Обнаружение системных дефектов цифровых устройств при имитации неисправностей с применением фаззинга

Detecting system defects of digital devices in the course of malfunction imitation using fuzzing

Панков Д.А.[1], Панков И.А.[2]*
Pankov D.A.[1], Pankov I.A.[2]*
[1]АО «ОНИИП», Омск, Российская Федерация, [2]Омский государственный технический университет, Омск, Российская Федерация
[1]AO ONIIP, Omsk, Russian Federation, [2]Omsk State Technical University, Omsk, Russian Federation
**pankovddd@ya.ru*

Панков Д.А.

Панков И.А.

Резюме. Цель. *В статье предложены подходы к организации испытаний цифровых систем с помощью имитации неисправностей для выполнения требований международных и отечественных стандартов по обеспечению устойчивости к отказам и сбоям с целью эффективного (по времени) поиска дефектов ПО для выпуска серийных изделий. Предлагается структура и алгоритм работы программно-аппаратного стенда имитации неисправностей для проведения испытаний устройств комплекса. Стенд реализует сбор и подготовку данных для фаззинга, выявления ошибок на аппаратном уровне, а также определение объема испытаний.* **Методы.** *Применялись основы системного анализа, классические методы теории вероятности и математической статистики, теории принятия решений, методы тестирования и разработки программно-аппаратных систем, математическая теория нечетких множеств и нечеткой логики.* **Результаты.** *Разработаны алгоритмы имитации неисправностей для тестирования программно-аппаратных систем с помощью техники фаззинга, обеспечивающие вероятностную оценку момента окончания испытаний по тестированию с заданной точностью.* **Выводы.** *Данный набор алгоритмов позволяет обнаружить системные дефекты при объединении программных и аппаратных средств в единый комплекс, приводящему к возникновению новых неисправностей (свойство эмерджентности), которые невозможно учесть на этапе проектирования.*

Abstract. Aim. *The paper proposes approaches to the organisation of testing of digital systems through malfunction imitation for the purpose of ensuring compliance of international and Russian failure and fault resistance standards for the purpose of efficient (in terms of time) detection of software defects as part of mass production of products. The paper proposes a structure and operating algorithm of a hardware and software test bed for malfunction simulation intended for testing a system's devices. The test bed collects and processes data for fuzzing, hardware error identification, as well as defines the scope of testing.* **Methods.** *The paper used basic systems approach, classical methods of the probability theory and mathematical statistics, decision theory, methods of hardware and software testing and development, mathematical theory of fuzzy sets and fuzzy logic.* **Results.** *Malfunction simulation algorithms were developed for the purpose of testing hardware and software systems using fuzzing that ensure probabilistic estimation of the termination time of testing with a specified accuracy.* **Conclusions.** *The above set of algorithms allows detecting system defects in the process of software and hardware integration into a single system that cause new malfunctions (emergence) that cannot be taken into consideration at the design stage.*

Ключевые слова: *имитация неисправностей, отказы и сбои программно-аппаратных систем, тестирование программного обеспечения, устойчивость к отказам и сбоям, фаззинг.*
Keywords: *malfunction imitation, failures and faults of hardware and software systems, software testing, failure and fault resistance, fuzzing.*

Для цитирования: *Панков Д.А., Панков И.А. Обнаружение системных дефектов цифровых устройств при имитации неисправностей с применением фаззинга // Надежность. 2023. №4. С. 51-58. https://doi.org/10.21683/1729-2646-2023-23-4-51-58*
For citation: *Pankov D.A., Pankov I.A. Detecting system defects of digital devices in the course of malfunction imitation using fuzzing. Dependability 2023;4:51-58. https://doi.org/10.21683/1729-2646-2023-23-4-51-58*

Поступила: *08.05.2023* / **После доработки:** *27.10.2023* / **К печати:** *20.11.2023*
Received on: *08.05.2023* / **Upon revision:** *27.10.2023* / **For printing:** *20.11.2023*

Введение

Согласно стандарту РФ для вычислительной техники Р 56939-2016 «Защита информации. Разработка безопасного программного обеспечения» и приказу ФСТЭК России № 76 от 02.06.2020 года для программно-аппаратных систем необходимо применять статический и динамический анализ кода программ, фаззинг-тестирование, а также проводить проверку на уязвимости и недекларированные возможности ПО. Однако в ряде случаев использование инструментов фаззинга не позволяет обнаружить дефекты ПО и системные ошибки и не эффективнее классического тестирования программ [1]. Под фаззингом понимают специальную технику поиска ошибок в программном обеспечении, нацеленную на их проявление при искажении входных данных. Тем не менее, современные программы фаззинга сложно адаптировать под существующие комплексы, что подтверждается поддержкой подобных проектов научными институтами (ИСП РАН – разработка стендов для программ комплексов с ОС Astra Linux), а также обеспечить эффективность поиска системных ошибок не отдельно в тестовой среде, а при загрузке пользовательских программ в реальную аппаратную среду устройства (в устройство с набором микроконтроллеров).

