УДК 681.32
https://doi.org/10.37661/1816-0301-2023-20-4-24-37

Оригинальная статья
Original Paper

Моделирование дискретных управляющих систем с параллелизмом поведения

Д. И. Черемисинов, Л. Д. Черемисинова✉

Объединенный институт проблем информатики
Национальной академии наук Беларуси,
ул. Сурганова, 6, Минск, 220012, Беларусь
✉*E-mail: cld@newman.bas-net.by*

Аннотация

Цели. Рассматривается задача функциональной верификации устройств управления относительно спецификации на их проектирование. При решении задач реализации и тестирования дискретных систем приходится иметь дело с наличием параллелизма в поведении взаимодействующих объектов управления, что отображается также и в задании на проектирование устройств управления ими. Цель исследования заключается в разработке метода имитационного моделирования описаний дискретных систем, который позволяет динамически тестировать поведение таких систем на области, ограниченной их возможным функционированием.

Методы. В работе рассматривается класс систем управления с параллелизмом происходящих в них процессов, позволяющим линеаризовать их выполнение. Для задания спецификации таких систем управления предлагается использовать язык ПРАЛУ параллельных алгоритмов управления, в основе которого лежат сети Петри и который позволяет упорядочивать во времени события, происходящие в процессе работы устройства. Предлагается объектно-ориентированный подход к моделированию описания алгоритма управления на уровне транзакций. Для этого разработана модель TLM (Transaction-Level Modeling) описаний на языке ПРАЛУ устройств с параллелизмом поведения. Модель уровня транзакций описывает систему набором взаимодействующих процессов, которые выполняются параллельно и определяют ее поведение во времени.

Результаты. Определены ключевые понятия модели TLM для моделирования описаний алгоритмов управления на языке ПРАЛУ: структура данных, транзакции, процессы и барьерной механизм синхронизации параллельно выполняющихся процессов. Предложен метод преобразования описания алгоритма на языке ПРАЛУ в модель TLM, который основан на представлении операций языка в виде композиций элементарных операций, выполняющихся последовательно. Набор таких операций составляет базис алгоритмического разложения параллельного алгоритма на языке ПРАЛУ в программу на промежуточном языке, которая выполняется строго последовательно. Разработаны трансляторы этой программы на языки Verilog и C, результаты их компиляции представляют симуляторы поведения системы управления.

Заключение. Предложенный метод имитационного моделирования может быть использован при создании испытательного стенда для функциональной верификации схемной реализации устройств управления с параллелизмом поведения. При этом тестовые последовательности для верификации схемной реализации могут генерироваться динамически – в процессе моделирования описания алгоритма на языке ПРАЛУ непосредственно устройства управления или системы, включающей алгоритм управления и алгоритмы поведения управляемых объектов.

Ключевые слова: параллельный алгоритм, устройство управления, имитационное моделирование, функциональная верификация, модель TLM, язык ПРАЛУ

Для цитирования. Черемисинов, Д. И. Моделирование дискретных управляющих систем с параллелизмом поведения / Д. И. Черемисинов, Л. Д. Черемисинова // Информатика. – 2023. − Т. 20, № 4. – С. 24–37. https://doi.org/10.37661/1816-0301-2023-20-4-24-37

Конфликт интересов. Авторы заявляют об отсутствии конфликта интересов.

Поступила в редакцию | Received 17.08.2023
Подписана в печать | Accepted 22.09.2023
Опубликована | Published 29.12.2023

Simulation of discrete control systems with parallelism of behavior

Dmitry I. Cheremisinov, Ljudmila D. Cheremisinova✉

The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, st. Surganova, 6, Minsk, 220012, Belarus
✉*E-mail: cld@newman.bas-net.by*

Abstract

Objectives. The problem of functional verification of control devices with respect to their design specification is considered. When solving the problems of implementing and testing of discrete systems, one has to deal with the presence of parallelism in the behavior of interacting control objects, which is also displayed in the assignment for designing control systems. The aim of the work is to develop a method for simulating descriptions of such systems, which allows their behavior testing dynamically on the area limited by their possible functioning.

Methods. The paper considers a class of control systems with parallelism of the processes occurring in them, which permits linearization of their execution. To specify the behavior of such control systems, it is proposed to use the PRALU language of parallel control algorithms, which is based on Petri nets and which allows to order events occurring during the device operation. An object-oriented approach to simulation of the description of the control algorithm at the transaction level is proposed. For this purpose, a TLM (Transaction-Level Modeling) model has been developed for describing the devices with behavior parallelism in PRALU language. The transaction level model describes a system as a set of interacting processes that run in parallel and determine the behavior of the system over time.

Results. The key concepts of the TLM model for simulating the descriptions of control algorithms in the PRALU language are defined: data structure, transactions, processes, and a barrier mechanism for synchronization of parallel processes. A method is proposed for transforming the description of an algorithm in the language into a TLM model, which is based on the representation of language operations as compositions of elementary operations that are performed sequentially. The set of these operations forms the basis for the algorithmic decomposition of a parallel algorithm in PRALU language into intermediate language program that is executed strictly sequentially. Translators of this program into the Verilog and C languages have been developed, the results of their compilation are simulators of the behavior of control system.

Conclusion. The proposed simulation method can be used to create a test bench for functional verification of the circuit implementation of control devices with behavior parallelism. In this case, test sequences for verifying the circuit implementation can be generated dynamically – in the process of simulating the description of the algorithm in the PRALU language directly the control device or system, which include the control algorithm and the algorithms of controlled objects behavior.

Keywords: concurrent algorithm, control device, simulation, functional verification, TLM model, PRALU language

For citation. Cheremisinov D. I., Cheremisinova L. D. *Simulation of discrete control systems with parallelism of behavior*. Informatika [*Informatics*], 2023, vol. 20, no. 4, pp. 24−37 (In Russ.). https://doi.org/10.37661/1816-0301-2023-20-4-24-37

Conflict of interest. The authors declare of no conflict of interest.

Введение. Рост сложности и количества требований к функционалу проектируемых в настоящее время управляющих систем увеличил трудоемкость не только их проектирования, но и верификации. Время, требуемое для верификации, растет даже значительно быстрее, чем функциональная сложность проектов. Тестирование становится одной из наиболее важных и необходимых стадий проверки схемных реализаций или программного обеспечения. По разным оценкам, в настоящее время на этап тестирования и отладки приходится до 60–80 % общих затрат на разработку управляющих систем [1], в отдельных случаях на его долю приходится до 90 % затрат.

