Московский Государственный Технический Университет

им. Н. Э. Баумана

На правах рукописи

Иванов Андрей Михайлович

УДК: 004.4’242

МЕТОДЫ И СРЕДСТВА ВЕРИФИКАЦИИ ПРОГРАММНОЙ МОДЕЛИ МАРШРУТИЗАТОРА ВЫСОКОСКОРОСТНОЙ СЕТИ ТОПОЛОГИИ «4D-TOP»

А в т о р е ф е р а т

квалификационной работы бакалавра по направлению 211000

Конструирование и технология электронных средств

Москва – 2012

Работа выполнена в Московском Государственном Техническом Университете
им. Н. Э. Баумана

Научный руководитель: канд. техн. наук, доцент кафедры «Проектирование и технология производства ЭА
МГТУ им.Н.Э,Баумана
Власов Андрей Игоревич

Научный консультант: ассистент кафедры «Проектирование и технология производства ЭА
МГТУ им.Н.Э,Баумана
Аверьянихин Артур Евгеньевич

Ведущее предприятие: Научно Исследовательский Центр Электронно - Вычислительной Техники
(НИЦЭВТ) (г. Москва)

Защита квалификационной работы бакалавра состоится ____ июня 2012 года на заседании Государственной аттестационной комиссии по направлению 211000: «Конструирование и технология электронных средств» в Московском Государственном Техническом Университете им. Н.Э.Баумана (ауд.278).

Ваши отзывы в двух экземплярах просьба высылать по адресу:
105005, г.Москва, 2-ая Бауманская ул., д.5, ИУ-4.

Автореферат разослан «___» _____ 2012 г.

Ученый секретарь Государственной квалификационной комиссии по направлению 211000: «Конструирование и технология электронных средств»,
доцент, кандидат технических наук Лавров А.В.

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Актуальность работы заключается в том, что верификация распределенных систем и программного обеспечения, в целом, является быстро развивающейся областью современного программирования, ролью сетевых коммуникаций постоянно возрастает. Ввиду большой сложности многих реальных распределенных систем вопрос о правильности их работы является нетривиальной задачей. Обнаружение ошибок в рассматриваемых системах математическими методами и доказательная проверка корректности их работы являются основными целями верификации распределенных систем. Естественный подход к проблеме верификации состоит в моделировании распределенных систем конечными автоматами или сетями Петри и в верификации полученных моделей. Среди наиболее важных подходов к верификации можно выделить симуляцию работы сети и анализ ее пространства состояний. При симуляции работа сети изучается пошаговым методом в ручном или полуавтоматическом режимах. Хотя многие ошибки в работе сети могут быть обнаружены на этапе симуляции, доказательная проверка корректности работы сети может быть получена для данного подхода только при полном моделировании работы сети и проведении так называемой проверки моделей. Метод проверки моделей заключается в описании требуемых свойств системы на языках логических спецификаций и доказательстве истинности или ложности данных спецификаций на построенном пространстве состояний системы. Метод проверки моделей активно развивался и показал себя как эффективное и многообещающее средство для верификации распределенных систем. Благодаря своей естественности и возможности быть интегрированным в среду разработки и анализа распределенных систем, метод проверки моделей был принят в качестве одного из стандартов для верификации спецификаций систем, описанных на каком-либо формальном языке. В совокупности с известными, достаточно эффективными алгоритмами для логик с сильной выразительной силой, методы проверки моделей дают нам эффективный метод верификации распределенных систем.

Коммуникационная сеть состоит из узлов, в каждом из которых есть сетевой адаптер, соединенный с одним или несколькими маршрутизаторами, которые в свою очередь соединяются между собой высокоскоростными каналами связи (Линками). Структура сети, определяющая, как именно связаны между собой узлы системы, задается топологией сети (обычно решетка, тор или толстое дерево) и набором структурных параметров: количество измерений, размеры сторон тора, число узлов сети, портов у маршрутизаторов.

Сеть «Ангара-С» имеет топологию 4D-тор. Поддерживается детерминированная маршрутизация, сохраняющая порядок передачи пакетов и предотвращающая появление дедлоков (взаимных блокировок), а также адаптивная маршрутизация, позволяющая одновременно использовать множество путей между узлами и обходить перегруженные и вышедшие из строя участки сети. Особое внимание было уделено поддержке коллективных операций (широковещательной рассылки и редукции), реализованных с помощью виртуальной подсети, имеющей топологию дерева, наложенного на многомерный тор. В сети на аппаратном уровне поддерживаются удаленные записи, чтения и атомарные операции двух типов (сложение и исключающее ИЛИ). В отдельном блоке реализована логика по агрегации пришедших из сети сообщений с целью повышения доли полезных данных на транзакцию при передаче через интерфейс с хостом (хостом называется связка процессор-память-мосты).

Межузловая коммуникационная сеть является одной из важнейших частей суперкомпьютера, определяющей его производительность и масштабируемость. Вычислительные комплексы последнего поколения строятся по принципу разделяемой памяти с общим адресным пространством, что приводит к повышению межузлового трафика сети. Требования, предъявляемые к ВОКС (Высокоскоростная Отказоустойчивая Коммуникационная Сеть) – минимальная задержка передачи данных по сети, максимальная пропускная способность, высокая отказоустойчивость, масштабируемость до сотен тысяч узлов.

Технологии, основанные на топологии «многомерный тор», как правило, используются при создании суперкомпьютеров высшего диапазона производительности такими компаниями, как Cray Inc., IBM и др. Суперкомпьютер Cray XT5 Jaguar, занимающий второе место в списке Top 500 наиболее производительных суперкомпьютеров в мире, использует интерконнект Cray SeaStar2+, реализующий топологию «многомерный тор». Коммуникационная сеть с данной топологией используется целым рядом суперкомпьютеров фирмы IBM. Данная топология представляет собой линии связи, расположенные по

направляющим геометрической фигуры тора, в точках пересечения которых находятся маршрутизаторы.

Коммуникационное оборудование для таких сетей, за редким исключением, является заказным и ограничено в распространении. Оно не только обеспечивает предельные значения эксплуатационных характеристик, обеспечивающих высокую эффективность решения прикладных задач, но и обладает существенно более высокой отказоустойчивостью за счёт децентрализации и применения современных адаптивных и безотказных алгоритмов передачи о. Топология «nD-тор» с размерностью n>2 обладает хорошей масштабируемостью, отказоустойчивостью, естественным образом компактно размещается по стойкам, позволяя минимизировать длину передающих линий связи. Более того, паттерн многих задач математической физики соответствует топологии трехмерного тора.

В настоящее время методы тестирования на основе моделей используют различные типы моделей и техник.

Обычные и расширенные конечные автоматы - наиболее глубоко разработаные, известны их точные ограничения и гарантии полноты выполняемых проверок. Методы, использующие расширенные автоматы, сводят их к обычным, но применяют более детальные критерии покрытия, основанные на использовании данных в расширенных автоматах. Наиболее известными инструментами создания тестов на основе таких моделей являются BZ-TT , использующий модели, описанные на языке «CTL», и Gotcha-TCBeans , использующий язык «UML Statecharts» в рамках набора инструментов AGEDIS.

