НАУЧНОЕ ИЗДАНИЕ МГТУ ИМ. Н. Э. БАУМАНА
НАУКА и ОБРАЗОВАНИЕ
Эл № ФС77 - 48211. Государственная регистрация №0421200025. ISSN 1994-0408
электронный научно-технический журнал

Верификация программных моделей коммуникационных сетей

 # 10, октябрь 2012
DOI: 10.7463/1012.0479500
Иванов А. М., Власов А. И.
УДК. 004.45

Россия, МГТУ им. Н.Э. Баумана
norb_d@bk.ru

Введение

На сегодняшний день в области создания распределенных систем передачи и обработки данных сложилась диспропорция темпов развития трёх её основных компонентов - микропроцессорной техники, средств телекоммуникации и разработки программного обеспечения [1]. Это приводит к тому, что разрыв между техническими возможностями и технологиями обработки и передачи информации всё время увеличивается. В значительной степени, это происходит из-за отсутствия удовлетворительного решения проблемы проверки соответствия программных систем и логических схем вычислительных устройств техническим требованиям. Из опыта разработки программного обеспечения (ПО) известно, что примерно 2/3 времени, затрачиваемого на создание системы, приходится на отладку, т.е. проверку её на соответствие изначальным требованиям [1].

Верификация - это процесс определения, выполняют ли программные средства и их компоненты требования, наложенные на них в последовательных этапах жизненного цикла разрабатываемой программной системы [2].

Различные методы верификации ПО можно разделить на **формальные методы**, использующие строгий анализ математических моделей и требуемых свойств, **методы статического анализа**, при которых возможные ошибки ищутся без исполнения проверяемого ПО, **методы динамического анализа**, проводящие проверку реального поведения проверяемой системы в рамках некоторых сценариев ее работы и **экспертные методы**, выполняемые на основе знаний и опыта экспертов [3 - 6]. На рисунке 1 изображена диаграмма зависимости требуемой скорости верификации (тактов в секунду) от сложности объектов.

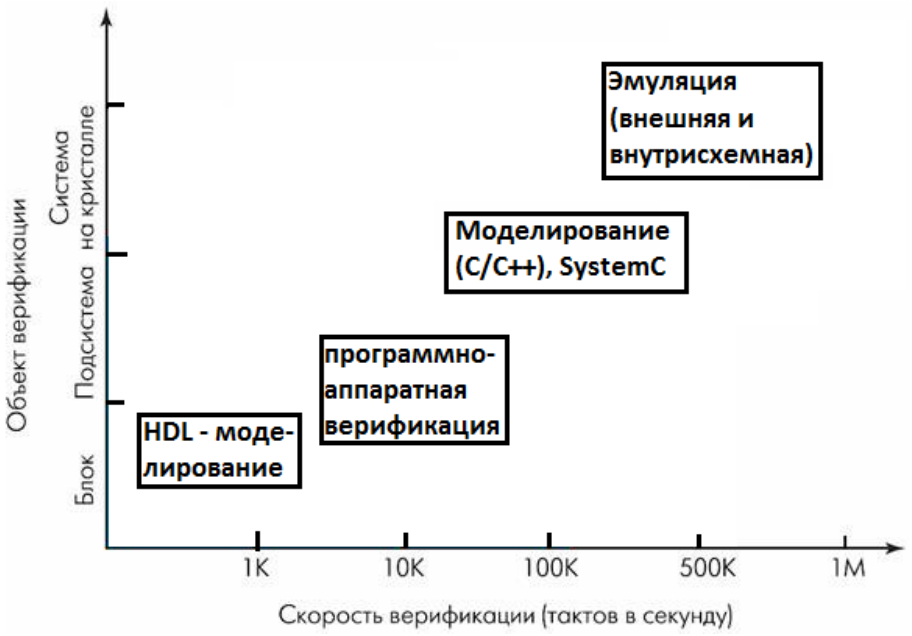

Рисунок 1 – Рост требуемой скорости верификации при переходе к более сложным объектам

Вопросы верификации программных моделей электронных устройств рассматривались в ряде работ [2-10]. Так, Куляминым В.В. в работе «Интеграция методов верификации программных систем» [2] проведено обобщение методов верификации и предложена концепция совместного применения комбинированных методов для более полного тестирования программных систем. А. Лохов в работе «Функциональная верификация СБИС в свете решений Mentor Graphics» провел анализ преимущества проведения функционального тестирования с использованием средств моделирования и верификации компании Mentor Graphics. В работе Дж. Холзмана «Модельно-управляемая верификация программного обеспечения» описаны недостатки тестирования ПО без применения специальных техник и инструментов верификации и предложена методика построения тестового окружения с помощью испытательных стендов [11]. Основная проблема методов верификации заключается в возрастающей сложности и временных затратах при их разработке, апробации и интеграции в существующие среды разработки ПО. Известны два основных подхода к решению основной проблемы верификации [5, 6, 8].

