

СИСТЕМЫ АВТОМАТИЗИРОВАННОГО ПРОЕКТИРОВАНИЯ CAD-SYSTEMS

УДК 004.4'242

А. Л. Стемпковский, д-р техн. наук, академик РАН, директор,
Д. В. Тельпухов, канд. техн. наук, нач. отдела, **Р. А. Соловьев**, канд. техн. наук, науч. сотр.,
В. С. Рухлов, аспирант, мл. науч. сотр., e-mail: do1p@ya.ru,
Институт проблем проектирования в микроэлектронике РАН

Тестовая система для сравнения алгоритмов, увеличивающих надежность комбинационных схем

Предложено описание системы, состоящей из большого набора тестов и программы судьи (judge program). Система используется для оценки эффективности работы произвольных алгоритмов защиты логических или арифметических схем без обратных связей. При разработке системы использовался опыт подготовки задач для соревнований по спортивному программированию. Приведены подробности по разработке закрытой системы тестов и по методике начисления очков. Раскрыты детали работы программы судьи. Разработанная тестовая система свободно доступна онлайн в системе SPOJ для всех специалистов по надежным схемам. Система ранжирует алгоритмы и выстраивает их в список по числу полученных очков. Чем больше очков получил алгоритм, тем более надежные схемы он сгенерировал на основе исходных.

Ключевые слова: надежность, резервирование, логические схемы, спортивное программирование, тестирование

Введение

Задача увеличения надежности микроэлектронных схем является одной из ключевых в некоторых отраслях промышленности, например в космической отрасли [1]. Множество исследователей занимаются проблемами защиты логических и арифметических схем [2, 17]. Разработано большое число разнообразных алгоритмов для этой задачи [3–5]. Методы увеличения надежности комбинационных схем в большинстве своем основаны на введении структурной избыточности, также есть методики, основанные на ресинтезе, изменении структуры элементов голосования на более надежные и т. д. Однако в большинстве своем сравнение разработанных методов проводится с архаичным методом Triple Modular Redundancy (TMR) [6]. В некоторых работах не учитывается влияние надежности декодеров на общую надежность схем, так как считается, что декодеры или мажоритарные элементы не подвержены сбоям [7, 8]. Как следствие, разработчикам микроэлектронных устройств трудно определить, какой из методов использовать для защиты микросхем, в связи с отсутствием явной классификации методов по их эффективности. Поэтому зачастую выбор делается в пользу TMR — как самого простого и при этом довольно эффективного метода. В работе предлагается система, которая будет ранжировать различные алгоритмы по их эффективности, и разработчики микроэлектронных схем

смогут использовать методики, более продвинутые, чем TMR.

В работе приводится детальное описание методики создания набора тестов, которые будут выступать своеобразным эталоном при сравнении различных алгоритмов защиты логических схем от сбоев. Реализация тестирования сделана на базе онлайн системы оценки решений SPOJ [9, 10], которая разработана специально для ранжирования эффективности алгоритмов. Мы следуем по пути крупных Интернет-соревнований, которые регулярно проводят Netflix, Яндекс, Google, TopCoder и другие крупные IT-гиганты на своих площадках.

Предполагается, что систему будут использовать различные группы исследователей, работающие над задачей защиты комбинационных схем. Для использования тестовой системы потребуются написать программу, реализующую разработанный метод и соответствующую формальным параметрам, которые будут описаны ниже. Ценность тестовой системы будет тем выше, чем больше исследователей пришлют свои программные реализации алгоритмов для защиты схем. Программа на первом месте рейтинга будет считаться наиболее эффективной. Таким образом, независимые исследователи получат возможность сравнить свои результаты с результатами других групп, не вступая в непосредственный контакт.

Формализованное описание задачи

Формализованное описание задачи для использования в автоматической системе ранжирования должно быть одновременно максимально простым и максимально полным. Система работает онлайн без вмешательства человека, поэтому нужно однозначно определить:

- набор и формат входных данных;
- набор и формат данных, который должен выдать алгоритм пользователя, т. е. выходные данные;
- состав тестовых последовательностей;
- систему начисления очков.

При этом все данные должны быть в машиночитаемом виде.