На этапе функциональной верификации устанавливается, реализует ли спроектированное устройство желаемое поведение, т. е. работает ли оно согласно установленным в его спецификации требованиям. Самым распространенным подходом к верификации в настоящее время является имитационное моделирование (simulation-based verification), для проведения которого необходима тестовая система. Эта система представляет собой специализированную программную среду, которая решает три основные задачи: генерацию тестовой последовательности, проверку правильности поведения модели тестируемого компонента и оценку полноты тестирования. Тест представляет собой последовательность наборов значений сигналов, подаваемых на вход тестируемой схемы, и наборов значений сигналов, ожидаемых на выходе [2]. Проверка правильности поведения тестируемой схемы выполняется в процессе ее моделирования на тестовой последовательности и сравнения полученных результатов с эталонными, заданными исходной спецификацией.

Качество тестирования напрямую зависит от используемых тестовых последовательностей. В основе традиционно применяемых в практике тестирования методов построения тестовых последовательностей лежит их ручная разработка, случайная и ограниченно случайная (constrained-random) генерация тестов. Хотя, несмотря на простоту, случайные тесты и позволяют быстро обнаруживать значительное число ошибок в проекте, однако при этом нельзя определить, насколько полно они покрывают область работы тестируемого устройства. Более выигрышной представляется [3] стратегия тестирования с использованием знаний о функциональности проектируемого устройства, которая предполагает направленную генерацию тестов (directed tests approach).

Существующие в настоящее время тестовые системы, предназначенные для автоматизации функционального тестирования, позволяют формализовать описание тестируемых устройств, генерировать тестовые последовательности и оценивать правильность поведения аппаратной реализации устройств в процессе их моделирования. Вместе с тем генерируемые тесты не привязаны жестко к этой реализации. В настоящее время стандартом для разработки тестовых систем является универсальная методология верификации UVM (Universal Verification Methodology) [4], которая представляет собой библиотеку языка программирования SystemVerilog с открытым исходным кодом, позволяющую создавать гибкие компоненты тестовых систем. UVM ориентирована на верификацию систем трансформационного типа [5], реализующих классические алгоритмы, которые по заданным исходным данным вычисляют результат за конечную последовательность шагов.

Наиболее развитыми подходами и средствами разработки функциональных тестов в настоящее время являются технологии AVM (Advanced Verification Methodology) [6] компании Mentor Graphics и OVM (Open Verification Methodology, URL: https://verificationacademy.com/archive/ovm-white-paper) компаний Mentor Graphics и Cadence Design Systems на основе языков SystemVerilog или SystemC. При разработке тестовых систем в рамках этих технологий предполагается: объектно-ориентированный подход, моделирование на уровне транзакций (TLM, Transaction-Level Modeling) [7] и направленная генерация псевдослучайных стимулов (constraint-random generation – генерация на основе ограничений). Можно упомянуть также академическую разработку – технологию UniTESK (Unified TEsting and Specification tool Kit) [8] Института системного программирования РАН. В этой работе основной упор делается на автоматизацию процесса генерации тестовых последовательностей на основе автоматных моделей устройств управления. Тестовые воздействия находятся в процессе обхода графа состояний конечного автомата, моделирующего тестируемую систему.

В настоящей работе рассматривается задача верификации систем реактивного типа [5], особенность которых (в отличие от систем трансформационного типа) заключается в непрерывном (и в общем случае бесконечном) обмене сигналами с внешней средой в процессе их функционирования. Наиболее известной моделью реактивных систем является конечный автомат [9, 10], который широко используется для описания протоколов. Однако наряду с традиционно организованными системами, реализующими чисто последовательное поведение, существует ряд систем, в которых выразительных средств аппарата конечных автоматов оказывается недостаточно. Важнейшим свойством таких систем является присущий им параллелизм происходящих в них процессов.

Проблема верификации на основе моделей для устройств с параллелизмом поведения еще недостаточно изучена. Наиболее известными моделями такого типа являются сеть конечных автоматов, система помеченных переходов (labelled transition systems, LTS) [11] и цветные сети Петри [12]. Модели на основе сетей Петри позволяют описать системы с «истинным параллелизмом», в которых некоторые из процессов, происходящих в одном и том же компоненте системы, могут выполняться параллельно и независимо друг от друга.

Большинство известных подходов к генерации тестовых последовательностей для систем, поведение которых задано на языках, в основе которых лежит аппарат сетей Петри, основаны на построении графа достижимых состояний. Тестовые наборы генерируются по этому графу достижимости путем его обхода [13, 14] аналогично тому, как это делается для случая конечных автоматов [15]. Недостатком такого метода является экспоненциальный рост размера пространства возможных состояний системы. В результате граф достижимости сталкивается с проблемой взрыва числа состояний, что негативно влияет на производительность методов тестирования сложных систем.

В настоящей работе рассматривается задача построения тестовой системы для верификации схемной (или программной) реализации устройства управления с параллелизмом поведения. Спецификация проектируемого устройства представляется на языке ПРАЛУ описания параллельных алгоритмов управления [5] и задает систему реактивного типа. Параллельный алгоритм на ПРАЛУ обладает свойством линеаризуемости [16]: результат параллельного выполнения операций алгоритма эквивалентен некоторому последовательному их выполнению. На языке ПРАЛУ можно также описать и поведение объекта, управляемого проектируемым устройством. В этом случае объект управления будет рассматриваться как часть тестового окружения. Тестовые последовательности формируются на основе описанных алгоритмов поведения устройства и объекта управления динамически – в процессе их функционирования. В работе показано, как при разработке тестовой системы для моделей устройств такого типа реализовать объектно-ориентированный подход к моделированию и симуляции на уровне транзакций. Данный подход широко используется в современных средствах моделирования цифровых устройств, так как он позволяет отделить процесс отладки системы и генерации тестовых последовательностей от ее реализации. Предложена модель TLM описаний на языке ПРАЛУ устройств с параллелизмом происходящих в них процессов. Результатом работы системы верификации устройства управления являются тест и программа, моделирующая поведение на этом тесте. Верификация реализации системы состоит в сравнении поведения, отображенного в результате моделирования, и реального эталонного поведения системы, заданного спецификацией на проектирование.

1. Язык описания алгоритма функционирования устройств с параллелизмом поведения. Проблема проектирования устройств управления является одной из важнейших при автоматизации производственных процессов в различных отраслях промышленности. При решении задачи реализации устройств управления приходится иметь дело с параллелизмом, присутствующим в объектах управления. Управление такими объектами заключается в обеспечении согласованной работы взаимодействующих компонентов, работающих параллельно и асинхронно. Параллелизм, присутствующий в объектах управления, отражается в функциональной модели цифровых устройств, управляющих данными объектами. Для цифровых устройств рассматриваемого класса характерно также и то, что управляющие воздействия и сигналы о состоянии объектов управления описываются булевыми переменными и лишь небольшой процент всей информации является числовым.

Для задания спецификации на проектирование устройств с параллелизмом поведения предлагается использовать язык ПРАЛУ [5] описания простых алгоритмов логического управления. Алгоритмы на этом языке представляются в виде причинно-временных зависимостей между событиями, происходящими в технической системе. Событие представляется в виде конъюнкции двоичных переменных алгоритма. Наступление некоторого события в системе происходит в результате выполнения одной из операций алгоритма. Основными операциями языка ПРАЛУ являются операции ожидания и действия.

