УДК 681.32 DOI: 10.31114/2078-7707-2020-3-9-15

Верификация цифровых устройств с параллелизмом поведения

Д.И. Черемисинов, Л.Д. Черемисинова

Объединений институт проблем информатики НАНБ, г. Минск {cher, cld}@newman.bas-net.by

Аннотация — **Рассматривается задача проверки функциональной корректности реализации системы управления с параллелизмом поведения относительно спецификации на ее проектирование. Предлагается методология построения тестовой системы, позволяющей генерировать тестовую последовательность в процессе моделирования спецификации, представленной на языке параллельных алгоритмов управления. Для описания алгоритмов управления предлагается использовать язык ПРАЛУ, который позволяет задавать временную упорядоченность событий, возникающих при работе не только самого устройства управления, но и системы в целом, включая его внешнее окружение.**

Ключевые слова — **параллельный алгоритм, верификация, моделирование, язык ПРАЛУ.**

I. Введение

С увеличением сложности и ростом требований к качеству проектируемых управляющих систем резко возрастает трудоемкость их тестирования, которая растет даже быстрее, чем сложность тестируемых систем. Тестирование становится одним из наиболее важных и широко используемых методов 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. С дальнейшим ростом функциональной сложности проектов время, требуемое на верификацию, увеличивается, доходя в отдельных случаях до 90% [1]. Это мотивирует все возрастающий интерес к этой задаче.

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 является имитационное тестирование, для проведения которого необходима тестовая система, которая представляет собой специализированную программную среду, которая решает три основные задачи: генерацию тестовой последовательности; проверку правильности поведения тестируемого компонента и оценку полноты тестирования относительно исходной спецификации.

В архитектуре тестовой системы выделяются такие компоненты, как генератор тестов и тестовый оракул. Тест представляет собой последовательность наборов значений сигналов, подаваемых на вход тестируемого устройства и набор ожидаемых значений сигналов, вырабатываемых им. Назначением теста верификации является выявление ошибок в результате возникновения ситуаций несовпадения ожидаемых результатов с результатами работы тестируемого устройства при подаче соответствующей тестовой последовательности. Тестовый оракул, оценивая поведение тестируемого компонента и требования к нему, выносит вердикт об их соответствии или несоответствии [2].

Качество тестирования напрямую зависит от используемых тестовых последовательностей. В основе традиционно применяемых в практике тестирования методов построения тестовых последовательностей лежит ручная разработка, случайная и направленная генерация тестов. Хотя, несмотря на простоту, случайные тесты и позволяют быстро обнаруживать значительное число ошибок в проекте, но в этом случае нельзя сказать, насколько полно они покрывают область работы тестируемого устройства.

Одной из новых технологий для решения проблемы тестирования, является верификация программных и схемных реализаций систем управления на основе моделей (model checking) [3, 4, 5]. При тестировании этого типа тестовая последовательность генерируется на основе модели, описывающей желаемое поведение системы, которое задается спецификацией на проектирование устройства на некотором языке. Тесты строятся на основе спецификации проектируемой системы алгоритмическим способом, реакции тестируемого устройства сравниваются с ожидаемыми значениями, заданными в спецификации. Если модель корректна и тестируемое устройство должно реализовать заданное спецификацией поведение (и только его), то успешное прохождение тестов, сгенерированных надлежащим образом на основе данной модели, может служить достаточной гарантией правильности реализации системы. Предполагается, что описание спецификации является правильным и корректным. Внутренняя структура тестируемой реализации может рассматриваться как черный ящик. Оценка полноты тестирования определяется степенью покрытия сценариев работы устройства, определяемых, исходя из спецификации.

Далее рассматривается задача верификации на основе моделей для случая реактивных систем [6]. Особенность этих систем (в отличие от систем трансформационного типа) заключается в непрерывном (и, в общем случае, бесконечном) обмене сигналами с внешней

средой для выполнения задачи. Наиболее популярным способом моделирования реактивных систем является конечный автомат [7, 8], который широко используется для описания протоколов.

