МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»

Кафедра ВС и ТС

Отчет по лабораторной работе №5
по дисциплине
Основы систем мобильной связи

по теме:
ЦИКЛИЧЕСКИЙ ИЗБЫТОЧНЫЙ КОД. CRC

Студент:
Группа ИА-331 *Я.А Гмыря*

Предподаватель:
Заведующая кафедрой ТС и ВС *В.Г Дроздова*

Новосибирск 2025 г.

СОДЕРЖАНИЕ

ЦЕЛЬ И ЗАДАЧИ

Цель:

Получить представление о том, как осуществляется проверка на наличие ошибок в пакетах с данными в современных системах связи (Error detection) посредством использования циклического избыточного кода CRC (Cyclic Redundancy Check).

Задачи:

Порядок выполнения работы:

1) Напишите программу на языке C/C++ для вычисления CRC для пакета данных длиной *N* бит (N= 20 + порядковый номер в журнале) и определения факта наличия ошибки при передаче пакета по каналу связи.

5

2) Порождающий полином *G* для делителя выберите в соответствии с вариантом. Номер варианта – порядковый номер в журнале группы.

Табл. 5.1 Варианты заданий.

№ варианта	Непрерывная периодическая функция	№ варианта	Непрерывная периодическая функция
1	$G=x^7+x^5+x^4+x^3+x^2+x+1$	15	$G=x^7+x^5+x^2+x+1$
2	$G=x^7+x^6+x^4+x^3+x^2+x+1$	16	$G=x^7+x^2+x+1$
3	$G=x^7+x^6+x^5+x^3+x^2+x+1$	17	$G=x^7+x^6+x^2+x$
4	$G=x^7+x^6+x^5+x^4+x^2+x+1$	18	$G=x^7+x^6+x^5+x^4+x^3+x^2+x+1$
5	$G=x^7+x^6+x^5+x^4+x^3+x+1$	19	$G=x^7+x^6+x+1$
6	$G=x^7+x^6+x^5+x^4+x^3+x^2+1$	20	$G=x^7+x^6+x^5+x^3+x^2+x$
7	$G=x^7+x^6+x^5+x^4+x^3+x^2+x$	21	$G=x^7+x^4+x^3+x^2+x+1$
8	$G=x^6+x^5+x^4+x^3+x^2+x+1$	22	$G=x^7+x^6+x^5+x^4$
9	$G=x^7+x^5+x^3+x^2+x+1$	23	$G=x^7+x^6$
10	$G=x^7+x^6+x^4+x^3+x^2+x$	24	$G=x^7+x^5+x^4+x^3+x^2$
11	$G=x^7+x^5+x^4+x^3+x+1$	25	$G=x^7+x^3+x^2+x+1$
12	$G=x^7+x^6+x^4+x^3+x^2+x$	26	$G=x^7+x^6+x^4+x^3+1$
13	$G=x^7+x^5+x^4+x^3+x^2+1$	27	$G=x^7+x^6+x^4+x^3+x^2$
14	$G=x^7+x^6+x^3+x^2+x$	28	$G=x^7+1$

3) Добавьте полученный остаток от деления на G к пакету исходными данными и на приемной стороне вычислите повторно остаток от деления пакета с данными+CRC на полином G. Определите есть ли ошибка в принятом пакете. Выведите в окно терминала полученное значение CRC и отчет об ошибках в принятом пакете.
4) Возьмите N, равное 250 битам. Проделайте п.1-3
5) Сделайте цикл из 250+CRC length итераций и в этом цикле по очереди искажайте по одному биту – с 0-го до 250+CRC-1, проверьте в соответствии с п.3 обнаружена ли ошибка на приемной стороне и выполните подсчет того сколько раз за этот цикл приемник обнаружил и не обнаружил ошибки. Результат выведите в окно терминала.

Рисунок 1 — Задание к лабораторной работе

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1 Что такое CRC?

CRC (Cyclic Redundancy Check) - циклический избыточный код, контрольный код или контрольная сумма. Представляет собой добавочную порцию бит, вычисляемых по заранее известному алгоритму на основе исходного передаваемого пакета, которая передаётся вместе с самим пакетом и служит для контроля его безошибочной передачи.

1.2 Формирование CRC

CRC – остаток от двоичного деления оригинального пакета с данными на какое-то двоичное n-разрядное число (порождающий полином), и его длина будет равна n-1 бит. Основной операцией, используемой при делении бинарных чисел, является исключающее ИЛИ (XOR). Делитель принято записывать в виде полинома. Если считать, что каждый разряд делителя — это коэффициент полинома, то этот полином будет иметь вид: x^n,$x^{n-1}...x^1, x^0$. Если член полинома отсутствует, то на его месте стоит 0.

