Національний технічний університет України
‘Київський політехнічний інститут’
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

КУРСОВИЙ ПРОЕКТ

дисципліна: «Автоматизація проектування комп’ютерних систем»

тема: «Задача трасуваня. Хвильовий алгоритм в багатошаровій платі»

Виконав Сергієнко Я.М.

Группа ІО-33, Факультет ІОТ,

Залікова книжка № 3324

Допущений до захисту _____

Викладач Чебаненко Тетяна Михайлівна

_____
(підпис викладача)

Київ - 2017

Національний технічний університет України
‘Київський політехнічний інститут’
Факультет інформатики та обчислювальної техніки
Кафедра обчислювальної техніки

КУРСОВИЙ ПРОЕКТ

дисципліна: «Автоматизація проектування комп’ютерних систем»

тема: «Задача трасуваня. Хвильовий алгоритм»

Виконав Сергієнко Я.М.

Группа ІО-33, Факультет ІОТ,

Залікова книжка № 3324

Допущений до захисту _____

Викладач Чебаненко Тетяна Михайлівна

_____

(підпис викладача)

Київ - 2017

ОПИС АЛЬБОМУ

№ сторінки	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість сторінок	№ примірника	ПРИМ.
1			Документація загальна			
2						
3			Розроблена заново			
4						
5	А4	ІАЛЦ.467449.001.ОА	Опис альбому	1		
6						
7	А4	ІАЛЦ.467449.002.ТЗ	Технічне завдання	3		
8						
9	А4	ІАЛЦ. 467449.003.ПЗ	Пояснювальна записка	20		
10						
11	А4	ІАЛЦ. 467449.004.ПЗ	Додаток А. Лістинг	21		
12			програмного додатку			
13						
14	А4	ІАЛЦ. 467449.004.ПЗ	Додаток Б. Блок-схема	1		
15			алгоритму			
16						
17	А4	ІАЛЦ. 467449.004.ПЗ	Додаток В. Інструкція	5		
18			користувача			
19						
20	А4	ІАЛЦ. 467449.004.ПЗ	Додаток Г. Приклад	3		
21			виконання програми			
22						
23						
24						
25						
26						
27						
28						
29						
30						
31						
32						
33						
34						
35						
36						

Зм.	Арк.	№ докум.	Підпис	Дат	ІАЛЦ.467449.001 ОА		
Розробив		Сергієнко			Опис альбому	Лит .	Аркуш 1 / Аркушів 1
Перевірив		Чебаненко				НТУУ «КПІ» ФІОТ гр. ІО-33	
Н. Конт р.							
Зат вердив							

ЗМІСТ

1 Найменування і область застосування

Найменування: “Задача трасування. Хвильовий алгоритм”.

В даному курсовому проекті розглядається реалізація алгоритму трасування плати хвильовим алгоритмом.

Областю застосування розроблюваного програмного забезпечення є проектування друкованих плат з різним ступенем інтеграції та розробка оптимальних шляхів з’єднання контактів.

2 Основа для розробки

Підставою для розробки слугує індивідуальне технічне завдання на курсовий проект.

3 Мета і призначення

Метою розробки програмного продукту є закріплення вмінь та навичок програмування хвильового алгоритму трасування на мовах високого рівня, поглиблення та розширення знань, отриманих при вивчені курсу “Технології проектування комп’ютерних систем – 2. Методологічне забезпечення САПР”. Розроблений програмний продукт має виконувати трасування заданої користувачем плати за допомогою хвильового алгоритму для отримання оптимальних показників довжини з’єднань контактів.

4 Технічні вимоги

4.1 Вимоги до програмної моделі

Програмна модель повинна виконувати трасування заданої користувачем плати за допомогою хвильового алгоритму.

Задача повинна бути промодельована програмним забезпеченням покроково, після цього необхідно представити загальну довжину створених з’єднань.

Програмне забезпечення має давати користувачу можливість змінювати порядок трасування контактів для отримання оптимальних результатів.

4.2 Вимоги до складу і параметрів технічних засобів

Програмне забезпечення повинне бути написане з використанням однієї з мов програмування високого рівня. Результати роботи повинні бути оформлені у вигляді технічної документації на проект згідно вимог чинних ДСТУ.

4.3 Вимоги до надійності

Надійність програмного продукту повинна бути перевірена в результаті тестування на комп'ютерах різної продуктивності, на різних версіях операційної системи Microsoft Windows і з різними вхідними параметрами.

4.4 Вимоги до програмної сумісності

Програмний продукт має працювати в будь-яких середовищах, сумісних з операційною системою Microsoft Windows.

5 Етапи проектування та розробки

- ✓ Одержання завдання на курсовий проект
- ✓ Створення програмного забезпечення
- ✓ Тестування програмного забезпечення
- ✓ Оформлення курсового проекту
- ✓ Захист курсового проекту

6 Перелік текстової і графічної документації

1. Відомість технічного проекту
2. Текстова документація
    а. Технічне завдання
    б. Пояснювальна записка
3. Додатки
    а. Лістинг програмного продукту
    б. Блок-схема алгоритму
    в. Інструкція користувача
    г. Приклад виконання програми

ЗМІСТ

ВСТУП

Однією з основних цілей діяльності людини в сфері створення матеріального продукту – це автоматизація самого процесу створення. Ця проблема особливо актуальна в сфері інформаційних технологій, де об'єкти, що проектуються, мають надзвичайно велику складність.

Зі збільшенням ступеня інтеграції сучасної радіоелектронної апаратури, зростає і складність її проектування. Це, насамперед, пов'язано з ростом розмірності задач, що вирішуються в процесі розробки. Ефективним рішенням цієї проблеми є використання комп'ютерної техніки на всіх етапах створення радіоелектронної апаратури.

Трасування провідників є, як правило, заключним етапом конструкторського проектування електронно-обчислювальних компонентів і є процесом, під час якого визначаються лінії, що поєднують контакти пристрою з однаковими потенціалами або цілі компоненти пристрою. Задача трасування є однією з найбільш складних в загальній проблематиці автоматизації проектування. Це пов'язано з кількома факторами, наприклад, з великою кількістю способів конструктивно-технологічної реалізації з'єднань, для кожного з яких, при алгоритмічному рішенні задачі, застосовуються специфічні критерії оптимізації та обмеження. З математичної точки зору, трасування – найскладніша задача вибору з величезного числа варіантів оптимального рішення.

Одночасна оптимізація всіх з'єднань при трасуванні за рахунок перебору всіх варіантів на даному етапі розвитку обчислювальних засобів неможлива. Тому, в основному, розробляються локально оптимальні методи трасування, коли траса оптимальна лише на даному кроці за наявності раніше проведених з'єднань.

Основне завдання трасування формулюється наступним чином: по заданій схемі з'єднань прокласти необхідну кількість провідників на площині,

щоб реалізувати задані електричні з’єднання з врахуванням попередньо заданих обмежень (ширина провідників, мінімальні відстані між ними і т.д.).

У курсовому проекті з дисципліни “Технології проектування комп’ютерних систем – 2. Методологічне забезпечення САПР” розглядається процес дослідження алгоритму трасування та створення програми моделювання.

Дана робота складається з двох частин; в першій – “Розробка алгоритму трасування” коротко роз’яснюється суть проблеми, надається огляд вже існуючих рішень – різних алгоритмів та пояснюється робота хвильового.

Другий розділ безпосередньо присвячений розробці програми моделювання. Для користувачів написано інструкцію використання, наведену в додатку В. Приведено прикладні інтерфейси (API) всіх класів системи.

Блок-схема хвильового алгоритму трасування наведено в додатку Б, а лістинг програмного забезпечення можна знайти в додатку А.

РОЗДІЛ 1. РОЗРОБКА АЛГОРИТМУ ТРАСУВАННЯ

1.1 Постановка задачі

Вихідною інформацією для рішення задачі трасування з'єднань зазвичай є список контактів, параметри конструкції елементів та комутаційного поля, а також дані по розміщенню елементів. Критеріями трасування можуть бути відсоток виконаних з'єднань, сумарна довжина провідників, кількість монтажних шарів, кількість міжшарових переходів, рівномірність розподілу провідників, мінімальна область трасування і т. д. Часто ці критерії є взаємовиключними, тому оцінка якості трасування ведеться по домінуючому критерію або застосовують адитивну чи мультиплікативну форму оцінювання функції, наприклад наступного вигляду: $F = \sum_{p} \lambda_i f_i$, де F – адитивний критерій; λ_i – ваговий коефіцієнт; f_i – критерій; p – число критеріїв.

Задача трасування завжди має *топологічний* та *метричний* аспекти. Перший з них пов'язаний з вибором допустимого просторового розміщення окремих фрагментів з'єднань без фіксації їх конкретного місцеположення при обмеженнях на число шарів і т.д. Метричний аспект передбачає врахування конструктивних особливостей елементів, з'єднань та комутаційного поля, а також метричних обмежень на трасування.

У відомих алгоритмічних методах рішення задачі трасування метричний аспект присутній завжди, а топологічний – в більшій чи меншій мірі. Всі існуючі методи трасування можна поділити на дві великі групи: послідовні та паралельні.

В *послідовних* методах траси проводяться одна за другою послідовно, до того ж проведення траси *a* на кроці *i - 1* може зробити важчим або навіть неможливим проведення на кроці *i*, оскільки це є перешкодою для прокладання траси *b*. Отже, особливо актуальним для даних алгоритмів є попереднє ранжування (впорядкування) з'єднань до трасування. Існують різні стратегії впорядкування, засновані на довжині з'єднань, степені охоплення з'єднанням

інших з’єднань та інших. Але об’єктивних оцінок ефективності даних методик на даний момент немає, тому найбільш розумним є використання кількох варіантів черговості і вибір з них кращого.

Задачу трасування можна умовно поділити на чотири етапи, що виконуються послідовно один за одним:

1. визначення (встановлення) точок з’єднань;
2. розміщення по шарах;
3. визначення порядку з’єднань;
4. власне трасування провідників.

На першому етапі формується точний список, що вказує, які з’єднання (електричне коло чи фрагменти цього кола) мають бути проведені. На другому етапі проводиться спроба вирішити, де кожне з’єднання може бути розміщене. На третьому етапі проводиться визначення порядку з’єднань для кожного шару, тобто вказується коли кожен провідник на цьому шарі буде проведений. На четвертому етапі дається відповідь, яким чином має бути проведено кожне з’єднання.

Розглянемо вказані етапи детальніше. Вихідна інформація для першого етапу – перелік з’єднань контактів кожного елемента з контактами інших елементів. Одиночні контакти записуються безпосередньо в перелік з’єднань (матрицю кіл чи список). Труднощі в реалізації виникають тоді, коли один контакт необхідно з’єднати з кількома іншими. Для вирішення цієї проблеми використовують алгоритми побудови найкоротших покриваючих дерев *Прима* чи *Штейнера*.

На другому етапі комп’ютер не проводить з’єднань, він лише вирішує, які провідники повинні бути розтрасовані.

При визначенні переліку з’єднань необхідно також вирішувати питання розподілу контактів елементів по контактах розніму. Часто на практиці намагаються виключити перетин при проведенні прямих ліній між контактами елементів і відповідних їм контактах розніму. При трасуванні багато елементів мають логічно еквівалентні контакти. Тому для спрощення трасування переглядають провідники, що підходять до таких контактів, і, при необхідності,

їх замінюють одне одним. Правило при заміні провідників зазвичай включає заборону на перетин прямих ліній.

Після визначення переліку з'єднань, що трасуються, вирішується задача розділення провідників по шарах. Вона зводиться до побудови плоскої укладки графа схеми. Остаточний варіант розміщення провідників по шарах можна покращити, якщо проаналізувати кожен провідник.

На третьому етапі відбувається дослідження кожного шару з метою точного визначення моменту, коли можна буде трасувати кожний провідник, іншими словами – визначення порядку обробки провідників в кожному шарі. Для двох провідників існує правило вибору: якщо контакт *B* з'являється в прямокутнику *A*, тобто в прямокутнику, що має в протилежних кутках контакти провідника, то провідник *B* потрібно трасувати першим. На жаль, це правило не завжди може бути застосоване. Сформулюємо загальне *евристичне правило Айкерса* про порядок трасування контактів. Провідники трасуються в порядку пріоритетних номерів. Пріоритетний номер провідника *V* рівний числу контактів в прямокутнику *W*. На практиці, зазвичай короткі горизонтальні й вертикальні відрізки провідників трасуються першими, далі оброблюються майже горизонтальні та вертикальні провідники, що їх оточують. Насамкінець, трасуються довгі з'єднання, котрі по відношенню до інших частіше за все розміщуються ззовні.

Зазвичай задачу розміщення провідників по шарах розглядають як задачу розфарбування спеціального графа. Далі проводиться дослідження кожного шару з метою вибору порядку обробки з'єднань всередині кожного з них. Після того, як всі з'єднання розподілені по шарах і визначено порядок їх обробки, вирішується задача трасування фрагментів всіх кіл, що й виконується на четвертому етапі.

