Федеральное агенство по образованию
Государственное образовательное учреждение высшего
профессионального образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Кафедра «Информационная безопасность систем и технологий»

ПОМЕХОУСТОЙЧИВОЕ КОДИРОВАНИЕ

Пояснительная записка к курсовой работе
по дисциплине «Передача дискретных сообщений»

ПГУ 3.090106.001 ПЗ

Руководитель КР,
д.т.н., профессор _____ Б. В. Султанов

Исполнитель КР,
студент _____ М. А. Захаров

Пенза 2009

«УТВЕРЖДАЮ»
Зав. кафедрой ИБСТ
_____С. Л. Зефиров
«____» _____ 2009 г.

ЗАДАНИЕ
на курсовую работу

по теме: «Помехоустойчивое кодирование»
1 Дисциплина Передача дискретных сообщений
2 Вариант задания 7
3 Студент Захаров М. А. группа 06УИ1
4 Исходные данные на курсовую работу

- код Хемминга. Задача 1.1.7;
- код БЧХ. Задача 2.3.7;
- код Рида—Соломона. Задача 3.4.7.

5 Структура работы
5.1 Пояснительная записка (содержание работы):

- расчётно-пояснительная записка объёмом 15–20 страниц содержит 3 раздела (в соответствии с пунктами задания).

5.2 Графическая часть

- схема регистра кодирующего устройства;
- таблица состояний ячеек регистра (1 лист формата А4).

5.3 Экспериментальная часть

- не предусмотрена.

6 Календарный план выполнения работы
6.1 Сроки выполнения работ по разделам:

- раздел первый к 25.09.2009 г.

– раздел второй к 20.10.2009 г.

– раздел третий к 25.11.2009 г.

– оформление пояснительной записки к 06.12.2009 г.

Дата защиты работы 6 декабря 2009 г.

Руководитель работы Султанов Б. В.

Задание получил 7 сентября 2009 г.

Студент Захаров М. А.

Нормоконтролёр Султанов Б. В.

РЕФЕРАТ

Пояснительная записка 48 с., 1 рис., 4 табл., 2 источника, 3 прил.

ПОМЕХОУСТОЙЧИВОЕ КОДИРОВАНИЕ, КОД ХЕММИНГА, КОД БОУЗА—ЧОУДХУРИ—ХОКВИНГЕМА, КОД РИДА—СОЛОМОНА, OCTAVE

Объектом исследования являются помехоустойчивые коды.

Цель работы — решение задач по синтезу следующих помехоустойчивых кодов: код Хемминга, код Боуза—Чоудхури—Хоквингема и код Рида—Соломона.

В процессе выполнения курсовой работы были рассчитаны помехоустойчивые коды в соответствии с заданием. Для расчётов в полях $GF(q)$ была использована программа *Octave*.

В результате исследования были получены навыки по расчёту помехоустойчивых кодов с заданными характеристиками.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Высокие требования к достоверности передачи информации в современных телекоммуникационных системах диктуют необходимость разработки и совершенствования методов кодирования дискретных сообщений, обеспечивающих обнаружение и исправление ошибок, возникающих в канале связи. Принципиальная возможность решения этой задачи была обоснована американским учёным Клодом Шенноном более пятидесяти лет тому назад, который однако не указал конкретных способов построения таких кодов. Поэтому с тех пор и по настоящее время интенсивно развивается теория помехоустойчивого кодирования, предметом которой является поиск алгоритмов кодирования и декодирования, обеспечивающих максимальную корректирующую способность в каналах с различными свойствами при минимально возможной избыточности и реализационных затратах [1].

Данная курсовая работа посвящена исследованию корректирующих кодов — кодов, служащих для обнаружения или исправления ошибок, возникающих при передаче информации под влиянием помех, а также при её хранении.

Курсовая работа состоит из 3 разделов.

В первом разделе исследуется код Хемминга, для него найден порождающий многочлен, сформирована разрешённая кодовая комбинация кода, в искажённом сообщении исправлена ошибка.

Во втором разделе рассматривается код БЧХ, для него найден порождающий многочлен, сформирована разрешённая кодовая комбинация, построен регистр кодирующего устройства для кода БЧХ.

Третий раздел посвящён коду Рида—Соломона, построены таблицы представления, сложения и умножения элементов в поле $GF(16)$, найден порождающий многочлен, определены синдромы для принятой комбинации. В дополнительной задаче в ошибочно принятой комбинации найдены и исправлены три ошибки.

Для вычислений в конечных полях использовался язык высо-

кого уровня *GNU Octave*, предназначенный, в основном, для численных расчетов, и по сути являющийся альтернативой коммерческому *MatLab*. Пакет может работать в режиме сценариев, интерактивно или посредством привязки к языку *C/C++*. В *Octave* реализован богатый язык программирования, обладающий очень большой библиотекой математических функций, в том числе специализированных функций обработки сигналов, изображений, звука и т. п.

В приложениях приведены регистр кодирующего устройства, таблица, иллюстрирующую состояние ячеек в процессе работы регистра, программа для нахождения этих состояний, расчёты в программе *GNU Octave*.

1 КОД ХЕММИНГА

1.1 Условие задачи

Определить порождающий многочлен $g(x)$ кода Хемминга, скорость которого $R \geqslant r_0$, рассматривая его как код БЧХ, исправляющий одиночные ошибки. Сформировать разрешённую комбинацию систематического кода, соответствующую заданной информационной комбинации $a(x) = 10011000111$. Исправить ошибку в принимаемой кодовой комбинации $V'(x) = 111110001000010$.

1.2 Решение задачи

В данном случае код Хемминга имеет размерность (15,11), т. к. по условию скорость $R \geqslant r_0$. При $k = 11$ и $n = 15$, получаем, что $R = \frac{k}{n} = 0{,}73$, что превышает значение r_0, равное 0,7.

