Астраханский Государственный Технический Университет

Кафедра «Автоматизированные системы обработки информации и управления»

Курсовая работа
по дисциплине
"Объектно-ориентированное программирование"

по специальности 220200

"Автоматизированные системы обработки информации и управления"

Выполнили
Морозов А.В.
Спандерашвили Д.В.
Алтуфьев М.Ю.

Проверил:
к.т.н., доц. Лаптев В.В.

АСТРАХАНЬ – 2001 г.

Оглавление

1. ВВЕДЕНИЕ

«К чему бы все это, лучше бы водочки выпили».

Из письма Гоголя Белинскому

«Куплю IBM»

объявление на столбе

«Каждая задача имеет простое, элегантное и … неправильное решение»

эпиграф из книги Дэйтла

«И да сэмулируется неэмулируемое»

Спандерашвили Д.В. «Метаморфозы»

Задачей данного программного продукта является предоставление инструментов необходимых для моделирования работы процессора. Существуют другие программы, преследующие подобные цели, но они либо рассматривают работу микропроцессора как целостного элемента с различными уровнями открытости внутренней организации, либо схемотехническую организацию его отдельных элементов (таких как триггеры, сумматоры, регистры и т.д.). Недостатком первого вида программных продуктов является, то, что они:

- Привязаны к определенной архитектуре, либо шаблону архитектуры.
- Не позволяют рассмотреть, или переопределить работу отдельных элементов процессора.
- Не уделяют внимание внутри процессорному представлению информации.

Второй вид программ напротив:

- Рассматривает физическую организацию отдельных элементов процессора, не позволяя рассмотреть работу в комплексе.
- Не позволяет моделировать работу отдельных элементов процессора в независимости от их физической структуры.
- Чаще всего имеет ограниченный набор базовых элементов.

В результате проведенной нами работы, мы пришли к выводу о необходимости совмещения основных достоинств этих двух подходов и нивелированию их недостатков. Для решения этой задачи мы предлагаем использовать несколько иной подход к моделированию работы процессора и составных частей.

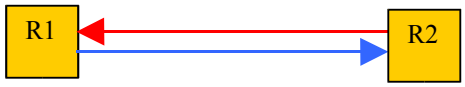
Для того чтобы обойти выше перечисленные недостатки мы должны рассматривать внутреннюю организацию вычислений в процессоре, но абстрактно от их физической реализации. Мы предлагаем использовать не схемотехническую модель процессора, а обобщенную логико-математическую, рассматривающую функциональные свойства элемента отвлеченно от его технической реализации.

2. ТЕОРИЯ

1. Общие понятия.

Основным постулатом, на котором строится и без которого не возможно построение подобной программы, является следующее утверждение:

> «Любой процесс обработки, хранения и передачи информации может быть представлен как обмен информацией между **запоминающими ячейками** (как особый вид логического элемента) посредством **передаточных функций**».



(из постановления XXII съезда MSA corporation).

Действительно любой вычислительный процесс можно свести к обмену информацией между запоминающими ячейками посредством передаточных функций. Притом передаточная функция есть линейный элемент, осуществляющий передачу и преобразование информации за конечный промежуток времени, а ячейка – нелинейный (с обратной связью), хранящий информацию. Рассмотрим определения составляющих частей элементарного вычислительного процесса.

- *Запоминающая ячейка* – элемент, имеющий n устойчивых состояний, способный перманентно находиться в одном из них, а при поступлении внешнего воздействия переходить в соответствующее ему устойчивое состояние. Множество запоминающих ячеек образует *пространство запоминающих ячеек.* Под *внутри процессорным пространством запоминающих ячеек* (далее ВПЗЯ) будем понимать совокупность ячеек логически инкапсулированных в процессорную модель.
- *Передаточная функция* – линейный элемент, имеющий фиксированное (конечное) количество входов и выходов, и изменяющий значения на выходах в зависимости от значений поданных на входы.

- *Микропрограмма* – нелинейный элемент, который фактически представляет собой регистры – массивы запоминающих ячеек – связанные посредством передаточных функций в единые преобразователи информации. Микропрограмма организуется посредством суперграфа. Вызов микропрограммы производится командно.

2. Реализация n-ричности.

Для предоставления более широких возможностей организации модели процессора нами было решено допустить построение модели, работа которой основана не на двоичной системе счисления. Т.е. допустить организацию процессора с n-ричной логикой. Для доказательства возможности организовать такую модель в пределах нашей программы мы использовали теорию полей Галуа.

Вещественные числа образуют известное множество математических объектов, которые можно складывать, умножать, вычитать и делить. Аналогично комплексные числа образуют множество объектов, которые можно складывать, вычитать, умножать и делить. Обе эти арифметические системы являются важнейшей основой всех инженерных дисциплин. Нам необходимо построить другие, менее известные арифметические системы, полезные при моделировании вычислительной систему. Такие новые арифметические системы состоят из множеств и операций над элементами этих множеств. Хотя мы будем, называть операции "сложением", "вычитанием", "умножением" и "делением", они не обязательно будут теми же операциями, что в элементарной арифметике.

Изучаемые в современной алгебре арифметические системы классифицируются в соответствии с усложнением их математической структуры. Такая формальная классификация будет приведена ниже. Пока мы дадим следующие неформальные определения:

1) абелева группа — множество математических объектов, которые можно "складывать" и "вычитать";
2) кольцо — множество математических объектов, которые можно "складывать", "вычитать" и "умножать";
3) поле — множество математических объектов, которые можно "складывать", "вычитать", "умножать" и "делить".

Заметим, что названия этих операций взяты в кавычки потому, что, вообще говоря, они не являются принятыми арифметическими операциями; эти названия употребляются из-за их сходства с принятыми.

Прежде чем переходить к формальным понятиям, выполним некоторые вычисления в простейшем из всех возможных полей, а именно в поле, состоящем только из двух элементов. (Поле вещественных чисел содержит бесконечное число элементов.) Обозначим через 0 и 1 два элемента поля и определим операции сложения и умножения равенствами.

+	0	1
0	0	1
1	1	1

*	0	1
0	0	0
1	0	1

Так определенные сложение и умножение называются сложением по модулю 2 и умножением по модулю 2. Используя это, легко проверить, что, за исключением деления на нуль, вычитание и деление всегда определены. Алфавит из двух символов 0 и 1 вместе со сложением по модулю 2 и умножением по модулю 2 называется полем из двух элементов и по приведенным в гл. 4 соображениям обозначается через GF (2).В указанной арифметической системе можно осуществлять известные алгебраические операции.

В нашей программе необходимо задать несколько базовых операций, через которые должны выражаться все остальные. Для удобства операции задаются таблицей с предопределенными входами и задаваемыми выходами. Из теории Галуа следует, что для любой системы счисления все операции могут быть выражены через обобщенные

операции сложения и умножения (в двоичной системе счисления это операции XOR и AND). Причем в системе счисления, основанием которой является простое число, эти операции определены однозначно, а в системе, основание которой не простое число, может быть найдено несколько таких операций сложения и умножения. Для системы с простым основанием значения операций получаются следующим образом:

A(*)B=(A*B)MOD N

A(+)B=(A+B)MOD N

Где:

+ - сложение в десятичной системе счисления

* - умножение в десятичной системе счисления

(*) - умножение в N-ричной системе счисления

(+) - сложение в N-ричной системе счисления

N - основание системы счисления

Приведем в качестве примера системы счисления с не простым числом в качестве основания 16-ричную систему. Операции сложения и умножения определяются следующим образом.

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
2	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
3	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
4	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
5	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
7	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
B	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

+	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
2	0	2	4	6	8	A	C	E	3	1	7	5	B	9	F	D
3	0	3	6	5	C	F	A	9	B	8	D	E	7	4	1	2
4	0	4	8	C	3	7	B	F	6	2	E	A	5	1	D	9
5	0	5	A	F	7	2	D	8	E	B	4	1	9	C	3	6
6	0	6	C	A	B	D	7	1	5	3	9	F	E	8	2	4
7	0	7	E	D	F	8	1	6	D	A	3	4	2	5	C	B
8	0	8	3	B	6	E	5	D	C	4	F	7	A	2	9	1
9	0	9	1	8	2	B	3	A	4	D	5	C	6	F	7	E

A	0	A	7	D	E	4	9	3	F	5	8	2	1	B	6	C
B	0	B	5	E	A	1	F	4	7	C	2	9	D	6	8	3
C	0	C	B	7	5	9	E	2	A	6	1	D	F	3	4	8
D	0	D	9	4	1	C	8	5	2	F	B	6	3	E	A	7
E	0	E	F	1	D	3	2	C	9	7	6	8	4	A	B	5
F	0	F	D	2	9	6	4	B	1	E	C	3	8	7	5	4

В общем случае пользователь может выбрать в качестве базовых и не операции сложения и умножений, а любые другие (так в двоичной системе счисления можно выбрать в качестве базовых и AND,OR и NOT или построить все на дешифраторах).

3. Регистры

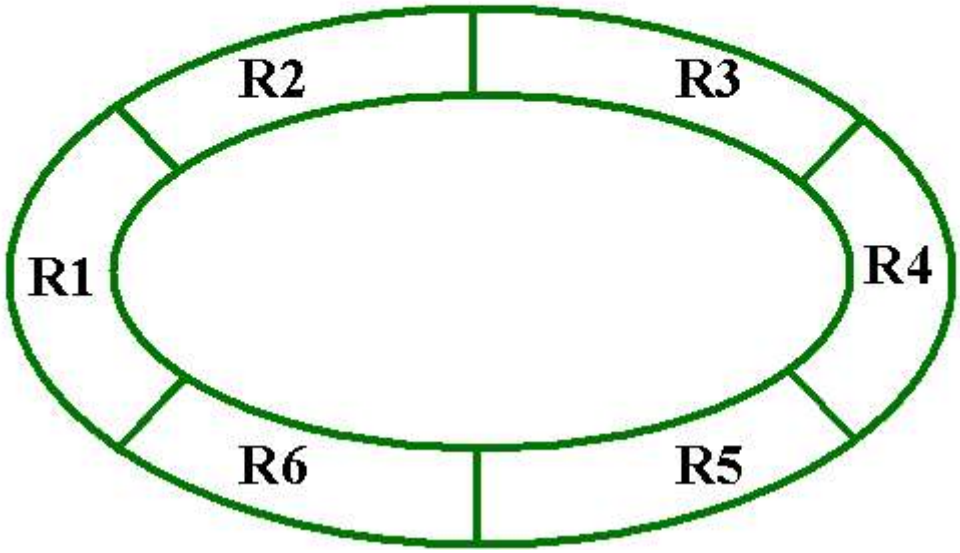
*«Отдельные запоминающие ячейки могут образовывать **регистры**. Основной признак и операция над регистром – **сдвиг**. Порядок сдвига в регистре может быть описан ориентированным графом с определенным порядком прохождения ребер. Сдвиг не может быть бесконечным циклическим».*

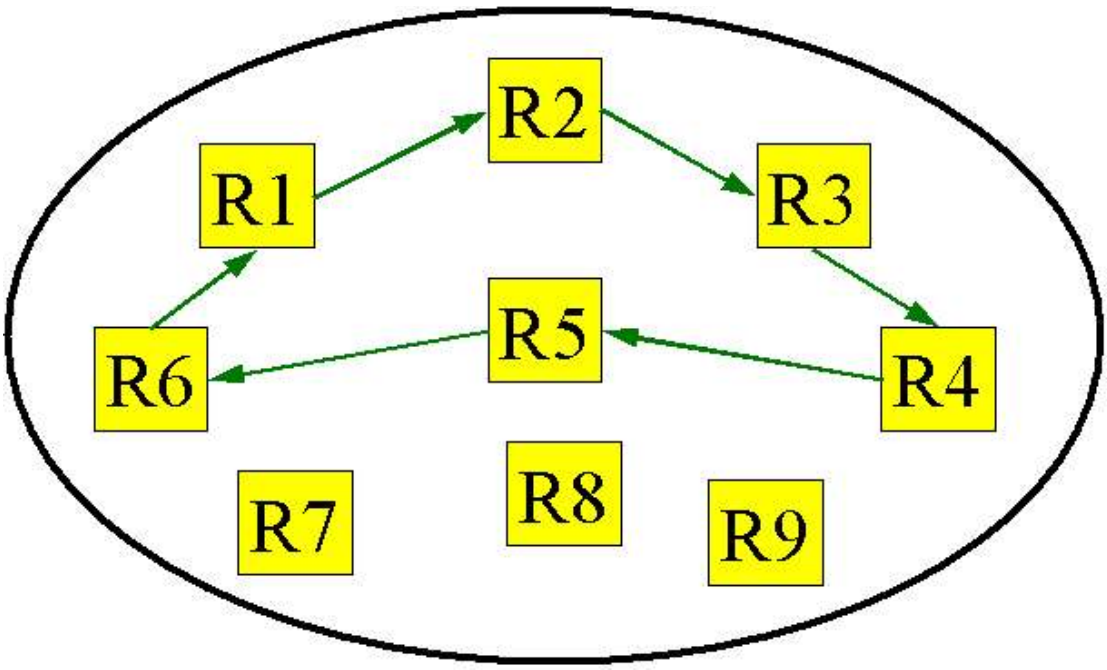
Морозов А.В. «размышления о природе вещей»

Регистры можно задавать путем перечисления запоминающих ячеек или ориентированным графом с определенным порядком прохождения ребер. В последнем случае можно конструировать нелинейные регистры любой сложности. *Нелинейные регистры* включают в себя следующие возможности:

- одна и та же запоминающая ячейка может встречаться в регистре несколько раз, таким образом, возможно образование «петель».

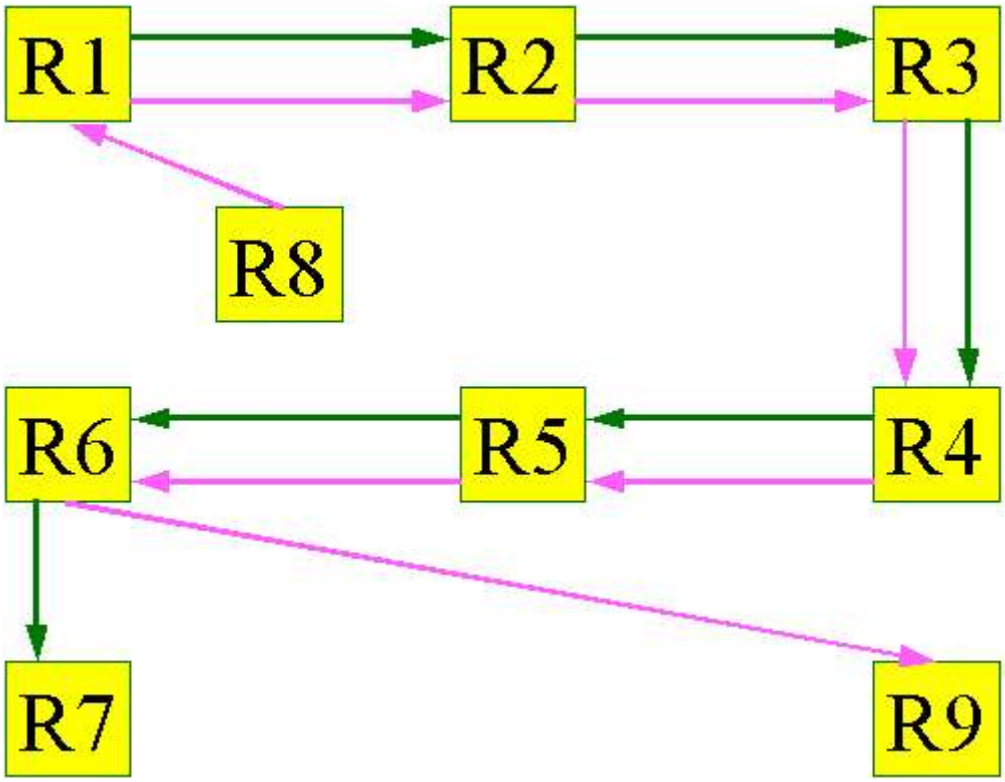
Если рассмотреть длину регистра на рисунке, то получается следующая ситуация: длина регистра – 7, притом, что в него входит лишь 6 ячеек. Это можно отобразить и на графе:





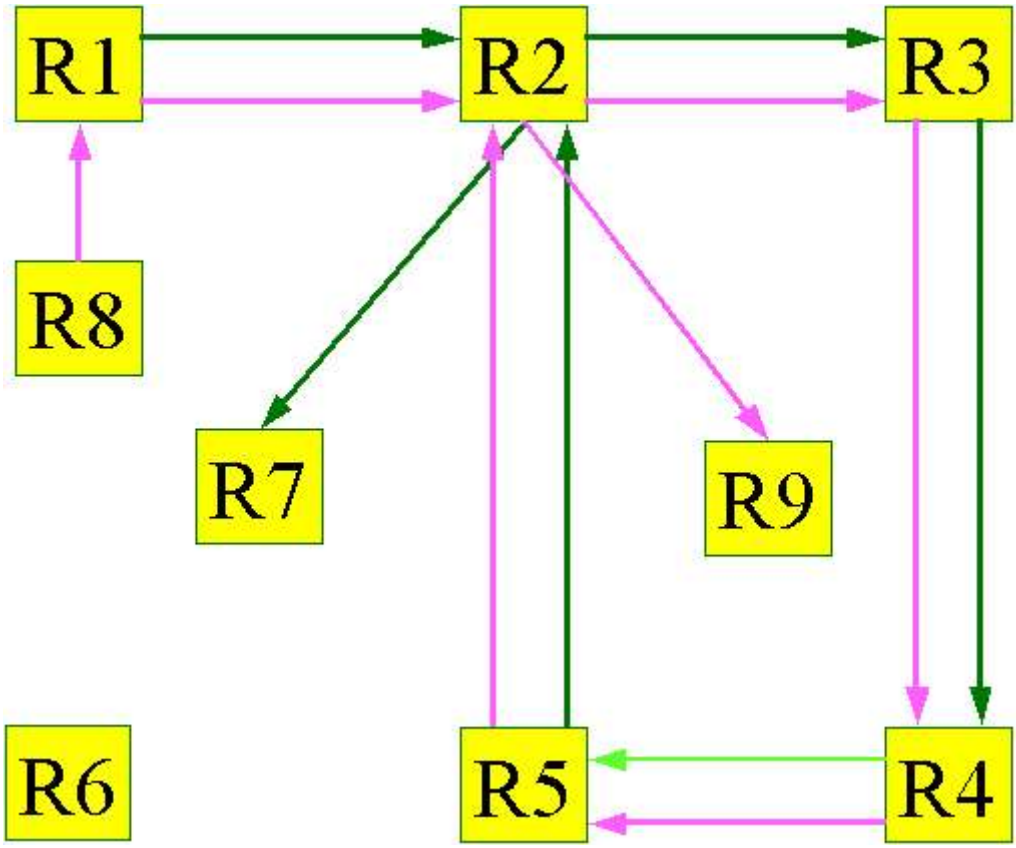
- одна и та же запоминающая ячейка может встречаться одновременно в нескольких регистрах.

