МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение

высшего образования

**«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Кафедра «Вычислительная техника»

Дисциплина «Автоматно-лингвистическое моделирование»

**Пояснительная записка**

**к курсовой работе по теме**

**«Программная реализация микропрограммного автомата с жесткой логикой: частичный микропрограммный автомат»**

Вариант 5

Выполнил:

студент группы ИВТАПбд-21

Кондратьев П. С.

Проверила:

ст. п. Лылова А. В.

Ульяновск, 2018

Содержание

Введение……………………………………………………………………………..….3

1. Техническое задание………………………………………………………………...4

2. Анализ предметной области………………………………………………………...5

3. Анализ исходного описания реализуемого микропрограммного автомата с жесткой логикой ……………………………………………………………………….7

4. Проектирование микропрограммного автомата с жесткой логикой

4.1. Составление схемы алгоритма……………………………………………..9

4.2. Проведение канонического метода структурного синтеза……………...11

5. Разработка формата протокола входных и выходных данных для разрабатываемого интерпретатора микропрограммного автомата с жесткой логикой………....14

6. Разработка интерпретатора микропрограммного автомата с жесткой логикой

6.1. Разработка для способа задания – логическое выражение……………...16

6.2. Разработка для способа задания – троичная матрица….………………..17

7. Разработка рабочей нагрузки и тестирование интерпретатора…………….……18

8. Сравнительный анализ данных полученных протоколов……………………..…22

Заключение………………………………………………………………………….....24

Используемые ресурсы……………………………...………………………………..25

Приложения………………………………………………………………….………..26

**Введение**

Целью дисциплины «Автоматно-лингвистическое моделирование» является изучение базовых понятий и принципов построения формальных грамматик и различных моделей автоматов. Любое цифровое устройство можно рассматривать как устройство, состоящее из двух частей: операционного и управляющего блоков. Операционный блок, например АЛУ, характеризуется совокупностью определённых в нём микроопераций, каждая из которых представляет собой некоторый выполняемый в данном операционном блоке акт передачи или преобразования информации. Часть цифрового вычислительного устройства, предназначенного для выработки последовательности управляющих функциональных сигналов, называется управляющим блоком или управляющим устройством.

Проектирование и реализация микропрограммых автоматов направлена на модернизацию информационных систем. Для повышения эффективности ЭВМ необходимо использование программных реализаций наиболее производительных и простых, создание которых невозможно без использования методов проектирования.

Целью работы является изучение методов проектирования и программной реализации микропрограммного автомата с жесткой логикой на основе исходных данных – частичный микропрограммный автомат.

**1. Техническое задание**

Требуется проанализировать исходные данные, на основе которых необходимо создать программную реализацию микропрограммного автомата с жесткой логикой (частичный микропрограммный автомат) на языке программирования высокого уровня С++.

На этапе проектирования необходимо провести анализ схемы алгоритма, на основе которого строится отмеченная ГСА по типу автомата Мили. Далее требуется построить граф на основе полученной отмеченной ГСА. На следующем этапе требуется провести канонический метод структурного синтеза, заключающийся в кодировании входных и выходных сигналов, а также состояний, построении семиколоночной таблицы и логических функций переходов и выходов на её основе.

* Состояния автомата должны иметь имена в формате Si, где i = 0..n (S0 – начальное и конечное состояния автомата);
* Условия в автомате (условные вершины в графе, входные сигналы автомата) кодируются как Xj, где j = 0..m;
* Действия в автомате (операторные вершины в графе, выходные сигналы автомата) кодируются как Yl, где l = 0..k;
* Функция возбуждения (функции переходов) должны иметь имена формата wz (переход с 0 на 1) и uz (переход с 1 на 0), где z =0..v.

Реализованный интерпретатор микропрограммного автомата с жесткой логикой должен иметь строго заданный формат протокола входных и выходных данных. Программа должна выдавать ответ на любые входные данные: результат, если они корректны, сообщение об ошибке, если нет. Программная реализация должна иметь методы для двух способов задания: «логическое выражение» и «граф-схема». После вывода результата программа должна выводить информацию о времени, затраченном на выполнение в зависимости от выбора способа задания.

**2. Анализ предметной области**

Предметной областью является проектирование и реализация микропрограммного автомата с жесткой логикой. Тип микропрограммного автомата определяет название всего управляющего устройства. В микропрограммном автомате с жесткой логикой выходные сигналы управления реализуются за счет однажды соединенных логических схем.[[1]](#footnote-1)

Управляющие автоматы с «жесткой» логикой представляют собой логические схемы, вырабатывающие распределенные во времени управляющие функциональные сигналы. В отличие от управляющих устройств с хранимой в памяти логикой у этих автоматов можно изменить логику работы только путем пере­делок схем автомата. Построение управляющих автоматов с «жесткой» логикой формализуется на основе интерпретации микропрограмм авто­матами.

Работу операционного блока можно описать микропро­граммой, например, на языке микроопераций или в виде графа. По микропрограмме строится соответствующий управляющий автомат типа Мура или Мили.[[2]](#footnote-2) Причем для выходных данных в автомате Мили (Рис.2.1) необходимы входные данные, в то время как в автомате Мура (Рис.2.2) выходные данные есть всегда.

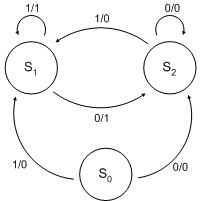
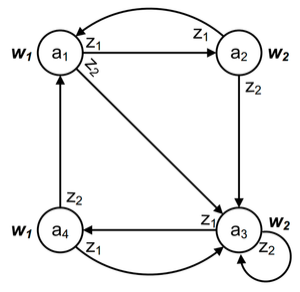
 

Рис.2.1. Граф автомата Мили Рис.2.2. Граф автомата Мура

Абстрактный синтез управляющего автомата начинается с отметки внутренних состояний кодированной ГСА. Отметка состояний должна соответствовать закону функционирования автомата Мура или Мили, то есть выполняется для них различным образом.[[3]](#footnote-3)

Для програмной реализации используется два способа задания: логическое выражение и граф-схема. Логическое выражение в программировании — конструкция [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F), результатом вычисления которой является «истина» или «ложь». Граф-схема алгоритма (ГСА) — конечный связный ориентированный граф. , вершины которого. соответствуют операторам, а дуги. задают порядок следования вершин (операторов) алгоритма.

**3. Анализ исходного описания реализуемого**

**микропрограммного автомата с жесткой логикой**

Для программной реализации дана уже готовая блок-схема микропрограммного автомата согласно данному варианту (частичный микропрограммный автомат). Ниже представлен рисунок данной блок - схемы (см. Рис. 3.1.).

