УДК 004.054 DOI: 10.31114/2078-7707-2021-4-86-91

Быстрый метод генерации псевдослучайных векторов большой размерности для тестирования систем на кристалле

М.И. Дябин[1], А.В. Решетников[2], Е.А. Саксонов[3]

[1]ООО «Каскад», г. Москва, dyabin@mail.ru

[2]ООО «Аккорд», г. Москва, a_reshetnikov@hush.com

[3]Московский технический университет связи и информатики, г. Москва, saksmiem@mail.ru

***Аннотация* — Предложен новый алгоритм генерации псевдослучайных векторов в случае, когда к ним не предъявляется требование равномерной распределённости. Особенность предлагаемого генератора в том, что он не использует какие-либо промежуточные генераторы псевдослучайных чисел. Основными достоинствами нового алгоритма являются высокая скорость его работы и простота реализации. Отмечена связь проблемы генерации псевдослучайных векторов с проблемой функционального тестирования микропроцессора.**

***Ключевые слова* — генератор псевдослучайных векторов, функциональное тестирование микропроцессора.**

I. Введение

При производстве какой-либо электронной микросхемы одним из обязательных этапов её разработки является проведение её функционального тестирования, включающего в себя как тестирование отдельных узлов разрабатываемой микросхемы, так и проведение комплексного тестирования (то есть тестирования системы как единого целого: важно проверить не только правильность работы отдельных функциональных блоков микросхемы, но и их взаимодействие между собой при решении какой-либо общей для них задачи). Функциональное тестирование микропроцессора не является исключением: этот процесс включает в себя несколько этапов и требует составления разного рода тестовых программ, каждая из которых обладает своими особенностями и проверяет ту или иную из заявленных возможностей разрабатываемого процессора.

В зависимости от того, каким образом спроектировано процессорное ядро, для его тестирования могут потребоваться и тесты отдельных инструкций микропроцессора, и отдельные от них тесты механизма адресации памяти, и специализированные тесты для проверки различных уровней кэша – если устройство такого типа входит в состав проектируемой системы; могут понадобиться как тесты, исполняемые на самой микросхеме, так и тесты, запускаемые в какой-либо компьютерной среде, моделирующей логику работы процессора. Наряду с программами, использующими для тестирования фиксированные наборы входных данных, могут быть составлены программы, которые в процессе своей работы генерируют входные данные псевдослучайным образом, а также тестовые программы, запускающие на исполнение псевдослучайным образом сгенерированный машинный код. Систематическое описание различных типов тестовых программ приведено в [1].

Тема функционального тестирования микропроцессоров в настоящее время освещена во многих научных работах. Так, проблеме тестирования когерентности кэширующих подсистем посвящена статья [2]. О тестировании функциональных блоков микросхем с помощью интерфейса JTAG рассказано, например, в работе [3]. Авторами настоящей статьи описана методика [4], использованная ими в процессе тестирования процессорного ядра системы на кристалле «Каскад-1»: это высокопроизводительная система на кристалле, предназначенная в первую очередь для передачи данных через входящий в её состав OFDM-модем (см. рис. 1; более подробное описание СнК «Каскад-1» приведено в статье [5]).

В данной работе мы рассмотрим вопрос о том, *каким образом может быть произведено заполнение некоторой области оперативной памяти значениями, сгенерированными программно псевдослучайным образом*. Такая задача зачастую возникает при разработке различных функциональных тестов микропроцессоров. Например, при разработке генератора псевдослучайного машинного кода обычно приходится иметь дело с блоками данных, последовательность которых должна обладать свойством псевдослучайности[1].

[1] Имеются в виду блоки данных, содержащие программный код, представленный в той или иной форме. Эти блоки могут хранить не обязательно готовый к исполнению машинный код, поскольку *произвольная последовательность байтов не всегда представляет собой корректную программу для микропроцессора*. В этом состоит основная трудность при написании таких генераторов: что если будет произведён запуск не вполне корректного машинного кода, то результат его исполнения может быть различным на разных процессорах одной и той же архитектуры.