Входные данные. Поскольку мы ориентируемся на современные средства разработки с проектированием на основе стандартных ячеек, то схемы нужно задать в виде набора логических ячеек и связей между ними. Технологические библиотеки могут содержать сотни элементов. Чтобы не усложнять чтение входных данных и последующее решение задачи, выберем только небольшую часть базовых ячеек, которые заведомо присутствуют в технологических библиотеках. Пусть их будет шесть основных из набора: инвертор (INV), логическое "И", логическое "ИЛИ", "И-НЕ", "ИЛИ-НЕ" и логическое "ИЛИ" (рис. 1). Предполагается, что разработанные алгоритмы могут быть легко адаптированы для случая наличия в библиотеке сотен логических элементов, в том числе многовыходных.

Наряду с остальными параметрами в технологической библиотеке, такими как площадь (S), каждая стандартная ячейка может характеризоваться, в том числе и вероятностью отказа (P). Скорее всего, это нелинейный параметр, который может зависеть от ряда факторов, но для простоты мы будем считать, что такой параметр задан однозначно — одним числом для одной ячейки.

Обычно заниматься увеличением надежности начинают после проектирования схемы, когда из-

вестен ее первоначальный размер. Из теории кодирования также известно, что чем больше избыточность, тем большей надежности можно достичь, однако производственные нормы накладывают ограничения на максимальный размер проекта. В задаче будем считать, что для каждой схемы нам задано число K — во сколько раз площадь защищенной схемы может превышать исходную. Мы также полагаем, что нас интересуют методы, альтернативные мажорированию, которые увеличивают площадь начальной схемы в разы. То есть K изменяется в довольно широких пределах от 2 и выше.

Чтобы разработчики алгоритма не занимались разработкой парсера для входных данных, подготовим их сразу в машиночитаемом формате. На первой строке укажем число тестовых схем Z . Далее следует Z схем, для каждой из которых сначала заданы требуемые параметры технологической библиотеки:

- в первой строке K — максимальная избыточность схемы по площади. На размер K требуется наложить некоторые ограничения, в нашем тестовом наборе $2 \leq K \leq 20$;
- в следующих шести строках указаны по два числа: площадь S ($1 \leq S \leq 100$) и вероятность сбоя q ($0 \leq q \leq 20$) в процентах, параметры каждого из логических элементов в следующем порядке: INV, AND, OR, NAND, NOR, XOR.

Затем идет описание самой схемы в следующем формате:

- число входов схемы: I ($0 < I \leq 250$), затем следует I названий входных узлов схемы (не более 20 символов в каждом), разделенных пробелами;
- число выходов схемы: O ($0 < O \leq 150$), затем следует O названий выходных узлов схемы (не более 20 символов в каждом), разделенных пробелами;
- число логических вентилях в схеме N , после чего следуют N строк с описанием каждого вентиля. Описание каждого вентиля начинается с его типа (одного из шести: INV, AND, OR, NAND, NOR, XOR), далее идут имена входных узлов, затем следует имя выходного узла. Входных узлов ровно два для каждого вентиля, кроме инвертора, где входной узел один. Выходной узел всегда один для всех вентилях.

В целом, описание схем напоминает структурный Verilog, но сделанный в более удобном для чтения из программ виде.

Выходные данные. Программа, реализующая алгоритм защиты, должна на основе исходной логической схемы (ИЛС) вывести модифицированную логическую схему (МЛС), которая выполняет ту же самую логическую функцию. То есть при любом наборе входных данных значения на всех выходах ИЛС и МЛС должны быть одинаковыми. Соответственно у МЛС должны быть в наличии все входные и выходные узлы, определенные в ИЛС. МЛС должна состоять из шести элементов, опре-

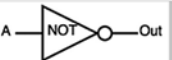

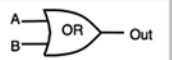



Название	Описание	Операция	Изображение
INV	Инвертор	$OUT=1A$	
AND	Логическое "И"	$OUT=A\&B$	
OR	Логическое "ИЛИ"	$OUT=A B$	
NAND	Инвертированное "И"	$OUT=\overline{(A\&B)}$	
NOR	Инвертированное "ИЛИ"	$OUT=\overline{(A B)}$	
XOR	Исключающее "ИЛИ"	$OUT=A\wedge B$	

Рис. 1. Базовые логические элементы

деленных в библиотеке, и ее площадь не должна превышать площадь более чем в K раз.