Поскольку ведущие ученые в области отказоустойчивости сложных систем утверждают [2], что существуют лишь частичные меры по повышению устойчивости устройств с ПО к отказам и сбоям, то можно выделить два направления процесса разработки [3]. Если рассмотреть фаззинг со смысловой, а не терминологической составляющей, то прототипами методики, из которой появилась техника фаззинга, можно считать системы «UDS» и «JPL-STAR», разработанные в Лаборатории реактивного движения Массачусетского технологического университета в период 1961-1972 гг., где, как считается, впервые была построена самовосстанавливающаяся (лабораторная, демонстрационная) система, получившая название «JPL-STAR» [2]. Эта система ориентирована на технологию биполярных элементов с малым и средним уровнем интеграции и памятью на цилиндрических магнитных доменах. В макете «JPL-STAR» для ОЗУ устройства производилась имитация неисправностей регулируемой длительности – с 0 на 1 и наоборот, причем этот эксперимент показал самовосстановление системы на высоком уровне.

Для проектирования надежных архитектур используют два направления [3]. К первому направлению относится построение качественной архитектуры систем, которая за счет избыточности подсистем защиты обеспечивает работоспособность при отказах и сбоях. Ко второму направлению относится изучение проектируемого объекта с помощью внесения ошибок для проявления отказов и сбоев в результате имитации неисправностей или при реальном тестировании, в рамках которого проводится имитация неисправностей с применением техники фаззинга. Интерес представляет рассмотрение этих подходов совместно для понимания зон ответственности каждого подхода и их объединения для решения задачи повышения устойчивости устройств к отказам и сбоям.

Модели фаззинга основаны на имитации неисправностей с помощью искажения входных данных для тестируемого ПО. Искажение производится путем мутации существующих наборов данных и/или генерации новых данных с помощью встроенных в фаззеры алгоритмов. В отличие от статического анализа, рассматриваемый подход направлен на исследование программ в процессе их выполнения. Поскольку основной вопрос исследования при имитации неисправностей состоит в том, каким образом найти критические ошибки в области пространства ошибок (которые оказывают значимое влияние на поведение системы), то обычно совместно используют сразу несколько подходов для обеспечения требуемого результата.

Например, применяется *анализ чувствительности* (используются статистически часто применяемые сигналы или параметры для изучения реакции системы; сигналы, направленные на задействование определенной области ПО или режима работы), *обучение с подкреплением* для эффективного исследования пространства ошибок и поиска критических неисправностей, а также т.н. *анализ реализма* (с помощью сбора статистических показателей), при котором вводимые ошибки определяются на основе ранее обнаруженных дефектов, исправленных в макетных образцах.

Способы имитации неисправностей

Создание средств имитации неисправностей возможно на различных уровнях проектируемого устройства, таких как уровень входных данных X, внутреннего состояния Y_n, результатов работы устройства Z и т.д. (рис. 1). Для внесения искажений возможно использовать программные и аппаратные средства. Аппаратные средства представляют технические системы (контактные – интерфейс прямого доступа в память и бесконтактные – облучение микросхем памяти и т.д., которые воздействуют на систему на физическом уровне). Программные средства искажают значения памяти, регистров в ОЗУ и на интерфейсах обмена с компонентами системы.

Выбор аргументов или функции Ψ исправного устройства позволяет определить тип используемого имитатора неисправностей, который удовлетворяет требованиям к стоимости разработки и направленности поиска в проектируемой системе. Фаззинг является одним из вариантов имитации неисправностей, который использует компонент X для внесения искажений.

Эффективность поиска дефектов ПО с помощью фаззинга для программно-аппаратных систем зависит от подготовленных входных данных. Фаззинг может просто «зависнуть» на определенных путях программы, не находя новых ветвлений. Увеличение объема

