**3. ПРОЕКТНИЙ РОЗДІЛ**

**3.1 Опис програми**

 Розробляється програма являє собою реалізацію методу згорткового кодування. Основні вимоги, які пред'являються до програми, зводяться до наступних:

1) Наочність і зрозумілість інтерфейсу, так як ПЗ розробляється орієнтовано на звичайного користувача ЕОМ;

2) Надійність, тобто повнота, точність, достовірність та своєчасність одержуваної результуючої інформації.

Т.ч. основний упор при розробці ПО був зроблений на проектування наочного інтерфейсу, а, отже, всі основні функції програми прив'язані до елементів форми.

Як середовище розробки було вибрано випущене компанією Borland засіб швидкої розробки додатків, що дозволяє створювати додатки на мові object pascal - Delphi 7. Обґрунтування такого вибору приведено в подальшому пункті.

В ході створення даного дипломного проекту був написаний кодер і декодер. Опис алгоритмів їх функціонування наведено нижче.

**3.2 Опис блок схем алгоритмів програми**

 На рисунку 3.1 представлені етапи перетворення даних в розробленій програмі.



Рисунок 3.1 - Етапи перетворення вихідних даних в програмі

 Нижче представлені блок схеми алгоритмів кожного етапу.



Рисунок 3.2 - Блок схема для алгоритму перетворення текстового повідомлення в двійковий код

 Як видно з рисунка 3.2 вихідне текстове повідомлення перетвориться в набір біт посимвольно. Кожен символ має своє двійкове подання (ASCII код (Див. Додаток Б)). На черговій ітерації застосовуючи операцію побітового маскування (1 shl j> 0) алгоритм визначає чи є поточний біт символом 1 або 0 і конкатенуються результат в підсумкову рядок str. Стандартний набір символів ASCII складається з 128 десяткових чисел в межах від 0 до 127, призначених буквах, цифрам, знакам пунктуації та найуживанішим спеціальним символам. Розширений набір символів ASCII додатково містить 128 десяткових чисел в межах від 128 до 255, що представляють додаткові спеціальні, математичні, графічні та іноземні символи. На рисунку 3.3 приведена блок схема для алгоритму функціонування згорткового кодера.

Рисунок 3.3 - Блок схема для алгоритму функціонування згорткового кодера

На черговому етапі перетворення повідомлення викликається функція coding класу MealyAutomaton, яка здійснює перетворення згідно з формулою наведеною на рисунку 3.3

На рисунку 3.4 приведена блок схема алгоритму імітації передачі через канал зв'язку.



Рисунок 3.4 - Блок схема для алгоритму імітації передачі через канал зв'язку

Як видно з наведеного вище алгоритму максимальна кількість генеруються помилок обчислюється за формулою:

*Max = Nповідомл* *\* Pпом* */ 100* (3.1)

 де *Мax* - максимальна кількість помилок

*Nповідомл* - довжина повідомлення

*Рпом* - Ймовірність помилки (%)

Тобто якщо довжина повідомлення дорівнює 100 біт, а ймовірність помилки - 10% то генератор згенерує від 0 до 10 помилок. Схема генерації помилок така: функція на черговий ітерації генерує випадкову величину X, використовуючи стандартну функцію rand наступним чином: (99) + 1

потім дане якщо дане значення менше вірогідності помилки: (random (99) + 1 <= errorProbability)

то відбувається генерації помилкових біт (нулі замінюються на одиниці і навпаки) в залежності від параметра «довжина пачки помилок» починаючи з позиції i (змінна циклу обробки). Потім відбувається інкремент змінної циклу i і все повторюється спочатку до тих пір, поки не закінчиться повідомлення або число згенерованих помилок не перевищить число допустимих помилок.

На малюнках нижче приведена статистика по тестуванню генератора помилок при довжині тестового повідомлення 100 біт.

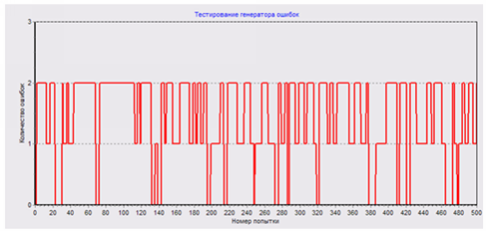


Рисунок 3.5 - Тестування генератора помилок:

Довжина тестового повідомлення - 100 біт; Ймовірність помилок: 0,02; Довжина пачки помилок: 1

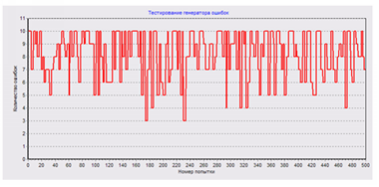


Рисунок 3.6 - Тестування генератора помилок:

Довжина тестового повідомлення - 100 біт; Ймовірність помилок: 0,1; Довжина пачки помилок: 1

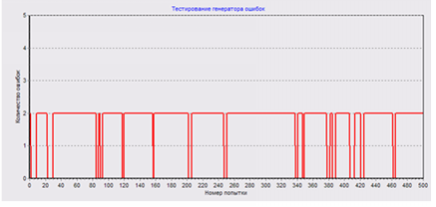


Рисунок 3.7 - Тестування генератора помилок:

Довжина тестового повідомлення - 100 біт; Ймовірність помилок: 0,02; Довжина пачки помилок: 2

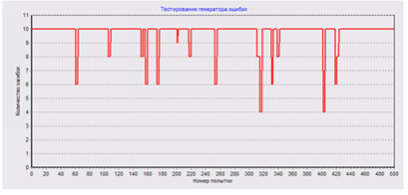


Рисунок 3.8 - Тестування генератора помилок:

Довжина тестового повідомлення - 100 біт; Ймовірність помилок: 0,1; Довжина пачки помилок: 2

 На рисунку 3.9 приведена блок схема алгоритму функціонування декодера.

