МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИССЛЕДОВАНИЕ МЕТОДОВ И УСТРОЙСТВ КОДИРОВАНИЯ ИНФОРМАЦИИ

Методические указания
к выполнению лабораторных работ
по дисциплине «Теория кодирования»
для студентов, обучающихся по направлению 09.03.02
«Информационные системы и технологии»
по учебному плану подготовки бакалавров
дневной и заочной форм обучения

Севастополь
СевГУ
2020

УДК 004.732
И88

Р е ц е н з е н т:
В.Н. Мирянова - канд. техн. наук, доцент,
доцент кафедры ИиУТС

Составитель: В.С. Чернега

И88 Исследование методов и устройств кодирования информации : метод. указания к лабораторным занятиям по дисциплине «Теория кодирования» / Сост. В.С. Чернега ; Министерство науки и высшего образования РФ, Севастопольский государственный университет. – Севастополь : СевГУ, 2020. – 44 с. – Текст : электронный.

Методические указания предназначены для проведения лабораторных работ по дисциплине «Теория кодирования». Целью методических указаний является помощь студентам в изучении методов и алгоритмов эффективного и помехоустойчивого кодирования. Излагаются теоретические и практические сведения необходимые для выполнения лабораторной работы, требования к содержанию отчета.

УДК 004.732

Методические указания рассмотрены и утверждены на методическом семинаре и заседании кафедры «Информационные системы», протокол № 1 от 30 августа 2019 г.

Изд. № 12/2020. Объем 2,75 п.л.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»

СОДЕРЖАНИЕ

Лабораторная работа 1

ИССЛЕДОВАНИЕ ПЕРВИЧНЫХ КОДОВ ОБМЕНА ИНФОРМАЦИЕЙ И ПРОСТЕЙШИХ МЕТОДОВ СЖАТИЯ ДАННЫХ

1 ЦЕЛЬ РАБОТЫ

Углубление теоретических знаний в области оптимального кодирования данных в информационных системах, исследование способов построения таблиц кодирования первичных кодов и простейших методов сжатия символьных последовательностей, приобретение практических навыков исследования процессов кодирования информационных сообщений.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

При начальном кодировании сообщений в процессе подготовки данных на различных носителях ввода, вывода и обработки данных используются так называемые **первичные коды**, которые являются безизбыточными. В настоящее время применяется несколько видов первичных кодов.

Одним из основных первичных кодов является 7-элементный код для обработки информации КОИ-7, созданный на основе международного телеграфного кода МТК-5. Кодовая таблица представляет собой матрицу из 8 столбцов и 16 строк и содержит 128 кодовых комбинаций (КК). Столбцы и строки нумеруются от 0 до 7 и от 0 до 15 соответственно. Комбинация битов кода обозначается $b_7b_6b_5b_4b_3b_2b_1$, где b_1 – младший бит КК. Каждая комбинация битов КОИ-7 имеет однозначное соответствие с позицией кодовой таблицы. Позиции определяются в форме дробного числа x/y, где x – номер столбца, y – номер строки.

Кодовая таблица разделена на области, которые предназначены для набора управляющих и графических символов в следующем виде:

1) столбцы 0 и 1 для представления 32 управляющих символов.
2) позиция 2/0 для символа ПРОБЕЛ, который может интерпретироваться как управляющий и как графический символ.
3) позиция 7/15 (последняя) для представления символа ЗАБОЙ.
4) столбцы с 2 по 7 – для представления набора 94 графических символов.

Основная (базисная) таблица кода приведена в таблице 2.1. С целью обеспечения совместимости национальных и проблемно-ориентированных версий введены кодовые позиции, которым не приписываются конкретные графические символы. Для обнаружения ошибок в кодовой комбинации к каждой КК добавляется восьмой контрольный разряд, значение которого равно сумме по модулю 2 всех 7-ми первых битов (проверка на четность).

Международным стандартом для использования в вычислительной технике и информационных системах является таблица ***ASCII*** (*American Standard Code*

for Information Interchange), построенная на основе международного телеграфного кода МТК-5. Таблица кодов ASCII делится на две части. Международным стандартом является лишь первая половина таблицы, т.е. символы с номерами от **0** (00000000), до **127** (01111111). В таблице кодировки буквы (прописные и строчные) располагаются в алфавитном порядке, а цифры упорядочены по возрастанию значений. Такое соблюдение лексикографического порядка в расположении символов называется принципом последовательного кодирования алфавита.

Для кодирования символов национальных алфавитов используются **расширенные кодировки ASCII.** В настоящее время существуют пять различных кодировок кириллицы: КОИ8-Р (наличие символов псевдографики), Windows (CP1251-Code Page1251, вместо псевдографики дополнительные кириллические символы украинского, белорусского и болгарского языков), MS-DOS (CP866, наличие символов псевдографики, IBM-PC), Macintosh и ISO (ISO 8859-5). Из-за этого часто возникают проблемы с переносом русского текста с одного компьютера на другой, из одной программной системы в другую. Для разрешения проблемы совместимости кодировок был разработан новый международный стандарт ***Unicode***.

Юникод — стандарт кодирования символов, включающий в себя знаки почти всех письменных языков мира, в том числе древних и экзотических. В настоящее время стандарт является доминирующим в информационных системах.

Стандарт состоит из двух основных частей: универсального набора символов (*Universal character set, **UCS***) и форматы кодировок (*Unicode transformation format, **UTF***). Универсальный набор символов ***UCS*** (алфавит кода) перечисляет допустимые по стандарту Юникод символы и присваивает каждому символу код в виде положительного целого числа, записываемого обычно в шестнадцатеричной форме с префиксом `U+`, например, `U+040F`. Форматы кодировок определяет способы преобразования кодов символов для передачи их в потоке или в файле.

Коды в стандарте Юникод разделены на несколько областей. Область с кодами от U+0000 до U+007F содержит символы набора ASCII, и коды этих символов совпадают с их кодами в ASCII. Далее расположены области символов других систем письменности, знаки пунктуации и технические символы. Часть кодов зарезервирована для использования в будущем.

Юникод имеет несколько форм представления символов в компьютере: ***UTF-8, UTF-16*** *(UTF-16BE, UTF-16LE) и* ***UTF-32*** *(UTF-32BE, UTF-32LE).* (*UTF – Unicode transformation format*).

Первой версией, созданной разработчиками консорциума Юникод, была система кодирования **UTF 32**. Цифра в названии кодировки означает количество бит, которое используется для кодирования одного символа (знака). Основным недостатком ее является то, что из-за кодирования символов четырьмя байтами объем хранимой (передаваемой) информации повышался в 4 раза.

В результате развития Юникода появилась **UTF-16**, которая получилась настолько удачной, что была принята по умолчанию как базовое пространство

для всех символов, которые применяются в информационных системах. В этой системе для кодирования одного знака используется два байта.

Но даже эта удачная версия кодировки Юникода не смогла удовлетворить тех, кто писал, например, программы только на английском языке, ибо у них, после перехода от расширенной версии ASCII к UTF-16, объем документов увеличивался в два раза (один байт на один символ в ASCII и два байта на тот же самый символ в ЮТФ-16).

Поэтому консорциумом Unicode была предложена **кодировка переменной длины,** получившая название **UTF-8**. Несмотря на восьмерку в названии, она действительно имеет переменную длину, т.е. каждый символ текста может быть закодирован в последовательность длиной от одного до шести байтов.

На практике же в UTF-8 используется только диапазон от одного до четырех байтов, потому что в алфавитах всего мира, включая самые экзотические, суммарное количество символов не превышает 2^{32}.

В кодировке UTF-8 одним байтом кодируются латинские буквы, цифры и специальные символы. Русские буквы (кириллица) представляются 16-битными (двухбайтными) кодами:

110XXXXX 10XXXXXX,

где **110 –** признак двухбайтного кода UTF-8; **10 –** признак продолжения многобайтного кода; X **–** двоичные разряды для размещения кода символа в соответствии с таблицей *UNICODE*. Другие языки кодируются 3-мя или 4-мя байтами. Так, например, грузинские символы кодируются тремя байтами.

Т.о. консорциум Юникод после создания UTF-16 и 8 решил основную проблему, в результате чего в инфокоммуникационных системах **в шрифтах существует единое кодовое пространство**.

Коды, использующие лишь определенную часть всех возможных комбинаций, называются избыточными. Оставшаяся часть комбинаций применяется для обнаружения или исправления ошибок. В этих кодах часть разрядов k используется для кодирования информационной части сообщения, а другая часть – для коррекции ошибок. В теории информации под избыточностью кода *R* понимают отношение числа проверочных разрядов *r* к общей длине кодовой комбинации *n*:

$$R = r / n = (n-k) / n. \qquad (2.1)$$

Кодирование информации осуществляют для устранения избыточности — *эффективное кодирование* или для введения дополнительной избыточности — *помехоустойчивое кодирование*, либо с целью защиты (закрытия информации для несанкционированных пользователей) — *шифрование*. Процедуру эффективного кодирования называют также сжатием данных.

Для оценки эффективности процедуры сжатия сообщений используется несколько показателей степени компрессии данных. При оценке эффективности сжатия текстовых сообщений наиболее широко используется коэффициент сжатия $K_{СЖ}$, который характеризует объем сообщения $V_{СЖ}$ (в битах или байтах) на выходе компрессора после сжатия по отношению к исходному объему $V_{И}$

$$K_{сж} = V_{сж} / V_{и} . \quad (2.2)$$

Часто коэффициент сжатия выражают в процентах. Для этого значение, вычисленное по (2.2), умножается на 100. Очевидно, что при отсутствии сжатия $K_{сж} = 100$ % и уменьшается с повышением эффективности процедуры компрессии.

Оценка уменьшения объема изображений в процессе их сжатия в основном производится с помощью коэффициента компрессии $K_С$, который определяется по формуле

$$K_с = V_И / V_{сж}. \quad (2.3)$$

Он показывает, во сколько раз уменьшился объем исходного сообщения после компрессии. Не трудно заметить, что $K_{сж}$ и $K_с$ являются обратными величинами, т. е. $K_{сж} = 1 / K_с$.

Некоторые авторы для определения степени сжатия используют коэффициент сжатия данных $K_{СД}$, определяемый соотношением

$$K_{СД} = (1 - V_{сж} / V_И)100 \% , \quad (2.4)$$

который характеризует объем данных, исключенных из сообщения в процессе его сжатия. При отсутствии эффекта сжатия $K_{СД} = 0$ % , а в случае максимального сжатия коэффициент $K_{СД}$ приближается к 100%.

В текстовых файлах часто встречаются относительно длинные последовательности одинаковых символов. В первую очередь это символы пробела, дефиса и некоторых специальных знаков. Избыточность таких сообщений может быть существенно сокращена за счет замены группы одинаковых символов последовательностью, состоящей из трёх байтов. Первый является специальным признаком компрессии Sк, индицирующим начало сжатой строки; второй *Ch* - собственно повторяющийся символ и третий *Cn* - счетчик количества одинаковых символов в сжимаемой последовательности. Этот метод получил название "*Run-Length encoding*" *RLE* - кодирование. Например, при поступлении от источника последовательности символов ABCCCCCDEAAAA7C строка сжатых данных приобретает вид ABSкC5DESкA47C, т. е. вместо 15 она занимает объем 12 байтов. Очевидно, что сжимать исходную последовательность целесообразно при длине одинаковых символов в строке не менее 4-х. Максимальное число одинаковых символов ограничивается разрядностью счетчика *Cn* и обычно не превышает 255.

В качестве признака компрессии можно выбрать любую неиспользуемую комбинацию кода КОИ-7 или КОИ-8. Для случаев, когда используются все наборы заданного кода обработки информации, можно применять двойные символы, которые не могут встречаться в тексте (например ЬЬ). В этом случае минимальное количество повторяющихся символов в блоке, которые целесообразно сжимать, равно пяти.

Способ *RLE* достаточно широко использовался в системах архивации до середины 80-х годов, в частности при сжатии псевдографических изображений.

В настоящее время он также находит применение при сжатии неподвижных изображений (факсимильные сообщения, файлы *PCX*-формата), а также является составной частью ряда комбинированных способов сжатия.

Если блоки данных преимущественно содержат числовую информацию, то сжатие сообщений может быть достигнуто путем уменьшения числа битов на знак с 7 до 4-х, то есть заменой (*упаковкой*) комбинаций кода КОИ-7 *(ASCII)* на четырёхразрядные. Числа в коде КОИ-7 всегда имеют в трёх старших разрядах комбинацию 011 и поэтому нет необходимости передавать эти биты. Кроме цифр в упакованной форме могут быть переданы знаки (+) и (-), а также десятичная точка (.). Это связано с тем, что младшие тетрады этих символов отличаются от младших тетрад десятичных чисел и при их распаковке могут быть легко преобразованы в соответствующий им эквивалент в *ASCII*-коде. Знак пробела, который часто встречается в цифровых последовательностях, целесообразно кодировать четырёхразрядной комбинацией, состоящей из четырёх единиц 1111, что соответствует младшей тетраде символа (/).

Для того чтобы при распаковке (на приёмной стороне) можно было определить начало упакованной последовательности, используются символ признака сжатия данных *Sк* и счетчик количества упакованных чисел *Cn*, которые размещаются непосредственно перед сжатой последовательностью. Пример фрагмента кадра с упакованной последовательностью десятичных чисел со знаком и десятичной точкой показан на рисунке 2.1. Значок апострофа показывает, что число представлено в упакованном виде.

Sk	*Cn*	2 ’	6 ’	. ‘	3 ‘	/ ‘	. .	/ ‘	5 ‘	7 ‘	4 ‘

Рисунок 2.1- Фрагмент кадра с упакованной последовательностью десятичных чисел

Коэффициент сжатия упакованной десятичной последовательности зависит от её длины:

$$K_{сж} = V_{УП} / V_{И} = (n_{Sk} + n_{ch} + 4\,N_{Б}) / 8\,N_{Б}\,, \qquad (2.5)$$

где $V_{УП}$ и $V_{И}$ – объем соответственно упакованной и исходной последовательности в битах; $N_{В}$ – количество байтов исходной числовой последовательности; nSk, n_{cn} – количество битов для кодирования символа признака сжатия данных и счетчика упакованных чисел соответственно.