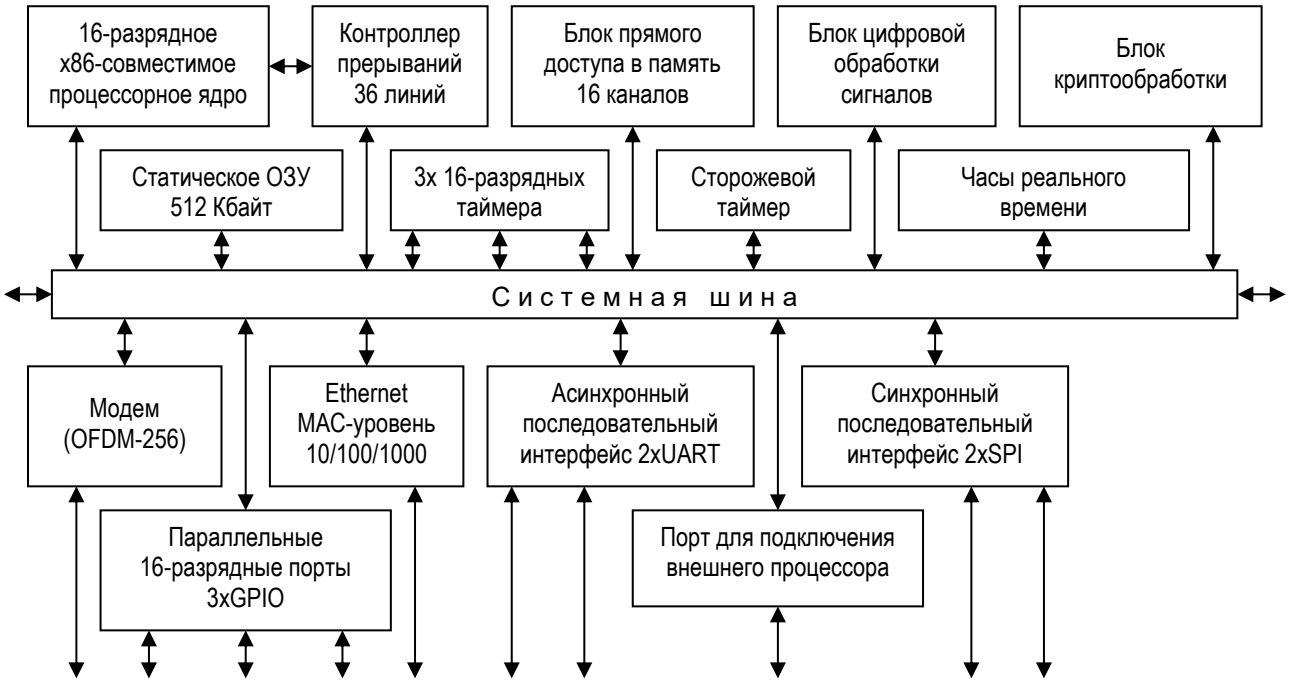

Рис. 1. Структурная схема СнК «Каскад-1»

Другой пример: для некоторых микропроцессоров существуют команды, результат выполнения которых зависит от содержимого того или иного массива. К такому типу относятся, например, команды работы со строками, предусмотренные архитектурой x86 (имеются в виду сложные команды, содержащие префиксы повторения), а также, например, команда «Leave» (тоже из набора команд x86): её входным аргументом является сложная структура данных, хранящаяся в вершине стека. При составлении тестов для таких команд важно, чтобы среди наборов их входных данных были как наборы, охватывающие различные «крайние случаи», так и большое число наборов, сформированных псевдослучайным образом.

Статья содержит следующие разделы. Суть проблемы, её математические аспекты и основной способ её решения рассмотрены в разделе 2. В ряде случаев – например, при разработке программного обеспечения для тестирования микропроцессоров – с той же целью может быть использован другой метод, разработанный авторами настоящей работы. Данный метод представлен в разделе 3. Его эффективность сравнивалась с эффективностью общего метода; результаты такого сравнения приведены в том же разделе. Далее следует заключительный раздел.