Так как код Хемминга предложено рассматривать как код БЧХ, то первым этапом при расчёте кода будет определение порождающего многочлена $g(x)$. Порождающий многочлен для кода БЧХ определяется из выражения:

$$g(x) = \text{НОК}\left[f_1(x), f_2(x) \ldots f_{2t_u}\right],$$

где НОК — наименьшее общее кратное;

$f_1(x), f_2(x), \ldots$ — минимальные многочлены корней $\alpha^1, \alpha^2 \ldots$ порождающего многочлена.

Корнем многочлена $g(x)$ называется число (элемент поля), при подстановке которого в выражение многочлена вместо x, многочлен обращается в 0. Минимальный многочлен элемента β поля $GF(q^m)$ определяется из выражения:

$$f(x) = (x - \beta^{g^0})(x - \beta^{g^1}) \dots (x - \beta^{g^{l-1}}),$$

где l — наименьшее целое число, при котором:

$$\beta^{g^0} = \beta^{g^l}.$$

На практике для определения значения порождающего многочлена можно воспользоваться таблицей минимальных неприводимых многочленов в поле $GF(2^m)$ [2].

Для определения порождающего многочлена необходимо, во-первых, по заданной длине кода n определить из выражения $n = 2^m - 1$ значение параметра m, который является степенью сомножителя $g(x)$. Затем из выражения $j = 2t_u - 1$ определяем максимальный порядок минимального многочлена, входящих в число сомножителей $g(x)$. После этого, пользуясь таблицей минимальных многочленов, определяем выражение для $g(x)$, зависимости от найденных m и j. Для этого из колонки соответствующей параметру m выбираются многочлены с номерами от 1 до j, которые в результате перемножения дают выражение для $g(x)$. В нашем случае $n = 15$, а $t_u = 1$, следовательно пользуясь описанной выше методикой, получаем $m = 4$, $j = 1$, откуда получаем $g(x) = 010011$, или в виде многочлена $g(x) = x^4 + x + 1$.

Следующим этапом построения является построение производящей матрицы $G_{(n,k)}$. Для систематического кода матрица $G_{(n,k)}$ имеет вид:

$$G_{(n,k)} = \left[I_k R_{(k,r)} \right],$$

где I_k — единичная матрица размером $k \times k$.

Строки матрицы $R_{(k,r)}$ определяются из выражений:

$$r_i(x) = R_{g(x)}\left(a_i(x) \times x^r\right), \qquad (1)$$

или

$$r_i(x) = R_{g(x)}\left(x^{n-i}\right),$$

где $a_i(x)$ — полином, соответствующий i-той строке матрицы I_k;

i — номер строки матрицы $R_{(k,r)}$;

$R_{g(x)}\big(a(x)\big)$ — остаток от деления $a(x)$ на $g(x)$.

Выполнив деление многочленов, согласно формуле (1), получаем:

$$R_{(15,11)} = \left[\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ \hline 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{array}\right].$$

Для нахождения остатков от деления использовалась программа *Octave* (см. Приложение Б, стр. 30)

В данном случае для кода (15,11) матрица I_k,будет иметь следующий вид:

$$I_{11} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Таким образом, производящая матрица будет иметь следующий вид:

$$G_{(15,11)} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}.$$

Для получения кодовой комбинации необходимо вектор, соответствующий кодовой комбинации $a(x)$, умножить на матрицу $G_{(15,11)}$. Полученный в результате умножения вектор и будет являться разрешённой кодовой комбинацией. В соответствии с заданием $a(x) = 10011000111$. Выполним умножение:

$$V(x) = a(x) \times G_{(15,11)} = 10011000111 \times$$

$$\times \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} =$$

$$= 100110001111001.$$

Проверочная матрица в систематическом виде строится на основе матрицы $G_{(n,k)}$, а именно:

$$H_{(n,k)} = \left[R^T_{(k,r)} I_r\right],$$

где I_r — единичная матрица;

$R^T_{(k,r)}$ — матрица $R_{(k,r)}$ из $G_{(n,k)}$ в транспонированном виде.

В соответствии с матрицей $G_{(15,4)}$ получаем:

$$H_{(15,11)} = \begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Для определения синдрома необходимо умножить полученную кодовую комбинацию на $H^T_{(n,r)}$. Полученный в результате умно-

жения вектор и будет являться синдромом, по которому можно судить о наличии и расположении ошибки, или её отсутствии. Принятая кодовая комбинация $V'(x) = 111110001000010$. Выполним умножение:

$$S(x) = V'(x) \times H^T_{(n,r)} = 111110001000010 \times \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = 1100.$$

В соответствии с этим синдромом определяем по матрице $H_{(15,4)}$, что ошибка произошла в 9 разряде, следовательно, исправленная комбинация будет 111110000000010.

1.3 Ответ

В результате решения были найдены образующий полином $g(x) = x^4+x+1$, разрешённая кодовая комбинация, соответствующая информационной комбинации a(x) — V(x) = 100110001111001, была

определена ошибка в кодовой комбинации $V'(x) = 111110001000010$, после исправления была получена исправленная кодовая комбинация — 111110000000010.

2 КОД БОУЗА—ЧОУДХУРИ—ХОКВИНГЕМА

2.1 Условие задачи

Определить порождающий многочлен $g(x)$ примитивного кода БЧХ над $GF(2)$ длины $n = 2^m - 1$, исправляющего ошибки кратностью $t_и = 2$. Сформировать разрешённую комбинацию систематического кода, соответствующую заданной информационной комбинации:

$$a(x) = 10011100001111000000.$$

Построить регистр кодирующего устройства систематического циклического кода с порождающим многочленом $g(x)$, привести таблицу, иллюстрирующую состояние ячеек в процессе работы регистра при поступлении на его вход информационного блока $a(x)$. Определить, является ли разрешённой принимаемая кодовая комбинация:

$$V'(x) = 1101111011011011000110001010000.$$

2.2 Решение задачи

Первым этапом решения задачи по синтезу кода БЧХ является определение порождающего многочлена $g(x)$. Для этого необходимо воспользоваться методикой, описанной выше при решении задачи синтеза кода Хемминга. По условию задачи кратность исправляемых ошибок $t_и = 2$, следовательно в соответствии с формулой $j = 2t_и - 1$, получаем $j = 3$. Длина кода $n = 31$ и $m = 5$. Таким

образом, полином $g(x)$ будет равен произведению полиномов записанных в первой, второй и третьей строках 4-го столбца таблицы минимальных многочленов, т. е.

$$g(x) = 45 \times 75 = (x^5 + x^2 + 1)(x^5 + x^4 + x^3 + x^2 + 1) = \\ = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1.$$

В двоичном виде $g(x) = 11101101001$.