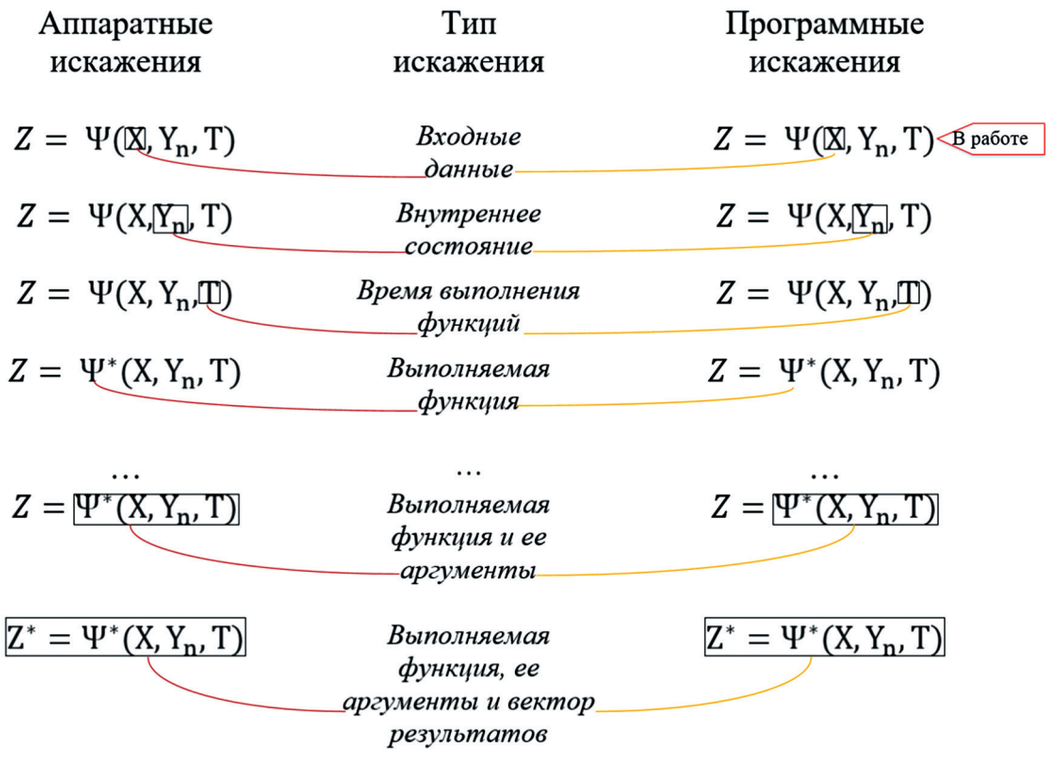

Рис. 1. Способы имитации неисправностей

тестируемого кода не всегда позволяет найти дефекты, так как они проявляются только в определенном наборе данных (проблема робастности).

Для программно-аппаратных систем предлагается использовать разработанный имитатор неисправностей (рис. 2) [4], совмещающий поиск дефектов как на аппаратном, так и программном уровне. Рабочая станция для имитации неисправностей представляет собой устройство с запущенным алгоритмом сбора статистики и *JTAG*-программой, которая на основе переданных команд алгоритма модифицирует содержимое памяти микропроцессоров на основе сигнатурного анализа. Также на рабочей станции запущен экспертный алгоритм, который корректирует зоны применения алгоритма для увеличения скорости моделирования отказов и сбоев [5]. Набор испытаний реализован в алгоритме анализа проблемных ситуаций, возникающих при работе автоматизированной радиостанции, интегрируемой в действующую инфраструктуру цифровых сетей с множественным доступом, и реализует тестирование функций устройства с помощью техники фаззинга.

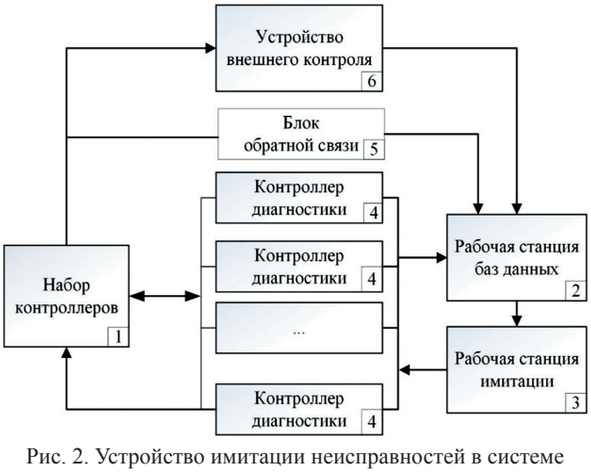

Рис. 2. Устройство имитации неисправностей в системе тестирования

Алгоритм тестирования с применением фаззинга

Алгоритм фаззинга дополняется предварительным разработанным статистическим анализом выполняемой программы и сбором данных перед проведением фаззинга в период приемо-сдаточных испытаний в рамках выполнения опытно-конструкторских работ. Алгоритм включает искажение входных данных с помощью устройства имитации неисправностей (модифицированный алгоритм фаззинга на основе сбора статистики о функциях работы устройства) и программный блок, который позволяет определить критерий окончания испытаний [6] с применением математической модели на основе нечеткого логического вывода (рис. 3 и 4).

Представим набор испытаний, который реализуется в рамках выполнения данного алгоритма:

1) сбор и анализ исходных данных для микропроцессорного устройства;

2) формирование и загрузка исходных данных в микропроцессорное устройство;