Однако наряду с традиционно организованными системами, которые реализуют чисто последовательное поведение, задаваемое на языках описания конечных автоматов, 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 не достаточно изучена. Одной из наиболее изученных типов моделей таких устройств является система, состоящая из одновременно работающих компонентов, которая моделируется как сеть конечных автоматов или систем помеченных переходов (labelled transition systems – LTS) [4]. Были предложены подходы к тестированию систем помеченных переходов [9], в которых эта задача рассматривалась как проверка системы на вход-выходное соответствие модели (input-output conformance – ioco relation) [10]. Тестируемое устройство соответствует спецификации в отношении ioco, если после любой последовательности входных воздействий, допускаемой спецификацией, наблюдаемые ответы тестируемого устройства соответствуют значениям, ожидаемым в спецификации.

Имеются также работы, в которых предлагаются обобщения отношения ioco на случай тестирования систем с использованием моделей «истинного параллелизма», таких как цветные сети Петри [11]. Большинство известных подходов к генерации тестовых последовательностей для систем с параллелизмом поведения, поведение которых задано на языках, в основе которых лежит аппарат сетей Петри, сначала строят граф достижимости [12]. Затем по этому графу генерируются тестовые наборы путем его обхода [13, 14, 15], аналогично тому, как это делается для случая конечных автоматов [16]. Верификация на основе графа достижимости является одним из наиболее изученных и естественных подходов к верификации систем с параллелизмом поведения. Его недостатком является экспоненциальный рост размера пространства возможных состояний системы. Как результат, граф достижимости сталкивается с проблемой взрыва числа состояний, что негативно влияет на производительность тестирования сложных систем.

В работе рассматривается задача построения тестовой системы для верификации схемной (или программной) реализации устройства управления с параллелизмом поведения. В рамках этой системы спецификация проектируемого устройства задается на языке ПРАЛУ описания параллельных алгоритмов управления [6]. На этом же языке описывается поведение объекта, управляемого проектируемым устройством. Объект управления рассматривается как часть тестового окружения. Реализация устройства рассматривается как черный ящик, для которого доступны только входы и выходы. Тестовые последовательности формируются на основе описанных алгоритмов поведения устройства и объекта управления динамически – в процессе моделирования алгоритма управления.

II. Язык задания спецификации на проектирование устройств с параллелизмом поведения

Параллелизм в спецификации возникает по разным причинам. Например, это может быть многоблочная система, в которой некоторые действия выполняются параллельно, но в разных компонентах. И, наконец, системы с «настоящим параллелизмом», когда некоторые действия выполняются параллельно и в одном и том же компоненте, и нет необходимости контролировать порядок выполнения этих действий.

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. В настоящее время в качестве языка задания спецификации на проектирование управляющих устройств используются сети взаимодействующих конечных автоматов и языки, базирующиеся на формальной модели сети Петри.

Для задания спецификации на проектирование устройств с параллелизмом поведения предлагается использовать параллельные алгоритмы логического управления, которые широко применяются при проектировании и тестировании цифровых систем. Одним из языков спецификации систем является язык ПРАЛУ [6] описания простых алгоритмов логического управления. Алгоритм на языке ПРАЛУ представляются в виде причинно-временных зависимостей между событиями, происходящими в технической системе, поведение которой описывается в терминах двоичных переменных: управляющие воздействия и сигналы о состоянии объекта управления – булевы переменные.

Основными операциями языка ПРАЛУ являются операции ожидания и действия. Операция ожидания «$\leftarrow k^{in}$» 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. В одной из интерпретаций операций действия в языке предполагается, что все внутренние переменные (если такие имеются в описании) и выходные (или управляющие) переменные сохраняют свои значения до тех пор, пока какая-либо из операций действия не

изменит их. Операции ожидания и действия могут интерпретироваться как опрос состояний датчиков объекта управления и выдача команд на исполнительную и сигнальную аппаратуру.

Алгоритм управления на ПРАЛУ представляется неупорядоченной совокупностью предложений, каждое из которых открывается меткой и состоит из одной или нескольких одинаково помеченных линейных цепочек операций языка, заканчивающихся метками перехода: «μ_i: $l_i \to \nu_i$», 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 элементов из множества $M = \{1, 2, \ldots, m\}$, которые могут интерпретироваться как частичные состояния (в том смысле, что могут существовать одновременно).

Порядок выполнения цепочек алгоритма управления в процессе его реализации определяется множеством N запуска [6], его текущие значения $N_t \subset M$. Среди предложений алгоритма выделяется одно – начальное, его метка заносится в множество N перед реализацией алгоритма.

