ТЕОРЕТИЧЕСКИЙ И ПРИКЛАДНОЙ НАУЧНО-ТЕХНИЧЕСКИЙ ЖУРНАЛ

Издается с сентября 2010 г. DOI 10.17587/issn.2220-3397 ISSN 2220-3397

Журнал издается при поддержке Отделения математических наук РАН, Отделения нанотехнологий и информационных технологий РАН, МГУ имени М.В. Ломоносова, МГТУ имени Н.Э. Баумана

СОДЕРЖАНИЕ

Журнал зарегистрирован в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций. Свидетельство о регистрации ПИ № ФС77-38590 от 24 декабря 2009 г.

Журнал распространяется по подписке, которую можно оформить в любом почтовом отделении (индексы: по каталогу агентства "Роспечать" — **22765**, по Объединенному каталогу "Пресса России" — **39795**) или непосредственно в редакции.
Тел.: (499) 269-53-97. Факс: (499) 269-55-10.
Http://novtex.ru/prin/rus E-mail: prin@novtex.ru
Журнал включен в систему Российского индекса научного цитирования.
Журнал входит в Перечень научных журналов, в которых по рекомендации ВАК РФ должны быть опубликованы научные результаты диссертаций на соискание ученой степени доктора и кандидата наук.

THEORETICAL AND APPLIED SCIENTIFIC AND TECHNICAL JOURNAL

SOFTWARE ENGINEERING

PROGRAMMNAYA INGENERIA

Vol. 8
N 1
2017

Published since September 2010 **DOI 10.17587/issn.2220-3397** **ISSN 2220-3397**

CONTENTS

Information about the jornal is available online at:
http://novtex.ru/prin/eng e-mail:prin@novtex.ru

УДК 004.4′233 DOI: 10.17587/prin.8.3-6

В. А. Галатенко, д-р физ.-мат. наук, ст. науч. сотр., зав. сектором, galat@niisi.ras.ru,
К. А. Костюхин, канд. физ.-мат. наук, ст. науч. сотр., kost@niisi.ras.ru,
М. Д. Дзабраев, инженер, mdzabraev@niisi.ras.ru, Федеральное государственное учреждение "Федеральный научный центр Научно-исследовательский институт системных исследований Российской академии наук", Москва

Применение протокола RSP в рамках концепции контролируемого выполнения встраиваемых приложений

Стандартизация и унификация интерфейсов и протоколов интерактивной отладки и мониторинга встраиваемых систем позволяют применять широкий спектр средств контролируемого выполнения ими прикладных задач. В статье рассмотрен протокол интерактивной отладки, de facto являющийся стандартом при создании распределенных средств отладки, а также его адаптация и расширения для отладки встраиваемых систем и систем реального времени.

Ключевые слова: *интерактивная отладка, мониторинг, RSP, встраиваемые системы, системы реального времени*

Введение

Для интерактивной удаленной отладки встраиваемых систем наиболее часто [1] используют отладчик GDB (GNU Debugger, http://www.gnu.org/software/gdb/). Такой выбор объясняется отсутствием необходимости оплаты, открытостью и масштабируемостью данного отладчика. Кроме того, GDB использует специально разработанный протокол взаимодействия с удаленной целевой системой, позволяющий проводить отладочные действия практически через любой коммуникационный интерфейс (сетевой кабель, последовательная линия и т. п.). Этот протокол получил название Remote Serial Protocol (RSP, протокол удаленной последовательной отладки).

На настоящее время существует широкий спектр встраиваемых систем, поддерживающих RSP и, таким образом, позволяющих использовать как GDB, так и любое другое средство, реализующее этот протокол. Следует отметить, что RSP можно использовать не только для интерактивной отладки, но и для мониторинга целевой системы. Простота реализации как самого протокола RSP, так и его расширений позволяет гибко настраивать целевую систему и средства ее отладки в рамках концепции контролируемого выполнения.

В настоящей статье проанализированы базовые функциональные возможности протокола RSP, а также его расширение, созданное авторами в рамках работ по реализации концепции контролируемого выполнения целевой системой приложений, выполняющихся в среде операционной системы реального времени (ОСРВ) Багет [2].

Описание протокола RSP

Remote Serial Protocol (далее RSP) представляет собой протокол для обмена информацией между клиентом отладки (например, GDB) и удаленным сервером отладки, который располагается на целевой системе. В задачи сервера отладки входит прием запросов от клиента, обработка принятых данных и отправка ответа клиенту.

В рамках протокола RSP инициатором обмена информацией всегда является клиент. Протоколом не предусмотрено, чтобы сервер самостоятельно отправлял данные клиенту.

В контексте RSP запросы и ответы называют пакетами. Общая структура пакета выглядит следующим образом. Пакет начинается с символа ‘$’, далее идет содержимое пакета, конец пакета помечается символом ‘#’. Затем следует контрольная сумма пакета, представленная числом от 0 до 255.

Таким образом, каждый пакет выглядит так:

$тело_пакета#контрольная_сумма

Тело пакета и контрольная сумма представляются ASCII-символами. Например, чтобы прочитать один байт по адресу 0x7ffff7ddb2d0 клиент сформирует пакет вида

$m7ffff7ddb2d0,1#c0

Здесь первая часть пакета — это адрес, затем следует разделитель (запятая) и после этого — размер в байтах фрагмента памяти целевой ЭВМ, содержимое которого нужно прочитать. Два символа

в конце — это шестнадцатеричное представление контрольной суммы пакета.

На такой запрос сервер отладки отвечает пакетом следующего формата:

$48#6c

Тело этого пакета — запрошенный байт. В случае если запрошенный адрес прочитать нельзя, то агент отладки возвратит пакет, сигнализирующий об ошибке:

$E01#a6

В данном случае "01" — это код ошибки. Также существуют пакеты для записи указанных байтов в память, пакеты для чтения и записи регистров, установки/удаления точек останова и т. д. Таким образом, с помощью этого протокола можно реализовать простейший сеанс интерактивной отладки.

Реализация протокола в формате обмена текстовыми строками позволяет адаптировать его под нужды разработчика с минимальными затратами. В частности, можно вручную отлаживать реализацию сервера отладки, просто посылая ASCII-строки через открытое соединение. Следует отметить, что простая структура протокола позволяет его легко модифицировать и создавать новые пакеты. Кроме того, RSP хорошо документирован. Документация по этому протоколу находится в свободном доступе по адресу https://sourceware.org/gdb/onlinedocs/gdb/Remote-Protocol.html. Также следует отметить, что на настоящее время многие встроенные системы поддерживают RSP.

Существенным недостатком RSP является то обстоятельство, что он ориентирован на отладку одного процесса. Выбор процесса отладки находится за рамками RSP. Обычно к моменту начала сеанса отладки сервер уже "знает", какой процесс будет отлаживаться. Для выбора процесса в ходе отладки потребуется расширение протокола. Для эффективной отладки многопроцессных систем и приложений необходимо уметь группировать потоки и процессы целевой системы. При этом для групп необходимо уметь задавать групповые операции (например, запуск, останов). К сожалению, ни возможностей группировки, ни групповых операций RSP на сегодня не предоставляет.

Описание целевой системы

Авторы статьи работают с 64-разрядными процессорами семейства MIPS, выполняющимися под управлением отечественной ОСРВ Багет [2]. К подобным системам предъявляется ряд требований. Например, все процессы ОС задаются на стадии конфигурации, до запуска ОС. После того как система начала выполнение, новые процессы создавать нельзя. В ОСРВ Багет процессы бывают двух типов: POSIX и ARINC [2]. Процессы ARINC описываются стандартом ARINC 653.

Стандарт ARINC 653 разрабатывался для авиационной промышленности. Стандарт описывает целевую систему на уровне независимых разделов (*partitions*). В каждом разделе исполняются или ARINC-, или POSIX-процессы. Процессы из разных разделов могут взаимодействовать только с помощью портов и каналов, соединяющих эти порты. Все каналы и порты задаются на стадии конфигурации ОС.

Опыт реализации RSP

Для внедрения RSP со стороны ОСРВ Багет был создан сервер отладки. На пользовательской стороне используется интерактивный отладчик GDB, в котором был осуществлен ряд модификаций, учитывающих особенности ОСРВ Багет.

Стандартный отладчик GDB не позволяет брать на отладку произвольный процесс при отладке по протоколу RSP. Чтобы преодолеть этот недостаток, в GDB были сделаны модификации, которые позволяют присоединиться к любому процессу ОСРВ Багет. В стандартном GDB сеанс отладки по RSP начинается с команды

target remote ip:port

После сделанных модификаций синтаксис данной команды был расширен:

target remote ip:port
target remote ip:port pid
target remote ip:port pid.tid

Последняя из приведенных команд осуществляет взятие на отладку потока с идентификатором **tid**, который выполняется в рамках процесса с идентификатором **pid**. Вторая команда выполняет взятие на отладку всех потоков процесса с идентификатором **pid**. Первая команда эквивалентна второй с **pid = 0**. Иными словами, первая команда осуществляет взятие на отладку всех потоков нулевого процесса (процесса ядра ОСРВ Багет).

Реализация удаленного сервера отладки базируется на использовании системного вызова **ptrace**. Системный вызов **ptrace** описан в стандарте POSIX. С его помощью можно реализовать следующие операции: взять поток на отладку; отсоединиться от потока; прочитать/изменить регистры и/или память целевого процессора; выполнить операции запуска потока (continue); выполнить шаг на одну ассемблерную команду и др.

Расширения RSP

Авторам статьи потребовалось реализовать расширение для трассировки памяти целевой ЭВМ с минимальными задержками по времени. Основная идея расширения заключается в том, что программист получает возможность протоколировать и изменять содержимое определенных участков памяти целевой ЭВМ. В параметрах соответствующего пакета можно задать периодичность изменения значений в этих участках памяти. Остановимся более подробнее на описании пакетов расширения протокола.

Сеанс трассировки начинается с проверки версии протокола. Для этого нужно отправить пакет **$.Version#xx**. Сервер отладки должен возвратить либо текущую версию протокола, либо ошибку EXX. Чтобы переключить сервер отладки на версию, например 1.1, нужно отправить пакет Version с аргументом:

$.Version;1.1#xx

Теперь опишем, как добавлять области памяти в трассу. Команда имеет следующий формат:

.Add;
<имя или адрес переменной>;
<размер в байтах>;
<raw>;
<флаг изменения>;
<значение переменной>;
<pid>"

После первого разделителя "**;**" нужно указать имя или адрес переменной. Затем указывается размер области памяти, которая будет взята на трассировку. Поле format на данный момент может принимать только значение raw. Далее идет флаг изменения. Он может принимать два значения: 1 или 0. Если флаг выставлен в 1, то сервер отладки будет брать поле **<значение переменной>** и через заданный промежуток времени инициализировать область памяти этим значением. Если **<флаг изменения>** выставлен в 0, то через заданный промежуток времени сервер будет сохранять значение этой переменной во внутренний буфер. Для того чтобы получить значение из внутреннего буфера, нужно отправить команду **.Get** (см. далее). И наконец, последний аргумент — это **<pid>**, идентификатор процесса, память которого необходимо брать на трассировку.

На практике описанные действия выглядят так, как приведено в табл. 1.

Еще один пример:

$.Add;0x10000490;32;raw;1;
ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff;2#00

В данном примере область памяти будет инициализироваться заданным значением. У процесса с pid, равным 2, область памяти с началом в 0x10000490 и длиной 32 байта будет через установленный промежуток времени инициализироваться единичными битами.

После того как все области памяти добавлены, нужно запустить трассировку. Это осуществляется с помощью команды **$.Start#00+**.

Чтобы получить значения всех областей памяти, взятых на трассировку, нужно отправить пакет **$.Get#00**. Ответ на эту команду имеет следующий формат:

S:S:...S:,

где **S** — строка, которая имеет вид

S = <pid>;<имя блока памяти или адрес>;<размер>;
<тип>;<содержимое памяти>

Каждая из строк **S** описывает одну из добавленных областей памяти.

Команду **.Get** можно послать с аргументами:

.Get;<имя блока памяти или адрес>;<raw>;<pid>

Эта команда предназначена для чтения одного блока памяти. Ответом на такую команду будет строка **S:**, формат которой описан выше.

Если область памяти больше не нуждается в трассировке, тогда ее можно удалить из-под контроля сервера отладки с помощью пакета **.Del**. Отметим, что перед отправкой пакета **.Del** необходимо остановить трассировку командой **$.Stop#00**. Команда **.Del** имеет следующий формат:

.Del;<имя или адрес переменной>;<pid>

Существует также запрос для единоразового изменения области памяти. Данный запрос имеет следующий формат:

.Change;<имя или адрес переменной>;
<raw>;<значение переменной>;<pid>

Перед тем как отправлять запрос типа **.Change**, необходимо добавить область памяти на трассировку с помощью пакета **.Add**.

Для полного удаления списка трассируемых переменных нужно отправить команду **$.Clear#00**.

Ранее в статье отмечался промежуток времени, через который будет происходить сохранение трассируемых областей памяти во внутренний буфер сервера отладки или инициализация заданных участков памяти определенным значением. Этот параметр можно менять с помощью отправки пакета **.Time;<время в миллисекундах>**.

Полный список кодов ошибок, которые может возвращать сервер отладки, представлен в табл. 2.

Таблица 1

Запрос на трассировку области памяти

Запрос клиента	Ответ сервера	Комментарий
$.Stop#00	$E03#a8	E03 свидетельствует о том, что трассировка не запущена
$.Version;1.1#00	OK	—
$.Add;0x10000000;4;raw;0;;2#00	OK	Запрос на трассировку области памяти, которая начинается с адреса 0x10000000 и имеет длину 4 байта. Далее идет поле raw, затем флаг изменения — 0, <значение переменной> — пустое поле и, наконец, <pid> процесса

Таблица 2

Коды ошибок

Описание ошибки	Код ошибки
Переменная не найдена	2
Трассировка уже остановлена	3
Трассировка уже запущена	4
Динамическая память не может быть выделена	5
Переменная уже добавлена на трассировку	6
Достигнуто максимальное число одновременно трассируемых переменных	7
Некорректный формат запроса	8
Временной период трассировки меньше допустимого значения	9
Некорректный идентификатор процесса	10
Внутренняя ошибка сервера отладки	11

Заключение

Использование открытых систем и стандартов позволяет осуществлять интеграцию новых компонентов в отечественные аппаратно-программные комплексы, разрабатываемые для решения ответственных задач, в частности, задач реального времени.

Рассмотренный в настоящей статье протокол RSP, а также его расширение, позволяют перейти от интерактивной отладки, требующей обязательного останова целевого процесса для проведения отладочных действий, к мониторингу и сбору трассы в ходе его (процесса) выполнения. Это расширение внедрено в комплекс контролируемого выполнения целевой системой встраиваемых приложений, который разработан в НИИСИ РАН и используется вместе с компонентами интерактивной отладки и мониторинга.

Список литературы

1. **Gatliff B.** Embedding with GNU: GNU Debugger, Embedded System Programming, September 1999, pp. 80—94.

2. **Годунов А. Н., Солдатов В. А.** Спецификация ARINC 653 и ее реализация в операционной системе реального времени Багет 3 // Программная инженерия. 2015. № 6. С. 3—17.

RSP Implementaion for the Purposes of the Embedded Applications Controlled Execution

V. A. Galatenko, galat@niisi.ras.ru, **K. A. Kostiukhin,** kost@niisi.ras.ru, **M. D. Dzabraev,** mdzabraev@niisi.ras.ru, Federal State Institution "Scientific Research Institute of System Analysis of the Russian Academy of Science"

Corresponding author:

Kostiukhin Konstantin A., Senior Research Fellow, Federal State Institution "Scientific Research Institute of System Analysis of the Russian Academy of Science", Moscow, 117218, Russian Federation
E-mail: kost@niisi.ras.ru

Received on October 28, 2016
Accepted on November 03, 2016

The article discusses the interactive debugging protocol (remote serial protocol, RSP), which is de facto standard of the distributed debugging tools.

At present, there is a wide range of embedded systems supporting the RSP and thus it makes possible to use RSP and one of it standard client (i.e GDB), as well as any other tools. It should be noted that RSP can be used not only for interactive debugging and monitoring purposes. It's easy to implement the protocol itself to build it into a new embedded system, and to create some extensions that enable flexible configuration of the target system and implementation of the embedded applications controlled execution.

In this article authors will examine the basic functionality of the RSP protocol, as well as its extension, created by the authors as a part of the work on the implementation of the concept of embedded and real-time systems controlled execution.

Keywords: *interactive debugging, monitoring, RSP, embedded systems, real-time systems*

For citation:

Galatenko V. A., Kostiukhin K. A., Dzabraev M. D. RSP Implementaion for the Purposes of the Embedded Applications Controlled Execution, *Programmnaya Ingeneria*, 2017, vol. 8, no 1, pp. 3—6.

DOI: 10.17587/prin.8.3-6

References

1. **Gatliff B.** Embedding with GNU: GNU Debugger, Embedded System Programming, September 1999, pp. 80—94.

2. **Godunov A. N., Soldatov V. A.** Specifikacija ARINC 653 i ee realizacija v operacionnoj sisteme real'nogo vremeni Baget 3 (ARINC 653 specification implementation in the real-time system Baget 3), *Programmnaya Ingeneria*, 2015, no. 6, pp. 3—17 (in Russian).

УДК 621.382 DOI: 10.17587/prin.8.7-15

П. Н. Бибило, д-р техн. наук, проф., зав. лаб., **В. И. Романов**, канд. техн. наук, доц., вед. науч. сотр., e-mail: bibilo@newman.bas-net.by, Объединенный институт проблем информатики Национальной академии наук Беларуси, г. Минск

Нахождение энергоемких тестов для логических схем, реализующих конечные автоматы

Предлагается проводить быструю оценку энергопотребления логических схем, реализующих конечные автоматы, используя результаты моделирования как структурных описаний логических схем, так и алгоритмических описаний, по которым строятся логические схемы. Применение предлагаемой методики получения энергоемких тестов позволяет примерно на 40 % повысить оценку энергопотребления при моделировании структурных описаний схем на найденных тестах по сравнению с использованием исходных тестов.

Ключевые слова: *конечные автоматы, логическая схема, КМОП-технология, энергопотребление, тесты, логическое моделирование*

Введение

Сокращение энергопотребления цифровых блоков является одной из важнейших задач, возникающих при проектировании СБИС, выполняемых по КМОП-технологии [1, 2]. Одним из аспектов решения этой задачи является оценка энергопотребления логических схем. Достаточно точная (но весьма трудоемкая) оценка энергопотребления может быть получена в системах схемотехнического моделирования на уровне транзисторных описаний таких схем.

Однако возрастание размерностей задачи синтеза схем и разнообразие вариантов их реализаций приводит к необходимости разработки средств для быстрой оценки энергопотребления с приемлемой точностью (погрешностью). Ускорение оценки энергопотребления может быть достигнуто при использовании систем логического моделирования, что требует построения соответствующих описаний логических элементов, позволяющих учитывать их переключательную активность [3, 4] и потребляемую мощность. Описание способа, позволяющего характеризовать логические КМОП-элементы по параметру их энергопотребления, и примеры соответствующего расширения функциональных возможностей описания библиотечного элемента на языке VHDL представлены в работе [5]. Расширение таких возможностей VHDL-описания элемента ориентировано на учет энергопотребления элемента при его срабатывании, т. е. при изменении состояния хотя бы одного из выходных полюсов элемента. В этой же работе приведены экспериментальные результаты быстрой оценки энергопотребления комбинационных КМОП-схем на различных видах тестов.

В данной работе предложена и проанализирована методика быстрой оценки энергопотребления синхронных логических схем, реализующих конечные автоматы. Методика базируется на предложенных в работе [5] функциональных VHDL-моделях КМОП-элементов и использовании программных систем логического VHDL-моделирования, например, системы ModelSim [6]. Оценка верхней границы энергопотребления сводится к нахождению тестов, вызывающих повышенное энергопотребление. Такие тесты названы *энергоемкими*.

Основные этапы методики оценки энергопотребления

Исходными данными для оценки энергопотребления являются два VHDL-описания: исходное алгоритмическое описание конечного автомата и структурное описание (*netlist*) логической схемы из КМОП-элементов, функционально эквивалентное исходному описанию. Будем для краткости называть исходную VHDL-модель "алгоритмической", а вторую VHDL-модель — "структурной". Синхронные схемы синтезируются по исходным VHDL-описаниям конечных автоматов в базисе библиотечных КМОП-элементов с помощью системы синтеза (синтезатора) *LeonardoSpectrum* [6].

Пример алгоритмической модели автомата *Mealy* представлен в следующем листинге:

```
library ieee;
use ieee.std _ logic _ 1164.all;
entity Mealy is
port(x1, x2, x3, x4, clk, rst: in std _ logic;
    y1, y2, y3, y4, y5, y6: out std _ logic);
end Mealy;
architecture rtl of Mealy is
type T _ state is (s1, s2, s3, s4, s5, s6);
signal NEXT _ state, state: T _ state;
signal w: std _ logic _ vector (1 to 6);
begin
y1 <= w(1); y2 <= w(2); y3 <= w(3);
y4 <= w(4); y5 <= w(5); y6 <= w(6);
NS : process (state, x1, x2, x3, x4)
begin
case state is
when s1 =>
NEXT _ state <= s2; w <= "010000";
when s2 =>
if (x1 and not x2 and not x3) = '1'
          then NEXT _ state <= s3; w <= "001000";
    elsif (x1 and x2) = '1'
          then NEXT _ state <= s2; w <= "010000";
elsif ((x1 and not x2 and x3) = '1')
then NEXT _ state <= s4; w <= "000100";
elsif (not x1 = '1')
then NEXT _ state <= s5; w <= "000010";
end if;
when s3 => NEXT _ state <= s4; w <= "000100";
when s4 => if (not x2 = '1')
then NEXT _ state <= s1; w <= "100000";
elsif (x2 = '1')
then NEXT _ state <= s6; w <= "000001";
end if;
when s5 =>
    if ((not x1 and x4) = '1')
then NEXT _ state <= s1; w <= "100000";
elsif ((not x4) or (x1 and x4)) = '1'
          then NEXT _ state <= s4; w <= "000100";
end if;
when s6 => NEXT _ state <= s1; w <= "100000";
end case;
end process NS;
state <= s1 when rst = '1' else
NEXT _ state when (clk'event and clk = '1') else state;
end rtl;
```

Автомат *Mealy* имеет асинхронный сброс `rst` в начальное состояние `s1`, состояния автомата переключаются по переднему фронту сигнала `clk` синхронизации. Реализующая автомат логическая схема содержит 38 логических КМОП-элементов, в том числе три синхронных триггера для хранения закодированных по коду Грея внутренних состояний.

Применение предлагаемой методики основывается на расширении функциональных возможностей VHDL-описаний логических элементов, составляющих целевую библиотеку проектирования. Модификация VHDL-описания логического элемента осуществляется в целях проведения оценки его энергопотребления [5], выполняется только один раз и касается библиотеки элементов, а не конкретной логической схемы, реализующей автомат. Применять библиотеку модифицированных VHDL-описаний КМОП-элементов можно для оценки логических схем, полученных по произвольным синтезируемым VHDL-описаниям цифрового блока СБИС.

Методика оценки энергопотребления состоит из перечисленных далее шести этапов. Результатом применения методики является энергоемкий тест $T_э$.

Этап 1. Подготовка *теста* T — последовательности входных тестирующих наборов — воздействий, подаваемых (при моделировании) на вход как алгоритмической модели, так и структурной модели автомата (рис. 1). Подготовку теста в форме текстового файла осуществляет обычно специальная программа — генератор тестов. Заметим, что далее в экспериментах будут использоваться только псевдослучайные тесты с равномерным распределением вероятностей появления нулей и единиц в тестирующих наборах.

Этап 2. Подготовка специальной тестирующей программы для алгоритмической модели. Такая программа должна считывать тест и записывать состояния автомата (проходимые по тактам) в текстовый файл. Чтобы обеспечить выполнение таких функций система моделирования должна иметь возможность работы со стандартом VHDL'2008 [7].

Этап 3. Выполнение VHDL-моделирования в системе *Questa Sim* для алгоритмической модели. Результатом такого моделирования является последовательность S (текстовый файл) внутренних состояний, соответствующих каждому из тактов функционирования автомата. В нулевом такте моделирования осуществляется подача на вход модели сигнала асинхронного сброса, который устанавливает VHDL-модель в начальное состояние. Каждый последующий такт моделирования включает:

- подачу входного тестового набора;
- переключение сигнала синхронизации в единичное состояние (формирование переднего фронта синхросигнала);
- переключение синхросигнала в нулевое состояние (формирование заднего фронта синхросигнала).

Этап 4. Выполнение моделирования в системе *Questa Sim* для структурной модели с тестом T. Результатом такого моделирования является текстовый файл, содержащий последовательность значений энергопотребления E, каждое из которых соответствует одному такту моделирования VHDL-описания логической схемы. По последовательности E легко получить среднее энергопотребление схемы на начальном тесте T.

Заметим, что энергопотребление в каждом такте складывается из потребления элементами схемы как при подаче тестового набора, так и при изменении синхросигнала из 0 в 1 (и обратно) (см. этап 3). В текстовый файл E для каждой пары (такта) сменяемых значений входных воздействий записывается суммарное значение энергии, потребляемой схемой в данном такте.

Таким образом, начальное моделирование позволяет получить для каждой пары тестовых воздействий (тестовых векторов — двоичных входных наборов) некоторое число, определяющее общее (динамическое и статическое) потребление, возникающее в результате переключений транзисторов, входящих в элементы синхронной схемы, реализующей конечный автомат.