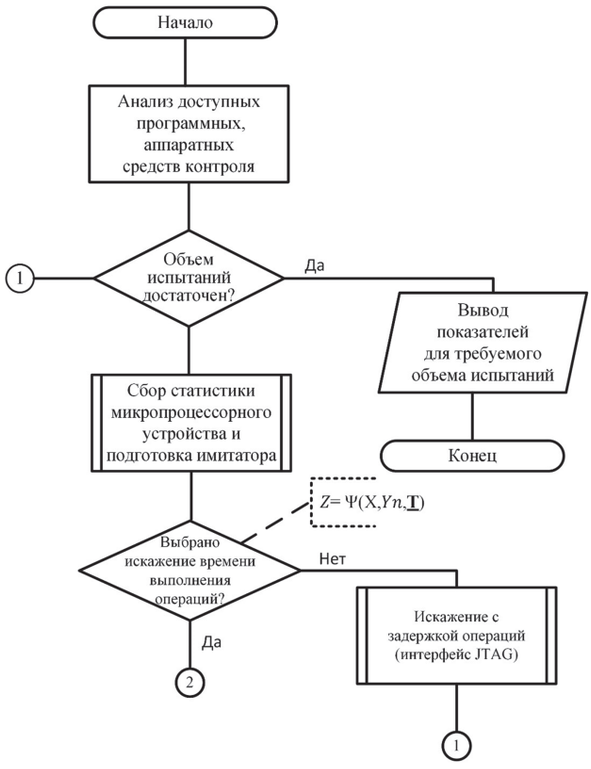

Рис. 3. Алгоритм анализа проблемных ситуаций устройства

3) подключение имитатора неисправностей и средств синхронизации для анализа и диагностики;

4) запуск управляющего стенда для проведения испытаний;

5) инициирование работы в основных режимах устройства;

6) проведение пошагового исследования для определения критерия окончания испытаний;

7) получение критерия окончания и косвенных признаков для критерия, параметров среднего времени наработки на отказ при проведении испытаний.

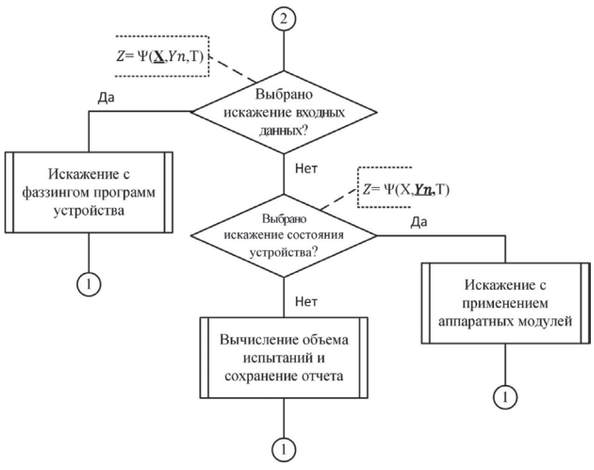

Рис. 4. Алгоритм анализа проблемных ситуаций устройства (продолжение)

Для осуществления данных пунктов в рамках методики производится ряд вычислений с помощью средств диагностики (на основе *JTAG*). Это позволяет автоматически выполнять диагностику технического состояния для дальнейшего устранения системных ошибок, проявляющихся при загрузке пользовательских программ в реальную аппаратную среду устройства (проявление свойства эмерджентности [6]).

Для испытания устройства важным является изучение свойств программы при ее эксплуатации на устройстве (в составе программно-аппаратного комплекса). Динамическими показателями программы является граф ветвлений программы, статистика эксплуатации ассемблерных команд и программных функций. Поскольку свойства программы зависят от режима работы устройства, то имеет смысл изучение свойств программы в рамках работы установленного режима. Следует отметить, что при разработке программ часто используют точки контроля. Под точкой контроля понимается текущий адрес памяти, для которого в момент времени выполнения возможно определить содержимое регистров и данных. Т.е. в точке контроля отклонение в значениях, хранимых в регистрах и памяти при различных входных данных минимально.

При сборе информации введем показатели, которые позволят сформировать статистику для режимов работы устройства (рис. 5 и 6):

