

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

федеральное государственное автономное образовательное учреждение высшего образования

«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

КАФЕДРА №52

Отчет защищен с оценкой _____

Преподаватель

ассистент

А.А. Бурков

должность, уч. степень,
звание

подпись, дата

инициалы,
фамилия

ОТЧЕТ О ЛАБОРАТОРНОЙ РАБОТЕ №1

ИСПОЛЬЗОВАНИЕ ЦИКЛИЧЕСКИХ КОДОВ ДЛЯ ОБНАРУЖЕНИЯ ОШИБОК В
СЕТЯХ ПЕРЕДАЧИ ДАННЫХ

по курсу: СЕТИ И СИСТЕМЫ ПЕРЕДАЧИ ИНФОРМАЦИИ

Студент гр. №

5912

И.К. Лобач

номер
группы

подпись,
дата

инициалы,
фамилия

Санкт-Петербург 2022

Цель работы: исследование типового алгоритма формирования контрольной суммы с использованием циклических кодов.

1 Описание моделируемой системы

Необходимо моделировать работу системы передачи данных в режиме обнаружения ошибок с применением следующего подхода: к передаваемым данным добавляют контрольную сумму, которая вычисляется на основе этих же данных. По каналу передается сообщение, состоящее из данных и контрольной суммы. Использование контрольной суммы позволяет определить, по принятому сообщению, возникли ли ошибки при передаче данного сообщения по каналу. Структурная схема такой системы передачи представлена

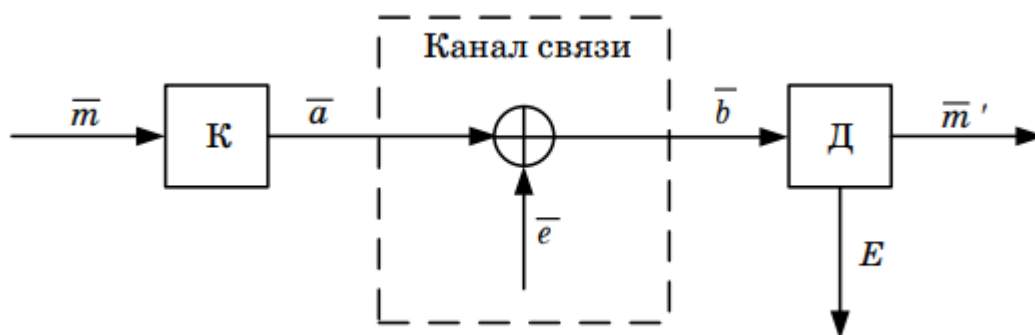


Схема 1 - Структурная схема системы передачи

На данной схеме \bar{m} – информационное сообщение, K – кодер, \bar{a} – закодированное сообщение, \bar{e} – вектор ошибок, \bar{b} – сообщение на выходе канала, D – декодер, E – принятое решение (о наличии ошибок), \bar{m}' – сообщение на выходе декодера.

На вход кодера поступает некоторое двоичное информационное сообщение \bar{m} . Кодер по некоторому алгоритму вычисляет контрольную сумму, дописывает ее к передаваемому сообщению и таким образом формирует закодированное двоичное сообщение \bar{a} . В канале могут произойти ошибки, в результате которых некоторые биты сообщения инвертируются, при этом канал может быть описан как операция XOR передаваемого сообщения и вектора ошибок. Декодер по некоторому алгоритму проверяет контрольную сумму в принятом сообщении и принимает одно из следующих решений:

$$E = \begin{cases} 0, & \text{если принято решение, что ошибки были} \\ 1, & \text{иначе} \end{cases}$$

2 Описание проводимого исследования и рассматриваемого алгоритма

Для описания рассматриваемого алгоритма будет рассмотрена работа кодера и декодера.