Формат вывода результирующей схемы похож на формат входных данных.

На первой строке число N — число логических элементов в МЛС. Далее следует N строк с описанием логических вентилях. Описание каждого вентиля начинается с его типа, далее идут имена входных узлов, затем — имя выходного узла.

Такую схему требуется вывести для каждого из Z тестов.

Описание и генерация схем. Поскольку задача поставлена для защиты комбинационных схем, то в наборе из тестовых схем используются только схемы без обратных связей. Для тестов было принято решение использовать схемы из известного бенчмарка ISCAS85 [11], а также несколько автоматически сгенерированных схем, поскольку набор ISCAS85 состоит всего из 11 схем, что недостаточно для полноценной тестовой системы. Некоторые схемы из набора ISCAS85 используются несколько раз, но с различными параметрами вероятности сбоя и избыточности.

В реальности ошибки в схемах происходят крайне редко. Поскольку процесс моделирования ошибок будет запускаться много раз, то нам желательно пропускать те случаи, когда схема работает верно. Иначе мы рискуем потратить на моделирование очень много времени. Поэтому для сочетания "библиотека плюс схема" необходимо подобрать вероятности таким образом, чтобы процесс моделирования не проходил впустую, т. е. чтобы хотя бы одна ошибка вносилась в схему достаточно часто и чтобы ошибок в схеме было не слишком много. Идеальным будет случай распределения ошибок, близкого к нормальному, пик которого находится около значения одна ошибка на схему.

В ходе исследований была разработана формула генерации вероятности появления ошибки для ячейки, чтобы выполнялись условия моделирования, близкие к требуемым:

$$P = \frac{1}{kN}, \quad (1)$$

где k — некий случайный коэффициент от 1 до 2; N — число логических элементов в схеме.

Для одной из схем с библиотекой, вероятности ошибок в которой получены по формуле (1), распределение числа ошибок во время моделирования Монте-Карло имеет вид, представленный на рис. 2.

Число ошибок на схему для всего тестового набора (даже для самых больших схем) редко превышает 12, и пик приходится на одиночную ошибку (SEU — single event upset) [20].

Для генерации случайных схем был разработан специализированный алгоритм. Для его реализации используется функция, которая принимает на



Рис. 2. График числа ошибок в зависимости от кратности для одного из тестов на 5000 запусков метода Монте-Карло

вход два числа W и H — желаемую "длину" и "высоту" логической схемы.

1. Число входов и выходов схемы берется как целое случайное число на промежутке от $H/2$ до H .

2. Генерируемая схема имеет W уровней, на каждом из которых число логических элементов считается по формуле $1 + \text{rand}(0, H - 1)$.

3. Тип логического элемента выбирается случайным образом из набора из шести элементов: INV, AND, NAND, AND, NOR, OR, XOR.

4. Соединения между элементами выбираются следующим образом. В массив `wires` помещаем все входные узлы. Далее двигаемся последовательно по уровням от входов к выходам схемы. Для входов элементов из каждого уровня случайным образом выбираем два различных элемента из массива `wires` (для инвертора один элемент). По окончании этого процесса для заданного уровня добавляем в массив `wires` все выходные узлы элементов из данного уровня.

Повторяем процесс для всех уровней, включая выходные узлы схемы.

Система начисления очков. Чтобы сравнить два алгоритма защиты схемы между собой на одной схеме, требуется сгенерировать защищенную версию исходной схемы каждым из алгоритмов, затем промоделировать обе эти схемы. Очевидно, что более эффективным будет алгоритм, который сгенерировал схему, число отказов на выходах которой в результате моделирования будет меньше.

