

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет

Вычислительной техники

Кафедра

Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

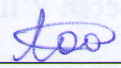
Профиль Вычислительные машины, комплексы, системы и сети

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ

РАБОТА БАКАЛАВРА

на тему

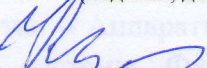
АППАРАТНЫЙ ПЛАНИРОВЩИК ЗАДАЧ С ЛОКАЛЬНОЙ
ОЧЕРЕДЬЮ

Студент 17.06.19  Лосяков Олег Андреевич

(подпись, дата)

Руководитель  Бикташев Р.А.

(подпись, дата)

Нормоконтролёр  Кучин А. В. 17.06.19

(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 31.05.19 № 38-09-14)

Заведующий кафедрой  Митрохин М.А.

(подпись)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 19.06.19 № 4)

Секретарь ГЭК  Гурин Е. И.

(подпись)

Пенза, 2019

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

Факультет

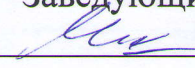
Вычислительной техники

Кафедра

Вычислительная техника

«Утверждаю»

Заведующий кафедрой

 М.А. Митрохин

«__» _____ 2019 г.

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Лосяков Олег Андреевич гр. 15BV1 факультета вычислительной техники.

2. Тема работы: «Аппаратный планировщик задач с локальной очередью».

Тема утверждена приказом ПГУ №554/о от "29" апреля 2019г.

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

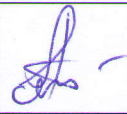
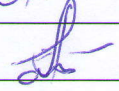
4. Задание на работу (назначение разработки, исходные данные и т.п.).

Назначение разработки – Аппаратный планировщик задач должен функционировать в составе многопроцессорной системы. Функцией планирования является формирование очереди в каждом из процессорных узлов. Планировщик должен направлять поступившую задачу в очередь процессорного узла с наименьшим количеством задач, запускать задачи на обработку в процессоре, удалять из системы обработанные задачи. Режим выполнения задачи: пакетный, с полным обслуживанием.

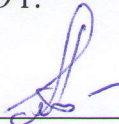
5. Перечень подлежащих разработке вопросов.

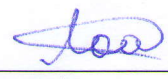
Требуется произвести анализ предметной области, разработать модель аппаратного планировщика задач с локальной очередью, которая будет отвечать поставленному на работу заданию. Произвести моделирование на языке VHDL. В модели должны быть: 10 процессорных блоков, блоки очереди на 8 мест, оформить выпускную квалификационную работу.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Анализ предметной области	20	29.04.19	
2.	Разработка алгоритма планирования/диспетчеризации	5	12.05.19	
3.	Разработка схемы устройства управления процессами с локальными очередями	10	12.05.19	
4.	Разработка схемы для проведения экспериментальных исследований	25	20.05.19	
5.	Моделирование, отладка и тестирование	20	28.05.19	
6.	Оформление выпускной работы	10	31.05.19	

Дата выдачи задания "29" апреля 2019 г.

Руководитель бакалаврской работы  Бикташев Р.А.
(подпись, дата)

Задание к исполнению принял студент  Лосяков О.А.
(подпись, дата)

Работу к защите допустить
Декан факультета  17.06.2019 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Бакалаврская работа содержит 83 листа, 19 рисунков, 5 источников, 2 приложения.

ПЛАНИРОВЩИК ЗАДАЧ, МНОГОПРОЦЕССОРНАЯ СИСТЕМА, МОДЕЛЬ НА ЯЗЫКЕ VHDL, АППАРАТНЫЙ ДИСПЕТЧЕР ЗАДАЧ.

Цель работы – разработка аппаратного планировщика задач с локальной очередью задач в составе многопроцессорной системы. Планировщик представлен в виде модели на языке VHDL.

В рамках выполнения выпускной квалификационной работы был создан проект, удовлетворяющий исходным требованиям. А именно, разработана структура устройства аппаратного планировщика задач с локальной очередью, которая состоит из трех составных частей: блока назначения задач, блока выбора очереди и n блоков очередей, где n – число процессоров в системе.

Благодаря разработанной модели, созданной на языке VHDL, удалось выявить, что данная схема диспетчеризации задач в многопроцессорной системе имеет ряд неоспоримых преимуществ перед планировщиком, который находится в ядре операционной.

Помимо этого, отсутствие конфликтов при обращении процессоров к блоку очереди выгодно отличает планировщик с локальной очередью задач от планировщика, где используется глобальная диспетчеризация.

Данную разработку рекомендуется использовать для организации планировщика задач в системах реального времени. Устройство может быть внедрено в многопроцессорные системы, где требуется увеличить производительность путём снижения временных затрат на переключение задач

					ПГУ 2.090301.15.001 ПЗ							
Изм.	Лист	№ докум.	Подпись	Дата								
Разраб.		Лосяков О.А.			Аппаратный планировщик задач Пояснительная записка			Лит.	Лист	Листов		
Провер.		Бикташев Р.А.								4	83	
Н. Контр		Кучин А.В.						ФВТ, каф. ВТ гр. 15ВВ1				
Утверд.		Митрохин М.А.										

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	7
1.1 Постановка задачи	16
1.2 Актуальность работы	17
2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ	19
2.1 Модель многопроцессорной системы	20
2.2 Интерфейс взаимодействия диспетчера задач	21
3 РАЗРАБОТКА СХЕМЫ ПЛАНИРОВЩИКА С ЛОКАЛЬНЫМИ ОЧЕРЕДЯМИ ЗАДАЧ	23
4 МОДЕЛИРОВАНИЕ НА ЯЗЫКЕ VHDL И АНАЛИЗ РЕЗУЛЬТАТОВ	27
4.1 Разработка модели на языке VHDL	27
4.2 Анализ результатов моделирования	32
ЗАКЛЮЧЕНИЕ	37
5 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	38
ПРИЛОЖЕНИЕ А ЛИСТИНГИ ПРОГРАММЫ	39
Приложение А.1 MainBlock.vhd	39
Приложение А.2 GenerationTask.vhd	50
Приложение А.3 UnitTaskAssignments.vhd	52
Приложение А.4 TheSelectionUnitQueue.vhd	57
Приложение А.5 FIFO.vhd	61
Приложение А.6 FIFO_only.vhd	65
Приложение А.7 Processor.vhd	69
ПРИЛОЖЕНИЕ В ПРЕЗЕНТАЦИЯ	73

ВВЕДЕНИЕ

Человечество постоянно пытается развить вычислительные мощности ЭВМ, с каждым разом превосходит предыдущие показатели и ставит всё новые и новые рекорды. Постоянная модернизация увеличила производительность ПЭВМ до порядка 300 операций в секунду, что было невиданной роскошью для людей, но и этого было мало. В современном мире компьютеры с легкостью обгоняют ПЭВМ в миллионы, а иногда и в десятки миллионов раз, но человечество не стоит на месте. Астрономы пытаются постичь суть Вселенной, биологи хотят разобраться в геноме человека, а авиаконструкторы заинтересованы в создании надежных и более экономичных самолетов, и всем им нужна более высокая скорость работы центральных процессоров. Постоянный рост производительности компьютеров – рождает все больший спрос на неё, поэтому мощности все равно не хватает. Постоянно наращивая мощность – человек, как правило, делал это с помощью увеличения тактовой частоты процессора, но в скором времени данному росту будет препятствовать ряд фундаментальных ограничений.

Хотелось бы сказать также, что большое число функций системы, которые используются наиболее часто, реализованы программным путем. К ним относятся планирования и диспетчеризации процессов, механизмы межпроцессорного взаимодействия, управления памятью, обработка прерываний. Так как большинство данных функций вынуждены использоваться очень часто, то страдает производительность, а значит и время, необходимое для выполнения пользовательских задач.

Именно поэтому наметилась тенденция к переходу от программной реализации алгоритмов ОС к аппаратной, которая вызвана стремлением к увеличению отношения производительность/стоимость ЭВМ и обусловлена

высоким развитием микроэлектроники. Действительно, аппаратная реализация алгоритмов ОС позволяет уменьшить время его выполнения и, следовательно, повысить производительность ЭВМ. С другой стороны, стоимость дополнительного оборудования для аппаратной реализации, благодаря прогрессу элементной базы, с каждым годом падает. Кроме того, аппаратная реализация алгоритмов ОС позволяет в определенной степени уменьшить трудности создания программного обеспечения, а также улучшить совместимость различных версий ОС для компьютеров разных платформ. Кроме того, в значительной степени устраняется трудоемкая операция отладки нижних уровней ОС и повышается корректность всего программного обеспечения, благодаря лучшей формализации функций, реализованных аппаратным способом по сравнению с программным способом.

Основной целью планировщика задач является непосредственное распределение времени между выполняющимися заданиями, которые были заданы пользователем или же системными процессами. Главной задачей является – удовлетворение требований, предъявляемым к системе.

Традиционное управление процессами и ресурсами, осуществляемое ядром операционной системы (ОС) на основе системного программного обеспечения, приводит к значительным замедлениям выполнения приложений, и, следовательно, к существенным потерям производительности вычислительной системы. Эта проблема вызвана наличием трудоемких компонентов ядра ОС, к которым относятся: диспетчеры задач, связанные с переключением процессов; планировщики задач, связанные с выделением и освобождением ресурсов; средства взаимодействия процессов, такие как каналы передачи сообщений; примитивы синхронизации и другие API-функции. [3]

В однопроцессорных операционных системах меж процессные коммуникации (IPC) ориентированы на локальные взаимодействия, т.е. между адресными пространствами на той же самой машине. Средства IPC традиционно реализуются в ядре, однако у такого подхода существуют проблемы. Приложения, которые выполняются с использованием быстро выполняющихся потоков, сталкиваются с проблемой эффективности, поскольку при взаимодействии на уровне ядра эффективность ограничена высокой трудоемкостью вызова функций ядра и переключения процессора из одного адресного пространства в другое. На мультипроцессорах с разделяемой памятью эта проблема может быть решена путем перемещения средств коммуникации и механизмов синхронизации из ядра на пользовательский уровень в пределах адресного пространства. При этом производительность увеличивается, но возникают проблемы надежности и безопасности из-за ограниченных возможностей современных средств тестирования программного обеспечения.

Эффективность функций планирования и диспетчеризации оценивается скоростью переключения процессов. Планировщик и диспетчер в многопроцессорной системе вызываются приложениями гораздо чаще, чем в однопроцессорной. Логично предположить, что в многопроцессорных системах планировщик и диспетчер должны работать быстрее, чтобы не создавать задержек. Однако возникает обратная картина, когда накладные расходы растут вместе с ростом числа процессоров в системе. Так между запрашивающими процессорами возникают конфликты за доступ к диспетчеру задач, в результате чего создаются очереди запрашивающих процессоров. На устранение конфликтов требуется дополнительная синхронизация работы процессоров. Задача планирования и диспетчеризации в многопроцессорных системах, в отличие от однопроцессорных, является двумерной, поскольку кроме функции назначения задачи необходимо

выполнять функцию выделения целевого процессора. Кроме того, в планировщиках с общей очередью существует явление перезагрузки кэш-памяти, связанное с переключением задач, когда прерванная задача с высокой вероятностью может быть направлена для продолжения обслуживания в другой процессорный узел. Названное явление увеличивает частоту кэш-промахов и неизбежно приводит к снижению производительности многопроцессорной системы.

Именно поэтому данная тема выпускной квалификационной работы является актуальной. Во-первых, данная модель диспетчера задач будет выполнена аппаратно, что снижает нагрузку на низшие уровни исполнения операционной системы, а кроме того имеет не одну общую очередь задач, а распределенную систему очередей. Каждое ядро (процессор) должен иметь свой блок очереди. В следствии чего решается проблема с перегрузкой кэш-памяти в моделях с одной глобальной очередью.

1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В данном разделе работы будет произведен обзор используемых алгоритмов управления процессами и ресурсами в многопроцессорных (многоядерных) системах с целью выявить преимущества и недостатки каждого из них. Стоит начать с того, что существует всего два подхода к аппаратно-ориентированной реализации: микропрограммный и схемный.

Программный способ предполагает внесение обоснованно выбранных функций управления процессами и ресурсами в систему команд целевого процессора. В настоящее время микропрограммно реализуют неизменяемые функции ОС, которые способны работать в пользовательском пространстве и могут вызываться выполняющимся приложением. Микропрограммно реализуют примитивы типа «тестирование и блокировка», «выборка и инкремент», «выборка и добавление» и др. Данные примитивы позволяют синхронизировать процессы посредством активного ожидания или же спин-блокировки. В свою очередь планировщики или диспетчеры задач чаще всего вызываются автоматически по истечении таймера и реализуются как процедура, выполняющаяся в пространстве ядра. Как правило такие функции требуют перенастройки для оптимизации вычислительного процесса под конкретный класс прикладных задач, что затрудняет использование микропрограммного подхода.

Схемная же реализация имеет смысл в том, что некоторый функционал ядра операционной системы переносится в специальный процессор или выносится на отдельное устройство управления. Именно такая реализация позволяет полностью освободить ядро ОС от функций управления переключением задач прерываниями или таймерами, синхронизации процессов, обмена сообщениями. Как можно понять, в результате API-функции, выполняемые в специальном процессоре, значительно ускоряются,

а тщательная отладка аппаратуры современными средствами проектирования ПЛИС гарантирует надежность и безопасность операционной системы. Схемная реализация обеспечивает повышение производительности многопроцессорной системы в следствии:

- 1) более быстрого выполнения функций операционной системы по диспетчеризации процессов и управлению ресурсами в аппаратных средствах;
- 2) освобождения центральных процессоров от диспетчеризации задач, что позволяет полностью занять их только выполнением пользовательских приложений. [5]

Возможно несколько вариантов включения аппаратной части ядра операционной системы в многопроцессорную систему. В данной работе мы рассмотрим три способа:

- 1) через системную общую шину (Рисунок 1);
- 2) через отдельную (специальную, периферийную) шину (Рисунок 2),
- 3) с непосредственными связями (Рисунок 3).

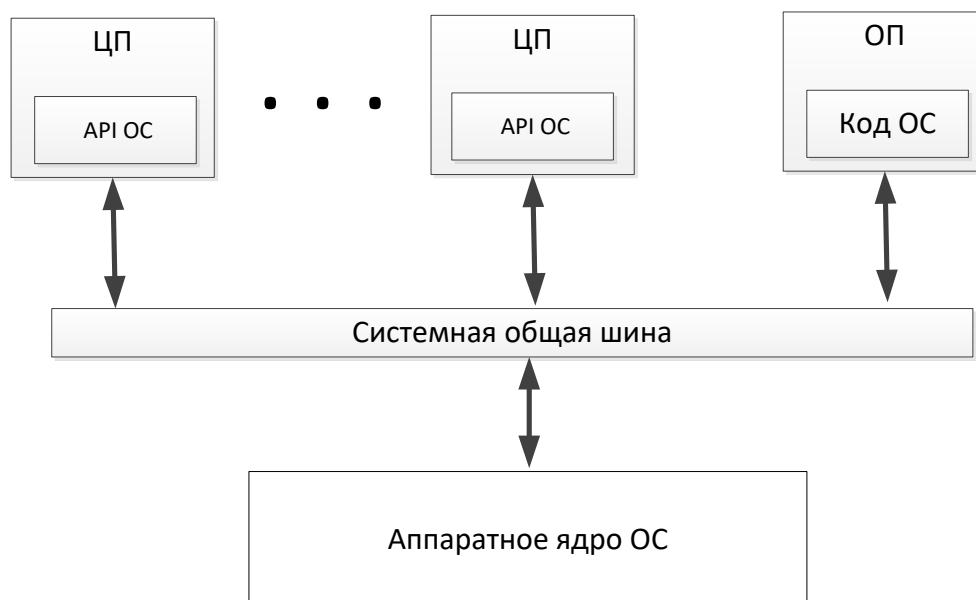


Рисунок 1. Включение аппаратного ядра через общую шину.