2.1 Кодер

Для описания работы с двоичными кодами используются многочлены с коэффициентами из $GF(2)$. Кодер хранит порождающий многочлен $g(x)$. Степень многочлена обозначается как $\deg(g(x)) = r$ и определяет количество бит контрольной суммы в кодовом слове. k – число информационных символов передаваемого сообщения \bar{m} . Передаваемое сообщение рассматривается как вектор длины k . Для каждого сообщения кодер выполняет следующие действия:

- 1) На основе последовательности \bar{m} формируется многочлен $m(x)$, степень которого $\deg(m(x)) \leq k - 1$;
- 2) Вычисляется $c(x) = m(x)x^r \bmod g(x)$, степень которого $\deg(c(x)) \leq r - 1$;
- 3) Вычисляется $a(x) = m(x)x^r + c(x)$, степень которого $\deg(a(x)) \leq k + r - 1$;
- 4) На основе $a(x)$ формируется последовательность \bar{a} длиной $n = k + r$.

2.2 Декодер

Декодер также хранит порождающий многочлен $g(x)$ и для каждого сообщения декодер выполняет следующие действия:

- 1) Принятое сообщение $\bar{b} = \bar{a} + \bar{e}$ преобразуется в многочлен $b(x)$;
- 2) Вычисляется синдром $s(x) = b(x) \bmod g(x)$;
- 3) Если $s(x) = 0$, то декодером принимается решение, что ошибок не было $E = 0$, в противном случае принимается решение $E = 1$.

Однако существует альтернативная реализация алгоритма декодера.

- 1) Принятое сообщение $\bar{b} = \bar{a} + \bar{e}$ состоит из информационной последовательности \bar{m}_b длины k и контрольной суммы \bar{c}_b длины r ;
- 2) Последовательность \bar{m}_b подается на вход кодеру, таким образом вычисляется контрольная сумма \bar{c}'_b ;
- 4) Если $\bar{c}'_b = \bar{c}_b$, то декодером принимается решение, что ошибок не было $E = 0$, в противном случае принимается решение $E = 1$.

3 Описание моделирующей программы в виде псевдокода или блок-схемы

Моделирующую программу можно разделить на три части. Описание каждой из них будет представлено в виде блок-схем.