В процессе реализации алгоритма управления цепочки запускаются независимо друг от друга. Если в некоторый момент времени для некоторой цепочки μ_i: $l_i \to \nu_i$ выполняется условие $\mu_i \subseteq N_t$ и реализуется событие k_i^{in}, с ожидания которого начинается цепочка l_i, то она запускается. При этом N_t заменяется на $N_t \setminus \mu_i$, а после завершения цепочки новое состояние N_t становится равным $(N_t \setminus \mu_i) \cup \nu_i$. Синтаксически параллельный алгоритм характеризуется наличием меток $|\mu_i| > 1$, $|\nu_i| > 1$. Альтернативное ветвление обеспечивается ограничением $(i \neq j) \wedge (\mu_i \cap \mu_j \neq \varnothing) \to (k_i^{in} \wedge k_j^{in} = 0)$.

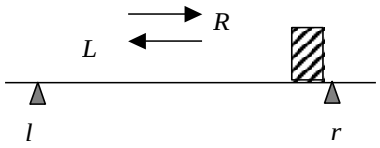

Рис. 1. Рабочий цикл манипулятора

В качестве примера приведем параллельный алгоритм, описывающий цикл работы манипулятора (рис. 1), который состоит в его перемещении между крайними позициями, регистрируемыми датчиками *r* и *l*. Движение влево и вправо инициируется сигналами *L* и *R*, соответственно. В исходном положении манипулятор находится в позиции *r* и начинает рабочий цикл после нажатия кнопки *s*. Манипулятор управляется кнопками пульта управления «включить» – *s* и «выключить» – *e*. Эти кнопки можно нажимать в течение рабочего цикла в любой последовательности, манипулятор реагирует на них только в позиции *r*: продолжает рабочий цикл, если последней была нажата кнопка *s*, или останавливается, если *e*.

Описание алгоритма управления на языке ПРАЛУ [17], определенное на множествах {*s, e, r, l*} и {*L, R*} входных (или условных) и выходных (управляющих) переменных:

РАБ_ЦИКЛ(*s, e, r, l* / *R, L*)

1: $\to$ 2.3

2: $- g \to L - l \to \bar{L} \to R - r \to \bar{R} \to 2$

3: $- s \to g - e \to \bar{g} \to 3$

Алгоритм управления является циклическим: будучи однажды запущен, он может функционировать бесконечно долго. Входными (или условными) переменными алгоритма являются переменные *s, e, r, l*, эти переменные фиксируют состояние окружающей среды. Выходные (или управляющие) переменные *L, R* инициируют движение влево и вправо. Кроме того имеется еще одна внутренняя переменная *g*, введенная для запоминания факта нажатия кнопки *s* или *e* во время рабочего цикла манипулятора: $g = 1$, если последней была нажата кнопка *s*, и $g = 0$, если *e*.

III. Моделирование реактивных систем с параллелизмом поведения

В работе [6] было предложено характеризовать цифровые устройства по типу алгоритмического описания. Устройства, моделью которых являются классические алгоритмы (алгоритмы планирования), относятся к трансформационному типу. Их целью является вычисление некоторого результата по исходным данным посредством конечной последовательности шагов. Примерами таких систем являются процессоры, компиляторы языков программирования, веб-серверы.

Цель реактивной системы [6] состоит в том, чтобы осуществлять взаимодействие с окружающей средой. Поведение реактивной системы задаётся алгоритмом управления. Функционирование реактивных систем в идеале никогда не заканчивается. Отсюда следует, что алгоритм реактивной системы не являются алгоритмом в смысле классической теории алгоритмов (отсутствует признак конечного числа шагов). В современной литературе алгоритмы управления называются протоколами взаимодействия. Тем не менее, для формализации этих алгоритмов можно использовать тот же подход, что и для трансформационных систем, – описание путем задания формального языка и абстрактного механизма вычислений. 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].