1.2 Класифікація алгоритмів трасування

Четвертий етап трасування є основним, найбільш трудомістким, визначаючим ефективність й якість виконання всієї задачі трасування. На рис. 1.1 приведено класифікацію алгоритмів цього етапу.

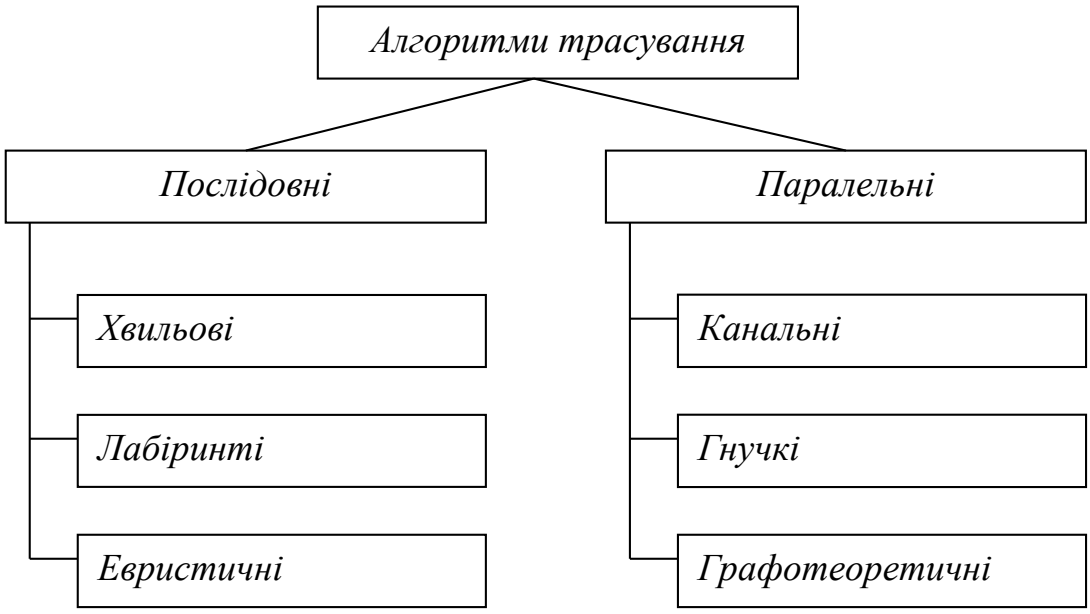


Рис. 1.1. Класифікація алгоритмів трасування

На практиці широко розповсюджені послідовні алгоритми трасування й серед них – хвильовий. Поруч з недоліками, що мають всі алгоритми трасування, що входять до класу послідовних, хвильовий алгоритм пов'язаний зі значними часовими затратами та витратою великої кількості пам'яті комп'ютера.

До *хвильових* відносяться алгоритми, засновані на ідеях динамічного програмування на дискретному робочому полі (ДРП). На сьогодні розроблено величезну кількість модифікацій цього алгоритму, запропонованого Лі ще в 1961 році, проте всі вони зберегли його основний недолік: потреби у великих об'ємах обчислень та оперативної пам'яті. Іншим недоліком цього сімейства алгоритмів є можливість побудови лише з'єднань типу "ввід-вивід". Про деякі з них буде детальніше розписано в пункті "Огляд існуючих рішень".

Часова складність хвильового алгоритму складає $O(M \cdot N)$, де N – число комірок ДРП; M – число точок, що потрібно з'єднати. Ефективне використання

цього алгоритму можливе лише для ДРП з кількістю комірок не більше 10^5. Число нерозведених трас зазвичай не перевищує 10%.

Всі комірки монтажного поля поділяють на заняті й вільні. Занятими називають комірки, в яких вже розміщено провідники, побудовані на попередніх кроках чи знаходяться монтажні виводи елементів, а також комірки, що відповідають межам плати й забороненим для прокладання провідника ділянкам. Щоразу при проведенні нової траси можна використовувати лише вільні комірки, число яких по мірі проведення трас скорочується.

На множині вільних комірок комутаційного поля моделюється хвиля з однієї комірки в іншу, що потім з'єднуються провідником. Першу з них, де зароджується хвиля, називають джерелом, а другу, де згасає – приймачем. Щоб мати можливість слідкувати за проходженням фронту хвилі, її фрагментам на кожному етапі присвоюються певні вагові коефіцієнти: $P_k = P_{k-1} + \varphi(f_1, f_2, \ldots, f_g)$, де P_k та P_{k-1} – ваги комірок k-го й k – 1 фронтів, $\varphi(f_1, f_f, \ldots, f_g)$ – вагова функція, що є показником якості прокладеного шляху, кожен параметр якої $f_i (i = 1,2,\ldots g)$ характеризує шлях з точки зору одного з критеріїв якості (довжини шляху, кількості перетинів і т.д.). На P_k накладають обмеження – ваги комірок попередніх фронтів не повинні бути більше ваг комірок наступних фронтів. Це добре характеризує рис. 1.2.

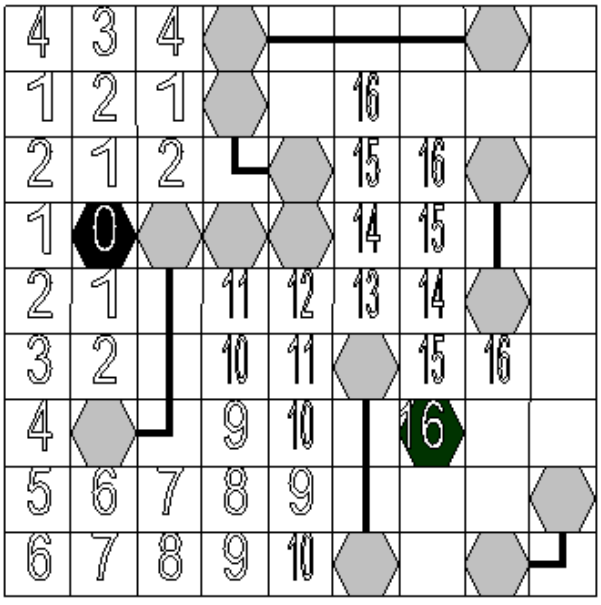


Рис. 1.2. Приклад хвильового алгоритму

Фронт поширюється лише на сусідні комірки, що мають з комірками попереднього фронту або спільний бік, або хоча б одну спільну точку. Процес розповсюдження хвилі продовжується до тих пір, поки її фронт, що розширюється, не знайде приймача, або на певному θ-ому кроці не знайдеться жодної вільної комірки, котра могла би бути включена у наступний фронт, що означає неможливість проведення траси в заданих обмеженнях.

Якщо в результаті поширення хвилі вона досягнула приймача, то виконують так зване “проведення шляху”, що є фактично рухом від приймача до джерела по пройдених на етапі поширення хвилі комірках, слідкуючи за тим, щоб значення P_k монотонно зменшувались. В результаті виходить шлях, що з’єднує дві точки. З опису алгоритму слідує, що всі умови, необхідні для проведення шляху закладаються в правила приписування ваг коміркам.

Щоб виключити неоднозначність при проведенні шляху для випадку, коли кілька точок мають однакову мінімальну вагу, вводять поняття шляхових координат, що задають переваги при виборі траси. Кожен напрям кодують двійковим числом по модулю q, де q – число сусідніх комірок що проглядаються. При цьому вибирається той напрям, який має найменший числовий код. Наприклад, якщо задаються пріоритетним порядком проведення шляхів зверху, справа, знизу й зліва, то коди відповідних шляхових координат будуть 00, 01, 10, 11. Приписування шляхових координат виконують на етапі поширення хвилі.

Основна ідея *лабіринтних* алгоритмів полягає в тому, щоб знайти шлях між двома комірками ДРП за допомогою послідовного обходу перешкод в лабіринті, що утворюється занятими й вільними комірками. На відміну від хвильового алгоритму, пошук має направлений характер, тому час пошуку значно скорочується. Лабіринтні алгоритми зазвичай забезпечують трасування близько 80% з’єднань.

Евристичні алгоритми трасування ефективні, як правило, лише на початкових стадіях трасування, коли на площині мало перешкод. Основна ідея полягає в генерації найдовших незайнятих відрізків у напрямку цілі з евристичними прийомами обходу перешкод.

В *паралельних* алгоритмах трасування спочатку визначається початкове допустиме розміщення трас, що оптимізується по заданому критерію, і лише потім траси фіксуються на комутаційному полі (КП).

В наш час перспективним виглядає використання *канальних* алгоритмів, що засновані на можливості розбиття КП на канали й трасування всередині каналів вертикальними й горизонтальними відрізками. Канал характеризується пропускною здатністю – допустимою кількістю магістралей, що проходять через нього. Канали утворюються, як правило, між рядами встановлених елементів чи компонентів.

Алгоритми працюють дуже швидко, проте для них є актуальною проблема якості отриманих рішень. Варто зазначити, що канальним алгоритмам властивий ієрархічний підхід до трасування, тобто коли вихідна задача розбивається на кілька підзадач. При цьому на кожному з рівнім доцільно використовувати певний алгоритм трасування.

Алгоритми *гнучкого трасування* складаються з двох етапів: макротрасування та мікротрасування. На першому етапі будуються оптимальні моделі топології трас кіл на дискретному топологічному робочому полі (ДТРП) (без жорсткої фіксації). В той час як на другому – моделі геометрії, котрі метрично точно визначають конфігурацію й місце розташування кожної траси. Інакше кажучи, спочатку розміщуються достатньо великі області, в межах яких вони можуть вільно “плавати”, створюючи оптимальні умови для прокладання інших трас. А вже потім відбувається фіксація з’єднань на монтажному просторі з визначенням їх геометричних характеристик.

В *графотеоретичних* методах використовують графові моделі елементів й компонентів схем, на основі яких синтезується топологічний ескіз пристрою, що проектується. На наступному етапі вирішується задача метризації топологічного ескізу.

1.3 Огляд існуючих рішень

На даний момент використовуються різні варіанти хвильового алгоритму, наприклад променевий та маршрутний.

Найпростішим видом хвильового алгоритму є хвильовий алгоритм знаходження найкоротшого шляху без перетину множини занятих й заборонених елементів (ділянок друкованої плати). Його доцільно використовувати при трасуванні з'єднань в одній площині, коли недопустимо виходити за межі цієї області. Визначаються початкова й кінцева точки і моделюється розповсюдження хвилі між ними. Значним недоліком цього алгоритму є те, що він не годиться для трасування багатошарових друкованих плат, провідники прокладаються по краях плати, значне число довгих паралельних провідників являються причиною великої взаємної індуктивності.

Більш довершеним є хвильовий алгоритм прокладки шляху з мінімальним числом перетину. В цьому випадку число перетинів раніше прокладених трас має бути мінімальним. Для подолання недоліку цього алгоритму, що полягає в тому, що траси прямують до однієї з меж плати й притискаються одна до одної, було запропоновано алгоритм для проведення шляхів, що мінімально наближаються одне до одного. Основою алгоритму є умова, при якій елементи даного з'єднання повинні мати мінімум сусідніх елементів, що належать до прокладених раніше трас.

Якщо однією з умов є вимога регулярності з'єднань (один шар горизонтальний, а інший – вертикальний і т.д.), то зручно використовувати хвильовий алгоритм прокладання шляху з мінімальним числом змін напряму, що дозволяє мінімізувати кількість міжшарових з'єднань.

В променевому алгоритмі трасування, запропонованому Л. Б. Абрайтисом, вибір комірок для визначення шляху між точками *A* та *B*, які поєднуються, виконують по раніше заданих напрямах, подібних до променів. Це дозволяє значно скоротити кількість комірок, які переглядає алгоритм, і, відповідно, й час на аналіз та кодування стану, але приводить до зниження

ймовірності знаходження шляху складної конфігурації і ускладнює облік конструкторських вимог до технології друкованої плати.

Роботу алгоритму можна представити наступним чином. Задається число променів, що розповсюджуються з точок *A* та *B*, а також порядок присвоєння шляхових координат (звичайно кількість променів для кожної комірки-джерела приймається однаковою). Промені A(1), A(2), …, A(n) та B(1), B(2), …, B(n) вважаються однойменними, якщо вони розповсюджуються з однойменних джерел *A* чи *B*. Промені A(i) та B(i) є різнойменними по відношенню одне до одного. Розповсюдження променів виконують одночасно з обох джерел до зустрічі двох різнойменних променів в деякій комірці *C*. Шлях проводиться з комірки *C*, і проходить через ті комірки, через які проходили промені.

При розповсюдженні променя може виникнути ситуація, за якої всі сусідні комірки будуть заняті. В такому випадку промінь вважається заблокованим і його розповсюдження припиняється.

Променевий алгоритм варто використовувати для трасування плат з невеликим ступенем заповненням комірок.

1.4 Хвильовий алгоритм трасування

Дискретне поле плати розбивається на три множини, які описуються за допомогою булевих матриць:

1. *C* – множина елементів поля, що вимагають з'єднання між собою;
2. *P* – множина елементів поля, що заборонені для трасування;
3. *S* – множина вільних елементів поля плати.