После нахождения порождающего многочлена, необходимо сформировать разрешённую кодовую комбинацию систематического кода, соответствующую заданной информационной последовательности, по следующей формуле:

$$V(x) = a(x) \times x^r + r(x), \tag{2}$$

где r — количество проверочных разрядов;

$r(x)$ — остаток от деления $a(x) \times x^r$ на $g(x)$.

Таким образом, первые $n - r$ разрядов будут совпадать с информационной последовательностью, а последние r разрядов будут проверочными. По условию задачи $a(x) = 10011100001111000000$, а полиному x^{10} соответствует последовательность 10000000000, таким образом, получаем:

$$a(x) \cdot x^r = 10011100001111000000000000000000.$$

Остаток от деления $r(x)$ найден в программе *Octave* (см. Приложение Б, стр. 35):

$$r(x) = 1111011111.$$

Таким образом, разрешённая кодовая комбинация по формуле (2):

$$V(x) = 10011100001111100000011111011111.$$

Следующим этапом решения задачи является построение регистра кодирующего устройства и приведение таблицы переключений состояний ячеек регистра при поступлении на вход информационной последовательности $a(x)$. Схема регистра кодирующего устройства приведена в приложении А на рисунке А.1, а таблице переключений соответствует таблица А.1.

После поступления на вход регистра информационно блока $a(x) = 10011100001111100000$ в его ячейках формируется проверочная последовательность $r(x) = 1111011111$, которая соответствует остатку от деления $a(x) \cdot x^r$ на $g(x)$.

По условию задачи так же необходимо определить, является ли разрешённой кодовая комбинация:

$$V'(x) = 11011110110110110001100010100000.$$

Для этого необходимо найти синдром $S(x)$, т.е. произвести деление многочлена $V'(x)$ на порождающий многочлен $g(x)$. Если остаток от деления будет равен нулю, то комбинация является разрешённой. В противном случае — неразрешённой. Операция производилась в программе *Octave*.

В результате выполнения деления, остаток равен 0, следовательно, принятая комбинация является разрешённой.

2.3 Ответ

В процессе решения задачи был определён порождающий многочлен:

$$g(x) = x^{10} + x^9 + x^8 + x^6 + x^5 + x^3 + 1;$$

сформирована разрешённая кодовая комбинация, соответствующая информационной последовательности $a(x)$:

$$V(x) = 1001110000111110000001111011111;$$

построен регистр кодирующего устройства и построена таблица состояний ячеек регистра при поступлении на его вход информационной последовательности $a(x)$. Так же было определено, что принятая кодовая комбинация $V'(x)$ является разрешённой.

3 КОД РИДА—СОЛОМОНА

3.1 Условие задачи

Построить таблицы представления, сложения и умножения элементов в поле $GF(q)$, $q = 2^4$. Определить порождающий многочлен кода Рида—Соломона над полем, исходя из условия, что код должен исправлять $t_u = 3$ неправильно принятых q-ичных символов. Сформировать разрешённую кодовую комбинацию систематического кода, соответствующую заданной информационной комбинации:

$$a(x) = 000000000100000001011010000100001001.$$

Вычислить синдром и определить, является ли разрешённой принимаемая кодовая комбинация:

$$V'(x) = 011111000110011001100110110100000100111101000001001010000101.$$

3.2 Решение задачи

Первым этапом решения задачи является построение таблицы представлений поля $GF(16)$, построенного на основе многочлена $p(z) = z^4 + z + 1$ с примитивным элементом z. Результат представлен в таблице 1.

Многочлен, представляющий элемент поля в графе «многочленное представление», может быть получен как остаток от де-

Таблица 1 – Таблица представлений поля $GF(16)$

Степенное представление	Многочленное представление	Кодовое представление	Десятичное представление
α^0	1	0001	1
α^1	z	0010	2
α^2	z^2	0100	4
α^3	z^3	1000	8
α^4	$z+1$	0011	3
α^5	z^2+z	0110	6
α^6	z^3+z^2	1100	12
α^7	z^3+z+1	1011	11
α^8	z^2+1	0101	5
α^9	z^3+z	1010	10
α^{10}	z^2+z+1	0111	7
α^{11}	z^3+z^2+z	1110	14
α^{12}	z^3+z^2+z+1	1111	15
α^{13}	z^3+z^2+1	1101	13
α^{14}	z^3+1	1001	9

ления элементов степенного представления на образующий многочлен $p(z)$, при этом элемент степенного представления α^i записывается как z^i.