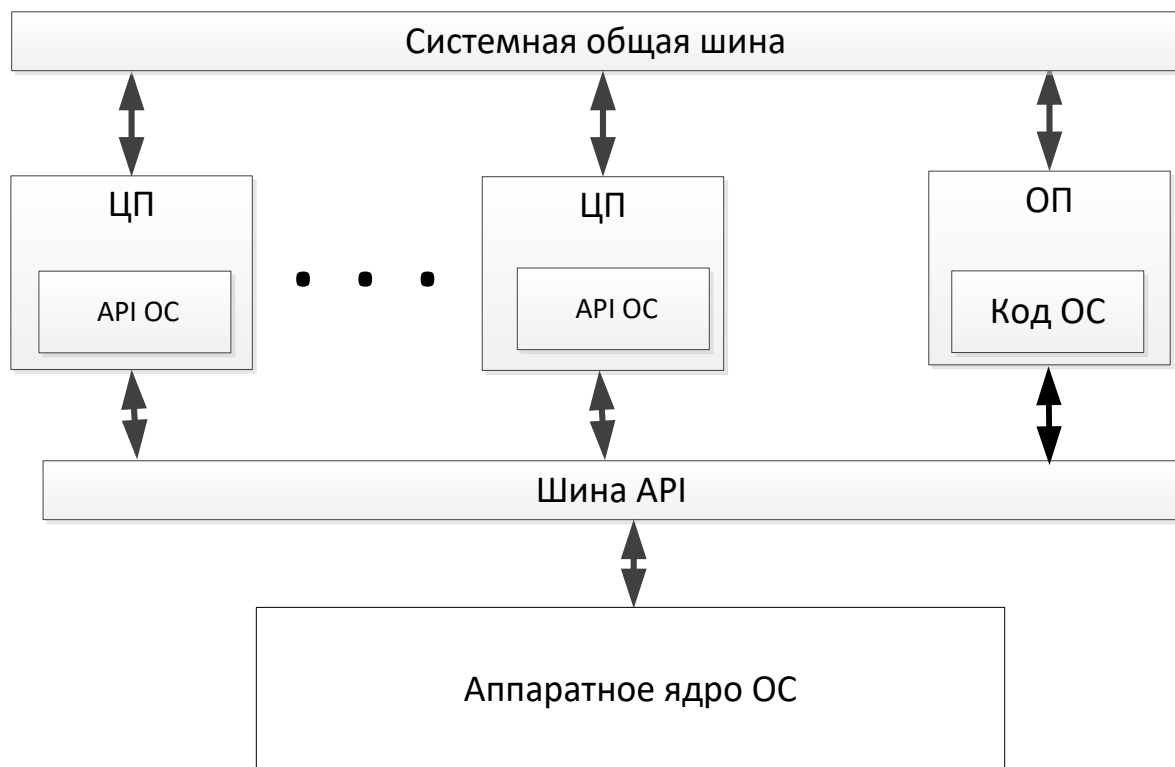


Рисунок 2. Включение аппаратного ядра через отдельную системную шину

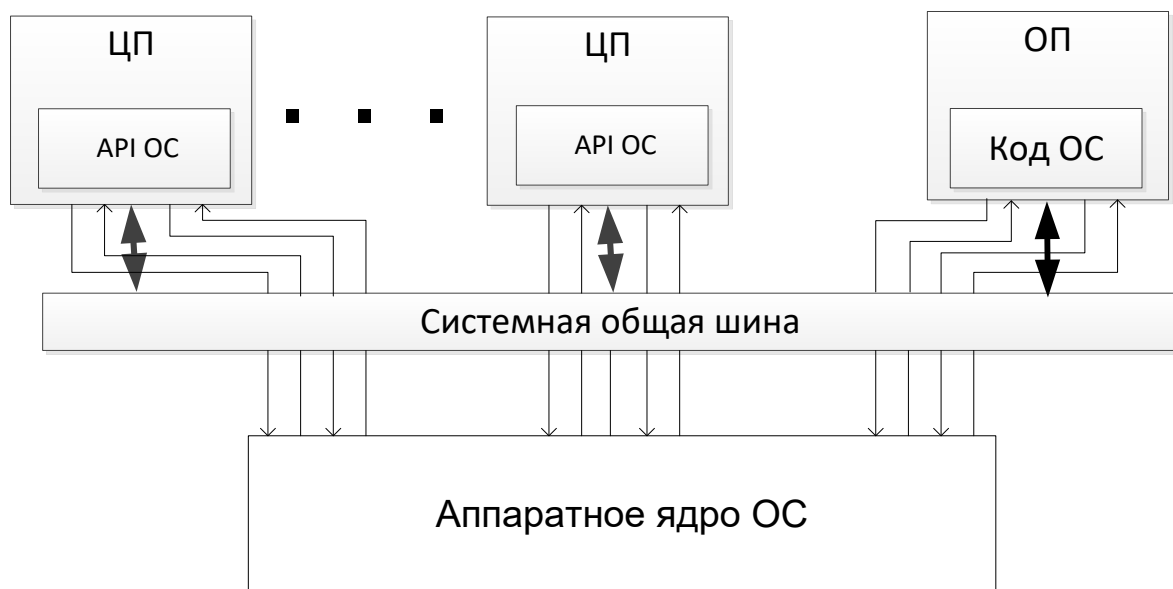


Рисунок 3. Включение аппаратного ядра с непосредственными связями

Первый вариант структурной схемы использует единственную общую шину для взаимодействия между процессорами, памятью системы, а также между API (Application Programming Interface) операционной системы (API ОС) и аппаратным ядром. API ОС является частью операционной системы, обеспечивающей программный интерфейс между выполняющимися приложениями через ядро ОС. API определяет функциональность, которую предоставляет программа операционной системы, при этом API позволяет абстрагироваться от того, как именно эта функциональность реализована. Самой важной частью API операционных систем является множество системных вызовов. Положительной стороной такой архитектуры является то, что единый интерфейс дает возможность использования типовых средств (стандартного интерфейса) для включения аппаратного ядра в многопроцессорную систему. Однако она имеет недостаток, заключающийся в последовательном характере взаимодействия через общую шину, поскольку только два устройства могут связываться на любом промежутке времени. Это обстоятельство не обеспечивает параллелизм связей, что может создать высокий трафик, приводящий к перегрузке общей шины.

Во втором варианте в архитектуре с отдельной (специальной) шины взаимодействие процессоров с модулем аппаратного ядра производится по специальной шине API. В такой архитектуре системная шина не нагружена дополнительным трафиком от команд и данных операционной системы, что позволяет увеличить общую производительность многопроцессорной системы.

В третьем варианте между API процессора и аппаратным ядром ОС устанавливается непосредственная связь с помощью управляющих сигналов. Для этого в составе API каждого процессора включается два дополнительных блока. Первый - для декодирования команд, преобразующий команды системного вызова в сигналы управления и передачи их в аппаратное ядро.

Второй – для приема сигналов, поступающих из ядра. Этот вариант наиболее производительный, поскольку несколько процессоров одновременно могут взаимодействовать с ядром. Однако для его реализации необходимы дополнительные входы/выходы, что приводит к увеличению числа внешних выводов процессора.

Как видно из вышеннаписанного: нет однозначного ответа на вопрос каким же способом пользоваться. В системе, где максимально важна производительность и не важны затраты на производство и размеры аппаратной составляющей выгодней использовать способ с непосредственными связями. В случае, если важно использовать стандартный интерфейс для удешевления разработки, то выгодно использовать способ с общей шиной данных. Золотой серединой же является включение аппаратной части ядра через собственную системную шину.

Рассмотрим более подробно устройство управления процессами и ресурсами (УУПР), которое составляет аппаратное ядро операционной системы и состоит из трех основных частей: интерфейса ядра, блока управления процессами (БУП) и блока управления ресурсами (БУР). На рисунках ниже показаны возможные способы структурной реализации УУПР. Интерфейс ядра состоит из логики, необходимой для декодирования инструкций API ОС и выполнения всех необходимых обменных взаимодействий. В качестве примера будет рассмотрено три варианта устройства управления процессами: аппаратное ядро ОС с общей шиной, аппаратное ядро ОС с непосредственными связями, аппаратное ядро ОС с общей памятью.

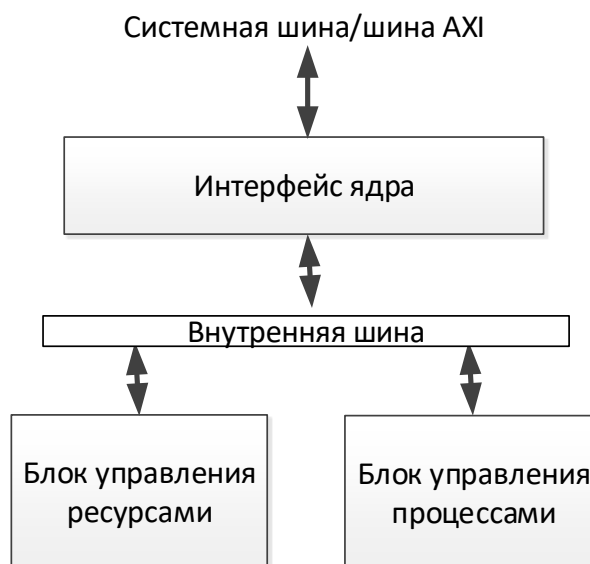


Рисунок 4. Структура аппаратного ядра ОС с общей шиной

Первый вариант (рисунок 4) использует единственную общую шину для взаимодействия между блоками УУПР, а также между API ОС и аппаратным ядром ОС. Положительной стороной является то, что единственный тип интерфейса позволяет использовать типовые средства для подключения аппаратного ядра в многопроцессорную систему. Такая архитектура имеет недостаток, вызванный последовательным характером взаимодействия через общую шину, что не способствует высокой производительности аппаратного ядра.

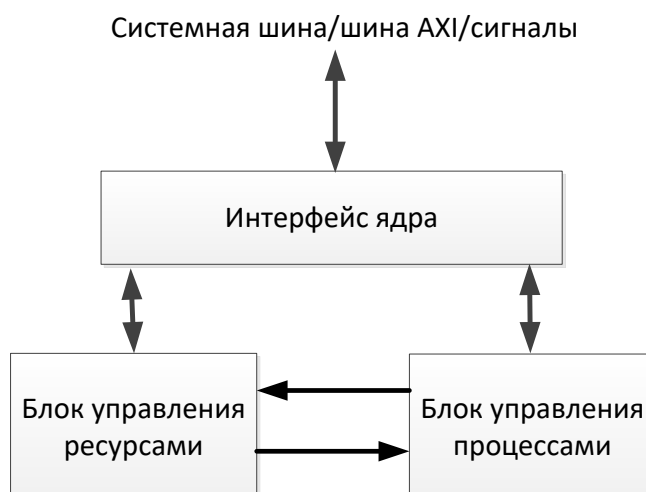


Рисунок 5. Структура аппаратного ядра ОС с непосредственными связями

Второй вариант реализации (рисунок 5) заключается в том, что взаимодействие внутренних модулей выполняется путем создания непосредственных связей. Такая архитектура связей имеет значительно более высокую пропускную способность, однако сложность её реализации несколько выше остальных.

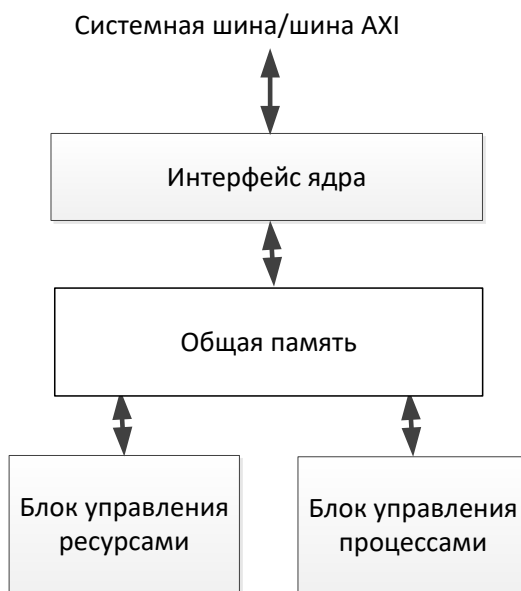


Рисунок 6. Структура аппаратного ядра ОС с общей памятью

Архитектура общей памяти показана на рисунке 6. Она характеризуется совместным использованием модулей памяти и регистров, принадлежащих каждому блоку БУП и БУР. Идея состоит в том, что каждый блок владеет рядом регистров и возможно модулей памяти, в которые они пишут и читают данные, причем каждый блок также в состоянии считывать данные из регистров и памяти других блоков, поскольку они имеют единое адресное пространство.

Таким образом, интерфейс каждого блока относительно простой, состоит только из регистров и логики доступа к памяти. Это обеспечивает единственную и довольно простую интерфейсную архитектуру, которую будет легче поддерживать, чем несколько различных интерфейсов. API ОС может связаться с аппаратным ядром путем записи в соответствующие регистры. Все модули ядра могут связаться друг с другом, поскольку и регистры, и сегменты памяти поддерживают несколько читателей и писателей. Архитектура общей

памяти обеспечивает параллелизм в связях как между блоками ядра, так и между аппаратурой ядра и API ОС. В то же время блоки ядра могут получать доступ только к частям регистров и памяти каждого из других модулей, а API-интерфейс в состоянии получать доступ ко всей карте распределения памяти.

Главными модулями блока БУП (рисунок 7): таймер, сторожевой таймер, диспетчер и планировщик задач. В его состав входит также таблица процессов, являющийся RAM-памятью для хранения дескрипторов готовых к выполнению процессов, которые содержат необходимые данные для управления сохранением и восстановлением контекстов.

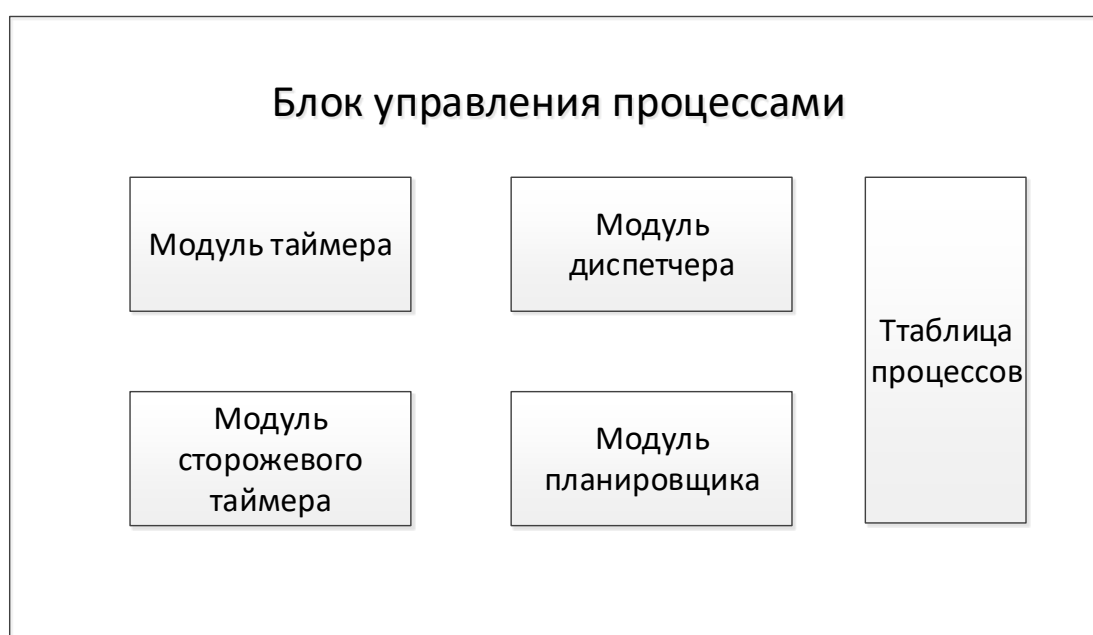


Рисунок 7. Состав блоков управления процессами аппаратного ядра ОС

БУР (рисунок 8) содержит в своём составе средства управления ресурсами (механизмы синхронизации процессов), которые обеспечивают взаимодействие параллельных процессов при доступе к общим (разделяемым) ресурсам, основанное на критических интервалах: мьютексы, бинарные и счетные семафоры, очереди сообщений. Этот блок может включать и другие механизмы синхронизации, например, флаги событий, почтовые ящики и т.д.

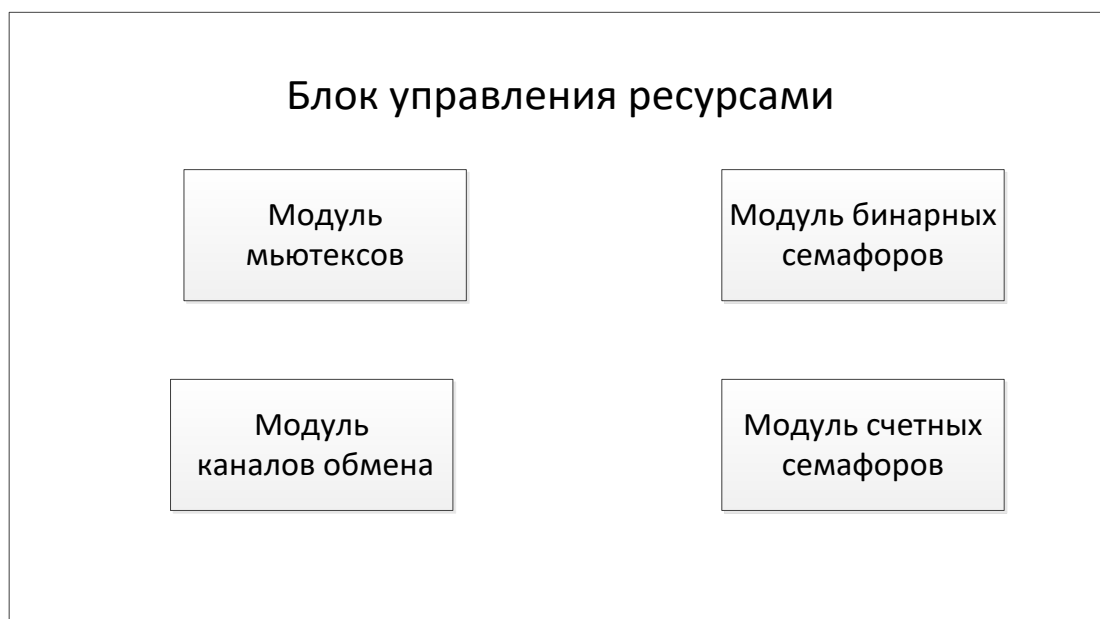


Рисунок 8. Состав блоков управления процессами ресурсами ядра ОС

В последующих разделах будут рассмотрены вопросы структурной реализации аппаратного ядра многопроцессорных операционных систем с использованием средств функционального моделирования. При проектировании устройств аппаратной поддержки важен выбор математического аппарата для формализации, обеспечивающего простоту описания, верификации и структурной реализации алгоритмов управления процессами и ресурсами. Наиболее подходящим аппаратом, на наш взгляд, является язык логики недетерминированных автоматов. Использование этого языка позволило разработать НДА- модели алгоритмов взаимодействия параллельных процессов, основанные на критических интервалах (модуль мьютексов), и модели очередей сообщений (модуль каналов). Из средств управления процессами разработаны модули диспетчера и частично планировщика задач, реализация которых показана в приведенных ниже разделах.

1.1 Постановка задачи

Необходимо разработать модель аппаратного планировщика задач с локальной очередью, который должен функционировать в составе многопроцессорной системы. Под функцией планирования понимается формирование очереди задач в каждом из процессорных узлов. Для обеспечения равномерного распределения нагрузки новая задача отправляется в процессорный узел с наименьшей очередью.

