СИСТЕМЫ И СРЕДСТВА ИНФОРМАТИКИ

Выпуск 21 № 1 Год 2011

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

В соответствии с решением Бюро Редакционно-издательского совета АН СССР, принятым в 1988 г., Институтом проблем информатики в период с 1989 по 2010 гг. выпускались ежегодные труды «Системы и средства информатики». Всего за это время вышло из печати 20 ежегодных выпусков (томов) указанного издания; кроме того, издано 12 специальных тематических выпусков ежегодника.

С целью расширения возможностей для публикации сотрудниками РАН и других организаций фундаментальных и прикладных результатов в области информационных технологий, повышения уровня и значимости указанных публикаций Бюро Отделения нанотехнологий и информационных технологий РАН приняло 22.06.2011 г. решение о реорганизации ежегодника ИПИ РАН «Системы и средства информатики» в журнал «Системы и средства информатики» Российской академии наук.

Учредителями журнала «Системы и средства информатики» являются Российская академия наук, Отделение нанотехнологий и информационных технологий РАН и Учреждение Российской академии наук Институт проблем информатики РАН. Журнал «Системы и средства информатики» является правопреемником ежегодника «Системы и средства информатики» (с тем же индексом ISSN 0869-6527). Организация издания этого журнала возложена на Институт проблем информатики РАН.

Периодичность издания журнала «Системы и средства информатики» — 2 номера в год (составляющие один том), с объемом выпуска каждого номера — до 20 печатных листов (с возможностью издания при необходимости дополнительных тематических номеров).

Решением Бюро Отделения нанотехнологий и информационных технологий РАН определена следующая тематика публикаций в журнале «Системы и средства информатики»:

- информационно-телекоммуникационные системы и средства их построения;
- архитектура и программное обеспечение вычислительных машин, комплексов и сетей;
- методы и средства защиты информации.

Редакционный совет и Редакционная коллегия журнала «Системы и средства информатики» выражают надежду, что данный журнал будет представлять интерес для научных работников, преподавателей, инженеров, аспирантов и студентов, интересующихся современным состоянием исследований в области информатики, информационных технологий и вычислительной техники. Мы приглашаем исследователей, работающих по указанным выше научным направлениям, войти в число авторов журнала.

Главный редактор журнала
«Системы и средства информатики» *академик И. А. Соколов*

РАЗВИТИЕ КОМПЬЮТЕРНОЙ ПОДДЕРЖКИ СТАТИСТИЧЕСКИХ НАУЧНЫХ ИССЛЕДОВАНИЙ СИСТЕМ ВЫСОКОЙ ТОЧНОСТИ И ДОСТУПНОСТИ*

***И. Н. Синицын*[1], *Э. Р. Корепанов*[2], *В. В. Белоусов*[3], *В. С. Шоргин*[4], *И. В. Макаренкова*[5], *Т. Д. Конашенкова*[6], *Е. С. Агафонов*[7], *Н. Н. Семендяев*[8]**

Аннотация: Изложены методологические основы компьютерной поддержки статистических научных исследований (СтНИ), основанных на канонических разложениях (КР) случайных функций (СФ). Приведены примеры инструментальных программных средств для астрометрических систем высокой точности и банковских систем высокой доступности.

Ключевые слова: компьютерная поддержка статистических научных исследований; системы высокой точности; системы высокой доступности; статистическая динамика вращения Земли; банковские системы; круговые (угловые) случайные величины, функции, процессы и системы

1 Введение

Как известно [1], компьютерная поддержка научных исследований (КПНИ) как неотъемлемая часть автоматизации научных исследований (АНИ) становится все более характерным признаком современных научных исследований (НИ) и оказывает сильное влияние на их интенсивность и эффективность, превращается в важнейший фактор дальнейшего прогресса науки. Современный этап развития КПНИ характеризуется интенсивным проникновением ее в новые сферы исследований и разработок, расширением контингента пользователей, охватом всех этапов исследований от сбора и первичной обработки данных, управления экспериментами до анализа и перспективного планирования основных направлений НИ и их информационных технологий (ИТ).

*Работа выполнена при финансовой поддержке РФФИ (проект №10–07–00021) и программы ОНИТ РАН «Информационные технологии и анализ сложных систем» (проект 1.5).

[1]Институт проблем информатики Российской академии наук, sinitsin@dol.ru
[2]Институт проблем информатики Российской академии наук, ekorepanov@ipiran.ru
[3]Институт проблем информатики Российской академии наук, vbelousov@ipiran.ru
[4]Институт проблем информатики Российской академии наук, vshorgin@ipiran.ru
[5]Институт проблем информатики Российской академии наук, imakarenkova@ipiran.ru
[6]Институт проблем информатики Российской академии наук, tkonashenkova@ipiran.ru
[7]Институт проблем информатики Российской академии наук, eagafonov@ipiran.ru
[8]Институт проблем информатики Российской академии наук, nsemendyaev@ipiran.ru

Под информационной технологией обычно понимают совокупность систематических и массовых способов создания, накопления, обработки, хранения, передачи и распределения информации (данных, знаний) с помощью средств вычислительной техники и связи. На практике обычно создается ИТ, рассчитанная на выполнение с ее помощью некоторой основной функции, что связано с необходимостью решения нескольких типовых задач исследований. Перечень основных функций довольно ограничен, а с другой стороны — выполнение этих функций может потребоваться во многих применениях. Это делает целесообразным выделение функционально-ориентированных, предметно-ориентированных и проблемно-ориентированных ИТ.

Функционально-ориентированные ИТ предназначены для реализации типовых (относительно автономных) задач исследования. Такие ИТ могут обладать довольно высокой степенью универсальности и быть доступными для разработки и воспроизводства при минимальном участии будущего пользователя.

Предметно-ориентированные ИТ предназначены для решения специфической задачи в конкретной области. Они максимальным образом удовлетворяют частным требованиям данного применения и могут обладать наименьшей степенью универсальности. Как правило, их появление невозможно без будущего пользователя.

Однако часто удается обобщить требования со стороны ряда конкретных приложений и выделить некоторые общие прикладные проблемы исследований. Отсюда возникает понятие проблемно-ориентированной ИТ, которая занимает промежуточное положение между функционально-ориентированной и предметно-ориентированной ИТ. Потенциальные пользователи такой ИТ могут принять участие в ее разработке только на начальной стадии обобщения и типизации конкретных задач или конечной стадии при разработке некоторых специальных дополнений. Это позволяет основную часть ИТ создавать автономно от пользователя и применять унифицированные программные и технические решения.

Рассмотрим методологические основы компьютерной поддержки СтНИ, а также новые результаты по созданию и применению инструментальных средств СтНИ применительно к системам высокой точности и доступности.

2 Принципы и подходы к компьютерной поддержке научных исследований

В Российской академии наук в настоящее время накоплен богатый опыт разработки и использования систем КПНИ, различающихся по:

- характеру научной работы (теоретические исследования, эксперименты, наблюдения и управление научной деятельностью);
- функциональной полноте (одно- и многофункциональные);

– пространственным характеристикам (локальные и территориально распределенные);
– временны́м характеристикам (on-line и off-line);
– структуре (с постоянной и перестраиваемой структурой);
– мобильности (стационарные, передвижные, бортовые);
– типичности исследования (уникальные и рутинные, широко распространенные, массовые);
– качеству и критериям мониторинга и управления экспериментом (экстремальные, оптимальные, адаптивные, обучающиеся, самонастраивающиеся и самоорганизующиеся системы);
– типу моделей мониторинга и управнения экспериментами (аналитические и эвристичестике, детерминированные и стохастические);
– количеству уровней в структуре системы (одно- и многоуровневые);
– степени централизации процессов (централизованные и децентализованные);
– количеству и уровню взаимодействия пользователей (одно- и многопользовательские, индивидуального автономного пользования и с коллективным взаимодейтствием).

Автоматизация НИ — это разновидность инженерной, научно-технической деятельности, заключающаяся в разработке инструментальных средств и методов организации и проведения исследований с использованием компьютеров. Автоматизация НИ является одной из областей применения компьютеров, поэтому обладает, как и другие области, чертами, общими для всех этих сфер применения компьютеров и специфичными для нее. В собственном процессе АНИ условно различают десять основных этапов:

(1) разработка теории и создание новых гипотез;
(2) планирование теоретических работ или экспериментов;
(3) развитие и реализация требуемых необходимых методических инструментов и средств для эксперимента;
(4) наладка экспериментального оборудования и подготовка эксперимента;
(5) проведение теоретического анализа и измерений, сбор данных, анализ и отображение измерений;
(6) оперативная обработки измеренных данных;
(7) управление экспериментом;
(8) заключительная и систематическая обработка всей информации;

(9) архивация и извлечение новой научной информации и знаний;

(10) использование полученной информации и знаний для подготовки дальнейших фаз исследований.

В случае теоретических разработок имеют смысл этапы 1–3, 5, 8–10. При исследованиях, включающих в себя также и экспериментальные работы или основанные прежде всего на экспериментах, к упомянутым этапам следует добавить этапы 4, 6, 7.

Развивая [2, 3], сформулируем общесистемные принципы АНИ–КПНИ на двух верхних системных уровнях: прикладном и логическом.

Прикладной уровень. Прикладной уровень (уровень пользователя, проблемный) отражает специфику класса решаемых задач с учетом современнных эргономических требований. На этом уровне отражается точка зрения пользователя АНИ–КПНИ безотносительно к способу физической реализации его требований. Эти требования многообразны и противоречевы, однако их общая составляющая имеет разумное содержание и отражает реальные тенденции в развитии АНИ:

- обеспечение функциональной полноты систем КПНИ;
- простота использования;
- обеспечение достаточной надежности;
- персонализация;
- интеллектуализация;
- обеспечение возможности интеграции и дифференциации систем и их частей;
- обеспечение модифицируемости и адаптивности;
- эргономичность;
- неуклонное снижение стоимости.

Функциональное наполнение конкретной КПНИ зависит от класса решаемых задач. Достаточно полный набор функционально-ориентированных средств АНИ, из которого можно выбрать соответствующее проблемной ориентации КПНИ подмножество, должен включать следующие средства:

- управления базами данных (БД);
- цифровой обработки сигналов;
- машинной графики;
- обработки текстов;
- обработки изображений;
- распознавания образов;
- статистической обработки результатов;

– цифрового (имитационного) моделирования;

– планирования экспериментов;

– стандартных вычислительных процедур общенаучного применения;

– аналитических преобразований, логического вывода и других, включаемых в понятия «искусственный интеллект», «извлечение знаний».

Фундаментальный *принцип простоты* на прикладном уровне означает простоту использования систем АНИ.

Концепция *персонализации* получила в настоящее время чрезвычайное развитие благодаря персональным ЭВМ. Однако эта концепция имеет гораздо более глубокие корни. Для АНИ этот принцип имеет фундаментальное значение, поскольку ориентирует проект системы на главное действующее лицо АНИ — исследователя. Персонализация — частный случай более общей эволюционной тенденции антропоморфизма инструментов человеческой деятельности. Будучи инструментом важнейшей разновидности деятельности — научной, преобразующей мир, АНИ неизбежно придет к высшей степени персонализации в силу специфических возможностей адаптации автоматизированных систем. Персонализация частично интерпертируется в виде концепции «интеллектуального интерфейса» пользователя с системой:

– интерактивный режим работы пользователя с системой на профессиональном проблемно-ориентированном языке;

– развитый многоуровневый тезаурус, обладающий способностью адаптации и генерации новых понятий и соответствующих им процедур;

– максимальное использование интерактивного графического режима, манипулирования графическими образами;

– информационное обслуживание пользователя по вопросам устройства системы;

– информационно-справочная подсистема, отражающая ход исследования и выдающая различные подсказки и напоминания пользователю относительно характера и реализации плана исследования;

– средства обработки текстов и доступа в Интернет;

– адаптация системы к уровню профессиональной подготовки и психологическим особенностям пользователя;

– интерактивные средства диагностики и контроля системы, тестового обеспечения задач.

Кроме того, в концепцию персонализации включается обязательность достаточно полного проблемно-ориентированного набора средств решения специальных прикладных задач. Для реализации этой концепции по отношению

к конкретным системам АНИ развиваются унифицированные наборы языков различных уровней, из которых можно выбирать язык, наиболее подходящий пользователю по проблемной ориентации и уровню.

Интеллектуализация понимается на прикладном уровне АНИ как принцип извлечения знаний и распределения «интеллекта» по всем подсистемам и блокам систем АНИ с целью обеспечения соответственно новых, постоянно меняющихся требований к их функциональным возможностям.

Принципы интеграции и дифференциации систем и их частей предусматривают развитие аппаратно-программных средств, позволяющих легко объединять и расчленять на автономно функционирующие блоки средства и целые системы АНИ в соответствии с требуемыми конфигурациями. На логическом уровне этот принцип обеспечивается, в основном, концепциями модульности унификации интерфейсов.

Принцип модифицируемости и адаптивности предполагает, во-первых, принципиальную возможность, а во-вторых — использование аппаратно-программных средств для вставки, удаления, замены отдельных блоков системы, их функциональной переориентации, параметрической настройки и т. п. с минимальной перестройкой других блоков и с сохранением (если это необходимо) базовых характеристик системы. На логическом уровне этот принцип обеспечивается, в основном, концепциями модульности, унификации, виртуальности, эмуляции и диверсификации.

Принцип эргономичности требует от разработчиков средств и систем АНИ учета психофизиологических особенностей пользователей. Это, в первую очередь, относится к интерфейсам пользователя с системой, устройствам, обеспечивающим интерактивный режим работы с системой, в том числе при ее обслуживании (диагностике, контроле, ремонте и т. п.).

Значение *принципов обеспечения достаточной надежности и неуклонного снижения стоимости* очевидно. Это — самые комплексные принципы, поскольку на их обеспечение прямо или косвенно работают все принципы логического уровня и методы их реализации — на физическом уровне.

Логический уровень, или уровень технического проектирования, отражает современные требования унификации проектных решений, обеспечивающий максимальную независимость функциональной структуры средств и систем АНИ от их аппаратно-программной реализации при сохранении максимальной проблемной ориентации и выполнении эрготехнических требований прикладного уровня. Логический уровень служит интерфесом между прикладным и физическим. Основные принципы логического уровня: унификация, специализация, интеграция, дифференциация, модульность, виртуальность, эмуляция, децентрализация, диверсификация.

В разные периоды развития КПНИ на первый план выходили те или иные проблемы, связанные с информационными технологиями и вычислительными

системами. Подробный анализ современных проблем информатики, важных для КПНИ, дан в [4, 5]).

3 Особенности компьютерной поддержки статистических научных исследований

3.1 Задачи статистических исследований

В стохастической информатике приходится встречаться с различными описаниями входных и выходных информационных процессов. Так, например, в системах передачи и обработки сигналов входные и выходные процессы с математической точки зрения представляют собой скалярные или векторные случайные процессы, в вычислительных устройствах — логические переменные, в системах массового обслуживания — потоки событий, в системах распознавания — случайные графы, изображения, СФ векторного аргумента и другие образы, как их принято называть в теории распознавания образов. Поэтому в общем случае необходимо считать входные и выходные процессы элементами произвольных абстрактных пространств.

В зависимости от принятого статистического описания процессов и связи между ними выделяют различные типы моделей стохастических систем (СтС). Для СтС с сосредоточенными параметрами наиболее широкое распространение получили стохастические модели, описываемые стохастическими конечными, разностными, дифференциальными, интегродифференциальными и более общими функционально-разностными и функционально-дифференциальными уравнениями. Для распределенных СтС используются стохастические уравнения в соответствующих гильбертовых и банаховых пространствах.

Центральной задачей исследования стохастических процессов и систем является задача построения их адекватных моделей. Решение этой задачи требует существенного обобщения известных результатов современной прикладной теории стохастических процессов и систем, и исследования неизмеримо усложняются в силу того, что статистическому оцениванию и идентификации подлежат не только параметры в уравнениях, их порядок и структура, но и статистические характеристики действующих на систему внешних и внутренних случайных возмущений.

Для оперативного оценивания текущего состояния и параметров СтС, а также оценивания будущих состояний СтС в реальном масштабе времени применяются оптимальные, субоптимальные методы, основанные на упрощении уравнений нелинейной фильтрации (фильтр метода нормальной аппроксимации, обобщенный фильтр Калмана–Бьюси, фильтры второго порядка и др.). Стремление получить практически реализуемые фильтры в задачах большой размерности приводит к теории условно оптимальной фильтрации Пугачёва. Теория условно-оптимальной фильтрации лежит в основе ряда эффективных методов построения

математических моделей по экспериментальным и статистическим данным в реальном масштабе времени.

При оптимизации сложных СтС обычно нельзя ограничиться каким-либо одним критерием, характеризующим работу системы достаточно полно и разносторонне, а надо принимать во внимание ряд локальных критериев, учитывающих несовпадение целей различных подсистем. К задачам оптимизации относится также задача оптимального преобразования и анализа информационных процессов в различных подсистемах. В частности, к этим задачам принадлежат задачи наилучшего прогнозирования работы системы, а также вопросы сжатия, передачи и архивации информационных процессов, задачи обеспечения их сохранения в условиях помех, важные для создания баз данных и знаний.

Следуя [3], введем понятие «обработки» информационных процессов. В понятие «обработка» информационных процессов с помощью компьютеров вкладывается более широкий смысл, чем в традиционный термин «цифровая обработка сигналов». Последний обычно означает компьютерное преобразование одного сигнала в другой с использованием математических методов. Современное понятие «обработка» информационных процессов с помощью компьютеров предполагает взаимозависимое в рамках единой ИТ решение комплекса методологических, алгоритмических и технических вопросов, обеспечивающих возможность использования информационных процессов для решения разнообразных практических задач. Такое понимание «обработки» с помощью компьютеров позволяет выделить общие аспекты решения задач, связанных с обработкой информационных процессов, и более четко сформулировать понятие ИТ исследования СтС как эффективных средств решения таких задач. Кроме того, это дает возможность создать единую теоретическую и практическую базу для таких направлений обработки информационных процессов, как «обработка сигналов, изображений и полей», «распознавание и понимание сигналов, изображений и полей», которые обычно рассматривались независимо.

В современной литературе по теории и практике обработки информационных процессов существуют различные варианты классификации задач, которые преследуют различные цели. Если стремиться к соответствию между принципами классификации и возможностью эффективной реализации задач статистических исследований на основе ИТ, то различают три класса задач обработки информационных процессов: синтез, преобразование и анализ. В рамках этих трех классов целесообразно выделить типовые, часто встречающиеся задачи обработки, ориентируясь на практическую их реализацию в виде соответствующей ИТ. Такая типизация становится особенно необходимой при создании персональных ИТ, ориентированных на использование персональных компьютеров в первую очередь.

В соответствии с классификацией к функционально-ориентированным ИТ исследования информационных процессов относятся, например, ИТ вычисления:

– математических ожиданий, дисперсий, ковариаций, коэффициентов корреляции, моментов и квазимоментов различных порядков для каждого момента времени;
– ковариационных и взаимных ковариационных функций (спектральных и взаимных спектральных плотностей), моментов и квазимоментов различных порядков для различных моментов времени;
– одномерных распределений (плотностей, характеристических функций и функционалов, доверительных областей) для каждого момента времени;
– многомерных распределений (плотностей, характеристических функций и функционалов, доверительных областей) для различных моментов времени и т. д.

К проблемно-ориентированным ИТ могут быть отнесены, например, следующие:

– анализ качества (точности, чувствительности, быстродействия, устойчивости, помехозащищенности, надежности);
– фильтрация, интерполяция, экстраполяция, реставрация и редактирование информационных процессов;
– кодирование и декодирование, компрессия и декомпрессия информационных процессов;
– идентификация и построение математических моделей, распознавание и понимание информационных процессов;
– оптимизация параметров, структуры, синтез оптимальных систем управления исследованиями и др.

Факт использования персональных компьютеров для реализации ИТ исследований определяет такие новые специфические (персональные) свойства технологии исследования, как интерактивность, доступность для пользователя, не являющегося профессионалом в области статистической информатики, относительная универсальность, сбалансированные технико-экономические показатели. Кроме того, ИТ должны обладать такими свойствами, как диалоговая реактивность, тиражируемость, приспособляемость к конкретной задаче, возможность создания банков информации (данных и знаний), самотестируемость, унифицированность программных и аппаратных средств, защищенность от некорректных действий оператора и др.

3.2 Требования к информационным технологиям анализа и синтеза фильтров для обработки информации

Для того чтобы создать достаточно эффективную стохастическую ИТ, требуется провести тщательный анализ этапов решения задачи по созданию систем

обработки результатов измерений в реальном масштабе времени. Анализ практического опыта создания систем такого рода позволяет выделить характерные этапы разработки [6]. При этом предполагается, что началу следующего этапа предшествует успешное завершение предыдущего. При неудаче на определенном этапе анализируются ее причины и происходит возврат к одному из предшествующих этапов разработки и выбор других решений.

Этап 1. Постановка задачи. На данном этапе коллектив исследователей должен поставить задачу по созданию фильтра для обработки информации на основе имеющихся исходных данных для ее решения. Исходные данные должны содержать общие характеристики объекта и измерительных устройств, данные о возможностях вычислительных устройств обработки информации в терминах КР. В состав исходных данных также включаются требования на разрабатываемую систему обработки, в том числе требования по точности, скорости, сложности и надежности. Крайне желательным элементом исходных данных служит доступная экспериментальная информация, полученная в ходе функционирования объекта и измерительных устройств в терминах КР. Однако это возможно только в тех случаях, когда задача создания системы обработки решается после создания реального объекта, а не в ходе его проектирования. Дополнительными данными, существенно влияющими на последующие этапы разработки, являются организационные вопросы, такие как сроки разработки, количество и квалификация специалистов, наличие необходимых вычислительных мощностей для решения задачи и др.

Этап 2. Построение адекватной математической модели объекта и измерительных устройств. На этом этапе должна быть построена на основе КР математическая модель реального объекта, а также используемых измерительных устройств. Построенная таким образом модель может содержать большое количество параметров, связанных между собой уравнениями различного типа, в том числе алгебраическими, разностными, дифференциальными, интегральными и т. д. При этом основной целью данного этапа должно быть построение именно адекватной модели без учета жестких требований к ее структуре. К сожалению, непосредственная проверка достоверности математической модели чаще всего сделана быть не может, и поэтому требуется выполнение следующего этапа.

Этап 3. Построение компьютерной модели рассматриваемого объекта. Для проверки адекватности математической модели на основе КР, построенной на предыдущем этапе, должна быть создана соответствующая ей компьютерная модель (программа) и должны быть проведены необходимые вычислительные эксперименты. На этом этапе для сравнения привлекаются экспериментальные данные.

Этап 4. Выбор метода обработки результатов измерений (фильтрации). Учитывая полученные на этапе 1 исходные данные, сложность и структуру математической модели, полученной на втором этапе, возможности вычислительной техники, необходимо выбирать наиболее подходящий класс фильтров (например, линейные оптимальные, нелинейные субоптимальные, нелинейные условно-оптимальные и т. д.), а также методы построения фильтров выбранного класса в терминах КР.

Этап 5. «Верификация» математической модели. В большинстве случаев математическая модель объекта и измерительных устройств, разработанная на втором этапе, не «попадают» под требования классических постановок задач фильтрации и, соответственно, не позволяют использовать отработанные методы их решения. Поэтому используются специальные методы, основанные на КР, и требуется приведение математической модели к виду, пригодному для решения задачи фильтрации, но с сохранением ее адекватности. Типичным примером такой модификации может служить замена системы дифференциальных уравнений на систему разностных уравнений. Во многих случаях путем введения дополнительных переменных удается упростить (унифицировать) типы уравнений.

Этап 6. Решение задачи синтеза фильтра. На этом этапе путем проведения математических выкладок на основе КР исходная система уравнений должна быть приведена к системе уравнений для вероятностных характеристик и коэффициентов фильтра. Далее необходимо использование аналитического и статистического моделирования. При использовании аналитического моделирования вероятностные моменты высоких порядков путем применения рекуррентных соотношений в терминах КР выражаются через моменты низших порядков. Таким образом, получается система уравнений, пригодная для проведения численных расчетов. Далее происходит последовательное вычисление всех необходимых характеристик и коэффициентов фильтра на каждом шаге. При использовании статистического моделирования применяются методы численного решения стохастических уравнений методами КР.

Этап 7. Вычислительные эксперименты по проверке фильтров. После того как проведены основные расчеты, связанные с нахождением коэффициентов фильтра, целесообразно провести вычислительные эксперименты по моделированию и анализу результатов фильтрации. При этом целесообразно также проверить помехоустойчивость поведения синтезированного фильтра.

Этап 8. Принятие решения о соответствии требованиям. На заключительном этапе проводится обоснование требований к качеству фильтров в составе других систем КПНИ.

4 Задачи статистической теории синтеза оптимальных, суб- и условно-оптимальных систем

4.1 Задачи синтеза оптимальных систем

Методы КР дают возможность исследовать системы КПНИ при наличии априорной информации о структуре и действующих возмущениях. Однако для создания (синтеза) таких систем эти методы недостаточны. При проектировании всякой системы КПНИ ее характеристики бывают неизвестны и их требуется определить так, чтобы система была в некотором смысле оптимальной с точки зрения выбранного критерия качества.

Всякая система КПНИ предназначена для работы в различных условиях, т. е. различных законах изменения подлежащих измерению, хранению, обнаружению, передаче, принятию решения и воспроизведению величин в зависимости от времени. Истинный закон изменения этих величин в каждом конкретном случае остается неизвестным. Поэтому естественно рассматривать подлежащие измерению, хранению, обнаружению, передаче, принятию решения и воспроизведению сигналы как СФ, а законы их изменения в различных конкретных условиях — как различные возможные реализации этих СФ. Тогда задача определения оптимальной системы КПНИ будет состоять в том, чтобы по данным вероятностным характеристикам, подлежащих измерению, хранению, обнаружению, передаче, принятию решения, воспроизведению СФ и помех, найти оператор системы, обеспечивающий в известном смысле наилучшее качество работы системы. На такой постановке задачи, когда подлежащие измерению, хранению, обнаружению, передаче, принятию решения и воспроизведению сигналы рассматриваются как СФ, основана вся современная статистическая теория оптимальных систем.

В простейших задачах оптимальная система может не обладать наибольшим возможным качеством в каждом конкретном случае ее работы. Однако при многократном применении этой системы в самых различных условиях, для работы в которых она предназначена, ее качество в среднем будет наилучшей возможной. Таким образом, простейшие оптимальные системы обладают наилучшим качеством лишь в среднем при всех возможных условиях их работы и могут не быть оптимальными в каждом конкретном случае их применения.

Современный уровень развития систем КПНИ дает возможность поставить вопрос о создании таких систем, которые могли бы производить анализ условий своей работы в каждом конкретном случае и на основании полученной в результате этого анализа информации решать поставленные задачи оптимальным образом в каждом конкретном случае. Простейшие системы такого рода, содержащие элементы автоматической настройки некоторых своих параметров в зависимости от результатов анализа входных и выходных величин, обычно называются *самонастраивающимися* или *адаптивными*. Более сложные системы такого рода, весь

образ действия которых в каждый данный момент определяется результатами анализа внешних условий и предшествующей работы, можно назвать *самоорганизующимися* или *самообучающимися*. Ясно, что для проектирования таких систем совершенно недостаточна теория, обеспечивающая наилучшую точность лишь в среднем для всей массы условий работы. Нужна такая теория, которая позволяет решать комплексную задачу обработки вводимой в системы КПНИ и оптимального ее использования в каждом конкретном случае. Современная теория оптимальных систем дает возможность решать подобные задачи.

Статистическая теория оптимальных систем не дает возможности непосредственно находить такие системы, которые могут быть воплощены в реальные конструкции. Она позволяет определять лишь оптимальные математические операции над входным сигналами, при которых достигается теоретический предел возможности системы при данных вероятностных характеристиках условий ее работы и помех, обусловленный самой природой задачи, внутренними свойствами тех данных, которыми располагаем для ее решения. В соответствии с этим практическое значение статистической теории оптимальных систем заключается, в основном, в том, что она позволяет находить теоретический оптимум, к которому разработчик должен стремиться при проектировании реальных систем КПНИ. Зная этот предел, разработчик может оценить, насколько близка к оптимальной уже спроектированная система КПНИ, и определить, имеет ли смысл дальнейшая работа над улучшением ее качества. Таким образом, статистическая теория оптимальных систем дает разработчику методы, с помощью которых он может сознательно оценивать качество создаваемых систем и избавляться от многих бесплодных попыток улучшить качество системы путем изменения ее структуры и параметров в тех случаях, когда существенное улучшение точности принципиально невозможно.

Максимальное достижимое качество решения задач измерения, хранения, обнаружения, передачи, принятия решения и воспроизведения сигналов в присутствии помех в значительной мере зависит от того, насколько сильно сигналы отличаются по своим свойствам от помех. Так, например, если сигнал представляет собой медленно изменяющуюся функцию, а помеха изменяется с большой частотой, то можно рассчитывать на хорошее подавление помехи и выделение сигнала. Если же сигнал также является высокочастотным и при этом не содержит таких частот, которых нет в помехе, то получить хорошее подавление помехи, не подавив одновременно и сигнал, практически невозможно. Конечно, методы статистической теории оптимальных систем дают формальное решение во всех случаях. Однако качество оптимальной системы получается низким, если сигнал слабо отличается от помехи.

Общая задача теории оптимальных систем может быть сформулирована как задача нахождения такой идеальной теоретической системы, выходная переменная которой с наибольшей возможной точностью воспроизводит данную СФ

(содержащийся во входной СФ полезный сигнал или результат заданного его преобразования). С математической точки зрения эта задача представляет собой задачу приближения одной СФ путем преобразования результатов наблюдения другой СФ. Но в таком виде эта задача является самой общей задачей математической статистики, когда по результатам наблюдения одной СФ требуется построить СФ (оценку), насколько возможно более близкую к другой случайной и неслучайной функции. С этой точки зрения статистическая теория оптимальных систем по существу представляет собой прикладную теорию статистических решений.

Иногда выражается сомнение в практической полезности статистической теории оптимальных систем, основанное на том факте, что для определения оптимальных систем необходимо знать распределения сигналов и помех или, по крайней мере, некоторые их вероятностные характеристики, в то время как вероятностные характеристики сигналов обычно бывают неизвестными. Однако это сомнение не является основательным по следующим причинам:

- во всех случаях, когда рассеивание сигналов велико по сравнению с рассеиванием помех, характеристики оптимальных систем слабо зависят от вероятностных характеристик сигналов (что же касается вероятностных характеристик помех, то они обычно сравнительно легко определяются экспериментально или теоретически);
- статистическая теория оптимальных систем дает алгоритмы для оценки сигналов, которые можно использовать для построения систем с накоплением опыта. Такие системы будут автоматически находить оценки распределений сигналов по результатам всех предшествующих циклов работы и использовать эти оценки вместо неизвестных истинных распределений сигналов для последующих циклов работы. Такие «самообучающиеся» системы будут с каждым новым циклом работы становиться все более и более близкими к оптимальным системам, соответствующими неизвестным истинным законам распределения сигналов. Таким образом, статистическая теория оптимальных систем дает возможность находить алгоритмы для построения оптимальных или, по крайней мере, самооптимизирующихся в процессе работы систем даже в тех случаях, когда распределения подлежащих обнаружению или воспроизведению сигналов неизвестны.

Основная задача статистической теории оптимальных систем может быть сформулирована математически следующим образом. Случайная функция $Z(t)$ наблюдается в некоторой области T изменения аргумента t. Требуется найти такой оператор A, чтобы СФ

$$\hat{W}(s) = AZ(t)$$

была насколько возможно более близкой к СФ $W(s)$ в области S изменения аргумента s. В соответствии с принятой в математической статистике терминологией

будем называть СФ $\hat{W}$ *оценкой* СФ W. В такой постановке задача приближения СФ является весьма общей и охватывает все встречающиеся в информатике задачи измерения, хранения, обнаружения, принятия решений, передачи и воспроизведения сигналов в присутствии помех. Случайные функции Z и W и их аргументы могут быть как скалярными величинами, так и векторами любого количества измерений. Область T, в которой наблюдается СФ Z, и область S изменения аргумента s, в которой система должна воспроизводить СФ W, могут быть произвольными множествами значений соответствующих аргументов, причем область T может зависеть от значения аргумента s.

В задачах КПНИ наблюдаемая СФ Z обычно представляет собой результат наложения помех на некоторую функцию, которую система должна воспроизвести или непосредственно, или после некоторого заданного преобразования. Таким образом, СФ W в задачах КПНИ в общем случае представляет собой результат некоторого заданного преобразования сигнала, содержащегося в наблюдаемой СФ Z, поступающей на вход системы. Вследствие этого мы будем называть СФ W для краткости просто *сигналом*.

Помехи, накладывающиеся на входной сигнал системы, могут представлять собой ошибки измерительных приборов, входящих в состав системы, шумы в элементах системы и флуктуации режима их работы, т. е. случайные отклонения условий, определяющих режим работы системы, от номинальных. Помехи могут быть также внешними по отношению к системе, в частности специально организованными для того, чтобы нарушить работу данной системы или ухудшить ее точность.

Аргумент t наблюдаемой СФ Z в задачах КПНИ обычно представляет собой время, а область T — некоторый интервал времени или дискретное множество моментов времени. При этом аргумент s обычно может трактоваться соответственно как момент конца интервала наблюдения или как последний момент из дискретного множества моментов наблюдения СФ Z. Область S представляет собой совокупность моментов времени s, в которые система должна воспроизводить сигнал W. В частном случае область S может состоять из одного фиксированного момента времени s. Таким образом, в общем случае аргумент s представляет собой текущий момент в интервале времени S, в котором система должна воспроизводить сигнал W по результатам наблюдения СФ Z в интервале $s - T \le t \le s$ или в дискретном ряде моментов времени $t_1 = s - T, t_2, \ldots, t_h = s$. При этом аргумент t будет представлять собой любой момент времени в интервале наблюдения $[\, s - T, s \,]$ или любой из моментов наблюдения $t_1, \ldots, t_h$.

За переменную s совершенно необязательно принимать момент окончания наблюдения. В некоторых задачах переменную s целесообразно выбирать иначе. Так, например, в задачах интерполяции, в которых требуется найти оценку значений сигнала внутри интервала наблюдения, за переменную s целесообразно принять любой момент внутри интервала наблюдения T. Другим примером

может служить задача одновременной экстраполяции сигнала на все моменты времени в данном интервале S по результатам наблюдения СФ Z в некотором предшествующем интервале времени T. В этом случае за переменную s целесообразно принять любой момент в интервале времени S. Область наблюдения T в подобных задачах не зависит от s. Заменой переменных интервал наблюдения всегда можно сделать независимым от переменной s, например привести его к интервалу $[0, 1]$. Таким образом, с математической точки зрения задачи, в которых область наблюдения T зависит от s, не отличаются от задач с постоянной областью наблюдения. Однако в практических задачах целесообразно учитывать зависимость области наблюдения от s, если такая зависимость существует.

Интервал наблюдения T в задачах КПНИ всегда бывает конечным. Однако иногда он бывает настолько большим, что оказывается возможным идеализировать задачу, приняв допущение, что процесс наблюдения продолжается неограниченно долго, т. е. в течение всего полубесконечного интервала времени, предшествующего данному моменту s. В большинстве же задач приходится учитывать конечность интервала наблюдения. Если система должна использовать для измерения, хранения, обнаружения, принятия решения, передачи и воспроизведения сигнала W все наблюдаемые значения входной функции Z, то началом наблюдения всегда является момент начала работы системы t_0. В этом случае интервал наблюдения $T = s - t_0$ растет с течением времени s. Однако кроме естественного ограничения интервала наблюдения моментом начала работы системы часто оказывается необходимым ограничить его некоторой конечной длительностью «памяти» системы, чтобы исключить возможность использования системой устаревшей информации, непригодной для решения задачи для данного момента времени s.

4.2 Статистические критерии оптимальности

Вопрос о выборе критерия для сравнения различных систем, имеющих одинаковое назначение, как и всякий вопрос о выборе критериев, следует решать с позиций здравого смысла, исходя из анализа условий работы системы информатики и ее назначения. В зависимости от выбора конкретного критерия решение задачи определения оптимальной системы будет различным. Однако в большинстве задач можно указать много практически целесообразных критериев, обладающих тем свойством, что система, оптимальная с точки зрения одного из критериев, близка к системам, оптимальным с точки зрения других критериев. А так как практически всегда важно лишь, чтобы система была близка к оптимальной, то вопрос о выборе критерия почти всегда имеет бесчисленное множество решений.

Качество статистических оценок часто бывает целесообразно характеризовать величиной средней квадратической ошибки. Естественно распространить применение этого критерия на более общий случай нахождения статистических

оценок сигналов, представляющих собой СФ. Тогда, вводя для ошибки системы обозначение

$$\delta W(s) = \hat{W}(s) - W(s) = AZ(t) - W(s)\,,$$

получим условие оптимальности системы в виде:

$$\eta = \mathrm{M}\left[\ |\delta W(s)|^2\ \right] = \min\ . \tag{1}$$

Величина η представляет собой начальный момент второго порядка ошибки приближения. Положительный корень квадратный из величины η обычно называется *средней квадратической ошибкой* (с.к.о.) системы. Поэтому условие (1) обычно называется критерием минимума с.к.о. Естественным обобщением критерия минимума с.к.о. является критерий минимума заданной функции ошибки, в частности некоторой степени модуля ошибки.

Величина η как момент второго порядка ошибки может быть выражена через математическое ожидание и дисперсию ошибки, т. е. является функцией математического ожидания и дисперсии ошибки. Поэтому вторым естественным обобщением критерия минимума с.к.о. является сложно-статистический критерий для экстремума заданной функции математического ожидания и дисперсии ошибки:

$$f(\mathrm{M}[\delta W(s))], D[\delta W(s)]) = \mathrm{extr}\,. \tag{2}$$

Если закон распределения ошибки полностью определяется ее математически ожиданием и дисперсией, например, если ошибка распределена нормально, то частным случаем критерия (2) является критерий максимума вероятности того, что ошибка не выйдет из заданных пределов:

$$P(|\delta W(s)| < \varphi(s)) = \max\,,$$

где $\varphi(s)$ — данная функция. При том же условии частным случаем критерия (2) является также критерий минимума предела, который с заданной вероятностью не превышается модулем ошибки:

$$\varphi(s) = \min \ \text{ при } \ P(|\delta W(s)| < \varphi(s)) = p\,, \tag{3}$$

где p — данное положительное число, меньшее единицы.

Критерий минимума с.к.о. в форме (1) применим как в случае скалярного, так и в случае векторного сигнала W. В последнем случае аргумент s включает номер составляющей векторной СФ W и критерий (1) дает одновременное и независимое приближение всех составляющих вектора W. Однако для векторных сигналов W можно выбрать за критерий качества системы некоторую величину, зависящую от всех составляющих вектора ошибки, например математическое ожидание суммы

квадратов модулей всех составляющих вектора ошибки. Естественным обобщением критерия (2) при такой постановке задачи является сложно-статистический критерий экстремума заданной функции вектора математических ожиданий $m^{\delta W}$ и ковариационной матрицы $K^{\delta W} = \mathrm{M}\left[\,\delta W^0\,\overline{\delta W^0}\,\right]$:

$$f(m^{\delta W}, K^{\delta W}) = \mathrm{extr}\,. \tag{4}$$

Критерий минимума вероятности попадания вектора ошибки в заданную область является частным случаем критерия (4) при условии, если распределение вектора ошибки полностью определяется его математическим ожиданием и ковариационной матрицей. Частным случаем критерия (4) при этом условии является и критерий минимума величины, которую с заданной вероятностью не превзойдет модуль вектора ошибки.

Все перечисленные критерии основаны на количественной оценке ошибки приближения $\delta W(s)$. В задачах обнаружения сигналов, для которых имеет значение не величина ошибки, а лишь качественный факт наличия или отсутствия ошибки, приходится пользоваться другими критериями. Часто для этой цели пользуются критерием минимума вероятности ошибочного решения. Предполагая, что система решает, что сигнал есть, если его оценка $\hat{W}$ превышает некоторый уровень c, и решает, что сигнала нет, если $\hat{W} \le c$, можем записать этот критерий в виде

$$p_{\text{ош}} = P\begin{pmatrix} W \neq 0 \\ \hat{W} \le c \end{pmatrix} + P\begin{pmatrix} W = 0 \\ \hat{W} > c \end{pmatrix} = \min\,. \tag{5}$$

Вторым критерием для решения задачи обнаружения сигналов может служить критерий условного минимума вероятности ошибочного решения при заданной условной вероятности ложного обнаружения сигнала, когда он отсутствует в наблюдаемой СФ, т. е. критерий (5) при дополнительном условии

$$P(\hat{W} > c \mid W = 0) = \alpha\,. \tag{6}$$

Этот критерий называется *критерием Неймана–Пирсона*.

Вероятность любого события может быть выражена как математическое ожидание характеристической функции соответствующего этому событию множества значений случайной величины (СВ). Поэтому критерий (5) можно представить в виде:

$$\mathrm{M}\left[\,\ell(W, \hat{W})\,\right] = \min\,, \tag{7}$$

где функция ℓ определяется равенствами

$$\ell(W, \hat{W}) = \begin{cases} 1 & (W \neq 0, \hat{W} \le c \text{ при } W = 0, \hat{W} > c)\,; \\ 0 & (W \neq 0, \hat{W} > c \text{ при } W = 0, \hat{W} \le c)\,. \end{cases}$$

Для отыскания условного минимума вероятности P при дополнительном условии (6), согласно методу неопределенных множителей Лагранжа, необходимо найти минимум величины

$$p_1 = p_{\text{ош}} + \lambda P(\hat{W} > c \mid W = 0)\,, \tag{8}$$

где λ — неопределенный множитель, и после этого определить λ_1 так, чтобы было выполнено условие (6). Обозначая через p и q соответственно вероятность наличия и вероятность отсутствия сигнала в наблюдаемой СФ ($q = 1-p$), можем переписать равенство (8) в виде:

$$p_1 = pP(\hat{W} \le c \mid W \ne 0) + q\left(1 + \frac{\lambda_1}{q}\right) P(\hat{W} > c) \mid W + 0)\,.$$

Отсюда имеем

$$p_1 = P\begin{pmatrix} W \ne 0 \\ \hat{W} \le c \end{pmatrix} + \lambda P\begin{pmatrix} W = 0 \\ \hat{W} > c \end{pmatrix}\,.$$

Критерий минимума величины p_1 также можно представить в форме (7), если определить функцию ℓ равенствами:

$$\ell(W, \hat{W}) = \begin{cases} 1 & (W \ne 0, \hat{W} \le c)\,; \\ \lambda & (W = 0, \hat{W} > c)\,; \\ 0 & (W \ne 0, \hat{W} > c \text{ или } W = 0, \hat{W} \le c)\,. \end{cases}$$

Критерий минимума вероятности ошибки и критерий условного минимума вероятности ошибки при заданной условной вероятности ложного обнаружения сигнала при его отсутствии в наблюдаемой СФ являются частными случаями критерия (7), причем для первого критерия функции ℓ определена полностью, а для второго критерия она содержит неопределенный параметр λ, который должен быть определен после нахождения минимума из дополнительного условия (6).

Критерий (1) может быть представлен в виде (7), если определить функцию ℓ формулой

$$\ell(W, \hat{W}) = \left|\hat{W} - W\right|\,.$$

Наконец, часто оказывается полезным также критерий (7) при выборе функции ℓ в виде

$$l(W, W) = 1 - e^{-k^2(\hat{W}-W)^2}\,.$$

Критерий (1) может быть представлен в форме (7), если принять за функцию ℓ характеристическую функцию соответствующего множества значений ошибки:

$$\ell(W, \hat{W}) = \begin{cases} 1 & \left(\left|\hat{W} - W\right| < \varphi(s)\right)\,; \\ 0 & \left(\left|\hat{W} - W\right| \le \varphi(s)\right)\,. \end{cases}$$

Частным случаем критерия (7) является также критерий максимума правдоподобия. Задача нахождения оценки $\hat{W} = \hat{w}(X)$ векторного параметра W по результатам наблюдения конечномерного случайного вектора X является частным случаем общей задачи теории оптимальных систем, когда области T и S состоят из конечного числа значений аргументов t и s соответственно. Полагая

$$\ell(W, \hat{W}) = c - \delta(\hat{W} - W)\,,$$

получим в этом случае:

$$\mathrm{M}\left[\,\ell(W,\hat{W})\,\right] = c - \int\limits_{-\infty}^{\infty}\int\limits_{-\infty}^{\infty} \delta(\hat{w}(x) - w)h(w)f(x \mid w)\,dxdw = \\ = c - \int\limits_{-\infty}^{\infty} h(\hat{w}(x))f(x \mid \hat{w}(x))\,dx\,. \quad (9)$$

Критерий максимума правдоподобия определяет такую функцию $\hat{w}(x)$, которая обеспечивает максимум подынтегральной функции в (9) при любом значении x. Следовательно, критерий максимума правдоподобия обеспечивает минимум величины (9).

Таким образом, получаем следующий общий принцип оценки качества системы и выбора критерия оптимальности. Качество решения задачи в каждом конкретном случае оценивается некоторой функцией $\ell(W, \hat{W})$, значение которой определяется конкретными реализациями сигнала W и его оценки $\hat{W}$. Эту функцию целесообразно назвать *функцией потерь*. Качество решения задачи в среднем для данной реализации сигнала W при всех возможных реализациях оценки $\hat{W}$, соответствующих данной реализации сигнала W, оценивается условным математическим ожиданием функции потерь при данной реализации сигнала:

$$\rho(A \mid W) = \mathrm{M}\left[\,\ell(W,\hat{W}) \mid W\,\right]\,.$$

Эту величину обычно называют *условным риском*. Условный риск зависит от оператора A, определяющего оценку $\hat{W}$, и от реализации сигнала W, что и отражено в его обозначении. Среднее качество решения задачи при всех возможных реализациях сигнала W и его оценки $\hat{W}$ характеризуется математическим ожиданием условного риска, равным безусловному математическому ожиданию функции потерь:

$$R(A) = \mathrm{M}\left[\,\rho(A \mid W)\,\right] = \mathrm{M}\left[\,\ell(W,\hat{W})\,\right]\,. \quad (10)$$

Эту величину естественно назвать *средним риском*. Установив таким образом меру качества оценки сигнала W, можно принять за критерий оптимальности системы критерий минимума среднего риска, который имеет форму (7).

Если требуется обеспечить наилучшее решение задачи приближения при некоторых дополнительных условиях, то, представив эти дополнительные условия в виде

$$\sigma_k(A) = \mathrm{M}\left[\varphi_k(W, \hat{W})\right] = \alpha_k \quad (k = \overline{1, N}) \tag{11}$$

и применяя метод неопределенных множителей Лагранжа, сведем задачу к отысканию минимума величины

$$R_1(A) = R(A) + \sum_{k=1}^{N} \lambda_k \sigma_k(A) = \mathrm{M}\left[\ell(W, \hat{W}) + \sum_{k=1}^{N} \lambda_k \varphi_k(W, \hat{W})\right].$$

Эту величину можно рассматривать как средний риск соответствующей функции потерь

$$\ell_1(W, \hat{W}) = \ell(W, \hat{W}) + \sum_{k=1}^{N} \lambda_k \varphi_k(W, \hat{W}),$$

содержащей неопределенные параметры $\lambda_1, \ldots, \lambda_N$. В рассмотренном выше случае критерия Неймана–Пирсона для обнаружения сигналов дополнительное условие (6) может быть представлено в форме (11), если принять

$$\varphi_1(W, \hat{W}) = \begin{cases} 0 & (W \neq 0 \text{ или } W = 0,\ \hat{W} \leq c); \\ \dfrac{1}{q} & (W = 0,\ \hat{W} > c). \end{cases}$$

Класс всех возможных критериев вида (7) не исчерпывается теми критериями этого вида, для которых функция потерь ℓ заранее задана или задана с точностью до конечного числа неопределенных параметров. В некоторых случаях функция потерь может зависеть от того, как строится оценка $\hat{W}$, т. е. от оператора A. В таких случаях функция потерь ℓ остается неопределенной до тех пор, пока не найден оптимальный оператор A, и задача приближения СФ сводится к совместному определению оптимального оператора A, обеспечивающего минимум среднего риска, и соответствующей этому оператору функции потерь ℓ. Так, например, критерий минимума потери информации может быть представлен в форме (7), если определить функцию потерь формулой

$$\ell(w, \hat{w}) = \log \frac{f_1(w)}{f_1(w \mid \hat{w})}, \tag{12}$$

где $f_1(w)$ — плотность вероятности сигнала W, а $f_1(w \mid \hat{w})$ — условная плотность вероятности сигнала W относительно оценки $\hat{W}$.

До сих пор рассматривались только такие критерии, для которых функция потерь зависит от значений сигнала и его оценки, соответствующих одному определенному значению аргумента s. Все подобные критерии обеспечивают оптимальное приближение оценки к сигналу при каждом данном значении s. В некоторых случаях приходится довольствоваться наилучшим приближением оценки к сигналу в среднем в некоторой области S изменения аргумента s. В подобных случаях функция потерь будет зависеть от всех значений сигнала и его оценки в области S, т. е. является функционалом от сигнала и его оценки. Так, например, критерий минимума усредненной по области S с.к.о. может быть представлен в форме (7), если принять

$$\ell(W, \hat{W}) = \int\limits_S |\delta w(s)|^2 \, p(s) \, ds \,, \tag{13}$$

или

$$\ell(W, \hat{W}) = \int\limits_S \sum_{k=0}^{N} p_k(s) \, \left| \delta W^{(k)}(s) \right|^2 \, ds \,, \tag{14}$$

где $p(s)$ и $p_k(s)$ — некоторые положительные функции веса, характеризующие относительную важность различных значений s и S. Функции потерь (13) и (14) являются функционалами от сигнала и его оценки.

Иногда в качестве критерия предлагают пользоваться критерием минимума математического ожидания числа выходов ошибки из заданных пределов в течение заданного интервала времени. Например, критерий минимума математического ожидания числа выходов СФ из заданных пределов $a(s) < \delta W(s) < b(s)$ в области S имеет форму (7) при

$$\ell(W, \hat{W}) = \int\limits_S \Big\{ \Big[\dot{a}(s) - \delta \dot{W}(s) \Big] \, \mathbf{1}(\dot{a}(s) - \delta \dot{W}(s)) \delta(a(s) - \delta W(s)) + \\ + \Big[\delta \dot{W}(s) - \dot{b}(s) \Big] \, \mathbf{1}(\delta \dot{W}(s) - \dot{b}(s)) \delta(\delta W(s) - b(s)) \Big\} \, ds \,.$$

В этом случае функция потерь также является функционалом от сигнала и его оценки.

Если качество решения задачи оценивается не потерей информации о значении сигнала при каждом данном значении аргумента s, а потерей информации о совокупности значений сигнала, соответствующих нескольким значениям s в области S, или даже потерей информации о сигнале в целом в области S,

то критерий минимума потери информации также может быть представлен в форме (7), где функция потерь определяется формулой (12), в которой W и $\hat{W}$ являются векторами соответствующего числа измерений. Таким образом, и для критерия минимума потери информации функция потерь может быть функционалом от сигнала и его оценки.

Критерии минимума среднего риска, соответствующие всем возможным определенным функциям или функционалам потерь, которые могут содержать неопределенные параметры, называются *бейесовыми* критериями. Критерии минимума среднего риска, соответствующие функциям или функционалам потерь, зависящим от оператора A, не относятся к классу бейесовых критериев. Таким образом, все рассмотренные выше критерии, кроме критерия минимума потери информации, являются бейесовыми критериями.

Для сложных систем используется комплекс согласованных в определенном смысле критериев. Так возникла статистическая теория многокритериальной оптимизации по Парето.

Вопрос о выборе критерия оптимума, который, очевидно, сводится к выбору функции потерь ℓ, не всегда может быть практически решен на основе анализа назначения и условий работы проектируемой системы. Решение этого вопроса в значительной мере определяется имеющимися в нашем распоряжении данными о вероятностных характеристиках сигнала и наблюдаемой СФ.

Для определения оптимальной системы по критерию (7) при произвольной функции потерь необходимо полное знание закона распределения сигнала и наблюдаемой СФ. Только для некоторых частных видов функции потерь задача определения оптимальной системы может быть решена на основе неполного знания закона распределения СФ Z.

Как будет показано позже, для определения оптимальной линейной системы по критерию минимума с.к.о. или по более общим критериям (1) и (3) достаточно знать математические ожидания СФ Z, W, ковариационную функцию СФ Z и взаимную ковариационную функцию СФ W, Z. С другой стороны, если известны только математические ожидания и ковариационные функции СФ Z, W и нет никаких других данных об их законе распределения, то нет никакого смысла искать оптимальную систему среди нелинейных систем. Действительно, при данных математических ожиданиях и ковариационных функциях СФ Z и W могут иметь самые разнообразные законы распределения и, в частности, могут быть распределены нормально. А при нормальном распределении СФ Z и W оптимальной системой в классе всех возможных систем по отношению к любому критерию вида (7), соответствующему функции потерь ℓ, представляющей собой функцию модуля ошибки, является некоторая линейная система. А так как нормальное распределение вследствие его широкого распространения во многих случаях может считаться наиболее вероятным, то оптимальная линейная система будет наиболее вероятной оптимальной системой в классе всех возможных систем.

Если известно только условное распределение наблюдаемой СФ Z относительного сигнала W, а распределение сигнала W неизвестно, то оптимальную систему можно искать по критерию минимума максимального возможного значения условного риска $\rho(A \mid W)$:

$$\max_W \rho(A \mid W) = \min,$$

т. е. наибольшего из значений условного риска, соответствующих всем возможным реализациям сигнала W. Этот критерий, называемый обычно *минимаксным*, обеспечивает наилучшую работу системы в наихудших возможных условиях. Вследствие этого он особенно подходит для расчета систем, работающих в условиях организованного противодействия, которое осуществляется путем выбора реализации сигнала, обеспечивающего наихудшую точность решения задачи. В случае, когда распределение сигнала неизвестно или даже вообще не существует, минимаксный критерий часто оказывается целесообразным потому, что в этом случае неизвестно, насколько часто могут встречаться сигналы, при которых точность системы близка к наихудшей, а минимаксный критерий обеспечивает наименьшую возможную потерю точности именно для таких сигналов.

В случае, когда система должна работать в условиях организованного противодействия, т. е. когда имеется нарушитель, стремящийся нарушить систему, требуемый выходной сигнал зависит от некоторых параметров, которыми нарушитель может распоряжаться. Его задача состоит в таком выборе значений этих параметров, чтобы нанести по возможности больший ущерб системе и воспрепятствовать решению поставленных задач. В таких случаях задачей системы является минимизация наносимого нарушителем ущерба системе.

Задачи определения оптимальных решений, когда имеются две стороны, интересы которых противоположны и каждая из которых стремится получить максимальную выгоду для себя и нанести максимальный ущерб противоположной стороне, являются предметом статистической теории игр.

4.3 Определение оптимальных параметров системы, имеющей заданную структуру

Если структура системы задана, то для определения оптимальных значений ее параметров U необходимо найти зависимость принятого критерия качества (10) от параметров: $J = R(A(U))$. В тех случаях, когда эта зависимость имеет аналитический вид, оптимальные значения параметров системы определяются системой уравнений, определяющих необходимые (достаточные) условия экстремума. В сложных случаях, когда уравнения, определяющие условия экстремума, нельзя решить аналитически или когда критерий $J = J(U)$ не удается выразить через параметр системы U, прибегают к приближенным методам нахождения экстремума функции, выражающей критерий качества [7].

Методы КР могут быть эффективно использованы, если удается векторы $Z = Z(t)$ и $U = U(t)$ представить каким-либо совместным КР.

4.4 Приближенные методы теории оптимальных систем

Эффективное точное решение задача синтеза оптимальных фильтров возможно для линейных или линейных относительно вектора состояния и аддитивных шумов как в задачах off-line, так и on-line обработки информации.

Необходимость обрабатывать результаты наблюдений в on-line режиме непосредственно в процессе эксперимента привела к появлению ряда приближенных методов оптимального синтеза, называемых обычно *субоптимальными* методами.

К первой группе методов относятся методы, основанные на приближенном решении фильтрационных уравнений, а ко второй группе — методы, основанные на превращении выражений для оптимальных оценок в приближенные уравнения для оценок путем разложения нелинейных функций в степенные ряды и отбрасывания остаточных членов. Ко второй группе методов относятся методы, основанные на линеаризации исходных уравнений в окрестности оценки и использовании методов оптимального линейного синтеза. Желание повысить точность оценивания ведет к различным попыткам получения уравнений субоптимального оценивания второго и высших порядков. Эффективное практическое использование методов субоптимального оценивания в on-line задачах зависит от возможности априорной оценки точности синтеза фильтров без использования результатов наблюдений.

Для решения задач on-line обработки информации особую ценность вследствие их простоты имеют методы условно-оптимальной фильтрации Пугачёва, основанные на критерии Парето [8].

5 Методическое и инструментальное программное обеспечение статистических научных исследований на основе канонических представлений случайных функций

В период 2007–2010 гг. в ИПИ РАН разработано методическое обеспечение анализа и синтеза систем для обработки информации в непрерывных и дискретных СтС на основе канонических представлений СФ КР и интегральных канонических представлений). Оно подробно описано в монографии [1]. Его развитие рассмотрено в [6, 9–11]. Разработано инструментальное программное обеспечение в среде MATLAB [12–14].

Начиная с 2010 г. в ИПИ РАН ведутся работы по созданию методического обеспечения анализа и синтеза круговых, сферических и кватернионных СтС для обработки информации [10]. Разработаны точные и приближенные методы анализа и синтеза, в том числе и на основе КР с независимыми компонентами.

6 Применения в системах высокой точности

Высокоточные системы обработки наблюдений о флуктуациях полюса Земли. В цикле работ [12–20] разработаны новые методы, алгоритмы, программное обеспечение и ИТ для off-line и on-line обработки информации в среде MATLAB. Созданы информационные ресурсы по проблеме «Статистическая динамика вращения Земли» (раздел «Полюс Земли»).

Высокоточные системы обработки наблюдений о флуктуациях неравномерности вращения Земли. В [21–27] рассмотрены новые методы off-line и on-line обработки информации о флуктуациях неравномерности вращения Земли. Начаты работы по созданию информационных ресурсов по проблеме «Статистическая динамика вращения Земли» (раздел «Неравномерность вращения Земли»).

7 Применения в банковских системах высокой доступности

Системы управления информационными активами

Как известно [28, 29], контроль информационных активов (ИА) является одной из важнейших задач для организаций, которые постоянно повышают эффективность вкладываемых ими средств, стараются рационализировать способы их расходования. Управление активами (Asset Management) — передовая развивающаяся технология, преследующая две основные цели: определение структуры расходов на ИТ и контроль отдачи от вложенных средств. Они должны обеспечивать тщательный контроль, анализ материальных и нематериальных информационных ресурсов в распределенной среде организации на протяжении всего их жизненного цикла и распространяться на многие подразделения и направления деятельности организации, включая информационные отделы, службы технической поддержки, руководство предприятия. Системы управления ИА (СУИА) относятся к пассивным системам, т. е. осуществляют только сбор и обработку информации, но не оказывают обратного воздействия на имеющиеся ИА.

В сфере технической эксплуатации под учетом (инвентаризацией) понимается автоматический сбор информации о наличии и состоянии оборудования, общесистемных и объектовых ресурсов, о путях приобретения оборудования, способах использования, порядке и сроках проведения технического обслуживания и ремонта, состоянии учитываемых объектов, а также информации об организации службы эксплуатации.

Специализированные программные средства для СУИА можно разделить, по «мощности», на две категории:

(1) программные средства ведения автономной БД с информацией об управлении ИА, включая стоимостные показатели;

(2) программные средства ведения БД информации управления ИА, с возможностью автоматизированного сбора данных и оперативного взаимодействия с ERP (Enterprise Resource Planning — управление ресурсами предприятия) системами компании, осуществляющими планирование ресурсов (финансовых, человеческих, материальных) для производства товаров (услуг), оперативное управление выполнением планов (включая снабжение, сбыт, ведение договоров), все виды финансового учета, анализ результатов хозяйственной деятельности.

В ИПИ РАН накоплен большой опыт создания и внедрения СУИА для Банка России (БР). Банк России представляет собой обширную сеть организаций и подразделений, распределенных по всей территории РФ. При этом территориальные подразделения БР не являются равнозначными по техническому оснащению и связаны друг с другом разными каналами связи: начиная от производительных оптоволоконных соединений и заканчивая каналами с низкой пропускной способностью, без возможности организации постоянного “on-line” соединения. Подобная организация приводит к необходимости создания информационной системы с распределенной структурой и независимыми БД. Рассмотрим основные успешные решения данной системы.

