МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет	*Кафедра*
вычислительной техники	«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

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ МОДЕЛИРОВАНИЯ КЛАССИЧЕСКИХ АЛГОРИТМОВ УПРАВЛЕНИЯ ПРОЦЕССАМИ И РЕСУРСАМИ

Студент _____ 23.06.2025 Жидков Алексей Михайлович
(подпись, дата) (ФИО полностью)

Руководитель _____ 23.06.2025 Бикташев Р.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ 23.06.2025 Бычков А.С.
(подпись, дата) (фамилия, инициалы)

Рецензент доцент ПензГТУ _____ Печерский С.В.
(должность, место работы) (подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 16)

Заведующий кафедрой _____ Митрохин М.А.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой удовлетворительно
(протокол заседания ГЭК от 23 июня 2025 № 14)

Секретарь ГЭК _____ Гурин Е.И.
(подпись) (фамилия, инициалы)

Пенза 2025

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	Вычислительная техника

«Утверждаю»
Заведующий кафедрой
Митрохин М.А.
«6» 05 2025 г.

ЗАДАНИЕ
НА ДИПЛОМНЫЙ ПРОЕКТ

1. Студент Жидков Алексей Михайлович гр. 20ВОЭ1 факультета вычислительной техники.

2.Тема работы: «Программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами».

Тема утверждена приказом ПГУ № 601/0 от 6 . 05 .2025 г.

3. Руководитель работы к.т.н., доцент каф. ВТ «ПГУ» Бикташев Р.А.

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

Разработать программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами. Программный комплекс должен обеспечивать: моделирование алгоритма управления параллельными процессами при обращении к критическим ресурсам, моделирование алгоритма управления взаимодействующими параллельными процессами в задаче «производители-потребители», моделирование алгоритма управления взаимодействующими параллельными процессами в задаче «спящий парикмахер». Программный комплекс должен разрабатываться с помощью среды разработки Xilinx Vivado с формализацией задач в виде недетерминированных автоматов.

5. Перечень подлежащих разработке вопросов.

Требуется произвести анализ предметной области, определить функциональный состав программного комплекса, выбрать стек технологий для реализации программного комплекса, формализовать алгоритмы, что будут реализованы в программном комплексе, реализовать программный комплекс.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	10	06.05.2025 - 09.05.2025	
2.	Проектирование программного комплекса	10	10.05.2025 - 13.05.2025	
3.	Разработка программного комплекса	20	14.05.2025 - 18.05.2025	
4.	Отладка и тестирование	10	19.05.2025 - 22.05.2025	
5.	Разработка экономической и бжд части	10	23.05.2025 - 27.05.2025	
6.	Оформление результатов моделирования	10	28.05.2025	
7.	Оформление выпускной работы	30	29.05.2025 - 10.06.2025	

Дата выдачи задания 6 « 05 » 2025 года.

Руководитель дипломного проекта _____ 06.05.2025 Бикташев Р.А.
(подпись, дата)

Задание к исполнению принял студент _____ 06.05.2025 Жидков А.М.
(подпись, дата)

Работу к защите допустить
Декан факультета _____ 25.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Пояснительная записка состоит из 66 листов, 8 рисунков, 10 таблиц и 4 приложений.

ПРОГРАММНЫЙ КОМПЛЕКС, ПРАЛЛЕЛЬНЫЕ ПРОЦЕССЫ, ВЗАИМОДЕЙСТВУЮЩИЕ ПРОЦЕССЫ

Объектом исследования является процесс моделирования классических алгоритмов управления процессами и ресурсами.

Цель работы – разработка программного комплекса для моделирования классических алгоритмов управления процессами и ресурсами.

В процессе работы необходимо выполнить анализ предметной области, сформулировать техническое задание для разрабатываемого программного комплекса, формализовать и промоделировать соответствующие алгоритмы.

В результате проделанной работы был разработан программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами.

Программный комплекс разработан в среде разработки Xilinx Vivado.

Содержание

Перечень сокращений и обозначений 5
Введение 6
1. Анализ предметной области 9
1.1 Описание предметной области 9
1.2 Анализ аналогов 9
1.3 Цели и задачи разрабатываемого программного комплекса 11
1.4 Анализ целевой аудитории 12
2. Техническое задание 13
2.1 Назначение разработки 13
2.2 Требования к программе 13
2.2.1 Требования к функциональным характеристикам 13
2.2.2 Требования к надежности 13
2.2.3 Требования к составу и параметрам технических средств 13
2.3 Требования к программной документации 14
2.4 Стадии и этапы разработки 15
2.5 Порядок контроля и приемки 16
3. Обоснование целесообразности выбранных заданных средств разработки. . 17
3.1 Недетерминированные автоматы 17
3.2 Язык разработки VHDL 18
3.3 Среда разработки Xilinx Vivado 19
4. Разработка программного комплекса 21
4.1 Формализация алгоритма управления параллельными процессами при обращении к критическим ресурсам 25
4.2 Формализация алгоритма управления взаимодействующими параллельными процессами в задаче «производители-потребители» 34
4.3 Формализация алгоритма управления взаимодействующими параллельными процессами в задаче «спящий парикмахер» 43
4.4 Методика испытаний 46

5. Оценка затрат на разработку и внедрение программного комплекса.....48
5.1 Организация производства разработки программного комплекса.....50
5.2 Расчет себестоимости программного обеспечения для программного комплекса.....51
5.3 Анализ рисков при разработке и внедрении программного комплекса ...57
6. Общие принципы обеспечения безопасности на рабочем месте операторапрограммного комплекса.....59
6.1 Актуальность вопросов безопасности на рабочем месте 59
6.2 Требования безопасности при работе с электроприборами.....60
6.3 Влияние ПЭВМ на физиологию человека.....60
6.4 Техника безопасности при работе с персональным компьютером 61
Заключение 65
Список использованных источников 66
Приложение А – Графы НДА.....67
Приложение Б – VHDL-код.....72
Приложение В – Результаты испытаний.....102
Приложение Г – Презентация.....109

Перечень сокращений и обозначений

В настоящей работе применяют следующие сокращения и обозначения

НДА – недетерминированный автомат

СКУ – система канонических уравнений

СНДА – событийный НДА

ГСАП – граф-схема алгоритма с параллельными ветвями

ОР – общий ресурс;

КИ – критический интервал;

МРОТ – минимальный размер оплаты труда

ОС – операционная система

ПО – программное обеспечение

ПЭВМ – персональная электронно-вычислительная машина

ЦНС – центральная нервная система

ЭВМ – электронно-вычислительная машина

Введение

Современные системы управления процессами и ресурсами сталкиваются с множеством сложных вызовов, которые требуют внимательного и продуманного подхода к эффективному распределению ресурсов между параллельными процессами. В условиях, когда конкуренция за доступ к критически важным ресурсам становится все более актуальной и напряженной, оптимизация взаимодействия процессов приобретает особую значимость. Стремительный рост сложности вычислительных систем, а также увеличение числа параллельных процессов, с которыми приходится работать, подчеркивают необходимость создания мощных и надежных инструментов для моделирования и анализа алгоритмов управления. Эти инструменты становятся неотъемлемой частью разработки современных программных решений, поскольку они позволяют обеспечить надежность и эффективность работы программного обеспечения, что, в свою очередь, подтверждает значимость данной разработки.

Целью данной работы является разработка функционального программного комплекса, который будет обеспечивать моделирование классических алгоритмов управления процессами и ресурсами. Для достижения этой амбициозной цели необходимо решить несколько ключевых задач, каждая из которых играет важную роль в общем процессе. Во-первых, требуется тщательно изучить и формализовать классические алгоритмы управления, чтобы создать четкое и глубокое понимание их работы и принципов функционирования. Это позволит не только лучше осознать их внутреннюю логику, но и выявить возможные области для улучшения, что в дальнейшем может привести к созданию более эффективных и адаптивных решений.

Во-вторых, необходимо разработать модели недетерминированных автоматов для выбранных алгоритмов, что позволит более точно отразить их поведение в различных условиях и сценариях. Это, в свою очередь, создаст

основу для более глубокого анализа и тестирования алгоритмов, что является критически важным для понимания их эффективности и надежности в реальных условиях эксплуатации. В-третьих, следует реализовать код на языке VHDL, что обеспечит возможность практического применения разработанных моделей в реальных системах. Это позволит не только протестировать алгоритмы в теории, но и проверить их работоспособность на практике, что является важным этапом в процессе разработки.

Кроме того, создание тестовых стендов для проверки корректности работы алгоритмов является важным этапом, который позволит выявить и устранить возможные ошибки, а также подтвердить надежность и эффективность разработанных решений. Тестирование в реальных условиях поможет убедиться в том, что алгоритмы способны справляться с поставленными задачами и адаптироваться к изменяющимся условиям. Наконец, подготовка подробной документации станет необходимым шагом для обеспечения удобства использования и понимания разработанного программного комплекса, что значительно упростит его внедрение и эксплуатацию. Документация будет включать в себя как технические детали, так и рекомендации по использованию, что сделает продукт более доступным для конечных пользователей.

Выполнение всех этих задач позволит создать инструмент, способный эффективно моделировать и анализировать алгоритмы управления процессами. Это будет полезным как для разработчиков, стремящихся оптимизировать свои решения и повысить их эффективность, так и для исследователей, работающих в области вычислительных систем. Предоставляя им новые возможности для изучения и анализа алгоритмических подходов в управлении ресурсами, данный проект может значительно способствовать развитию данной области и улучшению существующих практик. В конечном итоге, успешная реализация данного программного комплекса может стать важным шагом на пути к созданию более совершенных и адаптивных систем

управления, способных эффективно справляться с вызовами современного мира.

1 Анализ предметной области

1.1 Описание предметной области

Программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами представляет собой специализированную среду, предназначенную для анализа, отладки и демонстрации работы базовых и расширенных моделей взаимодействия процессов с критическими ресурсами. Он ориентирован на поддержку таких широко известных задач, как «обращение к критическому ресурсу», «производители–потребители», «спящий парикмахер», которые являются фундаментом для построения синхронизированных многопоточных и многопроцессорных систем.

Актуальность разработки подобного программного комплекса обусловлена бурным ростом интереса к многозадачным вычислениям, высокопроизводительным параллельным системам и гетерогенным архитектурам, в которых корректное распределение ресурсов между конкурентными процессами играет ключевую роль. Моделирование алгоритмов управления позволяет не только обеспечить более глубокое понимание принципов взаимодействия параллельных задач, но и отрабатывать сценарии на уровне системной логики и аппаратных интерфейсов до этапа их реальной реализации.

Таким образом, программный комплекс служит связующим звеном между теоретическим уровнем изучения алгоритмов синхронизации и их практической реализацией на уровне системного и аппаратного проектирования.

1.2 Анализ аналогов

Для обоснования необходимости создания разрабатываемого программного комплекса был проведён анализ существующих решений в

смежной области. Рассмотрены как академические, так и индустриальные средства, поддерживающие моделирование алгоритмов управления ресурсами:

– специализированные симуляторы операционных систем и планировщиков. Например, визуальные симуляторы алгоритмов планирования CPU (такие как Round Robin, Priority Scheduling, Multilevel Queues), используемые в рамках учебных курсов по операционным системам. Преимущества: высокая наглядность, простота в использовании, ориентированность на образовательный процесс. Недостатки: ограничены исключительно логикой планирования; не моделируют аппаратные конфликты, не поддерживают аппаратную реализацию; не интегрируются с языками описания аппаратуры (HDL);

– инструменты анализа производительности и графов зависимостей. Например, CPAT (Critical Path Analysis Tool) и аналогичные средства анализа. Преимущества: детектирование узких мест, построение диаграмм зависимостей и временных графов. Недостатки: не фокусируются на логике алгоритмов синхронизации; предназначены скорее для оценки готовых систем, чем для их проектирования или тестирования;

– академические симуляторы общего назначения. Включают такие проекты, как OS Simulator, Process Scheduler Visualizer, реализованные на Java, Python и других языках. Преимущества: открытый код, кроссплатформенность, богатый набор встроенных алгоритмов (FCFS, SJF, MLFQ и др.). Недостатки: ориентированы на визуализацию, не поддерживают генерацию или симуляцию VHDL/Verilog; отсутствие синхронизации с реальными архитектурами;

– коммерческие среды моделирования, например MATLAB/Simulink. В частности, блоки Simulink Real-Time, специализированные библиотеки моделирования распределённых систем. Преимущества: высокая гибкость, наличие готовых модулей, возможность генерации исполнимого кода. Недостатки: высокая стоимость лицензий, значительные системные

требования, переизбыточность для задач моделирования базовых алгоритмов синхронизации.

Вывод: существующие решения либо слишком общие, либо избыточные, либо не охватывают задачи синхронизации в их недетерминированном аппаратно-ориентированном аспекте. Таким образом, разработка собственного программного комплекса для моделирования классических алгоритмов взаимодействия процессов и ресурсов представляется обоснованной и своевременной.

1.3 Цели и задачи разрабатываемого программного комплекса

Целью разработки является создание специализированного программного комплекса, позволяющего моделировать, исследовать и отлаживать классические алгоритмы управления параллельными процессами и критическими ресурсами с возможностью дальнейшего перехода к их аппаратной реализации на языке VHDL.

Для достижения поставленной цели необходимо решить следующие задачи:

– реализовать поддержку классических алгоритмов синхронизации, таких как: “взаимное исключение”, «производители–потребители», «спящий парикмахер»;

– обеспечить возможность представления алгоритмов в виде систем канонических уравнений (СКУ), совместимых с теорией недетерминированных автоматов;

– создать модули тестирования (testbench), позволяющие воспроизводить различные сценарии взаимодействия процессов и проверять корректность функционирования алгоритмов;

– минимизировать вероятность ошибок в алгоритмах синхронизации за счёт формальной модели и автоматизации.

1.4 Анализ целевой аудитории

Исходя из характера и назначения разрабатываемого программного комплекса, можно выделить несколько основных групп потенциальных пользователей:

– исследователи и аспиранты, занимающиеся проблемами параллелизма, синхронизации и оптимизации вычислений. Им программный комплекс будет полезен как инструмент для апробации новых алгоритмов и тестирования гипотез;

– разработчики встраиваемых систем и аппаратных решений, которым необходимо реализовывать корректные схемы взаимодействия между модулями на уровне логики;

– студенты технических направлений, изучающие курсы по архитектуре компьютеров, операционным системам, моделированию и системному программированию. Комплекс может служить как учебный стенд, так и инструмент самостоятельного проектирования;

– преподаватели, заинтересованные в демонстрации классических задач синхронизации и поддержке лабораторных работ по курсам архитектуры вычислительных систем, ПЛИС и операционных систем.

Таким образом, программный комплекс ориентирован на достаточно широкий круг специалистов, работающих в области проектирования и анализа параллельных систем, и может применяться как в научных, так и в образовательных целях.

2 Техническое задание

2.1 Назначение разработки

Программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами предназначен для моделирования классических алгоритмов управления процессами и ресурсами.

Цель данного проекта – разработка моделей классических алгоритмов на языке VHDL.

2.2 Требования к программному комплексу

2.2.1 Требования к функциональным характеристикам

Программный комплекс должен выполнять следующие функции:

– моделирование алгоритма управления параллельными процессами при обращении к критическим ресурсам;

– моделирование алгоритма управления взаимодействующими праллельвми процессами в задаче «производители-потребители»;

– моделирование алгоритма управления взаимодействующими параллельными процессами в задаче «спящий парикмахер».

2.2.2 Требования к надежности

Программный комплекс должен корректно моделировать алгоритмы управления взаимодействующими процессами в соответствующих задачах. Корректно разрешать конфликтные ситуации, возникающие между синхронизируемыми им параллельными процессами.

2.2.3 Требования к составу и параметрам технических средств

Программный комплекс должен быть предназначен для использования на локальном компьютере. Рабочие станции должны иметь минимальные технические характеристики:

– операционная система Windows/Linux;

– процессор Intel Core 1,9 ГГц и выше;

– свободное место на жестком диске 2 ГБ и выше;

– оперативную память 2 ГБ и выше;

– монитор с диагональю не менее 16 дюймов;

– установленную среду разработки и моделирования Xilinx Vivado;

– клавиатура;

– мышь.

2.3 Требования к программной документации

Программный комплекс должен сопровождаться документацией, содержащей:

– техническое задание;

– описание формализации соответствующих алгоритмов;

– описание моделирования соответствующих алгоритмов;

– описание программного комплекса;

– программу и методику испытаний;

– описание применения.

Структура и содержание программной документации должны соседствовать требованиям стандартов Единой системы программной документации (ЕСПД).

2.4 Стадии и этапы разработки

Этапы разработки программного комплекса представлены в таблице 1.

Таблица 1 – Стадии и этапы разработки

№	Стадии	Этапы	Объем работы в процентах	Срок выполнения	Исполнитель
1	Анализ предметной области	Анализ предметной области	10		Жидков А.М.
2	Проектирование программного комплекса	Формализация соответствующих алгоритмов	10		Жидков А.М.
3	Разработка программного комплекса	Разработка программного комплекса (моделирование алгоритмов на VHDL)	20		Жидков А.М.
4	Отладка и тестирование	Отладка и тестирование	10		Жидков А.М.
5	Разработка экономической части	Оценка затрат на разработку и внедрение программного комплекса	5		Жидков А.М.
6	Разработка раздела БЖД	Оценка тяжести труда эксплуататора программного приложения	5		Жидков А.М.

№	Стадии	Этапы	Объем работы в процентах	Срок выполнения	Исполнитель
7	Оформление графической части	Оформление графической части	10		Жидков А.М.
8	Оформление Выпускной работы	Оформление пояснительной записки	30		Жидков А.М.

2.5 Порядок контроля и приемки

Для подтверждения работоспособности системы и соответствия её функционала техническому заданию следует составить наборы тестовых данных, охватывающие все возможные сценарии взаимодействия параллельных процессов. Каждый алгоритм управления моделируется на этих тестовых наборах: система считается соответствующей требованиям, если при любых заданных входных комбинациях она корректно выполняет предусмотренные операции.

Приёмка системы проводится в условиях её нормального функционирования в присутствии представителей заказчика. В ходе приёмных испытаний проверяется выполнение всех заявленных функций, а также наличие полной сопровождающей документации – пояснительной записки, исходного кода программы и отчёта о результатах прогонных примеров.

3 Обоснование целесообразности использования выбранных средств разработки

Исходя из заданных функциональных требований к комплексному средству моделирования классических алгоритмов управления процессами и ресурсами, была выполнена оценка и отобраны наиболее подходящие инструменты разработки.

3.1 Недетерминированные автоматы

Недетерминированные автоматы (НДА) представляют собой математическую модель, предназначенную для описания систем, где для одного состояния и входного символа возможно несколько исходов. Этот концепт был впервые предложен в 1956 году Майклом Рабином и Даной Скоттом в их работе 'Finite Automata and Their Decision Problems'. Основное отличие НДА от детерминированных автоматов (ДА) заключается в возможности наличия множества переходов для одной и той же пары состояния и символа. Это свойство делает НДА полезным инструментом для моделирования процессов, в которых присутствует неопределённость или параллельное выполнение. Важно подчеркнуть, что, несмотря на кажущуюся сложность, НДА и ДА эквивалентны по вычислительной мощности. Любой НДА может быть преобразован в ДА, что было доказано в теории вычислений.

Недетерминированные автоматы находят широкое применение в различных областях моделирования и анализа. Например, в разработке компиляторов они используются для описания грамматик языков программирования, что позволяет эффективно обрабатывать синтаксические конструкции. В системах управления НДА применяются для моделирования сложных процессов, таких как управление ресурсами в распределённых системах, где важно учитывать множество возможных сценариев. Кроме того, НДА играют ключевую роль в анализе и проверке моделей программного обеспечения, особенно в методах анализа конечных состояний, обеспечивая

возможность выявления ошибок и оптимизации процессов. Таким образом, НДА являются важным инструментом в моделировании и анализе современных систем.

3.2 Язык разработки VHDL

Язык VHDL (VHSIC Hardware Description Language) был создан в 1980-х годах в рамках программы VHSIC (Very High-Speed Integrated Circuits), инициированной Министерством обороны США. Основная цель разработки языка заключалась в предоставлении унифицированного средства для описания и проектирования сложных цифровых систем. Одной из ключевых особенностей VHDL является его способность описывать поведение цифровых схем на различных уровнях абстракции, начиная от логического уровня и заканчивая системным. Это делает VHDL универсальным инструментом для разработки и моделирования цифровых устройств, включая процессоры, контроллеры и специализированные интегральные схемы. Благодаря своей стандартизации и гибкости, язык VHDL широко используется в различных областях электроники и автоматизации.

Язык VHDL нашел широкое применение в проектировании цифровых систем благодаря своей способности эффективно описывать параллелизм и сложные алгоритмы управления. Это особенно важно при разработке систем управления процессами, где требуется моделирование взаимодействия множества компонентов в реальном времени. VHDL позволяет создавать точные модели работы алгоритмов, что способствует выявлению и устранению ошибок еще на этапе проектирования. Кроме того, язык поддерживает симуляцию и верификацию моделей, что делает его незаменимым инструментом для разработки надежных систем управления. Таким образом, использование VHDL в моделировании алгоритмов управления позволяет значительно сократить время и затраты на разработку сложных цифровых устройств.

3.3 Среда разработки Xilinx Vivado

Xilinx Vivado HL WebPack 2019.1 – это среда разработки для проектирования цифровых систем на базе FPGA и SoC от компании Xilinx. Vivado предлагает широкий набор инструментов для проектирования, синтеза, реализации и отладки аппаратных решений.

Основные особенности Vivado HL WebPack 2019.1:

– интуитивно понятный интерфейс: Vivado предоставляет удобный графический интерфейс, который облегчает навигацию по проектам и доступ к различным инструментам;

– поддержка языков описания аппаратуры: Vivado поддерживает VHDL, Verilog и SystemVerilog, что позволяет разработчикам использовать знакомые языки для описания своих проектов;

– инструменты для синтеза и реализации: Vivado включает в себя мощные инструменты для синтеза, которые оптимизируют проект для достижения высокой производительности и низкого потребления энергии;

– интеграция с IP-ядрами: Vivado предоставляет доступ к библиотеке IP-ядер, которые можно использовать для ускорения разработки и добавления функциональности в проекты;

– поддержка RTL-симуляции: Vivado HL WebPack 2019.1 включает в себя возможности для RTL-симуляции, что позволяет разработчикам проверять корректность работы их проектируемых систем на уровне регистровой передачи. Это достигается с помощью интеграции с такими инструментами, как XSIM, который позволяет выполнять симуляцию и отладку проектируемых систем;

– инструменты для анализа и отладки: Vivado предлагает различные инструменты для анализа производительности и отладки, включая встроенные средства для анализа временных характеристик и логического анализа;

– поддержка разработки на SoC: Vivado также поддерживает проектирование систем на кристалле (SoC), что позволяет интегрировать

программируемые логические элементы с процессорами и другими компонентами.