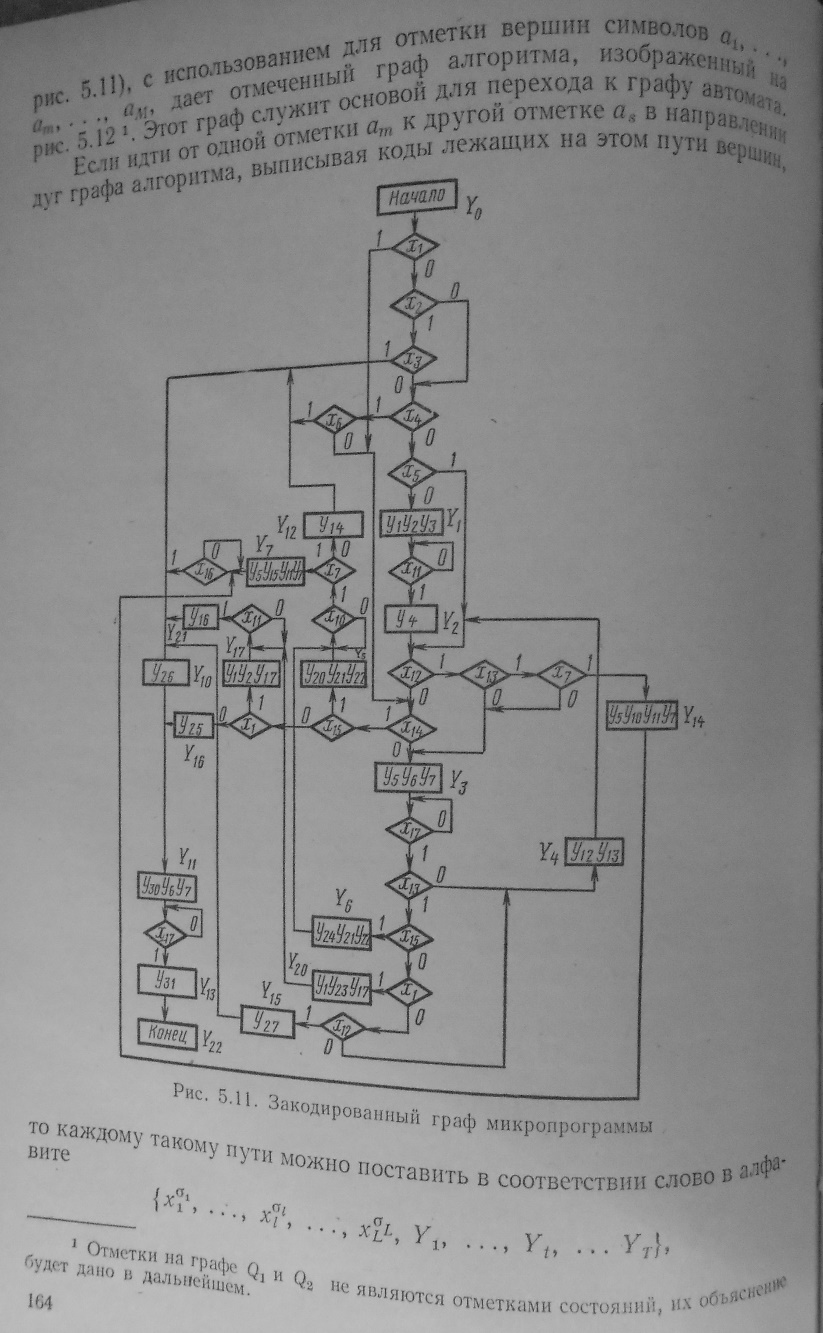


Рис.3.1. частичный микропрограммный автомат

**Определение закона функционирования частичного микропрограммного автомата.**

Будем называть микропрограммным автоматом конечный автомат Мили или Мура, синтезированный по графу алгоритма. Построение графа автомата Мили производиться в два этапа:

1. Получение отмеченного графа алгоритма
2. Построение графа автомата

Обычно при проектировании цифровых устройств предварительно составляется содержательный граф алгоритма, в котором внутри условных и операторных вершин записаны не элементы множества X и Y, а логические условия и микрооперации в содержательных терминах. После построения содержательного графа алгоритма логические условия и микрооперации кодируются символами , …, ,…, и ,…, соответственно. Кроме того, различным микрокомандам присваивается различные символы ,…,.

Пример закодированного графа алгоритма изображен на Рис.3.1.

Условимся в дальнейшем операторные вершины графа алгоритма обозначить символами микрокоманд, а условные вершины – символами логических условий. Если в проектируемом устройстве в каждый момент времени выполняется только одна операция из заданного множества операций, то с целью экономии оборудования выполнение всего множества операций возлагается на одну схему. Схема операционной части формируется путем объединения операционных схем, разработанных для отдельных операций.

При записи микроопераций, принадлежащих объединенному набору, разряды устройств нумеруются согласно скелету операционной части. Аналогично определяется объединенный набор информационных сигналов. В соответствии с правилами языка операционных схем по наборам микроопераций и информационных сигналов определяются шины, связывающие между собой операционные устройства.

**4.Проектирование микропрограммного автомата с жесткой логикой**

**4.1. Составление схемы алгоритма**

В исходном описании автомата количество использующихся значений входных (условий) и выходных состояний не соответствует действительному. В схеме пропущены некоторые входные и выходные значения. Поэтому было принято решение о перекодировании значений X-ов и Y-ов.

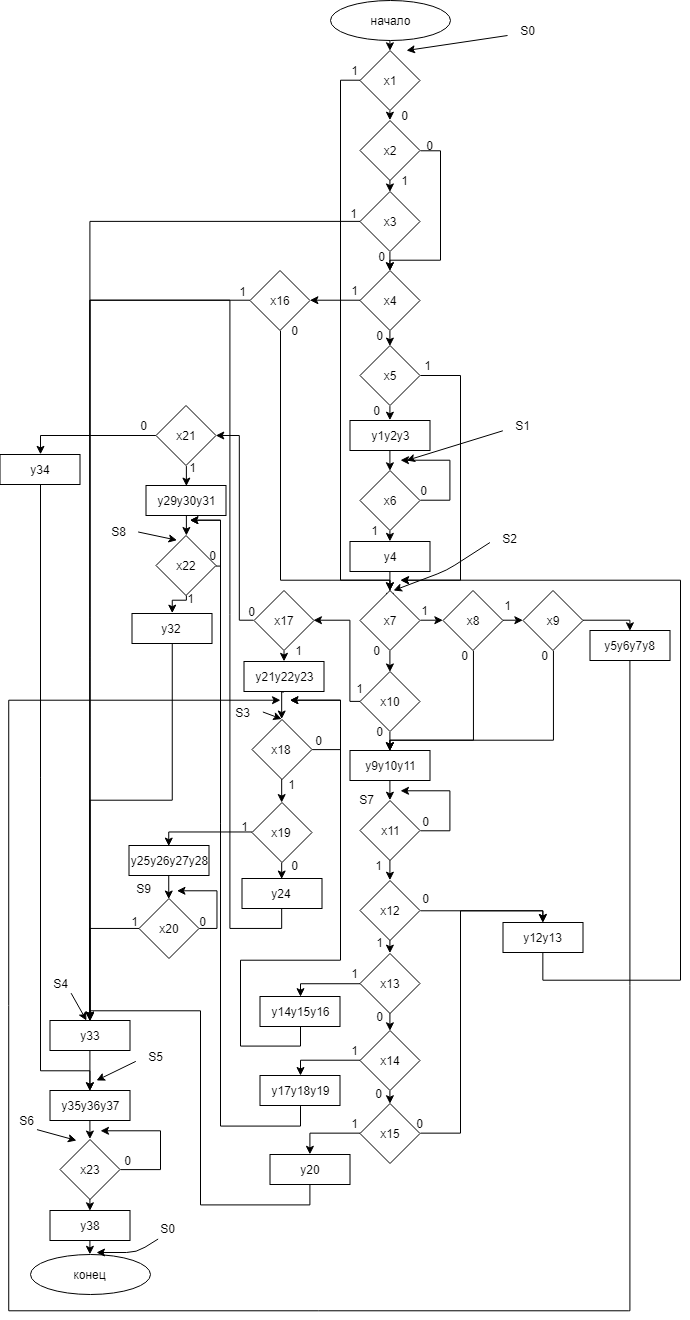


Рис.4.1.1. Схема алгоритма

Построена отмеченная ГСА заданного микропрограммного автомата Мили. S0 – вход вершины, следующей за объявлением начала и вход вершины конца микропрограммы. Входы всех вершин, следующих за операторными, помечаются символом S со своим индексом.

Граф переходов (Рис.4.1.2) на основе полученной отмеченной ГСА имеет столько вершин, сколько различных S имеет отмеченная ГСА. Между Si и Sj есть дуга, если в графе есть между ними путь. Над дугой ставится входной сигнал и выходной сигнал, который соответствует выполняемой микрооперации. Если в ГСА есть безусловный переход, то этой дуге приписывается входной сигнал 1.