Таблица 2 – Сложение в поле GF(16)

$+$	0	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
0	0	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
1	1	0	α^4	α^8	α^{14}	α^1	α^{10}	α^{13}	α^9	α^2	α^7	α^5	α^{12}	α^{11}	α^6	α^3
α^1	α^1	α^4	0	α^5	α^9	1	α^2	α^{11}	α^{14}	α^{10}	α^3	α^8	α^6	α^{13}	α^{12}	α^7
α^2	α^2	α^8	α^5	0	α^6	α^{10}	α^1	α^3	α^{12}	1	α^{11}	α^4	α^9	α^7	α^{14}	α^{13}
α^3	α^3	α^{14}	α^9	α^6	0	α^7	α^{11}	α^2	α^4	α^{13}	α^1	α^{12}	α^5	α^{10}	α^8	1
α^4	α^4	α^1	1	α^{10}	α^7	0	α^8	α^{12}	α^3	α^5	α^{14}	α^2	α^{13}	α^6	α^{11}	α^9
α^5	α^5	α^{10}	α^2	α^1	α^{11}	α^8	0	α^9	α^{13}	α^4	α^6	1	α^3	α^{14}	α^7	α^{12}
α^6	α^6	α^{13}	α^{11}	α^3	α^2	α^{12}	α^9	0	α^{10}	α^{14}	α^5	α^7	α^1	α^4	1	α^8
α^7	α^7	α^9	α^{14}	α^{12}	α^4	α^3	α^{13}	α^{10}	0	α^{11}	1	α^6	α^8	α^2	α^5	α^1
α^8	α^8	α^2	α^{10}	1	α^{13}	α^5	α^4	α^{14}	α^{11}	0	α^{12}	α^1	α^7	α^9	α^3	α^6
α^9	α^9	α^7	α^3	α^{11}	α^1	α^4	α^6	α^5	1	α^{12}	0	α^{13}	α^2	α^8	α^{10}	α^4
α^{10}	α^{10}	α^5	α^8	α^4	α^{12}	α^2	1	α^7	α^6	α^1	α^{13}	0	α^{14}	α^3	α^9	α^{11}
α^{11}	α^{11}	α^{12}	α^6	α^9	α^5	α^{13}	α^3	α^1	α^8	α^7	α^2	α^{14}	0	1	α^4	α^{10}
α^{12}	α^{12}	α^{11}	α^{13}	α^7	α^{10}	α^6	α^{14}	α^4	α^2	α^9	α^8	α^3	1	0	α^1	α^5
α^{13}	α^{13}	α^6	α^{12}	α^{14}	α^8	α^{11}	α^7	1	α^5	α^3	α^{10}	α^9	α^4	α^1	0	α^2
α^{14}	α^{14}	α^3	α^7	α^{13}	1	α^9	α^{12}	α^8	α^1	α^6	α^6	α^{11}	α^{10}	α^5	α^2	0

После того, как определены таблицы сложения и умножения в поле $GF(16)$ можно приступать к синтезу кода. Первым эта-

Таблица 3 – Умножение в поле $GF(16)$

$\times$	0	1	α^1	α^2	α^3	α^4	5	α^6	α^7	α^8	α^9	10	α^{11}	α^{12}	α^{13}	α^{14}
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}
α^1	0	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1
α^2	0	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1
α^3	0	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2
α^4	0	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3
α^5	0	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4
α^6	0	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4	α^5
α^7	0	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4	α^5	α^6
α^8	0	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7
α^9	0	α^9	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8
α^{10}	0	α^{10}	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9
α^{11}	0	α^{11}	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}
α^{12}	0	α^{12}	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}
α^{13}	0	α^{13}	α^{14}	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}
α^{14}	0	α^{14}	1	α^1	α^2	α^3	α^4	α^5	α^6	α^7	α^8	α^9	α^{10}	α^{11}	α^{12}	α^{13}

пом синтеза является определение порождающего многочлена $g(x)$. Для кода Рида—Соломона порождающий многочлен определяется по формуле:

$$g(x) = \prod_{i=\upsilon}^{i=2t_u+\upsilon-1} (x - \alpha^i),$$

где t_u — кратность исправляемых ошибок;

υ — чаще всего принимается равным 1.

Таким образом:

$$g(x) = \prod_{i=1}^{6}(x - \alpha^i) =$$

$$= (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6) =$$

$$= (x - 2)(x - 4)(x - 8)(x - 3)(x - 6)(x - 12) =$$

$$= x^6 + x^5\alpha^{10} + x^4\alpha^{14} + x^3\alpha^4 + x^2\alpha^6 + x\alpha^9 + \alpha^6.$$

После нахождения порождающего многочлена, необходимо сформировать разрешённую кодовую комбинацию систематического кода, соответствующую заданной информационной комбинации $a(x)$, по формуле:

$$V(x) = a(x)x^r + r(x),$$

где r — количество проверочных разрядов;

$r(x)$ — остаток от деления $a(x)x^r$ на $g(x)$.

$$N = 2^m - 1 = 15;$$
$$d = 2t_u + 1 = 7;$$
$$k = N - d + 1 = 9;$$
$$r = N - k = 6.$$

Таким образом,

$$r(x) = R_{g(x)}\left[a(x)x^6\right].$$

Все дальнейшие операции производились в программе *Octave* (см. Приложение Б, стр. 37).

Умножая $a(x)$ на x^6 и находя остаток от деления $r(x)$, получаем кодовую комбинацию в десятичном представлении $V(x) = 0\ 0\ 4\ 0\ 5\ 10\ 1\ 0\ 9\ 9\ 10\ 13\ 4\ 15\ 8$. Или в двоичном виде:

$$V(x) = [000000000100000001011010000100001001100110101101010011111000].$$

Синдром для кода Рида—Соломона определяется по формуле:

$$S = S_{d-2} \ldots S_1 S_0,$$
$$S_J = \sum_{i=0}^{N-1} V_i \cdot z_i^{m_0+j} \ (j = 0 \ldots d-2)$$

где $m_0 = 1$;

V_i — символы коэффициентов полинома, представляющего кодовую комбинацию;

$z_i = \alpha^i$ — локаторы, где α — примитивный элемент поля $GF(q)$.

