

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой №44

проф., д-р техн. наук, проф.

должность, уч. степень, звание

подпись, дата

М.Б.Сергеев

инициалы, фамилия

БАКАЛАВРСКАЯ РАБОТА

на тему Программное средство кодирования состояний конечных автоматов

выполнена Тегай Екатериной Дмитриевной

фамилия, имя, отчество студента в творительном падеже

по направлению подготовки

09.03.01

код

Информатика и вычислительная

наименование направления

техника

наименование направления

направленности

04

код

Компьютерные технологии, системы и

наименование направленности

сети

наименование направленности

Студент группы №

4143

09.06.2025

подпись, дата

Е.Д.Тегай

инициалы, фамилия

Руководитель

доц., канд. техн. наук

должность, уч. степень, звание

09.06.2025

подпись, дата

Т.Н.Соловьева

инициалы, фамилия

Санкт-Петербург 2025

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение высшего образования
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ»

УТВЕРЖДАЮ

Заведующий кафедрой № 44

проф., д-р техн. наук, проф.

должность, уч. степень, звание

подпись, дата

М.Б. Сергеев

инициалы, фамилия

ЗАДАНИЕ НА ВЫПОЛНЕНИЕ БАКАЛАВРСКОЙ РАБОТЫ

студенту группы

4143

номер

Тегай Екатерине Дмитриевне

фамилия, имя, отчество

на тему

Программное средство кодирования состояний конечных автоматов

утвержденную приказом ГУАП от

27.03.2025

№

11-387/25

Цель работы:

разработка программного средства, реализующего различные способы

кодирования состояний конечных автоматов для синтеза автомата в виде интегральной

схемы.

Задачи, подлежащие решению: обзор методов кодирования состояний автомата; обзор

аналогичных программных средств; формирование технического задания; выбор средств

разработки; реализация алгоритмов кодирования; реализация пользовательского

интерфейса; проверка работоспособности программного средства.

Содержание работы (основные разделы):

формирование технического задания;

разработка программного средства; проверка работоспособности программного средства.

Срок сдачи работы « 09 »

июня

2025

Руководитель

доц., канд. техн. наук

должность, уч. степень, звание

подпись, дата

Т.Н. Соловьева

инициалы, фамилия

Задание принял(а) к исполнению

студент группы №

4143

подпись, дата

Е.Д. Тегай

инициалы, фамилия

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	5
1 ФОРМИРОВАНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ	7
1.1 Обзор методов кодирования состояний конечных автоматов	7
1.1.1 Унитарный код	7
1.1.2 Последовательное кодирование	8
1.1.3 Код Джонсона	8
1.1.4 Код Грея	9
1.1.5 Метод кодирования с редукцией веса Хэмминга	10
1.1.6 Метод кодирования с редукцией расстояния Хэмминга	11
1.1.7 Генетический алгоритм для минимизации сложности логической схемы	12
1.2 Примеры выполнения алгоритмов кодирования	15
1.2.1 Унитарный код	16
1.2.2 Последовательное кодирование	16
1.2.3 Код Джонсона	17
1.2.4 Код Грея	17
1.2.5 Метод с редукцией веса Хэмминга	18
1.2.6 Метод кодирования с редукцией расстояния Хэмминга	20
1.2.7 Генетический алгоритм	24
1.3 Форматы описания конечных автоматов	26
1.4 Обзор программных средств, выполняющих автоматическое кодирование состояний	26
1.5 Техническое задание на разработку программного средства	28
1.5.1 Введение	29
1.5.2 Основания для разработки	29

1.5.3	Назначения разработки	29
1.5.4	Требования к программе	29
2	РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА	31
2.1	Выбор средств разработки	31
2.2	Схема работы программы	32
2.3	Работа с входными и выходными файлами	33
2.3.1	Загрузка и парсинг файлов	33
2.3.2	Сохранение результатов	35
2.4	Алгоритмы кодирования состояний	36
2.5	Реализация алгоритмов кодирования состояний	40
2.5.1	Программная реализация унитарного кода	41
2.5.2	Программная реализация последовательного кодирования	41
2.5.3	Программная реализация кодирования Джонсона	41
2.5.4	Программная реализация кодирования Грея	42
2.5.5	Программная реализация метода с редукцией веса Хэмминга	42
2.5.6	Программная реализация метода с редукцией расстояния Хэмминга	44
2.5.7	Программная реализация генетического алгоритма	45
2.6	Разработка пользовательского интерфейса	46
3	ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА	50
3.1	Проверка работоспособности алгоритмов на примере автомата Мура	50
3.1.1	Унитарный код	52
3.1.2	Последовательное кодирование	53
3.1.3	Код Джонсона	53
3.1.4	Код Грея	53
3.1.5	Метод с редукцией веса Хэмминга	54
3.1.6	Метод с редукцией расстояния Хэмминга	55

3.1.7	Генетический алгоритм.....	56
3.2	Проверка работоспособности алгоритмов на примере автомата Мили.....	57
3.2.1	Унитарный код	57
3.2.2	Последовательное кодирование	58
3.2.3	Код Джонсона	58
3.2.4	Код Грея.....	58
3.2.5	Метод кодирования с редукцией веса Хэмминга.....	58
3.2.6	Метод кодирования с редукцией расстояния Хэмминга	59
3.2.7	Генетический алгоритм.....	59
3.3	Чтение файлов.....	60
3.4	Сохранение файлов.....	61
3.5	Обработка граничных и ошибочных ситуаций.....	72
3.6	Анализ программного тестирования	73
3.6.1	Метод с редукцией веса Хэмминга.....	73
3.6.2	Метод с редукцией расстояния Хэмминга	74
3.6.3	Генетический алгоритм.....	75
3.7	Сравнительный анализ методов кодирования состояний конечных автоматов.	76
	ЗАКЛЮЧЕНИЕ	79
	СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	80
	ПРИЛОЖЕНИЕ А. Оценка результатов кодирования состояний автомата Мура	82
	ПРИЛОЖЕНИЕ Б. Оценка результатов кодирования состояний автомата Мили	83
	ПРИЛОЖЕНИЕ В Листинг кода программы	84

ВВЕДЕНИЕ

При разработке и реализации цифровых автоматов в виде интегральных схем одной из важных задач является выбор способа кодирования состояний цифрового автомата. Эффективное кодирование состояний напрямую влияет на такие ключевые характеристики проектируемых схем, как: быстродействие, надёжность, энергопотребление и аппаратные затраты.

Несмотря на наличие целого ряда уже существующих программных решений, большинство из них предоставляют пользователю лишь ограниченный набор наиболее простых методов кодирования.

Целью данной работы является разработка программного средства, реализующего различные способы кодирования состояний конечных автоматов, для синтеза автомата в виде интегральной схемы.

Для достижения поставленной цели в рамках текущей работы решаются следующие задачи:

- Обзор методов кодирования состояний автомата;
- Обзор аналогичных программных средств;
- Формирование технического задания;
- Выбор средств разработки;
- Реализация алгоритмов кодирования;
- Реализация пользовательского интерфейса;
- Проверка работоспособности программного средства.

Практическая значимость работы состоит в создании программного средства, которое позволяет автоматизировать процесс кодирования состояний конечных автоматов и тем самым сократить время проектирования и в некоторых случаях – объём интегральной схемы, реализующей автомат.

Работа содержит три основных раздела. В первом разделе приведён обзор существующих методов и средств кодирования с последующим сравнительным анализом. Заключающим этапом этого раздела является формирование технического задания на разработку программного средства. Во втором разделе подробно описывается процесс разработки программного

средства. В третьем разделе показаны результаты проверки работоспособности разработанного программного средства.

1 ФОРМИРОВАНИЕ ТЕХНИЧЕСКОГО ЗАДАНИЯ

1.1 Обзор методов кодирования состояний конечных автоматов

При проектировании конечных автоматов выбор метода кодирования состояний напрямую влияет на сложность реализации устройства и его характеристики. Основная проблема заключается в отсутствии универсального алгоритма, который был бы эффективно применим во всех задачах и условиях проектирования. Поэтому для поиска оптимального или близкого к нему решения применяются различные методы кодирования, каждый из которых имеет свои особенности, достоинства и недостатки. После выбора кодировки схема автомата реализуется классическим методом структурного синтеза, включающим построение таблиц переходов и выходов, выбор типа триггеров и синтез комбинационной схемы [1].

От выбора способа кодирования состояний автомата зависит число используемых элементов памяти и сложность логических функций, определяющая такие параметры реализации автомата в виде интегральной схемы на логических элементах (ИС), как площадь кристалла, потребляемая мощность, надёжность и быстродействие.

В следующих подразделах будет подробно рассмотрен ряд известных методов кодирования состояний, а также проведён сравнительный анализ.

1.1.1 Унитарный код

Унитарный код (One-Hot Encoding) представляет собой метод, при котором каждому состоянию автомата присваивается уникальный двоичный код с единственным единичным битом. Как отмечает Строгонов А.В. в своей работе [2], этот метод широко применяется в проектировании на программируемых логических интегральных схемах (ПЛИС).

Унитарный код использует N триггеров для автомата с N состояниями. К преимуществам метода относятся простота реализации и возможность определения текущего состояния по одному биту. Переход между состояниями требует изменения только двух битов – сброс одного и установка другого, что упрощает комбинационную логику переходов, как это показано в [3]. Однако

при частых переходах между состояниями это же свойство увеличивает энергопотребление по сравнению с методами, где меняется только один бит.

Высокая аппаратная сложность является его основным недостатком. Например, для 10 состояний этот метод потребует 10 триггеров вместо 4 при бинарном кодировании.

1.1.2 Последовательное кодирование

Последовательное кодирование – один из простейших методов представления состояний конечного автомата, при котором состояниям присваиваются последовательно возрастающие двоичные коды. Для представления N состояний требуется количество разрядов, определяемое по формуле:

$$N_{\text{разр}} = \lceil \log_2 N \rceil, \quad (1)$$

где $\lceil \rceil$ – округление вверх. Так, для автомата с 8 состояниями потребуется три триггера, как и для автомата с 5 состояниями.

Преимуществами последовательного кодирования являются простота реализации и удобство проектирования автомата, поскольку коды присваиваются в порядке двоичного счёта. Недостатком является то, что переходы между последовательными состояниями часто сопровождаются изменением нескольких битов одновременно, что увеличивает энергопотребление и усложняет функции возбуждения триггеров.

1.1.3 Код Джонсона

Кодирование Джонсона основано на последовательном циклическом сдвиге битов влево с инверсией переносимого бита, устанавливаемого в младший разряд. В основе метода лежит n -разрядный сдвиговый регистр с обратной связью через инвертор. Подробное описание принципа его работы приведено в работе Е.П.Угрюмова [4].

Для кодирования N состояний требуется количество разрядов, определяемое по формуле:

$$N_{\text{разр}} = \left\lceil \frac{N}{2} \right\rceil, \quad (2)$$

где $\lceil \cdot \rceil$ – округление вверх при нечётном N . Например, для 4 состояний достаточно двух триггеров, а для пяти – трёх.

n -разрядный код может принимать 2^n уникальных комбинаций – меньше, чем 2^n при бинарном кодировании, но больше, чем при унитарном. К преимуществам относится простота реализации и минимальное расстояние Хэмминга между соседними состояниями, равное единице, что упрощает функции возбуждения триггеров и снижает количество изменяемых переменных при переходах, и простота реализации, а к недостаткам – использование неминимального числа разрядов для кодирования заданного числа состояний. Это ограничивает применение метода в задачах, требующих максимально возможного набора уникальных кодов при заданной разрядности.

1.1.4 Код Грея

Другим методом, обеспечивающим изменение только одного бита между соседними состояниями, является кодирование Грея. Особенности его применения при проектировании конечных автоматов подробно изложены в пособии Е.П.Угрюмова [4]. Для представления N состояний требуется количество триггеров, определяемое по формуле (1).

Код Грея формируется из бинарного кода по правилу:

$$G_i = N \oplus (N \gg 1), \quad (3)$$

где N – десятичный номер состояния, $\ll 1$ – операция побитового сдвига числа на один разряд вправо, а \oplus – операция исключающего ИЛИ (XOR).

Существует ещё несколько способов формирования кода Грея, помимо вышеописанного. Альтернативами являются, например, метод побитового сложения по модулю 2 с его копией, сдвинутой на один разряд вправо, либо построение последовательности кодов Грея путём отражения списка двоичных чисел с добавлением ведущих битов.

Преимуществом метода, как и в кодировании Джонсона, является минимальное расстояние Хэмминга между соседними состояниями.

Особенностью метода является зависимость от исходного порядка состояний – код Грея преобразует уже существующую последовательность кодов.

1.1.5 Метод кодирования с редукцией веса Хэмминга

Как отмечается в работе [5], при минимизации конечных автоматов целесообразно учитывать структуру переходов между состояниями и их активность, так как это напрямую влияет на сложность функций переходов.

Метод использует минимизацию веса Хэмминга между кодами состояний с высокой частотой переходов [6]. Первоначально строится таблица пар последователей, в которой для каждой пары состояний фиксируется количество повторений этой пары. Парам последователей называются такие состояния автомата, в которые можно перейти из одного общего состояния по условиям перехода, имеющим хотя бы одну общую переменную.

Например, пусть из состояния H_0 в состояния H_1 и H_2 осуществляются переходы по условиям « $X_3 \wedge X_4$ » и « $X_3 X_4$ » соответственно. Тогда общей переменной здесь является « X_3 », что формирует первую пару-последователей (H_1, H_2) с количеством повторений 1.

Затем формируется матрица частоты переходов размером $N * N$, где N – количество состояний автомата, в которой элемент m_{ij} обозначает число переходов от состояния S_i к состоянию S_j . На основе матрицы переходов рассчитывается активность каждого состояния:

$$A(S_i) = \sum_{j=1}^N m_{ij} + \sum_{j=1}^N m_{ji} \quad (4)$$

Наиболее активному состоянию присваивается код с минимальным весом (например, 000 при трёхразрядной кодировке). При дальнейшем кодировании приоритет отдаётся тем парам последователей, которые имеют наибольшее число переходов. При этом алгоритм стремится выбрать код с минимальным весом Хэмминга. Таким образом, кодирование позволяет сократить количество единичных значений в логических выражениях и сложность функций возбуждения триггеров. Данный алгоритм целесообразно

применять при реализации автоматов на D-триггерах. Это обусловлено тем, что функция возбуждения D-триггера совпадает с входным сигналом.

1.1.6 Метод кодирования с редукцией расстояния Хэмминга

Как отмечено в работе Ю.В. Поттосина [3], минимизация энергопотребления и объёма схемы конечного автомата достигается снижением количества переключений элементов памяти. Автор предлагает метод, основанный на размещении состояний автомата в вершинах булева гиперкуба с минимизацией суммы произведений весов переходов и расстояния Хэмминга между кодами состояний.

Аналогичный критерий применяется в методе кодирования с редукцией расстояния Хэмминга [6], но реализуется с помощью итерационного алгоритма на основе матрицы весов переходов. Этот метод направлен на минимизацию переключений, что делает его эффективным при использовании триггеров, чувствительных к изменению состояния, например, Т- или JK-триггеров, где переключение зависит от различий между текущим и следующим состоянием.

В основе метода лежит анализ частоты взаимодействия между состояниями, представленный в виде матрицы частоты переходов, определённой в предыдущем подразделе. Активность состояния S_i выражается суммой всех весов пар, в которые оно входит:

$$w^*(S_i) = \sum_{j=1}^N w_{ij} \quad (5)$$

Оптимальная кодировка достигается минимизацией суммы произведений весов переходов и расстояний Хэмминга между кодами состояний. Критерий оптимальности выражается формулой:

$$W = \sum_{p,q \in Q} w(p,q) * d(r(p), r(q)), \quad (6)$$

где $r(p)$ и $r(q)$ – коды состояний p и q , а $d(r(p), r(q))$ – расстояние Хэмминга между ними, и Q – множество закодированных состояний.

Кодирование начинается с пары состояний с максимальным весом связи, которым назначаются соседние коды (например, 000 и 001). Каждому новому состоянию присваивается код, минимизирующий следующую величину суммы весов состояний:

$$S(\alpha) = \sum_{\substack{S \in Q_{\text{закодир.}} \\ w(S, S_k) > 0}} w(S, S_k) * d(\rho(S), \alpha) \quad (7)$$

где:

- $Q_{\text{закодир}}$ – множество закодированных состояний,
- $w(S, S_k)$ – вес пары состояний,
- $\rho(S)$ – код состояния S из множества $Q_{\text{закодир}}$,
- α – кандидат для кода состояния
- $d(\rho(S), \alpha)$ – расстояние Хэмминга между кодами.

Процесс продолжается до тех пор, пока не будут закодированы все состояния.

1.1.7 Генетический алгоритм для минимизации сложности логической схемы

Генетический алгоритм для минимизации сложности логической схемы (далее – генетический алгоритм) является эвристическим методом оптимизации, основанным на принципах естественного отбора. Его ценность заключается в способности находить близкие к оптимальным варианты кодирования состояний конечного автомата за разумное время и адаптивности под различные критерии. Генетический алгоритм реализуется для конечных автоматов на D-триггерах, где функции возбуждения строятся в соответствии с таблицей переходов для D-триггеров, где значение триггера на каждом такте соответствует целевому значению состояния.

В процессе работы алгоритм оперирует понятиями *особи* (хромосомы) – одного варианта кодирования состояний, *популяции* – набора этих вариантов, и *оценивающей функции*, которая количественно оценивает качество кодирования по совокупности критериев, объединённых в сводную функцию.

Размер популяции определяется случайным образом в диапазоне от 2 до 10 особей. Число итераций пользователь может задать вручную в диапазоне от 2 до 10000 итераций, либо использовать значения по умолчанию, вычисляемое по формуле:

$$num_iterations = \frac{num_r}{N} * C \quad (8)$$

где num_r – разрядность кода, N – количество состояний автомата, C – эмпирический коэффициент. Значение C в данном случае определено как 50 – это значение обеспечивает достаточное количество итераций при любом размере автомата, за которое находилось решение без чрезмерного увеличения работы алгоритма. Для автомата с 8 состояниями, согласно формуле (8), количество итераций будет равно 18.

Разрядность кода в генотипе хромосомы определяется формулой (1). Оценивающая функция определяет оценку качества кодирования по количеству логических элементов AND, OR, NOT в функциях возбуждения триггеров. Это рассчитывается по формуле:

$$F = \sum (numAnd + numOR + numNOT)$$

где $numAND$, $numOR$ и $numNOT$ – количество логических элементов типа «И», «ИЛИ» и «НЕ» соответственно.

Селекция проходит на основе значения оценок, полученных каждой хромосоме. Наименьшее значение оценки имеет большую ценность для селекции. Одноточечный кроссинговер выбирает точку «разрыва» для дальнейшего обмена случайным образом. Мутация происходит с вероятностью, равной 0.1 (10%), и применяется на каждый код потомка. То есть, в одном потомке может произойти несколько мутаций.

Д.Голдберг в работе [7] отмечает, что генетический алгоритм принципиально отличается от классических методов оптимизации тем, что он ведёт поиск по совокупности популяций, а не по одной популяции, а также применяет вероятностные правила в своей работе, а не детерминированные.

Как отмечено в работе [8], в этом алгоритме используются стандартные генетические операторы:

- Селекция – отбор наиболее приспособленных особей;
- Кроссинговер – комбинирование кодов состояний родителей для получения потомков. Схематично одноточечный кроссинговер представлен на рисунке 1, где а и b – родители, а с и d – потомки. В данном случае точка кроссинговера определена посередине – между первым и вторым битами, но может стоять в любом месте по всей длине кода.

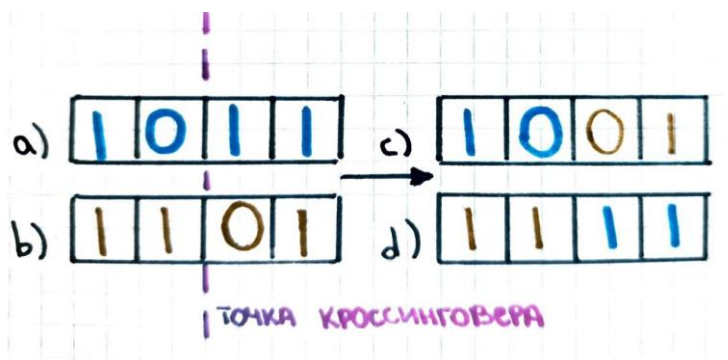


Рисунок 1 – Процесс кроссинговера

- Мутация – случайное изменение одного бита кода некоторого состояния с малой долей вероятности для генерации новых, потенциально более эффективных решений.

Однако, в процессе применения операторов кроссинговера и мутации возникает проблема границ, заключающаяся в возможности выхода значений генов за допустимые пределы, что приводит к формированию неподходящих для реализации кодировок состояний. Это требует применения специальных механизмов контроля, таких как исправление некорректных кодов (дублирующихся, например) или ограничение операторов, чтобы сохранить допустимость и корректность генотипов. Подробнее эта проблема и возможные пути её решения описаны в [9].

Следует отметить, что ввиду специфики применяемых критериев оптимизации, генетический алгоритм не формирует кодировки, основанные на жёстко заданных шаблонах, таких как кодирование Джонсона и унитарный код. Поскольку целью алгоритма является получение схем с минимальной

логической сложностью и минимальным числом разрядов, кодировки с избыточным числом бит или фиксированной структурой невозможны.

Завершение работы алгоритма осуществляется при достижении определённого числа итераций.

1.2 Примеры выполнения алгоритмов кодирования

Рассмотрим применение описанных выше методов кодирования для конечного автомата, имеющего структуру, показанную на рисунке 2.

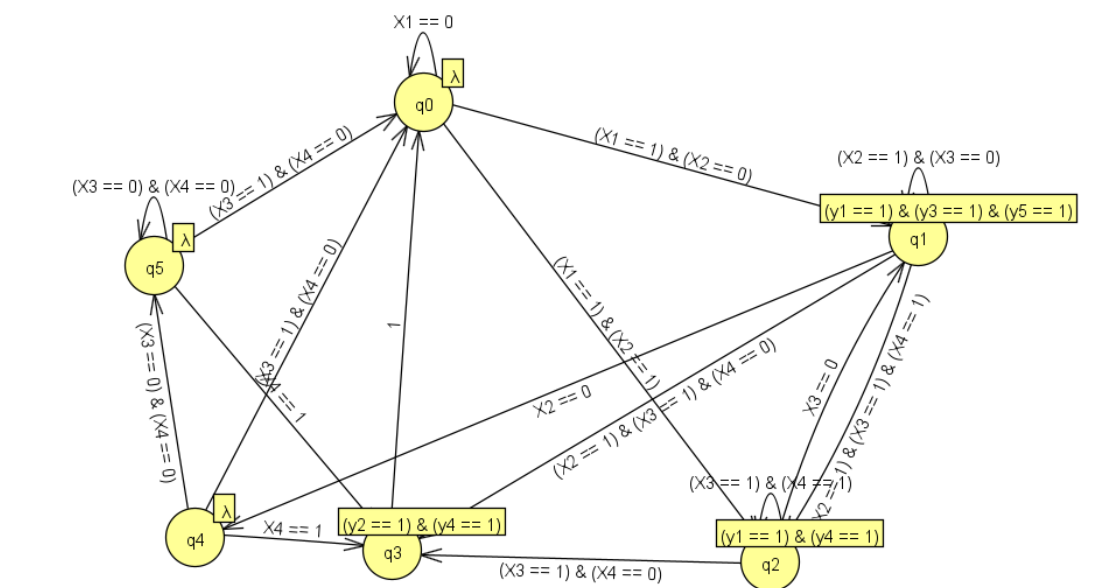


Рисунок 2 – Пример конечного автомата

Автомат на рисунке 2 имеет 6 состояний. Построим матрицу частоты переходов: строки – исходные состояния, а столбцы – состояния переходов. Если условие перехода истинно, в ячейке указывается имя переменной, а если ложно – перед именем добавляется «n» (от английского «not», обозначающего отрицание). Ячейка остаётся пустой, если переход между данной парой состояний отсутствует. Для упрощения восприятия опустим знак «&», считая, что последовательная запись переменных подразумевает их логическое умножение. Итог представлен в таблице 1.

Таблица 1.

Достижимое сост. Начальное сост.	q0	q1	q2	q3	q4	q5
q0	nX1	X1nX2	X1X2			
q1		X2nX3	X2X3X4	X2X3nX4	nX2	
q2		nX3	X3X4	X3nX4		
q3	1					
q4	X3nX4			X4		nX3nX4
q5	X3nX4			X4		nX3nX4

Далее будут представлены подробные расчёты по каждому из рассмотренных алгоритмов кодирования состояний.

1.2.1 Унитарный код

При унитарном кодировании разрядность совпадает с числом состояний и получаем такой результат:

q_0 : 000001,

q_1 : 000010,

q_2 : 000100,

q_3 : 001000,

q_4 : 010000,

q_5 : 100000.

1.2.2 Последовательное кодирование

Для определения разрядности кодов воспользуемся формулой (1):

$$N_{\text{разрядность}} = \lceil \log_2 N \rceil = \lceil \log_2 6 \rceil = 3$$

Получаем такой результат, где в данном случае код состояния соответствует двоичному коду его индекса :

$q_0 = 000$,

$q_1 = 001$,

$q_2 = 010$,

$q_3 = 011$,

$$q4 = 100,$$

$$q5 = 101.$$

1.2.3 Код Джонсона

Для определения разрядности кодов воспользуемся формулой (2):

$$N_{\text{триг}} = \left\lfloor \frac{N}{2} \right\rfloor = \frac{6}{2} = 3.$$

Первое состояние $q0$ пусть будет закодировано числом 000. Далее необходимо сдвинуть это число влево на один бит и смещаемый бит инвертировать и подставить справа. Получаем код для $q1$:

$$q1 = 001$$

Аналогично кодируем все оставшиеся состояния и в итоге получаем такой результат:

$$q0 = 000,$$

$$q1 = 001,$$

$$q2 = 011,$$

$$q3 = 111,$$

$$q4 = 110,$$

$$q5 = 100.$$

1.2.4 Код Грея

Определим разрядность кода с помощью формулы (1):

$$N_{\text{разрядность}} = \lceil \log_2 N \rceil = \lceil \log_2 6 \rceil = 3$$

Присвоим каждому состоянию код с помощью последовательного кодирования, результат которого приведён выше. Для получения кода первого состояния $q0$ воспользуемся формулой (3):

$$G_{q_0} = 000 \oplus (000 \gg 1) = 000 \oplus 000 = 000$$

Получаем в данном случае отсутствие изменений. Аналогичная ситуация и у $q1$. Далее кодируем состояние $q2$:

$$G_{q_2} = 010 \oplus (010 \gg 1) = 010 \oplus 001 = 011$$

Получаем код для $q2$, равный 011. Продолжаем кодировать оставшиеся состояния и получаем такие результаты:

$q_0: 000,$
 $q_1: 001,$
 $q_2: 011,$
 $q_3: 010,$
 $q_4: 110,$
 $q_5: 111.$

1.2.5 Метод с редукцией веса Хэмминга

Реализация начинается с поиска пар последователей на основе графа автомата. Пара состояний образует пару последователей, если они имеют общее исходное состояние и условия перехода к ним содержат хотя бы одну общую переменную.

Так, из состояния q_0 в состояния q_1 и q_2 переходы осуществляются по условиям « $X_1 \wedge X_2$ » и « $X_1 X_2$ » – общей переменной здесь является « X_1 », что формирует первую пару последователей (q_1, q_2) с количеством повторений 1.

Далее, из состояния q_1 по условиям « $X_3 X_4$ » и « $X_3 \wedge X_4$ » выполняются переходы в состояния q_2 и q_3 , имеющие общую переменную « X_3 » и не содержащие петлевых переходов. Пара (q_2, q_3) фиксируется с повторением 1.

Из состояния q_2 аналогично выявляется совпадение по переменной « X_3 » для переходов q_2 и q_3 , увеличивая количество повторений пары (q_2, q_3) до 2.

В состоянии q_3 возможен лишь один переход, совпадений нет. Из состояния q_4 переходы в q_0 и q_5 осуществляются по условиям « $X_3 \wedge X_4$ » и « $\neg X_3 \wedge X_4$ », общей переменной является « $\neg X_4$ ». Пара (q_0, q_5) фиксируется с повторением 1.

Из состояний q_5 обнаруживается аналогичное совпадение, увеличивая количество повторений пары (q_0, q_5) до 2. Все найденные пары и количество их повторений отражены в таблице 2.

Таблица 2.

Пары последователей	Количество повторений
q1, q2	1
q2, q3	2
q0, q5	2

Следующим этапом выполняется подсчёт количество входящих переходов для каждой вершины автомата, включая петлевые переходы. Результаты сведены в таблицу 3. Основа для подсчёта – граф переходов, представленный на рисунке 1.

Таблица 3.

