МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Вятский государственный университет»**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

Допущено к защите

Руководитель проекта

\_\_\_\_\_\_\_\_/Караваева О.В./

(подпись) (Ф.И.О)

«\_\_»\_\_\_\_\_\_\_\_\_\_2023г.

**Разработка программного эмулятора процессорного элемента систолической системы**

Пояснительная записка курсового проекта по дисциплине

«Комплекс знаний бакалавра в области программного и аппаратного обеспечения вычислительной техники»

ТПЖА.09.03.01.285 ПЗ

Разработал студент группы ИВТ-43 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Пестов М.А./

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Караваева О.В./

Консультант \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Крутиков А.К./

Работа защищена с оценкой «\_\_\_\_\_\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(оценка) (дата)

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ \_\_\_\_\_\_\_\_\_\_\_/

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ /\_\_\_\_\_\_\_\_\_\_\_\_/

(подпись)

Киров 2023

РЕФЕРАТ

Пестов М.А.. Разработка программного эмулятора процессорного элемента систолической системы. ТПЖА.090301.285 ПЗ: Курс. проект / ВятГУ, каф. ЭВМ; рук. Караваева О.В. - Киров, 2023. – ПЗ 43с, 29 рис., 4 прил.

Объект исследования – специализированные вычислительные системы с систолической архитектурой.

Предмет курсового проекта – **эмулятор** процессорного элементаспециализированной систолической вычислительной системы.

Цель курсового проекта – повышение качества обучения по дисциплине «Организация ЭВМ» за счёт разработки программного эмулятора процессорного модуля систолической вычислительной системы.

Результатом выполнения курсового проекта является программный эмулятор систолической вычислительной системы.

Содержание

[Введение 4](#_Toc135039741)

[1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ 5](#_Toc135039742)

[1.1 Общие сведения о систолических системах 6](#_Toc135039743)

[1.2 Обзор аналогов 9](#_Toc135039744)

[1.2.1 Структура процессора Intel Itanium 2 9](#_Toc135039745)

[1.2.2 Структура процессора AMD Athlon 10](#_Toc135039746)

[1.2.4 Универсальный лабораторный стенд 12](#_Toc135039747)

[1.3 Расширенное техническое задание 13](#_Toc135039748)

[1.3.1 Полное наименование системы и её условное обозначение 13](#_Toc135039749)

[1.3.2 Наименование разработчика системы и реквизиты заказчика 13](#_Toc135039750)

[1.3.3 Основания для создания системы 13](#_Toc135039751)

[1.3.4 Плановые сроки начала и окончания работы по созданию системы 13](#_Toc135039752)

[1.3.5 Источник финансирования работ по созданию системы. 13](#_Toc135039753)

[1.3.6 Порядок оформления и предъявления заказчику результатов работ по созданию системы: 13](#_Toc135039754)

[1.3.7 Назначение и цели создания системы 14](#_Toc135039755)

[1.3.9 Требования к системе 14](#_Toc135039756)

[1.3.9.1 Требования к системе в целом 14](#_Toc135039757)

[1.3.9.2 Требование к видам обеспечения. 15](#_Toc135039758)

[1.4 Выводы по первому разделу 16](#_Toc135039759)

[2. Разработка структуры процессорного элемента систолической вычислительной системы 17](#_Toc135039760)

[2.2 Выводы по разделу 2 20](#_Toc135039761)

[3. Разработка структуры программного эмулятора 21](#_Toc135039762)

[**3.1 Разработка общей структуры программного эмулятора** 21](#_Toc135039763)

[**3.2 Разработка окна отладки программы и построенной системы** 21](#_Toc135039764)

[**3.2 Разработка окна составления системы процессорных элементов** 22](#_Toc135039765)

[3.3 Выводы по разделу 3 24](#_Toc135039766)

[4. Разработка алгоритмов программного эмулятора 25](#_Toc135039767)

[4.1 Рассмотрение алгоритмов 25](#_Toc135039768)

[4.2 Алгоритмы внутренней работы системы 25](#_Toc135039769)

[4.3 Алгоритмы внешней работы системы 29](#_Toc135039770)

[4.4 Выводы по разделу 4 30](#_Toc135039771)

[5. Программная реализация 31](#_Toc135039772)

[5.1 Назначение разрабатываемого программного обеспечения 31](#_Toc135039773)

[6. Тестирование 32](#_Toc135039774)

[6.1 Перечисление возможных ошибок 32](#_Toc135039775)

[6.2 Выводы по разделу 6 33](#_Toc135039776)

[Заключение 34](#_Toc135039777)

[Приложение А 36](#_Toc135039778)

[Приложение Б 37](#_Toc135039779)

[Приложение В 38](#_Toc135039780)

[Приложение В 39](#_Toc135039781)

# Введение

На сегодняшний день существует множество архитектур и моделей вычислений. Каждые обладают рядом плюсов и минусов, поэтому сложно выбрать универсальный подход для решения поставленной задачи. Существуют системы, которые решают все задачи, но из-за этого скорость вычислений уменьшается. Но так же есть системы, которые являются ускоспециализированными, решают только класс поставленных задач. Преимущество такого подхода заключается в том, что каждая такая задача будет решаться быстрее. Недостатком же таких систем является то, что такая система будет решать только те задачи, под которые она настроена, а настраивать так систему это значить сделать для нее отдельную конфигурацию, что требует дополнительного времени и средств. Так же система сможет выполнять только узкий тип задач.

Систолическая система настраивается для каждой задачи отдельно, и выполняет только конкретно поставленную задачу. Само соединение процессорных элементов настроено для выполнения конкретных алгоритмов, что дает большой выигрыш в скорости. Так же в такой системе используется локальная память, чтобы при считывании из оперативной памяти операнд можно было бы обрабатывать и передавать другому процессорному элементу, чтобы вновь не записывать элемент в оперативную память. Это дает прирост скорости, потому что операция записи и чтения из памяти является достаточно долгой. Поэтому такую систему удобно использовать для конкретных алгоритмов.

# 1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данном разделе рассматривается развитие различных архитектур и систем, а так же техническое задания для лабораторной установки.

## 1.1 Общие сведения о систолических системах

Электронные вычислительные машины изначально создавались как помощь и ускорение вычислений. Главной целью все было сокращение времени вычисления, путем увеличения производительности системы. Задачи становились более сложными, количество данных становилось все больше. За этим требовалось повышать производительность систем, а так же надежность, так как ошибки в ряде вычислений могли привести к тяжелым последствиям (в зависимости от поставленной задачи). Основные архитектуры вычислительных систем были предложены еще в 80-х годах 20-го века, вносились только некоторые правки и усовершенствования. Далее производительность увеличивалась за счет увеличения элементной базы.

На сегодняшний день есть 2 пути ускорения вычислений с точки зрения элементной базы: увеличение количества элементов и усовершенствование элементов. Есть несколько прототипов новой элементной базы, но они работают не надежно и не полностью исследованы. Поэтому увеличением количества элементов заинтересованы большинство компаний, выпускающие элементы вычислительной системы.

Так же для ускорения вычислительных систем создаются узкоспециализированные системы, которые решают конкретные задачи, но для этого приходится разрабатывать для каждого такого класса задач специальную систему, что требует дополнительных вложений.

Систолическая система относится к классу последовательно групповой организации, в которых конструкции, образующие какой-либо алгоритм, объединяются в группы. То что выполняется внутри конструкции - может выполняться параллельно, а то что между – выполняются последовательно (в зависимости от системы и алгоритма).

