Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Защита информации и надёжность информационных систем

Студент: Лопатнюк П.В.

ФИТ 3 курс 1 группа

Преподаватель: Нистюк О.А.

Минск 2025

# Лабораторная работа № 7

# ПЕРЕМЕЖЕНИЕ/ДЕПЕРЕМЕЖЕНИЕ ДАННЫХ В ИНФОРМАЦИОННО-ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМАХ

**Цель**: приобретение практических навыков использования методов перемежения/деперемежения двоичных данных в информационных системах.

**Задачи**:

1. Закрепить теоретические знания по алгебраическому описанию и использованию методов перемежения/деперемежения двоичных данных в информационных системах.
2. Разработать приложение для реализации метода перемежения/деперемежения символов в сообщениях на основе двоичного алфавита.
3. Результаты выполнения лабораторной работы оформить в виде описания разработанного приложения, методики выполнения экспериментов с использованием приложения и результатов эксперимента.

# Теоретические сведения

Идея перемежения/деперемежения состоит в следующем. Если биты каждого кодового слова Хn передаются не в обычной последовательности, а через интервалы, превышающие ожидаемую длину пакета ошибок (в промежутки между битами одного слова вставляются биты других кодовых слов), то при возникновении такого типа ошибки обратная перемежению операция – деперемежение – разнесет («размажет») группу ошибок по всей совокупности кодовых слов, составляющих данное сообщение.

Длина пакета в нашем случае – это число рядом расположенных ошибочных битов.

Предложено много алгоритмов перемежения/деперемежения. Наиболее простыми являются блочные. При блочном перемежении входные биты делятся на блоки, которые последовательно записываются в строки некоторой таблицы.

Передаваемая последовательность (1010110011…) делится на блоки по 5 битов. Каждый блок записывается в отдельную строку таблицы по порядку. Сообщение для передачи или хранения формируется при считывании символов из таблицы по столбцам: 11010000001… . Деперемежение производится в обратной последовательности. Для данного примера глубина перемежения (разница между позициями одного и того же символа до и после перемежения) равна 4: например, второй символ после перемежения станет шестым. Особенностью является неизменная позиция первого символа.

В общем случае выбор глубины перемежения зависит от двух факторов. С одной стороны, чем больше расстояние между соседними символами, тем большей длины пакет ошибок может быть исправлен. С другой стороны, чем больше глубина перемежения, тем сложнее аппаратно-программная реализация оборудования и больше задержка сигнала.

Для борьбы с длинными пакетами ошибок желательно увеличивать размеры таблицы. Однако это приводит к увеличению задержки в отправке и декодировании сообщения.

# Практическое задание

1. Необходимо разработать авторское приложение в соответствии с целью лабораторной работы. По умолчанию используется блочный перемежитель/деперемежитель. По желанию студент может использовать иной. Задание выполняется по указанию преподавателя в соответствии с вариантом из таблицы

Таблица 1.1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Вариант | Используемый корректирующий код | Длина пакета ошибок | Число столбцов в матрице | Длина сообщения, байт | Длина информационного слова k, бит |
| 8 | Циклический | 3,6,7 | 7 | 13 | 7 |

Рассмотрим следующий пример:

Сообщение: 8 бит = 2 блока.

Блок 1 (Xk1): 1010

Блок 2 (Xk2): 1101

Для циклического кодирования возьмём порождающий полином G=x3+x+1, так как k=4, r=3 и n=7.

Закодириуем первый блок:

x6+x4/x3+x+1=x3+1 (остаток R=x+1~011)

Отсюда закодированный первый блок равен 1010011.

Второй блок:

X6+x5+x3/x3+x+1=x3+x2+1 (остаток R=x2+x=110)

Значит закодированный второй блок равен 1101001.

Перемежение меняет порядок следования бит, чтобы соседние биты в исходной последовательности оказались в разных местах в перемеженной. Это делается для того, чтобы пакетная ошибка в канале связи после деперемежения превратилась в несколько одиночных или малочисленных ошибок, распределенных по разным кодовым блокам.

Матрица 4x4. Записываем биты Xn в матрицу по строкам, слева направо, сверху вниз. Добавляем нули (паддинг) в конце, чтобы заполнить матрицу:

[ 1 0 1 0 ]

[ 0 1 1 1 ]

[ 1 0 1 0 ]

[ 0 1 0 0 ]

Xn\_perem (чтение по столбцам): 1010 0101 1110 0100 (16 битов).