Vivado HL WebPack 2019.1 является мощным инструментом для разработчиков, работающих с FPGA и SoC, обеспечивая все необходимые средства для успешного проектирования и верификации цифровых систем.

4 Разработка программного комплекса

Формализация алгоритмов управления процессами представляет собой ключевой и неотъемлемый элемент в разработке программного обеспечения, особенно в контексте систем реального времени, где требования к производительности и надежности играют критическую роль. Этот процесс включает в себя тщательное определение и описание основных параметров, а также ограничений, которые необходимо учитывать при проектировании системы. Например, при разработке таких систем крайне важно учитывать временные ограничения, связанные с выполнением различных операций, а также требования к надежности и безопасности функционирования системы. Даже малейшие отклонения от заданных параметров могут привести к серьезным последствиям, включая сбои в работе и потенциальные угрозы для безопасности пользователей.

Исследования в области программной инженерии подтверждают, что недостаточная формализация требований на этапе проектирования является одной из основных причин, приводящих к возникновению ошибок в программных системах. По данным компании IBM, около 40% всех ошибок в программных системах возникают именно из-за недостаточной четкости и полноты требований. Это подчеркивает важность этого этапа разработки, так как четкое и детализированное определение требований на начальном этапе разработки не только помогает минимизировать риски, связанные с возможными ошибками, но и значительно улучшает качество конечного продукта. Это, в свою очередь, может привести к повышению конкурентоспособности на рынке, что является важным фактором для любой компании, стремящейся к успеху.

Кроме того, формализация требований способствует более эффективному взаимодействию между членами команды разработки. Когда все участники проекта имеют общее понимание задач и целей, это создает условия для более слаженной работы и уменьшает вероятность

недоразумений. Такое взаимодействие не только повышает удовлетворенность пользователей, но и снижает затраты на последующее исправление ошибок, что делает процесс разработки более эффективным и экономически целесообразным. В конечном итоге, качественная формализация требований и алгоритмов управления процессами не только улучшает технические характеристики системы, но и создает основу для ее успешной эксплуатации в реальных условиях, обеспечивая тем самым долгосрочную стабильность и надежность работы программного обеспечения.

Таким образом, можно утверждать, что формализация алгоритмов управления процессами является не просто технической необходимостью, а стратегическим шагом, который определяет успех всего проекта и его соответствие современным требованиям и стандартам. Это требует от команды разработчиков не только глубокого понимания предметной области, но и способности к системному мышлению, что позволяет учитывать все аспекты функционирования системы. В условиях быстро меняющегося технологического ландшафта, где инновации происходят с невероятной скоростью, способность к эффективной формализации и адаптации алгоритмов управления процессами становится ключевым фактором, способствующим успешной реализации проектов и удовлетворению потребностей конечных пользователей.

Важно отметить, что такая формализация не только улучшает качество программного обеспечения, но и создает условия для его дальнейшего развития и масштабирования, что является необходимым в условиях постоянных изменений и требований рынка. В конечном счете, успешная формализация алгоритмов управления процессами позволяет не только избежать множества проблем на этапе эксплуатации, но и открывает новые горизонты для внедрения инновационных решений, что в свою очередь способствует созданию более устойчивых и адаптивных систем, способных эффективно реагировать на вызовы современного мира.

Таким образом, формализация алгоритмов управления процессами становится неотъемлемой частью стратегического планирования и разработки программного обеспечения, позволяя компаниям не только соответствовать текущим стандартам, но и предвосхищать будущие требования. Это в конечном итоге ведет к созданию более качественных и надежных продуктов, которые могут успешно конкурировать на рынке. Данный процесс также подчеркивает необходимость постоянного обучения и повышения квалификации специалистов в области программной инженерии, чтобы они могли эффективно справляться с новыми вызовами и использовать современные инструменты и методологии для достижения наилучших результатов в своей работе. В условиях стремительного развития технологий и изменения потребностей пользователей, способность к адаптации и внедрению новых подходов становится важнейшим аспектом для достижения успеха в данной области.

Таким образом, формализация алгоритмов управления процессами не только служит основой для создания высококачественного программного обеспечения, но и формирует стратегическую платформу для будущих инноваций и улучшений, что в конечном итоге способствует устойчивому развитию и процветанию компаний в условиях динамичного и конкурентного рынка. Это подчеркивает важность интеграции современных методологий и практик в процесс разработки, что позволяет не только оптимизировать текущие процессы, но и закладывать фундамент для будущих достижений и успехов в области программной инженерии.

Формализация задаёт единую основу для точного описания и последующей реализации любых сложных алгоритмов. Когда все ключевые понятия, состояния и переходы оформлены в виде строгих математических моделей или канонических уравнений, исчезают двусмысленности, свойственные простым словесным описаниям. Это позволяет сразу же выявить противоречия или упущения ещё на этапе проектирования, а не в процессе дорогостоящей интеграции или в поздней фазе тестирования. В результате

существенно снижается риск появления трудноуловимых «гонок» или логических разрывах, которые способны поставить под угрозу всю корректность работы системы.

Ещё одним важным эффектом формализации является возможность автоматизированной верификации и символического анализа. Инструменты, поддерживающие проверку моделей, могут с высокой степенью уверенности подтвердить отсутствие тупиковых состояний или бесконечных циклов, выявить недостижимые ветви логики, а иногда — предложить оптимальные преобразования для упрощения структуры. Без формальных спецификаций подобных проверок просто не провести, и именно эта автоматизация становится залогом высокого качества, особенно в системах, где надёжность критична.

Наконец, формальные описания легко документируются и служат великолепным средством передачи знаний между командами и поколениями разработчиков. В отличие от фрагментарных комментариев в коде или устных инструкций, единый формальный текст остаётся понятным и неизменным, даже когда технология и сама среда разработки устаревают. Это обеспечивает преемственность и позволяет новым участникам вникнуть в суть проекта без длительного «разбирательства в чужом коде». В итоге формализация выступает не только как инструмент надёжности, но и как своего рода «живая документация», сопутствующая любому значимому программному продукту.

В заключение, можно сказать, что формализация алгоритмов управления процессами является неотъемлемой частью успешной разработки программного обеспечения, которая требует от специалистов не только технических знаний, но и стратегического мышления. Это позволяет не только создавать качественные и надежные системы, но и адаптироваться к быстро меняющимся условиям рынка, что в свою очередь способствует долгосрочному успеху и устойчивому развитию организаций в условиях глобальной конкуренции.

4.1 Формализация алгоритма управления параллельными процессами при обращении к критическим ресурсам

Устройство управления ресурсом (УУР), или модуль синхронизации, работает по следующей схеме. За каждым разделяемым ресурсом закреплён отдельный УУР, который принимает от конкурирующих процессов (или потоков) входные запросы на доступ к этому ресурсу (см. рисунок 1). Когда процесс получает доступ, он входит в так называемый критический интервал – участок кода, в котором производится работа с общим ресурсом.

При поступлении запроса УУР проверяет, свободен ли ресурс. Если да, оно сразу формирует внутренний сигнал «запрос принят» и разрешает этому процессу начать работу. В случае одновременной подачи нескольких запросов (рисунок 2), УУР отдаёт предпочтение процессу с наибольшим приоритетом, активируя у него сигнал «вход в критический интервал». Остальным заявителям высылается сигнал взаимоисключения, и они остаются в ожидании, пока ресурс не освободится.

Когда работа в критической секции завершается, процесс посылает сигнал «выход из КИ». УУР, получив его, сбрасывает запрет для следующего ожидающего процесса и выдаёт ему управляющую команду «продолжить работу», после чего следующий процесс может войти в свой критический интервал.

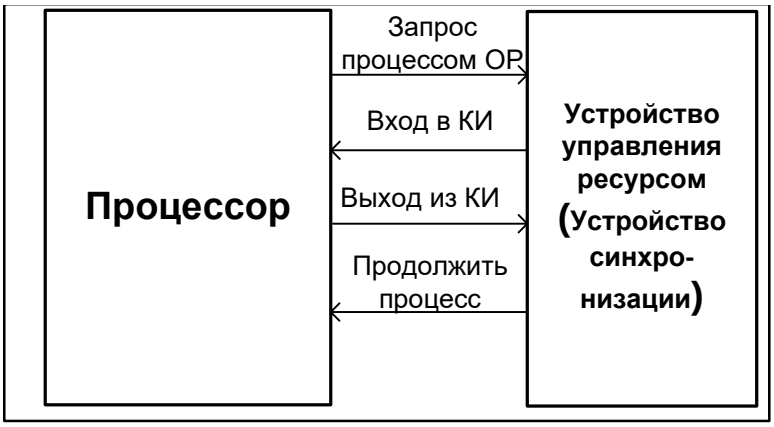


Рисунок 1. Интерфейс взаимодействия УУР с процессором

На рисунке 2 показано, как модуль синхронизации обменивается сигналами с процессами, запущенными в различных центральных процессорах. Принцип работы здесь ничем не отличается от классического подхода. Предположим, что в $ЦП_1$, $ЦП_2$, …, $ЦП_i$ выполняются i-е процессы, и все они запрашивают доступ к единому разделяемому ресурсу. Каждый процесс обрабатывает этот ресурс лишь на своём участке кода, называемом критическим интервалом. Для взаимодействия с модулем синхронизации в каждом процессоре предусмотрены специальные аппаратные выводы (в универсальных архитектурах их может и не быть). Когда процессу требуется войти в критическую секцию, он формирует на своём выводе аппаратный сигнал «Запрос ОР», который поступает непосредственно на вход синхронизатора.

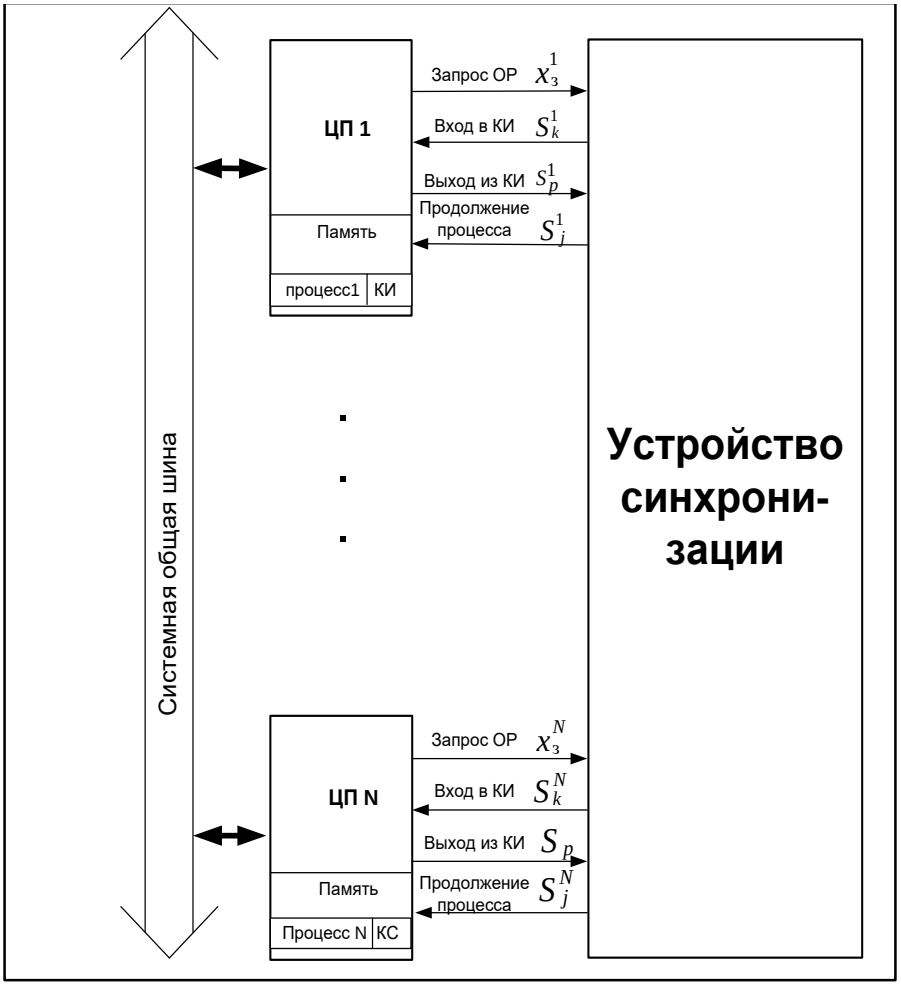


Рисунок 2. Структура многопроцессорной системы с устройством синхронизации процессов

Когда синхронизатор принимает решение допустить процесс в его критическую секцию, он устанавливает на соответствующем выходе уровень «1» – это сигнал «Вход в КИ». Процесс, отправивший команду «Запрос ОР», непрерывно контролирует этот вывод, ожидая разрешения. После завершения работы в критическом интервале процесс выдаёт на свой специальный вывод «Выход из КИ» сигнал «1». Синхронизатор, руководствуясь правилами взаимного исключения и очередности, выбирает из ранее полученных запросов следующий процесс и даёт ему разрешение на вход. Процесс, покинувший критическую секцию, по сигналу «Продолжить процесс» возобновляет выполнение основной логики за её пределами.

Далее на базе методики критических интервалов выстраивается схема работы синхронизатора, формализуется её поведение с помощью логики НДА и создаётся функциональная блок-схема устройства.

Алгоритм координации параллельных процессов при доступе к критическому ресурсу отличается следующими ключевыми моментами. Критическим называется ресурс, способный обслуживать лишь одного участника за раз. Чтобы устранить возможность конфликтов, все процессы перед обращением к такому ресурсу должны выстроить свою работу так, чтобы одновременно только один из них владел ресурсом, а остальные находились в состоянии ожидания. На основе этого выделяются три фундаментальных требования к любому алгоритму синхронизации:

– взаимное исключение: в каждый момент времени не больше одного процесса может находиться в своём критическом интервале;

– ограничение времени в секции: ни один из процессов не должен оставаться в критическом интервале бесконечно долго;

– гарантированное обслуживание: никакой процесс не должен томиться в ожидании доступа к ресурсу бесконечно.

Эти условия позволяют обеспечить корректную и эффективную организацию временного разделения ресурса между параллельными задачами.

В формулировке правил синхронизации ключевым понятием является «критический интервал» – участок программы, в котором выполняются операции с общим ресурсом и который не должен выполняться более чем одним процессом одновременно. На таких отрезках кода происходит обращение к разделяемому ресурсу, поэтому требование о том, чтобы в любой момент времени в своем критическом интервале находился не более одного процесса, реализуется через механизм взаимного исключения этих интервалов.

Соответственно, события, отвечающие за вход в критическую секцию и за само пребывание в ней, должны быть несовместимыми: никогда не должно срабатывать более одного события входа одновременно. Кроме того, порядок допуска процессов внутрь критического интервала определяется выбранной дисциплиной обслуживания (например, приоритетной или циклической), и это тоже учитывается при формализации соответствующих событий.

Второе и третье требования – недопущение бесконечного владения ресурсом одним процессом и бесконечного ожидания другим – обеспечиваются за счёт корректной формализации событий приёма заявок и их обслуживания. Алгоритм должен принимать новую заявку только в том случае, если предыдущая критическая секция этого процесса уже завершена. Такой подход исключает сценарии, при которых один процесс с высшим приоритетом постоянно возвращается в критический интервал и блокирует остальные, в отсутствие механизмов динамической смены приоритетов во время выполнения.

С учётом изложенных требований и приведённых способов их обеспечения, система канонических уравнений, описывающая события входа i-го процесса в соответствующий критический интервал, может быть записана следующим образом [1]:

$$S_k^i(t+1) = S_{\text{вп}}^i(\text{t}) \,\&\, S_{\text{вз}}^i(\text{t}) \,\&\, S_{\text{пр}}^i(\text{t}) \vee S_k^i(\text{t})\overline{S}_p^i,\ i = \overline{1,n}*,$$

где S_k^i – переменная события – вход i-го процесса;

$S^i_{\text{вп}}$ – событие – получение заявки i-го процесса на доступ к ОР;

S^i_p – переменная события – выход i-го процесса из критического участка по завершении работы с ОР;

$S^i_{\text{вз}}$ – гарантия взаимного исключения КИ за счёт несовместимости данного события с другими из ряда, состоящего из n элементов:

$$S^i_{\text{вз}} = \bigwedge_{(\forall\alpha)[\alpha\neq i]} \overline{S}^{\alpha}_k \,,\; i=\overline{1,n}\,,$$

$S^i_{\text{пр}}$ – приоритетное обслуживание i-го процесса.

Событие $S^i_{\text{пр}}$, иллюстрирующее циклическую дисциплину обслуживания с относительным приоритетом, может быть формализовано следующим образом [1]:

$$S^i_{\text{пр}}(t+1) = S^i_{\text{пр}}(0) \vee S^i_{\text{ВП}}(S^i_{pk} \bigwedge_{(\forall\alpha)[\alpha\neq i]} \overline{S}^{\alpha}_{\text{ВП}} \vee$$

$$\vee\; S^{i\oplus 1}_{pk} \bigwedge_{(\forall\alpha)[i\oplus 1<\alpha\le i-1]} \overline{S}^{\alpha}_{\text{ВП}} \vee \ldots \vee S^{i\oplus(n-1)}_{pk} \bigwedge_{(\forall\alpha)[i\oplus n-1<\alpha\le i-1]} \overline{S}^{\alpha}_{\text{ВП}})\,,$$

где $S^i_{\text{пр}}(0)$ – первая часть выражения – это сокращённое обозначение события, задающего начальный приоритет обслуживания i-го процесса,

$$S^i_{\text{пр}}(0) = S_0 x_n S^i_{\text{ВП}} \bigwedge_{(\forall\alpha)[\alpha<i]} \overline{S}^{\alpha}_{\text{ВП}}\,,$$

где S_0 – возникает перед обращением к ОР;

$$S_0(t+1) = x_0 \vee S_0 \overline{x}_n \vee S_{\text{Т}}\,,$$

x_n – сигнал запуска системы управления;

x_0 – сигнал установки системы управления в исходное состояние;

$\overline{S}^i_{\text{ВП}}$ – приём заявки i-го процесса на обслуживание;

S_{pk}^{i} – задержанная на один такт версия события S_{p}^{i}, используемого для реализации циклической дисциплины обслуживания.

$S_{\text{Т}}$ – завершение цикла обслуживания процессов, подавших заявки на обслуживание

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

Событие «приём запроса от i-го процесса» определяется следующим образом:

$$S_{\text{ВП}}^{i}(\text{t}+1) = \left(S_{\text{з}}^{i} \vee S_{\text{ВП}}^{i}\right)\overline{S}_{k}^{i},$$

где $S_{\text{з}}^{i}$ – взятие заявки процесса на обработку.

Исходя из основных требований синхронизации при работе с общими ресурсами, можно построить граф недетерминированного автомата (НДА), описывающий алгоритм взаимодействия между процессами. Этот граф отображает возможные переходы между состояниями процессов в рамках управления доступом к разделяемым данным.

Граф НДА, реализующий алгоритм координации параллельных процессов при работе с общим ресурсом в системе из n процессов, представлен в приложении А.

На данном графе выделены дополнительные события, которые интерпретируются следующим образом [1]:

$$S_{з}^{i}(t+1) = x_{з}^{i} \vee S_{з}^{i}\overline{S_{вп}^{i}} \ ,$$
$$S_{m}(t+1) = S_{k}^{1}S_{вп}^{1} \vee S_{k}^{2}S_{вп}^{2} \vee \cdots \vee S_{k}^{n}S_{вп}^{n},$$
$$S_{nk}(t+1) = S_{m},$$
$$S_{r}^{i}(t+1) = \left(S_{вп}^{1} \vee S_{r}^{i}\right)\overline{S}_{p}^{i},$$
$$S_{p}^{i}(t+1) = S_{nk}S_{k}^{i} \ ,$$
$$S_{pk}^{i}(t+1) = S_{p}^{i} \vee S_{pk}^{i}S_{np}^{i}S_{T} \ ,$$
$$S_{j}^{i}(t+1) = S_{r}^{i}S_{p}^{i} \ ,$$

где $x_{з}^{i}$ – сигнал запроса доступа i-го процесса к общему ресурсу;

S_{m} – событие, запускающее процедуру обращения к разделяемым данным;

S_{nk} – событие, завершающее процедуру обращения к разделяемым данным;

S_{r}^{i} – событие, обозначающее ожидание выхода i-го процесса из критического участка;

S_{j}^{i} – событие, инициирующее возобновление работы i-го процесса после выхода из критического интервала.

Соединительная вершина графа J($\vee$) реализует дизъюнкцию конъюнкций переменных S_{k}^{i} и $S_{вп}^{i}$. Такой тип соединения не требует внесения изменений в события алгоритма управления, поскольку объединяемые конъюнкции переменных являются несовместимыми.

Схема синхронизатора реализована на языке VHDL в виде двух отдельных модулей (см. рисунок 3). Первый модуль отвечает за приём и сохранение сигналов-запросов на доступ к общему ресурсу, а также фиксирует факт входа i-го процесса в его критическую секцию. Второй модуль включает в себя логику управления самим ресурсом – определение моментов начала (Sm) и конца (Snk) обслуживания – а также реализует механику взаимного

исключения и выбор приоритетного обращения из восьми параллельных запросов.

Первый модуль охватывает все булевы функции, относящиеся к блоку регистрации и допуска в критическую секцию, за исключением уравнений, задающих события начала и завершения доступа (Sm и Snk). В дальнейшем этот блок используется как компонент в составе второго модуля, который формирует полную модель восьмиканального синхронизатора. Всего в «сборке» второго модуля содержится восемь экземпляров первого – по одному на каждый канал запросов.