Региональный уровень СУИА предназначен для автоматизации функций ввода, хранения, обработки, выдачи по запросам и передачи в центральную подсистему учетной информации о наличии, движении и качественном состоянии материальных и нематериальных активов, находящихся в зоне ответственности того подразделения, для которого развернута своя региональная подсистема. Если региональное подразделение имеет сложную распределенную структуру, то в нем может быть развернуто несколько отдельных предприятий и структура СУИА приобретает вид дерева с многими уровнями. В зависимости от состояния конкретного подразделения БР используется несколько вариантов хранения и ведения его информации:

– единая централизованная БД регионального уровня для подразделений с качественными каналами связи высокой пропускной способности;

– отдельная БД регионального уровня для подразделения, которое не может быть подключено к централизованной региональной базе;

– в особых случаях допускается ведение отдельных файлов электронных таблиц Excel и их передача в БД вышестоящего подразделения.

Назначение центральной подсистемы СУИА состоит в обеспечении автоматизации функций сбора, хранения, сопровождения и предоставления полной информации о материальных и нематериальных активах организации, поступающей из регионального уровня. Центральный уровень содержит набор БД с полным перечнем учитываемых активов, результатов расчета обобщенной статистической информации по различным критериям. Центральная подсистема

обеспечивает доступ руководящего состава к сформированным данным. В ней также происходит централизованное изменение классификаторов СУИА, используемых для обеспечения терминологического единства.

Под коллективным (или совместным) ведением БД СУИА подразумевается участие двух и более пользователей в заполнении сведений об одном и том же информационном активе, а также связанные с этим процессы и технологии, обеспечивающие целостность и актуальность содержимого БД. Можно выделить два уровня коллективного ведения БД:

(1) локальный (в редактировании сведений участвуют сотрудники одной подсистемы СУИА);

(2) межсистемный (в редактировании сведений участвуют сотрудники разных подсистем СУИА).

Коллективное ведение является важным моментом при качественном формировании БД, без которого часто бывает очень сложно обойтись. Пользователи имеют возможность создавать и посылать друг другу специальные сообщения (уведомления) о необходимости выполнения каких-либо действий с объектами учета. Далее система сама отслеживает выполнение уведомлений.

Анализ качества ведения информации в СУИА необходим для повышения эффективности работы системы в целом, начиная с ввода данных и заканчивая расчетом актуальной и выверенной сводной информации на центральном уровне. Для решения поставленной задачи используются различные механизмы контроля ввода данных, а также анализа уже существующей информации с последующим формированием списка замечаний и рекомендаций по их устранению. При этом широко используются методическое и программное обеспечение, “off-line” и “on-line” обработка статистической информации, основанное на КР случайных функций (см. разд. 5).

При разработке программного обеспечения для БР остро встает вопрос организации связи между территориально разрозненными подразделениями. Данный вопрос затрагивает как технические, так и программные средства обеспечения функционирования территориально разрозненной организации как единого целого. При проектировании специального программного обеспечения (СПО) необходимо учитывать возникающие трудности в распределенной системе, такие как разнообразие программного обеспечения в отдельно взятом подразделении и возможные отказы связи по техническим каналам взаимодействия. Выходом из сложившейся ситуации может стать способ разработки СПО в рамках клиент-серверной архитектуры как на техническом, так и на программном уровне.

Примером данного подхода является разработанное в ИПИ РАН оригинальное СПО в рамках СУИА единой телекоммуникационной банковской сети БР, представленное на web-сервере корпоративного портала интранет (КПИ). Данное СПО представляет собой клиент-серверное web-приложение, работающее с СУБД Oracle и предоставляет пользователям КПИ различную статистическую

информацию по протоколу HTTP. Клиентская часть представляет собой web-приложение, написанное в рамках J2EE технологии, так называемый портлет, который размещается на web-сервере КПИ. Тонкими клиентами являются браузеры Internet Explorer или другие средства просмотра интернета (Netscape и др.). Специальное программное обеспечение является небольшой локальной частью глобальной автоматизированной системы, но при этом требует развитой инфраструктуры программного обеспечения на стороне БР. Под инфраструктурой понимается наличие мощных серверов, на которые можно разместить портальные сервера, такие как IBM WebSphere Portal, Apache и др. Но при наличии развитой инфраструктуры появляется возможность разработки глобальных распределенных автоматизированных систем в рамках технологии J2EE. На данный момент рассматривается возможность управления, заполнения и анализа БД через серверы КПИ при разработке узконаправленных web-компонент. В основу методов анализа качества СПО также положены материалы разд. 5.

8 Заключение

Разработаны методические основы компьютерной поддержки современных статистических научных исследований. Особое внимание уделено методам, основанным на канонических разложениях случайных функций. Рассмотрены примеры программной реализации для систем высокой точности и банковских системы высокой доступности.

Литература

1. *Синицын И. Н.* Канонические представления случайных функций и их применения в задачах компьютерной поддержки научных исследований. — М.: ТОРУС ПРЕСС, 2009.
2. *Велихов Е. П., Выставкин А. Н.* Проблемы развития работ по автоматизации научных исследований // Мат-лы I Международной школы по автоматизации научных исследований (г. Пущино, 1982). — Изд-во Научный центр биологических исследований АН СССР в Пущино, 1985. С. 5–40.
3. *Дымков В. И., Синицын И. Н.* Элементы концепции персональных видеосистем // Системы и средства информатики: Ежегодник ИПИ РАН, 1989. С. 100–108.
4. *Соколов И. А.* Информатика: состояние, проблемы, перспективы. Доклад на Сессии ОНИТ РАН 15.12.2008.
5. *Соколов И. А.* (ред.). Информатика: состояние, проблемы, перспективы // Босов А. В., Будзко В. И., Захаров В. Н., Козмидиади В. А., Корепанов Э. Р., Синицын И. Н., Шоргин С. Я., Ушмаев О. С. — М.: ИПИ РАН, 2009.
6. *Корепанов Э. Р.* Стохастические информационные технологии на основе фильтров Пугачёва // Информатика и её применения, 2011. Т. 5. Вып. 2. С. 36–57.
7. Справочник по теории автоматического управления / Под ред. Красовского А. А. — М.: Наука, 1987.

8. *Рыков А. С.* Модели и методы системного анализа: принятие решений и оптимизация: Уч. пособие для вузов. — М.: МИСиС; Изд. дом «Руда и металлы», 2005.
9. *Синицын И. Н., Марков Ю. Г., Синицын В. И., Корепанов Э. Р., Семендяев Н. Н.* Опыт построения моделей флуктуаций полюса Земли на основе астрономических измерений // Автоматика и телемеханика, 2010. № 3. С. 87–97.
10. *Синицын И. Н.* Канонические разложения случайных функций и их применение в стохастических информационных технологиях научных исследований: Курс лекций // Мат-лы 10-й междунар. конф. «Распознавание образов и анализ изображений: новые информационные технологии — РОАИ-10-2010». — СПб., 2010.
11. *Синицын И. Н., Синицын В. И., Корепанов Э. Р.* Владимир Семенович Пугачёв в Институте проблем информатики // Академик Пугачёв Владимир Семенович: к столетию со дня рождения / Предисл. академика С. В. Емельянова; под ред. И. Н. Синицына. — М.: ТОРУС ПРЕСС, 2011. С. 255–266.
12. *Марков Ю. Г., Синицын И. Н.* Корреляционная модель колебаний полюса Земли с параметрическими возмущениями // Астрономический журнал, 2008. Т. 85. № 4. С. 1–11.
13. *Марков Ю. Г., Киселев М. Л., Синицын И. Н.* Влияние параметрических и нелинейных возмущений на статистическую динамику полюса Земли // Космические исследования, 2008. С. 201–212.
14. *Синицын И. Н., Корепанов Э. Р., Семендяев Н. Н.* Методическое и программное обеспечение корреляционного анализа флуктуаций полюса Земли // Ежегодник ИПИ РАН «Системы и средства информатики». Спец. вып. «Геоинформационные технологии». — М.: ИПИ РАН, 2008. С. 379–396.
15. *Синицын И. Н.* Методы построения эредитарных информационных моделей флуктуаций в статистической динамике Земли // Информатика и её применения, 2009. Т. 3. Вып. 1. С. 2–7.
16. *Марков Ю. Г., Синицын И. Н., Синицын В. И., Корепанов Э. Р., Семендяев Н. Н.* Опыт построения моделей флуктуаций полюса Земли на основе астрометрических измерений // Автоматика и телемеханика, 2009. № 3. С. 44–53.
17. *Синицын И. Н., Синицын В. И., Корепанов Э. Р., Белоусов В. В., Семендяев Н. Н.* Оперативное построение информационных моделей движения полюса Земли методами линейных и линеаризованных фильтров // Информатика и её применения, 2010. Т. 4. Вып. 1. С. 2–11.
18. *Марков Ю. Г., Рыхлова Л. В., Синицын И. Н.* Развитие методов построения моделей движения полюса Земли // Астрономический журнал, 2010. № 9. С. 111–120.
19. *Марков Ю. Г., Синицын И. Н., Перепелкин В. В., Семендяев Н. Н.* Вращательно-колебательное движение Земли и глобальная составляющая сейсмического процесса // Докл. РАН, 2010. Т. 435. № 3. С. 1–5.
20. *Синицын И. Н., Синицын В. И., Корепанов Э. Р., Белоусов В. В.* Развитие алгоритмического обеспечения анализа стохастических систем, основанного на канонических разложениях случайных функций // Автоматика и телемеханика, 2011. № 2. С. 195–206.

21. *Марков Ю. Г., Синицын И. Н.* Корреляционная модель приливной неравномерности вращения Земли // Докл. РАН, 2008. Т. 419. № 3. С. 338–341.
22. *Синицын И. Н.* Квазилинейные методы построения информационных моделей флуктуаций неравномерности вращения Земли // Информатика и её применения, 2008. Т. 2. Вып. 1. С. 24–32.
23. *Марков Ю. Г., Синицын И. Н.* Влияние «окрашенных» флуктуаций на неравномерности вращения Земли // Докл. РАН, 2009. Т. 427. № 3. С. 323–329.
24. *Марков Ю. Г., Синицын И. Н.* Одно- и многомерные распределения флуктуаций неравномерностей вращения Земли // Докл. РАН, 2009. Т. 428. № 5. С. 616–619.
25. *Синицын И. Н.* Вероятностные методы построения информационных моделей неравномерности вращения Земли // Информатика и её применения, 2009. Т. 3. Вып. 4. С. 2–11.
26. *Марков Ю. Г., Синицын И. Н.* Вероятностные модели флуктуаций неравномерности вращения Земли при нестационарных возмущениях // Докл. РАН, 2010. Т. 432. № 3. С. 1–5.
27. *Марков Ю. Г., Перепелкин В. В., Синицын И. Н., Семендяев Н. Н.* Информационные модели неравномерности вращения Земли // Информатика и её применения, 2011. Т. 5. Вып. 2. С. 17–35.
28. *Синицын И. Н., Быстров И. И., Корепанов Э. Р., Белоусов В. В., Шоргин В. С., Агафонов Е. С.* Развитие систем управления информационными активами крупных предприятий // Сб. докладов X Междунар. науч.-техн. конф. «Кибернетика и высокие технологии XXI века» (C& T-2009). — Воронеж: НПФ «Саквоее», 2009. Т. 1. С. 148–159.
29. *Белоусов В. В.* Состояние исследований в области разработки методического программного обеспечения для систем управления и обработки информации // Системы высокой доступности, 2010. Т. 6. Вып. 4. С. 48–62.

ЭВОЛЮЦИЯ АРХИТЕКТУР СОВРЕМЕННЫХ МИКРОПРОЦЕССОРОВ

С. В. Замковец[1], *В. Н. Захаров*[2], *В. Е. Красовский*[3]

Аннотация: Рассмотрены причины разработки первых микропроцессоров с архитектурой CISC. Основные недостатки этой архитектуры связаны с ее избыточностью. В качестве альтернативы этой архитектуры были разработаны микропроцессоры с архитектурой RISC. В обеих архитектурах используется динамический способ управления выполнением команд. Статический способ управления используется в архитектурах VLIW, в которых основная часть по управлению выполнением возлагается на компиляторы.

Ключевые слова: микропроцессор; архитектура; CISC; RISC; VLIW; конвейер; внеочередное выполнение команд

1 CISC микропроцессоры

В 1971 г. фирма Intel выпустила микросхему, которая получила название микропроцессор. Традиционно микропроцессоры этой фирмы относятся к классу CISC (Complex Instruction Set Computer — компьютеры со сложным набором команд) [1]. Образно выражаясь, можно сказать, что CISC — это философия разработчиков процессоров, которая, с одной стороны, нацелена на обеспечение легкости программирования, а с другой стороны — делает эффективной использование памяти, поскольку память в то время, когда создавались первые микропроцессоры фирмы Intel, была небольшой, достаточно дорогой и медленной и ее при программировании необходимо было экономить. Каждая команда в CISC микропроцессорах может выполнять целую последовательность операций внутри процессора, что уменьшает количество команд, необходимых для реализации данной программы. Основоположником CISC-архитектуры можно считать компанию IBM с ее базовой архитектурой /360, ядро которой используется с 1964 г. и дошло до наших дней, например в таких современных мейнфреймах, как IBM ES/9000.

В соответствии с представленной философией многие компьютерные архитекторы пытались устранить семантический разрыв, который заключался в том, что объекты манипулирования и соответствующие им операции, реализуемые

[1]Институт проблем информатики Российской академии наук, SZamkovetc@ipiran.ru

[2]Институт проблем информатики Российской академии наук, VZakharov@ipiran.ru

[3]Институт электронных управляющих машин им. И. С. Брука (ИНЭУМ им. И. С. Брука), Krasovsky_v@ineum.ru

архитектурой вычислительной системы, редко имеют близкое родство с объектами и операциями, описываемыми в языках программирования [2]. Задача разработчиков заключалась в том, чтобы построить такой набор команд, который непосредственно поддерживал бы конструкции языков высокого уровня, например вызовы процедур, контроль циклов, сложные режимы адресации, обеспечивая доступ к структурам данных и массивам при помощи единичных инструкций. Для дальнейшего увеличения плотности кодов сами команды также подвергались специальному кодированию. Например, все режимы адресации кодировались при помощи одного байта. Компактная природа таких команд приводила к небольшому размеру программ и небольшому количеству обращений к основной медленной памяти. Это приводило к хорошей программной производительности как на ассемблере, так и на языках высокого уровня, таких как Фортран или Алгол. Это можно считать первой причиной, почему микропроцессоры пошли по такому пути развития.

Вторая причина заключается в том, что первоначально в ранних проектах процессоров разработчики использовали специальную аппаратную логику для декодирования и выполнения каждой команды. Для простых проектов с небольшим количеством команд такой подход оправдывал себя, но более сложную архитектуру реализовать было тяжелее из-за сложности построения логики управления. Тогда проектировщики изменили подход — они использовали простую логику для управления путями передач между различными элементами процессора и использовали упрощенный микрокод для управления этой логикой. Этот тип реализации известен как микропрограммная реализация.

Микропрограммный способ выполнения команд был предложен Уилксом в 1951 г., и при таком способе управление выполнением команд базируется на постоянной памяти, организованной в виде прямоугольной матрицы, где каждая строка соответствует одному машинному такту, а элементы в столбце управляют вентилями.

Таким образом, при использовании микропрограммной системы процессор имеет встроенную постоянную память, которая содержит группы микрокодов команд, соответствующие каждой команде машинного языка. Когда такая команда поступает в процессор, он выполняет соответствующую последовательность микрокодов.

Одним из последствий использования микропрограммного управления является то, что разработчики могли вносить больше функциональности в каждую команду. Это в значительной степени сокращало общее количество команд, требующихся для реализации программы, и поэтому делало более эффективным использование медленной основной памяти. Кроме того, реализация сложных команд процессоров осуществлялась достаточно просто.

Рассмотрим на примере микропроцессоров фирмы Intel основные архитектурные особенности процессоров класса CISC [1]. При этом под архитектурой

будем понимать набор атрибутов, которыми может пользоваться программист, разрабатывающий программу на машинном языке, т. е. на ассемблере.

В первую очередь, для CISC микропроцессоров характерно небольшое количество регистров общего назначения. В Pentium процессорах имеется 8 восьмиразрядных регистров, 8 шестнадцатиразрядных и 8 тридцатидвухразрядных. При выполнении команд все регистры общего назначения являются равноправными, однако некоторые из них имеют специальное назначение, так как используются в различных командах по умолчанию.

Второй отличительной чертой CISC микропроцессоров является очень большое количество различных инструкций. Для кодирования кодов операций в формате команд выделяется два байта. Некоторые команды имеют один и тот же код операций, но операции выполняются над разными размерами операндов — это осуществляется за счет использования специальных байтов-префиксов, которые указывают, какие размеры операнда надо использовать.

Очевидно, что столь большое количество команд приводит к некоторой их избыточности. Примером избыточности может служить команда BSWAP (byte swap), которая используется для изменения формы адресации. В качестве операнда в этой команде может быть использован только 32-разрядный регистр. При выполнении данной команды четвертый байт становится первым, третий — вторым, второй — третьим, а первый байт — четвертым. В микропроцессорах фирмы Intel информация располагается в памяти в соответствии с принципом «младший байт по младшему адресу» (little endian). В некоторых микропроцессорах других фирм (например, фирмы Motorola) информация располагается в соответствии с принципом «младший байт по старшему адресу» (big endian). Для преобразования информации при передаче ее между компьютерами с различной организацией байт может быть использована данная команда. Однако в современных системах преобразование информации осуществляется контроллерами, в связи с чем данную команду можно считать избыточной.

Следующей отличительной чертой микропроцессоров класса CISC является наличие большого количества режимов адресации. Это можно объяснить тем, что эти процессоры являются двухоперандными и один из операндов может находиться в памяти. Для обеспечения указания месторасположения этого операнда и необходимо использование большого количества различных режимов адресации. Для указания того, какой режим адресации используется, служит специальный байт, который называется байтом MOD R/M и который в формате команды располагается после кода операции. В зависимости от значений отдельных полей этого байта и будет использован тот или иной режим адресации. Если же должен быть использован режим адресации, при котором содержимое индексного регистра умножается на какой-либо коэффициент, то в этом случае после байта MOD R/M в команде микропроцессора следует байт, который называется SIB (scaled index byte) байт.

Наконец, последней отличительной чертой CISC микропроцессоров является переменная длина команд. Например, в микропроцессорах фирмы Intel x86 длина команд изменяется от 1 байта до максимального размера в 15 байт. В состав команды могут входить до 4 префиксных байт, которые влияют на выполнение данной команды, до 2 байт кода операции, 2 байта режимов адресации, до 4 байт непосредственного операнда и до 4 байт указания смещения. Очевидно, что такой сложный состав инструкций определяет и сложность функции дешифрации, которая реализуется в микропроцессорах для определения выполняемой команды. Кроме того, при использовании команд переменной длины невозможно осуществить декодирование следующей команды до того, как закончится декодирование предыдущей.

Из представленных особенностей CISC-микропроцессоров можно сделать вывод, что эти устройства являются достаточно сложными как в проектировании, так и в изготовлении, так как для реализации они требуют большого количества транзисторов. В качестве примера трудностей реализации описанных особенностей микропроцессоров CISC-формата можно привести следующий факт: дополнительный рост трудоемкости разработки микропроцессора Pentium в сравнении с микропроцессорами i8086/i486 привел не только к увеличению реального срока его проекта на 27% в сравнении с ожидаемым, но и к проявлению ошибок в первых моделях данного процессора.

Вторым фактором, определяющим сложность CISC-микропроцессоров, является необходимость для микропроцессоров следующих поколений обеспечивать аппаратную совместимость с микропроцессорами предыдущих поколений. Это определяется тем громадным объемом программных средств, которые были разработаны для этих микропроцессоров. Эта же причина, т. е. невозможность использовать наработанные программные ресурсы, определяла многие трагические судьбы разработанных в свое время микропроцессоров, которые обладали прекрасными характеристиками, работали на очень высоких скоростях, но не могли выполнять имеющиеся программы, поскольку работали с системой команд, отличной от микропроцессоров фирмы Intel. В качестве примера можно привести микропроцессор Alpha [3], разработанный фирмой DEC, который назывался микропроцессором XXI в. Этот микропроцессор показывал наивысшую производительность и работал на наибольшей частоте по сравнению с другими микропроцессорами, но в итоге не выдержал сложившейся конкуренции.

Кроме фактора сложности CISC-микропроцессоров необходимо также отметить и значительную избыточность этих устройств. Исследования по борьбе с избыточностью и по построению оптимальной системы команд проводились многими разработчиками ЭВМ. Для рационального выбора набора команд процессора в конце 1970-х гг. программистами США и Англии были проведены многочисленные исследования, в которых определялась частота использования отдельных инструкций. При этом эти исследования проводились над смесью

прикладных программ — это совокупность программ обработки текстов, управления базами данных, компиляции-трансляции, автоматизации проектирования, управления и обработки числовых данных [4].

Непосредственным итогом проведенных исследований является известное правило 80/20: в 80% кода типичной прикладной программы используется лишь 20% простейших машинных команд из всего доступного набора [5].

2 RISC микропроцессоры

Результатом проведенных исследований стало формирование нового класса микропроцессоров, получившего название RISC (Reduced Instruction Set Computers — компьютеры с сокращенным набором команд) [6]. Одной из основных целей этого класса микропроцессоров был выбор архитектур и технологий реализации для обеспечения эффективной работы с такими языками высокого уровня, как Си и Паскаль. На начальных этапах этой разработки было установлено, что рациональное ограничение числа команд небольшим набором наиболее употребительных операций в сочетании с архитектурой, ориентированной на быстрое выполнение всех команд этого набора, может обеспечить построение вычислительных машин с высокой производительностью. В это же время были сформулированы основные принципы построения RISC микропроцессоров [7], представленные ниже.

1. Система команд микропроцессора должна содержать минимальное количество наиболее часто используемых простейших инструкций. Все команды должны иметь фиксированную длину и простой формат. Например, в микропроцессорах Alpha все команды имеют размер 4 байта. Одинаковая длина команд позволяет реализовать декодирование команд параллельно (поскольку известна их длина), в то время как в CISC микропроцессорах, как уже отмечалось, декодирование следующей команды может быть начато только после окончания декодирования предыдущей, так как длина этих команд различна.
2. Для операций с памятью должны использоваться только специализированные команды чтения или записи. Операции вида «прочитать–изменить–записать» в системе команд отсутствуют. Любые операции «изменить» выполняются только над содержимым регистров (так называемая load-and-store архитектура).

   Операции обработки данных реализуются только в формате «регистр–регистр», при этом операнды выбираются из регистров микропроцессора и результат операции записывается также в регистр; обмен между регистрами и памятью выполняется только с помощью команд загрузки\записи.
3. Состав системы команд должен быть «удобен» для компиляции операторов языков высокого уровня. Практика написания компиляторов и достижения в

теории трансляции показали, что для эффективной компиляции лучше иметь небольшой набор простых регистровых команд (и, стало быть, быстрых), чем набор сложных команд, затрудняющих применение методов оптимизации кода транслируемой программы.

4. Все команды должны быть реализованы непосредственно аппаратным способом, т. е. обычные команды не интерпретируются микрокомандами. Устранение уровня интерпретации повышает скорость выполнения большинства команд. В компьютерах типа CISC более сложные команды могут разбиваться на несколько шагов, которые затем выполняются как последовательность микрокоманд. Эта дополнительная операция снижает быстродействие машины, но может использоваться для редко применяемых команд.

В настоящее время самыми крупными разработчиками RISC-процессоров считаются Sun Microsystems (архитектура SPARC – Ultra SPARC), IBM (многокристальные процессоры Power, однокристальные PowerPC), Mips Technologies (семейство Rxx00), а также Hewlett-Packard (архитектура PA-RISC — PA-8000). Ранее на этом рынке работала фирма Digital Equipment Corporation, которая выпускала микропроцессоры Alpha (Alpha 21164, 21264, 21364), — к сожалению, микропроцессоры Alpha в настоящее время не выпускаются.

Простота реализации инструкций RISC-микропроцессоров позволила разработчикам сосредоточить свои усилия на вопросах реализации архитектур, т.е. вплотную заняться вопросами микроархитектур. Самый большой вклад в развитие микроархитектур RISC-процессоры внесли внедрением конвейерных методов обработки команд.

Идею конвейера и его реализацию можно рассмотреть на следующем примере.

Анализ команд микропроцессора показал, что выполнение команд можно разделить на некоторое количество этапов (стадий):

- PF — предварительная выборка — на этом этапе команды выбираются из памяти или кэш и устанавливаются в очередь;
- D1 — декодирование команд — на этом этапе команды декодируются и разделяются на отдельные компоненты — код операции и операнды;
- D2 — генерация адреса — на этом этапе рассчитывается адрес операнда, если этот операнд находится в памяти;
- EX — процессор выполняет действия, требуемые для выполнения команды;
- WB — завершение выполнения команды, а данные, которые должны быть записаны в память, посылаются в буфер записи.

Если микропроцессор устроен таким образом, что работа каждого этапа выполнения команды производится в определенном месте своими аппаратными ресурсами, а после выполнения этой работы команда передается на следующий этап (а значит, на следующее определенное место), то получается, что каждая

Таблица 1 Нормальная работа конвейера

Такты	Этапы					Результат
	PF	D1	D2	EX	WB	
1	I_1					Старт команды 1
2	I_2	I_1				Старт команды 2
3	I_3	I_2	I_1			Старт команды 3
4	I_4	I_3	I_2	I_1		
5	I_5	I_4	I_3	I_2	I_1	Финиш команды 1
6	I_6	I_5	I_4	I_3	I_2	Финиш команды 2
7	I_7	I_6	I_5	I_4	I_3	Финиш команды 3

команда обрабатывается на конвейере. В этом случае время выполнения каждой команды равно времени нахождения этой команды на конвейере, а если считать, что время нахождения каждой команды на ступени равно, например, одному такту, то результат выполнения каждой команды появляется на выходе с конвейера каждый такт. Нормальная работа конвейера показана в табл. 1.

При использовании конвейерной обработки команд в показанном примере общее время выполнения каждой команды будет составлять 5 тактов, а в каждом такте аппаратура будет выполнять пять различных команд. В представленном примере конвейер имеет всего пять ступеней. Если же конвейер состоит из гораздо большего количества ступеней, то это означает, что на каждой ступени аппаратные средства выполняют меньше работы и эту работу они могут выполнять за меньшее время. Таким образом, за счет увеличения количества ступеней в конвейере можно увеличить частоту работы микропроцессора. Так, если в микропроцессоре Pentium длина конвейера составляла 5 ступеней (при максимальной тактовой частоте 200 МГц), то в Pentium-4 — уже 20 ступеней (при максимальной тактовой частоте 3,4 ГГц).

Большое преимущество микроархитектур с конвейерной обработкой перед последовательной наблюдается в идеальном конвейере, в котором отсутствуют конфликты. Конфликты — это такие ситуации при работе конвейера, которые препятствуют выполнению очередной команды в предназначенном для нее такте. Конфликты, возникающие в конвейере, можно разделить на три группы:

(1) структурные конфликты;

(2) конфликты по управлению;

(3) конфликты по данным.

Структурные конфликты возникают в тех случаях, когда аппаратные средства конвейера микропроцессора не могут поддерживать для всех команд одинаковое время нахождения на какой-либо ступени конвейера. Чаще всего такая ситуация возникает на ступени выполнения инструкции. В этом случае работа конвейера

приостанавливается до тех пор, пока команда не будет обработана на этой ступени.

Структурные конфликты можно исправить двумя способами. Во-первых, можно увеличить время такта до такого значения, при котором все этапы всех инструкций выполняются за один такт. Конечно, при этом существенно снижается эффект конвейерной обработки, поскольку при этом уменьшается тактовая частота работы микропроцессора. Обычно такое увеличение времени такта требуется лишь для небольшого количества команд.

Второй способ преодоления структурного конфликта предполагает применение таких аппаратных решений, которые позволяют снизить затраты времени на выполнение данного этапа (например, использование специальных схем умножения). Однако такой способ может привести к значительному усложнению схемы микропроцессора. Поскольку такие конфликты в работе конвейера возникают при работе команд, относительно редко встречающихся в программах, то обычно при разработке ищут компромисс между увеличением длительности такта и усложнением схемы микропроцессора.

Конфликты по управлению возникают в тех случаях, когда в конвейере находятся команды переходов или другие команды, изменяющие значение счетчика команд. Возникновение конфликтов по управлению можно показать на примере команд условного перехода. Пусть в программе команда $i+1$ является командой условного перехода, формирующая адрес следующей команды в зависимости от результата выполнения команды i. Если конвейер является пятиступенчатым, то команда i завершит свое выполнение в такте 5. В то же время команда $i+1$ условного перехода должна прочитать необходимые ей признаки в такте 3, чтобы правильно сформировать адрес следующей команды. Таким образом, конвейер должен остановиться, чтобы команда $i+1$ смогла сформировать требуемый адрес. Если же конвейер имеет большее количество ступеней, то промежуток времени между формированием признака результата и тактом, где он анализируется, может быть еще бóльшим.

Другим примером конфликта по управлению может служить команда безусловного перехода с косвенным адресом, при встрече с которой возникает необходимость полностью сбрасывать конвейер, так как в счетчике команд появляется абсолютно непредсказуемый адрес следующей команды.

В случае конфликта по управлению, возникающего с командами условного перехода, для снижения потерь работы конвейера в процессорах используется метод предсказания переходов. Основная идея этого метода заключается в том, что в процессоре используется специальный блок, который определяет наиболее вероятное направление перехода, не дожидаясь формирования признаков, на основании анализа которых этот переход реализуется. После этого процессор выполняет команды по предсказанной ветви программы (так называемое спекулятивное выполнение). Чтобы обеспечить корректное выполнение программы,

процессор не записывает получаемые результаты в память или регистры, а записывает их в специальный буфер. Если направление перехода предсказано правильно, то процессор продолжает выполнять программу дальше, а полученные результаты переписываются из буфера по местам назначения. В случае неправильного предсказания результаты уничтожаются.

Обычно для прогнозирования переходов учитывают направление переходов, реализовавшиеся этой командой ранее при выполнении программы. Например, подсчитывается количество переходов, выполненных ранее по различным направлениям, и на основании этого предсказывается направление перехода при следующем выполнении данной команды. В современных микропроцессорах вероятность правильного предсказания направления переходов достигает 90%–95%.

Конфликты по данным возникают в случаях, когда выполнение одной команды зависит от результата выполнения предыдущей команды. Рассмотрим следующую ситуацию: пусть команда i предшествует команде j и команда j должна прочитать операнд, значение которого формируется при помощи команды i. Если эти команды находятся достаточно близко в конвейере, то может возникнуть ситуация, при которой предшествующая команда не успевает записать значение этого операнда. Для борьбы с конфликтами по данным в микропроцессорах используется метод внеочередного исполнения команд (out-of-order), который впервые был реализован в 1990 г. в микропроцессоре POWER1. Логическая сложность реализации этого метода явилась основной причиной того, что он слабо использовался в вычислительных машинах. Другая причина заключалась в том, что для использования этой парадигмы требовалась большая площадь на кристалле. В настоящее время такие технологические ограничения, как правило, отсутствуют, и данный метод широко используется для избежания конфликтов.

Основная идея метода внеочередного исполнения команд заключается в том, что после декодирования команды поступают не на следующую ступень конвейера, а в специальный буфер, который носит название «буфер переупорядочения» (ReOrder Buffer, ROB).

В каждом такте в этом буфере осуществляется поиск операций, готовых к исполнению (т. е. операнды которых уже вычислены либо вычисляются и будут готовы к моменту попадания операции в функциональное устройство) вне зависимости от порядка, в котором они записывались в буфер. При нахождении команд, готовых к исполнению, они отправляются в конвейер, а после завершения операции осуществляется пометка в соответствующем элементе ROB. Это делается для того, чтобы сохранить результат выполнения программы в том виде, в котором он должен быть представлен в случае выполнения команд «по порядку».

Другой способ борьбы с конфликтами в конвейерах осуществляется путем увеличения количества однотипных функциональных устройств, которые могут

Таблица 2 Последовательность выполнения команд в микропроцессоре с двумя конвейерами

Этап	Такты													
	1		2		3		4		5		6		7	
IF	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_7	I_9	I_7	I_{10}	I_{11}	I_{12}
ID			I_1	I_2	I_3	I_4	I_5	I_6	I_5	I_8	I_5	I_9	I_7	I_{10}
OR					I_1	I_2	I_3	I_4	I_3	I_6	I_3	I_8	I_5	I_9
EX							I_1	I_2	I_1	I_4	I_1	I_6	I_3	I_8
WB										I_2		I_4	I_1	I_6

одновременно выполнять одни и те же или схожие функции. Например, в современных микропроцессорах часто используется Гарвардская архитектура, в которой разделяют кэш-память для хранения команд и кэш-память данных. Во многих микропроцессорах используют многопортовую схему доступа к регистровой памяти, при которой к регистрам можно одновременно обращаться по одному каналу для записи, а по другому — для считывания информации. Конфликты из-за исполнительных устройств обычно сглаживаются введением в состав микропроцессора дополнительных блоков.

Таким образом, возникла идея применить в одном микропроцессоре несколько конвейеров, а процессоры, имеющие в своем составе более одного конвейера, называются суперскалярными. Впервые несколько конвейеров были реализованы в RISC-микропроцессорах, а первым суперскалярным CISC-микропроцессором стал Pentium.

Недостатком суперскалярных микропроцессоров является необходимость синхронного продвижения команд в каждом из конвейеров. В табл. 2 представлена последовательность выполнения команд в микропроцессоре, имеющем два конвейера, при условии, что команде I_1 требуется 3 такта на этапе EX. При этом команды будут завершаться в последовательности: I_2–I_4–I_1–I_6–..., т. е. последовательность выполнения команд будет нарушена.

Следовательно, для обеспечения правильной работы суперскалярного микропроцессора при возникновении конфликта, требующего остановки в одном из конвейеров, должны приостанавливать свою работу и другие. В противном случае может нарушиться исходный порядок завершения команд программы. Но такие приостановки существенно снижают быстродействие процессора. Разрешение этой ситуации состоит в том, чтобы дать возможность выполняться командам в одном конвейере вне зависимости от ситуации в других конвейерах. Это приводит к неупорядоченному выполнению команд. При этом команды, стоящие в программе позже, могут завершиться ранее команд, стоящих впереди. Аппаратные средства микропроцессора должны гарантировать, что результаты

выполненных команд будут записаны в приемник в том порядке, в котором команды записаны в программе. Для этого в микропроцессоре результаты этапа выполнения команды обычно сохраняются в специальном буфере восстановления последовательности команд. Запись результата очередной команды из этого буфера в приемник результата проводится лишь после того, как выполнены все предшествующие команды и записаны их результаты.

Таким образом, можно сделать вывод, что современный процессор состоит из различных блоков или подсистем, работающих параллельно и независимо. С целью повышения производительности необходимо, чтобы микропроцессор как можно больше обеспечивал полную загрузку блоков. В описанных микропроцессорах такая загрузка обеспечивается при помощи аппаратных средств, работу которых можно охарактеризовать алгоритмом динамического выполнения команд.

Динамический подход позволяет повышать параллельность за счет усложнения логики проверок в аппаратуре при планировании команд. К недостатку динамического подхода можно отнести необходимость ограничивать сложность алгоритмов планирования, так как повышение сложности ведет к увеличению времени, затрачиваемого на динамическое распараллеливание.

2.1 VLIW микропроцессоры

Другим развитием микропроцессоров можно считать направление VLIW (Very Long Instruction Word — очень длинное командное слово, иногда в русской литературе используется термин архитектура широкого командного слова).

Идея VLIW базируется на том, что задача эффективного планирования параллельного выполнения нескольких команд осуществляется не аппаратными средствами, а возлагается на «разумный» компилятор. Такой компилятор вначале исследует исходную программу с целью обнаружить все команды, которые могут быть выполнены одновременно, причем так, чтобы это не приводило к возникновению конфликтов. В процессе анализа компилятор может даже частично имитировать выполнение рассматриваемой программы. На следующем этапе компилятор пытается объединить такие команды в пакеты, каждый из которых рассматривается как одна сверхдлинная команда. Объединение нескольких простых команд в одну сверхдлинную производится по следующим правилам:

– количество простых команд, объединяемых в одну команду сверхбольшой длины, равно числу имеющихся в процессоре функциональных (исполнительных) блоков (ФБ);

– в сверхдлинную команду входят только такие простые команды, которые исполняются разными ФБ, т. е. обеспечивается одновременное исполнение всех составляющих сверхдлинной команды.

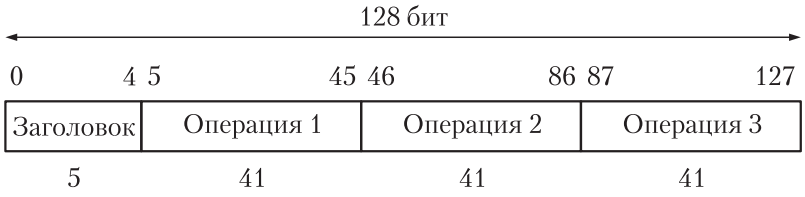


Рис. 1 Пример длинной команды

Длина сверхдлинной команды обычно составляет от 256 до 1024 бит. Такая метакоманда содержит несколько полей (по числу образующих ее простых команд), каждое из которых описывает операцию для конкретного функционального блока. Пример такой команды показан на рис. 1.

VLIW-архитектуру можно рассматривать как статическую суперскалярную архитектуру. Имеется в виду, что распараллеливание кода производится на этапе компиляции, а не динамически во время исполнения. То, что в выполняемой сверхдлинной команде исключена возможность конфликтов, позволяет предельно упростить аппаратуру VLIW-процессора и, как следствие, добиться более высокого быстродействия.

Впервые архитектура VLIW-компьютеров была предложена в 1979 г. сотрудником Йельского университета Дж. Фишером, наиболее полно эта архитектура была реализована в микропроцессоре Itanium, разработанном совместно фирмами Intel и HP. Для того чтобы подчеркнуть параллельную обработку инструкций в этих микропроцессорах, используется термин ILP (Instruction Level Parallelism — параллелизм на уровне команд, т. е. способность процессора исполнять несколько независимых машинных команд одновременно в рамках одного программного потока). VLIW-архитектуру имеет отечественный микропроцессор «Эльбрус» [8].

Преимуществом VLIW-архитектуры в сравнении с многоконвейерными суперскалярными микропроцессорами является упрощение аппаратуры за счет программного распределения операций между конвейерами. Другое преимущество заключается в том, что компилятор может быть более развитым, нежели устройства управления процессора, и он способен хранить больше контекстной информации для принятия более верных решений по оптимизации.

Представленный анализ различных архитектур микропроцессоров позволяет сделать предположение, что дальнейшее развитие микропроцессоров будет связано с объединением методов статического и динамического управления с целью обеспечения наибольших возможностей параллельной работы всех функциональных устройств микропроцессора, а следовательно, и повышения производительности его работы.

Литература

1. *Замковец С. В., Левин Н. А., Попкова Е. Я.* Архитектура микропроцессоров. — М.: МИРЭА, 2008.
2. *Майерс Г.* Архитектура современных ЭВМ. — Пер. с англ. — М.: Мир, 1985.
3. *Sites R. L., Witek R.* Alpha architecture reference manual. — Digital Press, Burlington, Mass., 1995.
4. Электроника СБИС. Проектирование микроструктур. — Пер. с англ. — М.: Мир, 1989.
5. *Johnson M.* Superscalar microprocessor design. — Prentice Hall, Englewood Cliffs N.G., 1990.
6. *Soni G. S.* Instruction issue logic for high-performance, interruptible, multiplie functions unit, pipelined computers // IEEE Nhfns. on Computers, March 1990.
7. *Kongetira P., Aingaran K., Olukotun K.* Niagara: A32-way multithreaded Sparc processor // IEEE Micro, 2005. Vol. 25. No. 2.
8. *Фельдман В. М.* Система на кристалле МЦСТ-R500S // Вопросы радиоэлектроники. Сер. ЭВТ, 2008. Вып. 2.

МОДЕЛЬ ПАРАЛЛЕЛЬНОГО ОБХОДА ДЕРЕВЬЕВ РАБОТ

В. А. Козмидиади[1]

Аннотация: Рассмотрен механизм, обобщающий MapReduce, который рассчитан на массово-параллельную обработку. Рассмотрение опирается на то, что общее задание образует дерево работ, сложные работы делятся на части вплоть по получения простых работ, которые могут выполняться параллельно. Предложена математическая модель выполнения дерева работ. Рассмотрены способы обхода таких деревьев.

Ключевые слова: MapReduce; массово-параллельная обработка; обходы деревьев; дерево работ

1 Введение

Сейчас большой популярностью пользуется конструкция (framework) MapReduce [1], которая представляет собой программную модель для обработки и создания больших наборов данных. Конструкция MapReduce предназначена для обработки на кластерах, т. е. системах, которые допускают массово-параллельную обработку (Massive parallel processing). Поскольку механизм, предложенный в статье, является расширением MapReduce, необходимо его описать. Кузнецов в [2] наряду со многим другим сделал прекрасное описание принципов работы MapReduce. Лучшего мне не сделать. Поэтому от 1.1 до 1.4 включительно — просто цитата из упомянутой работы.

1.1 MapReduce: модель и реализации

Программная модель MapReduce была придумана несколько лет тому назад в компании Google [1], и там же была выполнена первая реализация этой модели на основе распределенной файловой системы той же компании GFS (Google File System) [3]. Эта реализация активно используется в программных продуктах самой Google, но является сугубо проприетарной и недоступна для использования вне Google.

Альтернативная, свободно доступная реализация Hadoop MapReduce [4] (с открытыми исходными текстами) была выполнена в проекте Hadoop [5] сообщества Apache [6]. Она основана на использовании распределенной файловой

[1]Институт проблем информатики Российской академии наук, v.kozmidiady@gmail.com

системы HDFS (Hadoop Distributed File System) [7], также разработанной в проекте Hadoop. Реальную популярность MapReduce принесла именно реализация Hadoop в силу своей доступности и открытости, а широкое использование Hadoop MapReduce в различных исследовательских проектах приносит несомненную пользу этой системе, стимулируя разработчиков к ее постоянному совершенствованию.

Однако реализация Hadoop MapReduce полностью основана на спецификациях Google, и поэтому каноническим описанием технологии была и остается статья [1]. Заметим, что при этом в документации Hadoop MapReduce [8] используется терминология, несколько отличная от [1].

1.2 Общая модель программирования MapReduce

В этой модели вычисления производятся над множествами входных пар «ключ–значение», и в результате каждого вычисления также производится некоторое множество результирующих пар «ключ–значение». Для представления вычислений в среде MapReduce используются две основные функции: Map и Reduce. Обе функции явно кодируются разработчиками приложений в среде MapReduce.

Функция Map в цикле обрабатывает каждую пару из множества входных пар и производит множество промежуточных пар «ключ–значение». Среда MapReduce группирует все промежуточные значения с одним и тем же ключом I и передает их функции Reduce.

Функция Reduce получает значение ключа I и множество значений, связанных с этим ключом. В типичных ситуациях каждая группа обрабатывается (в цикле) таким образом, что в результате одного вызова функции образуется не более одного результирующего значения.

1.3 Реализация в распределенной среде

Реализации MapReduce от Google и Hadoop ориентированы на использование в кластерной распределенной среде со следующими основными характеристиками:

- узлы среды выполнения MR-приложений обычно представляют собой компьютеры общего назначения с операционной системой Linux;
- используется стандартное сетевое оборудование с адаптерами, рассчитанными на скорости передачи в 100 Мбит/с или 1 Гбит/с, но средняя пропускная способность существенно ниже;
- кластер состоит из сотен или тысяч машин, так что вполне вероятны отказы отдельных узлов;

– для хранения данных используются недорогие дисковые устройства, подключенные напрямую к отдельным машинам;

– для управления данными, хранящимися на этих дисках, используется распределенная файловая система;

– пользователи представляют свои задания в систему планирования; каждое задание состоит из некоторого набора задач, которые отображаются планировщиком на некоторый набор узлов кластера.

1.4 Выполнение MR-приложения

Вызовы Map распределяются по нескольким узлам кластера путем разделения входных данных на M непересекающихся групп (split). Входные группы могут параллельно обрабатываться на разных машинах. Вызовы Reduce распределяются путем разделения пространства ключей промежуточных ключей на R частей с использованием некоторой функции разделения (например, функции хеширования). Число разделов R и функция разделения задаются пользователем.

Выполнение MR-программы происходит следующим образом. Сначала среда MapReduce расщепляет входной файл на M частей, размер которых может задаваться пользователем. Затем сразу в нескольких узлах кластера запускается основная программа MapReduce. Один из экземпляров этой программы играет специальную роль и называется *распорядителем* (master). Остальные экземпляры являются *исполнителями* (worker), которым распорядитель назначает работу. Распорядитель должен назначить исполнителям для выполнения M задач Map и R задач Reduce.

Исполнитель, которому назначена задача Map, читает содержимое соответствующей группы, разбирает пары «ключ–значение» входных данных и передает каждую пару в определенную пользователем функцию Map. Промежуточные пары «ключ–значение», производимые функцией Map, буферизуются в основной памяти. Периодически буферизованные пары, разделяемые на R областей на основе функции разделения, записываются в локальную дисковую память исполнителя. Координаты этих сохраненных на диске буферизованных пар отсылаются распорядителю, который, в свою очередь, передает эти координаты исполнителям задачи Reduce. I-й Reduce-исполнитель снабжается координатами всех i-х областей буферизованных пар, произведенных всеми M Map-исполнителями.

После получения этих координат от распорядителя исполнитель задачи Reduce с использованием механизма удаленных вызовов процедур переписывает данные с локальных дисков исполнителей задачи Map в свою память или на локальный диск (в зависимости от объема данных). После переписи всех промежуточных данных выполняется их сортировка по значениям промежуточного ключа для образования групп с одинаковым значением ключа. Если объем

промежуточных данных слишком велик для выполнения сортировки в основной памяти, используется внешняя сортировка.

Далее Reduce-исполнитель организует цикл по отсортированным промежуточным данным и для каждого уникального значения ключа вызывает пользовательскую функцию Reduce с передачей ей в качестве аргумента значения ключа и соответствующее множество значений. Результирующие пары функции Reduce добавляются в окончательный результирующий файл данного Reduce-исполнителя. После завершения всех задач Map и Reduce распорядитель активизирует программу пользователя, вызывавшую MapReduce.

После успешного завершения выполнения задания MapReduce результаты размещаются в R файлах распределенной файловой системы (имена этих результирующих файлов задаются пользователем). Обычно не требуется объединять их в один файл, потому что часто полученные файлы используются в качестве входных для запуска следующего MR-задания или в каком-либо другом распределенном приложении, которое может получать входные данные из нескольких файлов.

2 Основные определения

Настоящая статья является продолжением [9]. Цель этой статьи — дать математическое описание выполнения задания, которое обладает следующими свойствами:

- задание допускает деление на работы, которые могут быть выполнены параллельно разными процессорами, образующими кластер;
- работы, на которые делится задание, образуют дерево;
- листовые работы этого дерева не требуют взаимодействия друг с другом;
- работы не меняются в процессе выполнения.

2.1 Дерево работ

Для параллельного выполнения некоторого задания это задание следует разделить на части, которые могут быть выполнены параллельно разными процессами, возможно, находящимися на разных процессорах. Минимальная неделимая часть задания, т. е. часть, которая не допускает деления на части, выполняемые параллельно, называется *простой работой*. Естественно, что набор простых работ зависит от задания. Для некоторых простых работ важен порядок их обработки по отношению к остальным простым работам. Минимальный набор простых работ $P = \{p_1, \ldots, p_n\}$, которые должны быть выполнены перед тем, как может начать выполняться некоторая другая простая работа p, называется *полной базой* простой работы p и обозначается как $B_F(p)$. Если $B_F(p) = \varnothing$, то p

называется *листовой работой*, а множество их обозначается L. Если $B_F(p) \neq \varnothing$, то p называется *управляющей работой*. Конечно, управляющие работы зависят от задания. Обозначим множество управляющих работ через M. Выбросим из $B_F(p)$ все такие листовые работы $B_F(p_i)$, что $p_{\in}B_F(p)$ и $B_F(p_i) \subseteq B_F(p)$; тогда образуется *база* p, которая обозначается через $B(p)$. Если $p_i \in B(p)$, то p_i *подчинена* p. Множеству листовых и управляющих работ одного задания можно сопоставить направленный *граф работ* (ортограф работ), вершинами которого являются листовые и управляющие работы, а дугами $p \to p_i$, $p_i \in B(p)$. Дальше рассматриваются только такие задания, для которых граф работ:

– не меняется в процессе выполнения;

– представляет собой направленное дерево. Корнем этого дерева в невырожденном случае (т. е. когда задание допускает деление на простые работы) является некоторая управляющая работа, которая соответствует всему заданию в целом.

Все листовые работы должны быть *идемпотентными*. Это означает, что работа, будучи прерванной в любом месте и повторена сначала, даст тот же результат, как если бы она не была прервана, а дошла до конца.

Обозначим направленное дерево работ через T и введем следующие обозначения:

– множество вершин $v(T)$;

– корень дерева $r \in v(T)$;

– $\Rightarrow$ — это транзитивное замыкание $\to$.

В статье [10] даются примерно те же определения, однако они уточнены.

2.2 Основная часть и отчеты

Листовая работа делится на две части: *основную часть* листовой работы и подготовку *промежуточного отчета* о листовой работе. Этот отчет обозначается как $R(l)$, $l \in L$. Каждая из этих частей может быть пустой. Части листовой работы выполняются как единое целое, и в этом смысле не отличаются друг от друга.

Выполнение управляющей работы p также состоит из двух частей: *основной части* управляющей работы и подготовки *промежуточного отчета* об управляющей работе. Основная часть не может быть пустой, по крайней мере, следует сформировать список листовых работ, которые можно выполнять. Однако подготовка промежуточного отчета может быть пустой. Части управляющей работы делаются в разные моменты: основная часть выполняется перед всеми работами из $B(p)$, а подготовка промежуточного отчета делается после всех работ из $B(p)$.

Обозначим через $R(p)$ промежуточный отчет для листовой работы p. $R(p)$ является функцией от $R(p_i)$, $p_i \in B(p)$, и только от них. Таким образом, промежуточный отчет используется при обработке управляющей работы, а основная часть — нет.

Промежуточный отчет для корня дерева работ r, т. е. $R(r)$, называется *итоговым отчетом*. Итоговый отчет иногда может служить результатом выполнения всего задания.

Далее рассматриваются отчеты, которые обладают определенными свойствами. Пусть $A = R(L) \cup R(M)$, где $R(L)$ — множество промежуточных отчетов для листовых работ, а $R(M)$ — множество промежуточных отчетов для управляющих работ, которое включает итоговый отчет $R(r)$. На A определена двуместная операция $+$, зависящая от задания, $A + A \to A$, которая ассоциативна и коммутативна. A содержит единственный нейтральный элемент 0 (пустой отчет), для которого при любом a, $a \in A$, $a + 0 = 0 + a = a$. Определим индуктивно работу уровня n. Листовая работа называется *работой уровня* 0. Управляющая работа называется *работой уровня* $n + 1$, если максимальная длина пути до листовой работы не больше n, т. е. все подчиненные работы имеют уровень не больше n. Промежуточный отчет управляющей работы определяется как $\sum R(p)$ (суммирование по всем подчиненным работам). Тогда итоговый отчет есть $R(r) = \sum R(p)$ (суммирование по всем листовым работам).

Нельзя предполагать идемпотентность операции $+$, $a + a = a$. Это естественно предположить только для промежуточных отчетов листовых работ, которые не опираются на другие промежуточные отчеты. Поэтому A группу не образует, так как для некоторых $a \in A$ может отсутствовать обратный элемент $-a$. Таким образом, A только моноид (полугруппа).

В отличие от [9] простая работа разделена на две части: основную часть и подготовку промежуточного отчета. Только листовые работы идемпотентны, а подготовка промежуточного отчета управляющих работ не является идемпотентным.

Выполнение работ должно удовлетворять двум условиям:

(1) пока не выполнена основная часть управляющей работы, не могут выполняться подчиненные ей листовые или управляющие работы;

(2) пока не подготовлены промежуточные отчеты для всех подчиненных работ, не может быть подготовлен промежуточный отчет управляющей работы.

2.3 Примеры

Оба примера связаны с параллельной (распределенной) файловой системой (ФС). *Параллельная* ФС — это такая ФС, которая допускает параллельное обращение многих клиентов-пользователей, возможно, расположенных на многих узлах локальной или глобальной сети. *Распределенной* ФС называется такая

ФС, которая использует для хранения данных кластер; подобная система предоставляет пользователю единое глобальное пространство имен файлов и полностью скрывает от него действительное физическое расположение файлов.

Предполагается, что в ФС отсутствуют как символические (symbolic), так и жесткие (hard) ссылки. В этом случае ФС представляет собой направленное дерево. Вершины его — это файлы и каталоги.

2.3.1 Пример с индексацией документов файловой системы

Будем считать, что в ФС хранятся только текстовые файлы, т. е. *документы*. Имеется функция $y = F(x)$, которая по документу x выдает индекс y — множество пар $\langle k, i\rangle$, где k — ключевое слово, а i — количество вхождений этого ключевого слова в документ, причем в этом множестве y ключевое слово встречается не более одного раза. Задача состоит в том, чтобы каждый документ и каждый каталог получили свои индексы. Индекс директории определяется индуктивно. Если каталог имеет только документы (подкаталогов нет), то индекс каталога — это индекс конкатенации всех файлов, входящих в каталог. Такой каталог назовем каталогом уровня 0. Если уже определен индекс каталога уровня n, то каталог уровня $n+1$ определим как индекс всех файлов, входящих в каталог, и конкатенацию индексов всех каталогов, не больших, чем n. Отсюда следует определение операции $+$. Очевидно, что не для всякого индекса a имеется $-a$. Индекс документа или каталога должен находиться в том же каталоге, что и сам документ или каталог.

Листовая работа заключается только в подготовке промежуточного отчета, который представляет индекс документа, основная часть отсутствует. Основная часть управляющей работы — это формирование листовых работ, которую можно выполнить, а подготовка промежуточного отчета — это создание индекса каталога. Таким образом, для листовых работ имеет место идемпотентность, а для управляющих работ — нет.

2.3.2 Пример с копированием файловой системы

Требуется сдублировать ФС, т. е. создать ее копию в пределах той же ФС. Игнорируем некоторую некорректность этой задачи, связанную с расположением копии.

Основная часть листовой работы представляет собой создание требуемого файла, перепись содержимого этого файла из оригинала в копию или создание в копии каталога, не содержащего ни файлов, ни подкаталогов. Промежуточный отчет листовой работы — это время, которое потребовалось на выполнение основной части. Основная часть управляющей работы — это создание каталога в копии, который содержит в оригинале подкаталоги и/или файлы. Далее

определяются листовые работы, которые можно начать выполнять. При завершении всех подчиненных работ изготавливается промежуточный отчет. Интересен только итоговый отчет — если используются при работе k узлов, то итоговый отчет — это суммарное время, потраченное всеми узлами для выполнения задания. Это время можно соотнести с абсолютным временем выполнения задания, чтобы узнать, насколько эффективно используется распараллеливание. И здесь для листовых работ имеет место идемпотентность, а для управляющих работ — нет. При отказах узлов времена повторенных листовых работ, которые не были доведены до конца, но повторены, не будут учтены.

3 Нумерация вершин деревьев

Пусть T — конечное направленное дерево, а $|v(T)|$ — мощность множества вершин, т. е. число вершин T. Назовем *нумерацией* дерева T функцию f, $f : v(T) \to N$ (N — натуральный ряд), которая удовлетворяет следующим условиям:

$$\forall a(a \in v(T))(0 \le f(a) \le |v(T)| - 1)\,;$$
$$\forall a(a \in v(T))\forall b(b \in v(T))(a \ne b \supset f(a) \ne f(b))\,.$$

Нумерацию можно рассматривать как способ задания порядка обхода дерева: вершины обходятся в порядке возрастания их номеров.

Важный пример нумерации — нумерация, которая удовлетворяет такому дополнительному условию:

$$\forall a(a \in v(T))\forall b(b \in v(T))(a \to b \supset f(a) < f(b))\,.$$

Это условие означает, что для любой вершины все ее потомки посещаются позже, чем она сама. Дальше рассматриваются только такие нумерации, поскольку требуется, чтобы выполнялось условие «пока не выполнена управляющая работа, не могут выполняться подчиненные ей листовые или управляющие работы». Кроме того, дальше будут рассматриваться *вычислимые нумерации*. Вычислимой называется такая нумерация, которая, будучи применена к направленному графу, выдает нумерацию, определенным образом описанному, выдает номера его вершин. Приводимые ниже примеры нумераций являются вычислимыми. Вычислимых нумераций, которые удовлетворяют приведенному условию, счетное число.

3.1 DFS-нумерация

В основе этой нумерации лежит алгоритм поиска в глубину (DFS — Depth-first search). *DFS-нумерация* описывается в [11] так:

1. Начать с выделенной вершины (например, с корня).
2. Присвоить вершине минимальный незанятый номер.
3. Найти не обойденную смежную вершину и повторить процедуру для нее (спуск), начиная с п. 2.
4. Если не обойденной смежной вершины нет, подняться к вершине, с которой происходил спуск, и повторить процедуру для нее (подъем), начиная с п. 2.

3.2 BFS-нумерация

В основе этой нумерации лежит алгоритм поиска в ширину (BFS, Breadth-first search). *BFS-нумерация* описывается следующим образом [11].

1. Начать с выделенной вершины (например, с корня), присвоить ей метку 0 и поместить в список FIFO. $N = 1$.
2. Присвоить смежным вершинам данной вершины с меткой N, которые не имеют меток, метку $N + 1$ и поместить их в список FIFO.
3. $N = N + 1$. Если меток с N нет, перейти к п. 4, иначе перейти к п. 2 процедуры.
4. Перенумеровать вершины в списке FIFO.

Замечание. От виртуального времени, введенного Лэмпортом [12], легко перейти к нумерации вершин DAG (Direct Acyclic Graph). Получается нумерация с обходом в ширину.

4 Производные графы для дерева работ

4.1 D_1-производный граф для дерева работ T

D_1-производный граф для T обозначается как $D_1(T)$. Все его вершины — это пары $\{X, Y\}$ такие, что $X \in v(T)$, $Y \in v(T)$, $X \cap Y = \varnothing$, $X \cup Y = v(T)$. Корнем является вершина R такая, что $\{\{r\}, v(T)\backslash\{r\}\}$. Дуга $\{X_1, Y_1\} \to \{X_2, Y_2\}$ присутствует, только если $x \in X_1$, $y \in Y_1$, $x \to y$, причем $X_2 = X_1 \cup \{y\}$, $Y_2 = Y_1\backslash\{y\}$.

Теорема 1. *Граф $D_1(T)$ не имеет циклов и представляет собой DAG.*

Доказательство. Рассмотрим дугу $\{X_1, Y_1\} \to \{X_2, Y_2\}$. Из определения графа $D_1(T)$ следует, что $|X_2| = |X_1|+1$, $|Y_2| = |Y_1|-1$. Поскольку компонент X монотонно возрастает при перемещении по любому пути, циклов быть не может.

Следствие 1. *В графе $D_1(T)$ имеется единственная вершина, которая не имеет потомков — это вершина $\{v(T), \varnothing\}$. Назовем эту вершину конечной (E).*

Следствие 2. *DAG $D_1(T)$ не является деревом. Пример: дерево работ T имеет 3 вершины $v(T) = \{x, y, z\}$ и дуги $x \to y$, $x \to z$. DAG $D_1(T)$: корень $R = \{\{x\}, \{y, z\}\}$; промежуточные вершины — $\{\{x, z\}, \{y\}\}$ и $\{\{x, y\}, \{z\}\}$, конечная вершина $E = \{\{x, y, z\}, \varnothing\}$. Дуги*:

- $R \to \{\{x, z\}, \{y\}\}$;
- $R \to \{\{x, y\}, \{z\}\}$;
- $\{\{x, z\}, \{y\}\} \to E$;
- $\{\{x, y\}, \{z\}\} \to E$.

Следствие 3. *Любая дуга, принадлежащая $D_1(T)$, принадлежит пути такому, что $R \Rightarrow E$.*

Следствие 4. *Любой путь такой, что $R \Rightarrow E$, имеет длину (количество дуг в нем), равную $|v(T)| - 1$.*

4.2 D_2-производный граф для дерева работ T

D_2-производный граф для T обозначается как $D_2(T)$. Все его вершины — это тройки $\{X, Y, Z\}$ такие, что $X \in v(T)$, $Y \in v(T)$, $Z \in v(T)$, $X \cap Y = \varnothing$, $X \cap Z = \varnothing$, $Y \cap Z = \varnothing$, $X \cup Y \cup Z = v(T)$. Корнем является вершина R $\{\{r\}, \varnothing, v(T)\backslash\{r\}\}$. Дуга $\{X_1, Y_1, Z_1\} \to \{X_2, Y_2, Z_2\}$ присутствует, если:

- $x \in X_1$, $z \in Z_1$, $x \to z$, причем $X_2 = X_1$, $Y_2 = Y_1 \cup \{z\}$, $Z_2 = Z_1\backslash\{z\}$ (дуга 1);
- если $y \in Y_1$, причем $X_2 = X_1 \cup \{y\}$, $Y_2 = Y_1\backslash\{y\}$, $Z_2 = Z_1$ (дуга 2).

