МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"

Я. С. Парамуд, А. М. Миц

ЛАБОРАТОРНИЙ ПРАКТИКУМ
з дисципліни
“ПЕРИФЕРІЙНІ ПРИСТРОЇ, ІНТЕРФЕЙСИ ТА ДРАЙВЕРИ”

Навчальний посібник

Рекомендувала Науково-методична рада
Національного університету “Львівська політехніка”

Львів
Видавництво Львівської політехніки
2021

УДК 004
П 18

Рецензенти:

Рак Т. Є., доктор технічних наук, доцент, професор кафедри інформаційних технологій, проректор "ІТ СТЕП Університет";

Кремінь В. Т., кандидат технічних наук, доцент, Senior Principal Engineer, фірма "Infineon Technologies";

Влах-Вигриновська Г. І., кандидат технічних наук, доцент, доцент кафедри комп'ютеризованих систем автоматики Національного університету "Львівська політехніка"

Рекомендувала Науково-методична рада
Національного університету "Львівська політехніка"
як навчальний посібник
для студентів спеціальності 123 "Комп'ютерна інженерія"
(протокол № 59 від 20.10.2021 р.)

Парамуд Я. С.

П 18 Лабораторний практикум з дисципліни "Периферійні пристрої, інтерфейси та драйвери" : навч. посібник / Я. С. Парамуд, А. М. Миц. – Львів: Видавництво Львівської політехніки, 2021. – 100 с. – Режим доступу: http://eom.lp.edu.ua/textbooks/npppid.pdf вільний. – Заголовок з екрана.

ISBN 978-966-941-694-0

Лабораторний практикум містить комплекс із методичних матеріалів до виконання восьми лабораторних робіт із дисципліни "Периферійні пристрої, інтерфейси та драйвери".

Навчальний посібник призначений для студентів спеціальності "Комп'ютерна інженерія" галузі знань "Інформаційні технології".

УДК 004

ISBN 978-966-941-694-0

ЗМІСТ

ВСТУП

Лабораторний практикум з навчальної дисципліни “Периферійні пристрої, інтерфейси та драйвери” призначений для студентів третього курсу освітньо-кваліфікаційного рівня “Бакалавр” галузі знань 12 “Інформаційні технології” спеціальності 123 “Комп’ютерна інженерія” спеціалізації “Комп’ютерні системи та мережі”. Лабораторний практикум містить методичні матеріали для восьми лабораторних робіт. Лабораторні роботи орієнтовані на поглиблене вивчення особливостей, характеристик найбільш поширених периферійних інтерфейсів та отримання навиків розроблення для цих інтерфейсів програмних драйверів, які є складовими компонентами багатьох комп’ютерних систем. Матеріали лабораторного практикуму дозволять студентам економніше використати час на якісне виконання комплексу лабораторних робіт.

ТЕОРЕТИЧНІ ВІДОМОСТІ ДО ЛАБОРАТОРНИХ РОБІТ № 1–4

Лабораторні роботи № 1–4 призначені для поглибленого вивчення особливостей, характеристик інтерфейса RS-232C, одного із найбільш поширених периферійних інтерфейсів, та отримання навиків розроблення для цього інтерфейса програмних драйверів, які є складовими компонентами багатьох комп'ютерних систем.

Інтерфейс RS-232C за класивікацією послідовний, радіальний, дуплексний асинхронний інтерфейс. Він реалізовує стандарт передачі даних RS-232C (Reference Standard number 232 version C – стандарт обміну номер 232 версії С), який позволяє будувати системи обміну даних між пристроями із різними швидкодіями. Сучасні комп'ютерні технології широко застосовують для передачі даних послідовні інтерфейси. Інтерфейс RS-232C є одним із найстаріших та класичним представником цього класу інтерфейсів. Відповідно вивчення цього інтерфейсу є обґрунтованим.

Інтерфейс RS-232C розроблено в 1969 році низкою корпорацій США для забезпечення з'єднання комп'ютерів та різноманітних периферійних пристроїв. Аналогом інтерфейсу RS – 232C для колишніх країн економічної співдружності є спрощений варіант інтерфейсу стик С2. Інтерфейс переважно використовується для обміну даних між периферійним пристроєм та комп'ютером чи між двома комп'ютерами.

Термін "послідовний" означає, що передача даних виконується по одиночному провіднику, а біти передаються послідовно один за другим. У певний момент часу в одному напрямку передається тільки один біт даних.

Термін "радіальний" означає, що інтерфейс забезпечує взаємодію тільки двох пристроїв, один із яких є комп'ютером.

Термін "дуплексний" означає, що інтерфейс забезпечує можливість передачі даних між пристроями в обох напрямах, використовуючи для цього два окремі сигнальні провідники чи дві окремі сигнальні шини.

Термін "асинхронний" означає, що при передачі даних не використовуються ніякі синхронізуючі сигнали, і окремі символи можуть передаватись з випадковими інтервалами, як, наприклад при вводі даних з клавіатури. Інтерфейс використовує оригінальну синхронізацію, коли кожному символу, який передається через послідовне з'єднання, мусить передувати стандартний стартовий сигнал, а завершувати його передачу – стоповий сигнал. Стартовий сигнал – це умовно нульовий біт, його ще називають стартовим бітом. Його призначення – повідомити від передавального пристрою другому пристроєві, який приймає дані, про те, що наступні біти представляють собою поле даних. Поле даних, залежно від наперед запрограмованих режимів функціонування, може мати 5, 6, 7, або 8 біт. Після поля даних передається запрограмований біт контролю, а завершується обмін одного слова передачею одного, чи півтора, чи двох стопових бітів, які дають сигнал про закінчення передачі відповідного слова. У пристроєві, який приймає дані, біти розпізнаються за появою стартового сигналу по середині кожного такту, а не до моменту їх передачі. Тривалість одного такту визначається запрограмованою швидкістю обміну даними для конкретного режиму функціонування інтерфейсу. Асинхронний інтерфейс, орієнтований на передачу

окремих слів, передбачає значні непродуктивні затрати для ідентифікації кожного слова. Такі затрати можуть перевищувати 25 %.

Інтерфейс RS-232C широко використовують у персональних та промислових комп'ютерах для обміну інформацією з периферійними пристроями у послідовному дуплексному режимі обміну. У багатьох комп'ютерах інтерфейс реалізований як комунікаційний порт (COM-Port). Таких портів в одному комп'ютері може бути декілька. Інтерфейсний стандарт передбачає конструктивну сумісність пристроїв. Відповідно для кожного порта використовується стандартизований 25- чи 9-контактний з'єднувач.

Багато компаній випускають доповняльні послідовні порти для комп'ютерів, звичайно ті порти встановлюються на багатофункціональних платах чи на платі з паралельним портом. На рис. 1 показано приклад використання стандартного 9-контактного з'єднувача послідовного порта.

До послідовного порта можна підключити самі різноманітні пристрої: модеми, плотери, принтери, інші комп'ютери, пристрої зчитування штрих-кодів, схему керування пристроями чи макетну плату. Загалом у всіх пристроях, для яких потрібен низькошвидкісний, простий, двонапрямлений зв'язок з комп'ютером, використовується відповідний порт.

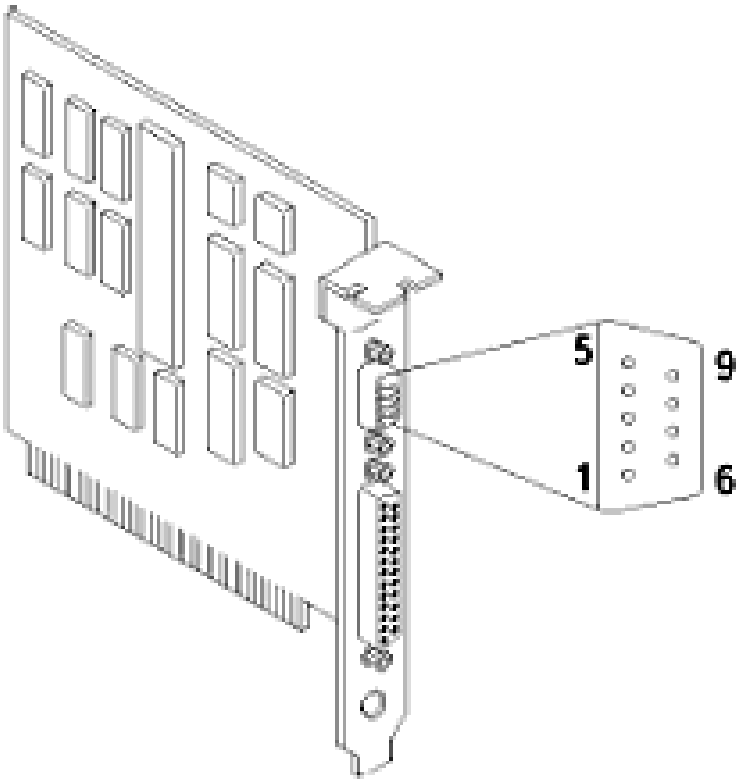


Рис. 1. Приклад використання 9-контактного з'єднувача послідовного порта

Стандарт RS-232 у загальному випадку описує 4 інтерфейсні функції:

1. Визначення керувальних сигналів, що передаються через інтерфейс.
2. Визначення формату даних користувача, що передаються через інтерфейс.

3. Передачу тактових сигналів для синхронізації потоку даних.

4. Формування електричних характеристик інтерфейсу.

Інтерфейс RS-232 є послідовним асинхронним інтерфейсом, в якому перед бітами даних передається спеціальний стартовий біт, після бітів даних йде біт паритета та 1 чи 2 стопових біти. Така сукупність бітів носить назву старт-стопного символу. Кожний старт-стопний символ, як правило, у якості бітів даних містить один інформаційний символ, наприклад, символ стандартного коду для обміну інформацією, що задається таблицею ASCII. Символи ASCII, що відображаються семибітовими кодовими словами, можуть передаватися із використанням 7-бітового поля даних. Так, наприклад, латинська буква А має код 1000001. Її передача рівнями транзисторно-транзисторної логіки (ТТЛ) зображена на часовій діаграмі (рис. 2, *а*). Передача рівнями сигналів RS - 232C для ліній зв'язку даних зображена на часовій діаграмі (рис. 2, *б*). Аналогічно формуються сигнали та відповідна часова діаграма при використанні 8-розрядної таблиці кодувань символів ASCII+.

Початок асинхронного символу завжди відмічається логічним нулем стартового сигналу (логічний "0"). Після нього йдуть 7 біт даних, потім біт паритету, та 2 стопових біти. Біт паритету встановлюється в логічні "1", або "0", наприклад, непарний паритет характеризується тим, що загальна кількість одиниць в групі з семи біт (сім даних + один біт паритету) повинно бути непарним числом. Стопові біти передаються високим рівнем (логічна "1").

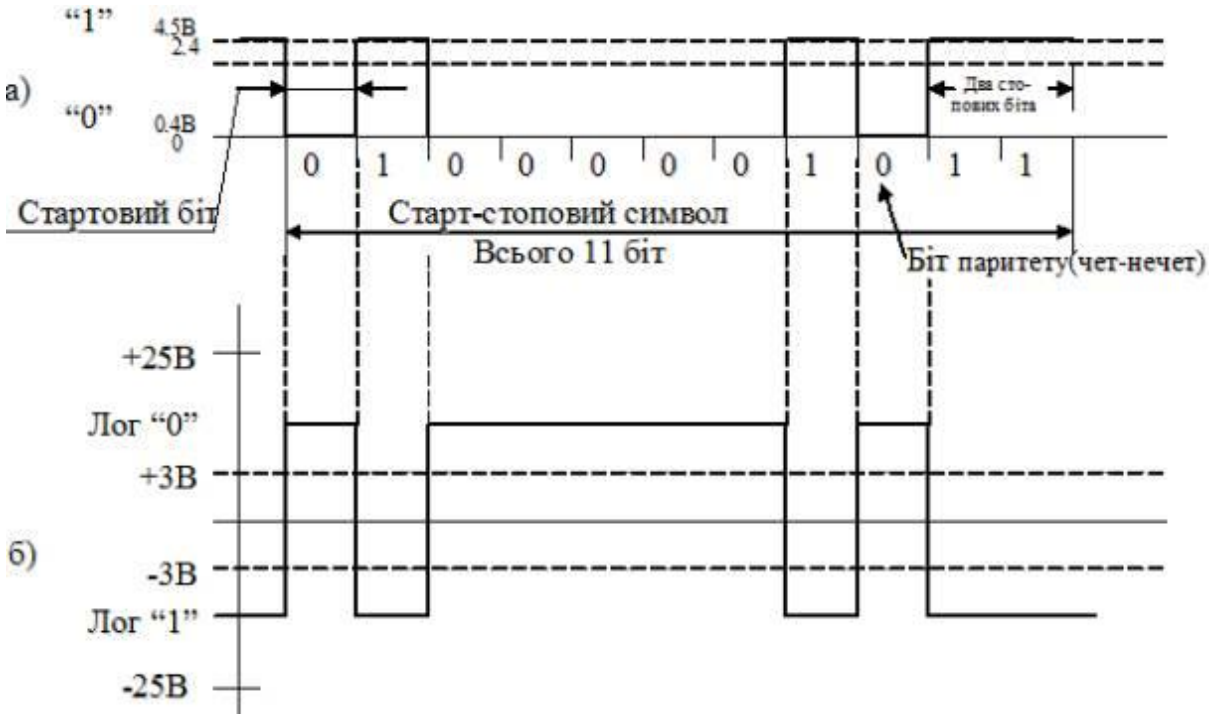


Рис. 2. Часові діаграми передачі символу для інтерфейсу RS-232C

Комп'ютер має 25-контактний (DB25P) або 9-контактний (DB9P) з'єднувачі для підключення пристрою інтерфейсу. Розташування контактів з'єднувачів зображене на

рис. 3. Рисунок засвідчує, що контакти з'єднувача DB25P використовуються неповно, тому більш ефективним є використання з'єднувача DB9P. Часто у з'єднувачі DB25P використовується контакт № 1 дла підєднання захисного екрана. Використання стандартизованих з'єднувачів спрощує підключення до комп'ютера різноманітних периферійних пристроїв.

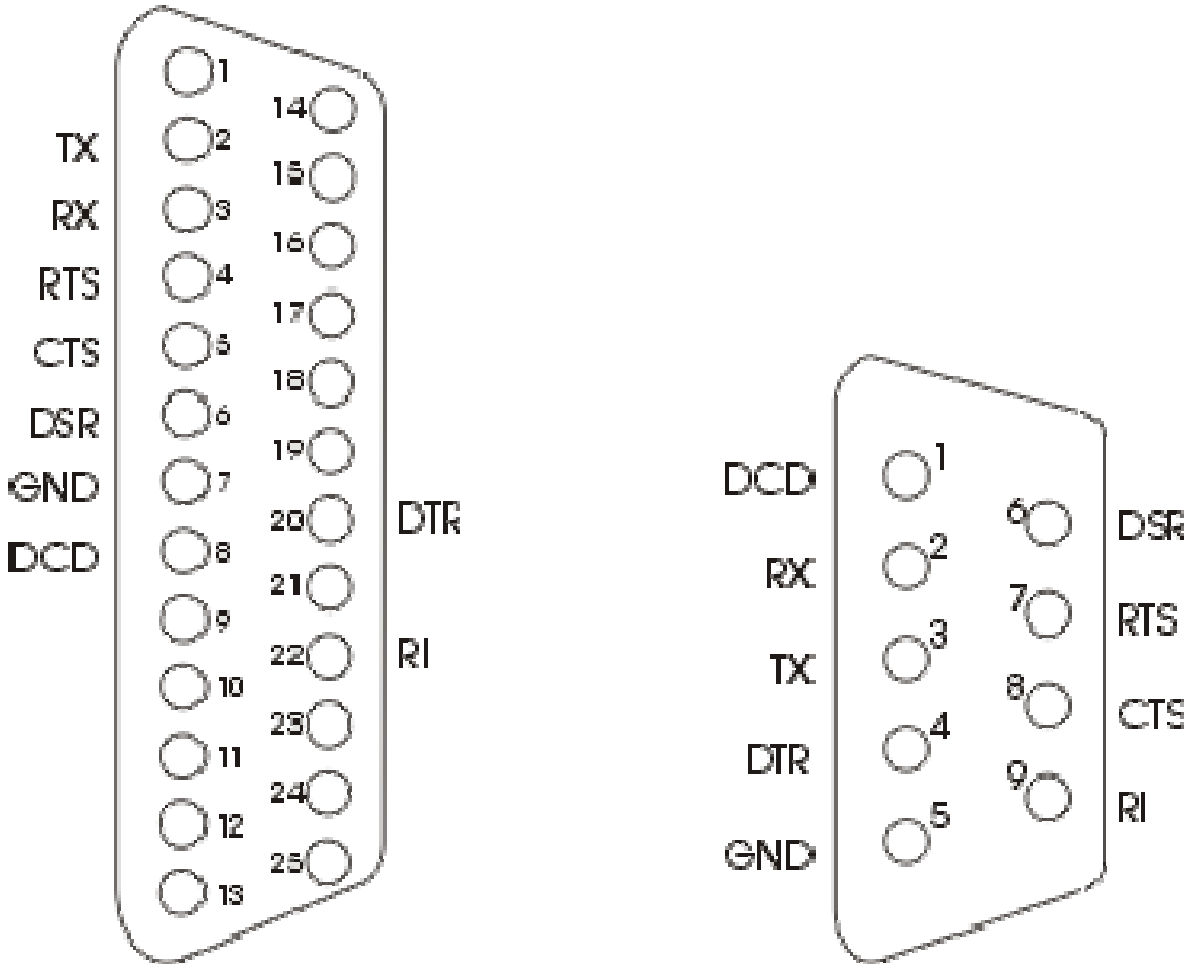


Рис. 3. Розташування контактів з'єднувачів DB25P та DB9P

Лінії зв'язку інтерфейсу можна умовно розділити на дві групи: шину даних та шину управління. Шина даних використовує дві лінії зв'язку для даних на передавання TXD та даних на приймання RXD. Цим забезпечується можливість реалізації повного дуплексного обміну даними. Шина управління має 6 ліній зв'язку, які використовуються для реалізації апаратного протоколу управління потоками даних. Призначення контактів з'єднувачів та відповідних сигналів приведене в табл. 1.

У випадку програмного протоколу управління потоками даних, коли застосовується спеціальний програмний драйвер. шина управління не використовуються тільки сигнали шини даних, що зменшує загальну кількість ліній зв'язку. Тоді застосовується три- або чотирипровідний зв'язок (для двонаправленої передачі). Схема з'єднання для чотирипровідної лінії зв'язку показана на рис. 4.

Таблиця 1

Призначення контактів з'єднувачів DB25P та DB9P

Наймену-вання	Контакт (з'єднувач DB25P)	Контакт (з'єднувач DB9P)	Напрям	Опис
DCD	8	1	IN	Carrie Detect (Визначення несучої)
- RXD	3	2	IN	Receive Data (Дані, що приймаються)
- TXD	2	3	OUT	Transmit Data (Дані, що передаються)
DTR	20	4	OUT	Data Terminal Ready (Готовність терміналу)
GND (SG)	7	5	-	System Ground (Сигнальна земля)
DSR	6	6	IN	Data Set Ready (Готовність даних)
RTS	4	7	OUT	Request to Send (Запит на відправку)
CTS	5	8	IN	Clear to Send (Готовність прийому)
RI	22	9	IN	Ring Indicator (Індикатор викликів)
FG	1	-	-	Захисне заземлення, екран

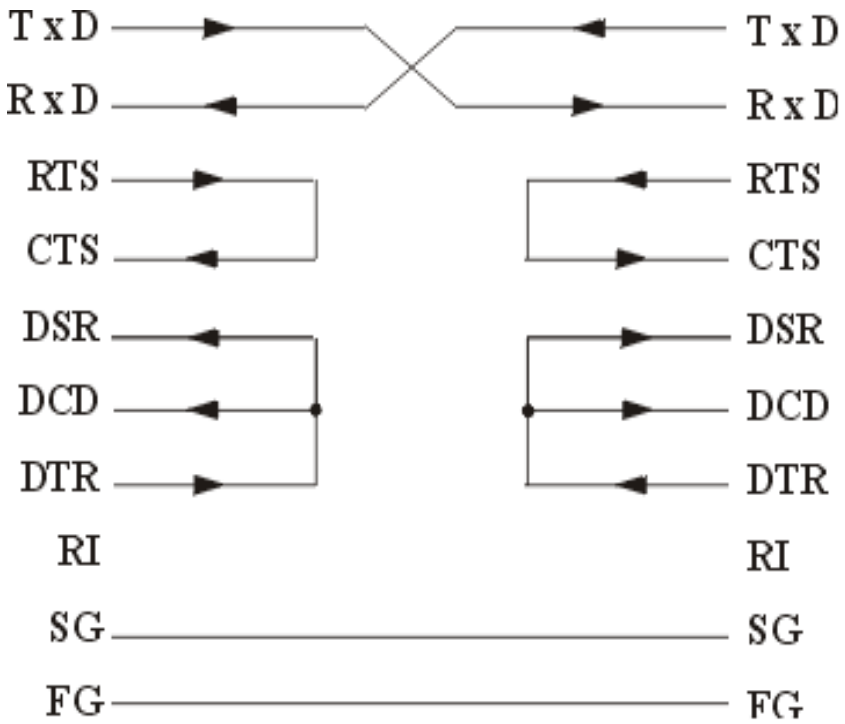


Рис. 4. Схема з'єднання 4-провідна для RS-232C

Для двопровідної лінії зв’язку у випадку передачі з комп’ютера на зовнішній пристрій використовуються сигнали SG і TxD. Всі 10 сигналів інтерфейсу задіюються лише при з’єднанні комп’ютера з модемом.

Формат слів, що передаються, показаний на рисунку 5. Рівень “1” відповідає логічній “1” , рівень “2” – логічному “0”. Формат програмується завчасно і має бути однаковим для передавача та приймача. Однаково мають бути запрограмованими швидкості передачі даних, які визначають тривалість одного такту. Тривалість одного такту є оберненою величиною до швидкості передавання даних. Якщо швидкість передавання даних задана в біт/с, тоді тривалість одного такту буде в долях секунди. Власне дані (5, 7 або 8 біти) супроводжуються стартовим бітом, бітом парності і одним або двома стоповими бітами. Отримавши передній фронт стартового біта, приймач переходить у режим самосинхронізації, формує тактові інтервали та очікує, коли пройде половина такту. Тоді приймач перевіряє наявність стартового сигналу. Якщо стартовий сигнал присутній, приймач встановлює, що йому від передавача поступає слово. Він вибирає з лінії зв’язку біти даних через визначені інтервали часу, на середині відповідного такта. Дуже важливо, щоб частоти синхронізуючих генераторів приймача і передавача були однаковими. Допустима розбіжність не більше 10 %, оскільки може відбутися втрата синхронізації і відповідно помилка обміну даними.

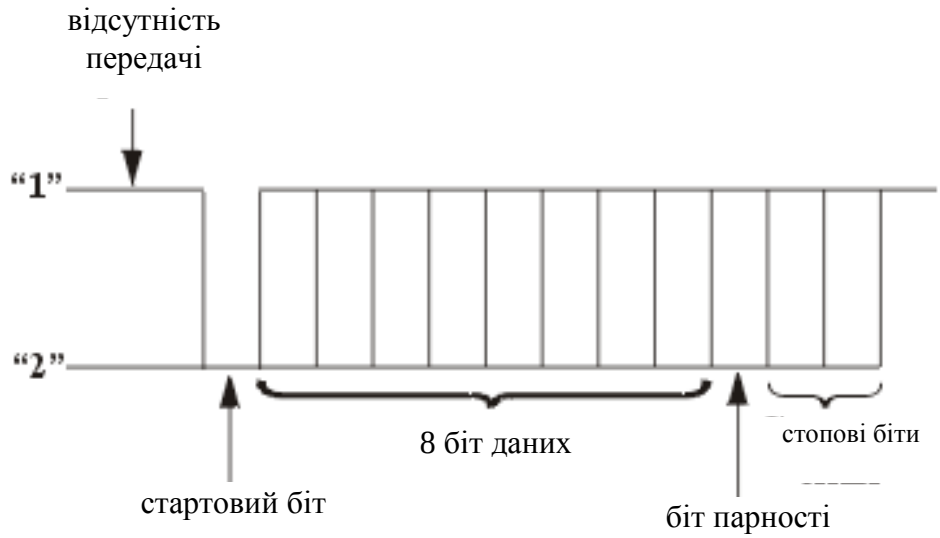


Рис. 5. Формат слова у логічних рівнях для RS-232C

Усі сигнали через лінії зв’язку RS-232C передаються спеціально вибраними рівнями, що забезпечують високу завадостійкість передачі даних (рис. 6). Відзначимо, що дані передаються в інверсному коді (логічній одиниці відповідає низький рівень, логічному нулю - високий рівень). Логічний нуль для передавача кодується величиною сигналу від +5 В до + 15 В, а для приймача від +3 В до + 15 В. Логічна одиниця для передавача кодується величиною сигналу від мінус 5 В до мінус 15 В, а для приймача від мінус 3 В до мінус 15 В. У

випадку використання нестабілізованих джерел живлення граничні значення сигналів можуть досягати 25 В.

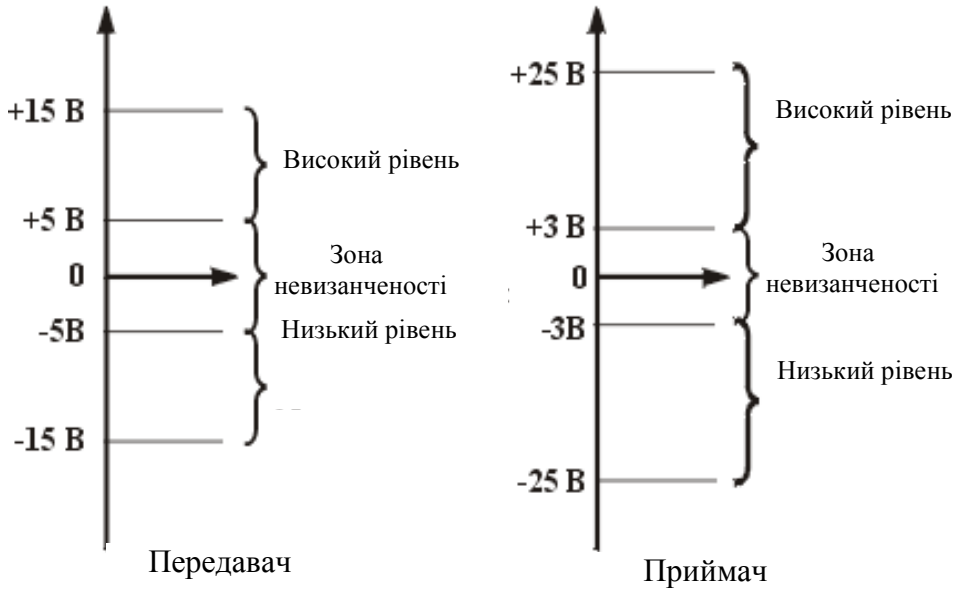


Рис. 6. Рівні сигналів RS-232C на передавальному і приймаючому кінцях лінії зв'язку

Швидкість передавання даних в асинхронному режимі може становити: 50, 75, 100, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, або 115200 біт/с. Швидкість програмується при налаштуванні режимів функціонування інтерфейсу і має бути однаковою як для передавального пристрою так для приймального пристрою.

Обмін по RS-232C здійснюється за допомогою звернень по спеціально виділених для цього портах COM1 (адреси 3F8h...3FFh, переривання IRQ4), COM2 (адреси 2F8h...2FFh, переривання IRQ3), COM3 (адреси 3F8h...3EFh, переривання IRQ10), COM4 (адреси 2E8h...2EFh, переривання IRQ11). Формати звернень по цих адресах можна знайти в описах мікросхем контролерів послідовного обміну UART (Universal Asynchronous Receiver/Transmitter), наприклад, i8250, КР580ВВ51.

Управління потоком даних (Flow Control) інтерфейсу має наступні особливості. Воно передбачає формування повідомлень про можливість чи неможливість передавання наступного слова. Можуть використовуватися два типи протоколу управління потоком даних: апаратний чи програмний.

Апаратний протокол управління потоком даних (Hardware Flow Control) на шинах даних передавання та приймання (TXD, RXD) використовує сигнали управління "Запит на передавання" (RTS), "Готовність приймання" (CTS). Сигнал CTS забезпечує зупинку передавання даних, якщо приймач не готовий їх приймати. Передавач передає у лінію зв'язку TXD чергове слово тільки при активному стані лінії CTS. Слово, яке вже почало

передаватися, зупинити сигналом CTS неможливо, його передавання має завершитися. Це гарантує цілісність передачі. Якщо у периферійній підсистемі апаратний протокол не використовується, у передавача має забезпечуватися стан “включено ” на лінії CTS перемиканням із лінією RTS. Апаратний протокол використовує лінії зв’язку шини управлінні, що збільшує вартість периферійної підсистеми. Однак забезпечує мінімальний час реакції передавача на стан при приймача, що підвищує загальну продуктивність інтерфейсних засобів. Оскільки вартість ліній зв’язку інтерфейсної магістралі достатньо велика, то на практиці застосування апаратного протоколу управління потоком даних постійно зменшується.

Програмний протокол управління потоком даних (Software Flow Control) використовує тільки шину даних (лінії TXD, RXD). При цьому використовуються спеціальні байт-символи XON/XOFF. Протокол функціонує наступним чином. Якщо приймач в процесі приймання даних встановить, що він не зможе правильно продовжувати приймання, він через другу лінію зв’язку шини даних посилає передавачу байт-символ XOFF (13h). Передавач, отримавши цей байт-символ, не передає наступного слова. Якщо приймач знову буде готовим до приймання даних, він посилає байт-символ XON (11h). Передавач після його отримання відновлює передавання даних. Час реакції передавача на зміну стану приймача у порівнянні із апаратним протоколом збільшується щонайменше на час передавання байт-символу XON чи XOFF плюс час реакції драйвера передавача на приймання символу. При такому протоколі управління приймач має мати буфер достатньої величини для приймання даних. Додатковим недоліком є складність забезпечення повного дуплексного режиму обміну даними. Однак мінімізація дорогих ліній зв’язку за рахунок виключення сигналів управління робить програмний протокол управління потоком даних широко вживаним.

Апаратні драйвери інтерфейсу переважно реалізовуються за допомогою спеціалізованих мікросхем. Перетворення паралельного коду від ядра комп’ютера в послідовний для передачі через лінію зв’язку та зворотнє перетворення при прийманні даних виконують спеціалізовані мікросхеми UART (Universal Asynchronous Receiver Transmitter – універсальний асинхронний передавач та приймач). Вони формують та опрацьовують керуючі сигнали інтерфейсу. Апаратні драйвери інтерфейсу в IBM PC сумісних комп’ютерах переважно реалізовуються на мікросхемах, сумісних на рівні регістрів з UART i8250: 8250; 8250A; 8250B; 16450; 16550. 16550A. Функціональні можливості та характеристики цих мікросхем постійно покращувалися і найбільш досконалою є остання.

Мікросхема UART i16550A як програмна модель є набором регістрів, доступ до яких відбувається за певною адресою (зміщенням адреси регістра відносно базової адреси порта) та значенням біта DLAB (біта 7 регістра налаштування параметрів канала LCR). В адресному просторі мікросхема займає 8 суміжних адрес. Регісри мікросхеми використовуються для програмування апаратного драйвера на одну із стандартних швидкостей обміну да-

ними, кількості біт (5,6,7,8) у полі даних слова, тип контролю (парність, непарність) чи відсутність контролю, кількості біт у полі стоп слова.

Мікросхеми UART використовують рівні сигналів ТТЛ (транзисторно-транзисторна логіка) із діапазоном зміни напруги 5В. Сигнали комунікаційних портів і відповідно інтерфейсних ліній зв'язку використовують RS сигнали в діапазоні від мінус 12 В до +12 В. Для узгодження із сигналами ліній зв'язку інтерфейсу використовуються спеціалізовані мікросхеми перетворення рівнів сигналів ТТЛ у рівні сигналів RS-232C при передаванні та зворотнього перетворення при прийманні.

Лабораторна робота № 1

РОЗРОБЛЕННЯ ТА ДОСЛІДЖЕННЯ ДРАЙВЕРА ПЕРЕДАВАЧА ІНТЕРФЕЙСА RS-232C

МЕТА РОБОТИ: опанування студентом технології та процесу налаштування передавального порту створення програми передавача пакетних даних через послідовний асинхронний інтерфейс RS-232C (COM-порт).

Завдання на роботу. Конкретний пакет даних та відповідне повідомлення студенту визначає викладач (переважно невелике повідомлення із великих букв кирилиці, що кодуються модифікованою таблицею ASCII+). Задається швидкість обміну даними, тип контролю, кількість стопових біт.

Технологія виконання роботи. Передавальний порт характеризується можливістю функціонування у двох режимах: програмування, тобто налаштування, та безпосереднього передавання інформації. Порт повинен бути налаштованим на характеристики, які індивідуально задані студенту. Це швидкість обміну даними, кількість розрядів поля даних, тип контролю, кількість стопових біт. За замовчуванням у полі даних 8 двійкових розряди. Після режиму налаштування передавальний порт має бути переведений у режим безпосереднього передавання даних.

Для створення драйвера необхідно розробити відповідну схему алгоритму. Розроблення схеми алгоритму є одним із показників поглибленого розуміння логіки первинного налаштування передавального порту.

Для створення програми передавача даних через послідовний асинхронний інтерфейс RS-232C (COM-порт) можна використовувати компоненту SerialGate (рис. 7), див. лістінги програми.

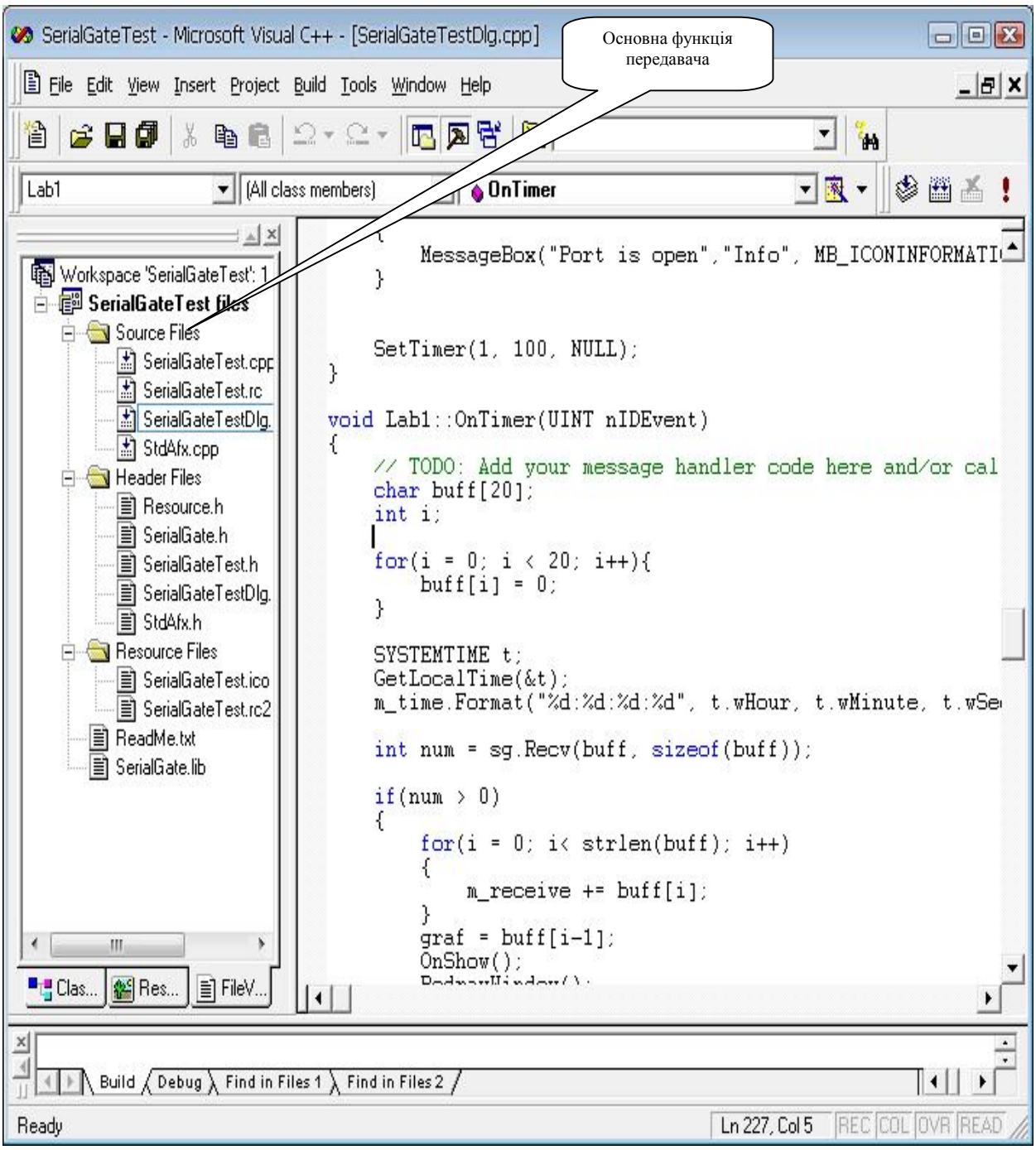


Рис. 7. Вікно створення функції

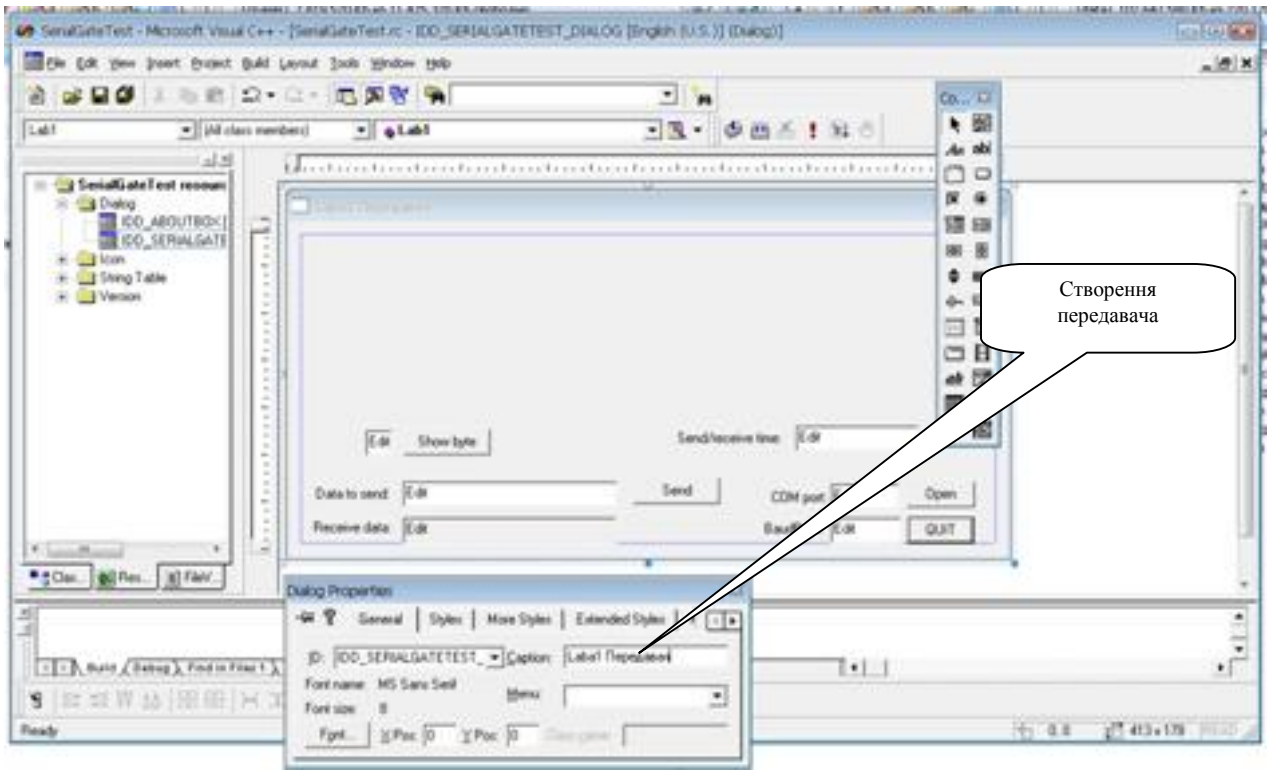


Рис. 8. Вікно створення назви програми

Для створення програми передавача даних через послідовний асинхронний інтерфейс RS-232C (COM-порт) використовуємо вікно змін назви проєкту (рис. 8).

На рис. 9 представлено результати побудови проєкту програми-передавача.

