаграма взаємодії процесів системи, розробленої у результаті виконання бакалаврського проектування, наведена на рисунку 3.3.

Робота системи починається з завантаження процесу початку/кінця робот програми. Цей процес взаємодіє з наступними процесами:

– Процесом запису/читання регістрів.

– Процесом виведення поточного стану послідовного порту.

Остатній процес, у свою чергу, взаємодіє з наступними процесами:

– Процесом запису/читання регістрів.

– Процесом виведення поточного стану регістрів.

– Процесом виведення зображення осцилографа.

– Процесом виведення стану контактів роз'єму порту.

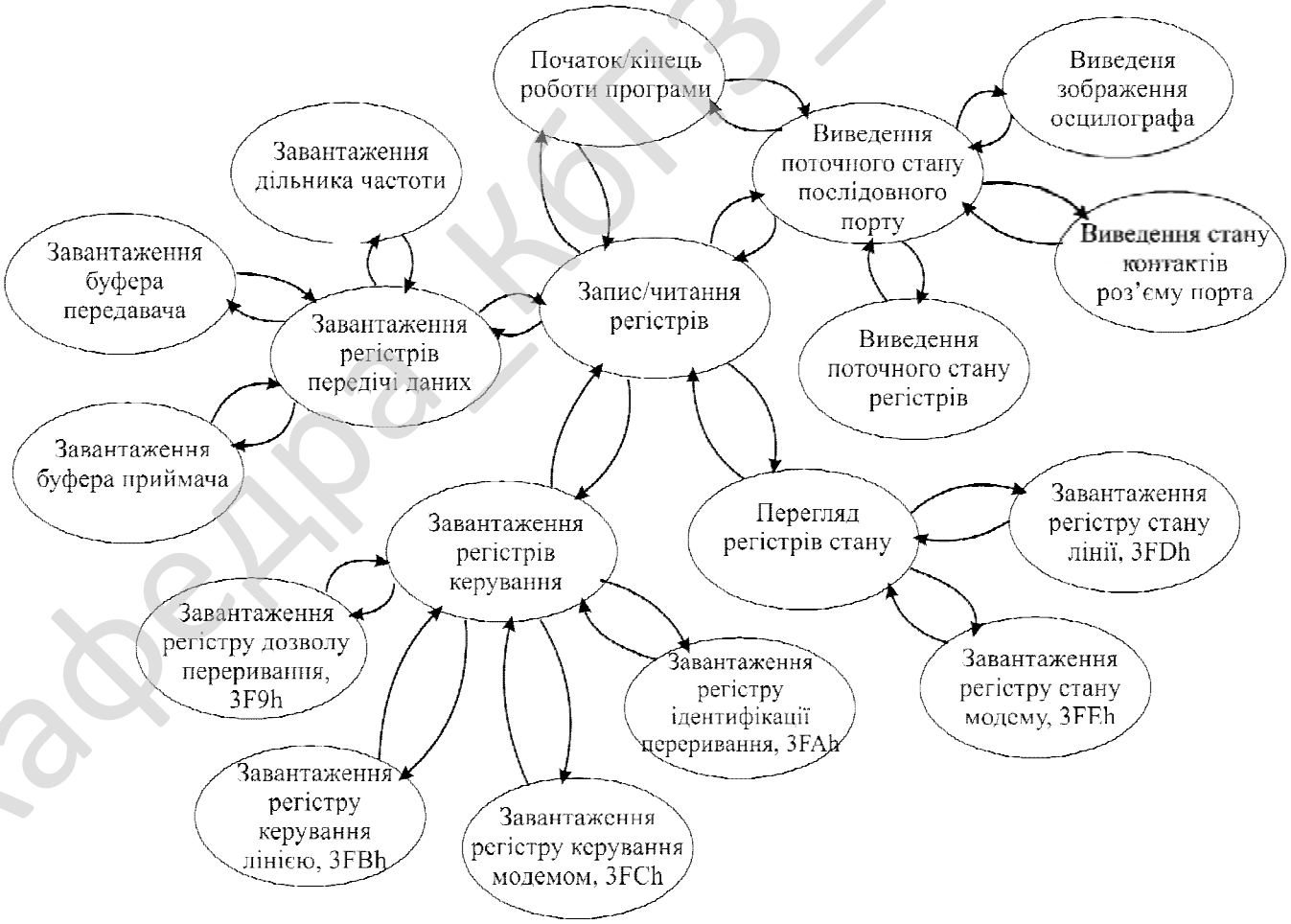

Рисунок 3.3 – Діаграма процесів розробленої системи

Процес запису/читання регістрів є одним з основних процесів та взаємодіє з наступними процесами:

– Процесом перегляду регістрів стану.

– Процесом завантаження регістрів керування.

– Процесом завантаження регістрів передачі даних.

Розглянемо. З якими процесами взаємодіють вищеперераховані процеси по черзі.

Спершу розглянемо взаємодію процесу перегляду регістрів стану. Він взаємодіє з наступними процесами:

– Процесом завантаження регістру стану лінії.

– Процесом завантаження регістру стану модему.

Наступним процесом, який ми розглянемо, буде процес завантаження регістрів керування. Він взаємодіє з наступними процесами:

– Процесом завантаження регістру ідентифікації переривання.

– Процесом завантаження регістру керування модемом.

– Процесом завантаження регістру керування лінією.

– Процесом завантаження регістру дозволу переривання.

Останнім блоковим процесом є процес завантаження регістрів передачі даних. Він взаємодіє з наступними процесами:

– Процесом завантаження буфер передавача.

– Процесом завантаження буфера приймача.

– Процесом завантаження дільника частоти.

Таким чином, розглянувши взаємодію процесів, які відбуваються у системі перейдемо до опису блок-схем алгоритмів основної програми та відповідних підпрограм.

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

4.1 Блок-схеми та опис алгоритмів функціонування системи

На рисунку 4.1 наведено блок-схему основної програми. Її робота складається з виконання наступних кроків.

Спершу відбувається виведення основного вікна програми. Після цього відбувається ініціалізація регістрів COM-порту.

Наступним кроком є подача сигналу готовності DTE пристрою DTR=1.

Після цього система переходить у стан очікування.

Якщо DCE готовий до з’єднання, то подається сигнал про наявність даних для передачі.

Після цього система переходить у стан очікування.

Якщо DCE готовий до прийому, то відбувається передача даних по лінії TD.

Після цього система переходить у стан очікування.

Якщо система отримує запит від DCE, то відбувається прийом даних по лінії RD.

Після цього користувач обирає завершувати зв'язок, або ні.

Розглянувши алгоритм роботи основної програми перейдемо до розгляду алгоритмів роботи підпрограм.

На рисунку 4.2 зображена блок-схема алгоритму ініціалізації послідовного інтерфейсу. З нього бачимо, що функція працює наступним чином.

Спершу задається кількість бітів даних у регістрі 3FBh, за допомогою бітів 0 та 1. Після цього задається кількість стоп-бітів у регістрі 3FBh за допомогою біту 2.

Рисунок 4.1 – Блок-схема алгоритму роботи основної програми

Наступним кроком є завдання способу контролю парності у регістрі 3FBh за допомогою бітів 3, 4 та 5.

Після цього користувач обирає у якому режимі йому працювати:

– Режимі звичайного функціонування.

– Режимі прийому/передачі.

Якщо користувач обирає працю у режимі звичайного функціонування, то виконуються наступні дії:

– Встановлюється у регістрі 3FBh біт 6 у стан «0».

– Встановлюється у регістрі 3FBh біт DLAB у стан «1».

– Завантажується молодший та старший байти дільника частоти в регістри 3F8h та 3F9h.

Якщо користувач обирає працю у режимі прийому/передачі, то виконуються наступні дії:

– Встановлюється у регістрі 3FBh біт 7 у стан «0».

– Встановлює сигнал на лінії DTR.

– Переходить у стан очікування.

Якщо користувач не обирає працю ні в одному з вищеперерахованих режимів, то виконуються наступні дії:

– Встановлюється у регістрі 3FBh біт 6 у стан «1».

– Видається сигнал BREAK.

Після виконання усіх вищеперерахованих дій відбувається виведення поточного стану послідовного інтерфейсу.

Нижче наведемо частини програмного коду, які реалізують вищеописані алгоритми.

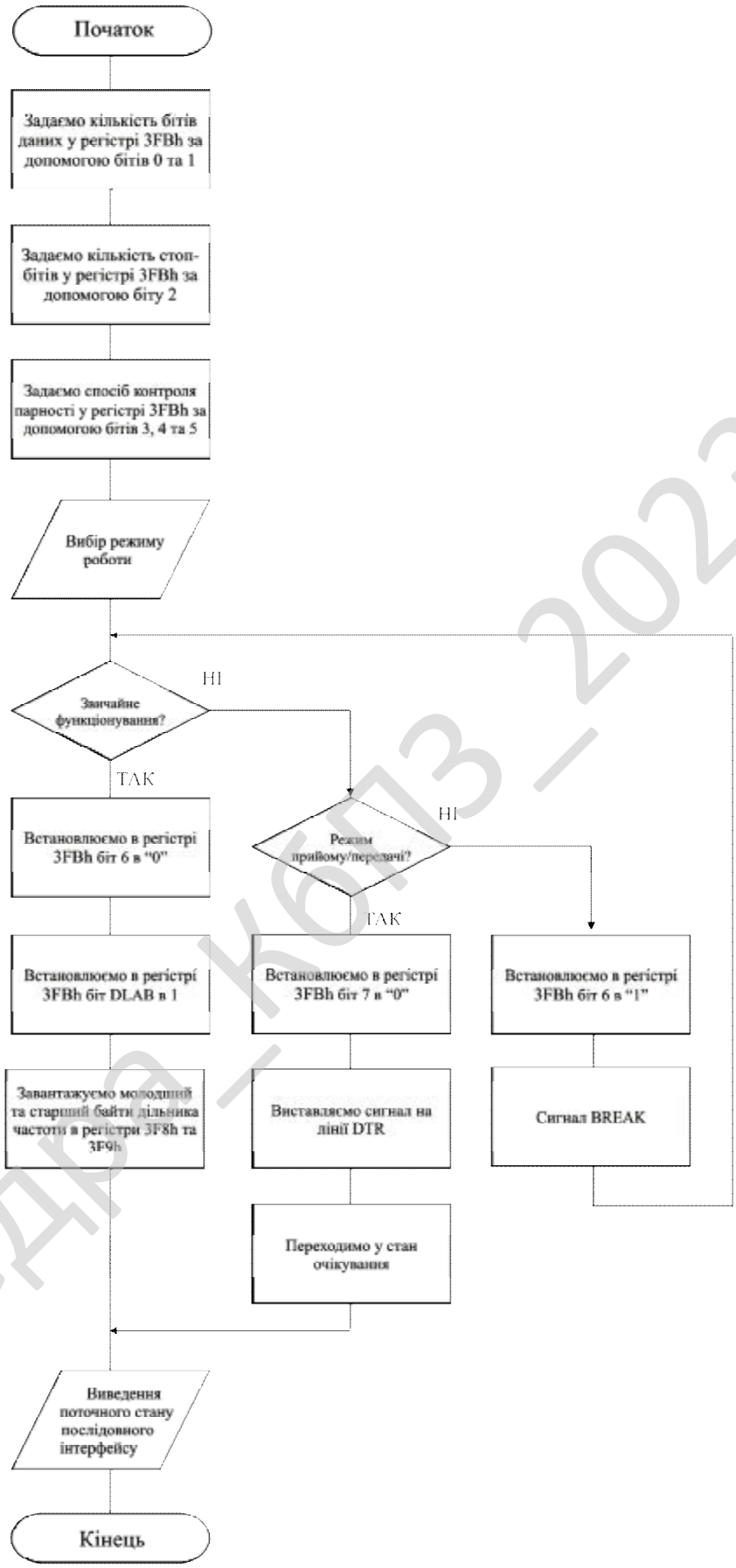

Рисунок 4.2 – Блок-схема алгоритму ініціалізації послідовного інтерфейсу

Наведемо код основної програми.

```
USEFORM("main.cpp", Form1);   // модуль основного вікна
USEFORM("graf.cpp", Form2);   // виведення на екран показів осцилографа
USEFORM("about.cpp", Form3);  // вікно "Про програму..."
USEFORM("help.cpp", Form4);   // довідка про регістри послідовного порту
USEFORM("help2.cpp", Form5);  // довідка про контакти послідовного порту
USEFORM("write1.cpp", Form6); // діалогове вікно запису у буфер передавача
                                          (регістр THR)
USEFORM("write2.cpp", Form7); // діалогове вікно запису у буфер приймача (регістр
                                          RBR)
USEFORM("write3.cpp", Form8); // діалогове вікно запису у молодший байт дільника
                                          (регістр LSB)
USEFORM("write4.cpp", Form9); // діалогове вікно запису у старший байт дільника
                                          (регістр MSB)
USEFORM("write5.cpp", Form10);// діалогове вікно запису у регістр 3F9h (IER)
USEFORM("write6.cpp", Form11);// діалогове вікно запису у регістр 3FAh (IIR)
USEFORM("write7.cpp", Form12);// діалогове вікно запису у регістр 3FBh (LCR)
USEFORM("write8.cpp", Form13);// діалогове вікно запису у регістр 3FCh ( MCR)
USEFORM("write9.cpp", Form14);// діалогове вікно запису у регістр 3FDh (LSR)
USEFORM("write10.cpp", Form15);// діалогове вікно запису у регістр 3FEh ( MSR)
//Опис головного вікна---------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
             Application->Initialize();
             Application->CreateForm(__classid(TForm1), &Form1);
             Application->CreateForm(__classid(TForm2), &Form2);
             Application->CreateForm(__classid(TForm3), &Form3);
             Application->CreateForm(__classid(TForm4), &Form4);
             Application->CreateForm(__classid(TForm5), &Form5);
             Application->CreateForm(__classid(TForm6), &Form6);
             Application->CreateForm(__classid(TForm7), &Form7);
             Application->CreateForm(__classid(TForm8), &Form8);
             Application->CreateForm(__classid(TForm9), &Form9);
             Application->CreateForm(__classid(TForm10), &Form10);
             Application->CreateForm(__classid(TForm11), &Form11);
             Application->CreateForm(__classid(TForm12), &Form12);
             Application->CreateForm(__classid(TForm13), &Form13);
             Application->CreateForm(__classid(TForm14), &Form14);
             Application->CreateForm(__classid(TForm15), &Form15);
             Application->Run();
```

```
        }

catch (Exception &exception)
        {
             Application->ShowException(&exception);
        }
        catch (...)
        {
                 try
                 {
                         throw Exception("");
                 }
                 catch (Exception &exception)
                 {
                     Application->ShowException(&exception);
                 }
        }
        return 0;
}
```

Наведемо код виведення поточного стану послідовного порту

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
      int x;
      if(MSR7->Text=='1') ImageDCD->Visible=false; else if(MSR7->Text=='0')
      ImageDCD->Visible=true;
      if(MCR0->Text=='1') ImageDTR->Visible=true; else if(MCR0->Text=='0')
      ImageDTR->Visible=false;
      if(MSR5->Text=='1') ImageDSR->Visible=false; else if(MSR5->Text=='0')
      ImageDSR->Visible=true;
      if(MCR1->Text=='1') ImageRTS->Visible=true; else if(MCR1->Text=='0')
      ImageRTS->Visible=false;
      if(MSR4->Text=='1') ImageCTS->Visible=false; else if(MSR4->Text=='0')
      ImageCTS->Visible=true;
      if(MSR6->Text=='1') ImageRI->Visible=false; else if(MSR6->Text=='0')
      ImageRI->Visible=true;
      if(LCR7->Text=='1') {
            Button1->Enabled=false;
            Button2->Enabled=false;
            Button3->Enabled=true;
            Button4->Enabled=true;
            Button5->Enabled=false;
            THR0->Enabled=false;
```