Системы переходов используются при тестировании распределенных систем, поскольку моделирование таких систем с помощью конечных автоматов очень трудоемко. Большинство этих методов не определяют практически применимых критериев полноты и не дают конечных тестовых наборов для реальных систем, поэтому использующие их инструменты опираются на те или иные наблюдения для обеспечения конечности набора тестов. Наиболее известны из инструментов TorX и TGV. Помимо обычных систем переходов используются и временные (расширенные с помощью таймеров). Построение тестов на их основе возможно с использованием инструмента UPPAAL, (Средство для верификации систем реального времени, разработанное в университетах Uppsala и Aalborg).

Программные контракты представляют собой набор программных контрактов, иногда с дополнительным определением явного преобразования состояния системы. Тесты строятся на основе автоматных моделей, являющихся абстракциями от этих контрактов. Примеры инструментальной поддержки таких методов — инструменты технологии UniTESK, созданной в ИСП РАН, и инструмент SpecExplorer, разработанный в Microsoft Research.

Методы на основе проверки моделей, в которых тестовые ситуации выбираются как представители классов эквивалентности, задаваемых критерием покрытия. Соответствующая ситуации тестовая последовательность строится как контрпример при проверке модели (model checking) на выполнение свойства, являющегося отрицанием условия достижения этой ситуации.

Методы построения тестов с помощью символического выполнения (symbolic execution). Такие методы используют символическое описание проходимого во время выполнения теста пути по коду программы (или формальных проверяемых спецификаций) в виде набора предикатов. Это описание позволяет выбирать новые тестовые ситуации так, чтобы они покрывали другие пути и строить тесты с помощью техник разрешения ограничений (constraint solving). Символическое выполнение в комбинации с конкретизацией тестовых данных используется и для построения тестов, нацеленных на типичные дефекты, такие, как использование неинициализированных объектов, тупики и гонки параллельных потоков.

Можно отметить, что наиболее развиты методы тестирования на основе моделей проверки согласованности автоматов и систем переходов. Поддерживающие их инструменты все шире начинают использоваться в промышленных проектах. Методы построения тестов на основе проверки моделей и на основе символического выполнения активно развиваются и начинают воплощаться в инструменты.

Мониторинг формальных свойств ПО (для этой области используется два термина — верификация в реальном времени и пассивное тестирование) стал развиваться как отдельное направление исследований в конце 1990-х годов. В основе этого подхода лежит фиксация формальных свойств ПО, которые необходимо проверить и встраивание их проверок в систему мониторинга. В дальнейшем выполняется мониторинг ПО, в ходе которого контролируются и

выделенные свойства. В публикациях упоминаются следующие методы мониторинга формальных свойств ПО.

Модели, использующие описание свойств с помощью обычных и временных логик осуществляют контроль свойств, записанных в виде формул временных логик, с помощью записи событий, происходящих в работающем ПО и анализа возникающих трасс. Один из инструментов, использующих такие техники — Temporal Rover (временной приемник).

Модели, использующие описание свойств в виде систем переходов или автоматов, в которых трасса анализируется не на выполнение некоторых формул, а на согласованность с заданной автоматной моделью правильного поведения. Использующие программные контракты. В рамках таких подходов корректность поведения системы проверяется во время ее работы с помощью программных контрактов, внедренных в систему мониторинга.

Одной из наиболее применяемых методик является построение верификационных моделей на основе языка моделирования и верификации цифровых систем SystemC. Преимуществом данного языка программирования является тесная связь с объектно-ориентированным языком C++. Это дает возможность разрабатывать высокоуровневые тестовые модели для RTL-моделей цифровых систем, создаваемых на вентильном уровне, без больших временных затрат на переход между уровнями.

В данной к работе исследована система верификации, позволяющая параллельно с разработкой блоков устройств производить контроль его параметров и функционала на соответствие техническому заданию, в частности для рассматриваемого примера:

- осуществлять справедливый арбитраж запросов и установление соединений посредством кроссбара с различными виртуальными каналами маршрутизатора;
- принимать данные с выхода кроссбара и помещать их во внутренний буфер;
- не дожидаясь завершения приёма передавать данные, принятые во внутренний буфер, на следующий маршрутизатор;
- осуществлять контроль и удаление из внутреннего буфера пакетов, успешно принятых следующим маршрутизатором, а также повторную передачу пакетов при возникновении ошибки на приёмной стороне;
- осуществлять приём пакетов из сети, их анализ и передачу в соответствующий виртуальный канал;
- обеспечивать контроль целостности данных пакета.

Цель работы:

Разработать тестовое окружение для блока маршрутизатора «Линк», обеспечивающего взаимодействие узлов коммуникационной сети топологии «4D-тор» в рамках проекта «Ангара-С с использованием средства проектирования и верификации SystemC, и средства симуляции цифровых проектов ModelSim производства кампании Altera.

Решаемые задачи:

1. Анализ архитектуры и принципов работы RTL-моделей блоков маршрутизатора на основе их программной реализации.
2. Разработка верификационных моделей блоков и их тестовых окружений с использованием языка программирования C/C++ и средства проектирования и верификации электронных средств SystemC
3. Отладка тестового окружения с использованием средств симуляции цифровых проектов ModelSim производства кампании Altera.
4. Анализ полученных тестовых данных и проверка соответствия характеристик RTL-моделей техническим требованиям.

Методы исследования: При решении поставленных задач использованы методы синтаксического анализа, синтеза, моделирования параллельных вычислений, симуляции цифровых проектов, теоретического и объектно-ориентированного программирования.

Научная новизна работы:

- разработан оригинальный подход к решению задач спецификации и верификации систем реального времени средствами языка проектирования и верификации SystemC, позволяющими сократить затраты времени и труда на связь низкоуровневых моделей систем и высокоуровневого тестового окружения;
- разработан алгоритм верификации свойств временных моделей маршрутизатора с использованием аппарата темпоральных логик реального времени, обеспечивающий контроль взаимодействия маршрутизаторов коммуникационной сети в рамках проекта «Ангара-С» в соответствии со спецификацией;

- предложен метод редукции числа верифицируемых сетевых состояний, позволяющий учитывать как параллелизм сети, так и существенность временных ограничений при проверке взаимодействия узлов коммуникационной сети.

Достоверность полученных научных результатов и рекомендаций диссертационной работы подтверждена результатами экспериментальных исследований:

Полученные в ходе работы результаты демонстрируют эффективность предложенной концепции верификации распределенных систем реального времени, для использования в различных целях – гражданских, научных, военных. Результаты работы подтверждаются функционированием тестового окружения блока «Линк» высокоскоростной коммуникационной сети:

- Тестирование данного блока на скорость передачи данных показало пропускную способность 6Гб/с, что соответствует требованиям ТЗ.
- Проверка «Линка» на отказоустойчивость, заключающаяся в имитации физических разрывов связей между узлами сети, так же показала корректную обработку нештатных ситуаций. Потерь данных при разрыве не обнаружено.
- Тесты на проверку целостности передачи данных также показали, что «Линк» корректно выполняет обработку критериев целостности и при возникновении ошибок осуществляет повторную передачу данных из предыдущего узла.