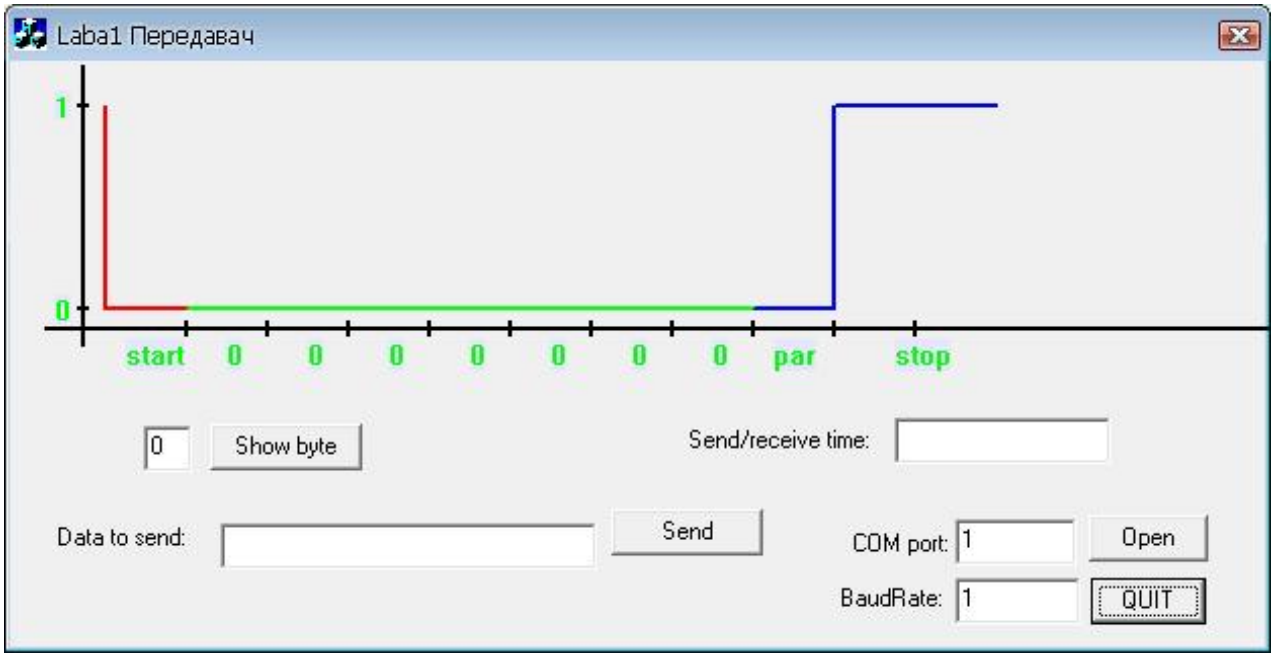


Рис. 9. Результат побудови проєкту

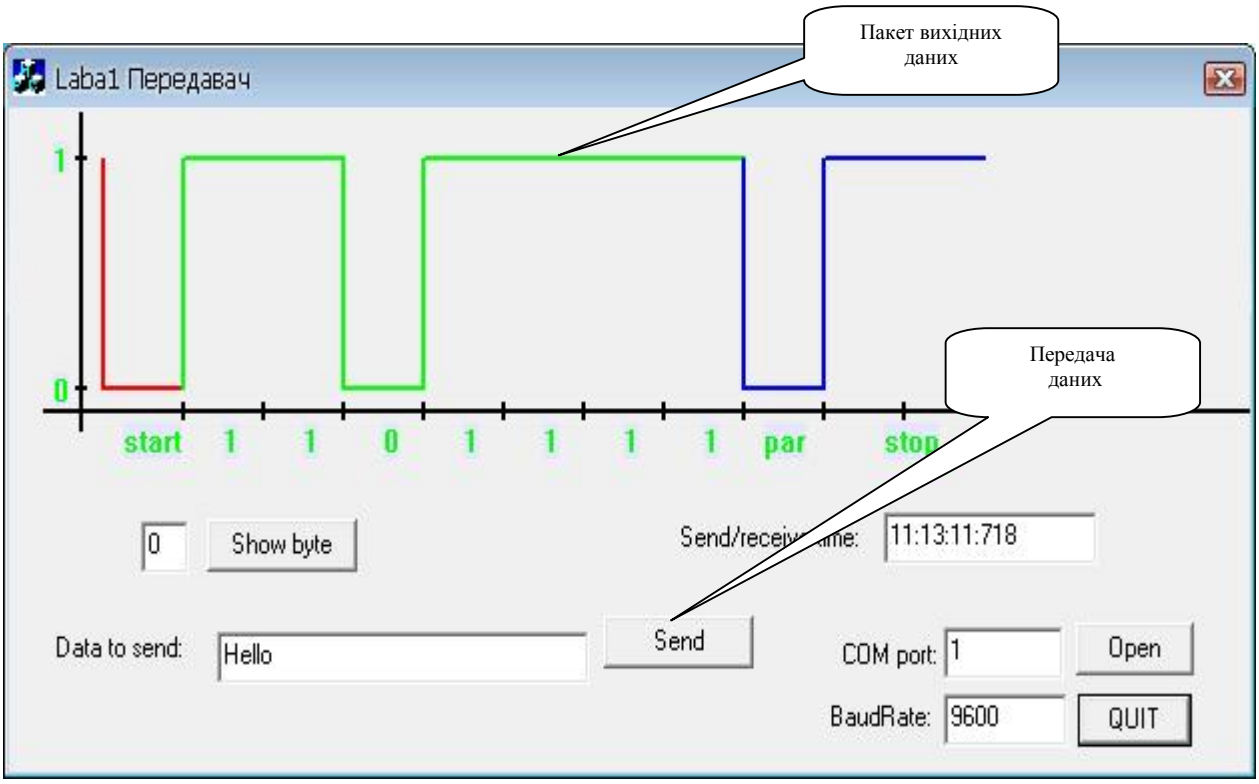


Рис. 10. Вікно програми передавача

При передаванні даних вводимо інформацію у вікно Data to send, налаштовуємо параметри передачі та натискаємо кнопку Send (рис. 10). Програма передавача перетворює паралельну однобайтову інформацію у послідовний код поля даних для кожного слова.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Привести основні характеристики інтерфейсу RS-232C .
2. Привести призначення сигналів інтерфейсу RS-232C.
3. Привести формат даних інтерфейсу RS-232C.
4. Привести формат слова передавального порту.
5. Привести особливості основних режимів функціонування передавального порту.
6. Пояснити, яким способом передавальний порт переводиться з режиму налаштування в режим основної роботи
7. Пояснити основні принципи розроблення схеми алгоритму первинного налаштування передавального порту.
8. Привести основні елементи схеми алгоритму первинного налаштування передавального порту.
9. Привести основні етапи відлагодження програми передавача.
10. Пояснити роботу написаної програми передавача.

ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Короткі теоретичні відомості.
3. Схема алгоритму налаштування передавача.
4. Хід роботи.
5. Висновки.
6. Текст програми (у додатку).
7. Демонстрація на комп’ютері програми для заданого пакету даних.

Лабораторна робота № 2

РОЗРОБЛЕННЯ ТА ДОСЛІДЖЕННЯ ДРАЙВЕРА ПРИЙМАЧА ІНТЕРФЕЙСА RS-232C

МЕТА РОБОТИ: опанування студентом технології створення програми приймача пакетних даних через послідовний асинхронний інтерфейс RS-232C (COM-порт).

Завдання на роботу. Конкретний пакет даних та відповідне повідомлення студенту визначає викладач (переважно невелике повідомлення із великих букв кирилиці, що кодуються модифікованою таблицею ASCII+). Задається швидкість обміну даними, тип контролю, кількість стопових біт.

Технологія виконання роботи. Приймальний порт характеризується можливістю функціонування у двох режимах: програмування, тобто налаштування, та безпосереднього приймання інформації. Порт повинен бути налаштованим на характеристики, які індивідуально задані студенту. Це швидкість обміну даними, кількість розрядів поля даних, тип контролю, кількість стопових біт. За замовчуванням у полі даних 8 двійкових розряди. Після режиму налаштування приймальний порт має бути переведений у режим безпосереднього приймання даних.

Для створення драйвера необхідно розробити відповідну схему алгоритму. Розроблення схеми алгоритму є одним із показників поглибленого розуміння логіки первинного налаштування приймального порту.

Для створення програми приймача даних через послідовний асинхронний інтерфейс RS-232C (COM-порт) можна використовувати компоненту SerialGate (рис. 11), див. лістінги програми.

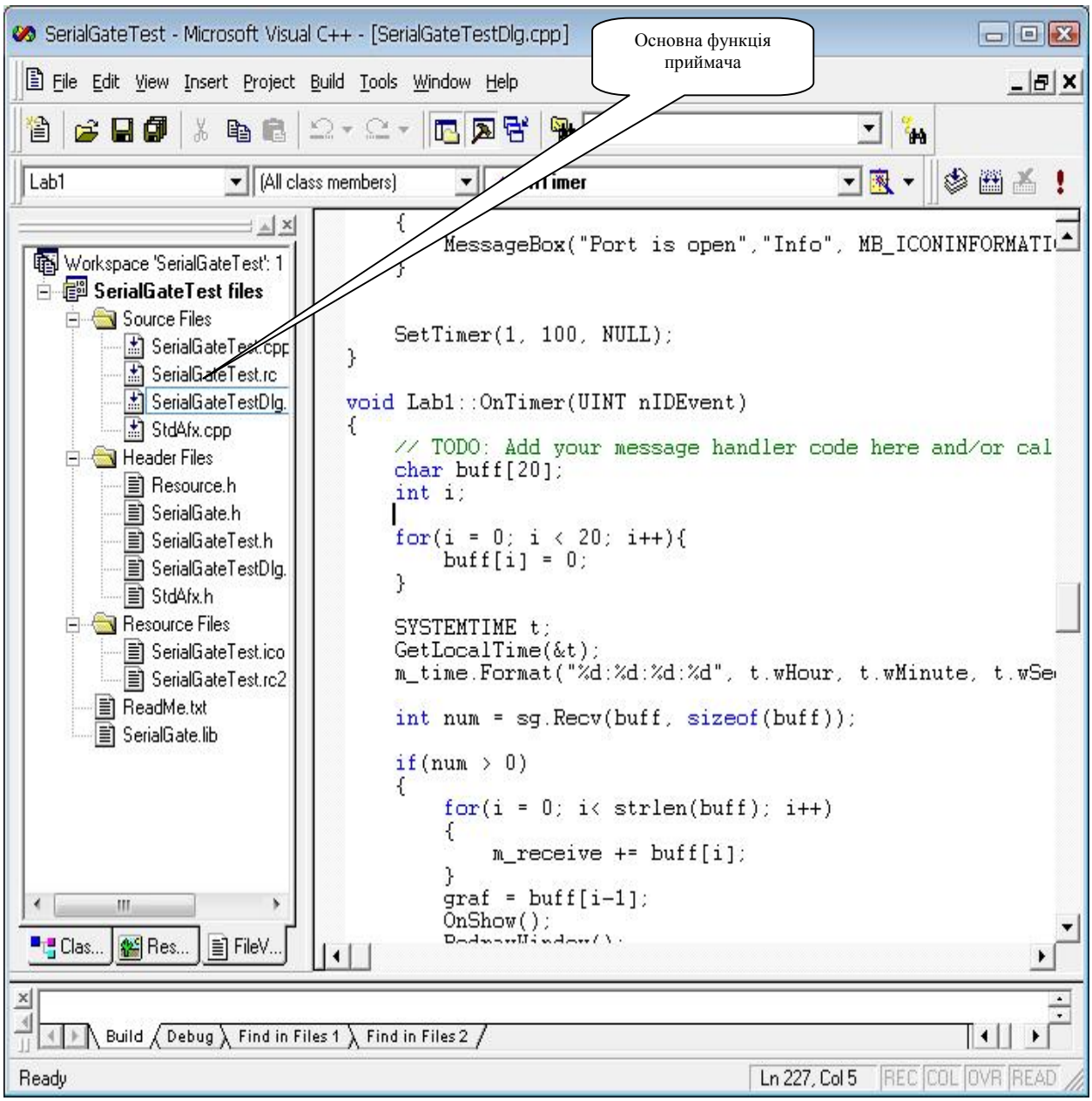


Рис. 11. Вікно створення функції

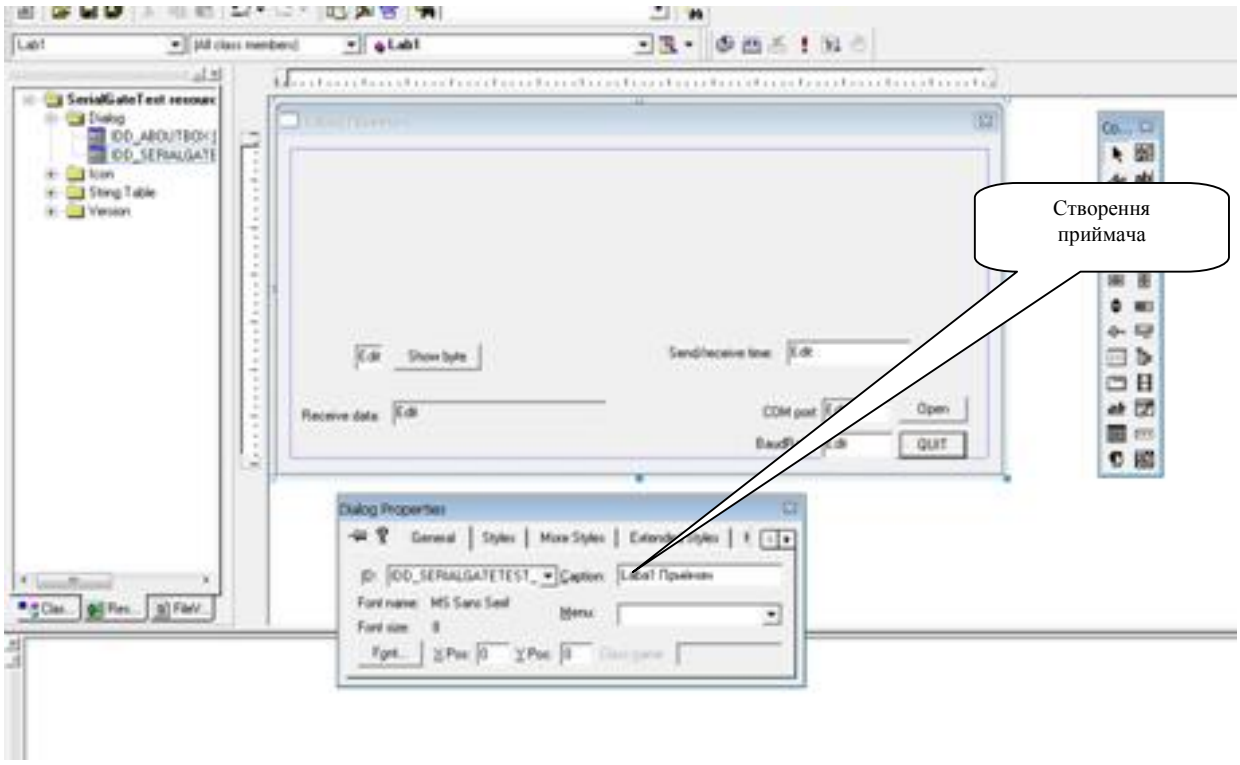


Рис. 12. Вікно створення назви програми

Для створення програми приймача даних через послідовний асинхронний інтерфейс RS-232C (COM-порт) використовуємо вікно змін назви проєкту (рис. 12).

На рис. 13 представлено результати побудови проєкту програми-передавача.

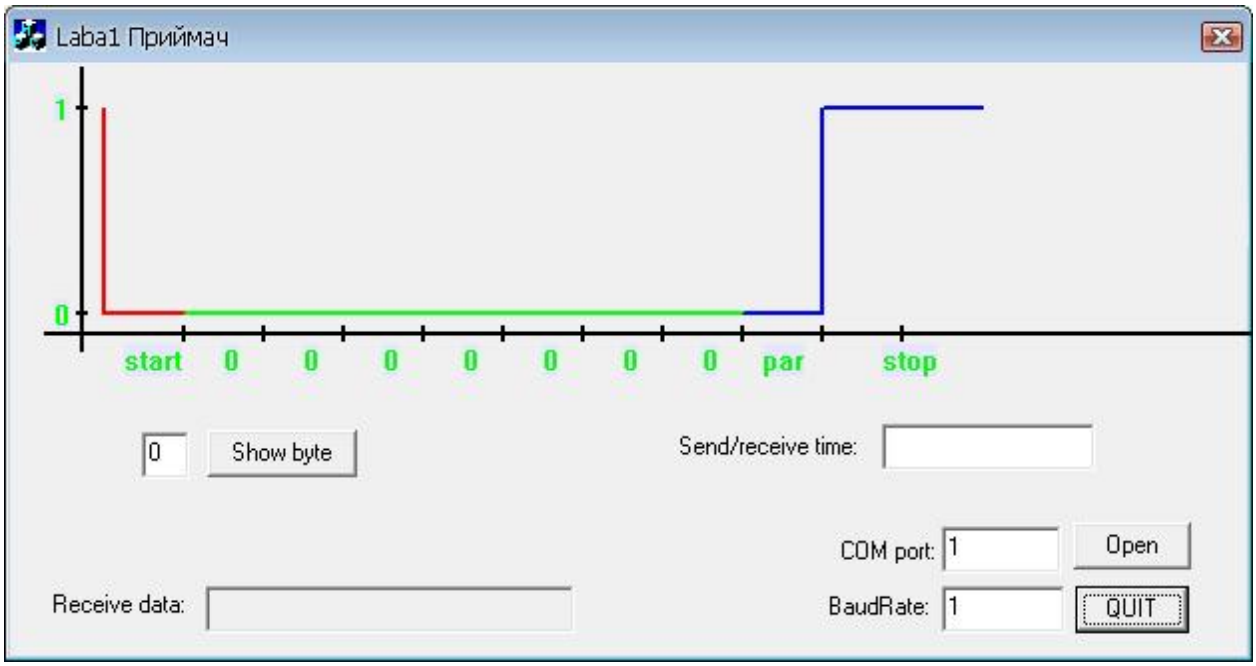


Рис. 13. Результат побудови проєкту

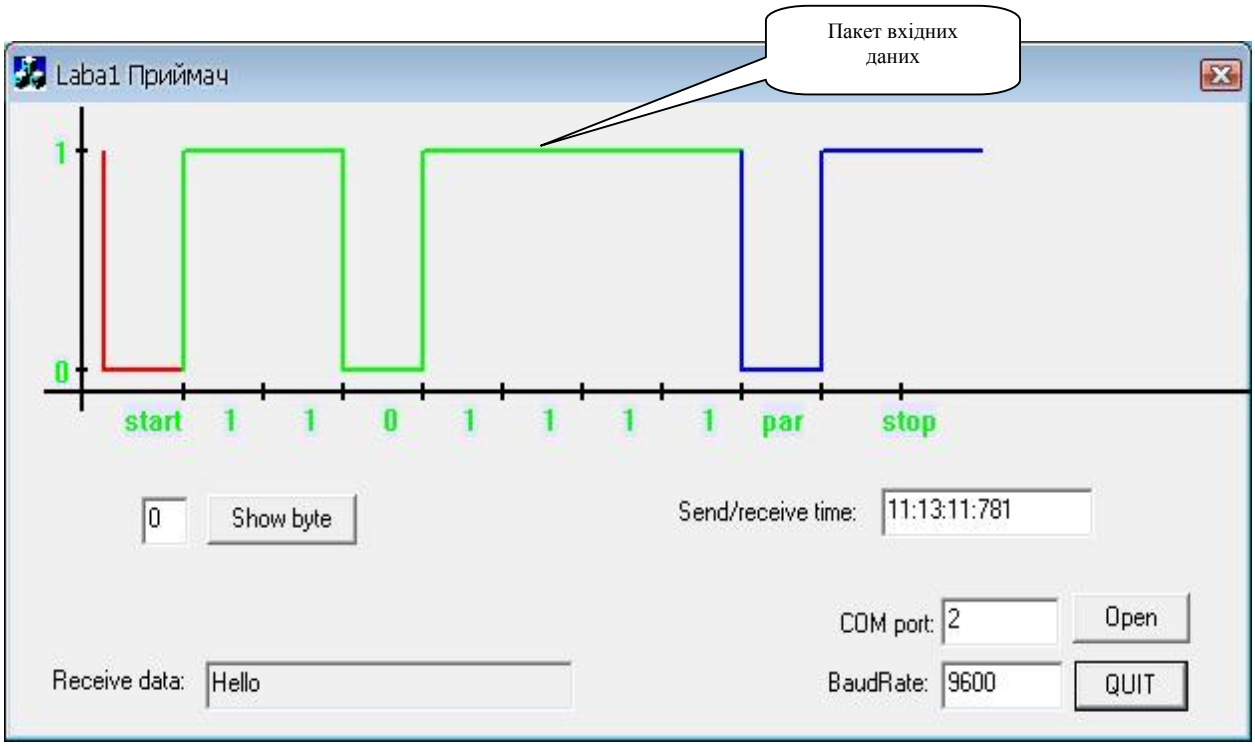


Рис. 14. Вікно програми приймача

Програма приймача перетворює прийнятий послідовний код поля даних паралельну однобайтову інформацію для кожного слова, декодовує у символи та відображає на екрані.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Привести ряд швидкостей обміну інформацією інтерфейсу RS-232C.
2. Привести призначення сигналів інтерфейсу RS-232C.
3. Привести формат даних інтерфейсу RS-232C.
4. Привести формат слова приймального порту.
5. Привести особливості основних режимів функціонування приймального порту.
6. Пояснити, яким способом приймальний порт переводиться з режиму налаштування в режим основної роботи.
7. Пояснити основні принципи розроблення схеми алгоритму первинного налаштування приймального порту.
8. Привести основні елементи схеми алгоритму первинного налаштування приймального порту.
9. Привести основні етапи відлагодження програми приймача.
10. Пояснити роботу написаної програми приймача.

ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Короткі теоретичні відомості.
3. Схема алгоритму налаштування приймача.
4. Хід роботи.
5. Висновки.
6. Текст програми (у додатку).
7. Демонстрація на комп’ютері програми для заданого пакету даних.

Лабораторна робота № 3

ДОСЛІДЖЕННЯ ГРАФІЧНОГО ПРЕДСТАВЛЕННЯ СИГНАЛІВ ЛІНІЇ ЗВ'ЯЗКУ

МЕТА РОБОТИ: опанування студентом особливостей передачі пакету даних через лінію зв'язку у графічному представленні сигналівдля послідовного асинхронного інтерфейса RS-232C (COM-порта).

Конкретний пакет даних студентові визначає викладач (переважно невелике повідомлення із великих букв кирилиці, що кодуються модифікованою таблицею ASCII+). Задається швидкість обміну даними, тип контролю, кількість стопових біт.

Для графічного представлення процесу передавання повідомлення доцільно використати графічний редактор VISIO. Дослідження також можна здійснити спеціальними програмними засобами, коли створюємо функцію OnShow (рис. 15), див. лістінги програми.

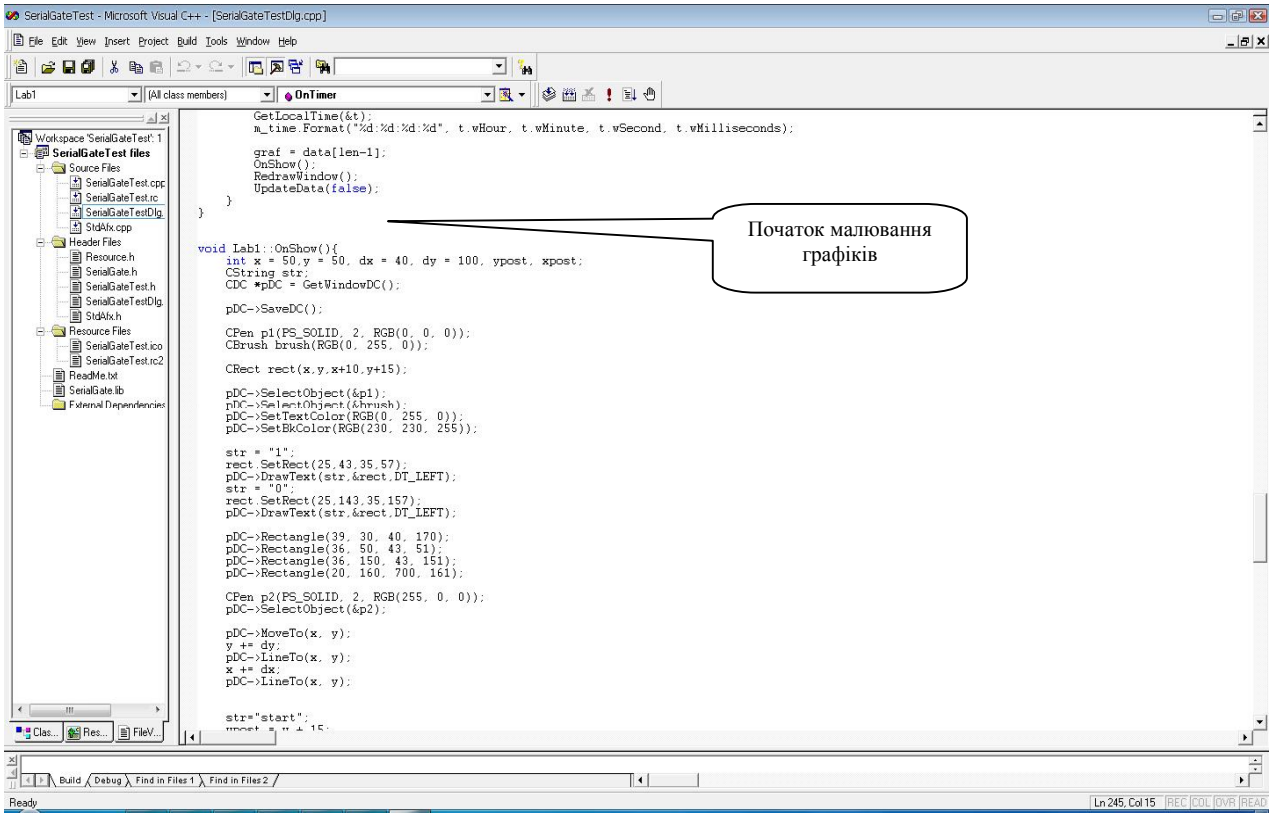


Рис. 15. Вікно створення функції малювання графіків

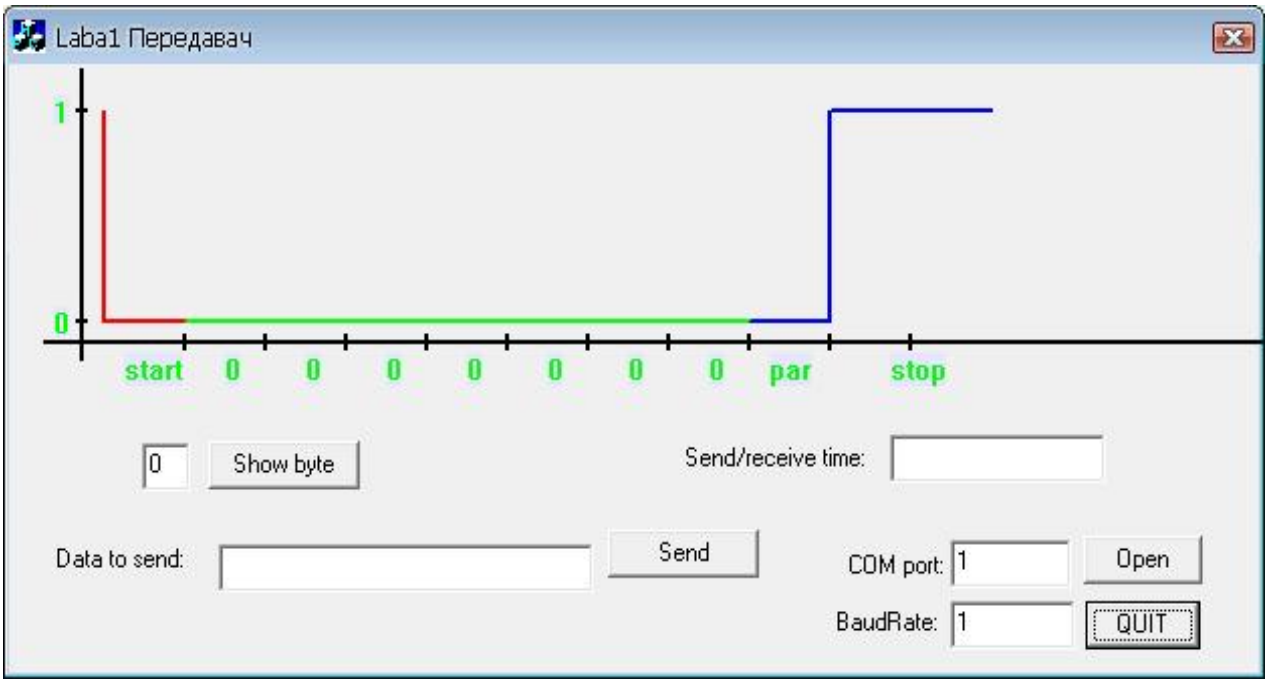


Рис. 16. Вікно програми передавача

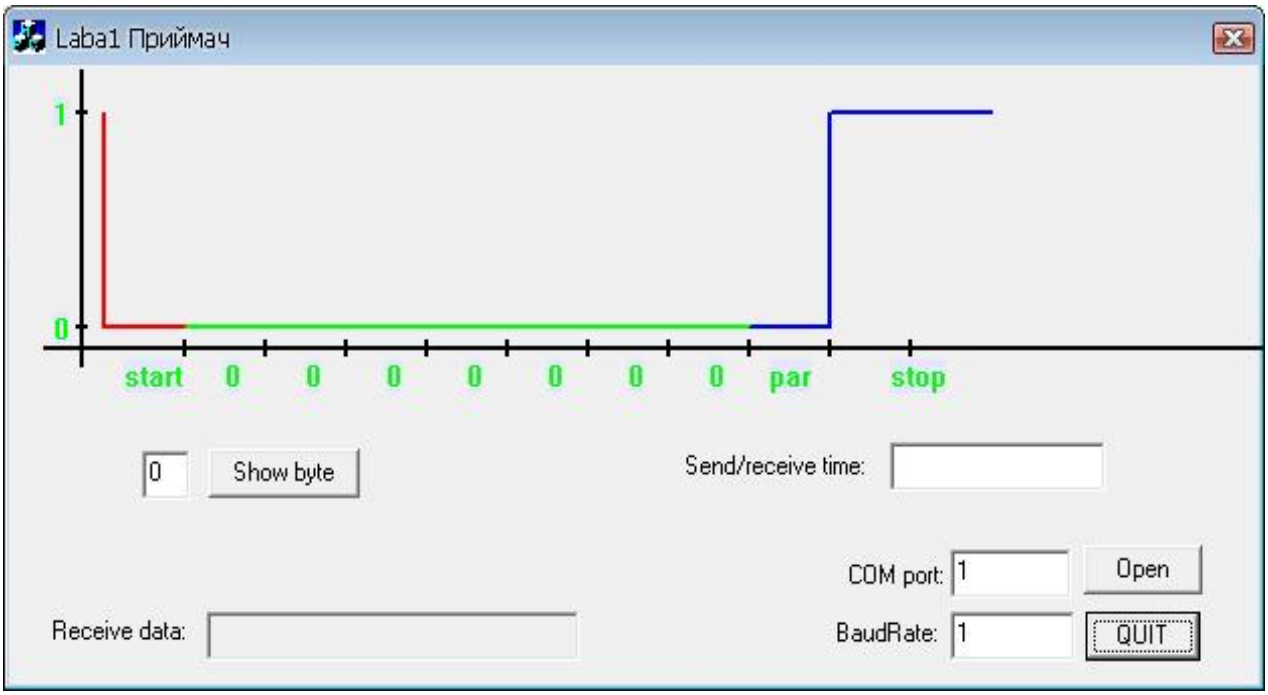


Рис. 17. Вікно програми приймача

На рис. 18 показано варіант реалізації функції побайтного виведення інформаційних даних із пакету (див. лістінги програми).

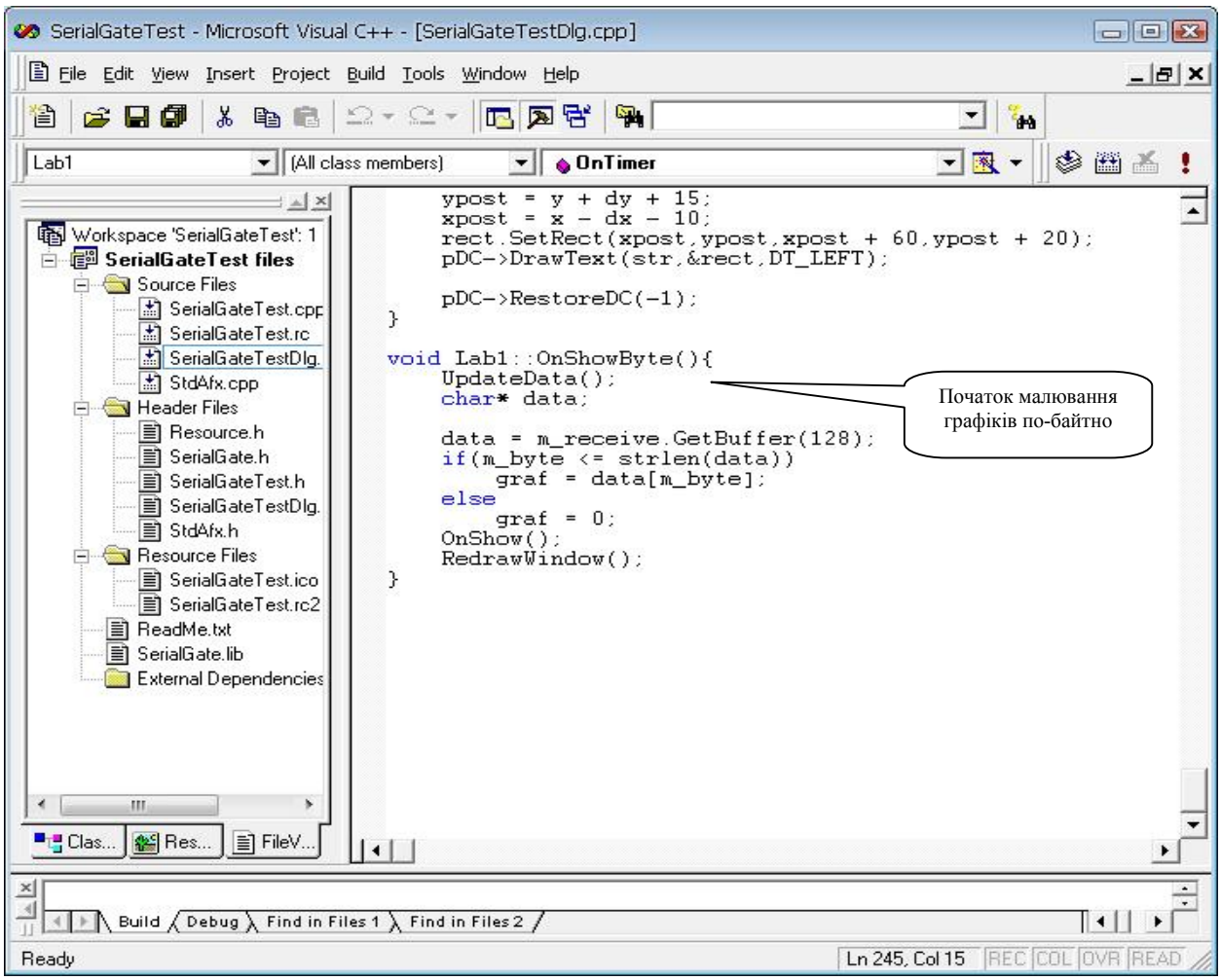


Рис. 18. Вікно створення функції

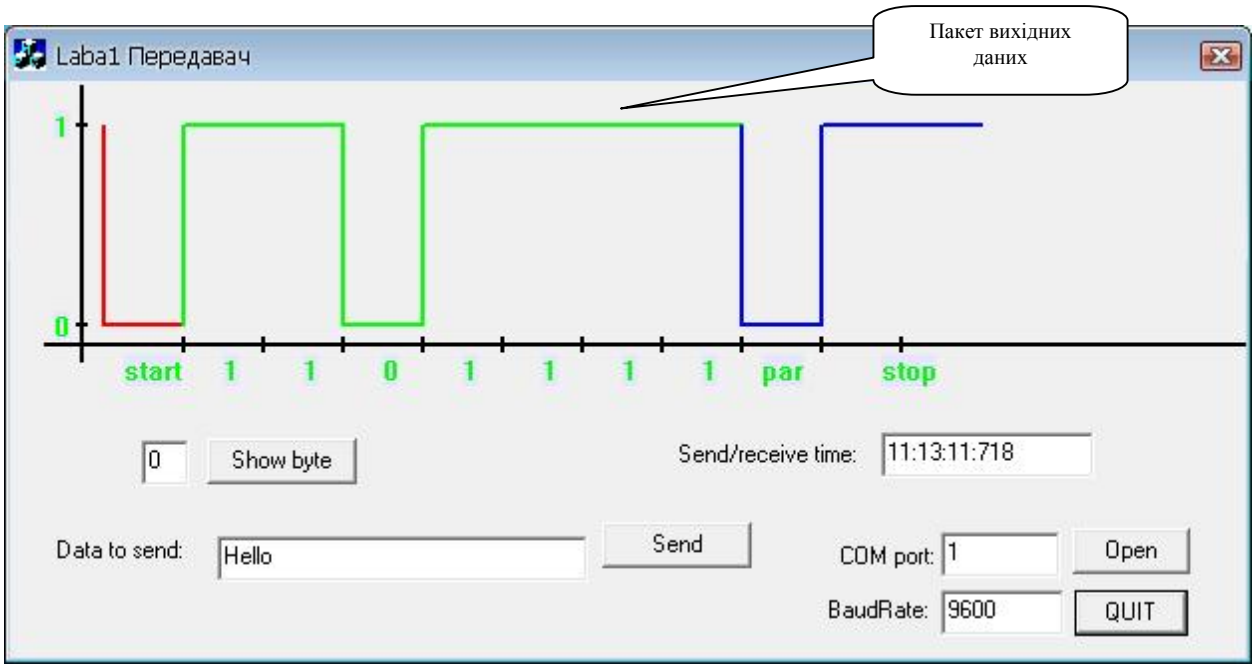


Рис. 19. Вікно часової діаграми для передавача

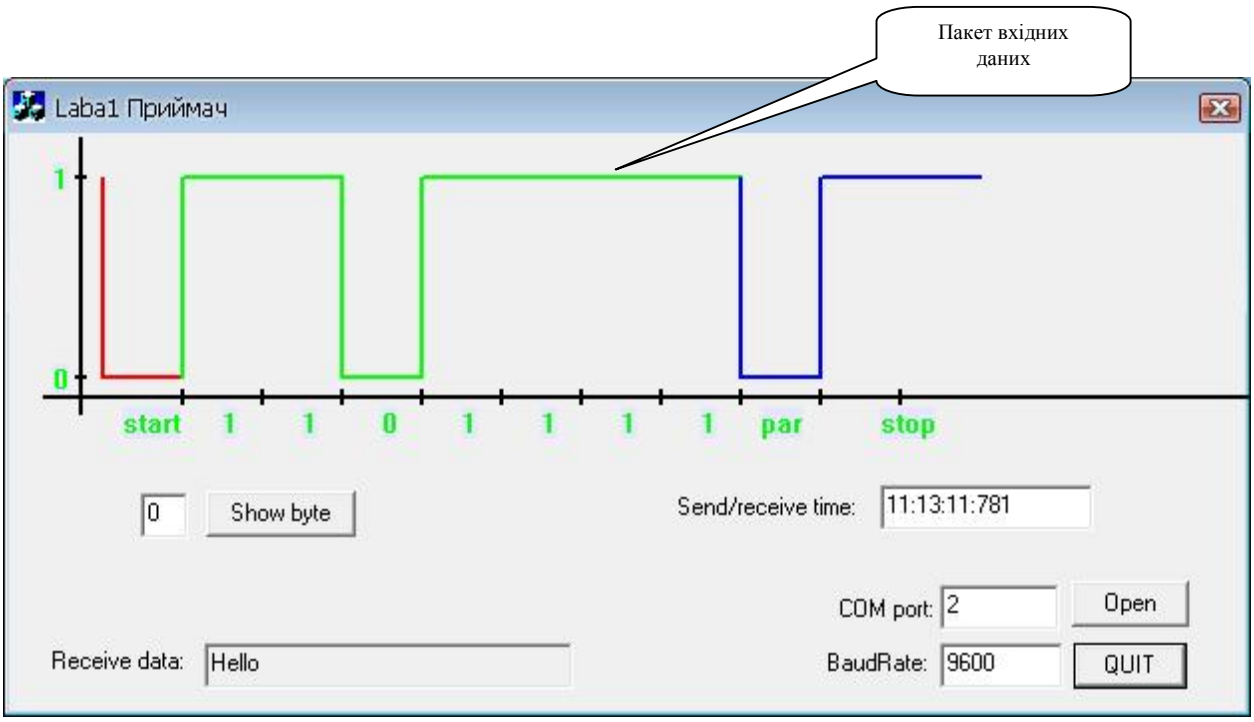


Рис. 20. Вікно часової діаграми для приймача

На рис. 19 та рис. 20 показано графічне представлення даних прийому – передачі пакета даних через послідовний асинхронний інтерфейс RS-232C (COM-порт) для одного символу. Вони ідентичні, тому доцільно провести дослідження тільки для передавача або тільки для приймача. При виконанні роботи розробити та навести часову діаграму для усього заданого повідомлення із паузою між символами тривалістю один такт.

У звіті до лабораторної роботи достатньо навести амплітудно-часову діаграму передачі заданого пакету даних або тільки для передавача, або тільки для приймача, оскільки вони мають бути ідентичними. Необхідно привести розрахунок зального часу передачі заданого повідомлення.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Привести співвідношення швидкості обміну інформацією та тривалості одного такту для інтерфейсу RS-232C.
2. Привести сигнальні рівні кодування логічної одиниці для передавача.
3. Привести сигнальні рівні кодування логічної одиниці для приймача.
4. Привести сигнальні рівні кодування логічного нуля для передавача.
5. Привести сигнальні рівні кодування логічного нуля для приймача.
6. Пояснити, яким способом можна розрахувати загальний час передавання заданого повідомлення.

ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Короткі теоретичні відомості.
3. Хід роботи, графічне подання амплітудно-часової діаграми сигналів на лінії зв’язку даних для заданого повідомлення.
4. Розрахунок загального часу передавання заданого повідомлення.
5. Висновки.
6. Текст програми (у додатку).
7. Демонстрація на комп’ютері програми для заданого пакету даних.

Лабораторна робота № 4

РОЗРОБЛЕННЯ ТА ДОСЛІДЖЕННЯ МОДЕЛІ ПЕРЕДАЧІ ПОВІДОМЛЕННЯ ІНТЕРФЕЙСА RS-232C

МЕТА РОБОТИ: опанування студентом технології налаштування основних параметрів прийому – передачі пакетних даних через послідовний асинхронний інтерфейс RS-232C (COM-порт) та реалізувати модель обміну заданим повідомленням.

Конкретний пакет даних студенту визначає викладач (переважно невелике повідомлення із великих букв кирилиці, що кодуються модифікованою таблицею ASCII+). Задається швидкість обміну даними, тип контролю, кількість стопових біт.

Щоб з'єднати два віртуальні порти, обираємо тип з'єднання типу "пара" (рис. 21).

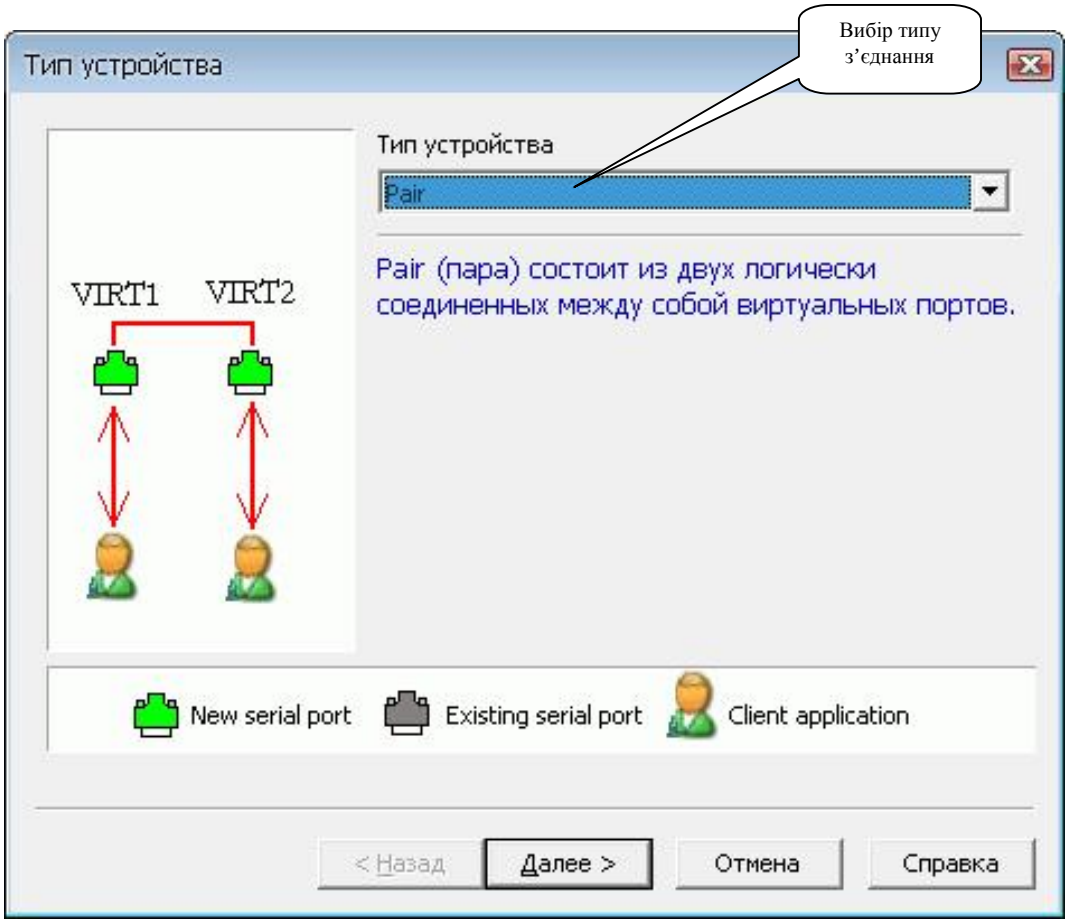


Рис. 21. Вікно вибору типу з'єднання

На рис. 22 продемонстровано вибір портів для обміну заданим повідомленням.