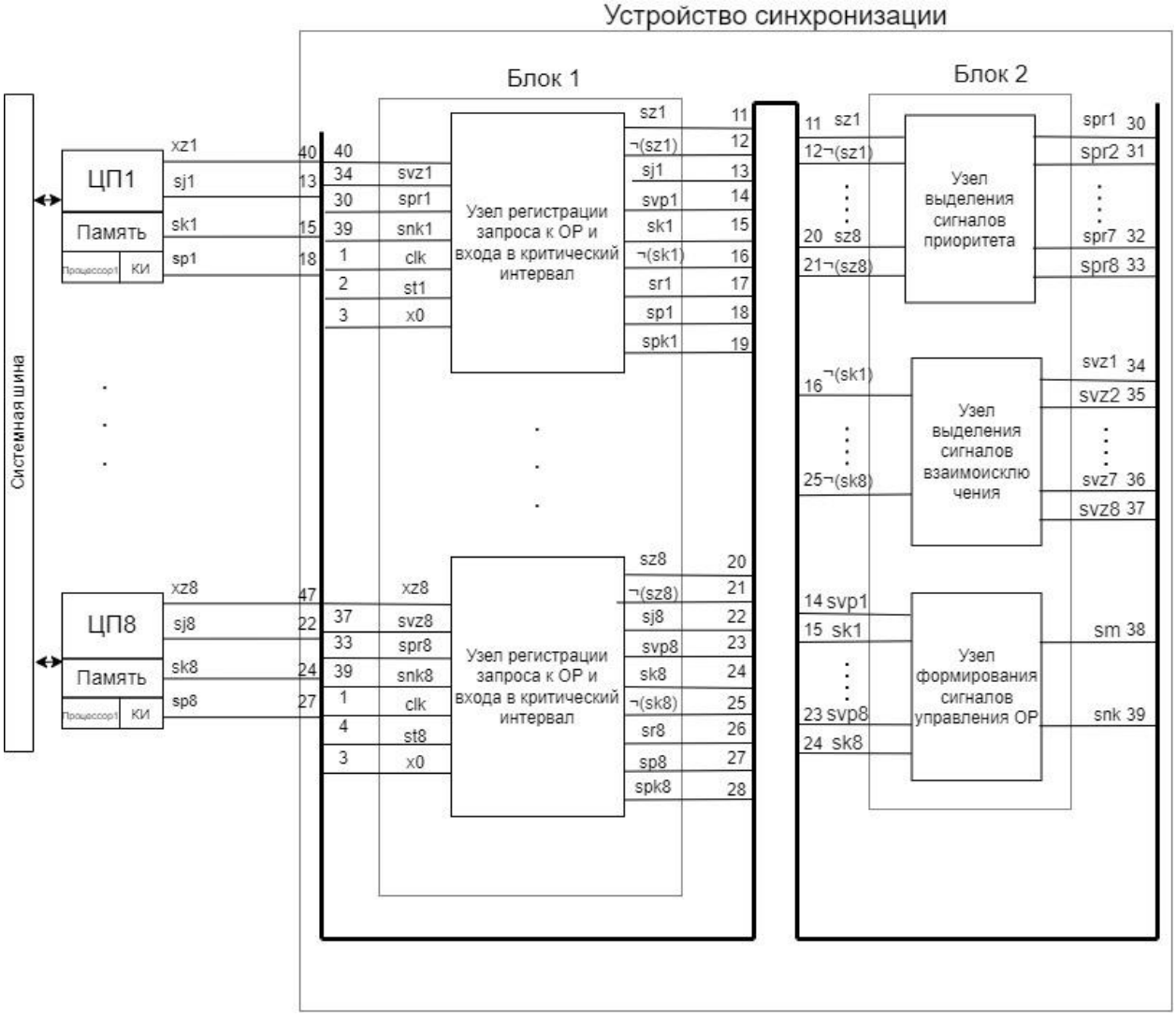


Рисунок 3 – Схема модели устройства синхронизации для восьми процессов

Во втором блоке выполняется начальная установка всех вспомогательных переменных. Третий блок отвечает за генерацию запросов процессов на доступ к общему ресурсу. После этого система проверяет, поступил ли запрос от i-го процесса: при отсутствии – возвращается к блоку 3; при наличии – переходит в блок 5, где оценивается, был ли запрос принят к обслуживанию (вычисляется флаг Svp). Если приём заявки не подтверждён, алгоритм возвращается к блоку 5 до момента фиксации.

Когда заявка одобрена, в шестом блоке рассчитывается переменная Sz, что означает её регистрацию; если же нет – возвращение к блоку 5 предотвращает потерю запроса. В седьмом блоке проверяется, свободен ли критический интервал: если он занят, управление вновь уходит к блоку 6, иначе – к блоку 8, где вычисляются функции взаимного исключения (Svz) и приоритетов (Spr). Далее определяется Sk, сигнализирующий о входе процесса в критическую зону.

Десятый блок выполняет саму работу внутри критического интервала, а в одиннадцатом проверяется её завершение: пока процесс не закончит, цикл остаётся в блоке 10; после окончания в блоке 12 вычисляется Sp, фиксирующий выход из КИ. В тринадцатом блоке определяется Sj, запускающий дальнейшее выполнение процесса за пределами критической секции. В четырнадцатом блоке происходит циклическая ротация приоритетов через расчёт Spk. Наконец, в блоке 15 проверяется, завершена ли вся работа с общим ресурсом: при «да» алгоритм завершает свою работу, иначе возвращается к блоку 3 для обработки новых запросов.

В ходе выполнения дипломной работы была разработана программа на языке функционального моделирования VHDL, которая полностью описывает систему канонических уравнений, код программы представлен в приложении В. Граф НДА для N > 1 представлен в приложении А.1. Расширенный граф НДА для устройства синхронизации представлен в приложении А.2.

4.2 Формализация алгоритма управления взаимодействующими параллельными процессами в задаче «производители-потребители»

Задача «производители–потребители» является одной из базовых проблем синхронизации параллельных процессов, которые взаимодействуют посредством общего буфера – разделяемой области памяти, используемой для передачи данных. В рассматриваемой модели буфер реализуется в виде кольцевой очереди: производитель помещает новые данные в её конец, а потребитель извлекает их с начала, соблюдая принцип FIFO (первым пришёл – первым обслужен).

В задаче «производители–потребители» кольцевой буфер выступает в роли «согласующего звена» между двумя классами процессов: теми, кто генерирует данные, и теми, кто их потребляет. Представьте себе массив фиксированного размера, который «закольцован» так, что после последней ячейки вновь следует первая. Это позволяет эффективно использовать память, избегая сдвигов при добавлении и извлечении элементов. Каждый раз, когда производитель помещает новую порцию данных в буфер, он записывает её в текущую «голову» кольца и передвигает указатель вперёд. Аналогично, потребитель читает элемент из «хвоста» и сдвигает соответствующий указатель дальше по кольцу.

Ключевая особенность кольцевой структуры заключается в том, что она естественным образом поддерживает FIFO-поведение: первыми обрабатываются именно те данные, которые были занесены раньше всех. При этом контроль за состоянием буфера ведётся с помощью двух счётчиков или указателей – на запись и на чтение. Разница между ними указывает, сколько элементов в настоящий момент занято, а их равенство сигнализирует о пустом буфере. Если же счётчики расходятся на величину, равную размеру буфера, это означает, что все ячейки заняты, и производитель должен приостановить добавление новых данных до освобождения места.

Такой подход позволяет обойтись без сложных операций сдвига памяти и обеспечивает высокую пропускную способность: и запись, и считывание производятся за константное время, независимо от наполненности буфера. При этом важно обеспечить строгую синхронизацию: если потребитель пытается забрать элемент из пустого буфера, он должен быть приостановлен до момента, когда в кольце появятся новые данные; аналогично, если производитель сталкивается с переполненным буфером, он ждёт, пока потребитель освободит хотя бы одну ячейку. В этом и состоит суть взаимодействия «производитель–потребитель» через кольцевой буфер – никакой «гонки» не происходит, и каждая операция выполняется как атомарное перемещение указателя с учётом текущего состояния системы.

В аппаратной или микропрограммной реализации такой механизм часто сопровождается дополнительными флагами переполнения и опустошения, а также внутренними средствами блокировки и разбудительных сигналов. Именно благодаря кольцевому буферу достигается баланс между производительностью и простотой структуры: он устраняет необходимость в громоздких очередях, сводя всю логику обмена к манипуляциям с двумя указателями и несколькими булевыми условиями.

Чтобы избежать конфликтных ситуаций – например, попытки чтения из пустого буфера или записи в уже переполненный, – требуется чётко выстроенный механизм синхронизации между процессами. Такой механизм описывается формально с использованием модели недетерминированного автомата (НДА), в рамках которой задаются все возможные события, сопровождающие функционирование алгоритма обмена.

Поведение кольцевого буфера формализуется как система состояний, в которых он может быть либо пустым, либо полным, либо находиться в промежуточной загрузке. С учётом этих состояний и требований к корректной синхронизации (в частности, взаимному исключению доступа), формируется граф НДА. Его вершины и переходы отражают ключевые события: инициирование и подтверждение запросов на запись, чтение, освобождение

буфера и других операций, обеспечивающих работу системы «производитель–потребитель».

Для реализации буфера используют два целочисленных счётчика: СчБз – индекс первой свободной (пустой) ячейки, СчБчт – индекс первой занятой (заполненной) ячейки.

При инициализации оба счётчика устанавливаются в 1 (например, при t=0 СчБз=1, СчБчт=1). Если размер буфера n=8, то оба счётчика работают по модулю 8, и при достижении n происходит «оборачивание» счётчика в начало. Переполнение, обнаруживаемое по совпадению указателей, служит сигналом о том, что буфер либо полностью заполнен, либо пуст, в зависимости от направления сравнения, и эти сигналы используются в логике управления состоянием буфера.

В основе механизма кольцевого буфера лежат два независимых счётчика – один для записываемых элементов, другой для читаемых. Счётчик записи указывает на «следующую» ячейку, куда производитель поместит новый фрагмент данных, а счётчик чтения – на ту ячейку, из которой потребитель извлечёт следующий элемент. После каждой операции оба счётчика автоматически инкрементируются, причём, дойдя до конца буфера, «перепрыгивают» на начальную позицию, обеспечивая циклическую организацию памяти.

Состояние буфера в любой момент определяется разностью между значениями этих счётчиков. Если они совпадают, буфер пуст — ещё не было незавершённых записей или все данные уже прочитаны. Когда разница достигает размера буфера, это означает полное заполнение, и запись новых данных становится невозможной до освобождения хотя бы одной ячейки. Благодаря этой логике каждый из процессов может в любой момент по своему указателю выяснить, доступна ли ячейка для операции: если счётчик записи догоняет счётчик чтения на максимальное расстояние, производитель приостанавливает работу; если же счётчик чтения сравнивается со счётчиком записи, потребитель временно блокируется.

Такой подход позволяет обойтись без сложных циклов сдвига элементов и дорогостоящих операций перемещения памяти: указатели движутся по кольцу и обслуживаются аппаратно или программно за константное время. Вдобавок два простых булевых флага «полон» и «пуст» – порождаемые сравнением счётчиков – дают прямую информацию о готовности буфера к приёму или выдаче данных. В результате поддерживается строгая очередность обслуживания FIFO и отсутствует риск перезаписи ещё не прочитанных данных или чтения мусора.

Рассмотрим пример работы кольцевого буфера. Представим буфер массивом из 8 ячеек (рисунок 4), а в каждую ячейку записывается одно сообщение.

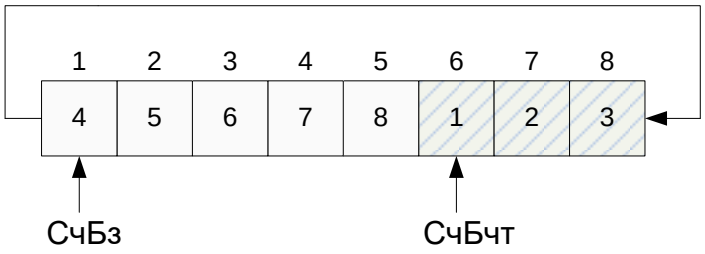


Рисунок 4 – Произвольное положение массива в буфере

Допустим, что каждое передаваемое сообщение занимает отдельную ячейку в буфере, при этом номер ячейки совпадает с порядковым номером сообщения. В начальном состоянии, до начала обмена между процессами, буфер полностью свободен – это отображается в верхнем ряду, где все ячейки обозначены как незанятые.

При записи новый такт работы буфера делится на два этапа:

1) запись в ячейку – данные вводятся именно в ту ячейку, на которую указывает счётчик записи (СчБз);

2) кольцевой сдвиг – после записи содержимое всей очереди в буфере циклически смещается на одну позицию влево.

На рисунке 4 показано, как после трёх таких тактов значение счётчика записи изменилось с 1 на 4 (поскольку 1+3=4 по модулю размера буфера), и ячейка под номером 4 стала первой свободной в конце очереди.

На рисунке 5 показано состояние буфера после ещё пяти циклов записи:

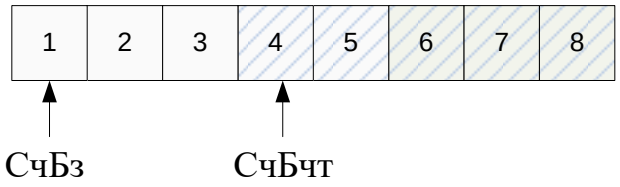


Рисунок 5 – Состояние буфера

После записи трёх порций информации за один такт буфер примет вид, представленный на рисунке 6:

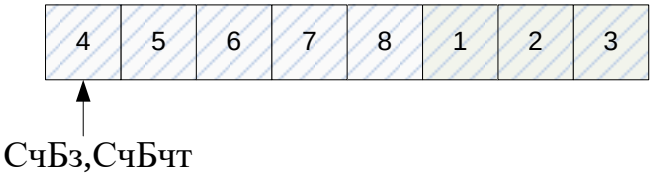


Рисунок 6 – Состояние буфера

В данном состоянии буфер считается заполненным, поскольку значения счётчиков равны.

В следующем такте возможна только операция чтения, так как буфер полностью заполнен. Предположим, что считывается 5 порций данных – тогда состояние буфера изменится, как показано на рисунке 7.

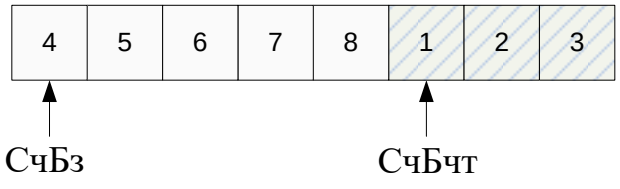


Рисунок 7 – Состояние буфера

После выполнения операции чтения значения буферных счётчиков изменяются следующим образом: счётчик записи (СчБз) остаётся равным 4, а счётчик чтения (СчБчт), учитывая циклическую природу буфера, принимает значение 4 + 5 = 9, что по модулю размера буфера эквивалентно 1.

Алгоритм взаимодействия параллельных процессов в задаче «производители–потребители» может быть формализован с помощью графа недетерминированного автомата (НДА), в котором переходы между состояниями определяются событиями записи и чтения, а также текущим состоянием буфера.

Для правильной организации обмена данными через синхронизирующий буфер между производителями и потребителями предъявляются следующие основные требования:

– обеспечение взаимного исключения – гарантированный эксклюзивный доступ к буферу сообщений как к критическому ресурсу, должен предоставляться только одному процессу в определённый момент времени: либо производителю, либо потребителю;

– контроль за состоянием буфера – алгоритм должен учитывать текущую заполняемость буфера:

1) если производитель пытается добавить сообщение в переполненный буфер, его выполнение должно быть приостановлено до освобождения места;

2) потребитель, пытающийся считать данные из пустого буфера, должен быть приостановлен до момента поступления новой информации.

Корректная организация доступа к разделяемому кольцевому буферу во многом определяет надёжность и предсказуемость всей системы «производитель–потребитель». Если бы два или более процесса могли одновременно обращаться к одной и той же ячейке памяти, неизбежно возникали бы состояния неконсистентности: производитель мог бы затереть данные, ещё не извлечённые потребителем, а потребитель – прочитать «пустое» или частично заполненное место. Такие ошибки сложно отловить на этапе тестирования, они проявляются лишь под определёнными, часто редко

встречающимися нагрузками, что делает их устранение крайне трудоёмким и затратным. Введение строгого механизма взаимоисключения нивелирует эту проблему: операции записи и чтения становятся атомарными, и буфер всегда остаётся в корректном, согласованном состоянии.

Не менее важным элементом надёжного обмена является отслеживание текущей степени заполненности кольца. Статические указатели на «голову» и «хвост» очереди позволяют мгновенно определить, пуст ли буфер или въехал в полный цикл, но этого оказалось недостаточно для комплексного управления потоками. Без учёта переполнения производитель мог бы неоднократно перезаписывать одни и те же ячейки, качественно теряя информацию, а потребитель, столкнувшись с отсутствием новых данных, – возвращать неподходящие «шумовые» значения. Блокировка соответствующего процесса до момента появления условий для безопасной операции исключает любые потери данных и гарантирует, что каждый элемент будет в точности передан и обработан.

В совокупности оба этих механизма – жесткое взаимное исключение и динамический контроль заполненности – формируют прочный каркас для любых сценариев взаимодействия «производитель–потребитель». Они позволяют обеспечить равномерную, бесперебойную работу даже при значительных объёмах запросов: пока один поток занят записью, другой наверняка не начнёт чтение из той же ячейки, а при необходимости приостановится, не вывалившись за пределы выделенной памяти. Такая структура не только упрощает формальную верификацию алгоритма, но и даёт устойчивое поведение в реальной аппаратной или программной среде – ключевой фактор для промышленных и образовательных приложений.

Граф НДА алгоритма:

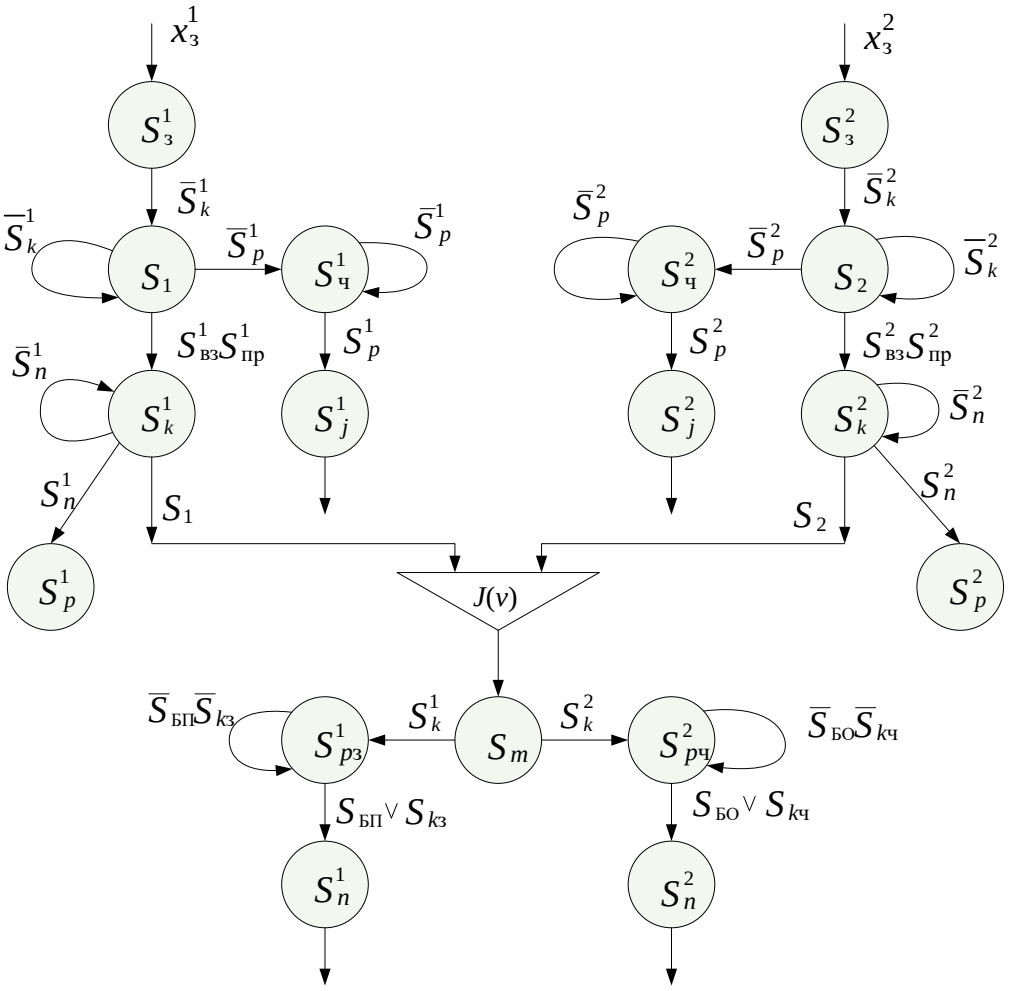


Рисунок 8. Граф НДА алгоритма управления межпроцессного взаимодействия в задаче «производители-потребители» с согласующим кольцевым буфером сообщений (для 2-х процессов: «клиент – сервер»).

Итоговые уравнения имеют вид [1]:

$$S_k^1(t+1) = S_1 \overline{S}_k^2 \overline{S}_{БП} \vee S_k^1 \overline{S}_n^1 ,$$

$$S_k^2(t+1) = S_2 \overline{S}_k^1 (S_{БП} \vee \overline{S}_{БО} \overline{S}_1) \vee S_k^2 \overline{S}_n^2 ,$$

$$S_1(t+1) = (S_{1,з} \vee S_1)\overline{S}_k^1 ,$$

$$S_2(t+1) = (S_{2,з} \vee S_2)\overline{S}_k^2 .$$

Представление событий S_1 и S_2 в виде формул свидетельствует о том, что заявки процессов на операции запись или чтения могут быть восприняты

только в том случае, если процессы не находятся в своих критических интервалах.

События $S^1_{\text{ч}}$ и $S^2_{\text{ч}}$, символизирующие ожидание взаимодействующими процессами окончания процедуры записи и чтения из буфера, чтобы продолжить свою работу, прерванную обращением к согласующему буферу, формализуются так:

$$S^1_{\text{ч}}(t+1) = (S_1 \vee S^1_{\text{ч}})\overline{S}^1_p,$$
$$S^2_{\text{ч}}(t+1) = (S_2 \vee S^2_{\text{ч}})\overline{S}^2_p.$$

Остальные события, реализуемые алгоритмом межпроцессного взаимодействия в задаче «производители-потребители», представленные на графе НДА (рисунок 8), формализуются таким образом:

$$S_m(t+1) = S^1_k S_1 \vee S^2_k S_2,$$
$$S^1_{p\text{з}}(t+1) = S_m S^1_k \vee S^1_{p\text{з}}(\overline{S}_{\text{БП}}\overline{S}_{k\text{з}}),$$
$$S^2_{p\text{ч}}(t+1) = S_m S^2_k \vee S^2_{p\text{ч}}(\overline{S}_{\text{БО}}\overline{S}_{k\text{ч}}),$$
$$S^1_n(t+1) = S^1_{p\text{з}}(S_{\text{БП}} \vee S_{k\text{з}}),$$
$$S^2_n(t+1) = S^2_{p\text{ч}}(S_{\text{БО}} \vee S_{k\text{ч}}),$$
$$S^1_p(t+1) = S^1_k S^1_n,$$
$$S^2_p(t+1) = S^2_k S^2_n,$$
$$S^1_j(t+1) = S^1_{\text{ч}} S^1_p,$$
$$S^2_j(t+1) = S^2_{\text{ч}} S^2_p.$$

Граф НДА для N > 1 представлен в приложении А.2. Расширенный граф НДА для устройства синхронизации представлен в приложении А.3.

4.3 Формализация алгоритма управления взаимодействующими параллельными процессами в задаче «спящий парикмахер»

Формулировка задачи «спящий парикмахер» такова: в маленьком городке действует парикмахерская с одним входом и выходом, где оборудовано несколько мест для ожидающих клиентов и одно кресло, занимаемое парикмахером. Если салон пуст, он садится в это кресло и засыпает. Когда приходит посетитель, он либо будит мастера, либо, если тот уже занят, садится в свободное кресло и ждёт своей очереди – а если все кресла заняты, уходит. После того как парикмахер заканчивает стрижку, он провожает клиента к выходу, закрывает за ним дверь и либо будит следующего посетителя, либо, если никого нет в зале, возвращается в своё кресло и снова засыпает [1].