Первый предполагает усовершенствование самих средств верификации, применение эффективных инструментов для верификации: имитационного моделирования, аппаратной эмуляции, программно-аппаратной верификации, аналогового, цифрового и смешанного моделирования [8]. При этом предполагается, что средства верификации должны поддерживать базовые языки проектирования и верификации (VHDL, Verilog, VHDL_AMS, Verilog_A, Spice, C, C++, SystemC, System Verilog, MATLAB, PSL assertions и др.).

Второй подход заключается в изменении самой методики верификации, а именно – в переносе соответствующих процедур верификации на более ранние этапы проектирования. Это, как правило, подразумевает создание системных тестов, моделирование на уровне транзакций, верификацию интерфейсов различных подсистем одновременно с их проектированием, т.е. возможность верификации системы, отдельные блоки которой представлены на разных уровнях абстракции [7]. В этом случае, проектирование на концептуально-абстрактном уровне предполагает наличие визуальных моделей, элементы которых далее, на системном уровне, описываются с использованием языков высокого уровня (C, C++, SystemC и System Verilog). Такой подход позволяет начинать верификацию системы, не ожидая детальной проработки всех блоков и интерфейсов. Причем интерфейсы между блоками и тестовыми моделями, описанными на разных уровнях абстракции, эффективно реализуют механизм транзакций [7].

Методы имитационного моделирования и тестирования [1] довольно эффективны на самых ранних стадиях отладки, когда проектируемая система всё ещё изобилует ошибками, но результативность этих методов быстро снижается, как только система становится чище.

Достойной альтернативой имитационному моделированию и тестированию являются методы **формальной верификации** [7]. При имитационном моделировании и тестировании исследуются только некоторые из возможных сценариев поведения проектируемой системы, поэтому остаётся открытым вопрос о том, не содержится ли фатальная ошибка в незадействованных траекториях. Формальная верификация же обеспечивает исчерпывающий анализ всех возможных вариантов поведения системы.

Техника верификации, получившая название **проверки на модели** [8], является одним из наиболее перспективных и широко используемых подходов к решению проблемы автоматизации отладки и проверки правильности программ. Если проектируемая система не обладает желаемым свойством, то результатом проверки на модели будет неверное поведение системы, опровергающее это свойство. Это поведение дает информацию для понимания причины ошибки, и ключ к решению возникшей проблемы.

В итоге можно отметить, что при проведении верификации возникают проблемы возрастающей сложности и временных затрат поиска ошибок в программной системе на различных этапах ее разработки.

Целью работы является разработка тестового окружения для верификации программных моделей цифровых коммуникационных устройств, направленная на уменьшение вычислительных затрат для проверки соответствия ПО техническим требованиям.

Для достижения поставленной цели в работе решены задачи формализации требований к системе, анализа существующих методов верификации, разработки на их основе тестового окружения для цифровых коммуникационных систем.

По сравнению с известными результатами в данной работе предложена модель тестового окружения, адаптированная для верификации программных моделей цифровых устройств высокоскоростных отказоустойчивых коммуникационных сетей.

1 Обобщенный маршрут верификации программной модели коммуникационной сети

Рассмотрим маршрут верификации RTL-модулей (Register-Transfer Level) высокоскоростной отказоустойчивой коммуникационной сети на примере сети с топологией «nD-тор» [9]. Объем и сложность проекта не позволяют ограничиться программно-аппаратной верификацией и требуют использования специализированных языков и сред проектирования, методов статического и динамического анализа [7].

Статический анализ проводится посредством формальных инспекций разрабатываемого кода.

Динамический анализ проводится посредством создания сценариев работы сети на языке C++. Параллельность работы всей системы, а так же связь RTL (написанном на низкоуровневом языке Verilog) с высокоуровневым сценарием осуществляется библиотекой SystemC.

RTL-модуль взаимодействуя с тестовым окружением образует единый работоспособный блок программной системы, тестируемый отдельно от остальной части программы.

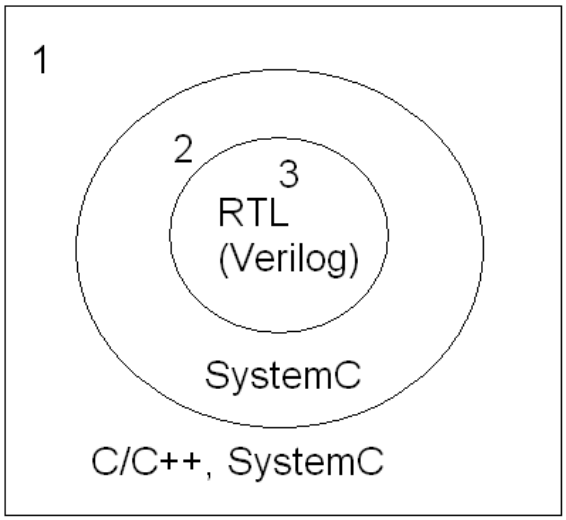

Рисунок 2 – Взаимодействие RTL с тестовым окружением

На рисунке 2 представлена общая модель реализации тестового окружения. Блок под номером 1 реализует симуляцию работы сети и тестовые сценарии, блок 2 – интерфейс между тестируемым блоком системы и системой верификации, блок 3 – RTL-модуль, представляемый в виде «черного ящика».