Операция ожидания – k^{in} сводится к ожиданию момента времени, когда конъюнкция k^{in} примет значение 1. Операция действия $\rightarrow k^{out}$ выполняется путем присвоения переменным, образующим конъюнкцию k^{out}, значений, обращающих ее в 1. Окончание операции действия приводит к изменению состояния системы в пространстве переменных и является причиной возникновения некоторого нового события. Операции ожидания и действия могут интерпретироваться как опрос состояний датчиков объекта управления и выдача команд на исполнительную и сигнальную аппаратуру.

В одной из интерпретаций операций действия в языке предполагается, что все внутренние (если такие имеются в описании) и выходные (или управляющие) переменные сохраняют свои значения до тех пор, пока их не изменит какая-либо из операций действия.

Для задания порядка выполнения операций в алгоритмах на ПРАЛУ используется механизм переходов, аналогичный сети Петри. Описание перехода описывается цепочкой операций. Она начинается перечнем меток позиций, из которых запускается переход, состоит из последовательности операций языка и заканчивается перечнем меток позиций, в которые происходит переход после выполнения всех операций. Алгоритм управления на ПРАЛУ представляется неупорядоченной совокупностью цепочек вида μ_i: $l_i \rightarrow \nu_i$, где через l_i обозначен некоторый линейный алгоритм, составленный из операций языка; μ_i и ν_i – начальная и конечная метки, которыми служат непустые подмножества множества позиций (задаваемых натуральными числами) $M = \{1, 2, \ldots, n\}$. Порядок выполнения цепочек алгоритма управления в процессе его реализации определяется множеством N запуска [5], его текущие значения $N_t \subset M$. Среди предложений алгоритма выделяется одно – начальное, его метка заносится в множество N_0 перед началом реализации алгоритма.

В процессе реализации алгоритма управления цепочки запускаются независимо друг от друга. Если в некоторый момент времени для некоторой цепочки μ_i: $l_i \rightarrow \nu_i$ выполняется условие $\mu_i \subseteq N_t$ и реализуется событие k_i^{in}, с ожидания которого начинается цепочка l_i, то она запускается. После завершения цепочки состояние N_t заменяется на $N_{t+1} = (N_t \setminus \mu_i) \cup \nu_i$. Синтаксически параллельность алгоритма отражается в наличии начальных $|\mu_i| > 1$ меток цепочки (когда несколько процессов сливаются, порождая один процесс) и конечных $|\nu_i| > 1$ меток (когда запускается одновременно несколько независимых процессов). В описании алгоритма могут быть также цепочки μ_i: $l_i \rightarrow \nu_i$ и μ_j: $l_j \rightarrow \nu_j$ с одинаковыми начальными метками $\mu_i = \mu_j$ (случай $\mu_i \cap \mu_j \neq \varnothing$, но $\mu_i \neq \mu_j$ не допускается). Такие цепочки являются связанными и не могут запускаться одновременно: их запуск зависит не только от текущей разметки сети (задаваемой множеством запуска N_t), но и от некоторых внешних событий. По определению связанные цепочки должны начинаться с операций ожидания несовместных событий: соответствующие им конъюнкции k_i^{in} и k_j^{in} должны быть попарно ортогональны.

Алгоритм на ПРАЛУ описывает замкнутую систему, если все события в пространстве переменных вызываются реализацией операций действия самого алгоритма. В случае незамкнутых систем можно выделить подмножество внешних (входных или условных) переменных алгоритма, значения которых задаются внешней средой.

Ниже приведен пример алгоритма управления на языке ПРАЛУ, который описывает незамкнутую систему. Значения входных переменных множества $\{x, y, z\}$, определяющих поведение системы, задаются извне:

1: $-y \rightarrow a\, c - \bar{y} \rightarrow b \rightarrow 2.3$
2: $-x \rightarrow \bar{a}\ \bar{b} \rightarrow 4.5$
3: $-\bar{x}y \rightarrow 6$

3: $-xy \rightarrow \overline{c} - \overline{x} \rightarrow c \rightarrow 3$
4: $- z \rightarrow a \rightarrow 7$
5: $- \overline{x}z \rightarrow b \rightarrow 8$
6.7.8: $\rightarrow \overline{a}\ \overline{b}\ \overline{c} \rightarrow$.

Кроме упомянутых операций ожидания и действия в расширенной версии языка – АЛУ – доступны их расширения, реализующие некоторые операции ограниченной арифметики, которые связаны со счетом событий и временными задержками. Для обеспечения реакции устройства на особые события введены также операции гашения, которые при реализации описания алгоритма управления дают возможность досрочно завершить некоторые или все параллельно выполняющиеся цепочки.

Язык алгоритмов управления обеспечивает возможность иерархического описания поведения системы, которое отражает структуру многокомпонентной системы и организацию взаимодействия отдельных ее частей. В частности, на этом же языке можно описать и поведение объекта, управляемого проектируемым устройством. Такое описание может рассматриваться как часть тестового окружения в процессе моделирования алгоритма функционирования устройства управления. Совместное описание устройства и объекта управления позволяет увеличить замкнутость моделируемой системы, сокращая число переменных, значения которых должны определяться извне.

2. Моделирование цифровых систем на уровне транзакций. В современных системах верификации цифровая система представляется в виде компонентов, поведение которых задается как набор параллельных взаимодействующих процессов. Взаимодействие процессов представляется как коммуникация, в которой актами коммуникации служат транзакции через общие структуры данных. Моделирование на уровне транзакций (TLM) представляет собой подход к моделированию обмена данными в цифровых системах, при котором организация связи между функциональными компонентами системы отделена от их реализации.

На уровне транзакций упор в большей степени делается на функциональность обмена данными между процессами и в меньшей – на их фактическую реализацию. Все выполняемые при этом операции изменения состояний, передачи данных и вычислений являются транзакциями. Каждая транзакция описывает преобразование данных, общих для взаимодействующих процессов. Транзакции создаются динамически в процессе моделирования в соответствии с заданными условиями. Произведенные ими изменения состояний данных становятся видимыми для остальных процессов после их завершения.

Модель уровня транзакций описывает компоненту системы набором процессов, которые определяют ее поведение и взаимодействуют одновременно. На уровне описания аппаратуры при моделировании для синхронизации процессов, как правило, используется потактовая смена значений сигналов системы. На уровне TLM генератор синхросигналов не применяется, и при моделировании процессы синхронизируются в моменты обмена сигналами между процессами, т. е. после выполнения транзакций.