Положения, выносимые на защиту:

- Методика верификации программной модели блока «Линк» маршрутизатора на основе синтетических моделей позволяет в режиме реального времени производить контроль соответствия характеристик, заданных в ТЗ;
- Верификационная модель на базе языка проектирования и верификации SystemC обеспечивает уменьшить время нахождения ошибок в программном обеспечении и снизить временные затраты на устранение неполадок до 30 процентов.

Практическая ценность работы. Разработанное в квалификационной работе тестовое окружение программного обеспечения найдет широкое применение в области проектирования цифровых систем. Результаты работы в виде концепции тестового окружения могут быть использованы при построении системы верификации моделей цифровых устройств. Тестовое окружение маршрутизатора «Ангара-С» обеспечивает 89 процентов покрытия исходного кода модели блока «Линк».

Реализация результатов. Разработанное в результате квалификационной работы тестовое окружение используется для верификации маршрутизатора высокоскоростной коммуникационной сети топологии «4D-тор» в «ОАО НИЦЭВТ» в рамках проекта «Ангара-С». В дальнейшем оно может найти применение в качестве прототипа для реализации тестов новых блоков маршрутизатора, а так же для верификации последующих модификаций данного маршрутизатора.

Апробация работы.

Работа была представлена на XIV Молодежной международной научно-технической конференции учащихся, студентов, спирантов и молодых ученых «Наукоемкие технологии и интеллектуальные системы-2012»

Публикации.

По материалам и основному содержанию работы опубликована 1 печатная работа в трудах конференции.

Структура и объем работы. Бакалаврская квалификационная работа состоит из введения, четырех глав, заключения. Общий объем работы 71 страниц, 42 рисунков, список использованных источников из 39 наименований.

СОДЕРЖАНИЕ И РЕЗУЛЬТАТЫ РАБОТЫ

Во введении обоснована актуальность решения задач проектирования программного обеспечения для тестирования моделей цифровых систем, сформулированы цель и задачи исследования, обоснована научная новизна и изложена структура работы, дана оценка метода проверки моделей при тестировании распределенных цифровых систем.

В **первой главе** проводится анализ маршрутизатора коммуникационной сети «Ангара-С». Данная сеть имеет топологию 4D-тор. Поддерживается детерминированная маршрутизация, сохраняющая порядок передачи пакетов и предотвращающая появление дедлоков (взаимных блокировок), а также адаптивная маршрутизация, позволяющая одновременно использовать множество путей между узлами и обходить перегруженные и вышедшие из строя участки сети. Особое внимание было уделено поддержке коллективных операций (широковещательной рассылки и редукции), реализованных с помощью виртуальной подсети, имеющей топологию дерева, наложенного на многомерный тор. В сети на аппаратном уровне поддерживаются удаленные записи, чтения и атомарные операции двух типов (сложение и исключающее ИЛИ). Схема выполнения удаленного чтения (посылка запроса и прием ответа) приведена на рисунке 2 (удаленная запись и атомарные операции выполняются аналогично). В отдельном блоке реализована логика по агрегации пришедших из сети сообщений с целью повышения доли полезных данных на транзакцию при передаче через интерфейс с хостом (хостом называется связка процессор-память-мосты).

Кратко проанализированы требования международных стандартов, касающихся верификации программного обеспечения. Основным стандартом, регулирующим планирование и проведение верификации ПО, является стандарт IEEE 1012 на процессы верификации и валидации. Этот стандарт содержит описание наборов отдельных задач верификации, соответствующих разным видам деятельности в рамках процессов, определенных в ISO 12207, рекомендуемый шаблон плана проведения верификации и валидации, с описанием содержания основных его разделов, определение 4-х уровней критичности программного обеспечения (рисунок 2). Стандарт IEEE 829 (документация тестирования программного обеспечения). Это базовый стандарт, описывающий вспомогательные артефакты для тестирования. Основными такими артефактами являются тестовый план (test plan) — основной документ, связывающий разработку тестов и тестирование с задачами проекта, тестовые варианты (test case) — сценарии проведения отдельных тестов.

Реализация проекта производится на платформе ОС Linux. Одной из базовых особенностей Linux и UNIX-подобных систем является концепция разделения прав пользователей, когда каждому пользователю системы разрешается определенный набор действий и доступ к определенным файлам и каталогам файловой системы.. Одно из основных достижений UNIX-подобных ОС состоит в том, что система обладает свойством высокой мобильности. Смысл этого качества состоит в том, что вся операционная система, включая ее ядро, сравнительно просто переносится на различные аппаратные платформы. Все части системы, не считая ядра, являются полностью машинно-независимыми. Ядро также поддерживает универсальный пул памяти для пользовательских программ и дискового кэша. При этом для кэша может использоваться вся память, и наоборот, кэш уменьшается при работе больших программ.

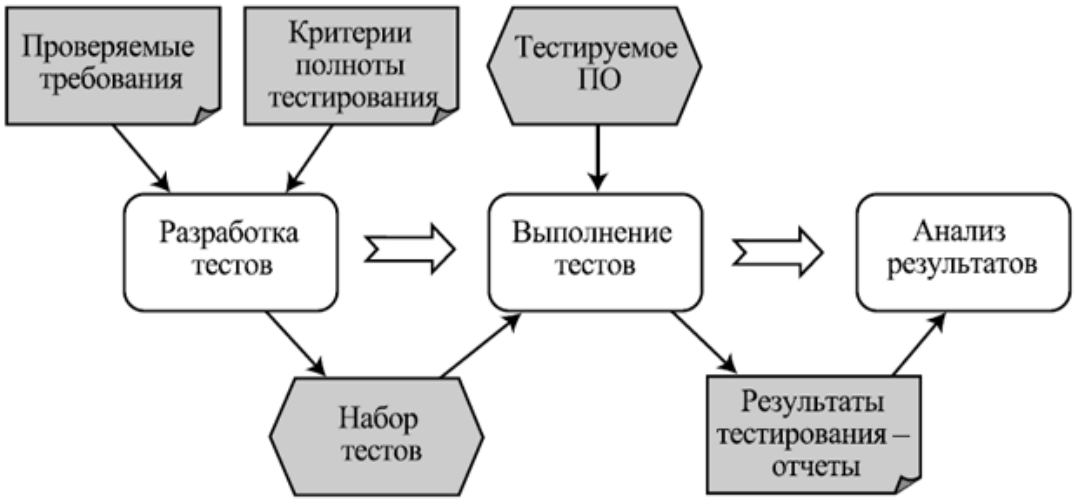

Рисунок 2 – Схема процесса тестирования

Третий этап – валидация (квалификационные испытания). На данном этапе работ проводятся квалификационные испытания системного и прикладного ПО на аппаратных средствах, идентичных штатным. Четвертый этап - Доработка ПО. Проведение по результатам испытаний (в случае необходимости) доработок ПО и внесение изменений в тексты программ и программную документацию, проведение повторных испытаний. Заключительный этап – Аттестация. На пятом этапе проводится аттестация интегрированной системы с использованием штатных средств информационного обмена, включая имитаторы систем более высокого уровня.