В этой же группе состоит векторно-конвейерная обработка. Введение векторных команд позволяет эффективнее использовать систему, так как с помощью них задается одна и та же операция над элементами одного или нескольких векторов. Вычислительные конструкции внутри групп исполняются конвейерным процессором. Плюсы такой системы заключаются в том, что для современного уровня технологии векторно-конвейерные системы являются более гибкими и эффективными с точки зрения стоимости, такая обработка достигает значительного уровня ускорения по сравнению со скалярными машинами. Самая сильная сторона того типа одновременно является и его самой слабой. С одной стороны, поскольку учитывается только форма представления программ и данных, имеется теоретическая возможность реализовать процесс векторизации с помощью специальных программных средств - так называемых автоматических векторизаторов. Но с другой стороны, что же делать, если эти программные векторизаторы не найдут формального преобразования, позволяющего превратить некоторую программно-информационную структуру в векторную. Структура векторной вычислительной системы представлен на рисунке 1.

Другой тип вычислительных машин, ориентированных на алгоритмы, содержащие главным образом векторные операции - это матричные системы. Общая идея построения таких систем чрезвычайно проста - имеется большое количество одинаковых обрабатывающих устройств, в каждый из которых поступают свои данные, но которое одновременно выполняет одну и ту же операцию. Множество процессорных элементов организованы таким образом, что они исполняют векторные команды, которые были заданы устройством управления. Каждый процессорный элемент работает отдельно и все они соединены через коммутационное устройство.

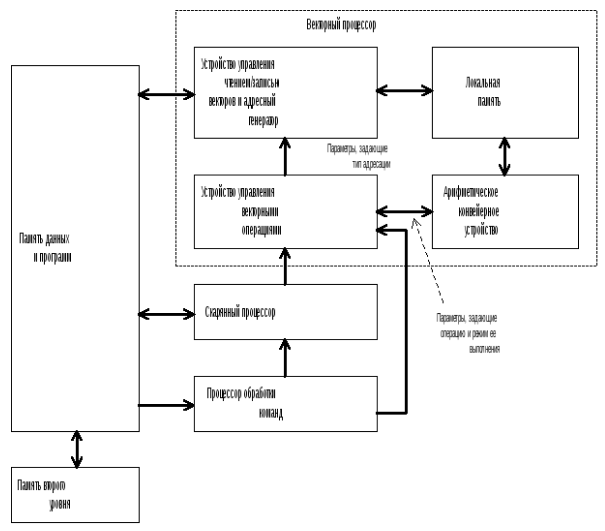


Рисунок 1 – Структура векторной вычислительной системы

Структура матричной системы показана на Рисунке 2. Доступ к данным происходит через коммутационную сеть. К недостатком можно отнести:

* в случае, когда число элементов обрабатываемых векторов не кратно числу процессоров, появляются незанятые процессоры и общая пропускная способность уменьшается;
* алгоритмы, как правило, содержат скалярные операции, выполнение которых не может перекрываться с выполнением векторных команд. Таким образом, весь массив процессоров какое-то время бездействует;
* коммутационная сеть имеет ограниченные возможности. Данные должны передаваться из памяти в соответствующие процессоры одновременно. Чтобы обеспечить возможность любому процессору работать с любым модулем памяти, необходимо использовать коммутатор перекрестных связей.

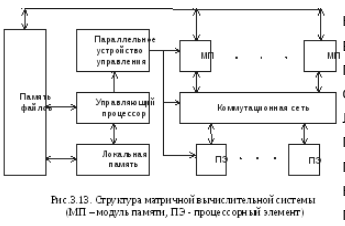


Рисунок 2– Структура матричной вычислительной системы

## 1.2 Обзор аналогов

В данном разделе можно сравнить проектируемый процессорный элемент как официально выпущенные процессорные элементы и с точки зрения учебной установки. Очевидно, что выпущенные модели будут гораздо производительнее, но их следует рассмотреть для изучения архитектур.

### 1.2.1 Структура процессора Intel Itanium 2

Первым примером послужит процессорный элемент Intel Itanium 2. Принципиальная схема представлена на Рисунке 6. Процессорный элемент работает на частоте 1.6 ГГц, размер кэша 6 MB. В нем есть 128 регистров, разрядность 32 бита. Производительность такого процессорного элемента будет больше, чем у проектируемового, но такая система имеет большую стоимость.

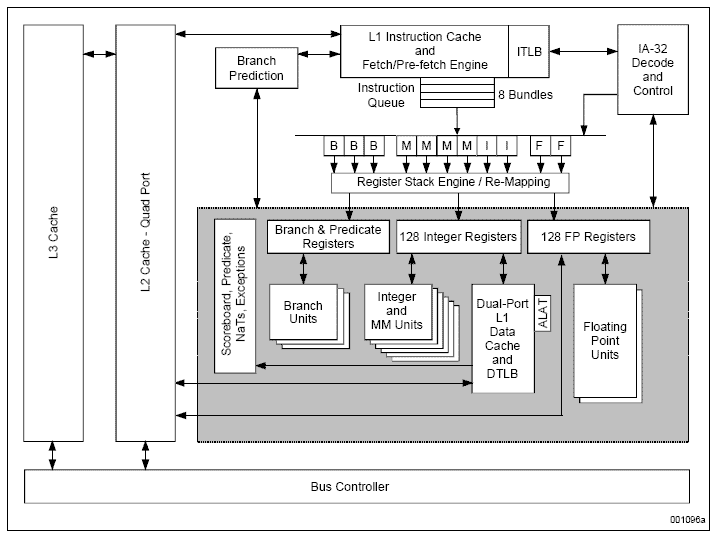


Рисунок 6 – Принципиальная схема Intel Itanium 2

### 1.2.2 Структура процессора AMD Athlon

Следующий пример будет архитектура процессорного элемента AMD. Ядро процессоров с архитектурой AMD использует другие принципы своего функционирования, по сравнению с intel. По сути Athlon является RISC процессором – в режиме реального времени выполняется преобразование потока CISC команд в унифицированные RISC команды.

1.2.3 Лабораторный стенд на основе ПКП К1810ВН59А

Далее следует рассмотреть лабораторные установки, сделанные для помощи студентам. Одной из таких установок является установка, которая обрабатывает приходящие вектора прерываний. Принцип работы ПКП К1810ВН59А зависит от типа используемого микропроцессора.

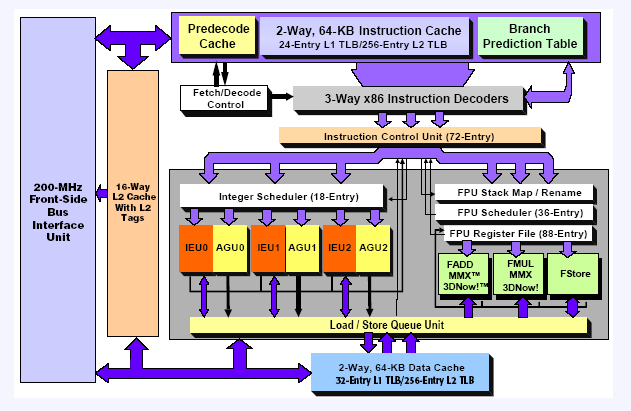


Рисунок 7 – Принципиальная схема AMD

Режим работы микросхемы устанавливается путём её программирования как устройства ввода/вывода с помощью программного обеспечения микропроцессорной системы. В процессе работы ПКП можно изменять алгоритмы обслуживания прерываний. Это осуществляется с помощью управляющих слов. Разница в проектируемом будет заключаться в том, что не нужно будет самим формировать управляющее слово, так как будут уже фиксированные команды. Структура установки показана на Рисунке 8.

