

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Специальность 09.05.01 «Применение и эксплуатация
автоматизированных систем специального назначения»

Специализация «Эксплуатация вычислительных машин, комплексов,
систем и сетей специального назначения»

ДИПЛОМНЫЙ ПРОЕКТ

на тему

УСТРОЙСТВО УПРАВЛЕНИЯ РЕСУРСАМИ
МНОГОПРОЦЕССОРНОЙ СИСТЕМЫ

Студент

Костин 8.06.22
(подпись, дата)

Костин Иван Сергеевич
(ФИО полностью)

Руководитель

Бикташев 8.06.22
(подпись, дата)

Бикташев Р.А.
(фамилия, инициалы)

Нормоконтролёр

Бычков 8.06.22
(подпись, дата)

Бычков А.С.
(фамилия, инициалы)

Рецензент

(должность, место работы)

Печерский 14.06.22
(подпись, дата)

Печерский С.В.
(фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 02.06.2022 г. № 13)

Заведующий кафедрой

Митрохин
(подпись)

М.А. Митрохин
(фамилия, инициалы)

Работа защищена с отметкой Отлично (протокол заседания ГЭК от

17 июня 2022 № 3)

Секретарь ГЭК

Гурин

Е.И. Гурин

Пенза 2022

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»

Заведующий кафедрой

М.А. Митрохин

«26» апреля 2022 г.

ЗАДАНИЕ НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студент Костин Иван Сереевич гр. 17В02 факультета вычислительной техники.

2. Тема работы: «Устройство управления ресурсами многопроцессорной системы».

Тема утверждена приказом ПГУ №357/о от "26" апреля 2022г.

3. Руководитель работы к.т.н., доцент кафедры ВТ Бикташев Р.А.

4. Задание на работу (назначение разработки, исходные данные и т.п.).

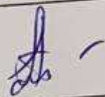
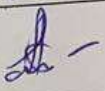
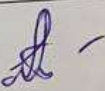
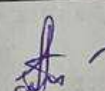
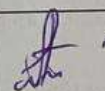
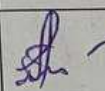
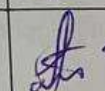
Устройство управления ресурсами предназначено для синхронизации взаимодействующих параллельных процессов в многопроцессорной системе. Функцией устройства управления ресурсами является обеспечение доступа к общему ресурсу на основе критических интервалов. Устройство управления ресурсами должно выполнять следующие условия:

- число процессоров (процессорных ядер) - до 8;
- число входов блока синхронизации - 8;
- тип запроса ресурса - приоритетный с циклической сменой приоритетов.

5. Перечень подлежащих разработке вопросов.

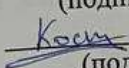
Требуется произвести анализ предметной области, выбрать инструмент разработки. Разработать схему работы системы, схему электрическую структурную, схему электрическую функциональную, схему программы функционально-логической модели устройства управления ресурсами. Произвести моделирование на языке VHDL. Разработать экономический раздел и раздел БЖД. Оформить техническую документацию на дипломный проект.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	26.04.22-30.04.22	
2.	Выбор инструментов разработки	5	01.05.22-03.05.22	
3.	Разработка схемы работы системы	10	04.05.22-07.05.22	
4.	Разработка схемы электрической структурной устройства управления ресурсами	15	10.05.22-15.05.22	
5.	Разработка схемы электрической функциональной устройства управления ресурсами	15	16.05.22-21.05.22	
6	Разработка схемы программы функционально-логической модели устройства управления ресурсами	10	22.05.22-28.05.22	
7.	Моделирование, отладка и тестирование	15	29.05.22-04.06.22	
8.	Разработка экономического раздела	5	05.06.22-07.06.22	
9.	Разработка раздела БЖД	5	08.06.22-10.06.22	
10.	Оформление технической документации ДП	10	11.06.22-14.06.22	

Дата выдачи задания "26" апреля 2022 г.

Руководитель дипломного проекта  26.04.22 Бикташев Р.А.
(подпись, дата)

Задание к исполнению принял студент  26.04.22 Костин И.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  25.06.2022 Фионова Л.Р.
(подпись, дата)

Реферат

Пояснительная записка состоит из 95 листов, 16 рисунков, 3 таблиц, 10 источников, 7 приложений.

УСТРОЙСТВО УПРАВЛЕНИЯ РЕСУРСАМИ, МНОГОПРОЦЕССОРНАЯ СИСТЕМА, КРИТИЧЕСКИЙ ИНТЕРВАЛ.

Цель работы – разработка аппаратного устройства управления ресурсами многопроцессорной системы. Устройство управления ресурсами представлено в виде модели на языке *VHDL*.

В рамках выполнения дипломного проекта был создан проект, удовлетворяющий исходным требованиям. А именно, разработана структура аппаратного устройства управления ресурсами, которая состоит из блока синхронизации на 8 входов.

Данную разработку рекомендуется использовать для организации доступа к общему ресурсу в системах реального времени. Устройство может быть внедрено в специализированные многопроцессорные системы, где требуется увеличить производительность путём снижения временных затрат на синхронизацию взаимодействующих параллельных процессов.

					ПГУ 1.090501.06.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Устройство управления ресурсами многопроцессорной системы Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Костин И.С.	<i>И.С. Костин</i>	8.06					
Провер.		Бикташев Р.А.	<i>Р.А. Бикташев</i>	8.06.02			2	95	
Н. Контр		Бычков А.С.	<i>А.С. Бычков</i>	9.06		Кафедра ВТ, Гр. 17В02			
Реконз.		Пегерид	<i>Пегерид</i>	14.06					
Утверд.		Митрохин М.А.	<i>М.А. Митрохин</i>						

Содержание

Введение	5
1 Устройство управления ресурсами многопроцессорной системы	7
1.1 Обзор предметной области	7
1.1.1 Общие сведения о процессах и их взаимодействиях	7
1.1.2 Анализ популярности	9
1.2 Постановка задачи	11
1.3 Выбор инструментов разработки	12
1.3.1 Программный подход к синхронизации процессов	12
1.3.1.1 Синхронизация в пространстве ядра ОС	15
1.3.2 Аппаратная реализация	17
1.3.3 Сравнение программной и аппаратной реализаций	18
1.3.4 Выбор <i>HDL</i> языка для разработки.	19
1.3.4.1 Сравнение <i>VHDL</i> , <i>Verilog</i> , <i>SystemVerilog</i>	20
1.3.4.2 Преимущества <i>VHDL</i>	22
1.4 Разработка структурной схемы устройства	23
1.4.1 Описание структурной схемы устройства.	23
1.5 Разработка функциональной схемы устройства.	27
1.5.1 Разработка графа переходов недетерминированного автомата и системы канонических уравнений, описывающих функционирование устройства	29
1.5.2 Синтез функциональной схемы устройства	32
1.6 Разработка программы на языке <i>VHDL</i> и результаты моделирования.	39
1.6.1 Составление программы моделирования	39
1.6.2 Анализ результатов моделирования	41
2 Техничко-экономическое обоснование разработки системы	47
2.1 Введение	47

Введение

На сегодняшний день архитектура современных операционных систем (ОС) включает типичные компоненты, которые наиболее часто используются выполняемыми задачами. С ними связаны различные механизмы взаимодействия между процессорами и управлением памяти, планированием или диспетчеризацией процессов, обработка прерываний. Функции некоторых из них, как правило, реализованы программным путем. Высокие затраты времени на использование компонентов многопроцессорных ОС приводят к большому количеству непроизводительных временных затрат, которые могут составлять несколько сотен процентов от общего количества времени на реализацию процессов пользовательских программ.

Давно наметилась тенденция к переходу от программной реализации алгоритмов ядра ОС к аппаратной. Эта тенденция связана со стремлением повысить отношение производительности/стоимости ЭВМ и обеспечить высокую степень интеграции логических элементов. И действительно, реализация алгоритмов ОС аппаратным путем может снизить время их выполнения и повысить производительность ЭВМ. На самом деле стоимость дополнительного оборудования, которое используется для создания аппаратных решений, благодаря прогрессу в развитии элементной базы, с каждым годом снижается. В то же время аппаратная реализация алгоритмов ОС позволяет значительно снизить сложность в создании программного обеспечения и улучшить совместимость различных версий операционной системы на разных платформах. Несмотря на это, значительно сокращается время и затраты на проверку нижних уровней операционной системы. Это достигается за счет лучшей формализации функций аппаратного способа по сравнению с программным способом, а также благодаря использованию современных инструментов для разработки.

Издержки операционной системы, основанной на программной реализации, особенно отражаются на системах реального времени (СРВ). При этом системы реального времени требуют высоких реакций на запросы, это и заставляет повышать производительность ОС. Это происходит, потому что в СРВ есть корототекущие процессы, которые часто поступаю. Следовательно, обращение к общему ресурсу происходит также часто [1].

Управление ресурсами, реализованное традиционной ОС приводит к значительным замедлениям выполнения приложений и, следовательно, значительному снижению производительности вычислительной системы.

Главные недостатки программного обеспечения, на котором базировались традиционные операционные системы, в т.ч. системы реального времени, могут быть устранены путем реализации функций ядра аппаратными средствами.

Временные потери снимаются путем использования блока аппаратной поддержки функции синхронизации процессов, производительность которого много выше, чем традиционная программная реализация такой же функции. При этом аппаратные затраты сравнимы с аппаратными затратами на контроллер прерываний с одинаковым числом входов запроса.

Предлагаемый подход заключается в том, что устройство синхронизации выполняется в виде независимого аппаратного устройства (специализированного процессора) в составе многопроцессорной системы.

1 Устройство управления ресурсами многопроцессорной системы

1.1 Обзор предметной области

1.1.1 Общие сведения о процессах и их взаимодействиях

При рассмотрении вопросов формального описания алгоритмов управления взаимодействующими процессами буду использовать некоторые основные понятия и определения, связанные с механизмом управления процессами и средствами их взаимодействия, которые приняты в теории и практике, как системного программного обеспечения при решении вопросов координации взаимодействующих процессов, так и прикладного программного обеспечения при решении задач организации параллельных вычислений.

К числу таких основных понятий и определений относятся понятия:

- процесс и основные типы их взаимодействия;
- разделяемые и критические ресурсы;
- критический интервал.

1.1.1.1 Основопологающим при рассмотрении операционных систем является понятие вычислительного процесса, или просто процесса. Как правило, процесс (иногда называют задача(*task*)), состоит из команд программы, данных и информации о выполняемой задаче. Информация может быть представлена как выделенная область памяти, открытые файлы и статус процесса.

Процессы, которые могут существовать одновременно, называются параллельными. Если работы этих процессов будут зависимы друг от друга, то говорят, что они взаимодействуют друг с другом, то есть должны периодически синхронизироваться и выполнять функции обмена данными.

Также основополагающим при рассмотрении операционных систем является понятие ресурса. Также существуют различные виды ресурсов: ресурсы устройств и ресурсы памяти.

Ресурсы устройств – это время, в течение которого устройство занято некоторой выполняющейся программой. Прежде всего, одним из важнейших ресурсов устройств является сам процессор, точнее — процессорное время.

Ресурс памяти – это емкость, требуемая для выполняющейся программы. В оперативной памяти одновременно может располагаться несколько задач одновременно.

Процессы при выполнении зачастую взаимодействуют между собой. В основном используются два типа взаимодействия:

- обмен данными между процессами;
- синхронизация процессов (согласование выполнения процессов при доступе к общему ресурсу).

1.1.1.2 Для повышения эффективности вычислительной системы (ВС) необходимо, чтобы ресурсы ВС использовались несколькими процессами, т.е. они должны быть разделяемыми. Исходя из этого, разделяемые ресурсы – это такие ресурсы, которые могут использоваться несколькими процессами.

Под критическим ресурсом подразумевается ресурс, который допускает обслуживание исключительно одного пользователя за один раз.

Если множество взаимодействующих процессов требуют одновременно доступа к общему (критическому) ресурсу, необходимо синхронизировать их работу так, чтобы только один процесс владел ресурсом, другие должны находиться в режиме ожидания. Следовательно, основными функциями подсистемы управления взаимодействующими параллельными процессами являются функция «взаимоисключение» и функция «ожидание».

1.1.1.3 Критический интервал (участок, секция) – это часть выполняющейся программы, которая непосредственно работает с

разделяемым (общим, критическим) ресурсом. Критический интервал обладает таким свойством, что в этой части выполняющейся программы переменные могут изменяться непредсказуемо другими процессами, хотя выполнение ее не завершено ни одним процессом. Чтобы исключить такого рода конфликт, необходимо чтобы критические интервалы были взаимоисключающими. То есть в любой момент времени только один процесс может находиться в своем критическом интервале. При формализации алгоритмов управления процессами и ресурсами на языке недетерминированных автоматов с использованием систем канонических уравнений, критические интервалы моделируются несовместимыми событиями [2].

1.1.2 Анализ популярности

В настоящее время большую популярность получили многопроцессорные вычислительные системы, в которых параллельно выполняется множество выполняющихся программ (процессов, потоков, нитей). Многопроцессорные, как и другие вычислительные системы, имеют разделяемые ресурсы. Причем выполнение процессов происходит в режиме «истинного» параллелизма, а не в режиме переключения процессов, что вносит в синхронизацию процессов некоторые нюансы, в частности нельзя использовать в качестве синхронизации механизм прерываний, т.к. прерывание одного процесса не приводит к прерыванию других процессов, выполняющихся в разных процессорах. Поэтому главным методом синхронизации процессов при доступе к разделяемому ресурсу в МПВС является метод блокирующей переменной [3]. Существуют варианты реализации этого метода, поэтому при проектировании новых вычислительных систем необходимо определить эффективность вариантов реализации. Одним из основных критериев эффективности является производительность.

Традиционно управление процессами и ресурсами осуществляется на основе системного программного обеспечения. Однако такой подход зачастую приводит к значительным потерям производительности (к замедлениям выполнения пользовательских приложений), т.к. значительное процессорное время тратится не на обработку приложений, а на обработку системных программ, что ложится накладными расходами на функционирование вычислительной системы.

Существующие функции ядра ОС: планировщики задач, связанные с выделением и освобождением ресурсов, диспетчеры задач, связанные с переключением процессов, средства синхронизации процессов, являясь трудоемкими, и требуют значительных процессорных ресурсов. Возможна реализация средств синхронизации на пользовательском уровне. При этом производительность выполнения функции синхронизации увеличивается, но возникают проблемы надежности и безопасности из-за ограниченных возможностей современных средств тестирования программного обеспечения.

Исследователями данной проблемы замечено, что системы реального времени, где особенно ярко высвечиваются недостатки реализации ядра прогаммными средствами, могут быть устранены путем реализации этих функций аппаратными средствами. Существует два подхода к аппаратно-ориентированной реализации алгоритмов управления процессами и ресурсами в многопроцессорных (многоядерных) системах: микропрограммная и схемная [4].

Микропрограммным способом включают механизмы управления процессами и ресурсами в систему команд разрабатываемого процессора. Они позволяют выполнять синхронизацию процессов методом активного ожидания (спин-блокировки) в пространстве пользователя. Этот способ применяют в некоторых языках программирования под названием мониторы.

Схемная реализация заключается в вынесении трудоемких функций ядра операционной системы за пределы самого ядра в виде ускорителя, реализованного как устройство управления или как управляющий спецпроцессор. При этом процессорная командка запроса на критический ресурс преобразуется в сигнал на выходе процессора для подачи его в УУР. Схемная реализация позволяет освободить ядро ОС от функции синхронизации процессов, в результате она значительно ускоряется (уменьшается латентность запроса), а отладка аппаратуры современными средствами проектирования ПЛИС и проверка её работоспособности на временных диаграммах обеспечивает надежность и безопасность операционной системы, что очень важно для систем реального времени.

Повышение производительности многопроцессорной системы при аппаратной реализации УУР происходит за счет:

- быстрого выполнения функций ОС по управлению ресурсами с аппаратной поддержкой;
- освобождения процессоров от программного выполнения функций операционных систем по управлению процессами и ресурсами и перенести высвободившийся ресурс процессора на выполнение приложений.

1.2 Постановка задачи

Необходимо разработать устройство управления ресурсами многопроцессорной системы.

Устройство управления ресурсами многопроцессорной системы должно выполнять следующие условия:

- число процессоров (процессорных ядер) - до 8;
- число входов блока синхронизации - 8;
- чип запроса ресурса - приоритетный с циклической сменой приоритетов.

Блок синхронизации предназначен для управления параллельными взаимодействующими процессами. Он обеспечивает доступ к общему ресурсу на основе критических интервалов.

Одним из путей повышения производительности многопроцессорной операционной системы является аппаратная реализация функций управления ресурсами (в частности, устройства синхронизации процессов), которая не только резко увеличивает реакцию на запрос ОР, но и позволяет повысить надежность операционной системы, что особенно важно для систем реального времени.

Временные потери снимаются путем использования блока аппаратной поддержки функции синхронизации процессов, производительность которого много выше, чем традиционная программная реализация такой же функции. При этом аппаратные затраты сравнимы с аппаратными затратами на контроллер прерываний с одинаковым числом входов запроса.

Предлагаемый подход заключается в том, что устройство синхронизации выполняется в виде независимого аппаратного устройства (специализированного процессора) в составе многопроцессорной системы (рисунок 1).

1.3 Выбор инструментов разработки

1.3.1 Программный подход к синхронизации процессов

Программный подход к синхронизации процессов является традиционным для метода, основанного на критических интервалах. Метод реализуется в пользовательском пространстве. Критическому интервалу ставится в соответствие общая двоичная переменная, которую называют блокирующей. Ей присваивается значение единицы, когда осуществляется вход в критический интервал, и значение нуля, когда производится выход из критического интервала. Процесс или поток, прежде чем войти в критический интервал, проверяет блокирующую переменную, и если она

установлена, то переходит в режим циклического опроса переменной. При освобождении критического интервала процесс устанавливает блокирующую переменную и входит в критический интервал. Выполнив необходимые действия с критическим ресурсом, процесс сбрасывает блокирующую переменную, тем самым предоставляя доступ в неё другим процессам.

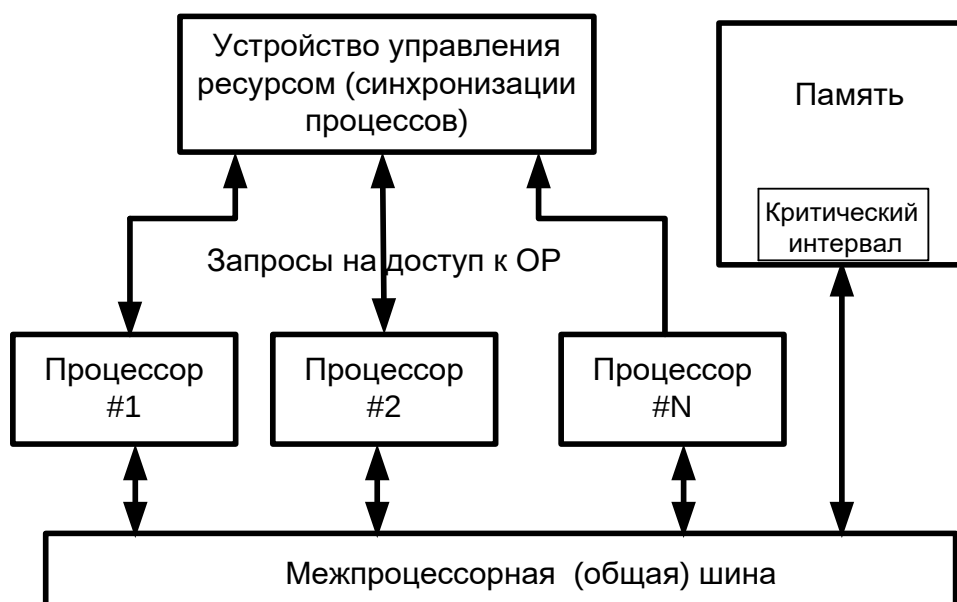


Рисунок 1 – Схема подключения устройства управления ресурсами в многопроцессорную систему

Запрос к некоторому ресурсу D с помощью блокирующей переменной $F(D)$ представлен на рисунке 2.

Такой способ синхронизации процессов быстроедействующий, т.к. происходит в пространстве пользователя. Однако у него имеются недостатки:

- запущенный процесс неопределенно долго может находиться в режиме циклического опроса (петле ожидания). Поскольку проверка производится при участии процессора, поэтому бесполезно тратится процессорное время, и, следовательно, высокие потери производительности ВС;
- поскольку программный код критического интервала реализуется

непосредственно программистом, то возможно внесение ошибок, проявляющихся крайне редко, которые приводят к нежелательным последствиям, а иногда и катастрофам. Такое возможно, если в критический интервал войдут одновременно два или более процессов. Например, пусть некоторый процесс, требующий доступа к некоторому ресурсам, не успев его установить в состояние «занят» был прерван, например, по истечении кванта. Другой процесс определяет, что ресурс свободен, занял ресурс, вошел в критический интервал и был прерван. Когда первый процесс получит управление, он считает, что ресурс свободен, занимает его, и так же входит в критический интервал. Таким образом, два процесса оказываются в критическом интервале, что не допустимо.

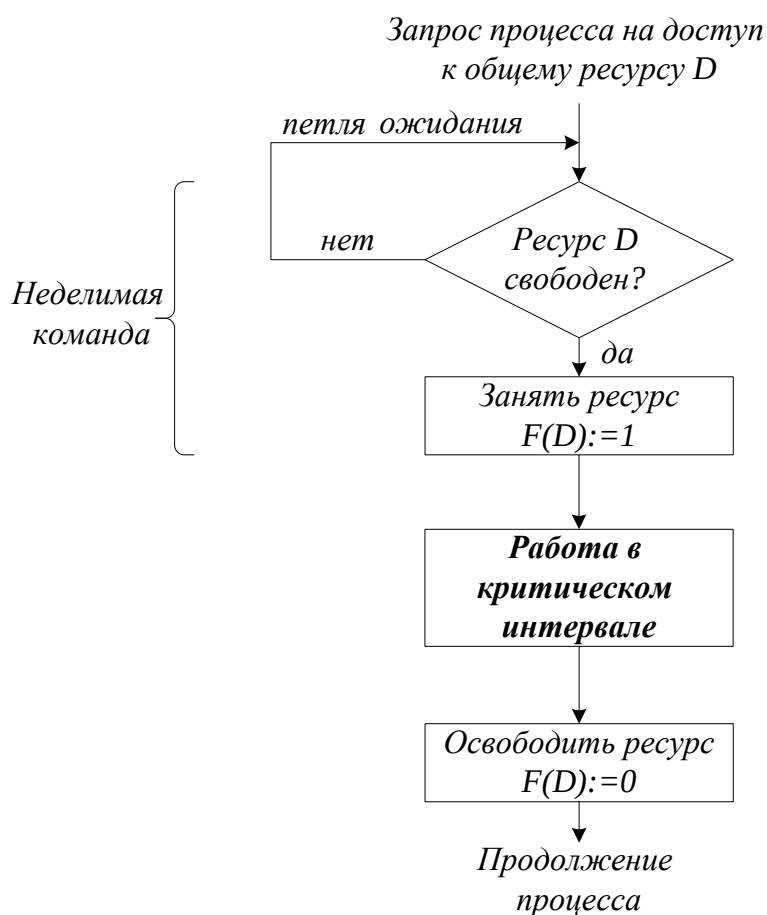


Рисунок 2 – Алгоритм программного подхода к синхронизации процессов

Существуют приемы, позволяющие исключить такую конфликтную ситуацию:

- при запросе ресурса запрещать прерывания. Однако, такой метод не годится для многопроцессорных систем, т.к. прерывание запрещается в том процессоре, где он выполняется, остальные процессоры активны и способны формировать запросы к общим ресурсам;

- использовать неделимую (единую) команду типа *TSL – test set and lock* («проверить и установить»), которая вначале проверяет ресурс на занятость, и только потом устанавливает блокирующую переменную в состояние «1» (занять) или «0» (освободить).