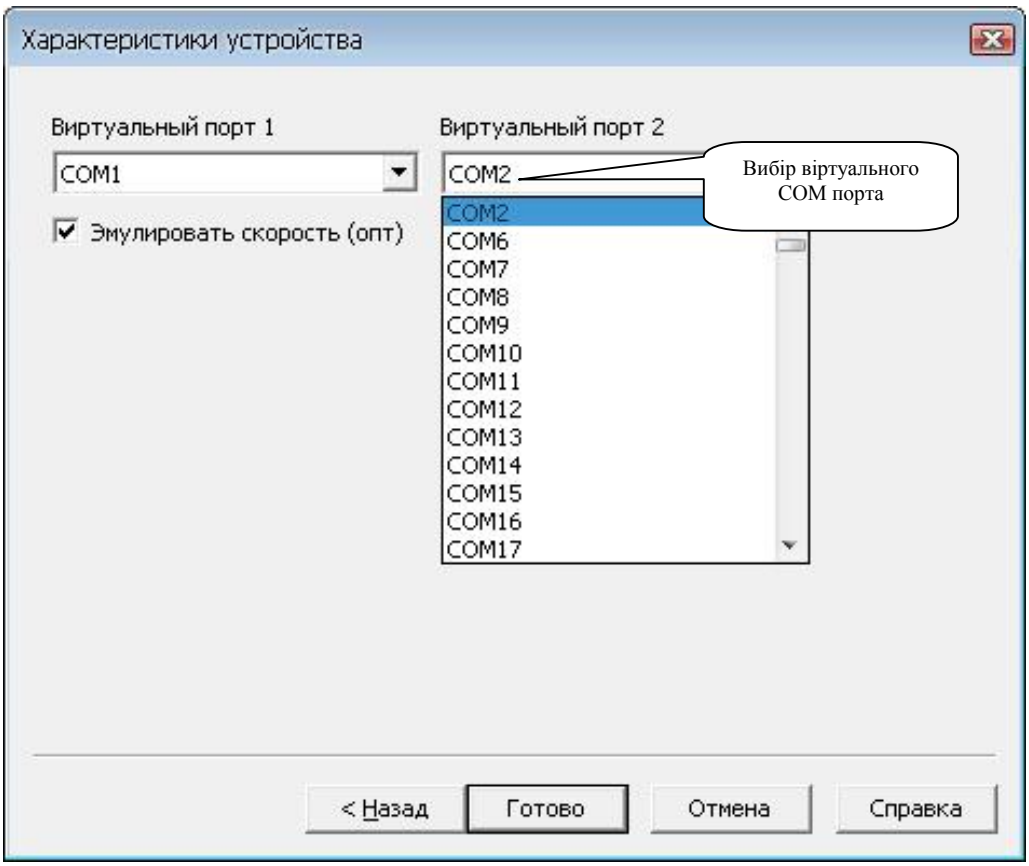


Рис. 22. Вікно вибору портів

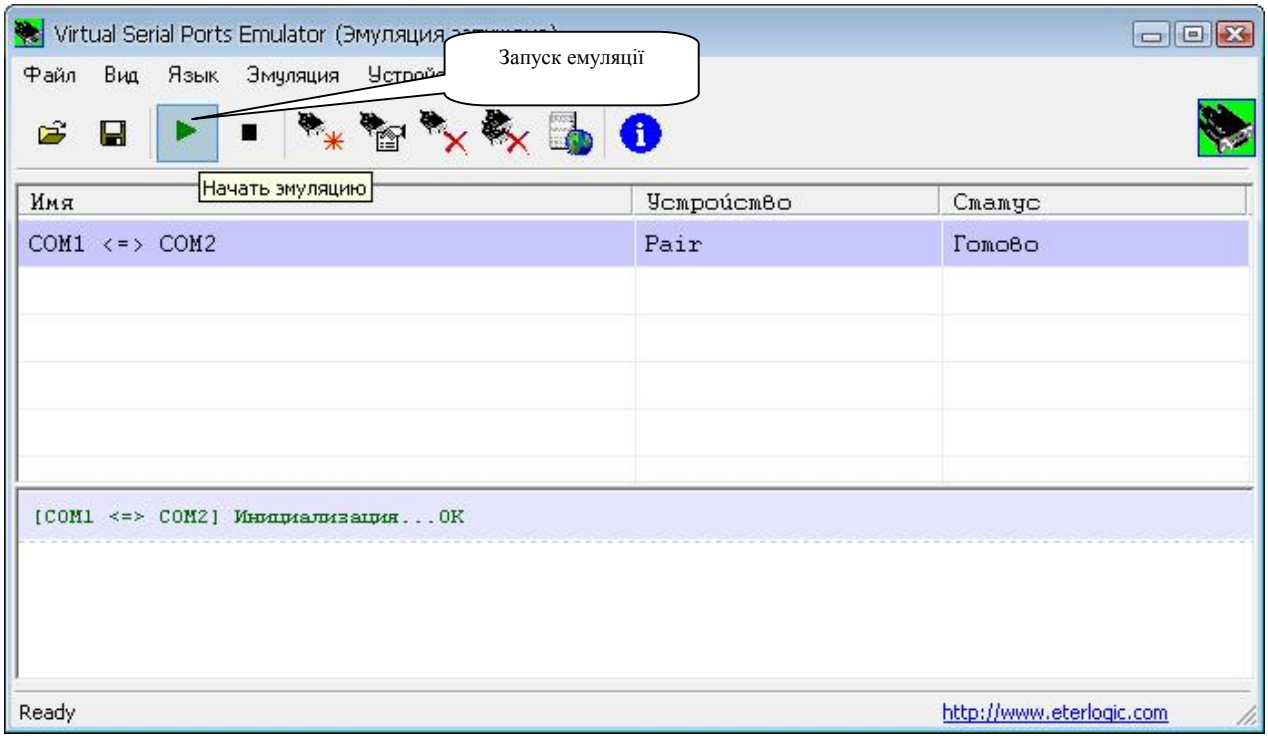


Рис. 23. Вікно запуску емуляції

Для коректного обміну даними встановлюємо номери портів та швидкості передачі даних для програми передавача та програми приймача (рис. 24, рис. 25).

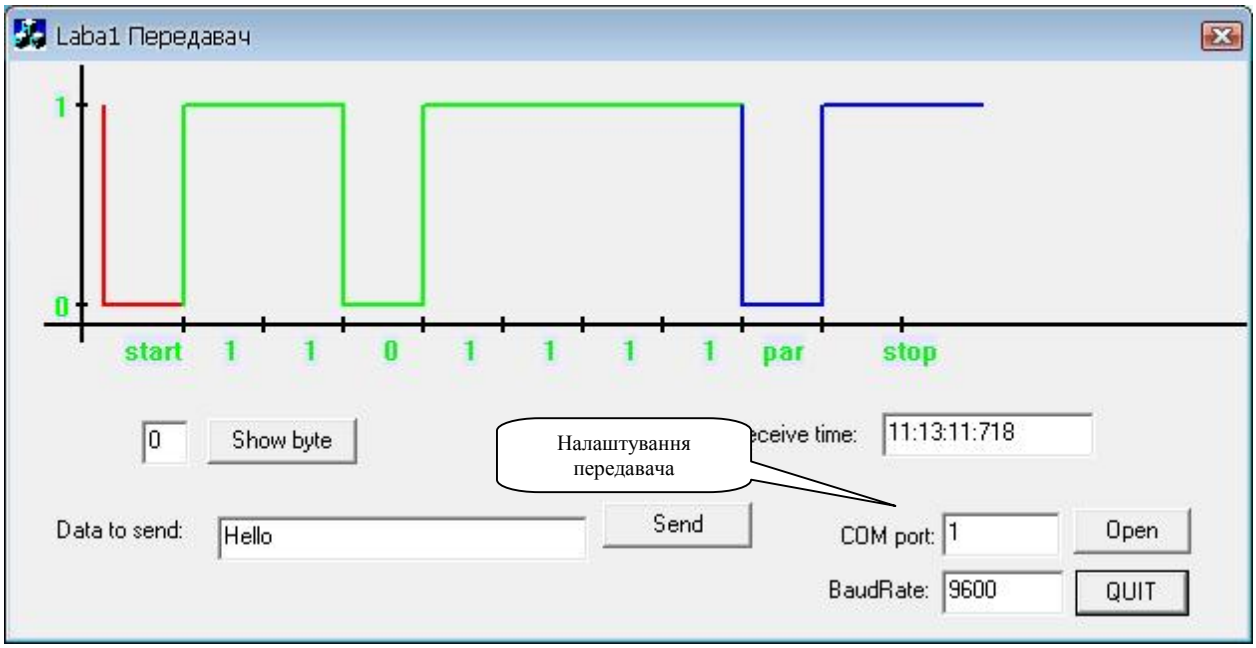


Рис. 24. Вікно програми передавача

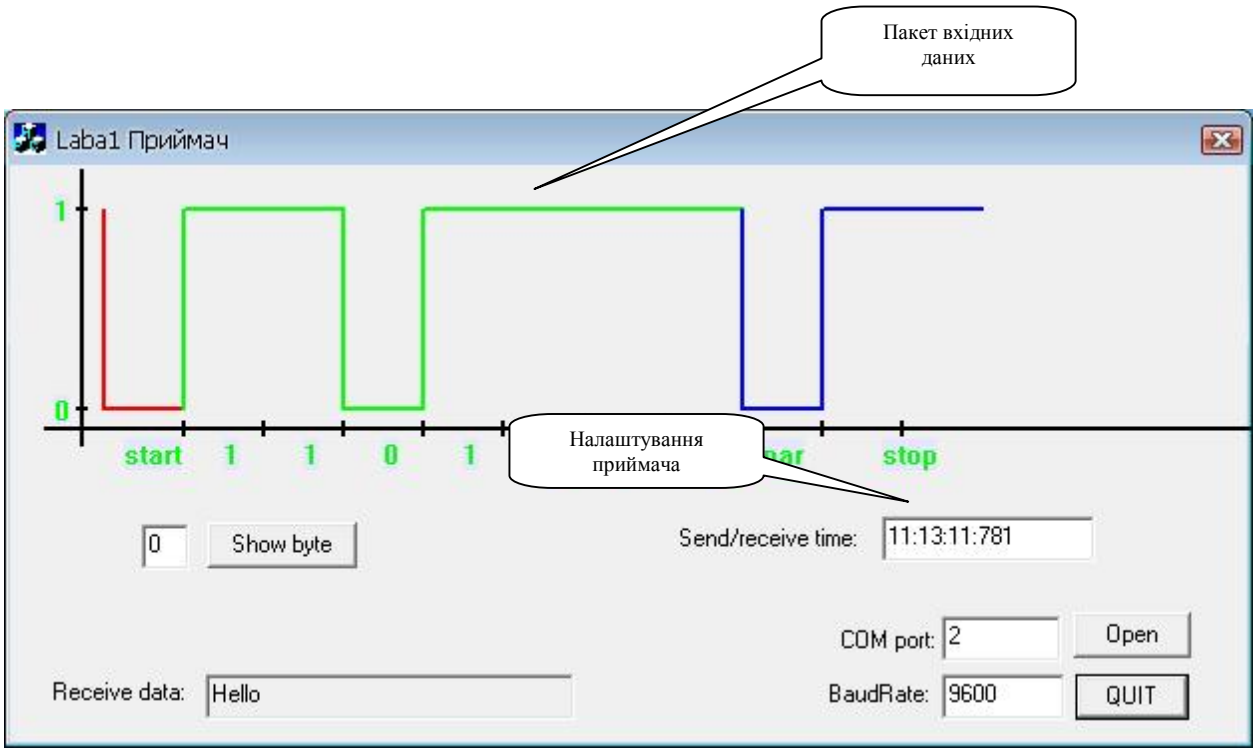


Рис. 25. Вікно програми приймача

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Привести основні характеристики інтерфейсу RS-232C.
2. Привести призначення сигналів інтерфейсу RS-232C.
3. Привести формат даних RS-232C.
4. Пояснити основні принципи розроблення схеми алгоритму обміну заданим повідомленням.
5. Привести основні елементи схеми алгоритму обміну заданим повідомленням.
6. Привести основні етапи відлагодження програми.
7. Пояснити роботу написаних програм.
8. Навести особливості передачі-приймання заданого повідомлення.

ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Короткі теоретичні відомості.
3. Розроблення та опис схеми алгоритму обміну заданим повідомленням.
4. Хід роботи. Описати програму передачі-прийому заданого повідомлення.
5. Висновки.
6. Текст програми (у додатку).
7. Демонстрація на комп’ютері програми для заданого повідомлення.

ПРИКЛАД ВИКОНАННЯ ОСНОВНИХ ЕТАПІВ РОБІТ 1–4

Подається інформація, яка допомагає створити проєкт програми в середовищі Visual C++ 6.0. Студент, за побажанням, може використати інше середовище та реалізувати програмні драйвери на основі власних оригінальних алгоритмів, шо підвищує якість виконання робіт.

Для того щоб створити новий проєкт необхідно запустити на виконання програму MS Visual C++ 6.0 (рис. 26).

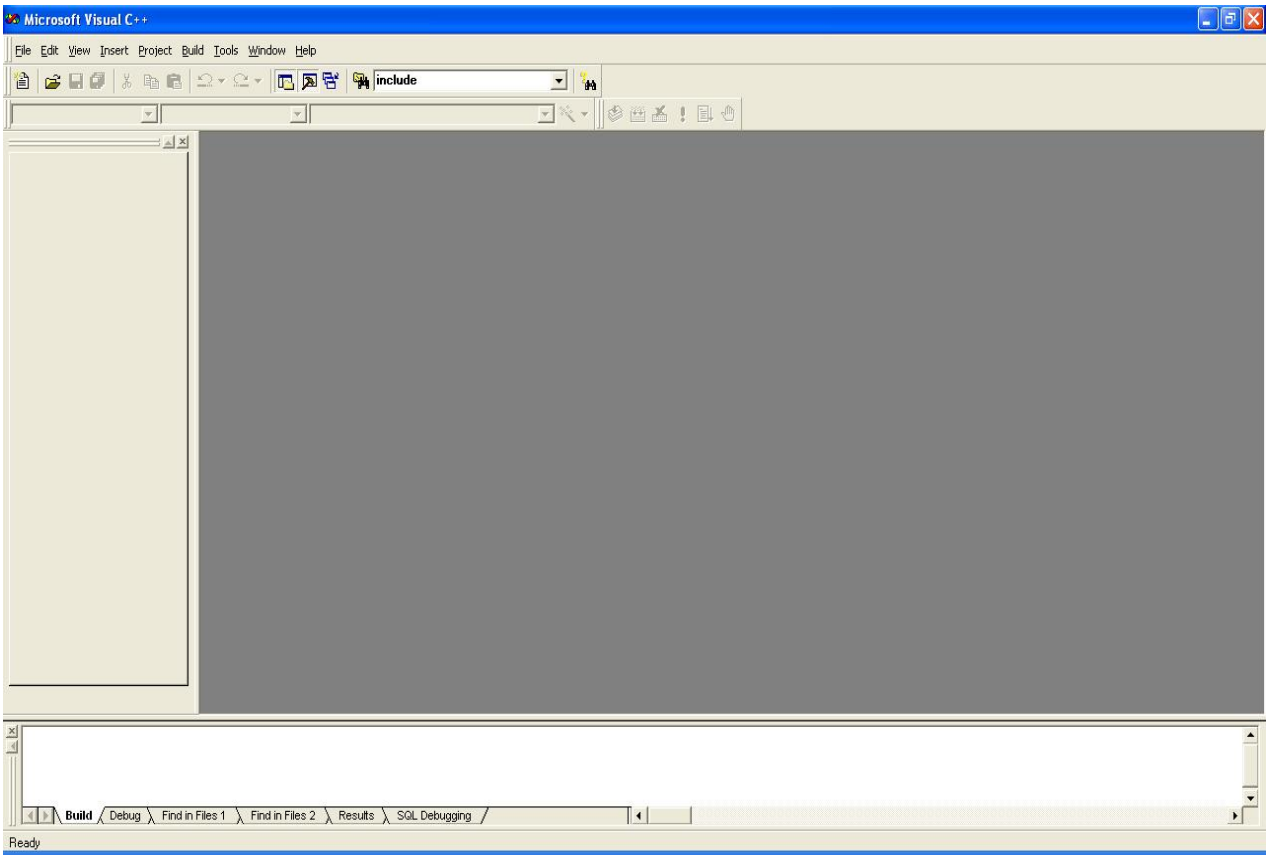


Рис. 26. Загальне вікно середовища програмування MS Visual C++ 6.0.

Далі в меню середовища MS Visual C++ 6.0 вибрати “File” –> “New” (рис. 27).

Необхідно обов’язково позначити тип нового проєкту (MFC AppWizard (exe)), вказати його назву і натиснути “OK”.

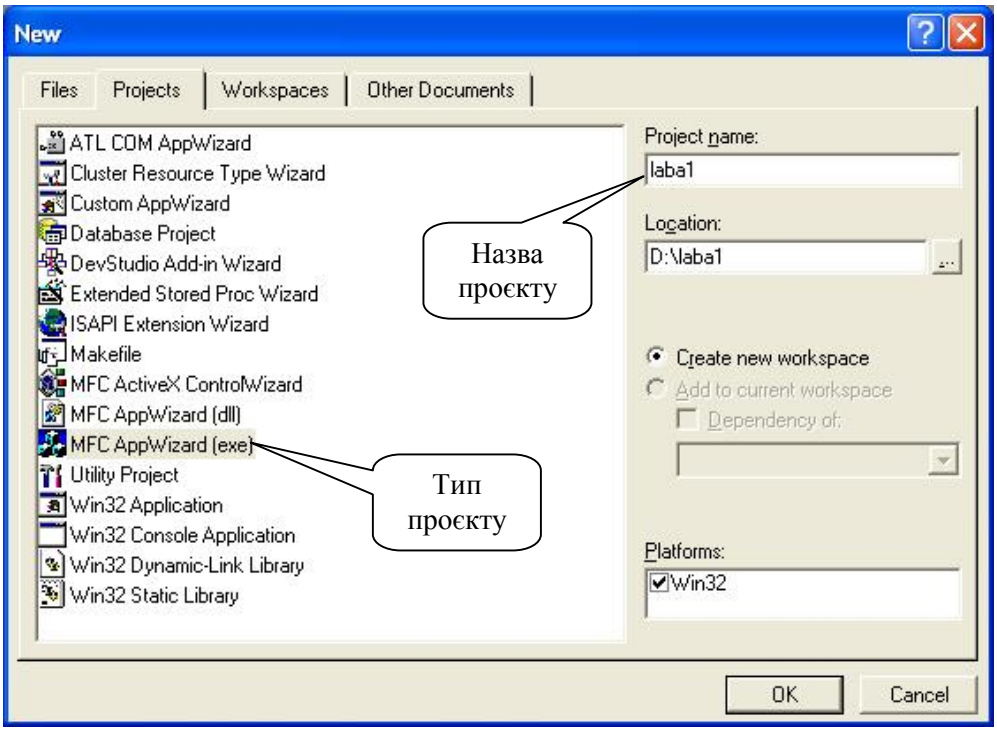


Рис. 27. Вікно створення нового MFC проєкту

Далі треба вибрати вигляд головного вікна нового MFC проєкту (рис. 28) і натиснути “OK”.

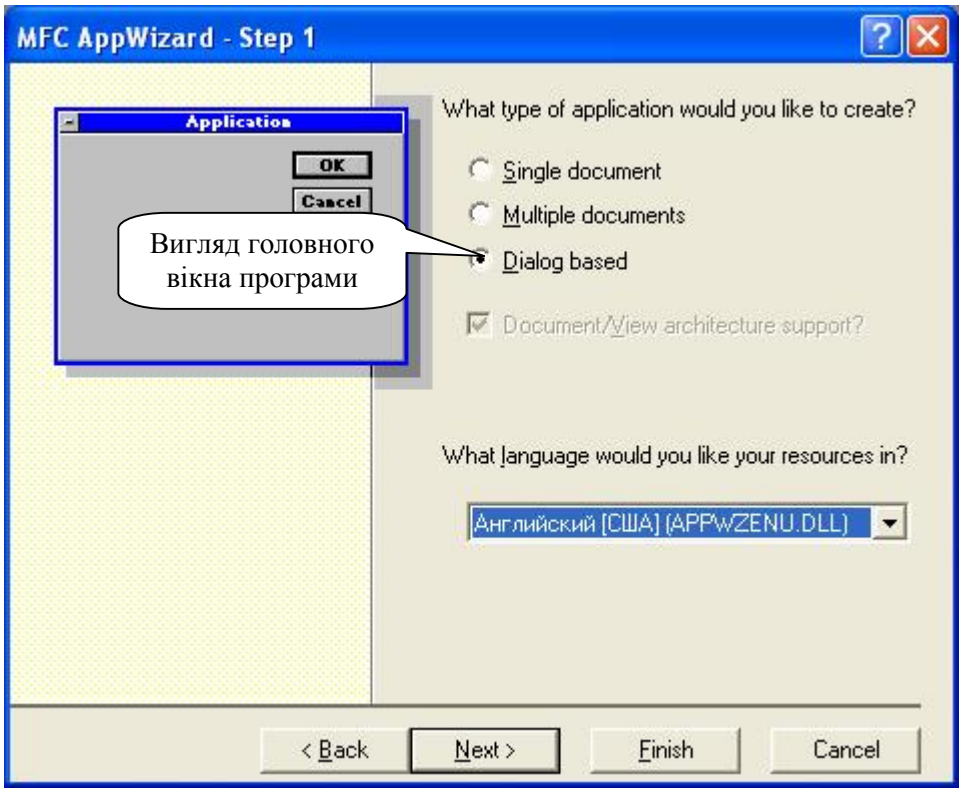


Рис. 28. Вікно вибору вигляду головного вікна нового MFC проєкту.

На наступному вікні треба зняти всі прапорці і ввести заголовок вікна діалогу (рис. 29).

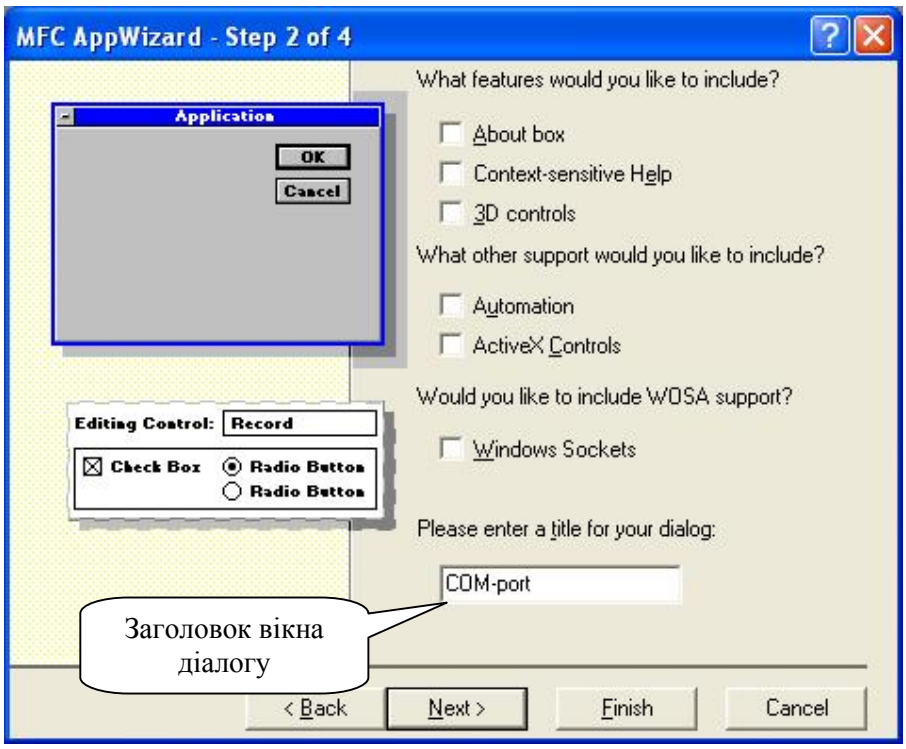


Рис. 29. Вікно вигляду елементів діалогу.

Далі треба натиснути “Next” (рис. 30)

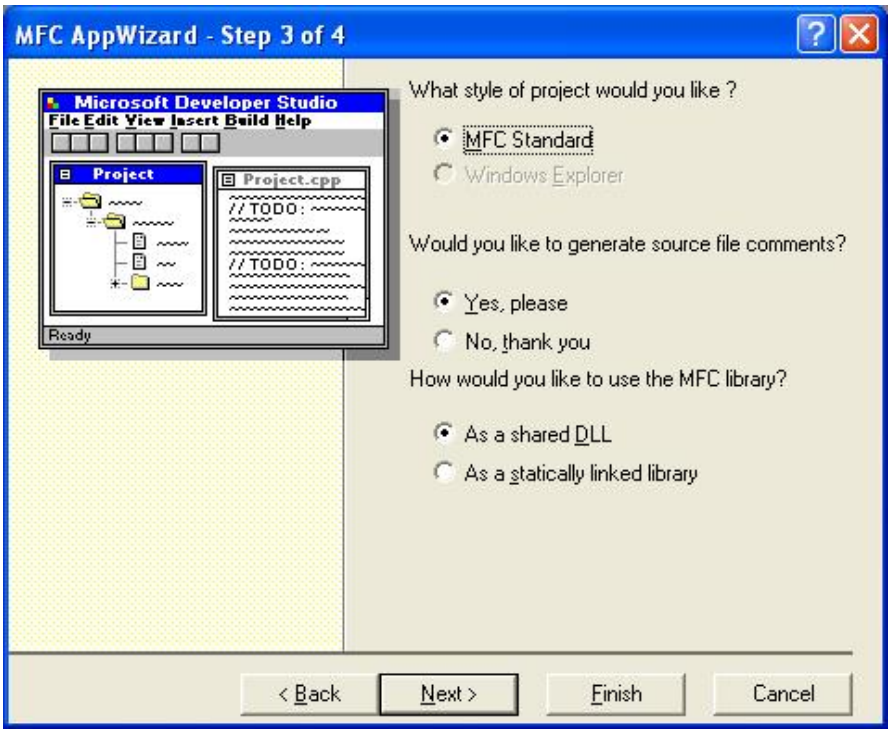


Рис. 30. Вікно вибору властивостей проєкту

Далі треба натиснути “Finish” (рис. 31)

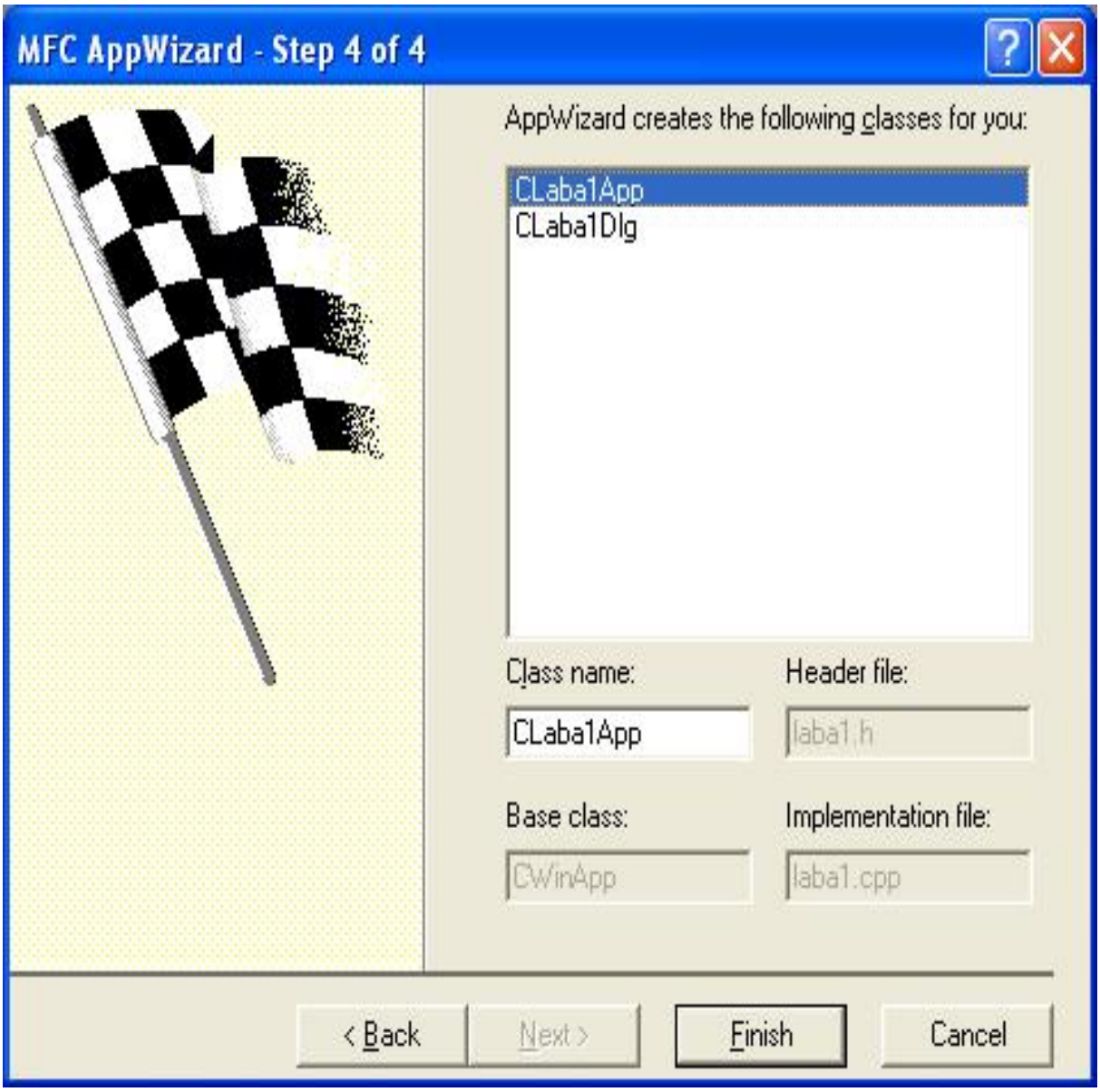


Рис. 31. Вікно вибору властивостей проєкту

Далі прочитати сумарну інформацію про новий MFC проєкт (рис. 32) і, якщо немає помилок, натиснути “OK”. В іншому випадку натиснути “Cancel” і відмінити створення нового проєкту.

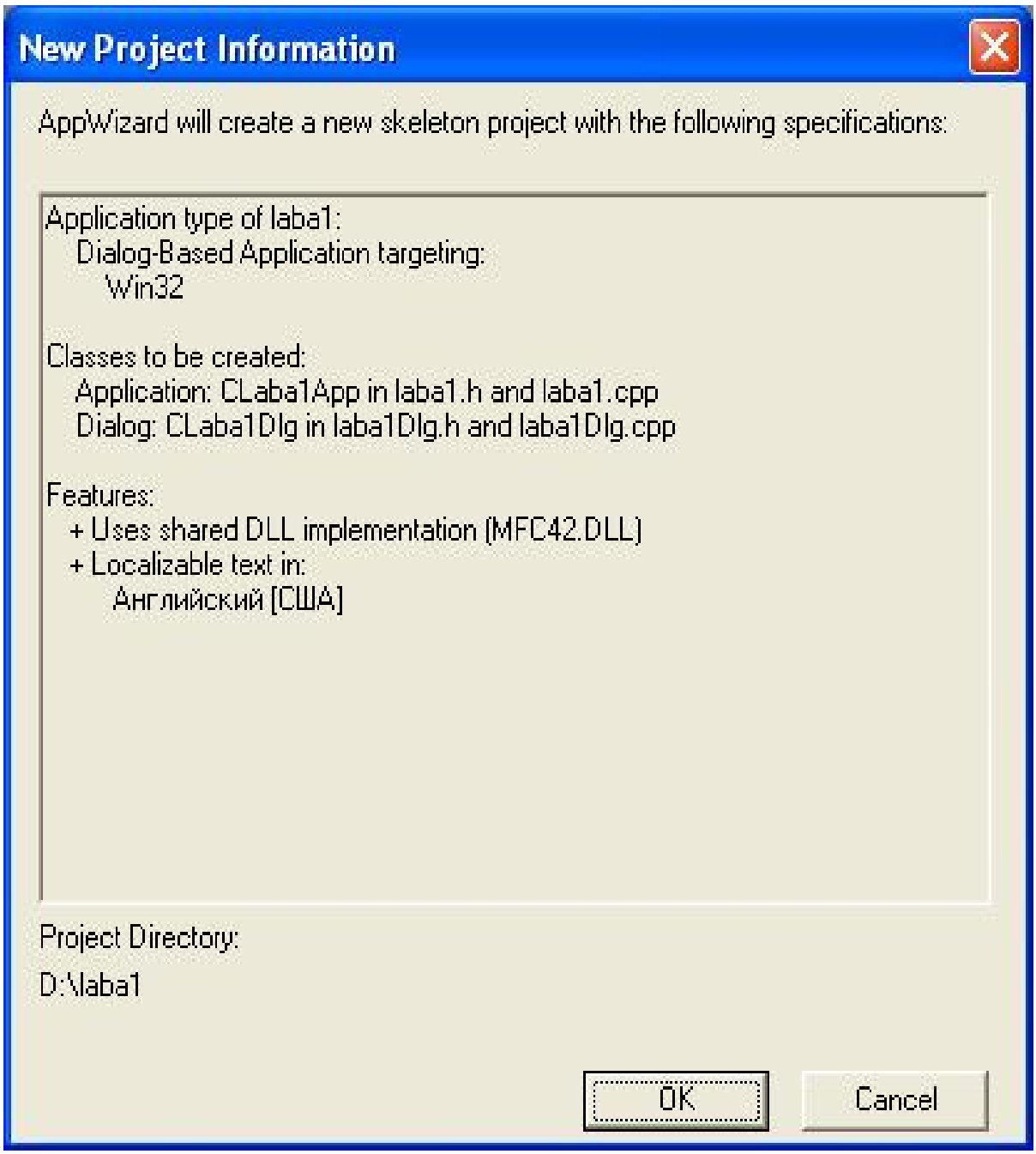


Рис. 32. Вікно сумарної інформації про новий MFC проєкт.

Для того, щоб відкрити існуючий проєкт, необхідно вибрати “Open Workspace…” або “Recent Workspaces” із “File” меню.

Новий MFC проєкт (рис. 33) містить класи, ресурси і файли програми (CTestApp) та вікна діалогу (CTestDlg).

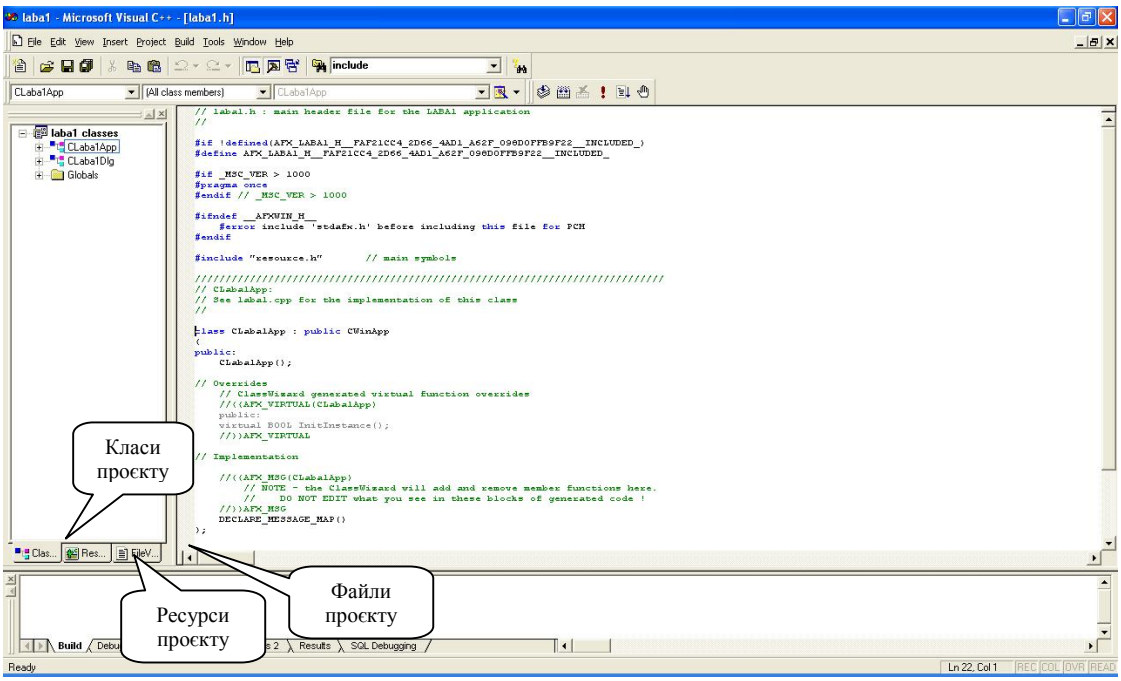


Рис. 33. Вікно середовища MS Visual C++ 6.0 з новим проєктом.

Клас **CLaba1App** містить функцію **InitInstance()**, з якої починається виконання програми і створення об’єкту програми **theApp** (рис. 34). У функції **InitInstance**() створюється об’єкт діалогу, до нього приєднується вікно діалогу і діалог відображається на екрані. Крім того є обробники натискання клавіш “**OK**” і “**Cancel**”. Там можна додати свій зміст.

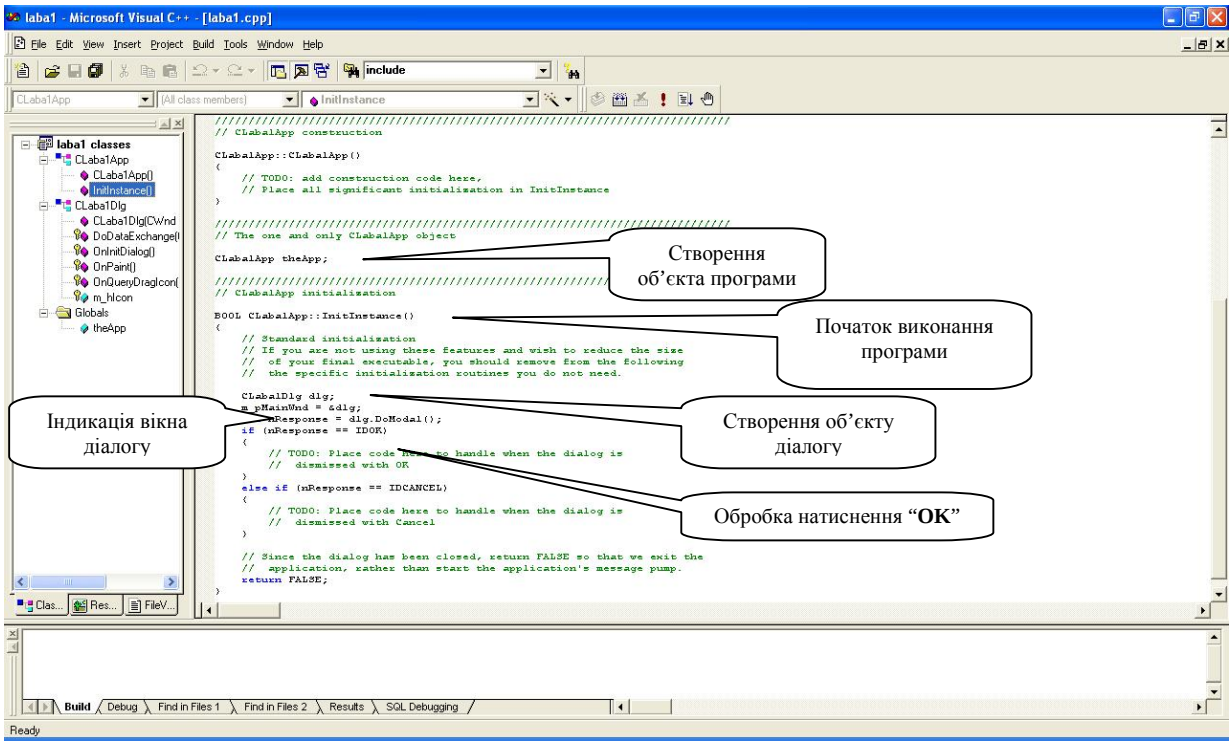


*Рис. 34. Клас **CLaba1App***

Клас **CLaba1Dlg** містить функції **OnInitDialog**() та **OnPaint**() (рис. 35). В функції **OnInitDialog**() налаштовується зовнішній вигляд вікна діалогу, а функція **OnPaint**() здійснює вивід вікна діалогу на екран.

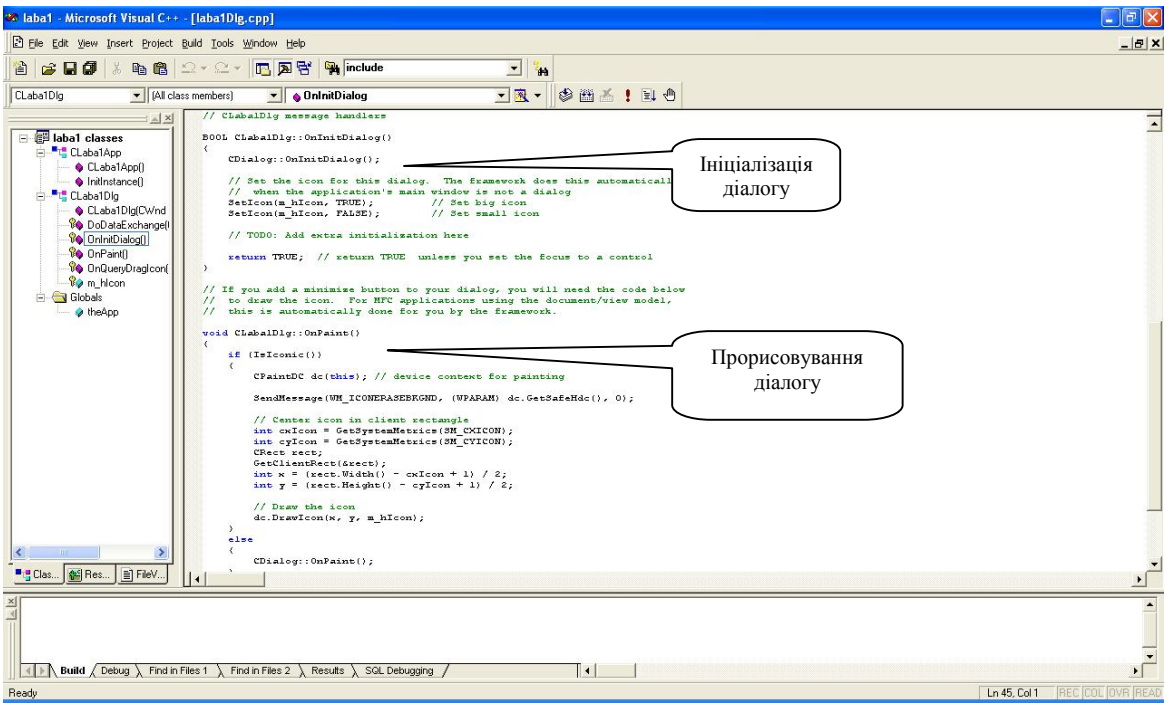


*Рис. 35. Клас **CLaba1Dlg***

Для додавання функціональності до діалогу спочатку треба нарисувати додаткові елементи керування в редакторі діалогів (рис. 36).