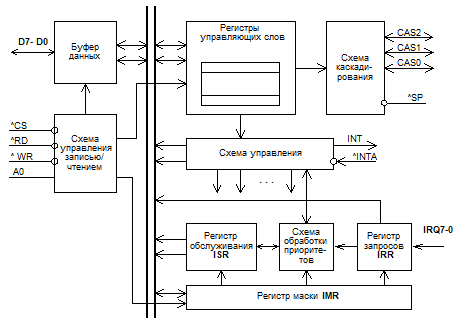


Рисунок 8 – Структура БИС

### 1.2.4 Универсальный лабораторный стенд

В следующей лабораторной установке программное обеспечение выполняет подачу управляющих сигналов в модули исследуемых устройств, принимает осведомительные сигналы, копирование состояния внутренних регистров (ячеек памяти) МИУ после каждого такта (цикла) работы; организация работы МИУ по тактам и в автоматическом режиме; обеспечение ввода пользователем управляющих воздействий (сигналов, микрокоманд) и их последовательностей (микропрограмм, программ); вывод на экран текущего и нового состояния МИУ после каждого такта (цикла) работы; запись и считывание с диска введенных микропрограмм (программ) и данных; в данной лабораторной установке набором команд можно выполнять операции, отличие в том, что выполнение происходит последовательно на одном процессоре.

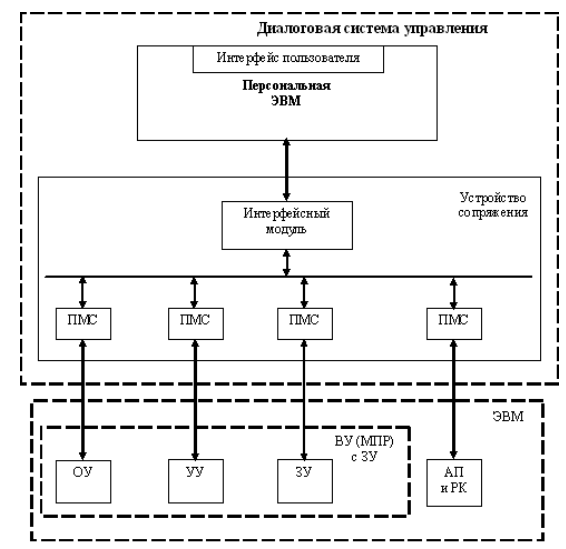


Рисунок 9 – Структура лабораторной установки

## 1.3 Расширенное техническое задание

### 1.3.1 Полное наименование системы и её условное обозначение

Фрагмент эмулятора процессорного элемента систолической системы

### 1.3.2 Наименование разработчика системы и реквизиты заказчика

ФГБОУ ВО «ВятГУ», кафедра электронных вычислительных машин.

Разработчик – студент группы ИВТб 4301-01-00 Пестов Максим Александрович

### 1.3.3 Основания для создания системы

Учебный план по дисциплине комплекс знаний бакалавра в области программного и аппаратного обеспечения вычислительной техники.

### 1.3.4 Плановые сроки начала и окончания работы по созданию системы

* Начало работ по созданию системы – осень 2022
* Окончание работ по созданию системы – конец весны 2022

### 1.3.5 Источник финансирования работ по созданию системы.

Собственные средства разработчика.

### 1.3.6 Порядок оформления и предъявления заказчику результатов работ по созданию системы:

К результату труда разработчика относится:

* Оригинальное программное обеспечение.
* Типовые проектные решения и особенности дизайна системы.
* Проектная и рабочая документация.

Результаты передаются заказчику частями по завершении каждой стадии работы по созданию системы

* Программное обеспечение.
* Документация – в электронном виде в формате PDF, на бумажных носителях.

Проектная документация должна быть разработана в соответствии с ГОСТ 34.201-89 и ГОСТ ЕСПД. Процедуры приемки – передачи результатов работ оформляются актами приемки-передачи.

### 1.3.7 Назначение и цели создания системы

*Назначение системы.*

Проектирование учебного эмулятора систолической системы для студентов.

*Цели создания системы.*

Целью создания системы является помощь в изучении систолической системы.

1.3.8 Характеристики объекта автоматизации

*Краткие сведения об объекте автоматизации.*

Требуется выставить процессорные элменны и соединить таким образом, чтобы возможно было решить поставленную задачу в соответсвии с выданным вариантом.

### 1.3.9 Требования к системе

1.3.9.1 Требования к системе в целом

*Требования к структуре и функционированию системы*

Эмулятор систолической системы должен представлять собой систему, включающую в себя подсистемы:

* п/с содержащая библеотеку процессорных элементов
* п/с соединения процессорных элеменотов
* п/с проверка задания, путем выполнения команд в программном эмуляторе

1. П/с содержащая библеотеку процессорных элементов:

* содержит все ссылки на все процессорные элементы;
* сожержит настройки отображения на панели и картинки, как должен выглядеть элемент;

1. П/с соединения процессорных элеменотов:

* Оторбражение каждой линии, которая соединяет процессорные элементы;
* Объек, которые описывает соединение каждого входа и выхода процессорных элементов;

1. П/с проверки задания, путем выполнения команд в программном эмуляторе:

* Система команд;
* Проверка правильности написания команды;
* Выдача ошибок при неправильных командах;

*Требования к средствам и способам связи для информационного обмена между компонентами системы.*

В качестве передаваемой информации между окном составления цепочки процессрных элементов и окна отладки массив созданных объектов и связь каждого процессорного элемента с другими.

*Требования к характеристикам взаимосвязи создаваемой системы со смежными системами, требования к её совместимости.*

Разработанное приложение должно быть совместима с операционными системи Windows 7 и выше, Linux и MacOs.

*Требования по диагностированию системы.*

При нахождении ошибок (логических или программных), разработчик должен модернизировать приложение.

*Перспективы развития, модернизация системы*

При модернизации программного обеспечения могут добавляться новые процессорные элементы, исправляться ошибки и добавляться новые операции.

*Требования к надёжности комплекса.*

Необходимо чтобы система обладала устойчивостью к отказам оборудования программных систем, а также электропитания. Требования надежности должны быть регламентированы для следующих аварийных ситуаций: Предлагать сохранение на каждом этапе работы.

1.3.9.2 Требование к видам обеспечения.

*Математическое обеспечение.*

Каждый процессорный элемент выполняет одну примитивную операцию. При соединении нескольких процессорных элементов возможно реализовать алгоритм умножения матриц.

Для параллельного сложения потребуются процессоные элементы, которые делают сложение. В данной операции можно складывать только по 4 бита. Основные числа разбиваются на части по 4 бита и складываются отдельно. Так же требуется прибавить единицу переноса если она есть.

*Требования к методическому обеспечению.*

Состав программной документации должен включать в себя:

* Техническое задание.
* Программное обеспечение.
* Руководство пользователя.
* Пояснительная записка.

## 1.4 Выводы по первому разделу

В данном разделе рассмотрены аналоги лабораторных установок и реальные архитектуры, рассмотрены их достоинства и недостатки. Так же в разделе выдвинуты основные требования для создаваемой системы.