TSL — простая неразрывная (атомарная) процессорная команда, которая копирует значение блокирующей переменной в регистр, и устанавливает её в состояние блокировки. Процессор не может прервать её выполнение и переключиться на выполнение другого процесса (потока). Если используется многопроцессорная архитектура, то, пока один процессор выполняет эту инструкцию с ячейкой памяти, где хранится переменная, другие процессоры не могут получить доступ к этой ячейке, что достигается обычно путём кратковременного блокирования шины памяти.

1.3.1.1 Синхронизация в пространстве ядра ОС

Этот метод использует системные вызовы. Заключается в том, что процесс (поток) прежде чем войти в критическую секцию выполняет системный вызов, в рамках которого идет проверка блокирующей переменной, и если ресурс занят, то запрашивающий процесс переходит в режим ожидания (засыпает). При этом делается отметка о том, что данный процесс или поток должен быть в дальнейшем активирован (разбужен). Перечисленные функции выполняет ядро ОС.

Алгоритм входа в критический интервал и выхода из критического интервала представлен на рисунке 3.

Такой метод исключает бесконечный цикл опроса блокирующей

переменной. Для этого процесс просто усыпляется, например, путем его переноса из очереди готовых в очередь ожидающих процессов. Но это действие требует входа в ядро ОС, который сопровождается сохранением контекста на входе и восстановлением контекста на выходе, на что тратится значительное процессорное время, и следовательно, снижается производительность многопроцессорной системы.

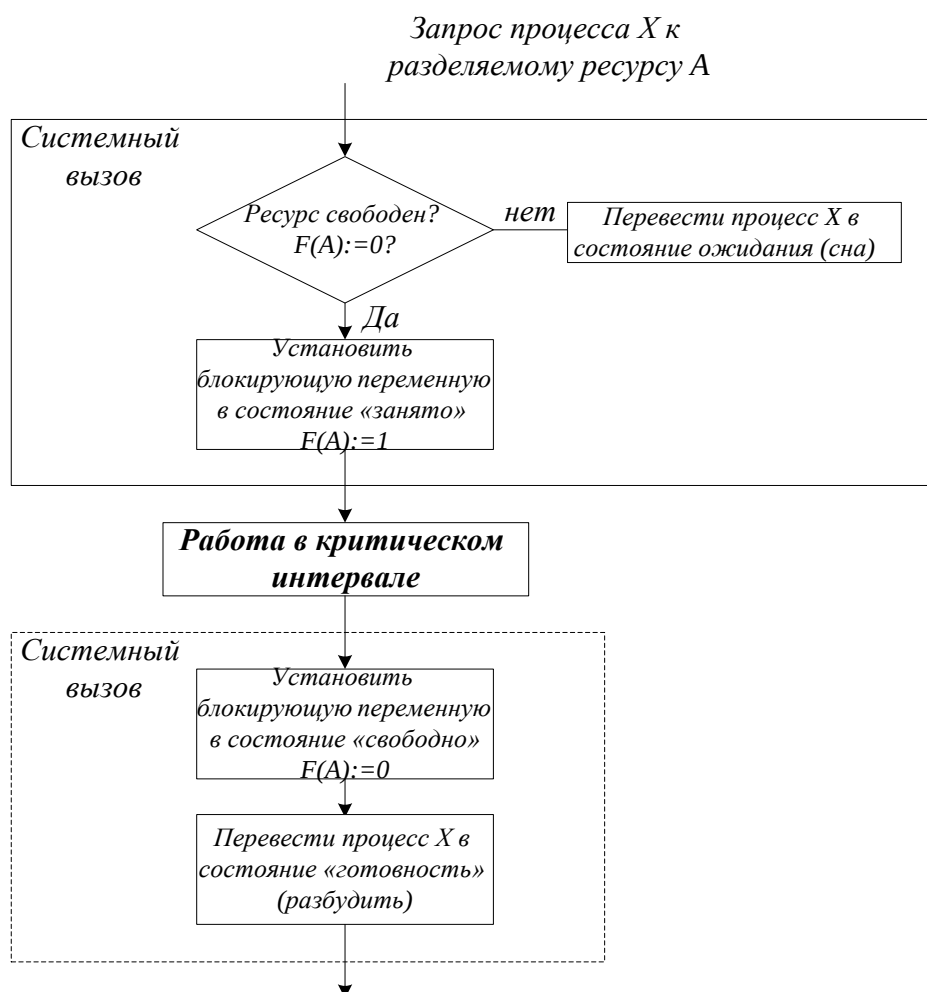


Рисунок 3 – Алгоритм программной синхронизации в пространстве ядра ОС

Традиционно блокирующие переменные, использующие системные вызовы, реализуют в виде битовых семафоров, мьютексов (*mutex* - сокращение от *mutual exclusion* – взаимное исключение), и других подобных примитивов синхронизации.

1.3.2 Аппаратная реализация

Аппаратная реализация заключается в том, что синхронизация процессов выполняется специальным устройством (рисунок 1), находящимся в составе процессорного узла (или в составе многопроцессорной системы), которое называется устройством управления ресурсами (УУР) или устройством синхронизации процессов (УСП). В этом случае сформированный процессом запрос к ресурсу направляется не в ядро ОС, где традиционно осуществляется синхронизация процессов, а перенаправляется к УУР, где и обрабатывается по алгоритму, реализующему механизм критических интервалов. Для реализации данного подхода процессорная команда, осуществляющая запрос (или соответствующий системный вызов), преобразуется в аппаратный сигнал на соответствующем выходе процессора, который соединяется с входом устройства управления ресурсом, где обрабатывается схемно. Устройство УУР должно иметь несколько входов, чтобы принимать сигналы запросов на доступ к ОР от множества процессов одновременно.

В приложении Б представлена схема работы системы управления ресурсами, которая описывает работу УУР при аппаратной реализации.

Поступающий запрос фиксируется в блоке приема и фиксации запросов, затем выбирается запрос с наивысшим приоритетом. Далее происходит проверка свободен ли критический интервал, если нет, то запрос переходит в режим «ожидания», иначе ему присваивается потолочный приоритет и в блоке захвата ресурса $S_k = 1$, тем самым устанавливается блокирующая переменная, что и означает, что критический интервал занят. Затем происходит работа процесса в критической секции. По окончании этой работы восстанавливаются приоритеты, сбрасывается блокирующая переменная и процессы находящиеся в режиме «ожидания» переводятся в режим «готовность».

Процессы (потoki), требующие синхронизации выполняют

соответствующий системный вызов (например, *sleep()* в ОС *UNIX*). Если ресурс занят, то вызывающий процесс усыпляется до тех пор, пока устройство УУР не выдаст сигнал на его пробуждение. Этим исключается бесконечный цикл ожидания, а также отпадает необходимость обращения к ядру ОС, где и происходят основные потери производительности многопроцессорной системы.

Таким образом, метод реализуется как бы в пользовательском пространстве, но аппаратным устройством, что обеспечивает высокую скорость его обработки и повышение производительности функций ядра операционной системы.

А поскольку функции ядра реализуются в сопроцессоре, то центральные процессоры освобождаются от выполнения системного программного кода, обеспечивающего синхронизацию процессов. Взамен этого они могут выполнять пользовательский код, что дополнительно экономит процессорное время.

1.3.3 Сравнение программной и аппаратной реализаций

Рассмотрим временные составляющие, которые образуют задержку при доступе процесса к ресурсу путем системного вызова. При входе процесса в критический интервал (КИ) затрачивается время на ожидание в очереди процессов, которые одновременно запрашивают общий ресурс ($t_{оч}$), плюс время входа в ядро ОС ($t_{вх}$) и время выхода из ядра ($t_{вых}$), которые составляют временные затраты на выполнение системного вызова ($t_{св} = t_{вх} + t_{вых}$). Кроме этого затрачивается значительное время на выделение наиболее приоритетного запроса $t_{пр}$ среди запрашивающих ресурс. Во время входа в КИ процесс дважды входит в ядро и дважды выходит из ядра.

Исходя из этого, общее время задержки составит:

$$t_{ож} = t_{оч} + t_{пр} + 2t_{св}.$$

Известно, что программно реализованный системный вызов требует больших затрат процессорного времени, так как он связан с переключением

процесса из режима пользователя в режим ядра. Такое переключение потребляет от нескольких сотен до нескольких тысяч процессорных тактов.

При аппаратной реализации время входа и выхода в КС составляет несколько единиц процессорных тактов. Таким образом, латентность устройства синхронизации процессов в сотни раз меньше, чем при программной реализации, что позволяет повысить производительность вычислительной системы от 5% до 20% в зависимости от частоты поступления запросов к общим ресурсам.

А так же аппаратная реализация освобождает процессор, как ресурс от работы в ядре ОС, так как этим занимается УУР (блок синхронизации), это тоже экономит значительное количество процессорных тактов.

Кроме того, аппаратное устройство стабилизирует время выполнения критического интервала, т.к. не допускает прерывания процесса, выполняющегося в аппаратуре.

Аппаратное устройство подвергается тщательной отладке на уровне временных диаграмм, что повышает надежность операционной системы, что очень важно для систем реального времени. Также аппаратное устройство менее подвержено вирусным атакам, что также повышает надежность ОС.

1.3.4 Выбор *HDL* языка для разработки

Язык *HDL* (*hardware description language, HDL*) используется разработчиками электронной аппаратуры, он позволяет описывать поведение и структуру системы, разрабатывать схемы. Наиболее известны сегодня *HDL*-языки *VHDL*, *Verilog* и *SystemVerilog*. Эти языки в отличие от обычных языков программирования, включают в себя описание времени задержки распространения сигнала и подразумевают одновременное выполнение некоторых логических процессов. Без этих специализированных языков описания аппаратуры невозможно разработать совершенные *SoC* (*System on Chip*).

Каждый из представленных языков *HDL* обладает своим собственным

стилем описания. *VHDL* и *Verilog* реализуют абстракции регистр - передача сигнала - логический уровень (*register-transfer-level, RTL*). В 1980 были впервые представлены эти языки, тогда они считались прорывными технологиями, так как позволяли инженерам работать на верхнем уровне абстракции с помощью *RTL*-симуляторов. До появления этих языков была лишь возможность симулировать поведение разработок на логического элемента или схемы.

SystemVerilog был разработан как революционный переход от *VHDL* и *Verilog* для поддержки сложности разработок *SoC*. Он является гибридным расширением *Verilog* и содержит в себе *HDL*-описание аппаратуры, так и объектно-ориентированный подход к программированию. Включает в себя возможности формальной проверки на основе логических утверждений и разработки текстов производительности.

Министерство Обороны США финансировало разработку языка *VHDL*.

Язык *Verilog* был разработан компанией *Gateway Design Automation*. После компания *Cadence Design Systems* приобрела компанию *Gateway* и предоставила *Verilog HDL* в общее пользование, и он стал индустриальным стандартом.

1.3.4.1 Сравнение *VHDL*, *Verilog*, *SystemVerilog*

VHDL и *Verilog* считаются обычными языками для разработки цифровых устройств, в то время как *SystemVerilog* представляет расширенную версию *Verilog*. Каждый язык имеет свой стиль и индивидуальные характеристики. Концепция и синтаксис языка *VHDL* были унаследованы от языка программирования *Ada*, в то время как *Verilog* ссылается на ранний *HDL*, который назывался *Hilo*, и имеет стиль языка программирования *C*.

Инженеры, разрабатываемые продукт на языках *Verilog* и *VHDL*, имеют возможность представить необходимую функциональность в виде текста программы. Затем для проверки симулируется модель, тем самым

можно убедиться, что дизайн работает так, как требуется. В модели могут быть подкорректированы любые проблемы, и симуляция проверит эту коррекцию.

VHDL это богатый, строго типизованный и строго детерминистический язык, более подробный в сравнении с *Verilog*. В результате проекты, написанные на *VHDL*, считаются самодокументированными. Инженеры, работающие в *VHDL*, постоянно сталкиваются с необходимостью явного преобразования из одного типа данных в другой, так как синтаксис сильно отличается от стиля языка *C*. Ошибки, которые может пропустить *Verilog*, *VHDL* сразу показывает. *VHDL* имеет подчеркнуто однозначно недвусмысленную семантику, и поэтому легче переносится между разными системами разработки (в том смысле, что перенос точнее отображает все тонкости работы исходного проекта).

Verilog в свою очередь более краткий, слабо проверяет типы. Также является детерминистическим. Все типы данных имеют битовое представление и заранее определены в *Verilog*. Синтаксис похож на *C*.

SystemVerilog включает ряд расширений *Verilog HDL* с целью помочь инженерам в разработке и проверки больших и более сложных проектов. Многие наблюдатели в сфере индустрии считают *SystemVerilog* первым языком типа *Hardware Description and Verification Language (HDVL)*, так как он объединяет в себе функции *VHDL* и *Verilog*, связанные с *Hardware Verification Language (HVL)* и поддерживает объекты. Таким же образом эволюционировал и язык *C*, когда на его основе был создан язык *C++*.

Из-за своей структуры *VDHL* отлавливает больше ошибок уже на ранних стадиях процесса разработки. С другой стороны *Verilog* позволяет инженерам быстро описывать модели. *SystemVerilog* пытается взять лучшие функции из обоих языков, и включает в себя функции *HVL* для поддержки разработки тестов производительности и техник формальной проверки.

Исторически сложилось, что язык *Verilog* получил наибольшее распространение в микроэлектронной индустрии. Благодаря небольшим требуемым вычислительным ресурсам прежних рабочих станций и достаточно точным результатам моделирования СБИС.

1.3.4.2 Преимущества VHDL.

При моделировании на уровне вентилях и транзисторах *VHDL* проигрывал в быстродействии *Verilog*. Однако *VHDL* – более универсальный и гибкий язык. *VHDL* - это строгий, стройный, универсальный и расширяемый язык, поэтому он получил широкое распространение в университетах и исследовательских учреждениях. Так, например, появились пакеты *VHDL* для аналогового моделирования, моделирования многозначной логики. Кроме того, симуляторы *VHDL* были гораздо дешевле симуляторов *Verilog*.

Все современные САПР микроэлектроники имеют компиляторы как с *Verilog*, так и с *VHDL*. Программист, освоивший *VHDL*, без особого труда может перейти к программированию на языке *Verilog*. В отличие от обратного.

Фирмы - изготовители микросхем в своих САПР обеспечивают генерацию моделей результатов размещения и трассировки, описанных на *VHDL*.

Конструкции языка, такие как *entity*, *port map*, *configuration*, обеспечивают надежную и быструю стыковку блоков, разработанных разными фирмами и разработчиками, в различных сочетаниях.

VHDL позволяет описывать (моделировать) и проверять (моделировать) поведение требуемой системы до того, как инструменты синтеза переведут проект в реальное оборудование (ворота и провода), это является ключевым преимуществом *VHDL*.

Не менее важным аспектом является то, что *VHDL* позволяет описывать параллельные системы. *VHDL* - это язык потока данных в котором

каждый оператор рассматривается для одновременного выполнения, в отличие от языков процедурных вычислений, таких как *BASIC*, *C* и ассемблерный код, где последовательность операторов выполняется последовательно по одной инструкции.

Проект *VHDL* многоцелевой. Один раз создав блок расчета, его можно использовать во многих других проектах. Тем не менее такие параметры, как емкость, размер памяти, элементная база, состав блока и структура взаимосвязей *могут* быть настроены, под конкретный проект.

Проект *VHDL* переносим. Созданный для одной элементной базы проект вычислительного устройства может быть перенесен на другую элементную базу, например СБИС с различными технологиями.

Немалое преимущество *VHDL* по сравнению с *Verilog* - полная система типов. Система типов может использоваться для написания гораздо более структурированного кода [5].

Также в ходе прохождения обучения по специальности, мне удалось с отличием пройти курс изучения языка *VHDL*.

Исходя из всего вышесказанного и было решено реализовать устройство управления ресурсами на языке *VHDL* с использованием свободно распространяемой версии системы *WebPack ISE* фирмы *Xilinx*.

1.4 Разработка структурной схемы устройства

1.4.1 Описание структурной схемы устройства

За каждым общим ресурсом закреплено устройство управления ресурсами. Интерфейс взаимодействия устройства с процессором представлен на рисунке 4.

Устройство управления ресурсами принимает запрос на доступ к общему ресурсу (ОР) от процессов (потоков), выполняющихся в процессорах (рисунок 4).



Рисунок 4 – Интерфейс взаимодействия устройства управления ресурсами с процессором

Доступ к ОР заключается во входе процесса в свой критический интервал (КИ), являющийся участком программы, в котором происходит работа с ОР.

В ответ на запрос устройство управления, при незанятости ресурса, формирует сигнал «запрос принят» (внутренний сигнал УУР). Если приняты на обслуживание несколько запросов одновременно (рисунок 4), то в критический интервал (секцию, участок) войдет процесс с наибольшим приоритетом. При этом УУР формирует сигнал «вход в критический интервал», разрешающий процессу выполнение необходимых действий над общим ресурсом. Если в критическом интервале уже находится некоторый процесс, то запрашивающий процесс не войдет в КИ, поскольку будет сформирован сигнал взаимного исключения. После завершения выполнения критического участка процесс (поток) выходит из критического интервала с выдачей сигнала «выход из КИ», а УУР в ответ выдает процессу сигнал «продолжить процесс».

Структура многопроцессорной системы с устройством синхронизации процессов представлена на рисунке 5.

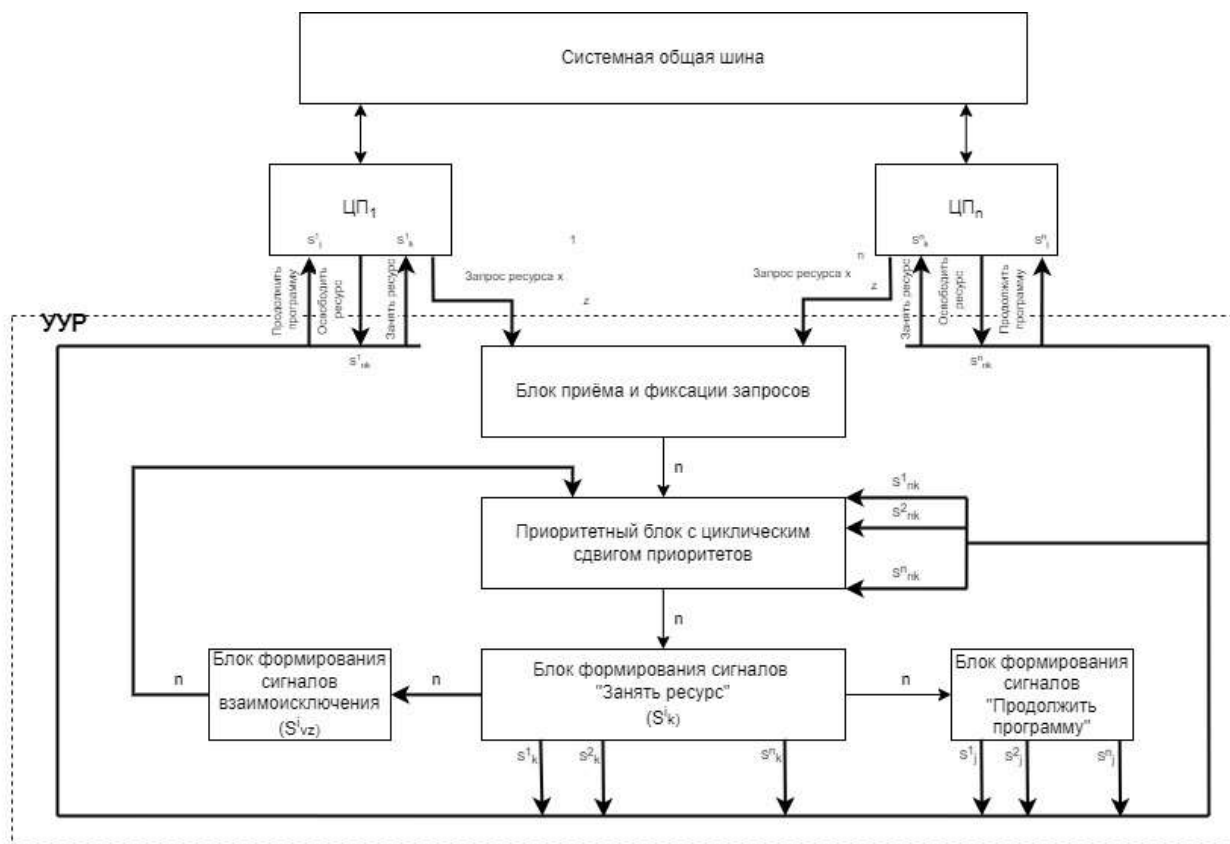


Рисунок 5 - Структура многопроцессорной системы с устройством синхронизации процессов

В ЦП n (центральный процессор) реализуется n процессов. Все процессы обращаются к одному общему ресурсу, но каждый из них работает с ОР на своем участке программы, который называется критическим интервалом (КИ). Все ЦП имеют специальные выходы для взаимодействия выполняющихся программ (процессов) с устройством синхронизации. Выходные данные из процессора: x_z – запрос ресурса, S_{nk} – сигнал освобождения ОР. Входные данные такие как: S_k – занять ОР, S_j – продолжить работу.

Таким образом, процесс выдает запрос к ОР, который преобразуется в специальный аппаратный сигнал на выводе процессора x_z . Если устройство

управления ресурсами разрешает процессу войти в критический интервал, то на вход процессора формируется специальный сигнал S_k .

По окончании работы в критическом интервале, процесс формирует сигнал на выводе S_{nk} . По этому сигналу устройство управления ресурсами в соответствии с принципами взаимоисключения и приоритетности разрешает вход в критический интервал следующему процессу, от которого были приняты запросы. Ранее вышедший из критического интервала процесс, сформировавший сигнал S_j выполняет основную работу, но за пределами критического интервала.