Выбор среды **SystemC** для реализации интерфейса между тестируемым блоком системы и системой верификации обусловлен тем, что эта среда представляет собой надстройку стандартного языка программирования C++, реализованную в виде отдельных библиотек специальных классов. Данные библиотеки содержат в себе конструкции, позволяющие создавать модели программных алгоритмов, аппаратных архитектур, интерфейсов и схем на системном уровне, т.е. практически всех компонентов встроенных систем [6]. При этом, развёрнутое описание процесса работы всей системы представляет собой программу C++, которая при исполнении ведёт себя так же, как и система. На рисунке 3 приведены основные этапы верификации программных моделей систем на кристалле [2].

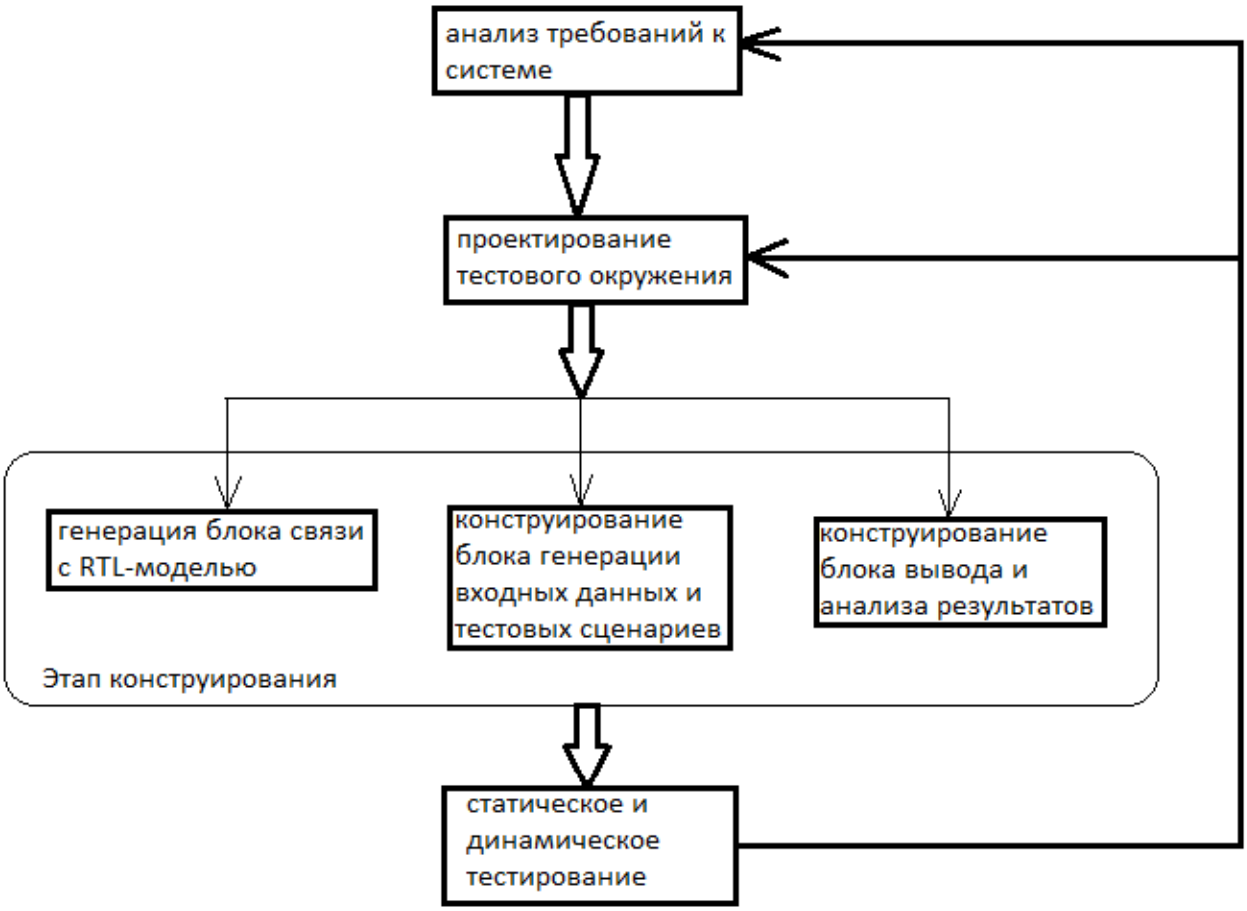

Рисунок 3 – Основные этапы создания тестового окружения программных моделей «Систем на кристалле»

Основная цель в использовании среды SystemC – добиться сокращения времени проектирования систем на кристалле [5] за счет исключения из процесса проектирования этапа перехода с одного языка описания на другой от поведенческой модели на языке C++ к синтезируемой RTL-модели на HDL-языке.