```
        THR1->Enabled=false;
        THR2->Enabled=false;
        THR3->Enabled=false;
        THR4->Enabled=false;
        THR5->Enabled=false;
        THR6->Enabled=false;
        THR7->Enabled=false;
        RBR0->Enabled=false;
        RBR1->Enabled=false;
        RBR2->Enabled=false;
        RBR3->Enabled=false;
        RBR4->Enabled=false;
        RBR5->Enabled=false;
        RBR6->Enabled=false;
        RBR7->Enabled=false;
        LSB0->Enabled=true;
        LSB1->Enabled=true;
        LSB2->Enabled=true;
        LSB3->Enabled=true;
        LSB4->Enabled=true;
        LSB5->Enabled=true;
        LSB6->Enabled=true;
        LSB7->Enabled=true;
        IER1->Enabled=false;
        IER2->Enabled=false;
        IER3->Enabled=false;
        IER4->Enabled=false;
        IER5->Enabled=false;
        IER6->Enabled=false;
        IER7->Enabled=false;
        MSB0->Enabled=true;
        MSB1->Enabled=true;
        MSB2->Enabled=true;
        MSB3->Enabled=true;
        MSB4->Enabled=true;
        MSB5->Enabled=true;
        MSB6->Enabled=true;
        MSB7->Enabled=true;
    }
    else if(LCR7->Text=='0')  {
        Button1->Enabled=true;
        Button2->Enabled=true;
```

```
        Button3->Enabled=false;
        Button4->Enabled=false;
        Button5->Enabled=true;
        THR0->Enabled=true;
        THR1->Enabled=true;
        THR2->Enabled=true;
        THR3->Enabled=true;
        THR4->Enabled=true;
        THR5->Enabled=true;
        THR6->Enabled=true;
        THR7->Enabled=true;
        RBR0->Enabled=true;
        RBR1->Enabled=true;
        RBR2->Enabled=true;
        RBR3->Enabled=true;
        RBR4->Enabled=true;
        RBR5->Enabled=true;
        RBR6->Enabled=true;
        RBR7->Enabled=true;
        LSB0->Enabled=false;
        LSB1->Enabled=false;
        LSB2->Enabled=false;
        LSB3->Enabled=false;
        LSB4->Enabled=false;
        LSB5->Enabled=false;
        LSB6->Enabled=false;
        LSB7->Enabled=false;
        IER1->Enabled=true;
        IER2->Enabled=true;
        IER3->Enabled=true;
        IER4->Enabled=true;
        IER5->Enabled=true;
        IER6->Enabled=true;
        IER7->Enabled=true;
        MSB0->Enabled=false;
        MSB1->Enabled=false;
        MSB2->Enabled=false;
        MSB3->Enabled=false;
        MSB4->Enabled=false;
        MSB5->Enabled=false;
        MSB6->Enabled=false;
        MSB7->Enabled=false;
    }
```

```
        x=StrToInt(LCR1->Text)*10+StrToInt(LCR0->Text);
        if(x==0) DataBit->Text=5;
        if(x==1) DataBit->Text=6;
        if(x==10) DataBit->Text=7;
        if(x==11) DataBit->Text=8;
        x=StrToInt(LCR2->Text);
        if(x==0) StopBit->Text=1;
        if(x==1) StopBit->Text=2;
        x=StrToInt(LCR4->Text)*10+StrToInt(LCR3->Text);
        if((x==0)|(x==10)) Parn->Text="немає";
        if(x==1) Parn->Text="контроль на непарність";
        if(x==11) Parn->Text="контроль на парність";
}
```

4.2 Захист розробленого програмного забезпечення

Захист розробленого програмного забезпечення буде відбуватися за допомогою Threefish – в криптографії симетричний блоковий криптоалгоритм, розроблений автором Blowfish та Twofish, американським криптографом Брюсом Шнайером 2008 року для використання в хеш-функції Skein і як універсальну заміну наявним блоковим шифрам. Основними принципами розробки шифру були: мінімальне використання пам'яті, необхідна для використання в хеш-функції стійкість до атак, простота реалізації та оптимізація під 64-розрядні процесори.

Структура алгоритму

Threefish має дуже просту структуру і може бути використаний для заміни алгоритмів блочного шифрування, бувши швидким і гнучким шифром, що працюють в довільному режимі шифрування. Threefish S-блоки не використовує, заснований на комбінації інструкцій виключаючого або, складання і циклічного зсуву.

Як і AES, шифр реалізований у вигляді підстановочно-перестановочної мережі на оборотних операціях, не будучи шифром мережі Фейстел.

Алгоритм передбачає використання tweak-значення, свого роду вектора ініціалізації, дозволяючи змінювати таким чином значення виходу, без зміни

ключа, що має позитивний ефект як для реалізації нових режимів шифрування, так і на криптостойкости алгоритму.

Як результат думки авторів, що кілька складних раундів часто гірше ніж застосування великого числа простих раундів, алгоритм має нетрадиційно велику кількість раундів – 72 або 80 при ключі 1024 біт, проте, за заявою творців, його швидкісні характеристики випереджають AES приблизно вдвічі. Варто зауважити, що через 64-бітної структури шифру, дана заява має місцево лише на 64-розрядної архітектури. Тому, Threefish, як і Skein [1], заснований на ньому, на 32-розрядних процесорах показує значно гірші результати ніж на «рідному» обладнанні.

Ядром шифру є проста функція «MIX», перетворювальна два 64-бітових беззнакових числа, в процесі якої відбувається складання, циклічний зсув (ROL / ROR), і додавання по модулю 2 (XOR) .

Нижче представлений код MIX-функції для Threefish-1024 [2]:

```
<syntaxhighlight lang="C">
// Константи для циклічного зсуву
int R16 [8] [8] = {
  {55, 43, 37, 40, 16, 22, 38, 12},
  {25, 25, 46, 13, 14, 13, 52, 57},
  {33, 8, 18, 57, 21, 12, 32, 54},
  {34, 43, 25, 60, 44, 9, 59, 34},
  {28, 7, 47, 48, 51, 9, 35, 41},
  {17, 6, 18, 25, 43, 42, 40, 15},
  {58, 7, 32, 45, 19, 18, 2, 56},
  {47, 49, 27, 58, 37, 48, 53, 56},
};
// D – раунд, j – індекс в таблиці циклічного зсуву
void mix (int j, int d) {
  usigned long long rotl;
  y [0] = x [0] + x [1];
  rotl = R16 [d% 8] [j];
  y [1] = (x [1] << rotl) | (x [1] >> (64 – rotl));
  y [1] ^ = y [0];
}
</Source>
```

Процедура розшифрування обернена процедурі зашифрування і містить зворотну функцію DEMIX.

Кожен з 72 раундів Threefish-256 і Threefish-512 має чотири MIX перетворення, Threefish-1024 – вісім звернень до MIX функції.

Безпека

За заявою авторів, алгоритм має більш високий рівень безпеки, ніж AES. Існує атака на 25 з 72 раундів Threefish, в той час як для AES – на 6 з 10. Threefish має показник фактора безпеки 2.9, в свою чергу, AES всього 1.7 [3]

Для досягнення повної дифузії, шифру Threefish-256 досить 9 раундів, Threefish-512 – 10 раундів і Threefish-1024 – 11 раундів. Виходячи з цього, 72 і 80 раундів відповідно в середньому, забезпечать кращі результати, ніж існуючі шифри. [4]

У той же час, алгоритм має набагато простішу структуру і функцію перетворення, проте виконання 72-80 раундів, на думку дослідників, забезпечує необхідну стійкість. Вживаний розмір ключа від 256 до 1024 біт зводить нанівець можливість повного перебору паролів при так званій атаці грубою силою (brute force attack) на сучасному обладнанні.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

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

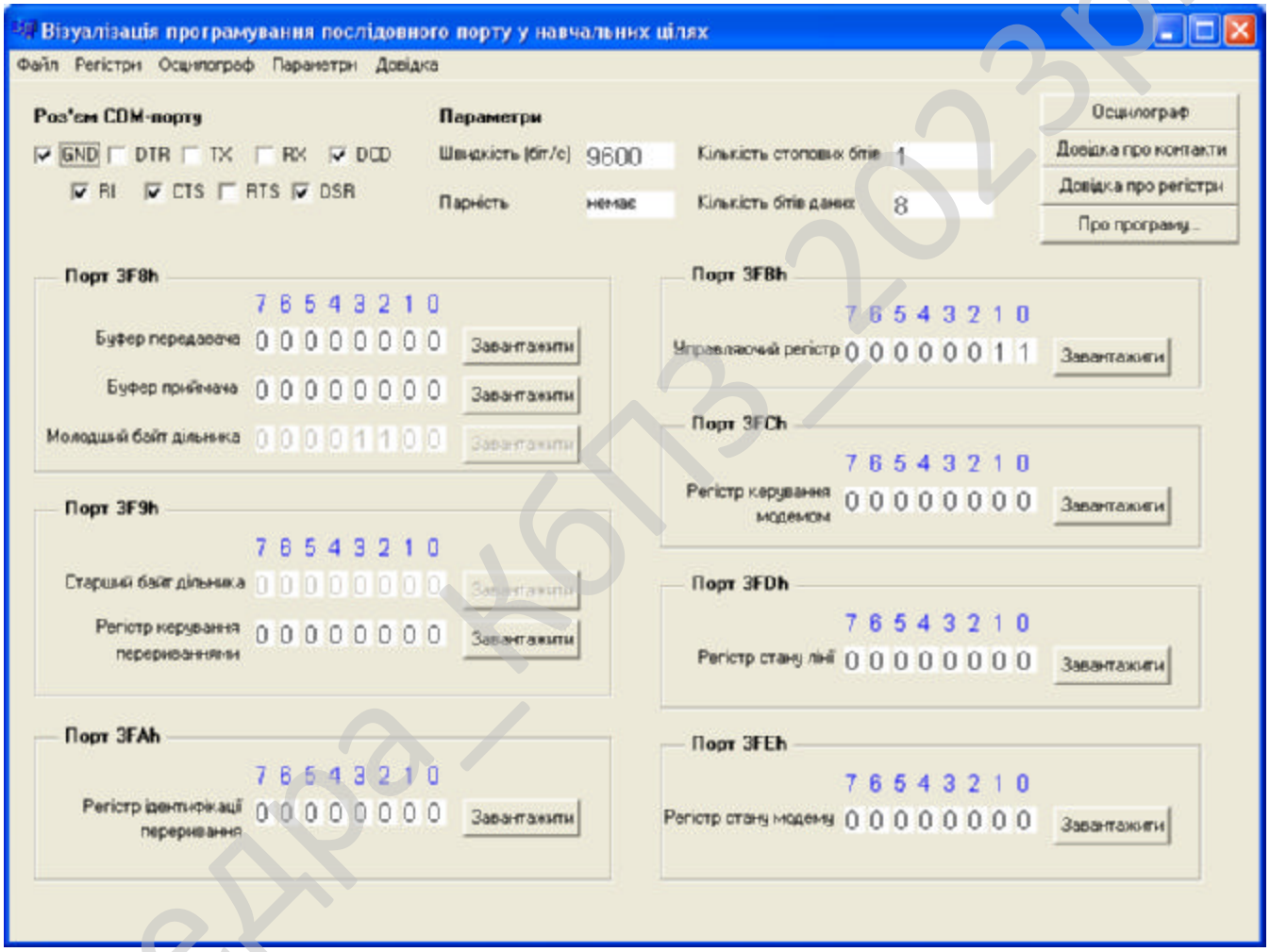

Рисунок 5.1 – Головне вікно програми

Програма складається з таких частин:

– Емуляція роз'єму послідовного порту.

– Емуляція регістрів.

– Покази осцилографа.

– Інтерактивна довідка.

У верхній правій частині головного вікна програми розміщене зображення стандартного роз'єму послідовного порту з назвами контактів. Якщо на контакті високий рівень сигналу, то його колір стає рожевим, у протилежному випадку – білим.

Також на головному вікні зображені всі регістри послідовного порту, їх назви, адреси та поточне значення. Існує можливість записати в них нове значення за допомогою кнопок "Завантажити" навпроти кожного регістра. При натисненні кнопки навпроти потрібного регістру, з'являється діалогове вікно, що зображене на рисунку 5.2.

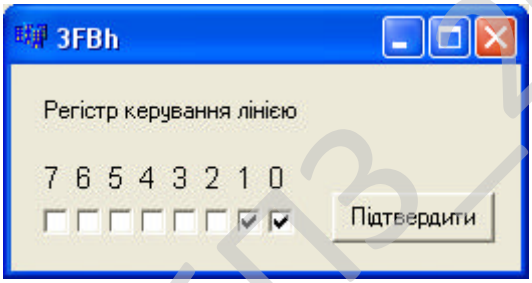

Рисунок 5.2 – Приклад діалогового вікна завантаження регістрів послідовного порту

Запис необхідного значення у регістр здійснюється за допомогою восьми чекбоксів, кожен з яких відповідає одному з восьми біт регістра. Порожній чекбокс відповідає логічному нулю, а з встановленим прапорцем – логічній одиниці. Після встановлення прапорців, для запису отриманого слова у регістр необхідно натиснути кнопку "Підтвердити".

В нижній частині головного вікна програми можна побачити обрані за допомогою управляючого регістра параметри передачі даних, в які входять:

– Швидкість передачі даних.

– Парність.

– Кількість стопових біт.

– Кількість біт даних.

Часову діаграму сигналів на контактах роз'єму послідовного порту можна переглянути за допомогою осцилографа (рисунок 5.3), вікно якого відкривається за допомогою кнопки "Осцилограф", або пункту меню Діагностика–>Осцилограф.

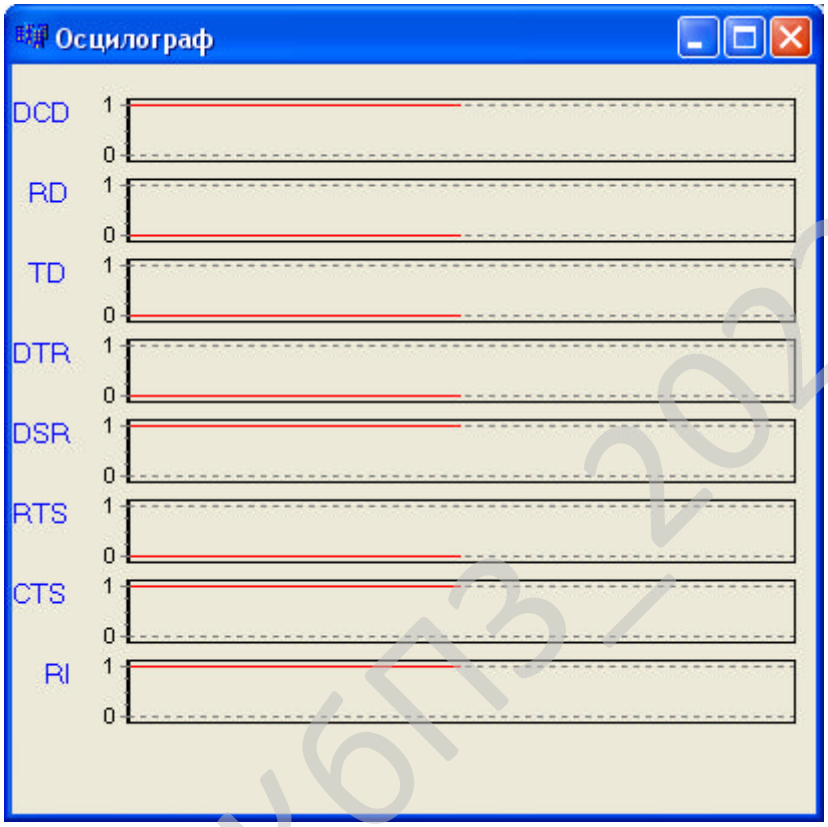

Рисунок 5.3 – Вікно осцилографа

Для зручності роботи з програмою були створені інтерактивна довідка про регістри (рисунок 5.4) та довідка про роз'єм послідовного порту (рисунок 5.5).

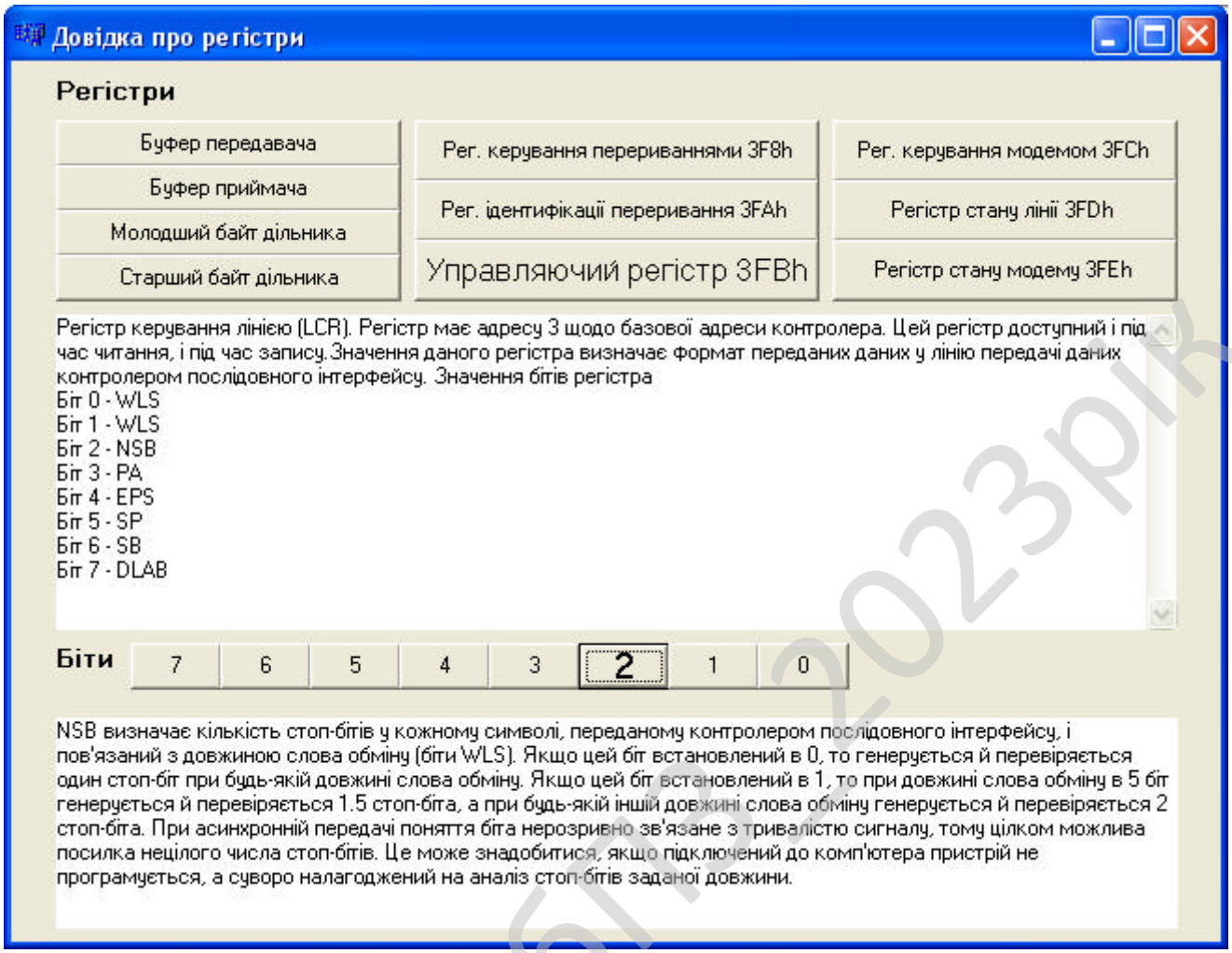

Рисунок 5.4 – Довідка про регістри послідовного порту

Довідка про регістри містить два поля, в першому розміщується інформація про обраний регістр, у другому інформація про обраний біт даного регістра. Вибір регістрів та їх біт здійснюється за допомогою кнопок, що містять назви регістрів та номери біт.

Довідка про роз'єм містить назви та призначення контактів послідовного порту.

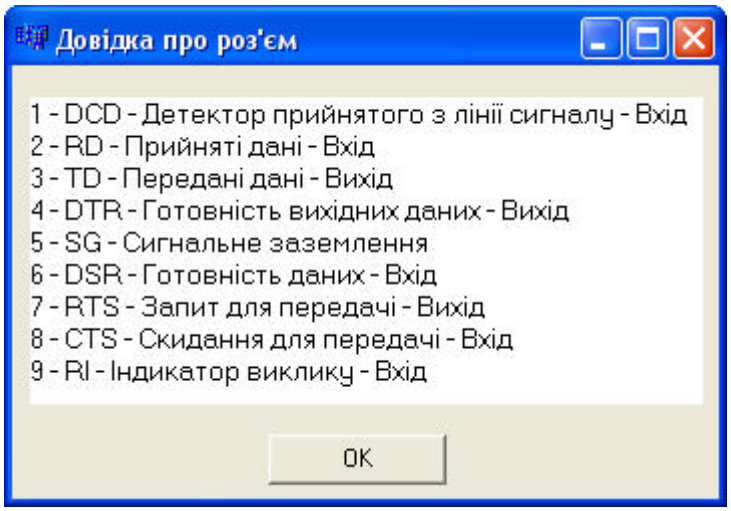

Рисунок 5.5 – Довідка про контакти послідовного порту

Коротку довідку про розроблену систему можна переглянути, натиснувши кнопку "Про програму…" (рисунок 5.8).

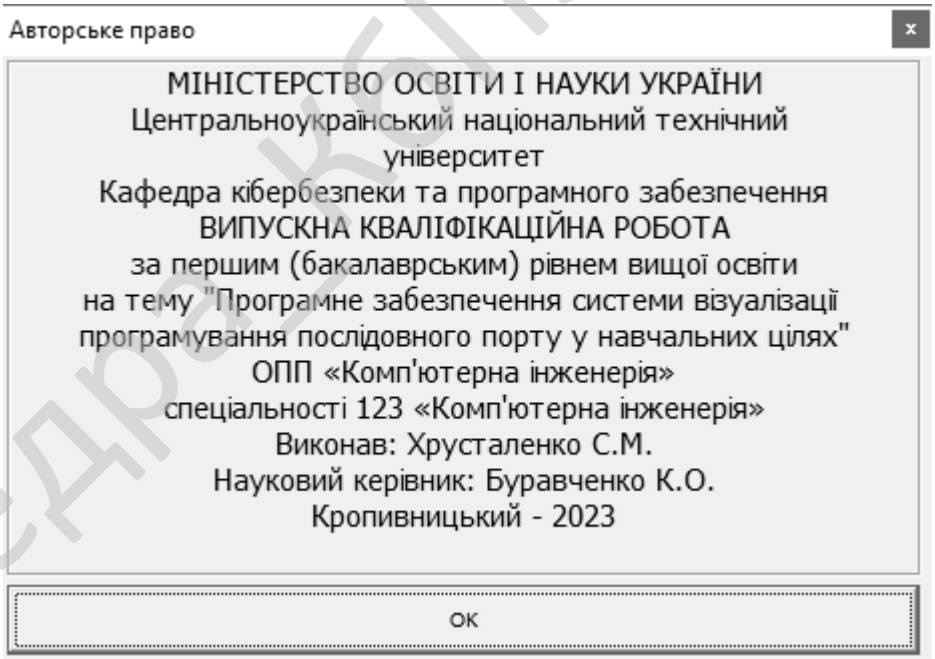

Рисунок 5.8 – Вікно "Про програму…"

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти, призначено для системи візуалізації програмування послідовного порту у навчальних цілях.

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 програмування послідовного порту у навчальних цілях.

– Досліджена система візуалізації програмування послідовного порту у навчальних цілях.

– На основі отриманих результатів досліджень створена програмна реалізація системи візуалізації програмування послідовного порту у навчальних цілях.

Розроблені під час виконання випускної кваліфікаційної роботи за першим (бакалаврським) рівнем вищої освіти алгоритми дозволяють успішно вирішувати завдання візуалізації програмування послідовного порту у навчальних цілях.

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 програмування послідовного порту у навчальних цілях. 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 Threefish.

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. Kovalenko O., Popereshnyak S., Grinenko S., Grinenko O., Radivilova T. «Methods for Assessing the Maturity Levels of Software Ecosystems». *CEUR Workshop Proceedings* Volume 2654, 2019, Pages 251-261. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=5633fba897776a6e0f3d5633fbcd3fbe **(Scopus)**.

2. Kovalenko O., Drieieva H., Simakhin V., Bondar S., Drieiev O., Zhumadilova M. «Multifractal Properties of Traffic Generator Based on Markov Chains ». *CEUR Workshop Proceedings* Volume 2588, 2019, Pages 567-579. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=176e2cada8976a6e0f3d5633fbcd3fbe **(Scopus)**.

3. Kovalenko Oleksandr Qualitative risk analysis of software development / Oleksandr Kovalenko, Jamil Al-Azzeh, Oleksii Smirnov, Anna Kovalenko, Serhii Smirnov // Asian Journal of Information Technology. – Volume 17 Issue 3. – Medwell Journals. – 2018. – P. 218-230. ISSN: 1682-3915. URL: http://medwelljournals.com/abstract/?doi=ajit.2018.218.230 Doi: ajit.2018.218.230

4. Kovalenko Oleksandr, The mathematical model of the testing technology for DOM XSS vulnerabilities / O. Kovalenko, O. Smirnov, A.Kovalenko, S. Smirnov, V. Vialkova // Scientific & practical cyber security journal (SPCSJ) Volume 2 Issue 1, P. 22-28. Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2018 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/04-3-o.kovalenko-a.kovalenko-o.smirnov-s.smirnov-v.vialkova.pdf

5. Коваленко А.В. Технология тестирования DOM XSS уязвимости / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Scientific & practical cyber security journal (SPCSJ) Volume 1. Issue 1. P. 38-45 Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2017 ISSN: 2587-

4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/8-dom-xss-testing-technology-vulnerabilities.pdf

6. Коваленко О.В. Моделі та методи розроблення програмного забезпечення комп'ютерних систем для підвищення безпеки даних: монографія / О.В. Коваленко // К.: Вид. «КОД» – 2019. – 305 с.

7. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

8. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

9. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

10. Коваленко А.В. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник наукових праць "Системи обробки інформації". – Випуск 4(120). – Х.: ХУПС – 2014. – С. 161-164.

11. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 5(142). – Х.: ХУПС – 2016. – С. 153-157.

12. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць "Системи обробки інформації". –

Випуск 3(140). – Х.: ХУПС – 2016. – С. 40-42.

13. Коваленко А.В. Метод качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

14. Коваленко А.В. Метод количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

15. Коваленко А.В. Использование псевдобулевых методов бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

16. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 2 (38). – Полтава: ПолтНТУ. – 2016. – С. 93-100.

17. Коваленко А.В. Технология тестирования уязвимости к SQL инъекциям / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 5 (45). – Полтава: ПолтНТУ. – 2017. – С. 66-71.

18. Коваленко А.В. Масштабирование имитационной модели технологии тестирования безопасности / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (46). – Полтава: ПолтНТУ. – 2017. – С. 181-184.

19. Коваленко А.В. Имитационная модель технологии тестирования безопасности Web-приложений / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 1 (47). – Полтава: ПолтНТУ. – 2018. – С. 114-123.

20. Коваленко О.В. Методи якісного аналізу та кількісної оцінки ризиків розроблення програмного забезпечення/ О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 3 (49). – Полтава: ПолтНТУ. – 2018. – С. 116-125.

21. Коваленко О.В. Управління ризиками розроблення програмного

забезпечення за умови обмеженості коштів виділених на усунення помилок безпеки/ О.В. Коваленко // Техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Випуск 31. – Кропивницький: ЦНТУ. – 2018. – С. 128-140.

22. Коваленко О.В. GERT-мережевий синтез технології тестування на вразливість WEB-додатків/ О.В. Коваленко // Захист інформації. – Випуск 20(2) – К.: НАУ. – 2018. – С. 89-94.

23. Коваленко О.В. Імітаційна модель технології тестування безпеки на основі положень теорії масштабування / О.В. Коваленко // Безпека інформації. – Випуск 24 (2). – К.: НАУ. – 2018. – С. 110-117.

24. Коваленко О.В. Оцінка ефективності технології тестування безпеки / О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 2, 2018. – С. 137-141

25. Коваленко О.В. Методи та засоби управління безпекою додатків / О.В. Коваленко // Інформаційно-керуючі системи на залізничному транспорті. №4, 2018. – С. 41-44.

26. Коваленко О.В. Розробка інформаційної технології передтестової компіляції та розподілу доступу / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 4 (50). – Полтава: ПолтНТУ. – 2018. – С. 115-119.

27. Коваленко О.В. Аналіз та дослідження інформаційних технологій розробки програмного забезпечення/ О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 5, 2018. – С. 131-137.

28. Коваленко О.В. Удосконалений метод управління ризиками розроблення програмного забезпечення на основі напівмарковської моделі прийняття рішень/ О.В. Коваленко // Сучасні інформаційні системи. – Випуск 2(3). – Харків. – 2018. – С. 41-48.

29. Коваленко О.В. Математичні моделі технології тестування DOM XSS

вразливості та вразливості до SQL ін'єкцій / О.В. Коваленко // Вісник Черкаського державного технологічного університету. Серія : Технічні науки №4, 2018. – С. 29-36.

30. Коваленко О.В. Математична модель технології тестування вразливості до SQL ін’єкцій / О.В. Коваленко // Системи управління, навігації та зв’язку. – Випуск 6 (58). – Полтава: ПолтНТУ. – 2019. – С. 43-47.

31. Коваленко О.В. Математична модель технології тестування комплексу DOM XSS вразливостей для аналітичної оцінки часових витрат / О.В. Коваленко // Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 173-180, 2019.

32. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць II міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 24-27 лютого 2016 р. – Київ: Європейський університет. – 2016. – С. 138-139.

33. Коваленко А.В. Анализ и оценка рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез «Securitea informationala 2015-2016». Conferenta internationala (editia a XII-a). Chisinau. Moldova. 3 martie 2016. – Chisinau: ADSEM. – 2016. – P. 96-102.

34. Коваленко А.В. Исследование источников и причин риска разработки программного обеспечения, этапов и работ, при выполнении которых возникает риск / А.В. Коваленко, А.А. Смирнов // Збірник тез VII всеукраїнської науково-практичної конференції "Інформатика та системні науки (ІСН-2016)". м. Полтава. 10-12 березня 2016 р. – Полтава.: ПУЕТ – 2016. – С. 264-266.

35. Коваленко А.В. Оценка показателя чистой приведенной стоимости для количественной оценки рисков проекта разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних

систем”. м. Київ. 10-11 березня 2016 р. – Київ: КНУ ім. Тараса Шевченко – 2016. – С. 81-82.

36. Коваленко А.В. Методика структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 24-25 березня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 71-72.

37. Коваленко А.В. Методы качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2016). м. Харків. 30 березня – 1 квітня 2016 р. – Харків: НТУ «ХПІ». – 2016. – С. 6-7.

38. Коваленко А.В. Структурная идентификация рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 15-16 квітня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 175-182.

39. Коваленко А.В. Исследование разработанной методики структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез VIII міжнародної науково-практичної конференції “Проблеми і перспективи розвитку IT-індустрії”. м. Харків. 28-29 квітня 2016 р. – Харків: ХНЕУ. – 2016. – С. 49.

40. Коваленко А.В. Исследование дерева рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез III міжнародної науково-практичної конференції «Інформаційна та економічна безпека» (INFECO-2016)». м. Харків. 28-30 квітня 2016 р. – Харків: ХННІ ДВНЗ «УБС». – 2016. – С. 174-178.

41. Коваленко А.В. Методы качественного анализа и количественной

оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Сборник тезисов XII международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 30 мая – 02 июня 2016 г – Варна. ТУВ. – 2016. – С. 585-589.

42. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Матеріали Всеукраїнської науково-практичної конференції «Кібербезпека в Україні: правові та організаційні питання». м. Одеса, 21 жовтня 2016 р. – Одеса : ОДУВС, 2016. – С.146-148.

43. Коваленко А.В. Метод управления рисками разработки программного обеспечения с использованием псевдобулевых методов бивалентного программирования / А.В. Коваленко, А.А. Смирнов // Матеріали Всеукраїнської науково-практичної конференції «Актуальні задачі та досягнення у галузі кібербезпеки». м. Кропивницький, 23-25 листопада 2016 року – Кропивницький: ЦНТУ, 2016. – С. 162.

44. Коваленко А.В. Псевдобулевые методы бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко, С.А. Смирнов // Збірник наукових праць III міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 22-25 лютого 2017 р. – Київ: Європейський університет. – 2017. – С. 158-162.

45. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез II науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем”. м. Київ. 23-24 березня 2017 р. – Київ: КНУ ім. Тараса Шевченко – 2017. – С. 203-205.

46. Коваленко А.В. Алгоритмы анализа уязвимостей при управлении рисками разработки программного обеспечения / А.В. Коваленко,

А.А. Смирнов, А.С. Коваленко // Conferenta internationala (editia a XIII-a). «Securitea informationala 2017». Chisinau. Republic of Moldova. 4-5 aprilie 2017. – Chisinau: ADSEM. – 2017. – P. 19-22.

47. Коваленко А.В. Алгоритм анализа DOM XSS уязвимости при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез дев'ятнадцятого міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кропивницький 7-8 квітня 2017 р. – Кропивницький: ГЛА НАУ. – 2017. – С. 125-127.

48. Коваленко А.В. Алгоритм анализа уязвимости SQL Injection для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез другої міжнародної науково-технічної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2017). м. Харків.10-12 квітня 2017 р. – Харків: НТУ «ХПІ». – 2017. – С. 27.

49. Коваленко А.В. Метод управления рисками разработки программного обеспечения на основе алгоритмов анализа уязвимостей / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 20-22 квітня 2017 р. Кіровоград: КНТУ. – 2017. – С. 92.

50. Коваленко А.В. Алгоритмы анализа DOM XSS уязвимости и уязвимости SQL Injection при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез IX міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 20-21 квітня 2017 р. – Харків: ХНЕУ. – 2017. – С. 61.

51. Kovalenko O.V. Method of testing the dom xss vulnerability / Kovalenko Oleksandr, Kovalenko Anna, Smirnov Oleksii, Smirnov Serhii // International

Conference «information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. – 2017. – P. 7.

52. Коваленко О.В. Метод тестування DOM XSS уразливості / О.В. Коваленко, О.А. Смірнов, А.С. Коваленко, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп’ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті». м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

53. Коваленко О.В. GERT-модель технології тестування DOM XSS уразливості / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць IV міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 21-24 лютого 2018 р. – Київ: Європейський університет. – 2018. – С. 65-70.

54. Коваленко О.В. Технології тестування уразливостей Web-застосунків з використанням GERT-моделі / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної конференції "Комп'ютерні інтелектуальні системи та мережі (КІСМ-2018)". м. Кривий Ріг. 21-23 березня 2018 р. – Кривий Ріг.: ДВНЗ КНУ – 2018. – С. 227-230.

55. Коваленко А.В. Тестирование уязвимости Web-приложений к атаке вида межсайтовый скриптинг / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез «Securitea informationala 2018». Conferenta internationala (editia a XIV-a). Chisinau. Moldova. 20-21 martie 2018. – Chisinau: ADSEM. – 2018. – P. 54-56.

56. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез X міжнародної науково-практичної

конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 19-20 квітня 2018 р. – Харків: ХНЕУ. – 2018. – С. 38.

57. Коваленко О.В. Розробка методу передтестової компіляції й розподілу доступу / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницькй. 19-20 квітня 2018 р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215.

58. Коваленко О.В. Оцінка ефективності технологій тестування безпеки уразливостей DOM XSS й SQL-ін'єкцій / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Сборник тезисов XIV международной конференции "Стратегия качества в промышленности и образовании", Варна, Болгария. 04-07 июня 2018 г – Варна. ТУВ. – 2018. – С. 271-274.

59. Коваленко О.В. Аналіз основних підходів математичного моделювання та методологій для забезпечення максимальних показників безпеки програмного забезпечення / О.В. Коваленко, А.С. Коваленко // Збірник наукових праць всеукр. наук.-практ. конф. здобувачів вищої освіти й молодих учених «Комп’ютерна інженерія і кібербезпека : досягнення та інновації, м. Кропивницькй. 27-29 листопада

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 програмування послідовного порту у навчальних цілях.

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. № 10-02 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 програмування послідовного порту у навчальних цілях.

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 програмування послідовного порту у навчальних цілях;

– 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 – 86 аркушів.

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

_____ Буравченко К.О.

Програмне забезпечення системи візуалізації програмування послідовного порту у навчальних цілях

Лістинг програми

Код документу 12

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

Загальна кількість аркушів: 56

Літера: РП

Кропивницький – 2023 року

Основна програма

Файл Project1.cpp основної програми

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

#include <vcl.h>
#pragma hdrstop
//Визначення основних модулів-----------------------------
USEFORM("main.cpp", Form1);   // модуль основного вікна
USEFORM("graf.cpp", Form2);// виведення на екран показів осцилографа
USEFORM("about.cpp", Form3);// вікно "Про програму..."
USEFORM("help.cpp", Form4);// довідка про регістри послідовного порту
USEFORM("help2.cpp", Form5);// довідка про контакти послідовного порту
USEFORM("write1.cpp",  Form6);// діалогове вікно запису у буфер передавача
                                        (регістр THR)
USEFORM("write2.cpp", Form7);// діалогове вікно запису у буфер приймача (регістр
                                        RBR)
USEFORM("write3.cpp", Form8);// діалогове вікно запису у молодший байт дільника
                                        (регістр LSB)
USEFORM("write4.cpp",  Form9);// діалогове вікно запису у старший байт дільника
                                        (регістр MSB)
USEFORM("write5.cpp", Form10);// діалогове вікно запису у регістр 3F9h (IER)
USEFORM("write6.cpp", Form11);// діалогове вікно запису у регістр 3FAh (IIR)
USEFORM("write7.cpp", Form12);// діалогове вікно запису у регістр 3FBh (LCR)
USEFORM("write8.cpp", Form13);// діалогове вікно запису у регістр 3FCh ( MCR)
USEFORM("write9.cpp", Form14);// діалогове вікно запису у регістр 3FDh (LSR)
USEFORM("write10.cpp", Form15);// діалогове вікно запису у регістр 3FEh ( MSR)
//Опис головного вікна-----------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
             Application->Initialize();
             Application->CreateForm(__classid(TForm1), &Form1);
             Application->CreateForm(__classid(TForm2), &Form2);
             Application->CreateForm(__classid(TForm3), &Form3);
             Application->CreateForm(__classid(TForm4), &Form4);
             Application->CreateForm(__classid(TForm5), &Form5);
             Application->CreateForm(__classid(TForm6), &Form6);
             Application->CreateForm(__classid(TForm7), &Form7);
             Application->CreateForm(__classid(TForm8), &Form8);
             Application->CreateForm(__classid(TForm9), &Form9);
             Application->CreateForm(__classid(TForm10), &Form10);
             Application->CreateForm(__classid(TForm11), &Form11);
             Application->CreateForm(__classid(TForm12), &Form12);
             Application->CreateForm(__classid(TForm13), &Form13);
             Application->CreateForm(__classid(TForm14), &Form14);
             Application->CreateForm(__classid(TForm15), &Form15);
             Application->Run();
        }

         catch (Exception &exception)
        {
             Application->ShowException(&exception);
        }
        catch (...)
        {
                 try
                 {
                         throw Exception("");
                 }
                 catch (Exception &exception)
                 {
                     Application->ShowException(&exception);
```

```
                }
        }
        return 0;
}
//---------------------------------------------------------------
```