В самом же устройстве управления ресурсами происходит обмен сигналами между блоками. Блок приема и фиксации запросов принимает сигналы x_z от процессоров и фиксирует их. В приоритетном блоке выбираются запросы с наивысшим приоритетом. В блоке формирования сигналов «Занять ресурс» формируются выходные сигналы S_k . В блоке формирования сигналов взаимоисключения формируются внутренние для устройства управления ресурсами сигналы S_{vz} , которые поступают на вход блока приоритетов. Также формируются выходные сигналы «Продолжить программу» S_j в блоке формирования сигналов «Продолжить программу», которые, в свою очередь, затем поступают на вход процессора.

Схема электрическая структурная представлена на рисунке 6 (см. также приложение В). Она была разработана на основе структуры многопроцессорной системы с устройством синхронизации процессов (рисунок 5). Схема содержит следующие блоки:

- ВхШД – входная шина данных;
- БЦП – блоки центральных процессоров;
- БПФЗ – блок приема и фиксации запросов;
- БЦСП – приоритетный блок с циклическим сдвигом приоритетов;
- БФСЗР – блок формирования сигналов «Занять ресурс»;
- БФСВ – блок формирования сигналов взаимоисключения;
- БФСПП – блок формирования сигналов «Продолжить программу»;

-ВыхШД – выходная шина данных.

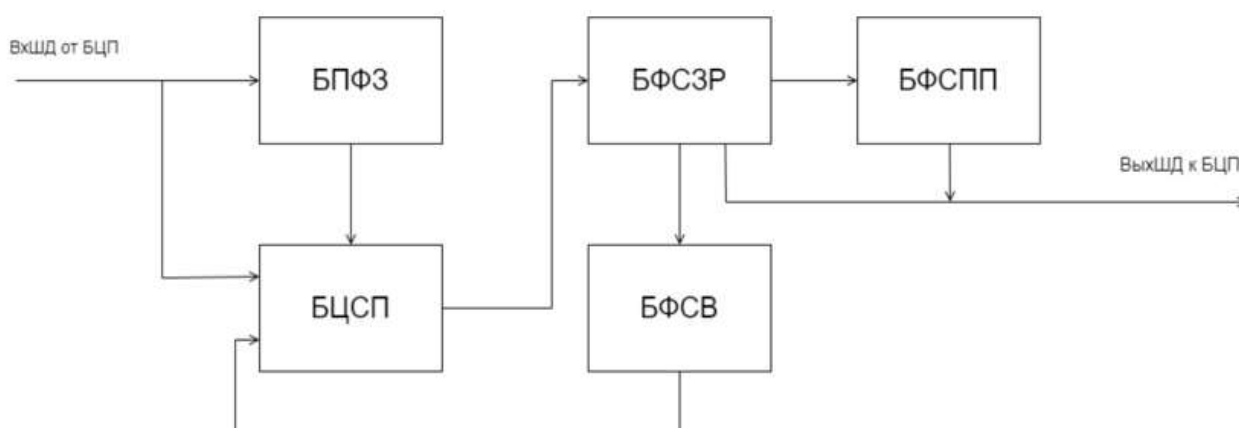


Рисунок 6 - Схема электрическая структурная устройства управления ресурсами многопроцессорной системы

БПФЗ предназначен для приема и фиксации запросов x_z , поступающих по входной шине данных от блоков центральных процессоров.

БЦСП предназначен для вычисления функций приоритетов. В зависимости от поступающих сигналов от БПФЗ, S_{vz} от БФСВ и сигналов S_{nk} от БПЦ по входной шине данных.

БФСЗР предназначен для формирования сигналов S_k «Занять ресурс». Подает сигналы на выходную шину данных к БЦП, а также на входы блоков БФСВ и БФСПП.

БФСВ предназначен для формирования сигналов взаимного исключения S_{vz} .

БФСПП предназначен для формирования сигналов S_j «Продолжить программу». Подает сигналы на выходную шину данных к БЦП.

1.5 Разработка функциональной схемы устройства

Взаимодействие устройства синхронизации с процессами, выполняющимися в центральных процессорах (ЦП), показано на рисунке 5. Алгоритм взаимодействия полностью аналогичен с традиционным. Будем считать, что в ЦП i реализуется i -й процесс. Пусть все процессы обращаются к

одному общему ресурсу, но каждый из них работает с ОР на своем участке программы, который называется критическим интервалом (КИ). Все ЦП имеют специальные выводы (в универсальных процессорах их нет) для взаимодействия выполняющихся программ (процессов) с устройством синхронизации. Процесс выдает запрос ОР, который преобразуется в аппаратный сигнал, действующий на специальном выводе процессора «Запрос ОР».

Если устройство синхронизации разрешает войти процессу в критический интервал, то формируется сигнал «Вход в КИ», воспринимаемый соответствующим процессом. Для этого процесс после выдачи сигнала «Запрос ОР» должен циклически опрашивать вывод «Вход в КИ» на предмет появления на нем единичного уровня.

По окончании работы в критическом интервале, процесс формирует единичный сигнал на выводе «Выход из КИ», по которому устройство синхронизации в соответствии с принципами взаимоисключения и приоритетности разрешает вход в КИ одному из процессов, от которых ранее были приняты запросы. Вышедший из КИ процесс по сигналу «Продолжить процесс» будет выполнять основную программу, но за пределами критического интервала.

Для синтеза УУР с аппаратной поддержкой будем использовать язык недетерминированных автоматов (НДА), позволяющий описывать все реализуемые в алгоритме управления события в виде графа или систем канонических уравнений (НД СКУ). Но только система канонических уравнений является основанием для структурного синтеза автомата, в результате которого получают функциональную схему автомата. Обычно при структурном синтезе производится преобразование НДА в детерминированный автомат, который синтезируется классическим образом.

Схема электрическая функциональная устройства управления ресурсами разрабатывалась с учетом структурной схемы, показанной на рисунке 6. Использовался интерфейс взаимодействия между отдельным процессором и

УУР, основанный на непосредственных связях. Это дает возможность освобождения системной общей шины от дополнительного трафика, вызванного синхронизацией процессов, что способствует сохранению пропускной способности общей шины для взаимодействия процессоров между собой и с общей памятью системы (общая память на рисунке не показана).

1.5.1 Разработка графа переходов недетерминированного автомата и системы канонических уравнений, описывающих функционирование устройства управлением ресурсов.

Ранее было отмечено, что управление доступом нескольких процессов к общему ресурсу заключается в избегании конфликтных ситуаций. Для этого необходимо синхронизировать свои действия таким образом, чтобы этот ресурс всегда находился в распоряжении только одного процесса из множества одновременно запрашивающих процессов. Все другие процессы переводятся в режим ожидания. В научной и учебной литературе сформулированы общие требования к алгоритму для наиболее эффективного решения такой задачи взаимодействия:

- в любой момент времени только один процесс может находиться внутри своего критического интервала;
- ни один процесс не может оставаться внутри критического интервала бесконечно долго;
- ни один процесс не должен ждать бесконечно долго входа в критический интервал.

Рассмотрим каждое из них с точки зрения формализации на логике недетерминированных автоматов. Первое из сформулированных требований должно обеспечиваться взаимоисключением критических интервалов. В формализованной модели это требование может быть обеспечено только несовместимыми событиями. При этом должна учитываться также и приоритетность процессов.

Второе и третье требование должно быть формализовано таким образом, чтобы события, определяющие прием заявки на обслуживание только в том случае, если процесс не находится в критическом интервале. Тем самым будет исключаться повторное обращение к ОР для процесса, имеющего наибольший приоритет при наличии заявок на обслуживание других процессов, если не имеется особой договоренности относительно изменения приоритетности в процессе обслуживания процессов [1].

Система канонических уравнений, имеющая следующий вид (для i -го процесса) [4]:

$$S_k^i = S_{vp}^i \& S_{vz}^i \& S_{pr}^i \vee S_k^i \overline{S_p^i}, i = \overline{1, n} ,$$

$$S_{vz}^i = \bigwedge_{(\forall \alpha)[\alpha \neq i]} \overline{S_k^\alpha}, i = \overline{1, n} ,$$

$$S_{pr}^i(t+1) = S_{pr}^i(0) \vee S_{vp}^i S_{pk}^i \bigwedge_{(\forall \alpha)[\alpha \neq i]} \overline{S_{vp}^\alpha} \vee \\ \vee S_{pk}^{i \oplus 1} \bigwedge_{(\forall \alpha)[i \oplus 1 < \alpha \leq i-1]} \overline{S_{vp}^\alpha} \vee \dots \vee S_{pk}^{i \oplus (n-1)} \bigwedge_{(\forall \alpha)[i \oplus n-1 < \alpha \leq i-1]} \overline{S_{vp}^\alpha},$$

$$S_{pr}^i(0) = S_0 x_n S_{vp}^i \bigwedge_{(\forall \alpha)[\alpha < i]} \overline{S_{vp}^\alpha},$$

$$S_0(t+1) = x_0 \vee S_0 \overline{x_n} \vee S_T,$$

$$S_T = \bigvee_{i=1}^n S_{pk}^i \bigwedge_{(\forall \alpha)[\alpha < i]} \overline{S_{pk}^\alpha} ,$$

$$S_z^i(t+1) = x_z^i \vee S_z^i \overline{S_{vp}^i},$$

$$S_m(t+1) = S_k^1 S_{vp}^1 \vee S_k^2 S_{vp}^2 \vee \dots \vee S_k^n S_{vp}^n,$$

$$S_{nk}(t+1) = s_m,$$

$$S_r^i(t+1) = (S_{vp}^1 \vee S_r^i) \overline{S_p^i},$$

$$S_p^i(t+1) = S_{nk} S_k^i,$$

$$S_{pk}^i(t+1) = S_p^i \vee S_{pk}^i S_{pr}^i S_T,$$

$$S_j^i(t+1) = S_r^i S_p^i,$$

$$S_{vp}^i(t+1) = (S_z^i \vee S_{vp}^i) \overline{S_k^i},$$

где S_k^i - события, определяющие вход и нахождение i -го процесса в своем критическом интервале;

S_{vp}^i - события, определяющие прием заявки i -го процесса на обслуживание для обращения к разделяемым данным;

S_p^i - события, обеспечивающие выход i -го процесса из критического интервала после окончания процедуры обращения к разделяемым данным;

S_{vz}^i - комбинационное событие, обеспечивающее взаимоисключение критических интервалов на основе несовместимости событий S_k^i с другими событиями из их общего числа, равного n ;

S_{pr}^i - события, обеспечивающие заданное приоритетное обслуживание i -го процесса;

$S_{pr}^i(0)$ - комбинационные события, определяющие начальный приоритет обслуживания i -го процесса,

S_0 - начальное событие системы управления перед обращением к разделяемому ресурсу;

χ_n - сигнал инициализации системы управления;

χ_0 - сигнал приведения системы управления в начальное состояние;

$\overline{S_{vp}^\alpha}$ - события, определяющие восприятие заявки i -го процесса на обслуживание;

S_{pk}^i - события, определяемые как событие S_p^i , задержанное на один такт; это событие используется, в принятом варианте, для организации циклической дисциплины обслуживания;

S_T - комбинационные события, свидетельствующего об окончании цикла обслуживания процессов, от которых были восприняты заявки на обслуживание;

S_z^i - сокращенное обозначение события, определяющего наличие заявки i -го процесса и их обслуживание;

X_z^i - сигнал обращения к общему ресурсу i -го процесса;

S_m - сокращенное обозначение события, обеспечивающего начало процедуры реализации обращения к разделяемым данным;

S_{nk} - сокращенное обозначение события, являющегося заключительным в процедуре обращения к разделяемым данным;

S_r^i - сокращенное обозначение события, символизирующего ожидание условий выхода i -го процесса из критического участка;

S_j^i - сокращенное обозначение события, инициирующего продолжение работы i -го процесса после выхода из критического интервала [2].

Граф НДА, представляющий алгоритм взаимодействия при обращении к разделяемым данным представлен на рисунке 7.

1.5.2 Синтез функциональной схемы устройства

Структурный синтез заключается в том, что схемы представляется в виде элементов памяти и логических схем. Память автомата состоит из триггеров, которые отображают все частные события. Существует несколько типов триггеров, с помощью которых можно отобразить события НДА: RS , JK , и D и T – триггеры [6].

Система уравнений, определяющие события S_{vz}^i для 8-х процессов будут иметь следующий вид:

$$S_{vz}^1 = \overline{S_k^2} \overline{S_k^3} \overline{S_k^4} \overline{S_k^5} \overline{S_k^6} \overline{S_k^7} \overline{S_k^8},$$

$$S_{vz}^2 = \overline{S_k^1} \overline{S_k^3} \overline{S_k^4} \overline{S_k^5} \overline{S_k^6} \overline{S_k^7} \overline{S_k^8},$$

$$S_{vz}^3 = \overline{S_k^1} \overline{S_k^2} \overline{S_k^4} \overline{S_k^5} \overline{S_k^6} \overline{S_k^7} \overline{S_k^8},$$

$$S_{vz}^4 = \overline{S_k^1} \overline{S_k^2} \overline{S_k^3} \overline{S_k^5} \overline{S_k^6} \overline{S_k^7} \overline{S_k^8},$$

$$S_{vz}^5 = \overline{S_k^1} \overline{S_k^2} \overline{S_k^3} \overline{S_k^4} \overline{S_k^6} \overline{S_k^7} \overline{S_k^8},$$

$$S_{vz}^6 = \overline{S_k^1} \overline{S_k^2} \overline{S_k^3} \overline{S_k^4} \overline{S_k^5} \overline{S_k^7} \overline{S_k^8},$$

$$S_{vz}^7 = \overline{S_k^1} \overline{S_k^2} \overline{S_k^3} \overline{S_k^4} \overline{S_k^5} \overline{S_k^6} \overline{S_k^8},$$

$$S_{vz}^8 = \overline{S_k^1} \overline{S_k^2} \overline{S_k^3} \overline{S_k^4} \overline{S_k^5} \overline{S_k^6} \overline{S_k^7},$$

где S_{vz}^i - комбинационное событие, обеспечивающее взаимоисключение критических интервалов на основе несовместимости событий S_k^i с другими событиями из их общего числа, равного n .

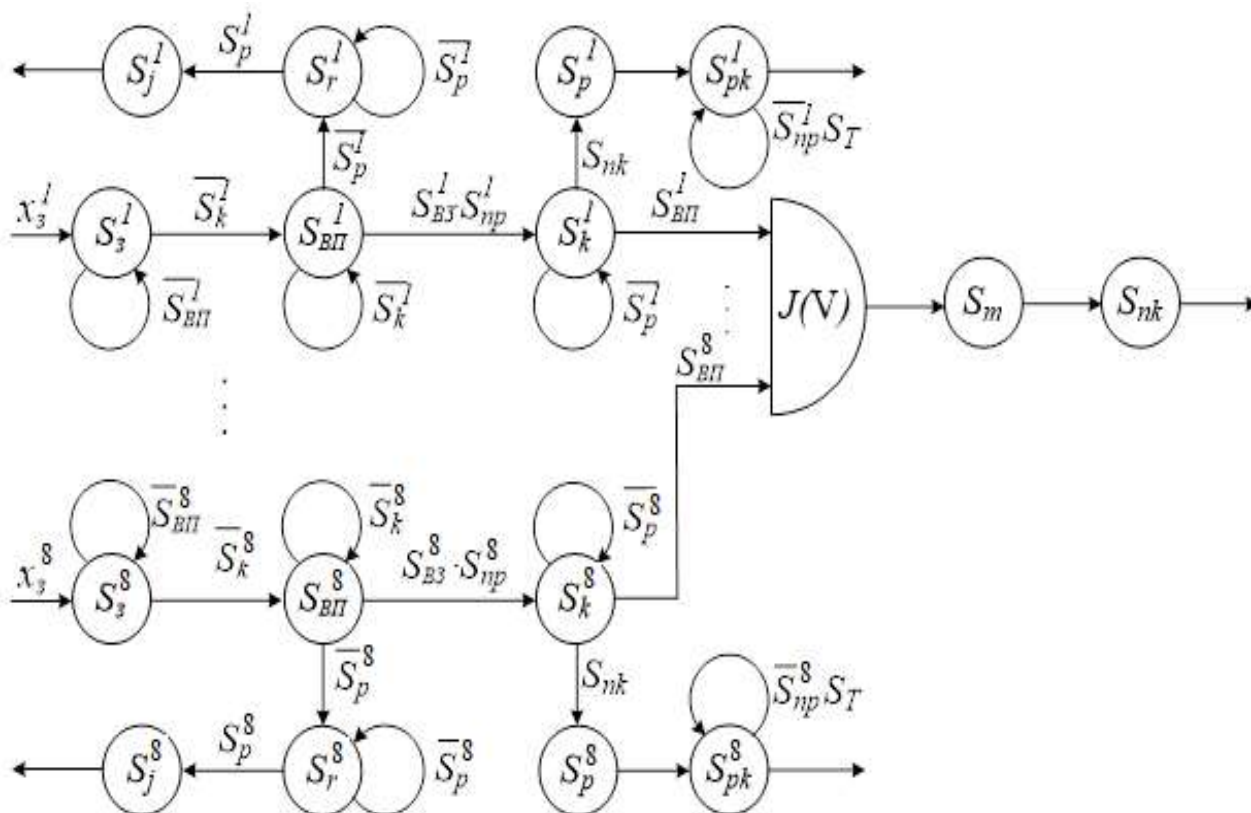


Рисунок 7 - Граф НДА алгоритма управления взаимодействующими параллельными процессами при обращении к общему ресурсу для 8-процессов

Схема блока выделения сигналов взаимоисключения представлена на рисунке 8.

Схема полностью соответствующая системе уровней определяющих события S_{vz}^i .

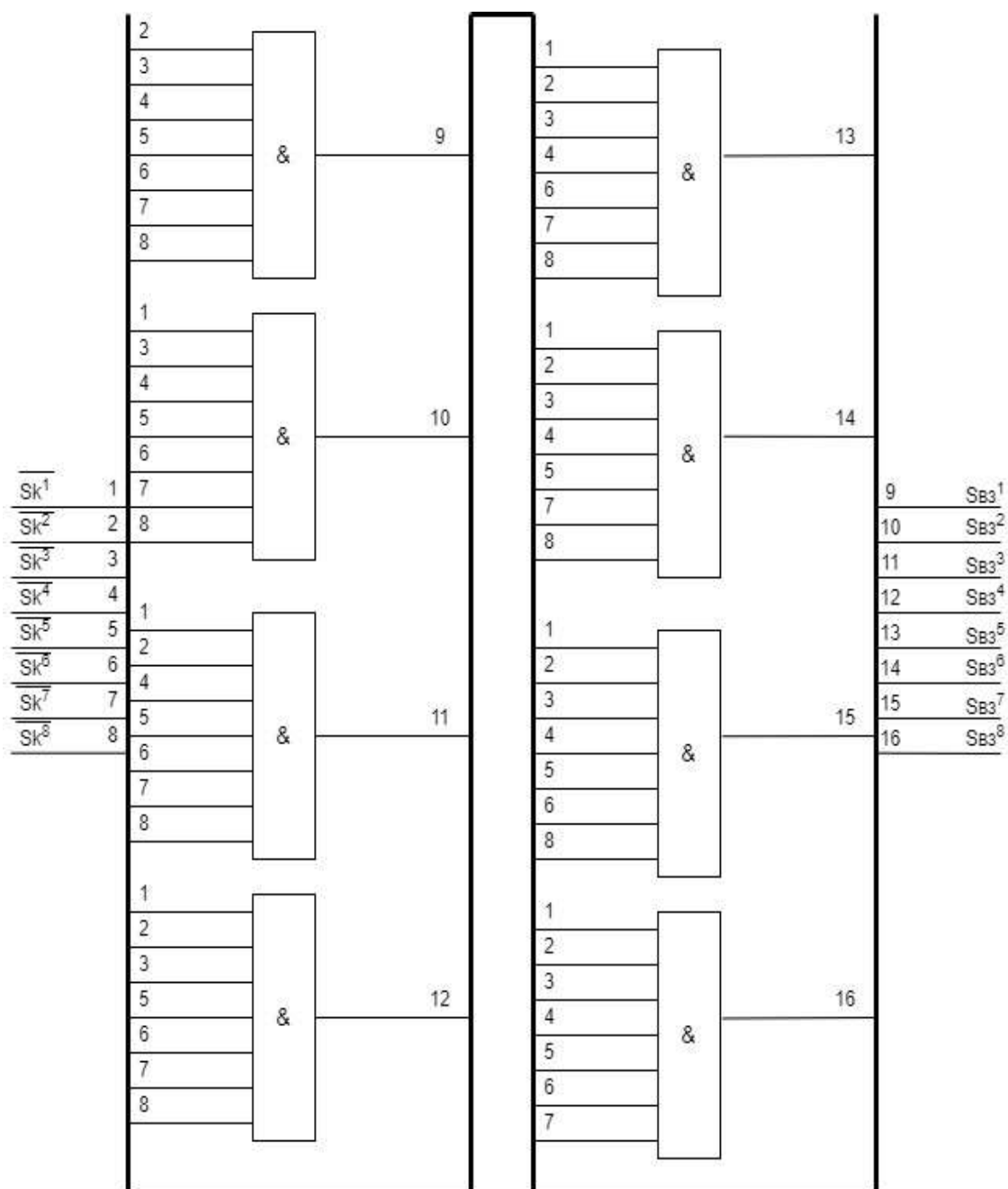


Рисунок 8 – Блок выделения сигналов взаимоисключения

Схема блока выделения начального приоритета представлена на рисунке 9.