1.3 Формирование пакета с CRC кодом

Полученный остаток от деления CRC добавляется на передающей стороне к исходным данным и уже эта битовая последовательность, преобразованная в радиосигнал, передается в канал связи. CRC добавляется в конец пакета.

1.4 Работа с CRC на приемной стороне

На приемной стороне для обнаружения ошибки (или ее отсутствия) с полученным пакетом осуществляется ровно такая же процедура – деление на порождающий CRC полином. Если полученный в результате данного деления остаток будет ненулевым, то фиксируется факт наличия ошибки.

1.5 Пример

Рассмотрим пример, где имеется 7 бит данных: 100100 и 4-битный порождающий полином 1101.

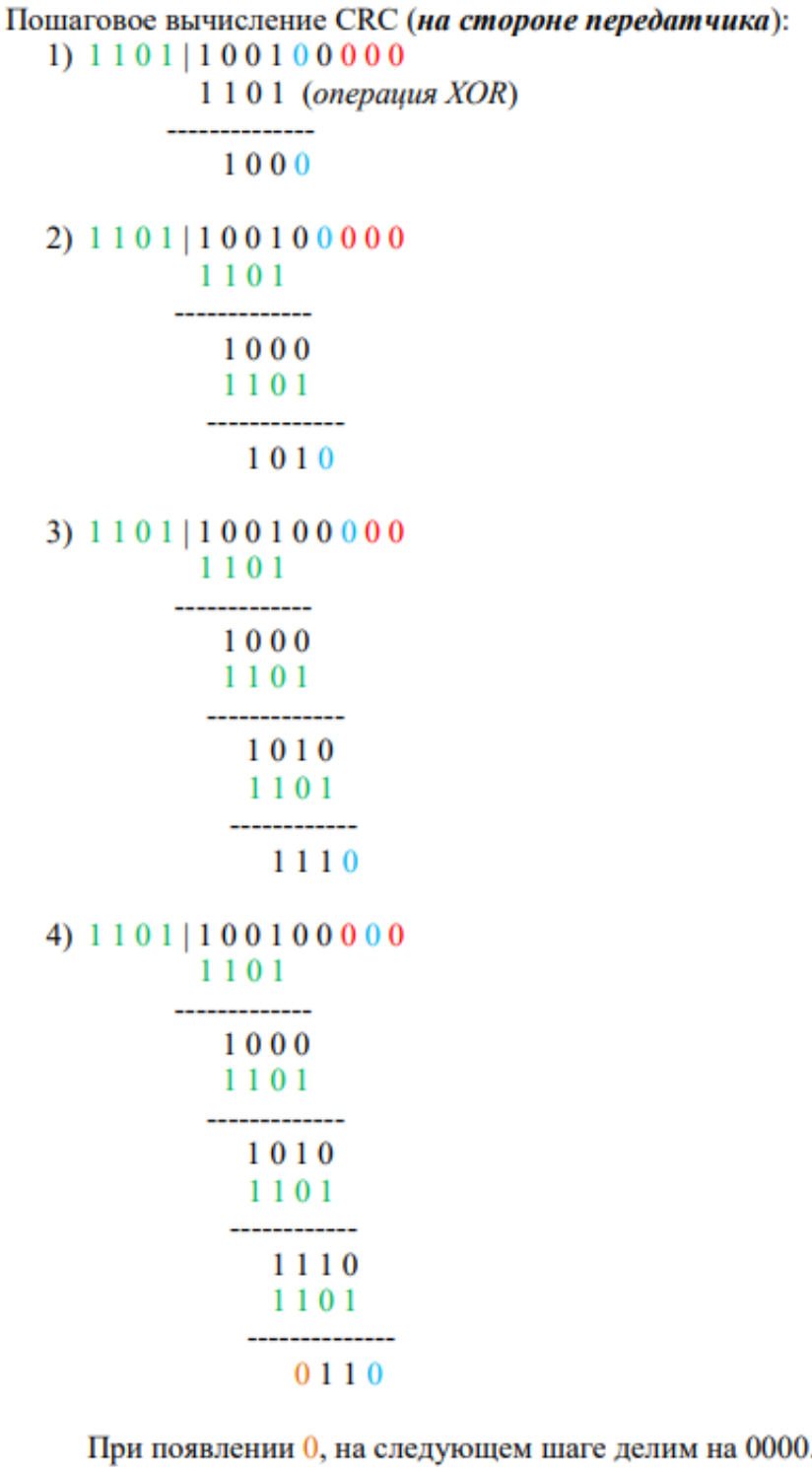

Рисунок 2 — Пример подсчета CRC

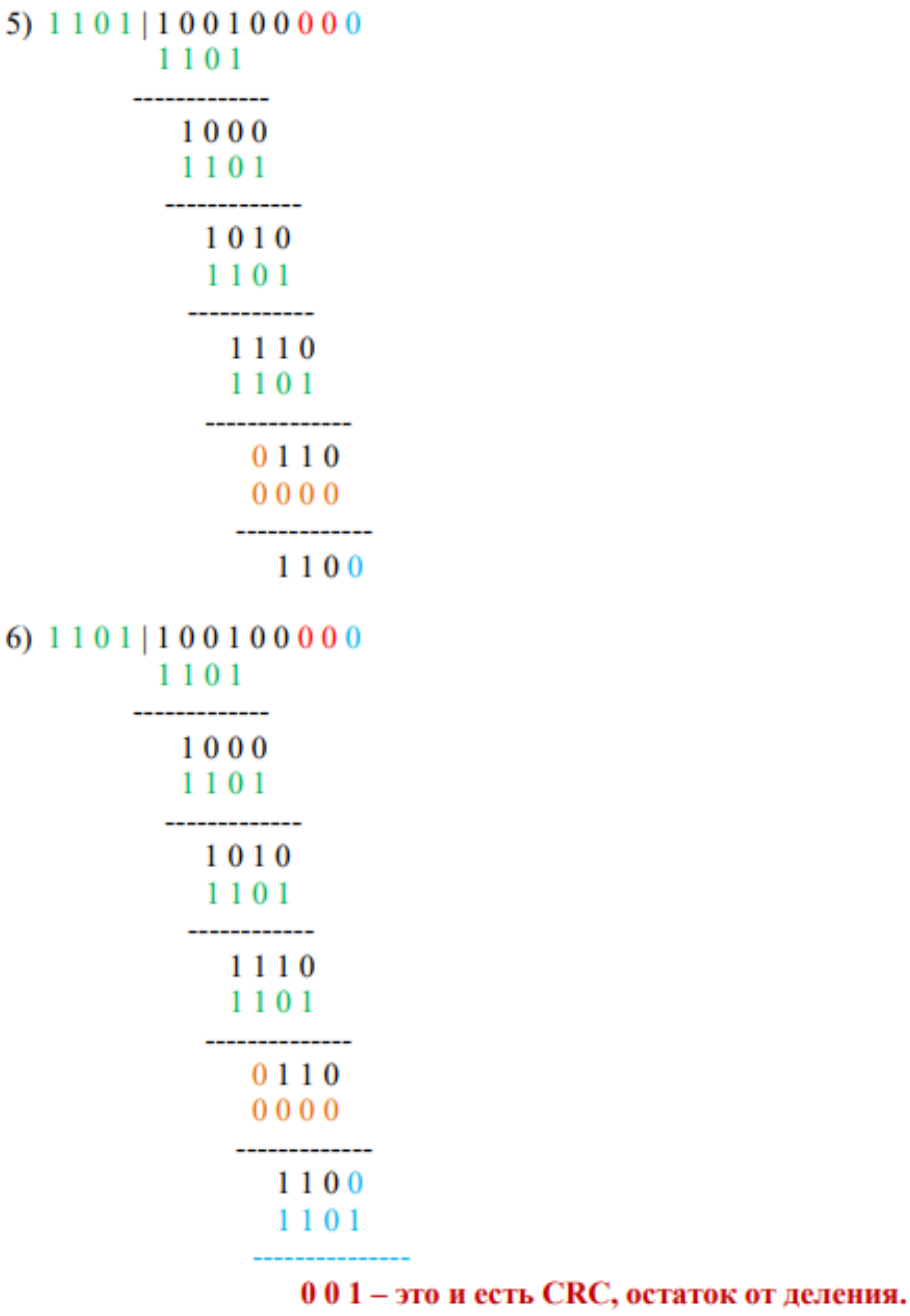

Рисунок 3 — Пример подсчета CRC

ПОДСЧЕТ CRC

2.1 Формирование информационной части пакета

Информационной частью пакета будет являться рандомная битовая последовательность. Напишем функцию для генерации такой последовательности:

```
std::vector<int8_t> generate_data(int N){

    std::vector<int8_t> result(N);

    for(int i = 0; i <= N; ++i){
        result[i] = rand() % 2;
    }

    return result;
}
```

Создаем вектор на N элементов и в цикле заполняем его рандомными битами.