Имя состояния	Количество входящих переходов
q0	4
q1	3
q2	3
q3	4
q4	1
q5	2

Переходим к кодированию состояний. Результаты будут представлены в таблице 5. Сначала определим разрядность кодов. Поскольку состояний 6, по формуле (1) требуется 3 разряда. В качестве первого состояния выберем q3, так как по таблице 3 оно имеет наибольшее число входящих переходов (альтернативно можно было бы выбрать q0 – результирующая кодировка отличалась бы, но была бы равнозначно эффективна). Состоянию q3 присваивается код с минимальным числом единиц – 000.

Далее проверим по таблице 2, составляет ли q3 пару последователей. Она есть – с состоянием q2. Для него подбираем код с минимальным расстоянием Хэмминга до кода q3. Возможные варианты: 001, 010, 100.

Выберем, например, 001. Продолжим, выбрав по таблице 3 следующее состояние с наибольшим числом входящих переходов – q0. Подходящие коды с минимальным расстоянием до уже закодированных состояний: 010, 100.

Возьмём 010. Проверим таблицу 2 – q_0 составляет пару с q_5 . Для q_5 возможные коды на расстоянии Хэмминга 1 от q_0 : 110, 011.

Выберем, например, 110. Далее – состояние q_1 . По таблице 2 оно образует пару с q_2 , у которого код 001. Возможные коды с расстоянием Хэмминга 1 от q_2 : 101, 011.

Выберем 011. Остаётся q_4 , не образующее пар последовательностей. Для него подойдёт любой из оставшихся кодов: 100, 101, 111.

Выберем, например, число с наименьшим числом единиц – 100. Таким образом, все состояния закодированы, алгоритм завершён и имеет такой результат:

q_0 : 010,

q_1 : 011,

q_2 : 001,

q_3 : 000,

q_4 : 100,

q_5 : 110.

1.2.6 Метод кодирования с редукцией расстояния Хэмминга

Разрядность кодов, как и ранее, определяется по формуле (1) и составляет 3. Первым этапом является построение матрицы частоты переходов состояний. Однако, она имеет дополнительный столбец для суммы ненулевых весов по строке. Каждая ячейка на пересечении строки и столбца отражает число переходов между соответствующими состояниями. При этом учитываются все переходы между парой состояний, за исключением петлевых переходов. Итог представлен в таблице 4.

Таблица 4.

Достижимое сост. Начальное сост.	q0	q1	q2	q3	q4	q5	w(p)
q0		1	1	1	1	1	5
q1	1		2	1	1		4
q2	1	2		1			3
q3	1	1	1		1	1	5
q4	1	1		1		1	4
q5	1			1	1		3

Приступаем к кодированию состояний. Согласно таблице 4, максимальный вес 2 имеет пара (q1, q2). Далее определим пары, содержащие одно из этих состояний и имеющее ненулевое значение веса:

$$(q0, q1), (q0, q2), (q1, q3), (q1, q4), (q2, q3)$$

Вычислим сумму весов для каждой пары по последнему столбцу из таблицы 4:

$$(q0, q1) = w(q0) + w(q1) = 5 + 4 = 9,$$

$$(q0, q2) = w(q0) + w(q2) = 5 + 3 = 8,$$

$$(q1, q3) = w(q1) + w(q3) = 4 + 5 = 9,$$

$$(q1, q4) = w(q1) + w(q4) = 4 + 4 = 8,$$

$$(q2, q3) = w(q2) + w(q3) = 3 + 5 = 8.$$

Выбираем любую из пар с максимальной суммой – например, (q0, q1). Общим состоянием для пар (q1, q2) и (q0, q1) является q1. Ему присваивается код 000.

Далее кодируем q2 – партнёра по паре с q1. Возможные коды с расстоянием Хэмминга 1:

$$\text{Относительно } q1 (000): 001, 010, 100$$

Проверив по таблице 4 существование ненулевого значения в ячейке (q1, q2), понимаем, что все варианты допустимы. Выбираем, например, 001.

Следующее состояние – q_0 , так как оно является единственным незакодированным из выбранных пар (q_1, q_2) и (q_0, q_1) . Возможные кандидаты:

Относительно q_1 (000): 010, 100

Относительно q_2 (001): 101, 011

Проверяем допустимость по таблице 4 – связи состояний (q_1, q_0) и (q_2, q_0) существуют. Рассчитаем по формуле (7):

$$S(100) = w(q_1, q_0)d(000, 100) + w(q_2, q_0)d(001, 100) = 1 * 1 + 1 * 2 = 3,$$

$$S(010) = w(q_1, q_0)d(000, 010) + w(q_2, q_0)d(001, 010) = 1 * 1 + 1 * 2 = 3,$$

$$S(101) = w(q_1, q_0)d(000, 101) + w(q_2, q_0)d(001, 101) = 1 * 2 + 1 * 1 = 3,$$

$$S(011) = w(q_1, q_0)d(000, 011) + w(q_2, q_0)d(001, 011) = 1 * 2 + 1 * 1 = 3.$$

Все варианты равнозначны, поэтому можно выбрать любой. Например, закодируем q_0 значением 100. Переходим к следующему этапу кодирования. необходимо выбрать очередное состояние для кодировки. Для этого определим пары состояний, содержащие уже закодированные – q_1, q_2 и q_0 . Получаем:

$(q_0, q_3), (q_0, q_4), (q_0, q_5), (q_1, q_3), (q_1, q_4), (q_2, q_3)$

Вычислим суммы весов по таблице 4 и получаем значения 10, 9, 8, 9, 8 и 8 соответственно. Выбираем пару с максимальной суммой – (q_0, q_3) . Следовательно, кодируем состояние q_3 . Возможные коды для q_3 :

Относительно q_0 (100): 110,

Относительно q_1 (000): 010,

Относительно q_2 (001): 101, 011

Проверяем допустимость по таблице 4 – связи состояний $(q_0, q_3), (q_0, q_1)$ и (q_0, q_2) существуют. Выполняем расчёт по формуле (7):

$$S(010) = w(q_1, q_3)d(000, 010) + w(q_2, q_3)d(001, 010)$$

$$+ w(q_0, q_3)d(100, 010) = 1 * 1 + 1 * 2 + 1 * 2 = 5,$$

$$S(101) = w(q_1, q_0)d(000, 101) + w(q_2, q_0)d(001, 101)$$

$$+ w(q_0, q_3)d(100, 101) = 1 * 2 + 1 * 1 + 1 * 1 = 4,$$

$$S(011) = w(q_1, q_0)d(000, 011) + w(q_2, q_0)d(001, 011)$$

$$+ w(q_0, q_3)d(100, 011) = 1 * 2 + 1 * 1 + 1 * 3 = 6,$$

$$S(110) = w(q1, q0)d(000, 110) + w(q2, q0)d(001, 110) + w(q0, q3)d(100, 110) = 1 * 2 + 1 * 3 + 1 * 1 = 6.$$

Минимальное значение в расчётах было получено от кода 101. Присваиваем его состоянию $q3$. Продолжаем по аналогии. Находим следующие пары с уже закодированными состояниями:

$$(q0, q4), (q0, q5), (q1, q4), (q3, q4), (q3, q5)$$

Вычисляем суммы весов и получаем значения 9, 8, 8, 9 и 8 соответственно. Выбираем любую пару с максимальным весом, например, $(q0, q4)$. Следовательно, следующим кодируем $q4$. Доступные варианты кодов:

Относительно $q0$ (100): 110,

Относительно $q1$ (000): 010,

Относительно $q2$ (001): 011,

Относительно $q3$ (101): 111.

Снова проверим допустимость кандидатов для состояния $q4$. По таблице 4 видно, что связи между $q2$ и $q4$ не существует, поэтому 011 исключается из кандидатов. Проведём расчёты по формуле (7) с оставшимися кандидатами:

$$\begin{aligned} S(111) &= w(q1, q4)d(000, 111) \\ &+ w(q0, q4)d(100, 111) + w(q3, q4)d(101, 111) = 1 * 3 + 1 * 2 + 1 * 1 = 6, \\ S(010) &= w(q1, q4)d(000, 010) \\ &+ w(q0, q4)d(100, 010) + w(q3, q4)d(101, 010) = 1 * 1 + 1 * 2 + 1 * 3 = 6, \\ S(110) &= w(q1, q4)d(000, 110) \\ &+ w(q0, q4)d(100, 110) + w(q3, q4)d(101, 110) = 1 * 1 + 1 * 2 + 1 * 2 = 5. \end{aligned}$$

Минимальное значение получилось у кода 110. Присваиваем его состоянию $q4$. Переходим к финальной итерации. Здесь уже можно обойтись без поиска пар и расчёта весов, перейдя сразу к подбору кандидатов для оставшегося состояния $q5$:

Относительно $q0$ (100): —,

Относительно $q1$ (000): 010,

Относительно $q2$ (001): 011,

Относительно $q3$ (101): —,

Относительно q_4 (110): 111.

По таблице 4 видно, что связи между q_5 и состояниями q_1 и q_2 не существует. Следовательно, коды 010 и 011 исключаются. Таким образом, остаётся единственный допустимый вариант – 111. Итого получаем результат:

q_0 : 100,

q_1 : 000,

q_2 : 001,

q_3 : 101,

q_4 : 110,

q_5 : 111.

1.2.7 Генетический алгоритм

Для наглядности рассмотрим одну итерацию генетического алгоритма. Пусть начальная популяция будет состоять из 3 случайных кодировок, как показано в таблице 5:

Таблица 5

Состояние	Кодировка 1	Кодировка 2	Кодировка 3
q_0	101	001	111
q_1	011	010	100
q_2	001	100	000
q_3	000	111	110
q_4	110	101	001
q_5	010	000	101

Для каждой кодировки вычисляется логическая сложность функции переходов, определяемая количеством термов в функции возбуждения с дальнейшей минимизацией по методу Куайна-Мак-Класки. Опустим шаги с расчётами и результатами минимизации. Кодировка 1 получила оценку 74, Кодировка 2 – 106, а Кодировка 3 – 77. Итого лучшими считаются Кодировка 1 и 3.

Применим односточным кроссинговер. Точка кроссинговера будет выбираться случайным образом. Пусть Родителем 1 будет Кодировка 1, а

Родителем 2 – Кодировка 3. Например, пусть точка кроссинговера проходит между первым и вторым битом. Возьмём коды состояния q_0 у обоих родителей – это 101 и 111. Итого получаем после процесса кроссинговера коды у потомков для q_0 – 111 и 101. Также применим мутацию к нулевому биту Потомка 1 и получим 110. Аналогично кодируются оставшиеся состояния со случайным выбором точки кроссинговера и малой долей вероятности мутации. Получаем потомков, показанных в таблице 5.1.

Таблица 5.1

Состояние	Потомок 1	Потомок 2
q_0	110	101
q_1	010	001
q_2	000	111
q_3	001	110
q_4	111	000
q_5	100	010

Оценим полученные кодировки с аналогичным опущением подробных расчётов. Итого получаем, что Потомок 1 получил оценку 77, а Потомок 2 – 91. Результат оценки Потомка 2 является наибольшим из множества допустимых вариантов, а именно: Кодировка 1 (Родитель 1) – 74, Кодировка 3 (Родитель 2) – 77, Потомок 1 – 77. Поэтому в это множество вариант кодировки Потомка 2 не вносится.

Итого получаем, что наименьшее значение оценки имеет Потомок 1 и Родитель 2 – 77. Так как они равноценны, итоговый вариант можно выбрать любой, так что выберем Родителя 2:

q_0 : 001,

q_1 : 010,

q_2 : 100,

q_3 : 111,

q_4 : 101,

q_5 : 000.

1.3 Форматы описания конечных автоматов

Для представления конечных автоматов в цифровой обработке и проектировании цифровых устройств используются стандартизированные форматы описания. Они позволяют однозначно задавать структуру автомата, включая состояния, переходы, входные и выходные сигналы. Основными форматами описания конечных автоматов являются:

- Verilog. Относится к языкам описания аппаратуры и широко применяется в HDL-проектировании. Описывает автомат в виде состояний и переходов с условиями.
- JFF (JFLAP File Format). Формат на основе XML, используемый в программе JFLAP. Описывает автомат как граф.
- SMF (State Machine Format). Упрощённый текстовый формат для задания графа автомата.

1.4 Обзор программных средств, выполняющих автоматическое кодирование состояний

Для учёта существующего опыта решения задачи автоматического кодирования состояний конечных автоматов целесообразно провести обзор программных средств, выполняющих данную функцию. Наиболее распространёнными и функционально близкими являются среды Quartus II от Altera (Intel) и Xilinx Vivado от AMD (Xilinx).

1.3.1 Quartus II (Altera Corporation)

Quartus II – интегрированная среда проектирования цифровых устройств на базе программируемых логических интегральных схем (ПЛИС). Платформа предоставляет набор инструментов для проектирования, моделирования, синтеза и оптимизации цифровых схем, включая реализацию конечных автоматов. Поддерживает следующие методы кодирования состояний: унитарный код (One-Hot), минимальное бинарное (Binary), кодирование Грея (Gray), Джонсона (Johnson), автоматический выбор метода кодирования на основе встроенных эвристик (Auto), а также возможность ручного задания кодов пользователем (User-Encoded).

Пользователь может описать конечный автомат следующими способами:

- HDL-описание (Verilog/VHDL). Автомат задаётся в виде процессов (always в Verilog или process в VHDL) с явным перечислением состояний и переходов.
- Графический редактор State Machine Viewer. Позволяет визуально редактировать граф автомата с автоматической генерацией HDL-кода.

После компиляции (Analysis & Synthesis) среда предоставляет таблицу соответствия состояний и их кодов в разделе State Machine Viewer с отображением выбранных кодов.

1.3.2 Xilinx Vivado (AMD/Xilinx Inc.)

Xilinx Vivado – современная интегрированная среда разработки для ПЛИС Xilinx/AMD. Поддерживает аналогичные Quartus II методы кодирования, за исключением кодирования Джонсона и ручного задания кодов состояний. В Vivado конечный автомат можно описать аналогично Quartus II с помощью HDL-описания. Однако полноценного графического редактора нет, но включена поддержка визуализации автоматов.

После синтеза в отчётах можно найти информацию о выбранном кодировании. А State Machine Viewer показывает текущие состояния и переходы.

1.3.3 Сравнительная характеристика

Для наглядности проведём сравнительный анализ программных средств по ключевым параметрам. Это отражено в таблице 6.

Таблица 6.

Программное средство	Поддерживаемые методы кодирования	Входные форматы описания автоматов	Выходные форматы описания автоматов
Quartus II	One-Hot, Gray, Johnson, Sequential, Minimal Bits, Auto, User-encoded	Verilog HDL, VHDL, SystemVerilog, State Machine Format, TCL-скрипты	Отчёты синтеза (Text/HTML), RTL-схема с экспортом в PNG/SVG, VWF, Verilog/VHDL testbench, HEX, State Machine Format
Xilinx Vivado	One-Hot, Gray, Sequential, Minimal Bits, Auto	Verilog, VHDL, SystemVerilog, FSM Viewer, HLS с автоматическим преобразованием в RTL, XDC	Отчёты синтеза (Text/HTML), RTL-схема с экспортом в PNG, VCD, SAIF, XSIM, BIT, BIN, Verilog, VHDL, FSM

Таким образом, рассмотренные программные средства имеют общие ограничения:

- Отсутствие возможности настройки критериев и алгоритмов выбора кодировки;
- Невозможность прямой конвертации в различные форматы после кодирования состояний.
- Алгоритм кодирования не зависит от конкретного типа триггера.

Разрабатываемое программное средство устраняет эти недостатки за счёт возможности конфигурирования параметров оптимизации пользователем.

1.5 Техническое задание на разработку программного средства

Техническое задание составлено в соответствии с требованиями ГОСТ к содержанию и оформлению [10] с учётом следующих особенностей. Опущены следующие пункты:

- «Требования к программной документации» по причине того, что это не коммерческий продукт;
- «Технико-экономические показатели» по причине того, что экономические расчёты обычно требуются для коммерческих проектов с обоснованием ROI;
- «Стадии и этапы разработки», так как это будет приведено в следующем разделе;

- «Порядок и контроль приёмки», так как программное средство не подразумевает коммерческой поставки или договорных обязательств;
- «Требования к маркировке и упаковке», так как это программный инструмент;
- «Требования к транспортированию и хранению», так как программа не имеет физического носителя или специальных условий хранения.

1.5.1 Введение

Наименование: программное средство кодирования состояний конечных автоматов. Областью применения является синтез автоматов в виде интегральных схем.

1.5.2 Основания для разработки

Разработка программного средства выполняется в рамках выполнения выпускной квалификационной работы на тему «Программное средство кодирования состояний конечных автоматов».

1.5.3 Назначения разработки

Разрабатываемое программное средство предназначено для автоматизации процесса кодирования состояний конечных автоматов с использованием различных методов кодирования.

Функциональное назначение программного средства заключается в обеспечении возможности выбора метода кодирования и расчёта кодов состояний по заданным критериям.

1.5.4 Требования к программе

Разрабатываемое программное средство должно работать под управлением операционной системы Windows на персональных компьютерах и обеспечить загрузку файла с устройства. Поддерживаемые форматы исходных автоматов: .jff, .v, .smf. При ошибочной загрузке программа должна уведомлять пользователя соответствующим сообщением.

Интерфейс приложения должен быть интуитивно понятным, позволяя пользователю выбирать алгоритм кодирования. Если алгоритм допускает параметризацию, должно быть предусмотрено поле для ввода значения с

ограничением по диапазону, указанным в этом поле до ввода вместе со значением, применимым по умолчанию в отсутствии ввода.

После выполнения кодирования программа обязана отобразить результат в формате «Имя состояния – код». Также должна быть реализована функция сохранения результатов, включающая возможность модификации исходного файла или создания нового – в случае выбора другого формата – с добавлением строк, содержащих полученные кодировки в соответствии с синтаксисом выбранного формата. Итоговое приложение должно быть представлено в виде исполняемого файла.

Поддерживаемыми алгоритмами кодирования должны быть все вышеописанные алгоритмы.

Разрабатываемый интерфейс должен состоять из следующих элементов:

- Кнопка для загрузки файла;
- Выпадающий список для выбора алгоритма;
- Текстовое поле/поля для ввода параметров при необходимости;
- Кнопка запуска процесса кодирования с учётом выбранных параметров;
- Текстовое поле для вывода результатов кодирования и ошибочных сообщений;
- Кнопка для сохранения результатов.

Для корректной работы программного средства необходима следующая конфигурация:

Минимальные требования

- Операционная система: Windows 10 и выше
- Процессор: x86-64, 2+ ядра
- Оперативная память: 8 ГБ
- Свободное место на диске: 1 ГБ
- Графика: интегрированное графическое ядро

2 РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА

2.1 Выбор средств разработки

Для реализации программного средства автоматического кодирования состояний конечных автоматов был выбран высокоуровневый язык программирования Python. Его выбор обусловлен тем, что он прост в своём синтаксисе, гибок и имеет большое количество поддерживаемых библиотек. В качестве среды разработки был выбран PyCharm. Это интегрированная среда разработки, способная обеспечить удобный и интуитивно понятный интерфейс, поддерживать отладку, подсвечивать синтаксис для удобного программирования и выявления ошибок. Использование PyCharm позволило значительно упростить процесс разработки и тестирования программного продукта.

Основной используемой библиотекой для реализации графического интерфейса является PyQt5, которая предоставляет средства для создания оконных приложений на языке программирования Python. Её выбор обусловлен тем, что она поддерживает множество виджетов и позволяет создавать интерфейсы любой сложности.

Дополнительно в разработанной программе использовались следующие библиотеки:

- Xml.etree.ElementTree для парсинга и работы с файлами XML, с помощью которых описывается формат .jff, где и задаются графы конечных автоматов.
- re для работы с регулярными выражениями. Они используются для извлечения логических условий перехода в исходном конечном автомате и обработке файлов формата .v;
- tabulate для форматирования и отображения табличных данных в консольном виде, что удобно при отладке и анализе работы алгоритмов кодирования состояний конечного автомата;
- collections.defaultdict для удобной работы со словарями со значениями по умолчанию. Эти структуры используются, например, для

хранения графа переходов автомата или в реализации генетического алгоритма для хранения популяции кодировок;

- `random` для реализации операций мутации и кроссинговера в генетическом алгоритме, так как в них заложен принцип случайности;

- `math` для выполнения математических расчётов. Например, при определении минимального количества триггеров при кодировании состояний или же для тригонометрических расчётов позиций элементов, из которых создан фон приложения, для реализации галактики при расстановке звёзд по окружности;

- `Wuine_mccluskey.qm` для минимизации логических функций методов Куайна-Мак-Класки. Эта минимизация применяется на этапе выполнения генетического алгоритма для оценки качества кодировок через подсчёт минимального числа логических элементов.

- `time` для возможности замерять время выполнения алгоритмов.

- `itertools` для создания всех возможных комбинаций элементов из нескольких итерируемых объектов без необходимости писать вложенные циклы.

- `tracemalloc` для установки специальных точек в местах программы, которые вычисляют объём оперативной памяти между собой.

2.2 Схема работы программы

Схема работы программы продемонстрирована на рисунке 3 в виде графа. Он отображает последовательность логических этапов работы программного средства от момента запуска до завершения работы. Переходы между состояниями осуществляются по действиям пользователя или завершении соответствующего процесса. Отметим, что на рисунке 3 изображена основная работа программы. В действительности, в состояние «Ожидание выбора алгоритма» можно попасть из любого состояния, а закрыть программу можно в любой момент времени.

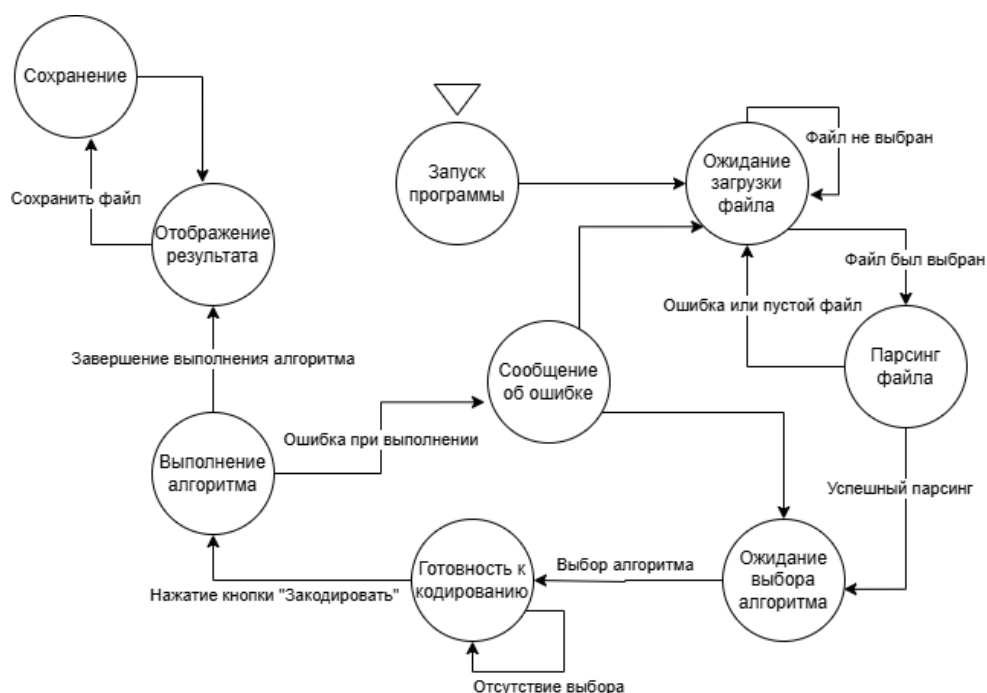


Рисунок 3 – Схема работы программы

2.3 Работа с входными и выходными файлами

Программа реализована в виде единого модуля, но логически разделена на несколько компонентов, взаимодействующих между собой.

2.3.1 Загрузка и парсинг файлов

Данный модуль отвечает за чтение и обработку входных файлов различных форматов: JFLAP .jff, Verilog .v и SMF .smf. Из этих файлов извлекается информация о состояниях и переходах конечного автомата. Модуль состоит из следующих компонентов.

1. **Метод load_file().** Основная функция загрузки файла, которая сбрасывает флаги, обращаясь к методу `reset_flags()`, и очищает поле вывода с помощью метода `output_text.clear()`. Также она открывает диалоговое окно для выбора файла и определяет тип файла по расширению и вызывает соответствующий парсер: для .jff – метод `parse_jff_file()`, .v – метод `parse_verilog_file()`, а для .smf – метод `parse_smf_file()`. Завершает свою работу формированием метода HTML-сообщения о результате загрузки (успех или ошибка) и выводит его в `output_text`.

В качестве вспомогательного метода использует `_add_transition()` для добавления переходов в формируемый граф. Также проверяет наличие

состояний и переходов после парсинга. Если их нет, выводит соответствующее сообщение в `output_text`.

2. Парсеры файлов.

– Метод `parse_jff_file()` – обрабатывает XML-файлы формата JFLAP. Структура файла устроена так, что состояния хранятся в тегах `<state>` с атрибутами `id`, `name` и вложенным тегом `<output>`. Переходы описываются в тегах `<transition>` с элементами `<from>`, `<to>`, `<read>` (условие перехода), `<transout>` (выход).

Для разбора XML используется библиотека `xml.etree.ElementTree`, в процессе чего извлекаются состояния в словарь `self.states` вида:

$$\{id: \{ 'name': str, 'output': str \} \}$$

Далее формируется список переходов `self.transitions` и граф `self.graph` в формате:

$$\{ 'from_state': [('to_state', 'condition'), \dots] \}$$

– Метод `parse_verilog_file()` – анализирует файлы Verilog. Структура файла устроена так, что состояния объявляются в блоке `parameter` с кодировками по умолчанию, отражающие последовательное кодирование. Переходы определяются в блоках `case...endcase` и условиях `if`.

Логика парсинга основана на использовании регулярных выражений для поиска состояний вида

$$re.search(r'parameter\s + (\^ ;] +);', content)$$

и переходов вида

$$re.finditer(r'if\s * \((.*?)\)\s * reg_fstate\s * <=\s * (\w+)\s *;', text).$$

Безусловные переходы обрабатываются отдельно. Они определяются строками, где значение регистра `reg_fstate` изменяется без условия. В таких случаях условием перехода считается константа «1».

– Метод `parse_smf_file()` – читает файлы в формате State Machine Format (SMF). Структура файлов устроена так, что состояния задаются в

блоках вида

$$STATE \{ NAME = "..."; STYPE = "..."; \},$$

а переходы – в блоках вида

$$TRANS \{ SSTATE = "from"; DSTATE = "to"; EQ = "condition"; \}.$$

Логика парсинга аналогично предыдущему методу основана на регулярных выражениях. Так, для извлечения состояний используется регулярное выражение вида:

$$re.compile(r'STATE\s * \{ \s * NAME\s * = \s * "([^\"]+)" \s * ; .*? STYPE\s * = \s * "([^\"]+)"', re.DOTALL),$$

а для извлечения переходов – выражение вида

$$re.compile(r'TRANS\s * \{ \s * SSTATE\s * = \s * "([^\"]+)" \s * ; \s * DSTATE\s * = \s * "([^\"]+)" \s * ; \s * EQ\s * = \s * "([^\"]+)"', re.DOTALL).$$

Для корректной работы парсеров программа предполагает, что входные файлы соответствуют следующим требованиям:

- файл формата .jff должен содержать валидный XML с тегами <state> и <transition>;
- файл формата .v должен быть описан через parameter и case/if;
- в файле формата .smf состояния и переходы должны строго следовать шаблону STATE {...} и TRANS {...}.

2.3.2 Сохранение результатов

Для сохранения результатов кодирования состояний конечного автомата реализованы следующие группы методов:

1. **Методы модификации исходных файлов.** Предназначены для добавления информации о кодировке состояний в файлы исходных форматов. `_save_modified_jff()` модифицирует JFF-файл, добавляя коды состояний в виде меток (label) к соответствующим узлам XML-структуры. `_save_modified_verilog()` обновляет блок parameter в Verilog-файле, заменяя значения состояний на их двоичные коды в десятичном представлении. `_save_modified_smf()` интегрирует коды состояний в SMF-файл, дополняя имена состояний в формате «Имя (Код)».

2. **Методы конвертации в другие форматы.** Обеспечивает генерацию файлов в форматах, отличных от исходного, с сохранением информации о кодировке. `_save_as_jff()` создаёт JFF-файл «с нуля», добавляя коды состояний как метки. `_save_as_verilog()` формирует полноценный Verilog-модуль, включая объявление входных сигналов, извлечённых из условий переходов, параметры состояний с десятичными значениями кодов, а также синхронную и комбинационную логику для обработки переходов. `_save_as_smf()` генерирует SMF-файл с визуальным расположением состояний по кругу и корректным отображением переходов.

3. **Управляющий метод `save_modified_file()`.** Координирует процесс сохранения:

- отображает диалоговое окно выбора формата;
- в зависимости от выбора пользователя вызывает соответствующий метод модификации или конвертации;
- обрабатывает ошибки и выводит уведомления о результате операции.