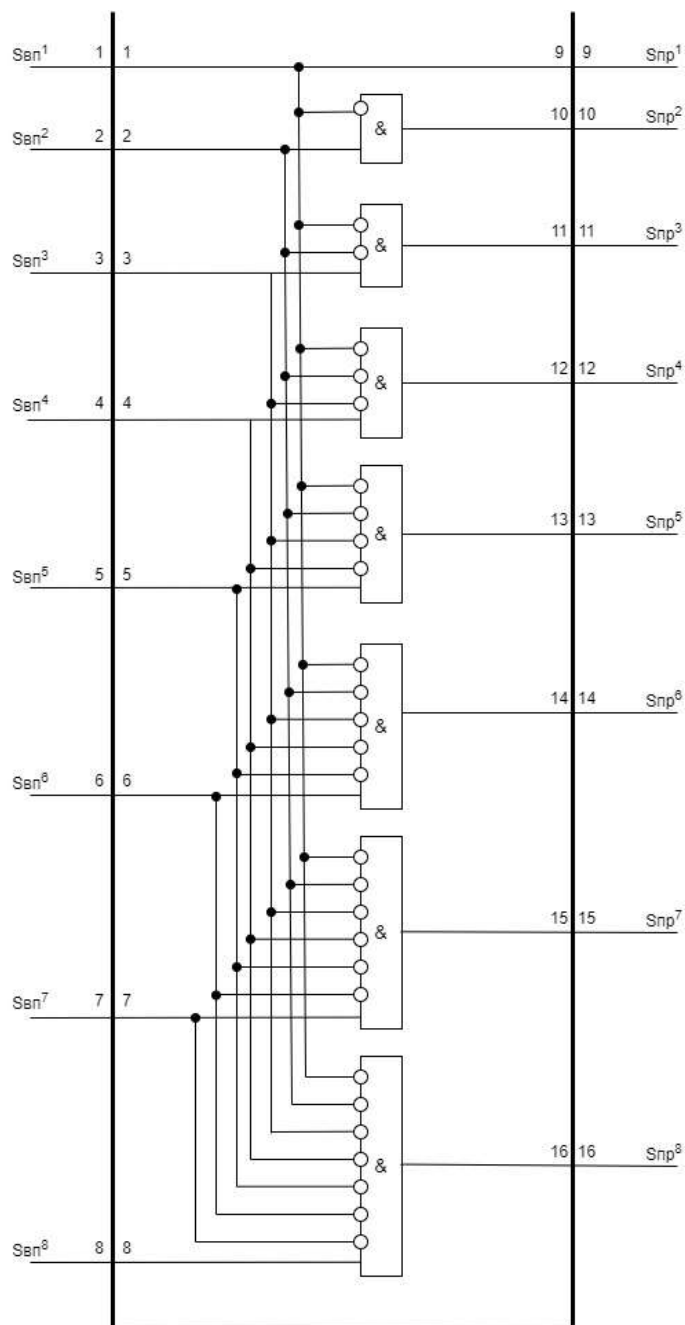


Рисунок 9 – Блок выделения начального приоритета

Блок выделения сигналов приоритета имеет динамическую смену приоритетов путем их сдвига влево, т.е. процесс, имевший первый приоритет, в следующем цикле обслуживания получит восьмой, второй процесс получит первый приоритет и т.д. Смена приоритетов определятся логическим выражением:

Изм.	Лист	№ докум.	Подпись	Дата

$$S_{pr}^i(t+1) = S_{pr}^i(0) \vee S_{vp}^i S_{pk}^i \bigwedge_{(\forall \alpha)[\alpha \neq i]} \bar{S}_{vp}^\alpha \vee \\ \vee S_{pk}^{i \oplus 1} \bigwedge_{(\forall \alpha)[i \oplus 1 < \alpha \leq i-1]} \bar{S}_{vp}^\alpha \vee \dots \vee S_{pk}^{i \oplus (n-1)} \bigwedge_{(\forall \alpha)[i \oplus n-1 < \alpha \leq i-1]} \bar{S}_{vp}^\alpha,$$

где $S_{pr}^i(0)$ - комбинационные события, определяющие начальный приоритет обслуживания i -го процесса.

Системы уравнений, определяющие события S_{pr}^i и для 8-х процессов будут иметь следующий вид:

$$S_{pr}^1(t+1) = S_{pr}^1(0) \vee S_{vp}^1 (S_{pk}^8 \vee S_{pk}^7 \bar{S}_{vp}^8 \vee S_{pk}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^4 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^3 \bar{S}_{vp}^4 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8), \\ S_{pr}^2(t+1) = S_{pr}^2(0) \vee S_{vp}^2 (S_{pk}^1 \vee S_{pk}^2 \bar{S}_{vp}^1 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^3 \bar{S}_{vp}^1 \bar{S}_{vp}^4 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^4 \bar{S}_{vp}^1 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^5 \bar{S}_{vp}^1 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^6 \bar{S}_{vp}^1 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^7 \bar{S}_{vp}^1 \bar{S}_{vp}^8 \vee S_{pk}^8 \bar{S}_{vp}^1), \\ S_{pr}^3(t+1) = S_{pr}^3(0) \vee S_{vp}^3 (S_{pk}^2 \vee S_{pk}^3 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^4 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^4 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^5 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^6 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^7 \bar{S}_{vp}^1 \bar{S}_{vp}^8 \vee S_{pk}^8 \bar{S}_{vp}^1 \vee S_{pk}^1 \bar{S}_{vp}^2), \\ S_{pr}^4(t+1) = S_{pr}^4(0) \vee S_{vp}^4 (S_{pk}^3 \vee S_{pk}^4 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^5 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^5 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^6 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^7 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^8 \vee S_{pk}^8 \bar{S}_{vp}^1 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \vee S_{pk}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \vee S_{pk}^2 \bar{S}_{vp}^3), \\ S_{pr}^5(t+1) = S_{pr}^5(0) \vee S_{vp}^5 (S_{pk}^4 \vee S_{pk}^5 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \bar{S}_{vp}^6 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^6 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \bar{S}_{vp}^7 \bar{S}_{vp}^8 \vee S_{pk}^7 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \bar{S}_{vp}^8 \vee \\ \vee S_{pk}^8 \bar{S}_{vp}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \vee S_{pk}^1 \bar{S}_{vp}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \vee S_{pk}^2 \bar{S}_{vp}^3 \bar{S}_{vp}^4 \vee S_{pk}^3 \bar{S}_{vp}^4),$$

$$\begin{aligned}
S_{pr}^6(t+1) &= S_{pr}^6(0) \vee S_{vp}^6 (S_{pk}^5 \vee S_{pk}^6 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^7} \overline{S_{vp}^8} \vee \\
&\vee S_{pk}^7 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^8} \vee S_{pk}^8 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \vee \\
&\vee S_{pk}^1 \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \vee S_{pk}^2 \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \vee S_{pk}^3 \overline{S_{vp}^4} \overline{S_{vp}^5} \vee S_{pk}^4 \overline{S_{vp}^5}), \\
S_{pr}^7(t+1) &= S_{pr}^7(0) \vee S_{vp}^7 (S_{pk}^6 \vee S_{pk}^7 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \overline{S_{vp}^8} \vee \\
&\vee S_{pk}^8 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \vee S_{pk}^1 \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \vee \\
&\vee S_{pk}^2 \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \vee S_{pk}^3 \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \vee S_{pk}^4 \overline{S_{vp}^5} \overline{S_{vp}^6} \vee S_{pk}^5 \overline{S_{vp}^6}), \\
S_{pr}^8(t+1) &= S_{pr}^8(0) \vee S_{vp}^8 (S_{pk}^7 \vee S_{pk}^8 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \overline{S_{vp}^7} \vee \\
&\vee S_{pk}^1 \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \overline{S_{vp}^7} \vee S_{pk}^2 \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \overline{S_{vp}^7} \vee \\
&\vee S_{pk}^3 \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \overline{S_{vp}^7} \vee S_{pk}^4 \overline{S_{vp}^5} \overline{S_{vp}^6} \overline{S_{vp}^7} \vee S_{pk}^5 \overline{S_{vp}^6} \overline{S_{vp}^7} \vee S_{pk}^6 \overline{S_{vp}^7}),
\end{aligned}$$

$$S_{pr}^1(0) = S_0 x_n S_{vp}^1,$$

$$S_{pr}^2(0) = S_0 x_n S_{vp}^2 \overline{S_{vp}^1},$$

$$S_{pr}^3(0) = S_0 x_n S_{vp}^3 \overline{S_{vp}^1} \overline{S_{vp}^2},$$

$$S_{pr}^4(0) = S_0 x_n S_{vp}^4 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3},$$

$$S_{pr}^5(0) = S_0 x_n S_{vp}^5 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4},$$

$$S_{pr}^6(0) = S_0 x_n S_{vp}^6 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5},$$

$$S_{pr}^7(0) = S_0 x_n S_{vp}^7 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6},$$

$$S_{pr}^8(0) = S_0 x_n S_{vp}^8 \overline{S_{vp}^1} \overline{S_{vp}^2} \overline{S_{vp}^3} \overline{S_{vp}^4} \overline{S_{vp}^5} \overline{S_{vp}^6} \overline{S_{vp}^7}.$$

Эти уравнения были использованы для описания приоритетного обслуживания на языке *VHDL* и для построения функциональной схемы блока выделения начального приоритета.

Фрагмент функциональной схемы устройства управления ресурсом для одного процесса приведен на рисунке 10, а полная электрическая функциональная для восьми процессов представлена в приложении Г.

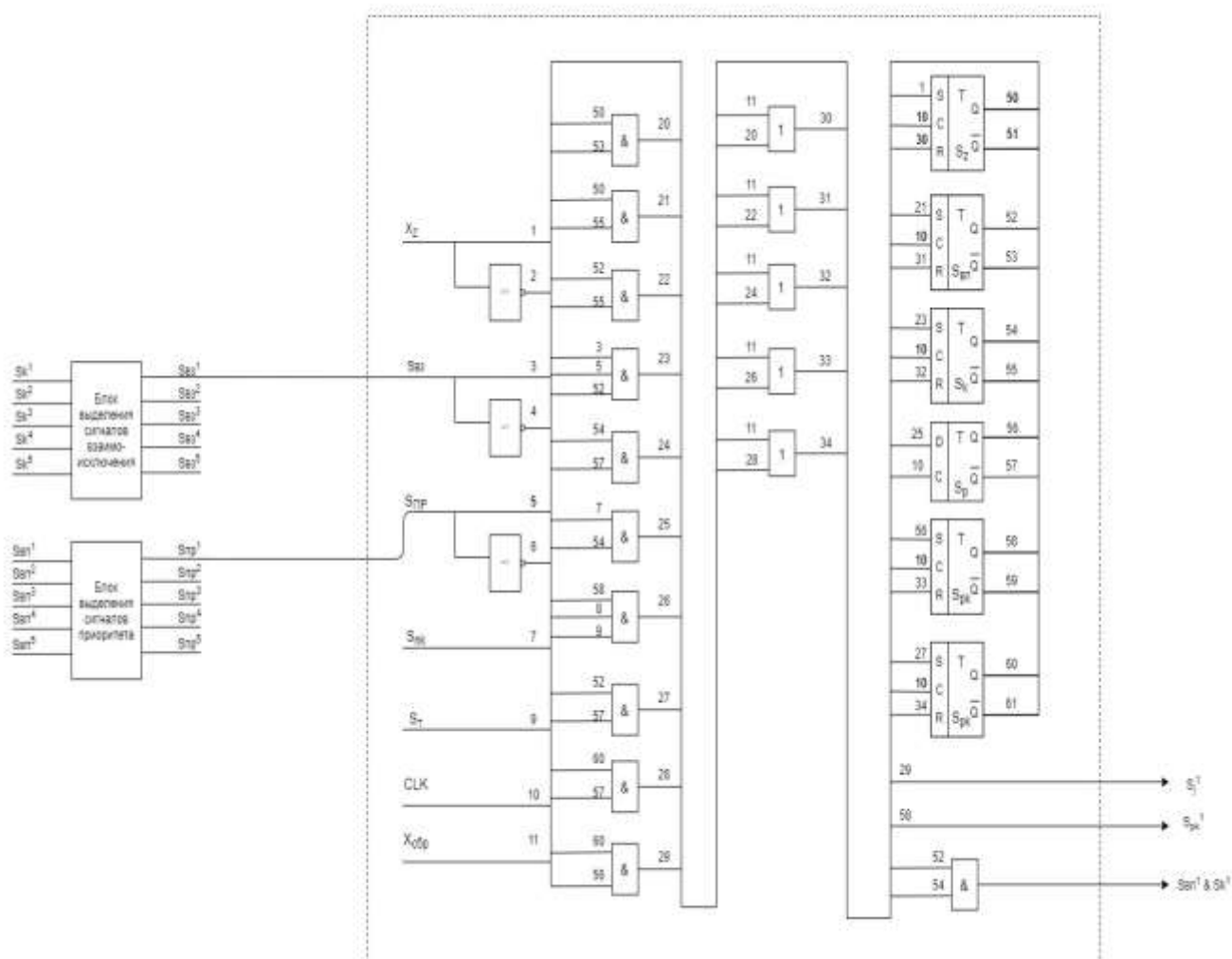


Рисунок 10 - Фрагмент схемы электрической функциональной устройства управления ресурсом для одного процесса

На схеме RS - триггеры фиксируют следующие события:

- S_z - события, определяющего наличие заявки i -го процесса и их обслуживание;
- S_{vp} - события, определяющие прием заявки i -го процесса на обслуживание для обращения к разделяемым данным;
- S_k - события, определяющие вход и нахождение i -го процесса в своем критическом интервале.

На схеме D - триггер фиксирует события:

- S_p - события, обеспечивающие выход i -го процесса из критического интервала после окончания процедуры обращения к разделяемым данным.

На схеме T - триггеры фиксируют следующие события:

- S_{pk} - события, определяемые как событие S_p^i , задержанное на один такт; это событие используется, в принятом варианте, для организации циклической дисциплины обслуживания.

Остальные логические элементы участвуют в выработке условий установки (зарождения события) и сброса (инверсный сигнал сохранения события) триггеров.

1.6 Разработка программы на языке VHDL и результаты моделирования

1.6.1 Составление программы моделирования

Модель устройства синхронизации была реализована на языке VHDL в виде двух программных модулей (рисунок 11).

Блок 1 регистрирует запросы к общему ресурсу и фиксирует вход i -го процесса в критический интервал. Блок 2 реализует логические функции управления общим ресурсом (отмечает начало S_m и конец S_{nk} процедуры обращения к ОР), а также функцию взаимного исключения и выделения приоритетного запроса на 8 запросов.

Первый модуль реализует логические функции 1-го блока, задаваемые выражениям, кроме формул для S_m и S_{nk} . Этот модуль используется в дальнейшем как компонент во втором модуле, представляющем собой модель восьмиканального устройства синхронизации. Число модулей первого типа в устройстве синхронизации равно числу каналов обслуживания запросов 8.

В приложении Д представлена схема программы функционально-логической модели устройства управления ресурсами.

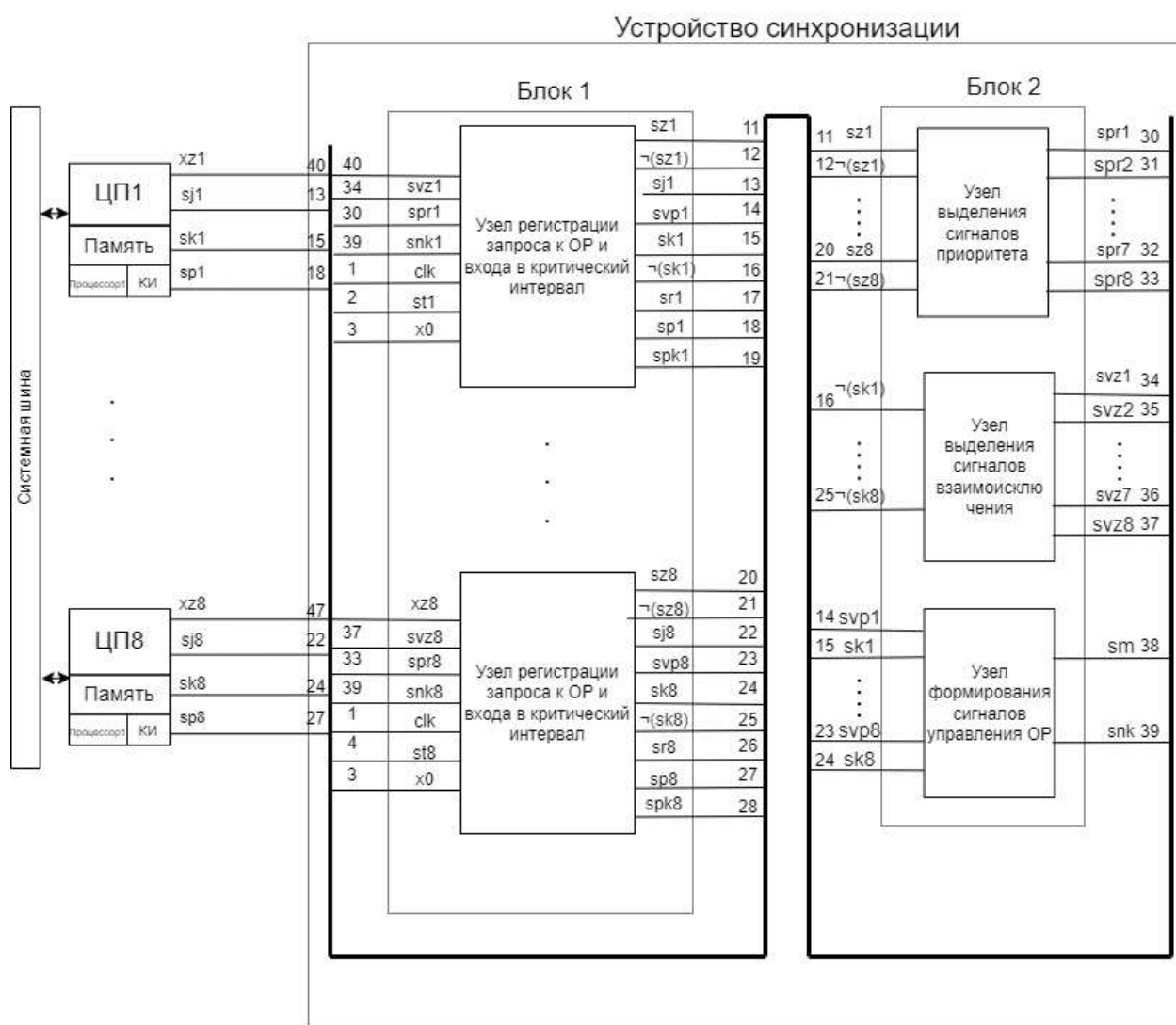


Рисунок 11 – Схема модели устройства синхронизации для восьми процессов

В блоке 2 происходит инициализация используемых переменных. В блоке 3 осуществляется формирование запросов на обращение к общему ресурсу. Затем происходит проверка поступления запроса i , если запрос i поступил, то в блоке 5 происходит проверка принятия запроса на обслуживание (расчет переменной Svp), если же запрос i не поступил происходит возврат к блоку 3. Если запрос принят на обслуживание, то в блоке 6 происходит расчет переменной Sz , тем самым запрос фиксируется, иначе происходит возврат к блоку 5, чтобы не произошла потеря запросов. В блоке 7 происходит проверка, занят ли на данный момент критический интервал. Критический интервал занят, то происходит возврат к блоку 6,

иначе переход к блоку 8. В котором происходят вычисления функций взаимного исключения Svz и приоритетов Spr . Затем в блоке осуществляется расчет переменной Sk , самым процесс входит в КИ. В блоке 10 возобновлена работа в КИ. В блоке 11 проверяется закончена ли работа в КИ. Если не закончена происходит возврат к блоку 10, иначе происходит расчет переменной Sp в блоке 12, тем самым реализуется функция выхода из КИ. В блоке 13 вычисляется переменная продолжения работы в процессе после выхода из КИ Sj . В блоке 14 реализуется функция циклического сдвига приоритетов (расчет переменных Spk). В блоке 15 происходит проверка, закончена ли работа с общим ресурсом, если да, то конец работы программы, иначе переход к блоку 3.

В ходе выполнения дипломной работы была разработана программа на языке функционального моделирования VHDL, которая полностью описывает систему канонических уравнений, код программы представлен в приложении А [7].

1.6.2 Анализ результатов моделирования

На рисунках 12 – 15 представлены результаты моделирования разрабатываемого устройства в виде временных диаграмм с использованием симулятора *Xilinx ISE Simulator*. События обозначены следующим образом:

- xz – сигнал запроса ресурса;
- szo – сигнал о фиксации запроса в регистре;
- $svpo$ – сигнал восприятия заявки процесса на обслуживание;
- sks – сигнал о входе в критическую секцию;
- $spro$ – сигнал об окончании работы в критической секции запроса;
- sjo – сигнал, инициирующий продолжение работы процесса после выхода из критического интервала;
- $spko$ – сигнал, определяемый как $spro$, задержанный на один такт (это событие используется, для организации циклической дисциплины обслуживания);

- *smo* – сигнал начала работы в критическом интервале (при моделировании процессу дали работать 1 такт);

- *snko* – сигнал конца работы в критическом интервале;

- *sto* – сигнал окончания цикла обслуживания;

- *s0o* – сигнал о смене приоритетов.

На рисунке 12 показана корректная работа устройства управления ресурсами, так как подобрана необходимая частота поступления запросов ресурса.

В первом цикле поступили 8 запросов одновременно (*xz(0-7)* в промежутке от 0 до 200 *ns*). Они фиксируются в регистре запросов *szo*, если запросы не находятся в критической секции и все они принимаются к обслуживанию (*svp(0-7)*). В строках *sko(0-7)* показано как процессы входят в критический интервал и выполняют какую либо работу в нем. Строки *sro(0-7)* сигнализируют об окончании работы в критическом интервале. Так же в строках *smo* и *snko* можно увидеть начало и конец работы в критическом интервале. В реальной системе количество тактов работы в критической секции может быть любым. Для моделирования работы процесса в критической секции процессу выделено 8 процессорных тактов, что не является критичным. Событие *sto*, сигнализирует об окончании цикла обслуживания (в данном случае моменты времени 600, 880, 1190, 1500, 1680 *ns*). Событие *s0o* сигнализирует устройству управления ресурсов о циклической смене приоритетов [8].

- частота поступления запросов в условиях отсутствия насыщения устройства синхронизации процессов была оценена в количестве тактов между запросами, которая равна 1/50 в каждом цикле обслуживания.

Латентность (реакция на запрос) — задержка от момента появления запроса на доступ к общему ресурсу и до момента входа в критическую секцию в условиях бесконфликтности. Соответственно, необходимо измерить задержку от переднего фронта сигнала $xz(0)$ до переднего фронта сигнала $sko(0)$ и она равна 35 ns.

Для определения латентности, нам необходимо так же определить продолжительность одного такта. Как видно на рисунке 13, продолжительность одного такта равно 5 ns.

$$\text{Латентность} = \frac{\text{Время задержки}}{\text{Продолжительность одного такта}} = \frac{35}{5} = 7.$$

Таким образом, мы определили латентность запроса в условиях бесконфликтности, которая составляет 7 тактов.