2.2 Формирование CRC

```
template <typename T>
std::vector<T> crc(std::vector<T> data, std::vector<T> poly) {

    // add to data poly.size()-1 zeros
    for (int i = 0; i < poly.size() - 1; ++i)
        data.push_back(0);

    std::vector<T> result = data;

    // main cycle
    while (result.size() >= poly.size()) {

        //xor bw current result and polynome
        for (int i = 0; i < poly.size(); ++i)
            result[i] ^= poly[i];
```

```
        //delete leading zeros in current result
        while (!result.empty() && result[0] == 0){
            result = rm_front(result);
        }
    }

    //add zeros
    while(result.size() < poly.size()-1){
        result = push_front(result, static_cast<T>(0));
    }

    return result;
}
```

Функция принимает два вектора: data - информационная часть пакета, poly - порождающий полином, и возвращает вектор, содержащий CRC. Сначала добавляем в конец информационной части poly.size() нулей, после этого в главном цикле во вложенном цикле итерируемся по информационной последовательности и порождающему полиному и выполняем операцию xor между их элементами (выполняем деление). После этого уже в другом вложенном цикле удаляем ведущие нули, которые могли образоваться после деление в переменной result. Выполняем эти действия до тех пор, пока CRC (переменная result) не станет меньше размера полинома. В конце добавляем ведущие нули к CRC, чтобы он соответсвовал размеру poly.size()-1. Возвращаем вектор result, содержащий CRC.

2.3 Вспомогательные функции

2.3.1 Вывод вектора

```
template <typename T>
void print_vec(const std::vector<T>& vec){

    //iterate on vector and printf elements
    for(T el : vec){
        printf("%d", el);
    }

    printf("\n");
```

```
}
```

Интерируемся по вектору и выводим каждый его элемент.

2.3.2 Удаление первого элемента вектора

```
template <typename T>
std::vector<T> rm_front(const std::vector<T>& vec){
    std::vector<T> result;

    for(int i = 1; i < vec.size(); ++i){
        result.push_back(vec[i]);
    }

    return result;
}
```

Создаем новый ввектор, в который копируем все элементы исходного вектора, кроме первого.

2.3.3 Добавление элемента в начало вектора

```
template <typename T>
std::vector<T> push_front(const std::vector<T>& vec, T value){
    std::vector<T> result(vec.size() + 1);

    for(int i = 1; i < vec.size()+1; ++i){
        result[i] = vec[i-1];
    }

    result[0] = value;

    return result;
}
```

Создаем новый вектор, размер которого на 1 больше размера исходного, пишем в него исходный вектор, начиная с первого элемента. В нулевой элемент записываем переданное значение.

2.3.4 Функция для тестирования

```
template <typename T>
void crc_test(const std::vector<T>& data, const std::vector<T>&
   polynome){

    //print input values
    printf("data:     ");
    print_vec(data);

    printf("polynome: ");
    print_vec(polynome);

    //compute and print crc on TX
    std::vector<int8_t> tx_crc = crc(data, polynome);
    printf("TX CRC: ");
    print_vec(tx_crc);

    //add crc to data
    std::vector<T> data_with_crc;
    data_with_crc.reserve(data.size() + tx_crc.size());
    data_with_crc.insert(data_with_crc.begin(), data.begin(),
       data.end());
    data_with_crc.insert(data_with_crc.end(), tx_crc.begin(),
       tx_crc.end());

    printf("data + CRC: ");
    print_vec(data_with_crc);

    //compute crc on RX
    std::vector<T> rx_crc = crc(data_with_crc, polynome);

    printf("RX CRC: ");
    print_vec(rx_crc);
}
```

Функция принимает информационную часть пакета и порождающий полином. В функции осуществляется тест по следующим шагам:

1. Сгенерировать CRC

2. Сформировать пакет, состоящий из информационной части и CRC (добавляем в конец)
3. Подсчитать CRC от пакета из прошлого шага (должны получить 0, если все ок)