В терминах параллельных процессов посетитель выступает в роли клиента, запрашивающего услугу, а парикмахер – как сервер, предоставляющий её. Необходимо синхронизировать их действия: должно произойти «рандеву» – парикмахеру ожидать прихода клиента; клиент, в свою очередь, ждёт освобождения кресла для стрижки. После окончания стрижки клиент освобождает кресло и покидает салон, а парикмахер, прежде чем снова закрыть дверь, дожидается ухода клиента. Затем цикл повторяется – либо с новым посетителем, либо с возвращением мастера в спящее состояние.

Чтобы формализовать взаимодействия параллельных процессов в рамках рассматриваемой задачи целесообразно применить методику систем канонических уравнений [1]:

$S_з^k$ – событие, отражающее факт прибытия клиента в парикмахерскую;

$S_{\text{оч}}^k$ – событие, обозначающее наличие свободного места в очереди к парикмахеру;

S_0^k и $S_з^n$ и $S_{ЗД}^n$ – события, указывающие на пребывание клиента и парикмахера внутри помещения соответственно;

$S_{БП}^k$ и $S_{БП}^n$ – события, фиксирующие, что клиент пробуждает парикмахера и, наоборот, парикмахер будит клиента;

$S_{КП}^k$ и $S_{ОК}^n$ – события, обозначающие, что клиент занимает кресло для стрижки, а парикмахер освобождает его для клиента;

S_C^k и S_C^n – события, отражающие состояния ожидания, когда клиент и парикмахер находятся в состоянии сна;

$S_{ГТ}^k$ и $S_{ГТ}^n$ – события, показывающие готовность клиента и парикмахера к началу процесса стрижки;

S_n^k и S_k^n – события, сигнализирующие о начале и завершении процесса стрижки;

$S_{ОК}^k$ и $S_{ОД}^n$ события, свидетельствующие о том, что клиент покидает кресло, а парикмахер открывает дверь для его выхода;

S_y^k и $S_{ЗД}^n$ – события, фиксирующие, что клиент вышел из парикмахерской, а парикмахер закрыл за ним дверь;

Опираясь на текстовое описание алгоритма управления в задаче «спящий парикмахер» и используя ранее введённые события, сформулируем систему канонических уравнений, отражающую динамику этих процессов. Также представим граф соответствующего недетерминированного автомата (НДА), моделирующего данное поведение [1]:

– для процесса – клиента:

$$S_0^k(t+1) = S_З^k S_{ОЧ}^k,$$

$$S_c^k(t+1) = S_0^k \overline{S_C^n} \vee S_C^k \overline{S_{БК}^n},$$

$$S_{БП}^k(t+1) = S_0^k S_C^n,$$

$$S^k_{КП}(t+1) = S^k_C S^n_{БК} \vee S^k_{БП} S^n_{ОК},$$

$$S^k_{ГТ}(t+1) = S^k_{КП} S^n_{ЗД} \vee S^k_{ГТ} S^n_{ГТ},$$

– для процесса – сервера (парикмахер):

$$S^n_C(t+1) = S^n_0 \overline{S^k_0} \vee S^n_0 \overline{S^k_{БП}},$$

$$S^n_{БК}(t+1) = S^n_0 S^k_0 S^k_c,$$

$$S^n_{ОК}(t+1) = (S^n_0 S^k_0 \overline{S^k_0} \vee S^n_c) S^k_{БП},$$

$$S^n_{ГТ}(t+1) = (S^n_{ОК} \vee S^n_{БК}) S^k_{КП} \vee S^n_{ГТ} \overline{S^k_{ГТ}},$$

– система канонических уравнений, описывающая события после рандеву:

$$S_H(t+1) = S^k_{ГТ} S^n_{ГТ},\ S^n_{ОД}(t+1) = S_K,$$

$$S_K(t+1) = S_H, \qquad S^k_y(t+1) = S^k_{ОК} S^k_{ОД},$$

$$S^k_{ОК}(t+1) = S_K, \qquad S^n_{PL}(t+1) = S^m_{ОД} S^k_У,$$

Граф НДА алгоритма управления взаимодействующими процессами в задаче «спящий парикмахер» представлен в приложении А.4.

4.4 Методика испытаний программы

Функциональное тестирование направлено на проверку корректности реализации всех заявленных функций с целью установления их соответствия требованиям, изложенным в техническом задании.

Рассмотрим работу алгоритма управления доступом к ОР.

Из рисунков в приложении В.1 – В.4 видно, что алгоритм работает корректно, предоставляя доступ к ресурсу запросившим его процессам по очереди – в том числе и в случае одновременного обращения сразу нескольких процессов.

Из этого можно сделать вывод, что алгоритм управления параллельными процессами при доступе к критическим ресурсам функционирует корректно, так как обеспечивает так как обеспечивает корректное разрешение обычных и конфликтных ситуаций.

Рассмотрим работу алгоритма управления в задаче "производители-потребители".

На рисунках В.5 – В.7 видно, что подаются заявки на запись – xz1 и на чтение – xz2, а также количество порций на которое нужно изменить буфер в соответствующих тактах. В начале выполнения алгоритма Sb0 = 1, т.е. буфер пуст. В процессе выполнения алгоритма Sb0 изменяется в зависимости от текущего состояния буфера.

Из этого можно сделать вывод, что алгоритм управления параллельными процессами в задаче «производители-потребители» функционирует корректно, так как обеспечивает так как обеспечивает корректное разрешение обычных и конфликтных ситуаций.

Рассмотрим работу алгоритма управления в задаче "спящий парикмахер".

На рисунке В.8 видно, что в момент времени 100 нс, парикмахеру поступает заявка от клиента (Skz(4)). В момент времени 140 нс он допускается к обслуживанию (Skocho(4)), и затем происходит обслуживание. Парикмахер

постепенно меняет состояния (Spso, Spbko, Spgto, Spodo, Spko). Завершается всё тем, что в момент времени 330 нс клиент уходит (Sky(4)), а парикмахер в 350 нс закрывает дверь (Spzdo).

В момент времени 490 нс приходит заявка от другого клиента (2) – процедура повторяется.

В момент времени 880 – 1360 нс (В.9) приходят сразу же два клиента, которые обслуживаются по очереди.

В момент времени 1510 нс приходит одиночный клиент, который уходит слишком рано, не успевая обслужиться.

В момент времени 1940 – 2740 (В.10) приходят сразу несколько клиентов вразноброс. Все корректно обслуживаются.

Из рисунков Б.8 – В.10 видно, что парикмахер спит в промежутках между прибытием новых клиентов (Spso). Из рисунков В.11 – В.13 видно, что клиенты спят в ожидании обслуживания их парикмахером (Skso).

Из этого можно сделать вывод, что алгоритм управления параллельными процессами в задаче «спящий парикмахер» функционирует корректно, так как обеспечивает так как обеспечивает корректное разрешение обычных и конфликтных ситуаций.

5 Оценка затрат на разработку и внедрение программного комплекса

5.1 Организация производства разработки программного комплекса

Разработка программного комплекса осуществлялась в помещении площадью 30 кв. м. Затраты на его содержание составляют 20 000 рублей в месяц.

В процессе реализации проекта были приобретены все необходимые технические средства, включая вычислительное оборудование, закуплено лицензионное программное обеспечение, а также использованы расходные материалы и сопутствующие ресурсы.

Обобщённые данные по оборудованию и программному обеспечению представлены в таблицах 2 и 3 соответственно.

Таблица 2 – Потребность в оборудовании

Оборудование	Источник	Количество	Ориентировочная цена за единицу, рублей	Потребляемая мощность, Ватт
Ноутбук ASUS X751MA	ООО «Мвидео»	3	30000	65

Таблица 3 – Потребность в лицензионном программном обеспечении

Лицензионное программное обеспечение	Источник	Цена, рублей	Срок действия лицензии, дней
Microsoft Windows 10	Входит в комплект «Ноутбук ASUS X751MA, 17"»	-	Бессрочная
Xilinx Vivado	Сайт разработчика	-	Бессрочная
Антивирус Kaspersky Plus 2025	Интернет-магазин Kaspersky URL: https://www.kaspersky.ru/	2199	365

Список дополнительных материалов, сведения об их поставщиках, объёмы и примерные стоимости приведены в таблице 4.

Таблица 4 – Потребность в дополнительных материалах

Материал	Источник	Количество, штук	Ориентировочная цена за единицу, рублей
Стол компьютерный МЕБЕЛЬ-М	«Моя родня»	3	5000
Лист МДФ 440x250x16	«Моя родня»	10	265
Лампа настольная бестеневая на струбцине, черная	«Моя родня»	3	4100
Офисное кресло Chairman 696 LT TW-01	«Моя родня»	3	5100
Канцтовары	Гипермаркет «Канцлер»	3	2000

Разработка программного комплекса осуществлялась командой разработчиков, представленных в таблице 5.

Таблица 5 – Состав коллектива разработчиков

Специалист	Количество, человек	Описание занятости
Руководитель проекта	1	– управление этапами проекта; – подбор команды; – распределение ресурсов; – контроль требований заказчика; – приёмка и внедрение; – анализ предметной области.
Программист	1	– формирование базы данных; – разработка, отладка и первоначальное тестирование программного кода; – подготовка и согласование программной документации; – сдача и внедрение программного продукта.
Тестировщик	1	– проверка и отладка серверной части; – тестирование клиентской части; – комплексное тестирование всего программного комплекса.

Информация о базовой заработной плате разработчиков информационно-автоматизированной системы и численности сотрудников представлена в таблице 6.

Таблица 6 – Сведения о заработной плате

Специалист	Количество, человек	Сумма заработной платы, рублей
Руководитель проекта	1	150000
Программист	1	90000
Тестировщик	1	60000

5.2 Расчет себестоимости программного обеспечения для программного комплекса

Себестоимость разработки информационно-автоматизированной системы складывается из следующих ключевых статей расходов:

– приобретение оборудования;

– покупка лицензионного программного обеспечения;

– расход дополнительных материалов;

– оплата труда разработчиков с учётом обязательных страховых взносов;

– затраты на сопровождение и техническое обслуживание программного обеспечения.

Расходы на оборудование в период реализации проекта составили 90 000 рублей (подробная информация приведена в таблице 2).

Затраты на программное обеспечение за указанный период составили 2 199 рублей (см. таблицу 3).

Расходы на дополнительные материалы составили 39 600 рублей (см. таблицу 4).

Заработная плата включает базовую оплату труда специалистов и обязательные страховые взносы. При пятидневной рабочей неделе и среднем

количестве 21 рабочего дня в месяц общая длительность проекта составила 3 месяца.

Ежемесячные расходы на оплату труда составляют 300000 рублей, что в итоге даёт 900000 рублей за весь период разработки.

Отчисления на страховые взносы:

– обязательные отчисления по нормам, установленным законодательством государственного социального страхования;

– пенсионные отчисления в Пенсионный фонд РФ;

– отчисления в фонд обязательного медицинского страхования;

– отчисления в фонд социального страхования РФ

от затрат на оплату труда работников, включаемых в себестоимость продукции (программного обеспечения), по элементу «Затраты на оплату труда» (кроме тех видов оплаты, на которые страховые взносы не начисляются).

Отчисления на страховые взносы производятся в соответствии с положениями «Налогового кодекса Российской Федерации» от 14 июля 2022 года № 239-ФЗ [2].

С 1 января 2023 года в стране действует единая базовая ставка страховых взносов. Согласно статье 425 НК РФ, общий тариф страховых отчислений, взимаемых с работодателей, составляет 30%.

Однако, учитывая, что компания ведёт деятельность в сфере информационных технологий, она имеет право на применение пониженного тарифа, установленного подпунктом 3 пункта 1 статьи 427 НК РФ. В этом случае применяется льготная ставка 7,6%.

Таким образом, расчёт страховых взносов осуществляется дифференцированно – с части заработной платы, не превышающей размер МРОТ (на 2024 год он составляет 19 242 рубля), взносы уплачиваются по ставке 30%, с суммы, превышающей МРОТ, применяется пониженная ставка 7,6%.

Подробная информация о расчётах и итоговых суммах страховых отчислений представлена в таблице 7.

Таблица 7 – Отчисления на страховые взносы

Сотрудник	Заработная плата, рублей	Взнос по единой ставке, рублей	Сумма, подлежащая пониженной ставке по взносу, рублей	Взнос по пониженной ставке, рублей	Общая сумма взноса с сотрудника за месяц, рублей
Руководитель проекта	150000	5772,6	130758	11400	14600
Программист	90000	5772,6	70758	6840	8992
Тестировщик	60000	5772,6	40758	4560	6080

Затраты на оплату труда разработчиков и начисленные страховые взносы, выполненные в соответствии с действующими нормативными актами, представлены в таблице 8.

Таблица 8 – Затраты на заработную плату

Сотрудник	Заработная плата, рублей	Общая сумма отчислений, рублей	Всего за 1 месяц, рублей	Всего за 3 месяца, рублей
Руководитель проекта	150000	17865	167865	503595
Программист	90000	10719	100719	302157
Тестировщик	60000	7146	67146	201438

Затраты на оплату труда команды, выполняющей проект по созданию информационно-автоматизированной системы, составят 1 007 190 рублей, из которых 35 730 рублей приходится на страховые взносы, а 320 000,3 рублей – непосредственно на заработную плату.

Обслуживание системы включает следующие статьи расходов:

– амортизационные отчисления на оборудование, составляющие 20 % от первоначальной стоимости;

– техническое обслуживание оборудования, равное 15 % от суммы амортизационных отчислений;

– расходы на электроэнергию для персональных компьютеров, оргтехники и бытовых приборов;

– затраты на содержание помещения;

– прочие накладные расходы.

Амортизационные отчисления рассчитываются как 20 % от стоимости оборудования. При покупке трёх ноутбуков по 30 000 рублей каждый, ежемесячные амортизационные отчисления будут равны:

$$\frac{(3 * 30000) * 0{,}2}{12} = 1500 \text{ руб}$$

Техобслуживание оборудования принимается в размере 15 % от суммы амортизационных отчислений.

Следовательно, расходы на обслуживание оборудования составят:

$$1500 * 0{,}15 = 225 \quad \text{руб.}$$

Предполагаем, что в день по 8 часов будут работать 3 ноутбука:

$$66 * 8 * 63 * 3 = 99{,}8 \quad \text{кВт.}$$

За 63-дневный период разработки программного обеспечения на питание ноутбуков будет израсходовано 100 кВт·ч электроэнергии, а на остальные устройства – ещё 0,5 кВт·ч, что в сумме даёт 123,5 кВт·ч.

$$100 + 0{,}5 = 100{,}5 \text{ кВт.}$$

Накладные расходы рассчитываются как 75 % от фонда заработной платы, следовательно:

$$320000{,}3 \times 0{,}75 = 240000.2 \text{ руб.}$$

Итоговые значения по статьям обслуживания представлены в таблице 9.

Таблица 9 – Затраты на обслуживание

Статья затрат	Показатель отчислений	Сумма отчислений в месяц, рублей	Итоговая сумма отчислений за период разработки, рублей
Амортизация оборудования	20%	1500	4500
Обслуживание оборудования	15%	225	675
Затраты на электроэнергию	1,23 на 1 кВт/ч	123.615	370.79
Накладные расходы	75%	240000.2	720000.6
Аренда офиса и расходы на его содержание	–	20000	60000
Итого, рублей	261848	785544	

Величина себестоимости разработки определяется как суммарная величина расходов по всем статьям, указанным в таблице 10.

Таблица 10 – Общая сумма затрат

Статья затрат	Сумма отчислений, рублей
Приобретение оборудования	90000
Приобретение лицензионного программного обеспечения	2199
Затраты на заработную плату	320000,3
Затраты на дополнительные материалы	16465
Затраты на обслуживание	785544
Итого:	1214208.3

Итак, общие расходы на реализацию проекта составят 1214208.3 рублей, что и является себестоимостью программного комплекса.

5.3 Анализ рисков при разработке и внедрении программного комплекса

Риски, возникающие при взаимодействии заказчика и исполнителя, связаны с недостаточным уровнем коммуникации между сторонами или их представителями.

Риски, связанные с планированием, обусловлены отсутствием у менеджера и исполнителей навыков правильной оценки сроков работ: часто недооценивается время на тестирование, исправление ошибок в коде и предпроектное обследование. Чтобы снизить этот риск, руководитель проекта строго контролировал соблюдение графика на каждом этапе.

Недостаточное согласование задач или архитектурных решений может отрицательно сказаться на качестве создаваемого программного комплекса. Для уменьшения этого риска на старте проекта в договоре зафиксировали регламент встреч, частоту согласований, а также утвердили кураторов со стороны заказчика и исполнителя и состав рабочей группы.

Функциональные риски связаны с тем, что система может не обеспечить выполнение возложенных на неё задач. Их минимизация достигается за счёт детального анализа требований заказчика, создания моделей и привлечения квалифицированных специалистов.

Риск текучести кадров возникает, когда ключевые сотрудники, обладающие критическими знаниями о проекте, покидают команду – это способно приостановить разработку. Для снижения этого риска руководитель проекта организует обмен знаниями внутри коллектива, планирует работу с дублированием ролей и проводит внутреннее обучение.

Риск хищения исходного кода проявляется, если уходящие разработчики забирают с собой результаты своей работы. Удерживать специалистов помогают системы материальной и нематериальной мотивации.

Риск юридической ответственности связан с использованием нелицензионного ПО: хотя это может снизить затраты, применение пиратских

программ грозит серьёзными последствиями. Для устранения этого риска компания закупает только официальные лицензии.

В рамках предпроектного анализа необходимо выявить все возможные риски и разработать для каждого из них меры по снижению вероятности или последствий. По результатам анализа составляется сводный отчёт с описанием рисков и способов их минимизации.

6 Общие принципы обеспечения безопасности на рабочем месте оператора программного комплекса

6.1 Актуальность вопросов безопасности человека на рабочем месте

С каждым годом информационные технологии всё глубже проникают в повседневную жизнь человека, чему во многом способствует массовое распространение персональных электронно-вычислительных машин. Поскольку ПЭВМ сегодня используется почти на каждом рабочем месте, особенно важно уделять внимание вопросам безопасности пользователей, сохранению их здоровья и поддержанию работоспособности.

Работа на ЭВМ связана с распознаванием печатного и рукописного текста, анализом графической информации, длительным пребыванием в статичной позе и другими факторами, которые при несоблюдении нормативов могут отрицательно сказаться на самочувствии пользователя.

Согласно ГОСТ 12.0.003-74[2], вредные производственные факторы подразделяются на четыре категории:

– физические: недостаточная или чрезмерная освещённость рабочего места, несоответствие температуры воздуха норме, повышенная влажность, высокий уровень шума и пр.;

– психофизиологические: нервно-психические и физические перегрузки, перенапряжение органов зрения и слуха;

– биологические: присутствие микро- и макроорганизмов, способных вызывать заболевания;

– химические: превышение допустимых концентраций аммиака, фенола, озона, окиси углерода и других вредных веществ в рабочей зоне.

6.2 Техника безопасности при работе с электроприборами

ПЭВМ является электрическим прибором, поэтому при работе с ней необходимо соблюдать следующие правила электробезопасности:

– подключать её только к заземлённым розеткам;

– после отключения от сети не тянуть за шнур и не трогать вилку мокрыми руками;

– не приступать к ремонту до полного отключения устройства от питания;

– перед включением проверить шнур на отсутствие изломов и повреждений изоляции;

– не оставлять включённые приборы без контроля.

Персональная электронно-вычислительная машина включает в себя такие компоненты, как монитор, блок питания, центральный процессор и другие элементы, каждый из которых является источником электромагнитного излучения. Подобные электромагнитные поля также возникают от распределительных щитов, электрической проводки и линий электропередачи.

6.3 Влияние ПЭВМ на физиологию человека

Работа за компьютером связана с длительной нагрузкой на зрение и опорно-двигательный аппарат. Без контроля времени, проводимого перед монитором, это может привести к ухудшению зрения, искривлению позвоночника и нарушению кровообращения.

Интенсивное напряжение глазных мышц негативно влияет на центральную нервную систему: при долгой работе снижается способность к фокусировке, падает острота зрения, повышается чувствительность глаз, они устают, слезятся и краснеют.

Длительное пребывание в статической позе вызывает быструю усталость организма. Для повышения комфорта и снижения рисков рекомендуется обратить внимание на следующие параметры рабочего места:

– регулируемая высота стола и кресла;

– наличие опоры для спины и подлокотников;

– оптимальный угол наклона головы при работе;

– правильная ориентация экрана относительно глаз.

Значительное влияние на самочувствие оказывает уровень шума. В соответствие с СанПиН 2.2.3670-20 «Санитарно-эпидемиологические требования к условиям труда» [3] эквивалентный шум не должен превышать 50 дБА. Постоянное воздействие шума выше нормы ухудшает условия труда и вредит здоровью, проявляясь снижением концентрации, повышенной утомляемостью и падением производительности. Для снижения шумовой нагрузки применяют звукоизоляционные материалы и конструкции.

6.4 Техника безопасности при работе с персональным компьютером

Согласно типовой инструкции по охране труда при работе на персональном компьютере ТОИ Р-45-084-01 [4], утверждённой Приказом Минсвязи РФ № 162 от 02.07.2001, все виды деятельности на ЭВМ разделены на три группы:

– группа А (чтение информации с экрана) требует два перерыва по 15 минут каждый, помимо обеденного;

– группа Б (ввод текста) – перерыв длительностью 10 минут каждый час работы;

– группа В (творческая деятельность) – перерывы по 15 минут каждые 60 минут работы.

Во время таких пауз рекомендуется выполнять физзарядку или специальные упражнения для глаз.

Исходя из изложенного, для обеспечения здоровья сотрудников и минимизации воздействия ПЭВМ необходимо применять только сертифицированную, исправную технику, а также организовывать

рациональный режим труда и отдыха. При этом сама сертификация техники не заменяет регулярного контроля её состояния, поэтому периодическая проверка оборудования обязательна.

Вопросы охраны труда при работе с персональными электронно-вычислительными машинами (ПЭВМ) имеют первостепенное значение, особенно в условиях длительной и интенсивной эксплуатации компьютерной техники. Современные рабочие места всё чаще требуют высокой концентрации внимания, быстрой реакции и многозадачности, что в совокупности с неподвижной позой, постоянной зрительной нагрузкой и влиянием электромагнитных излучений может существенно повлиять на физическое и психоэмоциональное состояние работника. Именно поэтому законодательное регулирование условий труда за компьютером становится необходимым элементом в обеспечении безопасности производственного процесса.