Если для представления nSk и n_{cn} выбирается по одному байту, то коэффициент сжатия рассчитывается по формуле:

$$K_{сж} = 2 / N_{В} + 0{,}5\,. \qquad (2.6)$$

Отсюда видно, что для того, чтобы Ксж был меньше 1, сжимать следует цифровые последовательности данных не менее пяти байтов. Минимально достижимый коэффициент сжатия равен

$$K_{\text{сж min}} = \lim_{N_B \to \infty} \left(\frac{n_{Sk} + n_{cn}}{8N_B} + 0{,}5 \right) = 0{,}5 \quad . \qquad (2.7)$$

Таким образом, исходная цифровая последовательность может быть сжата до 50% от своего первоначального вида, что эквивалентно повышению эффективной скорости передачи в 2 раза (или соответственно двукратному снижению объема занимаемой памяти).

3 ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

В качестве лабораторной установки используется персональный компьютер с установленной программой файлового менеджера Total Commander, позволяющий просматривать сообщения, выполненные в различных кодировках. Вид окна с анализируемым текстом и типами кодировок показан на рисунке 3.1.

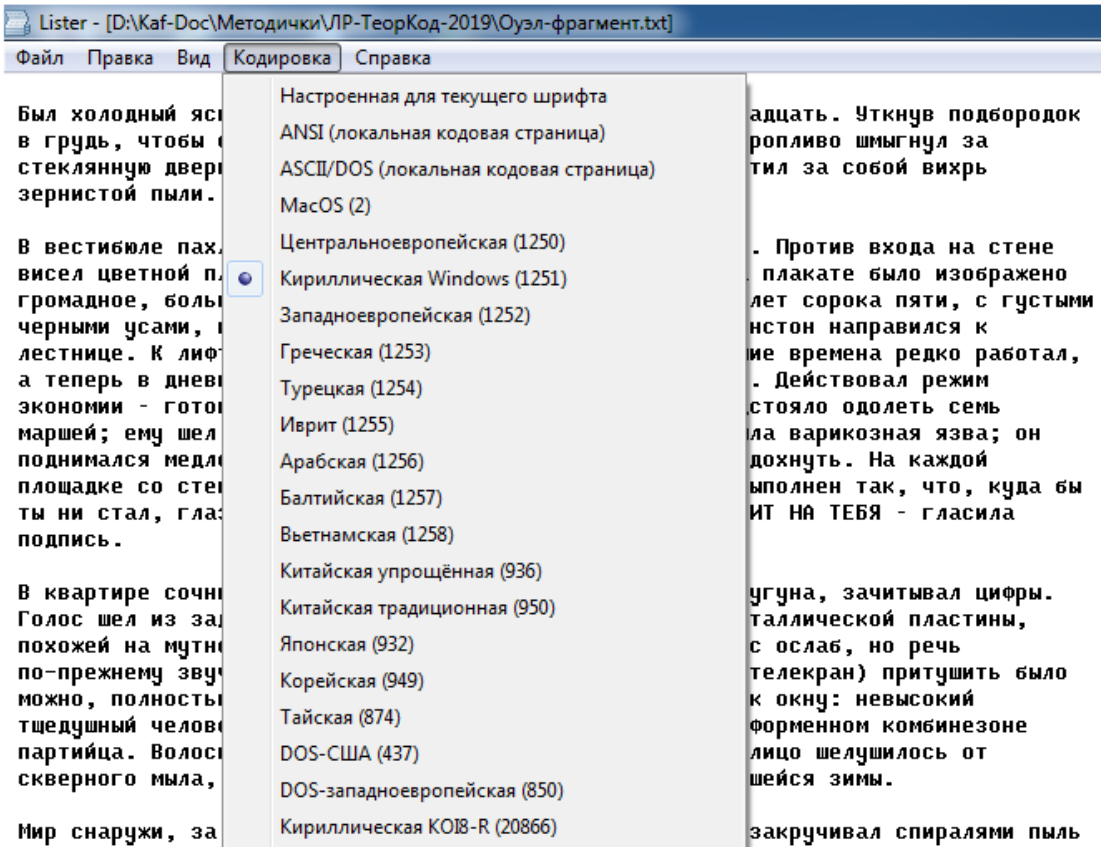


Рисунок 3.1 – Вид окна файлового менеджера с различными кодировками

4 ПРОГРАММА ЛАБОРАТОРНОЙ РАБОТЫ

4.1 Изучить разделы рекомендуемой литературы и конспекта, касающиеся основных понятий теории кодирования, первичных кодов и простейших методов сжатия. (Выполняется в процессе домашней подготовки).

4.2 Набрать в текстовом редакторе (Блокноте) строку произвольного сообщения размером 10-15 символов и сохранить ее в файле.

4.3 Открыть сохраненный файл в Total Commander в режиме просмотра (F3) и найти кодировки, в которых происходит правильное отображение текста.

4.4 Записать закодированную строку в 16-ричном коде.

4.5 Найти символы кодируемой строки в таблице CP-1251, выписать их десятичные коды и представить их в двоичном виде. Сравнить эти коды с представлением символов в 16-ричном коде.

4.6 Посмотреть кодируемую строку при кодировке ASCII/DOS, выписать 16-коды символов и сравнить их с кодами соответствующих символов при использовании кодовой страницы CP-1251.

4.7 Выполнить пункт 4.6 при кодовой странице KOI8-R и пояснить причину неверного отображения закодированной строки.

4.8 Вычислить объем видеофайла, содержащего данные для отображения на экране дисплея с разрешающей способностью 800×600 изображения, в котором на синем фоне в центре экрана располагается красный прямоугольник размером 20×20 пикселей.

4.9 Закодировать содержимое видеофайла методом RLE и определить объем сжатого файла и рассчитать коэффициент компрессии.

4.10 Составить отчет по выполненной работе.

5 СОДЕРЖАНИЕ ОТЧЕТА

5.1 Титульный лист.
5.2 Цель и программа лабораторной работы.
5.3 Исходные данные в соответствии с индивидуальным вариантом.
5.4 Скриншоты с текстом сообщения в разных кодировках и в шестнадцатеричной системе.
5.5 Расчет объема исходного и сжатого видеофайлов и коэффициента компрессии.
5.6 Выводы по результатам исследований.

6 КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1 Какие существуют виды кодирования и с какой целью они используются?
6.2 С какой целью в таблицы кодирования ASCII и КОИ-7 введены управляющие символы и в каких случаях они используются?

6.3 По какой причине была выполнена разработка стандарта кодирования «Юникод», какие существуют форматы этого кода и в чем их различие?
6.4 Каким образом символы русского алфавита отображаются в кодировке UTF-8?
6.5 Что такое избыточность кода и как она определяется количественно?
6.6 Какие существуют виды кодирования и в каких случаях используется тот или иной вид кодирования?
6.7 Какие показатели используются для оценки сжатия сообщений?
6.8 В каких случаях можно осуществлять сжатие сообщений с частичной потерей информации?
6.9 За счет чего осуществляется сжатие при использовании метода *RLE*?
6.10 Каким образом уменьшают объем сообщений, состоящих из последовательности десятичных цифр?

Лабораторная работа 2

ИССЛЕДОВАНИЕ СТАТИСТИЧЕСКИХ МЕТОДОВ СЖАТИЯ ДАННЫХ БЕЗ ПОТЕРЬ ИНФОРМАЦИИ

1 ЦЕЛЬ РАБОТЫ

Углубление теоретических знаний в области оптимального кодирования (компрессии) сообщений в информационных системах и исследование способов построения префиксных неравномерных кодов, приобретение практических навыков исследования процессов кодирования информационных сообщений.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Все методы компрессии сообщений делятся на две большие группы: *методы сжатия без потерь* и *методы сжатия с частичной потерей информации*. В случае сжатия без потерь восстановленное после декомпрессии сообщение полностью соответствует исходному. Эти методы используются в файловых архиваторах, при сжатии текстовых сообщений, в аппаратуре передачи данных. Они также применяются в большинстве систем передачи факсимильных сообщений.

Методы сжатия текстовых сообщений без потерь в свою очередь могут быть разделены на две подгруппы: *статические* и *динамические* методы. В статических методах компрессор на основе анализа статистических свойств сообщения составляет кодовую таблицу и передает ее декомпрессору. В процессе

сжатия сообщения эта таблица не изменяется. При динамических методах компрессия данных осуществляется в темпе поступления символов сообщения и одновременно строится таблица кодирования и декодирования. В процессе сжатия сообщения осуществляется синхронная модификация кодовых таблиц на кодирующей и декодирующей сторонах.

Целью оптимального сжатия (эффективного кодирования) является получение сжатого сообщения минимально возможного объема. Для получения оптимального кода, имеющего минимальную длину кодовой комбинации, необходимо добиваться наименьшей избыточности каждого из кодовых слов, которые в свою очередь должны строиться из равновероятных и взаимонезависимых символов. При этом каждый кодовый элемент должен принимать значения 0 или 1 по возможности с равными вероятностями, а выбор очередного элемента быть независимым от предыдущего.

Среднее количество информации *H(A),* приходящееся на один символ, генерируемый источником независимых дискретных сообщений, определяется как математическое ожидание дискретной случайной величины $I(a_i)$ и характеризует количество информации, содержащееся в одном случайно выбранном символе

$$H(A) = \overline{I(a_i)} = -\sum_{i=1}^{m} P(a_i) \log_2 P(a_i) . \qquad (2.1)$$

Величина *H(A)* называется энтропией источника независимых сообщений. Энтропия *H(A)* характеризует первичный алфавит ***A*** источника и является независимой информационной характеристикой.

Одним из основных параметров, характеризующих неравномерный код, является *средняя длина* кодовых слов во вторичном алфавите $l_{ср}$. Если дискретный источник без памяти имеет алфавит из *m* символов $a_1, a_2, \ldots, a_m$ с вероятностями $P(a_1), P(a_2), \ldots, P(a_m)$, то

$$l_{ср} = \sum_{i=1}^{m_1} P(a_i) l(a_i) , \qquad (2.2)$$

где $P(a_i)$-вероятность появления *i*-го символа алфавита источника; $l(a_i)$ - длина *i* -го кодового слова.

Алгоритм построения оптимального кода впервые был предложен К.Шенноном в 1948 году и несколько позже модифицирован Р.Фано [1,3,5].

Код строится следующим образом. Кодируемые знаки выписывают в таблицу в порядке убывания их вероятностей в сообщениях. Затем их разделяют на две группы так, чтобы значения сумм вероятностей в каждой группе были близкими. Все знаки одной из групп в соответствующем разряде кодируются, например, единицей, тогда знаки второй группы кодируются нулем. Каждую полученную в процессе деления группу подвергают вышеописанной операции о тех пор, пока в результате очередного деления в каждой группе не останется по одному

знаку. Примеры построения кода Шеннона-Фано для различных вариантов распределения вероятностей появления символов в сообщении приведены в [1,5].

Алгоритм Шеннона-Фано не всегда приводит к однозначному построению кода. Это вызвано тем, что при разбиении m символов источника на подгруппы можно сделать большей по вероятности как верхнюю, так и нижнюю подгруппы. При различном разбиении на подгруппы среднее число битов на символ может быть разным. Более эффективным является алгоритм, предложенный Хаффменом в 1952 г., который позволяет построить оптимальный код с наименьшим для данного распределения вероятностей средним числом битов на символ, то есть

$$l_{cp} = min\ [\ \sum_{i=1}^{m} P(a_i) l(a_i)\ ]\ . \qquad (2.3)$$

Для двоичного кода алгоритм Хаффмена сводится к следующему. Символы сообщения упорядочиваются по убыванию вероятностей и располагаются в основной столбец таблицы таким образом, что $P(a_i) \geq P(a_j)$ для всех $i<j$ (табл.2.8).

Два последних символа объединяются в один вспомогательный, вероятность которого равна суммарной вероятности составляющих его символов. Все оставшиеся символы, вместе с образованным вспомогательным символом, снова располагаются по убыванию вероятностей в дополнительном столбце. Два последних элемента столбца объединяются во второй вспомогательный символ, и образуется следующий дополнительный столбец, в котором все элементы расположены в порядке убывания вероятностей. Процедура продолжается до тех пор, пока не получится единственный вспомогательный символ, имеющий вероятность, равную единице. Код, построенный по рассмотренному алгоритму, получил название *кода Хаффмена*.

Примеры построения кода Хаффмена для различных вариантов распределения вероятностей появления символов в сообщении приведены в [1,5].

Для формирования кодовых комбинаций, соответствующих символам данного сообщения, необходимо проследить путь перехода символов по строкам и столбцам таблицы. При построении кодов Хаффмена наиболее часто используются кодовые деревья. С одной стороны это позволяет более наглядно отобразить процедуры кодирования и декодирования, а с другой - облегчить программную реализацию этих процедур.

Построение дерева начинается с корневого узла, вероятность которого равна 1. Из корня проводятся две ветви, причем ветви с большей вероятностью присваивается значение (*бит*) 1, а с меньшей вероятностью — 0. Вновь образованные узлы могут отображать одиночный или вспомогательный символы. В последнем случае узел является промежуточным и из каждого из них, в свою очередь, снова проводятся по две ветви. Такое последовательное ветвление продолжается до тех пор, пока не будет достигнут узел, соответствующий вероятности символа алфавита (*узел листа*). Двигаясь по кодовому дереву от корня сверху

вниз, можно записать для каждого символа соответствующую ему кодовую комбинацию.

Алгоритм Хаффмена является двухпроходным, так как при его реализации требуется дважды просматривать кодируемое сообщение. При первом проходе вычисляются вероятности (*частоты*) появления символов в сжимаемом сообщении и строится хаффменовское дерево.

При втором проходе осуществляется кодирование символов, поступивших от источника. В этом случае определяется значение ветвей дерева при движении от *листа*, соответствующему кодируемому символу, к корню.

3 ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

В качестве лабораторной установки используется персональный компьютер с установленной программой Eff_code_4. Вид главного окна программы изображен на рисунке 3.1. Интерфейс программы лабораторной установки простой и интуитивно понятен.

В главном меню отображены разделы теории эффективного кодирования, при активации одного из них на экран монитора выводится задание, которое необходимо выполнить в процессе лабораторного исследования.