Традиционно протокол моделировался как набор взаимодействующих процессов, где каждый процесс описывается как расширенный конечный автомат, который имеет конечное число состояний. В современных системах верификации взаимодействие процессов представляется как коммуникация, в которой актами коммуникации служат транзакции через общие

структуры данных, называемые каналами. Моделирование на уровне транзакций TLM (transaction-level model) является средством повышения эффективности (оно до 1000 раз быстрее, чем моделирование на уровне RTL). Наиболее популярным и широко используемым языком моделирования уровня TLM стал язык SystemC (стандарт IEEE 1666), который является расширением языка C++. Исполняемая программа, получаемая в результате компиляции модели на языке SystemC любым ANSI-совместимым компилятором C++, реализует симулятор с интегрированными средствами управления имитацией. Параллелизм в SystemC имеет семантику чередования операций последовательных процессов. Параллельные процессы языка SystemC – это потоки (thread), которые планируются для последовательного выполнения собственным планировщиком SystemC на основе невытесняющего (non-preemptive) мультипрограммирования (cooperative multitasking).

Модель многопоточности обладает существенным недетерминизмом, и процессы языка SystemC имеют специальную организацию для устранения этого недетерминизма. Атомарные операции процессов, каковыми являются транзакции, линеаризуемы. Линеаризуемость представляет собой свойство программы, в которой результат любого параллельного выполнения операций эквивалентен их некоторому последовательному выполнению [19]. Для любого другого потока выполнение линеаризуемой операции является мгновенным: операция либо не начата, либо завершена.

Синхронизация процессов модели TLM на уровне транзакций осуществляется барьерным механизмом [20]. Барьер – это точки исходного кода, в которых каждый процесс должен приостановиться и подождать достижения барьера всеми процессами группы. В модели TLM на языке SystemC точки барьера задаются вызовами функции wait (.).

IV. Моделирование описаний реактивных систем на языке ПРАЛУ

Алгоритмы на языке ПРАЛУ допускают интерпретацию моделью TLM, для этого требуется уточнение семантики операций ожидания и действия. Суть уточнения состоит в доопределении частичного порядка реализации операций, задаваемого исходным параллельным алгоритмом, до линейного порядка. Используется модель параллелизма типа «чередование» (interleaving), в которой одновременность понимается как возможность упорядочивать операции произвольным образом. Операции ожидания и действия рассматриваются в виде композиций некоторых элементарных операций. В такой интерпретации алгоритмы на ПРАЛУ обладают свойством линеаризуемости, т.е. результат параллельного выполнения алгоритма эквивалентен некоторому последовательному выполнению атомарных операций. Транзакции в алгоритмах на ПРАЛУ представлены операциями ожидания и действия, имеющими общую переменную и описывающими событие взаимодействия [21].

Структурой данных в модели TLM алгоритма на ПРАЛУ является вектор значений переменных, компонентами которого являются пары, представляющие текущее и планируемое значения каждой переменной. Доступ к компонентам вектора переменных осуществляется посредством операции установки планируемого значения переменной алгоритма и операции проверки значения условной переменной. Реализация операции ожидания алгоритма на ПРАЛУ состоит из последовательного выполнения операций приостановки и проверки значений переменных в векторе текущих значений, операция действия – из выполнения операций установки планируемых значений переменных.

При описании процедур планирования вычислений, связанных с линейным упорядочением частично-упорядоченных операций, традиционно используется понятие ветви как совокупности последовательных подпроцессов, начинающихся с некоторой заданной операции. Последовательным подпроцессом обычно называют максимальную цепь операций процесса, находящихся в отношении непосредственного следования. Ветвь является динамическим объектом, порождаемым операцией ее образования, и уничтожаемым операцией ее прекращения.

Синхронизация параллельных цепочек алгоритма на ПРАЛУ осуществляется с помощью барьерного механизма. Точки барьера задаются операцией приостановки выполнения ветвей. Структура данных барьера синхронизации представлена в памяти очередью ОГ готовых для выполнения ветвей и очередью ОЖ ждущих ветвей. Операция образования ветви заключается в занесении первой ее операции ветви в ОГ. Смысл операции прекращения ветви ясен из ее названия: ветвь удаляется из ОГ

Основополагающим моментом при моделировании алгоритмов на ПРАЛУ является соглашение о длительности выполнения операций языка, в частности это касается операций действия. Это соглашение существенно влияет на степень соответствия изменений сигналов, производимых эмулятором и появляющихся на выходах схемной реализации. Реализация (так же как и моделирование) алгоритмов на языке ПРАЛУ выполняется в некотором предположении о длительности выполнения операций языка. Наиболее естественно 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. Это значит, что приостановки выполнения ветвей будут происходить только в операциях ожидания. Для аппаратной реализации более естественно предположение об одинаковой, но не нулевой, длительности выполнения операций действия. В этом случае приостановка выполнения ветви производится после операций действия.

