

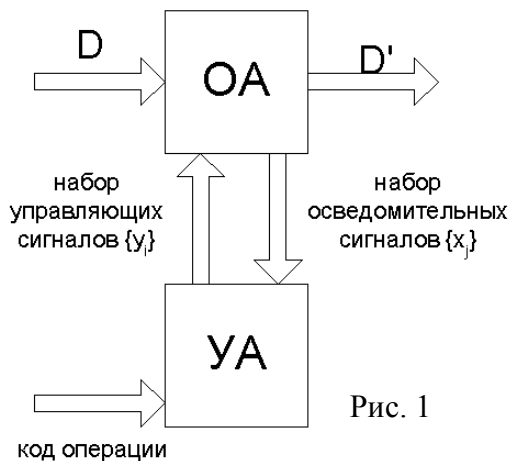
Краткий конспект лекций (по двум типам курсовых работ)

I. Синтез цифровых автоматов для реализации алгоритмов двоичной арифметики

1. Общие сведения о цифровых автоматах. Модель Глушкова. Синтез операционных автоматов.

Вначале приведем некоторые общие сведения о **цифровых, операционных и управляющих автоматах (ЦА, ОА, УА)**.

Согласно модели академика В.М. Глушкова **цифровой автомат (ЦА)** как устройство для автоматической обработки цифровой информации по заданным алгоритмам представляет собой совокупность **операционного автомата (ОА)** и **управляющего автомата (УА)** – рис.1



Первый автомат (**ОА**) служит для выполнения собственно набора требуемых операций алгоритма, а второй задает последовательность действий по алгоритму в зависимости от условий (которые также формируются операционным автоматом как логические сигналы).

Работа автомата разбивается на такты (дискретные интервалы времени). Каждая элементарная, атомарная (неделимая) операция, которая может выполняться в ОА за один такт, называется **микрооперацией**. Совокупность микроопераций, которые могут выполняться в

ОА параллельно в одном такте, называется **микрокомандой**. Последовательность микрокоманд, реализующих алгоритм, называется **микропрограммой**.

Управляющий автомат (УА) предназначен для выдачи управляющих сигналов в каждом такте работы ЦА, инициирующих выполнение определенных микроопераций (или микрокоманд) в ОА в соответствии с выполняемым алгоритмом и в зависимости от поступающих на входы УА информационных сигналов (условий). Фактически УА реализует **последовательность действий** по алгоритму, при этом содержание этих действий зависит от управляемого объекта, в данном случае – от ОА. Если **ОА "знает как"** делать, то **УА "знает что и когда"**, то есть в какой последовательности что-то делать. При этом для УА "что делать" – это просто коды команд, про их содержание он не знает.

Исходной информацией для синтеза **операционного автомата** является требуемый для выполнения определенного набора алгоритмов (или одного алгоритма) набор операций. В алгоритме выделяют переменные, выходные и внутренние слова, набор микроопераций и логических условий, после чего выполняется синтез.

Этапы синтеза ОА:

1) Каждому входному слову ставится в соответствие **шина** необходимой разрядности, которая соединяется со входом **регистра** также необходимой разрядности, предназначенного для хранения этого входного слова.

2) Каждому выходному слову ставится в соответствие шина, соединенная с выходом регистра.

3) Каждой внутренней переменной ставится в соответствие регистр необходимой разрядности, либо **счетчик** для хранения этой переменной.

4) Каждой микрооперации y_i , задаваемой в алгоритме оператором присваивания, ставится в соответствие **операционный элемент** (ОЭ - комбинационная схема), либо сдвиговый регистр, либо счетчик $y_i : b_i = \Psi_i(Q_0, Q_1, \dots, Q_k)$, входы которого соединены с выходами регистров операндов с помощью неуправляемых шин, а выход соединен со входом регистра результата с помощью шины, управляемой сигналом y_i .

5) Каждому логическому условию $x_j = \lambda_j(Q_0, Q_1, \dots, Q_k)$ ставится в соответствие комбинационная схема для его формирования, входы которой соединены с регистрами операндов, а выходы являются выходами автомата.

В результате этого синтеза получается автомат канонической структуры (рис. 2).

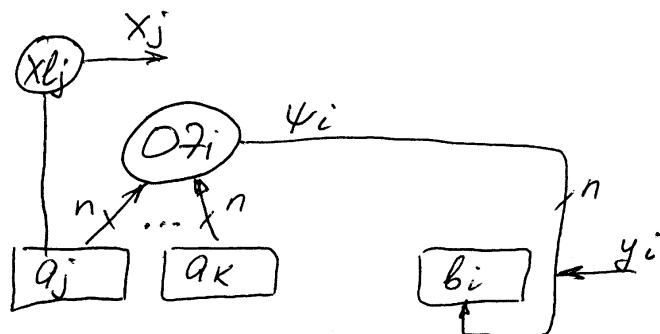


Рис.2

Часто такой автомат является избыточным, поскольку содержит избыточное количество ОЭ. Эти операционные элементы можно исключить, используя одни и те же ОЭ для выполнения различных микроопераций.

Исключив все элементы, которые являются избыточными, получим **I - автомат**. При этом избыточными являются элементы, которые нельзя использовать параллельно с другими и которые могут быть заменены другими существующими в автомате элементами и схемами коммутации.

2. Пример синтеза операционного автомата для выполнения косвенного умножения беззнаковых чисел

Рассмотрим пример синтеза автомата канонической структуры на примере автомата для выполнения умножения беззнаковых четырехразрядных чисел (модулей).

Используем схему умножения с младших разрядов множителя со сдвигом вправо суммы частичных произведений (СЧП). Пусть A - множимое, B - множитель, C - СЧП. Младший разряд имеет индекс n ($n = 4$). Получим алгоритм, изображенный на рис. 3.

Пример вычислений по алгоритму приведен ниже.

Пример:

$$A = 5/16 \quad B = 6/16$$

$$A = ,0101 \quad B = ,0110$$

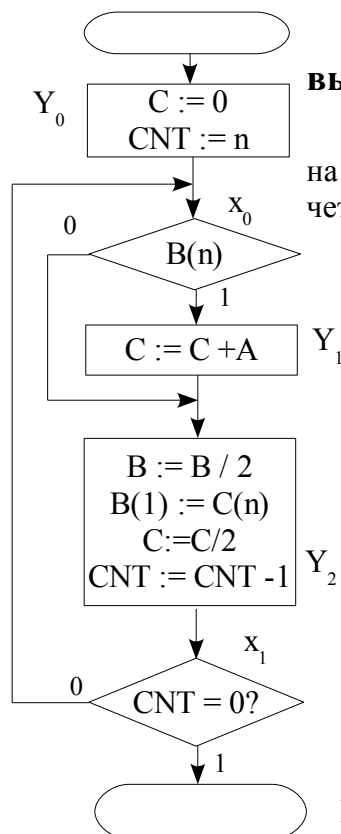


Рис. 3

Регистр С	Регистр В	В _n	Комментарий
0000	011	0	C:=0; CNT=4 (n=4) - счетчик
0000	001	1	B(4)=0 => B→; C→; CNT=4-1=3;
+0101	001	1	CNT<>0 B(4)=1 => C:=C+A
0101			
0010	000	1	→; CNT=3-1=2;
+0101			CNT<>0 B(4)=1 => C:=C+A
0111	000	1	
0011	100	0	→; CNT=2-1=1;
0001	110	1	CNT<>0 B(4)=0 => →; CNT=1-1=0;

Результат : произведение равно 30 / 256 (старшая часть в С, младшая - в В, множитель не сохраняется).

В алгоритме можно выделить следующие **переменные** :

A, В – входные переменные
В, С – выходные переменные
CNT – внутренняя переменная (счетчик)

Также в алгоритме можно выделить несколько **микроопераций**, часть из которых могут выполняться параллельно и их можно объединить в **микрокоманды**. Всего отдельных микрокоманд три, они обозначены на рис. 3 Y0 – Y2.

В одной микрокоманде можно выполнять совместные микрооперации. К несовместимым относятся:

- 1) Запись значений в один и тот же регистр.
- 2) Запись в регистр и его сдвиг (то есть параллельная и последовательная запись) и др.

Помимо микрокоманд в алгоритме выделяем также два логических условия (**информационных сигнала ОА**). Они обозначены на рис. 3 как x0 и x1.