II. Об общей проблеме генерации псевдослучайных векторов

Проблема заполнения буфера памяти значениями, сгенерированными псевдослучайным образом, сводится к проблеме генерации псевдослучайных векторов. К настоящему времени вопрос об их генерации изучен довольно подробно. Существенный вклад в разработку данной темы внёс Г. Нидеррайтер (например, в главе 10 его монографии [6] рассмотрены матричные и нелинейные методы генерации). Наоборот: построение псевдослучайных векторов можно выполнять *покомпонентно*, заполняя некоторую область памяти значениями, полученными с помощью какого-либо генератора псевдослучайных чисел.

Однако, если в генераторе псевдослучайных векторов их покомпонентное построение реализовано небрежно, то, как известно, последовательности векторов, полученных с помощью такого генератора, могут обладать нежелательными закономерностями. Это некритично, по мнению авторов, при разработке функциональных тестов микропроцессоров. Тем не менее, в целях полноты изложения материала мы напомним в данном разделе о том, что могут представлять из себя такие закономерности, и покажем, каким образом можно избежать их возникновения при использовании покомпонентного метода генерации.

Рассмотрим целочисленную последовательность x_1, x_2, …, заданную следующим рекуррентным выражением при некоторых значениях параметров a, c и m:

$$x_{i+1} = \left(a x_i + c\right) \mod m; \qquad (1)$$

положим

$$1 \le x_1 \le m-1. \qquad (2)$$

Такую последовательность можно использовать для генерации псевдослучайных чисел, если, полагая всякий раз, что последним из псевдослучайных чисел, полученных данным способом, является x_i , новое значение x_{i+1} вычислять с помощью формулы (1); перед началом работы такого генератора требуется задать параметру x_1 какое-либо значение, удовлетворяющее условию (2)

– оно инициализирует рассматриваемую последовательность. Генераторы, использующие в своей работе указанный метод, называются *линейными конгруэнтными генераторами*.

Псевдослучайный генератор целых чисел является «хорошим», если он производит последовательность, похожую на реализацию значений некоторой равномерно распределённой случайной величины. В зависимости от значений a, c и m, одни линейные конгруэнтные генераторы могут считаться достаточно хорошими; в то время как для других генераторов оказывается, что производимые ими последовательности содержат закономерности, появление которых маловероятно, если бы эти последовательности были сформированы из реализаций равномерно распределённых случайных величин.

Печально известен линейный конгруэнтный генератор, использующий параметры $a = 65539$, $c = 0$ и $m = 2^{31}$. Выбор таких значений параметров был обусловлен тем, что с ними имеется возможность существенно оптимизировать реализацию генератора. Но позже было замечено, что статистические свойства данного генератора оставляют желать лучшего: оказалось, что все производимые им последовательности чисел имеет определённую закономерность, несвойственную последовательностям реализаций равномерно распределённых случайных величин. А именно: если в выходной последовательности данного генератора $x_1, x_2, \ldots$ сгруппировать несколько подряд идущих её элементов в тройки, т.е. (x_1, x_2, x_3), (x_4, x_5, x_6) и т.д., а затем рассмотреть эти тройки как координаты точек в трёхмерном пространстве и нанести их на график, то окажется, что все указанные точки принадлежат определённому семейству параллельных плоскостей – что было б крайне маловероятно, если бы последовательность $x_1, x_2, \ldots$ состояла из реализаций какой-либо случайной величины, равномерно распределённой на множестве $\{1, 2, \ldots, m - 1\}$.

Аналогичным недостатком обладают многие другие линейные конгруэнтные генераторы: точно данный эффект описывает теорема 1 из статьи [7].

