МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Радиофизический факультет

ДОПУСТИТЬ К ЗАЩИТЕ В ГЭК
Руководитель ООП
к.ф.-м.н., доцент
_____ В.А. Мещеряков
«__1__» февраля 2021 г.

ДИПЛОМНАЯ РАБОТА

АППАРАТНАЯ РЕАЛИЗАЦИЯ АЛГОРИТМА ШИФРОВАНИЯ ШИФРА ПЕРЕСТАНОВКИ «МАГИЧЕСКИЙ КВАДРАТ» НА МИКРОКОНТРОЛЛЕРЕ

по основной образовательной программе подготовки специалиста
по специальности 11.05.01 «Радиоэлектронные системы и комплексы»

Шевцова Андрея Александровича

Руководитель ВКР
кандидат техн. наук, доцент
_____С.А. Прокопенко
«__1__» февраля 2021 г.

Автор работы
студент группы № 758
_____ А.А. Шевцов

Томск-2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Радиофизический факультет

УТВЕРЖДАЮ
Руководитель ООП
к.ф.-м.н., доцент
_____ В.А. Мещеряков
«21» сентября 2020 г.

ЗАДАНИЕ
на подготовку ВКР специалиста
студенту Шевцову Андрею Александровичу группы № 758

1. Тема ВКР: Аппаратная реализация алгоритма шифрования шифра перестановки «Магический квадрат» на микроконтроллере.

2. Срок сдачи студентом выполненной ВКР:
а) на кафедре 29.01.2021,
б) в ГЭК 06.02.2021.

3. Краткое содержание работы:

Работа направлена на изучение алгоритма шифрования шифра перестановки «Магический квадрат» и его реализации на микроконтроллере.

4. Календарный график выполнения ВКР:

а) изучение литературы	22.12.2020 – 28.12.2020
б разработка схемы, моделирование работы схемы	24.12.2020 – 28.12.2020
в) сборка макета, проведение экспериментов, анализ результатов	29.12.2020 – 25.01.2021
г) написание ВКР и сдача в ГЭК	12.01.2021– 04.02.2021

5. Дата выдачи задания «1» сентября 2020 г.

Руководитель ВКР –
канд. техн. наук,
доцент кафедры
ИТИДиС _____ С.А. Прокопенко

Задание принял к исполнению _____ А.А. Шевцов

РЕФЕРАТ

Выпускная квалификационная работа содержит: 49 страниц, 46 рисунков, 7 использованных источников, 3 приложения.

МАГИЧЕСКИЙ КВАДРАТ, МИКРОКОНТРОЛЛЕР STM32F100RBT6B, UART, ШИФРОВАНИЕ, PASCAL, COOCOX, ПРОГРАММНЫЙ КОД, КРИПТОГРАФИЯ

Целью работы является реализация на микроконтроллере STM32F100RBT6B алгоритма шифрования информации с помощью магического квадрата.

Для достижения поставленной цели необходимо решить следующие задачи: изучить способы построения магического квадрата, принцип шифрования с помощью магического квадрата, алгоритмизировать метод окаймленных квадратов и принцип шифрования, программно реализовать разработанные алгоритмы на языке Pascal и аппаратно на микроконтроллере STM32F100RBT6B, проверить правильность аппаратной реализации путем сравнения ее работы с программной реализацией.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Защита информации во все времена являлась актуальной задачей. Сейчас ввиду массовой цифровизации криптографические методы защиты информации активно развиваются. Идея криптографических методов заключается в том, чтобы засекречивать информацию так, чтобы только определенный круг лиц мог узнать смысловую составляющую данных. Процесс криптографического преобразования информации в непонятный для посторонних пользователей вид, называется шифрованием. [1, 2]. Современная криптография полностью основана на математике. Основная задача, которую преследует математика в криптографии, – обеспечение криптографической стойкости информации, то есть способность противостоять атакам, направленным на взлом шифра.

В данной работе рассматривается шифр перестановки «Магический квадрат». Построение магического квадрата и метода шифрования алгоритмизируется, алгоритмы программно реализуются на языке Pascal и аппаратно на микроконтроллере STM32F100RBT6B [3]. Правильность аппаратной реализации проверяется путем сравнения результатов ее работы с результатами работы программной реализации.

Выпускная квалификационная работа организована следующим образом.

В разделе 1 приводится классификация исторических шифров.

В разделе 2 проводится обзор методов построения магического квадрата и рассматривается алгоритм шифрования с помощью магического квадрата.

В разделе 3 содержится описание микроконтроллера STM32F100RBT6B и универсального асинхронного приемопередатчика *UART*.

Раздел 4 посвящен программной и аппаратной реализациям алгоритмов построения магического квадрата и шифрования с помощью магического квадрата, а также проверке правильности аппаратной реализации путем сравнения результатов ее работы с результатами работы программной реализации. Коды программ представлены в приложениях.

1 Исторические шифры

За долгое время люди нашли большое количество вариантов обезопасить свои секреты. Например, в Древнем Риме для передачи сообщения брили наголо рабов, татуировали письмо на коже и посылали его к адресату, как только отрастали волосы. Таким образом, защита информации заключалась в том, что о татуировке никто не знал.

Конечно, XXI век требует высоких скоростей шифрования и расшифрования информации и обеспечения высокой криптостойкости шифров. Существует много угроз шифрам, и ко всем есть возможность создать свой вариант защиты. Иногда мы сталкиваемся с шифрованием, даже не задумываясь об этом, например, заходя в электронную почту или на сайт онлайн-банкинга. Встроенное шифрование является популярным в настоящее время.

Исторически сложилось, что все шифры [1, 2] делятся на три класса: замены, перестановки и гаммирования (рисунок 1).

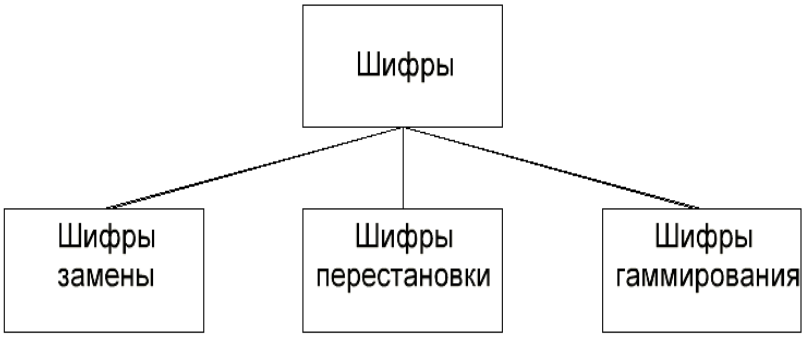

Рисунок 1 – Классы шифрования

Рассмотрим подробнее каждый из классов.

1.1 Шифры замены

Одним из популярных и распространенных видов шифров считаются шифры замены. Их смысл состоит в том, что символы открытого текста заменяются на символы из алфавита шифрованных текстов. Обязательным условием является возможность однозначного расшифрования зашифрованного текста.

Все шифры замены разделяются на шифры простой замены, шифры многоалфавитной замены, шифры полиграммной замены и шифры омофонной замены (рисунок 2).

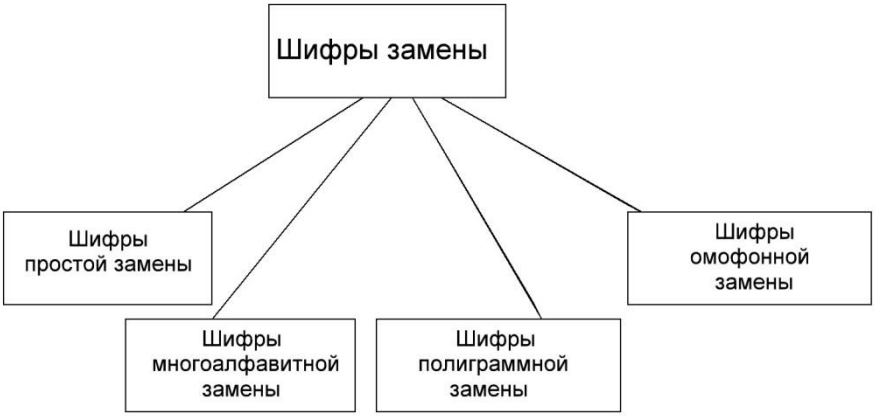

Рисунок 2 – Разновидности шифров замены

В шифрах простой замены каждый символ открытого текста однозначно заменяется одним символом из алфавита шифрованных текстов.

Шифры многоалфавитной замены отличаются от шифров простой замены только тем, что буква открытого текста заменяется буквой шифрованного текста в соответствии с порядком применения соответствующей замены. Соответственно одна и та же буква открытого текста может заменяться различными буквами шифротекста.

В шифрах полиграммной замены имеется возможность заменять не одну букву одной буквой, а так называемые *n*-граммы, т.е. последовательности, состоящие из *n* букв. Другими словами, *n*-грамма открытого текста заменяется *n*-граммой шифрованного текста.

И, наконец, в шифрах омофонной замены алфавит шифрованных текстов разбивается на *n* подмножеств, и каждой букве открытого текста ставится в соответствие свое подмножество. При шифровании буква открытого текста заменяется любой буквой из соответствующего ей подмножества.

Среди представителей шифров замены известны, например, шифр Цезаря, шифр Вижинера, шифр Плейфера, цифирь Петра I.

1.2 Шифры перестановки

В шифрах перестановки буквы переставляются по тексту в соответствии с заданным правилом. Криптостойкость данных шифров зависит от способов создания ключей. У этого вида шифрования есть слабые места, которые могут быть использованы для их дешифрования, например, необходимость разбивать длинные сообщения на блоки, равные размеру ключа; а так же, независимо от подстановки, периодичность встречаемости символов зашифрованного текста совпадает с периодичностью встречаемости символов в открытом тексте.

Шифрами перестановки являются шифр сцитала (шифр Древней Спарты), решетка Кардано и различные вариации шифров табличной перестановки.

1.3 Шифры гаммирования

Гаммирование является широко применяемым криптографическим преобразованием. Под гаммированием понимают процесс наложения ключевой последовательности на открытый текст по определенному правилу. Ключевая последовательность (или гамма) – это псевдослучайная последовательность, выработанная по специальному алгоритму для шифрования данных и их последующего расшифрования. Шифротекст получается сложным для взлома тогда, когда основная последовательность не имеет одинаковых участков. Стойкость шифра зависит от длины ключа. Данный метод становится бессмысленным, когда взломщик знает часть открытого текста и соответствующий ему зашифрованный текст, с помощью этой информации можно восстановить весь открытый текст.

К шифрам гаммирования относится широко известный одноразовый блокнот.

2 Шифр перестановки «Магический квадрат»

Магический квадрат порядка n – это матрица, имеющая n строк и n столбцов, в которую записаны натуральные числа от 1 то n^2, и сумма чисел в каждом столбце, строке и в каждой диагонали равна одному числу, называемому магической константой квадрата. На рисунке 3 приведен пример магического квадрата порядка 3, его магическая константа равна 15.

4	9	2
3	5	7
8	1	6

Рисунок 3 – Магический квадрат порядка 3

Рассмотрим методы построения магического квадрата.

2.1 Методы построения магического квадрата

Особое место в теории магических квадратов занимает разработка методов их построения [3].

2.1.1 Индийский (сиамский) метод

Индийский метод составления магических квадратов является самым древним методом построения магических квадратов произвольного нечетного порядка $n = 2m + 1$.

Правила построения магического квадрата данным следующие. Число 1 вписывается в середину верхней строки. Далее числа вписываются в естественном порядке по восходящей диагонали. Как только число выходит за пределы квадрата, оно переносится в эквивалентную ячейку внутри квадрата. Если число, оказавшееся за пределами квадрата, находится справа от квадрата, то оно записывается в том же горизонтальном ряду со смещением на n ячеек влево (например, число 30 на рисунке 4). Если число оказалось над квадратом, то следует записать его в том же вертикальном ряду, сместив на n ячеек вниз (например, число 31 на рисунке 4). Дойдя до числа kn, кратного порядку квадрата, запишем следующее число снизу от только что записанного числа и снова записываем числа по восходящей диагонали.

На рисунке 4 проиллюстрирован пример построения магического квадрата порядка 7.

	31	40	49	2	11	20	
30	39	48	1	10	19	28	30
38	47	7	9	18	27	29	38
46	6	8	17	26	35	37	46
5	14	16	25	34	36	45	5
13	15	24	33	42	44	4	13
21	23	32	41	43	3	12	21
22	31	40	49	2	11	20	

Рисунок 4 – Построение магического квадрата порядка 7 индийским методом

2.1.2 Метод Москопула

В методе византийского ученого Москопула, как и в индийском методе, указывается некоторый алгоритм последовательного заполнения клеток основного квадрата числами от 1 до n^2 . Порядок заполнения клеток квадрата в данном методе такой же, как порядок обхода шахматной доски конем, двигающимся вверх и направо. Поэтому метод Москопула иногда называется также методом коня. Правила обхода заключаются в следующем.

1. Если порядок квадрата n не кратен 3, то число 1 вписывается в любую ячейку квадрата; иначе число 1 вписывается в середину нижней строки.