По данному алгоритму в соответствии с рассмотренными выше этапами синтеза можно синтезировать следующую функциональную схему операционного автомата (рис. 4) :

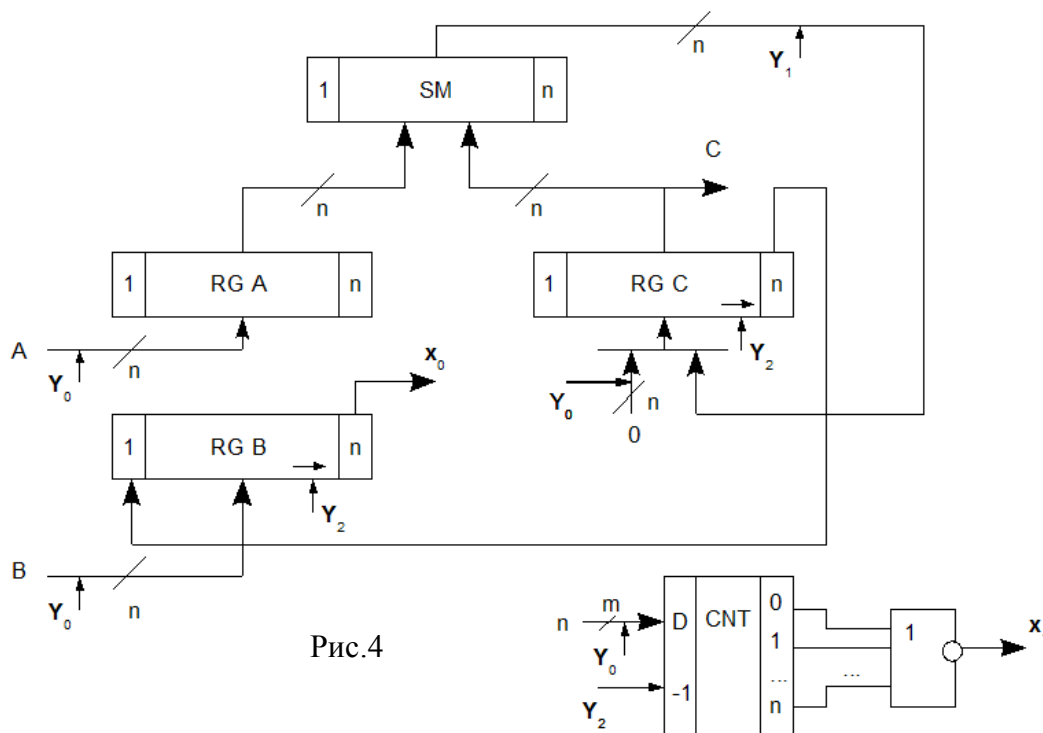


Рис.4

3. Виды управляющих автоматов. Структуры автоматов Мили и Мура.

Управляющие автоматы прежде всего делят на две большие группы – автоматы с жесткой логикой (**УАЖЛ**) и автоматы с программируемой (хранимой) логикой (**УАПЛ**).

Автоматы первой группы строятся на базе памяти состояний, которая обычно реализуется совокупностью триггеров, и комбинационной схемы, которая управляет переключением триггеров (то есть - сменой состояний) и формированием выходных (управляющих) сигналов в зависимости от входных сигналов и текущего состояния. Код текущего состояния хранится в триггерах. Схема жестко реализует закон функционирования автомата, откуда следует название автоматов данной группы.

Автоматы с программируемой или хранимой логикой строятся на базе **памяти микропрограмм (ПМП)**. В ПМП хранится микропрограмма (аналогичная по виду ассемблерному коду, но на еще более низком уровне), реализующая алгоритм (или алгоритмы).

УАЖЛ обычно характеризуются большим быстродействием (в среднем требуют меньшего числа тактов на выполнение алгоритмов), но для сложных алгоритмов или – большого числа реализуемых алгоритмов характеризуются большей аппаратной сложностью, кроме того, они узко специализированы. УАПЛ более универсальны, так как микропрограмму для конкретного УАПЛ в принципе можно изменить. Для более сложных алгоритмов или наборов алгоритмов они также требуют меньше аппаратных затрат, чем УАЖЛ. При этом они характеризуются меньшим быстродействием. Для простых алгоритмов затраты на УАПЛ могут быть выше, чем на УАЖЛ. УАПЛ часто используются в устройствах управления процессоров с полным набором команд (CISC), а УАЖЛ – в процессорах с сокращенным набором команд (RISC).

Для построения обоих типов автоматов, но прежде всего, УАЖЛ, используется теория **абстрактных конечных автоматов (КА)**. Для построения используется две базовые модели КА, функционально аналогичные : автомат **Мура** и автомат **Мили**.

С точки зрения абстрактной ТА автомат представляет собой совокупность множеств и отображений. Автомат **Мили** задается как шестерка объектов:

$$A = \langle X, Y, Q, q_0, \delta, \lambda \rangle$$

где:

- X – множество входных символов автомата
- Y – множество выходных символов автомата
- Q – множество состояний автомата
- q_0 – начальное состояние автомата
- δ – функция переходов $\delta: Q \times X \rightarrow Q$
- λ – функция выходов $\lambda: Q \times X \rightarrow Y$

Модель автомата Мура отличается видом функции выходов - выходные сигналы полностью определяются текущим состоянием и не зависят непосредственно от текущих входных сигналов. На практике для задания автоматов пользуются либо табличной формой задания функций переходов и выходов, либо – графовой.

Рассмотрим табличные и графовые способы задания автоматов Мили и Мура.

Для задания автомата Мили составляют таблицы переходов и выходов. Строки – входные сигналы (ВС), столбцы – предыдущие состояния (ПС). В клетках – следующие состояния или выходные сигналы.

Пример:

Таблица переходов (ТП)

BC	ПС	S0	S1	S2	S3
X1		S1	S3	S1	S3
X2		S2	S0	S2	S0

Таблица выходов (ТВ)

BC	ПС	S0	S1	S2	S3
X1		Y3	Y1	Y3	Y1
X2		Y2	Y0	Y2	Y0

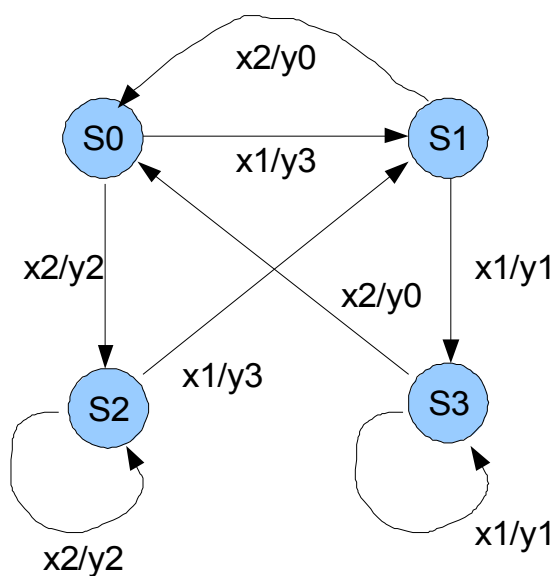
Для автомата Мура используется отмеченная таблица переходов (ОТП)

Пример:

BC	ПС	Y0	Y1	Y2	Y1
		S0	S1	S2	S3
X1		S1	S3	S1	S3
X2		S2	S0	S2	S0

Граф автомата - ориентированный граф, у которого в качестве вершин используются состояния, а в качестве дуг - переходы. В начале дуги фиксируется входной сигнал. Что касается выходного сигнала, то он для ЦА Мили ставится на конце дуги, а у Мура внутри вершины (рис.5).

Мили:



Мура:

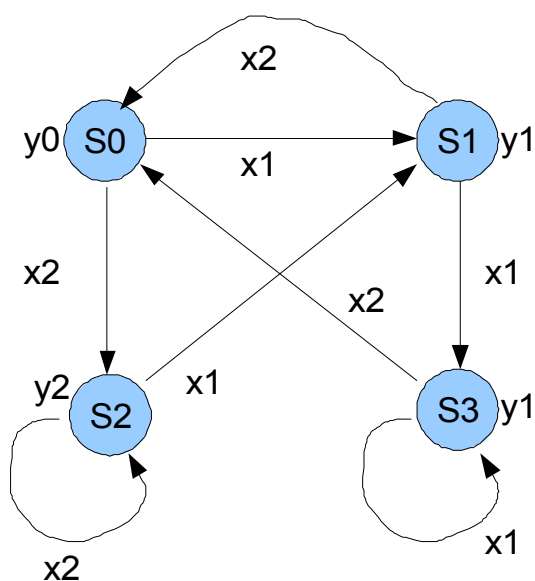


Рис. 5

Закон функционирования автомата можно составить по алгоритму. Для этого выполняют разметку алгоритма – для автомата Мили состояния помечаются на выходе операторных вершин (за исключением ветвлений), а для Мура – возле каждой операторной

вершины (также за исключением ветвлений). В общем случае у автомата Мура состояний больше, так как точке объединения двух ветвей в алгоритме, содержащих операторы, будет соответствовать минимум одно состояние автомата Мили и минимум два состояния в автомате Мура. Затем рассматриваются все возможные переходы между состояниями по алгоритму (см. далее пример).

Далее переходим к рассмотрению структур и реализации УАЖЛ.

Структурные схемы автоматов Мили и Мура представлены на рис. 6.