Функционально устройство планирования/диспетчеризации может быть реализовано в виде 3-х блоков:

- блок выбора очереди;
- n блоков очередей задач FIFO, где n – количество процессорных узлов;
- блок назначения задач.

Рассматриваемая подсистема должна быть разработана в виде набора программных модулей и являться частью многофункциональной системы, предназначенной для отображения целевой обстановки.

Блок выбора очереди предназначен для анализа наличия свободных мест в очереди задач и помещения идентификатора новой задачи в блок очереди FIFO. Кроме того, он же осуществляет выборку идентификатора задачи из очереди по принципу First In First Out т.е. «первым пришёл – первым обслужился» и дальнейшем помещением его в свободный процессор.

Блок очереди задач FIFO предназначен для хранения поступающих в него идентификаторов задач, которые могут состоять из нескольких полей. Одно из полей должно обязательно указывать на адрес каждой задачи в памяти многопроцессорной системы. Помимо этого, блок FIFO по запросу от блока управления должен извлекать идентификатор требуемой задачи из вершины списка.

Блок назначения задач принимает новый идентификатор задачи в многопроцессорной системе и принимает из блока выбора очереди тот блок очереди FIFO в котором находится меньше всего задач. Возможны варианты, когда данный блок также принимает участие в перераспределении очереди с одного процессора на другой в случае выхода первого из строя или же в случае исполнения трудоемкой задачи.

1.2 Актуальность работы

Традиционно планировщик и диспетчер задач выполняются программно и реализуются в SMP-системах методом вызова этих функций из общей памяти, в которой резидентно хранится программа операционной системы. Практическая программная реализация предполагает два варианта: 1) в пространстве пользователя; 2) в пространстве ядра.

Программная реализация в пространстве пользователя хотя и является быстрой, однако вызывает затруднения при программировании, поскольку для выполнения сопряженной с диспетчеризацией процедуры синхронизации требуется 3 семафора: один – счётчик – для подсчета числа мест, занятых готовыми для выполнения процессами; другой – счётчик – для подсчета числа процессоров, занятых обслуживанием; и третий – мьютекс – для реализации взаимного исключения, предотвращающего одновременный доступ нескольких свободных процессоров к единственной очереди, которая в данном случае играет роль общего ресурса. Эти обстоятельства могут привести к трудно выявляемым ошибкам и непредсказуемым последствиям при работе операционной системы, особенно реального времени.

Как можно понять, в качестве выхода из данной ситуации возможна реализация процедуры планирования/диспетчеризации в пространстве ядра, однако возрастающие при этом временные затраты на выполнение системных вызовов резко уменьшают производительность многопроцессорной системы.

Исходя из вышесказанного, для решения вышеперечисленных проблем со сложностью разработки системы или ростом временных затрат на выполнение диспетчеризации, возможна аппаратная реализация функций планирования/диспетчеризации, которая не только снимает проблему временных потерь, но и позволяет увеличить надежность операционной системы, что особенно важно для систем реального времени.

Предлагаемый подход заключается в том, что планировщик и диспетчер задач выполняются в виде независимого аппаратного устройства (специализированного процессора) в составе многопроцессорной системы. Вследствии чего ядро многопроцессорной системы освобождается от необходимости в диспетчеризации задач, увеличивается стабильность и производительность всей системы в целом. Данные факторы крайне необходимы для работы многопроцессорных систем в целом, систем реального времени в частности.

2 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Для начала проектирования многопроцессорной системы с аппаратным планировщиком задач на основе локальных очередей необходимо определить из каких элементов данная система должна состоять. Обобщенно схема подключения устройства управления процессами в многопроцессорную систему может выглядеть следующим образом.

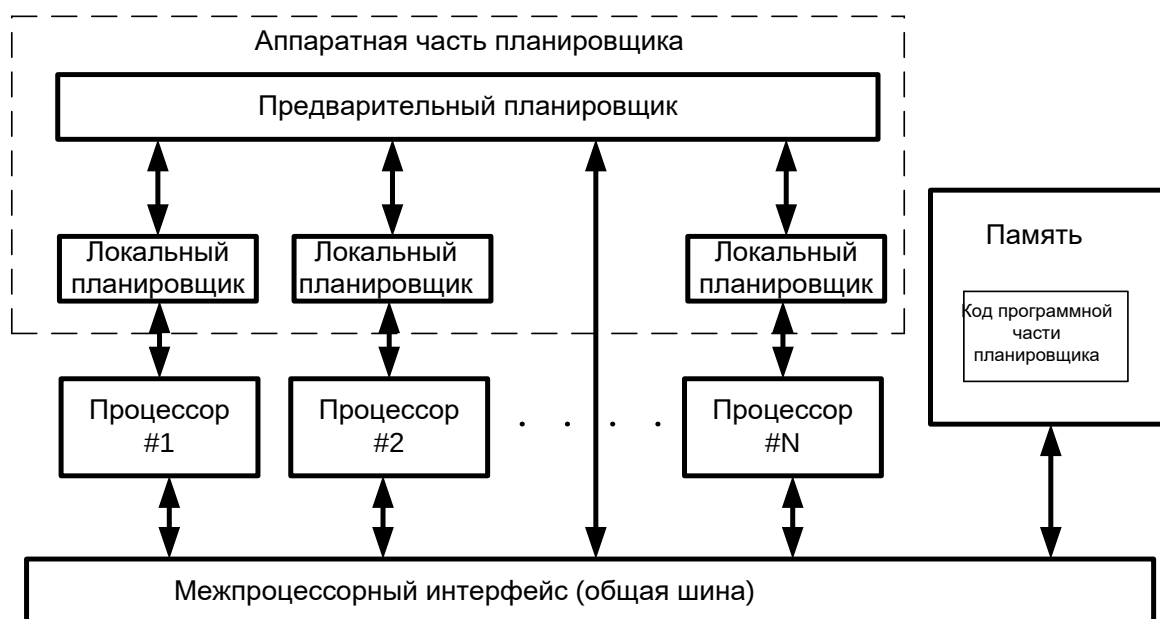


Рисунок 9. Схема подключения устройства управления процессами в многопроцессорную систему.

Аппаратная часть устройства управления процессами содержит два основных блока: планировщик задач и локальный диспетчер. Первый принимает извне новую задачу и передает её локальному диспетчеру. В реальной системе эта процедура заключается в следующем: новой задаче выделяется раздел общей памяти и присваивается соответствующий идентификатор, однозначно указывающий на адрес задачи в памяти. Идентификатор задачи помещается в очередь одного из локальных диспетчеров. Далее производится обработка задачи в соответствии с ее очередью в локальном диспетчере. Далее будет рассмотрена модель многопроцессорной системы и интерфейс взаимодействия локального диспетчера с планировщиком задач и процессором.

2.1 Модель многопроцессорной системы

Приведем пример на многопроцессорной системе реального времени. Главной задачей системы реального времени является планирование задач, которое обеспечивало бы предсказуемое поведение системы при всех обстоятельствах. В многозадачных системах общего назначения используется алгоритм круговой диспетчеризации, основанные на понятии непрерывного кванта времени ("time slice"), предоставляемого процессу для работы. Планировщик по истечении каждого кванта времени просматривает очередь активных процессов и принимает решение, кому передать управление, основываясь на приоритетах процессов. Приоритеты могут быть фиксированными или меняться со временем - это зависит от алгоритмов планирования в данной ОС, но рано или поздно процессорное время получат все процессы в системе. Планировщики операционных систем реального времени имеют возможность сменить процесс до истечения "time slice", если в этом возникла необходимость. ОС РВ отличаются богатством различных алгоритмов планирования: динамические, приоритетные, монотонные, адаптивные и пр., цель же всегда преследуется одна - предоставить инструмент, позволяющий в нужный момент времени исполнять именно тот процесс, который необходим.

Модель многопроцессорной системы с использованием аппаратного устройства управления процессами представлена на рисунке 10. Локальный диспетчер на основании идентификатора прерывает выполнение текущей задачи и организует запуск следующей по очереди задачи. Для упрощения программной модели принято, что часть функций планировщика и диспетчера задач, которые связаны с прерыванием по истечении кванта процессорного времени, сохранением и восстановлением контекстов задач и т.д., реализуются традиционно.

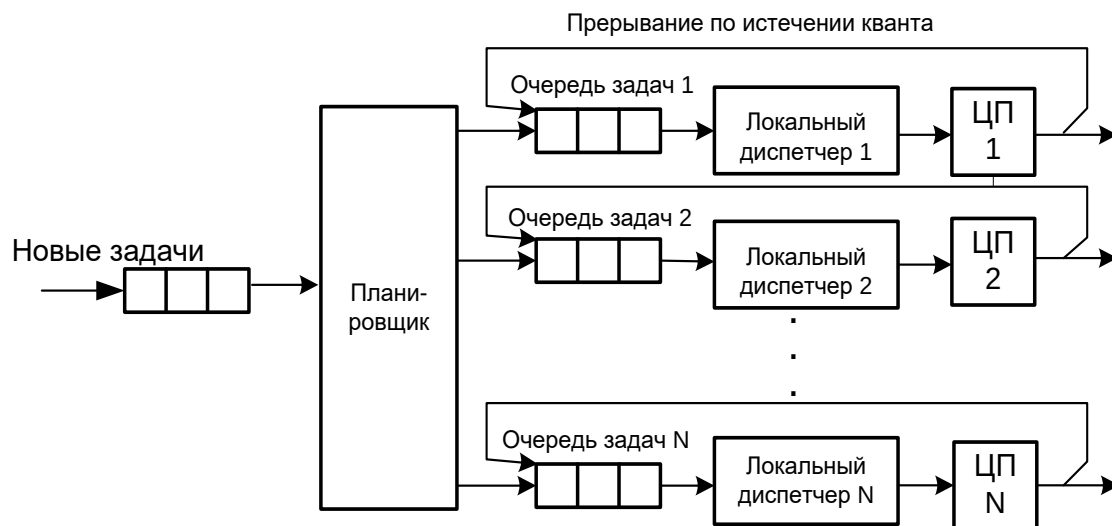


Рисунок 10. Модель многопроцессорной системы с локальным планировщиком задач

2.2 Интерфейс взаимодействия диспетчера задач

Интерфейс взаимодействия устройства локального диспетчера с каждым из процессоров и с планировщиком задач представлен на рисунке 11.

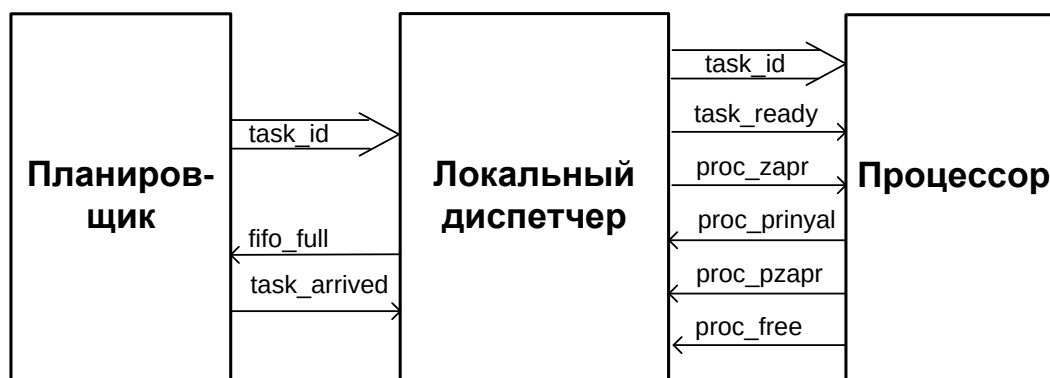


Рисунок 11. Интерфейс взаимодействия локального диспетчера с планировщиком задач и процессором

Функционирование взаимодействующих устройств по предложенному интерфейсу происходит следующим образом. Если планировщик получил новую задачу, он пытается поместить её в один из локальных диспетчеров. Если в очереди к локальному диспетчеру имеется процесс, ожидающая обслуживания, то он при условии, что процессор свободен, устанавливает

активный уровень сигнала «Запрос» на линии (*proc_zapr*). Процессор выходит из спящего режима, и сообщает локальному диспетчеру о том, что готов к приёму и выполнению задачи. Для этого процессор устанавливает активный уровень сигнала «Подтверждение запроса» (*proc_pzapr*), который воспринимает локальный диспетчер. Кроме того, процессор снимает сигнал «Свободен» (*proc_free*). Далее локальный диспетчер переходит к непосредственной передаче идентификатора задачи процессору. Для этого он извлекает идентификатор задачи из очереди и выставляет его на шину данных процессора (*task_id*), после чего передает в процессор сигнал «Процесс готова» (*task_ready*). Процессор воспринимает активный уровень сигнала «Задача готова», тем самым ему гарантируется, что данные на шине достоверны (*task_id*). Процессор фиксирует эти данные в своем внутреннем регистре, и когда его работа с шиной будет окончена, он выставляет сигнал «Задача принята» (*proc_prinyal*). [4]

Процессор, не занятый обслуживанием задачи, находится в спящем режиме. Сигнал «Свободен» (*proc_free*), поступающий от процессора к локальному диспетчеру, принимает активный уровень. Если в очереди имеются готовые задачи, процессор выходит из спящего режима, выбирает задачу из начала очереди (*task_read*), получает идентификатор задачи по шине (*task_id*), выставляет сигнал «Задача принята» (*proc_prinyal*) и переходит в режим «Занято». По завершению работы процессор извещает локальный диспетчер об освобождении установкой активного уровня сигнала «Свободен» (*proc_free*).

3 РАЗРАБОТКА СХЕМЫ ПЛАНИРОВЩИКА С ЛОКАЛЬНЫМИ ОЧЕРЕДЯМИ ЗАДАЧ

Согласно разработанной выше модели многопроцессорной системы и интерфейсу взаимодействия системы с локальным диспетчером задач было необходимо разработать конкретную схему устройства диспетчеризации для использования ее в моделировании на языке VHDL.

Аппаратно она должна быть представлена в виде трех типов блоков: блоки очередей со схемами их локального управления, блок назначения конкретной задачи каждому процессору, глобальный блок формирования очередей.

Блок очереди FIFO предназначен для хранения готовых к обслуживанию задач. Число этих блоков равно числу процессоров в системе. Функционально они не отличаются от тех же блоков в диспетчерах с глобальной очередью, но имеют меньшее количество ячеек для хранения задач. Не смотря на совпадение функциональной части, они имеют несколько другую организацию. А именно: записывать в очередь может только диспетчер, по сигналам записи S_wreq , подаваемым из блока планирования. Считывание содержимого регистра очереди производится по сигналу S_rdreq , поступающему от каждого процессора по индивидуальным каналам связи, которые имеются у каждого процессора.

Входная шина данных формируется блоком планирования (V_task_id [15..0]) и является общей для всех блоков очереди FIFO. Каждый процессор управляет только своей очередью, другими словами он может читать содержимое только из одного блока FIFO. Также каждый процессор постоянно сканирует свою очередь (семафорный сигнал S_fifo_empty) на предмет наличия идентификаторов задач. Если в регистрах FIFO имеется идентификатор хотя бы одной задачи, процессор читает его, если нет, то переходит в режим ожидания.

Выходная шина данных из каждого блока *FIFO* заводится в соответствующий ему процессор. Семафорный сигнал *S_fifo_full* используется блоком назначения задач для указания на заполнение очереди блока *FIFO*. По этому сигналу диспетчер прекращает прием нового идентификатора в данную очередь.

Входная шина данных формируется блоком планирования (*V_task_id* [15..0]) и является общей для всех блоков очереди *FIFO*. Каждый процессор управляет только своей очередью, другими словами он может читать содержимое только из одного блока *FIFO*. Также каждый процессор постоянно сканирует свою очередь (семафорный сигнал *S_fifo_empty*) на предмет наличия идентификаторов задач. Если в регистрах *FIFO* имеется идентификатор хотя бы одной задачи, процессор читает его, если нет, то переходит в режим ожидания. [4]

Выходная шина данных из каждого блока *FIFO* заводится в индивидуальный процессор. Семафорный сигнал *S_fifo_full* используется блоком назначения задач для указания на заполнение очереди блока *FIFO*. По этому сигналу диспетчер прекращает прием нового идентификатора в данную очередь.

Блок выбора очереди предназначен для определения блока *FIFO*, содержащего наименьшее количество записей. Номер очереди *FIFO* с наименьшим числом записей передается в блок планирования по 3-х разрядной выходной шине *id* [2..0].

Выбор наименее загруженной очереди осуществляется на основании специальных сигналов *use* [5..0], вырабатываемых блоками очереди *FIFO*. Число занятых ячеек в каждом блоке очереди формируется как разность содержимого счетчиков записи и чтения. Блок выбора очереди осуществляет вычисление наименьшего значения из всех выставленных на шинах *use* [5..0] с помощью комбинационной схемы сравнения. На основании этого делается

вывод о том, в какую очередь следует загружать идентификатор новой задачи. К сожалению данная организация не учитывает объем каждой задачи, находящейся в очереди задач.

Блок назначения задач принимает новый идентификатор задачи и под действием сигналов S_{wreq} помещает его в наименее заполненную очередь FIFO. Кроме того, на данный блок может возлагаться функция контроля состояния всех процессоров и функция перераспределения задач по очередям для выравнивания нагрузки на процессоры. Если один из процессоров выходит из строя, то запись новой задачи в его очередь не должна производиться, а его очередь необходимо перераспределить. Если очередь некоторых процессоров загружена трудоемкими процессами, то также должно осуществляться перераспределение. Однако с целью упрощения экспериментов функция контроля в данной модели не реализована.

Учитывая задачу на разработку, а также описание всех блоков подсистемы была разработана схема устройства управления процессами с локальными очередями, которая включает в себя блоки очередей FIFO, блок назначения задач, а также блок выбора очереди. На данной схеме (рисунок 12) представлены все каналы связи, а также шины, необходимые для работы данного устройства.