При моделировании алгоритма управления из ОГ последовательно извлекаются ветви и выполняются до приостановки. Выполнение ветви G приостанавливается, если: 1) если ее начальный фрагмент «$- k^{in} \to k^{out}$» не может выполниться на множестве текущих значений переменных; 2) если ее начальный фрагмент «$- k^{in} \to k^{out}$» выполнен. В первом случае G переносится с ОЖ, во втором случае в ОЖ заносится новая ветвь, начинающаяся с операции, которая должна выполняться в G следующей. Барьер достигнут, когда очередь ОГ становится пустой. При достижении барьера запускаются процессы: 1) пересылки элементов из ОЖ в ОГ (ОЖ становится пустой); 2) ввод очередных значений переменных в планируемые значения (если система незамкнута); 3) пересылка планируемых значений в текущие для каждой компоненты вектора переменных. Затем запускается первая операция из ОГ.

Достижение барьера фиксирует такты работы эмулятора, и произведенные изменения значений 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 планируемых значений. Таким образом, процесс верификации алгоритма управления может быть выполнен двумя путями: 1) динамически в процессе отладки алгоритма управления; 2) на тестовой последовательности, полученной после моделирования алгоритма управления.

Преобразование описания алгоритма на языке ПРАЛУ в модель TLM осуществляется путем его трансляции в выражения промежуточного процедурного языка, которая осуществляется подстановками вместо операций языка ПРАЛУ композиций из элементарных операций [21]. В модели TLM на языке ПРАЛУ нет необходимости явно указывать точки барьера (в отличие от модели на SystemC). Барьер формируется автоматически в процессе трансляции. Синхронизация процессов занимает больше времени, чем вычисления, особенно в распределенных вычислениях. Операции образования, прекращения и приостановки ветвей относятся к накладным расходам на организацию вычислений.

Барьерная синхронизация считается довольно затратным в смысле памяти и времени выполнения механизмом. Однако в модели TLM на языке ПРАЛУ каждая из операций барьерного механизма может быть реализована в современных микропроцессорах одной командой, в том числе и приостановка, которая является аналогом функции wait(.) в SystemC. Отдельный планировщик не требуется.

V. Методология построения тестовой системы на основе языка ПРАЛУ

Продемонстрируем процесс построения тестовой последовательности в процессе моделирования и отладки спецификации на проектирование устройства управления, заданного на языке ПРАЛУ. В качестве примера рассмотрим приведенный выше алгоритм РАБ_ЦИКЛ(s, e, r, l/) работы манипулятора.

На языке ПРАЛУ можно описать функционирование системы в целом, включая не только задание алгоритма управления, но и описание поведения окружающей среды. Это позволяет упростить моделирование алгоритма управления, так как в этом случае достаточно изменять значения только тех переменных, изменение значений которых не фиксируется в этих двух алгоритмах, в данном случае это переменные s и e. Описание поведения окружающей среды имеет следующий вид:

ОС (R, L / r, l/)

1: $- L \to \bar{r} \to l - R \to \bar{l} \to r \to 1$

Приведенные алгоритмы на ПРАЛУ описывают функционирование системы в целом, фиксируются изменения не только внутренней переменной g, но и значения остальных переменных, кроме переменных s и e. Перед началом запуска работы манипулятора переменные имеют следующие значения: $s = e = 0$, $r = 1$, $l = g = R = L = 0$, $r = 1$, задаваемые вектором 0010000.

Продемонстрируем процесс моделирования работы описанного манипулятора для случая синхронной реализации алгоритма управления. Для этого случая в алгоритмах выделяется 10 ветвей:

1: $- L \to \bar{r}\,|_4 \to l\,|_5 - R \to \bar{l}\,|_6 \to r \to 1$

2: $- g \to L\,|_7 - l \to \bar{L}\,|_8 \to R\,|_9 - r \to \bar{R} \to 2$

3: $- s \to g\,|_{10} - e \to \bar{g} \to 3$

В таблице 1 приведены такты моделирования алгоритмов на ПРАЛУ работы манипулятора. Каждая из подтаблиц соответствует одному такту моделирования, порождаемому достигаемым барьером B_i. Столбцы каждой из подтаблиц задают номера выбираемых из ОГ на каждом такте ветвей G_j, номера ветвей в ОГ и ОЖ, вектор текущих значений переменных алгоритма на соответствующем шаге T_i и векторы P_i планируемых значений после рассмотрения ветвей. В первой строке подтаблицы показаны состояния ОГ, ОЖ, T_i и P_i в начале соответствующего такта. При переходе от i-го такта к (i+1)-му: $ОГ_{i+1} = ОЖ_i$, $T_{i+1} = P_{i+1} = P_i$. Вектор T_i порождает своими компонентами, соответствующими условным и внутренним переменным, тестовое воздействие, а полученный после выполнения такта вектор P_i – эталонную реакцию на это воздействие, задаваемыми компонентами, соответствующими управляющим переменным.

Искомые тесты представляются парами векторов: пятикомпонентный вектор тестового воздействия, компонентам которого соответствуют значения переменных s, e, r, l, g, и трехкомпонентный вектор реакций, компонентам которого соответствуют значения переменных g, R, L. Для приведенного в табл. 1 фрагмента моделирования получается следующая тестовая последовательность для верификации устройства управления, находящегося в начальном состоянии:

10100 / 100; 00101 / 101; 00001 / 101; 00011 / 100;
00011 / 110; 10011 / 010; 01001 / 010; 00100 / 000;

Таблица 1

Такты моделирования

B_i	G_j	ОГ$_i$	ОЖ$_i$	T_i	P_i
B_1		1,2,3	∅	**1**0 100 00	00 100 00
	1	2,3	1		00 100 00
	2	3	1,2		00 100 00
	3	∅	1,2,10		00 10**1** 00
B_2		1,2,10	∅	00 101 00	00 101 00
	1	2,10	1		00 101 00
	2	10	1,7		00 101 0**1**
	10	∅	1,7,10		00 101 0**1**
B_3		1,7,10		00 101 01	00 101 01
	1	7,	4		00 **0**01 01
	7	10	4,7		00 **0**01 01
	10	∅	4,7,10		00 **0**01 01
B_4		4,7,10	∅	00 001 01	00 001 01
	4	7,10	5		00 0**1**1 01
	7	10	5,7		00 0**1**1 01
	10	∅	5,7,10		00 0**1**1 01
B_5		5,7,10	∅	00 011 01	00 011 01
	5	7,10	5		00 011 01
	7	10	5,8		00 011 0**0**
	10	∅	5,8,10		00 011 0**0**
B_6		5,8,10	∅	00 011 00	00 011 00
	5	8,10	5		00 011 00
	8	10	5,9		00 011 **1**0
	10	∅	5,9,10		00 011 **1**0
B_7		5,9,10	∅	**1**0 011 10	00 011 10
	5	9,10	6		00 0**0**1 10
	9	10	6,9		00 0**0**1 10
	10	∅	6,9,3		00 0**00** 10
B_8		6,9,3	∅	0**1** 001 10	00 001 10
	6	9,10	1		00 **1**01 10
	9	10	1,9		00 **1**01 10
	3	∅	1,9,3		00 **1**00 10
B_9		1,9,3	∅	00 100 10	00 100 10
	1	9,3	1		00 100 10
	9	3	1,2		00 100 **0**0
	3		1,2,3		00 100 **0**0
B_{10}		1,2,3	∅	00 100 00	00 100 00

VI. 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. Использование языка ПРАЛУ для описания алгоритма управления дает возможность описывать поведение системы управления в целом. В настоящее время существуют программная поддержка автоматизации проектирования и отладки систем управления на ПРАЛУ, которая включает средства моделирования и синтезаторы языка ПРАЛУ в модели аппаратуры на языках Verilog и C [21].

Литература

[1] Лохов А.Л. Современные методы функциональной верификации цифровых HDL-проектов: методология ABV, библиотеки OVL и QVL // Современная электроника. 2010. № 1. С. 56–59.

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

[3] Hoffman L. Talking Model-Checking Technology // Communications of the ACM. 2008. V. 51. № 07/08. P. 110–112.

[4] Tretmans J. Model based testing with labelled transition systems. Formal Methods and Testing: Lecture Notes in Computer Science (Springer). 2008. 4949. P. 1–38.