2 Принципы построения тестового окружения маршрутизатора коммуникационной сети

Коммуникационная сеть топологии «4D-тор» состоит из узлов, соединенных между собой в конфигурации «четырехмерного тора». В каждом узле сети находится маршрутизатор, выполняющий функции передачи пакетов по сети, проверки правильности информации, маршрутизации данных, а так же связи с локальными вычислительными узлами, находящимися так же в каждом узле сети. Основной задачей сети является связь и синхронизация работы вычислительных процессоров в едином вычислительном ядре.

Ввиду большого объема программной модели маршрутизатора коммуникационной сети создание тестового окружения для всей системы трудоемко и нерационально. Весь проект построен по модульной структуре и верификация проводится параллельно с разработкой сети, по мере создания (изменения) конкретных модулей системы. Кроме внешнего окружения, в рамках которого должна работать среда верификации, необходимо также определить ее **каркас** — некоторый базовый набор компонентов, реализующих основной набор функций и поддерживающие основные потоки данных внутри системы. К этому каркасу будут добавляться другие компоненты, поддерживающие вспомогательные и менее значимые функции [2].

Модель, представленная на рисунке 4, удобна в плане времени и трудоемкости анализа каждого конкретного окружения. Библиотека SystemC предоставляет возможность автоматического построения интерфейса (см. рисунок 4).

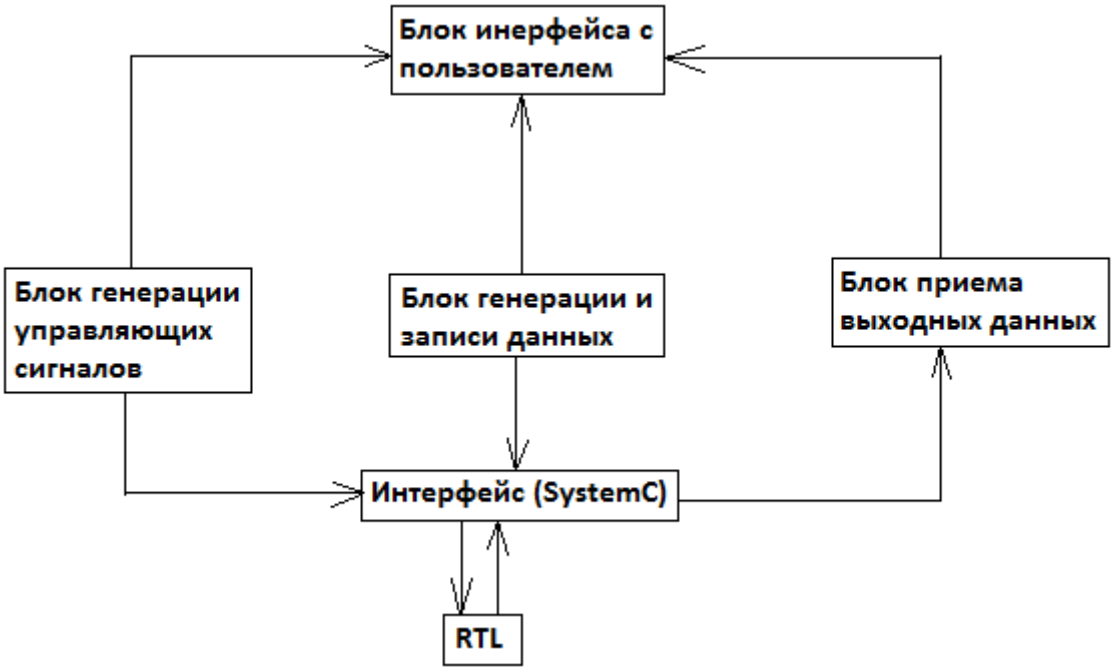

Рисунок 4 – Иерархия тестового окружения

Для симуляции проекта и проверки тестовых воздействий может использоваться среда моделирования и отладки ModelSim SE фирмы Mentor Graphics, в которой по временным диаграммам отдельных внутренних и интерфейсных сигналов проверяется корректность работы отдельных блоков и системы в целом [3].