2.4 Алгоритмы кодирования состояний

Программная реализация включает 5 алгоритмов кодирования, каждый из которых содержит уникальные методы для обработки состояний и переходов конечного автомата. Ниже представлено их структурированное описание.

1. **Код Джонсона.** Реализовано с помощью метода `encode_johnson_code()`, внутри которого описана подготовка данных, проверка их корректности, генерация кодов и сопоставления состояний и кодов. Результаты передаются вспомогательному методу `show_encoded_states()` для отображения в интерфейсе.

2. **Код Грея.** Реализовано с помощью метода `encode_gray_code()`, структурно реализующий аналогичные с кодом Джонсона шаги и так же обращающегося к `show_encoded_states()` для отображения в интерфейсе.

3. **Унитарный код.** Реализован с помощью метода `one_hot_encoding()`, внутри которого идёт определение разрядности кодов и их генерация согласно правилам этого метода с выводом получившихся результатов.

4. **Последовательное кодирование.** Реализовано с помощью метода `encode_sequential()`, структурно реализующий с унитарным кодом аналогичные этапы.

5. **Метод кодирования с редукцией веса Хэмминга.** Метод описан с помощью следующих методов:

- Метод `extract_variables()`. Данный метод предназначен для извлечения переменных из логических выражений, описывающих условия переходов между состояниями. В зависимости от формата входных данных метод использует различные регулярные выражения для корректного выделения переменных. Результатом работы является множество переменных, участвующих в условиях переходов.

- Метод `find_successor_pairs()`. Этот метод выполняет поиск пар последователей, которые имеют хотя бы одну общую переменную в условиях переходов. Для каждой вершины графа состояний анализируются переходы, исключая петли (петлевые переходы), и формируются пары состояний, чьи условия переходов содержат пересекающиеся множества переменных. Результат сохраняется в виде словаря, где ключами являются исходные состояния, а значениями – списки пар последователей.

- Метод `pairs_finding()`. Метод осуществляет подсчёт количества повторений пар последователей, обнаруженных методом `find_successor_pairs()`. Для каждой пары проверяется, в скольких состояниях исходного автомата они встречаются вместе, и формируется таблица повторений. На основе этой таблицы выбираются пары с наибольшим количеством повторений, что является ключевым для дальнейшего кодирования.

– Метод *count_incoming_transitions()*. Данный метод подсчитывает количество входящих переходов для каждого состояния автомата. результаты представляются в виде таблицы, которая используется для определения приоритета кодирования состояний: вершины с большим числом входящих переходов кодируются первыми. Метод завершается вызовом метода *coding_states()*, передавая ему словарь с количеством входящих переходов.

– Метод *coding_states()*. Основной метод, реализующий алгоритм кодирования с редукцией веса Хэмминга. На основе данных о входящих переходах, полученных от метода *count_incoming_transitions()* и таблицы пар последователей, сформированной методом *pairs_finding()*, назначаются двоичные коды состояниям. Для кодирования используются вспомогательные методы: *find_next_code()* для нахождения наименьшего незанятого двоичного кода, а также *find_code_with_power_of_two_gap()*, обеспечивающий назначение кодов с интервалами, равными степеням двойки, что минимизирует вес Хэмминга между кодами связанных состояний.

Результатом работы метода является словарь, где каждому состоянию сопоставлен уникальный двоичный код.

6. Метод кодирования с редукцией расстояния Хэмминга.

Алгоритм включает в себя следующие методы:

– Метод *on_clicked()*. Является точкой входа для запуска процесса кодирования. метод инициализирует построение таблицы весов переходов, обращаясь к методу *build_weight_table()*, выводит её в консоль, обращаясь к *print_weight_table()*, а затем последовательно кодирует состояния с помощью метода *encode_states()*. Результаты кодирования сохраняются и выводятся в отсортированном виде.

– Метод *build_weight_table()*. Строит таблицу весов переходов между состояниями автомата. для каждого перехода, исключая петли, увеличивается вес соответствующей пары состояний. дополнительно вычисляется столбец $w(p)$, отражающий общее количество переходов для

каждого состояния. Результатом работы метода является таблица весов и список имён состояний.

- *Method print_weight_table()*. Вспомогательный метод, предназначенный для визуализации таблицы весов переходов в консоли. Использует библиотеку `tabulate` для форматированного вывода данных.

- *Method hamming_distance()*. Вычисляет расстояние Хэмминга между двумя двоичными кодами, определяя количество различающихся битов. Этот метод является ключевым для оценки качества кодирования и минимизации расстояния между связанными состояниями.

- *Method generate_codes()*. Генерирует все возможные двоичные коды заданной длины, которые используются для кодирования состояний автомата. длина кода определяется количеством состояний.

- *Method find_min_code()*. Находит оптимальный код для текущего состояния на основе минимизации суммы взвешенных расстояний Хэмминга $S(\alpha)$. Для этого анализируются коды ближайших соседей (отличающихся на один бит) уже закодированных состояний. Метод исключает коды, не имеющие переходов с текущим состоянием (вес $w(q, p) = 0$), и выбирает код с минимальной суммой $S(\alpha)$.

- *Method generate_neighbors()*. Генерирует все возможные коды, которые отличаются от заданного кода на один бит.

- *Method encode_states()*. Основной метод, реализующий алгоритм кодирования с редукцией расстояния Хэмминга. Процесс кодирования включает следующие этапы: построение таблицы весов переходов, поиск пары состояний с максимальным весом переходов для начального кодирования и последовательное кодирование оставшихся состояний с использованием метода `find_min_code()`, минимизирующий взвешенное расстояние Хэмминга.

Результатом работы метода является словарь, где каждому состоянию сопоставлен уникальный двоичный код.

7. **Генетический алгоритм.** Включает в себя следующие методы:

– Метод *generate_encodings()*. Является основным методом, реализующим генетический алгоритм. Выполняет следующие функции: генерацию начальной популяции случайных кодировок состояний, оценку качества каждой кодировки через подсчёт логических элементов через обращение к методу *count_logical_elements()*, селекцию двух лучших кодировок с помощью обращения к методу *find_best_encoding()*, применение оператором кроссинговера (*crossover*) и мутации (*mutate*) для создания новых поколений, а также контроль уникальности кодов, обращаясь к методу *ensure_unique_codes()*.

– Метод *count_logical_elements()*. Оценивает качество кодировки через расчёт количества логических элементов, необходимых для её аппаратной реализации. Использует метод Куайна-Мак-Класки для минимизации логических функций.

– Метод *find_best_encoding()*. Отбирает две лучшие кодировки из популяции на основе минимального количества логических элементов. Результаты сохраняются для последующего использования в операциях генетического алгоритма.

– Методы генетических операторов. К ним можно отнести *crossover()* – реализует одноточечный кроссинговер для создания новых кодировок; *mutate()* – выполняет мутацию кодов с заданной вероятностью; *crossover_and_mutate()* – комбинирует оба оператора для генерации потомков.

– Вспомогательные методы. К ним относятся: *ensure_unique_codes()* – гарантирует уникальность кодов в каждой кодировке; *find_unique_code()* – находит уникальный код при обнаружении дубликатов; *flip_random_bit()* – инвертирует бит в коде для обеспечения вариативности.

2.5 Реализация алгоритмов кодирования состояний

В данном разделе будут рассмотрены все реализованные алгоритмы кодирования состояний конечного автомата с программной точки зрения и подробно описываться особенности программной реализации каждого выбранного алгоритма.

2.5.1 Программная реализация унитарного кода

Алгоритм унитарного кода реализован в виде следующей последовательности методов.

1. *Подготовка данных.* Извлечение имён состояний из словаря `states` с последующим определением разрядности кодов, которое равно числу состояний `num_states`.

2. *Генерация кодов.* Для каждого состояния создаётся битовая строка, где единица стоит на позиции, соответствующей порядковому номеру состояния.

Шаги 3 и 4 являются аналогичными шагам из предыдущего подраздела.

2.5.2 Программная реализация последовательного кодирования

1. *Подготовка данных.* После извлечения имён состояний из словаря `states` идёт их сортировка в алфавитном порядке. Затем вычисляется разрядность кодов.

2. *Генерация кодов.* Каждому состоянию присваивается двоичный код его порядкового номера.

3. Шаги 3 и 4 являются аналогичными шагам из подраздела 2.4.2.

2.5.3 Программная реализация кодирования Джонсона

Алгоритм кодирования Джонсона реализован в виде следующей последовательности.

1. *Подготовка данных.* Здесь извлекаются имена состояний из структуры автомата и сортируются в алфавитном порядке или по возрастанию. Далее определяется количество состояний `num_states` и вычисляется минимально необходимое количество бит `num_bits` по формуле:

$$num_bits = (state_names - 1).bit_length()$$

Это гарантирует возможность представления всех состояний уникальными кодами.

2. *Генерация кодов Джонсона.* Инициализируется начальный код, состоящий из всех нулей. Для каждого состояния выполняется:

– Сохранение текущего кода

- Циклический сдвиг вправо
- Инверсия первого бита сдвинутого кода

3. *Назначение кодов состояниям.* Создаётся словарь `coded_states`, где ключами являются имена состояний, а значениями – соответствующие коды Джонсона. Результат сохраняется в атрибуте класса `coded_states`.

2.5.4 Программная реализация кодирования Грея

Алгоритм кодирования Грея реализован в виде следующей последовательности.

1. *Подготовка данных.* Извлекаются и сортируются имена состояний из структуры автомата строкой вида:

```
state_names = sorted([state_data['name'] for  
state_data in self.states.values()])
```

Затем определяется количество состояний `num_states` и вычисляется минимально необходимое количество бит `num_bits`, аналогично программной реализации кодирования Джонсона.

2. *Генерация кодов Грея.* Реализована локальная функция преобразования числа в код Грея `gray_code(n)`:

```
return n ^ (n >> 1)
```

Таким образом, для каждого состояния генерируется соответствующий код Грея.

3. *Назначение кодов состояниям.* Создаётся словарь `coded_states`, где ключами являются имена состояний, а значениями – соответствующие коды Грея. Результат сохраняется в атрибуте класса `coded_states`.

2.5.5 Программная реализация метода с редукцией веса Хэмминга

Алгоритм редукции веса Хэмминга реализован в виде следующей последовательности.

1. *Извлечение переменных из условий переходов.* Метод `extract_variables()` анализирует строку условия перехода и извлекает переменные в зависимости от формата входного файла с помощью регулярных выражений.

2. *Поиск пар последователей.* Метод `find_successor_pairs()` обрабатывает граф переходов, исключая петли, и формирует пары вершин, в которые возможен переход из текущего состояния. Пары считаются валидными, если их условия переходов содержат общие переменные (проверка через пересечение множеств).
3. *Подсчёт повторяющихся пар.* В методе `pairs_finding()` для каждой пары вершин подсчитывается количество повторений – число состояний, из которых возможен переход в обе вершины пары с условиями одинаковой сложности (по количеству переменных). Результат выводится в виде таблицы, содержащей пары и частоту их повторений.
4. *Анализ входящих переходов.* Метод `count_incoming_transitions()` строит таблицу, отображающую количество переходов в каждую вершину. Это позволяет определить приоритеты при кодировании: вершины с большим числом входящих переходов кодируются первыми.
5. *Кодирование состояний.* основная логика в методе `coding_states()`:
 - Количество бит вычисляется как $bit_width = (N - 1).bit_length()$, где N – число состояний.
 - Приоритет отдаётся вершинам с максимальным числом входящих переходов.
 - Если вершина входит в пару с ненулевым числом повторений, её код назначается с учётом кода парной вершины: для новых пар используется минимальный свободный код и код со смещением на степень двойки (для минимизации веса Хэмминга). Если одна вершина пары уже закодирована, вторая получает код на основе `find_code_with_power_of_two_gap()`, обеспечивающий расстояния Хэмминга, равное степени двойки.
 - Одиночные вершины кодируются отдельно, начиная с кода $0 \dots 0$.Итоговые коды состояний выводятся в виде таблицы, где каждому состоянию сопоставлен уникальный бинарный код.

2.5.6 Программная реализация метода с редукцией расстояния

Хэмминга

Алгоритм редукции расстояния Хэмминга реализован в виде следующей последовательности.

1. *Построение таблицы весов.* Метод `build_weight_table()` формирует симметричную таблицу весов переходов между состояниями.

- Для каждого перехода ($p \rightarrow q$) инкрементируется значение `weight_table[p][q]` и `weight_table[q][p]` (если $p \neq q$).

- Добавляется столбец $w(p)$ – суммарное количество переходов из состояния p . Результат выводится в виде таблицы, где строки и столбцы соответствуют состояниями, а ячейки содержат частоту переходов.

2. *Вычисление расстояния Хэмминга.* Метод `hamming_distance()` принимает два бинарных кода и возвращает количество различающихся битов. Используется для оценки оптимальности кодирования.

3. *Генерация кодов и соседей.*

- `generate_codes` создаёт все возможные бинарные коды заданной длины (например, для 3 бит: 000, 001, ..., 111).

- `generate_neighbors` возвращает коды, отличающиеся от исходного ровно на один бит (например, соседи 001 – 101, 011, 000).

4. *Поиск оптимального кода для состояния.* Метод `find_min_code()` определяет код для состояния p по следующим правилам:

- Кандидаты – коды, являющиеся соседями уже закодированных состояний и не назначенные ранее.

- Фильтрация. Исключаются коды, сгенерированные из состояний q , для которых вес перехода $w(q, p) = 0$.

- Критерий выбора: минимизация суммы из формулы (7).

5. *Кодирование состояний.* Основная логика в методе `encode_states()`:

- Инициализация. Определяется разрядность кода `num_bits` на основе количества состояний.

– Первая пара: кодируются два состояния с максимальным весом перехода между ними (коды 000...0 и 000...1).

– Итеративное кодирование. Для каждого не закодированного состояния p выбирается состояние q с максимальным $w(q, p)$ и суммой весов $w(p) + w(q)$, после чего назначается код через `find_min_code()`.

2.5.7 Программная реализация генетического алгоритма

Генетический алгоритм реализован в виде следующей последовательности.

1. *Генерация начальной популяции.* Метод `generate_encodings()` создаёт случайные кодировки:

– Разрядность кода определяется как $\text{ceil}(\log_2(N))$, где N – количество состояний.

– Для каждого состояния генерируется случайный бинарный код, проверяется его отсутствие в уже использованных (`used_codes`).

2. *Оценка приспособленности.* Метод `count_logical_elements()` вычисляет «стоимость» кодировки:

– Для каждого бита кода состояния формируются минтермы на основе условий переходов и текущих кодов состояний.

– Используется алгоритм Квайна-Мак-Класки для упрощения булевых функций [11].

– Суммируются AND- и OR-элементы после минимизации. Чем меньше сумма – тем лучше кодировка.

3. *Селекция.* Метод `dinf_best_encoding` выбирает две лучшие кодировки из популяции. Критерием выбора является минимальное количество логических элементов. Лучшие кодировки сохраняются в `best_parents` для дальнейшего скрещивания.

4. *Скрещивание и мутация.* Методы `crossover()` и `mutate()` создают новое поколение:

– Кроссинговер: одноточечное скрещивание, пример которого приведён на рисунке 1.

- Мутация: инверсия случайного бита с вероятностью 10%.
- Если кодировка потомков совпадает, у одного из них инвертируется случайный бит методом `flip_random_bit`.

5. *Коррекция дубликатов.* Метод `ensure_unique_codes()` устраняет конфликты: если код дублируется, последовательно инвертируются биты до нахождения свободного варианта.

Алгоритм повторяет эти шаги заданное число раз `max_iterations`, сохраняя лучшие решения.

Полный код разработанного программного средства представлен в Приложении В.

2.6 Разработка пользовательского интерфейса

Структуру проекта интерфейса можно описать следующими словами. Главное окно содержит в себе три ключевые зоны, а именно: анимированный фон, в котором реализован эффект вращающихся звёзд и плавные градиентные шлейфы, исходящие от этих звёзд; панель управления, расположенная в верхней части окна, которая состоит из кнопки загрузки файла, выпадающего списка для выбора алгоритма, а также поле ввода числа итераций, появляющееся лишь по событию, когда выбирается генетический алгоритм; зона вывода, располагающаяся в нижней части окна, куда выводится текстом результат загрузки файла и сообщения в случае ошибок ввода или загрузки, а также непосредственно результат кодирования состояний конечных автоматов. Внешний вид проекта интерфейса показан на рисунке 4.

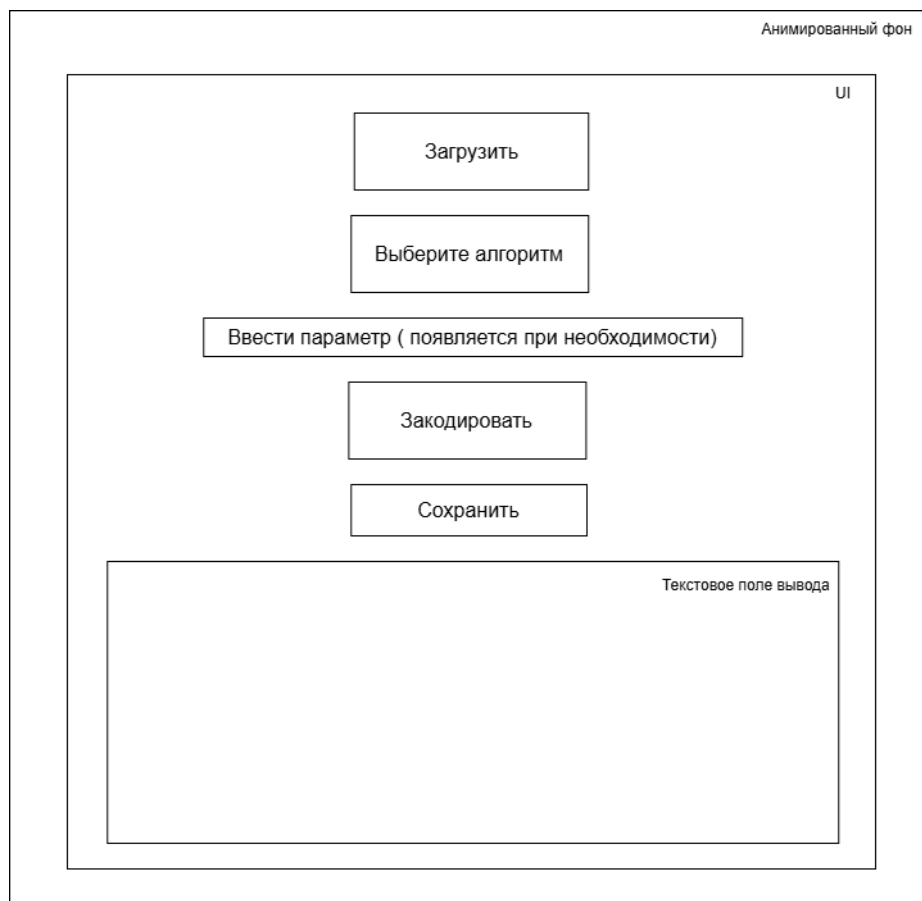


Рисунок 4 – Проект интерфейса

На текстовом поле также размещена кнопка, с помощью которой пользователь может сохранить полученный результат. При наведении вокруг неё появляется свечение.

Элементами управления разработанной программы выступают выпадающие списки, которые имеют скруглённые углы и дополнены анимацией-свечением при наведении. Также элементами управления являются кнопки, которые являются интерактивными с точки зрения дизайна. Это проявляется с помощью изменения цвета или появления тени при наведении.

Общий дизайн интерфейса выдержан в сиренево-фиолетовой гамме с добавлением розовых и белых оттенков при взаимодействии с элементами управления. Стилизация выполнена через Qt Style Sheets (QSS) – это CSS-подобный язык для элементов интерфейса PyQt5.

Графический интерфейс программы включает в себя следующие компоненты.

1. *Класс GalaxyBackground.* Данный класс является пользовательским виджетом, отвечающим за отрисовку анимированного фона с эффектом движущегося звёздного поля. Состоит из нескольких методов, а именно:

- `__init__()` – инициализирует параметры фона: количество звёзд, их радиус, размер, яркость и эффект «следа» от звёзд. Помимо этого, запуска таймер для обновления анимации.

- `Update_stars()` – обновляет позиции звёзд и их следов, вызывается таймером каждые 50 мс. Связан с методом `paintEvent()` через вызов `self.update()`.

- `paintEvent()` – создаёт фон, звёзды и их следы с использованием интерполяции цветов (от белого к розовому) и эффектом затухания и прозрачности.

2. *Класс Ui_Coding.* Класс отвечает за настройку и управление элементами интерфейса. Включает вложенный класс `Styles`, содержащий CSS-стили для кнопок, поля ввода и других элементов. Состоит из следующих методов:

- `setupUi()` – настраивает главное окно, создаёт и размещает элементы интерфейса:

- кнопку `pushButton`, функционально связанную с методом `load_file()` для загрузки файла,

- выпадающий список `comboBox_algorithm`, связанный с методом `on_algorithm_selected()` для выбора алгоритма кодирования,

- поле `iteration_input` для ввода числа итераций. Оно отображается или скрывается в зависимости от выбранного алгоритма,

- текстовое поле `output_text` для вывода результатов кодирования или сообщений об ошибках и успешной загрузке файла,

- кнопку `encodeButton`, функционал которой связан с методом `on_encode_clicked()` для запуска процесса кодирования,

- кнопку `save_button`, которая функционально связана с методом `save_modified_file()` для сохранения результатов в файл,

- `on_algorithm_selected()` – обработчик выбора алгоритма, управляет видимостью поля `iteration_input`,
- `on_encode_clicked()` – иницирует кодирование на основе выбранного алгоритма и введённых параметров, выводит результат в `output_text` и активирует кнопку `save_button`.

Анимация фона, описанная в классе `GalaxyBackground`, работает независимо, но интегрирована в главное окно через `setCentralWidget()`. Пользовательские действия, такие как нажатия кнопок и выбор алгоритма вызывают соответствующие методы, которые изменяют состояние интерфейса, например отображение поля `iteration_input` или запуск процесса кодирования. Результаты кодирования выводятся в `output_text`, после чего становится доступной кнопка сохранения `save_button`.

3 ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

В данном разделе описывается тестирование разработанного программного средства для кодирования состояний конечных автоматов. Раздел содержит в себе практическую демонстрацию работы с различными типами входных данных.

Оценка качества тестирования осуществляется путём сопоставления результатов автоматизированного и ручного кодирования, выполненного в подразделе 1.2, а также анализа корректности формирования итогового результата. При этом учитывается, что в ряде случаев возможны расхождения между результатами тестов и примерами ручного выполнения. Такие расхождения обусловлены случайным выбором из множества допустимых вариантов кодировок, которые являются равноценными по критериям условий алгоритма. В рамках практического тестирования предусмотрена возможность многократного применения процедуры кодирования, при которой случайным образом выбираются равноценные варианты кодировок. Это позволяет убедиться в наличии всех допустимых решений и корректности механизма случайного выбора, реализованного в программе.

Особое внимание будет уделено сравнительному анализу результатов, полученных разными методами, а также проверке устойчивости системы к граничным и ошибочным случаям.

3.1 Проверка работоспособности алгоритмов на примере автомата Мура

После запуска программы открывается окно (рисунок 5). Реализация пользовательского интерфейса соответствует схеме, изображённой на рисунке 4.

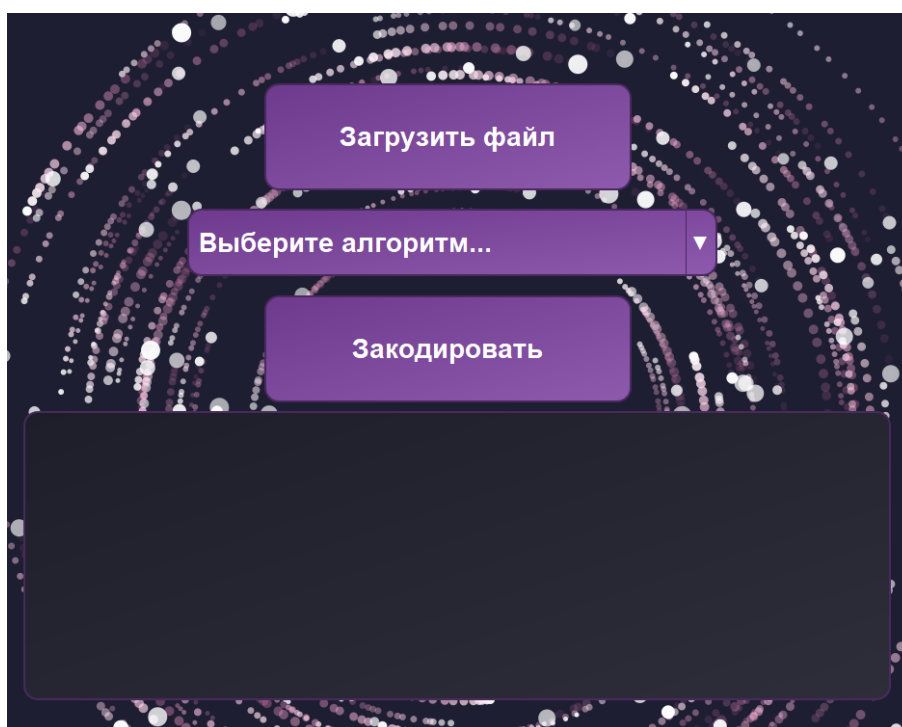


Рисунок 5 – Главный экран приложения

Для начала необходимо загрузить исходный файл, в котором содержится граф конечного автомата. Проведём тесты для автомата, который использовался в подразделе 1.2, описывающем ручное кодирование, и изображён на рисунке 2. После нажатия на соответствующую кнопку, открывается проводник, где можно выбрать исходный файл. Это показано на рисунке 6.

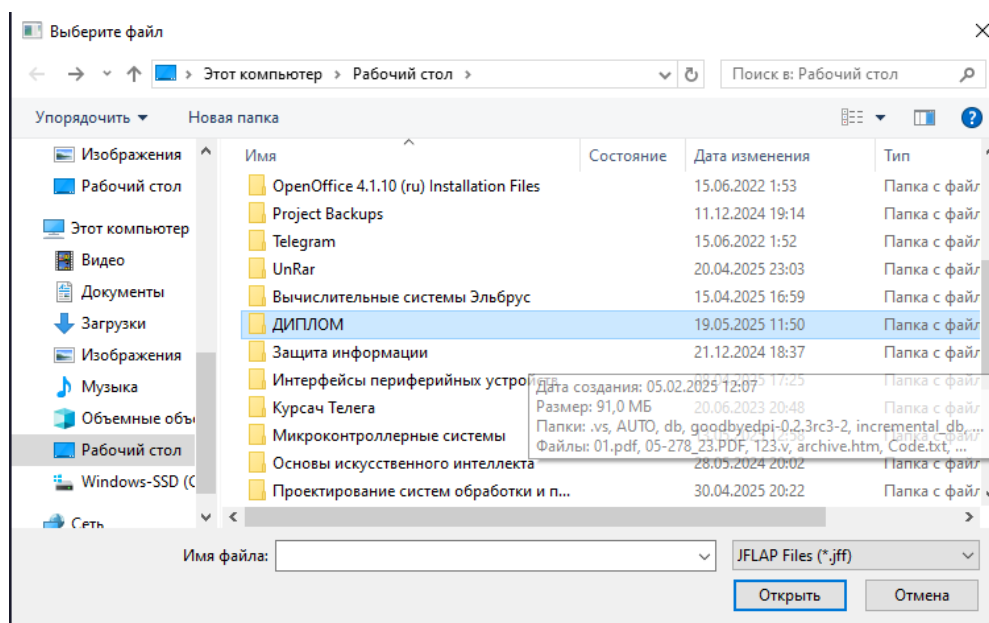


Рисунок 6 – Открытие проводника

После выбора файла в текстовое поле в приложении выводится оповещающее об удачной загрузке оповещение. Это показано на рисунке 7.

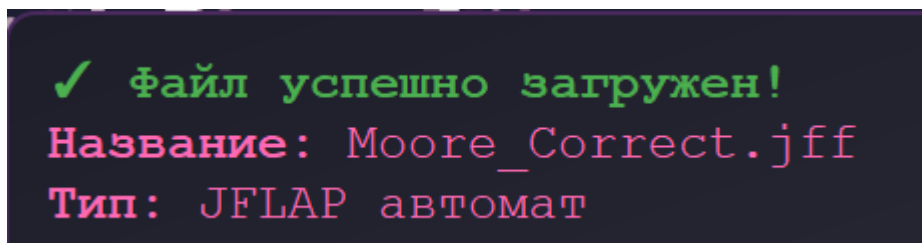


Рисунок 7 – Сообщение об удачной загрузке файла

Приступаем к этапу задания условий кодирования. Необходимо выбрать тип алгоритма. Нажимая на соответствующую кнопку, можно наблюдать выпадающий список с вариантами выбора. Это показано на рисунке 8.