Модели TLM используются в современных средствах верификации проектов аппаратуры. Такой подход позволяет отделить процесс отладки системы и генерации тестовых последовательностей от ее аппаратной реализации. Наиболее популярным языком моделирования уровня TLM является SystemC (стандарт IEEE 1666), который представляет собой расширение языка C++. Исполняемая программа, получаемая в результате компиляции модели на языке SystemC, реализует симулятор с интегрированными средствами управления имитацией. Параллелизм в SystemC имеет семантику чередования операций последовательных процессов. Параллельные процессы языка SystemC вводятся как потоки (threads), которые планируются для последовательного выполнения планировщиком SystemC на основе невытесняющего (non-preemptive) мультипрограммирования. Потоками являются последовательные процессы, имеющие общую память.

Модель многопоточности в SystemC обладает существенным недетерминизмом, и процессы языка SystemC имеют специальную организацию для устранения этого недетерминизма. Атомарные операции процессов, задаваемые транзакциями, линеаризуются в процессе их выполне-

ния таким образом, чтобы обеспечивать детерминизм результата выполнения операций над общими данными несколькими параллельными процессами. Линеаризуемость операций можно понимать и в том смысле, что результат параллельного выполнения любых операций эквивалентен их некоторому последовательному выполнению [16]. Выполнение линеаризуемой операции с точки зрения любого другого потока является мгновенным: операция либо не начата, либо завершена.

Синхронизация процессов модели TLM на уровне транзакций осуществляется барьерным механизмом [17]. Барьерами являются точки исходного кода, в которых каждый процесс должен приостановиться и подождать достижения барьера всеми параллельными процессами выполняемой группы потока. В модели TLM на языке SystemC точки барьера задаются вызовами функции wait (.). Изменения значений, запланированные в процессах группы, в общей структуре данных происходят мгновенно после достижения барьера всеми процессами.

В работе [5] было предложено характеризовать цифровые устройства по типу алгоритмического описания. Устройства, моделью которых являются классические алгоритмы (алгоритмы планирования), относятся к трансформационному типу. Их цель – вычисление некоторого результата по исходным данным посредством конечной последовательности шагов. Примерами таких систем являются процессоры, компиляторы языков программирования, веб-серверы. Модель TLM предложена для цифровых устройств трансформационного типа.

Функционирование реактивных систем заключается в непрерывном взаимодействии с окружающей средой: опросом и изменением ее состояния. Поведение реактивной системы задается алгоритмом управления (протоколом взаимодействия). Примерами таких устройств являются контроллеры периферийных устройств компьютера, подключаемые к общей шине, встроенные системы, устройства управления оборудованием. В последнее время термин «реактивная система» стал употребляться и для обозначения программных систем, в которых асинхронно обрабатываются потоки данных, объем которых не предопределен [18].

Алгоритмы управления реактивными системами не являются алгоритмами в смысле классической теории алгоритмов, тем не менее показано, что для формализации и моделирования таких алгоритмов также можно использовать подход TLM.

3. Модель TLM реактивных систем с параллелизмом поведения на языке ПРАЛУ. Алгоритмы на языке ПРАЛУ допускают интерпретацию моделью TLM. Для построения модели TLM алгоритма на языке ПРАЛУ требуется уточнить семантику операций ожидания и действия, определить понятие операции, транзакции, потока, барьера, а также уточнить определение параллельности выполнения операций. В работе используется модель параллелизма операций типа «чередование», в которой одновременность понимается как возможность упорядочивать операции произвольным образом. Суть принятого уточнения порядка выполнения операций состоит в доопределении частичного порядка на множестве операций, задаваемого исходным параллельным алгоритмом, до линейного порядка. В такой интерпретации алгоритм на ПРАЛУ обладает свойством линеаризуемости, т. е. результат параллельного выполнения операций алгоритма эквивалентен некоторому последовательному выполнению атомарных операций.

Ключевые моменты предлагаемой модели TLM описания алгоритма управления на языке ПРАЛУ состоят в следующем:

1. Структурой данных является вектор значений переменных алгоритма: внешних – входных (условных) и внутренних – выходных. Каждая компонента вектора соответствует одной из переменных и задает пару значений этой переменной: текущее и планируемое. Такое разделение значений одной и той же переменной вызвано необходимостью исключить в процессе вычислений изменение значений переменных до достижения барьера. Доступ к компонентам вектора данных осуществляется посредством операции установки значений переменных алгоритма, определяющей планируемые значения, и операции проверки значений условных переменных, читающей их текущие значения.

2. Транзакции представляются операциями ожидания и действия алгоритма, которые описывают события взаимодействия [19]. Эти операции рассматриваются в виде композиций некоторых элементарных операций, выполняемых последовательно. Реализация операции ожидания – k^{in} состоит в выполнении операции приостановки процесса и проверки текущих значений

переменных, указанных в конъюнкции k^{in}. Реализация операции действия $\rightarrow k^{out}$ заключается в установке планируемых значений переменных, указанных в конъюнкции k^{out}.

3. Цепочки операций языка ПРАЛУ интерпретируются как процессы модели TLM. Соответственно, последовательный процесс представляет собой совокупность последовательно выполняемых операций ожидания и действия алгоритма. Вводится понятие ветви, которая задает последовательный процесс, начинающийся с некоторой операции. Ветвь является динамическим объектом, который порождается операцией образования и уничтожается операцией ее прекращения. После образования ветвь может выполняться, а в процессе выполнения – приостанавливаться. Образование ветви заключается в занесении первой ее операции в очередь ветвей, готовых для выполнения (ОГ). Операция прекращения ветви состоит в удалении ее из ОГ. Операция приостановки ветви является аналогом функции wait(.) в языке SystemC, при ее реализации ветвь переносится из ОГ в очередь ветвей, ждущих выполнения (ОЖ). Введенные приведенным способом ветви аналогичны потокам в модели TLM на языке SystemC.

4. При моделировании алгоритма управления из ОГ последовательно извлекаются ветви и выполняются до приостановки. Выполнение ветви *G* приостанавливается, когда ее начальный фрагмент – k^{in} не может выполниться на множестве текущих значений переменных и тогда ветвь переносится в ОЖ или же он выполнен и тогда в ОЖ вносится ветвь, начинающаяся с операции, которая должна выполняться в ветви *G* следующей.

5. Синхронизация параллельно выполняемых ветвей осуществляется с помощью барьерного механизма. Достижение барьера фиксирует такты моделирования и изменения значений переменных. Точки барьера задаются операциями приостановки выполнения всех ветвей. Структура данных барьера синхронизации представлена в памяти очередями ветвей ОГ и ОЖ. Барьер достигнут, когда ОГ становится пустой. При достижении барьера: элементы из ОЖ переносятся в ОГ (ОЖ становится пустой); вводятся (если система незамкнута) новые значения входных (внешних) переменных в качестве планируемых; накопленные планируемые значения пересылаются в текущие для каждой компоненты вектора переменных; выгружаются значения выходных переменных и запускается выполнение первой ветви из ОГ.