Если отобразить это на графе:



Возможно совмещение этих двух возможностей: т.е. в одном регистре могут встречаться одинаковые запоминающие ячейки, и одновременно с этим другой регистр будет содержать запоминающие ячейки, встречающиеся в первом.

Или на графе:



Здесь мы не привязываемся к физической реализации регистров, т.к. наша цель дать возможность организовывать математические модели. Физическая реализация должна быть следствием тщательно проведенного анализа и выбора наиболее эффективного решения для конкретной задачи. Данный подход дает широкие возможности организации самых разнообразных регистров. Многократное включение ячейки в регистр позволяет, проводя сдвиг, **учитывать** содержимое нескольких отдельных ячеек при формировании результата в одной из них.

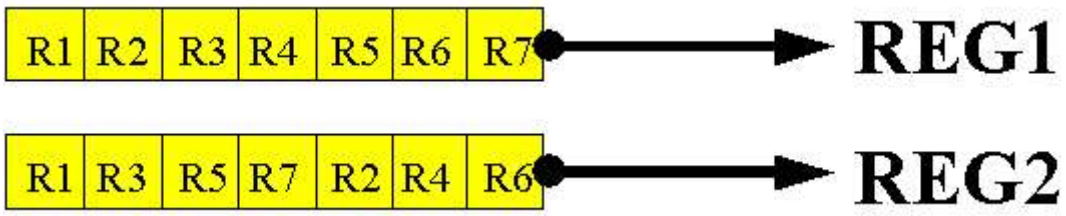
Например, возьмем регистр и проиллюстрируем пошаговое проведение сдвига вправо.

r1	r2	r3	r2	r4	r5	r6	r1
1	0	1	0	0	1	0	1
0	0	1	0	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	0
0	1	1	1	0	0	1	0
0	1	1	1	0	0	1	0
0	0	1	0	0	0	1	0

Из таблицы видно как по мере проведения сдвига содержимое некоторых запоминающих ячеек меняется несколько раз. Содержимое ячеек как бы перескакивает из одной ячейки в другую, но на самом деле перескок наблюдается только при рассмотрении регистра в линейном (векторном) виде, если рассматривать его на графе, то никакого перескока не происходит, – происходит лишь последовательный обмен содержимым запоминающих ячеек согласно направлению проведения сдвига.

Данный подход к организации регистров может использоваться, например, когда требуется несколько различных представлений одного регистра.

Пусть имеется два регистра:

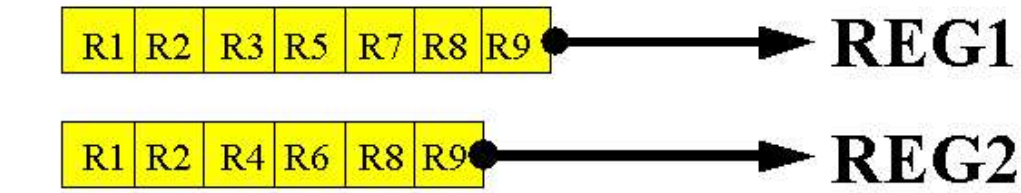


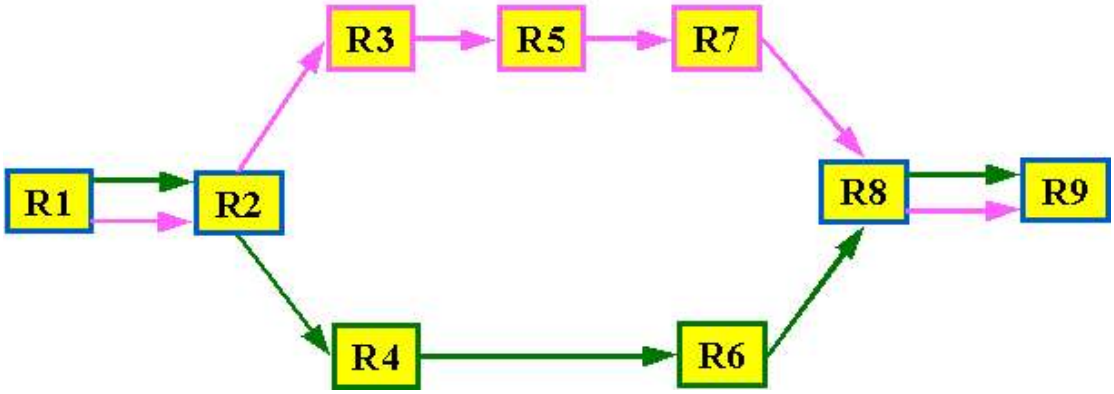
к каждому из них можно обратиться самостоятельно и получить содержимое. Если заполнить регистр REG1 последовательностью битов 0111011, а потом, не проводя никаких дополнительных операций считать ее из REG2, то можно получить последовательность 0101111, т.е. кодирование.

Другой способ осуществить кодирование с помощью нелинейных регистров – организовать разветвленный регистр, каждая ветвь которого представлена самостоятельным регистром и, значит, позволяет осуществлять сдвиг; притом, каждая ветвь может иметь длину, отличную от длин остальных ветвей. Информационный поток, поступая на вход, разделяется на два. При проведении сдвига поочередно по каждой из

ветвей, ветвях может получиться задержка одного из вновь образовавшихся потоков относительно другого.

Работу данного вида регистра иллюстрирует следующая схема:

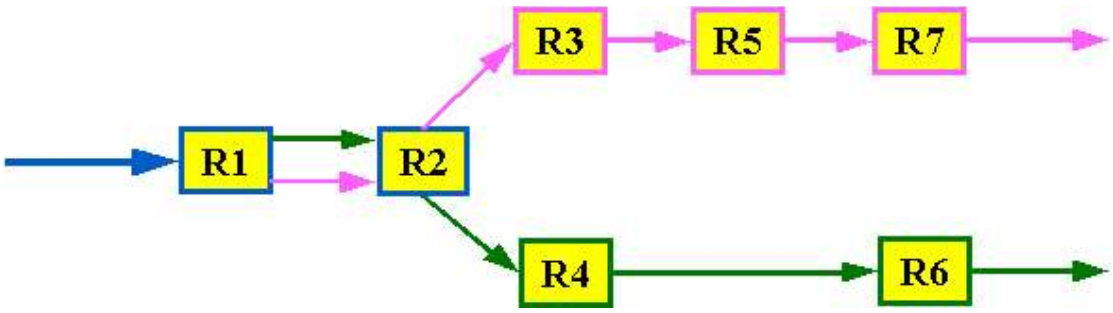


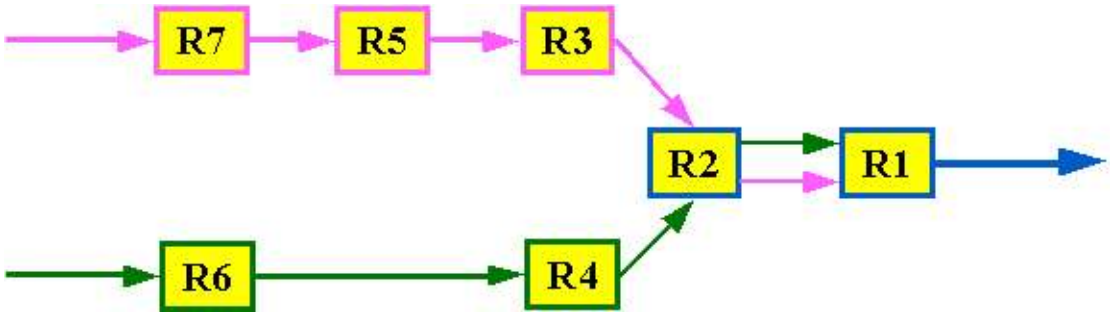


Возможно так же организовать регистры в которых будет происходить разделение потока

- при прохождении через регистр последовательности бит происходит их перестановка. Например, если в регистр поступает последовательность x1x2x3x4x5x 6x7x8 , то на выходе мы получим последо-вательность x2x1x4x3x6x5x8x7 . При более сложной организации регистра можно добиться более сложных перестановок. При прохождении зашифрованного потока бит через регистр в обратном направлении, на выходе получаем исходную последова-тельность.

бит на несколько при этом при обратном прохождении этот поток будет объединяться в исходный.





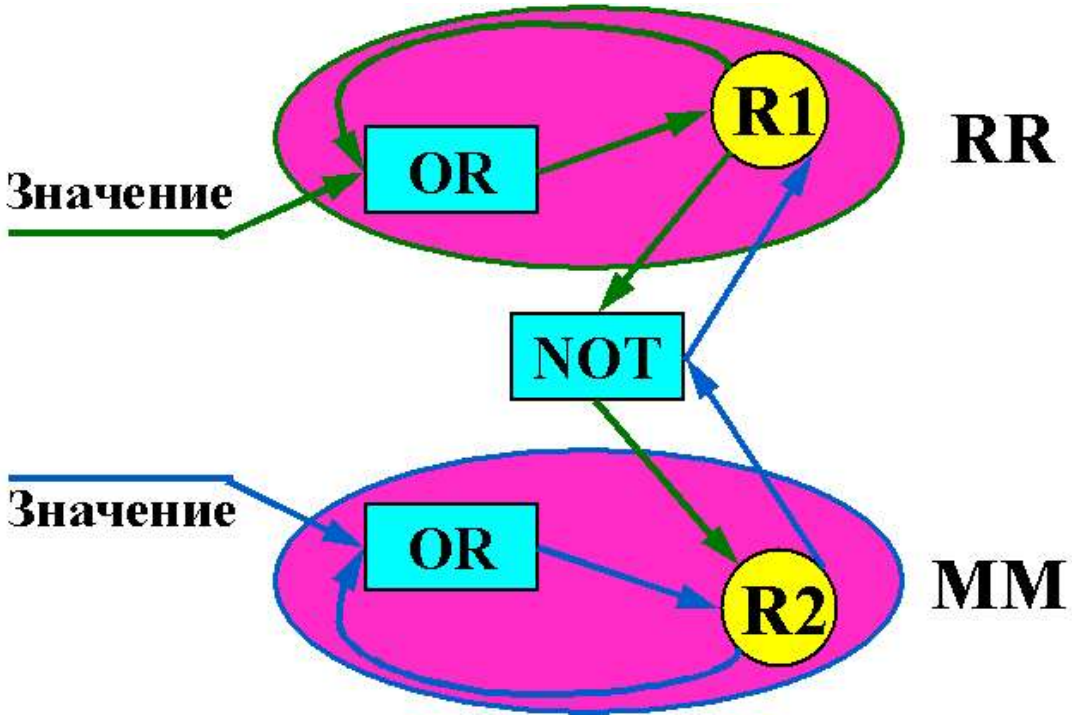
Микропрограмму, оперирующую одной запоминающей ячейкой, выполняющуюся непосредственно во время поступления значения можно представить, как особый вид запоминающей ячейки. В итоге практически все операции можно будет свести к производству сдвига в составном регистре из запоминающих ячеек такого типа.

Например, можно получить компаратор – ячейку, которая может менять свое значение только при поступлении на нее большего (либо только меньшего) значения, чем она содержит. Для двоичной системы счисления, используя сдвиг в компарирующих ячейках, можно выразить все логические и арифметические операции. Для этого нужно связать увеличивающую и уменьшающую ячейки как зеркальные.

Поступающий бит	куда поступает	регистр RR		регистр MM	
		до	после	до	после
0	RR	0	0	0	1
0	RR	1	1	1	0
1	RR	0	1	0	0
1	RR	1	1	1	0
0	MM	0	0	0	1
0	MM	1	1	1	0
1	MM	0	1	0	0

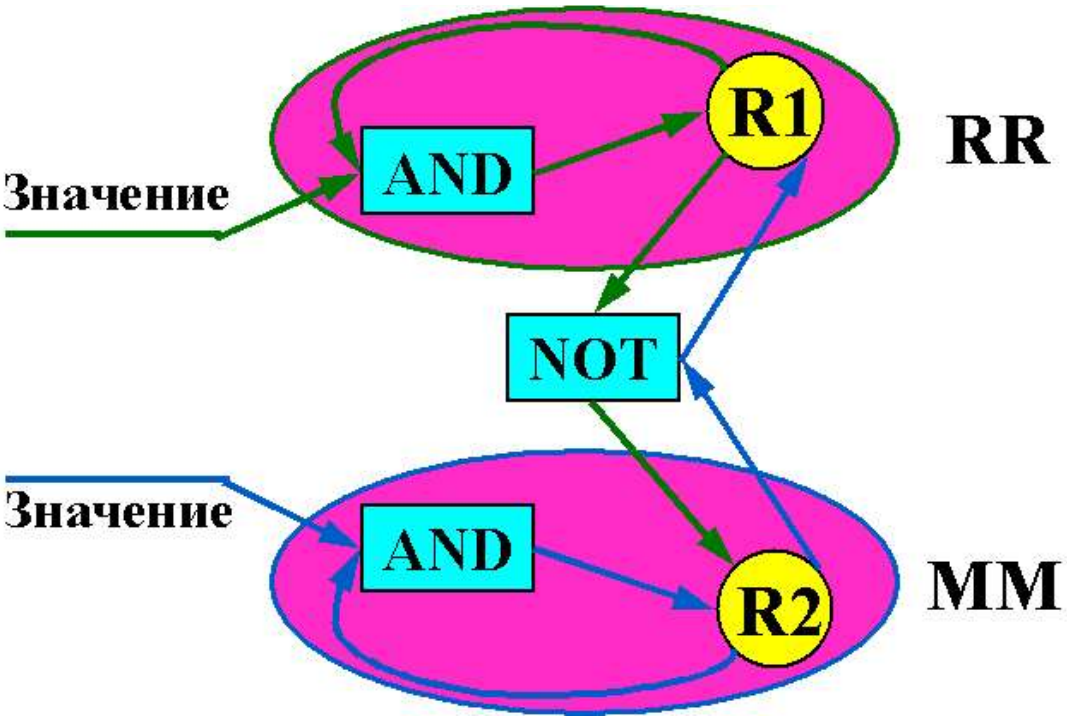
1	MM	1	1	1	0

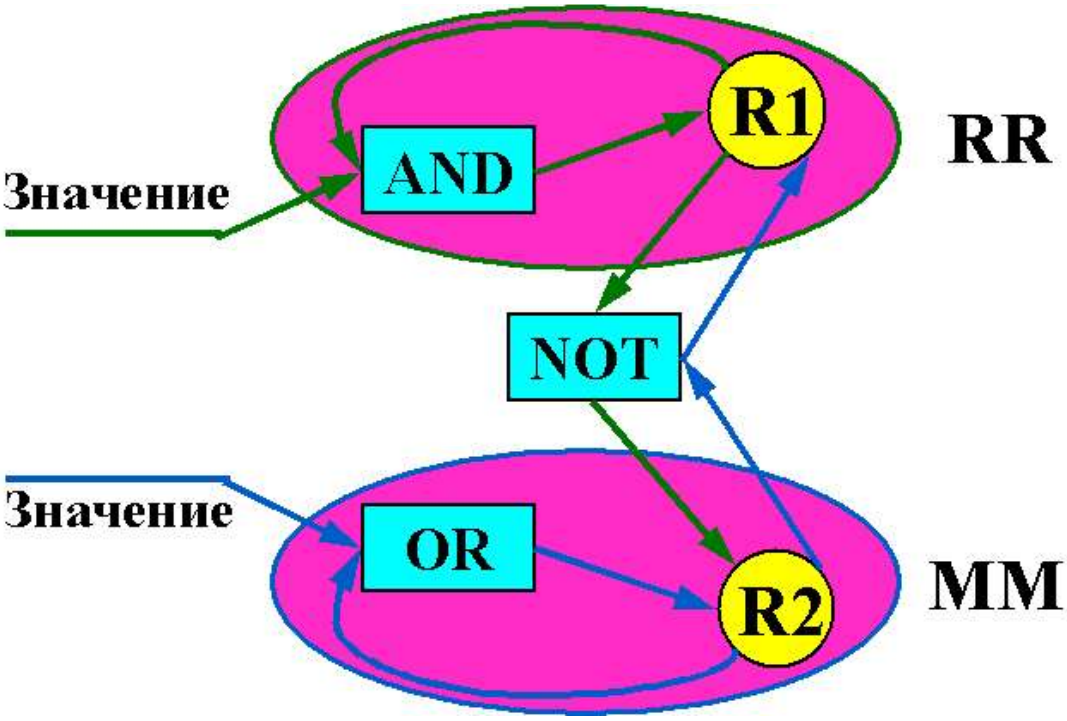
Такие ячейки можно организовать следующим образом:

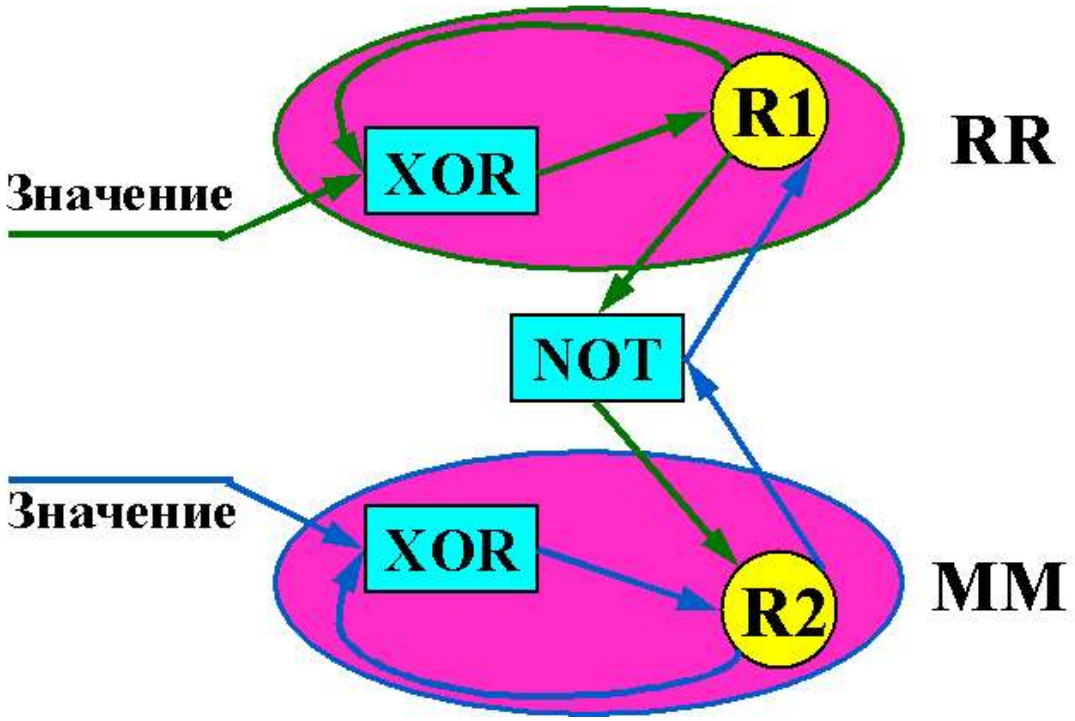


Замечание. Для N-ричной системы счисления компарирующих ячеек не достаточно.

Возможна также организация следующих ячеек, которые позволят значительно усовершенствовать и расширить возможности регистров.


Значение
AND
R1
RR
NOT
Значение
AND
R2
MM


Значение
AND
R1
RR
NOT
Значение
OR
R2
MM



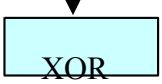
4. Передаточные функции.

« Вот блин!!!»

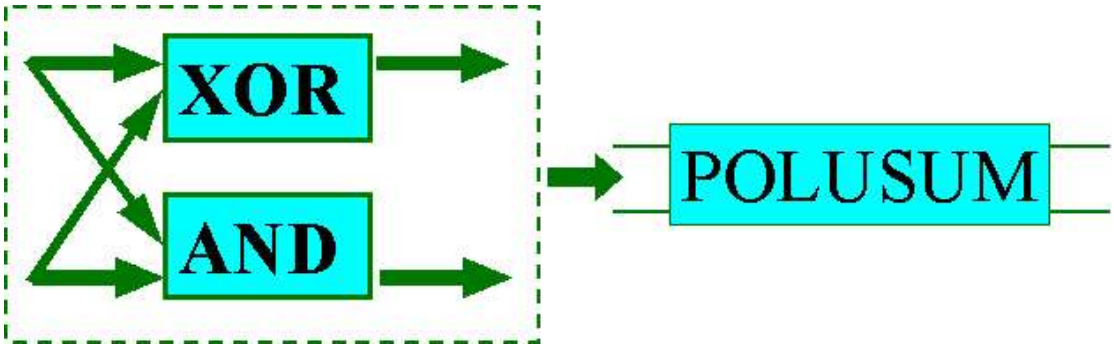
Алтуфьев М.Ю. « Избранное»

Передаточная функция (П.Ф.) - абстрактное понятие, подразумевающее некий элемент, состоящий из конечной линейной последовательности логико-математических операций, выполняемых над поступающими на его вход данными и передающий результат этих операций на выход. Таким образом, передаточная функция, по сути - процесс преобразования информации из одного состояния в другое. Для удобства рассмотрения физической реализации мы предполагаем, что преобразование информации посредством передаточной функции - физический процесс, занимающий конечный интервал времени. Простейшая передаточная функция должна быть представлена таблицей изменения состояния выходов, в зависимости от входов.

in1	in2	out1
пример	0	0
0	1	1
1	0	1
1	1	1



Набор простейших передаточных функций удобно в дальнейшем использовать для построения более сложных функций с помощью графов, а не таблиц. Граф передаточной функции -_ориентированный граф с определенным порядком прохождения, в котором в качестве вершин выступают ранее определенные передаточные функции, которые, в свою очередь могут быть графами, а в качестве ребер - каналы передачи информации от одной вершины к другой. На вход передаточной функции может подаваться как блок данных, так и единичный бит. Ни в коем случае не следует путать понятие передаточной функции с понятием *микропрограммы*.



пример организации П.Ф с помощью графа с последующей инкапсуляцией в одноэлементную П.Ф.

5. Микропрограммы.

Микропрограммы — средство обработки информации с помощью совокупности запоминающих ячеек, связанных передаточными функциями. Фактически, под такое понятие микропрограммы подходит множество объектов, в том числе информационная система в целом, однако мы рассмотрим это понятие на логическом уровне. В этом случае

микропрограмма - ориентированный граф, в котором вершины - запоминающие ячейки, а ребра - передаточные функции. Микропрограммы связывают процесс обработки информации с процессом ее хранения. Информация, поступающая на вход микропрограммы, помещается во входные запоминающие ячейки (внутренние или внешние), обрабатывается передаточными функциями и подается на выход. Возможно так же использование запоминающих ячеек для хранения промежуточных данных, так как процесс обработки информации может быть многошаговым и/или нелинейным.

Микропрограмма представляет собой объективную реальность, которая неизбежно присутствует при любом упоминании вычислительной, и даже более того, информационной системы. Никакая информационная система не может существовать без микропрограмм, прежде всего потому, что сама в широком смысле является микропрограммой. Чаще всего широчайшее понятие микропрограммы намеренно сужают с намерением использовать в каком-либо конкретном случае, просто потому, что дать общее и в то же время достаточно четкое определения понятия «микропрограмма» достаточно сложно. Например, мы, в соответствии с логикой описания задачи дали вышеприведенное определение микропрограммы, однако оно не является, в сколь бы то ни было значительной степени, общим. И, хотя для данной работы этого вполне достаточно, поскольку мы рассматриваем микропрограмму пока только как некое внутреннее представление операций ассемблера, т.е., фактически, описание средств взаимодействия пользователя с моделируемым объектом, в дальнейшем возможно расширение функций микропрограмм, в связи с тем, что они, при рассмотрении в более широком смысле, ни что иное, как *овеществленное* отображение передаточных функций с одновременным расширением их возможностей (в частности нелинейности).

Подытоживая вышесказанное отметим, что возможности микропрограмм ограничены только воображением создателя, постольку поскольку метаматематическая модель любого объекта может быть задана микропрограммой, а, в свою очередь, с помощью математического аппарата можно описать с большей или меньшей точностью практически любой из известных человечеству процессов. В нашей же программе на микропрограммы накладывается лишь одно серьезное ограничение – дискретность логики, которое позже может быть устранено.

3. ПРОГРАММА

1. Общее описание

Программа была реализована с помощью среды программирования Borland Delphi 4.0. В основы проектирования был положен многодокументный интерфейс, использующийся в большинстве современных программных систем, таких как Microsoft Office, Microsoft Visual Studio и т.д. Программа рассчитана на подготовленного пользователя, т.е. он должен иметь представление о том как читать временные диаграммы и строить графические схемы. В последствии в программу войдет обширная система помощи и, возможно, целый ряд мастеров, облегчающих создание базовых элементов.

Логика программы такова. Единовременно может быть открыт только один проект, который в данной версии содержит используемый набор передаточных функций. В последствии мы намерены ввести следующую классификацию проектов используемых нашей программой:

- Проекты-модели. Это описания готовых функционирующих систем имеющих реальную физическую реализацию. Это может быть реальная модель процессора, датчика и т.д. и т.п.
- Проекты-библиотеки. Содержат описания часто используемых элементов. Библиотеки можно подключать при создании первого вида проектов. Возможность повторного использования – это их главное отличительное свойство.

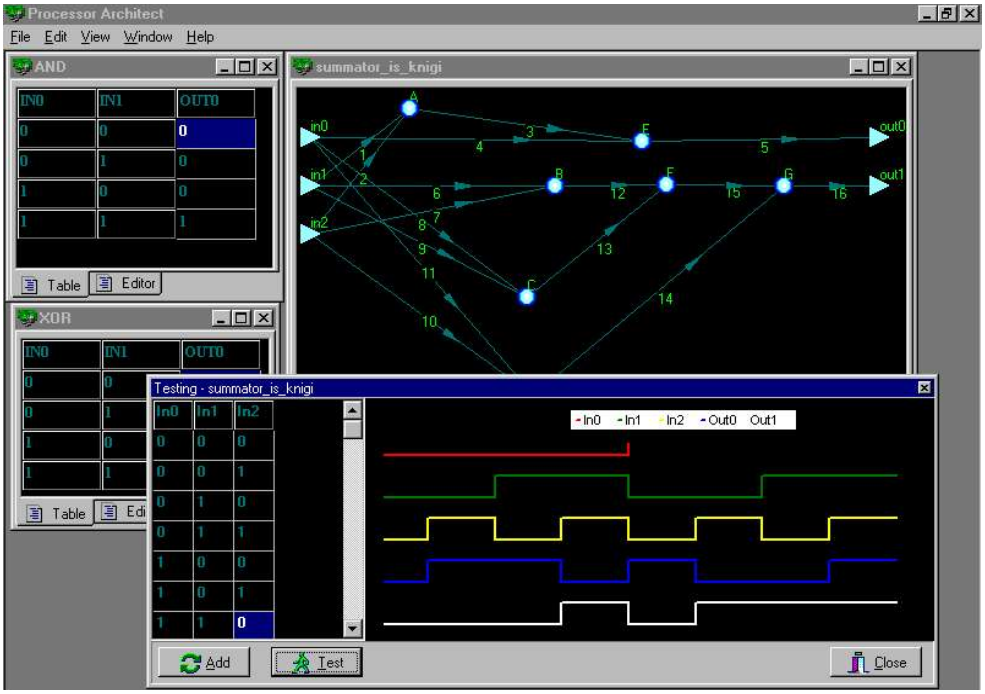
Если сравнивать нашу программу, допустим с интегрированной средой Delphi, то проекты-библиотеки – это пакеты компонентов, а проекты модели – готовые программы.

Как первый, так и второй вид проектов имеет в своем составе описания различных составляющих элементов. Открыть любой из составляющих элементов можно только при открытом проекте. Составляющие элементы бывают трех видов: передаточные функции, регистры и микропрограммы. Каждый из них имеет свой особый вид редактора. В текущей версии программы реализован лишь редактор передаточных функций. Главным (центральным) элементом всех трех редакторов является редактор графов, т.к. все три составляющих элемента могут быть представлены в виде графа. Включение в программу проектов-библиотек не единственное новшество которое планируется включить в следующие ее версии. Используя их можно сделать программно расширяющийся

интерфейс, позволяющий, например, создавать пользовательские мастера и инструменты. Подобный подход уже реализован в средах Visual C++ и Delphi.

2. Интерфейс.

При разработке интерфейса данной программы мы старались сделать его наиболее понятным и удобным для пользователя. К сожалению, мы не успели сделать его гибко настраиваемым, однако в последующих версиях мы это учтем. Стандартная настройка, по нашему мнению является оптимальной, поскольку она не утомляет глаза и пригодна для длительной работы. В текущей версии программы реализован лишь редактор передаточных функций, однако, в принципе по нему можно судить о будущей реализации остальных редакторов. Мы пытались сделать интерфейс максимально простым.



Описание меню:

- File
  - New – предлагает выбрать тип вновь создаваемого элемента, а также выводит необходимые диалоги для ввода параметров элемента (таких как число входов, выходов, название и т.д.).
  - Open – позволяет открыть ранее созданный элемент, включенный в проект. Один элемент нельзя открыть два раза.
  - Save – сохраняет текущий элемент в проекте.

- Load from file – загружает проект-модель. Все открытые окна редактора закрываются. В окне Open будут отображаться другие составные элементы.
- Exit – выходит из программы. Если проект менялся, то выдаст запрос на сохранение.

- Window
  - Tile – выстраивает окна мозаикой.
  - Cascade – выстраивает окна каскадом.
  - Arrange All – выравнивает значки свернутых окон.
- Help
  - Contents – вызов списка тем справки.
  - Search on help – поиск в справке.
  - How to use help – указания по использованию справочной системой.
  - About – информация о создателях.
- Edit (для редактора передаточных функций)
  - Insert Peak – вставляет новую вершину в граф.
  - Delete Peak – удаляет вершину из графа.
  - Function as – редактирует функцию в виде:
    - Graph – графа.
    - Table – таблицы соответствий.
  - Property – свойства текущего элемента управления.
- View (для редактора передаточных функций)
  - Graph (Table) – показывает редактор графов(таблицы).
  - Editor – показывает редактор внутреннего языка.
  - Test Window – показывает окно тестирования передаточной функции.

Описание окна редактирования:

Окно редактирования имеет две закладки. Первая закладка представляет собой редактор графов. Редактировать граф можно просто вставляя вершины и рисуя ребра между ними. Вставка вершин производится через главная меню или горячими клавишами. После

вставки вершины можно перемещать, удерживая их левой клавишей мыши. Таким образом, можно представить схему наиболее удобным образом. Ребра проводятся правым щелчком мыши на ребре. «Резиновая нить» показывает как ребро проводится. Последовательность прохождения ребер определяется последовательностью их рисования.

Второй вид первой закладки – таблица.

Двоичный полусуматор.

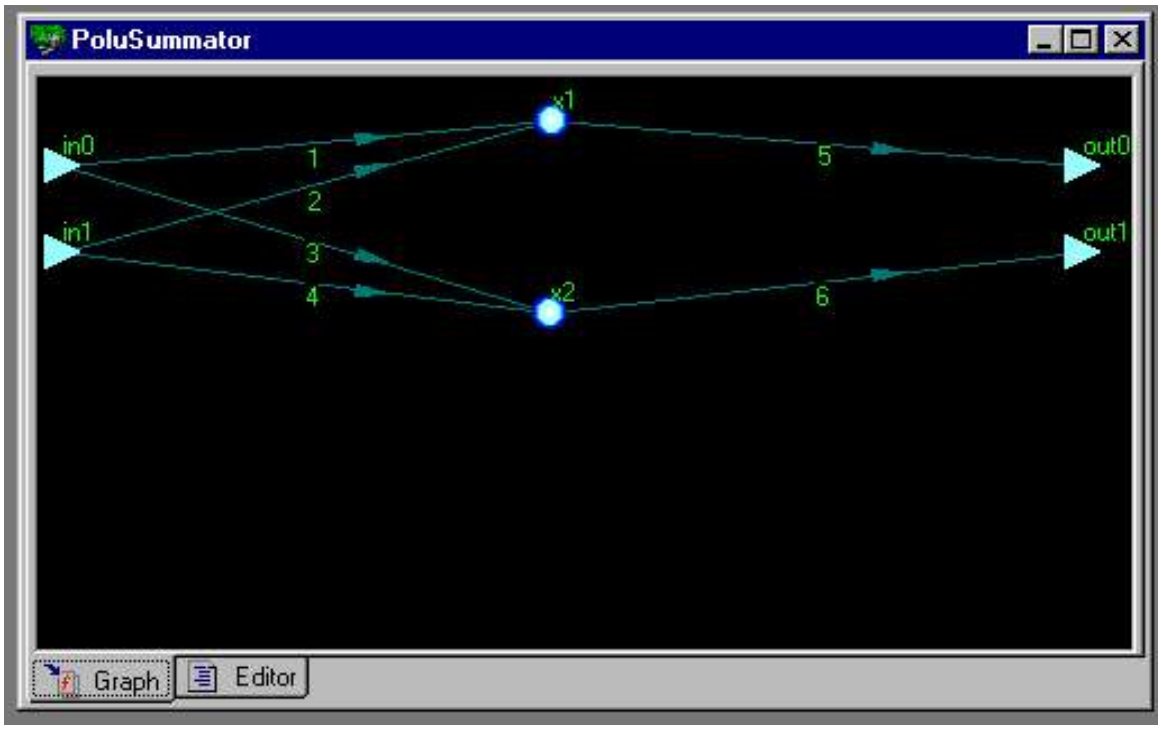
Описание графа.