Этап 5. Формирование энергоемкого теста $T_э$ по последовательностям T, S, E, полученным в результате начального моделирования. Алгоритм формирования энергоемкого теста будет описан далее.

Этап 6. Выполнение моделирования структурного описания схемы на энергоемком тесте $T_э$ в целях оценки среднего энергопотребления схемы, функционирующей в режиме повышенного энергопотребления.

Для проверки правильности функционирования логической схемы автомата на этапах 3, 4 можно формировать и записывать в соответствующие текстовые файлы значения выходов (реакций) автомата в каждом такте. Несовпадение (в соответствующих тактах) выходных значений сигналов алгоритмической модели и структурной модели (логической схемы) будет свидетельствовать о том, что этап синтеза схемы выполнен неправильно.

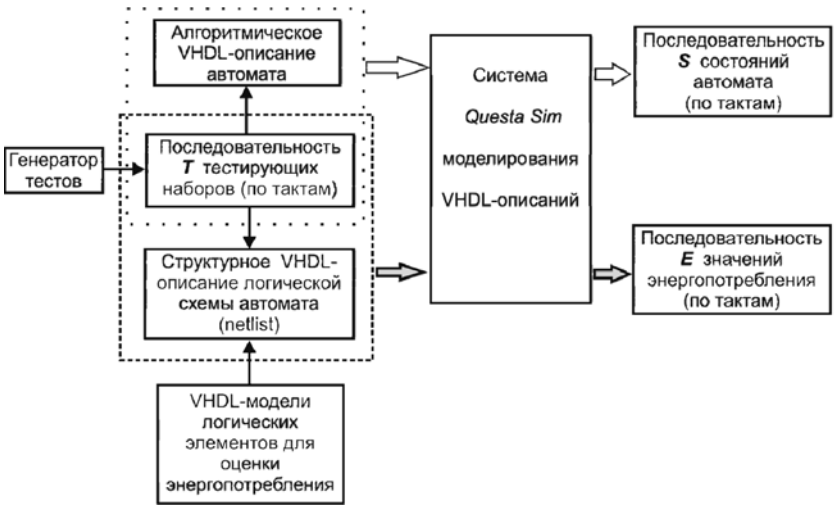

Рис. 1. Моделирование алгоритмического описания конечного автомата и логической схемы на одном и том же тесте

Алгоритм построения энергоемкого теста заданной длины по результатам моделирования

Исходными данными алгоритма являются потактовые последовательности тестирующих наборов T, проходимых состояний S и значений энергопотребления E. Дополнительно указывается имя S_1 начального состояния на графе переходов автомата и минимальная длина искомого цикла L, выступающая в качестве параметра управления алгоритмом. Для нулевого такта моделирования предполагается, что автомат находится в начальном состоянии S_1, для этого состояния не указывается

тестирующий набор, по которому автомат попадает в это состояние, и не указывается значение энергопотребления, поэтому длина последовательности S на единицу больше длин последовательностей T, E. Обозначим $n = |S| - 1$.

Идея алгоритма состоит в том, чтобы найти в последовательности S (пути в графе переходов автомата G) цикл L, который характеризуется максимальным средним значением энергопотребления, а затем найти путь от начального состояния к одной из вершин найденного цикла. В результирующем тесте $T_э$ тестовые наборы из L повторяются m-кратно, причем m можно задать как параметр. Алгоритм представляет три последовательно проходимых этапа.

Первый из этапов предназначен для устранения из данных информации, касающейся многократного прохождения петель на графе переходов автомата. Формально это означает, что в последовательностях T, S, E будут ликвидированы повторы "строк", т. е. удалены i-е элементы последовательностей, для которых $t_i = t_{i+1}$, $s_i = s_{i+1}$, $e_i = e_{i+1}$, $i = \overline{1, n-1}$. Возможность удаления повторов обусловливается тем фактом, что переключения транзисторов и узлов КМОП-схемы, выполняемые без смены состояния автомата, после многократной подачи одного и того же тестового набора вызывают одинаковое и незначительное энергопотребление, связанное только с изменением значений синхросигнала.

Второй этап алгоритма предназначен для определения наиболее энергоемкого цикла, при этом выполняется следующая последовательность шагов.

Шаг 1. Установка в нуль начальных значений позиций начала (p_{begin}) и конца (p_{end}) искомого цикла, а также оценка его среднего энергопотребления ($Av_{\max}$).

Шаг 2. Последовательный перебор элементов $s_i \in S$, $i = \overline{1, n}$, $n = |S| - 1$.

Шаг 3. Проверка для каждого элемента последовательности, была ли в данном такте моделирования смена состояний, что выражается формулой $s_i \neq s_{i+1}$. Если смены состояний не было, то переход алгоритма к анализу следующего $(i + 1)$-го элемента последовательности состояний S.

Шаг 4. В противном случае реализация поиска ближайшего упоминания в S имени s_i, т. е. индекса j, удовлетворяющего условию $s_i = s_j$, причем $(j - i) \geqslant |L|$ и имя s_i отсутствует в последовательности $s_{i+1}, \ldots, s_{j-1}$.

Шаг 5. Если анализируемое состояние s_i в последовательности больше не встречается, переход на следующую итерацию основного цикла $i = i + 1$.

Шаг 6. Иначе оценка среднего энергопотребления найденного цикла по формуле

$$Av_i = \frac{\sum_{k=i+1}^{j} e_k}{j - i}, \quad e_k \in E.$$

Если $Av_i > Av_{\max}$, то фиксация данных найденного цикла в наборе p_{begin}, p_{end}, $Av_{\max}$.

Третий этап алгоритма связан с построением итогового теста, он начинается после завершения основного цикла перебора элементов последовательности S состояний.

Шаг 1. Проверка найденной оценки $Av_{\max}$. Если в ней оказывается исходное нулевое значение, то это соответствует ситуации отсутствия циклов длины не меньше $|L|$ — построение энергоемкого теста $T_э$ в заданных условиях невозможно. Иначе реализуется процедура подготовки энергоемкого теста $T_э$.

Шаг 2. Проверка, не является ли начальный элемент цикла состояний одновременно и начальным состоянием проведенного моделирования, т. е. выполняется ли условие $s_{p_{begin}} = s_1$.

Шаг 3. При соблюдении этого условия итоговый тест строится путем m-кратного повторения последовательности тестовых наборов $t_{p_{begin}+1}, t_{p_{begin}+2}, \ldots, t_{p_{end}}$. Так для $m = 3$ тест будет выглядеть следующим образом:

$$T_э = t_{p_{begin}+1}, \ldots, t_{p_{end}}, t_{p_{begin}+1},$$
$$t_{p_{begin}+2} \ldots, t_{p_{end}}, t_{p_{begin}+1}, t_{p_{begin}+2}, \ldots, t_{p_{end}}.$$

Шаг 4. Если $s_{p_{begin}} \neq s_0$, то поиск в последовательности состояний ближайшего вхождения (позиция p_1) начального состояния s_1, предшествующего позиции $s_{p_{begin}}$. Искомый тест строится в форме кортежа, содержащего в себе последовательность тестирующих наборов от позиции $p_1 + 1$ до начальной позиции цикла p_{begin}, дополненную m-кратным повторением самого цикла. При $m = 3$ тест $T_э$ будет выглядеть следующим образом:

$$t_{p_1+1}, \ldots, t_{p_{begin}}, t_{p_{begin}+1}, \ldots, t_{p_{end}},$$
$$t_{p_{begin}+1}, \ldots, t_{p_{end}}, t_{p_{begin}+1}, \ldots, t_{p_{end}}.$$

Примером, иллюстрирующим такой алгоритм, является HDL-описание автомата *Mealy* в представленном ранее листинге.

Смена состояний автомата *Mealy* осуществляется по тактам, начало такта отсчитывается от переднего фронта синхросигнала `clk`. Исходному VHDL-описанию соответствует граф G переходов внутренних состояний автомата (рис. 2). Заметим, что на графе G не показаны асинхронные переходы из

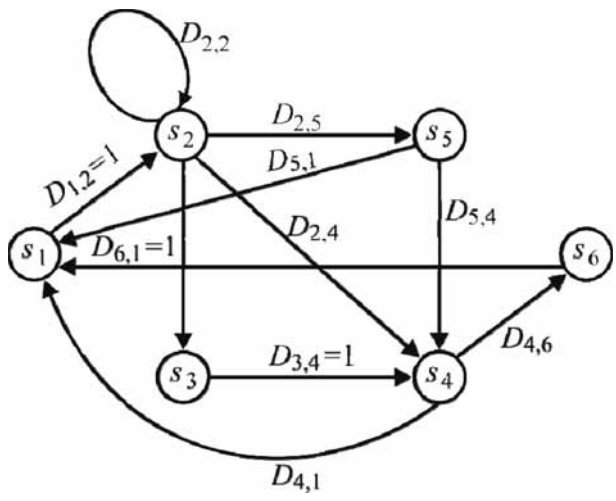

Рис. 2. Граф G переходов автомата *Mealy*

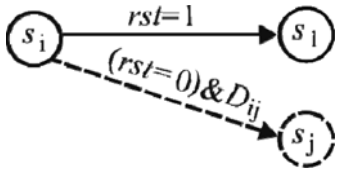

Рис. 3. Асинхронные переходы в начальное состояние s_1

любого состояния в начальное состояние s_1 при появлении единичного значения сигнала `rst` (рис. 3). Значения условий перехода на дугах графа переходов автомата заданы в виде дизъюнктивных нормальных форм булевых функций, зависящих от входных сигналов x_1, x_2, x_3, x_4 (табл. 1). Для каждого перехода (дуги) указывается также значение выходного сигнала автомата. Реализующая автомат логическая схема является синхронной, синхронизация в ней (так же как и в алгоритмической модели) осуществляется по переднему фронту синхросигнала `clk`.

Сформируем псевдослучайный тест T (табл. 2) из 24 входных наборов для вектора (x_1, x_2, x_3, x_4) входных переменных. Проведем моделирование для алгоритмической модели на данном тесте, получим последовательность S (второй столбец табл. 2), результаты моделирования структурного описания — это последовательность E значений энергопотребления для каждого такта. Исходным состоянием при моделировании является s_1.

Подача входных наборов теста T при моделировании алгоритмического описания автомата позволяет зафиксировать проходимые состояния автомата, об-

Таблица 1

Табличное задание конечного автомата *Mealy*

Текущее состояние (`state`)	Следующее состояние (`NEXT _ state`)	Условия перехода	Выходной сигнал
s_1	s_2	$D_{1,2} = 1$	y_2
s_2	s_2	$D_{2,2} = x_1x_2$	y_2
	s_3	$D_{2,3} = x_1\bar{x}_2\bar{x}_3$	y_3
	s_4	$D_{2,4} = x_1\bar{x}_2x_3$	y_4
	s_5	$D_{2,5} = \bar{x}_1$	y_5
s_3	s_4	$D_{3,4} = 1$	y_4
s_4	s_1	$D_{4,1} = \bar{x}_2$	y_1
	s_6	$D_{4,6} = x_2$	y_6
s_5	s_1	$D_{5,1} = \bar{x}_1x_4$	y_1
	s_4	$D_{5,4} = \bar{x}_4 \vee x_1x_4$	y_4
s_6	s_1	$D_{6,1} = 1$	y_1

Таблица 2

Результаты начального моделирования конечного автомата *Mealy* и реализующей его логической схемы

Такт	S	T x_1, x_2, x_3, x_4	E	Цикл 1	Цикл 2	Цикл 3	Цикл 4	Цикл 5
0	s_1	–						
1	s_2	1010	546					
2	s_4	1011	416					
3	s_1	0001	444	s_1				
4	s_2	1011	427	s_2				
5	s_4	1010	417	s_4				
6	s_1	1001	425	s_1	s_1			
7	s_2	0110	470		s_2			
8	s_2	1111	485		s_2			
9	s_5	0110	740		s_5			
10	s_1	0101	653		s_1	s_1		
11	s_2	1100	313			s_2		
12	s_5	0100	748			s_5		
13	s_4	0100	362			s_4		
14	s_1	0001	712			s_1	s_1	
15	s_2	1110	352				s_2	
16	s_2	1110	232				s_2	
17	s_4	1010	630				s_4	
18	s_1	0001	443				s_1	s_1
19	s_2	0011	432					s_2
20	s_2	1101	547					s_2
21	s_3	1001	427					s_3
22	s_4	0110	453					s_4
23	s_6	0100	502					s_6
24	s_1	0001	514					s_1
Тест для цикла				1011 1010 1001	0110 1111 0110 0101	1100 0100 0100 0001	1110 1110 1010 0001	0011 1101 1001 0110 0100 0001
Среднее энергопотребление, условные единицы				428	587	534	525	474

разующие последовательность S. Каждому переходу (дуге графа G) соответствует значение потребления в файле E потреблений и значение входного тестового вектора в последовательности T. Строке с номером 0 в табл. 2 соответствует состояние s_1. Для перехода из состояния s_1 в состояние s_2 (в строку с номером 1) требуется подать входной набор 1010, который вызовет переключение узлов логической схемы и которому соответствует потребление 546 условных единиц. На графе G данному переходу между состояниями автомата соответствует дуга $s_1 \to s_2$. Второй строке табл. 2 соответствует дуга $s_2 \to s_4$ при входном наборе 1011, переключение узлов логической схемы вызовет энергопотребление 416 условных единиц и т. д. Последовательность S проходимых состояний автомата

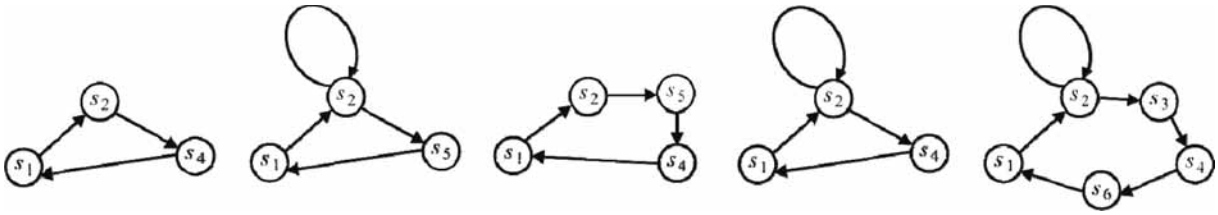

Рис. 4. Циклы, содержащиеся в последовательности *S* (табл. 2)

содержит пять циклов (рис. 4), они также приведены в табл. 2. Формирование энергоемкого теста заданной длины связано с нахождением в последовательности *S* цикла с максимальным средним значением энергопотребления.

Будем искать циклы, содержащие не менее четырех состояний, начальным состоянием в которых является s_1. Подсчитаем среднее энергопотребление по каждому из циклов:

Цикл 1: (427 + 417 + 425) : 3 = 423 условных единиц.

Цикл 2: (470 + 485 + 740 + 653) : 4 = 587 условных единиц. Обратим внимание на то, что тестовый набор 0110 оставляет автомат в том же самом состоянии s_2. Заметим, что при псевдослучайном характере входных наборов автомат может оставаться в одном и том же состоянии несколько (много) тактов. Поэтому повторяющиеся входные наборы, оставляющие автомат в том же состоянии и дающие одинаковые значения энергопотребления, следует исключать из теста, формируемого для соответствующего цикла. В формируемом тесте последовательность тестовых наборов, вызывающая повторение цикла, повторяется *m* раз, т. е. автомат проходит *m* раз выбранный цикл, считающийся самым энергоемким.

Цикл 3: (313 + 748 + 362 + 712) : 4 = 534 условных единиц.

Цикл 4: (232 + 630 + 443) : 3 = 435 условных единиц.

Цикл 5: (432 + 547 + 427 + 453 + 502 + 514): 6 = 479 условных единиц.

Таким образом, решением являются четыре тестирующих набора второго цикла, обеспечивающие максимальное среднее энергопотребление 587 условных единиц, при этом пиковое энергопотребление составляет 740 условных единиц.

Результаты моделирования для теста $T_э$ и оценка среднего энергопотребления на энергоемком тесте $T_э$ приведены в табл. 3. Среднее энергопотребление на тесте $T_э$ длины 4 составляет

(433 + 485 + 740 + 653) : 4 = 577,75 условных единиц.

В данном примере средняя оценка энергопотребления 577,75 условных единиц для теста $T_э$ оказалась немного ниже ожидаемой (587). Это связано с тем, что при начальном моделировании вход в состояние s_1 (цикл) осуществлялся из состояния s_4 (энергопотребление 470 условных единиц), а в тесте $T_э$ переход в состояние s_1 осуществляется из состояния s_5 (энергопотребление этого перехода 433 условных единиц).

Таблица 3

Энергопотребление логической схемы на энергоемком тесте $T_э$

Энергоемкий тест $T_э$ (по циклу 2)	*E*, условные единицы
0110	575
1111	485
0110	740
0101	653
0110	433
1111	485
0110	740
0101	653
0110	433
1111	485
0110	740
0101	653

Результаты экспериментов

Для оценки эффективности предлагаемой методики были проведены два эксперимента. В качестве примеров были взяты эталонные автоматы dk14, dk15, dk17, kirkman из широко используемого набора примеров [8]. Пример Sist16 представлял собой операционный автомат, взятый из практики проектирования арифметико-логических устройств. Начальное моделирование проводили на псевдослучайных тестах. Энергопотребление оценивали по среднему потребляемому току в микроамперах. Результаты первого эксперимента представлены в табл. 4.

Анализ результатов в табл. 4 показывает, что оценка энергопотребления при VHDL-моделировании с тестом $T_э$ увеличивается в среднем на 40 % по сравнению с использованием исходного теста *T*. Уменьшение длины цикла при этом приводит к увеличению энергоемкости теста. Эффективность замены схемотехнического Spice-моделирования в системе аналогового моделирования *AccuSim II* (на базе моделирующего ядра *Eldo,* фирма Mentor Graphics) быстродействующим VHDL-моделированием в системе моделирования Questa Sim (версия 10.2c) следующая: для автомата kirkman Spice-моделирование в *AccuSim II* потребовало 30 мин для 1000 псевдослучайных наборов, а VHDL-моделирование заняло несколько секунд. Предлагаемый алгоритм нахождения энергоемких тестов для примеров последова-

Таблица 4

Результаты эксперимента 1

Автомат	Число элементов схемы	Число состояний автомата	Начальное моделирование		Моделирование на тесте $T_э$		Увеличение энергопотребления, %
			Длина теста T	Средний ток, мкА	Длина цикла (для теста $T_э$)	Средний ток, мкА	
Mealy	36	6	8000	49,4	5	59,4	20
					4	61,7	25
					3	64,2	30
dk14	81	7	8000	91,1	6	125,2	37
					4	125,9	38
					3	127,3	40
					2	138,7	52
dk15	54	4	8000	61,7	7	84,5	37
					5	88,2	43
					3	96,9	57
					2	97,7	58
dk17	44	8	8000	63,0	13	83,8	33
					5	92,1	46
					3	93,0	48
kirkman	264	16	16 000	96,5	50	114,2	18
					32	138,2	43
					25	148,7	54
			1000	96,7	32	127,3	32
sist16	2237	6	16 000	5574,4	7	7050,0	26
					5	7652,0	37

тельностей *T, S, E* длины 16 000 находит энергоемкий тест в течение секунды. Замена Spice-моделирования VHDL-моделированием показала, что погрешность в оценке среднего потребляемого тока составляет 5...15 %, при этом VHDL-моделирование, как правило, показывает большее значение потребляемого тока, чем эталонное Spice-моделирование, т. е. VHDL-моделирование может быть использовано для получения верхней оценки среднего потребляемого тока.

Эксперимент 2 был проведен на тех же примерах автоматов с одинаковой длиной теста 512, при этом сравнивали VHDL-моделирование и Spice-моделирование как на начальных тестах, так и найденных энергоемких тестах. Результаты эксперимента 2 приведены в табл. 5.

Поясним приведенные числовые значения на примере dk14: увеличение среднего тока при VHDL-моделировании на найденном энергоемком тесте составило 40,8 % по сравнению со средним током при моделировании на начальном тесте. Увеличение среднего тока для того же автомата на том же начальном и том же энергоемком тесте при Spice-моделировании составило 44,6 %.

Результаты эксперимента 2 свидетельствуют о том, что проводить длительное Spice-моделирование на начальном тесте не требуется. Можно найти энергоемкий тест с помощью VHDL-моделирования, а затем на коротком энергоемком тесте провести только Spice-моделирование. Уменьшение длины начальной последовательности для достаточно сложных автоматов с большим числом состояний не позволяет найти короткий энергоемкий тест, например, для автомата kirkman алгоритм выбора энергоемкого теста из 16 000 наборов нашел цикл длины 25 (см. табл. 4), а из 512 наборов — цикл длины 41 (см. табл. 5).

Таблица 5

Результаты эксперимента 2. Сравнение VHDL-моделирования со Spice-моделированием, длина начального теста 512, $m = 10$

Автомат	Длина цикла L	Длина теста $T_э$	Увеличение среднего тока на энергоемком тесте $T_э$	
			VHDL-моделирование, %	Эталонное Spice-моделирование, %
dk14	2	21	40,8	44,6
dk15	2	20	59,0	47,8
dk17	3	33	49,3	28,1
kirkman	41	456	18,7	14,7

Заключение

Предлагаемая методика оценки энергопотребления использует не абстрактные модели переключательной активности логических элементов, а базируется на результатах характеризации логических КМОП-элементов по параметру энергопотребления и позволяет быстро с достаточной для практики точностью оценить энергопотребление логических схем, реализующих конечные автоматы. Возможность построения энергоемких тестов позволяет проводить испытания цифровых блоков в режиме повышенного энергопотребления, что важно при разработке автономной цифровой аппаратуры.

Список литературы

1. **Рабаи Ж. М., Чандракасан А., Николич Б.** Цифровые интегральные схемы, 2-е издание: Пер. с англ. М.: Вильямс, 2007. 912 с.

2. **Белоус А. И., Мурашко И. А., Сякерский В. С.** Методы минимизации энергопотребления при проектировании КМОП БИС // Технология и конструирование в электронной аппаратуре. 2008. № 2. С. 39—44.

3. **Ghosh A., Devadas S., Keutzer K., White J.** Estimation of Average Switching Activity in Combinational and Sequential Circuits // Proc. 29th ACM/IEEE Design Automation Conference. 1992. P. 253—259.

4. **Roy K., Prasad S. C.** Low Power CMOS VLSI Circuit Design. New York: John Wiley and Sons, Inc., 2000. 359 p.

5. **Бибило П. Н., Соловьев А. Л.** Оценка энергопотребления комбинационных КМОП-схем на основе логического моделирования с учетом временных задержек элементов // Управляющие системы и машины. 2014. № 6. С. 34—41.

6. **Бибило П. Н.** Системы проектирования интегральных схем на основе языка VHDL. StateCAD, ModelSim, LeonardoSpectrum. М.: СОЛОН-Пресс, 2005. 384 с.

7. **Ashenden P. J., Lewis J.** VHDL-2008. Just the New Stuff. Burlington, MA, USA: Morgan Kaufman Publishers, 2007. 244 p.

8. **Yang S.** Logic Synthesis and Optimization Benchmarks User Guide. Version 3.0. Technical Report. North Carolina. Microelectronics Center of North Carolina. 1991.

Finding Energy-Intensive Tests for Logic Circuits that Implement Finite State Machines

P. N. Bibilo, bibilo@newman.bas-net.by, **V. I. Romanov**, rom@newman.bas-net.by, The United Institute of Informatics Problems of the National Academy of Sciences of Belarus, Minsk, 220012, Belarus

Corresponding author:

Bibilo Petr N., Head of Laboratory, the United Institute of Informatics Problems of the National Academy of Sciences of Belarus, 220012, Minsk, Belarus
E-mail: bibilo@newman.bas-net.by

Received on September 13, 2016
Accepted on October 20, 2016

We propose to conduct an estimation of power consumption of finite state machines represented by logic circuits. We use structural and behavior VHDL-models of these circuits. Our approach to power consumption estimation of synchronous logic circuits of finite state machines is based on the implementation of extended functional VHDL-models

of CMOS elements and fast tools of logical VHDL-simulation. An extended functional VHDL-model contains not only functions of its description but states which take into account variations of power consumption of inputs and outputs of elements. The technique of upper limitation of power consumption of logic circuits concludes in the process of looking for tests which induce reducing of power consumption — energy-intensive tests. We propose an algorithm for energy-intensive tests generation which uses cycle-accurate sequences: a set of tests T flows through S-state of a finite state machine; values of power consumption E of a logic circuit of a finite state machine. A common idea of the algorithm concludes in the process of finding a loop L into S (path in a graph of transitions of automaton). This loop is characterized with the maximum in a set of average values. On the last step of the algorithm is in the finding a path from an initial state to a vertex from the found loop. A set of tests from L occur m-fold in the resulting energy-intensive test T_e, where m can vary. The experiments on VHDL benchmarks of finite states machines have shown that the proposed technique implementation of energy-intensive tests leads to 40 % increasing of power consumption for structural VHDL-models for the found tests. Generating energy-intensive tests allows us to test of digital blocks for operability in the high power consumption mode.

Keywords: *logical circuit, CMOS technology, power consumption, tests, logical simulation*

For citation:

Bibilo P. N., Romanov V. I. Finding Energy-Intensive Tests for Logic Circuits that Implement Finite State Machines, *Programmnaya Ingeneria*, 2017, vol. 8, no. 1, pp. 7—15.

DOI: 10.17587/prin.8.7-15

References

1. **Rabai Zh. M., Chandrakasan A., Nikolich B.** *Cifrovye integral'nye shemy* (Digital Integrated Circuits), Moscow, Williams, 2007, 912 p. (in Russian).