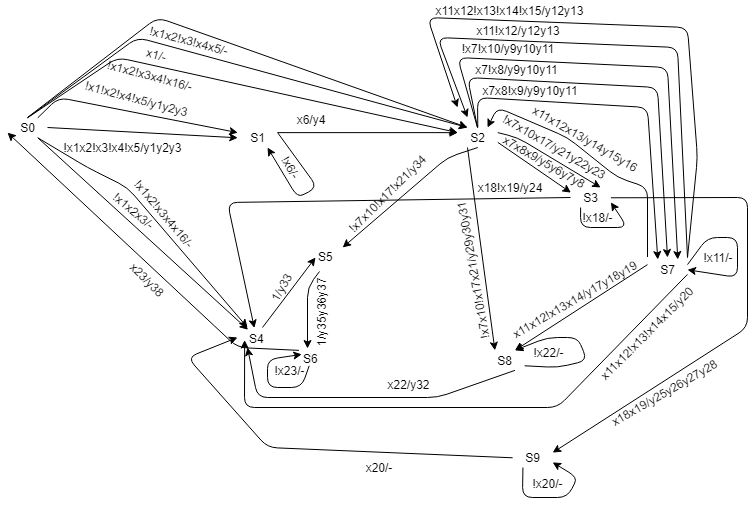


Рис.4.1.2. Граф переходов на основе полученной отмеченной ГСА

**4.2. Проведение канонического метода структурного синтеза**

На первом этапе канонического метода структурного синтеза необходимо произвести кодирование входных и выходных сигналов, а также состояний (S). Так как после построение схемы получилось 10 состояний (S0…S9), то для кодирования потребуется 4 бита (α1 α2 α3 α4), это вытекает из формулы:

На втором этапе строится семиколоночная таблица, на основе графа автомата, (Таблица 4.2.1), в которой указываются все возможные переходы между состояниями при различных входных сигналах, а также выходные сигналы при переходе и функции возбуждения wz – переход из 0 в 1, uz – переход из 1 в 0.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| нач.сос | код н.с. α1α2α3α4 | код сос. | код к.с. α1α2α3α4 | вх. с. | вых. с. | ф. Перехода (W: 0>1, U: 1>0) |
| S0 | 0000 | S1 | 0001 | !x1x2!x3!x4!x5 | y1y2y3 | W4 |
| S0 | 0000 | S1 | 0001 | !x1!x2!x4!x5 | y1y2y3 | W4 |
| S0 | 0000 | S2 | 0001 | !x1x2!x3x4!x16 | - | W4 |
| S0 | 0000 | S2 | 0010 | x1 | - | W3 |
| S0 | 0000 | S2 | 0010 | !x1!x2x4!x16 | - | W3 |
| S0 | 0000 | S2 | 0010 | !x1x2!x3!x4x5 | - | W3 |
| S0 | 0000 | S4 | 0100 | !x1x2!x3x4x16 | - | W2 |
| S0 | 0000 | S4 | 0100 | !x1!x2x4x16 | - | W2 |
| S0 | 0000 | S4 | 0100 | !x1x2x3 | - | W2 |
| S1 | 0001 | S1 | 0001 | !x6 | - | - |
| S1 | 0001 | S2 | 0010 | x6 | y4 | W3U4 |
| S2 | 0010 | S3 | 0011 | x7x8x9 | y5y6y7y8 | W4 |
| S2 | 0010 | S3 | 0011 | !x7x10x17 | y21y22y23 | W4 |
| S2 | 0010 | S5 | 0101 | !x7x10!x17!x21 | y34 | W2W4U3 |
| S2 | 0010 | S7 | 0111 | x7x8!x9 | y9y10y11 | W2W4 |
| S2 | 0010 | S7 | 0111 | x7!x8 | y9y10y11 | W2W4 |
| S2 | 0010 | S7 | 0111 | !x7!x10 | y9y10y11 | W2W4 |
| S2 | 0010 | S8 | 1000 | !x7x10!x17x21 | y29y30y31 | W1U3 |
| S3 | 0011 | S3 | 0011 | !x18 | - | - |
| S3 | 0011 | S4 | 0100 | x18!x19 | y24 | W2U3U4 |
| S3 | 0011 | S9 | 1001 | x18x19 | y25y26y27y28 | W1U3 |
| S4 | 0100 | S5 | 0101 | 1 | y33 | W4 |
| S5 | 0101 | S6 | 0110 | 1 | y35y36y37 | W3U4 |
| S6 | 0110 | S6 | 0110 | !x23 | - | - |
| S6 | 0110 | S0 | 0000 | x23 | y38 | U2U3 |
| S7 | 0111 | S7 | 0111 | !x11 | - | - |
| S7 | 0111 | S2 | 0010 | x11!x12 | y12y13 | U2U4 |
| S7 | 0111 | S2 | 0010 | x11x12!x13!x14!x15 | y12y13 | U2U4 |
| S7 | 0111 | S4 | 0100 | x11x12!x13!x14x15 | y20 | U3U4 |
| S7 | 0111 | S8 | 1000 | x11x12!x13x14 | y17y18y19 | W1U2U3U4 |
| нач.сос | код н.с. α1α2α3α4 | код сос. | код к.с. α1α2α3α4 | вх. с. | вых. с. | ф. Перехода (W: 0>1, U: 1>0) |
| S7 | 0111 | S3 | 0011 | x11x12x13 | y14y15y16 | U2 |
| S8 | 1000 | S8 | 1000 | !x22 | - | - |
| S8 | 1000 | S4 | 0100 | x22 | y32 | W2U1 |
| S9 | 1001 | S9 | 1001 | !x20 | - | - |
| S9 | 1001 | S4 | 0100 | x20 | - | W2U1U4 |

Таблица 4.2.1. Семиколоночная таблица

На третьем этапе строятся логические функции (выражения) переходов выходов. Для построения функций выходов необходимо учитывать тип автомата.

Для автомата Мили учитывают код начального состояния и входной сигнал при построении функции выхода (в отличии от автомата Мура).

Функции возбуждения зависят от начального состояния и входного сигнала всегда, независимо от типа автомата.

Дальнейшее кодирование и преобразование не требуется, так как в варианте используются логические выражение (функции выходов и переходов) и граф-схема (семиколоночная таблица с состояниями), для их реализации этого будет достаточно.

**5. Разработка формата протокола входных и выходных данных**

**для разрабатываемого интерпретатора**

**микропрограммного автомата с жесткой логикой**

Протокол входных и выходных данных необходим для корректной работы программы и оценки результатов её работы. Формат входных данных четко регламентируется, что позволяет избежать проблем с обработкой входных данных, а также структурировать их проверку.

На вход интерпретатора микропрограммного автомата с жесткой логикой подается строка, состоящая строго из 23 символов (слева на права, в файле input.txt в папке с проектом). Эти символы являются набором условий переходов Xi, которые принимают значения 0 или 1.

После ввода корректной строки с набором условий переходов необходимо выбрать один из способов задания: логическое выражение или граф-схема. Выбор происходит посредством ввода кода способа задания 1 или 2, если код введен некорректно, выводится сообщение об ошибке и происходит прерывание работы программы. Из этого следует, что данные о наборе кодов условий не зависят от способа задания и всегда вводятся единой строкой.

Выходные данные представляют собой пошаговое отображение работы программы.

1. Проверка входных данных на соответствие шаблону – 23-разрядная строка. В случае ввода строки меньшего или большего размера выводится сообщение об ошибке, работа программы завершается.

2. Проверка набора кодов условий переходов на наличие символов отличных от 0 и 1. В случае ввода другого символа, выводится сообщение о некорректном вводе. Работа программы завершается.

3. Выводится сообщение с кодами способов задания, после ввода которого идет проверка на соответствие. В случае ввода некорректного кода, выводится сообщение об ошибке, работа программы завершается.

4. После выбора способа задания, выводятся состояния программы и выходные данные при каждом переходе.

5. По завершению работы программы, выводится время, которое было затрачено программой на выполнение алгоритма в зависимости от способа задания.

Входные данные задаются посредством консольного ввода (выбор способа задания), выходные данные также отображаются в консоли.