Входы, на которые поступают биты информации: in0 и in1;

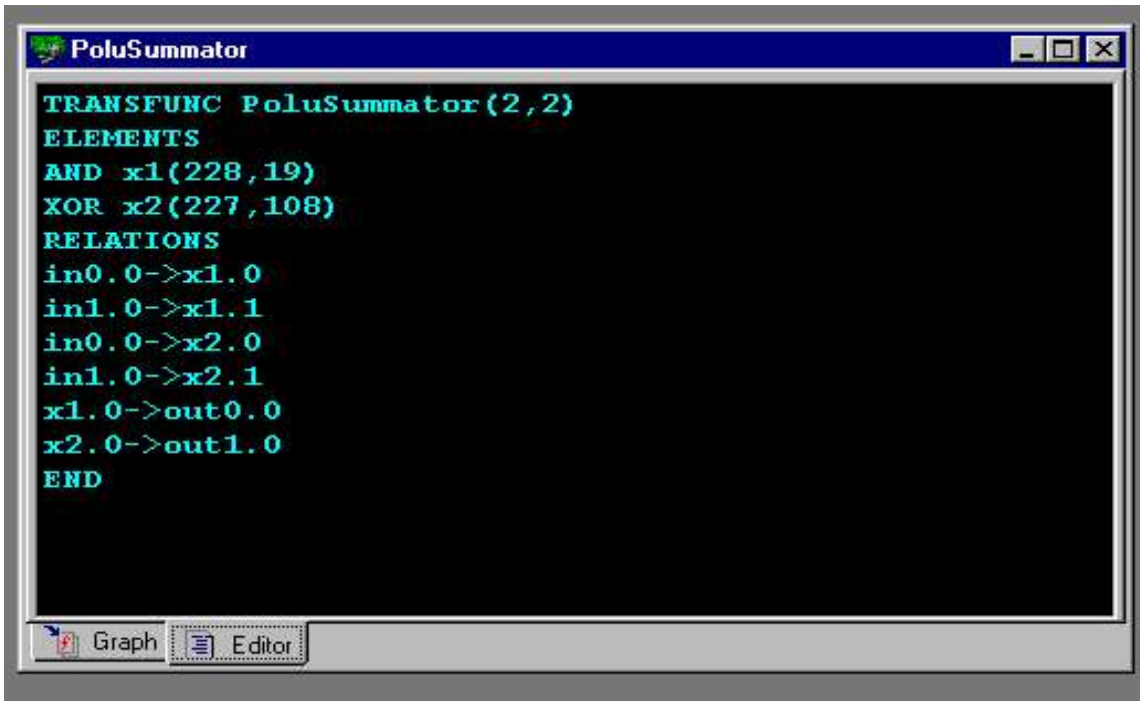
Выходы: out0 – выход суммы; out1 – выход переноса;

Вершины: x1 – двоичная операция обобщенного умножения (AND); x2 – двоичная операция обобщенного сложения (XOR). Каждая из них является передаточной функцией и задана таблицей, т.е. используется (в данном случае) как одна из двух базовых операций основания бинарной логики.

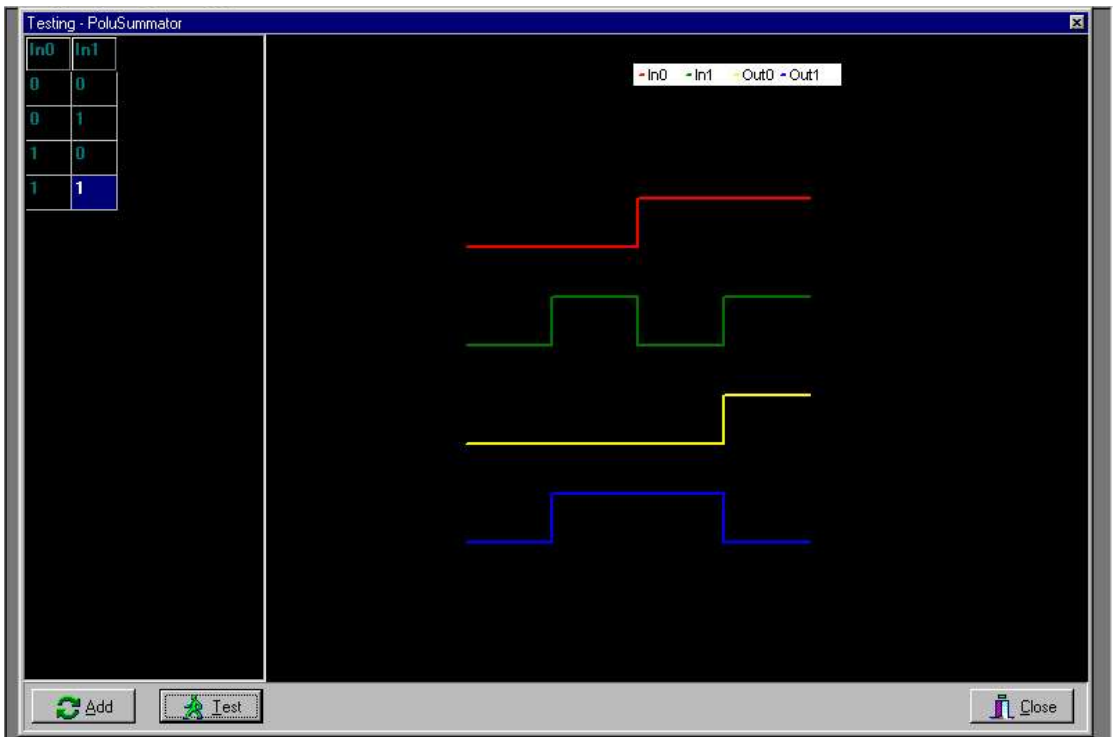
Ребра – веса (номера) ребер представляют собой последовательность их прохождения. Сначала со входов информация подается на вход двоичной операции обобщенного умножения (AND). Затем, опять же со входов (не забывайте, что мы считаем процесс обработки передаточной функции бесконечно быстрым, т.е. не занимающим времени), информация подается на вход двоичной операции обобщенного сложения (XOR). Далее вычисляется информация поступившая на вход двоичной операции обобщенного умножения (AND), и результат подается на первый выход двоичного полусумматора (сумма) – out0. После вычислений внутри двоичной операции обобщенного сложения (XOR) результат с ее выхода поступает на второй выход двочного полусумматора (перенос) – out1.



В окне редактора двоичного полусумматора можно воочию наблюдать последовательность всех описанных выше манипуляций с битами информации.



Окно тестирования двоичного полусумматора предоставляет визуальное отображение его работы. В этом окне мы можем наблюдать таблицу состояний входов и диаграмму изменений входов и выходов в течение условного «времени», которое, фактически, представляет собой аналоговое представление дискретной последовательности поступления бит информации на входы полусумматора.



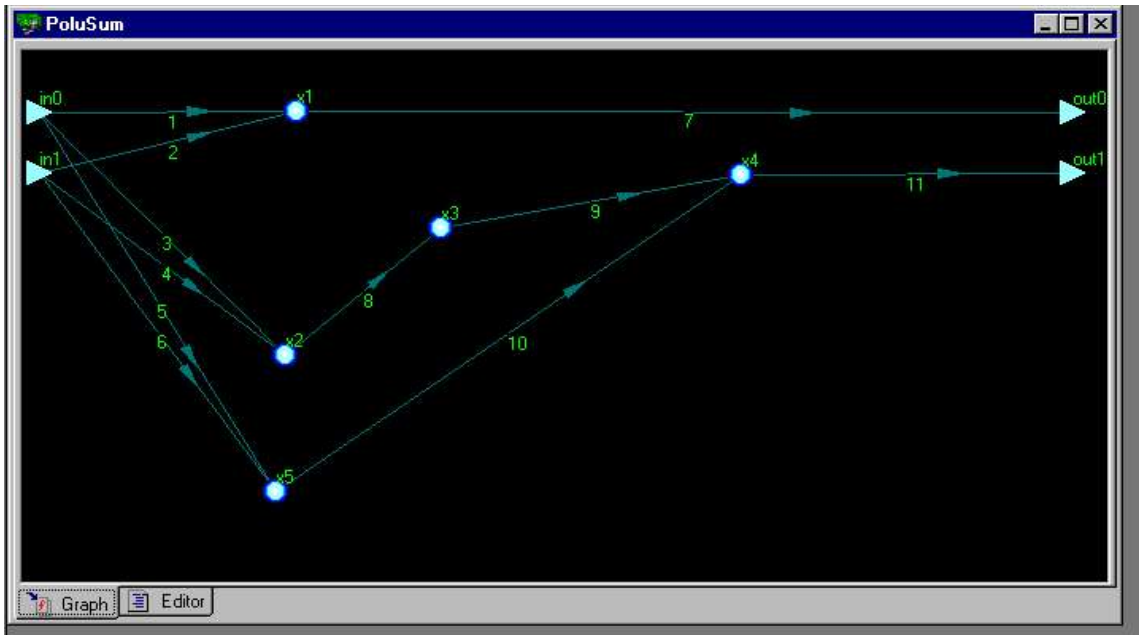
Троичный полусумматор.

Описание графа.

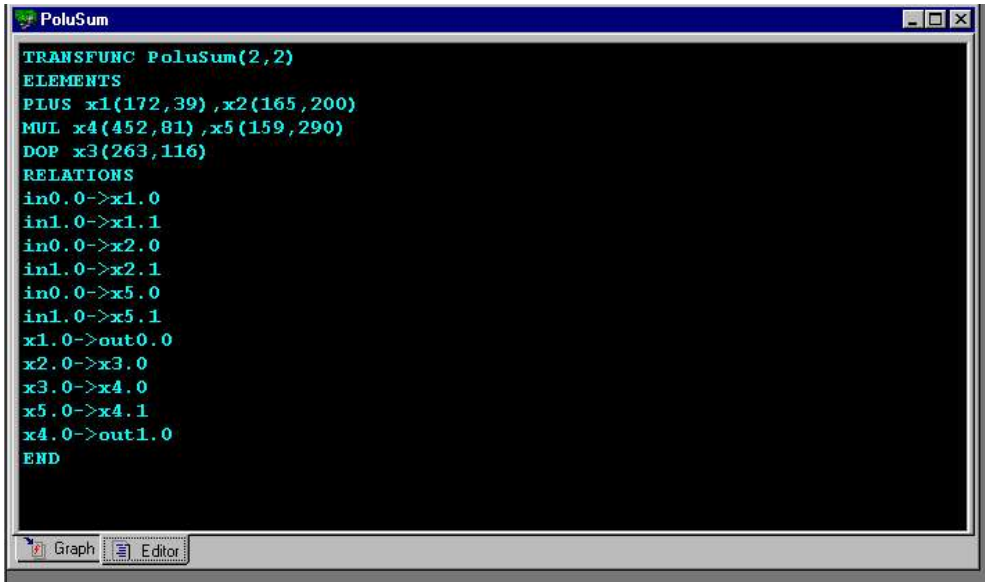
В троичном полусумматоре так же как и в двоичном два входа (in0, in1) и два выхода (out0, out1), однако, поскольку он основан на троичной логике, процесс вычисления суммы и переноса сильно усложняется. Это связано: во первых, с тем, что необходимо учитывать большее число возможных сочетаний битов информации, поступающих на вход в базовых троичных операциях обобщенного умножения (MUL) и сложения (PLUS), а также операции дополнения (DOP – в двоичной системе просто отрицания), заданных таблицами; во-вторых, тем что сама структура троичной логики не позволяет свести всё преобразование входной информации к выполнению всего лишь двух действий, как это было в предыдущем случае.

Итак, на графе мы можем видеть входы in0, in1, выходы out0, out1, и вершины x1, x2, x3, x4, x5. Где вершины x1 и x2 – передаточные функции, представляющие собой троичную операцию обобщенного сложения (PLUS), вершины x3 и x4 представляют троичную операцию обобщенного умножения (MUL), а вершина x5 – троичную операцию дополнения (DOP).

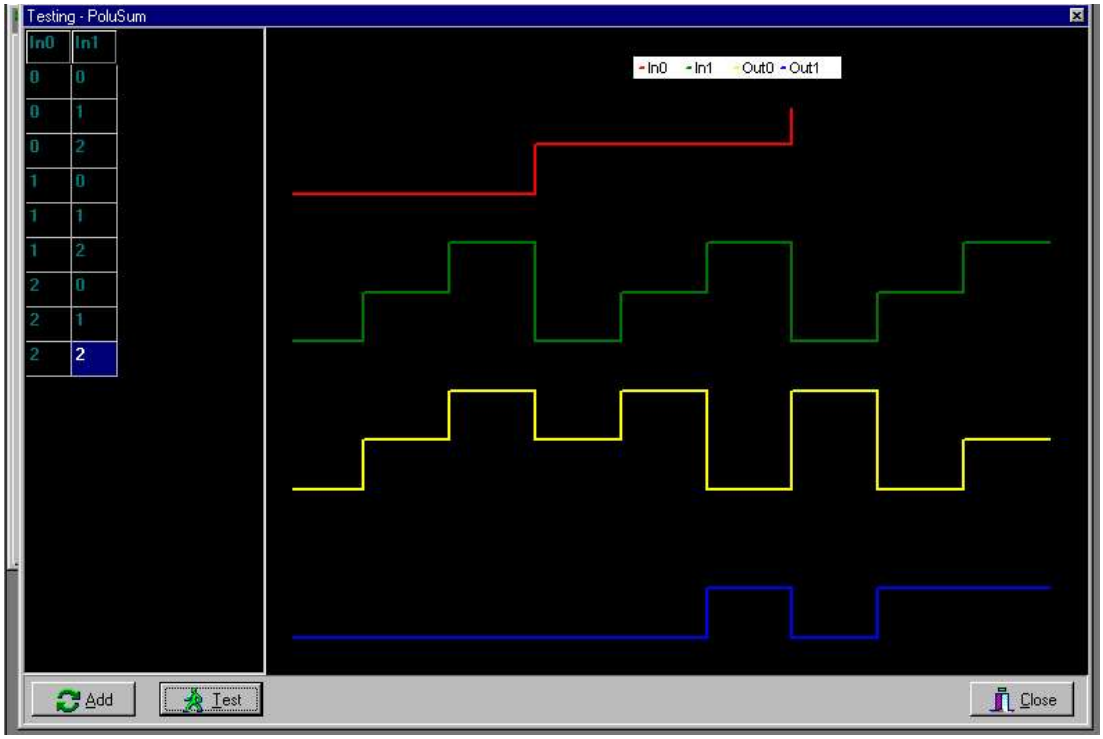
Как и в двоичном полусумматоре вес ребер представляет собой последовательность «прохождения» по ним информации. Как можно видеть, сначала троичные биты информации передаются со входов троичного полусумматора на входы базовых передаточных функций x1, x2 – PLUS, x5 – MUL, затем выполняется вычисление в x1 и результат поступает на выход out0 (сумма) троичного полусумматора. Потом результат такого же преобразования и результат дополнения информации пришедшей с выхода троичной операции обобщенного умножения (DOP(MUL)) поступают на x4 – MUL, и уже после таких экзекуций троичный бит передается на выход out1 (перенос).



Ниже мы можем увидеть четкое символизированное представление описанных ранее действий. В таком виде все преобразования выглядят намного проще и кажутся логически упорядоченными, однако без графа картина была бы неполной и непонятной.



Для просмотра результатов работы троичного полусумматора ниже приведены таблица состояний входов и диаграмма изменений входов и выходов в течение условного «времени», представляющего собой аналоговое отображение дискретной последовательности поступления троичных бит информации на входы полусумматора.



Описание внутреннего языка.

Для описания элементов в программе присутствует встроенный язык описания. Он является другой формой представления графа и позволяет производить редактирование элемента вручную, т.е. путем изменения исходного текста.

В качестве примера встроенного языка описания приведем текст описания троичного полусумматора

TRANSFUNC PoluSum(2,2)

ELEMENTS

PLUS x1(173,40),x2(167,110)

MUL x4(458,109),x5(167,205)

DOP x3(331,109)

RELATIONS

in0.0->x1.0

in1.0->x1.1

in0.0->x2.0

in1.0->x2.1

in0.0->x5.0

in1.0->x5.1

x1.0->out0.0

x2.0->x3.0

x3.0->x4.0

x5.0->x4.1

x4.0->out1.0

END

Первая строка программы (TRANSFUNC PoluSum(2,2)) представляет собой общее описание элемента, ее синтаксис:

<тип элемента><имя элемента>(<число входов>,<число выходов>).