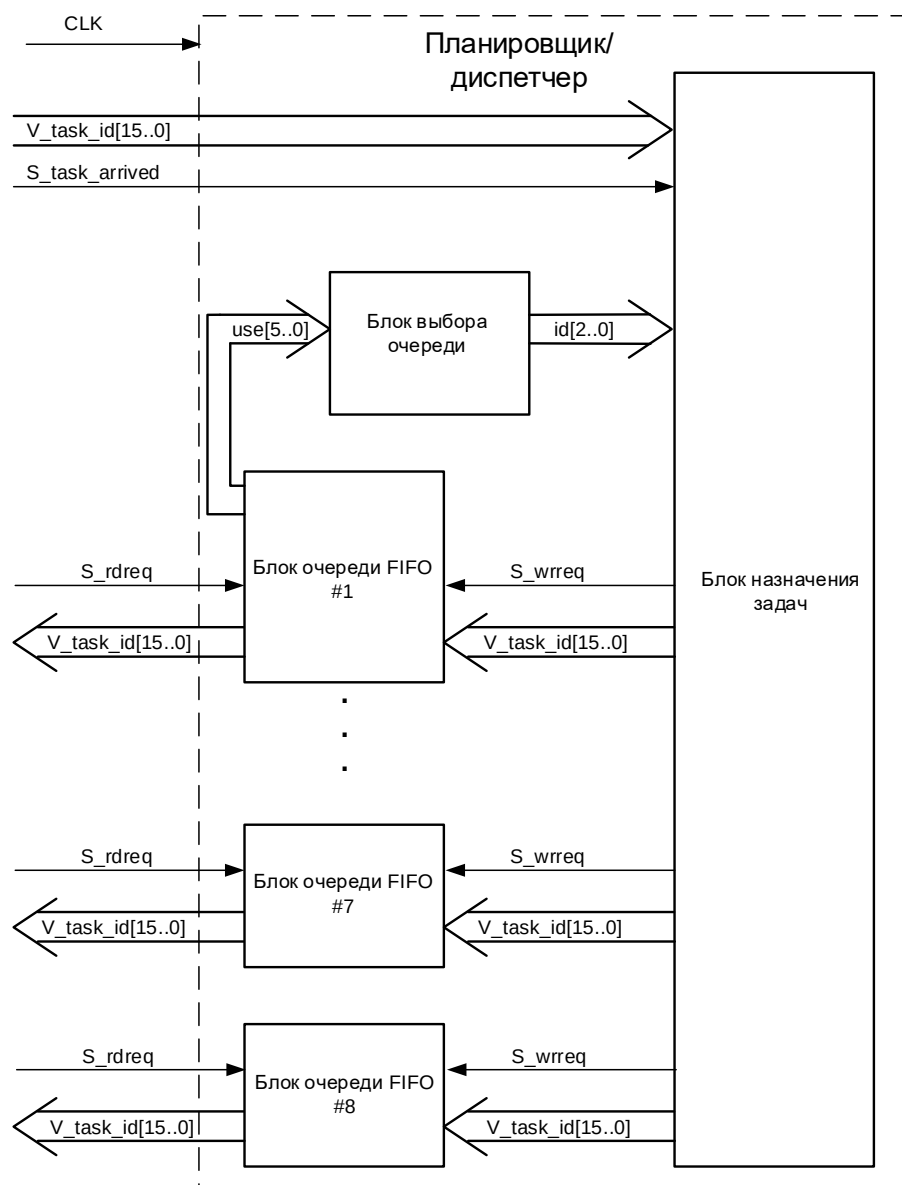


Рисунок 12. Структурная схема модели устройства управления процессами с локальными очередями

4 МОДЕЛИРОВАНИЕ НА ЯЗЫКЕ VHDL И АНАЛИЗ РЕЗУЛЬТАТОВ

4.1 Разработка модели на языке VHDL

Для начала разработки модели необходимо выбрать самое подходящее средство разработки, которое отвечало бы всем требованиям, а также обладало возможностью симуляции и быстрого форматирования. Ключевыми требованиями стали:

1. Редактор ввода на языках VHDL;
2. Синтезатор VHDL кода;
3. Редактор тестовых воздействий для программы моделирования;
4. Программа функционального и временного моделирования.

При анализировании данной области рынка, выяснилось, что соответствует всем требованиям, а также обладает пробным периодом только один продукт: Xilinx ISE Design Suite 14.7.

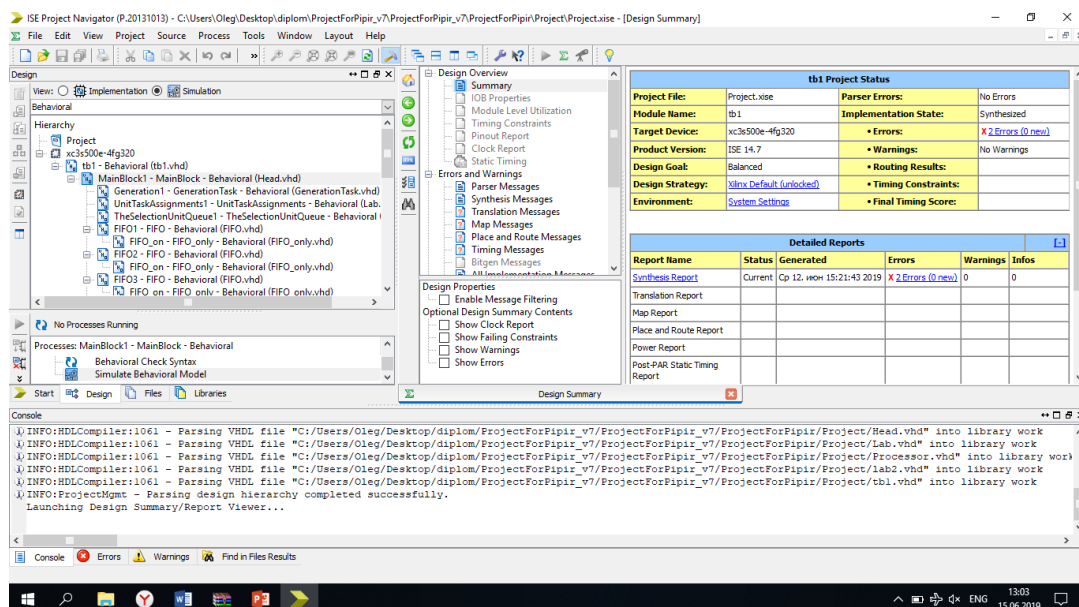


Рисунок 13. Окно программы Ise Project Navigator

Для моделирования работы планировщика/диспетчера с локальными (распределенными) очередями была применена схема, показанная на рисунке14.

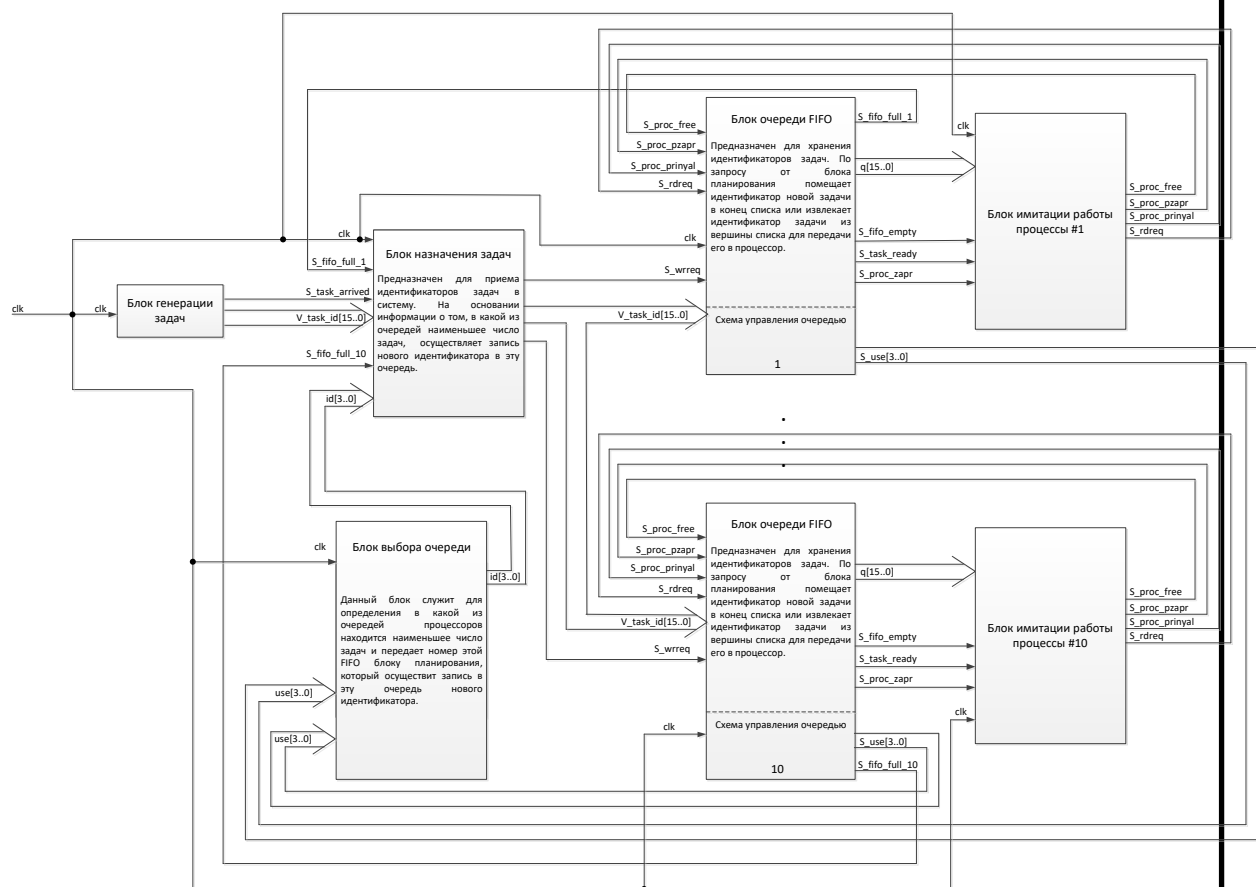


Рисунок 14. Схема для проведения экспериментальных исследований

Аппаратно устройство планировщика задач будет состоять из следующих блоков:

1. Блок генерации задач
2. Блок назначения задач
3. Блок выбора очереди
4. Блок локальной очереди FIFO (10 штук).
5. Блок имитации работы процессора (10 штук).

Модель была разработана с использованием системы разработки Design Ise Suite 14.7. В качестве семейства было выбрано Spartan 3E и

устройство XC3S500E. В качестве приложения моделирования был выбран iSim, который входит в стандартный пакет разработки Ise Suite 14.7.

Для начала хотелось бы рассмотреть модель диспетчера задач в целом, а затем перейти к его составляющим.

Диспетчер представляет собой объект, на вход которого приходит очередь с данными, а также информация о свободных в данный момент процессорах. Выбирается один из свободных процессоров, которому отправляется сигнал о наличии пакета для обработки (request, начало согласования передачи). Процессор устанавливает сигнал подтверждения запроса, затем диспетчер отправляет ему строб готовности пакета с идентификатором (ready). По этому стробу процессор читает пакет, устанавливает сигнал успешного приема и начинает обработку. Диспетчер начинает очередной цикл назначения задачи, в случае отсутствия свободных процессоров (заняты или очередь задач пуста) переходит в спящий режим.

Рассмотрим структуру диспетчера.

Диспетчер состоит из трех частей. В качестве основной можно выделить блок назначения задач. Он является ключевым звеном в работе диспетчера задач с локальной очередью. Внутри данного блока реализована основная логика: прием на вход задачи от блока генерации задач, а также прием порядкового номера блока очереди FIFO, который в данный момент времени является самым свободным. В процессе обработки входных сигналов блок назначения задач выбирает в какую очередь направить новую задачу. Запись производится путем отправки сигнала S_{wrreq} , а также номера задачи по специальной шине данных.

Для определения блока FIFO, содержащего наименьшее количество задач служит блок выбора очереди. На вход поступают сигналы от каждого блока FIFO, а выходным сигналом является номер процессора, который обладает самой маленькой очередью на исполнение.

Последним элементом, входящим в состав модели диспетчера задач, является блок очереди, который организован по принципу FIFO. Он представляет собой блок памяти 16x8 и пришедшие в него данные находятся там до тех пор, пока синхронизатор задач не пришлёт к контроллеру очереди запрос на чтение – сигнал S_{rdreq} , который поступает от каждого процессора.

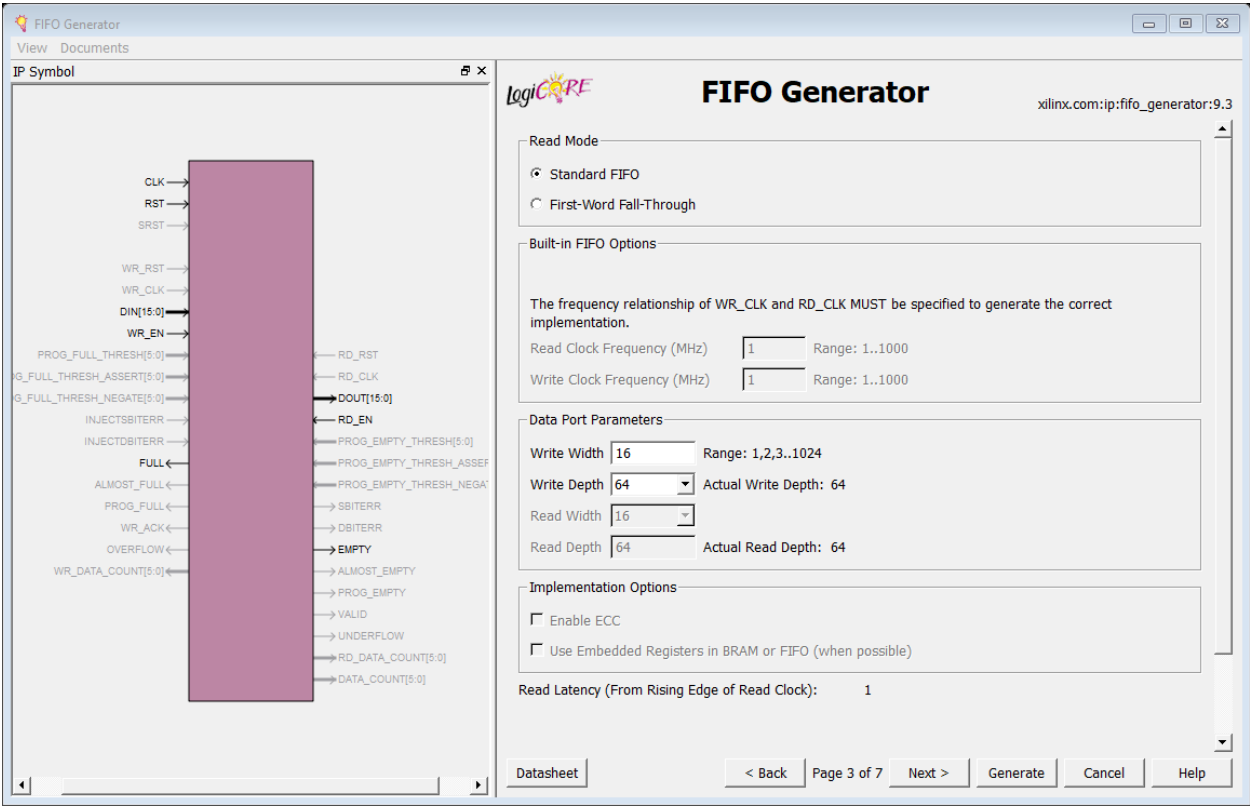


Рисунок 15. Схема блока очереди FIFO.

Процессор

Для тестирования работы диспетчера в симуляторе был написан квазипроцессор (Processor.vhd), который имитирует обработку принятого пакета.

Процессор обрабатывает задачи, полученные из локальной очереди. Вначале, если процессор находится в режиме ожидания, очередь посылает запрос на принятие заявки процессору S_{proc_zapr} . Процессор пробуждается и отвечает, что запрос принял S_{proc_pzapr} . Далее очередь подготавливает задачу и отправляет ее на процессор с сигналом задача готова S_{task_ready} .

Процессор принимает задачу, ставит ее на обработку и отвечает, что задачу принял *S_proc_prinyal*. Затем процессор обрабатывает задачу, а когда задача обработана, посылает запрос на следующую задачу *S_rdreq*. Если задача в очереди есть (сигнал *S_fifo_empty* неактивен), то процессор сразу принимает ее, иначе процессор выдает сигнал, о том, что он свободен *S_proc_free*.

Имитация обработки задачи реализована с помощью вычитающего счетчика, начальное значение которого определяется генератором псевдослучайных чисел, реализующим алгоритм xorshift32.

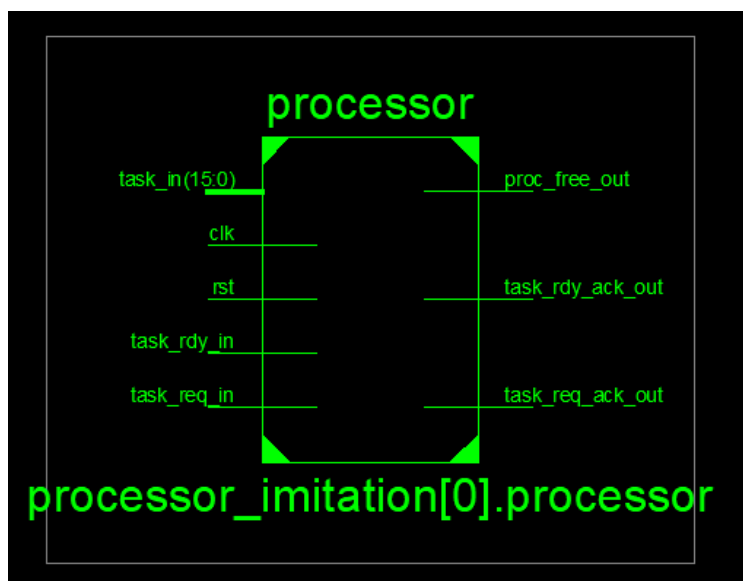


Рисунок16. Схема модели процессора.