Файл Main.cpp основної програми

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

#include <vcl.h>
#pragma hdrstop

#include "main.h"
#include "graf.h"
#include "about.h"
#include "help.h"
#include "help2.h"
#include "write1.h"
#include "write2.h"
#include "write3.h"
#include "write4.h"
#include "write5.h"
#include "write6.h"
#include "write7.h"
#include "write8.h"
#include "write9.h"
#include "write10.h"


//---------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
int d1,d2,d3,d4,d5,d6,d7,d8;
//---------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------

//Процедура, що обробляє натиснення кнопки вмикання/вимикання осцилографа

void __fastcall TForm1::Button12Click(TObject *Sender)
{
if(Form2->Visible==false) Form2->Show();
else if(Form2->Visible==true) Form2->Close();
}
//---------------------------------------------------------------

//Відкриття вікна довідки про роз'єми

void __fastcall TForm1::Button15Click(TObject *Sender)
{
Form5->Show();
}
//---------------------------------------------------------------



void __fastcall TForm1::N19Click(TObject *Sender)
{
Form5->Show();
}
//---------------------------------------------------------------

//Відкриття вікна "Про програму..."

void __fastcall TForm1::N15Click(TObject *Sender)
{
Form3->Show();
}
//---------------------------------------------------------------
```

```
void __fastcall TForm1::Button11Click(TObject *Sender)
{
Form3->Show();
}
//---------------------------------------------------------------------

//Відкриття вікна довідки про регістри

void __fastcall TForm1::Button14Click(TObject *Sender)
{
Form4->Show();
}
//---------------------------------------------------------------------

void __fastcall TForm1::N18Click(TObject *Sender)
{
Form4->Show();
}
//---------------------------------------------------------------------

//Відкриття вікна з показами осцилографа через відповідний пункт меню програми

void __fastcall TForm1::N17Click(TObject *Sender)
{
Form2->Show();
}
//---------------------------------------------------------------------

//Відкриття діалогового вікна запису у регістр THR

void __fastcall TForm1::Button1Click(TObject *Sender)
{
Form6->Show();
}
//---------------------------------------------------------------------

void __fastcall TForm1::N4Click(TObject *Sender)
{
Form6->Show();
}
//---------------------------------------------------------------------

//Відкриття діалогового вікна запису у регістр RBR

void __fastcall TForm1::Button2Click(TObject *Sender)
{
Form7->Show();
}
//---------------------------------------------------------------------

void __fastcall TForm1::N5Click(TObject *Sender)
{
Form7->Show();
}
//---------------------------------------------------------------------