В соответствии с типовой отраслевой инструкцией ТОИ Р-45-084-01, принятой Минсвязи РФ, весь спектр деятельности, связанный с использованием ЭВМ, классифицируется на три группы. Каждая из этих групп имеет свою специфику и, соответственно, разный режим труда и отдыха. Так, для работников, выполняющих чтение информации с экрана (группа А), устанавливаются два обязательных перерыва продолжительностью 15 минут помимо основного обеденного времени. Деятельность по вводу текстовой информации (группа Б), требующая высокой точности и внимания, сопровождается более частыми, но краткими перерывами – не менее 10 минут после каждого часа работы. Творческая и проектная деятельность (группа В), связанная с анализом, синтезом информации и принятием решений, требует полноценного отдыха через каждый рабочий час – по 15 минут, что обусловлено интенсивной умственной нагрузкой и риском переутомления.

Следует отметить, что организация регламентированных перерывов – это не формальность, а действенная мера по предупреждению развития профессиональных заболеваний. Важно не просто отводить время на отдых, но

и использовать его рационально: включение в режим дня простых упражнений для глаз, коротких комплексов для снятия мышечного напряжения шеи и спины позволяет существенно снизить утомляемость, нормализовать кровообращение и предотвратить развитие туннельных синдромов и нарушений осанки. Такие меры особенно актуальны при длительном непрерывном взаимодействии с экраном монитора, что характерно для большинства офисных и инженерных профессий.

Однако даже строгое соблюдение режима труда и отдыха не даёт должного эффекта без надлежащего технического оснащения. Поэтому применение исключительно сертифицированного оборудования с подтверждёнными санитарно-гигиеническими характеристиками – основа безопасной работы за компьютером. Эргономичность рабочего места, корректная настройка параметров изображения, наличие регулируемого кресла и освещения, соответствующего нормам – всё это снижает нагрузку на зрение и опорно-двигательный аппарат. Но сертификация техники сама по себе не отменяет необходимости в регулярной проверке её исправности. В процессе эксплуатации оборудование может терять заявленные характеристики, поэтому профилактические осмотры, проверка кабелей, мониторов, клавиатур и блоков питания должны проводиться в установленные сроки, с обязательной фиксацией результатов.

Комплексный подход к охране труда в условиях взаимодействия с ПЭВМ должен включать в себя как технические, так и организационные решения. Только при сочетании сертифицированного оборудования, чёткого соблюдения режима труда и отдыха, а также осведомлённости сотрудников о правилах безопасной работы можно говорить о снижении профессиональных рисков и сохранении здоровья специалистов, занятых в сфере информационных технологий и других отраслях, связанных с длительной работой за компьютером.

В заключение следует подчеркнуть, что эффективная организация труда при работе с ПЭВМ – это не просто формальное соблюдение нормативных

требований, а важнейшая составляющая общего качества рабочего процесса и профессионального долголетия сотрудников. Пренебрежение установленными правилами может привести к накоплению хронической усталости, снижению производительности, а в перспективе – к развитию заболеваний, связанных с нарушением зрения, опорно-двигательной системы и нервной регуляции.

Именно поэтому принципы охраны труда должны внедряться не только на этапе проектирования рабочих мест, но и в повседневную практику предприятий и организаций. Регулярные инструктажи, чёткое соблюдение графика отдыха, постоянный мониторинг состояния оборудования и формирование осознанного отношения сотрудников к собственному здоровью – всё это формирует безопасную и устойчивую производственную среду.

Таким образом, реализация мероприятий по охране труда при работе с ПЭВМ позволяет не только снизить профессиональные риски, но и повысить общую эффективность труда, создать комфортные условия для выполнения интеллектуально напряжённой и ответственной работы.

Заключение

В ходе выполнения дипломного проекта был разработан программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами. В рамках работы были формализованы и промоделированы алгоритмы управления параллельными процессами, включая задачи «производители-потребители», «читатели-писатели» и «спящий парикмахер». Для каждого алгоритма были созданы модели недетерминированных автоматов и выполнена их реализация на языке VHDL. Проведённое тестирование подтвердило корректность и эффективность разработанных решений, что подтверждает достижение поставленных целей исследования.

Программный комплекс реализован в соответствии с техническим заданием.

В разделе экономики была произведена оценка затрат на разработку и внедрение приложения.

В разделе безопасность жизнедеятельности были проанализированы и описаны общие принципы обеспечения безопасности на рабочем месте оператора программного приложения контроля выполнения задач.

Задание на дипломное проектирование выполнено в полном объеме.

Список использованных источников

1. Вашкевич Н.П., Бикташев Р.А. – «Недетерминированные автоматы и их использование для реализации систем параллельной обработки информации». – Монография. – 2016. – 393с.

2. ГОСТ 12.0.0003-74 ССБТ. Библиографическая запись. Опасные и вредные производственные факторы. Классификация (с Изменением № 1). Изд-во стандартов, 2002. – 5 с.

3. СанПиН 2.2.3670-20. Санитарно-эпидемиологические требования к условиям труда. Изд-во стандартов, 2012. – 47с.

4. Типовая инструкция по охране труда при работе на персональном компьютере. – 2001. – 3 с.

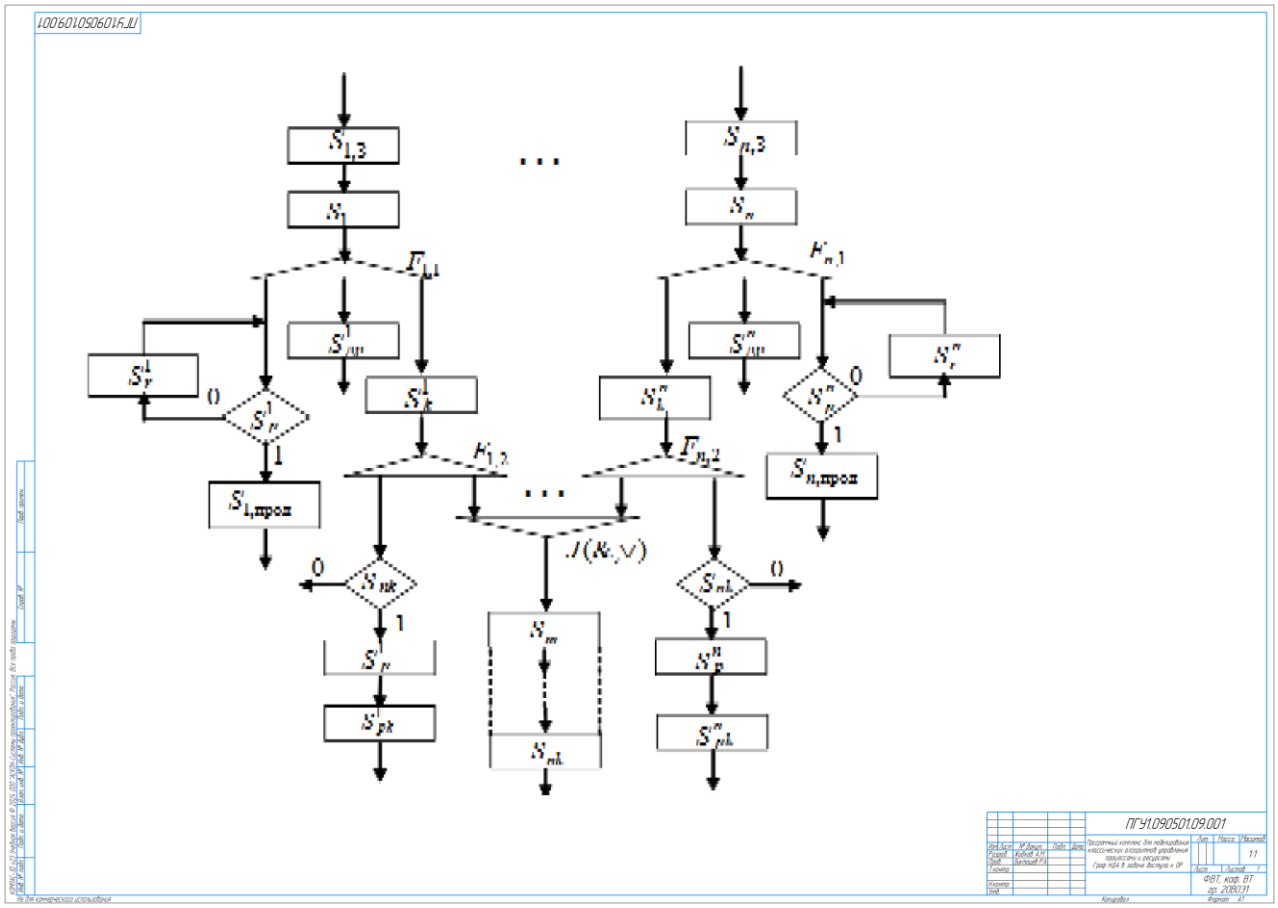
Приложение А

(обязательное)

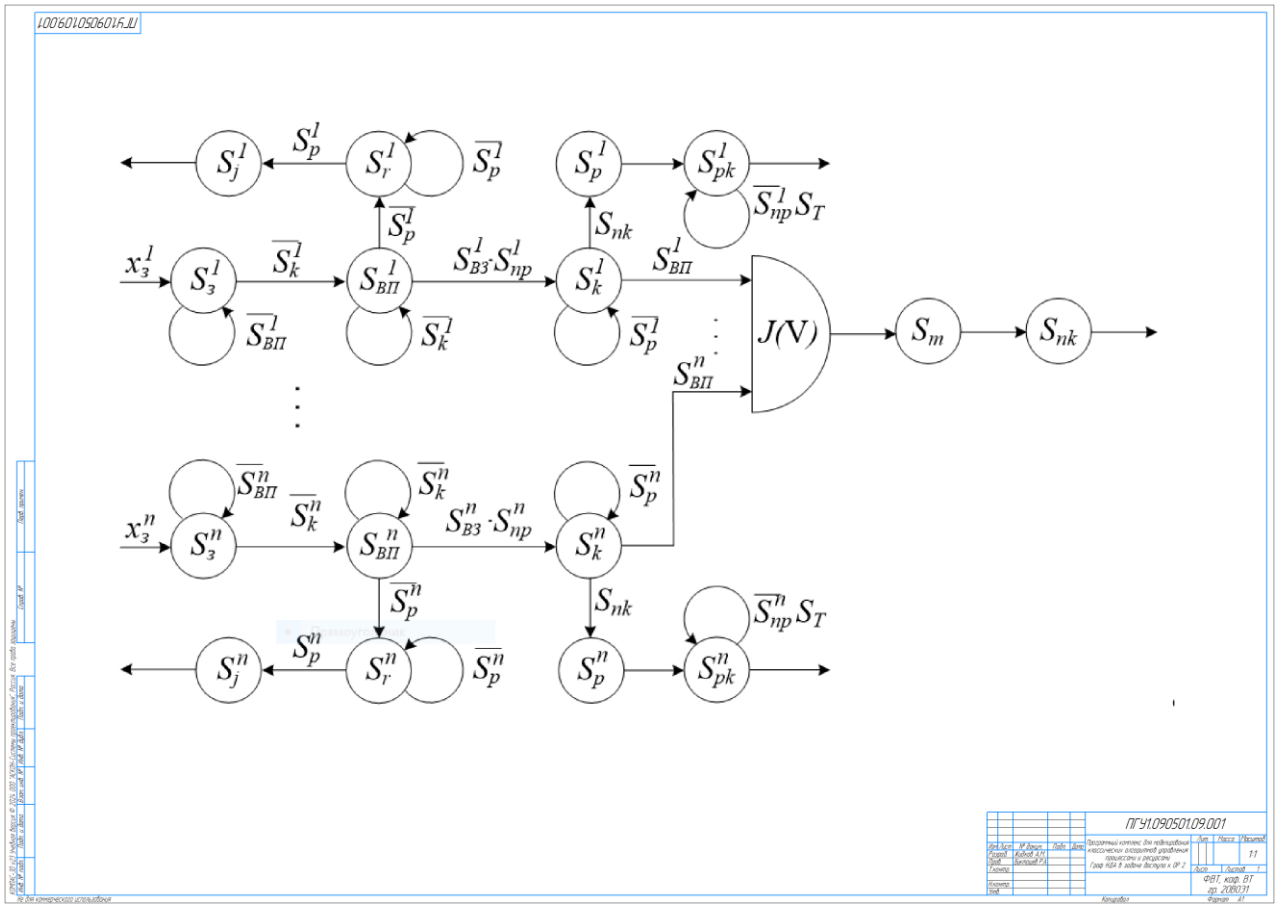
Программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами

Графы НДА

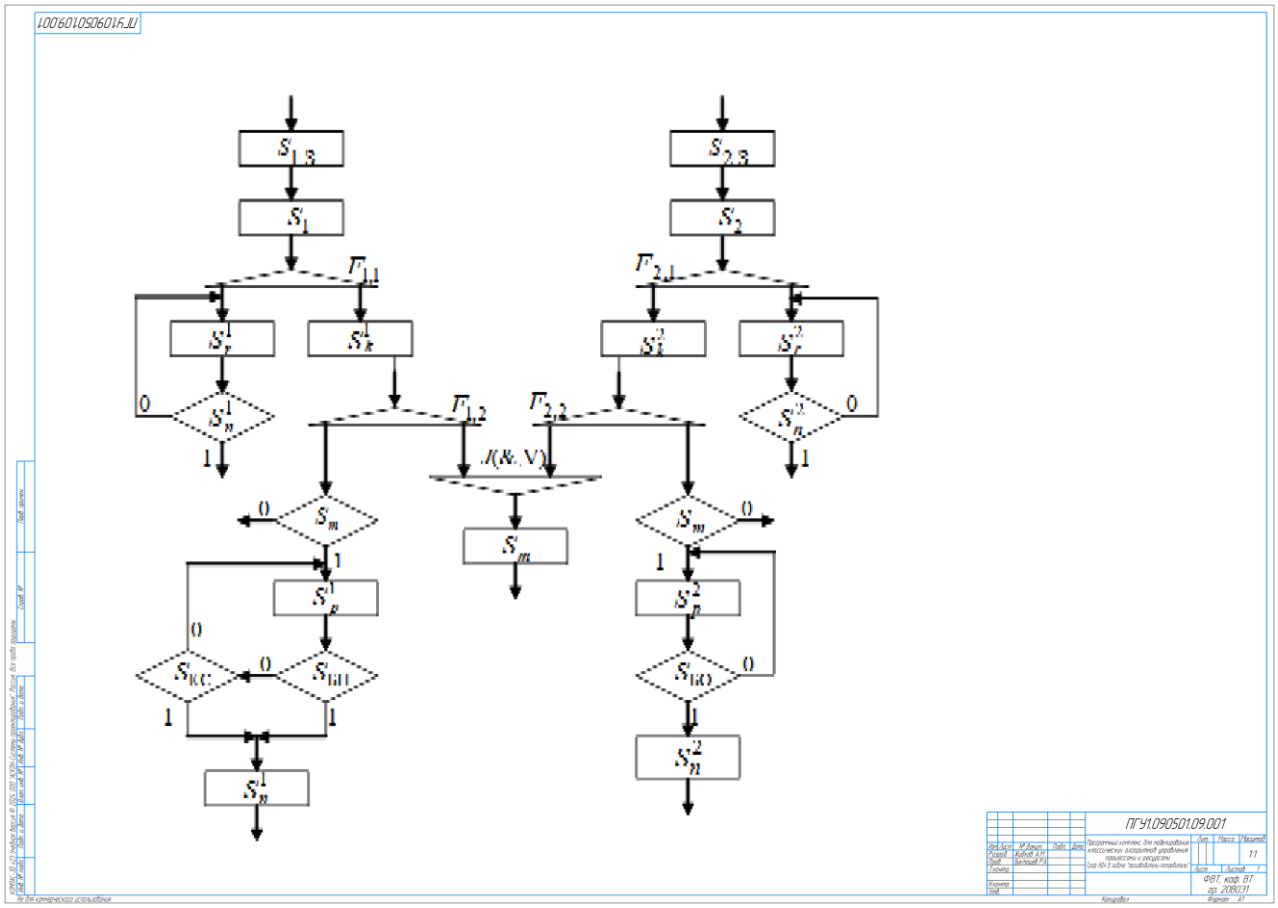
Приложение А.1 – Фрагмент графа НДА алгоритма управления взаимодействующими параллельными процессами при обращении к критическому ресурсу



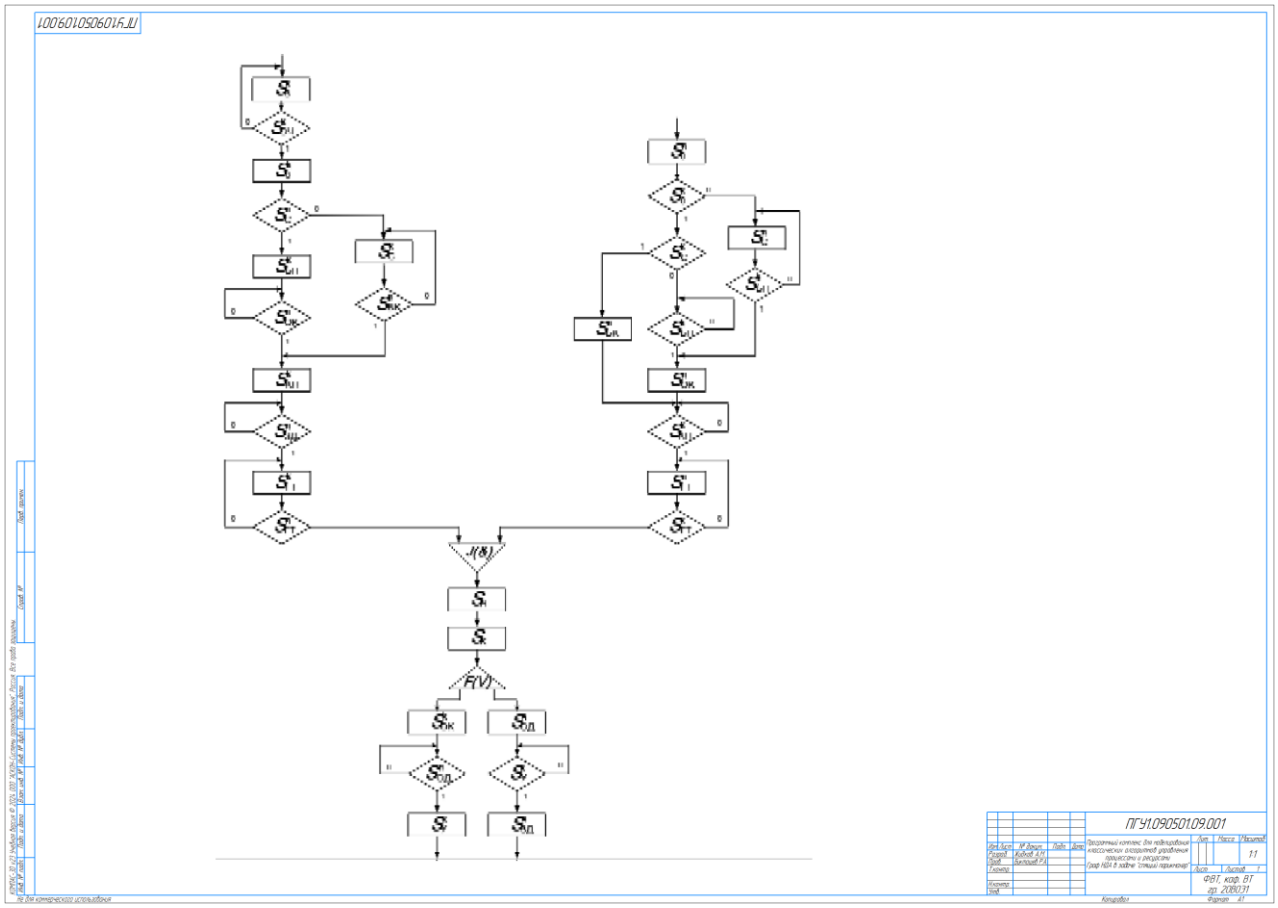
Приложение А.2 – Фрагмент графа НДА алгоритма управления взаимодействующими параллельными процессами при обращении к критическому ресурсу (расширенный)



Приложение А.3 – Фрагмент графа НДА алгоритма управления взаимодействующими параллельными процессами в задаче «производители-потребители»



Приложение А.4 – Граф НДА алгоритма управления взаимодействующими параллельными процессами в задаче «спящий парикмахер»



Приложение Б

(обязательное)

Программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами

VHDL-код

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity barber is
  port (
    clk : in std_logic;
    Skz : in std_logic_vector(4 downto 0);
    Skyo : out std_logic_vector(4 downto 0);

  Sp0o, Spso : out std_logic;
  Spbko, Spoko, Spgto, Spodo, Spko : out std_logic;
  Spzdo : out std_logic;
  Skocho : out std_logic_vector(4 downto 0);
  Sk0o, Skso, Skbpo, Skkpo, Skgto, Skno, Skoko : out
std_logic_vector(4 downto 0)
  );
end entity;

architecture beh of barber is
  signal Sp0 : std_logic := '1';
  signal Sps, Spbk, Spok, Spgt, Spod, Spk : std_logic := '0';
  signal Spzd : std_logic := '1';
  signal Skoch : std_logic_vector(4 downto 0) := "00000";
  signal Sk0, Sks, Skbp, Skkp, Skgt, Skn, Skok, Sky :
std_logic_vector(4 downto 0) := "00000";
begin
    process(clk)
        variable k : integer;
```

```
        variable barber_busy : std_logic := '0';
    begin
        if rising_edge(clk) then
            Sp0o <= Sp0;
            Spso <= Sps;
            Spbko <= Spbk;
            Spoko <= Spok;
            Spgto <= Spgt;
            Spodo <= Spod;
            Spko <= Spk;
            Spzdo <= Spzd;
            Skocho <= Skoch;
            Sk0o <= Sk0;
            Skso <= Sks;
            Skbpo <= Skbp;
            Skkpo <= Skkp;
            Skgto <= Skgt;
            Skno <= Skn;
            Skoko <= Skok;

            Skyo <= Sky;

            if (barber_busy = '0') and (Skz(0) = '1') then
                barber_busy := '1';
                Skoch(0) <= '1';
            elsif (barber_busy = '0') and (Skz(1) = '1') then
                barber_busy := '1';
                Skoch(1) <= '1';
```

```
            elsif (barber_busy = '0') and (Skz(2) = '1') then
                barber_busy := '1';
                Skoch(2) <= '1';
            elsif (barber_busy = '0') and (Skz(3) = '1') then
                barber_busy := '1';
                Skoch(3) <= '1';
            elsif (barber_busy = '0') and (Skz(4) = '1') then
                barber_busy := '1';
                Skoch(4) <= '1';
            end if;

            k := 0;
            while k < 5 loop
                if Sky(k) = '1' or ((Skz(k) = '0') and (Sky(k) =
'0')) then
                    Sky(k) <= '0';
                    Skok(k) <= '0';
                    Skn(k) <= '0';
                    Skgt(k) <= '0';
                    Skkp(k) <= '0';
                    Skbp(k) <= '0';
                    Sks(k) <= '0';
                    Sk0(k) <= '0';
                    Skoch(k) <= '0';

                    if Sky(k) = '1' then
                        barber_busy := '0';
                    end if;
```

```
                end if;


                if (Skoch(k) = '1') then
                    barber_busy := '1';


                    Sk0(k) <= Skz(k) and Skoch(k);
                    -- Sks(k) <= (Sk0(k) and (not Sps)) or (Sks(k)
and (not Spbk));
                    -- Sks(k) <= '0';
                    Skbp(k) <= Sk0(k) and Sps;
                    Skkp(k) <= (Sks(k) and Spbk) or (Skbp(k) and
Spok);
                    Skgt(k) <= (Skkp(k) and Spgt);
                    --          (Skkp(k) and Spzd) or (Skgt(k) and
Spgt) or , !!!!!!!!!!!!!!!