Математично завдання можна поставити так: необхідно, використовуючи елементи множини *S* з'єднати елементи множини *C* в одне коло, що не перетинає множину *P*.

Процес знаходження мінімального шляху складається з двох етапів:

1. розповсюдження хвилі від джерела до зустрічі з одним із приймачів;
2. визначення шляху від джерела до приймача.

В якості джерела вибирається один з елементів множини *C*, всі інші елементи є приймачами. Нехай через R_k позначимо множину елементів хвилі на кроці k й назвемо його k-тим *фронтом хвилі*, тоді R_{k+1} належить околу R_k.

На кожному кроці робиться перевірка перетину фронту хвилі з приймачем. Як тільки будь-який елемент приймача буде включений у хвилю, процес розповсюдження хвилі завершується і починається процес побудови шляху від найближчого до джерела елемента приймача.

На етапі побудови шляху визначається яким фронтом було досягнуто приймача: горизонтальним чи вертикальним. Від елемента перетину хвилі з приймачем у напрямку, що відповідає останньому фронту хвилі, визначаємо елементи, що не належать множині *P*. При зустрічі з першим же елементом множини точок повороту розглянутого напряму рух припиняється. Всі пройдені елементи включаються в знайдений шлях. Елемент зустрічі приймається за початковий для продовження руху по наступному фронту хвилі.

Таким чином, напрямки змінюються до тих пір, поки не буде досягнуто елемент джерела.

Процес закінчується якщо всі елементи з множини *C* будуть зв’язані в єдине коло, або коли елементи, що лишились в цій множині не можуть бути з’єднані з джерелом.

Блок-схема наведеного хвильового алгоритму трасування одношарової плати розміщена в додатку Б.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Опис програми

Згідно наведеного алгоритму, з використанням середовища Borland Builder 6 та на мові програмування C++ було розроблено програмне забезпечення, лістинг якого наведено в додатку А.

1. Файл `CourseWork.CPP`.

Даний файл є ядром всього проекту, адже містить код для запуску програмного додатку. Він розміщується у функції Win32 `WinMain`. Окрім цього, в даному фрагменті також відбувається ініціалізація всіх форм програми та проводиться обробка можливих виняткових ситуацій.

2. Файл `Main.H`.

Цей файл використовується для опису всіх компонентів головного вікна програми. Серед них варто виділити всі пункти головного та спливаючого меню `TMenuItem`, полотно для малювання плати `TImage`, діалоги відкриття та збереження файлів `TOpenDialog`, `TSaveDialog`. Окремо варто відзначити наявність структури `TPos`, що використовується для збереження даних стосовно позиції контакту на платі у вигляді значень по осі абсцис та ординат та `TContact`, що використовується для опису контактів. Для повного опису предметної області не вистачає лише структури, що позначає з'єднання і яка має назву `track`. Важливим є перерахування `cellType`, що визначає який тип може мати кожна комірка на платі: вільна, контакт, заблокована чи провідник.

3. Файл `Main.CPP`.

Даний файл містить спеціальний масив об'єктів структури `TPos`, що використовується при прокладанні фронту хвилі та реалізацію всіх оголошених в `Main.H` функцій, необхідних для роботи. Серед найважливіших варто згадати наступні:

- ✓ `miNewClick` – обробник події створення нової плати;

- ✓ `resizeImage` – використовується для масштабування зображення плати;
- ✓ `FormResize` – функція реакції вікна на зміну його розмірів;
- ✓ `FormClose` – дії, що виконуються при закритті головного вікна програми;
- ✓ `iMainMouseMove` – функція оновлює стан підказок залежно від того, над якою коміркою знаходиться курсор миші;
- ✓ `showCell` – використовується для малювання комірки;
- ✓ `pmClickPopup` – обробник події натиснення спливаючого меню;
- ✓ `pmiContactClick` – функція використовується для вибору контактів зі списку через спливаюче меню;
- ✓ `pmiLockClick` – функція позначає комірку як заблоковану;
- ✓ `pmiDeleteClick` – функція для видалення об'єкта під курсором;
- ✓ `miTracingClick` – виконання трасування;
- ✓ `miSaveAsClick` – обробник події натиснення на кнопку "Зберегти як";
- ✓ `SaveToFile` – дії для збереження плати в файлі;
- ✓ `miSaveClick` – обробник події для збереження;
- ✓ `miOpenClick` – обробник події натиснення на кнопку "Відкрити";
- ✓ `miEditEnabledClick` – перемикач режиму редагування;
- ✓ `miClearLinksClick` – обробник події для очищення плати від провідників;
- ✓ `setEditMode` – функціональність для встановлення режиму редагування;
- ✓ `clearLinks` – функціональність для очищення плати від провідників;
- ✓ `miOrderClick` – обробник події натиснення на виклик вікна порядку трасування;
- ✓ `addContact` – обробник події додавання контакту на плату;
- ✓ `delContact` – обробник події видалення контакту з плати;

- ✓ `addMenuItem` – обробник події додавання контакту в список спливаючого меню;
- ✓ `delMenuItem` – обробник події видалення контакту зі списку спливаючого меню;
- ✓ `FormCloseQuery` – використовується для запитування користувача про збереження файлу плати при виході;
- ✓ `miAboutClick` – обробник події виклику інформації про програмне забезпечення;
- ✓ `miFAQClick` – обробник події виклику довідки.

4. Файл `Ordering.H`.

Цей файл містить опис всіх компонентів вікна програми, що відповідає за порядок трасування контактів та оголошує функції для управління кнопками, розміщеними на цій формі.

5. Файл `Ordering.CPP`.

Даний файл містить реалізацію оголошених в `Ordering.H` функцій для управління кнопками, що пересувають пари контактів вище чи нижче в списку. Спочатку всі контакти додаються в список за порядком їх додавання на головне вікно, проте пересуваючи певні з них вище в списку можна підняти їх пріоритет при трасуванні й отримати оптимальні показники трасування плати.

6. Файл `Props.H`.

Цей файл містить опис всіх компонентів вікна програми, що відповідає за властивості плати та оголошує кілька службових функцій.

7. Файл `Props.CPP`.

Даний файл призначений для збереження реалізацій функцій, необхідних для отримання кольору фону провідників та контактів (`getMainColor`), кольору самих провідників (`getLinkColor`) та кольору фону заблокованих ділянок (`getLockColor`). Також в ньому присутня реалізація функції конвертування рядка в ціле число з обробкою помилок перетворень та дії при відображенні вікна.

2.2 Моделювання

Для тестування розробленої програми розглянемо приклад, наведений в додатку Г.

Особливістю даного прикладу є те, що при зміні порядку трасування провідників змінюється їх загальна довжина, і, отже, й оптимальність самого процесу трасування.

Також перший варіант трасування не є прийнятним також і тому, що процес був закінчений не до кінця, адже лишились не з’єднані контакти. В той же час в другому варіанті лише за допомогою зміни порядку трасування двох контактів, всі провідники були успішно проведені.

ОСНОВНІ РЕЗУЛЬТАТИ І ВИСНОВКИ ПО РОБОТІ

В ході виконання курсового проекту було вивчено різні методи трасування плат, розглянуто основні етапи трасування плати хвильовим алгоритмом, а також досліджено алгоритм трасування плати хвильовим алгоритмом.

Перевага даного підходу в його простоті, через що він став широко використовуватися у різних САПР та ліг в основу багатьох інших алгоритмів. Недолік – розроблений алгоритм має достатньо великі часові та затрати пам’яті, проте на даному етапі розвитку обчислювальної техніки це не є великою проблемою.

Також в ході роботи було розроблено програмне забезпечення для перевірки роботи хвильового алгоритму, що виконує трасування одношарових плат.

Програмне забезпечення розроблялася з використанням платформи Borland C++ Builder 6. В прикладі роботи програми було показано, що сумарна довжина всіх провідників на платі та саме виконання повного трасування дуже сильно залежить від порядку трасування контактів на платі.

Виконання даного курсового проекту в межах учбового плану вдосконалило навики роботи з алгоритмами трасування та закріпило знання курсу “Технології проектування комп’ютерних систем”.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Деньдобренко Б. Н., Малика А. С. Автоматизация конструирования РЭА: Учебник для вузов. – М. : Высшая школа, 1980. – 384 с. : ил.
2. Автоматизированное конструирование монтажных плат РЭА: Справ. специалиста / А. Т. Абрамов, В. Б. Артемов, В. П. Богданов и др.; под. ред. Л. П. Рябова. – М. : Радио и связь, 1986. – 192 с. : ил.
3. Курейчик В. М. Математическое обеспечение конструкторского и технологического проектирования с применением САПР: Учебник для вузов. – М. : Радио и связь, 1990. – 325 с. : ил.
4. Петухов Г. А. и др. Алгоритмические методы конструкторского проектирования узлов с печатным монтажом / Г. А. Петухов, Г. Г. Смолич, Б. И. Юлин. – М. : Радио и связь, 1987. – 152 с. : ил.
5. Проектирование монтажных плат на ЭВМ / В. А. Калашников, В. М. Курейчик, Б. К. Лебедев и др.; под. ред. К. К. Морозова. – М. : Советское радио, 1979. – 222 с. : ил.
6. Абрайтис Л. Б., Шейнаускас Р. И., Жилевичюс В. А. Автоматизация проектирования ЭВМ. Автоматизированное техническое проектирование конструктивных узлов цифровых устройств. – М. : Советское Радио, 1978. – 272 с. : ил.
7. Курейчик В. М., Глушань В. М., Щербаков Н. И. Комбинаторные аппаратные модели алгоритмов в САПР. – М. : Радио и связь, 1990. – 216 с.
8. Норенков И. П. Основы автоматизированного проектирования, изд. 2-е, пер. и доп. – М. : изд. МГТУ им. Н. Э. Баумана, 2002. – 333 с.
9. Конспект лекцій з курсу “Технології проектування комп’ютерних систем – 2. Методологічне забезпечення САПР”. Лектор – Чебаненко Т. М.

ДОДАТКИ

Додаток А. Лістинг програмного додатку

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("Main.cpp", TMain);
USEFORM("Props.cpp", TProperties);
USEFORM("Ordering.cpp", TOrdering);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {
        try {
                Application -> Initialize();
                Application -> Title = "Wave algorithm";
                Application -> CreateForm(__classid(TTMain), &TMain);
                Application -> CreateForm(__classid(TTProperties), &TProperties);
                Application -> CreateForm(__classid(TTOrdering), &TOrdering);
                Application -> Run();
        } catch (Exception &exception) {
                Application -> ShowException(&exception);
        } catch (...) {
                try {
                        throw Exception("");
                } catch (Exception &exception) {
                        Application -> ShowException(&exception);
                }
        }
        return 0;
}

//---------------------------------------------------------------------------
#ifndef MainH
#define MainH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <ComCtrls.hpp>
#include <list.h>
#include <vector.h>
#include <ImgList.hpp>
#include <ToolWin.hpp>
//---------------------------------------------------------------------------
struct TPos {
        TPos(unsigned c = 0, unsigned r = 0): col(c), row(r) {}
        bool __fastcall operator ==(const TPos & c) {return col == c.col && row == c.row;}
        bool __fastcall operator !=(const TPos & c) {return col != c.col || row != c.row;}
        TPos __fastcall operator +(const TPos & c) {return TPos(col + c.col, row + c.row);}
```

```
        unsigned col;
        unsigned row;
};
//---------------------------------------------------------------------------
class TTMain : public TForm {
__published:
        TMainMenu *mmMain;
        TMenuItem *miScheme;
        TMenuItem *miOpen;
        TMenuItem *miSave;
        TMenuItem *miSaveAs;
        TMenuItem *miProperties;
        TMenuItem *miNew;
        TPanel *pnlMain;
        TImage *iMain;
        TPopupMenu *pmClick;
        TMenuItem *pmiContacts;
        TMenuItem *pmiLock;
        TMenuItem *pmiDelete;
        TStatusBar *sbMain;
        TMenuItem *miActions;
        TMenuItem *miTracing;
        TMenuItem *miOrder;
        TSaveDialog *sdSave;
        TOpenDialog *odOpen;
        TMenuItem *miEditing;
        TMenuItem *miEditEnabled;
        TMenuItem *miClearLinks;
        TMenuItem *pmiSeparator;
        TMenuItem *pmiCurr;
        TMenuItem *pmiNext;
        TMenuItem *miHelp;
        TMenuItem *miFAQ;
        TMenuItem *miAbout;
        TMenuItem *miView;
        void __fastcall miPropertiesClick(TObject *Sender);
        void __fastcall miNewClick(TObject *Sender);
        void __fastcall FormResize(TObject *Sender);
        void __fastcall FormClose(TObject *Sender, TCloseAction &Action);
        void __fastcall iMainMouseMove(TObject *Sender, TShiftState Shift, int X, int Y);
        void __fastcall pmClickPopup(TObject *Sender);
        void __fastcall pmiContactClick(TObject *Sender);
        void __fastcall pmiLockClick(TObject *Sender);
        void __fastcall pmiDeleteClick(TObject *Sender);
        void __fastcall miTracingClick(TObject *Sender);
        void __fastcall miSaveAsClick(TObject *Sender);
        void __fastcall miSaveClick(TObject *Sender);
        void __fastcall miOpenClick(TObject *Sender);
        void __fastcall miEditEnabledClick(TObject *Sender);
        void __fastcall miClearLinksClick(TObject *Sender);
        void __fastcall miOrderClick(TObject *Sender);
        void __fastcall FormCloseQuery(TObject *Sender, bool &CanClose);
        void __fastcall miAboutClick(TObject *Sender);
```