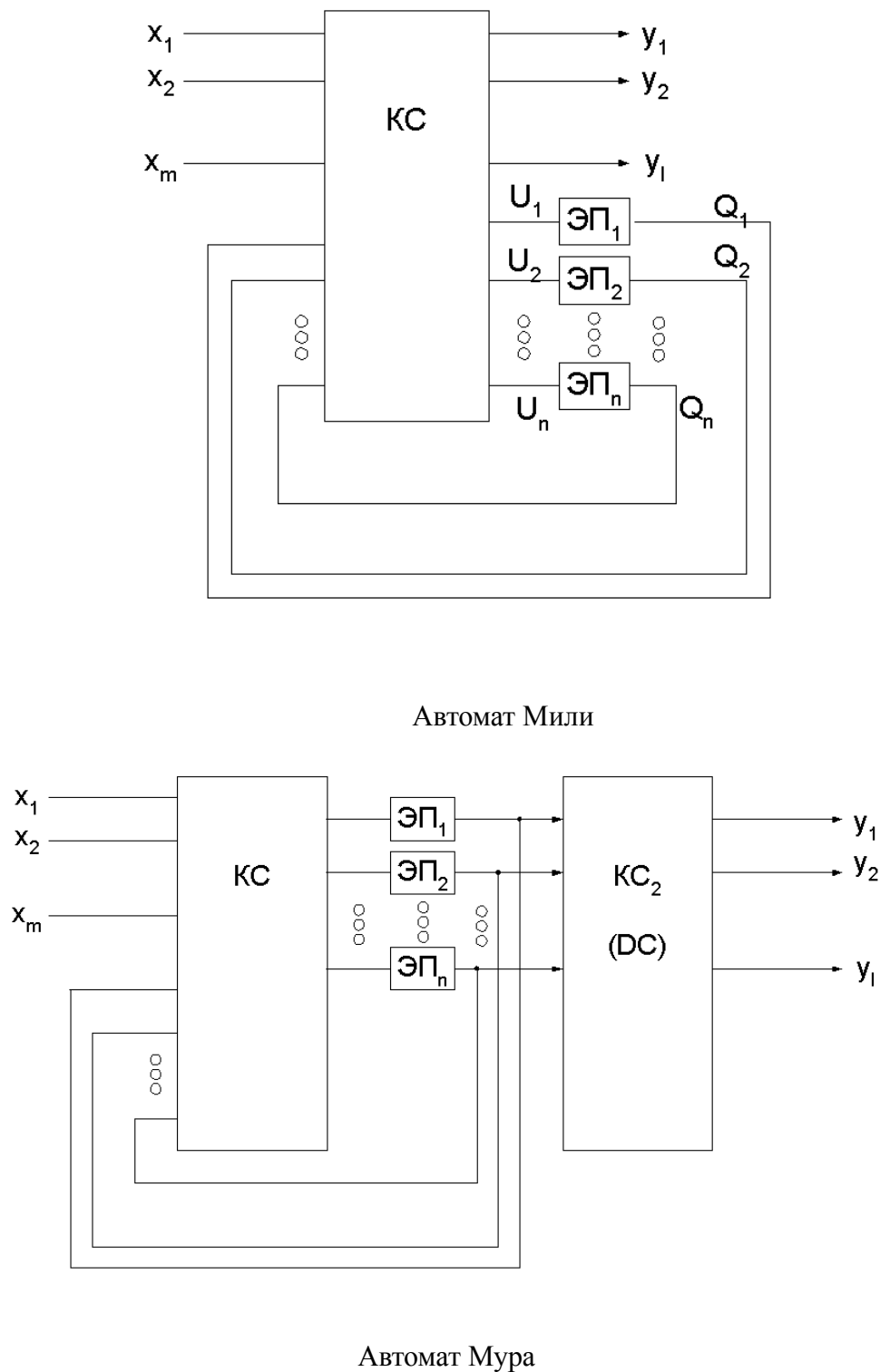


Рис. 6

Далее рассмотрим пример синтеза УАЖЛ по модели автомата Мура для рассмотренного выше алгоритма.

4. Пример синтеза управляющего автомата с жесткой логикой (УАЖЛ) для алгоритма умножения беззнаковых чисел в прямом коде

Рассмотрим синтез управляющего автомата с жесткой логикой для ЦА, реализующего алгоритм косвенного умножения модулей (беззнаковых чисел).

Схема алгоритма приведена на рис. 7. (дополнительно к рассмотренному выше алгоритму в этом алгоритме введен цикл ожидания начала работы и соответствующий управляющий сигнал Start).

Будем синтезировать **автомат Мура на базе D-триггеров**. Сначала обозначим состояния на алгоритме и составим отмеченную таблицу переходов (ОТП) для автомата Мура (см. Таблицу 1)

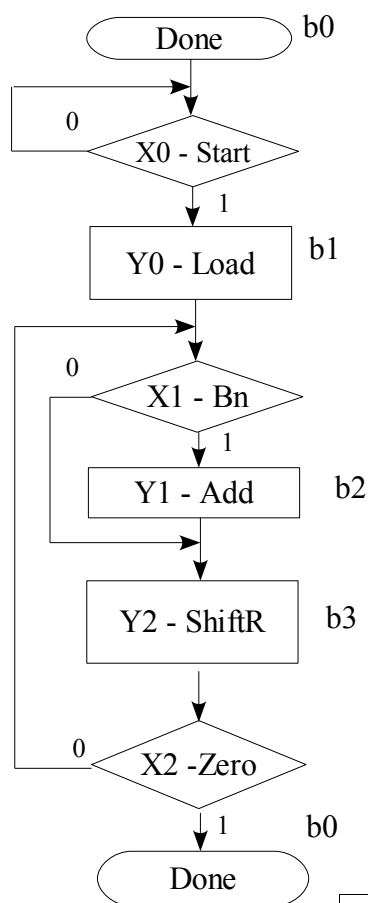


Рис. 7

Таблица 1

BC/ПС	b0	b1	b2	b3
Start	b1	-	-	-
!Start	b0	-	-	-
!X1!X2	-	b3	b3	b3
!X1X2	-	b3	b3	b2
X1!X2	-	b2	b3	b0
X1X2	-	b2	b3	b0

Теперь можно приступить собственно к синтезу УАЖЛ.

2.1 Закодируем состояния автомата с помощью обычных двоичных кодов :

Таблица 2

Состояние	Код
b0	00
b1	01
b2	10
b3	11

2.2 Для заданного типа триггеров (D-триггеры) составим его таблицу переходов как элементарного автомата :

Таблица 3

Переход	D
$0 \rightarrow 0$	0
$0 \rightarrow 1$	1
$1 \rightarrow 0$	0
$1 \rightarrow 1$	1

Для других типов триггеров соответствующие таблицы переходов (* - любой сигнал):

Переход	T	RS	JK
$0 \rightarrow 0$	0	*0	0*
$0 \rightarrow 1$	1	01	1*
$1 \rightarrow 0$	0	10	*1
$1 \rightarrow 1$	1	0*	*0

2.3 Составим закодированную таблицу переходов (выходные сигналы в таблице не отмечаем, так как для автомата Мура их можно получить с помощью дешифратора кода состояния).

Таблица 4

Состояние		Вх. сигнал	Управление триггерами	
Текущ.	След.		D1	D0
Q1 Q0	Q1 Q0			
00	00	!Start	0	0
00	01	Start	0	1
01	10	Bn	1	0
01	11	!Bn	1	1
10	11	1	1	1
11	10	!Zero Bn	1	0
11	11	!Zero !Bn	1	1
11	00	Zero	0	0

2.4 По закодированной таблице составляем логические функции для сигналов управления триггерами D1 и D0 (выбираем в данном случае нули из столбцов D1 и D0 соответственно и записываем для каждого нуля конъюнкцию кода текущего состояния и комбинации входных сигналов, объединяем конъюнкции дизъюнкциями, затем берем отрицание полученной функции; все отрицания записываем с помощью «!») :

$$D1 = !(Q1!Q0 + Q1Q0Zero)$$

$$D0 = !(Q1!Q0!Start + Q1Q0Zero + !Q1Q0Bn + Q1Q0!ZeroBn) =$$

$$!(Q1!Q0!Start + Q1Q0Zero + Q1Q0Bn + !Q1Q0Bn) =$$

$$!(Q1!Q0!Start + Q1Q0Zero + Q0Bn)$$

2.5 По полученным функциям составляем схему устройства в соответствии с его структурной схемой (рис. 8, схема составлена в системе Quartus II, поэтому в ней использованы обозначения ANSI, в ваших схемах используйте обозначения ЕСКД – прямоугольные – см. приложение к основным лекциям)