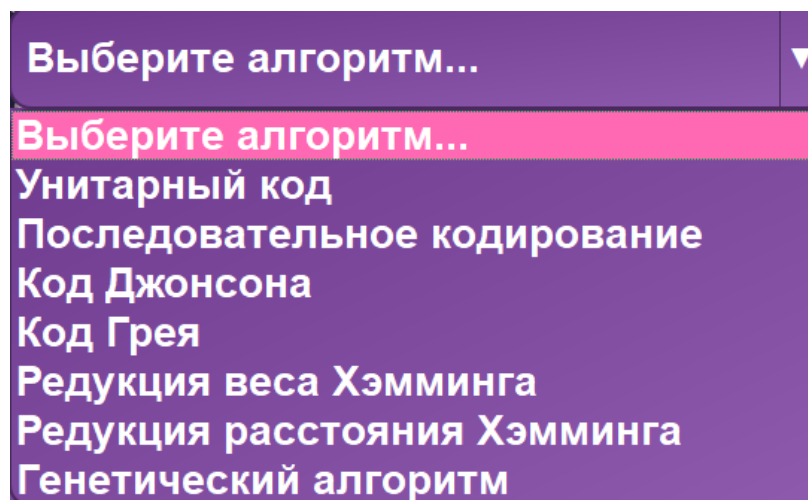


Рисунок 8 – Выбор алгоритма

3.1.1 Унитарный код

Результат работы унитарного кодирования представлен на рисунке 9.

```
q0: 000001
q1: 000010
q2: 000100
q3: 001000
q4: 010000
q5: 100000
Время выполнения кодирования: 50.18878 мс.
Количество логических элементов: 149
Количество триггеров: 6
```

Рисунок 9 – Результаты кодирования (унитарный код)

3.1.2 Последовательное кодирование

Результат работы последовательного кодирования представлен на рисунке 10.

```
q0: 000
q1: 001
q2: 010
q3: 011
q4: 100
q5: 101
Время выполнения кодирования: 33.40960 мс.
Количество логических элементов: 76
Количество триггеров: 3
```

Рисунок 10 – Результаты кодирования (последовательное кодирование)

3.1.3 Код Джонсона

Результат кодирования Джонсона показан на рисунке 11.

```
q0: 000
q1: 001
q2: 011
q3: 111
q4: 110
q5: 100
Время выполнения кодирования: 50.25339 мс.
Количество логических элементов: 58
Количество триггеров: 3
```

Рисунок 11 – Результаты кодирования (начало, кодирование Джонсона)

Результаты кодирования при смене формата загружаемого графа автомата остаются неизменными, что можно сказать и о кодировании Грея, унитарном и последовательном кодированиях.

3.1.4 Код Грея

Результат кодирования Грея продемонстрирован на рисунке 12.

```
q0: 000
q1: 001
q2: 011
q3: 010
q4: 110
q5: 111
Время выполнения кодирования: 33.62083 мс.
Количество логических элементов: 67
Количество триггеров: 3
```

Рисунок 12 – Результат кодирования (кодирование Грея)

3.1.5 Метод с редукцией веса Хэмминга

Выберем алгоритм с редукцией веса Хэмминга. Полученные результаты кодирования продемонстрированы на рисунках 13 – 14.

```
Выбранный алгоритм наиболее эффективен при использовании
D-триггеров
-----
Итоговые закодированные состояния:
q0: 010
q1: 101
q2: 001
q3: 000
q4: 100
```

Рисунок 13 – Результаты кодирования (начало, метод с редукцией веса Хэмминга)

```
q0: 010
q1: 101
q2: 001
q3: 000
q4: 100
q5: 011
Время выполнения кодирования: 55.23539 мс.
Количество логических элементов: 67
Количество триггеров: 3
```

Рисунок 14 – Результаты кодирования (окончание, метод с редукцией веса Хэмминга)

3.1.6 Метод с редукцией расстояния Хэмминга

Результаты работы алгоритма с редукцией расстояния Хэмминга показаны на рисунках 15 – 16.

```
Выбранный алгоритм наиболее эффективен при использовании
ЖК- и Т-триггеров

-----
Итоговые закодированные состояния:
q0: 011
q1: 000
q2: 001
q3: 010
q4: 110
```

Рисунок 15 – Результаты кодирования (начало, метод с редукцией расстояния Хэмминга)

```
q0: 011
q1: 000
q2: 001
q3: 010
q4: 110
q5: 111
Время выполнения кодирования: 62.65259 мс.
Количество логических элементов: 59
Количество триггеров: 3
```

Рисунок 16 – Результаты кодирования (окончание, метод с редукцией расстояния Хэмминга)

Результат работы алгоритма в случае, когда был загружен граф автомата в формате v., показан на рисунке 17.

```
Итоговые закодированные состояния:
q0: 101
q1: 000
q2: 001
q3: 100
q4: 110
q5: 111
Время выполнения кодирования: 34.74784 мс.
Количество логических элементов: 19
Количество триггеров: 3
```

Рисунок 17 – Результаты кодирования (метод с редукцией расстояния Хэмминга)

3.1.7 Генетический алгоритм

Выберем генетический алгоритм. При этом становится видимым текстовое поле, внутри которого указан параметр для ввода. Если ничего не ввести, будет использоваться значение по умолчанию. Это показано на рисунке 18. Результаты показаны на рисунке 19.

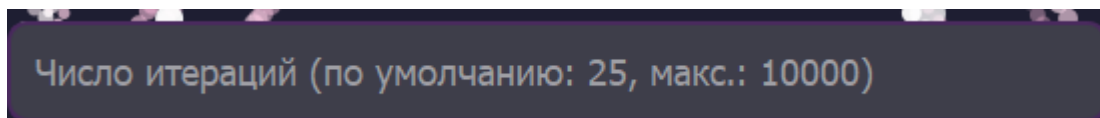


Рисунок 18 – Появившееся поле для ввода

```
q0: 000
q1: 110
q2: 100
q3: 101
q4: 011
q5: 001
Время выполнения кодирования: 4.04579 с.
Количество логических элементов: 53
Количество триггеров: 3
```

Рисунок 19 – Результаты кодирования (генетический алгоритм)

Результат работы алгоритма в случае, когда был загружен граф автомата в формате v., показан на рисунке 20.

```
q0: 011
q1: 010
q2: 101
q3: 001
q4: 111
q5: 000
Время выполнения кодирования: 1.30470 с.
Количество логических элементов: 7
Количество триггеров: 3
```

Рисунок 20 – Результаты кодирования (генетический алгоритм)

Результат работы алгоритма в случае, когда был загружен граф автомата в формате smf., продемонстрирован на рисунке 21.

```

Итоговые закодированные состояния:
q0: 010
q1: 000
q2: 100
q3: 011
q4: 101
q5: 001
Время выполнения кодирования: 2.08006 с.
Количество логических элементов: 9

```

Рисунок 21 – Результаты кодирования (генетический алгоритм)

3.2 Проверка работоспособности алгоритмов на примере автомата

Мили

Теперь протестируем конечный автомат, внешний вид которого продемонстрирован на рисунке 22. Как видно из рисунка 22 это автомат Мили.

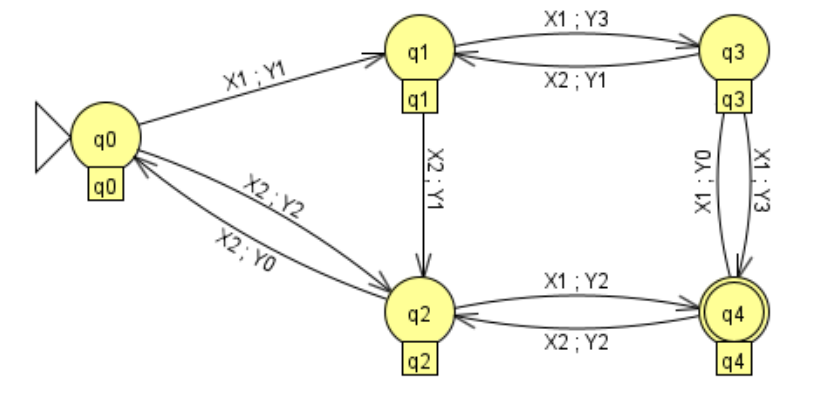


Рисунок 22 – Пример конечного автомата

3.2.1 Унитарный код

Результат работы унитарного кодирования представлен на рисунке 23.

```

Итоговые закодированные состояния:
q0: 00001
q1: 00010
q2: 00100
q3: 01000
q4: 10000
Время выполнения кодирования: 33.19502 мс.
Количество логических элементов: 55
Количество триггеров: 5

```

Рисунок 23 – Результаты кодирования (унитарный код)

3.2.2 Последовательное кодирование

Результат работы кодирования представлен на рисунке 24.

```
Итоговые закодированные состояния:  
q0: 000  
q1: 001  
q2: 010  
q3: 011  
q4: 100  
Время выполнения кодирования: 33.08797 мс.  
Количество логических элементов: 12  
Количество триггеров: 3
```

Рисунок 24 – Результаты кодирования (последовательное кодирование)

3.2.3 Код Джонсона

Результат кодирования представлен на рисунке 25.

```
Итоговые закодированные состояния:  
q0: 000  
q1: 001  
q2: 011  
q3: 111  
q4: 110  
Время выполнения кодирования: 37.64582 мс.  
Количество логических элементов: 18  
Количество триггеров: 3
```

Рисунок 25 – Результаты кодирования (кодирование Джонсона)

3.2.4 Код Грея

Результат кодирования представлен на рисунке 26.

```
Итоговые закодированные состояния:  
q0: 000  
q1: 001  
q2: 011  
q3: 010  
q4: 110  
Время выполнения кодирования: 12.97235 мс.  
Количество логических элементов: 10  
Количество триггеров: 3
```

Рисунок 26 – Результаты кодирования (кодирование Грея)

3.2.5 Метод кодирования с редукцией веса Хэмминга

Результат кодирования представлен на рисунке 27.

```
Итоговые закодированные состояния:  
q0: 100  
q1: 011  
q2: 000  
q3: 010  
q4: 001  
Время выполнения кодирования: 55.46474 мс.  
Количество логических элементов: 17  
Количество триггеров: 3
```

Рисунок 27 – Результаты кодирования (метод с редукцией веса Хэмминга)

3.2.6 Метод кодирования с редукцией расстояния Хэмминга

Результат кодирования представлен на рисунке 28.

```
Итоговые закодированные состояния:  
q0: 000  
q1: 101  
q2: 001  
q3: 111  
q4: 011  
Время выполнения кодирования: 84.83291 мс.  
Количество логических элементов: 14  
Количество триггеров: 3
```

Рисунок 28 – Результаты кодирования (метод с редукцией расстояния Хэмминга)

3.2.7 Генетический алгоритм

Результат кодирования представлен на рисунке 29.

```
Итоговые закодированные состояния:  
q0: 110  
q1: 000  
q2: 100  
q3: 001  
q4: 101  
Время выполнения кодирования: 7.85532 с.  
Количество логических элементов: 8  
Количество триггеров: 3
```

Рисунок 29 – Результаты кодирования (генетический алгоритм)

Результат работы алгоритма в случае, когда был загружен граф автомата в формате v., представлен на рисунке 30.

```
Итоговые закодированные состояния:  
q0: 110  
q1: 001  
q2: 000  
q3: 101  
q4: 100  
Время выполнения кодирования: 9.78006 с.  
Количество логических элементов: 9  
Количество триггеров: 3
```

Рисунок 30 – Результаты кодирования (генетический алгоритм)

Результат работы алгоритма в случае, когда был загружен граф автомата в формате smf., изображён на рисунке 31.

```
Итоговые закодированные состояния:  
q0: 101  
q1: 100  
q2: 001  
q3: 011  
q4: 000
```

Рисунок 31 – Результаты кодирования (генетический алгоритм)

3.3 Чтение файлов

Проверим работоспособность корректного чтения входных файлов на примере формата .jff. Показателем корректного чтения является вывод в консоли достоверной информации об именах состояний, их входных и выходных условий, а также о структуре, показанной в виде графа переходов. В качестве примера для проверки возьмём автомат Мура, использовавшийся в подразделе 3.1. Результат вывода в консоль таков (некоторые части текста будут опущены в силу экономии места):

Тип автомата: Moore

Состояния:

State ID: 0, Name: q0, Output: λ

State ID: 1, Name: q1, Output: $(y1 == 1) \& (y3 == 1) \& (y5 == 1)$

...

State ID: 5, Name: q5, Output: λ

Переходы:

From: q0, To: q2, Condition: $(X1 == 1) \& (X2 == 1)$, Output: $(y1 == 1) \& (y4 == 1)$

From: q2, To: q1, Condition: $X3 == 0$, Output: $(y1 == 1) \& (y3 == 1) \& (y5 == 1)$

...

From: q5, To: q3, Condition: $X4 == 1$, Output: $(y2 == 1) \& (y4 == 1)$

Граф переходов:

q0: [('q2', ' $(X1 == 1) \& (X2 == 1)$ '), ('q1', ' $(X1 == 1) \& (X2 == 0)$ '), ('q0', ' $X1 == 0$ ')]

q1: [('q3', ' $(X2 == 1) \& (X3 == 1) \& (X4 == 0)$ '), ('q2', ' $(X2 == 1) \& (X3 == 1) \& (X4 == 1)$ '), ('q4', ' $X2 == 0$ '), ('q1', ' $(X2 == 1) \& (X3 == 0)$ ')]

q2: [('q1', ' $X3 == 0$ '), ('q3', ' $(X3 == 1) \& (X4 == 0)$ '), ('q2', ' $(X3 == 1) \& (X4 == 1)$ ')]

q3: [('q0', '1')]

q4: [('q0', ' $(X3 == 1) \& (X4 == 0)$ '), ('q5', ' $(X3 == 0) \& (X4 == 0)$ '), ('q3', ' $X4 == 1$ ')]

q5: [('q5', ' $(X3 == 0) \& (X4 == 0)$ '), ('q0', ' $(X3 == 1) \& (X4 == 0)$ '), ('q3', ' $X4 == 1$ ')]

Из этого текста можно сделать выводы о том, что чтение входного файла прошло успешно, и информация об автомате является корректной.

3.4 Сохранение файлов

Теперь перейдём к блоку с проверкой работоспособности корректного сохранения файлов с полученными результатами. Для начала проверим случаи, когда формат загруженного и сохраняемого файлов совпадают. Проверка сводится к тому, чтобы убедиться в наличии соответствующей строки кода в полученном файле, информирующей о закодированных состояниях. Для начала проверим формат .jff. Пусть конечный автомат имеет внешний вид, как показано на рисунке 32.

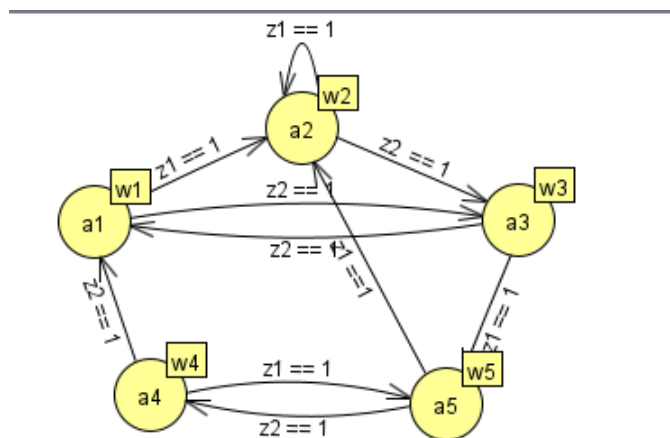


Рисунок 32 – Пример конечного автомата

Пусть после загрузки соответствующего этому автомату файла формата .jff в качестве алгоритма кодирования будет выбран метод с редукцией расстояния Хэмминга, результаты которого показаны на рисунке 33.

```

Итоговые закодированные состояния:
a1: 000
a2: 101
a3: 001
a4: 110
a5: 111
  
```

Рисунок 33 – Результат кодирования

Нажимаем соответствующую кнопку для сохранения файла и, например, задаём ему новое имя, как это показано на рисунке 34.

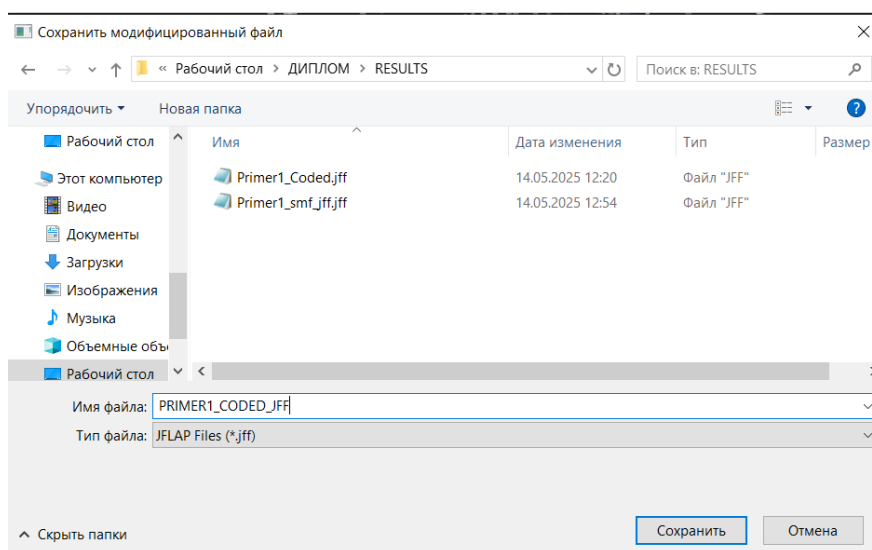


Рисунок 34 – Сохранение модифицированного файла под новым именем

В случае удачного сохранения программа информирует соответствующим окном поверх приложения об этом пользователя. Это показано на рисунке 35.

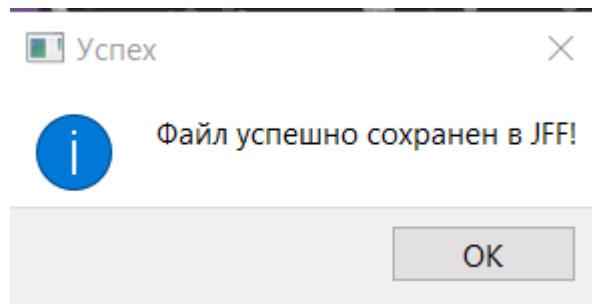


Рисунок 35 – Сообщение об удачном сохранении

Проверим, присутствует ли информация о закодированных состояниях в этом файле. Для начала проверим сам код. Это делалось с помощью открытия файла через блокнот или любой другой текстовый редактор. Заодно также откроем исходный файл, чтобы сравнить их. Листинги исходного и модифицированного файлов представлены ниже.

Часть листинга исходного файла Primer1.jff

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><!--Created with  
JFLAP 6.4.--><structure>#13;  
  <type>moore</type>#13;  
  <automaton>#13;  
    <!--The list of states.-->#13;  
    <state id="0" name="a1">#13;  
      <x>92.0</x>#13;  
      <y>106.0</y>#13;  
      <output>w1</output>#13;  
    </state>#13;  
    <state id="1" name="a2">#13;  
      <x>207.0</x>#13;  
      <y>54.0</y>#13;  
      <output>w2</output>#13;  
    </state>#13;
```



```

<state id="2" name="a3">#13;
    <x>327.0</x>#13;
    <y>105.0</y>#13;
    <output>w3</output>#13;

```

Часть листинга модифицированного файла *PRIMER1_CODED_JFF.jff*

```

<?xml version='1.0' encoding='utf-8'?>
<structure>

```

```

    <type>moore</type>
    <automaton>
        <state id="0" name="a1">
            <x>92.0</x>
            <y>106.0</y>
            <output>w1</output>
        <label>000</label></state>
        <state id="1" name="a2">
            <x>207.0</x>
            <y>54.0</y>
            <output>w2</output>
        <label>101</label></state>
        <state id="2" name="a3">
            <x>327.0</x>
            <y>105.0</y>
            <output>w3</output>
        <label>001</label></state>

```

Заметим отличие: у полученного файла есть дополнительный тег вида `<label></label>`, внутри которого находится информация о коде состояния. Более наглядна и ясна разница между этими файлами, если их оба открыть в программе JFLAP. На рисунке 36 показан модифицированный файл после кодирования.

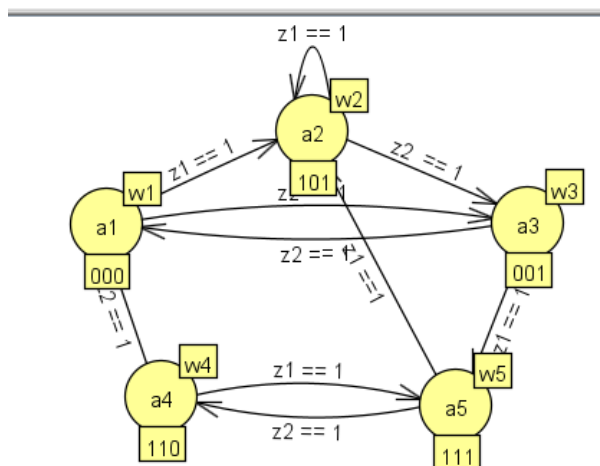


Рисунок 36 – Результат после сохранения

Теперь проверим аналогично форматы .v и .smf. Начнём с первого. Отметим, что проверка и сравнение двух файлов будет проводится в этом случае в Quartus II версии 9.1. После загрузки файла, что показано на рисунке 37, и произвольное выбора в качестве алгоритма метода с редукцией веса Хэмминга, как на рисунке 38.

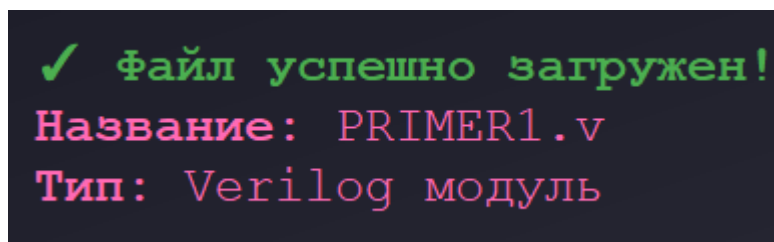


Рисунок 37 – Удачная загрузка файла

```
ИТОГОВЫЕ закодированные состояния:
a1: 011
a2: 000
a3: 010
a4: 100
a5: 001
```

Рисунок 38 – Результат работы алгоритма

На рисунке 39 и 40 продемонстрирована часть кода исходного файла и полученного после сохранения соответственно.

```
PRIMER1.v
1 module PRIMER1 (
2     reset, clock, z1, z2,
3     w1, w2, w3, w4, w5);
4
5     input reset;
6     input clock;
7     input z1;
8     input z2;
9     output reg w1, w2, w3, w4, w5;
10    tri0 reset;
11    tri0 z1;
12    tri0 z2;
13    reg [4:0] fstate;
14    reg [4:0] reg_fstate;
15    parameter a1=0, a2=1, a3=2, a4=3, a5=4;
16
17    always @(posedge clock)
18    begin
19        if (clock) begin
20            fstate <= reg_fstate;
21        end
22    end
23 end
```

Рисунок 39 – Часть кода исходного файла

```
../RESULTS/Primer1_verilog_coded.v
module PRIMER1 (
    reset, clock, z1, z2,
    w1, w2, w3, w4, w5);

    input reset;
    input clock;
    input z1;
    input z2;
    output reg w1, w2, w3, w4, w5;
    tri0 reset;
    tri0 z1;
    tri0 z2;
    reg [4:0] fstate;
    reg [4:0] reg_fstate;
    parameter a2 = 0, a5 = 1, a3 = 2, a1 = 3, a4 = 4;

    always @(posedge clock)
    begin
        if (clock) begin
            fstate <= reg_fstate;
        end
    end
end
```

Рисунок 40 – Часть кода полученного файла

В данном случае, строкой кода, в которой содержится информация о кодах состояний, является строка **parameter**. При сравнении этих двух рисунков 39 и 40 мы увидим разницу и убедимся, что значения кодов в

полученном файле совпадают со значениями, что были показаны в окне приложения на рисунке 38.

Проверим формат .smf. Результат загрузки файла показан на рисунке 41. Рисунок 42 демонстрирует результат работы программы при произвольном выборе в качестве алгоритма кодирования Джонсона. На рисунке 43 показан граф автомата в исходном файле, а на рисунке 44 – полученном. Отметим, что просмотр файлов этого формата так же осуществлялся в Quartus II версии 9.1.

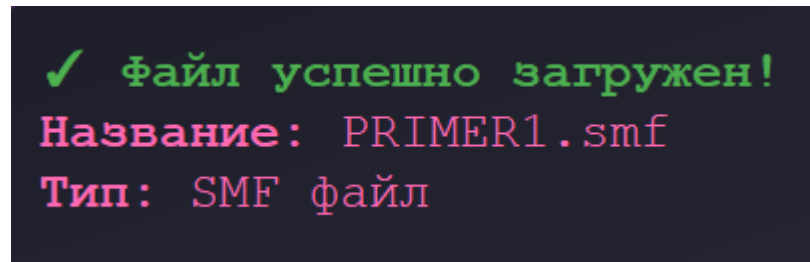


Рисунок 41 – Удачная загрузка файла

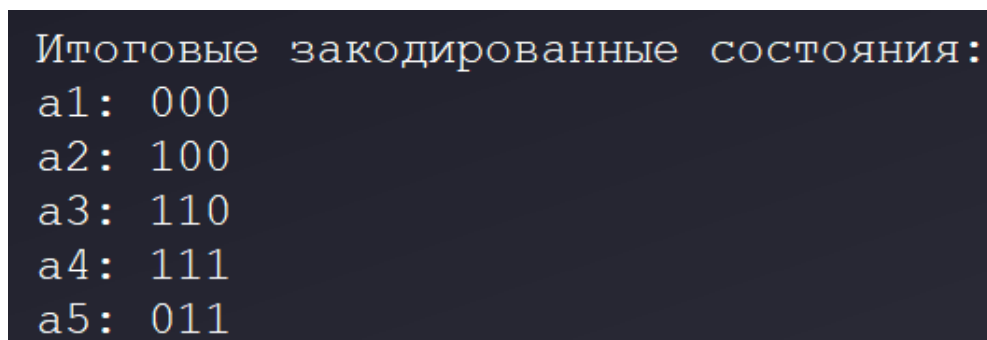


Рисунок 42 – Результат работы алгоритма

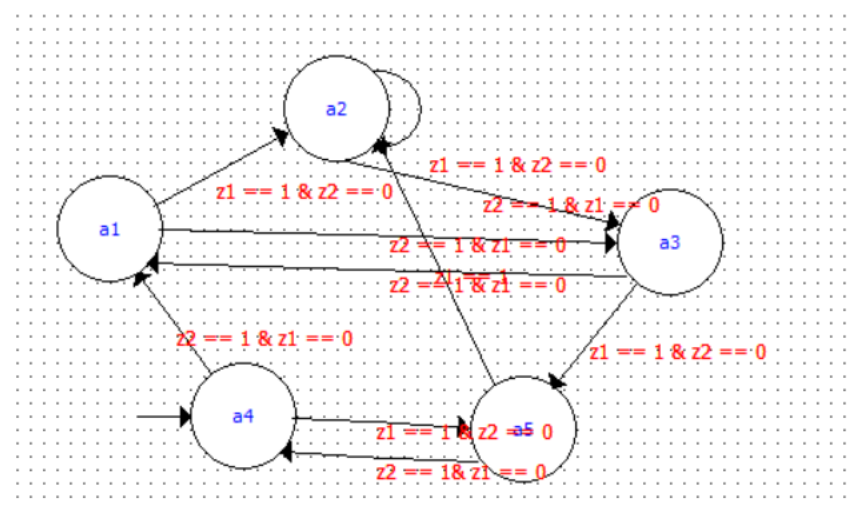


Рисунок 43 – Исходный внешний вид автомата

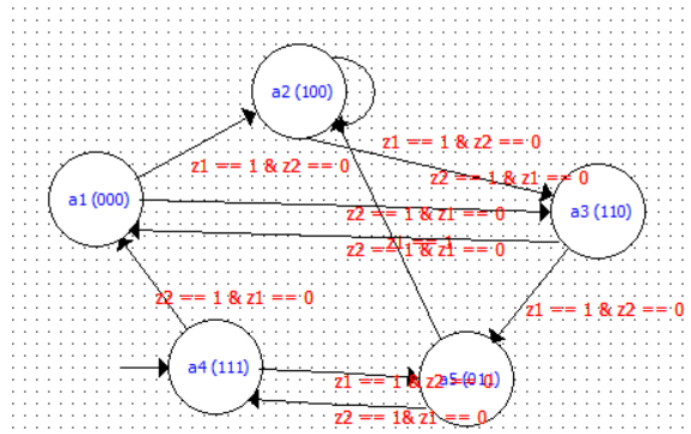


Рисунок 44 – Полученный внешний вид автомата

Делаем выводы, что программа корректно модифицирует и сохраняет файлы в случае совпадения исходного загруженного с желаемым сохранить.

Завершающим этапом является проверка и демонстрация функции сохранения в случае разных форматов относительно загруженного и сохраняемого. В качестве проверяемого файла снова воспользуемся автоматом формата .jff, показанного на рисунке 32 и попробуем его сохранить в формат .v. Опустим демонстрацию шагов при загрузке файла, продемонстрируем просто полученный результат кодирования, который изображён на рисунке 45 и сразу перейдем к просмотру сгенерированного под новым форматом файла. Это показано на рисунке 46. А на рисунке 47 аналогично показана часть кода, демонстрирующая сохранение структуры конечного автомата, что является подтверждением грамотной генерации.

Итоговые закодированные состояния:

```
a1: 000
a2: 100
a3: 110
a4: 111
a5: 011
```

Рисунок 45 – Результат работы алгоритма

```

1  module verilog_file (
2      input clk,
3      input reset,
4      input z1,
5      input z2
6  );
7
8
9      // State encoding
10     parameter a1=0,
11                a2=4,
12                a3=6,
13                a4=7,
14                a5=3;
15
16     reg [2:0] fstate;
17     reg [2:0] reg_fstate;
18
19     always @(posedge clk)
20     begin
21         if (clk) begin

```

Рисунок 46 – Часть кода сгенерированного файла в формате .v

```

always @(fstate or reset or z1 or z2)
begin
    if (reset) begin
        reg_fstate <= a1;
    end
    else begin
        case (fstate)
            a1: begin
                if (z1 == 1'b1 & z2 == 1'b0)
                    reg_fstate <= a2;
                else if (z2 == 1'b1 & z1 == 1'b0)
                    reg_fstate <= a3;
                else
                    reg_fstate <= a1;
            end
            a2: begin
                if (z1 == 1'b1 & z2 == 1'b0)
                    reg_fstate <= a2;
                else if (z2 == 1'b1 & z1 == 1'b0)
                    reg_fstate <= a3;

```

Рисунок 47 – Демонстрация грамотной генерации с сохранением структуры автомата

На основании рисунков 46 и 47 можно сделать вывод о том, что формирование файла под новым форматом с сохранением информации о кодировках состояний работает корректно.

Теперь, например, проверим случай, когда исходный файл загружается в формате .smf и сохраняется под новым форматом .jff. На рисунке 48 показан

результат кодирования состояний автомата, а на рисунке 49 показан полученный конечный автомат с закодированными состояниями.