Рисунок 13 – Определение латентности запроса

Для определения частоты поступления запросов (f) необходимо измерить длительность цикла обслуживания (T) - интервал между передним фронтом поступающего запроса $xz(7)$ и задним фронтом сигнала

завершения работы в критической секции $sko(7)$ (рисунок 14). Он составил 500 ns.

$$T = \frac{t_{\text{цo}}}{t_{\text{т}}} = \frac{500}{10} = 50,$$

где $t_{\text{цo}}$ – длительность цикла обслуживания,

$t_{\text{т}}$ – длительность такта, равная 5 ns.

$$f = \frac{1}{T} = \frac{1}{50} = 0,02 \text{ 1/такт.}$$

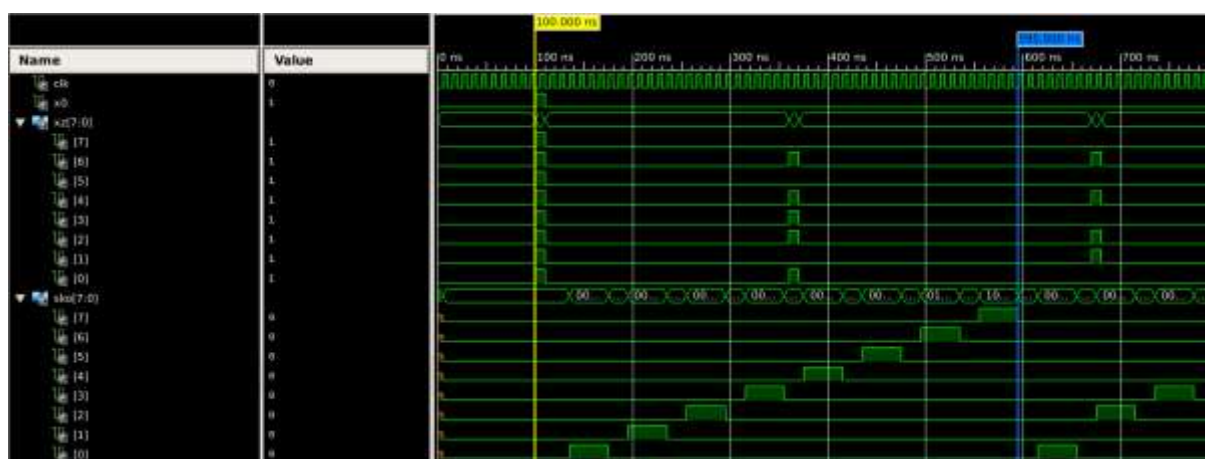


Рисунок 14 – Определение частоты поступления запросов

На рисунке 15 заданная интенсивность заявок выше пропускной способности устройства управления ресурсами, в результате чего утеряны запросы 4 и 6. Это решается изменением интервалов времени подачи заявок. Изображена некорректная работа устройства управления ресурсами, когда происходит насыщение работы устройства, из-за большой частоты поступления запросов общего ресурса (частота = 1/25 тактов).

Мы можем видеть, как четвертый и шестой запросы из второго цикла поступают в момент времени, когда в обслуживании находятся запросы от этих же источников из предыдущего цикла обслуживания (помечено в красные прямоугольники). Поэтому эти запросы теряются из-за слишком частого их поступления.

2 Технико-экономическое обоснование разработки системы

2.1 Введение

Чтобы определить цель технико-экономического обоснования, необходимо количественно и качественно доказать экономическую целесообразность улучшения приложения, а также определить организационно-экономические условия его эффективного функционирования.

Важным показателем эффективности программного модуля является его качество и эффективность процесса разработки. Качество программного изделия зависит от трех составляющих:

- с точки зрения пользователя данного программного продукта;
- с точки зрения использования ресурсов и их оценки;
- с точки зрения выполнения требований на программный продукт.

Чтобы программная продукция выполняла свои функции без лишних затрат ресурсов, необходимо разработать программу так чтобы она могла выполнять свои функции без дополнительных затрат ресурсов (оперативной памяти ПК и машинного времени, пропускной способности каналов передачи данных). Содержание технико-экономического обоснования разработки программы заключается в следующем:

- расчет технико-экономических показателей и выбор базы сравнения;
- определение трудоемкости и стоимости программного обеспечения (ПО);
- расчет цены ПО;
- расчет капитальных и эксплуатационных затрат на разработку;
- определение показателей финансово-экономической эффективности.

Представленные в данном разделе дипломного проекта результирующие

таблицы, позволяют сравнить результаты разработки с затратами на ее осуществление. Исходные данные для расчета экономических показателей приведены в таблице 1.

Таблица 1 – Исходные данные для расчета экономических показателей.

Обозначение	Наименование показателя	Единицы измерения	Значение показателя
C_n	Закупочная цена ЭВМ	Тыс. руб.	36
n_M	Рабочих дней в месяце	Дни	22
R_n	Рентабельность (норматив)	—	0,22
q_d	Коэффициент, дополнительной заработной платы разработчика	—	0,25
$q_{соц}$	Коэффициент, начислений органам социального страхования	—	0,25
q_n	Коэффициент, накладных расходов организации	—	1,35
w_I	Количество задач, решаемых потребителем	Зад. год	40
$t_{M.B}$	Время решения задачи разработанной программой	Маш. час	1
$m_{п}$	Количество организаций, которые приобретут данную программу	Шт.	1
$T_{эл}$	Стоимость 1 кВт/час	Руб.	3,89
C_c	Срок службы разработанной программы	Год	10
НДС	Налог на добавленную стоимость	%	20
K_p	Количество рабочих дней в году	Дн	264

Обозначение	Наименование показателя	Единицы измерения	Значение показателя
K_{CM}	Количество смен работы ЭВМ	—	2
t_{CM}	Продолжительность смены	Ч	10
A	Простои ЭВМ	%	5
M	Мощность, потребляемая ЭВМ	кВт	0,4
R_{CP}	Среднее количество ремонтов в год	—	1
C_D	Стоимость деталей, заменяемых при ремонте	Руб.	2000

2.2 Расчет затрат на разработку программы

Общая стоимость разработки программы оценивается по следующей формуле:

$$C_{RP} = C_{ЗП} + C_{НАК},$$

где $C_{ЗП}$ – затраты по заработной плате инженера-программиста;

$C_{НАК}$ – накладные расходы.

Заработная плата программиста рассчитывается по следующей формуле:

$$C_{ЗП} = ОЗП * (1 + q_{соц}) * (1 + q_d) * t_m,$$

где $ОЗП$ – основная заработная плата инженера-программиста за месяц (13000 руб.);

t_m – время, необходимое для разработки программы программистом i -го разряда (чел.-мес.);

q_d – коэффициент, учитывающий дополнительную заработную плату разработчика программы, в долях к сумме основной заработной платы;

$q_{\text{соц}}$ – коэффициент, учитывающий начисления органам социального страхования на заработную плату разработчика программы, в долях к сумме основной заработной плате разработчика.

Программа разрабатывалась 25 дней, если учесть, что в одном месяце 22 рабочих дня, то:

$$t_m = 25/22 = 1.1 \text{ (чел. –мес.)}.$$

Таким образом, затраты по заработной плате инженера-программиста составят:

$$C_{\text{ЗП}} = 13\,000 * (1+0,25) * (1+0,25) * 1,1 = 22\,343 \text{ (руб.)}.$$

Накладные затраты рассчитываются с учетом q_n – коэффициента, определяющего уровень накладных расходов организации по следующей формуле:

$$C_{\text{НАК}} = \text{ОЗП} * q_n * t_m ,$$

$$C_{\text{НАК}} = 13000 * 1,35 * 1,1 = 19\,305 \text{ (руб.)}.$$

Таким образом, суммарные затраты на разработку программы составляют:

$$C_{\text{рп}} = 22\,343 + 19\,305 = 41\,648 \text{ (руб.)}.$$

2.3 Расчет цены разработанной программы

Оптовая цена разработанной программы определяется по следующей формуле:

$$\text{ОЦ} = C_{\text{рп}} + (1 + \rho_n) ,$$

где ОЦ – оптовая цена (цена разработчика) (руб.);

$C_{\text{рп}}$ – суммарные затраты на разработку программы (руб.);

R_n – норматив рентабельности, учитывающий прибыль организации, разрабатывающей данную программу в долях ко всем затратам данной организации на разработку программы.

$$\text{ОЦ} = 41\,648 * (1 + 0,22) = 50\,810 \text{ (руб.)}.$$

Розничная цена программы рассчитывается с учетом налога на добавленную стоимость (НДС = 20%) по следующей формуле:

$$PC = OC + НДС = OC + OC * 0.2 = OC * 1.2,$$

$$PC = 50\,810 * 1.2 = 60\,972 \text{ (руб.)}.$$

Выручка от продаж при условии m_{Π} - количество организаций, желающих приобрести программу, составит:

$$B_{\text{пр}} = PC * m_{\Pi},$$

$$B_{\text{пр}} = 60\,972 * 1 = 60\,972 \text{ (руб.)}.$$

2.4 Расчет капитальных вложений

Капиталовложения, связанные с работой ЭВМ рассчитываются по следующей формуле:

$$C_{\text{ЭВМ}} = C_{\text{н}} + C_{\text{по}},$$

где $C_{\text{н}}$ – стоимость ЭВМ (руб.);

$C_{\text{по}}$ – стоимость программного обеспечения, необходимого для разработки устройства.

Примерная конфигурация ПК :

- процессор: *Intel Core i3-2,2 Гц*;
- ОЗУ: 4Гб ;
- видеокарта: *Nvidia Mx130-2048 Мб*;
- носитель: *SSD 256 Gb*.

Программа разрабатывается на бесплатной, свободно распространяемой версии системы *WebPack ISE* фирмы *Xilinx*. Поэтому $C_{\text{по}}=0$.

Капиталовложения в ЭВМ составляют:

$$C_{\text{ЭВМ}} = 36\,000 + 0 = 36\,000 \text{ (руб.)}.$$

2.5 Расчет эксплуатационных расходов

Расходы на эксплуатацию программы и системы, в комплексе с которой она работает, несет покупатель.

Эксплуатационные расходы на ЭВМ рассчитываются по следующей формуле:

$$P_3 = \frac{(t_{м.в} * P_ч) + PЦ}{C_c},$$

где $T_{м.в}$ – машинное время для решения задач с помощью разработанной программы, (маш.час/год);

$P_ч$ – эксплуатационные расходы, приходящиеся на 1 час работы ЭВМ;

$PЦ$ – цена, по которой продается программа (руб.);

C_c – срок службы программы (г).

Полезный фонд времени работы ЭВМ рассчитывается по следующей формуле:

$$\Phi_{пол} = \Phi_{общ} * t_{см} * (1 - a / 100),$$

где $\Phi_{общ}$ – общий фонд времени работы ЭВМ (дни); $T_{общ} = T_p$;

$K_{см}$ – количество смен работы ЭВМ;

$t_{см}$ – время одного рабочего дня (час);

a – простои ЭВМ (в % от общего фонда времени работы ЭВМ).

Полезный фонд времени работы ЭВМ получим:

$$\Phi_{пол} = 264 * 10 * (1 - 10 / 100) = 2\,376.$$

Машинное время для решения задач с помощью данной программы рассчитывается по следующей формуле:

$$T_{м.в} = w_1 * t_{м.в},$$

где w_1 – количество задач, решаемых потребителем в год(шт.);

$t_{м.в}$ – время решения задачи, разработанной программой (маш.час).

$$T_{м.в} = 40 * 1 = 40 \text{ (маш.час / год)}.$$

Эксплуатационные расходы, приходящиеся на 1 час работы ЭВМ, оцениваются по следующей формуле:

$$P_{\text{ч}} = \frac{(P_{\text{а}} + P_{\text{зп}} + P_{\text{эл}} + P_{\text{рм}})}{\Phi_{\text{пол}}},$$

где $P_{\text{а}}$ – амортизационные отчисления (руб.);

$P_{\text{зп}}$ – затраты по заработной плате инженера в год (руб./год);

$P_{\text{эл}}$ – стоимость потребляемой энергии (руб.);

$P_{\text{рм}}$ – затраты на ремонт ЭВМ (руб.);

$\Phi_{\text{пол}}$ – полезный годовой фонд работы ЭВМ, (маш.час/год).

Амортизационные отчисления рассчитываются с учетом нормы амортизации ($a_{\text{н}} = 12,5 \%$);

$$P_{\text{а}} = a_{\text{н}} * C_{\text{ЭВМ}} = 0,125 * 36\,000 = 4\,500 \text{ (руб.)}.$$

Затраты по заработной плате инженера за год рассчитывается по следующей формуле:

$$P_{\text{зп}} = (1 + q_{\text{соц}}) * (1 + q_{\text{д}}) * OЗП * 12,$$

где $q_{\text{с}}$ – коэффициент, учитывающий начисления органам социального страхования на заработную плату разработчика программы, в долях к сумме основной заработной плате разработчика.

$q_{\text{д}}$ – коэффициент, учитывающий дополнительную заработную плату разработчика программы, в долях к сумме основной заработной платы;

$OЗП$ – основная заработная плата инженера-программиста за месяц (13000 руб.);

Рассчитываем годовые затраты по заработной плате и социальным отчислениям для инженера-программиста:

$$P_{\text{зп}} = (1 + 0,25) * (1 + 0,25) * 13000 * 12 = 243\,750 \text{ (руб./ год)}.$$

Стоимость потребляемой энергии оценивается по следующей формуле:

$$P_{\text{эл}} = M * \Phi_{\text{пол}} * T_{\text{эл}},$$

где M – мощность, потребляемая ЭВМ (кВт);

$\Phi_{\text{пол}}$ – полезный годовой фонд работы ЭВМ (маш. час/год);

$T_{\text{эл}}$ – тариф за 1 кВт/час (руб. /кВт).

Итак, произведем расчет стоимости потребляемой энергии:

$$P_{ЭЛ} = 0,4 * 2\,376 * 3,89 = 3902,45 \text{ (руб.)}.$$

Затраты на ремонт ЭВМ вычисляются по следующей формуле:

$$P_{PM} = R_{CP} * C_d,$$

где R_{CP} – среднее количество ремонтов в год;

C_d – стоимость деталей заменяемых при одном ремонте, в среднем.

$$P_{PM} = 1 * 2000 = 2000 \text{ (руб.)}.$$

Произведем вычисление эксплуатационных расходов, приходящихся на 1 час работы ЭВМ:

$$P_{ч} = \frac{(4500 + 243\,750 + 3902,45 + 2000)}{2376} = 101,64 \text{ (руб. / час)}.$$

Далее вычислим эксплуатационные расходы на ЭВМ:

$$P_{э} = \frac{(40 * 102,64) + 60\,972}{10} = 6\,507 \text{ (руб.)}.$$

2.6 Расчет денежного годового экономического эффекта

Денежный годовой экономический эффект оценивается по следующей формуле:

$$ГЭ = M_{э} + k_n * K_{э},$$

где $M_{э}$ – экономия стоимости машинного времени, руб.;

E_n – нормативный коэффициент эффективности капитальных вложений.

$$E_n = 0,15;$$

$K_{э}$ – экономия капитальных вложений, руб..

Экономия капитальных вложений рассчитывается по следующей формуле:

$$K_{э} = \frac{T_{M.B} * C_{ЭВМ}}{\Phi_{пол}},$$

где $T_{M.B}$ – машинное время для решения задач с помощью разработанной программы (маш. час/год);

$$T_{M.B} = 40 \text{ маш. час/год}.$$

$C_{ЭВМ}$ – капиталовложения в ЭВМ (руб.);

$\Phi_{пол}$ – полезный годовой фонд работы ЭВМ (маш. час/год).

Произведем расчет экономии капитальных вложений по следующей формуле:

$$K_{Э} = \frac{40 \cdot 36\,000}{2\,376} = 581.$$

Расчет экономии стоимости машинного времени производится по следующей формуле:

$$M_{Э} = P_{ч} \cdot T_{м.в.},$$

где $P_{ч}$ – эксплуатационные расходы, приходящиеся на 1 час работы ЭВМ;

$T_{м.в.}$ – машинное время для решения задач с помощью разработанной программы (маш. час/год);

$$M_{Э} = 102,64 \cdot 40 = 4\,105,$$

$$ГЭ = 4\,105 + 0,15 \cdot 581 = 4\,192,1.$$

2.7 Итоговые расходы

Обобщенные технико-экономические показатели разработки программы сведены в таблицу 2.

Программа разрабатывалась 25 дней, если учесть, что в одном месяце 22 рабочих дня, то затраты на заработную плату программисту будут составлять – 40 625. То есть необходимо оплатить заработную плату за 2 месяца [9].

Таблица 2 - Итоговая таблица

Наименование показателя	Значения показателя (руб.)
Капиталовложения в ЭВМ	36 000
Затраты на разработку программы	41 648

Наименование показателя	Значения показателя (руб.)
Эксплуатационные затраты	6 507
Затраты на заработную плату программисту (руб.)	40 625
Итоговая стоимость	124 780

3 Безопасность жизнедеятельности. Рабочее место программиста

У людей, которые долгое время используют ПЭВМ могут быть такие реакции как нарушение функций зрения, быстрое общее утомление. По этой причине, чтобы избежать вреда от работы с вычислительной техникой необходимо соблюдать определенные меры безопасности и правильно организовать рабочее место. Обеспечение мер безопасности при работе ПЭВМ.

Освещенность рабочего места в помещении при использовании люминесцентных ламп должна составлять около 50–1000 лк.

Правильно подобранное расположение дисплеев по отношению к окну и осветительным приборам.

Время работы с дисплеем без перерыва не более 1 часа, продолжительность перерыва не менее 15 минут.

Некоторые специалисты считают работу с дисплеем не связанной со вредными радиобиологическими воздействиями. Для экрана на расстоянии 5 см от его поверхности допустимая мощность дозы рентгеновского излучения - не более 0.4 мр/ч. Уровень излучения экрана дисплея не превышает максимально допустимого. Допустимую дозу радиации и, следовательно, условия работы можно отнести к безопасным. Но в любом случае, необходимо соблюдать меры предосторожности: ограничить дневную продолжительность рабочей деятельности перед экраном, не размещать дисплеи концентрированно в рабочей зоне, использовать защитные экраны для дисплеев .

Размещение оборудованного дисплеем рабочего места необходимо так, чтобы не попадали окна или осветительные приборы, они должны находиться и непосредственно за спиной оператора. Потребуется уменьшить отражения на экране от разных источников искусственного и дневного света.

При смешивании искусственного света с естественным, следует использовать лампы наиболее близкие к солнечному свету. По отношению к ближайшему окружению соотношение яркости экрана должно быть не более 3:1.

Оптимальные значения температуры воздуха в помещении должны составлять 19-23 °С.

Рекомендуемая относительная влажность воздуха 55%.

Скорость движения воздуха не более 0.1 м/с.

3.1 Площадь и объем рабочих помещений

Если в помещении есть компьютеры, то помещение должно быть достаточно просторным и очень хорошо проветриваемым. Минимальная площадь на один компьютер – 6 м² минимальный объем – 20 м³.

3.2 Освещение

Для работы на компьютере чаще всего используются помещения с искусственным освещением. Искусственное освещение должно обеспечить правильную работу глаз и приблизить зрительное восприятие к оптимальному при естественном освещении.

Человек имеет как центральное (колбочковое), так или периферическое (палочковое) зрение. Первый вариант - для восприятия цветов и объектов малых размеров, второй – чтобы воспринимать окружающий фон с крупными объектами. Центральное зрение требует от человека большого количества ярких красок и яркого света в ночное время суток. С учетом того, что при работе с дисплеями используется центральное зрение. Поэтому в помещении, где находится компьютер необходимо достаточное освещение. Существуют следующие общие правила организации освещения:

- необходимо избегать большого контраста между яркостью монитора и окружающего пространства. Лучше всего их выравнять;

- компьютеры нельзя использовать в темных или полутемных помещениях. Освещение должно быть смешанным: естественным – за счет солнечного света - и искусственным.

Лучше всего использовать осветительные приборы, которые создают равномерную освещенность путем рассеянного или отраженного света (свет от ламп падает непосредственно на потолок) и исключают блики в экране монитора компьютера. Цифровой дисплей не имеет теней, а экран монитора и клавиши отображаются без помех. С учетом санитарных норм и правил, это могут быть преимущественно светодиодные лампы типа ЛБ с рассеивателями или экранирующими решетками. На зрение и нервную систему операторов влияет интенсивность пульсаций света люминесцентных ламп, поэтому для уменьшения коэффициента пульсаций используйте лампы с высокочастотными пускорегулирующими аппаратами.

3.2.1 Расчет освещенности рабочего места

Наилучшие условия для зрения обслуживающего персонала обеспечивают правильный выбор и расчет освещенности рабочего места. С точки зрения гигиены труда и культуры производства освещение должно соответствовать санитарным нормам. Хорошее освещение помогает устранить усталость и способствует улучшению темпа работы.

Для рационального освещения производственных помещений и рабочих мест большое значение имеет выбор цвета для окраски потолка, стен и производственного оборудования.

Рассчитаем люминесцентное освещение кабинета, предназначенного для выполнения работ с размером объекта различения от 0,3 мин до 1 мин. Размеры помещений: $A=3$ м, $B=4$ м, $H=2,5$ м. Площадь помещения определяем по следующей формуле:

$$S=A*B.$$

Подставив значения получим:

$$S=3*4=12 \text{ м}^2.$$

Освещение проектируется при помощи светильников ШОД минимальной освещенностью $E_{\min}=200$ лк, средней удельной мощностью 17-23 Вт/м². Высота подвеса над рабочей поверхностью $H_p=2$ м. Освещение выполняется лампами ЛБ-2*40, световой поток ламп $F_{\text{л}}=2480$ лм, длина 1.2 м, коэффициент запаса равен $\kappa=1.5$.

Определим показатель помещения по следующей формуле:

$$F=A*B/H_p(A+B).$$

Подставив значения получим:

$$F=3*4/2*(3+4)=0.86.$$

Затем, для $F=0,86$, коэффициентов отражения потолка $R_n=0,7$, стен $R_c=0,5$ и расчетной плоскости $R_p=0,3$ находим коэффициент использования светового потока $h=0,37$. Потребное число светильников определяется по следующей формуле:

$$N=E_{\min}*S*K/F_{\text{л}}*h*Z*n.$$

где $n = 2$ - общее число ламп в светильнике.

Подставив значения получим:

$$N=200*12*1,5/2480*0,37*0,9*2=2 \text{ штуки.}$$

Общее количество ламп равно $n=2*2=4$.

Таким образом, выполнение перечисленных выше требований к организации рабочего места обеспечит комфортные условия умственного труда.

3.3 Планировка и оснащение рабочего места

Организация рабочего места - это совокупность мер и мероприятий, направленных на обеспечение рабочего места средствами и предметами труда и размещение их в определенном порядке. Для создания рабочего места с использованием компьютера или видеомонитора необходимо учитывать расстояние между столами, которое должно быть не менее 2

метра. А расстояние между боковыми поверхностями видеомониторов - 1,5 м.

Для комфортной работы должны соблюдаться следующие требования к рабочему месту:

- оборудование рабочего места (стол, стул, подставка для ног) должны быть специальной конструкции, с возможностью индивидуальной регулировки;

- сиденье и спинка стула должны быть покрыты не электризующимися материалами;

- расположение рабочих поверхностей должно обеспечить согласованность компоновки рабочего места и маршрута движений, а также достаточную легкость для слежения за рабочими операциями,

- освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300 - 500 лк (при комбинированном освещении).

Для операторов на рабочем месте было выделено 20 пространственных параметров, которые представлены в таблице 3 и на рисунке 16.