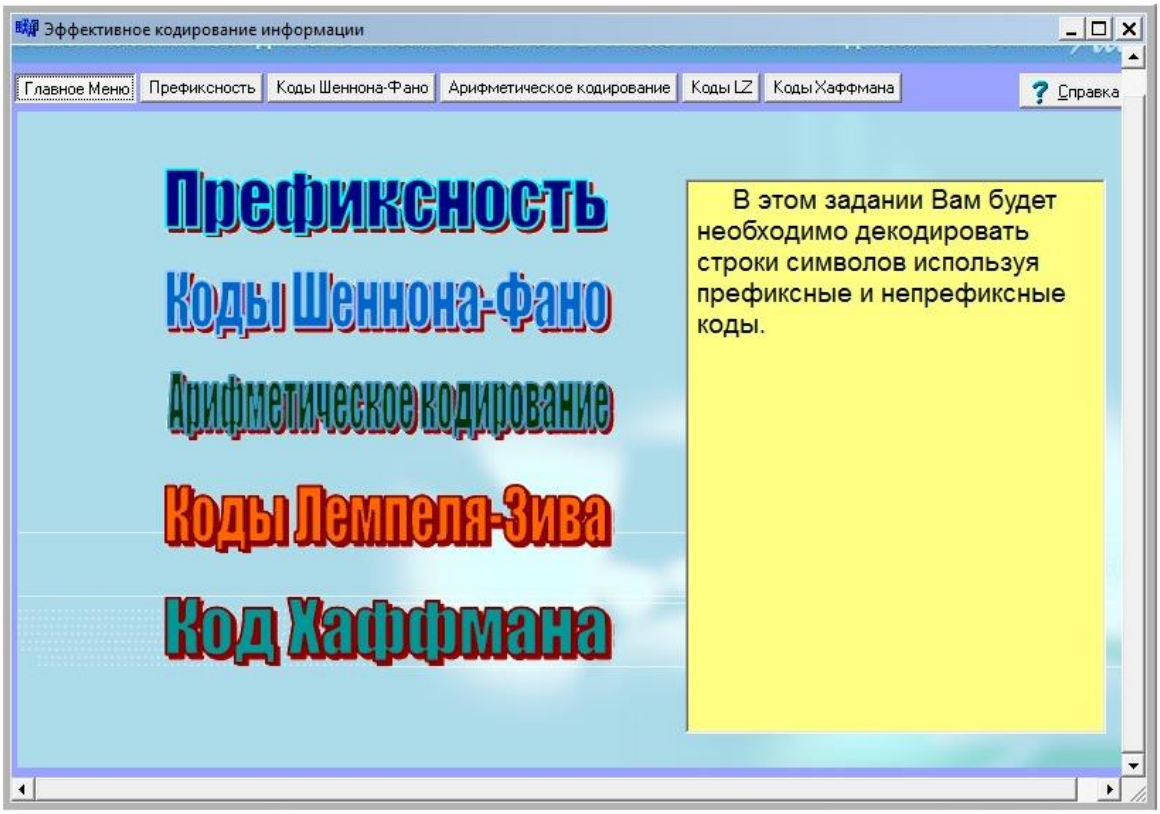


Рисунок 3.1 – Вид главного окна лабораторной установки для исследования способов статического сжатия сообщений

Для исследования способов сжатия сообщений, основанном на одном из кодов эффективного кодирования источников, необходимо перейти на закладку с названием соответствующего кода. При этом на экране появится задание, которое необходимо выполнить в процессе экспериментальных исследований.

4 ПРОГРАММА ЛАБОРАТОРНОЙ РАБОТЫ

4.1 Изучить по рекомендуемой литературе теоретический материал по теме статического кодирования источников информации неравномерными кодами и разобрать примеры построения префиксных кодов. Выполняется в процессе домашней подготовки.

4.2 Запустить программу Eff_code_4, выбрать закладку «Префиксность» и выполнить задания, предлагаемые на этой закладке. Поясните результаты выполнения задания.

4.3 Переключиться на закладку «Коды Шеннона-Фано» и выполнить задания, предлагаемые на этой закладке. Поясните ход построения кода.

4.4 Вычислить энтропию и среднюю длину кодовой комбинации построенного в п.4.3 кода Шеннона-Фано.

4.5 Переключиться на закладку «Коды Хаффмена» и выполнить задания, предлагаемые на этой закладке. Поясните ход построения кода.

4.6 Вычислить энтропию и среднюю длину кодовой комбинации, построенного в п.4.5 кода Хаффмена.

5 СОДЕРЖАНИЕ ОТЧЕТА

5.1 Титульный лист.
5.2 Цель и программа лабораторной работы.
5.3 Вид декодируемой строки и результат декодирования.
5.4 Таблица с исходными данными и построенным кодом Шеннона-Фано.
5.5 Кодовое дерево Хаффмена и полученная таблица кодирования.
5.6 Расчетные данные средней длины кода и энтропии для кодов Шеннона-Фано и Хаффмена.
5.7 Выводы по результатам исследований.

6 КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1 Какие показатели используются для оценки степени сжатия сообщений?
6.2 В каких случаях можно осуществлять сжатие сообщений с частичной потерей информации?
6.3 Чем отличаются статические способы сжатия сообщений от динамических?

6.4 Каким образом вычисляется энтропия источника независимых и зависимых источников дискретных сообщений?
6.5 Как определяется средняя длина комбинаций неравномерного кода и чем расчет этого параметра отличается от расчета энтропии источника?
6.6 Какие неравномерные коды относятся к префиксным кодам? Почему неравномерные коды должны обладать свойством префиксности?
6.7 Продемонстрируйте на примере построение эффективного кода шеннона-Фано. В чем состоят недостатки такого кода?
6.8 Поясните принцип построения неравномерного эффективного кода Хаффмена и продемонстрируйте на примере построение этого кода.
6.9 Поясните на примере построение кодового дерева при кодировании по методу Хаффмена.
6.10 С какой целью осуществляют укрупнение алгоритма и как это сказывается на параметрах неравномерного кода?
6.11 Как вычисляется энтропия источника и какой параметр сообщения она характеризует?

Лабораторная работа 3

ИССЛЕДОВАНИЕ ДИНАМИЧЕСКОГО СПОСОБА СЖАТИЯ СООБЩЕНИЙ НЕРАВНОМЕРНЫМИ КОДАМИ

1 ЦЕЛЬ РАБОТЫ

Углубление теоретических знаний в области оптимального кодирования (компрессии) сообщений в информационных системах и исследование динамического способа построения префиксных неравномерных кодов «на лету», приобретение практических навыков исследования процессов динамического кодирования информационных сообщений неравномерными кодами.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Классический метод кодирования сообщений неравномерными кодами Хаффмена предполагает до начала преобразования знание вероятностей появления символов на выходе источника информации, причем символы упорядочиваются по убыванию вероятностей их возникновения. Для упорядоченного списка составляется кодовая таблица, в которой длина выходной комбинации определяется вероятностью исходного символа.

Естественно, что на передающей и приемной сторонах должны быть известны таблицы (кодовые деревья) для каждого сообщения, подвергающегося сжатию. Этот метод обладает двумя существенными недостатками. Первый

заключается в том, что для его реализации требуется два прохода кодируемого массива. При первом просмотре вычисляются вероятности появления каждого символа в сообщении и составляется таблица кода Хаффмена. На следующем этапе осуществляется кодирование на основании статической структуры дерева Хаффмена и передача символов в сжатом виде. Таким образом, выигрыш, полученный за счет сжатия данных, может заметно снижаться, особенно при передаче относительно коротких сообщений, в связи с необходимостью передавать декодеру дополнительную информацию о кодовом дереве. Второй недостаток – наличие задержки от момента поступления данных от источника до выдачи соответствующих кодовых комбинаций, что ограничивает использование неравномерного кодирования в синхронных сетях передачи информации, а также в системах, функционирующих в реальном времени.

В начале 70-х годов были разработаны однопроходные методы сжатия информации, основанные на классической процедуре кодирования Хаффмена. Все эти методы незначительно отличаются друг от друга и их суть заключается в том, что передатчик строит дерево Хаффмена в темпе поступления данных от источника, т.е. *"на лету"*. В процессе кодирования происходит *"обучение"* кодера на основе статистических характеристик источника сообщений, в ходе которого вычисляются оценки исходных вероятностей сообщения и производится соответствующая модификация кодового дерева Хаффмена. В связи с непрерывным изменением кодового дерева этот процесс получил название *динамического кодирования Хаффмена*.

Очевидно, что для правильного восстановления сжатых данных, декодер также должен непрерывно “учиться” наряду с кодером, осуществляя синхронное изменение кодовой таблицы на приемной стороне. Для обеспечения синхронности процессов кодирования и декодирования кодер выдает символ в несжатом виде, если он впервые появился на выходе источника, и отмечает его на кодовом дереве. При повторном появлении символа на входе кодера он передается неравномерной кодовой комбинацией, определяемой позицией символа на текущем кодовом дереве. Кодер корректирует дерево Хаффмена увеличением частоты передачи символов, которые уже введены в дерево, или наращивает дерево, добавляя в него новые узлы.

Важнейшим условием, которое должно соблюдаться при модификации кодового дерева, является сохранение свойств хаффменовского дерева. Для формулирования этих свойств обратимся еще раз к алгоритму построения оптимального кода Хаффмена (Лаб. работа №2). При статическом кодировании символы размещаются в списке в невозрастающем порядке весов (вероятностей). Затем производится объединение двух узлов наименьшего веса W_i, W_j и замена их внутренним узлом с весом, равным сумме исходных весов W_i+W_j. Вновь образованный узел размещается в списке таким образом, чтобы не нарушался порядок расположения узлов по весам. Этот процесс повторяется до тех пор, пока в списке не останется один, так называемый корневой, узел.

Кодовое дерево, имеющее m внешних узлов, является *хаффменовским*, если оно обладает следующими свойствами:

а) внешние узлы (листья) хафменовского кодового дерева имеют неотрицательный вес $W > 0$; каждый внутренний *(родительский)* узел имеет подчиненные (дочерние) узлы, а его вес равен сумме весов подчиненных *(дочерних)*;

б) на каждом уровне дерева (за исключением корневого) должно быть не менее одной пары узлов, имеющих общий родительский узел;

в) все узлы нумеруются в возрастающем порядке таким образом, что узлы с номерами *(2j - 1)* и *2j* являются узлами одного уровня для $1 \le j \le m-1$, а их общий родительский узел имеет более высокий уровень;

г) нумерация узлов соответствует тому порядку, в котором узлы объединяются в соответствии со статическим алгоритмом Хаффмена.

Существует несколько алгоритмов реализации динамического сжатия сообщений по методу Хаффмена. Наиболее распространенным является алгоритм FGK, получившим название по фамилиям его разработчика (Феллера, Галлагера и Кнута).

Введем некоторые обозначения, которые будут использоваться при анализе и синтезе динамического кодового дерева Хаффмена по алгоритму *FGK*: m -размер алфавита источника сообщений; z_j − j-й символ алфавита; $M(k) = z(1), z(2), \ldots, z(k)$ − первые k символов в сообщении; k − число символов в сообщении, обработанных до текущего момента времени; $z(k)$ − k-й символ в сообщении; K − количество различных символов, обработанных на текущий момент времени; W_j − число (вес) символов z_j, поступивших на момент обработки сообщения; l_j − расстояние от корня дерева до z_j-го листа.

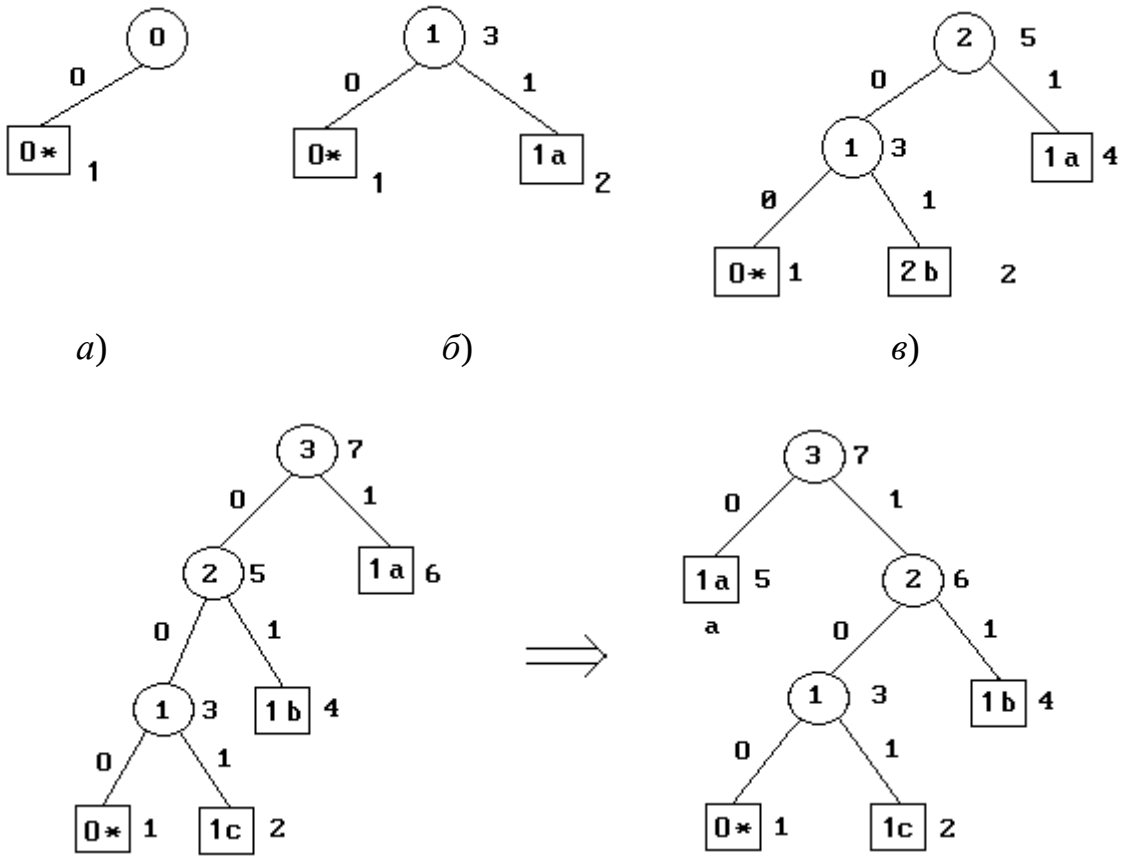
Суть алгоритма синтеза динамического кодового дерева Хаффмена состоит в процедуре вычисления листьев и построении бинарного дерева с минимальным весом пути $\sum_j W_j l_j$. Процедуру кодирования можно условно разбить на два этапа, хотя при реализации алгоритма они могут легко быть объединены в один. На первом этапе дерево Хаффмена, построенное после обработки сообщения $M(k)$, преобразуется в другое, эквивалентное исходному, которое затем простым приращением весов может быть преобразовано в хаффменовское дерево для $M(k+1)$.