2. Последовательно вписываем числа в естественном порядке, двигаясь ходом шахматного коня вверх и направо. Как только число выходит за пределы квадрата, переносим его в эквивалентную ячейку внутри квадрата.

3. Если ячейка, в которую должно быть вписано число z+1, уже занята другим числом, то вписываем число z+1в ячейку, расположенную в том же вертикальном ряду, что и ячейка с числом z, но находящуюся на четыре ячейки выше.

На рисунке 5 приведен пример построения магического квадрата порядка 5 методом Москопула.

			21		
	6				
	12	25	8	16	4
	18		14	22	10
11	24	7	20	3	
17	5	13	21	9	17
23	6	19	2	15	23
4	12	25	8	16	
10	18	1	14	22	

Рисунок 5 – Построение магического квадрата порядка 5 методом Москопула

2.1.3 Метод окаймленных квадратов

Данный метод состоит из следующих этапов.

1. Строим любым известным методом магический квадрат порядка n-2.

2. Увеличиваем все элементы построенного магического квадрата на величину 2(n-1).

3. Помещаем построенный магический квадрат порядка n-2 в матрицу $n \times n$ так, чтобы с каждой стороны квадрата был один свободный столбец (свободная строка).

4. Угловые ячейки матрицы заполняются так: в левую верхнюю ячейку записывается число d-3k-1, в правую верхнюю – число d-k-1, в левую нижнюю – число k+1, в правую нижнюю – число 3k+1, где k=[n/2], d=n^2+1.

5. Свободные ячейки верхней строки матрицы заполняются числами k и {m, d-2k-1-m}, где m=1, 2, … k-1, в произвольном порядке.

6. Свободные ячейки левого столбца матрицы заполняются в произвольном порядке числами d-2k-1 и {k+1+m, d-3k-1-m}, где m=1, 2, … k-1.

7. Свободные ячейки нижней строки матрицы заполняют числами, комплементарными (то есть дополнительными до d) числам, расположенным напротив соответствующей ячейки в верхней строке матрицы. Аналогично заполняются свободные ячейки правого столбца матрицы.

Рассмотрим построение вышеизложенным методом магического квадрата порядка 5. На рисунке 6 представлен магический квадрат порядка 3, полученный после выполнения шага 1.

8	1	6
3	5	7
4	9	2

Рисунок 6 – Магический квадрат порядка, построенный на 1 шаге алгоритма

Магический квадрат, полученный после выполнения шагов 2 и 3, изображен на рисунке 7.

	16	9	14	
	11	13	15	
	12	17	10	

Рисунок 7 – Магический квадрат порядка 3 в матрице 5×5

После выполнения шагов 4-6 магический квадрат имеет вид, как показано на рисунке 8.

19	1	2	20	23
4	16	9	14	
21	11	13	15	
18	12	17	10	
3				7

Рисунок 8 – Магический квадрат с заполненными свободными ячейками

И, наконец, магический квадрат, построенный методом окаймленных квадратов, изображен на рисунке 9.

19	1	2	20	23
4	16	9	14	22
21	11	13	15	5
18	12	17	10	8
3	25	24	6	7

Рисунок 9 – Магический квадрат порядка 5, построенный методом окаймленных квадратов

2.2 Алгоритм шифрования

Для того чтобы зашифровать открытый текст, необходимо его вписать в клетки квадрата слева направо сверху вниз. А зашифрованный текст получается по следующему правилу: выписываем буквы из квадрата в порядке увеличения сопоставленных им цифр. Следует заметить, что если открытый текст имеет длину, большую, чем количество клеток в магическом квадрате, то он делится на блоки длины n^2, и все блоки зашифровываются с помощью одного квадрата по алгоритму, который приведен выше.

Зашифруем, например, слово «экономика», используя магический квадрат, представленный на рисунке 10.

4	9	2
3	5	7
8	1	6

Рисунок 10 – Магический квадрат порядка 3

На рисунке 11 представлен квадрат, заполненный буквами открытого текста.

4/э	9/к	2/о
3/н	5/о	7/м
8/и	1/к	6/а

Рисунок 11 – Магический квадрат, заполненный буквами открытого текста

Выпишем буквы из квадрата: на первой позиции будет буква «к», на второй – буква «о», на третьей – буква «н» и т.д. В итоге получим шифротекст «конэоамик».

3 Обзор технического оборудования для аппаратной реализации шифрования

В данном разделе приведено описание микроконтроллера STM32F100RBT6B [4] и универсального асинхронного приемопередатчика *UART* [4], с помощью которых проводилась аппаратная реализация алгоритмов построения магического квадрата и шифрования открытого текста.

3.1 Микроконтроллер STM32F100RBT6B

Компания STMicroelectronics является одним из основных производителей микроконтроллеров на ядре ARM Cortex-M3.

Микроконтроллер STM32F100RBT6B располагается на отладочной плате STM32VLDISCOVERY (рисунок 12), которая помогает исследовать характеристики микроконтроллеров из серии STM32 Value Line.

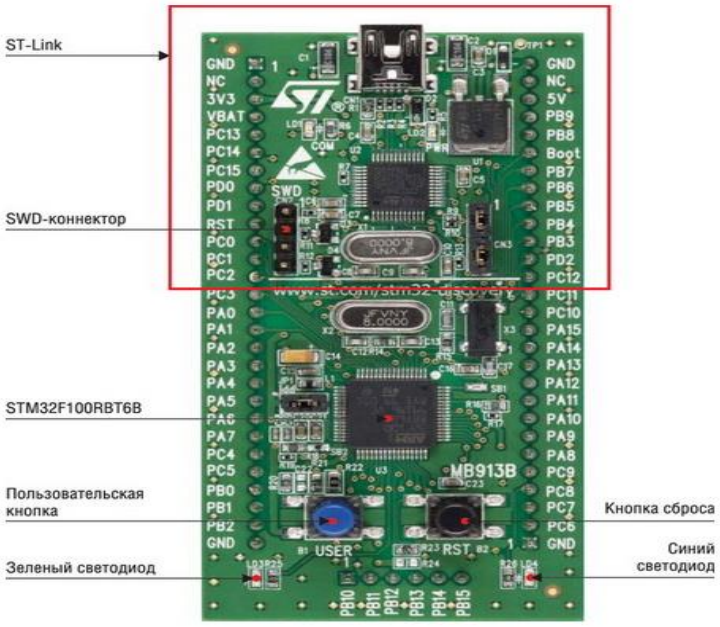

Рисунок 12 – Отладочная плата на базе MCU STM32F100RBT6B

Микроконтроллер STM32F100RBT6B обладает 128 килобайтами Flash памяти и 8 килобайтами оперативной памяти. На плате имеется отладчик/программатор ST-Link, который используется в случае программирования и отладки внутри схемы. Отладочная плата STM32VLDISCOVERY может соединяться с макетными платами и исследовать процессы на периферии микроконтроллера, используя разъем расширения.

Отличительные особенности микроконтроллера STM32F100RBT6B: Flash память 128 Кбайт, 32-битное ядро Cortex-M3RAM 8 КБайт, таймер с расширенными функциями, шесть таймеров общего назначения, коммуникационные интерфейсы, два модуля SPI, два модуля I2C, три модуля USART, двухканальный 12-битный ЦАП, 16-канальный 12-битный АЦП. На

плате имеются внутрисхемный отладчик/программатор ST-Link, интерфейс USB, переключатель для использования платы в качестве отдельного устройства ST-Link, два светодиода индикации состояния, два пользовательских светодиода, пользовательская кнопка, кнопка «Сброс», разъем расширения. Питание возможно от USB интерфейса или от внешнего источника. Все линии ввода/вывода микроконтроллера могут использоваться для подключения к макетной плате или другой отладочной системе

3.2 Универсальный асинхронный приемопередатчик *UART*

Современная цифровая электроника для осуществления взаимодействия отдельных микросхем использует общий протокол связи. За годы существования цифровой техники было разработано множество протоколов, которые можно разделить на параллельные и последовательные. Параллельные интерфейсы передают одновременно (параллельно) несколько битов информации. Для передачи данных такие интерфейсы требуют наличия шин, которые состоят из 8 (рисунок 13), 16 или более проводников.

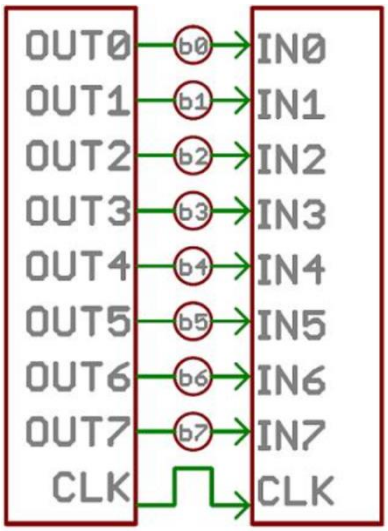

Рисунок 13 –Схема параллельного 8-разрядного интерфейса

Передача данных управляется тактирующим сигналом *CLK*, байт данных передается по каждому импульсу *CLK*, используются 10 проводов.

Последовательный интерфейс (рисунок 14) использует одну сигнальную линию для передачи данных, по которой биты информации последовательно передаются друг за другом. При последовательной передаче сокращается количество сигнальных линий, что уменьшает габариты устройства, упрощает разводку проводников на печатной плате и позволяет делать помехозащищенные интерфейсы. При последовательной передаче каждый информационный бит должен сопровождаться импульсом синхронизации (стробом).

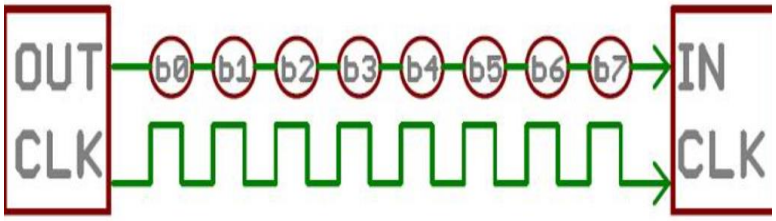

Рисунок 14 –Схема последовательного интерфейса

Если импульсы синхронизации передаются от одного устройства к другому по выделенной линии, то такой интерфейс называют синхронным, в этом случае генератор синхроимпульсов располагается на стороне устройства инициирующего передачу. Если же приемник и передатчик содержат каждый свой генератор синхроимпульсов, работающий на одной частоте, то такой интерфейс называется асинхронным. Получается, что приемник информации сам вырабатывает синхроимпульсы. Этот метод передачи идеально подходит для минимизации количества проводов, но для надежной передачи и приема данных нужно приложить дополнительные усилия.

Асинхронный последовательный протокол имеет ряд правил, которые помогают обеспечить надежную и безошибочную передачу данных, позволяют передавать данные без использования внешнего тактового сигнала. Благодаря сочетанию правил протокол очень гибкий. Для успешной связи нужно убедиться, что оба устройства на шине настроены на использование одинаковых правил.

Типичный представитель асинхронного последовательного интерфейса – универсальный асинхронный приемопередатчик *UART* (*Universal Asynchronous Receiver-Transmitter*) [4]. *UART* – узел вычислительных устройств, предназначенный для организации связи с другими цифровыми устройствами. Он преобразует передаваемые данные в последовательный вид так, чтобы было возможно передать их по одной физической цифровой линии другому аналогичному устройству. Метод преобразования хорошо стандартизован и широко применяется в компьютерной технике.

UART является логической схемой, с одной стороны подключенной к шине вычислительного устройства, а с другой имеющей два или более выводов для внешнего соединения. *UART* может представлять собой отдельную микросхему (рисунок 15) или являться частью большой интегральной схемы (например, микроконтроллера), используется для передачи данных через последовательный порт компьютера, часто встраивается в микроконтроллеры.

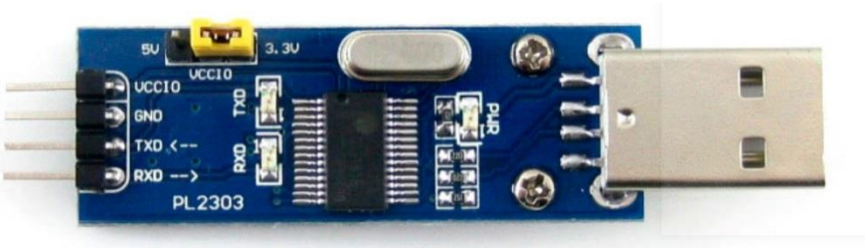

Рисунок 15 – Отдельная микросхема *UART*

Основными свойствами *UART* являются: полнодуплексный обмен по последовательному каналу; синхронный и асинхронный режимы работы; синхронизация как от ведущего, так и от ведомого устройства; скорость передачи может варьироваться в больших пределах; посылка может быть от 5 до 9 разрядов; аппаратная поддержка генерации и контроля бита четности; наличие трех прерываний «передача завершена», «регистр данных передатчика пуст», «прием завершен».

Каждый блок данных (обычно это байт) фактически отправляется в пакете или кадре битов (рисунок 16).