//Відкриття діалогового вікна запису у регістр LSB

void __fastcall TForm1::Button3Click(TObject *Sender)
{
Form8->Show();
}

//---------------------------------------------------------------------
void __fastcall TForm1::N6Click(TObject *Sender)
{
Form8->Show();
}
```

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

//Відкриття діалогового вікна запису у регістр MSB

void __fastcall TForm1::Button4Click(TObject *Sender)
{
Form9->Show();
}
//---------------------------------------------------------------

void __fastcall TForm1::N7Click(TObject *Sender)
{
Form9->Show();
}
//---------------------------------------------------------------

//Відкриття діалогового вікна запису у регістр IER

void __fastcall TForm1::Button5Click(TObject *Sender)
{
Form10->Show();
}

//---------------------------------------------------------------
void __fastcall TForm1::N8Click(TObject *Sender)
{
Form10->Show();
}
//---------------------------------------------------------------


//Відкриття діалогового вікна запису у регістр IIR

void __fastcall TForm1::Button6Click(TObject *Sender)
{
Form11->Show();
}

//---------------------------------------------------------------
void __fastcall TForm1::N9Click(TObject *Sender)
{
Form11->Show();
}
//---------------------------------------------------------------

//Відкриття діалогового вікна запису у регістр LCR

void __fastcall TForm1::Button7Click(TObject *Sender)
{
Form12->Show();
}

//---------------------------------------------------------------
void __fastcall TForm1::N10Click(TObject *Sender)
{
Form12->Show();
}
//---------------------------------------------------------------

//Відкриття діалогового вікна запису у регістр MCR

void __fastcall TForm1::Button8Click(TObject *Sender)
{
Form13->Show();
}
//---------------------------------------------------------------

void __fastcall TForm1::N11Click(TObject *Sender)
{
```

```
Form13->Show();
}
//---------------------------------------------------------------

//Відкриття діалогового вікна запису у регістр LSR

void __fastcall TForm1::Button9Click(TObject *Sender)
{
Form14->Show();
}
//---------------------------------------------------------------

void __fastcall TForm1::N12Click(TObject *Sender)
{
Form14->Show();
}
//---------------------------------------------------------------

//Відкриття діалогового вікна запису у регістр MSR

void __fastcall TForm1::Button13Click(TObject *Sender)
{
Form15->Show();
}
//---------------------------------------------------------------

void __fastcall TForm1::N14Click(TObject *Sender)
{
Form16->Show();
}
//---------------------------------------------------------------

//Виведення поточного стану послідовного порту

void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
int x;

if(MSR7->Text=='1') ImageDCD->Visible=false; else if(MSR7->Text=='0') ImageDCD-
>Visible=true;
if(MCR0->Text=='1') ImageDTR->Visible=true; else if(MCR0->Text=='0') ImageDTR-
>Visible=false;
if(MSR5->Text=='1') ImageDSR->Visible=false; else if(MSR5->Text=='0') ImageDSR-
>Visible=true;
if(MCR1->Text=='1') ImageRTS->Visible=true; else if(MCR1->Text=='0') ImageRTS-
>Visible=false;
if(MSR4->Text=='1') ImageCTS->Visible=false; else if(MSR4->Text=='0') ImageCTS-
>Visible=true;
if(MSR6->Text=='1') ImageRI->Visible=false; else if(MSR6->Text=='0') ImageRI-
>Visible=true;
if(LCR7->Text=='1') {
Button1->Enabled=false;
Button2->Enabled=false;
Button3->Enabled=true;
Button4->Enabled=true;
Button5->Enabled=false;
THR0->Enabled=false;
THR1->Enabled=false;
THR2->Enabled=false;
THR3->Enabled=false;
THR4->Enabled=false;
THR5->Enabled=false;
THR6->Enabled=false;
THR7->Enabled=false;
RBR0->Enabled=false;
RBR1->Enabled=false;
RBR2->Enabled=false;
RBR3->Enabled=false;
RBR4->Enabled=false;
```

```
RBR5->Enabled=false;
RBR6->Enabled=false;
RBR7->Enabled=false;
LSB0->Enabled=true;
LSB1->Enabled=true;
LSB2->Enabled=true;
LSB3->Enabled=true;
LSB4->Enabled=true;
LSB5->Enabled=true;
LSB6->Enabled=true;
LSB7->Enabled=true;

IER1->Enabled=false;
IER2->Enabled=false;
IER3->Enabled=false;
IER4->Enabled=false;
IER5->Enabled=false;
IER6->Enabled=false;
IER7->Enabled=false;
MSB0->Enabled=true;
MSB1->Enabled=true;
MSB2->Enabled=true;
MSB3->Enabled=true;
MSB4->Enabled=true;
MSB5->Enabled=true;
MSB6->Enabled=true;
MSB7->Enabled=true;
}
else if(LCR7->Text=='0')  {
Button1->Enabled=true;
Button2->Enabled=true;
Button3->Enabled=false;
Button4->Enabled=false;
Button5->Enabled=true;
THR0->Enabled=true;
THR1->Enabled=true;
THR2->Enabled=true;
THR3->Enabled=true;
THR4->Enabled=true;
THR5->Enabled=true;
THR6->Enabled=true;
THR7->Enabled=true;
RBR0->Enabled=true;
RBR1->Enabled=true;
RBR2->Enabled=true;
RBR3->Enabled=true;
RBR4->Enabled=true;
RBR5->Enabled=true;
RBR6->Enabled=true;
RBR7->Enabled=true;
LSB0->Enabled=false;
LSB1->Enabled=false;
LSB2->Enabled=false;
LSB3->Enabled=false;
LSB4->Enabled=false;
LSB5->Enabled=false;
LSB6->Enabled=false;
LSB7->Enabled=false;

IER1->Enabled=true;
IER2->Enabled=true;
IER3->Enabled=true;
IER4->Enabled=true;
IER5->Enabled=true;
IER6->Enabled=true;
IER7->Enabled=true;
MSB0->Enabled=false;
MSB1->Enabled=false;
MSB2->Enabled=false;
```

```
MSB3->Enabled=false;
MSB4->Enabled=false;
MSB5->Enabled=false;
MSB6->Enabled=false;
MSB7->Enabled=false;
}

x=StrToInt(LCR1->Text)*10+StrToInt(LCR0->Text);
if(x==0) DataBit->Text=5;
if(x==1) DataBit->Text=6;
if(x==10) DataBit->Text=7;
if(x==11) DataBit->Text=8;

x=StrToInt(LCR2->Text);
if(x==0) StopBit->Text=1;
if(x==1) StopBit->Text=2;

x=StrToInt(LCR4->Text)*10+StrToInt(LCR3->Text);
if((x==0)|(x==10)) Parn->Text="немає";
if(x==1) Parn->Text="контроль на непарність";
if(x==11) Parn->Text="контроль на парність";

}
//---------------------------------------------------------------------

void __fastcall TForm1::N2Click(TObject *Sender)
{
Form1->Close();
}
//---------------------------------------------------------------------
```

Файл Main.h – бібліотека для файлу Main.cpp

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

#ifndef mainH
#define mainH
//---------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Graphics.hpp>
#include <Buttons.hpp>
#include <Menus.hpp>
#include <Chart.hpp>
#include <Series.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:      // Компоненти IDE-управління
        TImage *Image1;
        TImage *ImageDCD;
        TImage *Image3RX;
        TImage *ImageTX;
        TImage *ImageDTR;
        TImage *ImageGND;
        TImage *ImageDSR;
        TImage *ImageRTS;
        TImage *ImageCTS;
        TImage *ImageRI;
        TLabel *Label1;
        TLabel *Label2;
        TEdit *THR7;
        TEdit *THR6;
        TEdit *THR5;
        TEdit *THR4;
        TEdit *THR3;
        TEdit *THR2;
        TEdit *THR1;
        TEdit *THR0;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TLabel *Label10;
        TEdit *RBR7;
        TEdit *RBR6;
        TEdit *RBR5;
        TEdit *RBR4;
        TEdit *RBR3;
        TEdit *RBR2;
        TEdit *RBR1;
        TEdit *RBR0;
        TEdit *LSB7;
        TEdit *LSB6;
        TEdit *LSB5;
        TEdit *LSB4;
        TEdit *LSB3;
        TEdit *LSB2;
        TEdit *LSB1;
        TEdit *LSB0;
```

```
TButton *Button1;
TButton *Button2;
TButton *Button3;
TEdit *MSB7;
TEdit *MSB6;
TEdit *MSB5;
TEdit *MSB4;
TEdit *MSB3;
TEdit *MSB2;
TEdit *MSB1;
TEdit *MSB0;
TLabel *Label13;
TLabel *Label14;
TLabel *Label15;
TLabel *Label16;
TLabel *Label17;
TLabel *Label18;
TLabel *Label19;
TLabel *Label20;
TEdit *IER7;
TEdit *IER6;
TEdit *IER5;
TEdit *IER4;
TEdit *IER3;
TEdit *IER2;
TEdit *IER1;
TEdit *IER0;
TButton *Button4;
TButton *Button5;
TEdit *IIR7;
TEdit *IIR6;
TEdit *IIR5;
TEdit *IIR4;
TEdit *IIR3;
TEdit *IIR2;
TEdit *IIR1;
TEdit *IIR0;
TLabel *Label22;
TLabel *Label23;
TLabel *Label24;
TLabel *Label25;
TLabel *Label26;
TLabel *Label27;
TLabel *Label28;
TLabel *Label29;
TButton *Button6;
TEdit *LCR7;
TEdit *LCR6;
TEdit *LCR5;
TEdit *LCR4;
TEdit *LCR3;
TEdit *LCR2;
TEdit *LCR1;
TEdit *LCR0;
TLabel *Label31;
TLabel *Label32;
TLabel *Label33;
TLabel *Label34;
TLabel *Label35;
TLabel *Label36;
TLabel *Label37;
TLabel *Label38;
TButton *Button7;
TEdit *MCR7;
TEdit *MCR6;
TEdit *MCR5;
TEdit *MCR4;
TEdit *MCR3;
TEdit *MCR2;
```

```
TEdit *MCR1;
TEdit *MCR0;
TLabel *Label40;
TLabel *Label41;
TLabel *Label42;
TLabel *Label43;
TLabel *Label44;
TLabel *Label45;
TLabel *Label46;
TLabel *Label47;
TButton *Button8;
TEdit *LSR7;
TEdit *LSR6;
TEdit *LSR5;
TEdit *LSR4;
TEdit *LSR3;
TEdit *LSR2;
TEdit *LSR1;
TEdit *LSR0;
TLabel *Label49;
TLabel *Label50;
TLabel *Label51;
TLabel *Label52;
TLabel *Label53;
TLabel *Label54;
TLabel *Label55;
TLabel *Label56;
TButton *Button9;
TMainMenu *MainMenu1;
TMenuItem *N1;
TMenuItem *N2;
TMenuItem *N3;
TMenuItem *N4;
TMenuItem *N5;
TMenuItem *N6;
TMenuItem *N7;
TMenuItem *N8;
TMenuItem *N9;
TMenuItem *N10;
TMenuItem *N11;
TMenuItem *N12;
TMenuItem *N13;
TMenuItem *N14;
TMenuItem *N15;
TButton *Button10;
TButton *Button12;
TBevel *Bevel1;
TBevel *Bevel2;
TBevel *Bevel3;
TBevel *Bevel4;
TBevel *Bevel5;
TBevel *Bevel6;
TBevel *Bevel7;
TBevel *Bevel8;
TLabel *Label11;
TLabel *Label12;
TBevel *Bevel9;
TLabel *Label21;
TBevel *Bevel10;
TLabel *Label30;
TBevel *Bevel11;
TBevel *Bevel12;
TLabel *Label39;
TLabel *Label48;
TLabel *Label57;
TLabel *Label58;
TLabel *Label59;
TLabel *Label60;
TLabel *Label61;
```

```
TLabel *Label62;
TLabel *Label63;
TLabel *Label64;
TLabel *Label65;
TLabel *Label66;
TLabel *Label67;
TLabel *Label68;
TLabel *Label69;
TLabel *Label70;
TLabel *Label71;
TLabel *Label72;
TLabel *Label73;
TEdit *MSR7;
TEdit *MSR6;
TEdit *MSR5;
TEdit *MSR4;
TEdit *MSR3;
TEdit *MSR2;
TEdit *MSR11;
TEdit *MSR0;
TButton *Button13;
TBevel *Bevel13;
TBevel *Bevel14;
TLabel *Label74;
TLabel *Label75;
TMenuItem *N16;
TMenuItem *N17;
TMenuItem *N18;
TMenuItem *N19;
TLabel *Label76;
TEdit *Edit81;
TLabel *Label77;
TEdit *DataBit;
TLabel *Label78;
TEdit *Parn;
TLabel *Label79;
TEdit *StopBit;
TLabel *Label80;
TButton *Button11;
TButton *Button15;
TButton *Button14;
TMenuItem *MSR1;
TTimer *Timer1;
TLabel *Label81;
TLabel *Label82;
TLabel *Label83;
TLabel *Label84;
TLabel *Label85;
TLabel *Label86;
TLabel *Label87;
void __fastcall Button12Click(TObject *Sender);
void __fastcall Button15Click(TObject *Sender);
void __fastcall N15Click(TObject *Sender);
void __fastcall Label57Click(TObject *Sender);
void __fastcall Button11Click(TObject *Sender);
void __fastcall Button14Click(TObject *Sender);
void __fastcall N19Click(TObject *Sender);
void __fastcall N18Click(TObject *Sender);
void __fastcall N17Click(TObject *Sender);
void __fastcall Button1Click(TObject *Sender);
void __fastcall Button2Click(TObject *Sender);
void __fastcall Button3Click(TObject *Sender);
void __fastcall Button4Click(TObject *Sender);
void __fastcall Button5Click(TObject *Sender);
void __fastcall Button6Click(TObject *Sender);
void __fastcall Button7Click(TObject *Sender);
void __fastcall Button8Click(TObject *Sender);
void __fastcall Button9Click(TObject *Sender);
void __fastcall Button13Click(TObject *Sender);
```

```
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall N5Click(TObject *Sender);
        void __fastcall N6Click(TObject *Sender);
        void __fastcall N4Click(TObject *Sender);
        void __fastcall N7Click(TObject *Sender);
        void __fastcall N8Click(TObject *Sender);
        void __fastcall N9Click(TObject *Sender);
        void __fastcall N10Click(TObject *Sender);
        void __fastcall N11Click(TObject *Sender);
        void __fastcall MSR1Click(TObject *Sender);
        void __fastcall N12Click(TObject *Sender);
        void __fastcall N2Click(TObject *Sender);
        void __fastcall Button10Click(TObject *Sender);
private:    // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Файл graf.cpp – виведення на екран показів осцилографа

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

#include <vcl.h>
#pragma hdrstop
#include "graf.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm2 *Form2;
int t1,t2,t3,t4,t5,t6,t7,t8;
//---------------------------------------------------------------------
__fastcall TForm2::TForm2(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

//Часова діаграма сигналу на контакті DCD

void __fastcall TForm2::Timer1Timer(TObject *Sender)
{
int d;
d=(StrToInt(Form1->MSR7->Text)+1)%2;
Series1->Add(d,t1,clRed);
t1++;
if(t1==20) {t1=0; Series1->Clear();}
}
//---------------------------------------------------------------------

//Часова діаграма сигналу на контакті RD

void __fastcall TForm2::Timer2Timer(TObject *Sender)
{
Series2->Add(0,t2,clRed);
t2++;
if(t2==20) {t2=0; Series2->Clear();}
}
//---------------------------------------------------------------------

//Часова діаграма сигналу на контакті TD

void __fastcall TForm2::Timer3Timer(TObject *Sender)
{

Series3->Add(0,t3,clRed);
t3++;
if(t3==20) {t3=0; Series3->Clear();}
}
//---------------------------------------------------------------------



//Часова діаграма сигналу на контакті DTR

void __fastcall TForm2::Timer4Timer(TObject *Sender)
{
int d;
d=StrToInt(Form1->MCR0->Text);
Series4->Add(d,t4,clRed);
t4++;
if(t4==20) {t4=0; Series4->Clear();}
}
//---------------------------------------------------------------------
```

```
//Часова діаграма сигналу на контакті DSR

void __fastcall TForm2::Timer5Timer(TObject *Sender)
{
int d;
d=(StrToInt(Form1->MSR5->Text)+1)%2;
Series5->Add(d,t5,clRed);
t5++;
if(t5==20) {t5=0; Series5->Clear();}
}
//---------------------------------------------------------------

//Часова діаграма сигналу на контакті RTS

void __fastcall TForm2::Timer6Timer(TObject *Sender)
{
int d;
d=StrToInt(Form1->MCR1->Text);
Series6->Add(d,t6,clRed);
t6++;
if(t6==20) {t6=0; Series6->Clear();}
}
//---------------------------------------------------------------

//Часова діаграма сигналу на контакті CTS

void __fastcall TForm2::Timer7Timer(TObject *Sender)
{
int d;
d=(StrToInt(Form1->MSR4->Text)+1)%2;
Series7->Add(d,t7,clRed);
t7++;
if(t7==20) {t7=0; Series7->Clear();}
}
//---------------------------------------------------------------
//Часова діаграма сигналу на контакті RI

void __fastcall TForm2::Timer8Timer(TObject *Sender)
{
int d;
d=(StrToInt(Form1->MSR6->Text)+1)%2;
Series8->Add(d,t8,clRed);
t8++;
if(t8==20) {t8=0; Series8->Clear();}
}
```

Файл graf.h – бібліотека для файлу graf.cpp

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

#ifndef grafH
#define grafH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Chart.hpp>
#include <ExtCtrls.hpp>
#include <Series.hpp>
#include <TeEngine.hpp>
#include <TeeProcs.hpp>
//---------------------------------------------------------------------
class TForm2 : public TForm
{
__published:      // Компоненти IDE-управління
        TChart *Chart1;
        TLineSeries *Series1;
        TLabel *Label1;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TLabel *Label9;
        TChart *Chart2;
        TLineSeries *Series2;
        TChart *Chart3;
        TLineSeries *Series3;
        TChart *Chart4;
        TLineSeries *Series4;
        TChart *Chart5;
        TLineSeries *Series5;
        TChart *Chart6;
        TLineSeries *Series6;
        TChart *Chart7;
        TLineSeries *Series7;
        TChart *Chart8;
        TLineSeries *Series8;
        TTimer *Timer1;
        TTimer *Timer2;
        TTimer *Timer3;
        TTimer *Timer4;
        TTimer *Timer5;
        TTimer *Timer6;
        TTimer *Timer7;
        TTimer *Timer8;
        TLabel *Label2;
        void __fastcall Timer1Timer(TObject *Sender);
        void __fastcall Timer2Timer(TObject *Sender);
        void __fastcall Timer3Timer(TObject *Sender);
        void __fastcall Timer4Timer(TObject *Sender);
        void __fastcall Timer5Timer(TObject *Sender);
        void __fastcall Timer6Timer(TObject *Sender);
        void __fastcall Timer7Timer(TObject *Sender);
        void __fastcall Timer8Timer(TObject *Sender);
private:    // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm2(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm2 *Form2;
//---------------------------------------------------------------------
```

```
#endif
```