```
        void __fastcall miFAQClick(TObject *Sender);
private:
        void __fastcall resizeImage();
        void __fastcall showCell(TPos pos, bool selected);
        void __fastcall addContact(unsigned numb, TPos pos);
        void __fastcall delContact(unsigned numb, TPos pos);
        void __fastcall addMenuItem(unsigned numb, TPos pos);
        void __fastcall delMenuItem(unsigned numb);
        unsigned __fastcall index(unsigned col, unsigned row) {return row * (brdWidth + 2) + col;}
        unsigned __fastcall index(TPos pos) {return index(pos.col, pos.row);}
        void __fastcall SaveToFile(AnsiString fName);
        void __fastcall setEditMode(bool enabled);
        void __fastcall clearLinks();
        enum cellType {ctClear, ctContact, ctLocked, ctLink};
        cellType *board;
        unsigned *descr;
        unsigned char *links;
        TPos currPos;
        unsigned selectedIdx;
        unsigned currCont;
        bool newContact;
        struct TContact {
                TContact(unsigned n, TPos pos0): numb(n), hasPair(false) {pos[0] = pos0;}
                unsigned numb;
                bool hasPair;
                TPos pos[2];
        };
        vector <TContact> contacts;
        typedef vector <TContact> :: iterator cIter;
        struct track {
                track(unsigned l, TPos p): level(l), pos(p) {}
                unsigned level;
                TPos pos;
        };
        typedef list <track> :: iterator tIter;
        AnsiString savedFile;
        bool boardEdited;
        unsigned totalLength;
public:
        unsigned brdHeight;
        unsigned brdWidth;
        unsigned cellSize;
        bool newBoard;
        bool brdLoaded;
        __fastcall TTMain(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TTMain *TMain;
//---------------------------------------------------------------------------
#endif