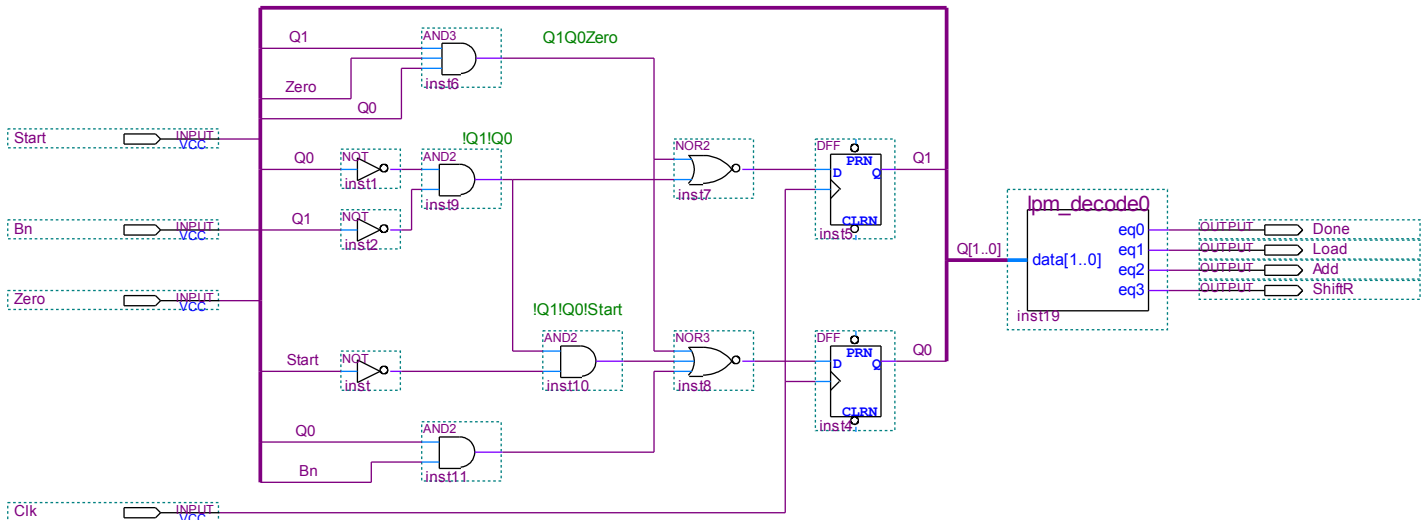


Рис.8

Теперь мы можем создать схему ЦА для умножения из двух частей – ОА и УА - см. рис. 9 :

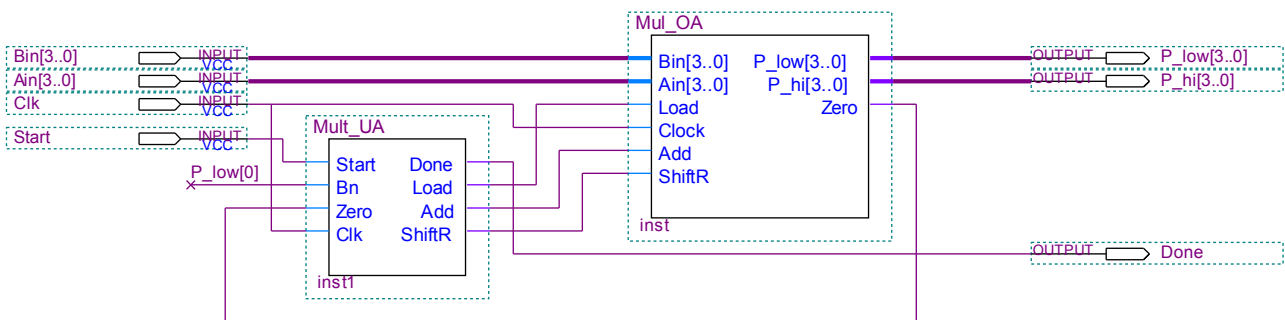


Рис. 9

Отличие в синтезе автомата Мили заключается в том, что в соответствии с его законом функционирования и структурной схемой необходимо синтезировать не только логические функции для переключения триггеров, но и логические функции формирования выходных сигналов Y в явном виде (в таблицу 4 вводятся дополнительные столбцы, соответствующие каждому Y_i и отмечаются единицами строки, то есть переходы, при которых формируется такой сигнал). В автомате Мура выходные сигналы формируются на обычном дешифраторе (см. рис. 6).

5. Контрольные вопросы

1. Опишите модель дискретного преобразователя Глушкова.
2. Каково назначение Операционного Автомата (ОА) ?
3. Роль информационных и управляющих сигналов.
4. Перечислите этапы синтеза ОА процедурного типа канонической структуры.
5. Проведите синтез ОА для выполнения умножения (используйте упрощенный алгоритм косвенного умножения беззнаковых чисел).
6. Приведите пример умножения чисел по алгоритму косвенного умножения.
7. Опишите основные варианты (схемы) косвенного умножения.
8. Постройте временную диаграмму ОА для умножения. Поясните ее.
9. Синтезируйте алгоритм выполнения арифметической операции по вариантам.
10. Синтезируйте схему ОА по вашему алгоритму.
11. Каково назначение Управляющего Автомата (УА) ?
12. Роль информационных и управляющих сигналов.
13. Типы УА.
14. Перечислите этапы синтеза УА с жесткой логикой.
15. Опишите основные триггеры как элементарные конечные автоматы.
16. Каковы особенности синтеза УАЖЛ на разных типах триггеров ?
17. Приведите структурные схемы автоматов Мили и Мура. В чем их отличия ?
18. Опишите модели абстрактных автоматов Мили и Мура.
19. Какие формы описания абстрактных конечных автоматов Вы знаете ?
20. Постройте таблицы переходов / выходов и графы автоматов по схеме алгоритма.
21. Проведите синтез УА для реализации алгоритма умножения (используйте упрощенный алгоритм косвенного умножения беззнаковых чисел) как автомат Мили / Мура.
22. Постройте временную диаграмму УА для умножения. Поясните ее.
23. Синтезируйте схему УАЖЛ для вашего алгоритма по вариантам.

II Использование регулярных выражений (РВ). Программная реализация автоматов

1 Понятие о регулярных выражениях и автоматах-распознавателях

1.1 Краткие сведения о регулярных выражениях (РВ). Диалекты РВ.

Регулярные выражения (РВ) являются мощным средством **обработки текстов**. Ядром механизма работы регулярного выражения является **автомат-распознаватель**, позволяющий определить, принадлежит ли текст (фрагмент текста) запрашиваемому языку.

Абстрактный автомат-распознаватель задается как пятерка объектов:

$$A = \langle X, Q, q_0, \delta, F \rangle$$

где:

- X – множество входных символов автомата
- Q – множество состояний автомата
- q_0 – начальное состояние автомата
- δ – функция переходов $\delta: Q \times X \rightarrow Q$
- F – множество допускающих (конечных) состояний автомата, $F \subseteq Q$

Автомат **допускает** входную цепочку языка, если после ее обработки он оказывается в одном из своих допускающих состояний. Множество всех допускаемых автоматом цепочек образует язык, распознаваемый этим автоматом. Конечные автоматы могут распознавать только определенный класс языков, называемых **автоматными** или **регулярными** (к таким языкам не относятся, например, большинство языков программирования, а также естественные языки. В общем случае формальный способ определения, является ли язык автоматным, основывается на использовании **леммы о накачке** – см. основной конспект). Таким образом, автомат задает язык через способ его распознавания. Другим механизмом задания автомата является порождающая процедура, в качестве которой обычно выступает **порождающая грамматика** (задается множеством терминальных и нетерминальных символов и множеством правил – продукций, ставящих в соответствие одни символы другим – см. основной конспект лекций). Порождающая грамматика определяет способ формирования всех допустимых цепочек языка.

Регулярное выражение представляет собой текстовую формулу, являющуюся аналогом порождающей грамматики, в том смысле, что она также задает способ формирования допустимых цепочек языка. Язык описывается с помощью строки текста – сформулированного регулярного выражения. Поэтому можно считать язык регулярных выражений метаязыком описания других автоматных языков.

В теории автоматов и языков РВ определяются обычно через понятие **регулярных множеств**. **Регулярные множества (РМ)** – это множества, определяемые по индукции : регулярным является пустое множество, множество, состоящие из символов, а также множества, полученные из этих двух вариантов множеств в результате использования базовых операций над их элементами : соединения или конкатенации ($\cdot$), объединения ($+$) и итерации (повторения - $*$). **Регулярные выражения** – это конечные цепочки, задающие бесконечные регулярные множества. Они включают сами символы, конкатенацию ($\cdot$), обычно точку опускают), операцию объединения ($+$) и операцию итерации ($*$). Последняя операция имеет смысл повторения определенной части цепочки от 0 до бесконечного числа раз. В принципе этой минимальной нотации достаточно для представления всех РМ.

На практике, однако, как правило, используют более удобные нотации, включающие дополнительные обозначения, например, для диапазонов символов (типа '[A-Z]' и др.), для исключения конкретных символов или диапазонов и т.д.

Разных форм записей РВ в программных системах (диалектов РВ), к сожалению, довольно много, что связано с их параллельным историческим развитием. В описании лабораторной работы № 3 приводятся примеры двух нотаций, используемых в Microsoft Visual Studio.NET и в библиотеке классов .NET Framework.