Генератор пакетов

Описан в файле *GenerationTask.vhd*. Содержит в себе счетчик, имитирующий наполнение пакетов. Генерация строба валидности приходящих данных осуществляется с использованием регистра сдвига с линейной обратной связью. Пакеты и строб данных подаются на вход блока назначения задач.

4.2 Анализ результатов моделирования

На рисунках в дальнейшем представлен результат моделирования разрабатываемого устройства в виде временных диаграмм с использованием симулятора Xilinx ISE Simulator. События обозначены следующим образом:

1. s_task_arrived – сигнал поступления новой задачи;
2. v_task_id_in – номер заявки;
3. use_1... use_10 – количество заявок в очереди;
4. s_proc_free – сигнал того, что процессор свободен;
5. s_proc_zapr – сигнал, что в очереди к этому процессу появилась заявка;
6. s_task_ready – сигнал того, что заявка из очереди выведена на работу в процессоре;
7. s_proc_work – сигнал занятости процессора;

На рисунке 17 представлено состояние 5 младших процессов, которые начинают работать первыми. Как видно из рисунка, задача удаляется из очереди только после того, как процессор закончил её обрабатывать. Также каждый процессор после окончания обработки заявки подаёт запрос в очередь для следующей заявки.

Для этого моделирования частота заявок составляет 5 тактов, а время обработки одним процессором одной заявки – 60 тактов.

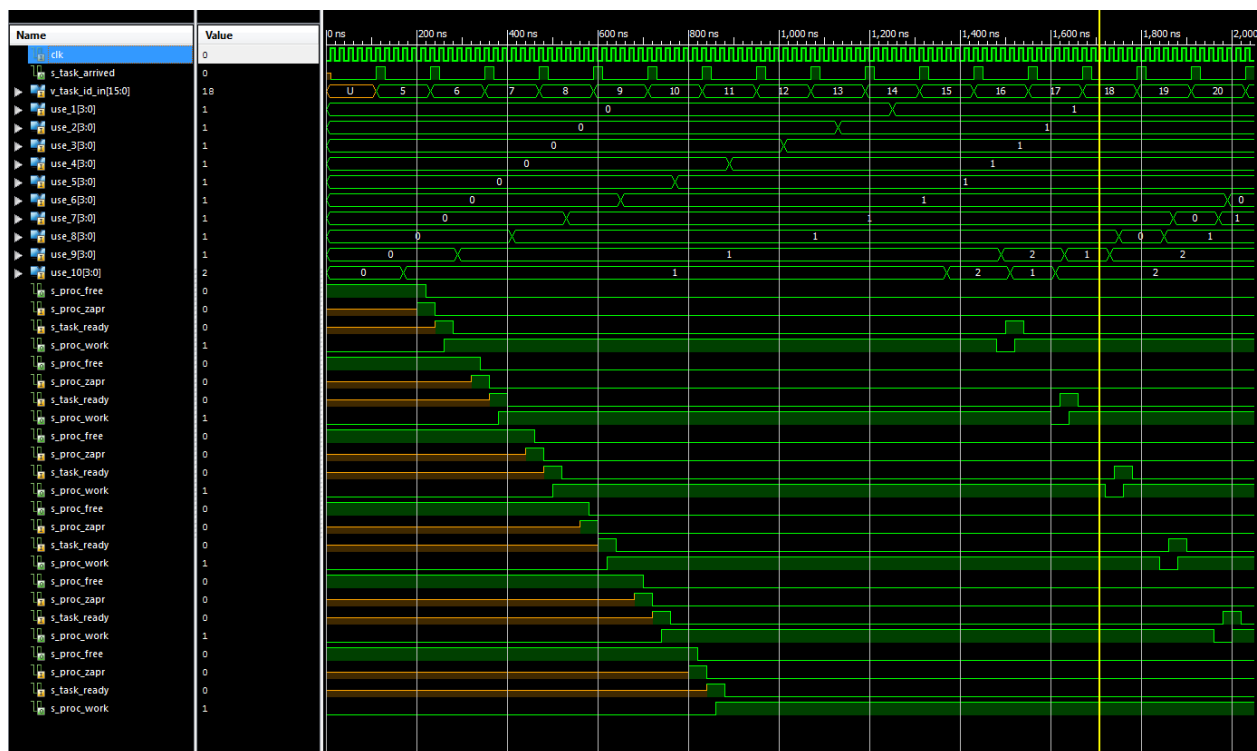


Рисунок 17. Временная диаграмма работы

На рисунке 18 представлен результат моделирования после окончания работы процессора и при отсутствии новой заявки к нему в очереди. Для построения данной ситуации время обработки одной заявки одним процессором было уменьшено до 15 тактов.

На диаграмме также представлено состояние 5 младших процессов, которые начинают работать первыми. Как видно из рисунка, при заданных параметрах достаточно использовать только 4 процессора, а пятый не будет использоваться вообще. Если посмотреть на 960ns, то можно увидеть запрос от процессора на новую задачу, а его очередь пуста (use_7). При этом через 340ns процессор выставляет состояние свободен, поскольку за это время в его очередь не пришла ни одна задача.

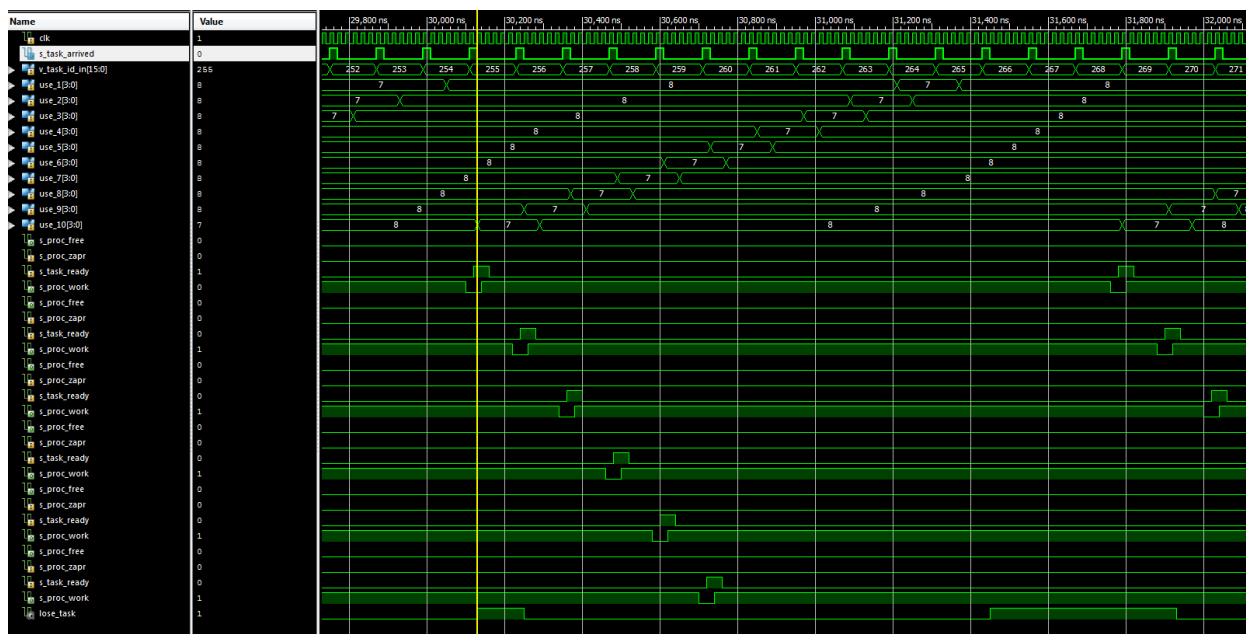


Рисунок 19. Временная диаграмма работы процессоров при ситуации переполнения очередей и потери задачи

Чтобы иметь возможность сравнить результаты работы обоих вариантов реализации аппаратного диспетчера в многопроцессорной системе, при моделировании использовались те же самые блоки имитации работы процессора, и тот же блок генерации задач. Таким образом, условия моделирования системы остались прежними: задачи поступают в систему с периодичностью в 5 тактов, любой процессор обслуживает одну задачу на протяжении 60 тактов.

Главное отличие в результатах заключается в том, что при варианте организации диспетчера с глобальными очередями задачи накапливались в FIFO и постепенно заполняли её. В результате наступал момент, когда, новые идентификаторы не принимались в систему, она становилась перегружена. Время обработки вновь поступившей задачи в этом случае значительно возрастает. При этом в системе постоянно присутствовали свободные процессоры, но возникающий конфликт не позволял им получать задачи.

Планировщик/диспетчер с локальными очередями демонстрирует результат лучше, система успевает принимать и назначать все поступающие

задачи. Она не перегружается на любом отрезке времени, кроме того, способна обрабатывать и более интенсивный поток задач. Причина увеличения быстродействия проста и очевидна – диспетчер во втором случае работает в десять раз быстрее, при этом выполняя все возложенные на него функции. Недостаток планировщика/диспетчера с локальными очередями заключается в отсутствии автоматической балансировки загрузки процессоров. Известно, что примерно 20% всех задач загружают процессоры почти на 80% от всего времени. Из-за этого нельзя оценивать загруженность процессора, основываясь лишь на том, сколько задач присутствует в его собственной очереди. Ведь не исключено, что у одного процессора в очереди может быть десять коротких задач, в то время как у другого их будет всего пять, но они будут занимать намного больше времени, чем десять коротких. Поступает новый процесс, и, её справедливо было бы отправить в очередь процессора с десятью процессами, но в данном случае она еще больше нагрузит второй процессор с пятью трудоемкими процессами. Поэтому при распределении задач по процессорам необходимо учитывать то обстоятельство, сколько по времени выполняется каждый процесс, но это потребует значительного объема дополнительного оборудования или поддержки соответствующими функциями медленно действующей традиционной операционной системы.

ЗАКЛЮЧЕНИЕ

В результате выполнения представленной выпускной квалификационной работы была разработана модель планировщика задач с локальной очередью в многопроцессорной системе. Планировщик направляет поступившую задачу в очередь процессорного узла с наименьшим количеством задач, запускает задачи на обработку в процессоре и удаляет их из системы после обработки.

Анализ временных диаграмм многопроцессорной системы, в которой используется планировщик/диспетчер с локальной очередью задач, позволил выявить существенные преимущества такой организации, повышающие производительность всей системы в целом. Причина кроется в том, что переключение задачи посредством ядра операционной системы реального занимает порядка 1000 тактов, а переключение задачи с помощью аппаратного планировщика задач с локальной очередью производит ту же операцию за 7 тактов.

Проектирование и тестирование экспериментальной модели проводилось в среде разработки Xilinx ISE Design Suite 14.7.

Модель, разработанная и описанная в этой выпускной квалификационной работе, удовлетворяет всем исходным требованиям и содержит все блоки, необходимые для представления работы системы с планировщиком задач с локальной очередью:

- блок выбора очереди;
- n блоков очередей задач FIFO, где n – количество процессорных узлов;
- блок назначения задач.

На данный момент времени проект, описанный в выпускной квалификационной работе, является экспериментальной моделью.

5 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вашкевич Н.П., Бикташев Р.А., Меркурьев А.И., Аппаратная поддержка диспетчера задач с глобальной очередью в многопроцессорных системах. 2011 – с.3-14. Известия высших учебных заведений. Поволжский регион. Технические науки.
2. Вашкевич Н.П., Бикташев Р.А., Формализация алгоритмов управления параллельными процессами на основе использования механизма рандеву. 2011 – с.115-124. XXI век: итоги прошлого и проблемы настоящего плюс.
3. Таненбаум Э., Бос Х., Современные операционные системы. 2015 – 1119 с.
4. Бикташев Р.А., Емельянов Р.И., Структурная реализация и оценка временных характеристик многопроцессорных планировщиков с распределенной очередью задач. – 2017 – с. 66-75. Вестник Пензенского государственного университета №3.
5. Вашкевич Н.П., Бикташев Р.А., Киселев С.В., Разработка устройства планирования процессов для многопроцессорной системы. – 2016 – с. 304-308 Международная научно-техническая и научно-методическая конференция «Современные технологии в науке и образовании». Сборник трудов. Том 1.

ПРИЛОЖЕНИЕ А

ЛИСТИНГИ ПРОГРАММЫ

Приложение А.1 MainBlock.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity MainBlock is
Port ( CLK : in  STD_LOGIC);
end MainBlock;


architecture Behavioral of MainBlock is
component GenerationTask
Port ( CLK : in  STD_LOGIC;
      S_task_arrived : out STD_LOGIC;
      V_task_id_out : out  STD_LOGIC_VECTOR(15 downto 0));

end component;

component UnitTaskAssignments
Port ( CLK : in  STD_LOGIC;
      S_fifo_full_1 : in  STD_LOGIC;
      S_fifo_full_2 : in  STD_LOGIC;
      S_fifo_full_3 : in  STD_LOGIC;
      S_fifo_full_4 : in  STD_LOGIC;
      S_fifo_full_5 : in  STD_LOGIC;
      S_fifo_full_6 : in  STD_LOGIC;
```

```

S_fifo_full_7 : in  STD_LOGIC;
S_fifo_full_8 : in  STD_LOGIC;
S_fifo_full_9 : in  STD_LOGIC;
S_fifo_full_10 : in  STD_LOGIC;
S_task_arrived : in  STD_LOGIC;
V_task_id_in : in  STD_LOGIC_VECTOR(15 downto 0);
id : in STD_LOGIC_VECTOR(3 downto 0);
S_wrreq_1 : out  STD_LOGIC;
    S_wrreq_2 : out  STD_LOGIC;
    S_wrreq_3 : out  STD_LOGIC;
    S_wrreq_4 : out  STD_LOGIC;
    S_wrreq_5 : out  STD_LOGIC;
    S_wrreq_6 : out  STD_LOGIC;
    S_wrreq_7 : out  STD_LOGIC;
    S_wrreq_8 : out  STD_LOGIC;
    S_wrreq_9 : out  STD_LOGIC;
    S_wrreq_10 : out  STD_LOGIC;
    V_task_id_out : out  STD_LOGIC_VECTOR(15 downto 0));
end component;

```

```

component TheSelectionUnitQueue
Port ( CLK : in  STD_LOGIC;
    use_1 : in  STD_LOGIC_VECTOR(3 downto 0);
    use_2 : in  STD_LOGIC_VECTOR(3 downto 0);
    use_3 : in  STD_LOGIC_VECTOR(3 downto 0);
    use_4 : in  STD_LOGIC_VECTOR(3 downto 0);
    use_5 : in  STD_LOGIC_VECTOR(3 downto 0);
    use_6 : in  STD_LOGIC_VECTOR(3 downto 0);
    use_7 : in  STD_LOGIC_VECTOR(3 downto 0);
    use_8 : in  STD_LOGIC_VECTOR(3 downto 0);

```

```

        use_9 : in  STD_LOGIC_VECTOR(3 downto 0);
        use_10 : in  STD_LOGIC_VECTOR(3 downto 0);
        id : out  STD_LOGIC_VECTOR(3 downto 0));
end component;

component FIFO
Port ( CLK : in  STD_LOGIC;
      S_rdreq : in  STD_LOGIC;
      S_wrreq : in  STD_LOGIC;
      S_proc_free : in  STD_LOGIC;
      S_proc_pzapr : in  STD_LOGIC;
      S_proc_prinyal : in  STD_LOGIC;
      V_task_id_in : in  STD_LOGIC_VECTOR(15 downto 0);
      S_fifo_full : out  STD_LOGIC;
      S_fifo_empty : out  STD_LOGIC;
      S_task_ready : out  STD_LOGIC;
      S_proc_zapr : out  STD_LOGIC;
      S_use : out  STD_LOGIC_VECTOR(3 downto 0);
      q : out  STD_LOGIC_VECTOR(15 downto 0));
end component;

component Processor
Port ( CLK : in  STD_LOGIC;
      S_fifo_empty : in  STD_LOGIC;
      S_proc_zapr : in  STD_LOGIC;
      S_task_ready : in  STD_LOGIC;
      q : in  STD_LOGIC_VECTOR(15 downto 0);
      S_proc_free : out  STD_LOGIC;
      S_proc_pzapr : out  STD_LOGIC;
      S_proc_prinyal : out  STD_LOGIC;
      S_proc_work : out  STD_LOGIC;

```

```

        task_id_proc : out  STD_LOGIC_VECTOR(15
downto 0);