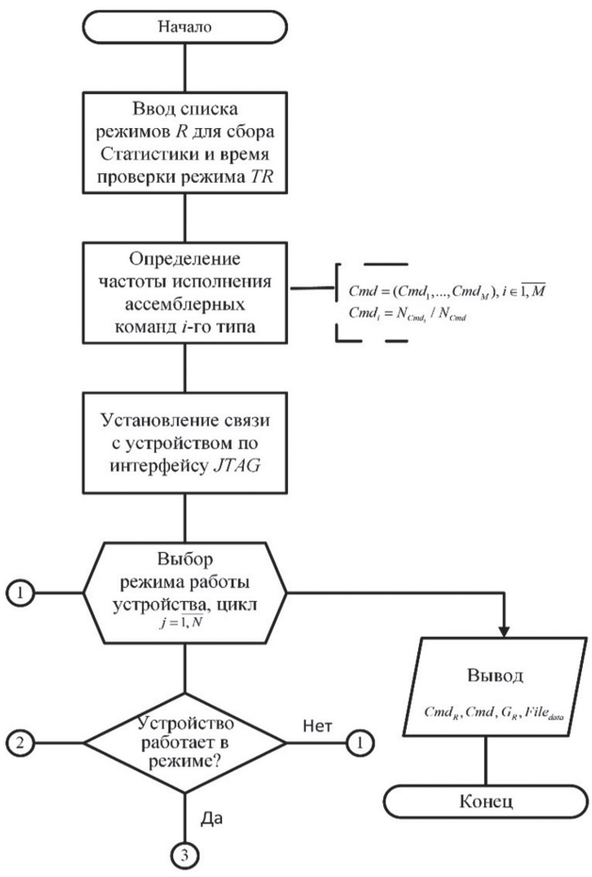

Рис. 5. Алгоритм предобработки исходных данных

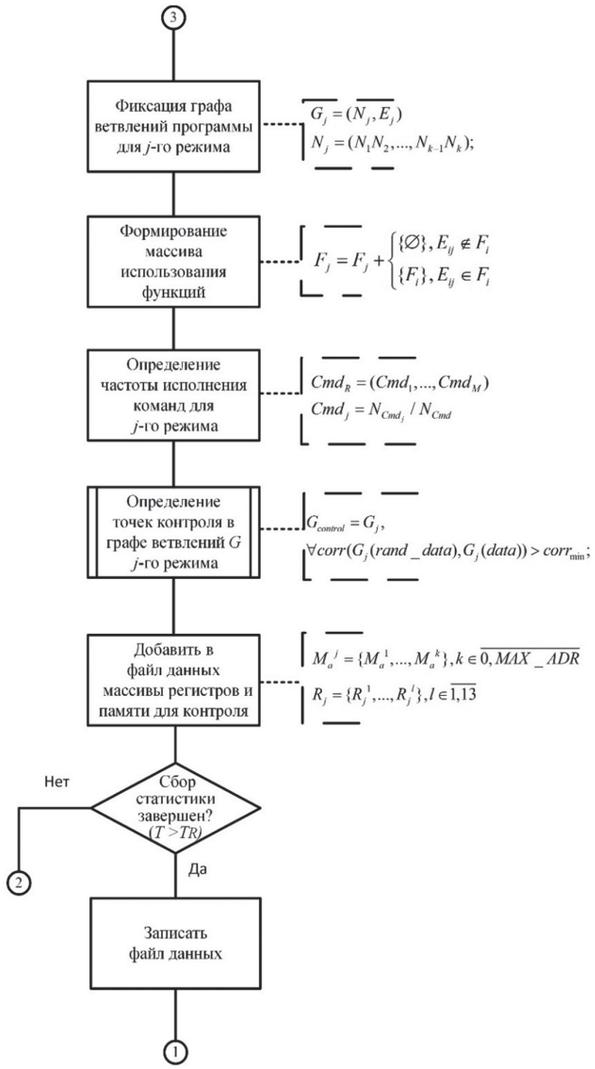

Рис. 6. Алгоритм предобработки исходных данных (продолжение)

Cmd_R – массив частот исполнения ассемблерных команд в заданном режиме;

Cmd – статический массив частот исполнения ассемблерных команд;

G_R – граф вызовов функций для выбранного режима;

M_a – массив состояний оперативной памяти микропроцессорного устройства;

R_j – массив состояния регистров микропроцессорного устройства;

ε_β – допустимая погрешность оценки среднего времени наработки на отказ;

T_R – время сбора статистики на одну программную функцию.

Сбор статистики производится в основных режимах работы устройства, для которых требуется обеспечить устойчивость к отказам и сбоям. Полученная статистика используется в качестве начальных данных для работы алгоритма фаззинга, который загружает шаблоны, позволяющие достигнуть узлов графа, где располагаются основные функции, которые эксплуатируются при работе устройства в целевых режимах микропроцессорного устройства.

Граф ветвлений записывается для каждого шага работы алгоритма в виде файлового массива. Для массивов выполняется ранжирование. Ранжированные данные используются при загрузке в алгоритм фаззинга, что позволяет обойти ряд ограничений для алгоритма фаззинга, которые трудно реализуемы без модулей предварительного контроля и анализа реакции (или анализа параметров только программы без учета характеристик проектируемого устройства).

Результатом работы алгоритма является набор массивов данных, который передается эвристическому итерационному алгоритму фаззинга. За счет предобработанных массивов данных удается достигнуть большего количества обнаруженных ошибок в режимах работы устройства. Стенд имитации интегрирован с системой разработки программного кода. В рамках данной системы используется подход по непрерывной интеграции, где стенд применяется при каждом релизе ПО.

Локализация места возникновения дефекта в программном коде, который реализует доступ к аппаратным модулям, является важным для обеспечения отказоустойчивости всего устройства. Для локализации зон ответственности каждой функции все ветвления отслеживаются с помощью средств анализа ветвлений (*control flow graph*). Если внутри ветвления используются функции доступа к аппаратным ресурсам, то возможно подключение контроля выполнения аппаратных средств в момент тестирования для зон, где сложно установить корректность работы на программном уровне. Процесс тестирования должен происходить внутри программной среды, которая хранит все тесты для версий и процесс правок после нахождения дефектов как программных, так и аппаратных компонентов.