2. **Belous A. I., Murashko I. A., Sjakerskij V. S.** Metody minimizacii jenergopotreblenija pri proektirovanii KMOP BIS (Methods of minimizing the power consumption in design of CMOS LSI), *Tehnologija i konstruirovanie v jelektronnoj apparature*, 2008, no. 2, pp. 39—44 (in Russian).

3. **Ghosh A., Devadas S., Keutzer K., White J.** Estimation of Average Switching Activity in Combinational and Sequential Circuits, *Proc. 29th ACM/IEEE Design Automation Conference*, 1992, pp. 253—259.

4. **Roy K., Prasad S. C.** *Low Power CMOS VLSI Circuit Design*, New York, John Wiley and Sons, Inc., 2000, 359 p.

5. **Bibilo P. N., Solov'ev A. L.** Ocenka jenergopotreblenija kombinacionnyh KMOP-shem na osnove logicheskogo modelirovanija s uchetom vremennyh zaderzhek jelementov (Estimation of power consumption of combinational CMOS circuits on the base of logical simulation taking into consideration the element time delay), *Upravljajushhie sistemy i mashiny*, 2014, no. 6, pp. 34—41 (in Russian).

6. **Bibilo P. N.** *Sistemy proektirovanija integral'nyh shem na osnove jazyka VHDL. StateCAD, ModelSim, LeonardoSpectrum* (CAD of integrated circuits based on the VHDL. StateCAD, ModelSim, LeonardoSpectrum), Moscow, SOLON-Press, 2005, 384 p. (in Russian).

7. **Ashenden P. J., Lewis J.** *VHDL-2008. Just the New Stuff*, Burlington, MA, USA, Morgan Kaufman Publishers, 2007, 244 p.

8. **Yang S.** *Logic Synthesis and Optimization Benchmarks User Guide. Version 3.0.* Technical Report, North Carolina, Microelectronics Center of North Carolina, 1991.

ИНФОРМАЦИЯ

УДК 004.4 DOI: 10.17587/prin.8.16-25

И. Б. Бурдонов, д-р физ.-мат. наук, вед. науч. сотр., igor@ispras.ru,
А. С. Косачев, канд. физ.-мат. наук, вед. науч. сотр., kos@ispras.ru,
Институт системного программирования РАН, Москва

Исследование ориентированного графа коллективом неподвижных автоматов

Настоящая работа — третья в серии статей, посвященных исследованию графов автоматами. В первых двух работах были описаны автоматы, а также коллектив автоматов, которые двигались по дугам графа, обменивались между собой сообщениями по независимой от графа сети связи, в данной статье описаны автоматы, которые не двигаются, находятся в вершинах графа и обмениваются сообщениями, пересылаемыми по дугам графа. Кроме исследования графа, также рассмотрена задача параллельных вычислений на графе, в том числе динамически меняющемся.

Ключевые слова: *ориентированный граф, упорядоченный граф, нумерованный граф, динамический граф, исследование графа, параллельные вычисления, автомат, робот, полуробот*

Введение

Данная работа является последней в серии из трех статей по исследованию графов автоматами. Граф предполагается ориентированным. В первых двух статьях [1, 2] была рассмотрена модель, аналогичная машине Тьюринга, в которой лента заменена графом: автоматы двигаются по дугам графа, а вершины графа используются как ячейки памяти для чтения/записи.

Если автомат один [1], то эта модель эквивалентна инвертированной модели, в которой автоматы неподвижно "сидят" в вершинах графа, а по дугам пересылается сообщение от автомата к автомату. Одному автомату соответствует одно сообщение, но оно меняется при прохождении через вершину, т. е. меняется автоматом, находящимся в этой вершине. В случае коллектива двигающихся автоматов [2] использовалась дополнительная сеть связи, по которой двигающиеся автоматы могли пересылать сообщения друг другу. В данной статье рассмотрена инвертированная модель со многими автоматами и многими сообщениями. В каждой вершине находится неподвижный автомат, а по дугам перемещается множество сообщений. Это, в каком-то смысле, более "суровая" постановка задачи по сравнению с постановкой задачи в работе [2], поскольку нет никакой независимой от графа сети связи автоматов, позволяющей пересылать сообщение от автомата любому другому автомату по его адресу. Сам исследуемый граф и есть такая сеть связи, и автомат в вершине может послать сообщение только ближайшим автоматам, а именно автоматам, находящимся на концах дуг, выходящих из этой вершины.

Как и в работах [1, 2] *упорядоченным* графом будем называть граф, в котором дуги, выходящие из вершины, перенумерованы, начиная с 1, — для каждой вершины задана своя нумерация выходящих дуг. В работах [1, 2] автомат указывал номер выходящей дуги, чтобы по ней пройти, в настоящей статье номер дуги указывается, чтобы послать по ней сообщение. Будем также считать, что в графе выделена начальная вершина, которую будем для краткости называть *корнем* графа. Вначале все автоматы находятся в своих начальных состояниях, а на дугах нет сообщений. Если для исследования графа требуется "толчок" извне, то он реализуется как сообщение, которое извне поступает в автомат в корне.

В работах [1, 2] были выделены три типа автомата в зависимости от размера их памяти. Автомат, который конечен на всем рассматриваемом классе графов, назывался *роботом*. Если автомат не конечный, но граф не помещается в его память, то такой автомат назывался *полуроботом*. По сути, это то же самое, что локальный алгоритм на графе. Если граф помещается в память автомата, то это *неограниченный автомат*. В данной статье все автоматы будут считаться полуроботами.

Рассмотрим два типа графа и две задачи, решаемые коллективом автоматов. Граф *статический*, если он не меняется в процессе работы автоматов, и *динамический*, если может изменяться: его дуги могут исчезать, появляться и менять свой конец. Первая задача — исследование графа с целью узнать, как он устроен. Вторая задача — параллельные вычисления на графе.

1. Неподвижные автоматы на статическом графе

Пусть имеется ориентированный упорядоченный граф, в каждой вершине которого находится автомат. Автомат "знает" только полустепень исхода вершины.

За одно срабатывание автомат принимает сразу все поступившие к нему сообщения по всем входящим дугам и может послать сразу несколько сообщений по выходящим дугам с указанием номеров этих дуг.

Дуга — это буфер сообщений емкостью k, т. е. на ней может одновременно находиться не более k сообщений. Если на ней уже k сообщений, еще не дошедших до автомата конца дуги, то автомат начала дуги не может посылать по этой дуге сообщения.

Говоря о емкости дуги предполагаем, что размер сообщения ограничен сверху. Очевидно, что посылка k сообщений максимального размера эквивалентна посылке одного большого сообщения, максимальный размер которого в k раз больше. Отличие лишь в том, что если на дуге меньше k маленьких сообщений, то на нее можно добавлять сообщения. Однако мы будем рассматривать алгоритм, для которого оценка времени работы не меняется по порядку, если такого добавления не делать и не посылать по дуге сообщения, пока она не освободится, а потом сразу послать несколько сообщений, но, конечно, не больше k.

Одни сообщения посылаются из автомата вершины сразу по всем выходящим дугам, другие — только по части выходящих дуг, а третьи — только по одной выходящей дуге. Тем не менее, для простоты будем считать, что сообщение ожидает в вершине освобождения сразу всех выходящих дуг. Понятно, что время работы алгоритма от этого не уменьшится. Таким образом, для автомата вершины не нужны сигналы об освобождении каждой выходящей дуги, а достаточно общего сигнала об освобождении всех выходящих дуг.

Для оценки времени работы алгоритма будем пренебрегать временем срабатывания автомата и считать, что сообщение находится на дуге не более одного такта, т. е. не более чем через один такт после посылки сообщения из автомата начала дуги оно будет принято автоматом конца дуги.

2. Разметка статического графа

Задачу исследования статического графа сформулируем следующим образом: когда извне в автомат корня придет сообщение, дающее старт работе автоматов, нужно выполнить *разметку* графа и сообщить об этом в ответном сообщении. Такая разметка означает, что автоматы в вершинах графа приведены в нужные состояния, т. е. в памяти каждого автомата хранится некоторая локальная информация о графе, а совокупная память автоматов описывает граф и его структуру.

Разметка графа включает в себя следующее.

1. *Прямой остов* графа, ориентированный от корня. В памяти автомата каждой вершины отмечаются все выходящие *прямые* дуги, т. е. дуги прямого остова. Остальные выходящие дуги — *хорды* прямого остова.

2. *Обратный остов* графа, ориентированный к корню. В памяти автомата каждой вершины, кроме корня, хранится номер выходящей обратной дуги, т. е. дуги обратного остова. Из вершины может выходить не более одной обратной дуги.

3. В памяти автомата каждой вершины запоминается число входящих обратных дуг.

Такую разметку графа можно понимать как результат его исследования. Когда разметка сделана, гарантируется, что по каждой дуге передано хотя бы одно сообщение, т. е. совершен обход графа. Кроме того, такая разметка используется в дальнейшем для параллельных вычислений на графе.

Алгоритм разметки графа подробно описан в работах [3, 4]. В настоящей статье дадим только идею этого алгоритма и приведем оценки времени его работы без доказательств. Условно разобьем алгоритм разметки на следующие три части: построение прямого и обратного остовов; определение конца построения; установка счетчиков числа входящих обратных дуг.

2.1. Построение прямого и обратного остовов

Построение остовов использует четыре типа сообщений:

1) ***Старт*** — идентифицирует вершины графа;
2) ***Поиск*** — ищет пути от вершин к корню;
3) ***Прямое*** — размечает прямой остов;
4) ***Обратное*** — размечает обратный остов.

Автомат корня получает сообщение ***Старт*** извне, с этого начинается работа по разметке графа. Далее сообщение ***Старт*** рассылается веером: получив первый раз это сообщение, автомат вершины рассылает его по всем выходящим дугам. Повторно приходящие сообщения ***Старт*** игнорируются. Это сообщение накапливает в себе *вектор маршрута*, который оно проходит, как список номеров дуг маршрута. Идентификатором вершины становится *вектор вершины*, который определяется как вектор маршрута из первого полученного автоматом вершины сообщения ***Старт***. Вектор корня пустой.

Сообщение ***Поиск*** ищет путь от вершины к корню. Автомат вершины, получив ***Старт***, инициирует сообщение ***Поиск***. Это сообщение содержит вектор инициатора и накапливает в себе *обратный вектор* как вектор маршрута, который проходит. Сообщение ***Поиск*** также рассылается веером. Повторно приходящие сообщения ***Поиск*** от того же инициатора игнорируются. Однако инициаторов много — это все вершины. Поэтому, чтобы игнорировать повторные сообщения от тех же инициаторов, автомат вершины хранит в своей памяти множество векторов инициаторов из проходивших через него сообщений ***Поиск***. Автомат корня, приняв сообщение ***Поиск***, получает вектор инициатора, т. е. вектор прямого пути от корня до инициатора, и обратный вектор пути от инициатора до корня. Автомат корня создает сообщение ***Прямое***, переписывая в него вектор инициатора и обратный вектор.

Сообщение ***Прямое*** размечает прямой путь от корня до инициатора. Это сообщение двигается по прямому пути от корня до инициатора, для чего используется вектор инициатора в сообщении. Автомат

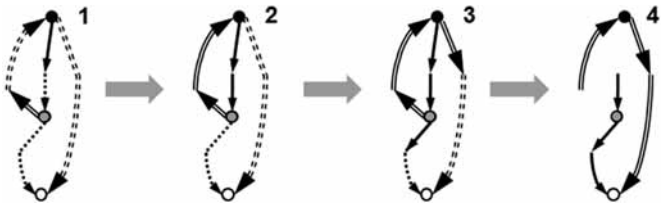

Рис. 1. Сообщение *Обратное* идет до корня

каждой вершины отмечает как прямую ту дугу, по которой посылает сообщение ***Прямое***. Инициатор, получив сообщение ***Прямое***, создает сообщение ***Обратное***, переписывая в него обратный вектор. Заметим, что вектор каждой вершины на пути, который проходит сообщение ***Прямое***, является префиксом вектора инициатора в сообщении.

Сообщение ***Обратное*** двигается по обратному пути и размечает обратные дуги. Это сообщение содержит остаток обратного вектора: автомат каждой вершины запоминает номер той дуги, по которой посылает сообщение ***Обратное***, как номер обратной дуги и удаляет этот номер из обратного вектора в сообщении. Если раньше был запомнен другой номер, то он забывается.

Почему сообщение ***Обратное*** идет до корня, а не до какой-нибудь вершины, где уже есть обратная дуга? Дело в том, что иначе может возникнуть цикл обратных дуг. Это видно на рис. 1. Здесь одновременно размечаются два обратных пути от разных инициаторов (черный и серый кружки) до корня (белый кружок внизу), обозначенные одинарной и двойной линиями (1), соответственно. Пунктир или двойная штриховая линия — еще не пройденная часть пути. Если сообщения идут не до корня, то может получиться цикл обратных дуг (2). Если же сообщения продолжают движение, то возникают новые обратные дуги (3), а старые забываются (4). И дальше до самого корня получается фрагмент обратного остова.

2.2. Определение конца построения остовов

Как определить конец построения прямого и обратного остовов? Идея решения основана на том, что у дуги есть одно начало и один конец. Считаем дуги по их началу, т. е. выходящие дуги, и по их концу, т. е. входящие дуги. Оба числа должны, в конце концов, совпасть. Для этого в автомате корня ведется подсчет числа дуг графа. Сначала счетчик равен числу дуг, выходящих из корня.

Модифицируем сообщение ***Поиск***, добавив в него параметр "число дуг, выходящих из вершины-инициатора". Это число автомат корня прибавит к счетчику дуг, когда получит сообщение ***Поиск***. Это подсчет дуг по их началу.

Еще добавим два новых сообщения: ***Финиш*** и ***Минус***. Сообщение ***Финиш*** посылается из начала дуги в ее конец при получении сообщения ***Прямое*** и заставляет конец дуги учесть эту входящую дугу. Сообщение ***Минус*** предназначено для того, чтобы вычитать из счетчика дуг число учтенных дуг, входящих в вершину. Это подсчет дуг по их концу.

Сообщение ***Минус*** посылается по обратному остову до корня. До того как появилась обратная дуга, автомат вершины ведет счетчик учтенных входящих дуг, значение которого и посылает в первом сообщении ***Минус***, когда появляется обратная дуга. Но и после этого в автомат вершины могут продолжать приходить сообщения ***Финиш*** по еще неучтенным входящим дугам, которые будут вызывать уже немедленную отправку сообщений ***Минус***.

Проиллюстрируем как все перечисленные механизмы работают. Пусть у нас есть две смежные прямые дуги *ab* и *bc*. На рис. 2 показана временная диаграмма с последовательностями сообщений.

Сообщение ***Поиск*** от автомата вершины *a* прибавляет единицу к счетчику дуг в автомате корня для дуги *ab*, а сообщение ***Поиск*** от автомата вершины *b* прибавляет единицу к этому счетчику для дуги *bc*. Сообщение ***Минус*** от автомата вершины *b* вычитает единицу из счетчика для дуги *ab*. Видим, что для каждой дуги *ab* вычитание единицы из счетчика происходит не только после прибавления к нему единицы для этой дуги *ab*, но и после прибавления единицы для каждой следующей дуги *bc*. В силу этого счетчик дуг в автомате корня обнулится только после того, как прямой и обратный остовы будут построены.

2.3. Установка счетчиков числа входящих обратных дуг

После того как построены прямой и обратный остовы и определен конец построения, выполняется установка счетчиков входящих обратных дуг. Для этого используются два сообщения: ***Начало*** и ***Конец*** подсчета. Сообщение ***Начало*** создает автомат корня и рассылает по прямому остову. Получив такое сообщение, автомат вершины создает сообщение ***Конец***, которое посылает по обратному остову до корня.

Сначала в сообщении ***Конец*** есть признак "*первая обратная дуга*". По этому признаку автомат вершины добавляет единицу к своему счетчику входящих обратных дуг, а далее сообщение ***Конец*** пересылается уже без этого признака.

Сообщения ***Конец*** нужны автомату корня, чтобы определить завершение этого этапа, да и всей разметки. Каким образом? От каждой вершины до корня дойдет ровно одно сообщение ***Конец***. Поэтому автомату корня достаточно "знать" число вершин. А он это "знает" как число элементов во множестве

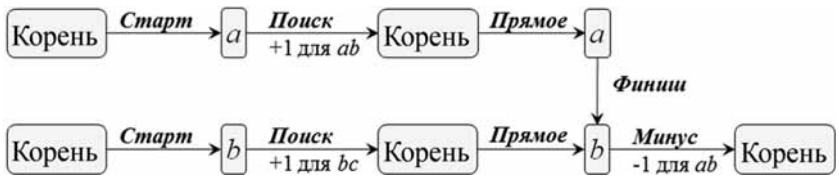

Рис. 2. Временная диаграмма для определения конца построения остовов

инициаторов, которое хранилось в автомате корня, когда передавались сообщения ***Поиск***.

2.4. Оценки алгоритма

Время работы алгоритма равно $\boldsymbol{O}(n/k + d)$, где n — число вершин графа; k — емкость дуги; d — диаметр графа (длина максимального пути). Память автомата вершины равна $\boldsymbol{O}(nd\log s_{out})$, где s_{out} — ограничение сверху на полустепень исхода вершин. Размер сообщения равен $\boldsymbol{O}(d\log s_{out})$.

3. Параллельные вычисления и агрегатные функции

Переходим к решению второй задачи — параллельным вычислениям на графе. Что будет параллельно вычисляться? Это будет функция от мультимножества значений в вершинах графа. Можно считать, что эти значения записаны в вершины графа, а автоматы могут их читать. Или у каждого автомата есть операция — получить значение для той вершины, в которой он находится. Способ, каким автомат узнает значение, приписанное вершине, не важен.

Сначала посмотрим, как устроены те функции, которые будут вычисляться. Пусть X — это некоторое базовое множество. Вершинам будут приписываться значения из этого множества X. Через $X^{\#}$ обозначим множество всех конечных мультимножеств элементов из X. Нам нужно мультимножество, потому что разным вершинам могут быть приписаны одинаковые значения.

Функция g на множестве $X^{\#}$ называется *агрегатной*, если существует такая функция e, что значение функции g от объединения двух мультимножеств a и b можно вычислить так: сначала вычисляем g для каждого из этих мультимножеств, а потом к двум полученным значениям применяем функцию e. Формально: $\forall a,\ b \in X^{\#},\ g(a \cup b) = e(g(a),\ g(b))$. Иными словами, агрегатную функцию можно вычислить "по частям".

Примеры агрегатных функций: сумма, минимум и сумма квадратов:

$$\Sigma\colon \{a_1, \ldots, a_n\} \to (a_1 + \ldots + a_n), \text{ здесь } e = +;$$

$$\min\colon \{a_1, \ldots, a_n\} \to \min\{a_1, \ldots, a_n\}, \text{ здесь } e = \min;$$

$$Q\colon \{a_1, \ldots, a_n\} \to (a_1^2 + \ldots + a_n^2), \text{ здесь } e = +.$$

Понятно, что не все функции являются агрегатными. Например, среднее арифметическое не агрегатно. Однако можно использовать *агрегатное расширение* функции. Если функцию f можно представить как композицию $f = h \circ g$ двух функций h и g, где g — агрегатная функция, то эта функция g называется агрегатным расширением функции f. Это значит, что можно делать вычисления "по частям", используя агрегатное расширение g, а в конце один раз применить функцию от одного аргумента — функцию h.

Однако возможны такие расширения, которые на практике ничего не дают. Например, можно взять в качестве g тождественную функцию на $X^{\#}$, а в качестве функции h — саму функцию f. Чтобы избежать этого используется *минимальное* агрегатное расширение. Интуитивно, это агрегатная функция, вычисляющая минимум информации, по которой еще можно восстановить f. Формально функция $g\colon X^{\#} \to B$ минимальное агрегатное расширение f, если $g(X^{\#}) = B$ и для каждого агрегатного расширения $g`\colon X^{\#} \to B`$ функции f существует такая функция $j\colon B` \to B$, что $g = j \circ g`$. Минимальное расширение для любой функции f существует и единственно с точностью до изоморфизма. Примеры минимальных агрегатных расширений:

- среднее арифметическое, для $f\colon \{a_1, \ldots, a_n\} \to (a_1 + \ldots + a_n)/n$ имеем

  $g\colon \{a_1, \ldots, a_n\} \to (a_1 + \ldots + a_n, n)$ и $h\colon (a, n) \to a/n$;

- среднее геометрическое, для $f\colon \{a_1, \ldots, a_n\} \to (a_1 \times \ldots \times a_n)^{1/n}$ имеем

  $g\colon \{a_1, \ldots, a_n\} \to (a_1 \times \ldots \times a_n, n)$ и $h\colon (a, n) \to a^{1/n}$;

- среднее квадратичное, для $f\colon \{a_1, \ldots, a_n\} \to ((a_1^2 + \ldots + a_n^2)/n)^{1/2}$ имеем

  $g\colon \{a_1, \ldots, a_n\} \to (a_1^2 + \ldots + a_n^2, n)$ и $h\colon (a, n) \to (a/n)^{1/2}$.

Такая теория агрегатных функций — это модификация теории индуктивных функций [5], доказательства утверждений см. в работе [4].

4. Параллельные вычисления на статическом графе

Теперь перейдем к алгоритму вычисления функции на статическом графе (подробнее см. в работах [4, 6]). Для этого будем использовать разметку графа из разд. 2. Заметим, что разметка делается один раз, а потом может использоваться для многократного вычисления различных функций.

Пусть вершинам приписаны значения из базового множества X. Вычисление начинается, когда автомат корня получает извне сообщение, содержащее три функции: h, g и e. На практике могут присылаться программы вычисления значений этих функций, или какие-то ссылки на эти программы, по которым автоматы могут получить к ним доступ. Вычисление основано на алгоритме пульсации.

От корня вверх по прямому остову распространяется сообщение ***Вопрос***, содержащее две функции: g и e. А вниз по обратному остову распространяется сообщение ***Ответ***, содержащее значение функции g от мультимножества значений в вышележащих вершинах обратного остова.

Автомат листовой вершины обратного остова получив ***Вопрос*** вычисляет функцию g от значения, записанного в этой вершине, и посылает его как параметр сообщения ***Ответ*** по обратной дуге.

Автомат нелистовой вершины таким же образом вычисляет функцию g от значения в этой вершине,

запоминает его, но не посылает, пока не получит ***Ответы*** по всем входящим обратным дугам. Число таких дуг уже установлено в автоматах вершин при разметке графа. Каждый раз, получив ***Ответ*** по входящей обратной дуге, автомат вершины применяет функцию *e*, параметрами которой будут значение, полученное в сообщении ***Ответ***, и запомненное значение. Когда получены ***Ответы*** по всем входящим обратным дугам, автомат вершины сам посылает по выходящей обратной дуге ***Ответ*** с накопленным значением функции *g*.

Автомат корня посылает ***Ответ*** вовне, предварительно применив к накопленному значению функции *g* завершающую вычисления функцию *h*.

Время вычисления не более $3d$, память автомата равна $\boldsymbol{O}(s_{out} + \log s_{in} + y + z)$, размер сообщения равен $\boldsymbol{O}(y + z)$, где d — диаметр графа; $s_{out} \leqslant m$ — максимальная полустепень исхода вершины; $s_{in} \leqslant m$ — максимальная степень захода вершины; m — число дуг графа; y — размер вопроса (функций h, g и e), z — размер ответа (значения функции).

5. Неподвижные автоматы на динамическом графе

Динамическим графом будем называть такой граф, дуги которого могут меняться со временем. При этом будем считать, что вершины графа, в которых "сидят" автоматы, не изменяются. Есть перечисленные далее три вида изменения дуг.

1. **Дуга может появиться.** Об этом автомат начала дуги извещается специальным сигналом *появления дуги* с параметром номер дуги.

2. **Дуга может исчезнуть.** Если по дуге передавалось сообщение, то оно пропадает, о чем автомат начала дуги извещается сигналом *исчезновения дуги* с параметром "номер дуги". Если же на дуге не было сообщений, то при ее исчезновении никаких сигналов не возникает. Однако, если автомат вершины попытается послать сообщение по исчезнувшей выходящей дуге, сообщение не будет передано, а автомат получит сигнал исчезновения дуги.

3. **Дуга может поменять свою конечную вершину.** В этом случае никаких сигналов не предусмотрено.

Заметим, что выбрана модель с минимальным набором сигналов, все еще позволяющих рано или поздно распознавать изменение дуги. Кроме указанных, есть еще сигнал *освобождения дуги*, означающий, что сообщение, которое по дуге передавалось, уже передано и можно послать новое сообщение. Будем предполагать, что емкость дуги равна 1: одновременно по дуге передается не более одного сообщения.

Если дуга меняется слишком часто и автомат не успевает обрабатывать сигналы от этой дуги, то могут возникнуть два необработанных сигнала — старый и новый. Заметим, что новым сигналом может быть только сигнал появления дуги. В этом случае работает правило замещения, которое определяет, какой из двух сигналов остается, а какой теряется: появление + появление → появление; исчезновение + появление → появление; освобождение + появление → освобождение или появление. Для того чтобы автоматы могли отслеживать изменения дуг, важно, чтобы после сигнала исчезновения дуги не потерялся сигнал ее появления, иначе автомат останется "в убеждении", что дуги нет. Если же после сигнала освобождения дуги возникает сигнал ее появления, то это означает, что сообщение передано по дуге, затем дуга исчезла (без сигнала), а потом опять появилась. Это эквивалентно смене конца дуги, поэтому все равно, какой из двух сигналов остается, а какой теряется.

Граф будем считать *нумерованным*: все вершины пронумерованы и в каждой вершине записан ее номер, который автомат может прочитать.

6. Мониторинг динамического графа

Ставится задача *мониторинга* графа: сбор полной информации о графе и отслеживание изменений дуг. Будем предполагать, что такая информация собирается не в каком-то выделенном автомате, а в каждом автомате. На самом деле для динамического графа это все равно.