        S_rdreq : out  STD_LOGIC);

end component;

signal VAR_S_task_arrived : STD_LOGIC  := '0';

signal VAR_V_task_id_in : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

signal VAR_S_fifo_full_1 : STD_LOGIC  := '0';
signal VAR_S_fifo_full_2 : STD_LOGIC := '0';
signal VAR_S_fifo_full_3 : STD_LOGIC := '0';
signal VAR_S_fifo_full_4 : STD_LOGIC := '0';
signal VAR_S_fifo_full_5 : STD_LOGIC := '0';
signal VAR_S_fifo_full_6 : STD_LOGIC := '0';
signal VAR_S_fifo_full_7 : STD_LOGIC := '0';
signal VAR_S_fifo_full_8 : STD_LOGIC := '0';
signal VAR_S_fifo_full_9 : STD_LOGIC := '0';
signal VAR_S_fifo_full_10 : STD_LOGIC := '0';

signal VAR_id : STD_LOGIC_VECTOR(3 downto 0) := "0000";

signal VAR_S_wrreq_1 : STD_LOGIC := '0';
signal VAR_S_wrreq_2 : STD_LOGIC := '0';
signal VAR_S_wrreq_3 : STD_LOGIC := '0';
signal VAR_S_wrreq_4 : STD_LOGIC := '0';
signal VAR_S_wrreq_5 : STD_LOGIC := '0';
signal VAR_S_wrreq_6 : STD_LOGIC := '0';
signal VAR_S_wrreq_7 : STD_LOGIC := '0';
signal VAR_S_wrreq_8 : STD_LOGIC := '0';
signal VAR_S_wrreq_9 : STD_LOGIC := '0';
signal VAR_S_wrreq_10 : STD_LOGIC := '0';

signal VAR_V_task_id_o : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

signal VAR_use_1 : STD_LOGIC_VECTOR(3 downto 0);

```

```

signal VAR_use_2 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_3 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_4 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_5 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_6 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_7 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_8 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_9 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_10 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_S_rdreq_1 : STD_LOGIC := '0';
signal VAR_S_rdreq_2 : STD_LOGIC := '0';
signal VAR_S_rdreq_3 : STD_LOGIC := '0';
signal VAR_S_rdreq_4 : STD_LOGIC := '0';
signal VAR_S_rdreq_5 : STD_LOGIC := '0';
signal VAR_S_rdreq_6 : STD_LOGIC := '0';
signal VAR_S_rdreq_7 : STD_LOGIC := '0';
signal VAR_S_rdreq_8 : STD_LOGIC := '0';
signal VAR_S_rdreq_9 : STD_LOGIC := '0';
signal VAR_S_rdreq_10 : STD_LOGIC := '0';
signal VAR_S_proc_free_1 : STD_LOGIC := '0';
signal VAR_S_proc_free_2 : STD_LOGIC := '0';
signal VAR_S_proc_free_3 : STD_LOGIC := '0';
signal VAR_S_proc_free_4 : STD_LOGIC := '0';
signal VAR_S_proc_free_5 : STD_LOGIC := '0';
signal VAR_S_proc_free_6 : STD_LOGIC := '0';
signal VAR_S_proc_free_7 : STD_LOGIC := '0';
signal VAR_S_proc_free_8 : STD_LOGIC := '0';
signal VAR_S_proc_free_9 : STD_LOGIC := '0';
signal VAR_S_proc_free_10 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_1 : STD_LOGIC := '0';

```

```

signal VAR_S_proc_pzapr_2 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_3 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_4 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_5 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_6 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_7 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_8 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_9 : STD_LOGIC := '0';
signal VAR_S_proc_pzapr_10 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_1 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_2 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_3 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_4 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_5 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_6 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_7 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_8 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_9 : STD_LOGIC := '0';
signal VAR_S_proc_prinyal_10 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_1 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_2 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_3 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_4 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_5 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_6 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_7 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_8 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_9 : STD_LOGIC := '0';
signal VAR_S_fifo_empty_10 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_1 : STD_LOGIC := '0';

```

```

signal VAR_S_proc_zapr_2 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_3 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_4 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_5 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_6 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_7 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_8 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_9 : STD_LOGIC := '0';
signal VAR_S_proc_zapr_10 : STD_LOGIC := '0';
signal VAR_S_task_ready_1 : STD_LOGIC := '0';
signal VAR_S_task_ready_2 : STD_LOGIC := '0';
signal VAR_S_task_ready_3 : STD_LOGIC := '0';
signal VAR_S_task_ready_4 : STD_LOGIC := '0';
signal VAR_S_task_ready_5 : STD_LOGIC := '0';
signal VAR_S_task_ready_6 : STD_LOGIC := '0';
signal VAR_S_task_ready_7 : STD_LOGIC := '0';
signal VAR_S_task_ready_8 : STD_LOGIC := '0';
signal VAR_S_task_ready_9 : STD_LOGIC := '0';
signal VAR_S_task_ready_10 : STD_LOGIC := '0';

signal VAR_V_task_id_out_1 : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

signal VAR_V_task_id_out_2 : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

signal VAR_V_task_id_out_3 : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

signal VAR_V_task_id_out_4 : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

signal VAR_V_task_id_out_5 : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

signal VAR_V_task_id_out_6 : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

signal VAR_V_task_id_out_7 : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

```

```

signal VAR_V_task_id_out_8 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_V_task_id_out_9 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_V_task_id_out_10 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";


signal VAR_WORK_1 : STD_LOGIC := '0';

signal VAR_ID_WUT_1 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_2 : STD_LOGIC := '0';

signal VAR_ID_WUT_2 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_3 : STD_LOGIC := '0';

signal VAR_ID_WUT_3 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_4 : STD_LOGIC := '0';

signal VAR_ID_WUT_4 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_5 : STD_LOGIC := '0';

signal VAR_ID_WUT_5 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_6 : STD_LOGIC := '0';

signal VAR_ID_WUT_6 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_7 : STD_LOGIC := '0';

signal VAR_ID_WUT_7 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_8 : STD_LOGIC := '0';

signal VAR_ID_WUT_8 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_9 : STD_LOGIC := '0';

signal VAR_ID_WUT_9 : STD_LOGIC_VECTOR(15 downto 0) :=
"00000000000000000";

signal VAR_WORK_10 : STD_LOGIC := '0';

```

```

signal VAR_ID_WUT_10 : STD_LOGIC_VECTOR(15 downto 0) :=
"0000000000000000";

begin

Generation1: GenerationTask port map
(CLK,VAR_S_task_arrived,VAR_V_task_id_in);

UnitTaskAssignments1: UnitTaskAssignments port map (CLK,
VAR_S_fifo_full_1,VAR_S_fifo_full_2, VAR_S_fifo_full_3,
VAR_S_fifo_full_4,

VAR_S_fifo_full_5, VAR_S_fifo_full_6, VAR_S_fifo_full_7,
VAR_S_fifo_full_8, VAR_S_fifo_full_9, VAR_S_fifo_full_10,

VAR_S_task_arrived, VAR_V_task_id_in, VAR_id, VAR_S_wrreq_1,
VAR_S_wrreq_2, VAR_S_wrreq_3, VAR_S_wrreq_4, VAR_S_wrreq_5,
VAR_S_wrreq_6, VAR_S_wrreq_7,

VAR_S_wrreq_8, VAR_S_wrreq_9, VAR_S_wrreq_10, VAR_V_task_id_o);

TheSelectionUnitQueue1: TheSelectionUnitQueue port map
(CLK,VAR_use_1,VAR_use_2,VAR_use_3,VAR_use_4,VAR_use_5,VAR_use_6
,VAR_use_7,VAR_use_8,VAR_use_9,VAR_use_10,VAR_id);

FIFO1: FIFO port map
(CLK,VAR_S_rdreq_1,VAR_S_wrreq_1,VAR_S_proc_free_1,VAR_S_proc_pz
apr_1,VAR_S_proc_prinyal_1,VAR_V_task_id_o,VAR_S_fifo_full_1,VAR
_S_fifo_empty_1,VAR_S_task_ready_1,VAR_S_proc_zapr_1,VAR_use_1,
VAR_V_task_id_out_1);

FIFO2: FIFO port map
(CLK,VAR_S_rdreq_2,VAR_S_wrreq_2,VAR_S_proc_free_2,VAR_S_proc_pz
apr_2,VAR_S_proc_prinyal_2,VAR_V_task_id_o,VAR_S_fifo_full_2,VAR
_S_fifo_empty_2,VAR_S_task_ready_2,VAR_S_proc_zapr_2,VAR_use_2,
VAR_V_task_id_out_2);

FIFO3: FIFO port map
(CLK,VAR_S_rdreq_3,VAR_S_wrreq_3,VAR_S_proc_free_3,VAR_S_proc_pz
apr_3,VAR_S_proc_prinyal_3,VAR_V_task_id_o,VAR_S_fifo_full_3,VAR
_S_fifo_empty_3,VAR_S_task_ready_3,VAR_S_proc_zapr_3,VAR_use_3,
VAR_V_task_id_out_3);

FIFO4: FIFO port map
(CLK,VAR_S_rdreq_4,VAR_S_wrreq_4,VAR_S_proc_free_4,VAR_S_proc_pz
apr_4,VAR_S_proc_prinyal_4,VAR_V_task_id_o,VAR_S_fifo_full_4,VAR

```

					ПГУ 2.090301.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		47

_S_fifo_empty_4,VAR_S_task_ready_4,VAR_S_proc_zapr_4,VAR_use_4,
VAR_V_task_id_out_4);

FIFO5: FIFO port map

(CLK,VAR_S_rdreq_5,VAR_S_wrreq_5,VAR_S_proc_free_5,VAR_S_proc_pz
apr_5,VAR_S_proc_prinyal_5,VAR_V_task_id_o,VAR_S_fifo_full_5,VAR
_S_fifo_empty_5,VAR_S_task_ready_5,VAR_S_proc_zapr_5,VAR_use_5,
VAR_V_task_id_out_5);

FIFO6: FIFO port map

(CLK,VAR_S_rdreq_6,VAR_S_wrreq_6,VAR_S_proc_free_6,VAR_S_proc_pz
apr_6,VAR_S_proc_prinyal_6,VAR_V_task_id_o,VAR_S_fifo_full_6,VAR
_S_fifo_empty_6,VAR_S_task_ready_6,VAR_S_proc_zapr_6,VAR_use_6,
VAR_V_task_id_out_6);

FIFO7: FIFO port map

(CLK,VAR_S_rdreq_7,VAR_S_wrreq_7,VAR_S_proc_free_7,VAR_S_proc_pz
apr_7,VAR_S_proc_prinyal_7,VAR_V_task_id_o,VAR_S_fifo_full_7,VAR
_S_fifo_empty_7,VAR_S_task_ready_7,VAR_S_proc_zapr_7,VAR_use_7,
VAR_V_task_id_out_7);

FIFO8: FIFO port map

(CLK,VAR_S_rdreq_8,VAR_S_wrreq_8,VAR_S_proc_free_8,VAR_S_proc_pz
apr_8,VAR_S_proc_prinyal_8,VAR_V_task_id_o,VAR_S_fifo_full_8,VAR
_S_fifo_empty_8,VAR_S_task_ready_8,VAR_S_proc_zapr_8,VAR_use_8,
VAR_V_task_id_out_8);

FIFO9: FIFO port map

(CLK,VAR_S_rdreq_9,VAR_S_wrreq_9,VAR_S_proc_free_9,VAR_S_proc_pz
apr_9,VAR_S_proc_prinyal_9,VAR_V_task_id_o,VAR_S_fifo_full_9,VAR
_S_fifo_empty_9,VAR_S_task_ready_9,VAR_S_proc_zapr_9,VAR_use_9,
VAR_V_task_id_out_9);

FIFO10: FIFO port map

(CLK,VAR_S_rdreq_10,VAR_S_wrreq_10,VAR_S_proc_free_10,VAR_S_proc
_pzapr_10,VAR_S_proc_prinyal_10,VAR_V_task_id_o,VAR_S_fifo_full_
10,VAR_S_fifo_empty_10,VAR_S_task_ready_10,VAR_S_proc_zapr_10,VA
R_use_10, VAR_V_task_id_out_10);

Processor1: Processor port map

(CLK,VAR_S_fifo_empty_1,VAR_S_proc_zapr_1,VAR_S_task_ready_1,VAR
_V_task_id_out_1,VAR_S_proc_free_1,VAR_S_proc_pzapr_1,VAR_S_proc
_prinyal_1,VAR_WORK_1,VAR_ID_WUT_1,VAR_S_rdreq_1);

Processor2: Processor port map

(CLK,VAR_S_fifo_empty_2,VAR_S_proc_zapr_2,VAR_S_task_ready_2,VAR
_V_task_id_out_2,VAR_S_proc_free_2,VAR_S_proc_pzapr_2,VAR_S_proc
_prinyal_2,VAR_WORK_2,VAR_ID_WUT_2,VAR_S_rdreq_2);

Processor3: Processor port map

(CLK,VAR_S_fifo_empty_3,VAR_S_proc_zapr_3,VAR_S_task_ready_3,VAR

```
_V_task_id_out_3,VAR_S_proc_free_3,VAR_S_proc_pzapr_3,VAR_S_proc
_prinyal_3,VAR_WORK_3,VAR_ID_WUT_3,VAR_S_rdreq_3);
```

Processor4: Processor port map

```
(CLK,VAR_S_fifo_empty_4,VAR_S_proc_zapr_4,VAR_S_task_ready_4,VAR
_V_task_id_out_4,VAR_S_proc_free_4,VAR_S_proc_pzapr_4,VAR_S_proc
_prinyal_4,VAR_WORK_4,VAR_ID_WUT_4,VAR_S_rdreq_4);
```

Processor5: Processor port map

```
(CLK,VAR_S_fifo_empty_5,VAR_S_proc_zapr_5,VAR_S_task_ready_5,VAR
_V_task_id_out_5,VAR_S_proc_free_5,VAR_S_proc_pzapr_5,VAR_S_proc
_prinyal_5,VAR_WORK_5,VAR_ID_WUT_5,VAR_S_rdreq_5);
```

Processor6: Processor port map

```
(CLK,VAR_S_fifo_empty_6,VAR_S_proc_zapr_6,VAR_S_task_ready_6,VAR
_V_task_id_out_6,VAR_S_proc_free_6,VAR_S_proc_pzapr_6,VAR_S_proc
_prinyal_6,VAR_WORK_6,VAR_ID_WUT_6,VAR_S_rdreq_6);
```

Processor7: Processor port map

```
(CLK,VAR_S_fifo_empty_7,VAR_S_proc_zapr_7,VAR_S_task_ready_7,VAR
_V_task_id_out_7,VAR_S_proc_free_7,VAR_S_proc_pzapr_7,VAR_S_proc
_prinyal_7,VAR_WORK_7,VAR_ID_WUT_7,VAR_S_rdreq_7);
```

Processor8: Processor port map

```
(CLK,VAR_S_fifo_empty_8,VAR_S_proc_zapr_8,VAR_S_task_ready_8,VAR
_V_task_id_out_8,VAR_S_proc_free_8,VAR_S_proc_pzapr_8,VAR_S_proc
_prinyal_8,VAR_WORK_8,VAR_ID_WUT_8,VAR_S_rdreq_8);
```

Processor9: Processor port map

```
(CLK,VAR_S_fifo_empty_9,VAR_S_proc_zapr_9,VAR_S_task_ready_9,VAR
_V_task_id_out_9,VAR_S_proc_free_9,VAR_S_proc_pzapr_9,VAR_S_proc
_prinyal_9,VAR_WORK_9,VAR_ID_WUT_9,VAR_S_rdreq_9);
```

Processor10: Processor port map

```
(CLK,VAR_S_fifo_empty_10,VAR_S_proc_zapr_10,VAR_S_task_ready_10,
VAR_V_task_id_out_10,VAR_S_proc_free_10,VAR_S_proc_pzapr_10,VAR_
S_proc_prinyal_10,VAR_WORK_10,VAR_ID_WUT_10,VAR_S_rdreq_10);
```

```
process (CLK)
```

```
begin
```

```
if (CLK'event and CLK='1') then
```

```
end if;
```

```
end process;
```

```
--
```

```
--
```

```
end Behavioral;
```

					ПГУ 2.090301.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		49

Приложение А.2 GenerationTask.vhd

```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.std_logic_arith.all;

entity GenerationTask is
    generic (
        --Задержка
        g_Delay      : integer := 5
    );
    Port ( CLK : in  STD_LOGIC;
          S_task_arrived : out STD_LOGIC;
          V_task_id_out  : out  STD_LOGIC_VECTOR(15 downto 0));
end GenerationTask;

architecture Behavioral of GenerationTask is
    signal VAR_WAIT : integer := 0;
    signal VAR_TASK : integer range 0 to 65536 := 5;
begin
    process (CLK)
    begin
        if (CLK'event and CLK='1') then
            if (VAR_WAIT = g_Delay) then
                VAR_TASK <= VAR_TASK +1;
                V_task_id_out <= conv_std_logic_vector
(VAR_TASK,16);
                S_task_arrived <= '1';
                VAR_WAIT <= 0;
            end if;
        end if;
    end process;
end Behavioral;

```

					ПГУ 2.090301.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		50

```

else
    S_task_arrived <= '0';
    VAR_WAIT <= VAR_WAIT + 1;
end if;
end if;
end process;

end Behavioral;

```

					ПГУ 2.090301.15.001 ПЗ	Лист
						51
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.3 UnitTaskAssignments.vhd

```
library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity UnitTaskAssignments is
    Port ( CLK : in  STD_LOGIC;

          S_fifo_full_1 : in  STD_LOGIC;
          S_fifo_full_2 : in  STD_LOGIC;
          S_fifo_full_3 : in  STD_LOGIC;
          S_fifo_full_4 : in  STD_LOGIC;
          S_fifo_full_5 : in  STD_LOGIC;
          S_fifo_full_6 : in  STD_LOGIC;
          S_fifo_full_7 : in  STD_LOGIC;
          S_fifo_full_8 : in  STD_LOGIC;
          S_fifo_full_9 : in  STD_LOGIC;
          S_fifo_full_10 : in  STD_LOGIC;
          S_task_arrived : in  STD_LOGIC;
          V_task_id_in : in  STD_LOGIC_VECTOR(15 downto 0);
          id : in STD_LOGIC_VECTOR(3 downto 0);

          S_wrreq_1 : out  STD_LOGIC;
          S_wrreq_2 : out  STD_LOGIC;
          S_wrreq_3 : out  STD_LOGIC;
          S_wrreq_4 : out  STD_LOGIC;
          S_wrreq_5 : out  STD_LOGIC;
          S_wrreq_6 : out  STD_LOGIC;
          S_wrreq_7 : out  STD_LOGIC;
```

Изм.	Лист	№ докум.	Подпись	Дата