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

```
#include <stdio.h>
#include "Main.h"
#include "Ordering.h"
#include "Props.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma hdrstop
#pragma resource "*.dfm"
TTMain *TMain;

const unsigned char dir[4] = {0x04, 0x02, 0x08, 0x01};
const TPos delta[4] = {TPos(-1, 0), TPos(0, 1), TPos(1, 0), TPos(0, -1)};
//---------------------------------------------------------------------------
__fastcall TTMain :: TTMain(TComponent* Owner): TForm(Owner), brdHeight(25), brdWidth(45), cellSize(16) {}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miPropertiesClick(TObject *Sender) {
        if (TProperties -> ShowModal() == mrOk) resizeImage();
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miNewClick(TObject *Sender) {
        newBoard = true;
        if (TProperties -> ShowModal() == mrOk) {
                if (board) delete [] board;
                if (descr) delete [] descr;
                if (links) delete [] links;
                board = new cellType [(brdHeight + 2) * (brdWidth + 2)];
                descr = new unsigned [(brdHeight + 2) * (brdWidth + 2)];
                links = new unsigned char [(brdHeight + 2) * (brdWidth + 2)];
                for (unsigned r = 1; r <= brdHeight; r++)
                        for (unsigned c = 1; c <= brdWidth; c++) {
                                unsigned idx = index(c, r);
                                board[idx] = ctClear;
                                descr[idx] = 0;
                                links[idx] = 0;
                        }
                for (unsigned r = 0; r <= brdHeight + 1; r++) {
                        board[index(0, r)] = ctLocked;
                        board[index(brdWidth + 1, r)] = ctLocked;
                }
                for (unsigned c = 1; c <= brdWidth; c++) {
                        board[index(c, 0)] = ctLocked;
                        board[index(c, brdHeight + 1)] = ctLocked;
                }
                resizeImage();
                miActions -> Enabled = true;
                miProperties -> Enabled = true;
                miSave -> Enabled = false;
                miSaveAs -> Enabled = true;
                brdLoaded = true;
                currCont = 0;
                newContact = true;
                contacts.clear();
                pnlMain -> Caption = "";
```

```
                boardEdited = true;
                savedFile = "";
        }
        newBoard = false;
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: resizeImage() {
        iMain -> Visible = false;
        pnlMain -> Height = brdHeight*cellSize ;
        pnlMain -> Width  = brdWidth*cellSize;
        ClientHeight = pnlMain -> Height + sbMain -> Height;
        ClientWidth  = pnlMain -> Width;
        iMain -> Height = pnlMain -> Height;
        iMain -> Width  = pnlMain -> Width;
        iMain -> Canvas -> Brush -> Color = clWhite;
        iMain -> Canvas -> FillRect(TRect(0, 0, iMain -> Height, iMain -> Width));
        iMain -> Canvas -> Pen -> Style = psDot;
        iMain -> Canvas -> Pen -> Color = clBlack;
        for (unsigned i = 0; i < brdHeight; i++) {
                iMain -> Canvas -> MoveTo(0, i*cellSize);
                iMain -> Canvas -> LineTo(iMain -> Width, i*cellSize);
        }
        for (unsigned i = 0; i < brdWidth; i++) {
                iMain -> Canvas -> MoveTo(i*cellSize, 0);
                iMain -> Canvas -> LineTo(i*cellSize, iMain -> Height);
        }
        for (unsigned c = 1; c <= brdWidth; c++)
                for (unsigned r = 1; r <= brdHeight; r++)
                        showCell(TPos(c, r), false);
        iMain -> Visible = true;
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: FormResize(TObject *Sender) {
        if (ClientHeight > pnlMain -> Height + sbMain -> Height)
                ClientHeight = pnlMain -> Height + sbMain -> Height;
        if (ClientWidth > pnlMain -> Width) ClientWidth = pnlMain -> Width;
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: FormClose(TObject *Sender, TCloseAction &Action) {
        if (board) delete [] board;
        if (descr) delete [] descr;
        if (links) delete [] links;
        contacts.clear();
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: iMainMouseMove(TObject *Sender, TShiftState Shift, int X, int Y) {
        TPos newPos(X/cellSize + 1, Y/cellSize + 1);
        if (newPos != currPos) {
                showCell(newPos, true);
                if (currPos.col && currPos.row) showCell(currPos, false);
                currPos = newPos;
                selectedIdx = index(currPos);
                AnsiString hint;
```

```
                switch (board[selectedIdx]) {
                        case ctClear:
                                hint = "Empty cell";
                                break;
                        case ctContact:
                                hint = "Contact № " + IntToStr(descr[selectedIdx]);
                                break;
                        case ctLink:
                                hint = "Wire between contacts № " + IntToStr(descr[selectedIdx]);
                                break;
                        case ctLocked:
                                hint = "Locked cell";
                                break;
                }
                sbMain -> Panels -> Items[0] -> Text = hint;
                sbMain -> Panels -> Items[1] -> Text = IntToStr(currPos.col) + ":" + IntToStr(currPos.row);
        }
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: showCell(TPos pos, bool selected) {
        TColor color[2];
        switch (board[index(pos)]) {
                case ctClear:
                        color[0] = clWhite;
                        color[1] = clLtGray;
                        break;
                case ctContact:
                        color[0] = TProperties -> getLinkColor();
                        color[1] = TProperties -> getMainColor();
                        break;
                case ctLink:
                        color[0] = TProperties -> getLinkColor();
                        color[1] = TProperties -> getMainColor();
                        break;
                case ctLocked:
                        color[0] = TProperties -> getLockColor();
                        color[1] = TProperties -> getMainColor();
                        break;
        }
        unsigned active = selected ? 1 : 0;
        iMain -> Canvas -> Brush -> Color = color[active];
        iMain -> Canvas -> FillRect(TRect((pos.col - 1)*cellSize + 1,
                (pos.row - 1)*cellSize + 1, pos.col*cellSize, pos.row*cellSize));
        iMain -> Canvas -> Pen -> Color = color[1 - active];
        iMain -> Canvas -> Pen -> Width = 2;
        iMain -> Canvas -> Pen -> Style = psDash;
        switch (board[index(pos)]) {
                case ctContact:
                        iMain -> Canvas -> Ellipse((pos.col - 1)*cellSize + 1,
                                (pos.row - 1)*cellSize + 1, pos.col*cellSize, pos.row*cellSize);
                        break;
                case ctLocked:
                        iMain -> Canvas -> MoveTo((pos.col - 1)*cellSize + 2, (pos.row - 1)*cellSize + 2);
```

```
                        iMain -> Canvas -> LineTo(pos.col*cellSize - 1, pos.row*cellSize - 1);
                        iMain -> Canvas -> MoveTo((pos.col - 1)*cellSize + 2, pos.row*cellSize - 1);
                        iMain -> Canvas -> LineTo(pos.col*cellSize - 1, (pos.row - 1)*cellSize + 2);
                        break;
                case ctLink:
                        unsigned char direction = links[index(pos)];
                        unsigned half = cellSize >> 1;
                        if (direction & 0x08) {
                                iMain -> Canvas -> MoveTo((pos.col - 1)*cellSize + half, (pos.row -
1)*cellSize + half);
                                iMain -> Canvas -> LineTo(pos.col*cellSize, (pos.row - 1)*cellSize + half);
                        }
                        if (direction & 0x04) {
                                iMain -> Canvas -> MoveTo((pos.col - 1)*cellSize + half, (pos.row -
1)*cellSize + half);
                                iMain -> Canvas -> LineTo((pos.col - 1)*cellSize, (pos.row - 1)*cellSize +
half);
                        }
                        if (direction & 0x02) {
                                iMain -> Canvas -> MoveTo((pos.col - 1)*cellSize + half, (pos.row -
1)*cellSize + half);
                                iMain -> Canvas -> LineTo((pos.col - 1)*cellSize + half, pos.row*cellSize);
                        }
                        if (direction & 0x01) {
                                iMain -> Canvas -> MoveTo((pos.col - 1)*cellSize + half, (pos.row -
1)*cellSize + half);
                                iMain -> Canvas -> LineTo((pos.col - 1)*cellSize + half, (pos.row -
1)*cellSize);
                        }
                        break;
        }
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: pmClickPopup(TObject *Sender) {
        TMenuItem *item;
        switch (board[selectedIdx]) {
                case ctClear:
                        pmiContacts -> Enabled = true;
                        pmiLock -> Enabled = true;
                        pmiDelete -> Enabled = false;
                        pmiCurr -> Visible = ! newContact;
                        pmiCurr -> Tag = currCont;
                        pmiNext -> Caption = "New (№ " + IntToStr(currCont + 1) + ")";
                        pmiNext -> Tag = currCont + 1;
                        for (unsigned i = 3; i < (unsigned) pmiContacts -> Count; i++)
                                if ((unsigned) pmiContacts -> Items[i] -> Tag == currCont) {
                                        pmiCurr -> Caption = pmiContacts -> Items[i] -> Caption;
                                        pmiContacts -> Items[i] -> Visible = false;
                                } else pmiContacts -> Items[i] -> Visible = true;
                        break;
                case ctContact:
                case ctLink:
                case ctLocked:
```

```
                        pmiContacts -> Enabled = false;
                        pmiLock -> Enabled = false;
                        pmiDelete -> Enabled = true;
                        break;
        }
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: pmiContactClick(TObject *Sender) {
        board[selectedIdx] = ctContact;
        descr[selectedIdx] = ((TMenuItem *) Sender) -> Tag;
        addContact(((TMenuItem *) Sender) -> Tag, currPos);
        if (Sender == pmiNext) {
                currCont = pmiNext -> Tag;
                newContact = false;
        } else newContact |= Sender == pmiCurr;
        showCell(currPos, true);
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: pmiLockClick(TObject *Sender) {
        board[selectedIdx] = ctLocked;
        showCell(currPos, true);
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: pmiDeleteClick(TObject *Sender) {
        switch (board[selectedIdx]) {
                case (ctLink):
                        if (MessageDlg("Are you sure to remove all wires from scheme?",
                                mtConfirmation, mbOKCancel, 0) == mrOk) clearLinks();
                        break;
                case(ctContact):
                        delContact(descr[selectedIdx], currPos);
                        newContact ^= descr[selectedIdx] == currCont;
                case(ctLocked):
                        board[selectedIdx] = ctClear;
                        descr[selectedIdx] = 0;
                        showCell(currPos, true);
                        break;
        }
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miTracingClick(TObject *Sender) {
        clearLinks();
        bool allIsDone = true;
        cIter currPair = contacts.begin();
        while (currPair != contacts.end()) {
                if (currPair -> hasPair) {
                        unsigned numb = currPair -> numb;
                        TPos srcPos = currPair -> pos[0];
                        TPos dstPos = currPair -> pos[1];
                        list <track> trace;
                        trace.clear();
                        trace.push_back(track(0, srcPos));
                        tIter currStep = trace.begin();
```

```
                        bool reached = false;
                        unsigned cl = trace.begin() -> level;
                        do {
                                TPos step = currStep -> роконs;
                                for (unsigned i = 0; i < 4 && ! reached; i++) {
                                        TPos tmpPos = step + delta[i];
                                        unsigned idx = index(tmpPos);
                                        reached = tmpPos == dstPos;
                                        if (board[idx] == ctClear && descr[idx] == 0) {
                                                if (cl != currStep -> level + 1) {
                                                        Application -> MessageBox(("Wave: " +
IntToStr(cl)).c_str(), "Look", MB_OK);
                                                        cl = currStep -> level + 1;
                                                }
                                                trace.push_back(track(currStep -> level + 1, tmpPos));
                                                descr[idx] = currStep -> level + 1;
                                                showCell(tmpPos, true);
                                        }
                                }
                                if (!reached) currStep++;
                        } while (!reached && currStep != trace.end());
                        if (reached) {
                                unsigned currLevel = currStep -> level;
                                TPos step = dstPos;
                                links[index(dstPos)] = 0;
                                while (currLevel) {
                                        reached = false;
                                        for (unsigned i = 0; i < 4 && ! reached; i++) {
                                                TPos tmpPos = step + delta[i];
                                                unsigned idx = index(tmpPos);
                                                if (board[idx] == ctClear && descr[idx] == currLevel) {
                                                        board[idx] = ctLink;
                                                        descr[idx] = numb;
                                                        links[idx] = dir[i ^ 0x02];
                                                        links[index(step)] |= dir[i];
                                                        showCell(step, false);
                                                        totalLength++;
                                                        reached = true;
                                                        step = tmpPos;
                                                        currLevel --;
                                                }
                                        }
                                }
                                for (unsigned i = 0; i < 4; i++)
                                        if (step + delta[i] == srcPos)
                                                links[index(step)] |= dir[i];
                                showCell(step, false);
                                totalLength++;
                        } else allIsDone = false;
                        do {
                                unsigned idx = index(trace.begin() -> pos);
                                showCell(trace.begin() -> pos, false);
                                if (board[idx] == ctClear) descr[idx] = 0;
```

```
                                trace.pop_front();
                        } while (!trace.empty());
                } else allIsDone = false;
                currPair++;
        }
        Beep();
        AnsiString lenStr = "Wires total length: " + IntToStr(totalLength);
        if (allIsDone) ShowMessage("Wire routing was succesfully ended! " + lenStr);
        else ShowMessage("Wire routing was not ending! " + lenStr);
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miSaveAsClick(TObject *Sender) {
        if (sdSave -> Execute()) {
                SaveToFile(sdSave -> FileName);
                miSave -> Enabled = true;
        }
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: SaveToFile(AnsiString fName) {
        savedFile = fName;
        FILE *out = fopen(fName.c_str(), "wb");
        fwrite(&brdHeight, sizeof(brdHeight), 1, out);
        fwrite(&brdWidth,  sizeof(brdWidth),  1, out);
        for (unsigned r = 1; r <= brdHeight; r++)
                fwrite(&board[index(1, r)], sizeof(board[0]), brdWidth, out);
        for(unsigned r = 1; r <= brdHeight; r++)
                fwrite(&descr[index(1, r)], sizeof(descr[0]), brdWidth, out);
        fclose(out);
        boardEdited = false;
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miSaveClick(TObject *Sender) {
        SaveToFile(savedFile);
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miOpenClick(TObject *Sender) {
        if (odOpen -> Execute()) {
                savedFile = odOpen -> FileName;
                FILE* in = fopen(savedFile.c_str(), "rb");
                fread(&brdHeight, sizeof(brdHeight), 1, in);
                fread(&brdWidth,  sizeof(brdWidth),  1, in);
                if (board) delete [] board;
                if (descr) delete [] descr;
                if (links) delete [] links;
                board = new cellType [(brdHeight + 2)*(brdWidth + 2)];
                descr = new unsigned [(brdHeight + 2)*(brdWidth + 2)];
                links = new unsigned char [(brdHeight + 2)*(brdWidth + 2)];
                for (unsigned r = 0; r <= brdHeight + 1; r++) {
                        board[index(0, r)] = ctLocked;
                        board[index(brdWidth + 1, r)] = ctLocked;
                }
                for (unsigned c = 1; c <= brdWidth; c++) {
                        board[index(c, 0)] = ctLocked;
```

```
                        board[index(c, brdHeight + 1)] = ctLocked;
                }
                for (unsigned r = 1; r <= brdHeight; r++)
                        fread(&board[index(1, r)], sizeof(board[0]), brdWidth, in);
                for (unsigned r = 1; r <= brdHeight; r++)
                        fread(&descr[index(1, r)], sizeof(descr[0]), brdWidth, in);
                fclose(in);
                contacts.clear();
                currCont = 0;
                newContact = true;
                for (unsigned c = 1; c <= brdWidth; c++)
                        for (unsigned r = 1; r <= brdHeight; r++) {
                                unsigned idx = index(c, r);
                                currPos = TPos(c, r);
                                switch (board[idx]) {
                                        case (ctContact):
                                                addContact(descr[idx], currPos);
                                                if (descr[idx] > currCont) {
                                                        currCont = descr[idx];
                                                        newContact = false;
                                                } else
                                                        if (descr[idx] == currCont) newContact = true;
                                                break;
                                        case (ctLink):
                                                board[idx] = ctClear;
                                        default:
                                                descr[idx] = 0;
                                }
                        }
                brdLoaded = true;
                TProperties -> ShowModal();
                resizeImage();
                miActions -> Enabled = true;
                miProperties -> Enabled = true;
                miSave -> Enabled = true;
                miSaveAs -> Enabled = true;
                setEditMode(true);
                pnlMain -> Caption = "";
                boardEdited = false;
        }
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miEditEnabledClick(TObject *Sender) {
        setEditMode(!miEditEnabled -> Checked);
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miClearLinksClick(TObject *Sender) {
        clearLinks();
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: setEditMode(bool enabled) {
        miEditEnabled -> Checked = enabled;
        miClearLinks -> Enabled = enabled;
```

```
        pmClick -> AutoPopup = enabled;
        AnsiString mode = enabled ? "on" : "off";
        sbMain -> Panels -> Items[2] -> Text = "Editing mode: " + mode;
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: clearLinks() {
        for (unsigned c = 1; c <= brdWidth; c++)
                for (unsigned r = 1; r <= brdHeight; r++) {
                        unsigned idx = index(c, r);
                        if (board[idx] == ctLink) {
                                board[idx] = ctClear;
                                descr[idx] = 0;
                                showCell(TPos(c, r), false);
                        }
                }
        totalLength = 0;
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: miOrderClick(TObject *Sender) {
        TOrdering -> lvOrdering -> Items -> Clear();
        for (unsigned i = 0; i < contacts.size(); i++) {
                TListItem *item = TOrdering -> lvOrdering -> Items -> Add();
                item -> Caption = IntToStr(contacts[i].numb);
                item -> SubItems -> Add(IntToStr(contacts[i].pos[0].col) + ":" +
IntToStr(contacts[i].pos[0].row));
                if (contacts[i].hasPair)
                        item -> SubItems -> Add(IntToStr(contacts[i].pos[1].col) + ":" +
IntToStr(contacts[i].pos[1].row));
                else
                        item -> SubItems -> Add("--");
                item -> Data = (void *)i;
        }
        if (TOrdering -> ShowModal() == mrOk) {
                vector <TContact> ordered;
                for(unsigned i = 0; i < contacts.size(); i++)
                        ordered.push_back(contacts[(unsigned)TOrdering -> lvOrdering -> Items -> Item[i] ->
Data]);
                contacts.clear();
                contacts = ordered;
                boardEdited = true;
        }
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: addContact(unsigned numb, TPos pos) {
        boardEdited = true;
        for (cIter i = contacts.begin(); i != contacts.end(); i++)
        if (i -> numb == numb) {
                i -> hasPair = true;
                i -> pos[1] = pos;
                delMenuItem(numb);
                return;
        }
        contacts.push_back(TContact(numb, pos));
```

```
        addMenuItem(numb, pos);
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: delContact(unsigned numb, TPos pos) {
        boardEdited = true;
        cIter i = contacts.begin();
        while (i -> numb != numb) i++;
        if (i -> hasPair) {
                i -> hasPair = false;
                if (i -> pos[1] == pos) i -> pos[0] = pos;
                addMenuItem(numb, pos);
        } else {
                contacts.erase(i);
                delMenuItem(numb);
        }
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: addMenuItem(unsigned numb, TPos pos) {
        TMenuItem *item = new TMenuItem(Application);
        item -> Caption = "Contact № " + IntToStr(numb) + " (" + IntToStr(pos.col) +
                ":" + IntToStr(pos.row) + ")";
        item -> Tag = numb;
        item -> OnClick = pmiContactClick;
        pmiContacts -> Add(item);
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: delMenuItem(unsigned numb) {
        unsigned i = 2;
        while ((unsigned) pmiContacts -> Items[++i] -> Tag != numb);
        TMenuItem *item = pmiContacts -> Items[i];
        pmiContacts -> Delete(i);
        delete item;
}
//---------------------------------------------------------------------------
void __fastcall TTMain :: FormCloseQuery(TObject *Sender, bool &CanClose) {
        if (boardEdited) {
                int answer = MessageDlg("Do you want to save changes in scheme?",
                        mtConfirmation, mbYesNoCancel, 0);
                CanClose = answer != mrCancel;
                if (answer == mrYes)
                        if (savedFile.IsEmpty()) miSaveAsClick(NULL);
                        else SaveToFile(savedFile);
        } else CanClose = true;
}
//---------------------------------------------------------------------------
void __fastcall TTMain::miAboutClick(TObject *Sender) {
        ShowMessage("Software for wire routing. Written by Rudnitsky Miroslav, group IO-01, 2014.");
}
//---------------------------------------------------------------------------
void __fastcall TTMain::miFAQClick(TObject *Sender) {
        ShowMessage("Software for wire routing with user-friendly interface :)");
}
```

```
//---------------------------------------------------------------------------
#ifndef OrderingH
#define OrderingH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <Buttons.hpp>
//---------------------------------------------------------------------------
class TTOrdering : public TForm {
__published:
        TListView *lvOrdering;
        TBitBtn *btnUp;
        TBitBtn *btnDown;
        TBitBtn *btnDone;
        TBitBtn *btnCancel;
        void __fastcall btnUpClick(TObject *Sender);
        void __fastcall btnDownClick(TObject *Sender);
private:
public:
        __fastcall TTOrdering(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TTOrdering *TOrdering;
//---------------------------------------------------------------------------
#endif

//---------------------------------------------------------------------------
#include <vcl.h>
#include "Ordering.h"
//---------------------------------------------------------------------------
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
TTOrdering *TOrdering;
//---------------------------------------------------------------------------
__fastcall TTOrdering::TTOrdering(TComponent* Owner) : TForm(Owner) {}
//---------------------------------------------------------------------------
void __fastcall TTOrdering::btnUpClick(TObject *Sender) {
        if (!lvOrdering -> Selected || !lvOrdering -> Selected -> Index) return;
        unsigned index = lvOrdering -> Selected -> Index;
        lvOrdering -> Items -> Insert(index - 1);
        lvOrdering -> Items -> Item[index - 1] = lvOrdering -> Items -> Item[index + 1];
        lvOrdering -> Items -> Delete(index + 1);
        lvOrdering -> Selected = lvOrdering -> Items -> Item[index - 1];
        btnDone -> ModalResult = mrOk;
}
//---------------------------------------------------------------------------
void __fastcall TTOrdering::btnDownClick(TObject *Sender) {
        if (!lvOrdering -> Selected || lvOrdering -> Selected -> Index == lvOrdering -> Items -> Count - 1)
return;
```