Барьерная синхронизация считается довольно затратной в смысле памяти и времени выполнения. Синхронизация процессов занимает больше времени, чем вычисления, особенно в распределенных вычислениях. В модели TLM на языке ПРАЛУ точки барьера в явном виде не указываются (в отличие от модели TLM на языке SystemC), барьер формируется автоматически в процессе моделирования. Операции образования, прекращения и приостановки ветвей относятся к накладным расходам на организацию вычислений. Однако в модели TLM описаний на языке ПРАЛУ каждая из операций барьерного механизма может быть реализована в современных процессорах одной командой, в том числе и наиболее сложная из них – приостановка, которая является аналогом функции wait(.) в SystemC. Реализовать отдельный планировщик не требуется.

4. Реализация симулятора моделей TLM на языке ПРАЛУ. Преобразование описания алгоритма на языке ПРАЛУ в модель TLM осуществляется путем представления операций языка в виде композиций элементарных операций, которые выполняются строго последовательно. Набор этих операций (табл. 1) составляет базис алгоритмического разложения параллельных алгоритмов на языке ПРАЛУ.

Начальными состояниями ветвей TLM модели являются цепочки алгоритма на языке ПРАЛУ. Связанные цепочки (с одинаковыми метками) порождают разные ветви, и операция их запуска заменяется операцией запуска соответствующих ветвей. Например, в приведенном в разд. 1 алгоритме две связанные цепочки (третья и четвертая) с меткой $\mu = 3$ порождают две связанные ветви H3 и H4, и операция перехода $\rightarrow 3$ заменяется запуском этих двух ветвей. Однако выполняться в текущем такте может только одна из ветвей. Для правильного запуска группы связанных ветвей вводится переменная маски *w*. Значение $w = 1$ фиксирует факт выполнения условия запуска (наступление ожидаемого события) для одной из ветвей. Значение этой переменной опрашивается перед выполнением каждой из ветвей с тем, чтобы исключить выполнение остальных цепочек, если $w = 1$ (используется операция «Прекращение процесса по условию»).

Таблица 1
Базовые элементарные операции
Table 1
Basic elementary operations

Операция *Operation*	Функция *Function*	Описание *Description*
П	Приостановка ветви	Текущая ветвь переносится из ОГ в ОЖ, начиная с операции, следующей за «П», и выбирается следующая ветвь из ОГ
?	Переход и запуск ветвей ?(H1, H2,…)	Ветви H1, H2,… вносятся в ОГ
A	Прекращение процесса по условию Aw	Прекращение реализации ветви, если $w = 1$
К	Остановка алгоритма	Конец работы алгоритма
S	Установка выходных переменных Sk^{out}	Устанавливаются планируемые значения переменных (в соответствии с конъюнкцией k^{out})
O	Проверка условных переменных Ok^{in}	Проверяется выполнение условия $k^{in} = 1$ на множестве текущих значений переменных
P	Установка переменной синхронизации Pr	Устанавливается единичное значение переменной синхронизации r
R	Сброс переменной синхронизации Rr	Устанавливается нулевое значение переменной синхронизации r
M	Установка маски Mw	Устанавливается единичное значение маски w для связанных цепочек
C	Сброс маски Cw	Устанавливается нулевое значение маски w для связанных цепочек

Аналогично для синхронизации слияния k параллельных цепочек алгоритма на ПРАЛУ используются k переменных синхронизации. Значения переменных синхронизации опрашиваются перед выполнением ветви, соответствующей слиянию этих k цепочек. Например, последняя цепочка алгоритма начинается меткой $\mu = 6.7.8$ и может выполняться, когда множество запуска $N_t \supseteq \{6, 7, 8\}$. Для фиксации данного факта вводятся три переменные синхронизации r_1, r_2 и r_3, которым присваивается значение 1 соответственно при переходах $\rightarrow 6$, $\rightarrow 7$ и $\rightarrow 8$. При запуске ветви H7, соответствующей последней цепочке алгоритма, производится опрос значений этих переменных посредством использования операции ожидания события $r_1 r_2 r_3 = 1$.

Операции действия $\rightarrow y_1 y_2 \ldots y_l$ соответствует команда установки значений (планируемых) переменных промежуточного языка: S$y_1, y_2, \ldots, y_l$. Операция ожидания – $x_1 x_2 \ldots x_l$ транслируется во фрагмент, состоящий из двух операций приостановки и проверки значений (текущих) переменных: П O$x_1, x_2, \ldots, x_l$. Операции ожидания – $x_1 x_2 \ldots x_l$, открывающей одну из связанных цепочек, соответствует фрагмент П Aw : O$x_1, x_2, \ldots, x_l$: Mw. Реализация операции перехода $\rightarrow p.q. \ldots r$ состоит в запуске всех ветвей с начальными метками $p, q, \ldots, r$.

Приведенный выше пример описания алгоритма управления на языке ПРАЛУ переводится в следующее описание на промежуточном языке:

H1 ПOy : Sa,c : ПО $\bar{y}$: Sb : ?2,3,4
H2 ПOx : S $\bar{a}$, $\bar{b}$: ?5,6
H3 ПAw_1 : Oy, $\bar{x}$: Mw_1 Cw_2 : Pr_1 : ?7
H4 ПAw_1 : Oy,x : Mw_1 S $\bar{c}$: ПО $\bar{x}$: Sc : ППCw_1 : ?4,3
H5 ПOz : Sa : Cw_2 : Pr_2 : ?7
H6 ПО $\bar{x},z$: Sb : Cw_2 : Pr_3 : ?7
H7 ПAw_2 : Or_1,r_2,r_3 : Mw_2 : Rr_1,r_2,r_3 : S $\bar{a}$, $\bar{b}$, $\bar{c}$: К

5. Моделирование описаний систем управления на уровне модели TLM на языке ПРАЛУ. Симуляторы электронных устройств интегрируют редактор описаний функционирования, механизм моделирования (процессор, представляющий собственно симулятор) и средство экранного отображения формы сигналов. В предлагаемой реализации симулятора описаний управляющих устройств механизм моделирования отделен от средства отображения формы сигнала, что позволяет интегрировать его с разными средствами отображения результатов модели-

рования. Программа процессора симуляции сохраняет результаты проведенного моделирования в специальной базе данных, форма представления которых согласована с формой представления данных средства экранного отображения сигналов. Это позволяет сохранять и анализировать результаты моделирования, а также сопоставлять результаты при использовании разных входных данных.