Первый этап начинается после получения от источника символа $z(k+1)$ с присвоения статуса *текущего* узла листу $z(k+1)$. Затем происходит обмен текущего узла (включая образованное им поддерево), с узлом, имеющим *наибольший порядковый номер* с таким же весом. После этого в качестве нового текущего узла инициализируется родительский узел последнего текущего узла. Обмен узлами в случае необходимости многократно повторяется, пока не будет достигнут корень дерева. Несложно убедиться, что максимальное количество перестановок, которые могут понадобиться при модификации кодового дерева, равно высоте дерева $l_{\max}$.

На втором этапе инкрементируется лист дерева, соответствующий обрабатываемому символу и последующие промежуточные узлы, расположенные на пути движения от листа к корню дерева. Примеры построения и модификации кодового дерева приведены в [5].

3 МЕТОДИКА ПОСТРОЕНИЯ КОДОВОГО ДЕРЕВА ПРИ ДИНАМИЧЕСКОМ КОДИРОВАНИИ ПО МЕТОДУ ХАФФМЕНА

Рассмотрим алгоритм построения и модификации кодового дерева Хаффмена на примере кодирования строки символов "abcbdefe". Последовательность преобразований показана на рисунке 3.1. Кодер и декодер начинают строить кодовое дерево с корневого узла *(*рисунок 3.1,*а*), соединенного нулевой ветвью с листом с нулевым весом, обозначенного символом *. Существует лишь один такой узел в дереве и его позиция (кодовое слово) изменяется при построении хаффменовского дерева. Кодер считывает первый символ строки "a" и создает новый узел. Так как на каждом уровне, за исключением корневого, должно быть не менее двух узлов, имеющих общий родительский узел, кодер соединяет его с корневым узлом единичной ветвью (рисунок 3.1,*б*). Образованный лист является первичным месторасположением символа "a". На приемную сторону эта буква передается в несжатом виде стандартным *ASCII*-кодом.



а) *б*) *в*)

г)

Рисунок 3.1 – Пример построения хаффменовского кодового дерева при динамическом сжатии текстового сообщения

Дерево декодера на начальном этапе имеет вид, подобный дереву кодера до получения первого символа (рисунок 3.1,*а*) и таблица кодов Хаффмена

остается пустой. Декодер интерпретирует этот символ как неподвергавшийся сжатию с дальнейшим обозначением его на дереве аналогично передающей стороне. Для каждого последующего символа кодер и декодер производят сверку с символами, которые к этому времени представлены на дереве. Если символ уже имеется, то он передается в сжатом виде, причем кодовое слово определяется согласно позиции символа на дереве, где старший разряд является первым со стороны корня.

Если символ на дереве отсутствует, то он выдается кодером в несжатом виде, а нулевой узел превращается в родительский, расщепляясь на пару листьев: нулевой, соединенный нулевой ветвью с родительским и лист для нового символа, соединенный единичной ветвью.

Таким образом, при поступлении от источника буквы "b" кодовое дерево будет иметь вид, изображенный на рисунке 3.1,*в*. Как видно из рисунка, дерево является хаффменовским.

Введение в дерево третьего символа "c" нарушает свойства хаффменовского дерева и оно нуждается в модификации. Текущим узлом дерева на рисунке 3.1,*г* является лист 2. Так как символ "c" появился впервые, то вес листа $q = 0$. В связи с тем, что на дереве нет листьев, имеющих больший порядковый номер с нулевым весом, то увеличиваем q на 1 и назначаем новым текущим узлом родителя узла 2. Узел 3 имеет пока еще нулевой вес и нет узла аналогичного веса с большим порядковым номером. Поэтому обмена узлами не происходит, вес узла 3 увеличивается на единицу и следующим текущим узлом назначается 5-й узел, имеющий на данный момент вес 1. Обмениваем его вместе с образованным им поддеревом (узлы 1,2,3 и 4) с узлом 6 и увеличиваем вес q на единицу. Следующим текущим узлом назначается узел 7, который является корневым. Следовательно, модификация дерева завершена. Скорректированное дерево Хаффмена изображено на правой части рисунка 3.1,*г*.

Появление на входе кодера символа "b", который уже отмечен на дереве, также нарушает его свойства. Изменение конфигурации дерева осуществляется в соответствии с алгоритмом. Текущим узлом теперь является лист 4, имеющий вес W=*1*. Обмениваем его с узлом 5 и увеличиваем вес текущего узла на единицу. Дальнейшее построение кодового дерева при поступлении на вход кодера символов заданной последовательности можно посмотреть в [5].

Важнейшей проблемой при построении динамических кодов Хаффмена является однозначность декодирования в случае приема сообщения, когда еще не все символы используемого источником алфавита отмечены на кодовом дереве. Без специальных мер декомпрессор не в состоянии различить, отображает ли поступившая кодовая комбинация несжатый 8-ми битовый символ, либо это одна или несколько кодовых комбинаций неравномерного кода Хаффмена, уже отмеченных на дереве.

Оригинальный способ решения этой проблемы был предложен Д.Кнутом. Его суть состоит в том. что все символы используемого алфавита, которые еще не появились на выходе источника, отмечаются на дереве нулевым узлом. Поэтому, когда символ, подлежащий сжатию, генерируется впервые, кодер

"отмечает" его, вырабатывая *префиксную комбинацию.* Вслед за ней на выход поступает код несжатого символа. В качестве префиксной комбинации используется код, соответствующий *листу нулевого веса*.

4 ПРОГРАММА ЛАБОРАТОРНОЙ РАБОТЫ

4.1 Изучить по рекомендуемой литературе теоретический материал по теме динамического кодирования источников информации неравномерными кодами и разобрать примеры построения префиксных кодов. Выполняется в процессе домашней подготовки.
4.2 Закодировать путем построения кодового дерева символьную строку, приведенную в таблице вариантов, динамическим кодом Хаффмена.
4.3 Выполнить процесс декомпрессии сжатой строки, построить кодовое дерево и сравнить его с кодовым деревом на стороне компрессора.
4.4 Вычислить энтропию и среднюю длину кодовой комбинации динамического кода Хаффмена
4.5 Сформулировать выводы по результатам исследований.

Примечание: Номер варианта определяется последней цифрой зачетной книжки.

5 СОДЕРЖАНИЕ ОТЧЕТА

5.1 Титульный лист.
5.2 Цель и программа лабораторной работы.
5.3 Вид кодируемой строки и динамическая таблица кодирования неравномерным кодом.
5.4 Кодовые деревья при получении каждого из символов кодируемого сообщения.
5.5 Расчетные данные средней длины кода и энтропии для динамического кода Хаффмена.
5.6 Выводы по результатам исследований.

6 КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1 Какие показатели используются для оценки степени сжатия сообщений?
6.2 В чем состоит отличие динамического метода кодирования от статического?
6.3 Назовите свойства хаффменовского кодового дерева.
6.4 Продемонстрируйте на примере модификацию кодового дерева для сохранения свойств хаффменовского дерева.

6.5 Покажите на конкретном примере, как декодер разделяет принимаемые символы и различает, что принятая последовательность битов относится к символу, передаваемому впервые.
6.6 Проиллюстрируйте на примере, как осуществляется построение “на лету” кодового дерева на декодирующей стороне.
6.7 Объясните на примере процедуру построения и модификации кодового дерева по алгоритму FGK.
6.8 Запишите выражения для вычисления энтропии дискретного источника независимых и зависимых сообщений и поясните физический смысл энтропии.
6.9 Какой источник дискретных сообщений называют «Марковским» и как вычисляется энтропия марковского источника?
6.10 Запишите матрицу переходных вероятностей марковского источника дискретных сообщений.

Лабораторная работа 4

ИССЛЕДОВАНИЕ ДИНАМИЧЕСКОГО ЭФФЕКТИВНОГО КОДИРОВАНИЯ СООБЩЕНИЙ РАВНОМЕРНЫМИ КОДАМИ

1 ЦЕЛЬ РАБОТЫ

Углубление теоретических знаний в области оптимального кодирования (компрессии) сообщений в информационных системах и исследование динамического способа строк переменной длины равномерным кодом (метод LZW), приобретение практических навыков исследования процессов динамического кодирования информационных сообщений равномерными кодами.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Метод кодирования группы символов переменной длины равномерным кодом предложен Т. Уелчем в 1984 г. Метод получил название по именам его авторов *“*метод LZW*”*(*Lempel-Ziv-Welch*). При кодировании LZW-методом используется таблица строк, состоящая как из входных одиночных символов, так и из некоторых буквосочетаний. Причем каждой строке соответствует своя двоичная комбинация *(кодовое слово)* фиксированного размера.

Процедура динамического кодирования LZW-методом состоит в следующем. При начальном заполнении LZW-таблицы в нее вносятся строки, состоящие из одиночных символов. Затем, по мере поступления данных от источника информации, формируются строки, состоящие из нескольких символов. Так как

таблица имеет ограниченный размер, то строки, встречающиеся в тексте редко, исключаются, а на их место вносятся строки, имеющие большую частоту появления. Таким образом в процессе накопления статистики о сжимаемом сообщении происходит динамическая перестройка таблицы кодирования и адаптация ее к характеру передаваемых данных.

Компрессия данных начинается с инициализации таблицы, при которой в нее вносятся строки, состоящие из одиночных символов. Затем поступает первый входной символ, рассматриваемый как префикс некоторой строки PREFIX. После этого вводится следующий символ CHARACTER и образуется расширенная строка путем объединения префикса и одиночного символа PREFIX+CHARACTER. Далее осуществляется сопоставление вновь образованной строки со строками, существующими в таблице кодирования. Если строка PREFIX+CHARACTER имеется в таблице, то она становится новым префиксом, т.е. PREFIX=PREFIX+CHARACTER и вводится следующий символ CHARACTER, и процедура сопоставления строки в таблице повторяется снова. В противном случае, если последовательности PREFIX+CHARACTER в таблице строк нет, то на выход кодера выводится кодовая комбинация, соответствующая строке PREFIX, в таблицу вносится дополнительная строка PREFIX+CHARACTER, а символ CHARACTER становится новым префиксом. Если входная последовательность не исчерпана, то процедура образования строк и их сопоставления повторяется. Алгоритм сжатия данных имеет следующий вид:

Алгоритм 2.1

```
            ROUTINE LZW_COMPRESS
Initiale table to contain single-character strings
      INPUT  CHARACTER
      PREFIX = CHARACTER
      WHILE exists iput character  DO:
Next:   INPUT NEXT_CHARACTER
      CHARACTER = NEXT_CHARACTER
Step:   IF PREFIX + CHARACTER exists in the table THEN:
      PREFIX = PREFIX + CHARACTER
      REPEAT Step
      ELSE
      OUTPUT the code for PREFIX
      ADD PREFIX + CHARACTER  to the strig table
      PREFIX = CHARACTER
      REPEAT Next
      END.
```

Пример построения кода по методу LZW можно посмотреть в [5].

Однозначность декомпрессии сжатых сообщений достигается тем, что в процессе LZW-декодирования создается такая же таблица строк, что и при кодировании. Полученная кодовая комбинация разделяется с помощью таблицы строк на префиксную строку PREFIX и одиночный символ CHARACTER. Символ расширения CHARACTER выдается на выход, а префиксная строка PREFIX вновь разделяется на префикс и одиночный символ. Эта операция рекурсивна и продолжается до тех пор, пока префиксная строка не выродится в одиночный символ, который и завершает процедуру декодирования. Символ, выводимый последним, является крайним левым элементом строки, то есть первым символом, с которым оперировал компрессор при синтаксическом анализе поступающих данных.

Изменения в таблице строк декомпрессора осуществляются для каждой полученной кодовой комбинации, за исключением единственного первого символа. В процессе преобразования кода последний символ, используемый в качестве расширения, объединяется с предыдущей строкой, образуя тем самым новую строку. Этой строке ставится в соответствии очередная кодовая комбинация, которая заносится в таблицу строк декодера. Очевидно, что внесенная комбинация совпадает с аналогичной строкой таблицы, образованной компрессором. Таким образом, декомпрессор путем наращивания создает такую же самую таблицу строк, которая используется при сжатии.

Процедура декодирования сводится к следующему. Первая поступившая кодовая комбинация, обозначаемая переменной OLD_Code, всегда является кодом одиночного символа, и она без изменения выдается на выход декодера. Затем вводится новое кодовое слово New_Code, которое для дальнейшего использования при формировании новой строки сохраняется в виде переменной NEX_Code. Если NEW_Code отображает расширенную строку PREFIX+ CHARACTER, то выделяется символ расширения строки CHARACTER и выдается на выход, а префикс вновь подвергается синтаксическому анализу. Операция повторяется до тех пор, пока префикс не будет представлен одиночным символом CHARACTER. После этого в таблице строк декодера образуется новая строка путем объединения первой комбинации OLD_Code и идентифицированным последним символом, а новое кодовое слово NEXT_Code становится для следующего цикла старым кодом OLD_Code. Затем вводится очередная комбинация и процедура повторяется снова. Алгоритм декомпрессии может быть представлен в следующем виде:

Алгоритм 2.2

```
    ROUTINE LZW_DECOMPRESS
READ  OLD_Code
OUTPUT  OLD_Code
```

```
WHILE  exists input character  DO:
 READ NEW_Code
 NEXT_Code = NEW_Code
  IF NEW_Code = PREFIX + CHARACTER
    OUTPUT CHARACTER
    NEW_Code = PREFIX
  END of IF
  ELSE  IF NEW_Code = CHARACTER
  OUTPUT CHACTER
 ADD OLD_CODE + CHARACTER to string table
 OLD_Code =  NEXT_Code
END of WHILE.
```

Отличительной особенностью рассмотренного LZW-алгоритма является простота реализации и относительно высокая скорость декомпрессии. Однако ему присущи два недостатка. Первый заключается в том, что декомпрессор восстанавливает символы обрабатываемой строки в обратном порядке, то есть первым выделяется последний символ строки.

Устранить этот недостаток просто, если в процессе декодирования строки выходные символы перед выдачей их потребителю помещать в стек, а по завершении декодирования символы считывать из стека в нужной последовательности (поступивший последним, выводится первым), т.е. в той, в которой они поступали от источника.