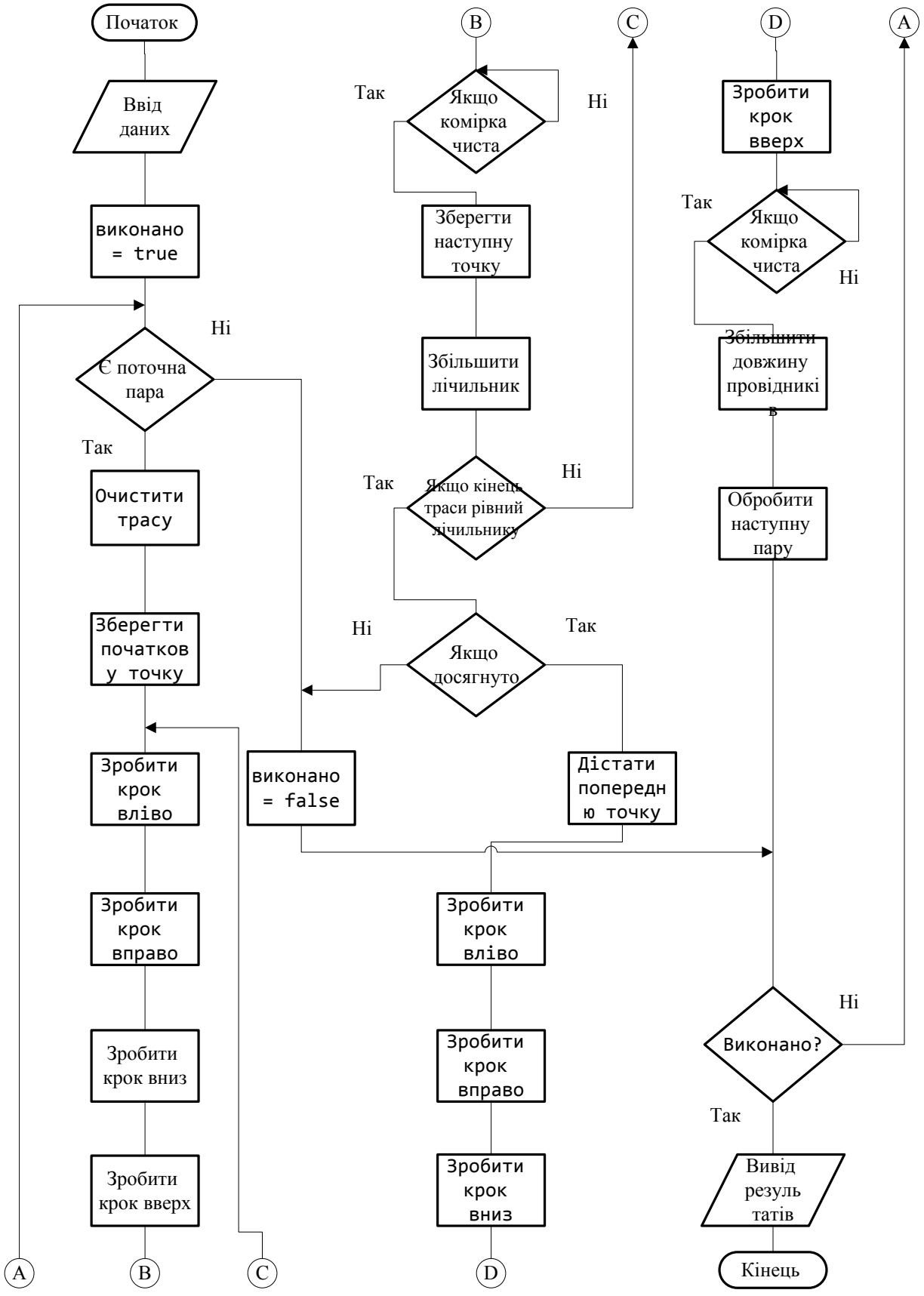
```
        unsigned index = lvOrdering -> Selected -> Index;
        lvOrdering -> Items -> Insert(index + 2);
        lvOrdering -> Items -> Item[index + 2] = lvOrdering -> Items -> Item[index];
        lvOrdering -> Items -> Delete(index);
        lvOrdering -> Selected = lvOrdering -> Items -> Item[index + 1];
        btnDone -> ModalResult = mrOk;
}

//---------------------------------------------------------------------------
#ifndef PropsH
#define PropsH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Buttons.hpp>
//---------------------------------------------------------------------------
class TTProperties : public TForm {
__published:
        TGroupBox *gbSizes;
        TGroupBox *gbPicture;
        TLabel *lHeight;
        TLabel *lWidth;
        TLabel *lCellSize;
        TEdit *eHeight;
        TEdit *eWidth;
        TEdit *eCellSize;
        TColorBox *colorLocked;
        TColorBox *colorLink;
        TLabel *lColorLocked;
        TLabel *lColorLink;
        TBitBtn *btnApply;
        TBitBtn *btnCancel;
        TColorBox *colorMain;
        TLabel *lColorMain;
        void __fastcall btnApplyClick(TObject *Sender);
        void __fastcall FormShow(TObject *Sender);
        TColor __fastcall getLockColor();
        TColor __fastcall getLinkColor();
        TColor __fastcall getMainColor();
private:
        bool __fastcall tryStrToInt(TCustomEdit *edit, unsigned &var, unsigned max = 0);
public:
        __fastcall TTProperties(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TTProperties *TProperties;
//---------------------------------------------------------------------------
#endif

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