1.2 Применение РВ в программировании

При программировании систем польза регулярных выражений незаменима во многих прикладных задачах. Например, контроль входных данных, трансляция данных из одного формата в другой, выборка необходимых данных из входного потока. Более кратко можно сказать: регулярные выражения позволяют с минимальными затратами управлять данными.

Распространенность систем, библиотек, языков с поддержкой механизма регулярных выражений позволяет применять их повсеместно. В число таких языков программирования входят: Java, JScript, Visual Basic, JavaScript, ECMAScript, C, C++, C#, ELisp, Perl, Python, TCL, Ruby, PHP, sed, awk и многие другие. К сожалению, разные системы программирования предлагают различающиеся метаязыки для описания регулярных выражений. Поэтому переносимость регулярных выражений в ряде случаев затруднена. Это связано как с историческими причинами параллельного развития этих систем, так и с механизмом распознавания в работе регулярного выражения. Разные способы записи в различных системах называют диалектами регулярного выражения.

Следует заметить, что регулярные выражения при поиске в Visual Studio .NET и регулярные выражения библиотеки RegEx.NET различаются. То есть это разные диалекты.

Существуют четыре основных подхода к разбору регулярного выражения и обрабатываемому тексту.

Тип механизма	Используемые системы
Детерминированный конечный автомат (ДКА)	Awk, egrep, flex, lex, MySQL, Procmail
Традиционный недетерминированный автомат (НКА)	GNU Emacs, Java, grep, less, more, библиотека RegEx.NET, библиотека PCRE, Perl, PHP, Python, Ruby, sed, vi
POSIX НКА	Mawk, Mortice Kern Systems utilites, GNU Emacs
Гибридные системы НКА/ДКА	GNU awk, GNU grep/egrep, Tcl

Если убрать исторические причины и механизмы оптимизации запросов по регулярным выражениям, то можно выделить два основных механизма работы с регулярным выражением. Это НКА и ДКА. Первый механизм ориентирован на управление регулярным выражением, а второй на управление обрабатываемым текстом.

НКА – это недетерминированный конечный автомат в противоположность детерминированному. Приведенные выше модели автоматов являются детерминированными. НКА отличается тем, что в нем :

1) возможен **переход** из текущего состояния при заданном входном символе не в одно следующее состояние, а **во множество состояний** параллельно;

2) допускается переход в другое состояние по пустой цепочке ϵ (**ϵ -НКА**).

НКА разбирает регулярное выражение и осуществляет необходимые проверки по тексту. Входными символами этого автомата являются символы регулярного выражения. Переходя в новое состояние, автомат производит необходимые проверки по обрабатываемому

тексту. Если проверки не удачны по всем возможным условиям, автомат пропускает часть символов для поиска метасимвола «ИЛИ». То есть задача механизма на основе НКА перебрать все возможные варианты соответствия. ДКА – типичный детерминированный автомат-распознаватель. Сначала строится на основе регулярного выражения сам автомат. Затем по входному символу текста этот автомат производит анализ. Соответственно, при разных условиях эти различные алгоритмы будут проявлять себя по-разному. Т.к. для ДКА сначала строится автомат, то при частых отличиях первых символов регулярного выражения с текстом становится эффективнее метод НКА. Если же количество выражений ИЛИ «|» в регулярном выражении велико и само выражение объемно, то эффективнее механизм на основе ДКА. В общем случае при составлении ДКА по РВ сначала может получиться НКА, который всегда можно преобразовать в эквивалентный ему ДКА (см. далее пример).

На прикладном уровне регулярные выражения могут быть использованы для следующих низкоуровневых операций:

- соответствие текста регулярному выражению (например, проверка пользовательского ввода),
- поиск в тексте подстроки/подстрок соответствующих регулярному выражению (например, в текстовых процессорах, поисковых системах, в БД, в лексических анализаторах и др.)
- замена подтекста соответствующего регулярному выражению,
- разделение текста на части по границам, соответствующим регулярному выражению (в системах обработки текстов, в лексических анализаторах и пр.).

2. Пример составления и использования регулярного выражения (можно пропустить и перейти к п. 3)

2.1 Пример формирования регулярного выражения

Рассмотрим прикладную практическую задачу поиска/замены IP адресов в тексте. Это может быть необходимо для осуществления выборки из введенного пользователем текста IP адресов, или, например, контроля правильности IP адреса в поле ввода диалоговой формы. Краткая таблица для описания диалекта библиотеки RegEx.NET приведена в описании лабораторной работы № 3.

IP адрес представляет собой четыре числа от 0 до 255, разделенных точками. Например: «1.2.3.4». Числа могут дополняться нулями до 3 разрядов: «001.002.003.004». Первый вариант проверки напрашивается сразу же:

`«[0-9]*\.[0-9]*\.[0-9]*\.[0-9]*»`.

Данное решение совпадает с многими строками, не являющимися IP адресами. Например: «Если так то ...», «...», «999.876.3456789.234890». Регулярное выражение требует наличия трех точек, возможно «разбавленных» символами десятичных цифр. Первое уточнение это обязательное наличие в регулярном выражении чисел между точками:

`«[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+»`.

Замена квантификатора «*» на «+» требует нахождения как минимум одного символа десятичного числа между точками внутри текста. Строки вида «...» отпадают. Чтобы вся строка представляла запись IP адреса требуется привязка регулярного выражения к виду строки, т.е. указанию что строка должна состоять только из структуры IP адреса:

`«^[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+$»`.

Для большей наглядности можно заменить регулярное выражение выше более компактным метасимволом, означающим запись десятичной цифры. Для диалекта среды Visual Studio 2003: это будет выглядеть так:

`«^:z+\.:z+\.:z+\.:z+$»` (а для диалекта библиотеки RegEx.NET: `«^\d+\.\d+\.\d+\.\d+$»`)

Это выражение по-прежнему совпадает со многими записями, не являющимися IP адресами. Например, «12.01.1994.10».

Каждое число IP адреса должно находиться в диапазоне 0-255, т.е. требует записи максимально 3 знаков. Введем следующее выражение:

«^:z:z:z\.:z:z:z\.:z:z:z\.:z:z:z\$» («^d\d\d\d\d\d\d\d\d\d\d\d\d\$»)

Новое регулярное выражение с одной стороны слишком жестко – адреса в виде «127.0.0.1» не будут совпадать с ним. Требуется указывать «127.000.000.001». С другой стороны конструкции в виде «999.885.456.794» тоже будут улавливаться.

Сделать выражение гибче можно за счет введения интервального квантификатора.

К сожалению не все диалекты поддерживают его. Так, например, библиотека RegEx.NET поддерживает, а среда программирования Visual Studio 2003 .NET – нет. Для библиотеки RegEx.NET данное выражение будет выглядеть так:

«^d{1,3}\.d{1,3}\.d{1,3}\.d{1,3}\$»

Выражение с интервальным квантификатором делает поиск гибче, но все еще не все найденные строки будут являться IP адресом. Причина кроется в диапазоне допустимых чисел (0-255). Чтобы контролировать их необходимо разбить возможные числа на группы:

1. число, состоящее из одного десятичного разряда;
2. число, состоящее из двух десятичных разрядов;
3. число, состоящее из трех десятичных разрядов.

Для первой и второй группы подходит полный диапазон десятичных цифр, а вот для третьей группы – будут ограничения.

Если для первой цифры используется десятичная цифра «2», то нужно проанализировать следующие разряды. Если вторая десятичная цифра – в диапазоне от «0» до «4», то третья не имеет ограничений. Если она «5», то на третье десятичное число накладывается ограничение. Оно может быть только в диапазоне от «0» до «5». Остальные варианты недопустимы.

Если первая десятичная цифра в трехразрядном числе «0» или «1», то ограничений на второе и третье число не накладывается.

Формулируя в регулярном выражении то, что мы описали выше, получаем выражение для задания чисел в диапазоне от 0 до 255, возможно, с ведущими нулями:

«:z(:z:z)(2[0-4]:z)(25[0-5])([01]:z:z)» («d(\d\d)(2[0-4]\d)(25[0-5])([01]\d\d)»)

Указанное выражение можно упростить, объединяя несколько вариантов «или» в один.

Если выполняется одно из следующих условий:

1. число начинается с нуля или единицы;
2. оно имеет один или два разряда,

то неучтенные в регулярном выражении цифры могут содержать полный диапазон десятичных чисел. Для этого используем квантификатор необязательного символа и, возможно, интервальный квантификатор. Следующие регулярные выражения идентичны по результату:

«[01]?d\d?» или «[01]?d{1-2}»

Данное условие будет удовлетворять числам от 0 до 199, записанным, возможно, с начальными нулями.

