Министерство образования Республики Беларусь

Учреждение образования

“Белорусский государственный университет информатики и радиоэлектроники”

Факультет компьютерных систем и сетей  
Кафедра электронных вычислительных машин  
Дисциплина: Основы компьютерных сетей

Отчет по лабораторной работе № 3

на тему

“Программная реализация кода Хэмминга”

Выполнил:  
студент группы 150501 Климович А.Н.

Проверил:  
старший преподаватель Глецевич И.И.

Минск 2023

**1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ**

**1.1 Исходные данные**

Для написания и отладки программы использовался Qt Creator. В качестве языка программирования был выбран язык C++ и фреймворк Qt для создания графического интерфейса.

Для эмуляции COM-портов использовался Virtual Serial Ports Emulator, задействующий архитектуру UART 16550.

Программа запускалась на ОС Windows 10.

**1.2 Исходные требования к коду Хэмминга**

Параметры кода Хэмминга, которые были учтены при выполнении задания.

* Количество информационных битов (m) – это количество битов, содержащие исходную информацию, которую необходимо закодировать. Значение данного параметра равно 11.
* Количество проверочных битов (k) – это количество битов, которые добавляются к информационным битам для обнаружения и исправления ошибок. По формуле 2k >= k + m + 1 значение данного параметра равно 4.
* Длина кодируемого слова – это сумма k+m для каждого кодового слова. С учетом предыдущих параметров длина кодируемого слова равна 15.
* Минимальное кодовое расстояние (dmin). Значение данного параметра определяется по формуле dmin >= 2\*t + 1, где t – количество исправляемых ошибок. Согласно исходным требованиям к коду Хэмминга t = 1. Тогда dmin = 3.

dmin удовлетворяет нашему коду, так как 3 >= 2 \* 1 + 1. В правильности найденного минимального кодового расстояния можно убедиться, если проанализовать все возможные пары кодовых слов и определить минимальное количество различающихся битов между ними. Из пары кодовых слов 000000000000000 111000000000000 определяем, что минимальное кодовое расстояние действительно равно 3. Такой код будет исправлять одиночные ошибки.

* **2 ПРАКТИЧЕСКАЯ ЧАСТЬ**

**2.1 Изменения и дополнения кода программы**

**Файл Frame.cpp:**

void Frame::encodeHammingCode()

{

QString encodedData = data.leftJustified(DATA\_SIZE, '?');

int amountOfControlBits = 0;

for(int i = 1; i <= encodedData.size(); i \*= 2, ++amountOfControlBits)

encodedData.insert(i - 1, '0');

QList<char> controlBits(amountOfControlBits);

for(int i = 0, step = 1; i < controlBits.length(); ++i, step \*= 2)

{

int counterOfOne = 0;

for(int j = step - 1; j < encodedData.size(); j += 2 \* step)

counterOfOne += encodedData.mid(j, step).count('1');

controlBits[i] = counterOfOne % 2 == 0 ? '0' : '1';

}

fcs.clear();

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

fcs.append(controlBits[i]);

updateFrame();

}

void Frame::decodeHammingCode()

{

Frame newFrame;

newFrame.setData(data);

// Заново кодируем Data c искаженным битом

newFrame.encodeHammingCode();

QString unmatchedControlBits = "";

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

{

if(newFrame.getFcs()[i] != fcs[i])

unmatchedControlBits.append('1');

else

unmatchedControlBits.append('0');

}

this->setFcs(newFrame.getFcs());

unsigned long indexOfBadBit = 0;

for(int i = 0, bitValue = 1; i < unmatchedControlBits.size(); ++i, bitValue \*= 2)

indexOfBadBit += unmatchedControlBits[i] == '1' ? bitValue : 0;

auto isPowerOfTwo = [](int n) {

return (n > 0) && ((n & (n - 1)) == 0);

};

if(indexOfBadBit > 0 && !isPowerOfTwo(indexOfBadBit))

{

QString encodedData = data;

unsigned long controlBitIndex = 1;

for(; controlBitIndex <= encodedData.size(); controlBitIndex \*= 2)

encodedData.insert(controlBitIndex - 1, '0');

encodedData[indexOfBadBit - 1] = encodedData[indexOfBadBit - 1] == '1' ? '0' : '1';

for(; controlBitIndex > 0; controlBitIndex /= 2)

encodedData.remove(controlBitIndex - 1, 1);

data = encodedData;

}

data = data.leftJustified(DATA\_SIZE, '?');

updateFrame();

}

QString Frame::sendFrame(QSerialPort& port)

{

this->encodeHammingCode();

this->setData(data.leftJustified(DATA\_SIZE, '?'));

this->makeBitError();

this->setFrame(this->doBitStaffing());

for(auto bit : frame)

port.write(QString(bit).toStdString().c\_str());

return frame;

}

void Frame::makeBitError()

{

srand(time(NULL));

if((rand() % (100 - 0 + 1) + 0) <= 70)

{

unsigned long indexOfDataEnd = frame.indexOf('?') - 1;

if(indexOfDataEnd == -2UL)

indexOfDataEnd = frame.size() - fcs.size() - 1;

std::pair<unsigned long, unsigned long> dataRange = {16, indexOfDataEnd};

unsigned long randomIndex = rand() % (dataRange.second - dataRange.first + 1) + dataRange.first;

frame[randomIndex] = frame[randomIndex] == '1' ? '0' : '1';

}

}

**Файл MainWindow.cpp:**

void MainWindow::sendMessage()

{

.

. // Лабораторная работа №2

.

if(!message.isEmpty())

{

unsigned long bytesSent = 0;

unsigned long countOfFrames = message.length() / DATA\_SIZE;

unsigned long position = 0;

for(size\_t i = 0; i < countOfFrames; ++i)

{

Frame frame(message.mid(position, DATA\_SIZE), port);

QString modifiedFrame = frame.sendFrame(port);

position += DATA\_SIZE;

bytesSent += modifiedFrame.size();

}

if(message.size() % DATA\_SIZE != 0)

{

Frame frame(message.mid(position).leftJustified(DATA\_SIZE, '?') , port);

QString modifiedFrame = frame.sendFrame(port);

bytesSent += modifiedFrame.size();

}

.

. // Лабораторная работа №2

.

}

}

void MainWindow::receiveMessage()

{

.

. // Лабораторная работа №2

.

for(int i = 0; i < listOfFrames.length(); ++i)

{

listOfFrames[i].replace(modifiedFlag, highlightFlag);

QString receivedFrame = frame.getFlag() + listOfFrames[i].mid(0, listOfFrames[i].length() - 4) + " ";

listOfFrames[i].replace(highlightFlag, modifiedFlag);

frame.setFrame(frame.getFlag() + listOfFrames[i]);

frame.setFrame(frame.doDeBitStaffing(frame.getFrame()));

frame.decodeHammingCode();

receivedData += frame.getData().remove('?');

ui->status\_textBrowser->append(receivedFrame + frame.getFcs());

}

ui->output\_textBrowser->append(receivedData);

}