Следовательно, при использовании покомпонентного метода получения псевдослучайных векторов – того самого метода, который в общих чертах был описан в начале раздела, – иногда (но не всегда, как мы увидим далее) может оказаться, что генератор обладает той нежелательной особенностью, что производит последовательности векторов, не похожие на последовательности реализаций случайной векторной величины, равномерно распределённой внутри какого-либо многомерного куба. Подобные ситуации могут возникать в тех случаях, когда используемый промежуточный генератор псевдослучайных чисел не достаточно хорош – в том смысле, в каком мы охарактеризовали выше «хорошие» и «плохие» псевдослучайные генераторы.

Возникает вопрос: а в противном случае покомпонентный метод всегда срабатывает или нет? То есть, если взять действительно «хороший» генератор псевдослучайных чисел, можно ли ему доверить покомпонентную генерацию «хороших», равномерно распределённых псевдослучайных векторов? И есть ли на этот счёт какие-либо строгие ограничения на генераторы и какие-либо точные оценки качества выходных последовательностей, получаемых таким способом?

Достаточно подробный ответ на данный вопрос содержится в монографии [8] в разделе 3.3.4 «Спектральный критерий». Оказывается, *для того, чтобы покомпонентный генератор псевдослучайных векторов производил последовательность n-мерных векторов, обладающую свойством равномерной распределённости при заданном значении n, достаточно, чтобы в качестве промежуточного генератора псевдослучайных чисел был выбран какой-либо линейный конгруэнтный генератор, показывающий хороший результат при его проверке спектральным критерием для той же размерности n*. Это следует из теоремы N, приведённой в том же разделе и впервые доказанную Нидеррайтором, которая позволяет оценить *разброс* векторов, полученных с использованием промежуточного числового линейного конгруэнтного генератора.

Основной принцип, лежащий в основе покомпонентного метода генерации векторов, Д. Э. Кнут формулирует следующим образом [8, раздел 3.3.4, часть F «Связь с критерием серий»]: «Одним из основных результатов его [Нидеррайтер] теории было следующее: он показал, что генератор случайных чисел проходит проверку с помощью критерия серий для нескольких измерений, если этот генератор выдерживает проверку спектральным критерием, даже когда вместо полного периода рассматривается большая его часть». Упомянутая выше теорема N является не чем иным, как строгой формулировкой данного утверждения.

На практике это означает следующее: если при написании программы возникает необходимость однократно или многократно выполнить построение псевдослучайного вектора, то, как правило, можно использовать для этой цели обычный способ – например:

```
#include <stdlib.h>
#include <limits.h>

void get_random_tuple(unsigned char* output,
 size_t length)
{size_t i;
      for (i = 0; i < length; i++) output[i] =
       rand( ) >> CHAR_BIT * (sizeof(int)-1);}
```

Коль скоро наиболее часто используемые реализации стандартной библиотеки языка C включают в себя «хорошие» генераторы псевдослучайных чисел, то псев-

дослучайные векторы, которые производит представленный код, также являются достаточно «хорошими», по крайней мере для большинства практических применений (если, конечно, при сборке данного кода в качестве стандартной библиотеки C была использована подходящая её реализация).

Обращаем внимание, что *при покомпонентной генерации псевдослучайных векторов следует использовать именно старшие, а не младшие разряды чисел, производимых промежуточным псевдослучайным генератором*: это даёт более равномерную распределённость векторов в выходной последовательности генератора. Действительно, спектральный критерий учитывает в первую очередь старшие разряды производимых генератором чисел и может пренебречь их младшими разрядами. Таким образом, если вернуться к приведённой выше реализации функции «get_random_tuple»: команда, присваивающая переменной «output[i]» значение указанного в коде выражения, отнюдь не является переусложнённой; присвоение

output[i] = rand()

кажущееся более естественным в данном случае, на самом деле является ошибочным – с точки зрения метода покомпонентной генерации векторов. Это даже заметно при проведении экспериментов, описанных в следующем разделе.