Теорема 2. *Граф $D_2(T)$ не имеет циклов и представляет собой DAG.*

Доказательство. Рассмотрим дугу $\{X_1, Y_1, Z_1\} \to \{X_2, Y_2, Z_2\}$. Из определения графа $D_2(T)$ следует, что

- $|X_2| = |X_1|$, $|Y_2| = |Y_1| + 1$, $|Z_2| = |Z_1| - 1$ или
- $|X_2| = |X_1| + 1$, $|Y_2| = |Y_1| - 1$, $|Z_2| = |Z_1|$.

Поскольку сумма $|X| + |Y|$ монотонно возрастает при движении по любому пути, циклов быть не может.

Следствия 1–3 из теоремы 1 выполняются, хотя пример из следствия 2 следует заменить. Следствие 3: любой путь такой, что $R \Rightarrow E$, имеет длину (количество дуг в нем), равную $2|v(T)| - 2$.

4.3 Семантика производных графов для дерева работ T

Производный DAG $D_1(T)$ описывает все возможные варианты выполнения задания. При этом считается, что любая листовая или управляющая работа делается «мгновенно», ей соответствует переход по одной дуге. Вершины $\{X, Y\}$ производного DAG $D_1(T)$ отражают следующее:

- X — множество листовых и управляющих работ, которые выполнены;
- Y — множество листовых и управляющих работ, которые не выполняются, так как не обнаружены.

Производный DAG $D_2(T)$ описывает все варианты выполнения задания, учитывающие ограничения на распараллеливание. Вершины $\{X, Y, Z\}$ производного DAG $D_2(T)$ отражают следующее:

- X — множество листовых и управляющих работ, которые выполнены;
- Y — множество листовых и управляющих работ, которые параллельно выполняются;
- Z — множество листовых и управляющих работ, которые еще не обнаружены.

Дуга 1 соответствует порождению новой листовой или управляющей работы z. Она сразу же начинает выполняться. Дуга 2 соответствует завершению работы.

Любой путь $R \Rightarrow E$ называется *обходом дерева* T.

Принимающееся решение, т. е. вычисление z, которое фигурирует в дуге 1, делается на основании X_1, Y_1 и Z_1. Например, можно учитывать ограничение $|Y_1| \leq C$ (общее количество параллельно выполняемых листовых и управляющих работ ограничено). Аналогично решение, которое принимается при следовании по дуге 2, зависит от того, выполнена ли или нет работа. Скажем, это может быть связано со временем, которое требуется для выполнения работы.

5 История выполнения задания

Выполнение задания осуществляют его *участники*, т. е. процессы или потоки (threads) одного процессора или многих процессоров, если они образуют кластер. Будем считать, что имеется конечное множество участников $W = \{w_1, w_2, \ldots, w_n\}$, каждый из которых может выполнить любую из листовых и управляющих работ, когда ее можно будет выполнять. Один участник не может одновременно делать больше одной работы, хотя он может простаивать.

С каждой вершиной дерева работ свяжем ее состояние, которое может со временем меняться:

- «*не обнаружена*»;
- «*обнаружена*». Работа еще не назначена никому из участников;

– «*выполняется основная часть*». За выполнение работы ответствен какой-то участник, который выполняет основную часть работы. Если это листовая работа, то основная часть и промежуточный отчет делаются вместе;
– «*завершена основная часть*». Основная часть управляющей работы выполнена, но не сделан промежуточный отчет этой работы, потому что не выполнены подчиненные ей управляющие и листовые работы;
– «*изготавливается промежуточный отчет*». Только для управляющей работы. Изготовление промежуточного отчета;
– «*полностью завершена*». Выполнена как основная часть, так и подготовлен промежуточный отчет.

Пусть имеется дерево работ T. *История выполнения задания* — последовательность T_i, $0 \le i < N$ (N — длина истории), того же дерева работ, однако каждой вершине приписано состояние, причем у разных членов последовательности приписанные состояния отличаются. Первым членом T_0 последовательности является дерево, в котором одна единственная вершина — корень дерева r — находится в состоянии «*обнаружена*»; все остальные вершины имеют состояние «*не обнаружена*». Последним членом T_{N-1} является дерево, все вершины которого находятся в состоянии «*полностью завершена*». Состояние каждой вершины меняется в строго оговоренной последовательности. Если вершине соответствует управляющая работа, смена состояний совпадает с перечислением состояний. Для листовых работ смена состояний такая:

– «*не обнаружена*»;
– «*обнаружена*»;
– «*выполняется основная часть*»;
– «*полностью завершена*».

Предшественник вершины, которая меняет свое состояние с «*не обнаружена*» на «*обнаружена*», должен иметь состояние, отличающееся от состояния «*не обнаружена*». Переход от одного члена истории выполнения задания к следующему зависит от времени выполнения соответствующего действия, от этих времен зависит также вся последовательность.

Кроме того, каждой вершине, находящейся в состоянии «*выполняется основная часть*», «*обнаружена*» или «*изготавливается промежуточный отчет*», соответствует участник, который этим занимается.

Расширим требования, которым должна подчиняться история выполнения задания:

– пока не выполнена основная часть управляющей работы, не могут выполняться подчиненные ей листовые или управляющие работы;
– пока не подготовлены промежуточные отчеты для всех подчиненных работ, не может быть подготовлен промежуточный отчет управляющей работы;
– каждый участник не может делать больше одной работы одновременно.

Отсюда следуют следующие утверждения, касающиеся члена истории выполнения задания T_i, например такие:

- если некоторая вершина находится в состоянии «*не обнаружена*», то все вершины, ей подчиненные (потомки), находятся в том же состоянии;
- если некоторая вершина находится в состоянии «*полностью завершена*», то все ее потомки находятся в том же состоянии.

Назовем *координатором* процесс-участник, который делает следующее:

- обнаруживает новые работы, т. е. переводит вершины из состояния «*не обнаружена*» в состояние «*обнаружена*»;
- назначает свободного участника для выполнения основной части работы, т. е. переводит вершины из состояния «*обнаружена*» в состояние «*выполняется основная часть*».

Для того чтобы принять решение о назначении, координатор должен располагать сведениями об освобождении участников. Это означает, что участник должен сигнализировать о завершении работы, которой он занят. Принципиальным является вопрос о том, сколько координаторов управляют выполнением задания. Варианты: один; больше одного, но фиксированное число; все участники. Распределение остальных работ, а именно: выполнение основной части и подготовка промежуточного отчета управляющей работы — зависит от типа задания и может варьироваться. Обычно, например при реализации MapReduce [1, 4, 8], предполагается, что распорядитель (координатор) — один-единственный, причем он не делает листовые работы и не подготавливает промежуточные отчеты. Все остальные участники — это исполнители, которые занимаются листовыми работами. В [10] нет понятия промежуточных отчетов; основная часть управляющих отчетов делается координатором. Все остальные — это исполнители.

6 Выполнение задания

Ясно, что критерием эффективности выполнения задания является время его выполнения при определенных ограничениях (конечно, эти ограничения взаимосвязаны). Рассмотрим ограничения, которые должны соблюдаться при эффективном выполнении задания.

1. Координаторы, которые планируют выполнение задания, в частности выбирают способ обхода дерева работ, не должны иметь слишком большой контекст. Под *контекстом координатора* понимаются данные, которые необходимо хранить координатору для своих работ. Например, этот контекст должен умещаться в основную память.
2. Исполнители, которые выполняют простые работы, не должны иметь слишком большой контекст. Под *контекстом исполнителя* понимаются данные, которые необходимо хранить исполнителю для своих работ.

3. Число координаторов и исполнителей должно удовлетворять ограничениям кластера.
4. Должны быть приемлемыми объемы служебных пересылок между узлами, на которых находятся координаторы и исполнители.
5. Должна быть обеспечена такая степень параллелизма, при которой не простаивают простые работы, готовые к выполнению.
6. Восстановление выполнения задания после отказа узла или узлов не должно быть расточительным. Идеальный случай — гарантируется, что повторяются только работы, порученные отказавшим узлам. Если считать, что многократные отказы происходят значительно реже однократных, можно ослабить это требование. А именно, потребовать, что при однократных отказах повторяются только пострадавшие работы, а при многократных отказах может повторяться существенно больший объем работ.
7. Узел, добавленный к кластеру, сразу же включается в выполнение текущего задания.

Огромную роль здесь играет способ обхода дерева работ, который задается выбранной нумерацией, описанной в разд. 3 и подразд. 4.2 и 4.3. Предпочтительным, как мне представляется, является обход, похожий на DFS. Я не могу точно сформулировать, что такое «похожий на DFS». Это же касается BFS. DFS-алгоритмы обхода в большинстве случаев требуют меньше памяти для контекста координатора, чем BFS-алгорит мы (это открытая проблема[1]), хотя всегда DFS-алгоритмы требуют не больше памяти, чем BFS-алгоритмы (это тоже не доказано).

Остановимся на самом простом варианте: один координатор, который делает то, что ему положено по роли, однако не выполняет листовых работ. Имеется, кроме того, N исполнителей, в задачи которых входит выполнение листовых работ и подготовка промежуточных отчетов для управляющих работ. Поэтому суммарное количество вершин, которые одновременно находятся в состояниях «*выполняется основная часть*» или «*изготавливается промежуточный отчет*», не должно превышать N. Однако, если допустить очередь к исполнителю, от этого ограничения можно отказаться. Некоторые подходы к эффективному выполнению заданий намечены в [9, 10].

7 Заключение

Описанная выше модель позволяет точно сформулировать определенный круг вопросов, которые касаются эффективного выполнения заданий. Отметим среди них следующие проблемы:

[1] Эта проблема сродни проблемам по асимптотической сложности, которые решал О. Б. Лупанов. Однако она осложнена тем, что нумерация должна быть вычислимой.

1. Можно ли спроектировать эффективный алгоритм, в котором все участники равноправны, т. е. любой участник выступает и как координатор, и как исполнитель? Привлекательность подобного алгоритма в том, что «все знают всё». Эта симметрия позволяет надеяться, что отказ любого количества узлов (но не всего кластера в целом) приведет к повтору только пострадавших узлов.
2. Как спроектировать эффективный алгоритм с фиксированным количеством координаторов k, $k = K$, $K \geq 2$. Если проблема 1 получает отрицательное решение (такой алгоритм невозможен или неэффективен), можно построить эффективный алгоритм, ограничив количество координаторов небольшим числом.
3. Какой обход дерева выбрать, исходя из $D_2(T)$? Этот вопрос может иметь разные ответы в зависимости от K (числа координаторов).
4. Следует ли допускать очереди к исполнителям?

Литература

1. *Dean J., Ghemawat S.* MapReduce: Simplified data processing on large clusters // Communications of the ACM — 50th anniversary issue: 1958–2008, 2008. Vol. 51. No. 1. P. 107–114.
2. *Кузнецов С. Д.* MapReduce: внутри, снаружи или сбоку от параллельных СУБД? // Труды Института системного программирования РАН / Под ред. В. П. Иванникова. — М.: ИСП РАН, 2010. Т. 19. С. 35–70.
3. *Ghemawat S., Gobioff H., Leung S.-T.* The Google File System // 19th ACM Symposium on Operating Systems Principles Proceedings. — N.Y., 2003. Vol. 37. Issue 5. P. 29–43.
4. Hadoop MapReduce Home Page. URL: http://hadoop.apache.org/mapreduce/ (дата обращения: 10.03.2011).
5. Apache Hadoop Home Page. URL: http://hadoop.apache.org/ (дата обращения: 10.03.2011).
6. The Apache Software Foundation Home Page. URL: http://www.apache.org/ (дата обращения: 10.03.2011).
7. Hadoop Distributed File System Home Page. URL: http://hadoop.apache.org/hdfs/ (дата обращения: 10.03.2011).
8. MapReduce Tutorial. URL: http://hadoop.apache.org/mapreduce/docs/current/mapred_tutorial.html (дата обращения: 10.03.2011).
9. *Козмидиади В. А.* Дерево работ и массово-параллельная обработка // Информатика и её применения, 2011. Т. 5. Вып. 2. С. 90–99.
10. *Дьячков В. А., Захаров В. Н., Козмидиади В. А., Кузьмин А. В., Попов А. С., Шулятников Д. С.* Сервисные приложения для работы с иерархическими данными в распределенной файловой системе // Системы и средства информатики / Под ред. И. А. Соколова. — М.: Наука, 2008. Вып. 18. С. 20–35.
11. *Knuth D. E.* The art of computer programming. Vol. 1. — 3rd ed. — Boston: Addison-Wesley, 1997. 650 p.
12. *Lamport L.* Time, clocks, and the ordering of events in a distributed system // Communications of the ACM, 1978. Vol. 21. No. 7. P. 558–565.

МЕТОД ПОВЫШЕНИЯ ОТКАЗОУСТОЙЧИВОСТИ СИСТЕМ ПОДДЕРЖАНИЯ КОГЕРЕНТНОСТИ КЭШ

Б. З. Шмейлин[1]

Аннотация: Представлен метод повышения отказоустойчивости системы поддержания когерентности кэш-памяти с протоколом наблюдения. В предлагаемом методе кодируются запросы, состояния и сигналы в схеме реализации протокола. Для достижения устойчивости к кратковременным отказам в логике поддержания когерентности к закодированным строкам запросов, состояний и сигналов применяется код, корректирующий одиночные и обнаруживающий двойные ошибки.

Ключевые слова: когерентность кэш; протоколы наблюдения; отказоустойчивость; корректирующие коды

1 Введение

В современных микропроцессорных системах все возрастающее значение приобретает их отказоустойчивость. Отказоустойчивость — это свойство системы, которое обеспечивает ей возможность продолжения действий, заданных программой, после возникновения неисправностей. Введение отказоустойчивости требует избыточного аппаратного и программного обеспечения. В настоящее время микропроцессорные системы подвержены кратковременным отказам из-за уменьшающихся размеров кристаллов, плотности размещения элементов и снижения напряжения питания. Кратковременные отказы (сбои), вызванные как различными воздействиями окружающей среды, так и внутренними перекрестными связями, становятся более частыми явлениями, чем отказы аппаратуры, обусловленные их производством.

В последнее время все большее развитие получают многопроцессорные системы (МС) как с разделенной, так и с общей памятью. В этих системах каждый процессор непосредственно связан со своей частной кэш-памятью. При этом возникает проблема поддержания согласованности или когерентности кэш-памяти. Когерентность кэш-памятей обеспечивает возможность какому-либо процессору получить достоверную копию данных после модификации их другим процессором. Для поддержания когерентности кэш-памятей служит специальный механизм — протокол когерентности. Одним из распространенных протоколов

[1]Институт проблем информатики Российской академии наук, shmeilin@mail.ru

является протокол наблюдения (snooping protocol). Работа по этому протоколу заключается в наблюдении контроллером частной кэш-памяти за событиями на шине общей памяти. Сбой в системе поддержания когерентности приводит к серьезным последствиям. Так, например, наличие недостоверной копии строки может привести к распространению ошибки сразу в нескольких процессорах.

Данная статья посвящена разработке схемы контроллера кэш, реализующего алгоритм поддержания когерентности по протоколу наблюдения и устойчивого к ошибкам, вызванных кратковременными отказами. Вопросам контроля и повышения отказоустойчивости схем поддержания когерентности в последние годы уделяется большое внимание [1–4]. В работах [2, 4] исследуются отказы и сбои системы с протоколом когерентности на основе справочника. В [4] предлагаются схемы контроля и коррекции строк справочника, работа [2] посвящена вопросам отказоустойчивости межпроцессорных связей в протоколах со справочником. В статье [3] рассматривается отказоустойчивая система с маркерами. В статье [1] предлагается устройство контроля работы системы с протоколом наблюдения. Это устройство контроля позволяет обнаружить ошибки в логике поддержания когерентности, однако оно не предназначено для повышения отказоустойчивости схемы, реализующей эту логику.

2 Протоколы наблюдения поддержания когерентности кэш-памятей

Рассмотрим структуру многопроцессорной системы, представленную на рис. 1. Процессор посылает запрос на чтение или запись на свою кэш-память.

При запросе на чтение и наличии валидной строки в кэш происходит чтение данных. Если таковая строка отсутствует в кэш-памяти, запрос через контроллер

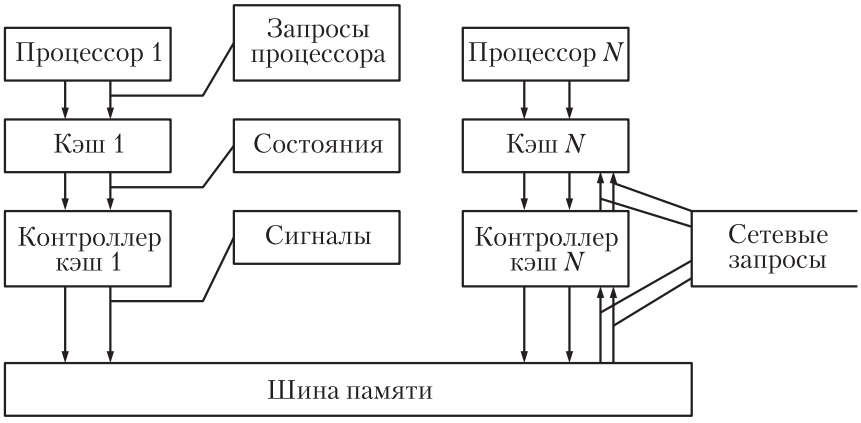


Рис. 1 Структура многопроцессорной системы

кэш-памяти передается на шину. Затем данные передаются из кэш-памяти, обладающей ими. Если такая кэш-память не находится, то данные передаются из главной памяти. При запросе процессора на запись в свою кэш-память управление должно выдать разрешение на запись, т. е. разрешить конфликт между несколькими процессорами, выставившими запросы на запись. После получения разрешения процессор пишет в валидную строку, имеющуюся на данный момент в его кэш-памяти, и помечает эту строку как модифицированную. Если таковой строки не находится, то запрос передается на шину главной памяти, так же как и при чтении. При получении запрашиваемой строки процессор записывает в нее информацию и при этом посылается запрос по шине на инвалидацию этой строки во всех кэш-памятях других процессоров и в главной памяти.

Имеется несколько протоколов наблюдения. Мы рассмотрим в этой работе два наиболее распространенных протокола — MESI и MOESI. В названиях этих протоколов использована аббревиатура из названий состояний. В протоколах поддержания когерентности кэш-памяти каждой строке присваивается одно из рассматриваемых ниже состояний:

- M — модифицированное, достоверная копия строки имеется только в данной кэш-памяти. Кэш должен затем записать строку в главную память перед тем, как другой процессор даст запрос на чтение. После записи в главную память состояние строки меняется на E;
- E — эксклюзивное, достоверная копия строки имеется только в данной кэш-памяти и главной памяти. При удовлетворении запроса на чтение другим процессором состояние строки меняется на S;
- S — разделяемое состояние, достоверная копия строки имеется в главной памяти и в кэш-памятях других процессоров. При удовлетворении запроса на запись состояние строки меняется на I;
- I — невалидное (недостоверное) состояние;
- O — состояние владельца похоже на состояние S, отличие его в том, что только строка в одной кэш-памяти может быть в состоянии O, а остальные — в состоянии S.

Использование протокола MOESI позволяет оптимизировать процесс поиска кэш-памяти с запрашиваемой строкой, обратившись к кэш-памяти, строка в которой имеет состояние O.

При работе по этим протоколам имеются следующие запросы от процессора и сетевые сигналы:

- PrRd — запрос на чтение от локального процессора;
- PrWr — запрос на запись от локального процессора;
- BusRd — сигнал и запрос на чтение от сети;

– BusRdX — эксклюзивный сигнал и запрос на чтение от сети;

– BusWB — сигнал обратной запись из кэш-памяти;

– Flush — сигнал и запрос на инвалидацию.

2.1 Протокол MESI

В табл. 1 представлены запросы от процессора и сетевые запросы, переходы из одного состояния в другое и сигналы для протокола MESI.

Локальный запрос процессора на чтение (PrRd) в любом состоянии, кроме невалидного, обслуживается внутри узла и не вызывает действий сети. Такие запросы не изменяют состояние строки. Чтение невалидной строки (или строки, отсутствующей в данной кэш-памяти) приводит к запросу сети BusRd на чтение из памяти. Рассматриваемая здесь архитектура использует подход, при котором другие кэш-памяти по возможности отвечают на запрос по чтению. Если другие кэш-памяти не содержат запрашиваемых данных, то отвечает главная память. Если на запрос отвечает главная память, то строка кэш не представлена в других

Таблица 1 Запросы, переходы из одного состояния в другое и сигналы для протокола MESI

Запрос	Исходное состояние	Конечное состояние	Сигнал
PrRd	M	M	—
PrRd	E	E	—
PrRd	S	S	—
PrRd	I	E/S	BusRd
PrWr	M	M	—
PrWr	E	M	—
PrWr	S	M	Flush
PrWr	I	M	BusRdX
BusRd	M	S	BusWB
BusRd	E	S	BusWB
BusRd	S	S	BusWB
BusRd	I	I	—
BusRdX	M	I	BusWB
BusRdX	E	I	BusWB
BusRdX	S	I	BusWB
BusRdX	I	I	—
Flush	M	—	SigErr
Flush	E	—	SigErr
Flush	S	I	—
Flush	I	I	—

кэш, так что ее состояние становится E. В противном случае другие кэш-памяти разделяют данные, так что состояние становится S.

Локальная запись процессора (PrWr) в строку с состоянием M не меняет состояния и не вызывает сообщений на шине. Этот же запрос к строке с состоянием E меняет состояние строки на M, но не приводит к трафику по сети.

Запрос на запись PrWr в строку с состоянием S меняет состояние на M и распространяет запрос по шине на инвалидацию — Flush, что делает невалидными соответствующие строки в других кэш-памятях.

Если возникает промах при локальной записи либо подлежащая записи строка невалидна (состояние I), то на шину выдается сигнал BusRdX — эксклюзивный запрос на чтение от сети и инвалидацию. При получении валидной копии строки осуществляется запись и строка переходит в состояние M.

Когда контроллер кэш «видит» на шине запрос на чтение (BusRd) из имеющейся в данной кэш-памяти валидной строки, он меняет состояние этой строки на S (или оставляет строку в этом состоянии) и выдает сигнал на шину об обратной записи данных в главную память — BusWB. Если строка невалидна, то сигнал на шину не выдается.

Когда контроллер кэш «видит» на шине эксклюзивный сигнал и запрос на чтение (BusRdX), он меняет состояние этой строки на I (или оставляет строку в этом состоянии) и в случае валидной строки выдает сигнал на шину об обратной записи данных в главную память — BusWB. Если строка невалидна, то сигнал на шину не выдается.

При запросе Flush из сети строка переходит в состояние I, если до этого она была в состоянии S, или остается в состоянии I. При этом на шину сигнал не выдается. Если строка была в состоянии M или E, то возникает сигнал об ошибке (SigErr), так как это недопустимая ситуация — запрос Flush посылается только для строк с состоянием S.

2.2 Протокол MOESI

В табл. 2 представлены запросы от процессора и сетевые запросы, переходы из одного состояния и сигналы для протокола MESI.

Отличия протокола MOESI от протокола MESI состоят в следующем:

- при запросе на чтение процессора (PrRd) строка остается в состоянии O;
- при запросе на запись процессора (PrWr) строка из состояния O переходит в состоянии M и выдает сигнал на инвалидацию (Flush);
- при сетевом запросе на чтение (BusRd) строка остается в состоянии O;
- при эксклюзивном запросе и запросе на инвалидацию переход из состояния строки O аналогичен переходу из состояния S.

Таблица 2 Запросы, переходы из одного состояния в другое и сигналы для протокола MOESI

Запрос	Исходное состояние	Конечное состояние	Сигнал
PrRd	M	M	—
PrRd	O	O	—
PrRd	E	E	—
PrRd	S	S	—
PrRd	I	E/S	BusRd
PrWr	M	M	—
PrWr	O	M	Flush
PrWr	E	M	—
PrWr	S	M	Flush
PrWr	I	M	BusRdX
BusRd	M	O	BusWB
BusRd	O	O	BusWB
BusRd	E	S	BusWB
BusRd	S	S	BusWB
BusRd	I	I	—
BusRdX	M	I	BusWB
BusRdX	O	I	BusWB
BusRdX	E	I	BusWB
BusRdX	S	I	BusWB
BusRdX	I	I	—
Flush	M	—	SigErr
Flush	O	I	—
Flush	E	—	SigErr
Flush	S	I	—
Flush	I	I	—

Перечисленные отличия обусловлены тем, что состояние O подобно состоянию S. Исключение составляет сетевой запрос на чтение — BusRd, при котором состояние M меняется на O, так как запрашиваемая строка при таком запросе будет находиться в нескольких кэш-памятях и данный процессор становится владельцем этой строки.

2.3 Классификация ошибок в логике поддержания когерентности

Следует отметить, что не все ошибки в логике реализации протокола когерентности приводят к искажению данных в кэш-памяти процессора, а лишь к потере производительности. Разделим все ошибки на три категории. К первой отнесем фатальные ошибки, т. е. ошибки, которые могут привести к неверному выполнению программы. Ко второй категории отнесем ошибки, которые приво-

дят лишь к потере производительности. И к третьей — остальные ошибки, т. е. ошибки, не влияющие на работу системы.

Так, к первой категории относятся такие ошибки, при которых состояние строки кэш-памяти вместо I меняется на любое другое. При этом процессор читает недостоверную копию данных или при записи данных процессором не будет выдан сигнал на эксклюзивный запрос, т. е. копии строки в кэш-памятях других процессоров не будут инвалидированы. К этой же группе относятся также ошибки, при которых любое состояние меняется на M. При этом процессору разрешается писать в данный блок, так как он считает, что у него имеется единственная копия данных.

Ко второй категории относятся такие, при которых состояние строки кэш-памяти вместо M, O, E или S меняется на I. При этом контроллер кэш, считая строку невалидной, запрашивает данные из кэш-памяти другого процессора или из главной памяти, что и приводит к потере производительности. К этой же группе относятся также ошибки, при которых состояние M меняется на любое другое. При этом вырабатывается сигнал эксклюзивный либо на инвалидацию, что приводит к дополнительному трафику по сети.

К третьей группе ошибок относятся такие, которые приводят, например, к изменению состояния S на E или наоборот.

В табл. 3 приведена полная классификация ошибок по категориям. В столбцах таблицы показаны исходные состояния, в рядах — искаженные, а на пересечении столбца и колонки — соответствующая категория ошибки.

Таблица 3 Классификация ошибок в протоколах поддержания когерентности

Исходное состояние	Ошибочное состояние				
	M	O	E	S	I
M	X	2	2	2	2
O	1	X	2	2	2
E	1	3	X	3	2
S	1	3	3	X	2
I	1	1	1	1	X

3 Отказоустойчивая схема поддержания когерентности кэш-памяти

С целью создания отказоустойчивой схемы поддержания когерентности кэш-памяти, реализующей рассмотренные выше протоколы, закодируем все запросы, состояния и сигналы, а затем применим код, корректирующий одиночные и обнаруживающий двойные ошибки. Эти коды, называемые кодами SECDED (single error correction double error detection), широко применяются при проектировании модулей памяти [5].

3.1 Кодирование запросов, состояний и сигналов

Протокол MESI. В табл. 4–6 для протокола MESI приведена кодировка запросов состояний и сигналов соответственно.

Тогда каждая строка табл. 1 может быть представлена в виде векторов запросов, состояний и сигналов. Эти векторы для протокола MESI представлены в табл. 7. Для коррекции и обнаружения ошибок в логике поддержания когерентности применим к векторам запросов, состояний и сигналов один из SECDED кодов, например код Хэмминга [6]. Так как каждый вектор включает 24 бита, то применим код 31,26, т. е. код с 26-ю информационными разрядами и 5-ю контрольными разрядами.

Таблица 4 Кодирование запросов для протокола MESI

Запрос	Байт	Кодировка (шестнадцатеричная)
PrRd	0	80
PrWr	0	40
BusRd	0	20
BusRdX	0	10
Flush	0	08

Таблица 5 Кодирование состояний для протокола MESI

Состояние	Байт	Кодировка (шестнадцатеричная)
Предыдущее		
M	1	80
E	1	40
S	1	20
I	1	10
Последующее		
M	1	08
E	1	04
S	1	02
I	1	01

Таблица 6 Кодирование сигналов для протокола MESI

Сигнал	Байт	Кодировка (шестнадцатеричная)
BusRd	2	80
BusRdX	2	40
Flush	2	20
BusWB	2	10
SigErr	2	08

Таблица 7 Векторы запросов, состояний и сигналов для протокола MESI

Вектор	Содержимое	Вектор	Содержимое
1	808800	11	202210
2	804400	12	201100
3	802200	13	108110
4	801480	14	104110
5	408800	15	102110
6	404800	16	101100
7	402820	17	088008
8	401840	18	084008
9	208210	19	082100
10	204210	20	081100

В табл. 7 приведены векторы запросов, состояний и сигналов для протокола MESI.

Контрольные разряды формируются следующим образом.

Первый контрольный разряд — дополняет до четности информационные разряды с нечетными номерами — 1, 3, 5 и т. д. Второй контрольный разряд — дополняет до четности информационные разряды с номерами, у которых имеется 1 во втором разряде — 2, 3, 6, 7, 10 и т. д. Третий контрольный разряд — дополняет до четности информационные разряды с номерами, у которых имеется 1 в третьем разряде — 4, 5, 6, 7, 12, 13 и т. д.

Четвертый контрольный разряд — дополняет до четности информационные разряды с номерами, у которых имеется 1 в четвертом разряде — с 8 по 15 и 24.

Таблица 8 Кодирование состояний для протокола MOESI

Состояние	Байт	Кодировка (шестнадцатеричная)
Предыдущее		
M	1	80
E	1	40
S	1	20
I	1	10
O	0	04
Последующее		
M	1	08
E	1	04
S	1	02
I	1	01
O	0	02

Пятый контрольный разряд — дополняет до четности информационные разряды с номерами, у которых имеется 1 в пятом разряде — с 16 по 24.

Нарушение четности в каких-либо группах разрядов позволяет однозначно установить и скорректировать одиночную ошибку в любом информационном разряде. Если зафиксирована ошибка в незадействованных разрядах кода — разряды 6–8 байта 0, разряды 6–8 байта 2, то такая ошибка игнорируется. Для снижения избыточности в этих разрядах размещаются контрольные разряды.

Протокол MOESI. В табл. 8 приведена кодировка состояний для протокола MOESI.

Векторы запросов, состояний и сигналов для протокола MOESI представлены в табл. 9.

Таблица 9 Векторы запросов, состояний и сигналов для протокола MOESI

Вектор	Содержимое	Вектор	Содержимое
1	808800	14	202210
2	860000	15	201100
3	804400	16	108010
4	802200	17	140110
5	801480	18	104010
6	408800	19	102110
7	440810	20	101100
8	404800	21	088008
9	402820	22	0C0100
10	401840	23	084008
11	208210	24	082100
12	260010	25	081100
13	204210		

3.2 Реализация схемы коррекции в контроллере кэш

На рис. 2 представлен алгоритм работы контроллера кэш в части коррекции одиночных и обнаружения двойных ошибок в логике реализации протокола поддержания когерентности кэш. После получения информации о запросах, состояниях и сигналах эта информация кодируется в виде описанного ранее вектора. Этот вектор сравнивается с одним из эталонных векторов (табл. 6 для протокола MESI или табл. 8 для протокола MOESI). При несовпадении формируется SECDED код. При обнаружении двойной ошибки посылается соответствующее сообщение. При выдаче некоторых сигналов на шину памяти контроллер должен получить ответ. При отсутствии ответа происходит «зависание» системы. Чтобы избежать такой ситуации, включается таймер и по истечении установленного времени ожидания выводится сообщение об отсутствии ответа.

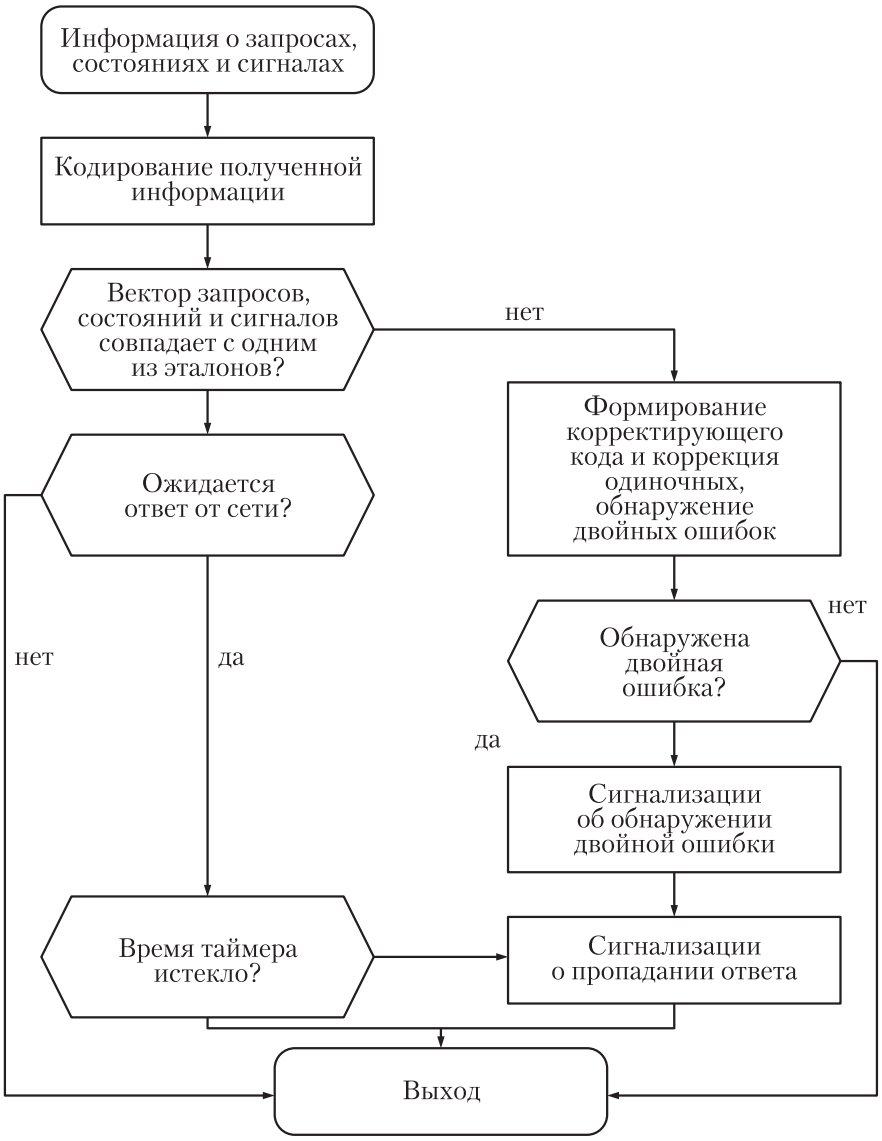


Рис. 2 Алгоритм работы контроллера кэш

Следует отметить, что предлагаемая схема коррекции предполагает определенные затраты на реализацию кода SECDED и возможно небольшую потерю производительности. Однако ошибки в логике когерентности могут привести к таким серьезным последствиям, что на обнаружение и корректировку потребуются большие затраты.

4 Выводы

1. На основании анализа работы по протоколу поддержания когерентности кэш выявлены возможные ошибки, вызванные кратковременными отказами.
2. Предложена классификация ошибок в логике реализации протокола когерентности кэш, согласно которой такие ошибки разделены на фатальные, т. е. влияющие на корректность выполнения программ, и влияющие только на производительность МС.
3. Для двух наиболее распространенных протоколов наблюдения — MESI и MOESI — предложена схема коррекции одиночных и обнаружения двойных ошибок с использованием кода SECDED.
4. Предложенная схема коррекции хотя и связана с введением избыточности в контроллере кэш и некоторой потерей производительности МС, однако ошибки в логике поддержания когерентности приводят к весьма серьезным последствиям.

Литература

1. *Borodin D., Juurlink B.* A low-cost cache coherence verification method for snooping systems // 11th EUROMICRO Conference on Digital System Design Architectures, Methods and Tools Proceedings, 2008. P. 219–227.
2. *Fernandez-Pascual R., Garcia J. M., Acacio M., Duato J.* A fault-tolerant directory-based cache coherence protocol for CMP architectures // IEEE Conference (International) on Dependable Systems and Networks With FTCS and DCC Proceedings, 2008. P. 267–276.
3. *Fernandez-Pascual R., Garcia J. M., Acacio M. E., Duato J.* Extending the TokenCMP cache coherence protocol for low overhead fault tolerance in CMP architectures // IEEE Trans. on Parallel and Distributed Systems Issue Date, 2008. Vol. 19. Issue 8. P. 1044–1056.
4. *Lee H., Cho S., Childers B.* PERFECTORY: A fault-tolerant directory memory architecture // IEEE Trans. on Computer, 2010. Vol. 59. No. 5. P. 638–650.
5. *Ando H.* Accelerated testing of 90 mm SPARC64V microprocessor for neutron SER // 3rd Workshop System Effects on Logic Soft Errors Proceedings, 2007. P. 187–196.
6. *Питерсон У., Уэлдон Э.* Коды, исправляющие ошибки. — М.: Мир, 1976. С. 593.

АНАЛИЗ НА САМОСИНХРОННОСТЬ НЕКОТОРЫХ ТИПОВ ЦИФРОВЫХ УСТРОЙСТВ*

***Ю. А. Степченков*[1], *Ю. Г. Дьяченко*[2], *Ю. В. Рождественский*[3], *Н. В. Морозов*[4]**

Аннотация: Представлен подход к проверке цифровой схемы на самосинхронность с использованием программных средств, реализующих событийный метод анализа. Показано, что стопроцентная тестовая полнота анализа на самосинхронность для регистров сдвига и памяти обеспечивается относительно простыми средствами. Предложена методика отладки схемы в процессе анализа на самосинхронность произвольной схемы. Обоснована необходимость иерархического подхода к анализу сложной схемы на самосинхронность.

Ключевые слова: самосинхронные схемы; анализ на самосинхронность; тестовая полнота анализа; замыкание; иерархический анализ

1 Введение

При создании устройств цифровой обработки данных в последние годы все большее значение приобретают такие их характеристики, как энергопотребление и надежность работы при воздействии спецфакторов и низком напряжении питания. В связи с этим возрастает интерес разработчиков больших интегральных схем (БИС) к классу самосинхронных (СС) схем. Этот класс включает в себя несколько подклассов; один из наиболее востребованных и имеющих практический интерес — подкласс схем, поведение которых не зависит от задержек элементов, входящих в их состав. В данной статье под самосинхронными понимаются именно такие схемы.

Создание СС-устройств требует подтверждения принадлежности разрабатываемой схемы к данной категории устройств. С этой целью используется анализ схемы на самосинхронность [1]. Эта задача чрезвычайно затратная, ее сложность зависит от количества входных и внутренних переменных схемы. Полнота анализа на самосинхронность определяется *алгоритмической* полнотой используемых программных средств и *тестовой* полнотой входных воздействий, задаваемых

*Работа выполнена при частичной финансовой поддержке по Программе фундаментальных исследований ОНИТ РАН на 2011 г. (проект 1.5).

[1]Институт проблем информатики Российской академии наук, YStepchenkov@ipiran.ru

[2]Институт проблем информатики Российской академии наук, YDiachenko@ipiran.ru

[3]Институт проблем информатики Российской академии наук, YRogdest@ipiran.ru

[4]Институт проблем информатики Российской академии наук, NMorozov@ipiran.ru

в процессе анализа. Алгоритмическая полнота обеспечивается программой анализа и гарантирует проверку всех возможных промежуточных состояний при переключении схемы из заданного стабильного состояния в другое стабильное состояние, зависящее от начального состояния и комбинации входных сигналов. Тестовая полнота в общем случае обеспечивается перебором всех возможных стабильных начальных состояний схемы (значений входов схемы и выходов ее элементов) и заданием для каждого начального состояния всех возможных комбинаций входов.

Понятие тестовой полноты (полноты тестов) давно известно в синхронной схемотехнике. Оно характеризует качество тестовых воздействий, использующихся для проверки функционирования разработанной схемы. Стопроцентной полнотой обладают тесты, гарантирующие выявление возможной неисправности во всех элементах схемы на основе анализа поведения выходных сигналов в зависимости от заданных комбинаций ее входов. Тестовая полнота анализа на самосинхронность в событийных методах, рассматриваемая ниже, аналогична полноте тестов в синхронных схемах, но имеет свою специфику. Эта специфика связана с необходимостью использования замыкания схемы и позволяет в ряде случаев существенно сократить объем и время анализа (тестирования) за счет выбора соответствующего блока замыкания.

Данная работа посвящена рассмотрению процедуры анализа на самосинхронность и решению проблемы тестовой полноты такого анализа в событийных методах, использующих замыкание анализируемой схемы [1], для регистров сдвига и хранения.

2 Процедура анализа на самосинхронность

Одно из свойств СС-схемы заключается в том, что при любом заданном начальном состоянии, соответствующем одному из ее рабочих состояний, и при *правильно* реализованном замыкании она способна бесконечно переключаться между своими рабочими состояниями без нарушения самосинхронности при произвольных задержках переключения ее элементов. Это свойство проверяется в методах анализа схемы, использующих замыкание анализируемой схемы.

Разработчик, желающий проанализировать спроектированную схему на самосинхронность, должен решить: (1) в каком виде надо анализировать схему; (2) как обеспечить стопроцентную тестовую полноту анализа; (3) как найти и исправить нарушение самосинхронности в схеме.

2.1 Подготовка схемы для анализа

В общем случае анализируемая схема является частью какого-то цифрового устройства. Цель анализа на самосинхронность и итеративной отладки схемы — добиться того, чтобы она вела себя как СС-устройство в составе общей БИС.

Самосинхронность схемы обеспечивается организацией правильной дисциплины изменения входных, промежуточных и выходных сигналов, с одной стороны, и реализацией достоверной и полной индикации выходов элементов схемы, с другой стороны.

Все сигналы в СС-схеме разделяются на информационные, установочные, управляющие и индикаторные. Правильная дисциплина изменения согласованных информационных и управляющих входов предполагает [1], во-первых, строгое чередование рабочей и спейсерной фаз работы схемы и, во-вторых, инициирование перехода в следующую фазу работы только после завершения переключения всех возбужденных элементов схемы. Переход в очередную фазу работы инициируется парафазными информационными входами [1] и/или управляющими входами схемы. Окончание переключения всех элементов схемы фиксируется индикаторными выходами. Следовательно, изменения входов в процессе анализа должны быть привязаны к изменениям индикаторных выходов. А изменения бифазных и унарных входов, кроме того, должны быть согласованы с изменениями соответствующих управляющих входов.

В событийных методах анализа [2, 3] соблюдение таких зависимостей обеспечивается построением *блока замыкания*. На основе индикаторных выходов анализируемой схемы он реализует необходимое согласование ее входных сигналов и обеспечивает правильное чередование фаз ее работы. В результате анализируемая схема оказывается замкнутой, как показано на рис. 1. Здесь D_i, D_t, D_o — входные, промежуточные и выходные информационные сигналы; E_i и E_t — входные и промежуточные управляющие сигналы; I_o — выходной индикаторный сигнал.

Блок замыкания при анализе на самосинхронность служит функциональным аналогом схемы, реализующей запросно-ответное взаимодействие между анализируемой схемой и ее гипотетическим окружением. В разомкнутых схемах требуемые причинно-следственные зависимости между входными и выходными сигналами только предполагаются или описаны в спецификации схемы. Реализацию этих причинно-следственных зависимостей обеспечивает замыкание схемы либо в отдельности, либо в составе общей схемы. Построение блока замыкания в процессе анализа схемы на самосинхронность помогает разработчику понять, как строить запросно-ответное взаимодействие данной схемы с ее «соседями» в реальной БИС, и отладить это взаимодействие.

Функциональный метод анализа схемы на самосинхронность [4] работает с разомкнутой схемой, не имеющей блока замыкания.

2.2 Обеспечение стопроцентной тестовой полноты анализа

Задача анализа на самосинхронность — проверить, что все инициированные переключения элементов в схеме завершаются в текущей фазе до изменения значения соответствующего индикаторного выхода.

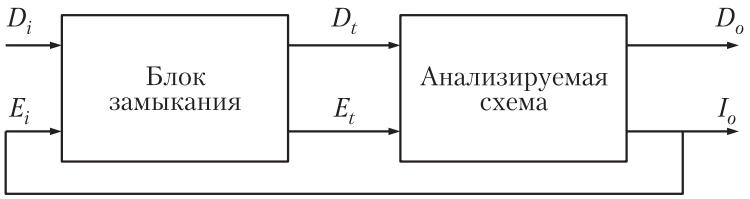


Рис. 1 Анализируемая схема с блоком замыкания

Назовем *стопроцентно полным*, с точки зрения тестовой полноты, анализ, подтверждающий сохранение самосинхронности анализируемой схемы при ее любом допустимом стабильном (равновесном [5, с. 51]) состоянии (произвольной комбинации входов и любом не противоречащем ей внутреннем состоянии), взятом в качестве начального состояния. Необходимые и достаточные условия достижения стопроцентной тестовой полноты анализа зависят от типа анализируемой схемы.

При использовании программных средств АСИАН [2] или АСПЕКТ [3] разработчик для обеспечения стопроцентной тестовой полноты анализа должен подключить к схеме блок замыкания, который при автономной (циклической) работе с ним анализируемой схемы гарантировал бы в общем случае прохождение всех значимых (с точки зрения тестовой полноты анализа) стабильных состояний схемы плюс переходы между ними. Построение блока замыкания требует от разработчика знания особенностей работы схемы и в общем случае является нетривиальной задачей, сложность которой пропорциональна сложности анализируемой схемы. Однако для ряда типовых цифровых устройств, например регистров сдвига и хранения, она решается достаточно просто.

В статье [5, с. 53–55] дается обоснование *достаточных* условий обеспечения полноты анализа на самосинхронность: схема может считаться полностью проанализированной, если в процессе анализа она пройдет по всем своим рабочим состояниям и при этом будет проверен переход из каждого рабочего состояния в любое другое достижимое из него рабочее состояние. Автор указывает, что это не является необходимым условием для подтверждения самосинхронности схемы. Действительно, для многих типовых цифровых устройств такой анализ избыточен. Рассмотрим конкретные примеры: регистр сдвига с последовательной записью и регистр хранения с параллельной записью.

Регистр сдвига имеет один информационный вход и большое количество внутренних состояний, которое для n-разрядного регистра можно оценить снизу по формуле $4 \cdot (2{,}6^{n-1})$, полученной эмпирическим путем. Но проверять все внутренние состояния в процессе анализа не требуется.

Пусть все разряды регистра сдвига имеют индикаторный выход (рис. 2). Каждый разряд функционирует в соответствии со своими входами управления:

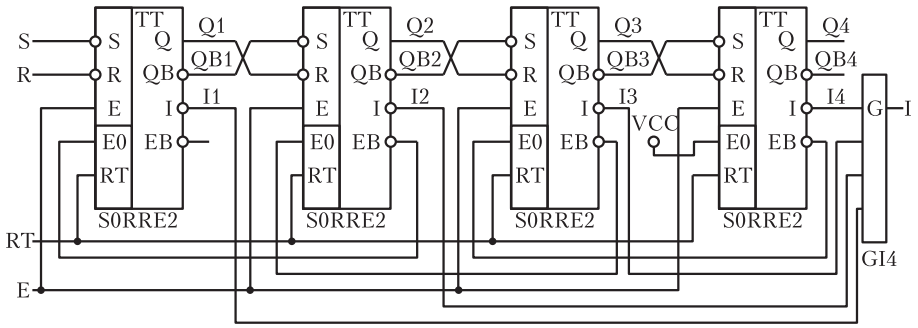


Рис. 2 Четырехразрядный регистр сдвига

общим E и поразрядным E0. Информационный вход $\{R, S\}$ формируется предшествующим разрядом, а для первого разряда — внешним устройством. Сигнал управления E0, наоборот, формируется последующим разрядом регистра сдвига. Состояние i-го разряда не оказывает никакого влияния на состояния разрядов с первого по $(i-2)$-й и с $(i+2)$-го по n-й. Для проверки самосинхронности поведения регистра сдвига на рис. 2 необходимо и достаточно сформировать такую последовательность входных сигналов, которая бы обеспечила запись в каждый разряд «0 после 0», «0 после 1», «1 после 0» и «1 после 1».

Такая последовательность представляет собой непрерывное чередование пар нулей и единиц $\{00110011\ldots\}$ на информационном входе первого разряда. Необходимая и достаточная длина ее равна $n+4$. Блок замыкания для такого регистра сдвига реализуется аппаратно с помощью двухразрядного двоичного СС-счетчика. Информационный вход регистра сдвига подключается к информационному выходу второго разряда счетчика. Индикаторный выход счетчика формирует общий сигнал управления для всех разрядов регистра. Счетный вход счетчика формируется индикаторным выходом регистра (I), собранным из поразрядных индикаторных выходов (сигналы I1–I4 на рис. 2) с помощью Г-триггера (GI4). В процессе циклической работы замкнутой схемы счетчик за $n+4$ циклов обеспечит формирование указанной выше последовательности пар нулей и единиц.

В общем случае n-разрядный *регистр памяти* с параллельной записью имеет n информационных входов (унарных, бифазных или парафазных). Пусть все разряды регистра памяти реализованы на одинаковых двухтактных триггерах с бифазным информационным входом и имеют индикаторный выход. Число внутренних состояний такого регистра будет равно 4^n. Но с точки зрения записи и хранения информации все разряды регистра памяти не зависят друг от друга. Поэтому необходимо и достаточно проверить самосинхронность регистра в режимах записи во все разряды одновременно «0 после 0», «0 после 1», «1

после 0» и «1 после 1». Это обеспечивается формированием последовательности $\{00110\}$ на информационных входах разрядов.

В качестве блока замыкания в данном случае достаточно использовать двухразрядный двоичный СС-счетчик, второй разряд которого и формирует информационные входы регистра. В процессе циклической работы замкнутой схемы счетчик за четыре цикла обеспечит формирование указанной выше последовательности значений информационных входов для всех разрядов регистра.

Рассмотренные примеры показывают, что для обеспечения стопроцентной тестовой полноты анализа на самосинхронность регистры сдвига и памяти, имеющие регулярную структуру, не требуют полного перебора всех возможных стабильных состояний. Для регистра сдвига необходимо и достаточно $n + 4$ циклов, где n — число его разрядов, а для регистра памяти — четырех циклов. При этом каждый разряд регистра должен иметь индикаторный выход, обеспечивающий стопроцентную индикацию внутренних элементов разряда, его входов и выходов.

Во всех рассмотренных типах цифровых устройств блок замыкания строится на основе двоичного СС-счетчика. Блок замыкания произвольной схемы может включать в себя несколько таких счетчиков разной разрядности, связанных или не связанных друг с другом какими-либо отношениями, для обеспечения последовательностей тестовых воздействий, подаваемых на разные части анализируемой схемы.

Достичь стопроцентной тестовой полноты анализа на самосинхронность для относительно простых схем нетрудно. Для одного из выбранных начальных рабочих состояний достаточно руководствоваться следующими принципами:

- формирование связанной группы из m информационных сигналов, являющихся входами одного функционального блока в составе анализируемой схемы, реализуется самосинхронным двоичным $(m + 1)$-разрядным счетчиком;
- бифазные информационные входы анализируемой схемы подключаются непосредственно к информационным выходам счетчика, начиная со второго разряда;
- индикаторный выход счетчика с помощью инвертора или повторителя формирует сигнал управления, сопровождающий бифазные информационные входы анализируемой схемы;
- если информационные входы анализируемой схемы — парафазные со спейсером, информационные выходы счетчика пропускаются через дополнительную подсхему преобразования бифазных сигналов в парафазные с требуемым типом спейсера; для этой подсхемы входом управления служит индикаторный выход счетчика, а ее собственный индикаторный выход используется для формирования счетного сигнала счетчика в блоке замыкания вместе с индикаторным выходом анализируемой схемы;

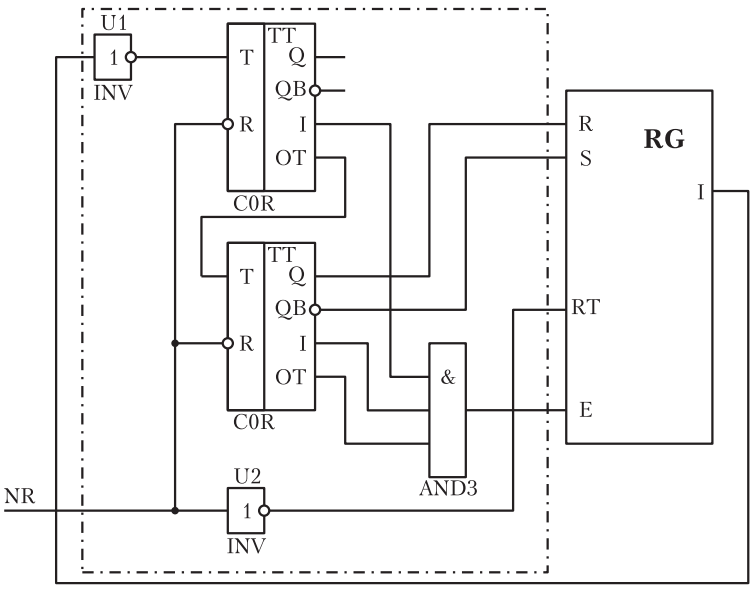


Рис. 3 Схема 4-разрядного регистра сдвига (RG) с блоком замыкания

- если у анализируемой схемы несколько индикаторных выходов, они объединяются Г-триггером, входящим в состав блока замыкания;
- если у анализируемой схемы несколько входов управления с одинаковым спейсером, связанных с одним подмножеством информационных сигналов, они подключаются к индикаторному выходу блока замыкания с помощью повторителей;
- повторить перечисленные шаги для других значимых и не покрытых рабочих состояний.

К относительно простым схемам относятся, в первую очередь, комбинационные схемы, а также схемы с небольшим числом входов и переменных памяти (не более 2–3). Однако сюда же можно отнести и многоразрядные регистры сдвига и памяти. На рис. 3 показана схема 4-разрядного регистра сдвига, подготовленная для анализа на самосинхронность. Сам регистр представлен функциональным блоком RG, реализация которого приведена на рис. 2. Обведенная штрихпунктирной линией часть схемы представляет собой блок замыкания. Двоичный двухразрядный счетчик построен на элементах C0R, элемент AND3 является индикатором счетчика. Инвертор U1 обеспечивает согласование фаз индикаторного выхода регистра и счетного входа счетчика. Инвертор U2 согласовывает активные уровни входов начального сброса счетчика и регистра. Вход несамосин-

хронного сброса схемы (NR) во время анализа задается неактивным статическим значением.

Разработка блока замыкания — чрезвычайно ответственная задача. Из-за неправильно построенного блока замыкания схема может не пройти анализ на самосинхронность, даже если потенциально и является самосинхронной. Чем сложнее анализируемая схема, чем меньше в ней регулярных устройств, тем труднее разработать для нее блок замыкания, который обеспечил бы стопроцентную тестовую полноту анализа на самосинхронность. В результате проверить на самосинхронность функционально законченный реальный блок или БИС целиком становится проблематичным.

Практическим решением этой задачи является использование метода иерархического анализа на самосинхронность. В настоящее время проблема иерархического анализа находится в стадии исследования. В работе [6] изложен один из возможных подходов для его реализации, правомерность и эффективность которого требуют более основательного анализа.

3 Исправление нарушений самосинхронности

Причиной появления нарушений самосинхронности в процессе анализа могут служить некорректно заданное начальное состояние, ошибки в организации запросно-ответного взаимодействия между составными частями анализируемой схемы, неправильно построенный блок замыкания. Для ее выявления рекомендуется следующий порядок действий.

1. Сделать замыкание тупиковым, например ввести лишний инвертор в цепь индикаторного выхода анализируемой схемы. В этом случае в процессе анализа схема должна перейти в стабильное состояние и остановиться в нем.
2. Ввести полученное стабильное состояние в задание на анализ в качестве начального состояния.
3. Исправить замыкание, сделав его из тупикового регенеративным.
4. Запустить анализ на самосинхронность. Появление нарушений самосинхронности в этом случае будет означать наличие каких-то реальных проблем в самой схеме или блоке замыкания.
5. Найти в схеме элемент, выход которого не успел переключиться в противоположное состояние во время текущей фазы работы и тем самым вызвал нарушение самосинхронности (фактически это означает, что выход данного элемента оказался *неиндицируемым* при заданном режиме работы анализируемой схемы).
6. Определить причину, по которой значение выхода данного элемента оказалось не играющим роли при формировании индикаторного выхода схемы:

- выход элемента не индицируется (не участвует в формировании индикаторного выхода схемы);
- рассогласование фаз информационных и/или управляющих сигналов и т. д.

7. Устранить найденную причину, исправить при необходимости начальное состояние и вернуться к п. 4.

Отладка схемы при анализе на самосинхронность — итеративный процесс, сопровождающийся изменением начального состояния схемы, ее реализации, реализации блока замыкания. Как показывает практика, наиболее существенно при этом будет меняться схема управления запросно-ответным взаимодействием функционально законченных фрагментов схемы, в том числе подсхема индикации.

4 Заключение

Стопроцентная тестовая полнота анализа на самосинхронность как практическая категория, гарантирующая самосинхронность схемы при любых реальных обстоятельствах и правильно организованной дисциплине формирования информационных и управляющих сигналов и взаимодействия с окружением, чрезвычайно важна. Она обеспечивает безошибочность проектирования действительно самосинхронной схемы.

С практической точки зрения, вопрос о стопроцентной тестовой полноте анализа должен решаться для каждого типа разрабатываемой схемы индивидуально. Приведенные примеры показывают, что в зависимости от функциональной особенности состава и реализации схемы стопроцентная тестовая полнота анализа требует разного количества циклов работы замкнутой схемы, но технически достижима относительно несложным блоком замыкания.

Для сложных многоразрядных цифровых устройств стопроцентную тестовую полноту анализа целесообразно обеспечивать за счет использования иерархического подхода к анализу на самосинхронность.

Литература

1. Автоматное управление асинхронными процессами в ЭВМ и дискретных системах / Под ред. В. И. Варшавского. — М.: Наука, 1986. 400 с.
2. *Рождественский Ю. В., Морозов Н. В., Степченков Ю. А., Рождественскене А. В.* Универсальная подсистема анализа самосинхронных схем // Ежегодник трудов ИПИ РАН «Системы и средства информатики». — М.: Наука, 2006. Вып. 16. С. 463–475. (URL: http://samosinhron.ru/native/full/54.shtml, дата обращения 30.04.2011.)

3. *Рождественский Ю. В., Морозов Н. В., Рождественскене А. В.* Подсистема событийного анализа самосинхронных схем АСПЕКТ // Проблемы разработки перспективных микро- и наноэлектронных систем-2010: Сборник науч. тр. / Под общ. ред. А. Л. Стемпковского. — М.: ИППМ РАН, 2010. С. 418–423. (URL: http://www.mes-conference.ru/data/year2010/papers/m10-145-69491.pdf, дата обращения 30.04.2011.)
4. *Плеханов Л. П.* Реализация функционального метода анализа самосинхронности электронных схем // Системы и средства информатики. Вып. 19. — М.: Наука, 2009. С. 142–148.
5. *Плеханов Л. П.* Полнота анализа электронных схем на самосинхронность // Системы и средства информатики / Под ред. И. А. Соколова. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 48–58.
6. *Степченков Ю. А., Дьяченко Ю. Г., Рождественский Ю. В., Морозов Н. В., Степченков Д. Ю.* Разработка вычислителя, не зависящего от задержек элементов // Системы и средства информатики / Под ред. И. А. Соколова. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 5–23.

О СВОЙСТВЕ САМОСИНХРОННОСТИ ЦИФРОВЫХ ЭЛЕКТРОННЫХ СХЕМ*

Л. П. Плеханов[1]

Аннотация: Обсуждаются понятие самосинхронности с практической точки зрения и его связь с классическим определением независимости от задержек. Показано, что одного свойства независимости от задержек недостаточно для самосинхронности. Приводятся практические следствия теоретических положений при разработке самосинхронных (СС) схем. Дается связь самосинхронности с недавно вступившим в действие новым стандартом по надежности.

Ключевые слова: самосинхронность; самосинхронные схемы; анализ самосинхронности

1 Введение

Самосинхронные схемы являются важной частью класса асинхронных цифровых схем, в них принцип асинхронности доведен до своего возможного предела — схемы «сами себя синхронизируют».

