МИКРОЭЛЕКТРОНИКА, 2020, том 49, № 1, с. 66–80

СХЕМОТЕХНИКА

УДК 004.3

ВЕРИФИКАЦИЯ И СХЕМНАЯ РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ АВТОМАТОВ

© 2020 г. Н. А. Авдеев[a], П. Н. Бибило[a], *, В. И. Романов[a], **

[a]Объединенный институт проблем информатики НАН Беларуси, ул. Сурганова, 6, г. Минск, 220012 Республика Беларусь

**e-mail: bibilo@newman.bas-net.by*

***e-mail: rom@newman.bas-net.by*

Поступила в редакцию 18.03.2019 г.
После доработки 29.05.2019 г.
Принята к публикации 29.05.2019 г.

Рассматривается параллельный автомат – функциональная модель дискретного устройства управления, позволяющая учитывать параллелизм логического управления. Верификация параллельных автоматов заключается в проверке всех переходов в графе полных состояний автомата и проводится на основе моделирования. Предлагаются программные средства построения компактных тестов для выполнения верификации параллельного автомата и для получения VHDL-моделей параллельных автоматов с целью их схемной реализации. Получаемые алгоритмические VHDL-модели автоматов являются синтезируемыми, что позволяет получать схемные реализации параллельных автоматов в различных библиотеках проектирования.

DOI: 10.31857/S0544126919060024

ВВЕДЕНИЕ

Для описания поведения и схемной реализации цифровых систем на современной элементной базе заказных СБИС (сверхбольших интегральных схем) либо в базисе программируемых логических схем (ПЛИС) типа FPGA (FPGA – Field-Programmable Gate Array) разработан ряд языков, основными из которых являются Verilog и VHDL [1]. Практически все современные развитые САПР (системы автоматизированного проектирования) позволяют использовать исходные спецификации проектируемых цифровых устройств на одном из этих языков (либо на обоих вместе). Процесс синтеза в САПР СБИС автоматизирован, с увеличением сложности проектов актуальной проблемой является верификация исходных Verilog- и VHDL-описаний проектов. Последние стандарты языка VHDL [2] ориентированы на решение проблем верификации – добавлены конструкции, облегчающие написание тестирующих программ, генерацию управляемых тестов и сбор статистики. Аналогичные потребности привели к созданию языка SystemVerilog [3, 4], в котором можно применять объектно-ориентированный подход для выполнения моделирования и верификации исходных описаний.

Для проектов на Verilog и VHDL предложен язык PSL [5], позволяющий использовать аппарат ассертов – формальных утверждений о свойствах проектов. По существу, проблема верификации исходных спецификаций проектов [6], наряду с проблемой сокращения энергопотребления логических схем, реализующих исходные описания, являются основными проблемами, стоящими перед разработчиками САПР СБИС. Для повышения эффективности проектирования предпринимаются различные попытки использования языков и средств повышения уровня абстракции исходных описаний, а именно, использовать описания на языке С, графические описания в подсистеме State Flow системы MatLab и др. Например, в работе [7] показано, что описания, подготовленные в State Flow были конвертированы в VHDL-описания, которые затем были успешно реализованы на ПЛИС. САПР становятся “многоязыковыми”, например, система моделирования Questa Sim [8] поддерживает языки VHDL, Verilog, PSL, System Verilog, SystemC и C/C++.

Типовые решения (высокоуровневые описания) для схем, реализующих вычислительные операции, обычно имеются в библиотеках проектных решений, основные проблемы при вери-

Сокращения: СБИС – сверхбольшие интегральные схемы; ПЛИС – программируемые логические интегральные схемы; САПР – системы автоматизированного проектирования; ПРАЛУ – Простые Алгоритмы Логического Управления; ПЛМ – программируемая логическая матрица.

фикации возникают для управляющей логики, реализующей параллельные процессы управления.

Язык ПРАЛУ (Простых Алгоритмов Логического Управления) предназначен для описания параллельных алгоритмов логического управления. Его характерными особенностями являются логическая стройность, простота, компактность получаемых описаний, использование двоичных (булевых) переменных для входных и выходных переменных устройств управления, алгоритм функционирования которых задан на языке ПРАЛУ. Полное описание языка ПРАЛУ содержится в работе [9], где предложена модель параллельного автомата и преобразование ее в модель секвенциального автомата [10], являющейся промежуточной при схемной реализации параллельных автоматов, представленных на языке ПРАЛУ.

В данной работе описывается верификация исходных ПРАЛУ-описаний параллельных автоматов. Верификация может быть проведена на основе моделирования исходных описаний и на основе проверки семантических свойств автомата. Предлагаются средства перехода к эквивалентным по поведению VHDL-моделям параллельных автоматов и VHDL-моделям секвенциальных автоматов, получаемые алгоритмические VHDL-модели автоматов являются синтезируемыми, т.е. по ним возможно автоматическое построение логических схем в заданных технологических базисах. Отметим, что в работах [9, 10] предлагается схемная реализация секвенциальных (и тем самым и параллельных) автоматов в виде программируемой логической матрицы (ПЛМ) с памятью в виде регистра RS-триггеров. Получение VHDL-моделей секвенциальных автоматов позволяет получать логические схемы в произвольном функционально полном базисе логических элементов ASIC либо FPGA. Кроме того, использование VHDL-моделей позволяет также привлечь для верификации мощные средства промышленных систем моделирования VHDL-описаний цифровых систем, например средства системы Questa Sim (ф. Mentor Graphics). Таким образом, предлагаемый подход к верификации имеет комплексный характер – при верификации проверяются как семантические свойства, так и осуществляется функциональная верификация проекта, а получение VHDL-моделей позволяет проводить как быстрое моделирование для функциональной верификации, так и проводить синтез логических схем, в различных технологических базисах, для чего может быть использован синтезатор Leonardo Spectrum [11]. В отличие от формальной верификации, когда на эквивалентность поведения проверяются два VHDL-описания цифровой системы, в данной статье под *верификацией* далее будет пониматься проверка правильности исходного VHDL-описания, т.е. проверка соответствия составленного синтезируемого VHDL-описания проектируемой цифровой системы спецификациям на проектирование [6]. Так как параллельные автоматы могут входить в состав более сложных проектов цифровых систем, то предлагаются также средства построения компактных функциональных тестов для выполнения функциональной верификации параллельных автоматов.