На кожній ітерації декодер (TTreeHandler) викликає процедуру createLevel яка виходячи з вхідного коду (2 декодованих біта) будує черговий рівень мережі (який може мати 1, 2 або 4 вузла) і розраховує метрики шляхів. Згідно структурі TListElemPtr всі рівні мережі пов'язані покажчиками pnext і pprev. Після повної побудови мережі декодер вибирає вузол на останньому рівні з мінімальною метрикою шляху, а потім, повертаючись за вказівниками pprev, будує декодований шлях (процедура decode). Після чого відображає побудовану мережу (функції printTree);



Рисунок 3.9 - Блок схема для алгоритму функціонування декодера



Рисунок 3.10 - Блок схема для алгоритму перетворення з двійкового коду в текстове повідомлення

Як видно з рисунка 3.10 вихідний набір біт перетвориться в текстове повідомлення по блоках (по 8 біт - 1сімвол в ASCII поданні).На черговій ітерації перетворення мінлива sym (це майбутній текстовий символ) зсувається на 1 біт вправо (sym shr j), після чого до неї додається черговий біт X. Після того як в змінну sym попало 8 біт, результуюча рядок конкатенуються зі змінною sym.

На малюнках нижче приведена статистика по тестуванню декодера при довжині тестового повідомлення 100 біт.

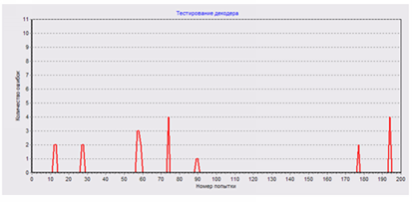


Рисунок 3.11 - Тестування згорткового декодера:

Довжина тестового повідомлення - 100 біт; Ймовірність помилок: 0,05; Довжина пачки помилок: 1

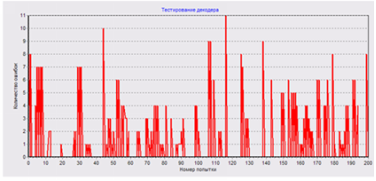


Рисунок 3.12 - Тестування згорткового декодера:

Довжина тестового повідомлення - 100 біт; Ймовірність помилок: 0,1; Довжина пачки помилок: 1

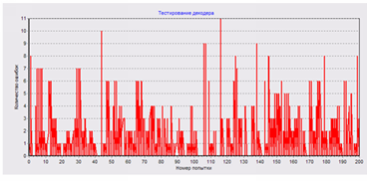


Рисунок 3.13 - Тестування згорткового декодера:

Довжина тестового повідомлення - 100 біт; Ймовірність помилок: 0,05; Довжина пачки помилок: 2

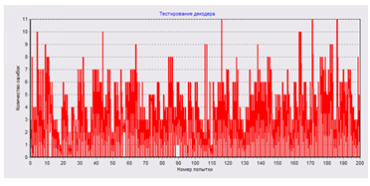


Рисунок 3.14 - Тестування згорткового декодера:

Довжина тестового повідомлення - 100 біт; Ймовірність помилок: 0,1; Довжина пачки помилок: 2.

**3.3 Обґрунтування вибору мови програмування**

 Сучасне візуальне програмування дозволило звести проектування користувацького інтерфейсу до простих і наочним процедурам, які дають можливість за хвилини або години зробити те, на що раніше йшли місяці роботи. Тобто програмування зводиться, фактично, до розміщення компонентів на формі, завданням деяких їхніх властивостей і написання при необхідності обробників подій.

Даний програмний продукт був написаний в середовищі розробки Borland Delphi на мові Object Pascal. Borland Delphi 7.0 - це сучасний програмний продукт, що дозволяє створювати широкий спектр додатків для середовища Microsoft Windows. Обґрунтованість даного вибору полягає в наступному:

* Дозволяє швидко створювати (навіть початківцям програмістам) віконний інтерфейс, що має професійний вигляд, для будь-яких додатків, написаних на будь-якій мові;
* Об'єктна орієнтованість. Основним призначенням застосування в Delphi моделі компонентів є забезпечення можливості багаторазового використання компонентів і створення нових.
* Динамічна ідентифікація типу даних або інтроспекція (можливість визначити тип і структуру об'єкту під час виконання програми).
* Підтримка візуального проектування
* Підтримка методології подієвого програмування

Виходячи з усього вищесказаного, обрана середу розробки і мова програмування повністю задовольняють сучасним вимогам і є зручним засобом для програмування під ОС Windows.