Моделируем повреждение данных при передаче по каналу. Мы предполагаем, что ошибка происходит после перемежения (т.е., в канале). Принимаем Yn\_perem с ошибками в 2 и 3 битах:  1100 0101 1110 0100(16 битов)

На приемной стороне выполняется обратная операция перемежению, чтобы восстановить исходный порядок следования кодовых блоков перед декодированием. Записываем Yn\_perem в матрицу 6x4 по столбцам (сверху вниз, слева направо):

[ 1 0 1 0 ]

[ 1 1 1 1 ]

[ 0 0 1 0 ]

[ 0 1 0 0 ]

Читаем Yn по строкам (16 битов): Yn = 1010 111|1 0010 01|00.

Как видим, ошибки разбросаны по разным местам полученного сообщения.

Бит 6 относится к блоку 1 (Xn1) и является проверочным битом.

Бит 21 относится к блоку 2 (Xn2) и является информационны битом.

Циклический декодер пытается исправить ошибки в принятом блоке Yn, используя избыточность (проверочные биты).

Находим ошибку в Xn1:

x6+x4+x3+x2+x+1/x3+x+1=x3.

Остаток R=x=010 соответсвует 6 столбцу в проверочной матрице, что говорит о ошибке в шестом бите.

Находим ошибку в Xn2:

x6+x3+1/x3+x+1=x3+x.

Остаток R=x2+x+1=111 соответсвует 2 столбцу в проверочной матрице, что говорит о ошибке во втором бите.