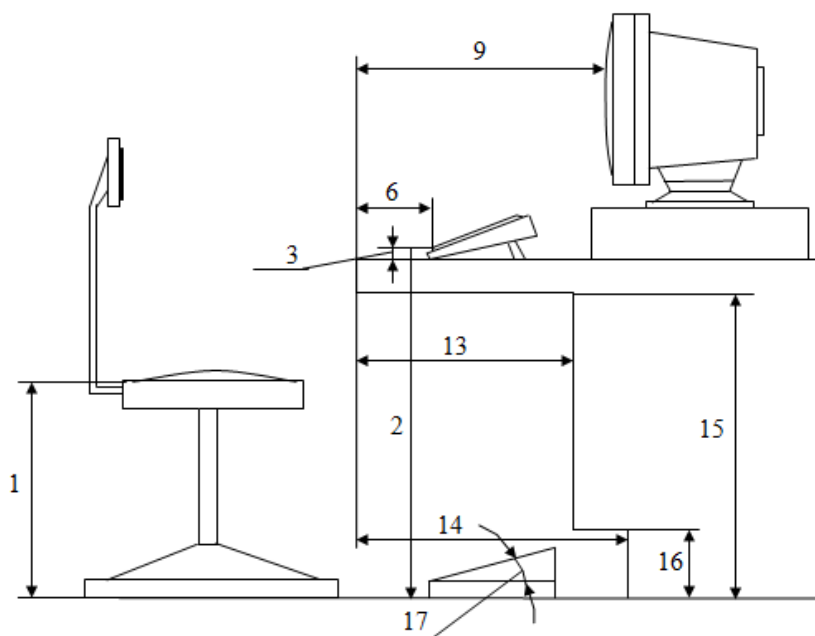


Рисунок 16 - Организация рабочего места оператора ПЭВМ

Таблица 3- Параметры рабочего места

Пространственные параметры	L, мм
1 Высота сидения	400-500
2 Высота клавиатуры от пола	600-750
3 Угол наклона клавиатура	7-15°
4 Ширина основной клавиатуры	≤ 400
5 Глубина основной клавиатуры	≥ 200
6 Удаление клавиатуры от края стола	80-100
7 Высота экрана от уровня пола	950-1000
8 Угол наклона экрана и нормали	0-30°
9 Удаленность экрана от края стола	500-700
10 Высота поверхностей для записей	670-850
11 Площадь поверхности для записей	600x400
12 Угол наклона поверхности для записей	0-100
13 Глубина пространства для ног в коленях	<400
14 Глубина пространства на уровне ступней	<600
15 Высота пространства для ног в коленях	<600
16 Высота пространства на уровне ступней	<100
17 Ширина пространства для ног на уровне	<500
18 Высота подставки для ног	50-130
19 Угол подставки для ног	0-25
20 Ширина подставки для ног	300

3.4 Основные способы защиты от вредного воздействия ПК

На сегодняшний день борьба с вредным воздействием ПК на человека ведется в нескольких направлениях. В первую очередь, это создание и внедрение стандартов, а также других регламентирующих документов, которые позволяют производить мониторы только с очень низким уровнем электромагнитных излучений. В соответствии со стандартом ТСО-91 напряженность электрического поля в диапазоне от 5 Гц до 2 кГц на расстоянии 50 см от экрана должна составлять 10 вольт/м, а в диапазоне от 2 кГц до 400 кГц - не более 1 вольт/м.

Второе направление - для мониторов с высоким уровнем электромагнитных излучений необходимо широко использовать специальные экраны-фильтры для защиты от излучения. Это средство обладает довольно большой эффективностью, позволяя использовать дешёвые мониторы и продолжать эксплуатировать имеющиеся видеотерминалы [10].

Заключение

В результате выполнения представленного дипломного проекта было разработано устройство управления ресурсами многопроцессорной системы. Устройство управления ресурсами предназначено для управления параллельными взаимодействующими процессами. Оно обеспечивает доступ параллельных процессов к общему ресурсу на основе метода критических интервалов.

Анализ временных диаграмм полученных в процессе моделирования устройства управления ресурсами, позволил выявить существенные преимущества такой организации в специализированных многопроцессорных системах, которые заключаются в повышении реакции (латентности) операционной системы на запрос к общим ресурсам и производительности всей системы в целом. Аппаратная реализация так же освобождает процессор от работы в ядре ОС при выполнении функции управления ресурсами, перекладывая эту работу на специальное устройство управления, это дополнительно способствует повышению производительности многопроцессорной системы.

Проектирование и тестирование устройства производилось на языке *VHDL* в среде разработки *Xilinx ISE 14.7*.

Список использованных источников

1. Вашкевич Н.П., Бикташев Р.А., Гурин Е.И. Аппаратная реализация функций синхронизации параллельных процессов при обращении к разделяемому ресурсу на основе ПЛИС / Известия вузов. Поволжский регион. Технические науки. № 2, 2007, с.3-12.
2. Бикташев Р.А. Многопроцессорные системы. Архитектура, топология, анализ производительности: Учебное пособие / Р.А. Бикташев, В.С. Князьков.- Пенза: изд-во Пенз. гос. ун-та, 2004.-108 с.
3. Бикташев Р.А. Структурная организация многопроцессорных систем с аппаратной поддержкой ядра ОС. Новые информационные технологии и системы: сборник статей XII Международной научно-технической конференции. - Пенза: Изд-во ПГУ, 2015. - С.90-93.
4. Бикташев Р.А. Структурный синтез устройств управления ресурсами в многопроцессорных системах на основе логики недетерминированных автоматов / Р.А. Бикташев, Н.П. Вашкевич.- Материалы докладов XXIII международной научно-технической конференции, посвященной 100-летию НГТУ - Нижегородского политехнического института. 2017.- С. 641-646.
5. В чем разница между *VHDL*, *Verilog* и *SystemVerilog*? // [Электронный ресурс] URL: <http://microsin.net/programming/xilinx/difference-between-vhdl-verilog-systemverilog.html> (дата обращения: 06.05.2022).
6. Вашкевич Н.П., Бикташев Р.А. Недетерминированные автоматы и их использование для реализации систем параллельной обработки информации. Монография. Пенза: Изд-во ПГУ. 2016.-396с.
7. Бибило П.Н. Основы языка *VHDL*; Изд. 3-е, доп. - М.: Издательство ЛКИ, 2007. - 328 с.

8. Зотов В.Ю. Проектирование цифровых устройств на основе ПЛИС фирмы *Xilinx* в САПР *WebPACK ISE*. – М.: Горячая линия - Телеком, 2003, 624 с.

9. Экономика предприятия: Учебник для вузов /Под ред. проф. В.Я. Горфинкеля, проф. В.А. Швандара. — 4-е изд., перераб. и доп. - М.: ЮНИТИ-ДАНА, 2007. - 670 с.

10. Козловский В. А. Безопасность жизнедеятельности: учебное пособие / В. А. А. В. Козловский Козловский, О. Л. Упоров. Екатеринбург: Изд-во Рос. гос. проф.-пед. ун-та, 2013. 314 с.

Приложение А

(обязательное)

Устройство управления ресурсами многопроцессорной системы

Листинг программы

					ПГУ 1.090501.06.001 ПЗ	Лист
						67
Изм.	Лист	№ докум.	Подпись	Дата		

Код программы файла VHDL:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity labVhdl is
Port (
clk, x0 : in std_logic;
xz, snk : in std_logic_vector(7 downto 0);
szo, sjo, svpo, sko, sro, spo, spko ,svzo,spro : out
std_logic_vector(7 downto 0);
smo, snko, sto ,s0o: out std_logic);

end labVhdl;
architecture Behavioral of labVhdl is
signal ssm,ssnk,st,s0 : std_logic:='0';
signal spr, svz, sz, sj, svp, sk, sr, sp, spk :
std_logic_vector(7 downto 0):="00000000";

begin

process (CLK)
variable i:integer;
begin
if CLK='1' and CLK'event then
i := 0;
while (i<8) loop
ssnk<=ssm;
sz(i) <= xz(i) or (sz(i) and (not svp(i)));
sr(i) <= (svp(0) or sr(i)) and (not sp(i));
svp(i) <= (sz(i) or svp(i)) and (not sk(i));
sk(i) <= (svp(i) and svz(i) and spr(i)) or (sk(i) and (not
sp(i)));
sp(i) <= ssnk and sk(i);
spk(i) <= sp(i) or (spk(i) and (not spr(i)) and st); -- !!!
sj(i) <= sr(i) and sp(i);

svz(0)<= (not sk(1)) and (not sk(2)) and (not sk(3)) and
(not sk(4))
and (not sk(5)) and (not sk(6))
and (not sk(7));

svz(1)<= (not sk(0)) and (not sk(2)) and (not sk(3)) and
(not sk(4))
and (not sk(5)) and (not sk(6))
and (not sk(7));

svz(2)<= (not sk(1)) and (not sk(0)) and (not sk(3)) and
(not sk(4))
```

Изм.	Лист	№ докум.	Подпись	Дата

```

                                and (not sk(5)) and (not sk(6))
and (not sk(7));

    svz(3)<= (not sk(1)) and (not sk(2)) and (not sk(0)) and
(not sk(4))
                                and (not sk(5)) and (not sk(6))
and (not sk(7));

    svz(4)<= (not sk(1)) and (not sk(2)) and (not sk(3)) and
(not sk(0))
                                and (not sk(5)) and (not sk(6))
and (not sk(7));

    svz(5)<= (not sk(1)) and (not sk(2)) and (not sk(3)) and
(not sk(4))
                                and (not sk(0)) and (not sk(6))
and (not sk(7));

    svz(6)<= (not sk(1)) and (not sk(2)) and (not sk(3)) and
(not sk(4))
                                and (not sk(5)) and (not sk(0))
and (not sk(7));

    svz(7)<= (not sk(1)) and (not sk(2)) and (not sk(3)) and
(not sk(4))
                                and (not sk(5)) and (not sk(6))
and (not sk(0));

    i := i + 1;
end loop;

    ssm<=(sk(0) and svp(0)) or (sk(1) and svp(1)) or (sk(2) and
svp(2)) or (sk(3) and svp(3)) or (sk(4) and svp(4))
    or (sk(5) and svp(5)) or (sk(6) and svp(6)) or (sk(7) and
svp(7));
    -----
    -----

    st<=
    (spk(0) and (not svp(1)) and (not svp(2)) and (not svp(3))
and (not svp(4)) and (not svp(5)) and (not svp(6)) and (not
svp(7))) or
    (spk(1) and (not svp(2)) and (not svp(3)) and (not svp(4))
and (not svp(5)) and (not svp(6)) and (not svp(7))) or
    (spk(2) and (not svp(3)) and (not svp(4)) and (not svp(5))
and (not svp(6)) and (not svp(7))) or
    (spk(3) and (not svp(4)) and (not svp(5)) and (not svp(6))
and (not svp(7))) or
    (spk(4) and (not svp(5)) and (not svp(6)) and (not svp(7)))
or
    (spk(5) and (not svp(6)) and (not svp(7))) or

```

```

(spк(6) and (not svp(7))) or
(spк(7));

s0<= x0 or st;

spr(0)<= (sz(0) and s0)
or (svp(0) and ((spк(7))
or (spк(0) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(4)) and (not svp(5)) and (not svp(6)) and
(not svp(7)))
or (spк(1) and (not svp(2)) and (not svp(3)) and (not
svp(4)) and (not svp(5)) and (not svp(6)) and (not svp(7)))
or (spк(2) and (not svp(3)) and (not svp(4)) and (not
svp(5)) and (not svp(6)) and (not svp(7)))
or (spк(3) and (not svp(4)) and (not svp(5)) and (not
svp(6)) and (not svp(7)))
or (spк(4) and (not svp(5)) and (not svp(6)) and (not
svp(7)))
or (spк(5) and (not svp(6)) and (not svp(7)))
or (spк(6) and (not svp(7))));
-----
spr(1)<= (sz(1) and (not sz(0)) and s0)
or (svp(1) and ((spк(0))
or (spк(1) and (not svp(0)) and (not svp(2)) and (not
svp(3)) and (not svp(4)) and (not svp(5)) and (not svp(6)) and
(not svp(7)))
or (spк(2) and (not svp(3)) and (not svp(4)) and (not
svp(5)) and (not svp(6)) and (not svp(7)) and (not svp(0)))
or (spк(3) and (not svp(4)) and (not svp(5)) and (not
svp(6)) and (not svp(7)) and (not svp(0)))
or (spк(4) and (not svp(5)) and (not svp(6)) and (not
svp(7)) and (not svp(0)))
or (spк(5) and (not svp(6)) and (not svp(7)) and (not
svp(0)))
or (spк(6) and (not svp(7)) and (not svp(0)))
or (spк(7) and (not svp(0))));
-----
spr(2)<= (sz(2) and (not sz(0)) and (not sz(1)) and s0)
or (svp(2) and ((spк(1))
or (spк(2) and (not svp(0)) and (not svp(1)) and (not
svp(3)) and (not svp(4)) and (not svp(5)) and (not svp(6)) and
(not svp(7)))
or (spк(3) and (not svp(1)) and (not svp(4)) and (not
svp(5)) and (not svp(6)) and (not svp(7)) and (not svp(0)))
or (spк(4) and (not svp(1)) and (not svp(5)) and (not
svp(6)) and (not svp(7)) and (not svp(0)))
or (spк(5) and (not svp(1)) and (not svp(6)) and (not
svp(7)) and (not svp(0)))
or (spк(6) and (not svp(1)) and (not svp(7)) and (not
svp(0)))
or (spк(7) and (not svp(1)) and (not svp(0)))

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.06.001 ПЗ

Лист

70

```

    or (spk(0) and (not svp(1))));
    ----
    spr(3)<= (sz(3) and (not sz(0))and (not sz(1))and (not
sz(2)) and s0)
    or (svp(3) and ((spk(2))
    or (spk(3) and (not svp(0)) and (not svp(1)) and (not
svp(2)) and (not svp(4)) and (not svp(5)) and (not svp(6)) and
(not svp(7)))
    or (spk(4) and (not svp(1)) and (not svp(2)) and (not
svp(5)) and (not svp(6)) and (not svp(7)) and (not svp(0)))
    or (spk(5) and (not svp(1)) and (not svp(2)) and (not
svp(6)) and (not svp(7)) and (not svp(0)))
    or (spk(6) and (not svp(1)) and (not svp(2)) and (not
svp(7)) and (not svp(0)))
    or (spk(7) and (not svp(1)) and (not svp(2)) and (not
svp(0)))
    or (spk(0) and (not svp(1)) and (not svp(2)))
    or (spk(1) and (not svp(2))));
    -----
    spr(4)<= (sz(4) and (not sz(0))and (not sz(1))and (not
sz(2))and (not sz(3)) and s0)
    or (svp(4) and ((spk(3))
    or (spk(4) and (not svp(0)) and (not svp(1)) and (not
svp(2)) and (not svp(3)) and (not svp(5)) and (not svp(6)) and
(not svp(7)))
    or (spk(5) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(6)) and (not svp(7)) and (not svp(0)))
    or (spk(6) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(7)) and (not svp(0)))
    or (spk(7) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(0)))
    or (spk(0) and (not svp(1)) and (not svp(2)) and (not
svp(3)))
    or (spk(1) and (not svp(3)) and (not svp(2)))
    or (spk(2) and (not svp(3))));
    -----
    spr(5)<= (sz(5) and (not sz(0))and (not sz(1))and (not
sz(2))and (not sz(3))and (not sz(4)) and s0)
    or (svp(5) and ((spk(4))
    or (spk(5) and (not svp(0)) and (not svp(1)) and (not
svp(2)) and (not svp(3)) and (not svp(4)) and (not svp(6)) and
(not svp(7)))
    or (spk(6) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(4)) and (not svp(7)) and (not svp(0)))
    or (spk(7) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(4)) and (not svp(0)))
    or (spk(0) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(4)))
    or (spk(1) and (not svp(4)) and (not svp(2)) and (not
svp(3)))
    or (spk(2) and (not svp(3)) and (not svp(4)))

```

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.06.001 ПЗ

Лист

71

```

    or (spk(3) and (not svp(4))));
    -----
    spr(6)<= (sz(6) and (not sz(0))and (not sz(1))and (not
sz(2))and (not sz(3))and (not sz(4))and (not sz(5)) and s0)
    or (svp(6) and ((spk(5))
    or (spk(6) and (not svp(0)) and (not svp(1)) and (not
svp(2)) and (not svp(3)) and (not svp(4)) and (not svp(5)) and
(not svp(7)))
    or (spk(7) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(4)) and (not svp(5)) and (not svp(0)))
    or (spk(0) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(4)) and (not svp(5)))
    or (spk(1) and (not svp(5)) and (not svp(2)) and (not
svp(3)) and (not svp(4)))
    or (spk(2) and (not svp(4)) and (not svp(5)) and (not
svp(3)))
    or (spk(3) and (not svp(5)) and (not svp(4)))
    or (spk(4) and (not svp(5))));
    -----
    spr(7)<= (sz(7) and (not sz(0))and (not sz(1))and (not
sz(2))and (not sz(3))and (not sz(4))and (not sz(5))and (not
sz(6)) and s0)
    or (svp(7) and ((spk(6))
    or (spk(7) and (not svp(0)) and (not svp(1)) and (not
svp(2)) and (not svp(3)) and (not svp(4)) and (not svp(5)) and
(not svp(6)))
    or (spk(0) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(4)) and (not svp(5)) and (not svp(6)))
    or (spk(1) and (not svp(6)) and (not svp(2)) and (not
svp(3)) and (not svp(4)) and (not svp(5)))
    or (spk(2) and (not svp(5)) and (not svp(6)) and (not
svp(3)) and (not svp(4)))
    or (spk(3) and (not svp(4)) and (not svp(5)) and (not
svp(6)))
    or (spk(4) and (not svp(5)) and (not svp(4)))
    or (spk(5) and (not svp(6))));
    end if;
    end process;
    process (CLK)

    variable i:integer;

    begin

    if CLK='1' and CLK'event then

    i := 0;
    while ( i<8) loop
    szo(i)<=sz(i);
    sro(i)<=sr(i);
    sjo(i)<=sj(i);

```

```

svpo(i)<=svp(i);
sko(i)<=sk(i);
spo(i)<=sp(i);
spko(i)<=spk(i);
svzo(i)<=svz(i);
sto<=st;
s0o<=s0;
smo<=ssm;
snko<=ssnk;
spro(i)<=spr(i);
i := i + 1;
end loop;
end if;
end process;
end Behavioral;

```

Код файла *Test Brench* :

```

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY tb1 IS
END tb1;

ARCHITECTURE behavior OF tb1 IS
  COMPONENT labVhdl
  PORT( clk : IN std_logic;
        x0 : IN std_logic;
        xz : IN std_logic_vector(7 downto 0);
        snk : IN std_logic_vector(7 downto 0);
        szo : OUT std_logic_vector(7 downto 0);
        sjo : OUT std_logic_vector(7 downto 0);
        svpo : OUT std_logic_vector(7 downto 0);
        sko : OUT std_logic_vector(7 downto 0);
        sro : OUT std_logic_vector(7 downto 0);
        spo : OUT std_logic_vector(7 downto 0);
        spko : OUT std_logic_vector(7 downto 0);
        svzo : OUT std_logic_vector(7 downto 0);
        spro : OUT std_logic_vector(7 downto 0);
        smo : OUT std_logic;
        snko : OUT std_logic;
        sto : OUT std_logic;
        s0o : OUT std_logic
        );
  END COMPONENT;

  --Inputs
  signal clk : std_logic := '0';
  signal x0 : std_logic := '0';
  signal xz : std_logic_vector(7 downto 0) := (others => '0');

```

```

    signal snk : std_logic_vector(7 downto 0) := (others =>
'0');
--Outputs
signal szo : std_logic_vector(7 downto 0);
signal sjo : std_logic_vector(7 downto 0);
signal svpo : std_logic_vector(7 downto 0);
signal sko : std_logic_vector(7 downto 0);
signal sro : std_logic_vector(7 downto 0);
signal spo : std_logic_vector(7 downto 0);
signal spko : std_logic_vector(7 downto 0);
signal svzo : std_logic_vector(7 downto 0);
signal spro : std_logic_vector(7 downto 0);
signal smo : std_logic;
signal snko : std_logic;
signal sto : std_logic;
signal s0o : std_logic;
-- Clock period definitions
constant clk_period : time := 10 ns;

BEGIN
    uut: labVhdl PORT MAP (snk => snk, s0o => s0o, smo => smo,
xz => xz, clk => clk, svzo => svzo, x0 => x0, spro => spro, sto
=> sto, snko => snko, szo => szo, sro => sro, sjo => sjo, svpo
=> svpo, sko => sko, spo => spo, spko => spko);

    -- Clock process definitions
    clk_process :process
    begin
        clk <= '0';
        wait for clk_period/2;
        clk <= '1';
        wait for clk_period/2;
    end process;

    -- Stimulus process
    stim_proc: process
    begin
        -- hold reset state for 100 ns.
        --while (true) loop
        wait for 100 ns;
        xz(0)<='1';
        xz(1)<='1';
        xz(2)<='1';
        xz(3)<='1';
        xz(4)<='1';
        xz(5)<='1';
        xz(6)<='1';
        xz(7)<='1';
        wait for 10 ns;
    end process;

```

```

xz(0)<='0';
xz(1)<='0';
xz(2)<='0';
xz(3)<='0';
xz(4)<='0';
xz(5)<='0';
xz(6)<='0';
xz(7)<='0';

wait for 500 ns;

xz(0)<='1';
xz(2)<='1';
xz(3)<='1';
xz(4)<='1';
xz(6)<='1';
wait for 10 ns;

xz(0)<='0';
xz(2)<='0';
xz(3)<='0';
xz(4)<='0';
xz(6)<='0';
wait for 300 ns;

xz(1)<='1';
xz(2)<='1';
xz(4)<='1';
xz(6)<='1';

wait for 10 ns;

xz(1)<='0';
xz(2)<='0';
xz(4)<='0';
xz(6)<='0';
wait for 250 ns;

xz(0)<='1';
xz(1)<='1';
xz(3)<='1';
xz(4)<='1';
xz(7)<='1';
wait for 10 ns;

xz(0)<='0';
xz(1)<='0';
xz(3)<='0';
xz(4)<='0';
xz(7)<='0';
wait for 300 ns;

```

```

xz(1)<='1';
xz(2)<='1';
xz(3)<='1';
xz(6)<='1';

wait for 10 ns;

xz(1)<='0';
xz(2)<='0';
xz(3)<='0';
xz(6)<='0';

wait for clk_period*10;

wait;
--end loop;
end process;

END;