Определение объема испытаний на стенде

Основной целью, достигаемой с помощью фаззинга, является выявление максимального числа ошибок за установленный временной лимит, а также увеличение числа рассмотренных ветвей графа программы. Как показано в работах ряда авторов, фаззинг позволяет решать задачу об оптимизации количества найденных уязвимостей за установленный временной диапазон [7]

$$\sum_{i=1}^{N} b_x \to \max. \qquad (1)$$

Каждый отдельный сбой или отказ фиксируется для набора данных, который вызвал его возникновение

$$c_{ij} = \begin{cases} 1, j\text{-й } \textit{отказ включает } i\text{-й } \textit{набор данных}; \\ 0, \textit{в противном случае.} \end{cases} \qquad (2)$$

Определим признак обнаружения ошибки на тестовых данных, который фиксирует ошибки для уникальных программных функций

$$b_x = \begin{cases} 1, \textit{тогда и только тогда, когда } \exists i, j : f(c_{ij}) = x; \\ 0, \textit{в противном случае.} \end{cases} \quad (3)$$

При этом следует учесть ряд ограничений для выявления бо́льшего числа ошибок за установленное время поиска [8]:

$$\forall i, j : c_{i,j+1} \le c_{i,j};$$

$$\sum_{i,j} c_{ij} \cdot t_{ij} \le t_{thres};$$

$$\forall i, j : c_{ij} \le b_x, \text{ где } f(c_{ij}) = x;$$

$$\forall x : b_x \le \sum_{i,j} c_{ij}. \quad (4)$$

Для подготовки данных перед проведением фаззинга используются описанные выше алгоритмы сбора тестовых данных, которые реализованы как самостоятельные программно-аппаратные средства. Поскольку проведение полного тестирования программы устройства может потребовать слишком много времени, фаззинг применяют к критическим режимам работы устройства. Для этого выделяется конечный набор функций, которые необходимо протестировать с возможностью оценки результата их выполнения. В результате работы алгоритма фаззинга на каждой итерации формируется матрица ветвлений, суммирующая количество переходов для каждой из вершин графа (рис. 7).

При этом мутации исходных данных нацелены на исследование отклонений реальной работы программы от ее алгоритма. Вариантом организации фаззинга является комбинация генерационного и мутационного искажения данных. Выбор типа искажения в конкретном случае производится на основе определяемой разработчиками стратегии тестирования. Для этого в алгоритме происходит динамический поиск и добавление новых условных переходов (ветвлений).

Архитектура ПО (обобщенная модель информационной системы) позволяет декомпозировать программу на подсистемы, которые проще контролировать в процессе работы. Поэтому контроль показателей (формула (5)) на каждом уровне подсистемы позволяет повысить качество за счет ускорения обнаружения отклонения системы по отклонению в поведении ее компонентов. Для спроектированной архитектуры часто используют определение коэффициента надежности ПО [2, 9]:

$$R = \sum_{j=1}^{M} \sum_{i=1}^{N_j} PU_{ij} R_{ij}, \quad (5)$$

где M – число уровней архитектуры ПО, N_j – число компонентов на j-м уровне, $j = \overline{1, M}$, PU_{ij} – вероятность использования компонента, R_{ij} – надежность компонента.

Для заданной архитектуры производится связывание времени наработки на отказ по уровням системы с вероятностью применения в конкретном случае работы устройства (формула (6)) [9]

$$MTTF = \sum_{j=1}^{j=M} \sum_{i=1}^{i=N_j} \Big\{ PU_{ij} \cdot \left(1 - PF_{ij}\right) \cdot \Big[TU_{ij} + \sum_{(m=1)\&\,(m \ne j)}^{m=M} \sum_{n=1}^{n=N_m} [(1 - PL_{nm}^{ij}) \cdot [TU_{mn} + \sum_{i \in D_{nm}} \Big((1 - PL_{nm}^{ij}) \cdot TU_{lm} + \sum_{n=1}^{n=N_m} [(1 - PL_{nm}^{ij}) \cdot [TU_{mn} + \sum_{i \in D_{nm}} [(1 - PL_{nm}^{ij}) \cdot TU_{lm}]] \sum_{l \in D_{nm}} \left(1 - PL_{nm}^{ij}\right) \cdot TU_{lm} \Big) \Big) \Big)] \} \quad (6)$$

где N_j – число компонентов на уровне j, $j \in \{1, \ldots, M\}$;

D_{ij} – множество индексов компонентов, зависящих от компонента i на уровне j, $i \in \{1, \ldots, N_j\}$;

PU_{ij} – вероятность использования компонента i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$;

PF_{ij} – вероятность сбоя в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$;

PL_{nm}^{ij} – условная вероятность сбоя в компоненте m на уровне n при сбое в компоненте i на уровне j, $i \in \{1, \ldots, N_j\}$, $j \in \{1, \ldots, M\}$, $n \in \{1, \ldots, N_m\}$, $m \in \{1, \ldots, M\}$;