1. ПРАЛУ-ОПИСАНИЯ ПАРАЛЛЕЛЬНЫХ АВТОМАТОВ

Модель параллельного автомата [9, 10] позволяет учитывать параллелизм логического управления и является функциональной моделью дискретного устройства управления. Данная модель состоит из множества X входных булевых переменных, множества Y выходных булевых переменных и множества цепочек (строк-переходов). Согласно [9] цепочки имеют вид

$$\mu_i: -k_i^1 \rightarrow k_i^2 \Rightarrow \nu_i, \qquad (1)$$

в которой операция $-k_i^1$ или $\rightarrow k_i^2$ (или обе вместе) может отсутствовать. В общем случае цепочка состоит из четырех частей:

μ_i – множество начальных частичных состояний i-ой цепочки;

$-k_i^1$ – операция ожидания события k_i^1;

$\rightarrow k_i^2$ – операция действия;

ν_i – множество заключительных частичных состояний цепочки.

В формуле (1) двоеточие служит разделителем, а стрелка $\Rightarrow$ перед ν_i играет роль операции внесения элементов в текущее множество запуска цепочек. Множества μ_i, ν_i – это подмножества из множества M частичных состояний, в работе [9] частичные состояния названы метками. Будем предполагать, что начальное частичное состояние всегда имеет номер 1, и что частичные состояния последовательно нумеруются (номера идут без пропусков). Сначала поясним, что представляют собой операции ожидания и действия, а затем – каким образом выполняется алгоритм в целом, т.е. как выполняются (срабатывают) цепочки, как они взаимодействуют друг с другом и какую роль играет множество запуска цепочек. Ниже представлен пример параллельного автомата Zak178 из книги [9, с. 178]

```
  1 :  -u    →ab   ⇒9
  9 :  -u′         ⇒2.3
  2 :  -v′w  →b′c  ⇒10
 10 :  -w′   →b    ⇒11
 11 :        →c′   ⇒2
  2 :  -v    →a′c  ⇒4.5
  3 :  -uw   →d    ⇒6
  4 :  -u′v′ →a    ⇒12
 12 :  -u    →a′   ⇒4
  4 :  -u    →ab′  ⇒7
  5 :  -v′w  →c′   ⇒8
6.7.8 :      →a′d′ ⇒13
 13 :  -w′          ⇒1
```

В данном примере X = {u, v, w} – множество *входных* булевых (двоичных) переменных, Y = {a, b, c, d} – множество *выходных* булевых переменных параллельного автомата. Множество M образуют частичные состояния с номерами от 1 до 13. В записи (1) k_i^1, k_i^2 представляют собой элементарные конъюнкции булевых переменных (с инверсиями либо без инверсий), штрихами помечены инверсии булевых переменных. Конъюнкции k_i^1 образуются из булевых переменных множества X, конъюнкции k_i^2 образуются из булевых переменных множества Y. Если переменные входят как во множество X, так и во множество Y, то такие переменные в ПРАЛУ-описаниях объявляются *внутренними*. В рассматриваемом примере (табл. 1) внутренних переменных нет – в разделе INTER нет переменных.

Если конъюнкция k_i^1 отсутствует в (1), то предполагается, что она тождественно равна 1. Операция $-k_i^1$ представляет собой операцию ожидания события k_i^1. Выполнение этой операции сводится к ожиданию события, когда переменные, входящие в конъюнкцию k_i^1, примут значения, обращающие k_i^1 в единицу. Операция действия $\rightarrow k_i^2$ означает присвоение таких значений переменным конъюнкции k_i^2, при которых k_i^2 обращается в единицу. Если же k_i^2 отсутствует в (1), то предполагается, что никакие операции действия в данной цепочке не выполняются.

Предполагается синхронная модель параллельного автомата – в каждом такте любая из цепочек может сработать (либо не сработать). Управление цепочками осуществляется в дискретном (тактируемом) времени с помощью множества $N \subseteq M$, называемого множеством запуска. Цепочка (1) срабатывает, если в текущем множестве запуска содержится множество μ_i и если выполнилась операция ожидания $-k_i^1$. В начале работы алгоритма в текущее множество запуска включается частичное состояние 1. Срабатывание цепочки в текущем такте предполагает удаление множества μ_i из текущего множества запуска, выполнение операции действия $\rightarrow k_i^2$, и включении множества ν_i в текущее множество запуска. Алгоритм заканчивает работу, если множество запуска содержит заключительное частичное состояние, т.е. частичное состояние, которого нет в множествах μ_i начальных частичных состояний цепочек. В рассматриваемом примере заключительного частичного состояния нет – алгоритм “зациклен”.

Полное состояние (далее просто *состояние*) параллельного автомата в текущем такте образуют частичные состояния из множества N запуска цепочек.

В процессе работы алгоритма некоторые цепочки могут выполняться одновременно (параллельно), поэтому такой формализм позволяет описывать параллельные алгоритмы логического управления. При этом множество цепочек должно удовлетворять определенным требованиям, например, цепочки, имеющие одинаковые множества начальных частичных состояний, должны иметь ортогональные конъюнкции в операциях ожидания. Другие требования к корректности исходных ПРАЛУ-описаний изложены в [9, 10].

2. ВЕРИФИКАЦИЯ ПАРАЛЛЕЛЬНЫХ АВТОМАТОВ

Верификация исходных описаний параллельных автоматов, представленных текстовыми файлами (первый столбец табл. 1) осуществляется с помощью специальной программы QPralu, которая позволяет провести моделирование и проверить семантические свойства автомата, о которых речь будет идти далее. Пример автомата Zak178 в формате программы QPralu задан в левой части табл. 1.

Результаты моделирования параллельного автомата Zak178 на 33 псевдослучайных наборах заданы в табл. 2: во втором столбце – входные наборы (тест); в третьем столбце – значения выходных переменных (выходные реакции), в четвертом столбце – состояния (состояния параллельного автомата для следующего такта моделирования). По результатам моделирования может быть построен граф G (рис. 1) переходов, вершинами которого являются состояния параллельного автомата (см. строки из четвертого столбца табл. 1).