Второй недостаток – аварийный сбой процедуры декомпрессии при наличии во входной символьной строке повторяющихся групп символов вида *CsCsCs* (здесь С- одиночный символ, *s*-некоторая произвольная строка), при условии, что строка *Cs* уже содержится в таблице компрессора. В этом случае кодер, анализируя строку, выдает из таблицы код *(Cs)* и вводит в таблицу следующую строку *CsC*. Затем, обнаружив во входной последовательности строку *CsC*, кодер выводит только что образованную кодовую комбинацию *(CsC)*. Декомпрессор не может декодировать эту комбинацию, так как ее еще нет в таблице строк декомпрессора. Очевидно, что она не может быть сформирована на основе предыдущей строки *Cs*, поскольку декомпрессор не знает символа расширения (в данном случае *C*) до приема следующей строки. В [5] приведен модифицированный алгоритм, в котором второй недостаток отсутствует.

3 ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

В качестве лабораторного стенда используется персональный компьютер с установленной программой для исследования процедуры компрессии и декомпрессии текстовых сообщений методом LZW. Для запуска лабораторной

установки следует активировать файл Project1, находящийся в папке KIZI2 v2+step+code_length. В результате откроется окно (рисунок 3.1).

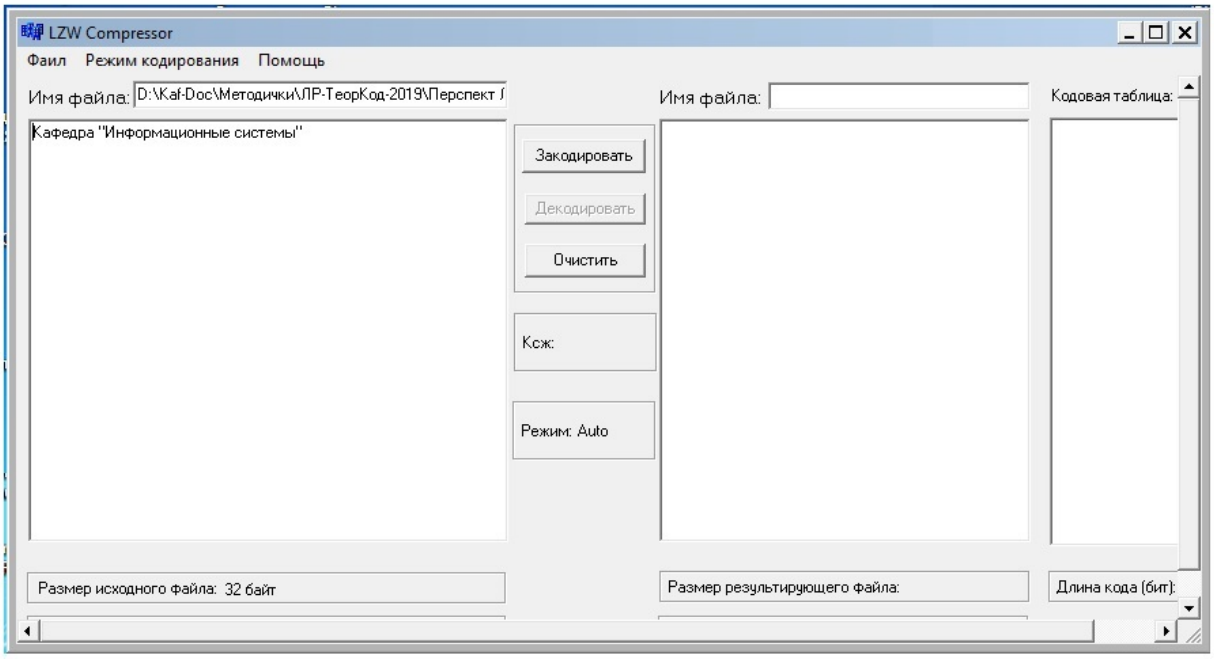


Рисунок 3.1 – Интерфейс лабораторной установки для исследования сжатия данных методом LZW

В лабораторной работе исследуется процесс сжатия сообщений, представленных в виде текстовых файлов. Процесс сжатия может быть выбран непрерывным или пошаговым.

В программе имеются следующие кнопки и пункты меню:

Кнопки:

а) Закодировать – выполняет сжатие открытого исходного файла в файл формата lzw.

б) Декодировать – выполняет распаковку открытого исходного файла в файл формата txt.

в) Очистка –очищает все поля вывода информации, а также закрывает исходный и результирующий файлы.

Пункты меню:

а) Открыть... – выбор исходного файла для сжатия/распаковки.

б) Очистить –очищает все поля вывода информации, а также закрывает исходный и результирующий файлы.

в) Пошаговый – задает пошаговый режим сжатия файла. В поле «Режим» выводится надпись «Step».

г) Автоматический – задает автоматический режим сжатия файла. В поле «Режим» выводится надпись «Auto».

д) Справка – выводит справочную систему программы.

е) О Программе – выводит сведения о разработчике программы, версии программы и т.п.

Так же для управления выводом текста исходного и результирующего файла служат два CheckBox под соответствующими окнами. Если в CheckBox стоит галочка, то текст файла будет выведен на экран, иначе – не будет выведен на экран.

4 ПРОГРАММА ЛАБОРАТОРНОЙ РАБОТЫ

4.1 Изучить по рекомендуемой литературе теоретический материал по теме динамического кодирования строк переменной длины равномерными кодами и разобрать примеры построения кодов LZW. Выполняется в процессе домашней подготовки.
4.2 Закодировать путем построения кодового дерева символьную строку, приведенную в таблице вариантов, динамическим кодом LZW вручную с построением таблицы кодирования.
4.3 Выполнить вручную декодирование сжатой строки, используя построенную таблицу кодирования.
4.4 Выполнить процедуру кодирования и декодирования заданной строки на стенде и сравнить таблицы кодирования, построенные вручную и автоматически, а также закодированные последовательности.
4.5 Рассчитать коэффициент сжатия информации и среднее количество бит на символ.
4.6 Закодировать произвольный текстовый файл длиной не менее 1 килобайта и вычислить вручную коэффициенты сжатия и компрессии.
4.7 Сформулировать выводы по работе.

5 СОДЕРЖАНИЕ ОТЧЕТА

5.1 Титульный лист.
5.2 Цель и программа лабораторной работы.
5.3 Вид исходной кодируемой и выходной строки и динамическая таблица кодирования.
5.4 Расчетные данные средней длины кода и энтропии для динамического кода LZW.
5.5 Выводы по результатам исследований.

6 КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1 Какие показатели используются для оценки степени сжатия сообщений?
6.2 В чем состоит отличие динамического метода кодирования от статического?

6.3 Расскажите о принципе сжатия сообщений методом LZ.
6.4 Как представляется кодовая комбинация, отображающая кодируемую подстроку при сжатии методом LZ?
6.5 Продемонстрируйте на примере формирование таблицы кодирования и сжатого сообщения при сжатии методом LZW.
6.6 Почему на начальном этапе сжатия методом LZW при незаполненной таблице кодирования вместо сжатия происходит расширение сообщения?
6.7 Как можно ускорить процедуру поиска подстроки в таблице кодирования?
6.8 Каковы недостатки алгоритма компрессии методом LZW и как они могут быть устранены?
6.9 Какой источник дискретных сообщений называют «Марковским» и как вычисляется энтропия марковского источника?
6.10 Запишите матрицу переходных вероятностей марковского источника дискретных сообщений.

Лабораторная работа 5

ИССЛЕДОВАНИЕ МЕТОДОВ И УСТРОЙСТВ ИСПРАВЛЕНИЯ ОДИНОЧНЫХ ОШИБОК

1 ЦЕЛЬ РАБОТЫ

Углубление теоретических знаний в области помехоустойчивого кодирования данных в информационных системах, исследование способов построения кода, исправляющего одиночные ошибки и схемной реализации кодера и декодера кода Хемминга, а также приобретение практических навыков исследования процессов помехоустойчивого кодирования информационных сообщений на программно-аппаратном уровне.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Код Хэмминга – один из наиболее эффективных кодов, позволяющий исправлять одиночные ошибки. Кодовое расстояние d=3. Этот код образуется дополнением информационной части передаваемого блока, состоящей из k битов, r проверочными элементами. При выборе длинны передаваемого блока n и количества проверочных элементов r руководствуются неравенством [1,2,4,6]

$$2^r \geq n+1.$$

Учитывая, что $r = n-k$, неравенство может быть представлено в виде

$$2^k \leq 2^n / (n+),$$

где n и k принимают только целые значения. Неравенство является исходным для определения длинны кодовой комбинации по заданному числу k.

Первый проверочный бит $П_1$ кода Хэмминга образуется суммированием по модулю 2 всех нечетных элементов кодируемого блока, начиная с первого.

$$П_1=a_1 \oplus a_3 \oplus a_5 \oplus a_7 \ldots \quad (2.1)$$

Результат проверки $П_2$ определяет второй разряд проверочной комбинации (синдрома ошибки). Он вычисляется суммированием тех элементов блока, номера которых соответствуют n–разрядным двоичным числам, имеющим единицу во втором разряде, то есть

$$П_2 = a_2 \oplus a_3 \oplus a_6 \oplus a_7 \oplus a_{10} \oplus a_{11} \ldots \quad (2.2)$$

Третья проверка $П_3$ охватывает разряды, номера которых соответствуют n разрядам, имеющим единицу в третьем разряде.

$$П_3=a_4 \oplus a_5 \oplus a_6 \oplus a_7 \oplus a_{12} \oplus a_{13} \oplus a_{14} \oplus a_{15} \ldots \quad (2.3)$$

Аналогичным образом находятся разряды, охватываемые четвертой, пятой и т.д. проверками.

$$П_4=a_8 \oplus a_9 \oplus a_{10} \oplus a_{11} \oplus a_{12} \oplus a_{13} \oplus a_{14} \oplus a_{15} \ldots \quad (2.4)$$

$$П_5=a_{16} \oplus a_{17} \oplus a_{18} \oplus a_{19} \oplus a_{12} \oplus a_{20} \oplus a_{21} \ldots \quad (2.5)$$

В принципе местоположение проверочных элементов не имеет значения. Их можно размещать перед информационными символами, после них, чередуя с информационными. Если их расположить на местах, кратных степени 2, то есть на позициях 1, 2, 4, 8 и т.д., то код двоичного числа, образованного проверочными элементами, на приемной стороне будет указывать номер разряда, в котором произошла ошибка [1,4].

Основной операцией в кодирующих и декодирующих устройствах кода Хэмминга является суммирование по модулю 2 передаваемых единичных элементов в соответствии с формулами (2.1-2.5). Для упрощения технической реализации (исключения многоразрядных параллельных сумматоров, входного накопителя) вначале в канал посылаются информационные биты, а затем – проверочные [4]. При таком способе формирование контрольных элементов осуществляется с помощью одноразрядных последовательных сумматоров по

модулю 2 одновременно с передачей информационных разрядов. Чтобы сохранить корректирующие свойства кода Хэмминга, необходимо произвести перестановку разрядов в проверочных соотношениях (2.1-2.5) с учетом изменения номеров позиций суммируемых элементов за счет вынесения в конец блока проверочных битов. При такой перестановке уравнения проверки будут охватывать следующие разряды:

$$\begin{aligned} &П1: \ 1, 2, 4, 5, 7, 9, 11, 12, 14, \ldots ; \\ &П2: \ 1, 3, 4, 6, 7, 10, 11, 13, 14, \ldots ; \\ &П3: \ 2, 3, 4, 8, 9, 10, 11, \ldots ; \\ &П4: \ 5, 6, 7, 8, 9, 10, 11, \ldots . \end{aligned} \qquad (2.6)$$

Бит первой проверки будет располагаться на (k+1)-й позиции блока, второй — на (k+2)-й, последний — на n-й позиции.

Кодирования сообщений помехозащищенным кодом Хемминга может осуществляться программным или аппаратным способом. Недостатком программного способа кодирования является его сравнительно невысокое быстродействие. Поэтом в серверных устройствах защиты памяти и в высокоскоростной аппаратуре передачи данных кодер и декодер обычно реализуется аппаратно.

Обобщенная функциональная схема кодирующего устройства Хемминга изображена на рисунке 2.1. Устройство осуществляет кодирование 4-разрядных информационных слов по коду Хемминга и побитную передачу информационных и проверочных битов в канал связи. Кодирующее устройство для четырехразрядного информационного слова a_1,a_2,a_3,a_4 содержит три сумматора по модулю 2 и параллельно-последовательный регистр RG. На выходах сумматоров формируются проверочные биты b_1,b_2 и b_3 значения которых определяются по правилам (2.6).

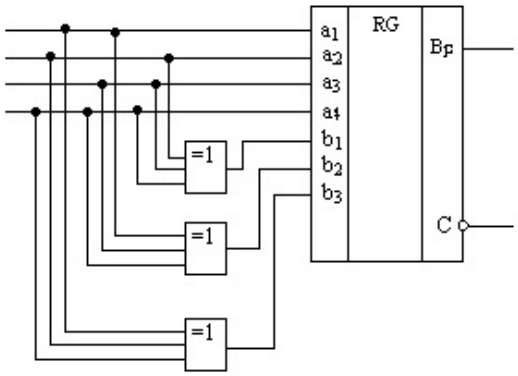


Рисунок 2.1 – Функциональная схема кодирования четырехразрядных слов кодом Хемминга

Функциональная схема декодера кода Хемминга показана на рисунке 2.2. В ее состав входит преобразователь последовательного кода в параллельный, реализованный на основе регистра RG сдвига, блок формирования синдрома ошибок, выполненный на основе сумматоров по модулю 2 и блок исправления

ошибки, построенный основе логических схем И и сумматоров по модулю 2. Коррекция ошибки выполняется путем суммированию по модулю 2 ошибочного элемента и логической 1, поступающей с одной из схем И. В результате суммирования по модулю 2 осуществляется инвертирование искаженного бита.