                    Skn(k) <= Skgt(k) and Spgt;
                    Skok(k) <= Skgt(k) and Spgt;
                    Sky(k) <= Skok(k) and Spod;
                -- elsif (Skoch(k) = '0') and (Skz(k) = '1') then
                    -- Sks(k) <= '1';
                end if;


                --Skoch(k) <= not (Spod and Sky(k)); -- !!
                --Sp0 <= Spzd; -- !!
            k := k+1;
            end loop;


            -- Sps <= ((Sp0 and (not Sk0(0))) or (Sp0 and
Skbp(0))) or ((Sp0 and (not Sk0(1))) or (Sp0 and Skbp(1))) or
```

```
((Sp0 and (not Sk0(2))) or (Sp0 and Skbp(2))) or ((Sp0 and (not
Sk0(3))) or (Sp0 and Skbp(3))) or ((Sp0 and (not Sk0(4))) or (Sp0
and Skbp(4)));

            Sps <= not barber_busy;

            Spbk <= (Sp0 and Sk0(0) and Sks(0)) or (Sp0 and Sk0(1)
and Sks(1)) or (Sp0 and Sk0(2) and Sks(2)) or (Sp0 and Sk0(3) and
Sks(3)) or (Sp0 and Sk0(4) and Sks(4));

            Spok <= ((((Sp0 and Sk0(0))) or Sps) and Skbp(0)) or
((((Sp0 and Sk0(1))) or Sps) and Skbp(1)) or ((((Sp0 and Sk0(2)))
or Sps) and Skbp(2)) or ((((Sp0 and Sk0(3))) or Sps) and Skbp(3))
or ((((Sp0 and Sk0(4))) or Sps) and Skbp(4));

            --                          and (not Sk0(k))

            Spgt <= ((Spok or Spbk) and Skkp(0)) or ((Spok or
Spbk) and Skkp(1)) or ((Spok or Spbk) and Skkp(2)) or ((Spok or
Spbk) and Skkp(3)) or ((Spok or Spbk) and Skkp(4));



            Spk <= Skn(0) or Skn(1) or Skn(2) or Skn(3) or Skn(4);

            Spod <= Spk;

            Spzd <= (Spod and Sky(0)) or (Spod and Sky(1)) or
(Spod and Sky(2)) or (Spod and Sky(3)) or (Spod and Sky(4));

        end if;

    end process;

end architecture;




library ieee;

use ieee.std_logic_1164.all;

use ieee.numeric_std.all;


entity tb_barber is

end entity;
```

```
architecture test of tb_barber is
  signal clk : std_logic := '0';
  signal Skz : std_logic_vector(4 downto 0);
  signal Skyo : std_logic_vector(4 downto 0);

  signal Sp0o, Spso : std_logic;
  signal Spbko, Spoko, Spgto, Spodo, Spko : std_logic;
  signal Spzdo : std_logic;
  signal Skocho : std_logic_vector(4 downto 0);
  signal Sk0o, Skso, Skbpo, Skkpo, Skgto, Skno, Skoko :
std_logic_vector(4 downto 0);

  constant clk_period : time := 20 ns;

begin
  UUT: entity work.barber
    port map (
      clk => clk,
      Skz => Skz,

      Sp0o => Sp0o,
      Spso => Spso,
      Spbko => Spbko,
      Spoko => Spoko,
      Spgto => Spgto,
      Spodo => Spodo,
      Spko => Spko,
      Spzdo => Spzdo,
```

```
      Skocho => Skocho,
      Sk0o => Sk0o,
      Skso => Skso,
      Skbpo => Skbpo,
      Skkpo => Skkpo,
      Skgto => Skgto,
      Skno => Skno,
      Skoko => Skoko,

      Skyo => Skyo
    );

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

-- Stimulus process
stim_proc: process
begin
  Skz <= "00000";
  wait for 100 ns;
```

```
Skz <= "10000";
wait for 240 ns;
Skz <= "00000";


wait for 150 ns;


Skz <= "00100";
wait for 240 ns;
Skz <= "00000";


wait for 150 ns;


Skz <= "00101";
wait for 240 ns;
Skz <= "00000";

Skz <= "00100";
wait for 240 ns;
Skz <= "00000";


wait for 150 ns;
```

```
        Skz <= "00100";
        wait for 130 ns;
        Skz <= "00000";


        wait for 300 ns;


        Skz <= "11000";
        wait for 240 ns;
        Skz <= "00000";

        Skz <= "10110";
        wait for 80 ns;
        Skz <= "00000";

        Skz <= "00100";
        wait for 240 ns;
        Skz <= "00000";

        Skz <= "10000";
        wait for 240 ns;
        Skz <= "00000";

        wait for 150 ns;

    --    wait for 5000 ns;
```

```
  end process;


end architecture;



library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity sync_device is
    Port (
        clk, x0 : in std_logic;
        xz, snk : in std_logic_vector(4 downto 0);
        szo, sjo, svpo, sko, sro, spo, spko ,svzo,spro : out
std_logic_vector(4 downto 0);
        smo, snko, sto ,s0o: out std_logic);
end sync_device;


architecture Behavioral of sync_device is
    signal ssm,ssnk,st,s0 : std_logic:='0';
    signal spr, svz, sz, sj, svp, sk, sr, sp, spk :
std_logic_vector(4 downto 0):="00000";
begin
    process (CLK)
    variable i:integer;
    begin
        if CLK='1' and CLK'event then
            i := 0;
```

```
            while (i<5) 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));


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


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


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


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

            ----------------------


            st<=

            (spk(0) and (not svp(1)) and (not svp(2)) and (not
svp(3)) and (not svp(4))) or

            (spk(1) and (not svp(2)) and (not svp(3)) and (not
svp(4))) or

            (spk(2) and (not svp(3)) and (not svp(4))) or

            (spk(3) and (not svp(4))) or

            (spk(4));


            s0<= x0 or st;


            spr(0)<= (sz(0) and s0)

            or (svp(0) and ((spk(4))

            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(2)) and (not svp(3)) and (not
svp(4)))

            or (spk(2) and (not svp(3)) and (not svp(4)))

            or (spk(3) and (not svp(4)))));

            -----

            spr(1)<= (sz(1) and (not sz(0)) and s0)

            or (svp(1) and ((spk(0))

            or (spk(1) and (not svp(0)) and (not svp(2)) and (not
svp(3)) and (not svp(4)))

            or (spk(2) and (not svp(3)) and (not svp(4)) and (not
svp(0)))

            or (spk(3) and (not svp(4)) and (not svp(0)))
```

```
            or (spk(4) and (not svp(0)))));

            -----

            spr(2)<= (sz(2) and (not sz(0))and (not sz(1)) and s0)

            or (svp(2) and ((spk(1))

            or (spk(2) and (not svp(0)) and (not svp(1)) and (not
svp(3)) and (not svp(4)))

            or (spk(3) and (not svp(1)) and (not svp(4)) and (not
svp(0)))

            or (spk(4) and (not svp(1)) and (not svp(0)))

            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)))

            or (spk(4) 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)))

            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))));

            ------
```

```
        end if;
    end process;


    process (CLK)
    variable i:integer;
    begin
        if CLK='1' and CLK'event then
            i := 0;
            while ( i<5) 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;
```

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;


ENTITY sync_device_tb IS

END sync_device_tb;


ARCHITECTURE behavior OF sync_device_tb IS

    COMPONENT sync_device

    PORT( clk : IN std_logic;

    x0 : IN std_logic; -- (x_0) сигнал приведения системы
управления в начальное состояние;

    xz : IN std_logic_vector(4 downto 0); -- (xi_z,i) сигнал
запроса

    snk : IN std_logic_vector(4 downto 0); -- (S_nk) событие,
являющееся заключительным в процедуре обращения к разделяемым
данным;

    szo : OUT std_logic_vector(4 downto 0); -- (Si_z) сокращенное
обозначение события, определяющего наличие заявки i-го процесса и
их обслуживание.

    sjo : OUT std_logic_vector(4 downto 0); -- (Si_j) событие,
инициирующее продолжение работы i-го процесса после выхода из
критического интервала.

    svpo : OUT std_logic_vector(4 downto 0); -- (Si_vp)
сокращенное обозначение события, определяющего восприятие заявки
i-го процесса на обслуживание;

    sko : OUT std_logic_vector(4 downto 0); -- (Si_k) сокращенное
обозначение события, определяющего вход и нахождение i-го
процесса в своем критическом интервале;

    sro : OUT std_logic_vector(4 downto 0); -- (Si_r) событие,
символизирующее ожидание условий выхода i-го процесса из
критического участка

    spo : OUT std_logic_vector(4 downto 0); -- (Si_p) сокращенное
обозначение события, обеспечивающего выход i-го процесса из
```

```
критического интервала после окончания процедуры обращения к
разделяемым данным;

    spko : OUT std_logic_vector(4 downto 0); -- (Si_pk)
сокращенное обозначение события, определяемого как событие Si_p,
задержанное на один такт; это событие используется, в принятом
варианте, для организации циклической дисциплины обслуживания.

    svzo : OUT std_logic_vector(4 downto 0); -- (Si_vz)
комбинационное событие, обеспечивающее взаимоисключение
критических интервалов на основе несовместимости событий Si_k с
другими событиями из их общего числа, равного n:

    spro : OUT std_logic_vector(4 downto 0); -- (Si_pr)
сокращенное обозначение события, обеспечивающего заданное
приоритетное обслуживание i-го процесса.

    smo : OUT std_logic; -- (S_m) событие, обеспечивающее начало
процедуры реализации обращения к разделяемым данным;

    snko : OUT std_logic; -- (S_nk) событие, являющееся
заключительным в процедуре обращения к разделяемым данным;

    sto : OUT std_logic; -- (S_T) сокращенное обозначение
комбинационного события, свидетельствующего об окончании цикла
обслуживания процессов, от которых были восприняты заявки на
обслуживание

    s0o : OUT std_logic -- (S_0) начальное событие системы
управления перед обращением к разделяемому ресурсу;

    );

END COMPONENT;


    --Inputs

    signal clk : std_logic := '0';

    signal x0 : std_logic := '0';

    signal xz : std_logic_vector(4 downto 0) := (others => '0');

    signal snk : std_logic_vector(4 downto 0) := (others => '0');

    --Outputs

    signal szo : std_logic_vector(4 downto 0);

    signal sjo : std_logic_vector(4 downto 0);
```

```
    signal svpo : std_logic_vector(4 downto 0);
    signal sko : std_logic_vector(4 downto 0);
    signal sro : std_logic_vector(4 downto 0);
    signal spo : std_logic_vector(4 downto 0);
    signal spko : std_logic_vector(4 downto 0);
    signal svzo : std_logic_vector(4 downto 0);
    signal spro : std_logic_vector(4 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: sync_device 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)<='0';
    x0<='1';
    wait for 10 ns;

    xz(0)<='0';
    xz(1)<='0';
    xz(2)<='0';
    xz(3)<='0';
    xz(4)<='0';
    x0<='0';

    wait for 285 ns;



    xz(0)<='1';
    xz(1)<='1';
```

```
xz(2)<='1';
xz(3)<='0';
xz(4)<='0';
x0<='1';
wait for 10 ns;

xz(0)<='0';
xz(1)<='0';
xz(2)<='0';
xz(3)<='0';
xz(4)<='0';
x0<='0';

wait for 285 ns;



xz(0)<='1';
xz(1)<='0';
xz(2)<='1';
xz(3)<='0';
xz(4)<='0';
x0<='1';
wait for 10 ns;

xz(0)<='0';
xz(1)<='0';
xz(2)<='0';
```

```
xz(3)<='0';
xz(4)<='0';
x0<='0';

wait for 285 ns;



xz(0)<='1';
xz(1)<='1';
xz(2)<='0';
xz(3)<='1';
xz(4)<='1';
x0<='1';
wait for 10 ns;

xz(0)<='0';
xz(1)<='0';
xz(2)<='0';
xz(3)<='0';
xz(4)<='0';
x0<='0';

wait for 285 ns;



xz(0)<='1';
```

```
        xz(1)<='1';
        xz(2)<='0';
        xz(3)<='0';
        xz(4)<='1';
        x0<='1';
        wait for 10 ns;

        xz(0)<='0';
        xz(1)<='0';
        xz(2)<='0';
        xz(3)<='0';
        xz(4)<='0';
        x0<='0';

        wait for 285 ns;

        x0<='0';
        wait for clk_period*10;

        wait;
        --end loop;
    end process;

END;
```

```
LIBRARY ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--Entity Declaration
ENTITY prod_cons IS
PORT (
  clk       : in STD_LOGIC;
  xz1       : in STD_LOGIC;
  xz2       : in STD_LOGIC;
  portionl  : in unsigned (3 downto 0);
  sk1_o     : out STD_LOGIC;
  sk2_o     : out STD_LOGIC;
  sofBZ     : out STD_LOGIC;
  sofBcht   : out STD_LOGIC
);
END prod_cons;

-- Architecture Body
ARCHITECTURE Behavioral of prod_cons IS
  signal s1, s2           : STD_LOGIC := '0';
  signal sz1, sz2         : STD_LOGIC := '0';
  signal sk1, sk2         : STD_LOGIC := '0';
  signal spch, spz        : STD_LOGIC := '0';
  signal sn1, sn2         : STD_LOGIC := '0';
  signal sch1, sch2       : STD_LOGIC := '0';
  signal sj1, sj2         : STD_LOGIC := '0';
  signal sm               : STD_LOGIC := '0';
```

```
  signal sbp              : STD_LOGIC := '0';
  signal sb0              : STD_LOGIC := '1';
  signal skz, skch        : STD_LOGIC := '0';
  signal sp1, sp2         : STD_LOGIC := '0';
  signal spr1, spr2       : STD_LOGIC := '0';
  signal svz1, svz2       : STD_LOGIC := '0';
  signal schBz            : unsigned(4 downto 0) := "00001";
  signal SchBch           : unsigned(4 downto 0) := "00001";
  signal bufch, bufz      : unsigned(4 downto 0) := "00001";
  signal schz             : unsigned(3 downto 0) := "0000";
  signal schch            : unsigned(3 downto 0) := "1100";
BEGIN

  sk1_o <= sk1;
  sk2_o <= sk2;

  process(clk)
    variable portion : unsigned(3 downto 0) := "0000";
    variable size    : unsigned(3 downto 0) := "1100";
  begin
    if rising_edge(clk) then
      portion := portionl;
      sb0 <= '0';
      sbp <= '0';

      sz1 <= xz1;
      sz2 <= xz2;
```

```
        svz1 <= not sk2;
        svz2 <= not sk1;

        s1 <= (sz1 or s1) and (not sk1);
        s2 <= (sz2 or s2) and (not sk2);

        spr1 <= not sbp;
        spr2 <= ((not sb0) and (not s1)) or sbp;

        sk1 <= (s1 and svz1 and spr1) or (sk1 and (not sn1));
        sk2 <= (s2 and svz2 and spr2) or (sk2 and (not sn2));

        sch1 <= (s1 or sch1) and (not sp1);
        sch2 <= (s2 or sch2) and (not sp2);

        sm <= (sk1 and s1) or (sk2 and s2);

        spz <= (sm and sk1) or (spz and ((not sbp) and (not skz)));
        spch <= (sm and sk2) or (spch and ((not sb0) and (not
skch)));

        sn1 <= spz and (sbp or skz);
        sn2 <= spch and (sb0 or skch);

        sp1 <= sk1 and sn1;
        sp2 <= sk2 and sn2;

        sj1 <= sch1 and sp1;
```

```
sj2 <= sch2 and sp2;

skz  <= spz;
skch <= spch;

if sz1 = '1' then
  SchBz <= SchBz + portion;
  schz <= schz + portion;
  schch <= schch - portion;
end if;

if sz2 = '1' then
  SchBch <= SchBch + portion;
  schz <= schz - portion;
  schch <= schch + portion;
end if;

if SchBz > size then
  SchBz <= SchBz - size;
  sofBZ <= '1';
  sofBcht <= '0';
end if;

if SchBch > size then
  SchBch <= SchBch - size;
  sofBZ <= '0';
  sofBcht <= '1';
end if;
```

```
      if schz >= size then
        sbp <= '1';
        sb0 <= '0';
      end if;

      if schch >= size then
        sbp <= '0';
        sb0 <= '1';
      end if;
    end if;
  end process;

END Behavioral;

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity tb_prod_cons is
end entity;

architecture sim of tb_prod_cons is
  signal clk       : std_logic := '0';
  signal xz1       : std_logic := '0';  -- производитель
  signal xz2       : std_logic := '0';  -- потребитель
  signal portionl  : unsigned(3 downto 0) := (others => '0');
  signal sk1_o     : std_logic;
```

```
  signal sk2_o     : std_logic;
  signal sofBZ     : std_logic;
  signal sofBcht   : std_logic;

  constant clk_period : time := 10 ns;

begin

  -- Instantiate the DUT
  uut: entity work.prod_cons
    port map (
      clk       => clk,
      xz1       => xz1,
      xz2       => xz2,
      portionl  => portionl,
      sk1_o     => sk1_o,
      sk2_o     => sk2_o,
      sofBZ     => sofBZ,
      sofBcht   => sofBcht
    );

  -- Clock generator
  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
  -- начальная инициализация
  wait for 20 ns;

  -- Производитель кладёт порцию 3
  xz1 <= '1';
  portionl <= "0011";  -- 3
  wait for clk_period;
  xz1 <= '0';

  wait for 40 ns;

  -- Потребитель запрашивает порцию 2
  xz2 <= '1';
  portionl <= "0010";  -- 2
  wait for clk_period;
  xz2 <= '0';

  wait for 40 ns;

  -- Производитель снова кладёт 4
  xz1 <= '1';
  portionl <= "0100";  -- 4
  wait for clk_period;
  xz1 <= '0';
```

```
    wait for 30 ns;

    -- Потребитель запрашивает 5
    xz2 <= '1';
    portionl <= "0101";  -- 5
    wait for clk_period;
    xz2 <= '0';

    wait for 50 ns;

    -- Проверка переполнения
    xz1 <= '1';
    portionl <= "1111";  -- 15
    wait for clk_period;
    xz1 <= '0';

    wait for 100 ns;

    assert false report "Simulation complete" severity failure;
  end process;

end architecture;
```

Приложение В

(обязательное)

Программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами

Результаты испытаний

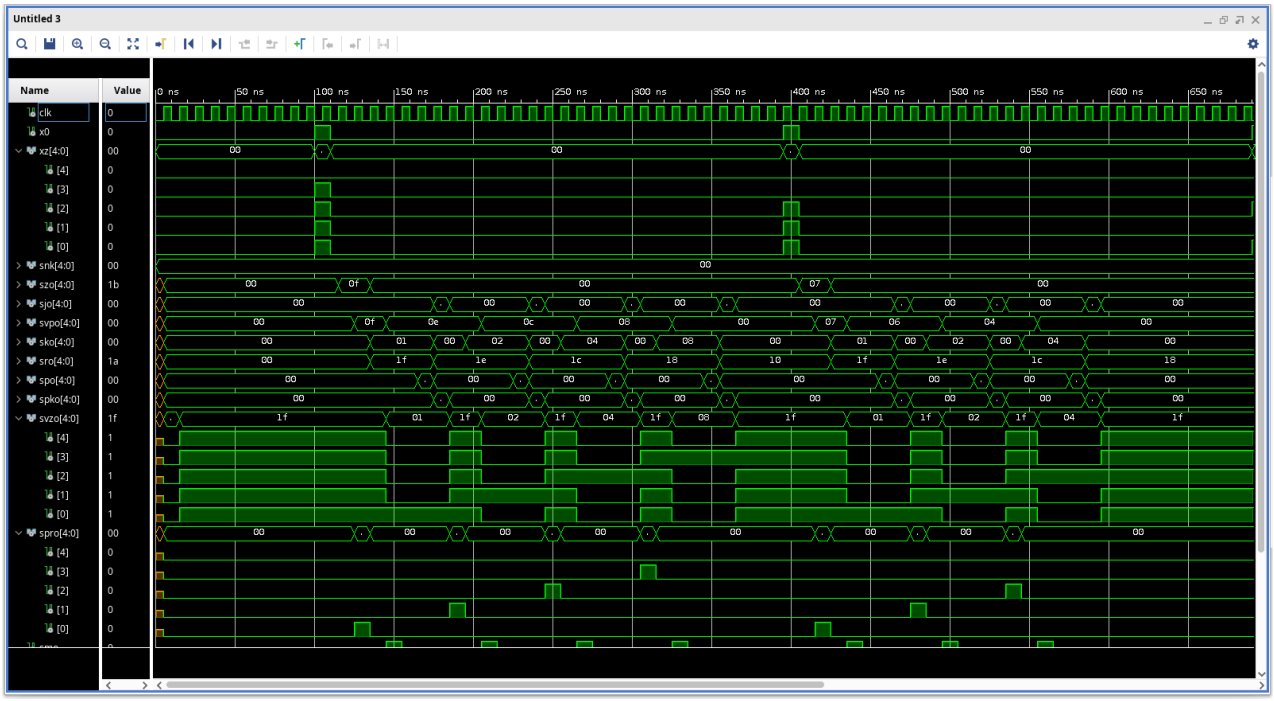


Рисунок В.1 – Результаты тестирования алгоритма управления доступом к критическому ресурсу