Если число начинается с 2, то следующие символы должны ограничивать определенный диапазон. Так для чисел от 200 до 249 подходит регулярное выражение «2[0-4]:z». А для чисел от 250 до 255 – «25[0-5]». Получаем следующее регулярное выражение:

«([01]?d{0-2})(2[0-4]\d)(25[0-5])»

Оно проверяет соответствие чисел диапазону от 0 до 255. Другие варианты того же ограничения:

«([01]?d\d?)(2[0-4]\d)(25[0-5])»

«([01]?d\d?)(2([0-4]\d)(5[0-5]))»

Учитывая все вышесказанное, полный вариант регулярного выражения может выглядеть, например, так:

«^([01]?d\d?)(2[0-4]\d)(25[0-5])\.([01]?d\d?)(2[0-4]\d)(25[0-5])\.([01]?d\d?)(2[0-4]\d)(25[0-5])\.([01]?d\d?)(2[0-4]\d)(25[0-5])\$»

Но стоит ли строить такое сложное выражение ? Существует ли другие варианты поиска IP адресов? Регулярное выражение должно содержать сложные условия в том случае, когда дополнительной обработки другими методами невозможно предпринять. Например, конечный пользователь не согласен отфильтровывать лишние значения вручную. В случае же если после поиска по регулярному выражению можно произвести дополнительный анализ средствами алгоритмического языка следует использовать более простые варианты. Для IP адреса это вариант:

«^d+\.d+\.d+\.d+\$»

2.2 Пример работы с регулярным выражением для контроля вводимых пользователем IP адресов.

При построении программных систем разумно предварительный фильтр на основе регулярного выражения обработать дополнительным алгоритмом. Рассмотрим этот способ для проверки введенного IP адреса с помощью языка C# и библиотеки RegEx.NET.

В примере создается форма Windows.Forms с полем для ввода IP адреса. Сам ввод не контролируется, но при нажатии кнопки «Check» происходит проверка с выдачей сообщения о корректности введенного пользователем IP адреса.

Ниже приводится полный текст программы, включая сгенерированный средой код для создания формы ввода (в среде она создается графическим редактором). Собственно код, реализующий проверку с помощью PB и уточнение дополнительной проверкой, совмещен в обработчике Check_click (обведен в рамку).

```
using System;
using System.Windows.Forms;
using System.Text.RegularExpressions;

namespace IPCheck
{
    ///Форма для ввода IP адреса
    public class MainForm : System.Windows.Forms.Form
    {
        ///Контролируемое поле ввода
        private System.Windows.Forms.TextBox IP;
        ///Кнопка проверки
        private System.Windows.Forms.Button Check;

        /// Required designer variable.
        private System.ComponentModel.Container components = null;

        public MainForm()
        {
            // Required for Windows Form Designer support
            InitializeComponent();
        }

        /// Clean up any resources being used.
        protected override void Dispose( bool disposing )
        {
            if( disposing )
            {
                if (components != null)
                {
                    components.Dispose();
                }
            }
        }
    }
}
```

```

    }
    }
    base.Dispose( disposing );
}

#region Windows Form Designer generated code
/// Required method for Designer support - do not modify
/// the contents of this method with the code editor.
private void InitializeComponent()
{
    this.IP = new System.Windows.Forms.TextBox();
    this.Check = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // IP
    //
    this.IP.Location = new System.Drawing.Point(8, 8);
    this.IP.Name = "IP";
    this.IP.Size = new System.Drawing.Size(184, 20);
    this.IP.TabIndex = 2;
    this.IP.Text = "0.0.0.0";
    //
    // Check
    //
    this.Check.Location =
        new System.Drawing.Point(208, 8);
    this.Check.Name = "Check";
    this.Check.TabIndex = 1;
    this.Check.Text = "Check";
    this.Check.Click += new
        System.EventHandler(this.Check_Click);
    //
    // MainForm
    //
    this.AutoScaleBaseSize =
        new System.Drawing.Size(5, 13);
    this.ClientSize = new System.Drawing.Size(292, 38);
    this.Controls.AddRange(new
        System.Windows.Forms.Control[]
            {this.Check, this.IP});
    this.Name = "MainForm";
    this.Text = "Regular Expression";
    this.ResumeLayout(false);
}
#endregion

/// The main entry point for the application.
[STAThread]
static void Main()
{
    Application.Run(new MainForm());
}

```

```

//Проверка введенного значения IP адреса
private void Check_Click(object sender,
    System.EventArgs e)
{
    //Используем статическую функцию объекта Regex
    Match match =
        Regex.Match(IP.Text, @"^(\d+)\. (\d+)\. (\d+)\. (\d+)$");
    bool success=match.Success;
    // Проверяем совпавшие подстроки на предмет

```

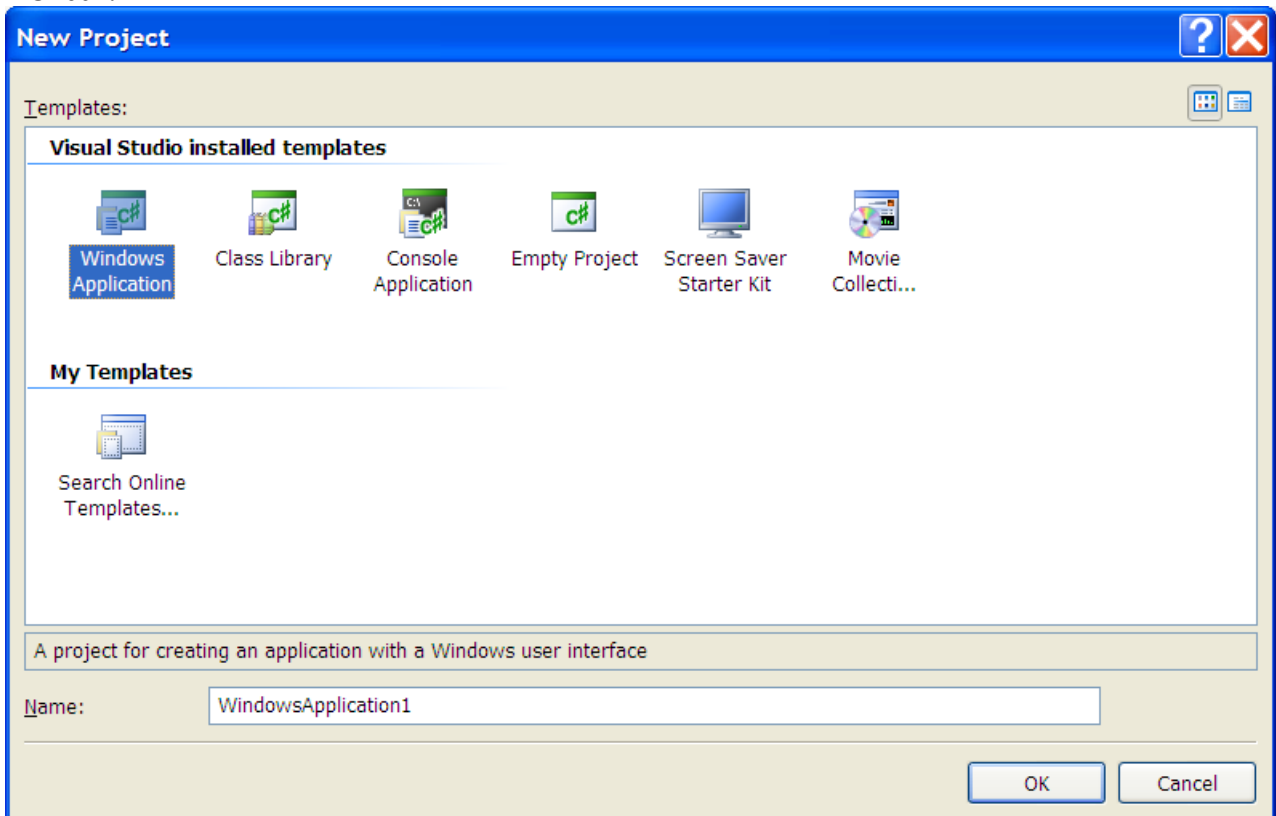
```

        // принадлежности
        // к текстовому представлению числа в диапазоне
        // [0-255]
        for (int i=1; success&& i<5; i++)
        {
            try
            {
                System.Convert.ToByte(match.Groups[i].Value);
            }
            catch (System.Exception)
            {
                success=false;
            }
        }
        MessageBox.Show("IP address '"+IP.Text+"'
            is "+success);
    }
}
}

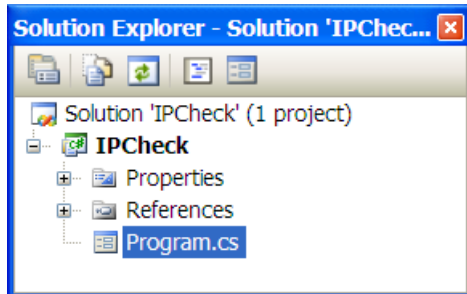
```

Чтобы проверить работоспособность этого примера, создать свои примеры, проверить работу поиска по текстовым файлам в системе Visual Studio 2005 Express, необходимо сначала создать новый проект.