```
Итоговые закодированные состояния:  
a1: 010  
a2: 000  
a3: 100  
a4: 110  
a5: 001
```

Рисунок 48 – Результат работы алгоритма

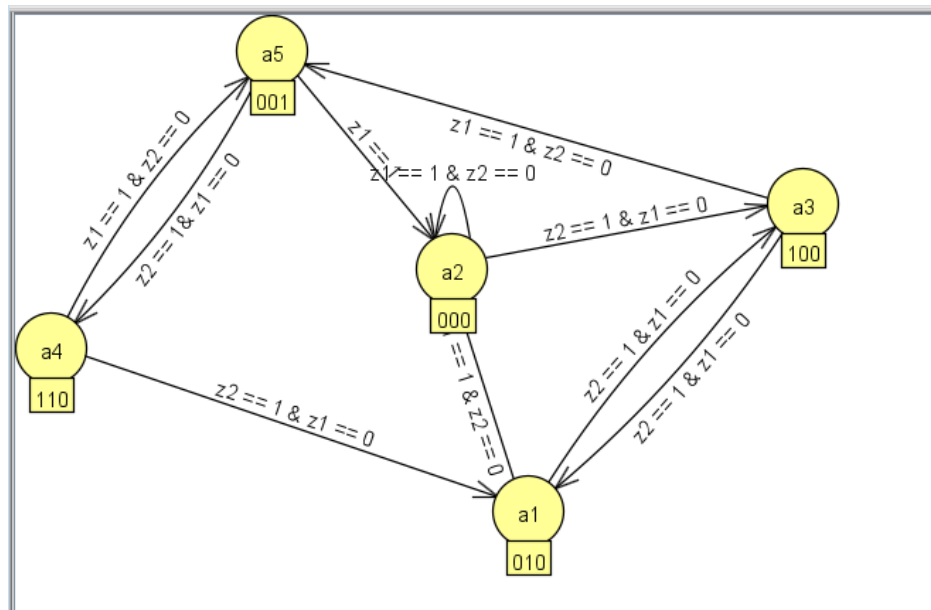


Рисунок 49 – Полученный конечный автомат в новом формате

Завершающим этапом проверки функции сохранения является корректная конвертации в формат .smf из форматов .jff и .v. Начнём с первого случая. После загрузки файла и произвольного выбора кодирования получился результат, показанный на рисунке 50.

```
Итоговые закодированные состояния:  
a1: 011  
a2: 000  
a3: 010  
a4: 100  
a5: 001
```

Рисунок 50 – Результат кодирования

После нажатия на соответствующую кнопку и задания имени новому файлу, теперь его можно открыть и посмотреть содержимое. Результат продемонстрирован на рисунке 51.

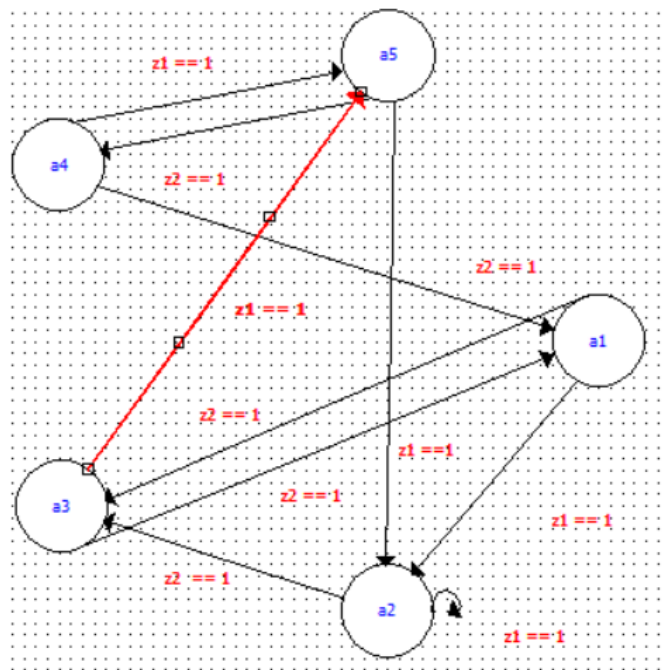


Рисунок 51 – Результат конвертации из .jff в .smf

Сравнивая полученный результат на рисунке 51 и исходный граф в формате .jff из рисунка 22 можно сделать вывод о том, что структурно автомат построен в сгенерированном файле корректно.

Проверим теперь результат конвертации из .v в .smf. В результате произвольного выбора кодирования получаем такие результаты, как показано на рисунке 53.

```

Итоговые закодированные состояния:
a1: 011
a2: 000
a3: 010
a4: 100
a5: 001
  
```

Рисунок 53 – Результаты кодирования

После сохранения нового файла наблюдаем результат, продемонстрированный на рисунке 54.

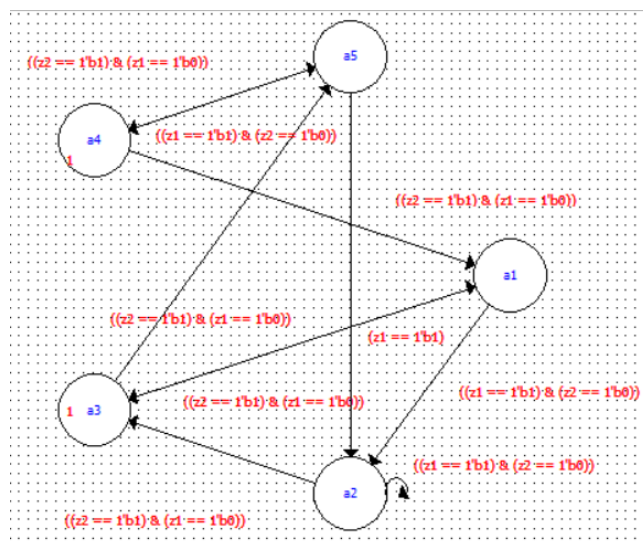


Рисунок 54 – Результат конвертации из .v в .smf

Аналогично сравнивая результаты на рисунках 54 и 22 так же приходим к выводу, что конвертация является успешной и корректной.

3.5 Обработка граничных и ошибочных ситуаций

Рассмотрим теперь возможные граничные или ошибочные ситуации, которые могут возникнуть при работе программного средства. Например, после запуска приложения пользователь нажимает на кнопку «Загрузить файл», но не загружает его. Тогда программа реагирует соответственно, как показано на рисунке 55.

Файл не выбран!

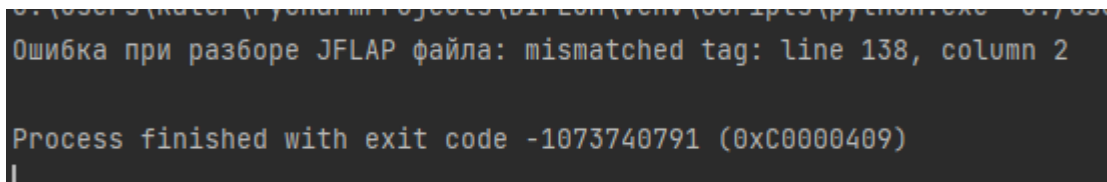
Рисунок 55 – Файл загружен не был

Если пользователь загрузит файл поддерживаемого файла, но в котором отсутствует описание конечного автомата, то программа укажет на это, как показано на рисунке 56.

✓ файл успешно загружен!
 Название: ВЕРИЛОГ_ПУСТОЙ.v
 Тип: Verilog модуль
 ⚠ Загружен пустой файл или автомат ⚠

Рисунок 56 – Загрузка пустого файла

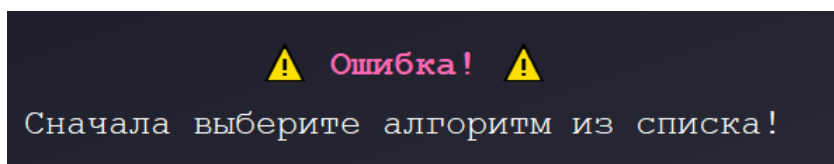
Если пользователь попытается загрузить файл поддерживаемого формата, но с синтаксическими ошибками, то приложение закроется, а в консоли появится ошибка. Например, в данном случае был загружен файл без тега <automation>. Результат работы программы показан на рисунке 57.



```
Ошибка при разборе JFLAP файла: mismatched tag: line 138, column 2
Process finished with exit code -1073740791 (0xC0000409)
```

Рисунок 57 – Загрузка повреждённого файла

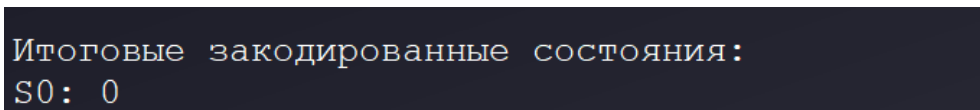
Если пользователь загрузил файл, но забыл выбрать алгоритм кодирования и сразу нажал на кнопку «Закодировать», программа напомнит ему о пропущенном действии, как показано на рисунке 58.



⚠ Ошибка! ⚠
Сначала выберите алгоритм из списка!

Рисунок 58 – Отсутствие выбранного алгоритма

Также, если пользователь загрузит файл с автоматом, не имеющим переходов, то получит, например, такой результат, как на рисунке 59.



```
Итоговые закодированные состояния:
S0: 0
```

Рисунок 59 – Загрузка файла без переходов

3.6 Анализ программного тестирования

Проанализируем результаты ручного и программного кодирования автомата, изображённого на рисунке 2. Отметим, что результаты ручного и программного кодирования унитарного, последовательного кодирования, а также кодирования Джонсона и Грея совпадают.

3.6.1 Метод с редукцией веса Хэмминга

Рассмотрим теперь результаты кодирования методом с редукцией веса Хэмминга. В данном случае отметим, что результаты на рисунке 14 получились отличным от результатов ручного кодирования. Например, в ручном кодировании получились такие коды у некоторых состояний:

$$q1 = 011, q5 = 110$$

А в программной реализации, результаты которой показаны на рисунке 14, такие:

$$q1 = 101, q5 = 011$$

Выясним, почему так получилось. Начнём с первого состояния $q5$, так как оно кодировалось раньше $q1$. На этапе его кодирования можно было выбрать несколько вариантов, так как они были равноценны: 110 и 011. В результате ручного кодирования выбор пал на первый вариант. Программа выбирает случайным образом код в таких случаях, и она решила выбрать второй вариант. При этом в консоль она вывела такие результаты на этом этапе:

Выбрали вершину $q0$

$\{q3: '000', q2: '001'\}$

Выбран код: 010 (used: {0, 1})

Закодировали обе вершины: $q0 \rightarrow 010, q5 \rightarrow 011$

Аналогично на этапе кодирования $q1$ можно было сделать выбор между вариантами 101 и 011. При ручном кодировании был выбран второй вариант. В программной же реализации был выбран первый вариант, и это отразилось в консольном выводе:

Выбрали вершину $q1$

Закодировали: $q1 \rightarrow 101$

3.6.2 Метод с редукцией расстояния Хэмминга

Рассмотрим и сравним результаты кодирования методом с редукцией расстояния Хэмминга. Так, в ручном кодировании перечислим некоторые состояния и их коды:

$$q0 = 100, q3 = 101$$

А в программной реализации, результаты которой показаны на рисунке 15, такие:

$$q0 = 011, q3 = 010$$

Напомним, что сначала кодировалось состояние $q0$. У него было целых 4 кандидата: 101, 010, 100, 011. все они имели одинаковое значение

получившейся суммы, рассчитанной по формуле (7). Здесь аналогичная ситуация с случайным выбором равноценных вариантов, как и в предыдущем алгоритме. При ручном кодировании был выбран код 100, тогда как при программном – 011. При этом программа в консоли отражала этап кодирования q_0 следующим образом:

Доступные соседи для состояния q_0 : ['010', '101', '100', '011']

Допустимые соседи после исключения: ['010', '101', '100', '011']

Для состояния q_1 : $w(q_1, q_0) = 1$, $d(000, 010) = 1$, вклад = 1

Для состояния q_2 : $w(q_2, q_0) = 1$, $d(001, 010) = 2$, вклад = 2

Сумма $S(\alpha)$ для кандидата 010: 3

Для состояния q_1 : $w(q_1, q_0) = 1$, $d(000, 101) = 2$, вклад = 2

Для состояния q_2 : $w(q_2, q_0) = 1$, $d(001, 101) = 1$, вклад = 1

Сумма $S(\alpha)$ для кандидата 101: 3

Для состояния q_1 : $w(q_1, q_0) = 1$, $d(000, 100) = 1$, вклад = 1

Для состояния q_2 : $w(q_2, q_0) = 1$, $d(001, 100) = 2$, вклад = 2

Сумма $S(\alpha)$ для кандидата 100: 3

Для состояния q_1 : $w(q_1, q_0) = 1$, $d(000, 011) = 2$, вклад = 2

Для состояния q_2 : $w(q_2, q_0) = 1$, $d(001, 011) = 1$, вклад = 1

Сумма $S(\alpha)$ для кандидата 011: 3

Случайно выбран код из кандидатов с минимальной суммой: 011

Закодировано состояние $q_0 \rightarrow 011$

По тем же причинам получились различные результаты кода для состояния q_3 при ручном и программном кодировании, а также отличность результатов от ручного кодирования на рисунке 17.

3.6.3 Генетический алгоритм

Все результаты генетического алгоритма при ручной и программной реализации, где программная показана на рисунках 19 – 21. Это объясняется тем, что при ручном кодировании рассматривалась лишь одна итерация, что существенно мало для генетического алгоритма. Отсюда можно сделать вывод о том, что при ручном кодировании получился результат, имеющий большую

оценку, чем любой из программных результатов. Помимо этого, результат отличается ещё в силу того, что генетический алгоритм полон вероятностных событий, а именно: возникновение мутации, количество исходных особей, место точки кроссинговера.

Итого, в зависимости от того, какой вариант кода выберет программа случайным образом в методах с редукцией веса или расстояния Хэмминга, а также каким образом и с какой вероятностью программа решит выполнять какой-либо из этапов генетического алгоритма, коды в кодируемых состояниях могут отличаться от ручного кодирования. Этим и объясняются отличность результаты программного и ручного кодирования.

3.7 Сравнительный анализ методов кодирования состояний конечных автоматов.

Для проведения обоснованного анализа необходимо определить критерии, по которым будет оцениваться эффективность алгоритмов кодирования состояний конечных автоматов с целью выбора наилучшего варианта для конкретных условий проектирования. Эти критерии целесообразно разделить на две группы: аппаратные и программные.

Аппаратные критерии отражают влияние выбранного метода физическую реализацию автомата. К ним относится *сложность структурной реализации* – она определяет количество и тип элементов, необходимых для реализации конечного автомата, включая триггеры и логические элементы.

Программные критерии оценивают и эффективность работы алгоритмов кодирования. К таковым критериям можно отнести:

1. *Объём оперативной памяти* – определяет общий объём оперативной памяти, необходимый для выполнения алгоритма.
2. *Время выполнения алгоритма* – продолжительность (в секундах, минутах или иных единицах времени), которая требуется для кодирования всех состояний конечного автомата.

Для вычисления значений ранее введённых критериев использовались автоматы из рисунка 2 и рисунка 22. Результаты первого автомата представлены в приложении А, а второго – в приложении Б. Значения являются средним временным показателем в рамках проведения 10 попыток кодирования каждым методом. Отметим, что красным выделены ячейки, которые демонстрируют худший результат среди всех методов кодирования по текущему критерию, зелёным – ячейки с лучшим результатом.

Можно сделать выводы о том, что в условиях тестирования на автомате Мура меньше всего памяти занимает программная реализация кодирования Джонсона, а больше всего – генетический алгоритм. С точки зрения времени выполнения лучшие результаты показало кодирование Грея, а худшие – генетический алгоритм. По критерию сложности структурной реализации можно сказать, что меньше всего триггеров для тестируемого случая потребуется при выборе генетического алгоритма, а больше всего элементов понадобится при выборе унитарного кода.

В случае с автоматом Мили были получены следующие результаты. Больше всего объема оперативной памяти занимает кодирование Грея, а меньше всего – кодирование Джонсона. Дольше выполняется генетический алгоритм, тогда как быстрее – метод с редукцией веса Хэмминга. Больше всего потребует логических элементов унитарный код, а меньше – генетический алгоритм.

Алгоритм с редукцией веса Хэмминга может быть выбран за высокую эффективность для D-триггеров. Алгоритм с редукцией расстояния Хэмминга – за аналогичную эффективность для JK- и T-триггеров и целенаправленное снижение энергопотребления путём уменьшения числа переключений.

Генетический алгоритм выделяется своей гибкостью, способностью учитывать несколько критериев одновременно и адаптироваться к различным условиям, что делает его подходящим для работы со сложными автоматами.

Выбор использования кодирований Джонсона и Грея обусловлен тем, что первый оптимален для циклических автоматов и имеет простую

реализацию на сдвиговом регистре с инверсией, что вдвое эффективнее унитарного кода.

Второй отличается высокой помехоустойчивостью и эффективной работой с автоматами любого размера, сохраняя универсальность для разных типов триггеров.

Унитарный код имеет высокую эффективность на автоматах с малым или средним количеством состояний, а также прост в реализации. Последней причиной так же обуславливается выбор последовательного кодирования.

ЗАКЛЮЧЕНИЕ

В ходе выполнения данной работы было создано программное средство, реализующее следующие методы кодирования состояний конечных автоматов:

1. Унитарный код
2. Последовательное кодирование
3. Код Джонсона
4. Код Грея
5. Алгоритм с редукцией веса Хэмминга
6. Алгоритм с редукцией расстояния Хэмминга
7. Генетический алгоритм для минимизации сложности логической схемы при синтезе на тригтерах типа D

Программное средство поддерживает следующие форматы описания конечных автоматов: .v, .smf, .jff.

На основании критериев также был проведён сравнительный анализ всех рассмотренных в работе алгоритмов кодирования с целью выявления у каждого достоинств и недостатков.

Результаты тестирования разработанного программного средства подтвердили работоспособность программного средства.

Практическая значимость работы заключается в создании такого инструмента, который автоматизирует процесс кодирования состояний конечного автомата и позволяет находить оптимальные варианты кодирования для синтеза автоматов в виде интегральных схем.

Разработанный продукт подлежит дальнейшему развитию, которое может заключаться в расширении поддерживаемых алгоритмов кодирования и форматов входных и выходных файлов, оптимизации производительности для работы с автоматами с большим количеством состояний.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Баранов С.И. Синтез микропрограммных автоматов (граф-схемы и автоматы). – 2-е изд., перераб. и доп. – Ленинград : Энергия, Ленинградское отделение, 1979. – 232 с.
2. Строгонов А.В. Проектирование конечных автоматов по методу ONE NOT / А.В. Строгонов // Компоненты и технологии. — 2007. — № 10. — С. 124–128.
3. Поттосин Ю.В. Кодирование состояний дискретного автомата, ориентированное на уменьшение энергопотребления реализующей схемы / Ю.В. Поттосин // Прикладная дискретная математика. — 2011. — № 4(14). — С. 62–71.
4. Угрюмов Е. П. Цифровая схемотехника: учеб. Пособие для вузов. — 2-е изд., перераб. И доп. — СПб.: БХВ-Петербург, 2007. — 800 с.
5. Salauyou V., Bulatow W. Optimized Sequential State Encoding Methods for Finite-State Machines in Field-Programmable Gate Array Implementations // Applied Sciences. – 2024. – 19 p. – Vol. 14, no. 13. – Article no. 5594. – DOI; URL: <https://www.mdpi.com/2076-3417/14/13/5594> (дата обращения: 06.06.2025).
6. Буркатовская Ю.Б., Веремеенко Е.С. Теория автоматов: учебно-методическое пособие. – Томск: Изд-во Томского политехнического университета, 2010. – 108 с.
7. Goldberg D.E. Genetic algorithms in search, optimization, and machine learning. — Addison-Wesley, 1989. — 432 p. — ISBN 0-201-15767-5.
8. Берёза А.Н., Ляшов М.В. Эволюционный синтез конечных автоматов // Известия ЮФУ. Технические науки. – 2010. – Тематический выпуск. – С. 210–216.
9. Куликова И. В. Построение генетического алгоритма для решения задач оптимизации с различными ограничениями для параметров // Современные наукоемкие технологии. 2020. №2. С. 40-44; URL: <https://top-technologies.ru/ru/article/view?id=37912> (дата обращения: 19.05.2025).

10. ГОСТ 19.201-78. Единая система программной документации. Техническое задание. Требования к содержанию и оформлению [Текст] : межгосударственный стандарт : дата введения 1980-01-01 / Гос. ком. СССР по стандарта. – Введ. 1980-01-01. – М. : Стандартинформ, 2010. – 4 с.
11. Quine McCluskey Method [Электронный ресурс] // GeekforGeeks. – URL: <https://www.geeksforgeeks.org/quine-mccluskey-method/> (дата обращения: 07.06.2025).

ПРИЛОЖЕНИЕ А. Оценка результатов кодирования состояний автомата Мура

Критерий	Код Грея	Код Джонсона	Метод с редукцией веса Хэмминга	Метод с редукцией расстояния Хэмминга	Генетический алгоритм	Унитарный код	Последовательное кодирование
Объём оперативной памяти (Кб)	55.90	0.86	58.70	20.35	63.69	27.33	43.77
Время выполнения алгоритма (мс)	37.79479	44.24185	59.75792	73.81034	3.86996 с.*	55.29689	51.89452
Сложность структурной реализации (лог. элементов)	67	58	67	59	53	149	70
Число триггеров	6	3	3	3	3	3	3

* — данное значение было получено в случае, когда использовалось значение итераций по умолчанию (25).
Значение указано в секундах.

ПРИЛОЖЕНИЕ Б. Оценка результатов кодирования состояний автомата

Мили

Критерий	Код Грея	Код Джонсона	Метод с редукцией веса Хэмминга	Метод с редукцией расстояния Хэмминга	Генетический алгоритм	Унитарный код	Последовательное кодирование
Объём оперативной памяти (Кб)	93.08	0.51	7.13	12.04	71.54	41.57	68.55
Время выполнения алгоритма (мс)	50.83280	59.18963	33.36541	45.58494	3.00818 с.**	39.71565	36.69622
Сложность структурной реализации (лог. элементов)	11	18	17	14	7	55	12
Число триггеров	5	3	3	3	3	3	3

** — данное значение было получено в случае, когда использовалось значение итераций по умолчанию (30).
Значение указано в секундах.

ПРИЛОЖЕНИЕ В Листинг кода программы

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QFileDialog, QComboBox, QLabel
import xml.etree.ElementTree as ET
import re
import time
from tabulate import tabulate
from collections import defaultdict
from itertools import product
import random
from PyQt5.QtCore import Qt, QTimer
import math
import tracemalloc
from quine_mccluskey.qm import QuineMcCluskey
# ----- ГРАФИЧЕСКИЙ ИНТЕРФЕЙС -----
class GalaxyBackground(QtWidgets.QWidget):
    def __init__(self, parent=None):
        super().__init__(parent)
        self.setGeometry(0, 0, 1112, 965)
        self.stars = [
            {
                "angle": random.uniform(0, 2 * math.pi), # Угол
                "radius": random.randint(200, 550), # Радиус
                "size": random.randint(5, 10), # Размер
                "brightness": random.randint(150, 255), # Яркость
                "trail": [], # След звёздочки
            }
            for _ in range(200)
        ]

        self.timer = QtCore.QTimer(self)
        self.timer.timeout.connect(self.update_stars)
        self.timer.start(50) # Обновление каждые 50 мс

    def update_stars(self):
        # Обновляем позиции звездочек и их следы
        for star in self.stars:
            star["angle"] += random.uniform(0.01, 0.05) # Поворачиваем звезду
            if star["angle"] > 2 * math.pi:
                star["angle"] -= 2 * math.pi # Сбрасываем угол после полного круга

            # Добавляем текущую позицию в след
            center_x, center_y = self.width() // 2, self.height() // 2
            x = center_x + star["radius"] * math.cos(star["angle"])
            y = center_y + star["radius"] * math.sin(star["angle"])
            star["trail"].append((x, y))

            # Ограничиваем длину следа
            if len(star["trail"]) > 20: # Максимальная длина следа
                star["trail"].pop(0)
        self.update() # Перерисовываем фон

    def paintEvent(self, event):
        painter = QtGui.QPainter(self)
        painter.setRenderHint(QtGui.QPainter.Antialiasing)
        painter.fillRect(self.rect(), QtGui.QColor(30, 30, 50)) # Фон космоса
```

```

# Центр экрана
center_x, center_y = self.width() // 2, self.height() // 2

# Рисуем следы звездочек
for star in self.stars:
    for i, (x, y) in enumerate(star["trail"]):
        # Интерполяция цвета от белого к розовому
        white = QtGui.QColor(255, 255, 255)
        pink = QtGui.QColor(255, 105, 180)
        progress = i / len(star["trail"]) # Прогресс от 0 (начало) до 1 (конец)
        color = QtGui.QColor(
            int(white.red() + (pink.red() - white.red()) * progress),
            int(white.green() + (pink.green() - white.green()) * progress),
            int(white.blue() + (pink.blue() - white.blue()) * progress),
        )
        alpha = int(star["brightness"] * (1 - progress)) # Уменьшаем прозрачность
        color.setAlpha(alpha)
        painter.setBrush(color)
        painter.setPen(Qt.Core.Qt.NoPen)
        painter.drawEllipse(Qt.Core.QPointF(x, y), star["size"] / 2, star["size"] / 2)

# Рисуем звездочки
for star in self.stars:
    x = center_x + star["radius"] * math.cos(star["angle"])
    y = center_y + star["radius"] * math.sin(star["angle"])
    painter.setBrush(QtGui.QColor(255, 255, 255, star["brightness"]))
    painter.setPen(Qt.Core.Qt.NoPen)
    painter.drawEllipse(Qt.Core.QPointF(x, y), star["size"], star["size"])

class MainWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("Galaxy Background with Gradient Trails")
        self.setGeometry(100, 100, 1112, 965)
        self.central_widget = GalaxyBackground(self)
        self.setCentralWidget(self.central_widget)

class Ui_Coding(object):
    #CSS-стили
    class Styles:
        button_main = """
        QPushButton {
            background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #6E3A8E, stop:1 #8E5AAE);
            color: white;
            border-radius: 15px;
            padding: 20px;
            font-size: 28px;
            font-weight: bold;
            border: 2px solid #4A2A5E;
            font-family: Arial;
            box-shadow: 10px 10px 0px #FF69B4;
            transition: all 0.3s ease;
        }
        QPushButton:hover {
            background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #8E5AAE, stop:1 #6E3A8E);
            box-shadow: 0px 0px 20px 5px #FF69B4;
        }