Основополагающим моментом при моделировании алгоритмов на языке ПРАЛУ является соглашение о длительности выполнения операций языка, в частности это касается операций действия. Данное соглашение существенно влияет на степень соответствия изменений сигналов, производимых симулятором и появляющихся на выходах программной или схемной реализации алгоритма. Реализация (так же, как и моделирование) алгоритмов выполняется в соответствии с некоторым предположением о длительности выполнения операций языка. Наиболее естественно принятое предположение об одинаковой длительности выполнения всех операций (и, в частности, операций действия). Один из путей повышения быстродействия вычислений состоит в предположении о нулевой длительности операций. В этом случае вычисления по одной ветви выполняются до тех пор, пока для их продолжения не потребуется изменение состояний условных переменных. Такое предположение означает, что приостановки выполнения ветвей будут происходить только при выполнении операций ожидания. Для схемной реализации алгоритмов управления более естественно предположение об одинаковой, но не нулевой длительности выполнения операций действия. В этом случае приостановка ветви будет производиться после выполнения операции действия, а не только тогда, когда операция ожидания не может выполниться на множестве текущих значений переменных.

Достижение барьера при моделировании алгоритма фиксирует такты работы устройства, а полученные изменения значений выходных переменных (отмеченные в векторе планируемых значений) соответствуют изменениям значений сигналов на выходах схемной реализации алгоритма управления при подаче на ее входы значений сигналов, соответствующих значениям в векторе текущих значений.

Внешнее поведение процессора, симулирующего алгоритм на языке ПРАЛУ, представляет собой циклы, называемые циклами сканирования. Каждый цикл состоит из ввода новых значений условных переменных – опроса датчиков системы; выполнения программы до достижения барьера и вывода текущих значений выходных переменных – выдачи команд на исполнительные механизмы (рис. 1). Длительность цикла сканирования непостоянна и зависит от выполняемого алгоритма.

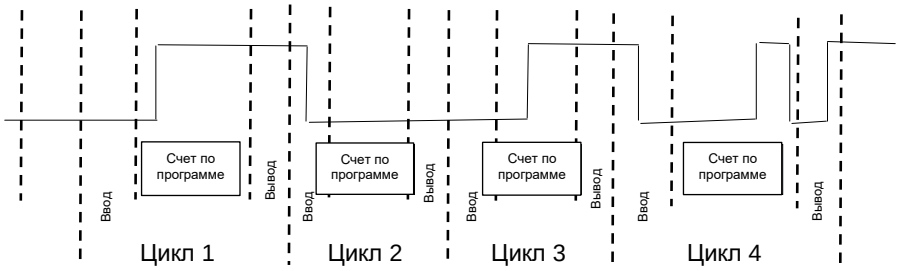

Рис. 1. Циклы сканирования при симуляции алгоритма на языке ПРАЛУ

Fig. 1. Scan cycles when simulating the algorithm on the PRALU language

Продемонстрируем процесс моделирования работы приведенного в разд. 1 алгоритма на промежуточном языке. В табл. 2 показаны девять тактов моделирования этого алгоритма. Каждая из подтаблиц соответствует одному такту, порождаемому достижением барьера. Столбцы каждой из подтаблиц задают: выбираемые из ОГ ветви G_j в форме $H_i.O_{il}$ (здесь O_{il} – начальная операция ветви H_i), состояния очередей ОГ и ОЖ готовых и ждущих ветвей, векторы масок, текущих $\mathbf{t_i}$ и планируемых $\mathbf{p_i}$ значений переменных алгоритма. В первой строке каждой подтаблицы показаны состояния $ОГ_i$, $ОЖ_i$, векторов масок, текущих $\mathbf{t_i}$ и планируемых $\mathbf{p_i}$ значений переменных в начале соответствующего такта (после ввода значений внешних переменных). При переходе от i-го такта к (i+1)-му $ОЖ_i \Rightarrow ОГ_{i+1}$, $\mathbf{p_i} \Rightarrow \mathbf{t}_{i+1} = \mathbf{p}_{i+1}$. Вектор масок состоит из двух частей: первая часть задает значения масок w_i, вторая – значения переменных синхронизации r_i.

Векторы $\mathbf{t}_i$ и $\mathbf{p}_i$ отражают изменения всех переменных: сначала входных (условных), затем внутренних (если они есть) и выходных (управляющих). Производимые при выполнении операций изменения значений переменных выделяются жирным шрифтом. Положим, что перед началом моделирования все переменные x, y, z, a, b, c имеют нулевые значения: $\mathbf{t}_1 = \mathbf{p_1} = 0\,0\,0\,0\,0\,0$, переменные масок и синхронизации тоже: 0 0 000. При моделировании использовался следующий тестбенч, задающий изменение внешних (входных) переменных в начале тактов моделирования: **0 0 0**, 0 **1** 0, 0 **0** 0, **1 1** 0, 1 1 **1**, **0** 1 1, 0 1 **0**, 0 **0 1**, 0 0 **0**. Выходные реакции устройства управления отражают изменение значений управляющих переменных в конце каждого такта: 0 0 0, **1** 0 **1**, 1 **1** 1, **0 0 0**, **1** 0 0, 1 **1 1**, 1 1 1, 1 1 1, **0 0 0**.