3 Методика синтеза каркаса верификации модели маршрутизатора

Для верификации модели маршрутизатора предлагается использовать в качестве основы для построения среды верификации архитектурный каркас **RTL-модели** коммуникационной сети. Это решение вызвано тем обстоятельством, что на построение собственной модели для тестирования приходится 1/2 от общих временных затрат на разработку цифрового устройства. При этом, вместо полноценной тестовой модели создается надстройка над *RTL*-модулем в виде интерфейса с высокоуровневым тестовым окружением. Обычно в ходе тестирования на основе моделей необходимо сделать следующее [2]:

- определить модель поведения тестируемой системы, формализующую требования к этому поведению;

- проанализировать структуру модели для выбора критериев покрытия и отдельных целей тестирования, и определить эти критерии и цели;

- построить среду выполнения тестов, включающую средства мониторинга для протоколирования внешних действий, реакций системы, и, возможно, внутренних ее событий, а также тестовые оракулы (рис. 4) — программные компоненты, определяющие соответствие или несоответствие наблюдаемого поведения системы и модели (обычно такая среда состоит из библиотеки поддержки выполнения тестов, набора тестовых оракулов для всех проверяемых компонентов и набора адаптеров, связывающих эти компоненты с их дескрипторами, дескрипторы в большинстве случаев генерируются автоматически из ранее построенной модели);

- построить, автоматически или с привлечением человека, набор тестовых сценариев, определяющих последовательности вызова различных операций тестируемой системы или посылки ей сообщений или сигналов и данные, передаваемые в качестве параметров операций и сообщений;

- выполнить тестовые сценарии, протоколируя всю информацию, касающуюся соответствия наблюдаемого поведения системы и ее модели, а также покрытых во время тестирования ситуаций;

- провести анализ результатов тестов, в ходе которого выявляются и анализируются ошибки в системе или ее модели (проявляющиеся как несоответствия между ожидаемым и реальным поведением), а также анализируется достигнутое тестовое покрытие и принимается решение либо о создании дополнительных тестов, либо об окончании их разработки [3-5].

На рисунке 5 отображена методика разработки тестового окружения применительно к верификации моделей цифровых устройств.

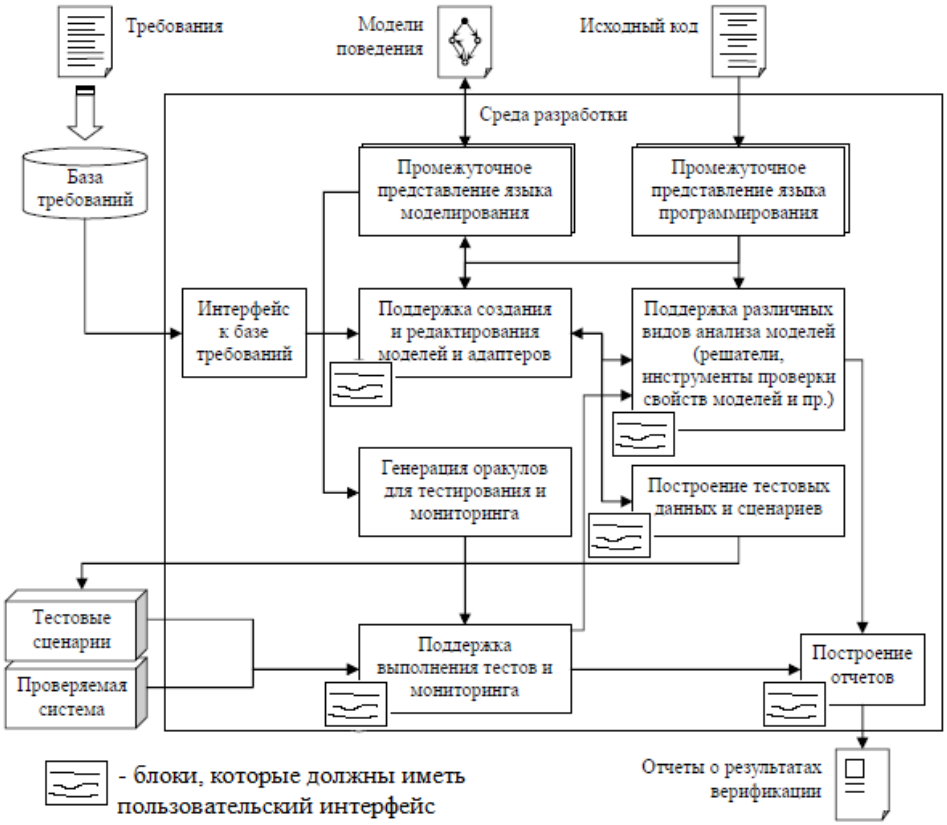

Рисунок 5 – архитектура системы верификации программного обеспечения [2, 8]

Тестирование любой системы начинается с анализа требований к системе. На данном этапе определяются цели и задачи будущих тестов. Для обоснования адекватности проводимой верификации необходимо, чтобы каждая часть используемых моделей и каждый элемент отчетов о найденных ошибках могли быть соотнесены с каким-то элементом исходных требований. Далее производится проектирование системы верификации. Детально разрабатываются блоки, процедуры и функции, необходимые для выполнения требований, поставленных на предыдущем этапе. На данном этапе учитываются особенности библиотеки SystemC, позволяющие имитировать параллельную работу цифровых устройств. Так все процедуры представляются в виде потоков (частей программы, выполняющихся каждый раз при появлении положительного фронта тактирующего сигнала). Далее по разработанному проекту производится создание тестового окружения. Сперва, средствами SystemC, производится генерация класса, обеспечивающего связь тестируемой модели с тестовым окружением. Далее необходимо провести конструирование всех потоков, описанных на предыдущем этапе. Следующим этапом является непосредственно симуляция разработанной системы и тестирование модели. По результатам тестирования могут вноситься изменения в проект и последующая доработка кода. При обнаружении несоответствий тестируемой RTL-модели требованиям, после чего исправляются найденные ошибки и процесс верификации