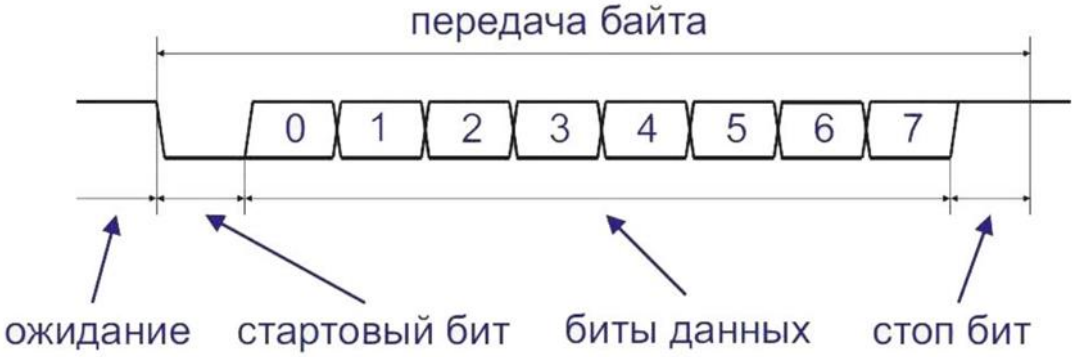

Рисунок 16 – Структура передачи информации

Кадры создаются путем добавления битов синхронизации и битов четности к битам данным. Самая важная часть каждого пакета – это блок данных, так как именно в этом блоке содержатся непосредственно пересылаемые данные. Количество данных в каждом пакете может быть установлено от 5 до 9 битов. Стандартный размер данных – байт, но иногда используются и другие размеры. После согласования длины блока оба устройства на последовательной шине также должны согласовать достоверность своих данных. Если не указано иное, обычно предполагается, что сначала передается младший бит (*LSB*).

Биты синхронизации представляют собой два или три специальных бита, передаваемых с каждым фрагментом данных. Эти биты отмечают начало и конец пакета. Всегда есть только один стартовый бит, но количество стоповых битов настраивается отдельно. Может быть один или два стоп-бита (чаще всего используется один). Старт-бит

всегда определяется линией данных по спаду передаваемого сигнала (переходу от 1 в 0), в то время как стоп-биты определяются по фронту сигнала, то есть по переходу из 0 в 1.

Для проверки ошибок используется контроль четности (нечетности). Чтобы создать бит четности, значения всех 5-9 битов блока данных складываются по модулю 2, а четность суммы определяет, установлен бит четности или нет. Пусть, например, проводится проверка четности. Байт данных в двоичном представлении имеет вид 01011101. Сложив по модулю 2 значения всех битов, получим 1. Поэтому бит проверки четности следует установить в 1. Если проверять нечетность, то соответственно, бит проверки будет сброшен в 0.

Для подключения *UART* к микроконтроллеру, нужно соединить передатчик с приемником крест-накрест, как это показано на рисунке 17.

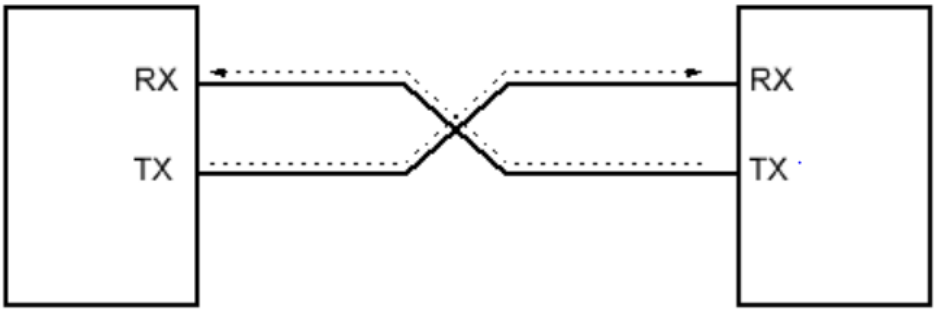

Рисунок 17 –Подключение двух устройств по *UART*

Внутри *UART* находится накопительный сдвиговый регистр, в котором происходит сборка байта из битов и регистры данных *UDR*, куда этот бит передается. Такая структура исключает возможность не до конца считать полученный байт. Буфер приема состоит из двух байтов, что позволяет ему запоминать два байта и еще один помещать в сдвиговый регистр.

Для работы с данным модулем используется программа *Terminal* (рисунок 18).

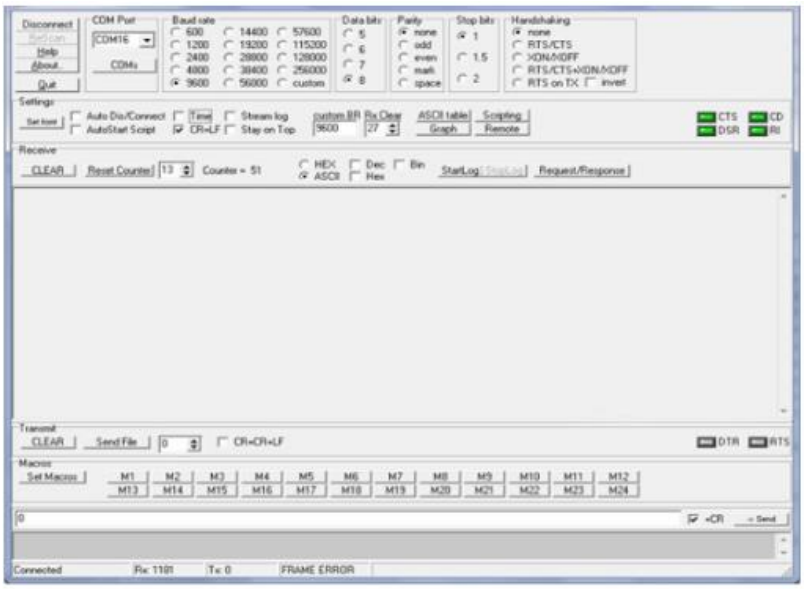

Рисунок 18 – Рабочее окно программы *Terminal*

Информация, которую требуется шифровать (расшифровывать) представлена в *ASCII* коде (*American Standard Code for Information Interchange* – американский стандартный код для обмена информацией). Поскольку первоначально *ASCII* предназначался для обмена информации, в нем, кроме информационных символов, используются символы-команды для управления связью. На рисунке 19 такие символы показаны многоточием.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
1.	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
2.		!	"	#	$	%	&	'	(	)	*	+	,	—	.	/
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4.	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5.	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6.	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7.	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	…

Рисунок 19 – *ASCII* код

Таблица *ASCII* кодов содержит 8 строк и 16 столбцов, каждая строка и столбец пронумерованы в шестнадцатеричной системе счисления. Шестнадцатеричное представление *ASCII* кода складывается из номера столбца и номера строки, в которых располагается символ, при этом номер строки образуют первую цифру (старшие четыре

бита), а номер столбца вторую цифру (младшие 4 бита). Так, например, *ASCII* код символа «*E*» есть число 0x45, а символа «\» – 0x5*C*.

Легко заметить, что в таблице *ASCII* кодов представлено 128 символов, один символ кодируется байтом – восемью битами. Дело в том, что верхние значения (128–255) могут занимать различные дополнительные символы, например, буквы русского алфавита, это зависит от конкретного типа кодировки.

4 Программная и аппаратная реализация шифрования с использованием магического квадрата

В данной работе программная реализация алгоритмов построения магического квадрата и шифрования осуществлена на языке программирования Pascal, а аппаратная реализация – на микроконтроллере STM32F100RBT6B.

4.1 Программная реализация

В данном разделе рассматривается программная реализация алгоритма построения магического квадрата и алгоритма шифрования на языке программирования Pascal. Для программной реализации выбран метод окаймленных квадратов, так как он позволяет строить различные квадраты одного порядка.

Блок-схема алгоритма построения магического квадрата приведена на рисунке 20. На вход алгоритма поступает начальный порядок квадрата 3, и порядок квадрата, который необходимо построить, на выходе алгоритма – магический квадрат необходимого порядка [6].

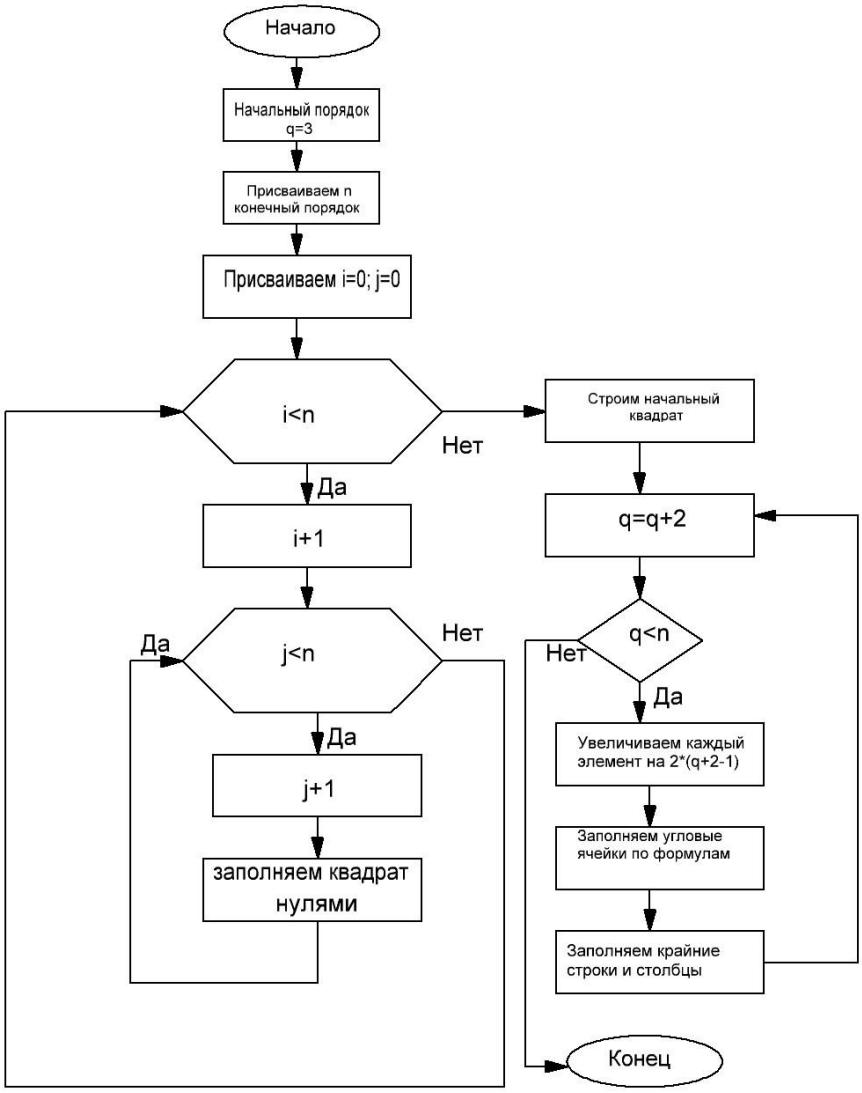

Рисунок 20 – Блок схема алгоритма построения магического квадрата

Текст программы построения магического квадрата представлен в приложении А.

Блок-схема алгоритма шифрования представлена на рисунке 21. На вход алгоритма поступает массив с открытым текстом, а на выходе получается массив с зашифрованным текстом.

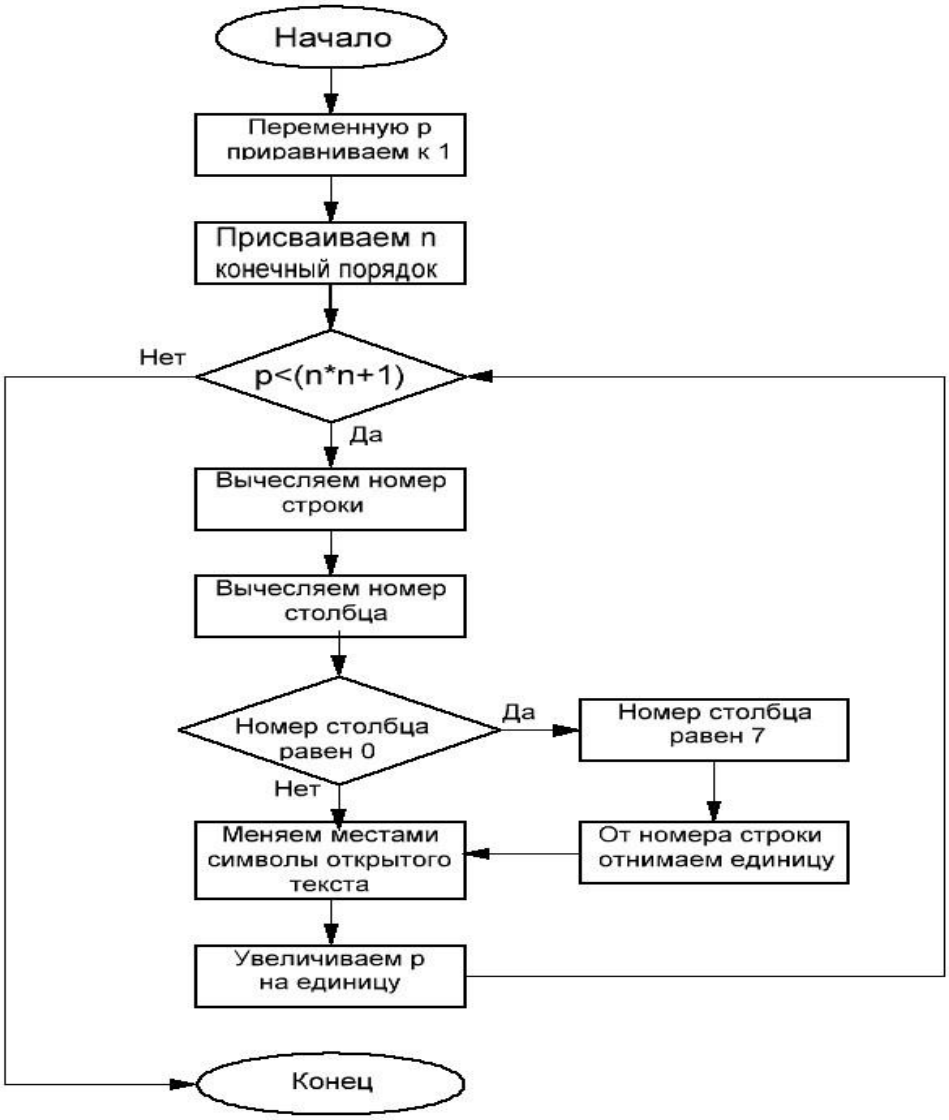

Рисунок 21 – Блок схема алгоритма шифрования

Программная реализация алгоритма шифрования заключается в том, что программа считывает из файла открытый текст (пример файла с открытым текстом представлен на рисунке 22), разбивает его на равные блоки длинной n^2, где n – порядок магического квадрата, и записывает открытый текст в магический квадрат сверху вниз слева направо. После этого программа формирует зашифрованный текст, выписывая буквы в порядке увеличения соответствующих номеров в клетках квадрата, и сохраняет его в отдельный файл [5]. Пример файла с шифрованным текстом изображен на рисунке 23.