3.1 Ввод данных

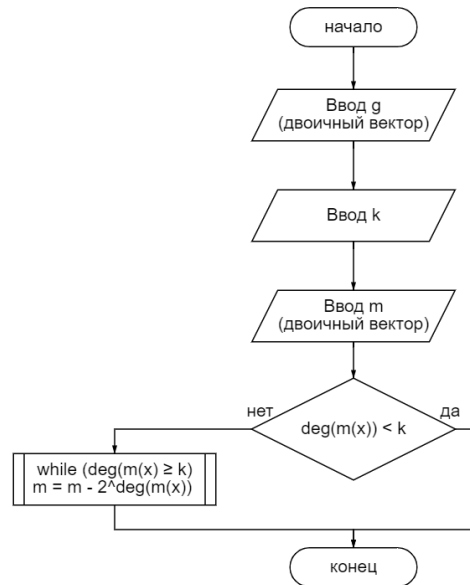


Схема 2 - Блок-схема ввода данных

3.2 Кодер

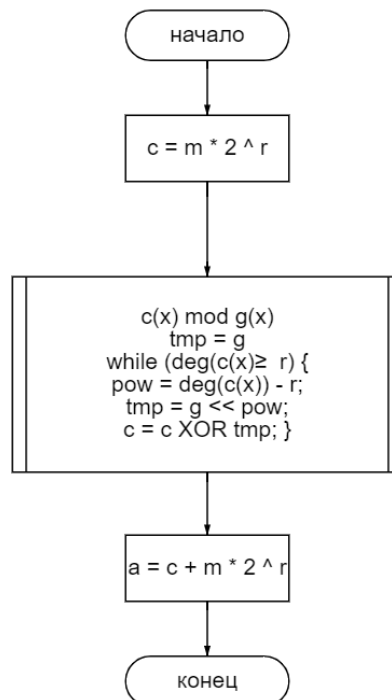
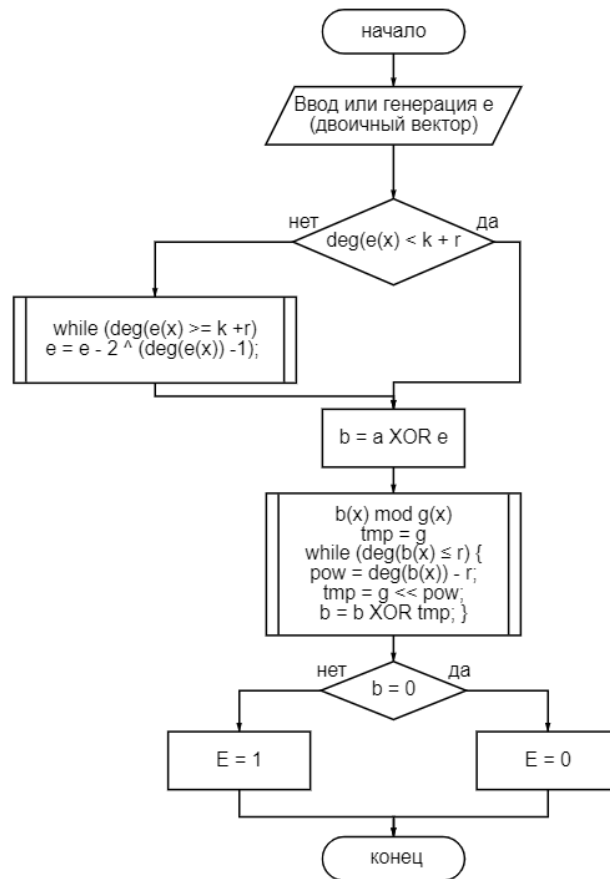
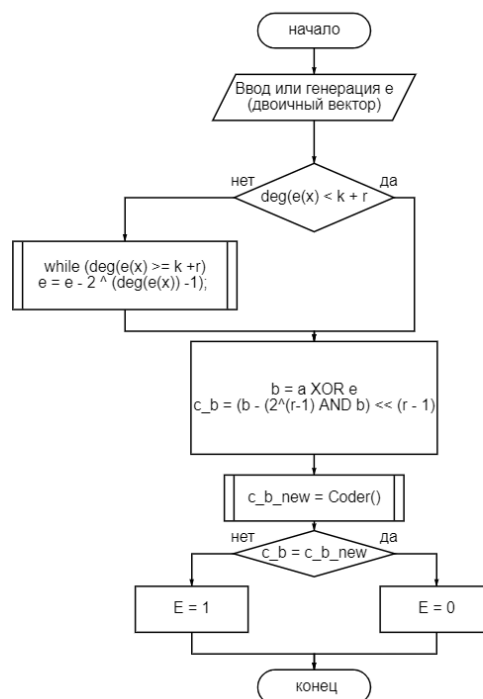


Схема 3 - Блок-схема кодера

3.3 Декодер



Альтернативный алгоритм декодирования:



4 Описание результатов проводимых исследований и зависимостей

Пример работы программы приведен на рисунках ниже:

```
Type g as a binary vector: 1011
g(x) = x^3 + x^1 + 1
g = 1011
r = 3

Type k: 4
Type m as a binary vector: 111010
m(x) = x^3 + x^1
m = 1010
k = 4
```

Рисунок 1 - Ввод данных

```
c(x) = x^1 + 1
c = 11

a(x) = x^6 + x^4 + x^1 + 1
a = 1010011
n = 7
```

Рисунок 2 - Работа кодера

```
Type e as a binary vector: 0000001
e(x) = + 1
e = 1

E = 1 first
```

Рисунок 3 - Работа декодера

```

c'(x) = x^1
c' = 10

c(x) = x^1 + 1
c = 11

E = 1 second

```

Рисунок 4 - Работа декодера (альтернативный алгоритм)

В результате проводимых исследований были получены решения E декодера, полученные двумя способами и которые показывали, была ли обнаружена ошибка. Также программа позволяла посмотреть все промежуточные этапы вычисления контрольной суммы и синдрома.