[5] Карпов Ю. Г. Model Checking: верификация параллельных и распределенных программных систем. СПб.: БХВ-Петербург, 2010. 560 с.

[6] Закревский А. Д. Параллельные алгоритмы логического управления. Минск: Ин-т техн. кибернетики НАН Беларуси. 1999. 202 с.

[7] Lee D., Yannakakis M. Principles and methods of testing finite state machine – a survey // Proceedings of the IEEE. 1996. V. 84. № 8. P. 1090–1123.

[8] Kanso B., Chebaro O. Compositional testing for FSM-based models // International Journal of Software Engineering & Applications (IJSEA). 2014. V. 5. № 3. P. 9–23

[9] Ponce de Leon H., Delphine Longuet S.H. Model-based Testing for Concurrent Systems with Labeled Event Structures // Software Testing, Verification & Reliability. 2014. V. 24. № 7. P. 558–590.

[10] Tretmans J. Test Generation with Inputs, Outputs and Repetitive Quiescence // Software Concepts and Tools. 1996. V. 17. № 3. P. 103–120.

[11] Питерсон Дж. Теория сетей Петри и моделирование систем: пер. с англ. М. В. Горбатовой, В. Л. Торхова, В. Н. Четверикова. М.: Мир, 1984. 264 с.

[12] Watanabe H., Kudoh T. Test Suite Generation Methods for Concurrent Systems based on Coloured Petri Nets // Proceedings of the 2nd Asia-Pacific Software Engineering Conference. 1995. P. 242–251.

[13] Farooq U., Lam C.P., Li H. Towards Automated Test Sequence Generation // Proceedings of the 19th Australian Conference on Software Engineering. 2008. P. 441–450.

[14] Zhu H., He X.D. A Methodology of Testing High-level Petri Nets // Information and Software Technology. 2002. V. 44. P. 473–489.

[15] Liu J., Ye1 X., Zhou J., Song X. I/O Conformance Test Generation with Colored Petri Nets // Applied Mathematics and Information Sciences. 2014. V. 8. № 6. P. 2695–2704.

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

[17] Черемисинова Л.Д. Реализация параллельных алгоритмов логического управления. Минск: Ин-т техн.кибернетики НАН Беларуси, 2002. 246с.

[18] Halbwachs N. Synchronous Programming of Reactive Systems. Springer-Verlag. 2010. 192 p.

[19] Herlihy M.P., Wing J.M. Linearizability: A Correctness Condition for Concurrent Objects // ACM Trans. Program. Language Systems. 1990. P. 463–492.

[20] Solihin Y. Fundamentals of Parallel Multicore Architecture. CRC Press, 2015. 494 p.

[21] Черемисинов Д. И. Проектирование и анализ параллелизма в процессах и программах. Минск: Беларуская навука. 2011. 300 с.

Verification of Digital Devices with Concurrency Behavior

D.I. Cheremisinov, L.D. Cheremisinova

The United Institute of Informatics Problems of NAS of Belarus

cher@newman.bas-net.by, cld@newman.bas-net.by

***Abstract* — The problem of verifying the functional correctness of implementations of reactive control systems with respect to their design specification is considered. When solving the problem of implementing control devices, one has to deal with the parallelism present in control objects. Such objects control goal is to ensure the coordinated operation of interacting components working asynchronously and in parallel. To describe the specification for such device designs it is proposed to use the PRALU language [6] of concurrent control algorithms, which allows temporal ordering of events that occur during operation of the control device.**

The paper discusses the task of analyzing control device implementations for model input-output conformance [4, 5]. A methodology for constructing a test system for simulation-based verification is proposed. Simulation of the control device for verification should be carried out in the area of its desired operation. Using the PRALU language to describe the control algorithm makes it possible to describe the behavior of the control system as a whole: not only the control algorithm, but also the behavior of the object controlled by the designed device. The control object is considered as a part of the test environment. The implementation of the test device is considered as a black box for which only inputs and outputs are available. Test sequences are generated dynamically, i.e. in the process of simulation of the control system.

Currently, there is software support for the automation of design and debugging of the control systems, the design specification of which is in the PRALU language [21]. The software includes simulation tools and synthesizers of the PRALU language in the hardware models in Verilog and C languages.