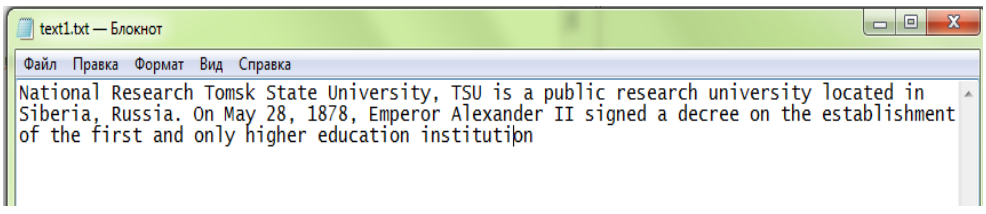

Рисунок 22 – Открытый текст в файле

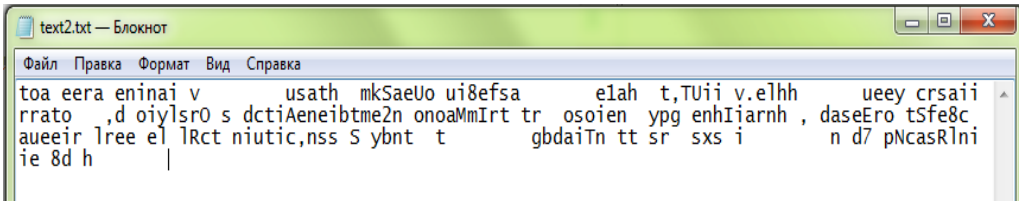

Рисунок 23 – Зашифрованный текст в файле

Текст программной реализации алгоритма шифрования представлен в приложении Б.

4.2 Аппаратная реализация

В данном разделе рассмотрим, каким образом реализуется магический квадрат на микроконтроллере STM32F100RBT6B.

В процессе работы составлена схема экспериментальной установки (рисунок 24).

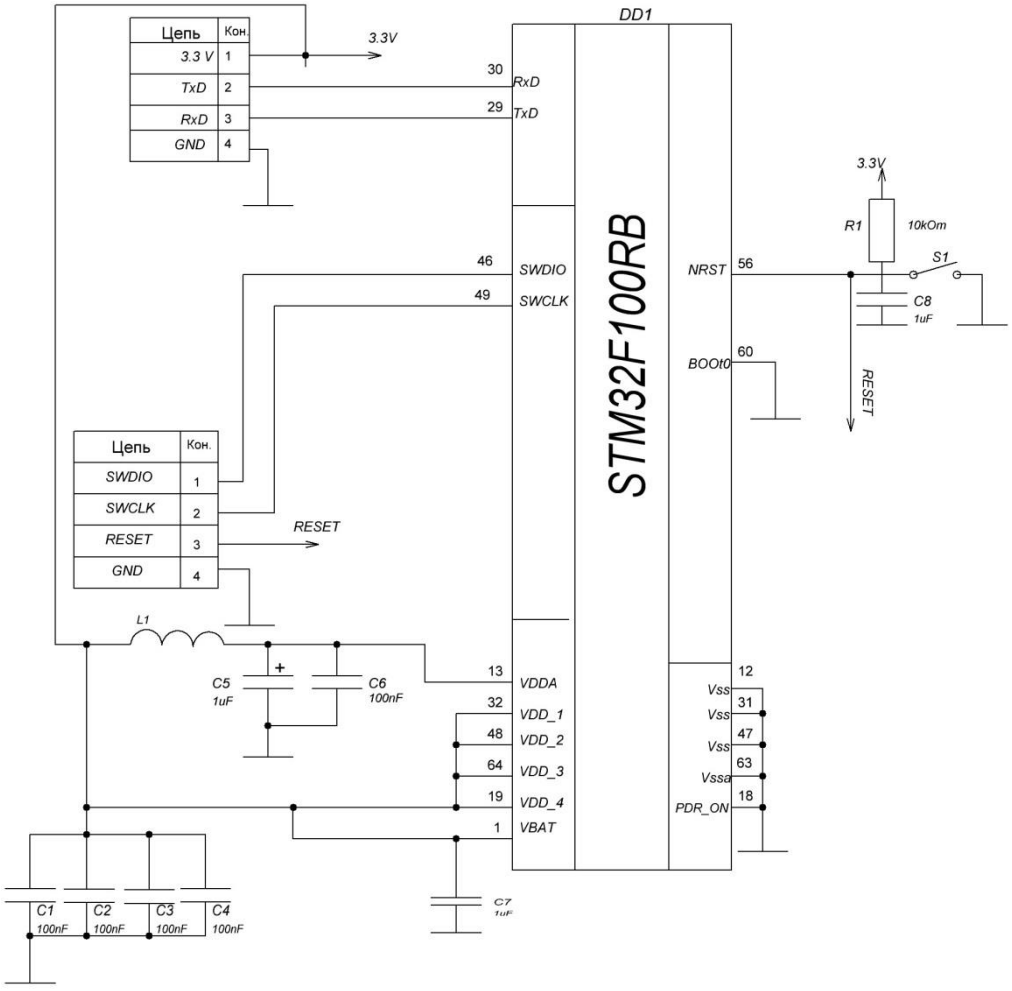

Рисунок 24 – Чертеж электрической схемы в программе Splan

С помощью данной схемы собрана непосредственно сама экспериментальная установка, которая представлена на рисунке 25.

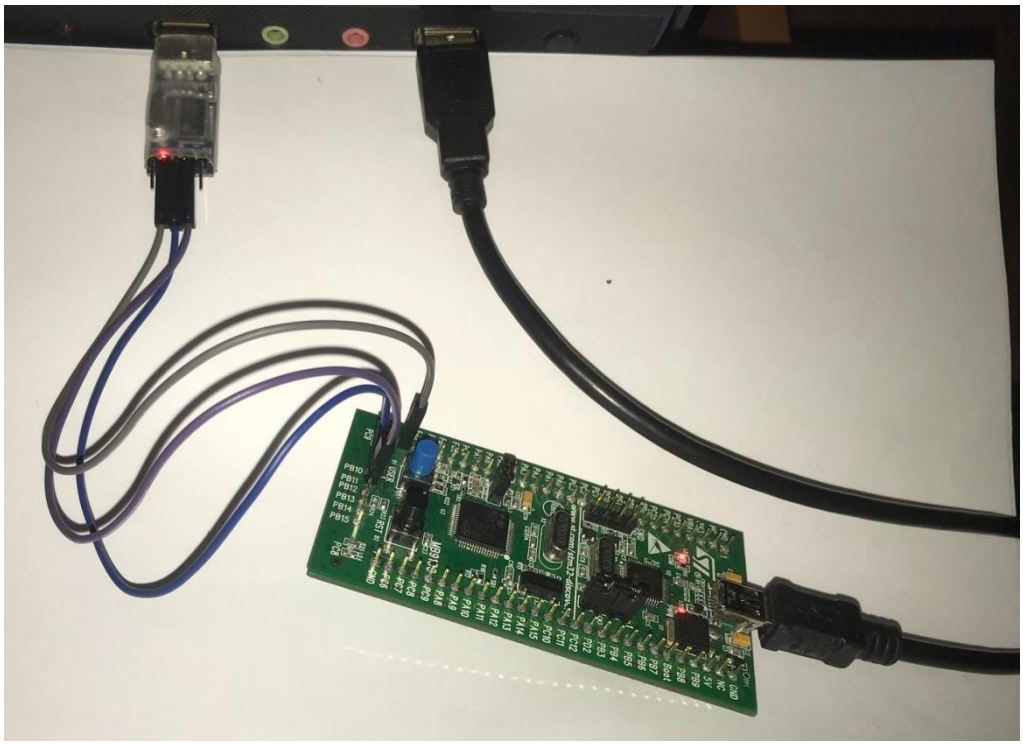

Рисунок 25 – Экспериментальная установка

4.2.1 Программная среда CooCox CoIDE

Для разработки кода микроконтроллера архитектуры ARM использовалась высокоинтегрированная программная среда CooCox CoIDE [7].

CooCox CoIDE является одним из самых простых и быстрых в плане установки, освоения и настройки решений, позволяющих работать с ним даже начинающим пользователям. Успешный старт первых проектов обеспечивает мастер, помогающий пройти через все основные этапы разработки путем ответов на простые вопросы. Качественно сделанная среда CooCox CoIDE позволяет загружать исходный код программы, редактировать его, проводить компиляцию (сторонними средствами), прошивать микроконтроллер и проводить отладку.

На рисунке 26 представлен интерфейс программы CooCox CoIDE.

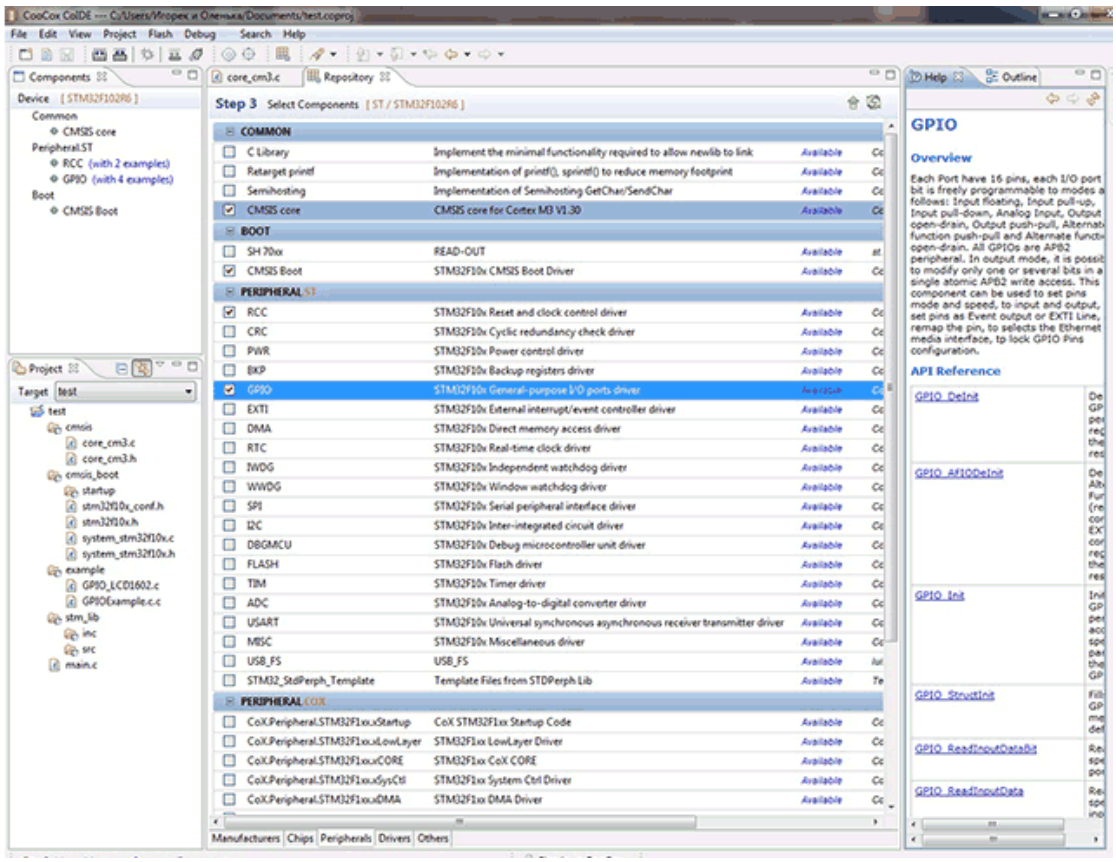

Рисунок 26 – Интерфейс программы CooCox CoIDE

Программа разработана на базе Eclipse и имеет все ее достоинства. Редактор кода включает в себя подсветку синтаксиса и всплывающие подсказки. Кроме того, присутствуют функции глобальной замены переменной и предложения вариантов окончания кода. Среда Eclipse поддерживает микроконтроллеры серии ST, а также ряд других семейств: Atmel, Holtek, Freescale, Nuvoton, NXP, Energy Micro, Texas Instruments и некоторые другие. Список чипов постоянно увеличивается с каждой версией программы. Встроенный отладчик ST-Link поддерживает все основные режимы отладки. Данная среда разработки совершенно бесплатна и имеет открытый код.