```

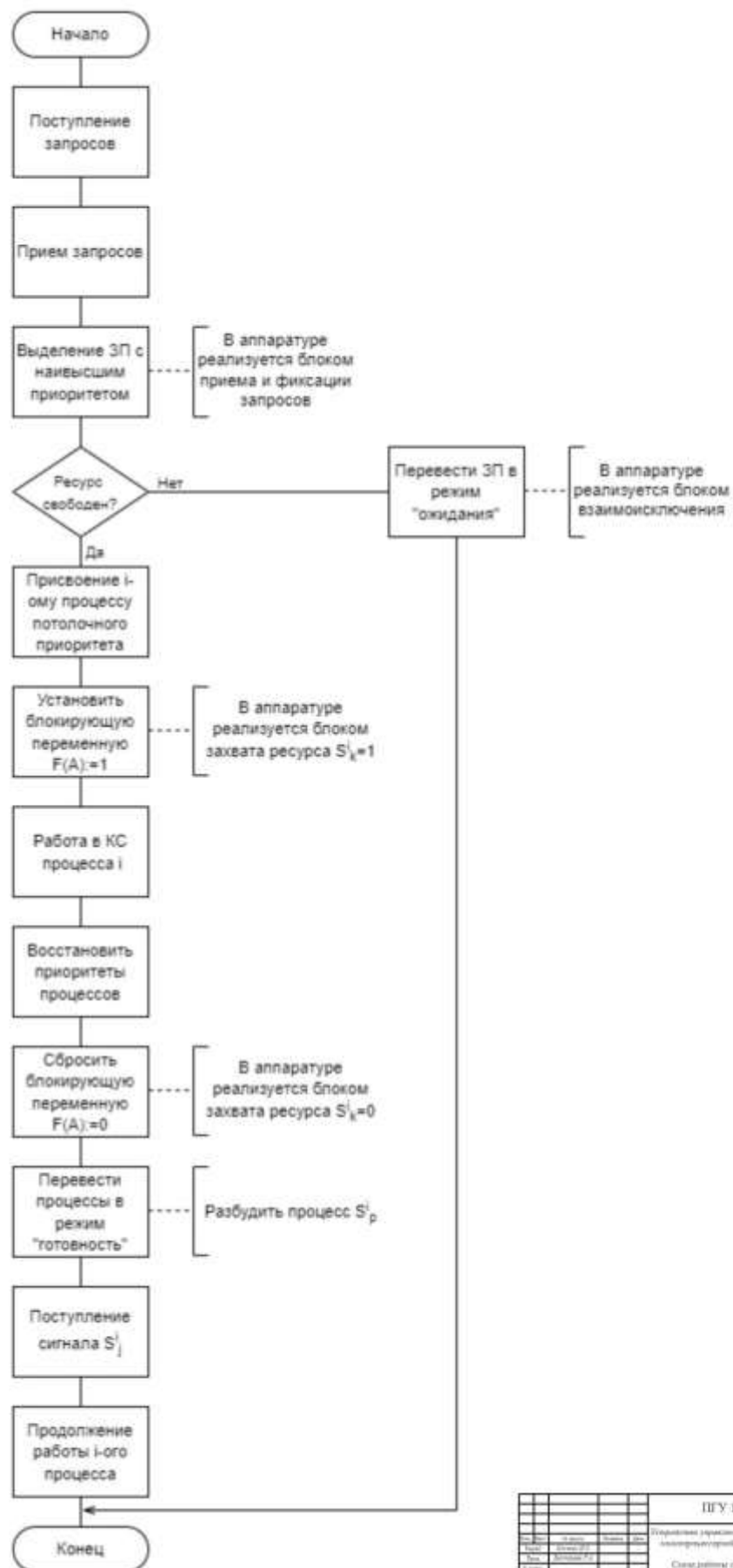
Приложение Б

(обязательное)

Устройство управления ресурсами многопроцессорной системы

Схема работы системы

					ПГУ 1.090501.06.001 ПЗ	Лист
						77
Изм.	Лист	№ докум.	Подпись	Дата		



ПГУ 1.090501.06.002 ДЗ									
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель
Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель	Исполнитель

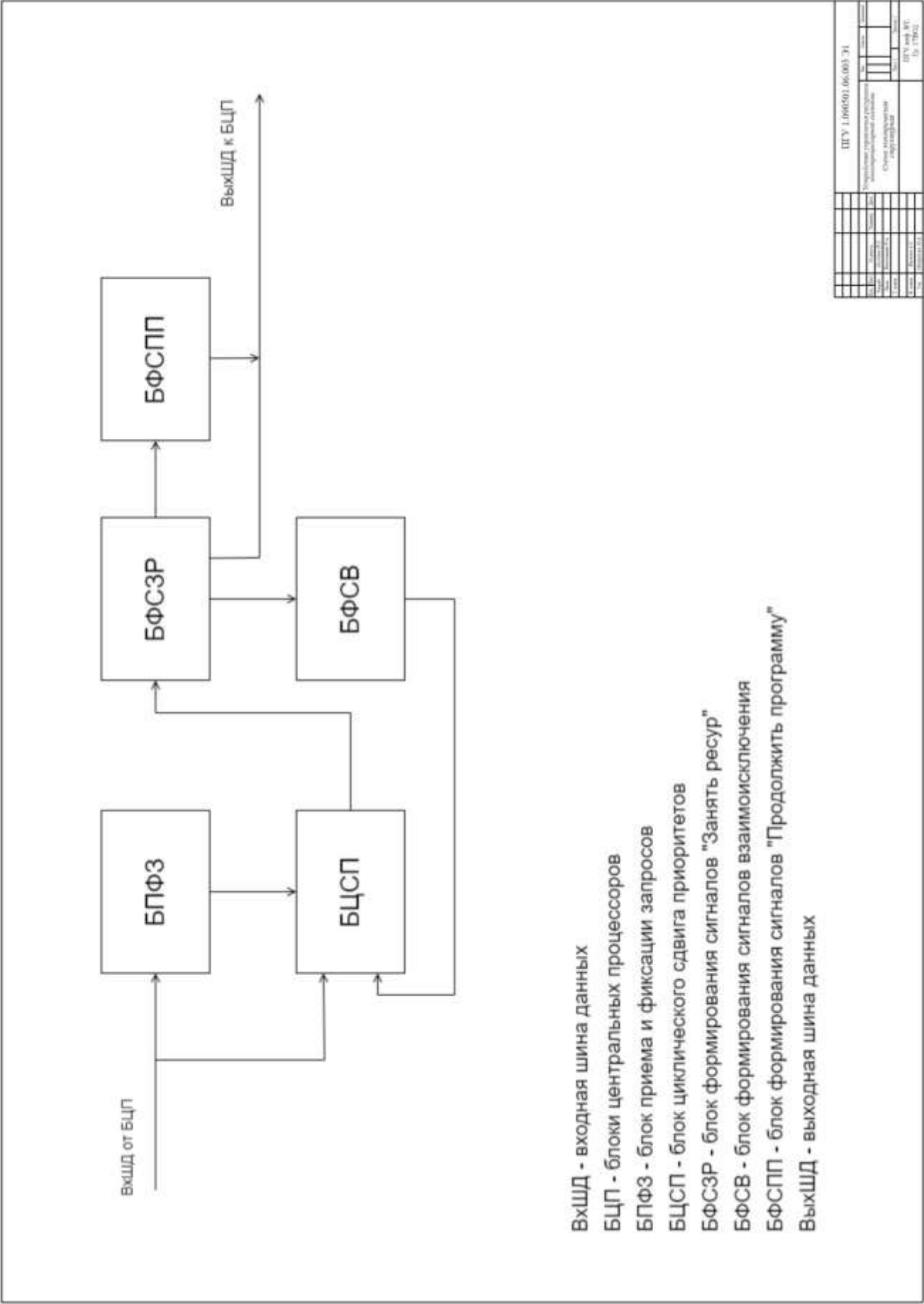
Приложение В

(обязательное)

Устройство управления ресурсами многопроцессорной системы

Схема электрическая структурная

					ПГУ 1.090501.06.001 ПЗ	Лист
						79
Изм.	Лист	№ докум.	Подпись	Дата		



ПГУ 1.090501.06.001.Э1									
№	Имя	Фамилия	И.О.	Дата	Подпись	И.О.	Дата	Подпись	И.О.
1	Иванов	Иван	Иванович	10.10.2020					
2	Петров	Петр	Петрович	10.10.2020					
3	Сидоров	Сидор	Сидорович	10.10.2020					
4	Смирнов	Смирнов	Смирнович	10.10.2020					
5	Климов	Климов	Климович	10.10.2020					
6	Васильев	Васильев	Васильевич	10.10.2020					
7	Попов	Попов	Попович	10.10.2020					
8	Морозов	Морозов	Морозович	10.10.2020					
9	Михайлов	Михайлов	Михайлович	10.10.2020					
10	Кузнецов	Кузнецов	Кузнецович	10.10.2020					

Изм.	Лист	№ докум.	Подпись	Дата

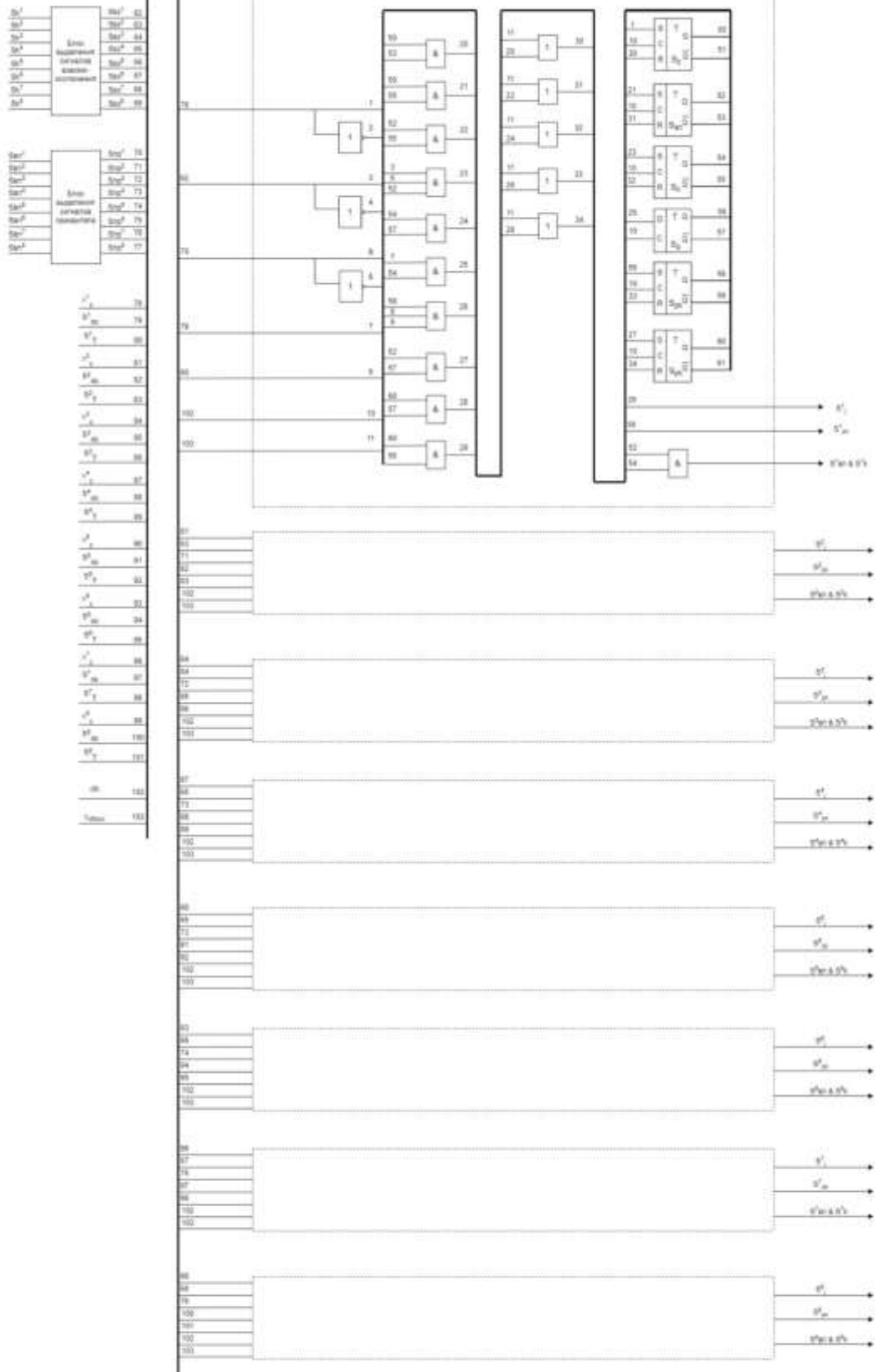
Приложение Г

(обязательное)

Устройство управления ресурсами многопроцессорной системы

Схема электрическая функциональная

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		81



ПГУ 1.090501.06.004 ПЗ									
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				
Изм.	Лист	№ докум.	Подпись	Дата	Содержание: программа управления газотурбинным двигателем				

Изм.	Лист	№ докум.	Подпись	Дата

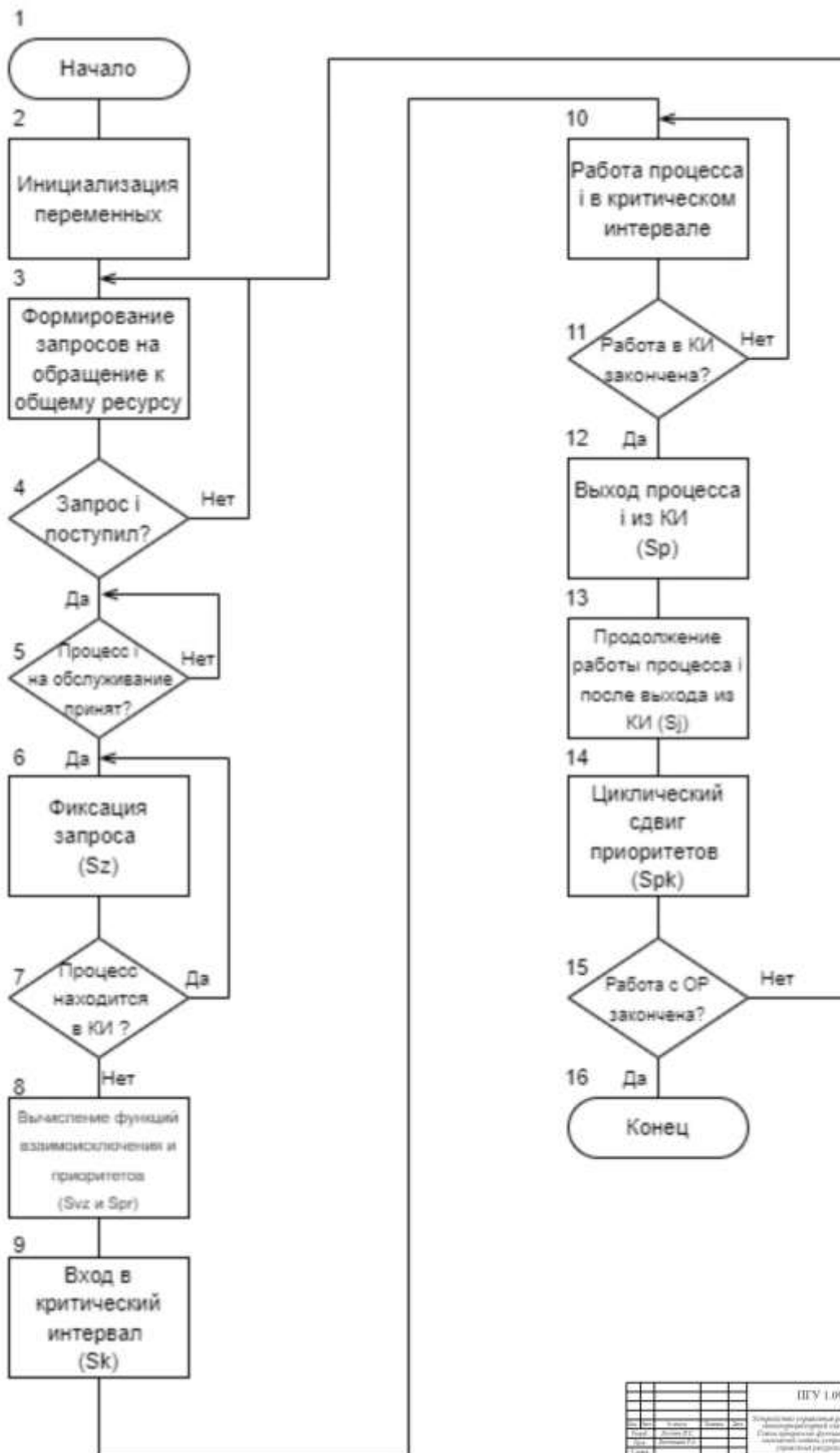
ПГУ 1.090501.06.001 ПЗ

Приложение Д

(обязательное)

Устройство управления ресурсами многопроцессорной системы
Схема программы функционально-логической модели устройства
управления ресурсами

					ПГУ 1.090501.06.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		83



ПГУ 1.090501.06.005 Д1				
Имя	Фамилия	Имя	Фамилия	Подпись
Имя	Фамилия	Имя	Фамилия	Подпись
Имя	Фамилия	Имя	Фамилия	Подпись
Имя	Фамилия	Имя	Фамилия	Подпись
Имя	Фамилия	Имя	Фамилия	Подпись
Имя	Фамилия	Имя	Фамилия	Подпись
Имя	Фамилия	Имя	Фамилия	Подпись
Имя	Фамилия	Имя	Фамилия	Подпись
Имя	Фамилия	Имя	Фамилия	Подпись

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.06.001 ПЗ

Приложение Е

(обязательное)

Устройство управления ресурсами многопроцессорной системы

Временная диаграмма работы устройства

					ПГУ 1.090501.06.001 ПЗ	Лист
						85
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение Ж

(обязательное)

Устройство управления ресурсами многопроцессорной системы

Презентация

					ПГУ 1.090501.06.001 ПЗ	Лист
						87
Изм.	Лист	№ докум.	Подпись	Дата		



ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ДИПЛОМНЫЙ ПРОЕКТ НА ТЕМУ:

«Устройство управления ресурсами многопроцессорной
системы»

Автор проекта: студент гр. 17BO2
Костин И.С.

Руководитель проекта: к.т.н., доцент
кафедры ВТ Бикташев Р.А.

Анализ необходимости разработки

Традиционное управление процессами и ресурсами, осуществляемое операционной системы на основе системного программного обеспечения, приводит к значительным замедлениям выполнения приложений, и, следовательно, к потерям производительности вычислительной системы.

Эта проблема вызвана наличием трудоемких компонент ядра ОС, к которым относятся: планировщики задач, связанные с формированием очереди задач, выделением и освобождением ресурсов; диспетчеры задач, связанные с переключением процессов; средства взаимодействия процессов, такие как каналы передачи сообщений; примитивы синхронизации.

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090501.06.001 ПЗ

Лист

88

Постановка задачи

- Цель работы – разработка аппаратного устройства управления ресурсами многопроцессорной системы

Устройство управления ресурсами многопроцессорной системы должно выполнять следующие условия:

- число процессоров (процессорных ядер) - до 8;
- число входов блока синхронизации - 8;
- чип запроса ресурса - приоритетный с циклической сменой приоритетов.

Устройство предназначено для управления параллельными взаимодействующими процессами. Оно обеспечивает доступ к общему ресурсу на основе критических интервалов.

Схема работы системы

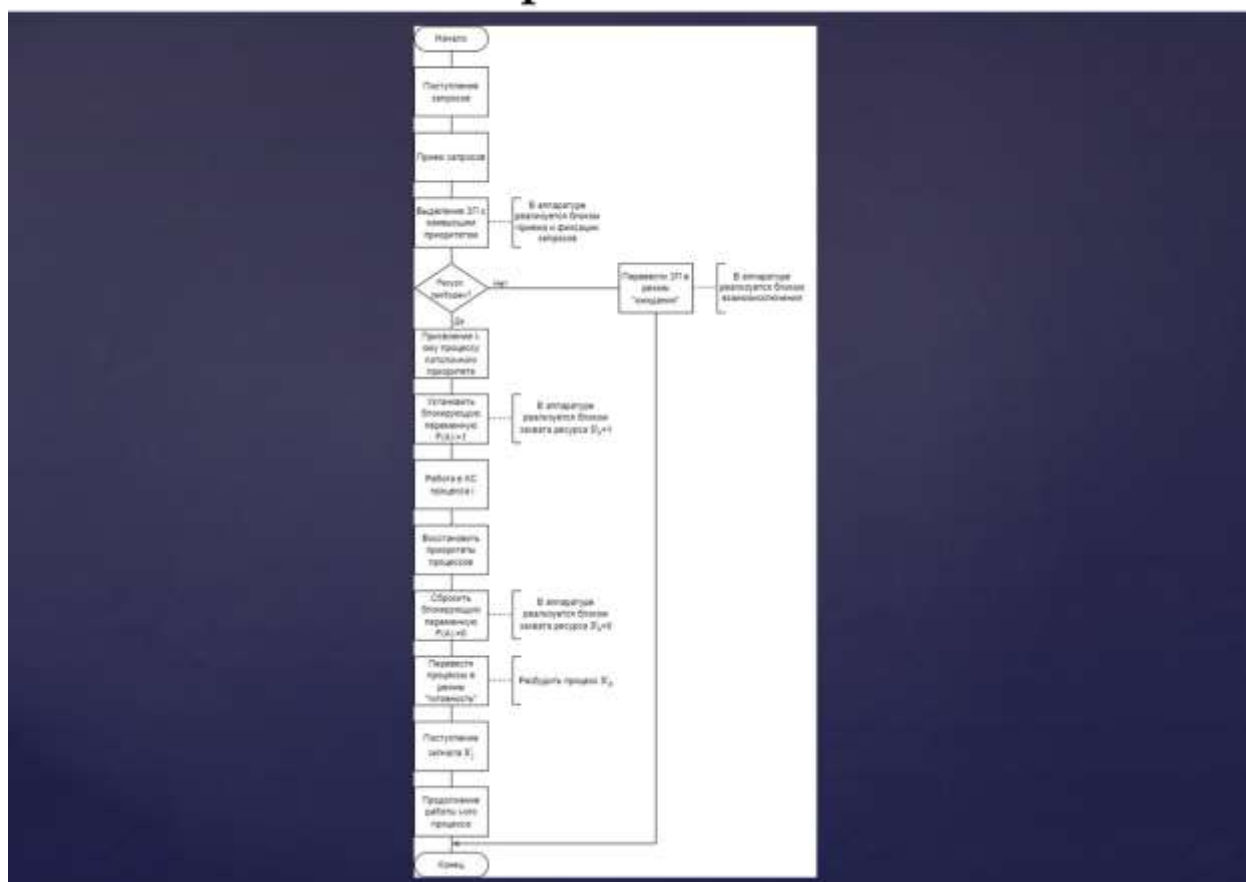


Схема подключения устройства управления ресурсами в многопроцессорную систему



Запрос к ресурсу направляется не в ядро ОС, где традиционно осуществляется синхронизация процессов, а перенаправляется к УУР, где и обрабатывается по алгоритму, реализующему механизм критических интервалов.