повторяется. Так же по мере разработки самого цифрового устройства могут возникать изменения в требованиях проекта. В таком случае повторяется вышеописанная последовательность действий. Представленная архитектура позволяет разделить между собой функциональные блоки тестового окружения, что позволяет быстро изменять отдельные ее части и адаптироваться под постоянно развивающуюся в процессе проектирования модель цифрового устройства.

4 Отработка методики верификации цифровых систем на примере верификации блока «Link» маршрутизатора

В качестве примера рассмотрим применение предлагаемой методики верификации на примере одного из основных блоков модели маршрутизации коммуникационной сети - **блока «*Link*».** Он состоит из двух частей: «блок анализа и демультиплексор» (БАД), который предназначен для приема информационных пакетов из сети и направлении их по одному из виртуальных каналов в соответствии с маршрутом и «блок передачи данных» (БПД), который осуществляет справедливый арбитраж пакетов (пропускание пакетов в сеть из нескольких виртуальных каналов последовательно в соответствии с очередностью). Данные блоки должны удовлетворять требованиям быстродействия (обеспечивать минимальные задержки прохождения пакетов) и отказоустойчивости (обеспечивать гарантии сохранения целостности пакета при прохождении его через сеть).

Целью построения тестового окружения является:

- проверка правильности приема пакетов;
- оценка быстродействия прохождения пакетов через *Link;*
- проверка арбитража пакетов;
- оценка отказоустойчивости блока.

Тестовое окружение концептуально состоит из трех блоков: блок связи с RTL-моделью, драйвер (генератор входных данных и тестовых сценариев) и монитор (блок приема выходных сигналов и генерации пользовательских сообщений). Блок связи создан автоматически средствами библиотеки SystemC. Данный блок осуществляет связь модели цифрового устройства с тестовым окружением и состоит из SystemC-портов, передающих данные в модель. Драйвер формирует пакеты данных для передачи их через тестируемую модель и необходимые управляющие сигналы для обеспечения ее работы. Так же драйвер содержит процедуры тестовых сценариев, которые имитируют разрыв связи между узлами коммуникационной сети, переполнение внутренних буферов исследуемой модели, потерю и изменение данных при прохождении их по сети, одновременные запросы на передачу

данных через один узел из различных каналов приема. Монитор состоит из процедур, принимающих выходные данные из RTL-модели, анализирующих принятую информацию и сохраняющих ее в собственных буферах памяти. Данный блок содержит процедуры вывода информации о результатах тестирования и накопления статистики о работе тестируемого устройства. Кроме того, в мониторе ведется учет скорости передачи информации между узлами коммуникационной сети.

В результате тестирования проверялись следующие функции приемо/передающего блока.

Для блока БАД:

- приём пакетов из сети;
- анализ пакетов и передача в соответствующий виртуальный канал или входную очередь локального вычислительного узла;
- контроль целостности данных пакета;
- определение направления передачи пакетов на основании номера виртуального канала;
- контроль целостности данных путём подсчёта пакетов, контроля CRC (контрольной суммы) заголовочной части – и тела пакета.

Тестовые сценарии для блока БАД строились на основании диаграммы состояний, показанной на рисунке 6.

Для блока БПД:

- способность осуществлять справедливый арбитраж запросов и установление соединений посредством кроссбара с различными виртуальными каналами маршрутизатора;
- прием данных с выхода кроссбара и помещение их во внутренний буфер БПД;
- передача принятых во внутренний буфер БПД данных, не дожидаясь завершения приёма, на следующий маршрутизатор;
- осуществление контроля и удаления из внутреннего буфера БПД пакетов, успешно принятых следующим маршрутизатором, а также повторную передачу пакетов при возникновении ошибки на приёмной стороне).