В качестве аналогов SystemC-моделей рассмотрены модели на языка «UML» и автомата «Mur». В настоящее время UML применяется, в основном, как язык спецификации моделей систем. Существующие UML-средства позволяют строить различные диаграммы и автоматически создавать по диаграмме классов «скелет» кода на целевом языке программирования (языки Java и C#). Некоторые их этих средств также предоставляют возможность автоматически генерировать код логики программы по диаграммам состояний. Однако можно считать, что в настоящее время указанная функциональность существует в «зачаточном состоянии», так как известные инструменты не позволяют в полной мере эффективно связывать модель поведения, которую можно описывать с помощью четырех типов диаграмм (состояний, деятельностей, кооперации или последовательностей), с генерируемым кодом. Технология автоматного программирования использует такие модели, как автомат Мили, автомат Мура и смешанный автомат, которые легко интерпретируются с помощью модели Крипке. Первым шагом в процессе верификации автоматной программы является преобразование графа переходов исходного автомата в модель Крипке, для которой удобно формулировать проверяемые свойства программ. При этом возникают следующие проблемы:

-трудности с выполнением композиции автоматов;

-неоднозначность интерпретации формулы языка Computational Tree Logic (CTL) (компьютерная древовидная логика) в исходном автомате.

При решении первой проблемы, как правило, возникает вторая. Для ее решения применялась модификация языка CTL. Методы моделирования, (в частности, «редуцированная» схема), проводят изменение семантики языка CTL для того, чтобы воспрепятствовать экспоненциальному росту числа состояний. При этом пути, построенные в качестве сценариев для CTL-формул, однозначно преобразуются из модели в автомат Мили. Это удобно, особенно если моделирование производится совместно с исполнением автомата, его визуализацией и отладкой. Кроме того, при использовании автоматного программирования число управляющих состояний относительно невелико. Это позволяет не применять специальные техники для сжатия автоматов с большим числом состояний (упорядоченные двоичные разрешающие диаграмммы), а использовать достаточно простые и наглядные алгоритмы, которые работают быстро при небольшом числе состояний, один из которых будет дополнен алгоритмом генерации сценариев.

Так же рассмотрена топология коммуникационной сети «nD-тор». На рисунке 3 показана структура такой сети. Узлы сети (маршрутизаторы) показаны крсными сферами, а желтые линии связи – физические каналы (линки), связывающие узлы в сеть. Существующие высокопроизводительные системы используют принцип параллельной обработки данных на многих вычислительных узлах. Каждый такой узел содержит несколько процессоров с локальной памятью. Для обмена информацией и синхронизации работы узлы соединяются между собой коммуникационной сетью.

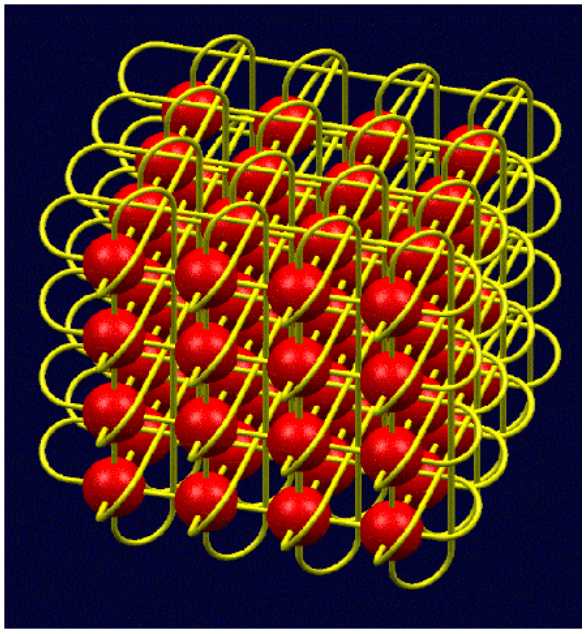

Рисунок 3 – топология сети «nD-тор»

Рассмотрены задачи тестового окружения коммуникационной сети. Они включают во-первых, удостовериться, что в проектируемом устройстве не возникает запрещенных ситуаций и, во-вторых, – все остальные ситуации отрабатываются именно так, как он ожидает. Средства проверки, следовательно, должны обеспечить анализ возможности возникновения запрещенных состояний и правильности поведения проектируемого устройства. Кроме того, разработчику должна быть постоянно доступна информация о том, насколько далеко достижение обеих целей.

Во второй главе проводится анализ моделей, применяемых при конструировании тестового окружения. Основная идея в тестировании системы как черного ящика состоит в том, что все материалы, которые доступны тесту, - это требования на систему, описывающие ее поведение, и сама система, работать с которой он может, только подавая на ее входы некоторые внешние воздействия и наблюдая на выходах некоторый результат. Все внутренние особенности реализации системы скрыты от теста. Таким образом, система представляет собой "черный ящик", правильность поведения которого по отношению к требованиям и предстоит проверить.

С точки зрения программного кода черный ящик (в данном случае блок «Линк» маршрутизатора) представляет собой набор классов (или модулей) с известными внешними интерфейсами, но недоступными исходными текстами.

Основная задача теста для данного метода тестирования состоит в последовательной проверке соответствия поведения системы требованиям. Кроме того, тест проверяет работу «Линка» в критических ситуациях - что происходит в случае подачи неверных входных значений. Все варианты критических ситуаций описаны в требованиях на систему (пропускная способность, быстродействие и другие, см. главу 1) и в тестовом окружении синтезируются воздействия для отработки данных критических ситуаций. В результате такого тестирования выявляется два типа проблем приемо-передающего блока маршрутизатора:

– Несоответствие поведения системы требованиям
– Неадекватное поведение системы в ситуациях, не предусмотренных требованиями.

Тестовое окружение для программного кода на языке программирования SystemC состоит из двух компонентов: драйвера, который обеспечивает запуск и выполнение тестируемого модуля, и заглушек, которые моделируют функции, вызываемые из данного модуля. Разработка тестового драйвера представляет собой отдельную задачу тестирования, сам драйвер отдельно тестируется, чтобы исключить неверное тестирование.

Драйвер данного тестового окружения выполняет следующие функции: вызов тестируемого модуля, передача в тестируемый модуль входных значений и прием результатов, вывод выходных значений, протоколирование процесса тестирования и ключевых точек программы

Заглушки выполняют следующие функции: не производить никаких действий (такие заглушки нужны для корректной сборки тестируемого модуля); вывод сообщения о том, что

заглушка была вызвана; вывод сообщения со значениями параметров, переданных в функцию; возврат значения, заранее заданного во входных параметрах теста; прием от тестируемого модуля значений и передача их в драйвер.