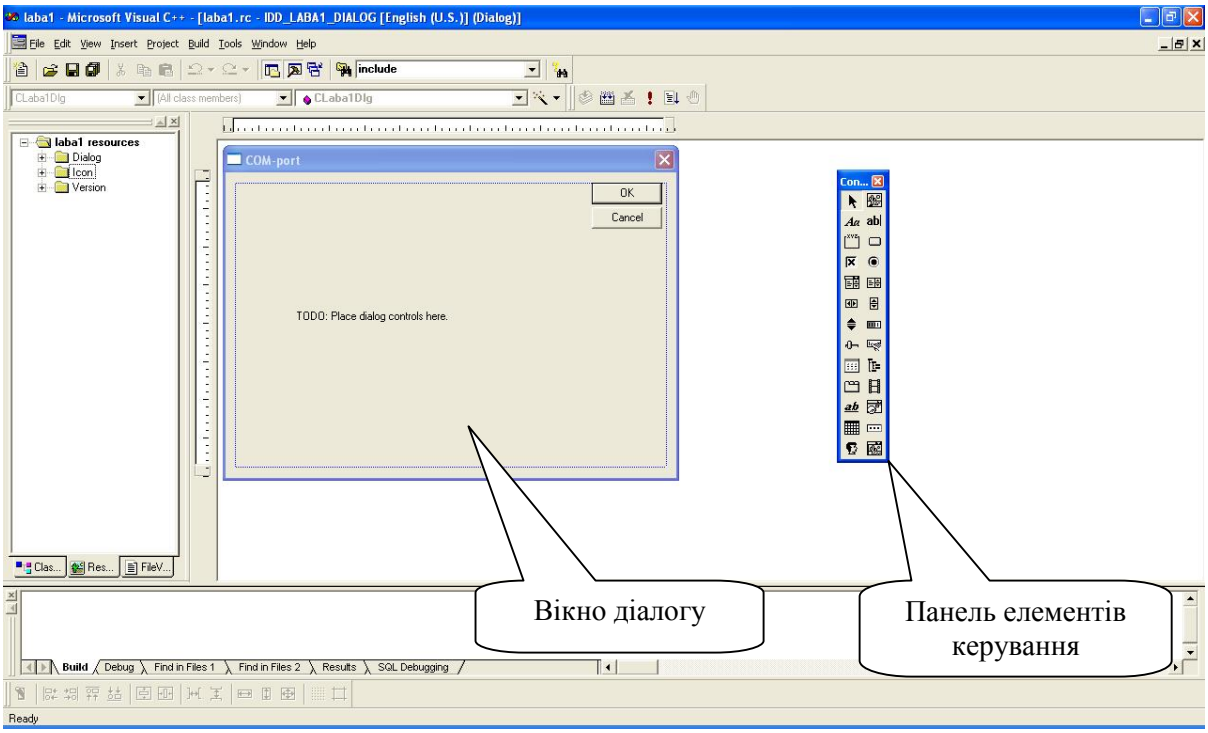


Рис. 36. Редактор діалогу

Для цього на панелі елементів керування вибирається один елемент. На вікні діалогу мишею натягується прямокутник. Після відпускання кнопки миші, на вікні діалогу з'являється зображення елемента керування. Для даної лабораторної роботи необхідно на вікні діалогу встановити об'єкти типу Edit (поле редагування) та Button (кнопка) (рис. 38).

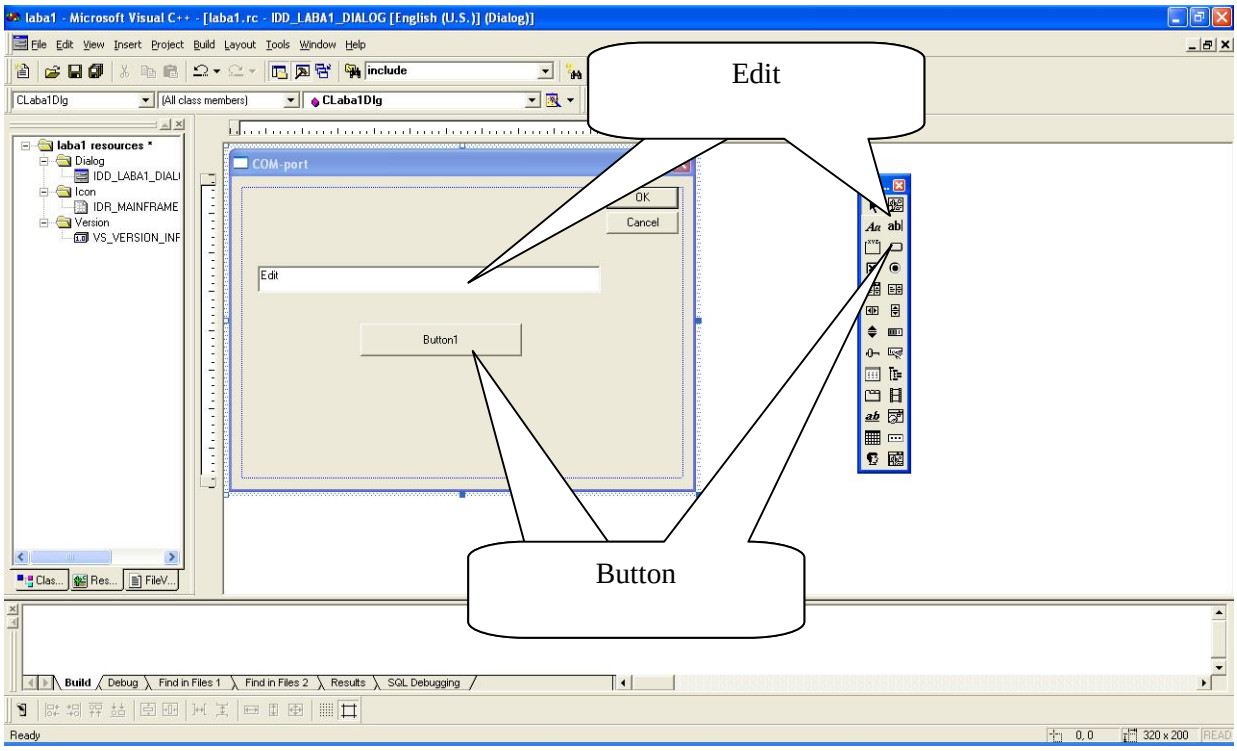


Рис. 37. Розміщення елементів керування

Для зв'язування елемента керування з об'єктом елемента керування треба викликати для нього "майстер" класів (рис. 38). На першій закладці знаходяться елементи карти повідомлень (рис. 39).

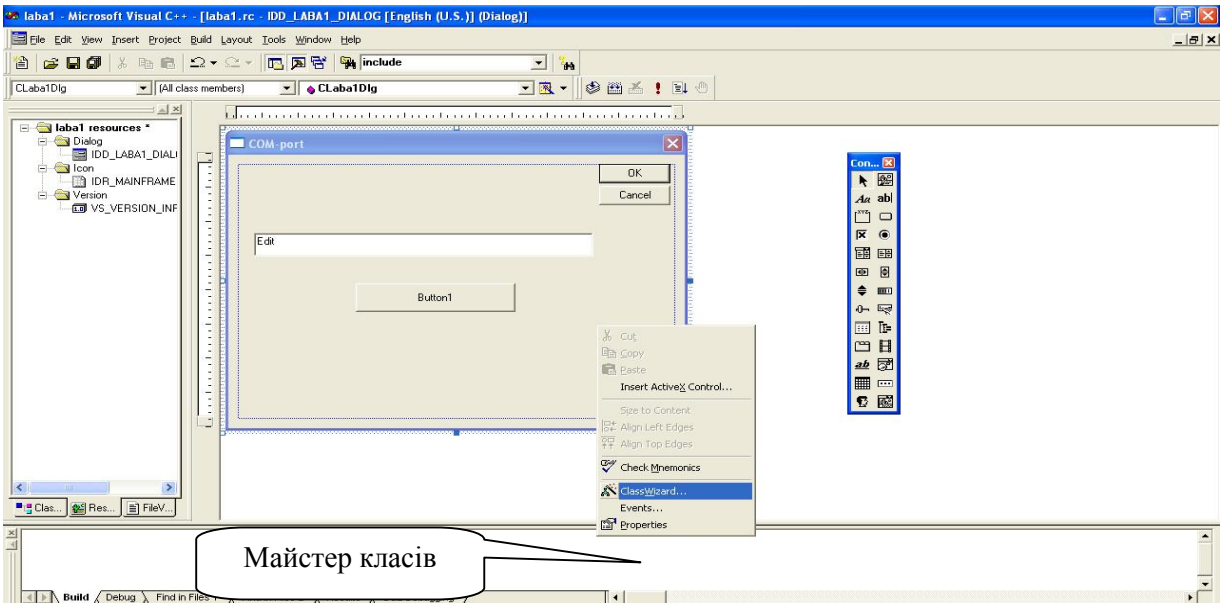


Рис 38. Виклик майстра класів через контекстне меню діалогу

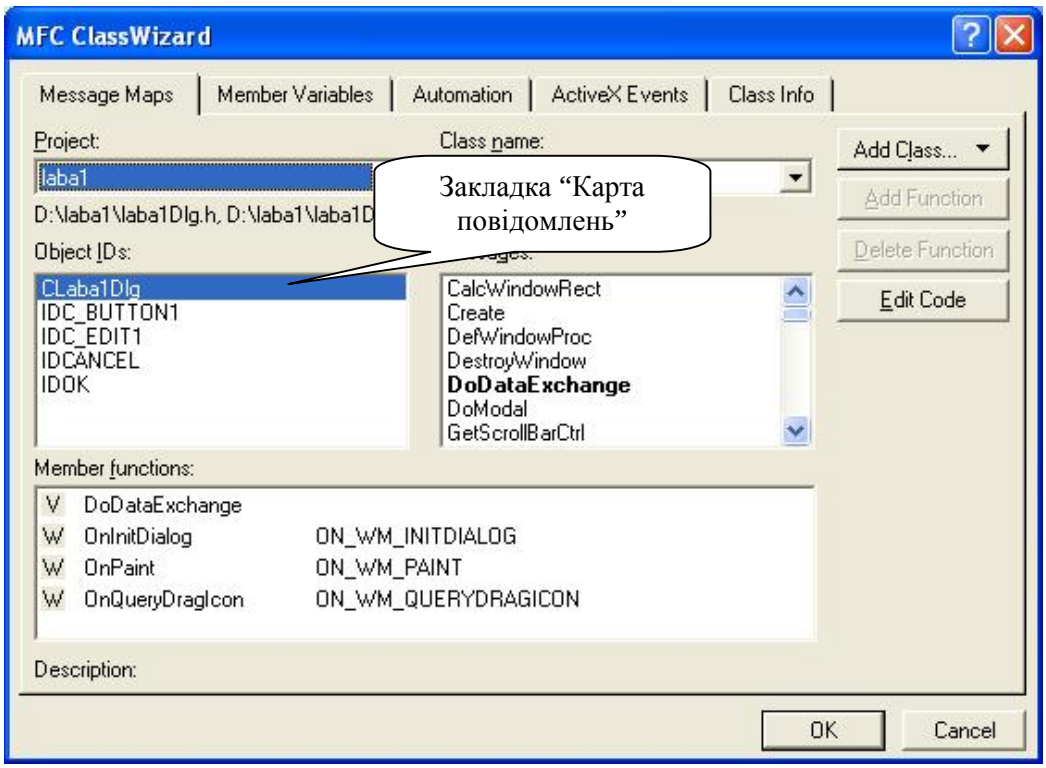


Рис. 39. Вікно "майстра" класів, закладка "Карта повідомлень"

На другій закладці знаходяться змінні класу діалогу (рис. 40).

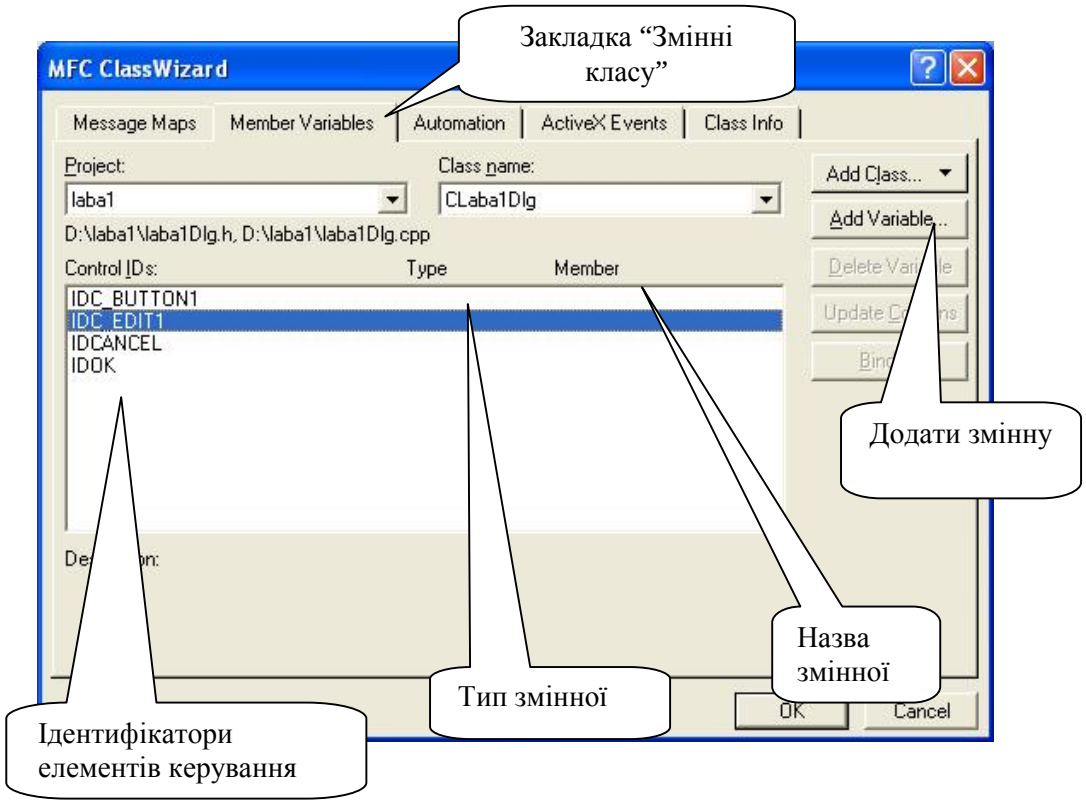


Рис. 40. Вікно "майстра" класів, закладка "Змінні класу".

Для додавання нової змінної треба натиснути “Add Variable…”. З’явиться вікно створення нової змінної класу (рис. 41). У ньому обов’язково треба вказати назву змінної, яка починається на m_, категорію (по значенню або об’єктна) та тип змінної.

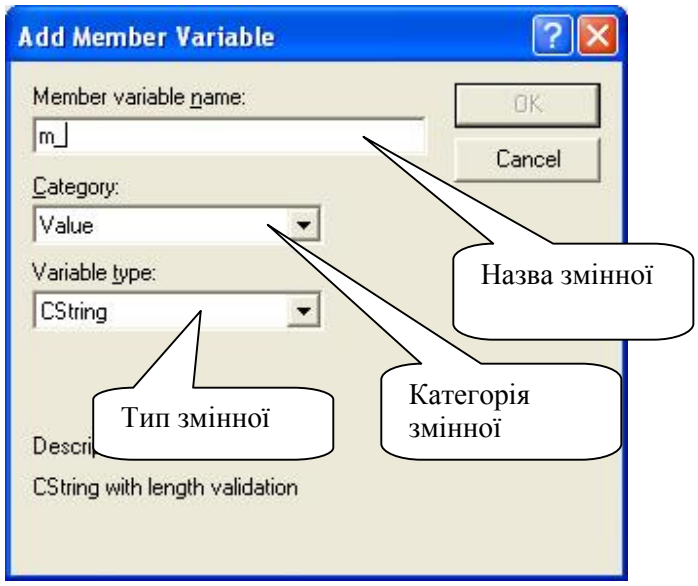


Рис. 41. Вікно створення нової змінної класу

Після створення нової змінної вона з’являється на закладці “Змінні класу” “майстра” класів (рис. 42). Після натиснення “**OK**” можна побачити, що нова змінна з’явилась в середовищі у вікні перегляду класів і в тексті програми (рис. 43).

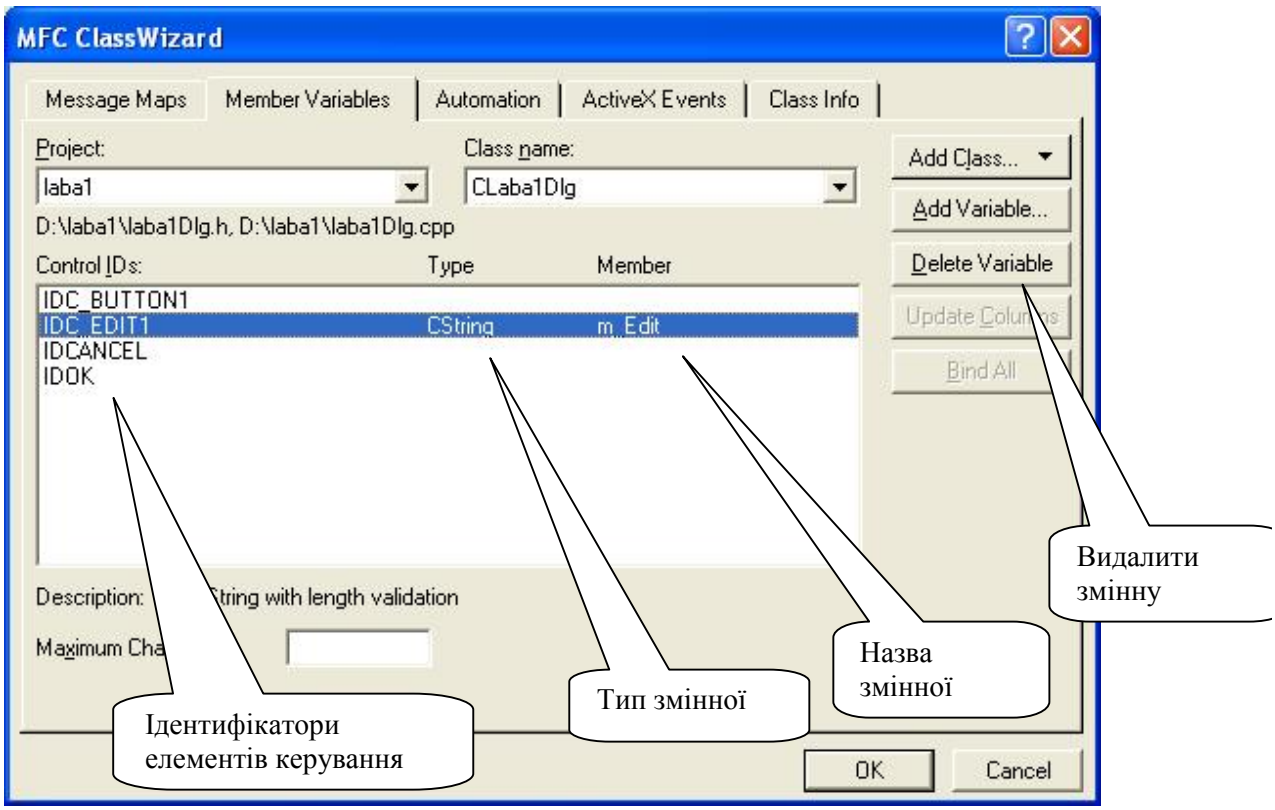


Рис. 42. Вікно “майстра” класів після створення змінної

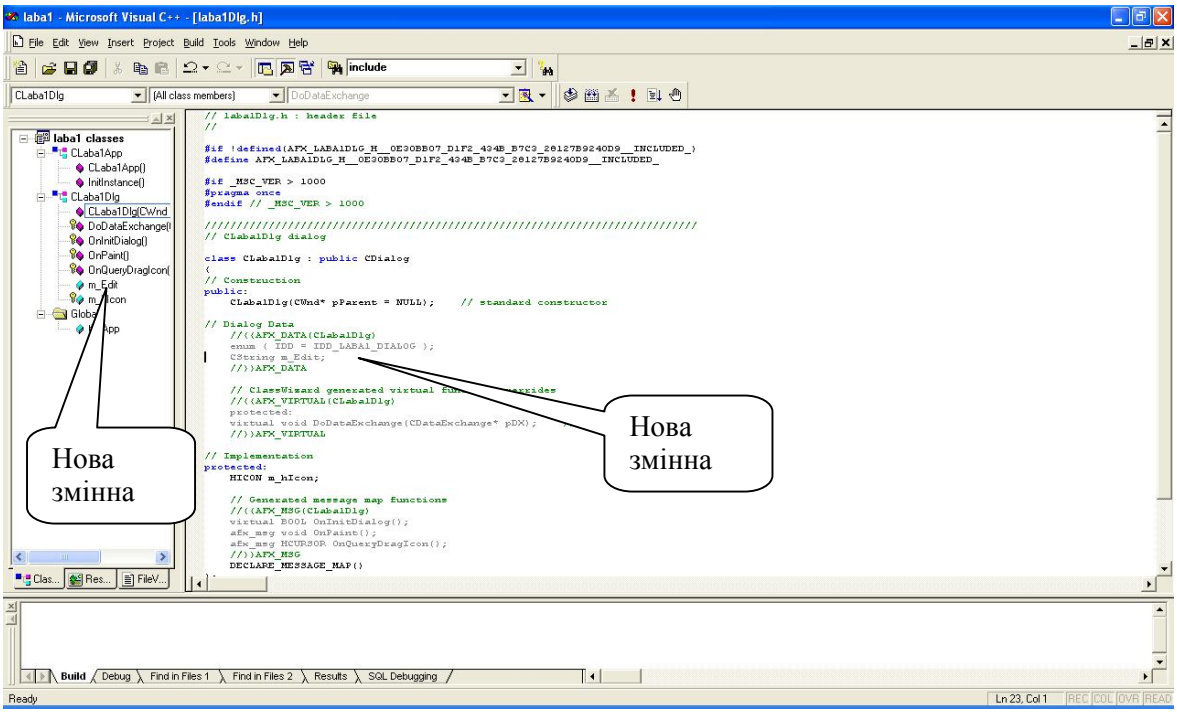


Рис. 43. Вікно середовища з новою змінною

У відповідь на дії користувача в віконній програмі виникають повідомлення. Для обробки повідомлень треба включити повідомлення в карту повідомлень і додати до класу спеціальну функцію – обробник повідомлення. Найбільш коректно ці дії виконує майстер повідомлень (рис. 44). Для додавання нового обробника повідомлень треба вибрати елемент керування, повідомлення якого будуть оброблятися, вибрати повідомлення, для якого потрібен обробник, і натиснути клавішу “Add Function…”.

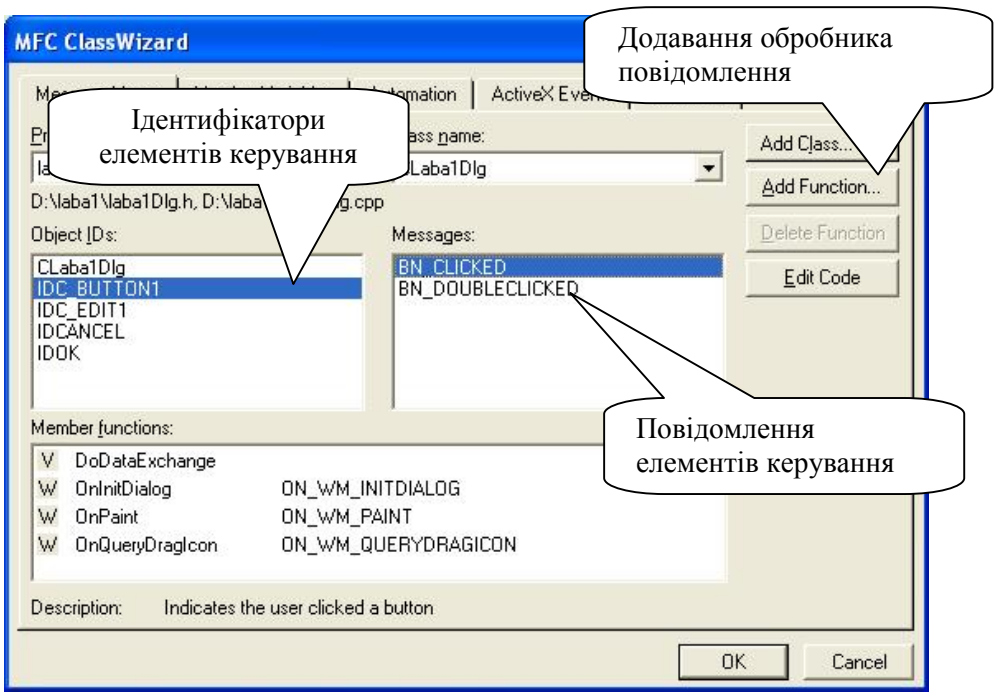


Рис. 44. Вікно “майстра” класів при створенні обробника повідомлення

У вікні додавання нового обробника треба ввести назву функції-обробника (рис. 45), яка обов’язково повинна починатися з On, і натиснути “**OK**”.

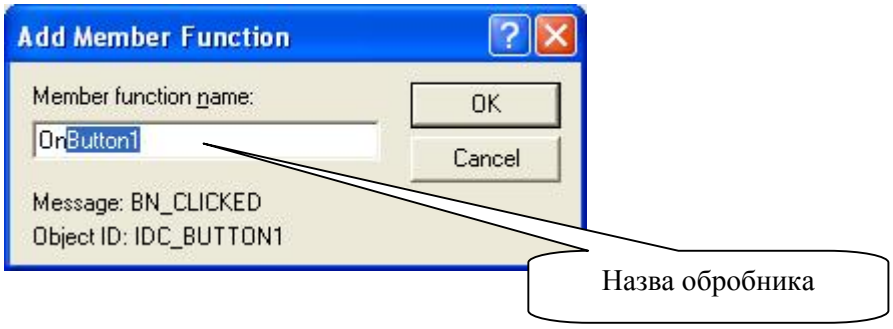


Рис. 45. Вікно додавання обробника повідомлення

У нижній частині вікна “майстра” класів з’явиться нова функція – член класу і повідомлення, яке ця функція обробляє (рис. 46).

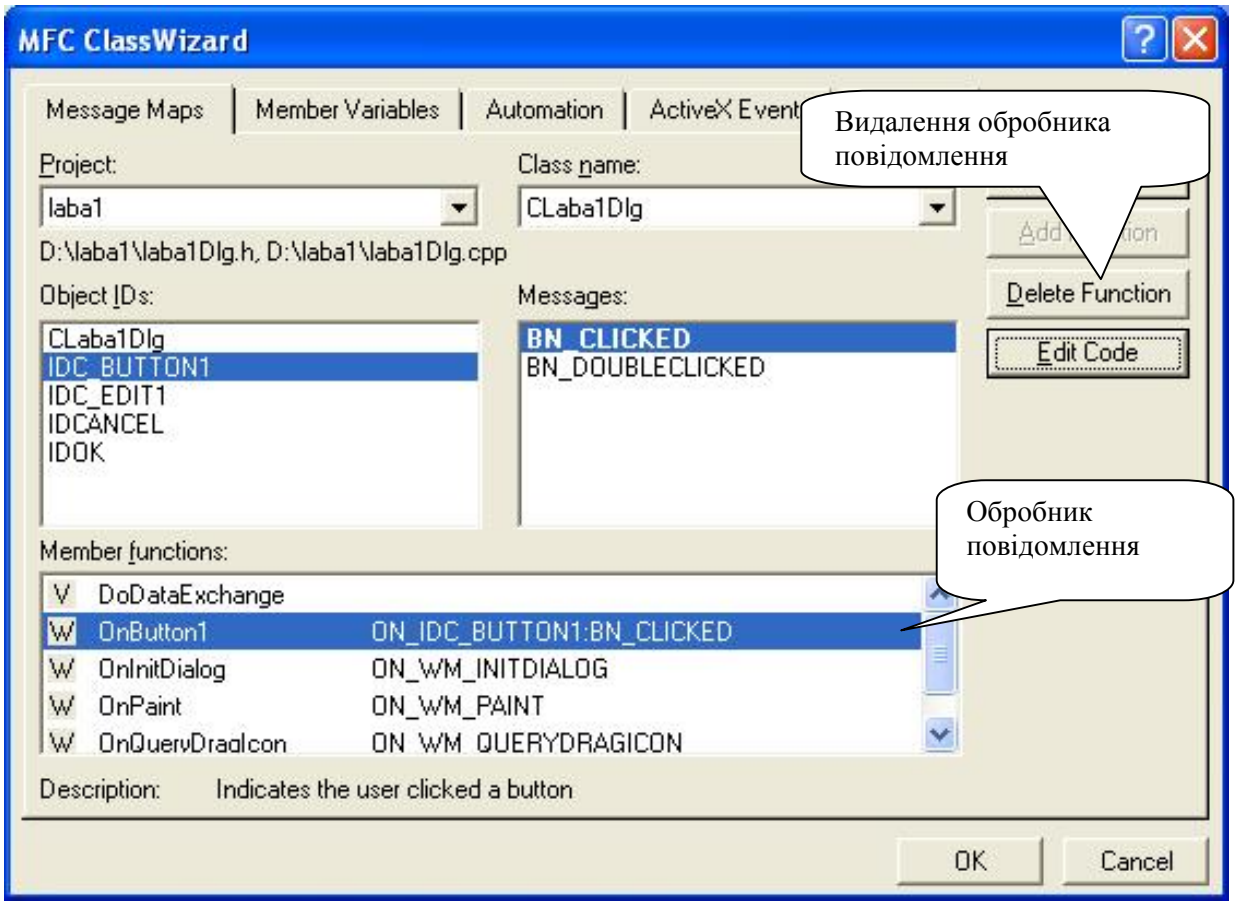


Рис. 46. Вікно “майстра” класів після створення обробника повідомлення

Після натиснення “**OK**” можна побачити, що нова функція з’явилася в середовищі у вікні перегляду класів і в тексті програми (рис. 47).

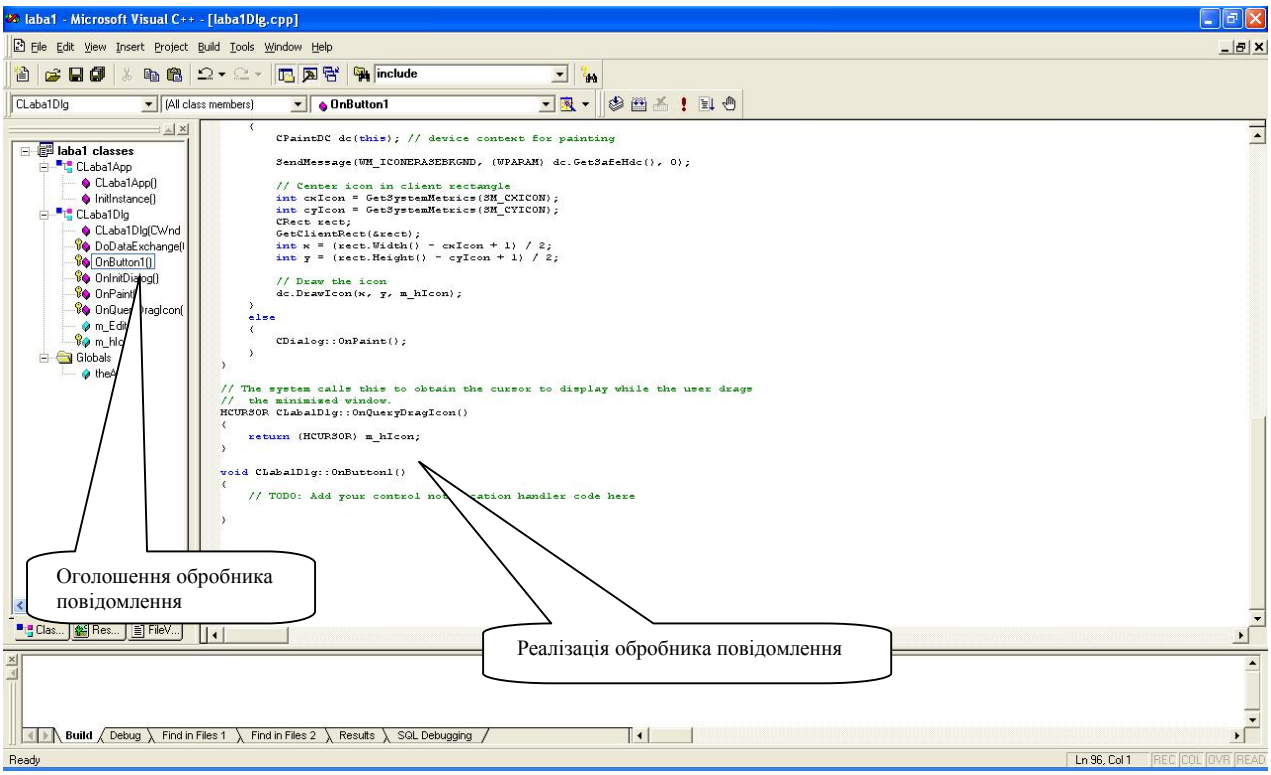


Рис. 47. Вікно середовища з новим обробником повідомлення

Якщо запустити програму на компіляцію і виконання, то вона створить вікно діалогу з елементами керування (рис. 48).

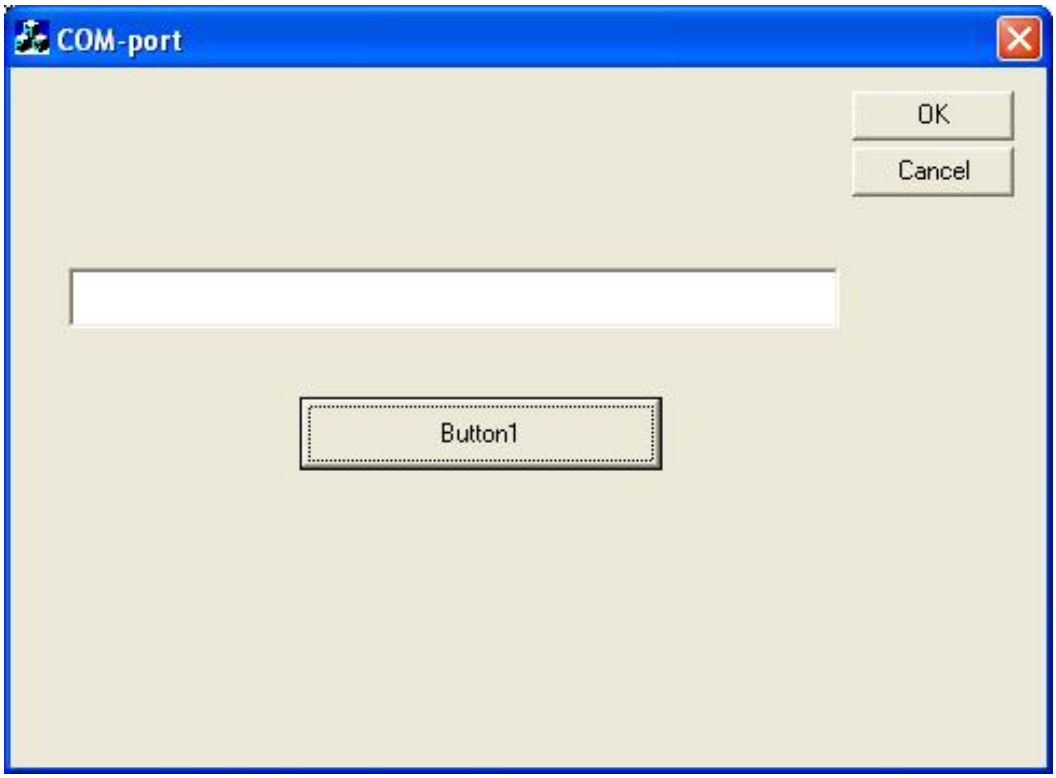


Рис. 48. Вікно програми після виконання

Елемент Edit можна використовувати для введення інформації на передавання в програмі-передавачі і для приймання інформації в програмі-приймачі. Для цього потрібно в обробнику повідомлення натиснення кнопки Button1 написати реалізацію програми передавання даних з рядка редагування Edit (використовуючи вже створену змінну m_Edit) в COM-порт в програмі-передавачі і так само в програмі-приймачі написати реалізацію програми відображення даних, прийнятих з COM-порту, в рядку редагування Edit.

Розглянемо приклад створення програми-передавача даних. У вікні діалогу потрібно змінити напис на кнопці Button1. Для цього у властивостях даного елементу керування змінюємо поле Caption з “Button1” на “Передача даних” (рис. 49 і рис. 50).

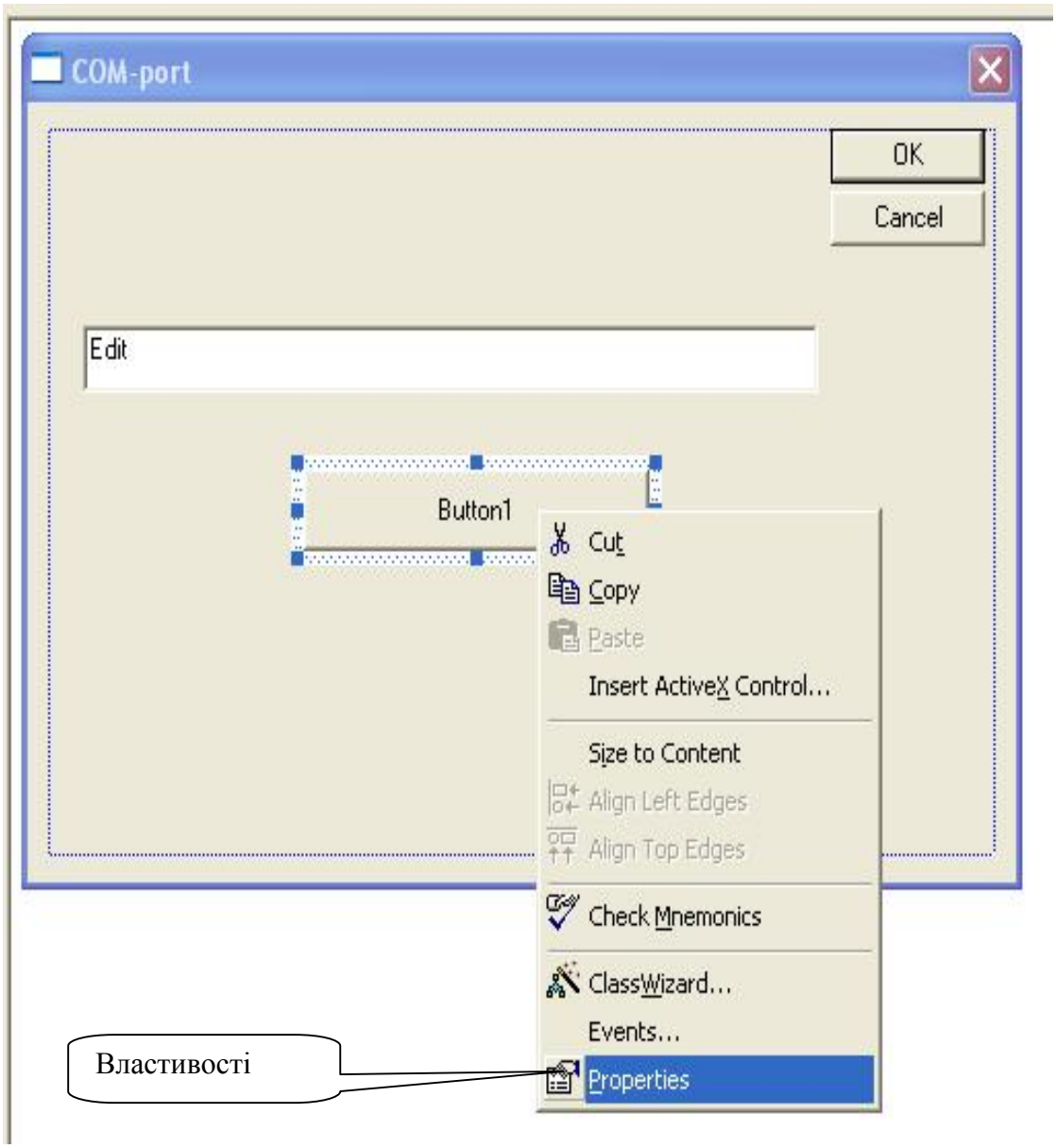


Рис 49. Контекстне меню елемента Button1

Рис 50. Закладка General властивостей елемента Button1

У результаті вікно діалогу має вигляд (рис. 51):

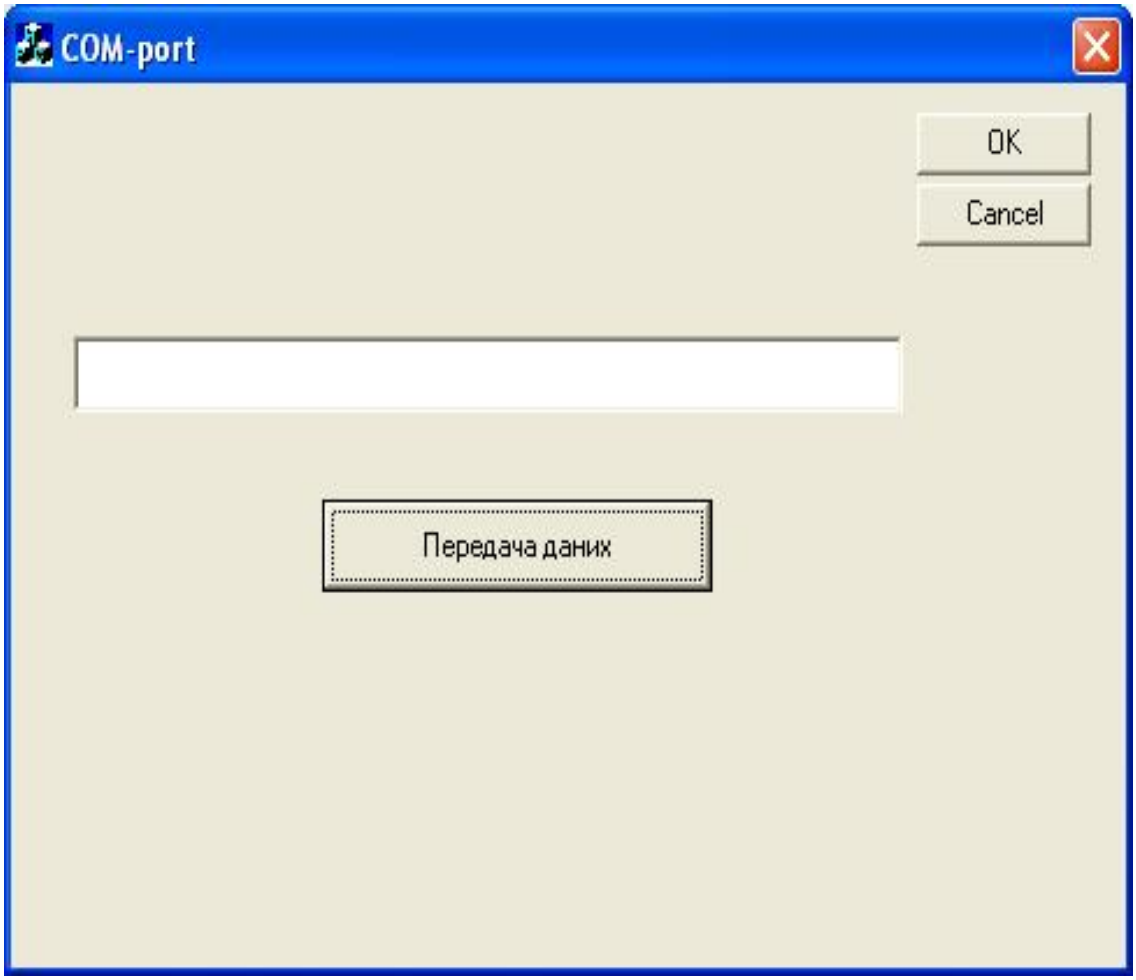


Рис 51. Вікно програми

Тепер залишається тільки в обробнику повідомлення натиснення кнопки “Передача даних” (Button1) написати функції, що будуть здійснювати передачу інформації з елемента Edit на COM-порт.