Далее следует пункт ELEMENTS, он обозначает начало раздела в котором идет описание используемых в этом элементе передаточных функций.

Далее идут писания передаточных функций – PLUS x1(173,40),x2(167,110), синтаксис этого описания следующий:

<тип передаточной функции> < имя передаточной функции >(<координата x на графе>,< координата y на графе >)…< имя передаточной функции >(< координата x на графе >,< координата y на графе >)

RELATIONS

Далее следует пункт RELATIONS, он обозначает начало раздела в котором идет описание используемых в этом элементе связей передаточных функций, на графе им соответствуют ребра.

После пункта RELATIONS начинается описание ребер ,например –

in0.0->x1.0

x1.0->out0.0

x3.0->x4.0

синтаксис описания ребер следующий:

<источник><направление связи><приемник>

Заканчивается описание элемента ключевым словом END.

4. ИСПОЛЬЗОВАНИЕ И ПЕРСПЕКТИВЫ ПРИМЕНЕНИЯ

Данная программа задумывалась как наиболее универсальный инструмент построения вычислительных систем, поэтому трудно сказать в каких конкретно областях она будет применяться, легче сказать где она применяться не будет. В программе предоставляется доступ ко всем уровням организации, что дает пользователю возможность подойти к организации системы на максимально низком уровне. Вместе с тем есть возможность объединения имеющихся схем в отдельный элемент и дальнейшее использование этого элемента в организации других. Таким образом пользователь может работать и на сколь угодно высоком уровне, используя готовые элементы, построенные им ранее либо построенные его коллегой. Все это открывает возможности моделирования как целых систем, так и отдельных элементов. Возможно также использование каждой из частей программы отдельно, т.е. можно использовать только конструктор передаточных функций и эмулировать их работу, можно использовать только конструктор запоминающих элементов эмулировать работу регистров и каналов передачи информации. А теперь подробней о главных отраслях применения программы.

1.коммутационные регистровые системы.

Одним из направлений применения данной программы является образования многочисленных, изменяющихся во времени, разветвляющихся и переплетающихся в пространстве каналов связи между большим числом перерабатывающих информацию объектов, расположенных в ограниченной области пространства, между которыми осуществляется одновременный интенсивный обмен большими потоками информации.

С решением подобной проблемы приходится сталкиваться при конструировании однородных вычислительных структур и универсальных ЭВМ современной архитектуры, при разработке адаптивных обучающихся систем управления, при создании распознающих устройств и бионических моделей различных органов чувств, при разработке самостоятельно взаимодействующих с внешней средой роботов, наконец, при синтезе нейроноподобных и мозгоподобных структур.

Еще в прошлом тысячелетии появились первые ламповые цифровые вычислительные машины, казавшиеся в то время чудом техники двадцатого века. Исходя из принципиальной способности первых ЭВМ решать любые математические и логические задачи, в тот период предполагали, что такие машины скоро смогут достичь высокого

уровня развития и приблизиться по своим возможностям к человеческому мозгу. Однако в течение достаточно короткого времени стало ясно, что на этом пути имеются серьезные принципиальные трудности теоретического, конструкторского и технологического характера.

С момента своего рождения цифровые вычислительные машины стали интенсивно использоваться для цифрового моделирования и управления, для решения широкого круга научных, экономических и творческих задач, для воспроизведения функций живого мозга и органов чувств живых организмов, для решения проблем искусственного интеллекта, для создания различных роботов, кибернетических и бионических устройств.

Решение всех этих задач требовало резкого улучшения характеристик и расширения возможностей ЭВМ. Оказалось необходимым обеспечить адекватность процессов, происходящих в ЭВМ, процессам в моделируемых объектах не только в математическом и логическом смысле, но и в пространственно-временном отношении. Потребовалось добиться функционирования ЭВМ с высокой точностью в реальном масштабе времени. Необходимо было обеспечить высокую надежность, а в ряде случаев и высокую живучесть ЭВМ, способность к перестройке и наращиванию структуры. Одновременно с резким усложнением структуры ЭВМ возникла проблема обеспечения малых габаритов и весов, а также минимального потребления энергии. Усложнение структуры, методов программирования и математического обеспечения ЭВМ потребовало решения проблемы простоты общения ЭВМ с человеком и окружающей средой, а также эффективных методов управления ЭВМ. Стало очевидным, что необходимо обеспечить высокое быстродействие цифровой вычислительной машины, колоссальный объем памяти, параллельность переработки информации в процессорах.

Все это привело, в свою очередь, к необходимости создания принципиально новой технологической базы ЭВМ, к бурному развитию вначале полупроводниковой техники, затем микроэлектроники и, наконец, больших и сверхбольших интегральных схем (СБИС). В последние годы появились и интенсивно внедряются в практику микропроцессоры, выполненные на одном кристалле.

Достижения в области технологии ЭВМ, в области микроэлектроники заставили по-новому подойти к решению задач архитектуры ЭВМ, потребовали обеспечить однородность структуры ЭВМ на уровне крупных микропроцессоров и других типов микроузлов: микроблоков памяти, коммутации управления и т. п. Эти же достижения технологии позволили наряду с обычными принципами программирования процесса

переработки информации в ЭВМ сформировать принципы программирования структуры ЭВМ в целом и структуры отдельных ее микропроцессоров.

Проведенные в последние годы исследования показали, что решение всех перечисленных проблем лежит на пути создания однородных микропроцессорных вычислительных структур, представляющих собой однородные системы однотипных микропроцессоров, каналы связи, между которыми образуются программным способом с помощью однородной коммутационной структуры, а каждый микропроцессор может перестраиваться на выполнение одной макрооперации из заданного множества макроопераций.

Действительно, повышение надежности и живучести требует возможности замены вышедших из строя микропроцессоров, коммутирующих элементов и других микроблоков аналогичными, что в свою очередь делает необходимой однотипность как микропроцессоров, так и других микроблоков, а также однотипность коммутирующих элементов.

Высокая точность и работа в реальном масштабе времени и даже опережение последнего обеспечиваются при распараллеливании вычислительного процесса, что достигается для широкого класса задач лишь при однотипности блоков и однородности структуры.

Наращивание структуры в процессе работы вычислительного устройства, его перестройка при изменении внешней ситуации, внешних условий также требуют однородности и однотипности вычислительных блоков, входящих в структуру.

Наконец, получить малые габариты и вес, обеспечить малое потребление энергии и высокую надежность невозможно без широкого применения больших интегральных схем. В свою очередь, конструирование автоматов и вычислительных устройств на основе микроэлектронных схем ведет к необходимости обеспечения однородности автоматов в смысле однотипности крупных функциональных решающих блоков — микропроцессоров, реализуемых в виде СБИС с малым количеством вводов и выводов, и однородности соединений таких блоков. Это следует из того, что применение микроэлектроники оказывается эффективным лишь тогда, когда имеется возможность выполнять узлы вычислительного устройства в виде СБИС с высокой степенью интеграции, число различных типов которых относительно невелико. В противном случае придется производить сотни и тысячи типов СБИС, что, очевидно, совершенно нерентабельно и нетехнологично.

Таким образом, изложенное говорит о целесообразности построения современных

вычислительных устройств и тем более вычислительных устройств и автоматов будущего в форме однородных вычислительных структур, состоящих из большого числа достаточно крупных однотипных, выполненных в виде СБИС с малым числом информационных, управляющих и энергетических входов и выходов, микропроцессоров, которые соединяются между собой с помощью однородной коммутирующей структуры в соответствии с заданной программой или решаемой задачей.

Следует подчеркнуть, что однородные вычислительные структуры обладают такими же свойствами универсальности, как и классические ЭВМ. Для этого достаточно строить эти структуры на основе универсальных микропроцессоров. Под универсальным микропроцессором здесь понимается процессор, выполненный на едином кристалле в виде СБИС, структурно перестраиваемый на выполнение любой макрооперации из заданного ограниченного универсального множества макроопераций, которое обеспечивает решение любых математических и логических задач. К числу указанных макроопераций можно отнести: операции интегрирования и дифференцирования; функциональные, матричные, векторные и тензорные преобразования; операторные и вероятностные преобразования; логические и арифметические операции.

Каждый микропроцессор перестраивается на одну из указанных операций с помощью элементарной команды без какого-либо сложного программирования. Это достигается структурным программированием микропроцессора, которое выполняется под воздействием элементарной команды: каждой команде соответствует настройка процессора на конкретную структуру, с помощью которой реализуется выполнение одной из перечисленных операций.

Между микропроцессорами однородной вычислительной структуры осуществляется интенсивный обмен информацией. Каналы связи между микропроцессорами структуры изменяются во времени в зависимости от моделируемого объекта, от внешней ситуации и от результатов переработки информации внутри структуры. В связи с этим возникает задача образования многочисленных, изменяющихся во времени, разветвляющихся и переплетающихся в пространстве каналов связи, но которым осуществляется обмен информацией между микропроцессорами. Указанная задача решается путем образования однородной коммутационной структуры, которая должна заполнять пространство между микропроцессорами и обеспечивать автоматическую настройку каналов связи между ними в соответствии с заданной программой коммутации.

Таким образом, настройка однородной вычислительной структуры осуществляется на

двух уровнях: на уровне настройки каждого микропроцессора на заданную макрооперацию из ограниченного множества макроопераций и на уровне настройки соединений микропроцессоров на заданную коммутацию. При этом решение любой задачи или моделирование какого-либо объекта в подобной структуре оказывается максимально простым, так как сводится к отображению в однородную вычислительную структуру достаточно крупных операций объекта, выполняемых в структуре отдельными микропроцессорами или группами микропроцессоров, и пространственно-временных связей моделируемого объекта, реализуемых однородной коммутационной структурой.

Проблемы коммутаций требуют своего решения не только в однородных микропроцессорных вычислительных структурах.

В настоящее время все чаще возникает задача синтеза весьма сложных кибернетических устройств, содержащих чрезвычайно большое число расположенных в ограниченном объеме пространства элементарных ячеек переработки информации, между которыми осуществляется интенсивный одновременный обмен большими потоками информации, причем каналы связи между элементарными ячейками изменяются во времени в зависимости от внешней ситуации и от результатов переработки информации внутри устройства. Отсюда — все чаще оказывается, что решение многих проблем кибернетики в сильной степени зависит от решения проблемы массового синтеза и преобразования большого числа разветвленных в пространстве каналов связи, предназначенных для одновременной передачи больших потоков информации между многими объектами.

С решением подобной проблемы приходится сталкиваться не только при создании современных однородных вычислительных структур, но и при разработке адаптивных обучающихся систем управления, при конструировании распознающих устройств, при моделировании различного рода органов чувств, при разработке роботов, самостоятельно взаимодействующих с внешней средой, наконец, при синтезе нейроноподобных и мозгоподобных структур.

Во всех этих случаях возникает острая необходимость в решении задачи автоматического образования гибких пространственных каналов передачи информации, в разработке методов управления большими скоплениями переплетающихся и разветвленных в пространстве каналов связи, в создании принципов распределения потоков информации по каналам связи в пространстве и способов управления сложными пространственными информационными потоками.

Решение этих проблем принципиально отличается от решения задачи образования

небольшого числа каналов связи между двумя или несколькими объектами. Различие здесь не только чисто техническое, связанное со сложностью образования и перестройки больших массивов каналов связи, со сложностью управления этими процессами, но прежде всего принципиальное, заключающееся в том, что во всех рассмотренных случаях массовая перестройка каналов связи качественно изменяет структуру кибернетического устройства, позволяет осуществлять программирование структуры устройства и таким образом непосредственно оказывает влияние на процесс и результаты переработки информации.

Решение проблемы массового синтеза и перестройки каналов связи позволяет решить проблему управления сложными кибернетическими системами и особенно проблему их самонастройки, а также проблему структурного и коммутационного программирования автоматов.

Без решения этих проблем не представляется возможным решить задачу создания самостоятельно функционирующих и активно взаимодействующих с внешней средой нейроноподобных и мозгоподобных структур, так как обучение и функционирование таких структур связано прежде всего с настройкой и перестройкой каналов передачи информации между нейронами и нейронными ансамблями. В естественных нейронных ансамблях, в живых нервных системах, в любом естественном живом мозге важнейшей функцией является образование каналов связи для обмена информацией между нейронами, между органами чувств и мозгом, между нейронными ансамблями и исполнительными мышечными механизмами.

Все изложенное приводит к необходимости постановки и решения актуальной задачи кибернетики — разработки теории и принципов построения гибкой коммутации большого числа объектов, расположенных в ограниченной области пространства и систематически, одновременно и в произвольном порядке обменивающихся между собой большими переплетающимися и разветвляющимися в пространстве потоками информации.

В настоящей книге указанная задача решается на основе нового дискретного принципа коммутации, сформулированного автором.

Известно, что в существующих системах коммутации для целей передачи информации образуются непрерывные каналы связи, по которым информация транспортируется от одного объекта к другому непрерывно как в пространстве, так и во времени. Такие коммутационные системы, которые называются в книге непрерывными, обладают

серьезными недостатками, главными среди которых являются сильное затухание передаваемой информации, некалиброванные и несогласованные в разных каналах задержки информации во времени, существенное взаимное влияние коммутационных ячеек друг на друга, малая помехоустойчивость, надежность и живучесть коммутационной системы.

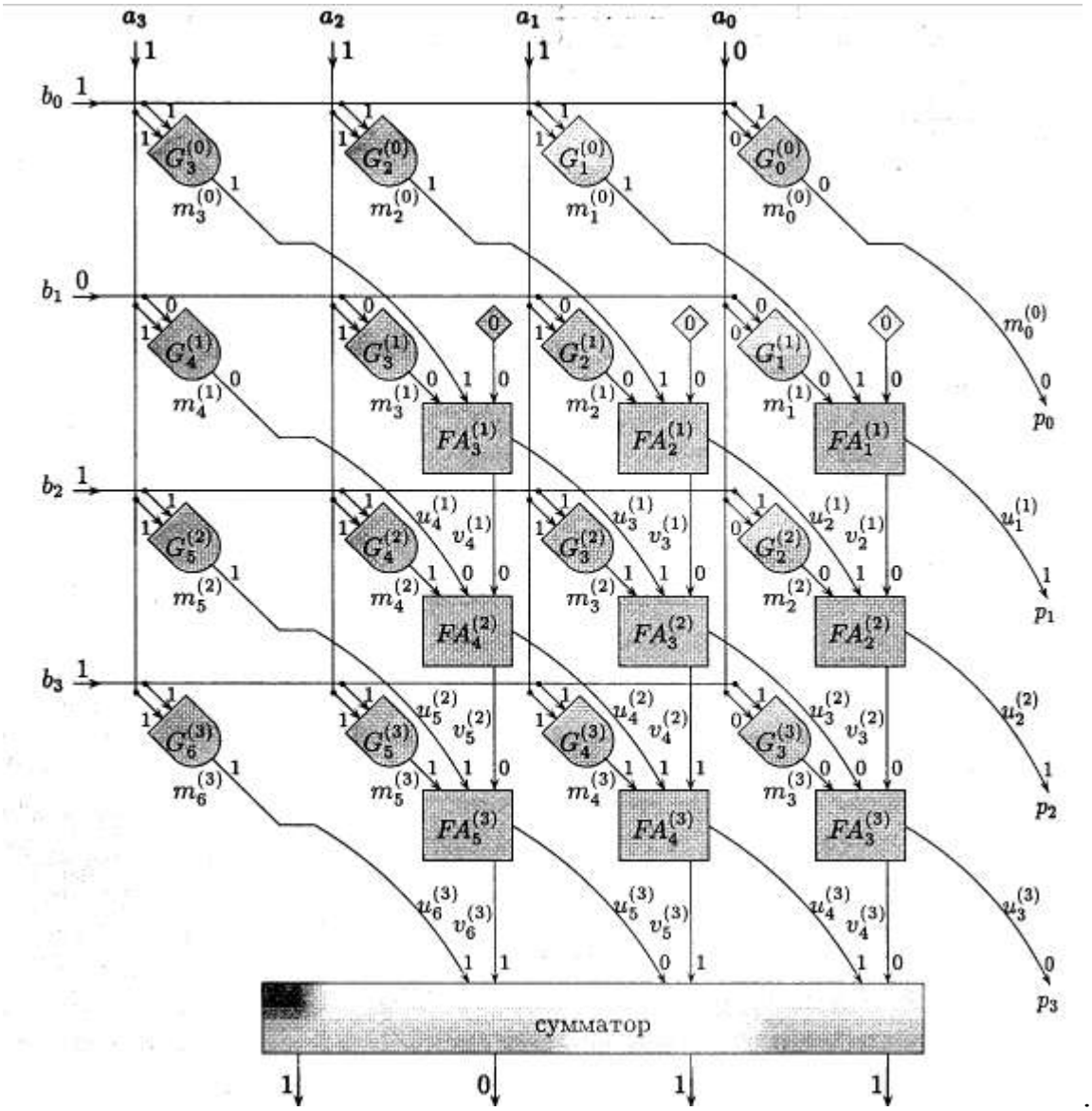
В работе выдвинута и обоснована идея создания дискретных коммутационных систем, которые реализуются на основе разработанных автором однородных коммутационных регистровых структур. В коммутационных регистровых структурах автоматически образуются многочисленные, разветвляющиеся и пересекающиеся в пространстве дискретные регистровые каналы связи, по которым информация передается в цифровой форме дискретно как в пространстве, так и во времени, без затухания и с калиброванными задержками. Однородные коммутационные регистровые структуры отличаются большой гибкостью, высокой помехоустойчивостью, надежностью, живучестью и простотой управления.