```
#include <vcl.h>
#include "Props.h"
#include "Main.h"
//---------------------------------------------------------------------------
#pragma hdrstop
#pragma package(smart_init)
#pragma resource "*.dfm"
TTProperties *TProperties;
//---------------------------------------------------------------------------
__fastcall TTProperties :: TTProperties(TComponent* Owner): TForm(Owner) {}
//---------------------------------------------------------------------------
bool __fastcall TTProperties :: tryStrToInt(TCustomEdit *edit, unsigned &var, unsigned max) {
        if (!edit -> Enabled) return true;
        int val = 0;
        try {
                val = StrToInt(edit -> Text);
        } catch(EConvertError *) {}
        if ((val > 0) && (!max || ((unsigned) val < max))) {
                var = val;
                return true;
        }
        ShowMessage("Illegal value!");
        ModalResult = mrNone;
        edit -> Text = IntToStr(var);
        return false;
}
//---------------------------------------------------------------------------
void __fastcall TTProperties :: btnApplyClick(TObject *Sender) {
        if (!tryStrToInt(eHeight, TMain -> brdHeight)) return;
        if (!tryStrToInt(eWidth, TMain -> brdWidth)) return;
        if (!tryStrToInt(eCellSize, TMain -> cellSize, 50)) return;
}
//---------------------------------------------------------------------------
void __fastcall TTProperties :: FormShow(TObject *Sender) {
        eHeight -> Enabled = TMain -> newBoard;
        eWidth -> Enabled = TMain -> newBoard;
        eCellSize -> Enabled = TMain -> newBoard || TMain -> brdLoaded;
        eHeight -> Text = IntToStr(TMain -> brdHeight);
        eWidth -> Text = IntToStr(TMain -> brdWidth);
        eCellSize -> Text = IntToStr(TMain -> cellSize);
}
//---------------------------------------------------------------------------
TColor __fastcall TTProperties :: getLockColor() {
        return colorLocked -> Selected;
}
//---------------------------------------------------------------------------
TColor __fastcall TTProperties :: getLinkColor() {
        return colorLink -> Selected;
}
//---------------------------------------------------------------------------
TColor __fastcall TTProperties :: getMainColor() {
        return colorMain -> Selected;
}
```

Додаток Б
Блок-схема алгоритму



Зм.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ 467449.004 ПЗ		
Розробив		Сергієнко			Блок-схема алгоритму трасування	Лит.	Аркуш 1 / Аркушів 1
Перевірила		Чебаненко Т.М.					
Н. контр					НТУУ “КПІ” ФІОТ гр. ІО-33		
Затвердив							

Додаток В. Інструкція користувача

Після запуску програмного додатку, перед користувачем з’явиться головне вікно програми, зображене на рис. 2.1.

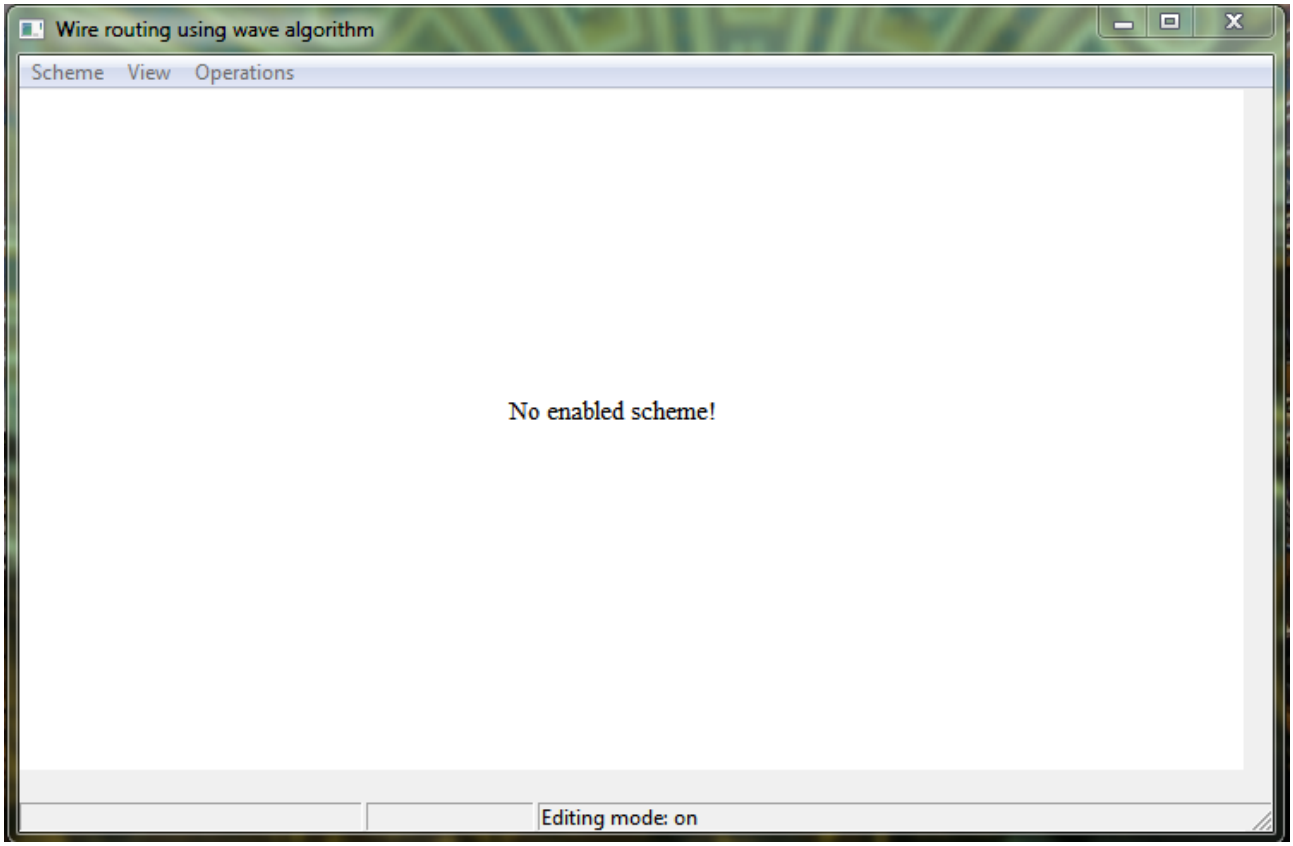


Рис. 2.1 Головне вікно програми

Після цього користувач може виконати завантаження файлу плати з попередньо збереженого файлу, або виконати його створення. Для першого випадку варто скористатись пунктом меню “Scheme – Open”, а для другого – пунктом меню “Scheme – New”.

Після натиснення на кнопку створення нової плати, з’являється діалогове вікно, зображене на рис. 2.2, в якому необхідно ввести розміри плати та комірки. Варто відмітити, що користувач, за бажанням, може змінити колір фону провідника та контакту (wire cell color), колір фону забороненої комірки (locked cell color) та колір самого провідника (wires color), але це не є обов’язковою процедурою.

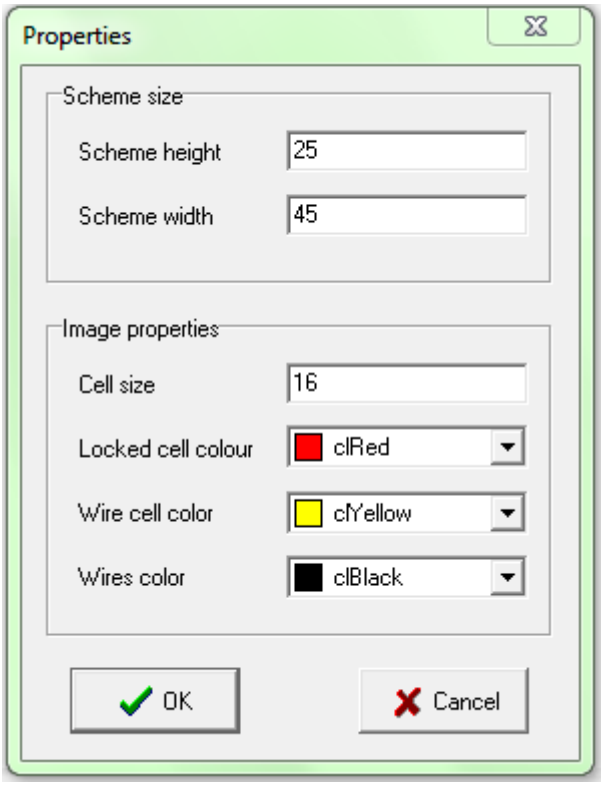


Рис. 2.2 Діалогове вікно створення плати

Після натиснення “OK”, головне вікно набуде вигляду, показаного на рис. 2.3. Зверніть увагу на панель стану: вона відображає стан поточної комірки (empty, locked, contact, wire), її координати та режим програми.

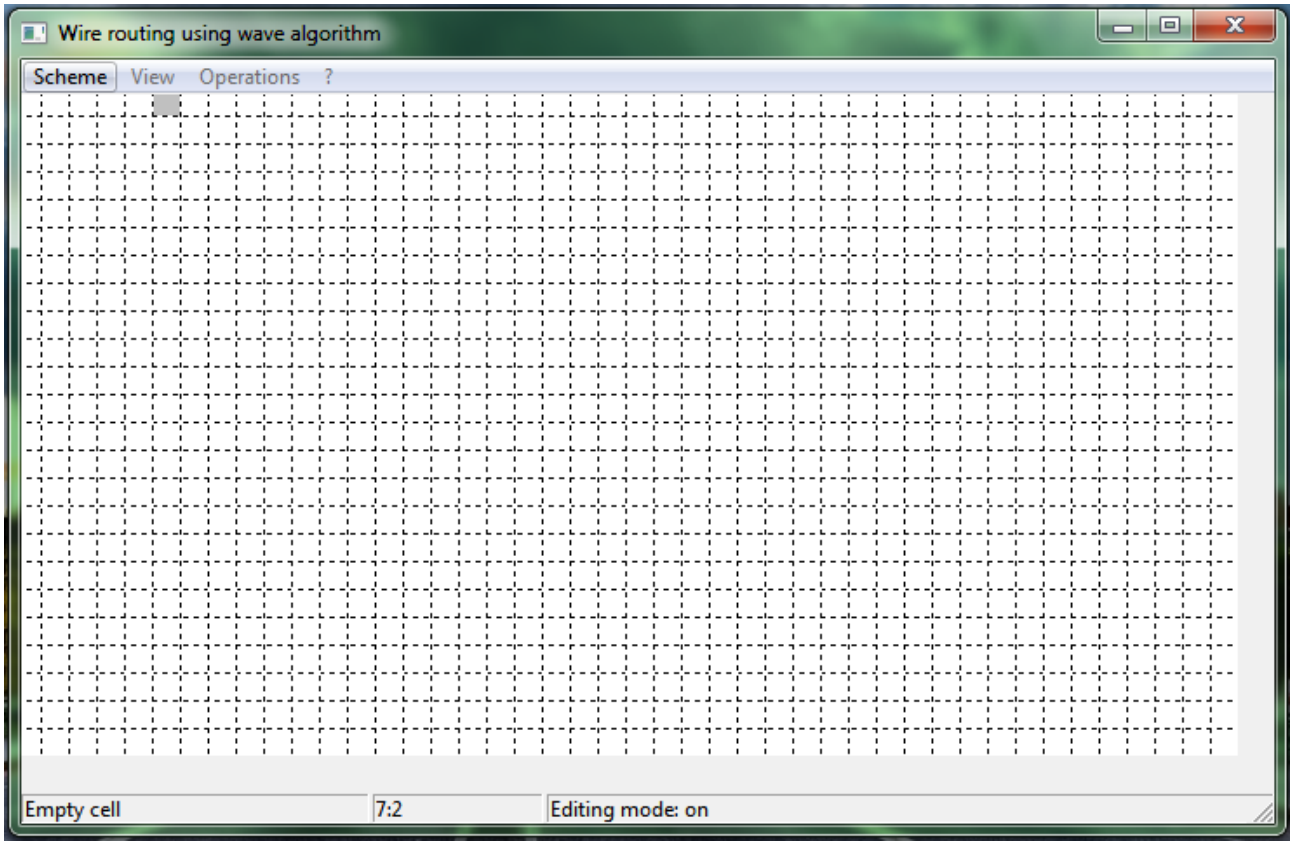


Рис. 2.3 Головне вікно програми з новою платою