Таким образом, синдромы:

$$S_0 = \sum_{i=0}^{14} V_i z_i^1 = V_0 + V_1 z_1 + V_2 z_2 + V_3 z_3 + V_4 z_4 + V_5 z_5 + V_6 z_6 + V_7 z_7 +$$
$$+ V_8 z_8 + V_9 z_9 + V_{10} z_{10} + V_{11} z_{11} + V_{12} z_{12} + V_{13} z_{13} + V_{14} z_{14} =$$
$$= \alpha^8 + \alpha^4 + \alpha^3 + \alpha^3 + \alpha^6 + \alpha^2 + \alpha^8 + \alpha^6 + \alpha^{14} + 1 + \alpha +$$
$$+ \alpha^2 + \alpha^4 + \alpha^9 = 0;$$

Остальные синдромы $S_1 - S_5$ в соответствии с описанной методикой вычислялись в программе *Octave* и также равны 0. В результате вычислений синдром $S = 000000$, т. е. принятая кодовая комбинация не содержит ошибок.

3.3 Ответ

В результате решения задачи были построены таблицы представлений, сложения и умножения в $GF(16)$, на основе примитивного многочлена $p(z) = z^4 + z + 1$ с примитивным элементом z. Была определена кодовая комбинация, соответствующая информационной комбинации $a(x)$:

$$V(x) = [000000000100000001011010000100001001100110101101010011111000].$$

Также было определено, что принятая комбинация $V'(x)$, является разрешённой кодовой комбинацией.

3.4 Дополнительная задача

Найти расположение ошибок и их значения в принятой кодовой комбинации:

$$V'(x) = 3\ 8\ 2\ 0\ 5\ 0\ 11\ 13\ 4\ 3\ 9\ 9\ 4\ 3\ 1.$$

в поле $GF(q)$, $q = 2^4$, $t_u = 3$.

На практике известна только принятая кодовая комбинация $V'(x)$, однако также известно, что если полином ошибок имеет вид:

$$e(x) = e_{j1}x^{j1} + e_{j2}x^{j} + \cdots + e_{j\nu}x^{j\nu}, \tag{3}$$

где индексы $1, 2, 3, \ldots, \nu$ обозначают номер ошибки;

$j_1, j_2, \ldots, j_\nu$ — характер расположения ошибки с этим номером, то $2t_u$ значения синдрома, определенным показанным выше способом позволяют записать систему уравнений:

$$\begin{gathered} S_1 = e_{j1}\beta_1 + e_{j2}\beta_2 + \ldots + e_{j\nu}\beta_\nu; \\ S_2 = e_{j1}\beta_1^2 + e_{j2}\beta_2^2 + \ldots + e_{j\nu}\beta_\nu^2; \\ \ldots \\ S_{2tu} = e_{j1}\beta_1^{2t_u} + e_{j2}\beta_2^{2t_u} + \ldots + e_{j\nu}\beta_\nu^{2t_u}. \end{gathered} \tag{4}$$

Система содержит $2t_u$ уравнений относительно $2t_u$ неизвестных t_u расположений ошибок и t_u значений ошибок, характеризуемых значением j.

Однако эта система является нелинейной, т. к. β_i входит в уравнения в различных степенях.

Общего метода решения таких систем не существует. В данной задаче рассмотрена методика, называемая алгоритмом декодирования Рида—Соломона.

Введём в рассмотрение полином локатора ошибок:

$$\sigma(x) = (1 + \beta_1 x)(1 + \beta_2 x) \ldots (1 + \beta_\nu x) = \\ = 1 + \sigma_1 x + \sigma_2 x^2 + \ldots + \sigma_\nu x^\nu. \quad (5)$$

Корнем $\sigma(x)$ будут:

$$\frac{1}{\beta_1}, \quad \frac{1}{\beta_2}, \quad \ldots, \quad \frac{1}{\beta_i}.$$

Величины β_i $(i = \overline{1, \nu})$ представляют номера расположения ошибок в полиноме $e(x)$.

Определение коэффициентов $\sigma_1 \ldots \sigma_2$ полинома (5) осуществляется на основе т. н. авторегрессионной методики моделирования, в соответствии с которой составляется матричное уравнение, связывающее первые $2t_u - 1$ значений синдрома со следующим $2t_u$-м значением.

В общем виде это уравнение выглядит так:

$$\begin{bmatrix} S_1 & S_2 & S_3 & \cdots & S_{t_u-1} & S_{t_u} \\ S_2 & S_3 & S_4 & \cdots & S_{t_u} & S_{t_u+1} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ S_{t_u-1} & S_{t_u} & S_{t_u+1} & \cdots & S_{t_u-3} & S_{t_u-2} \\ S_{t_u} & S_{t_u+1} & S_{t_u+2} & \cdots & S_{t_u-2} & S_{t_u-1} \end{bmatrix} \begin{bmatrix} \sigma_{t_u} \\ \sigma_{t_u-1} \\ \vdots \\ \sigma_2 \\ \sigma_2 \end{bmatrix} = \begin{bmatrix} -S_{2t_u+1} \\ -S_{2t_u+2} \\ \vdots \\ -S_{2t_u-1} \\ -S_{2t_u} \end{bmatrix}. \quad (6)$$

Поскольку для элементов конечных полей справедливо равенство $-S_i = S_i$, знаком «$-$» в правой части можно пренебречь.