Таблица 2
Такты моделирования алгоритма управления

Table 2
Cycles of the control algorithm simulation

Номер такта *Measure number*	G_j	Очередь готовых ветвей *The queue of ready branches*	Очередь ждущих ветвей *The queue of waiting branches*	Маски *Masks*	Текущие значения $\mathbf{t}_i$ *Current values* $\mathbf{t}_i$	Планируемые значения $\mathbf{p}_i$ *Planned values* $\mathbf{p}_i$
1		H1	∅	0 0 000	000 000	000 000
	H1	∅	**H1.O*y***			
2		H1.O*y*	∅	0 0 000	0**1** 0 000	0 1 0 000
	H1.O*y*	∅	**H1.O $\bar{y}$**			0 1 0 **1** 0 **1**
3		H1.O $\bar{y}$	∅	0 0 000	0**0** 0 101	000 101
	H1.O $\bar{y}$	**H2, H3, H4**∅	∅			000 1**1** 1
	H2	H3, H4	**H2.O*x***			
	H3	H4	H2.O*x*, **H3.Aw_1**			
	H4	∅	H2.O*x*, H3.Aw_1, **H4.Aw_1**			
4		H2.O*x*, H3.Aw_1, H4.Aw_1	∅	0 0 000	**1 1** 0 111	1 1 0 111
	H2.O*x*	H3.Aw_1, H4.Aw_1, **H5,H6**	∅			1 1 0 **0 0** 1
	H3.Aw_1	H4.Aw_1, H5, H6	**H3.Aw_1**			
	H4.Aw_1	H5,H6	H3.Aw_1, **H4.O $\bar{x}$**	**1** 0 000		1 1 0 0 0 **0**
	H5	H6	H3.Aw_1, H4.O $\bar{x}$, **H5.Oz**			1 1 0 000
	H6	∅	H3.Aw_1, H4.O $\bar{x}$, H5.Oz, **H6.O $\bar{x}z$**			1 1 0 000
5		H3.Aw_1, H4.O $\bar{x}$, H5.Oz, H6.O $\bar{x}z$	∅	1 0 000	1 1**1** 000	1 1 1 000
	H3.Aw_1	H4.O $\bar{x}$, H5.Oz, H6.O $\bar{x}z$	∅			
	H4.O $\bar{x}$	H5.Oz, H6.O $\bar{x}z$	**H4.O $\bar{x}$**			1 1 1 000
	H5.Oz	H6.O $\bar{x}z$	H4.O $\bar{x}$, **H7**	1**0** 0 1 0		1 1 1 **1** 0 0
	H6.O $\bar{x}z$	∅	H4.O $\bar{x}$, H7, **H6.O $\bar{x}z$**			
6		H4.O $\bar{x}$, H7, H6.O $\bar{x}z$	∅	1 0 0 1 0	**0** 1 1 1 0 0	0 1 1 1 0 0
	H4.O $\bar{x}$	H7, H6.O $\bar{x}z$	**H4.П**			0 1 1 1 0 **1**
	H7	H6.O $\bar{x}z$	H4.П, **H7**			
	H6.O $\bar{x}z$	∅	H4.П, H7	1**0** 0 1 1		0 1 1 1 **1** 1
7		H4.П, H7	∅	1 0 0 1 1	0 1**0** 1 1 1	0 1 0 1 1 1
	H4.П	H7,H4, H3	∅	**0** 0 0 1 1		
	H7	H4,H3	∅			
	H4	H3	**H7, H4.O*y,x***			
	H3	∅	H7, H4.O*y,x*	**0 0 1** 1 1		
8		H7,H4. O*y,x*	∅		0**0 1** 1 1 1	0 0 1 1 1 1
	H7	H4.O*y,x*	**H7.Aw_2**			
	H4.O*y,x*	∅	H7.Aw_2			
9		H7.Aw_2	∅	0 0 1 1 1	0 0**0** 1 1 1	0 0 0 1 1 1
	H7.Aw_2	∅	∅	0 0 0 0 0		0 0 0 **0 0 0**
				0 0 0 0 0	**0 0 0 0 0 0**	

Преобразование представления описания алгоритма функционирования устройства на промежуточном языке в программу симулятора выполняется однопроходным текстовым преобразователем, конвертирующим операторы промежуточного языка в вызовы процедур в синтаксисе языка С. Эффективность построенной таким образом программы можно оценить на примере реализации одной из самых трудоемких операций промежуточного языка – операции приостановки. Работа, выполняемая этой операцией, заключается в запоминании адреса следующего вызова процедуры в стеке данных и переходе к выполнению процедуры, адрес которой извлечен из стека возвратов. Компилятор С строит для осуществления этих действий фрагмент кода, состоящий всего из двух машинных команд.

Программа симулятора алгоритмов на ПРАЛУ сохраняет результат моделирования алгоритма в файловом формате VCD (Value Change Dump – дамп изменения значений) [20]. Формат VCD является частью стандарта IEEE для языка описания оборудования на языке Verilog. Историю изменения сигналов, хранящихся в базе в текстовом формате VCD, можно впоследствии просмотреть с помощью инструментов просмотра формы сигналов. Программа просмотра формы сигнала позволяет разработчику СБИС видеть изменение сигналов во времени и анализировать взаимосвязь выходных сигналов модели с входными сигналами. Имеющиеся средства просмотра формы сигналов (например, свободно доступная программа GTKWave, URL: https://gtkwave.sourceforge.net/), входящие в состав промышленных и свободно доступных САПР, позволяют не только отображать графики изменения сигналов, но и изменять масштаб временной последовательности, а также сравнивать и вычислять разницу между значениями сигнала, соответствующими двум точкам, отмеченным курсором на графике.

Заключение. Поведение встроенного устройства управления существенно зависит от объекта, которым оно управляет, и среды, в которой оно работает. Моделирование устройства управления с целью верификации должно производиться на области его запланированного функционирования. С использованием имитационного моделирования верификация схемной реализации устройства управления может быть выполнена двумя путями: динамически в процессе отладки описания алгоритма его функционирования и на тестовой последовательности, полученной в процессе моделирования этого описания. В статье предложены модель TLM для моделирования описаний алгоритмов управления на языке ПРАЛУ, а также метод преобразования описания алгоритма на языке ПРАЛУ в программу на промежуточном языке, которая выполняется строго последовательно. Разработаны трансляторы этой программы на языки Verilog и С, результаты компиляции которых на машинный язык представляют собой программы имитационного моделирования заданной системы управления. Исходными данными для программ имитационного моделирования служат тестовые последовательности.

Вклад авторов. *Л. Д. Черемисинова* предложила модель TLM реактивных систем с параллелизмом поведения, проанализировала полученные результаты и подготовила текст статьи. *Д. И. Черемисинов* разработал и программно реализовал метод преобразования описания алгоритма на языке ПРАЛУ в программу на промежуточном языке.

Список использованных источников

1. Слинкин, Д. И. Анализ современных методов тестирования и верификации проектов сверхбольших интегральных схем / Д. И. Слинкин // Программные продукты и системы. – 2017. – № 3(30). – С. 401–407.

2. Камкин, А. Обзор современных технологий имитационной верификации аппаратуры / А. Камкин, М. Чупилко // Программирование. – 2011. – № 3. – С. 42–49.

3. Kaner, C. What Is a Good Test Case? Software Testing Analysis & Review Conference (STAR) East / C. Kaner. – Mode of access: https://profinit.eu/wp-content/uploads/2016/03/WhatIsGoodTestcase.pdf. – Date of access: 10.02.2023.

4. The Open Source VHDL Verification Methodology. User’s Guide Rev. 1.2 / ed. J. Lewis. – Mode of access: https://www.doulos.com/knowhow/vhdl/the-open-source-vhdl-verification-methodology-osvvm/. – Date of access: 02.09.2023.

5. Закревский, А. Д. Параллельные алгоритмы логического управления / А. Д. Закревский. – Минск : Ин-т техн. кибернетики НАН Беларуси, 1999. – 202 с.

6. Advanced Verification Methodology Cookbook / A. Rose [et al.] ; ed. M. Glasser. – Mentor Graphics Corporation, 2008. – 338 p.

7. Cai, L. Transaction level modeling: an overview / L. Cai, D. Gajski // First IEEE/ACM/IFIP Intern. Conf. on Hardware/ Software Codesign and Systems Synthesis, Newport Beach, CA, USA, 1–3 Oct. 2003. – Newport Beach, 2003. – P. 19–24.

8. Подход UniTesK к разработке тестов / В. В. Кулямин [и др.] // Программирование. – 2003. – № 6(29). – С. 25–43.

9. Lee, D. Principles and methods of testing finite state machine – a survey / D. Lee, M. Yannakakis // Proceedings of the IEEE. – 1996. – Vol. 84, no. 8. – P. 1090–1123.