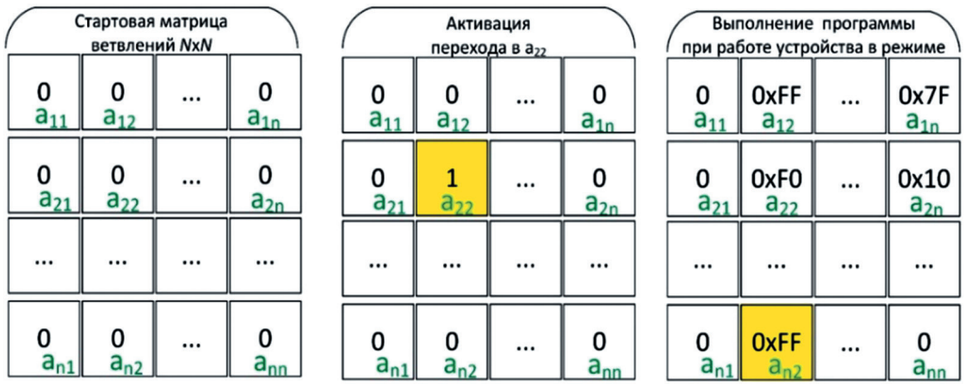

Рис. 7. Матрица ветвлений для алгоритма фаззинга

TA_{ij} – время доступа к компоненту i на уровне $j, i \in \{1, \ldots, N_j\}, j \in \{1, \ldots, M\}$;

TC_{ij} – время анализа сбоя в компоненте i на уровне $j, i \in \{1, \ldots, N_j\}, j \in \{1, \ldots, M\}$;

TE_{ij} – время устранения сбоя в компоненте i на уровне $j, i \in \{1, \ldots, N_j\}, j \in \{1, \ldots, M\}$;

TU_{ij} – время использования компонента i на уровне $j, i \in \{1, \ldots, N_j\}, j \in \{1, \ldots, M\}$.

В результате оценки показателя *MTTF* при разработке и тестировании производится корректировка программных и аппаратных модулей системы, после чего тестирование системы повторяется до появления необходимых параметров.

Момент окончания испытаний вычисляется с помощью вероятностного и статистического подхода [6]. Для нормального закона распределения вероятности наработки на отказ *MTTF* можно перейти к ее оценке по частоте отказов и сбоев, определив нижнюю и верхнюю границу такой вероятности [4, 8]. Затем производится ее динамическая оценка на основе установленного в ТЗ времени между отказами (*MTTF*) устройства по формуле (7):

$$p_{1,2} = \frac{p^* + \frac{1}{2}\frac{t_\beta^2}{n} \pm t_\beta \sqrt{\frac{p^*(1-p^*)}{n} + \frac{1}{4}\frac{t_\beta^2}{n^2}}}{1 + \frac{t_\beta^2}{n}}, \qquad (7)$$

где t_β – коэффициент для доверительной вероятности β, определяемый таблично, p^* – оценка вероятности по частоте, n – число измерений.

В результате экспериментальных исследований для 3-х режимов цифрового устройства с помощью имитатора неисправностей обнаружено на 82% больше неисправностей, чем при классическом тестировании (179 дефектов) [5].

Полученный объем испытаний, в результате которого установлена с заданной точностью необходимая вероятность наработки на отказ *MTTF*, считается достаточным для завершения испытаний по тестированию устройства с помощью фаззинга. Зная коэффициент надежности ПО *R* и доверительную оценку *MTTF*, можно локализовать зоны и режимы, которые в первую очередь требуется тестировать с помощью стенда имитации неисправностей.

Заключение

Поскольку используемые микропроцессорные элементы аппаратных средств относятся к классу высоконадежных изделий, то основной проблемой обеспечения работоспособности проектируемых устройств является выявление отказов, связанных с дефектами ПО, проявляемыми в реальной аппаратной среде. Разработана структура имитатора неисправностей с набором алгоритмов для ускорения поиска дефектов с помощью техники фаззинга. Определяется момент окончания испытаний с заданной точностью. За счет совмещения разработанных алгоритмов и техники фаззинга удалось выявить дефекты, являющиеся сложно обнаруживаемыми классическими методами тестирования для режимов работы цифрового устройства комплекса.

В дальнейшем планируется синтезировать алгоритмы фаззинга с алгоритмами мультиверсионного контроля для отладки качества работы инструментов фаззинга на известных наборах ошибок.

Библиографический список

1. Козачок А.В. Козачок В.И., Осипова Н.С. и др. Обзор исследований по применению методов машинного обучения для повышения эффективности фаззинг-тестирования // Вестник ВГУ, серия: системный анализ и информационные технологии. 2021. № 4. С. 83-106.

2. Mathur F.P., Avižienis A. Reliability analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redundancy with self-repair // Proceedings of the May 5-7, 1970, spring joint computer conference. 1970. Pp. 375-383.

3. Ковалев И.В. Анализ проблем в области исследования надежности программного обеспечения: многоэтапность и архитектурный аспект // Вестник СибГАУ. 2014. № 3. С. 78–92.