На основании (6):

$$\begin{bmatrix} S_1 & S_2 & S_3 \\ S_2 & S_3 & S_4 \\ S_3 & S_4 & S_5 \end{bmatrix} \begin{bmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} S_4 \\ S_5 \\ S_6 \end{bmatrix}. \quad (7)$$

Значения синдромов были вычислены в программе *Octave* (см. Приложение Б, стр. 41), с учётом их значений:

$$\begin{bmatrix} 15 & 1 & 3 \\ 1 & 3 & 1 \\ 3 & 1 & 12 \end{bmatrix} \begin{bmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{bmatrix} = \begin{bmatrix} 1 \\ 12 \\ 1 \end{bmatrix}. \tag{8}$$

Стандартный метод решения (8) предполагает нахождение обратной матрицы m_s^{-1}.

$$m\_s^{-1} = \begin{bmatrix} 7 & 4 & 11 \\ 4 & 8 & 14 \\ 11 & 14 & 10 \end{bmatrix}.$$

$$\begin{bmatrix} \sigma_3 \\ \sigma_2 \\ \sigma_1 \end{bmatrix} = m\_s^{-1} \begin{bmatrix} 1 \\ 12 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 & 4 & 11 \\ 4 & 8 & 14 \\ 11 & 14 & 10 \end{bmatrix} \begin{bmatrix} 1 \\ 12 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 0 \\ 5 \end{bmatrix}$$

В соответствии с (5) полином локаторов ошибок $\sigma(x)$:

$$\sigma = 1 + 5x + 9x^3. \tag{9}$$

Корни полинома (9):

$$\beta_1 = 6(\alpha^5), \quad \beta_2 = 14(\alpha^{11}), \quad \beta_3 = 13(\alpha^{13}).$$

Поскольку рассматриваемый код является недвоичным, помимо расположения ошибок, нужно найти и их значения. Это можно сделать так: найденные значения β_1, β_2, β_3 позволяют записать значения элементов синдрома S_1, S_2, S_3, определённые из полинома ошибок (3) в виде:

$$\begin{aligned} S_1 &= e(\alpha) = e_1\beta_1 + e_2\beta_2; \\ S_2 &= e(\alpha^2) = e_1\beta_1^2 + e_2\beta_2^2; \\ S_3 &= e(\alpha^3) = e_1\beta_1^3 + e_2\beta_2^3. \end{aligned} \tag{10}$$

Выражение (10) представляет собой линейную систему с 3-мя неизвестными e_1, e_2, e_3, представляющих собой искомое значение ошибок. В рассматриваемом примере систему можно решить с помощью матрицы.

В матричной форме записи выражение (10) принимает вид:

$$\begin{bmatrix} \beta_1 & \beta_2 & \beta_3 \\ \beta_1^2 & \beta_2^2 & \beta_3^2 \\ \beta_1^3 & \beta_2^3 & \beta_3^3 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} S_1 \\ S_2 \\ S_3 \end{bmatrix}.$$

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} 6 & 14 & 13 \\ 6^2 & 14^2 & 13^2 \\ 6^3 & 14^3 & 13^3 \end{bmatrix}^{-1} \begin{bmatrix} 15 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 10 & 3 & 1 \\ 1 & 3 & 15 \\ 11 & 10 & 12 \end{bmatrix} \begin{bmatrix} 15 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 12 \\ 14 \\ 14 \end{bmatrix}.$$

Таким образом,

$$e(x) = 14x^{13} + 14x^{11} + 12x^5 = \alpha^{11}x^{13} + \alpha^{11}x^{11} + \alpha^6 x^5.$$

Складывая найденный полином ошибок $e(x)$ с ошибочно принятой комбинацией $V'(x)$, получаем исправленную кодовую комбинацию:

$$V(x) = 3\ 6\ 2\ 14\ 5\ 0\ 11\ 13\ 4\ 15\ 9\ 9\ 4\ 3\ 1.$$

В ошибочно принятой комбинации исправлены все 3 символьных ошибки.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы были получены навыки по расчёту помехоустойчивых кодов: кода Хеммина, кода БЧХ, РС-кода.

Так же были приобретены навыки по вычислениям в полях $GF(q)$. Все задачи, предложенные в задании, были решены полностью, все решения снабжены необходимыми комментариями. Таким образом, задание на курсовую работу было выполнено в полном объёме.

Приложение А

(обязательное)

Регистр кодирующего устройства

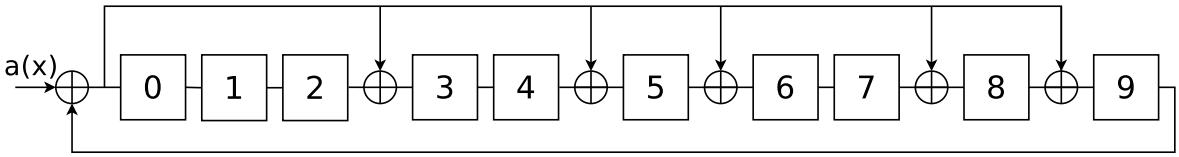

Рисунок А.1 – Схема регистра КДУ

Таблица А.1 – Таблица переключений ячеек регистра

Вход	**Состояние ячеек**									
	0	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	1	1	0	1	1
0	1	1	0	1	1	1	0	1	1	0
0	0	1	1	0	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	0	1
1	0	0	0	1	1	0	1	1	1	0
1	1	0	0	1	1	0	1	1	0	0
0	0	1	0	0	1	1	0	1	1	0
0	0	0	1	0	0	1	1	0	1	1
0	1	0	0	0	0	1	0	1	1	0
0	0	1	0	0	0	0	1	0	1	1
1	0	0	1	0	0	0	0	1	0	1
1	0	0	0	1	0	0	0	0	1	0
1	1	0	0	1	1	1	1	0	1	0
1	1	1	0	1	1	0	0	1	1	0
1	1	1	1	1	1	0	1	0	0	0
0	0	1	1	1	1	1	0	1	0	0
0	0	0	1	1	1	1	1	0	1	0
0	0	0	0	1	1	1	1	1	0	1
0	1	0	0	1	1	0	0	1	0	1
0	1	1	0	1	1	0	1	0	0	1
0	1	1	1	1	1	0	1	1	1	1

Приложение Б
(обязательное)