В результате моделирования было установлено, что при возникновении ошибки $e \in A$, ошибка не будет обнаружена ни одним из способов. Пример приведен ниже:

```

a(x) = x^6 + x^4 + x^1 + 1
a = 1010011
n = 7

Type e as a binary vector: 1010011
e(x) = + x^6 + x^4 + x^1 + 1
e = 1010011

E = 0 first

c'(x) =
c' = 0

c(x) =
c = 0

E = 0 second

```

Рисунок 5 - Ошибка не обнаружена

Так же моделирующая программа показала, что результаты, полученные типовым алгоритмом декодирования и альтернативным всегда совпадают, т.е., если ошибка была обнаружена первым декодером, то и второй ее обнаружит, в противном случае оба декодера примут неверное решение.

На вход декодерам подается $\bar{b} = \bar{a} + \bar{e}$. Как видно, на результат декодирования влияет произошедшая в канале ошибка \bar{e} . Тогда, чтобы проверить результаты работы 2 декодеров зафиксируем кодовое слово \bar{a} и переберем всевозможные векторы ошибки \bar{e} .

Пусть порождающий многочлен $g(x)$ и кодовое слово \bar{a} имеют вид:

```
Type g as a binary vector: 1011
r = 3
g(x) = x^3 + x^1 + 1
g = 1011

Type k: 4
Type m as a binary vector: 1010
m(x) = + x^3 + x^1
m = 1010

a(x) = x^6 + x^4 + x^1 + 1
a = 1010011
```

Рисунок 6 - Исходные данные

Моделирующая программа показала, что результаты работы, полученные типовым алгоритмом декодирования и альтернативным совпадают.

e = 1111010 E = 1 first E = 1 second	e = 10001 E = 1 first E = 1 second	e = 100 E = 1 first E = 1 second	e = 11100 E = 1 first E = 1 second
e = 1111011 E = 1 first E = 1 second	e = 10010 E = 1 first E = 1 second	e = 101 E = 1 first E = 1 second	e = 11101 E = 0 first E = 0 second
e = 1111100 E = 1 first E = 1 second	e = 10011 E = 1 first E = 1 second	e = 110 E = 1 first E = 1 second	e = 11110 E = 1 first E = 1 second
e = 1111101 E = 1 first E = 1 second	e = 10100 E = 1 first E = 1 second	e = 111 E = 1 first E = 1 second	e = 11111 E = 1 first E = 1 second
e = 1111110 E = 1 first E = 1 second	e = 10101 E = 1 first E = 1 second	e = 1000 E = 1 first E = 1 second	e = 100000 E = 1 first E = 1 second
e = 1111111 E = 0 first E = 0 second	e = 10110 E = 0 first E = 0 second	e = 1001 E = 1 first E = 1 second	e = 100001 E = 1 first E = 1 second

Рисунок 7 - Части работы моделирующей программы

Результаты работы 2 декодеров всегда совпадут. Это объясняется следующим образом. Пусть на вход декодерам подается один и тот же вектор $b(x)$, также оба декодера используют одинаковый порождающий многочлен $g(x)$. Докажем от противного, а именно, пусть декодеры примут разное решение: первый декодер не найдет ошибку, а второй найдет.