Если же требуется получить больший контроль над выходной последовательностью векторов, то можно выбрать конкретный линейный конгруэнтный генератор из тех, которые проходят проверку спектральным критерием (см., например, таблицу 1 в разделе 3.3.4 монографии [8]); после чего с его помощью можно генерировать псевдослучайные векторы покомпонентно – то есть по алгоритму, реализованному в функции «get_random_tuple»: производимая указанным способом последовательность векторов (согласно упоминавшейся выше теореме Нидеррайтера) будет выглядеть так, как будто она представляет собой последовательность реализаций случайной векторной величины, равномерно распределенной внутри некоторого многомерного куба.

Заметим попутно, что проблема построения эффективных числовых генераторов, которые обладали бы заранее заданными теми или иными специфическими свойствами, в настоящее время является актуальной: см., например, работу [9], где авторы представляют ряд новых результатов, связанных с данным направлением исследований.

III. Способ генерации псевдослучайных векторов большой размерности

Любые закономерности в псевдослучайных последовательностях нежелательны. Последовательности чисел, обладающие закономерностями, могут быть небезопасными для использования в большинстве криптографических алгоритмов; наличие закономерностей в выходных последовательностях псевдослучайного генератора может приводить к ошибкам при проведении с его помощью моделирования по методу Монте-Карло. Однако, существуют и такие задачи, где статистические свойства генератора не слишком существенны. Составление тестов системы команд микропроцессора, на наш взгляд, относится именно к таким задачам.

Вообще говоря, если в программе требуется производить генерацию псевдослучайных векторов большой размерности, то скорее всего, даже если генератор будет вызван очень большое число раз, общее количество произведённых им векторов окажется настолько мало, что говорить об их равномерной или неравномерной распределённости по гиперкубу можно будет лишь с большой долей условности.

В тех случаях, когда от генератора векторов требуется не столько псевдослучайность его выходной последовательности, сколько её повторяемость и высокая скорость генерации, авторы предлагают выполнять построение векторов с использованием следующего алгоритма.

Важное замечание. *Результат работы приведённой ниже функции зависит не только от значения глобальной переменной «seed», но и от содержимого массива «array» (длина которого полагается равной «length»).*

```
#define INITIAL_SEED 0

#include <stddef.h>

unsigned long seed = INITIAL_SEED;

void randomize_array(unsigned char* array,
 size_t length)
{   size_t i, j, s, w;
    s = 0;
    for (i = 0; i < length; i++) {
        w = (seed * s + i) & ((1lu << 32) - 1);
        j = w % length;
        array[j] = array[i] + w;
        s += array[i];    }
    seed += s;
    seed = (seed << 1)
 + ((seed & (1lu << 31)) == 0 ? 0 : 1);}
```

Данная функция на выходе формирует новое содержимое массива «array», элементы которого можно рассматривать как компоненты вектора, имеющего размерность «length».

Заметим, что хотя в предложенной реализации функции «randomize_array» используется операция получения остатка от деления на число, в большинстве случаев она легко может быть заменена на операцию конъюнкции – а именно, когда значение переменной «length» постоянно и является при этом степенью числа 2.

Указанный метод проводит рандомизацию массива «array» даже в том случае, если он изначально содержит только нули, и даже если при этом значение переменной «seed» равно 0. То есть, при использовании данного метода можно присваивать любые значения переменной компилятора «INITIAL_SEED» и использовать её, чтобы многократно получать те или иные фиксированные последовательности псевдослучайных векторов.

Оценка эффективности предложенного метода проводилась экспериментально по следующей формуле:

$$Q = N * length - b * best,$$

где Q – вычисляемая оценка качества (чем больше значение этой величины, тем лучшей признаётся работа алгоритма при заданных начальных значениях его параметров), N – сколько раз в цикле был выполнен вызов функции рандомизации фиксированного массива байтов; величины b и $best$ вычисляются особым образом в зависимости от того, как часто возникает ситуация, когда одним и тем же компонентам различных векторов выходной последовательности оказываются присвоены одни и те же числовые значения.