Самосинхронные схемы обладают уникальными свойствами (подробнее — далее в статье), о которых неоднократно заявлялось в теоретическом плане [1–4] и которые впервые подтверждены рядом прямых экспериментов [5].

В основополагающей работе [1] доказано существование схем, поведение которых не зависит от величин задержек элементов, и выявлены их основные детали поведения. Схемы получили название *speed-independent* (SI). Данный термин и схожие с ним широко используются в зарубежной литературе.

С практической точки зрения, при разработке и применении обсуждаемых схем понятие SI необходимо, но не достаточно.

Во-первых, независимость от задержек формально определена для одного начального состояния элементов схемы и неизменных значений входов. Для реальных схем требуется независимость для всех реальных состояний и по всем реальным переходам, связанным с изменениями входов, что порождает проблему полноты, обсуждаемую ниже.

Во-вторых, свойство SI не обеспечивает в полной мере диагностических свойств схем, что следует из [3] и показано ниже.

*Работа выполнена при частичной финансовой поддержке по Программе фундаментальных исследований ОНИТ РАН на 2011 г. (проект 1.5).

[1]Институт проблем информатики Российской академии наук, LPlekhanov@ipiran.ru

В третьих, SI-схемы могут быть разомкнутыми и замкнутыми, и эти разновидности имеют разные свойства при практическом использовании.

Термин «самосинхронные схемы» введен в книге В. И. Варшавского с соавторами [3] (в оригинале — «самосинхронизирующиеся» или «апериодические», но в последующем сами авторы перешли на термин «самосинхронные»). Понятие самосинхронности включает в себя, помимо SI, диагностические свойства схем. По мнению авторов, «решающим достоинством апериодических схем являются их самодиагностические свойства».

В статье обсуждается, какие практические следствия и свойства вытекают из теоретических определений, а также условия выполнения самосинхронности, в том числе в новой стандартной терминологии.

2 Потребительские свойства самосинхронных схем

В связи с вводом в действие нового стандарта [6], в котором прежние определения были изменены, термины из стандарта в этом пункте выделены жирным шрифтом.

Одна из главных проблем работы цифровых электронных схем — возникновение **ошибок** на выходах элементов и схем. Ошибки могут порождаться внешними воздействиями и внутренними причинами.

Внешние воздействия — это электромагнитные наводки, удары энергичных элементарных частиц и другие кратковременные явления. Будем отличать их от долговременных факторов — условий работы схем (главные из которых — радиационный фон, температура и напряжение питания). Вопрос защиты от внешних воздействий составляет отдельную проблему и здесь не рассматривается.

Внутренние причины ошибок обусловлены способом построения схемы и условиями ее работы. Таких причин две:

1. Состязания сигналов (гонки) на входах элементов, т. е. такие изменения этих входов, которые вызывают ошибочные изменения их выходов. Эта причина порождается недостатками построения схем.

2. Возникновение **отказов** схемы. Отказы происходят при потере физической работоспособности внутренних структур, вызванной условиями работы: температурой, напряжением питания, а также старением и другими подобными процессами.

Принципиально важным является вопрос: можно ли создать схему, не имеющую ошибок от приведенных причин?

В существующих подходах построения синхронных и асинхронных (помимо самосинхронного) схем гонки не могут быть устранены полностью. Можно устранить их в каком-либо блоке, но в окружающих его схемах надо обеспечить

дисциплину сигналов для этого блока. Окружающие схемы, однако, для ликвидации гонок тоже требуют обеспечения дисциплины в своих окружениях, и т. д. Задача оказывается нереальной.

Предвидеть отказы на этапе проектирования и предотвратить их на этапе функционирования невозможно. Однако ошибок не будет, если сразу при возникновении отказа схема остановится и не выдаст на выходах ошибочных значений. Наиболее близкий стандартный термин, отражающий данное свойство, — **отказобезопасность**.

В книге [3] доказано, что самосинхронные схемы имеют уникальные свойства безошибочной работы: отсутствие гонок и отказобезопасность.

Рассмотрим условия выполнения этих свойств.

Отсутствие гонок обеспечивается независимостью от задержек элементов [1], т. е. свойством SI. Условиями применимости этой теории служат требования, известные как *модель Маллера*. Согласно этой модели под элементом понимается устройство, имеющее один выход и описываемое одним логическим уравнением. Задержки элементов приведены к их выходам, а задержки межсоединений (проводов) после разветвлений пренебрежимо малы по сравнению с задержками элементов. Задержка считается конечной любой величины.

Под отказобезопасностью СС-схем понимается их остановка, вызванная константными **неисправностями** типа залипания на 0 и 1 (КНЗ-01), одиночными и кратными, на выходах элементов. Эти неисправности характеризуются состоянием постоянного значения 0 или 1 на выходах одного или нескольких элементов. В технической диагностике цифровых схем они считаются наиболее часто встречающимися и практически наиболее значимыми.

Помимо того, что отказобезопасность важна сама по себе (отсутствие непредусмотренных значений на выходах, во многих случаях опасных), она дает возможность **самотестирования**, т. е. построения схем, оценивающих свое состояние во время работы, **диагностирования неисправностей** и в конечном итоге получения надежных схем.

Далее, согласно работам [2, 3] будем рассматривать схемы, построенные по принципу двухфазной работы — чередования рабочей фазы и промежуточной (спейсера). Информация в таких схемах кодируется самосинхронными кодами, здесь для простоты берется наиболее простой и используемый — парафазный код со спейсером (ПФС). В этом коде один бит исходной информации кодируется двумя битами кода: 01 и 10, а в фазе спейсера оба бита кода одинаковы: 00 или 11.

3 Разомкнутые самосинхронные схемы

Входные и выходные сигналы разомкнутой схемы должны иметь определенный тип, связанный как с информацией, так и с фазой работы. Типы

сигналов можно разделить на три группы: информационные, контрольные и вспомогательные.

К информационным сигналам относятся парафазные со спейсером (*ПФС-сигналы*) и бистабильные — выходы бистабильных ячеек (*БС-сигналы*). Контрольные сигналы предназначены для организации переходов из одной фазы в другую. К ним относятся управляющие на входе (*У-сигналы*) и индикаторные на выходе (*И-сигналы*): У-сигналы инициируют переключения из одной фазы в другую, И-сигналы показывают завершение перехода схемы в текущую фазу.

Контрольные и ПФС-сигналы специфичны для фазы работы и вместе называются *фазовыми* сигналами. И на входе, и на выходе схемы должно присутствовать хотя бы по одному фазовому сигналу.

Вспомогательные сигналы предназначены для вспомогательных целей (например, предустановки) и далее учитываться не будут.

Описанный интерфейс будем называть *типовым фазовым* интерфейсом разомкнутой схемы. Для разомкнутых схем важнейшим свойством, введенным в [3], является *индицируемость*. Под индицируемостью какого-либо сигнала — выхода элемента — понимается способность схемы реагировать на изменение этого сигнала изменением фазовых выходов (хотя бы одного). Под *полной индицируемостью* схемы понимается индицируемость всех элементов при всех реальных переходах между состояниями схемы. (Реальные переходы должны быть допустимыми, что обеспечивается структурой интерфейса и дисциплиной изменения входов.)

Индицируемость сигнала можно определить в любой фазе, сравнивая значения фазовых выходов в двух вариантах: при нормальном изменении сигнала и при его неизменности (залипании).

В работе [3] ряд необходимых здесь результатов сформулирован для конечных автоматов и их моделей. Эти результаты справедливы также и для реализаций автоматов — цифровых схем. В терминах данной статьи можно сформулировать следующий результат из [3, п. 4.8] в применении к разомкнутым схемам:

Разомкнутая схема, имеющая типовой фазовый интерфейс, будет самосинхронной, если в каждой фазе (рабочей и спейсерной) обеспечиваются два условия:

(1) *отсутствие гонок на всех элементах при любых задержках элементов;*

(2) *полная индицируемость схемы*.

Отсутствие гонок обеспечивается независимостью от задержек по Малеру [1]. Обнаружить гонки в разомкнутой схеме можно различными путями. Один из способов — воспользоваться событийным методом анализа полумодулярности. В данном случае разомкнутая схема по Маллеру — тупиковая, и нет необходимости работы с контрольным слоем, что существенно сокращает вычислительные затраты по сравнению с анализом замкнутых схем. При этом должна быть обеспечена полнота переходов между состояниями.

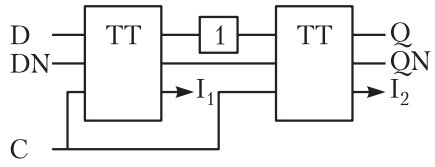


Рис. 1 Разомкнутая схема, не зависящая от задержек элементов: TT — двухступенчатые триггеры; D, DN — бистабильные входы; Q, QN — бистабильные выходы; I_1, I_2 — индикаторные сигналы; C — управляющий сигнал

Есть также способы нахождения гонок и в функциональном подходе (по описаниям логических функций), применимые к двухфазному принципу и самосинхронному кодированию информации.

При возникновении КНЗ-01 на выходе какого-либо индицируемого элемента хотя бы один выходной фазовый сигнал схемы не перейдет в значение, соответствующее текущей фазе, что обнаружится во внешнем окружении. Таким образом, индицируемость схемы обеспечивает ее отказобезопасность.

Как видно, для разомкнутых схем одного SI-свойства — независимости от задержек — недостаточно для самосинхронности.

Например, разомкнутая схема на рис. 1 является независимой от задержек. Однако в спейсерной фазе при некоторых значениях входов и внутренних состояний триггеров выход повторителя не индицируется на фазовом сигнале I_2, и схема в результате не самосинхронна.

4 Замкнутые самосинхронные схемы

Независимость от задержек элементов (SI-свойство) замкнутых схем определяется событийными методами на основе математического свойства полумодулярности диаграмм переходов состояний [1]. Полумодулярность проверяется по конфликтности перехода из одного состояния в другое, т. е., по существу, локальной проверке наличия гонки при каждом переходе.

При практической разработке замкнутые схемы получают из разомкнутых (с типовым интерфейсом) путем корректного замыкания. Процедура замыкания имеет целью имитацию внешней среды для схемы, в частности имитацию взаимодействия с окружающей схемой.

Опуская подробности, корректное замыкание сводится к двум основным действиям:

1. Выходные фазовые сигналы сводятся к одному индикаторному сигналу по известным правилам с помощью Г-триггеров [3] — специальных триггерных устройств, предназначенных для уменьшения числа индикаторных сигналов. Этот общий индикаторный сигнал подается на вход как сигнал обратной связи.
2. Сигнал обратной связи инициирует изменение входов схемы. При этом БС-сигналы и связанные с ними У-сигналы должны меняться в определенном

порядке, имитирующем их изменение во внешней схеме. В остальном изменения входов должны быть независимы.

При проверке SI-свойства таких схем выявляются не только гонки на элементах, но и неиндицируемые элементы исходной разомкнутой схемы.

Предположим, что некоторый сигнал (выход элемента) не индицируется на фазовых выходах и, в конечном итоге, на общем индикаторном сигнале схемы. Поскольку задержки элементов могут быть любыми, пусть неиндицируемый элемент не успел переключиться в значение, соответствующее текущей фазе, т. е. остался возбужденным. Так как выходной сигнал элемента не индицируется, то общий индикатор «не заметит» отсутствия переключения и по обратной связи инициирует переход в другую фазу. В течение следующей фазы возбуждение элемента будет снято, что означает конфликт и нарушение полумодулярности.

Таким образом, отсутствие индицируемости элемента в разомкнутой схеме ведет себя как наличие гонки в замкнутой схеме.

Однако одного свойства полумодулярности (SI-свойства) замкнутых схем также недостаточно для самосинхронности. Соответствующее утверждение из [3, п. 10.3] в терминах данной статьи записывается так:

Замкнутая схема будет самосинхронной при выполнении двух условий:

(1) *если она полумодулярна по Маллеру;*

(2) *если ее диаграмма переходов не содержит фиктивных классов эквивалентности.*

Фиктивный класс эквивалентности (ФКЭ) — это замкнутый цикл в диаграмме переходов, в котором хотя бы одна переменная всегда возбуждена и не меняется [3].

Присутствие ФКЭ в диаграмме переходов обусловлено наличием в схеме автогенерирующих подсхем [3]. Обнаружить эти классы можно различными путями.

Если замкнутая схема получена из разомкнутой, то следует проверить исходную разомкнутую схему на устойчивость: при всех рабочих входных наборах и значениях внутренних состояний схема должна устанавливаться в устойчивое состояние.

При анализе замкнутых схем для обнаружения ФКЭ необходимо проверять все рабочие финальные циклы диаграммы переходов на наличие переменных, находящихся в возбужденном состоянии и не меняющихся.

Наконец, можно анализировать диаграммы переходов схемы при задании конкретных КНЗ-01. В этих случаях процесс переходов должен заканчиваться тупиковыми состояниями.

5 Полнота по состояниям и переходам

Проблема полноты соблюдения свойства самосинхронности возникает при необходимости учета начальных состояний элементов и реальных переходов между состояниями [7]. Требование полноты заложено в определениях самосинхронности [3] в виде условия соблюдения этого свойства при всех допустимых переходах.

Расчет состояний схемы следует проводить при разомкнутой общей обратной связи (в нормальном режиме работы замкнутые СС-схемы не имеют устойчивых состояний).

Начальные состояния схемы устанавливаются при включении ее питания. Они бывают *равновесными* и *неравновесными*. Равновесные состояния сохраняются неограниченно долго при неизменных входных сигналах. Неравновесные состояния без внешних воздействий со временем переходят в равновесные. Например, если вход и выход инвертора имеют одинаковые значения, то это состояние неравновесно. Период установления схемы в стабильное состояние после включения питания обычно не считается рабочим, и его следует исключить из рассмотрения.

Любое равновесное состояние разомкнутой схемы определяется значениями входных сигналов и переменных памяти (сигналов запоминающих ячеек в схеме). Равновесное состояние можно определить, задавая входной набор и вычисляя значения внутренних сигналов. В процессе такого вычисления некоторые сигналы (переменные памяти) могут оставаться произвольными, например один из выходов триггера. В общем случае переменные памяти зависят от значений входных сигналов, и они должны участвовать в общем переборе.

Все возможные равновесные состояния, вычисленные описанным способом, будем называть *основными*. В разных фазах эти состояния будут разными.

В процессе реальной работы схема переходит из одного основного состояния в другое через ряд промежуточных неравновесных состояний. Очевидно, что свойство самосинхронности должно соблюдаться при всех этих переходах.

Необходимым условием самосинхронности будет полнота по состояниям (СС-полнота) — выполнение этого свойства при прохождении всех основных состояний схемы и всех реальных переходах между основными состояниями.

6 Заключение

С точки зрения потребительских свойств СС-схемы обеспечивают безошибочность своей работы:

- отсутствие логических состязаний (гонок) элементов;
- отказобезопасность — остановку схемы при возникновении выходных константных неисправностей элементов типа залипания на 0 и 1, одиночных и кратных.

Существующее понятие независимости от задержек элементов (speed-independent), широко применяемое в зарубежной литературе как базовое, является необходимым, но не достаточным для самосинхронности.

Обеспечение самосинхронности требует дополнительных вычислительных затрат при проектировании по сравнению с обеспечением независимости от задержек.

Литература

1. *Muller D. E., Bartky W. C.* A theory of asynchronous circuits // Symposium (International) on the Theory of Switching Proceedings. Part 1. — Harvard University Press, 1959. P. 204–243.
2. Апериодические автоматы / Под ред. В. И. Варшавского. — М.: Наука, 1976. 423 с.
3. Автоматное управление асинхронными процессами в ЭВМ и дискретных системах / Под ред. В. И. Варшавского. — М.: Наука, 1986. 400 с.
4. *Филин А. В., Степченков Ю. А.* Компьютеры без синхронизации // Системы и средства информатики. — М.: Наука, 1999. Вып. 9. С. 247–261.
5. *Плеханов Л. П., Степченков Ю. А.* Экспериментальная проверка некоторых свойств строго самосинхронных схем // Системы и средства информатики. Вып. 16. — М.: Наука, 2006. С. 476–485. (http://elibrary.ru/item.asp?id=13060494, дата обращения 12.04.2011).
6. ГОСТ Р 53480-2009. Надежность в технике. Термины и определения. — М.: Стандартинформ, 2010. (http://www.vsegost.com/Catalog/49/49170.shtml.)
7. *Плеханов Л. П.* Полнота анализа электронных схем на самосинхронность // Системы и средства информатики / Под ред. И. А. Соколова. — М.: ТОРУС ПРЕСС, 2010. Вып. 20. № 1. С. 48–58.

ОСОБЕННОСТИ КЛАССИФИКАЦИОННОГО АНАЛИЗА САМОСИНХРОННЫХ СХЕМ*

***Ю. В. Рождественский*[1], *Н. В. Морозов*[2], *А. В. Рождественскене*[3]**

Аннотация: Предметом статьи является исследование методов анализа асинхронных схем на независимость их поведения от задержек логических элементов. Предлагаемый метод в теоретической части базируется на диаграммах переходов (метод в глобальных моделях) с последующим тождественным преобразованием к событийным моделям. Полученные алгоритмы анализа обладают строгой фундаментальностью метода в глобальных моделях, но не требует полного перебора достижимых состояний схемы. Сложность задачи изменилась с экспоненциальной на полиномиальную. Классификационный анализ уточняет свойства исследуемой схемы, представляет развернутую диагностику и определяет возможные причины возникших нарушений.

Ключевые слова: самосинхронные схемы; событийный анализ; автоматизированное проектирование

1 Введение

Существующие системы автоматизированного проектирования самосинхронных (не зависящих от задержек) интегральных схем базируются на разработанных ранее библиотечных элементах вентильного уровня. Анализ функционирования таких схем и проверка их самосинхронности производятся с помощью программных средств, базирующихся на «*глобальных моделях*» — АСИАН [1] и ТРАНАЛ [2]. Эти программные средства обеспечивают точный и корректный анализ небольших самосинхронных элементов, но неприменимы к более крупным схемным решениям вследствие экспоненциального роста вычислительных затрат, зависящих от количества внутренних параллельных переключений. Проектирование крупных блоков непосредственно из библиотечных элементов вентильного уровня приводит к получению громоздких и неэффективных решений, часто дискредитируя саму идею самосинхронизации.

*Работа выполнена при частичной финансовой поддержке по Программе фундаментальных исследований ОНИТ РАН на 2011 г. (проект 1.5).

[1]Институт проблем информатики Российской академии наук, YRogdest@ipiran.ru

[2]Институт проблем информатики Российской академии наук, NMorozov@ipiran.ru

[3]Институт проблем информатики Российской академии наук, ARogdest@ipiran.ru

Выходом из данного «тупика» могут быть событийные модели функционирования самосинхронных схем, обладающие полиномиальной сложностью к внутреннему параллелизму. Основная проблема в реализации таких программ заключается в обеспечении строгого тождественного соответствия фундаментальным методам анализа на «глобальных моделях». Она имеет теоретическую и техническую составляющие.

Основы теоретического подхода алгоритмизации данной задачи были проработаны группой В. И. Варшавского [3].

Детализация ряда теоретических положений и техническая реализация оставались до последнего времени невыполненными. Решению этой задачи посвящена данная статья.

2 Теоретические основы анализа самосинхронных схем на базе гипотезы Маллера

Классическое определение самосинхронных схем было дано в работах Маллера [4]. Для четкого определения круга решаемых задач приведем еще раз описание схемы по Маллеру.

Схемой называется совокупность логических элементов $\{Z_1, \ldots, Z_n\}$, где каждый вход логического элемента присоединен к одному выходу и никакие два выхода не соединены между собой. Состояние схемы в каждый момент представляет собой набор значений сигналов в ее узлах — выходах двоичных логических элементов. Входы схемы также можно считать логическими элементами без входов — генераторами нулей или единиц.

Состояние выхода элемента определяется значениями сигналов на его входах. Поведение i-го элемента схемы можно описать булевым уравнением:

$$Z_i' = F_i(Z_1, \ldots, Z_i, \ldots Z_n)\,, \tag{1}$$

где $Z_1, \ldots, Z_{i-1}, Z_{i+1}, \ldots, Z_n$ — значения сигналов во входных узлах i-го элемента в момент времени t; Z_i — значение выхода i-го элемента в момент времени t; Z_i' — значение, которое должен принять выход i в следующий момент времени t'; F_i — собственная функция i-го элемента.

Для схемы, состоящей из n элементов, моделью Маллера называется система из n уравнений вида Z_i', $i = 1, \ldots, n$.

Сказанное означает, что переключение любого логического элемента может происходить в течение любого, но ограниченного интервала времени. При этом результат переключения, появляющийся на выходе элемента Z_j, одновременно появляется на входах всех элементов Z_i, связанных с выходом элемента Z_j. (Все задержки переключения «приводятся» к входу сработавшего элемента.)

По Маллеру, схема будет *самосинхронной*, если в процессе ее работы ни для каких переключающихся элементов не возникнет ситуация, когда два различных

события на входах элемента, действуя в противоположных направлениях, могут переключить его в прямом и обратном направлении одновременно (состояние «гонки»).

Самосинхронные схемы по Маллеру (speed-independent) обладают рядом ценных качеств. Они характеризуются независимостью функционирования от времени переключения элементов схемы.

Предложенная модель допускает и более широкое применение. Понятие самосинхронности по Маллеру может быть распространено и на специальный класс схем, независимых как от времени переключения логических элементов, так и от задержек в соединениях элементов (delay-insensitive).

Действительно, если на всех входах каждого логического элемента поставить повторитель, то логика функционирования схемы останется неизменной, а времена появления входных сигналов на входах окажутся произвольными, ограниченными величинами, отсчитывающимися от момента срабатывания переключающих их выходов. Система логических уравнений, описывающая исходную схему (1), модифицируется к виду:

$$Z_i' = F_i(a_{ij})\,, \quad a_{ij}' = Z_i \quad \forall j\,,$$

где a_{ij} — логическая функция повторителя для j-го входа i-го логического элемента схемы.

Самосинхронность этой схемы по Маллеру будет означать независимость ее функционирования во времени как от задержек логических элементов, так и от задержек в соединениях.

Следовательно, средства анализа самосинхронности схемы Маллера позволяют анализировать оба вида времянезависимых схем и обладают, с этой точки зрения, существенной фундаментальностью.

Универсальные алгоритмы на базе «глобальных моделей» представлены в работе [3] и позволяют получить однозначный ответ о наличии нарушений самосинхронности в схеме Маллера. Своей универсальностью они обязаны полному просмотру всех допустимых и достижимых состояний схемы с целью проверки наличия таких нарушений. В этом их достоинство, но и главный недостаток. Время анализа схем с помощью универсальных алгоритмов определяется экспоненциальным характером зависимости числа анализируемых состояний от «внутренней параллельности» схемы, т. е. числа одновременно происходящих переключений логических элементов. В [1] описана реализация алгоритма проверки самосинхронности на базе *диаграмм переходов* (ДП), позволяющая максимально использовать практически все возможности современной вычислительной техники для решения этой задачи. Достигнутый уровень сложности исследуемых схем составляет не более 24 параллельных процессов. Это соответствует схемным решениям небольших электронных узлов по 500–1000 вентилей [5]. Анализировать более сложные схемы по этим алгоритмам не представляется возможным.

3 Событийный подход к анализу самосинхронных схем

В качестве альтернативы группой проф. В. И. Варшавского был предложен событийный подход к анализу самосинхронных схем. В этом случае под событием понимается изменение (срабатывание) одной из переменных в левой части уравнения (1). Предшествующее срабатыванию состояние переменной левой части, соответствующее временно́му интервалу, когда все переменные правой части завершили свое срабатывание, называется состоянием возбуждения переменной левой части логического уравнения. Оно демонстрирует приведение задержки срабатывания к выходу логического элемента для схемы Маллера. Множество возбужденных состояний переменной образует ее зону возбуждения в диаграмме переходов Маллера. Именно величина этой зоны и определяет характер экспоненциальной зависимости анализа в ДП, поскольку эта зона состоит из полного перебора состояний схемы по независимым (параллельным) переменным, отличным от возбужденной переменной. В событийном подходе вся зона возбуждения заменяется набором событий, равным числу независимых выходов из этой зоны. Для событий типа «И» это единственное событие — переключение элемента. Для события «ИЛИ» это число событий, равное числу независимых переменных, вызывающих переключение данного элемента.

Следовательно, переход от анализа на всем множестве допустимых состояний ДП Маллера к событийному анализу позволяет убрать экспоненциальную сложность анализа и сделать возможной проверку самосинхронности схемы Маллера размерностью до 10^4–10^5 логических уравнений. Это практически полностью покрывает потребности программных средств для анализа крупных блоков вычислительных систем. Такие блоки уже имеют шинную организацию внешних данных. Анализ самосинхронности устройств, содержащих эти блоки, сводится к анализу схемы управления их взаимодействием и полностью обеспечивается средствами событийного анализа.

Варшавским и его группой была доказана фундаментальность *диаграмм изменений* (ДИ), описывающих поведение событийной модели, и сформулированы основные принципы эквивалентности ДИ и ДП [2]. Базируясь на ДП, он доказал, что «*корректная*» (*полумодулярная*) ДИ тождественно описывает переключения самосинхронной схемы Маллера.

Диаграмму изменений можно представить как направленный граф специального вида. Узлами этого графа являются события, соответствующие переключениям логических переменных схемы, характеризующиеся знаком переключения, именем логической переменной и номером переключения от инициального запуска. Дуги, соединяющие события в графе, представлены двумя типами. Дуги «*строгого предшествования*» являются дугами направленного графа и соединяют события, для которых последующее событие невозможно без предыдущего (они описывают события типа «И»). Дуги «*слабого предшествования*», соединяющие

событие-инициатор с событием-последствием, демонстрируют способность, но не обязательность срабатывания последствия от данного инициатора (но обязательное срабатывание последствия от всех инициаторов). Они описывают события типа «ИЛИ» и могут нарушать общую направленность графа.

Корректная (полумодулярная) ДИ для процесса переключений системы уравнений должна удовлетворять условиям:

- построение ДИ начинается с некоторого инициального состояния послойно;
- в каждый новый слой включаются только события, имеющие непосредственных предшественников в предыдущем слое;
- события одного слоя не связаны между собой отношениями строгого предшествования (дугами первого типа);
- события нового слоя бесконфликтны (отсутствуют входящие дуги, требующие противоположных переключений события);
- построение нового слоя производится только для уже построенного корректного фрагмента ДИ;
- построение ДИ для разомкнутых схем заканчивается, если реализованы все допустимые события; для замкнутых схем — если обнаруживается точное совпадение циклических фрагментов ДИ.

Задача построения «корректной» ДИ для *дистрибутивных* схем (подмножества полумодулярных схем, не содержащих событий «ИЛИ») была полностью решена группой Варшавского практически (система ТРАСПЕК). Корректная ДИ для дистрибутивных схем представляет собой направленный граф, однозначно устанавливающий причинно-следственную связь между конкретными событиями (переключениями элементов схемы Малера).

Варшавским были сформулированы условия и общие предложения по построению корректной ДИ для полумодулярных (не дистрибутивных) схем, содержащих оба типа событий («И» и «ИЛИ»). Диаграмма изменений в этом случае представляет собой некоторое расширение класса направленных графов, включающее в себя разновидность понятия одновременности — слабое предшествование, что не позволяет использовать обычные графовые модели для анализа полумодулярных схем [3].

Основные проблемы построения корректной ДИ для полумодулярных схем связаны с «размазыванием» события переключения элемента «ИЛИ» по слоям и требованием «корректности» уже построенного фрагмента ДИ при построении очередного слоя. Это приводит к многократным перестроениям уже сформированного фрагмента ДИ при обнаружении новых событий «ИЛИ». Дополнительно серьезные проблемы создают случаи «*вырождения*» событий «ИЛИ» и «*перехвата термов*» [6], когда реальное событие «ИЛИ» в силу определенных взаимодействий его входных переменных между собой или с его выходом переключается как событие «И».

Если в процессе анализа для схемы была построена корректная ДИ, соответствующая модели Малера, то это является гарантией независимости ее функционирования от задержек элементов. Сама ДИ содержит полное описание всех процессов переключений и представляет собой чрезвычайно удобный инструмент для определения и последующей оптимизации временны́х и энергетических характеристик схемы. Если в процессе построения ДИ обнаруживаются конфликты, то дальнейший анализ прекращается и требуются изменения в реализации логики функционирования исследуемой схемы.

4 Виды нарушений самосинхронности и способы их обнаружения

В работе [3] было доказано, что все возможные виды конфликтов реализуются в одном слое ДИ. Это важное замечание существенно упрощает процедуру выявления всех возможных конфликтных ситуаций.

Теоретически это означает, что для всех возбужденных в предыдущем слое переменных может быть только одно значение сработавшей переменной в текущем слое. Появление двух устойчивых противоположных состояний и есть проявление конфликта, ведущего к нарушению полумодулярности схемы.

Для методов анализа в «глобальных моделях» (на базе ДП) эта проверка сводится к выполнению двух условий:

(1) отсутствие одинаковых переменных в списках сработавших и вновь возбужденных переменных для каждого нового слоя состояний схемы;

(2) отсутствие одинаковых переменных в списках, сработавших в более ранних слоях, но не вызвавших переключений и вновь возбужденных переменных для каждого нового слоя состояний схемы.

В случае применения событийного анализа появляется дополнительное требование, связанное с использованием определенной модели зон возбуждения переменных схемы в представлении событий в ДИ. Эта модель требует введения некоторых фиктивных переменных для описания событий, в которых нарушается свойство дистрибутивности. Число таких переменных в каждом случае нарушения дистрибутивности равно числу вновь возникающих в этих случаях параллельных независимых между собой последовательностей событий, что в точности соответствует структуре зоны возбуждения для переменной типа «ИЛИ» в «глобальных моделях». Это соответствие и определяет адекватность обоих методов анализа.

Отсюда возникает дополнительное условие для ДИ: *совпадение знаков переменной, порождающей фиктивное событие, и самого фиктивного события, встречающихся в одном слое для любого слоя исследуемой схемы*. Наличие различий в знаках будет говорить об одновременном присутствии противоположных устойчивых состояний фиктивной переменной в одном слое, что соответствует нарушению полумодулярности для недистрибутивных схем.

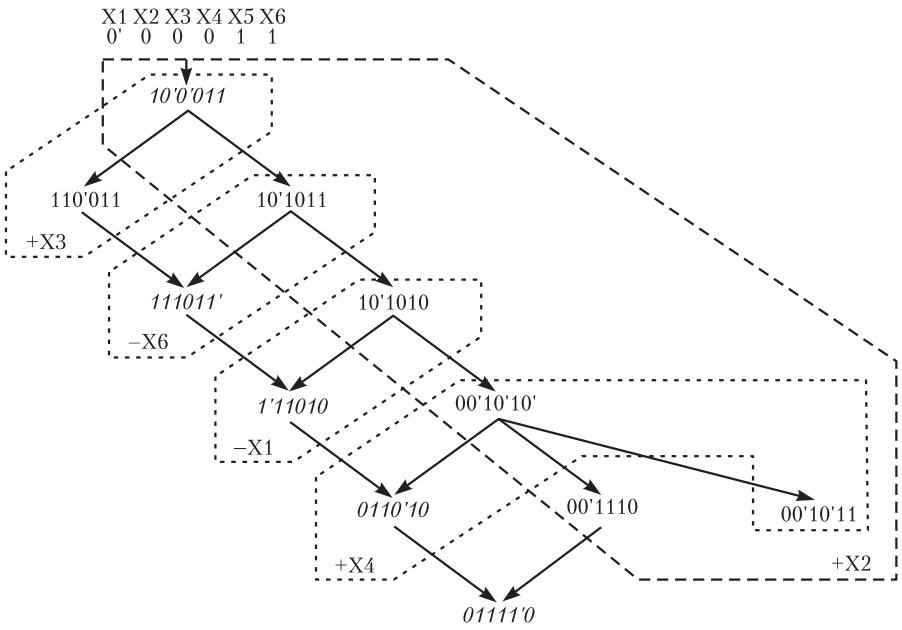


Рис. 1 Диаграмма переходов для случая нарушения полумодулярности при «перехвате термов». Курсивом приведены «скелетные» состояния; пунктиром обведены зоны «возбуждения» переменных схемы. Помеченные символом ′ — возбужденные состояния

Перечисленных проверок достаточно для корректного анализа возможных нарушений полумодулярности на базе событийного подхода в рамках одного слоя ДИ, за исключением специального «пограничного» случая.

Таким сложным случаем для событийного анализа является ситуация, когда дистрибутивная модель функционирования элементов реализуется как вырождение полумодулярной за счет строгих причинно-следственных связей между входными переменными. Здесь часто возникает ситуация, получившая название «*перехват термов*». Она характеризуется последовательным взаимосвязанным срабатыванием и выключением отдельных импликант переменной типа «ИЛИ», обеспечивающим для этой переменной дисциплину срабатывания типа «И». Сам «перехват термов» не вызывает нарушений полумодулярности, но ошибочная его реализация может приводить к «скрытым» формам нарушений, не видимым на одном слое ДИ. Пример «перехвата термов» в виде ДП со сложным случаем нарушения полумодулярности представлен на рис. 1. На рис. 2 представлена событийная модель этого процесса в виде ДИ.

В представленном примере в процессе переключений задействованы три импликанты переменной X6, две из которых срабатывают последовательно

по +X1, +X3 и +X3, +X2, вызывая переключение −X6. Последнее, в свою очередь, отключает первую импликанту по −X1, которая и включает третью по +X4. Между отключением первой и срабатыванием третьей импликант значение −X6 удерживается второй импликантой. Выделенные курсивом на рис. 1 и приведенные в виде последовательности на рис. 2 состояния схемы, называемые «*скелетными*» состояниями, полностью описывают поведение дистрибутивной схемы и не обнаруживают никаких признаков нарушения полумодулярности. Диаграмма изменений демонстрирует отсутствие специальных дуг, нарушающих дистрибутивность схемы, и выглядит корректной. Однако ДП на рис. 1 явно показывает наличие нарушения полумодулярности сразу в двух зонах возбуждения: +X2 и +X4. Здесь переменная X6 имеет одновременно противоположные значения в одном слое. Классификационный анализ определяет их как «*внескелетное*» по +X2 и «*прозрачное*» +X4 нарушения полумодулярности, невидимые на «скелетных» состояниях схемы.

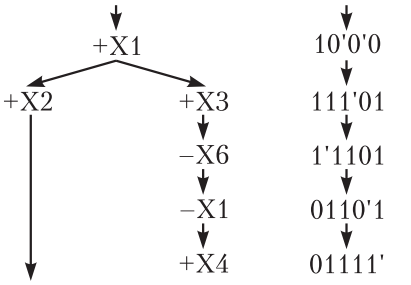


Рис. 2 Диаграмма изменений и соответствующая последовательность «скелетных» состояний схемы для случая нарушения полумодулярности при «перехвате термов»

Причина расхождения в результатах анализа данного случая по ДП и ДИ заключается в том, что в результате срабатывания первых двух импликант переменной X6 образуется вырожденное «ИЛИ», которое требует совместного срабатывания +X2 и −X6 до начала срабатывания −X1. Поскольку в данном примере этого не было предусмотрено, то после срабатывания −X1 переменная X2 могла иметь любое значение, что и стало причиной конфликта, не наблюдаемого на «скелетных» состояниях и не отраженного в ДИ.

Следовательно, анализ ситуаций, связанных с «перехватом термов», требует дополнительной проверки завершенности реализаций вырожденных «ИЛИ» для обеспечения корректной индикации нарушений полумодулярности. Классификационный анализ, определяющий тип конфликтной ситуации, позволяет в этом случае и определить причину, вызвавшую это нарушение, и дать важное предметное указание разработчику схемы.

Описанная методика событийного анализа самосинхронных схем реализована в подсистеме АСПЕКТ [7]. Полученное подтверждение полиномиального характера зависимости временных затрат от величины внутренней параллельности процессов переключений элементов схем проиллюстрировано в табл. 1. В ней приведены данные временны́х затрат процессорного времени персонального компьютера (ПК) для решения задачи анализа n-разрядного арифметико-логиче-

Таблица 1 Зависимость временны́х затрат ПК для анализа самосинхронных АЛУ от внутренней параллельности (на примере n-разрядных самосинхронных АЛУ)

Схема	Число элементов схемы	Параллельность	Число событий ДИ	Время счета
АЛУ-2	92	14	624	< 1 с
АЛУ-4	156	26	1244	~ 1 с
АЛУ-8	283	50	2444	~ 6 с
АЛУ-16	537	98	4808	~ 80 с
АЛУ-32	1047	192	9568	24 мин 30 с

ского устройства (АЛУ), где $n = 2, 4, 8, 16, 32$, позволяющие приблизительно оценить степенную зависимость временны́х затрат как $\sim n^5$. Эти временны́е затраты включают в себя затраты как на алгоритмы анализа, так и на алгоритмы предобработки входных данных и организации диагностических сообщений. Это несколько искажает вид приведенной зависимости для небольших схем.

5 Организация диагностики в подсистеме классификационного анализа

Модули диагностических сообщений в подсистемах анализа самосинхронных схем выполняют функцию основного внешнего интерфейса с разработчиком электронных схем. От точности, конкретности и наглядности его сообщений в значительной степени зависят скорость разработки и эффективность схемотехнических решений.

В процессе развития программных средств анализа модули диагностики претерпели серьезные эволюционные изменения.

В подсистеме ТРАНАЛ [2] диагностировалась пара переменных, ответственных за возникновение конфликта и нарушение полумодулярности. Этого достаточно для системы из 10–20 уравнений.

В подсистеме АСИАН [1] диагностический модуль позволял увидеть весь процесс переключений в виде графического представления диаграммы изменений и диагностировать на ней нарушение самосинхронности. Для диагностирования ситуаций нарушения самосинхронности в подсистеме АСПЕКТ этого уже недостаточно. Диаграмма изменений устройств, исследуемых в этой подсистеме, может содержать до нескольких миллионов событий переключения элементов. Анализировать причины возникновения нарушений самосинхронности по такой объемной диаграмме трудно и неэффективно. Диагностика в подсистеме АСПЕКТ реализует выделение фрагментов ДИ, в которых локализуется обнаруженное нарушение, и организует их визуализацию в виде графа событий, содержащего причину и результат этого нарушения.

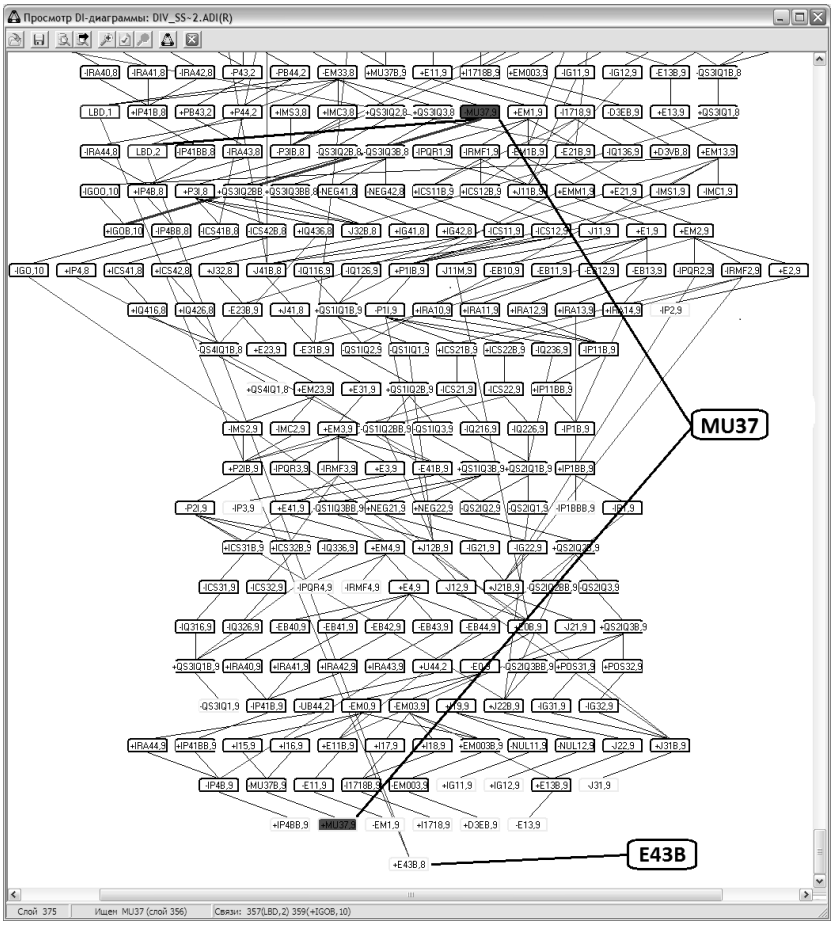


Рис. 3 Диаграмма изменений схемы управления 32-разрядного самосинхронного вычислительного блока

Фрагмент ДИ на рис. 3 иллюстрирует ситуацию нарушения полумодулярности в схеме управления 32-разрядного самосинхронного вычислительного устройства (блока деления/извлечения корня [8]). Элемент MU37, переключаясь из 1 в 0, вызывает срабатывание элемента E43B, а спустя 19 слоев ДИ снова переключается из 0 в 1. Это последнее переключение никак не зависит от срабатывания элемента E43B, что и создает нарушение самосинхронности.

На рис. 4 представлен фрагмент ДИ, созданной диагностической системой АСПЕКТ. Этот фрагмент отражает описанное нарушение: в нем наглядно демонстрируется отсутствие элемента E43B в цепи переключений элемента MU37

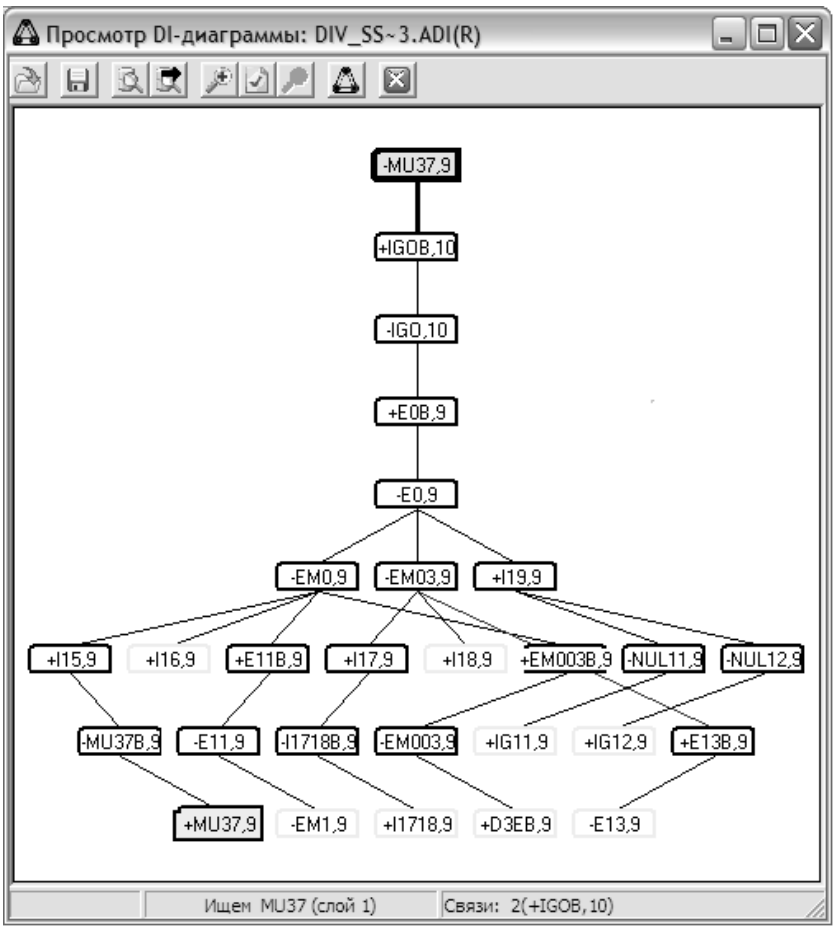


Рис. 4 Пример диагностического сообщения подсистемы АСПЕКТ

из 1 в 0 и обратно. Этот пример демонстрирует, как специальные средства диагностической системы выделяют из всей ДИ только события и связи между ними, ответственные за возникновение конфликта, и представляют их в удобной для анализа форме.

Дальнейшее расширение интеллектуализации процесса разработки самосинхронных схем достигается за счет использования классификационного анализа. Классифицируя характер процессов в схеме (взаимодействия событий) как дистрибутивный или полумодулярный, тип диагностируемых нарушений (конфликтов) как «скелетные», «скелетные» с добавлением фиктивных событий,

«внескелетные» или «прозрачные», можно получить дополнительную информацию о причинах конфликтов и способах их разрешения [3]. Например:

– возникновение последних двух типов конфликтов практически однозначно указывает на некорректную форму вырождения «ИЛИ» и требует ее уточнения;

– обнаруженные «прозрачные» конфликты могут быть легко исправлены с помощью простых алгоритмов изменений схемы [3].

Развитые и интеллектуализованные диагностические средства в подсистемах событийного анализа являются необходимым условием эффективной разработки самосинхронных схем вследствие повышенной степени сложности разрабатываемых изделий.

6 Выводы

1. Разработана методика анализа времянезависимых схем на базе построения событийных моделей асинхронных процессов переключений элементов схемы. Создан набор алгоритмов, обеспечивающий полную и корректную процедуру анализа.
2. С помощью методов классификационного анализа исследована сложная конфликтная ситуация в процессе переключения элементов, реализующем «перехват термов», и предложена методика диагностики таких нарушений.
3. Разработанные средства диагностики и визуализации ДИ позволяют выделять и исследовать отдельные фрагменты событийной модели процесса переключения логических элементов, в которых выявлены нарушения самосинхронности. Применение методов классификационного анализа конфликтных ситуаций в ДИ повышает эффективность диагностических средств и облегчает понимание причин этих нарушений.

Авторы выражают благодарность Ю. Г. Дьяченко за помощь в работе.

Литература

1. *Рождественский Ю. В., Морозов Н. В., Степченков Ю. А., Рождественскене А. В.* Универсальная подсистема анализа самосинхронных схем // Системы и средства информатики. — М.: Наука, 2006. Вып. 16. С. 463–475. URL: http://elibrary.ru/item.asp?id=13060493 (дата обращения 30.05.2011).
2. *Варшавский В. И., Карпов С. А., Кондратьев А. Ю., Степченков Ю. А.* Инструментальные средства автоматизации проектирования самосинхронных схем // Системы и средства информатики. — М.: Наука, 1993. Вып. 5. С. 196–213.
3. *Varshavsky V., Kishinevsky M., Marakhovsky V., et al.* Self-timed control of concurrent processes. — Kluver Academic Publs., 1990. 245 p.

4. *Muller D. E., Bartky W. S.* A theory of asynchronous circuits // Symposium (International) on Theory of Switching Proceedings. — Cambridge, MA: Harvard University Press, 1959. P. 204–243.
5. *Степченков Ю. А., Денисов А. Н., Дьяченко Ю. Г., Гринфельд Ф. И.* Библиотека элементов базовых матричных кристаллов для критических областей применения // Системы и средства информатики. — М.: Наука, 2004. Вып. 14. С. 318–361.
6. *Морозов Н. В., Рождественский Ю. В.* Средство анализа системы булевых уравнений на полумодулярность и дистрибутивность. Свидетельство об официальной регистрации программы для ЭВМ № 2001610157 от 14.02.2001.
7. *Рождественский Ю. В., Морозов Н. В., Рождесвенскене А. В.* Самосинхронный вычислитель для высоконадежных применений // Всеросс. науч.-техн. конф. «Проблемы разработки перспективных микро- и наноэлектронных систем». — М.: ИППМ РАН, 2010. С. 26–31. URL: http://elibrary.ru/item.asp?id=15257392 (дата обращения 30.05.2011).
8. *Степченков Ю. А., Дьяченко Ю. Г., Рождественский Ю. В., Морозов Н. В., Степченков Д. Ю.* Квазисамосинхронная реализация устройства деления и извлечения квадратного корня // Системы и средства информатики. — М.: Наука, 2008. Вып. 18. С. 234–260. URL: http://elibrary.ru/item.asp?id=13043595 (дата обращения 30.05.2011).

СРЕДСТВА ПОДДЕРЖКИ ИСПОЛНЯЕМОГО КОДА, СИНТЕЗИРОВАННОГО ПО СПЕЦИФИКАЦИЯМ, НА ЯЗЫКЕ CELL

***О. А. Бондаренко*[1], *К. И. Волович*[2], *В. А. Кондрашев*[3]**

Аннотация: В статье рассматриваются аспекты генерации служебного программного кода, обеспечивающего функционирование протокольных автоматов, разработанных на языке Cell. Описывается понятие «исполняющей системы» как совокупности программного кода, включаемого компилятором языка Cell в процедурный код с целью обеспечения функционирования основного алгоритма протокольного автомата. Сериализация процедурного кода, выполняемая компилятором Cell во время его синтеза, предполагает возможность эффективного функционирования автомата без использования операционной системы (ОС), что накладывает на исполняющую систему задачи управления всеми ресурсами, используемыми автоматом.

Ключевые слова: язык Cell; ячейка; исполняющая система; управление ресурсами; сериализация; иерархический автомат; синтез программного обеспечения; телекоммуникационный протокол

1 Введение

Современные телекоммуникационные системы функционируют на базе широкого спектра протоколов, каждый из которых описывается соответствующим стандартом. Как правило, описание телекоммуникационного протокола в таком стандарте состоит из формальной (блок-схемы алгоритмов, описание блоков данных) и неформальной части, содержащей словесное описание функционирования системы. Создание на основе такого описания программного обеспечения, реализующего телекоммуникационный протокол, является творческой задачей для инженера-программиста, фактически не поддающейся автоматизации.

Язык Cell, предлагаемый авторами в качестве средства разработки программного обеспечения телекоммуникационных протоколов, дает возможность описать алгоритм функционирования компонентов телекоммуникационной системы формализованным способом. Наличие такого языка позволяет производить

[1]Институт проблем информатики Российской академии наук, olga@ipi.ac.ru
[2]Институт проблем информатики Российской академии наук, kv@ipi.ac.ru
[3]Институт проблем информатики Российской академии наук, vd@ipi.ac.ru

автоматизированный синтез программного обеспечения путем трансляции формальных спецификаций на языке Cell на язык реализации, подходящий для использования на конкретных вычислительных платформах. В качестве языка реализации авторами выбран язык C.

Однако простой трансляции кода протокольного автомата на язык реализации, а затем в машинные коды недостаточно для построения функционирующей системы. Необходимо также решить вопросы управления распределением памяти, времени, других ресурсов вычислительной платформы, межпроцессного взаимодействия.

Для решения данных вопросов обычно применяются два подхода:

(1) исполнение кода с использованием ОС как средства управления ресурсами и предоставления сервиса приложениям;

(2) исполнение целевого кода непосредственно вычислительной платформой без использования ОС.

В первом случае все функции по управлению памятью, распределением процессорного времени между процессами, межпроцессному взаимодействию могут быть возложены на службы ОС. Во втором случае эти функции должны быть включены в код протокольного автомата.

Каждый из подходов имеет свои достоинства и недостатки в условиях применения во встраиваемых системах. Использование ОС исключает необходимость подключения дополнительного кода на этапе компиляции автомата, что облегчает разработку компилятора языка Cell. Однако операционная система требует дополнительных ресурсов со стороны оборудования, что в условиях встраиваемых систем является существенным. Кроме того, механизм обеспечения параллельности (псевдопараллельности) функционирования компонентов протокольного автомата средствами ОС (процессы, нити) не требуется в рамках концепции сериализации, заложенной в основу языка Cell. Более того, для обеспечения механизма последовательного исполнения кода, сгенерированного по спецификациям языка Cell, в условиях функционирования под управлением ОС потребуется использование механизма блокировок, семафоров и иных средств ОС, обеспечивающих параллельность исполнения, т. е. использование именно тех средств организации вычислительного процесса, от которых позволяет отказаться концепция сериализации.

Второй подход, предусматривающий включение кода исполняющей системы непосредственно в код протокольного автомата, позволяет полностью реализовать механизмы сериализации. Управление одновременностью исполнения логически независимых частей протокольного автомата в этом случае обеспечивается самим кодом протокольного автомата. Параллельно функционирующие логические модули в данном случае не оформляются как отдельные объекты в рамках ОС (процессы, нити), а выделяются исключительно в рамках логики исполне-

ния одного процесса протокольного автомата, исполняемого последовательно на аппаратных средствах встраиваемой системы.

Исходя из вышеизложенных посылок, авторы предлагают при генерации целевого кода протокольного автомата использовать второй подход, т. е. включать в состав протокольного автомата элементы исполняющей системы, обеспечивающих функционирование без использования ОС.

Данный подход обеспечивает мобильность кода между различными аппаратными платформами, поскольку не содержит аппаратно зависимых компонентов. Привязка к конкретному оборудованию может осуществляться путем применения аппаратного драйвера, обеспечивающего прием и отправку низкозкоуровневых данных через буфер в памяти с использованием прерываний. Интерфейс взаимодействия с драйвером может предоставляться внешней по отношению к автомату библиотекой и не являться непосредственно принадлежностью протокольного автомата.

Разработка алгоритмов функционирования исполняющей системы является отдельной задачей при создании компилятора. Программный код исполняющей системы, включаемый в результирующий код протокольного автомата, является частью библиотек компилятора. Взаимодействие собственно автоматного кода и библиотек обеспечивается через программный интерфейс, вызовы которого формируются на этапе компиляции.

Далее в статье показаны структурные модули исполняющей системы и алгоритмы их функционирования, обеспечивающие работу протокольного автомата без использования механизмов ОС.

2 Структура исполняющей системы

Основной функцией исполняющей системы является распределение ресурсов и управление исполнением независимыми частями автомата.

Авторами предлагается алгоритм генерации целевого кода протокольного автомата, при котором компилятор языка Cell производит статическое выделение ресурсов для каждого объекта независимого исполнения (контекста процесса, очередей, области данных). Таким образом, первичное распределение ресурсов уже проведено на этапе периода компиляции. На этапе исполнения от диспетчера ресурсов исполняющей системы требуется лишь активация того или иного статического объекта в памяти.

Также к функциям исполняющей системы относится управление очередями сигналов, обеспечивающее последовательное (сериализованное) исполнение процедурного кода. Можно выделить основные модули, составляющие исполняющую систему:

(1) модуль управления очередями;

(2) модуль управления контекстами;

(3) модуль управления реакциями автомата (модуль переходов);

(4) модуль управления областями данных сигналов;

(5) модуль контроля использования ресурсов;

(6) модуль обработки исключительных ситуаций.

3 Алгоритмы функционирования исполняющей системы

3.1 Модуль управления очередями

Модуль управления очередями предоставляет автомату интерфейс к функциям помещения сигналов в очередь и извлечение сигналов из очередей. Основной программный цикл (message loop), интегрируемый компилятором в исходный код автомата, использует функции GetSignal и DispatchSignal для получения сигнала из внешней очереди и обработки его кодом протокольного автомата соответственно.

Спецификой построения логики работы автомата является наличие двух очередей для внешних и внутренних сигналов. Внешние сигналы принимаются от аппаратно-зависимого драйвера и размещаются в очереди внешних сигналов, в то время как внутренние сигналы генерируются самим автоматом и размещаются, соответственно, в очереди внутренних сигналов.

Указанная пара очередей привязывается к контексту ячейки. Как показано в [1], каждой корневой ячейке сопоставляется свой контекст.

При генерации программного кода компилятор интегрирует неявным образом код управления очередями в протокольный автомат, причем функции управления очередями в качестве одного из аргументов получают указатель на контекст, с которым ассоциируется корневая ячейка. Таким образом обеспечивается реентерабельность кода управления очередями и возможность работы с очередями как базовой ячейки, так и клонов.

Область хранения сигналов (тело очередей) размещается в каждом контексте и представляет собой с точки зрения программирования массив структур, включающих в себя указатель на дескриптор сигнала, и поля аргументов, передаваемых вместе с сигналом. Функции управления очередью производят размещение сигналов в ту или иную часть очереди в зависимости от их типов [2].

Функция размещения сообщения в очереди копирует аргументы из области данных аппаратного драйвера в область данных автомата, обеспечивая тем самым отсутствие необходимости хранения данных в области аппаратных очередей.

Функция извлечения сообщений из очереди производит копирование данных в выделенную область памяти контекста с последующим освобождением очереди, пересортировкой оставшихся сигналов, пересчетом индексов.

Глубина очередей не является конечной и не определяется на этапе компиляции. Выделение памяти для элементов очереди происходит динамически. По этой

причине имеется необходимость включения в исполняемый код блока динамического управления распределением памяти. Блок динамического распределения памяти включается в модуль контроля управления ресурсами и в случае необходимости позволяет выделить память для объектов очереди из нераспределенной области (кучи), выделяемой на этапе компиляции.

Очевидно, что определение размера кучи на этапе компиляции представляет собой трудноразрешимую задачу. Явных указаний в тексте протокольного автомата на возможную длину очередей не содержится, поэтому решение этой задачи носит эвристический характер.

Управление размером кучи в период компиляции производится специальными директивами компилятору, на основании которых он осуществляет статическое распределение памяти в сегменте данных. Проверку правильности распределения необходимо проводить путем верификации и отладки результирующего кода.

Отметим, что контроль за переполнением очередей в период исполнения производится модулем контроля использования ресурсов, функции которого будут рассмотрены ниже. В случае проведения верификации модуль контроля можно использовать для мониторинга заполнения очередей каждого контекста.

3.2 Модуль управления контекстами

Модуль управления контекстами обеспечивает хранение полного списка контекстов корневых ячеек протокольного автомата. Список контекстов является статическим и определяется на этапе компиляции.

Основной структурной единицей исполняемого кода является контекст корневой ячейки. Исполняющая система содержит в своем составе диспетчер управления контекстами, обеспечивающий переключение между независимо функционирующими частями автомата.

Для простоты будем считать, что контекст базовой ячейки всегда один, а множественность контекстов появляется при клонировании.

С целью сохранения ресурсов целевого вычислителя компилятор языка Cell генерирует для автоматов-клонов реентерабельный процедурный код, позволяющий использовать один экземпляр исполняемого кода для всех однотипных автоматов-клонов.

При этом исполнение кода того или иного клона обеспечивается переключением контекста средствами рассматриваемого модуля.

Модуль управления контекстами представляет собой библиотеку периода исполнения, подключаемую к целевому коду протокольного автомата на этапе компиляции. В задачи модуля входит:

– поддержание списка активных контекстов на каждый момент времени функционирования протокольного автомата;

– переключение контекста текущего процесса в зависимости от сигнала, обрабатываемого функцией DispatchSignal модуля управления очередями.

Для реализации указанных функций модуль управления контекстами должен хранить таблицу ассоциаций областей данных контекстов и идентификаторов процессов клонов.

При выборе из очереди внешних сигналов сигнала, адресованного конкретному клону, модуль управления контекстами настраивает скрытые переменные среды (генерируемые компилятором неявно) на конкретную область памяти в массиве контекстов. Дальнейшие вызовы функций-обработчиков производятся с передачей в них настроенного контекста и, соответственно, выполняются в рамках конкретного клона. Поскольку контексты содержат полный состав пользовательских данных локального и общего классов видимости, то такое переключение позволяет функциям реализации переходов протокольного автомата работать на пространстве переменных конкретного клона до тех пор, пока очередной сигнал из очереди внешних сигналов не переключит контекст клона.

Поскольку распределение памяти для хранения контекстов производится на этапе компиляции, то при управлении контекстами (активации и деактивации) каждой корневой ячейки выполняется проверка доступности ресурса и отсутствия конфликта между количеством запрашиваемых и выделенных при компиляции контекстов. Количество контекстов каждого типа настраивается с помощью директив компилятору и не может быть изменено в период исполнения. В отличие от размера очередей сигналов количество клонов каждого типа можно оценить более точно, поскольку в структуре протокольного автомата каждый клон отвечает за некоторый сервис, а количество сервисов каждого типа можно определить на этапе проектирования автомата по его спецификациям. Тем не менее, компилятор включает в протокольный автомат код проверки переполнения списка контекстов клонов. Контроль производится с помощью модуля контроля использования ресурсов. Аналогично случаю контроля заполнения очередей сигналов модуль контроля может использоваться для мониторинга утилизации памяти контекстов при верификации автомата. В случае обнаружения переполнения модуль контроля генерирует исключительную ситуацию, которая также обрабатывается специализированной библиотекой, подключаемой к автомату на этапе компиляции.

Каждый контекст клона содержит собственные экземпляры очередей внешних и внутренних сигналов, поэтому функции отправки сигналов взаимодействуют с модулем управления контекстами с целью получения доступа к нужной очереди. Взаимодействие осуществляется неявно за счет скрытого кода, интегрируемого на этапе компиляции, позволяющего транслировать идентификатор клона в адрес структуры-контекста, содержащей конкретные очереди. Механизм установления указанной ассоциации скрывается от разработчика автомата на языке Cell, который в качестве процесса-получателя сигнала использует идентификатор клона, возвращенный функцией клонирования.