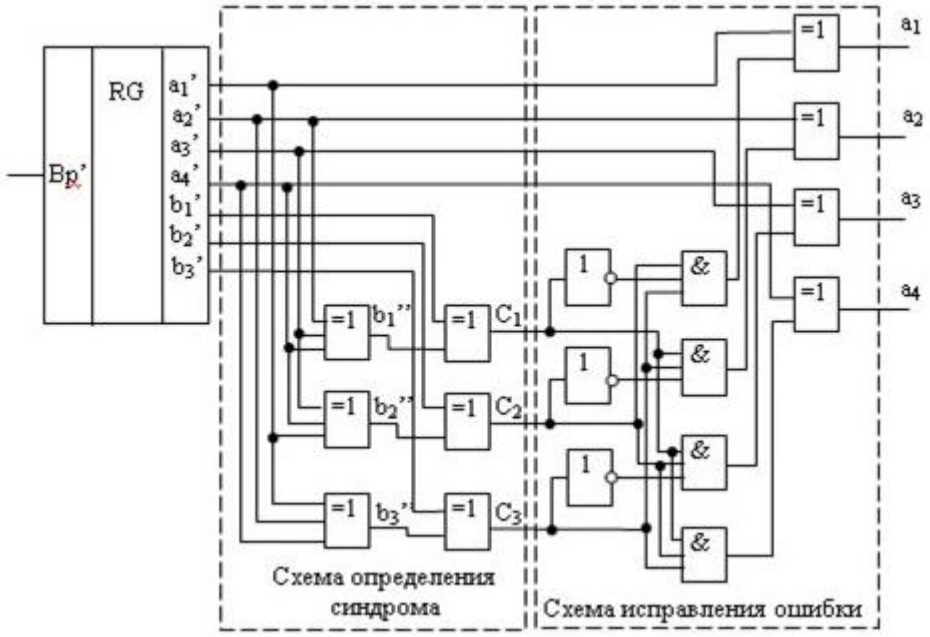


Рисунок 2.2 – Функциональная схема декодирования четырехразрядных слов, закодированных кодом Хемминга

3 ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

В качестве лабораторного стенда используется персональный компьютер с установленной демоверсией программы для моделирования процессов функционирования электронных устройств «Proteus 8». С особенностями работы с этим инструментальным средством студенты знакомы по предыдущим курсам («Электроника», «Инструментальные средства информационных систем»). **Proteus** — это пакет программ для систем автоматического проектирования (САПР), объединяющий в себе две основных программы: **ISIS** - средство разработки и отладки в режиме реального времени электронных схем и **ARES** - средство разработки печатных плат. В данной дисциплине используется только система **ISIS.** При запуске программы «Proteus» открывается окно, показанное на рисунке 3.1.

Схема модели кодирования четырехбитовых слов помехозащищенным кодом Хемминга изображена на рисунке 3.2. Основным узлом устройства является блок вычисления синдрома ошибки, который реализован на основе трехразрядных сумматоров по модулю два U1:A U1:B (микросхема 74S135). Эта микросхема содержит четыре трехвходовые сумматоры по модулю 2. Причем, каждые два сумматора имеют один общий вход.

Для фиксации информационных и проверочных бит используются два четырехбитовых параллельных регистра U2 и U3 (микросхема 74179). Индикация

состояния регистров осуществляется с помощью светодиодного индикатора U5 (микросхема LED BARGRAF-GRN).

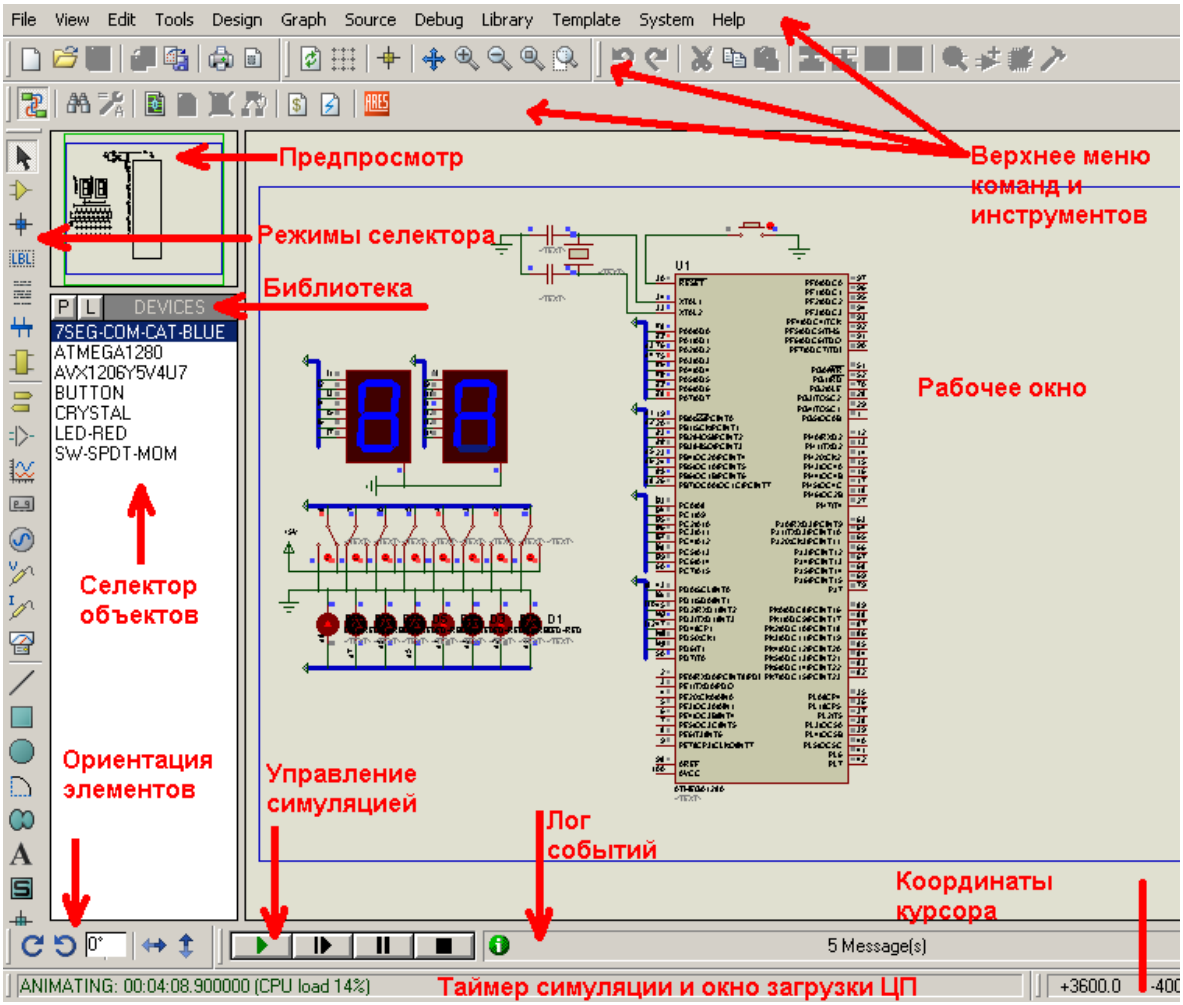


Рисунок 3.1 – Окно симулятора электронных схем в реальном времени

Входные кодовые комбинации задаются с помощью ключей SW-1…SW-4. Запись входных данных в регистры производится кратковременным нажатием кнопки SW-5.

Схема декодера Хемминга показана на рисунке 3.3. Декодер состоит из следующих функциональных модулей:

– модуля формирования синдрома ошибки, построенного на трех сумматорах по модулю 2 (элементы U1:A U1:B, построенные на основе микросхемы 74S135);

– модуля сравнения синдрома ошибки, входной комбинации с синдромом ошибки, сформированным декодером (элементы U2:A…U2:C на основе микросхемы 74HC386);

– дешифратора ошибки (элементы U3:A…U3:C, микросхема 7404 и U6:A…U7:A, микросхема 7411);

– модуля исправления ошибки (элементы U5:A U1:D, на базе микросхемы 74HC386).

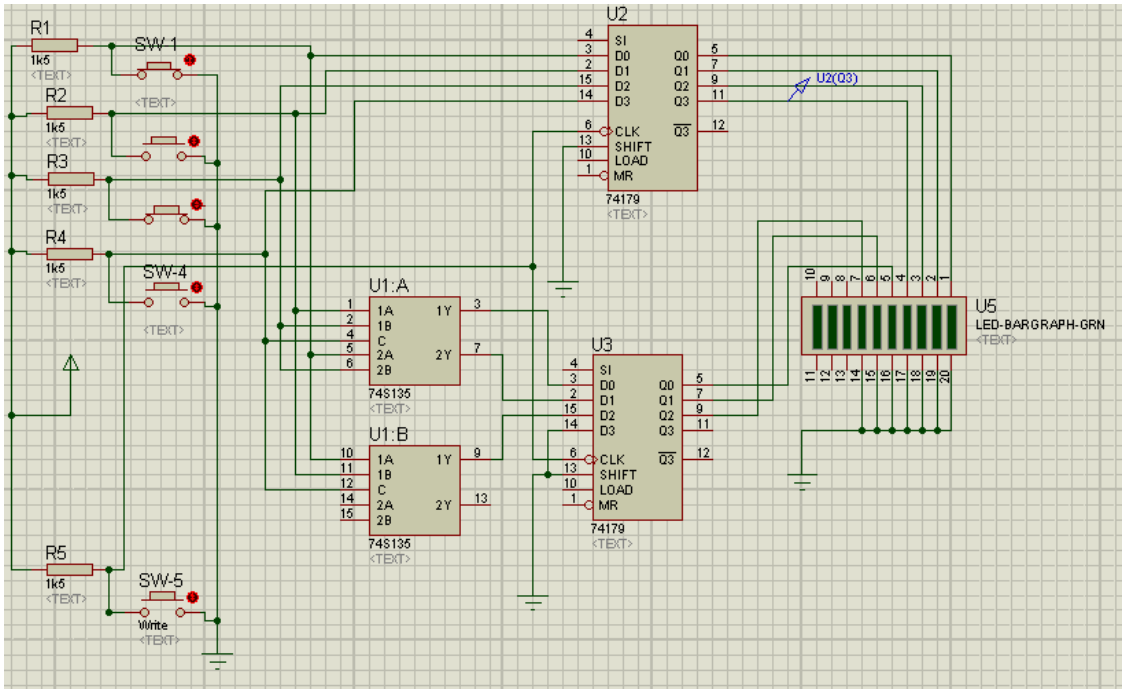


Рисунок 3.2 – Схема модели кодера Хемминга

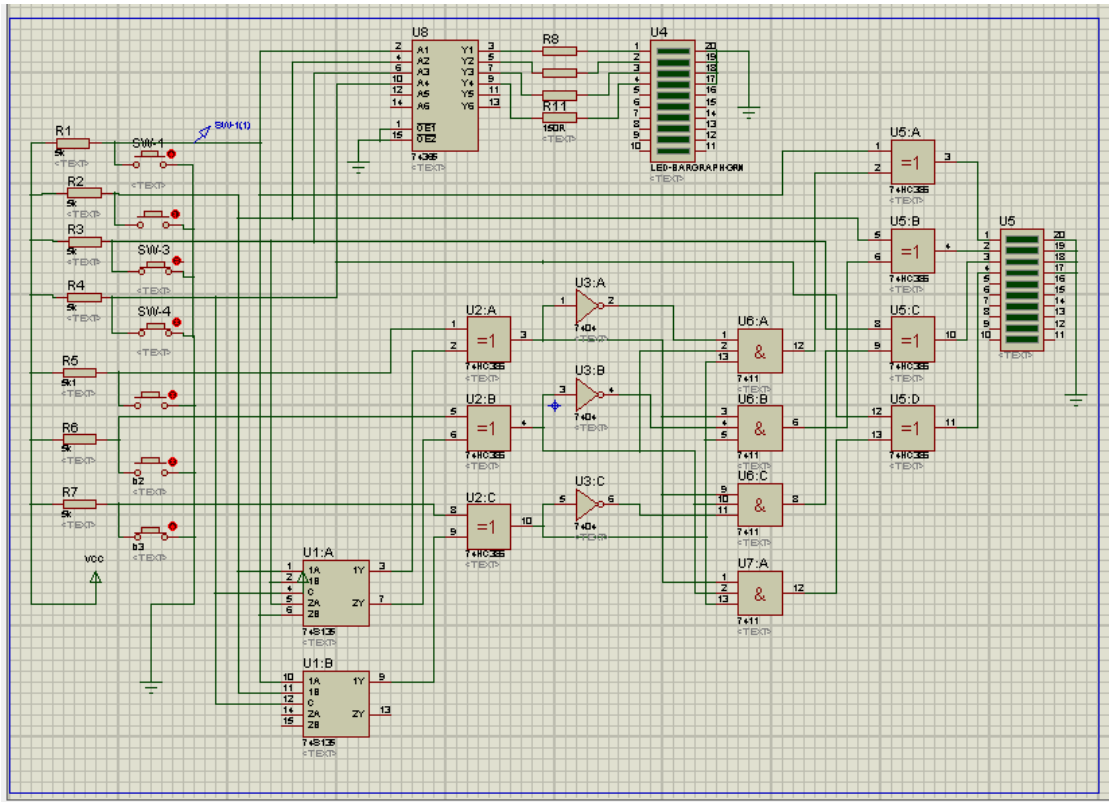


Рисунок 3.3 – Схема модели декодера, корректирующего одиночные ошибки

Для фиксации и индикации информационных элементов входной кодовой комбинации используется параллельный четырехразрядный регистр (U8, микросхема 74365) и индикатор (U4, LED-BARGRAF). Резисторы R8…R11 служат для ограничения входного тока, потребляемого индикатором. Индикатор U5 отображает исправленную кодовую комбинацию.

Ключевые элементы SW-1…SW-4 и резисторы R1…R4 предназначены для установки информационных бит кодовой комбинации, а элементы SW-5…SW-7 и резисторы R5…R7 служат для задания проверочных бит этой же кодовой комбинации. С помощью этих же ключей имитируются одиночные ошибки во входной кодовой комбинации.

4 ПРОГРАММА ЛАБОРАТОРНОЙ РАБОТЫ

4.1 Изучить по рекомендуемой литературе теоретический материал по теме динамического кодирования строк переменной длины равномерными кодами и разобрать примеры построения кодов LZW. Выполняется в процессе домашней подготовки.
4.2 Рассчитать по выражениям (2.6) значения проверочных бит для всех возможных 16-ти значений информационных комбинаций.
4.3 Составить в рабочем поле эмулятора схему кодера, изображенную на рисунке 3.2.
4.4 Набрать с помощью ключей SW-1…SW-4 16 кодовых комбинаций и записать значения проверочных битов.
4.5 Сравнить полученные экспериментально проверочные комбинации с комбинациями, вычисленными теоретически.
4.6 Составить в рабочем поле эмулятора схему декодера Хемминга.
4.7 Подавать последовательно с помощью ключей SW-1…SW-7 все закодированные кодом Хемминга комбинации, полученные в п.4.4 и проследить отображаемую информацию на светодиодных индикаторах.
4.8 Изменяя поочередно один из битов входной информации, имитируя одиночную ошибку, проследить процесс коррекции ошибки декодером.
4.9 Сформулировать выводы по работе и оформить отчет.