# 2. Разработка структуры процессорного элемента систолической вычислительной системы

В данном разделе описывается функциональная схема и компоненты из которых она состоит. Так же показана методика работы с процессором.

**2.1. Определение взаимодействия узлов и блоков процессора**

Основная идея систолической обработки заключается в том, чтобы выполнить все стадии обработки каждого элемента данных, извлеченного из памяти, прежде чем вновь поместить в память результат этой обработки. Этот принцип реализуется систолической матрицей ПЭ, в которой отдельные ПЭ объединены между собой прямыми и регулярными связями, образующими конвейеры. По этим конвейерам как бы прокачиваются операнды, т.е. каждый элемент данных извлекается из памяти и ритмически продвигается по матрице ПЭ прежде, чем опять попадет в память. Таким образом, может формироваться несколько потоков данных, каждый из которых образован исходными операндами (элементами структуры данных, хранящейся в памяти), промежуточными результатами, получаемыми при выполнении элементарных операций в каждом ПЭ, и элементами результирующей структуры. Потоки данных синхронизированы единой для всех ПЭ системой тактовых сигналов. Во время тактового интервала все ПЭ выполняют короткую неизменную последовательность команд. Есть существенные плюсы:

1. Минимальное обращение в память, что повышает производительность;
2. Меньше операций ввода-вывода, значит меньше конфликтов при обращении в оперативную память;
3. Минимизируются связи между процессорными элементами за счет регулярности потоков данных и управляющих сигналов.

Выше рассматривалось как процессорные элементы работают между собой и по каким алгоритмам. А отдельно взятый процессорный элемент необходимо должным образом разработать так, что бы он выполнял элементарные действия, имел локальную память и правильно передавать данные в зависимости от строения системы.

При разработке структуры процессорного элемента требуется настроить процессорный элемент так, чтобы он выполнял одну операцию с поступившими на вход операндами. Входные операнды должны быть записаны в буферные регистры. По окончании такта и при выполнении операции результат должен быть помещен в выходной регистр. Входные данные в следующем такте должны быть отправлены к другим процессорным элементам.

В операционное устройство будут поступать операнды, проходя через входной буфер RgВх и будут передаваться в регистры RgO1 и RgO2, из которых начнется вычисление. Далее в блоке АЛУ будет совершаться операция. Результат операции записывается через шину данных в локальную память процессорного элемента и в выходной буфер RgВых. Структура представлена на Рисунке 17. Функциональная схема представлена на рисунке 18.

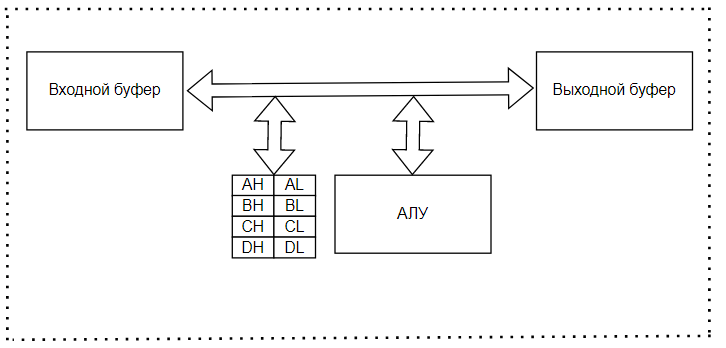


Рисунок 17 – Структура процессорного элемента

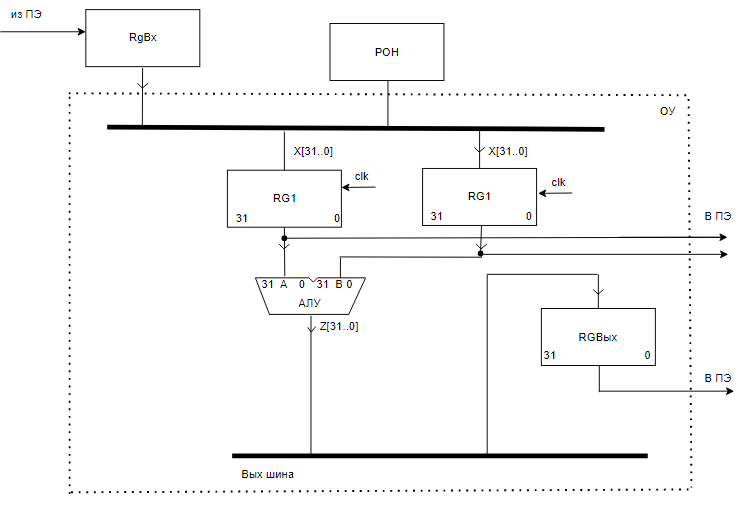


Рисунок 18 - Функциональная схема процессорного элемента

Структура элементов будет похожа, кроме операции, которые выполняет процессорный элемент. Для разных алгоритмов потребуются разные процессорные элементы. Для алгоритма параллельного сложения потребуется элемент, который складывает 4 бита и отправляет единицу переноса, которая складывается позже. Вид элемента для сложения по 4 бита предствлен на Рисунке 19, вид элемента для сложения размерностью операндов 32 бита представлен на Рисунке 20. Для алгоритма умножения матриц потребуется процессорный элемент, процессоный элемент, умножающий операнды и складывающий частичные суммы внутри процессорного элемента. Вид элемента для умножения операндов и сложения их частичных сумм представлен на Рисунке 21.

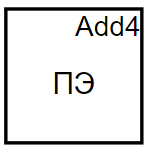


Рисунок 19 – Вид элемента сложение по 4 бита

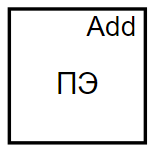


Рисунок 20 – Сложение операндов

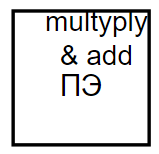


Рисунок 21 – Умножение операндов и сложения их частичных сумм

## 2.2 Выводы по разделу 2

В данном разделе рассмотрены блоки функциональной схемы, и как они функционируют между собой. Так же разработана структура и вид элементов.

# 3. Разработка структуры программного эмулятора

В данном разделе описывается структура программного эмулятора и передача данных между компонентамами системы.

**3.1 Разработка общей структуры программного эмулятора**

Программный эмулятор должен состоять из двух крупных компонентов.

Первый компонент содержит окно отладки собранной студентом систолической системы. Оно должно содержать таблицу ввода команд и возможность потактово делать отладку. Должна быть возможность открывать второе окно (второй большой компонент) для формирования системы процессорных элементов для выполнения задания.

Второй компонент должен содержать библиотеку элементов для составления системы процессорных элементов. Так же должен содержать поле для расстановки процессорных элементов. Так же должны быть реализована возможность редактировать положение элементов. Должна быть возможность соединения процессорных элементов, удаления связей, удаления ненужных процессорных элементов.

**3.2 Разработка окна отладки программы и построенной системы**

Данное окно является основным. Оно открывается при открытии приложения, значит на нем должны находится все функции для сохранения уже разработанной программы, открытия программы, выбор варианта задания, выданного преподавателем.

Основным в этом окне является таблица ввода команд для управления составленной систолической системы, в которой происходит отладка программы. Данная таблица должна выполняться потактово, чтобы студент выполнял команды по одной для того, чтобы отслеживать правильно ли идет вычисление. В таблице есть номер команды, сама команда и комментарии к этой команде, которые вводят сами пользователи для удобства в работе.

Должен быть раздел в котором показаны значения всех процессорных элементов в текущий момент времени. Там должны быть числовые значения, которые вычисляются на процессорном элементе в ходе его работы.