3.3 Модуль управления реакциями автомата (модуль переходов)

Модуль управления реакциями автомата обеспечивает поддержку актуального списка сигналов, обрабатываемых автоматом в текущий момент времени, а также вызов необходимой функции перехода по идентификатору сигнала.

Список реакций протокольного автомата хранится в контексте в виде списка адресов функций переходов, возможных в текущий момент времени. Учитывая иерархичность протокольного автомата, на каждом уровне иерархии имеется свой набор возможных реакций.

Основная задача модуля — формирование нового списка активных сигналов в векторе перехода после выполнения каждой функции перехода. Для этого компилятор включает в тело каждой функции перехода код, обеспечивающий вызов функции, производящей модификацию вектора перехода. Данная функция удаляет из вектора реакции, определяющие предыдущее состояние автомата, и устанавливает текущие.

Модуль управления реакциями предоставляет модулю управления очередями интерфейс для обращения к функции перехода по идентификатору сигнала.

Таким образом, модуль управления очередями выполняет алгоритм обработки сигнала путем выборки из очереди внешних сигналов, а затем обработки очереди внутренних сигналов до полного ее опустошения. При этом для каждого сигнала (внешнего или внутреннего) вызывается функция из состава модуля управления реакциями для поиска адреса перехода, а каждый переход завершается вызовом функции установки новых реакций.

Указанный алгоритм функционирования модуля управления реакциями автомата позволяет постоянно поддерживать в контексте протокольного автомата полный набор реакций на всех уровнях иерархии.

При возникновении исключительных ситуаций модуль управления реакциями вызывает соответствующие функции модуля обработки исключительных ситуаций.

Исключения периода исполнения могут возникать в следующих случаях:

- отсутствие в векторе переходов функции-обработчика для данного состояния;
- попытка записи в вектор перехода нескольких функций-обработчиков для одного сигнала.

Вообще говоря, в нормальных условиях функционирования указанные ситуации не должны возникать в период исполнения, поскольку на этапе компиляции производится проверка соответствия устанавливаемых реакций состоянию, в которое переходит протокольный автомат. Однако при нарушении логики функционирования протокольного автомата, описанного на языке Cell, вызванного зацикливанием, бесконечной рекурсией, другими ошибками периода исполнения, не диагностируемыми на этапе компиляции, такая ситуация возможна. Поэтому

обработка исключительных ситуаций такого характера особенно важна в период отладки и верификации автоматного кода.

В режиме отладки функции модуля управления реакциями, а также функции обработки исключительных ситуаций могут быть переопределены с целью получения диагностической информации. В рабочем режиме функционирования автомата исключительная ситуация, связанная со множественной записью вектора перехода, является критической и должны приводить к остановке или перезапуску корневой ячейки текущего автомата.

3.4 Модуль управления областями данных сигналов

Модуль управления областями данных сигналов предназначен для хранения и обработки протокольных блоков данных (PDU — Protocol Data Unit), передаваемых внешними сигналами.

Основная задача модуля — предоставление доступа коду протокольного автомата к данным, содержащимся в PDU.

Алгоритм обработки PDU предполагает размещение PDU в контексте протокольного автомата (базового или клона) и обеспечения к нему доступа процедурного кода из любой функции перехода.

Код модуля обработки PDU не генерируется компилятором, а предоставляется разработчиком в качестве внешней программной библиотеки для компоновки с кодом протокольного автомата. В состав библиотечных функций входят функции для кодирования/раскодирования полей PDU, используемых протокольным автоматом. Протокольный автомат использует для доступа к данным интерфейс, предоставляемый библиотекой. В функции раскодирования передается адрес области памяти, содержащей собственно PDU и адрес области для размещения декодированных данных. Структура области декодированных данных определяется в ходе описания переменных диспетчерского, общего или локального классов видимости в коде протокольного автомата [2–4]. Таким образом, автомат получает возможность обращения к области декодированных данных как к структуре, определенной на этапе компиляции.

Для корректного обращения к областям декодированных данных разработчик должен следить, чтобы все обращения выполнялись после инициализации данных путем вызова функции раскодирования, причем вызов должен завершиться успешно.

Для размещения данных в существующий PDU используется функция кодирования, предоставляемая указанной библиотекой. Функция обеспечивает упаковку структурированных данных протокольного автомата в битовый поток и размещение его в указанной области памяти. Кодирование вызывается из тела перехода протокольного автомата, и обработка кодов завершения указанной функции возлагается на разработчика автомата на языке Cell.

Контроль использования ресурсов и отсутствие нарушений в использовании памяти возлагаются на саму библиотеку модуля управления областями данных PDU. С целью контроля границ областей памяти в функции библиотеки передается размер каждой области, подлежащей заполнению, вычисляемый на этапе компиляции. Функции модуля сами отслеживают выход за допустимые границы областей, в том числе и при рекурсивной обработке. В случае недостаточности выделенной области функция раскодирования прекращает работы и возвращает код ошибки.

При кодировании данных в существующий PDU производятся аналогичные проверки границ, и в случае некорректно выделенной области памяти возвращается ошибка периода исполнения. При этом сохранность предыдущих данных, расположенных в PDU, определяется реализацией функций библиотеки. В общем случае можно считать, что старые данные, содержащиеся в PDU, при неуспешном завершении функции кодирования будут уничтожены.

Таким образом, внешний по отношению к коду протокольного автомата модуль обеспечивает доступ кода автомата к распакованным структурированным данным, передаваемым с внешними сигналами, а также упаковку структурированных данных в PDU для передачи с использованием сигналов.

3.5 Модуль контроля использования ресурсов

Модуль контроля использования ресурсов предназначен для проверки границ областей памяти, распределенной на этапе компиляции. Как отмечалось выше, при функционировании протокольного автомата может возникнуть нехватка ресурсов, выделенных компилятором. В частности, это относится к количеству контекстов той или иной корневой ячейки.

Код модуля контроля использования ресурсов генерируется компилятором и включает в себя в качестве данных таблицы распределения ресурсов всех типов, распределенных компилятором. В каждую функцию, занимающую ресурс (например, клонирование) компилятор включает вызов функции проверки доступности запрашиваемого ресурса. В случае наличия свободного ресурса модуль помечает его в таблице как занятый и возвращает код завершения, свидетельствующий о возможности использовать ресурс. В случае отсутствия возможности выделить ресурс модуль возвращает код ошибки. Задача проверки кода ошибки ложится на разработчика автомата на языке Cell.

Другой функцией модуля контроля ресурсов является динамическое выделение памяти для размещения сигналов в очередях. Выделение происходит неявным образом. В каждую функцию отправки сигнала добавляется код вызова модуля с целью получения указателя на область данных для размещения сигнала. Модуль контроля выделяет из кучи запрашиваемое количество памяти, ассоциирует его в своих внутренних таблицах с текущим контекстом и возвращает указатель на

область памяти. Таким образом, для каждого сигнала выделяется необходимое и достаточное количество памяти (с учетом передаваемых с сигналом данных).

Возвращение памяти в кучу производится при извлечении сигналов из очередей. В функцию извлечения сигнала из очереди добавляется вызов функции, освобождающей память. Данная функция помечает блок данных как неиспользуемый и возвращает его в кучу для дальнейшего распределения.

Необходимо отметить, что такой механизм распределения памяти неизбежно порождает фрагментацию. Это снижает эффективность функционирования и может приводить к невозможности выделения ресурсов для конкретного сигнала, несмотря на то, что общее количество памяти в куче достаточно, но разбито на большое количество мелких фрагментов, каждого из которых недостаточно для выделения достаточного непрерывного блока.

Для борьбы с фрагментацией традиционно применяются механизмы «сборки мусора» либо виртуальной памяти.

Авторы считают, что программная реализация механизма виртуальной памяти, предусматривающая страничную организацию памяти, трансляцию виртуальных адресов в физические слишком громоздка для встраиваемых систем. Такая задача обычно выполняется средствами ОС, от которой авторы решили отказаться. Ряд процессоров предлагают аппаратную реализацию виртуальной памяти, однако использование такой возможности нарушило бы принцип мобильности и независимости кода от аппаратной платформы.

Реализация же сборки мусора не требует применения подобных решений и с точки зрения авторов более подходит для использования в коде протокольного автомата. Вместе с тем, необходимо отметить, что сборка мусора может потребовать больше ресурсов периода исполнения, чем виртуальная память.

Программный код сборщика мусора включается компилятором в код автомата и вызывается каждый раз при извлечении сигналов из очереди. Однако производить сортировку кучи при каждом извлечении сигнала представляется нецелесообразным, поскольку приводит к большим затратам процессорного времени.

Для определения необходимости запуска сборки мусора используется алгоритм оценки фрагментации, позволяющий вычислить некоторый индекс (степень фрагментации памяти). В случае превышения индексом порогового значения при очередном извлечении сигналов из очереди запускается механизм дефрагментации. Также механизм дефрагментации запускается в период простоя системы.

Чтобы иметь возможность осуществлять не только слияние соседних освобожденных участков кучи, но и перемещать используемые участки памяти, функции распределения памяти возвращают указатель не непосредственно на распределенную память, а на ячейку, содержащую указатель на распределенную память. Таким образом, сборщик мусора при сортировке имеет возможность переметить используемую область памяти, поменяв ее адрес в указанной ячейке. При этом

не произойдет сбоя функционирования основного кода, поскольку все обращения к памяти производятся по указателям, а те, в свою очередь, поддерживаются в актуальном состоянии.

Описанный механизм управления памятью скрыт от разработчика протокольного автомата на языке Cell и включается в код автомата компилятором.

Отметим, что вычисление индекса фрагментации и его порогового значения носит эвристический характер. По умолчанию он принимается равным количеству фрагментов в куче, а порог срабатывания — половине общего количества возможных в системе сигналов. Очевидно, что такой алгоритм не является оптимальным, однако позволяет время от времени запускать дефрагментацию.

Более оптимальное управление дефрагментацией достигается путем вычисления индекса с учетом количества и размера фрагментов и сильно зависит от самого протокольного автомата. Поэтому в исполняющей системе предусмотрен механизм переопределения функции подсчета индекса и установки порога срабатывания. Пользователь может определить свою функцию и включить ее в исполняемый код на этапе компоновки.

В случае невозможности выделения памяти для размещения сигнала механизм дефрагментации вызывается принудительно, если же сигнал невозможно разместить и после дефрагментации, вызывается исключительная ситуация.

3.6 Модуль обработки исключительных ситуаций

Модуль обработки исключительных ситуаций вызывается из любой точки системы в случае нештатного функционирования протокольного автомата.

Как показано выше, исключительные ситуации могут возникать в случае нарушений в работе протокольного автомата или в случае нехватки ресурсов (в частности, памяти для размещения сигналов в очередях).

По умолчанию в случае возникновения исключительной ситуации модуль производит освобождение всех ресурсов, относящихся к контексту, вызвавшему исключение, и помечает данный контекст как свободный. Если указанные действия относятся к автомату-клону, то происходит рекурсивное освобождение всех ресурсов, относящихся к контекстам текущего и вышележащих клонов. Если исключительная ситуация возникает в контексте базовой ячейки, то происходят полное освобождение всех ресурсов и перезапуск автомата.

Функции модуля обработки исключений могут быть переопределены пользователем для нужд отладки и верификации кода.

4 Заключение

Поддержка исполняемого кода протокольного автомата, синтезированного по спецификациям, на языке Cell осуществляется путем интеграции компилятором специализированного скрытого кода в целевой код протокольного автомата.

Основные принципы функционирования указанного кода состоят в сокрытии механизмов управления ресурсами от разработчика программы на языке Cell и обеспечения функционирования протокольного автомата в условиях отсутствия ОС.

Данное решение позволяет использовать протокольные автоматы, специфицированные на языке Cell в условиях встраиваемых систем, характеризующихся ограниченными ресурсами вычислительных платформ.

Литература

1. *Бондаренко Т. В., Волович К. И., Кондрашев В. А.* Язык Cell — инструмент для синтеза программного обеспечения многоуровневого телекоммуникационного протокола по частично формализованным спецификациям // Вестник Воронежского государственного ун-та, 2011. Т. 7. № 3. С. 120–125.
2. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Сигнальный механизм языка Cell // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 45–66.
3. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Язык Cell: модель обработки клонов // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 67–81.
4. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Базовая модель функционирования автомата в системе программирования Cell // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 82–97.

АЛГОРИТМЫ ФУНКЦИОНИРОВАНИЯ КОМПИЛЯТОРА ЯЗЫКА CELL

О. А. Бондаренко[1], *К. И. Волович*[2], *В. А. Кондрашев*[3]

Аннотация: Описаны особенности компилятора языка Cell, разрабатываемого для программирования иерархических автоматов, применяемых в том числе для спецификации поведения телекоммуникационных протоколов (ТКП), приведены алгоритмы работы его синтаксического и семантического анализаторов, а также рассмотрены аспекты синтеза исполняемого кода иерархического автомата (ИА) на процедурном языке Си, сериализованного для последовательного выполнения команд.

Ключевые слова: язык Cell; иерархический автомат; сериализация вычислений; контекст иерархии; синтез программного обеспечения; телекоммуникационный протокол

1 Введение

Программирование ТКП — одна из актуальнейших задач создания новых средств коммуникации.

Официальные стандарты для формального описания и программирования ТКП появились более десяти лет назад [1, 2]. Параллельно с процессом стандартизации специализированных языков описания и программирования протоколов развивались технологии программирования событийно-управляемых (реактивных, реагирующих) систем. Одним из подходов для программирования таких систем были попытки расширить существующие процедурные и объектно-ориентированные языки средствами программирования автоматов (см., например, [3, 4]). К середине 2000-х гг. в России в области технологии программирования автоматов сформировалось направление «автоматного программирования» [5]. Тем не менее, общепризнанных универсальных подходов и средств для программирования ТКП пока не существует. Современные исследования в этой области связаны как с совершенствованием технологий применения официально признанных моделей MSC/SDL/UML (Message Sequence Chart / Specification and Description Language / Unified Modeling Language) (см., например, [6]), так и с созданием новых языков (см., например, [7]).

[1]Институт проблем информатики Российской академии наук, olga@ipi.ac.ru
[2]Институт проблем информатики Российской академии наук, kv@ipi.ac.ru
[3]Институт проблем информатики Российской академии наук, vd@ipi.ac.ru

Язык Cell — попытка авторов создать систему программирования многоуровневого ТКП, специфицированного иерархическим набором диаграмм состояний и переходов. Далее будут рассмотрены особенности языка Cell и алгоритмы функционирования его компилятора.

2 Особенности языка Cell

В предыдущих работах [8–11] авторы предложили использовать для программирования многоуровневого ТКП, специфицированного набором диаграмм состояний и переходов (ДСП), разработанную модель ИА, а также язык Cell, спроектированный для формального описания ИА. Предлагаемый в этих работах подход системы программирования Cell позволяет:

- формально связать традиционные спецификации ТКП в виде фрагментарных ДСП в единый ИА;
- специфицировать ИА на языке Cell, синтаксис которого спроектирован для описания таких автоматов с последующей их реализации в виде встраиваемых систем с ограниченными вычислительными ресурсами;
- синтезировать процедурный код ИА, ориентированный на последовательное выполнение команд алгоритма, на этапе компиляции программы на языке Cell, выполняя сериализацию вычислений ИА.

В отличие от других языков и систем программирования автоматов ИА-модель, разработанная для системы Cell, позволяет с помощью неявных связей описывать блок взаимодействующих автоматов, алгоритм функционирования которых подчиняется правилам древовидной активации одних автоматов из состояний других автоматов. В ИА-модели такая операция называется порождением подавтомата более высокого уровня иерархии состоянием подавтомата более низкого уровня иерархии (или операцией вложения подавтомата-потомка в состояние подавтомата-предка). Вместе с операциями возвратных межуровневых переходов, существующими в ИА-модели, такая организация автомата органичным образом описывает поведение ТКП как многоуровневой системы связанных (часто вербально) подавтоматов, каждый из которых специфицируется формальной ДСП.

Ключевой элемент языка Cell — ячейка (cell) — характеризует состояние подавтомата и определяет часть реакций совокупного ИА, заданных переходами из этого состояния в достижимые. Ячейки организуются в иерархическую структуру с целью систематизации переходов и создания неявных правил связывания подавтоматов (правила вложенности) в соответствии с ИА-моделью, что позволяет органично описать на языке Cell иерархическую структуру автомата, аналогичную иерархической структуре ТКП.

Функции перехода (реакции) на языке Cell задаются при спецификации ячеек. Переход связывает между собой одну исходную ячейку (состояние подавтомата) и одну или более целевых ячеек, определяемых множеством достижимых состояний из исходного. Переход инициируется (открывается) приемом сигнала и представляет собой фрагмент процедурного кода с реакцией на сигнал. Для спецификации процедурного кода в языке Cell решено использовать какой-либо существующий процедурный язык, дополненный директивами языка Cell для описания целевых состояний, отправляемых сигналов и других действий (см. ниже описание синтеза функций переходов). На первом этапе разработки системы программирования Cell — это язык Си.

Состояние ИА в системе Cell определяется как совокупность всех возможных переходов в заданный момент времени (вектор состояния), специфицированных группой ячеек, описывающих состояние подавтоматов в текущий момент времени на разных уровнях ИА. На этапе компиляции программы ИА на языке Cell генерируется служебный процедурный код, обеспечивающий логику последовательного исполнения автомата так, чтобы специфицированная сигналом реакция в текущем состоянии содержалась в описании этого сигнала в векторе состояний. Служебный код, выполняемый при завершении любого перехода, модифицирует вектор состояния: сначала отменяет текущие реакции на сигналы актуального состояния, а затем устанавливает необходимые реакции на сигналы, обрабатываемые в новом состоянии. Такая логика исполнения ИА позволяет сериализовать функционирование автомата — выполнять процедуры переходов автомата последовательно без использования методологии организации параллельных (псевдопараллельных) вычислений.

Сериализация вычислений также обеспечивается определенной последовательностью обработки принимаемых и отправляемых сигналов, что позволяет считать функцию перехода атомарной операцией: функция перехода переводит автомат в определенное состояние, не зависящее от сигналов, которые могут быть приняты в процессе исполнения перехода. Другими словами — переход не может быть прерван сигналом, а память автомата на протяжении исполнения перехода может модифицироваться только телом этого перехода.

Механизмом поддержки атомарности переходов является отложенная доставка сигналов. Отправка сигнала синтаксически может находиться в произвольном месте тела перехода, при этом отправляемые сигналы буферизуются и их фактическая отправка задерживается до завершения тела перехода. Таким образом, каждый переход формирует собственную очередь выходных сигналов, которая обрабатывается по завершении перехода, что реализует приоритетную обработку внутренних сигналов перед внешними (см. ниже правила формирования очереди сигналов). Механизм отложенной доставки сигналов гарантирует, что в любой момент времени может выполняться не более одного процедурного тела перехода. Соответственно состояние всех ячеек, кроме ячейки, выполняющей текущий

переход, будет вполне определенным и все разделяемые данные, кроме данных, изменяемых текущим переходом явно, будут находиться в определенных состояниях. При этом становится возможным совместный доступ различных ячеек к разделяемым данным без использования каких-либо механизмов синхронизации.

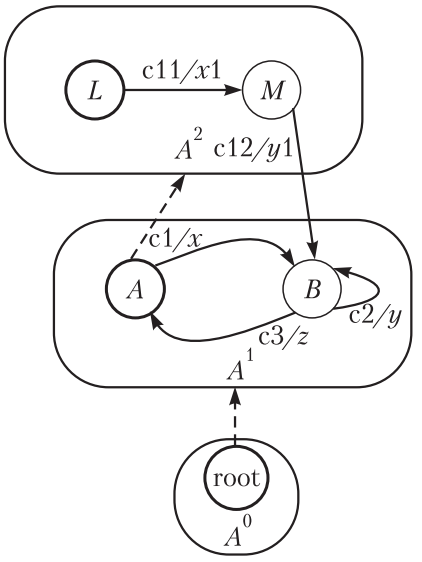
3 Компилятор языка Cell

Синтез процедурного кода ИА выполняется компилятором языка Cell. Для достижения этой цели компилятор решает следующие задачи:

- структурный дизайн ИА в ходе лексического и синтаксического разбора программы на языке Cell;
- семантический анализ построенной структуры ИА, выявление на его основе полного набора сигналов, состояний и переходов;
- сериализация функций переходов и синтез их процедурного кода.

Рассмотрим алгоритмы решения каждой из этих задач. Ряд положений статьи будет проиллюстрирован с использованием примера ИА ha (рис. 1).

Левая часть рисунка содержит диаграмму состояний и переходов автомата ha с графическим представлением его структуры. На правой части рисунка приведена




```
root{
  cell A, B;
  global {extern int e; }
  common {int k; }
  local {int i; }
  A {
    cell L, M;
    global {int aa; }
    common {int a; }
    local {int i; }
    on c1 @{x(); aa++; $a=@i++; @jump(B); @}
    L {
      global {int ll; }
      common {int l; }
      on c11 @{x1(); ll++; $l=@i++; @jump(M); @}
   }//L
   M{
     on c12 @{y1(); @jump(B); @}
   }//M
 }//A
 B{
  on c2 @{y(); @jump(B); @}
  on c3 @{z(); @jump(A); @}
 }//B
 } //root
```

Рис. 1 Пример ИА ha

полная спецификация автомата ha на языке Cell, включая детальное описание переходов автомата и его переменных.

Диаграмма автомата ha описывает ИА с тремя подавтоматами A^0, A^1, A^2 и одной цепью порождения: подавтомат A^2 порождается состоянием (ячейкой) A подавтомата A^1, который, в свою очередь, порождается корневой ячейкой root подавтомата A^0. Толщиной линий выделены стартовые состояния. Обычные стрелки обозначают переходы. Пунктирные стрелки обозначают операции порождения.

Подавтомат A^2 включает два состояния $\{L, M\}$ и два перехода $L \to M$, $M \to B$. Переход $L \to M$ открывается сигналом c11 и выполняет реакцию $x1$. Переход $M \to B$ открывается сигналом c12 и выполняет реакцию $y1$.

Подавтомат A^1 включает два состояния $\{A, B\}$ и три перехода $A \to B$, $B \to B$, $B \to A$. Переход $A \to B$ открывается сигналом c1 и выполняет реакцию x. Переход $B \to B$ открывается сигналом c2 и выполняет реакцию y. Переход $B \to A$ открывается сигналом c3 и выполняет реакцию z.

Корневой подавтомат A^0 включает единственную корневую ячейку root, порождающую иерархию автомата.

3.1 Структурный дизайн

Целью структурного дизайна является построение дерева ячеек ИА, специфицированного программой на языке Cell в ходе ее лексического и синтаксического анализа. Базовым элементом (узлом) этого дерева является структура ячейки, которая объединяет описания элементов, составляющих понятие ячейки языка Cell (рис. 2).

Основными элементами ячейки, необходимыми для дальнейшего семантического анализа и генерации процедурного кода, являются ее идентификаторы (имя, уровень иерархии), указатели на другие ячейки иерархии (предка, потомка, соседа), указатели на переменные различных классов видимости (см. ниже описание семантического анализа), список реакций ячейки.

Особенностью лексического и синтаксического анализа описания ИА на языке Cell является ограниченный синтаксический разбор кода функции перехода при формировании списка реакций ячейки в составе ее структуры.

Как уже отмечалось, основу для программирования алгоритмов переходов ИА на языке Cell составляет какой-либо существующий процедурный язык, синтаксически расширенный директивами языка Cell. Компилятор языка Cell в ходе структурного дизайна проводит частичный лексический и синтаксический разбор процедурного кода функции перехода, в результате которого выявляются директивы языка Cell, используемые в этом переходе. Далее только эти выявленные участки процедурного кода функции перехода будут анализироваться и модифицироваться компилятором языка Cell.

```
struct Cell {
  char *cell_name;                /* указатель на строку с именем ячейки */
  int cell_layer;                 /* номер уровня иерархии */
  union {
      struct Cell *u_parent;      /* указатель на родительскую ячейку для
                                     порожденных ячеек */
#define cell_parent     (u.u_parent)
      struct ContextParam *u_context; /* указатель на структуру параметров
                                         контекста иерархии для корневой ячейки */
#define cell_context    (u.u_context)
   } u;
 struct Cell *cell_child;         /* указатель на стартовую ячейку
                                     вложенного подавтомата */
 struct Cell *cell_next;          /* указатель на список ячеек текущего
                                     подавтомата */
 char *cell_global;               /* указатель на строку с описанием
                                     глобальных переменных автомата */
 char *cell_common;               /* указатель на строку с описанием
                                     общих переменных иерархии */
 char *cell_local;                /* указатель на строку с описанием
                                     локальных переменных ячейки */
 struct Transition *cell_trans;   /* указатель на список реакций ячейки */
}
```

Рис. 2 Основные элементы структуры ячейки

В ходе частичного лексического и синтаксического разбора процедурного кода функции перехода создается список процедурных блоков, структура каждого из которых содержит описание выявленного фрагмента (рис. 3).

Описание фрагмента кода отличается по типу кода, который он содержит. Для фрагмента кода на языке Си описание содержит символьную строку с кодом, которая без изменения будет перенесена в функцию перехода, генерируемую на заключительном этапе компиляции. Для функциональных директив языка Cell, связанных с отправкой сигналов, порождением клонов и переходом в новое состояние, структура фрагмента кода содержит параметры, необходимые для дальнейшей семантической обработки и генерации служебного кода. Аналогичный подход принят для описания директив языка Cell, связанных с доступом к переменным автомата определенного класса видимости.

Список разобранных процедурных блоков (фрагментов) собирается в описание реакции, структура которой помимо этого списка содержит список обрабатываемых сигналов (рис. 4). Организованные в список структуры описания реакций одной ячейки включаются в структуру ячейки (см. рис. 2).

В результате лексического и синтаксического анализа программы на языке Cell на шаге структурного дизайна компилятором создается древовидная структура связанных между собой ячеек, сигналов и реакций, используемая на следующих шагах компиляции. Будем называть полученную структуру деревом ячеек компилятора Cell. Рисунок 5 иллюстрирует дерево ячеек компилятора

```
struct Fragment {
  int fr_type;                    /* тип фрагмента кода */
#define frType_C        1         /* для фрагмента на языке C */
#define frType_Jump     2         /* для директивы @jump */
#define frType_Send     3         /* для директивы @send */
#define frType_Clone        4     /* для директивы @clone */
#define frType_Local        5     /* для директивы @variable с коротким именем */
#define frType_LocalFull    6     /* для директивы @variable с полным именем */
#define frType_Common       7     /* для директивы $variable */
  struct Transition *fr_trans;    /* указатель на реакцию */
  struct Frament *fr_next;        /* указатель на следующий фрагмент */
  union {
       char *u_C;               /* указатель на строку с фрагментом C-кода */
#define fr_C            (u.u_C)
       char *u_Jump;            /* указатель на имя целевой ячейки перехода */
#define fr_Jump         (u.u_Jump)
       struct {                 /* описание директивы @send */
           char *s_Signal  ;    /* указатель на имя исходящего сигнала */
           char *s_Target;      /* указатель на имя корневой ячейки
                                   получателя сигнала */
           int *s_Index;        /* индекс корневой ячейки получателя сигнала */
           char *s_Arguments;   /* указатель на строку аргументов сигнала */
#define fr_Signal       (u.s3.s_Signal)
#define fr_Target       (u.s3.s_Target)
#define fr_Index        (u.s3.s_Index)
#define fr_Arguments    (u.s3.s_Arguments)
       } s3;
       char *u_Clone   ;        /* указатель на имя корневой ячейки клона */
#define fr_Clone        (u.u_Clone)
       char *u_Local;           /* указатель на имя локальной
                                   переменной ячейки */
#define fr_Local        (u.u_Local)
       char *u_Common;          /* указатель на имя глобальной переменной
                                   дерева*/
#define fr_Common      (u.u_Common)
  } u
}
```

Рис. 3 Основные элементы структуры процедурного блока

```
struct Transition {               /* список реакций ячейки */
  struct Transition *tr_cell;     /* указатель на ячейку этой реакции */
  struct Transition *tr_next;     /* указатель на следующую реакцию ячейки */
  struct SignalList *tr_slist;    /* указатель на список открывающих переход
                                     сигналов */
  struct Fragment *tr_flist;      /* указатель на список фрагментов перехода */
}
struct SignalList {               /* список входящих сигналов */
  struct SignalList *sl_next;     /* указатель на следующий сигнал списка */
  char *sl_Signal;                /* указатель на имя сигнала */
}
```

Рис. 4 Основные элементы списков реакций ячейки и списков сигналов

Рис. 5 Дерево ячеек автомата ha

Cell для иерархического автомата ha (см. рис. 1). Структура ContextParam, входящая в дерево ячеек и описывающая параметры контекста иерархии, будет рассмотрена в следующем разделе.

3.2 Семантический анализ

Семантический анализ дерева ячеек, полученного в ходе структурного дизайна, решает следующие задачи компиляции программы на языке Cell:

- связывание именованных объектов языка Cell (ячеек, сигналов, переменных) в соответствии с правилами видимости и классами видимости, размещение их в требуемых областях видимости;
- вычисление множества сигналов, определение их параметров, в том числе классификация их по назначению (внешние/внутренние, входящие/исходящие и пр.).

Напомним, что в языке Cell различают диспетчерские (корневые), встроенные и внешние (типизированные) ячейки.

Диспетчерская ячейка является корнем иерархии, декларирует область памяти, связанную с иерархией (контекст иерархии), определяет процедуры извлечения и буферизации внешних сигналов, а также процедуры доставки внутренних сигналов, что образует подсистему ввода–вывода. Иерархии, задаваемые корневыми ячейками, взаимодействуют между собой через процедуры ввода-вывода, используя сигнальный механизм языка Cell. Помимо описания основного ИА диспетчерские ячейки применяются для спецификации клонов [12].

Встроенные ячейки размещаются по месту своего определения (декларации), и их расположение в иерархии определено однозначно.

Внешние ячейки декларируются вне иерархии. Ссылки на внешние ячейки могут размещаться в различных частях нескольких иерархий. При этом появляется возможность создавать множественные экземпляры однотипных ячеек, работающих независимо друг от друга. В ходе связывания именованных объектов языка Cell во время семантического анализа внешние именованные ячейки помещаются в дерево ячеек по местам размещения ссылок на них.

В ходе семантического анализа для каждой корневой (диспетчерской) ячейки создается внутренняя структура компилятора с параметрами контекста иерархии (рис. 6). Контекст иерархии описывает состояние иерархического автомата (вектор переходов), а также содержит глобальные и локальные переменные, используемые в реакциях автомата. Необходимая для генерации контекста иерархии информация содержится как в дереве ячеек, так и в структуре параметров контекста иерархии, созданной после семантического анализа этого дерева. Основным элементом структуры параметров контекста иерархии, вычисляемым в ходе семантического анализа, является список параметров сигналов, который служит основой для генерации вектора состояния автомата. Вопросы, связанные

```
struct ContextParam {              /* параметры контекста иерархии */
   struct Cell *ctx_cell;          /* указатель на корневую ячейку иерархии */
   int ctx_Size;                   /* максимальное число экземпляров иерархии */
   struct SigParamList *ctx_slist; /* указатель на список сигналов
                                      и их параметров */
}
struct SigParamList {              /* список сигналов и их параметров */
   struct SigParamList *spl_next;  /* указатель на следующий сигнал списка */
   char *spl_Signal;               /* указатель на имя сигнала */
   int spl_Size;                   /* размер стека для передачи сигнальных данных */
   int spl_Type;                   /* тип сигнала */
\#define slpType_input       1     /* бит установлен - есть переход
                                      обработки сигнала */
\#define slpType_output      2     /* бит установлен - есть неадресная отправка
                                      сигнала */
\#define slpType_internal    3     /* внутренний сигнал = slpType_input +
                                      + slpType_output */
\#define slpType_clone       4     /* бит установлен - есть адресная отправка
                                      сигнала */
}
```

Рис. 6 Структура параметров контекста иерархии

со структурой контекста иерархии и ее генерацией в исполняемом коде, будут рассмотрены в следующем разделе.

Определение параметров для распределения памяти автомата с учетом классов видимости данных — следующая задача семантического анализа. В языке Cell доступность (видимость) данных в различных ячейках (точнее — в переходах соответствующих ячеек) характеризуется классами видимости. Часть классов видимости определяют области памяти, доступные лишь единственной ячейке или переходу, другие — области разделяемой памяти, доступные многим ячейкам. При этом блокирование данных от множественного доступа не выполняется, поскольку процедура сериализации переходов допускает исполнение процедурного кода только одного перехода в моменты, когда автомат находится в определенном состоянии и, следовательно, все его переменные имеют значения, обусловленные алгоритмом. В языке Cell применяются пять классов видимости. Данные трех из них обрабатываются в ходе семантического анализа. Рассмотрим классы видимости и связанную с ними семантическую обработку подробнее.

1. Диспетчерский класс (декларации корневой ячейки). Входящие в него данные могут быть доступны либо всем иерархиям (в терминах процедурного языка — глобальные данные), либо только переходам одной иерархии. В последнем случае они разделяются всеми экземплярами иерархии и соответствуют статическим данным процедурного языка. Данные этого класса именуются и декларируются по правилам процедурного языка в декларативных директивах языка Cell с именем global, их именование при использовании в командах функций переходов также соответствует правилам процедурного

языка. Компилятор языка Cell не контролирует использование данных этого класса.

2. Общую область (общий класс) образуют глобальные данные, доступные по короткому имени в пределах функций переходов одного экземпляра иерархии. Они декларируются и именуются по правилам процедурного языка в декларативных директивах языка Cell с именем common. Декларации этого класса образуют неструктурированный набор именованных данных, доступный всем ячейкам одного экземпляра иерархии. Повторные декларации имен в общей области недопустимы. При использовании этих данных в командах перехода к их имени добавляется префикс "$". Компилятор языка Cell выполняет преобразование таких имен для доступа к требуемой области памяти.
3. Локальную область данных (локальный класс) образуют данные, продекларированные в родительской ячейке и доступные по короткому имени переходам порожденных ячеек, если путь порождения не содержит других ячеек с локальными декларациями. Переходы любой порожденной ячейки имеют доступ по короткому имени к единственному набору локальных данных родительской ячейки в пределах одного экземпляра иерархии. Локальные данные декларируются и именуются по правилам процедурного языка в декларативных директивах языка Cell с именем local. При использовании этих данных в командах перехода к их имени добавляется префикс "@". Компилятор языка Cell выполняет преобразование таких имен для доступа к требуемой области памяти. В пределах одного экземпляра иерархии функции перехода могут обращаться к локальным данным не только родительской ячейки. В этом случае используется полное имя переменной, включающее иерархическое имя декларирующей ячейки.
4. Автоматический класс образуют данные, размещаемые на процедурном стеке перехода. Время их жизни ограничено продолжительностью исполнения перехода, видимость — процедурным переходом, в котором они декларированы. Этот класс соответствует автоматическим данным процедурного языка. В отличие от предыдущих классов автоматические данные не являются разделяемыми и доступны только продекларировавшему их переходу. Компилятор языка Cell не контролирует использование этих данных.
5. Программный стек. На программном стеке размещаются служебные данные, связанные с исходящими сигналами и ассоциированными с ними блоками данных. Следует помнить, что механизм сериализации системы Cell задерживает передачу сигналов до завершения тела перехода. Поэтому доставка сигналов, сформированных переходом, начинается уже после уничтожения его автоматических данных. Компилятор языка Cell для каждого сигнала вычисляет максимальный размер памяти, необходимый для передачи связанных с ним данных, с тем, чтобы зарезервировать требуемую память в

очереди сигналов. Заметим, что явное размещение данных, передаваемых с сигналом, на стеке в большинстве случаев не требуется: необходимые данные могут быть размещены в переменных разделяемых классов видимости (глобальном, общем, локальном). Процедура сериализации гарантирует, что в каждый момент времени будут обрабатываться только один входной сигнал и соответственно одна функция перехода. Соответственно, можно ограничиться единственным элементом данных, представляющим текущий протокольный блок данных. Этот экземпляр данных может использоваться всеми переходами ИА без избыточного копирования, «обрастая» по мере обработки декодированными полями.

3.3 Синтез процедурного кода

Синтез процедурного кода — завершающий этап компиляции программы на языке Cell. Исходными данными для этого этапа являются внутренние структуры компилятора, созданные на предыдущих этапах. Основные из них — дерево ячеек и структура параметров контекста иерархии. Используя содержащуюся в этих структурах информацию, кодогенератор компилятора для каждой иерархии генерирует отдельный программный модуль на процедурном языке Си.

Уникальность имен, порожденных кодогенератором в программном модуле, обеспечивается префиксом "_cell". Пользователю запрещается использовать на языке Cell идентификаторы, начинающиеся с этого префикса.

Созданный программный модуль содержит три раздела кода на языке Си:

(1) декларации данных диспетчерского класса видимости (статические и глобальные данные программного модуля на процедурном языке);

(2) декларация контекста иерархии;

(3) функции автомата.

Рассмотрим некоторые аспекты генерации каждого из этих разделов.

3.3.1 Генерация процедурного кода для раздела декларация данных диспетчерского класса видимости

Правила генерации кода для этого раздела наиболее просты. Основной код раздела, называемый «пользовательские данные», создается кодогенератором в соответствии с содержимым декларативных директив *global* языка Cell путем их объединения без какого-либо контроля. Контроль полученного кода осуществляет компилятор Си.

Дополнительно раздел декларации данных диспетчерского класса содержит служебные данные — глобальные векторы пользовательских функций инициализации подавтомата каждого уровня при его активизации и деактивации. Векторы доступны вне процедурного кода модуля и должны быть переопределены

```
/* модуль root.c - процедурный код для иерархии с корневой ячейкой root */

/* Раздел 1 - Декларации данных диспетчерского класса видимости */

#define \cell\root_lySize       3       /* число уровней иерархии автомата */
/* 1.1. Вектор инициализации при активизации подавтомата */
int (*_cell_root_init[ _cell_root_lySize])() ={0, 0, 0};
/* 1.2. Вектор инициализации при деактивации подавтомата */
void (*_cell_root_destroy[ _cell_root_lySize])() ={0, 0, 0};

/* 1.3. Пользовательские данные */
extern int e;          /* декларация ячейки root */
int aa;                /* декларация ячейки root.A */
int ll;                /* декларация ячейки root.A.L */

/* Раздел 2 - Декларация контекста иерархии root */
...
/* Раздел 3 - Функции автомата */
...
```

Рис. 7 Декларация данных диспетчерского класса видимости автомата ha

пользователем, если требуется выполнение пользовательского кода в указанных случаях. Формат и место вызова пользовательских функций рассматривается далее при описании процедур формирования кода функций автомата («Раздел 3 — Функции автомата»).

Рисунок 7 содержит пример декларации данных диспетчерского класса видимости для ИА ha (см. рис. 1).

3.3.2 Генерация процедурного кода для раздела декларация контекста иерархии

Напомним, что контекст иерархии — это память внутренних данных ИА, в которой содержатся служебные и пользовательские переменные, необходимые для функционирования автомата. Каждый экземпляр иерархии использует свой контекст. Поэтому для автоматов, допускающих функционирование нескольких экземпляров одной иерархии (клонов), справедливо говорить о таблице контекстов одной иерархии (рис. 8).

Контекст иерархии хранит данные четырех агрегированных переменных. Две из них включают служебные данные для организации вектора состояния экземпляра иерархии и его очереди исходящих сигналов, другие две — данные общего и локального классов видимости (пользовательские и служебные):

- вектор состояния;
- очередь отправляемых сигналов;
- данные общего класса видимости;
- данные локального класса видимости.

```
#define _cell_<имя>_ctxSize   1   /* число экземпляров контекста иерархии <имя> */
struct _cell_<имя>_context {
  /* ctx_signal  - 2.1. Вектор состояния экземпляра иерархии */
  /* ctx_queue   - 2.2. Очередь исходящих сигналов экземпляра иерархии */
  /* ctx_common  - 2.3. Агрегат данных общего класса видимости экземпляра иерархии */
  /* ctx_local   - 2.4. Агрегат данных общего класса видимости экземпляра иерархии */
} _cell_<имя>_context[_cell_<имя>_ctxSize];
```

Рис. 8 Таблица контекстов иерархии

Вектор состояния — это набор дескрипторов всех обслуживаемых в иерархии сигналов. Иерархия обрабатывает следующие виды сигналов:

- внешние входящие сигналы, приходящие из внешнего мира и открывающие явно специфицированные переходы ИА;
- внутренние входящие сигналы (они же внутренние исходящие сигналы, далее просто внутренние сигналы), сгенерированные в результате выполнения перехода автомата и открывающие явно специфицированные переходы этой же иерархии;
- исходящие сигналы для обработки в другой иерархии (исходящие сигналы клону), сгенерированные в результате выполнения перехода текущего автомата и открывающие явно специфицированные переходы другой иерархии, активированной оператором clone языка Cell (клонированная иерархия);
- исходящие сигналы для обработки во внешнем мире (внешние исходящие сигналы).

Особенности обработки каждого из этих типов сигналов будут рассмотрены далее совместно с описанием генерируемых для этого функций переходов («Раздел 3 — Функции автомата»). Здесь лишь отметим, что для обработки последних двух видов исходящие сигналов компилятор генерирует служебные внутренние сигналы иерархии (неявные сигналы) и их дескрипторы. Обработка неявных сигналов выполняется служебными переходами корневой ячейки, которые также генерируются компилятором языка Cell. В результате вектор состояния для всех (в том числе неявных) сигналов содержит адреса актуальных в настоящий момент времени обработчиков сигналов (функций переходов), указанных в дескрипторах сигнала. При завершении выполнения функции перехода содержимое набора дескрипторов сигналов меняется. В результате в каждый момент времени текущее содержимое вектора состояний описывает актуальное состояние ИА. Дескриптор сигнала (рис. 9) дополнительно к адресу актуального обработчика сигнала содержит номер уровня иерархии обработчика, который используется для деактивизации подавтоматов определенного уровня при возвратных межуровневых переходах.

Адрес дескриптора сигнала в каждый момент времени однозначно определяет обработчик сигнала и используется в качестве идентификатора сигнала при его

```
typedef struct _cell_Signal {    /* структура дескриптора сигнала */
  void (*sig_transition)();      /* адрес функции перехода обработки сигнала */
  int sig_layer;                 /* уровень иерархии обслуживания сигнала */
} _cell_Signal;
```

Рис. 9 Дескриптор сигнала

```
typedef struct _cell_sArg {    /* структура описания передаваемых данных */
  int     sa_size;             /* размер данных в байтах */
  char        *sa_args;        /* указатель на данные сигнала */
} _cell_sArg;

typedef struct _cell_qItem {   /* структура элемента очереди сообщений */
  _cell_Signal    *qi_signal;  /* идентификатор сигнала */
  _cell_sArg      *qi_args;    /* передаваемые данные */
  _cell_qItem *qi_prev;        /* предыдущий элемент очереди */
  _cell_qItem *qi_next;        /* следующий элемент очереди */
} _cell_qItem;

typedef struct _cell_Queue {   /* структура очереди */
  _cell_qItem * qu_iSignal;    /* первый сигнал в очереди */
  _cell_qItem * qu_cSignal;    /* начало очереди служебных сигналов */
  _cell_qItem * qu_oSignal;    /* конец очереди сигналов */
} _cell_Queue
```

Рис. 10 Очередь отправляемых сигналов

отправке в ходе выполнения функции перехода. Отправляемые сигналы накапливаются с очереди исходящих сигналов, сохраняемой контекстом экземпляра иерархии. Вместе с идентификатором отправляемого сигнала в очереди хранятся данные, передаваемые с сигналом (рис. 10).

Очередь организована таким образом, что обеспечивает соблюдение следующих принятых в языке Cell правил образования и обработки очереди сигналов, отправляемых в ходе выполнения перехода:

- внутренние сигналы от таймера накапливаются в начале очереди в порядке «последний пришел – первым обработан»;
- остальные внутренние сигналы накапливаются друг за другом в порядке «последний пришел – последним обработан» после сигналов от таймера;
- служебные сигналы, соответствующие внешним исходящим сигналам, накапливаются в конце очереди в порядке «последний пришел – последним обработан»;
- служебные сигналы, соответствующие исходящим сигналам клону, накапливаются друг за другом в порядке «последний пришел – последним обработан» после внутренних сигналов;
- все накопленные в очереди сигналы должны быть обработаны перед обработкой нового внешнего входящего сигнала.

```
/* модуль root.c - процедурный код для иерархии с корневой ячейкой root */
...
/* Раздел 2 - Декларация контекста иерархии root */

#define _cell_root_ctxSize     1     /* число экземпляров контекста
                                        иерархии root */
#define _cell_root_sigNumber   5     /* число входных сигналов иерархии root */

static int _cell_root_ctxIndex;     /* индекс текущего экземпляра иерархии */
#define _cell_root_ctx  _cell_root_context[0]
                                    /* _cell_root_context[_cell_root_ctxIndex] */

enum { _cell_root_s_c1, _cell_root_s_c2, _cell_root_s_c3, _cell_root_s_c11,
       _cell_root_s_c12 };

struct _cell_root_context {

/* 2.1. Вектор состояния экземпляра иерархии */
    _cell_Signal ctx_signal[_cell_root_sigNumber];
#define _cell_root_signal           _cell_root_ctx.ctx_signal
#define _cell_root_signal_c1        _cell_root_ signal[_cell_root_s_c1]
#define _cell_root_signal_c2        _cell_root_ signal[_cell_root_s_c2]
#define _cell_root_signal_c3        _cell_root_ signal[_cell_root_s_c3]
#define _cell_root_signal_c11       _cell_root_ signal[_cell_root_s_c11]
#define _cell_root_signal_c12       _cell_root_ signal[_cell_root_s_c12]

/* 2.2. Очередь исходящих сигналов экземпляра иерархии */
    _cell_Queue     ctx_queue;
#define _cell_root_queue        _cell_root_ctx.ctx_queue

/* 2.3. Данные общего класса видимости */
    struct {
      int k;          /* декларация ячейки root */
      int a;          /* декларация ячейки root.A */
      int l;          /* декларация ячейки root.A.L */
    } ctx_common;
#define _cell_root_common   _cell_root_ctx.ctx_common

    int ctx_isActiv[_cell_root_lySize]; /* вектор активности уровней иерархии */
#define _cell_root_isActiv      _cell_root_ctx.ctx_isActiv

/* 2.4. Данные локального класса видимости */
    struct {
        struct {                        /* декларация ячейки root */
            int i;
            struct {                    /* декларация ячейки root.A */
                int i;
            } A;
        } root;
    }   ctx_local;
#define _cell_root_local                _cell_root_ctx.ctx_local

} _cell_root_context[_cell_root_ctxSize];
...
```

Рис. 11 Контекст иерархии автомата ha

Рисунок 11 иллюстрирует на примере ИА ha (см. рис. 1) содержимое структуры контекста иерархии. Данные общего и локального класса видимости собираются и структурируются соответствующим образом с использованием директив common и local языка Cell. К данным общего класса видимости иерархического автомата компилятор добавляет служебную переменную _cell_isActiv, которая индицирует активность подавтомата соответствующего уровня иерархии.

3.3.3 Генерация процедурного кода для раздела с функциями автомата

Это наиболее емкий раздел сгенерированного кода. Он включает в себя несколько подразделов:

- функции активизации подавтоматов и экземпляра иерархии;
- функция деактивации подавтоматов;
- функция обработки внешнего входящего сигнала;
- функции обработки служебных сигналов;
- функции явно специфицированных переходов.

Для корневой и всех стартовых ячеек ИА генерируются функции активизации подавтомата соответствующего уровня иерархии. Процедуры активизации подавтомата нулевого уровня иерархии (корневой автомат) и остальных подавтоматов отличаются.

Рассмотрим процедуры активизации подавтоматов ненулевого уровня иерархии. Они недоступны вне функций автомата. В процедурах активизации используются данные структуры контекста текущего экземпляра иерархии. Процедура активизации выполняется, если подавтомат неактивен. Попытка активизировать активный подавтомат игнорируется. В процессе активизации подавтомата ненулевого уровня иерархии вызывается пользовательская функция инициализации (если она определена в соответствующем глобальном векторе инициализации диспетчерского класса видимости), в вектор состояния экземпляра иерархии добавляются реакции для его стартового состояния, после чего подавтомат считается активизированным.

Рисунок 12 содержит функции активизации подавтоматов первого и второго уровней иерархии автомата ha (см. рис. 1). Поскольку стартовое состояние подавтомата первого уровня иерархии автомата ha порождает подавтомат второго уровня иерархии, то активизация ячейки *A* сопровождается активизаций подавтомата второго уровня иерархии.

Активизация корневого подавтомата означает активизацию иерархического автомата (экземпляра иерархии). Эта функция является глобальной и доступна внутри и вне процедурного модуля иерархии. Функция пытается определить свободный экземпляр контекста иерархии, очищает его и инициализирует, в случае если задана пользовательская функция инициализации в соответствующем

```
/* модуль root.c - процедурный код для иерархии с корневой ячейкой root */
...
/* Раздел 3 - Функции автомата */

/* 3.1. Функции активизации подавтоматов ненулевого уровня иерархии */
static void _cell_root_Activ1() {
  if(_cell_root_ isActiv[1]) return;
  if(_cell_root_init[1] && !(*_cell_root_init[1])(&_cell_root_ctx)) return;
  _cell_root_signal_c1.sig_transition = _cell_root_A_2_root_B_on_c1;
  _cell_root_signal_c1.sig_layer=1;
  _cell_root_ isActiv[1]=1;
  _cell_root_Activ2();
}
static void _cell_root_Activ2() {
  if(_cell_root_ isActiv[2]) return;
  if(_cell_root_init[2] && !(*_cell_root_init[2])(&_cell_root_ctx)) return;
  _cell_root_signal_c11.sig_transition = _cell_root_A_L_2_root_A_M_on_c11;
  _cell_root_signal_c11.sig_layer=2;
  _cell_root_ isActiv[2]=1;
}
```

Рис. 12 Функции активации подавтоматов автомата ha

```
/* модуль root.c - процедурный код для иерархии с корневой ячейкой root */
...
/* Раздел 3 - Функции автомата */
...
/* 3.2. Функция активации экземпляра иерархии */
int _cell_root_Activ(void) {
    for(_cell_root_ctxIndex=0; _cell_root_ctxIndex <_cell_root_ctxSize;
    _cell_root_ctxIndex++) {
      if(!_cell_root_ isActiv[0]) {
         memset(&_cell_root_ctx, sizeof(_cell_root_context),0); /* сброс контекста */
         if(_cell_root_init[0] && !(*_cell_root_init[0])(&_cell_root_ctx)) return -1;
         /* служебных сигналов нет, реакции на них не инициализируются */
         _cell_root_ isActiv[0]=1;
         _cell_root_Activ1();
         return _cell_root_ctxIndex;
    } }
    return -1;
}
```

Рис. 13 Функции активации экземпляра иерархии автомата ha

глобальном векторе диспетчерского класса видимости. После этого в подготовленном контексте экземпляра иерархии задаются реакции на служебные сигналы и активизируется подавтомат первого уровня иерархии. В случае успеха инициатор активизации иерархии (например, с помощью директивы clone языка Cell) получает номер контекста активизированного экземпляра иерархии, в противном случае — значение «-1».

Рисунок 13 демонстрирует процедурный код активизации иерархии автомата ha (см. рис. 1).

Для деактивации подавтомата (или цепи подавтоматов) определенного уровня иерархии при межуровневых возвратных переходах (или иных случаях функционирования иерархического автомата) выполняется функция деактивации цепи порожденных подавтоматов. Функция деактивации цепи подавтоматов служебная и доступна внутри процедурного модуля иерархии. Предполагается, что она используется из сгенерированного компилятором кода, поэтому достоверность упорядоченной цепи порождения подавтоматов в этой функции не проверяется. Глобальной функцией является функция деактивации экземпляра иерархии, которая доступна внутри и вне процедурного модуля иерархии. Функция деактивации цепи подавтоматов предусматривает отмену в векторе состояний текущего экземпляра иерархии реакций, связанных с цепью подавтоматов. Процедура деактивации иерархии выполняет в контексте заданного экземпляра иерархии очистку очереди сигналов и всех данных самого контекста. Рисунок 14 содержит упрощенные алгоритмы функций деактивации подавтоматов автомата ha (см. рис. 1).

Внешние входные сигналы приходят в автомат из внешнего мира и формируются специальным образом вне автомата, например обработчиком аппаратного прерывания или процедурой чтения входного потока. Обработка внешнего входного сигнала начинается в автомате выполнением глобальной функцией, доступ-

```
/* модуль root.c - процедурный код для иерархии с корневой ячейкой root */
...
/* Раздел 3 - Функции автомата */
...
/* 3.3. Функция деактивации экземпляра иерархии */
void _cell_root_deActiv0(int i) {
    if( i != -1) _cell_root_ctxIndex=i;
    if(_cell_root_destroy[0]) (*_cell_root_destroy[0])(&_cell_root_ctx);
    clearQueue();
    memset(_cell_root_context[_cell_root_ctxIndex],sizeof(_cell_root_context),0);
}

/* 3.4. Функция деактивации подавтоматов цепи порождения */
static void _cell_root_deActiv(<возрастающая цепь порождения>) {
    if( <минимальный номер уровня иерархии>  < 1) {
         _cell_root_deActiv0(-1);
         return;
    }
    for(l=0; l <n; l++) {
        int layer = <возрастающая цепь порождения>[l];
        if(_cell_root_destroy[layer]) (*_cell_root_destroy[layer])(&_cell_root_ctx);
        _cell_root_isActiv[layer]=0;
        for (int i=0; i < _cell_root_sigNumber; i++) {
            if(_cell_root_ signal[i].sig_layer==layer) {
                memset(&_cell_root_ signal[i], sizeof(_cell_Signal), 0);
}   }   }   }
```

Рис. 14 Упрощенные функции деактивации подавтоматов автомата ha

```
/* модуль root.c - процедурный код для иерархии с корневой ячейкой root */
...
/* Раздел 3 - Функции автомата */
...
/* 3.5. Функция обработки внешнего входящего сигнала */
_cell_root_TransitionOnSignal(int Index, _cell_Signal *input, <данные сигнала>) {
    if(input && input->sig_transition) {
        _cell_root_ctxIndex=Index;
        (*input->sig_trasition)( <данные сигнала>);
        while (_cell_qItem *qi=getFirst(&_root_cell_queue)) {
            void (*transition)()= qi->qi_signal->sig_trasition;
            if(trasition) (*trasition)(<данные сигнала из qi->qi_args>);
            freeItem(qi);
}   }   }
```

Рис. 15 Функция обработки внешнего входящего сигнала автомата ha

ной вне процедурного кода модуля иерархии. Функция вызывается инициатором обработки сигнала, задавшим идентификатор сигнала, данные, передаваемые с сигналом, и контекст иерархии. Автомат выполняет переход, связанный с полученным сигналом и далее обслуживает внутренние сигналы, сгенерированные этим переходом. Возврат из функции выполняется после обработки всех внутренних сигналов. Рисунок 15 описывает алгоритм обработки внешних сигналов на примере автомата ha (см. рис. 1).

Рисунок 16 представляет правила обработки служебных сигналов, отправляемых либо в другой ИА (например, автомат-клон), либо во внешний мир.

Отправка сигнала в автомат-клон означает передачу ему входящего сигнала, для обслуживания которого процедурный модуль автомата-клона содержит синтезированную функцию обработки внешних входных сигналов, алгоритм работы которой обсуждался ранее (см. рис. 15).

Для обслуживания исходящих сигналов во внешний мир компилятор языка Cell генерирует только заголовки функций, которые вызываются при функционировании автомата. Содержание этих функций должно быть определено разработчиком автомата самостоятельно с учетом особенностей окружения.

И, наконец, последними в процедурном модуле генерируются функции явно описанных на языке Cell переходов. Особенности их работы уже обсуждались в предыдущих работах, посвященных языку Cell, при рассмотрении модели исполнения автомата в этой системе программирования [10–12]. Напомним, что эти функции синтезируются компилятором рассматриваемой системы программирования с использованием кода перехода, написанного на процедурном языке, в спецификации автомата на языке Cell.

Для каждого специфицированного перехода компилятор языка Cell создает функцию, именуемую по клише «полное имя текущего состояния – имя открывающего сигнала». Функция содержит модифицированный процедурный код, специфицирующий переход на языке Cell. Модификация процедурного кода

```
/* модуль root.c --- процедурный код для иерархии с корневой ячейкой root */
...
/* Раздел 3 --- Функции автомата */
...
/* 3.6. Функции обработки служебных сигналов для отсылки клону < имя-корня-клона >
сигнал <сигнал> клону <корень-клона> экземпляр  iCtx идет в очередь как
служебный сигнал _cell_root_signal_clone_<корень-клона>
обработчику, определенному в корне текущей иерархии с именем функции клона,
_cell_<корень-клона>_TransitionOnSignal(int, _cell\_Signal *, <данные сигналу клона>)
данными, передаваемыми со служебным сигналом в функцию клона
iCtx,
&_cell_< корень-клона>_context[iCtx].ctx_ signal[_cell_< корень-клона>_s_ <сигнал>],
 <данные сигналу клона>
*/

/* 3.7. Функции обработки сигналов во внешний мир
сигнал <сигнал> идет в очередь как служебный сигнал
_cell_root_signal_output_<сигнал>
обработчику, определенному в корне текущей иерархии с именем
_cell_root\_Output_<сигнал> (<данные сигнала>)
данные, передаваемые со служебным сигналом в функцию
<данные сигнала >
обработчик реализуется пользователем
*/
```

Рис. 16 Правила обработки служебных сигналов

заключается в компиляции директив языка Cell, используемых в процедурном коде, и синтезе соответствующего служебного процедурного кода. В коде перехода могут присутствовать директивы языка Cell двух типов:

тип 1 — функциональные директивы для отправки сигналов (директива @send), порождения клона (директива @clone), перехода в новое состояние (директива @jump);

тип 2 — директивы доступа к переменным общего и локального класса видимости — операторы «$» и «@» соответственно.

По директиве отправки сигнала генерируется вызов функции для постановки исходящего сигнала в соответствующее место внутренней очереди сигналов, содержащейся в контексте текущей иерархии, что обсуждалось ранее.

По директиве порождения клона генерируется вызов функции для активизации нового экземпляра иерархии с указанной корневой ячейкой (функции активизации иерархии также обсуждались ранее).

По директиве перехода в новое состояние генерируется процедурный код, изменяющий содержимое вектора состояния иерархии (текущее содержимое дескрипторов сигналов). Этот код выполняет очистку дескрипторов сигналов, обрабатываемых в текущем состоянии, и установку дескрипторов сигналов, обрабатываемых в новом состоянии, указанном в директиве @jump. При этом выявляются и обрабатываются попытки одновременной установки нескольких

реакций на единственный сигнал. Если переход является межуровневым, то в служебный код дополнительно добавляются функции по деактивации промежуточных автоматов.

```
/* модуль root.c --- процедурный код для иерархии с корневой ячейкой root */
...
/* Раздел 3 --- Функции автомата */
...
/* 3.8. Функции обработки явно специфицированных переходов */
#define _cell_check1 (a, b)     if(b/**/.sig_transition)
{ _cell_exeption(ONE_MORE, ‘‘a’’, ‘‘b’’); }

_cell_root_A_on_c1 {
    x(); aa++; _cell_root_common.a=_cell_root_local.root.i++;
    memset(&_cell_root_signal_c1, sizeof(_cell_Signal), 0);
    _cell_check1(_cell_root_A_on_c1, _cell_root_signal_c2);
    _cell_root_signal_c2.sig_transition = _cell_root_B_on_c2;
    _cell_root_signal_c2.sig_layer=1;
    _cell_check1(_cell_root_A_on_c1, _cell_root_signal_c3);
    _cell_root_signal_c3.sig_transition = _cell_root_B_2_root_A_on_c3;
    _cell_root_signal_c3.sig_layer=1;
}
_cell_root_A_L_on_c11 {
    x1(); ll++; _cell_root_common.l=_cell_root_local.root.A.i++;
    memset(&_cell_root_signal_c11, sizeof(_cell_Signal), 0);
    _cell_check1(_cell_root_A_L_on_c11, _cell_root_signal_c12);
    _cell_root_signal_c12.sig_transition = _cell_root_A_M_on_c12;
    _cell_root_signal_c12.sig_layer=2;
}
_cell_root_A_M_on_c12 {
    y1();
    _cell_root_deActiv(2);
    for (int i=0; i <_cell_root_sigNumber;  i++) {
        if(_cell_root_ signal[i].sig_layer==1) {
            memset(&_cell_root\_ signal[i], sizeof(_cell_Signal), 0);
    }   }
    _cell_check1(_cell_root_A_M_on_c12, _cell_root_signal_c2);
    _cell_root_signal_c2.sig_transition = _cell_root_B_on_c2;
    _cell_root_signal_c2.sig_layer=1;
    _cell_check1(_cell_root_A_M_on_c12, _cell_root_signal_c3);
    _cell_root_signal_c3.sig_transition = _cell_root_B_on_c3;
    _cell_root_signal_c3.sig_layer=1;
}
_cell_root_B_on_c2 {
    y();
}
_cell_root_B_on_c3 {
    z();
    memset(&_cell_root_signal_c2, sizeof(_cell_Signal), 0);
    memset(&_cell_root_signal_c3, sizeof(_cell_Signal), 0);
    _cell_check1(_cell_root_B_on_c3, _cell_root_signal_c1);
    _cell_root_signal_c1.sig_transition = _cell_root_A_on_c1;
    _cell_root_signal_c1.sig_layer=1;
}
```

Рис. 17 Функции явно специфицированных переходов автомата ha

По директиве доступа к переменной общего или локального класса видимости генерируется полное имя переменной, что обеспечивает доступ к необходимой области памяти.