2.4 Тесты

2.4.1 Подсчет CRC для небольшого пакета данных

Сгенерируем битовую последовательность (информационную часть пакета) длиной 26 бит. В качестве порождающего полинома будем использовать $x^7+x^6+x^5+x^4+x^3+x^2+1$, т.е битовую последовательность 1, 1, 1, 1, 1, 1, 0, 1. Далее подсчитаем CRC и добавим его в конец информационной части. Далее реализуем логику приемника: используя битовую последовательность, полученную на прошлом шаге, и порождающий полином, подсчитаем CRC. Если биты не были повреждены, то такая операция даст нам 0. Если была ошибка, то получим число, отличное от нуля.

```
int main(){
    int N = 20;
    int num_in_journal = 6;

    printf("\n\n############################ TEST 1
       ########################################\n\n");

    std::vector<int8_t> data = generate_data(N + num_in_journal);

    std::vector<int8_t> polynome = {1,1,1,1,1,0,1,1};

    crc_test(data, polynome);

    return 0;
}
```

```
############################ TEST 1 ########################################

data:     10111100110101100000101100
polynome: 11111011
TX CRC: 1100010
data + CRC: 10111100110101100000101100 1100010
RX CRC: 0000000
```

Рисунок 4 — Тест №1

Можем видеть, что CRC имеет длину на единицу меньше, чем полином, а при подсчете CRC на приемной стороне получаем 0.

2.4.2 Подсчет CRC для большого пакета данных

Выполним те же действия, но уже с длиной информационной в 250 бит

```
printf("\n\n############################ TEST 2
   ########################################\n\n");

data = generate_data(250);
polynome = {1,1,1,1,1,1,0,1};

crc_test(data, polynome);
```

```
######################## TEST 2 ###################################

data:     1111000111010111010010101001000111010101110101010010100000111010000100001100011101000100111010111111111101101011001011100010101010111100010100000101111100110110101111100000001110010101010011001000111001001100001011100100000100000100000111001111100
polynome: 11111101
TX CRC: 1111100
data + CRC: 1111000111010111010010101001000111010101110101010010100000111010000100001100011101000100111010111111111101101011001011100010101010111100010100000101111100110110101111100000001110010101010011001000111001001100001011100100000100000100000111001111100 1111100
RX CRC: 0000000
```

Рисунок 5 — Тест №2

CRC на приемной стороне равен 0, т.е пакет дошел без искажений.

2.4.3 Тест с искажением пакета

Теперь в цикле поочередно будем искажать каждый бит и подсчитывать CRC. Заведем счетчик ошибок, и если CRC не нулевой, то будем увеличивать счетчик

```
printf("\n\n############################ TEST 3
   ########################################\n\n");
```

```
    int errors_count = 0;

    //compute and print crc on TX
    std::vector<int8_t> tx_crc = crc(data, polynome);

    //add crc to data
    std::vector<int8_t> data_with_crc = data;

    for(int i = 0; i < tx_crc.size(); ++i){
        data_with_crc.push_back(tx_crc[i]);
    }

    for(int i = 0; i < data_with_crc.size(); ++i){

        //distort packet
        data_with_crc[i] ^= 1;

        //compute crc on RX
        std::vector<int8_t> rx_crc = crc(data_with_crc,
           polynome);

        if(std::accumulate(data_with_crc.begin(),
           data_with_crc.end(), 0)){
            ++errors_count;
        }

        //recovery packet
        data_with_crc[i] ^= 1;

    }

    printf("Errors_count: %d\n", errors_count);
```

```
######################## TEST 3 ######################################

Errors_count: 257
```

Рисунок 6 — Тест №3

Видим, что при помощи CRC мы в 100% случаев обнаружили ошибку в пакете. Изменение значения любого бита во время передачи данных ведет

к тому, что CRC на приемной стороне будет ненулевой. Таким образом при помощи CRC можно легко узнавать, повредились ли данные при передаче.

2.5 Контрольные вопросы

1. Для чего в мобильных сетях используются CRC-проверки?
2. Что такое порождающий полином?
3. Как вычислить CRC для пакета с данными?

Ответы

1. CRC-проверки в мобильных сетях используются для проверки факта искажения информации при передаче по каналу.
2. Порождающий полином - битовая последовательность, которая позволяет высчитать CRC. По сути этот полином - делитель пакета с данными.
3. Необходимо в конец битовой последовательности добавить n-1 нулей, где n - длина порождающего полинома. После этого с помощью операции XOR выполнить двоичное деление битовой последовательности с данными на порождающий полином.

ВЫВОД

В ходе работы я получил представление о том, как осуществляется проверка на наличие ошибок в пакетах с данными в современных системах связи (Error detection) посредством использования циклического избыточного кода CRC (Cyclic Redundancy Check).