Должен быть отдельный блок, в котором должна быть преобразована для наглядности схема, составленная во втором окне. Система, составленная в окне построение системы будет преобразована в картинку и будет показано в какой последовательности будет обрабатываться поток данных.

Макет начального окна представлен на Рисунке 22.

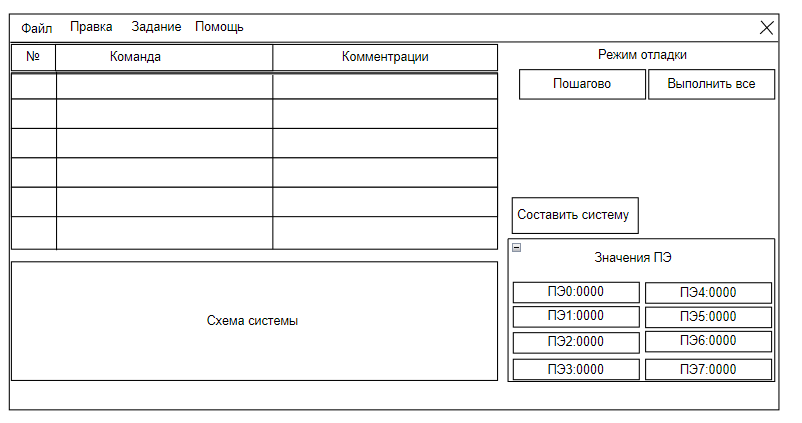


Рисунок 22 – Макет начального окна эмулятора

**3.2 Разработка окна составления системы процессорных элементов**

Окно для составления системы процессорных элементов состоит из нескольких блоков:

* Блок панели элементов;
* Блок расстановки элементов;
* Блок действий над элементами.

1. В блоке панели элементов должны быть добавлены кнопки, которые добавляют соотвестсвующие элементы на поле, где эти элементы должны быть расставлены. Элемент должен описывать размеры элемента, его функциональность, сколько элемент имеет входов и выходов, какая операция будет выполняться.

2. Блок расстановки элементов имеет вид сетки, в которую добавляются процессорные элементы из библиотеки элементов. Здесь их можно перемещать, менять их местоположение и выполнять действия, которые есть в блоке действий над элементами.

3. Блок действий над элементами содержит функции, которые можно выполнять над процессорными элементами. Такие как: соединить – при нажатии этой кнопки у процессорных элементов появляются значки входов и выходов, которые можно соединить. Удалить – при нажатии этой кнопки активируется режим удаления элементов. В нем при нажатии на элемент, он убирается с поля, а так же убираются все его связи с другими элементами.

Все эти элементы связаны между собой состояниями, данные передаются с помощью общей памяти (в программной реализации). При нажатии кнопки «Сохранить» данные сохраняются и передаются в первое окно, которые преобразуются в схему. Макет окна составления системы процессорных элементов предствален на Рисунке 23.



Рисунок 23 – Макет окна составления элементов

## 3.3 Выводы по разделу 3

В данном разделе описаны 2 больших компонента, которые обмениваются между собой информацией о расстановке процессорных элементов. Так же описаны как связаны маленькие компоненты в двух больших.

# 4. Разработка алгоритмов программного эмулятора

В данном разделе будут описаны алгоритмы, которые использовались для релизации программного эмулятора

## 4.1 Рассмотрение алгоритмов

Требутся рассмотреть алгоримты как для внутренней работы системы расстоновки процессорных элементов, так и для работы самой учебной установки.

В качестве внутренних алгоримтов будет описано заполнение связей между процессорными элементами, распознавания начальных элементов и элементов, на которых заканчиваются вычисления. В качестве алгоримтов работы будет рассматриваться алгоримты расстановки элементов по полю, визуальное соединение процессорных элементов.

## 4.2 Алгоритмы внутренней работы системы

Главными алгоримами в этом пункте являются выялвение первых элемнтов, с которых начинается вычисление и как соединяются процессорные элементы между собой.

Для каждого процессорного элемента существует объект, в который требуется записать соединение каждого из выходов и входов элемента. В нем так же содержится признак соединения с оперативной памятью.

Главным признаком, что вычисления начнутся с конкретного процессорного элемента является: оба операнда приходят из оперативной памяти.

Алгоритм выявления первого элемента вычисления:

1. Поиск элемента, у которого входы имеют признак соединения с оперативной памятью.

2. Относительно этого элемента начинать вычисления (на него подавать первые данные).

3. Дальше требуется проверять, имеет ли выход процессорного элемента в ОП. Для этого при передаче данных нужно делать проверку:

Если один или оба выхода процессорного элемента соединины с ОП, то п. 5, иначе п. 4

4. Передавать вектор данных дальше по системе процессорных элементов. Перейти к пункту 3.

5. При появлении признака выхода в ОП требуется прекратить передавать данные через этот выход.

Схема алгоритма приведена на Рисунке 24.

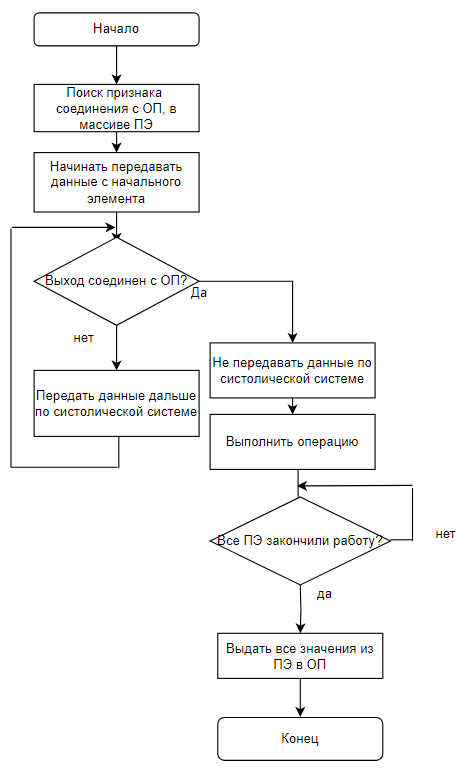


Рисунок 24 – Схема алгоритма определение начала и конца вычислений

Алгоритм соединения процессорных элементов между собой. У каждого ПЭ есть свойства, которые показывают как соединены процессорные элементы между собой.

Алгоритм соединения процессорных элементов между собой.

1. Если было нажато вход первым то, надо скрыть все остальные входы (чтобы случайно не соединить вход и вход), иначе скрыть все выходы.

2. Проверить: это был первый клик. Если это был первый клик, то переход к пункту 3, иначе к пункту 7

3. Если нажата ОП, то поставить признак соединения элемента с ОП с конкретным входом или выходом.

4. Поменять курсор на прицел.

5. Найти индекс нажатого ПЭ.

6. Найти первый вход был нажат или второй текущего ПЭ. Переход к пункту 13

7. Вычислить индекс второго ПЭ.

8. Вычислить в первый или второй выход произойдет соединение.

9. Записать в первый процессорный элемент: входу занести индекс процессроного элемента с кем он соединен, и записать номер выхода, с которым соединен вход.

10. Во второй ПЭ: записать к какому выходу соединен вход, и записать номер ПЭ, с которым он соединен.

11. Отрисовать линию соединения элементов.

12. Поменять курсор на обычный.

13. Конец.

Схема алгоритма приведена на рисунке 25.