Рисунок 17 демонстрирует алгоритмы синтезированных функций для явно специфицированных переходов на примере ИА ha (см. рис. 1).

4 Заключение

Разработанные и рассмотренные в статье алгоритмы функционирования компилятора языка Cell делают возможным выполнение практической части проекта — программирование компилятора, следующего шага на пути создания системы программирования Cell для решения трудоемкой задачи автоматизированного синтеза программного обеспечения многоуровневого телекоммуникационного протокола.

Разработанные ранее модели ИА системы программирования Cell, а также подходы к исполнению ИА в целевом вычислителе, реализуемые с помощью алгоритмически спроектированного компилятора языка Cell, после создания компилятора позволят авторам проверить предлагаемые решения на практике и подтвердить основные достоинства системы программирования Cell:

- создание взаимоувязанной формальной спецификации ИА по фрагментам формализованного описания протокола;
- сериализацию вычислений — выстраивание последовательной логики выполнения переходов автомата на этапе компиляции исполняемого кода, что позволяет исполнять увязанные автоматы разной иерархии без использования методологии и технологии организации параллельных вычислений.

Литература

1. ITU-T Recommendation Z.100: Specification and description language (SDL). 1999. 244 p.
2. ITU-T Recommendation Z.120: Message Sequence Char (MSC). 1999. 136 p.
3. *Boussinot F.* Reactive C: An extension of C to program reactive systems // Software Practice Experience, 1991. Vol. 21. No. 4. P. 401–428.
4. *Lucas P. J.* An object-oriented language system for implementing concurrent, hierarchical, finite state machines. — University of Illinois at Urbana–Champagn, 1993. 86 p. http://chsm.sourceforge.net.
5. *Поликарпова Н. И., Шалыто А. А.* Автоматное программирование. — СПб.: Питер, 2008. 176 с.
6. *Terekhov A., Sokolov V.* Implementation of the conformation of MSC and SDL diagrams in the REAL technology // Programming Computer Software, 2007. Vol. 33. No. 1. P. 24–33.

7. *Поздняков В. А.* Особенности языка программирования, предназначенного для описания модулей в рамках универсальной системы обработки потоков данных // Наука и Образование: электронное научно-техническое издание, 2008. № 12. http://technomag.edu.ru.
8. *Бондаренко Т. В.* Язык описания иерархического автомата для генерации программного обеспечения телекоммуникационного протокола // Приложение к журналу «Открытое образование», 2010. № 6. Мат-лы XXXVII междунар. конф. «Информационные технологии в науке, социологии, экономике и бизнесе» (IT + S&E'10). — Ялта–Гурзуф, Украина, 2010. С. 21–22.
9. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Сигнальный механизм языка Cell // Системы и средства информатики. — М.: ИПИРАН, 2010. Вып. 20. № 3. С. 45–66.
10. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Базовая модель функционирования автомата в системе программирования Cell // Системы и средства информатики. — М.: ИПИРАН, 2010. Вып. 20. № 3. С. 82–97.
11. *Бондаренко Т. В., Волович К. И., Кондрашев В. А.* Язык Cell — инструмент для синтеза программного обеспечения многоуровневого телекоммуникационного протокола по частично формализованным спецификациям // Вестник Воронежского Государственного ун-та, 2011. Т. 7. № 3. С. 120–125.
12. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Язык Cell: модель обработки клонов // Системы и средства информатики. — М.: ИПИРАН, 2010. Вып. 20. № 3. С. 67–81.

СИСТЕМЫ И СРЕДСТВА ИНФОРМАТИКИ, 2011. Вып. 21. № 1. С. 141–151

ОСОБЕННОСТИ РЕАЛИЗАЦИИ УСТРОЙСТВ ИЗМЕРЕНИЯ ВРЕМЕНИ В ВИРТУАЛЬНЫХ МАШИНАХ

***В. Ю. Егоров*[1], *М. А. Шпадырев*[2]**

Аннотация: Описаны устройства измерения времени, входящие в состав персонального компьютера, а также проблемы разработки аналогичных устройств в виртуальных машинах (ВМ) и способы их решения.

Ключевые слова: устройства измерения времени; виртуальная машина; гипервизор; гостевая операционная система

1 Введение

В состав современного персонального компьютера входят несколько аппаратных устройств измерения времени, без которых практически невозможна работа любого установленного на нем системного программного обеспечения. Все операционные системы (ОС) так или иначе используют периодические сигналы (аппаратные прерывания) от этих устройств для выполнения каких-то служебных действий: например, подсчет прошедшего времени, планирование процессов и нитей в многозадачных ОС и т. д. [1]. Таким образом, работа гостевых ОС в ВМ без виртуальных аналогов подобных устройств также невозможна.

Правильная программная реализация виртуальных устройств (ВУ) измерения времени (ВУИВ) для разработчиков ВУ представляет собой довольно сложную задачу. Причина кроется в том, что в отличие от других ВУ, для которых достаточно повторить логику работы соответствующего реального аппаратного устройства, для ВУИВ необходимо обеспечить жесткие рамки временны́х отрезков между определенными событиями внутри устройства, а это зачастую оказывается невозможным. Для своей работы они используют аппаратные средства измерения времени хоста (компьютер, на котором запускается и работает ВМ). Другим способом реализовать свою функциональность невозможно, поэтому очевидно, что из-за накладных расходов поведение ВУИМ и реальных физических устройств измерения времени будет различаться. Эти различия могут быть причиной некоторых неточностей в работе ВУИМ, что негативным образом сказывается на стабильности и корректности работы гостевого программного обеспечения (работающего внутри ВМ).

[1]Пензенский государственный университет, vec@mail.ru
[2]Пензенский государственный университет, lordm@nm.ru

В данной статье рассматриваются возможные пути решения подобных проблем, а также дается обзор существующих средств измерения времени в IBMPC-совместимом компьютере и особенности реализации их виртуальных аналогов.

2 Устройства измерения времени IBMPC-совместимого компьютера

2.1 Общие сведения

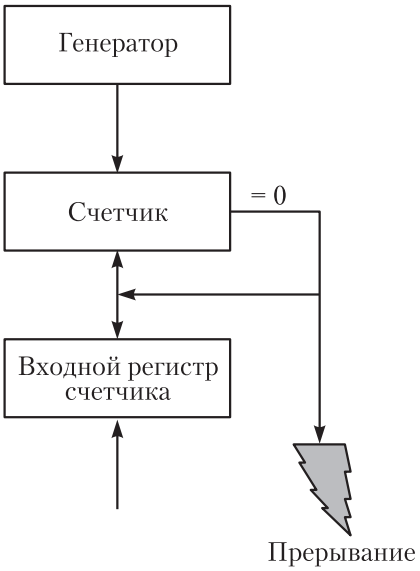


Рис. 1 Абстрактное устройство измерения времени

Исторически персональный компьютер содержит несколько устройств, используемых для подсчета времени. Различные гостевые ОС по-разному выбирают, какие из этих устройств использовать и каким образом. Для некоторых ОС важно использование нескольких устройств совместно. Например, устройство, работающее с известной частотой, используется для измерения скорости работы другого. Или устройство измерения времени с более высокой частотой используется для добавления точности устройству, отсчитывающему время более грубо. Поэтому поддержка всех этих устройств в ВМ очень важна, а главное, что значения временны́х отсчетов, которые читаются из разных устройств, должны быть согласованы между собой. Это условие часто оказывается важнее идентичности работы ВУИВ реальным устройствам.

На рис. 1 показан состав типового устройства измерения времени. Реальные устройства, входящие в состав персонального компьютера, немного отличаются от приведенного на рисунке, однако его можно рассматривать как некоторую абстракцию.

Генератор подает на вход счетчика импульсы с одинаковой частотой. Частота может быть известна заранее, или же ОС измеряет ее при старте. Счетчик может быть программно доступен для чтения и/или записи. Счетчик декрементирует свое значение в каждый момент получения импульса от генератора, а когда достигает нуля, то генерирует выходной сигнал — например, аппаратное прерывание. После этого, если таймер настроен на режим одновибратора, он останавливает свою работу. Если на режим мультивибратора — продолжает считать. Входной регистр счетчика предназначен для загрузки начального значения в счетчик. Таким образом, системное программное обеспечение может контролировать период

генерации аппаратных прерываний. Некоторые таймеры ведут счет в сторону приращения счетчика и имеют дополнительный регистр, хранящий значение, с которым сравнивается текущее значение счетчика. Если они совпадают, то генерируется прерывание, а счет начинается с нуля.

В состав персонального компьютера могут входить следующие устройства измерения времени [2]:

- программируемый таймер (Programmable Interval Timer — PIT);
- часы реального времени CMOS (CMOS real time clock — RTC);
- таймеры Local APIC (Advanced Programmable Interrupt Controller — продвинутый контроллер прерываний);
- таймер ACPI (Advanced Configuration and Power Interface — продвинутый интерфейс конфигурации и управления питанием);
- счетчик временно́й отметки (Time Stamp Counter — TSC);
- высокоточный таймер (High Precision Event Timer — HPET).

2.2 Programmable Interval Timer

Programmable Interval Timer — одно из самых старых устройств измерения времени в архитектуре IBMPC. Он использует генератор с частотой 1,193182 МГц, имеет в составе 16-битный счетчик и входной регистр счетчика. Частота генератора не соответствует ни одной из широко используемых единиц измерения времени — исторически она была удобной для использования в первом компьютере IBMPC. Устройство PIT имеет в своем составе три идентичных таймера, подключенных к различным выходам. Таймер 0 генерирует прерывание и удобен для подсчета системного времени. Таймер 1 используется для обновления ОЗУ и программируется BIOS соответствующим образом. Таймер 2 используется системным спикером для генерации тона.

2.3 CMOS Real Time Clock

Часы CMOS входят в состав энергонезависимого устройства, хранящего настройки BIOS даже после выключения питания компьютера. В состав часов CMOS входят часы времени дня (time of day — TOD), хранящие время в формате год/месяц/число; час/мин/с, а также таймер, который может генерировать прерывания с частотой от 2 до 8192 Гц (кратно степени двойки). Он подобен устройству на рис. 1 с той лишь разницей, что счетчик не доступен для чтения и записи, а во входной регистр счетчика можно записать только число, кратное двум.

Кроме того, можно включить еще два прерывания: прерывание обновления, которое генерируется каждую секунду (на каждое обновление часов TOD) и прерывание по будильнику, которое генерируется в заданное время.

2.4 Таймер Local Advanced Programmable Interrupt Controller

Таймер Local APIC входит в состав логики, обрабатывающей запросы прерываний в современных персональных компьютерах. В мультипроцессорной системе существует одно устройство Local APIC на каждый центральный процессор (ЦП). В современных процессорах оно интегрировано прямо в процессорный чип. Устройство включает в себя 32-битный таймер со входными регистрами. Входная частота, как правило, соответствует частоте работы памяти ЦП, поэтому точность этого таймера значительно выше, чем у таймеров PIT и CMOS.

2.5 Таймер Advanced Configuration and Power Interface

Таймер ACPI — дополнительный таймер, который должен присутствовать в компьютере согласно спецификации ACPI. Также его называют таймером управления питанием или таймером чипсета. Он имеет 24-битный счетчик, который инкрементируется с частотой 3,579545 МГц (в 3 раза больше, чем у PIT). Таймер можно запрограммировать на генерацию прерывания в момент, когда его старший бит меняет свое значение. Входного регистра счетчика нет, поэтому он просто переполняется и начинает считать с нуля снова. Таймер ACPI продолжает работать даже в некоторых энергосберегающих режимах работы, когда другие устройства приостанавливаются. Кроме того, стоит отметить, что он сравнительно медленный для чтения.

2.6 Time Stamp Counter

Time Stamp Counter — 64-битный счетчик, появившийся в процессорах Intel Pentium. Частота его работы соответствует внутренней частоте работы самого ЦП, хотя для некоторых моделей процессоров Intel это условие не выполняется [3]. При скоростях работы современных процессоров для переполнения этого счетчика требуются годы. Time Stamp Counter не генерирует прерывания и не имеет входного регистра счетчика. Значение этого счетчика можно прочитать с помощью инструкции RDTSC или из специального MSR-регистра [4]. Стоит отметить, что счетчик TSC является самым точным и одновременно удобным в использовании для измерения прошедшего времени среди всех устройств, однако и он имеет некоторые недостатки:

- программно трудно точно определить его частоту работы;
- в некоторых системах частота работы TSC меняется динамически;
- некоторые процессоры останавливают счетчик TSC при переходе в энергосберегающий режим.

2.7 High Precision Event Timer

Высокоточный таймер HPET — устройство, появившееся сравнительно недавно как замена PIT и CMOS, и большинству современных ОС не требуется его наличие для работы. Поэтому в большинстве случаев нет необходимости реализовывать соответствующее ВУ, и в рамках этой статьи мы его рассматривать не будем.

2.8 Разработка виртуальных устройств измерения времени

Следует отметить, что большинство гостевых ОС не требуют для своей работы всего набора ВУ, перечисленных выше, поэтому их разрабатывают исходя из конкретных требований, предъявляемых к ОС.

Большинство ВУИВ реализуются методом полной программной эмуляции [5], используя механизмы коррекции времени, описанные в п. 3.

Технология аппаратной виртуализации упрощает реализацию некоторых ВУИВ. Например, в Intel VTx можно задавать величину коррекции счетчика TSC для гостевой ОС, что позволяет компенсировать задержки в исполнении отдельных инструкций, связанные с необходимостью обработки выхода из ВМ (эмуляция инструкций) [4].

3 Базовые сведения о подсчете времени в опреционной системе и проблемы обеспечения корректности работы в виртуальной машине

Как правило, любая ОС, работающая на персональном компьютере, использует два способа измерения прошедшего времени [6]:

1. Подсчет тиков. В данном случае ОС настраивает аппаратное устройство на периодическую генерацию прерываний (тиков) с известной частотой (например, 100 раз/с), а затем считает их и хранит их число, чтобы определить, сколько времени прошло с момента старта. Примером такого устройства является таймер PIT.
2. Подсчет времени без тиков. При таком способе аппаратное устройство хранит число временных отсчетов, прошедших с момента старта ОС, а она читает его значение при необходимости. Этот способ имеет ряд преимуществ: он не нагружает процессор обработкой прерываний и позволяет хранить время в удобной системе подсчета, хотя подсчет времени без тиков можно использовать только на машинах, имеющих подходящее аппаратное устройство: оно должно работать с постоянной частотой и позволять довольно быстро выполнять чтение счетчика. Кроме того, переполнение (или антипереполнение) должно происходить достаточно редко для того, чтобы ОС могла

соответственно обнаружить переполнение и учесть его (например, расширив диапазон подсчитанных временны́х отсчетов). Примером такого устройства является счетчик TSC процессора.

Кроме прошедшего времени ОС используют абсолютное время, которое при старте читается из энергонезависимых часов реального времени, расположенных на материнской плате либо синхронизируется с помощью запроса на сервер времени. Затем ОС может использовать один из вышеназванных методов для подсчета времени, прошедших с момента первой синхронизации. Кроме того, для корректировки часов на длительных промежутках работы и исправления других ошибок в измерении в ОС может быть запущена специальная служба, выполняющая периодическую синхронизацию часов по сети.

3.1 Измерение времени через подсчет тиков

Большинство ОС используют первый способ подсчета прошедшего времени — подсчет тиков, но, к сожалению, обеспечение корректной работы этого механизма в ВМ связано с некоторыми трудностями.

Как правило, ВМ разделяют между собой аппаратные ресурсы, предоставляемые ОС хоста. Кроме того, на той же машине (хосте) вместе с ВМ, являющейся обычной задачей, могут работать и другие приложения, также потребляющие процессорное время. Поэтому в момент, когда ВМ должна сгенерировать очередное прерывание от виртуального таймера[1], она может не исполняться на реальном процессоре, а ожидать получения кванта от планировщика задач в ОС хоста. И в момент получения ею управления уже может накопиться некоторое число этих прерываний.

К тому же, ВМ может запаздывать с отправкой очередного прерывания от виртуального таймера, даже если исполняется в этот момент. Дело в том, что любое ВУ для выполнения каких-либо периодических действий использует специальный механизм так называемых периодических таймеров. Это функции, которые устройство регистрирует в мониторе ВМ (МВМ — программа управления всеми ВМ) и которые он вызывает в определенные моменты времени — например, в момент получения прерывания от реального аппаратного таймера хоста. Однако большинство ОС хоста не предоставляют возможности ВМ изменять частоту генерации прерываний в реальном устройстве, так как это нарушило бы нормальную работу их самих.

Из-за того, что гостевая ОС измеряет время, подсчитывая прерывания, оно будет отставать от реального времени, если к моменту получения управления ВУ уже накоплено несколько «прошедших» прерываний виртуального таймера. Виртуальные устройства, как правило, решают эту проблему, запоминая момент

[1] Здесь и далее «виртуальным таймером» или просто «таймером» называется одно из ВУИВ.

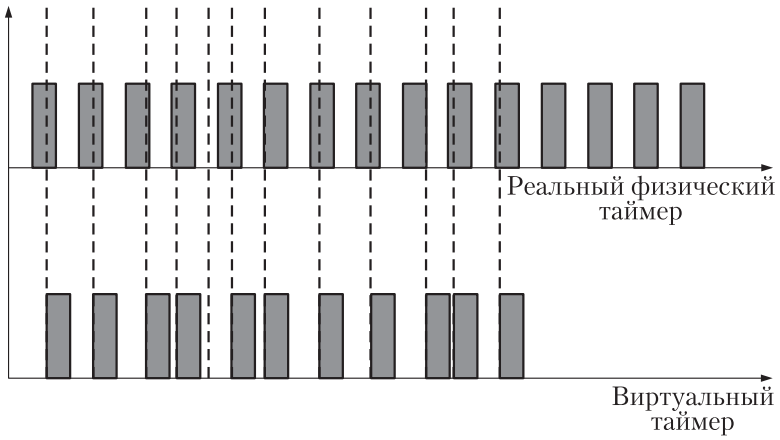


Рис. 2 Генерация прерываний виртуальным и физическим таймером: штриховые линии — моменты времени, в которые виртуальный таймер получает управление

времени, в который сгенерировано последнее виртуальное прерывание, и увеличивая частоту их генерации, если требуется, для того, чтобы дать возможность виртуальному времени «догнать» реальное.

На рис. 2 показано сравнение генерации прерываний виртуальным и физическим таймером. Пунктиром показаны временны́е отсчеты, в которые ВУ таймера получает управление. Их период не является постоянным: он может быть меньше или больше периода генерации прерываний физического таймера. Поэтому, как видно из рисунка, в каждый момент времени, когда ВУ получает управление, оно начинает внедрять «пропущенные» прерывания (если с момента предыдущего внедрения прошло достаточное время), и, таким образом, частота прерываний от ВУ увеличивается или уменьшается.

Работа такого механизма осложняется тем, что очередное прерывание таймера не может быть внедрено, пока гостевая ОС не обработала предыдущее, иначе она может пропустить очередное прерывание и, таким образом, потерять его. Это явление называют «потерей тика». Оно приводит к тому, что в гостевой ОС течение времени «замедляется».

Как видно из рис. 3, обработка прерываний от виртуального таймера использует стандартную последовательность действий, принятую в архитектуре IBMPC при работе программируемого контроллера прерываний (Programmable Interrupt Controller — PIC): ВУ таймера посылает запрос прерывания (Interrupt Request — IRQ) в контроллер прерываний. Тот, в свою очередь, сообщает о наличии прерывания ЦП сигналом INTR (Interrupt Request). Центральный процессор передает управление обработчику аппаратного прерывания и отправляет в контроллер прерываний сигнал подтверждения обработки (Interrupt Acknowledge — INTA).

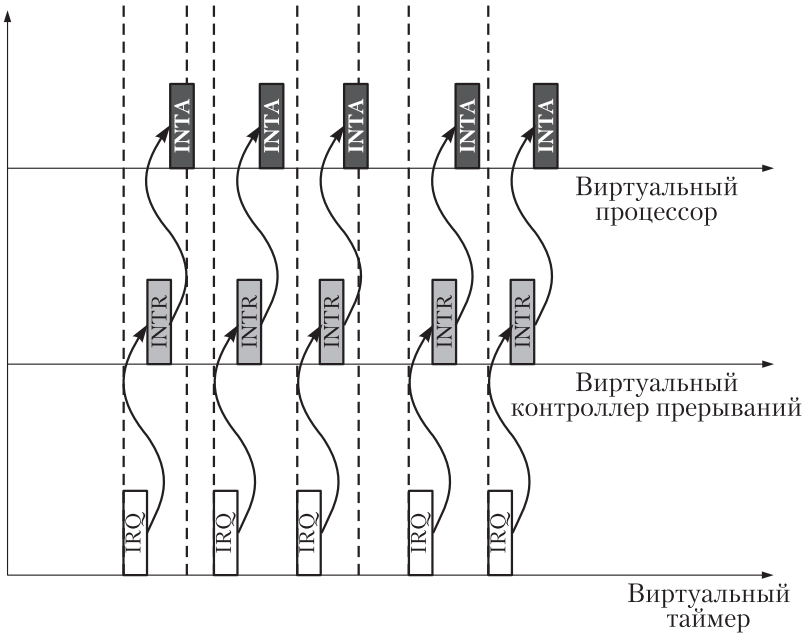


Рис. 3 Временны́е диаграммы обработки прерываний от виртуального таймера: штриховые линии — моменты времени, в которые виртуальный таймер получает управление

Пока виртуальным контроллером не получен сигнал INTA от ЦП, ВУ таймера не должно внедрять очередное прерывание, так как контроллер не сможет сообщить о нем ЦП и оно потеряется. Для этого виртуальный контроллер прерываний может подсчитывать число пришедших подтверждений (сигналов INTA), а ВУ таймера в момент получения управления проверять, пришло ли очередное подтверждение от ЦП и только в этом случае устанавливать новый запрос IRQ.

Для улучшения работы механизма внедрения «накопленных» прерываний технология аппаратной виртуализации предлагает механизм так называемого «выхода по окну прерываний» (Interrupt-window exiting) [4]. Он позволяет при очередном внедрении аппаратного прерывания в ВМ сообщить процессору о наличии еще одного прерывания, обработка которого в настоящий момент невозможна (из-за сброса флага IF по входу в обработчик или по какой-то другой причине). После этого ЦП, продолжив исполнять код ВМ, обеспечивает получение гипервизором управления, как только в ВМ будут выполнены все необходимые условия по возникновению очередного прерывания (в простейшем случае — установка флага IF). В этот момент гипервизор может максимально быстро передать управление виртуальному таймеру, который через виртуальный контроллер PIC выставит очередной запрос INTR в ЦП.

Однако внедрение накопленных прерываний с повышенной частотой не решает проблему отставания времени, если ВМ исполняется медленно (например, из-за большого числа одновременно работающих других ВМ на том же хосте или из-за загруженности ЦП какими-либо другими задачами в ОС хоста), так как прерывания просто не успевают внедряться и обрабатываться гостевой ОС. Как правило, если «время запаздывания» пропущенных прерываний достигает какого-то предельного значения, счетчик пропущенных прерываний сбрасывается и время все-таки теряется. Для осведомленных ОС в данном случае может получить управление специальный сервис, выполняющий синхронизацию времени гостевой ОС и ОС хоста. Однако для неосведомленных ОС такого механизма нет, и восстановить «потерянный тик» невозможно.

Интервал времени наибольшего запаздывания может, например, зависеть от установленного пользователем типа ВМ (гостевой ОС), учитывая особенности измерения времени конкретной ОС.

Даже если ВМ простаивает, она должна потреблять процессорное время хоста в каждый момент получения прерывания от таймера. Если, например, ВМ требует 100 прерываний в секунду (что типично для Windows-подобных ОС), она должна получать управление (при грубом приближении) как минимум 100 раз/с. Таким образом, если работает N ВМ, то правильная обработка прерываний таймера требует $100 \times N$ переключений контекста в секунду, даже если все N ВМ простаивают.

3.2 Подсчет времени без тиков

Для гостевых ОС, использующих подсчет времени без тиков, намного проще организовать правильную работу виртуального таймера, хотя есть некоторые особенности, на которые необходимо обратить внимание.

С одной стороны, если гостевая ОС не подсчитывает тики, нет необходимости замерять время, прошедшее с момента последнего внедрения прерывания от виртуального таймера для того, чтобы, если требуется, «догнать» реальное время. Все прошедшие прерывания с момента последнего чтения счетчика можно просто сложить вместе и возвратить скорректированное значение, не заботясь о «потерянных тиках». Это существенно экономит процессорное время (хоста), что, в свою очередь, делает «течение» времени в гостевой ОС более корректным даже с учетом того, что ВМ может исполняться очень медленно.

Однако, для того чтобы использовать все эти преимущества, МВМ должен знать, что гостевая ОС использует подсчет времени без тиков, поскольку если она на самом деле считает тики, то пропускать их будет некорректно. Существует несколько методов определения подсчета времени без тиков:

1. Если гостевая ОС не запрограммировала ни одно из ВУИВ на периодическую генерацию прерываний, можно полагать, что она использует подсчет

времени без тиков, хотя существуют некоторые ОС, не подчиняющиеся этому правилу.

2. Тип измерения времени можно определить по установленному пользователем типу гостевой ОС (при создании ВМ).
3. Осведомленные ОС могут сообщить используемый тип измерения времени, сделав соответствующий гипервызов (вызов в МВМ).

Стоит отметить, что в некоторых случаях эмуляция подсчета тиков корректнее, чем без них: в частности, для некоторых программ, очень чувствительных ко времени, например при подсчете числа итераций цикла, работающего в течение заданного интервала времени. В некоторых случаях такой код будет работать правильнее при измерении времени с подсчетом тиков, так как в таких гостевых ОС время «замедляется», когда сама ВМ замедляет свою работу.

3.3 Реальное время

Любой ОС как при работе на физической машине, так и на виртуальной, требуется инициализировать часы корректным значением реального времени при загрузке и постоянно обновлять его.

Виртуальная машина предоставляет те же возможности для работы с реальным временем, что и физическая:

– ВУ часов реального времени CMOS, которое, как правило, при старте инициализируется значением, хранящимся в реальном физическом устройстве;
– синхронизация времени по сети с сервером времени через ВУ сетевого адаптера;
– синхронизация времени с ОС хоста (только для осведомленных ОС); при этом для передачи значения лучше использовать формат UTC (Coordinated Universal Time — универсальное координированное время). В этом случае гостевая ОС и ОС хоста могут использовать разные временны́е зоны.

Виртуальные машины также имеют следующую особенность: если ВМ возобновляет свою работу после приостановки или из мгновенного снимка состояния, то значение реального времени в ней остается тем, которое было на момент остановки ее работы. Осведомленные ОС в этом случае используют гипервызовы для синхронизации времени с ОС хоста. Однако в некоторых случаях требуется фиктивное время, не связанное с хостом, поэтому такую синхронизацию, как правило, делают опциональной.

На длительных промежутках времени реальные устройства измерения времени могут накапливать погрешность, которая составляет десятитысячные доли в бóльшую или меньшую сторону в зависимости от температуры внутри микросхемы. В ВУИВ к этой погрешности прибавляется еще и погрешность округления,

которая накапливается при вычислении величины коррекции времени в ВУ, поэтому, если гостевая ОС работает в ВМ очень долго, необходимо предусмотреть механизм коррекции времени, например синхронизировать время через сеть.

4 Заключение

В статье перечислены основные устройства измерения времени персонального компьютера, а также основные трудности, с которыми может столкнуться разработчик любого из ВУИВ, и показаны возможные пути их решения.

Обеспечить корректность работы ВУИВ и соответственно точность измерения времени в гостевой ОС очень сложно, а зачастую — невозможно. Однако стоит отметить, что в большинстве случаев для нормальной работы гостевой ОС требуется не точное соответствие работы ВУИВ его аппаратному аналогу, а согласованность работы всех ВУ в составе платформы виртуализации.

Литература

1. *Таненбаум Э.* Современные операционные системы. — 3-е изд. — Спб.: Питер, 2010. 1120 с.
2. *Гук М.* Аппаратные средства IBM PC. — 3-е изд. — Спб.: Питер, 2006.
3. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide. Part 1. September 2008. Intel Corporation. (www.intel.com)
4. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide. Part 2. September 2008. Intel Corporation. (www.intel.com)
5. *Егоров В. Ю., Карпов И. В., Матвеев Е. А.* Технологии реализации аппаратуры компьютера в составе виртуальных машин // Системы и средства информатики. — М.: Наука, 2009. Доп. вып. С. 68–76.
6. Timekeeping in VMware Virtual Machines. 2008. (www.vmware.com)

ОБСЛУЖИВАНИЕ ЗАПРОСОВ ПРЯМОГО ДОСТУПА К ПАМЯТИ В КОНТРОЛЛЕРАХ ЖЕСТКИХ ДИСКОВ ВИРТУАЛЬНЫХ МАШИН

М. А. Шпадырев[1]

Аннотация: Статья посвящена особенностям обслуживания запросов прямого доступа к памяти (ПДП) в реальных и виртуальных контроллерах жестких дисков. Описаны существующие методы обслуживания таких запросов в виртуальных машинах (ВМ), а также предложен новый метод, основанный на обработке данных в нескольких нитях.

Ключевые слова: контроллер жестких дисков; прямой доступ к памяти; виртуальная машина; виртуальное устройство; гипервизор

1 Введение

Современные платформы виртуализации включают в себя один или несколько компьютеров, называемых хостами, операционную систему (ОС) хоста, а также модуль управления ВМ (монитор виртуальных машин) вместе с виртуальными устройствами [1].

На сегодня известно несколько методов реализации виртуальных устройств (ВУ) в составе ВМ, каждый из которых имеет свои преимущества и недостатки [1]. Однако наиболее значимым критерием выбора того или иного метода является производительность каждого из ВУ, поскольку именно она в значительной степени определяет производительность ВМ в целом. А это напрямую связано с качеством услуг, предоставляемых платформой виртуализации.

К сожалению, любое ВУ (за редким исключением) всегда будет проигрывать в производительности своему аппаратному аналогу. Тем не менее, при разработке ВУ всегда стремятся максимально приблизить показатели производительности к аналогичным показателям реального устройства. Для разных устройств степень этого приближения будет разной. При этом очевидно, что в составе любой вычислительной системы не должно быть медленно работающих компонентов, поскольку они будут тормозить работу других. И, как показывают результаты тестов, опубликованные в открытых источниках, подсистема хранения данных сегодня наиболее трудно поддается виртуализации.

Устройства хранения данных — одна из неотъемлемых частей ВМ. Прежде всего, это PCI-совместимый IDE-контроллер жестких дисков и CD-ROM,

[1]Пензенский государственный университет, lordm@nm.ru

входящий в состав микросхемы южного моста [2]. Данная статья посвящена вопросам обслуживания запросов прямого доступа к памяти (ПДП) в виртуальном контроллере IDE.

2 PIO и DMA запросы в реальном контроллере IDE

Программный ввод/вывод (*англ.* Programmed input/output, PIO) — метод передачи данных между двумя устройствами, использующий центральный процессор (ЦП) как часть маршрута данных.

В реальном контроллере IDE, работающем в режиме PIO, для передачи данных используется специальный порт данных [3, 4]. При этом для чтения очередного сектора диска ЦП после подготовки соответствующего запроса чтения и отправки его в контроллер ожидает появления запроса аппаратного прерывания от него. Аппаратное прерывание служит сигналом о том, что данные готовы (или произошла ошибка, но этот случай мы здесь не рассматриваем), после чего ЦП начинает читать данные из порта данных и записывать их в память. За один доступ к порту ввода/вывода ЦП не может считать больше двойного слова (32 бита), а для некоторых контроллеров не больше 16-битного слова, и поэтому вынужден выполнять эту операцию в цикле до тех пор, пока все данные не будут перемещены из устройства в память.

Таким образом, главным недостатком режима PIO является то, что ЦП загружен выполнением операции ввода/вывода до момента ее непосредственного завершения, что, естественно, не лучшим образом сказывается на общей производительности системы. Этого недостатка лишен режим ПДП.

Прямой доступ к памяти (*англ.* Direct Memory Access, DMA) — режим обмена данными между устройствами или же между устройством и оперативной памятью без участия ЦП.

В режиме ПДП по сравнению с PIO-режимом скорость передачи данных возрастает благодаря тому, что данные пересылаются, минуя ЦП. От последнего требуется лишь соответствующим образом инициализировать контроллер и инициировать операцию передачи данных в режиме ПДП. После этого ЦП может вернуться к исполнению каких-либо других операций (в многозадачной ОС), а непосредственную запись или чтение данных из контроллера IDE в оперативную память выполняет контроллер DMA (микросхема Intel 8237), который, как правило, входит в состав PCI-совместимого IDE-контроллера.

DMA-контроллер может получать доступ к системной шине независимо от ЦП и содержит несколько регистров, доступных ЦП для чтения и записи. Регистры контроллера задают направление передачи данных (чтение/запись), единицу передачи (побайтно/пословно), число байтов, которое следует передать.

Кроме того, в режиме ПДП данные пересылаются сразу большим блоком слов, идущих по адресам, расположенным последовательно, что существенно

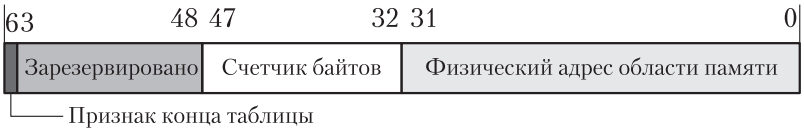


Рис. 1 Формат дескриптора PRD

ускоряет работу системной шины, так как выполняется всего один цикл адреса и многочисленные циклы данных.

Область оперативной памяти для передачи данных задается для контроллера DMA при помощи таблицы дескрипторов физических областей памяти PRDT (Physical Region Descriptor Table — таблица дескрипторов физических регионов). Таблица состоит из одного или нескольких 8-байтных дескрипторов PRD (Physical Region Descriptor — дескриптор физического региона), которые и описывают используемые при передаче данных участки оперативной памяти. Размер таблицы не должен превышать 64 Кбайт, т. е. таблица может содержать до 8192 элементов.

Формат дескриптора PRD показан на рис. 1. Физический адрес участка памяти выравнивается на слово, т. е. должен быть четным 32-разрядным числом.

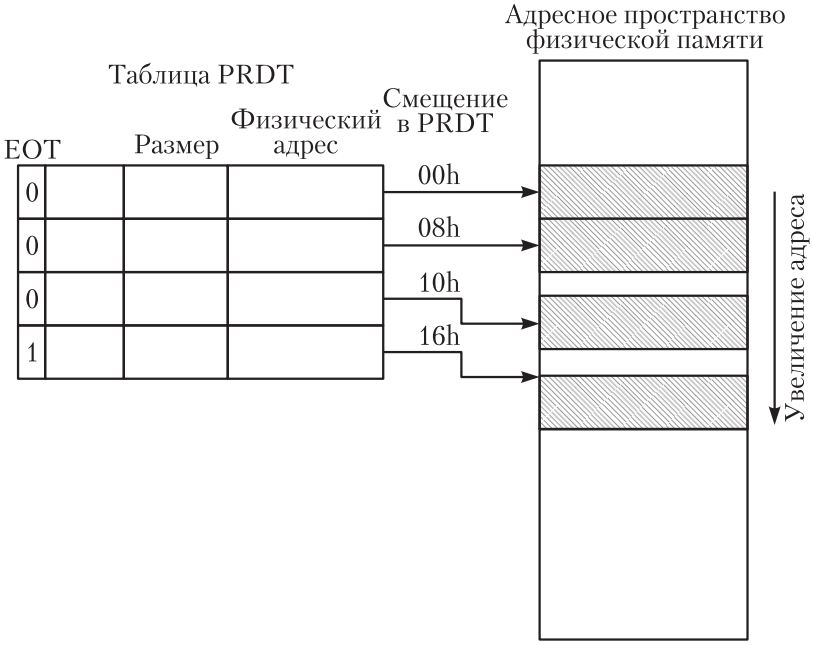


Рис. 2 Таблица PRDT и адресуемые участки физической памяти

Счетчик байт должен быть четным 16-разрядным числом. В слове признака конца таблицы используется только старший разряд, который принимает значение 1 у последнего элемента таблицы, а у остальных элементов равен 0. Суммарный размер всех областей, описанных в PRDT, не должен быть меньше объема данных, заданных контроллеру IDE в команде передачи данных в режиме ПДП.

В поле счетчика байтов может находиться любое число от 0 до FFFEh. При этом значение 0 соответствует 10000h, т. е. размер участка может варьироваться от 2 байт до 64 Кбайт. На практике драйвер жесткого диска, входящий в состав ОС, инициализирует счетчик байтов значением 64 Кбайта (максимально возможный блок данных), 4 Кбайта (размер страницы) или размером одной из дисковых структур — сектора или кластера. Размер всех используемых участков, как правило, одинаковый.

На рис. 2 показан пример таблицы PRDT и адресуемые участки физической памяти.

3 PIO и DMA запросы в виртуальном контроллере IDE

3.1 Общие сведения

Любое ВУ в составе ВМ может взаимодействовать с процессором и с другими ВУ следующими способами [1]:

- чтение или запись в память ВУ, отображенную в пространство физической памяти;
- чтение или запись в память ВУ, отображенную в пространство ввода/вывода;
- чтение или запись ВУ данных в оперативную память ВМ;
- установка ВУ сигнала прерывания (либо в контроллере прерываний, либо напрямую в процессор).

Применительно к виртуальному контроллеру IDE можно отметить, что в режиме PIO данные передаются исключительно через адресное пространство ввода/вывода ЦП, а в режиме ПДП через порты ввода/вывода идет только передача команд контроллеру, в которых содержатся все необходимые параметры передачи данных, а также сигнал инициации передачи. После принятия команды начала передачи контроллер не нагружает ЦП, а начинает читать или записывать данные в режиме Bus Master напрямую в оперативную память ВМ.

После того как передача данных завершена, контроллер устанавливает запрос прерывания в виртуальном контроллере прерываний, что освобождает контроллер IDE для обработки следующего запроса. Контроллер прерываний в свою очередь сообщает ЦП о поступившем запросе. В ВМ, как правило, наличие запросов аппаратных прерываний проверяется с определенной периодичностью,

поэтому, например, после очередной передачи управления из ВМ в гипервизор ЦП будет вызван соответствующий обработчик прерываний, а затем он вновь продолжит исполнять код ВМ, ожидая прихода очередного запроса прерывания.

Существуют несколько вариантов реализации обработки запросов ПДП в ВМ:

(1) обработка запросов ПДП в основной нити виртуального процессора;

(2) обработка запросов ПДП в отдельной нити виртуального контроллера IDE;

(3) обработка запросов ПДП в нескольких отдельных нитях виртуального контроллера IDE.

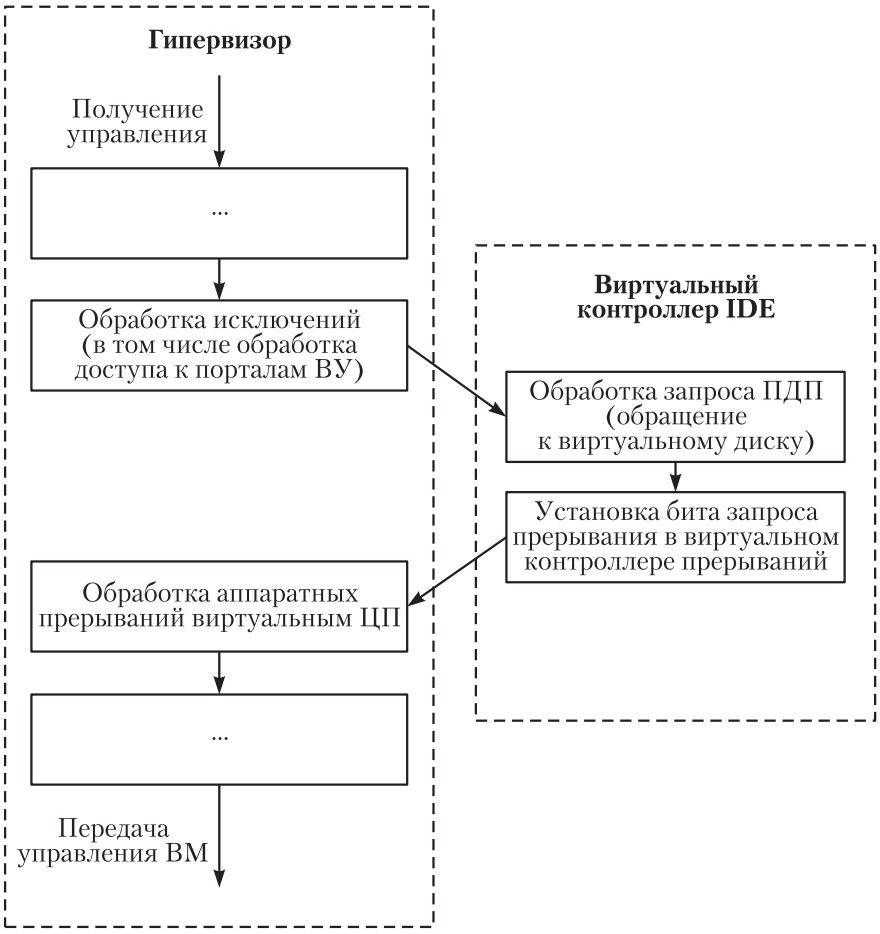


Рис. 3 Обработка запросов ПДП в основной нити

3.2 Обработка запросов прямого доступа к памяти в основной нити виртуального процессора

Данный способ программной реализации обработки запросов ПДП подразумевает наличие специальной функции, выполняющей обработку всех поступивших запросов синхронно в основной нити виртуального процессора. Как видно из рис. 3 (стрелками показана передача управления), в этой же нити гипервизор выполняет все необходимые действия по управлению ВМ: обработка исключений (следствием которых может быть эмуляция инструкций доступа к портам ввода/вывода, т. е. передача управления в соответствующее виртуальное устройство [5, 6]), обработка поступивших аппаратных прерываний от виртуальных устройств и т. п.

Поэтому, несмотря на то что данный способ наиболее прост для реализации, он фактически не дает никакого преимущества обработки DMA-запросов по сравнению с PIO. Так как обработка запроса происходит синхронно, основная нить виртуального ЦП вынуждена простаивать, дожидаясь завершения операций с виртуальным диском и установки запроса прерывания в виртуальном контроллере прерываний, что тормозит работу всей ВМ в целом.

3.3 Обработка запросов прямого доступа к памяти в отдельной нити

Данный способ программной реализации обработки запросов ПДП по логике своей работы наиболее приближен к работе реального контроллера IDE (рис. 4), поскольку обработка происходит асинхронно в отдельной нити, работающей в виртуальном контроллере, и поэтому он используется в большинстве современных платформ виртуализации (если анализировать платформы с открытым исходным кодом).

Гипервизор при обработке очередного исключения, связанного с доступом к портам ввода/вывода контроллера IDE, передает управление виртуальному контроллеру, который лишь регистрирует запрос ПДП (запоминая всего его параметры) и сразу же возвращает управление основной нити виртуального ЦП, которая может продолжить свою «привычную» работу. Также виртуальный контроллер IDE пробуждает собственную нить, ожидающую поступление запросов ПДП. Эта нить начинает обмен между оперативной памятью ВМ и виртуальным жестким диском, в то время как в многозадачной ОС хоста код гостевой ОС продолжает исполняться в другой нити. Когда обмен завершен, нить виртуального контроллера IDE выставляет запрос прерывания в контроллере прерываний, и этот запрос уже обрабатывается гипервизором в основной нити виртуального ЦП на очередной итерации получения им управления.

Потери производительности в данном случае возникают только в ситуациях пиковой нагрузки на контроллер, когда запросы ПДП поступают так быстро, что очередной запрос не успевает обрабатываться до момента прихода следующего.

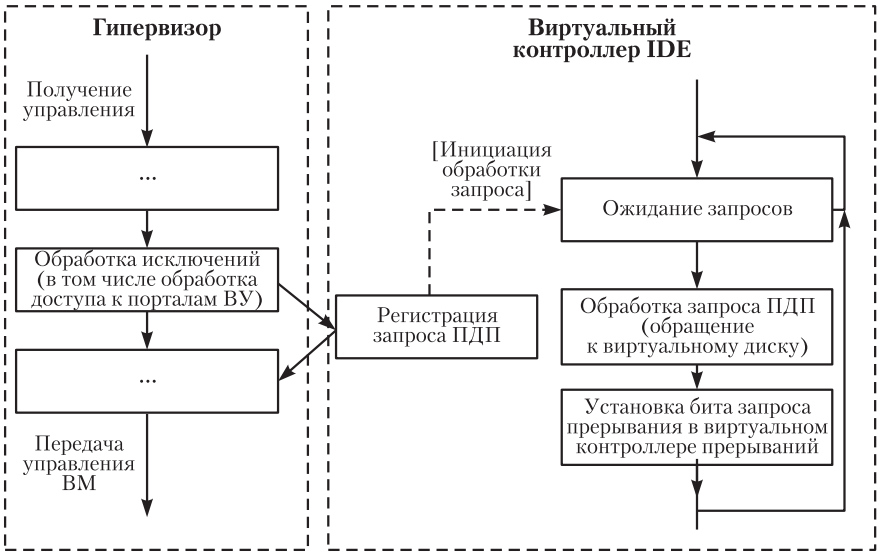


Рис. 4 Обработка запросов ПДП в отдельной нити

Однако предложенный ниже способ позволяет частично нивелировать подобные задержки.

3.4 Обработка запросов прямого доступа к памяти в нескольких нитях

На рис. 5 схематично показана реализация обработки запросов ПДП в нескольких нитях (на рисунке показан только виртуальный контроллер IDE). В данном случае внутри виртуального контроллера работают сразу несколько нитей, каждая из которых обрабатывает определенный тип запросов. Маршрутизация запроса происходит сразу же после его регистрации (при доступе к адресному пространству ввода/вывода ВУ).

Запросы можно разделять по разным критериям, классифицирующим запросы соответственно возможности их одновременной (параллельной) обработки. Например, исходное и целевое виртуальное устройство (жесткий диск, CD-ROM), доступ на чтение и запись, адреса доступа к оперативной памяти и т. п.

Данный способ дает определенный прирост производительности дисковой подсистемы в ВМ, поскольку позволяет обрабатывать несколько запросов параллельно. Технически это работает даже интеллектуальнее, чем реальное аппаратное устройство, поскольку последнее позволяет захватывать шину в режиме Bus Master только одному из подключенных к ней устройств в один момент времени. Однако такая реализация требует дополнительной логики, которая

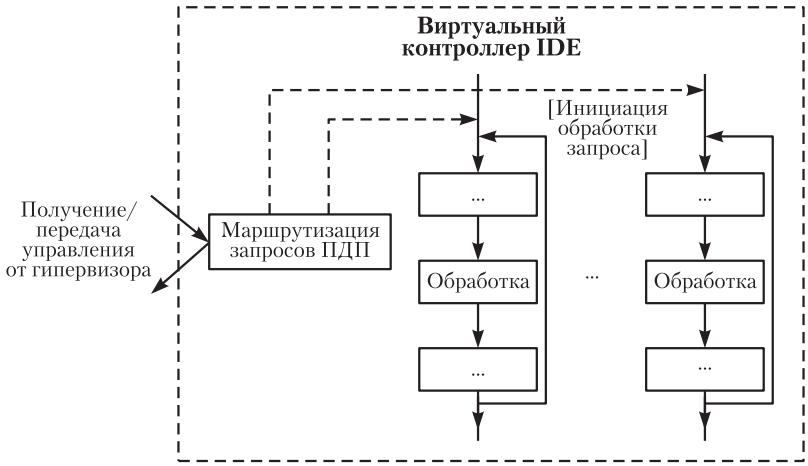


Рис. 5 Обработка запросов ПДП в нескольких нитях

будет осуществлять синхронизацию доступа к нескольким критическим ресурсам в составе ВМ. Это следующие ресурсы:

(1) сектора виртуального диска;

(2) ячейки оперативной памяти;

(3) запрос в виртуальном контроллере прерываний от определенного канала IDE.

Про первые два пункта следует отметить следующее. Очевидно, что мы не можем предоставить доступ на чтение определенного сектора виртуального диска или ячейки оперативной памяти, если для нее в данный момент времени не завершена отложенная операция записи.

Также виртуальный контроллер прерываний должен ожидать подтверждения принятия ЦП запроса прерывания от одного из каналов IDE (которому соответствует свой номер IRQ) до того, как сможет отправить следующий запрос.

4 Заключение

Каждый из описанных в данной статье методов обработки запросов ПДП имеет свои преимущества и недостатки, а также отличается по трудоемкости своей практической реализации.

Проблема производительности дисковой подсистемы в ВМ по-прежнему остается очень актуальной, и при невозможности усовершенствования методов реализации виртуальных контроллеров IDE ее решают простым увеличением

вычислительных мощностей, что, естественно, увеличивает и финансовые затраты. Новые методы и разработки в данной области позволят сократить данные расходы.

Литература

1. *Егоров В. Ю., Карпов И. В., Матвеев Е. А.* Технологии реализации аппаратуры компьютера в составе виртуальных машин // Системы и средства информатики. — М.: Наука, 2009. Доп. вып. С. 68–76.
2. Intel® I/O Controller Hub 7 (ICH7) Family. Datasheet. April 2007. Intel Corporation. (www.intel.com)
3. *Кулаков В.* Программирование дисковых подсистем. — СПб.: Питер, 2002. 768 с.
4. *Гук М.* Аппаратные средства IBM PC. — 3-е изд. — СПб.: Питер, 2006. 1072 с.
5. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide. Part 1. September 2008. Intel Corporation. (www.intel.com)
6. Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide. Part 2. September 2008. Intel Corporation. (www.intel.com)

АКАДЕМИК В. С. ПУГАЧЁВ: КРАТКИЙ ОЧЕРК НАУЧНОЙ, ПЕДАГОГИЧЕСКОЙ, НАУЧНО-ОРГАНИЗАЦИОННОЙ И ОБЩЕСТВЕННОЙ ДЕЯТЕЛЬНОСТИ

***И. А. Соколов*[1], *И. Н. Синицын*[2]**

Академик Владимир Семёнович Пугачёв — выдающийся советский ученый, основоположник статистической теории управляемых систем, автор фундаментальных работ в области авиационной баллистики и динамики полета, теории дифференциальных уравнений и теории вероятностей, теории управления и информатики.

В. С. Пугачёв родился 25 марта 1911 г. в городе Рязани. Его отец Семён Андреевич Пугачёв был одним из активных участников строительства Красной Армии, соратником М. Н. Тухачевского, Г. К. Орджоникидзе и М. В. Фрунзе.

С детских лет он интересовался автомобилями и летательными аппаратами, увлекался астрономией и математикой. В 15-летнем возрасте самостоятельно изучил основы аналитической геометрии и математического анализа, читал книги на английском и французском языках, занимался музыкой и живописью. В. С. Пугачёв окончил школу-семилетку в г. Моске в 1926 г. Следуя традициям семьи по линии матери, В. С. Пугачёв решил стать инженером-путейцем и поступил в 1928 г. в Московский институт инженеров транспорта.

В 1929 г. по спецнабору был переведен на второй курс инженерного факультета ВВА им. Н. Е. Жуковского. Вся постановка обучения в академии способствовала быстрому формированию инженеров и ученых в области авиации. Этому помогла и практика, которую В. С. прошел в конструкторских бюро А. Н. Туполева и С. В. Ильюшина. Он окончил академию за два года и в 1931 г. был оставлен в ней адъюнктом. Однако ввиду необходимости усилить практические работы для обеспечения быстрейшего развития авиационной техники через полгода летом 1932 г. В. С. Пугачёв был назначен начальником вычислительного сектора Научно-испытательного института ВВС.

В. С. Пугачёв хорошо понимал, что создавать авиационную науку невозможно без глубоких знаний в области математики. В течение полугода в конце 1932 и в начале 1933 г. он сдал экстерном экзамены за механико-математический факультет МГУ. В это время В. С. Пугачёв начал сотрудничать с такими выдающимися

[1]Институт проблем информатики Российской академии наук, isokolov@ipiran.ru
[2]Институт проблем информатики Российской академии наук, sinitsin@dol.ru

учеными и педагогами, как член-корреспондент АН СССР В. В. Голубев, профессора Н. Н. Бухгольц, И. И. Привалов и В. В. Степанов. Участвовал в работе семинара профессора А. А. Витта по нелинейным колебаниям.

Решая важные задачи по составлению баллистических таблиц для авиабомб, В. С. Пугачёв разработал новый метод расчета траекторий авиабомб, применив впервые в мировой науке метод малого параметра Пуанкаре к задачам баллистики. Эта работа была высоко оценена выдающимся советским специалистом в области внешней баллистики профессором Д. А. Вентцелем. Начиная с осени 1932 г. по март 1934 г. написал кандидатскую диссертацию «О применении метода Пуанкаре к интегрированию уравнения движения бомбы, сброшенной с самолета».

В 1935 г. В. С. Пугачёв был назначен начальником кафедры воздушной стрельбы ВВА им. Н. Е. Жуковского. В короткий срок им были разработаны основы теории воздушной стрельбы — новой отрасли авиационной науки, создан учебный курс, написан учебник и подготовлены научно-педагогические кадры. Впоследствии первоначально изданный ВВА им. Н. Е. Жуковского учебник лег в основу книги «Теория воздушной стрельбы» (1940). В апреле 1939 г. защитил докторскую диссертацию на тему «Общая задача о движении вращающегося артиллерийского снаряда в воздухе».

В этом же году В. С. Пугачёв выполнил работу по комплексной оценке эффективности стрельбы в воздушном бою, впервые применив методы исследования операций и системного анализа для научного обоснования направлений развития авиационной техники. Таким образом, В. С. Пугачёв является основоположником исследования операций и системного анализа в нашей стране, на несколько лет опередившим аналогичные исследования за рубежом.

При разработке вопросов теории воздушной стрельбы В. С. Пугачёву приходилось решать задачи, определившие новое направление во внешней баллистике — авиационную баллистику. В поисках методов решения новых задач баллистики В. С. Пугачёв внес существенный вклад в аналитическую теорию дифференциальных уравнений — развил теорию новых классов асимптотических разложений решений линейных дифференциальных уравнений, содержащих параметр.

В условиях нараставшей угрозы со стороны фашистской Германии В. С. Пугачёв участвовал в качестве руководителя и ответственного исполнителя в выполнении работ, направленных непосредственно на повышение эффективности авиационной техники и в годы войны вместе со всем коллективом ВВА им. Н. Е. Жуковского продолжал во все увеличивавшихся масштабах работу по подготовке авиационных инженеров для фронта. С октября 1943 г. по сентябрь 1944 г. В. С. Пугачёв работал заместителем начальника Научно-испытательного института авиавооружения по научной работе, продолжая по совместительству работу начальника кафедры ВВА им. Н. Е. Жуковского.

Для решения ряда важных задач, связанных с повышением эффективности стрельбы и бомбометания, требовалось изучение процессов управления в условиях случайных возмущений. В то время в мировой литературе для подобных задач не существовало методов решения. В поисках их В. С. Пугачёв разработал общую теорию систем, описываемых стохастическими дифференциальными уравнениями. Работой «Случайные функции, определяемые обыкновенными дифференциальными уравнениями» (1944) были заложены основы нового научного направления — статистической теории процессов управления (статистической динамики) задолго до появления публикаций в этой области за рубежом. В. С. Пугачёвым впервые изучались стохастические дифференциальные уравнения с любыми процессами с независимыми приращениями.

В период 1946–1949 гг. В. С. Пугачёв вел научную работу по совместительству в НИИ-2 Министерства авиационной промышленности. В 1949 г. приказом министра обороны СССР был назначен по совместительству на должность постоянного члена Авиационного технического комитета ВВС и работал на этой должности до 1950 г. В этом же году был назначен научным руководителем предприятия п/я 1323 и работал там до 1953 г., не прекращая работать по совместительству начальником кафедры Военно-воздушной инженерной академии им. профессора Н. Е. Жуковского (ВВИА им. Н. Е. Жуковского).

В послевоенный период научная деятельность В. С. Пугачёва была связана с дальнейшей разработкой статистической теории процессов управления и ее применений в различных областях науки и техники. В 1947 г. им были завершены исследования по общей статистической теории линейных систем и приближенных методов исследования точности нелинейных систем. Разработанные им методы теории управления послужили основой его последующих работ в области динамики управляемого полета, выполненных в период 1947–1952 гг. В 1947 г. В. С. Пугачёв был избран в члены-корреспонденты Академии артиллерийских наук. В 1948 г. за теоретические исследования в области баллистики В. С. Пугачёву была присуждена Государственная премия СССР.

В 1956 г. В. С. Пугачёв был приглашен в Институт автоматики и телемеханики АН СССР для организации лаборатории и работ в области статистических методов в теории управления. В то же время он начал работу в качестве председателя комиссии по статистическим проблемам Научного совета по комплексной проблеме «Кибернетика» при Президиуме АН СССР, организовал общемосковский семинар по статистическим проблемам в кибернетике.

В 1950-е гг. В. С. Пугачёвым были созданы методы статистической теории оптимальных систем. В этой области им были развиты общие методы определения оптимальных линейных систем, основанные на разработанной им же теории канонических разложений случайных функций. Кроме того, им созданы общие методы оптимизации динамических систем по любым статистическим критериям. Работы В. С. Пугачёва в области статистической теории оптимальных систем

занимают ведущее место в мировой науке и получили широкое распространение. Всего в области статистической динамики управляемых систем В. С. Пугачёвым опубликовано свыше 70 работ. Результаты работ В. С. в области статистических методов теории процессов управления обобщены в монографии «Теория случайных функций и ее применение к задачам автоматического управления» (1957, 1960, 1962).

В начале 1960-х гг. В. С. Пугачёв создал новый курс теории автоматического управления, который он читал в ВВИА им. Н. Е. Жуковского. В это же время В. С. Пугачёв совместно со своими учениками И. Е. Казаковым, Д. И. Гладковым, Л. Г. Евлановым, С. В. Мальчиковым, А. Ф. Мишаковым, В. Д. Седовым, В. И. Соколовым написал монографию «Основы автоматического управления», в которой впервые систематически излагались применявшиеся в то время методы исследований автоматических систем, включая статистические методы (1963, 1968, 1974). В период 1965–1970 гг. главным направлением научной работы В. С. Пугачёва была разработка статистической теории обучающихся автоматических систем. В 1966 г. В. С. Пугачёв был избран членом-корреспондентом АН СССР по специальности «автоматическое управление».

В 1970-е гг. В. С. Пугачёвым заложены основы общей структурной теории стохастических систем. За участие в создании адаптивной системы управления сложным технологическим процессом (горячая прокатка труб) в 1976 г. В. С. Пугачёв удостоен Государственной премии СССР. При создании этой системы были использованы методы теории обучающихся систем, разработанные В. С. Пугачёвым.

Научная деятельность В. С. Пугачёва тесно переплеталась с педагогической деятельностью в ВВИА им. Н. Е. Жуковского в течение почти 40 лет. За годы работы в ВВИА им. Н. Е. Жуковского им были созданы четыре кафедры, разработаны соответствующие специальные курсы, написаны учебники и учебные пособия, организованы соответствующие лаборатории, подготовлены научно-педагогические кадры.

С 1973 г. педагогическая работа В. С. Пугачёва сосредоточена в Московском авиационном институте им. Серго Орджоникидзе (МАИ), где он вел педагогическую работу в конце 1930-х гг. В МАИ он организовал кафедру теории вероятностей и математической статистики на факультете прикладной математики, создал новые специальные курсы. По материалам лекций был написан учебник «Теория вероятностей и математическая статистика» (1979). Книга переведена и издана на английском (1984) и французском языках (1982).

В конце 1970-х – начале 1980-х гг. В. С. Пугачёвым заложены основы нового научного направления — теории условно оптимальной фильтрации и экстраполяции процессов в стохастических системах.