10. Kanso, B. Compositional testing for FSM-based models / B. Kanso, O. Chebaro // Intern. J. of Software Engineering & Applications (IJSEA). – 2014. – Vol. 5, no. 3. – P. 9–23.

11. Ponce de Leon, H. Model-based testing for concurrent systems with labeled event structures / H. Ponce de Leon, S. H. Delphine Longuet // Software Testing, Verification & Reliability. – 2014. – Vol. 24, no. 7. – P. 558–590.

12. Питерсон, Дж. Теория сетей Петри и моделирование систем : пер. с англ. / Дж. Питерсон. – М. : Мир, 1984. – 264 с.

13. Zhu, H. A methodology of testing high-level Petri nets / H. Zhu, X. D. He // Information and Software Technology. – 2002. – Vol. 44. – P. 473–489.

14. I/O conformance test generation with colored Petri nets / J. Liu [et al.] // Applied Mathematics and Information Sciences. – 2014. – Vol. 8, no. 6. – P. 2695–2704.

15. Бурдонов, И. Б. Неизбыточные алгоритмы обхода ориентированных графов. Детерминированный случай / И. Б. Бурдонов, А. С. Косачев, В. В. Кулямин // Программирование. – 2003. – № 5. – С. 11–30.

16. Herlihy, M. P. Linearizability: a correctness condition for concurrent objects / M. P. Herlihy, J. M. Wing // ACM Transactions on Programming Languages and Systems. – 1990. – Vol. 12, no. 3. – P. 463–492.

17. Solihin, Y. Fundamentals of Parallel Multicore Architecture / Y. Solihin. – CRC Press, 2015. – 494 p.

18. Halbwachs, N. Synchronous Programming of Reactive Systems / N. Halbwachs. – Springer-Verlag, 2010. – 192 p.

19. Черемисинов, Д. И. Проектирование и анализ параллелизма в процессах и программах / Д. И. Черемисинов. – Минск : Беларуская навука, 2011. – 300 с.

20. Dtrgeron, J. Writing Test Benches & Functional Verification of HDL Models 2gd Edition / J. Dtrgeron. – Springer, 1993. – 508 p.

References

1. Slinkin D. I. *Analysis of modern methods for testing and verification of vlsi projects software products and systems*. Programmnyye produkty i sistemy [*Software Products and Systems*], 2017, no. 3(30), pp. 401–407 (In Russ.).

2. Kamkin A., Chupilko M. *Overview of modern technologies for simulation verification of equipment*. Programmirovaniye [*Programming*], 2011, no. 3, pp. 42–49 (In Russ.).

3. Kaner C. What is a good test case? Software Testing Analysis & Review Conference (STAR) East. Available at: https://profinit.eu/wp-content/uploads/2016/03/WhatIsGoodTestcase.pdf (accessed 10.02.2023).

4. Lewis J. (ed.). *Open Source VHDL Verification Methodology. User's Guide Rev. 1.2*. Available at: https://www.doulos.com/knowhow/vhdl/the-open-source-vhdl-verification-methodology-osvvm/ (accessed 02.09.2023).

5. Zakrevskiy A. D. Parallel'nyye algoritmy logicheskogo upravleniya. *Parallel Logic Control Algorithms*. Minsk, Institut tehnicheskoj kibernetiki Nacional'noj akademii nauk Belarusi, 1999, 202 p. (In Russ.).

6. Rose A., Fitzpatrick T., Rich D., Foster H. *Advanced Verification Methodology Cookbook*. In M. Glasser (ed.). Mentor Graphics Corporation, 2008, 338 p.

7. Cai L., Gajski D. Transaction level modeling: an overview. *First IEEE/ACM/IFIP International Conference on Hardware/ Software Codesign and Systems Synthesis, Newport Beach, CA, USA, 1–3 October 2003*. Newport Beach, 2003, pp. 19–24.

8. Kulyamin V. V., Petrenko A. K., Kosachev A. S., Burdonov I. B. *UniTesK approach to test development*. Programmirovaniye [*Programming*], 2003, no. 6(29), pp. 25–43 (In Russ.).

9. Lee D., Yannakakis M. Principles and methods of testing finite state machine – a survey. *Proceedings of the IEEE*, 1996, vol. 84, no. 8, pp. 1090–1123.

10. Kanso B., Chebaro O. Compositional testing for FSM-based models. *International Journal of Software Engineering & Applications (IJSEA)*, 2014, vol. 5, no. 3, pp. 9–23.

11. Ponce de Leon H. Delphine Longuet S. H. Model-based testing for concurrent systems with labeled event structures. *Software Testing, Verification & Reliability*, 2014, vol. 24, no. 7, pp. 558–590.

12. Peterson J. L. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Englewood Cliffs, 1981, 290 p.

13. Zhu H., He X. D. A methodology of testing high-level Petri nets. *Information and Software Technology*, 2002, vol. 44, pp. 473–489.

14. Liu J., Ye1 X., Zhou J., Song X. I/O conformance test generation with colored Petri nets. *Applied Mathematics and Information Sciences*, 2014, vol. 8, no. 6, pp. 2695–2704.

15. Burdonov I. B., Kosachev A. S., Kulyamin V. V. *Irredundant algorithms for traversing directed graphs. Deterministic case*. Programmirovaniye [*Programming*], 2003, no. 5, pp. 11–30 (In Russ.).

16. Herlihy M. P., Wing J. M. Linearizability: a correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems*, 1990, vol. 12, no. 3, pp. 463–492.

17. Solihin Y. *Fundamentals of Parallel Multicore Architecture*. CRC Press, 2015, 494 p.

18. Halbwachs N. *Synchronous Programming of Reactive Systems*. Springer-Verlag, 2010, 192 p.

19. Cheremisinov D. I. Proyektirovaniye i analiz parallelizma v protsessakh i programmakh. *Design and the Analysis of Parallelism in Processes and Programs.* Minsk, Belaruskaja navuka, 2011, 300 p. (In Russ.).

20. Dtrgeron J. *Writing Test Benches & Functional Verification of HDL Models 2gd Edition*. Springer, 1993, 508 p.

Информация об авторах

Черемисинов Дмитрий Иванович, кандидат технических наук, доцент, ведущий научный сотрудник, Объединенный институт проблем информатики Национальной академии наук Беларуси.
E-mail: cher@newman.bas-net.by

Черемисинова Людмила Дмитриевна, доктор технических наук, профессор, главный научный сотрудник, Объединенный институт проблем информатики Национальной академии наук Беларуси.
E-mail: cld@newman.bas-net.by

Information about the authors

Dmitry I. Cheremisinov, Ph. D. (Eng.), Assoc. Prof., Leading Researcher, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus.
E-mail: cher@newman.bas-net.by

Ljudmila D. Cheremisinova, D. Sc. (Eng.), Prof., Chief Researcher, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus.
E-mail: cld@newman.bas-net.by