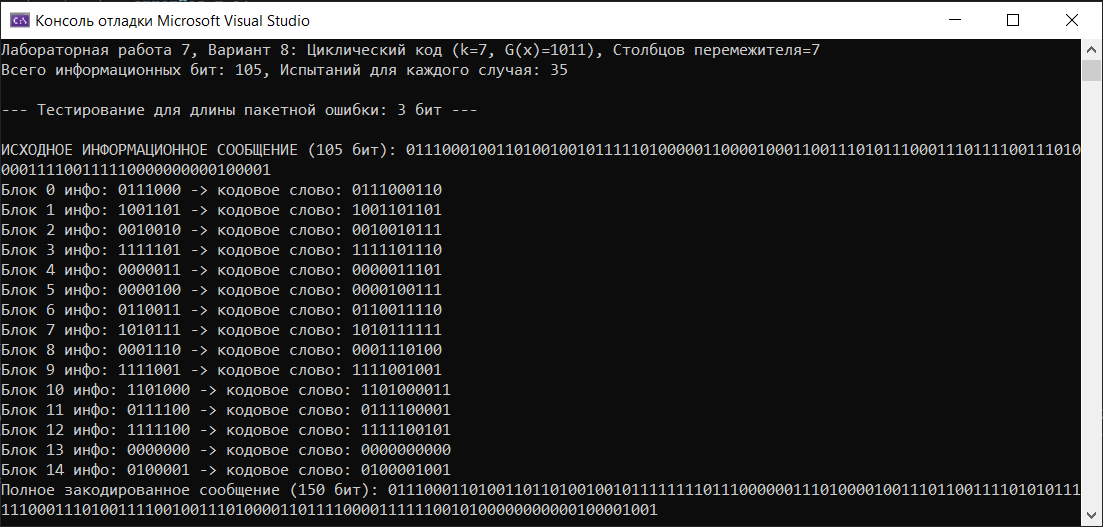


Рисунок 1.1 – Исходные данные

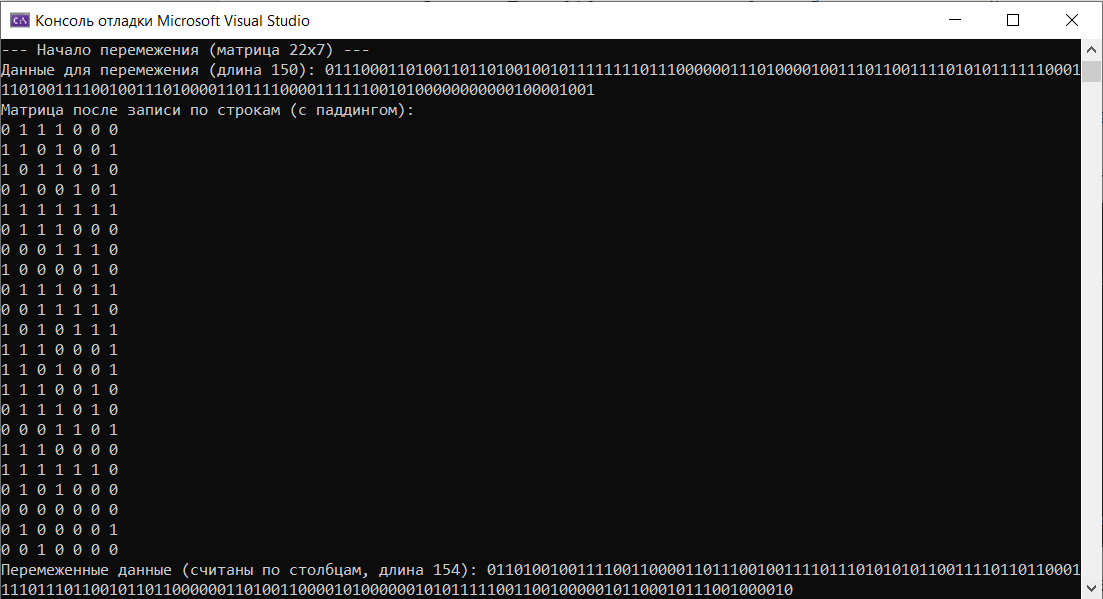


Рисунок 1.2 – Запись матрицы перемежения

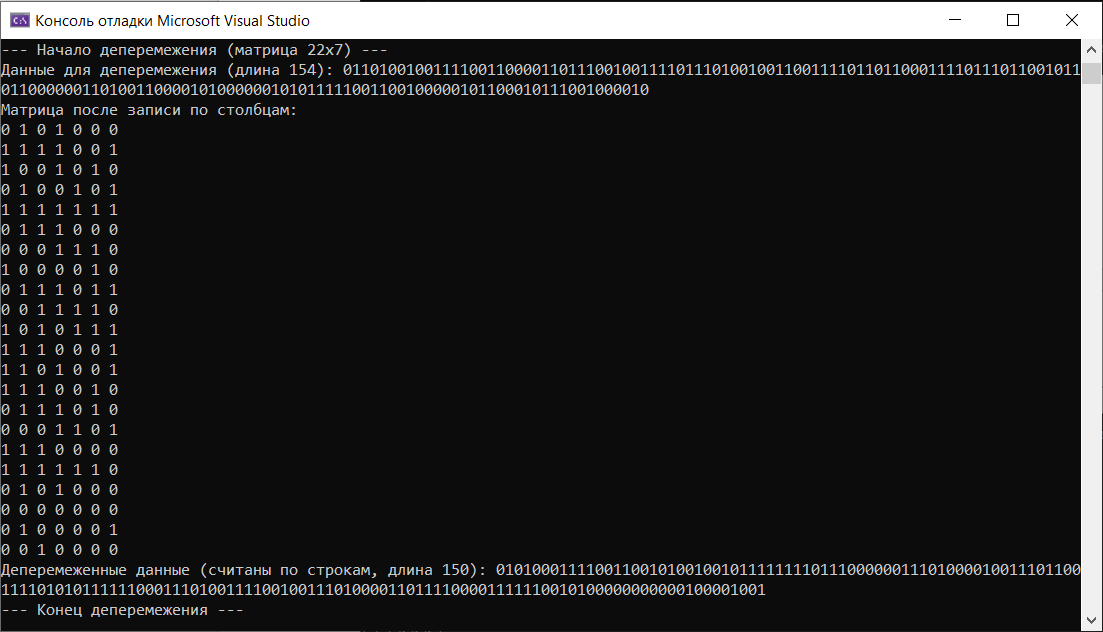


Рисунок 1.3 – Запись матрицы деперемежения

|  |
| --- |
| public class Interleaver  {  private int rows;  private int cols;  private int originalDataLength; // Длина данных до паддинга  public Interleaver(int dataLength, int numCols)  {  if (numCols <= 0) throw new ArgumentOutOfRangeException(nameof(numCols), "Количество столбцов должно быть положительным.");  this.originalDataLength = dataLength;  this.cols = numCols;  this.rows = (int)Math.Ceiling((double)dataLength / numCols);  if (this.rows == 0 && dataLength > 0) this.rows = 1; // Если данных меньше чем столбцов, нужна 1 строка  else if (dataLength == 0) this.rows = 0;  }  public int PaddedLength => rows \* cols;  public int[] Interleave(int[] data) // Убрал verbose, т.к. для этого есть SimulateInterleave  {  if (data.Length != originalDataLength)  throw new ArgumentException($"Входная длина данных {data.Length} не соответствует ожидаемой {originalDataLength}.");  int[,] matrix = new int[rows, cols];  int k\_idx = 0;  int paddedLength = rows \* cols;  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < cols; j++)  {  if (k\_idx < data.Length)  matrix[i, j] = data[k\_idx++];  else  matrix[i, j] = 0;  }  }  int[] interleavedData = new int[paddedLength];  k\_idx = 0;  for (int j = 0; j < cols; j++)  {  for (int i = 0; i < rows; i++)  {  interleavedData[k\_idx++] = matrix[i, j];  }  }  return interleavedData;  }  public int[] Deinterleave(int[] interleavedData) // Убрал verbose  {  if (interleavedData.Length != PaddedLength)  throw new ArgumentException($"Длина перемеженных