4. Патент № 2697629 Российская Федерация, МПК G06F 11/261. Устройство для имитации неисправностей в программно-аппаратных системах: № 2018105476 заявл. 13.02.18; опубл.15.08.19 / Д.А. Панков; патентообладатель АО ОНИИП.

5. Панков Д.А., Панков И.А., Денисова Л.А. Автоматизация разработки и тестирования цифровых систем связи с многоуровневой архитектурой // Автоматизация в промышленности. 2023. № 1. С. 31–35.

6. Панков Д.А., Денисова Л.А. Проектирование программно-аппаратного комплекса: определение объема тестовых испытаний микропроцессорных устройств // Автоматизация в промышленности. 2020. № 12. С. 23-29. DOI: 10.25728/avtprom.2020.12.04

7. Rebert A., Cha S.K., Avgerinos T. et al. Optimizing seed selection for fuzzing // IEEE Access. 2018. Vol. 6. Pp. 861-875. URL: https://www.usenix.org/conference/usenixsecurity14/technicalsessions/presentation/rebert (дата обращения 08.05.2023)

8. Ушаков И.А. Надежность: прошлое, настоящее, будущее // Reliability: Theory & Applications. 2006. № 1. С.17– 25.

9. Русаков М.А. Многоэтапный анализ архитектурной надежности в сложных информационно-управляющих системах : дис. … канд. техн. наук. Красноярск, 2005. 168 с.

References

1. Kozachok A.V., Kozachok V.I., Osipova N.S. et al. Overview of studies on the application of machine learning methods to improve the efficiency of fuzzing testing. *Proceedings of Voronezh State University. Series: Systems Analysis and Information Technologies* 2021;4:83-106. (in Russ.)

2. Mathur F.P., Avižienis A. Reliability analysis and architecture of a hybrid-redundant digital system: Generalized triple modular redundancy with self-repair. Proceedings of the May 5-7, 1970, spring joint computer conference; 1970. P. 375-383.

3. Kovalev I.V. Analysis of problems in the research area of software reliability: a lot of stages and architectural aspect. *The Siberian Aerospace Journal* 2014;3:78-92. (in Russ.)

4. Pankov D.A. Patent no. 2697629 Russian Federation, MPK G06F 11/261. [Device for imitating malfunctions in hardware and software systems]: no. 2018105476 appl. 13.02.18; publ.15.08.19. Patent holder AO ONIIP. (in Russ.)

5. Pankov D.A., Pankov I.A., Denisova L.A. [Automation of the development and testing of digital telecommunication systems with multi-layer architecture]. *Avtomatizatsiya v promyshlennosti* 2023;1:31-35. (in Russ.)

6. Pankov D.A., Denisova L.A. [Designing a hardware and software system: defining the scope of testing of computer-based devices]. *Avtomatrizatsiya v promyshlennosti* 2020;12:23-29. DOI: 10.25728/avtprom.2020.12.04. (in Russ.)

7. Rebert A., Cha S.K., Avgerinos T. et al. Optimizing seed selection for fuzzing. *IEEE Access* 2018;6:861-875. (accessed 08.05.2023). Available at: https://www.usenix.org/conference/usenixsecurity14/technicalsessions/presentation/rebert.

8. Ushakov I.A. [Reliability: past, present, future]. *Reliability: Theory & Applications* 2006;1:17-25.

9. Rusakov M.A. [Multistage analysis of the architectural reliability in complex information management systems: a Candidate of Engineering thesis]. Krasnoyarsk; 2005. (in Russ.)

Сведения об авторах

Панков Денис Анатольевич – кандидат технических наук, заместитель начальника отдела по научно-технической работе АО «ОНИИП», г. Омск, Российская Федерация e-mail: pankovddd@yandex.ru

Панков Илья Анатольевич – аспирант ОмГТУ кафедры АСОИУ, г. Омск, Российская Федерация, e-mail: pankov99ai@yandex.ru

About the authors

Denis A. Pankov, Candidate of Engineering, Deputy Head of Research and Engineering Department, AO ONIIP, Omsk, Russian Federation, e-mail: pankovddd@yandex.ru.

Ilia A. Pankov, postgraduate student, OmSTU, Department of Automated Information Processing and Control Systems, Omsk, Russian Federation, e-mail: pankov99ai@yandex.ru.

Вклад авторов в статью

Авторами предложены подходы к организации испытаний цифровых систем с помощью имитации неисправностей и анализ объема испытаний с помощью нечеткого логического вывода и методов теории вероятности для выявления системных ошибок, проявляющих в ПО. Показано направление внедрения фаззинга для оценки качество ПО.

Панков Д.А. предложил направление внедрения техники фаззинга и структуру имитации неисправностей для оценки качества ПО проектируемых систем, разработал алгоритм определения объема испытаний.

Панков И.А. разработал алгоритмы тестирования, а также провел экспериментальные исследования, выявил ограничения и затруднения, которые необходимо будет устранить для использования техник имитации неисправностей на уровне оборудования и программных средств разработки.

Конфликт интересов

Авторы заявляют об отсутствии конфликта интересов.