Для оценки алгоритмов будем использовать метод Монте-Карло [12] и метрику COF (*correct output factor*) [13]. Метод Монте-Карло — это статистический метод, который характеризуется числом запусков моделирования TN . Чем больше TN , тем более точными получаются результаты моделирования. Обычно используют TN от 1000 и выше. По итогам каждого отдельного моделирования возможны два результата:

- на всех выходах схемы получены значения, совпадающие с эталоном, т. е. сбоя не было или сбой не привел к некорректному функционированию схемы;

- хотя бы на одном выходе результат отличается от эталона, т. е. в схеме произошел сбой, и он привел к неверному значению на выходе схемы.

Обозначим число сбойных тестов как ET . Тогда значение COF можно рассчитать по формуле

$$COF = \frac{TN - ET}{TN}. \quad (2)$$

Из формулы (2) видно, что значение COF лежит на промежутке от 0 до 1. Чем ближе значение к единице, тем более эффективно защищена схема. Надо также отметить, что в обычных условиях даже для схемы без защиты значение COF не равно 0. И даже для наиболее эффективно защищенных схем значение COF не равно единице из-за возможности сбоя на элементе, который подсоединен к выходу.

Чтобы сравнить эффективность алгоритмов защиты, недостаточно сравнить их работу только на одной схеме и при одних условиях работы. Для этого требуется некоторый набор тестовых схем. Так как значение COF нормировано на интервале от 0 до 1, то для сравнения достаточно посчитать эффективность $SCORE$ как сумму COF_i для всех TN тестов:

$$SCORE = \sum_{i=0}^{TN} COF_i. \quad (3)$$

Из формулы (3) видно, что значение $SCORE$ находится на интервале от 0 до N .

Задача в системе SPOJ

Система SPOJ работает без участия человека и оценивает присланные решения в автоматическом режиме. Поэтому задача требует полной формализации, а также защиты от накруток в системе начисления очков.

Для реализации задачи в системе SPOJ требуются:

- Input.txt — набор входных данных в текстовом формате;
- Output.txt — набор шаблонных выходных данных. Для нашей задачи этот файл не требуется, так как шаблон для выхода отсутствует;
- Judge.c — программа "Судья" на языке C (строго говоря программу "Судья" можно писать на любом языке, который поддерживает SPOJ).

На вход программы "Судья" подаются:

- *spoj_p_in — входные данные задачи (input.txt);
- *spoj_p_out — выходные данные задачи (output.txt);
- *spoj_t_out — выходные данные тестируемой программы;
- *spoj_t_src — исходный код тестируемой программы.

На выходе программа "Судья" может выдавать следующие данные:

- *spoj_score — число очков, которые заработала программа (обязательный параметр);

*spoj_p_info — информация о моделировании для автора задачи (по сути, закрытая информация);

*spoj_u_info — информация о моделировании для автора алгоритма (открытая для автора решения информация).

В нашей программе "Судья" используются: spoj_p_in — отсюда читаются исходные схемы в формате, описанном в разделе "Входные данные", файл spoj_t_out, который был сгенерирован тестируемой программой. В этом файле находятся описания для схем, защищенных от сбоев в формате, описанном в разделе "Выходные данные".

На выходе программа выдает файл spoj_score, с числом очков, полученным тестируемой программой в формате float, например 12.046564, а также служебную информацию с подробностями о работе программы в файле spoj_p_info, который доступен для просмотра только администраторам. Эта информация включает: число очков за каждый отдельный тест, число ошибок, внедренных в каждом конкретном тесте, используемую площадь в процентах от максимально возможной и время, затраченное на моделирование каждого теста.

Пример служебной информации для одного теста для числа итераций Monte Carlo $TN = 5000$:

```
Score for Test #2: 0.564800 [Err tests: 2670, Incorrect: 2176]
Error distribution: [0 — 2330] [1 — 1870] [2 — 662] [3 — 124]
[4 — 11] [5 — 3]
Area used: 354.0 from 1451.4 (24.39 %)
Time for test modeling: 0.003 sec
```