Под *функциональной верификацией* (далее верификацией) *параллельного автомата* будет пони-

Таблица 1. ПРАЛУ-описание параллельного автомата Zak178

ПРАЛУ-описание	Комментарии
TITLE Zak178	Имя описания - Zak178
FORMAT PA	Язык (формат) описания – PA
AUTHOR Zakrevskij	Автор – Zakrevskij
	Дата создания – 29 октября 2018 года
DATE 29/10/2018	Имя проекта – VLSI
	DCL_PIN – Ключевое слово начала декларации
PROJECT VLSI	переменных
DCL_PIN	EXT – Ключевое слово декларации внешних
EXT	переменных
INP	INP – Ключевое слово декларации входных
u v w	переменных
OUT	u v w – Список имен входных переменных
a b c d	OUT – Ключевое слово декларации выходных
INTER	переменных
END_PIN	a b c d – Список имен выходных переменных
BLOCK main	INTER – Ключевое слово декларации внутренних
1>9: u>a*b;	переменных
9>2.3: ^u>; 2>10:	END_PIN – Ключевое слово окончания декларации
	переменных
^v*w>^b*c;	BLOCK – Ключевое слово
10>11: ^w>b;	Первая цепочка
11>2: >^c;	
2>4.5: v>^a*c;	
3>6: u*w>d;	
4>12: ^u*^v>a;	
4>7: u>a*^b;	…… Цепочки
12>4: u>^a;	
5>8: ^v*w>^c;	
6.7.8>13: >^a*^d;	
13>1: ^w>;	Последняя цепочка, переход на начальное состояние
	END_BLOCK – Ключевое слово окончания описания блока
	END_Zak178 – Ключевое слово окончания описания автомата Zak178
END_BLOCK main	
END_Zak178	

маться проверка выполнения всех переходов (обход всех дуг) в графе *G*. Позже будет показано, что анализ графов переходов, построенных по результатам моделирования, позволяет строить компактные тесты для верификации параллельного автомата. От длины теста (числа тестирующих наборов) в общем случае зависит вид графа *G* (табл. 3) и, соответственно, длина компактного теста, требуемого для его верификации.

Для параллельных алгоритмов в работе [9] сформулированы пять семантических свойств: безызбыточность, восстанавливаемость, непротиворечивость, устойчивость, самосогласованность. Формальные методы проверки таких свойств для параллельных автоматов имеют отдельный интерес, они представлены в [10], и в данной работе не рассматриваются. Однако некоторые из таких свойств могут быть проверены на уровне VHDL-описаний параллельных автоматов.

Таблица 2. Результат моделирования параллельного автомата Zak178 на 33 наборах

Такт	Входные наборы (тест) u v w	Выходные реакции a b c d	Следующее состояние параллельного автомата	Циклы в графе (рис. 1)	
				Цикл 1	Цикл 2
1	2	3	4	5	6
0			1		
1	000	0000	1.	1	
2	111	1100	9.	9.	
3	010	1100	2.3.	2.3.	
4	110	0110	3.4.5.	3.4.5.	
5	000	1110	3.5.12.	3.5.12.	
6	001	1100	3.8.12.	3.8.12.	
7	001	1100	3.8.12.	3.8.12.	
8	001	1100	3.8.12.	3.8.12.	
9	100	0100	3.4.8.	3.4.8.	
10	001	1100	3.8.12.	3.8.12.	
11	011	1100	3.8.12.	3.8.12.	
12	010	1100	3.8.12.	3.8.12.	
13	110	0100	3.4.8.	3.4.8.	
14	010	0100	3.4.8.	3.4.8.	
15	011	0100	3.4.8.	3.4.8.	
16	011	0100	3.4.8.	3.4.8.	
17	111	1001	6.7.8.	6.7.8.	
18	100	0000	13.	13.	
19	100	0000	1.	1.	1.
20	110	1100	9.		9.
21	001	1100	2.3.		2.3.
22	011	0110	3.4.5.		3.4.5.
23	100	1010	3.5.7.		3.5.7.
24	000	1010	3.5.7.		3.5.7.
25	000	1010	3.5.7.		3.5.7.
26	100	1010	3.5.7.		3.5.7.
27	100	1010	3.5.7.		3.5.7.
28	110	1010	3.5.7.		3.5.7.
29	111	1011	5.6.7.		5.6.7.
30	111	1011	5.6.7.		5.6.7.
31	001	1001	6.7.8.		6.7.8.
32	111	0000	13.		13.
33	000	0000	1.		1.

Таблица 3. Результаты моделирования параллельного автомата Zak178

Число псевдослучайных тестирующих наборов	Результаты моделирования		Результаты построения компактных тестов для верификации	
	Граф G		Число наборов компактного теста (дуг в циклах)	Число циклов
	Число вершин	Число дуг		
33	11	18	25	3
200	18	44	110	11
20000	21	63	181	18