Для тестирования данной программной модели используются драйверы, представляющие собой программу с точкой входа (например, функцией main()), функциями запуска тестируемого модуля и функциями сбора результатов. Драйвер имеет одну функцию - точку входа, которой передается управление при его вызове.

Функции-заглушки могут помещаются в тот же файл исходного кода, что и основной текст драйвера. Имена и параметры заглушек совпадают с именами и параметрами "заглушаемых" функций реального блока маршрутизатора. Это требование важно не столько с точки зрения корректной сборки системы (при сборке тестового драйвера и тестируемого ПО используется приведение типов), сколько для того, чтобы максимально точно моделировать поведение реальной системы по передаче данных.

Программные модели цифровых систем (в частности, модель маршрутизатора сети «4D-тор») относятся к классу программ, использующих межпроцессное взаимодействие. Такие программы образуют комплекс, предназначенный для решения общей задачи. Каждая запущенная программа образует один или более процессов. Каждый из процессов либо использует для решения задачи свои собственные данные и обменивается с другими процессами только результатом своей работы, либо работает с общей областью данных, разделяемых между разными процессами. Для решения особо сложных задач процессы могут быть запущены на разных физических компьютерах и взаимодействовать через сеть. Преимущество использования межпроцессного взаимодействия заключается в еще большей универсальности - взаимодействующие процессы могут быть заменены независимо друг от друга при сохранении интерфейса взаимодействия. Другое преимущество состоит в том, что вычислительная нагрузка распределяется между процессами. Это позволяет операционной системе управлять приоритетами выполнения отдельных частей программного комплекса, выделяя большее или меньшее количество ресурсов ресурсоемким процессам.

При выполнении многих процессов, решающих общую задачу, используются несколько типичных механизмов взаимодействия между ними, направленных на решение следующих задач:

- передача данных от одного процесса к другому;
- совместное использование одних и тех же данных несколькими процессами;
- извещения об изменении состояния процессов.

Во всех этих случаях типичная структура каждого процесса представляет собой конечный автомат с набором состояний и переходов между ними. Находясь в определенном состоянии, процесс выполняет обработку данных, при переходе между состояниями - пересылает данные другим процессам или принимает данные от них [12].

Модель Крипке служит для описания программ. Для описания требований к программе используются темпоральные логики. Темпоральные логики являются языком, на котором удобно формулировать утверждения, использующие понятия времени (рисунок 4). Блок приема/передачи в маршрутизаторе имеет управляющие регистры для определения очередности приема данных из различных направлений коммуникационной сети. Темпоральные логики позволяют формулировать утверждения типа: Значение «a» всегда будет равно значению b (маршрутизатор всегда принимает пакеты из направления b)

- Наступит момент, когда «a» станет равно 0 (то есть маршрутизатор перестанет принимать данные)
- Значение d принимает значение 1 бесконечно много раз (маршрутизатор может принять сколько угодно данных из направления 1)

Для построения математической модели тестового окружения используются два типа темпоральных логик: CTL и CTL* (Computation Tree Logic).

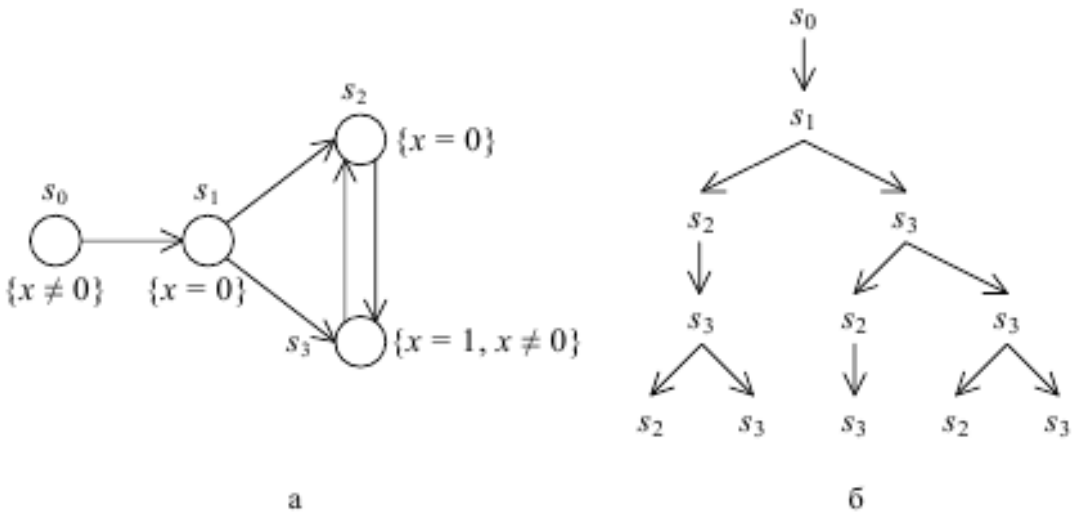

Рисунок 4 - Пример CTL модели (а) и одно из его вычисительных деревьев (б)

Для того, чтобы охарактеризовать темпоральную логику, определяют ее синтаксис и семантику. Введем понятия синтаксиса и семантики темпоральной логики. Синтаксисом темпоральной логики являются правила составления формальных утверждений. Семантикой темпоральной логики являются интерпретации данных формальных утверждений. На рисунке 4 Приведен пример модели CTL.

Модель Крипке может быть рассмотрена как диаграмма переходов с конечным набором начальных состояний. Модель Крипке часто представляют в виде набора деревьев вычислений (forest of computation trees). Для каждого начального состояния такой набор деревьев может быть построен путем разворачивания графа в дерево конечной или бесконечной высоты. В общем случае количество деревьев вычислений равняется количеству начальных состояний в модели Крипке.

Семантика логики CTL* представлена следующими обозначениями:

M,s |=f формула состояния f выполнена на модели M со стартовой вершиной s.

Отношение |=определяется естественным образом индукцией по строению формулы.

Для практической реализуемости тестового окружения модели цифрового устройства выделяются темпоральные логики, построенные с использованием согласования ограничений. Среди них наиболее простой и приемлемой по вычислительной сложности для применения в составе современных интеллектуальных систем является точечная временная логика. Эта логика может быть расширена до темпоральной логики ветвящегося времени. Реализация алгоритмов вывода в точечной временной логике основывается на решении задачи согласования временных ограничений (ЗСВО). ЗСВО задается следующим набором:

Z = (V,D,BTR,C),

где V={V1,V2,…,Vm} – конечное множество временных переменных (моментов времени);

D – область значений временных переменных (множество целых чисел);

BTR={r1,r2, rn} – конечное множество взаимоисключающих бинарных базовых временных ограничений, полное объединение которых является универсальным ограничением U (ненакладывающим вообще никаких ограничений);

C={Cij|Cij={r1,…rk};k>0;r1,…rk BTR;i,j≤m}, конечное множество ограничений, где:

Cij – ограничение над временными переменными Vi и Vj, интерпретируемое как (Vi,r1,Vj) … (Vi,rk,Vj). Если Cij состоит только из одного дизъюнкта, то оно называется точным. Для решения задачи выполнимости ЗСВО необходимо найти множество не противоречащих друг другу ограничений C*={Cij*|Cij*={rl},rl Cij}. Если ЗСВО имеет по крайне мере одно решение, то она является согласованной. Ограничения между временными примитивами представлены в виде качественных бинарных ограничений, т.к. с их помощью можно представить любые ограничения более высокого порядка.