**6. Разработка интерпретатора микропрограммного автомата с жесткой логикой**

Работа программы начинается до ввода входных данных – объявляются переменные, задается кодировка для вывода сообщений на русском языке. Далее происходит считывание строки из консоли и проверка её на корректность. Если входная строка соответствует шаблону: состоит из 23 символов, которые принимают значения 0 или 1, - на консоль выводится сообщение с кодами возможных способов задания на выбор. В противном случае выводится сообщение о вводе некорректных данных, работа программы прерывается. Введенный код способа задания также проходит проверку на соответствие, и, в случае ввода несуществующего кода, выводится сообщение об ошибке ввода.

**6.1. Разработка для способа задания – логическое выражение**

При выборе способа задания «логическое выражение» происходит вызов функции logic(). Функция использует булевские массивы для хранения значений функций выходов (y[]) и возбуждения: для перевода с 0 в 1 - w[], а с 1 в 0 - u[]. Код текущего состояния и коды условий переходов хранятся в массивах state[] и x[], соответственно. Сначала в функции определяются логические выражения – функции выходов и возбуждения. Результатом вычисления логических выражений является булевское значение true или false. Обновление состояния функций выходов и возбуждения происходит после каждого перехода между состояниями.

После расчета логических выражений выводится код текущего состояния, записанный в массиве state[]. На основе результатов вычисления функций возбуждения происходит изменение в массиве, хранящем код текущего состояния. При истинном значении ячейки в массивах w[] и u[] в массиве state[] происходит замена в соответствующей позиции с 0 на 1 или с 1 на 0. После обновления состояния, оно выводится в консоль. Далее следует вывод массива y[], хранящего информацию о функциях вывода, где 1 в позиции y[i] указывает на вывод Yi при совершении перехода между состояниями.

В случае попадания из состояния Si в состояние S0, программа завершает работу. Также происходит отслеживание циклов. Для этого создана переменная cycle, хранящий информацию о количестве проходов через вершину. Если вершина уже была посещена, происходит прерывание работы программы и выводится сообщение о зацикливании. По завершении выводится время работы программы.

Реализация данного способа задания составила около 100 строк кода. Основная сложность реализации заключается в том, что необходимо прописать все логические выражения для функций выходов и возбуждения, которые без минимизации довольно громоздки. Наибольшие затраты времени выполнения приходятся на обновление и повторный расчет всех функций выхода и возбуждения, заданных логическими выражениями, после каждого перехода между состояниями. Также следует учитывать, что консольный вывод также требует времени.

**6.2. Разработка для способа задания – граф-схема**

При выборе способа задания «граф-схема» происходит вызов функции graf (). Сначала в функции задается первоначальное состояние.

Далее происходит передвижение по семиколоночной таблице, с помощью бесконечного цикла с проверкой текущего состояния и набора X для перехода в другое состояние - и возвращающее выходные сигналы (X) после чего обновляется состояние.

В случае попадания из состояния Si в состояние S0, программа завершает работу. Также происходит отслеживание циклов. Для этого создана переменная cycle, хранящий информацию о количестве проходов через вершину в соответствующей её номеру ячейке. Если вершина уже была посещена, происходит прерывание работы программы и выводится сообщение о зацикливании. По завершении выводится время работы программы.

Реализация данного способа задания составила около 200 строк кода. При реализации способа задания «граф-схема».

**7. Разработка рабочей нагрузки и тестирование интерпретатора**

Разработка рабочей нагрузки (Таблица 7.1) необходима для проверки работы программы при всех возможных вариантах. Рабочая нагрузка представляет собой строку, подающуюся на вход программы. Рабочая нагрузка для способа задания «логическое выражение» (Таблица 7.2) идентична рабочей нагрузке для способа задания «троичная матрица» (Таблица 7.3), но из-за различия результатов, выходных данных, необходима демонстрация всех вариантов.

Таблица 7.1. Рабочая нагрузка

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 111111111111111111111117 | Error:Входные данные должны содержать 23 позиции! |
| 71111111111111111111111 | Error:Введенная строка должна содержать 0 или 1! |
| 3 | Выбор алгоритма (Выбирите числовой пункт):  1 Логическое выражение  2 Граф-схема  Ваш выбор: 3  Выбран неправильный пункт! |

Таблица 7.2. Рабочая нагрузка для способа задания «логическое выражение»

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 11111111111111111111111 | 0000 -> 0010 : 00000000000000000000000000000000000000  0010 -> 0011 y5 y6 y7 y8 : 00001111000000000000000000000000000000  0011 -> 1001 y25 y26 y27 y28 : 00000000000000000000000011110000000000  1001 -> 0100 : 00000000000000000000000000000000000000  0100 -> 0101 y33 : 00000000000000000000000000000000100000  0101 -> 0110 y35 y36 y37 : 00000000000000000000000000000000001110  0110 -> 0000 y38 : 00000000000000000000000000000000000001  Время выполнения: 28 мс |
| 00000000000000000000000 | 0000 -> 0001 y1 y2 y3 : 11100000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  Зацикливание...  Время выполнения: 54 мс |

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 01001000011100110001011 | 0000 -> 0010 : 00000000000000000000000000000000000000  0010 -> 0101 y29 y30 y31 y34 : 00000000000000000000000000001110010000  0101 -> 0110 y35 y36 y37 : 00000000000000000000000000000000001110  0110 -> 0000 y20 y38 : 00000000000000000001000000000000000001  Время выполнения: 15 мс |
| 01100010000000000000001 | 0000 -> 0100 : 00000000000000000000000000000000000000  0100 -> 0101 y33 : 00000000000000000000000000000000100000  0101 -> 0110 y35 y36 y37 : 00000000000000000000000000000000001110  0110 -> 0000 y38 : 00000000000000000000000000000000000001  Время выполнения: 28 мс |
| 00000100000000001011010 | 0000 -> 0001 y1 y2 y3 : 11100000000000000000000000000000000000  0001 -> 0010 y4 : 00010000000000000000000000000000000000  0010 -> 0111 : 00000000000000000000000000000000000000  0111 -> 0111 : 00000000000000000000000000000000000000  0111 -> 0111 : 00000000000000000000000000000000000000  0111 -> 0111 : 00000000000000000000000000000000000000  0111 -> 0111 : 00000000000000000000000000000000000000  0111 -> 0111 : 00000000000000000000000000000000000000  0111 -> 0111 : 00000000000000000000000000000000000000  Зацикливание...  Время выполнения: 27 мс |
| 00000100001100111111111 | 0000 -> 0001 y1 y2 y3 : 11100000000000000000000000000000000000  0001 -> 0010 y4 : 00010000000000000000000000000000000000  0010 -> 0111 : 00000000000000000000000000000000000000  0111 -> 0101 : 00000000000000000000000000000000000000  0101 -> 0110 y35 y36 y37 : 00000000000000000000000000000000001110  0110 -> 0000 y20 y38 : 00000000000000000001000000000000000001  Время выполнения: 70 мс |