Однородные коммутационные регистровые структуры из-за своей регулярности, то есть однотипности коммутационных элементов и их соединений, очень хорошо приспособлены для реализации в виде больших интегральных схем.

Подобные структуры являются основой для быстрого развития однородных вычислительных структур, однородных нейроноподобных и мозгоподобных структур, бионических моделей органов чувств, перцептронных распознающих систем управления, роботов и других кибернетических устройств.

2. Арифметические схемы.

Одним из наиболее ярких примеров применения данной программы является моделирование линейных арифметических схем. Функциональности настоящей версии для этого вполне достаточно. Сложение на них производится достаточно просто и понятно. А вот умножение уже сложнее. Чаще всего для умножения двух чисел используется схема матричного умножителя. Что же она из себя представляет.



5. ЗЛОКЛЮЧЕНИЕ

1. Итог.

Итак, программа представляет собой наиболее универсальный (по нашему мнению на данный момент) инструмент построения вычислительных систем. И, хотя, мы считаем, что данная программа способна удовлетворить большинство пользователей, это не значит, что нет новых направлений ее развития. В качестве возможных – мы можем предложить следующие:

- усовершенствование математического аппарата, а именно:

- применение системы счисления основание которой является функцией времени например, логарифмической.
- применение нецелой системы счисления, в том числе иррациональных чисел.
- Применение ячеек, состояния которых являются функцией времени.

- расширение уровней организации:
  - Использование программы для организации многопроцессорных систем.
- организация сетевых моделей.
- использование программы для эмулирования АЦП и ЦАП.
- использование для эмуляции различных датчиков.

Но все это требует серьезных исследований, и может является темой других работ.

2. Оценка подхода.

Подход, примененный в данной работе представляется нам достаточно универсальным, а посему, мы считаем вполне возможным использовать его в разработке других задач, причем эти задачи могут быть совсем не схожи с теми, которые мы решали в здесь. Вот некоторые принципы этого подхода:

- Представление элемента будет более понятным, если подходить к нему абстрагируясь от его природы и сконцентрировав внимание лишь на его функциональных особенностях.
- Суперпозицию функций можно представить в виде графа с определенным порядком прохождения ребер.
- Сдвиг – основной процесс, протекающий в информационных системах, собственно и представляет собой передачу информации.
- Обратные связи в цепях одной природы можно свести к наличию в них запоминающего звена. Любую информационную можно построить, используя запоминающие ячейки и передаточные звенья.

В процессе разработки, нами выявлены недостатки однобокого рассмотрения тех или иных элементов. Например, можно использовать модель, составленную из передаточных функций (функциональность при сдвиге) и простых запоминающих ячеек или модель из инкапсулирующих запоминающих ячеек (функциональность при запоминании) с проведением простого сдвига, который можно представить в виде элементарной

передаточной функции. И тот и другой подходы имеют свои плюсы и минусы, и выбор между ними не может быть однозначен.

3. Благодарности.

А благодарить есть кого.… Начнем с начала. Прежде всего спасибо нашим родителям за то, что вырастили нас и воспитали, а в последние полгода давали нам неоценимые советы. Спасибо нашим преподавателям и преподавателям нашей кафедры, которые не отказывали нам в помощи, когда она была нам необходима. Особо хотелось отметить Мичника Ю.О., который первым натолкнул нас на эту тему; Саркисова А.Ш., своим присутствием и советами ведшим нас по правильному пути; Лаптева В.В., человека, который посоветовал нам прочесть очень интересные книги и, как мы надеемся, нашему будущему руководителю; Хоменко Т.В., которая помогла нам решить много организационных проблем перед конференцией и, опять таки, порекомендовавшая хорошую литературу. Спасибо!

Также благодарность мы выражаем: Ветровой А.А., Филину В.А., Красненко С., Воронкову С.С., Жиле В.В., Попову Г.А., Квятковской И.Ю., Лайко Т.В., Шамайло О.Н., Плехановой А.В., Косик Е.Д., Зайнутдиновой Л.Х., Васильевой Э.М.. Особая благодарность Корнильеву за то, что последние полгода он не приносил нам плохих вестей. Благодарность администрации Астраханской области, не за что-нибудь, а просто так, на будущее. А также специальная благодарность Астраханским воронам за то, что ни разу на нас не попали, и Мишиной собаке Вите за то, что не покусала.

Огромнейшая благодарность сканеру Mustek-3500.

4. Автобиографии.

Участники группы разработчиков MSA corp.:

- **Морозов Александр Васильевич** – родился 18 апреля 1982 в г. Астрахань. Учился в школе №51, экономико-правовом лицее №10, техническом лицее №1. В 1999 году был зачислен на второй курс Астраханского Государственного Технического Университета. Программистский опыт – 5 лет. В 1995 году стал одним из создателей и участников группы разработчиков MorG inc. В 2000 году вошел в состав группы MSA corp. Увлечения – музыка и создание литературных произведений.

- **Спандерашвили Дмитрий Викторович** – родился 10 июня 1981 г. в г. Кизляр, республики Дагестан. Обучался в школе №19 г. Рустави, республика Грузия, школе №12 г. Рустави, республика Грузия, гимназии №1 г. Кизляр, республика Дагестан. Впервые познакомился с компьютером (intel 286) в 1989 г. в Научно Исследовательском Проектном Институте Автоматики г. Рустави, где благополучно грохнул систему. Впервые начал программировать в 1997 г. В 1998 г. поступил в АГТУ. В 2000 году вошел в состав группы MSA corp. Увлечения – биология, химические основы производства ВВ, особенно бризантных, карточные игры (джокер, преферанс), музыка (Depeche Mode), пешие походы (горы).
- **Алтуфьев М. Ю. (Nosferatum de Ving)** (1??? – 1???) – национальная принадлежность не определена, место рождения неизвестно, место смерти не найдено, захоронение утеряно. С 1??? г. – ученик 8-й школы, с 1??? г. – ученик и последователь идей 6-й школы, с 1??? по 1??? гг. принадлежал к сторонникам технического лицея, позже примкнул к университету. Несмотря на некоторые изменения в мировоззрении со времени знакомства с компьютером в 1??? г. всегда придерживался традиций parahumanoidarianizeров. Спустя ? лет общения с компьютером вошел в состав MSA corp. Увлечения не обнаружены, привязанности не выяснены.

5. Литература.

1. J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications*. American Elsevier, 1976.
2. Richard J. Anderson and Gary L. Miller. *Deterministic parallel list-ranking*. In John H. Reif, editor, 1988 *Aegean Workshop on Computing*, volume 319 of *Lecture Notes in Computer Science*, pages 81-90. Springer-Werlag, 1988.
3. Richard P. Brent. *The parallel evaluation of general arithmetic expressions*. Journal of the *ACM*, 21(2):201-206, 1974.
4. Блейкхут Р.Х. *Теория и практика кодов контролирующих ошибки.*
5. Каляев А.В. *Однородные коммутационные регистровые структуры*. М.: Советское радио, 1978.
6. Кормен Т., Лейзерсон Ч., Ривест Р. *Алгоритмы построение и анализ.* М.: МСНМО, 2000.
7. Кнут Д. *Искусство программирования для ЭВМ. Т.1.: Основные алгоритмы.* М.: Мир, 1977.
8. Кнут Д. *Искусство программирования для ЭВМ. Т.2.: Получисленные алгоритмы.* М.: Мир, 1977.
9. Кнут Д. *Искусство программирования для ЭВМ. Т.3.: Сортировка и поиск.* М.: Мир, 1977.
10. Мкртчан С.О. *Нейроны и нейронные сети*. М., Энергия, 1971.

Главная форма

```
unit main;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  Menus, Placemnt, newitems, transfer_function_options,
transfer_functions,
  OpenItem, Registers;

type
  Tfm_main = class(TForm)
    fs_Main: TFormStorage;
    mm_Main: TMainMenu; mdd_File: TMenuItem; mi_Exit: TMenuItem;
N1: TMenuItem; mi_LoadContext: TMenuItem;
    N2: TMenuItem;   mi_Open: TMenuItem;  mi_New: TMenuItem;
mdd_Window: TMenuItem;   mi_ArrangeAll: TMenuItem;
    mi_Cascade: TMenuItem; mi_Tile: TMenuItem; mdd_Help:
TMenuItem;  mi_About: TMenuItem;  mi_HowtoUseHelp: TMenuItem;
    mi_SearchforHelpOn: TMenuItem;  Contents1: TMenuItem;
    OpenDialog: TOpenDialog;
    mi_Save: TMenuItem;
////////////////////////////////////////////////////////////
///////////////
    procedure mi_ExitClick(Sender: TObject);
    procedure mi_NewClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action:
TCloseAction);
    procedure mi_LoadContextClick(Sender: TObject);
    procedure mi_OpenClick(Sender: TObject);
    procedure mi_SaveClick(Sender: TObject);
    procedure mi_TileClick(Sender: TObject);
    procedure mi_CascadeClick(Sender: TObject);
    procedure mi_ArrangeAllClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
```

```
    FncDesc  : TStringList;
    Profile  : AnsiString;
    NARNOST  : Word;
  end;

var
  fm_main: Tfm_main;

implementation

{$R *.DFM}

procedure Tfm_main.mi_ExitClick(Sender: TObject);
begin  Close; end;

procedure Tfm_main.mi_NewClick(Sender: TObject);
var
  Child : Tfm_transfer_function;
  ChReg : Tfm_registers;
begin
  if not(Assigned(fm_new_elements)) then fm_new_elements :=
Tfm_new_elements.Create(Self);
  if fm_new_elements.ShowModal = mrOk then begin
     case fm_new_elements.switch of
     0:  begin
           if not Assigned(fm_trfnc_options) then
fm_trfnc_options := Tfm_trfnc_options.Create(Self);
           fm_trfnc_options.Reset;
           if(fm_trfnc_options.ShowModal = mrOk) then begin
              Child := Tfm_transfer_function.Create(Self);
              Child.CreateNew(fm_trfnc_options.ed_Name.Text,
                     Round(fm_trfnc_options.rse_inputs.Value),
                     Round(fm_trfnc_options.rse_outputs.Value),
                     NARNOST,true);

           end;
         end;
     1:  begin
           ChReg := Tfm_registers.Create(Self);
         end;
     else
     end;
  end;
```

```
end;

procedure Tfm_main.FormCreate(Sender: TObject);
begin
  FncDesc := TStringList.Create;
////////////////////////////////////////////////////////////////
///////////////
  vw_Peaks[1] := TBitmap.Create;
  vw_Peaks[1].LoadFromResourceName(HInstance, 'INPUT');
  vw_Peaks[1].Transparent := true;
  vw_Peaks[1].TransparentColor := clBlack;
  vw_Peaks[2] := TBitmap.Create;
  vw_Peaks[2].LoadFromResourceName(HInstance, 'VERSH');
  vw_Peaks[2].Transparent := true;
  vw_Peaks[2].TransparentColor := clBlack;
////////////////////////////////////////////////////////////////
///////////////
//  Screen.Cursors[1]:= LoadCursor(HInstance,'INPUT');
//  Screen.Cursors[2]:= LoadCursor(HInstance,'DELETE');
end;

procedure Tfm_main.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  vw_Peaks[1].Free;
  vw_Peaks[2].Free;
end;

procedure Tfm_main.mi_LoadContextClick(Sender: TObject);
begin
  if OpenDialog.Execute then begin
    FncDesc.LoadFromFile(OpenDialog.FileName);
    mi_New.Enabled := true; mi_Open.Enabled := true;
    Profile := OpenDialog.FileName;
    NARNOST := StrToInt(Copy(FncDesc[0],4,Length(FncDesc[0])-
3));
  end;
end;

procedure Tfm_main.mi_OpenClick(Sender: TObject);
var i: Integer; str : AnsiString;
    Child :Tfm_transfer_function;
begin
```

```
  if not Assigned(dlg_Open) then dlg_Open:=Tdlg_Open.Create
(Self);
  dlg_Open.lb_OpenItems.Items.Clear;
  for i:=0 to FncDesc.Count-1 do
    if Pos('TRANSFUNC ',FncDesc[i])<>0 then begin
      str := FncDesc[i]; delete(str,1,10);
dlg_Open.lb_OpenItems.Items.Add(Trim(str));
    end;
  if dlg_Open.ShowModal = mrOk then begin
    Child := Tfm_transfer_function.Create(Self);
    Child.CreateNew('',1,1,NARNOST,true);
    for i:=0 to FncDesc.Count-1 do
      if Pos('TRANSFUNC '+dlg_Open.lb_OpenItems.Items
[dlg_Open.lb_OpenItems.ItemIndex],FncDesc[i])<>0 then begin
        Child.memo_code.Lines.Clear; Child.memo_code.Lines.Add
(FncDesc[i]);
        repeat
          FncDesc.Delete(i); Child.memo_code.Lines.Add(FncDesc
[i]);
        until FncDesc[i]='END';
        FncDesc.Delete(i); Child.ParseText; exit;
      end;
  end;
end;

procedure Tfm_main.mi_SaveClick(Sender: TObject);
var str : AnsiString; i:Integer;
begin
  (ActiveMDIChild as Tfm_transfer_function).SaveIntoProfile;
  FncDesc.SaveToFile(Profile);
  str:=(ActiveMDIChild as Tfm_transfer_function).memo_code.Lines
[0];
  i:=FncDesc.IndexOf(str); while FncDesc[i]<>'END' do
FncDesc.Delete(i);
  FncDesc.Delete(i);
end;

procedure Tfm_main.mi_TileClick(Sender: TObject);
begin Tile; end;

procedure Tfm_main.mi_CascadeClick(Sender: TObject);
begin Cascade;end;
```

```
procedure Tfm_main.mi_ArrangeAllClick(Sender: TObject);
begin ArrangeIcons; end;

end.
```

ФОРМА ВЫБОРА ЭЛЕМЕНТОВ

```
unit newitems;
interface
uses
  Windows, Messages, SysUtils, Forms, Classes, Controls,
StdCtrls, Buttons, ComCtrls,
  ImgList;

type
  Tfm_new_elements = class(TForm)
    bb_Ok: TBitBtn;   bb_Help: TBitBtn;   bb_Cancel: TBitBtn;
    lv_Items: TListView;  il_Items: TImageList;
    procedure lv_ItemsSelectItem(Sender: TObject; Item:
TListItem;
      Selected: Boolean);
    procedure FormShow(Sender: TObject);
    procedure lv_ItemsDblClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    switch : Integer;
  end;

var
  fm_new_elements : Tfm_new_elements;

implementation

{$R *.DFM}
procedure Tfm_new_elements.lv_ItemsSelectItem(Sender:
TObject;Item: TListItem; Selected: Boolean);
begin
  if Selected then switch := lv_Items.Items.IndexOf(Item);
end;
```

```
procedure Tfm_new_elements.FormShow(Sender: TObject);
begin
  switch := 0; lv_Items.Selected := lv_Items.Items[0];
lv_Items.SetFocus;
end;

procedure Tfm_new_elements.lv_ItemsDblClick(Sender: TObject);
begin
  ModalResult := mrOk;
end;

end.
```

ФОРМА ОПЦИЙ ПЕРЕДАТОЧНЫХ ФУНКЦИЙ

```
unit transfer_function_options;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  RXSpin, ExtCtrls, StdCtrls, RXCtrls, Buttons;

type
  Tfm_trfnc_options = class(TForm)
    bb_Ok: TBitBtn;   bb_Cancel: TBitBtn;
    RxLabel1: TRxLabel;   RxLabel2: TRxLabel; ed_Name: TEdit;
rse_inputs: TRxSpinEdit;   rse_outputs: TRxSpinEdit;
    RxLabel3: TRxLabel;
    Bevel1: TBevel;

    procedure FormShow(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
    procedure Reset;
    procedure SetValues(name : AnsiString; inp,outp,nar: Word);
  end;

var
```

```
  fm_trfnc_options: Tfm_trfnc_options;

implementation

{$R *.DFM}

procedure Tfm_trfnc_options.Reset;
begin
  SetValues('Noname',1,1,2);
end;

procedure Tfm_trfnc_options.SetValues(name : AnsiString;
inp,outp,nar: Word);
begin
  ed_Name.Text := name;
  rse_inputs.Value := inp;
  rse_outputs.Value := outp;
end;

procedure Tfm_trfnc_options.FormShow(Sender: TObject);
begin ed_Name.SetFocus; end;

end.
```

ФОРМА ПАРАМЕТРОВ РЕБРА

```
unit newedge;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Forms,
StdCtrls, RXSpin,
  Controls, Buttons, RXCtrls;

type
  Tdlg_Edge = class(TForm)
    RxLabel1: TRxLabel;
    RxLabel2: TRxLabel;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    RxSpinEdit1: TRxSpinEdit;
```

```
    RxSpinEdit2: TRxSpinEdit;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  dlg_Edge: Tdlg_Edge;

implementation

{$R *.DFM}

end.
```