Понятно, что если граф постоянно меняется, то нельзя гарантировать, что описания текущего состояния всех его дуг отражены во всех автоматах: сообщения о последних изменениях дуг могут просто не дойти до каких-то автоматов. Поэтому требуется только то, чтобы через некоторое конечное время T_0 после изменения дуги все автоматы "узнали" об этом или более позднем изменении дуги. Если после данного изменения дуга больше не меняется, по крайней мере, в течение времени T_0, то во всех автоматах будет одинаковое (и верное) описание этой дуги.

Если дуга меняется так часто, что никакое сообщение по ней не успевает дойти или уходит неизвестно куда, то нельзя гарантировать возможность доставки сообщения в любую вершину. Эту неопределенность нужно как-то ограничить. Будем считать, что дуга *долгоживущая*, если за то время, пока она не меняется, по ней может успеть пройти хотя бы одно сообщение. Будем пренебрегать временем срабатывания автомата, а время передачи сообщения по дуге ограничим сверху одним тактом. Таким образом, долгоживущая дуга должна не меняться хотя бы в течение одного такта. Ограничение, которое нам нужно, звучит так: в каждый момент времени подграф, порожденный долгоживущими дугами, является сильно связным суграфом, т. е. содержит все вершины графа.

Алгоритм мониторинга подробно описан в работе [7], в настоящей статье рассмотрим только основную идею алгоритма и приведем оценки времени работы без доказательств.

6.1. Описания дуг и сообщения

Полная информация о графе, которую нужно собрать, это множество описаний всех его дуг. Описание дуги — это тройка: номер начала дуги, т. е. начальной вершины дуги, номер дуги (в ее начале)

и номер конца дуги, т. е. конечной вершины дуги. Если номер конца дуги еще не известен, то указывается номер 0 (считаем, что вершины пронумерованы, начиная с 1).

Сразу можно сказать, что в одном сообщении автомат должен посылать описания всех известных ему дуг. Это *правило одного сообщения*. Его необходимость объясняется тем, что по долгоживущей дуге гарантированно проходит только одно сообщение, поэтому в него надо поместить всю имеющуюся информацию. Во-вторых, сообщение нужно посылать по дуге не тогда, когда информация в автомате обновилась, а каждый раз, когда появляется такая возможность, т. е. при появлении дуги и при освобождении дуги. В противном случае информация от автомата данной вершины может не достигнуть других автоматов, поскольку выходящая из вершины дуга может несколько раз менять свой конец без какого-либо сигнала для автомата этой вершины.

Кто и когда обнаруживает изменение дуги, в результате чего создает или корректирует описание дуги? Если дуга появляется, то это обнаруживает автомат в начале дуги по сигналу появления дуги. Однако он еще не "знает" конца дуги. Если дуга исчезает, то это обнаружит тоже автомат начала дуги по сигналу исчезновения дуги. Этот сигнал он получит либо сразу, если по дуге передавалось сообщение, либо позже, когда при обработке предыдущего сигнала освобождения дуги попытается послать по дуге сообщение. Если дуга меняет свой конец, то это обнаруживает автомат нового конца дуги, когда получает по дуге сообщение, в котором описание дуги содержит другой номер конца. В частности, номер конца в сообщении будет нулевым, если автомат начала дуги еще не "знает" конца дуги, т. е. сразу после появления дуги. Для того чтобы автомат конца дуги мог это делать, он должен "знать", по какой дуге передавалось сообщение. Для этого сообщение, кроме множества описаний дуг, содержит указание на описание дуги, по которой оно передается.

6.2. Ранг дуги

Поскольку дуга может меняться несколько раз, автомат, получая сообщение, должен "понять", получил ли он описание дуги более новое, чем то, что у него есть, или нет. Новое описание дуги должно заменять собой старое описание. Для этого вводится ранг описания дуги, который будем называть просто *рангом дуги*. Каждый раз, когда какой-то автомат обнаруживает изменение дуги и поэтому создает или корректирует ее описание, ранг дуги в этом описании увеличивается.

Однако недостаточно просто увеличить ранг дуги, например, на единицу. Нужно быть уверенным, что после такого увеличения в любом другом автомате дуга имеет строго меньший ранг. Здесь возникает неопределенность вследствие серии изменений конца дуги, происходящих в течение времени, пока автомат начала дуги еще не "узнал" ни об одном из этих изменений. Автоматы этих сменяющих друг друга концов дуги увеличивают один и тот же ранг, естественно, одинаково, например, на единицу. Поэтому сразу в нескольких автоматах эта дуга будет иметь одинаковый, увеличенный на единицу, ранг. Эту задачу решает автомат начала дуги, который "оставляет последнее слово всегда за собой". Каждый раз, когда он получает описание дуги с бóльшим рангом, чем у него хранится, он еще раз увеличивает ранг дуги на единицу. Соответственно, при появлении или исчезновении дуги автомат начала дуги тоже увеличивает ее ранг сразу на два.

6.3. Оценки

Время T_0 имеет порядок числа вершин $\boldsymbol{O}(n)$. Однако если все изменения в графе прекращаются, то время T_1, которое необходимо для того чтобы во всех автоматах оказалась правильная информация, учитывающая последние изменения, еще меньше — порядка диаметра графа $\boldsymbol{O}(d)$.

Размер памяти автомата (и сообщения) довольно большой, он равен $\boldsymbol{O}(m\log snv)$, где m — число дуг графа; s — ограничение сверху на полустепень исхода вершин; v — максимальное число изменений одной дуги. Но это и естественно, поскольку в памяти автомата создается описание всего графа. Это же описание, по сути, передается в сообщении, размер которого имеет тот же порядок.

Можно обратить внимание, что размер памяти зависит от максимального числа изменений одной дуги. Это затраты на ранг дуги. Для того чтобы оценка памяти не зависела от числа изменений, можно предложить две модификации модели.

6.4. Модификации модели

Первая модификация самая простая: потребуем, чтобы все дуги были долгоживущими. Тогда число изменений одной дуги не превосходит времени работы алгоритма в тактах, а оно имеет порядок числа вершин n. Следовательно, ранг можно сделать циклическим с максимальным значением порядка n.

Вторая модификация — наличие таймера, который посылает каждому автомату временной сигнал каждый такт. Тогда все те действия, что автомат выполнял при обработке сигнала освобождения или появления дуги, он будет делать при обработке временного сигнала. Сообщения будут посылаться по дуге не чаще одного за такт. При этом время T_0 не изменится по порядку, а тогда опять ранг можно сделать циклическим с максимальным значением порядка n.

Правда, при второй модификации время существования долгоживущих дуг придется увеличить с одного до двух тактов. В противном случае может оказаться, что каждый раз при появлении дуги сообщение посылается по ней не сразу, а спустя какое-то время (до получения временного сигнала), в силу чего сообщение не успевает передаться по дуге до ее изменения.

7. Параллельные вычисления на динамическом графе

Параллельные вычисления на динамическом графе подробно описаны в работе [8], в настоящей статье рассмотрим только основные идеи алгоритмов и приведем оценки времени и памяти без доказательств. Для параллельных вычислений на динамическом графе нам тоже потребуется предварительная разметка графа. Мониторинг не дает такой разметки — у него вообще другая задача. На статическом графе вычисления выполнялись с помощью сообщений ***Вопрос*** и ***Ответ***, которые использовали разметку графа, состоящую из прямого и обратного остовов и счетчиков обратных дуг. Однако, если граф динамически меняется, возникает следующая задача: когда меняется дуга остова, нужно этот остов скорректировать. В общем случае это может потребовать полной перестройки остовов, т. е. фактически придется заново их строить. Кроме того, не гарантируется доставка сообщений по дугам остова: эти дуги могут исчезать или менять свой конец до того, как по ним будет передано нужное сообщение.

В то же время, как в случае мониторинга графа, предположение о долгоживущих дугах гарантирует возможность доставки информации из любой вершины в любую вершину. Но для этого автомат вершины должен следовать правилу одного сообщения. Получив сообщение с нужной информацией, он сохраняет ее в своей памяти и далее помещает во все сообщения, которые постоянно посылает по всем выходящим дугам, когда появляется такая возможность, т. е. по сигналам освобождения и появления дуги.

Для доставки сообщения ***Вопрос*** этого достаточно, и никакого прямого дерева не требуется. Точно также можно было бы доставить в корень значения из всех вершин графа и уже в автомате корня выполнить вычисление функции от получившегося мультимножества. Однако, поскольку необходимо всю информацию помещать в одно сообщение, размер такого сообщения был бы максимальным и пропорциональным числу вершин графа. Обратный остов и счетчики обратных дуг позволяют аккумулировать ответы: автомат вершины, получив ответы по всем входящим обратным дугам, вычисляет промежуточное значение агрегатной функции и посылает дальше по выходящей обратной дуге только один ответ. Тем самым, сообщение содержит ответ только от автомата одной вершины каждой ветви обратного остова от корня до листа. Размер сообщения становится пропорциональным не числу вершин, а ширине обратного остова.

Итак, обратный остов и счетчики обратных дуг нужны только для уменьшения размера сообщения, содержащего ответы, т. е. вычисленные промежуточные значения агрегатной функции. Такой остов может быть виртуальным. Вершины в нем настоящие, а дуги — виртуальные. Дуга задается просто парой ее вершин: начальной и конечной. Поскольку остов виртуальный, появляется свобода — какой остов

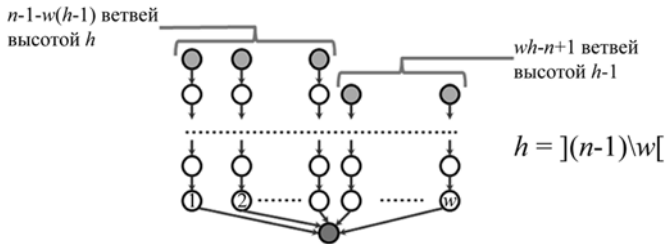

Рис. 3. "Сбалансированный веник"

лучше построить. Его ширина w (число листовых вершин, кроме корня) определяет размер сообщения, а высота h (длина максимальной ветви от корня до листа) — время вычисления, поскольку вычисления распараллеливаются только между разными ветвями остова, а по одной ветви выполняются последовательно от листа к корню.

Возникает задача: для данного числа вершин определить минимальную высоту дерева при заданной ширине и описать вид такого дерева. Оно будет выглядеть как "сбалансированный веник" (рис. 3)

"Веник" можно задать с помощью двухуровневых индексов вершин, отличных от корня. Индекс вершины состоит из номера ветви веника от 1 до w и номера на этой ветви от 1 до h, считая от корня. В таком "венике" счетчик входящих обратных дуг нужен только в автомате корня, его значение равно w. Автомату другой вершины достаточно "знать", является вершина листовой (счетчик равен 0) или внутренней (счетчик равен 1) вершиной "веника".

7.1. Разметка динамического графа

Разметка выполняется в два этапа:

1) сначала в автомате корня собирается информация о всех вершинах; используется сообщение ***Прямое***;

2) затем автомат корня строит в своей памяти "веник" и рассылает его во все вершины; используется сообщение ***Обратное***.

Получив стартовое сообщение извне, корень создает сообщение ***Прямое***, которое потом циркулирует по всему графу с помощью постоянной веерной рассылки. В этом сообщении и в автоматах вершин, через которые оно проходит, накапливаются описания дуг. Описание дуги, как и в случае мониторинга динамического графа, содержит номер начальной вершины дуги, номер дуги в ее начальной вершине и номер конечной вершины дуги, если он известен, или знак "вопрос", если неизвестен. Долгоживующие дуги гарантируют доставку сообщения до любой вершины и из любой вершины до корня.

Автомат корня проверяет замкнутость по дугам: нет знаков вопроса, т. е. для каждой дуги известна конечная вершина этой дуги. Однако такая "статическая" замкнутость гарантирует, что все вершины известны только для статического графа. В динамическом графе конец дуги может меняться без всяких сигналов, поэтому замкнутость по дугам ничего не гарантирует.

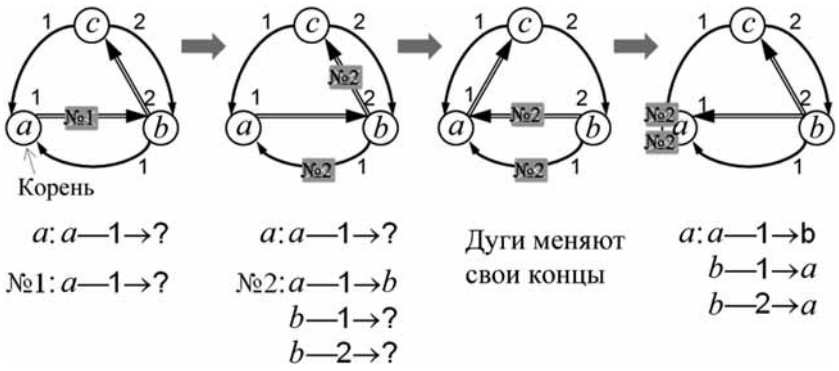

Рис. 4. Недостаток статической замкнутости

Рассмотрим пример на рис. 4. Автомат корня (вершины *a*) посылает сообщение № 1, в котором одна дуга с неизвестным концом. Автомат вершины *b*, получив сообщение, записывает вершину *b* как конец этой дуги и рассылает по двум выходящим дугам сообщение № 2, в котором неизвестны концы этих двух дуг. Одно направляется в корень *a*, а другое — в вершину *c*. Однако две дуги, помеченные двойной линией, меняют свои концы, поэтому оба сообщения № 2 приходят в корень. И автомат корня, как ему и предписано, записывает корень как конечную вершину этих дуг. В результате в автомате корня получается замкнутое множество дуг, однако вершина *c* осталась неизвестной.

Для того чтобы замкнутость гарантировала "известность" всех вершин, требуются два дополнительных условия.

Первое условие — это новое правило замещения сигналов: освобождение + появление → освобождение. Дело в том, что важно знать, дошло сообщение по дуге или нет, т. е. сигнал освобождения не должен теряться.

Второе условие: понадобятся *начальные дуги*, которые обладают двумя свойствами:

1) начальная дуга существует с самого начала и не меняется до тех пор, пока по ней не пройдет первое сообщение;

2) по начальным дугам из корня достижимы все вершины.

Теперь появилось модифицированное условие "динамической" замкнутости: для каждой известной дуги либо известна конечная вершина этой дуги (по дуге послано первое сообщение, оно дошло до конца дуги и вернулось к корню), либо по этой дуге не прошло первое сообщение (дуга гарантированно не начальная).

Важно, что автомат каждой вершины регистрирует только те выходящие из этой вершины дуги, которые появились до получения им первого сообщения. А те, что появляются позже, не регистрируются. Почему?

Во-первых, этого достаточно, поскольку все начальные дуги будут зарегистрированы, так как появляются с самого начала. Таким образом, будут зарегистрированы все вершины, поскольку они достижимы по начальным дугам.

Во-вторых, это важно для минимизации времени сбора информации о вершинах. Оно будет порядка числа вершин *n*. Если бы автоматы регистрировали все *m* дуг, то время увеличилось бы. Действительно, дуги могут все время появляться, и, если они продолжают регистрироваться, то замкнутость в корне может возникнуть после появления всех *m* дуг, потому что для каждой появившейся дуги придется ждать выяснения, какой же у нее конец. Рассмотрим пример на рис. 5. Каждый раз появляется одна петля, т. е. когда в корень приходит информация о конце *i*-й петли, к нему же приходит информация о появлении $(i + 1)$-й петли, но без конца. Поэтому корень ждет, пока появятся все *m* петель, т. е. $\boldsymbol{O}(m)$ тактов.

Заметим также, что для разметки динамического графа можно обойтись без нумерованности графа, если использовать идентификацию вершин с помощью динамически создаваемых векторов вершин, как это делалось для разметки статического графа. Для этого используется сообщение ***Старт***, которое накапливает в себе вектор маршрута, который оно проходит. Корень получает ***Старт*** извне с пустым вектором. Автомат каждой вершины, получив первый раз сообщение ***Старт***, запоминает его вектор маршрута как вектор вершины и посылает ***Старт*** по всем выходящим дугам. Регистрация выходящих дуг прекращается. Повторные сообщения ***Старт*** игнорируются. Начальные дуги гарантируют, что каждая вершина получит сообщение ***Старт*** и, тем самым, получит свой вектор. Поскольку по каждой дуге посылается не более одного сообщения ***Старт***, вектор вершины будет уникальным в графе.

Только после этого автомат вершины начинает выполнять первый этап разметки, т. е. работать с сообщением ***Прямое***: каждый раз, когда дуга освобождается или появляется, по ней посылается сообщение ***Прямое*** с накопленным множеством описаний дуг. При получении сообщения ***Прямое*** множество описаний дуг из сообщения объединяется с множеством описаний дуг, хранящимся в памяти автомата. При этом вектор вершины-получателя вставляется в описание дуги, по которой принято сообщение, как вектор конца дуги.

Когда автомату корня становятся известны все вершины, он создает описание виртуального "веника", которое рассылается веером по всему графу в сообщении ***Обратное***, начинается второй этап разметки. Описание "веника" — это множество описаний вершин. Для каждой вершины по ее номеру указывается ее индекс в "венике" и признак: листовая или не

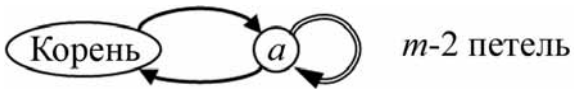

Рис. 5. Пример ожидания регистрации всех *m* дуг

листовая. Когда автомат вершины первый раз получает сообщение с описанием "веника", он запоминает отдельно описание своей вершины, удаляет его из описания "веника" и такое уменьшенное описание "веника" сохраняет и рассылает дальше. При получении повторного сообщения автомат вершины строит пересечение множества описаний вершин, которое хранится в нем, и которое содержится в сообщении. Это пересечение сохраняется и рассылается дальше. Рано или поздно сообщение становится пустым и через какое-то время описание "веника" в автомате корня тоже становится пустым. Этот факт и означает завершение второго этапа разметки. Далее система готова к параллельным вычислениям.

7.2. Алгоритм пульсации

Как и в случае статического графа для параллельных вычислений на динамическом графе применяется алгоритм пульсации вопросов и ответов. Однако по правилу одного сообщения вся нужная информация должна рассылаться по графу в одном сообщении, распространяемом веером с размножением в каждой вершине. Поэтому на этапе вычисления функции распространяется сообщение ***Вопрос—Ответ***, которое содержит как вопрос, так и ответы. При этом для каждой ветви "веника" в сообщении не больше одного ответа, а именно ответа от автомата вершины с наименьшим номером по ветви. Ответ задается вместе с индексом отправителя.

После завершения вычисления по графу может продолжать циркулировать такое сообщение ***Вопрос—Ответ***. Оно должно вытесняться сообщением для следующего вычисления. Для этого автомат корня нумерует вопросы в порядке их поступления извне и номер вопроса помещается в сообщение.

Итак, сообщение ***Вопрос—Ответ*** содержит: номер вопроса; вопрос, т. е. функции g и e; множество пар (ответ как значение функции g, индекс вершины-отправителя ответа).

7.3. Оценки

Время разметки графа равно $\boldsymbol{O}(n)$, время вычисления функции равно $\boldsymbol{O}(n^2/w)$. Память автомата и размер сообщения: на этапе разметки равны $\boldsymbol{O}(m\log n s_{out})$, на этапе вычисления равны $\boldsymbol{O}(y + \log N + wz + w\log n)$. Здесь n — число вершин; m — число дуг; s_{out} — ограничение на полустепень исхода вершин; y — размер вопроса (описаний функций h, g и e); z — размер ответа (значения функции); N — максимальное число вопросов.

Заключение

В серии из работ [1, 2] и данной статьи представлены результаты по исследованию графов с помощью автоматов. В работе [1] изучались возможности одного автомата, в том числе конечного (робота), а в следующих статьях — коллектива автоматов. В работах [1, 2] были описаны автоматы, которые двигались по дугам графа и, если их было несколько, обменивались между собой сообщениями по независимой от графа сети связи, гарантирующей доставку сообщения "от точки к точке" по адресу получателя. В данной статье описаны автоматы, которые ассоциированы с вершинами графа и никуда не двигаются, но обмениваются сообщениями, пересылаемыми по дугам графа.

В частности, в работе [2] рассматривалась работа коллектива двигающихся автоматов на недетерминированном графе, а в данной статье — работа коллектива неподвижных автоматов на динамически меняющемся графе. Темой дальнейших исследований могла бы стать, наоборот, работа двигающихся автоматов на динамическом графе и неподвижных автоматов на недетерминированном графе. К числу нерешенных задач можно отнести также Δ-обход недетерминированного графа коллективом достаточно большого числа двигающихся полуроботов.

Для конечного автомата (робота) не решена основная задача "объяснения" разрыва между длиной минимального обхода ориентированного графа $\boldsymbol{O}(nm)$ и обхода наилучшим из известных роботов с оценкой $\boldsymbol{O}(nm + n^2\log\log n)$. Прежде всего, не известно, существует ли робот, выполняющий обход с минимальной оценкой $\boldsymbol{O}(nm)$, и если не существует, то какова наилучшая возможная оценка для робота. Для двух взаимодействующих роботов достигается оценка $\boldsymbol{O}(nm)$, но неизвестно, может ли она быть улучшена и насколько при наличии большого числа роботов.

Если формально инвертировать модель неподвижных автоматов из данной статьи, то получится модель двигающихся автоматов, которая отличается от модели, описанной в работе [2], двумя свойствами. Во-первых, автоматы не обмениваются сообщениями между собой, а только через память вершин. Предполагается, что срабатывание автомата, при котором он читает из вершины, меняет свое состояние, записывает в вершину и указывает номер выходящей дуги, по которой будет двигаться, является атомарным действием. Таким образом осуществляется синхронизация по доступу к памяти вершин. Во-вторых, автоматы могут исчезать и генерироваться другими автоматами в любой вершине, а не только в корне (в работе [2] регуляторы, генерируемые в любой вершине, теряют возможность перемещаться по графу). Такая модель может стать темой дальнейших исследований.

Следует отметить, что рассматриваемые модели и полученные результаты в этих трех статьях не исчерпывают всей области автоматов на графах.

Список литературы

1. **Бурдонов И. Б., Косачев А. С.** Исследование графа автоматом // Программная инженерия. 2016. Т. 7, № 11. С. 498—508.
2. **Бурдонов И. Б., Косачев А. С.** Исследование графов коллективом двигающихся автоматов // Программная инженерия. 2016. Т. 7, № 12. С. 559—567.
3. **Бурдонов И. Б., Косачев А. С.** Построение прямого и обратного остовов автоматами на графе // Труды Института системного программирования РАН. 2014. Т. 26, № 6. С. 57—62.
4. **Бурдонов И. Б., Косачев А. С., Кулямин В. В.** Параллельные вычисления на графе // Программирование. 2015. № 1. С. 3—20.

5. **Кушнеренко А. Г., Лебедев Г. В.** Программирование для математиков. М.: Наука, Главная редакция физико-математической литературы, 1988. 382 с.
6. **Бурдонов И. Б., Косачев А. С., Кулямин В. В**. Параллельные вычисления автоматами на прямом и обратном остовах графа // Труды Института системного программирования РАН. 2014. Т. 26, № 6. С. 63—66.
7. **Бурдонов И. Б., Косачев А. С.** Мониторинг динамически меняющегося графа // Труды Института системного программирования РАН. 2015. Т. 27, № 1. С. 69—96.
8. **Бурдонов И. Б., Косачев А. С.** Параллельные вычисления на динамически меняющемся графе // Труды Института системного программирования РАН. 2015. Т. 27, № 2. С. 189—220.

Graph Learning by Group of Motionless Automata

Igor B. Bourdonov, igor@ispras.ru, **Alexander S. Kossatchev,** kos@ispras.ru,
Institute for System Programming RAS, Moscow, 109004, Russian Federation

Corresponding author:

Bourdonov Igor B., Leading Researcher, Institute for System Programming RAS, 109004, Moscow, Russian Federation,
E-mail: igor@ispras.ru

Received on August 01, 2016
Accepted on October 19, 2016

Graph learning or exploration tasks can be found in many applications, in particular, in exploration of networks, including Internet parts or GRIDs. For example, we can consider a WEB-page as a vertex, and each hyperlink as a directed arc. In this case an exploration of hypertext structure can be performed by executable agent tied with the vertices and communicating with each other by messages sent along the arcs.

This paper is third in a series of works devoted to graph exploration by automata. In the first and second papers we consider graph exploration tasks performed by automata moving along graph arcs. In the first paper we consider a single automaton, in the second one — several automata, which can communicate by messages sent through a network independent of the graph. In this paper we consider automata that are immobile and fixed in the graph vertices and communicate only by messages sent along directed graph arcs.

In the paper we investigate static and dynamic graph cases, and two tasks for each of the cases. While a static graph remains the same in the process of exploration, a dynamic graph can change — it's arcs can appear and disappear, and they can change their ends. The first task considered is the graph exploration task, which is performed to learn its structure. The second task considered is parallel computation on the graph, more specifically, computation of some function of a multiset of values stored in graph vertices.

Keywords: *directed graph, ordered graph, numerated graph, unknown graph, dynamic graph, graph learning, parallel computations, automaton, semirobot, automata group*

For citation:

Bourdonov I. B., Kossatchev A. S. Graph Learning by Group of Motionless Automata, *Programmnaya Ingeneria,* 2017, vol. 8, no. 1, pp. 16—25.