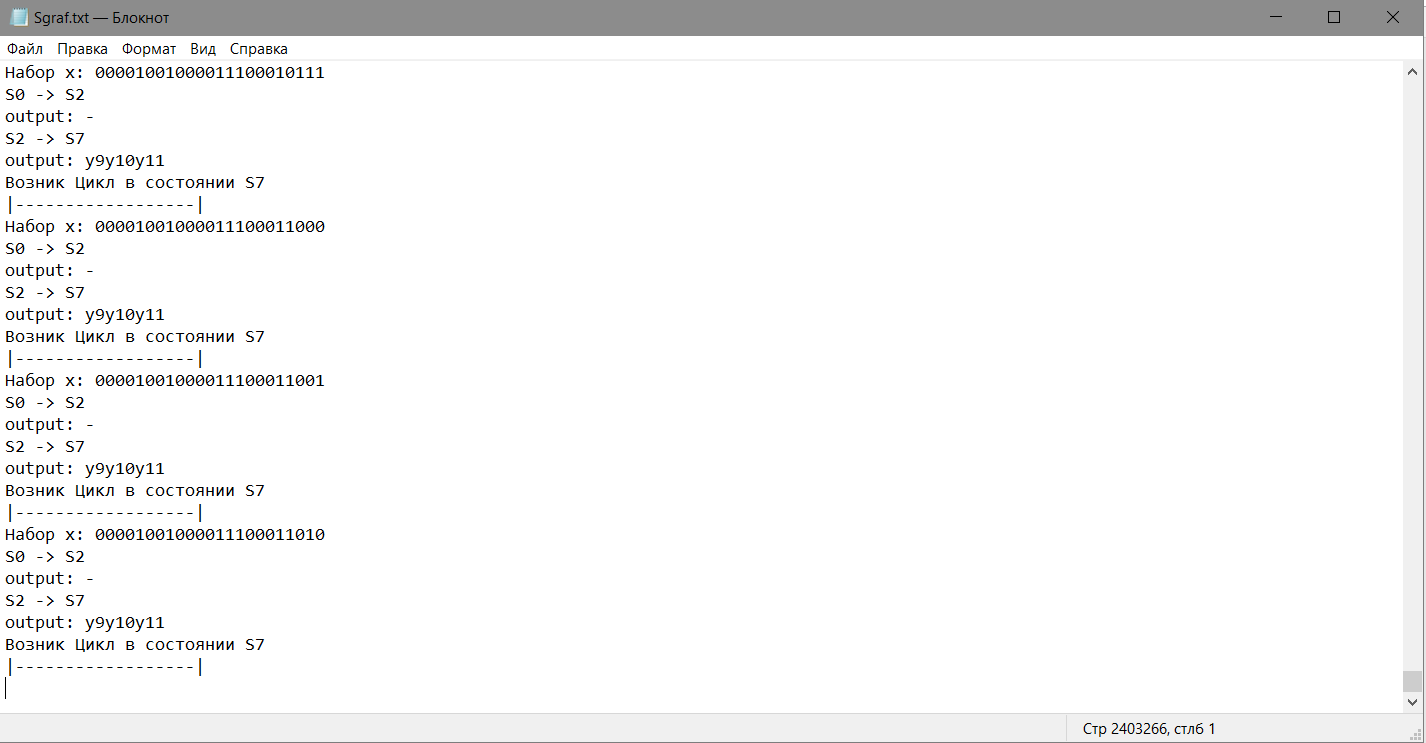
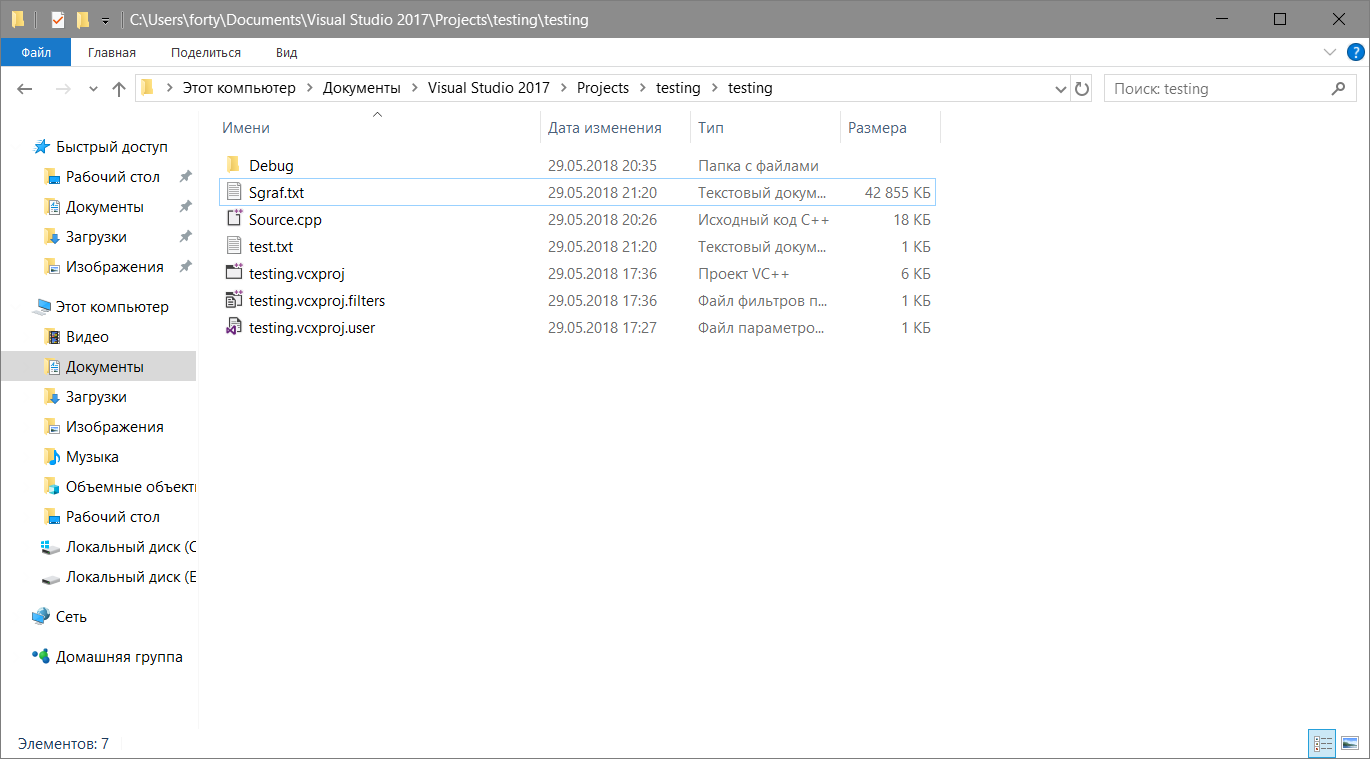
Таблица 7.3. Рабочая нагрузка для способа задания «граф-схема»

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 11111111111111111111111 | S0 -> S2  output: -  S2 -> S3  output: y5y6y7y8  S3 -> S9  output: y25y26y27y28  S9 -> S4  output: -  S4 -> S5  output: y33 |
| Входные данные | Выходные данные |
| 11111111111111111111111 | S5 -> S6  output: y35y36y37  S6 -> S0  output: y38  Время выполнения: 13 мс |

|  |  |
| --- | --- |
| Входные данные | Выходные данные |
| 00000000000000000000000 | S0 -> S1  output:y1 y2 y3  S1 -> S1  output: -  S1 -> S1  output: -  Возник Цикл в состоянии S1  Время выполнения: 5 мс |
| 01001000011100110001011 | S0 -> S2  output: -  S2 -> S5  output: y34  S5 -> S6  output: y35y36y37  S6 -> S0  output: y38  Время выполнения: 4 мс |
| 01100010000000000000001 | S0 -> S4  output: -  S4 -> S5  output: y33  S5 -> S6  output: y35y36y37  S6 -> S0  output: y38  Время выполнения: 4 мс |
| 00000100000000001011010 | S0 -> S1  output:y1 y2 y3  S1 -> S2  output: y4  S2 -> S7  output: y9y10y11  S7 -> S7  output: -  S7 -> S7  output: -  Возник Цикл в состоянии S7  Время выполнения: 7 мс |

Так же было произведено тестирование метода граф-схемы. Был создан метод testing (), в котором генерируются все переборы X (test.txt), после чего вызывается метод graf (). После выполнения метода все выходные данные записываются в файл Sgraf.txt, после метод graf (), берет следующую комбинацию X и проделывает тоже самое, пока не закончатся переборы.

Всего переборов это нерентабельно проверять все наборы X, для проверки выходных данных. Это показывает, что все проверки, переборов X, охватить сложно, из-за большого их количества.



**8. Сравнительный анализ данных полученных протоколов**

Сравнение результатов тестирования реализации двух способов задания: «логическое выражение» и «граф-схема» - осуществляется на основе ранее полученных результатов тестирования рабочей нагрузки.

Для получения результатов тестирования (Таблица 8.1) двух протоколов выходных данных использовались одинаковые входные данные. Время работы двух интерпретатора при различных способах задания крайне разнятся. Большое преимущество имеет способ задания «граф-схема», это особо заметно при вводе рабочей нагрузке.