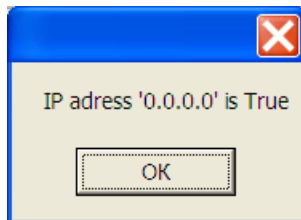
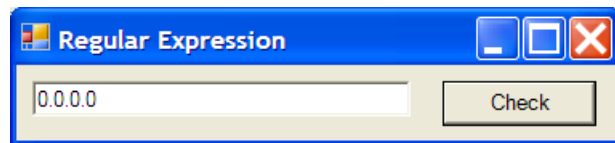
Для этого запускаем среду, выбираем пункт меню **File -> New Project**. Открывается окно Мастера создания проектов, в котором предлагается выбрать тип шаблона проекта и указать его имя (папка проекта будет создана внутри папки проектов по умолчанию, которая указана в параметрах среды (меню **Tools -> Options -> Projects and Solutions**)). Для создания Windows приложения с графической формой ввода выбираем шаблон по умолчанию Windows Application. Имя проекта можно задать любое, например IPCheck.



Система создаст проект приложения Windows с графическим интерфейсом пользователя (одной формой по умолчанию). Так как у нас уже есть текст программы с параметрами формы, можно удалить из проекта форму (всю ветку **Form1.cs** вместе с подпунктами), а также созданный по умолчанию код внутри модуля **Program.cs**. В модуль **Program.cs** необходимо поместить приведенный выше текст примера.

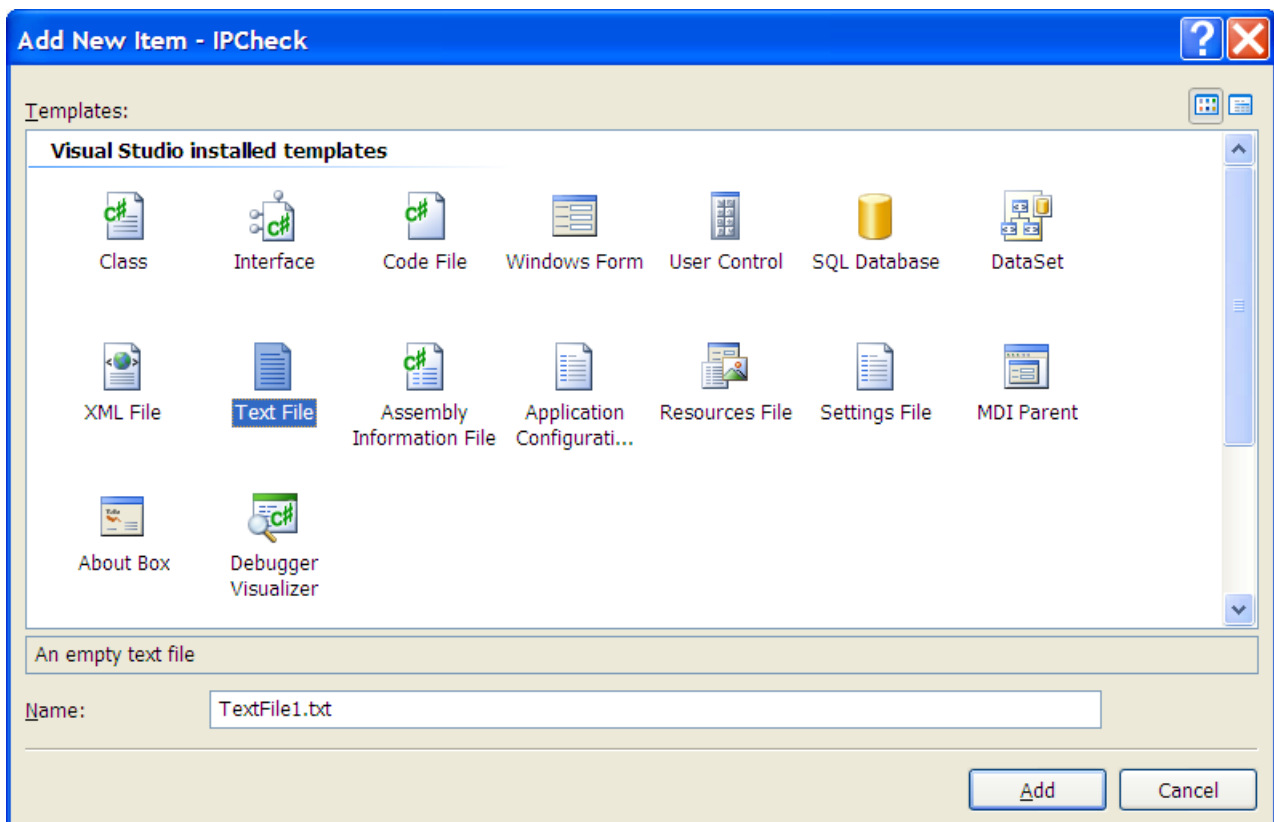


После этого выполните построение проекта (меню **Build -> Build Solution** или **F6**) и запустить программу (меню **Start -> Start Without Debugging** или **Ctrl + F5**). Откроется небольшое диалоговое окно, в котором можно ввести IP-адрес и проверить его на правильность (нажав кнопку **Check**).



Для проверки возможностей поиска по тексту файлов проекта с помощью РВ можно создать текстовый файл и включить его в проект (контекстное меню проекта в окне **Solution Explorer -> IPCheck -> Add ->NewItem -> Text File**).

Затем можно набрать в тексте несколько образцов текста и попробовать выполнить поиск с помощью РВ (пункт меню **Edit -> Find and Replace -> Find in Files**).



3. Синтез детерминированного автомата для распознавания языка, задаваемого регулярным выражением, и его программная реализация. (!!!)

Рассмотрим пример получения РВ для задания языка, синтеза НКА для распознавания задаваемым этим РВ языка, получения для него ДКА и его программной реализации. Для проверки правильности полученного ДКА преобразуем его опять в РВ и сравним с исходным.

В качестве примера рассмотрим РВ для распознавания целых десятичных чисел со знаком без ведущих нулей. Считаем, что плюс не задается (то есть, если знак не задан, то число положительное).

Будем использовать стандартную минимальную нотацию РВ, в которой используются символы алфавита $V : \{ '0', '1', \dots, '9', '-' \}$, круглые скобки для задания приоритетов, операции '+', конкатенация и итерация '*'. Дополнительно будем использовать квадратные скобки для задания отдельного символа или диапазона символов. То есть любая цифра от 0 до 9 задается как '[0–9]', а минус задается как '-' или '[-]'. Для выделения отдельных частей РВ будем использовать пробелы, а сам пробел при необходимости (хотя в данном примере он не нужен) будем также задавать в квадратных скобках : '[]'.

Тогда РВ R, задающее целые числа в десятичной системе без ведущих нулей :

$$R = 0 + [1-9][0-9]^* .$$

Если учесть, что числа, отличные от нуля, могут быть отрицательными, тогда :

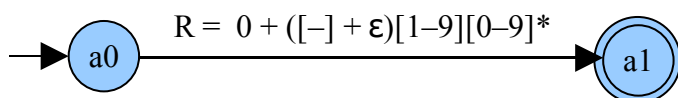
$$R = 0 + ([-] + \epsilon)[1-9][0-9]^* ,$$

где ϵ – пустая цепочка.

3.1 Преобразование РВ в НКА с ϵ – переходами

Для преобразования РВ в КА нужно воспользоваться теоремой Клини (вернее, ее доказательством – см. основной конспект). Мы последовательно будем заменять части РВ вершинами и дугами автомата, преобразуя граф переходов до тех пор, пока дуги не будут отмечены только символами алфавита V или ϵ .

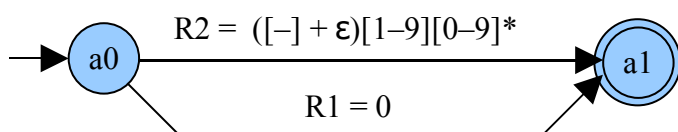
Исходный граф переходов :



Представляя исходное выражение как сумму РВ :

$$R = R1 + R2 = 0 + ([-] + \epsilon)[1-9][0-9]^* ,$$

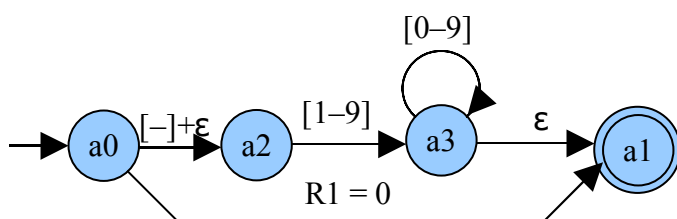
получим :



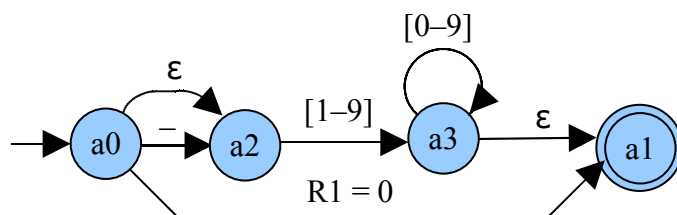
Преобразуя R2 :

$$R2 = ([-] + \epsilon)[1-9][0-9]^* = R3R4R5^* ,$$

получаем :



и окончательно получаем :