Нижче наводиться приклад програми передачі даних на COM-порт.

Дана програма містить нові функції (Open(),WriteString(string Str), Close()). Їх в проєкт можна додати наступним чином:

У вікні перегляду класів середовища правою кнопкою миші клікнути по назві класу і вибрати Add Member Function (рис. 52).

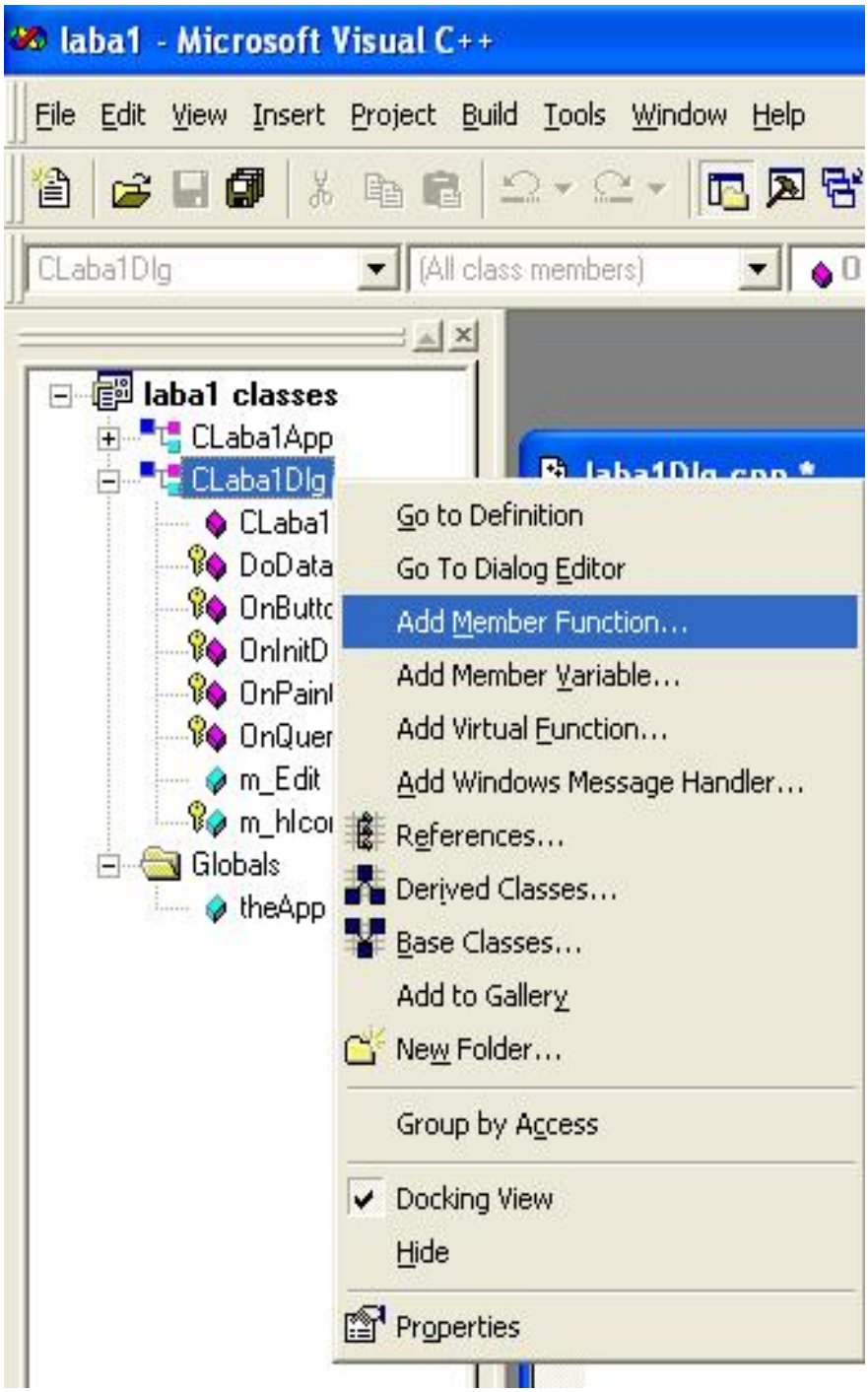


Рис. 52. Створення нової функції

У вікні створення нової функції дати назву фукції та її тип (рис. 53).

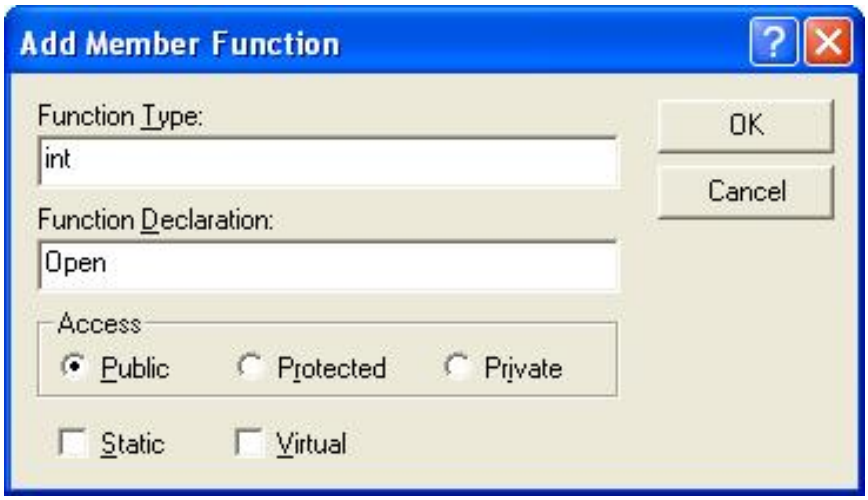


Рис. 53. Вікно створення функції

У результаті отримуємо функцію int CLaba1Dlg:Open(). Аналогічно створюються решта функцій.

Приклад програмНИХ ДРАЙВЕРІВ ДЛЯ робіт 1–4

```
// laba1Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "laba1.h"
#include "laba1Dlg.h"
#include "ComPort.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLaba1Dlg dialog

CLaba1Dlg::CLaba1Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CLaba1Dlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CLaba1Dlg)
	m_Edit = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CLaba1Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CLaba1Dlg)
	DDX_Text(pDX, IDC_EDIT1, m_Edit);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CLaba1Dlg, CDialog)
	//{{AFX_MSG_MAP(CLaba1Dlg)
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

```
/////////////////////////////////////////////////////////////////////////////
// CLaba1Dlg message handlers

BOOL CLaba1Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CLaba1Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
```

```
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CLaba1Dlg::OnQueryDragIcon()
{
        return (HCURSOR) m_hIcon;
}

int CLaba1Dlg::WriteString(string Str)
{
   PurgeComm(hPort,PURGE_TXCLEAR | PURGE_RXCLEAR);
   Str = Str + "\r";
   DWORD dwError , dwNumBytesWritten;
   WriteFile(
      hPort,
      Str.c_str(),
      Str.length(),
      &dwNumBytesWritten,
      NULL);
   return (int)dwNumBytesWritten;
}

int CLaba1Dlg ::Open()
{
   hPort = CreateFile(
      ComPortName.c_str(),
      GENERIC_READ | GENERIC_WRITE,
      0,
      NULL,
      OPEN_EXISTING,
      0,
      NULL
      );
   if(hPort == INVALID_HANDLE_VALUE)
   {
      return 0;
   }
   DCB PortDCB;

   PortDCB.DCBlength = sizeof(DCB);


   GetCommState(hPort , &PortDCB);
```

```
    PortDCB.BaudRate = 9600;
    PortDCB.fBinary = true;  //!
    PortDCB.fParity = true;
    PortDCB.fOutxCtsFlow = false;
    PortDCB.fOutxDsrFlow = false;
    PortDCB.fDtrControl = DTR_CONTROL_ENABLE;

    PortDCB.fDsrSensitivity = false;
    PortDCB.fTXContinueOnXoff = false;
    PortDCB.fOutX = false;
    PortDCB.fInX = false;
    PortDCB.fErrorChar = true;
    PortDCB.fNull = false;
    PortDCB.fRtsControl = RTS_CONTROL_ENABLE;

    PortDCB.fAbortOnError = false;

    PortDCB.ByteSize = 8;
    PortDCB.Parity = NOPARITY;
    PortDCB.StopBits = ONESTOPBIT;
//    PortDCB.EofChar = '!';

    if(!SetCommState(hPort , &PortDCB))
    {
       return 0;
//       ShowMessage("Unable to configure the serial port");
    }

    COMMTIMEOUTS CommTimeouts;

    GetCommTimeouts(hPort,&CommTimeouts);



    CommTimeouts.ReadIntervalTimeout = 50;
    CommTimeouts.ReadTotalTimeoutMultiplier = 1;
    CommTimeouts.ReadTotalTimeoutConstant = 50;
    CommTimeouts.WriteTotalTimeoutMultiplier = 1;
    CommTimeouts.WriteTotalTimeoutConstant = 50;

    SetCommTimeouts(hPort,&CommTimeouts);
    FlushFileBuffers(hPort);
    Opened = true;
    return 1;
}
```

```
int CLaba1Dlg ::Close()
{
   if(Opened)
   {
      CloseHandle(hPort);
      Opened = false;
      return 1;
   }
   else
      return 0;
}