Вычисления в среде GNU Octave

Нахождение остатка от деления для полинома в 1-м разделе:

```
[maxim@home-dekstop ~]$ octave
GNU Octave, version 3.2.3
Copyright (C) 2009 John W. Eaton and others.
This is free software; see the source code for copying
conditions. There is ABSOLUTELY NO WARRANTY; not even for
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
For details, type 'warranty'.

Octave was configured for "i386-redhat-linux-gnu".

Additional information about Octave is available at http://
   www.octave.org.

Please contribute if you find this software useful.
For more information, visit http://www.octave.org/help-wanted
   .html

Report bugs to <bug@octave.org> (but first, please read
http://www.octave.org/bugs.html to learn how to write a
   helpful report).

For information about changes from previous versions, type '
   news'.

warning: mark_as_command is obsolete and will be removed from
    a future version of Octave
octave:1> p1 = gf([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0], 2);
octave:2> p2 = gf([1, 0, 0, 1, 1], 2);
octave:3> [chastn, remd] = deconv(p1, p2)
chastn =
```

```
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0   0   1   1   0   1   0   1   1   1

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   0   0   0   0   0   0   0   0   1   0   0   1

octave:4> p1 = gf([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 
    2);
octave:5> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0   0   1   1   0   1   0   1   1

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   0   0   0   0   0   0   0   1   1   0   1

octave:6> p1 = gf([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 2)
   ;
octave:7> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0   0   1   1   0   1   0   1
```

```
remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   0   0   0   0   0   0   1   1   1   1

octave:8> p1 = gf([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 2);
octave:9> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0   0   1   1   0   1   0

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   0   0   0   0   0   1   1   1   0

octave:10> p1 = gf([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 2);
octave:11> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0   0   1   1   0   1

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   0   0   0   0   0   1   1   1
```

```
octave:12> p1 = gf([1, 0, 0, 0, 0, 0, 0, 0, 0], 2);
octave:13> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0   0   1   1   0

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   0   0   0   1   0   1   0

octave:14> p1 = gf([1, 0, 0, 0, 0, 0, 0, 0], 2);
octave:15> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0   0   1   1

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   0   0   0   1   0   1

octave:16> p1 = gf([1, 0, 0, 0, 0, 0, 0], 2);
octave:17> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =
```

```
   1   0   0   1

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   0   1   0   1   1

octave:18> p1 = gf([1, 0, 0, 0, 0, 0, 0], 2);
octave:19> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0   0

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   1   1   0   0

octave:20> p1 = gf([1, 0, 0, 0, 0, 0], 2);
octave:21> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1   0

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =
```

```
   0   0   0   1   1   0

octave:22> p1 = gf([1, 0, 0, 0, 0], 2);
octave:23> [chastn, remd] = deconv(p1, p2)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   1

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

   0   0   0   1   1

octave:24> quit

[maxim@home-dekstop ~]$
```

Нахождение разрешённой комбинации систематического кода и остатков от деления полиномов для 2 раздела:

```
octave:1> a = gf([1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1,
   1, 0, 0, 0, 0, 0, 0], 2);
octave:2> x_r =  gf([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], 2);
octave:3> proizv = conv(a, x_r)
proizv =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

 Columns 1 through 22:

 1 0 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0

 Columns 23 through 31:
```

```
 0 0 0 0 0 0 0 0 0

octave:4> g = gf([1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1], 2);
octave:5> [chastn, remd] = deconv(proizv, bg)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

 1 1 0 0 0 1 0 0 1 0 0 0 1 1 1 0 0 0 1 1 1

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

 Columns 1 through 22:

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

 Columns 23 through 31:

 1 1 1 0 1 1 1 1 1

octave:6> sum = proizv + remd
sum =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

 Columns 1 through 22:

 1 0 0 1 1 1 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 1

 Columns 23 through 31:

 1 1 1 0 1 1 1 1 1
```

```
octave:7> v = gf([1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0,
   1, 1, 0,
0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0],2);
octave:8> [chastn, remd] = deconv (v, g)
chastn =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

 1 0 1 0 1 1 0 1 1 0 1 0 0 1 1 0 1 0 0 0 0

remd =
GF(2^2) array. Primitive Polynomial = D^2+D+1 (decimal 7)

Array elements =

 Columns 1 through 22:

 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

 Columns 23 through 31:

 0 0 0 0 0 0 0 0 0
```

Нахождение порождающего полинома, разрешённой кодовой комбинации, синдромов для кода БЧХ в разделе 3:

```
octave:1> g1 = gf([1,2],4);
octave:2> g2 = gf([1,4],4);
octave:3> g3 = gf([1,8],4);
octave:4> g4 = gf([1,3],4);
octave:5> g5 = gf([1,6],4);
octave:6> g6 = gf([1,12],4);
octave:7> proizv1 = conv(g1,g2);
octave:8> proizv2 = conv(g3,g4);
octave:9> proizv3 = conv(g5,g6);
octave:10> proizv4 = conv(proizv1,proizv2);
octave:11> g_x = conv(proizv3, proizv4)
g_x =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)
```

```
Array elements =

    1    7    9    3   12   10   12

octave:12> a = gf([0,0,4,0,5,10,1,0,9],4);
octave:13> x6 = gf([1,0,0,0,0,0,0],4);
octave:14> a_x6 = conv(a,x6)
a_x6 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

    0    0    4    0    5   10    1    0    9    0    0    0
      0    0    0

octave:15> [chastn, ost] = deconv(a_x6, g_x)
chastn =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

    0    0    4   15   12   10    3   13   15

ost =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

    0    0    0    0    0    0    0    0    0    9   10   13
      4   15    8

octave:16> v_x = a_x6 + ost
v_x =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =
```

```
    0    0    4    0    5   10    1    0    9    9   10   13
      4   15    8

octave:17> z = gf([1,2,4,8,3,6,12,11,5,10,7,14,15,13,9],4);
octave:18> v = gf([5,8,2,1,4,15,4,0,13,6,6,6,6,12,7],4);
octave:19> proizv = v .* z
proizv =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

    5    3    8    8   12    4    5    0   12    9    1    2
      4    3   10

octave:20> sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:21> z_2 = z .^ 2
z_2 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

    1    4    3   12    5    7   15    9    2    8    6   11
     10   14   13

octave:22> proizv = v .* z_2;
octave:23> sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0
```

```
octave:24> z_3 = z .^ 3;
octave:25> proizv = v .* z_3;
octave:26> sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:27> z_4 = z .^ 4;
octave:28> proizv = v .* z_4;
octave:29> sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:30> z_5 = z .^ 5;
octave:31> proizv = v .* z_5;
octave:32> sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:33> z_6 = z .^ 6;
octave:34> proizv = v .* z_6;
octave:35> sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =
```

```
   0
```

