Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

ОТЧЁТ

По лабораторной работе № 3

На тему

**МЕТОДЫ ЗАЩИТЫ ОТ ОШИБОК, ИСПОЛЬЗУЕМЫЕ В СЕТЯХ**

Выполнил: Проверил:

Студент группы 050503 Преподаватель

Деруго Д. В. Марцинкевич В. А.

Минск 2022

1. **Цель работы:**

Изучить методы защиты от ошибок, применяемые в СПД. Отработать программы, реализующие процедуры формирования помехозащищенных кадров и получения информации из них.

1. **Задание к лабораторной работе:**

Разработать программный модуль реализации процедуры кодирования и декодирования в коде Хемминга с 8 разрядами исходного кода и исправлением 1 ошибки.

**3. Теоретическая часть**

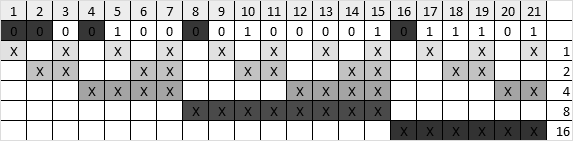
Из-за воздействия различных помех большинство типов линий связи претерпевает 1 ошибку на 104 – 106 бит, достоверность передачи информации достаточно низкая. Для защиты от ошибок предпринимаются системы с обратной связью, многократные посылки данных и избыточное кодирование специальными кодами. Характеристиками кодов являются кодовое расстояние, коэффициент обнаружения и исправления ошибок, избыточность и другие. Суть кодов Хемминга в том, что контрольные биты на фиксированных позициях подсчитывают единицы в посылке блоками различной длины, зависящей от позиции бита. Если сумма нечетная, то бит принимает значение единицы. При раскодировании путем контроля на четность получают комбинацию, и если она отлична от нуля в двоичном коде, то полученная комбинация указывает на позицию, в которой произошла ошибка. Существенным недостатком кодов Хемминга является то, что они исправляют лишь одиночные ошибки, хотя и распознают множественные.

Рис. 1 – Пример кодирования данных длиной в 16 бит

В практической части в качестве данных передается строка «01101101», которая кодируется четырьмя избыточными контрольными битами. После проводится тест исправления ошибки путем изменения одного из битов посылки.

1. **Практическая часть**

import serial

def encode\_hamming(hamming\_encoded, r):

n = len(hamming\_encoded)

for i in range(r):

count = 0

for j in range(2 \*\* i - 1, n, (2 \*\* i) \* 2):

# Check all bits in

for k in range(j, j + 2 \*\* i):

if len(hamming\_encoded) > k and hamming\_encoded[k] == '1':

count += 1

hamming\_encoded = hamming\_encoded[:2 \*\* i - 1] + str(count % 2) + hamming\_encoded[2 \*\* i:]

return hamming\_encoded

def detect\_errors(hamming\_encoded):

c = (len(hamming\_encoded) - 1).bit\_length()

n = len(hamming\_encoded)

wrong\_bits = []

for i in range(c):

count = 0

for j in range(2 \*\* i - 1, n, (2 \*\* i) \* 2):

# Check all bits in

for k in range(j, j + 2 \*\* i):

if len(hamming\_encoded) > k != 2 \*\* i - 1 and hamming\_encoded[k] == '1':

count += 1

if hamming\_encoded[2 \*\* i - 1] != str(count % 2):

wrong\_bits.append(2 \*\* i)

if len(wrong\_bits) != 0:

wrong\_bit = sum(wrong\_bits)

print(f"Error in bits {str(wrong\_bits)}. Must change {wrong\_bit} bit")

hamming\_encoded = hamming\_encoded[:wrong\_bit - 1] + str(

int(not int(hamming\_encoded[wrong\_bit - 1]))) + hamming\_encoded[wrong\_bit:]

return hamming\_encoded

def control\_bits\_count(m):

for i in range(m):

if 2 \*\* i >= m + i + 1:

return i

return 0

def find\_bits\_position(data, r):

if r == 0:

return data

j, k, m, res = (0, 0, len(data), '')

for i in range(1, m + r + 1):

if i == 2 \*\* j:

res = res + 'X'

j += 1

else:

res += data[k]

k += 1

return res

def decode\_hamming(hamming\_encoded):

c = (len(hamming\_encoded) - 1).bit\_length()

for i in range(c, -1, -1):

if len(hamming\_encoded) > 2 \*\* i:

hamming\_encoded = hamming\_encoded[:2 \*\* i - 1] + hamming\_encoded[2 \*\* i:]

return hamming\_encoded

def main():

# Test case:

data = "01101101"

print(f"Unencoded data: {data}")

c = control\_bits\_count(len(data))

arr\_x = find\_bits\_position(data, c)

print(f"Defined position: {arr\_x}")

arr1 = encode\_hamming(arr\_x, c)

print(f"Encoded data: \t {arr1}")

arr1 = "000111010101" # with error

# decoded XX0X110X1101

# right 000111011101

# error: 000111010101 error in bit 9 from

print(f"Wrong transmitted: {arr1}")

no\_errors = detect\_errors(arr1)

print(f"Decoded: \t {decode\_hamming(no\_errors)}")

if \_\_name\_\_ == '\_\_main\_\_':

main()