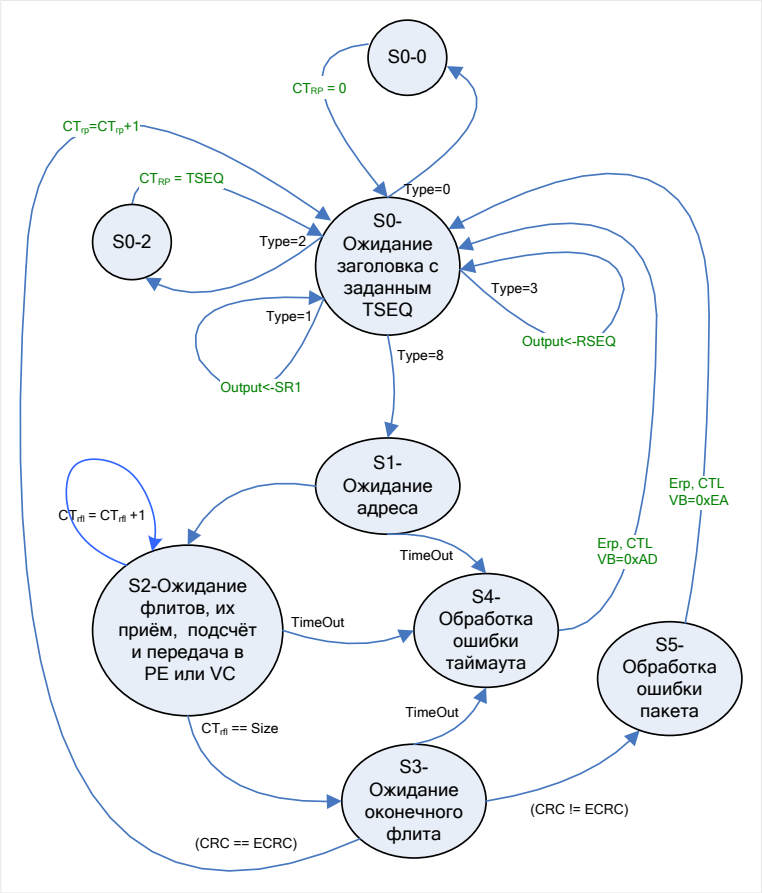

Рисунок 6 – Диаграмма состояний блока приема

Исследования работы приемо/передающего блока проводились с использованием языка моделирования и верификации SystemC, а так же с применением среды разработки тестового окружения ModelSim фирмы «Mentor Graphics» [3].

По результатам исследований предлагается использовать представленную на рисунке четыре иерархию тестового окружения для верификации моделей цифровых систем. Данная иерархия тестового окружения предполагает разделение основных функциональных блоков, что способствует отладке и увеличивает «время жизни» тестов по мере разработки самой тестируемой модели. При проектировании тестового окружения с использованием среды разработки ModelSim наблюдается уменьшение временных затрат на 40 % по сравнению с классической разработкой.

Результаты работы получены в рамках проекта "Ангара-С" по разработке высокоскоростной отказоустойчивой коммуникационной сети на предприятии НИЦЭВТ.

Список литературы

1. Гаранина Н.О. Верификация распределенных систем с использованием аффинного представления данных, логик знаний и действий: дисс…. канд. физ.-мат. наук. Новосибирск, 2004. 165 с.
2. Кулямин В.В. Интеграция методов верификации программных систем // Программирование. 2009. Т. 35, № 4. С. 41-55.
3. Лохов А. Функциональная верификация СБИС в свете решений Mentor Graphics // Бюллетень выставки «Метрология и измерительные системы». 2004.
4. Власов А.И., Зинченко Л.А., Макарчук В.В., Родионов И.А. Автоматизированное проектирование наносистем: учеб. пособие. М.: Изд-во МГТУ им.Н.Э.Баумана, 2011. 184 с.: ил. (Библиотека "Наноинженерия": в 17 кн. Кн. 13).
5. Шахнов В.А., Мороз А.А., Михненко А.Е., Власов А.И. Операционная система реального времени - MeatrixRealTime как основа для построения экспериментальных систем обработки сигналов в реальном времени // 2-ая Межд. конф. СНГ "Молодые ученые - науке, технологиям и профобразованию для устойчивого развития: проблемы и новые решения" (Москва, октябрь 2000 г.). М., 2000. Часть 2, 3. С. 100-103.
6. Maraia V. The Build Master: Microsoft's Software Configuration Management Best Practices. Addison-Wesley Microsoft Technology, 2005. 288 p.
7. Robles G. Debian Counting. Available at: http://libresoft.dat.escet.urjc.es/debian-counting/ .
8. Кулямин В.В. Методы верификации программного обеспечения // XV Всероссийская научно-методическая конференция Телематика'2008. Всероссийский конкурс обзорно-аналитических статей по приоритетному направлению "Информационно-телекоммуникационные системы", 2008.
9. Шагурин И.В., Канышев В.А. Применение языка SystemC и средств разработки на его основе для проектирования «Систем на кристалле» // Инженерная практика. 2006. Т. 32, № 9. С. 23–32.
10. Fin A., Fummi F., Martignano M., Signoretto M. SystemC : a homogenous environment to test embedded systems // Proceedings of the 9th international symposium on Hardware / software codesign (Copenhagen, Denmark, April 2001). ACM, New York, NY, USA, 2001. P. 17-22. DOI: 10.1145/371636.371657

11. Gerard J. Holzmann, Rajeev Joshi. Model-Driven Software Verification // Model Checking Software: Proc. Of the 11th International SPIN Workshop. Berlin: Heidelberg Springer, 2004. P. 76-91. (Lecture Notes in Computer Science, vol. 2989). DOI: 10.1007/978-3-540-24732-6_6