Файл write1.cpp – діалогове вікно запису у буфер передавача (регістр THR)

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

#include <vcl.h>
#pragma hdrstop

#include "write1.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm6 *Form6;
//---------------------------------------------------------------------
__fastcall TForm6::TForm6(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm6::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->THR0->Text=1; else Form1->THR0->Text=0;
if(CheckBox2->State==cbChecked) Form1->THR1->Text=1; else Form1->THR1->Text=0;
if(CheckBox3->State==cbChecked) Form1->THR2->Text=1; else Form1->THR2->Text=0;
if(CheckBox4->State==cbChecked) Form1->THR3->Text=1; else Form1->THR3->Text=0;
if(CheckBox5->State==cbChecked) Form1->THR4->Text=1; else Form1->THR4->Text=0;
if(CheckBox6->State==cbChecked) Form1->THR5->Text=1; else Form1->THR5->Text=0;
if(CheckBox7->State==cbChecked) Form1->THR6->Text=1; else Form1->THR6->Text=0;
if(CheckBox8->State==cbChecked) Form1->THR7->Text=1; else Form1->THR7->Text=0;
Form6->Close();
}
//---------------------------------------------------------------------
```

Файл write1.h – бібліотека для файлу write1.cpp

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

#ifndef write1H
#define write1H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm6 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:  // Визначається користувачем
public:     // Визначається користувачем
        __fastcall TForm6(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm6 *Form6;
//---------------------------------------------------------------------
#endif
```

Файл write2.cpp – діалогове вікно запису у буфер приймача (регістр RBR)

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

#include "write2.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm7 *Form7;
//---------------------------------------------------------------------
__fastcall TForm7::TForm7(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------

void __fastcall TForm7::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->RBR0->Text=1; else Form1->RBR0->Text=0;
if(CheckBox2->State==cbChecked) Form1->RBR1->Text=1; else Form1->RBR1->Text=0;
if(CheckBox3->State==cbChecked) Form1->RBR2->Text=1; else Form1->RBR2->Text=0;
if(CheckBox4->State==cbChecked) Form1->RBR3->Text=1; else Form1->RBR3->Text=0;
if(CheckBox5->State==cbChecked) Form1->RBR4->Text=1; else Form1->RBR4->Text=0;
if(CheckBox6->State==cbChecked) Form1->RBR5->Text=1; else Form1->RBR5->Text=0;
if(CheckBox7->State==cbChecked) Form1->RBR6->Text=1; else Form1->RBR6->Text=0;
if(CheckBox8->State==cbChecked) Form1->RBR7->Text=1; else Form1->RBR7->Text=0;
Form7->Close();
}
//---------------------------------------------------------------------
```

Файл write2.h – бібліотека для файлу write2.cpp

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

#ifndef write2H
#define write2H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm7 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private: // Визначається користувачем
public:     // Визначається користувачем
        __fastcall TForm7(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm7 *Form7;
//---------------------------------------------------------------------
#endif
```

Файл write3.cpp – діалогове вікно запису у молодший байт дільника (регістр LSB)

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

#include <vcl.h>
#pragma hdrstop

#include "write3.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm8 *Form8;
//---------------------------------------------------------------------
__fastcall TForm8::TForm8(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm8::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->LSB0->Text=1; else Form1->LSB0->Text=0;
if(CheckBox2->State==cbChecked) Form1->LSB1->Text=1; else Form1->LSB1->Text=0;
if(CheckBox3->State==cbChecked) Form1->LSB2->Text=1; else Form1->LSB2->Text=0;
if(CheckBox4->State==cbChecked) Form1->LSB3->Text=1; else Form1->LSB3->Text=0;
if(CheckBox5->State==cbChecked) Form1->LSB4->Text=1; else Form1->LSB4->Text=0;
if(CheckBox6->State==cbChecked) Form1->LSB5->Text=1; else Form1->LSB5->Text=0;
if(CheckBox7->State==cbChecked) Form1->LSB6->Text=1; else Form1->LSB6->Text=0;
if(CheckBox8->State==cbChecked) Form1->LSB7->Text=1; else Form1->LSB7->Text=0;
Form8->Close();
}
//---------------------------------------------------------------------
```

Файл write3.h – бібліотека для файлу write3.cpp

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

#ifndef write3H
#define write3H
//---------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------
class TForm8 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:    // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm8(TComponent* Owner);
};
//---------------------------------------------------------------
extern PACKAGE TForm8 *Form8;
//---------------------------------------------------------------
#endif
```

Файл write4.cpp – діалогове вікно запису у старший байт дільника (регістр MSB)

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

#include <vcl.h>
#pragma hdrstop

#include "write4.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm9 *Form9;
//---------------------------------------------------------------------
__fastcall TForm9::TForm9(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm9::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->MSB0->Text=1; else Form1->MSB0->Text=0;
if(CheckBox2->State==cbChecked) Form1->MSB1->Text=1; else Form1->MSB1->Text=0;
if(CheckBox3->State==cbChecked) Form1->MSB2->Text=1; else Form1->MSB2->Text=0;
if(CheckBox4->State==cbChecked) Form1->MSB3->Text=1; else Form1->MSB3->Text=0;
if(CheckBox5->State==cbChecked) Form1->MSB4->Text=1; else Form1->MSB4->Text=0;
if(CheckBox6->State==cbChecked) Form1->MSB5->Text=1; else Form1->MSB5->Text=0;
if(CheckBox7->State==cbChecked) Form1->MSB6->Text=1; else Form1->MSB6->Text=0;
if(CheckBox8->State==cbChecked) Form1->MSB7->Text=1; else Form1->MSB7->Text=0;
Form9->Close();

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

Файл write4.h – бібліотека для файлу write4.cpp

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

#ifndef write4H
#define write4H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm9 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private: // Визначається користувачем
public:     // Визначається користувачем
        __fastcall TForm9(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm9 *Form9;
//---------------------------------------------------------------------
#endif
```

Файл write5.cpp – діалогове вікно запису у регістр 3F9h (IER)

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

#include <vcl.h>
#pragma hdrstop

#include "write5.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm10 *Form10;
//---------------------------------------------------------------------
__fastcall TForm10::TForm10(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm10::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->IER0->Text=1; else Form1->IER0->Text=0;
if(CheckBox2->State==cbChecked) Form1->IER1->Text=1; else Form1->IER1->Text=0;
if(CheckBox3->State==cbChecked) Form1->IER2->Text=1; else Form1->IER2->Text=0;
if(CheckBox4->State==cbChecked) Form1->IER3->Text=1; else Form1->IER3->Text=0;
if(CheckBox5->State==cbChecked) Form1->IER4->Text=1; else Form1->IER4->Text=0;
if(CheckBox6->State==cbChecked) Form1->IER5->Text=1; else Form1->IER5->Text=0;
if(CheckBox7->State==cbChecked) Form1->IER6->Text=1; else Form1->IER6->Text=0;
if(CheckBox8->State==cbChecked) Form1->IER7->Text=1; else Form1->IER7->Text=0;
Form10->Close();
}
//---------------------------------------------------------------------
```

Файл write5.h – бібліотека для файлу write5.cpp

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

#ifndef write5H
#define write5H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm10 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:// Визначається користувачем
public:     // Визначається користувачем
        __fastcall TForm10(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm10 *Form10;
//---------------------------------------------------------------------
#endif
```

Файл write6.cpp – діалогове вікно запису у регістр 3FAh (IIR)

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

#include <vcl.h>
#pragma hdrstop

#include "write6.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm11 *Form11;
//---------------------------------------------------------------------
__fastcall TForm11::TForm11(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm11::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->IIR0->Text=1; else Form1->IIR0->Text=0;
if(CheckBox2->State==cbChecked) Form1->IIR1->Text=1; else Form1->IIR1->Text=0;
if(CheckBox3->State==cbChecked) Form1->IIR2->Text=1; else Form1->IIR2->Text=0;
if(CheckBox4->State==cbChecked) Form1->IIR3->Text=1; else Form1->IIR3->Text=0;
if(CheckBox5->State==cbChecked) Form1->IIR4->Text=1; else Form1->IIR4->Text=0;
if(CheckBox6->State==cbChecked) Form1->IIR5->Text=1; else Form1->IIR5->Text=0;
if(CheckBox7->State==cbChecked) Form1->IIR6->Text=1; else Form1->IIR6->Text=0;
if(CheckBox8->State==cbChecked) Form1->IIR7->Text=1; else Form1->IIR7->Text=0;
Form11->Close();

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

Файл write6.h – бібліотека для файлу write6.cpp

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

#ifndef write6H
#define write6H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm11 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:  // Визначається користувачем
public:     // Визначається користувачем
        __fastcall TForm11(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm11 *Form11;
//---------------------------------------------------------------------
#endif
```

Файл write7.cpp – діалогове вікно запису у регістр 3FBh (LCR)

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

#include <vcl.h>
#pragma hdrstop

#include "write7.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm12 *Form12;
//---------------------------------------------------------------------
__fastcall TForm12::TForm12(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm12::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->LCR0->Text=1; else Form1->LCR0->Text=0;
if(CheckBox2->State==cbChecked) Form1->LCR1->Text=1; else Form1->LCR1->Text=0;
if(CheckBox3->State==cbChecked) Form1->LCR2->Text=1; else Form1->LCR2->Text=0;
if(CheckBox4->State==cbChecked) Form1->LCR3->Text=1; else Form1->LCR3->Text=0;
if(CheckBox5->State==cbChecked) Form1->LCR4->Text=1; else Form1->LCR4->Text=0;
if(CheckBox6->State==cbChecked) Form1->LCR5->Text=1; else Form1->LCR5->Text=0;
if(CheckBox7->State==cbChecked) Form1->LCR6->Text=1; else Form1->LCR6->Text=0;
if(CheckBox8->State==cbChecked) Form1->LCR7->Text=1; else Form1->LCR7->Text=0;
Form12->Close();

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

Файл write7.h – бібліотека для файлу write7.cpp

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

#ifndef write7H
#define write7H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm12 : public TForm
{
__published: // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:     // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm12(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm12 *Form12;
//---------------------------------------------------------------------
#endif
```

Файл write8.cpp – діалогове вікно запису у регістр 3FCh (MCR)

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

#include <vcl.h>
#pragma hdrstop

#include "write8.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm13 *Form13;
//---------------------------------------------------------------------
__fastcall TForm13::TForm13(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm13::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->MCR0->Text=1; else Form1->MCR0->Text=0;
if(CheckBox2->State==cbChecked) Form1->MCR1->Text=1; else Form1->MCR1->Text=0;
if(CheckBox3->State==cbChecked) Form1->MCR2->Text=1; else Form1->MCR2->Text=0;
if(CheckBox4->State==cbChecked) Form1->MCR3->Text=1; else Form1->MCR3->Text=0;
if(CheckBox5->State==cbChecked) Form1->MCR4->Text=1; else Form1->MCR4->Text=0;
if(CheckBox6->State==cbChecked) Form1->MCR5->Text=1; else Form1->MCR5->Text=0;
if(CheckBox7->State==cbChecked) Form1->MCR6->Text=1; else Form1->MCR6->Text=0;
if(CheckBox8->State==cbChecked) Form1->MCR7->Text=1; else Form1->MCR7->Text=0;
Form13->Close();

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

Файл write8.h – бібліотека для файлу write8.cpp

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

#ifndef write8H
#define write8H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm13 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:    // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm13(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm13 *Form13;
//---------------------------------------------------------------------
#endif
```

Файл write9.cpp – діалогове вікно запису у регістр 3FDh (LSR)

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

#include "write9.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm14 *Form14;
//---------------------------------------------------------------------
__fastcall TForm14::TForm14(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm14::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->LSR0->Text=1; else Form1->LSR0->Text=0;
if(CheckBox2->State==cbChecked) Form1->LSR1->Text=1; else Form1->LSR1->Text=0;
if(CheckBox3->State==cbChecked) Form1->LSR2->Text=1; else Form1->LSR2->Text=0;
if(CheckBox4->State==cbChecked) Form1->LSR3->Text=1; else Form1->LSR3->Text=0;
if(CheckBox5->State==cbChecked) Form1->LSR4->Text=1; else Form1->LSR4->Text=0;
if(CheckBox6->State==cbChecked) Form1->LSR5->Text=1; else Form1->LSR5->Text=0;
if(CheckBox7->State==cbChecked) Form1->LSR6->Text=1; else Form1->LSR6->Text=0;
if(CheckBox8->State==cbChecked) Form1->LSR7->Text=1; else Form1->LSR7->Text=0;
Form14->Close();

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

Файл write9.h – бібліотека для файлу write9.cpp

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

#ifndef write9H
#define write9H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm14 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:   // Визначається користувачем
public:     // Визначається користувачем
        __fastcall TForm14(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm14 *Form14;
//---------------------------------------------------------------------
#endif
```

Файл write110.cpp – діалогове вікно запису у регістр 3FEh (MSR)

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

#include "write10.h"
#include "main.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm15 *Form15;
//---------------------------------------------------------------------
__fastcall TForm15::TForm15(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm15::Button1Click(TObject *Sender)
{
if(CheckBox1->State==cbChecked) Form1->MSR0->Text=1; else Form1->MSR0->Text=0;
if(CheckBox2->State==cbChecked) Form1->MSR11->Text=1; else Form1->MSR11->Text=0;
if(CheckBox3->State==cbChecked) Form1->MSR2->Text=1; else Form1->MSR2->Text=0;
if(CheckBox4->State==cbChecked) Form1->MSR3->Text=1; else Form1->MSR3->Text=0;
if(CheckBox5->State==cbChecked) Form1->MSR4->Text=1; else Form1->MSR4->Text=0;
if(CheckBox6->State==cbChecked) Form1->MSR5->Text=1; else Form1->MSR5->Text=0;
if(CheckBox7->State==cbChecked) Form1->MSR6->Text=1; else Form1->MSR6->Text=0;
if(CheckBox8->State==cbChecked) Form1->MSR7->Text=1; else Form1->MSR7->Text=0;
Form15->Close();

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

Файл write10.h – бібліотека для файлу write10.cpp

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

#ifndef write10H
#define write10H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm15 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TLabel *Label8;
        TCheckBox *CheckBox2;
        TCheckBox *CheckBox1;
        TCheckBox *CheckBox3;
        TCheckBox *CheckBox4;
        TCheckBox *CheckBox5;
        TCheckBox *CheckBox6;
        TCheckBox *CheckBox7;
        TCheckBox *CheckBox8;
        TLabel *Label9;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:    // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm15(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm15 *Form15;
//---------------------------------------------------------------------
#endif
```