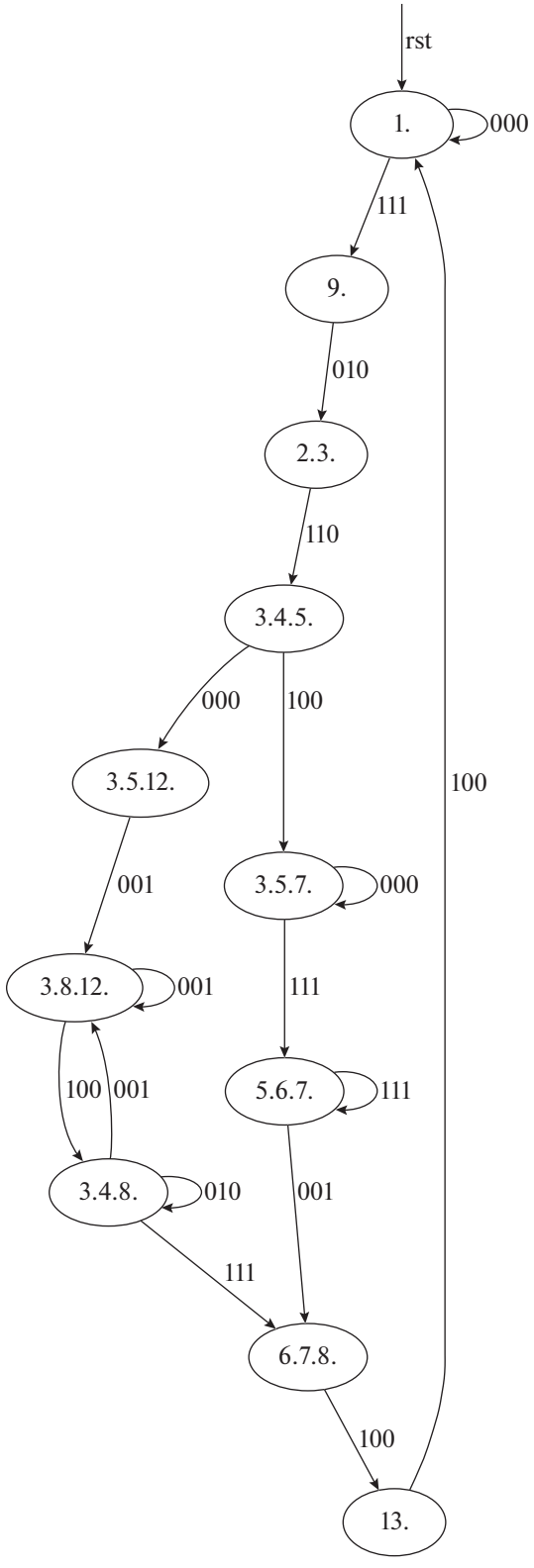

Рис. 1. Граф *G* переходов параллельного автомата, построенный по результатам моделирования (табл. 1) на 33 входных наборах.

3. СИНХРОННАЯ VHDL-МОДЕЛЬ ПАРАЛЛЕЛЬНОГО АВТОМАТА

Предлагается представлять VHDL-описание параллельного автомата в виде двух взаимодействующих процессов (оператор *process*). Для каждого текущего частичного состояния i вводится соответствующий сигнал текущего Li и сигнал n_Li следующего частичного состояния. В первом процессе (*process p1*) в текущем такте t_i определяются элементарные цепочки, которые работают параллельно и следующие (для следующего такта t_{i+1} дискретного времени) значения переменных y_j действия и значения сигналов частичных состояний. Подготавливаемые значения сигналов для переменных действия обозначаются через n_y_j, а подготавливаемые значения переменных множества запуска – через n_Li. Во втором процессе (*process p2*) осуществляется смена состояния параллельного автомата по переднему фронту сигнала синхронизации *clk*. Для этого используется оператор *if* (*clk = '1' and clk'event*) *then* нахождения переднего фронта сигнала *clk*. Установка автомата в начальное состояние (переменные действия имеют значение 0, а в множество запуска включается только сигнал $L1$) осуществляется по единичному значению сигнала *rst* разрешения, который имеет более высокий приоритет по отношению к другим входным сигналам. На момент смены такта значения сигналов, соответствующих переменным ожидания из конъюнкций k_i^1, должны быть установившимися. Заметим, что в исходном описании параллельного автомата сигнал *clk* отсутствует. В Приложении 1 дана VHDL-модель Zak178_PA параллельного автомата Zak178, ПРАЛУ описание которого дано в табл. 1. Вектор *state* в VHDL-модели автомата отражает состояние параллельного автомата и нужен для тестирующей программы, которая выдает результаты VHDL-моделирования в таком же формате, что и программа QPralu моделирования исходного описания автомата. Формат входных тестов является одинаковым для обоих вариантов моделирования.

4. ВЕРИФИКАЦИЯ VHDL-ОПИСАНИЙ ПАРАЛЛЕЛЬНЫХ АВТОМАТОВ

Верификация осуществляется на основе моделирования в системе Questa Sim. Моделирование может быть выполнено на тех же тестах, что и моделирование исходных описаний, результаты VHDL-моделирования выдаются в той же форме, что и результаты моделирования в программе QPralu. Некоторые из семантических свойств могут быть проверены на VHDL-описаниях параллельных автоматов.

Свойство безызбыточности: алгоритм *безызбыточен,* если в нем нет операций, которые никогда не выполняются, ни при каких условиях не могут быть выполнены [9, с. 61]. Очевидно, что удаление соответствующих фрагментов из алгоритма не скажется на его функциональных свойствах. На уровне VHDL-описаний проверка свойства осуществляется выполнением моделирования с покрытием VHDL-кода в системе Questa Sim. Покрытие VHDL-кода при моделировании подробно описано в [2]. Непокрытые конструкции (строки VHDL-кода) будут означать возможную избыточность описания, по которой можно легко (однозначно) установить избыточность в исходном описании параллельного автомата.

Свойство восстанавливаемости: алгоритм *восстанавливаем*, если из любого достижимого состояния он может вернуться в исходное. На уровне VHDL-описаний решение осуществляется построением и анализом графа переходов между состояниями - из любой вершины графа должен быть путь в начальное состояние. В число функциональных тестов могут входить тесты, позволяющие приходить из начального состояния в заданное состояние (заданную вершину графа переходов) и выходить из заданного состояния в начальное состояние.

Свойство непротиворечивости: алгоритм *непротиворечив*, если при его выполнении не может возникнуть ситуация, в которой одновременно выполняются несовместимые между собой операции. Для языка ПРАЛУ такая противоречивая ситуация возникает при совместном выполнении двух операций действия с ортогональными конъюнкциями: в этом случае некоторой внутренней либо выходной переменной присваиваются два различных значения, что невозможно.

5. ПОСТРОЕНИЕ КОМПАКТНЫХ ФУНКЦИОНАЛЬНЫХ ТЕСТОВ ПО РЕЗУЛЬТАТАМ МОДЕЛИРОВАНИЯ

Основной подход к верификации VHDL-моделей схем управляющей логики реализуется на основе моделирования и комплексного тестирования, целью которых являются соответствующие проверки для подтверждения того, что функциональность VHDL-описания будет правильной [2, 6, 7]. Данный подход не ограничен стилями исходных описаний, основные проблемы заключаются в генерации направленных функциональных тестов и организации тестирования. В рамках данного подхода, являющегося наиболее распространенным подходом к верификации, предлагается решать поставленную задачу верификации VHDL-описаний параллельных автоматов. Обзор существующих моделей параллельно функционирующих распределенных систем, методов и инструментальных средств их верификации представлен в [11], где указывается, что одной из основных проблем, возникающих при верификации взаимодействующих цифровых систем, описываемых в терминах состояний, является комбинаторный взрыв числа возможных параллельных состояний всей системы в целом. Это касается как формальной верификации, так и верификации на основе моделирования.