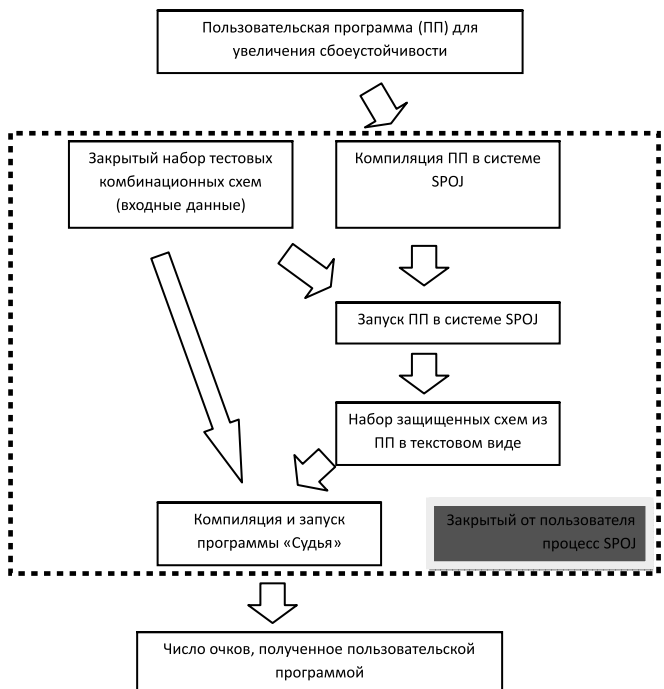


Рис. 3. Поток данных в системе SPOJ

Рейтинг участников для задачи в системе SPOJ

Место	Дата	Пользователь	Результат	Время	Память	Язык программирования
1	2015-02-18 11:49	Sergey	21,509933	12,49 с	10 Мбайт	C
2	2014-11-27 12:58	Flago	11,552267	32,98 с	24 Мбайт	PHP
3	2014-08-09 00:22	Robert Gerbicz	0,1058	2,36 с	11 Мбайт	C++4.3.2
4	2014-08-09 02:08	Mitch Schwartz	0,1058	0,1 с	1,9 Мбайт	BF

Последовательность работы программы "Судья" для каждого отдельного теста:

1. Чтение тестовой схемы.
2. Чтение схемы, сгенерированной тестируемой программой.
3. Проверка условия, что площадь тестируемой схемы не превышает требуемую.
4. Тестирование по методу Монте-Карло, при котором определяется, что схемы работают одинаково при случайных тестовых последовательностях на входах. В случае обнаружения ошибки программа "Судья" выдает сообщение об ошибке.
5. Моделирование неисправностей по методу Монте-Карло и подсчет значения COF.

Исходный код программы "Судья" доступен по ссылке [14]. Задача, оформленная с текстом для неспециалистов, доступна для решения по ссылкам [18, 19]. Решения принимаются более чем на 40 языках программирования. Поток данных в системе показан на рис. 3.

Ограничения в системе SPOJ

В процессе реализации системы тестирования мы столкнулись с некоторыми ограничениями. Процесс моделирования методом Монте-Карло оказался довольно затратным по времени. В итоге, чтобы программа "Судья" работала в приемлемые сроки, пришлось:

- снизить общее число тестов с планируемых 2000 до примерно 400;
- уменьшить число итераций Монте-Карло с 10 000 до 5000 для маленьких схем и до 3000 для больших схем;
- оптимизировать код программы "Судья".

После всех этих манипуляций удалось сократить время работы программы "Судья" с нескольких минут до примерно 50...60 с.

Сама система SPOJ накладывает ограничения (правда не строгие) на размер исходного кода. В данный момент он ограничен 50 Кбайт, а также на время выполнения тестируемой программы. Администраторы системы не рекомендуют ставить очень большие значения. Сейчас время выполнения ограничено 50 с.

Метод Монте-Карло является недетерминированным и зависит от генератора случайных чисел, поэтому число полученных очков, хотя и незначительно, но может варьироваться для схожих решений. Также существует вероятность, что тестируе-

мая схема, не эквивалентная исходной, пройдет тест на эквивалентность (но вероятность астрономически мала).