Пусть первый декодер примет решение, что ошибки не было, т.е. посчитанный им синдром $s(x)$ будет равен 0:

$$s(x) = 0 = b(x) = a(x) + e(x) \bmod g(x)$$

Известно, что, как и кодовое слово, $b(x)$ состоит из k бит информационной части и r бит, относящихся к КС. Обозначим пришедшую на вход декодеру информационную часть, как $m'(x)$, а КС как $c'(x)$. Тогда можно записать так:

$$a(x) + e(x) = m'(x)x^r + c'(x) = 0 \bmod g(x)$$

Тогда:

$$m'(x)x^r = c'(x) \bmod g(x)$$

Вспомним, что альтернативных декодер принимает решение путем сравнения двух КС. За первую КС принимается та, которая приходит ему на вход, т.е. $c'(x)$, а с помощью кодера

формируется еще одна КС, которую обозначим как $c''(x)$. Альтернативный декодер по нашему предположению принимает решение, что КС $c'(x)$ и $c''(x)$ не совпадают, т.е.

$$c'(x) \neq c''(x)$$

Теперь подробнее рассмотрим КС $c''(x)$. Она формируется путем кодирования информационной части $m'(x)$. Тогда, ее можно записать, как:

$$c''(x) = m'(x)x^r \bmod g(x)$$

Теперь объединим все полученные выражения:

$$c'(x) \neq c''(x) \Rightarrow m'(x)x^r \neq m'(x)x^r \bmod g(x)$$

Очевидно, что такая ситуация невозможна, а, значит, результаты декодеров всегда совпадут.

5 Выводы по проводимым исследованиям

В ходе выполнения лабораторной работы была разработана программа, демонстрирующая работу кодера и декодера для типового алгоритма формирования циклических кодов, а также работу альтернативного декодера.

В ходе работы моделирующей программы было установлено, что декодер принимает неверное решение для всех ошибок, принадлежащих множеству кодовых слов.

Так же моделирующая программа показала, что результаты, полученные типовым алгоритмом декодирования и альтернативным всегда совпадают, т.е., если ошибка была обнаружена первым декодером, то и второй ее обнаружит, в противном случае оба декодера примут неверное решение.

6 Листинг моделирующей программы

```
package com.suai;

import java.util.Scanner;

public class CyclicCodes {

    private int g;
    private int r;
    private int k;
    private int m;
    private int a;
    private int e;
    private double p = 0.5; // вероятность единицы
```

```

private class Pair {

    private int first;
    private int second;

    public Pair(int a, int b) {
        first = a;
        second = b;
    }
}

public CyclicCodes() throws Exception {
    Scanner in = new Scanner(System.in);

    System.out.print("Type g as a binary vector: ");
    if (in.hasNextInt()) {
        Pair pair = binaryStringToInt(in.nextLine());
        g = pair.first;
        r = pair.second;
    } else {
        throw new Exception("Incorrect g");
    }
    System.out.println("r = " + (r - 1));
    print("g", g, r);

    System.out.print("Type k: ");
    in = new Scanner(System.in);
    if (in.hasNextInt()) {
        k = in.nextInt();
        if (k <= 0) {
            throw new Exception("Incorrect k");
        }
    }

    System.out.print("Type m as a binary vector: ");
    in = new Scanner(System.in);
    if (in.hasNextInt()) {
        Pair pair = binaryStringToInt(in.nextLine());
        m = pair.first;
        int l = pair.second;
        if (l > k) {
            m = clipping(m, l, k);
        }
    }

    } else {

```

```

        throw new Exception("Incorrect m");
    }
    print("m", m, k + r);
}

private void print(String v, int val, int size) {
    System.out.print(v + "(x) = ");
    binaryVectorToPolynomial(val, size);
    System.out.println(v + " = " + Integer.toBinaryString(val));
    System.out.println();
}

private void setE() throws Exception {
    System.out.print("Type e as a binary vector: ");
    Scanner in = new Scanner(System.in);
    if (in.hasNextInt()) {
        Pair pair = binaryStringToInt(in.nextLine());
        e = pair.first;
        if (pair.second > (k + r)) {
            e = clipping(e, pair.second, (k + r));
        }
    } else {
        throw new Exception("Incorrect e");
    }
}

private int clipping(int num, int size, int requiredSize) { // обрезать delta единиц слева
    while (size != requiredSize) {
        num -= Math.pow(2, size - 1);
        size--;
    }
    return num;
}

private Pair binaryStringToInt(String str) {
    int size = str.length();
    int result = 0;
    for (int i = 0; i < str.length(); i++) {
        char h = str.charAt(i);
        if ((str.charAt(i) - '0') == 0 && result == 0) { // убираем 0 слева
            size--;
        } else {
            result += Math.pow(2, str.length() - 1 - i) * (str.charAt(i) - '0');
        }
    }
}