Таблица 8.1. Сравнительный анализ протоколов выходных данных

|  |  |
| --- | --- |
| Логическое выражение | граф-схема |
| 0000 -> 0010 : 00000000000000000000000000000000000000  0010 -> 0101 y29 y30 y31 y34 : 00000000000000000000000000001110010000  0101 -> 0110 y35 y36 y37 : 00000000000000000000000000000000001110  0110 -> 0000 y20 y38 : 00000000000000000001000000000000000001  Время выполнения: 15 мс | S0 -> S2  output: -  S2 -> S5  output: y34  S5 -> S6  output: y35y36y37  S6 -> S0  output: y38  Время выполнения: 4 мс |
| 0000 -> 0001 y1 y2 y3 : 11100000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  0001 -> 0001 :  …………………………………………………  00000000000000000000000000000000000000  0001 -> 0001 : 00000000000000000000000000000000000000  Зацикливание...  Время выполнения: 54 мс | S0 -> S1  output:y1 y2 y3  S1 -> S2  output: y4  S2 -> S7  output: y9y10y11  S7 -> S7  output: -  S7 -> S7  output: -  Возник Цикл в состоянии S7  Время выполнения: 7 мс |
| 0000 -> 0001 y1 y2 y3 : 11100000000000000000000000000000000000  0001 -> 0010 y4 : 00010000000000000000000000000000000000  0111 -> 0101 : 00000000000000000000000000000000000000  0101 -> 0110 y35 y36 y37 : 00000000000000000000000000000000001110  0110 -> 0000 y20 y38 : 00000000000000000001000000000000000001  Время выполнения: 70 мс | S0 -> S2 output: -  S2 -> S3 output: y5y6y7y8  S3 -> S9 output: y25y26y27y28  S9 -> S4 output: -  S4 -> S5 output: y33  S5 -> S6 output: y35y36y37  S6 -> S0 output: y38  Время выполнения: 13 мс |

Для сравнения двух способов задания следует учитывать также сложность реализации. Для реализации способа задания «логическое выражение» потребовалось написать все пересчеты Y, W, U, что очень влияет на скорость работы данной реализации. Среднее время логических выражений примерно 46 мс, во много раз больше нежели реализация с помощью граф-схемы (8 мс), в которой всего лишь приходиться проверять текущее состояние с входными сигналами и после изменять текущее состояние. К тому же в логических выражениях на каждом прохождении через состояния приходиться пересчитывать выходные сигналы и функций переходов.

**Заключение**

В ходе выполнения курсовой работы были изучены основные аспекты проектирования и создания программной реализации частичного микропрограммного автомата.

На основе исходных данных был проведен анализ и проектирование алгоритма, построена отмеченная ГСА и граф автомата Мили. Также был проведен канонический метод структурного синтеза, состоящий из следующих этапов:

* Кодирование входных и выходных сигналов, а также состояний;
* Построение семиколоночной таблицы;
* Построение логических функций переходов и выходов;

Была создана программная реализация частичного микропрограммного автомата для двух способов задания: «логическое выражение» и «граф-схема» - на языке программирования С++.

Для тестирования программы были созданы протоколы входных и выходных данных, а также рабочая нагрузка, позволяющая проверить работу программу при всех возможных вариантах переходов. Также тестирование показывает работу программы при возможных ошибках, как например, ввод некорректных данных или зацикливании.

Был проведен сравнительный анализ программных реализаций двух способов задания: «логическое выражение» и «граф-схема» – на основе результатов тестирования, с учетом времени выполнения программы, и сложности реализации методов, с учетом количества написанных строк кода.

**Используемые ресурсы**

1. <https://studfiles.net/preview/5514261/page:13/> - Булевые функции (Дата обращения: 28.05.2018)
2. С.А. Майоров, Г.И. Новиков, О.Ф. Немолочнов, С.И. Баранов, П.А. Шипилов, В.И. Скорубский, Г.А. Петухов, Б.Д. Тимченок Проектирование цифровых вычислительных машин. Под. ред. С.А. Майорова. Учебное пособие для студентов вузов. М., «Высш. школа», 1972. - 344 с.
3. https://ru.wikipedia.org/wiki/- Процессор: микропрограммный автомат с жесткой логикой (Дата обращения: 28.05.2018)
4. И.В. Жукалина Теория автоматов: методические указания к курсовому проекту для специальности 230101 / И.В. Жукалина. - Оренбург: ГОУ ОГУ, 2008. – 29 с.
5. <https://studfiles.net/preview/5179828/> - Управляющие автоматы с жесткой логикой (Дата обращения: 28.05.2018)

**Приложения**

Исходный код:

#include <iostream>

#include <fstream>

#include <string>

#include <bitset>

#include <ctime>

using namespace std;

ifstream inFile;

//ofstream test, Sgraf, Slogic;

bool state[4];

string inputx, error;

int x[23], s[10];

bool y[38];

bool w[4];

bool u[4];

int cycle = 0;

void reader() {

/\*

inFile.open("test.txt");

getline(inFile, inputx);

\*/

inFile.open("input.txt");

getline(inFile, inputx);

if (inputx.size() > 23 || inputx.size() < 23)

error += "Error:Входные данные должны содержать 23 позиции!\n";

for (int i = 0; i <= inputx.size() - 1; i++) {

if (inputx[i] != '0' && inputx[i] != '1') {

error += "Error:Введенная строка должна содержать 0 или 1!\n";

break;

}

}

if (!error.empty()) {

cout << error;

system("pause");

exit(0);

}

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

x[i] = inputx[i] - '0';

}

inFile.close();

}

void graf() {

state[0] = 0; state[1] = 0; state[2] = 0; state[3] = 0;

while (1) {

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 0 && !x[0] && x[1] && !x[2] && !x[3] && !x[4]) {

cout << "S0 -> S1\noutput: y1 y2 y3\n";

//Sgraf << "S0 -> S1\noutput: y1 y2 y3\n";

state[0] = 0; state[1] = 0; state[2] = 0; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 0 && !x[0] && !x[1] && !x[3] && !x[4]) {

cout << "S0 -> S1\noutput:y1 y2 y3\n";

//Sgraf << "S0 -> S1\noutput:y1 y2 y3\n";

state[0] = 0; state[1] = 0; state[2] = 0; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 0 && !x[0] && x[1] && !x[2] && x[3] && !x[15]) {

cout << "S0 -> S2\noutput: -\n";

//Sgraf << "S0 -> S2\noutput: -\n";

state[0] = 0; state[1] = 0; state[2] = 0; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 0 && x[0]) {

cout << "S0 -> S2\noutput: -\n";

//Sgraf << "S0 -> S2\noutput: -\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 0;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 0 && !x[0] && x[1] && !x[2] && !x[3] && x[4]) {

cout << "S0 -> S2\noutput: -\n";

//Sgraf << "S0 -> S2\noutput: -\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 0;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 0 && !x[0] && x[1] && !x[2] && !x[3] && x[15]) {

cout << "S0 -> S4\noutput: -\n";

//Sgraf << "S0 -> S4\noutput: -\n";

state[0] = 0; state[1] = 1; state[2] = 0; state[3] = 0;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 0 && !x[0] && x[1] && x[2]) {

cout << "S0 -> S4\noutput: -\n";

//Sgraf << "S0 -> S4\noutput: -\n";

state[0] = 0; state[1] = 1; state[2] = 0; state[3] = 0;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 1 && !x[5]) {

if (cycle > 1) {

cout << "Возник Цикл в состоянии S1";

//Sgraf << "Возник Цикл в состоянии S1";

break;

}

cout << "S1 -> S1\noutput: -\n";

//Sgraf << "S1 -> S1\noutput: -\n";

state[0] = 0; state[1] = 0; state[2] = 0; state[3] = 1;

cycle++;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 1 && x[5]) {

cout << "S1 -> S2\noutput: y4\n";

//Sgraf << "S1 -> S2\noutput: y4\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 0;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 0 && x[6] && x[7] && x[8]) {

cout << "S2 -> S3\noutput: y5y6y7y8\n";

//Sgraf << "S2 -> S3\noutput: y5y6y7y8\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 0 && !x[6] && x[9] && x[16]) {

cout << "S2 -> S3\noutput: y21y22y23\n";