Файл help.cpp – довідка про регістри

послідовного порту

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

#include <vcl.h>
#pragma hdrstop

#include "help.h"
//---------------------------------------------------------------------#pragma
package(smart_init)
#pragma resource "*.dfm"
TForm4 *Form4;
int reg=1;
//---------------------------------------------------------------------
__fastcall TForm4::TForm4(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm4::Button1Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр буфера передавача (THR). Має адресу 0 щодо базової
адреси контролера. Даний регістр доступний тільки під час запису й при значенні
біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR),
рівному 0. Регістр THR містить вісім бітів даних (біт 0 є молодшим значущим
розрядом і посилається першим у канал передачі).");
Button1->Font->Size=14;
Button2->Font->Size=8;
Button3->Font->Size=8;
Button4->Font->Size=8;
Button5->Font->Size=8;
Button6->Font->Size=8;
Button7->Font->Size=8;
Button8->Font->Size=8;
Button9->Font->Size=8;
Button10->Font->Size=8;
reg=1;
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button2Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр буфера приймача (RBR). Має адресу 0 щодо базової
адреси контролера. Цей регістр доступний під час читання (IN) і при значенні
біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR),
рівному 0. Регістр RBR містить вісім бітів даних (біт 0 є молодшим значущим
розрядом і приймається першим з каналу передачі).");

Button1->Font->Size=8;
Button2->Font->Size=14;
Button3->Font->Size=8;
Button4->Font->Size=8;
Button5->Font->Size=8;
Button6->Font->Size=8;
Button7->Font->Size=8;
Button8->Font->Size=8;
Button9->Font->Size=8;
Button10->Font->Size=8;
reg=2;
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button3Click(TObject *Sender)
{
```

```
Memo1->Clear();
Memo1->Lines->Add("Регістр буфера молодшого байта дільника (Divisor Latch LSB). Регістр має адресу 0 щодо базової адреси контролера. Цей регістр доступний і під час читання, і під час запису тільки при значенні біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR), рівному 1. При записі в цей регістр нового значення дільник перезавантажується негайно.");
Memo1->Lines->Add("Дільник     Швидкість передачі в бодах");
Memo1->Lines->Add("1040        110");
Memo1->Lines->Add("768          150");
Memo1->Lines->Add("384          300 ");
Memo1->Lines->Add("192          600 ");
Memo1->Lines->Add("96           1200 ");
Memo1->Lines->Add("48           2400 ");
Memo1->Lines->Add("24           4800 ");
Memo1->Lines->Add("12           9600  ");
Memo1->Lines->Add("6            19200 ");
Memo1->Lines->Add("3            38400");
Memo1->Lines->Add("2            57600  ");
Memo1->Lines->Add("1            115200  ");
Button1->Font->Size=8;
Button2->Font->Size=8;
Button3->Font->Size=14;
Button4->Font->Size=8;
Button5->Font->Size=8;
Button6->Font->Size=8;
Button7->Font->Size=8;
Button8->Font->Size=8;
Button9->Font->Size=8;
Button10->Font->Size=8;
reg=3;
}
//---------------------------------------------------------------
void __fastcall TForm4::Button4Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр буфера старшого байта дільника (Divisor Latch MSB). Регістр має адресу 1 щодо базової адреси контролера. Цей регістр доступний по читанню й запису тільки при значенні біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR), рівному 1. При записі в цей регістр нового значення дільник відразу перезавантажується.");
Button1->Font->Size=8;
Button2->Font->Size=8;
Button3->Font->Size=8;
Button4->Font->Size=14;
Button5->Font->Size=8;
Button6->Font->Size=8;
Button7->Font->Size=8;
Button8->Font->Size=8;
Button9->Font->Size=8;
Button10->Font->Size=8;
reg=4;
}
//---------------------------------------------------------------
void __fastcall TForm4::Button5Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр дозволу переривань (IER). Має адресу 1 щодо базової адреси контролера. Цей регістр доступний по читанню й запису, але тільки при значенні біта дозволу доступу до дільника (DLAB) у регістрі керування лінією (LCR), рівному 0. Цей регістр дозволяє управляти чотирма типами переривань, породжуваними контролером послідовного інтерфейсу. Формат регістра:");
Memo1->Lines->Add("Біт 0 - IDA ");
Memo1->Lines->Add("Біт 1 - IFB ");
Memo1->Lines->Add("Біт 2 - ICL ");
Memo1->Lines->Add("Біт 3 - ICM ");
Memo1->Lines->Add("Біт 4 - зарезервований, =0");
Memo1->Lines->Add("Біт 5 - зарезервований, =0");
Memo1->Lines->Add("Біт 6 - зарезервований, =0");
Memo1->Lines->Add("Біт 7 - зарезервований, =0");
```

```
Button1->Font->Size=8;
Button2->Font->Size=8;
Button3->Font->Size=8;
Button4->Font->Size=8;
Button5->Font->Size=14;
Button6->Font->Size=8;
Button7->Font->Size=8;
Button8->Font->Size=8;
Button9->Font->Size=8;
Button10->Font->Size=8;
reg=5;
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button6Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр ідентифікації переривання (IIR). Регістр має адресу 2
щодо базової адреси контролера. Цей регістр доступний тільки по читанню й
дозволяє одержати інформацію від контролера про очікуюче переривання. Значення
бітів регістра:");
Memo1->Lines->Add("Біт 0 - II ");
Memo1->Lines->Add("Біт 1 - I Type ");
Memo1->Lines->Add("Біт 2 - I Type ");
Memo1->Lines->Add("Біт 3 - зарезервований, =0 ");
Memo1->Lines->Add("Біт 4 - зарезервований, =0");
Memo1->Lines->Add("Біт 5 - зарезервований, =0");
Memo1->Lines->Add("Біт 6 - зарезервований, =0");
Memo1->Lines->Add("Біт 7 - зарезервований, =0");
Button1->Font->Size=8;
Button2->Font->Size=8;
Button3->Font->Size=8;
Button4->Font->Size=8;
Button5->Font->Size=8;
Button6->Font->Size=14;
Button7->Font->Size=8;
Button8->Font->Size=8;
Button9->Font->Size=8;
Button10->Font->Size=8;
reg=6;
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button7Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр керування лінією (LCR). Регістр має адресу 3 щодо
базової адреси контролера. Цей регістр доступний і під час читання, і під час
запису.Значення даного регістра визначає формат переданих даних у лінію передачі
даних контролером послідовного інтерфейсу. Значення бітів регістра");
Memo1->Lines->Add("Біт 0 - WLS ");
Memo1->Lines->Add("Біт 1 - WLS ");
Memo1->Lines->Add("Біт 2 - NSB ");
Memo1->Lines->Add("Біт 3 - PA ");
Memo1->Lines->Add("Біт 4 - EPS ");
Memo1->Lines->Add("Біт 5 - SP ");
Memo1->Lines->Add("Біт 6 - SB ");
Memo1->Lines->Add("Біт 7 - DLAB ");
Button1->Font->Size=8;
Button2->Font->Size=8;
Button3->Font->Size=8;
Button4->Font->Size=8;
Button5->Font->Size=8;
Button6->Font->Size=8;
Button7->Font->Size=14;
Button8->Font->Size=8;
Button9->Font->Size=8;
Button10->Font->Size=8;
reg=7;
```

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

void __fastcall TForm4::Button8Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр керування модемом (MCR). Регістр керування модемом
має адресу 4 щодо базової адреси контролера. Цей регістр доступний по читанню й
запису. За допомогою регістра можна управляти роботою модему. Формат
регістра:");
Memo1->Lines->Add("Біт 0 - DTR ");
Memo1->Lines->Add("Біт 1 - RTS ");
Memo1->Lines->Add("Біт 2 - Out1 ");
Memo1->Lines->Add("Біт 3 - Out2 ");
Memo1->Lines->Add("Біт 4 - LB ");
Memo1->Lines->Add("Біт 5 - зарезервований, =0 ");
Memo1->Lines->Add("Біт 6 - зарезервований, =0 ");
Memo1->Lines->Add("Біт 7 - зарезервований, =0 ");
Button1->Font->Size=8;
Button2->Font->Size=8;
Button3->Font->Size=8;
Button4->Font->Size=8;
Button5->Font->Size=8;
Button6->Font->Size=8;
Button7->Font->Size=8;
Button8->Font->Size=14;
Button9->Font->Size=8;
Button10->Font->Size=8;
reg=8;
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button9Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр стану лінії (LSR). Регістр стану лінії має адресу 5
щодо базової адреси контролера й доступний тільки по читанню. Регістр LSR надає
інформацію про стан обміну даних. Формат регістра:");
Memo1->Lines->Add("Біт 0 - DR ");
Memo1->Lines->Add("Біт 1 - OR ");
Memo1->Lines->Add("Біт 2 - PE ");
Memo1->Lines->Add("Біт 3 - FE ");
Memo1->Lines->Add("Біт 4 - BI ");
Memo1->Lines->Add("Біт 5 - THRE ");
Memo1->Lines->Add("Біт 6 - TEMT ");
Memo1->Lines->Add("Біт 7 - зарезервований, =0 ");
Button1->Font->Size=8;
Button2->Font->Size=8;
Button3->Font->Size=8;
Button4->Font->Size=8;
Button5->Font->Size=8;
Button6->Font->Size=8;
Button7->Font->Size=8;
Button8->Font->Size=8;
Button9->Font->Size=14;
Button10->Font->Size=8;
reg=9;
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button10Click(TObject *Sender)
{
Memo1->Clear();
Memo1->Lines->Add("Регістр стану модему (MSR). Регістр має адресу 6 щодо базової
адреси контролера й доступний тільки по читанню. Регістр надає інформацію про
стан керуючих ліній модему. Крім того, цей регістр містить 4 біти, які
відображають зміну стану модему й встановлюються в значення 0 після операції
читання з регістра MSR.");
Memo1->Lines->Add("Біт 0 - DCTS ");
```

```
Memo1->Lines->Add("Біт 1 - DDSR ");
Memo1->Lines->Add("Біт 2 - TERI ");
Memo1->Lines->Add("Біт 3 - DDCD ");
Memo1->Lines->Add("Біт 4 - CTS ");
Memo1->Lines->Add("Біт 5 - DSR ");
Memo1->Lines->Add("Біт 6 - RI ");
Memo1->Lines->Add("Біт 7 - DCD ");
Button1->Font->Size=8;
Button2->Font->Size=8;
Button3->Font->Size=8;
Button4->Font->Size=8;
Button5->Font->Size=8;
Button6->Font->Size=8;
Button7->Font->Size=8;
Button8->Font->Size=8;
Button9->Font->Size=8;
Button10->Font->Size=14;
reg=10;
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button11Click(TObject *Sender)
{

Button11->Font->Size=14;
Button12->Font->Size=8;
Button13->Font->Size=8;
Button14->Font->Size=8;
Button15->Font->Size=8;
Button16->Font->Size=8;
Button17->Font->Size=8;
Button18->Font->Size=8;
if(reg==1){
Memo2->Clear();
Memo2->Lines->Add("біт даних для передачі");
}
else if(reg==2){
Memo2->Clear();
Memo2->Lines->Add("прийнятий біт даних");
}
else if(reg==3){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==4){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==5){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==6){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==7){
Memo2->Clear();
Memo2->Lines->Add("DLAB управляє доступом до регістрів буфера дільника. Якщо біт
дорівнює 1, операція читання й запису по адресі  1 щодо базової адреси
виконуються з регістрами буфера дільника програмувального генератора. Для
доступу до регістрів RBR, THR і IER біт повинен мати нульове значення.");
}
else if(reg==8){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==9){
Memo2->Clear();
```

```
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==10){
Memo2->Clear();
Memo2->Lines->Add("DCD є інвертованим сигналом Data Carrier Detect (DCD). При
встановленому режимі 'шлейфа' (біт LB регістра MCR має значення 1) цей біт
еквівалентний біту Out2 регістри MCR.");
}

}
//---------------------------------------------------------------------
void __fastcall TForm4::Button12Click(TObject *Sender)
{
Button11->Font->Size=8;
Button12->Font->Size=14;
Button13->Font->Size=8;
Button14->Font->Size=8;
Button15->Font->Size=8;
Button16->Font->Size=8;
Button17->Font->Size=8;
Button18->Font->Size=8;
if(reg==1){
Memo2->Clear();
Memo2->Lines->Add("біт даних для передачі");
}
else if(reg==2){
Memo2->Clear();
Memo2->Lines->Add("прийнятий біт даних");
}
else if(reg==3){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==4){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==5){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==6){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==7){
Memo2->Clear();
Memo2->Lines->Add(" SB встановлює стан 'пауза', коли дорівнює 1. У цьому стані
на виході контролера послідовного інтерфейсу встановлюється значення 0, що не
може бути змінене ніякими іншими діями, крім як переустановкою біта в 0.");
}
else if(reg==8){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==9){
Memo2->Clear();
Memo2->Lines->Add("TEMT являється індикатором звільнення передавача. Встановка
цього біта в 1 означає, що як регістр THR, так і регістр TSR вільний. Цей біт
встановлюється в значення 0, якщо кожний з регістрів THR і TSR містить
символ.");
}
else if(reg==10){
Memo2->Clear();
Memo2->Lines->Add("RI є інвертованим сигналом Ring Indicator (RI). При
встановленому режимі 'шлейфа' (біт LB регістра MCR має значення 1) еквівалентний
біту Out1 регістра MCR.");
}
}
```

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

void __fastcall TForm4::Button13Click(TObject *Sender)
{
Button11->Font->Size=8;
Button12->Font->Size=8;
Button13->Font->Size=14;
Button14->Font->Size=8;
Button15->Font->Size=8;
Button16->Font->Size=8;
Button17->Font->Size=8;
Button18->Font->Size=8;
if(reg==1){
Memo2->Clear();
Memo2->Lines->Add("біт даних для передачі");
}
else if(reg==2){
Memo2->Clear();
Memo2->Lines->Add("прийнятий біт даних");
}
else if(reg==3){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==4){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==5){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==6){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==7){
Memo2->Clear();
Memo2->Lines->Add(" SP управляє установкою режиму незмінного біта контролю
парності. Значення біта 1 задає режим, а значення 0 - скасовує. При встановленні
біта SP в 1 повинен встановлюватися в 1 і біт PA, так як ці два біти зв'язані.
Коли значення біта EPS дорівнює 0, посилається й контролюється значення біта
контролю парності, рівне 1 (Mark Parity). При одиничному значенні біта EPS
посилається й контролюється значення біта контролю парності, рівне 0 (Space
Parity).");
}
else if(reg==8){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==9){
Memo2->Clear();
Memo2->Lines->Add("THRE є індикатором звільнення регістра THR. Установка цього
біта в 1 означає, що з регістра THR символ переданий у зсувовий регістр
передавача (TSR) і регістра THR готовий прийняти наступний байт. Якщо в регістрі
IER дозволене переривання по звільненню регістра THR, то при установці цього
біта в значенні 1 відбувається також переривання по звільненню регістра THR.");
}
else if(reg==10){
Memo2->Clear();
Memo2->Lines->Add("DSR є інвертованим сигналом Data Set Ready (DSR). У режимі
'шлейфа' (біт LB регістра MCR має значення 1) еквівалентний біту DTR регістра
MCR.");
}
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button14Click(TObject *Sender)
{
```