```

        S_wrreq_8 : out  STD_LOGIC;

        S_wrreq_9 : out  STD_LOGIC;

        S_wrreq_10 : out  STD_LOGIC;

        V_task_id_out : out  STD_LOGIC_VECTOR(15 downto 0));
end UnitTaskAssignments;

architecture Behavioral of UnitTaskAssignments is
signal VAR_S_fifo_full_1 : STD_LOGIC;
signal VAR_S_fifo_full_2 : STD_LOGIC;
signal VAR_S_fifo_full_3 : STD_LOGIC;
signal VAR_S_fifo_full_4 : STD_LOGIC;
signal VAR_S_fifo_full_5 : STD_LOGIC;
signal VAR_S_fifo_full_6 : STD_LOGIC;
signal VAR_S_fifo_full_7 : STD_LOGIC;
signal VAR_S_fifo_full_8 : STD_LOGIC;
signal VAR_S_fifo_full_9 : STD_LOGIC;
signal VAR_S_fifo_full_10 : STD_LOGIC;
signal VAR_S_task_arrived : STD_LOGIC;
signal VAR_V_task_id_in : STD_LOGIC_VECTOR(15 downto 0);
signal VAR_id : STD_LOGIC_VECTOR(3 downto 0);
signal LOSE_TASK : STD_LOGIC := '0';

begin

        process (CLK)
        begin

                if (CLK'event and CLK='1') then

                        VAR_S_fifo_full_1<=S_fifo_full_1;
                        VAR_S_fifo_full_2<=S_fifo_full_2;
                        VAR_S_fifo_full_3<=S_fifo_full_3;
                        VAR_S_fifo_full_4<=S_fifo_full_4;
                        VAR_S_fifo_full_5<=S_fifo_full_5;

```

```

VAR_S_fifo_full_6<=S_fifo_full_6;
VAR_S_fifo_full_7<=S_fifo_full_7;
VAR_S_fifo_full_8<=S_fifo_full_8;
VAR_S_fifo_full_9<=S_fifo_full_9;
VAR_S_fifo_full_10<=S_fifo_full_10;
VAR_S_task_arrived<=S_task_arrived;
VAR_V_task_id_in<=V_task_id_in;
VAR_id<=id;

end if;

end process;

process (VAR_S_task_arrived)
begin
    if (VAR_S_task_arrived='0') then

        S_wrreq_1 <= '0';
        S_wrreq_2 <= '0';
        S_wrreq_3 <= '0';
        S_wrreq_4 <= '0';
        S_wrreq_5 <= '0';
        S_wrreq_6 <= '0';
        S_wrreq_7 <= '0';
        S_wrreq_8 <= '0';
        S_wrreq_9 <= '0';
        S_wrreq_10 <= '0';

    elsif (VAR_S_task_arrived='1') then
        V_task_id_out<=VAR_V_task_id_in;

        if (VAR_S_fifo_full_1 = '1' and
VAR_S_fifo_full_2 = '1' and VAR_S_fifo_full_3 = '1' and
VAR_S_fifo_full_4 = '1' and VAR_S_fifo_full_5 = '1' and
VAR_S_fifo_full_6 = '1' and VAR_S_fifo_full_7 = '1' and

```

```

VAR_S_fifo_full_8 = '1' and VAR_S_fifo_full_9 = '1' and
VAR_S_fifo_full_10 = '1') then

                                LOSE_TASK <= '1';

                                elsif (VAR_id = "0001" and
VAR_S_fifo_full_1 = '0') then

                                S_wrreq_1 <= '1';

                                LOSE_TASK <= '0';

                                elsif (VAR_id = "0010" and
VAR_S_fifo_full_2 = '0') then

                                S_wrreq_2 <= '1';

                                LOSE_TASK <= '0';

                                elsif (VAR_id = "0011" and
VAR_S_fifo_full_3 = '0') then

                                S_wrreq_3 <= '1';

                                LOSE_TASK <= '0';

                                elsif (VAR_id = "0100" and
VAR_S_fifo_full_4 = '0') then

                                S_wrreq_4 <= '1';

                                LOSE_TASK <= '0';

                                elsif (VAR_id = "0101" and
VAR_S_fifo_full_5 = '0') then

                                S_wrreq_5 <= '1';

                                LOSE_TASK <= '0';

                                elsif (VAR_id = "0110" and
VAR_S_fifo_full_6 = '0') then

                                S_wrreq_6 <= '1';

                                LOSE_TASK <= '0';

                                elsif (VAR_id = "0111" and
VAR_S_fifo_full_7 = '0') then

                                S_wrreq_7 <= '1';

                                LOSE_TASK <= '0';

                                elsif (VAR_id = "1000" and
VAR_S_fifo_full_8 = '0') then

```

```

S_wrreq_8 <= '1';
LOSE_TASK <= '0';
elseif (VAR_id = "1001" and
VAR_S_fifo_full_9 = '0') then
S_wrreq_9 <= '1';
LOSE_TASK <= '0';
elseif (VAR_id = "1010" and
VAR_S_fifo_full_10 = '0') then
S_wrreq_10 <= '1';
LOSE_TASK <= '0';
end if;
end if;
end process;
end Behavioral;
```

Приложение А.4 TheSelectionUnitQueue.vhd

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity TheSelectionUnitQueue is
    Port ( CLK : in  STD_LOGIC;
          use_1 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_2 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_3 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_4 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_5 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_6 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_7 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_8 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_9 : in  STD_LOGIC_VECTOR(3 downto 0);
          use_10 : in  STD_LOGIC_VECTOR(3 downto 0);
          id : out  STD_LOGIC_VECTOR(3 downto 0));
end TheSelectionUnitQueue;


architecture Behavioral of TheSelectionUnitQueue is
    signal VAR_use_1 : STD_LOGIC_VECTOR(3 downto 0);
    signal VAR_use_2 : STD_LOGIC_VECTOR(3 downto 0);
    signal VAR_use_3 : STD_LOGIC_VECTOR(3 downto 0);
    signal VAR_use_4 : STD_LOGIC_VECTOR(3 downto 0);
    signal VAR_use_5 : STD_LOGIC_VECTOR(3 downto 0);

```

```

signal VAR_use_6 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_7 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_8 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_9 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_use_10 : STD_LOGIC_VECTOR(3 downto 0);
signal VAR_id : STD_LOGIC_VECTOR(3 downto 0);

begin

    process (CLK)
    begin
        if (CLK'event and CLK='1') then
            VAR_use_1<=use_1;
            VAR_use_2<=use_2;
            VAR_use_3<=use_3;
            VAR_use_4<=use_4;
            VAR_use_5<=use_5;
            VAR_use_6<=use_6;
            VAR_use_7<=use_7;
            VAR_use_8<=use_8;
            VAR_use_9<=use_9;
            VAR_use_10<=use_10;

        else
            --
            if (VAR_use_1 <= VAR_use_2 and
VAR_use_1 <= VAR_use_3 and VAR_use_1 <= VAR_use_4 and VAR_use_1
<= VAR_use_5 and VAR_use_1 <= VAR_use_6 and VAR_use_1 <=
VAR_use_7 and VAR_use_1 <= VAR_use_8 and VAR_use_1 <= VAR_use_9
and VAR_use_1 <= VAR_use_10) then
                VAR_id <= "0001";
            end if;

            if (VAR_use_2 <= VAR_use_1 and
VAR_use_2 <= VAR_use_3 and VAR_use_2 <= VAR_use_4 and VAR_use_2
<= VAR_use_5 and VAR_use_2 <= VAR_use_6 and VAR_use_2 <=

```

```

VAR_use_7 and VAR_use_2 <= VAR_use_8 and VAR_use_2 <= VAR_use_9
and VAR_use_2 <= VAR_use_10) then

    VAR_id <= "0010";

end if;

    if (VAR_use_3 <= VAR_use_1 and
VAR_use_3 <= VAR_use_2 and VAR_use_3 <= VAR_use_4 and VAR_use_3
<= VAR_use_5 and VAR_use_3 <= VAR_use_6 and VAR_use_3 <=
VAR_use_7 and VAR_use_3 <= VAR_use_8 and VAR_use_3 <= VAR_use_9
and VAR_use_3 <= VAR_use_10) then

    VAR_id <= "0011";

end if;

    if (VAR_use_4 <= VAR_use_1 and
VAR_use_4 <= VAR_use_2 and VAR_use_4 <= VAR_use_3 and VAR_use_4
<= VAR_use_5 and VAR_use_4 <= VAR_use_6 and VAR_use_4 <=
VAR_use_7 and VAR_use_4 <= VAR_use_8 and VAR_use_4 <= VAR_use_9
and VAR_use_4 <= VAR_use_10) then

    VAR_id <= "0100";

end if;

    if (VAR_use_5 <= VAR_use_1 and
VAR_use_5 <= VAR_use_2 and VAR_use_5 <= VAR_use_3 and VAR_use_5
<= VAR_use_4 and VAR_use_5 <= VAR_use_6 and VAR_use_5 <=
VAR_use_7 and VAR_use_5 <= VAR_use_8 and VAR_use_5 <= VAR_use_9
and VAR_use_5 <= VAR_use_10) then

    VAR_id <= "0101";

end if;

    if (VAR_use_6 <= VAR_use_1 and
VAR_use_6 <= VAR_use_2 and VAR_use_6 <= VAR_use_3 and VAR_use_6
<= VAR_use_4 and VAR_use_6 <= VAR_use_5 and VAR_use_6 <=
VAR_use_7 and VAR_use_6 <= VAR_use_8 and VAR_use_6 <= VAR_use_9
and VAR_use_6 <= VAR_use_10) then

    VAR_id <= "0110";

end if;

    if (VAR_use_7 <= VAR_use_1 and
VAR_use_7 <= VAR_use_2 and VAR_use_7 <= VAR_use_3 and VAR_use_7
<= VAR_use_4 and VAR_use_7 <= VAR_use_5 and VAR_use_7 <=
VAR_use_6 and VAR_use_7 <= VAR_use_8 and VAR_use_7 <= VAR_use_9
and VAR_use_7 <= VAR_use_10) then

    VAR_id <= "0111";

```

```

        end if;

        if (VAR_use_8 <= VAR_use_1 and
VAR_use_8 <= VAR_use_2 and VAR_use_8 <= VAR_use_3 and VAR_use_8
<= VAR_use_4 and VAR_use_8 <= VAR_use_5 and VAR_use_8 <=
VAR_use_6 and VAR_use_8 <= VAR_use_7 and VAR_use_8 <= VAR_use_9
and VAR_use_8 <= VAR_use_10) then

            VAR_id <= "1000";

        end if;

        if (VAR_use_9 <= VAR_use_1 and
VAR_use_9 <= VAR_use_2 and VAR_use_9 <= VAR_use_3 and VAR_use_9
<= VAR_use_4 and VAR_use_9 <= VAR_use_5 and VAR_use_9 <=
VAR_use_6 and VAR_use_9 <= VAR_use_7 and VAR_use_9 <= VAR_use_8
and VAR_use_9 <= VAR_use_10) then

            VAR_id <= "1001";

        end if;

        if (VAR_use_10 <= VAR_use_1 and
VAR_use_10 <= VAR_use_2 and VAR_use_10 <= VAR_use_3 and
VAR_use_10 <= VAR_use_4 and VAR_use_10 <= VAR_use_5 and
VAR_use_10 <= VAR_use_6 and VAR_use_10 <= VAR_use_7 and
VAR_use_10 <= VAR_use_8 and VAR_use_10 <= VAR_use_9) then

            VAR_id <= "1010";

        end if;

        --

    end if;

end process;

id<=VAR_id;

end Behavioral;

```

Приложение А.5 FIFO.vhd

```

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity FIFO is

    Port ( CLK : in  STD_LOGIC;

          S_rdreq : in  STD_LOGIC;

          S_wrreq : in  STD_LOGIC;

          S_proc_free : in  STD_LOGIC;

          S_proc_pzapr : in  STD_LOGIC;

          S_proc_prinyal : in  STD_LOGIC;

          V_task_id_in : in  STD_LOGIC_VECTOR(15 downto 0);

          S_fifo_full : out  STD_LOGIC;

          S_fifo_empty : out  STD_LOGIC;

          S_task_ready : out  STD_LOGIC;

          S_proc_zapr : out  STD_LOGIC;

          S_use : out  STD_LOGIC_VECTOR(3 downto 0);

          q : out  STD_LOGIC_VECTOR(15 downto 0));

end FIFO;


architecture Behavioral of FIFO is

    component FIFO_only

        Port ( CLK : in  STD_LOGIC;

              i_wr_en : in  std_logic;

              i_wr_data : in  std_logic_vector(15 downto

0);

              i_rd_en : in  std_logic;

```

```

                                o_rd_data : out std_logic_vector(15 downto
0);

                                o_full      : out std_logic;

                                o_empty     : out std_logic;

                                o_fifo_count : out std_logic_vector(3 downto
0));

    end component;

    signal VAR_S_rdreq : STD_LOGIC;
    signal VAR_S_wrreq : STD_LOGIC;
    signal VAR_S_proc_free : STD_LOGIC;
    signal VAR_S_proc_pzapr : STD_LOGIC;
    signal VAR_S_proc_prinyal : STD_LOGIC;
    signal VAR_S_fifo_empty : STD_LOGIC;
    signal VAR_id : STD_LOGIC_VECTOR(15 downto 0);

    signal VAR_wr_fifo : STD_LOGIC;

begin

    FIFO_on: FIFO_only port map (CLK, VAR_S_wrreq, VAR_id,
VAR_S_rdreq, q, S_fifo_full, VAR_S_fifo_empty, S_use);

        process (CLK)
        begin
            if (CLK'event and CLK='1') then

                VAR_S_rdreq <= S_rdreq;
                VAR_S_wrreq <= S_wrreq;
                VAR_S_proc_free <=
S_proc_free;
                VAR_S_proc_pzapr <=
S_proc_pzapr;
                VAR_S_proc_prinyal <=
S_proc_prinyal;
                VAR_id <= V_task_id_in;
            end if;
        end process;
    end begin;

```

```

else

VAR_S_fifo_empty = '0') then
    if (VAR_S_rdreq = '1' and
        S_task_ready <= '1';
    end if;

VAR_S_fifo_empty = '0') then
    if (VAR_S_proc_free = '1' and
        S_proc_zapr <= '1';
    end if;

then
    if (VAR_S_proc_pzapr = '1')
        S_proc_zapr <= '0';
        S_task_ready <= '1';
    end if;

then
    if (VAR_S_proc_prinyal = '1')
        --?
        S_task_ready <= '0';
    end if;

        end if;
    end process;

S_fifo_empty <= VAR_S_fifo_empty;

```

```
--process ()  
--begin  
  
--end process;  
  
end Behavioral;
```

					ПГУ 2.090301.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		64

Приложение А.6 FIFO_only.vhd

```

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

entity FIFO_only is
    generic (
        --ширина
        g_WIDTH      : natural := 16;
        --глубина
        g_DEPTH      : integer := 8
    );
    port (
        CLK          : in std_logic;
        i_wr_en      : in  std_logic;
        i_wr_data    : in  std_logic_vector(g_WIDTH-1 downto 0);
        i_rd_en      : in  std_logic;
        o_rd_data    : out std_logic_vector(g_WIDTH-1 downto 0);
        o_full       : out std_logic;
        o_empty      : out std_logic;
        o_fifo_count : out std_logic_vector(3 downto 0)
    );
end FIFO_only;

architecture Behavioral of FIFO_only is
    --Двумерный массив для очереди
    type t_FIFO_DATA is array (0 to g_DEPTH-1) of
        std_logic_vector(g_WIDTH-1 downto 0);

```

```

    signal VAR_FIFO_DATA : t_FIFO_DATA := (others => (others =>
'0')));

    -- индексы записи и чтения
    signal VAR_WR_INDEX   : integer range 0 to g_DEPTH-1 := 0;
    signal VAR_RD_INDEX   : integer range 0 to g_DEPTH-1 := 0;

    -- Текущее количество элементов в очереди
    signal VAR_FIFO_COUNT : integer range -1 to g_DEPTH+1 := 0;

    -- очередь заполнена или пуста
    signal VAR_FULL       : std_logic;
    signal VAR_EMPTY      : std_logic;

begin

    p_CONTROL : process (CLK) is
    begin
        if (CLK'event and CLK='1') then

            -- Отслеживание количества заданий в очереди
            if (i_wr_en = '1' and (not i_rd_en = '1')) then
VAR_FIFO_COUNT <= VAR_FIFO_COUNT + 1;
            elsif (i_wr_en = '0' and (not i_rd_en = '0')) then
                VAR_FIFO_COUNT <= VAR_FIFO_COUNT - 1;
            end if;

            -- Сдвиг индекса записи на единицу по кольцевому
            буферу
            if (i_wr_en = '1' and VAR_FULL = '0') then

```

```

        if (VAR_WR_INDEX = g_DEPTH-1) then
            VAR_WR_INDEX <= 0;
        else
            VAR_WR_INDEX <= VAR_WR_INDEX + 1;
        end if;
    end if;

    -- Сдвиг индекса чтения на единицу по кольцевому
    буферу
    if (i_rd_en = '1' and VAR_EMPTY = '0') then
        if (VAR_RD_INDEX = g_DEPTH-1) then
            VAR_RD_INDEX <= 0;
        else
            VAR_RD_INDEX <= VAR_RD_INDEX + 1;
        end if;
    end if;

    -- Записываем данные по индексу записи
    if (i_wr_en = '1') then
        VAR_FIFO_DATA(VAR_WR_INDEX) <= i_wr_data;
    end if;

    -- Выставляем флаги
    if (VAR_FIFO_COUNT = g_DEPTH) then
        VAR_FULL <= '1';
    else
        VAR_FULL <= '0';
    end if;

    if (VAR_FIFO_COUNT = 0) then