DOI: 10.17587/prin.8.16-25

References

1. **Bourdonov I. B.**, **Kossatchev A. S.** Issledovanie grafa avtomatom (Graph learning by automaton), *Programmnaya Inzheneria*, 2017, vol. 7, no. 11, pp. 498—508 (in Russian).
2. **Bourdonov I. B.**, **Kossatchev A. S.**. Issledovanie grafov kollektivom dvigajushhihsja avtomatov (Graph learning by a set of moving automata), *Programmnaya Inzheneria*, 2017, vol. 7, no. 12, pp. 559—567 (in Russian).
3. **Bourdonov I. B.**, **Kossatchev A. S.** Postroenie prjamogo i obratnogo ostovov avtomatami na grafe (Building direct and back spanning trees by automata on a graph), *Trudy ISP RAN*, 2014, vol. 26, no. 6, pp. 57—62 (in Russian).
4. **Bourdonov I. B.**, **Kossatchev A. S., Kulyamin V. V.** Parallel Computations on a Graph, *Programming and Computer Software,* 2015, vol. 41, no. 1, pp. 1—13.
5. **Kushnerenko A. G., Lebedev G. V.** *Programmirovanie dlja matematikov (Programming for mathematicians)*, Moscow, Nauka, 1988, 382 p. (in Russian).
6. **Bourdonov I. B., Kossatchev A. S., Kulyamin V. V.** Parallel'nye vychislenija avtomatami na prjamom i obratnom ostovah grafa (Parallel calculations by automata on direct and back spanning trees of a graph), *Trudy ISP RAN,* 2014, vol. 26, no. 6, pp. 63—66 (in Russian).
7. **Bourdonov I. B.**, **Kossatchev A. S.** Monitoring dinamicheski menjajushhegosja grafa (Monitoring of dynamically changed graph), *Trudy ISP RAN,* 2015, vol. 27, no. 1, pp. 69—96 (in Russian).
8. **Bourdonov I. B.**, **Kossatchev A. S.** Parallel'nye vychislenija na dinamicheski menjajushhemsja grafe (Parallel Calculations on Dynamic Graph), *Trudy ISP RAN,* 2015, vol. 27, no. 2, pp. 189—220 (in Russian).

УДК 004.89 DOI: 10.17587/prin.8.26-37

В. П. Тельнов, канд. техн. наук, доц., e-mail: telnov@bk.ru,
Институт атомной энергетики НИЯУ "МИФИ", Обнинск

Контекстный поиск как технология извлечения знаний в сети Интернет

Приведено описание пилотного проекта, посвященного созданию и применению в образовательной деятельности университетов поискового агента, который обеспечивает студентам и преподавателям средства контекстного поиска учебных ресурсов в глобальной сети. Предложена и реализована адаптивная технология систематизации и кластеризации найденного сетевого контента на основе бинарного отношения Парето при совместном учете многих аспектов, включая релевантность и пертинентность информации.

Ключевые слова: *информационные технологии, высшее образование, контекстный поиск, агент, Интернет, отношение Парето*

Актуальность проекта

Вполне закономерен вопрос: "Зачем нужен дополнительный поисковый агент, если существуют общедоступные поисковые системы (Google, Yandex, Yahoo и др.)?" Для ответа на него отметим несколько характерных особенностей популярных поисковых систем, которые хорошо известны большинству пользователей.

1. Найденные документы ранжируются поисковой машиной в соответствии с ее внутренним алгоритмом, который далеко не всегда отвечает интересам конкретного пользователя.

2. Пользователям не всегда удобно управлять контекстом поискового запроса, уточнять и направлять поиск.

3. Ссылки на коммерческие сайты обычно имеют больший рейтинг в сравнении с прочими результатами поиска. Такой результат достигается за счет использования так называемой поисковой оптимизации сайтов (SEO) [1] для искусственного поднятия позиций коммерческих сетевых ресурсов на страницах выдачи популярных поисковых систем в целях увеличения потока потенциальных клиентов для последующей монетизации трафика.

Представляется, что перечисленные выше обстоятельства и тенденции делают публичные поисковые системы все менее адекватным инструментарием извлечения знаний в сети Интернет для целей образования. Чтобы проиллюстрировать справедливость такого вывода в статье даны примеры первых страниц выдачи поисковых машин Yandex и Google при поиске по слову "граф" в сравнении с аналогичной выдачей агента "Контекстный поиск". Другой пример — с поиском по слову "технология" — приведен в работе [2]. Несмотря на растущую коммерциализацию публичных поисковых систем, с большой долей уверенности можно полагать, что они, наряду с Википедией, в обозримом будущем будут оставаться наиболее доступными "универсальными учебниками" для той многочисленной категории студентов, которые не всегда требовательны к качеству и полноте учебного материала.

Для многих пользователей сети Интернет основным средством получения информации продолжают оставаться публичные поисковые системы. Вместе с тем в сфере образования и науки существуют иные технологии извлечения знаний из мировой глобальной сети. Например, семантический веб в настоящее время интенсивно развивается в англоязычном сегменте всемирной паутины для обмена связанными открытыми данными [3—5].

Семантический веб (семантическая паутина) представляет собой общедоступную глобальную компьютерную сеть, сформированную на базе существующей всемирной паутины (сети Интернет) путем стандартизации представления данных в виде, удобном для их машинной обработки. Семантический веб состоит из машинно-читаемых элементов, так называемых узлов семантической сети, с опорой на онтологии. Благодаря этому свойству семантического веба компьютеры имеют возможность непосредственно, без участия человека, получать из сети Интернет утверждения вида "объект — тип взаимосвязи — другой объект" и вычислять по ним логические заключения, т. е. самостоятельно извлекать из сети элементы знаний.

Термин "связанные открытые данные" подразумевает такой способ представления и публикации информации в сети, когда отдельные фрагменты данных связываются друг с другом и становятся доступными компьютерам через так называемые семантические запросы. Для создания и поиска связанных открытых данных применяют специализированные семантические технологии, такие как RDF, OWL, SKOS, SPARQL [6] и т. д. При этих обстоятельствах выглядит оправданным известный скептицизм относительно того, что семантические интернет-технологии в их нынешнем виде получат признание

Рис. 1. Образовательный портал "Кафедра онлайн"

и широкое распространение среди тех студентов и преподавателей университетов, которые сами не специализируются в области информатики или в компьютерных дисциплинах [5].

Современные реалии российской высшей школы таковы, что подавляющее число учащихся не подозревают о существовании семантического веба и вообще связанных открытых данных. Они продолжают практиковать поиск информации во всемирной паутине по ключевым словам, используя для этого публичные поисковые системы. Немалую роль здесь играют традиции, а также простота и высокая скорость формирования поискового запроса, в сравнении с поисковыми запросами к семантической паутине.

В интересах этой обширной категории потребителей сетевой информации был создан поисковый агент "Контекстный поиск" (рис. 1), возможности которого представлены в настоящей статье. Агент "Контекстный поиск" интегрирован в образовательный портал "Кафедра онлайн" [7]. Доступ к нему возможен через главную страницу образовательного портала по сетевому адресу http://ksst.obninsk.ru или по прямой ссылке http://ksst.obninsk.ru/semantic.

Смежные работы и новизна

Внешне агент "Контекстный поиск" выглядит как метапоисковая система, поскольку использует ресурсы нескольких поисковых машин для увеличения вероятности нахождения нужной информации и обеспечения полноты результатов поиска. Если не рассматривать ранние (созданные до середины 2000-х гг.) сравнительно простые метапоисковые системы, то среди современных онлайн-сервисов, предназначенных для извлечения знаний из глобальной сети, следует упомянуть поисковый портал AskNet.ru, который по сути является самообучающейся аналитической вопросно-ответной системой. Портал AskNet.ru выполняет лингвистический анализ текстов (морфология, синтаксис, семантика) с применением онтологий, баз знаний и методов логического вывода. Другая интеллектуальная метапоисковая система Нигма.рф ведет собственную базу данных и осуществляет дополнительное индексирование найденного сетевого контента, позволяя пользователю уточнять поисковый запрос, группировать и фильтровать результаты поиска по заданным темам.

Отдельного упоминания заслуживает поисково-аналитическая система поддержки научно-образовательной деятельности Exactus Expert, которая предназначена для исследования конкретных предметных областей и анализа качества научных публикаций. Особенность Exactus Expert состоит в том, что контент извлекается не из глобальной сети в режиме онлайн, а из собственной коллекции документов объемом около 2 млн единиц. Пользовательский интерфейс системы при этом расположен в так называемом "глубоком вебе".

Рассматриваемый агент "Контекстный поиск" дополняет линейку интеллектуальных метапоисковых систем, предлагая адаптивную технологию упорядочения и кластеризации найденного сетевого контента на основе бинарного отношения Парето при совместном учете многих аспектов, включая релевантность и пертинентность информации [8]. Нечеткое сравнение текстов выполняется с использованием расстояния Левенштейна [9]. Контекст поиска в этом агенте понимается не как традиционный формальный триплет вида "объекты, атрибуты, инциденции" [10], а как некоторое объединение таксономий и ключевых слов образовательного портала "Кафедра онлайн", а также произвольных текстов и сетевых документов. Пользователь может гибко управлять этим объединением в процессе осуществления поиска.

Поисковые машины

В основе пилотного проекта "Контекстный поиск" лежит простая идея — создать такого посредника (поискового агента) между учащимся и публичными поисковыми системами, который позволит упорядочивать и систематизировать результаты поиска в соответствии с образовательными потребностями конкретного человека, эффективно фильтруя неадекватный контент и информационный мусор. При этом ставится задача в максимальной степени задействовать мощность современных поисковых машин, включая встроенные языки запросов и другие средства управления поиском.

При работе агента "Контекстный поиск" глобальный поиск документов, также как и поиск по специализированным ресурсам, изначально выполняется штатными поисковыми машинами Google Ajax Search, Yandex, Yahoo, Mail.ru, взаимодействие с которыми по сети происходит асинхронно через стандартный API и динамический пул прокси-серверов, который размещается на облачной платформе Google App Engine [11] (рис. 2).

Результаты работы штатных поисковых машин являются своего рода "сырьем" для дальнейшей обработки. Прокси-серверы на облачной платформе

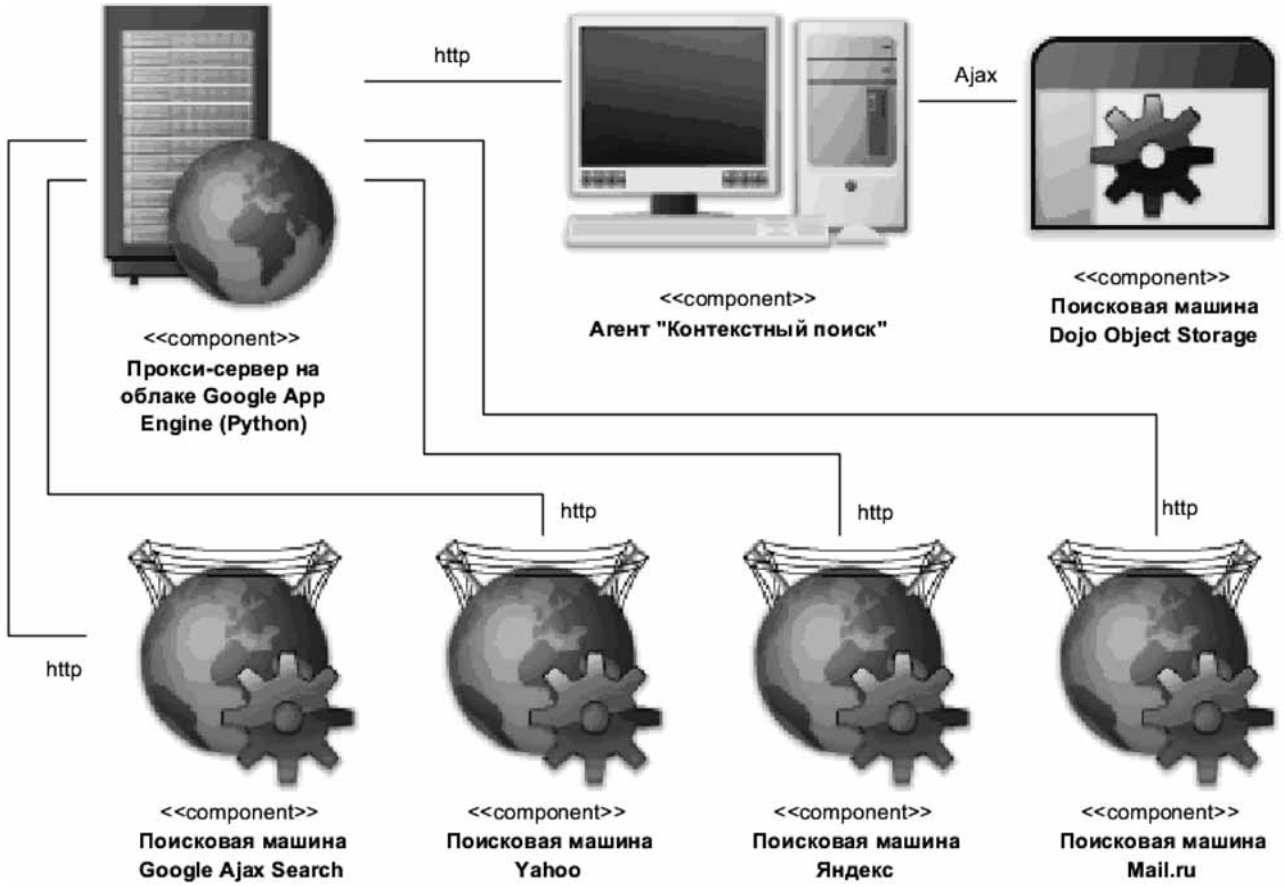

Рис. 2. Укрупненная диаграмма компонентов для агента "Контекстный поиск"

Google App Engine выполняют парсинг этих результатов и формируют фиды (feeds) по стандарту Atom [12] в формате JSON, которые далее поступают на клиентский компьютер в JavaScript-движок агента "Контекстный поиск", где из них (фидов) формируются сниппеты [13]. Эти сниппеты перед тем как появиться на мониторе клиентского компьютера проходят дополнительную обработку, селекцию и сортировку, как описано далее в статье. В частности, для каждого сниппета вычисляется его релевантность, пертинентность и ряд других показателей (аспектов), которые далее используются для систематизации и кластеризации результатов поиска.

Особым образом реализован поиск документов, которые представлены в "Облачном кабинете" (репозитории) образовательного портала "Кафедра онлайн" (http://ksst.obninsk.ru/cloud). Число сущностей в "Облачном кабинете" может исчисляться многими тысячами единиц. Образовательные ресурсы физически расположены в удаленных хранилищах данных где угодно в глобальной сети, поэтому не всегда индексируются публичными поисковыми машинами. Непосредственно в "Облачном кабинете" хранятся метаданные [14] соответствующих образовательных объектов. Доступ ко многим объектам изначально ограничен (например, только для студентов, для преподавателей, для членов проектных групп). Навигация по таким объектам возможна как непосредственно через деревья "Облачного кабинета", так и с помощью агента "Контекстный поиск". Фактическая работа с деревьями репозитория при этом выполняется внутренней поисковой машиной Dojo Object Storage [15, 16].

Контекст поиска

Язык запросов некоторых поисковых машин (Google, Yandex, Yahoo) [17] может включать в себя так называемый "поисковый контекст". Речь идет об использовании непосредственно в тексте запроса специальных операторов, которые позволяют уточнять наличие и взаимное расположение конкретных слов в искомых документах. В настоящей статье под "контекстом поиска" понимается иное, а именно — некий ограниченный по размеру текст, характеризующий ту предметную область, которая в текущий момент представляет интерес для пользователя. При формировании контекста поиска доступны следующие источники данных: таксономии, ключевые слова, категории и свойства образовательного портала, файлы с клиентского компьютера, произвольные сетевые документы. Определенный таким образом контекст позволяет выполнять селекцию, сорти-

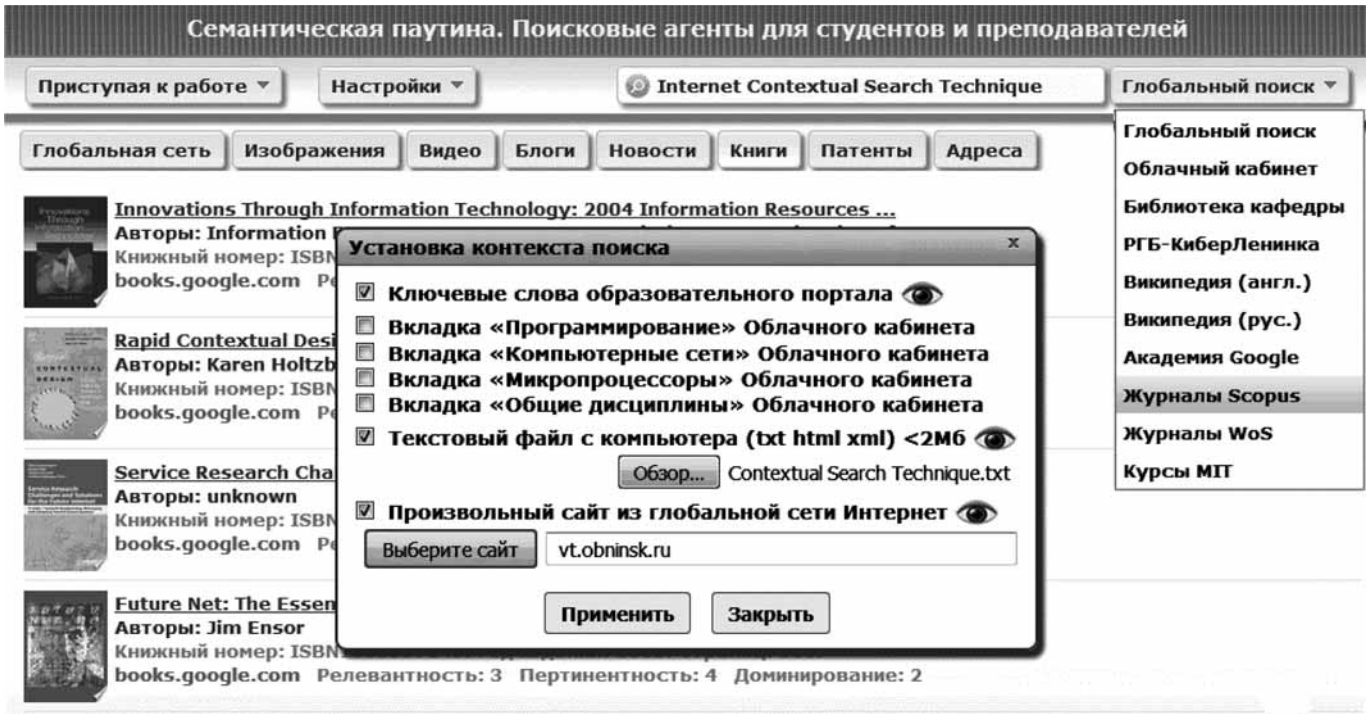

Рис. 3. Установка контекста поиска (пример)

ровку и систематизацию информации, которая поступает от поисковых машин через прокси-серверы. На рис. 3 показаны возможные варианты установки контекста поиска.

Простейший способ задать контекст поиска — это задействовать ключевые слова образовательного портала, через который в текущий момент времени работает пользователь (студент, преподаватель). Фактически, это все слова из тегов <meta> HTML-кода текущей веб-страницы портала. Таким образом, заданный контекст поиска является достаточно широким и может характеризовать лишь основные направления специализации кафедры (факультета).

Более совершенный механизм установки контекста поиска подразумевает использование "Облачного кабинета" образовательного портала "Кафедра онлайн" (http://ksst.obninsk.ru/cloud). Номенклатура дисциплин кафедры (факультета) с соответствующими таксономиями хранится во вкладках (деревьях) "Облачного кабинета". Во вкладках, выбранных пользователем, будут просканированы все имеющиеся образовательные объекты, из метаданных этих объектов в контекст поиска будут включены наименования документов и соответствующие ключевые слова. Такой способ задания контекста поиска позволяет более точно очертить предметную область (или комбинацию областей), например, определить конкретные направления подготовки студентов или специализации кафедры.

Далее, контекст поиска может быть задан еще более точно, путем указания текстового файла на клиентском компьютере (форматы TXT, HTML, XML). Это может быть файл с текстом статьи по интересующей тематике, нужный раздел учебника или другой подобный материал. Теги HTML, XML, стили CSS и код JavaScript, если они обнаружатся в тексте, автоматически вычищаются.

Наконец, контекст поиска можно задать в режиме онлайн, выбрав и указав определенный сайт или документ в сети Интернет. Соответствующий сетевой ресурс будет получен через прокси-сервер. При этом из него будут удалены теги HTML, XML, стили CSS и код JavaScript, получившийся текст будет добавлен в контекст поиска. На рис. 3 приведен пример установки контекста поиска с использованием ключевых слов образовательного портала, текстового файла с клиентского компьютера и сетевого документа.

Для файлов с клиентского компьютера и произвольных сайтов из сети Интернет автоматически распознается кодировка русскоязычных текстов. Допускаются любые комбинации из перечисленных выше способов установки контекста поиска. Результирующий контекст является объединением выбранных опций.

Релевантность, пертинентность, метрики

В контексте настоящей статьи под релевантностью [8] сниппета поисковой системы понимается мера соответствия текста сниппета тексту поискового запроса. Под пертинентностью [8] сниппета поисковой системы понимается мера соответствия текста сниппета контексту поиска, который был определен ранее. Эти и другие меры вычисляют путем нечеткого сравнения соответствующих текстов. Для количественной

оценки этих мер в агенте "Контекстный поиск" применяют метрики (расстояния) Левенштейна [9]. Алгоритм вычисления релевантности одного конкретного сниппета выглядит следующим образом.

Каждую лексическую единицу [17] (лексема) из текста сниппета последовательно сравнивают с каждой лексемой из текста поискового запроса. В случае полного совпадения лексем к релевантности сниппета добавляют число 3. Если для полного совпадения лексем требуется применение одной из операций Левенштейна (вставка, удаление, замена одного символа), тогда к релевантности сниппета добавляют число 2, а не 3 (2 = 3 − 1). Здесь число 1 есть цена одной операции. Если для полного совпадения лексем требуется применение двух операций, тогда к релевантности сниппета добавляют число 1 (1 = 3 − 2). В случае если для совпадения лексем потребуется более двух операций Левенштейна, релевантность сниппета не увеличивается вовсе.

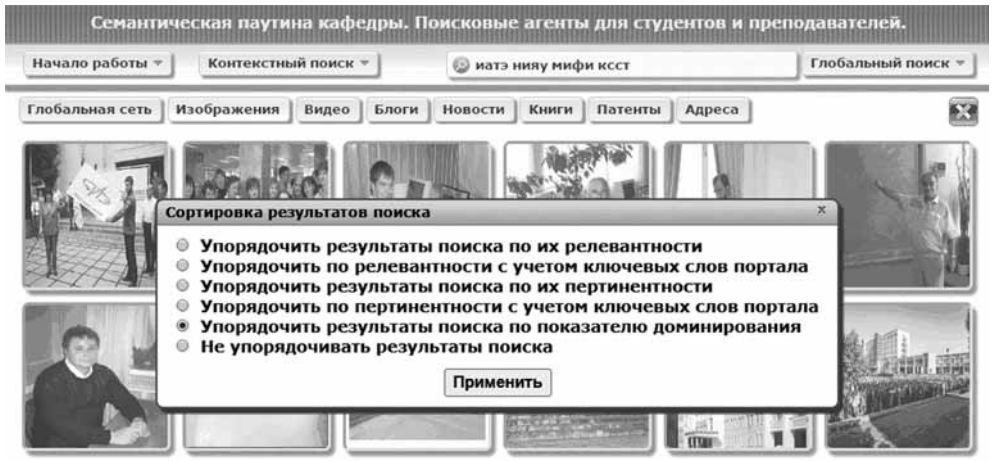

Рис. 4. Выбор вариантов сортировки результатов поиска (пример)

Будем считать, что каждый сниппет $x \in \Omega$ характеризуется конечным набором аспектов. Обозначим $A = \{1, \ldots, m\}$ – множество номеров аспектов, учитываемых для сниппетов. Для каждой пары сниппетов (x, y) определим семейство функций $\alpha_j(x, y)$:

$$\alpha_j(x,y) = \left\{ \begin{array}{l} 1, \text{ если } x \text{ превосходит } y \text{ по } j\text{-му аспекту} \\ 0, \text{ если } y \text{ превосходит } x \text{ по } j\text{-му аспекту} \end{array} \right\}; \quad j \in A;$$

Пусть имеется два сниппета $x, y \in \Omega$. Обозначим

$d(y,x) = \sum_{j \in A} \alpha_j(y,x)$ – число аспектов, по которым y превосходит x. Тогда величина

$\mathbf{D}_\Omega(x) = \max_{y \in \Omega} d(y,x)$ есть показатель доминирования x при предъявлении Ω.

Рис. 5. Вычисление показателей доминирования

Допускается тонкая пользовательская настройка цен (весов) операций Левенштейна каждого вида, которые изначально (по умолчанию) все равны единице. Алгоритм вычисления пертинентности сниппетов выглядит аналогичным образом с той лишь разницей, что каждая лексема из текста сниппета последовательно сравнивается с каждой лексемой из контекста поиска. Как видно из приведенных выше описаний алгоритмов, процесс вычисления релевантности и пертинентности сниппетов является формальным, без анализа возможных связей отдельных лексем и их словарного окружения. Предполагается, что такой анализ в той или иной мере был выполнен в ходе первичного поиска документов и их полнотекстовом индексировании в базах данных штатных поисковых машин (Google, Yandex, Yahoo, Mail.ru).