```

```

        border: 2px solid #FF69B4;
    }
    QPushButton:pressed {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #4A2A5E, stop:1 #6E3A8E);
        box-shadow: 5px 5px 0px #FF69B4;
    }
    """
button_save_jff = """
    QPushButton {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #6E3A8E, stop:1 #8E5AAE);
        color: white;
        border-radius: 10px;
        padding: 8px;
        font-size: 18px;
        font-weight: bold;
        border: 2px solid #4A2A5E;
        font-family: Arial;
    }
    QPushButton:hover {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #8E5AAE, stop:1 #6E3A8E);
        border: 2px solid #FF69B4;
    }
    """
line_edit = """
    QLineEdit {
        background: #3E3E4A;
        color: white;
        border: 2px solid #4A2A5E;
        border-radius: 10px;
        padding: 10px;
        font-size: 18px;
    }
    """
combo_box = """
    QComboBox {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #6E3A8E, stop:1 #8E5AAE);
        color: white;
        border-radius: 15px;
        padding: 10px;
        font-size: 28px;
        font-weight: bold;
        border: 2px solid #4A2A5E;
        font-family: Arial;
        box-shadow: 10px 10px 0px #FF69B4;
    }
    QComboBox QAbstractItemView {
        color: white;
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #6E3A8E, stop:1 #8E5AAE);
        border: 2px solid #4A2A5E;
        border-radius: 15px;
        selection-background-color: #FF69B4;
        selection-color: white;
    }
    QComboBox:hover {
        background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #8E5AAE, stop:1 #6E3A8E);
        box-shadow: 0px 0px 20px 5px #FF69B4;
        border: 2px solid #FF69B4;
    }
    """

```

```

    }
    QComboBox::drop-down {
        subcontrol-origin: padding;
        subcontrol-position: top right;
        width: 30px;
        border-left-width: 1px;
        border-left-color: #4A2A5E;
        border-left-style: solid;
        border-top-right-radius: 15px;
        border-bottom-right-radius: 15px;
    }
    QComboBox::down-arrow {
        image: url(/icons/down_arrow.png);
    }
}
"""
text_edit = """
QTextEdit {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #1E1E2A, stop:1 #2E2E3A);
    color: white;
    border-radius: 15px;
    padding: 15px;
    font-family: Courier;
    font-size: 24px;
    border: 2px solid #4A2A5E;
    box-shadow: 5px 5px 10px rgba(0, 0, 0, 0.3);
}
.warning-header {
    color: #FF69B4;
    font-size: 24px;
    font-weight: bold;
}
.warning-text {
    color: #FFFFFF;
    font-size: 24px;
}
.trigger-name {
    color: #00FFFF;
    font-weight: bold;
    text-decoration: underline;
}
}
"""
save_jff_style = """
QPushButton {

    background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #6E3A8E, stop:1 #8E5AAE);
    color: white;
    border-radius: 10px;
    padding: 8px;
    font-size: 18px;
    font-weight: bold;
    border: 2px solid #4A2A5E;
    font-family: Arial;
}
QPushButton:hover {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #8E5AAE, stop:1 #6E3A8E);
    border: 2px solid #FF69B4;
}
}
"""

```



```

"""

def setupUi(self, Coding):
    Coding.setObjectName("Coding")
    Coding.setEnabled(True)
    Coding.resize(1112, 965)
    Coding.setStyleSheet("background-color: #2E2E3A;") # Фиолетовый фон

    self.centralwidget = GalaxyBackground(Coding)
    self.centralwidget.setObjectName("centralwidget")

    # Кнопка загрузки файла
    self.pushButton = QtWidgets.QPushButton(self.centralwidget)
    self.pushButton.setGeometry(QtCore.QRect(350, 120, 381, 111))
    self.pushButton.setStyleSheet(self.__class__.Styles.button_main)
    self.pushButton.setObjectName("pushButton")
    self.pushButton.setText("Загрузить файл")
    self.pushButton.clicked.connect(self.load_file)

    # Кнопка ввода количества итераций (генетический алгоритм)
    self.iteration_input = QtWidgets.QLineEdit(self.centralwidget)
    self.iteration_input.setGeometry(QtCore.QRect(270, 340, 550, 50))
    self.iteration_input.setStyleSheet(self.__class__.Styles.line_edit)
    self.iteration_input.setPlaceholderText("Число итераций (по умолч.: 100), макс.: 10000")
    self.iteration_input.setVisible(False)
    self.iteration_input.setValidator(QtGui.QIntValidator(10, 10000)) # Ограничения

    # Выпадающий список для выбора алгоритма
    self.comboBox_algorithm = QComboBox(self.centralwidget)
    self.comboBox_algorithm.setStyleSheet(self.__class__.Styles.combo_box)
    self.comboBox_algorithm.setGeometry(QtCore.QRect(270, 250, 550, 70))
    self.comboBox_algorithm.addItem("Выберите алгоритм...")
    self.comboBox_algorithm.setItemData(0, QtGui.QColor("#A0A0A0"), QtCore.Qt.TextColorRole)
    self.comboBox_algorithm.addItem("Код Джонсона")
    self.comboBox_algorithm.addItem("Код Грея")
    self.comboBox_algorithm.addItem("Редукция веса Хэмминга")
    self.comboBox_algorithm.addItem("Редукция расстояния Хэмминга")
    self.comboBox_algorithm.addItem("Генетический алгоритм")
    self.comboBox_algorithm.addItem("Унитарный код")
    self.comboBox_algorithm.addItem("Последовательное кодирование")
    self.comboBox_algorithm.currentIndexChanged.connect(self.on_algorithm_selected)

    # Стрелка
    self.combo_label = QLabel(self.centralwidget)
    self.combo_label.setGeometry(787, 250, 30, 70)
    self.combo_label.setText("▼")
    self.combo_label.setAlignment(Qt.AlignCenter)
    self.combo_label.setStyleSheet("font-size: 20px; color: white; border: none; background: transparent;")
    self.combo_label.mousePressEvent = lambda e: self.comboBox_algorithm.showPopup()

    # Поле вывода текста
    self.output_layout = QtWidgets.QVBoxLayout()
    self.output_text = QtWidgets.QTextEdit(self.centralwidget)
    self.output_text.setAcceptRichText(True)
    self.output_text.setGeometry(QtCore.QRect(100, 460, 900, 300))
    self.output_text.setStyleSheet(self.__class__.Styles.text_edit)

```

```

self.output_text.setReadOnly(True)
self.output_text.setObjectName("output_text")
self.output_text.setHorizontalScrollBarPolicy(QtCore.Qt.ScrollBarAlwaysOff)

# Кнопка "Сохранить"
self.save_button = QtWidgets.QPushButton(self.centralwidget)
self.save_button.setGeometry(QtCore.QRect(350, 530, 381, 90))
self.save_button.setStyleSheet(self.__class__.Styles.button_main)
self.save_button.setText("Сохранить файл")
self.save_button.setVisible(False) # Скрыта до кодирования
self.save_button.clicked.connect(self.save_modified_file)

self.main_layout = QtWidgets.QVBoxLayout(self.centralwidget)
self.main_layout.addLayout(self.output_layout)

# Кнопка "Закодировать"
self.encodeButton = QtWidgets.QPushButton(self.centralwidget)
self.encodeButton.setGeometry(QtCore.QRect(350, 340, 381, 111))
self.encodeButton.setStyleSheet(self.__class__.Styles.button_main)
self.encodeButton.setObjectName("encodeButton")
self.encodeButton.setText("Закодировать")
self.encodeButton.clicked.connect(self.on_encode_clicked)

Coding.setCentralWidget(self.centralwidget)

def __init__(self):
    self.is_Moore = False
    self.is_Mealy = False
    self.is_fa = False

# ----- ГРАФИЧЕСКИЙ ИНТЕРФЕЙС -----
# ----- ВЫВОД РЕЗУЛЬТАТОВ -----
def show_encoded_states(self, coded_states):
    html_text = """
    <span style='font-size: 24px; background: -webkit-linear-gradient(...) '>
        Итоговые закодированные состояния:<br>
    """
    for state, code in sorted(coded_states.items()):
        html_text += f" {state}: {code}<br>"
    html_text += "</span>"
    self.output_text.setHtml(html_text)

# ----- ВЫВОД РЕЗУЛЬТАТОВ -----
def reset_flags(self):
    self.is_jff = False
    self.is_verilog = False
    self.is_smf = False
    self.coded_states = {}
    self.best_parents = None
    self.selected_trigger = None
    self.selected_algorithm = None
    self.comboBox_algorithm.setCurrentIndex(0)
    self.on_algorithm_selected(0)
    self.iteration_input.setVisible(False)
    self.iteration_input.setPlaceholderText("Число итераций (по умолчанию: 100), макс.: 10000")

self.encodeButton.setGeometry(QtCore.QRect(350, 340, 381, 111))
self.output_text.setGeometry(QtCore.QRect(100, 470, 900, 300))
self.save_button.setGeometry(QtCore.QRect(390, 551, 301, 111))

```

```

        if hasattr(self, 'save_button'):
            self.save_button.setVisible(False)

        self.output_text.clear()
# ----- МОДИФИКАЦИЯ ИЛИ КОНВЕРТАЦИЯ ФАЙЛА -----
    def save_modified_file(self):
        formats = "JFLAP Files (*.jff);;Verilog Files (*.v);;SMF Files (*.smf)"
        file_path, selected_filter = QtWidgets.QFileDialog.getSaveFileName(
            None,
            "Сохранить модифицированный файл",
            "",
            formats
        )

        if not file_path:
            return

        if selected_filter.startswith("JFLAP"):
            if self.is_jff:
                self._save_modified_jff(file_path)
            else:
                self._save_as_jff(file_path)
        elif selected_filter.startswith("Verilog"):
            if self.is_verilog:
                self._save_modified_verilog(file_path)
            else:
                self._save_as_verilog(file_path)
        elif selected_filter.startswith("SMF"):
            if self.is_smf:
                self._save_modified_smf(file_path)
            else:
                self._save_as_smf(file_path)
        else:
            QtWidgets.QMessageBox.warning(None, "Ошибка", "Формат файла не поддерживается!")
# ----- МОДИФИКАЦИЯ -----
    def _save_modified_jff(self, file_path):
        try:
            tree = ET.parse(self.file_path)
            root = tree.getroot()

            print("Текущий coded_states:", self.coded_states)

            for state in root.findall("./state"):
                state_id = state.get('id')
                state_name = self.states[state_id]['name']

                if state_name in self.coded_states:
                    label = state.find('label')
                    if label is None:
                        label = ET.SubElement(state, 'label')
                    label.text = self.coded_states[state_name]

            tree.write(file_path, encoding="utf-8", xml_declaration=True)
            QtWidgets.QMessageBox.information(None, "Успех", "Файл успешно сохранен!")

        except Exception as e:

```

```

        QtWidgets.QMessageBox.critical(None, "Ошибка", f"Ошибка при сохранении: {str(e)}")

def _save_modified_verilog(self, file_path):
    try:
        with open(self.file_path, 'r', encoding='utf-8') as f:
            content = f.read()
        param_pattern = re.compile(r'(parameter\s*)([^\;]+);')
        match = param_pattern.search(content)
        if match:
            new_params = []
            for state_name, code in self.coded_states.items():
                code_decimal = int(code, 2)
                new_params.append(f'{state_name} = {code_decimal}')

            new_param_line = f'parameter {' '.join(new_params)};'
            content = content[:match.start()] + new_param_line + content[match.end():]

        if file_path:
            with open(file_path, 'w', encoding='utf-8') as f:
                f.write(content)
            QtWidgets.QMessageBox.information(None, "Успех", "Файл успешно сохранен!")

    except Exception as e:

        QtWidgets.QMessageBox.critical(None, "Ошибка", f"Ошибка при сохранении: {str(e)}")

def _save_modified_smf(self, file_path):
    try:
        with open(self.file_path, 'r') as f:
            content = f.read()
        for state_name, code in self.coded_states.items():
            new_name = f'{state_name} ({code})'
            content = re.sub(
                rf'(NAME\s*=\s*)" {state_name} ("',
                rf'\1 {new_name} \2',
                content
            )
            content = re.sub(
                rf'(SSTATE\s*=\s*)" {state_name} ("',
                rf'\1 {new_name} \2',
                content
            )
            content = re.sub(
                rf'(DSTATE\s*=\s*)" {state_name} ("',
                rf'\1 {new_name} \2',
                content
            )
        with open(file_path, 'w', encoding='utf-8') as f:
            f.write(content)

        QtWidgets.QMessageBox.information(None, "Успех", "SMF-файл успешно модифицирован с кодировками!")

    except Exception as e:
        QtWidgets.QMessageBox.critical(None, "Ошибка", f"Ошибка при сохранении: {str(e)}")
# ----- МОДИФИКАЦИЯ -----

```

```

# ----- КОНВЕРТАЦИЯ -----
def _save_as_jff(self, file_path):
    if not hasattr(self, 'states') or not hasattr(self, 'transitions'):
        QtWidgets.QMessageBox.warning(None, "Ошибка", "Нет данных для сохранения.")
        return

    automaton = ET.Element("structure")
    type_elem = ET.SubElement(automaton, "type")
    if self.is_Moore:
        type_elem.text = "moore"
    elif self.is_Mealy:
        type_elem.text = "mealy"
    else:
        type_elem.text = "fa"
    auto_elem = ET.SubElement(automaton, "automaton")

    id_counter = 0
    state_name_to_id = {}

    # Добавление состояний
    for state_name in self.states:
        state_elem = ET.SubElement(auto_elem, "state", id=str(id_counter), name=state_name)
        code = self.coded_states.get(state_name, None)
        if code:
            label_elem = ET.SubElement(state_elem, "label")
            label_elem.text = code
            state_name_to_id[state_name] = str(id_counter)
            id_counter += 1

    # Добавление переходов
    for t in self.transitions:
        if t['condition'] != '1': # Игнорируем безусловные переходы
            trans_elem = ET.SubElement(auto_elem, "transition")
            ET.SubElement(trans_elem, "from").text = state_name_to_id[t['from']]
            ET.SubElement(trans_elem, "to").text = state_name_to_id[t['to']]
            ET.SubElement(trans_elem, "read").text = t['condition']

    # Создание XML-дерева
    tree = ET.ElementTree(automaton)
    try:
        tree.write(file_path, encoding="utf-8", xml_declaration=True)
        QtWidgets.QMessageBox.information(None, "Успех", "Файл успешно сохранен в JFF!")
    except Exception as e:
        QtWidgets.QMessageBox.critical(None, "Ошибка", f"Ошибка при сохранении: {str(e)}")

def _save_as_verilog(self, file_path):
    if not hasattr(self, 'states') or not hasattr(self, 'transitions'):
        QtWidgets.QMessageBox.warning(None, "Ошибка", "Нет данных для сохранения.")
        return

    input_signals = set()
    for t in self.transitions:
        if t['condition'] != '1':
            signals = re.findall(r'\b([A-Za-z]\d+)\b', t['condition'])
            input_signals.update(signals)

    # Определение разрядности

```

```
num_bits = max((len(code) for code in self.coded_states.values()), default=1)
```

```
with open(file_path, 'w', encoding='utf-8') as f:
```

```
    f.write(f'module verilog_file (\n")
```

```
    f.write(f'    input clk,\n")
```

```
    f.write(f'    input reset")
```

```
    # Добавление входных сигналов
```

```
    for signal in sorted(input_signals, key=lambda x: (x[0], int(x[1:])))
```

```
        f.write(f',\n    input {signal}"))
```

```
    f.write("\n);\n\n")
```

```
    # Параметры состояний
```

```
    f.write("\n    // State encoding\n")
```

```
    param_lines = []
```

```
    for state, code in self.coded_states.items():
```

```
        decimal_value = int(code, 2)
```

```
        param_lines.append(f'{state}={decimal_value}"))
```

```
    f.write("    parameter " + ",\n        ".join(param_lines) + ";\n\n")
```

```
    # Регистры состояний
```

```
    f.write(f'    reg [{num_bits - 1}:0] fstate;\n")
```

```
    f.write(f'    reg [{num_bits - 1}:0] reg_fstate;\n\n")
```

```
    # Блок синхронной логики
```

```
    f.write("    always @(posedge clk)\n")
```

```
    f.write("    begin\n")
```

```
    f.write("        if (clk) begin\n")
```

```
    f.write("            fstate <= reg_fstate;\n")
```

```
    f.write("        end\n")
```

```
    f.write("    end\n\n")
```

```
    # Блок комбинационной логики
```

```
    f.write("    always @(fstate or reset")
```

```
    for signal in input_signals:
```

```
        f.write(f' or {signal}"))
```

```
    f.write(")\n")
```

```
    f.write("    begin\n")
```

```
    f.write("        if (reset) begin\n")
```

```
    initial_state = sorted(self.coded_states.keys())[0]
```

```
    f.write(f'        reg_fstate <= {initial_state};\n")
```

```
    f.write("    end\n")
```

```
    f.write("    else begin\n")
```

```
    f.write("        case (fstate)\n")
```

```
    # Группировка переходов по состояниям
```

```
    transitions_by_state = defaultdict(list)
```

```
    for t in self.transitions:
```

```
        transitions_by_state[t['from']].append((t['to'], t['condition']))
```

```
    # Генерация условных блоков
```

```
    for state in sorted(self.coded_states.keys()):
```

```
        transitions = transitions_by_state.get(state, [])
```

```
        f.write(f'        {state}: begin\n")
```

```
        if transitions:
```

```

        for i, (to_state, condition) in enumerate(transitions):
            if_str = "if" if i == 0 else "else if"
            converted_condition = self._convert_condition_to_verilog(condition)
            f.write(f"            {if_str} ( {converted_condition} )\n")
            f.write(f"                reg_fstate <= {to_state};\n")

            f.write("        else\n")
            f.write(f"            reg_fstate <= {state};\n")
        else:
            f.write("            reg_fstate <= {state};\n")

        f.write("    end\n")

    f.write("endcase\n")
    f.write("end\n")
    f.write("end\n\n")
    f.write("endmodule\n")

    QtWidgets.QMessageBox.information(None, "Успех", "Файл успешно сохранен в Verilog!")

def _convert_condition_to_verilog(self, condition):
    # Заменяем X == 0 на X == 1'b0 и X == 1 на X == 1'b1
    condition = re.sub(r'([A-Za-z]\d+)\s*==\s*1\b', r"\1 == 1'b1", condition)
    condition = re.sub(r'([A-Za-z]\d+)\s*==\s*0\b', r"\1 == 1'b0", condition)

    return condition

def point_on_circle_edge(self, center, target, state_radius, direction):
    dx = target[0] - center[0]
    dy = target[1] - center[1]
    dist = math.hypot(dx, dy)
    rasst = 5
    if dist == 0:
        return (center[0] + state_radius, center[1])
    dx /= dist
    dy /= dist

    return (int(center[0] + dx * state_radius * direction), int(center[1] + dy * state_radius * direction))

def _save_as_smf(self, file_path):
    state_radius = 32
    if not hasattr(self, 'states') or not hasattr(self, 'transitions'):
        QtWidgets.QMessageBox.warning(None, "Ошибка", "Нет данных для сохранения.")
        return

    with open(file_path, 'w', encoding='utf-8') as f:
        f.write('VERSION = "2.0";\nHEADER\n\n')

        # GENERAL блок
        f.write('    GENERAL {\n        RMODE = "S";\n        RA_LEVEL = "H";\n        HOPT = "VLOG";\n    }\n\n')

        input_signals = set()
        for t in self.transitions:
            if t['condition'] != '1':
                signals = re.findall(r'\b([A-Za-z]\d+)\b', t['condition'])
                input_signals.update(signals)

```

```

# SPORT блоки
f.write(
    '    SPORT{\n        NAME = "reset";\n        PTYPE = "RI";\n        REG = "N";\n        OUTS =
"N";\n    }\n')
f.write(
    '    SPORT{\n        NAME = "clock";\n        PTYPE = "CI";\n        REG = "N";\n        OUTS =
"N";\n    }\n')

for signal in sorted(input_signals, key=lambda x: (x[0], int(x[1:]) if x[1:].isdigit() else 0)):
    f.write(
        f'    SPORT{{\n        NAME = "{signal}";\n        PTYPE = "OI";\n        REG = "N";\n        OUTS
= "N";\n    }}\n')

f.write("\n")

all_state_names = set()
for t in self.transitions:
    all_state_names.add(t['from'])
    all_state_names.add(t['to'])

all_state_names = sorted(all_state_names)

# Расположение состояний по кругу
num_states = len(all_state_names)
center_x, center_y, radius = 300, 300, 200
positions = {}
for i, state in enumerate(all_state_names):
    angle = 2 * math.pi * i / num_states
    x = int(center_x + radius * math.cos(angle))
    y = int(center_y + radius * math.sin(angle))
    positions[state] = (x, y)
    print("Состояние:", state)
    print("Координаты: ", x, y)

# STATE блоки
for state in all_state_names:
    stype = self.states.get(state, {}).get('type', 'NR')
    x, y = positions[state]
    f.write(
        f'    STATE{{\n        NAME = "{state}";\n        STYPE = "{stype}";\n        PT = ({x},{y});\n
}}\n')

f.write("\n")

# TRANS блоки
for t in self.transitions:
    from_state = t['from']
    to_state = t['to']
    cond = t['condition']

    f.write(
        f'    \nTRANS{{\n        SSTATE = "{from_state}";\n        DSTATE = "{to_state}";\n        EQ =
"{cond}";\n')

```



```

x1, y1 = positions[from_state]
x2, y2 = positions[to_state]

center_from = (x1 + state_radius, y1 + state_radius)
center_to = (x2 + state_radius, y2 + state_radius)
print ("Состояние: ", from_state)
print ("Координаты: ", center_from, center_to)

if from_state == to_state:
    cx, cy = center_from
    f.write(f'    PT = ({cx + 30},{cy});\n')
    f.write(f'    PT = ({cx + state_radius},{cy - state_radius});\n')
    f.write(f'    PT = ({cx + 2 * state_radius},{cy});\n')
    f.write(f'    PT = ({cx + state_radius + 10},{cy + state_radius - 30});\n')
    f.write(f'    PT = ({cx},{cy});\n')
else:
    start_pt = self.point_on_circle_edge(center_from, center_to, state_radius, 1)
    end_pt = self.point_on_circle_edge(center_to, center_from, state_radius, 1)
    for i in range(5):
        ratio = i / 3
        pt_x = int(start_pt[0] + ratio * (end_pt[0] - start_pt[0]))
        pt_y = int(start_pt[1] + ratio * (end_pt[1] - start_pt[1]))
        f.write(f'    PT = ({pt_x},{pt_y});\n')

    f.write('    }\n')

f.write('\n\nEND\n')
QtWidgets.QMessageBox.information(None, "Успех", "Файл успешно сохранен в SMF!")
# ----- КОНВЕРТАЦИЯ -----
# ----- ПОЯВЛЕНИЕ ПОЛЯ ВВОДА ДЛЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА -----
# -----
def on_algorithm_selected(self, index):
    if index > 0:
        self.selected_algorithm = self.comboBox_algorithm.currentText() # Сохраняем выбранный
алгоритм
        self.is_genetic = "Генетический" in self.comboBox_algorithm.currentText()
        self.iteration_input.setVisible(self.is_genetic)
        if self.is_genetic:
            num_states = len(self.states) if hasattr(self, 'states') and self.states else 0
            num_triggers = math.ceil(math.log2(num_states)) if num_states > 0 else 1 # защита от log2(0)
            iterations_umolch = (num_triggers / num_states) * 50 if num_states > 0 else 100 # если автомат
пустой — пусть будет 100

            default_iterations = min(iterations_umolch, 10000)

            # Обновление placeholder текста
            self.iteration_input.setPlaceholderText(
                f"Число итераций (по умолчанию: {int(default_iterations)}, макс.: 10000)"
            )
            if self.save_button.isVisible() == False:
                self.encodeButton.setGeometry(QtCore.QRect(350, 430, 381, 111))
                self.output_text.setGeometry(QtCore.QRect(100, 570, 900, 300))
            else:
                self.encodeButton.setGeometry(QtCore.QRect(350, 420, 381, 111))
                self.save_button.setGeometry(QtCore.QRect(390, 550, 301, 90))
                self.output_text.setGeometry(QtCore.QRect(100, 650, 900, 300))
        else:

```

```

        if self.save_button.isVisible() == False:
            self.encodeButton.setGeometry(QtCore.QRect(350, 340, 381, 111))
            self.output_text.setGeometry(QtCore.QRect(100, 470, 900, 300))
        else:
            self.save_button.setGeometry(QtCore.QRect(390, 470, 301, 80))
            self.iteration_input.setPlaceholderText("Число итераций (по умолчанию: 100), макс.:
10000")
            self.encodeButton.setGeometry(QtCore.QRect(350, 340, 381, 111))
            self.output_text.setGeometry(QtCore.QRect(100, 570, 900, 300))
# ----- ПОЯВЛЕНИЕ ПОЛЯ ВВОДА ДЛЯ ГЕНЕТИЧЕСКОГО АЛГОРИТМА -----
# ----- ВЫПАДАЮЩИЙ СПИСОК И ЛОГИКА КНОПКИ "ЗАКОДИРОВАТЬ" -----
def on_encode_clicked(self):
    #Общие стили
    warning_style = "color: #FF69B4; font-size: 24px; font-weight: bold; text-align: center;"
    message_style = "color: #FFFFFF; font-size: 24px; text-align: center;"
    highlight_style = "color: #00FFFF; font-weight: bold;"
    divider = "<pre>-----</pre>"

    if hasattr(self, 'save_button'):
        self.save_button.setVisible(False)
        self.output_text.setGeometry(QtCore.QRect(100, 570, 900, 300))
        self.save_button.setGeometry(QtCore.QRect(390, 360, 301, 80))
        self.output_text.clear()

    # Проверка, загружен ли файл с автоматом
    if not hasattr(self, 'states') or not self.states:
        error_message = f"""
        <p style="{warning_style}">⚠ Ошибка! ⚠ </p>
        <p style="{message_style}">Сначала загрузите файл с автоматом!</p>
        """
        self.output_text.insertHtml(error_message)
        self.output_text.insertPlainText("\n")
        return

    # Проверка, выбран ли алгоритм
    current_text = self.comboBox_algorithm.currentText()
    if current_text == "Выберите алгоритм..." or not hasattr(self, 'selected_algorithm'):
        error_message = f"""
        <p style="{warning_style}">⚠ Ошибка! ⚠ </p>
        <p style="{message_style}">Сначала выберите алгоритм из списка!</p>
        """
        self.output_text.insertHtml(error_message)
        self.output_text.insertPlainText("\n")
        return

    #Буфер для вывода
    output_buffer = []

    # Начало выполнения (первая точка отсчёта времени)
    start_time = time.time()

    if "Редукция веса Хэмминга" in self.selected_algorithm:
        trigger_type = "D-триггеров"
        show_warning = True
    elif "Редукция расстояния Хэмминга" in self.selected_algorithm:

```

```

        trigger_type = "JK- и Т-триггеров"
        show_warning = True
    else:
        show_warning = False

    if show_warning:
        trigger_info = f"""
        <div style="{message_style}">
            Выбранный алгоритм наиболее эффективен при использовании<br>
            <span style="{highlight_style}">{trigger_type}</span>
        </div>
        {divider}
        """
        output_buffer.append(trigger_info)

    try:
        if "Редукция веса Хэмминга" in self.selected_algorithm:
            self.pairs_finding()
            success = True
        elif "Редукция расстояния Хэмминга" in self.selected_algorithm:
            self.on_clicked()
            success = True
        elif "Генетический алгоритм" in self.selected_algorithm:
            self.generate_encodings()
            success = True
        elif "Код Грея" in self.selected_algorithm:
            self.encode_gray_code()
            success = True
        elif "Код Джонсона" in self.selected_algorithm:
            self.encode_johnson_code()
            success = True
        elif "Унитарный код" in self.selected_algorithm:
            self.one_hot_encoding()
            success = True
        elif "Последовательное кодирование" in self.selected_algorithm:
            self.encode_sequential()
            success = True

    else:
        success = False

    if success:
        # Добавляем результаты кодирования в буфер
        if hasattr(self, 'coded_states') and self.coded_states:
            output_buffer.append("Итоговые закодированные состояния:")
            for state, code in sorted(self.coded_states.items()):
                output_buffer.append(f"{state}: {code}")

        if hasattr(self, 'save_button'):
            self.save_button.setVisible(True)
            self.output_text.setGeometry(QtCore.QRect(100, 570, 900, 300))
            self.save_button.setGeometry(QtCore.QRect(390, 470, 301, 80))
            if self.is_genetic:
                self.output_text.setGeometry(QtCore.QRect(100, 650, 900, 300))
                self.save_button.setGeometry(QtCore.QRect(390, 550, 301, 80))
    except Exception as e:
        output_buffer.append(f"Ошибка: {str(e)}")