4.2.2 Реализация алгоритма построения магического квадрата на микроконтроллере

Изучив язык программирования Си, был написан программный код построения магического квадрата. С помощью среды программирования CooCox CoIDE код загружен в микроконтроллер STM32F100RBT6B.

Программный код начинается с инициализации переменных (рисунок 27).

```
int mas2[300],mas3[300];//максимальный размер зашифрованного и расшифрованного текста в символах

int mas1[300];//максимальный размер открытого текста в символах
int a[17][17];//задаём порядок квадрата
int a2[17][17];//порядок квадрата
int i,z,t,ii,jj,j,m,o,o1,o2,p,k,d,q,www,u,RXc,a1; //обьявление переменных
```

Рисунок 27 – Инициализация переменных

Затем идет подпрограмма настройки портов (рисунок 28).

```
//подпрограмма настройки портов
void port(void)
{
GPIO_InitTypeDef GPIO_InitStruct; //обьявление структуры настройки портов

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_8|GPIO_Pin_9; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим - вывод, тип - push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение обьявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_10; //выбор настраиваемых выводов - Tx
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение аксимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_AF_PP; //режим - вывод,
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение обьявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_11; //выбор настраиваемых выводов - Rx
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим - дифференциальный вход
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение обьявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим - дифференциальный вход
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение обьявленной структуры
GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0); //назначение вывода 0 порта A источником внешних прерываний
}
```

Рисунок 28 – Подпрограмма настройки портов

Далее написан код построения магического квадрата. Фрагмент программного кода представлен на рисунке 29.

```
        //угловые ячейки
        a[(n - q)/2 - 1][(n - q)/2 - 1] = a[(n - q)/2 - 1][(n - q)/2 -
1] + d - k - k - k - 1; // верхний левый
        a[(n - q)/2 - 1][(n - q)/2 + q] = a[(n - q)/2 - 1][(n - q)/2 + q] +
d - k - 1; // верхний правый
        a[(n - q)/2 + q][(n - q)/2 - 1] = a[(n - q)/2 + q][(n - q)/2 - 1] +
k + 1; // нижний левый
        a[(n - q)/2 + q][(n - q)/2 + q] = a[(n - q)/2 + q][(n - q)/2 +
q] + k + k + k + 1; // нижний правый


        //верхняя строка

        a[IndexI(n, q + 2, 0)][IndexJ(n, q + 2, 1)] = k;
        for(m = 1 ; m <= k - 1; ++m)
        {
            a[IndexI(n, q + 2, 0)][IndexJ(n, q + 2, 1 + 2*m)] = d - 2*k - 1
- m;
            a[IndexI(n, q + 2, 0)][IndexJ(n, q + 2, 2*m)] = m;
        }


       //левый столб!!!!!!!
        a[IndexI(n, q + 2, 1)][IndexJ(n, q + 2, 0)] = d - 2*k - 1;
        for(m = 1 ; m <= k - 1; ++m)
        {
             a[IndexI(n, q + 2, 1 + 2*m)][IndexJ(n, q + 2, 0)] = d -
3*k - 1 - m;
             a[IndexI(n, q + 2, 2*m)][IndexJ(n, q + 2, 0)] = m + 1 +
k;
        }


         //правый столбец!!!!!!!!

        for (i = 0 ; i< q  ; ++i )
            a[IndexI(n, q + 2, 1 + i)][IndexJ(n, q + 2, q+1 )] = d -
a[IndexI(n, q + 2, i +1)][IndexJ(n, q + 2, 0)];


        // нижняя строка!!!!!!!!!
        for (j = 0; j < q; ++j )
            a[IndexI(n, q, q )][IndexJ(n, q, j)]= d - a[IndexI(n, q,
-1)][IndexJ(n, q, j)];
        q = q + 2;
```

Рисунок 29 – Фрагмент программного кода построения магического квадрата.

На рисунках 30-32 представлены квадраты порядков 7, 11 и 17 соответственно. При этом построение начиналось с использования изначального общего магического квадрата порядка 3.

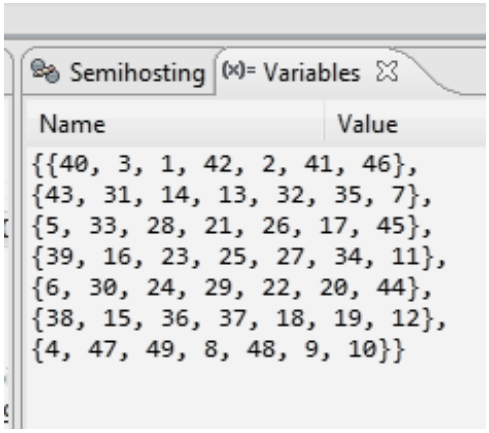

Рисунок 30 – Результат аппаратной реализации магического квадрата порядка 7 на микроконтроллере STM32F100RBT6B

```
{{106, 5, 1, 110, 2, 109, 3, 108, 4, 107, 116},
{111, 89, 24, 21, 92, 22, 91, 23, 90, 97, 11},
{7, 93, 76, 39, 37, 78, 38, 77, 82, 29, 115},
{105, 26, 79, 67, 50, 49, 68, 71, 43, 96, 17},
{8, 88, 41, 69, 64, 57, 62, 53, 81, 34, 114},
{104, 27, 75, 52, 59, 61, 63, 70, 47, 95, 18},
{9, 87, 42, 66, 60, 65, 58, 56, 80, 35, 113},
{103, 28, 74, 51, 72, 73, 54, 55, 48, 94, 19},
{10, 86, 40, 83, 85, 44, 84, 45, 46, 36, 112},
{102, 25, 98, 101, 30, 100, 31, 99, 32, 33, 20},
{6, 117, 121, 12, 120, 13, 119, 14, 118, 15, 16}}
```

Рисунок 31 – Результат аппаратной реализации магического квадрата порядка 11 на микроконтроллере STM32F100RBT6B

```
{{265, 8, 1, 272, 2, 271, 3, 270, 4, 269, 5, 268, 6, 267, 7, 266, 281},
{273, 236, 39, 33, 242, 34, 241, 35, 240, 36, 239, 37, 238, 38, 237, 250, 17},
{10, 243, 211, 66, 61, 216, 62, 215, 63, 214, 64, 213, 65, 212, 223, 47, 280},
{264, 41, 217, 190, 89, 85, 194, 86, 193, 87, 192, 88, 191, 200, 73, 249, 26},
{11, 235, 68, 195, 173, 108, 105, 176, 106, 175, 107, 174, 181, 95, 222, 55, 279},
{263, 42, 210, 91, 177, 160, 123, 121, 162, 122, 161, 166, 113, 199, 80, 248, 27},
{12, 234, 69, 189, 110, 163, 151, 134, 133, 152, 155, 127, 180, 101, 221, 56, 278},
{262, 43, 209, 92, 172, 125, 153, 148, 141, 146, 137, 165, 118, 198, 81, 247, 28},
{13, 233, 70, 188, 111, 159, 136, 143, 145, 147, 154, 131, 179, 102, 220, 57, 277},
{261, 44, 208, 93, 171, 126, 150, 144, 149, 142, 140, 164, 119, 197, 82, 246, 29},
{14, 232, 71, 187, 112, 158, 135, 156, 157, 138, 139, 132, 178, 103, 219, 58, 276},
{260, 45, 207, 94, 170, 124, 167, 169, 128, 168, 129, 130, 120, 196, 83, 245, 30},
{15, 231, 72, 186, 109, 182, 185, 114, 184, 115, 183, 116, 117, 104, 218, 59, 275},
{259, 46, 206, 90, 201, 205, 96, 204, 97, 203, 98, 202, 99, 100, 84, 244, 31},
{16, 230, 67, 224, 229, 74, 228, 75, 227, 76, 226, 77, 225, 78, 79, 60, 274},
{258, 40, 251, 257, 48, 256, 49, 255, 50, 254, 51, 253, 52, 252, 53, 54, 32},
{9, 282, 289, 18, 288, 19, 287, 20, 286, 21, 285, 22, 284, 23, 283, 24, 25}}
```

Рисунок 32 – Результат аппаратной реализации магического квадрата порядка 17 на микроконтроллере STM32F100RBT6B

Правильность работы аппаратной реализации алгоритма построения магического квадрата проверялась путем сравнения результатов ее работы с результатами программной реализации построения магического квадрата (рисунки 33-35).

```
40   3   1  42   2  41  46
43  31  14  13  32  35   7
 5  33  28  21  26  17  45
39  16  23  25  27  34  11
 6  30  24  29  22  20  44
38  15  36  37  18  19  12
 4  47  49   8  48   9  10
```

Рисунок 33 – Магический квадрат порядка 7, построенный программной реализацией

```
106   5   1 110   2 109   3 108   4 107 116
111  89  24  21  92  22  91  23  90  97  11
  7  93  76  39  37  78  38  77  82  29 115
105  26  79  67  50  49  68  71  43  96  17
  8  88  41  69  64  57  62  53  81  34 114
104  27  75  52  59  61  63  70  47  95  18
  9  87  42  66  60  65  58  56  80  35 113
103  28  74  51  72  73  54  55  48  94  19
 10  86  40  83  85  44  84  45  46  36 112
102  25  98 101  30 100  31  99  32  33  20
  6 117 121  12 120  13 119  14 118  15  16
```

Рисунок 34 – Магический квадрат порядка 11, построенный программной реализацией

```
265   8   1 272   2 271   3 270   4 269   5 268   6 267   7 266 281
273 236  39  33 242  34 241  35 240  36 239  37 238  38 237 250  17
 10 243 211  66  61 216  62 215  63 214  64 213  65 212 223  47 280
264  41 217 190  89  85 194  86 193  87 192  88 191 200  73 249  26
 11 235  68 195 173 108 105 176 106 175 107 174 181  95 222  55 279
263  42 210  91 177 160 123 121 162 122 161 166 113 199  80 248  27
 12 234  69 189 110 163 151 134 133 152 155 127 180 101 221  56 278
262  43 209  92 172 125 153 148 141 146 137 165 118 198  81 247  28
 13 233  70 188 111 159 136 143 145 147 154 131 179 102 220  57 277
261  44 208  93 171 126 150 144 149 142 140 164 119 197  82 246  29
 14 232  71 187 112 158 135 156 157 138 139 132 178 103 219  58 276
260  45 207  94 170 124 167 169 128 168 129 130 120 196  83 245  30
 15 231  72 186 109 182 185 114 184 115 183 116 117 104 218  59 275
259  46 206  90 201 205  96 204  97 203  98 202  99 100  84 244  31
 16 230  67 224 229  74 228  75 227  76 226  77 225  78  79  60 274
258  40 251 257  48 256  49 255  50 254  51 253  52 252  53  54  32
  9 282 289  18 288  19 287  20 286  21 285  22 284  23 283  24  25
```

Рисунок 35 – Магический квадрат порядка 17, построенный программной реализацией

Как видно из представленных рисунков, магические квадраты одинаковых порядков совпадают. Следовательно, аппаратная реализация алгоритма построения магического квадрата выполнена корректно.

4.2.3 Реализация алгоритма шифрования с помощью магического квадрата на микроконтроллере

Для того чтобы отправить текст из компьютера в память микроконтроллера, к последнему подключен универсальный асинхронный приемопередатчик *UART*. Работа с данным устройством осуществлялась в программе *Terminal*. Для согласования микроконтроллера и *UART* были заданы настройки, представленные на рисунках 36 и 37. Передатчик и приемник связаны с помощью соединительных проводов по схеме, представленной на рисунке 17.