Примечание. Демоверсия обладает полным функционалом, за исключением возможности сохранения проекта. Поэтому перед началом выполнения работы необходимо сначала составить схемы исследуемых устройств в бумажном виде, а в процессе выполнения работы в лаборатории составить ее в рабочем окне программы моделирования и исследовать в соответствии с программой исследования.

5 СОДЕРЖАНИЕ ОТЧЕТА

5.1 Титульный лист.
5.2 Цель и программа лабораторной работы.
5.3 Кодовые комбинации, рассчитанные теоретически и получены при кодировании кодом Хемминга.
5.4 Электрические принципиальные схемы кодера и декодера.
5.5 Выводы по результатам исследований.

6 КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1 Какие показатели используются для оценки помехоустойчивых кодов?
6.2 Как определяется кодовое расстояние кода и какие соотношения должны выполняться при необходимости обнаружения и исправления ошибок?
6.3 Сколько проверочных битов должно быть в кодовой последовательности, состоящей из 64 битов?
6.4 Запишите правило вычисления пятого проверочного элемента при кодировании по Хеммингу.
6.5 Поясните по принципиальной схеме, как вычисляются проверочные комбинации.
6.6 Поясните по принципиальной схеме декодера, как исправляются одиночные ошибки.
6.7 В чем состоит отличие блочных кодов от непрерывных?
6.8 Поясните, каким образом исправляются ошибки в итеративном коде.

Лабораторная работа 6

ИССЛЕДОВАНИЕ ЦИКЛИЧЕСКИХ КОДОВ

1 ЦЕЛЬ РАБОТЫ

Углубление теоретических знаний в области помехоустойчивого кодирования данных в информационных системах, исследование способов построения циклических кодов и реализации кодирующих и декодирующих устройств, приобретение практических навыков моделирования и исследования устройств кодирования информационных сообщений циклическими кодами.

2 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Циклические коды находят наибольшее распространение в системах передачи данных с решающей обратной связью, что обусловлено их высокими корректирующими свойствами, сравнительно простой реализацией, невысокой избыточностью. Особенно эффективны эти коды при обнаружении пакетов ошибок. Циклические коды относят к блочным систематическим кодам, где каждая комбинация кодируется самостоятельно в виде блока таким образом, что информационные k и проверочные r элементы всегда находятся на определенных местах. Для упрощения процедуры кодирования и декодирования проверочные биты размещают в конце блока. Кодирование передаваемого сообщения осуществляется умножением двоичной последовательности $G(x)$ на одночлен x^r, имеющий ту же степень, что и образующий полином $P(x)$, с добавлением к этому произведению остатка $R(x)$, полученного после деления произведения $G(x)x^r$ на образующий полином. Таким образом, передаваемое в канал связи сообщение $F(x)$ имеет вид:

$$F(x)= G(x)x^r +R(x). \qquad (2.1)$$

При декодировании принимаемая последовательность $F(x)$ снова делится на образующий полином $P(x)$. Полученный нулевой остаток $R(x)=0$ свидетельствует об отсутствии ошибок в принятом блоке, а отличие от нуля — о наличии ошибок. Анализируя вид остатка, можно определить номера искаженных разрядов и скорректировать их.

Для построения циклических кодов в качестве образующих полиномов используются неприводимые многочлены. Неприводимыми называются многочлены, делимые без остатка только на себя и на единицу. $P(x)$ может быть представлен в алгебраической форме, либо в виде двоичного или восьмеричного числа. Например, для полинома вида $x^8+ x^4+ x^3+ x+1$ двоичная запись имеет вид *100 011 011*, а соответствующая ему восьмеричная – 433.

При выборе образующего полинома $P(x)$ следует иметь в виду, что степень полинома не может быть меньше числа проверочных элементов r.

Для систем передачи данных общего пользования с решающей обратной связью и использованием циклических кодов в режиме обнаружения ошибок стандартом (ГОСТ-17422-82) предусмотрено несколько видов полиномов:

$x^8+ x^4+ x^3+ x+1$ - кодовое расстояние равно 4, длина блоков должна быть не более 2^7 битов, а длина проверочной области 8 битов.

$x^8+ x^4+ x^3+ x+1$ - минимальное кодовое расстояние 4, длина блока должна быть не более 2^{15} битов. При повышенных требованиях к вероятности необнаруженной ошибки и при низком качестве незащищенного от ошибок канала должен использоваться образующий полином 16-й степени с минимальным кодовым расстоянием равным 6:

$x^{16}+ x^{15}+ x^{13}+ x^{11}+ x^5+ x^3+ x+1$. При этом длина блока должна быть не более 2^{16} битов. Полиномам 16-й степени соответствует проверочная область длиной

16 битов. Стандартом разрешается в обоснованных случаях использовать полиномы 24-й и 32-й степени с длиной проверочных областей 24 и 32 бита соответственно.

Для построения кодирующего устройства циклического кода необходимо в соответствии с (2.1) выполнить две процедуры: умножить многочлен $Q(x)$ на x^r и полученное произведение разделить на образующий полином $P(x)$ по модулю $P(x)$. Для проведения первой операции не требуется специального устройства, так как умножение многочлена на x^r означает добавление к нему r нулевой со стороны младшего разряда, т.е., после передачи k информационных элементов за ними следуют r проверочных. Реализация процедуры циклического кодирования программными методами требует больших затрат машинного времени. Так при делении 32-х разрядного слова на 16-ти разрядное восьмиразрядный процессор затрачивает время около 2,5 мс. В то же время схемотехнические кодеры и декодеры циклических кодов отличаются малыми аппаратурными затратами и способностью работать в реальном времени. Поэтому в современных программируемых устройствах защиты от ошибок (УЗО) и различных адаптерах связи циклическое кодирование и декодирование реализуется схемно.

В качестве делителей информационного полинома на образующий полином в кодеках циклических кодов применяются устройства, построенные на основе регистров сдвига с обратными связями и сумматоров по модулю 2, причем схема делителя определяется видом образующего полинома []. При этом:

- количество триггеров регистра сдвига выбирается равным степени образующего полинома r, причем триггер нулевой ячейки не ставится (его функцию исполняет выходной элемент источника сигналов);
- число сумматоров на единицу меньше степени образующего полинома;
- сумматор ставится после каждой ячейки, начиная с нулевой, для которой существует ненулевой член в полиноме. После ячейки, соответствующей старшему разряду, сумматор не ставится.

На первые входы сумматоров подаются сигналы с предыдущих ячеек регистра, а на вторые ─ с выхода делителя, т.е. с ячейки x^r. Очевидно, нет необходимости ставить сумматор после ячейки x^r.

Структурная схема кодирующего устройства циклического кода с образующим полиномом $P(x)=x^4+x+1$ изображена на рисунке 2.1. Прямоугольниками T на схеме обозначены ячейки памяти (триггеры), а кружками сумматоры по модулю 2. В исходном состоянии К1 находится в нижнем положении, а К2 в разомкнутом состоянии. Информация, подлежащая кодированию, поступает одновременно на вход регистра сдвига и через сумматор - на схему деления на $P(x)$. Затем схема управления кодирующего устройства К1 в верхнее положение, а ключ К2 замкнутое состояние, с помощью которого осуществляется обратная связь в делителе. В течение последующих k тактов на выход устройства выводятся информационные элементы и выполняется деление информационной части на образующий пол ином. После этого ключ К1 снова переключается в нижнее положение, а К2 размыкается. В последующие тактов осуществляется выдача на выход остатка от деления, зафиксированного триггерами делителя.

Причем, при выводе остатка из делителя выполняется ввод следующих данных от источника.

Основу декодирующих устройств циклических кодов также составляют делители многочленов на образующий полином [4,6]. Признаком наличия ошибок принятой последовательности является ненулевой остаток от деления этой последовательности на *P(x)*.

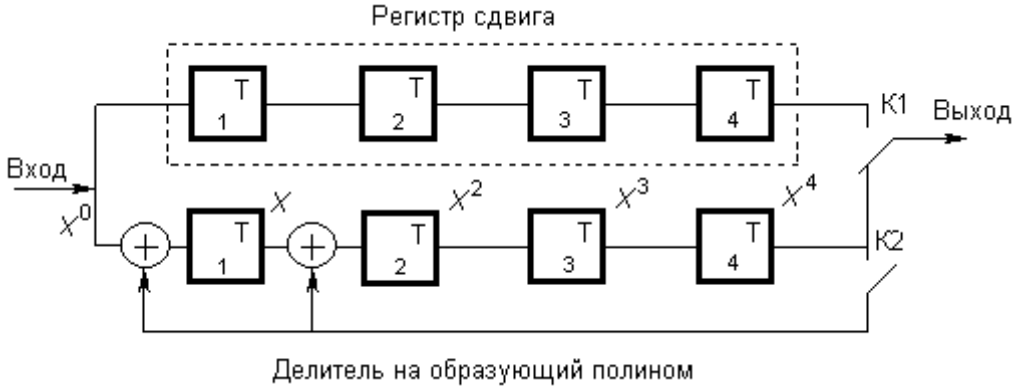


Рисунок 2.1 - Структурная схема циклического кодера

До завершения процесса деления необходимо запомнить поступивший блок в буферном накопителе. После окончания цикла производится опрос делителя, и в случае ошибки принятый блок стирается. При нулевом остатке блок выводится получателю через ключевой элемент КЛ, а на его место записывается следующее. Структурная схема декодера изображена на рисунке 2.2.

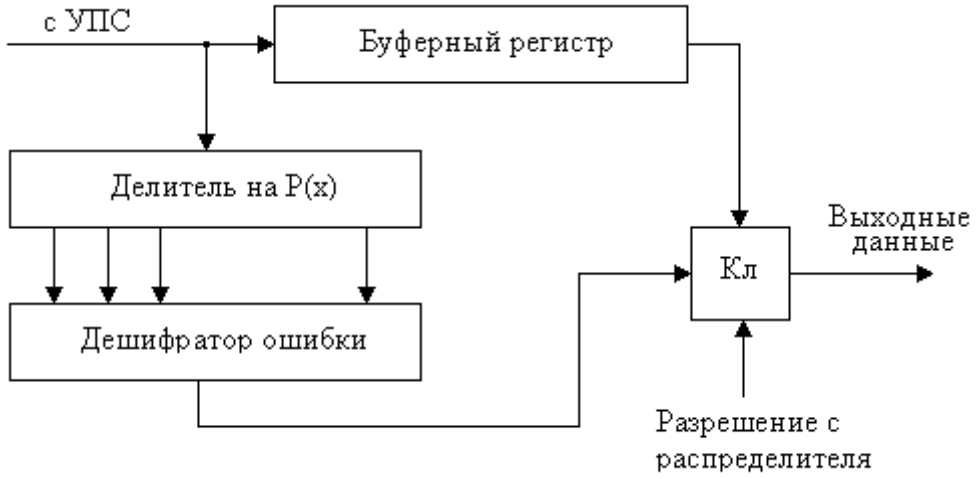


Рисунок 2.2 – Структурная схема декодера циклического кода

В процессе приема устройство управления (на схеме не показано) вырабатывает управляющие импульсы таким образом, что в буферный регистр заносятся только информационные символы, а на делитель подаются все информационные элементы, которые участвовали в процессе деления в кодере, а также проверочная последовательность *R(x)*.

3 ОПИСАНИЕ ЛАБОРАТОРНОЙ УСТАНОВКИ

В качестве лабораторного стенда используется персональный компьютер с установленной демоверсией программы для моделирования процессов функционирования электронных устройств «Proteus 8». С особенностями работы с этим инструментальным средством студенты знакомы по предыдущим курсам («Электроника», «Инструментальные средства информационных систем»). **Proteus** — это пакет программ для систем автоматического проектирования (САПР), объединяющий в себе две основных программы: **ISIS** - средство разработки и отладки в режиме реального времени электронных схем и **ARES** - средство разработки печатных плат. В данной дисциплине используется только система **ISIS.** Работа с этой системой подробно рассмотрена в методических указаниях по исследованию кодирования Хемминга.

Схема модели кодирования четырехбитовых слов помехозащищенным циклическим кодом с образующим полиномом вида x^4+x^2+x+1 изображена на рисунке 3.1.