Все прикладные работы В. С. Пугачёва всегда были ориентированы на применение вычислений. В 1930-е гг. он сам выполнял большие вычислительные

работы в области авиационной баллистики, для него были характерны доведенные до числа научные результаты, так необходимые практике. В. С. Пугачёв был одним из первых ученых, начавших широко применять вычислительную технику в своей практической работе. Развитие ЭВМ значительно расширило возможность практического применения разработанных В. С. Пугачёвым статистических методов. Эти методы в задачах теории управления и информатики вследствие своей большой вычислительной сложности могут быть практически реализованы только с использованием вычислительной техники. Поэтому их дальнейшее развитие и применение, предъявляющее все более и более высокие требования к ЭВМ, тесно связано с развитием аппаратных и программных средств вычислительной техники новых поколений. В 1981 г. В. С. Пугачёв был избран действительным членом АН СССР по специальности «теория управления, вычислительная техника».

С 1984 г. В. С. Пугачёв работает во вновь созданном Институте проблем информатики АН СССР, возглавляет отдел статистических основ информатики. В 1986–1990 гг. руководил комплексным научным проектом «Новые алгоритмы и архитектуры обработки информации» в рамках приоритетного направления «электронизация» комплексной программы научно-технического прогресса стран СЭВ и «Концепции по созданию новых поколений вычислительных систем».

В период 1989–1993 гг. как советник при дирекции ИПИ РАН В. С. Пугачёв руководил научным направлением «Статистические основы информатики и управления». Результаты многолетних работ В. С. Пугачёва по теории стохастических систем, описываемых дифференциальными уравнениями, обобщены и систематически изложены в монографии «Стохастические дифференциальные системы. Анализ и фильтрация», написанной совместно с И. Н. Синицыным (1985, 1987, 1990). В 1990 г. за комплекс работ по статистической теории процессов управления, выполненных в 1944–1987 гг., В. С. Пугачёв был удостоен Ленинской премии. С 1993 г. до последних дней как советник РАН В. С. Пугачёв возглавлял научное направление «Методы функционального анализа в информатике».

Научная, научно-организационная, педагогическая и общественная деятельность В. С. Пугачёва была по достоинству оценена нашей Родиной. Он заслуженный деятель науки и техники РСФСР, лауреат Ленинской и двух Государственных премий СССР, награжден многими орденами и медалями.

Подробные биобиблиографические материалы о В. С. Пугачёве опубликованы в сборнике «Академик Владимир Семёнович Пугачёв: к столетию со дня рождения» // Предисловие академика С. В. Емельянова. — М.: ТОРУС ПРЕСС, 2011 (под ред. И. Н. Синицына).

БИБЛИОГРАФИЯ НАУЧНЫХ ТРУДОВ СОТРУДНИКОВ ИПИ РАН ЗА 2010 ГОД

1 МОНОГРАФИИ

1.1 Монографии, изданные в ИПИ РАН

1. *Егоров В. Б.* Аппаратные платформы пакетной коммутации и маршрутизации на основе интегрированных коммуникационных микроконтроллеров. — М.: ИПИ РАН, 2010. 147 с.
2. *Зацаринный А. А., Ионенков Ю. С., Козлов С. В.* Некоторые вопросы проектирования информационно-телекоммуникационных систем / Под ред. д.т.н. А. А. Зацаринного. — М.: ИПИ РАН, 2010. 218 с.
3. Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. 319 с.
4. *Ильин В. Д.* S-модель нормализованной экономической системы (электронная публикация). — М.: ИПИ РАН, 2010. 103 с.
5. *Ильин А. В., Ильин В. Д.* S-моделирование объектов информатизации (электронная публикация). — М.: ИПИ РАН, 2010. ISBN 978-5-902030-86-7. 412 с.
6. *Колесников А. В., Кириков И. А., Листопад С. В., Доманицкий А. А., Румовская С. Б.* Решение сложных задач коммивояжера методами функциональных гибридных интеллектуальных систем. — М.: ИПИ РАН, 2011. 393 с.
7. *Соколов И. А., Синицын И. Н., Андерс Б. Н., Баженова Т. В., Бигдай Л. К., Гиглавый А. В., Гольдина Л. Л., Захаров В. Н., Козмидиади В. А., Курц А. Л., Лабунская О. А., Лавренюк Ю. А., Неменман М. А., Фридман А. Л., Юдашкин М. В.* Игорь Яковлевич Ландау / Под ред. И. А. Соколова, И. Н. Синицына. — М.: ИПИ РАН, 2010. 50 с. Сер. «Мат-лы к биобиблиографии ученых ИПИ РАН».
8. *Соколов И. А., Синицын И. Н., Доступова С. Б., Баженова Т. В., Захаров В. Н., Лавренюк Ю. А., Левина Г. А., Веревкин Г. Ф., Матов В. И., Ким Н. В., Половников В.А., Жуков Г. А., Гулянский Л. А., Коршунов А. А.* Борис Григорьевич Доступов / Под ред. И. А. Соколова, И. Н. Синицына. — М.: ИПИ РАН, 2010. 52 с. Сер. «Мат-лы к биобиблиографии ученых ИПИ РАН».

1.2 Монографии, вышедшие в других издательствах России

1. *Архипова М. Ю., Мхитарян В. С.* Использование нелинейных моделей в эконометрических исследованиях. — М.: МЭСИ, 2010. 91 с.
2. *Астафьева О. Н., Колин К. К.* Концептуальные основы государственной политики в области духовной культуры для обеспечения единства российского народа и национальной безопасности Российской Федерации. — Челябинск: ЧГАКИ, 2010. 67 с.
3. *Колин К. К.* Философские проблемы информатики. — М.: БИНОМ Лаборатория знаний, 2010. 264 с.
4. *Сейфуль-Мулюков Р. Б.* Нефть — углеводородные последовательности: анализ моделей генезиса и эволюции. — М.: 11-й формат, 2010. 173 с.

1.3 Монографии, изданные за рубежом

1. *Гуревич И. М.* Информационные характеристики физических систем. — Севастополь: Кипарис, 2010. 260 с.

2 УЧЕБНИКИ, УЧЕБНЫЕ ПОСОБИЯ

1. *Архипова М. Ю., Архипов В. Ю.* Денежные потоки в инвестиционной деятельности: Учебно-методический комплекс. — М.: ЕАОИ, 2010. 152 с.
2. *Колин К. К.* Информационное общество: Учебно-методическое пособие для вузов. — Челябинск: ЧГАКИ, 2010. 27 с.
3. *Фомичёв В. М.* Методы дискретной математики в криптологии: Учебно-справочное издание. — М.: Диалог-МИФИ, 2010. 424 с.

3 СТАТЬИ СОТРУДНИКОВ ИНСТИТУТА В ПЕРИОДИЧЕСКИХ ИЗДАНИЯХ ИПИ РАН

3.1 Ежегодник «Системы и средства информатики». Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010

1. *Степченков Ю. А., Дьяченко Ю. Г., Рождественский Ю. В., Морозов В. Н., Степченков Д. Ю.* Разработка вычислителя, не зависящего от задержек элементов // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 5–22.
2. *Зеленов Р. А., Степченков Ю. А., Волчек В. Н., Хилько Д. В., Шнейдер А. Ю., Прокофьев А. А.* Система капсульного программирования и

отладки // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 23–29.

3. *Степченков Ю. А., Волчек В. Н., Петрухин В. С., Прокофьев А. А., Зеленов Р. А.* Механизмы обеспечения поддержки алгоритмов цифровой обработки речевых сигналов в рекуррентном обработчике сигналов // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 30–46.
4. *Плеханов Л. П.* Полнота анализа электронных схем на самосинхронность // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 47–56.
5. *Захаров В. Н., Шмейлин Б. З.* Влияние кратковременных отказов на надежность кэш микропроцессорных систем // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 57–71.
6. *Архипов О. П., Зыкова З. П.* RGB-характеризация пространства цветовосприятия // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 72–89.
7. *Архипов О. П., Маньяков Ю. А., Сиротинин Д. О.* Метод регистрации морфинга трехмерного объекта на основе данных натурного эксперимента // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 90–104.
8. *Иванов А. В.* Математические модели базовых процессов функционирования информационного web-портала // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 105–131.
9. *Гринченко С. Н., Зацман И. М., Захаров В. Н.* К 75-летию заслуженного деятеля науки Российской Федерации К. К. Колина // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 160–163.
10. *Шоргин С. Я., Корепанов Э. Р., Синицын В. И.* К 70-летию заслуженного деятеля науки Российской Федерации И. Н. Синицына // Системы и средства информатики. Вып. 20. № 1. — М.: ТОРУС ПРЕСС, 2010. С. 164–168.

3.2 Ежегодник «Системы и средства информатики». Вып. 20. № 2. Специальный тематический сборник «Методы и технологии информатики, применяемые в научных исследованиях». — М.: ИПИ РАН, 2010

1. *Веревкин Г. Ф., Зацман И. М., Норекян Т. П., Хаванских В. А.* Принципы учета результатов научно-технической деятельности институтов РАН // Системы и средства информатики. Вып. 20. № 2. Методы и технологии

информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 5–22.

2. *Зацман И. М., Шубников С. К.* Методы верифицируемого оценивания целевых программ научных исследований // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 23–48.
3. *Кожунова О. С.* Семантический словарь: направление развития в системах информационного мониторинга // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 49–63.
4. *Лощилова Е. Ю.* Методы и технологии верификации данных для аналитических отчетов в системах информационного мониторинга // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 64–74.
5. *Лукьянов Г. В.* Проблема методологии оценивания результативности научных программ и проектов // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 75–87.
6. *Лунева Н. В.* Категории информационных ресурсов ИТСМ РАН // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 88–99.
7. *Никишин Д. А.* Методические и технологические решения для картографического представления индикаторов научной деятельности в информационно-технологической системе мониторинга РАН // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 100–112.
8. *Зацман И. М., Лукьянов Г. В.* Зарубежные подходы к выбору приоритетных научных направлений // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 113–123.
9. *Кузнецов И. П., Сомин Н. В.* Особенности настройки объектно-ориентированного лингвистического процессора на тексты предметной области // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 124–148.
10. *Лукьянов Г. В.* Проблемы защиты персональных данных в информационных системах сферы науки // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 149–159.

11. *Чавтараев Р. Б., Стефанович А. И.* Применение средств WPF при реализации инструментария для создания и публикации электронных коллекций // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 160–177.
12. *Маркова Н. А., Адамович И. М.* Коллекции Персоналий // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 178–198.
13. *Богданова Д. А. Федосеев А. А.* Цифровые образовательные ресурсы. Когда забывают о качестве // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 199–208.
14. *Вихрев В. В.* Новая парадигма информатизации образования: эволюция терминов в контексте проблемы терминологической размытости // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 209–239.
15. *Гуревич И. М.* Фундаментальные ограничения на информационную емкость устройств хранения данных // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 240–253.
16. *Дулин С. К., Розенберг И. Н., Уманский В. И.* Анализ неопределенности геоданных в моделях географического пространства // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 254–286.
17. *Зацман И. М., Бунтман Н. В.* Компьютерное кодирование целевых систем знаний и процессов их эволюции в лингвистике // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 287–302.
18. *Никишин Д. А.* Методы индексирования пространственных данных для представления в электронных геобиблиотеках // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 303–323.
19. *Сейфуль-Мулюков Р. Б., Арутюнов Е. Н.* Законы информатики в познании генезиса сложных природных систем на примере нефти // Системы и средства информатики. Вып. 20. № 2. Методы и технологии информатики, применяемые в научных исследованиях. — М.: ИПИ РАН, 2010. С. 324–340.

3.3 Ежегодник «Системы и средства информатики». Вып. 20. № 3. — М.: ИПИ РАН, 2010

1. *Френкель С. Л., Курц А. Л., Либуркин Д. Л., Фандюшина Н. А., Андерс Б. Н.* Транслятор табличных представлений автоматов Мили в программы на языке SMV для автоматизации верификации проектов вычислительных устройств на основе Проверки Моделей // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 4–16.
2. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Сигнальный механизм языка Cell // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 45–66.
3. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Язык Cell: модель обработки клонов // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 67–81.
4. *Бондаренко Т. В., Бондаренко О. А., Волович К. И., Кондрашев В. А.* Базовая модель функционирования автомата в системе программирования Cell // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 82–97.
5. *Босов А. В. Зацаринный А. А., Сучков А. П.* Некоторые общие подходы к формированию функциональных требований к ситуационным центрам и их реализации // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 98–125.
6. *Зацаринный А. А., Чупраков К. Г.* Об одном подходе к обоснованию требований к компонентам оборудования ситуационного центра // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 126–156.
7. *Зацаринный А. А., Гаранин А. И., Ионенков Ю. С.* Методический подход к обоснованию требований к надежности информационно-телекоммуникационных сетей // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 157–173.
8. *Зацаринный А. А., Гаранин А. И., Козлов С. В.* Стенд главного конструктора — организационно-техническая основа разработки крупномасштабных информационно-телекоммуникационных систем // Системы и средства информатики. Вып. 20. № 3. — М.: ИПИ РАН, 2010. С. 174–190.

3.4 Журнал «Информатика и её применения», 2010. Т. 4. Вып. 1–4 (включен в список ВАК)

1. *Синицын И. Н., Синицын В. И., Корепанов Э. Р., Белоусов В. В., Семендяев Н. Н.* Оперативное построение информационных моделей движения

полюса Земли методами линейных и линеаризованных фильтров // Информатика и её применения, 2010. Т. 4. Вып. 1. С. 2–11.

2. *Бенинг В. Е., Сипина А. В.* Асимптотическое разложение для мощности критерия, основанного на выборочной медиане, в случае распределения Лапласа // Информатика и её применения, 2010. Т. 4. Вып. 1. С. 18–23.
3. *Илюшин Г. Я., Соколов И. А.* Организация управляемого доступа пользователей к разнородным ведомственным информационным ресурсам // Информатика и её применения, 2010. Т. 4. Вып. 1. С. 24–40.
4. *Сейфуль-Мулюков Р. Б.* Нефть как носитель информации о своем происхождении, структуре и эволюции // Информатика и её применения, 2010. Т. 4. Вып. 1. С. 41–49.
5. *Коновалов М. Г.* О планировании потоков в системах вычислительных ресурсов // Информатика и её применения, 2010. Т. 4. Вып. 2. С. 3–12.
6. *Кривенко М. П.* Непараметрическое оценивание элементов байесовского классификатора // Информатика и её применения, 2010. Т. 4. Вып. 2. С. 13–24.
7. *Бенинг В. Е., Королёв Р. А.* О предельном поведении мощностей критериев в случае распределения Лапласа // Информатика и её применения, 2010. Т. 4. Вып. 2. С. 63–74.
8. *Козеренко Е. Б.* Лингвистические фильтры в статистических моделях машинного перевода // Информатика и её применения, 2010. Т. 4. Вып. 2. С. 83–92.
9. *Frenkel S. L., Pechinkin A. V.* Estimation of self-healing time for digital systems under transient faults // Информатика и её применения, 2010. Т. 4. Вып. 3. С. 2–8.
10. *Зейфман А. И., Коротышева А. В., Сатин Я. А., Шоргин С. Я.* Об устойчивости нестационарных систем обслуживания с катастрофами // Информатика и её применения, 2010. Т. 4. Вып. 3. С. 9–15.
11. *Кудрявцев А. А., Шоргин С. Я.* Байесовские модели массового обслуживания и надежности: характеристики среднего числа заявок в системе $M|M|1|\infty$ // Информатика и её применения, 2010. Т. 4. Вып. 3. С. 16–21.
12. *Зацаринный А. А., Чупраков К. Г.* Некоторые аспекты выбора технологии для построения систем отображения информации ситуационного центра // Информатика и её применения, 2010. Т. 4. Вып. 3. С. 59–68.
13. *Козеренко Е. Б., Кузнецов И. П.* Когнитивно-лингвистические представления в системах обработки текстов // Информатика и её применения, 2010. Т. 4. Вып. 3. С. 69–76.

14. *Buntman N., Minel J.-L., Le Pesant D., Zatsman I.* Typology and computer modeling of translation difficulties // Информатика и её применения, 2010. Т. 4. Вып. 3. С. 77–83.
15. *Соколов И. А.* О работах заслуженного деятеля науки Российской Федерации И. Н. Синицына в области информационных технологий и автоматизации (к 70-летию со дня рождения) // Информатика и её применения, 2010. Т. 4. Вып. 3. С. 84–86.
16. *Архипов О. П., Зыкова З. П.* Интеграция гетерогенной информации о цветных пикселях и их цветовосприятии // Информатика и её применения, 2010. Т. 4. Вып. 4. С. 15–26.
17. *Baranov S., Frenkel S., Zakharov V.* Semiformal verification for pipelined digital designs based on Algorithmic State Machines // Информатика и её применения, 2010. Т. 4. Вып. 4. С. 49–60.
18. *Колесников А. В., Солдатов С. А.* Алгоритм координации для гибридной интеллектуальной системы решения сложной задачи оперативно-производственного планирования // Информатика и её применения, 2010. Т. 4. Вып. 4. С. 61–67.
19. *Чупраков К. Г.* К вопросу о размещении коллективных средств отображения в ситуационном зале с заданными параметрами // Информатика и её применения, 2010. Т. 4. Вып. 4. С. 89–96.

3.5 Журнал «Системы высокой доступности», 2010. Т. 6. №№ 1–4 (включен в список ВАК)

1. *Козмидиади В. А.* Хранение очень больших объемов данных // Системы высокой доступности, 2010. Т. 6. № 1. С. 4–18.
2. *Абдулин Е. Р.* Разработка прикладного программного обеспечения для систем реального времени с заданными параметрами функционирования // Системы высокой доступности, 2010. Т. 6. № 1. С. 19–34.
3. *Синицын И. Н., Синицын В. И., Корепанов Э. Р., Белоусов В. В., Конашенкова Т. Д., Семендяев Н. Н., Сергеев И. В., Басилашвили Д. А.* Инструментальное программное обеспечение анализа и синтеза стохастических систем высокой доступности (II) // Системы высокой доступности, 2010. Т. 6. № 2. С. 4–45.
4. *Кривенко М. П.* Выборочный контроль при неполных данных // Системы высокой доступности, 2010. Т. 6. № 2. С. 46–52.
5. *Синицын В. И.* Рецензия на книгу В. М. Фомичёва «Методы дискретной математики в криптологии» // Системы высокой доступности, 2010. Т. 6. № 2. С. 53–54.

6. *Синицын В. И.* Рецензия на книгу А. С. Рыкова «Системный анализ: модели и методы принятия решений и поисковой оптимизации» // Системы высокой доступности, 2010. Т. 6. № 2. С. 55–56.
7. *Будзко В. И.* Руководящие принципы для авторов статей в журнале «Системы высокой доступности» // Системы высокой доступности, 2010. Т. 6. № 3. С. 5–12.
8. *Беленков В. Г.* Вопросы методического обеспечения построения перспективного КСА (VII) // Системы высокой доступности, 2010. Т. 6. № 3. С. 13–69.
9. *Абдулин Е. Р.* Решение задачи разработки прикладного программного обеспечения для интегрированных систем управления на базе операционных систем реального времени в условиях ограниченности временных ресурсов // Системы высокой доступности, 2010. Т. 6. № 4. С. 4–13.
10. *Синицын И. Н., Синицын В. И., Корепанов Э. Р., Белоусов В. В., Конашенкова Т. Д., Сергеев И. В., Агафонов Е. С., Басилашвили Д. А.* Инструментальное программное обеспечение анализа и синтеза стохастических систем высокой доступности (III) // Системы высокой доступности, 2010. Т. 6. № 4. С. 23–47.
11. *Белоусов В. В.* Состояние исследований в области разработки методического и программного обеспечения для систем управления и обработки информации // Системы высокой доступности, 2010. Т. 6. № 4. С. 48–62.

4 СТАТЬИ В ДРУГИХ ЖУРНАЛАХ И СБОРНИКАХ

4.1 Статьи, опубликованные в журналах, включенных в список ВАК

1. *Архипова М. Ю., Зацман И. М., Шульга С. Ю.* Индикаторы патентной активности в сфере информационно-коммуникационных технологий и методика их вычисления // Экономика, статистика и информатика. Вестник УМО, 2010. № 4. С. 93–104.
2. *Беленков В. Г.* Вопросы методического обеспечения построения перспективного КСА (VI) // Наукоемкие технологии, 2010. № 1. С. 38–72.
3. *Богданова Д. А.* Интернет-безопасность (опыт Австралии) // Дистанционное и виртуальное обучение, 2010. № 6. С. 117–128.
4. *Богданова Д. А., Федосеев А. А.* Внимание: Интернет! // Открытое образование, 2010. № 2. С. 89–99.
5. *Богданова Д. А., Федосеев А. А.* Зарубежные образовательные порталы: опыт Евросоюза, Великобритании, Австралии // Дистанционное и виртуальное обучение, 2010. № 5. С. 78–90.

6. *Бодякин В. И., Дубровский Д. И., Колин К. К., Лекторский В. А., Мелик-Гайказян И. В., Пружинин В. И., Урсул А. Д.* Информационный подход в междисциплинарной перспективе: Мат-лы круглого стола // Вопросы философии, 2010. № 2. С. 84–112.
7. *Борисов А. В., Босов А. В., Стефанович А. И.* Оптимальное оценивание показателей функционирования информационного web-портала // Автоматика и телемеханика, 2010. № 3. С. 16–33.
8. *Гончаров В. М., Бенинг В. Е.* Использование методов математического моделирования при агрофизической оценке почвенного покрова // Вестник Тверского государственного ун-та. Сер. Прикладная математика, 2010. № 16. С. 43–54.
9. *Горьков И. Д.* Свойства σ-периодических последовательностей // Наукоемкие технологии, 2010. № 1. С. 34–37.
10. *Гринченко С. Н.* Мировоззренческое значение современных концепций информатики // Открытое образование, 2010. № 6. С. 107–119.
11. *Гринченко С. Н., Щапова Ю. Л.* Модели периодизации истории человечества // Вестник РАН, 2010. № 12. С. 1076–1084.
12. *Гуревич И. М.* Информационные методы исследования физических систем: обзор первых достижений // Информационные технологии, 2010. № 12. С. 2–6.
13. *Гуревич И. М.* Информация как универсальная неоднородность // Информационные технологии, 2010. № 4. С. 66–74.
14. *Ильин В. Д.* Технология научной деятельности: подход к повышению продуктивности // Управление большими системами. Вып. 29. — М.: ИПУ РАН, 2010. С. 88–107.
15. *Касконе А., Мандзо Р., Печинкин А., Салерно С.* Система MAP/G/1/∞ в дискретном времени с инверсионной вероятностной дисциплиной обслуживания // Автоматика и телемеханика, 2010. № 12. С. 57–69.
16. *Кириков И. А., Колесников А. В., Листопад С. В.* Исследование эффекта самоорганизации в компьютерных системах поддержки принятия решения на примере многоагентных систем // Вестник Российского государственного ун-та им. Иммануила Канта. Вып. 10. Сер. Физико-математические науки. — Калининград: РГУ им. И. Канта, 2010. С. 79–90.
17. *Колин К. К.* Гуманитарные аспекты проблем национальной и международной безопасности // Открытое образование, 2010. № 1. С. 62–68.
18. *Колин К. К.* Духовная культура общества как стратегический фактор обеспечения национальной и международной безопасности // Вестник Челябинской государственной академии культуры и искусств, 2010. Т. 21. № 1. С. 27–45.

19. *Колин К. К.* Информационная культура и качество жизни в информационном обществе // Открытое образование, 2010. № 6. С. 84–89.

20. *Колин К. К.* Качество жизни в информационном обществе // Человек и труд, 2010. № 1. С. 39–43.

21. *Колин К. К.* Обращение к читателям // Вестник Челябинской государственной академии культуры и искусств, 2010. Т. 21. № 1. С. 6.

22. *Колин К. К.* Системная модернизация России и проблемы развития информационного общества // Государственная служба, 2010. № 2. С. 32–37.

23. *Колин К. К.* Философия информации и фундаментальные проблемы информатики // Информационные ресурсы России, 2010. № 1. С. 25–28.

24. *Колин К. К.* Философия информации и фундаментальные проблемы современной информатики // Alma mater (Вестник высшей школы), 2010. № 1. С. 29–35.

25. *Колин К. К., Буцык С. В.* О проблемах подготовки специалистов в сфере культуры и искусств в условиях формирования в России информационного общества // Информационное общество, 2010. № 6. С. 45–49.

26. *Королёв В. И.* Методология построения комплексной защиты информации на объектах информатизации // Наукоемкие технологии, 2010. № 1. С. 4–24.

27. *Королев В. Ю., Назаров А. Л.* Разделение смесей вероятностных распределений при помощи сеточных методов моментов и максимального правдоподобия // Автоматика и телемеханика, 2010. № 3. С. 98–116.

28. *Королев В. Ю., Шевцова И. Г.* Новая моментная оценка скорости сходимости в теореме Ляпунова // Теория вероятностей и ее применения, 2010. Т. 55. Вып. 3. С. 577–582.

29. *Королев В. Ю., Шевцова И. Г.* Уточнение верхней оценки абсолютной постоянной в неравенстве Берри–Эссеена для смешанных пуассоновских случайных сумм // Докл. РАН, 2010. Т. 431. Вып. 1. С. 16–19.

30. *Королев В. Ю., Шевцова И. Г.* Уточнение неравенства Берри–Эссеена // Докл. РАН, 2010. Т. 430. Вып. 6. С. 738–742.

31. *Королев В. Ю., Шевцова И. Г.* Уточнение неравенства Берри–Эссеена с приложениями к пуассоновским и смешанным пуассоновским случайным суммам // Обозрение прикладной и промышленной математики, 2010. Т. 17. Вып. 1. С. 25–56.

32. *Марков Ю. Г., Перепелкин В. В., Синицын И. Н., Семендяев Н. Н.* Вращательно-колебательное движение Земли и глобальная составляющая сейсмического процесса // Докл. РАН, 2010. Т. 435. № 3. С. 1–5.

33. *Марков Ю. А., Рыхлова Л. В., Синицын И. Н.* Развитие методов построения моделей движения полюса Земли // Астрономический журнал, 2010. № 9. С. 111–120.
34. *Марков Ю. Г., Синицын И. Н.* Вероятностные модели флуктуаций неравномерности вращения Земли при нестационарных возмущениях // Докл. РАН, 2010. Т. 432. № 3. С. 332–336.
35. *Марков Ю. Г., Синицын И. Н., Синицын В. И., Корепанов Э. Р., Семендяев Н. Н.* Опыт построения моделей флуктуаций полюса Земли на основе астрономических измерений // Автоматика и телемеханика, 2010. № 3. С. 87–97.
36. *Печинкин А. В., Стальченко И. В.* Система MAP/G/1/∞ с инверсионным порядком обслуживания и вероятностным приоритетом, функционирующая в дискретном времени // Вестник Российского ун-та дружбы народов. Сер. Математика. Информатика. Физика. № 2. — М.: РУДН, 2010. ISSN 0869-8732. С. 26–36.
37. *Розенберг И. Н., Дулин С. К.* Геоинформационный портал отрасли. Гарантировать достоверность данных // Железнодорожный транспорт, 2010. № 2. С. 12–17.
38. *Сейфуль-Мулюков Р. Б.* Нефтеродные функции нефтематеринских свит: неочевидность догмы // Известия высших учебных заведений: Геология и разведка, 2010. № 4. С. 60–65.
39. *Синицын И. Н.* Компьютерная математика для радиоинженеров (к семидесятилетию В. П. Дьяконова) // Наукоемкие технологии, 2010. № 10. С. 63–64.
40. *Ушаков В. Г., Ушаков Н. Г.* Неравенства для среднеквадратичной ошибки многомерных ядерных оценок плотности // Вестник Московского ун-та. Сер. 15. Вычислительная математика и кибернетика, 2010. № 1. С. 22–26.
41. *Ушмаев О. С.* Концепция мультибиометрической идентификации в информационно-аналитических приложениях // Вестник компьютерных и информационных технологий, 2010. № 6. С. 31–37.
42. *Фомичёв В. М.* Краткий обзор материалов 8-й школы-семинара SIBECRYPT'09 // Наукоемкие технологии, 2010. № 1. С. 25–27.
43. *Фомичёв В. М.* Свойства h-периодических последовательностей // Прикладная дискретная математика. — Томск: НТЛ, 2010. № 2(8). С. 16–21.
44. *Фомичёв В. М.* Свойства путей в графах и в мультиграфах // Прикладная дискретная математика. — Томск: НТЛ, 2010. № 1(7). С. 118–124.
45. *Фомичёв В. М., Фомичёв Н. В.* Исследование линейных подсистем нелинейных систем уравнений гаммообразования // Наукоемкие технологии, 2010. № 1. С. 28–33.

46. *Шмейлин Б. З.* Организация тестирования Internet-приложений // Вопросы радиоэлектроники, 2010. Вып. 2. С. 22–29.

4.2 Статьи, опубликованные в научных сборниках и журналах, не включенных в список ВАК

1. *Архипов О. П.* Штрихи к портрету Игоря Александровича Мизина // Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. С. 228–231.
2. *Архипов О. П., Зыкова З. П.* Функциональное описание индивидуального цветовосприятия // Известия ОрелГТУ. Сер. Информационные системы и технологии, 2010. № 5. С. 5–12.
3. *Архипова М. Ю.* Проблемы коммерциализации радикальных изобретений // Математико-статистический анализ социально-экономических процессов: Межвузовский сборник научных трудов. — М.: МЭСИ, 2010. Вып. 7. С. 12–16.
4. *Архипова М. Ю.* Проблемы оценивания влияния научных исследований на развитие технологий // Математико-статистический анализ социально-экономических процессов: Межвузовский сборник научных трудов. — М.: МЭСИ, 2010. Вып. 7. С. 16–18.
5. *Вдовицын В. Т., Калиниченко Л. А., Когаловский М. Р., Кузнецов С. Д., Мазалов В. В.* // XI Всеросс. науч. конф. RCDL'2009 «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» (Петрозаводск, 17–21 сентября 2009 г.). Труды Карельского научного центра Российской академии наук, 2010. № 3. С. 98–100.
6. *Гавриленко С. В., Королев В. Ю.* Об оценках вероятности разорения страховой компании, резерв которой описывается классическим процессом риска // Статистические методы оценивания и проверки гипотез. Вып. 22. — Пермь: ПГУ, 2010. С. 137–147.
7. *Гринченко С. Н.* Историческое познание и кибернетика: о предсказуемости-непредсказуемости процессов развития Человечества // Проблемы исторического познания. — М.: Институт всеобщей истории РАН, 2010. С. 22–44.
8. *Гринченко С. Н.* Мировой кризис: финансово-экономический, мировоззренческий или системный? // Научное, экспертно-аналитическое и информационное обеспечение национального стратегического проектирования, инновационного и технологического развития России. Ч. 1. — М.: ИНИОН, 2010. С. 566–569.
9. *Гринченко С. Н.* Об оптимизации пространственно-временных масштабов компонентов социально-технологической системы // Социальные процессы

и технологии: моделирование и управление: Сборник научных трудов. — М.: МИГКУ, 2010. С. 37–41.

10. *Гринченко С. Н.* Пространство и время с позиции кибернетики. Ч. 1 // Пространство и время, 2010. № 2. С. 43–54.

11. *Гринченко С. Н.* Социально-технологическая система Человечества в эпоху глобализации: геополитические аспекты // Погранология: Мат-лы постоянно действующего межведомственного научного семинара. — М.: Пограничная академия ФСБ России, 2010. № 2. С. 30–49.

12. *Гринченко С. Н.* Универсальный эволюционизм — на информатико-кибернетическом языке // Международный научный альманах. Вып. 7. — М.: МНЭПУ, 2010. С. 386–392.

13. *Гринченко С. Н.* Целевой подход при моделировании системы Мироздания // Биокосмология (Biocosmology) — neo-Aristotelism: Электронный журнал, 2010. 18 с. URL: http://www.biocosmology.ru/elektronnyj-zurnal-biokosmologia- biocosmology-neo-Aristotelism/postupivsie-stati.

14. *Гринченко С. Н.* Человек, технологии его интеллектуализации и развивающееся Человечество // Философские проблемы биологии и медицины. Вып. 4. Фундаментальное и прикладное. — М.: Принтберри, 2010. С. 197–202.

15. *Гринченко С. Н.* Человеческий потенциал России: реалии и теоретические потребности // Россия: тенденции и перспективы развития. Ежегодник. — М.: ИНИОН РАН, 2010. Вып. 5. Ч. I. С. 91–93.

16. *Гринченко С. Н.* Этапы формирования информационного общества и информационная безопасность // Погранология: Мат-лы постоянно действующего межведомственного научного семинара. — М.: Пограничная академия ФСБ России, 2010. № 3. С. 34–42.

17. *Гринченко С. Н., Барыбина И. А.* «Системная картина Мироздания»: программа курса высшего и послевузовского образования // Вестник КемГУКИ, 2010. № 12. С. 52–61.

18. *Егоров В. Б.* Архитектурные инновации в многоядерных ИКМ QorIQ // Электронные компоненты, 2010. № 10. С. 66–72.

19. *Захаров В. Н.* Воспоминания о директоре // Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. С. 223–227.

20. *Зацаринный А. А.* Академик И. А. Мизин: военная наука и практика // Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. С. 96–128.

21. *Зацаринный А. А.* Воспоминания в эпизодах // Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. С. 216–220.
22. *Колин К. К.* Глобализация общества и нравственное измерение современного кризиса цивилизации // Вестник Кемеровского государственного ун-та культуры и искусств, 2010. № 10. С. 10–16.
23. *Колин К. К.* Гуманитарные проблемы информационной безопасности // Погранология: Мат-лы постоянно действующего межведомственного научного семинара. — М.: Пограничная академия ФСБ России, 2010. № 3. С. 108–141.
24. *Колин К. К.* Гуманитарные проблемы формирования информационного общества // Вестник Кемеровского государственного ун-та культуры и искусств, 2010. № 12. С. 8–19.
25. *Колин К. К.* Информатизация общества в России и роль в ней академика И. А. Мизина // Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. С. 188–192.
26. *Колин К. К.* Обращение к читателям // Вестник Кемеровского государственного ун-та культуры и искусств, 2010. № 1. С. 6–7.
27. *Колин К. К.* У истоков российской философии информации // Природа информации: Философский очерк / Под ред. А. Д. Урсула. — Челябинск, 2010. С. 5–14.
28. *Колин К. К.* Философия информации — актуальное направление исследований в области философии науки // Вестник Кемеровского государственного ун-та культуры и искусств, 2010. № 12. С. 43–51.
29. *Колин К. К., Буцык С. В.* Об основных направлениях деятельности Научно-образовательного центра по комплексной проблеме «Информационное общество» в вузе культуры и искусств // Вестник Кемеровского государственного ун-та культуры и искусств, 2010. № 13. С. 6–9.
30. *Синицын И. Н.* О развитии определения «информатика» в 1980-х годах // Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. С. 184–187.
31. *Скворцов Н. А.* Специфика подходов к отображению онтологий // Вопросы искусственного интеллекта (Вестник НСМИИ РАН), 2010. № 2. С. 82–88.
32. *Соколов И. А.* Вклад академика И. А. Мизина в развитие отечественных информационных технологий и систем // Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. С. 85–95.

33. *Темнов А. И.* И. А. Мизин в спорте // Игорь Александрович Мизин — ученый, конструктор, человек / Под ред. акад. И. А. Соколова. — М.: ИПИ РАН, 2010. С. 234–237.

34. *Чаплыгин В. В.* Многолинейная система массового обслуживания с конечным накопителем, блокировкой полумарковского потока заявок и выбиванием всех заявок из системы // Информационные процессы, 2010. Т. 10. № 3. С. 224–236.

4.3 Статьи, опубликованные в журналах, изданных за рубежом

1. *Кузнецов И. П. Козеренко Е. Б., Мацкевич А. Г.* Принципы организации объектно-ориентированных систем обработки неформализованной информации // Искусственный интеллект: Журнал НАН Украины, 2010. Вып. 3. С. 227–237.

2. *Chaplygin V. V.* A multiserver queueing system with a discontinuous semi-Markovian demand flow and demand removal from an infinite capacity storage // J. Communications Technology and Electronics, 2010. Vol. 55. No. 12. P. 1491–1498.

3. *Gurevich I. M.* Optimal black holes are the cosmological objects, which minimize volume of information in areas of the Universe and in the Universe as a whole // Cosmology and Extragalactic Astrophysics. — Cornell University Library, 5 Aug 2010. 15 p. — URL: http://arxiv.org/abs/1008.0947v1.

4. *Korolev V., Shevtsova I.* An improvement of the Berry–Esseen inequality with applications to Poisson and mixed Poisson random sums // Scandinavian Actuarial J., 2010. Online first: http://www.informaworld.com/10.1080/03461238.2010.485370. June 04, 2010.

5. *Miller B., Miller G., Siemenikhin K.* Towards the optimal control of Markov chains with constraints // Automatica, 2010. Vol. 46. P. 1495–1502.

6. *Torchigin V. P., Torchigin A. V.* Ball lightning as an optical incoherent space spherical soliton // Handbook of solitons: Research, technology and applications / Eds. S. P. Lang and Salim H. Bedore. — New York: Novapublishers, 2010. P. 3–54.

7. *Torchigin V. P., Torchigin A. V.* On phenomenon of light radiation from miniature balls immersed in water // Physics Letters A, 2010. Vol. 374. P. 588–591.

8. *Van Doorn E. A., Zeifman A. I., Panfilova T. L.* Bounds and asymptotics for the rate of convergence of birth-death processes // Theory Probab. Appl., 2010. Vol. 54. P. 97–113.

5 ДОКЛАДЫ

5.1 Доклады, опубликованные в трудах конференций и других научных мероприятий, проведенных в России

1. *Арутюнян А. Р., Ушмаев О. С.* Деформации изображений отпечатков пальцев // Кибернетика и высокие технологии XXI века (C&T-2010): Сборник докладов XI Междунар. науч.-техн. конф. (Воронеж, 12–14 мая 2010 г.). — Воронеж: Саквоее, 2010. Т. 2. С. 621–626.
2. *Архипова М. Ю., Архипов К. В.* Моделирование региональной структуры предприятия в условиях кризиса // Управление инновациями — 2010: Мат-лы V Междунар. науч.-практич. конф. — М.: ЛЕНАНД, 2010. С. 210–215.
3. *Архипова М. Ю., Сиротин В. П.* Сектор информационно-коммуникационных технологий как фактор инновационного развития российской экономики // Реструктурирование экономики: ресурсы и механизмы: Междунар. науч.-практич. конф. (Санкт-Петербург, 25–27 января 2010 г.). — СПб.: СПбГУ, 2010. С. 128–129.
4. *Афанасьев А. П., Калиниченко Л. А., Посыпкин М. А., Ступников С. А., Сухомлин В. А., Сухорослов О. В.* Подготовка кадров в сфере грид-технологий и распределенного компьютинга // Распределенные вычисления и грид-технологии в науке и образовании: Труды 4-й Междунар. конф. (Дубна, 28 июня – 3 июля 2010 г.). — Дубна: ОИЯИ, 2010. С. 284–289.
5. *Богданова Д. А., Федосеев А. А.* Образовательные порталы для школ на примере портала Евросоюза и Российской единой коллекции цифровых образовательных ресурсов // Информатизация образования — 2010: Мат-лы Междунар. науч.-метод. конф. (Кострома, 14–17 июня 2010 г.). — Кострома: КГУ им. Н. А. Некрасова, 2010. С. 328–333.
6. *Боярский К. К., Каневский Е. А., Лезин Г. В., Калиниченко Л. А., Скворцов Н. А.* Автоматизация процесса извлечения онтологической информации из вербальных терминологических словарей (на примере терминологического словаря задачи межзвездного поглощения) // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 257–264.
7. *Быстров И. И., Радоманов С. И., Тимофеев А. Е.* Сравнительный анализ программного обеспечения управления проектами на примере исследований по развитию системы управления проектной деятельностью в сфере информатизации Банка России // От теории к практике управления проектами: IX Конф. ПМСОФТ по управлению проектами (Москва, 27–28 мая 2010 г.): Сборник докладов. — М.: Университет управления проектами, 2010. С. 17.

8. *Вихрев В. В.* Смена парадигмы: от электронного учебника к ЦОРу // Новые образовательные технологии в вузе (НОТВ-2010): Сборник мат-лов VII Междунар. науч.-метод. конф. (Екатеринбург, 8–10 февраля 2010 г.). Ч. 2. — Екатеринбург: УГТУ-УПИ им. Б. Н. Ельцина, 2010. С. 20–24. URL: http://dist.ustu.ru/ioit/show.asp?file=notv2010.

9. *Вихрев В. В., Шпакова Т. Ю.* Компьютерное творчество учителей как ресурс информатизации образования // Новые образовательные технологии в вузе (НОТВ-2010): Сборник мат-лов VII Междунар. науч.-метод. конф. (Екатеринбург, 8–10 февраля 2010 г.). Ч. 2. — Екатеринбург: УГТУ-УПИ им. Б. Н. Ельцина, 2010. С. 24–28. URL: http://dist.ustu.ru/ioit/show.asp?file= notv2010.

10. *Вовченко А. Е.* Сопряжение языков программирования с предметными посредниками для решения научных задач // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 368–372.

11. *Вовченко А. Е., Захаров В. Н., Калиниченко Л. А., Ковалёв Д. Ю., Рябухин О. В., Скворцов Н. А., Ступников С. А.* От спецификаций требований к концептуальной схеме // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 375–382.

12. *Вовченко А. Е., Калиниченко Л. А., Костюков М. Ю.* Методы и средства доступа к потоковым данным из предметных посредников // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 273–278.

13. *Вовченко А. Е., Калиниченко Л. А., Ступников С. А.* Семантический грид, основанный на концепции предметных посредников // Распределенные вычисления и грид-технологии в науке и образовании: Труды 4-й Междунар. конф. (Дубна, 28 июня – 3 июля 2010 г.). — Дубна: ОИЯИ, 2010. С. 309–318.

14. *Горелиц О. В.* Влияние водохозяйственных мероприятий на режим стока Волги и ее дельты // Труды V Всесоюзн. гидрологического съезда. Т. 9. — СПб.: Гидрометеоиздат, 2010. С. 53–58.

15. *Гринченко С. Н.* Мировой кризис: финансово-экономический и системный? // По ту сторону кризиса: модернизационный потенциал фундаментального образования, науки и культуры: Мат-лы конф. (19–20 апреля 2010 г.) / Под общ. ред. А. Колганова и Р. Крумма. — М.: Культурная революция, 2010. С. 397–401.

16. *Гринченко С. Н.* Пред-история и пост-история человечества: что это такое? // Футурологический конгресс: будущее России и мира: Мат-лы Всеросс. науч. конф. (Москва, 4 июня 2010 г.). — М.: Научный эксперт, 2010. С. 395–402.

17. *Гринченко С. Н.* Прогресс Человечества и Россия // Россия в мире: гуманитарное, политическое и экономическое измерение: Мат-лы Всеросс. конф. (Москва, 19 марта 2010 г.). — М.: Научный эксперт, 2010. С. 186–195.

18. *Гринченко С. Н.* Человечество и язык с кибернетических позиций // Язык и общество в современной России и других странах: Междунар. конф. (Москва, 21–24 июня 2010 г.): Доклады и сообщения / Отв. ред. В. А. Виноградов, В. Ю. Михальченко. — М.: Институт языкознания РАН, 2010. С. 99–103.

19. *Дулин С. К., Дулина Н. Г., Уманский В. И.* Интеллектуализация формирования ресурсов геоинформационного портала // Труды XII национальной конф. по искусственному интеллекту КИИ-2010 (Тверь, 20–24 сентября 2010 г.). Т. 1. — М.: Физматлит, 2010. С. 223–231.

20. *Дьяченко Ю. Г., Морозов Н. В., Степченков Д. Ю.* Характеризация псевдодинамических элементов // Проблемы разработки перспективных микро- и наноэлектронных систем: IV Всеросс. науч.-техн. конф. (МЭС-2010): Сборник научных трудов / Под общ. ред. А. Л. Стемпковского. — М.: ИППМ РАН, 2010. С. 32–35.

21. *Кожунова О. С.* Семантический словарь системы информационного мониторинга и элементы формализации его статей // Труды XII национальной конф. по искусственному интеллекту КИИ-2010 (Тверь, 20–24 сентября 2010 г.). Т. 1. — М.: Физматлит, 2010. С. 209–217.

22. *Козеренко Е. Б., Кузнецов И. П.* Эволюция лингво-семантических представлений в интеллектуальных системах на основе расширенных семантических сетей // Компьютерная лингвистика и интеллектуальные технологии: По материалам ежегодной Междунар. конф. «Диалог» (Бекасово, 26–30 мая 2010). Вып. 9(16). — М.: РГГУ, 2010. С. 205–212.

23. *Колин К. К.* Духовная культура как стратегический фактор национальной и международной безопасности // Вестник Международной академии наук (Русская секция), 2010. Спец. вып. «Ученые России за интеллектуально-нравственное развитие общества»: Мат-лы симпозиума. С. 17–22.

24. *Колин К. К.* Стратегия модернизации России и актуальные проблемы формирования информационного общества // Информационно-коммуникативные технологии в системе культурно-цивилизационных преобразований: Мат-лы Всеросс. науч. конф. (Челябинск, 21 октября 2010 г.). — Челябинск: ЧГАКИ, 2010. С. 28–46.

25. *Кудрявцев А. А., Шоргин С. Я.* Байесовские модели массового обслуживания и надежности: постановка задачи и примеры // XI Всеросс. симпозиум по прикладной и промышленной математике (осенняя открытая сессия): Научные доклады. Ч. I. Обозрение прикладной и промышленной математики, 2010. Т. 17. Вып. 4. С. 571–575.

26. *Кузнецов И. П., Сомин Н. В.* Особенности лексико-морфологического анализа при извлечении информационных объектов и связей из текстов естественного языка // Компьютерная лингвистика и интеллектуальные технологии: По мат-лам ежегодной Междунар. конф. «Диалог» (Бекасово, 26–30 мая 2010 г.). Вып. 9(16). — М.: РГГУ, 2010. С. 254–264.

27. *Маркова Н. А., Адамович И. М.* Электронные биографические ресурсы // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 168–180.

28. *Плеханов Л. П.* Проектирование самосинхронных схем: функциональный подход. Подсистема событийного анализа самосинхронных схем АСПЕКТ // Проблемы разработки перспективных микро- и наноэлектронных систем: IV Всеросс. науч.-техн. конф. (МЭС-2010): Сборник научных трудов / Под общ. ред. А. Л. Стемпковского. — М.: ИППМ РАН, 2010. С. 424–429.

29. *Рождественский Ю. В., Морозов В. Н., Рождественскене А. В.* Подсистема событийного анализа самосинхронных схем АСПЕКТ // Проблемы разработки перспективных микро- и наноэлектронных систем: IV Всеросс. науч.-техн. конф. (МЭС-2010): Сборник научных трудов / Под общ. ред. А. Л. Стемпковского. — М.: ИППМ РАН, 2010. С. 26–31.

30. *Рябухин О. В.* Полуавтоматическое GLAV-отображение спецификаций в канонической информационной модели предметных посредников // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 356–361.

31. *Синицын И. Н., Сергеев И. В.* Методическое обеспечение измерения, контроля и испытаний вычислительного оборудования в условиях ударных воздействий // Технические и программные средства систем управления, контроля и измерения (УКИ-2010): Труды конф. (Москва, 18–22 октября 2010 г.). — М.: ИПУ РАН, 2010. Секция 4. С. 1–12.

32. *Синицын И. Н., Синицын В. И., Корепанов Э. Р., Белоусов В. В., Сергеев И. В.* Компьютерное моделирование стохастических систем на базе канонических разложений // Кибернетика и высокие технологии XXI века (C&T-2010): Сборник докладов XI Междунар. науч.-техн. конф. (Воронеж, 12–14 мая 2010 г.). — Воронеж: Саквоее, 2010. Т. 2. С. 798–809.

33. *Степченков Ю. А., Волчек В. Н., Петрухин В. С., Прокофьев А. А., Зеленов Р. А.* Цифровой сигнальный процессор с нетрадиционной рекуррентной потоковой архитектурой // Проблемы разработки перспективных микро- и наноэлектронных систем: IV Всеросс. науч.-техн. конф. (МЭС-2010): Сборник научных трудов / Под общ. ред. А. Л. Стемпковского. — М.: ИППМ РАН, 2010. С. 283–288.

34. *Степченков Ю. А., Дьяченко Ю. Г., Рождественский Ю. В., Морозов В. Н., Степченков Д. Ю.* Самосинхронный вычислитель для высоконадежных применений // Проблемы разработки перспективных микро- и наноэлектронных систем: IV Всеросс. науч.-техн. конф. (МЭС-2010): Сборник научных трудов / Под общ. ред. А. Л. Стемпковского. — М.: ИППМ РАН, 2010. С. 418–423.

35. *Ступников С. А.* Семантическая трансформация канонической информационной модели в формальный язык спецификаций для верификации уточнения // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 383–391.

36. *Ступников С. А., Скворцов Н. А.* Взаимное отображение канонической информационной модели и языка OWL 2 // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 392–398.

37. *Agalarov Y.* Algorithm of nodes load estimation in the network with repetitions from source and static buffer management scheme // Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT): 2010 International Congress (Moscow, October 18–20, 2010). — Moscow: IEEE, 2010. ISSN 2157-0221. ISBN 978-1-4244-7285-7. P. 1073–1077.

38. *Atencia I., Pechinkin A.* A discrete-time Geo/G/1/∞ queueing system with total renewal discipline // Distributed Computer and Communication Networks (DCCN 2010): Workshop (International) Proceedings (Moscow, October 26–28, 2010). — Moscow: Information and Networking Technologies, 2010. ISBN 978-5-9901871-2-2. P. 36–43.

39. *Kalinichenko L. A., Stupnikov S. A., Vovchenko A. E.* Mediation based semantic grid // Distributed Computing and Grid-Technologies in Science and Education (GRID'2010): 4th Conference (International) Proceedings (Dubna, June 28 – July 3, 2010). — Dubna: JINR, 2010. P. 309–318.

40. *Konovalov M.* Multiagent model for jobs flows planning and pricing in distributed computing systems // Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT): 2010 International Congress (Moscow,

October 18–20, 2010). — Moscow: IEEE, 2010. ISSN 2157-0221. ISBN 978-1-4244-7285-7. P. 1121–1124.

41. *Kudryavtsev A., Shorgin S.* On the Bayesian approach to the analysis of queueing systems and reliability // Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT): 2010 International Congress (Moscow, October 18–20, 2010). — Moscow: IEEE, 2010. ISSN 2157-0221. ISBN 978-1-4244-7285-7. P. 1042–1045.

42. *Pechinkin A., Razumchik R.* Waiting characteristics of queueing system Geo/Geo/1/∞ with negative claims and a bunker for superseded claims in discrete time // Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT): 2010 International Congress (Moscow, October 18–20, 2010). — Moscow: IEEE, 2010. ISSN 2157-0221. ISBN 978-1-4244-7285-7. P. 1051–1055.

43. *Sinitsyn I.* Canonical expansions of random functions and its applications in the scientific support stochastic informational technologies // 10th Conference (International) on Pattern Recognition and Image Analysis: New Information Technologies (PRIA-10-2010) Proceedings (St. Petersburg, December 5–12, 2010). Vol. I. — SPb.: Politechnika, 2010. P. 77–78.

44. *Ushmaev O., Sinitsyn I.* Multimodal biometrics: Empirical study of performance-throughput trade-off // 10th Conference (International) on Pattern Recognition and Image Analysis: New Information Technologies (PRIA-10-2010) Proceedings (St. Petersburg, December 5–12, 2010). Vol. II. — SPb.: Politechnika, 2010. P. 221–224.

45. *Zeifman A., Korotysheva A., Satin Ya.* On stability for $M(t)/M(t)/N/N/$ queue // Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT): 2010 International Congress (Moscow, October 18–20, 2010). — Moscow: IEEE, 2010. ISSN 2157-0221. ISBN 978-1-4244-7285-7. P. 1102–1105. URL: http://dx.doi.org/10.1109/ICUMT.2010.5676515.

5.2 Доклады, опубликованные в трудах конференций и других научных мероприятий, проведенных за рубежом

1. *Архипова М. Ю.* Оценка программной деятельности: основные методы и приемы // Acta Universitatis Pontica Euxinus. Спец. вып. Стратегия качества в промышленности и образовании: Труды VI Междунар. конф. (Варна, Болгария, 4–11 июня 2010 г.). Т. 2. — Варна: Технический ун-т; Киев: ДИПОпром, 2010. С. 370–375.

2. *Богданова Д. А.* Образовательные порталы для школ — ожидания и реальность // Современное электронное образование (Modern e-Learning —

MeL-2010): Мат-лы V Междунар. конф. (Киев, Украина, 6–11 сентября 2010 г.). — Sofia: c/o Jusautor, 2010. С. 323–331.

3. *Быстров И. И., Радоманов С. И.* Методологические основы управления программой развития крупных ИТ-систем // Системный анализ, управление и навигация: Тезисы докладов XIV Междунар. конф. (Украина, Крым, Евпатория, 27 июня – 4 июля 2010 г.). — М.: МАИ-ПРИНТ, 2010. С. 91–92.
4. *Гуревич И. М.* Разработка и применение информационных методов исследования физических систем // Современные проблемы прикладной математики, информатики и автоматизации: Мат-лы Междунар. науч.-практич. семинара (Севастополь, 4–7 октября 2010 г.). — Севастополь: СевНТУ, 2010. С. 8–14.
5. *Колин К. К.* Духовная культура и проблемы национальной и международной безопасности // Н. П. Румянцев и полиэтнические проблемы современности: Мат-лы Междунар. конф. (Гомель, Беларусь, 6–8 апреля 2010 г.). — М.: РГБ, 2010. С. 8–16.
6. *Кузнецов И. П. Козеренко Е. Б., Мацкевич А. Г.* Принципы организации объектно-ориентированных систем обработки неформализованной информации // Искусственный интеллект. Интеллектуальные системы (ИИ-2010): Мат-лы Междунар. науч.-техн. конф. (пос. Кацивели, Крым, Украина, 20–24 сентября 2010 г.). Т. 1. — Донецк: Наука і освіта, 2010. С. 205–210.
7. *Лунева Н. В.* Семантические функции метаданных в многоязычной лингвистической базе знаний // Искусственный интеллект. Интеллектуальные системы (ИИ-2010): Мат-лы Междунар. науч.-техн. конф. (пос. Кацивели, Крым, Украина, 20–24 сентября 2010 г.). Т. 1. — Донецк: Наука і освіта, 2010. С. 214–218.
8. *Синицын И. Н.* Канонические разложения случайных функций в задачах компьютерной поддержки научных исследований // Современные проблемы прикладной математики, информатики и автоматизации: Мат-лы Междунар. науч.-практич. семинара (Севастополь, 4–7 октября 2010 г.). — Севастополь: СевНТУ, 2010. С. 3–7.
9. *Arkhipova M., Sirotin V.* Small and medium enterprises in Russia: Main tendencies structure and features of development // Technology as the Foundation for Economic Growth: 19th Conference (International) Proceedings (Cairo, Egypt, March 8–11, 2010). — Miami, USA: IAMOT, 2010. P. 203–205.
10. *Arkhipova M., Sirotin V.* Small enterprises and innovative development of the country // Современные проблемы управления предприятием: Мат-лы III Междунар. конф. (Бельско-Бяла, Польша, 10–12 июня 2010 г.). — Bielsko-Biala: ATH, 2010. P. 269–288.

11. *Cascone A., Manzo R., Pechinkin A. V., Shorgin S. Ya.* A Geom/G/1/n queueing system with LIFO discipline, service interruptions and resumption, and restrictions on the total volume of demands // WCE 2010: World Congress on Engineering Proceedings (London, June 30 – July 2, 2010). Vol. III. — London: WCE, 2010. P. 1765–1769.
12. *Durnovo A., Zatsman I.* Semiotic models for cognitive processing of language information about translation difficulties // Cognitive Modeling in Linguistics: XII Conference (International) Proceedings (Dubrovnik, Croatia, September 7–14, 2010). — Kazan: KSU, 2010. P. 135–139.
13. *Gurevich I. M.* Optimal black holes // 61st Astronautical Congress (International) Proceedings (Prague, Czech Republic, September 27 – October 1, 2010). 8 p.
14. *Kolesnikov A., Listopad S., Kirikov I.* Investigation of self-organization phenomena and processes in decision support systems according to relationships of participants' goals // Information Technologies 2010 (IT 2010): 16th Conference (International) on Information and Software Technologies Proceedings (Kaunas, Lithuania, April 21–23, 2010). — Kaunas (Lithuania): Kaunas University of Technology, 2010. P. 88–94.
15. *Kozerenko E. B.* Linguistic constraints for statistical machine translation // WORLDCOMP'10: ICAI-10 Proceedings (Las Vegas, Nevada, USA, July 14–17, 2010). — CRSEA Press, USA, 2010. Vol. I. P. 927–933.
16. *Kozhunova O.* Semantic dictionary: Heading toward ontology of the Information Monitoring System // WORLDCOMP'10: ICAI-10 Proceedings (Las Vegas, Nevada, USA, July 14–17, 2010). — USA: CRSEA Press, 2010. Vol. I. P. 934–941.
17. *Kuznetsov I. P., Kozerenko E. B., Matskevich A. G.* Deep and shallow semantic presentations in intelligent fact extractors // WORLDCOMP'10: ICAI-10 Proceedings (Las Vegas, Nevada, USA, July 14–17, 2010). — USA: CRSEA Press, 2010. Vol. I. P. 921–926.
18. *Pechinkin A., Shorgin S.* A $\mathrm{Geo}_m/\mathrm{G}/1/n$ queueing system with LIFO discipline, service interruptions and repeat again service, and restrictions on the total volume of demands // LNCS. Vol. 6235. Multiple Access Communication (MACOM 2010): 3rd Workshop (International) Proceedings (Barcelona, Spain, September 2010). — Berlin–Heidelberg–New York: Springer-Verlag, 2010. P. 98–106.
19. *Rumovskaya S., Kolesnikov A., Kirikov I.* Knowledge-based system for decision making support at diagnosing of the arterial hypertension // JCKBSE'10: 9th Joint Conference on Knowledge-Based Software Engineering Proceedings (Kaunas, Lithuania, August 25–27, 2010). — Kaunas (Lithuania): Kaunas University of Technology, 2010. P. 55–67.

20. *Stepchenkov Yu., Diachenko Yu., Zakharov V., Rogdestvenski Yu., Morozov N., Stepchenkov D.* Quasi-delay-insensitive computing device: Methodological aspects and practical implementation // LNCS 5953, Integrated Circuit and System Design, Power and Timing Modeling, Optimization and Simulation: 19th Workshop (International), PATMOS 2009 (Delft, The Netherlands, September 2009): Revised Selected Papers. — Berlin–Heidelberg: Springer-Verlag, 2010. P. 276–285.

21. *Zatsman I., Kozhunova O.* Evaluation system for the Russian Academy of Sciences: Objectives-resources-results approach and R&D indicators // IEEE Xplore Digital Library. Atlanta Conference on Science and Innovation Policy 2009: E-print Proceedings of the International Conference ATLC-2009. — Georgia Institute of Technology. Downloaded on January 4, 2010 at 14:35. 6 p. URL: http://smartech.gatech.edu/bitstream/1853/32300/1/104-674-1-PB.pdf.

6 ТЕЗИСЫ ДОКЛАДОВ

6.1 Тезисы докладов, опубликованные в трудах конференций и других научных мероприятий, проведенных в России

1. *Агаларов Я. М., Шоргин С. Я.* Об одном численном методе расчета телекоммуникационной сети с учетом повторных передач // Обозрение прикладной и промышленной математики, 2010. Т. 17. Вып. 2. XVII Всеросс. школа-коллоквиум по стохастическим методам, XI Всеросс. симпозиум по прикладной и промышленной математике (весенняя сессия) и II Региональный макросимпозиум «Насущные задачи прикладной математики в Ставрополье» (Кисловодск, 1–8 мая 2010 г.): Тезисы докладов. Ч. I. С. 244–245.

2. *Арутюнян А. Р., Синицын И. Н., Ушмаев О. С.* Статистический анализ естественных деформаций изображений отпечатков пальцев // Оптико-электронные приборы и устройства в системах распознавания образов, обработки изображений и символьной информации (Распознавание-2010): Матлы IX Междунар. конф. (Курск, 18–20 мая 2010 г.). — Курск: КурскГТУ, 2010. С. 181–183.

3. *Архипов О. П., Архипов П. О.* Конверсия биометрических данных в машиночитаемые зоны документов // Информационные технологии в науке, образовании и производстве (ИТНОП): Мат-лы Междунар. науч.-техн. конф. (Орел, 22–23 апреля 2010 г.). — Орел: ОрелГТУ, 2010. Т. 1. С. 11–17.