3.2 Преобразование НКА с ε-переходами в НКА без ε-переходов

ε-замыканиями для полученного автомата (то есть подмножествами состояний, в которые может перейти из каждого состояния автомата по пустой цепочке – см. основной конспект) будут следующие подмножества состояний :

$$Q(a_0) = \{a_0, a_2\}; Q(a_2) = \{a_2\}; Q(a_3) = \{a_1, a_3\}; Q(a_1) = \{a_1\}.$$

Теперь составим таблицу переходов между этими замыканиями ($\emptyset$ – пустое мн.) :

	$b_0 = \{a_0, a_2\}$	$b_1 = \{a_2\}$	$b_2 = \{a_1, a_3\} +$	$b_3 = \{a_1\} +$
–	$\{a_2\}$	$\emptyset$	$\emptyset$	$\emptyset$
$[1-9]$	$\{a_1, a_3\}$	$\{a_1, a_3\}$	$\{a_1, a_3\}$	$\emptyset$
0	$\{a_1\}$	$\emptyset$	$\{a_1, a_3\}$	$\emptyset$

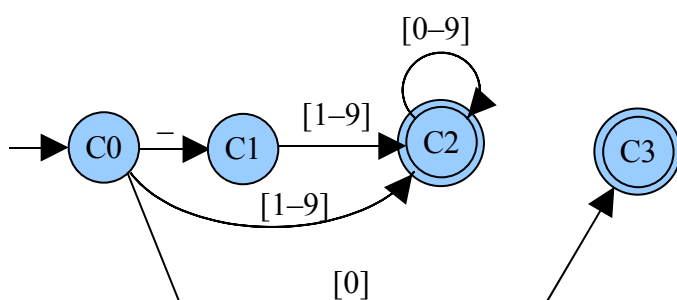
Мы получили НКА без **ε**-переходов (это по-прежнему НКА, так как у него 2 начальных состояния b_0 и b_1). Плюсами отмечены заключительные состояния.

3.3 Получение ДКА по НКА без ε-переходов

Составим еще одну таблицу, учитывая, что начальное состояние нашего автомата ДКА $c_0 = \{b_0, b_1\}$. Следующие состояния определяем по таблице переходов выше :

	$c_0 = \{b_0, b_1\}$	$c_1 = \{b_1\}$	$c_2 = \{b_2\} +$	$c_3 = \{b_3\} +$
–	$\{b_1\}$	$\emptyset$	$\emptyset$	$\emptyset$
$[1-9]$	$\{b_2\}$	$\{b_2\}$	$\{b_2\}$	$\emptyset$
0	$\{b_3\}$	$\emptyset$	$\{b_2\}$	$\emptyset$

Таким образом мы получили ДКА с состояниями $c_0 - c_3$.



3.4 Минимизация ДКА

В данном случае минимизация ДКА (см. основной конспект) невозможна, но это неочевидно, поэтому проведем минимизацию формально. Для этого получим нулевое разбиение состояний, состоящее из 2 подмножеств состояний : заключительные c_2 и c_3 и остальные c_0 и c_1 :

$$A_0 = \{c_0, c_1\}; B_0 = \{c_2, c_3\}.$$

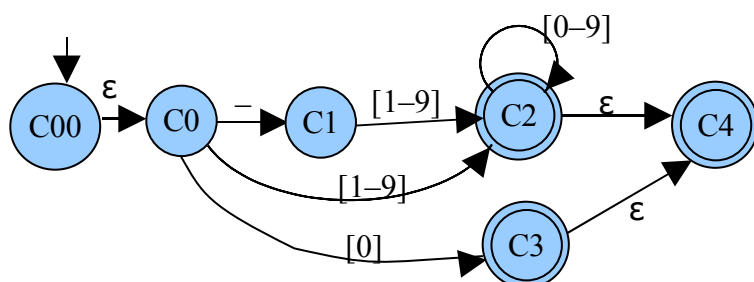
Составим таблицу переходов по исходной таблице, но в ячейках укажем блоки разбиения :

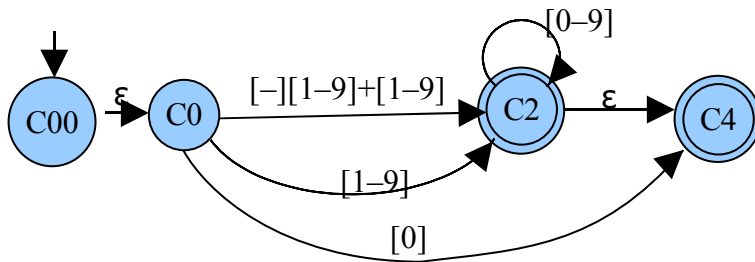
	c_0	c_1	c_2+	c_3+
$-$	A_0	$\emptyset$	$\emptyset$	$\emptyset$
$[1-9]$	B_0	B_0	B_0	$\emptyset$
0	B_0	$\emptyset$	B_0	$\emptyset$

Из этой таблицы видно, что следующее разбиение будет содержать блоки, состоящие каждый из одного состояния, то есть автомат не минимизируется.

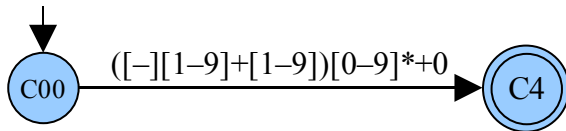
3.5 Получение РВ по КА

Выполним проверку полученного ДКА, построив по нему РВ по теореме Клини. Для этого сначала преобразуем автомат в нормализованную форму, а затем выполним последовательно редукцию вершин и ребер :





в результате которых получим граф



и РВ

$$R' = 0 + ((-)[1-9] + [1-9])[1-9]^*.$$

После несложных преобразований РВ получаем :

$$R' = 0 + ((-) + \epsilon)[1-9][0-9]^* = R.$$

Таким образом, мы получили исходное РВ ! Это позволяет судить о том, что полученный ДКА верный. Несложно видеть, что можно было сразу получить ДКА, если бы мы остановились на РВ R' в качестве первоначального. Но, во-первых, мы хотели показать, что в любом случае, при любом РВ можно получить для него ДКА. Во-вторых, в общем случае РВ могут быть более сложными и их преобразования тоже. При этом все приведенные этапы преобразования можно автоматизировать.

3.6 Программная реализация ДКА-распознавателя

Программная реализация ДКА-р может быть различной, но наиболее традиционен вариант просто использования вложенных управляющих конструкций типа if ... else if ... или оператора выбора. Чтобы упростить восприятие соответствующих конструкций можно реализовать отдельную функцию для проверки принадлежности символов входного потока диапазону [0-9].

Вариант программы на C# (вернее, фрагмента, реализующего автомат) приведен далее. Здесь необходимо отметить, что в автомат введено дополнительное тупиковое состояние c4, в которое автомат попадает из всех состояний по неразрешенным символам (в частности, по любому символу из c3).

```

public enum States { c0, c1, c2, c3, c4 };

private bool checkFSM(string Src)
{
    int nPos = 0;
    States st = States.c0;

    string Digits = "123456789";

    while (nPos < Src.Length) {

```

```

// Собственно сам автомат ...
switch (st)
{
    case States.c0:
        if (Src[nPos] == '-')
            st = States.c1;
        else if (Src[nPos] == '0')
            st = States.c3;
        else if (Digits.IndexOf(Src[nPos]) >= 0)
            st = States.c2;
        else
            st = States.c4;
        break;
    case States.c1:
        if (Digits.IndexOf(Src[nPos]) >= 0)
            st = States.c2;
        else
            st = States.c4;
        break;
    case States.c2:
        if (Src[nPos] == '0' ||
            (Digits.IndexOf(Src[nPos]) >= 0))
            st = States.c2;
        else
            st = States.c4;
        break;
    case States.c3:
        st = States.c4;
        break;
}
nPos++;
}
return st == States.c2 || st == States.c3;
}

```

4. Контрольные вопросы :

- 4.1 Что такое регулярное выражение ?
- 4.2 Где используются РВ ?
- 4.3 Какие Вы знаете способы задания РВ ?
- 4.4 С помощью каких автоматов распознаются языки, задаваемые РВ ?
- 4.5 Что такое НКА ? ДКА ?
- 4.6 Как построить по РВ КА – распознаватель ?
- 4.7 Как построить по РВ ДКА – распознаватель ?
- 4.8 Как устранить ε-переходы в КА ?
- 4.9 Как минимизировать КА – распознаватель ?
- 4.10 Как используются РВ в среде VS ?
- 4.11 Как РВ поддерживаются в .NET Framework ?
- 4.12 Опишите заданные цепочки с помощью РВ.
- 4.13 Какие цепочки задает данное РВ (примеры, характеристика).
- 4.14 Стратегии реализации поддержки РВ в программных системах.
- 4.15 Способы использования поддержки РВ при составлении программ обработки текстов.
- 4.16 Какие задачи обработки текстов решаются с помощью РВ ?
- 4.17 Перечислите известные Вам программные системы, поддерживающие РВ.