Таблица 4. Состояния (вершины) графа *G* переходов

Имя состояния	Состояние в графе переходов (рис. 2)
s0	1.
s1	9.
s2	2.3.
s3	4.5.6.
s4	5.6.7.
s5	6.7.8.
s6	13.
s7	3.4.5.
s8	3.5.12.
s9	4.6.8.
s10	6.8.12.
s11	3.5.7.
s12	3.7.8.
s13	3.8.12.
s14	3.4.8.
s15	3.10.
s16	3.11.
s17	2.6.
s18	6.10.
s19	6.11.
s20	5.6.12.

В отличие от VHDL-модели конечного автомата, граф переходов которого извлекается в системе Questa Sim и может быть проанализирован [2], из VHDL-модели (см. Приложение 1) параллельного автомата система Questa Sim граф переходов не извлекает, так как модель параллельно автомата является более общей, чем модель конечного автомата. Поэтому для автоматизированного построения графа *G* состояний параллельных автоматов и получения компактного функционального теста, обеспечивающего покрытие всех дуг данного графа, были разработаны и модифицированы соответствующие программы [12, 13] позволяющие обрабатывать результаты моделирования (см. табл. 1), содержащие тестовые последовательности с миллионами тестовых наборов. Входные тестовые наборы, соответствующие дугам, вошедшим в покрытие графа, образуют тест

Таблица 5. Циклы, покрывающие все дуги графа *G* переходов

Номер цикла	Последовательность вершин
1	s0 s0 s1 s1 s2 s2 s15 s15 s16 s2 s18 s18 s19 s17 s17 s3 s3 s4 s4 s5 s6 s6 s0
2	s0 s1 s2 s7 s7 s8 s8 s3 s20 s20 s10 s10 s9 s9 s10 s9 s5 s6 s0
3	s0 s1 s2 s15 s16 s17 s18 s19 s17 s3 s5 s6 s0
4	s0 s1 s2 s7 s8 s7 s13 s13 s14 s14 s13 s9 s5 s6 s0
5	s0 s1 s2 s15 s18 s19 s17 s3 s5 s6 s0
6	s0 s1 s2 s7 s8 s13 s14 s12 s12 s5 s6 s0
7	s0 s1 s2 s7 s8 s3 s20 s9 s5 s6 s0
8	s0 s1 s2 s7 s8 s3 s20 s3 s5 s6 s0
9	s0 s1 s2 s7 s8 s3 s10 s9 s5 s6 s0
10	s0 s1 s2 s7 s8 s13 s14 s5 s6 s0
11	s0 s1 s2 s3 s20 s10 s9 s5 s6 s0
12	s0 s1 s2 s7 s11 s11 s12 s5 s6 s0
13	s0 s1 s2 s7 s11 s4 s5 s6 s0
14	s0 s1 s2 s7 s8 s9 s5 s6 s0
15	s0 s1 s2 s7 s11 s5 s6 s0
16	s0 s1 s2 s7 s4 s5 s6 s0
17	s0 s1 s2 s7 s5 s6 s0
18	s0 s0

для функциональной верификации параллельного автомата. Ориентированный граф *G* переходов, построенный по результатам моделирования, должен удовлетворять следующему условию: из начального состояния в каждую вершину должен существовать путь и из каждой вершины в начальную вершину также должен быть путь. При выполнении данного условия может быть построен компактный тест (в виде циклов), покрывающий все дуги графа.

Покрытие дуг графа *G* осуществляется циклами (замкнутыми контурами) из начальной вершины 1. Например, граф переходов параллельного автомата, построенный по результатам моделирования на 20000 входных наборов (см. табл. 2 и рис. 2) содержит 21 вершину. Имена вершин графа *G* даны в табл. 4. Граф *G* имеет 63 дуги и покрывается 18 циклами (табл. 5), которые содержат 181 дугу. Так как каждой дуге соответствует входной набор, то функциональный тест, обеспечивающий переходы по всем 63 дугам графа, содержит 181 входной набор. Результаты моделирования на 23 входных наборах компактного теста, соответствующих первому циклу, показаны в табл. 6.

6. СХЕМНАЯ РЕАЛИЗАЦИЯ ПАРАЛЛЕЛЬНЫХ И СЕКВЕНЦИАЛЬНЫХ АВТОМАТОВ

Будем предполагать, что исходное ПРАЛУ-описание, по которому осуществляется схемная реализация, является корректным. Синтез логических схем, реализующих параллельные автоматы, может выполняться двумя способами.

Первый способ – переход к синтезируемой VHDL-модели параллельного автомата и выполнение синтеза в одном из промышленных синтезаторов схем ASIC либо ПЛИС типа FPGA. Далее в качестве такого синтезатора будет использоваться синтезатор LeonatdoSpectrum [14].

Второй способ – получение матричного описания секвенциального автомата, поведение которого совпадает с поведением параллельного авто-

Таблица 6. Результаты моделирования на входных наборах компактного теста (по циклу 1)

Такт	Входные наборы (тест) u v w	Цикл 1	Выходные реакции a b c d
0		s0	
1	000	s0	0000
2	000	s0	0000
3	100	s1	1100
4	101	s1	1100
5	000	s2	1100
6	000	s2	1100
7	001	s15	1010
8	011	s15	1010
9	100	s16	1110
10	010	s2	1100
11	101	s18	1011
12	101	s18	1011
13	000	s19	1111
14	001	s17	1101
15	000	s17	1101
16	110	s3	0111
17	010	s3	0111
18	100	s4	1011
19	111	s4	1011
20	101	s5	1001
21	111	s6	0000
22	101	s6	0000
23	010	s0	0000

мата, и переход к синтезируемой VHDL-модели секвенциального автомата [10].