В третей главе представлены этапы проектирования тестового окружения по методологии RUP (Rational Unified Process – рциональный объединенный процесс). Тестовое окружение является программным обеспечением, используемым внутри компании-разработчика. Доступ к нему имеют верификаторы, разрабатывающие данное ПО и выполняющие предварительную проверку

проектируемой программной системы. Также доступ к окружению имеют системные администраторы, осуществляющие регрессивное тестирование, то есть круглосуточное выполнения тестов на отдельных серверах в соответствии с программой верификации. Через определенные промежутки времени отчет о результатах тестирования направляется в отдел верификации.

Модель последовательности действий (Sequence diagram) представляет собой визуализацию взаимодействий актеров с и прецедентов, акцентирующая внимание на временной упорядоченности сообщений. Графически такая модель представляет собой таблицу, объекты в которой располагаются вдоль оси X, а сообщения в порядке возрастания времени - вдоль оси Y. Модель последовательности действий иллюстрирует распределенный во времени процесс обмена данными между объектами тестового окружения. Она отражает поток событий, происходящих в рамках работы системы.

Проанализировав диаграмму вариантов использования, представим последовательность событий при взаимодействии верификатора с тестовым окружением в процессе верификации RTL-модели приемо-передающего блока (рисунок 5).

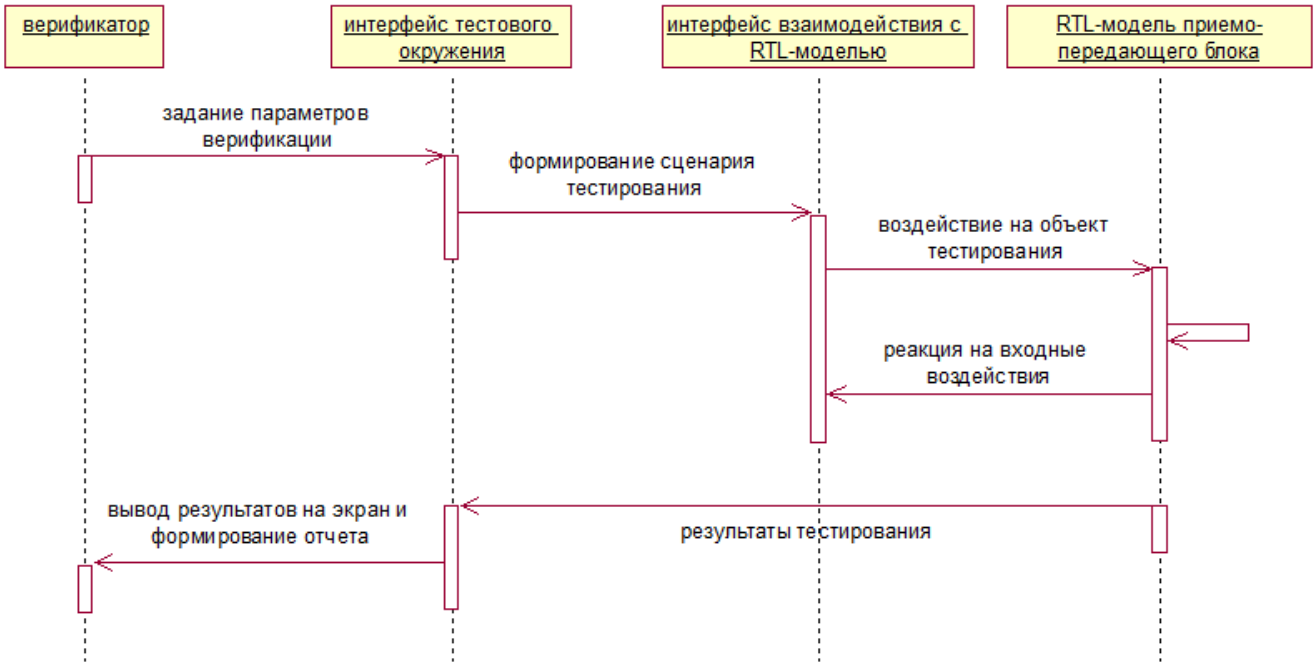

Рисунок 5 - Диаграмма последовательности действий верификации RTL-модели приемо-передающего блока

Модель кооперации отражает аналогичную информацию, что и диаграмма последовательности действий. Основное отличие заключается в том, что акцент кооперативной модели делается на распределение процессов между объектами, и визуализацию связей между объектами. На кооперативной диаграмме не показывается временная последовательность процессов. Процесс верификации, представленный на модели кооперации начинается одновременно с процессом конструирования RTL-модели приемо-передающего блока маршрутизатора и прекращается с принятием решения о завершении ее разработки.

При разработке сложных информационных систем, особенно работающих в реальном времени, используются пакеты. Пакет (Package) – это способ объединения семантически близких элементов, имеющих тенденцию изменяться совместно, и организации этих элементов в более крупные блоки.

При проектировании тестовое окружение было разбито на четыре концептуально разделенные части, представленные в виде пакетов: «Пользовательский интерфейс», «Тесты», «Входные данные», «Тестовое окружение». Для полноты представления структуры системы на уровне классов целесообразно указать системный пакет «SystemC» и пакет тестируемого блока маршрутизатора «RTL». Диаграмма пакетов представлена на рисунке 6.