//Sgraf << "S2 -> S3\noutput: y21y22y23\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 0 && !x[6] && x[9] && !x[16] && !x[20]) {

cout << "S2 -> S5\noutput: y34\n";

//Sgraf << "S2 -> S5\noutput: y34\n";

state[0] = 0; state[1] = 1; state[2] = 0; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 0 && x[6] && x[7] && !x[8]) {

cout << "S2 -> S7\noutput: y9y10y11\n";

//Sgraf << "S2 -> S7\noutput: y9y10y11\n";

state[0] = 0; state[1] = 1; state[2] = 1; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 0 && x[6] && !x[7]) {

cout << "S2 -> S7\noutput: y9y10y11\n";

//Sgraf << "S2 -> S7\noutput: y9y10y11\n";

state[0] = 0; state[1] = 1; state[2] = 1; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 0 && !x[6] && !x[9]) {

cout << "S2 -> S7\noutput: y9y10y11\n";

//Sgraf << "S2 -> S7\noutput: y9y10y11\n";

state[0] = 0; state[1] = 1; state[2] = 1; state[3] = 1;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 0 && !x[6] && x[9] && !x[16] && x[20]) {

cout << "S2 -> S8\noutput: y29y30y31\n";

//Sgraf << "S2 -> S8\noutput: y29y30y31\n";

state[0] = 1; state[1] = 0; state[2] = 0; state[3] = 0;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 1 && !x[17]) {

if (cycle > 1) {

cout << "Возник Цикл в состоянии S3";

//Sgraf << "Возник Цикл в состоянии S3";

break;

}

cout << "S3 -> S3\noutput: -\n";

//Sgraf << "S3 -> S3\noutput: -\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 1;

cycle++;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 1 && x[17] && !x[18]) {

cout << "S3 -> S4\noutput: y24\n";

//Sgraf << "S3 -> S4\noutput: y24\n";

state[0] = 0; state[1] = 1; state[2] = 0; state[3] = 0;

}

if (state[0] == 0 && state[1] == 0 && state[2] == 1 && state[3] == 1 && x[17] && x[18]) {

cout << "S3 -> S9\noutput: y25y26y27y28\n";

//Sgraf << "S3 -> S9\noutput: y25y26y27y28\n";

state[0] = 1; state[1] = 0; state[2] = 0; state[3] = 1;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 0 && state[3] == 0) {

cout << "S4 -> S5\noutput: y33\n";

//Sgraf << "S4 -> S5\noutput: y33\n";

state[0] = 0; state[1] = 1; state[2] = 0; state[3] = 1;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 0 && state[3] == 1) {

cout << "S5 -> S6\noutput: y35y36y37\n";

//Sgraf << "S5 -> S6\noutput: y35y36y37\n";

state[0] = 0; state[1] = 1; state[2] = 1; state[3] = 0;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 1 && state[3] == 0 && !x[22]) {

if (cycle > 1) {

cout << "Возник Цикл в состоянии S6";

//Sgraf << "Возник Цикл в состоянии S6";

break;

}

cout << "S6 -> S6\noutput: -\n";

//Sgraf << "S6 -> S6\noutput: -\n";

state[0] = 0; state[1] = 1; state[2] = 1; state[3] = 0;

cycle++;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 1 && state[3] == 0 && x[22]) {

cout << "S6 -> S0\noutput: y38\n";

//Sgraf << "S6 -> S0\noutput: y38\n";

state[0] = 0; state[1] = 0; state[2] = 0; state[3] = 0;

break;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 1 && state[3] == 1 && !x[10]) {

if (cycle > 1) {

cout << "Возник Цикл в состоянии S7";

//Sgraf << "Возник Цикл в состоянии S7";

break;

}

cout << "S7 -> S7\noutput: -\n";

//Sgraf << "S7 -> S7\noutput: -\n";

state[0] = 0; state[1] = 1; state[2] = 1; state[3] = 1;

cycle++;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 1 && state[3] == 1 && x[10] && !x[11]) {

if (cycle > 1) {

cout << "Возник Цикл в состоянии S7";

//Sgraf << "Возник Цикл в состоянии S7";

break;

}

cout << "S7 -> S2\noutput: y12y13\n";

//Sgraf << "S7 -> S2\noutput: y12y13\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 0;

cycle++;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 1 && state[3] == 1 && x[10] && x[11] && !x[12] && !x[13] && !x[14]) {

if (cycle > 1) {

cout << "Возник Цикл в состоянии S7";

//Sgraf << "Возник Цикл в состоянии S7";

break;

}

cout << "S7 -> S2\noutput: y12y13\n";

//Sgraf << "S7 -> S2\noutput: y12y13\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 0;

cycle++;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 1 && state[3] == 1 && x[10] && x[11] && !x[12] && !x[13] && x[14]) {

cout << "S7 -> S4\noutput: y20\n";

//Sgraf << "S7 -> S4\noutput: y20\n";

state[0] = 0; state[1] = 1; state[2] = 0; state[3] = 0;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 1 && state[3] == 1 && x[6] && x[7] && x[8]) {

cout << "S7 -> S8\noutput: y17y18y19\n";

//Sgraf << "S7 -> S8\noutput: y17y18y19\n";

state[0] = 1; state[1] = 0; state[2] = 0; state[3] = 0;

}

if (state[0] == 0 && state[1] == 1 && state[2] == 1 && state[3] == 1 && x[10] && x[11] && x[12]) {

cout << "S7 -> S3\noutput: y14y15y16\n";

//Sgraf << "S7 -> S3\noutput: y14y15y16\n";

state[0] = 0; state[1] = 0; state[2] = 1; state[3] = 1;

}

if (state[0] == 1 && state[1] == 0 && state[2] == 0 && state[3] == 0 && !x[21]) {

if (cycle > 1) {

cout << "Возник Цикл в состоянии S8";

//Sgraf << "Возник Цикл в состоянии S8";

break;

}

cout << "S8 -> S8\noutput: -\n";

//Sgraf << "S8 -> S8\noutput: -\n";

state[0] = 1; state[1] = 0; state[2] = 0; state[3] = 0;

cycle++;

}

if (state[0] == 1 && state[1] == 0 && state[2] == 0 && state[3] == 0 && x[21]) {

cout << "S8 -> S4\noutput: y32\n";

//Sgraf << "S8 -> S4\noutput: y32\n";

state[0] = 0; state[1] = 1; state[2] = 0; state[3] = 0;

}

if (state[0] == 1 && state[1] == 0 && state[2] == 0 && state[3] == 1 && !x[19]) {

if (cycle > 1) {

cout << "Возник Цикл в состоянии S9";

//Sgraf << "Возник Цикл в состоянии S9";

break;

}

cout << "S9 -> S9\noutput: -\n";

//Sgraf << "S9 -> S9\noutput: -\n";

state[0] = 1; state[1] = 0; state[2] = 0; state[3] = 1;

cycle++;

}

if (state[0] == 1 && state[1] == 0 && state[2] == 0 && state[3] == 1 && x[19]) {

cout << "S9 -> S4\noutput: -\n";

//Sgraf << "S9 -> S4\noutput: -\n";

state[0] = 0; state[1] = 1; state[2] = 0; state[3] = 0;

}

}

//Sgraf << "\n|------------------|\n";

//Sgraf.close();