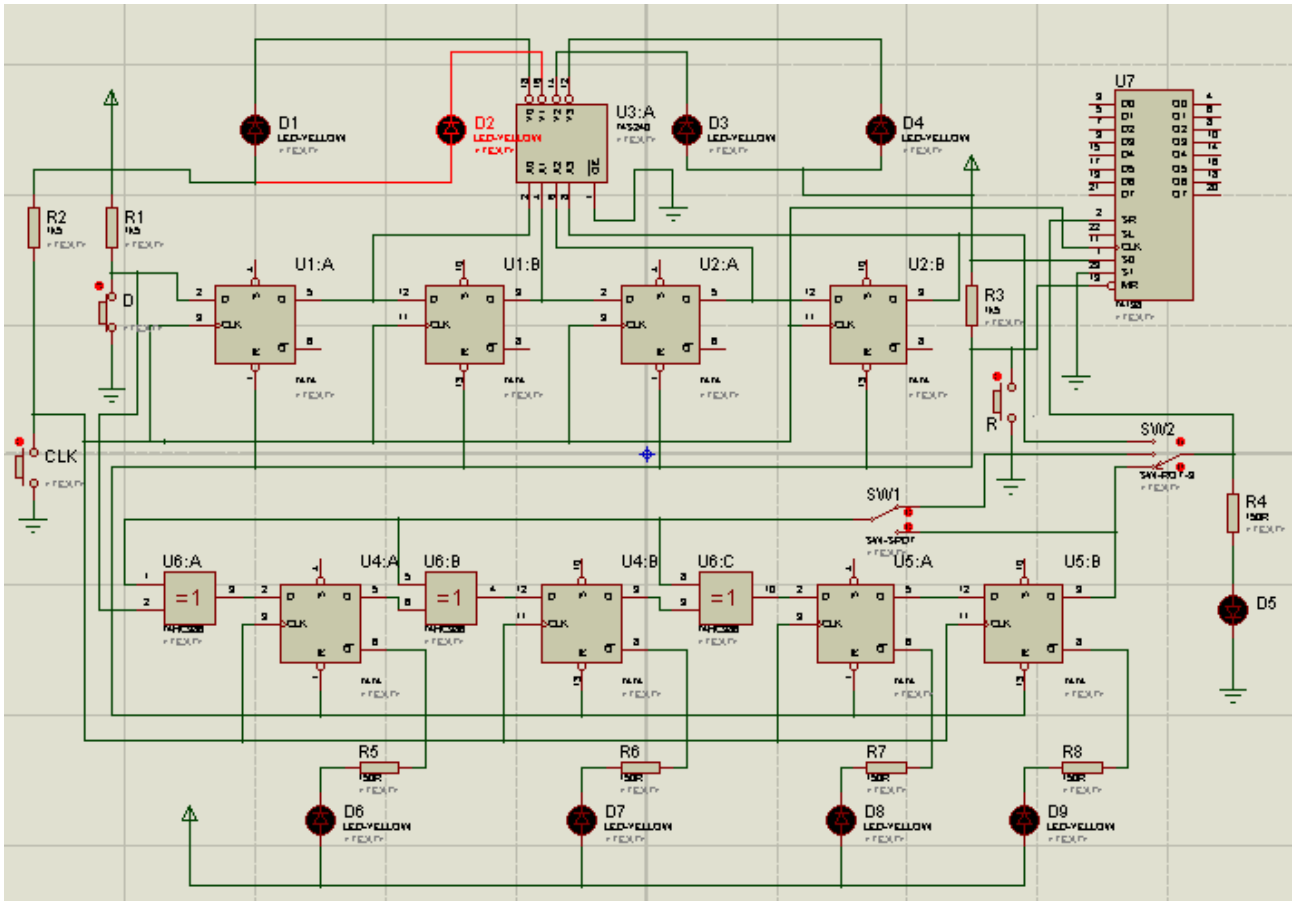


Рисунок 3.1 – Схема модели циклического кодера

Основным узлом устройства является блок деления информационной последовательности на образующий полином, который реализован на основе D-

триггеров U4:A…U5:B (микросхема 7474, содержащая 2 триггера в одном корпусе) и сумматоров по модулю 2 U6:A…U6C (микросхема 74HC386). Эта микросхема содержит четыре двухвходовых сумматоров по модулю 2. Для индикации состояния триггеров делителя используются светодиоды D6-D9 типа LED YELLOV, подключенные через ограничительные резисторы сопротивлением 150 Ом к выходам триггеров.

Регистр сдвига информационных элементов построен на аналогичных триггерах U1:A-U2:B. Для усиления сигналов управления светодиодными индикаторами D1-D4 используются буферные усилители с тремя состояниями, содержащиеся в микросхеме U3:A типа 74S240. Выходные информационные и проверочные биты последовательно выдаются на выход R4-D5 и одновременно заносятся в последовательно-параллельный регистр U7 в качестве которого используется микросхема типа 74198.

Побитная подача входных данных осуществляется путем замыкания (0) или размыкания (1) кнопки D, запись которых в регистр сдвига происходит при нажатии и отпускании кнопки CLK.

Схема модели декодера циклического кода показана на рисунке 3.2.

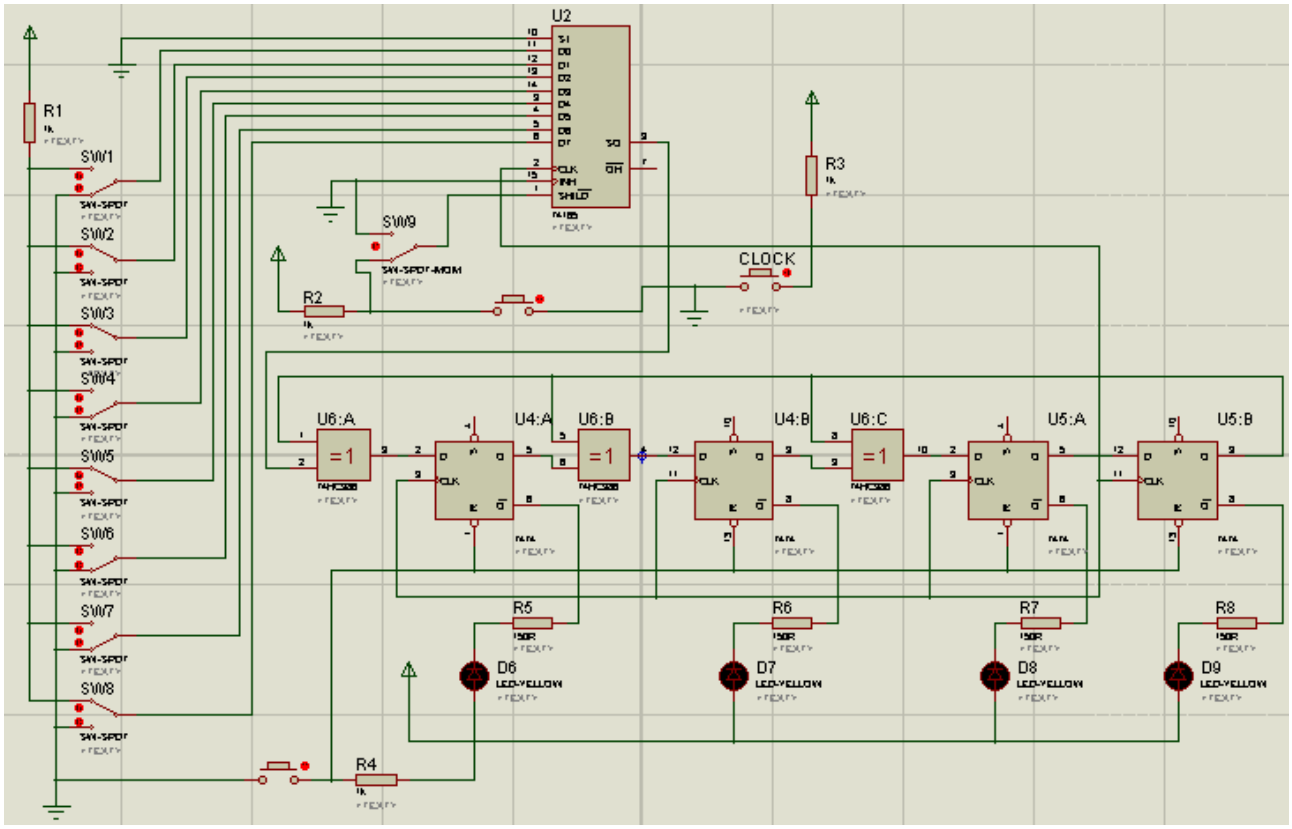


Рисунок 3.2 – Схема модели циклического декодера

Декодер состоит из следующих функциональных модулей:

– параллельно-последовательного регистра U2 (микросхема 74165);

– делителя входной последовательности на образующий полином (элементы U4 – U6). Сумматоры по модулю 2 реализованы микросхемой 74HC386, а триггеры регистра сдвига микросхемой 7474).

Переключатель SW9 переводит регистр из режима параллельной загрузки декодируемой последовательности в режим побитной выдачи на делитель ее на образующий полином.

Назначение остальных элементов понятно из самой схемы декодера.

4 ПРОГРАММА ЛАБОРАТОРНОЙ РАБОТЫ

4.7 Изучить по рекомендуемой литературе теоретический материал по теме циклического кодирования. Выполняется в процессе домашней подготовки.

4.8 Рассчитать на основании выражения (2.1) значения проверочных бит для всех возможных 16-ти значений информационных комбинаций.

4.9 Составить в рабочем поле эмулятора схему кодера, изображенную на рисунке 3.1.

4.10 Ввести побитно с помощью ключей D и CLK в регистры сдвигов информационных элементов U1-U2 и делителя на образующий полином U4-U5 одну из 16-ти информационных кодовых комбинаций и проследить индикацию ее светодиодами.

4.11 Сравнить полученные экспериментально проверочные комбинации с комбинациями, вычисленными теоретически.

4.12 Повторить пп.4.4-4.5 для всех 16 информационных комбинаций.

4.13 Перевести переключатели SW1 и SW2 для задания режимов вывода информационных элементов, деления их на образующий полином и вывести закодированную последовательность на выход кодера и сохранить ее в регистре сдвига U7.

4.14 Записать все 16 закодированных последовательностей в рабочую тетрадь.

4.15 Составить в рабочем окне симулятора схему циклического декодера, изображенную на рисунке 3.2.

4.16 Задавая по очереди с помощью переключателей SW1-SW8 в параллельном виде сохраненные в рабочей тетради кодовые комбинации, полученные при исследовании кодера, выводить их побитно путем подачи тактовых сигналов с помощью кнопки CLOCK, проверить правильность декодирования сообщения при отсутствии ошибок.

4.17 Изменяя поочередно от одного до 4 битов во входной комбинации, имитируя одиночную или групповую ошибки, исследовать обнаружение их декодером.

4.18 Сформулировать выводы по работе и оформить отчет.

Примечание. Демоверсия обладает полным функционалом, за исключением возможности сохранения проекта. Поэтому перед началом выполнения

работы необходимо сначала составить схемы исследуемых устройств в бумажном виде, а в процессе выполнения работы в лаборатории составить ее в рабочем окне программы моделирования и исследовать в соответствии с программой исследования.

5 СОДЕРЖАНИЕ ОТЧЕТА

5.7 Титульный лист.
5.8 Цель и программа лабораторной работы.
5.9 Кодовые комбинации, рассчитанные теоретически и получены при кодировании циклическим кодом.
5.10 Электрические принципиальные схемы кодера и декодера.
5.11 Выводы по результатам исследований.

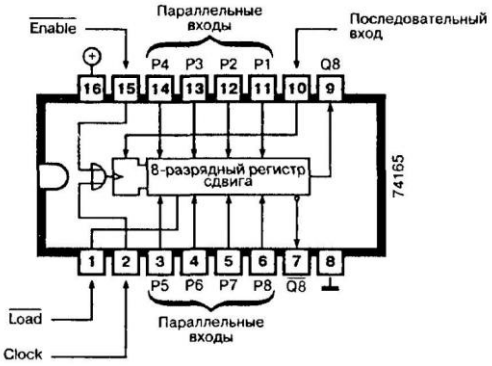
6 КОНТРОЛЬНЫЕ ВОПРОСЫ

6.1 Какие показатели используются для оценки помехоустойчивых кодов?
6.2 Как определяется кодовое расстояние кода и какие соотношения должны выполняться при необходимости обнаружения и исправления ошибок?
6.3 С какой целью используются циклические коды и как формируются проверочные элементы?
6.4 Продемонстрируйте на практике образование проверочных комбинаций.
6.5 Поясните по принципиальной схеме, как вычисляются проверочные комбинации.
6.6 Поясните по принципиальной схеме декодера, как обнаруживаются одиночные и групповые ошибки.
6.7 Чем отличаются параллельные регистры от последовательных, последовательно-параллельные от параллельно-последовательных?
6.8 Зачем последовательно со светодиодами включаются резисторы и как выбирается их номинал?
6.9 В чем состоит отличие блочных кодов от непрерывных?
6.10 Поясните, каким образом исправляются ошибки в итеративном коде.

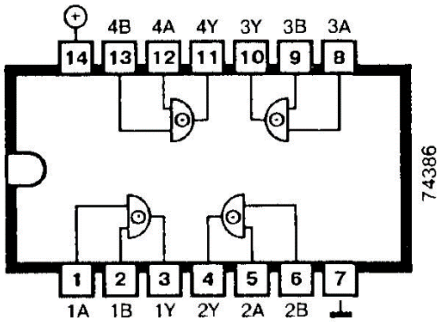
БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Вернер М. Основы кодирования: Учебник для вузов / М. Вернер. — М.: Техносфера, 2006. — 288 с.
2. Матвеев Б. Основы корректирующего кодирования. Теория и лабораторный практикум. Лань, 2014. – 192 с.
3. Осокин А.Н. Теория информации: Учебное пособие для прикладного бакалавриата / А.Н. Осокин, А.Н.Мальчуков. – М.: Изд-во Юрайт, 2019. – 205 с. https://biblio-online.ru/viewer/teoriya-informacii-434040?share _image_id=#page/2
4. Чернега В.С. Расчет и проектирование технических средств обмена и передачи информации / В.С. Чернега, В.А. Василенко, В.Н. Бондарев. — М.: Изд-во «Высшая школа», 1990. — 224 с.
5. Чернега В.С. Сжатие информации в компьютерных сетях. — Севастополь: Изд-во СевГТУ. 1997. — 214 с.
6. Щеглов А.Ю. Защита информации: основы теории: учебник для бакалавриата и магистратуры / А. Ю. Щеглов, К. А. Щеглов. — М. : Издательство Юрайт, 2017. — 309 с. https://urait.ru/catalog/scientific school/ED09AF94-1A5F-4A5D-86E5-EEA1A2E635B6

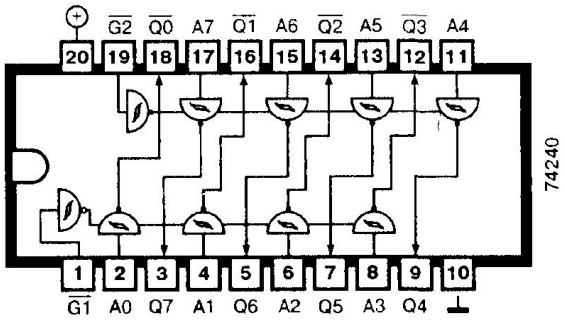
ПРИЛОЖЕНИЕ. Цоколевки используемых в работе микросхем



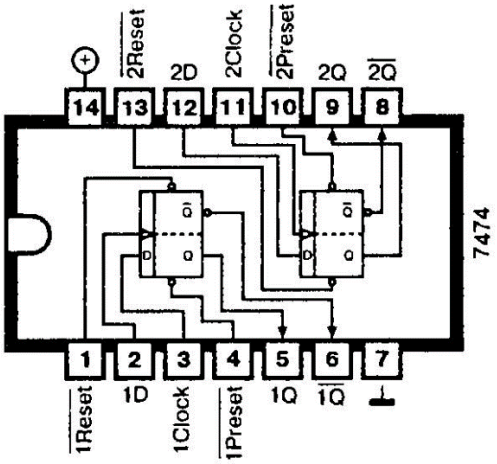
Параллельно-последовательный регистр



Сумматоры по модулю 2



Буферные усилители



Два D-триггера