```

```

    return new Pair(result, size);
}

private void binaryVectorToPolynomial(int num, int size) { // size - фактический размер
    if (num - Math.pow(2, size - 1) >= 0) {
        System.out.print("x^" + (size - 1));
        num -= Math.pow(2, size - 1);
    }
    for (int i = size - 2; i > 0; i--) {
        if (num - Math.pow(2, i) >= 0) {
            System.out.print(" + x^" + i);
            num -= Math.pow(2, i);
        }
    }
    if (num == 1) {
        System.out.print(" + 1");
    }
    System.out.println();
}

private int getSize(int num, int upperBound) {
    int i = upperBound;
    for (; i >= 0; i--) {
        if (num - Math.pow(2, i) >= 0) {
            break;
        }
    }
    return (i + 1);
}

private int division(int num, int size) { // num - делимое, size - фактический размер
    int tmp = g;
    while (size - r >= 0) {
        int pow = size - r;
        tmp = g << pow;
        num = num ^ tmp;
        size = getSize(num, size);
    }
    return num;
}

public void encoding() {
    int c = m * (int) Math.pow(2, r - 1);
    c = division(c, k + r);
    a = c + m * (int) Math.pow(2, r - 1);
}

```

```

}

private void generateE() {
    e = 0;
    for (int i = 0; i < (k + r - 1); i++) {
        double tmp = Math.random();
        if (tmp <= p) {
            e += Math.pow(2, i);
        }
    }
}

// генерация кодовых слов
private void E(int bin, int size, int cur, byte mArray[]) throws Exception {
    if (cur == size) {
        e = 0;
        for (int f = 0; f < size; f++) {
            e += Math.pow(2, f) * mArray[size - f - 1];
        }
        System.out.println("e = " + Integer.toBinaryString(e));
        decoding1();
        decoding2();
    } else {
        for (byte num = 0; num < bin; num++) {
            mArray[cur] = num;
            E(bin, size, cur + 1, mArray);
        }
    }
}

// прохожусь по всем ошибкам с фиксированным кодовым словом
private void testGenerateE() throws Exception {
    byte[] mArray = new byte[k + r - 1];
    E(2, k + r - 1, 0, mArray);
}

public void decoding1() throws Exception {
    int b = a ^ e;
    if (division(b, k + r) == 0) {
        System.out.println("E = 0 first");
    } else {
        System.out.println("E = 1 first");
    }
}

//print("b", b, getSize(b,k+r));

```

```

}

public void decoding2() throws Exception {
    int b = a ^ e;
    int controlSum = 0;
    for (int i = 0; i < r - 1; i++) {
        controlSum += Math.pow(2, i);
    }
    controlSum = controlSum & b;
    m = (b - controlSum) >> (r - 1);
    //print("c", controlSum, getSize(controlSum, r));

    encoding();
    int newControlSum = 0;
    for (int i = 0; i < r - 1; i++) {
        newControlSum += Math.pow(2, i);
    }
    newControlSum = newControlSum & a;
    //print("c", newControlSum, getSize(newControlSum, r));

    if (newControlSum == controlSum) {
        System.out.println("E = 0 second");
    } else {
        System.out.println("E = 1 second");
    }
    System.out.println();
}

public static void main(String[] arg) {
    try {
        CyclicCodes c = new CyclicCodes();
        c.encoding();
        c.print("a", c.a, c.getSize(c.a, c.k + c.r));

        c.testGenerateE();
        //c.setE();
        //    c.decoding1();
        //    c.decoding2();

    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
}

```

}
}