SCIENTIFIC PERIODICAL OF THE BAUMAN MSTU
SCIENCE and EDUCATION
EL № FS77 - 48211. №0421200025. ISSN 1994-0408
electronic scientific and technical journal

Verification of software models of communications networks

 # 10, October 2012
DOI: 10.7463/1012.0479500
Ivanov A.M., Vlasov A.I.

Russia, Bauman Moscow State Technical University
norb_d@bk.ru

Development of software models of digital devices involves verification of compliance of the developed software project with initial requirements to logical functions of the device. In this paper the authors consider principles of developing test software environment for design of digital logic modules of a high-speed communications network. The verification methods for software models of logic modules of distributed data transmission and processing systems. The test software environment for a router block is responsible for reception and transmission of data packets.

Publications with keywords:test bench, router
Publications with words:test bench, router

References

1. Garanina N.O. *Verifikatsiia raspredelennykh sistem s ispol'zovaniem affinnogo predstavleniia dannykh, logik znanii i deistvii. Kand. diss.* [Verification of distributed systems using the affine representation of data, the logics of knowledge and action. Cand. diss.]. Novosibirsk, 2004. 165 p.

2. Kuliamin V.V. Integratsiia metodov verifikatsii programmnykh sistem [Integration of methods of verification of software systems]. *Programmirovanie*. 2009, vol. 35, no. 4, pp. 41-55.

3. Lokhov A. Funktsional'naia verifikatsiia SBIS v svete reshenii Mentor Graphics [VLSI functional verification in light of Mentor Graphics]. *Biulleten' vystavki «Metrologiia i izmeritel'nye sistemy»* [Bulletin of the exhibition "Metrology and measurement systems"]. 2004.

4. Vlasov A.I., Zinchenko L.A., Makarchuk V.V., Rodionov I.A. *Avtomatizirovannoe proektirovanie nanosistem* [Computer-aided design of nanosystems]. Moscow, Bauman MSTU Publ., 2011. 184 p. (*Biblioteka "Nanoinzheneriia":* v 17 kn. Kn. 13 [Library "Nanoengineering". In 17 books. Book 13]).

5. Shakhnov V.A., Moroz A.A., Mikhnenko A.E., Vlasov A.I. Operatsionnaia sistema real'nogo vremeni - MeatrixRealTime kak osnova dlia postroeniia eksperimental'nykh sistem obrabotki signalov v real'nom vremeni [Real Time Operating System - MeatrixRealTime as the basis for the construction of experimental systems for signal processing in real time]. *2-aia Mezhd. konf. SNG "Molodye uchenye - nauke, tekhnologiiam i profobrazovaniiu dlia ustoichivogo razvitiia: problemy i novye resheniia"* [2nd Int. Conf. of CIS "Young Scientists - Science, Technology and Vocational education for sustainable development: challenges and new solutions"]. Moscow, October 2000, Ch. 2, 3, Moscow, 2000, pp. 100-103.

6. Maraia V. *The Build Master: Microsoft's Software Configuration Management Best Practices*. Addison-Wesley Microsoft Technology, 2005. 288 p.

7. Robles G. *Debian Counting*. Available at: http://libresoft.dat.escet.urjc.es/debian-counting/ .

8. Kuliamin V.V. Metody verifikatsii programmnogo obespecheniia [Methods of verification of software]. *15-ia Vserossiiskaia nauchno-metodicheskaia konferentsiia Telematika'2008. Vserossiiskii konkurs obzorno-analiticheskikh statei po prioritetnomu napravleniiu "Informatsionno-telekommunikatsionnye sistemy"* [15-th all-Russian scientific-methodical conference Telematics'2008. All-Russian competition of review and analytical articles on the priority direction "Information-telecommunication systems"], 2008.

9. Shagurin I.V., Kanyshev V.A. Primenenie iazyka SystemC i sredstv razrabotki na ego osnove dlia proektirovaniia «Sistem na kristalle» [Application of the SystemC language and development tools on its basis for the design of «Systems on chip»]. *Inzhenernaia praktika* [Engineering practice], 2006, vol. 32, no. 9, pp. 23–32.

10. Fin A., Fummi F., Martignano M., Signoretto M. SystemC : a homogenous environment to test embedded systems. *Proceedings of the 9th international symposium on Hardware / software codesign*, Copenhagen, Denmark, April 2001. ACM, New York, NY, USA, 2001, pp. 17-22. DOI: 10.1145/371636.371657

11. Gerard J. Holzmann, Rajeev Joshi. Model-Driven Software Verification // *Model Checking Software: Proc. Of the 11th International SPIN Workshop*. Berlin, Heidelberg Springer, 2004, pp. 76-91. (*Lecture Notes in Computer Science*, vol. 2989). DOI: 10.1007/978-3-540-24732-6_6