Нахождение синдромов, обратых матриц для дополнительной задачи в 3 разделе:

```
octave:1> v = gf([1,3,4,9,9,3,4,13,11,0,5,0,2,8,3],4);
octave:2> z = gf([1,2,4,8,3,6,12,11,5,10,7,14,15,13,9],4);
octave:3> z_2 = z .^ 2; z_3 = z .^ 3; z_4 = z .^ 4; z_5 = z
   .^ 5; z_6 = z .^ 6;
octave:4> proizv1 = v .* z; proizv2 = v .* z_2; proizv3 = v
   .* z_3;
octave:5> proizv4 = v .* z_4; proizv5 = v .* z_5; proizv6 = v
    .* z_6;
octave:6>  s_1 = sum(proizv1)
s_1 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   15

octave:7> s_2 = sum(proizv2)
s_2 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   1

octave:8> s_3 = sum(proizv3)
s_3 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   3

octave:9> s_4 = sum(proizv4)
s_4 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)
```

```
Array elements =

   1

octave:10> s_5 = sum(proizv5)
s_5 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   12

octave:11> s_6 = sum(proizv6)
s_6 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   1

octave:12> m_s = gf([[15,1,3];[1,3,1];[3,1,12]],4);
octave:13> m_s_1 = (m_s)^(-1)
m_s_1 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

    7    4   11
    4    8   14
   11   14   10

octave:14> a = gf([7,4,11],4);
octave:15> b = gf([1,12,1],4);
octave:16> c = a .* b;
octave:17> sum(c)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)
```

```
Array elements =

   9

octave:18> a = gf([4,8,14],4);
octave:19> c = a .* b;
octave:20> sum(c)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:21> a = gf([11,14,10],4);
octave:22> c = a .* b;
octave:23> sum(c)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   5

octave:24> g_x = gf([9,0,5,1],4);
octave:25> roots(g_x)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   3   4   7

octave:26> temp = gf([1,1,1],4);
octave:27> root = roots(g_x); betta = temp ./ root
betta =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =
```

```
   14   13    6

octave:28> betta_2 = betta .^ 2
betta_2 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   11   14    7

octave:29> betta_3 = betta .^ 3
betta_3 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

    8   10    1

octave:30> betta_matrix = gf
   ([[6,14,13];[7,11,14];[1,8,10]],4);
octave:31> betta_matrix_1 = (betta_matrix)^(-1)
betta_matrix_1 =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   10    3    1
    1    3   15
   11   10   12

octave:32> a = gf([10,3,1],4);
octave:33> s = gf([15,1,3],4);  tim = a .* s; sum(tim)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   12
```

```
octave:34> a = gf([1,3,15],4); tim = a .* s; sum(tim)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   14

octave:35> a = gf([11,10,12],4); tim = a .* s; sum(tim)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   14

octave:36> e = gf([0,0,0,0,0,12,0,0,0,0,0,14,0,14,0],4);
octave:37> v_x = v + e
v_x =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

    1    3    4    9    9   15    4   13   11    0    5   14
      2    6    3

octave:38> z = gf([1,2,4,8,3,6,12,11,5,10,7,14,15,13,9],4);
octave:39> proizv = v_x .* z; sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:40> proizv = v_x .* z_2;
octave:41> sum(proizv)
ans =
```

```
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:42> proizv = v_x .* z_3; sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:44> proizv = v_x .* z_4; sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:45> proizv = v_x .* z_5; sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0

octave:46> proizv = v_x .* z_6; sum(proizv)
ans =
GF(2^4) array. Primitive Polynomial = D^4+D+1 (decimal 19)

Array elements =

   0
```

Приложение В

(справочное)

Программа для нахождения таблицы состояний регистра

```
#include <iostream>
int main(int argc, char *argv[])
{
     using namespace std;
     const int WORD_SIZE = 21;
     const int SCHEME_SIZE = 10;
     int coding_scheme[SCHEME_SIZE];
     int info_word[WORD_SIZE] =
  {1,0,0,1,1,1,0,0,0,0,1,1,1,1,1,0,0,0,0,0,0};

     for (int i = 0; i < SCHEME_SIZE; i++) {
          coding_scheme[i] = 0;
     }

     for (int j = 0; j < WORD_SIZE; j++) {
          int word_symbol = info_word[j] ^ coding_scheme[9];

          int k = (SCHEME_SIZE - 1);
          for (k; k > 0; k--) {
               coding_scheme[k] = coding_scheme[k-1];
          }
          coding_scheme[0] = word_symbol;
          coding_scheme[3] = coding_scheme[3] ^ word_symbol;
          coding_scheme[5] = coding_scheme[5] ^ word_symbol;
          coding_scheme[6] = coding_scheme[6] ^ word_symbol;
          coding_scheme[8] = coding_scheme[8] ^ word_symbol;
          coding_scheme[9] = coding_scheme[9] ^ word_symbol;
          cout << info_word[j] << " | ";
          for (int m = 0; m < SCHEME_SIZE; m++)
            cout << coding_scheme[m];
          cout << endl;
     }
}
```

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Султанов Б. В., Иванов А. П., Геращенко С. М. — Математические основы построения помехоустойчивых блочных кодов. Учебное пособие. — Пенза, 2005. —69 с.
2. Кузнецов Ю. А., Грунтович М. М. — Проектирование кодирующих и декодирующих устройств средств связи. Методические указания к курсовой работе. — Пенза, 2004 — 19 с.