ФОРМА ОПЦИЙ ВЕРШИНЫ

```
unit newPeak;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons, RXCtrls;

type
  Tdlg_Insert_peak = class(TForm)
    ed_Name: TEdit;
    ed_Type: TEdit;
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    RxLabel1: TRxLabel;
    RxLabel2: TRxLabel;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  dlg_Insert_peak: Tdlg_Insert_peak;
```

```
implementation

{$R *.DFM}

end.
```

ФОРМА ПЕРЕДАТОЧНЫХ ФУНКЦИЙ

```
unit transfer_functions;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs, Operations,
  Math, ComCtrls, ImgList, StdCtrls, Menus, ExtCtrls, RXCtrls,
Grids, NewEdge, NewPeak,
  transfer_function_options,transfer_test;

type
//Peak
  TPeak = class
  public
     Name : AnsiString;
     eType: AnsiString;
     x,y,N: LongInt;
     constructor Create(aName,aeType:AnsiString;
ax,ay,an:LongInt);
  end;
//Edge
  TEdge = class
  public
     p1,p2: Word;
     constructor Create(ap1,ap2: Word);
  end;
//Form
  Tfm_transfer_function = class(TForm)
    pc_Pages: TPageControl;  ts_Graph: TTabSheet;  ts_Table:
TTabSheet;  ts_Editor: TTabSheet; mm_trfFnc: TMainMenu;
mdd_Edit: TMenuItem; mdd_View: TMenuItem;
```

```
    memo_code: TMemo;  sb_Graph: TScrollBox; sp_Graph:
TSecretPanel;  sg_Reference: TStringGrid;
    mi_Insertpeak: TMenuItem; mi_Deletepeak: TMenuItem; N1:
TMenuItem;  mi_Properties: TMenuItem;
    N2: TMenuItem;  mdd_Functionas: TMenuItem;  mi_Graph:
TMenuItem;  mi_Table: TMenuItem;
    mi_GraphSel: TMenuItem;  mi_TableSel: TMenuItem;
mi_EditorSel: TMenuItem;  mi_Cut: TMenuItem;
    mi_Copy: TMenuItem;  mi_Paste: TMenuItem;  N3: TMenuItem;
mi_TestWindow: TMenuItem;ilPages: TImageList;

    procedure FormResize(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action:
TCloseAction);
    procedure sp_GraphPaintClient(Sender: TObject; Canvas:
TCanvas; Rect: TRect);
    procedure pc_PagesChange(Sender: TObject);
    procedure sp_GraphMouseDown(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure sp_GraphMouseMove(Sender: TObject; Shift:
TShiftState; X,Y: Integer);
    procedure sp_GraphMouseUp(Sender: TObject; Button:
TMouseButton; Shift: TShiftState; X, Y: Integer);
    procedure mi_DeletepeakClick(Sender: TObject);
    procedure mi_InsertpeakClick(Sender: TObject);
    procedure mi_PropertiesClick(Sender: TObject);
    procedure mi_GraphSelClick(Sender: TObject);
    procedure mi_TableSelClick(Sender: TObject);
    procedure mi_EditorSelClick(Sender: TObject);
    procedure mi_GraphClick(Sender: TObject);
    procedure mi_TableClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure FormDeactivate(Sender: TObject);
    procedure sg_ReferenceSetEditText(Sender: TObject; ACol,
ARow: Integer; const Value: String);
    procedure mi_TestWindowClick(Sender: TObject);
  private
    { Private declarations }
    trfFncName : AnsiString;
    inputs, outputs , N : Word;
    is_graph : boolean;
```

```
    ////////////////////////////////////////////////////////
////////////////
    Peaks : TList; Edges : TList;
    ////////////////////////////////////////////////////////
////////////////
    buffer : Graphics.TBitmap;  del_state : byte; NewName,
NewType : AnsiString;
    SelPeak : Integer; old_line : TPoint;
  public
    { Public declarations }
    procedure CreateNew(name: AnsiString; inp, outp, nary :
Word; graph : boolean);
    procedure CreateGraphTemplate;
    procedure CreateTableTemplate;
    procedure ParseText;
    function  IsCorrect : boolean;
    procedure SaveIntoProfile;
    procedure DrawOnBuffer(rt : TRect);
  end;

var
  vw_Peaks : array [1..2] of Graphics.TBitmap;


implementation

uses main;

{$R *.DFM}
////////////////////////////////////////////////////////
////////////////
constructor TPeak.Create(aName,aeType:AnsiString; ax,ay,an :
LongInt);
begin Name := aName; eType := aeType; x:=ax; y:=ay; N := an;
end;

constructor TEdge.Create(ap1,ap2: Word);
begin p1 := ap1; p2 := ap2; end;

function GetMaxXFrom(list : TList) : LongInt;
var x,i : LongInt;
begin
  x := 0;
```

```
  for i:=0 to list.Count-1 do
    if x < TPeak(list.Items[i]).x then x := TPeak(list.Items
[i]).x;
  Result := x;
end;

function GetMaxYFrom(list : TList) : LongInt;
var x,i : LongInt;
begin
  x := 0;
  for i:=0 to list.Count-1 do
    if x < TPeak(list.Items[i]).y then x := TPeak(list.Items
[i]).y;
  Result := x;
end;

function IndexOfName(list : TList; str : AnsiString): Word;
var i: word;
begin
  i:=0; while TPeak(list.Items[i]).Name <> str do inc(i);
  Result := i;
end;
////////////////////////////////////////////////////////////
///////////////
procedure Tfm_transfer_function.CreateNew(name: AnsiString; inp,
outp, nary : Word; graph : boolean);
var
  i,j : Word; str : AnsiString;
begin
  trfFncName := name; Caption := name; is_graph := graph;
  inputs := inp; outputs := outp; N := nary;
  if not(is_graph) then begin
    //If Table
    sg_Reference.ColCount := inp+outp; sg_Reference.RowCount :=
Round(IntPower(N,inp))+1; sg_Reference.FixedCols := inp;
    for i:=0 to inp-1 do sg_Reference.Cells[i,0] :=
'IN'+IntToStr(i);
    for i:=0 to outp-1 do sg_Reference.Cells[i+inp,0] :=
'OUT'+IntToStr(i);
    for i:=0 to sg_Reference.RowCount do begin
      str := GetNaryView(i,N); while Length(str)<inp do str :=
'0'+str;
      for j:=1 to inp do sg_Reference.Cells[j-1,i+1] := str[j];
```

```
      for j:=1 to outp do sg_Reference.Cells[inp+j-1,i+1] :=
'0';
    end;
    CreateTableTemplate;
  end
  else begin
    //If Graph
    CreateGraphTemplate;
  end;
  ParseText;
end;

procedure Tfm_transfer_function.CreateGraphTemplate;
begin
  memo_code.Lines.BeginUpdate;
  memo_code.Lines.Clear;
  memo_code.Lines.Add('TRANSFUNC '+trfFncName+'('+IntToStr
(inputs)+','+IntToStr(outputs)+')');
  memo_code.Lines.Add('ELEMENTS');
  memo_code.Lines.Add('RELATIONS');
  memo_code.Lines.Add('END');
  memo_code.Lines.EndUpdate;
end;

procedure Tfm_transfer_function.CreateTableTemplate;
var i,j: Integer; str: AnsiString;
begin
  memo_code.Lines.BeginUpdate;
  memo_code.Lines.Clear;
  memo_code.Lines.Add('TRANSFUNC '+trfFncName+'('+IntToStr
(inputs)+','+IntToStr(outputs)+')');
  memo_code.Lines.Add('TABLE');
  for i:=1 to sg_Reference.RowCount-1 do begin
    str:='0'; for j:=2 to outputs do str:=str+' 0';
    memo_code.Lines.Add(str);
  end;
  memo_code.Lines.Add('END');
  memo_code.Lines.EndUpdate;
end;

procedure Tfm_transfer_function.ParseText;
var
  i,j : Integer; str,str1,str2,strx,stry : AnsiString;
```

```
begin
  //If correct
  if IsCorrect then begin
     str := memo_code.Lines[0];  Delete(str,1,Pos(' ',str));
     trfFncName := Trim(Copy(str,1,Pos('(',str)-1)); Caption :=
trfFncName;
     Delete(str,1,Pos('(',str)); inputs := StrToInt(Trim(Copy
(str,1,Pos(',',str)-1)));
     Delete(str,1,Pos(',',str)); outputs:= StrToInt(Trim(Copy
(str,1,Pos(')',str)-1)));
     str := Trim(memo_code.Lines[1]);
     if str='ELEMENTS' then begin if not is_graph then begin
        ts_Table.TabVisible := false; ts_Graph.TabVisible :=
true;
        mi_Graph.Checked := true; mi_GraphSel.Visible := true;
mi_TableSel.Visible := false;
        mi_GraphSelClick(Self);
        is_graph := true;
       end;
     end
     else if is_graph then begin
        ts_Table.TabVisible := true; ts_Graph.TabVisible :=
false;
        mi_Table.Checked := true; mi_GraphSel.Visible := false;
mi_TableSel.Visible := true;
        mi_TableSelClick(Self);
        is_graph := false;
        sg_Reference.ColCount := inputs +outputs;
sg_Reference.RowCount := Round(IntPower(N,inputs))+1;
sg_Reference.FixedCols := inputs;
        for i:=0 to inputs-1 do sg_Reference.Cells[i,0] :=
'IN'+IntToStr(i);
        for i:=0 to outputs-1 do sg_Reference.Cells
[i+inputs,0] := 'OUT'+IntToStr(i);
        for i:=0 to sg_Reference.RowCount do begin
          str := GetNaryView(i,N); while Length(str)<inputs do
str := '0'+str;
          for j:=1 to inputs do sg_Reference.Cells[j-1,i+1] :=
str[j];
          for j:=1 to outputs do sg_Reference.Cells[inputs+j-
1,i+1] := '0';
        end;
     end;
     //If Graph
```

```
     if is_graph then begin
       Peaks.Clear; Edges.Clear;
       //Inputs and Outputs
       for i:=1 to inputs do Peaks.Add(TPeak.Create
('in'+IntToStr(i-1),'INPUT',10,i*40,1));
       for i:=1 to outputs do Peaks.Add(TPeak.Create
('out'+IntToStr(i-1),'OUTPUT',sb_Graph.Width-26,i*40,1));
       i := 2; str := memo_code.Lines[i];
       //Peaks
       while(str<>'RELATIONS') do begin
         str1 := Trim(Copy(str,1,Pos(' ',str)-1)); Delete
(str,1,Pos(' ',str)); str := Trim(str);
         while str<>'' do begin
            str2 := Trim(Copy(str,1,Pos('(',str)-1)); Delete
(str,1,Pos('(',str));
            strx := Trim(Copy(str,1,Pos(',',str)-1)); Delete
(str,1,Pos(',',str));
            stry := Trim(Copy(str,1,Pos(')',str)-1)); Delete
(str,1,Pos(')',str));
            Peaks.Add(TPeak.Create(str2, str1, StrToInt(strx),
StrToInt(stry),2));
            Delete(str,1,Pos(',',str)); str := Trim(str);
         end;
         inc(i); str := memo_code.Lines[i];
       end;
       inc(i); str := memo_code.Lines[i];
       //Edges
       while(str<>'END') do begin
         str1 := Trim(Copy(str,1,Pos('->',str)-1));Delete
(str1,Pos('.',str1),Length(str1)-Pos('.',str1)+1);
         Delete(str,1,Pos('->',str)+1);str := Trim(str);Delete
(str,Pos('.',str),Length(str)-Pos('.',str)+1);
         Edges.Add(TEdge.Create(IndexOfName(Peaks,str1),
IndexOfName(Peaks,str)));
         inc(i); str := memo_code.Lines[i];
       end;
       sp_Graph.Invalidate;
     end
     //If Table
     else begin
       for i:=1 to sg_Reference.RowCount-1 do begin
            str := memo_code.Lines[i+1];
            for j:=1 to outputs do begin
```

```
                if Pos(' ',str)>0 then begin
                  str1 := Copy(str,1,Pos(' ',str)-1); Delete
(str,1,Pos(' ',str));
                end else str1 := str;
                sg_Reference.Cells[j+inputs-1,i]:=str1;
            end;
       end;
     end;
  end;
end;

function  Tfm_transfer_function.IsCorrect : boolean;
begin
  if memo_code.Lines.Count < 3 then Result := false
  else  Result := true;
end;

procedure Tfm_transfer_function.DrawOnBuffer(rt : TRect);
var
  k,j,i,x1,x2,y1,y2,xb1,xb2,yb1,yb2,xg1,xg2,yg1,yg2 : Integer;
  alfa,betta,gamma : Extended;
begin
  with buffer.Canvas do begin
    Brush.Color := clBlack;
    Pen.Color := clTeal;
    Pen.Width := 1;
    Font.Color := clLime;
    SetBkMode(Handle,TRANSPARENT);
    FillRect(rt);
    for i:=0 to Edges.Count-1 do begin
      j := TEdge(Edges.Items[i]).p1; k := TEdge(Edges.Items[i]).
p2;
      x1 := TPeak(Peaks.Items[j]).x; x2 := TPeak(Peaks.Items
[k]).x;
      y1 := TPeak(Peaks.Items[j]).y; y2 := TPeak(Peaks.Items
[k]).y;
      MoveTo(x1,y1);LineTo(x2,y2);
      TextOut(x1+(x2-x1)div 2,y1+(y2-y1)div 2,IntToStr(i+1));
      //strelki
      if x1<>x2 then alfa := arctan((y1-y2)/(x1-x2)) else begin
if y1<y2 then alfa := pi/2 else alfa := -pi/2; end;
      if x2<x1 then alfa := alfa+pi;
      betta := alfa+10*pi/180;  gamma:=alfa-10*pi/180;
```

```
      xb1:=Round((2.0/3.0)*(x2-x1)+x1); yb1:=Round((2.0/3.0)*
(y2-y1)+y1);
      xb2:=Round(xb1-15*cos(betta));    yb2:=Round(yb1-15*sin
(betta));
      xg1:=Round((2.0/3.0)*(x2-x1)+x1); yg1:=Round((2.0/3.0)*
(y2-y1)+y1);
      xg2:=Round(xg1-15*cos(gamma));    yg2:=Round(yg1-15*sin
(gamma));
      Brush.Color := clTeal; Polygon([Point(xb2,yb2),Point
(xb1,yb1),Point(xg2,yg2)]); Brush.Color := clBlack;
    end;
    for i:=0 to Peaks.Count-1 do begin
        Draw(TPeak(Peaks.Items[i]).x-8,TPeak(Peaks.Items[i]).y-
8,vw_Peaks[TPeak(Peaks.Items[i]).N]);
        SetBkMode(Handle,TRANSPARENT);TextOut(TPeak(Peaks.Items
[i]).x,TPeak(Peaks.Items[i]).y-16,TPeak(Peaks.Items[i]).Name);
    end;
  end;
end;

procedure Tfm_transfer_function.SaveIntoProfile;
var i:Integer;
begin
  with fm_main do begin
    if FncDesc.IndexOf(memo_code.Lines[0])<>-1 then begin
    end;
    for i:=0 to memo_code.Lines.Count-1 do
      FncDesc.Add(memo_code.Lines[i]);
  end;
end;

procedure Tfm_transfer_function.FormResize(Sender: TObject);
var w,h : LongInt;
begin
  ParseText;
  w := GetMaxXFrom(Peaks);
  h := GetMaxYFrom(Peaks);
  if (w>sb_Graph.Width)or(h>sb_Graph.Height) then begin
    sp_Graph.Align := alNone;
    sp_Graph.Width  := w + 16;
    sp_Graph.Height := h + 16;
  end
  else sp_Graph.Align := alClient;
```

```
end;

procedure Tfm_transfer_function.FormCreate(Sender: TObject);
begin
  SelPeak := -1; del_state := 0;
  Peaks := TList.Create;
  Edges := TList.Create;
  buffer :=TBitmap.Create;
  FormResize(Self);
end;

procedure Tfm_transfer_function.FormClose(Sender: TObject; var
Action: TCloseAction);
begin
  SaveIntoProfile;
  buffer.Free;
  Edges.Free; Peaks.Free;
  Action := caFree;
end;

procedure Tfm_transfer_function.sp_GraphPaintClient(Sender:
TObject; Canvas: TCanvas; Rect: TRect);
var
  rt : TRect;
begin
//Draw
  buffer.Width := sp_Graph.Width; buffer.Height :=
sp_Graph.Height;
  rt.Left := 0; rt.Top := 0; rt.Right := sp_Graph.Width;
rt.Bottom := sp_Graph.Height;
  DrawOnBuffer(rt);
  Canvas.CopyRect(rt,buffer.Canvas,rt);
end;

procedure Tfm_transfer_function.pc_PagesChange(Sender: TObject);
begin ParseText; end;

procedure Tfm_transfer_function.sp_GraphMouseDown(Sender:
TObject; Button: TMouseButton; Shift: TShiftState; X, Y:
Integer);
var
  i,k,j, x1, y1 : Longint;
  str : AnsiString;
```

```
begin
  if del_state=2 then begin
  //Insert peak To Elements
    k:=2; while Trim(memo_code.Lines[k])<>'RELATIONS' do begin
      if Trim(Copy(memo_code.Lines[k],1,Pos(' ',memo_code.Lines
[k])-1)) = NewType then begin
         memo_code.Lines[k] := memo_code.Lines[k] +
','+NewName+'('+IntToStr(X)+','+IntToStr(Y)+')';
         ParseText; del_state := 0;  exit;
      end;
      inc(k);
    end;
    memo_code.Lines.Insert(k,NewType + ' ' +
NewName+'('+IntToStr(X)+','+IntToStr(Y)+')');
    ParseText; del_state := 0; sp_Graph.Cursor := crArrow; exit;
  end;
  if(SelPeak=-1) then begin
        //Select new peak
        for i:=0 to Peaks.Count-1 do begin
           x1 := TPeak(Peaks.Items[i]).x; y1 := TPeak
(Peaks.Items[i]).y;
           if(sqrt((x1-X)*(x1-X)+(y1-Y)*(y1-Y))<16) then
               if del_state=0 then begin
                 SelPeak := i; old_line.x := x1; old_line.y :=
y1;
                 exit;
               end
               else if del_state=1 then begin
                 //delete peak from Elements
                 k:=2; while Trim(memo_code.Lines[k])
<>'RELATIONS' do begin
                    if Pos(TPeak(Peaks.Items[i]).
Name,memo_code.Lines[k])>0 then begin
                      str := memo_code.Lines[k]; j:=Pos(TPeak
(Peaks.Items[i]).Name,memo_code.Lines[k]);
                      while str[j]<>')' do Delete(str,j,1);
Delete(str,j,1); Delete(str,j,1);
                      if Pos(' ',str) = Length(str) then
memo_code.Lines.Delete(k)
                      else memo_code.Lines[k] := str;
                    end
                    else inc(k);
                 end;
```

```
                  //delete peak relations
                  inc(k); while memo_code.Lines[k]<>'END' do
begin
                    if Pos(TPeak(Peaks.Items[i]).
Name,memo_code.Lines[k])>0 then memo_code.Lines.Delete(k)
                    else inc(k);
                  end;
                  del_state := 0; ParseText;sp_Graph.Cursor :=
crArrow; exit;
               end
        end;
        SelPeak := -1;
  end
  else begin
        //New edge
        for i:=0 to Peaks.Count-1 do begin
           x1 := TPeak(Peaks.Items[i]).x;  y1 := TPeak
(Peaks.Items[i]).y;
           if(sqrt((x1-X)*(x1-X)+(y1-Y)*(y1-Y))<16)then begin
                  if not(Assigned(dlg_Edge))then dlg_Edge :=
Tdlg_Edge.Create(Self);
                  if dlg_Edge.ShowModal=mrOk then with dlg_Edge
do begin
                    str := TPeak(Peaks.Items[SelPeak]).Name
+'.'+IntToStr(Round(RxSpinEdit2.Value)) +
                       '->' +TPeak(Peaks.Items[i]).Name +'.'+
IntToStr(Round(RxSpinEdit1.Value));
                    if memo_code.Lines.IndexOf(str)<>-1 then
                       memo_code.Lines.Delete
(memo_code.Lines.IndexOf(str))
                    else memo_code.Lines.Insert
(memo_code.Lines.Count-1,str);
                  end;
                  break;
           end
        end;
        ParseText;
        SelPeak := -1;
    end
end;

procedure Tfm_transfer_function.sp_GraphMouseMove(Sender:
TObject; Shift: TShiftState; X, Y: Integer);
```

```
begin
//If vershina not selected
    if(SelPeak=-1) then exit;

    if(ssLeft in Shift) then begin
        //if peretaskivanie
        TPeak(Peaks.Items[SelPeak]).x := X;
        TPeak(Peaks.Items[SelPeak]).y := Y;
        sp_Graph.Invalidate;
    end
    else begin
        // vedenie rebra
        with sp_Graph.Canvas do begin
          Pen.Mode := pmXor;Pen.Color := clTeal;Pen.Width := 1;
          MoveTo(TPeak(Peaks.Items[SelPeak]).x,TPeak(Peaks.Items
[SelPeak]).y); LineTo(old_line.x,old_line.y);
          old_line.x := X; old_line.y := Y;
          MoveTo(TPeak(Peaks.Items[SelPeak]).x,TPeak(Peaks.Items
[SelPeak]).y); LineTo(old_line.x,old_line.y);
          Pen.Mode := pmCopy;
        end;
    end;
end;

procedure Tfm_transfer_function.sp_GraphMouseUp(Sender: TObject;
Button: TMouseButton; Shift: TShiftState; X, Y: Integer);
var
  i, k : Integer;
  str,strF,strT : AnsiString;
begin
  if (SelPeak=-1)or(Button= mbRight) then exit;

  i:=2; strF := TPeak(Peaks.Items[SelPeak]).Name;
  while not(memo_code.Lines.Strings[i]='RELATIONS')and
(memo_code.Lines.Count>i) do begin
     str := memo_code.Lines.Strings[i]; StrT:=Copy(str,1,Pos('
',str)); Delete(str,1,Pos(' ',str));
     if(Pos(strF,str)>0)then begin
        k := Pos(strF,str);
        while not(str[k]=')')do Delete(str,k,1); Delete
(str,k,1);
        Insert(strF +'('+ IntToStr(X) +','+ IntToStr(Y)
+')',str,k);
```

```
        memo_code.Lines.Strings[i] := StrT+str;
        break;
     end;
     inc(i);
  end;
  SelPeak := -1;
  FormResize(Self);
end;

procedure Tfm_transfer_function.mi_DeletepeakClick(Sender:
TObject);
begin sp_Graph.Cursor := 2; del_state := 1; end;

procedure Tfm_transfer_function.mi_InsertpeakClick(Sender:
TObject);
begin
  if not Assigned(dlg_Insert_peak) then dlg_Insert_peak :=
Tdlg_Insert_peak.Create(Self);
  if dlg_Insert_peak.ShowModal = mrOk then begin
    sp_Graph.Cursor := 1;
    NewName := dlg_Insert_peak.ed_Name.Text; NewType :=
dlg_Insert_peak.ed_Type.Text; del_state := 2;
  end;
end;

procedure Tfm_transfer_function.mi_PropertiesClick(Sender:
TObject);
begin
  if not Assigned(fm_trfnc_options) then fm_trfnc_options :=
Tfm_trfnc_options.Create(Self);
  if fm_trfnc_options.ShowModal=mrOk then
    if is_graph then begin
      trfFncName:=fm_trfnc_options.ed_Name.Text; inputs:=Round
(fm_trfnc_options.rse_inputs.Value);  outputs:=Round
(fm_trfnc_options.rse_outputs.Value);
      memo_code.Lines[0] := 'TRANSFUNC '+trfFncName+'('+IntToStr
(inputs)+','+IntToStr(outputs)+')';
      ParseText;
    end
    else
end;
```

```
procedure Tfm_transfer_function.mi_GraphSelClick(Sender:
TObject);
begin
  pc_Pages.ActivePage := ts_Graph;
  mi_GraphSel.Checked := true;
  mi_TableSel.Checked := false;
  mi_EditorSel.Checked := false;
end;

procedure Tfm_transfer_function.mi_TableSelClick(Sender:
TObject);
begin
  pc_Pages.ActivePage := ts_Table;
  mi_GraphSel.Checked := false;
  mi_TableSel.Checked := true;
  mi_EditorSel.Checked := false;
end;

procedure Tfm_transfer_function.mi_EditorSelClick(Sender:
TObject);
begin
  pc_Pages.ActivePage := ts_Editor;
  mi_GraphSel.Checked := false;
  mi_TableSel.Checked := false;
  mi_EditorSel.Checked := true;
end;

procedure Tfm_transfer_function.mi_GraphClick(Sender: TObject);
begin
  is_graph := true; ts_Table.TabVisible := false;
ts_Graph.TabVisible := true;
  mi_Graph.Checked := true; mi_GraphSel.Visible := true;
mi_TableSel.Visible := false;
  CreateNew(trfFncName,inputs,outputs,N,true);
  mi_GraphSelClick(Sender);
end;

procedure Tfm_transfer_function.mi_TableClick(Sender: TObject);
begin
  is_graph := false; ts_Table.TabVisible := true;
ts_Graph.TabVisible := false;
  mi_Table.Checked := true; mi_GraphSel.Visible := false;
mi_TableSel.Visible := true;
```

```
  CreateNew(trfFncName,inputs,outputs,N,false);
  mi_TableSelClick(Sender);
end;

procedure Tfm_transfer_function.FormActivate(Sender: TObject);
begin  fm_main.mi_Save.Enabled := true; end;

procedure Tfm_transfer_function.FormDeactivate(Sender: TObject);
begin fm_main.mi_Save.Enabled := false; end;

procedure Tfm_transfer_function.sg_ReferenceSetEditText(Sender:
TObject; ACol, ARow: Integer; const Value: String);
var
  new_str,str : AnsiString;
  i : integer;
begin
  if Value='' then exit;
  str := memo_code.Lines[ARow+1];
  for i:=1 to ACol-inputs do begin
    new_str:=new_str+Copy(str,1,Pos(' ',str));
    Delete(str,1,Pos(' ',str));
  end;
  new_str := new_str+Trim(Value); Delete(str,1,Pos(' ',str)-1);
  if Pos(' ',str)=0 then str:='';  new_str := new_str+str;
  memo_code.Lines[ARow+1] := new_str;
end;

procedure Tfm_transfer_function.mi_TestWindowClick(Sender:
TObject);
var i: Integer;
begin
  if not Assigned(fm_test_and_time) then
     fm_test_and_time :=Tfm_test_and_time.Create(fm_main);
  fm_test_and_time.Caption := 'Testing - '+Caption;
  fm_test_and_time.inp := inputs; fm_test_and_time.outp :=
outputs;
  fm_test_and_time.N := N; fm_test_and_time.CurrTF.Clear;
  for i:=1 to memo_code.Lines.Count do
    fm_test_and_time.CurrTF.Add(memo_code.Lines[i-1]);
  fm_test_and_time.ShowModal;
end;

end.
```