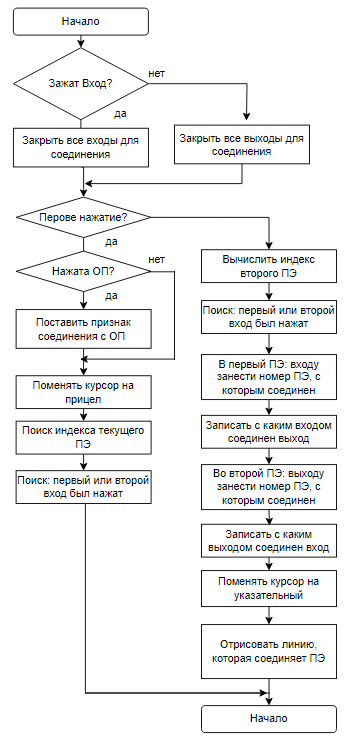


Рисунок 25 – Алгоритм соединения ПЭ

Алгоритм отрисовки линии между процессорными элементами сводится к тому, чтобы содинять элемента на заходя на границы самих процессорных элементов. Поэтому для правильной отрисовки требуется чтобы линия загибалась на 90 градусов, и делала это на половине расстояния между входом и выходом.

## 4.3 Алгоритмы внешней работы системы

В данном пункте описываются алгоритмы, которые внешне отображают действие с процессорными элементами на поле.

Алгоритм перемещение элемента с помощью нажатия.

1. Нажать на нужный элемент.

2. Вычислить координате курсора отностельно края поля.

3. Если нет второго нажатия, то с помощью таймера перерисовывать положение элемента относительно положения курсора и перейти к п.2, иначе п.4.

4. Зафиксировать положение элемента с учетом сетки, на которых расположены элементы.

5 Конец.

Схема алгоритма перемещения элементов представлена на Рисунке 26.

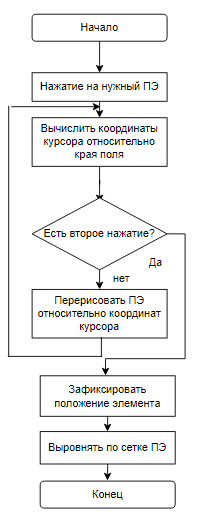


Рисунок 26 – Схема алгоритма перемещения ПЭ по полю

Алгоритм удаления элемента.

1. Нажать на кнопку «Удаление».

2. Нажатие на нужный элемент.

3. Если кнопка «Удаление до сих пор нажата», то переход к п.4, иначе к п. 7.

4. Сделать невидимым ПЭ.

5. Удалить входы и выходы.

6. Очистить все связи с другими ПЭ.

7. Конец.

Схема алгоритма удаления элементов представлена на Рисунке 27.

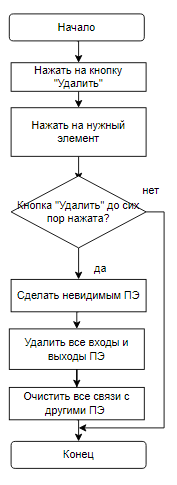


Рисунок 27 - Схема алгоритма удаления элементов

## 4.4 Выводы по разделу 4

В данном разделе описаны алгоритмы функционирования программного эмулятора. Описаны соединения процессорных элементов, а так же алгоритмы, связанные с визуальной частью программы.

# 5. Программная реализация

В данном разделе будут показаны функции которые были реализованы при написании программного эмулятора.

## 5.1 Назначение разрабатываемого программного обеспечения

Назначение ПО – сделать возможным самому составлять систолическую систему, и отлаживать ее из доступных компонентов для помощи студентам в изучении систолической системы.

Студент может использовать элементы из библиотеки процессорных элементов, соединять их в систему, которую можно проверять в окне отладки, и в случае ошибки менять систему, меняя соединения или удалять ненужные элементы.

Разработанное программное обеспечение должно обеспечивать выполнение следующих функций:

- выбор копонентов из библиотеки процессорных элементов;

- расстановка копманентов на поле элементов;

- перемещение элементов;

- соединение элементов с другими;

- удаление элементов;

- передача данных из одного окна в другое;

Обеспечена автоматическая проверка корректности данных, введённых пользователем. В случае, когда пользователь вводит неверные данные, например, точку вместо запятой в качестве десятичного разделителя, или литеру вместо цифры, программа сообщает ему об этом и предлагает повторить ввод.

# 6. Тестирование

В качестве объекта тестирование выбрано тестирования окна расстановки процессорных элементов и их соединение.

## 6.1 Перечисление возможных ошибок

Требуется описать состояния, которые могут нарушать надежность программы и описать их решение.

|  |  |  |
| --- | --- | --- |
| Входные условия | Неправильные классы эквивалентности | Правильные классы эквивалентности |
| Управление элементами | Элемент при нажатии действий перешел в неправильное состояние (1)  Удалены не все связи между ПЭ(2)  При нажатии перемещается не тот элемент или элементы друг на друге (3)  Соединение элемента неправильно отрисовано (4) | Отрисованы все линии, был нажат правильный элемент, при удалении удалены все связи (5) |

Меры для избежания ошибок:

1) Для перехода в нужное состояние были добавлены дополнительные переменные, которые проверяют нажатия на блоке «Действия».

2) Для удаления всех связей был создан объект содержащий все нарисованные линии, и если свойство существование линии равно нулю, то эта линия удаляется из списка линий на отрисовку.

3) Было принято решение активировать верхний элемент, чтобы не было неоднозначности в выборе элемента.

4) Соединение сделано так, чтобы линии были шли под углом 90 градусов, даже если линия будет отрисована под другим ПЭ, при перемещении ПЭ в другое более удобное положение для отрисовки – линия перерисуется лучшим образом.

При использовании данных мер, ошибки будут исключены.

## 6.2 Выводы по разделу 6

Были проведены меры по повышению надежности программного эмулятора и проведены тесты для проверки наличия ошибок и доказания надежности системы.

# Заключение

В процессе выполнения курсового проекта был разработан процессорный элемент систолической системы.

В ходе выполнения проекта выполнены основные задачи, которые необходимо было реализовать, а именно понять как и откуда будут поступать данные в процессорный элемент, где данные будут храниться и использоваться, а так же как эти данные будут передаваться другим процессорным элементам, либо записываться в операционную память. Определены команды и операции, которые сможет выполнять процессорный элемент.

В дальнейшем планируется объединить несколько процессорных элементов для составления систолической системы которая будет решать поставленную задачу. При этом организуя различные структуры системы, такие как линейная, прямоугольная, гексагональная. Расширением может так же служить добавление новых операций над массивами чисел или символов.