Результаты такой оценки в случае N = 100 000, $length$ = 100 оказались следующими: покомпонентный метод, т.е. основанный на функции «get_random_tuple», привёл к значению Q = 9 837 878, а в случае нового генератора «randomize_tuple» была получена оценка несколько более высокая оценка Q = 9 857 197. Необходимо уточнить, что при проведении данного эксперимента время от времени (а именно, через каждые 123 итерации) проводился намеренный сброс содержимого входного массива к начальным значениям его элементов: это было сделано для того, чтобы ужесточить условия тестирования.

Подсчитывалось также число коллизий контрольных сумм: они оказались равны, соответственно, 16 803 и 16 893 для алгоритмов обоих типов (соответственно, покомпонентного и нового): по данному показателю покомпонентный алгоритм демонстрирует чуть более эффективную работу.

Скорость работы обоих алгоритмов, как и следовало ожидать, оказалась приблизительно одинаковой.

Важно, что разница в оценках не сильно варьируется при изменении значения переменной компилятора «INITIAL_SEED» и изменении начальных значений элементов массива.

IV. Выводы

При составлении функциональных тестов микропроцессоров иногда требуется заполнить некоторую область памяти значениями, произведёнными псевдослучайным образом. Такая задача (по крайней мере, с формальной точки зрения) сводится к задаче генерации псевдослучайных векторов и может быть решена различными способами. Тривиальный способ состоит в том, чтобы заполнить память подряд идущими элементами некоторой последовательности псевдослучайных чисел (по сути этот способ представляет собой покомпонентный алгоритм генерации псевдослучайных векторов). Другой способ предложен в настоящей работе.

Одно из требований к функциональным тестам состоит в том, чтобы их результаты поддавались повторному воспроизведению. Это исключает, например, возможность использования в тестовых программах стандартной библиотечной функции «rand» языка C. Поэтому для включения в тестовую программу покомпонентного генератора псевдослучайных векторов промежуточный генератор псевдослучайных чисел также должен быть включён в ту же программу.

Также при проведении функционального тестирования существенными оказываются такие качества используемых генераторов, как скорость их работы и простота их реализации. Перечисленными качествами обладает генератор псевдослучайных векторов, предложенный в данной работе. Результаты его проверки, приведённые в разделе 3, показывают, что и значение величины Q, и число коллизий для функции «randomize_array» близки к соответствующим значениям функции «get_random_tuple». Это означает, что предложенный нами метод вполне применим для решения таких задач, как составление функциональных тестов микропроцессоров.

Литература

[1] Бобков С.Г. Методика тестирования микросхем для компьютеров серии «Багет» // Программные продукты и системы. 2007. №3. С. 2–5.

[2] Смирнов А.В., Чибисов П.А. Генератор тестов для проверки когерентности кэш-памятей многоядерных микропроцессоров (ristretto) // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2018. Вып. 2. С. 31–38.

[3] Ерохин В.В., Мальцев П.П. Самотестирование сложных функциональных блоков // Проблемы разработки перспективных микроэлектронных систем – 2005. Сборник научных трудов / под общ. ред. А.Л.Стемпковского. М.:ИППМ РАН, 2005. С. 500–507.

[4] Дябин М.И., Решетников А.В., Саксонов Е.А. Методика тестирования процессорного ядра системы на кристалле с x86-совместимым микропроцессором // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Вып. 3. С. 172–179.