Секвенциальный автомат является моделью цифровой системы, функционирующей в дискретном времени, и состоит из множества S секвенций s_i. Каждая секвенция s_i имеет форму $f_i \vdash k_i$, где f_i – булева функция входных и внутренних переменных, k_i – элементарная конъюнкция внутренних и выходных переменных. Каждая секвенция $f_i \vdash k_i$ описывает определенное требование к поведению цифровой системы: если в некоторый момент времени f_i принимает значение 1, то непосредственно вслед за этим (в следующем такте дискретного времени) k_i также принимает значение 1. Будем интерпретировать данную систему секвенций как инерционный секвенциальный автомат. В инерционном секвенциальном автомате предполагается следующее [10]: если во всех элементарных конъюнкциях k_i, соответствующих функциям f_i, принимающим в текущий момент времени значение 1, отсутствует символ некоторой переменной, то считается, что эта переменная сохраняет свое значение. В табл. 7 дано матричное описание секвенциального автомата Zak178 в формате программы QPralu. Переход от модели параллельного автомата к эквивалентной по поведению синхронной модели секвенциального автомата основыва-

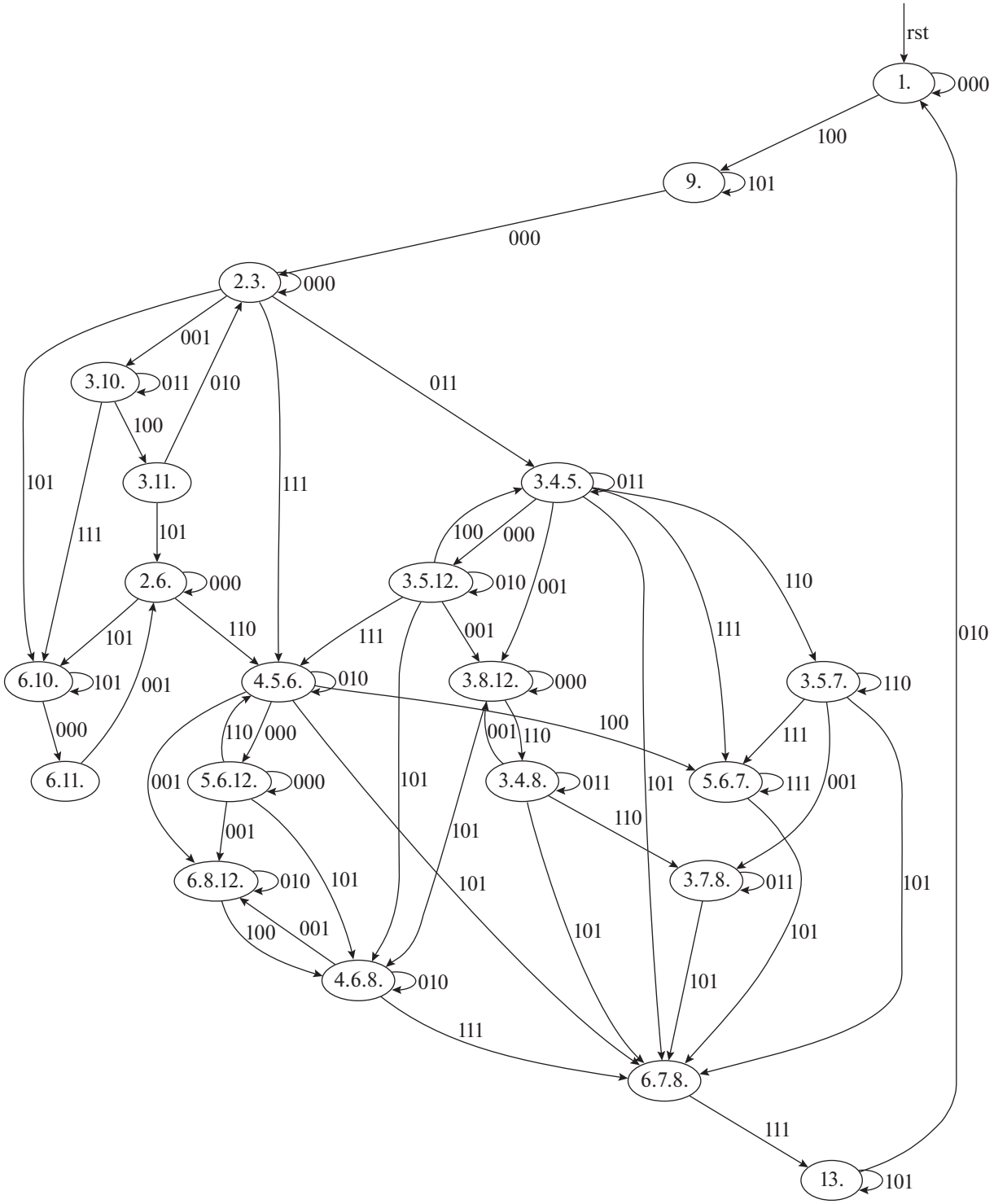

Рис. 2. Граф *G* переходов параллельного автомата, построенный по результатам моделирования на 20000 входных наборах.

ется на сложных методах кодирования состояний параллельного автомата с нахождением параллельных состояний. Такие методы подробно описаны в [9, 10], три из таких методов программно реализованы в QPralu, основное их различие – длина получаемого кода состояний автомата. В примере (табл. 7) для кодирования тринадцати частичных состояний потребовалось шесть кодирующих переменных Z0,..., Z5. В аналитической форме секвенциальный автомат (табл. 7) задается формулами

Таблица 7. Секвенциальный автомат

Матричное описание секвенциального автомата	Комментарии
TITLE Zak178	Имя описания – Zak178
FORMAT SA	Язык (формат) описания – SA
AUTHOR Zakrevskij	Автор – Zakrevskij
DATE 29/10/2018	Дата создания – 29 октября 2018 года
PROJECT VLSI	Имя проекта – VLSI
DCL_PIN	DCL_PIN – Ключевое слово начала декларации
EXT	переменных
INP	EXT – Ключевое слово декларации внешних
u v w	переменных
OUT	INP – Ключевое слово декларации входных
a b c d	переменных
INTER	u v w – Список имен входных переменных
Z0 Z1 Z2 Z3 Z4 Z5	OUT – Ключевое слово декларации выходных
END_PIN	переменных
FUNCTION	a b c d – Список имен выходных переменных
1--111--- 0-----11--	INTER – Ключевое слово декларации внутренних
	переменных
0--01---- -0111-----	Z0 Z1 Z2 Z3 Z4 Z5 – Список имен внутренних
	переменных
-01-0111- --0----01-	END_PIN – Ключевое слово окончания декларации
	переменных
--0-001-- --1-0--1--	FUNCTION – Ключевое слово
	Первая секвенция в матричной форме
----0110- ----1---0-	
-1-0111- --000-0-1-	
1-100--- 1--------1	
	…… Секвенции в матричной форме
00--0-00- ----101---	
1---0-00- ----1110--	
1---0-010 ----0-0---	
	Последняя секвенция в матричной форме
-01-000-- --1-----0-	END_FUNCTION – Ключевое слово
	END_Zak178 – Ключевое слово окончания описания
---101011 -10---0--0	Zak178
-0110--- --1-------	
END_FUNCTION	
END_Zak178	