На рис. 4 показаны варианты сортировки результатов поиска в итоговой выдаче агента "Контекстный поиск". Заслуживает отдельного упоминания такой аспект, как "показатель доминирования" (см. вторую снизу строку панели на рис. 4), который обеспечивает совместный учет значений многих метрик, характеризующих адекватность данных. Например, показатель доминирования кроме релевантности и пертинентности сниппетов может интегрально учитывать также меру соответствия текста сниппета ключевым словам, категориям и свойствам образовательного портала.

Для практического вычисления значений показателя доминирования представляется целесообразным использовать формализм отношений Парето, поскольку многокритериальное ранжирование по Парето не предполагает априорного знания относительной важности критериев (например, что важнее, релевантность или пертинентность?). Соответствующий математический аппарат дан в работе [18], вычислительные алгоритмы опубликованы в работе [19], расчетные формулы показаны на рис. 5.

Сниппеты, имеющие максимальное значение показателя доминирования, образуют так называемое "множество Парето". Множество Парето включает в себя сниппеты, наилучшие по совокупности всех учтенных метрик, включая релевантность и пертинентность. Группы сниппетов с одинаковым значением показателя доминирования образуют кластеры, которые в итоговой выдаче поискового агента расположены в порядке убывания этого показателя.

В качестве иллюстрации к предыдущим утверждениям на рис. 6 приведен пример работы агента "Контекстный поиск". Сниппеты отсортированы по убыванию значения показателя доминирования при совместном учете шести метрик, включая релевантность и пертинентность данных. Когда сниппеты упорядочены по значению показателя доминирования, в пределах групп элементов с одинаковой величиной показателя доминирования (т. е. в пределах кластера) сниппеты упорядочиваются по каждой из метрик, учтенной в расчетах. Доступны иные способы упо-

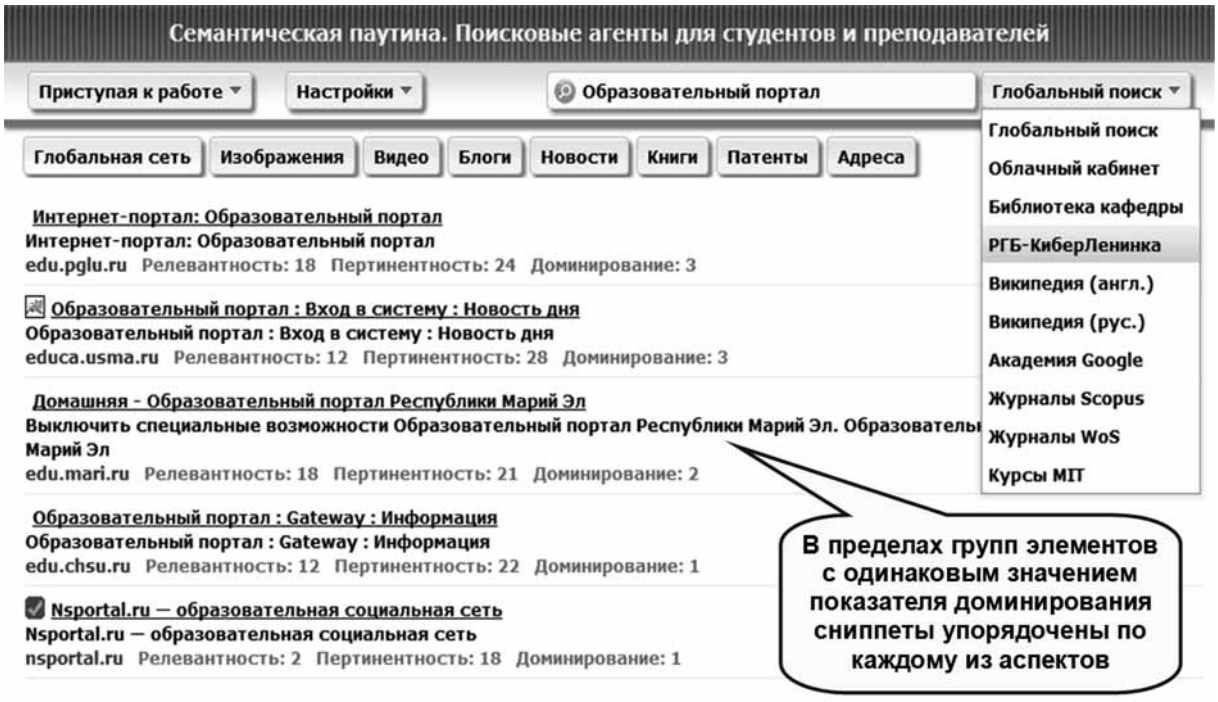

Рис. 6. Результаты поиска, отсортированные по значению показателя доминирования (пример)

рядочения и систематизации найденного сетевого контента при любых комбинациях метрик, которые характеризуют адекватность найденных данных.

Сравнение результатов работы поисковых систем

Приведем примеры сравнения результатов работы поисковых машин Yandex и Google с аналогичными результатами работы агента "Контекстный поиск". На рис. 7—10 приведены первые страницы выдачи этих поисковых систем, полученные в ходе первичного поиска в глобальной сети учебных материалов из области теории графов. Текст поискового запроса с уточняющими словами имел следующий вид: "+граф алгебра группа математика теория терция".

На рис. 7 и 8 показаны результаты чистого эксперимента, когда была обнулена история запросов, т. е. сброшены все "куки" (cookies) и "кеш" (cash) браузера, а вход в сеть Интернет был выполнен со случайного IP-адреса. Это важно для корректности сравнения, поскольку поисковые системы Yandex и Google хранят историю поисковых запросов своих пользователей и пытаются выдавать контент, схожий с тем, который пользователь искал ранее.

На рис. 9 и 10 показаны примеры выдачи агента "Контекстный поиск", когда поиск материалов из области теории графов вновь был выполнен поисковой машиной Yandex, но при посредничестве агента "Контекстный поиск", который работал в составе образовательного портала кафедры "Компьютерные системы, сети и технологии" НИЯУ МИФИ. Предметной областью, представляющей интерес для пользователя, в данном случае являются компьютерные науки и информатика.

Применительно к результатам, представленным на рис. 9, контекст поиска был задан непосредственно в тексте запроса и уточнен через ключевые слова образовательного портала, подробнее см. верхнюю строку панели на рис. 3. Эти ключевые слова характеризуют кафедру "Компьютерные системы, сети и технологии" и имеют отношение к компьютерным дисциплинам. На рис. 9 видно, что "граф" как математическая сущность достаточно полно представлен в выдаче агента "Контекстный поиск". Более половины ссылок, представленных на рис. 9, ведут к учебным материалам университетов, что в полной мере отвечает информационным потребностям студентов.

На рис. 10 показана выдача агента "Контекстный поиск", когда контекст был задан сетевым документом https://ru.wikipedia.org/wiki/Теория_графов. Непосредственно в тексте поискового запроса указано единственное слово "граф". На рис. 10 видно, что результаты поиска упорядочены по убыванию их пертинентности, причем все показанные сниппеты вполне адекватны образовательному контексту.

Заключительные замечания

Агент "Контекстный поиск" является частью программного компонента "Семантическая паутина" (http://ksst.obninsk.ru/semantic) наряду с двумя другими агентами семантического поиска, которые имеют дело с международными базами знаний

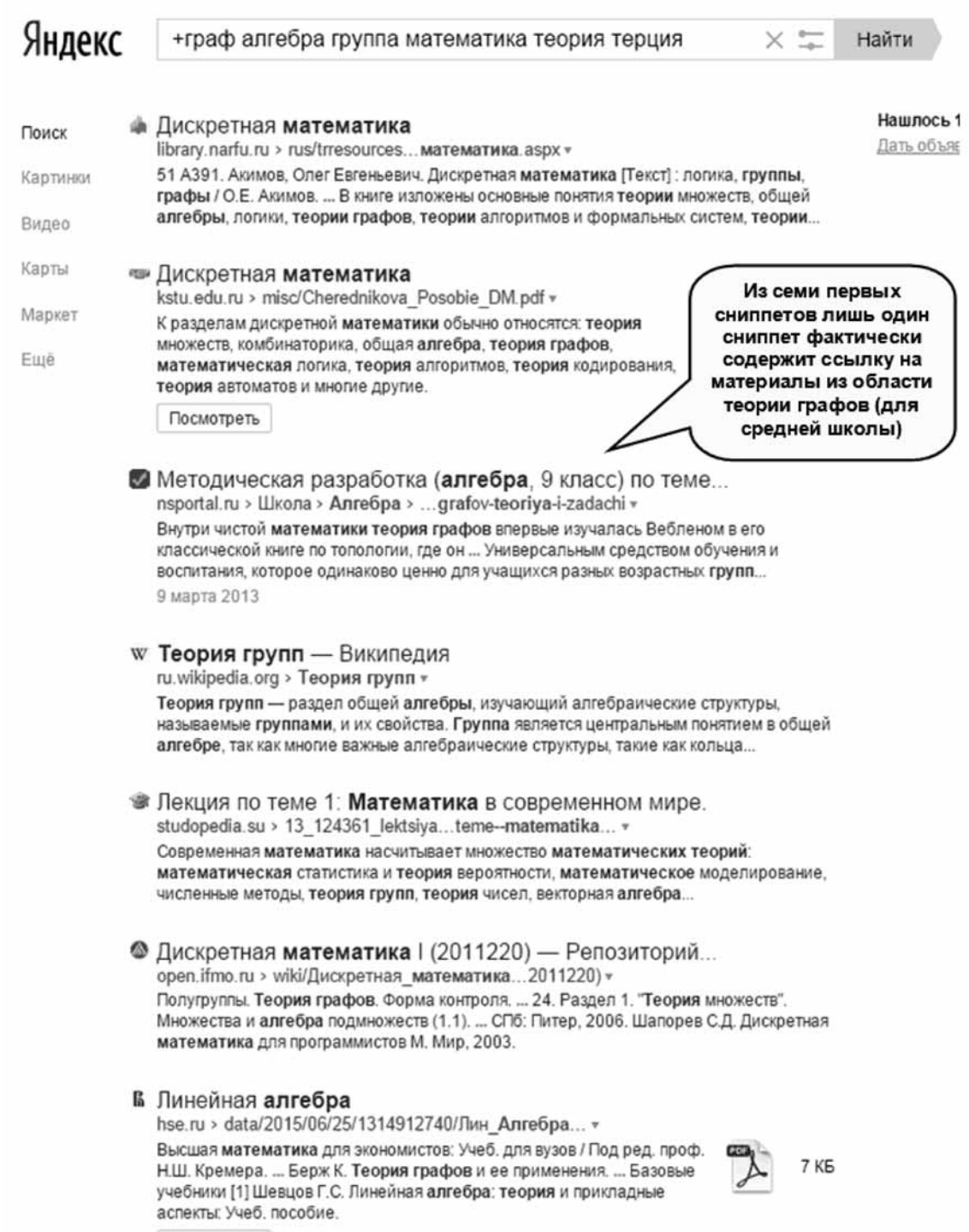

Рис. 7. Результаты работы Yandex при поиске материалов из области теории графов. Контекст поиска задан непосредственно в тексте запроса

"DBpedia" [20] и "Wikidata" [21]. Компонент "Семантическая паутина", в свою очередь, входит в состав образовательного портала "Кафедра онлайн" [7]. В течение 2016 г. силами студентов НИЯУ МИФИ [22] выполнялось тестирование производительности агента "Контекстный поиск". После обработки собранной статистики была получена оценочная регрессионная зависимость. Обозначим символом a среднее время отклика штатной поисковой машины (Google, Yandex, Yahoo, Mail.ru) на стандартный поисковый запрос. Обычно a имеет значение около 0,5 с. Тогда, при одновременной работе десяти пользователей с интенсивностью до десяти транзакций в минуту среднее время отработки

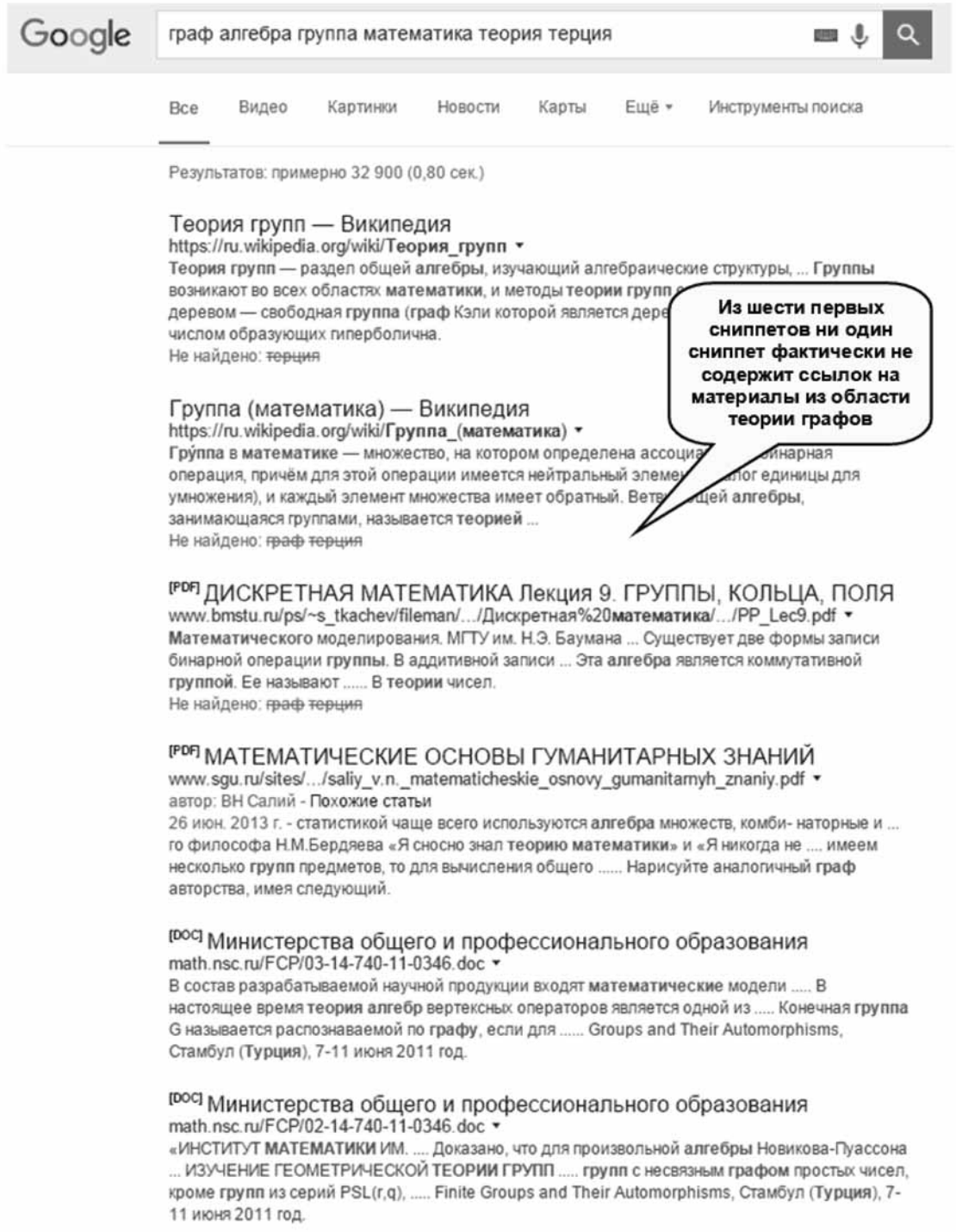

Рис. 8. Результаты работы Google при поиске материалов из области теории графов. Контекст поиска задан непосредственно в тексте запроса

стандартного поискового запроса агентом "Контекстный поиск" может быть оценено в секундах по следующей формуле:

$$4(a + b) + 2,$$

где b — оценка среднего времени прохождения сетевого POST-запроса (протокол HTTP) от клиентского компьютера к прокси-серверу и обратно, см. рис. 2. Общее время отклика агента "Контекстный поиск" на типичный поисковый запрос обычно находится в пределах 3...5 с.

Рис. 9. Результаты работы агента "Контекстный поиск" при поиске материалов из области теории графов. Контекст поиска задан непосредственно в тексте запроса

Масштабируемость серверной части рабочего кода поискового агента обеспечивается непосредственно облачной платформой Google App Engine [11]. В ходе нагрузочного тестирования было обнаружено, что при интенсивной работе поискового агента могут возникать ощутимые "пробуксовки" прокси-серверов, что обусловлено ограниченностью пула свободных IP-адресов на облачном сервисе.

Работа выполнена при финансовой поддержке НБО "Благотворительный фонд В. Потанина", проект № ГК160001360.

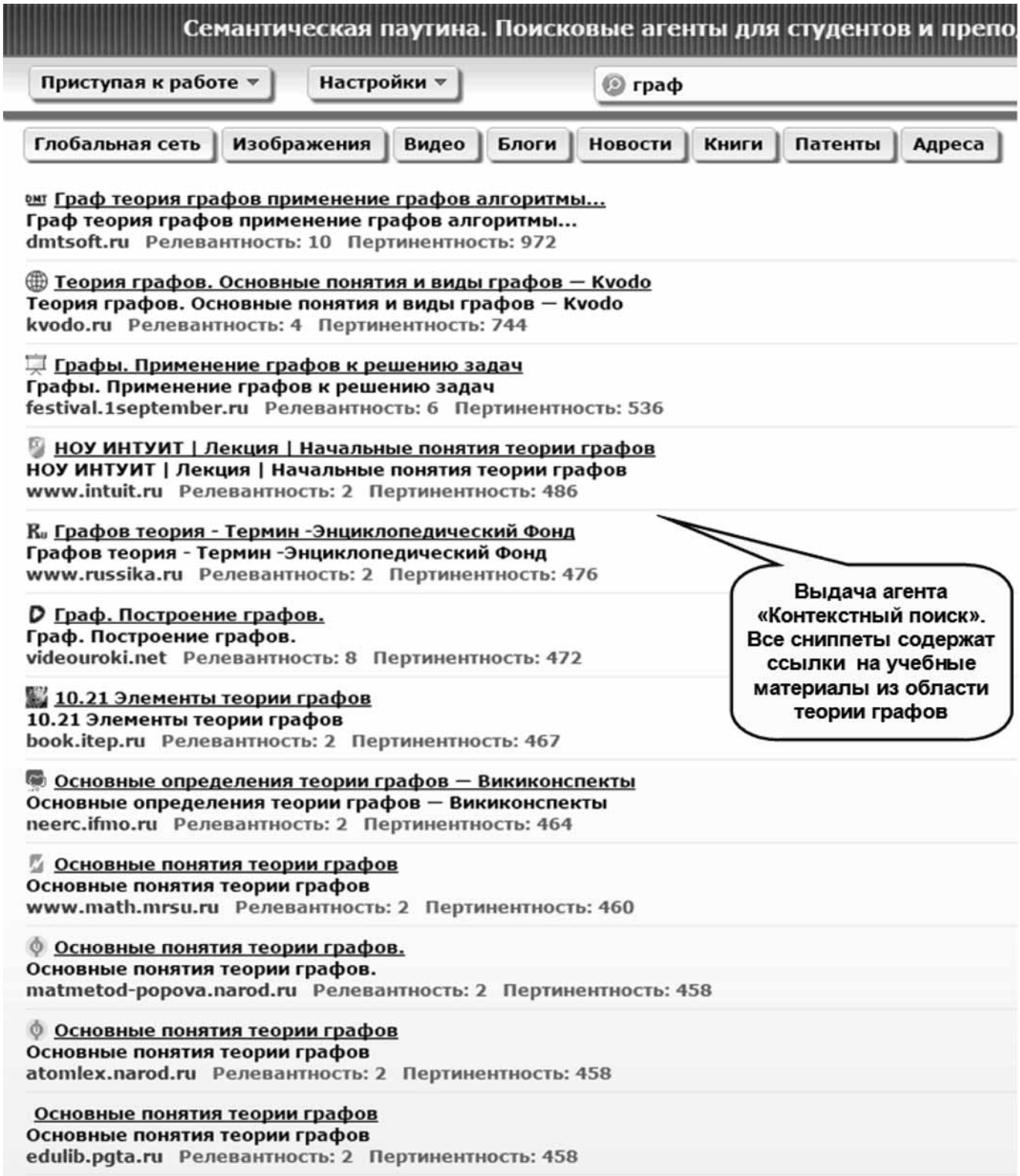

Рис. 10. Результаты работы агента "Контекстный поиск" при поиске материалов из области теории графов. Контекст поиска задан сетевым документом https://ru.wikipedia.org/wiki/Теория_графов

Список литературы

1. **Поисковая** оптимизация. URL: http://dic.academic.ru/dic.nsf/ruwiki/106655 (дата обращения 10.08.2016).
2. **Telnov V.** Semantic Web and Search Agents for Russian Higher Education. A Pilot Projec//CEUR Workshop Proceedings. Vol-1536. Selected Papers of the XVII International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2015). Obninsk, Russia, October 13-16, 2015 / Eds. L. Kalinichenko, S. Starkov, 2015. P. 195—204. URL: http://ceur-ws.org/Vol-1536/paper29.pdf (дата обращения 10.08.2016).
3. **Berners-Lee T.** Linked Data. URL: http://www.w3.org/DesignIssues/LinkedData.html (дата обращения 10.08.2016).
4. **EUDAT:** the collaborative Pan-European infrastructure providing research data services, training and consultancy. URL: https://www.eudat.eu/ (дата обращения 10.08.2016).
5. **Тельнов В. П., Мышев А. В.** Семантическая паутина и поисковые агенты для высшей школы // Программная инженерия. 2015. № 9. С. 43—48.
6. **SPARQL.** URL: https://ru.wikipedia.org/wiki/SPARQL (дата обращения 10.08.2016).
7. **Тельнов В. П., Мышев А. В.** "Кафедра онлайн": облачные технологии в высшем образовании // Программные продукты и системы. 2014. № 4. С. 166—172.
8. **Поиск** и распространение информации. Термины и определения // ГОСТ 7.73—96 СИБИД. 1998. С. 3.5.1—3.5.2. URL: http://docs.cntd.ru/document/gost-7-73-96-sibid (дата обращения 10.08.2016).
9. **Задача** о редакционном расстоянии, алгоритм Вагнера-Фишера. URL: http://neerc.ifmo.ru/wiki/index.php?title=Задача_о_редакционном_расстоянии,_алгоритм_Вагнера-Фишера (дата обращения 10.08.2016).
10. **Анализ** формальных понятий. URL: http://www.machinelearning.ru/wiki/index.php?title=Анализ_формальных_понятий (дата обращения 10.08.2016).
11. **Google** App Engine. URL: https://cloud.google.com/appengine/ (дата обращения 10.08.2016).
12. **Atom** (standard). URL: https://en.wikipedia.org/wiki/Atom_%28standard%29 (дата обращения 10.08.2016).
13. **Сниппет.** URL: https://ru.wikipedia.org/wiki/Сниппет (дата обращения 10.08.2016).
14. **IEEE** 1484.12.1—2002. Learning Object Metadata standard. URL: http://en.wikipedia.org/wiki/Learning_object_metadata (дата обращения 10.08.2016).
15. **Dojo** Toolkit. URL: http://dojotoolkit.org (дата обращения 10.08.2016).
16. **W3C** Candidate Recommendation. IndexedDB object store API. URL: http://www.w3.org/TR/IndexedDB/#object-store (дата обращения 10.08.2016).
17. **Информационно-поисковые языки.** Термины и определения // ГОСТ 7.74—96 СИБИД. 1996. С. 2.1, 3.1.1. URL: http://docs.cntd.ru/document/gost-7-74-96-sibid (дата обращения 10.08.2016).
18. **Макаров И. М., Виноградская Т. М., Рубчинский А. А., Соколов В. Б.** Теория выбора и принятия решений: учеб. для вузов. М.: Наука, 1982. С. 28—30.
19. **Тельнов В. П., Мышев А. В.** Задачи выбора в оценочной деятельности // Программные продукты и системы. 2013. № 2. С. 159—165.
29. **DBpedia**. URL: https://ru.wikipedia.org/wiki/DBpedia (дата обращения 10.08.2016).
21. **Wikidata**. URL: http://www.wikidata.org (дата обращения 10.08.2016).
22. **Институт** атомной энергетики НИЯУ МИФИ. URL: http://www.iate.obninsk.ru/ (дата обращения 10.08.2016).

The Contextual Search as a Technique for Extracting Knowledge on the Internet

V. P. Telnov, telnov@bk.ru, National Research Nuclear University "MEPhI", Obninsk, 249040, Russian Federation

Corresponding author:

Telnov Victor P., Associate Professor, National Research Nuclear University "MEPhI", 249040, Obninsk, Russian Federation
E-mail: telnov@bk.ru

Received on August 11, 2016
Accepted on September 01, 2016

The subject of this paper is a pilot project devoted to creation and use of a retrieval agent in educational activity of university, that provides to students, teachers and professors a means of contextual search of educational resources on the Internet. It describes an agent that provides a process for the extraction of training materials and other data from the World Wide Web and their systematization. The primary search of content on the Internet is carried out by the known search engines (Google Ajax Search, Yandex, Yahoo, Mail.ru).

An adaptive technology of sorting and clustering of the found network content on the basis of the Pareto dominance relation under the joint consideration of many aspects (relevance, pertinence, etc.) has been offered and implemented. Fuzzy text comparison is performed with use of the Levenshtein distance.

Key architectural, technological and design solutions are presented. Samples of the user interface are shown. Results of testing the software productivity ware are given. The article is aimed at a wide range of professionals who are interested in the application of contextual search on the Internet, as well as university students and teachers, who specialize in the field of informatics and computer science.