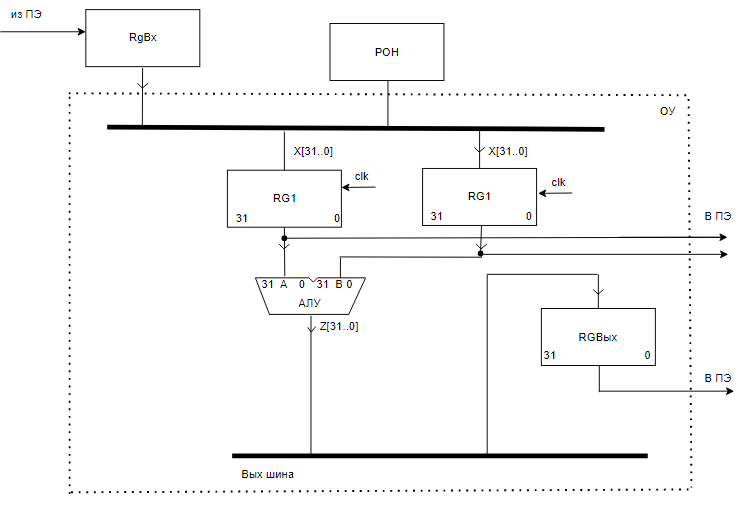
Список используемой литературы

1. Studfile.net[Электронный ресурс]. – Режим доступа: <https://studfile.net/preview/6266279/page:7/>
2. Fpga-e.ru[Электронный ресурс]. – Режим доступа: <https://fpga-e.ru/design/sistolicheskie-kih/>
3. scatspb.ru[Электронный ресурс]. – Режим доступа: <https://scatspb.ru/protsessor-intel-xeon-5430/>

Приложение А

(обязательное)

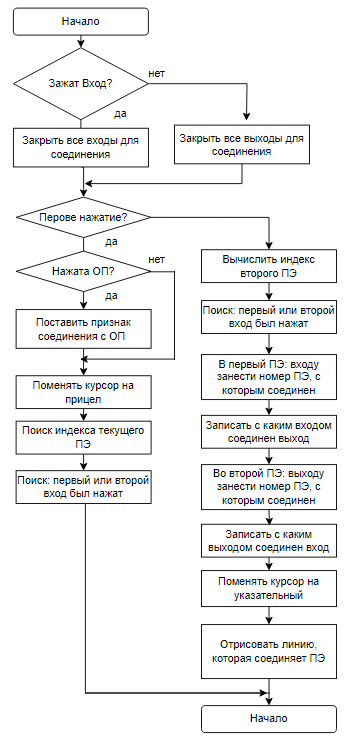
Функциональная схема



Приложение Б

(обязательное)

Схемы алгоритмов



Приложение В

(обязательное)

Экранные формы

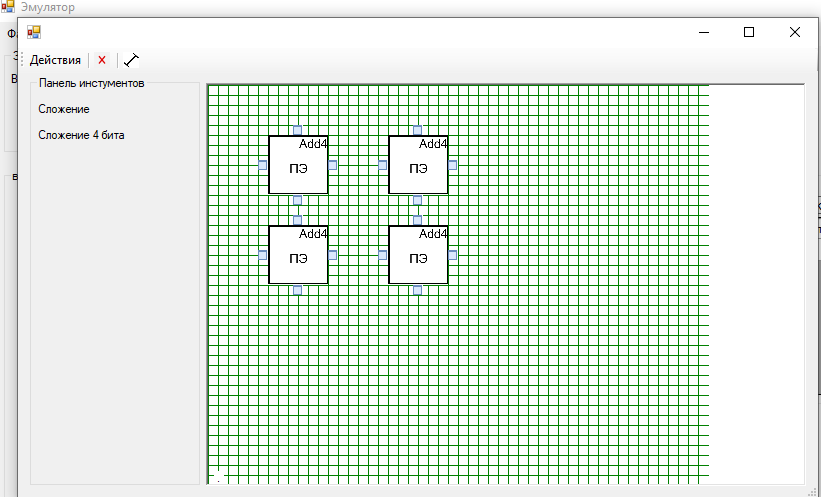


Рисунок 28 – Окно расстановки элементов

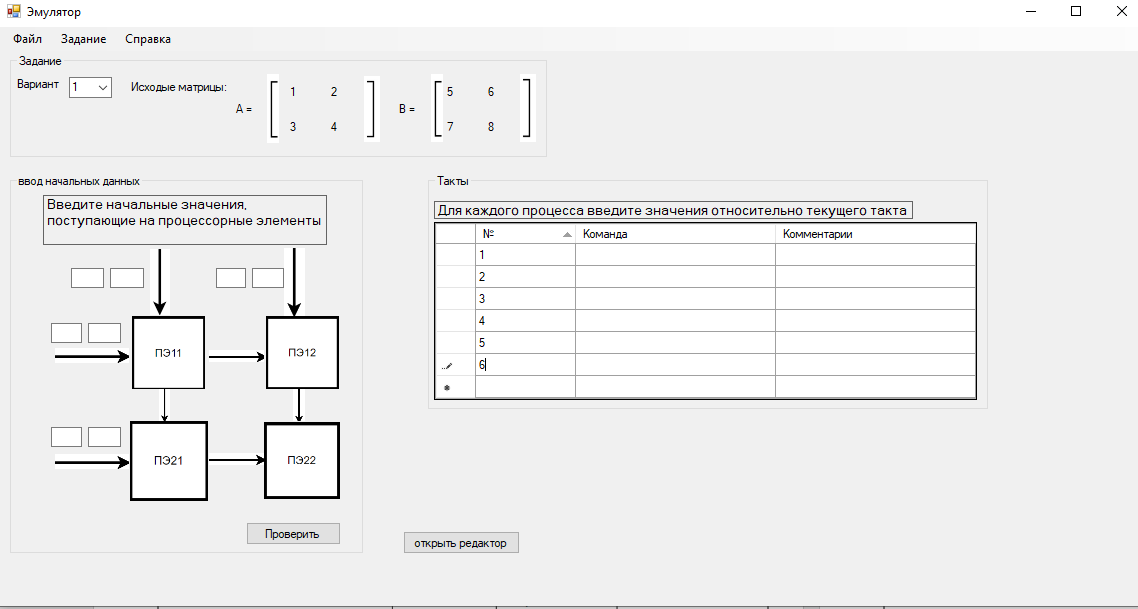


Рисунок 29 – Основное окно

Приложение В

(обязательное)

Код программы

private void add4bits\_Click(object sender, EventArgs e)

{

PictureBox imageControl = new PictureBox();

imageControl.Width = 60;

imageControl.Height = 60;

Bitmap image = new Bitmap("F:\\4 КУРС\\8 СЕМЕСТР\\Эмулятор\\SistolSystem\\SistolSystem\\add4bits.png");

//imageControl.Dock = DockStyle.Fill;

imageControl.SizeMode = PictureBoxSizeMode.Zoom;

imageControl.Image = (Image)image;

imageControl.Left = shiftLeft;

imageControl.Top = shiftTop;

imageControl.MouseDown += changeLocationDown;

imageControl.MouseUp += changeLocationUp;

imageControl.LocationChanged += changeLocation;

imageControl.Click += deletePE;

panel1.Controls.Add(imageControl);

shiftLeft = shiftLeft + 5;

shiftTop = shiftTop + 5;

elements[n] = imageControl;

typeElement[n] = 1;

PE[n] = new ProcessorElement();

PE[n].function = 1;

PE[n].exists = 1;

//создание квадратиков

//1

PE[n].pictureInput1 = new PictureBox();

PE[n].pictureInput1.Width = 10;

PE[n].pictureInput1.Height = 10;

Bitmap image1 = new Bitmap("F:\\4 КУРС\\8 СЕМЕСТР\\Эмулятор\\SistolSystem\\SistolSystem\\square.png");

PE[n].pictureInput1.SizeMode = PictureBoxSizeMode.Zoom;

PE[n].pictureInput1.Image = (Image)image1;

PE[n].pictureInput1.Left = elements[n].Left - 10;//координаты

PE[n].pictureInput1.Top = elements[n].Top+25;

PE[n].pictureInput1.Click += startPoint;//начать соединение

//PE[n].pictureInput1.MouseUp += endPointOutput;//закончить соединение

PE[n].pictureInput1.Visible = false;

panel1.Controls.Add(PE[n].pictureInput1);