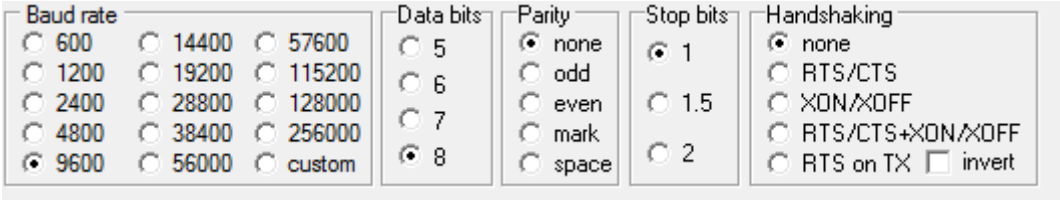

Рисунок 36 – Настройка параметров в программе *Terminal*

```
362    USART_InitTypeDef USART_InitStructure; //объявление структуры настройки USART
363    USART_InitStructure.USART_BaudRate = 9600; //скорость
364    USART_InitStructure.USART_WordLength = USART_WordLength_8b;//8 бит данных
365    USART_InitStructure.USART_StopBits = USART_StopBits_1;//один стоп бит
366    USART_InitStructure.USART_Parity = USART_Parity_No; //четность - нет
367    USART_InitStructure.USART_HardwareFlowControl =
```

Рисунок 37 – Настройка параметров в программе CooCox CoIDE

Идея аппаратной реализации шифрования заключается в том, что открытый текст в программе *Terminal* считывается из файла (рисунок 38), затем передается в микроконтроллер (рисунок 39), разбивается на блоки длины n^2 и записывается в элементы магического квадрата слева направо сверху вниз. Зашифрованный текст получается путем выписывания букв из магического квадрата в порядке увеличения сопоставленных им чисел. После шифрования текст отправляется в программу *Terminal* (рисунок 40) и выводится в ее окне (рисунок 41). Зашифрованный текст, представленный на рисунке 41, получен с помощью квадрата порядка 11, изображенного на рисунке 34.

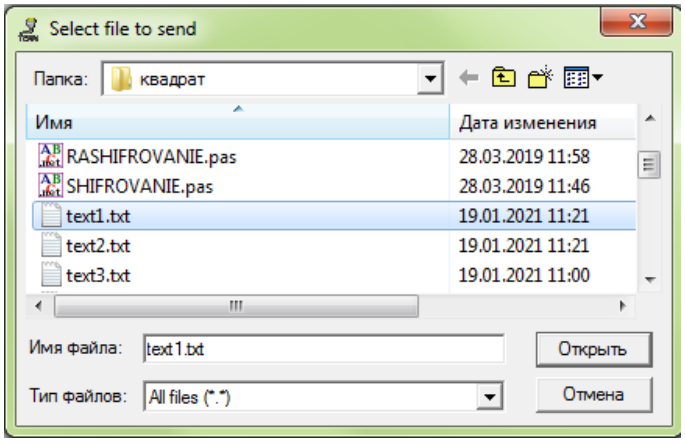

Рисунок 38 – Отправка открытого текста из файла в *UART*

```
//подпрограмма отправки 1 байта по UART
void send_to_uart(uint8_t data)
{
while(!(USART3->SR & USART_SR_TC)); //при равенстве 1 бита TC в регистре SR
USART3->DR=data; //отправка байта через UART
}

void USART3_IRQHandler(void)
{

        RXc = USART_ReceiveData(USART3);
        if (RXc==0x0D)
        {
            www=1;
            //USART_Cmd(USART3, DISABLE); //включение USART3
        }
        else
        {
            mas1[o]=RXc;
            o++;
        }
}

//Обработчик прерывания EXTI0
```

Рисунок 39 – Подпрограмма записи открытого текста в память микроконтроллера

```
   // отправка открытого текста в uart
         for(i=0;i<o;i++)
        {
            send_to_uart(mas1[i]);
        }
        send_to_uart('  ');
        //send_to_uart('    ');

 // построение квадрата
        square_design();

//шифрование
        u=o/289;
        u=1;
        for (z=0;z<u;z++)
        {
        shifr();
        }
        for(i=0;i<o;i++)
        {
            send_to_uart(mas2[i]);
        }

        send_to_uart('  ');
        send_to_uart('  ');
```

Рисунок 40 – Подпрограмма отправки зашифрованного текста в *UART*

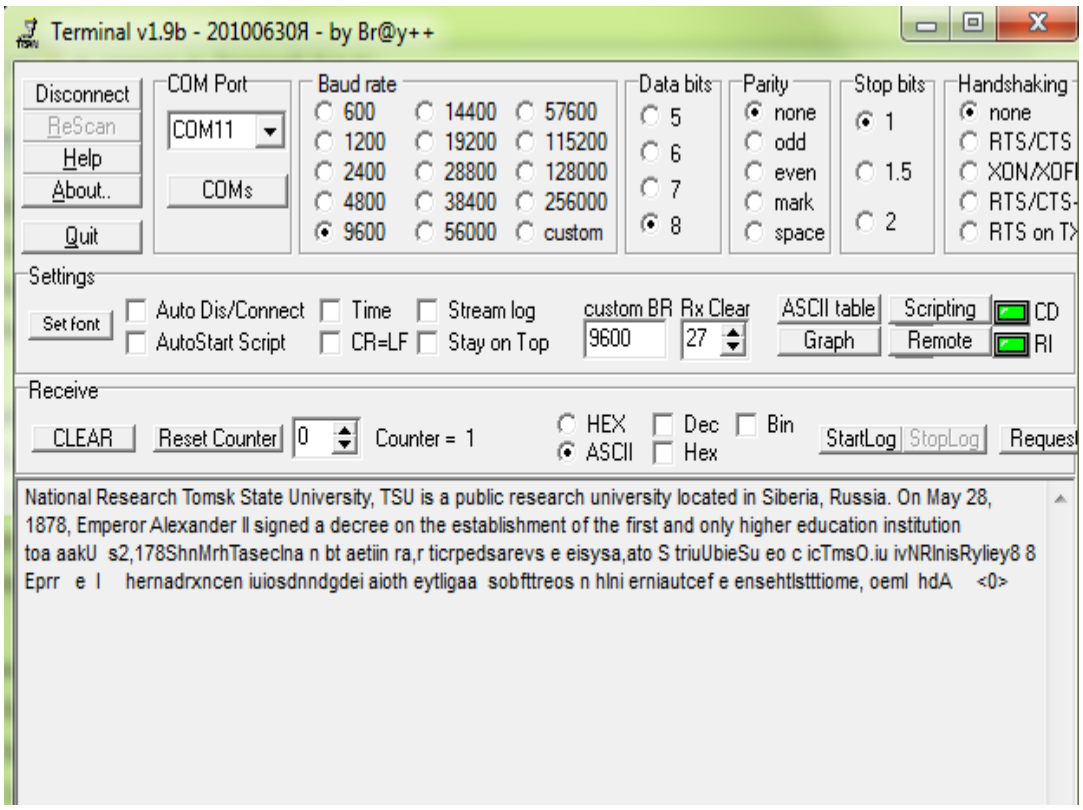

Рисунок 41 – Открытый и зашифрованный текст в окне программы *Terminal*

Правильность работы аппаратной реализации алгоритма шифрования с помощью магического квадрата проверялась путем сравнения результата ее работы с результатом работы программной реализации данного алгоритма на языке Pascal. На рисунке 42 приведен файл с открытым текстом, а на рисунке 43 – файл с зашифрованным текстом, полученным в программе на языке Pascal.

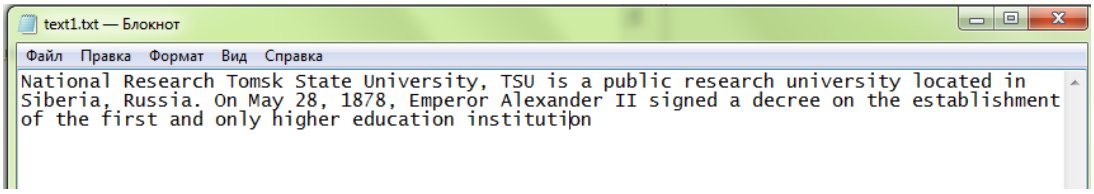

Рисунок 42 – Открытый текст

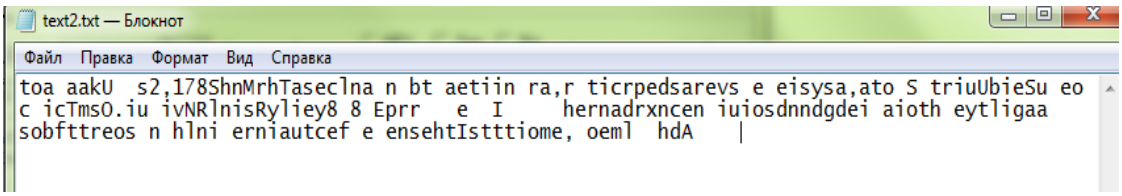

Рисунок 43 – Зашифрованный в программе Pascal текст

Как видно из рисунков 41 и 43, шифротексты являются одинаковыми. Следовательно, аппаратная реализация алгоритма шифрования выполнена корректно.

Затем тот же открытый текст был зашифрован с помощью магического квадрата порядка 17. На рисунке 44 показан результат аппаратной реализации алгоритма шифрования.

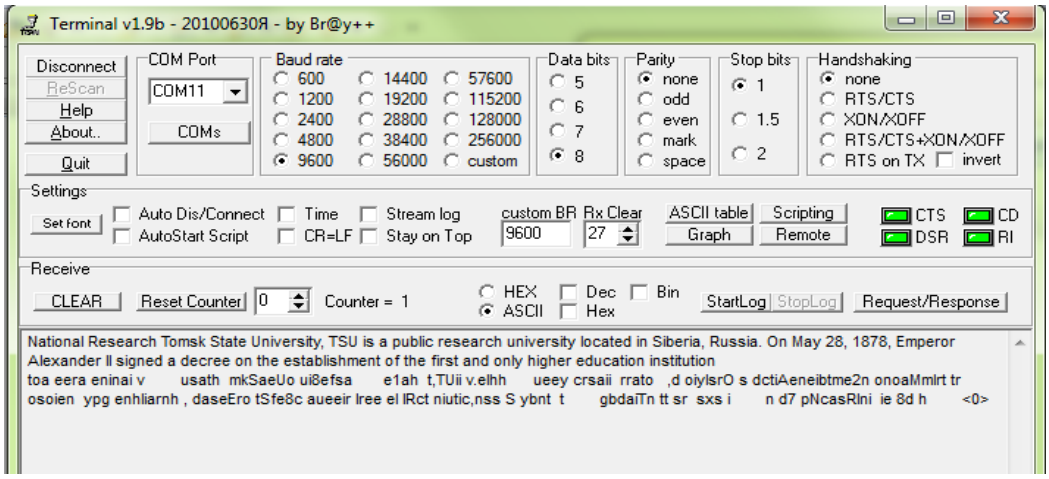

Рисунок 44 – Открытый и зашифрованный текст с использованием квадрата порядка 17

Нетрудно убедиться, что открытому тексту, зашифрованному с помощью магических квадратов различных порядков, соответствуют разные шифротексты.

Код подпрограммы алгоритма шифрования представлен на рисунке 45.

```
int shifr(void)
{
p=1;
n=17;
t=0;
o1=o;
u=o/289;

    while (p < (n*n+1))
    {
    ii=p/17+1;
    jj=p%17;
        if (jj==0)
        {
            jj=17;
            ii=ii-1;
        }
    mas2[a2[ii-1][jj-1]-1+z*289]=mas1[p-1+z*289] ;
    p++;
    }

o=o1;

}
```

Рисунок 45 – Код подпрограммы алгоритма шифрования

В результате работы аппаратно реализовано построение магического квадрата необходимого порядка и шифрование/расшифрование текста. Блок-схема программы на языке Си представлена на рисунке 46. На вход программы поступает порядок магического квадрата и массив с открытым текстом, на выходе получаем массивы шифрованного и расшифрованного текстов. Функция Port выполняет настройку портов. Функция Exit выполняет настройку внешних прерываний. Функция square_design выполняет построение магического квадрата необходимого порядка. Функция shifr выполняет шифрование открытого текста. Функция rashifr выполняет расшифрование шифрованного текста.