[5] Архипкин В.Я., Дябин М.И., Ерохин В.В., Леохин Ю.Л. Построение высокопроизводительной СнК на основе 16-разрядного процессорного ядра // Проблемы разработки

перспективных микро- и наноэлектронных систем (МЭС). 2020. Вып. 4. С. 134–139.
[6] Neiderreiter H. Random Number Generation and Quasi-Monte Carlo Methods. Philadelphia, SIAM, 1992. 250 p.
[7] Marsagila G. Random Numbers Fall Mainly in the Planes // Proc. Natl. Acad. Sci. U.S.A. 1968. Vol. 61, no. 1. P. 25–28.
[8] Knuth D.E. The Art of Computer Programming. V. 2. Seminumerical Algorithms. 3rd ed. Reading, Massachusetts: Addison – Wesley, 1997. 762 p. (Кнут Д.Э. Искусство программирования. Том 2. Получисленные алгоритмы. 3-е изд.: Пер. с англ. М.: Издательский дом «Вильямс», 2004. 832 с.)
[9] Riera C., Roy T., Sarkar S., Stănică P. A Hybrid Inversive Congruential Pseudorandom Number Generator with High Period // European J. of Pure and Applied Math. 2021. Vol. 14, no. 1. P. 1–18.

A Fast Method of Generating Pseudo-Random Vectors of High Dimension for Testing Systems-on-Chip

M.I. Dyabin[1], A.V. Reshetnikov[2], E.A. Saksonov[3]

[1]"Kaskad" Ltd., Moscow, dyabin@mail.ru

[2]"Accord" Ltd., Moscow, a_reshetnikov@hush.com

[3]Moscow Technical University of Communications and Informatics, Moscow, saksmiem@mail.ru

***Abstract* — This paper offers a new algorithm for generating pseudo-random vectors in case when they are not required to be distributed uniformly. It uses an internal variable and the components of an input vector in order to build an output vector, but a feature of the generator is it not based on an auxiliary pseudo-random number generator. A testing procedure is described, which shows some of the method's advantages before the trivial method of filling the components of output vector by subsequent elements of a sequence of pseudo-random numbers.**

While testing microprocessors sometimes it is necessary to fill a buffer in memory with values generated in a pseudorandom way. Formally, such task can be reduced to the task of building a pseudorandom vector. But the quality of the generated pseudo-random vector sequence is not meaningful in this case: instead, the speed of the generator and the ease of its implementation may be important. Such is the method described in this article.

***Keywords* — pseudorandom vector generator, functional testing of microprocessor.**

References

[1] Bobkov S.G. Metodika testirovaniya mikroskhem dlya kompyuterov serii "Baget" (A methodology for testing circuits for the "Baget" series of computers) // Programmnyye producty I sistemy. 2007. №3. S. 2–5 (in Russian).
[2] Smirnov A.V., Chibisov P.A. Random Test Generator for Multicore Microprocessor Cache Coherence Verification (Ristretto) // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2018. Issue 2. P. 31-38. doi:10.31114/2078-7707-2018-2-31-38
[3] Erokhin V.V., Maltsev P.P. Self-testing of complex functional blocks // Problems of Perspective Microelectronic Systems Development - 2005. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2005. P. 500-507.
[4] Dyabin M.I., Reshetnikov A.V., Saksonov E.A. A methodology for testing the microprocessor core of a system on chip with a x86-compatible microprocessor // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 3. P. 172-179. doi:10.31114/2078-7707-2020-3-172-179 (in Russian).
[5] Arkhipkin V.Ya., Dyabin M.I., Erokhin V.V., Leokhin Yu.L. Designing a high-performance SoC based on a 16-bit processor core // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 4. P. 134-139. doi:10.31114/2078-7707-2020-4-134-139 (in Russian).
[6] Neiderreiter H. Random Number Generation and Quasi-Monte Carlo Methods. Philadelphia, SIAM, 1992. 250 p.
[7] Marsagila G. Random numbers fall mainly in the planes // Proc. Natl. Acad. Sci. U.S.A. 1968. Vol. 61, no. 1. P. 25–28.
[8] Knuth D.E. The Art of Computer Programming. V. 2. Seminumerical Algorithms. 3rd ed. Reading, Massachusetts: Addison – Wesley, 1997. 762 p.
[9] Riera C., Roy T., Sarkar S., Stănică P. A Hybrid Inversive Congruential Pseudorandom Number Generator with High Period // European J. of Pure and Applied Math. 2021. Vol. 14, no. 1. P. 1–18.