//2

PE[n].pictureInput2 = new PictureBox();

PE[n].pictureInput2.Width = 10;

PE[n].pictureInput2.Height = 10;

image1 = new Bitmap("F:\\4 КУРС\\8 СЕМЕСТР\\Эмулятор\\SistolSystem\\SistolSystem\\square.png");

PE[n].pictureInput2.SizeMode = PictureBoxSizeMode.Zoom;

PE[n].pictureInput2.Image = (Image)image1;

PE[n].pictureInput2.Left = elements[n].Left + 25;//координаты

PE[n].pictureInput2.Top = elements[n].Top - 10;

PE[n].pictureInput2.Click += startPoint;//начать соединение

//PE[n].pictureInput2.MouseDown += changeLocationDown;//начать соединение

//PE[n].pictureInput2.MouseUp += changeLocationUp;//закончить соединение

PE[n].pictureInput2.Visible = false;

panel1.Controls.Add(PE[n].pictureInput2);

//3

PE[n].pictureOutput1 = new PictureBox();

PE[n].pictureOutput1.Width = 10;

PE[n].pictureOutput1.Height = 10;

image1 = new Bitmap("F:\\4 КУРС\\8 СЕМЕСТР\\Эмулятор\\SistolSystem\\SistolSystem\\square.png");

PE[n].pictureOutput1.SizeMode = PictureBoxSizeMode.Zoom;

PE[n].pictureOutput1.Image = (Image)image1;

PE[n].pictureOutput1.Left = elements[n].Left +60;//координаты

PE[n].pictureOutput1.Top = elements[n].Top + 25;

PE[n].pictureOutput1.Click += startPoint;//начать соединение

//PE[n].pictureOutput1.MouseDown += changeLocationDown;//начать соединение

//PE[n].pictureOutput1.MouseUp += changeLocationUp;//закончить соединение

PE[n].pictureOutput1.Visible = false;

panel1.Controls.Add(PE[n].pictureOutput1);

//4

PE[n].pictureOutput2 = new PictureBox();

PE[n].pictureOutput2.Width = 10;

PE[n].pictureOutput2.Height = 10;

image1 = new Bitmap("F:\\4 КУРС\\8 СЕМЕСТР\\Эмулятор\\SistolSystem\\SistolSystem\\square.png");

PE[n].pictureOutput2.SizeMode = PictureBoxSizeMode.Zoom;

PE[n].pictureOutput2.Image = (Image)image1;

PE[n].pictureOutput2.Left = elements[n].Left +25;//координаты

PE[n].pictureOutput2.Top = elements[n].Top + 60;

PE[n].pictureOutput2.Click += startPoint;//начать соединение

//PE[n].pictureOutput2.MouseDown += changeLocationDown;//начать соединение

//PE[n].pictureOutput2.MouseUp += changeLocationUp;//закончить соединение

PE[n].pictureOutput2.Visible = false;

panel1.Controls.Add(PE[n].pictureOutput2);

n++;

}

private void deletePE(object sender, EventArgs e) //удаления элемента click

{

if (ifDelete)

{

PictureBox imageControl = sender as PictureBox;

for (int i = 0; i < n; i++)

{

if (imageControl.Handle == elements[i].Handle)

{

PE[i].exists = 0;

elements[i].Visible = false;

PE[i].pictureInput1.Visible = false;

PE[i].pictureInput2.Visible = false;

PE[i].pictureOutput1.Visible = false;

PE[i].pictureOutput2.Visible = false;

if (PE[i].input1PE != -1)//1

{

int a = PE[i].input1PE;

if (PE[i].input1WhichOut == 1)

{

PE[a].output1WhichIn = -1;

PE[a].output1PE = -1;

}

else

{

PE[a].output2WhichIn = -1;

PE[a].output2PE = -1;

}

}

if (PE[i].input2PE != -1)//2

{

int a = PE[i].input2PE;

if (PE[i].input2WhichOut == 1)

{

PE[a].output1WhichIn = -1;

PE[a].output1PE = -1;

}

else

{

PE[a].output2WhichIn = -1;

PE[a].output2PE = -1;

}

}

if (PE[i].output1PE != -1)//3

{

int a = PE[i].output1PE;

if (PE[i].output1WhichIn == 1)

{

PE[a].input1WhichOut = -1;

PE[a].input1PE = -1;

}

else

{

PE[a].input2WhichOut = -1;

PE[a].input2PE = -1;

}

}

if (PE[i].output2PE != -1)//4

{

int a = PE[i].output2PE;

if (PE[i].output2WhichIn == 1)

{

PE[a].input1WhichOut = -1;

PE[a].input1PE = -1;

}

else

{

PE[a].input2WhichOut = -1;

PE[a].input2PE = -1;

}

}

}

}

}

}

private void startPoint(object sender, EventArgs e)//нажали на вход, click

//нужно на момент хажатия сделать невидимыми все входы

{

PictureBox imageControl = sender as PictureBox;

if (choise == 0)

{

for (int i = 0; i < n; i++)

{

if (imageControl.Handle == PE[i].pictureInput1.Handle || imageControl.Handle == PE[i].pictureInput2.Handle)

{

choise = 1; break;

}

}

if (choise == 0)

{

choise = 2;

}

}

if (choise == 1)

{

if (!ifSqareOneClick)//если до этого не было нажато

{

ifSqareOneClick = true;

panel1.Cursor = System.Windows.Forms.Cursors.Cross;

for (int i = 0; i < n; i++)

{

if (PE[i].exists == 1)

{

PE[i].pictureInput1.Visible = false;

PE[i].pictureInput2.Visible = false;

}

}

//записали в глобальную

for (int i = 0; i < n; i++)

{

if (imageControl.Handle == PE[i].pictureInput1.Handle)

{

input = i;

inputOneOrTwo = 1;

break;

}

if (imageControl.Handle == PE[i].pictureInput2.Handle)

{

input = i;

inputOneOrTwo = 2;

break;

}

}

}

else

{

ifSqareOneClick = false;

panel1.Cursor = System.Windows.Forms.Cursors.Arrow;

for (int i = 0; i < n; i++)

{

if (imageControl.Handle == PE[i].pictureOutput1.Handle)

{

output = i;

outputOneOrTwo = 1;

break;

}

if (imageControl.Handle == PE[i].pictureOutput2.Handle)

{

output = i;

outputOneOrTwo = 2;

break;

}

}

if (inputOneOrTwo == 1)

{

if (outputOneOrTwo == 1)

{

PE[input].input1PE = output;

PE[input].input1WhichOut = 1;

PE[output].output1PE = input;

PE[output].output1WhichIn = 1;

}

else

{

PE[input].input1PE = output;

PE[input].input1WhichOut = 2;

PE[output].output1PE = input;

PE[output].output1WhichIn = 1;

}

}

else

{

if (outputOneOrTwo == 1)

{

PE[input].input1PE = output;

PE[input].input1WhichOut = 1;

PE[output].output1PE = input;

PE[output].output1WhichIn = 2;

}

else

{

PE[input].input1PE = output;

PE[input].input1WhichOut = 2;

PE[output].output1PE = input;

PE[output].output1WhichIn = 2;

}

}

for (int i = 0; i < n; i++)

{

if (PE[i].exists == 1)

{

PE[i].pictureInput1.Visible = true;

PE[i].pictureInput2.Visible = true;

}

}

choise = 0;