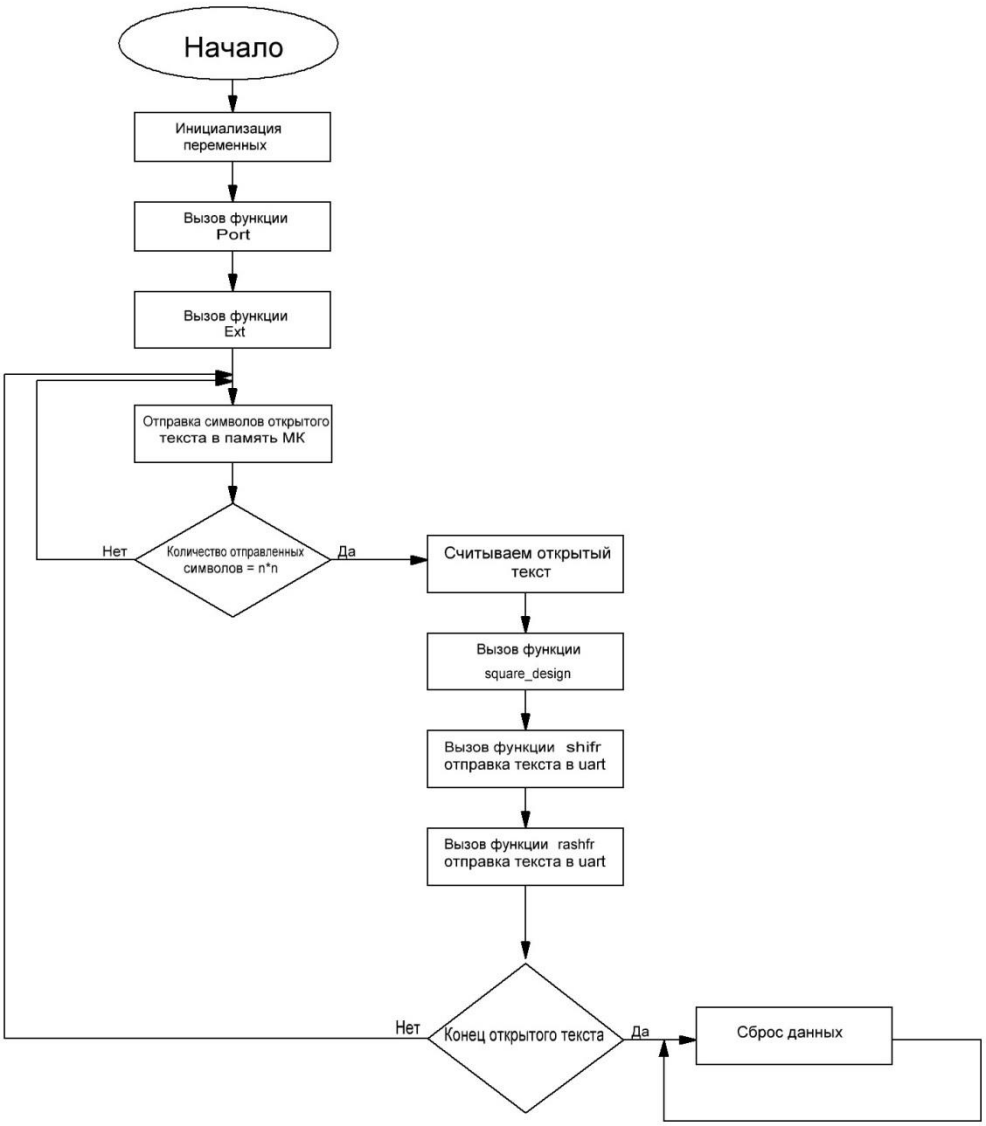

Рисунок 46 – Общая блок-схема программы

Текст итоговой программы представлен в приложении В.

ЗАКЛЮЧЕНИЕ

В процессе выполнения выпускной квалификационной работы изучены методы построения магического квадрата, который позволяет реализовать одноименный шифр перестановки; принцип шифрования с использованием магического квадрата. Для реализации шифра «Магический квадрат» выбран метод окаймленных квадратов, так как данный метод позволяет строить разные магические квадраты одного порядка. Поскольку магический квадрат является ключом шифрования, то данный метод позволяет увеличить количество ключей одной длины. Метод построения магического квадрата методом окаймленных квадратов алгоритмизирован.

Алгоритм построения магического квадрата и алгоритм шифрования открытого текста программно реализованы на языке Pascal и аппаратно реализованы на микроконтроллере STM32F100RBT6B.

Правильность работы аппаратной реализации проверена на тестовых примерах путем сравнения результатов работы аппаратной реализации с результатами работы программы на Pascal. В ходе тестирования установлено, что аппаратная реализация работает корректно и отвечает разработанной спецификации. Проводились эксперименты по шифрованию одного открытого текста с использованием магических квадратов различных порядков. Полученные в каждом случае шифротексты оказываются различными, но при расшифровании получился один текст.

Кроме того, выяснилось, что технические возможности микроконтроллера STM32F100RBT6B позволяют обрабатывать не более трехсот символов открытого текста. Если размер открытого текста превышает данный лимит символов, то программный код, созданный в среде Coo Cox, не запускается.

Результаты работы будут использованы в учебном процессе в курсе «Защита информации» для проверки правильности студенческих реализаций алгоритмов построения магического квадрата и шифрования.

Результаты работы были доложены на двух всероссийских конференциях студенческих научно-исследовательских инкубаторов [5, 6] и опубликованы в изданиях, входящих в базу цитирования РИНЦ.

ЛИТЕРАТУРА

1 Шнайер Б. Прикладная криптография: протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер – М. : Триумф, 2002. – 815 с.

2 Шнайер Б. Секреты и ложь: безопасность данных в цифровом мире / Б. Шнайер – СПб. : Питер, 2003. – 367 с.

3 Простые числа. – URL: http://e-science.ru/sites/default/files/upload_forums_files/8q/prost2.pdf. (Дата обращения: 24.04.18).

4 Практическое руководство по программированию STM-микроконтроллеров: учебное пособие / Торгаев С.Н., Тригуб М.В., Мусоров И.С., Чертихина Д.С.; Томский политехнический университет. – Изд-во Томского политехнического университета, 2015. – 111 с. – Режим доступа: https://portal.tpu.ru:7777/SHARED/t/TORGAEV/ academic/Tab4/ Posobie3.pdf.

5 Сапсай М.Е., Шевцов А.А., Программная реализация шифра перестановки «Магический квадрат» // Труды пятнадцатой всероссийской конференции студенческих научно-исследовательских инкубаторов. Под редакцией В.В. Демина. 2018. С. 92-94.

6 Сапсай М.Е., Шевцов А.А., Аппаратная реализация алгоритма построения магического квадрата // Труды шестнадцатой всероссийской конференции студенческих научно-исследовательских инкубаторов. Под редакцией В.В. Демина. 2019. С. 191-193.

7 Основы программирования на языке СИ: учебное пособие / Солдатов А.И., Костина М.А., Лежнина И.А., Торгаев С.Н., Громов М.Л., Хан В.; Томск: ТУСУР: 2018. – 122 с. – Режим доступа: https://edu.tusur.ru/publications/8872.

ПРИЛОЖЕНИЕ А

Программная реализация алгоритма построения магического квадрата

```
program one;

function IndexI (n, q, i: integer):longint;
begin
  IndexI := ((n - q) div 2) + i;
end;

function IndexJ (n, q, j: integer):longint;
begin
  IndexJ := ((n - q) div 2) + j;
end;

type
matrix = array[1..100,1..100] of integer ;
var
a:matrix;
i,j,n,m,k,d,q,www:integer;
begin
 q:=3;

 n:=55;
 for i:=1 to n do for j:=1 to n do a[i,j]:=0;

 // строчка 1
 a[IndexI(n, 3, 1),IndexJ(n, 3, 1)] := 8;
 a[IndexI(n, 3, 1),IndexJ(n, 3, 2)] := 1;
 a[IndexI(n, 3, 1),IndexJ(n, 3, 3)] := 6;

 // строчка 2
 a[IndexI(n, 3, 2),IndexJ(n, 3, 1)] := 3;
 a[IndexI(n, 3, 2),IndexJ(n, 3, 2)] := 5;
 a[IndexI(n, 3, 2),IndexJ(n, 3, 3)] := 7;

 // строчка 3
 a[IndexI(n, 3, 3),IndexJ(n, 3, 1)] := 4;
 a[IndexI(n, 3, 3),IndexJ(n, 3, 2)] := 9;
 a[IndexI(n, 3, 3),IndexJ(n, 3, 3)] := 2;

// ПРОВЕРОЧНЫЙ ВЫВОД
writeln(q);
for i:=1 to n do
 begin
  for j:=1 to n do
  write(a[i,j]:4);
  writeln;
 end;

 while q<n do
 begin
 k:=(q+1)div 2;
 d:=(sqr(q+2))+1;
 for i:=(n-q) div 2+1 to (n-q) div 2+q do
 for j:=(n-q) div 2+1 to (n-q) div 2+q do
 begin
  a[i,j]:=a[i,j]+(2*(q+2-1));
 end;

 a[(n-q) div 2 +1-1,(n-q) div 2+1-1] := a[(n-q) div 2 +1-1,(n-q) div 2+1-1] +d-
k-k-k-1;
```

```
 a[(n-q) div 2 +1-1,(n-q) div 2+1+q] := a[(n-q) div 2 +1-1,(n-q) div 2+1+q] +d-
k-1;
 a[(n-q) div 2 +1+q,(n-q) div 2+1-1] := a[(n-q) div 2 +1+q,(n-q) div 2+1-1]
+k+1;
 a[(n-q) div 2 +1+q,(n-q) div 2+1+q] := a[(n-q) div 2 +1+q,(n-q) div 2+1+q]
+k+k+k+1;

// ПРОВЕРОЧНЫЙ ВЫВОД
writeln(q);
for i:=1 to n do
 begin
  for j:=1 to n do
  write(a[i,j]:4);
  writeln;
 end;



//верхняя строка
a[IndexI(n,q,0),IndexJ(n,q,1)]:=k;
for m:=1 to k-1 do
 begin
 a[IndexI(n,q,0),IndexJ(n,q,1+2*m)]:=d-2*k-1-m;
 a[IndexI(n,q,0),IndexJ(n,q,2*m)]:=m;
 end;
 //левый столбец
 a[IndexI(n,q,1),IndexJ(n,q,0)]:=d-2*k-1;
for m:=1 to k-1 do
 begin
 a[IndexI(n,q,1+2*m),IndexJ(n,q,0)]:=d-3*k-1-m;
 a[IndexI(n,q,2*m),IndexJ(n,q,0)]:=k+1+m;
 end;


// ПРОВЕРОЧНЫЙ ВЫВОД
writeln(q);
for i:=1 to n do
 begin
  for j:=1 to n do
  write(a[i,j]:4);
  writeln;
 end;


 // правый столбец
 for i:=1 to q do
 a[IndexI(n,q,i),IndexJ(n,q,q+1)]:=d-a[IndexI(n,q,i),IndexJ(n,q,0)];

 // нижняя строка
 for j:=1 to q do
 a[IndexI(n,q,q+1),IndexJ(n,q,j)]:=d-a[IndexI(n,q,0),IndexJ(n,q,j)];

// ПРОВЕРОЧНЫЙ ВЫВОД
writeln(q);
for i:=1 to n do
 begin
  for j:=1 to n do
  write(a[i,j]:4);
  writeln;
 end;


 q:=q+2;
 end;
```

```
// ПРОВЕРОЧНЫЙ ВЫВОД
writeln(q);


 for i:=1 to n do
 begin
  for j:=1 to n do
  write(a[i,j]:4);
  writeln;
 end;
      end.
```

ПРИЛОЖЕНИЕ Б
Программная реализация алгоритма шифрования

```
Program File_text;
type
matrix = array[1..100,1..100] of integer ;

function IndexI (n, q, i: integer):longint;
begin
  IndexI := ((n - q) div 2) + i;
end;

function IndexJ (n, q, j: integer):longint;
begin
  IndexJ := ((n - q) div 2) + j;
end;

var
a:matrix;
i,ii,jj,j,n,m,k,d,q,www:integer;
p:integer;
f1 : text;
f2:text;
st : char;
res:string[9];



begin
 n:=3;
 for i:=1 to n do for j:=1 to n do a[i,j]:=0;

 // строчка 1
 a[IndexI(n, 3, 1),IndexJ(n, 3, 1)] := 8;
 a[IndexI(n, 3, 1),IndexJ(n, 3, 2)] := 1;
 a[IndexI(n, 3, 1),IndexJ(n, 3, 3)] := 6;

 // строчка 2
 a[IndexI(n, 3, 2),IndexJ(n, 3, 1)] := 3;
 a[IndexI(n, 3, 2),IndexJ(n, 3, 2)] := 5;
 a[IndexI(n, 3, 2),IndexJ(n, 3, 3)] := 7;

 // строчка 3
 a[IndexI(n, 3, 3),IndexJ(n, 3, 1)] := 4;
 a[IndexI(n, 3, 3),IndexJ(n, 3, 2)] := 9;
 a[IndexI(n, 3, 3),IndexJ(n, 3, 3)] := 2;

// ПРОВЕРОЧНЫЙ ВЫВОД

for i:=1 to n do
 begin
  for j:=1 to n do
  write(a[i,j]:4);
  writeln;
 end;
assign   (f1, 'D:\text2.txt'); {связать с файлом file1.txt  файловую переменную
f1  }

reset (f1); { открыть файл для чтения }

while not eof (f1) do { пока не конец файла  f1}
begin
  p:=1;
```

```
  res := '          ';
  while (not eof(f1)) and (p < 10) do
  begin
    read(f1,st);
    {write(st);}
    ii:=p div 3 + 1;
    jj:=p mod 3;
    if jj = 0 then
    begin
      jj := 3;
      ii := ii - 1;
     end;
    res[a[ii,jj]] := st;
    inc(p);
  end;
  write(res);          { выводим на экран }
end;

close (f1); { закрыть файл  для чтения}
Rewrite (f2, 'D:\text3.txt');
write(f2,res);
close(f2);

      end .
```