}

void logic() {

string S;

while (1) {

// выходные

y[0] = y[1] = y[2] = (!state[0] & !state[1] & !state[2] & !state[3] & !x[0] & x[1] & !x[2] & !x[3] & !x[4]) | (!state[0] & !state[1] & !state[2] & !state[3] & !x[0] & !x[1] & !x[3] & !x[4]);

y[3] = !state[0] & !state[1] & !state[2] & state[3] & x[5];

y[4] = y[5] = y[6] = y[7] = !state[0] & !state[1] & state[2] & !state[3] & x[6] & x[7] & x[8];

y[8] = y[9] = y[10] = (!state[0] & !state[1] & state[2] & !state[3] & x[6] & x[7] & !x[8]) | (!state[0] & !state[1] & state[2] & !state[3] & x[6] & !x[7]) | (0 & 0 & 1 & 0 & !x[6] & !x[9]);

y[11] = y[12] = (!state[0] & state[1] & state[2] & state[3] & x[10] & !x[11]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & !x[13] & !x[14]);

y[13] = y[14] = y[15] = !state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & x[12];

y[16] = y[17] = y[18] = !state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & x[13];

y[19] = !state[0] & state[1] & state[2] & !state[3] & x[10] & x[11] & !x[12] & !x[13] & x[14];

y[20] = y[21] = y[22] = !state[0] & !state[1] & state[2] & !state[3] & !x[6] & x[9] & x[16];

y[23] = !state[0] & !state[1] & state[2] & state[3] & x[17] & !x[18];

y[24] = y[25] = y[26] = y[27] = !state[0] & !state[1] & state[2] & state[3] & x[17] & x[18];

y[28] = y[29] = y[30] = !state[0] & !state[1] & state[2] & !state[3] & !x[6] & x[9] & !x[16] & !x[20];

y[31] = state[0] & !state[1] & !state[2] & !state[3] & x[21];

y[32] = !state[0] & state[1] & !state[2] & !state[3];

y[33] = !state[0] & !state[1] & state[2] & !state[3] & !x[6] & x[9] & !x[16] & !x[20];

y[34] = y[35] = y[36] = !state[0] & state[1] & !state[2] & state[3];

y[37] = !state[0] & state[1] & state[2] & !state[3] & x[22];

//функции перехода

w[0] = (!state[0] & !state[1] & state[2] & !state[3] & x[9] & !x[16] & x[20]) | (!state[0] & !state[1] & state[2] & state[3] & x[17] & x[18]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & x[13]);

w[1] = (!state[0] & !state[1] & !state[2] & !state[3] & !x[0] & x[1] & !x[2] & x[3] & x[4]) | (!state[0] & !state[1] & !state[2] & !state[3] & !x[0] & x[1] & x[2]) | (!state[0] & !state[1] & state[2] & !state[3] & !x[6] & x[9] & !x[16] & !x[20]) | (!state[0] & !state[1] & state[2] & !state[3] & x[6] & x[7] & !x[8]) | (!state[0] & !state[1] & state[2] & !state[3] & x[6] & !x[7]) | (!state[0] & !state[1] & state[2] & !state[3] & !x[6] & !x[9]) | (!state[0] & !state[1] & state[2] & state[3] & x[17] & !x[18]) | (state[0] & !state[1] & !state[2] & !state[3] & x[21]) | (state[0] & !state[1] & !state[2] & state[3] & x[19]);

w[2] = (!state[0] & !state[1] & !state[2] & !state[3] & x[0]) | (!state[0] & !state[1] & !state[2] & !state[3] & !x[0] & x[1] & !x[2] & !x[3] & x[4]) | (!state[0] & !state[1] & !state[2] & state[3] & x[5]) | (!state[0] & state[1] & !state[2] & state[3]);

w[3] = (!state[0] & !state[1] & !state[2] & !state[3] & !x[0] & x[1] & !x[2] & !x[3] & !x[4]) | (!state[0] & !state[1] & !state[2] & !state[2] & !state[3] & !x[0] & !x[1] & !x[3] & !x[4]) | (!state[0] & !state[1] & !state[2] & !state[3] & !x[0] & x[1] & !x[2] & x[3] & !x[15]) | (!state[0] & !state[1] & state[2] & !state[3] & x[6] & x[7] & x[8]) | (!state[0] & !state[1] & state[2] & !state[3] & !x[6] & x[9] & x[16]) | (!state[0] & !state[1] & state[2] & !state[3] & !x[6] & x[9] & !x[16] & !x[20]) | (!state[0] & !state[1] & state[2] & !state[3] & x[6] & x[7] & !x[8]) | (!state[0] & !state[1] & state[2] & !state[3] & x[6] & !x[7]) | (!state[0] & !state[1] & state[2] & !state[3] & !x[6] & !x[9]) | (!state[0] & state[1] & !state[2] & !state[3]);

u[0] = (state[0] & !state[1] & !state[2] & !state[3] & x[21]) | (state[0] & !state[1] & !state[2] & state[3] & x[19]);

u[1] = (!state[0] & state[1] & state[2] & !state[3] & x[22]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & !x[11]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & !x[13] & !x[14]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & x[13]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & x[12]);

u[2] = (!state[0] & !state[1] & state[2] & !state[3] & !x[6] & x[9] & !x[16] & !x[20]) | (!state[0] & !state[1] & state[2] & !state[3] & !x[6] & x[9] & !x[16] & x[20]) | (!state[0] & !state[1] & state[2] & state[3] & x[17] & !x[18]) | (!state[0] & !state[1] & state[2] & state[3] & x[17] & x[18]) | (!state[0] & state[1] & state[2] & !state[3] & x[22]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & !x[13] & x[14]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & x[13]);

u[3] = (!state[0] & !state[1] & !state[2] & state[3] & x[5]) | (!state[0] & !state[1] & state[2] & state[3] & x[17] & !x[18]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & !x[11]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & !x[13] & !x[14]) | (!state[0] & state[1] & state[2] & !state[3] & x[10] & x[11] & !x[12] & !x[13] & x[14]) | (!state[0] & state[1] & state[2] & state[3] & x[10] & x[11] & !x[12] & x[13]) | (state[0] & !state[1] & !state[2] & state[3] & x[19]) | (!state[0] & state[1] & !state[2] & state[3]);

for (int i = 0; i < 4; i++) {

cout << state[i];

if (state[i] == 1)

S += '1';

else

S += '0';

}

for (int i = 0; i < 4; i++) {

if (w[i])

state[i] = 1;

if (u[i])

state[i] = 0;

}

cout << " -> ";

for (int i = 0; i < 4; i++) {

cout << state[i];

}

cout << " ";

for (int i = 0; i < 38; i++) {

if (y[i])

cout << "y" << i + 1 << " ";

}

cout << ": ";

for (int i = 0; i < 38; i++) {

cout << y[i];

}

if (state[0] == 0 && state[1] == 0 && state[2] == 0 && state[3] == 0)

break;

if (s[bitset<4>(S).to\_ulong()] > 7) {

cout << "\nЗацикливание...";

break;

} else

s[bitset<4>(S).to\_ulong()]++;

cout << "\n";

}

}

/\*

void testing() {

int i = 0;

string buf;

while (bitset<23>(i).to\_string() != "11111111111111111111111") {

Sgraf.open("Sgraf.txt", ios\_base::app);

test.open("test.txt");

buf = bitset<23>(i).to\_string();

test << bitset<23>(i);

test.close();

reader();

Sgraf << "Набор x: " << buf;

Sgraf << "\n";

graf();

i++;

}

}

\*/

int main() {

double start, end;

srand(time(0));

setlocale(LC\_ALL, "Russian");

int choice;

//testing();

reader();

cout << "Выбор алгоритма(Выбирите числовой пункт):\n 1 Логическое выражение \n 2 Граф-схема\nВаш выбор: ";

cin >> choice;

switch (choice) {

case 1:

start = clock();

logic();

end = clock();

break;

case 2:

start = clock();

graf();

end = clock();

break;

default:

cout << "Выбран неправильный пункт!\n";

system("pause");

exit(0);

break;

}

cout << "\nВремя выполнения: " << end - start << " мс" << endl;

return 0;

}

1. URL: <http://perscom.ru/2012-01-20-09-26-05/7-ustroistva-upravlenia-microprogrammniy-avtomat/18-microprogramni-avtomat-s-jestkoi-logikoi-s-programmiruemoi-logicoi> [↑](#footnote-ref-1)
2. https://ru.wikipedia.org/wiki/- Управляющие автоматы с жесткой логикой [↑](#footnote-ref-2)
3. И.В. Жукалина Теория автоматов: методические указания к курсовому проекту для специальности 230101 / И.В. Жукалина. - Оренбург: ГОУ ОГУ, 2008. – 29 с. [↑](#footnote-ref-3)