Интерфейс взаимодействия устройства управления ресурсами с процессором



Структура многопроцессорной системы с устройством синхронизации процессов

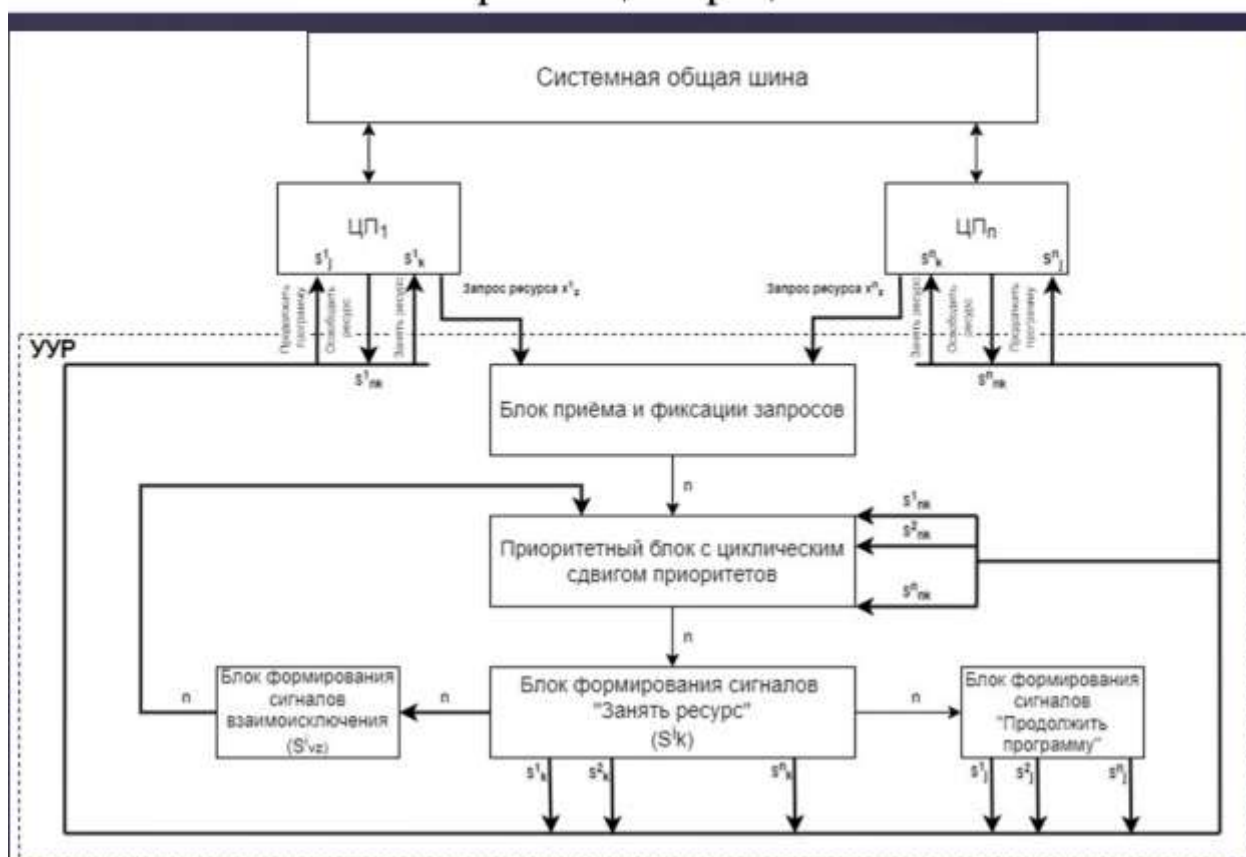
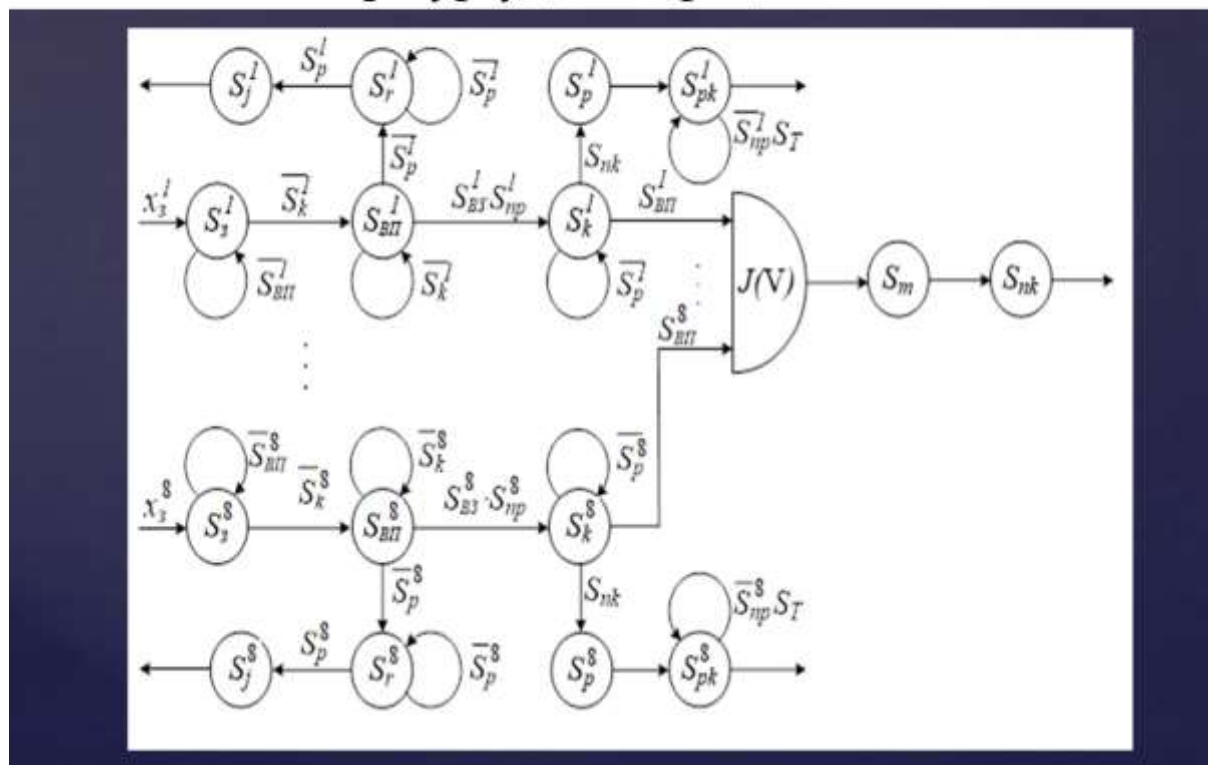


Схема электрическая структурная



Граф НДА алгоритма управления взаимодействующими параллельными процессами при обращении к общему ресурсу для 8-процессов



Функциональная схема устройства управления ресурсами

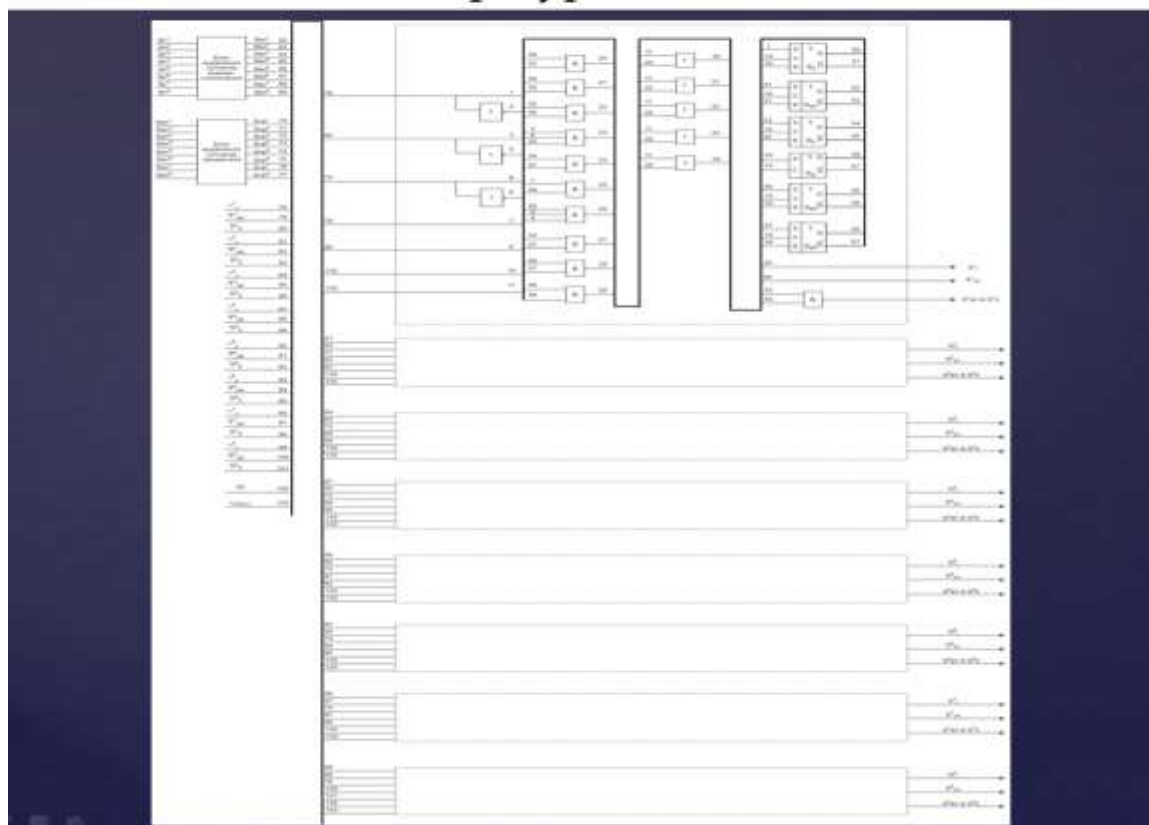


Схема модели устройства управления ресурсами для восьми процессов

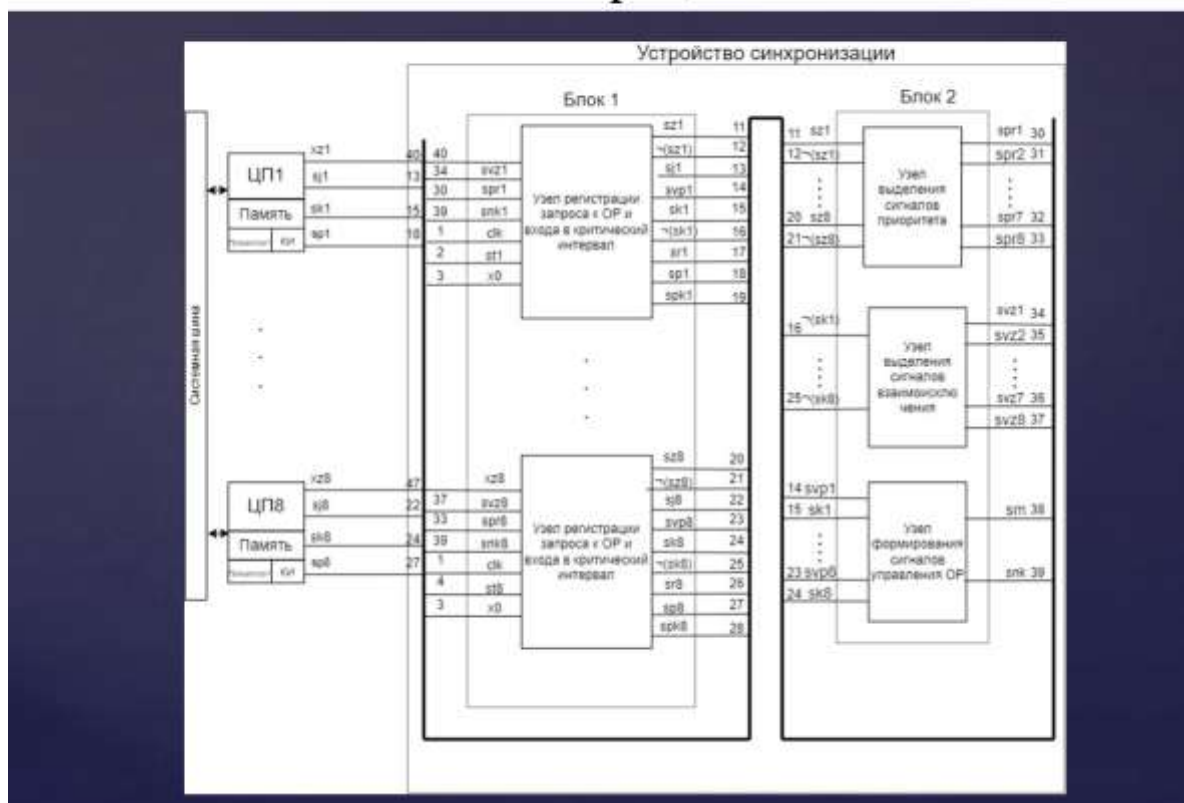
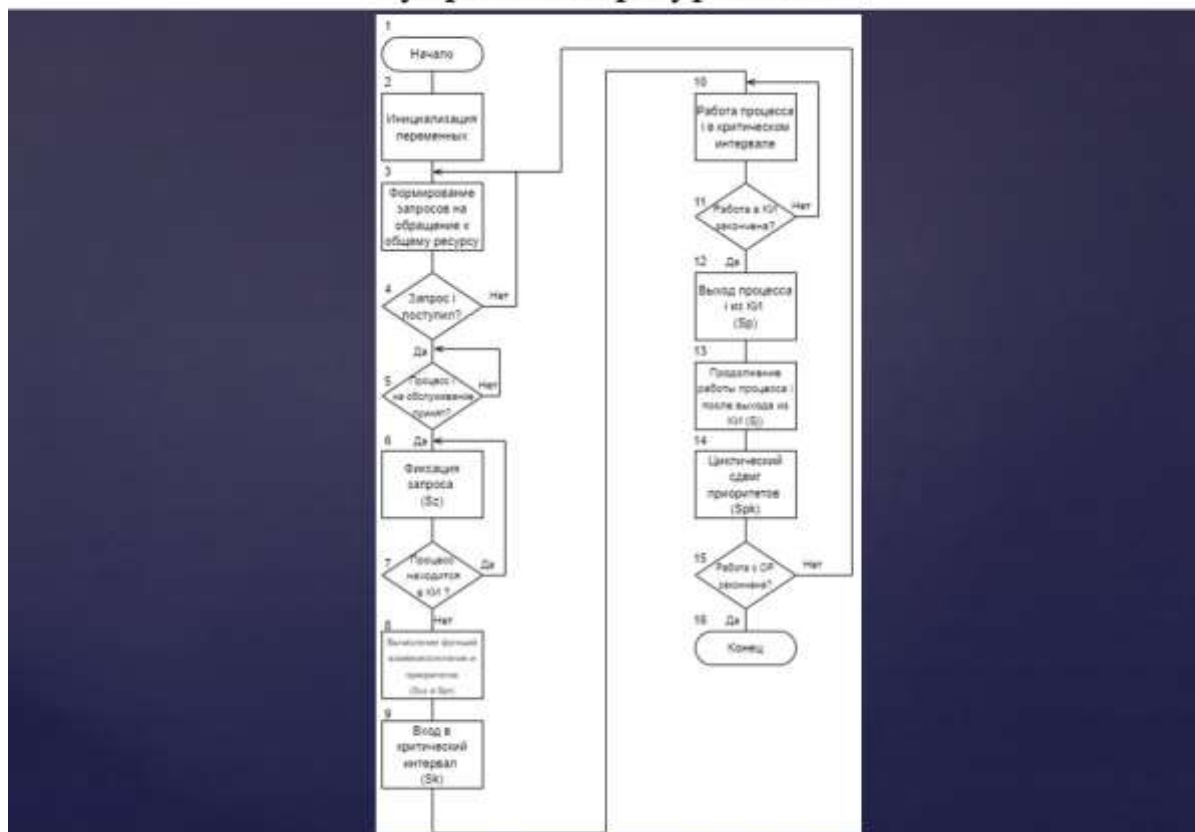
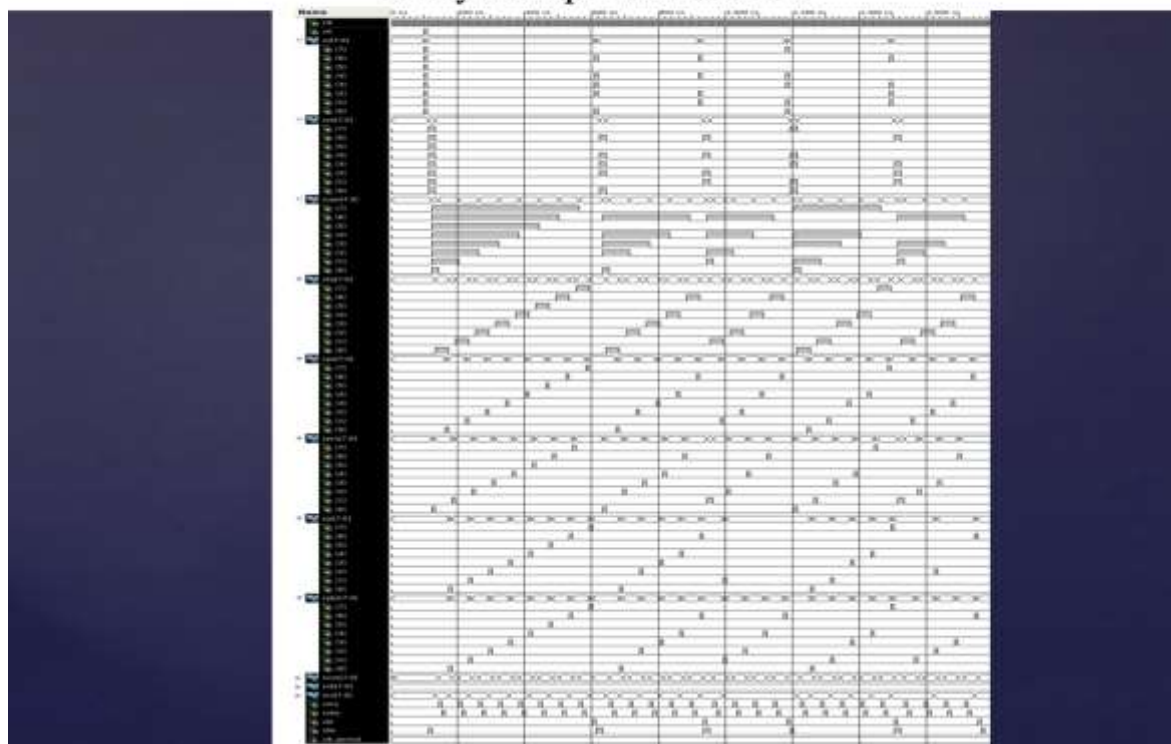


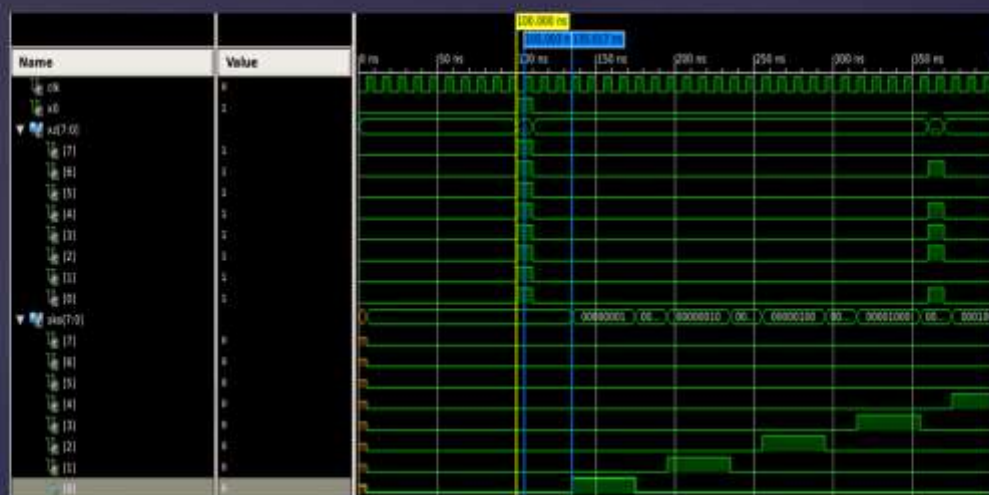
Схема программы функционально-логической модели устройства управления ресурсами



Результат моделирования разрабатываемого устройства на языке VHDL в виде временной диаграммы с использованием симулятора Xilinx ISE.

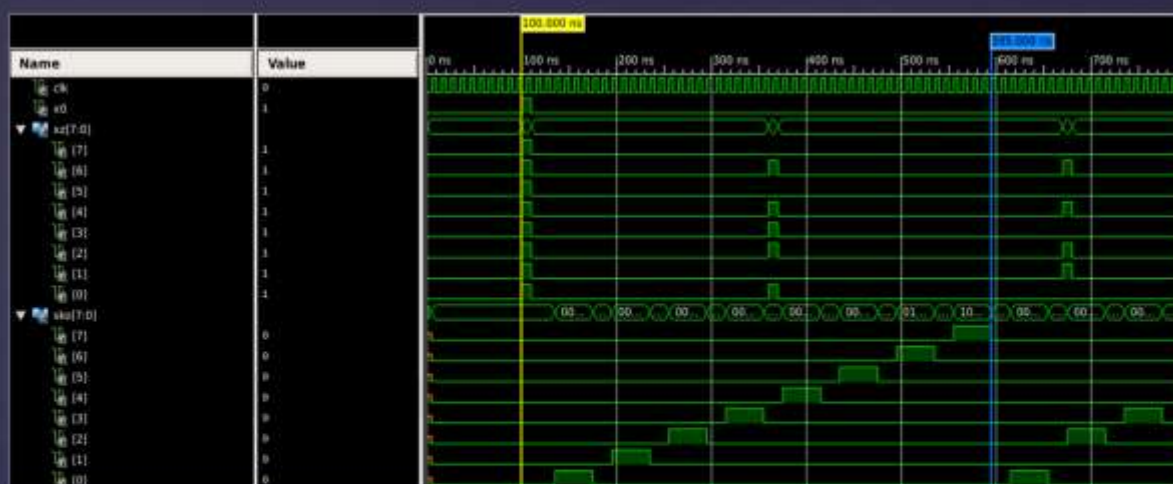


Показатели производительности работы устройства



Латентность (реакция на запрос) — задержка от момента появления запроса на доступ к общему ресурсу и до момента входа в критическую секцию в условиях бесконфликтности.

Показатели производительности работы устройства



Частота поступления запросов в условиях отсутствия насыщения устройства синхронизации процессов была оценена в количестве тактов между запросами, которая равна 1/50 в каждом цикле обслуживания.

СПАСИБО
ЗА ВНИМАНИЕ!