void CLaba1Dlg::OnButton1()
{
         // TODO: Add your control notification handler code here
         Open();
         WriteString(m_Edit.LoadString);
        Close();
}
```

ТЕОРЕТИЧНІ ВІДОМОСТІ ДО ЛАБОРАТОРНИХ РОБІТ № 5–8

Лабораторні роботи № 5–8 призначені для поглибленого вивчення особливостей, характеристик інтерфейсу універсальної послідовної шини, одного із найбільш поширених периферійних інтерфейсів, та отримання навиків розроблення для цього інтерфейсу програмних драйверів, які є складовими компонентами багатьох комп'ютерних систем.

Універсальна послідовна шина УПШ (англ. USB - Universal Serial Bus, абревіатура читається ю-ес-бі) призначена для з'єднання периферійних пристроїв у периферійну підсистему комп'ютера. Шина USB представляє собою послідовний інтерфейс передачі даних для середньошвидкісних та низькошвидкісних периферійних пристроїв. Для високошвидкісних пристроїв на сьогодні кращим вважається FireWire. USB-кабель представляє собою дві виті пари: по одній парі відбувається передача даних в кожному напрямку (диференціальне включення), а інша пара використовується для живлення периферійного пристрою (+5 В, +3,3В). Завдяки вбудованим лініям живлення, що запезпечують струм до 500 мА, USB часто дозволяє використовувати пристрої без власного блоку живлення (якщо ці пристрої споживають струм потужністю не більше 500 мА). До одного контролера шини USB можна під'єднати до 127 пристроїв через ланцюжок концентраторів (вони використовують топологію «зірка»). На відміну від багатьох інших стандартних з'єднувачів, для USB з'єднувачів характерні довговічність та механічна міцність. Інтерфейс USB є послідовною, напівдуплексною, двонаправленою шиною. Шина дозволяє підключити до ПК до 127 фізичних пристроїв. Кожен фізичний пристрій може, у свою чергу, складатися з декількох логічних (наприклад, клавіатура з вбудованим манипулятором-трекболом). Кабельна розводка USB починається з вузла головного контролера (хост-host). Хост володіє інтегрованим кореневим концентратором (root hub), який надає декілька роз'ємів USB для підключення зовнішніх пристроїв. Потім кабелі йдуть до інших пристроїв USB, які також можуть бути концентраторами, і функціональних компонентів (наприклад, модем або акустична система). Концентратори часто вбудовуються в монітори і клавіатури (які є типовими складеними пристроями). Концентратори можуть містити до семи "витікаючих" портів. Для передачі сигналів шина USB використовує чотирипровідний інтерфейс. Одна пара провідників ("+5В" чи "+3,3В" і "загальний") призначена для живлення периферійних пристроїв з навантаженням до 500 мА. Дані передаються по іншій парі ("D+" "D-"). Для передачі даних використовується диференціальна напруга до 3 В (з метою зниження впливу шуму) і схема кодування NRZI (що позбавляє від необхідності виділяти додаткову пару провідників під тактовий сигнал). Всі концентратори повинні підтримувати на своїх витікаючих портах пристрої обох типів, не дозволяючи високошвидкісному трафіку досягати низькошвидкісних пристроїв. Високопродуктивні пристрої підключаються за допомогою екранованого кабелю, довжина якого не повинна перевищувати 3 м. Якщо ж пристрій не формулює особливих вимог до смуги пропускання, його можна підключити і неекранованим кабелем (який може бути тоншим і гнучкішим). Максимальна довжина кабелю для низькошвидкісних пристроїв – 5 м. Вимоги пристрою до живлення (діаметр провідників, споживана

потужність) можуть зумовити необхідність використання кабелю меншої довжини. Із-за особливостей розповсюдження сигналу по кабелю число послідовно сполучених концентраторів обмежене шістьма і сімома п'ятиметровими відрізками кабелю. Хост дізнається про підключення або відключення пристрою із повідомлення від концентратора (ця процедура називається опитуванням шини - bus enumeration). Потім хост привласнює пристрою унікальну адресу USB (1:127). Після відключення пристрою від шини USB його адреса стає доступною для інших пристроїв. Для індивідуального звернення до конкретних функціональних можливостей складеного пристрою застосовується 4-бітове поле кінцевої крапки. У низькошвидкісних пристроях за кожною функцією закріплюється не більше двох адрес кінцевих крапок: нульова кінцева крапка використовується для конфігурації і визначення стану USB, а також управління функціональним компонентом; а інша крапка - відповідно до функціональних можливостей компоненту. Пристрої з максимальною продуктивністю можуть підтримувати до 16 кінцевих крапок, резервуючи нульову крапку для завдань конфігурації і управління USB. Хост опитує всі пристрої і видає їм дозволу на передачу даних (розсилаючи для цього пакет-маркер - Token Packet). Таким чином, пристрої позбавлені можливості безпосереднього обміну даними - всі дані проходять через хост. Ця умова сильно заважала впровадженню інтерфейсу USB на ринок портативних пристроїв. У результаті в кінці 2001 року було прийнято доповнення до стандарту USB 2.0 - специфікація USB OTG (On-The-Go), призначена для з'єднання периферійних USB-пристроїв один з одним без необхідності підключення до хосту (наприклад, цифрова камера і фотопринтер). Пристрій, підтримуючий USB OTG, здатний частково виконувати функції хоста і розпізнавати, коли він підключений до повноцінного хосту (на основі ПК), чи до іншого периферійного пристрою. Специфікація описує також протокол узгодження вибору ролі хоста при з'єднанні двох USB OTG-пристроїв.

Дані на шині передаються транзакціями, інтервал між якими складає 1 мс. Передбачено чотири типи транзакцій. Передачі, що управляють, використовуються для конфігурації заново підключених пристроїв (наприклад, привласнення ним адреси USB) та їх компонентів. Пристрої з максимальною продуктивністю можуть бути налаштовані на роботу з конфігураційними повідомленнями завдовжки 8, 16, 32 або 64 байти (за умовчанням - 8 байт). Пристрої з низькою продуктивністю в змозі розпізнавати повідомлення, що управляють, завдовжки не більше 8 байт. Групова передача (bulk) використовується для адресної пересилки даних великого об'сягу (до 1023 байт). Як приклад можна привести передачу даних на принтер або від сканера. Пристрої з низькою продуктивністю не підтримують цей режим. Передача даних переривання, наприклад, введених з клавіатури даних або відомостей про переміщення миші. Ці дані мають бути передані достатньо швидко для того, щоб користувач не відмітив ніякої затримки. Відповідно до специфікацій час затримки USB складає декілька мілісекунд. Можливі ізохронні передачі (передачі в реальному масштабі часу). Пропускна спроможність і затримка доставки обмовляються до початку передачі даних. До ізохронних даних алгоритми корекції помилок непридатні (оскільки час на повторну їх ретрансляцію перевищує допустимий інтервал затримки). За один сеанс в такому режимі може бути передано до 1023 байт. Пристрої з низькою про-

дуктивністю не підтримують цей режим. Варто також відзначити, що різними виробниками пропонувалися специфікації, що описують інтерфейс різних апаратних реалізацій контроллера USB. Фірмою Intel була запропонована специфікація UHCI (Universal Host Controller Interface), яка передбачає надзвичайно просту апаратну реалізацію контроллера USB. У межах цієї специфікації основні функції контролю і арбітражу шини покладаються на програмний драйвер. Альтернативна специфікація була запропонована компаніями Compaq, Microsoft і National Semiconductor - OHCI (Open Host Interface). Контролери зі специфікації OHCI володіють уніфікованим абстрактним інтерфейсом, що передбачає апаратну реалізацію більшості функцій, що управляють інтерфейсом, Це полегшує їх програмування.

Для інтерфейсу USB розроблено декілька версій. Версія USB 1.0 представлена в січні 1995 року. Вона забезпечувала наступні технічні характеристики:

- високошвидкісне з'єднання – 12 Мбіт/с;
- максимальна довжина кабеля для високошвидкісного з'єднання – 3 м;
- низькошвидкісне з'єднання – 1,5 Мбіт/с;
- максимальна довжина кабелю для низькошвидкісного з'єднання – 5 м;
- максимальна кількість пристроїв підімкнення (враховуючи) концентратори – 127;
- можливість підключення пристроїв із різними швидкостями обміну інформацією;
- напруга живлення для периферійних пристроїв – 5 В;
- максимальний струм споживання на один пристрій – 500 мА.

Версія USB 1.1 випущено у вересні 1998 року. Виправлені проблеми, виявлені у версії 1.0, в основному пов'язані з концентраторами. Інтерфейс USB 1.1 декларує два режими:1) низькошвидкісний підканал (пропускна спроможність – 1,5 Мбіт/с), призначений для таких пристроїв, як миші і клавіатури;

2) високопродуктивний канал, що забезпечує максимальну пропускну спроможність 12 Мбіт/с, що може використовуватися для підключення зовнішніх накопичувачів або пристроїв обробки і передачі аудіо- і відеоінформації.

Версія USB 2.0 випущена в квітні 2000 року. USB 2.0 відрізняється від USB 1.1 лише швидкістю передачі, яка зросла, та незначними змінами в протоколі передачі даних для режиму Hi-speed (480 Мбіт/сек). Існує три швидкості роботи пристроїв USB 2.0:

- Low-speed 10–1500 Кбіт/с (використовується для інтерактивних пристроїв: клавіатури, мишки, джойстики);
- Full-speed 0,5–12 Мбіт/с (аудіо/відео пристрої);
- Hi-speed 12–480 Мбіт/с (відео пристрої, пристрої зберігання інформації);

Насправді хоча й швидкість USB 2.0 може досягати 480Мбит/с, пристрої типу жорстких дисків чи взагалі будь-які інші носії інформації ніколи не досягають її по шині USB, хоча і могли б. Це можна пояснити доволі просто, шина USB має доволі велику затримку між запитом на передачу інформацію і самою передачею даних ("довгий ping"). Наприклад,

шина FireWire забезпечує максимальну швидкість у 400 Мбіт/с, тобто на 80Мбіт/с менше чим у USB, дозволяє досягнути більшої швидкості обміну даними з носіями інформації.

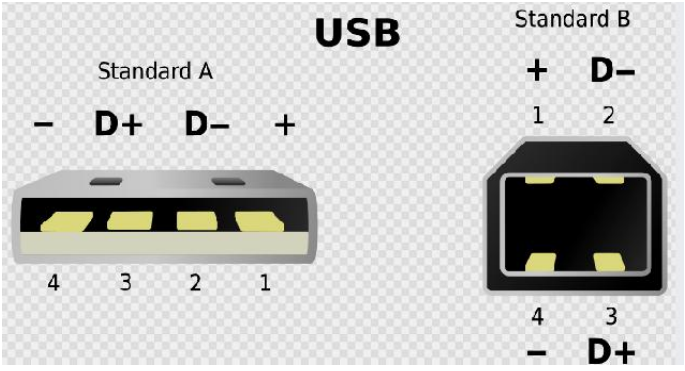
Технологія USB On-The-Go розширює специфікації USB 2.0 для легкого з'єднання між собою периферійних USB-пристроїв безпосередньо між собою без задіяння комп'ютера. Прикладом застосування цієї технології є можливість підключення фотоапарату напряму до друкарки. Цей стандарт виник через об'єктивну потребу надійного з'єднання особливо поширених USB-пристроїв без застосування комп'ютера, який в потрібний момент може і не виявитися під руками.

Офіційна специфікація протоколу бездротового USB була анонсована в травні 2005 року. Дозволяє організовувати бездротовий зв'язок з високою швидкістю передачі даних до 480 Мбіт/с на відстані 3 м та до 110 Мбіт/с на відстані 10 м. Для бездротового USB часом використовують абревіатуру WUSB. Розробник протоколу USB-IF віддає перевагу практиці іменування протокол офіційно Certified Wireless USB.

Протокол USB 3.0 орієнтований на передавання сигналів за допомогою оптоволоконного кабелю. USB 3.0 є зворотно сумісним з USB 2.0 та USB 1.1. Створенням USB 3.0 займаються компанії: Intel, Microsoft, Hewlett-Packard, Texas Instruments, NEC и NXP Semiconductors.

Теоретична пікова пропускна здатність інтерфейсу складає 4,8 Гбіт/с. Специфікація периферійної шини USB була розроблена лідерами комп'ютерної і телекомунікаційної промисловості (Compaq, DEC, IBM, Intel, Microsoft, NEC і Northern Telecom) для підключення комп'ютерної периферії поза корпусом ПК з автоматичною автоконфігурацією (Plug&Play). Перша версія стандарту з'явилася в 1996 р. Агресивна політика Intel по впровадженню цього інтерфейсу стимулює поступове зникнення таких низькошвидкісних інтерфейсів, як RS 232C, Access.bus і тому подібне Проте для високошвидкісних пристроїв строгішими вимогами до продуктивності (наприклад, доступ до видаленого накопичувача або передача оцифрованого відео) конкурентом USB є інтерфейс IEEE 1394.

Інтерфейс регламентує конструктивну сумісність пристроїв, стандартизуючи типи з'єднувачів та розведення сигналів на контактах. Для інтерфейсу стандартизовано два типи з'єднувачів: а та в (див. рисунок)



У кабелях використовуються провідники згідно з таблицею.

Номер контакту	Позначення	Колір провідника
1	V BUS	червоний
2	D-	білий
3	D+	зелений
4	GND	чорний

GND – ланцюг “корпусу” для живлення периферійних пристроїв, VBus – +5 В, також для ланцюгів живлення. Дані передаються по провідникам D+ і D− диференційно (стан 0 и 1 (в термінології офіційної документації diff0 и diff1 відповідно) визначаються по різниці потенціалів між лініями більше 0,2 В і при умові, що на одній із ліній (D− у випадку diff0 и D+ при diff1) потенціал відносно GND вище 2,8 В. Диференційний спосіб передачі є основним, проте не єдиним (наприклад, при ініціалізації пристрій повідомляє хосту про режим, що підтримується пристроєм (Full-Speed або Low-Speed), підтягуванням однієї із ліній даних до V_BUS через резистор 1.5 кОм (D – для режиму Low-Speed і D+ для режиму Full-Speed, пристрої, що працюють в режимі Hi-Speed, ведуть себе на цій стадії як пристрої в режимі Full-Speed). Також інколи навколо проводу присутня волокниста обмотка для захисту від фізичних пошкоджень.

Інтегруванням напівпровідникової пам’яті та інтерфейсу USB створена флеш пам’ять. Це тип пам’яті, яка може на довготривало зберігати певну інформацію на своїй платі, зовсім не використовуючи живлення. У додаток можна сказати, що флеш пам’ять пропонує високу швидкість доступу до інформації (хоча вона не настільки висока як у DRAM) і кращий опір до кінетичного шоку, ніж у вінчестерах. Ці характеристики пояснюють популярність флеш пам’яті для приладів, що залежать від батарейок. Іншою приманкою флеш пам’яті є те, що коли вона скомпресована в суцільну “карту пам’яті”, стає майже неможливо зруйнувати її стандартними фізичними методами, що дає змогу витримувати високий тиск і кип’ячу воду.

Флеш-пам’ять зберігає інформацію в масиві “комірок”, кожна з яких традиційно зберігає по одному біту інформації. Кожна комірка – це транзистор із плаваючим затвором. Новіші пристрої (інколи їх ще називають багатозарядними пристроями) можуть містити більше, ніж 1 біт в комірці, використовуючи два чи більше рівні електричних зарядів, розташованих при плаваючому затворі комірки. У флеш пам’яті типу NOR кожна комірка схожа на стандартний MOSFET (оксидний напівпровідниковий польовий транзистор), але у ній є не один затвор, а два. Як і будь-який інший польовий транзистор, вони мають контрольний затвор (КЗ), а, окрім нього, ще й інший – плаваючий (ПЗ), замкнений всередині оксидного шару. ПЗ розташований між КЗ і підкладкою. Оскільки ПЗ відокремлений власним заізольованим шаром оксиду, будь-які електрони, що попадають на нього відразу потрапляють в пастку, що дозволяє зберігати інформацію. Захоплені плаваючим затвором електрони змінюють (практично компенсують) електричне поле контрольного затвору, що змінює порогову напругу (Vп) затвору. Коли з комірки “зчитують” інформацію, до КЗ прикладають певну напругу, залежно від якої в каналі транзистора протікатиме або не

протікатиме електричний струм. Ця напруга залежить від Vп комірки, яка в свою чергу контролюється числом захоплених плаваючим затвором електронів. Величина порогової напруги зчитується і перекодовується в одиницю чи нуль. Якщо плаваючий затвор може мати кілька зарядових станів, то зчитування відбувається за допомогою вимірювання сили струму в каналі транзистора. Для запису інформації в комірку NOR необхідно зарядити плаваючий затвор. Цього досягають, пропускаючи через канал транзистора сильний струм, при якому виникають гарячі електрони, що мають достатню енергію для подолання оксидного шару. Для очищення плаваючого затвору від електронів (стирання інформації) до контрольного затвору прикладають значну напругу, яка створює сильне електричне поле. Захоплені плаваючим затвором електрони висмоктуються цим полем, тунелюючи через оксидний шар. У приладах з однотипною напругою (теоретично всі чіпи, які доступні нам на сьогоднішній день) ця висока напруга створюється генератором підкачки заряду. Більшість сучасних компонентів NOR пам'яті розділені на чисті сегменти, які часто називають блоками чи секторами. Всі комірки пам'яті в блоці повинні бути очищені одночасно. На жаль, метод NOR може в загальному випадку обробляти лише одну частину інформації типу byte чи word. NAND пам'ять використовує тунельну інжекцію для запису і тунельний випуск для вилучення. NAND'ова флеш пам'ять формує ядро легкого USB інтерфейсу запам'ятовуючих приладів, які також відомі як USB флешки. Тоді, коли розробники збільшують густину флеш приладів, індивідуальні комірки діляться і кількість електронів у будь-якій комірці стає дуже малою. Парування між суміжними плаваючими затворами може змінити характеристики запису комірки. Нові реалізації, як-от заряджені пастки флеш-пам'яті, намагаються забезпечити кращу ізоляцію між суміжними комірками.

Лабораторна робота № 5

РОЗРОБЛЕННЯ ТА ДОСЛІДЖЕННЯ ДРАЙВЕРА ПЕРЕДАВАЧА ІНТЕРФЕЙСА УПШ

Мета роботи: опанування студентом технології створення програмного драйвера передавання повідомлення через послідовний інтерфейс УПШ (USB).

Завдання на роботу. Конкретний пакет даних та відповідне повідомлення студенту визначає викладач (переважно невелике повідомлення із великих букв кирилиці, що кодуються модифікованою таблицею ASCII+). Задається швидкість обміну даними.