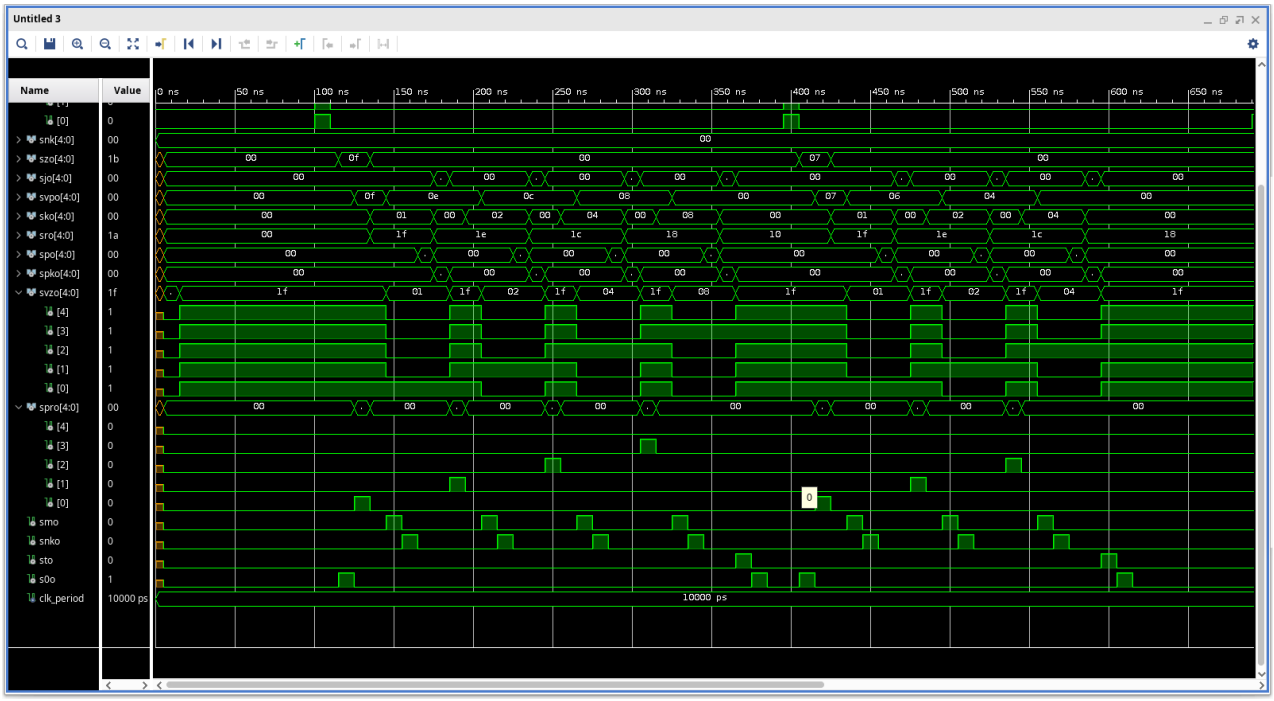


Рисунок В.2 – Результаты тестирования алгоритма управления доступом к критическому ресурсу

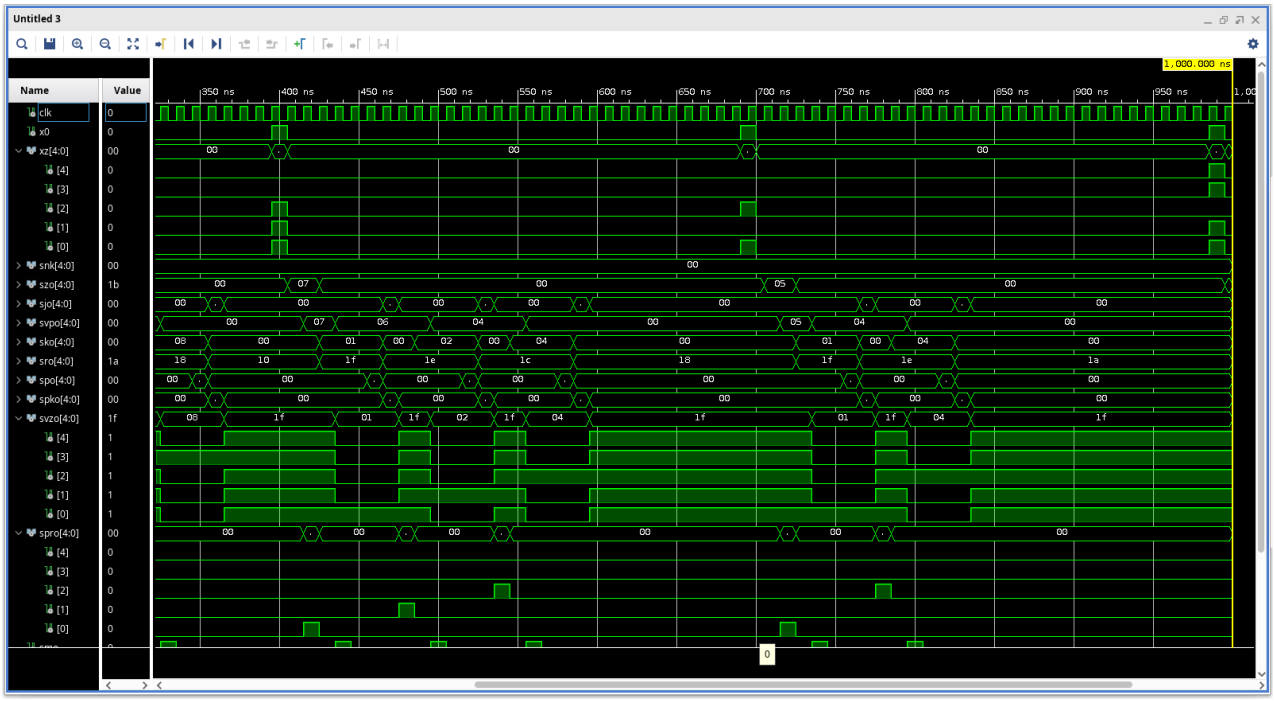


Рисунок В.3 – Результаты тестирования алгоритма управления доступом к критическому ресурсу

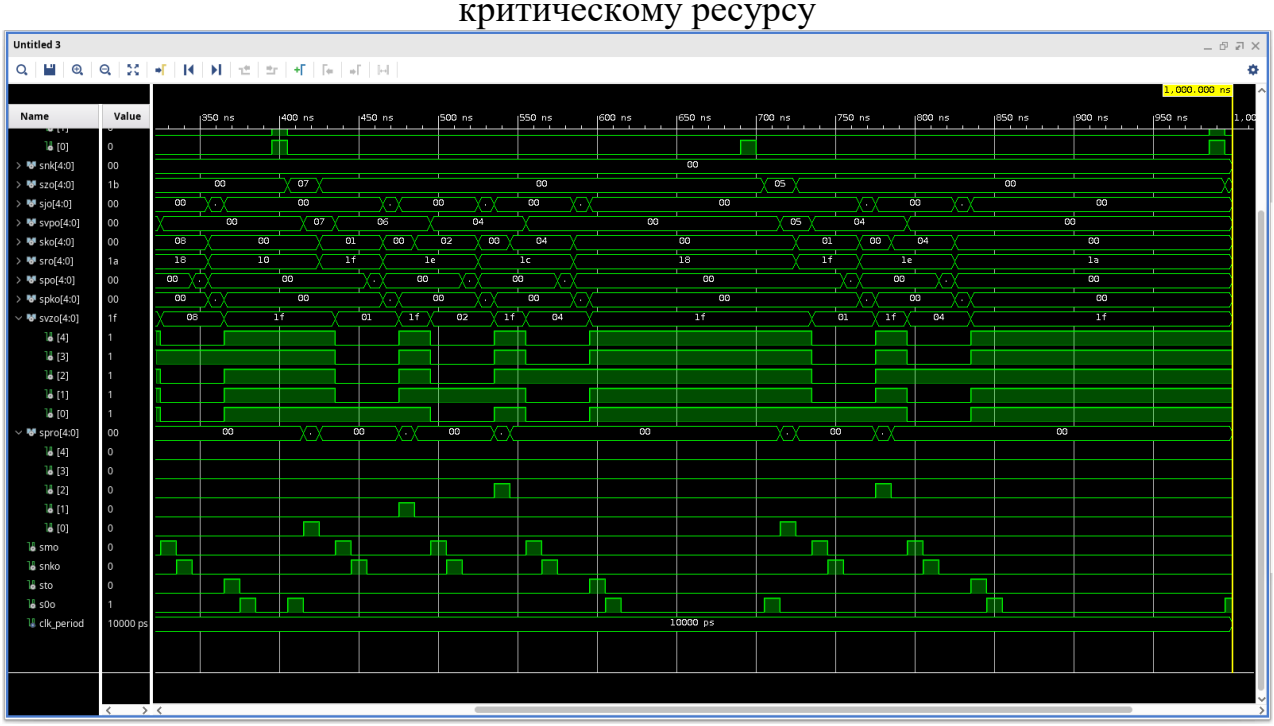


Рисунок В.4 – Результаты тестирования алгоритма управления доступом к критическому ресурсу

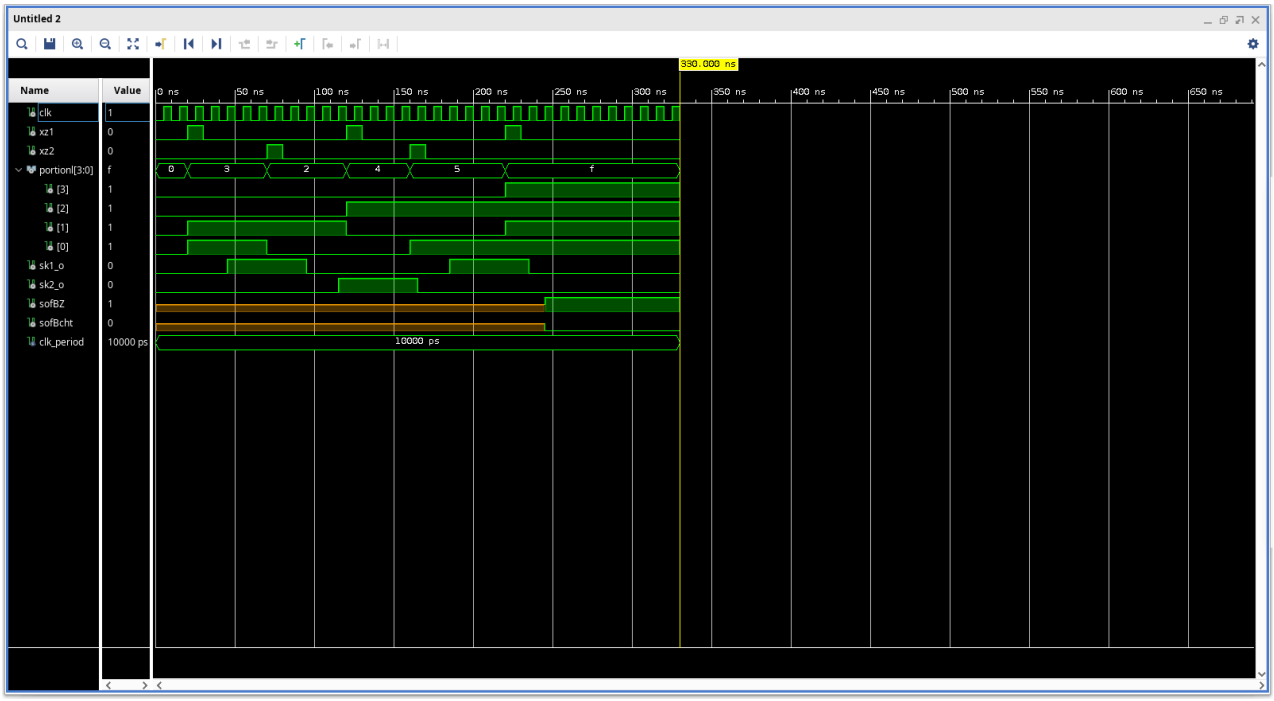


Рисунок В.5 – Результаты тестирования алгоритма управления в задаче «производители-потребители»

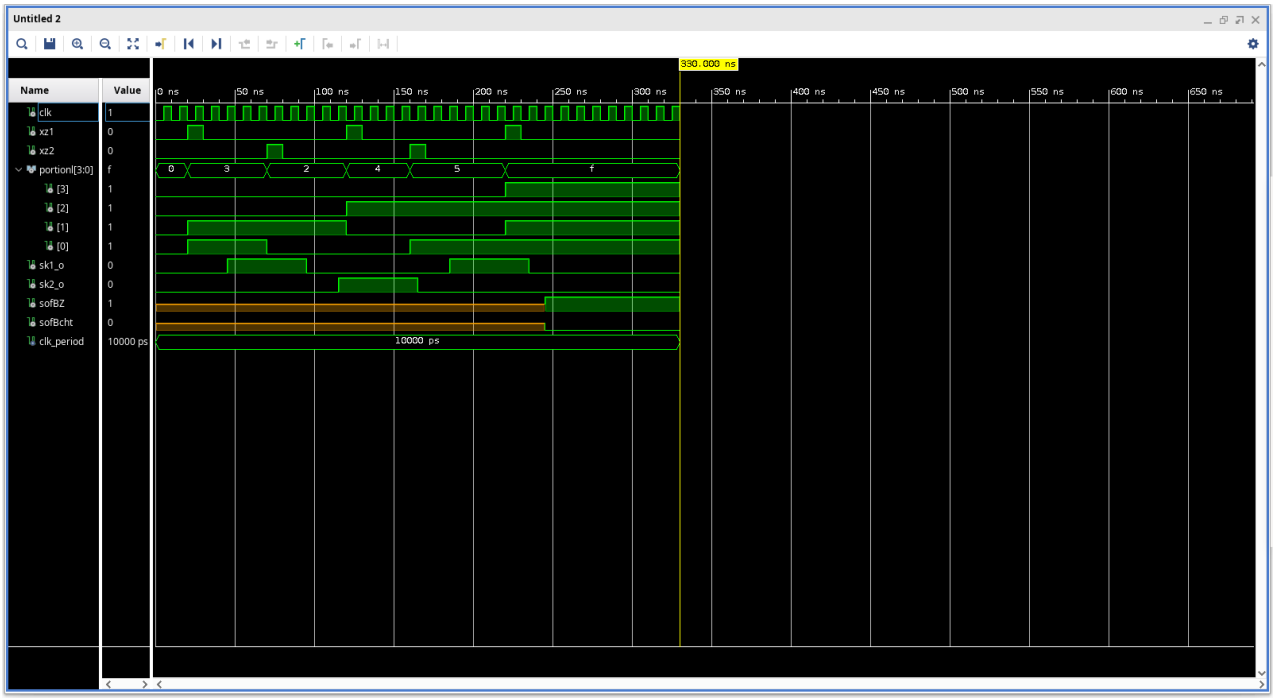


Рисунок В.6 – Результаты тестирования алгоритма управления в задаче «производители-потребители»

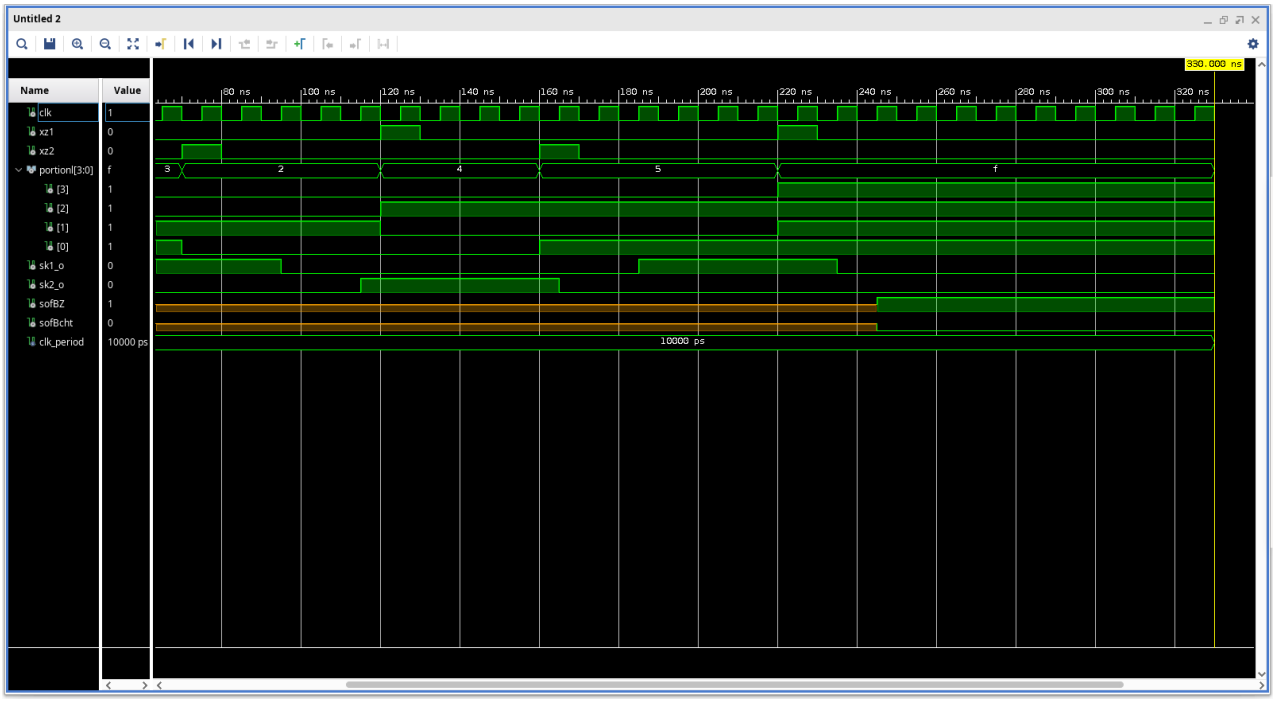


Рисунок В.7 – Результаты тестирования алгоритма управления в задаче «производители-потребители»

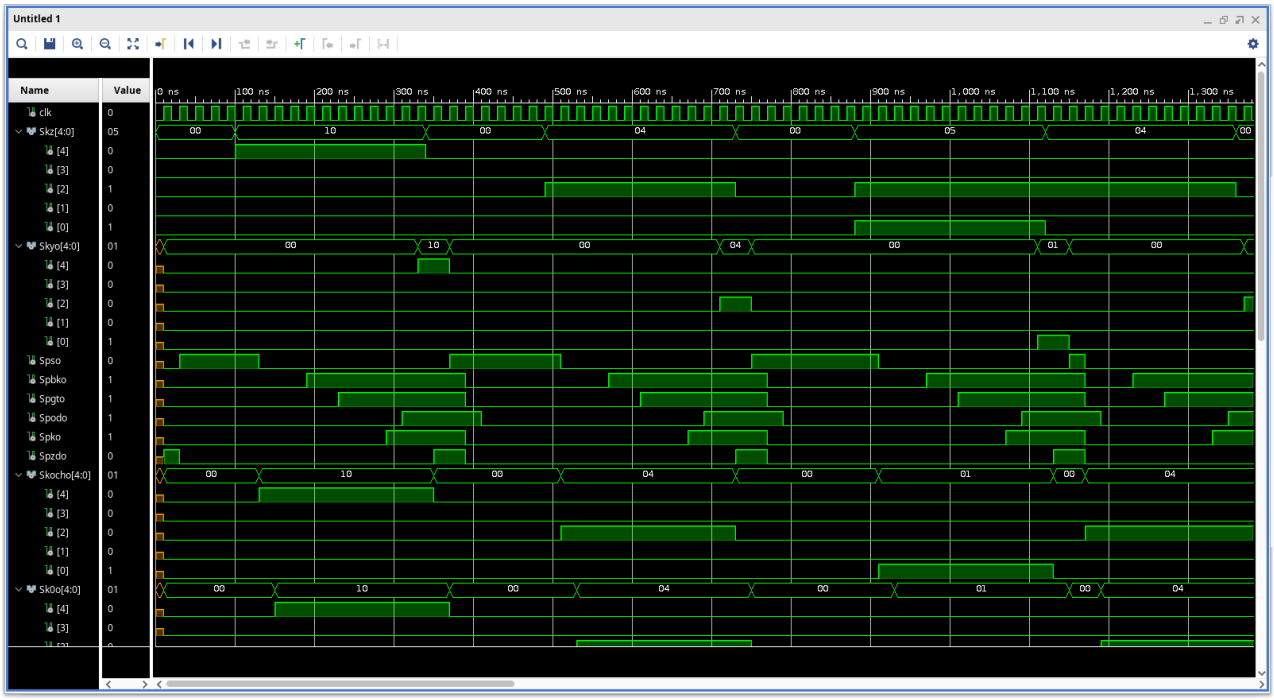


Рисунок В.8 – Результаты тестирования алгоритма управления в задаче «спящий парикмахер»

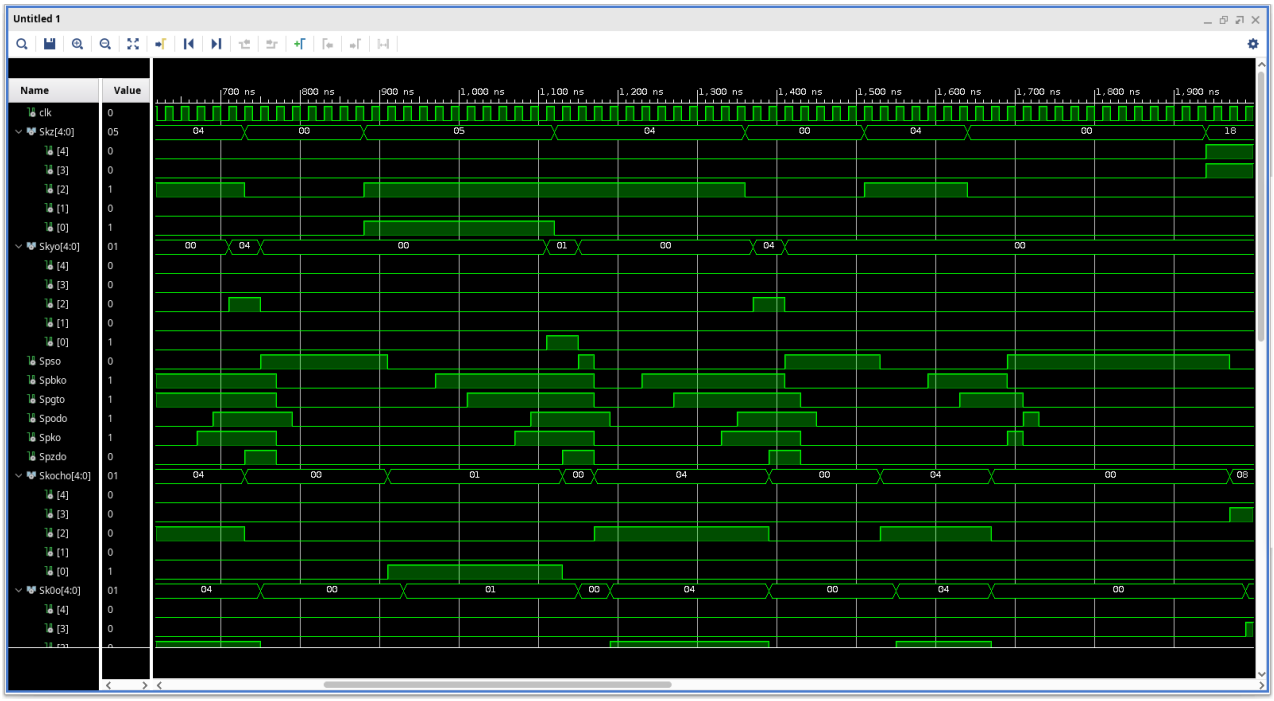


Рисунок В.9 – Результаты тестирования алгоритма управления в задаче «спящий парикмахер»