Одним из простейших решений задачи является вариант — вывести исходные схемы в качестве ответа. Выяснилось, что значение SCORE при этом получается достаточно большим по отношению к теоретическому максимуму, и разница между этим значением и реальными решениями получается не очень существенной. Поскольку нас интересуют только алгоритмы, в которых происходит улучшение надежностных характеристик схемы, то было принято решение изменить формулу расчета очков (3) на следующую:

$$SCORE = \left(\sum_{i=0}^{TN} COF_i \right) - DEF,$$

где DEF — это суммарное значение COF для исходной схемы без защиты. В нашем случае DEF получилось равным 276. В случае если SCORE равен отрицательному числу, то такое решение получает 0 очков. После этого изменения все реальные решения задачи стали визуально сильно отличаться от стандартных решений (см. таблицу).

Чтобы облегчить авторам алгоритмов задачу, мы упростили формат входных данных, сократили число требуемых для обработки параметров задачи (уменьшили число стандартных ячеек и оставили площадь и вероятность отказа в виде цифровых коэффициентов), а также подготовили:

1. Программу-пример на языке C, которая читает входные данные и выдает схему без изменений на выход [15].
2. Набор тестов, сходный с набором тестов на сервере для локального тестирования алгоритмов [16].
3. Программа "Судья", которая используется на сервере для отладки своей программы с точки зрения эффективности решения [14].

Исследование выполнено за счет гранта Российского научного фонда (проект № 14-19-01036).

Список литературы

1. **Niranjan S., Frenzel J. F.** A comparison of fault-tolerant state machine architectures for space-borne electronics // Reliability, IEEE Transactions on. 1996. Vol. 45, N. 1. P. 109—113.
2. **Choudhury M. R., Mohanram K.** Reliability Analysis of Logic Circuits // IEEE transactions on computer-aided design of integrated circuits and systems. 2009. Vol. 28, N. 3.

3. **Huang C.** Robust Computing with Nano-scale Devices. Springer, 2010. 180 p.
4. **Han J., Leung E., Liu L., Lombardi F.** A Fault-Tolerant Technique using Quadded Logic and Quadded Transistors // IEEE Transactions on VLSI Systems. 2014. P. 1–5.
5. **Poolakaparambil M., Mathew J., Jabir A.** Multiple Bit Error Tolerant Galois Field Architectures Over GF(2^m) // Electronics. 2012. N. 1. P. 3–22.
6. **Shubham C. A., Kolte M. T.** Fault Tolerant and Correction System Using Triple Modular Redundancy // International Journal of Emerging Engineering Research and Technology. 2014. Vol. 2. P. 187–191.
7. **Vial J., Bosio A., Girard P.** et al. Using TMR architectures for yield improvement // Defect and Fault Tolerance of VLSI Systems, 2008 in DFTVS'08. IEEE International Symposium on. IEEE. 2008. P. 7–15.
8. **Ebrahimi M., Miremadi S. G., Asadi H.** ScTMR: A scan chain-based error recovery technique for TMR systems in safety-critical applications // Design, Automation & Test in Europe Conference & Exhibition (DATE). 2011. IEEE, 2011. P. 1–4.
9. **Онлайн** система оценки решений SPOJ. URL: <http://spoj.com>
10. **Описание** системы SPOJ. URL: <http://www.spoi.com/info/>
11. **Bryan D.** The ISCAS'85 benchmark circuits and netlist format. North Carolina State University. 1985. P. 4.

12. **Соболь И. М.** Численные методы Монте-Карло. М.: Наука, 1973. 312 с.
13. **Стемпковский А. Л., Тельпухов Д. В., Соловьев Р. А., Соловьев А. Н., Мячиков М. В.** Моделирование возникновения неисправностей для оценки надежных характеристик логических схем // Информационные технологии. 2014. № 11. С. 30–36.
14. **Исходный** код программы "судья". URL: <http://vscrip ts.ru/docs/zrely/Judge.zip>
15. **Программа** пример на языке Си. URL: <http://vscrip ts.ru/docs/zrely/ZRELY/prog.cpp>
16. **Набор** тестов для локального тестирования алгоритмов. URL: <http://vscrip ts.ru/docs/zrely/input.zip>
17. **Lynch J. D.** Stochastic fault simulation of triple-modular redundant asynchronous pipeline circuits: Thesis (Ph. D.). — Oregon Health & Science University. 2009.
18. **Русская** версия задачи. URL: <http://www.spoj.com/ZELARCH/problems/ZRELY/>
19. **Английская** версия задачи. URL: <http://www.spoj.com/problems/ZRELY1/>
20. **Описание** типа ошибки. URL: <http://radhome.gsfc.nasa.gov/radhome/see.htm>