Технологія виконання роботи. Драйвер передавача повинен бути налаштованим на характеристики, які індивідуально задані студенту. Насамперед це швидкість обміну даними та кількість байт інформаційного повідомлення. Спочатку потрібно визначитися із структурою транзакції. Якщо передавачем буде контролер комп'ютера, тоді транзакція має включати три пакети: маркерний, даних та підтвердження. Маркерний пакет має забезпечити необхідний режим синхронізації пристроїв, визначити папрям передавання даних, встановити кількість байт у пакеті даних. Кількість байт у пакеті даних має відповідати стандарту та має бути більшою або рівною кількості байт у заданому інформаційному повідомленні. Якщо у пакеті даних є надлишок інформаційних байт, то цей надлишок заповнюється нулями.

Для створення драйвера необхідно розробити відповідну схему алгоритму. Розроблення схеми алгоритму є одним із показників поглибленого розуміння логіки взаємодії пристроїв у периферійній підсистемі на основі інтерфейсу УПШ.

На рис. 54 наведено приклад реалізації програмного драйвера передавача даних через послідавний USB інтерфейс (див. лістінги програми).

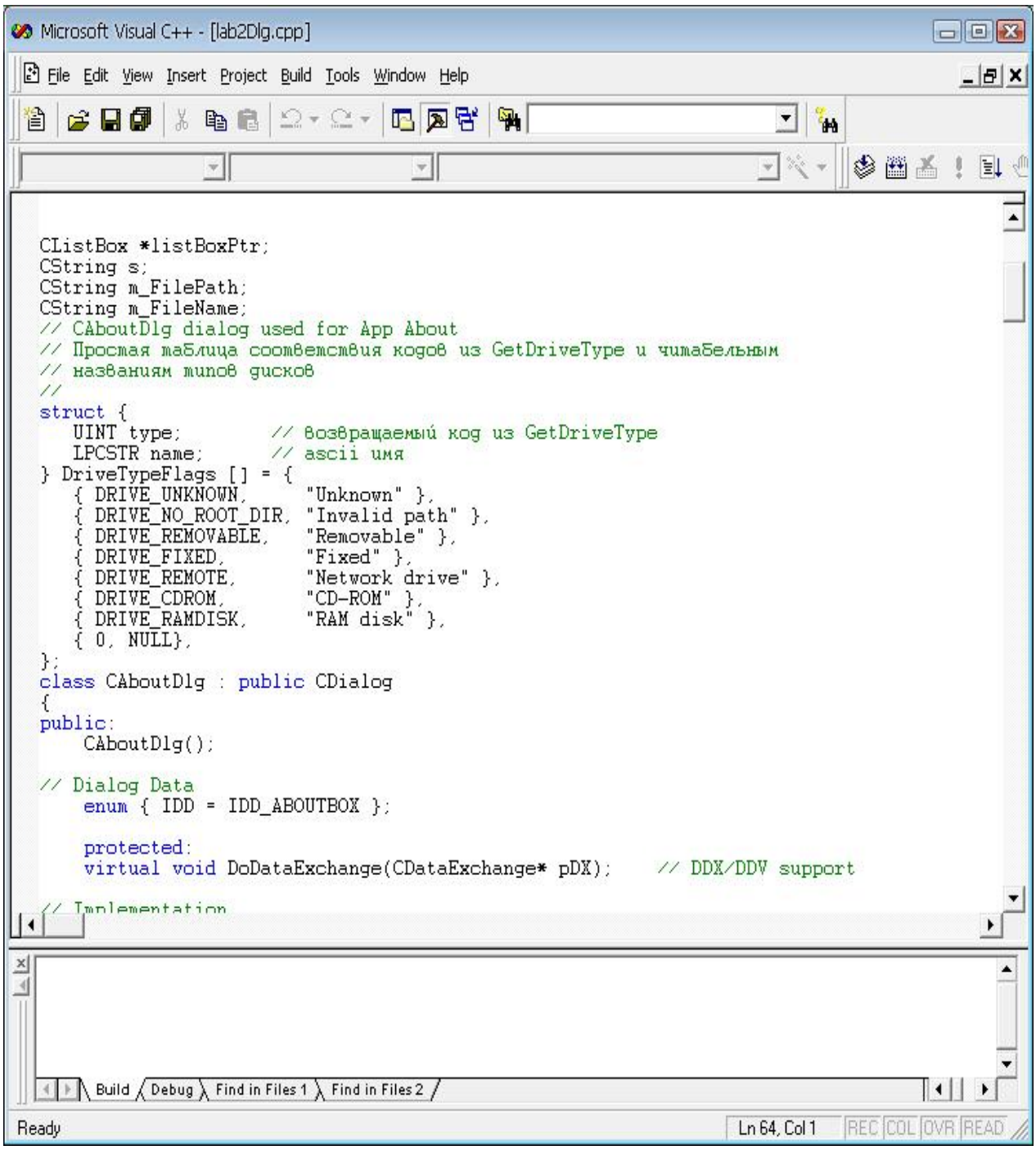


Рис. 54. Вікно програмного драйвера передавання даних через USB інтерфейс

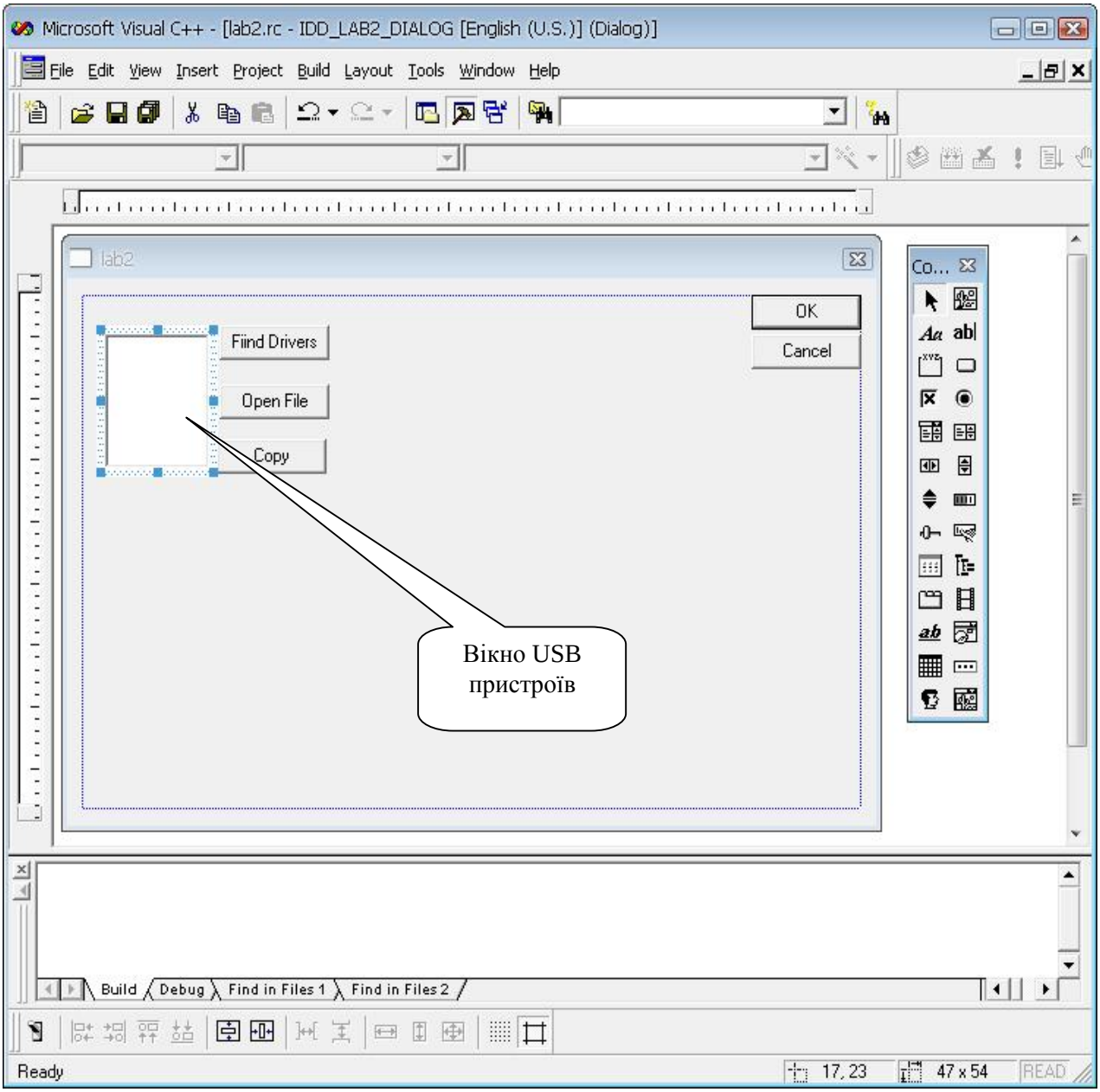


Рис. 55. Діалогово вікно USB пристроїв

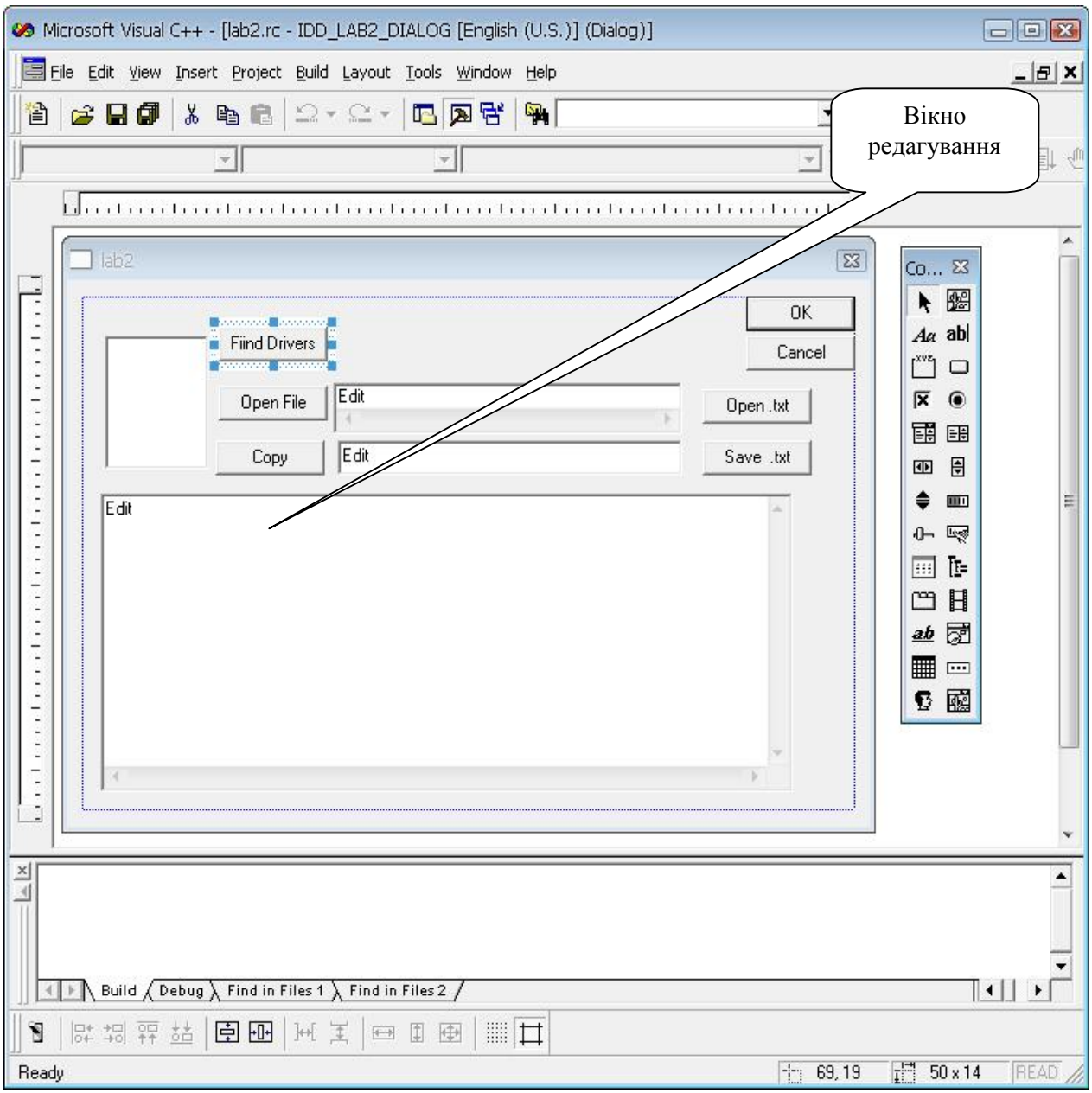


Рис. 56. Діалогово вікно програми передавача

На рис. 55 та рис. 56 продемонстровано створення елементів діалогового вікна програми передачі даних через послідовний USB інтерфейс. Зокрема наведено ComboBox, в якому при натисненні кнопки “Find Drive” відображаються периферійні пристрої під’єднані до USB порта (рис. 55), та ComboBox відкриття та редагування текстових файлів для їх подальшого запису через послідовний USB інтерфейс (рис. 56).

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Привести основні характеристики інтерфейсу USB.
2. Привести призначення контактів роз’ємів USB.
3. Привести особливості структур транзакцій інтерфейсу USB.
4. Привести основні етапи відлагодження програмого драйвера.
5. Пояснити роботу написаного програмного драйвера.
6. Пояснити структуру тразакцій інтерфейсу USB .

ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Короткі теоретичні відомості.
3. Хід роботи. Навести структуру транзакції інтерфейсу.
4. Висновки.
5. Текст програми (у додатку).
6. Демонстрація програми на комп’ютері.

Лабораторна робота № 6

СТВОРЕННЯ ЕЛЕМЕНТІВ ПРОГРАМНОГО ДРАЙВЕРА ДЛЯ СКАНУВАННЯ УПШ-ПОРТІВ

Мета роботи: опанування студентом технології створення елементів програмного драйвера для сканування УПШ (USB) портів.

Завдання на роботу: визначити наявність підключених пристроїв до послідовного USB інтерфейсу

Технологія виконання роботи. На рис. 57 наведено відрізок тексту програми передачі даних через послідовний USB інтерфейс, який безпосередньо сканує USB порти на наявність підключених до них пристроїв та присвоює їм назви (див. лістінги програми).

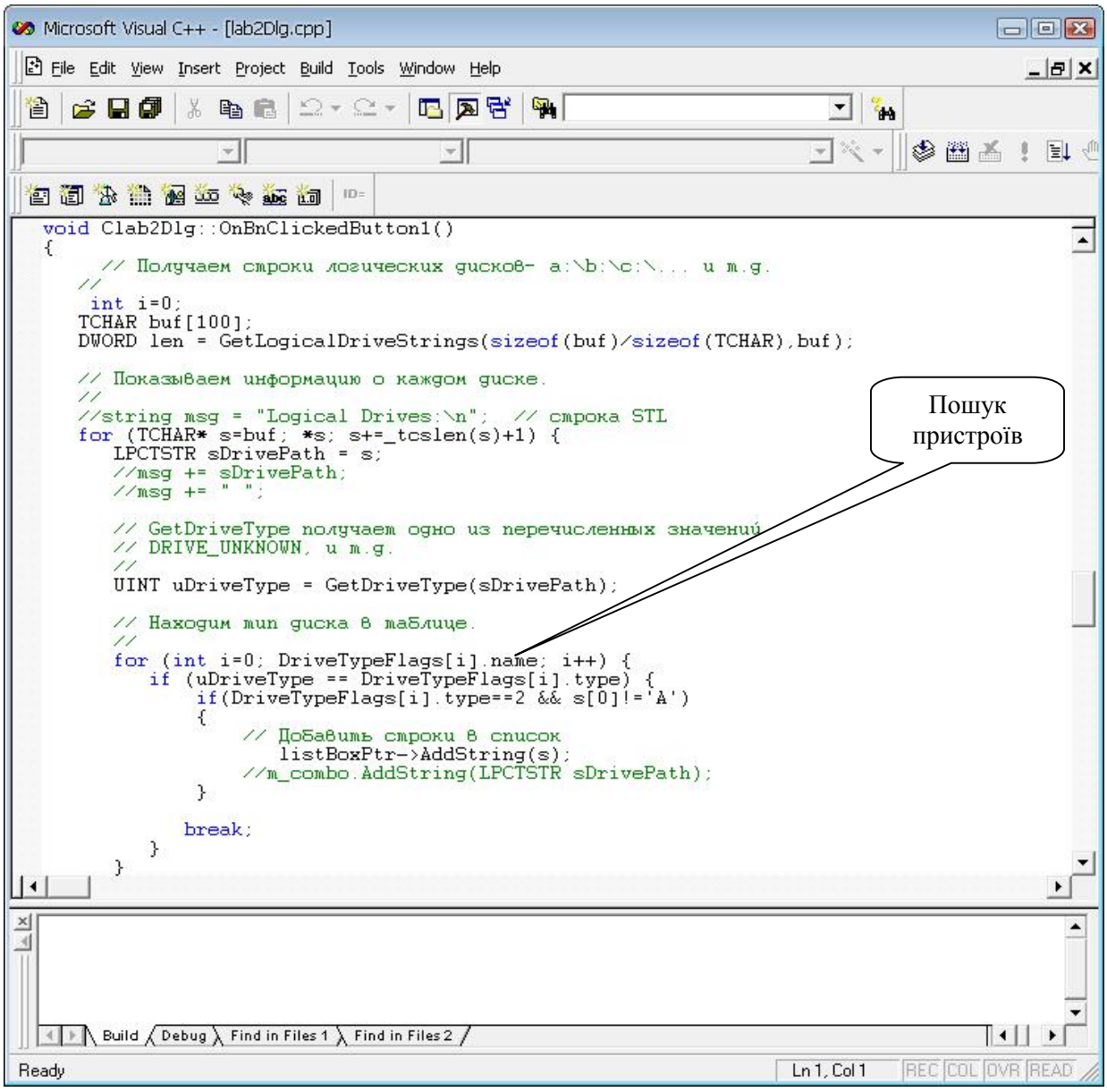


Рис. 57. Сканування переносних носіїв

На рис. 58 наведено створення елемента діалолгового вікна програми передачі даних через послідовний USB інтерфейс, який відповідає за сканування USB портів на наявність підключених до них пристроїв та присвоює їм назви.

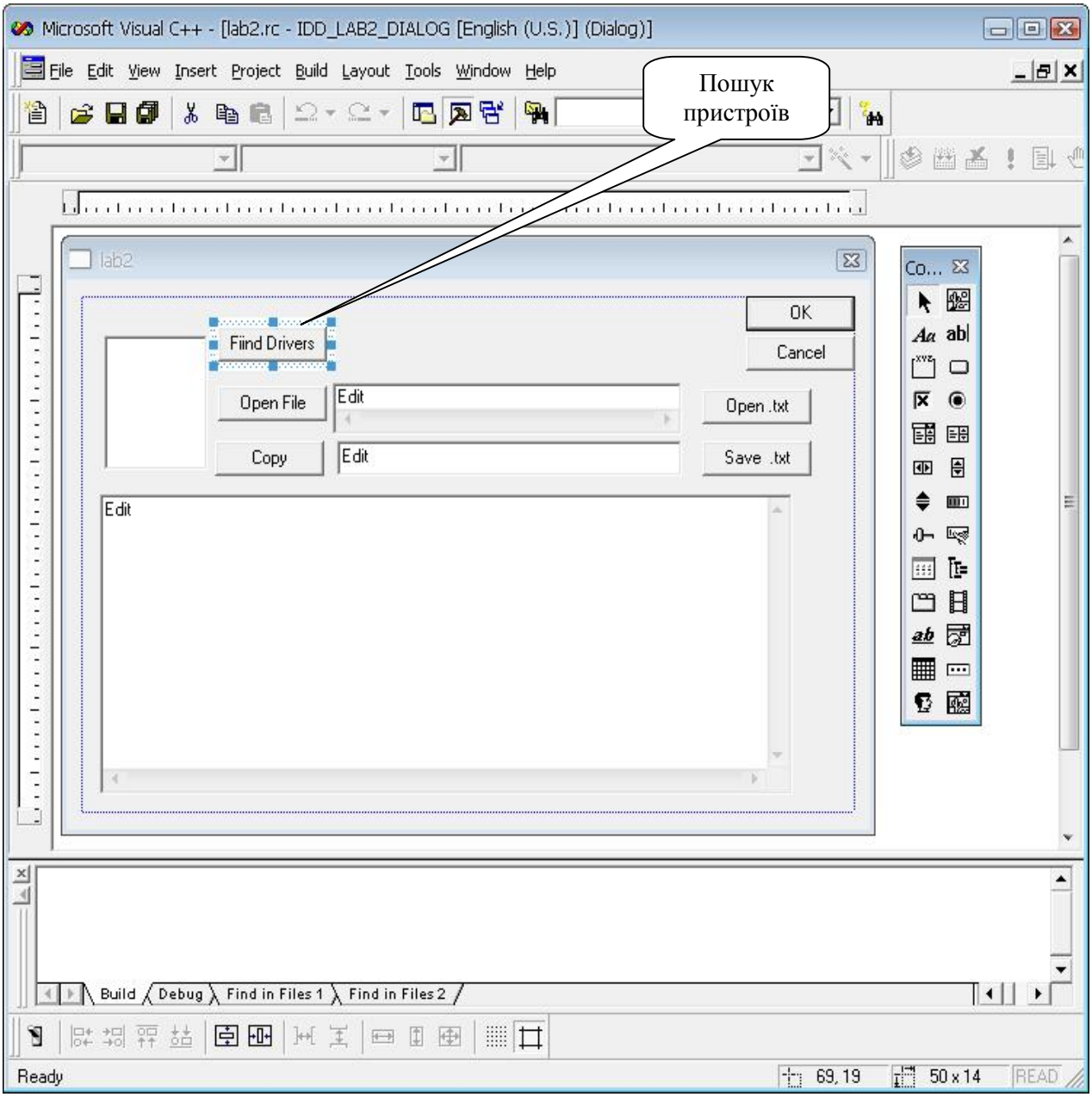


Рис. 58. Елементи сканування ереносних носіїв

На рис. 59 наведено приклад виконання програми передачі даних через послідовний USB інтерфейс, а саме процес сканування, який безпосередньо сканує USB порти на наявність підключених до них пристроїв та присвоює їм назви. У цьому випадку – це 2 флешносія ”G та I”

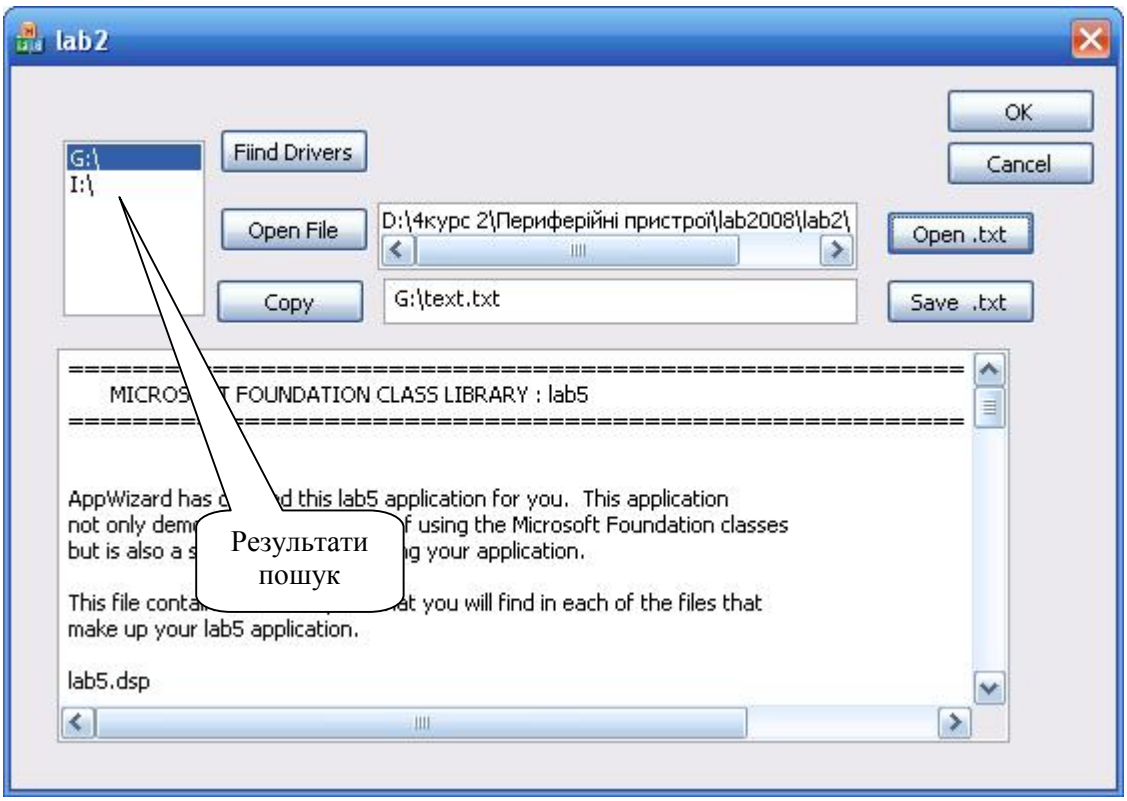


Рис. 59. Присвоєння назв USB пристроям

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Привести основні характеристики інтерфейсу USB.
2. Привести особливості структур транзакцій інтерфейсу USB.
4. Привести основні етапи відлагодження програми сканування інтерфейсних портів.
5. Пояснити роботу написаного програмного драйвера.
6. Пояснити структуру тразакцій інтерфейсу USB .

ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Короткі теоретичні відомості.
3. Хід роботи: схема алгоритму сканування інтерфейсних портів.
4. Висновки.
5. Текст програми (у додатку).
6. Демонстрація програми на комп’ютері.

Лабораторна робота № 7

СТВОРЕННЯ ЕЛЕМЕНТІВ ПРОГРАМНОГО ДРАЙВЕРА ДЛЯ НАЛАШТУВАННЯ УПШ-ПОРТІВ

Мета роботи: опанування студентом технології створення та налаштування основних параметрів програми передачі даних через послідовний USB інтерфейс.

Завдання на роботу: розробити програмний драйвер налаштування інтерфейсних портів для передачі даних через послідовний USB інтерфейс, який призначений для вибору даних для їх подальшого опрацювання.

Технологія виконання роботи. розробити програмний драйвер налаштування інтерфейсних портів для передачі даних через послідовний USB інтерфейс, який призначений для вибору даних для їх подальшого опрацювання, можна наступними засобами. На рис. 60 наведено приклад створення елементу діалогового вікна виконання такої програми передачі даних.

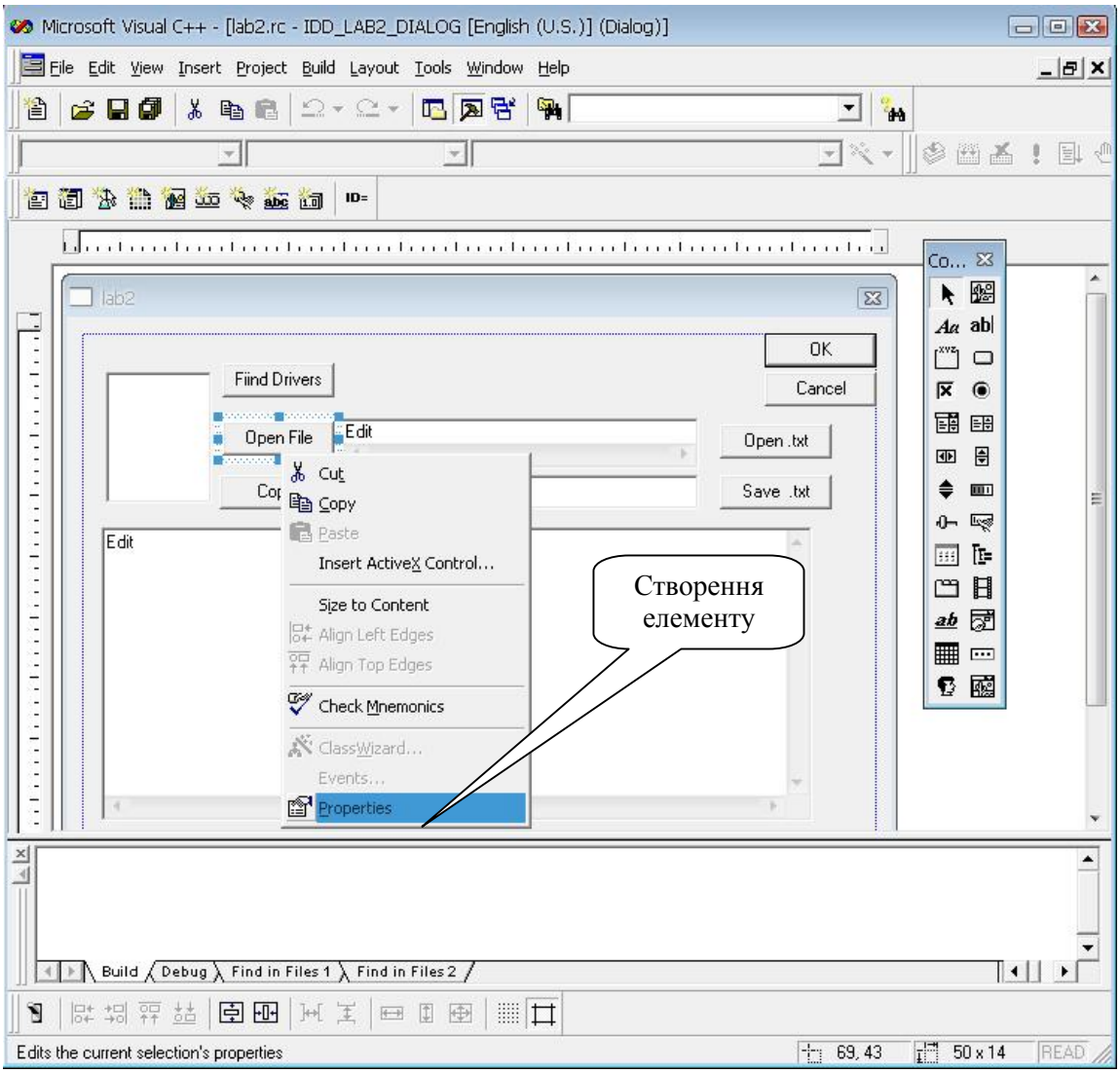


Рис. 60. Створення елементу для вибору файла

На рис. 61 та рис 62 наведено приклад створення елементів діалогового вікна програми передачі даних через послідовний USB інтерфейс, які забезпечують відкриття, редагування та передачу даних на пристрій , який був підключений до USB порта та отримав свій ідентифікатор (див. лістіни програми).

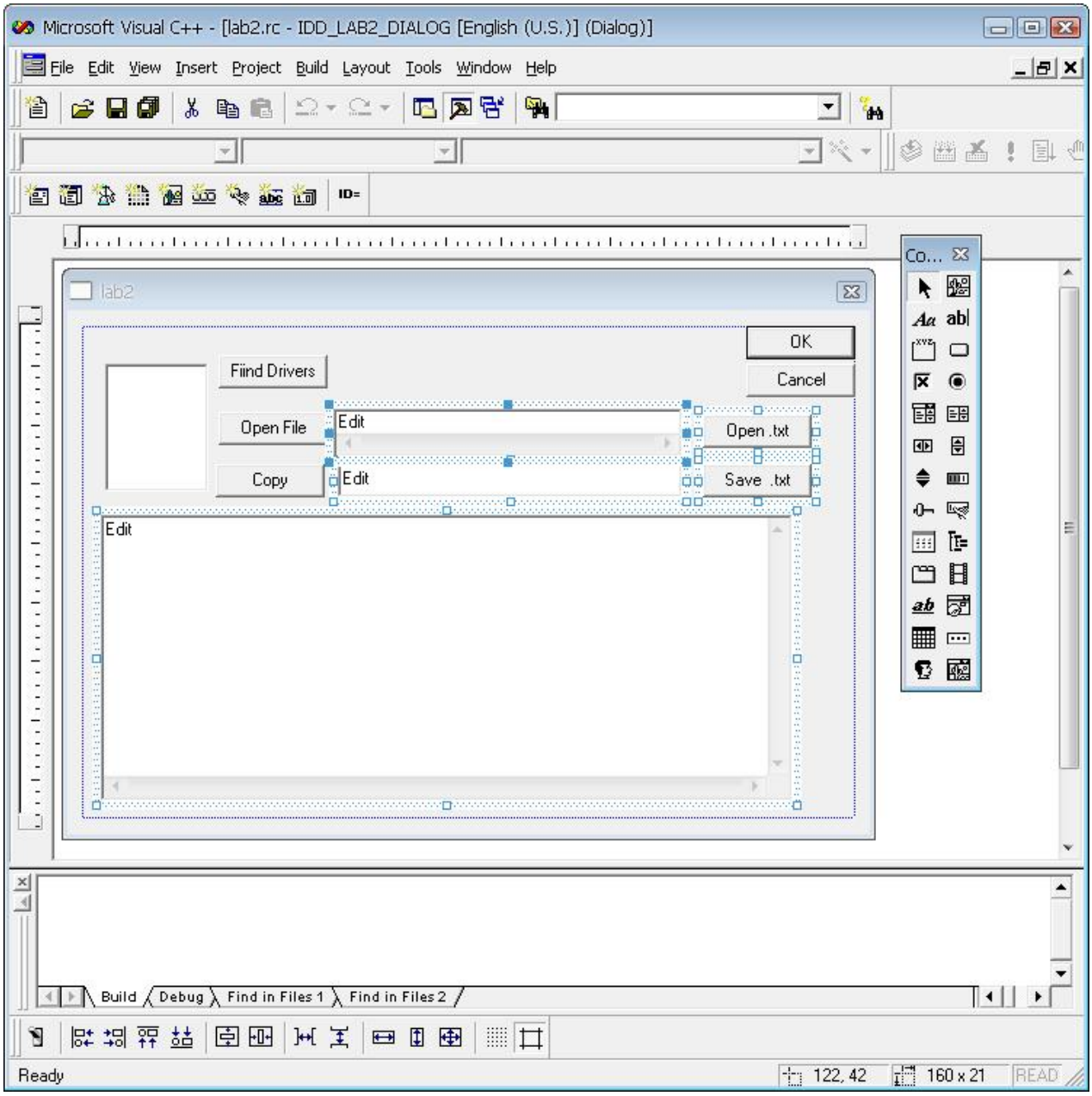


Рис. 61. Основні елементи для вибору відкриття та редагування файла

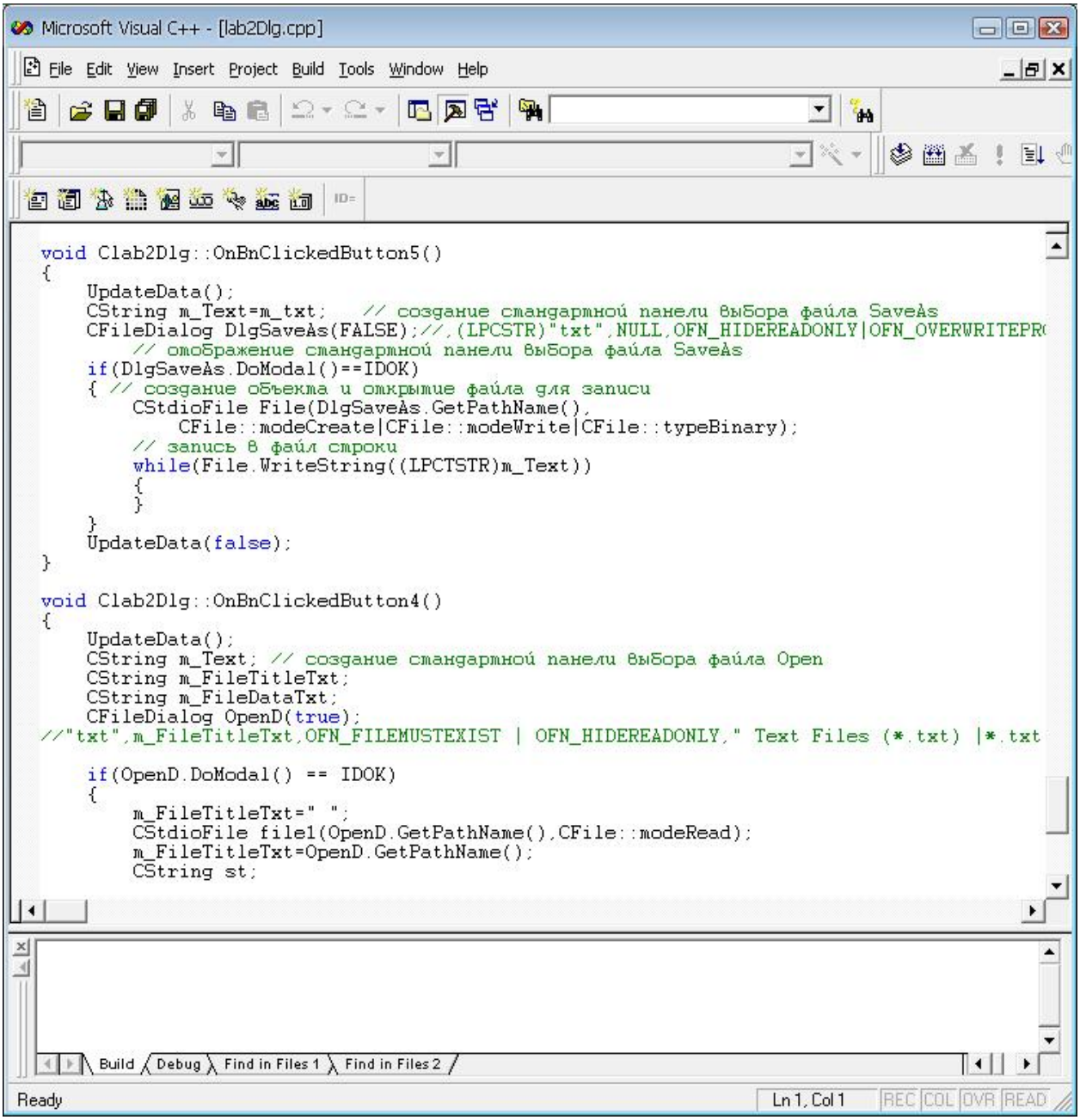


Рис. 62. Відкриття файла для редагування та його запис

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Привести основні характеристики інтерфейсу USB.
2. Привести особливості структур транзакцій інтерфейсу USB.
4. Привести основні етапи відлагодження програмного драйвера налаштування інтерфейсних портів.
5. Пояснити роботу написаного програмного драйвера.
6. Пояснити структуру тразакцій інтерфейсу USB.

ЗМІСТ ЗВІТУ

1. Мета роботи.
2. Короткі теоретичні відомості.
3. Хід роботи; Схема алгоритму роботи програмного драйвера налаштування інтерфейсних портів.
4. Висновки.
5. Текст програми (у додатку).
6. Демонстрація програми на комп’ютері.

Лабораторна робота № 8

ГРАФІЧНЕ ПРЕДСТАВЛЕННЯ ПОСИМВОЛЬНОГО КОДУВАННЯ ІНТЕРФЕЙСА УПШ

Мета роботи: опанування студентом особливостей формування та передачі посимвольних даних у графічному представленні в схемі кодування NRZI через інтерфейс УПШ (USB). Конкретні посимвольні дані для студента визначає викладач (переважно невелике повідомлення із великих букв кирилиці, що кодуються модифікованою таблицею ASCII+). Задається інтерфейсна швидкість обміну даними.

Для передачі сигналів шина USB використовує чотирипровідну магістраль. Одна пара провідників (“+5В чи +3,3В” і “загальний”) призначена для живлення периферійних пристроїв з навантаженням до 500 мА. Дані передаються по іншій парі (“D+” “D-”). Для передачі даних використовується диференціальна напруга до 3 В (з метою зниження впливу шуму) і схема кодування NRZI (що позбавляє від необхідності виділяти додаткову пару провідників під тактовий сигнал).

На рис 63 наведено приклади часових діаграм передачі символьних даних у системі кодування NRZI. Основною особливістю цього кодування є перемикання сигналу при наявності у коді даних логічного “0” та відсутність перемикання сигналу при наявності у коді даних логічної “1”. Така система кодування відрізняється від інших простотою та ефективністю. Недоліком є кодування даних, коли один за одним кодується велика кількість логічних “1”.

Кодування заданого повідомлення можна здійснювати розробленням спеціальної програми або із використанням текстового редактора VISIO.

Розрахунок часу на передавання заданого повідомлення доцільно виконати для усього кадру даних транзакції УПШ.

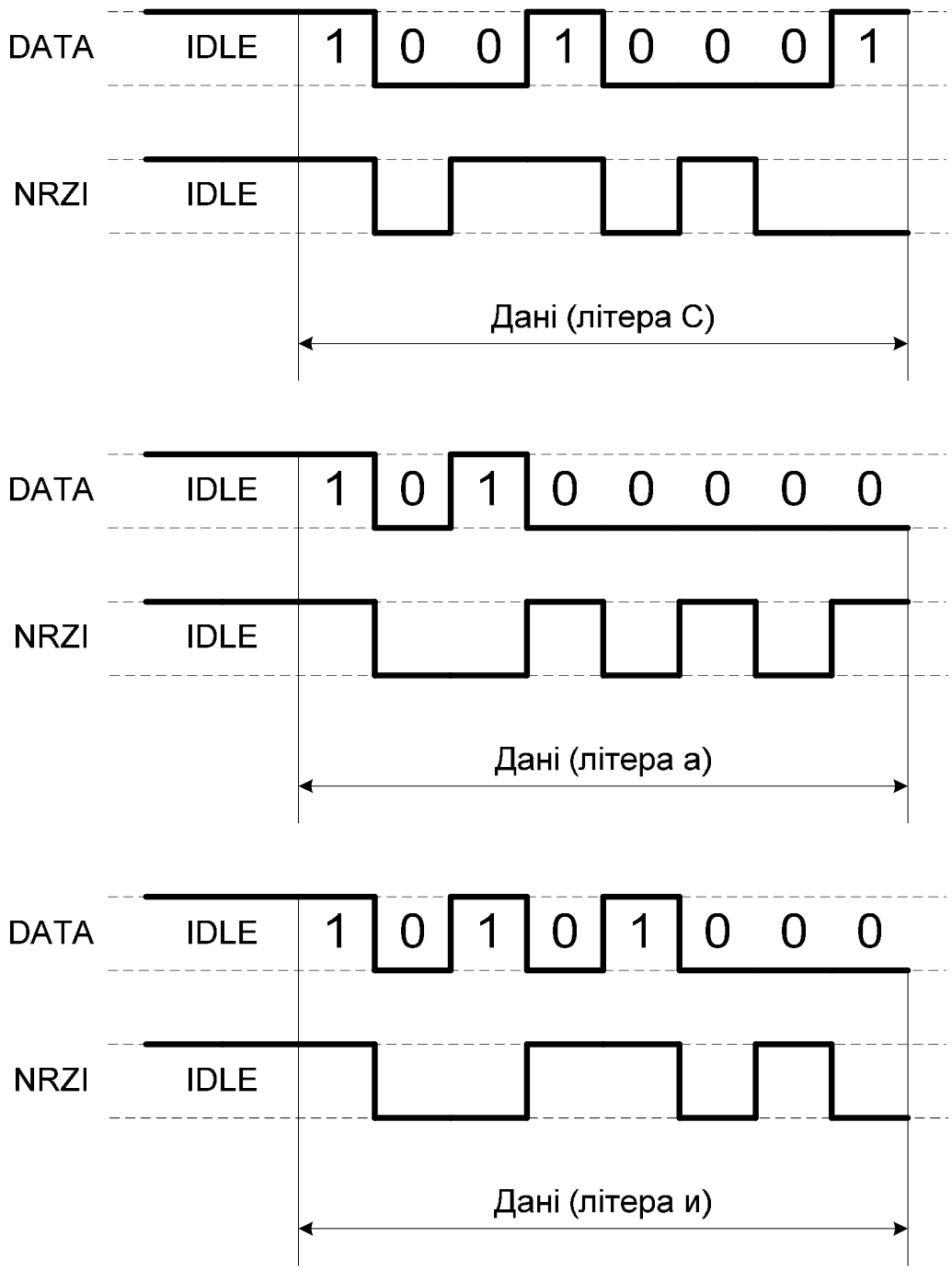


Рис. 63. Кодування NRZI при посимвольній передачі даних

При передачі пакетних повідомлень кількість байт даних у полі даних має відповідати вимогам стандарту.

ПИТАННЯ ДЛЯ САМОПЕРЕВІРКИ

1. Привести основні характеристики інтерфейсу USB.
2. Привести особливості системи кования NRZI.
3. Привести основні етапи відлагодження програмого драйвера.
4. Пояснити роботу написаного програмного драйвера.
5. Пояснити структуру тразакцій інтерфейсу USB.

ЗМІСТ ЗВІТУ

1. Мета роботи.
2 Короткі теоретичні відомості.
3 Хід роботи. Навести часові діаграми закодованих системою NRZI символів заданого повідомлення.
4 Розрахувати час передавання символьного повідомлення.
5 Висновки.
6 Текст програми (у додатку).
7 Демонстрація програми на комп’ютері.

ПРИКЛАД ВИКОНАННЯ ОКРЕМИХ ЕТАПІВ РОБІТ 5–8

Тут подана інформація, яка допомагає створити проєкт програми в середовищі Visual C++ 6.0. Студент, за побажанням, може використати інше середовище та реалізувати програмні драйвери на основі власних оригінальних алгоритмів, шо підвищує якість виконання робіт.

Для того щоб створити новий проєкт, необхідно запустити на виконання програму MS Visual C++ 6.0 (рис. 64).

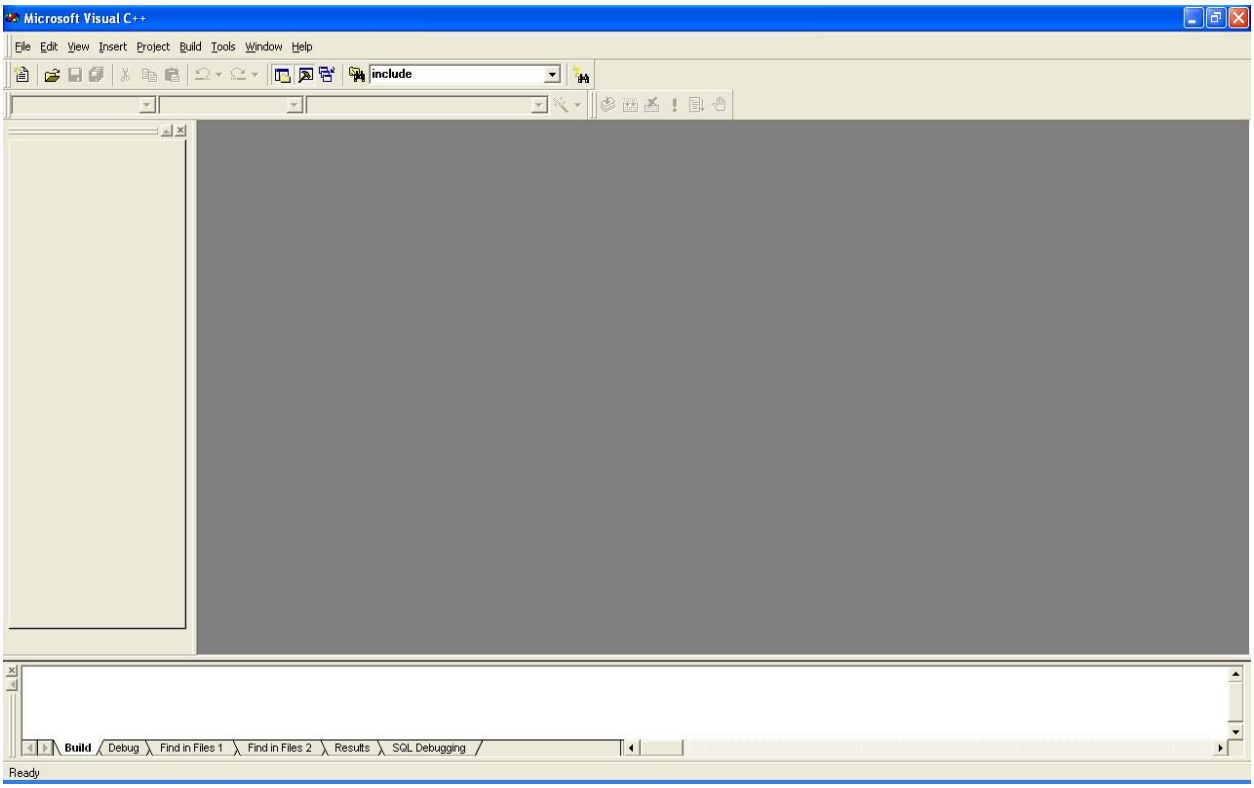


Рис. 64. Головне вікно середовища програмування MS Visual C++ 6.0

Далі в меню середовища MS Visual C++ 6.0 вибрати “File” –> “New” (рис. 65).

Необхідно обов’язково позначити тип нового проєкту (MFC AppWizard (exe)), вказати його назву і натиснути “OK”.

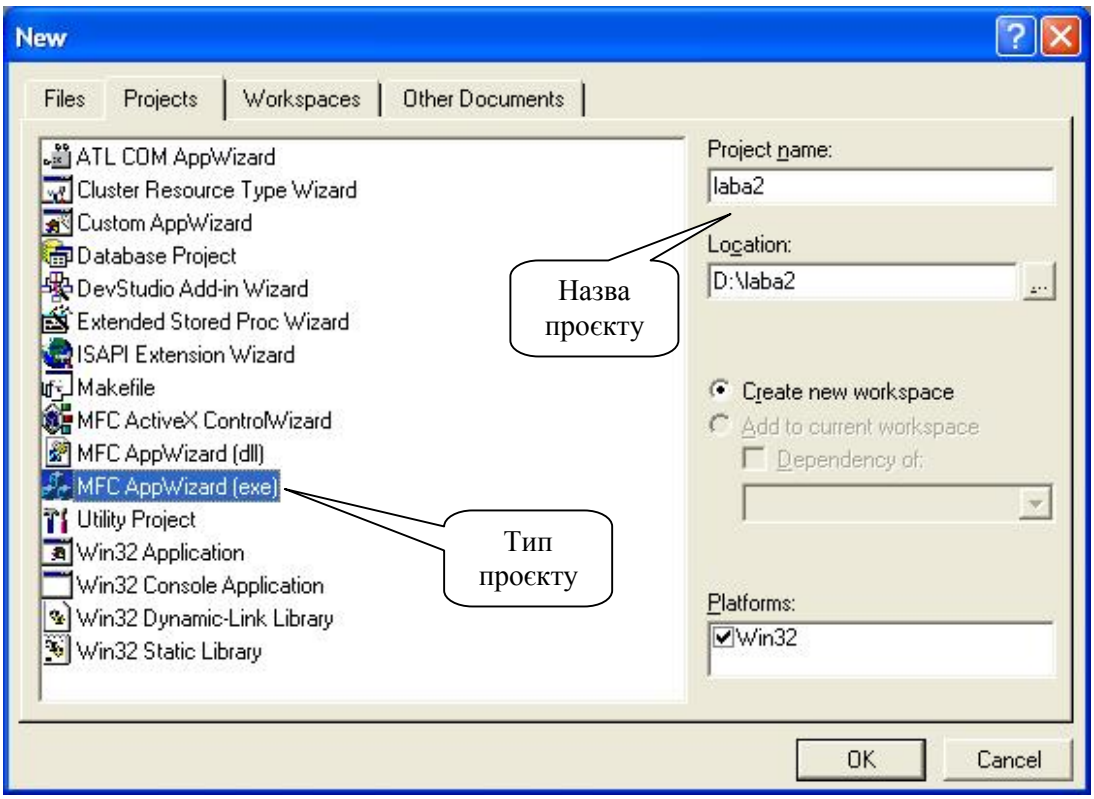


Рис. 65. Вікно створення нового MFC проєкту

Далі треба вибрати вигляд головного вікна нового MFC проєкту (рис. 66) і натиснути “OK”.

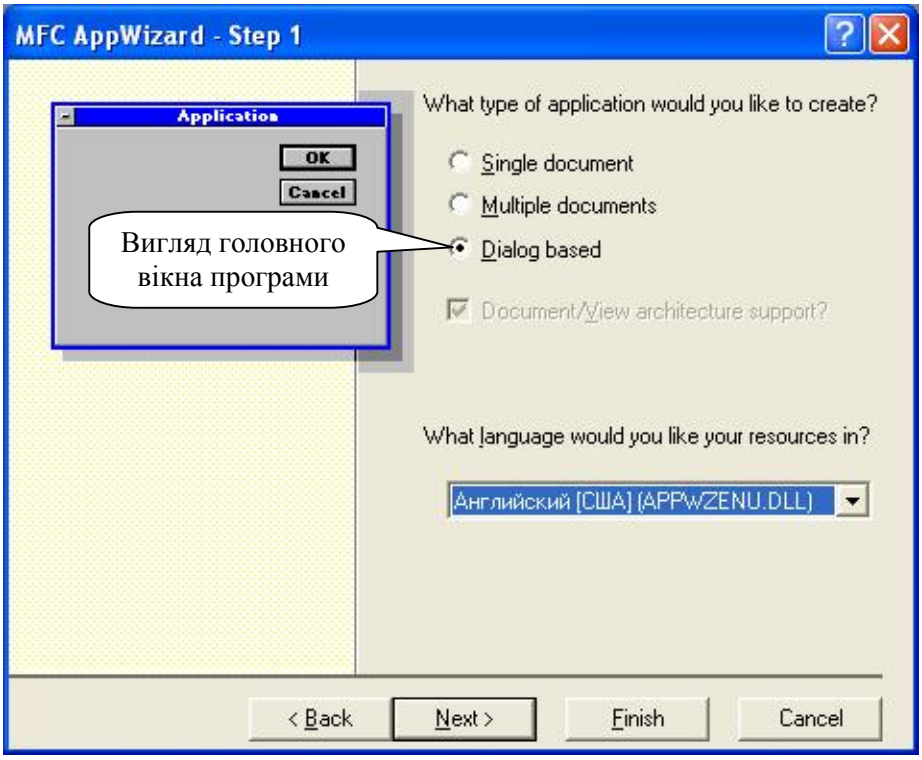


Рис. 66. Вікно вибору вигляду головного вікна нового MFC проєкту

На наступному вікні треба зняти всі прапорці та ввести заголовок вікна діалогу (рис. 67).

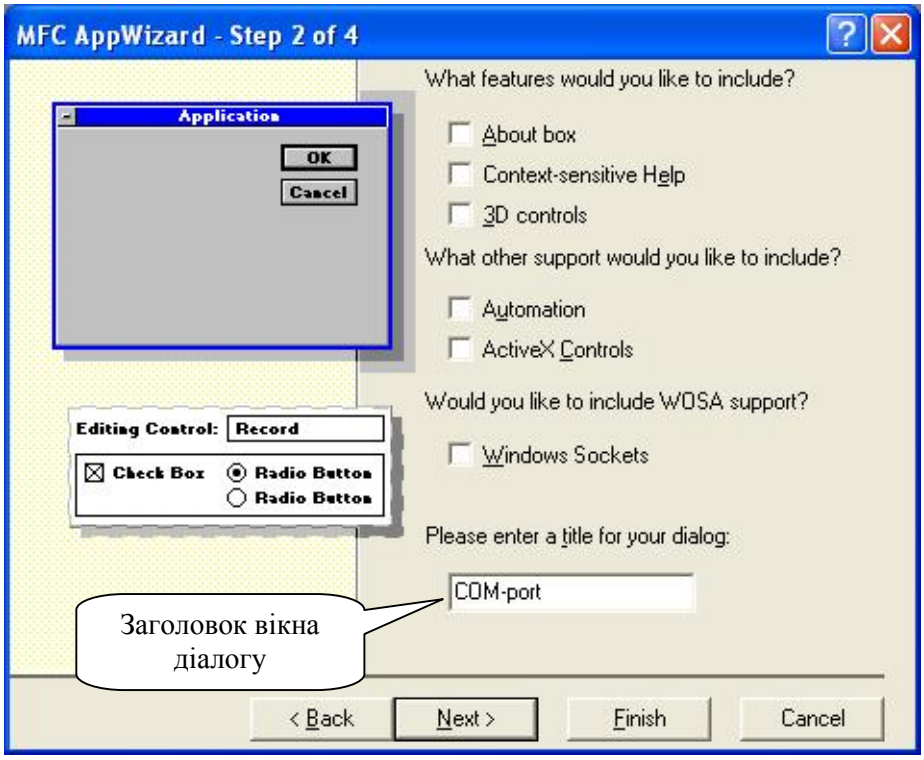


Рис. 67. Вікно вигляду елементів діалогу

Далі треба натиснути “Next” (рис. 68).

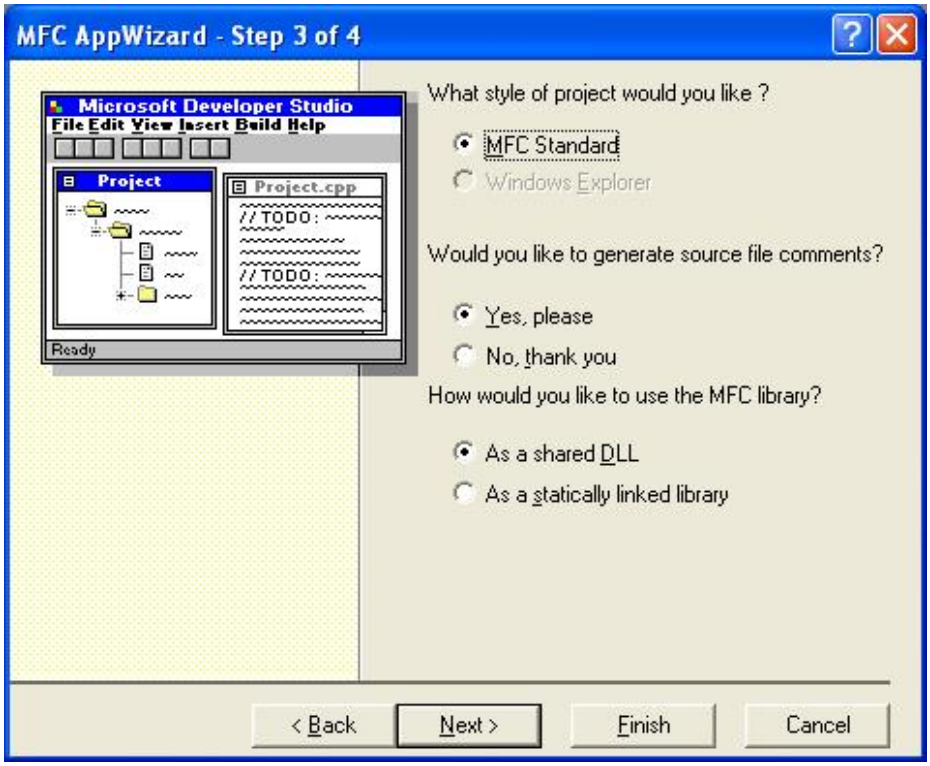


Рис. 68. Вікно вибору властивостей проєкту

Далі треба натиснути “Finish” (рис. 69).

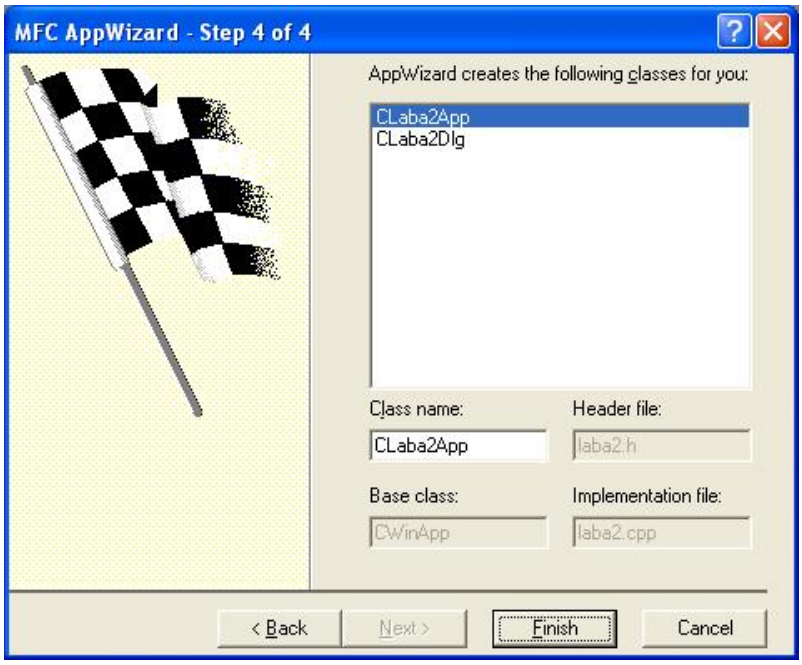


Рис. 69. Вікно вибору властивостей проєкту

Далі прочитати сумарну інформацію про новий MFC проєкт (рис. 70) і, якщо немає помилок, натиснути “OK”. В іншому випадку натиснути “Cancel” і відмінити створення нового проєкту.

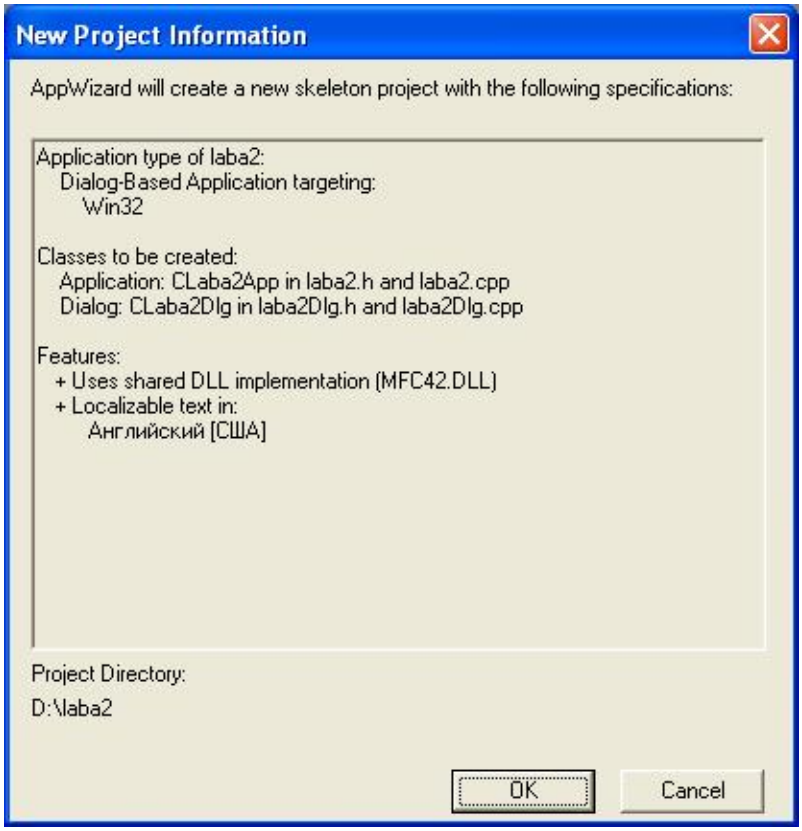


Рис. 70. Вікно сумарної інформації про новий MFC проєкт

Для того, щоб відкрити існуючий проєкт, необхідно вибрати “Open Workspace…” або “Recent Workspaces” з “File” меню.

Новий MFC проєкт (рис. 71) містить класи, ресурси і файли програми (CTestApp) та вікна діалогу (CTestDlg).

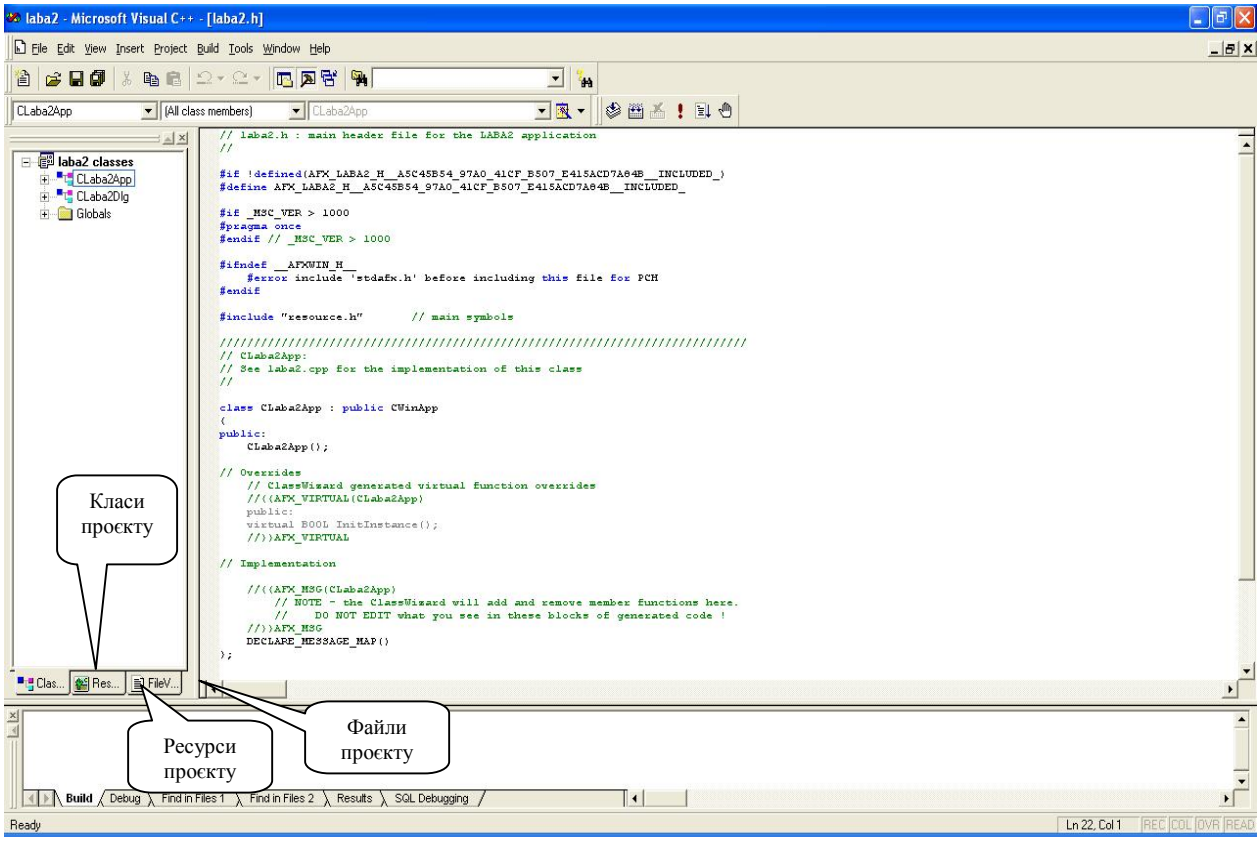


Рис. 71. Вікно середовища MS Visual C++ 6.0 з новим проєктом

Клас **CLaba2App** містить функцію **InitInstance()**, із якої починається виконання програми і створення об’єкту програми **theApp** (рис. 72). У функції **InitInstance**() створюється об’єкт діалогу, до нього приєднується вікно діалогу і діалог відображається на екрані. Крім того є обробники натискання клавіш “**OK**” і “**Cancel**”. Там можна додати свій зміст.

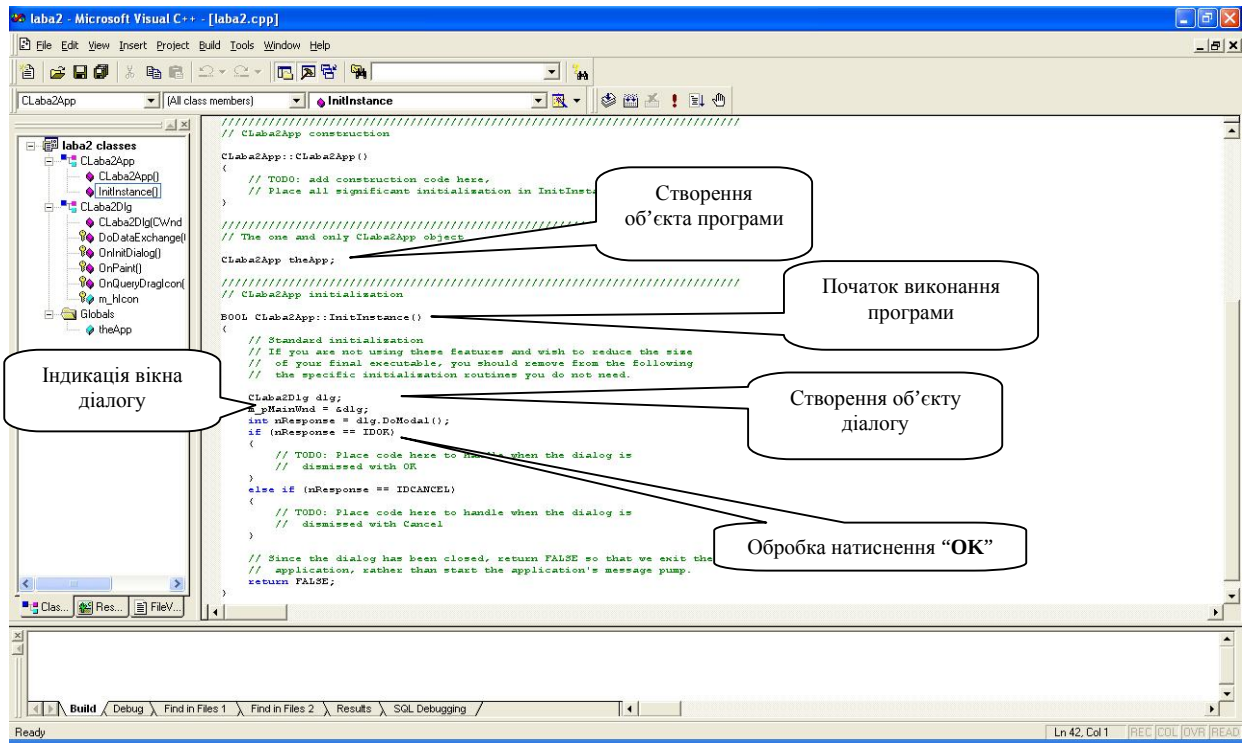


Рис. 72. Клас ***CLaba2App***

Клас **CLaba2Dlg** містить функції **OnInitDialog**() та **OnPaint**() (рис. 73). В функції **OnInitDialog**() налаштовується зовнішній вигляд вікна діалогу, а функція **OnPaint**() здійснює вивід вікна діалогу на екран.

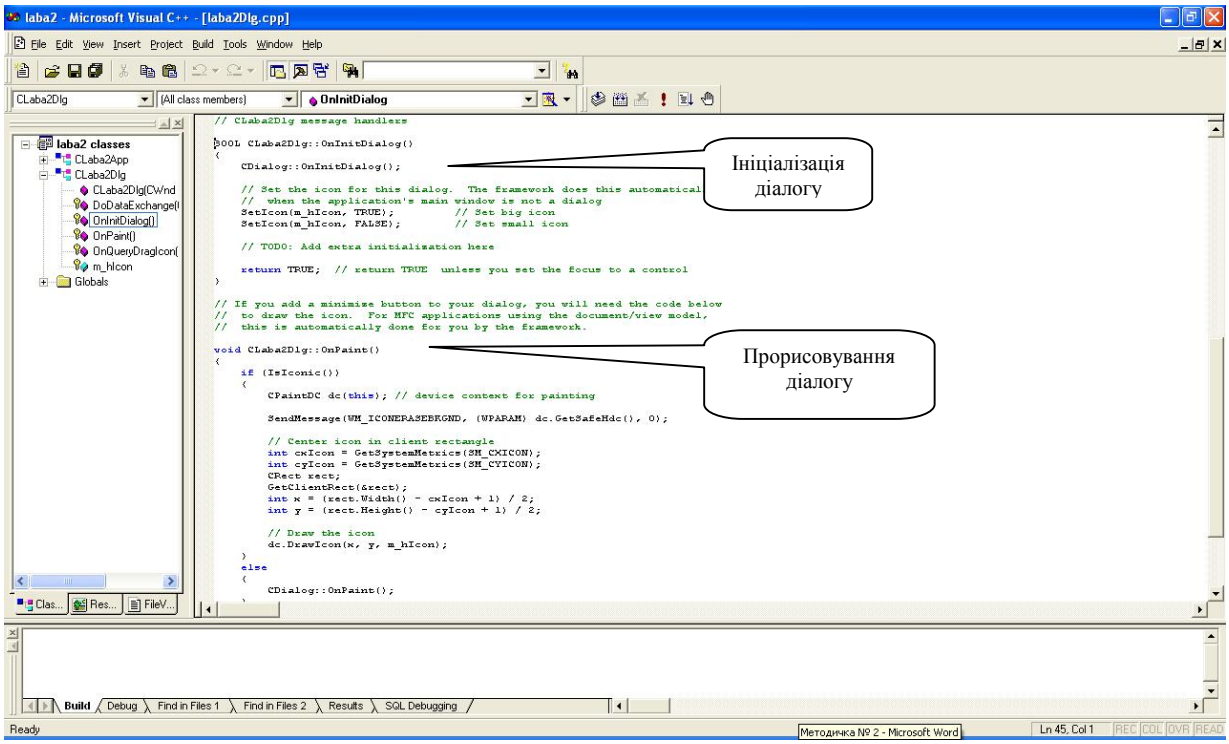


Рис. 73. Клас ***CLaba2Dlg***

Для додавання функціональності до діалогу спочатку треба нарисувати додаткові елементи керування в редакторі діалогів (рис. 74).

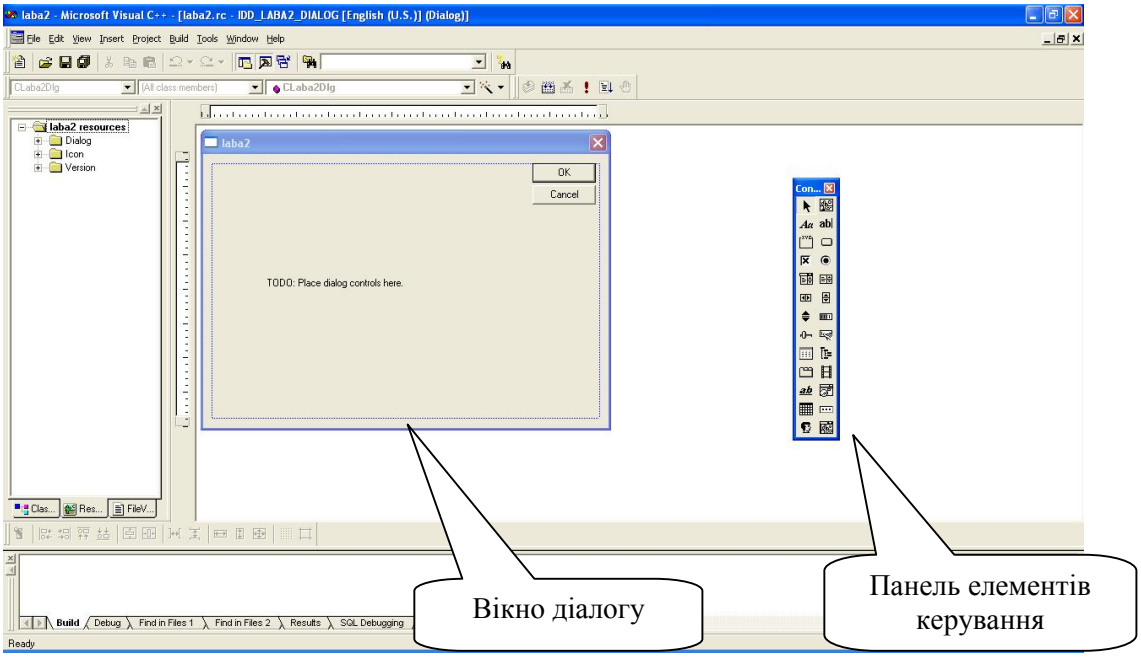


Рис. 74. Редактор діалогу

Для цього на панелі елементів керування вибирається один елемент. На вікні діалогу мишею натягується прямокутник. Після відпускання кнопки миші, на вікні діалогу з'являється зображення елемента керування. Для даної лабораторної роботи необхідно на вікні діалогу встановити об'єкти типу Edit (поле редагування) та Button (кнопка) (рис. 75).

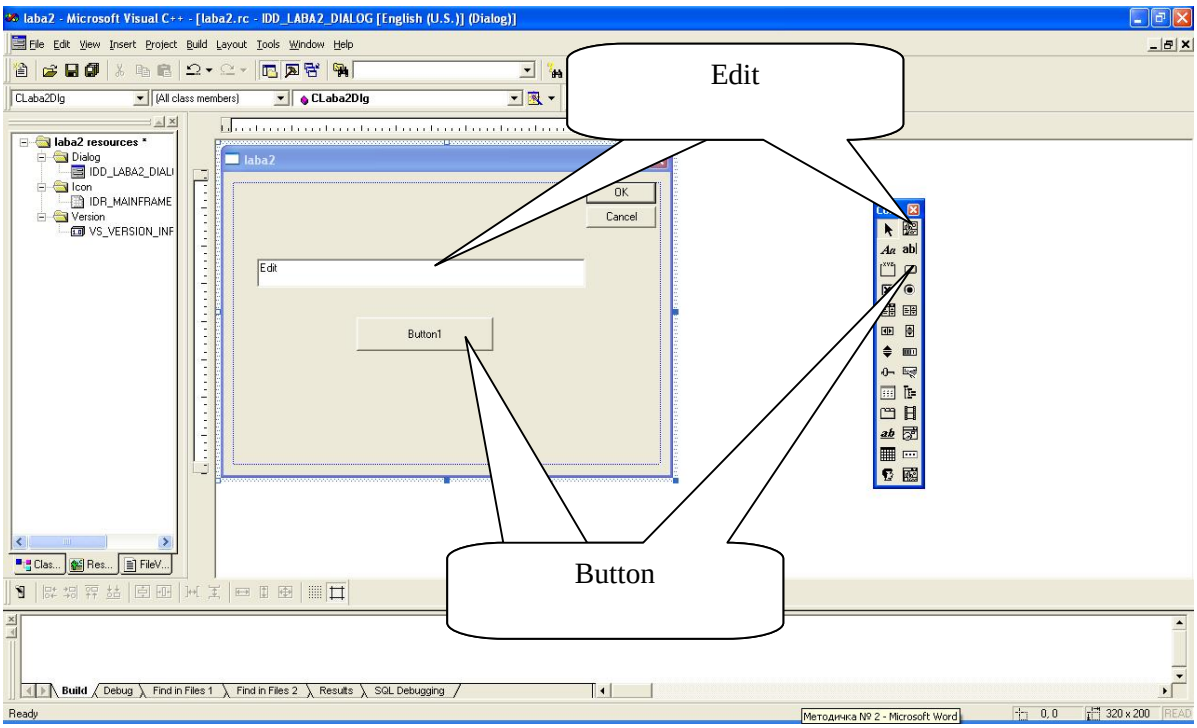


Рис. 75. Розміщення елементів керування

Для зв’язування елемента керування з об’єктом елемента керування треба викликати для нього “майстер” класів (рис. 76). На першій закладці знаходяться елементи карти повідомлень (рис. 77).

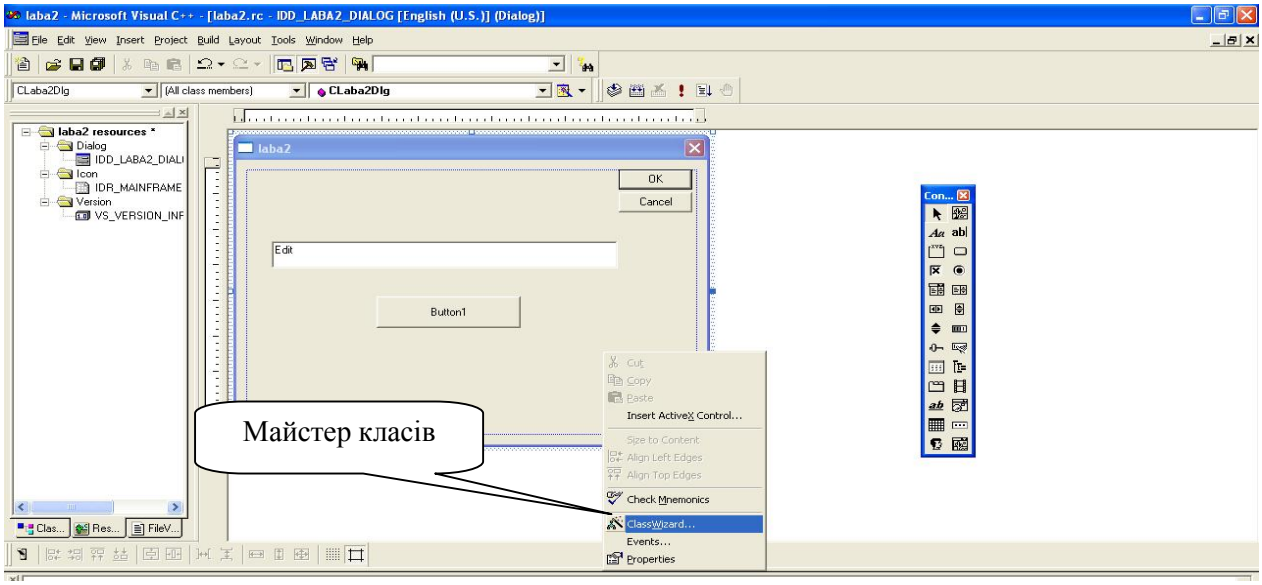


Рис 76. Виклик майстра класів через контекстне меню діалогу

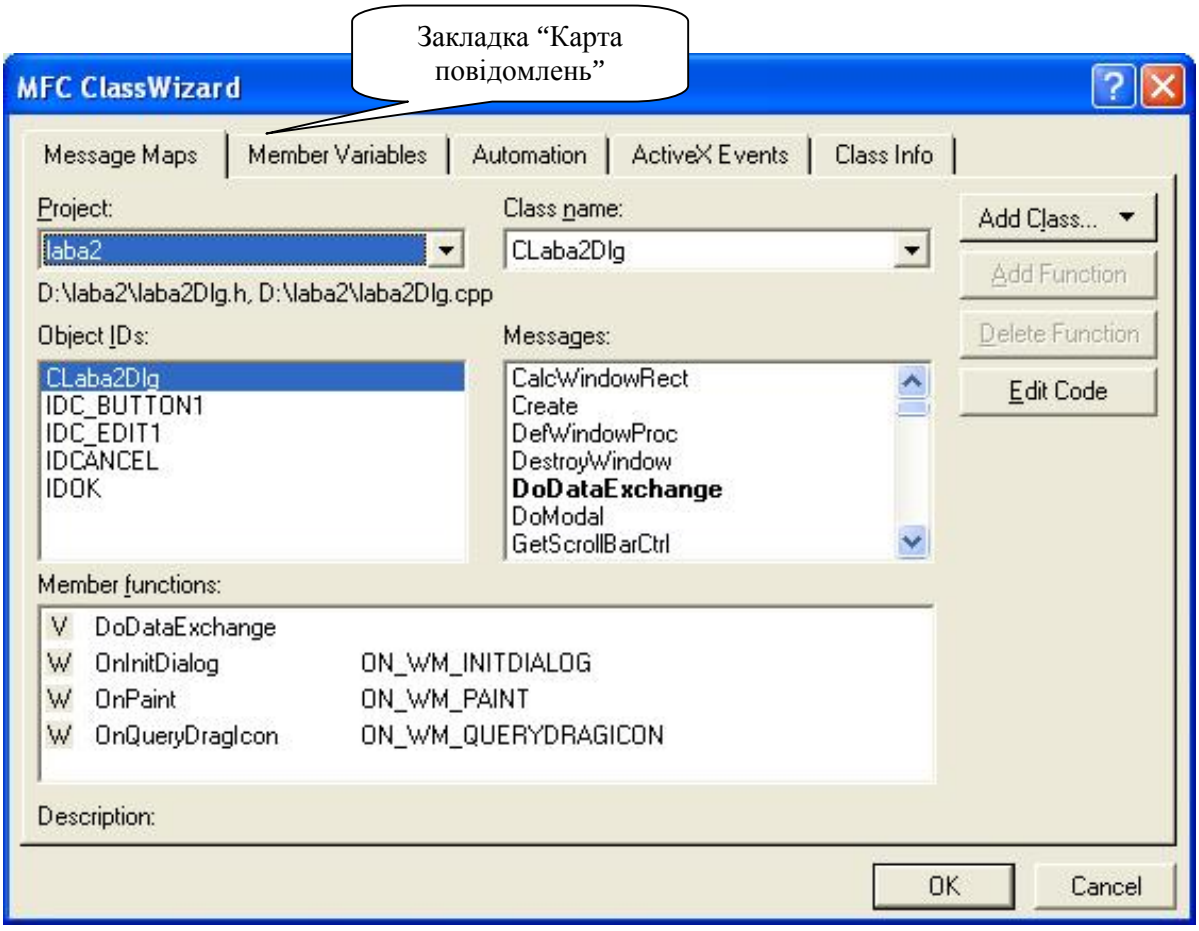


Рис. 77. Вікно “майстра” класів, закладка “Карта повідомлень”

На другій закладці знаходяться змінні класу діалогу (рис. 78).

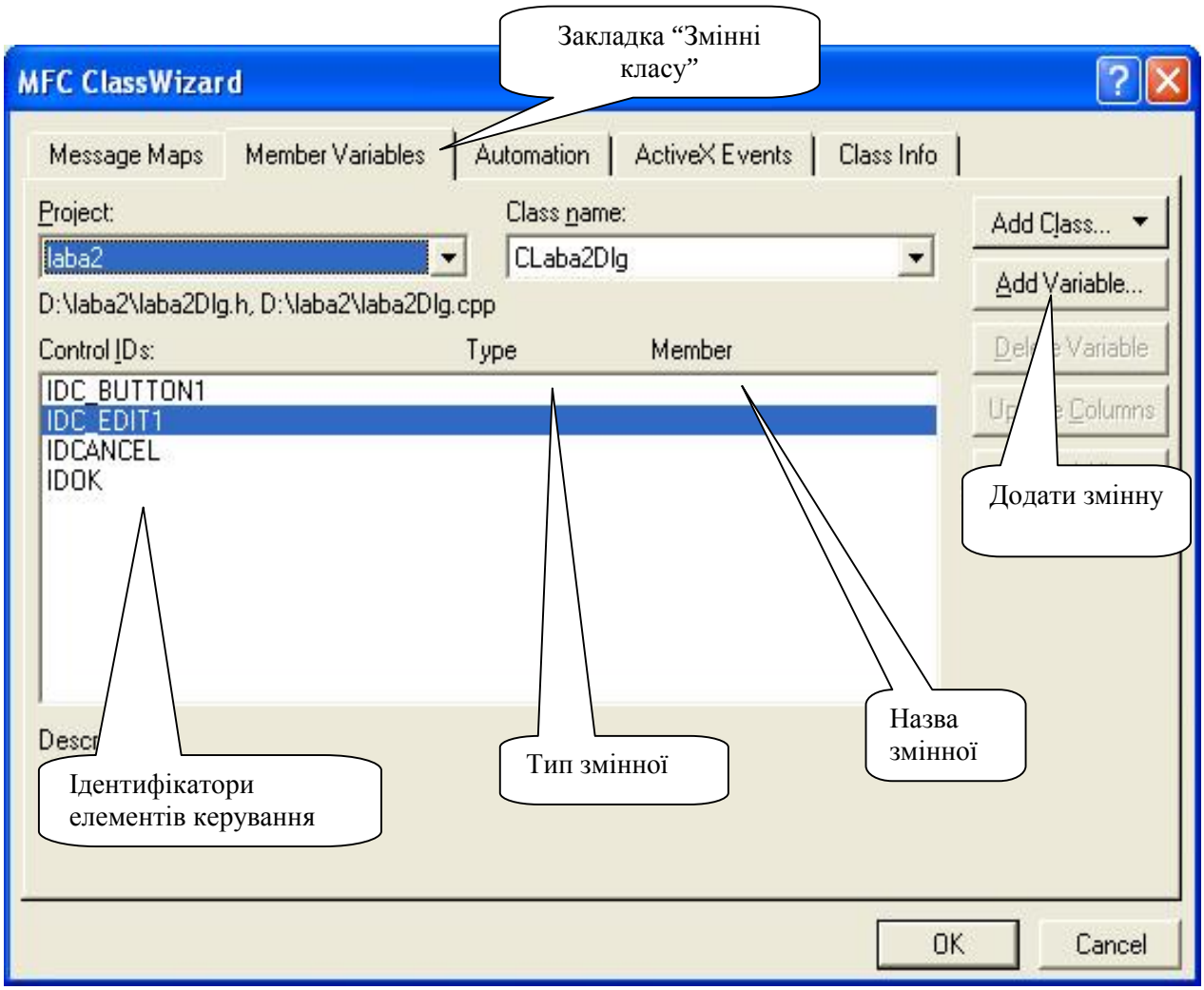


Рис. 78. Вікно “майстра” класів, закладка “Змінні класу”

Для додавання нової змінної треба натиснути “Add Variable…”. З’явиться вікно створення нової змінної класу (рис. 79). У ньому обов’язково треба вказати назву змінної, яка починається на m_, категорію (по значенню або об’єктна) та тип змінної.

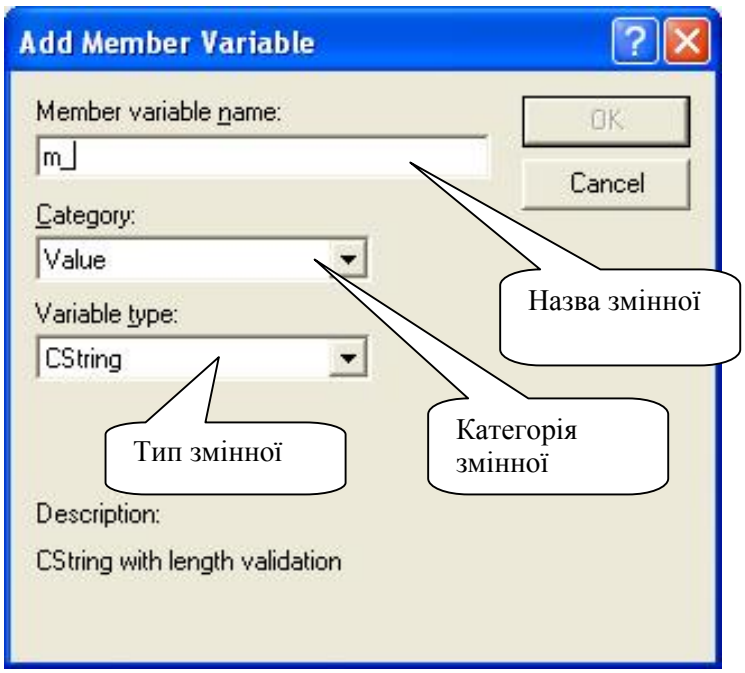


Рис. 79. Вікно створення нової змінної класу

Після створення нової змінної вона з'являється на закладці "Змінні класу" "майстра" класів (рис. 80). Після натиснення "**OK**" можна побачити, що нова змінна з'явилась у середовищі у вікні перегляду класів і в тексті програми (рис. 81).

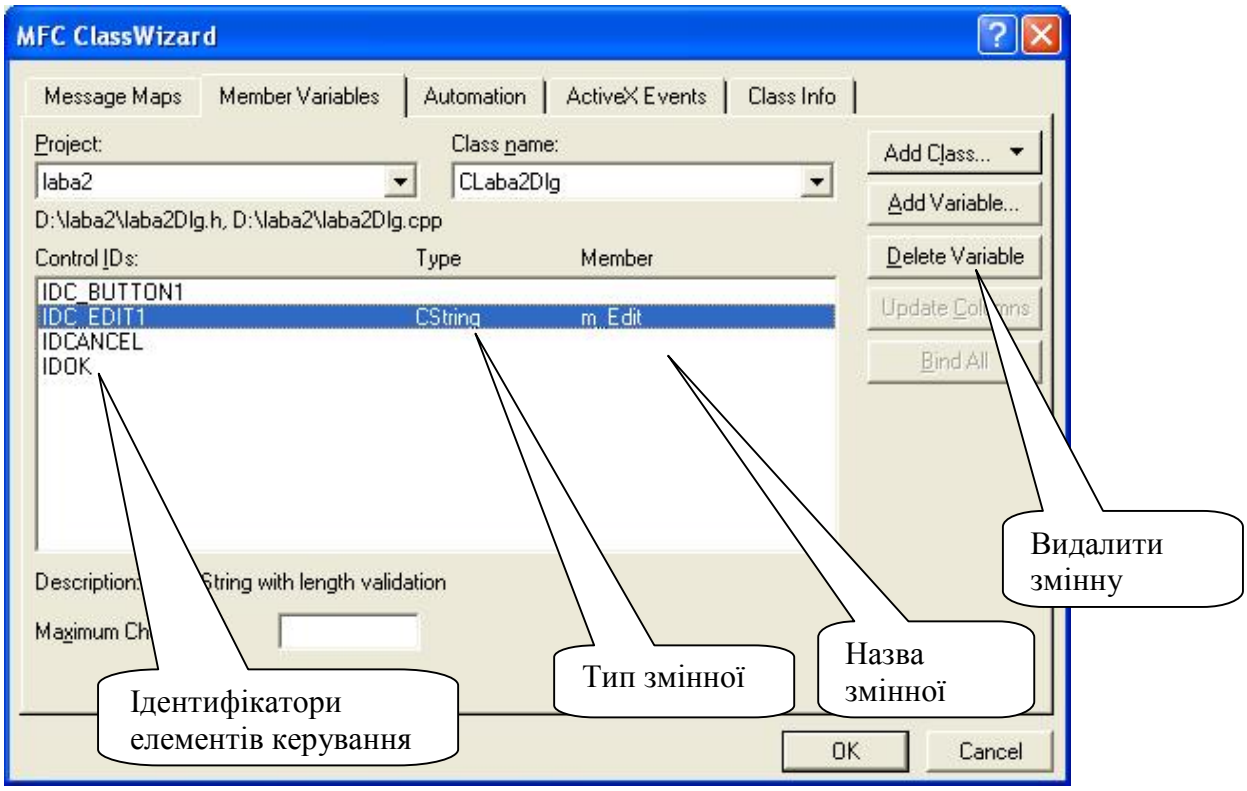


Рис. 80. Вікно "майстра" класів після створення змінної

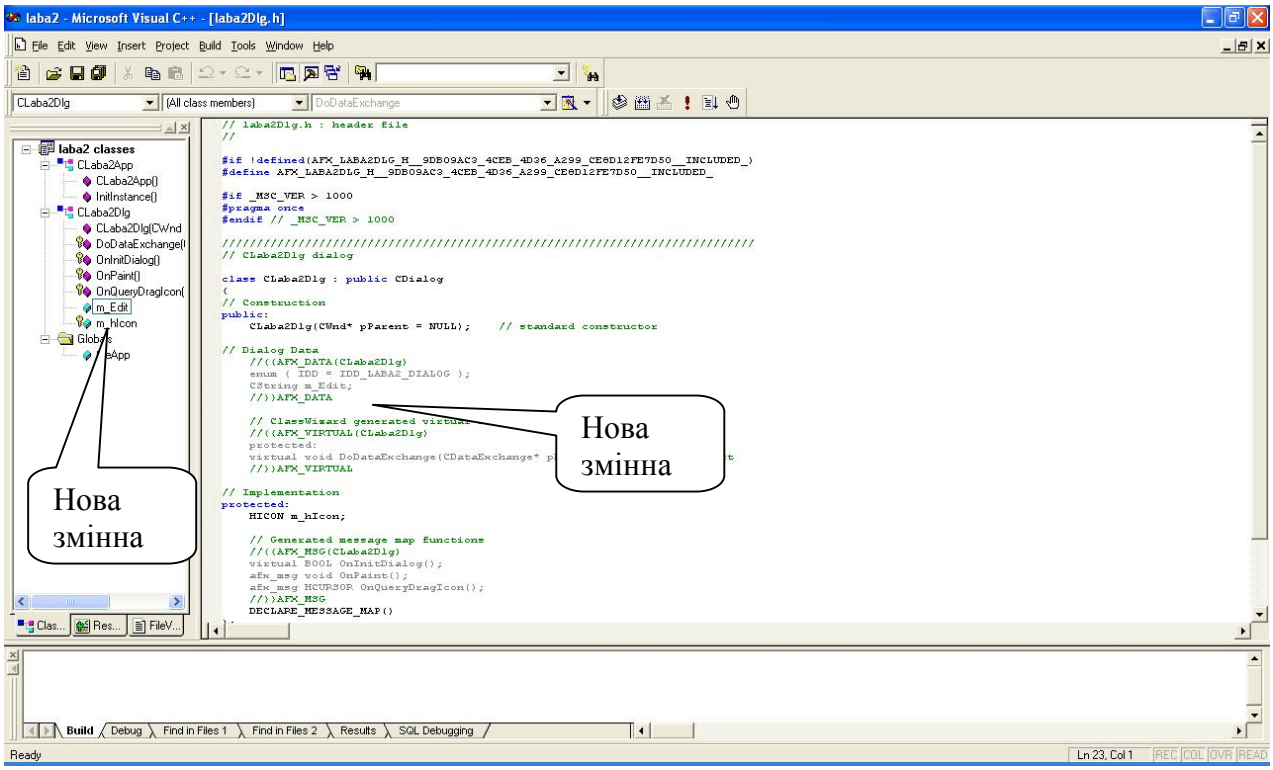


Рис. 81. Вікно середовища з новою змінною

У відповідь на дії користувача в віконній програмі виникають повідомлення. Для обробки повідомлень треба включити повідомлення в карту повідомлень і додати до класу спеціальну функцію – обробник повідомлення. Найбільш коректно ці дії виконує майстер повідомлень (рис. 82). Для додавання нового обробника повідомлень треба вибрати елемент керування, повідомлення якого будуть оброблятися, вибрати повідомлення, для якого потрібен обробник, і натиснути клавішу “Add Function…”.

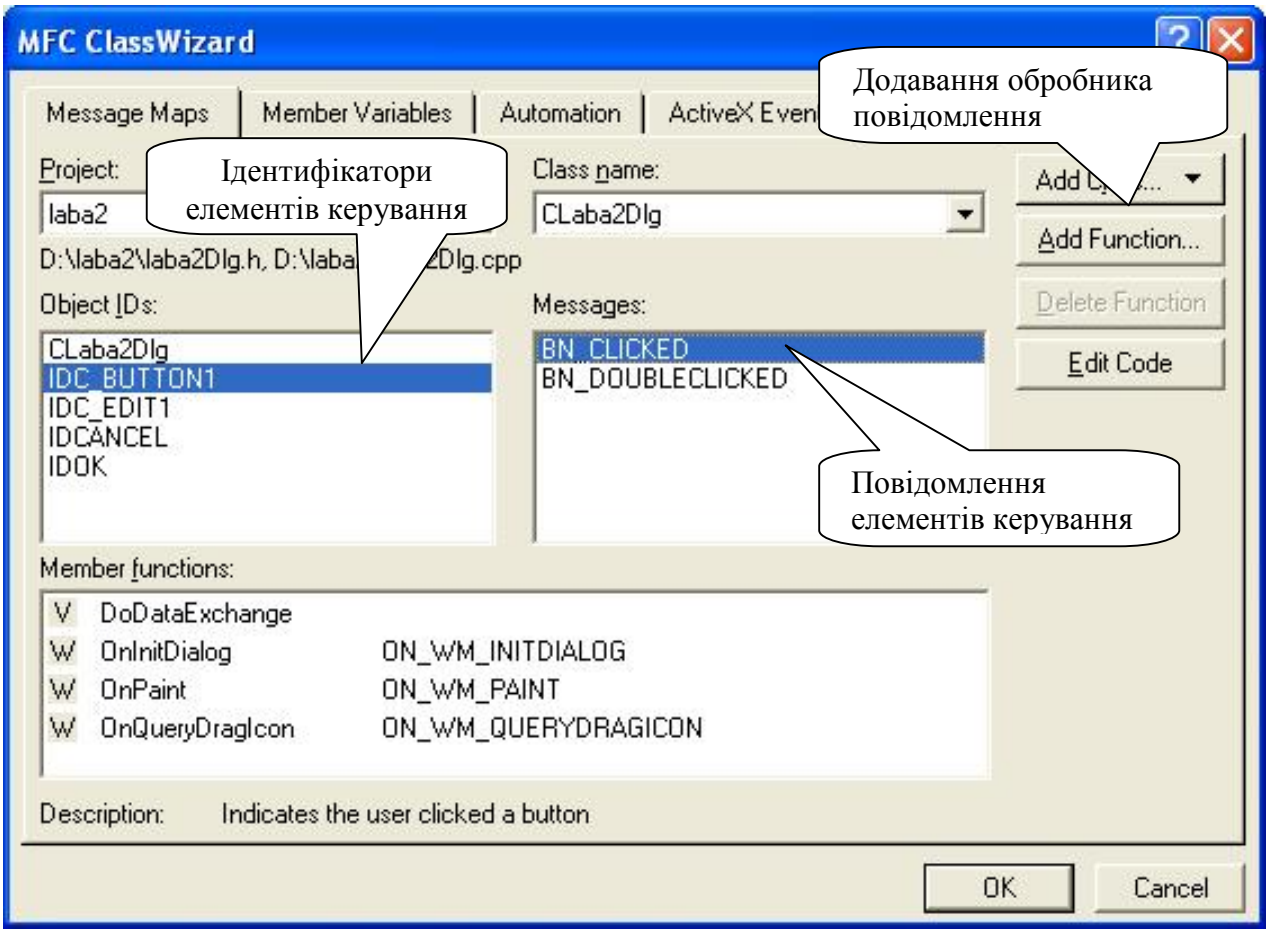


Рис. 82. Вікно "майстра" класів при створенні обробника повідомлення

У вікні додавання нового обробника треба ввести назву функції-обробника (рис. 83), яка обов'язково повинна починатися з On, і натиснути "**OK**".

Рис. 83. Вікно додавання обробника повідомлення

У нижній частині вікна "майстра" класів з'явиться нова функція – член класу і повідомлення, яке ця функція обробляє (рис. 84).

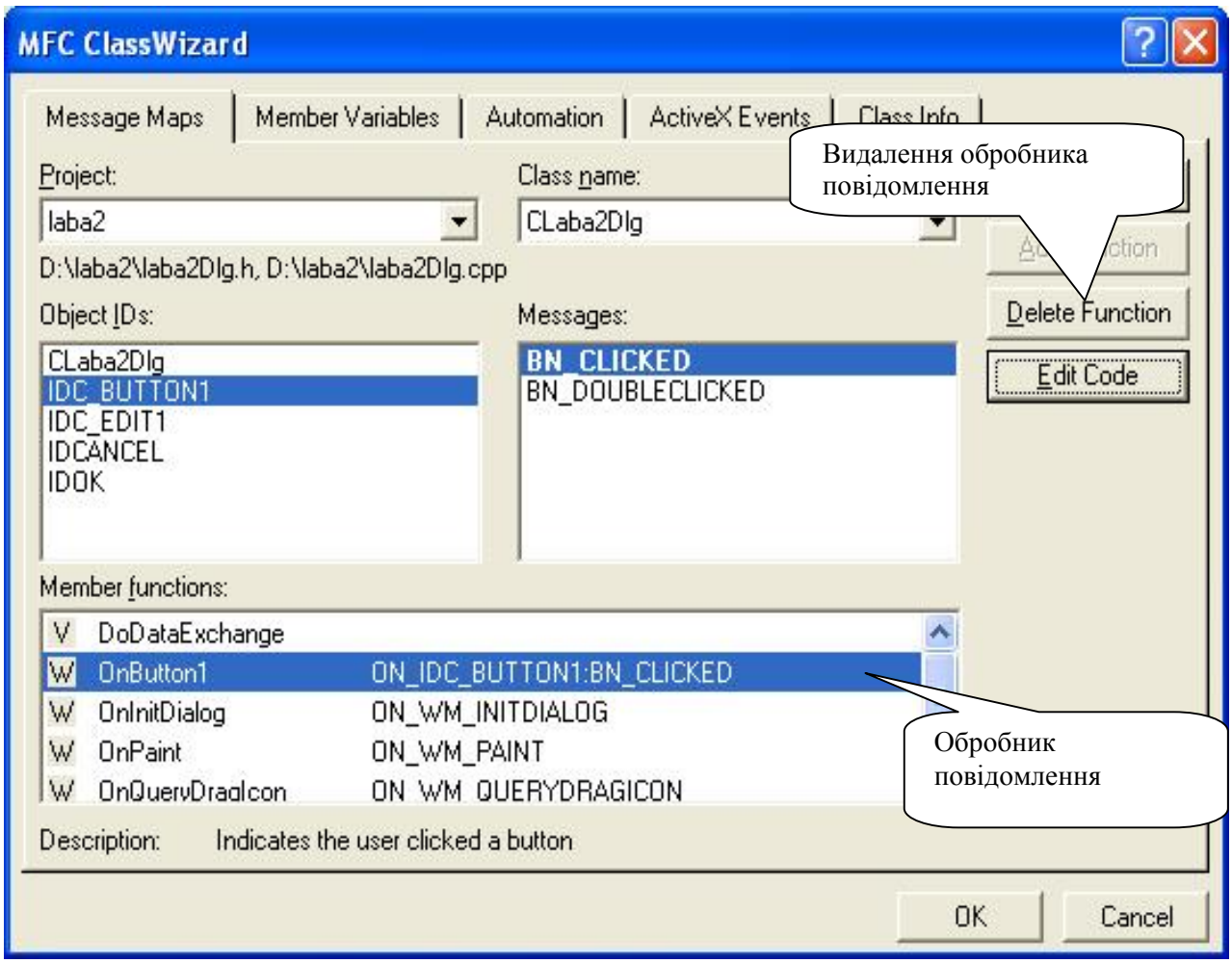


Рис. 84. Вікно "майстра" класів
після створення обробника повідомлення

Після натиснення "**OK**" можна побачити, що нова функція з'явилась в середовищі у вікні перегляду класів і в тексті програми (рис. 85).

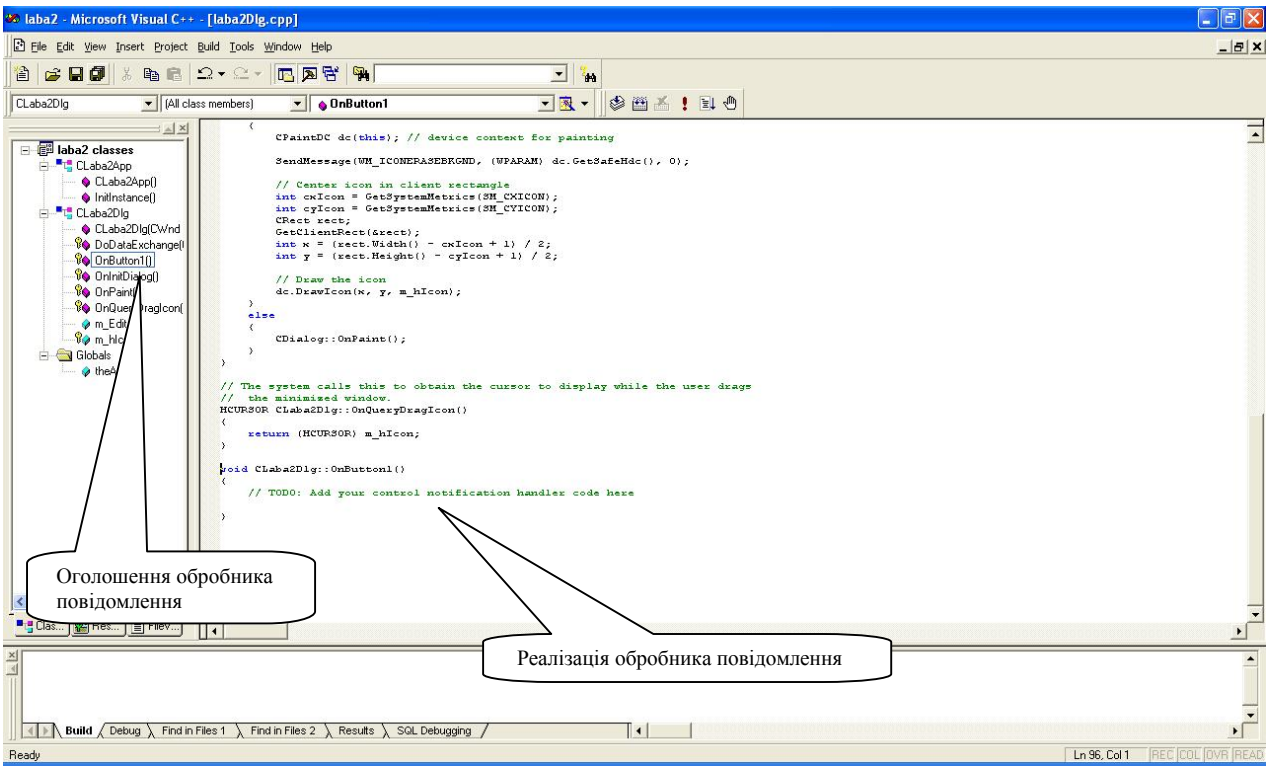


Рис. 85. Вікно середовища з новим обробником повідомлення

Якщо запустити програму на компіляцію і виконання, то вона створить вікно діалогу з елементами керування (рис. 86).

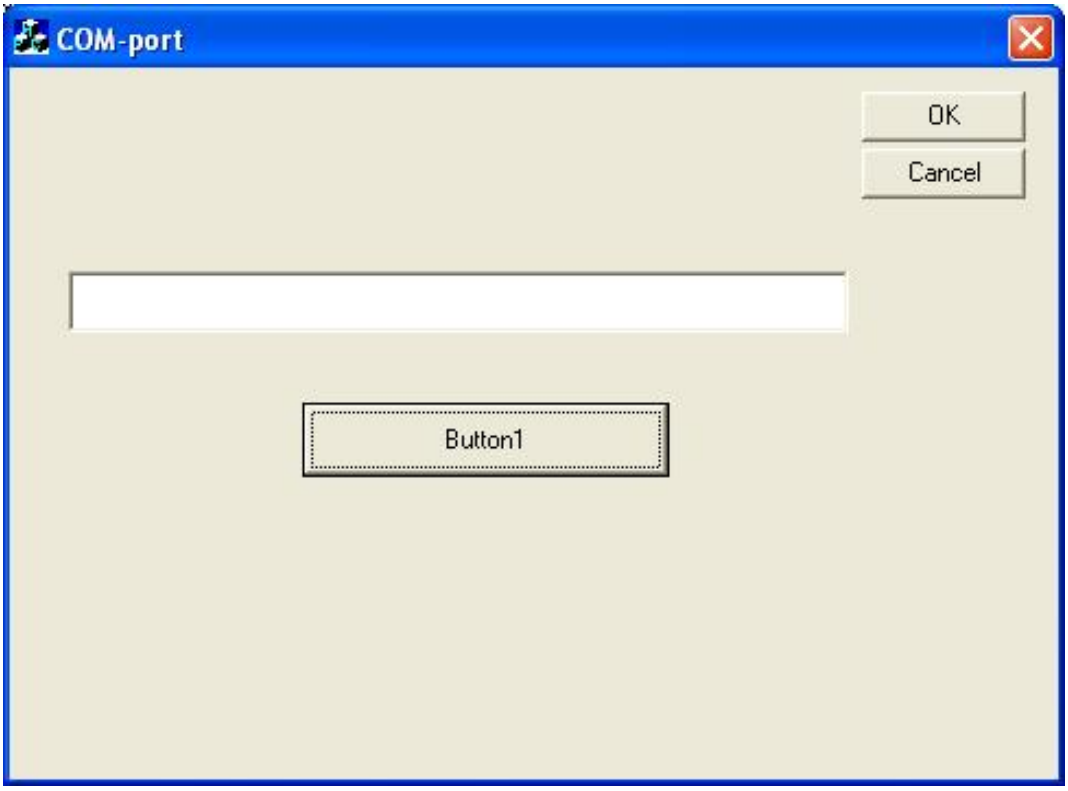


Рис. 86. Вікно програми після виконання

Елемент Edit можна використовувати для введення інформації на передавання. Для цього потрібно в обробнику повідомлення натиснення кнопки Button1 написати реалізацію програми передавання даних з рядка редагування Edit (використовуючи вже створену змінну m_Edit) на USB-порт .

У вікні діалогу потрібно змінити напис на кнопці Button1. Для цього у властивостях даного елементу керування змінюємо поле Caption з “Button1” на “Передача даних” (рис. 87 і рис. 88).

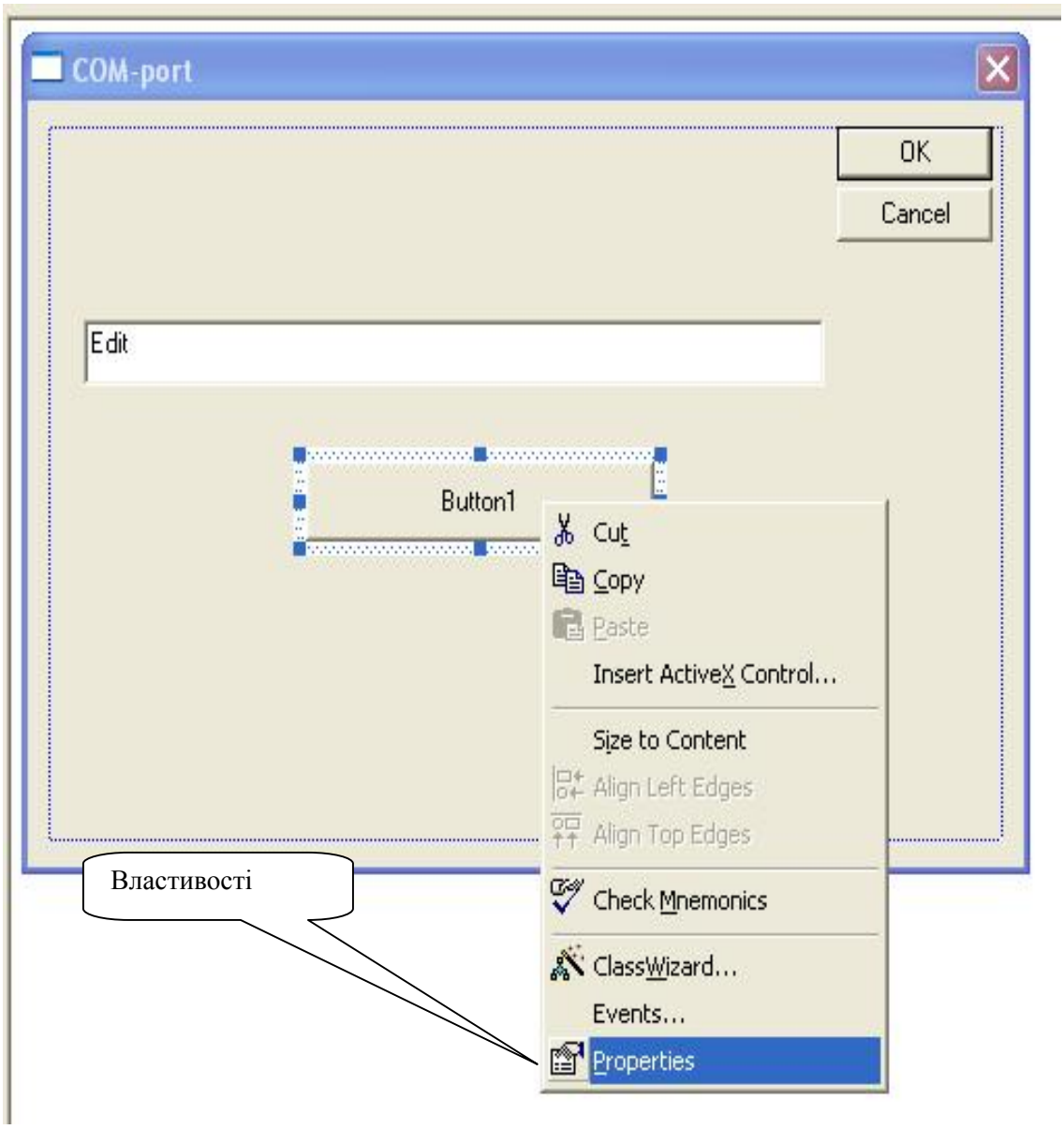


Рис 87. Контекстне меню елемента Button1

Рис 88. Закладка General властивостей елемента Button1

У результаті вікно діалогу має вигляд (рис. 89).

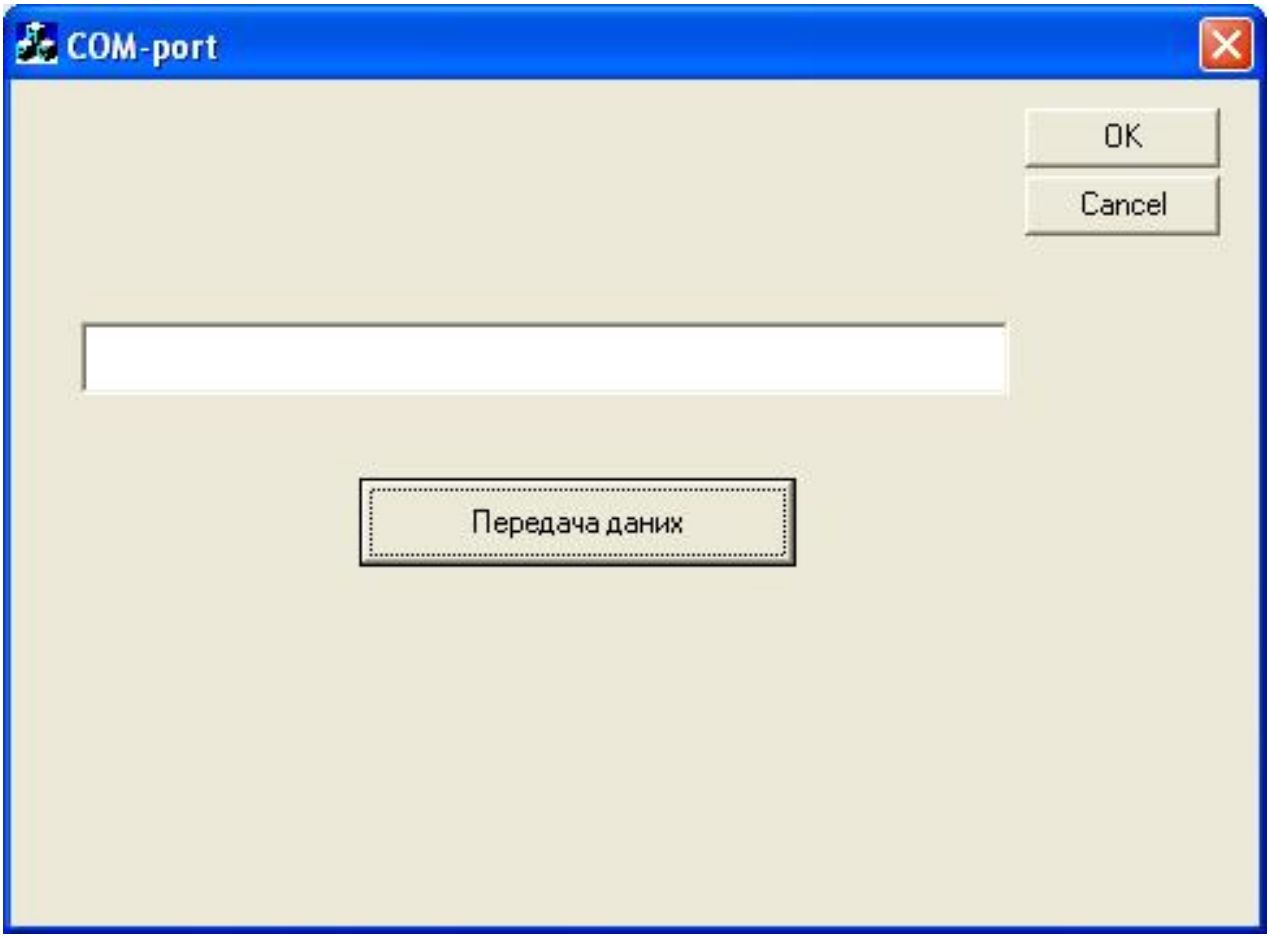


Рис 89. Вікно програми

До нашого проєкту додамо ще одну кнопку “Прочитати з файлу” (рис. 90).

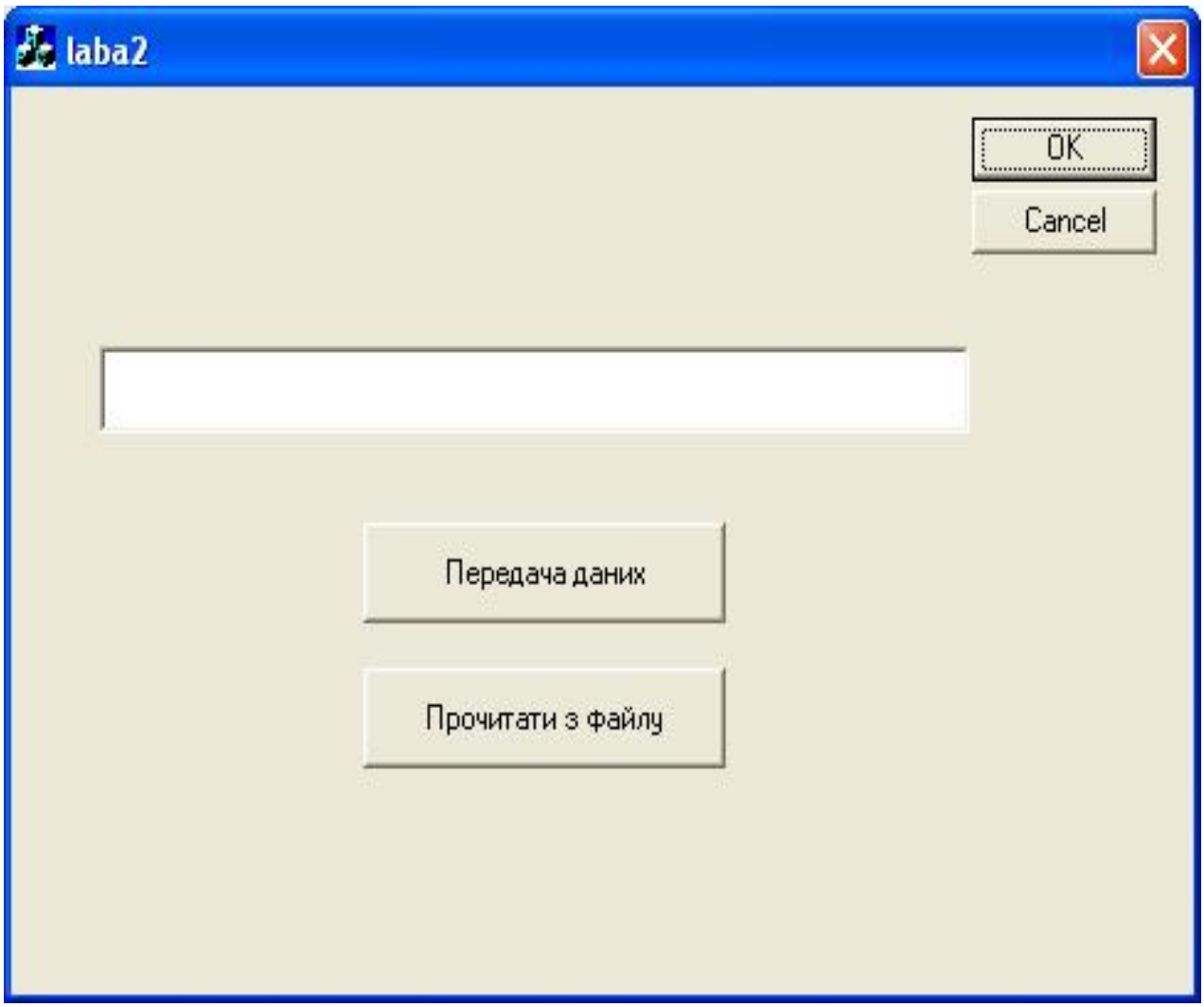


Рис 90. Вікно програми читання файлу

Edit повинен бути багато стрічковий (Multiline) і підтримувати перехід на нову стрічку за допомогою “Want return” (рис. 91 і рис. 92)

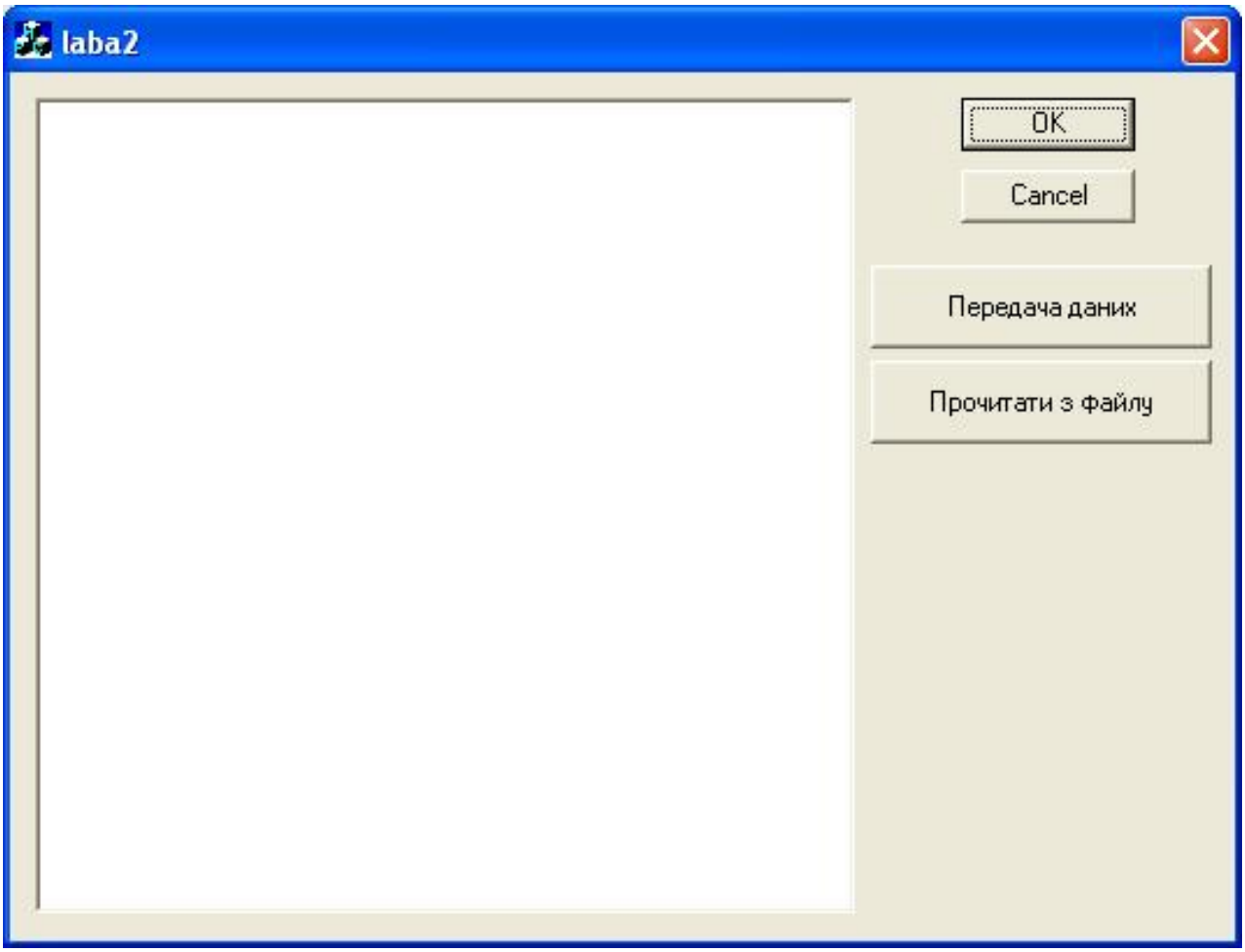


Рис 91. Вікно переходу на іншу стрічку

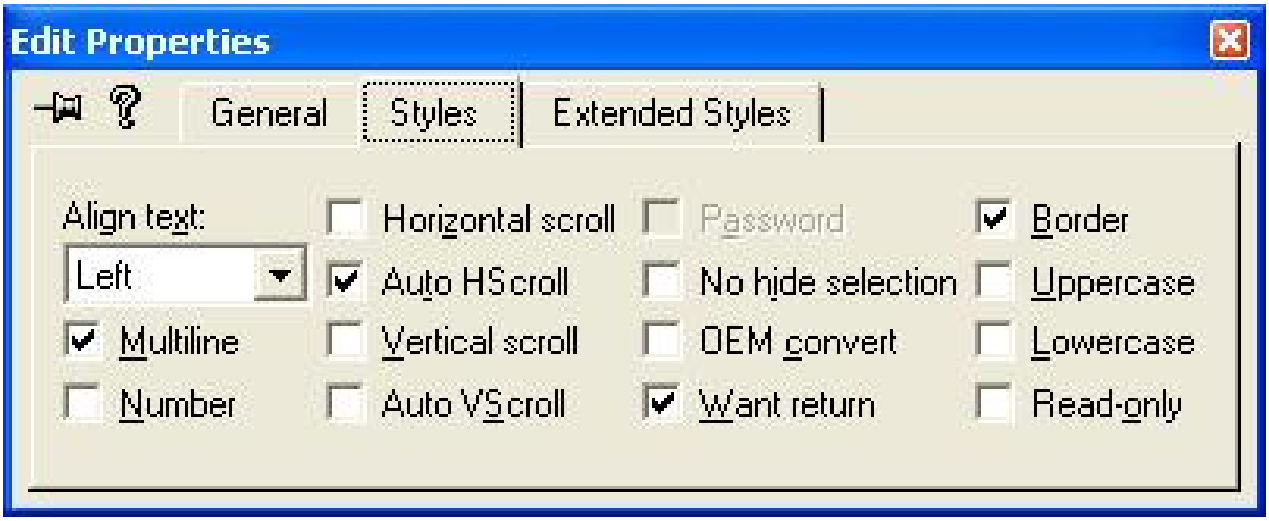


Рис 92. Закладка Styles властивостей елемента Edit1

Далі за допомогою “майстра” класів створимо обробник натиснення клавіші “Прочитати з файлу” (рис. 93).

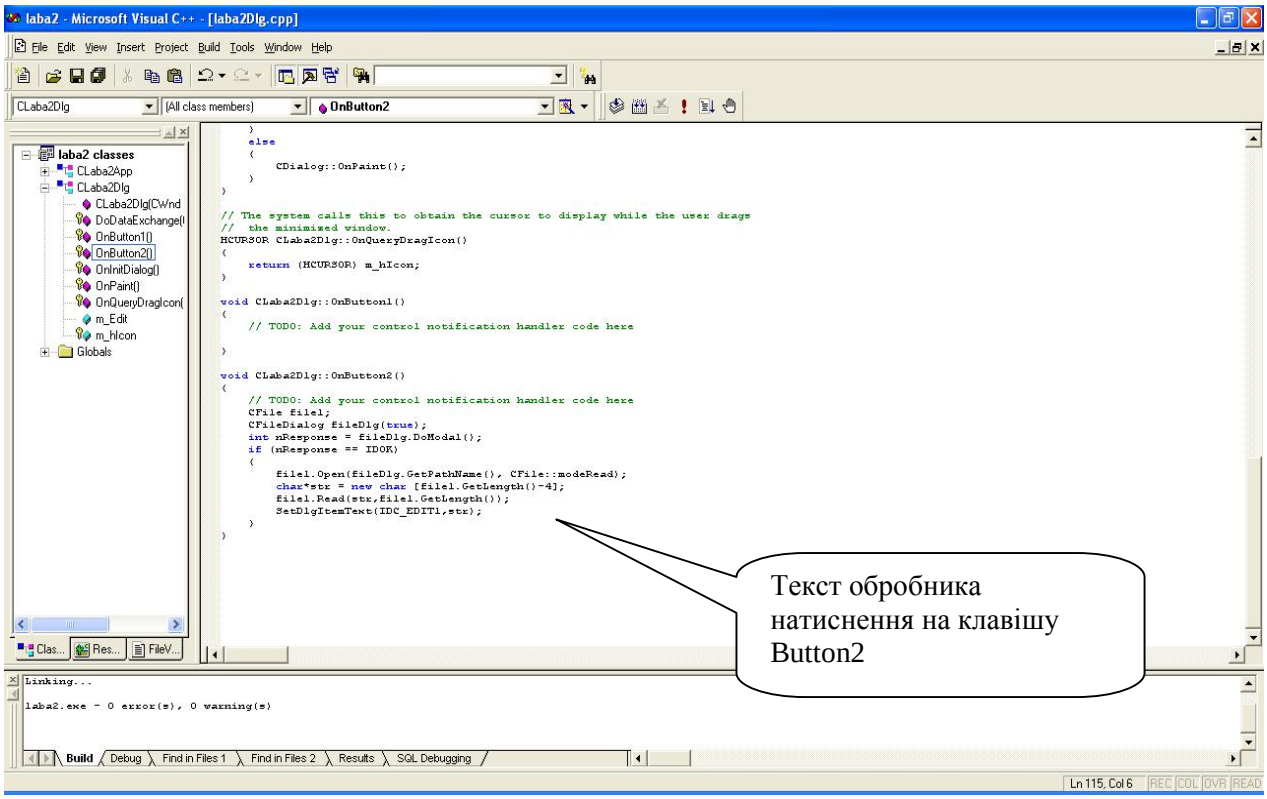


Рис 93. Вікно середовища з текстом програми

Після натиснення клавіші “Прочитати з файлу” на екрані з’являється діалог вибору файлів (рис. 94), який дозволяє здійснювати навігацію по файловій системі. Треба обов’язково вибрати файл і натиснути клавішу “Открыть”. Після натиснення клавіші “Отмена” операція відкривання файлу не відбудеться.

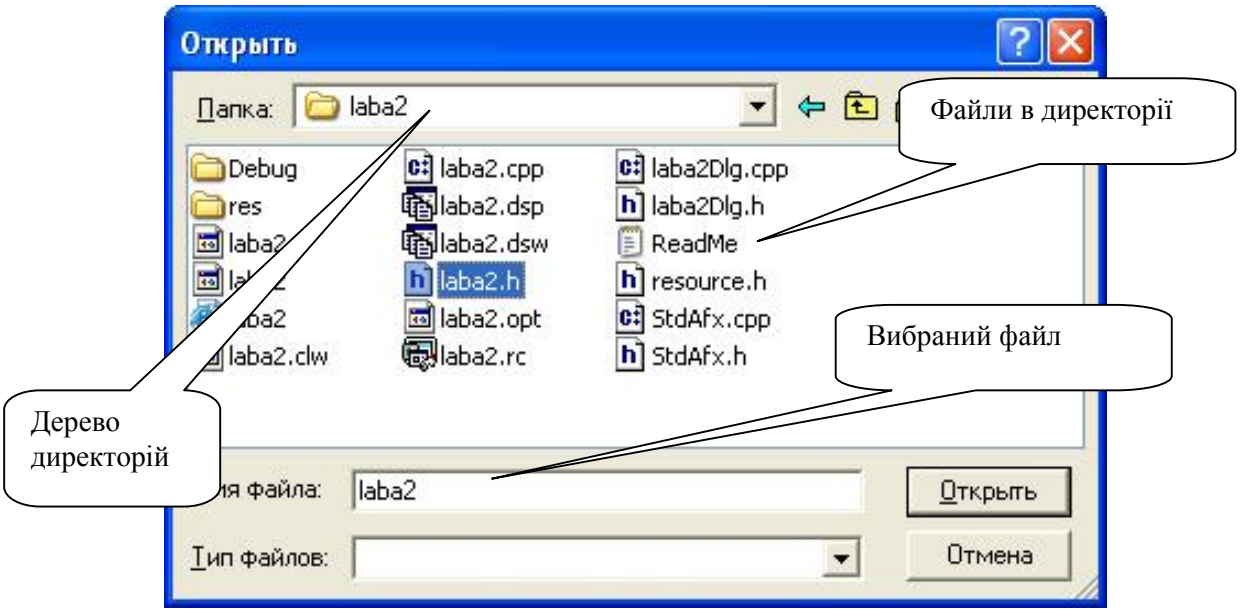


Рис 94. Вікно діалогу вибору файлів

Після натиснення клавіші “Open” з’явиться зміст файлу (рис. 95).

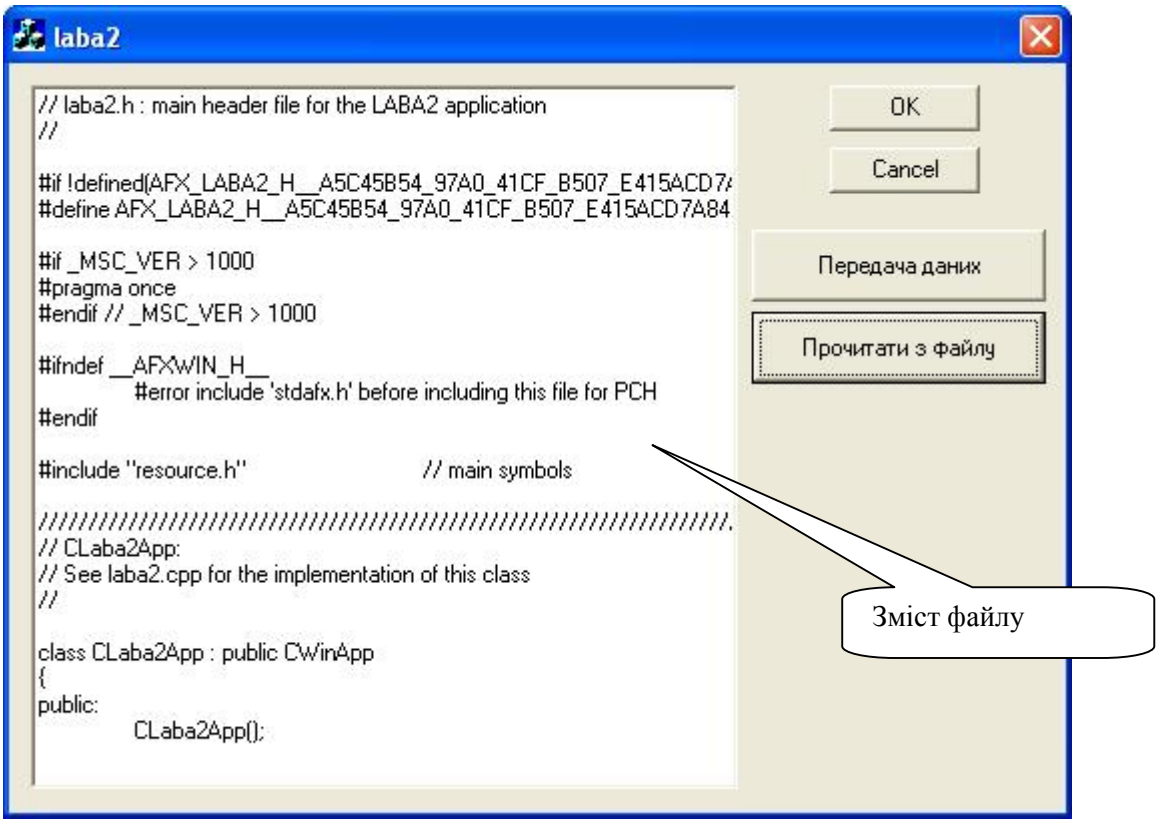


Рис 95. Вікно програми після вибору файла

Тепер залишається тільки в обробнику повідомлення, натисненням кнопки “Передача даних” (Button1) написати функцію, що буде здійснювати передачу інформації з елемента Edit на USB-порт.

ПРИКЛАД ПРОГРАМНИХ ДРАЙВЕРІВ ДЛЯ РОБІТ 5-8

Нижче наводиться приклад програм передачі даних через УПШ-порт та часових діаграм посимвольного кодування в системі NRZI.