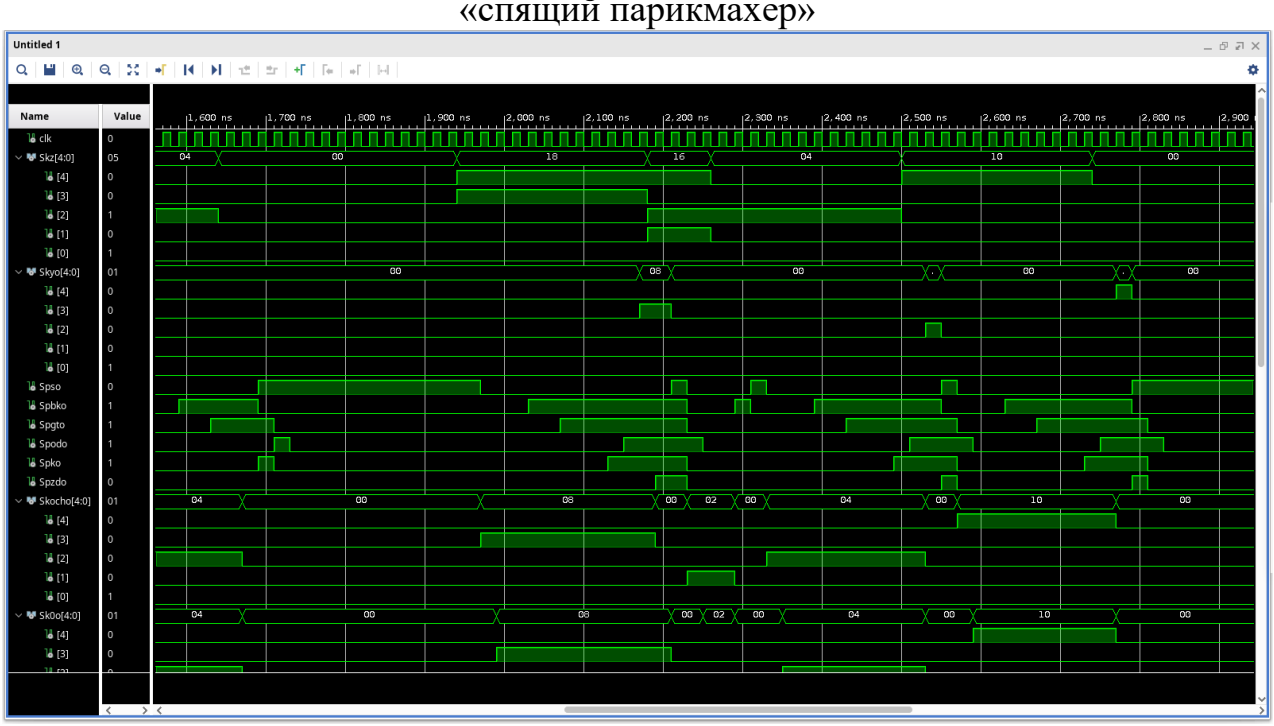


Рисунок В.10 – Результаты тестирования алгоритма управления в задаче «спящий парикмахер»

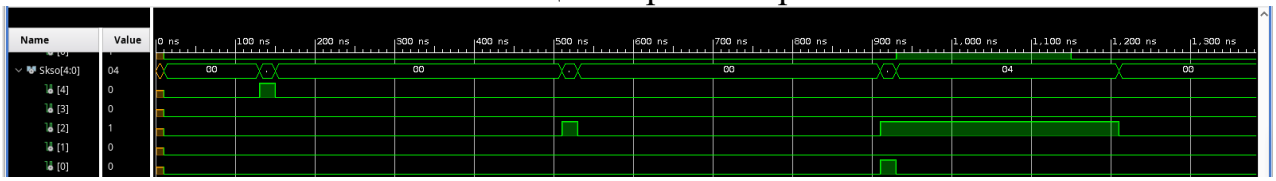


Рисунок В.11 – Результаты тестирования алгоритма управления в задаче «спящий парикмахер»

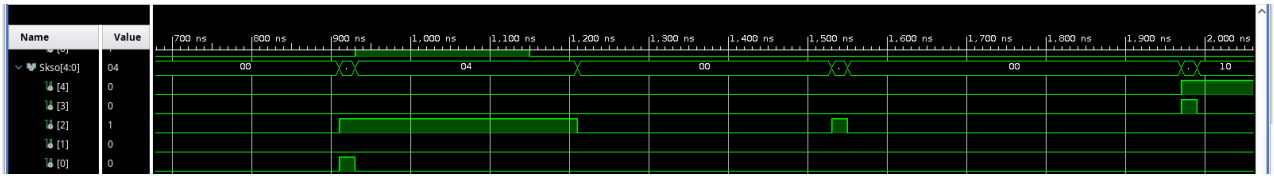


Рисунок В.12 – Результаты тестирования алгоритма управления в задаче «спящий парикмахер»

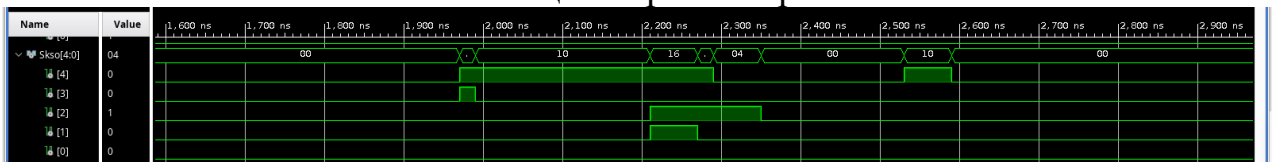


Рисунок В.13 – Результаты тестирования алгоритма управления в задаче «спящий парикмахер»

Приложение Г

(обязательное)

Программный комплекс для моделирования классических алгоритмов управления процессами и ресурсами

Презентация

Рисунок Г.1 – Презентация, страница 1

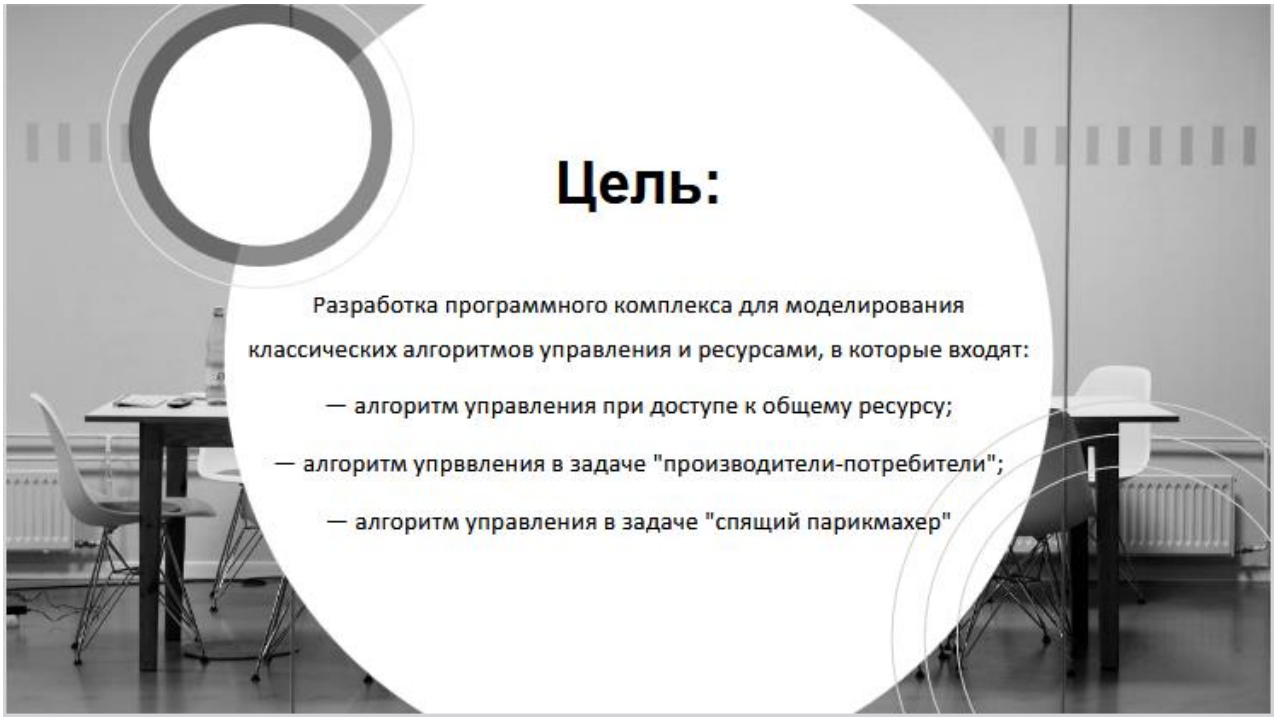


Рисунок Г.2 – Презентация, страница 2

Задача "Доступ к общему (критическому) ресурсу"

Общим ресурсом считается объект или участок памяти, способный обслуживать только один процесс за раз. Задача состоит в том, чтобы организовать такой порядок работы, при котором в любой момент времени не более одного процесса, находящегося в своей «критической секции» — фрагменте кода, где производится работа с этим ресурсом. Для этого вводится механизм синхронизации, который при поступлении заявки от процесса либо допускает его в критическую секцию (если ресурс свободен), либо переводит в состояние ожидания до тех пор, пока ресурс не освободится.

Рисунок Г.3 – Презентация, страница 3

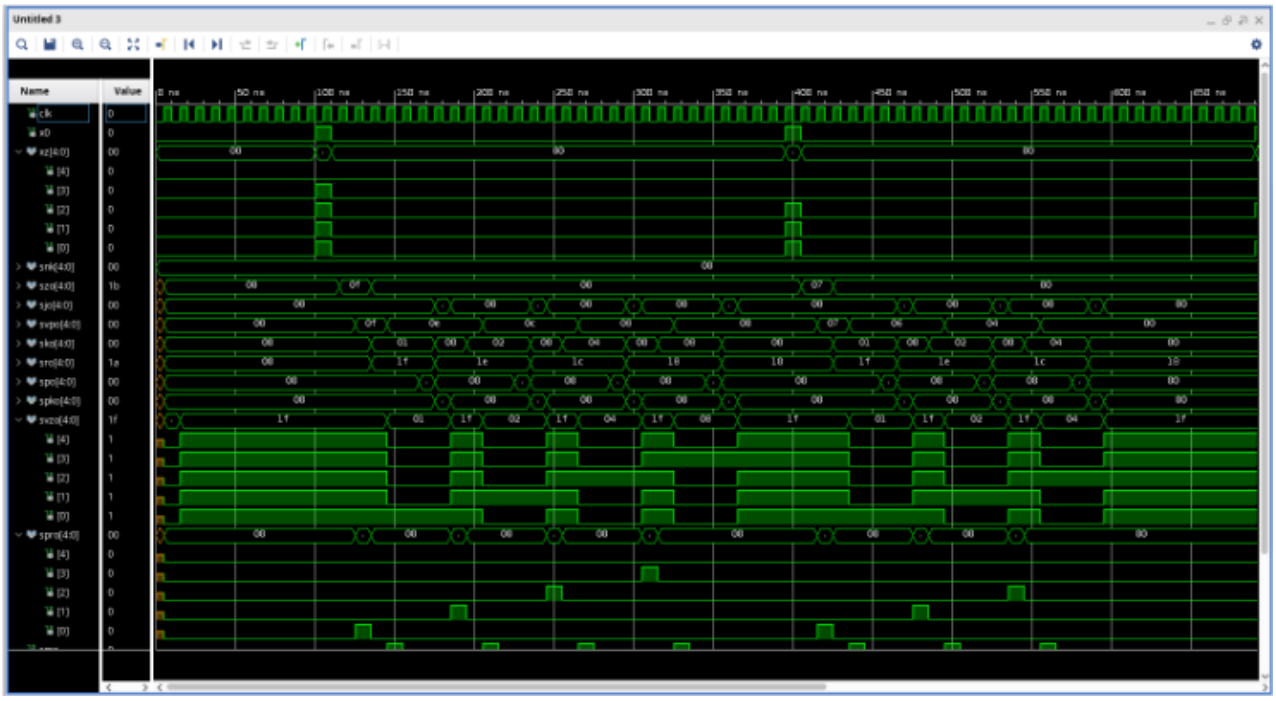


Рисунок Г.4 – Презентация, страница 4

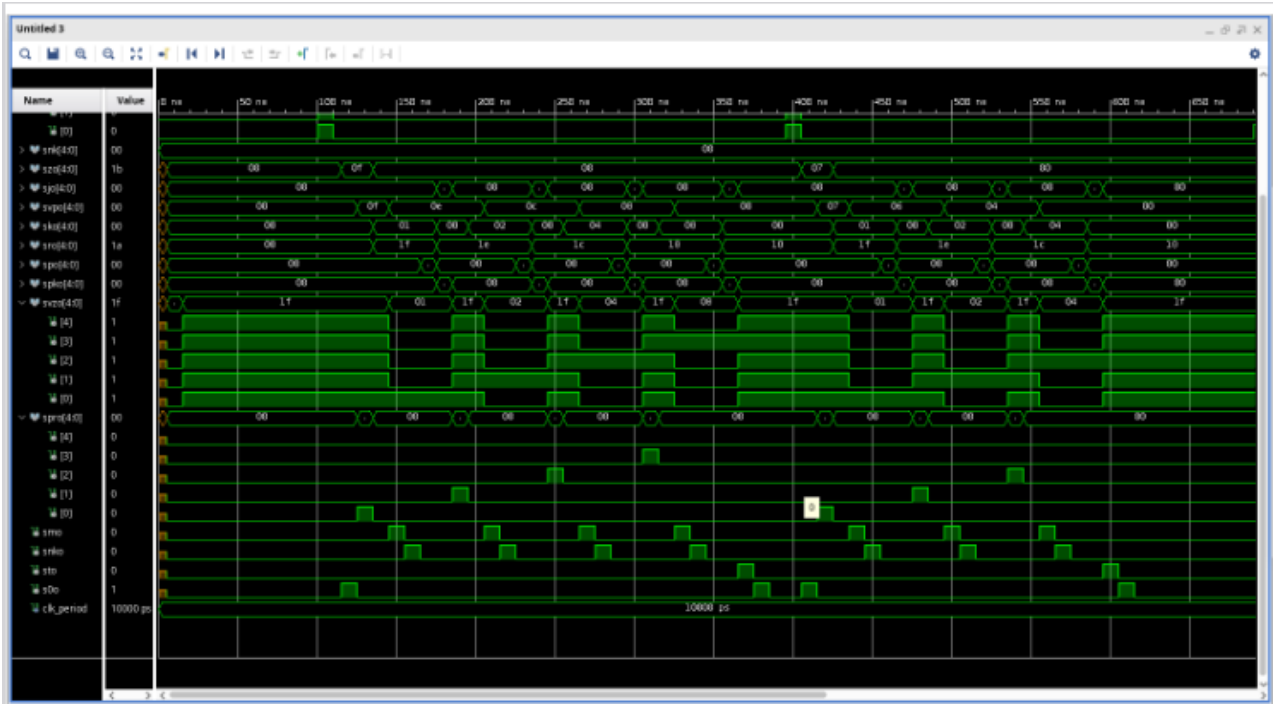


Рисунок Г.5 – Презентация, страница 5

Задача "производители-потребители"

Система состоит из двух видов процессов: «производители», формирующие новые данные, и «потребители», извлекающие их из **общего буфера** фиксированного размера. Буфер обычно реализуют как кольцевую очередь: производитель помещает элемент в конец, потребитель читает из начала. Синхронизация сводится к тому, чтобы предотвратить запись в уже полностью заполненный буфер и чтение из пустого. Алгоритм должен блокировать производителя при отсутствии свободных ячеек и задерживать потребителя, если в буфере нет данных, обеспечивая тем самым корректное взаимодействие и порядок FIFO.

Рисунок Г.6 – Презентация, страница 6

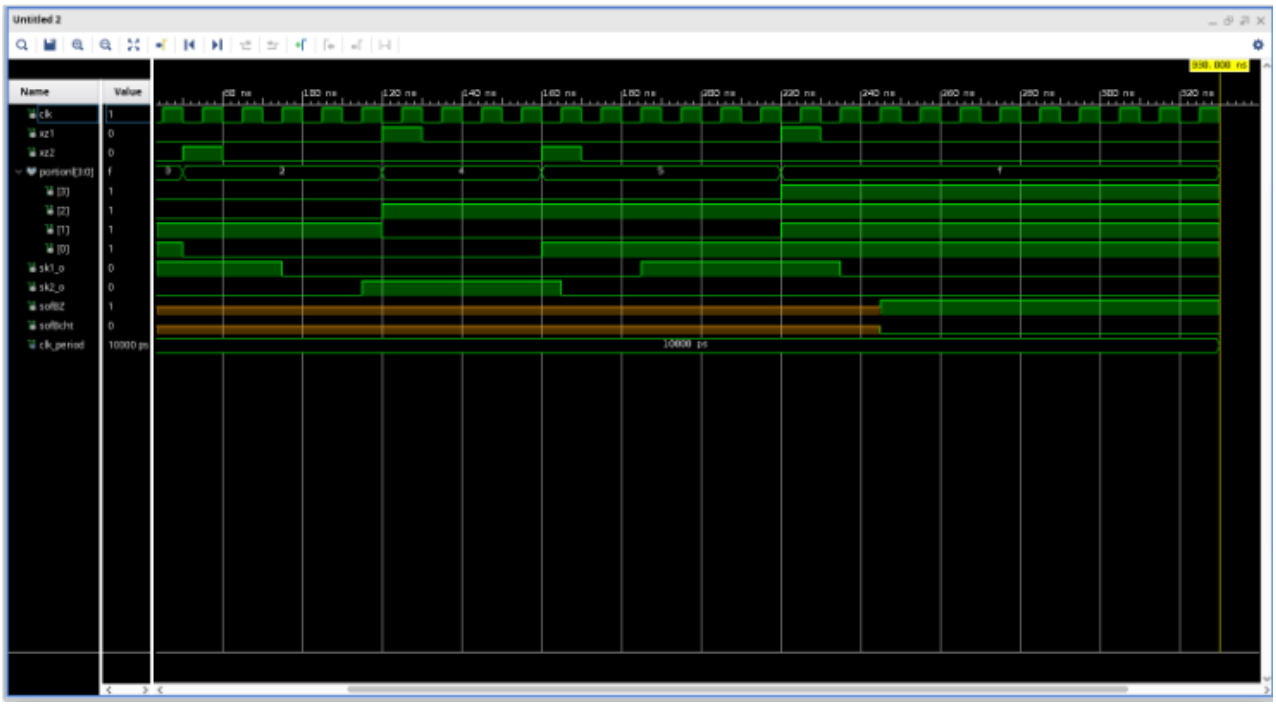


Рисунок Г.7 – Презентация, страница 7

Задача "спящий парикмахер"

Модель описывает одну службу обслуживания (парикмахера) и множество клиентов (до NNN). Если клиентов нет — парикмахер «засыпает». При появлении нового клиента он просыпается и обслуживает его. Если клиент приходит и видит, что парикмахер занят, он садится в кресло и засыпает. После окончания стрижки парикмахер проверяет, есть ли в очереди следующие клиенты — если да, обслуживает их по порядку, если нет — снова засыпает. Эта модель демонстрирует задачу асинхронного обслуживания с ограниченным буфером ожидания.

Рисунок Г.8 – Презентация, страница 8

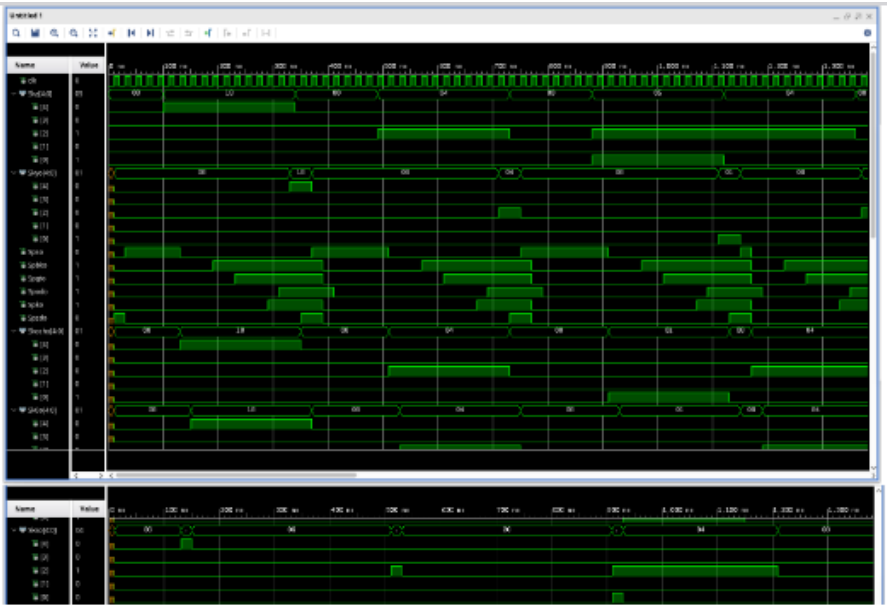


Рисунок Г.9 – Презентация, страница 9

Рисунок Г.10 – Презентация, страница 0

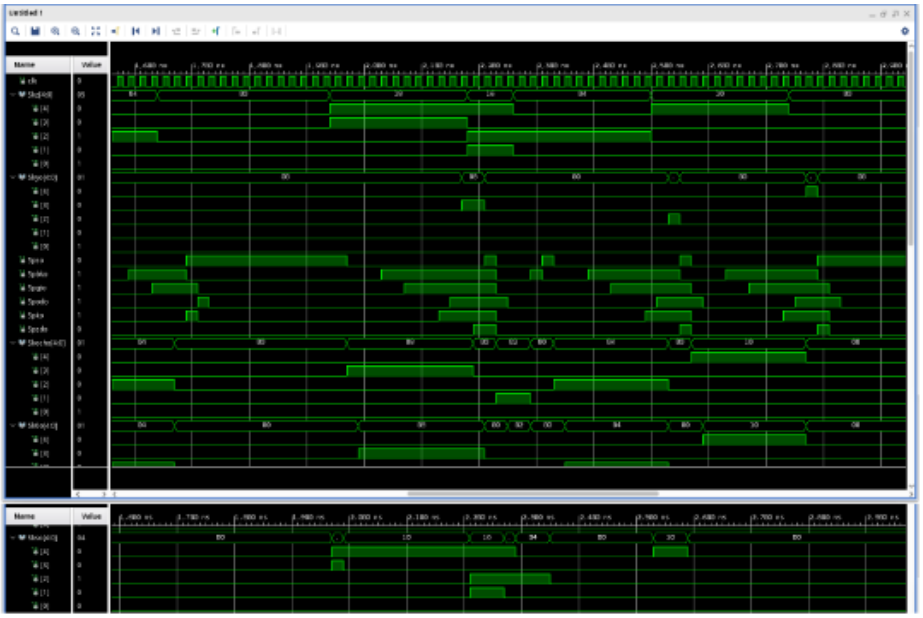


Рисунок Г.11 – Презентация, страница 11