Таблица 8. Сравнение схемных реализаций VHDL-моделей параллельных и секвенциальных автоматов

Имя автомата	VHDL-модель							
	Параллельный автомат				Секвенциальный автомат			
	S	*L*	*Tr*	τ	*S*	*L*	*Tr*	τ
Zak178	75966	92	17	***4.31***	***53959***	81	10	6.48
Zak135	75040	90	17	***4.11***	***53479***	79	10	7.02
Pott1	56258	65	13	***4.16***	***56090***	65	13	***4.16***
Wagon	***59762***	62	15	***3.33***	60102	80	13	4.78
Kamkin_2	60164	86	11	***4.60***	***41007***	51	9	4.96

$$uz_0z_1z_2 \vdash \bar{z}_0ab,$$
$$\bar{u}\bar{z}_0z_1 \vdash \bar{z}_1z_2z_3z_4,$$
$$\bar{v}w\bar{z}_1z_2z_3z_4 \vdash \bar{z}_2\bar{b}c,$$
$$\bar{w}\bar{z}_1\bar{z}_2z_3 \vdash z_2\bar{z}_4b,$$
$$\bar{z}_1z_2z_3\bar{z}_4 \vdash z_4\bar{c}$$
$$v\bar{z}_1z_2z_3z_4 \vdash \bar{z}_2\bar{z}_3\bar{z}_4\bar{a}c,$$
$$uw\bar{z}_0\bar{z}_1 \vdash z_0d,$$
$$\bar{u}\bar{v}\bar{z}_1\bar{z}_3\bar{z}_4 \vdash z_4\bar{z}_5a,$$
$$u\bar{z}_1\bar{z}_3\bar{z}_4 \vdash z_4z_5a\bar{b},$$
$$u\bar{z}_1\bar{z}_3z_4\bar{z}_5 \vdash \bar{z}_4\bar{a},$$
$$\bar{v}w\bar{z}_1\bar{z}_2\bar{z}_3 \vdash z_2\bar{c},$$
$$z_0\bar{z}_1z_2\bar{z}_3z_4z_5 \vdash z_1\bar{z}_2\bar{a}\vec{d},$$
$$\bar{w}z_0z_1\bar{z}_2 \vdash z_2.$$

VHDL-модель секвенциального автомата в целом аналогична VHDL-модели параллельного автомата, например, первая секвенция $uz_0z_1z_2 \vdash \bar{z}_0ab$, описывается следующим фрагментом VHDL-кода:

```
if ((u and Z0 and Z1 and Z2) = '1') then
      n_Z0 <= '0'; n_a<= '1'; n_b<= '1';
end if;
```

Результаты схемной реализации VHDL-моделей параллельных и секвенциальных автоматов в библиотеке проектирования заказных КМОП СБИС, выполненные в синтезаторе LeonardoSpectrum даны в табл. 8.

Первые три примера Zak178, Zak135, Pott1, Wagon взяты из [9, 10], Kamkin_2 – пример параллельной сети из двух узлов (конечных автоматов), рассматриваемый в [16]. Параметры схемной реализации VHDL-моделей: *S* – площадь в условных единицах; *L* – общее число элементов (комбинационных и элементов памяти – триггеров); *Tr* – число триггеров; τ – задержка (нс).

Результаты схемной реализации, приведенные в табл. 8, показывают, что реализация VHDL-моделей секвенциальных автоматов чаще дает схемы меньшей сложности, однако схемные реализации VHDL-моделей параллельных автоматов являются более быстродействующими, все зависит от вида автомата – числа входов, выходов, цепочек и распределения подмножеств меток по цепочкам.

Программный инструментарий для моделирования ПРАЛУ-описаний параллельных автоматов разработан на языке программирования C++ в среде QT [15]. Разработанная программа QPralu позволяет провести моделирование параллельных автоматов, выполнить проверку их семантических свойств, получить секвенциальные автоматы и синтезируемые VHDL-модели автоматов.

ЗАКЛЮЧЕНИЕ

Более половины времени и других ресурсов при разработке проектов цифровых систем расходуется на верификацию, поэтому использование компактных, ясных ПРАЛУ-описаний алгоритмов логического управления позволяет более эффективно провести их запись и последующую верификацию. Преобразование модели параллельного автомата в модель секвенциального автомата позволяет уменьшать сложность логических схем при схемной реализации VHDL-моделей автоматов. Предложенный в данной статье подход в целом повышает эффективность проектирования и может сократить сроки разработки проектов цифровых систем, реализующих параллельные алгоритмы логического управления, характеризуемые большим числом состояний и управляющих сигналов.

Приложение 1.