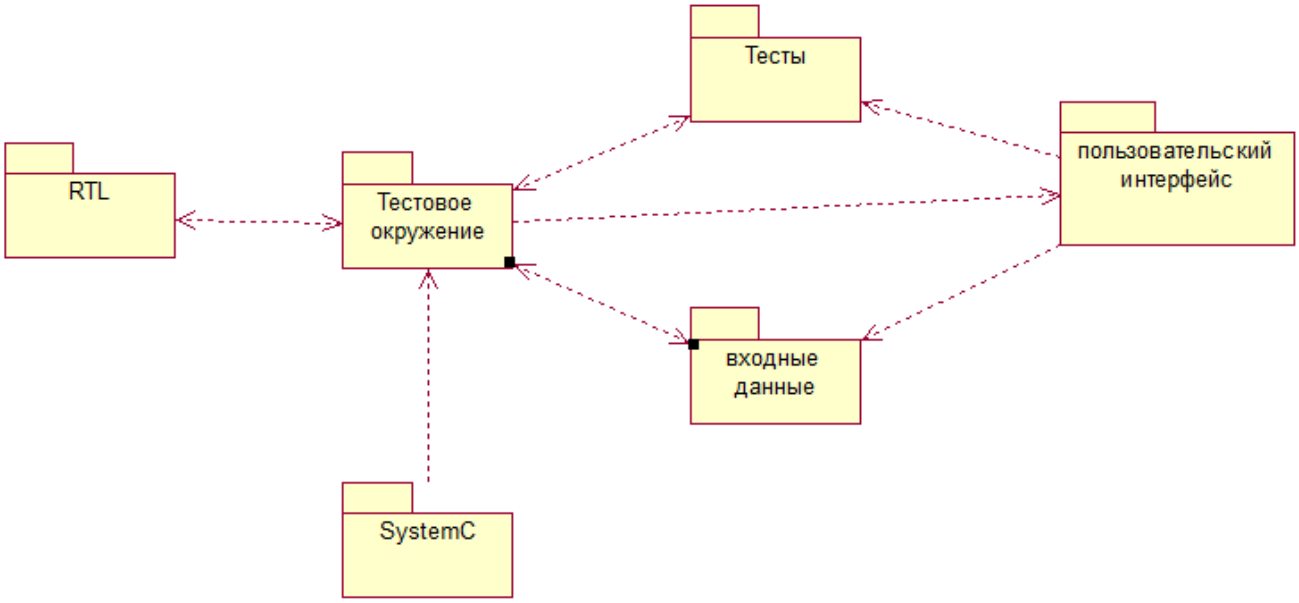

Рисунок 6 - Диаграмма пакетов тестового окружения

Диаграмма классов тестового окружения служит для графического представления программной модели системы. Данный этап проектирования программного обеспечения является заключительным перед этапом кодирования. Для избежания ошибок при дальнейшем написании программы необходимо как можно более детально продумать атрибуты классов (методы и переменные), а так же взаимодействие между ними.

Тестовое окружение работает следующим образом. Перед запуском системы вводятся конфигурационные параметры в зависимости от целей тестирования (данные, которые необходимо получить от системы). При запуске системы начинается сборка тестового окружения, исходя из введенных параметров: подключаются определенные элементарные тесты, инициализируются интерфейсы в RTL-модели. Далее выполнение программы (тестового окружения) представляет собой повторяющуюся одновременную отработку потоков класса TestBench. Происходит запись данных в RTL-модуль под управлением тестов, чтение результатов работы тестируемого блока, сохранение результатов в тестовых переменных. При выполнении условия завершения работы, отработка потоков прекращается, а интерфейсный класс выводит результаты тестирования в консоль и формирует соответствующие отчеты. При регрессивном тестировании предусмотрен периодический вывод результатов через определенные промежутки времени.

Модель развертывания тестового окружения приемопередающего блока маршрутизатора представляет собой структуру, определяющую размещение и способы взаимодействия между собой различных тестовых компонентов на реальном оборудовании. Данная диаграмма демонстрирует принципы размещения и функционирования тестов, а так же, показывает конфигурацию узлов, где производится обработка информации, и какие компоненты размещены на каждом узле.

Идея развертывания тестового окружения состоит в следующем. Блок пользователя (User A) проектирует тесты для RTL-модели, находящейся в отдельном хранилище (RTL DataBase). Подключаясь через локальную сеть (Network) к вычислительному серверу (Computational server), пользователь загружает в него RTL и посредством своего теста запускает определенные сценарии его функционирования. Все вычисления осуществляются на стороне сервера, а их результат выводится на компьютер User. После отладки очередного теста, он сохраняется в хранилище TestBench DataBase. Помимо пользовательского тестирования в режиме реального времени осуществляется регрессивное тестирование. В автоматическом режиме в вычислительный сервер загружаются тесты в порядке очередности и необходимые для них RTL-модели. Результаты регрессивного тестирования также поступают пользователю. В модели развертывания участвуют несколько вычислительных серверов, задачи между которыми распределены равномерно по критерию вычислительных затрат. Связь между объектами осуществляется посредством протокола Ethernet. Для управления совместной работой разработчиков тестового окружения локальная сеть оснащена системой управления версиями SVN.

В четвертой главе проведены экспериментальные исследования тестового окружения и процесс верификации. При разработке тестового окружения необходимо убедиться в его качестве и в том, что оно удовлетворяет требованиям спецификаций. Для этого необходимо провести испытания созданной тестовой системы. Структурная схема экспериментального стенда представлена на рисунке 7:

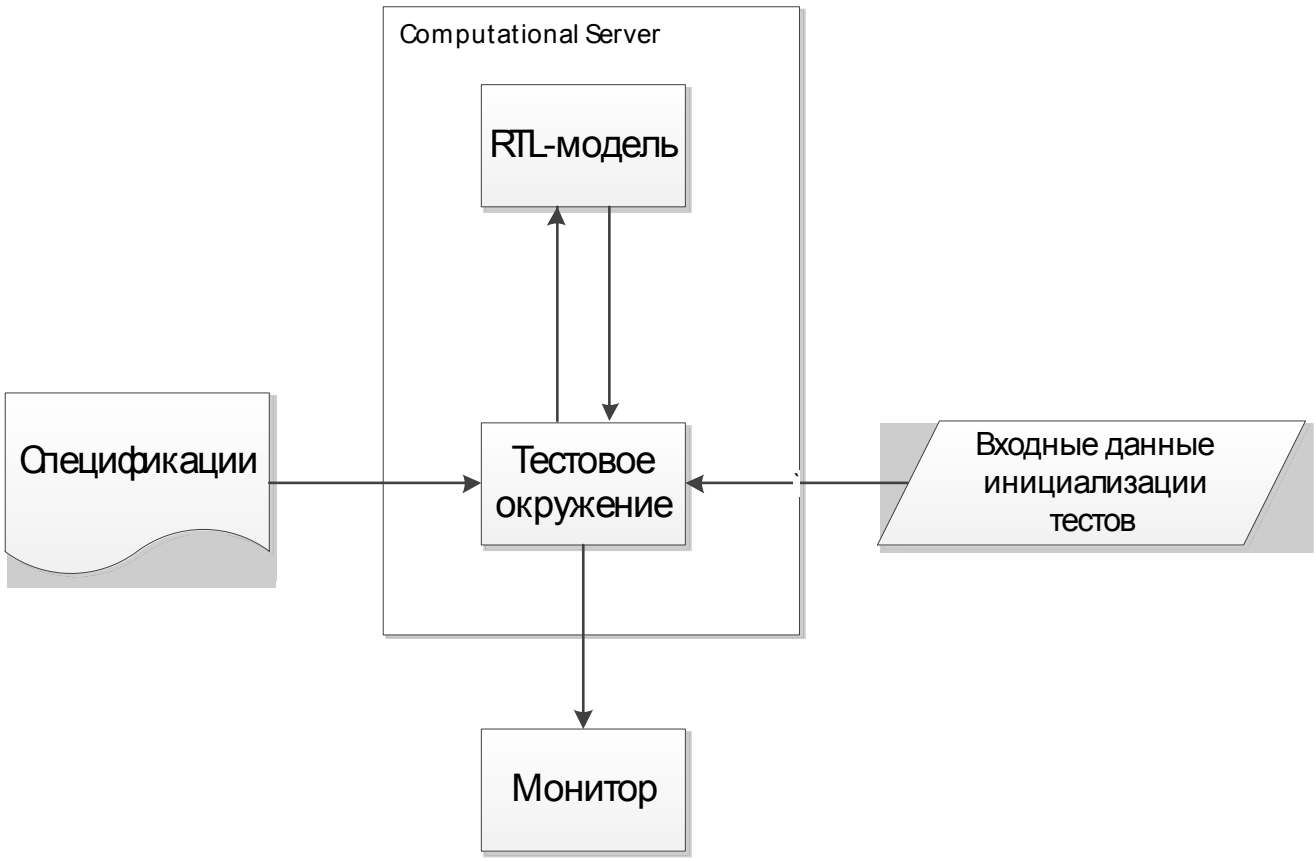

Рисунок 7 – Структура экспериментального стенда тестового окружения

В спецификациях представлены требования, которым должен удовлетворять тестируемый RTL-блок (то есть те аспекты, которые должно проверять тестовое окружение). Входные данные инициализации тестов определяют сценарий работы экспериментов для отладки тестового окружения.