```
Button11->Font->Size=8;
Button12->Font->Size=8;
Button13->Font->Size=8;
Button14->Font->Size=14;
Button15->Font->Size=8;
Button16->Font->Size=8;
Button17->Font->Size=8;
Button18->Font->Size=8;
if(reg==1){
Memo2->Clear();
Memo2->Lines->Add("біт даних для передачі");
}
else if(reg==2){
Memo2->Clear();
Memo2->Lines->Add("прийнятий біт даних");
}
else if(reg==3){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==4){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==5){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==6){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==7){
Memo2->Clear();
Memo2->Lines->Add("EPS задає вибір режиму контролю парності. Якщо біт
встановлений в 0 і біт PA установлений в 1, генерується й перевіряється парна
кількість одиничних бітів символу посилки й біта контролю парності. Якщо біт
установлений в 1 і біт PA встановлений в 1, генерується й перевіряється непарна
кількість одиничних бітів символу посилки й біта контролю парності.");
}
else if(reg==8){
Memo2->Clear();
Memo2->Lines->Add("LB задає режим 'шлейфа' (Loopback) для діагностичних
цілей.");
}
else if(reg==9){
Memo2->Clear();
Memo2->Lines->Add("BI є індикатором стану 'пауза' (Break Interrupt). Стан
'пауза' фіксується в тому випадку, якщо  рівень прийнятого сигналу встановлений
в 0 на час прийому повної посилки, тобто загальний час стартового біта, бітів
даних, біта контролю парності й стоп-біта. Цей біт приймає значення 0 після
операції читання регістра LSR. Біти 4-1 є індикаторами помилки й установка
кожного із цих бітів у значення 1 проводить до породження переривання по стану
ліній приймача.");
}
else if(reg==10){
Memo2->Clear();
Memo2->Lines->Add("CTS - інвертований сигнал Clear to Send (CTS). При
встановленому режимі 'шлейфа' (біт LB регістра MCR має значення 1) цей біт
еквівалентний біту RTS регістра MCR. Біти  DDCD, TERI, DDSR і DCTS є
індикаторами зміни стану модему й установка кожного із цих бітів у значення 1
приводить до породження переривання по стану модему, якщо воно дозволено в
регістрі IER.");
}
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button15Click(TObject *Sender)
{
```

```
Button11->Font->Size=8;
Button12->Font->Size=8;
Button13->Font->Size=8;
Button14->Font->Size=8;
Button15->Font->Size=14;
Button16->Font->Size=8;
Button17->Font->Size=8;
Button18->Font->Size=8;
if(reg==1){
Memo2->Clear();
Memo2->Lines->Add("біт даних для передачі");
}
else if(reg==2){
Memo2->Clear();
Memo2->Lines->Add("прийнятий біт даних");
}
else if(reg==3){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==4){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==5){
Memo2->Clear();
Memo2->Lines->Add("ICM  задає переривання при зміні стану модема:");
Memo2->Lines->Add("1 - переривання виробляється;");
Memo2->Lines->Add("0 - переривання заборонене.");
}
else if(reg==6){
Memo2->Clear();
Memo2->Lines->Add("Зарезервований, =0");
}
else if(reg==7){
Memo2->Clear();
Memo2->Lines->Add("PA є бітом дозволу контролю парності. Якщо біт установлений в
1, то генерується біт контролю парності між останнім бітом переданого символу й
стоп-бітом.");
}
else if(reg==8){
Memo2->Clear();
Memo2->Lines->Add("Out2 управляє сигналом Out2. При одиничному значенні біта
сигнал Out2 встановлюється рівним 1. Сигнал Out2  управляє генерацією переривань
контролера послідовного інтерфейсу. При одиничному знанні сигнал контролер
генерує переривання у відповідності  зі значенням регістра IER. При нульовому
значенні сигналу Out2 контролер не генерує переривань незалежно від значення
регістра IER.");
}
else if(reg==9){
Memo2->Clear();
Memo2->Lines->Add("FE є індикатором 'помилки стоп-бітів' (Framing Error).
Помилка стоп-бита фіксується в тому випадку, коли в прийнятому символі не
виявлено коректного стоп-біта, тобто біт, що слідує за останнім бітом даних або
за бітом контролю парності (у випадку контролю парності), має значення 0. Цей
біт приймає значення 0 після операції читання регістра LSR.");
}
else if(reg==10){
Memo2->Clear();
Memo2->Lines->Add("DDCD є індикатором зміни сигналу Data Carrier Detect (DCD).
Цей біт приймає значення 1 при зміні сигналу DCD після останньої операції
читання регістра MSR.");
}
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button16Click(TObject *Sender)
{
Button11->Font->Size=8;
```

```
Button12->Font->Size=8;
Button13->Font->Size=8;
Button14->Font->Size=8;
Button15->Font->Size=8;
Button16->Font->Size=14;
Button17->Font->Size=8;
Button18->Font->Size=8;
if(reg==1){
Memo2->Clear();
Memo2->Lines->Add("біт даних для передачі");
}
else if(reg==2){
Memo2->Clear();
Memo2->Lines->Add("прийнятий біт даних");
}
else if(reg==3){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==4){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==5){
Memo2->Clear();
Memo2->Lines->Add("ICL визначає переривання при зміні стану лінії приймача:");
Memo2->Lines->Add("1 - переривання виробляється;");
Memo2->Lines->Add("0 - переривання заборонене.");

}
else if(reg==6){
Memo2->Clear();
Memo2->Lines->Add("1 біт I Type. Біти  I Type визначають тип очікуючого
переривання, якщо воно зберігається контролером (що визначається бітом II):");
Memo2->Lines->Add("11 - змінився стан лінії приймача;  ");
Memo2->Lines->Add("10 - прийняті дані доступні; ");
Memo2->Lines->Add("01 - звільнений регістр буфера;");
Memo2->Lines->Add("00 - змінився стан модему. ");

}
else if(reg==7){
Memo2->Clear();
Memo2->Lines->Add("NSB визначає кількість стоп-бітів у кожному символі,
переданому контролером послідовного інтерфейсу, і пов'язаний з довжиною слова
обміну (біти WLS). Якщо цей біт встановлений в 0, то генерується й перевіряється
один стоп-біт при будь-якій довжині слова обміну. Якщо цей біт встановлений в 1,
то при довжині слова обміну в 5 біт генерується й перевіряється 1.5 стоп-біта, а
при будь-якій іншій довжині слова обміну генерується й перевіряється 2 стоп-
біта. При асинхронній передачі поняття біта нерозривно зв'язане з тривалістю
сигналу, тому цілком можлива посилка нецілого числа стоп-бітів. Це може
знадобитися, якщо підключений до комп'ютера пристрій не програмується, а суворо
налагоджений на аналіз стоп-бітів заданої довжини.");
}
else if(reg==8){
Memo2->Clear();
Memo2->Lines->Add("Out1 управляє сигналом Out1. Якщо біт встановлений в 1,
сигнал Out1  встановлюється 1. При завданні значення 0 сигнал встановлюється в
нульовий рівень.");
}
else if(reg==9){
Memo2->Clear();
Memo2->Lines->Add("PE є індикатором 'помилки парності' (Parity Error). Помилка
парності фіксується, якщо в прийнятому символі виявлене некоректне значення біта
контролю парності. Цей біт приймає значення 0 після читання регістра LSR.");
}
else if(reg==10){
Memo2->Clear();
```

```
Memo2->Lines->Add("TERI є індикатором заднього фронту сигналу RI. Цей біт
приймає значення 1 при зміні сигналу RI з рівня логічної 1 на рівень логічного
нуля.");
}
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button17Click(TObject *Sender)
{
Button11->Font->Size=8;
Button12->Font->Size=8;
Button13->Font->Size=8;
Button14->Font->Size=8;
Button15->Font->Size=8;
Button16->Font->Size=8;
Button17->Font->Size=14;
Button18->Font->Size=8;
if(reg==1){
Memo2->Clear();
Memo2->Lines->Add("біт даних для передачі");
}
else if(reg==2){
Memo2->Clear();
Memo2->Lines->Add("прийнятий біт даних");
}
else if(reg==3){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==4){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==5){
Memo2->Clear();
Memo2->Lines->Add("IFB  задає переривання при звільненні регістра буфера
прийнятих даних:");
Memo2->Lines->Add("1 - переривання виробляється;");
Memo2->Lines->Add("0 - переривання заборонене.");

}
else if(reg==6){
Memo2->Clear();
Memo2->Clear();
Memo2->Lines->Add("2 біт I Type. Біти  I Type визначають тип очікуючого
переривання, якщо воно зберігається контролером (що визначається бітом II):");
Memo2->Lines->Add("11 - змінився стан лінії приймача;  ");
Memo2->Lines->Add("10 - прийняті дані доступні; ");
Memo2->Lines->Add("01 - звільнений регістр буфера;");
Memo2->Lines->Add("00 - змінився стан модему. ");
}
else if(reg==7){
Memo2->Clear();
Memo2->Lines->Add("1 біт WLS. Біти WLS визначають довжину слова обміну:");
Memo2->Lines->Add("00 - 5 бітів;  ");
Memo2->Lines->Add("01 - 6 бітів; ");
Memo2->Lines->Add("10 - 7 бітів; ");
Memo2->Lines->Add("11 - 8 бітів.");

}
else if(reg==8){
Memo2->Clear();
Memo2->Lines->Add("RTS управляє сигналом 'запит на передачу' (Request to Send).
При значенні цього біта, рівному 1, сигнал 'запит на передачу' встановлюється
рівним 1. При завданні значення 0 сигнал встановлюється в нульовий рівень.");
}
else if(reg==9){
Memo2->Clear();
```

```
Memo2->Lines->Add("OR є індикатором 'помилки переповнення' (Overrun Error).
Помилка переповнення фіксується в тому випадку, якщо при записі чергового
символу в регістр RBR виявлено, що попередній вміст цього регістра не зчитане й,
таким чином, воно загублено. Цей біт приймає значення 0 після операції читання
регістра LSR.");
}
else if(reg==10){
Memo2->Clear();
Memo2->Lines->Add("DDSR  є індикатором зміни сигналу Data Set Ready (DSR). Цей
біт приймає значення 1 при зміні сигналу DSR після останньої операції читання
регістра MSR.");
}
}
//---------------------------------------------------------------------

void __fastcall TForm4::Button18Click(TObject *Sender)
{
Button11->Font->Size=8;
Button12->Font->Size=8;
Button13->Font->Size=8;
Button14->Font->Size=8;
Button15->Font->Size=8;
Button16->Font->Size=8;
Button17->Font->Size=8;
Button18->Font->Size=14;
if(reg==1){
Memo2->Clear();
Memo2->Lines->Add("біт даних для передачі");
}
else if(reg==2){
Memo2->Clear();
Memo2->Lines->Add("прийнятий біт даних");
}
else if(reg==3){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==4){
Memo2->Clear();
Memo2->Lines->Add("один з бітів дільника");
}
else if(reg==5){
Memo2->Clear();
Memo2->Lines->Add("IDA  визначає переривання при доступності прийнятих даних:");
Memo2->Lines->Add("1 - переривання виробляється;");
Memo2->Lines->Add("0 - переривання заборонене.");
}
else if(reg==6){
Memo2->Clear();
Memo2->Lines->Add("Біт II є індикатором очікуючого переривання:");
Memo2->Lines->Add("0 - контролер послідовного інтерфейсу зберігає
переривання;");
Memo2->Lines->Add("1 - немає переривань, що очікують обробки.");

}
else if(reg==7){
Memo2->Clear();
Memo2->Lines->Add("2 біт WLS. Біти WLS визначають довжину слова обміну:");
Memo2->Lines->Add("00 - 5 бітів;  ");
Memo2->Lines->Add("01 - 6 бітів; ");
Memo2->Lines->Add("10 - 7 бітів; ");
Memo2->Lines->Add("11 - 8 бітів.");
}
else if(reg==8){
Memo2->Clear();
Memo2->Lines->Add("DTR задає рівень сигналу 'готовність термінала' (Data
Terminal Ready).  Якщо біт встановлений в 1, сигнал 'готовність термінала'
встановлюється рівним 1. При задані значення 0 сигнал встановлюватися в нульовий
рівень.");
```

```
}
else if(reg==9){
Memo2->Clear();
Memo2->Lines->Add("DR індикатор доступності прийнятих даних. Цей біт завжди
встановлюється в 1, коли приймачем повністю прийняті символ і поміщений у
регістр RBR. Біт приймає значення 0 після операції читання з регістра RBR.");
}
else if(reg==10){
Memo2->Clear();
Memo2->Lines->Add("DTCS є індикатором зміни сигналу Clear to Send (CTS). Цей біт
приймає значення 1 при зміні сигналу CTS після останньої операції читання
регістра MSR.");
}
}
//---------------------------------------------------------------------
```

Файл help.h – бібліотека для файлу help.cpp

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

#ifndef helpH
#define helpH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm4 : public TForm
{
__published: // Компоненти IDE-управління
        TLabel *Label1;
        TMemo *Memo1;
        TButton *Button1;
        TButton *Button2;
        TButton *Button3;
        TButton *Button4;
        TButton *Button5;
        TButton *Button6;
        TButton *Button7;
        TButton *Button8;
        TButton *Button9;
        TButton *Button10;
        TLabel *Label2;
        TButton *Button11;
        TButton *Button12;
        TButton *Button13;
        TButton *Button14;
        TButton *Button15;
        TButton *Button16;
        TButton *Button17;
        TButton *Button18;
        TMemo *Memo2;
        void __fastcall Button1Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall Button4Click(TObject *Sender);
        void __fastcall Button5Click(TObject *Sender);
        void __fastcall Button6Click(TObject *Sender);
        void __fastcall Button7Click(TObject *Sender);
        void __fastcall Button8Click(TObject *Sender);
        void __fastcall Button9Click(TObject *Sender);
        void __fastcall Button10Click(TObject *Sender);
        void __fastcall Button11Click(TObject *Sender);
        void __fastcall Button12Click(TObject *Sender);
        void __fastcall Button13Click(TObject *Sender);
        void __fastcall Button14Click(TObject *Sender);
        void __fastcall Button15Click(TObject *Sender);
        void __fastcall Button16Click(TObject *Sender);
        void __fastcall Button17Click(TObject *Sender);
        void __fastcall Button18Click(TObject *Sender);
private:    // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm4(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm4 *Form4;
//---------------------------------------------------------------------
#endif
```

Файл help2.cpp – довідка про контакти послідовного порту

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

#include <vcl.h>
#pragma hdrstop

#include "help2.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm5 *Form5;
//---------------------------------------------------------------------
__fastcall TForm5::TForm5(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm5::Button1Click(TObject *Sender)
{
Form5->Close();
}
//---------------------------------------------------------------------
```

Файл help2.h – бібліотека для файлу help2.cpp

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

#ifndef help2H
#define help2H
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
//---------------------------------------------------------------------
class TForm5 : public TForm
{
__published:      // Компоненти IDE-управління
        TMemo *Memo1;
        TButton *Button1;
        void __fastcall Button1Click(TObject *Sender);
private:    // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm5(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm5 *Form5;
//---------------------------------------------------------------------
#endif
```

Файл about.cpp – Вікно "Про програму..."

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

#include <vcl.h>
#pragma hdrstop

#include "about.h"
//---------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm3 *Form3;
//---------------------------------------------------------------------
__fastcall TForm3::TForm3(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------
void __fastcall TForm3::Button1Click(TObject *Sender)
{
Form3->Close();
}
//---------------------------------------------------------------------
```

Файл about.h – бібліотека для файлу about.cpp

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

#ifndef aboutH
#define aboutH
//---------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <jpeg.hpp>
//---------------------------------------------------------------------
class TForm3 : public TForm
{
__published:      // Компоненти IDE-управління
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TLabel *Label4;
        TLabel *Label5;
        TLabel *Label6;
        TLabel *Label7;
        TButton *Button1;
        TLabel *Label8;
        TImage *Image1;
        void __fastcall Button1Click(TObject *Sender);
private:    // Визначається користувачем
public:           // Визначається користувачем
        __fastcall TForm3(TComponent* Owner);
};
//---------------------------------------------------------------------
extern PACKAGE TForm3 *Form3;
//---------------------------------------------------------------------
#endif
```