VHDL-описание параллельного автомата Zak178

```
Library IEEE;
use IEEE.std_logic_1164.all;

entity Zak178_PA is
port (clk, rst  : in std_logic;
u, v, w : in std_logic;
a, b, c, d : out std_logic);
end Zak178_PA;

architecture behavior of Zak178_PA is
signal L1, L2, L3, L4, L5, L6, L7, L8, L9,
        L10, L11, L12, L13:  std_logic;
signal n_L1, n_L2, n_L3, n_L4, n_L5, n_L6, n_L7, n_L8, n_L9,
        n_L10, n_L11, n_L12, n_L13:  std_logic;
signal n_a, n_b, n_c, n_d :  std_logic;
signal state : std_logic_vector (1 to 13);
signal a_net, b_net, c_net, d_net : std_logic;

begin
state<= L1 & L2 & L3 & L4 & L5 & L6 & L7 & L8 & L9 &
          L10 & L11 & L12 & L13;
a <= a_net; b <= b_net; c <= c_net; d <= d_net;

p1: process ( u, v, w, L1, L2, L3, L4, L5, L6, L7, L8, L9,
               L10, L11, L12, L13, a_net, b_net, c_net, d_net)
begin
    n_L1 <= L1; n_L2 <= L2;   n_L3 <= L3;   n_L4 <= L4;
    n_L5 <= L5; n_L6 <= L6;   n_L7 <= L7;   n_L8 <= L8;
    n_L9 <= L9; n_L10 <= L10; n_L11 <= L11; n_L12 <= L12;
    n_L13 <= L13;
n_a<= a_net; n_b<= b_net; n_c<= c_net; n_d<= d_net;

if ((L1 and u) = '1') then
 n_L1 <= '0'; n_a<= '1'; n_b<= '1'; n_L9 <= '1';
end if;
if ((L9 and not u) = '1') then
      n_L9 <= '0'; n_L2 <= '1'; n_L3 <= '1';
end if;
if ((L2 and not v and w) = '1') then
n_L2 <= '0'; n_b<= '0';n_c<= '1'; n_L10 <= '1';
end if;
if ((L10 and not w) = '1') then
n_L10 <= '0'; n_b<= '1'; n_L11 <= '1';
end if;
if ((L11) = '1') then
n_L11 <= '0'; n_c<= '0'; n_L2 <= '1';

end if;
if ((L2 and v) = '1') then
```

```
n_L2 <= '0'; n_a<= '0'; n_c<= '1'; n_L4 <= '1'; n_L5 <= '1';
end if;
if ((L3 and u and w) = '1') then
n_L3 <= '0'; n_d<= '1'; n_L6 <= '1';
end if;
if ((L4 and not u and not v) = '1') then
n_L4 <= '0'; n_a<= '1'; n_L12 <= '1';
end if;
 if ((L4 and u) = '1') then
n_L4 <= '0'; n_a<= '1'; n_b<= '0'; n_L7 <= '1';
end if;
if ((L12 and u) = '1') then
n_L12 <= '0'; n_a<= '0'; n_L4 <= '1';

end if;
if ((L5 and not v and w) = '1') then
n_L5 <= '0'; n_c<= '0'; n_L8 <= '1';
end if;
if ((L6 and L7 and L8) = '1') then
n_L6 <= '0'; n_L7 <= '0'; n_L8 <= '0'; n_a<= '0'; n_d<= '0';
n_L13 <= '1';
end if;
if ((L13 and not w) = '1') then
n_L13 <= '0'; n_L1 <= '1';

end if;
end process p1;

p2: process (clk, rst)
begin
if (rst = '1') then
a_net<= '0'; b_net<= '0'; c_net<= '0'; d_net<= '0';
      L1 <= '1'; L2 <= '0'; L3 <= '0'; L4 <= '0';
      L5 <= '0'; L6 <= '0'; L7 <= '0'; L8 <= '0';
      L9 <= '0'; L10 <= '0'; L11 <= '0'; L12 <= '0';
      L13 <= '0';
elsif (rst = '0') then
if (clk = '1' and clk'event) then
a_net<= n_a; b_net<= n_b; c_net<= n_c; d_net<= n_d;
      L1 <= n_L1; L2 <= n_L2; L3 <= n_L3;
      L4 <= n_L4; L5 <= n_L5; L6 <= n_L6;
      L7 <= n_L7; L8 <= n_L8; L9 <= n_L9;
      L10 <= n_L10; L11 <= n_L11; L12 <= n_L12;
      L13 <= n_L13;
end if;
end if;
end process p2;
end behavior;
```

СПИСОК ЛИТЕРАТУРЫ

1. *Поляков А.К.* Языки VHDL и VERILOG в проектировании цифровой аппаратуры. М.: СОЛОН-Пресс, 2003. 320 с.
2. *Бибило П.Н., Авдеев Н.А.* Моделирование и верификация цифровых систем на языке VHDL. М.: ЛЕНАНД, 2017. 344 с.
3. *Хаханов В.И., Хаханова И.В., Литвинова Е.И., Гузь О.А.* Проектирование и верификация цифровых систем

на кристаллах. Verilog & SystemVerilog. Харьков: ХНУРЭ, 2010. 528 с.

4. SystemVerilog 3.1a. Language Reference Manual. http://www.ece.uah.edu/~gaede/cpe526/SystemVerilog_3.1a.pdf
5. *Eisner C., Fismat D.* A Practical Introduction to PSL. N.Y., USA: Springer. 2006, 240 p.
6. *Чэнь М., Цинь К., Ку Х.-М., Мишра П.* Валидация на системном уровне. Высокоуровневое моделирование и управление тестированием. М.: Техносфера, 2014. 296 с.
7. *Янкин Ю.Ю., Шалыто А.А.* Разработка резервированного блока управления электроприводом на основе автоматного подхода // Научно-технический вестник информационных технологий, механики и оптики. 2014. Вып. 6(94). С. 146–152.
8. *Лохов А., Рабоволюк А.* Комплексная функциональная верификация СБИС. Система Questa компании MentorGraphics // Электроника: наука, технология, бизнес. 2007. № 3. С. 102–109.
9. *Закревский А.Д.* Параллельные алгоритмы логического управления. Мн.: Ин-т техн. кибернетики НАН Беларуси. 1999, 202 с.
10. *Закревский А.Д., Поттосин Ю.В., Черемисинова Л.Д.* Логические основы проектирования дискретных устройств. М.: Физматлит. 2007, 589 с.
11. *Бурдонов И.Б., Косачев А.С., Пономаренко В.Н., Шнитман В.З.* Обзор подходов к верификации распределенных систем. М.: Институт системного программирования РАН. 2003. Препринт № 16. 51 с. http://www.ispras.ru/preprints/docs/prep_16_2006.pdf.
12. *Бибило П.Н., Романов В.И.* Построение компактных тестов для функциональной верификации VHDL-описаний конечных автоматов // Управляющие системы и машины. 2017. № 1. С. 35–45.
13. *Витязь К.А., Романов В.И.* Алгоритмы построения функциональных тестов для цифровой схемы на основе автоматной модели ее поведения. Танаевские чтения // Доклады Восьмой Международной научной конференции (27 марта–30 марта 2018 г., Минск). Минск: ОИПИ НАН Беларуси. 2018. С. 52–56.
14. *Бибило П.Н.* Системы проектирования интегральных схем на основе языка VHDL. StateCAD, ModelSim, LeonardoSpectrum. М.: СОЛОН-Пресс, 2005. 384 с.
15. *Шлее М.* Qt 5.3. Профессиональное программирование на C++. СПб.: БХВ-Петербург. 2015, 928 с.
16. *Камкин А.С.* Проецирование систем переходов: преодоление комбинаторного взрыва при верификации параллельных систем // Программирование. 2015. № 6. С. 53–71.