```
// laba2Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "laba2.h"
#include "laba2Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CLaba2Dlg dialog

CLaba2Dlg::CLaba2Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CLaba2Dlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CLaba2Dlg)
	m_Edit = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CLaba2Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CLaba2Dlg)
	DDX_Text(pDX, IDC_EDIT1, m_Edit);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CLaba2Dlg, CDialog)
	//{{AFX_MSG_MAP(CLaba2Dlg)
	ON_WM_PAINT()
```

```
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_BUTTON1, OnButton1)
	ON_BN_CLICKED(IDC_BUTTON2, OnButton2)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CLaba2Dlg message handlers

BOOL CLaba2Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CLaba2Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;
```

```
		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CLaba2Dlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

void CLaba2Dlg::OnButton1()
{
	// TODO: Add your control notification handler code here
	char My_Data[] =m_Edit.LoadString; // Дані для запису
	DWORD dwBytesWritten;
	   f or (int i = 0; i < m_Edit.GetLength; i++)  {
	      char *DscT="X:";
	      DscT[0]= char(65+i) ; // наступна буква диску
	       int DR_T = GetDriveType(DscT);  // повертає тип диску
	      cout << DscT << DR_T;  // 3 - жорсткий 5 - CD/DVD 2 – флеш-пам’ять
	            if  (DR_T == 2) // якщо флеш-пам’ять, то
	            {
	               cout << " - FLASH ";
	               char *My_File="X:\MyFile.txt"; // шлях та назва файла, який потрібно
створити
	               My_File[0]=DscT[0]; // замість X записуєм потрібну букву диску
	               HANDLE hFile;
	               //відкриваєм файл, якщо його немає, то створюємо
	              hFile=CreateFile(My_File, GENERIC_READ|GENERIC_WRITE , 0,NULL,
	             OPEN_ALWAYS,0,NULL);
	              ShowMessage(SysErrorMessage(GetLastError())); // результат виконання
	              // Записуєм дані з масиву My_Data
	             WriteFile(hFile, My_Data,sizeof(My_Data), &dwBytesWritten, NULL);
	             ShowMessage(SysErrorMessage(GetLastError()));  // результат виконання
	              // "Закриваем файл" дескриптор.
	             CloseHandle(hFile);
	             ShowMessage(SysErrorMessage(GetLastError()));  // результат выполнение
```

```
                    }
                cout << "\n" ;
}

getc(stdin);
return 0;

}

void CLaba2Dlg::OnButton2()
{
        // TODO: Add your control notification handler code here
        CFile file1;
        CFileDialog fileDlg(true);
        int nResponse = fileDlg.DoModal();
        if (nResponse == IDOK)
   {
                file1.Open(fileDlg.GetPathName(), CFile::modeRead);
                char*str = new char [file1.GetLength()-4];
                file1.Read(str,file1.GetLength());
                SetDlgItemText(IDC_EDIT1,str);
   }
}
```

ЕЛЕКТРОННЕ НАВЧАЛЬНЕ ВИДАННЯ

Парамуд Ярослав Степанович
Миц Андрій Мирославович

ЛАБОРАТОРНИЙ ПРАКТИКУМ
з дисципліни
“ПЕРИФЕРІЙНІ ПРИСТРОЇ, ІНТЕРФЕЙСИ ТА ДРАЙВЕРИ”

Навчальний посібник

Редактор *Ірина Черевко*
Комп’ютерне верстання *Марти Гарасимів*

Режим доступу:
http://eom.lp.edu.ua/textbooks/np-ppid.pdf

Видавець і виготівник: Видавництво Львівської політехніки
Свідоцтво суб’єкта видавничої справи ДК № 4459 від 27.12.2012 р.

вул. Ф. Колесси, 4, Львів, 79013
тел. + 380 32 2584103, факс +380 32 2584101
vlp.com.ua, ел. пошта: vmr@vlp.com.ua