A. L. Stempkovskiy, Academician, Director, **D. V. Telpukhov**, Ph. D., Head of the Department, **R. A. Solovyev**, Ph. D., Researcher of the Department, **V. S. Rukhlov**, Junior Researcher, IPPM RAS

Test System for Comparing Algorithms which Increases the Fault Tolerance of Combinational Circuits

Description of the system consisting of a large set of tests and a judge program is proposed. The system is used for performance evaluation of arbitrary algorithms of protection of logic and arithmetic circuits without feedback. While developing the system, experience in preparing tasks for sports programming competitions was used. The paper presents the details of the closed system of tests development and of the scoring methodology. Details of the judge program work are revealed. The developed test system is freely available online from the SPOJ system for all reliability professionals. The system ranks the algorithms and arranges them in a list according to their total score. The more points the algorithm received, the more reliable circuits based on the initial version it generated.

Keywords: reliability, redundancy, logic circuits, sports programming, testing

References

1. **Niranjan S., Frenzel J. F.** A comparison of fault-tolerant state machine architectures for space-borne electronics, *Reliability, IEEE Transactions on*, 1996, vol. 45, no. 1, pp. 109–113.
2. **Choudhury M. R., Mohanram K.** Reliability Analysis of Logic Circuits, *IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEM*, 2009, vol. 28, no. 3.
3. **Huang C.** Robust Computing with Nano-scale Devices, Springer, 2010, 180 p.
4. **Han J., Leung E., Liu L., Lombardi F.** A Fault-Tolerant Technique using Quadded Logic and Quadded Transistors. *IEEE Transactions on VLSI Systems*, 2014, pp. 1–5.
5. **Poolakaparambil M., Mathew J., Jabir A.** Multiple Bit Error Tolerant Galois Field Architectures Over GF(2^m), *Electronics*, 2012, no. 1, pp. 3–22.
6. **Shubham C. A., Kolte M. T.** Fault Tolerant and Correction System Using Triple Modular Redundancy, *International Journal of Emerging Engineering Research and Technology*, 2014, vol. 2, pp. 187–191.
7. **Vial J., Bosio A., Girard P.** et al. Using TMR architectures for yield improvement. *Defect and Fault Tolerance of VLSI Systems, 2008, DFTVS'08. IEEE International Symposium on. IEEE*, 2008, pp. 7–15.
8. **Ebrahimi M., Miremadi S. G., Asadi H.** ScTMR: A scan chain-based error recovery technique for TMR systems in safety-critical

- applications. *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011, IEEE*, 2011, pp. 1–4.
9. <http://spoj.com>
10. <http://www.spoj.com/info/>
11. **Bryan D.** The ISCAS'85 benchmark circuits and netlist format. North Carolina State University. 1985.
12. **Sobol' I. M.** Chislennyye metody Monte-Karlo. Moscow: Nauka, 1973. 312 p.
13. **Stempkovskij A. L., Tel'puhov D. V., Solov'ev R. A., Solov'ev A. N., Mjachikov M. V.** Modelirovanie vozniknovenija neispravnostej dlja ocenki nadezhnostnyh harakteristik logicheskikh shem, *Informacionnyye tehnologii*, 2014, no. 11, pp. 30–36.
14. <http://vscrip ts.ru/docs/zrely/Judge.zip>
15. <http://vscrip ts.ru/docs/zrely/prog.cpp>
16. <http://vscrip ts.ru/docs/zrely/input.zip>
17. **Lynch J. D.** Stochastic fault simulation of triple-modular redundant asynchronous pipeline circuits: Thesis (Ph. D.). Oregon Health & Science University, 2009.
18. <http://www.spoj.com/ZELARCH/problems/ZRELY/> — russkaja versija zadachi
19. <http://www.spoj.com/problems/ZRELY1/> — anglijskaja versija zadachi
20. <http://radhome.gsfc.nasa.gov/radhome/see.htm>