ФОРМА ТЕСТА ПЕРЕДАТОЧНОЙ ФУНКЦИИ

```
unit transfer_test;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  ExtCtrls, TeeProcs, TeEngine, Chart, Grids, StdCtrls, Buttons,
Series, Operations;

type
  Tfm_test_and_time = class(TForm)
    sg_Patterns: TStringGrid;   chrt_timeDiagram: TChart;
    Panel1: TPanel;  BitBtn1: TBitBtn;  BitBtn2: TBitBtn;
BitBtn3: TBitBtn;

    procedure FormCreate(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure BitBtn1Click(Sender: TObject);
    procedure BitBtn2Click(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action:
TCloseAction);
    procedure FormDestroy(Sender: TObject);
    procedure FormResize(Sender: TObject);
  private
    { Private declarations }
    FileList : TStringList;
  public
    { Public declarations }
    inp,outp,N : Integer;
    CurrTF : TStringList;

    function Process(fnc_name,param:AnsiString):AnsiString;
  end;

var
  fm_test_and_time: Tfm_test_and_time;

implementation
```

```
uses main;

{$R *.DFM}

function Tfm_test_and_time.Process(fnc_name,param : AnsiString):
AnsiString;
var
  cursor,types,f_list,in_list : TStringList;
  i,j,inp,outp,err : integer; res,msg,str,str_1,str_2,str_3 :
AnsiString;
begin
  cursor:=TStringList.Create; f_list:= TStringList.Create;
  in_list:= TStringList.Create;  types:= TStringList.Create;
  ////////////////////////////////////////////////////////
//Ищем и копируем
  //Access denied
//Если граф
  //Получить список используемых функций
  //Access denied
  //Сканирование
  //Access denied
  //Если вход то получить его
  //Иначе рекурсивный спуск
  //Access denied
  //Если выход то записать в него
  //Access denied
  //Иначе записать в параметр
  //Access denied
//Если таблица
  //Access
denied ////////////////////////////////////////////////////
/////
  f_list.Free;
  cursor.Free;
  in_list.Free;
  types.Free;
  Process := res;
end;

procedure Tfm_test_and_time.FormCreate(Sender: TObject);
begin
  FileList := TStringList.Create;
  CurrTF := TStringList.Create;
```

```
end;

procedure Tfm_test_and_time.FormShow(Sender: TObject);
var i:integer; ser: TLineSeries;
begin
  FileList.LoadFromFile(fm_main.Profile);
  for i:=1 to CurrTF.Count do
    FileList.Insert(i-1,CurrTF[i-1]);
////////////////////////////////////////
  for i:=1 to inp do begin
    ser := TLineSeries.Create(Self); ser.LinePen.Color := RGB
(Random(255),Random(255),Random(255));
    ser.LinePen.Width :=2; ser.Title := 'In'+IntToStr(i-1);
    chrt_timeDiagram.AddSeries(ser);
  end;
  for i:=1 to outp do begin
    ser := TLineSeries.Create(Self); ser.LinePen.Color := RGB
(Random(255),Random(255),Random(255));
    ser.LinePen.Width :=2; ser.Title := 'Out'+IntToStr(i-1);
    chrt_timeDiagram.AddSeries(ser);
  end;
  sg_Patterns.ColCount := inp; sg_Patterns.RowCount := 2;
  for i:=1 to inp do begin sg_Patterns.Cells[i-1,0]:
='In'+IntToStr(i-1); sg_Patterns.Cells[i-1,1]:='0' end;
end;

procedure Tfm_test_and_time.BitBtn1Click(Sender: TObject);
var i:integer;
begin
  sg_Patterns.RowCount := sg_Patterns.RowCount + 1;
  for i:=1 to inp do sg_Patterns.Cells[i-1,sg_Patterns.RowCount-
1]:='0';
end;

procedure Tfm_test_and_time.BitBtn2Click(Sender: TObject);
var
  i,j : Integer; str,str1,str2 : AnsiString;
begin
//Inputs
  for i:=1 to inp do chrt_timeDiagram.SeriesList[i-1].Clear;
  for i:=1 to inp do begin
    for j:=1 to sg_Patterns.RowCount-1 do begin
```

```
      chrt_timeDiagram.SeriesList[i-1].AddXY(j-1,StrToInt
(sg_Patterns.Cells[i-1,j])-i*N,'',clDefault);
      chrt_timeDiagram.SeriesList[i-1].AddXY(j,  StrToInt
(sg_Patterns.Cells[i-1,j])-i*N,'',clDefault);
    end;
  end;
  str := Caption; Delete(str,1,Pos('-',str)+1);
  //Outputs
  for i:=inp+1 to outp+inp do chrt_timeDiagram.SeriesList[i-1].
Clear;
  for j:=1 to sg_Patterns.RowCount-1 do begin
    str2:=''; for i:=1 to sg_Patterns.ColCount do
str2:=str2+sg_Patterns.Cells[i-1,j];
    str1:=Process(str,str2);
    for i:=inp+1 to outp+inp do begin
      chrt_timeDiagram.SeriesList[i-1].AddXY(j-1,StrToInt(str1
[i-inp])-i*N,'',clDefault);
      chrt_timeDiagram.SeriesList[i-1].AddXY(j,  StrToInt(str1
[i-inp])-i*N,'',clDefault);
    end;
  end;
end;

procedure Tfm_test_and_time.FormClose(Sender: TObject; var
Action: TCloseAction);
begin
  chrt_timeDiagram.SeriesList.Clear;
end;

procedure Tfm_test_and_time.FormDestroy(Sender: TObject);
begin  FileList.Free; CurrTF.Free; end;

procedure Tfm_test_and_time.FormResize(Sender: TObject);
begin BitBtn3.Left := Panel1.Width-BitBtn3.Width-10;  end;

end.
```

ФОРМА ОТКРЫТИЯ ЭЛЕМЕНТА

```
unit OpenItem;

interface
```

```
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs,
  StdCtrls, Buttons;

type
  Tdlg_Open = class(TForm)
    BitBtn1: TBitBtn;
    BitBtn2: TBitBtn;
    lb_OpenItems: TListBox;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  dlg_Open: Tdlg_Open;

implementation

{$R *.DFM}

end.
```

БАЗОВАЯ МАТЕМАТИКА.

```
unit Operations;

interface

const
  alphabet: array[0..35] of char =
('0','1','2','3','4','5','6','7','8','9',
  'a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p
','q','r',
  's','t','u','v','w','x','y','z');

{Перевод между система счисления}
function GetNaryView(i,N:Integer):ANSIString;
function Get10View(str:ANSIString;N:Integer):Integer;

implementation
```

```
function GetNaryView(i,N:Integer):ANSIString;
var
  r : AnsiString; k,ost : word;
begin
  K:=i;
  while K<>0 do begin
    ost:=K mod N;
    r:=alphabet[ost]+r;
    K:=K div N;
  end;
  Result := r;
end;

function Get10View(str:ANSIString;N:Integer):Integer;
var
  i,j,sum,st : word;
begin
  sum := 0; st := 1;
  for i:=Length(str) downto 1 do begin
    j:=0; while str[i]<>alphabet[j] do inc(j);
    sum := sum + j*st;  st := st*N;
  end;
  Result := sum;
end;

end.
```