***Keywords* — concurrent algorithm, hardware verification, simulation, PRALU language.**

References

[1] Lokhov A.L. Sovremennyye metody funktsional'noy verifikatsii tsifrovykh HDL-proyektov: metodologiya ABV, biblioteki OVL i QVL (Modern methods of functional verification of digital HDL projects: ABV methodology, OVL and QVL libraries) // Sovremennaya elektronika. 2010. № 1. P. 56–59.

[2] Kamkin A., Chupilko M. Obzor sovremennykh tekhnologiy imitatsionnoy verifikatsii apparatury (Overview of modern technologies for simulation verification of equipment) // Programmirovaniye. 2011. № 3. P. 42–49.

[3] Hoffman L. Talking Model-Checking Technology // Communications of the ACM. 2008. V. 51. № 07/08. P. 110–112.

[4] Tretmans J. Model based testing with labelled transition systems. Formal Methods and Testing: Lecture Notes in Computer Science (Springer). 2008. 4949. P. 1–38.

[5] Karpov YU. G. Model Checking: verifikatsiya parallel'-nykh i raspredelennykh programmnykh sistem (Model Checking: Verification of Parallel and Distributed Software Systems). SPb.: BKHV-Peterburg, 2010. 560 p.

[6] Zakrevskij A. D. Parallel'nye algoritmy logicheskogo upravlenija. Parallel Logic Control Algorithms. Minsk: Institut tehnicheskoj kibernetiki Nacional'noj akademii nauk Belarusi, 1999. 202 p. (in Russian).

[7] Lee D., Yannakakis M. Principles and methods of testing finite state machine – a survey // Proceedings of the IEEE. 1996. V. 84. № 8. P. 1090–1123.

[8] Kanso B., Chebaro O. Compositional testing for FSM-based models // International Journal of Software Engineering & Applications (IJSEA). 2014. V. 5. № 3. P. 9–23

[9] Ponce de Leon H., Delphine Longuet S.H. Model-based Testing for Concurrent Systems with Labeled Event Structures // Software Testing, Verification & Reliability. 2014. V. 24. № 7. P. 558–590.

[10] Tretmans J. Test Generation with Inputs, Outputs and Repetitive Quiescence // Software Concepts and Tools. 1996. V. 17. № 3. P. 103–120.

[11] Peterson J.L. Petri Net Theory and the Modeling of Systems. New York: Prentice Hall, 1981. 290 p.

[12] Watanabe H., Kudoh T. Test Suite Generation Methods for Concurrent Systems based on Coloured Petri Nets // Proceedings of the 2nd Asia-Pacific Software Engineering Conference. 1995. P. 242–251.

[13] Farooq U., Lam C.P., Li H. Towards Automated Test Sequence Generation // Proceedings of the 19th Australian Conference on Software Engineering. 2008. P. 441–450.

[14] Zhu H., He X.D. A Methodology of Testing High-level Petri Nets // Information and Software Technology. 2002. V. 44. P. 473–489.

[15] Liu J., Ye1 X., Zhou J., Song X. I/O Conformance Test Generation with Colored Petri Nets // Applied Mathematics and Information Sciences. 2014. V. 8. № 6. P. 2695–2704.

[16] Burdonov I. B, Kosachev A. S., Kuljamin V. V. Neizbytochnye algoritmy obhoda orientirovannyh grafov. Determinirovannyj sluchaj (Irredandant algorithms for traversal of directed graphs. The determinate case) // Programmirovanie. 2003. № 5. P. 11–30 (in Russian).

[17] Cheremisinova L.D. Realizatsiya parallel'nykh algoritmov logicheskogo upravleniya (Implementation of parallel logic control algorithms). Minsk: In-t tekhn.kibernetiki NAN Belarusi. 2002. 246 p.

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

[19] Herlihy M.P., Wing J.M. Linearizability: A Correctness Condition for Concurrent Objects // ACM Trans. Program. Language Systems. 1990. P. 463–492.

[20] Solihin Y. Fundamentals of Parallel Multicore Architecture. CRC Press. 2015. 494 p.

[21] Cheremisinov D. I. Proektirovanie i analiz parallelizma v processah i programmah (Design and the Analysis of Parallelism in Processes and Programs). Minsk: Belaruskaja navuka Publ. 2011. 300 p. (in Russian).