В процессе верификации участвует несколько вычислительных серверов, управляемых операционной системой Suse Linux Enterprise 2011. При отладке тестового окружения применяется графическое средство симуляции и верификации RTL-моделей ”ModelSim” компании Altera (рисунок 8). ModelSim позволяет представить работу цифровых устройств в виде временных диаграмм их портов и сигналов. Это позволяет отслеживать во времени правильность работы как RTL-модели так и тестового окружения.

Экспериментирование Тестового окружения заключаются в запуске отдельных тестов, проверяющих различные аспекты RTL-модели приемопередающего блока маршрутизатора. Сборка проекта посредством утилиты Make операционной системы Linux.

Тестовое окружение предназначено для верификации исходного проекта маршрутизатора коммуникационной сети и оно не должно содержать в себе дефектов. В противном случае это существенно усложнит этап конструирования программной системы и сделает его более дорогостоящим. Представленная методика проведения экспериментов и отладки включает отработку всех функциональных модулей тестового окружения и гарантирует обнаружение всех возможных типов дефектов. Симуляция проекта посредством программы ModelSim позволяет визуализировать работу приемо-передающего блока маршрутизатора в каждый момент времени и обеспечивает в среднем обнаружение 90 процентов возможных неисправностей как тестового окружения так и RTL-модели.

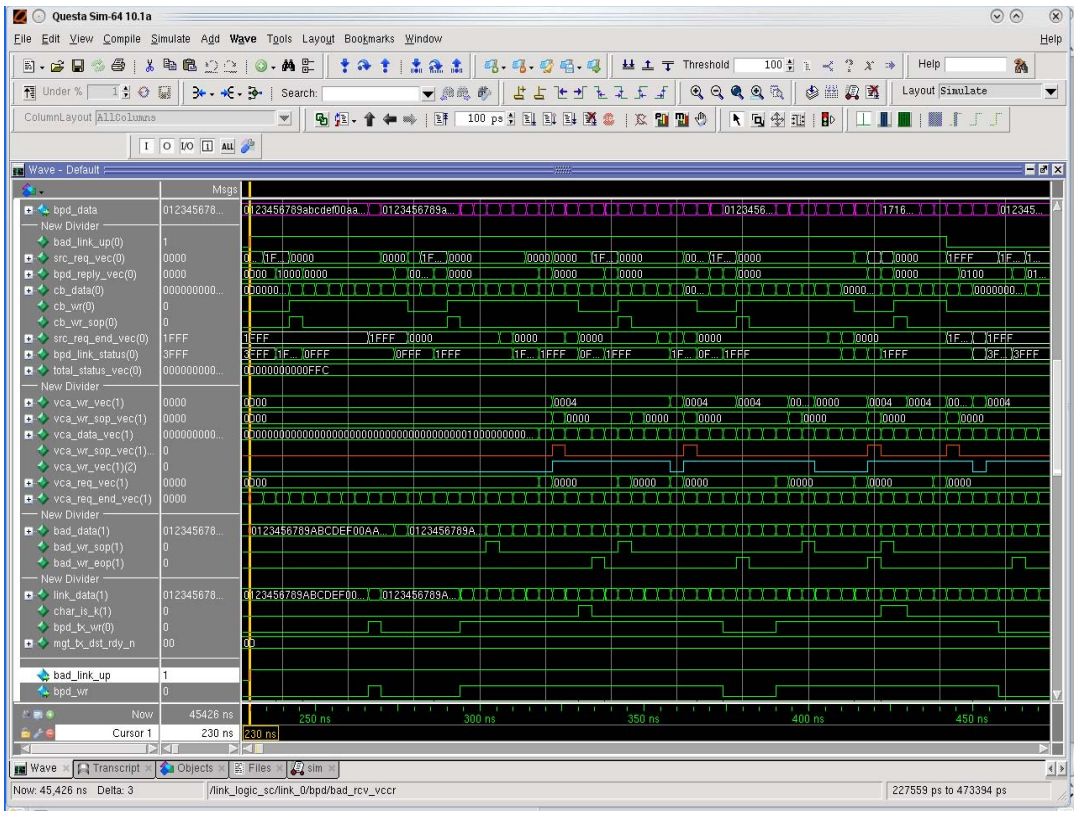

Рисунок 8- Внешний вид средства проектирования и верификации ModelSim

После разработки и отладки тестового окружения наступает непосредственно процесс верификации блока маршрутизатора. Верификация новых или недавно измененных узлов RTL-модели проводится разработчиками тестового окружения. Как только данные узлы становятся стабильными (дефекты перестают выявляться), они подключаются к общей RTL-модели маршрутизатора и подвергаются регрессивному тестированию. Данный вид тестирования необходим для определения дефектов в связях между блоками.

В заключении подводится итог выполненной работы. Представлены научные результаты. Приведены результаты проведения верификации и дана оценка применимости созданной системы для тестирования моделей цифровых устройств.

ОСНОВНЫЕ РЕЗУЛЬТАТЫ РАБОТЫ

1. Разработано тестовое окружение блока «Линк» маршрутизатора «Ангара-С», покрывающее 89 процентов исходного кода модуля и позволяющее в режиме реального времени производить контроль скорости передачи данных не менее 6Гб/с.
2. Разработаны SystemC-модули, позволяющие отслеживать пакеты данных на передающем и приемном узлах сети, контролировать целостность передачи данных, производить имитацию разрывов соединений сети и операций прерывания.
3. Проведен анализ работ по созданию тестового окружения с использованием языка SystemC, показавший уменьшение временных затрат до 30 процентов.

По результатам исследований опубликованы следующие работы

1. Иванов А. М. Принципы верификации программных моделей цифровых систем // Сборник трудов XIV Молодежная международная научно-техническая конференция учащихся, студентов, спирантов и молодых ученых «Наукоемкие технологии и интеллектуальные системы-2011». – М.: МГТУ им.Н.Э.Баумана, 24 апреля 2012 г.
2. Власов А. И., Иванов А.М. Верификация программных моделей коммуникационных сетей // Наука и образование: электронное научно-техническое издание. 2012. № [в печати].