4. *Архипов О. П., Зыкова З. П.* Характеристики цветового пространства восприятия // Информационные технологии в науке, образовании и производстве (ИТНОП): Мат-лы Междунар. науч.-техн. конф. (Орел, 22–23 апреля 2010 г.). — Орел: ОрелГТУ, 2010. Т. 3. С. 19–24.
5. *Архипова М. Ю.* Исследование взаимосвязи между научными исследованиями и инновационной активностью в России // Применение многомерного статистического анализа в экономике и оценке качества: Труды IX Междунар. конф. (Москва, 24–26 августа 2010 г.). — М.: ГУ-ВШЭ, 2010. С. 44–46.
6. *Архипова М. Ю., Архипов К. В.* Динамическая модель оптимизации логистических процессов // Системное моделирование социально-экономических процессов: Тезисы докладов XXXIII Междунар. школы-семинара им. акад. С. С. Шаталина (Звенигород, Моск. обл., 1–5 октября 2010 г.). — Воронеж: ВГУ, 2010. ISSN 978-5-9273-1732-5. С. 37–38.
7. *Архипова М. Ю., Архипов К. В.* Этапы развития статистического обследования инновационных процессов // Инновационное развитие российской экономики: Труды III Междунар. науч.-практич. конф. (Москва, 9–10 декабря 2010 г.). — М.: МЭСИ, 2010. Ч. 1. С. 26–27.
8. *Архипова М. Ю., Сиротин В. П.* Состояние и тенденции инновационного развития регионов России // Трансформация социально-экономического пространства: Мат-лы Междунар. науч.-практич. конф. (Улан-Удэ, 16–19 сентября 2010 г.). — Улан-Удэ, ВСГТУ, 2010. Т. 2. С. 82–85.
9. *Архипова М. Ю., Хаванских В. А.* Исследование инновационного потенциала Российской академии наук и его влияние на развитие технологий // Инновационное развитие российской экономики: Труды III Междунар. науч.-практич. конф. (Москва, 9–10 декабря 2010 г.). — М.: МЭСИ, 2010. Ч. 1. С. 27–30.
10. *Архипова М. Ю., Шульга С. Ю.* Оценка программной деятельности в сфере науки: опыт США // Инновационное развитие российской экономики: Труды III Междунар. науч.-практич. конф. (Москва, 9–10 декабря 2010 г.). — М.: МЭСИ, 2010. Ч. 1. С. 30–32.
11. *Богданова Д. А.* Медиаграмотность в школах Великобритании // Новые информационные технологии в образовании — Байкал (НИТО-Байкал): Мат-лы Междунар. науч.-практич. конф. (Улан-Удэ, 12–14 июля 2010 г.). — Улан-Удэ: БФФК; РГППУ; ОмГУ, 2010. С. 230–232.
12. *Богданова Д. А.* Об использовании новых информационно-коммуникационных технологий в выставочном сервисе // Наука XXI века — индустрии сервиса: Мат-лы IX Всеросс. науч.-практич. конф. (Ростов-на-Дону, 17–18 марта 2010 г.). — Ростов-на-Дону: Ростовский технологический институт

сервиса и туризма (филиал ГОУ ВПО «Южно-Российский государственный университет экономики и сервиса»), 2010. С. 8–12.

13. *Богданова Д. А., Федосеев А. А.* Информационные образовательные ресурсы: пора заняться качеством // Развивающие информационные технологии в образовании: использование учебных материалов нового поколения в образовательном процессе (ИТО-Томск-2010): Мат-лы Всеросс. науч.-практич. конф. (Томск, 23–25 марта 2010 г.). — Томск: Открытый молодежный университет, 2010. С. 63–66.

14. *Богданова Д. А., Федосеев А. А.* Schome Park — школа будущего? // Вестник Марийского государственного университета, 2010. № 5. Применение информационно-коммуникационных технологий в образовании (ИТО-Марий Эл-2010): Мат-лы VII Всеросс. науч.-практич. конф. (Йошкар-Ола, 19–21 мая 2010 г.). С. 49–51.

15. *Борохов С. В., Будзко В. И.* Информационная безопасность при консолидированной обработке на мэйнфрейме // Информационные технологии и математическое моделирование систем 2009–2010: Труды Междунар. науч.-техн. конф. — М.: Радиотехника, 2010. С. 182–183.

16. *Будзко В. И.* Создание катастрофоустойчивой территориально-распределенной системы централизованной обработки информации Банка России // Информационные технологии и математическое моделирование систем 2009–2010: Труды Междунар. науч.-техн. конф. — М.: Радиотехника, 2010. С. 184–185.

17. *Бунтман Н. В., Зацман И. М.* О проекте создания компьютерного ресурса трудностей перевода: заметки на полях // Маргиналии-2010: границы культуры и текста: Тезисы II Междунар. конф. (Каргополь, 24–27 сентября 2010 г.). — М.: МГУ, 2010. С. 41–43. — URL: http://www.srcc.msu.su/uni-persona/site/conf/marginalii-2010/thesis.htm.

18. *Вихрев В. В.* Единая коллекция Цифровых образовательных ресурсов и информационное пространство общего среднего образования // Новые информационные технологии в образовании — Байкал (НИТО-Байкал): Мат-лы Междунар. науч.-практич. конф. (Улан-Удэ, 12–14 июля 2010 г.). — Улан-Удэ: БФФК; РГППУ; ОмГУ, 2010. С. 163–165.

19. *Вихрев В. В.* Информатизация школы: потребности, парадигмы и Единая коллекция ЦОР // Вестник Марийского государственного ун-та, 2010. № 5. Применение информационно-коммуникационных технологий в образовании (ИТО-Марий Эл-2010): Мат-лы VII Всеросс. науч.-практич. конф. (Йошкар-Ола, 19–21 мая 2010 г.). С. 61–64.

20. *Вихрев В. В.* О методической поддержке учителя, ключевой фигуры информатизации школы // Информационные технологии для Новой школы:

Мат-лы конф. (Санкт-Петербург, 23–24 марта 2010 г.). — СПб.: Региональный центр оценки качества образования и информационных технологий, 2010. С. 92–95.

21. *Вихрев В. В.* Об одной вполне назревшей задаче информатизации школы // Применение новых технологий в образовании (ИТО-Троицк-2010): Мат-лы XXI Междунар. конф. (Троицк, 28–29 июня 2010 г.). — Троицк: МОО Фонд новых технологий в образовании «Байтик», 2010. С. 334–335.

22. *Вихрев В. В.* ЦОР единой коллекции как новая парадигма: сущность парадигматического сдвига // Развивающие информационные технологии в образовании: использование учебных материалов нового поколения в образовательном процессе (ИТО-Томск-2010): Мат-лы Всеросс. науч.-практич. конф. (Томск, 23–25 марта 2010 г.). — Томск: Открытый молодежный университет, 2010. С. 71–74.

23. *Вихрев В. В., Шпакова Т. Ю.* Единая коллекция ЦОР как ориентир для процесса информатизации школы // Применение новых технологий в образовании (ИТО-Троицк-2010): Мат-лы XXI Междунар. конф. (Троицк, 28–29 июня 2010 г.). — Троицк: МОО Фонд новых технологий в образовании «Байтик», 2010. С. 361–365.

24. *Вовченко А. Е., Вольнова А. А., Денисенко Д. В., Калиниченко Л. А., Куприянов В. В., Позаненко А. С., Скворцов Н. А., Ступников С. А.* Применение средств виртуальной обсерватории для выбора вторичных стандартов поля при фотометрии оптического послесвечения гамма-всплесков // От эпохи Галилея до наших дней: Всеросс. астрономическая конф. ВАК-2010 (Нижний Архыз, Карачаево-Черкесия, 13–18 сентября 2010 г.). — URL: http://agora.guru.ru/VAK-2010/files/257_Volnova_Abstract_VAK2010.rtf.

25. *Гершкович М. М., Бирюкова Т. К., Синицын В. И.* Проблемы проектирования федеральных информационно-телекоммуникационных систем, организация взаимодействия хранилищ данных в территориально-распределенных системах и задачи распознавания информационных объектов // Оптико-электронные приборы и устройства в системах распознавания образов, обработки изображений и символьной информации (Распознавание-2010): Мат-лы IX Междунар. науч.-техн. конф. (Курск, 18–20 мая 2010 г.). — Курск: КурскГТУ, 2010. С. 127–129.

26. *Гордон Л. Г.* Международный стандарт, ориентированный на пользователя // Культурное наследие и информационные технологии: XIV ежегодная конф. АДИТ-2010 (Краснодар, 7–11 июня 2010 г.). — Краснодар: Краснодарский краевой художественный музей им. Ф. А. Коваленко, 2010. С. 5.

27. *Гордон Л. Г.* Стандарты веб-дизайна и пользовательского интерфейса // Информационные технологии в образовании: Сборник трудов XX Между-

нар. конф.-выставки (Москва, 1–3 ноября 2010 г.). Ч. III. — М.: МИФИ, 2010. С. 16.

28. *Гринченко С. Н.* Биокосмология — взгляд из России // I Междунар. семинар по биокосмологии (Великий Новгород, 22–25 июля 2010 г.): Сборник тезисов. — Великий Новгород: НовГУ им. Ярослава Мудрого, 2010. С. 15–16.
29. *Гринченко С. Н.* Геохронологическая шкала и кибернетика человеческого общества // Доклады МОИП, 2010. Т. 44. Секция математики в геологии. Математические методы анализа цикличности в геологии (циклы природы и общества): Мат-лы XV Междунар. московской науч. конф. памяти Виктора Ефимовича Хаина (Москва, 19 марта 2010 г.). — М.: МОИП, 2010. С. 44–47.
30. *Гринченко С. Н.* Идентичность России в условиях глобализации: кибернетическая точка зрения // Россия и современный мир: проблемы политического развития: Тезисы VI Междунар. межвузовской науч. конф. (Москва, 15–17 апреля 2010 г.). — М.: Ин-т бизнеса и политики, 2010. С. 208–209.
31. *Гринченко С. Н.* Информатико-кибернетический аспект Мироздания // I Междунар. семинар по биокосмологии (Великий Новгород, 22–25 июля 2010 г.): Сборник тезисов. — Великий Новгород: НовГУ им. Ярослава Мудрого, 2010. С. 54–57.
32. *Гринченко С. Н.* О поисково-оптимизационном подходе к иерархической системе неживой природы // Философия физики: Актуальные проблемы: Мат-лы науч. конф. 17–18 июня 2010 г. — М.: ЛЕНАНД, 2010. С. 286–289.
33. *Гринченко С. Н.* Общество знаний и его языки: с информатико-кибернетических позиций // Этнос, нация, общество: российская реальность и перспективы: Всеросс. социологическая конф. (Москва, 1–3 ноября 2010 г.). — М.: РОС; ИС РАН, 2010. ISBN 978-5-904804-02-2. CD.
34. *Гринченко С. Н.* Понадобятся ли России в будущем ученые сверхвысокой квалификации? // Модернизация России: наука, образование, высокие технологии: Тезисы выступлений участников II Всеросс. конф. по науковедению (Москва, 15–17 ноября 2010 г.). — М.: МГПУ, 2010. С. 140–143.
35. *Гринченко С. Н.* Экология в современном мире: информатико-кибернетическое представление // Вестник Международной академии наук (Русская секция), 2010. Спец. вып. № 2: Экология, технология, культура в современном мире: проблемы vs. решения: Мат-лы Междунар. конф. (Москва, 26–27 октября 2010 г.). С. 88–90.
36. *Гуревич И. М.* Информационная оценка массы начальной неоднородности Вселенной // От эпохи Галилея до наших дней: Всероссийская астрономи-

ческая конф. ВАК-2010 (Нижний Архыз, 13–18 сентября 2010 г.). — URL: http://agora.guru.ru/VAK-2010/files/353_Gurevich_.doc.

37. *Гуревич И. М.* Информационные ограничения на образование и слияние черных дыр // От эпохи Галилея до наших дней: Всеросс. астрономическая конф. ВАК-2010 (Нижний Архыз, 13–18 сентября 2010 г.). — URL: http://agora.guru.ru/VAK-2010/files/352_Gurevich_1_Information_restrictions_on_formation_and_merge_of_black_holes.doc.

38. *Гуревич И. М.* Основные информационные характеристики нашей Вселенной // От эпохи Галилея до наших дней: Всеросс. астрономическая конф. ВАК-2010 (Нижний Архыз, 13–18 сентября 2010 г.). — URL: http://agora.guru.ru/VAK-2010/files/354_Gurevich_3_Basic_information_characteristics_of_our_Universe.doc.

39. *Гуревич И. М.* Фундаментальные ограничения на информационные характеристики систем // Мехатроника, автоматизация, управление (МАУ-2010): 7-я науч.-техн. конф. Мат-лы 3-й Российской мультиконференции по проблемам управления (Санкт-Петербург, 12–14 октября 2010 г.). — СПб.: Электроприбор, 2010. С. 144–147.

40. *Дулин С. К., Дулина Н. Г., Уманский В. И.* Моделирование неопределенности в ГИС // Труды научной сессии НИЯУ МИФИ — 2010. Т. V. — М.: НИЯУ МИФИ, 2010. С. 41–44.

41. *Ермаков П. В., Кожунова О. С.* Применение средств функционального программирования в задачах представления языковых и межъязыковых структур // 1-я школа молодых ученых ИПИ РАН: Сборник докладов. — М.: ИПИ РАН, 2010. С. 15–22.

42. *Иванов А. В.* Идентификация параметров в моделях процессов функционирования информационного портала // Обозрение прикладной и промышленной математики, 2010. Т. 17. Вып. 3. XVII Всеросс. школа-коллоквиум по стохастическим методам; XI Всеросс. симпозиум по прикладной и промышленной математике (весенняя сессия) и II Региональный макросимпозиум «Насущные задачи прикладной математики в Ставрополье» (Кисловодск, 1–8 мая 2010 г.): Тезисы докладов. Ч. II. С. 408–410.

43. *Кириков И. А., Колесников А. В., Листопад С. В.* Моделирование процессов самоорганизации в коллективах интеллектуальных агентов // Искусственный интеллект: философия, методология, инновации: Мат-лы IV Всеросс. конф. студентов, аспирантов и молодых ученых (Москва, 10–12 ноября 2010 г.). Ч. 1. — М.: Радио и связь, 2010. С. 9–11.

44. *Козеренко Е. Б.* Моделирование «значения как употребления» в лингвистических процессорах // Теоретическая и прикладная лингвистика: пути развития: Мат-лы науч. конф., посвященной 100-летию со дня рождения

В. А. Звегинцева (Москва, 29–30 октября 2010 г.). — М.: МГУ, 2010. С. 87–88.

45. *Козлов М. В., Коновалов М. Г., Малашенко Ю. Е., Назарова И. А.* Имитационная модель управления гетерогенным вычислительным комплексом // Междунар. конф. по прикладной математике и информатике, посвященная 100-летию со дня рождения академика А. А. Дородницына (Москва, 7–10 декабря 2010 г.): Тезисы докладов. — М.: ВЦ РАН, 2010. С. 237–239.
46. *Коновалов М. Г., Малашенко Ю. Е., Назарова И. А.* Оперативное управление потоком заданий в системе распределенных вычислительных ресурсов // VI Московская международная конф. по исследованию операций: ORM-2010 (19–23 октября 2010 г.): Труды. — М.: МАКС Пресс, 2010. С. 301–302.
47. *Кривенко М. П.* Адаптивная комбинированная оценка плотности многомерного распределения // Обозрение прикладной и промышленной математики, 2010. Т. 17. Вып. 2. XVII Всеросс. школа-коллоквиум по стохастическим методам; XI Всеросс. симпозиум по прикладной и промышленной математике (весенняя сессия) и II Региональный макросимпозиум «Насущные задачи прикладной математики в Ставрополье» (Кисловодск, 1–8 мая 2010 г.): Тезисы докладов. Ч. I. С. 278–279.
48. *Кружков М. Г., Архипова М. Ю.* Подходы к разработке методики индикаторного оценивания процессов трансформации знаний в новые технологии // Инновационное развитие российской экономики: Труды III Междунар. науч.-практич. конф. (Москва, 9–10 декабря 2010 г.). — М.: МЭСИ, 2010. Ч. 1. С. 55–58.
49. *Куров Б. Н.* Применение теории S-моделирования в практике преподавания экономико-математических дисциплин // Информационные технологии в образовании: Сборник трудов XX Междунар. конф.-выставки ИТО-2010 (Москва, 1–3 ноября 2010 г.). Ч. IV. — М.: МИФИ, 2010. С. 25–27.
50. *Логинова Т. З.* Особенности экранной типографики в образовательных ресурсах // Информационные технологии в образовании (ИТО-Москва-2010): Мат-лы Междунар. конф. (Москва, МИРЭА, 14–16 ноября 2010 г.). Ч. 2 — М.: МИРЭА, 2010. С. 266–270.
51. *Логинова Т. З.* Федеральные порталы интернет-ресурсов для среднего образования // Новые информационные технологии в образовании — Байкал (НИТО-Байкал): Мат-лы междунар. науч.-практич. конф. (Улан-Удэ, 12–14 июля 2010 г.). — Улан-Удэ: БФФК; РГППУ; ОмГУ, 2010. С. 251–253.
52. *Маньяков Ю. А.* Технология регистрации поведения объектов в трехмерном пространстве // Информационные технологии в науке, образовании и производстве (ИТНОП): Мат-лы Междунар. науч.-техн. конф. (Орел, 22–23 апреля 2010 г.). — Орел: ОрелГТУ, 2010. Т. 3. С. 182–186.

53. *Обухова О. Л., Бирюкова Т. К., Гершкович М. М., Соловьев И. В., Чочиа А. П.* Конструктор запросов интеллектуального поиска // Электронные библиотеки: перспективные методы и технологии, электронные коллекции (RCDL'2010): Труды XII Всеросс. науч. конф. (Казань, 13–17 октября 2010 г.). — Казань: КГУ, 2010. С. 557–559.

54. *Печинкин А. В., Шоргин С. Я.* Исследование систем обслуживания с ограничением на суммарный объем находящихся в них заявок // Обозрение прикладной и промышленной математики, 2010. Т. 17. Вып. 2. XVII Всеросс. школа-коллоквиум по стохастическим методам; XI Всеросс. симпозиум по прикладной и промышленной математике (весенняя сессия) и II Региональный макросимпозиум «Насущные задачи прикладной математики в Ставрополье» (Кисловодск, 1–8 мая 2010 г.): Тезисы докладов. Ч. I. С. 294–295.

55. *Сейфуль-Мулюков Р. Б.* Законы информатики в изучении генезиса сложных природных систем на примере нефти // Дегазация Земли: Мат-лы Всеросс. конф. — М.: ГЕОС, 2010. С. 495–498.

56. *Синицын И. Н., Сергеев И. В., Агафонов Е. С.* Применение канонических представлений случайных функций в задачах расчета виброзащитных систем для компьютерного оборудования // Системы компьютерной математики и их приложения (СКМП-2010): Труды IX Междунар. науч.-техн. конф. (Смоленск, 17–19 мая 2010 г.). — Смоленск: СмолГУ, 2010. Вып. 11. С. 239–241.

57. *Синицын И. Н., Синицын В. И., Корепанов Э. Р., Белоусов В. В.* Развитие программного обеспечения для анализа, фильтрации и распознавания на основе канонических разложений случайных функций // Оптико-электронные приборы и устройства в системах распознавания образов, обработки изображений и символьной информации (Распознавание-2010): Мат-лы IX Междунар. конф. (Курск, 18–20 мая 2010 г.). — Курск: КурскГТУ, 2010. С. 28–29.

58. *Сиротин В. П., Архипова М. Ю.* Индикаторы связи между научными исследованиями и инновациями // Инновационное развитие российской экономики: Труды III Междунар. науч.-практич. конф. (Москва, 9–10 декабря 2010 г.). — М.: МЭСИ, 2010. Ч. 1. С. 100–101.

59. *Сиротинин Д. О.* Информационная модель характеризации натурного объекта в пространстве сцены на основе двумерных изображений // Информационные технологии в науке, образовании и производстве (ИТНОП): Мат-лы Междунар. науч.-техн. конф. (Орел, 22–23 апреля 2010 г.). — Орел: ОрелГТУ, 2010. Т. 3. С. 292–298.

60. *Стефанович А. И.* Информационные технологии оптимизации функционирования web-портала // Обозрение прикладной и промышленной ма-

тематики, 2010. Т. 17. Вып. 3. XVII Всеросс. школа-коллоквиум по стохастическим методам; XI Всеросс. симпозиум по прикладной и промышленной математике (весенняя сессия) и II Региональный макросимпозиум «Насущные задачи прикладной математики в Ставрополье» (Кисловодск, 1–8 мая 2010 г.): Тезисы докладов. Ч. II. С. 463–464.

61. *Стычук А. А., Архипов П. О., Засимов А. С., Голенков А. В.* Разработка дополнительных сервисов интернет-системы обеспечения конфиденциальности документов // Информационные технологии в науке, образовании и производстве (ИТНОП): Мат-лы Междунар. науч.-техн. конф. (Орел, 22–23 апреля 2010 г.). — Орел: ОрелГТУ, 2010. Т. 3. С. 304–311.

62. *Фомичёв В. М.* Свойства внешних управляющих последовательностей // Прикладная дискретная математика, 2010. Приложение № 3(9). Компьютерная безопасность и криптография (SIBECRYPT'10): Тезисы докладов IX Сибирской науч. школы-семинара с международным участием (Тюмень, 6–11 сентября 2010 г.). — Томск: НТЛ, 2010. С. 15–19.

63. *Христочевский С. А.* Дистанционное повышение квалификации учителей // Информационные технологии в образовании (ИТО-Москва-2010): Мат-лы Междунар. конф. (Москва, МИРЭА, 14–16 ноября 2010 г.). Ч. 2. — М.: МИРЭА, 2010. С. 196–197.

64. *Христочевский С. А.* К вопросу эффективности использования ИКТ в образовательном процессе // Информационные технологии в образовании: Сборник трудов XX Междунар. конф.-выставки ИТО-2010 (Москва, 1–3 ноября 2010 г.). Ч. III. — М.: МИФИ, 2010. С. 74–75.

65. *Христочевский С. А.* Образовательные продукты «1С»: от количества к новому качеству // Новые информационные технологии в образовании: повышение эффективности обучения и управления образовательными учреждениями с использований технологий «1С»: Сборник науч. трудов X Междунар. науч.-практич. конф. (Москва, 2–3 февраля 2010 г.). Ч. 2. — М., 2010. С. 308–310.

66. *Христочевский С. А.* Проблемы использования коллекций информационных образовательных ресурсов // Информационные технологии в образовании: Сборник трудов XX Междунар. конф.-выставки ИТО-2010 (Москва, 1–3 ноября 2010 г.). Ч. IV. — М.: МИФИ, 2010. С. 53–54.

67. *Христочевский С. А.* Программа по информатике, какой ей быть? // Информационные технологии в образовании (ИТО-Москва-2010): Мат-лы Междунар. конф. (Москва, МИРЭА, 14–16 ноября 2010 г.). Ч. 1. — М.: МИРЭА, 2010. С. 61–63.

68. *Чавтараев Р. Б.* Применение средств WPF для реализации инструментария подготовки электронных коллекций. // Обозрение прикладной и промышленной математики, 2010. Т. 17. Вып. 3. XVII Всеросс. школа-коллоквиум

по стохастическим методам; XI Всеросс. симпозиум по прикладной и промышленной математике (весенняя сессия) и II Региональный макросимпозиум «Насущные задачи прикладной математики в Ставрополье» (Кисловодск, 1–8 мая 2010 г.): Тезисы докладов. Ч. II. С. 474–475.

69. *Grinchenko S. N.* About restrictions on forecasting time-frame during behaviour modeling of ipseity-social-technological systems // Математическое моделирование социальной и экономической динамики (MMSED-2010): Труды III Междунар. конф. (Москва, 23–25 июня 2010 г.). — М.: ЛЕНАНД, 2010. С. 81–83.

6.2 Тезисы докладов, опубликованные в трудах конференций и других научных мероприятий, проведенных за рубежом

1. *Богданова Д. А.* Национальная образовательная сеть Великобритании и Российская единая коллекция цифровых образовательных ресурсов // Информатизация образования — 2010: педагогические аспекты создания информационно-образовательной среды: Мат-лы Междунар. науч. конф. (Минск, 27–30 октября 2010 г.). — Минск: БГУ, 2010. С. 64–67.
2. *Бондаренко Т. В.* Язык описания иерархического автомата для генерации программного обеспечения телекоммуникационного протокола // Информационные технологии в науке, социологии, экономике и бизнесе (IT + S&E‘10): Мат-лы XXXVII Междунар. конф. (Украина, Крым, Ялта–Гурзуф, 1–10 октября 2010 г.). Приложение к журналу «Открытое образование» № 6. — М.: МЭСИ, 2010. С. 21–22.
3. *Волович К. И.* Синтез программного кода клонирования иерархических автоматов для поддержки множества сессий // Информационные технологии в науке, социологии, экономике и бизнесе (IT + S&E’10): Мат-лы XXXVII Междунар. конф. (Украина, Крым, Ялта–Гурзуф, 1–10 октября 2010 г.). Приложение к журналу «Открытое образование» № 6. — М.: МЭСИ, 2010. С. 23–24.
4. *Горшенин А. К., Королёв В. Ю., Шоргин С. Я.* CPC-методы как методы интеллектуального анализа данных при исследовании реальных хаотических процессов // Интеллектуализация обработки информации: VIII Междунар. конф. (Пафос, Кипр, 17–24 октября 2010 г.): Сборник докладов. — М.: МАКС Пресс, 2010. С. 224–227.
5. *Гуревич И. М.* Основные информационные характеристики физических, химических и биологических систем // Modern Trends in Theoretical and Applied Biophysics, Physics and Chemistry (BPPC-2010): 6th Science-Technical Conference (International) Proceedings. Vol. 1. Common Questions of Physics and Chemistry. — Sevastopol: SevNTU, 2010. P. 83–86.

6. *Зацаринный А. А. Шабанов А. П.* Исследование и разработка методического обеспечения и технологических решений по управлению производительностью контрольно-технологических трактов // Информационные технологии в науке, социологии, экономике и бизнесе (IT + S&E'10): Мат-лы XXXVII Междунар. конф. (Украина, Крым, Ялта–Гурзуф, 1–10 октября 2010 г.). Приложение к журналу «Открытое образование» № 6. — М.: МЭСИ, 2010. С. 44–45.
7. *Зейфман А. И., Панфилова Т. Л., Сатин Я. А., Шилова Г. Н.* Об устойчивости процессов рождения и гибели с катастрофами // Теория вероятностей, математическая статистика и их приложения: Мат-лы Междунар. конф. (Минск, 22–25 февраля 2010 г.). Вып. 3. — Минск: РИВШ, 2010. С. 111–113.
8. *Ионенков Ю. С.* Об одном подходе к выбору технических средств для построения телекоммуникационных сетей // Информационные технологии в науке, социологии, экономике и бизнесе (IT + S&E'10): Мат-лы XXXVII Междунар. конф. (Украина, Крым, Ялта–Гурзуф, 1–10 октября 2010 г.). Приложение к журналу «Открытое образование» № 6. — М.: МЭСИ, 2010. С. 154–156.
9. *Козеренко Е. Б.* Семантико-статистические модели разрешения неоднозначности синтаксических структур в машинном переводе // Актуальные проблемы теоретической и прикладной лингвистики: Труды Междунар. науч. конф., посвященной памяти проф. Р. Г. Пиотровского (Минск, Беларусь, 15–16 июня 2010 г.). — Минск: МГЛУ, 2010. С. 130–134.
10. *Кондрашев В. А.* Создание программного обеспечения телекоммуникационного протокола по частично формализованным спецификациям иерархического автомата // Информационные технологии в науке, социологии, экономике и бизнесе (IT + S&E'10): Мат-лы XXXVII Междунар. конф. (Украина, Крым, Ялта–Гурзуф, 1–10 октября 2010 г.). Приложение к журналу «Открытое образование» № 6. — М.: МЭСИ, 2010. С. 28–29.
11. *Кузнецов И. П. Сомин Н. В., Соловьева Н. С., Мацкевич А. Г., Николаев В. Г.* Особенности работы одного класса лингвистических процессоров при извлечении объектов и связей из документов на естественном языке // Актуальные проблемы теоретической и прикладной лингвистики: Труды Междунар. науч. конф., посвященной памяти проф. Р. Г. Пиотровского (Минск, Беларусь, 15–16 июня 2010 г.). — Минск: МГЛУ, 2010. С. 126–130.
12. *Ушмаев О. С.* Комплексирование биометрических классификаторов // Интеллектуализация обработки информации (ИОИ-2010): VIII Междунар. конф. (Пафос, Кипр, 17–24 октября 2010 г.): Сборник докладов. — М.: МАКС Пресс, 2010. С. 465–468.

13. *Чупраков К. Г.* Метод выбора технологий и системотехнических решений для ситуационного центра на основе схем симплификации в условиях неполноты информации // Информационные технологии в науке, социологии, экономике и бизнесе (IT + S&E'10): Мат-лы XXXVII Междунар. конф. (Украина, Крым, Ялта–Гурзуф, 1–10 октября 2010 г.). Приложение к журналу «Открытое образование» № 6. — М.: МЭСИ, 2010. С. 52–53.

14. *Чупраков К. Г.* Методические подходы к обоснованию решений по размещению средств отображения информации в ситуационном центре // Информационные технологии в науке, социологии, экономике и бизнесе (IT + S&E'10): Мат-лы XXXVII Междунар. конф. (Украина, Крым, Ялта–Гурзуф, 1–10 октября 2010 г.). Приложение к журналу «Открытое образование» № 6. — М.: МЭСИ, 2010. С. 53–55.

15. *Dulina N., Kozhunova O.* Information monitoring system: A problem of linguistic resources consistency and verification // Problems of Cybernetics and Informatics (PCI-2010): 3rd Conference (International) (Baku, Azerbaijan, September 6–8, 2010). Vol. 1. — Baku: Elm, 2010. P. 56–58.

16. *Gudkov V. Yu., Ushmaev O. S.* A topological approach to user-dependent key extraction from fingerprints // XX Conference (International) on Pattern Recognition (ICPR-2010) Proceedings. — Istanbul, Turkey, 2010. P. 1281–1284.

17. *Gurevich I. M.* Information methods of research of laws and properties of the nature // Problems of Cybernetics and Informatics (PCI-2010): 3rd Conference (International) (Baku, Azerbaijan, September 6–8, 2010). Vol. 2. — Baku: Elm, 2010. P. 55–58.

18. *Gurevich I. M.* Physical informatics — information methods of natural systems research // Towards a new science of information: 4th Conference (International) on the Foundations of Information Science: FIS-2010 (Beijing, China, August 21–24, 2010). — Beijing, 2010. P. 31.

19. *Gurevich I. M.* The basic information characteristics of our Universe // Problems of Cybernetics and Informatics (PCI-2010): 3rd Conference (International) (Baku, Azerbaijan, September 6–8, 2010). Vol. 2. — Baku: Elm, 2010. P. 59–62.

20. *Kolin K.* Philosophy of information and the fundamentals of informatics // Problems of Cybernetics and Informatics (PCI-2010): 3rd Conference (International) (Baku, Azerbaijan, September 6–8, 2010). Vol. 1. — Baku: Elm, 2010. P. 290–293.

21. *Korolev V., Shevtsova I.* An improvement of the Berry–Esseen inequality // 10th Vilnius Conference (International) on Probability Theory and Mathematical Statistics (Vilnius, June 28 – July 3, 2010): Abstracts of Communications. — Vilnius: TEV, 2010. P. 89.

22. *Korolev V., Shevtsova I.* On the exactness of the Berry–Esseen–Katz inequality // Book of Abstracts of Prague Stochastics-2010 (Prague, Czech Republic, August 30 – September 2, 2010) / Eds. L. Fajrova, D. Hlubinka. — Prague: Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, 2010. P. 126.
23. *Kozerenko E. B.* Fuzzy rules in grammar formalisms // Геометрия в Одессе-2010: Мат-лы Междунар. конф. (Одесса, 24–29 мая 2010 г.). — Одеса: Благодійний фонд наукових досліджень «Наука», 2010. С. 81.
24. *Sirotin V., Arkhipova M.* Fuzzy distribution of households on income // Leipziger Stochastik-Tage: IX German Open Conference on Probability and Statistics (Leipzig, March 2–5, 2010). — Leipzig: University Leipzig, 2010. P. 270–271.
25. *Ushmaev O. S., Arutyunyan A. R.* Averaging of fingerprint template with respect to Elastic deformations // Graphicon-2010: XX Conference (International) on Computer Graphics and Vision Proceedings (Saint-Petersburg, September 20–24, 2010). — СПб.: СПбГУ ИТМО, 2010. P. 324–326.
26. *Zatsman I., Durnovo A.* Incompleteness problem for indicators system of research programme // 11th Conference (International) on Science and Technology Indicators (STI-2010): Book of abstracts (September 9–11, 2010). — Leiden: Universiteit Leiden, 2010. P. 309–311.
27. *Zeifman A.* On $M(t)/M(t)/N/N$ queue // Modern Stochastics: Theory and Applications II: Abstracts of the International Conference (Kyiv, September 7–11, 2010). — Kyiv, 2010. P. 76.
28. *Zeifman A.* Some open problems for a queueing model // XIV Summer Conference (International) on Probability and Statistics (Sozopol, Bulgaria, June 19–26, 2010): Abstracts. — Sozopol, 2010. P. 28.

7 ОБЪЕКТЫ ИНТЕЛЛЕКТУАЛЬНОЙ СОБСТВЕННОСТИ

7.1 Свидетельсова об официальной регистрации программ для ЭВМ и баз данных, выданные Роспатентом

1. *Печинкин А. В., Соколов И. А., Стальченко И. В., Чаплыгин В. В.* Программа расчета характеристик системы $\mathrm{GEO}/\mathrm{G}/1/\infty$ с вероятностным приоритетом. Свидетельство о государственной регистрации программы для ЭВМ № 2010610023 от 11.01.2010.
2. *Печинкин А. В., Соколов И. А., Чаплыгин В. В.* Программа расчета характеристик системы $\mathrm{GEO}/\mathrm{GEO}/1/r$. Свидетельство о государственной регистрации программы для ЭВМ № 2010610024 от 11.01.2010.

3. *Коновалов М. Г., Шоргин С. Я., Душин Ю. А.* Программа моделирования и оптимизации взаимодействия потребителей с удаленными ресурсами через посредников. Свидетельство о государственной регистрации программы для ЭВМ № 2010610025 от 11.01.2010.
4. *Печинкин А. В., Соколов И. А., Стальченко И. В., Чаплыгин В. В., Разумчик Р. В., Зарядов И. С., Милованова Т А.* WEB-ориентированный программный комплекс удаленного расчета стационарных характеристик систем массового обслуживания. Свидетельство о государственной регистрации программы для ЭВМ № 2010610026 от 11.01.2010.
5. *Печинкин А. В., Соколов И. А., Чаплыгин В. В.* Программа расчета характеристик системы GEO/GEO/1/ с бесконечным накопителем. Свидетельство о государственной регистрации программы для ЭВМ № 2010610027 от 11.01.2010.
6. *Коновалов М. Г., Душин Ю. А.* Программа имитационного моделирования и оптимизации работы специализированного вычислительного центра. Свидетельство о государственной регистрации программы для ЭВМ № 2010610028 от 11.01.2010.
7. *Волчек В. Н., Прокофьев А. А., Зеленов Р. А.* Система управления содержимым сайта «CMS Breeze». Свидетельство о государственной регистрации программы для ЭВМ № 2010610714 от 20.01.2010.
8. *Зеленов Р. А., Степченков Ю. А., Волчек В. В., Петрухин В. С., Прокофьев А. А., Хилько Д. В.* Система капсульного программирования и отладки (СКАТ). Свидетельство о государственной регистрации программы для ЭВМ № 2010610715 от 20.01.2010.
9. *Агаларов Я. М.* Программа расчета методом компьютерного моделирования характеристик мультисервисной СМО с ограниченным накопителем, ненадежными каналами и повторным обслуживанием. Свидетельство о государственной регистрации программы для ЭВМ № 2010611151 от 09.02.2010.
10. *Агаларов Я. М.* Программа аналитического расчета характеристик мультисервисной СМО с ограниченным накопителем, ненадежными каналами и повторным обслуживанием. Свидетельство о государственной регистрации программы для ЭВМ № 2010611152 от 09.02.2010.
11. *Морозов Н. В., Степченков Ю. А., Дьяченко Ю. Г., Степченков Д. Ю.* Функциональная полузаказная библиотека самосинхронных элементов ML03. Свидетельство о государственной регистрации программы для ЭВМ № 2010611908 от 12.03.2010.
12. *Батов А. В., Горшенин А. К., Королёв В. Ю., Кузьмин В. Ю., Назаров А. Л.* Комплекс программ для декомпозиции волатильности финансовых

индексов и процессов, наблюдаемых в экспериментах с плазменной турбулентностью, на основе статистического разделения смесей нормальных законов с помощью различных модификаций EM-алгоритма. Свидетельство о государственной регистрации программы для ЭВМ № 2010611909 от 12.03.2010.

13. *Батов А. В., Горшенин А. К., Королёв В. Ю., Кузьмин В. Ю., Назаров А. Л.* Комплекс программ для декомпозиции волатильности финансовых индексов и процессов, наблюдаемых в экспериментах с плазменной турбулентностью, в режиме реального времени на основе сеточных методов статистического разделения смесей нормальных законов. Свидетельство о государственной регистрации программы для ЭВМ № 2010611910 от 12.03.2010.

14. *Батов А. В., Горшенин А. К., Королёв В. Ю., Кузьмин В. Ю., Назаров А. Л.* Комплекс программ для анализа стохастической структуры информационных потоков в телекоммуникационных и вычислительных системах на основе статистического разделения смесей гамма-распределений. Свидетельство о государственной регистрации программы для ЭВМ № 2010611911 от 12.03.2010.

15. *Косарик В. В.* База данных системной энциклопедии. Свидетельство о государственной регистрации базы данных № 2010620223 от 12.03.2010.

16. *Френкель С. Л., Курц А. В., Либуркин Д. Л., Андерс Б. Н., Фандюшина Н. А.* Транслятор табличных представлений автоматов Мили в программы на языке SMV для автоматизации верификации проектов вычислительных устройств на основе Проверки Моделей. Свидетельство о государственной регистрации программы для ЭВМ № 2010612828 от 26.04.2010.

17. *Архипов О. П., Зыкова З. П.* Программная система многокритериального выбора и тестирования множества тест-пикселей для исследования цветовосприятия (IP_9_3). Свидетельство о государственной регистрации программы для ЭВМ № 2010613793 от 09.06.2010.

18. *Косарик В. В.* Программа вычисления индикаторов возрастного распределения публикаций. Свидетельство о государственной регистрации программы для ЭВМ № 2010614126 от 25.06.2010.

19. *Шубников С. К.* Программа планирования научных исследований. Свидетельство о государственной регистрации программы для ЭВМ № 2010614127 от 25.06.2010.

20. *Лощилова Е. Ю.* Программа генерации данных для формирования диаграмм и таблиц интерактивных аналитических отчетов. Свидетельство о государственной регистрации программы для ЭВМ № 2010614128 от 25.06.2010.

21. *Шубников С. К.* Программа описания, вычисления и представления показателей объектов мониторинга. Свидетельство о государственной регистрации программы для ЭВМ № 2010614129 от 25.06.2010.
22. *Чавтараев Р. Б., Босов А. В., Стефанович А. И.* Программный комплекс для подготовки электронных коллекций «Медиа альбом». Свидетельство о государственной регистрации программы для ЭВМ № 2010614853 от 26.07.2010.

7.2 Патенты на изобретения, зарегистрированные в Государственном реестре изобретений РФ

1. *Плеханов Л. П., Степченков Ю. А., Денисов А. Н., Дьяченко Ю. Г., Филимоненко О. П.* Самосинхронный триггер для связи с удаленным приемником. Патент РФ № 2382487 от 20.02.2010.
2. *Степченков Ю. А., Дьяченко Ю. Г., Рождественский Ю. В., Петрухин В. С.* Однотактный самосинхронный RS-триггер с предустановкой. Патент РФ № 2390092 от 20.05.2010.
3. *Степченков Ю. А., Дьяченко Ю. Г., Захаров В. Н., Гринфельд Ф. И.* Двухтактный самосинхронный RS-триггер с предустановкой и входом управления. Патент РФ № 2390093 от 20.05.2010.
4. *Степченков Ю. А., Дьяченко Ю. Г., Филин А. В., Морозов Н. В.* Двухтактный самосинхронный RS-триггер с предустановкой. Патент РФ № 2390923 от 27.05.2010.
5. *Степченков Ю. А., Дьяченко Ю. Г., Филин А. В., Морозов Н. В.* Однотактный самосинхронный RS-триггер с предустановкой и входом управления. Патент РФ № 2391772 от 10.06.2010.
6. *Торчигин А. В.* Устройство для формирования на экране изображений. Патент РФ № 2392650 от 20.06.2010.
7. *Степченков Ю. А., Плеханов Л. П., Дьяченко Ю. Г.* Двоичный самосинхронный счетчик с предустановкой. Патент РФ № 2392735 от 20.06.2010.
8. *Торчигин В. П., Торчигин А. В.* Устройство для формирования и наблюдения изображений. Патент РФ № 2400787 от 27.09.2010.
9. *Торчигин В. П., Торчигин А. В*. Светодиодный цифровой проектор. Патент РФ № 2400789 от 27.09.2010.
10. *Соколов И. А., Степченков Ю. А., Дьяченко Ю. Г.* Самосинхронный триггер с однофазным информационным входом. Патент РФ № 2405246 от 27.11.2010.

ABSTRACTS

DEVELOPMENT OF COMPUTER-AIDED STATISTICAL SCIENTIFIC SUPPORT FOR HIGH PRECISION AND ACCESSIBILITY SYSTEMS

I. N. Sinitsyn[1], *E. R. Korepanov*[2], *V. V. Belousov*[3], *V. S. Shorgin*[4], *I. V. Makarenkova*[5], *T. D. Konashenkova*[6], *E. S. Agafonov*[7], *and N. N. Semendyaev*[8]

[1]IPI RAN, sinitsin@dol.ru
[2]IPI RAN, ekorepanov@ipiran.ru
[3]IPI RAN, vbelousov@ipiran.ru
[4]IPI RAN, vshorgin@ipiran.ru
[5]IPI RAN, imakarenkova@ipiran.ru
[6]IPI RAN, tkonashenkova@ipiran.ru
[7]IPI RAN, eagafonov@ipiran.ru
[8]IPI RAN, nsemendyaev@ipiran.ru

Methodological foundations of computer-aided support of statistical research based on canonical expansions of random functions are considered. Examples of software tools for high-precision astronomentric systems and bank systems with high accessibility are presented.

Keywords: circular (angular) random variables, functions, processes, and systems; computer-aided statistical scientific support systems; high precision systems; systems with high accessibility; statistical dynamics of Earth rotation

EVOLUTION OF MODERN MICROPROCESSORS ARCHITECTURE

S. Zamkovetc[1], *V. Zakharov*[2], *and V. Krasovsky*[3]

[1]IPI RAN, SZamkovetc@ipiran.ru
[2]IPI RAN, VZakharov@ipiran.ru
[3]IPI RAN, Krasovsky_v@ineum.ru

Preconditions of the first microprocessors with CISC architecture development are considered. Its shortages, partially connected with its redundancy, are discussed. The development of RISC architecture microprocessors as an alternative to CISC is considered. It is pointed that both architectures apply dynamical control of

instructions execution. Static method of control is used in VLIW architectures, which features are also described. In such case, the main part of the control process is performed by compilers.

Keywords: microprocessor; architecture; CISC; RISC; VLIW; pipeline; out-of-order

MODEL OF PARALLEL ROUND OF WORK TREES

V. Kozmidiady

IPI RAN, v.kozmidiady@gmail.com

A mechanism is considered, summarizing MapReduce that is aimed at Massive parallel processing. Consideration is based on the fact that a common task forms the tree of works, difficult works are divided into parts up to the obtaining of simple works that can be executed in parallel. The mathematical model of work tree execution is proposed. The methods of round of such trees and their influence on common time of execution are considered.

Keywords: MapReduce; massive parallel processing; round of trees; work tree

FAULT TOLERANCE METHOD OF CACHE COHERENCE MAINTENANCE SYSTEM

B. Z. Shmeilin

IPI RAN, shmeilin@mail.ru

The increase fault tolerance method of cache coherence maintenance system with snooping protocol is presented. In the suggested method, requests, states, and signals in protocol realization scheme are coded. For fault tolerance achievement in coherence maintenance logic, the code, correcting single and detecting double error, is applied to coded requests, states, and signal lines. An improvement of fault tolerance method of cache coherence maintenance systems with snooping protocol is considered. Codes for requests, states, and signals in protocol realization scheme are developed. Codes for correction single and detecting double errors in requests, states, and signals lines are presented.

Keywords: cache coherence; snooping protocols; fault tolerance; error-correcting code

SELF-TIMED ANALYSIS OF SOME TYPES OF DIGITAL DEVICE

Y. Stepchenkov[1], *Yu. Diachenko*[2], *Yu. Rogdestvenski*[3], *and N. Morozov*[4]

[1]IPI RAN, YStepchenkov@ipiran.ru
[2]IPI RAN, YDiachenko@ipiran.ru
[3]IPI RAN, YRogdest@ipiran.ru
[4]IPI RAN, NMorozov@ipiran.ru

An approach to verification of digital circuits for self-timed by means of software tools that implement the event-based method is presented. It is shown that relatively simple means provide a total-lot test fullness of the self-timed analysis for shift registers as well as for memory registers. A technique for debugging an arbitrary circuit during its self-timed analysis is suggested. The necessity of a hierarchical approach to self-timed analysis of a complicated circuit is grounded.

Keywords: self-timed circuits; self-timed analysis; test completeness of the analysis; closing; hierarchical analysis

ON SELF-TIMED PROPERTY OF DIGITAL ELECTRONIC CIRCUITS

L. Plekhanov

IPI RAN, LPlekhanov@ipiran.ru

The self-timed concept (both independence from delays and diagnostic properties) from the practical point of view and its connection with classical definition of independence from delays are discussed. It is shown that only independence from the delays is not enough for self-timed property. Practical outcomes of the theoretical statements in the area of self-timed circuits development are presented. A link with recent state standard on reliability is shown.

Keywords: self-timed circuits; independent from delays circuits; self-timed analysis

PARTICULARITIES OF TAXONOMIC SELF-TIMED CIRCUITS ANALYSIS

Y. Rogdestvensky[1], *N. Morozov*[2], *and A. Rogdestvenskene*[3]

[1]IPI RAN, YRogdest@ipiran.ru
[2]IPI RAN, NMorozov@ipiran.ru
[3]IPI RAN, ARogdest@ipiran.ru

The article deals with a method of asynchronous circuits analysis examining their functionality independence on gate's delays. Suggested method theoretically is

based on transition diagrams (in global models) with their following equivalent transforming into the event models. Developed algorithms of analysis have a strict fundamentality of global models method but do not require a complete inspection of accessible states of the circuit. As a result, the complexity of the problem has changed from exponential to the polynomial one. Taxonomic analysis clarifies the properties of the investigated schemes, presents a detailed diagnostics and determines possible reasons of violations.

Keywords: self-timed circuits; event analysis; computer-aided design

EXECUTING SYSTEM FOR CODE SYNTHESIZED ON THE BASE OF SPECIFICATIONS ON THE LANGUAGE CELL

***O. Bondarenko*[1], *K. Volovich*[2], *and V. Kondrashev*[3]**

[1]IPI RAN, olga@ipi.ac.ru
[2]IPI RAN, kv@ipi.ac.ru
[3]IPI RAN, vd@ipi.ac.ru

Issues of the program code generation that provides the functioning of the hierarchical state machines developed on the language Cell are considered. The concept of 'runtime system' as a program code included by compiler of the language Cell into procedural code in order to provide the functioning of the main algorithm of the hierarchical state machines is described. The serialization of the procedural code which is executed by compiler Cell during its synthesis suggests a possibility of the effective functioning of the state machine without an operating system. It assigns to executing system a task to manage all the resources used by the hierarchical state machine.

Keywords: language Cell; cell; executing system; resource management; serialization; hierarchical state machine; software synthesis; telecommunication protocol

ALGORITHMS FOR THE COMPILER OF THE LANGUAGE CELL

***O. Bondarenko*[1], *K. Volovich*[2], *and V. Kondrashev*[3]**

[1]IPI RAN, olga@ipi.ac.ru
[2]IPI RAN, kv@ipi.ac.ru
[3]IPI RAN, vd@ipi.ac.ru

The main features of the compiler of the language Cell, developed for the programming of hierarchical state machines used in the specification including the behavior

of telecommunication protocols are considered. The algorithms of its syntactic and semantic analyzers are presented. Also, the aspects of the synthesis of executable code of the hierarchical state machines in a procedural language C serialized for sequential execution of commands are considered.

Keywords: language Cell; hierarchical state machine; executing serialization; context of hierarchy; software synthesis; telecommunication protocol

SPECIFIC FEATURES OF THE TIME-MEASURING DEVICES IMPLEMENTATION IN VIRTUAL MACHINES

***V. Yu. Egorov*[1] *and M. A. Shpadyrev*[2]**

[1]Department of Computer Science, Penza State University, vec@mail.ru
[2]Penza State University, lordm@nm.ru

Time-measuring devices included in the personal computer as well as the problems of developing similar devices in virtual machines and methods of their solution are described.

Keywords: time-measuring devices; virtual machine; hypervisor; guest operating system

DIRECT MEMORY ACCESS REQUEST SERVICING IN HARD DISK CONTROLLERS OF VIRTUAL MACHINES

M. A. Shpadyrev

Penza State University, lordm@nm.ru

The peculiarities of a direct memory access request servicing in physical and virtual hard disk controllers are considered. The existing methods of such requests servicing in virtual machines are described. A new method based on a multithread data processing is introduced.

Keywords: hard disk controller; direct memory access; virtual machine; virtual device; hypervisor

ОБ АВТОРАХ

Агафонов Егор Сергеевич (р. 1981) — младший научный сотрудник, ИПИ РАН

Белоусов Василий Владимирович (р. 1977) — кандидат технических наук, заведующий сектором ИПИ РАН

Бондаренко Ольга Алексеевна (р. 1970) — младший научный сотрудник ИПИ РАН

Волович Константин Иосифович (р. 1970) — старший научный сотрудник ИПИ РАН

Дьяченко Юрий Георгиевич (р. 1958) — кандидат технических наук, старший научный сотрудник ИПИ РАН

Егоров Валерий Юрьевич (р. 1974) — кандидат технических наук, доцент кафедры «Вычислительная техника» Пензенского государственного университета

Замковец Сергей Всеволодович (р. 1944) — кандидат технических наук, заведующий лабораторией ИПИ РАН

Захаров Виктор Николаевич (р. 1948) — доктор технических наук, доцент, ученый секретарь ИПИ РАН

Козмидиади Владимир Александрович (р. 1936) — доктор технических наук, главный научный сотрудник ИПИ РАН

Конашенкова Татьяна Дмитриевна (р. 1964) — ведущий программист, ИПИ РАН

Кондрашев Вадим Адольфович (р. 1963) — старший научный сотрудник ИПИ РАН

Корепанов Эдуард Рудольфович (р. 1966) — кандидат технических наук, заведующий сектором ИПИ РАН

Красовский Виктор Евгеньевич (р. 1945) — кандидат технических наук, ученый секретарь Института электронных управляющих машин (ИНЭУМ)

Макаренкова Ирина Вячеславовна (р. 1964) — ведущий программист, ИПИ РАН

Морозов Николай Викторович (р. 1956) — старший научный сотрудник ИПИ РАН

Плеханов Леонид Петрович (р. 1943) — кандидат технических наук, старший научный сотрудник отдела Архитектур перспективных компьютерных систем ИПИ РАН

Рождественскене Аста Винценто (р. 1964) — научный сотрудник ИПИ РАН

Рождественский Юрий Владимирович (р. 1952) — кандидат технических наук, заведующий сектором ИПИ РАН
Семендяев Николай Николаевич (р. 1983) — программист, ИПИ РАН
Синицын Игорь Николаевич (р. 1940) — доктор технических наук, профессор, заведующий отделом ИПИ РАН
Соколов Игорь Анатольевич (р. 1954) — академик (действительный член) Российской академии наук, доктор технических наук, директор ИПИ РАН
Степченков Юрий Афанасьевич (р. 1951) — кандидат технических наук, заведующий отделом ИПИ РАН
Шмейлин Борис Захарьевич (р. 1939) — кандидат технических наук, доцент, старший научный сотрудник ИПИ РАН
Шоргин Всеволод Сергеевич (р. 1978) — кандидат технических наук, старший научный сотрудник, ИПИ РАН
Шпадырев Михаил Алексеевич (р. 1986) — аспирант Пензенского государственного университета

ABOUT AUTHORS

Agafonov Egor S. (b. 1981) — junior scientific researsher, Institute of Informatics Problems, Russian Academy of Sciences

Belousov Vasiliy V. (b. 1977) — Candidate of Science (PhD) in technology; Head of Laboratory, Institute of Informatics Problems, Russian Academy of Sciences

Bondarenko Olga A. (b. 1970) — junior scientist, Institute of Informatics Problems, Russian Academy of Sciences

Diachenko Yuri G. (b. 1958) — Candidate of Science (PhD) in technology, senior scientist, Institute of Informatics Problems, Russian Academy of Sciences

Egorov Valery Yu. (b. 1974) — Candidate of Science (PhD) in technology, associate professor, Department of Computer Science, Penza State University

Konashenkova Tatyana D. (b. 1964) — managing programmer, Institute of Informatics Problems, Russian Academy of Sciences

Kondrashev Vadim A. (b. 1963) — senior scientist, Institute of Informatics Problems, Russian Academy of Sciences

Korepanov Edward R. (b. 1966) — Candidate of Science (PhD) in technology; Head of laboratory, Institute of Informatics Problems, Russian Academy of Sciences

Kozmidiady Vladimir A. (b. 1936) — Doctor of Science in technology; proncipal scientist, Institute of Informatics Problems, Russian Academy of Sciences

Krasovsky Victor E. (b. 1945) — Candidate of Science (PhD) in technology, Scientific Secretary, Control Computer Institute (INEUM)

Makarenkova Irina V. (b. 1964) — managing programmer, Institute of Informatics Problems, Russian Academy of Sciences

Morozov Nikolai V. (b. 1956) — senior scientist, Institute of Informatics Problems, Russian Academy of Sciences

Plekhanov Leonid P. (b. 1943) — Candidate of Science (PhD) in technology, senior scientist of the Department of Architectures of Perspective Computer Systems, Institute of Informatics Problems, Russian Academy of Sciences

Semendyaev Nikolay N. (b. 1983) — programmer, Institute of Informatics Problems, Russian Academy of Sciences

Shmeilin Boris Z. (b. 1939) — Candidate of Science (PhD) in technology, associate professor, senior scientist, Institute of Informatics Problems, Russian Academy of Sciences

Shorgin Vsevolod S. (b. 1978) — Candidate of Science (PhD) in technology, senior scientist, Institute of Informatics Problems, Russian Academy of Sciences

Shpadyrev Michael A. (b. 1986) — PhD student, Penza State University

Sinitsyn Igor N. (b. 1940) — Doctor of Science in technology, professor, Institute of Informatics Problems, Russian Academy of Sciences
Sokolov Igor A. (b. 1954) — Academician of the Russian Academy of Sciences, Doctor of Technical Sciences; director, Institute of Informatics Problems, Russian Academy of Sciences
Stepchenkov Yuri A. (b. 1951) — Candidate of Science (PhD) in technology, Head of Department, Institute of Informatics Problems, Russian Academy of Sciences
Rogdestvenskene Asta V. (b. 1964) — scientist, Institute of Informatics Problems, Russian Academy of Sciences
Rogdestvenski Yuri V. (b. 1952) — Candidate of Science (PhD) in technology, Head of Laboratory, Institute of Informatics Problems, Russian Academy of Sciences
Volovich Konstantin I. (b. 1970) — senior scientist, Institute of Informatics Problems, Russian Academy of Sciences
Zakharov Victor N. (b. 1948) — Doctor of Science (PhD) in technology, associate professor; Scientific Secretary, Institute of Informatics Problems, Russian Academy of Sciences
Zamkovetc Sergey V. (b. 1944) — Candidate of Science (PhD) in tecnology, Head of Laboratory, Institute of Informatics Problems, Russian Academy of Sciences

Правила подготовки рукописей статей для публикации в журнале «Системы и средства информатики»

Журнал «Системы и средства информатики» публикует теоретические, обзорные и дискуссионные статьи, посвященные научным исследованиям и разработкам в области информационных технологий. Журнал издается на русском языке. Тематика журнала охватывает следующие направления:

- информационно-телекоммуникационные системы и средства их построения;
- архитектура и программное обеспечение вычислительных машин, комплексов и сетей;
- методы и средства защиты информации.

1. В журнале печатаются результаты, ранее не опубликованные и не предназначенные к одновременной публикации в других изданиях. Публикация не должна нарушать закон об авторских правах. Направляя свою рукопись в редакцию, авторы автоматически передают учредителям и редколлегии неисключительные права на издание данной статьи на русском языке и на ее распространение в России и за рубежом. При этом за авторами сохраняются все права как собственников данной рукописи. В связи с этим авторами должно быть представлено в редакцию письмо в следующей форме:
   Соглашение о передаче права на публикацию:
   «Мы, нижеподписавшиеся, авторы рукописи “. . . ”, передаем учредителям и редколлегии журнала «Системы и средства информатики» неисключительное право опубликовать данную рукопись статьи на русском языке как в печатной, так и в электронной версиях журнала. Мы подтверждаем, что данная публикация не нарушает авторских прав других лиц или организаций. Подписи авторов: (ф. и. о., дата, адрес)».
   Указанное соглашение может быть представлено как в бумажном виде, так и в виде отсканированной копии (с подписями авторов).
   Редколлегия вправе запросить у авторов экспертное заключение о возможности опубликования представленной статьи в открытой печати.
2. Статья подписывается всеми авторами. На отдельном листе представляются данные автора (или всех авторов): фамилия, полные имя и отчество, телефон, факс, e-mail, почтовый адрес. Если работа выполнена несколькими авторами, указывается фамилия одного из них, ответственного за переписку с редакцией.
3. Редакция журнала осуществляет самостоятельную экспертизу присланных статей. Возвращение рукописи на доработку не означает, что статья уже принята к печати. Доработанный вариант с ответом на замечания рецензента необходимо прислать в редакцию.
4. Решение редакционной коллегии о принятии статьи к печати или ее отклонении сообщается авторам.
   Редколлегия не обязуется направлять рецензию авторам отклоненной статьи.

5. Корректура статей высылается авторам для просмотра. Редакция просит авторов присылать свои замечания в кратчайшие сроки.
6. При подготовке рукописи в MS Word рекомендуется использовать следующие настройки.
   Параметры страницы: формат — A4; ориентация — книжная; поля (см): внутри — 2,5, снаружи — 1,5, сверху — 2, снизу — 2, от края до нижнего колонтитула — 1,3. Основной текст: стиль — «Обычный»: шрифт Times New Roman, размер 14 пунктов, абзацный отступ 0,5 см, 1,5 интервала, выравнивание по ширине. Рекомендуемый объем рукописи — не свыше 25 страниц указанного формата. Ознакомиться с шаблонами, содержащими примеры оформления, можно по адресу в Интернете: http://www.ipiran.ru/journal/template.doc.
7. К рукописи, предоставляемой в 2-х экземплярах, обязательно прилагается электронная версия статьи, как правило, в форматах MS WORD (.doc) или LaTex (.tex), а также — дополнительно — в формате .pdf на дискете, лазерном диске или по электронной почте. Сокращения слов, кроме стандартных, не применяются. Все страницы рукописи должны быть пронумерованы.
8. Статья должна содержать следующую информацию на русском и английском языках: название, Ф.И.О. авторов, места работы авторов и их электронные адреса, подробные сведения об авторах, оформленные в соответствии с форматом, определяемым файлами http://www.ipiran.ru/journal/issues/2011_05_01/authors.asp и http://www.ipiran.ru/journal/issues/2011_01_eng/authors.asp, аннотация (не более 100 слов), ключевые слова, литература. Ссылки на литературу в тексте статьи нумеруются (в квадратных скобках) и располагаются в порядке их первого упоминания. В списке литературы не должно быть позиций, на которые нет ссылки в тексте статьи. Все фамилии авторов, заглавия статей, названия книг, конференций и т. п. даются на языке оригинала, если этот язык использует кириллический или латинский алфавит.
9. Присланные в редакцию материалы авторам не возвращаются.
10. При отправке файлов по электронной почте просим придерживаться следующих правил:
    - указывать в поле subject (тема) название журнала и фамилию автора;
    - использовать attach (присоединение);
    - в случае больших объемов информации возможно использование общеизвестных архиваторов (ZIP, RAR);
    - в состав электронной версии статьи должны входить: файл, содержащий текст статьи, и файл(ы), содержащий(е) иллюстрации.
11. Журнал является некоммерческим изданием. Плата за публикацию с авторов не взимается, гонорар авторам не выплачивается.

Адрес редакции: Москва 119333, ул. Вавилова, д. 44, корп. 2, ИПИ РАН
Тел.: +7 (499) 135-86-92 Факс: +7 (495) 930-45-05 e-mail: rust@ipiran.ru