```

```

        success = False

# Окончание выполнения (вторая точка отсчёта времени)
end_time = time.time()
execution_time = end_time - start_time

print(f"Время выполнения кодирования ({self.selected_algorithm}): {execution_time:.5f} с.")

if "Генетический алгоритм" in self.selected_algorithm:
    time_message = (
        f"<span style='color: #00FFFF; font-size: 24px; font-family: Courier;'>Время выполнения  
кодирования:</span> '
        f"<span style='color: #FFFFFF; font-size: 24px; font-family: Courier;'>{execution_time:.5f}  
с.</span>'
    )
    else:
        execution_time_ms = execution_time * 1000 # перевод в мс
        time_message = (
            f"<span style='color: #00FFFF; font-size: 24px; font-family: Courier;'>Время выполнения  
кодирования:</span> '
            f"<span style='color: #FFFFFF; font-size: 24px; font-family: Courier;'>{execution_time_ms:.5f} мс.</span>'
        )
        output_buffer.append(time_message)
        if hasattr(self, 'logical_elements') and self.logical_elements is not None:
            output_buffer.append(
                f"<span style='color: #FFA500; font-size: 24px; font-family: Courier;'>Количество логических  
элементов:</span> '
                f"<span style='color: #FFFFFF; font-size: 24px; font-family: Courier;'>{self.logical_elements}</span>'
            )
        if hasattr(self, 'num_bits') and self.num_bits is not None:
            output_buffer.append(
                f"<span style='color: #32CD32; font-size: 24px; font-family: Courier;'>Количество  
триггеров:</span> '
                f"<span style='color: #FFFFFF; font-size: 24px; font-family: Courier;'>{self.num_bits}</span>'
            )
        self.output_text.clear()
        self.output_text.setHtml("<br>".join(output_buffer))
# ----- ВЫПАДАЮЩИЙ СПИСОК И ЛОГИКА КНОПКИ "ЗАКОДИРОВАТЬ" -----
# ----- ЗАГРУЗКА ФАЙЛА -----
def load_file(self):
    self.reset_flags()
    self.output_text.clear()
    success_style = ""
    <span style='color: #4CAF50; font-size: 24px; font-weight: bold;'>
        ✓ Файл успешно загружен!
    </span>
    ""
    file_info_style = ""
    <span style='color: #FF69B4; font-size: 24px;'>
        {content}
    </span>
    ""

    file_path, _ = QtWidgets.QFileDialog.getOpenFileName(

```

```

None,
"Выберите файл",
"""
"JFLAP Files (*.jff);;Verilog Files (*.v);;SMF Files (*.smf)"
)

if file_path:
    self.file_path = file_path
    file_name = file_path.split('/')[-1] # Получение только имени файла

    ext_actions = {
        ".jff": ("jff", "JFLAP автомат", self.parse_jff_file),
        ".v": ("verilog", "Verilog модуль", self.parse_verilog_file),
        ".smf": ("smf", "SMF файл", self.parse_smf_file)
    }

    message = f"""
    {success_style}
    <br>
    {file_info_style.format(content=f"<b>Название:</b> {file_name}")}
    """

    # Определение типа файла и вызов соответствующего парсера
    for ext, (flag, label, parser) in ext_actions.items():
        if file_path.endswith(ext):
            self.is_jff = (flag == "jff")
            self.is_verilog = (flag == "verilog")
            self.is_smf = (flag == "smf")

            message += f"""
            <br>
            {file_info_style.format(content=f"<b>Тип:</b> {label}")}
            """

            parser()

            if not self.states and not self.transitions:
                empty_style = """
                <span style='color: #FF5252; font-size: 24px; font-weight: bold;'>
                ⚠ Загружен пустой файл или автомат ⚠
                </span>
                """
                message += f"<br><br>{empty_style}"

            break
    self.output_text.setHtml(message)
else:
    self.output_text.setHtml("""
    <span style='color: #FF5252; font-size: 24px;'>
    Файл не выбран!
    </span>
    """)

# ----- ЗАГРУЗКА ФАЙЛА -----
# ----- ПАРСИНГ ФАЙЛА -----

def parse_jff_file(self):
    if not self.file_path:
        print("Файл не выбран!")
    return

```

```

try:
    self.is_Moore = False
    self.is_Mealy = False
    self.is_fa = False
    # Загрузка и разбор XML
    tree = ET.parse(self.file_path)
    root = tree.getroot()

    # Определение типа автомата
    type_elem = root.find('type')
    if type_elem is not None:
        type_text = type_elem.text.lower()
        if 'moore' in type_text:
            self.is_Moore = True
        elif 'mealy' in type_text:
            self.is_Mealy = True
        else:
            self.is_fa = True
    else:
        self.is_fa = True

    print(f"Тип автомата: {'Moore' if self.is_Moore else 'Mealy' if self.is_Mealy else 'FA'}")

    # Извлечение состояний
    self.states = {}

    for state in root.findall("./state"):
        state_id = state.get('id')
        state_name = state.get('name')
        output = state.find('output').text if state.find('output') is not None else "

        self.states[state_id] = {
            'name': state_name,
            'output': output
        }

    print("\nСостояния:")
    for state_id, state_info in self.states.items():
        print(f"State ID: {state_id}, Name: {state_info['name']}, Output: {state_info['output']}")

    # Извлечение переходов
    self.transitions = []
    self.graph = {}

    for transition in root.findall("./transition"):
        from_state_id = transition.find('from').text
        to_state_id = transition.find('to').text
        from_state_name = self.states[from_state_id]['name'] if from_state_id in self.states else
from_state_id
        to_state_name = self.states[to_state_id]['name'] if to_state_id in self.states else to_state_id
        read_condition = transition.find('read').text
        transout = transition.find('transout').text if transition.find('transout') is not None else "

        self.transitions.append({
            'from': from_state_name,
            'to': to_state_name,
            'condition': read_condition,

```

```

        'output': transout
    })

    # Добавление в граф
    if from_state_name not in self.graph:
        self.graph[from_state_name] = []

    self.graph[from_state_name].append((to_state_name, read_condition))

    print("\nПереходы:")
    for t in self.transitions:
        print(f"From: {t['from']}, To: {t['to']}, Condition: {t['condition']}, Output: {t['output']}")

    print("\nГраф переходов:")
    sorted_keys = sorted(self.graph.keys(), key=self.extract_number)
    for key in sorted_keys:
        print(f"{key}: {self.graph[key]}")

    except Exception as e:
        print(f"Ошибка при разборе JFLAP файла: {e}")

    def parse_verilog_file(self):
        if not self.file_path:
            print("Файл не выбран!")
            return

        try:
            with open(self.file_path, 'r') as f:
                content = f.read()

            self.states = {}
            self.transitions = []
            self.graph = defaultdict(list)
            self.is_verilog = True
            self.is_Moore = True
            self.is_Mealy = False

            # Извлечение состояний
            param_match = re.search(r'parameter\s+([^\s;]+);', content)
            if param_match:
                params = [p.strip() for p in param_match.group(1).split(',')]
                for param in params:
                    state_match = re.match(r'(\w+)\s*=\s*(\d+)', param)
                    if state_match:
                        state_name = state_match.group(1)
                        state_id = state_match.group(2)
                        self.states[state_name] = {'id': state_id, 'name': state_name, 'output': None}

            # Извлечение условных блоков
            case_blocks = re.finditer(r'case\s*(.*?)(.*?)endcase', content, re.DOTALL)
            for case in case_blocks:
                case_content = case.group(1)
                state_blocks = re.split(r'\b(\w+)\s*begin', case_content)[1:]

                for i in range(0, len(state_blocks), 2):
                    from_state = state_blocks[i].strip()

```

```

if from_state == 'default':
    continue

transitions_block = state_blocks[i + 1]

if from_state not in self.graph:
    self.graph[from_state] = []

    # Извлечение выходных выражений
    output_matches = re.findall(r'([a-zA-Z_][a-zA-Z0-9_]*)\s*<=\s*([^;]+);',
transitions_block)
    if output_matches:
        output_list = []
        for output_name, output_expr in output_matches:
            output_expr = output_expr.replace('&', 'and').replace('|', 'or').replace('~',
                                                                                       'not ').replace(
                '1'b1", '1').replace("1'b0", '0')
            output_list.append(f'{output_name} = {output_expr}')
        self.states[from_state]['output'] = output_list
        print(f"[Verilog] Выходы для состояния {from_state}: {output_list}")

transitions = re.finditer(
    r'if\s*\(((.*?)\)\s*begin\s*reg_fstate\s*<=\s*(\w+)\s*;\s*((?:[a-zA-Z_][a-zA-Z0-9_]*\s*<=\s*([^\s;]+);\s*))',
    transitions_block, re.DOTALL
)
for trans in transitions:
    condition = trans.group(1).strip()
    to_state = trans.group(2).strip()
    outputs_block = trans.group(3).strip()
    output_list = []
    if outputs_block:
        # Извлечение всех выходных переменных
        output_matches = re.findall(r'([a-zA-Z_][a-zA-Z0-9_]*)\s*<=\s*([^\s;]+);', outputs_block)
        for output_name, output_expr in output_matches:
            output_expr = output_expr.replace('&', 'and').replace('|', 'or').replace('~', 'not
').replace("1'b1", '1').replace("1'b0", '0')
            output_list.append(f'{output_name} = {output_expr}')
        if output_list:
            self.is_Moore = False
            self.is_Mealy = True
        self._add_transition(from_state, to_state, condition, output_list if output_list else None)

# Безусловные переходы
unconditional = re.finditer(
    r'reg_fstate\s*<=\s*(\w+)\s*;\s*((?:[a-zA-Z_][a-zA-Z0-9_]*\s*<=\s*([^\s;]+);\s*))',
    transitions_block
)
for uncond in unconditional:
    to_state = uncond.group(1).strip()
    outputs_block = uncond.group(2).strip()
    output_list = []
    if outputs_block:
        output_matches = re.findall(r'([a-zA-Z_][a-zA-Z0-9_]*)\s*<=\s*([^\s;]+);', outputs_block)
        for output_name, output_expr in output_matches:
            output_expr = output_expr.replace('&', 'and').replace('|', 'or').replace('~', 'not
').replace("1'b1", '1').replace("1'b0", '0')

```



```

        output_list.append(f'{output_name} = {output_expr}')
    if output_list:
        self.is_Moore = False
        self.is_Mealy = True
    if to_state != from_state:
        self._add_transition(from_state, to_state, '1', output_list if output_list else None)

# Добавление состояний без переходов
for state in self.states:
    if state not in self.graph:
        self.graph[state] = []

print("\nCостояния (Verilog):")
for state_name, state_info in self.states.items():
    print(f'State: {state_name}, ID: {state_info['id']}, Output: {state_info.get('output', 'None')}')

print("\nПереходы (Verilog):")
for t in self.transitions:
    print(f'From: {t['from']}, To: {t['to']}, Condition: {t['condition']}, Output: {t.get('output', 'None')}')

print("\nГраф переходов:")
sorted_keys = sorted(self.graph.keys(), key=lambda x: int(self.states[x]['id']))
for key in sorted_keys:
    print(f'{key}: {self.graph[key]}')

except Exception as e:
    print(f'Ошибка при разборе Verilog файла: {e}')

def parse_smf_file(self):
    try:
        with open(self.file_path, 'r', encoding='utf-8') as f:
            content = f.read()

        self.states = {}
        self.transitions = []
        self.graph = {}
        self.is_smf = True
        self.is_Moore = False
        self.is_Mealy = False

        # Парсинг STATE блоков
        state_pattern = re.compile(r'STATE\s*\{s*NAME\s*=\s*"([^\"]+)"\s*;\s*?STYPE\s*=\s*"([^\"]+)"',
re.DOTALL)
        for match in state_pattern.finditer(content):
            state_name = match.group(1)
            state_type = match.group(2)
            self.states[state_name] = {'name': state_name, 'type': state_type}
            if state_type.lower() == 'moore':
                self.is_Moore = True
            elif state_type.lower() == 'mealy':
                self.is_Mealy = True

        # Парсинг TRANS блоков
        trans_pattern = re.compile(

```

```

r'TRANS\s*{\s*SSTATE\s*=\s*"([^"]+)"\s*;\s*DSTATE\s*=\s*"([^"]+)"\s*;\s*EQ\s*=\s*"([^"]*)"',
re.DOTALL)
    for match in trans_pattern.finditer(content):
        from_state = match.group(1)
        to_state = match.group(2)
        condition = match.group(3).strip()
        if not condition:
            condition = '1'
        self.transitions.append({'from': from_state, 'to': to_state, 'condition': condition})

    # Формирование графа переходов
    if from_state not in self.graph:
        self.graph[from_state] = []
    self.graph[from_state].append((to_state, condition))

    # Добавление состояний без переходов
    for state in self.states:
        if state not in self.graph:
            self.graph[state] = []

    print("\nCостояния (SMF):")
    for s in self.states:
        print(f'{s}: {self.states[s]}')

    print("\nПереходы (SMF):")
    for t in self.transitions:
        print(t)

    print("\nГраф переходов (SMF):")
    for g in self.graph:
        print(f'{g}: {self.graph[g]}')

    except Exception as e:
        print(f'Ошибка при разборе SMF файла: {e}')

def _add_transition(self, from_state, to_state, condition, output=None):
    transition = {'from': from_state, 'to': to_state, 'condition': condition}
    if output:
        transition['output'] = output
    self.transitions.append(transition)
    self.graph[from_state].append((to_state, condition))

def extract_number(self, state_name): #Извлечение числовой части для сортировки
    import re
    match = re.search(r'\d+', state_name)
    return int(match.group()) if match else float(
        'inf')

def __init__(self):
    self.graph = {} # Граф состояний
# ----- ПАРСИНГ ФАЙЛА -----
# ----- УНИТАРНЫЙ КОД -----
def one_hot_encoding(self):
    tracemalloc.start()
    start_time = time.perf_counter()
    state_names = sorted([state_data['name'] for state_data in self.states.values()])

```

```

num_states = len(state_names)
num_bits = num_states
coded_states = {}

for i, state in enumerate(state_names):
    code = ['0'] * num_bits
    code[i] = '1'
    coded_states[state] = ''.join(code)

try:
    logical_elements = self.count_logical_elements(coded_states)
except Exception as e:
    print(f"Ошибка при подсчёте логических элементов: {e}")
    logical_elements = None

current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()
end_time = time.perf_counter()
execution_time = (end_time - start_time) * 1000 # в мс
print(f"Пиковое использование памяти: {peak / 1024:.2f} КБ")
output_buffer = []
output_buffer.append(f"<h3 style='color:#00FFFF;'>Унитарный код:</h3>")
for state, code in sorted(coded_states.items()):
    output_buffer.append(f"{state}: {code}")

output_buffer.append(
    f"<span style='color: #00FFFF;'>Время выполнения:</span> <span style='color: #FFFFFF;'>{execution_time:.5f} мс</span>"
)
output_buffer.append(
    f"<span style='color: #FFA500;'>Логических элементов:</span> <span style='color: #FFFFFF;'>{logical_elements}</span>"
)

self.coded_states = coded_states
self.logical_elements = logical_elements
self.num_bits = num_bits
self.output_text.setHtml("<br>".join(output_buffer))
# ----- УНИТАРНЫЙ КОД -----
# ----- ПОСЛЕДОВАТЕЛЬНОЕ КОДИРОВАНИЕ -----
# -----

def encode_sequential(self):
    tracemalloc.start()
    start_time = time.perf_counter()
    state_names = sorted([state_data['name'] for state_data in self.states.values()])
    num_states = len(state_names)
    num_bits = (num_states - 1).bit_length()
    coded_states = {}
    for i, state in enumerate(state_names):
        code = format(i, f'0{num_bits}b')
        coded_states[state] = code

    try:
        logical_elements = self.count_logical_elements(coded_states)
    except Exception as e:
        print(f"Ошибка при подсчёте логических элементов: {e}")
        logical_elements = None

```

```

current, peak = tracemalloc.get_traced_memory()

end_time = time.perf_counter()
execution_time = (end_time - start_time) * 1000
tracemalloc.stop()
print(f"Пиковое использование памяти: {peak / 1024:.2f} КБ")
output_buffer = []
output_buffer.append(f'<h3 style="color:#00FFFF;">Последовательное кодирование:</h3>')
for state, code in sorted(coded_states.items()):
    output_buffer.append(f'{state}: {code}')

output_buffer.append(
    f'<span style="color: #00FFFF;">Время выполнения:</span> <span style="color:
#FFFFFF;">{execution_time:.5f} мс</span>'
)
output_buffer.append(
    f'<span style="color: #FFA500;">Логических элементов:</span> <span style="color:
#FFFFFF;">{logical_elements}</span>'
)

self.coded_states = coded_states
self.logical_elements = logical_elements
self.num_bits = num_bits
self.output_text.setHtml("<br>".join(output_buffer))
# ----- ПОСЛЕДОВАТЕЛЬНОЕ КОДИРОВАНИЕ -----
# ----- КОДИРОВАНИЕ ГРЕЯ -----
def encode_gray_code(self):
    tracemalloc.start()
    state_names = sorted([state_data['name'] for state_data in self.states.values()])
    num_states = len(state_names)
    num_bits = (num_states - 1).bit_length()

    def gray_code(n):
        return n ^ (n >> 1)
    codes = []
    for i in range(num_states):
        gray = gray_code(i)
        code_str = format(gray, f'0{num_bits}b')
        codes.append(code_str)

    coded_states = {}
    for i, state in enumerate(state_names):
        coded_states[state] = codes[i]

    try:
        logical_elements = self.count_logical_elements(coded_states)
        print(f"Количество логических элементов (Код Джонсона): {logical_elements}")
    except Exception as e:
        print(f"Ошибка при подсчете логических элементов: {e}")
        logical_elements = None

current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Пиковое использование памяти: {peak / 1024:.2f} КБ")

```

```

self.coded_states = coded_states
self.logical_elements = logical_elements
self.num_bits = num_bits

self.output_text.clear()
self.show_encoded_states(self.coded_states)
# ----- КОДИРОВАНИЕ ГРЕЯ -----
# ----- КОДИРОВАНИЕ ДЖОНСОНА -----
----
def encode_johnson_code(self):
    tracemalloc.start()
    state_names = sorted([state_data['name'] for state_data in self.states.values()])
    num_states = len(state_names)
    num_bits = (num_states - 1).bit_length()

    max_codes = 2 ** num_bits
    if num_states > max_codes:
        self.output_text.setHtml(
            "<span style='color: red; font-size: 24px;'>Ошибка: количество состояний превышает
ВОЗМОЖНОЕ количество кодов Джонсона!</span>"
        )
        return

    current_code = ['0'] * num_bits
    codes = []

    for _ in range(num_states):
        codes.append("".join(current_code))
        shifted = current_code[-1:] + current_code[:-1]
        shifted[0] = '1' if current_code[-1] == '0' else '0'
        current_code = shifted

    coded_states = {}
    for i, state in enumerate(state_names):
        coded_states[state] = codes[i]

    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    try:
        logical_elements = self.count_logical_elements(coded_states)
        print(f"Количество логических элементов: {logical_elements}")
    except Exception as e:
        print(f"Ошибка при подсчете логических элементов: {e}")
        logical_elements = None

    print(f"Пиковое использование памяти: {peak / 1024:.2f} КБ")

    self.coded_states = coded_states

    output_buffer = []
    output_buffer.append("Итоговые закодированные состояния:")
    for state, code in sorted(coded_states.items()):
        output_buffer.append(f"{state}: {code}")
    execution_time = time.time() - tracemalloc.start_time if hasattr(tracemalloc, 'start_time') else 0

```

```

        result_html = "<div><h3 style='color: #00FFFF; font-size: 26px; font-family: Courier;'>Итоговые  

закодированные состояния:</h3>"
        for state, code in sorted(coded_states.items()):
            result_html += f"<p style='color: #FFFFFF; font-size: 24px; font-family: Courier;'>{state}:  

{code}</p>"

        self.logical_elements = logical_elements
        self.num_bits = num_bits

        self.output_text.clear()
        self.output_text.setHtml(result_html)
# ----- КОДИРОВАНИЕ ДЖОНСОНА -----
----
# ----- РЕДУКЦИЯ РАССТОЯНИЯ ХЭММИНГА -----
-----
def on_clicked(self):
    tracemalloc.start()
    weight_table, state_names = self.build_weight_table()
    self.print_weight_table(weight_table, state_names)

    coded_states = {}

    while len(coded_states) < len(state_names):
        previous_length = len(coded_states)
        coded_states = self.encode_states()

        if len(coded_states) == previous_length:
            print("Невозможно закодировать оставшиеся состояния.")
            break
    # Сохранение закодированных состояний
    self.coded_states = coded_states
    print(coded_states)
    print("Итоговые закодированные состояния:")
    # Сортировка состояний
    sorted_states = sorted(coded_states.keys())
    for state in sorted_states:
        print(f"{state}: {coded_states[state]}")

    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    try:
        logical_elements = self.count_logical_elements(coded_states)
        print(f"Количество логических элементов (Код Джонсона): {logical_elements}")
    except Exception as e:
        print(f"Ошибка при подсчете логических элементов: {e}")
        logical_elements = None

    print(f"Пиковое использование памяти: {peak / 1024:.2f} КБ")

    self.logical_elements = logical_elements

    self.output_text.clear()
    self.show_encoded_states(self.coded_states)

def build_weight_table(self):
    state_names = sorted([state_data['name'] for state_data in self.states.values()])

```

```

# Инициализация таблицы весов
weight_table = {p: {q: "" for q in state_names} for p in state_names}

# Её заполнение
for transition in self.transitions:
    from_state = transition['from']
    to_state = transition['to']
    if from_state != to_state:
        if weight_table[from_state][to_state] == "":
            weight_table[from_state][to_state] = 0
            weight_table[from_state][to_state] += 1

        if weight_table[to_state][from_state] == "":
            weight_table[to_state][from_state] = 0
            weight_table[to_state][from_state] += 1

# Добавление столбца w(p)
for p in state_names:
    weight_table[p]['w(p)'] = sum(1 for q in state_names if weight_table[p][q] != "")

print("ПОСТРОЕНИЕ ТАБЛИЦЫ ЗАКОНЧИЛ")
return weight_table, state_names

def print_weight_table(self, weight_table, state_names):
    print("\nТаблица с весами состояний:")

    header = ["State"] + state_names + ["w(p)"]
    rows = []

    # Заполнение строк таблицы
    for p in state_names:
        row = [p] + [weight_table[p][q] if weight_table[p][q] != "" else "" for q in state_names] + [
            weight_table[p]['w(p)']]
        rows.append(row)

    print(tabulate(rows, headers=header, tablefmt="grid"))

def hamming_distance(self, code1, code2):
    return sum(c1 != c2 for c1, c2 in zip(code1, code2))

def generate_codes(self, num_bits):
    #Генерация кодов заданной длины
    return [".join(bits) for bits in product('01', repeat=num_bits)]

def find_min_code(self, weight_table, coded_states, state, codes):
    #Нахождение кода S(a) с минимальной суммой
    min_sum = float('inf')
    best_codes = []

    # Генерация кандидатов
    neighbors = set()
    for q in coded_states:
        neighbors.update(self.generate_neighbors(coded_states[q]))

    # Только те, что не использованы
    available_neighbors = [code for code in neighbors if code not in coded_states.values()]

```

```

print(f'Доступные соседи для состояния {state}: {available_neighbors}')

# Исключение кодов, для которых  $w(q, p) = 0$ 
valid_neighbors = []
for code in available_neighbors:
    for q in coded_states:
        if self.hamming_distance(coded_states[q], code) == 1:
            # Если  $w(q, p) = 0$ , исключаем этот код
            if weight_table[q][state] == "":
                print(f'Код {code} исключён, так как  $w(\{q\}, \{state\}) = 0$ ')
                break
        else:
            valid_neighbors.append(code)

print(f'Допустимые соседи после исключения: {valid_neighbors}')

for code in valid_neighbors:
    s = 0
    for q in coded_states:
        if q != 'w(p)' and q in weight_table and state in weight_table[q]: # Исключение 'w(p)'
            if weight_table[q][state] != "":
                # Вес перехода  $w(q, p)$ 
                weight_qp = weight_table[q][state]
                # Расстояние Хэмминга  $d(p(q), \alpha)$ 
                hamming_dist = self.hamming_distance(coded_states[q], code)
                # Вклад в сумму  $S(\alpha)$ 
                s += weight_qp * hamming_dist
            print(
                f'Для состояния {q}:  $w(\{q\}, \{state\}) = \{weight\_qp\}$ ,  $d(\{coded\_states[q]\}, \{code\}) =$ 
                {hamming_dist}, вклад = {weight_qp * hamming_dist}"
            )

    print(f'Сумма  $S(\alpha)$  для кандидата {code}: {s}')
    if s < min_sum:
        min_sum = s
        best_codes = [code]
    elif s == min_sum:
        best_codes.append(code)

# Случайный выбор одного из кодов с минимальной суммой  $S(\alpha)$ 
if best_codes:
    best_code = random.choice(best_codes)
    print(f'Случайно выбран код из кандидатов с минимальной суммой: {best_code}')
    return best_code
else:
    return None

def generate_neighbors(self, code):
    #Генерация кодов с отличием в 1 бит
    neighbors = []
    for i in range(len(code)):
        neighbor = code[:i] + ('1' if code[i] == '0' else '0') + code[i + 1:]
        neighbors.append(neighbor)

    return neighbors

def encode_states(self):

```



```

weight_table, state_names = self.build_weight_table()

states = state_names
num_states = len(states)
num_bits = (num_states - 1).bit_length()
codes = self.generate_codes(num_bits)

coded_states = {}

# Поиск пары с наибольшим весом
max_weight = -1
best_pair = None
for p in weight_table:
    if p != 'w(p)': # Исключение 'w(p)' из обработки
        for q in weight_table[p]:
            if q != 'w(p)' and q != p and weight_table[p][q] != "" and weight_table[p][q] > max_weight:
                max_weight = weight_table[p][q]
                best_pair = (p, q)
if best_pair:
    print(f"Найдена пара с наибольшим весом: {best_pair} с весом {max_weight}")
else:
    print("Не удалось найти пару состояний для кодирования.")
    return coded_states

# Кодирование пары
if best_pair:
    state1, state2 = best_pair # Использование имен состояний из пары с максимальным весом
    coded_states[state1] = codes[0] # Первый код
    coded_states[state2] = codes[1] # Второй код
    print(
        f"Закодированы первые два состояния: {state1} -> {coded_states[state1]}, {state2} ->
{coded_states[state2]}")
else:
    print("Не удалось найти пару состояний для кодирования.")
    return coded_states

# Кодирование остальных состояний
while len(coded_states) < num_states:
    # Поиск пары (q, p), где q закодировано, а p — нет
    candidates = []
    for p in weight_table:
        if p != 'w(p)' and p not in coded_states: # Исключение 'w(p)' из обработки
            for q in weight_table[p]:
                if q != 'w(p)' and q in coded_states and weight_table[p][q] != "":
                    # weight_table[p][q] существует и не пуст
                    weight_pq = weight_table[p][q] if weight_table[p][q] != "" else 0
                    weight_qp = weight_table[q].get(p, 0) if weight_table[q].get(p, "") != "" else 0

                    # Проверка, что ключи 'w(p)' существуют
                    if 'w(p)' in weight_table[p] and 'w(p)' in weight_table[q]:
                        # Формула для вычисления суммы w'
                        w_sum = weight_table[p]['w(p)'] + weight_table[q]['w(p)']
                        candidates.append(
                            (q, p, weight_pq, w_sum)
                        )
                    print(f"Кандидат: (q={q}, p={p}, weight_pq={weight_pq}, w_sum={w_sum})")
    else:

```

```

        print(f"Ошибка: ключ 'w(p)' отсутствует в weight_table для состояния {p} или
{q}")
        continue

    # Если кандидатов нет, прерываем цикл
    if not candidates:
        print("Невозможно закодировать оставшиеся состояния.")
        break

    # Выбираем пару с максимальным весом и суммой w'
    best_candidate = max(candidates, key=lambda x: (x[2], x[3]))
    q, p, weight_pq, w_sum = best_candidate
    print(f"Выбран лучший кандидат: (q={q}, p={p}, weight_pq={weight_pq}, w_sum={w_sum})")

    # Поиск кода для p
    code = self.find_min_code(weight_table, coded_states, p, codes)
    coded_states[p] = code
    print(f"Закодировано состояние {p} -> {code}")

    self.num_bits = num_bits

    return coded_states

# ----- РЕДУКЦИЯ РАССТОЯНИЯ ХЭММИНГА -----
# ----- РЕДУКЦИЯ ВЕСА ХЭММИНГА -----
def extract_variables(self, expression):
    if not expression: # Защита от пустых выражений
        return set()

    try:
        if self.is_jff:
            return set(re.findall(r'[A-Za-z]\d+', expression))
        elif self.is_verilog:
            variables = set()
            matches = re.finditer(r'\b([A-Za-z]\d+)\s*(?:==|!=)', expression)
            for match in matches:
                var = match.group(1)
                if var:
                    variables.add(var)
            return variables
        else:
            print("Предупреждение: формат файла не определён!")
            return set()
    except Exception as e:
        print(f"Ошибка в extract_variables: {e}")
        return set()

def find_successor_pairs(self, multiple_transitions):
    # Поиск пар последовательностей
    successor_pairs = {}

    for state, transitions in multiple_transitions.items():
        # Исключение петлевых переходов
        valid_transitions = [(target, cond) for target, cond in transitions if target != str(state)]

        if len(valid_transitions) < 2:

```

```

        continue # Пропуск состояний с менее чем 2 переходами

    conditions = [self.extract_variables(cond) for _, cond in valid_transitions]

    pairs = []
    for i in range(len(conditions)):
        for j in range(i + 1, len(conditions)):
            if not conditions[i].isdisjoint(conditions[j]): # Проверка пересечения множеств условий
                pairs.append((valid_transitions[i], valid_transitions[j]))
    if pairs:
        successor_pairs[state] = pairs
    return successor_pairs

def pairs_finding(self):
    #Счёт количества повторений пар последователей
    tracemalloc.start()
    multiple_transitions = {state: transitions for state, transitions in self.graph.items() if len(transitions) >
1}
    successor_pairs = self.find_successor_pairs(multiple_transitions)

    # Список для хранения пар с ненулевым количеством повторений
    pairs = set()

    # Сортировка вершин и добавление пары в список, только если количество повторений > 0
    for state in sorted(successor_pairs.keys()):
        for (p1, _), (p2, _) in sorted(successor_pairs[state]):
            pair_count = 0
            for state, transitions in multiple_transitions.items():
                transition_states = {dest for dest, _ in transitions}
                if {p1, p2}.issubset(transition_states):
                    filtered_transitions = [condition for dest, condition in transitions if dest in {p1, p2}]
                    if len(filtered_transitions) == 2:
                        variables_1 = self.extract_variables(filtered_transitions[0])
                        variables_2 = self.extract_variables(filtered_transitions[1])
                        if len(variables_1) == len(variables_2):
                            pair_count += 1

            if pair_count > 0:
                pairs.add(tuple(sorted([p1, p2]))) #Сортировка

    self.pairs = list(pairs)
    pair_counts = {}

    # Подсчет повторений
    for pair in self.pairs:
        pair_count = 0
        for state, transitions in multiple_transitions.items():
            transition_states = {dest for dest, _ in transitions}
            if set(pair).issubset(transition_states):
                filtered_transitions = [condition for dest, condition in transitions if dest in pair]
                if len(filtered_transitions) == 2:
                    variables_1 = self.extract_variables(filtered_transitions[0])
                    variables_2 = self.extract_variables(filtered_transitions[1])
                    if len(variables_1) == len(variables_2):
                        pair_count += 1

        if pair_count > 0:

```

```

        pair_counts[tuple(pair)] = pair_count

table_data = [{"", ".join(pair), count] for pair, count in pair_counts.items()}

if table_data:
    print("\nТаблица повторений пар:")
    print(tabulate(table_data, headers=["Пары", "Повторения"], tablefmt="grid", numalign="center",
                    stralign="center"))
else:
    print("\nНет пар с повторениями.")

self.pair_repeats = pair_counts
self.count_incoming_transitions()

def count_incoming_transitions(self):
    #Счёт количества входящих переходов
    if not self.graph:
        print("Граф пуст, невозможно выполнить анализ.")
        return

    incoming_transitions = {state: 0 for state in self.graph.keys()}

    for transitions in self.graph.values():
        for to_state, _ in transitions:
            if to_state in incoming_transitions:
                incoming_transitions[to_state] += 1
            else:
                incoming_transitions[to_state] = 1

    incoming_transitions = dict(sorted(incoming_transitions.items(), key=lambda x:
self.extract_number(x[0])))

    table_data = [[state, count] for state, count in
                    sorted(incoming_transitions.items(), key=lambda x: self.extract_number(x[0]))]
    print("\nТаблица количества входящих переходов:")
    print(tabulate(table_data, headers=["Вершина", "Число входящих переходов"], tablefmt="grid",
                    stralign="center",
                    numalign="center"))

    self.coding_states(incoming_transitions)

def find_next_code(self, coded_states, bit_width):
    #Поиск наименьшего незанятого кода
    used_codes = {int(code, 2) for code in coded_states.values()}
    candidate = 0
    while candidate in used_codes:
        candidate += 1
    print(f"Выбран код: {bin(candidate)[2:].zfill(bit_width)} (used: {used_codes})")
    return bin(candidate)[2:].zfill(bit_width)

def find_code_with_power_of_two_gap(self, base_code, used_codes, bit_width):
    #Поиск ближайшего кода с разницей в степень двойки
    power = 0
    while True:
        candidate_code = base_code + (2 ** power)
        bin_code = bin(candidate_code)[2:].zfill(bit_width)

```

```

    if candidate_code not in used_codes:
        return bin_code

    power += 1

def coding_states(self, incoming_transitions):
    result_text = ""
    coded_states = {}
    print(incoming_transitions)
    num_states = len(incoming_transitions)
    bit_width = (num_states - 1).bit_length()

    coded_states = {}
    second_in_trans = incoming_transitions.copy()

    # Пока не закодированы все состояния
    while len(coded_states) < len(second_in_trans):
        max_transitions = -1
        max_state = None
        max_repeats = -1
        best_pair = None

        for state, count in second_in_trans.items():
            if state not in coded_states and count >= max_transitions:
                max_transitions = count
                max_state = state

        if max_state is None:
            break

        print(f"Выбрали вершину {max_state}")
        print(coded_states)

        # Проверка, есть ли у max_state пара
        has_pair = any(max_state in (from_state, to_state) for from_state, to_state in self.pairs)

        if not has_pair:
            coded_states[max_state] = self.find_next_code(coded_states, bit_width)
            print(f"Вершина {max_state} не имеет пар, закодирована отдельно -> {coded_states[max_state]})
            continue

        for from_state, to_state in self.pairs:
            if max_state in (from_state, to_state):
                pair = (from_state, to_state)
                repeats = self.pair_repeats.get(pair, 0)

                # Выбор последней подходящей пары
                if repeats > max_repeats:
                    max_repeats = repeats
                    best_pair = pair

        # Если обе вершины еще не закодированы
        if from_state not in coded_states and to_state not in coded_states:
            # Сначала определение, какая вершина должна кодироваться первой
            if max_state == from_state:
                # Присвоение кодов для обеих вершин, начиная с минимального интервала

```

```

        base_code = self.find_next_code(coded_states, bit_width)
        coded_states[from_state] = base_code
        used_codes = {int(code, 2) for code in coded_states.values()}

        # Кодирование второй вершины с интервалом и учетом веса
        coded_states[to_state] = self.find_code_with_power_of_two_gap(
            int(coded_states[from_state], 2),
            used_codes, bit_width)
        print(
            f"Закодировали обе вершины: {from_state} -> {coded_states[from_state]},
{to_state} -> {coded_states[to_state]}")
        break

    elif max_state == to_state:
        base_code = self.find_next_code(coded_states, bit_width)
        coded_states[to_state] = base_code
        used_codes = {int(code, 2) for code in coded_states.values()}

        coded_states[from_state] = self.find_code_with_power_of_two_gap(
            int(coded_states[to_state], 2),
            used_codes, bit_width)
        print(
            f"Закодировали обе вершины: {to_state} -> {coded_states[to_state]}, {from_state} -
> {coded_states[from_state]}")
        break

    # Если одна из вершин закодирована, а другая нет
    elif from_state in coded_states and to_state not in coded_states:
        base_code = int(coded_states[second], 2)
        used_codes = {int(code, 2) for code in coded_states.values()}
        coded_states[first] = self.find_code_with_power_of_two_gap(base_code, used_codes,
bit_width)
        print(f"Закодировали: {to_state} -> {coded_states[to_state]}")

    elif to_state in coded_states and from_state not in coded_states:
        base_code = int(coded_states[to_state], 2)
        used_codes = {int(code, 2) for code in coded_states.values()}
        coded_states[from_state] = self.find_code_with_power_of_two_gap(base_code,
used_codes, bit_width)
        print(f"Закодировали: {from_state} -> {coded_states[from_state]}")

    if len(coded_states) == 0:
        coded_states[max_state] = '0' * bit_width

    print(f"Итоговые закодированные состояния: {coded_states}")

    sorted_states = sorted(coded_states.items()) #Сортировка
    result_text = "Итоговые закодированные состояния:\n" + "\n".join(
        f"{state}: {code}" for state, code in sorted_states)

    current, peak = tracemalloc.get_traced_memory()
    tracemalloc.stop()

    try:
        logical_elements = self.count_logical_elements(coded_states)
        print(f"Количество логических элементов (Код Джонсона): {logical_elements}")

```

```

except Exception as e:
    print(f"Ошибка при подсчете логических элементов: {e}")
    logical_elements = None

print(f"Пиковое использование памяти: {peak / 1024:.2f} КБ")

self.logical_elements = logical_elements
self.coded_states = coded_states
self.num_bits = bit_width
self.show_encoded_states(self.coded_states)
# ----- РЕДУКЦИЯ ВЕСА ХЭММИНГА -----
----
# ----- ГЕНЕТИЧЕСКИЙ АЛГОРИТМ -----
---

def ensure_unique_codes(self, encoding):
    #Проверка уникальности кодов
    tracemalloc.start()

    used_codes = {}
    new_encoding = encoding.copy()

    for state, code in new_encoding.items():
        if code in used_codes.values():
            new_code = self.find_unique_code(code, set(used_codes.values()))
            new_encoding[state] = new_code
            used_codes[state] = new_encoding[state]

    return new_encoding

def generate_encodings(self):
    if not hasattr(self, 'states') or not self.states:
        print("Ошибка: состояния не инициализированы!")
        return [], 0
    if not hasattr(self, 'best_parents') or self.best_parents is None:
        self.best_parents = {state_info['name']: [] for state_id, state_info in self.states.items()}
        self.best_parents["scores"] = []

    # Получение числа итераций
    iterations_text = self.iteration_input.text()
    num_states = len(self.states)
    num_triggers = math.ceil(math.log2(num_states))
    iterations_umolch = (num_triggers/num_states) * 50
    try:
        if iterations_text.strip():
            max_iterations = int(iterations_text)
        else:
            max_iterations = min(iterations_umolch, 10000)
    except ValueError:
        max_iterations = 100 # Значение по умолчанию при ошибке ввода

    #Проверка, что значение в диапазоне
    max_iterations = max(2, min(max_iterations, 10000))
    iteration = 0

    print(f"\nКоличество состояний: {num_states}")
    print(f"Количество триггеров: {num_triggers}")
    num_encodings = random.randint(2, 10)

```

```

print(f"Количество кодировок: {num_encodings}")

encodings = []

for _ in range(num_encodings):
    # Создание случайной кодировки
    encoding = {}
    used_codes = set()

    # Прохождение по всем состояниям в self.states
    for state_id, state_info in self.states.items():
        state_name = state_info['name']
        while True:
            # Генерация кода
            code = ''.join(random.choice(['0', '1']) for _ in range(num_triggers))
            if code not in used_codes:
                used_codes.add(code)
                encoding[state_name] = code
                break
    encodings.append(encoding)

print("\nСгенерированные кодировки:")
for i, encoding in enumerate(encodings):
    print(f"Кодировка {i + 1}:")
    for state, code in encoding.items():
        print(f" {state}: {code}")

# После генерации кодировок поиск двух лучших
best_encodings = self.find_best_encoding(encodings, num_triggers)

# Проверка, что найдено хотя бы две кодировки
if len(best_encodings) < 2:
    raise ValueError("Недостаточно кодировок для кроссинговера и мутации.")

while iteration < max_iterations:
    print(f"\n--- Итерация {iteration + 1} ---")

    # Применение кроссинговера и мутации для каждого состояния
    new_population = []
    for state in best_encodings[0].keys():
        parent1 = best_encodings[0][state]
        parent2 = best_encodings[1][state]
        child1, child2 = self.crossover_and_mutate(parent1, parent2)

        new_population.append({state: child1})
        new_population.append({state: child2})

    # Группировка кодировок по состояниям
    grouped_encodings = {}
    for encoding in new_population:
        for state, code in encoding.items():
            if state not in grouped_encodings:
                grouped_encodings[state] = []
            grouped_encodings[state].append(code)

    # Проверка, что все состояния имеют одинаковое количество кодов
    min_length = min(len(codes) for codes in grouped_encodings.values())

```



```

for state in grouped_encodings:
    grouped_encodings[state] = grouped_encodings[state][:min_length]

# Проверка и исправление дубликатов в каждой кодировке
for i in range(min_length):
    child_encoding = {state: codes[i] for state, codes in grouped_encodings.items()}
    child_encoding = self.ensure_unique_codes(child_encoding)
    for state, code in child_encoding.items():
        grouped_encodings[state][i] = code

print("\nНовая популяция после кроссинговера и мутации:")
for i in range(min_length):
    print(f"Кодировка {i + 1}:")
    for state, codes in grouped_encodings.items():
        print(f" {state}: {codes[i]}")

# Оценка
for i in range(min_length):
    child_encoding = {state: codes[i] for state, codes in grouped_encodings.items()}

    try:
        score = self.count_logical_elements(child_encoding)
        print(f"Оценка (логические элементы): {score}")
    except Exception as e:
        print(f"Ошибка при оценке кодировки {i + 1}: {e}")
        continue

    self.best_parents["scores"].append(score)
    print(self.best_parents)

    try:
        for state, code in child_encoding.items():
            if state in self.best_parents:
                if code is not None:
                    self.best_parents[state].append(code)
                else:
                    print(f"Ошибка: код для состояния {state} равен None")
            else:
                print(f"Ошибка: состояние {state} отсутствует в best_parents")
    except Exception as e:
        print(f"Ошибка при добавлении кодировки {i + 1}: {e}")
        continue

# Если это не первая итерация, проверяем и удаляем кодировки с оценкой больше
# минимальной
if iteration > 0 and self.best_parents["scores"]:
    min_score = min(self.best_parents["scores"])
    indices_to_remove = [idx for idx, s in enumerate(self.best_parents["scores"]) if s > min_score]
    for idx in sorted(indices_to_remove, reverse=True):
        self.best_parents["scores"].pop(idx)

    for state in self.best_parents:
        if state != "scores":
            if len(self.best_parents[state]) > idx:
                self.best_parents[state].pop(idx)
    print(self.best_parents)
iteration += 1

```

```

print("\nСодержимое best_parents после завершения всех итераций:")
print(self.best_parents)

output_text = "Итоговые закодированные состояния:\n"
if not hasattr(self, 'coded_states'):
    self.coded_states = {}
for state, codes in self.best_parents.items():
    if state != "scores":
        if codes:
            output_text += f"{state}: {codes[0]}\n"
            self.coded_states[state] = codes[0]

current, peak = tracemalloc.get_traced_memory()
tracemalloc.stop()

print(f"Пиковое использование памяти для редукции веса Хэмминга: {peak / 1024:.2f} КБ")
self.output_text.setPlainText(output_text)

if hasattr(self, 'coded_states') and self.coded_states:
    try:
        logical_elements = self.count_logical_elements(self.coded_states)
        print(f"Количество логических элементов (Генетический алгоритм): {logical_elements}")
        self.logical_elements = logical_elements

        output_text += (
            f"<p style='color: #FFA500; font-size: 24px; font-family: Courier;'">
            f"Количество логических элементов: '
            f"<span style='color: #FFFFFF;'">{logical_elements}</span></p>"
        )
    except Exception as e:
        print(f"Ошибка при подсчете логических элементов: {e}")

self.output_text.clear()
self.output_text.setHtml(output_text)
self.num_bits = num_triggers

return new_population, num_triggers

def find_best_encoding(self, encodings, num_triggers):
    #Поиск двух лучших кодировок
    print("\nНачало поиска лучших кодировок...")

    if not hasattr(self, 'best_parents'):
        self.best_parents = {}
    if hasattr(self, 'states') and self.states:
        self.best_parents = {state_info['name']: [] for state_id, state_info in self.states.items()}
        self.best_parents["scores"] = []

    scored_encodings = []

    for i, encoding in enumerate(encodings):
        print(f"\nОценка кодировки {i + 1}:")
        score = self.count_logical_elements(encoding)
        scored_encodings.append((score, encoding, i + 1))
        print(f"Кодировка {i + 1} получила оценку (логических элементов): {score}")

    # Сортировка кодировки по оценке (чем меньше, тем лучше)

```

```

scored_encodings.sort(key=lambda x: x[0])
print("\nКодировки отсортированы по количеству логических элементов.")

best_encodings = scored_encodings[:2]
print("\nДве лучшие кодировки:")

try:
    for _, encoding, _ in best_encodings:
        for state, code in encoding.items():
            if code is None:
                print(f"Ошибка: код для состояния {state} равен None")
                continue
            self.best_parents[state].append(code)
except Exception as e:
    print(f"Ошибка при добавлении кодировок: {e}")
    print(f"Состояние: {state}, Код: {code}")
    print(f"best_parents: {self.best_parents}")

if "scores" not in self.best_parents:
    self.best_parents["scores"] = []

for score, _, _ in scored_encodings[:2]:
    self.best_parents["scores"].append(score)

for i, (score, encoding, original_index) in enumerate(best_encodings):
    print(f"Кодировка {original_index}:")
    for state, code in encoding.items():
        print(f"  {state}: {code}")

print(self.best_parents)

return [encoding for (score, encoding, original_index) in best_encodings]

def crossover(self, parent1, parent2):
    #Одноточечный кроссинговер
    # Выбор случайной точки кроссинговера
    crossover_point = random.randint(1, len(parent1) - 1)
    # Создание детей
    child1 = parent1[:crossover_point] + parent2[crossover_point:]
    child2 = parent2[:crossover_point] + parent1[crossover_point:]

    print(f"\nКроссинговер:")
    print(f"  Родитель 1: {parent1}")
    print(f"  Родитель 2: {parent2}")
    print(f"  Точка кроссинговера: {crossover_point}")
    print(f"  Ребенок 1: {child1}")
    print(f"  Ребенок 2: {child2}")

    return child1, child2

def mutate(self, child, mutation_rate=0.1):
    #Мутация с вероятностью
    child = list(child)
    for i in range(len(child)):
        if random.random() < mutation_rate:
            child[i] = '1' if child[i] == '0' else '0'
            print(f"  Мутация: бит {i} изменен на {child[i]}")

```

```

return ".join(child)

def crossover_and_mutate(self, parent1, parent2, mutation_rate=0.1):
    #Применение кроссинговера и мутации, получение детей
    child1, child2 = self.crossover(parent1, parent2)
    child1 = self.mutate(child1, mutation_rate)
    child2 = self.mutate(child2, mutation_rate)

    # Гарант уникальности
    if child1 == child2:
        # Если дети одинаковые, изменение одного бита у второго ребенка
        child2 = self.flip_random_bit(child2)
    return child1, child2

def find_unique_code(self, original_code, used_codes):
    #Поиск уникального кода
    code = original_code
    attempts = 0
    max_attempts = len(original_code) * 2 # Максимальное количество попыток

    while code in used_codes and attempts < max_attempts:
        # Инвертирование случайного бита
        code = self.flip_random_bit(code)
        attempts += 1

    # Если не удалось найти уникальный код случайным образом,
    # перебор всех возможных вариантов
    if code in used_codes:
        for i in range(len(original_code)):
            code_list = list(original_code)
            code_list[i] = '1' if code_list[i] == '0' else '0'
            code = ".join(code_list)
            if code not in used_codes:
                return code

    # Если все варианты с одним измененным битом заняты,
    # изменение двух битов и т.д.
    for num_bits in range(2, len(original_code) + 1):
        from itertools import combinations
        for bits_to_flip in combinations(range(len(original_code)), num_bits):
            code_list = list(original_code)
            for bit in bits_to_flip:
                code_list[bit] = '1' if code_list[bit] == '0' else '0'
            code = ".join(code_list)
            if code not in used_codes:
                return code
    return code

def flip_random_bit(self, code):
    #Инвертирование случайного бита
    if not code:
        return code
    bit_to_flip = random.randint(0, len(code) - 1)
    code_list = list(code)
    code_list[bit_to_flip] = '1' if code_list[bit_to_flip] == '0' else '0'
    return ".join(code_list)

```

```

# ----- ГЕНЕТИЧЕСКИЙ АЛГОРИТМ -----
---
# ----- ПОДСЧЁТ ЛОГИЧЕСКИХ ЭЛЕМЕНТОВ -----
-----

def count_logical_elements(self, coded_states):
    if not coded_states:
        return 0

    # Проверка, что все коды имеют одинаковую длину
    code_lengths = {len(code) for code in coded_states.values()}
    if len(code_lengths) != 1:
        raise ValueError("Все коды состояний должны быть одинаковой длины!")

    num_bits = next(iter(code_lengths))
    if num_bits == 0:
        raise ValueError("Коды состояний не могут быть пустыми!")

    # Проверка, что все коды состоят из '0' и '1'
    for state, code in coded_states.items():
        if not all(c in '01' for c in code):
            raise ValueError(f"Некорректный код состояния {state}: {code}")

    # Подсчет элементов для КС1 (функции возбуждения)
    kc1_elements = self._count_kc1_elements(coded_states)

    # Подсчет элементов для КС2 (выходная функция)
    kc2_elements = self.count_output_elements(coded_states)

    # Общее количество элементов
    total_elements = kc1_elements + kc2_elements
    return total_elements

def minterm_to_numbers(self, minterm):
    # Преобразование строкового минтерма с '-' в список числовых минтермов
    numbers = []
    def expand_minterm(prefix, rest):
        if not rest:
            numbers.append(int(prefix, 2))
            return
        if rest[0] == '-':
            expand_minterm(prefix + '0', rest[1:])
            expand_minterm(prefix + '1', rest[1:])
        else:
            expand_minterm(prefix + rest[0], rest[1:])

    expand_minterm("", minterm)
    return numbers

def _condition_to_minterm(self, condition, state_code):
    # Преобразование условия перехода в минтерм
    print(f"[КС1] Преобразование условия '{condition}' с кодом состояния '{state_code}'")

    if not hasattr(self, 'input_variables'):
        variables = set()
        pattern = r'[A-Z][A-Za-z]*[0-9]*'
        for t in self.transitions:
            cond = t['condition']

```

```

        found_vars = re.findall(pattern, cond)
        variables.update(found_vars)
        self.input_variables = sorted(list(variables))
        print(f"[KC1] Обнаружены входные переменные: {self.input_variables}")

    # Формирование списка всех переменных (входы + биты состояния)
    state_bits = [f'Q{i}' for i in range(len(state_code))]
    all_variables = self.input_variables + state_bits
    print(f"[KC1] Все переменные (входы + биты состояния): {all_variables}")

    # Инициализация значения переменных: '-' для неопределённых
    var_values = {var: '-' for var in all_variables}
    for i, bit in enumerate(state_code):
        var_values[f'Q{i}'] = bit

    # Обработка безусловного перехода
    if condition.strip() == '1':
        bitstring = ".join(var_values[var] for var in all_variables)
        print(f"[KC1] Безусловный переход -> минтерм: {bitstring}")
        return bitstring

    # Парсинг условия
    condition = condition.replace(' ', '')
    print(f"[KC1] Обработка условия: {condition}")

    clauses = re.findall(r'([A-Z][A-Za-z]*[0-9]*[0-9]+)==([0-1])', condition)
    for var, value in clauses:
        if var in self.input_variables:
            var_values[var] = value
            print(f"[KC1] Установка {var} = {value}")
        else:
            print(f"[KC1] Предупреждение: Переменная {var} не найдена в input_variables")

    # Формирование минтерма как строки
    bitstring = ".join(var_values[var] for var in all_variables)
    print(f"[KC1] Итоговый минтерм: {bitstring}")
    return bitstring

def _count_kc1_elements(self, coded_states):
    print("\n[KC1] Начинаем подсчёт логических элементов для функций возбуждения")
    qm = QuineMcCluskey()
    num_bits = len(next(iter(coded_states.values())))
    print(f"[KC1] Количество триггеров (бит состояния): {num_bits}")

    transitions_by_state = defaultdict(list)
    for t in self.transitions:
        transitions_by_state[t['from']].append((t['to'], t['condition']))

    total_elements = 0

    for bit in range(num_bits):
        print(f"\n[KC1] Анализ бита {bit} триггера:")
        minterms = []

        for state in coded_states:
            from_code = coded_states[state]
            transitions = transitions_by_state.get(state, [])

```

```

for to_state, condition in transitions:
    to_code = coded_states.get(to_state)
    if to_code is None:
        raise ValueError(f"Состояние {to_state} отсутствует в coded_states!")
    if len(to_code) != num_bits:
        raise ValueError(
            f"Код состояния {to_state} имеет некорректную длину ({len(to_code)} вместо
{num_bits})")

    if to_code[bit] == '1':
        print(f"[KC1] Переход {state} -> {to_state} (условие: '{condition}') активирует бит
{bit}")
        minterm = self._condition_to_minterm(condition, from_code)
        if minterm is None:
            print(f"[KC1] Минтерм не сформирован для условия: {condition}")
            continue
        minterms.append(minterm)
        print(f"[KC1] Добавлен минтерм: {minterm}")

if not minterms:
    print(f"[KC1] Нет минтермов для бита {bit} - пропускаем")
    continue

# Преобразование строковых минтермов в числовые
numeric_minterms = []
for minterm in minterms:
    numeric_minterms.extend(self.minterm_to_numbers(minterm))
print(f"[KC1] Все числовые минтермы для бита {bit}: {[bin(m) for m in numeric_minterms]}")

# Минимизация числовых минтермов
minimized = qm.simplify(numeric_minterms)
print(f"[KC1] Минимизированные термы: {minimized}")

bit_elements = 0
for term in minimized:
    conj_count = term.count('1') + term.count('0')
    bit_elements += conj_count
    print(f"[KC1] Терм '{term}': {conj_count} элементов (конъюнкция)")

or_count = len(minimized) - 1 if minimized else 0
if or_count > 0:
    print(f"[KC1] Добавляем {or_count} элементов за дизъюнкцию (OR)")
    bit_elements += or_count

print(f"[KC1] Итого элементов для бита {bit}: {bit_elements}")
total_elements += bit_elements

print(f"\n[KC1] ОБЩЕЕ КОЛИЧЕСТВО ЭЛЕМЕНТОВ KC1: {total_elements}")
return int(total_elements)

def count_output_elements(self, coded_states):
    if not (self.is_Moore or self.is_Mealy):
        print("[KC2] Автомат не является ни Мура, ни Мили. Элементов: 0")
        return 0

    elements = 0

```

```

outputs = set()

if self.is_Moore:
    print("[KC2] Автомат Мура: анализируем выходы в состояниях")
    for state_id, state_info in self.states.items():
        output_list = state_info.get('output', [])
        if isinstance(output_list, str):
            output_list = [output_list] # Для обратной совместимости
        for output in output_list:
            if output and output not in outputs:
                outputs.add(output)
            print(f" Состояние {state_id}: выход = '{output}'")
else:
    print("[KC2] Автомат Мили: анализируем выходы в переходах")
    for transition in self.transitions:
        output_list = transition.get('output', [])
        if isinstance(output_list, str):
            output_list = [output_list] # Для обратной совместимости
        for output in output_list:
            if output and output not in outputs:
                outputs.add(output)
            print(f" Переход {transition['from']} -> {transition['to']}: выход = '{output}'")

print("\n[KC2] Подсчёт элементов для каждого выхода:")
for output in outputs:
    if output in ('0', '1'):
        print(f" Выход '{output}': константа, элементов: 0")
        continue

    expr_elements = self._count_elements_in_output(output)
    elements += expr_elements
    print(f" Выход '{output}': элементов = {expr_elements}")

print(f"\n[KC2] Всего элементов в KC2: {elements}")
return elements

def _count_elements_in_output(self, expr):
    # Подсчёт элементов в одном выходном выражении
    if not expr:
        return 0

    expr_clean = expr.replace(" ", "")
    count = 0
    details = []

    # Обработка присваивания
    if '=' in expr_clean:
        _, expr_clean = expr_clean.split('=', 1)

    # Подсчёт операторов
    and_count = expr_clean.count('&')
    or_count = expr_clean.count('|')
    xor_count = expr_clean.count('^')
    not_count = expr_clean.count('==0') + expr_clean.count('~')

    count += and_count * 1 # AND
    count += or_count * 1 # OR

```



```

count += xor_count * 2 # XOR
count += not_count * 1 # NOT

if and_count > 0:
    details.append(f'{and_count} AND')
if or_count > 0:
    details.append(f'{or_count} OR')
if xor_count > 0:
    details.append(f'{xor_count} XOR (2 элемента каждый)')
if not_count > 0:
    details.append(f'{not_count} NOT')

if '(' in expr_clean and ')' in expr_clean:
    count += 1
    details.append("1 элемент за скобки")

if details:
    print(f"  Разбор: '{expr}' -> {' '.join(details)}")
else:
    print(f"  Разбор: '{expr}' -> простой сигнал (0 элементов)")

return count

# ----- ПОДСЧЁТ ЛОГИЧЕСКИХ ЭЛЕМЕНТОВ -----
-----
if __name__ == "__main__":
    import sys

    app = QtWidgets.QApplication(sys.argv)
    Coding = QtWidgets.QMainWindow()
    ui = Ui_Coding()
    ui.setupUi(Coding)
    Coding.show()
    sys.exit(app.exec_())

```