Keywords: *information technologies, higher education, contextual search, agent, Internet, Pareto dominance relation*

Acknowledgements: *The work was supported by the NBO "Vladimir Potanin Charity Fund", project № ГК160001360*

For citation:

Telnov V. P. The Contextual Search as a Technique for Extracting Knowledge on the Internet, *Programmnaya Ingeneria*, 2017, vol. 8, no. 12, pp. 26—37.

DOI: 10.17587/prin.8.26-37

References

1. **Poiskovaja** optimizacija (Search Engine optimization), available at: http://dic.academic.ru/dic.nsf/ruwiki/106655 (in Russian).

2. **Telnov V. P.** Semantic Web and Search Agents for Russian Higher Education. A Pilot Project, *CEUR Workshop Proceedings*, 2015, vol. 1536, Selected Papers of the XVII International Conference on Data Analytics and Management in Data Intensive Domains (DAMDID/RCDL 2015), Obninsk, Russia, October 13—16, 2015 / Ed. L. Kalinichenko, S. Starkov, 2015, pp. 195—204, available at: http://ceur-ws.org/Vol-1536/paper29.pdf (in Russian)

3. **Berners-Lee T.** Linked Data, available at: http://www.w3.org/DesignIssues/LinkedData.html.

4. **EUDAT**: the collaborative Pan-European infrastructure providing research data services, training and consultancy, available at: https://www.eudat.eu/

5. **Telnov V. P., Myshev A. V.** Semanticheskaja pautina i poiskovye agenty dlja vysshej shkoly (The Semantic Web and Search Agents for High School), *Programmnaya Ingeneria,* 2015, no. 9, pp. 43—48 (in Russian).

6. **SPARQL,** available at: https://en.wikipedia.org/wiki/SPARQL.

7. **Telnov V. P., Myshev A. V.** "Kafedra onlajn": oblachnye tehnologii v vysshem obrazovanii ("Online Chair": Cloud Technology in Higher Edication), *Programmnye Produkty i Sistemy*, 2014, no. 4, pp. 166—172 (in Russian).

8. **Poisk** i rasprostranenie informacii. Terminy i opredelenija (Search and Distribution of Information. Terms and Definitions), *GOST 7.73-96 SIBID*, 1998, pp. 3.5.1—3.5.2, available at: http://docs.cntd.ru/document/gost-7-73-96-sibid (in Russian).

9. **Zadacha** o redakcionnom rasstojanii, algoritm Vagnera-Fishera (Task about Editorial Distance, Wagner-Fischer Algorithm), available at: http://neerc.ifmo.ru/wiki/index.php?title=Задача_о_редакционном_расстоянии,_алгоритм_Вагнера-Фишера (in Russian).

10. **Analiz** formal'nyh ponjatij (Formal Concept Analysis), available at: http://www.machinelearning.ru/wiki/index.php?title=Анализ_формальных_понятий (in Russian).

11. **Google** App Engine, available at: https://cloud.google.com/appengine/

12. **Atom** (standard), available at: https://en.wikipedia.org/wiki/Atom_%28standard%29.

13. **Snippet,** available at: https://ru.wikipedia.org/wiki/Сниппет (in Russian).

14. **IEEE** 1484.12.1—2002 Standard for Learning Object Metadata, available at: http://en.wikipedia.org/wiki/Learning_object_metadata.

15. **Dojo** Toolkit, available at: http://dojotoolkit.org.

16. **W3C** Recommendation. Indexed Database API, available at: http://www.w3.org/TR/IndexedDB/#object-store.

17. **Informacionno-poiskovye jazyki**. Terminy i opredelenija (Information Retrieval Languages. Terms and Definitions), *GOST 7.74—96 SIBID*, 1996. pp. 2.1, 3.1.1, available at: http://docs.cntd.ru/document/gost-7-74-96-sibid (in Russian).

18. **Makarov I. M., Vinogradskaja T. M., Rubchinskij A. A., Sokolov V. B.** *Teorija vybora i prinjatija reshenij*: ucheb. dlja vuzov (The Theory of Choice and Decision-Making: proc. for universities), Moscow, Nauka, 1982, pp. 28-30 (in Russian).

19. **Telnov V. P., Myshev A. V.** Zadachi vybora v ocenochnoj dejatel'nosti (Tasks of Choice in Valuation Activities), *Programmnye Produkty i Sistemy (Software and Systems)*, 2013, no. 2, pp. 159—165 (in Russian).

20. **DBpedia,** available at: http://wiki.dbpedia.org/

21. **Wikidata**, available at: http://www.wikidata.org/

22. **Institut** atomnoj jenergetiki NIJaU MIFI (Institute for Nuclear Power Engineering of NRNU MEPhI), available at: http://www.iate.obninsk.ru/ (in Russian).

ИНФОРМАЦИЯ

3—7 апреля 2017 г. в Казанском (Приволжском) федеральном университете состоится международная научная конференция, одиннадцатая в серии ежегодных конференций, посвященных развитию и применению параллельных вычислительных технологий в различных областях науки и техники

"Параллельные вычислительные технологии (ПаВТ) 2017"

Главная цель конференции — предоставить возможность для обсуждения перспектив развития параллельных вычислительных технологий и представления результатов, полученных ведущими научными группами в использовании суперкомпьютерных технологий для решения задач науки и техники.

Тематика конференции покрывает все аспекты применения высокопроизводительных вычислений в науке и технике, включая приложения, аппаратное и программное обеспечение, специализированные языки и пакеты.

В первый день работы конференции будет объявлена 26-я редакция списка Top50 самых мощных компьютеров СНГ.

Во все дни работы конференции будет действовать суперкомпьютерная выставка, на которой ведущие производители аппаратного и программного обеспечения представят свои новейшие разработки в области высокопроизводительных вычислений.

Официальный сайт конференции: http://agora.guru.ru/pavt2017/

УДК 629.7.058.74 DOI: 10.17587/prin.8.38-46

Е. В. Книга[1], канд. техн. наук, вед. инженер, e-mail: ekovinskaya@gmail.com,
А. В. Шукалов, канд. техн. наук, первый зам. ген. директора — гл. конструктор[1], доц.[2], e-mail: aviation78@mail.ru, **А. В. Гурьянов**[1], ген. директор, e-mail: postmaster@elavt.spb.ru,
И. О. Жаринов, д-р техн. наук, доц., руководитель уч.-науч. центра[1], зав. кафедрой[2], e-mail: igor_rabota@pisem.net, **О. О. Жаринов**[3], канд. техн. наук, доц., e-mail: zharinov73@hotbox.ru
[1] АО "ОКБ "Электроавтоматика", г. Санкт-Петербург,
[2] Университет ИТМО, г. Санкт-Петербург,
[3] Санкт-Петербургский государственный университет аэрокосмического приборостроения (ГУАП)

Программно-алгоритмическое обеспечение для решения практической задачи поворота графических изображений в авиационном приборостроении

Рассмотрена практическая задача поворота графических изображений, актуальная для разработки бортовых систем картографической информации в частности и для авиационного приборостроения в целом. Проведено исследование различных математических алгоритмов поворота растрового изображения, ориентированных на их последующую схемотехническую реализацию на базе компонентов программируемых логических интегральных схем, входящих в состав графических модулей авионики. Представлены результаты математического моделирования рассмотренных алгоритмов поворота изображения и фрагменты кода программы в программном пакете MATLab. Представлены тестовые графические изображения и фрагменты электрических схем (прошивок), предназначенных для реализации алгоритмов поворота изображения при моделировании процессов в аппаратно-ориентированном программном пакете синтезирования логики Altera Quartus II. Получены временны́е и топологические характеристики работы различных алгоритмов на стендовом макете бортовой авиационной аппаратуры.

Ключевые слова: *графические модули, авионика, поворот изображения, алгоритмы, программы, Оуэн-Македон*

Введение

Информационно-управляющее поле кабины пилота летательного аппарата (ЛА) определяет интерфейс взаимодействия экипажа и вычислительной системы самолетовождения, входящей в состав пилотажно-навигационного комплекса. Наиболее информативным для летного состава является изображение индикационных кадров авионики, выводимое на жидкокристаллический или светодиодный экран многофункциональных цифровых индикаторов (МФЦИ) [1, 2]. На экране МФЦИ отображаются значения параметров полета летательного аппарата, состояние общесамолетного оборудования и пр.

Одним из режимов работы МФЦИ является режим индикации навигационной обстановки, содержащий индикационные кадры цифровой карты местности в зоне полетов. Во время движения ЛА изображение карты местности смещается на экране МФЦИ пропорционально выбранному экипажем масштабу отображения и скорости движения объекта. При выполнении разворотов ЛА должна обеспечиваться также функция поворота на экране МФЦИ изображения цифровой карты местности на заданный угол, реализуемая программно-аппаратными средствами графических модулей авионики. Графические модули могут входить в состав аппаратных средств как МФЦИ, так и специализированных бортовых цифровых вычислителей, построенных в соответствии с принципами [3—5] интегрированной модульной авионики.

В состав графических модулей авионики входят [6—8] процессоры и компоненты программируемых логических интегральных схем (ПЛИС). В работах [9—14] показано, что для построения эффективных по критерию производительности кар-

тографических вычислительных систем необходимо использовать способ формирования изображения, основанный на перераспределении программно реализуемых функций вычислительного ядра модуля и аппаратно реализуемых функций компонентов ПЛИС, решающих задачу графоускорителя.

Наибольшее практическое применение получили [15] способы поворота изображения, основанные на использовании:

- прямой или обратной синусно-косинусной матрицы поворота;
- алгоритма Оуэна-Македона.

Целью настоящей статьи является представление широкому кругу читателей практических результатов исследования различных алгоритмов поворота растрового изображения на произвольный угол с возможностью реализации этих алгоритмов на схемотехнике ПЛИС, применяемой в модулях авионики.

1. Поворот изображения с помощью синусно-косинусной матрицы поворота

Для анализа способов практической реализации функции поворота растрового изображения и их моделирования в аппаратно-ориентированных программных пакетах целесообразно использовать графическое изображение (тестовый пример), представленное на рис. 1, см. третью сторону обложки.

Изображение на рис. 1 состоит из трех вертикальных полос разного цвета, а также отдельных вертикальных, горизонтальных и диагональных линий заданной толщины. Синусно-косинусная матрица поворота изображения на произвольный угол α имеет следующий вид:

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}. \quad (1)$$

Поворот графического изображения осуществляется путем умножения матрицы поворота (1) на вектор-столбец, представляющий собой координаты точки вращения:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Таким образом, координатные уравнения поворота с использованием матрицы (1) имеют следующий вид:

$$\begin{cases} x' = x\cos\alpha - y\sin\alpha \\ y' = -x\sin\alpha + y\cos\alpha \end{cases},$$

где (x', y') — координаты новой точки изображения, полученные в результате поворота изображения; (x, y) — координаты точки до поворота изображения, т. е. координаты исходной точки изображения.

Для поворота графического изображения относительно произвольной точки необходимо использовать координатные уравнения вида

$$\begin{cases} x' = x_0 + (x - x_0)\cos\alpha + (y_0 - y)\sin\alpha \\ y' = y_0 + (x - x_0)\sin\alpha + (y - y_0)\cos\alpha \end{cases}, \quad (2)$$

где (x_0, y_0) — координаты точки, относительно которой поворачивается изображение. Такой точкой может быть центр индицируемого графического изображения, например, точка проекции местоположения ЛА на цифровую карту местности в заданной системе координат, или точка вершины (края) индицируемого растра.

За точку, относительно которой осуществляется поворот графического изображения, в общем случае целесообразно принять точку центра экрана МФЦИ, совмещенную с точкой позиционирования силуэта самолета на цифровой карте местности. Тогда уравнение (2) для графического изображения с размерами 800×800 пикселей и с центром в точке с координатами $x_0 = 400$, $y_0 = 400$ примет вид

$$\begin{cases} x' = 400 + (x - 400)\cos\alpha + (400 - y)\sin\alpha \\ y' = 400 + (x - 400)\sin\alpha + (y - 400)\cos\alpha \end{cases},$$

где x — координаты пикселей графического изображения по оси абсцисс, $x \in \{0, 1, \ldots, 799\}$; y — координаты пикселей изображения по оси ординат, $y \in \{0, 1, \ldots, 799\}$.

Ниже представлен фрагмент кода программы в среде MATLab, реализующий функцию поворота изображения с помощью синусно-косинусной матрицы поворота.

```
for i = 1:800
    for k = 1:800
        X2X = int16(X0X + (i-X0X)*cosd(a) + (Y0Y-k)*sind(a));
        Y2Y = int16(Y0Y + (i-X0X)*sind(a) + (k-Y0Y)*cosd(a));
        if(X2X>0 && X2X<800 && Y2Y<800 && Y2Y>0)
           AHH(X2X,Y2Y,1) = QV(i,k,1);
           AHH(X2X,Y2Y,2) = QV(i,k,2);
           AHH(X2X,Y2Y,3) = QV(i,k,3);
        end
    end
end
```

В коде программы использованы следующие обозначения: `a` — угол поворота графического изображения; (`i, k`) — координаты точки до поворота графического изображения; (`X2X, Y2Y`) — координаты точки после поворота графического изображения; (`X0X, Y0Y`) — координаты точки, относительно которой проводится поворот графического изображения. Координатами в коде программы в среде MATLab считаются номер строки и номер столбца, на пересечении которых находится пиксель растра изображения, подлежащего повороту.

На рис. 2, см. третью сторону обложки, представлены результаты программно реализованного на инструментальной ЭВМ поворота графического изображения, приведенного на рис. 1, при различных фиксированных углах поворота.

На рис. 3 приведено графическое представление схемотехнической реализации на ПЛИС функции поворота изображения с использованием синусно-косинусной матрицы поворота в программном пакете Altera Quartus II. Рис. 3 условно разделен на три области. В первой (верхней) области представлены входные воздействия: координата x — (x[31...0]); координата y — (y[31...0]); значение косинуса угла поворота — (cos$a$[31...0]); значение синуса угла поворота — (sin$a$[31...0]); координаты точки, относительно которой проводится поворот изображения, — ($x0x$[31...0], $y0y$[31...0]). Вторая и третья области рис. 3 представляют собой вычислительные узлы расчета изменений значений координат x и y после поворота изображения на основе формулы (2).

На рис.4 представлена временна́я диаграмма для фрагмента схемы, изображенной на рис. 3. Заданы следующие обозначения входных воздействий: *clk* — тактовая частота, равная 100 МГц; x — координата x, дискретно изменяющаяся от 0 до 5; y — координата y, принятая равной 0; cos$a$ — косинус угла поворота, равный 0x40000000; sin$a$ — синус угла поворота, равный 0x0. Показано, что после выполнения всех расчетов координаты повернутого изображения действительны на шине данных через 42,6 нс модельного времени.

При использовании ПЛИС семейства Stratix II такая схема аппаратной реализации функции поворота занимает 8 % от всех топологических элементов: 3,519 ALUTs (Adaptive Look Up Table — адаптивные логические блоки табличного типа) и 1,98 выделенных логических регистров.

При использовании матрицы поворота (1) для реализации функции поворота графического изображения при углах поворота, отличных от 0, $\pi/2$, π и $3\pi/2$, в изображении появляются дефекты — фрагменты изображения, отсутствующие в исходном изображении до его поворота. Появления таких дефектов синтезированного (повернутого) изображения можно избежать при использовании математического алгоритма Оуэна-Македона.

2. Поворот растрового изображения с помощью алгоритма Оуэна-Македона

Алгоритм Оуэна-Македона [15] основан на использовании способа разложения синусно-косинусной матрицы поворота (1) на следующие составляющие:

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} = \\ = \begin{bmatrix} 1 & -\operatorname{tg}\frac{\alpha}{2} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ \sin\alpha & 1 \end{bmatrix} \begin{bmatrix} 1 & -\operatorname{tg}\frac{\alpha}{2} \\ 0 & 1 \end{bmatrix}. \quad (3)$$

Таким образом, в алгоритме Оуэна-Македона поворот графического изображения проводится в три последовательных этапа. Первый этап представляет собой сдвиг строк, второй этап — сдвиг столбцов и третий этап — сдвиг строк. Координатные уравнения поворота для каждого этапа соответственно имеют следующий вид:

$$\begin{cases} x' = x - \operatorname{tg}\frac{\alpha}{2} y, \\ y' = y \end{cases} \quad (4)$$

$$\begin{cases} x'' = x' \\ y'' = x' \sin\alpha + y' \end{cases}, \quad (5)$$

$$\begin{cases} x''' = x'' - \operatorname{tg}\frac{\alpha}{2} y'' \\ y''' = y'' \end{cases}. \quad (6)$$

При повороте графического изображения относительно произвольной точки (x_0, y_0) координатные уравнения (4)—(6) для каждого из трех последовательных этапов поворота растрового изображения примут следующий вид, соответственно:

$$\begin{cases} x' = x - \operatorname{tg}\frac{\alpha}{2}(y - y_0), \\ y' = y \end{cases} \quad (7)$$

$$\begin{cases} x'' = x' \\ y'' = (x' - x_0)\sin\alpha + y' \end{cases}, \quad (8)$$

$$\begin{cases} x''' = x'' - \operatorname{tg}\frac{\alpha}{2}(y'' - y_0) \\ y''' = y'' \end{cases}. \quad (9)$$

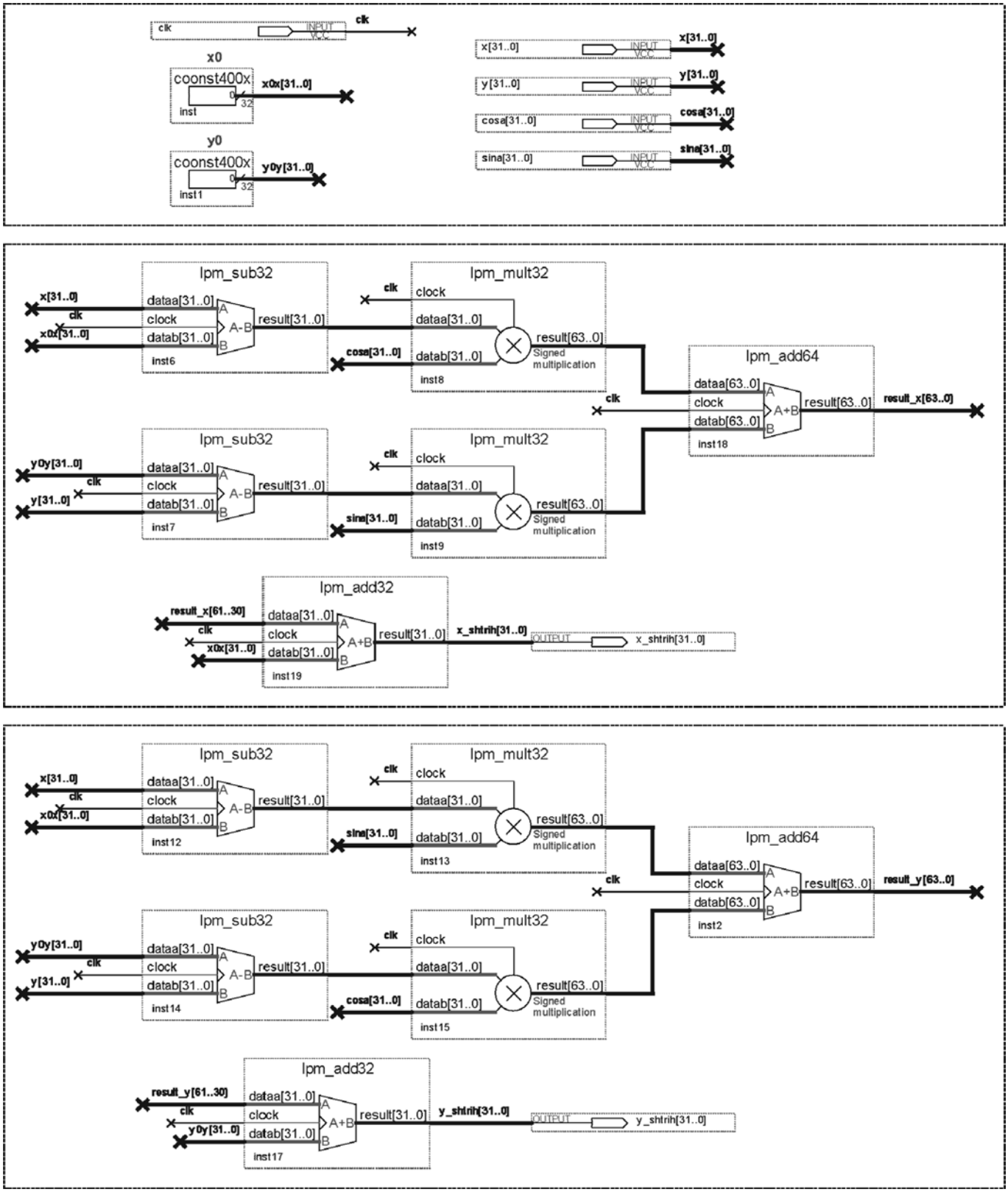

Рис. 3. Графическое представление узла электрической схемы, обеспечивающего поворот изображения в программном пакете синтезирования логики Altera Quartus II

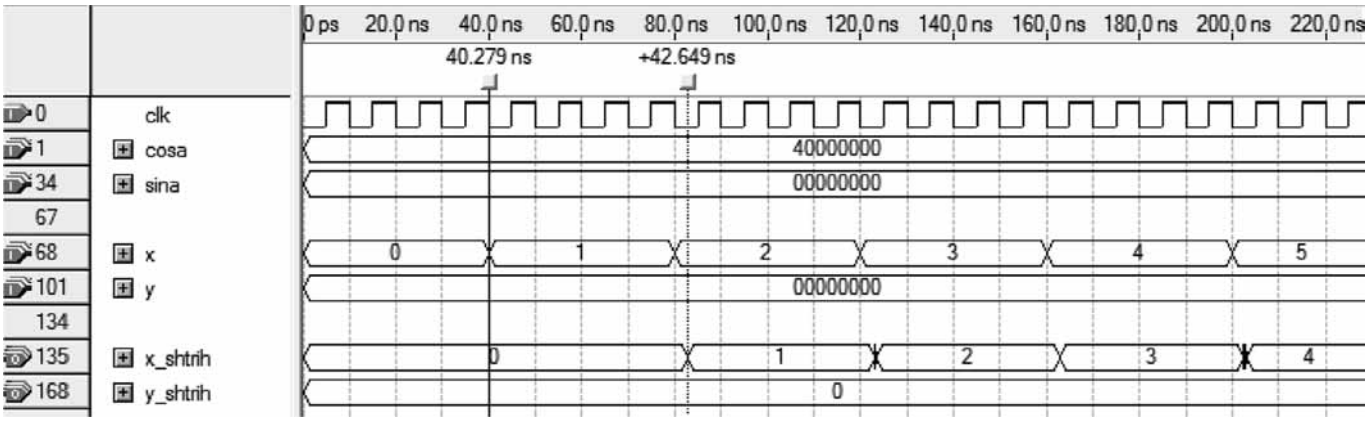

Рис. 4. Временная диаграмма работы узла электрической схемы, обеспечивающего поворот изображения на основе синусно-косинусной матрицы в программном пакете Altera Quartus II

Ниже приведен фрагмент кода программы в среде MATLab, реализующий поворот изображения с помощью алгоритма Оуэна-Македона

```
for i = 1:800
    for k = 1:800
        X1SX = int16(i-(tand(a/2))*(k-Y0Y));
        Y1SY = int16(k);
        X2SX = int16(X1SX);
        Y2SY = int16((X1SX-X0X)*sind(a) + Y1SY);
        X3SX = int16(X2SX-(tand(a/2))*(Y2SY-Y0Y));
        Y3SY = int16(Y2SY);
        if(X3SX>0 && X3SX<800 && Y3SY<800 && Y3SY>0)
           OH(X3SX,Y3SY,1) = QV(i,k,1);
           OH(X3SX,Y3SY,2) = QV(i,k,2);
           OH(X3SX,Y3SY,3) = QV(i,k,3);
        end
    end
end
```

В коде программы использованы следующие обозначения: `a` — угол поворота графического изображения; (`i`, `k`) — координаты точки до поворота графического изображения; (`X1SX`, `Y1SY`) — координаты точки после первого этапа применения алгоритма; (`X2SX`, `Y2SY`) — координаты точки после второго этапа применения алгоритма; (`X3SX`, `Y3SY`) — координаты точки после третьего этапа применения алгоритма; (`X0X`, `Y0Y`) — координаты точки, относительно которой проводится поворот графического изображения.

На рис. 5 приведено графическое представление схемотехнической реализации на ПЛИС функции поворота изображения с использованием алгоритма Оуэна-Македона в программном пакете Altera Quartus II. Поле рис. 5 условно разделено на три области. В первой (верхней) области представлены входные воздействия: координата x — (x[31...0]); координата y — (y[31...0]); значение синуса угла поворота — (sin$a$[31...0]); значение тангенса половины угла поворота — (tg$a2x$[31...0]); координаты точки, относительно которой проводится поворот изображения — ($x0x$[31...0], $y0y$[31...0]). Вторая и третья области рис. 5 представляют собой вычислительные узлы расчета изменения значений координат x и y после поворота изображения согласно формулам (7)—(9).

На рис.6 представлена временна́я диаграмма работы фрагмента электрической схемы, приведенной на рис. 5. Заданы следующие обозначения входных воздействий: *clk* — тактовая частота, равная 100 МГц; x — координата x, дискретно изменяющаяся от 2 до 7; y — координата y, дискретно изменяющаяся от 0 до 1; sin$a$ — синус угла поворота, равный 0x0; tg$a2x$ — тангенс угла поворота, равный 0x0.

Показано, что после выполнения всех расчетов координаты изображения после поворота оказываются действительными на шине данных через 25,2 нс модельного времени. При использовании ПЛИС семейства Stratix II такая схема аппаратной реализации алгоритма поворота занимает 10 % от всех топологических элементов кристалла: 4,264 ALUTs и 2,228 выделенных логических регистров.