Після цього можна безпосередньо перейти до створення плати. Клік правою кнопкою миші над робочою областю викликає контекстне меню, в якому можна зробити наступні дії: додати на плату першу точку нового контакту, додати на плату другу точку контакту (якщо перша вже була встановлена), заблокувати комірку та видалити об’єкт.

Після виконання набору необхідної структури плати можна виконати визначення порядку трасування провідників. Для цього необхідно скористатись пунктом меню “Operations – Routing order”, що викличе вікно, зображене на рис. 2.4.

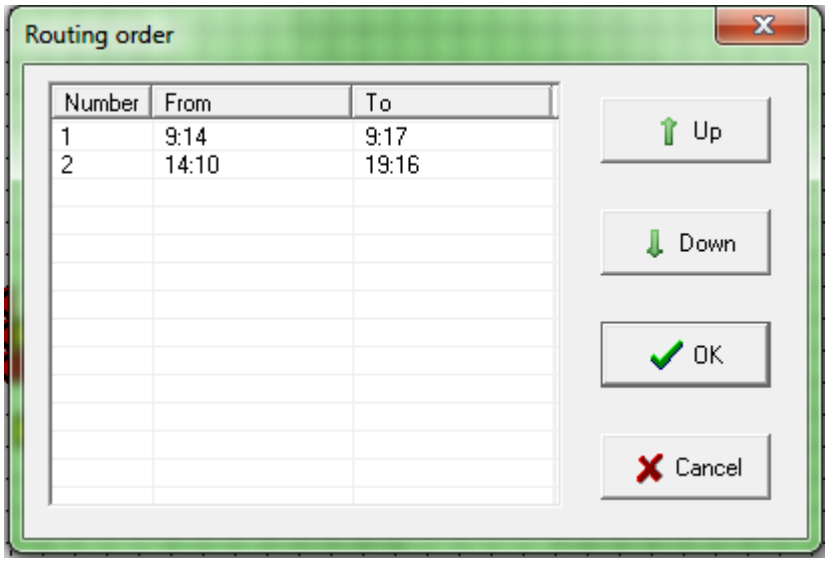


Рис. 2.4 Діалогове вікно зміни порядку трасування

Спочатку всі контакти додаються в список за порядком їх додавання на головне вікно, проте пересуваючи певні з них вище в списку можна підняти їх пріоритет при трасуванні й отримати оптимальні показники трасування плати.

За необхідності, на плату можна додати області, що будуть оминатися при трасуванні, так звані locked cells.

Нарешті плата готова до трасування. Для цього за допомогою пункту меню “Operations – Routing” необхідно його розпочати. Під час процесу трасування кожен крок поширення хвилі буде підсвічуватись на платі сірим кольором та позначатись відповідним інформаційним повідомленням. Після виконання всього обсягу роботи, буде виведено повідомлення з сумарною довжиною провідників.

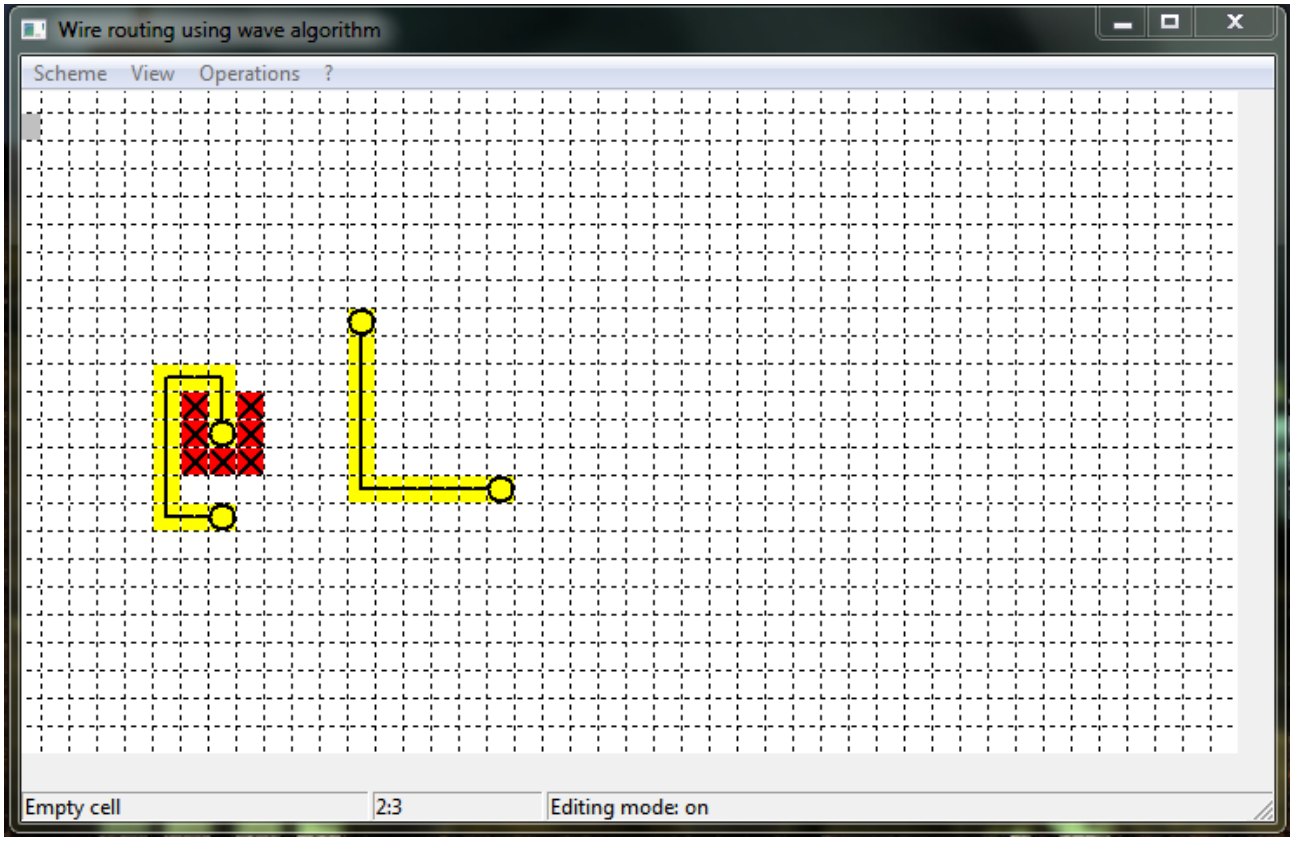


Рис. 2.5 Результат трасування

На рис. 2.5 зображено результат трасування. Якщо він з якихось причин не задовольняє користувача, то за допомогою пункту меню “Operations – Delete all wires”, можна видалити всі провідники й розпочати трасування знову.

Пункт меню “View” призначений для налаштування відображення плати. Тут можна вимкнути режим редагування, що вельми корисно при перегляді результатів роботи, коли внесення випадкових змін в плату вже не є необхідним та переглянути діалогове вікно створення плати (рис. 2.2). Це дає можливість змінити розмір комірки та кольори на платі.

В будь-який момент часу користувач може зберегти плату за допомогою пункту меню “Scheme – Save” або під певним ім’ям, використовуючи пункт меню “Scheme – Save as”. Варто зазначити, що поки збереження ще не відбулось, пункт меню “Scheme – Save” неактивний.

Користувач може також переглянути інформацію про автора в пункті меню “? – About” та зазирнути в короткий файл довідки за допомогою пункту меню “? – Help”. Перед виходом, додаток запитує підтвердження на виконання операції, що може допомогти запобігти втраті даних.

Архітектура додатку надає роботі значної гнучкості, адже користувач може повернутися на попередній крок, змінити плату та провести трасування знову.

Зручний та інтуїтивно зрозумілий інтерфейс користувача знижує поріг входження для початку роботи з додатком, незважаючи на всю складність предметної області та робить саму роботу максимально простою.

Додаток Г
Приклад виконання програми

Додаток Г. Приклад виконання програми

Нехай було задано плату з наступними параметрами, зображеними на рис. 3.1.

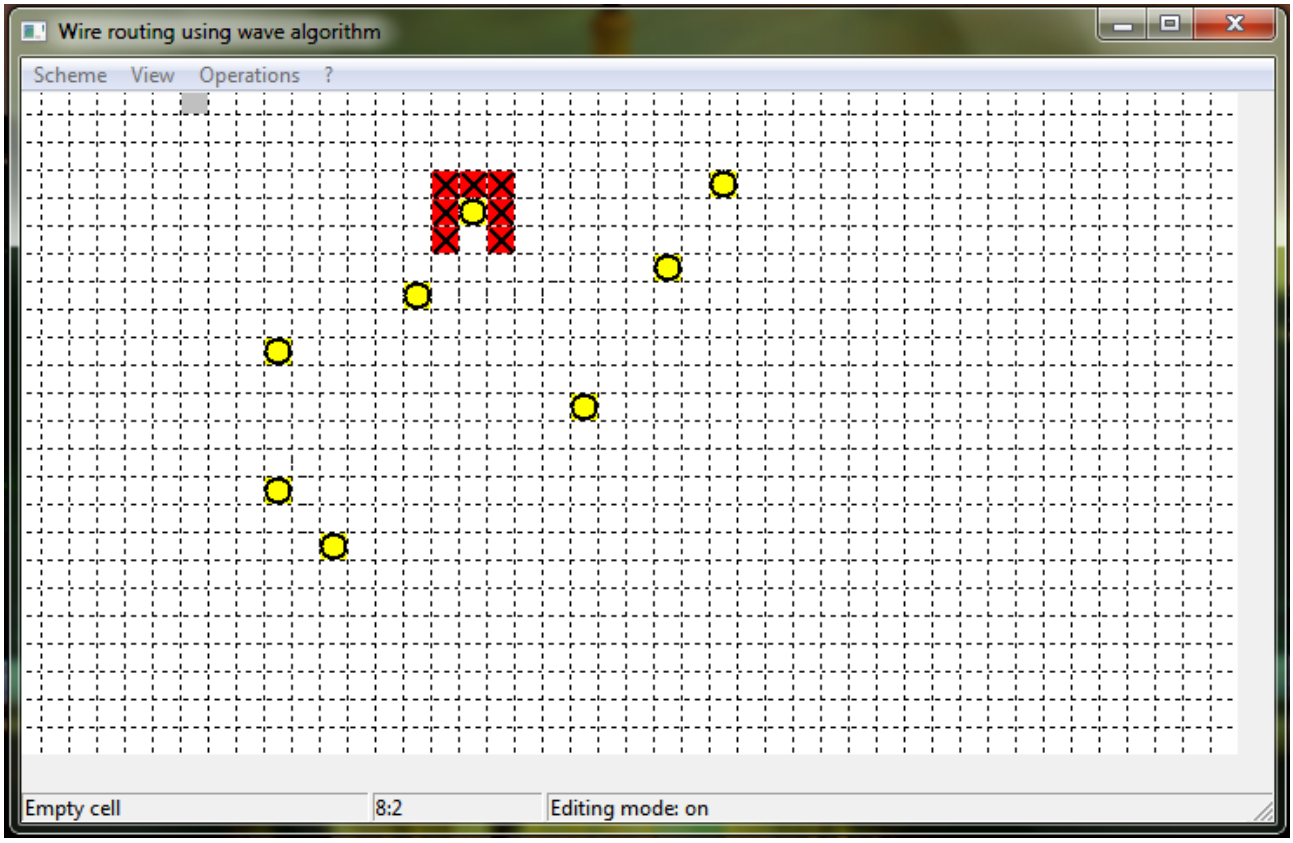


Рис. 3.1 Параметри плати

На цей момент порядок трасування виглядає так, як зображено на рис. 3.2, оскільки в такому порядку контакти додавалися на плату.

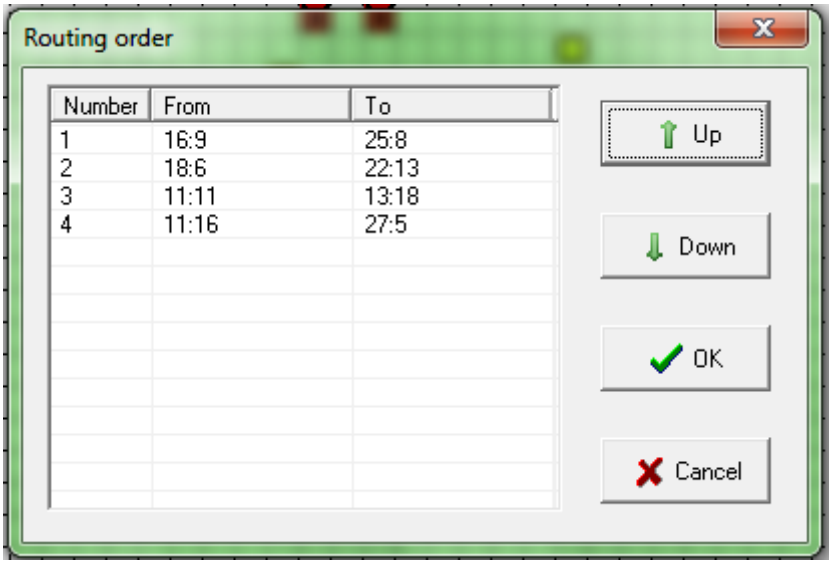


Рис. 3.2 Перший варіант порядку трасування

Якщо почати виконувати трасування в даному випадку, то воно буде виконане не повністю, з сумарною довжиною провідників 48. Зверніть увагу на рис. 3.3. Контакт, оточений заблокованими комірками, так і не зміг бути трасований.

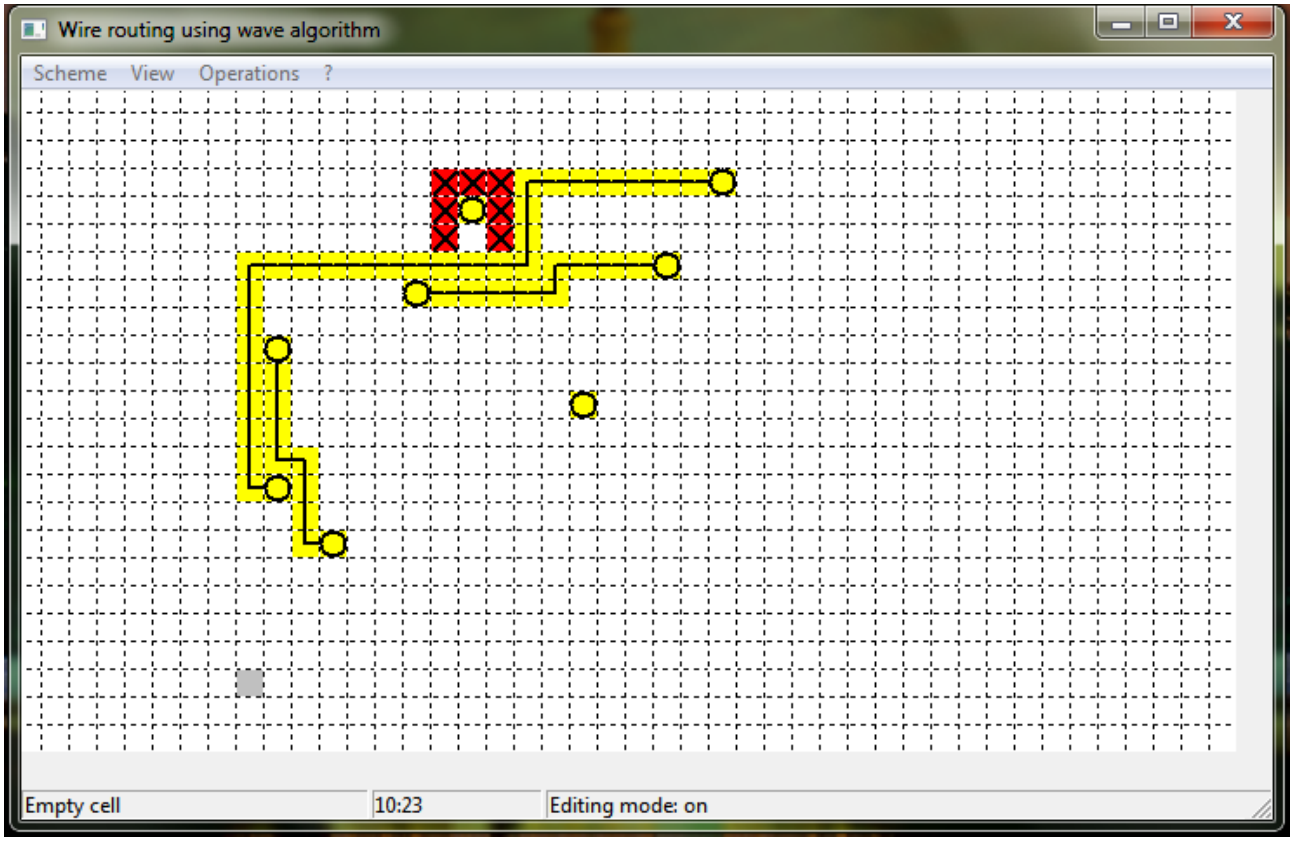


Рис. 3.3 Перший варіант трасованої плати

Для того, щоб “захопити” й цей контакт, необхідно трохи змінити порядок виконання трасування. Зміни зображено на рис. 3.4.

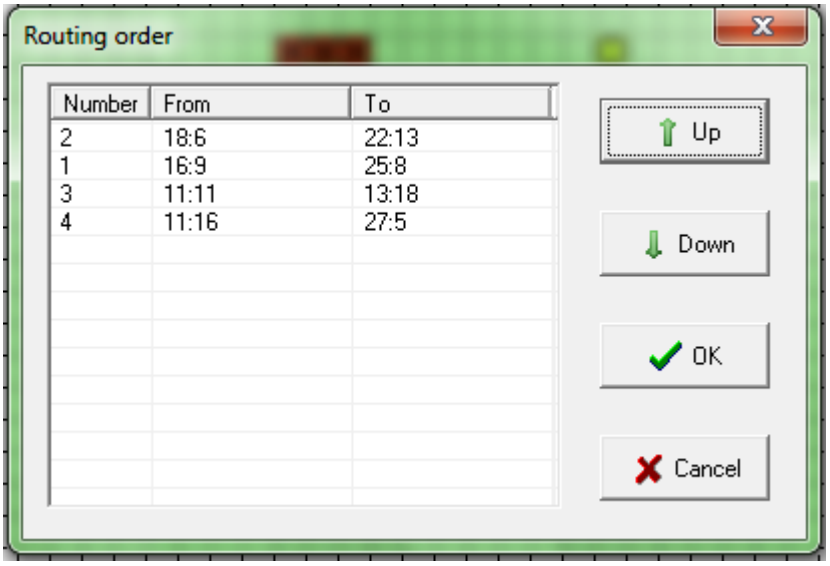


Рис. 3.4 Другий варіант порядку трасування

На рис. 3.5 зображено остаточний результат трасування. Всі контакти поєднано, сумарна довжина провідників – 71.

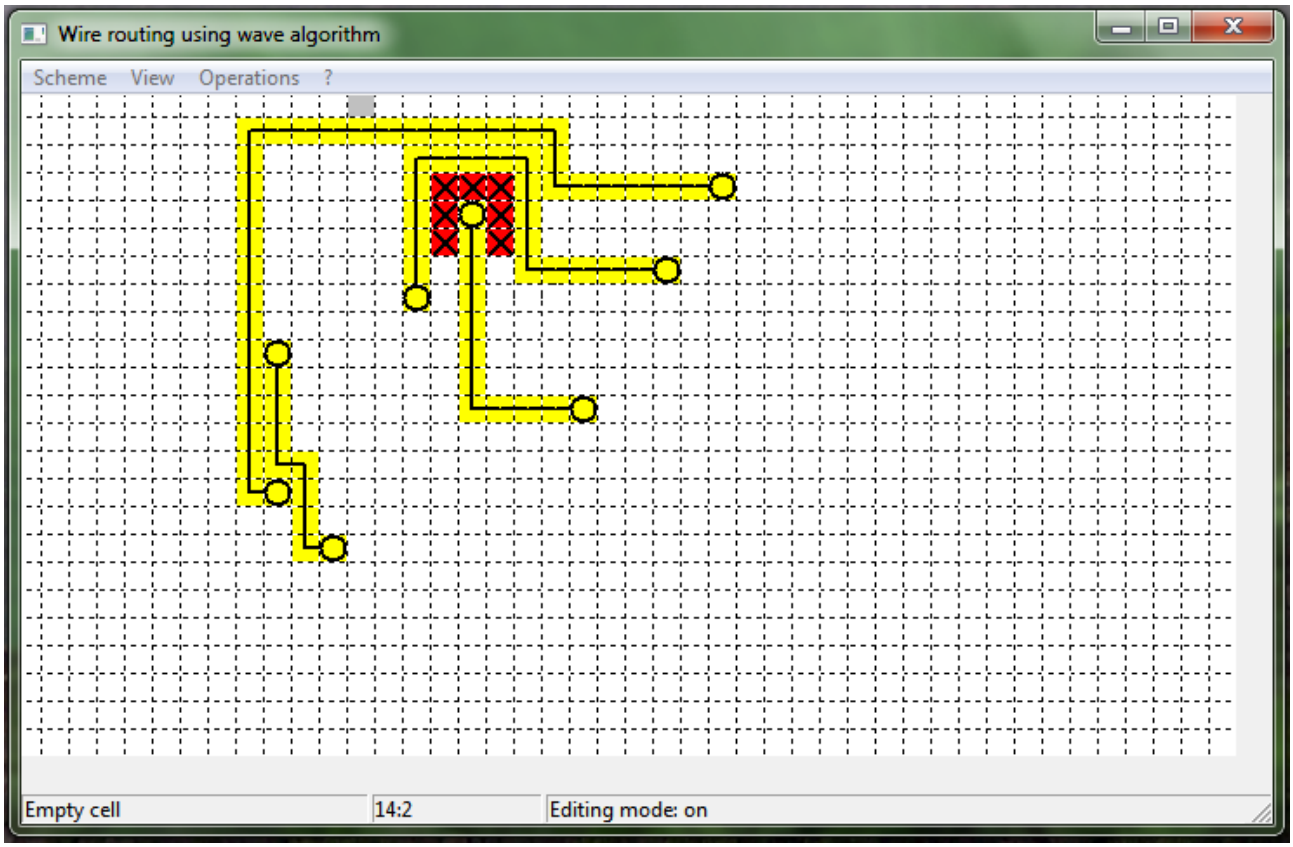


Рис. 3.5 Другий варіант трасованої плати