```

```

            VAR_EMPTY  <= '1';

        else

            VAR_EMPTY  <= '0';

        end if;

    end if;

end process p_CONTROL;


-- Считываем данные по индексу чтения
o_rd_data <= VAR_FIFO_DATA(VAR_RD_INDEX);
o_full   <= VAR_FULL;
o_empty  <= VAR_EMPTY;
o_fifo_count <= conv_std_logic_vector (VAR_FIFO_COUNT,4);


end Behavioral;

```

Приложение А.7 Processor.vhd

```

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

entity Processor is
    Port ( CLK : in  STD_LOGIC;

          S_fifo_empty : in  STD_LOGIC;
          S_proc_zapr : in  STD_LOGIC;
          S_task_ready : in  STD_LOGIC;
          q : in  STD_LOGIC_VECTOR(15 downto 0);
          S_proc_free : out  STD_LOGIC;
          S_proc_pzapr : out  STD_LOGIC;
          S_proc_prinyal : out  STD_LOGIC;
          S_proc_work : out  STD_LOGIC;
          task_id_proc : out  STD_LOGIC_VECTOR(15
downto 0);

          S_rdreq : out  STD_LOGIC);

end Processor;

architecture Behavioral of Processor is
    signal VAR_S_fifo_empty : STD_LOGIC;
    signal VAR_S_proc_zapr : STD_LOGIC;
    signal VAR_S_task_ready : STD_LOGIC;
    signal VAR_S_proc_free : STD_LOGIC := '1';
    signal VAR_S_proc_work : STD_LOGIC := '0';
    signal VAR_WAIT : integer range 0 to 33 := 0;
    signal VAR_id : STD_LOGIC_VECTOR(15 downto 0);
    signal VARNUM : integer range 0 to 100000 := 0;

```

```

signal g_Delay  : integer range 0 to 100000 := 60;

begin

    process (CLK)
    begin
        if (CLK'event and CLK='1') then
            VAR_S_fifo_empty <= S_fifo_empty;
            VAR_S_proc_zapr <= S_proc_zapr;
            VAR_S_task_ready <= S_task_ready;
            S_rdreq <= '0';
        else
            --Если запрос, то отвечает, что
серв не отвалился

            if (VAR_S_proc_zapr = '1') then
                VAR_S_proc_free <= '0';
                S_proc_pzapr <= '1';
            end if;

            --Если задача готова, то принимаем
задачу и отвечаем, что приняли, начинаем обработку

            if (VAR_S_task_ready = '1') then
                S_rdreq <= '0';
                S_proc_pzapr <= '0';
                VAR_id <= q;
                S_proc_prinyal <= '1';
                VAR_S_proc_work <= '1';

                if (VARNUM = 300) then
                    g_Delay <= 45;
                    VARNUM <= VARNUM + 1;
                else

```

Изм.	Лист	№ докум.	Подпись	Дата

```

        VARNUM <= VARNUM + 1;

        end if;

    end if;

    --Обработка задачи процессором
    if (VAR_S_proc_work = '1') then
        --Если отработали, то
        процессор закончил обработку

        if (VAR_WAIT = g_Delay) then
            VAR_S_proc_work <= '0';
            VAR_WAIT <= 0;

            --Если в очереди есть еще
            задачи, требуем их, иначе ставим сигнал свободного процессора
            if (VAR_S_fifo_empty =
                '0') then

                S_rdreq <= '1';
            else
                VAR_S_proc_free <=
                    '1';

                end if;
            else
                VAR_WAIT <= VAR_WAIT + 1;
                S_proc_prinyal <= '0';
                task_id_proc <= VAR_id;

                end if;

            end if;

        end if;
    
```

Изм.	Лист	№ докум.	Подпись	Дата

```

--
end if;

end process;

S_proc_free <= VAR_S_proc_free;

S_proc_work <= VAR_S_proc_work;

end Behavioral;

```

					ПГУ 2.090301.15.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		72

ПРИЛОЖЕНИЕ В ПРЕЗЕНТАЦИЯ

ТЕМА РАБОТЫ: «АППАРАТНЫЙ ПЛАНИРОВЩИК ЗАДАЧ С ЛОКАЛЬНОЙ ОЧЕРЕДЬЮ»

Автор работы: студент группы 15ВВ1 Лосяков О.А.
Научный руководитель: к.т.н., профессор Бикташев Р.А.



ПРОИЗВОДИТЕЛЬНОСТЬ



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.15.001 ПЗ

Лист

73

ЦЕЛИ И ЗАДАЧИ

Цель данной ВКР – разработка модели аппаратного планировщика задач с локальной очередью. Для обеспечения равномерного распределения нагрузки новая задача отправляется в процессорный узел с наименьшей очередью.

Задачи:

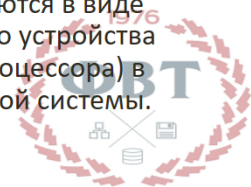
1. проанализировать предметную область;
2. разработать модель аппаратного планировщика задач с локальной очередью;
3. произвести моделирование на языке VHDL;
4. проанализировать результаты моделирования.



АКТУАЛЬНОСТЬ



В качестве альтернативы уже существующим планировщикам предлагается аппаратная реализация функций планирования/диспетчеризации, которая не только снимает проблему временных потерь, но и позволяет увеличить надежность операционной системы, что особенно важно для систем реального времени. Предлагаемый подход заключается в том, что планировщик и диспетчер задач выполняются в виде независимого аппаратного устройства (специализированного процессора) в составе многопроцессорной системы.



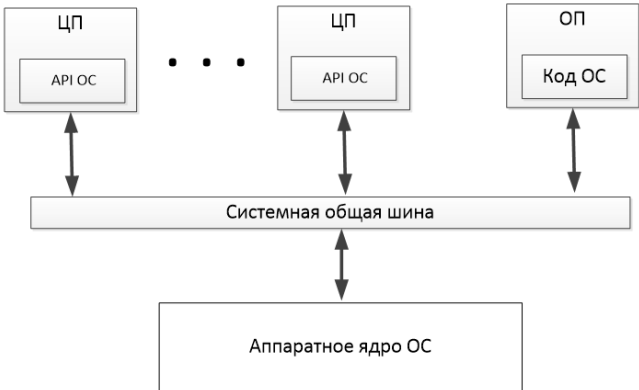
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.15.001 ПЗ

Лист

74

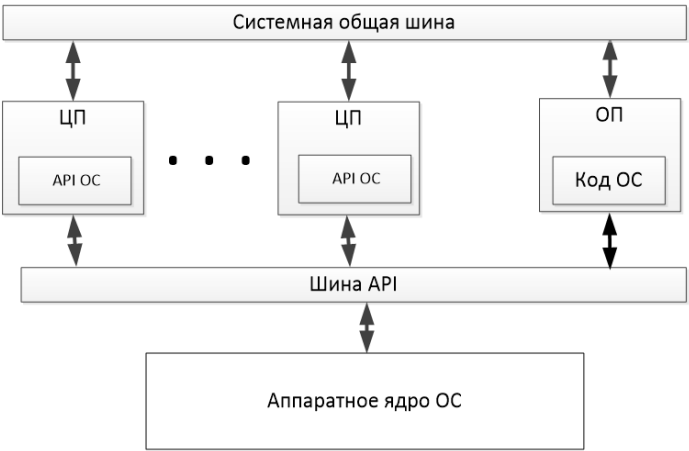
ВАРИАНТЫ ПОДКЛЮЧЕНИЯ



Включение аппаратного ядра через общую шину.



ВАРИАНТЫ ПОДКЛЮЧЕНИЯ

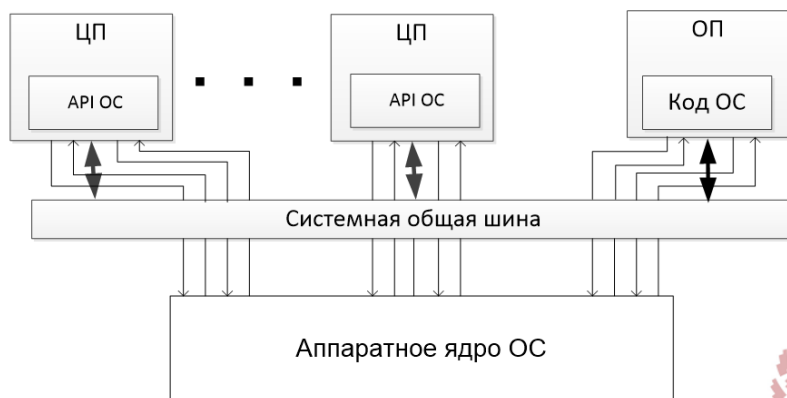


Включение аппаратного ядра через отдельную системную шину.



Изм.	Лист	№ докум.	Подпись	Дата

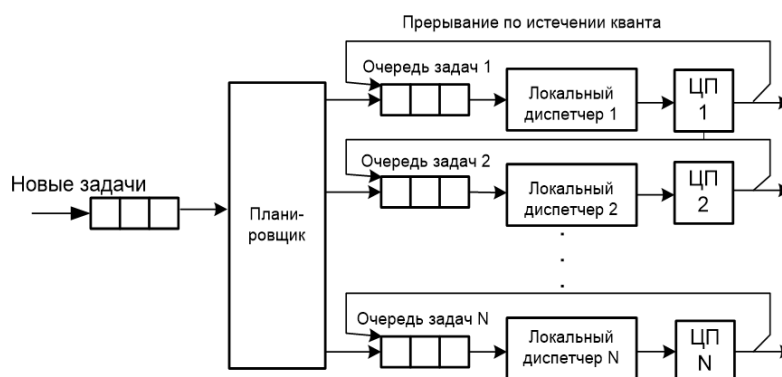
ВАРИАНТЫ ПОДКЛЮЧЕНИЯ



. Включение аппаратного ядра с непосредственными связями.



МОДЕЛЬ МНОГОПРОЦЕССОРНОЙ СИСТЕМЫ



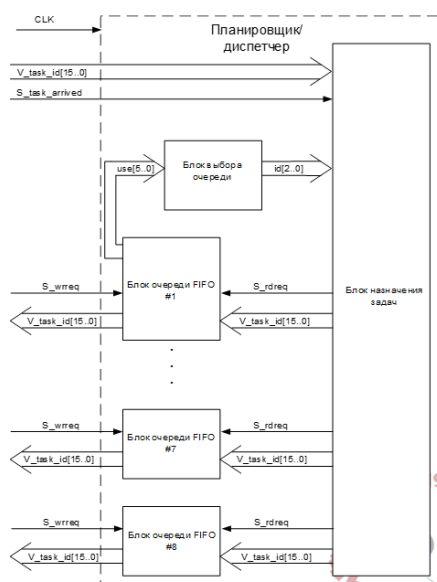
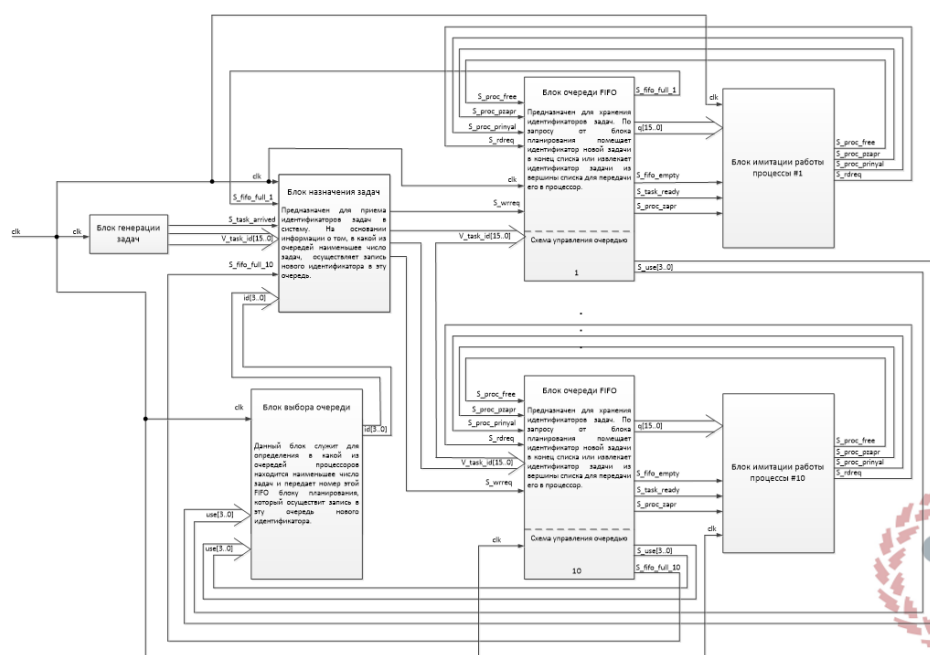
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.15.001 ПЗ

Лист

76

СТРУКТУРНАЯ СХЕМА МОДЕЛИ УСТРОЙСТВА УПРАВЛЕНИЯ ПРОЦЕССАМИ С ЛОКАЛЬНЫМИ ОЧЕРЕДЯМИ



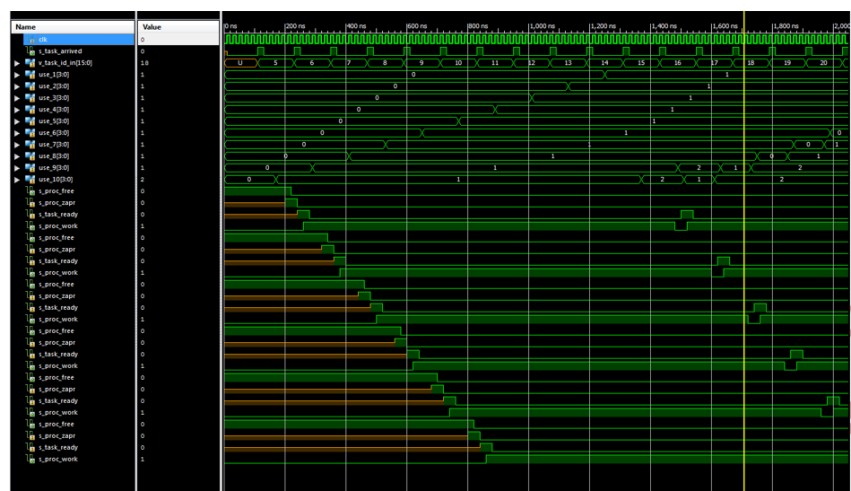
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.15.001 ПЗ

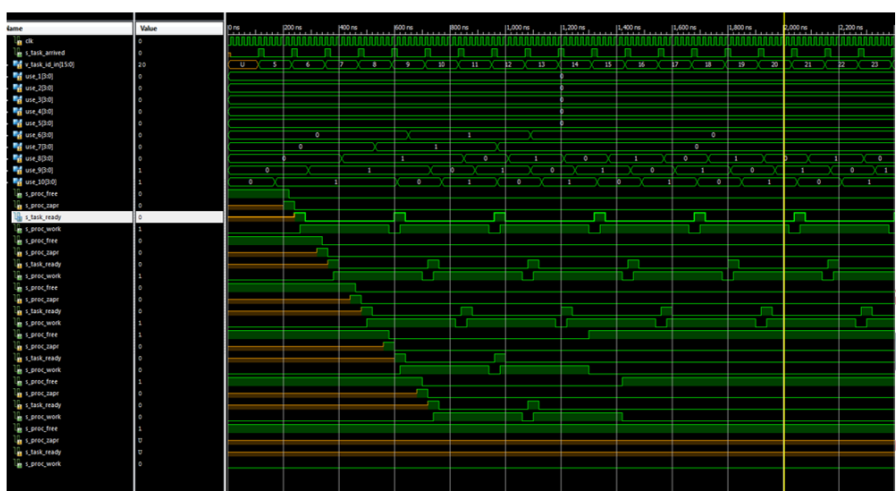
Лист

77

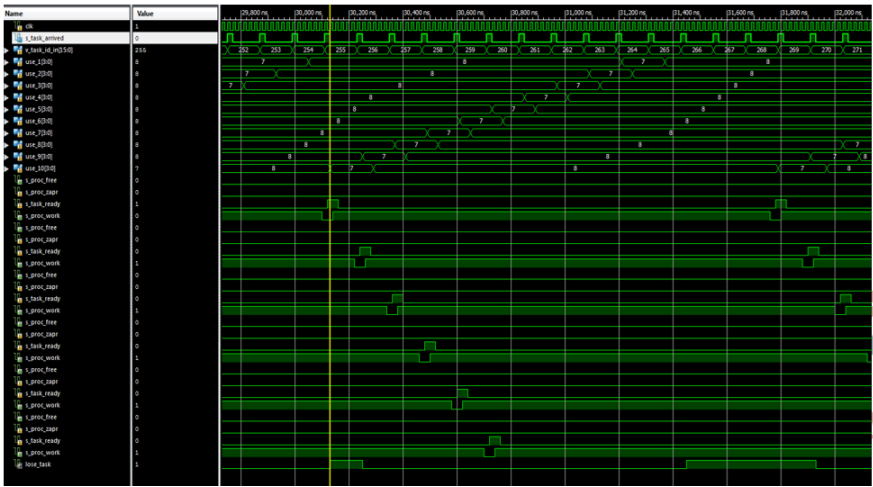
ВРЕМЕННАЯ ДИАГРАММА РАБОТЫ



ВРЕМЕННАЯ ДИАГРАММА РАБОТЫ



ВРЕМЕННАЯ ДИАГРАММА РАБОТЫ



ЗАКЛЮЧЕНИЕ

В результате выполнения представленной выпускной квалификационной работы была разработана модель планировщика задач с локальной очередью в многопроцессорной системе. Планировщик направляет поступившую задачу в очередь процессорного узла с наименьшим количеством задач, запускает задачи на обработку в процессоре и удаляет их из системы после обработки.



СПАСИБО ЗА ВНИМАНИЕ



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 2.090301.15.001 ПЗ

Лист

80