3. Поворот растрового изображения с помощью обратной синусно-косинусной матрицы

Для поворота графического изображения можно также использовать обратную синусно-косинусную матрицу поворота следующего вида:

$$R^{-1}(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}. \qquad (10)$$

В этом случае координатные уравнения поворота графического изображения относительно произвольной точки с использованием обратной синусно-косинусной матрицы примут вид

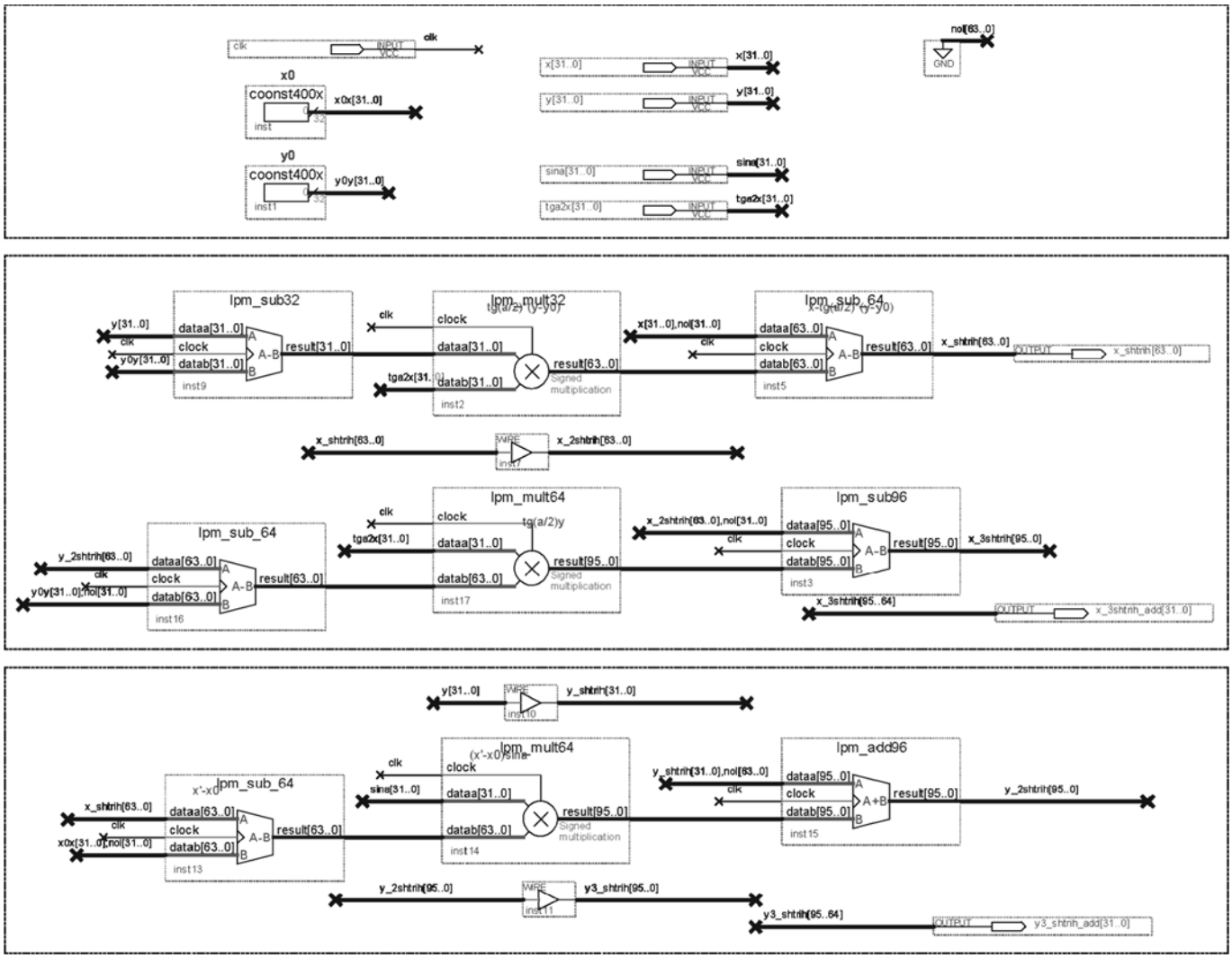

Рис. 5. Графическое представление узла электрической схемы, обеспечивающего поворот изображения по алгоритму Оуэна-Македона в программном пакете Altera Quartus II

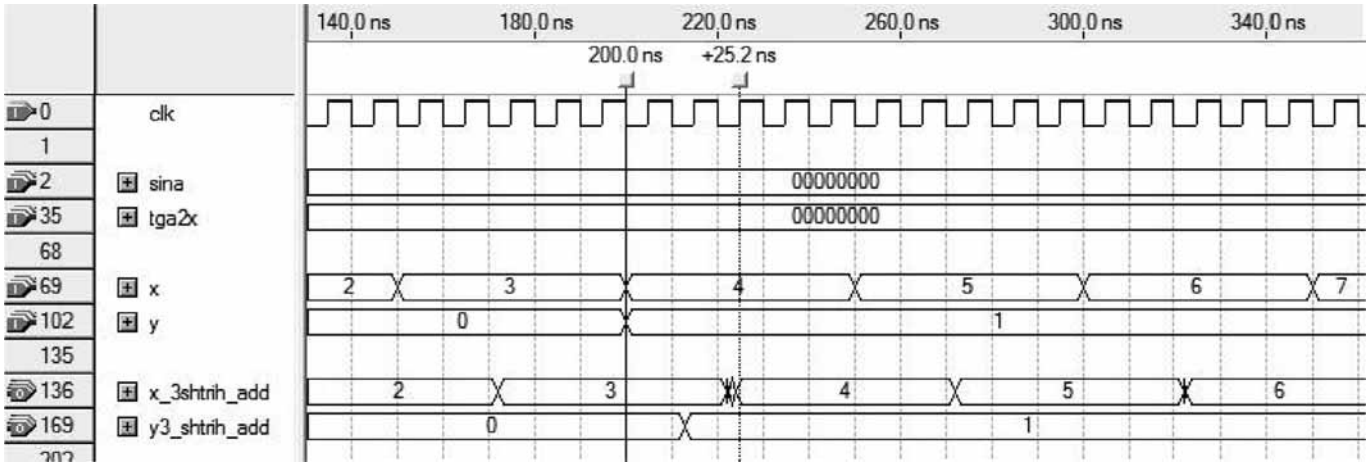

Рис. 6. Временна́я диаграмма работы узла электрической схемы, обеспечивающего поворот изображения на основе алгоритма Оуэна-Македона в программном пакете Altera Quartus II

$$\begin{cases} x = x_0 + (x' - x_0)\cos\alpha - (y_0 - y')\sin\alpha \\ y = y_0 - (x' - x_0)\sin\alpha + (y' - y_0)\cos\alpha \end{cases},$$

где x — координаты пикселей изображения по оси абсцисс, $x \in \{0, 1, ..., 799\}$; y — координаты пикселей изображения по оси ординат, $y \in \{0, 1, ..., 799\}$.

Код программы в среде MATLab, реализующий поворот растрового изображения с помощью обратной синусно-косинусной матрицы, имеет следующий вид:

```
for i = 1:800
    for k = 1:800
        X2X = int16(X0X + (i-X0X)*cosd(a) + (Y0Y-k)*sind(a));
        Y2Y = int16(Y0Y + (i-X0X)*sind(a) + (k-Y0Y)*cosd(a));
        if(X2X>0 && X2X<800 && Y2Y<800 && Y2Y>0)
           AH(i,k,1) = QV(X2X,Y2Y,1);
           AH(i,k,2) = QV(X2X,Y2Y,2);
           AH(i,k,3) = QV(X2X,Y2Y,3);
        else
           AH(i,k,1) = 1;
           AH(i,k,2) = 1;
           AH(i,k,3) = 1;
        end
    end
end
```

В коде программы в среде MATLab использованы следующие обозначения: `a` — угол поворота графического изображения; `(i, k)` — координаты точки до поворота графического изображения; `(X2X, Y2Y)` — координата точки после поворота графического изображения; `(X0X, Y0Y)` — координаты точки, относительно которой проводится поворот графического изображения.

Практическая реализация способа поворота на основе применения обратной синусно-косинусной матрицы поворота показывает, что аппаратная реализация способа вычисления обратной матрицы поворота аналогична электрическим схемам, представленным на рис. 3, с эквивалентными временны́ми характеристиками работы вычислительного узла, представленными на рис. 4.

Заключение

В результате исследования были программно в среде MATLab и аппаратно в пакете Altera Quartus II и на стендовом макете реализованы три различных способа поворота растрового изображения: с помощью матрицы поворота (1), с помощью алгоритма Оуэна-Македона (3) и с помощью расчета обратной матрицы поворота (10).

Основными результатами исследования являются следующие:

— поворот изображения с помощью матрицы поворота (1) осуществляется с дополнительно вносимыми искажениями изображения, визуально проявляющимися в виде "потерянных" пикселей, что недопустимо при отображении цифровых карт местности в авионике;

— поворот изображения с помощью алгоритма Оуэна-Македона требует применения значительных вычислительных ресурсов для своей практической реализации в формате схемотехники ПЛИС;

— поворот изображения с помощью расчета обратной матрицы поворота совмещает в себе достоинства первых двух способов.

Появление искажений повернутого изображения по отношению к исходному приводит к ухудшению визуального восприятия индикационного кадра за счет появления в нем элементов "ступенчатости" графических примитивов (линия, знак и пр.). Наиболее чувствительными для элементов изображения оказываются повороты линий, заданных не только в растровом, но и в векторном формате, на малые углы, например, $\pi/12...\pi/10$. Для устранения таких искажений синтезируемого изображения в авионике широко применяют специализированные алгоритмы сглаживания, подробно рассмотренные, в частности, в работе [16].

Полученные результаты могут быть использованы для создания программного и аппаратного обеспечения графических модулей авионики, входящих в состав бортовых графических станций и индикаторов МФЦИ, реализующих обработку видеоизображений, в частности, поворот растрового изображения.

Результаты моделирования получены с использованием инструментальной ЭВМ со следующими характеристиками: ASUS K56CB-X0391H, процессор Intel(R) Core(TM) i5-3337U, 4 ядра, тактовая частота 1,8 ГГц, оперативная память 6 Гбайт под управлением операционной системы Windows 8.1.

Список литературы

1. **Жаринов И. О., Жаринов О. О.** Бортовые средства отображения информации на плоских жидкокристаллических панелях: учеб. пособие. Информационно-управляющие системы, СПб.: ГУАП, 2005. 144 с.

2. **Жаринов И. О., Емец Р. Б.** Индикационное оборудование в авиации XXI века // Научно-технический вестник информационных технологий, механики и оптики. 2003. № 5 (11). С. 193—196.

3. **Парамонов П. П., Жаринов И. О.** Интегрированные бортовые вычислительные системы: обзор современного состояния и анализ перспектив развития в авиационном приборостроении // Научно-технический вестник информационных технологий, механики и оптики. 2013. № 2. С. 1—17.

4. **Гатчин Ю. А., Жаринов И. О.** Основы проектирования вычислительных систем интегрированной модульной авионики: монография. М.: Машиностроение, 2010. 224 с.

5. **Копорский Н. С., Видин Б. В., Жаринов И. О.** Организация вычислительного процесса в многомашинном бортовом вычислительном комплексе // Известия высших учебных заведений. Приборостроение. 2006. Т. 49, № 6. С. 41—50.

6. **Шукалов А. В., Парамонов П. П., Книга Е. В., Жаринов И. О.** Принципы построения вычислительных ком-

понентов систем интегрированной модульной авионики // Информационно-управляющие системы. 2014. № 5. С. 64—71.

7. **Книга Е. В., Жаринов И. О., Богданов А. В., Виноградов П. С.** Принципы организации архитектуры перспективных бортовых цифровых вычислительных систем в авионике // Научно-технический вестник информационных технологий, механики и оптики. 2013. № 2. С. 163—165.

8. **Книга Е. В., Жаринов И. О.** Принципы построения комбинированной топологии сети для перспективных бортовых вычислительных систем // Научно-технический вестник информационных технологий, механики и оптики, 2013. № 6. С. 92—98.

9. **Парамонов П. П., Костишин М. О., Жаринов И. О., Нечаев В. А., Сударчиков С. А.** Принцип формирования и отображения массива геоинформационных данных на экране средств бортовой индикации // Научно-технический вестник информационных технологий, механики и оптики, 2013. № 6. С. 136—142.

10. **Парамонов П. П., Ильченко Ю. А., Жаринов И. О., Тарасов П. Ю.** Структурный анализ и синтез графических изображений на экранах современных средств бортовой индикации на плоских жидкокристаллических панелях // Авиакосмическое приборостроение. 2004. № 5. С. 50—57.

11. **Парамонов П. П., Ильченко Ю. А., Жаринов И. О.** Теория и практика статистического анализа картографических изображений в системах навигации пилотируемых летательных аппаратов // Датчики и системы. 2001. № 8. С. 15—19.

12. **Костишин М. О., Жаринов И. О., Жаринов О. О., Богданов А. В.** Оценка частоты обновления информации в видеопотоке индикации бортовых геоинформационных данных авионики // Вестник Череповецкого государственного университета. 2014. № 4. С. 9—15.

13. **Жаринов И. О., Кирсанова Ю. А., Коновалов П. В., Костишин М. О.** Алгоритм формирования и вывода картографических изображений в системах навигации пилотируемых летательных аппаратов // Мехатроника, автоматизация, управление. 2014. № 8. С. 68—72.

14. **Костишин М. О., Шукалов А. В., Парамонов П. П., Жаринов И. О., Жаринов О. О.** Алгоритмы автоматизации конфигурирования загрузочных компонентов аэронавигационной информации и геоинформационных данных авионики // Мехатроника, автоматизация, управление. 2014. № 9. С. 64—72.

15. **Owen Ch.B., Makedon F.** High quality alias free image rotation // Proceeding of 30th Asilomar Conference on Signals, Systems, and Computers Pacific Grove, California, November 2—6, 1996. URL: http://www.cs.dartmouth.edu/reports/TR96-301.pdf

16. **Dan S., Jie B., Zhibo Sh**. Anti-aliasing in aircraft cockpit display system passed on modified Bresanhem algorithm and virtual technology // Journal of multimedia. 2014. Vol. 9, N. 6. P. 765—773.

Software and Brainware for Solving Practical Tasks of Graphics Rotation in Aircraft Instrumentation

E. V. Kniga, ekovinskaya@gmail.com, Design Bureau "Electroavtomatika", Saint Petersburg, 198095, Russian Federation, **A. V. Shukalov**, aviation78@mail.ru, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University), Saint Petersburg, 197101, Russian Federation, Design Bureau "Electroavtomatika", Saint Petersburg, 198095, Russian Federation, **A. V. Gurjanov**, postmaster@elavt.spb.ru, Design Bureau "Electroavtomatika", Saint Petersburg, 198095, Russian Federation, **I. O. Zharinov**, igor_rabota@pisem.net, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University), Saint Petersburg, 197101, Russian Federation, Design Bureau "Electroavtomatika", Saint Petersburg, 198095, Russian Federation, **O. O. Zharinov**, zharinov73@hotbox.ru, Saint-Petersburg State University of Aerospace Instrumentation, Saint Petersburg, 190000, Russian Federation

Corresponding author:

Zharinov Igor O., Chef of Department, Saint Petersburg National Research University of Information Technologies, Mechanics and Optics (ITMO University), 197101, Saint Petersburg, Russian Federation
E-mail: igor_rabota@pisem.net

Received on November 01, 2016
Accepted on November 07, 2016

The practical task of graphical images rotation, which is actual for map information onboard system design in particular and for aviation instrumentation in general, is considered in the paper. Different mathematical bitmap rotation algorithms, which are aimed at next schematic realization based on Field-Programmable Gate Array in graphical modules, are studied in the paper. Mathematical modeling results of graphic image rotation algorithms and program code fragments for MATLab are presented in the paper. Test graphical images and graphical images rotation algorithms schematic fragments for the "Altera Quartus II" software are presented in the paper. Temporal and topological characteristics for different algorithms on onboard aviation hardware prototype are received in the study.

Keywords: *graphic modules, avionics, image rotation, algorithms, programs, Owen-Makedon*

For citation:

Kniga E. V., Shukalov A. V., Gurjanov A. V., Zharinov I. O., Zharinov O. O. Software and Brainware for Solving Practical Tasks of Graphics Rotation in Aircraft Instrumentation, *Programmnaya Inzheneria*, 2017, vol. 8, no. 1 pp. 38-46.

DOI: 10.17587/prin.8.38-46

References

1. **Zharinov I. O., Zharinov O. O.** *Bortovye sredstva otobrazhenija informacii na ploskih zhidkokristallicheskih paneljah* (Onboard Display on Flat Liquid Crystal Panels), Saint Petersburg, Informacionno-upravljajushhie sistemy, 2005, 144 p. (in Russian).

2. **Zharinov I. O., Emets R. B.** Indikacionnoe oborudovanie v aviacii XXI veka (Evidence Equipment, Aviation XXI Century), *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki,* 2003, no. 11, pp. 193—195 (in Russian).

3. **Paramonov P. P., Zharinov I. O.** Integrirovannye bortovye vychislitel'nye sistemy: obzor sovremennogo sostojanija i analiz perspektiv razvitija v aviacionnom priborostroenii (Integrated On-board Computing Systems: Present Situation Review and Development Prospects Analysis in the Aviation Instrument-making Industry), *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2013, no. 2, pp. 1—17 (in Russian).

4. **Gatchin Iu. A., Zharinov I. O.** *Osnovy proektirovaniia vychislitel'nykh sistem integrirovannoi modul'noi avioniki* (Basics of Designing Computer Systems Integrated Modular Avionics), Moscow, Mashinostroenie Publ., 2010. 224 p. (in Russian).

5. **Koporskiy N. S., Vidin B. V., Zharinov I. O.** Organizacija vychislitel'nogo processa v mnogomashinnom bortovom vychislitel'nom komplekse (Organization of Computing Processing Multicomputer Onboard Computer Complex), *Izvestiia vuzov. Priborostroenie*, 2006, vol. 49, no. 6. pp. 41—50 (in Russian).

6. **Shukalov A. V.**, **Paramonov P. P.**, **Kniga E. V.**, **Zharinov I. O.** Principy postroenija vychislitel'nyh komponentov sistem integrirovannoj modul'noj avioniki (The Principles of Design of Computing Systems Components Integrated Modular Avionics), *Informacionno-upravljajushhie sistemy*, 2014, no. 5, pp. 64—71 (in Russian).

7. **Kniga E. V., Zharinov I. O., Bogdanov A. V., Vinogradov P. S.** Principy organizacii arhitektury perspektivnyh bortovyh cifrovyh vychislitel'nyh sistem v avionike (Rules of Architecture Design for Advanced Onboard Digital Computer Systems in Avionics), *Nauchno-tekhnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2013, no. 2, pp. 163—165 (in Russian).

8. **Kniga E. V., Zharinov I. O.** Principy postroenija kombinirovannoj topologii seti dlja perspektivnyh bortovyh vychislitel'nyh sistem (Design Principles of a Combined Network Topology for Advanced On-board Computing System), *Nauchno-tehnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2013, no. 6, pp. 92—98 (in Russian).

9. **Paramonov P. P., Kostishin M. O., Zharinov I. O., Nechaev V. A., Sudarchikov S. A.** Princip formirovanija i otobrazhenija massiva geoinformacionnyh dannyh na jekrane sredstv bortovoj indikacii (Formation and display principles for an array of geoinformation data by means of onboard display screen), *Nauchno-tehnicheskii vestnik informatsionnykh tekhnologii, mekhaniki i optiki*, 2013, no. 6, pp. 136—142 (in Russian).

10. **Paramonov P. P., Il'chenko Yu. A., Zharinov I. O., Tarasov P. Yu.** Strukturnyj analiz i sintez graficheskih izobrazhenij na jekranah sovremennyh sredstv bortovoj indikacii na ploskih zhidkokristallicheskih paneljah (Structural analysis and synthesis of graphic images on the screens of modern on-board display on flat liquid crystal panels), *Aviakosmicheskoe priborostroenie*, 2004, no. 5, pp. 50—57 (in Russian).

11. **Paramonov P. P., Il'chenko Iu. A., Zharinov I. O.** Teorija i praktika statisticheskogo analiza kartograficheskih izobrazhenij v sistemah navigacii pilotiruemyh letatel'nyh apparatov (Theory and Practice of Statistical Analysis of Cartographic Images in Navigation Systems Manned Aircraft), *Datchiki i sistemy*, 2001, no. 8, pp. 15—19 (in Russian).

12. **Kostishin M. O., Zharinov I. O., Zharinov O. O., Bogdanov A. V.** Ocenka chastoty obnovlenija informacii v videopotoke indikacii bortovyh geoinformacionnyh dannyh avioniki (Estimation of the refresh frequency in the video display board geoinformation data avionics), *Vestnik Cherepovezhkogo gosudarstvennogo universiteta*, 2014, no. 4, pp. 9—15 (in Russian).

13. **Zharinov I. O., Kirsanova Ju. A., Konovalov P. V., Kostishin M. O.** The Algorithm for Generating and Outputting the Image Map Navigation Systems Manned Aircraft, *Mehatronika, avtomatizacija, upravlenie*, 2014, no. 8, pp. 68—72 (in Russian).

14. **Kostishin M. O., Shukalov A. V., Paramonov P. P., Zharinov I. O., Zharinov O. O.** Algoritm formirovanija i vyvoda kartograficheskih izobrazhenij v sistemah navigacii pilotiruemyh letatel'nyh apparatov (Algorithms Automation Configuration Bootable Components Aeronautical Information and Geoinformation Data Avionics), *Mehatronika, avtomatizacija, upravlenie*, 2014, no. 9, pp. 64—72 (in Russian).

15. **Owen Ch. B.**, **Makedon F.** High quality alias free image rotation. *Proceeding of 30th Asilomar Conference on Signals, Systems, and Computers Pacific Grove*, California, November 2—6, 1996, available at: http://www.cs.dartmouth.edu/reports/TR96-301.pdf

16. **Dan S., Jie B., Zhibo Sh.** Anti-aliasing in aircraft cockpit display system passed on modified Bresanhem algorithm and virtual technology, *Journal of multimedia*, 2014, vol. 9, no. 6, pp. 765—773.

Итоги двенадцатой конференции "Разработка ПО/CEE-SECR 2016"

В конце октября в Москве в центре Digital October состоялась конференция "Разработка ПО/CEE-SECR 2016". На площадке собрались 500 участников: представители ИТ-компаний, профессора, члены научного сообщества, предприниматели, студенты и инвесторы — все те, кто желает получить широкий обзор отрасли и заинтересован в отслеживании последних трендов в разработке.

Доклады шли одновременно в пять потоков, поэтому у участников конференции была возможность посещать наиболее интересные для них выступления и проводить все время на конференции с наибольшей пользой. В одном из залов проходили мастер-классы, на которых гости CEE-SECR 2016 могли на практике научиться новым техникам в области разработки ПО.

На конференции были затронуты все аспекты разработки ПО: от руководства командой и обучения ИТ-специалистов до технических узкоспециализированных тем. Многие участники в отзывах отмечают, что разноплановость тем, обсуждаемых на конференции, является одним из главных ее преимуществ.

На CEE-SECR 2016 выступили приглашенные ключевые спикеры и докладчики, выступления которых прошли конкурсный отбор. Большинство докладов перетекали в продолжительную дискуссию в специальных уголках общения с выступающими. Среди ключевых спикеров присутствовали такие эксперты, как член комитета по стандартизации C++ Майкл Вонг (Канада), вице-президент по технологической стратегии компании MapR Кристал Валентайн (США), автор учебников по объектному мышлению Дэвид Уэст (США) и другие.

30 октября на площадке Digital October прошли мастер-классы от ключевых спикеров CEE-SECR 2016, охватывающие темы микросервисной архитектуры, гетерогенного программирования в C++, создания системы потоковой обработки данных, цифровой трансформации.

Церемония закрытия конференции состоялась вечером 29 октября в Главном зале. Традиционно Программный Комитет вручил ежегодную премию имени Бертрана Мейера лучшим исследовательским работам в области программной инженерии: Михаилу Малеванному (Донской государственный техниче-

ский университет) — работа "Устойчивая привязка к синтаксическим конструкциям в изменяющемся коде" и Дмитрию Щитинину (Yandex.Classifieds) — работа "Потоковая фильтрация событий".

Закрыл конференцию торжественной речью представитель руководства АО "РВК" Евгений Кузнецов, отметив, как важно развивать отрасль разработки программного обеспечения в нашей стране, в том числе посредством таких мероприятий, как конференция CEE-SECR.

Конференция состоялась благодаря поддержке компаний-партнеров, многие из которых сотрудничают с CEE-SECR уже не первый год, делая весомый вклад в развитие ИТ-сферы в России.

Приглашаем принять участие в конференции "Разработка ПО/CEE-SECR 2017". Следите за подробностями на сайте конференции http://secr.ru/

ООО "Издательство "Новые технологии". 107076, Москва, Стромынский пер., 4
Технический редактор *Е. М. Патрушева*. Корректор *Е. В. Комиссарова*

Сдано в набор 07.11.2016 г. Подписано в печать 20.12.2016 г. Формат 60×88 1/8. Заказ РІ117
Цена свободная.

Оригинал-макет ООО "Адвансед солюшнз". Отпечатано в ООО "Адвансед солюшнз".
119071, г. Москва, Ленинский пр-т, д. 19, стр. 1. Сайт: **www.aov.ru**