ПРИЛОЖЕНИЕ В
Программа для аппаратной реализации шифра

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include <misc.h>
#include "stm32f10x_usart.h"
#include "stm32f10x.h"

void port(void); //подпрограмма настройки портов
void ext(void); //подпрограмма настройки внешних прерываний
void usart3_init(void);
int square_design(void);
int shifr(void);
int rashfr (void);
int IndexI (int n1,int q,int i);
int IndexJ (int n1,int q,int j);

int mas2[300],mas3[300];

int mas1[300];
int a[9][9];
int a2[7][7];
int i,z,t,ii,jj,j,m,o,o1,o2,p,k,d,q,www,u,RXc,a1; //объявление переменных
int n=0;

u32 hh; //объявление переменной


//основная программа


int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC|RCC_APB2Periph_GPIOB|

RCC_APB2Periph_GPIOA, ENABLE); //включение тактирования
RCC_APB2PeriphClockCmd(RCC_APB2ENR_AFIOEN, ENABLE); //включение тактирования
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE); //включение тактирования

port(); //вызов подпрограммы настройки портов
ext(); //вызов подпрограммы настройки внешних прерываний
usart3_init();
o=0;

square_design();


      while(1) //бесконечный цикл
      {
            if(www==1)
            {
            o1=49-o%49;
              if(o%49==0)
              {}
              else
              {
                   for ( o2= 0;o2<o1; o2++)
                   {
                        mas1[o]=' ';
                        o++;
                  }
            }
```

```
      // отправка открытого текста в uart
              for(i=0;i<o;i++)
            {
                  send_to_uart(mas1[i]);
            }
            send_to_uart('    ');
            //send_to_uart('  ');

  // построение квадрата
            square_design();

 //шифрование
            u=o/49;
            u=1;
            for (z=0;z<u;z++)
            {
            shifr();
            }
            for(i=0;i<o;i++)
            {
                  send_to_uart(mas2[i]);
            }

            send_to_uart('    ');
            send_to_uart('    ');

//расшифрование
            for (z=0;z<u;z++)
                        {
            rashfr();
                        }
            for(i=0;i<o;i++)
            {
                  send_to_uart(mas3[i]);
            }
            www=0;
            }
      }
}


//подпрограмма отправки 1 байта по UART
void send_to_uart(uint8_t data)
{
while(!(USART3->SR & USART_SR_TC)); //при равенстве 1 бита TC в регистре SR
USART3->DR=data; //отправка байта через UART
}

void USART3_IRQHandler(void)
{

            RXc = USART_ReceiveData(USART3);
            if (RXc==0x0D)
            {
                  www=1;
                  //USART_Cmd(USART3, DISABLE); //включение USART3
            }
            else
        {
                  mas1[o]=RXc;
                o++;
        }
```

```
}

//Обработчик прерывания EXTI0
void EXTI0_IRQHandler(void)
{
if(a1==0x0001)
{
GPIOC->ODR^=GPIO_Pin_8; //инверсия состояния светодиода
EXTI->PR|=0x01; //очистка флаг
send_to_uart('o'); // отправка данных
send_to_uart('f');
send_to_uart('f');
a1--; //декремент переменной
}
else
{
GPIOC->ODR^=GPIO_Pin_8; //инверсия состояния светодиода
EXTI->PR|=0x01; //очистка флага
send_to_uart('o'); //отправка данных
send_to_uart('n');
a1++; //инкремент переменной
}
}



int square_design(void)
{

      q = 3;
      n = 7;
for(i=0 ; i<n;++i)
      {
      for (j=0; j<n;++j)
            {
            a[i][j]=0;
            }
      }

// строчка 1
a[IndexI(n, q, 0)][IndexJ(n, q, 0)] = 8;
a[IndexI(n, q, 0)][IndexJ(n, q, 1)] = 1;
a[IndexI(n, q, 0)][IndexJ(n, q, 2)] = 6;

// строчка 2
a[IndexI(n, q, 1)][IndexJ(n, q, 0)] = 3;
a[IndexI(n, q, 1)][IndexJ(n, q, 1)] = 5;
a[IndexI(n, q, 1)][IndexJ(n, q, 2)] = 7;

// строчка 3
a[IndexI(n, q, 2)][IndexJ(n, q, 0)] = 4;
a[IndexI(n, q, 2)][IndexJ(n, q, 1)] = 9;
a[IndexI(n, q, 2)][IndexJ(n, q, 2)] = 2;

while(q < n)
       {
            k = (q + 1)/2;
            d = ((q + 2)*(q + 2)) + 1;
           for(i = (n - q)/2; i < (n-q)/2 + q; ++i)
                 for(j = (n - q)/2; j < (n - q)/2 + q; ++j)
                 a[i][j]=a[i][j]+(2*(q+2-1));


          //угловые ячейки
```

```
            a[(n - q)/2 - 1][(n - q)/2 - 1] = a[(n - q)/2 - 1][(n - q)/2 -
1] + d - k - k - k - 1; // верхний левый
          a[(n - q)/2 - 1][(n - q)/2 + q] = a[(n - q)/2 - 1][(n - q)/2 + q] +
d - k - 1; // верхний правый
          a[(n - q)/2 + q][(n - q)/2 - 1] = a[(n - q)/2 + q][(n - q)/2 - 1] +
k + 1; // нижний левый
            a[(n - q)/2 + q][(n - q)/2 + q] = a[(n - q)/2 + q][(n - q)/2 +
q] + k + k + k + 1; // нижний правый


           //верхняя строка

           a[IndexI(n, q + 2, 0)][IndexJ(n, q + 2, 1)] = k;
           for(m = 1 ; m <= k - 1; ++m)
             {
             a[IndexI(n, q + 2, 0)][IndexJ(n, q + 2, 1 + 2*m)] = d - 2*k - 1
- m;
             a[IndexI(n, q + 2, 0)][IndexJ(n, q + 2, 2*m)] = m;
             }


          //левый столб!!!!!!!
             a[IndexI(n, q + 2, 1)][IndexJ(n, q + 2, 0)] = d - 2*k - 1;
             for(m = 1 ; m <= k - 1; ++m)
             {
                    a[IndexI(n, q + 2, 1 + 2*m)][IndexJ(n, q + 2, 0)] = d -
3*k - 1 - m;
                    a[IndexI(n, q + 2, 2*m)][IndexJ(n, q + 2, 0)] = m + 1 +
k;
             }


              //правый столбец!!!!!!!!!

             for (i = 0 ; i< q  ; ++i )
                   a[IndexI(n, q + 2, 1 + i)][IndexJ(n, q + 2, q+1 )] = d -
a[IndexI(n, q + 2, i +1)][IndexJ(n, q + 2, 0)];


             // нижняя строка!!!!!!!!!!
             for (j = 0; j < q; ++j )
                   a[IndexI(n, q, q )][IndexJ(n, q, j)]= d - a[IndexI(n, q,
-1)][IndexJ(n, q, j)];
             q = q + 2;

}
for(i=0 ; i<n;++i)
      {
      for (j=0; j<n;++j)
             {
             a2[i][j]=a[i][j];
             }
      }

}
```

```
int shifr(void)
{
p=1;
n=7;
t=0;
o1=o;
u=o/49;

      while (p < (n*n+1))
      {
    ii=p/7+1;
      jj=p%7;
            if (jj==0)
            {
                  jj=7;
                  ii=ii-1;
            }
      mas2[a2[ii-1][jj-1]-1+z*49]=mas1[p-1+z*49] ;
      p++;
      }

o=o1;

}


int IndexI (int n1,int q,int i)
  {
      int b;
      b=((n1 - q)/2) + i;
      return b;
  }


int IndexJ (int n1,int q,int j)
  {
      int c;
      c=((n1 - q)/2) + j;
      return c;
  }

int rashfr (void)
{

      n=7;


for (i=0;i<n;i++)
{
      for(j=0;j<n;j++)
      {
            mas3[(i)*n+j+49*z]=mas2[a2[i][j]-1+49*z];
      }
}
}



//подпрограмма настройки внешних прерываний
void ext(void)
{
```

```
EXTI_InitTypeDef EXTI_InitStruct; //объявление структуры настройки внешних
прерываний
EXTI_InitStruct.EXTI_Line=EXTI_Line0; //выбор настраиваемой линии
EXTI_InitStruct.EXTI_LineCmd=ENABLE; //разрешение прерывания на выбранной линии
EXTI_InitStruct.EXTI_Mode=EXTI_Mode_Interrupt; //режим прерывания
EXTI_InitStruct.EXTI_Trigger=EXTI_Trigger_Rising; //прерывание по переднему
фронту
EXTI_Init(&EXTI_InitStruct); //заполнение объявленной структуры
NVIC_EnableIRQ (EXTI0_IRQn); //разрешение прерывания
}

//подпрограмма настройки портов
void port(void)
{
GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_8|GPIO_Pin_9; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим - вывод, тип - push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_10; //выбор настраиваемых выводов - Tx
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение аксимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_AF_PP; //режим - вывод,
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_11; //выбор настраиваемых выводов - Rx
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим - дифференциальный вход
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим - дифференциальный вход
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение объявленной структуры
GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0); //назначение вывода
0 порта А источником внешних прерываний
}

void usart3_init(void)
{
      NVIC_InitTypeDef NVIC_InitStructure;
      NVIC_InitStructure.NVIC_IRQChannel = USART3_IRQn;
      NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
      NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
      NVIC_Init(&NVIC_InitStructure);

      USART_InitTypeDef USART_InitStructure; //объявление структуры настройки
USART
      USART_InitStructure.USART_BaudRate = 9600; //скорость
      USART_InitStructure.USART_WordLength = USART_WordLength_8b;//8 бит данных
      USART_InitStructure.USART_StopBits = USART_StopBits_1;//один стоп бит
      USART_InitStructure.USART_Parity = USART_Parity_No; //четность - нет
      USART_InitStructure.USART_HardwareFlowControl =

USART_HardwareFlowControl_None; //управлени потоком - нет
      USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

//разрешение приема и передачи
      USART_Init(USART3, &USART_InitStructure); //заполнение объявленной
структуры
      USART_Cmd(USART3, ENABLE); //включение USART3
      USART_ITConfig(USART3, USART_IT_RXNE, ENABLE); // разрешение прерывания
приемника
```

```
        //NVIC_EnableIRQ (USART3_IRQn); //разрешение прерываний UART3
}
```

АНТИПЛАГИАТ
ТВОРИТЕ СОБСТВЕННЫМ УМОМ

Отчет о проверке на заимствования №1

Автор: Сапсай Михаил maiksapsay@gmail.com / ID: 8627637
Проверяющий: Сапсай Михаил (maiksapsay@gmail.com / ID: 8627637)

Отчет предоставлен сервисом «Антиплагиат»- http://users.antiplagiat.ru

ИНФОРМАЦИЯ О ДОКУМЕНТЕ

№ документа: 8
Начало загрузки: 28.01.2021 16:44:18
Длительность загрузки: 00:00:01
Имя исходного файла: ШЕВЦОВ ВКР.pdf
Название документа: ШЕВЦОВ ВКР
Размер текста: 47 кБ
Символов в тексте: 48257
Слов в тексте: 6259
Число предложений: 296

ИНФОРМАЦИЯ ОБ ОТЧЕТЕ

Последний готовый отчет (ред.)
Начало проверки: 28.01.2021 16:44:21
Длительность проверки: 00:00:02
Комментарии: не указано
Модули поиска: Модуль поиска Интернет

ЗАИМСТВОВАНИЯ	САМОЦИТИРОВАНИЯ	ЦИТИРОВАНИЯ	ОРИГИНАЛЬНОСТЬ
21,49%	0%	0%	78,51%

Заимствования — доля всех найденных текстовых пересечений, за исключением тех, которые система отнесла к цитированиям, по отношению к общему объему документа.
Самоцитирования — доля фрагментов текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника, автором или соавтором которого является автор проверяемого документа, по отношению к общему объему документа.
Цитирования — доля текстовых пересечений, которые не являются авторскими, но система посчитала их использование корректным, по отношению к общему объему документа. Сюда относятся оформленные по ГОСТу цитаты; общеупотребительные выражения; фрагменты текста, найденные в источниках из коллекций нормативно-правовой документации.
Текстовое пересечение — фрагмент текста проверяемого документа, совпадающий или почти совпадающий с фрагментом текста источника.
Источник — документ, проиндексированный в системе и содержащийся в модуле поиска, по которому проводится проверка.
Оригинальность — доля фрагментов текста проверяемого документа, не обнаруженных ни в одном источнике, по которым шла проверка, по отношению к общему объему документа.
Заимствования, самоцитирования, цитирования и оригинальность являются отдельными показателями и в сумме дают 100%, что соответствует всему тексту проверяемого документа.
Обращаем Ваше внимание, что система находит текстовые пересечения проверяемого документа с проиндексированными в системе текстовыми источниками. При этом система является вспомогательным инструментом, определение корректности и правомерности заимствований или цитирований, а также авторства текстовых фрагментов проверяемого документа остается в компетенции проверяющего.

№	Доля в отчете	Источник	Ссылка	Актуален на	Модуль поиска
[01]	8,49%	Микроконтроллеры STM	http://portal.tpu.ru:7777	09 Мар 2018	Модуль поиска Интернет
[02]	3,45%	Регистр диабета персональные данные | Кусивака . RU	http://kusivaka.ru	04 Апр 2016	Модуль поиска Интернет
[03]	2,67%	makarova n.v. volshebnyj mir magicheskih kvadratov. (2009).pdf	http://inethub.olvi.net.ua	26 Апр 2014	Модуль поиска Интернет

Еще источников: 17
Еще заимствований: 6,89%