данных {interleavedData.Length} не соответствует ожидаемой {PaddedLength}.");  int[,] matrix = new int[rows, cols];  int k\_idx = 0;  for (int j = 0; j < cols; j++)  {  for (int i = 0; i < rows; i++)  {  if (k\_idx < interleavedData.Length)  matrix[i, j] = interleavedData[k\_idx++];  else // Защита, хотя не должна срабатывать при правильной PaddedLength  matrix[i, j] = 0;  }  }  int[] deinterleavedData = new int[originalDataLength];  k\_idx = 0;  for (int i = 0; i < rows; i++)  {  for (int j = 0; j < cols; j++)  {  if (k\_idx < originalDataLength)  deinterleavedData[k\_idx++] = matrix[i, j];  else  break;  }  if (k\_idx >= originalDataLength) break;  }  return deinterleavedData;  }  } |

Листинг 1.1 – Класс для перемежения и деперемежения

1. Местоположение заданной группы ошибок выбирается (генерируется) случайным образом. Необходимо для группы ошибок каждой длины сгенерировать 30−40 случайных ситуаций. После деперемежения и исправления ошибок в сообщении сравнить передаваемую последовательность и полученную после исправления ошибок. Проанализировать эффективность перемежения/деперемежения.

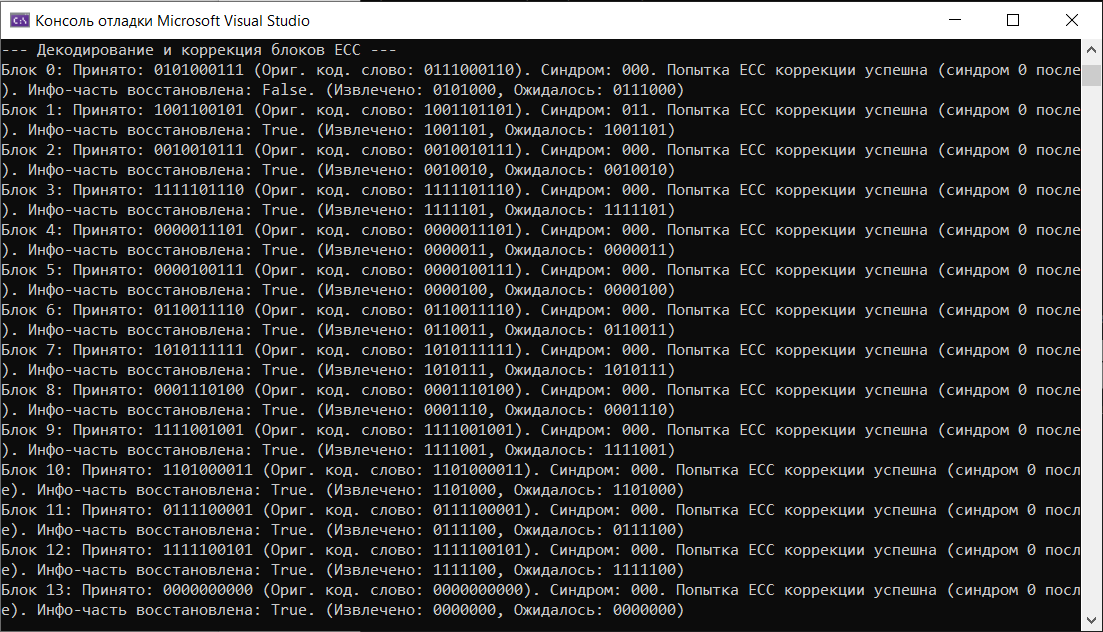


Рисунок 2.1 – Коррекция ошибок

# Вывод

В ходе выполнения данной лабораторной работы было проведено исследование влияния перемежения на способность итеративного корректирующего кода (product code k=6, n=11 с проверками по строкам и столбцам) исправлять пакетные ошибки различной длины (b=3, 5, 6 бит). Моделирование осуществлялось путем внесения пакетных ошибок случайной длины и положения в закодированную последовательность данных как без использования перемежителя, так и с его применением (матричный перемежитель с 6 столбцами). Для каждой длины пакета было проведено 35 случайных испытаний.